СТАНДАРТ ТРЕТЬЕГО ПОКОЛЕНИЯ

Ф. А. Новиков

Дискретная математика

ДЛЯ БАКАЛАВРОВ И МАГИСТРОВ

3-е издание

Рекомендовано Учебно-методическим объединением
по университетскому политехническому образованию в качестве
учебника для студентов высших учебных заведений, обучающихся
по направлению подготовки «Системный анализ и управление»

Санкт-Петербург · Москва · Екатеринбург · Воронеж
Нижний Новгород · Ростов-на-Дону
Самара · Минск

2017

ББК 22.174я7
УДК 519.1(075)
Н75

Рецензенты: Абрамов С. М., директор Института программных систем им. А. К. Айламазяна РАН, чл.-корр. РАН, д.ф.-м.н.;
ГОУВПО «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»

Новиков Ф.
Н75 Дискретная математика: Учебник для вузов. 3-е изд. Стандарт третьего поколения. — СПб.: Питер, 2017. — 496 с.: ил. — (Серия «Учебник для вузов»).

ISBN 978-5-496-02044-2

Новое издание учебника было существенно переработано и дополнено, в нем изложены все основные разделы дискретной математики и описаны важнейшие алгоритмы на дискретных структурах данных. Основу книги составляет материал лекционного курса, который автор читает в Санкт-Петербургском политехническом университете Петра Великого.

Книга имеет обширный справочный аппарат: указатель обозначений, детальный предметный указатель с переводом всех терминов на английский язык, развернутый библиографический список. Содержание учебника полностью соответствует Федеральному государственному образовательному стандарту высшего профессионального образования. Для студентов вузов, обучающихся по направлениям подготовки «Системный анализ и управление», «Прикладная математика и информатика», «Информатика и вычислительная техника», а также для всех желающих изучить дискретную математику.

Рекомендовано Учебно-методическим объединением по университетскому политехническому образованию в качестве учебника для студентов высших учебных заведений, обучающихся по направлению подготовки «Системный анализ и управление».

ББК 22.174я7
УДК 519.1(075)

ISBN 978-5-496-02044-2

Краткое оглавление

Предисловие 12
Введение 14
Глава 1. Множества и отношения 21
Глава 2. Алгебраические структуры 104
Глава 3. Булевы функции 155
Глава 4. Логические исчисления 198
Глава 5. Комбинаторный анализ 252
Глава 6. Кодирование 298
Глава 7. Графы 333
Глава 8. Связность 366
Глава 9. Деревья 399
Глава 10. Циклы, независимость и раскраска 445
Указатель основных обозначений 476
Список литературы 479
Предметный указатель 480

Оглавление

Предисловие 12

Введение 14

Глава 1. Множества и отношения 21
1.1. Множества 21
1.1.1. Элементы и множества 21
1.1.2. Задание множеств 23
1.1.3. Парадокс Рассела 24
1.1.4. Мультимножества 25
1.1.5. Конечные последовательности 25
1.2. Алгебра подмножеств 27
1.2.1. Сравнение множеств 27
1.2.2. Равномощные множества 28
1.2.3. Конечные и бесконечные множества 29
1.2.4. Счётные и несчётные множества 29
1.2.5. Мощность конечного множества 31
1.2.6. Операции над множествами 33
1.2.7. Разбиения и покрытия 34
1.2.8. Булеан 35
1.2.9. Свойства операций над множествами 36
1.3. Представление множеств в программах 37
1.3.1. Битовые шкалы 37
1.3.2. Представление натуральных чисел 39
1.3.3. Перебор подмножеств заданного множества 42
1.3.4. Алгоритм построения бинарного кода Грея 44
1.3.5. Представление множеств упорядоченными списками 47
1.3.6. Проверка включения слиянием 47
1.3.7. Вычисление объединения слиянием 48
1.3.8. Вычисление пересечения слиянием 49
1.3.9. Представление множеств итераторами 49
1.4. Отношения 52
1.4.1. Упорядоченные пары и наборы 52
1.4.2. Прямое произведение множеств 53
1.4.3. Размеченное объединение множеств 54
1.4.4. Бинарные отношения 55
1.4.5. Композиция отношений 57
1.4.6. Свойства отношений 58
1.4.7. Степень отношения и циклы 60
1.4.8. Ядро отношения 61
1.4.9. Представление отношений в программах 62
1.5. Замыкание и сокращение отношений 63
1.5.1. Транзитивное и рефлексивное замыкание 64
1.5.2. Алгоритм Уоршалла вычисления транзитивного замыкания отношения 64
1.5.3. Транзитивное сокращение 65
1.5.4. Диаграммы Хассе 67
1.6. Функции 69
1.6.1. Функциональные отношения 70

1.6.2. Инъекция, сюръекция и биекция . . . 70
1.6.3. Образы и прообразы . . . 71
1.6.4. Суперпозиция функций . . . 72
1.6.5. Степень функции . . . 73
1.6.6. Представление функций в программах . . . 74
1.7. Отношения эквивалентности . . . 75
1.7.1. Классы эквивалентности . . . 75
1.7.2. Ядро функционального отношения . . . 77
1.7.3. Фактормножество . . . 78
1.7.4. Множества уровня . . . 79
1.7.5. Толерантность . . . 79
1.7.6. Гомоморфизмы и изоморфизмы . . . 80
1.8. Отношения порядка . . . 82
1.8.1. Предпорядок . . . 82
1.8.2. Частичный и линейный порядок . . . 83
1.8.3. Минимальные и наименьшие элементы . . . 86
1.8.4. Алгоритм топологической сортировки . . . 87
1.8.5. Верхние и нижние границы . . . 88
1.8.6. Монотонные и непрерывные функции . . . 89
1.8.7. Наименьшая неподвижная точка функции . . . 90
1.8.8. Вполне упорядоченные множества . . . 92
1.8.9. Индукция . . . 93
1.9. Характеристические функции . . . 94
1.9.1. Классификация характеристических функций . . . 94
1.9.2. Характеристические функции множеств . . . 97
1.9.3. Характеристические функции отношений . . . 99
1.9.4. Характеристические функции мультимножеств . . . 100
1.9.5. Классификаторы . . . 100
1.9.6. Нечёткие множества . . . 102

Глава 2. Алгебраические структуры . . . 104
2.1. Алгебры и морфизмы . . . 104
2.1.1. Операции и их носители . . . 104
2.1.2. Замыкания и подалгебры . . . 105
2.1.3. Система образующих . . . 106
2.1.4. Свойства операций . . . 107
2.1.5. Гомоморфизмы и изоморфизмы алгебр . . . 108
2.2. Алгебры с одной операцией . . . 110
2.2.1. Полугруппы . . . 110
2.2.2. Определяющие соотношения . . . 111
2.2.3. Системы подстановок термов . . . 112
2.2.4. Алгоритм Кнута–Бендикса . . . 114
2.2.5. Моноиды . . . 116
2.2.6. Группы . . . 117
2.2.7. Действие группы на множестве . . . 119
2.2.8. Группа перестановок . . . 119
2.2.9. Перестановочные матрицы . . . 121
2.3. Алгебры с двумя операциями . . . 122
2.3.1. Кольца . . . 122
2.3.2. Области целостности . . . 123
2.3.3. Поля . . . 124
2.4. Элементарная теория чисел . . . 125
2.4.1. Делимость чисел . . . 126
2.4.2. Наибольший общий делитель . . . 127
2.4.3. Наименьшее общее кратное . . . 129
2.4.4. Простые числа . . . 130
2.4.5. Сравнения . . . 133
2.4.6. Системы вычетов . . . 134
2.4.7. Китайская теорема об остатках . . . 135
2.4.8. Вычисления в остаточных классах . . . 136
2.4.9. Функция Эйлера . . . 136
2.5. Векторные пространства и модули . . . 138

2.5.1. Векторные пространства . . . 138
2.5.2. Линейные комбинации . . . 139
2.5.3. Базис и размерность . . . 140
2.5.4. Конечные поля . . . 141
2.5.5. Модули . . . 143
2.6. Решётки . . . 144
2.6.1. Дистрибутивные и ограниченные решётки . . . 144
2.6.2. Решётки с дополнением . . . 145
2.6.3. Частичный порядок в решётке . . . 146
2.6.4. Полные решётки . . . 147
2.6.5. Полурешётки . . . 147
2.6.6. Булевы алгебры . . . 149
2.7. Матроиды и жадные алгоритмы . . . 149
2.7.1. Матроиды . . . 150
2.7.2. Максимальные независимые подмножества . . . 150
2.7.3. Базисы . . . 151
2.7.4. Жадный алгоритм . . . 152
2.7.5. Примеры матроидов . . . 153

Глава 3. Булевы функции . . . 155
3.1. Элементарные булевы функции . . . 155
3.1.1. Функции алгебры логики . . . 155
3.1.2. Существенные и несущественные переменные . . . 156
3.1.3. Булевы функции одной переменной . . . 157
3.1.4. Булевы функции двух переменных . . . 157
3.2. Функции k-значной логики . . . 158
3.2.1. Формулы . . . 159
3.2.2. Реализация функций формулами . . . 159
3.2.3. Равносильные формулы . . . 162
3.2.4. Подстановка и замена . . . 163
3.2.5. Алгебра булевых функций . . . 164
3.3. Двойственность и симметрия . . . 165
3.3.1. Двойственные и самодвойственные функции . . . 165
3.3.2. Реализация двойственной функции . . . 167
3.3.3. Принцип двойственности . . . 167
3.3.4. Симметрические функции . . . 168
3.4. Нормальные формы . . . 168
3.4.1. Разложение булевых функций по переменным . . . 168
3.4.2. Минимальные термы . . . 169
3.4.3. Совершенные нормальные формы . . . 170
3.4.4. Эквивалентные преобразования . . . 172
3.4.5. Минимальные дизъюнктивные формы . . . 174
3.4.6. Геометрическая интерпретация . . . 175
3.4.7. Сокращённые дизъюнктивные формы . . . 176
3.5. Представление булевых функций в программах . . . 177
3.5.1. Табличные представления . . . 177
3.5.2. Строковые представления . . . 178
3.5.3. Алгоритм вычисления значения булевой функции . . . 179
3.5.4. Представление булевых функций арифметическими полиномами . . . 180
3.5.5. Карты Карно . . . 183
3.5.6. Представление булевой функции на карте Карно . . . 185
3.5.7. Минимизация формул картами Карно . . . 186
3.5.8. Деревья решений . . . 189
3.6. Полные системы булевых функций . . . 192
3.6.1. Замкнутые классы . . . 192
3.6.2. Полные системы функций . . . 194
3.6.3. Полнота двойственной системы . . . 195
3.6.4. Теорема Поста . . . 195

Глава 4. Логические исчисления . . . 198
4.1. Логические связки и кванторы . . . 199
4.1.1. Высказывания и связки . . . 199

4.1.2. Формулы . . . 199
4.1.3. Интерпретация . . . 200
4.1.4. Логическое следование и логическая эквивалентность . . . 201
4.1.5. Подстановка и замена . . . 202
4.1.6. Кванторы . . . 202
4.2. Формальные теории . . . 206
4.2.1. Определение формальной теории . . . 206
4.2.2. Выводимость . . . 207
4.2.3. Интерпретация . . . 207
4.2.4. Общезначимость и непротиворечивость . . . 208
4.2.5. Полнота, независимость и разрешимость . . . 208
4.3. Исчисление высказываний . . . 209
4.3.1. Классическое определение исчисления высказываний . . . 209
4.3.2. Частный случай формулы . . . 209
4.3.3. Алгоритмы унификации . . . 210
4.3.4. Конструктивное определение исчисления высказываний . . . 212
4.3.5. Производные правила вывода . . . 213
4.3.6. Дедукция . . . 214
4.3.7. Некоторые теоремы исчисления высказываний . . . 215
4.3.8. Множество теорем исчисления высказываний . . . 217
4.3.9. Другие аксиоматизации исчисления высказываний . . . 218
4.4. Исчисление предикатов . . . 220
4.4.1. Определение исчисления предикатов . . . 220
4.4.2. Интерпретация . . . 222
4.4.3. Общезначимость . . . 223
4.4.4. Непротиворечивость и полнота чистого исчисления предикатов . . . 223
4.4.5. Логическое следование и логическая эквивалентность . . . 224
4.4.6. Теория равенства . . . 225
4.4.7. Формальная арифметика . . . 225
4.4.8. Неаксиоматизируемые теории . . . 226
4.4.9. Теоремы Гёделя о неполноте . . . 227
4.5. Наивная теория алгоритмов . . . 228
4.5.1. Алгоритмы . . . 228
4.5.2. Вычислимые функции . . . 230
4.5.3. Перечислимые множества . . . 231
4.5.4. Разрешимые множества . . . 234
4.5.5. Протокол выполнения алгоритма . . . 235
4.5.6. Программы . . . 236
4.5.7. Невычислимые функции и неразрешимые множества . . . 238
4.5.8. Истинность и доказуемость . . . 240
4.5.9. Множество истин арифметики . . . 242
4.6. Автоматическое доказательство теорем . . . 244
4.6.1. Постановка задачи автоматического доказательства теорем . . . 244
4.6.2. Доказательство от противного . . . 245
4.6.3. Сведе́ние к дизъюнктам . . . 245
4.6.4. Правило резолюции для исчисления высказываний . . . 246
4.6.5. Правило резолюции для исчисления предикатов . . . 247
4.6.6. Опровержение методом резолюций . . . 248
4.6.7. Дерево вывода . . . 249
4.6.8. Алгоритм метода резолюций . . . 251

Глава 5. Комбинаторный анализ . . . 252
5.1. Комбинаторные задачи . . . 253
5.1.1. Комбинаторные конфигурации . . . 253
5.1.2. Размещения . . . 254
5.1.3. Размещения без повторений . . . 254
5.1.4. Перестановки . . . 255
5.1.5. Сочетания . . . 256
5.1.6. Сочетания с повторениями . . . 257
5.1.7. Дискретная вероятность . . . 258
5.2. Перестановки . . . 259
5.2.1. Циклы в перестановках . . . 260

5.2.2. Порядок перестановки . . . 262
5.2.3. Инверсии . . . 263
5.2.4. Генерация перестановок . . . 263
5.2.5. Двойные факториалы . . . 265
5.2.6. Быстрая сортировка . . . 266
5.2.7. Трудоёмкость в среднем . . . 268
5.2.8. Гармонические числа . . . 269
5.2.9. Оценка в среднем для быстрой сортировки . . . 269
5.3. Биномиальные коэффициенты . . . 270
5.3.1. Элементарные тождества . . . 271
5.3.2. Бином Ньютона . . . 271
5.3.3. Свойства биномиальных коэффициентов . . . 272
5.3.4. Треугольник Паскаля . . . 273
5.3.5. Генерация подмножеств . . . 275
5.3.6. Расширенные биномиальные коэффициенты . . . 277
5.3.7. Мультимножества и последовательности . . . 278
5.3.8. Мультиномиальные коэффициенты . . . 279
5.3.9. Биномиальная система счисления . . . 280
5.4. Разбиения . . . 281
5.4.1. Числа Стирлинга второго рода . . . 281
5.4.2. Числа Стирлинга первого рода . . . 282
5.4.3. Вычисление чисел Стирлинга . . . 283
5.4.4. Число Белла . . . 284
5.4.5. Треугольник Белла . . . 284
5.5. Включения и исключения . . . 286
5.5.1. Объединение конфигураций . . . 286
5.5.2. Формула включений и исключений . . . 287
5.5.3. Число булевых функций, существенно зависящих от всех своих переменных . . . 288
5.5.4. Комбинации свойств . . . 289
5.5.5. Беспорядки и субфакториал . . . 289
5.6. Формулы обращения . . . 290
5.6.1. Теорема обращения . . . 291
5.6.2. Формулы обращения для биномиальных коэффициентов . . . 291
5.6.3. Формулы для чисел Стирлинга . . . 292
5.7. Производящие функции . . . 292
5.7.1. Формальные степенные ряды . . . 293
5.7.2. Метод неопределённых коэффициентов . . . 293
5.7.3. Числа Фибоначчи . . . 294
5.7.4. Числа Каталана . . . 295

Глава 6. Кодирование . . . 298
6.1. Алфавитное кодирование . . . 299
6.1.1. Таблица кодов . . . 299
6.1.2. Разделимые схемы . . . 300
6.1.3. Префиксные схемы . . . 300
6.1.4. Неравенство Крафта–Макмиллана . . . 301
6.2. Кодирование с минимальной избыточностью . . . 303
6.2.1. Минимизация длины кода сообщения . . . 303
6.2.2. Цена кодирования . . . 304
6.2.3. Алгоритм Фано . . . 305
6.2.4. Оптимальное кодирование . . . 306
6.2.5. Алгоритм Хаффмена . . . 308
6.3. Помехоустойчивое кодирование . . . 310
6.3.1. Кодирование с исправлением ошибок . . . 310
6.3.2. Возможность исправления всех ошибок . . . 312
6.3.3. Расстояния Левенштейна и Хэмминга . . . 314
6.3.4. Кодовое расстояние . . . 315
6.3.5. Мощность кодирования . . . 316
6.3.6. Виды кодирования . . . 318
6.3.7. Систематическая форма кодовых сообщений . . . 320
6.3.8. Коды Хэмминга . . . 321
6.3.9. Исправление одного замещения . . . 323

6.4. Сжатие и шифрование данных . . . 324
6.4.1. Сжатие текстов . . . 324
6.4.2. Предварительное построение словаря . . . 324
6.4.3. Алгоритм Лемпела–Зива . . . 325
6.4.4. Криптография . . . 328
6.4.5. Шифрование с помощью случайных чисел . . . 329
6.4.6. Криптостойкость . . . 329
6.4.7. Шифрование с открытым ключом . . . 330
6.4.8. Цифровая подпись . . . 332

Глава 7. Графы . . . 333
7.1. Определения графов . . . 333
7.1.1. История теории графов . . . 333
7.1.2. Основное определение . . . 335
7.1.3. Инцидентность, смежность и дополнение . . . 336
7.1.4. Диаграмма графа . . . 336
7.1.5. Орграфы, псевдографы, мультиграфы и гиперграфы . . . 337
7.1.6. Изоморфизм графов . . . 338
7.2. Элементы графов . . . 339
7.2.1. Подграфы . . . 339
7.2.2. Валентность . . . 340
7.2.3. Маршруты, цепи, циклы . . . 341
7.2.4. Связные и несвязные графы . . . 342
7.2.5. Расстояние между вершинами, ярусы и диаметр графа . . . 342
7.2.6. Эксцентриситет и центр . . . 343
7.2.7. Степенные последовательности . . . 343
7.3. Виды графов и операции над графами . . . 346
7.3.1. Полные и пустые графы . . . 347
7.3.2. Двудольные графы . . . 347
7.3.3. Направленные орграфы и сети . . . 348
7.3.4. Операции над графами . . . 349
7.3.5. Простые циклы . . . 351
7.3.6. Рёберные графы . . . 352
7.4. Представление графов в программах . . . 354
7.4.1. Требования к представлению графов . . . 354
7.4.2. Матрица смежности . . . 354
7.4.3. Матрица инциденций . . . 355
7.4.4. Списки смежности . . . 355
7.4.5. Массив дуг . . . 356
7.4.6. Обходы графов . . . 356
7.5. Графы и отношения . . . 360
7.5.1. Орграфы и бинарные отношения . . . 360
7.5.2. Граф инциденций . . . 362
7.5.3. Гиперграфы и многоместные отношения . . . 363
7.5.4. Достижимость и частичное упорядочение . . . 364
7.5.5. Достижимость и транзитивное замыкание . . . 365

Глава 8. Связность . . . 366
8.1. Компоненты связности . . . 366
8.1.1. Объединение графов и компоненты связности . . . 366
8.1.2. Точки сочленения, мосты и блоки . . . 367
8.1.3. Вершинная и рёберная связность . . . 371
8.1.4. Оценка числа рёбер . . . 372
8.2. Теорема Менгера . . . 373
8.2.1. Непересекающиеся цепи и разделяющие множества . . . 373
8.2.2. Теорема Менгера в «вершинной форме» . . . 374
8.2.3. Варианты теоремы Менгера . . . 375
8.3. Паросочетания . . . 375
8.3.1. Задачи о свадьбах, трансверсалях и паросочетаниях . . . 376
8.3.2. Теорема Холла . . . 376
8.3.3. Устойчивые паросочетания . . . 377
8.4. Потоки в сетях . . . 380

8.4.1. Определение потока . . . 380
8.4.2. Разрезы . . . 381
8.4.3. Теорема Форда–Фалкерсона . . . 381
8.4.4. Алгоритм нахождения максимального потока . . . 383
8.4.5. Связь между теоремой Менгера и теоремой Форда–Фалкерсона . . . 384
8.5. Связность в орграфах . . . 385
8.5.1. Сильная, односторонняя и слабая связность . . . 385
8.5.2. Компоненты сильной связности . . . 386
8.5.3. Выделение компонент сильной связности . . . 386
8.6. Кратчайшие пути . . . 388
8.6.1. Взвешенные графы . . . 388
8.6.2. Кратчайшие пути в бесконтурном орграфе . . . 390
8.6.3. Алгоритм Беллмана–Форда . . . 391
8.6.4. Алгоритм Дейкстры . . . 392
8.6.5. Алгоритм Флойда–Уоршалла . . . 396
8.6.6. Алгоритмы с предобработкой . . . 397

Глава 9. Деревья . . . 399
9.1. Свободные деревья . . . 399
9.1.1. Свободные деревья и их элементы . . . 399
9.1.2. Основные свойства деревьев . . . 400
9.1.3. Центр дерева . . . 403
9.2. Ориентированные, упорядоченные и бинарные деревья . . . 403
9.2.1. Ориентированные деревья . . . 403
9.2.2. Эквивалентное определение ордерева . . . 405
9.2.3. Упорядоченные деревья . . . 405
9.2.4. Бинарные деревья . . . 408
9.3. Представление деревьев в программах . . . 408
9.3.1. Представление свободных деревьев . . . 408
9.3.2. Представление упорядоченных корневых деревьев . . . 411
9.3.3. Число упорядоченных ориентированных деревьев . . . 412
9.3.4. Проверка правильности скобочной структуры . . . 414
9.3.5. Представление бинарных деревьев . . . 414
9.3.6. Обходы бинарных деревьев . . . 416
9.3.7. Алгоритмы симметричного обхода бинарного дерева . . . 417
9.4. Деревья сортировки . . . 418
9.4.1. Ассоциативная память . . . 418
9.4.2. Способы реализации ассоциативной памяти . . . 419
9.4.3. Алгоритм бинарного (двоичного) поиска . . . 420
9.4.4. Алгоритм поиска в дереве сортировки . . . 421
9.4.5. Алгоритм вставки в дерево сортировки . . . 422
9.4.6. Алгоритм удаления из дерева сортировки . . . 423
9.4.7. Вспомогательные алгоритмы для дерева сортировки . . . 425
9.4.8. Сравнение представлений ассоциативной памяти . . . 426
9.5. Специальные деревья . . . 426
9.5.1. Выровненные и полные деревья . . . 426
9.5.2. Сбалансированные деревья . . . 428
9.5.3. Балансировка деревьев . . . 429
9.5.4. Двоичные кучи . . . 430
9.5.5. Восстановление свойства двоичной кучи . . . 432
9.5.6. Реализация операций двоичной кучи . . . 433
9.5.7. Построение графа по степенному ряду . . . 434
9.5.8. Дерево отрезков . . . 435
9.5.9. Оценки эффективности дерева отрезков . . . 438
9.6. Кратчайший остов . . . 440
9.6.1. Стягивающие деревья и остовы . . . 440
9.6.2. Схема алгоритма построения кратчайшего остова . . . 441
9.6.3. Алгоритм Краскала . . . 442
9.6.4. Алгоритм Прима . . . 443

Глава 10. Циклы, независимость и раскраска . . . 445
10.1. Фундаментальные циклы и разрезы . . . 445

10.1.1. Циклы и разрезы . . . 445
10.1.2. Фундаментальная система циклов и циклический ранг . . . 447
10.1.3. Фундаментальная система разрезов и коциклический ранг . . . 448
10.1.4. Подпространства циклов и коциклов . . . 450
10.2. Эйлеровы циклы . . . 451
10.2.1. Эйлеровы графы . . . 451
10.2.2. Алгоритм построения эйлерова цикла в эйлеровом графе . . . 452
10.2.3. Оценка числа эйлеровых графов . . . 453
10.3. Гамильтоновы циклы . . . 453
10.3.1. Гамильтоновы графы . . . 454
10.3.2. Задача коммивояжёра . . . 455
10.3.3. Теорема Поша . . . 456
10.4. Независимые и покрывающие множества . . . 457
10.4.1. Независимые и покрывающие множества вершин и рёбер . . . 457
10.4.2. Связь чисел независимости и покрытий . . . 459
10.4.3. Оценка числа вершинной независимости . . . 460
10.5. Построение независимых множеств вершин . . . 461
10.5.1. Постановка задачи отыскания наибольшего независимого множества вершин . . . 461
10.5.2. Поиск с возвратами . . . 462
10.5.3. Улучшенный перебор . . . 464
10.5.4. Алгоритм построения независимых множеств . . . 465
10.6. Доминирующие множества . . . 466
10.6.1. Минимальное и наименьшее доминирующее множество . . . 466
10.6.2. Доминирование и независимость . . . 466
10.6.3. Задача о наименьшем покрытии . . . 467
10.6.4. Связь задачи о наименьшем покрытии с другими задачами . . . 467
10.7. Раскраска графов . . . 468
10.7.1. Оценки хроматического числа . . . 468
10.7.2. Хроматические числа графа и его дополнения . . . 469
10.7.3. Точный алгоритм раскрашивания . . . 470
10.7.4. Приближённые алгоритмы раскрашивания . . . 471
10.8. Планарность . . . 472
10.8.1. Укладка графов . . . 473
10.8.2. Эйлерова характеристика . . . 473
10.8.3. Теорема о пяти красках . . . 475

Указатель основных обозначений . . . 476

Список литературы . . . 479

Предметный указатель . . . 480

Предисловие

Предлагаемое издание учебника «Дискретная математика для программистов» является результатом достаточно длительного развития.

В 2000 году издательство «Питер» выпустило в свет учебное пособие Ф. А. Новикова «Дискретная математика для программистов». Книга оказалась удачной, о чём свидетельствовали тиражи и письма читателей. Поэтому в 2004 году тем же издательством было выпущено второе, а в 2008 году — третье издание, которые также пользовались широким спросом. Резонно предположить, что объём, стиль и содержание книги отвечают возрастающим потребностям студентов, изучающих дискретную математику, и инженеров, применяющих математические модели в своей работе. В то же время за десять лет изменились требования в системе высшего профессионального образования, поэтому в 2010 году было выпущено новое издание, учебник «Дискретная математика», созданный на основе учебного пособия «Дискретная математика для программистов», но существенно исправленный и дополненный. Содержание учебника было приведено в соответствие с новым федеральным государственным образовательным стандартом высшего профессионального образования, введённым в действие в 2009 году. Учебник получил рекомендацию Учебно-методического объединения по университетскому политехническому образованию. В 2012 году вышло в свет второе издание учебника «Дискретная математика», исправленное и дополненное по сравнению с первым изданием. Все эти книги имеют единый план и стиль, но каждый следующий вариант более точен в деталях и подробностях, а потому имеет несколько больший объём, консервативно расширяя и дополняя предыдущие издания.

Опираясь на тридцатилетний опыт преподавания дискретной математики в Санкт-Петербургском политехническом университете Петра Великого и пятнадцатилетний опыт разработки и использования учебника, в данном издании предпринята попытка синтезировать основу, поддерживающую более широкую область применения, нежели только чтение бумажного учебника. Предлагаемый учебник, наряду со своим основным предназначением, служит текстуальной основой для создания электронных учебных материалов: слайдов и электронных книг.

Это обстоятельство объясняет причины, по которым были сохранены некоторые особенности предыдущих книг и добавлены новые отличительные черты.

Сохранена предельная лаконичность в изложении материала: по мнению автора, каждая страница, без которой можно обойтись, является заметным недостатком. Электронные книги не должны быть пространными.

Добавлено значительное количество иллюстраций, которые в электронном варианте могут быть обогащены цветом и анимацией.

Сохранена мелкая рубрикация: параграф помещается на нескольких страницах и может быть прочтен за один присест, не поднимая головы (или не отрывая взгляда от экрана).

Добавлены многочисленные перекрёстные ссылки и отступления, а также существенно обогащен справочный аппарат: предметный указатель и указатель обозначений. В электронном варианте это даёт возможность организации гипертекстовых связей.

Сохранена ориентация на практическое применение: описание объектов дискретной математики в большинстве случаев доводится до кода алгоритмов работы с этими объектами.

Таким образом, предлагаемое издание продолжает и развивает линию предыдущих изданий и в то же время привносит новое качество, отвечающее, по мнению автора, потребностям современного образования.

Ф. А. Новиков

8 июня 2016 года

Введение

В этой книге рассматриваются некоторые элементарные понятия «дискретной», или «конечной», математики, то есть математики, прежде всего изучающей конечные множества и различные структуры на них. Это означает, что понятия бесконечности, предела или непрерывности не являются предметом изучения, хотя могут использоваться как вспомогательные средства. Дискретная математика имеет широкий спектр приложений, прежде всего в областях, связанных с информационными технологиями и компьютерами. В самом первоначальном, ныне редко используемом названии компьютера — «электронная цифровая вычислительная машина» — слово «цифровая» указывает на принципиально дискретный характер работы данного устройства. Современные компьютеры неотделимы от современных программистов, для которых и написана эта книга.

Назначение и особенности книги

Данная книга представляет собой учебник по дискретной математике, включающий в себя описания важнейших алгоритмов над объектами дискретной математики. Учебник ориентирован на студентов программистских специальностей и практикующих программистов, которым по роду их занятий приходится иметь дело с конструированием и анализом алгоритмов.

В настоящее время имеется масса доступной литературы, покрывающей обе эти темы. С одной стороны, существуют десятки прекрасных книг по дискретной математике, начиная с элементарных учебников для начинающих и заканчивая исчерпывающими справочниками для специалистов. С другой стороны, большое количество монографий, многие из которых стали классическими, посвящено способам конструирования эффективных алгоритмов. Как правило, такие монографии содержат детальное описание и анализ важнейших и известнейших алгоритмов. Однако книги, которые рассматривали бы обе эти темы одновременно и во взаимосвязи, практически отсутствуют. В качестве редких исключений можно назвать книгу В. Липского «Комбинаторика для программистов» [14], перевод которой выдержал уже два издания в России, и всеобъемлющую энциклопедию Д. Кнута «Искусство программирования для ЭВМ» [10–12]. Первая из упомянутых книг очень невелика по объёму и в значительной степени пересекается по выбранному материалу с данным учебником. Вторая отличается максимальной глубиной изложения и большим объёмом, но не затрагивает некоторых важных для программистов разделов дискретной математики. Данный учебник принадлежит именно к такому жанру математической литературы, в котором математическое изложение доводится до уровня практически исполнимых программ. Он отличается большей *широтой*, но, пожалуй, меньшей *глубиной* охвата материала.

Учебник основан на лекционном курсе, который автор в течение многих лет читает студентам кафедры «Прикладная математика» Санкт-Петербургского политехнического университета Петра Великого, что наложило определённый отпечаток на выбор материала. Учебник охватывает почти все основные разделы дискретной математики: теорию множеств, математическую логику, общую алгебру, комбинаторику и теорию графов. Кроме того, представлены и некоторые более специальные разделы, необходимые программистам, такие как теория кодирования и булевы функции. Данный список можно было бы продолжить, включив в него, например,

вычислительную геометрию, математическое программирование и т. д., но основная цель состояла в том, чтобы написать *базовый* учебник, охватывающий весь аппарат, используемый в последующих специальных курсах, но не выходя за рамки отведённого объёма учебной нагрузки на студентов.

В целом учебник преследует три основные цели:

1. Познакомить читателя с максимально широким кругом понятий дискретной математики. Количество определяемых и упоминаемых понятий и специальных терминов намного превышает количество понятий, обсуждаемых более детально. Тем самым у студента формируется терминологический запас, необходимый для самостоятельного изучения специальной математической и теоретико-программистской литературы.
2. Сообщить читателю необходимые конкретные сведения из дискретной математики, предусматриваемые федеральным государственным образовательным стандартом. Разбор доказательств приведенных утверждений и выполнение упражнений позволят студенту овладеть методами дискретной математики, наиболее употребительными при решении практических задач.
3. Пополнить запас примеров нетривиальных алгоритмов. Изучение алгоритмов решения типовых задач дискретной математики и способов представления математических объектов в программах абсолютно необходимо практикующему программисту, поскольку позволяет уменьшить трудозатраты на «изобретение велосипеда» и существенно обогащает навыки конструирования алгоритмов.

Структура книги

Книга содержит 10 глав, которые можно разделить на три группы. Первая группа, несколько бóльшая по объёму (главы 1, 2 и 5), содержит самые общие сведения из основных разделов дискретной математики. Доля алгоритмов в главах первой группы несколько меньше, а доля определений несколько больше, чем во второй и третьей группах. Во второй группе (главы 3, 4 и 6) собраны различные специальные разделы, которые можно включать или не включать в учебный курс в зависимости от конкретных обстоятельств. Третья группа глав (главы 7–10) целиком посвящена теории графов (в частности, алгоритмам над графами), поскольку теория графов наиболее широко применяется при построении дискретных математических моделей.

Главы делятся на разделы, которые, в свою очередь, делятся на параграфы. Каждый раздел посвящён одному конкретному аспекту темы главы и по объёму соответствует экзаменационному вопросу. Параграфы нужны для внутренних ссылок и более детальной структуризации материала. Как правило, в параграфе рассматривается какое-нибудь одно понятие, теорема или алгоритм. Параграфы в пределах раздела упорядочены по сложности: сначала приводятся основные определения, а затем рассматриваются более сложные вопросы, которые, в принципе, при первом чтении можно пропустить без ущерба для понимания остального материала.

Главы книги более или менее независимы, и их можно изучать в любом порядке, за исключением первой главы, которая содержит набор базисных определений для дальнейшего изложения, и главы 7, в которой вводится специфическая терминология теории графов.

Используемые обозначения

Данная книга предназначена для программистов, то есть предполагается, что читатель не испытывает затруднений в понимании текстов программ на языке высокого

уровня. Именно поэтому в качестве нотации для записи алгоритмов используется *некоторый* неспецифицированный алгоритмический язык (*псевдокод*), похожий по синтаксису на Паскаль (но, конечно, Паскалем не являющийся). Ключевые слова псевдокода выделены полужирным шрифтом, идентификаторы объектов записываются курсивом и часто состоят из одной буквы, как это принято для математических объектов, имена процедур записываются прямым шрифтом, как названия математических функций, примечания отделяются двойной косой чертой //.

В языке используются общеупотребительные структуры управления: ветвления в краткой (**if** ... **then** ... **end if**) и полной (**if** ... **then** ... **else** ... **end if**) формах, циклы со счётчиком (**for** ... **from** ... **to** ... **do** ... **end for**), с постусловием (**repeat** ... **until**), с предусловием (**while** ... **do** ... **end while**) и по множеству (**for** ... **do** ... **end for**), а также переходы (**go to** ...)[1].

Для обозначения присваивания используется традиционный знак :=, вызов процедуры или функции ключевыми словами не выделяется, выход из процедуры вместе с возвратом значения обозначается ключевым словом **return**.

Подразумевается строгая статическая типизация, даже приведения не используются, за исключением разыменования, которое подразумевается везде, где оно нужно по здравому программистскому смыслу. В то же время объявления локальных переменных почти всегда опускаются, если их тип ясен из контекста.

Из структур данных считаются доступными массивы (**array** [...] **of** ...), структуры (**struct {** ... **}**) и указатели (↑).

В программах широко используются математические обозначения, которые являются самоочевидными в конкретном контексте. Например, конструкция

for $x \in M$ **do**
  $P(x)$
end for

означает применение процедуры P ко всем элементам множества M.

«Лишние» разделители систематически опускаются, функции могут возвращать несколько значений, и вообще, разрешаются любые вольности, которые позволяют сделать тексты программ более краткими и читабельными.

Особого упоминания заслуживают три «естественных» оператора, аналоги которых в явном виде редко встречаются в настоящих языках программирования. Оператор

select $m \in M$

означает выбор *произвольного* элемента m из множества M. Этот оператор часто необходим в «переборных» алгоритмах. Оператор

yield x

означает возврат значения x, но при этом выполнение функции не прекращается, а продолжается со следующего оператора. Этот оператор позволяет очень просто записать «генерирующие» алгоритмы, результатом которых является некоторое заранее неизвестное множество значений. Оператор

next for x

означает прекращение выполнения текущего фрагмента программы и продолжение выполнения со следующего шага цикла по переменной x. Этот оператор должен находиться в теле цикла по x (на любом уровне вложенности). Такого рода «структурные переходы» позволяют описывать обработку исключительных ситуаций в циклах

[1] «Структурных» программистов, сомневающихся в допустимости использования оператора **go to** в учебнике, автор отсылает к статье Д. Кнута «Structured Programming with **go to** Statements».

непосредственно, без введения искусственных переменных, отслеживающих состояние вычислений в цикле.

Операторы **select**, **yield** и **next for** не являются чем-то необычным и новым: первый из них легко моделируется использованием датчика псевдослучайных чисел, второй — присоединением элемента к результирующему множеству, а третий — переходом по метке. Однако указанные операторы сокращают запись и повышают её наглядность.

Поскольку эта книга написана на «программно-математическом» языке, в ней не только математические обозначения используются в программах, но и некоторые программистские обозначения используются в математическом тексте.

Прежде всего обсудим способы введения обозначений объектов и операций. Если обозначение объекта или операции является глобальным (в пределах учебника), то используется знак $\stackrel{\text{Def}}{=}$, который следует читать «по определению есть». При этом слева от знака $\stackrel{\text{Def}}{=}$ может стоять обозначение объекта, а справа — выражение, определяющее этот объект. Например,

$$\mathbb{R}_+ \stackrel{\text{Def}}{=} \{x \in \mathbb{R} \mid x > 0\}.$$

В случае определения операции в левой части стоит *выражение*. В этом случае левую часть следует неформально понимать как заголовок процедуры выполнения определяемой операции, а правую часть — как тело этой процедуры. Например, формула

$$A = B \stackrel{\text{Def}}{=} A \subset B \;\&\; B \subset A$$

означает следующее: «чтобы вычислить значение выражения $A = B$, нужно вычислить значения выражений $A \subset B$ и $B \subset A$, а затем вычислить конъюнкцию этих значений».

Отметим широкое использование символа присваивания $:=$, который в математическом контексте можно читать как «положим». Если в левой части такого присваивания стоит простая переменная, а в правой части — выражение, определяющее конкретный объект, то это имеет очевидный смысл введения локального (в пределах доказательства, определения и т. п.) обозначения. Например, запись $Z' := Z \cup Z_{e_k} \setminus \{e_k\}$ означает: «положим, что Z' содержит все элементы Z и Z_{e_k} за исключением e_k». Наряду с обычным для математики «статическим» использованием знака $:=$, когда выражению в левой части придается постоянное в данном контексте значение (определение константы), знак присваивания используется и в «динамическом», программистском смысле. Например, пусть в графе G есть вершина v. Тогда формулу $G := G - v$ следует читать и понимать так: «удалим в графе G вершину v».

Некоторые обозначения являются глобальными, то есть означают одно и то же в любом контексте (в этой книге). Например, буква $\mathbb{R}$ всегда обозначает поле вещественных чисел, а пара вертикальных чёрточек слева и справа от объекта $|X|$ — количество элементов в объекте X. Таких стандартных обозначений не так уж много. Они собраны в указателе обозначений в конце книги и используются в тексте учебника без дополнительных пояснений. Если смысл какого-то обозначения не ясен и не определён в данном контексте, то следует посмотреть это обозначение в указателе обозначений.

Одной из задач книги является выработка у студентов навыка чтения математических текстов. Поэтому, начиная с самой первой страницы, без дополнительных

объяснений интенсивно используются язык исчисления предикатов и другие общепринятые математические обозначения. При этом стиль записи формул совершенно свободный и неформальный, но соответствующий общепринятой практике записи формул в математических текстах. Например, вместо формулы

$$\forall\, k\ \ ((k < n) \Longrightarrow P(k))$$

может быть написано

$$\forall\, k < n\ \ (P(k))$$

или даже

$$\forall\, k < n\ P(k)$$

в предположении, что читатель знает или догадается, какие синтаксические упрощения используются. В первых главах книги основные утверждения (формулировки теорем) дублируются, то есть приводятся на естественном языке и на языке формул. Тем самым на примерах объясняется используемый язык формул. Но в последних главах книги и в доказательствах использование естественного языка сведено к минимуму. Это существенно сокращает объём текста, но требует внимания при чтении.

Для краткости в тексте книги иногда используются обороты, которые подразумевают восстановление читателем опущенных слов по контексту. Например, если символ A означает множество, то вместо аккуратного, но длинного оборота «где объект x является элементом множества A» может быть использована более короткая фраза «где x — элемент A» или даже формула «где $x \in A$».

В целом используемые обозначения строго следуют классическим образцам, принятым в учебной математической и программистской литературе. Непривычным может показаться только совместное использование этих обозначений в одном тексте. Но такое взаимопроникновение обозначений продиктовано основной задачей книги.

Выделения в тексте

В учебнике имеются следующие основные виды текстов: определения, теоремы, леммы и следствия, доказательства и обоснования, замечания, отступления, алгоритмы и примеры. Фактически, обычный связующий текст сведен к минимуму в целях сокращения объёма книги.

Определения никак специально не выделяются, поскольку составляют львиную долю основного текста книги. Вместо этого курсивом выделяются *определяющие вхождения* терминов, а текст, соседствующий с термином, и есть определение. Все определяющие вхождения вынесены в предметный указатель, помещённый в конце книги. Таким образом, если при чтении попадается незнакомый термин, следует найти его определение с помощью указателя (или убедиться, что определения в книге нет). Математические термины в предметном указателе переведены на английский язык. Автор надеется, что предметный указатель послужит читателю как краткий математический русско-английский словарь.

Формулировки теорем, лемм и следствий в соответствии с общепринятой в математической литературе практикой выделены *курсивом*. При этом формулировке предшествует соответствующее ключевое слово: «теорема», «лемма» или «следствие». Как правило, утверждения не нумеруются, за исключением случаев вхождения нескольких однородных утверждений в один параграф. Для ссылок на утверждения используются либо номера параграфов, в которых утверждения сформулированы, либо собственные имена утверждений. Дело в том, что в данном учебнике

теоремами оформлены как простые утверждения, являющиеся непосредственными следствиями определений, так и глубокие факты, действительно заслуживающие статуса теоремы. В последнем случае приводится собственное имя (название теоремы), которое либо выносится в название параграфа (или раздела), либо указывается в скобках после слова «теорема».

Доказательства относятся к предшествующим утверждениям, а *обоснования* — к предшествующим алгоритмам. В учебнике практически нет недоказанных утверждений и приведенных без достаточного обоснования алгоритмов. Из этого правила имеется несколько исключений, то есть утверждений, доказательства которых не приводятся, поскольку автор считает их технически слишком сложными для выбранного уровня изложения. Отсутствие технически сложного доказательства всегда явно отмечается в тексте. Иногда же доказательство опускается, потому что утверждение легко может быть обосновано читателем самостоятельно. Такие случаи не отмечаются в тексте. Доказательства и обоснования начинаются с соответствующего ключевого слова («доказательство» или «обоснование») и заканчиваются специальным значком □. Если формулировка теоремы содержит несколько утверждений или если доказательство состоит из нескольких шагов (частей), то доказательство делится на абзацы, а в начале каждого абзаца указывается [в скобках], что именно в нем доказывается.

Замечания и *отступления* также начинаются с соответствующего ключевого слова: «замечание» или «отступление». Назначение этих абзацев различно. Замечание сообщает некоторые специальные или дополнительные сведения, прямо относящиеся к основному материалу учебника. Отступление не связано непосредственно с основным материалом, его назначение — расширить кругозор читателя и показать связь основного материала с вопросами, лежащими за пределами курса.

ЗАМЕЧАНИЕ

В очень редких случаях *курсив* используется не для выделения определяющих вхождений терминов и формулировок утверждений, а для того, чтобы сделать *эмфатическое* ударение на каком-то слове в тексте.

ОТСТУПЛЕНИЕ

Использование отступлений необходимо, в противном случае у читателя может сложиться ошибочное впечатление о замкнутости изучаемого предмета и его оторванности от других областей знания.

Алгоритмы, как уже было сказано, записаны на псевдокоде, синтаксис которого кратко описан выше. Как правило, перед текстом алгоритма на естественном языке указываются его назначение, а также входные и выходные данные. После алгоритма приводятся его обоснование и иногда пример протокола выполнения.

Примеры

Как правило, примеры приводятся непосредственно вслед за определением понятия, поэтому не используется никаких связующих слов, поясняющих, к чему относятся примеры. В самих примерах интенсивно используются факты, которые должны быть известны читателю из курса математики средней школы, и понятия, рассмотренные в предшествующем тексте книги.

Благодарности

Автор выражает благодарность члену-корреспонденту РАН Святославу Сергеевичу Лаврову за сотни ценных замечаний по тексту первого издания книги, многочисленным студентам, прослушавшим этот курс, за выявление ошибок в тексте, научному редактору, профессору Никите Юрьевичу Нецветаеву, за огромную помощь в работе над учебником и студенткам Александре Медведевой и Анастасии Шабалиной за творческое соучастие в подготовке последнего издания.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция). Можно обратиться непосредственно к автору по адресу fedornovikov51@gmail.com Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства http://www.piter.com.

Глава 1 Множества и отношения

Понятия «множество», «отношение», «функция» и близкие к ним составляют основной словарь дискретной (равно как и «непрерывной») математики. Именно эти базовые понятия рассматриваются в первой главе, тем самым закладывается необходимая основа для дальнейших построений. Особенность изложения состоит в том, что здесь рассматриваются почти исключительно *конечные* множества, а тонкие и сложные вопросы, связанные с рассмотрением бесконечных множеств, излагаются с «программистской» точки зрения. С другой стороны, значительное внимание уделяется «представлению» множеств в программах. Эти вопросы не имеют прямого отношения к собственно теории множеств в её классическом виде, но очень важны для практикующего программиста.

1.1. Множества

При построении доступной для рационального анализа картины мира часто используется термин «объект» для обозначения некой сущности, отделимой от остальных. Выделение объектов — это не более чем произвольный акт нашего сознания. В одной и той же ситуации объекты могут быть выделены по-разному, в зависимости от точки зрения, целей анализа и других обстоятельств. Но как бы то ни было, выделение объектов и их совокупностей — естественный (или даже единственно возможный) способ организации нашего мышления, поэтому неудивительно, что он лежит в основе главного инструмента описания точного знания — математики.

1.1.1. Элементы и множества

Множество — это любая определённая совокупность объектов. Объекты, из которых составлено множество, называются его *элементами*. Элементы множества различны и отличимы друг от друга. Как множествам, так и элементам можно давать имена или присваивать символьные обозначения.

ЗАМЕЧАНИЕ

В современной математике основным способом определения фундаментальных понятий является *аксиоматический метод*. В рамках этого метода группа понятий, образующих основу некоторой теории, определяется путем постулирования набора *аксиом* (утверждений об этих понятиях, принимаемых без доказательства), так что остальные утверждения выводятся из аксиом с помощью логических доказательств. Применение аксиоматического метода предполагает наличие развитого логического языка для формулировки утверждений и, как правило, требует значительных усилий, времени и места для построения строгих формальных доказательств. Теория множеств не является исключением — для неё предложено несколько систем аксиом, весьма поучительных и оказавших значительное влияние на развитие математики в целом. Альтернативой аксиоматическому методу является апелляция к интуиции, здравому смыслу и использование неопределяемых понятий. Применительно к теории множеств такой подход получил название «наивной теории множеств», элементы которой здесь излагаются.

Примеры

Множество S страниц в данной книге. Множество $\mathbb{N}$ *натуральных* чисел {1, 2, 3, …}. Множество $\mathbb{P}$ *простых* чисел {2, 3, 5, 7, 11, …}. Множество $\mathbb{Z}$ целых чисел

$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$. Множество $\mathbb{R}$ *вещественных* чисел. Множество A различных символов на этой странице.

Если объект x является элементом множества M, то говорят, что элемент x *принадлежит* множеству M. Обозначение: $x \in M$. В противном случае говорят, что x *не принадлежит* M. Обозначение: $x \notin M$.

ЗАМЕЧАНИЕ

Обычно множества обозначают прописными буквами латинского алфавита, а элементы множеств — строчными буквами.

Множество, не содержащее элементов, называется *пустым* и обозначается $\varnothing$.

ЗАМЕЧАНИЕ

Введение в рассмотрение пустого множества является сильным допущением. Например, известно, что синих лошадей в природе не бывает. Тем не менее мы позволяем себе рассматривать «множество синих лошадей» как полноправный объект, вводить для него обозначения и т. д.

ОТСТУПЛЕНИЕ

Понятия множества, элемента и принадлежности, которые на первый взгляд представляются интуитивно ясными, при ближайшем рассмотрении такую ясность утрачивают. Во-первых, даже возможность различить элементы при более глубоком рассмотрении представляет некоторую проблему. Например, символы «а» и «*а*», которые встречаются на этой странице, — это один элемент множества A или два разных элемента? Или же, например, два вхождения символа «о» в слово «множество» — графически они неразличимы невооружённым глазом, но это символы букв, которые обозначают звуки, а читаются эти две гласные по-разному: первая под ударением, а вторая — безударная. Во-вторых, проблематична возможность (без дополнительных усилий) указать, принадлежит ли данный элемент данному множеству. Например, является ли число 869584769215374850678574 67 простым?

Множества, как объекты, могут быть элементами других множеств. Множество, элементами которого являются множества, иногда называют *семейством*.

ЗАМЕЧАНИЕ

Семейства множеств часто обозначают прописными «рукописными» буквами латинского алфавита.

Совокупность объектов, которая *не* является множеством, называется *классом*. Неформально говоря, называя совокупность элементов классом, а не множеством, мы берём на себя сравнительно мéньшую ответственность за определённость, отличимость и бесповторность элементов.

ОТСТУПЛЕНИЕ

Термин «класс» в объекто-ориентированном программировании (ООП) употребляется правомерно. Можно было бы сказать, что класс в ООП — это множество объектов, являющихся экземплярами класса. Однако такое определение не вполне корректно. Например, возможность динамического изменения класса объекта, существующая в некоторых языках ООП, ставит под сомнение статическую определённость класса, а возможность клонирования объектов ставит под сомнение возможность различить экземпляры как элементы класса.

Обычно в конкретных рассуждениях элементы всех рассматриваемых множеств (и семейств) берутся из некоторого одного, достаточно широкого множества U (своего для каждого случая), которое называется *универсальным* множеством (или *универсумом*).

1.1.2. Задание множеств

Чтобы задать множество, нужно указать, какие элементы ему принадлежат. Это указание заключают в пару фигурных скобок, оно может иметь одну из следующих основных форм:

перечисление элементов: $M := \{a, b, c, \ldots, z\}$;
характеристический предикат: $M := \{x \mid P(x)\}$;
порождающая процедура: $M := \{x \mid x := f\}$.

При задании множеств перечислением обозначения элементов разделяют запятыми. *Характеристический предикат* — это некоторое условие, выраженное в форме логического утверждения или процедуры, возвращающей логическое значение, и позволяющее проверить, принадлежит ли любой данный элемент множеству. Если для данного элемента условие выполнено, то он принадлежит определяемому множеству, в противном случае — не принадлежит. Порождающая процедура — это процедура, которая в процессе работы порождает объекты, являющиеся элементами определяемого множества.

Примеры

1. $M_9 := \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
2. $M_9 := \{n \mid n \in \mathbb{N} \;\&\; n < 10\}$.
3. $M_9 := \{n \mid n := 0;$ **for** i **from** 1 **to** 9 **do** $n := n + 1;$ **yield** n **end for**$\}$.

При задании множеств перечислением обозначения элементов иногда снабжают индексами и указывают множество, из которого берутся индексы. В частности, запись $\{a_i\}_{i=1}^{k}$ означает то же, что $\{a_1, \ldots, a_k\}$, а запись $\mathcal{M} = \{M_\alpha\}_{\alpha \in A}$ означает, что $\mathcal{M}$ является семейством, элементами которого являются множества M_α, причём индекс α «пробегает» множество A. Знак многоточия (...), который употребляется при задании множеств, является сокращённой формой записи, в которой порождающая процедура для множества индексов считается очевидной.

Пример. $\{a_1, a_2, a_3, \ldots\} = \{a_i\}_{i=1}^{\infty}$.

Множество целых чисел в диапазоне от m до n в этой книге обозначается так: $m..n \stackrel{\text{Def}}{=} \{k \in \mathbb{Z} \mid m \leqslant k \;\&\; k \leqslant n\} = \{k \in \mathbb{Z} \mid$ **for** k **from** m **to** n **do yield** k **end for**$\}$.

Перечислением элементов можно задавать только *конечные* множества. *Бесконечные* множества задаются характеристическим предикатом или порождающей процедурой.

Пример. $\mathbb{N} \stackrel{\text{Def}}{=} \{n \mid n := 0;$ **while true do** $n := n + 1;$ **yield** n **end while**$\}$.

Если $x \notin A$, то элемент x можно *добавить* в множество A:
$A + x \stackrel{\text{Def}}{=} \{y \mid y \in A \vee y = x\}$.
Если $x \in A$, то элемент x можно *удалить* из множества A:
$A - x \stackrel{\text{Def}}{=} \{y \mid y \in A \ \& \ y \neq x\}$.

Примеры

$M_9 := \varnothing + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9$.

Множество натуральных чисел с нулём: $\mathbb{N}_0 \stackrel{\text{Def}}{=} \mathbb{N} + 0$.

ОТСТУПЛЕНИЕ

Операции добавления и удаления элементов являются частными случаями операций объединения и разности множеств, рассматриваемых в п. 1.2.6 (а именно: $A + x = A \cup \{x\}$, $A - x = A \setminus \{x\}$), и потому, строго говоря, излишни. В стандартных учебниках по теории множеств такие операции не рассматриваются и не упоминаются. Здесь они введены для упрощения обозначений. Такой подход акцентирует «программистское» отношение к математике: мы вводим новые операции, если это практически удобно, даже если это теоретически излишне.

1.1.3. Парадокс Рассела

Возможность задания множеств характеристическим предикатом зависит от предиката. Использование некоторых предикатов для этой цели может приводить к противоречиям. Например, все рассмотренные в примерах множества не содержат себя в качестве элемента. Рассмотрим множество Y всех множеств, не содержащих себя в качестве элемента: $Y := \{X \mid X \notin X\}$. Если множество Y существует, то мы должны иметь возможность ответить на следующий вопрос: $Y \in Y$? Пусть $Y \in Y$, тогда $Y \notin Y$. Пусть $Y \notin Y$, тогда $Y \in Y$. Получается неустранимое логическое противоречие, которое известно как *парадокс Рассела*[1]. Три способа избежать этого конкретного парадокса.

1. Ограничить используемые характеристические предикаты видом
$$P(x) = x \in A \ \& \ Q(x),$$
где A — известное, заведомо существующее множество (*универсум*). Обычно при этом используют обозначение $\{x \in A \mid Q(x)\}$. Для Y универсум не указан, а потому Y множеством не является.
2. Теория типов. Объекты имеют тип 0, множества элементов типа 0 имеют тип 1, множества элементов типа 0 и 1 — тип 2 и т. д. Y не имеет типа и множеством не является.
3. Явный запрет принадлежности множества самому себе: $X \in X$ — недопустимый предикат. При аксиоматическом построении теории множеств соответствующая аксиома называется *аксиомой регулярности*.

ОТСТУПЛЕНИЕ

Существование и анализ парадоксов в теории множеств способствовали развитию так называемого *конструктивизма* — направления в математике, в рамках которого рассматриваются

[1] Бертран Рассел (1872–1970).

только такие объекты, для которых известны процедуры (алгоритмы) их порождения. В конструктивной математике исключаются из рассмотрения те понятия и методы классической математики, которые не заданы алгоритмически.

Парадокса Рассела можно избежать, ограничив рассматриваемые множества. Например, достаточно запретить рассматривать в качестве множеств классы, содержащие самих себя. При этом, однако, нет полной уверенности в том, что не обнаружатся другие противоречия. Полноценным выходом из ситуации являлось бы аксиоматическое построение теории множеств и доказательство непротиворечивости построенной формальной теории. Однако исследование парадоксов и непротиворечивости систем аксиом является технически трудной задачей и уводит далеко в сторону от программистской практики, для которой важнейшими являются конечные множества. Поэтому формальная аксиоматика теории множеств здесь не приводится. Мы излагаем необходимые сведения полуформально, опираясь везде, где это возможно, на программистскую интуицию читателя.

1.1.4. Мультимножества

В множестве все элементы различны, а значит, входят в множество ровно один раз. В некоторых случаях оказывается полезным рассматривать совокупности элементов, в которые элементы входят по нескольку раз.

Пусть $X = \{x_1, \ldots, x_n\}$ — некоторое множество и пусть $a_1, \ldots, a_n$ — неотрицательные целые числа. Тогда *мультимножеством* $\widehat{X}$ называется совокупность элементов множества X, в которую элемент x_i входит a_i раз. Мультимножество обозначается одним из следующих способов:

$$\widehat{X} = [x_1^{a_1}, \ldots, x_n^{a_n}] = \langle x_1, \ldots, x_1; \ldots; x_n, \ldots, x_n\rangle = \langle a_1(x_1), \ldots, a_n(x_n)\rangle.$$

Пример. Пусть $X = \{a, b, c\}$. Тогда $\widehat{X} = \left[a^0b^3c^4\right] = \langle b, b, b; c, c, c, c\rangle = \langle 0(a), 3(b), 4(c)\rangle$.

Пусть $\widehat{X} = \langle a_1(x_1), \ldots, a_n(x_n)\rangle$ — мультимножество над множеством $X = \{x_i\}_{i=1}^n$. Тогда число a_i называется *показателем* элемента x_i, множество X — *носителем*, число $m = a_1 + \ldots + a_n$ — *мощностью*, а $\underline{\widehat{X}} = \{x_i \in X \mid a_i > 0\}$ называется *составом* мультимножества $\widehat{X}$.

Пример. Пусть $\widehat{X} = \left[a^0b^3c^4\right]$ — мультимножество над множеством $X = \{a, b, c\}$. Тогда $\underline{\widehat{X}} = \{b, c\}$.

Мультимножество $\widehat{X} = \langle a_1(x_1), \ldots, a_n(x_n)\rangle$ над множеством $X = \{x_1, \ldots, x_n\}$ называется *индикатором*, если $\forall\, i \in 1..n\ \ (a_i = 0 \vee a_i = 1)$.

1.1.5. Конечные последовательности

К понятию мультимножества тесно примыкает широко используемое понятие (конечной) последовательности. Пусть задано множество $X = \{x_1, \ldots, x_n\}$. Рассмотрим мультимножество $\widehat{X} = \langle a_1(x_1), \ldots, a_n(x_n)\rangle$ над X, $a_1 + \cdots + a_n = m$. Элементы мультимножества можно *перенумеровать*, назначив им номера из отрезка $1..m$. Это можно сделать многими различными способами. Мультимножество с конкретной нумерацией называется *последовательностью* длины m над множеством элементов X. Другими словами, если задана последовательность, то всегда можно сказать,

какой элемент стоит на i-м месте (i-й позиции) в последовательности. Отсюда вытекает общепринятый способ записи последовательностей: выписать элементы мультимножества слева направо в порядке возрастания номеров.

Пример. Пусть $X = \{a, b\}$ — исходное множество элементов, $\widehat{X} = \langle 2(a), 2(b)\rangle$ — мультимножество, тогда *aabb*, *abab*, *abba*, *baab*, *baba*, *bbaa* — все шесть возможных в данном случае последовательностей.

ЗАМЕЧАНИЕ

Конечную последовательность чисел часто называют *вектором*.

ОТСТУПЛЕНИЕ

Одномерный массив A : **array** $[1..n]$ **of** X является программным представлением последовательности над множеством X. При этом множество X называется *типом* (элементов) массива, а выражение $A[i]$ доставляет i-й элемент последовательности. Выражение $A[i]$ вычисляется очень эффективно, при условии, что все элементы массива (последовательности) занимают одинаковый объём памяти компьютера.

Пусть теперь элементами исходного множества являются *буквы*, или *символы*. В таком случае множество букв называется *алфавитом* и обычно обозначается A.

Последовательность букв называется *словом*, или *цепочкой*, в данном алфавите A. Одна буква может входить в слово несколько раз, каждый отдельный экземпляр буквы в слове называется *вхождением* буквы в слово.

Если слово $\alpha = a_1 \dots a_k, a_i \in A$, то количество букв в слове называется *длиной* слова: $|\alpha| = |a_1 \dots a_k| = k$. *Пустое слово* обозначается ε; $|\varepsilon| \stackrel{\text{Def}}{=} 0$.

Множество слов в алфавите A, включая пустое слово, обозначается A^*. Пустое слово не содержит букв алфавита, но считается словом: $\varepsilon \notin A$, но $\varepsilon \in A^*$. Если пустое слово исключается из рассмотрения, то множество слов в алфавите A обозначается A^+.

Для слов определена операция *конкатенации*, или *сцепления*. Обычно операция конкатенации никак не обозначается, а сцепляемые слова просто записываются подряд.

Пример. Конкатенация слов «пара» и «план» образует слово «параплан».

Если $\alpha = \alpha_1\alpha_2$, то α_1 называется *началом*, или *префиксом*, слова α, а α_2 — *окончанием*, или *постфиксом*, слова α.

Пример. Слова «д» и «до» являются префиксами, а слова «ом» и «м» являются постфиксами слова «дом».

Слова α и β могут *пересекаться*. При пересечении возможны два случая: либо одно слово является частью другого, $\alpha = \alpha_1\beta\alpha_2$, либо префикс одного слова является постфиксом другого, $\alpha = \alpha_1\gamma, \beta = \gamma\beta_2$.

Произвольное множество L слов в некотором алфавите называется *языком* в этом алфавите; $L \subset A^*$.

1.2. Алгебра подмножеств

Самого по себе понятия множества ещё недостаточно — нужно определить способы конструирования новых множеств из уже имеющихся, то есть определить *операции* над множествами.

Рассматриваемый совместно набор операций, определённых на некотором множестве, образует *алгебру*. При этом рассматриваемое множество объектов называют *носителем* алгебры (см. п. 2.1.1).

Пример. *Арифметика*, изучаемая в начальной школе, — это алгебра, включающая операции сложения ($+$), умножения ($\times$), вычитания ($-$) и деления с остатком ($/$), причём носителем является множество натуральных чисел и ноль.

Наряду с операциями, результаты которых принадлежат носителю, часто бывают необходимы и такие операции, результаты которых не принадлежат носителю.

Пример. В арифметике, наряду с арифметическими операциями, используются *операции сравнения*: больше ($>$), меньше ($<$) и другие операции.

1.2.1. Сравнение множеств

Множество A *содержится* в множестве B (множество B *включает* множество A), если каждый элемент множества A есть элемент множества B:

$$A \subset B \stackrel{\text{Def}}{=} x \in A \Longrightarrow x \in B.$$

При этом говорят, что A — *подмножество* B, B — *надмножество* A. По определению $\forall M \ \ (\varnothing \subset M)$.

ЗАМЕЧАНИЕ

Нетрудно видеть, что понятие подмножества множества X равнообъёмно понятию индикатора над множеством X. Действительно, если $X = \{x_1, \ldots, x_n\}$ — некоторое множество, а Y — любое подмножество множества X, $Y \subset X$, то существует единственный индикатор $\widehat{Y} = \langle a_1(x_1), \ldots, a_n(x_n)\rangle$ над множеством X, такой, что $\forall\, i \in 1..n \ \ (a_i = 1 \Longleftrightarrow x_i \in Y)$.

Если $A \subset B$ и $A \neq B$, то A называется *собственным подмножеством* B, а B — *собственным надмножеством* A.

ЗАМЕЧАНИЕ

Для обозначения включения собственных подмножеств иногда используется знак $\subsetneq$, а для несобственных — $\subseteq$.

ТЕОРЕМА. *Включение множеств обладает следующими свойствами:*

1) $\forall A \ \ (A \subset A)$;
2) $\forall A, B \ \ (A \subset B \ \&\ B \subset A \Longrightarrow A = B)$;
3) $\forall A, B, C \ \ (A \subset B \ \&\ B \subset C \Longrightarrow A \subset C)$.

Доказательство.

[1] Имеем $x \in A \Longrightarrow x \in A$ и, значит, $A \subset A$.

[2] По определению.

[3] Если $x \in A \Longrightarrow x \in B$ и $x \in B \Longrightarrow x \in C$, то $x \in A \Longrightarrow x \in C$ и, значит, $A \subset C$. □

СЛЕДСТВИЕ. $A \subsetneq B \subseteq C \Longrightarrow A \subsetneq C$.

Два множества *равны*, если они являются подмножествами друг друга:

$$A = B \stackrel{\text{Def}}{=} A \subset B \ \& \ B \subset A.$$

1.2.2. Равномощные множества

Говорят, что между множествами A и B установлено *взаимно-однозначное соответствие*, если каждому элементу A поставлен в соответствие один и только один элемент B, и для каждого элемента B один и только один элемент A поставлен в соответствие этому элементу B. В этом случае говорят также, что множества A и B *изоморфны*, и используют обозначение $A \sim B$. Если при заданном соответствии элементу $a \in A$ соответствует элемент $b \in B$, то данное обстоятельство обозначают следующим образом: $a \mapsto b$.

ЗАМЕЧАНИЕ

Весьма общее понятие соответствия трактуется здесь с программистских позиций. Если сказано, что между множествами A и B установлено соответствие, то подразумевается, что задан способ по любому элементу $a \in A$ определить соответствующий ему элемент $b \in B$. Способ может быть любым, если возможность его применения не вызывает сомнений. Например, это может быть массив **array** $[A]$ **of** B, или процедура типа **proc** $(A) : B$, или же выражение, зависящее от переменной типа A и доставляющее значение типа B.

Пример. Соответствие $n \mapsto 2n$ устанавливает взаимно-однозначное соответствие между множеством натуральных чисел $\mathbb{N}$ и множеством чётных натуральных чисел $2\mathbb{N}$; $\mathbb{N} \sim 2\mathbb{N}$.

Если между двумя множествами, A и B, может быть установлено взаимно-однозначное соответствие, то говорят, что множества имеют одинаковую *мощность*, или что множества *равномощны*, и записывают это так: $|A| = |B|$. Другими словами,

$$|A| = |B| \stackrel{\text{Def}}{=} A \sim B.$$

ТЕОРЕМА. *Равномощность множеств обладает следующими свойствами:*

1) $\forall A \ (|A| = |A|)$;
2) $\forall A, B \ (|A| = |B| \Longrightarrow |B| = |A|)$;
3) $\forall A, B, C \ (|A| = |B| \ \& \ |B| = |C| \Longrightarrow |A| = |C|)$.

Доказательство.

[1] любое множество взаимно-однозначно соответствует самому себе: $A \sim A$, то есть достаточно рассмотреть соответствие $a \mapsto a$, где $a \in A$;

[2] если $A \sim B$, то $B \sim A$ — достаточно использовать соответствие $a \mapsto b$ для построения соответствия $b \mapsto a$, где $a \in A, b \in B$;

[3] если $A \sim B$ и $B \sim C$, то $A \sim C$ — соответствие устанавливается с использованием промежуточного элемента $b \in B$: $a \mapsto b \mapsto c$, где $a \in A, b \in B, c \in C$. □

1.2.3. Конечные и бесконечные множества

Вопрос о том, чем *конечное* отличается от *бесконечного*, неподготовленного человека может поставить в тупик. Здесь мы рассматриваем применительно к множествам один из возможных ответов на этот вопрос. С античных времен известен принцип: «часть меньше целого». Оказывается, этот принцип можно использовать в качестве характеристического свойства конечных множеств. Для бесконечных множеств этот принцип не имеет места.

Множество A называется *конечным*, если у него нет равномощного собственного подмножества: $\forall B \ ((B \subset A \ \& \ |B| = |A|) \Longrightarrow (B = A))$. Для конечного множества A используется запись $|A| < \infty$. Все остальные множества называются *бесконечными*: $\exists B \ (B \subset A \ \& \ |B| = |A| \ \& \ B \neq A)$, то есть бесконечное множество равномощно некоторому своему собственному подмножеству. Для бесконечного множества A используется запись $|A| = \infty$.

Пример. Множество натуральных чисел бесконечно, $|\mathbb{N}| = \infty$, поскольку оно равномощно своему собственному подмножеству чётных чисел.

ТЕОРЕМА. *Множество, имеющее бесконечное подмножество, бесконечно:*

$$(B \subset A \ \& \ |B| = \infty) \Longrightarrow (|A| = \infty).$$

Доказательство.
Множество B бесконечно, значит $B \sim C$, причём $C \subset B$ (рис. 1.1). Обозначим это соответствие $x \mapsto x'$. Построим соответствие между множеством A и его собственным подмножеством D: $x \mapsto$ **if** $x \in B$ **then** x' **else** x **end if**. Другими словами, на элементах из B мы пользуемся заданным соответствием, а остальным элементам сопоставляем их самих. Это взаимно-однозначное соответствие между множеством A и его собственным подмножеством D, и, значит, $|A| = \infty$. □

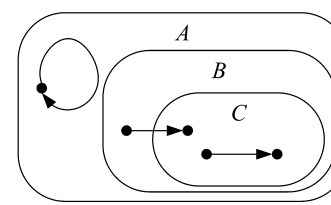


Рис. 1.1. К доказательству теоремы 1.2.3

ЗАМЕЧАНИЕ

В обозначениях п. 1.2.6 $D = C \cup (A \setminus B)$.

СЛЕДСТВИЕ. *Все подмножества конечного множества конечны.*

1.2.4. Счётные и несчётные множества

Бесконечные множества, равномощные множеству натуральных чисел, называются *счётными*. Счётные множества отличаются тем, что для каждого элемента можно указать взаимно-однозначно соответствующее натуральное число, то есть *номер*. Другими словами, множество счётно, если его элементы можно *перенумеровать*.

Пример. Множество целых чисел $\mathbb{Z}$ счётно. Соответствие задаётся следующим образом: $0 \mapsto 1$, если $n > 0$, то $n \mapsto 2n$; если $n < 0$, то $n \mapsto 2|n| + 1$.

Фактическая нумерация элементов счётного множества не всегда столь проста, иногда требуются более изощрённые построения.

Пример. Множество точек на плоскости с натуральными координатами счётно. Для построения нумерации используются так называемые *треугольные числа* $T(k)$, вычисляемые по формуле $T(k) = k(k+1)/2 = \sum_{i=1}^{k} i = T(k-1)+k$. На рис. 1.2 слева показано начало одной из многих возможных нумераций точек плоскости с натуральными координатами с помощью треугольных чисел. Заметим, что для всех точек (m, n), лежащих на каждой «обратной диагонали», сумма $m + n$ постоянна, а точки, лежащие на вертикальной оси $m = 1$, имеют номера, являющиеся треугольными числами (рис. 1.2 справа).

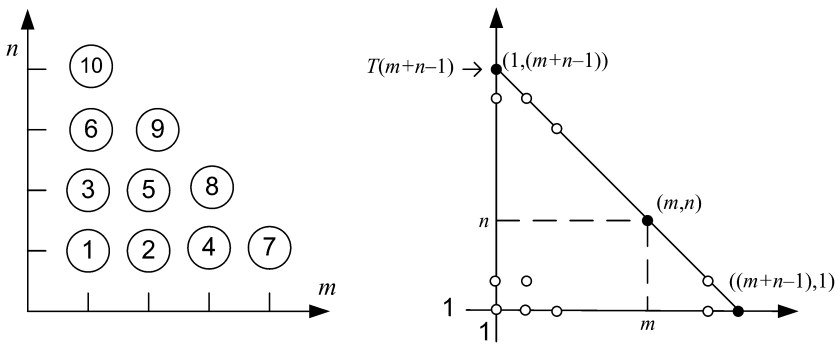


Рис. 1.2. Нумерация точек плоскости с помощью треугольных чисел

Вернёмся теперь к установлению взаимно-однозначного соответствия между точками плоскости и натуральными числами. Точка с координатами (m, n) получает номер k, вычисляемый следующим образом:

$$k := T(m + n - 1) - m + 1 = (m + n)(m + n - 1)/2 - m + 1.$$

Если точка имеет номер k в данной нумерации, то её координаты (m, n) вычисляются следующим образом:

$i := 1;\ T := 1$
while $T < k$ **do**
  $i := i + 1;\ T := T + i$
end while
$m := T - k + 1;\ n := i - m + 1$

ЗАМЕЧАНИЕ

Почему числа $T(n)$ называются треугольными, объяснено на рис. 1.3.

Не все бесконечные множества являются счётными. Существуют *несчётные* множества.

ТЕОРЕМА. *Множество всех подмножеств натурального ряда несчётно.*

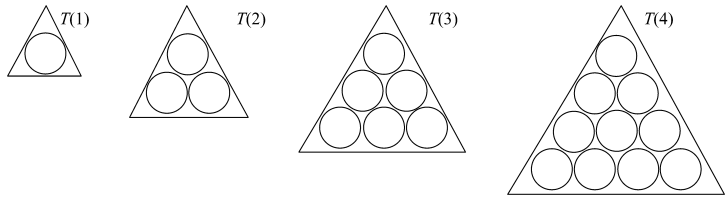


Рис. 1.3. Треугольные числа

Доказательство. Допустим, что множество всех подмножеств натурального ряда счётно. Тогда их можно перенумеровать, и каждое множество чисел получит свой номер. Обозначим $N(X)$ — номер множества X. Может статься, что некоторые множества чисел содержат свой номер в качестве элемента, а другие не содержат. Рассмотрим числовое множество Y, которое состоит из номеров множеств, не содержащих свой номер в качестве элемента:

$$Y := \{x \in \mathbb{N} \mid \exists X \subset \mathbb{N} \ \ (x = N(X) \ \& \ x \notin X)\}.$$

Пусть множество Y имеет номер $y := N(Y)$. Тогда, если $y \notin Y$, то множество Y не содержит свой номер, а значит, по своему определению, $y \in Y$. Обратно: если $y \in Y$, то $y \notin Y$. Получается неустранимое логическое противоречие, и значит, множество Y невозможно. Но множество Y было построено на основании предположения о существовании нумерации всех числовых множеств. □

Однако несчётные множества могут иметь счётные подмножества. Следующий алгоритм перечисляет все *конечные* подмножества натурального ряда, откуда следует, что бесконечное множество всех конечных подмножеств натурального ряда счётно.

Алгоритм 1.1. Перечисление конечных подмножеств натурального ряда

```
n := 0  // добавляемое число
Q ← ∅  // очередь множеств содержит пустое множество
yield ∅  // первое подмножество пустое
while true do
  S ← Q  // извлекаем первое множество из очереди
  if S = ∅ then n := n + 1 end if  // следующее добавляемое число
  Q ← S  // возвращаем извлечённое множество в очередь
  S := S + n  // добавляем число n в множество S
  Q ← S  // добавляем новое множество в очередь
  yield S  // очередное множество
end while
```

Обоснование. Множество всех конечных подмножеств отрезка $1..n$ состоит из двух равных частей: множества всех подмножеств отрезка $1..(n-1)$ и множества этих же подмножеств, в каждое из которых добавлено число n. □

1.2.5. Мощность конечного множества

ТЕОРЕМА 1. *Любое непустое конечное множество равномощно некоторому отрезку натурального ряда:* $\forall A \ (A \neq \varnothing \ \& \ |A| < \infty \Longrightarrow \exists k \in \mathbb{N} \ (|A| = |1..k|))$.

Доказательство. Рассмотрим следующую программу:

$i := 0$ // счётчик элементов
while $A \neq \varnothing$ **do**
  select $x \in A$ // выбираем элемент
  $i := i + 1$ // увеличиваем счётчик
  $x \mapsto i$ // ставим элементу в соответствие его номер
  $A := A - x$ // удаляем элемент из множества
end while

Если эта программа не заканчивает работу, то она даёт соответствие $B \sim \mathbb{N}$ для некоторого множества $B \subset A$, что невозможно ввиду конечности A. Значит, процедура заканчивает работу при $i = k$. Но в этом случае построено взаимно-однозначное соответствие $A \sim 1..k$. □

Лемма. *Любое непустое подмножество множества натуральных чисел содержит наименьший элемент.*

Доказательство. Пусть A — произвольное подмножество множества натуральных чисел, конечное или бесконечное. Рассмотрим задание множества A следующей порождающей процедурой:

$n := 0$ // натуральное число
while true do
  $n := n + 1$ // следующее натуральное число
  if $n \in A$ **then**
    yield n // число входит в множество A
  end if
end while

Ясно, что множество A действительно порождается этой процедурой, причём тот элемент, который порождается первым, является наименьшим. □

Теорема 2. *Любой отрезок натурального ряда конечен:* $\forall\, n \in \mathbb{N}\ (|1..n| < \infty)$.

Доказательство. От противного. Пусть существуют бесконечные отрезки натурального ряда. Рассмотрим наименьшее n такое, что $|1..n| = \infty$. Тогда отрезок $1..n$ равномощен некоторому своему собственному подмножеству A: $|1..n| = |A|$, то есть $1..n \sim A$. Пусть при этом соответствии $n \mapsto i$. Рассмотрим соответствие между отрезком $1..(n-1)$ и его собственным подмножеством $A - i$, задаваемое соответствием между $1..n$ и A. Это соответствие является взаимно-однозначным, а значит, отрезок $1..(n-1)$ изоморфен своему собственному подмножеству и является бесконечным, что противоречит выбору n. □

Следствие. *Различные отрезки натурального ряда неравномощны:* $n \neq m \Longrightarrow |1..n| \neq |1..m|$.

Доказательство. Пусть для определённости $n > m$. Тогда $1..m \subset 1..n$ и $1..m \neq 1..n$. Если $|1..n| = |1..m|$, то $|1..m| = \infty$, что противоречит теореме. □

Говорят, что конечное множество A имеет *мощность* k (обозначения: $|A| = k$, $\operatorname{card} A = k$, $\#A = k$), если оно равномощно отрезку $1..k$: $|A| = k \stackrel{\text{Def}}{=} A \sim 1..k$.

ЗАМЕЧАНИЕ

Таким образом, если множество A конечно, $|A| = k$, то элементы A всегда можно *перенумеровать*, то есть поставить им в соответствие номера из отрезка $1..k$ с помощью некоторой процедуры. Наличие такой процедуры подразумевается, когда употребляется запись $A = \{a_1, \ldots, a_k\}$.

По определению $|\varnothing| \stackrel{\text{Def}}{=} 0$.

ЗАМЕЧАНИЕ

Последнее доказанное следствие часто излагают в следующей форме: если $n \neq m$, то n кроликов невозможно рассадить в m ящиков так, чтобы в каждом ящике было по одному кролику. Это утверждение называется *принцип Дирихле*. Действительно, если бы рассадить кроликов было возможно, то было бы установлено взаимно-однозначное соответствие между различными отрезками натурального ряда.

1.2.6. Операции над множествами

Обычно рассматриваются следующие операции над множествами:

объединение: $A \cup B \stackrel{\text{Def}}{=} \{x \mid x \in A \vee x \in B\}$;

пересечение: $A \cap B \stackrel{\text{Def}}{=} \{x \mid x \in A \;\&\; x \in B\}$;

разность: $A \setminus B \stackrel{\text{Def}}{=} \{x \mid x \in A \;\&\; x \notin B\}$;

симметрическая разность: $A \vartriangle B \stackrel{\text{Def}}{=} (A \cup B) \setminus (A \cap B)$;

$A \vartriangle B \stackrel{\text{Def}}{=} \{x \mid (x \in A \;\&\; x \notin B) \vee (x \notin A \;\&\; x \in B)\}$;

дополнение: $\overline{A} \stackrel{\text{Def}}{=} \{x \mid x \notin A\}$.

Результаты операций объединения, пересечения, разности и симметрической разности множеств всегда определены и являются множествами. Операция дополнения подразумевает, что задан некоторый универсум U, и в этом случае $\overline{A} = U \setminus A$. В противном случае операция дополнения не определена.

Пример. Пусть $A := \{1, 2, 3\}$, $B := \{3, 4, 5\}$. Тогда $A \cup B = \{1, 2, 3, 4, 5\}$, $A \cap B = \{3\}$, $A \setminus B = \{1, 2\}$, $A \vartriangle B = \{1, 2, 4, 5\}$.

В некоторых случаях операции над множествами выразимы через комбинации других операций. В следующих двух таблицах приведены выражения для каждой из четырёх операций через пару других операций.

	$\cup, \cap$	$\cup, \setminus$	$\cup, \vartriangle$
$A \cup B =$	$A \cup B$	$A \cup B$	$A \cup B$
$A \cap B =$	$A \cap B$	$A \setminus (A \setminus B)$	$(A \cup B) \vartriangle (A \vartriangle B)$
$A \setminus B =$	—	$A \setminus B$	$(A \cup B) \vartriangle B$
$A \vartriangle B =$	—	$(A \setminus B) \cup (B \setminus A)$	$A \vartriangle B$

	$\cap, \setminus$	$\cap, \vartriangle$	$\setminus, \vartriangle$
$A \cup B =$	—	$(A \vartriangle B) \vartriangle (A \cap B)$	$(A \vartriangle B) \vartriangle (A \setminus (A \setminus B))$
$A \cap B =$	$A \cap B$	$A \cap B$	$A \setminus (A \setminus B)$
$A \setminus B =$	$A \setminus B$	$A \vartriangle (A \cap B)$	$A \setminus B$
$A \vartriangle B =$	—	$A \vartriangle B$	$A \vartriangle B$

В то же время, в четырех случаях выражения не существует. Например, разность невыразима через объединение и пересечение. Действительно, если $A = B \neq \varnothing$, то разность $A \setminus B = \varnothing$, при этом любые комбинации объединения и пересечения непустого множества с самим собой дают это же самое множество. Объединение невыразимо через пересечение и разность. А именно, результат пересечения и разности всегда является подмножеством первого аргумента, и поэтому значение любого выражения, составленного из пересечений и разностей, будет подмножеством одного из исходных аргументов.

На рис. 1.4 приведены *диаграммы Венна*, иллюстрирующие операции над множествами. Сами исходные множества изображаются фигурами (в данном случае прямоугольниками), а результат выделяется графически (в данном случае для выделения использована заливка серым).

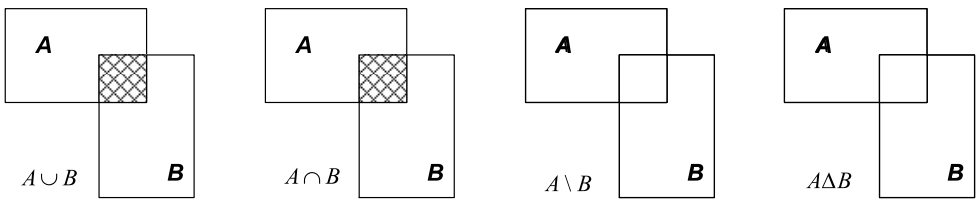


Рис. 1.4. Диаграммы Венна операций над множествами

Если множества A и B конечны, то из определений и диаграмм Венна нетрудно видеть, что

$$|A \cup B| = |A| + |B| - |A \cap B|,\ |A \setminus B| = |A| - |A \cap B|,\ |A \triangle B| = |A| + |B| - 2|A \cap B|.$$

Операции пересечения и объединения двух множеств допускают следующее обобщение. Пусть задано семейство множеств $\{A_i\}_{i \in I}$. Тогда

$$\bigcup_{i \in I} A_i \stackrel{\text{Def}}{=} \{x \mid \exists\, i \in I\ \ (x \in A_i)\}, \qquad \bigcap_{i \in I} A_i \stackrel{\text{Def}}{=} \{x \mid \forall\, i \in I\ \ (x \in A_i)\}.$$

ЗАМЕЧАНИЕ

Если операция ассоциативна (см. п. 1.2.9), то неважно, как расставлены скобки в выражении с этой операцией. В таком случае можно считать, что операция допускает переменное количество аргументов, два или более, и считать операцию *групповой*. Примерами часто используемых групповых операций являются введённые объединение $\bigcup$ и пересечение $\bigcap$, а также хорошо известные сумма $\sum$ и произведение $\prod$.

1.2.7. Разбиения и покрытия

Пусть $\mathcal{E} = \{E_i\}_{i \in I}$ — семейство подмножеств множества M; $E_i \subset M$. Семейство $\mathcal{E}$ называется *покрытием* множества M, если каждый элемент M принадлежит хотя бы одному из E_i, то есть $\forall\, x \in M\ \ (\exists\, i \in I\ \ (x \in E_i))$. Семейство $\mathcal{E}$ называется *дизъюнктным*, если элементы этого семейства попарно не пересекаются, то есть каждый элемент M принадлежит не более чем одному из множеств E_i, то есть $\forall\, i, j \in I\ \ (i \neq j \Longrightarrow E_i \cap E_j = \varnothing)$. Дизъюнктное покрытие называется *разбиением* множества M. Элементы разбиения, то есть подмножества множества M, часто называют *блоками* разбиения.

ТЕОРЕМА. *Если $\mathcal{E} = \{E_i\}_{i\in I}$ есть дизъюнктное семейство подмножеств множества M, то существует разбиение $\mathcal{B} = \{B_i\}_{i\in I}$ множества M такое, что каждый элемент дизъюнктного семейства $\mathcal{E}$ является подмножеством блока разбиения $\mathcal{B}$: $\forall\, i \in I\ (E_i \subset B_i)$.*

Доказательство. Выберем произвольный элемент $i_0 \in I$ и построим семейство $\mathcal{B} = \{B_i\}_{i\in I}$ следующим образом: $B_{i_0} := M \setminus \bigcup\limits_{i\in I - i_0} E_i$, $\forall\, i \in I - i_0\ \ (B_i := E_i)$. Семейство $\mathcal{B}$ по построению является дизъюнктным покрытием, то есть разбиением. Ясно, что $E_{i_0} \subset B_{i_0}$, а для остальных элементов требуемое включение имеет место по построению. □

Пример. Пусть $M := \{1, 2, 3\}$. Тогда элементы дизъюнктного семейства $\{\{1\}, \{2\}\}$ являются подмножествами блоков разбиения $\{\{1\}, \{2, 3\}\}$.

СЛЕДСТВИЕ. *Если семейство $\mathcal{E} = \{E_i\}_{i\in I}$ дизъюнктно, то* $\left|\bigcup\limits_{i\in I} E_i\right| = \sum\limits_{i\in I} |E_i|$.

1.2.8. Булеан

Множество всех подмножеств множества M называется *булеаном* множества M и обозначается 2^M, то есть $2^M \stackrel{\text{Def}}{=} \{A \mid A \subset M\}$.

ТЕОРЕМА 1. *Если множество M конечно, то $|2^M| = 2^{|M|}$.*

Доказательство. Индукция по $|M|$. База: если $|M| = 0$, то $M = \varnothing$ и $2^{\varnothing} = \{\varnothing\}$. Пусть $\forall\, M\ \ \big(|M| < k \Longrightarrow |2^M| = 2^{|M|}\big)$. Рассмотрим $M = \{a_1, \ldots, a_k\}$, $|M| = k$. Положим $\mathcal{M}_1 := \big\{X \in 2^M \mid a_k \notin X\big\}$ и $\mathcal{M}_2 := \big\{X \in 2^M \mid a_k \in X\big\}$.

Имеем $2^M = \mathcal{M}_1 \cup \mathcal{M}_2$, $\mathcal{M}_1 \cap \mathcal{M}_2 = \varnothing$, $\mathcal{M}_1 \sim \mathcal{M}_2$ за счёт взаимно-однозначного соответствия $X \mapsto X + a_k$ и $\mathcal{M}_1 \sim 2^{\{a_1,\ldots,a_{k-1}\}}$. По индукционному предположению $|2^{\{a_1,\ldots,a_{k-1}\}}| = 2^{k-1}$, и, значит, $|\mathcal{M}_1| = |\mathcal{M}_2| = 2^{k-1}$. Следовательно,
$|2^M| = |\mathcal{M}_1| + |\mathcal{M}_2| = 2^{k-1} + 2^{k-1} = 2^k$. □

Применение операций к подмножествам универсума не выводит за его пределы. Множество всех подмножеств множества U с операциями пересечения, объединения, разности и дополнения образует *алгебру подмножеств* множества U.

В булеане 2^M естественно выделяются семейства подмножеств $\mathcal{C}^k(M)$, состоящие из подмножеств множества M, имеющих одинаковую мощность k, то есть

$$\mathcal{C}^k(M) = \{S \subset M \mid k = |S|\}.$$

Такие семейства называются *семействами равномощных подмножеств*. Заметим, что семейства равномощных подмножеств, за исключением $\mathcal{C}^0(M)$, являются покрытиями множества M, а $\mathcal{C}^1(M)$ — семейство одноэлементных подмножеств множества M — является разбиением. Кроме того, семейства $\mathcal{C}^k(M)$ не пересекаются. Если $|M| = n$, то очевидно, что

$$2^M = \bigcup_{k=0}^{n} \mathcal{C}^k(M) \quad \text{и} \quad 2^n = \sum_{k=0}^{n} |\mathcal{C}^k(M)|.$$

ЗАМЕЧАНИЕ

В п. 5.1.5 показано, что мощность семейства равномощных подмножеств $\mathfrak{C}^k(M)$ является биномиальным коэффициентом, $|\mathfrak{C}^k(M)| = C(n,k)$.

Неравномощные множества можно сравнивать. Если множества A и B неравномощны, $|A| \neq |B|$, и при этом в множестве B существует собственное подмножество C, $C \subsetneq B$, такое что $|A| = |C|$, то говорят, что множество B *мощнее* множества A и записывают это так: $|A| < |B|$.

ТЕОРЕМА 2. *Булеан множества мощнее множества:* $|A| < |2^A|$.

Доказательство. От противного. Пусть A — некоторое множество и существует взаимно-однозначное соответствие $f\colon A \to 2^A$. Положим $Y := \{x \in A \mid x \notin f(x)\}$. Имеем $Y \in 2^A$, значит $\exists\, y \in A\ (f(y) = Y)$. Рассмотрим элемент y и множество Y. Если $y \in Y$, то $y \notin Y$, а если $y \notin Y$, то $y \in Y$ — противоречие. Следовательно, соответствие f не существует, а значит множество и его булеан неравномощны. Но множество A равномощно множеству одноэлементных подмножеств, которое является собственным подмножеством булеана. □

СЛЕДСТВИЕ. *Не существует множества всех множеств.*

ЗАМЕЧАНИЕ

Последнее утверждение известно как *парадокс Кантора*[1], основоположника теории множеств.

1.2.9. Свойства операций над множествами

Операции над множествами обладают целым рядом важных свойств, которые выражаются в форме равенств и имеют специальные названия:

1) *идемпотентность*: $A \cup A = A$, $A \cap A = A$;
2) *коммутативность*: $A \cup B = B \cup A$, $A \cap B = B \cap A$;
3) *ассоциативность*: $A \cup (B \cup C) = (A \cup B) \cup C$, $A \cap (B \cap C) = (A \cap B) \cap C$;
4) *дистрибутивность*: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$, $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
5) *поглощение*: $(A \cap B) \cup A = A$, $(A \cup B) \cap A = A$;
6) свойства нуля: $A \cup \varnothing = A$, $A \cap \varnothing = \varnothing$;
7) свойства единицы: $A \cup U = U$, $A \cap U = A$;
8) *инволютивность*: $\overline{\overline{A}} = A$;
9) *законы де Моргана* $\overline{A \cap B} = \overline{A} \cup \overline{B}$, $\overline{A \cup B} = \overline{A} \cap \overline{B}$;
10) свойства дополнения: $A \cup \overline{A} = U$, $A \cap \overline{A} = \varnothing$;
11) выражение для разности: $A \setminus B = A \cap \overline{B}$.

В справедливости перечисленных равенств можно убедиться различными способами. Например, можно нарисовать диаграммы Венна для левой и правой частей

[1] Георг Кантор (1845–1918)

равенства и убедиться, что они совпадают, или же провести формальное рассуждение для каждого равенства. Рассмотрим для примера первое равенство: $A \cup A = A$. Возьмем произвольный элемент x, принадлежащий левой части равенства, $x \in A \cup A$. По определению операции объединения $\cup$ имеем $x \in A \vee x \in A$. В любом случае $x \in A$. Взяв произвольный элемент из множества в левой части равенства, обнаружим, что он принадлежит множеству в правой части. Отсюда по определению включения множеств получаем, что $A \cup A \subset A$. Пусть теперь $x \in A$. Тогда, очевидно, верно $x \in A \vee x \in A$. Отсюда по определению операции объединения имеем $x \in A \cup A$. Таким образом, $A \subset A \cup A$. Следовательно, по определению равенства множеств, $A \cup A = A$. Это рассуждение можно записать короче:

$$x \in A \cup A \Longleftrightarrow x \in A \vee x \in A \Longleftrightarrow x \in A.$$

Аналогичные рассуждения нетрудно провести и для остальных равенств.

1.3. Представление множеств в программах

Термин «представление» применительно к программированию означает следующее. Представить в программе какой-либо объект (в данном случае множество) — это значит описать в терминах системы программирования структуру данных, используемую для хранения информации о представляемом объекте, и алгоритмы над выбранными структурами данных, которые реализуют присущие данному объекту операции. Применительно к множествам определение представления подразумевает описание способа хранения информации о принадлежности элементов множеству и описание алгоритмов для вычисления объединения, пересечения и других введённых операций. Следует подчеркнуть, что, как правило, один и тот же объект может быть представлен многими разными способами, причём нельзя указать способ, который является наилучшим для всех возможных случаев. В одних случаях выгодно использовать одно представление, а в других — другое. Выбор представления зависит от целого ряда факторов: особенностей представляемого объекта, набора операций, относительной частоты использования операций в конкретной задаче и т. д. Умение выбрать наилучшее для данного случая представление является основой искусства практического программирования. Хороший программист отличается тем, что он знает *много* разных способов представления и умело выбирает наиболее подходящий.

1.3.1. Битовые шкалы

Булевский массив **array** $[1..n]$ **of** $0..1$ называют *битовой шкалой* (или *двоичным вектором*, или *машинным словом*, или *двоичным кодом*) длины n. Элементы битовой шкалы называют *разрядами*, или *битами*. Значения разрядов обозначают цифрами 0 и 1 и считают, что к ним применимы групповые операции, перечисленные в таблице.

Название операции	Обозначение
конъюнкция, или *логическое умножение*	$\&$
дизъюнкция, или *логическое сложение*	$\vee$
сложение по модулю 2, или *исключающее или*	$+_2$
логическое отрицание, или *инвертирование*	$\neg$

ОТСТУПЛЕНИЕ

Исключающее или часто обозначают знаком $\oplus$. Однако в этой главе знак $\oplus$ используется для обозначения прямой суммы (см. п. 1.4.3) и ради однозначности обозначений мы пренебрегаем традицией в этом случае.

ЗАМЕЧАНИЕ

В большинстве компьютеров эти операции реализованы аппаратно как машинные команды, причём для некоторых характерных значений n (сейчас наиболее распространены случаи $n = 32$, $n = 64$) время выполнения поразрядных операций не зависит от длины машинного слова n (выполняется за один такт).

Пусть задано конечное множество $U = \{u_1, \ldots, u_n\}$. Подмножество A множества U представляется *кодом* (*машинным словом*, или *битовой шкалой*) C : **array** $[1..n]$ **of** 0..1, в котором: $\forall\, i \in 1..n\ \ (C[i] = u_i \in A)$. Тогда код пересечения множеств A и B есть поразрядная конъюнкция кода множества A и кода множества B, код объединения множеств A и B есть поразрядная дизъюнкция кода множества A и кода множества B, код симметрической разности множеств A и B есть поразрядное сложение по модулю 2 кода множества A и кода множества B, код дополнения множества A есть поразрядное инвертирование кода множества A.

Специальной машинной операции для разности множеств обычно не реализуют в компьютерах, а пользуются равенством $A \setminus B = A \cap \overline{B}$ (п. 1.2.9). Включение множеств легко проверить, используя тот факт, что $A \subset B \iff A = A \cap B$. Для проверки порязрядного равенства шкал в компьютерах также есть машинная команда. Таким образом, в этом представлении *все* операции над множествами выполняются весьма эффективно.

ЗАМЕЧАНИЕ

Если мощность множества превосходит размер машинного слова, но не очень велика, то для представления подмножеств используются массивы битовых шкал. В этом случае операции над множествами реализуются с помощью циклов по элементам массива.

Эту же идею можно использовать для представления мультимножеств. Если $\widehat{X} = [x_1^{a_1}, \ldots, x_n^{a_n}]$ — мультимножество над множеством $X = \{x_1, \ldots, x_n\}$, то массив A : **array** $[1..n]$ **of integer**, где $\forall\, i \in 1..n\ \ (A[i] = a_i)$, является представлением мультимножества $\widehat{X}$.

ЗАМЕЧАНИЕ

Представление индикатора совпадает с представлением его состава. Полезно сравнить это наблюдение с первым замечанием в п. 1.2.1.

ТЕОРЕМА. *Существует* 2^n *битовых шкал длины* n.

Доказательство. Рассмотрим множество M из n элементов и перенумеруем элементы. Тогда каждое подмножество множества M представляется единственным образом битовой шкалой, и каждая битовая шкала длины n является представлением определённого подмножества множества M. Таким образом, множество битовых шкал длины n и множество всех подмножеств n-элементного множества равномощны, откуда по теореме п. 1.2.8 следует требуемый результат. □

1.3.2. Представление натуральных чисел

ЛЕММА 1. $\forall k \in \mathbb{N}\ \left(\exists!\, m \in \mathbb{N}_0\ \left(2^m \leqslant k < 2^{m+1}\right)\right)$.

Доказательство. Положим $m := \lfloor \log_2 k \rfloor$. □

ЛЕММА 2. $\sum\limits_{i=0}^{m} 2^i = 2^{m+1} - 1$.

Доказательство. Сумма геометрической прогрессии. □

ТЕОРЕМА. *Для любого натурального числа k существует единственное подмножество $B(k) = \{m_1, \ldots, m_s\}$ множества всех целых неотрицательных степеней числа 2 такое, что данное число k равно сумме чисел подмножества $B(k)$, то есть $k = \sum\limits_{i=1}^{s} 2^{m_i}$.*

Доказательство. Индукция по числу m, которое существует для любого натурального k по лемме 1. При $m = 0$ имеем $k = 1 = 2^0$. Пусть для всех чисел $j < 2^m$ существует единственное множество $B(j) = \{m_1, \ldots, m_s\}$ такое, что $j = \sum_{i=1}^{s} 2^{m_i}$. Рассмотрим число k такое, что $2^m \leqslant k < 2^{m+1}$. Если $k = 2^m$, то одноэлементное множество $B(2^m) = \{m\}$ является требуемым. Если же число $k > 2^m$, то рассмотрим число $j := k - 2^m$. Имеем $j < 2^m$, по индукционному предположению $\exists!B(j) = \{m_1, \ldots, m_s\}$ $(j = \sum_{i=1}^{s} 2^{m_i})$. Но тогда добавим элемент m и получим требуемое множество $B(k) = \{m_1, \ldots, m_s, m\}$. □

СЛЕДСТВИЕ 1. *Любое натуральное число k такое, что $2^m \leqslant k < 2^{m+1}$, можно представить в виде линейной комбинации степеней числа 2: $k = \sum\limits_{i=0}^{m} a_i 2^i$, где коэффициенты a_i являются двоичными цифрами 0, 1, причём $a_m = 1$.*

Доказательство. Рассмотрим число k: $2^m \leqslant k < 2^{m+1}$. По теореме существует единственное множество $B(k) = \{m_1, \ldots, m_s\}$ такое, что $k = \sum_{i=1}^{s} 2^{m_i}$. Не ограничивая общности можно считать, что $m_1 < \ldots < m_s$. По лемме 2 $m_s = m$, иначе суммы степеней не достанет до числа k. Положим коэффициент $a_i := 1$, если $i \in B(k) = \{m_1, \ldots, m_s\}$, и положим $a_i := 0$, если $i \notin B(k)$. □

СЛЕДСТВИЕ 2. *Существует взаимно однозначное соответствие между битовыми шкалами длины n и целыми числами $k \in 0..(2^n - 1)$.*

Доказательство. Одно из возможных соответствий можно установить следующим образом. Поставим степени числа 2 в соответствие разрядам двоичной шкалы B : **array** $[1..n]$ **of** 0..1. Пусть $B[1]$ соответствует 2^{n-1}, $B[2]$ соответствует 2^{n-2}, и так далее, $B[n \breve{}1]$ соответствует 2^1, $B[n]$ соответствует 2^0. Сами значения разрядов шкалы будем считать коэффициентами линейной комбинации из следствия 1, причём $a_i = B[n - i]$. Тогда всякому неотрицательному числу соответствует линейная комбинация и битовая шкала, и обратно, всякой битовой шкале соответствует число из указанного диапазона. □

Указанный способ интерпретации двоичных шкал называется *двоичным представлением*, или *двоичным кодом* числа. При этом число 0 представляется кодом, состоящим из нулей; число 2^k представляется кодом, содержащим ровно одну единицу в $(k+1)$-м (считая справа) разряде; число 2^k-1 представляется кодом, состоящим из k единиц, считая справа. Следующий алгоритм строит двоичный код числа. В этом разделе двоичный код числа обозначается буквой B_n, причём, если длина кода n ясна из контекста, то нижний индекс опускается.

Алгоритм 1.2. Построение двоичного кода натурального числа.

Вход: натуральное число k: $0 \leqslant k \leqslant 2^n - 1$.
Выход: битовая шкала B : **array** $[1..n]$ **of** $0..1$ — двоичное представление числа k.

```
for i from n downto 1 do
    B[i] := k mod 2  //очередной бит
    k := k div 2  //оставшееся число
end for
```

Обоснование. Операция $k \bmod 2$ в любом случае дает 0 или 1, так что B действительно является битовой шкалой длины n.

Заметим, что $\forall\, a \in \mathbb{N}\ \ (a = 2(a \operatorname{div} 2) + (a \bmod 2))$. Таким образом, если вырезка первых $n-1$ разрядов из битовой шкалы B_{n-1} : **array** $[1..n-1]$ **of** $0..1$ является двоичным кодом числа $(k \operatorname{div} 2)$, то и вся шкала B_n является двоичным кодом числа k, и последний разряд определён правильно. Далее, применяя это рассуждение к числу $(k \operatorname{div} 2)$, к числу $((k \operatorname{div} 2) \operatorname{div} 2) = k \operatorname{div} 2^2$ и т. д., получаем, что все левые отрезки битовой шкалы B_{n-i} : **array** $[1..n-i]$ **of** $0..1$ являются двоичными кодами чисел $k \operatorname{div} 2^i$, а все правые отрезки битовой шкалы **array** $[(n-i+1)..n]$ **of** $0..1$ являются двоичными кодами чисел $k \bmod 2^i$. □

ЗАМЕЧАНИЕ

Обратный алгоритм вычисления числа по его двоичному коду и связанные с этим полезные формулы см. в п. 3.6.1.

Пример. Построение двоичного кода числа «десять».

i	k	$k \operatorname{div} 2$	$k \bmod 2$	$B[n-i+1..n]$	$\sum_{j=i}^{n} B[j]2^{n-j}$
4	10	5	0	$B[4..4] = \ \ 0$	0
3	5	2	1	$B[3..4] = \ \ 10$	2
2	2	1	0	$B[2..4] = \ 010$	2
1	1	0	1	$B[1..4] = 1010$	10

ОТСТУПЛЕНИЕ

В этом учебнике считается, что коэффициент a_i при 2^{n-i} хранится в разряде с индексом $n-i$, то есть в первом разряде хранится коэффициент при старшей степени, а в последнем разряде хранится коэффициент при младшей степени. Такой порядок называется «от старшего к младшему» (big-endian по-английски) и используется во многих случаях, например,

при обычной записи чисел в десятичной системе, в протоколе TCP/IP, в процессорах компаний IBM и Motorola. Однако применяются и другие порядки хранения коэффициентов. Так, в процессорах архитектуры X86 компании Intel применяется противоположный порядок — little-endian.

Аналогичным способом можно представлять не только натуральные, но и дробные числа, используя отрицательные степени $2^{-1} = \frac{1}{2}$, $2^{-2} = \frac{1}{4}$, $2^{-3} = \frac{1}{8}, \ldots$, в качестве значений разрядов дробной части числа. Для представления отрицательных чисел можно использовать дополнительный разряд знака и т. д. Важно осознать, что числа *возможно* представлять в программах битовыми шкалами, и такое представление оказывается вполне удовлетворительным с практической точки зрения.

ЗАМЕЧАНИЕ

Все утверждения и алгоритмы этого параграфа легко обобщаются на случай позиционной системы счисления с произвольным *основанием* $b > 1$.

ОТСТУПЛЕНИЕ

Известно множество принципиально различных представлений чисел в программах. *Представления чисел*, или *системы счисления*, то есть правила записи чисел с помощью системы знаков, которые в этом случае называют *цифрами*, принято делить на три класса. К первому классу относятся *позиционные системы счисления* с определённым основанием, в которых число представляется как линейная комбинация степеней основания (например: двоичная, восьмеричная, шестнадцатеричная, десятичная). Второй класс образуют *смешанные системы счисления*, которые подобны позиционным в том отношении, что число представляется линейной комбинацией, но вместо степеней основания используется иной возрастающий ряд чисел (например: принятое исчисление времени — секунды, минуты, часы; *факториальная система счисления*; *фибоначчиева система счисления*). Третий класс характерен тем, что значение цифры в числе определяется не позицией в записи, а иным способом (например: римская система счисления, *биномиальная система счисления*, система *остаточных классов*).

Двоичный код в программах является самым распространённым, но не потому, что этот код «самый лучший» — причины скорее исторические и технические, а не математические. Например, двоичная система кажется самой экономной с точки зрения расхода памяти, но это не так — троичная система экономичнее.

Действительно, пусть b — основание системы счисления, тогда для записи *всех* не более чем k-разрядных чисел в этой системе потребуется $s = kb$ знаков (экземпляров цифр). Например, для записи всех трёхразрядных десятичных чисел от 0 до 999 потребуется 30 знаков. *Экономичностью системы счисления* называется максимальное количество чисел N, которые возможно записать при заданном количестве знаков s. В системе с основанием b с помощью k разрядов можно записать $N = b^k = b^{\frac{s}{b}}$ чисел. Например, тридцатью знаками в десятичной системе можно записать $10^3 = 1000$ чисел, а в двоичной системе можно записать $2^{15} = 32768$ чисел. Двоичная система явно экономичнее. Средствами математического анализа нетрудно найти максимум выражения $b^{\frac{s}{b}}$ при заданном числе знаков s. Максимум достигается при $b = e$, где e — основание натурального логарифма. Число 3 ближе к числу e, чем число 2, поэтому троичная система несколько экономичнее двоичной.

Арифметические операции над двоичными числами кажутся достаточно эффективными. Сложение выполняется за время $O(n)$, а умножение за $O(n^2)$, если умножать «в столбик», и несколько быстрее, если применять современные изощрённые алгоритмы. Умножение труднее сложения, это кажется законом природы, но это не так — в системах остаточных классов обе эти операции выполняются за линейное время.

Вывод из этих наблюдений состоит в том, что в каждой математической модели для конкретной предметной области необходимо анализировать фактические требования к системе счисления и выбирать наиболее подходящую, а не довольствоваться той системой счисления, которая предоставлена системой программирования по умолчанию.

ЗАМЕЧАНИЕ

Используемый здесь и далее символ O в выражениях вида $f(n) = O(g(n))$ читается «О большое» и означает, что функция f асимптотически ограничена по сравнению с g (ещё говорят, что f «имеет тот же порядок малости», что и g):

$$f(n) = O(g(n)) \stackrel{\text{Def}}{=} \exists\, n_0, c > 0 \;\; (\forall\, n > n_0 \;\; (f(n) \leqslant cg(n)))\,.$$

Таким образом, между множеством всех битовых шкал длины n, множеством всех подмножеств множества $\{a_1, \ldots, a_n\}$ и множеством всех чисел из диапазона $0..(2^n - 1)$ установлены взаимно-однозначные соответствия, что позволяет представлять подмножества числами, а также представлять как числа, так и множества битовыми шкалами.

1.3.3. Перебор подмножеств заданного множества

Во многих переборных алгоритмах требуется последовательно рассмотреть все подмножества заданного n-элементного множества, то есть требуется рассмотреть все возможные битовые шкалы длины n. Пусть $B(i)$ — это функция, доставляющая двоичный код числа i. Тогда следующий алгоритм перечисляет все подмножества n-элементного множества.

Алгоритм 1.3. Генерация всех подмножеств n-элементного множества

Вход: число $n \geqslant 0$ — мощность множества.
Выход: последовательность кодов подмножеств $B(i)$.

```
for i from 0 to 2^n − 1 do
    yield B(i)  // код очередного подмножества
end for
```

Обоснование. Алгоритм выдаёт 2^n различных целых чисел, следовательно, 2^n различных кодов. Таким образом, все подмножества генерируются, причём ровно по одному разу. □

Недостаток этого алгоритма состоит в том, что порядок генерации подмножеств никак не связан с их составом. Например, вслед за подмножеством с кодом 0111, состоящим из трёх элементов, будет перечислено подмножество с кодом 1000, состоящее из одного элемента.

Рассмотрим $B(0), B(1), \ldots$ — последовательность подмножеств, которую строит предыдущий алгоритм. Введем функцию $Q(i)$, равную количеству разрядов, в которых коды $B(i)$ и $B(i-1)$ отличаются (в п. 6.3.3 показано, что эта функция является расстоянием на множестве битовых шкал длины n). На рис. 1.5 показано начало графика этой функции, для наглядности точки графика соединены линиями, а под значениями аргументов указаны соответствующие битовые шкалы.

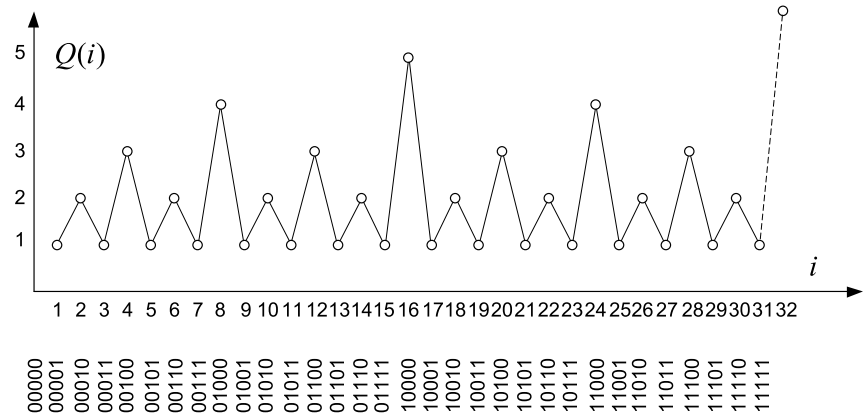


Рис. 1.5. График функции $Q(i)$

ТЕОРЕМА. *Для любого натурального числа $k < n$ в последовательности битовых шкал $B(0), \ldots, B(2^n - 1)$, k правых разрядов кода $B(i)$ при $i < 2^k$ совпадают с k правыми разрядами кода $B(2^k + i)$:*

$$\forall\, k \in \mathbb{N}\ \big(\forall\, i < 2^k\ \big(B(i)[n - i + 1..n] = B(2^k + i)[n - i + 1..n]\big)\big)\,.$$

Доказательство. Непосредственно следует из наблюдения, состоящего в том, что k правых разрядов двоичного кода числа n являются двоичным кодом числа $n \bmod 2^k$. □

СЛЕДСТВИЕ 1. $\forall\, k, m, i \in \mathbb{N}\ \big(i < 2^k \Longrightarrow B(i)[n-i+1..n] = B(m2^k+i)[n-i+1..n]\big)$.

Доказательство. $\forall\, m, i \in \mathbb{N}\ \big(i < 2^k \Longrightarrow i = (m2^k + i) \bmod 2^k\big)$. □

СЛЕДСТВИЕ 2. $\forall\, k \in \mathbb{N}\ \big(\forall\, m \in 0..(2^k - 1)\ \big(Q(m) = Q(2^k + m)\big)\big)$.

Доказательство. $\forall\, i < 2^k\ \big(B(i)[1..i] = B(2^k + i)[1..i]\big)$. □

СЛЕДСТВИЕ 3. $\forall\, k \in \mathbb{N}\ \big(\forall\, m \in 0..(2^k - 1)\ \big(Q(2^k + m) = Q(2^k - m)\big)\big)$.

Доказательство. Индукция по числу k. База: для $k = 1, 2, 3, \ldots$ (см. рис. 1.5). Индукционное предположение: пусть выполнено для $k - 1$, рассмотрим k. Пусть $m = 2^{k-1} + l$. Тогда $Q(2^k + m) = Q(m) = Q(2^{k-1} + l) = Q(2^{k-1} - l) =$
$= Q(2^k - (2^{k-1} + l)) = Q(2^k - m)$. □

СЛЕДСТВИЕ 4. $\forall\, i, j\ \ (i + j = 2^n - 1 \Longrightarrow B(i) = \neg B(j))$.

Доказательство. Индукция по числу n. Для $n = 1$ имеем: $B(0) = 0 = \neg 1 = \neg B(1)$. Для $n = 2$ имеем: $B(0) = 00 = \neg 11 = \neg B(3)$, $B(1) = 01 = \neg 10 = \neg B(2)$. Индукционное предположение: пусть следствие выполнено для n, рассмотрим $n + 1$. Пусть $i + j = 2^{n+1} - 1$ и для определенности $i < j$. Тогда $0 \leqslant i < 2^n \leqslant j \leqslant 2^{n+1} - 1$. Положим $k := j - 2^n$. Тогда $i + k = i + j - 2^n = 2^{n+1} - 1 - 2^n = 2^n - 1$. Рассмотрим правые n битов соответствующих кодов. По теореме имеем:
$B(k)[2..(n+1)] = B(2^n+k)[2..(n+1)] = B(j)[2..(n+1)]$, и по индукционному предположению $B(k)[2..(n + 1)] = \neg B(i)[2..(n + 1)]$. Но $B(i)[1] = 0$ и $B(j)[1] = 1$, откуда $B(i) = \neg B(j)$. □

СЛЕДСТВИЕ 5. *Значение функции $Q(m)$ равно количеству нулей на конце кода $B(m)$, увеличенному на единицу.*

Доказательство. Действительно, код нечётного числа имеет в последнем разряде 1 и именно в этом разряде отличается от кода предшествующего чётного числа. Коды чётных чисел имеют на конце столько нулей, какова степень числа 2, на которую данное число делится нацело. □

1.3.4. Алгоритм построения бинарного кода Грея

Бинарный код Грея[1] n-элементного множества — это последовательность всех подмножеств этого множества (то есть битовых шкал), в которой любые два соседних подмножества в последовательности, в том числе первое и последнее подмножество, различаются в точности одним элементом (разрядом).

Последовательности битовых шкал с указанным свойством существуют. Действительно, для $n = 1$ искомая последовательность кодов есть $0, 1$. Пусть известна искомая последовательность кодов $G_1, G_2, \ldots, G_{2^k}$ для $n = k$. Тогда построим из этой последовательности новую следующим образом: сначала выпишем все коды известной последовательности, прибавляя слева разряд 0, а затем эти же коды в обратном порядке, но прибавляя слева разряд 1. Имеем последовательность $0G_1, 0G_2, \ldots, 0G_{2^k}$, $1G_{2^k}, \ldots, 1G_2, 1G_1$. В этой последовательности 2^{k+1} кодов, они все различны, и любые два соседних, в том числе первый и последний, различаются ровно в одном разряде по построению.

Алгоритм 1.4. Построение бинарного кода Грея

Вход: число $n \geqslant 0$ — мощность множества.
Выход: последовательность кодов подмножеств B.

```
B : array [1..n] of 0..1  // битовая шкала
for i from 1 to n do B[i] := 0 end for  // инициализация
yield B  // пустое множество
for i from 1 to 2^n − 1 do
  p := Q(i)  // определение номера элемента
  B[p] := 1 − B[p]  // добавление или удаление элемента
  yield B  // очередное подмножество
end for
```

Функция Q, с помощью которой определяется номер разряда, подлежащего изменению, возвращает количество нулей на конце двоичной записи числа i, увеличенное на 1.

[1] Фрэнк Грей (1887–1969).

Алгоритм 1.5. Функция Q определения номера изменяемого разряда

Вход: i — номер подмножества.
Выход: номер изменяемого разряда.

```
q := 1; j := i
while j mod 2 = 0 do
    j := j div 2; q := q + 1
end while
return q
```

Обоснование. На нулевом шаге алгоритм выдаёт правильное подмножество B (пустое). Пусть за первые $2^k - 1$ шагов алгоритм выдал последовательность значений B. При этом изменения происходили только в правых k разрядах, а все левые $n-k$ разрядов оставались нулевыми: $B[1] = B[2] = \ldots = B[n-k] = 0$. На 2^k-м шаге разряд $B[n-k]$ изменяет своё значение с 0 на 1. После этого повторяется последовательность изменений значений $B[n-k+1..n]$ в обратном порядке, поскольку $Q(2^k + m) = Q(2^k - m)$ для $0 \leqslant m \leqslant 2^k - 1$ по следствию 3 предыдущей теоремы. □

ЗАМЕЧАНИЕ

Функция Q может быть вычислена достаточно эффективно. Действительно, результат операции $j \bmod 2$ — это младший разряд двоичного кода, который можно вычислить за один такт, например, взяв конъюнкцию кода числа j и кода числа 1. Результат операции $j \operatorname{div} 2$ — это сдвинутый вправо на один разряд код числа j. В большинстве компьютеров сдвиги кодов влево (умножение на 2) и вправо (деление на 2) также выполняются за один такт.

Пример. Протокол выполнения алгоритма для $n = 3$.

i	p	B		
		0	0	0
1	1	0	0	1
2	2	0	1	1
3	1	0	1	0
4	3	1	1	0
5	1	1	1	1
6	2	1	0	1
7	1	1	0	0

ЗАМЕЧАНИЕ

Построенная последовательность отнюдь не единственная последовательность, обладающая указанным свойством. Действительно, можно начать не с последовательности $0, 1$, а с последовательности $1, 0$, можно отражать не вправо, а влево, можно приписывать разряды не слева, а справа и т. д. Поэтому иногда построенный пример кода для точности называют *левым зеркальным кодом Грея*.

Вычислять коды Грея можно различными способами. В частности, левые зеркальные коды обладают свойством, позволяющим прямо вычислить код по его номеру. Обозначим $B(i)$ — двоичный код числа i, а $G(i)$ — i-й левый зеркальный код Грея.

Пример. Рассмотрим двоичные коды, результаты некоторых операций с кодами и зеркальный код Грея для $n = 2$.

i	$B(i)$	$B(i)\,\mathbf{div}\,2$	$B(i) +_2 (B(i)\,\mathbf{div}\,2)$	$G(i)$
0	00	00	00	00
1	01	00	01	01
2	10	01	11	11
3	11	01	10	10

ТЕОРЕМА. $G(i) = B(i) +_2 (B(i)\,\mathbf{div}\,2)$.

Доказательство. Индукция по n. Для $n = 1$ утверждение теоремы тривиально, для $n = 2$ проверено в предыдущем примере. Пусть утверждение теоремы верно для битовых шкал длины n, рассмотрим битовые шкалы длины $n + 1$. Рассмотрим $G_{n+1}(i)$. Возможны два случая.

[$0 \leqslant i < 2^n$] Заметим, что $B_{n+1}(i)[1] = 0$. По алгоритму построения кода Грея $G_{n+1}(i) = 0 \cdot G_n(i)$, где $\cdot$ — операция конкатенации. По индукционному предположению $G_n(i) = B_n(i) +_2 (B_n(i)\,\mathbf{div}\,2)$, откуда

$$\begin{aligned} G_{n+1}(i) &= 0 \cdot G_n(i) = 0 \cdot (B_n(i) +_2 (B_n(i)\,\mathbf{div}\,2)) = \\ &= 0 \cdot B_n(i) +_2 (0 \cdot B_n(i)\,\mathbf{div}\,2) = B_{n+1}(i) +_2 (B_{n+1}(i)\,\mathbf{div}\,2). \end{aligned}$$

[$2^n \leqslant i < 2^{n+1}$] Заметим, что $B_{n+1}(i)[1] = 1$. Положим $j := i - 2^n$ и рассмотрим $k := 2^{n+1} - i - 1 = 2^{n+1} - 2^n - j - 1 = 2^n - j - 1$. Тогда $G_{n+1}(i) = 1 \cdot G_n(k)$. При этом $j + k = 2^n - 1$ и выполняется условие следствия 4. Заметим также, что для любых битовых шкал A и B выполняется равенство $A +_2 B = \neg A +_2 \neg B$. По индукционному предположению $G_n(k) = B_n(k) +_2 (B_n(k)\,\mathbf{div}\,2)$, получаем:

$$\begin{aligned} G_{n+1}(i) &= 1 \cdot G_n(k) = 1 \cdot (B_n(k) +_2 (B_n(k)\,\mathbf{div}\,2)) = \\ &= 1 \cdot (\neg B_n(j) +_2 (\neg B_n(j)\,\mathbf{div}\,2)) = 1 \cdot (B_n(j) +_2 (B_n(j)\,\mathbf{div}\,2)) = \\ &= 1 \cdot (B_n(i) +_2 (B_n(i)\,\mathbf{div}\,2)) = B_{n+1}(i) +_2 (B_{n+1}(i)\,\mathbf{div}\,2). \end{aligned}$$

□

Операции с битовыми шкалами можно выполнять не только параллельно, сразу для всех разрядов, но и последовательно, вычисляя биты один за другим и используя полученные значения при вычислении следующих битов. Такие вычисления называются *рекуррентными*. Иногда это бывает весьма полезно.

Пример. Алгоритм сложения натуральных чисел X и Y, представленных битовыми шкалами длины n.

```
p := 0  // разряд переноса
for i from n downto 1 do
  q := X[i] + Y[i] + p  // сумма цифр с учётом переноса
  Z[i] := q mod 2  // очередная цифра суммы
  p := q div 2  // перенос
end for
```

Заметим, что в этом алгоритме не учтено возможное переполнение разрядной сетки при сложении.

Применяя идею рекуррентного вычисления битов, легко вывести следующие следствия из основной теоремы данного параграфа.

СЛЕДСТВИЕ 1. $\forall 1 < i \leqslant n\ (G[i] = B[i] +_2 B[i-1]), G[1] = B[1].$

Доказательство. $\forall k\ ((B(k)\ \mathbf{div}\ 2)[1] = 0\ \&\ (B(k)\ \mathbf{div}\ 2)[i] = B(k)[i-1]).$ □

СЛЕДСТВИЕ 2. $\forall 1 < i \leqslant n\ (B[i] = G[i] +_2 B[i-1]), G[1] = B[1].$

Доказательство. $G[i] +_2 B[i-1] = B[i] +_2 B[i-1] +_2 B[i-1] = B[i].$ □

СЛЕДСТВИЕ 3. $\forall i\ \left(1 < i \leqslant n \Longrightarrow B[i] = \sum_{j=1}^{i} G[j]\right).$

Доказательство. Подставляя в рекуррентное соотношение следствия 2, имеем $B[i] = G[i] +_2 B[i-1] = G[i] + G[i-1] +_2 B[i-2]$ и т. д. Учитывая, что $B[0] = 0$ по определению (это возникающий слева «ниоткуда» разряд при сдвиге кода вправо), получаем требуемое. □

1.3.5. Представление множеств упорядоченными списками

Если универсум очень велик (или бесконечен), а рассматриваемые подмножества универсума не очень велики, то представление с помощью битовых шкал не является эффективным с точки зрения экономии памяти. В этом случае множества обычно представляются *списками элементов*. Элемент списка при этом представляется записью с двумя полями: информационным полем i info и указателем n на следующий элемент. Весь список задаётся указателем на первый элемент.

elem = **struct {** i: info; n: $\uparrow$ elem **}**

При таком представлении трудоёмкость операции $\in$ составит $O(n)$, а трудоёмкость операций $\subset$, $\cap$, $\cup$ составит $O(nm)$, где n и m — мощности участвующих в операции множеств.

Если элементы в списках упорядочить, например, по возрастанию значения поля i, то трудоёмкость всех операций составит $O(n+m)$. Эффективная реализация операций над множествами, представленными в виде упорядоченных списков, основана на весьма общем алгоритме, известном как *алгоритм слияния*. Алгоритм слияния параллельно просматривает два множества, представленных упорядоченными списками, причём на каждом шаге продвижение происходит в том множестве, в котором текущий элемент меньше.

1.3.6. Проверка включения слиянием

Рассмотрим алгоритм слияния, который определяет, является ли множество A подмножеством множества B.

Обоснование. На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае текущий элемент множества A заведомо меньше, чем текущий и все последующие элементы множества B, а потому он не содержится в множестве B и можно завершить выполнение алгоритма. Во втором случае происходит продвижение по множеству B в надежде отыскать элемент, совпадающий с текущим элементом множества A. В третьем случае найдены совпадающие элементы и происходит продвижение сразу в обоих множествах. По завершении основного цикла возможны два варианта: либо $pa =$ **nil**, либо $pa \neq$ **nil**. Первый означает, что для всех элементов множества A удалось найти совпадающие элементы в множестве B. Второй — что множество B закончилось раньше, то есть не для всех элементов множества A удалось найти совпадающие элементы в множестве B. □

Алгоритм 1.6. Проверка включения слиянием

Вход: проверяемые множества A и B, которые заданы указателями a и b.
Выход: **true**, если $A \subset B$, в противном случае **false**.

```
pa := a; pb := b  // инициализация
while pa ≠ nil & pb ≠ nil do
  if pa.i < pb.i then
    return false  // элемент A отсутствует в B
  else if pa.i > pb.i then
    pb := pb.n  // элемент A, может быть, присутствует в B
  else
    pa := pa.n  // здесь pa.i = pb.i, то есть
    pb := pb.n  // элемент A точно присутствует в B
  end if
end while
return pa = nil  // true, если A исчерпано, false — в противном случае
```

1.3.7. Вычисление объединения слиянием

Рассмотрим алгоритм слияния, который вычисляет объединение двух множеств, представленных упорядоченными списками.

Алгоритм 1.7. Вычисление объединения слиянием

Вход: объединяемые множества A и B, заданные указателями a и b.
Выход: объединение $C = A \cup B$, заданное указателем c.

```
pa := a; pb := b; c := nil; e := nil  // инициализация
while pa ≠ nil & pb ≠ nil do
  if pa.i < pb.i then
    d := pa.i; pa := pa.n  // добавляем элемент из A
  else if pa.i > pb.i then
    d := pb.i; pb := pb.n  // добавляем элемент из B
  else
    d := pa.i  // здесь pa.i = pb.i, и можно взять любой
    pa := pa.n; pb := pb.n  // продвижение в обоих
  end if
  Append(c, e, d)  // добавление элемента d в конец списка c
end while
p :=nil  // указатель «хвоста»
if pa ≠ nil then
  p := pa  // нужно добавить в результат оставшиеся элементы A
end if
if pb ≠ nil then
  p := pb  // нужно добавить в результат оставшиеся элементы B
end if
while p ≠ nil do
  Append(c, e, p.i)  // добавление элемента pa.i в конец списка c
  p := p.n  // продвижение указателя «хвоста»
end while
```

Обоснование. На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае в результирующий список добавляется текущий элемент множества A и происходит продвижение в этом множестве, во втором — аналогичная операция производится с множеством B, а в третьем случае найдены совпадающие элементы и происходит продвижение сразу в обоих множествах. Таким образом, в результат попадают все элементы обоих множеств, причём совпадающие элементы попадают ровно один раз. По завершении основного цикла один из указателей, pa и pb (но не оба вместе!), может быть не равен **nil**. В этом случае остаток соответствующего множества без проверки добавляется в результат. □

Вспомогательная процедура Append(c, e, d) присоединяет элемент d к хвосту e списка c.

Алгоритм 1.8. Процедура Append — присоединение элемента в конец списка

Вход: указатель c на первый элемент списка, указатель e на последний элемент списка, добавляемый элемент d.

Выход: указатель c на первый элемент списка, указатель e на последний элемент списка.

```
q := new(elem); q.i := d; q.n := nil  // новый элемент списка
if c = nil then
  c := q  // пустой список
else
  e.n := q  // непустой список
end if
e := q
```

Обоснование. До первого вызова функции Append имеем: $c =$ **nil** и $e =$ **nil**. После первого вызова указатель c указывает на первый элемент списка, а указатель e — на последний элемент. Если указатели c и e не пустые, то после очередного вызова функции Append это свойство сохраняется. □

1.3.8. Вычисление пересечения слиянием

Рассмотрим алгоритм слияния, который вычисляет пересечение двух множеств, представленных упорядоченными списками (см. с. 49).

Обоснование. На каждом шаге основного цикла возможна одна из трёх ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае текущий элемент множества A не принадлежит пересечению, он пропускается, и происходит продвижение в этом множестве. Во втором случае то же самое производится с множеством B. В третьем случае найдены совпадающие элементы, один экземпляр элемента добавляется в результат, и происходит продвижение сразу в обоих множествах. Таким образом, в результат попадают все совпадающие элементы обоих множеств, причём ровно один раз. □

1.3.9. Представление множеств итераторами

Если структуры данных предполагаются известными и используются при программировании операций, то:

Алгоритм 1.9. Вычисление пересечения слиянием

Вход: пересекаемые множества A и B, заданные указателями a и b.
Выход: пересечение $C = A \cap B$, заданное указателем c.

```
pa := a;  pb := b;  c := nil;  e := nil  // инициализация
while pa ≠ nil & pb ≠ nil do
  if pa.i < pb.i then
    pa := pa.n  // элемент множества A не принадлежит пересечению
  else if pa.i > pb.i then
    pb := pb.n  // элемент множества B не принадлежит пересечению
  else
    // здесь pa.i = pb.i — данный элемент принадлежит пересечению
    Append(c, e, pa.i); pa := pa.n; pb := pb.n  // добавление элемента
  end if
end while
```

1) возникают затруднения при совместном использовании нескольких *альтернативных* представлений для одного типа объектов;
2) необходимо заранее определять набор операций, которым открыт доступ к внутренней структуре данных объектов (такие операции часто называют *первичными*).

ОТСТУПЛЕНИЕ

Парадигма объекто-ориентированного программирования предполагает совместное согласованное определение структур данных и процедур доступа к ним. Другими словами, в объекто-ориентированном программировании первичные операции класса объектов заранее фиксированы в форме *методов*, а представление данных для объектов класса единственно и фиксировано в форме *свойств*. Парадигма объекто-ориентированного программирования — весьма полезная, но отнюдь не универсальная парадигма программирования.

Для множеств понятие принадлежности является первичным и не может быть запрограммировано без знания структуры данных для хранения множеств. Остальные операции над множествами можно запрограммировать независимо от представления множеств. Для представления множества достаточно *итератора* — процедуры, которая перебирает элементы множества и делает с ними то, что нужно.

Пример. Итератор для множества, представленного списком, где p — указатель на первый элемент.

```
while p ≠ nil do
  yield p.i  // очередной элемент
  p := p.n  // продвижение указателя
end while
```

Наличие итератора X позволяет написать цикл

```
for x ∈ X do
  S(x)  // где S — любая процедура, зависящая от x
end for
```

и этот цикл означает применение оператора S ко всем элементам множества X по одному разу в некотором неопределённом порядке.

ЗАМЕЧАНИЕ

В доказательствах цикл **for** $x \in X$ **do** $S(x)$ **end for** используется и в том случае, когда $|X| = \infty$. Это не означает реального процесса исполнения тела цикла в дискретные моменты времени, а означает всего лишь мысленную возможность применить оператор S ко всем элементам множества X независимо от его мощности.

В таких обозначениях итератор пересечения множеств $X \cap Y$ может быть задан алгоритмом 1.10.

Алгоритм 1.10. Итератор пересечения множеств

```
for x ∈ X do
  for y ∈ Y do
    if x = y then
      S(x)  // тело цикла
      next for x  // следующий x
    end if
  end for
end for
```

ЗАМЕЧАНИЕ

Оператор **next for** x — это *оператор структурного перехода*, выполнение которого в данном случае означает прерывание выполнения цикла по y и переход к следующему шагу в цикле по x.

Аналогично итератор разности множеств $X \setminus Y$: может быть задан алгоритмом 1.11.

Алгоритм 1.11. Итератор разности множеств

```
for x ∈ X do
  for y ∈ Y do
    if x = y then
      next for x  // следующий x
    end if
  end for
  S(x)  // тело цикла
end for
```

Заметим, что $A \cup B = (A \setminus B) \cup B$ и $(A \setminus B) \cap B = \varnothing$. Поэтому достаточно рассмотреть итератор объединения дизъюнктных множеств $X \cup Y$, где $X \cap Y = \varnothing$.

Алгоритм 1.12. Итератор объединения дизъюнктных множеств

```
for x ∈ X do
  S(x)  // тело цикла
end for
for y ∈ Y do
  S(y)  // тело цикла
end for
```

Стоит обратить внимание на то, что итератор пересечения реализуется вложенными циклами, а итератор дизъюнктного объединения — последовательностью циклов.

Сложность алгоритмов для операций с множествами при использовании представления, не зависящего от реализации, составляет в худшем случае $O(nm)$, где m и n — мощности участвующих множеств, в то время как для представления упорядоченными списками сложность в худшем случае равна $O(n+m)$.

ЗАМЕЧАНИЕ

На практике почти всегда оказывается так, что самая изощрённая реализация операции, не зависящая от представления, оказывается менее эффективной по сравнению с самой прямолинейной реализацией, ориентированной на конкретное представление.

ОТСТУПЛЕНИЕ

При современном состоянии аппаратной базы вычислительной техники оказывается, что во многих задачах можно пренебречь некоторым выигрышем в эффективности, например различием между $O(nm)$ и $O(n+m)$. Другими словами, нынешние компьютеры настолько быстры, что часто можно не гнаться за самым эффективным представлением, ограничившись «достаточно эффективным».

Пример. В развитом пакете программ для решения задач линейной алгебры используются матрицы различных видов (разреженные, треугольные, блочно-диагональные и т. д.), причем виды матриц настолько отличаются, что для их представления удобно использовать *различные* структуры данных (одномерные и двумерные массивы, списки ненулевых элементов и т. д.). В таком случае нерационально программировать для каждой структуры данных свои функции, реализующие операции с матрицами. Тогда можно реализовать для каждой структуры данных первичные операции (скажем, чтение и запись значения элемента матрицы) и дальше программировать требуемые функции, которые уже будут работать с первичными операциями, независимо от структур данных, выбранных для представления матриц.

1.4. Отношения

Обычное широко используемое понятие «отношение» имеет вполне точное математическое значение, которое вводится в этом разделе.

Формально отношения являются множествами. Поэтому главу можно было бы назвать «Множества, в том числе отношения». Однако с помощью отношений вводят в рассмотрение целый ряд понятий и обозначений, которые находят, может быть, даже больше применений, чем рассмотренные операции с множествами. Поэтому понятие «отношение» вполне правомерно поставить в один ряд с понятием «множество».

1.4.1. Упорядоченные пары и наборы

Если a и b — объекты, то через (a,b) обозначим *упорядоченную пару*. *Равенство упорядоченных пар* определяется следующим образом:

$$(a,b) = (c,d) \stackrel{\text{Def}}{=} a = c \ \& \ b = d.$$

Вообще говоря, $(a,b) \neq (b,a)$.

ЗАМЕЧАНИЕ

Формально упорядоченные пары можно определить как множества, если определить их так:

$$(a,b) \stackrel{\text{Def}}{=} \{a, \{a,b\}\}.$$

Таким образом, введённое понятие упорядоченной пары не выводит рассмотрение за пределы теории множеств.

Аналогично упорядоченным парам можно рассматривать упорядоченные тройки, четвёрки и т. д. В общем случае подобные объекты называются *n-ками*, *кортежами*, *наборами* или (конечными) *последовательностями*. Упорядоченный набор из n элементов обозначается $(a_1, \ldots, a_n)$.

ЗАМЕЧАНИЕ

Обычно элементы последовательности считают однотипными, то есть принадлежащими одному множеству, хотя, вообще говоря, это не обязательно.

Набор $(a_1, \ldots, a_n)$ можно определить рекурсивно, используя понятие упорядоченной пары: $(a_1, \ldots, a_n) \stackrel{\text{Def}}{=} ((a_1, \ldots, a_{n-1}), a_n)$. Количество элементов в наборе называется его *длиной* и обозначается следующим образом: $|(a_1, \ldots, a_n)|$.

ТЕОРЕМА. *Два набора одной длины равны, если равны их соответствующие элементы:* $\forall\, n \left((a_1, \ldots, a_n) = (b_1, \ldots, b_n) \iff \bigwedge\limits_{i=1}^{n} a_i = b_i \right)$.

Отсюда следует, что порядок «отщепления» элементов в рекурсивном определении кортежа не имеет значения.

Доказательство. Индукция по n. База: при $n = 2$ по определению равенства упорядоченных пар. Пусть теперь $(a_1, \ldots, a_{n-1}) = (b_1, \ldots, b_{n-1}) \iff \bigwedge\limits_{i=1}^{n-1} a_i = b_i$. Тогда

$$\begin{aligned}(a_1, \ldots, a_n) = (b_1, \ldots, b_n) &\iff ((a_1, \ldots, a_{n-1}), a_n) = ((b_1, \ldots, b_{n-1}), b_n) \iff \\ &\iff (a_1, \ldots, a_{n-1}) = (b_1, \ldots, b_{n-1}) \;\&\; a_n = b_n \iff \\ &\iff \left(\bigwedge_{i=1}^{n-1} a_i = b_i \right) \;\&\; (a_n = b_n) = \bigwedge_{i=1}^{n} a_i = b_i.\end{aligned}$$

□

ЗАМЕЧАНИЕ

Наиболее естественным представлением в программе n-ки с элементами из множества A является массив **array** $[1..n]$ **of** A.

1.4.2. Прямое произведение множеств

Пусть A и B — два множества. *Прямым (декартовым) произведением* двух множеств A и B называется множество всех упорядоченных пар, в которых первый элемент принадлежит A, а второй принадлежит B: $A \times B \stackrel{\text{Def}}{=} \{(a, b) \mid a \in A \;\&\; b \in B\}$.

ЗАМЕЧАНИЕ

Точка на плоскости может быть задана упорядоченной парой координат, то есть двумя точками на координатных осях. Таким образом, $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$. Своим появлением метод координат обязан Декарту[1], отсюда и название «декартово произведение».

[1] Рене Декарт (1596–1650).

ТЕОРЕМА. *Если множества A и B конечны, то* $|A \times B| = |A|\,|B|$.

Доказательство. Первый компонент упорядоченной пары можно выбрать $|A|$ способами, второй — $|B|$ способами, причём независимо от выбора первого элемента. Таким образом, всего имеется $|A||B|$ различных упорядоченных пар. □

Понятие прямого произведения допускает обобщение.

$$A_1 \times \cdots \times A_n \stackrel{\text{Def}}{=} \{(a_1, \ldots, a_n) \mid a_1 \in A_1 \;\&\; \ldots \;\&\; a_n \in A_n\}.$$

Множества A_i не обязательно различны.

СЛЕДСТВИЕ. $|A_1 \times \cdots \times A_n| = \prod_{i=1}^{n} |A_i|$.

Прямое произведение нескольких множеств ассоциативно, как показано в следующей лемме.

ЛЕММА. $(A \times B) \times C \sim A \times (B \times C)$.

Доказательство. $(A \times B) \times C = \{((a,b)c) \mid (a,b) \in A \times B \;\&\; c \in C\} \sim$
$\sim \{(a,b,c) \mid a \in A \;\&\; b \in B \;\&\; c \in C\} \sim \{(a,(b,c)) \mid a \in A \;\&\; (b,c) \in B \times C\} =$
$= A \times (B \times C)$. □

Степенью множества A называется его n-кратное прямое произведение самого на себя. Обозначение: $A^n \stackrel{\text{Def}}{=} \underbrace{A \times \cdots \times A}_{n \text{ раз}}$. Соответственно, $A^1 \stackrel{\text{Def}}{=} A$, $A^2 \stackrel{\text{Def}}{=} A \times A$ и вообще $A^n \stackrel{\text{Def}}{=} A \times A^{n-1}$.

СЛЕДСТВИЕ. $|A^n| = |A|^n$.

1.4.3. Размеченное объединение множеств

Пусть $X_1, \ldots, X_n$ — семейство множеств, возможно имеющих общие элементы. *Размеченным объединением* (употребляются также термины *дизъюнктное объединение множеств* и *прямая сумма множеств*) n множеств $X_1, \ldots, X_n$ называется множество всех упорядоченных пар, в которых первый элемент принадлежит множеству X_i, а второй элемент — натуральное число i:

$$X_1 \oplus X_2 \oplus \ldots \oplus X_n \stackrel{\text{Def}}{=} \{(x,i) \mid 1 \leqslant i \leqslant n \;\&\; x \in X_i\}.$$

Операция размеченного объединения приписывает каждому элементу нового множества индекс, по которому можно понять, из какого конкретно множества элемент попал в результат, в отличие от операции объединения, в которой эта информация теряется.

Пример. Пусть X — множество задач, а $X_i \subset X$ — подмножество задач, решённых студентом i. Тогда $X_1 \oplus \ldots \oplus X_n$ — множество задач, решённых n студентами.

ЗАМЕЧАНИЕ

Размеченное объединение множеств $X_1, \ldots, X_n$ есть объединение декартовых произведений $X_i \times \{i\}$: $X_1 \oplus X_2 \oplus \ldots \oplus X_n = \bigcup_{i=1}^{n} (X_i \times \{i\})$. Эта операция ассоциативна, поэтому скобки в записи $X_1 \oplus X_2 \oplus \ldots \oplus X_n$ опускаются.

ТЕОРЕМА. *Для конечных множеств* $|X_1 \oplus X_2 \oplus \cdots \oplus X_n| = \sum_{i=1}^{n} |X_i|$.

Доказательство. Множества пар $X_i \times \{i\}$ с различными метками i не имеют общих элементов, значит: $\left|\bigcup_i X_i \times \{i\}\right| = \sum_i |X_i \times \{i\}| = \sum_i |X_i|\ |\{i\}| = \sum_i |X_i|$. Учитывая замечание, получаем: $|X_1 \oplus X_2 \oplus \cdots \oplus X_n| = \sum_{i=1}^{n} |X_i|$. □

Пример. Пусть S и U — два множества двоичных последовательностей длины n. Ясно, что они могут иметь неотличимые элементы. Пусть для S задано отображение в целые числа, которое трактует элемент S как двоичное представление числа со знаком; для U задано отображение в натуральные числа, трактующее его элемент как двоичное представление числа без знака. Тогда для размеченного объединение множеств $X = S \oplus U$ можно построить отображение в целые числа, которое в зависимости от метки элемента будет переводить его в целое число по соответствующему правилу.

ОТСТУПЛЕНИЕ

В программировании операция размеченного объединения поддерживается структурой данных, которая обычно также называется *размеченное объединение*, используемой для хранения значения одного из нескольких типов. Текущим типом является тип значения, записанного последним. В языках семейства Си объединение типов задается ключевым словом **union**. При этом информация о текущем типе нигде не сохраняется, и за безошибочное обращение со значением полностью отвечает программист. Размеченное объединение **record case** в языке Паскаль имеет специальное поле (*тег*), хранящее тип текущего значения. Анализ тега позволяет программисту избегать ошибок. На первый взгляд то же поведение можно обеспечить проще. А именно, можно взять тип прямого произведения (**struct** в языках семейства Си), завести в нем поля необходимых типов и поле тега. Преимуществом размеченного объединения (прямой суммы) **record case** перед типом структуры (прямого произведения) **struct** является выигрыш по памяти. В случае размеченного объединения достаточно выделить память под значение наиболее объёмного по памяти типа и под поле тега, а в случае прямого произведения необходимо выделить память под значения всех типов в сумме и под поле тега.

1.4.4. Бинарные отношения

Бинарным отношением между множествами A и B называется тройка $\langle A, B, R\rangle$, где R — подмножество прямого произведения A и B, то есть $R \subset A \times B$. Множество R называется *графиком отношения*, A называется *областью отправления*, а B — *областью прибытия*. Если множества A и B определены контекстом, то часто просто говорят, что задано отношение R. При этом для краткости отношение обозначают тем же символом, что и график отношения.

ЗАМЕЧАНИЕ

Принято говорить «отношение между множествами», хотя порядок множеств имеет значение и отношение между A и B совсем не то же самое, что отношение между B и A. Поэтому иногда, чтобы подчеркнуть это обстоятельство, употребляют оборот «отношение R из множества A в множество B».

Среди элементов множеств A и B могут быть такие, которые входят в одну или несколько пар, определяющих отношение R, а могут быть такие, которые не входят ни в одну из пар, определяющих отношение R.

Множество элементов области отправления, которые присутствуют в парах отношения, называется *областью определения* отношения R и обозначается $\mathrm{Dom}\, R$, а множество элементов области прибытия, которые присутствуют в парах отношения, называется *областью значений* отношения R и обозначается $\mathrm{Im}\, R$:

$\mathrm{Dom}\, R \stackrel{\mathrm{Def}}{=} \{a \in A \mid \exists\, b \in B\ \ ((a,b) \in R)\}$, $\mathrm{Im}\, R \stackrel{\mathrm{Def}}{=} \{b \in B \mid \exists\, a \in A\ \ ((a,b) \in R)\}$.

Если $A = B$ (то есть $R \subset A^2$), то говорят, что R есть *отношение на множестве* A.

Для бинарных отношений обычно используется *инфиксная форма записи*:

$$aRb \stackrel{\mathrm{Def}}{=} (a,b) \in R \subset A \times B.$$

Инфиксная форма позволяет более кратко записывать некоторые формы утверждений относительно отношений: $aRbRc \stackrel{\mathrm{Def}}{=} (a,b) \in R\ \&\ (b,c) \in R$.

Пример. Пусть задано множество U. Тогда $\in$ (принадлежность) — отношение между элементами множества U и элементами булеана 2^U, а $\subset$ (включение) и $=$ (равенство) — отношения на 2^U. Хорошо известны отношения $=, <, \leqslant, >, \geqslant, \neq$, определённые на множестве чисел.

Пусть R — отношение между множествами A и B: $R \subset A \times B$. Введём следующие понятия:

Обратное отношение: $R^{-1} \stackrel{\mathrm{Def}}{=} \{(b,a) \mid (a,b) \in R\} \subset B \times A$.

Дополнение отношения: $\overline{R} \stackrel{\mathrm{Def}}{=} \{(a,b) \mid (a,b) \notin R\} \subset A \times B$.

Универсальное отношение: $U \stackrel{\mathrm{Def}}{=} \{(a,b) \mid a \in A\,\&\, b \in B\} = A \times B$.

Пример. На рис. 1.6 показаны графики отношения $R = \{(x,y) \mid 4 > x - y > 1\}$, $x \in 1..5$, $y \in 1..4$ и обратного отношения $R^{-1} = \{(x,y) \mid 4 > y - x > 1\}$, $x \in 1..4$, $y \in 1..5$.

На любом множестве можно определить *тождественное* отношение:

$I \stackrel{\mathrm{Def}}{=} \{(a,a) \mid a \in A\} \subset A^2$. Обычно для обозначения тождественного отношения используют знак равенства $=$.

Примеры

Равенство множеств и равенство чисел являются тождественными отношениями.

ЗАМЕЧАНИЕ

График тождественного отношения на множестве A иногда называют *диагональю* в $A \times A$.

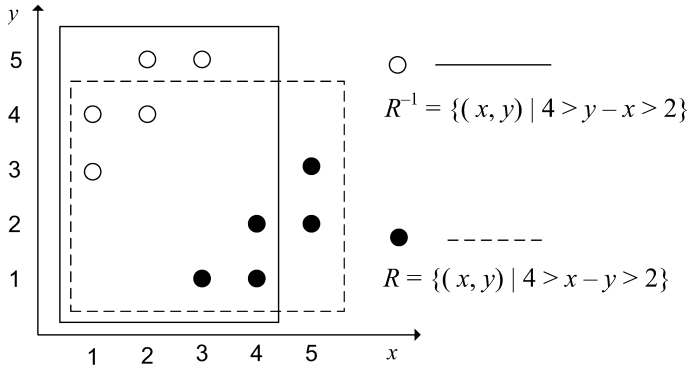


Рис. 1.6. Графики отношений R и R^{-1}

Обобщённое понятие отношения: *n-местное* (*n-арное*) отношение R — это подмножество прямого произведения n множеств, то есть множество *кортежей*:

$$R \subset A_1 \times \cdots \times A_n \stackrel{\text{Def}}{=} \{(a_1, \ldots, a_n) \mid a_1 \in A_1 \ \& \ \ldots \ \& \ a_n \in A_n\}.$$

Обычно вместо $(a_1, \ldots, a_n) \in R$ пишут $R(a_1, \ldots, a_n)$.

ЗАМЕЧАНИЕ

Число n, то есть длину кортежей отношения, называют иногда *вместимостью*.

ОТСТУПЛЕНИЕ

Многоместные отношения используются, например, в теории баз данных. Само название «реляционная» база данных происходит от слова relation (отношение).

Далее рассматриваются только двуместные (бинарные) отношения, при этом слово «бинарные» опускается.

1.4.5. Композиция отношений

Пусть $R_1 \subset A \times C$ — отношение между A и C, а $R_2 \subset C \times B$ — отношение между C и B. *Композицией* двух отношений R_1 и R_2 называется отношение $R \subset A \times B$ между A и B, определяемое следующим образом:

$$R \stackrel{\text{Def}}{=} R_1 \circ R_2 \stackrel{\text{Def}}{=} \{(a, b) \mid a \in A \ \& \ b \in B \ \& \ \exists c \in C \ \ (aR_1c \ \& \ cR_2b)\}.$$

Другими словами, $a \ R_1 \circ R_2 \ b \iff \exists c \in C \ \ (aR_1c \ \& \ cR_2b)$.

Пример. На рис. 1.7 показана геометрическая иллюстрация композиции отношений.

ТЕОРЕМА. *Композиция отношений ассоциативна, то есть*
$\forall R_1 \subset A \times B, R_2 \subset B \times C, R_3 \subset C \times D \ ((R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3))$.

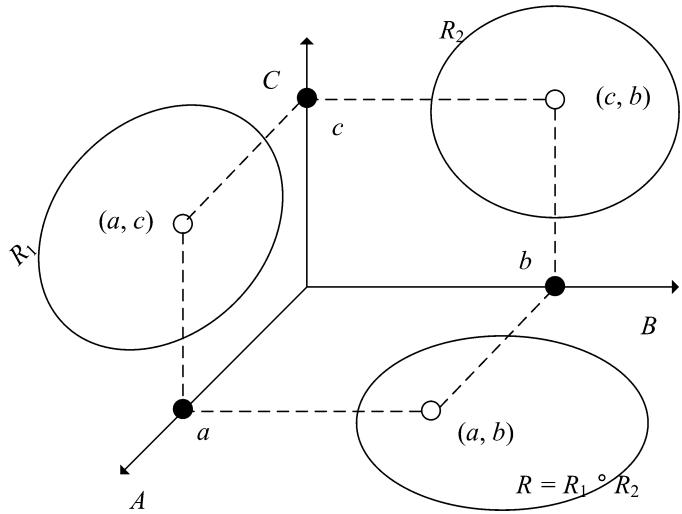


Рис. 1.7. Геометрическая иллюстрация композиции отношений

Доказательство. $a(R_1 \circ R_2) \circ R_3 d \iff \exists\, c \in C\ \ (aR_1 \circ R_2 c\ \&\ cR_3 d) \iff$
$\iff \exists\, c \in C\ \ ((\exists\, b \in B\ \ (aR_1 b\ \&\ bR_2 c))\ \&\ cR_3 d) \iff$
$\iff \exists\, b \in B, c \in C\ \ (aR_1 b\ \&\ bR_2 c\ \&\ cR_3 d) \iff$
$\iff \exists\, b \in B\ \ (aR_1 b\ \&\ (\exists\, c \in C\ \ (bR_2 c\ \&\ cR_3 d))) \iff$
$\iff \exists\, b \in B\ \ (aR_1 b\ \&\ bR_2 \circ R_3 d) \iff aR_1 \circ (R_2 \circ R_3) d.$ □

Композиция отношений на множестве A является отношением на множестве A.

Пример. Композиция отношений $<$ и $\leqslant$ на множестве вещественных чисел совпадает с отношением $<$: $< \circ \leqslant\ = <$.

ЗАМЕЧАНИЕ

В общем случае композиция отношений не коммутативна, то есть $R_1 \circ R_2 \neq R_2 \circ R_1$.

Пример. На множестве людей определены отношения кровного родства и супружества. Отношения «кровные родственники супругов» и «супруги кровных родственников» различны.

Из определения непосредственно следует, что если $S_1 \subset R_1\ \&\ S_2 \subset R_2$, то $S_1 \circ S_2 \subset R_1 \circ R_2$.

1.4.6. Свойства отношений

Пусть $R \subset A^2$. Тогда отношение R называется:

рефлексивным, если $\forall\, a \in A\ \ (aRa)$;
антирефлексивным, если $\forall\, a \in A\ \ (\neg aRa)$;
симметричным, если $\forall\, a, b \in A\ \ (aRb \Longrightarrow bRa)$;
антисимметричным, если $\forall\, a, b \in A\ \ (aRb\ \&\ bRa \Longrightarrow a = b)$;

транзитивным, если $\forall\, a, b, c \in A\ \ (aRb\ \&\ bRc \Longrightarrow aRc)$;
линейным, если $\forall\, a, b \in A\ \ (a = b \vee aRb \vee bRa)$.

ЗАМЕЧАНИЕ

Иногда линейное отношение R называют *полным*. Такой термин является оправданным, поскольку для любых двух различных элементов a и b либо пара (a, b), либо пара (b, a) принадлежит отношению R. Отношение, не обладающее свойством полноты, при этом вполне естественно называть *частичным*. Однако использование термина «полное отношение» может порождать терминологическую путаницу. Дело в том, что словосочетание «вполне упорядоченное множество» оказывается созвучным словосочетанию «множество с полным порядком», хотя эти словосочетания обозначают существенно различные объекты. Поэтому здесь термин «линейное отношение» считается антонимом термину «частичное отношение».

ТЕОРЕМА. *Пусть $R \subset A \times A$ — отношение на A. Тогда:*

1) *R рефлексивно* $\Longleftrightarrow I \subset R$;
2) *R симметрично* $\Longleftrightarrow R = R^{-1}$;
3) *R транзитивно* $\Longleftrightarrow R \circ R \subset R$;
4) *R антисимметрично* $\Longleftrightarrow R \cap R^{-1} \subset I$;
5) *R антирефлексивно* $\Longleftrightarrow R \cap I = \varnothing$;
6) *R линейно* $\Longleftrightarrow R \cup I \cup R^{-1} = U$.

Доказательство. Используя определения свойств отношений, имеем:

[1) $\Longleftrightarrow$] $\forall\, a \in A\ \ (aRa) \Longleftrightarrow \forall\, a \in A\ \ ((a, a) \in R) \Longleftrightarrow I \subset R$.

[2) $\Longleftrightarrow$] $\forall\, a, b \in A\ \ (aRb \Longrightarrow bRa) \Longleftrightarrow \forall\, a, b \in A\ \ ((a, b) \in R \Longrightarrow (b, a) \in R) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, b \in A\ \ (((a, b) \in R \Longrightarrow (b, a) \in R)\ \&\ ((b, a) \in R \Longrightarrow (a, b) \in R)) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, b \in A\ \ \big(((a, b) \in R \Longrightarrow (a, b) \in R^{-1})\ \&\ \big((a, b) \in R^{-1} \Longrightarrow (a, b) \in R\big)\big) \Longleftrightarrow$
$\Longleftrightarrow R \subset R^{-1}\ \&\ R^{-1} \subset R \Longleftrightarrow R = R^{-1}$.

[3) $\Longleftrightarrow$] $\forall\, a, b, c \in A\ \ (aRb\ \&\ bRc \Longrightarrow aRc) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, b, c \in A\ \ ((a, b) \in R\ \&\ (b, c) \in R \Longrightarrow (a, c) \in R) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, c \in A\ \ (\exists\, b \in B\ \ ((a, b) \in R\ \&\ (b, c) \in R) \Longrightarrow (a, c) \in R) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, c \in A\ \ ((a, c) \in R \circ R \Longrightarrow (a, c) \in R) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, c \in A\ \ (aR \circ Rc \Longrightarrow aRc) \Longleftrightarrow R \circ R \subset R$.

[4) $\Longrightarrow$] От противного. $R \cap R^{-1} \not\subset I \Longrightarrow \exists\, a \neq b\ \ \big(aRb\ \&\ aR^{-1}b\big) \Longrightarrow$
$\Longrightarrow \exists\, a \neq b\ \ (aRb\ \&\ bRa)$.

[4) $\Longleftarrow$] $R \cap R^{-1} \subset I \Longrightarrow (aRb\ \&\ aR^{-1}b \Longrightarrow a = b) \Longrightarrow (aRb\ \&\ bRa \Longrightarrow a = b)$.

[5) $\Longrightarrow$] От противного. $R \cap I \neq \varnothing \Longrightarrow$
$\Longrightarrow \exists\, a \in A\ \ (aRa\ \&\ aIa) \Longleftrightarrow \exists\, a \in A\ \ (aRa) \Longleftrightarrow \neg\forall\, a \in A\ \ (\neg aRa)$.

[5) $\Longleftarrow$] $R \cap I = \varnothing \Longrightarrow \neg\exists\, a \in A\ \ (aRa) \Longrightarrow \forall\, a \in A\ \ (\neg aRa)$.

[6) $\Longleftrightarrow$] $\forall\, a, b \in A\ \ (a = b \vee aRb \vee bRa) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, b \in A\ \ \big((a, b) \in I \vee (a, b) \in R \vee (a, b) \in R^{-1}\big) \Longleftrightarrow$
$\Longleftrightarrow U \subset I \cup R \cup R^{-1} \Longleftrightarrow U = R \cup I \cup R^{-1}$. □

Пример.
Пусть на множестве $M = \{1, 2, 3, 4\}$ задано бинарное отношение R, график которого показан на рис. 1.8 Какими из перечисленных свойств обладает это отношение? Отношение R не является ни рефлексивным, ни симметричным, ни транзитивным, ни антирефлексивным, ни антисимметричным, ни линейным.

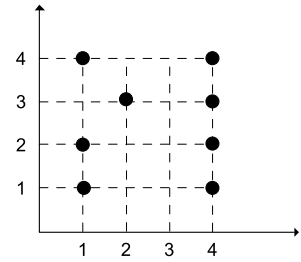


Рис. 1.8. К примеру 1.4.6

1.4.7. Степень отношения и циклы

Пусть R — отношение на множестве A. *Степенью отношения* R на множестве A называется его n-кратная композиция с самим собой. Обозначение:

$$R^n \stackrel{\text{Def}}{=} \underbrace{R \circ \cdots \circ R}_{n \text{ раз}}.$$

Соответственно, $R^0 \stackrel{\text{Def}}{=} I$, $R^1 \stackrel{\text{Def}}{=} R$, $R^2 \stackrel{\text{Def}}{=} R \circ R$ и вообще $R^n \stackrel{\text{Def}}{=} R^{n-1} \circ R$.

Если $(a, b) \in R^n$, то это значит, что $\exists\, c_1, \ldots, c_{n-1} \in A\ (a\, R\, c_1\, R \ldots R\, c_{n-1}\, R\, b)$. Последовательность промежуточных элементов, если их индивидуальные имена не важны, обозначим для краткости $\langle a\, R \ldots R\, b\rangle$. Количество промежуточных элементов на 1 меньше степени отношения. Заметим, что не все элементы в последовательности элементов обязательно различны. Вполне может быть, что $(a, a) \in R^n$. В таком случае последовательность $\langle a\, R \ldots R\, a\rangle$ называется *циклом*. Количество вхождений отношения R в запись цикла (то есть степень отношения) называется *длиной цикла*.

Примеры

Простейший цикл длины 2 образует симметричная пара (a, b), такая что aRb и bRa. Тройка элементов (a, b, c), такая что $(a, b) \in R\ \&\ (b, c) \in R\ \&\ (c, a) \in R$, образует цикл длины 3 и называется *треугольником* и т. д.

ЗАМЕЧАНИЕ

Строго говоря, рефлексивный элемент a, такой что $(a, a) \in R = R^1$, также образует цикл длины 1 и называется *петлёй*. Однако такой цикл является тривиальным и неинтересным с точки зрения свойств отношения в целом. Действительно, если aRa, то $\forall\, n\ (aR^n a)$. По петле можно «накручивать» сколь угодно длинные циклы, и это не даёт никакой новой информации. Поэтому петли обычно не считают циклами.

Отношение R на множестве A ($R \subset A^2$) называется *ациклическим*, если оно не содержит циклов, то есть последовательностей $\langle a\, R \ldots R\, a\rangle$. Свойство ацикличности тесно связано с другими свойствами отношений.

ЛЕММА 1. *Если отношение R на множестве A антисимметрично и транзитивно, то оно ациклично.*

Доказательство. От противного. Пусть $\langle a\, R\, b\, R \ldots R\, a\rangle$ — цикл. Тогда, с одной стороны, aRb, но, с другой стороны, bRa по транзитивности, что противоречит антисимметричности. □

ЛЕММА 2. *Если отношение R на множестве A ациклично, то оно не симметрично и не содержит симметричных пар aRb & bRa.*

Доказательство. Каждая такая пара образует цикл длины 2. □

Вообще говоря, допустимо рассматривать произвольные степени отношения, то есть последовательности элементов и, в частности, циклы произвольной длины. Однако это не имеет особого смысла, как показывает следующая теорема.

ТЕОРЕМА. *Если некоторая пара (a, b) принадлежит какой-либо степени отношения R на множестве A мощности n, то эта пара принадлежит и некоторой степени R не выше $n - 1$:*

$$R \subset A^2 \ \&\ |A| = n \Longrightarrow \forall\, a, b \in A\ \left(\exists\, k\ \left(aR^k b\right) \Longrightarrow \exists\, k < n\ \left(aR^k b\right)\right).$$

Доказательство. Обозначим $c_0 := a$, $c_k := b$. По определению существует последовательность $c_1, \ldots, c_{k-1}$ такая, что $c_0 R c_1 R c_2 R \ldots R c_{k-1} R c_k$. По принципу Дирихле (п. 1.2.5), если $|A| = n$ & $k \geqslant n$, то $\exists\, i, j\ \ (c_i = c_j\ \&\ i \neq j)$, то есть цикл $\langle c_i R \ldots R c_j \rangle$, который можно удалить, получив последовательность меньшей длины $c_0 R c_1 R \ldots R c_i R c_{j+1} R \ldots R c_{k-1} R c_k$, то есть $(a, b) \in R^{k-(j-i)}$. Обозначим через d процедуру, которая вычисляет пару чисел (i, j). Существование требуемого числа k (степени отношения R) обеспечивается следующим построением:
while $k \geqslant n$ **do** $(i, j) := d()$; $k := k - (j - i)$ **end while**. □

СЛЕДСТВИЕ. $R \subset A^2\ \&\ |A| = n \Longrightarrow \bigcup\limits_{i=1}^{\infty} R^i = \bigcup\limits_{i=1}^{n-1} R^i$.

1.4.8. Ядро отношения

Если $R \subset A \times B$ — отношение между множествами A и B, то композиция $R \circ R^{-1}$ называется *ядром* отношения R и обозначается $\ker R$. Другими словами,

$$a_1 \ker R\, a_2 \Longleftrightarrow \exists\, b \in B\ \ (a_1 R\, b\ \&\ a_2 R\, b).$$

Ядро отношения R между A и B является отношением на A:

$$R \subset A \times B \Longrightarrow \ker R \subset A^2.$$

Примеры

1. Пусть отношение N_1 «находиться на расстоянии не более 1» на множестве целых чисел определено следующим образом: $N_1 := \{(n, m) \mid 1 \geqslant |n - m|\}$. Тогда ядром этого отношения является отношение N_2 «находиться на расстоянии не более 2»: $N_2 := \{(n, m) \mid 2 \geqslant |n - m|\}$.
2. Ядром отношения «быть знакомыми» является отношение «быть знакомыми или иметь общего знакомого».
3. Ядро отношения $\in$ между U и 2^U универсально: $\forall\, a, b \in U\ \ \left(\{a, b\} \in 2^U\right)$.
4. Ядро отношения $\subset$ на 2^U также универсально.
5. Рассмотрим отношение «тесного включения» на 2^U:
   $X \overset{+1}{\subset} Y \overset{\text{Def}}{=} X \subset Y\ \&\ |X| + 1 = |Y|$. Ядром отношения $\overset{+1}{\subset}$ является отношение «отличаться не более чем одним элементом»:
   $\ker \overset{+1}{\subset} = \left\{X, Y \in 2^U \mid |X| = |Y|\ \&\ |X \cap Y| \geqslant |X| - 1\right\}$.

ЗАМЕЧАНИЕ

Термин «ядро» и обозначение ker используются также и для других объектов. Их не следует путать — из контекста обычно ясно, в каком смысле используется термин «ядро».

ТЕОРЕМА. *Ядро любого отношения рефлексивно и симметрично на области определения.*

Доказательство. Рассмотрим произвольное отношение $R \subset A \times B$.

[Рефлексивность] Пусть $a \in \operatorname{Dom} R$. Тогда $\exists\, b \in B\ \ (aRb)$ и $\exists\, b \in B\ \ \big(aRb\ \&\ bR^{-1}a\big)$. Имеем $(a,a) \in R \circ R^{-1} \Longrightarrow a \ker R\, a$.

[Симметричность] Пусть $a, b \in \operatorname{Dom} R\ \&\ a \ker R\, b$. Тогда $\exists\, c \in B\ \ \big(aRc\ \&\ cR^{-1}b\big)$, откуда $\exists\, c \in B\ \ \big(bRc\ \&\ cR^{-1}a\big)$, и значит $(b,a) \in R \circ R^{-1} \Longrightarrow b \ker R\, a$. □

Пример.
Пусть на множестве $M = \{1,2,3,4,5,6\}$ задано бинарное отношение $R = \{(x,y) \mid xy = 6\}$. Обратное отношение совпадает с исходным, а ядро рефлексивно и симметрично, хотя и не линейно. Эти отношения изображены на рис. 1.9. График исходного отношения (и обратного отношения) изображён закрашенными кружками, график ядра (и квадрата) — незакрашенными кружками.

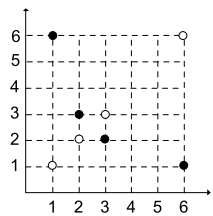


Рис. 1.9. К примеру 1.4.8

1.4.9. Представление отношений в программах

Пусть R — отношение на A, $R \subset A^2$ и $|A| = n$. Перенумеруем элементы множества A, $A = \{a_1, \ldots, a_n\}$. Тогда отношение R можно представить *булевой* матрицей $\boxed{R}$: **array** $[1..n, 1..n]$ **of** 0..1, где $\forall\, i,j \in 1..n\ \ \Big(\boxed{R}\ [i,j] = a_i R a_j\Big)$.

ЗАМЕЧАНИЕ

Термин «булева матрица» означает, что элементы матрицы — булевы значения и операции над ними выполняются по соответствующим правилам. В частности, если $\boxed{A}$ и $\boxed{B}$ — булевы матрицы, то операции на ними определяются следующим образом:

транспонирование: $\boxed{A}^{\mathrm{T}}[i,j] \stackrel{\text{Def}}{=} \boxed{A}\ [j,i]$;
вычитание: $\Big(\boxed{A} - \boxed{B}\Big)\ [i,j] \stackrel{\text{Def}}{=} \boxed{A}\ [i,j]\ \&\ \Big(1 - \boxed{B}\ [i,j]\Big)$;
инвертирование: $\overline{\boxed{A}}\ [i,j] \stackrel{\text{Def}}{=} 1 - \boxed{A}\ [i,j]$;
умножение: $\Big(\boxed{A} \times \boxed{B}\Big)\ [i,j] \stackrel{\text{Def}}{=} \bigvee\limits_{k=1}^{n} \Big(\boxed{A}\ [i,k]\ \&\ \boxed{B}\ [k,j]\Big)$;
дизъюнкция: $\Big(\boxed{R_1} \vee \boxed{R_2}\Big)\ [i,j] \stackrel{\text{Def}}{=} \boxed{R_1}\ [i,j] \vee \boxed{R_2}\ [i,j]$.

В следующих утверждениях предполагается, что все рассматриваемые отношения определены на множестве A, причём $|A| = n$.

ТЕОРЕМА 1. $\boxed{R^{-1}} = \boxed{R}^{\mathrm{T}}$.

Доказательство. $(b,a) \in R^{-1} \Longleftrightarrow (a,b) \in R \Longleftrightarrow \boxed{R}\,[a,b] = 1 \Longleftrightarrow \boxed{R}^{\mathrm{T}}\,[b,a] = 1.$ □

В универсальное отношение U входят все пары элементов, поэтому все элементы матрицы универсального отношения $\boxed{U}$ равны 1.

ТЕОРЕМА 2. $\boxed{\overline{R}} = \boxed{U} - \boxed{R}$.

Доказательство. $(a,b) \in \overline{R} \Longleftrightarrow (a,b) \notin R \Longleftrightarrow$
$\Longleftrightarrow \boxed{R}\,[a,b] = 0 \Longleftrightarrow \boxed{U}\,[a,b] - \boxed{R}\,[a,b] = 1 \Longleftrightarrow \left(\boxed{U} - \boxed{R}\right)[a,b] = 1.$ □

СЛЕДСТВИЕ. $\boxed{\overline{R}} = \overline{\boxed{R}}$.

ТЕОРЕМА 3. $\boxed{R_1 \circ R_2} = \boxed{R_1} \times \boxed{R_2}$.

Доказательство. $(a,b) \in R_1 \circ R_2 \Longleftrightarrow \exists\, c \in A\ \ (aR_1c\ \&\ cR_2b) \Longleftrightarrow$
$\Longleftrightarrow \exists\, c \in A\ \left(\boxed{R_1}\,[a,c] = 1\ \&\ \boxed{R_2}\,[c,b] = 1\right) \Longleftrightarrow$
$\Longleftrightarrow \exists\, c \in A\ \left(\left(\boxed{R_1}\,[a,c]\ \&\ \boxed{R_2}\,[c,b]\right) = 1\right) \Longleftrightarrow$
$\Longleftrightarrow \left(\bigvee_{k=1}^{n} \boxed{R_1}\,[a,k]\ \&\ \boxed{R_2}\,[k,b]\right) = 1 \Longleftrightarrow \left(\boxed{R_1} \times \boxed{R_2}\right)\ [a,b]\ =\ 1.$ □

СЛЕДСТВИЕ. $\boxed{R^k} = \boxed{R}^k$.

ТЕОРЕМА 4. $\boxed{R_1 \cup R_2} = \boxed{R_1} \vee \boxed{R_2}$.

Доказательство. $(a,b) \in R_1 \cup R_2 \Longleftrightarrow aR_1b \vee aR_2b \Longleftrightarrow$
$\Longleftrightarrow \boxed{R_1}\,[a,b] = 1 \vee \boxed{R_2}\,[a,b] = 1 \Longleftrightarrow \left(\boxed{R_1} \vee \boxed{R_2}\right)\ [a,b] = 1.$ □

ЗАМЕЧАНИЕ

Представление отношений с помощью булевых матриц — это только один из возможных способов представления отношений. Другие варианты рассматриваются при обсуждении представления графов.

1.5. Замыкание и сокращение отношений

Замыкание является весьма общим математическим понятием. Неформально говоря, замкнутость означает, что многократное выполнение допустимых шагов не выводит за определённые границы. Например, предел сходящейся последовательности чисел из *замкнутого* интервала $[a,b]$ обязательно принадлежит этому интервалу, а предел сходящейся последовательности чисел из открытого (то есть *не* замкнутого) интервала (a,b) может лежать вне этого интервала. В некоторых случаях можно «расширить» незамкнутый объект так, чтобы он оказался замкнутым.

Сокращение, или *редукция*, в некотором смысле противоположно замыканию. Если в замыкание включается всё, что только можно получить из исходного объекта, то в сокращение включается минимум информации из исходного объекта, но так, чтобы можно было получить всё то, что несёт в себе исходный объект. Другими словами, замыкание редукции некоторого объекта совпадает с замыканием исходного объекта.

1.5.1. Транзитивное и рефлексивное замыкание

Пусть R и R' — отношения на множестве M. Отношение R' называется *замыканием* R относительно свойства C, если:

1) R' обладает свойством C: $C(R')$;

2) R' является надмножеством R: $R \subset R'$;

3) R' является наименьшим таким объектом: $C(R'') \ \&\ R \subset R'' \Longrightarrow R' \subset R''$.

Обозначим R^+ — объединение положительных степеней R, а R^* — объединение неотрицательных степеней R: $\quad R^+ \stackrel{\text{Def}}{=} \bigcup\limits_{i=1}^{\infty} R^i, \quad R^* \stackrel{\text{Def}}{=} \bigcup\limits_{i=0}^{\infty} R^i.$

ТЕОРЕМА. *R^+ — транзитивное замыкание R.*

Доказательство. Проверим выполнение свойств замыкания при условии, что свойство C — это транзитивность.

[$C(R^+)$] Пусть $aR^+b \ \&\ bR^+c$. Тогда $\exists n \ \ (aR^nb) \ \&\ \exists m \ \ (bR^mc)$. Следовательно, $\exists c_1, \ldots, c_{n-1} \ \ (aRc_1R \ldots Rc_{n-1}Rb)$ и $\exists d_1, \ldots, d_{m-1} \ \ (bRd_1R \ldots Rd_{m-1}Rc)$. Таким образом, $aRc_1R \ldots Rc_{n-1}RbRd_1R \ldots Rd_{m-1}Rc \Longrightarrow aR^{n+m+1}c \Longrightarrow aR^+c$.

[$R \subset R^+$] $\ R = R^1 \Longrightarrow R \subset \bigcup\limits_{i=1}^{\infty} R^i \Longrightarrow R \subset R^+$.

[$C(R'') \ \&\ R \subset R'' \Longrightarrow R^+ \subset R''$] $\ aR^+b \Longrightarrow \exists k, c_1, \ldots, c_{k-1} \ \ (aRc_1R \ldots Rc_{k-1}Rb)$; $R \subset R'' \Longrightarrow aR''c_1R'' \ldots R''c_{k-1}R''b \Longrightarrow aR''b$. Таким образом, $R^+ \subset R''$. □

СЛЕДСТВИЕ. *R^* — рефлексивное транзитивное замыкание R.*

Вычислить транзитивное замыкание заданного отношения можно следующим образом: $R^+ = \bigcup\limits_{i=1}^{\infty} R^i = \bigcup\limits_{i=1}^{n-1} R^i \Longrightarrow \boxed{R^+} = \bigvee\limits_{i=1}^{n-1} \boxed{R^i} = \bigvee\limits_{i=1}^{n-1} \boxed{R}^i$. Такое вычисление имеет сложность $O(n^4)$.

1.5.2. Алгоритм Уоршалла вычисления транзитивного замыкания отношения

Рассмотрим *алгоритм* Уоршалла[1] вычисления транзитивного замыкания отношения R на множестве M, $|M| = n$, имеющий сложность $O(n^3)$.

Обоснование. На каждом шаге основного цикла (по i) в транзитивное замыкание добавляются такие пары элементов с номерами j и k (то есть $T[j,k] := 1$), для которых существует последовательность промежуточных элементов с номерами в диапазоне $1 \ldots i$, связанных отношением R. Действительно, последовательность промежуточных элементов с номерами в диапазоне $1..i$, связанных отношением R, для

[1] Стефан Уоршалл (1935–2006).

Алгоритм 1.13. Вычисление транзитивного замыкания отношения

Вход: матрица отношения R: **array**$[1..n, 1..n]$ **of** 0..1.
Выход: матрица замыкания T: **array**$[1..n, 1..n]$ **of** 0..1.

```
S := R
for i from 1 to n do
  for j from 1 to n do
    for k from 1 to n do
      T[j,k] := S[j,k] ∨ S[j,i] & S[i,k]
    end for
  end for
  S := T
end for
```

элементов с номерами j и k существует в одном из двух случаев: либо уже существует последовательность промежуточных элементов с номерами в диапазоне $1..(i-1)$ для пары элементов с номерами j и k, либо существуют *две* последовательности элементов с номерами в диапазоне $1..(i-1)$ — одна для пары элементов с номерами j и i и вторая для пары элементов с номерами i и k. Именно это отражено в операторе $T[j,k] := S[j,k] \vee S[j,i]\ \&\ S[i,k]$. После окончания основного цикла в качестве промежуточных рассмотрены все элементы, и, таким образом, построено транзитивное замыкание. □

1.5.3. Транзитивное сокращение

Транзитивным сокращением (или *транзитивной редукцией*, или *минимальным представлением*) отношения R на множестве A называется наименьшее (по количеству элементов) отношение, обозначаемое R^-, такое, что $R^{-+} = R^+$. Транзитивное сокращение содержит минимально необходимые данные, по которым, используя транзитивность, можно восстановить транзитивное замыкание отношения в полном объёме.

Пример. На рис. 1.10 представлены (слева направо) графики отношения, его транзитивного замыкания и транзитивной редукции, определенные на множестве 1..5.

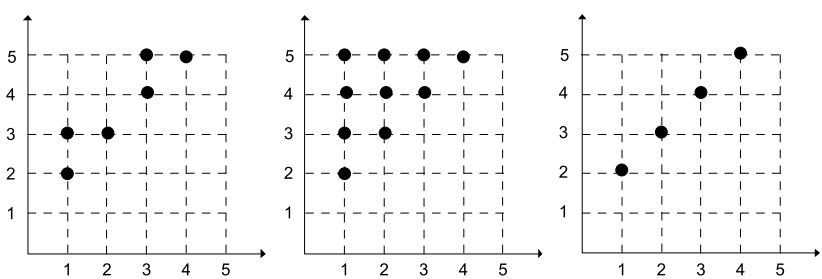


Рис. 1.10. Графики отношения, замыкания и редукции

Задача отыскания транзитивного сокращения заданного отношения R является практически значимой и не является тривиальной. Действительно, обозначим $\mathfrak{R}(R) = \{S \subset A^2 \mid S^+ = R^+\}$ — семейство отношений, транзитивные замыкания которых совпадают с транзитивным замыканием исходного отношения. Ясно, что $\mathfrak{R}(R) \neq \varnothing$, поскольку $R \in \mathfrak{R}(R)$, и ясно, что семейство отношений $\mathfrak{R}(R)$ может быть весьма велико, грубая оценка сверху $|\mathfrak{R}(R)| \leqslant 2^{|A|^2}$, поскольку состав семейства $\mathfrak{R}(R)$ не очевиден. Таким образом, налицо экстремальная задача: найти глобальный минимум в обширном пространстве поиска. К счастью, некоторые свойства исходного отношения упрощают нахождение его транзитивного сокращения.

ЗАМЕЧАНИЕ

Полезно сравнить содержание этого параграфа с параграфом 8.5.4. Исторически идея транзитивной редукции возникла применительно к ориентированным графам, как средство построить минимальный по числу дуг орграф, имеющий заданное отношение достижимости. В такой постановке задача транзитивной редукции особенно наглядна. Поэтому алгоритмы отыскания минимального представления удобнее формулировать на языке графов, хотя они применимы к любым отношениям.

ЛЕММА 1. *Пусть $S, T \subset A^2$ — конечные ациклические отношения на множестве A и $S^+ = T^+$. Тогда если $\exists\,(a,b)\ ((a,b) \in S\ \&\ (a,b) \notin T)$, то $(S \setminus \{(a,b)\})^+ = S^+$.*

Доказательство. Пусть $(a,b) \in S\ \&\ (a,b) \notin T$. Поскольку $S^+ = T^+$, существует последовательность промежуточных элементов $\langle aT \ldots Tb\rangle$. Обозначим один из промежуточных элементов c. Тогда существуют последовательности $\langle aT \ldots Tc\rangle$ и $\langle cT \ldots Tb\rangle$, но в силу того, что $S^+ = T^+$, существуют и последовательности $\langle aS \ldots Sc\rangle$ и $\langle cS \ldots Sb\rangle$. Если предположить, что $b \in \langle aS \ldots Sc\rangle$, получаем две последовательности $\langle bS \ldots Sc\rangle$ и $\langle cS \ldots Sb\rangle$, то есть цикл, что противоречит ацикличности S. Аналогично, если предположить, что $a \in \langle cS \ldots Sb\rangle$, получаем две последовательности $\langle cS \ldots Sc\rangle$ и $\langle cS \ldots Sa\rangle$, то есть цикл, что также противоречит ацикличности S. Значит, пара (a,b) для построения транзитивного замыкания S избыточна, что и доказывает требуемое равенство $(S \setminus (a,b))^+ = S^+$. □

ЛЕММА 2. *Если отношение R ациклично и конечно, то семейство $\mathfrak{R}(R)$ замкнуто относительно пересечения, то есть $\forall\, S, T \in \mathfrak{R}(R)\ (S \cap T \in \mathfrak{R}(R))$.*

Доказательство. Пусть $S, T \in \mathfrak{R}(R)$ — любые два элемента семейства. Обозначим $\{(a_1,b_1),(a_2,b_2),\ldots,(a_k,b_k)\} := S \setminus T = S \setminus (S \cap T)$. Последовательно применяя лемму 1 имеем: $(S \setminus \{(a_1,b_1)\})^+ = S^+$, $(S \setminus \{(a_1,b_1)\} \setminus \{(a_2,b_2)\})^+ = S^+$, …, $(S \setminus \{(a_1,b_1)\} \setminus \{(a_2,b_2)\} \setminus \cdots \setminus \{(a_k,b_k)\})^+ = S^+ = R^+$. Но $S \setminus \{(a_1,b_1)\} \setminus \{(a_2,b_2)\} \setminus \cdots \setminus \{(a_k,b_k)\} = S \cap T$, значит $(S \cap T)^+ = S^+ = R^+$ и $(S \cap T) \in \mathfrak{R}(R)$. □

ТЕОРЕМА. *Если отношение R ациклично и конечно, то транзитивная редукция R^- единственна и определяется по формуле $R^- = \bigcap\limits_{S \in \mathfrak{R}(R)} S$.*

Доказательство. Отношение R конечно, поэтому и семейство $\mathfrak{R}(R)$ конечно, следовательно, R^- всегда определено и единственно по построению. По лемме 2 имеем $R^- \in \mathfrak{R}(R)$. Наконец, $\forall\, S \in \mathfrak{R}(R)\ (|R^-| \leqslant |S|)$. □

СЛЕДСТВИЕ. *Если отношение R ациклично и конечно, то транзитивное сокращение R^- обладает следующими свойствами:*

1) $R^- \subset R$;
2) $\forall\,(a,b) \in R^-\ (\neg\exists\langle aR^- \ldots R^-b\rangle)$;
3) $R^- = R \setminus (R \circ R^+)$.

Доказательство.

[1] $R^- = \bigcap\limits_{S\in\mathcal{R}(R)} S \subseteq R$, поскольку $R \in \mathcal{R}(R)$;

[2] если $\exists\langle aR^- \ldots R^-b\rangle$, то $(R^- \setminus \{(a,b)\})^+ = (R^-)^+$, и значит, пару (a,b) можно исключить из транзитивного сокращения, что противоречит его минимальности;

[3] следует из того, что $(a,b) \in R \circ R^+ \iff \exists\langle aR \ldots Rb\rangle$. □

ЗАМЕЧАНИЕ

Транзитивное замыкание определено для любого отношения, единственно и содержит исходное отношение. Если выполнены условия теоремы, то транзитивное сокращение также определено, единственно и содержится в исходном отношении. Однако если условия теоремы не выполнены, то транзитивное сокращение может быть не определено, не единственно и может не входить в исходное отношение.

Пример. На рис. 1.11 слева представлен график отношения на множестве $\{1,2,3\}$, которое содержит четыре элемента (два цикла длины 2) и имеет в качестве транзитивного замыкания универсальное отношение. При этом ни одну из пар нельзя исключить из отношения так, чтобы транзитивное замыкание сохранилось. Однако справа на этом же рисунке представлены графики двух различных отношений, у которых транзитивное замыкание также является универсальным отношением, но при этом в них всего по три элемента. Обратите внимание, что ни одно из транзитивных сокращений не является подмножеством исходного отношения!

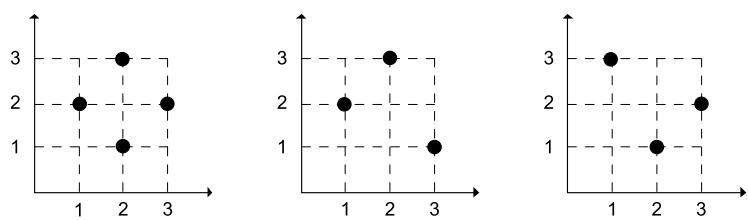


Рис. 1.11. График отношения и двух его транзитивных сокращений

1.5.4. Диаграммы Хассе

В самом общем случае *диаграммой Хассе* для отношения R на множестве A (обозначение $\mathcal{H}(R)$) называется отношение

$$\mathcal{H}(R) \stackrel{\text{Def}}{=} \{(a,b) \in R \mid \neg\exists\, c \in A\ \ (aRc\ \&\ cRb)\}\,.$$

Другими словами, в диаграмму Хассе заданного отношения включаются только такие пары элементов, которые являются «соседними».

Пример. Рассмотрим отношение «тесного включения» на 2^U (см. пример 5 п. 1.4.8): $X \overset{+1}{\subset} Y \overset{\text{Def}}{=} X \subset Y \ \&\ |X|+1=|Y|$. Ядром отношения $\overset{+1}{\subset}$ является отношение «отличаться не более чем одним элементом»:
$\ker \overset{+1}{\subset} = \left\{X, Y \in 2^U \mid |X|=|Y| \ \&\ |X \cap Y| \geqslant |X|-1\right\}$. При этом отношение «тесного включения» является диаграммой Хассе для отношения включения на булеане 2^U, то есть $\mathcal{H}(\subset) = \overset{+1}{\subset}$.

Диаграмма Хассе, подобно транзитивному сокращению, более экономно представляет исходное отношение. Однако диаграмма Хассе не тождественна транзитивному сокращению хотя бы потому, что диаграмма Хассе обязательно является подмножеством исходного отношения, в то время как транзитивное сокращение может даже не пересекаться с исходным отношением. Более того, в некоторых случаях диаграммы Хассе может быть недостаточно для восстановления самого отношения или его транзитивного замыкания.

Примеры

1. Все отношения, графики которых представлены на рис. 1.11, совпадают со своими диаграммами Хассе. В то же время транзитивное сокращение отношения, график которого приведен на рис. 1.11 слева, не совпадает с исходным отношением.
2. Два отношения, диаграммы которых приведены на рис. 1.11 справа, являются транзитивными сокращениями друг для друга.
3. Для универсального отношения на трех элементах диаграмма Хассе пуста.

ТЕОРЕМА. *Если конечное отношение R антисимметрично и транзитивно, то*

$$\mathcal{H}(R) = R^-.$$

Доказательство. Антисимметричное транзитивное отношение ациклично (п. 1.4.7), значит, действует теорема п. 1.5.3 и следствия к ней. По первому следствию $R^- \subset R$, а по второму следствию $\forall (a,b) \in R^- \ \ (\neg\exists \langle aR^- \ldots R^- b\rangle)$, и тем более $\forall (a,b) \in R^- \ \ (\neg\exists\, c \in A \ \ (a\, R\, c\, R\, b))$. Значит, для транзитивного сокращения в данном случае выполняются условия определения диаграммы Хассе. □

ЗАМЕЧАНИЕ

Для отношений на бесконечных множествах диаграмма Хассе может не существовать.

Пример. Множество вещественных чисел $\mathbb{R}$ и множество рациональных чисел $\mathbb{Q}$ с обычным отношением порядка не имеют диаграмм Хассе: между любыми двумя элементами всегда можно вставить третий.

Диаграмма Хассе существует для любого отношения на конечном множестве, что показывает алгоритм 1.14, прямо следующий из определения.

Диаграммы Хассе часто используют для представления упорядоченных множеств (см. п. 1.8.2) и решёток (см. раздел 2.6).

Особенно удобным является графическое представление, из-за которого диаграммы Хассе и получили название диаграмм. При этом элементы множества A изображаются в определённых фигурах, например, в кругах или овалах, а элементы самой

Алгоритм 1.14. Построение диаграммы Хассе

Вход: отношение, заданное матрицей R: **array** $[1..n, 1..n]$ **of** 0..1.
Выход: диаграмма Хассе, заданная матрицей H: **array**$[1..n, 1..n]$ **of** 0..1.

```
for i from 1 to n do
  for j from 1 to n do
    if R[i,j] = 1 then
      H[i,j] := 1  // возможно, это соседи
      for k from 1 to n do
        if R[i,k] = 1 & R[k,j] = 1 & k ≠ i & k ≠ j then
          H[i,j] := 0  // найден промежуточный элемент
          exit for k  // можно больше не искать
        end if
      end for
    else
      H[i,j] := 0  // диаграмма Хассе — подмножество отношения
    end if
  end for
end for
```

диаграммы, то есть упорядоченные пары (a, b), изображаются в виде стрелочки, ведущей от a к b. На рис. 1.12 приведен пример диаграммы Хассе для отношения включения на множестве $2^{\{1,2,3\}}$.

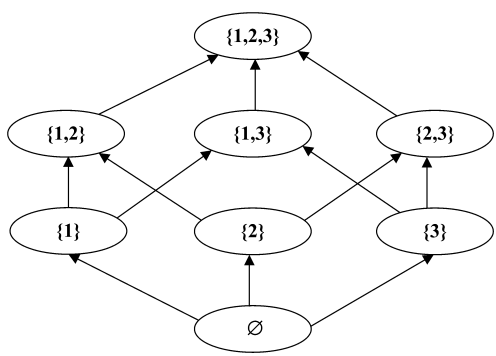


Рис. 1.12. Диаграмма Хассе для отношения включения на множестве $2^{1,2,3}$

1.6. Функции

Понятие «функция» является одним из основополагающих в математике. В данном случае подразумеваются прежде всего функции, отображающие одно конечное множество объектов в другое конечное множество. Термин «отображение» мы считаем синонимом термина «функция», но различаем контексты употребления. Термин «функция» используется в общем случае и в тех случаях, когда элементы отображаемых множеств не имеют структуры или она не важна. Термин «отображение» применяется, напротив, только в тех случаях, когда структура отображаемого множества имеет первостепенное значение. Например, мы говорим «булева функция», но «отображение групп». Такое предпочтение слову «функция» оказывается в

расчёте на постоянное сопоставление читателем математического понятия функции с понятием функции в языках программирования.

1.6.1. Функциональные отношения

Пусть f — отношение между A и B такое, что

$$\forall a \ ((a,b) \in f \ \& (a,c) \in f \Longrightarrow b = c) .$$

Такое свойство отношения называется *однозначностью*, или *функциональностью*, а само отношение называется *функцией* из A в B, причём для записи используется одна из следующих форм: $f\colon A \to B$ или $A \xrightarrow{f} B$.

Если $f\colon A \to B$, то обычно используется *префиксная* форма записи:

$$b = f(a) \stackrel{\text{Def}}{=} (a,b) \in f.$$

Если $b = f(a)$, то a называют *аргументом*, а b — *значением* функции. Если из контекста ясно, о какой функции идёт речь, то иногда используется обозначение $a \mapsto b$. Поскольку функция является отношением, для функции $f\colon A \to B$ можно использовать уже введенные понятия: множество A называется *областью отправления*, множество B — *областью прибытия*. Также используют термины *область определения* функции: $\mathrm{Dom}\, f \stackrel{\text{Def}}{=} \{a \in A \mid \exists\, b \in B \ (b = f(a))\}$; *область значений* функции: $\mathrm{Im}\, f \stackrel{\text{Def}}{=} \{b \in B \mid \exists\, a \in A \ (b = f(a))\}$.

Область определения является подмножеством области отправления, $\mathrm{Dom}\, f \subset A$, а область значений является подмножеством области прибытия, $\mathrm{Im}\, f \subset B$. Если $\mathrm{Dom}\, f = A$, то функция называется *тотальной*, а если $\mathrm{Dom}\, f \neq A$ — *частичной*. *Сужением* функции $f\colon A \to B$ на множество $M \subset A$ называется функция $f|_M$, определяемая следующим образом: $f|_M \stackrel{\text{Def}}{=} \{(a,b) \mid (a,b) \in f \ \&\ a \in M\}$. Функция f называется *продолжением* функции g, если g является сужением f.

ОТСТУПЛЕНИЕ

В математике, как правило, рассматриваются тотальные функции, а частичные функции объявлены «ущербными». Программисты же имеют дело с частичными функциями, которые определены не для всех допустимых значений. Например, математическая функция x^2 определена для всех x, а стандартная функция языка программирования sqr даёт правильный результат отнюдь не для всех возможных значений аргумента.

Далее, если явно не оговорено противное, речь идёт о тотальных функциях, поэтому области отправления и определения совпадают.

Тотальная функция $f\colon M \to M$ называется *преобразованием* над M.

Функция $f\colon A_1 \times \ldots \times A_n \to B$ называется функцией n *аргументов*, или n*-местной* функцией. Такая функция отображает кортеж $(a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$ в элемент $b \in B$, $b = f((a_1, \ldots, a_n))$. При записи лишние скобки обычно опускают: $b = f(a_1, \ldots, a_n)$.

1.6.2. Инъекция, сюръекция и биекция

Пусть $f\colon A \to B$. Тогда функция f называется:

инъективной, или *инъекцией*,	если $b = f(a_1)\ \&\ b = f(a_2) \Longrightarrow a_1 = a_2$;
сюръективной, или *сюръекцией*,	если $\forall\, b \in B \ (\exists\, a \in A \ (b = f(a)))$;
биективной, или *биекцией*,	если она инъективная и сюръективная.

ЗАМЕЧАНИЕ

Биективную функцию также называют *взаимно-однозначной*. Введенное в п. 1.2.2 взаимно-однозначное соответствие есть биекция.

Понятия функциональности, инъективности, сюръективности и тотальности применимы к любым отношениям, а не только к функциям. Таким образом, получаем четыре парных свойства отношения $f \subset A \times B$, которые для удобства запоминания можно свести в следующую таблицу:

A	B
Функциональность $(a,b) \in f \;\&\; (a,c) \in f \Longrightarrow b = c$	Инъективность $(a,b) \in f \;\&\; (c,b) \in f \Longrightarrow a = c$
Тотальность $\forall\, a \in A \;\; (\exists\, b \in B \;\; ((a,b) \in f))$	Сюръективность $\forall\, b \in B \;\; (\exists\, a \in A \;\; ((a,b) \in f))$

ТЕОРЕМА. *Если $f\colon A \to B$ — биекция, то отношение $f^{-1} \subset B \times A$ (обратная функция) является биекцией.*

Доказательство. Поскольку f — биекция, имеем $b_1 = f(a) \;\&\; b_2 = f(a) \Longrightarrow b_1 = b_2$, $b = f(a_1) \;\&\; b = f(a_2) \Longrightarrow a_1 = a_2$, $\forall\, b \in B \;\; (\exists\, a \in A \;\; (b = f(a)))$.

[f^{-1} — функция] Имеем $f^{-1} = \{(b,a) \mid a \in A \;\&\; b \in B \;\&\; b = f(a)\}$.
Пусть $a_1 = f^{-1}(b) \;\&\; a_2 = f^{-1}(b)$. Тогда $b = f(a_1) \;\&\; b = f(a_2) \Longrightarrow a_1 = a_2$.

[f^{-1} — инъекция] Пусть $a = f^{-1}(b_1) \;\&\; a = f^{-1}(b_2)$.
Тогда $b_1 = f(a) \;\&\; b_2 = f(a) \Longrightarrow b_1 = b_2$.

[f^{-1} — сюръекция] От противного. Пусть $\exists\, a \in A \;\; \big(\neg \exists\, b \in B \;\; \big(a = f^{-1}(b)\big)\big)$. Тогда $\exists\, a \in A \;\; \big(\forall\, b \in B \;\; \big(a \neq f^{-1}(b)\big)\big)$. Обозначим этот элемент a_0.
Имеем $\forall\, b \;\; \big(a_0 \neq f^{-1}(b)\big) \Longrightarrow \forall\, b \;\; (b \neq f(a_0)) \Longrightarrow a_0 \notin \mathrm{Dom}\, f \subset A \Longrightarrow a_0 \notin A$. □

СЛЕДСТВИЕ. *Если $f\colon A \to B$ — инъективная функция, то отношение $f^{-1} \subset B \times A$ является функцией (возможно, частичной) $f^{-1}\colon B \to A$.*

Рис. 1.13 иллюстрирует понятия отношения, функции, инъекции, сюръекции и биекции.

1.6.3. Образы и прообразы

Пусть $f\colon A \to B$ и пусть $A_1 \subset A$, $B_1 \subset B$. Тогда множество

$$F(A_1) \stackrel{\text{Def}}{=} \{b \in B \mid \exists\, a \in A_1 \;\; (b = f(a))\}$$

называется *образом* множества A_1 (при отображении f), а множество

$$F^{-1}(B_1) \stackrel{\text{Def}}{=} \{a \in A \mid \exists\, b \in B_1 \;\; (b = f(a))\}$$

называется *прообразом* множества B_1 (при отображении f). Введём обозначения $f_A := \mathrm{Dom}\, f$, $f_B := \mathrm{Im}\, f$. Заметим, что F является отношением между множествами 2^{f_A} и 2^{f_B}:

$$F \stackrel{\text{Def}}{=} \{(A_1, B_1) \mid A_1 \subset A \;\&\; B_1 \subset B \;\&\; B_1 = F(A_1)\},$$

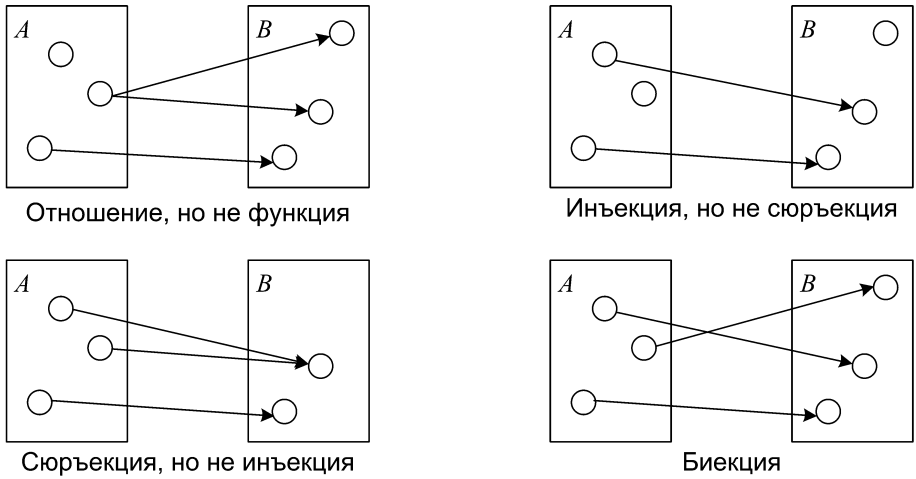


Рис. 1.13. Некоторые виды отношений

а F^{-1} является обратным отношением между множествами 2^{f_B} и 2^{f_A}:

$$F^{-1} \stackrel{\text{Def}}{=} \{(B_1, A_1) \mid A_1 \subset A \ \& \ B_1 \subset B \ \& \ B_1 = F(A_1)\}.$$

Заметим, что если f — функция, то для любого множества образ и прообраз определены однозначно. Из этого наблюдения немедленно следует теорема.

ТЕОРЕМА. *Если $f\colon A \to B$ — функция, то $F\colon 2^{f_A} \to 2^{f_B}$ (переход к образам) и $F^{-1}\colon 2^{f_B}$ (переход к прообразам) — тоже функции.*

ЗАМЕЧАНИЕ

Фактически это означает, что функцию можно применять не только к элементам области определения, но и к любым её подмножествам. На практике в целях упрощения записи для обозначения применения функции к подмножеству используют ту же самую букву, что и для применения функции к отдельному элементу.

Пусть задана функция $f\colon M \to M$, а функции $F, F^{-1}\colon 2^M \to 2^M$ обозначают функции перехода к образам и прообразам соответственно. Тогда нетрудно показать, что

$$\forall X, Y \in M \ \ (F(X \cap Y) \subset F(X) \cap F(Y)) \text{ и } \forall X, Y \in M \ \ (F(X \cup Y) \subset F(X) \cup F(Y)).$$

Обычно по определению полагают, что $F(\varnothing) \stackrel{\text{Def}}{=} F^{-1}(\varnothing) \stackrel{\text{Def}}{=} \varnothing$.

1.6.4. Суперпозиция функций

Поскольку функции являются частным случаем отношений, для них определена композиция. Композиция функций называется *суперпозицией*. Для обозначения суперпозиции применяют тот же знак $\circ$, но операнды записывают в обратном порядке: если $f\colon A \to B$ и $g\colon B \to C$, то суперпозиция функций f и g записывается так: $g \circ f$. Этот способ записи суперпозиции функций объясняется тем, что обозначение

функции принято писать слева от списка аргументов:

$$(g \circ f)(x) \stackrel{\text{Def}}{=} g(f(x)).$$

ТЕОРЕМА. *Суперпозиция функций является функцией:*

$$f\colon A \to B \ \ \& \ \ g\colon B \to C \Longrightarrow g \circ f\colon A \to C.$$

Доказательство. Пусть $b = (g \circ f)(a)$ и $c = (g \circ f)(a)$. Тогда $b = g(f(a))$ и $c = g(f(a))$. Но f и g — функции, поэтому $b = c$. □

СЛЕДСТВИЕ. *Суперпозиция биекций является биекцией.*

1.6.5. Степень функции

Поскольку функция является отношением, можно ввести степень функции относительно суперпозиции. *Степенью* n функции f (обозначение f^n) называется n-кратная суперпозиция функции f. Соответственно, $f^0(x) = x$, $f^1(x) = f(x)$ и вообще $f^n = f \circ f^{n-1}$. При записи суперпозиций функций знак $\circ$ часто опускают.

ЗАМЕЧАНИЕ

В математическом анализе часто используют такое же обозначение для степени значения функции относительно умножения. Например, $\sin^2(x) = \sin(x) \cdot \sin(x)$. Следует различать по контексту, в каком смысле используется степень функции. В частности, в контексте данной книги $\sin^2(x) = \sin(\sin(x))$.

Пусть f — любая функция. Рассмотрим последовательность степеней:

$$f^0 = x, \quad f^1 = f, \quad f^2 = ff, \quad \ldots, \quad f^n = ff^{n-1}.$$

Заметим, что порядок сомножителей в определении степени несуществен.

ЛЕММА. $ff^n = f^n f$.

Доказательство. По индукции. База: $ff = ff$. Пусть $ff^n = f^n f$. Тогда $f^{n+1}f = (ff^n)f = f(f^n f) = f(ff^n) = ff^{n+1}$. □

Порядок вычисления степеней функции также несуществен, важна только сама степень.

ТЕОРЕМА 1. $f^n f^m = f^{n+m}$.

Доказательство. $f^n f^m = (ff^{n-1})(f^m) = (f^{n-1}f)(f^m) = (f^{n-1})(ff^m) = (f^{n-1})(f^{m+1}) = \ldots = (f^0)(f^{n+m}) = f^{n+m}$. □

Хотя композиция отношений в общем случае не коммутативна, но степени функции коммутируют.

СЛЕДСТВИЕ. $f^n f^m = f^m f^n$.

Доказательство. $f^n f^m = f^{n+m} = f^{m+n} = f^m f^n$. □

Бесконечная последовательность элементов $(a_i)_{i=1}^{\infty}$ называется *заключительно периодической*, если $\exists m, k \in \mathbb{N}\ (\forall i \geqslant m\ \ (a_i = a_{i+k}))$. Число k называется *периодом*. Если число $m = 1$, то последовательность называется просто *периодической*.

Пусть задана функция $f\colon M \to M$, $|M| = n$, $\mathrm{Dom}\, f = M$. Во многих приложениях используются последовательности $(f_i(a))_{i=1}^{\infty}$ для некоторого $a \in M$.

ТЕОРЕМА 2. *Если функция на конечном множестве тотальна, то последовательность значений степеней этой функции для любого аргумента заключительно периодическая.*

Доказательство. Рассмотрим последовательность $(f_i(a))_{i=1}^{\infty}$ для некоторого элемента $a \in M$. Эта последовательность существует, поскольку функция f тотальна. По принципу Дирихле в этой последовательности есть равные элементы. Пусть i и j, $i < j$ — наименьшая пара чисел, такая что $f^i(a) = f^j(a)$. Тогда положим $m := i$, $k := j - i$. □

Пример. Рассмотрим числа, записанные четырьмя цифрами. Отсортируем цифры в порядке невозрастания и в порядке неубывания, и вычтем меньшее число из большего. Получится число, записанное четырьмя цифрами. Например, взяв число 0231, имеем $3210 - 0123 = 3087$. Таким образом, определена тотальная функция $f\colon 0..9999 \to 0..9999$. Поразительный факт: если исходное число содержит различные цифры, то не боле чем за семь применений функции f получается число 6174, которое далее периодически повторяется.

1.6.6. Представление функций в программах

Пусть $f\colon A \to B$, множество A конечно и не очень велико: $|A| = n$. Наиболее общим представлением такой функции является массив **array** [A] **of** B, где A — тип данных, значения которого представляют элементы множества A, а B — тип данных, значения которого представляют элементы множества B.

ЗАМЕЧАНИЕ

Если среда программирования допускает массивы только с натуральными индексами, то элементы множества A нумеруются (то есть $A = \{a_1, \ldots, a_n\}$) и функция представляется с помощью массива **array** $[1..n]$ **of** B. Таким образом, последовательности являются частным случаем функций.

Функция нескольких аргументов представляется многомерным массивом.

ОТСТУПЛЕНИЕ

Представление функции с помощью массива является эффективным по времени, поскольку реализация массивов в большинстве случаев обеспечивает вычисление значения функции для заданного значения аргумента (индекса) за постоянное время, не зависящее от размера массива и значения индекса.

Если множество A велико или бесконечно, то для представления функций используется особый вид процедур, возвращающих единственное значение для заданного значения аргумента. Обычно такие процедуры также называются «функциями». В некоторых языках программирования определения функций вводятся ключевым словом **function**. Многоместные функции представляются с помощью нескольких формальных параметров в определении функции. Свойство функциональности обеспечивается *оператором возврата*, часто обозначаемым ключевым словом

return, который прекращает выполнение тела функции и одновременно возвращает значение.

ОТСТУПЛЕНИЕ

В языке программирования Фортран и в некоторых других языках вызов функции и обращение к массиву синтаксически неразличимы, что подчеркивает родственность этих понятий.

1.7. Отношения эквивалентности

Различные встречающиеся на практике отношения могут обладать (или не обладать) теми или иными свойствами. Свойства, введенные в п. 1.4.6, встречаются особенно часто (потому им и даны специальные названия). Более того, оказывается, что некоторые устойчивые комбинации этих свойств встречаются настолько часто, что заслуживают отдельного названия и специального изучения. Здесь рассматриваются классы отношений, обладающих определённым набором свойств. Такое «абстрактное» изучение классов отношений обладает тем преимуществом, что, один раз установив некоторые следствия из наличия у отношения определённого набора свойств, далее эти следствия можно автоматически распространить на все конкретные отношения, которые обладают данным набором свойств. Рассмотрение начинается отношениями эквивалентности (в этом разделе) и продолжается отношениями порядка (в следующем разделе).

1.7.1. Классы эквивалентности

Рефлексивное симметричное транзитивное отношение называется *отношением эквивалентности*. Обычно отношение эквивалентности обозначают знаком $\equiv$.

Примеры

1. Отношения равенства чисел, равенства множеств.
2. Отношение равномощности множеств.
3. Отношение между мультимножествами «иметь одинаковый состав».
4. Отношения между конечными последовательностями: «быть последовательностями над одним и тем же мультимножеством» и «быть последовательностями над мультимножествами одного состава». При этом первое из этих отношений является собственным подмножеством второго.

Пусть $\equiv$ — отношение эквивалентности на множестве M, и $x \in M$. Подмножество элементов множества M, эквивалентных x, называется *классом эквивалентности* для x: $[x]_{\equiv} \stackrel{\text{Def}}{=} \{y \in M \mid y \equiv x\}$.

Если отношение подразумевается, то нижний индекс у обозначения класса эквивалентности (знак отношения) обычно опускают.

ЛЕММА 1. $\forall\, a \in M\ ([a] \neq \varnothing)$.

Доказательство. $a \equiv a \Longrightarrow a \in [a]$. □

ЛЕММА 2. $a \equiv b \Longrightarrow [a] = [b]$.

Доказательство. Если $a \equiv b$, то $x \in [a] \Longleftrightarrow x \equiv a \Longleftrightarrow x \equiv b \Longleftrightarrow x \in [b]$. □

ЛЕММА 3. $a \not\equiv b \Longrightarrow [a] \cap [b] = \varnothing$.

Доказательство. От противного: $[a] \cap [b] \neq \varnothing \Longrightarrow \exists c \in M \ (c \in [a] \cap [b]) \Longrightarrow$ $\Longrightarrow c \in [a]\ \&\ c \in [b] \Longrightarrow c \equiv a\ \&\ c \equiv b \Longrightarrow a \equiv c\ \&\ c \equiv b \Longrightarrow a \equiv b$, противоречие. □

ТЕОРЕМА. *Если $\equiv$ — отношение эквивалентности на множестве M, то классы эквивалентности по этому отношению образуют разбиение множества M, причём среди элементов разбиения нет пустых. И обратно, всякое разбиение $\mathcal{B} = \{B_i\}$ множества M, не содержащее пустых элементов, определяет отношение эквивалентности на множестве M, классами эквивалентности которого являются элементы разбиения.*

Доказательство.

[$\Longrightarrow$] Требуемое разбиение строится следующим алгоритмом.

Вход: множество M, отношение эквивалентности $\equiv$ на M.

Выход: разбиение $\mathcal{B}$ множества M на классы эквивалентности.

```
B := ∅  // вначале разбиение пусто
for a ∈ M do
  for B ∈ B do
    select b ∈ B  // берём представителя из B
    if a ≡ b then
      B := B + a  // пополняем класс эквивалентности
      next for a  // переходим к новому элементу из M
    end if
  end for
  B := B ∪ {{a}}  // вводим новый класс эквивалентности
end for
```

[$\Longleftarrow$] Положим $a \equiv b := \exists i \ (a \in B_i\ \&\ b \in B_i)$. Тогда отношение $\equiv$ рефлексивно, поскольку $M = \bigcup B_i \Longrightarrow \forall a \in M \ (\exists i \ (a \in B_i))$ и $a \in B_i \Longrightarrow a \equiv a$. Отношение $\equiv$ симметрично, поскольку $a \equiv b \Longrightarrow \exists i \ (b \in B_i\ \&\ a \in B_i) \Longrightarrow b \equiv a$. Отношение $\equiv$ транзитивно, поскольку $a \equiv b\ \&\ b \equiv c \Longrightarrow [a] = [b]\ \&\ [b] = [c] \Longrightarrow [a] = [c] \Longrightarrow a \equiv c$. □

Пусть $\mathcal{B}_1, \mathcal{B}_2$ — два разбиения множества M. Говорят, что разбиение $\mathcal{B}_1$ есть измельчение разбиения $\mathcal{B}_2$, если каждый элемент разбиения $\mathcal{B}_2$ является объединением некоторых элементов разбиения $\mathcal{B}_1$.

Пример. На рис. 1.14 приведён график отношения $\{(i,j) \mid i \in 1..6\ \&\ j \in 1..6\ \&\ (i+j) \bmod 2 = 1\}$. Далее приведены два разбиения: по столбцам и по строкам — и справа приведено их общее измельчение.

Измельчение разбиений порождает измельчение отношений эквивалентности. Предельным случаем измельчения отношения эквивалентности является *отношение равенства*, когда каждый элемент эквивалентен (равен) только самому себе.

Примеры

1. Упорядоченную пару точек (*направленный отрезок*) часто называют *вектором* (в двухмерном или трехмерном евклидовом пространстве). Говорят, что векторы *коллинеарны*, если они лежат на параллельных прямых или на одной прямой.

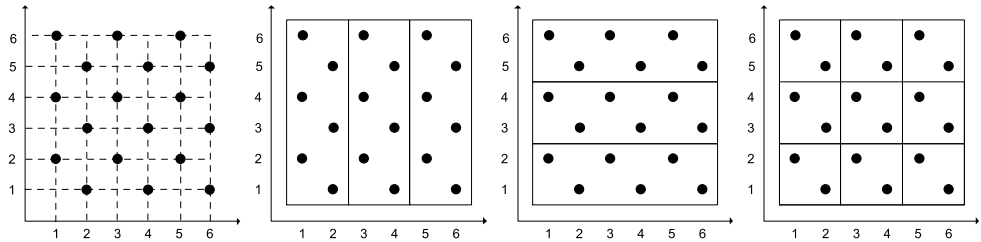


Рис. 1.14. График отношения, два его разбиения и их общее измельчение

Легко видеть, что коллениарность векторов является отношением эквивалентности. Тогда классы отношения эквивалентности состоят из всех коллинеарных векторов. Эти классы можно измельчить, различая сонаправленные и противонаправленные векторы.

2. Отношение между конечными последовательностями «быть последовательностями над одним и тем же мультимножеством» является измельчением отношения «быть последовательностями над мультимножествами одного состава».

1.7.2. Ядро функционального отношения

Всякая функция f, будучи отношением, имеет ядро $f \circ f^{-1}$, которое является отношением на области определения функции.

ЗАМЕЧАНИЕ

Даже если f — функция, f^{-1} отнюдь не обязательно функция, поэтому здесь $\circ$ — знак композиции отношений, а не суперпозиции функций.

Термин «ядро» применительно к функции f и обозначение $\ker f$ обычно резервируются для других целей, поэтому в формулировке следующей теоремы вместо слова «функция» употреблён несколько тяжеловесный оборот «функциональное отношение».

ТЕОРЕМА. *Ядро функционального отношения является отношением эквивалентности на множестве определения.*

Доказательство. Пусть $f\colon A \to B$. Достаточно рассматривать случай, когда $\mathrm{Dom}\, f = A$ и $\mathrm{Im}\, f = B$.

[Рефлексивность] Пусть $a \in A$. Тогда $\exists b \in B \ \ (b = f(a))$ и $a \in f^{-1}(b)$, то есть $(a,b) \in f \ \&\ (b,a) \in f^{-1}$ и значит $(a,a) \in f \circ f^{-1}$.

[Симметричность] Пусть $(a_1,a_2) \in f \circ f^{-1}$. Тогда $\exists b \ \big(b = f(a_1) \ \&\ a_2 \in f^{-1}(b)\big)$, $a_1 \in f^{-1}(b) \ \&\ b = f(a_2)$ и $b = f(a_2) \ \&\ a_1 \in f^{-1}(b)$, откуда $(a_2,a_1) \in f \circ f^{-1}$.

[Транзитивность] Пусть $(a_1,a_2) \in f \circ f^{-1}$ и $(a_2,a_3) \in f \circ f^{-1}$. Это означает, что $\exists b_1 \in B \ \big(b_1 = f(a_1) \ \&\ a_2 \in f^{-1}(b_1)\big)$ и $\exists b_2 \in B \ \big(b_2 = f(a_2) \ \&\ a_3 \in f^{-1}(b_2)\big)$. Тогда $b_1 = f(a_1) \ \&\ b_1 = f(a_2) \ \&\ b_2 = f(a_2) \ \&\ b_2 = f(a_3) \implies b_1 = b_2$. Положим $b := b_1$ (или $b := b_2$). Тогда $b = f(a_1) \ \&\ b = f(a_3) \implies b = f(a_1) \ \&\ a_3 \in f^{-1}(b)$, откуда $(a_1,a_3) \in f \circ f^{-1}$. □

СЛЕДСТВИЕ. *Ядром биекции является тождественное отношение.*

Пример. На рис. 1.15 слева представлен график функционального отношения $\{(x, y) \in 1..3 \times 1..3 \mid y - 1 = (x - 2)^2\}$, а справа представлен график ядра этого отношения. Классы эквивалентности отмечены заливкой точек графика.

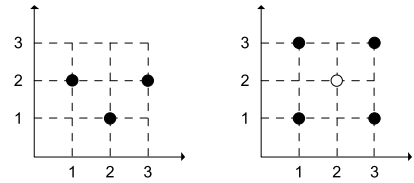


Рис. 1.15. Функциональное отношение и его ядро

Пример. На рис. 1.16 слева представлен график биективного отношения $\{(x, y) \in 1..3 \times 1..3 \mid x + y = 4\}$, а справа представлен график ядра этого отношения. Классы эквивалентности отмечены заливкой точек графика.

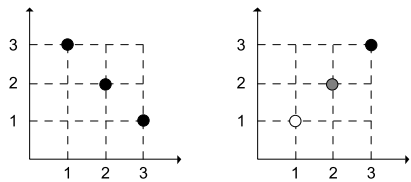


Рис. 1.16. Биективное отношение и его ядро

1.7.3. Фактормножество

Если R — отношение эквивалентности на множестве M, то множество классов эквивалентности называется *фактормножеством* множества M относительно эквивалентности R и обозначается M/R, то есть $M/R \stackrel{\text{Def}}{=} \{[x]_R\}_{x \in M}$. Фактормножество является подмножеством булеана: $M/R \subset 2^M$. Ясно, что фактормножество по ядру функционального отношения равномощно множеству значений:

$$|\operatorname{Dom} f/\left(f \circ f^{-1}\right)| = |\operatorname{Im} f|.$$

Пример. Рассмотрим множество $A = \{1, 2, 3, 4\}$ и отношение эквивалентности $R = \{(1,1), (2,2), (3,3), (4,4), (1,2), (2,1), (3,4), (4,3)\}$. Отношение R и фактормножество по этому отношению изображены на рис. 1.17. Элементы исходного отношения обозначены закрашенными кружками, а элементы фактормножества — незакрашенными квадратами.

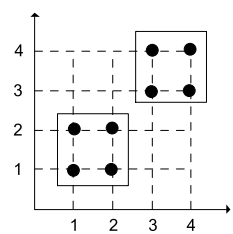


Рис. 1.17. Отношение эквивалентности и фактормножество

Функция $\operatorname{nat} R\colon M \to M/R$ называется *отождествлением* (или *натуральным отображением*, или *канонической проекцией*) и определяется следующим образом:

$$\operatorname{nat} R(x) \stackrel{\text{Def}}{=} [x]_R.$$

Использование фактормножества вместо множества называется *факторизацией*. Пусть задана функция $f\colon A \to B$. Тогда $\ker f$ — это эквивалентность, которая определяет фактормножество $A/\ker f$ с элементами $[x]_{\ker f}$ и функцию отождествления $\operatorname{nat}\colon A \to A/\ker f$. Определим *индуцированную функцию* из фактормножества в множество B: $f/\ker f := \{(s, b) \mid s = [x]_{\ker f}\ \&\ b = f(x)\}$. Тогда исходная функция может быть *факторизована*, или *разложена*, в суперпозицию

$$f = (\operatorname{nat} f) \circ (f/\ker f).$$

Заметим, что при этом сначала применяется отождествление, которое сюръективно, а потом индуцированная функция, которая инъективна.

Пример. Пусть A — множество студентов, а B — множество проводимых занятий (в данный момент времени). Отношение «посещаемости» между студентами и занятиями функционально: студент может посещать (или прогуливать) только одно занятие. Значит, ядро этого функционального отношения является эквивалентностью. Классы эквивалентности по отношению посещаемости называются группами. Студентов одной группы объединяет общий учебный план. При составлении расписания занятий обычно применяется факторизация по группам (а также по курсам и факультетам). На стене вывешивается таблица с расписанием, которая представляет индуцированную функцию. Функцию отождествления студент должен выполнить сам, и таким образом, удается компактно представить довольно сложное отношение (количество элементов в исходных множествах исчисляется тысячами).

1.7.4. Множества уровня

Пусть $f\colon A \to B$ — функция и $f \circ f^{-1}$ — отношение эквивалентности на $\operatorname{Dom} f$. Рассмотрим фактормножество $\operatorname{Dom} f / \left(f \circ f^{-1}\right)$. Класс эквивалентности (элемент фактормножества) — это подмножество элементов A, которые отображаются в один и тот же элемент $b \in B$. Такие множества называются *множествами уровня* функции f. Напомним, что $|\operatorname{Dom} f / \left(f \circ f^{-1}\right)| = |\operatorname{Im} f|$.

Неформально множество уровня функции f, соответствующее значению c, — это множество решений уравнения $f(x) = c$.

Пример. Множество уровня 0 для функции $\sin(x)$ — это $\pi\mathbb{Z}$.

ОТСТУПЛЕНИЕ

Термин «множество уровня» опирается на очевидные геометрические ассоциации. Если нарисовать график функции одной переменной в обычной декартовой системе координат и провести горизонтальную прямую на определённом уровне, то абсциссы точек пересечения с графиком функции дадут множество уровня функции.

Пример. Пусть задано множество M; $|M| = n$. Рассмотрим функциональное отношение (мощность) P между булеаном 2^M и множеством неотрицательных целых чисел: $P \subset 2^M \times 0..n$, где $P := \{(X, k) \mid X \subset M \;\&\; k \in 0..n \;\&\; |X| = k\}$. Ядром является отношение равномощности, а множествами уровня — семейства равномощных подмножеств.

1.7.5. Толерантность

Рефлексивное симметричное отношение называется *толерантностью*. Отношение эквивалентности является частным случаем отношения толерантности. В отличие от отношения эквивалентности, дающего разбиение множества элементов, на котором оно определено, отношение толерантности даёт покрытие этого множества. Действительно, пусть $T \subset A^2$ — отношение толерантности. Определим *класс толерантности* для элемента x по отношению толерантности T как множество $\{y \in A \mid x\,T\,y\}$. Тогда

семейство классов толерантности образует покрытие множества A. На содержательном уровне толерантность означает следующее: любой объект толерантен самому себе (рефлексивность), а толерантность двух объектов не зависит от того, в каком порядке они рассматриваются (симметричность). Однако, если один объект толерантен с другим, а этот другой — с третьим, то это вовсе не значит, что все три объекта толерантны между собой.

Примеры

1. Отношение знакомства на множестве людей. Иванов может быть знаком с Петровым, Петров — с Сидоровым, но при этом Иванов и Сидоров могут не быть знакомы между собой.
2. Отношение похожести между поисковыми запросами в системах информационного поиска.
3. Отношение на множестве слов, которое задаётся как наличие хотя бы одной общей буквы. В этом отношении находятся, например, пересекающиеся слова кроссворда.

1.7.6. Гомоморфизмы и изоморфизмы

Пусть R — n-местное отношение на множестве X, причём $R \subset X^n$; а S — n-местное отношение на множестве Y, причём $S \subset Y^n$, $n > 1$. Тогда функция $f\colon X \to Y$ называется *гомоморфизмом*, если

$$\forall\, x_1, \ldots, x_n \in X \;\; (R(x_1, \ldots, x_n) = S(f(x_1), \ldots, f(x_n))) .$$

Пример. Пусть R — некоторое отношение эквивалентности на множестве A, а S — тождественное отношение на фактор-множестве A/R. Тогда функция отождествления $\operatorname{nat} R\colon A \to A/R$ — это гомоморфизм.

ЗАМЕЧАНИЕ

Образно говорят, что гомоморфизм из множества в множество «уважает» соответствующие отношения на этих множествах.

Функция может быть гомоморфизмом и уважать одни отношения на множествах и при этом не уважать другие отношения на этих же множествах.

Пример. Рассмотрим функцию $n \operatorname{\mathbf{div}} 2 + 1$ на множестве натуральных чисел. Эта функция уважает отношение порядка $\leqslant$ на множестве натуральных чисел, поскольку $n \leqslant m \iff (n \operatorname{\mathbf{div}} 2 + 1) \leqslant (m \operatorname{\mathbf{div}} 2 + 1)$, но не уважает отношение неравенства $\neq$, поскольку $4 \neq 5$, но $4 \operatorname{\mathbf{div}} 2 + 1 = 3 = 5 \operatorname{\mathbf{div}} 2 + 1$.

Биективный гомоморфизм называется *изоморфизмом*. Любая биекция $f\colon X \to Y$ уважает тождественные отношения $\overset{X}{=}$ и $\overset{Y}{=}$ на множествах X и Y:

$$x_1 \overset{X}{=} x_2 \iff f(x_1) \overset{Y}{=} f(x_2).$$

Пример. Взаимно-однозначное соответствие между множествами является изоморфизмом. Равномощные множества изоморфны относительно тождественных отношений.

Пусть X и Y — множества с определёнными на них n-местными отношениями R и S соответственно, и $f\colon X \to Y$ — изоморфизм.

ТЕОРЕМА 1. *Если $f\colon X \to Y$ — изоморфизм, то $f^{-1}\colon Y \to X$ — тоже изоморфизм.*

Доказательство. Пусть $x_1, \ldots, x_n \in X$. Обозначим $y_1 := f(x_1), \ldots, y_n := f(x_n)$, где $y_1, \ldots, y_n \in Y$. Поскольку f — биекция, имеем $x_1 = f^{-1}(y_1), \ldots, x_n = f^{-1}(y_n)$. Тогда: $S(y_1, \ldots, y_n) = S(f(x_1), \ldots, f(x_n)) = R(x_1, \ldots, x_n) = R(f^{-1}(y_1), \ldots, f^{-1}(y_n))$. □

Если $f\colon X \to Y$ — изоморфизм, то множества X и Y называют *изоморфными* и этот факт обозначают так: $X \overset{f}{\sim} Y$. Если f ясно из контекста или просто неважно в конкретном рассуждении, то пишут $X \sim Y$.

ЗАМЕЧАНИЕ

Не всякая биекция является изоморфизмом.

Пример. Рассмотрим функцию изменения знака $f\colon \mathbb{Z} \to \mathbb{Z}$, $f(x) := -x$. Это биекция, которая не уважает, например, отношение порядка $<$.

Даже если множества равномощны, $|X| = |Y|$, отношения $R \subset X^n$ и $S \subset Y^n$ могут быть таковы, что не существует изоморфизма, уважающего эти отношения.

Пример. Пусть $|X| = |Y|$. Положим $R := \overset{X}{=}$, $S := \overset{Y}{\neq}$.
Тогда $\forall\, f\colon X \to Y \;\; (x_1 = x_2 \Longrightarrow \neg\,(f(x_1) \neq f(x_2)))$.

В то же время, если задано n-местное отошение R на множестве X и биекция $f\colon X \to Y$, то на множестве Y всегда возможно задать n-местное отношение S так, чтобы $X \sim Y$. Для этого достаточно определить

$$S := \left\{(y_1, \ldots, y_n) \in Y^n \mid R(f^{-1}(y_1), \ldots, f^{-1}(y_n))\right\}.$$

Таким образом, если задано семейство множеств $\mathfrak{X} = \{X_1, \ldots, X_n\}$, и на каждом множестве $X_i \in \mathfrak{X}$ задано n-местное отношение $R_i \subset X_i^n$, то некоторые пары множеств могут оказаться изоморфными: $X_i \sim X_j$, а другие — нет. Всё зависит от заданных отношений. Семейство пар изоморфных множеств образует отношение *изоморфности* на множестве $\mathfrak{X}$.

ЗАМЕЧАНИЕ

Неравномощные множества не могут быть изоморфны.

ТЕОРЕМА 2. *Изоморфность является отношением эквивалентности.*

Доказательство.

[Рефлексивность] Функция $x \mapsto x$ — требуемый изоморфизм.

[Симметричность] Следует из предыдущей теоремы.

[Транзитивность] Пусть X, Y и Z — три множества с заданными на них n-местными отношениями R_x, R_y и R_z соответственно, $f\colon X \to Y$ и $g\colon Y \to Z$ — изоморфизмы. Тогда $R_x(x_1, \ldots, x_n) = R_y(f(x_1), \ldots, f(x_n)) = R_z(g(f(x_1)), \ldots, g(f(x_n)))$. □

Всякий гомоморфизм индуцирует связанный изоморфизм.

ТЕОРЕМА 3 (О гомоморфизме). *Если $f\colon X \to Y$ — гомоморфизм, то*

$$\operatorname{Dom} f / \ker f \sim \operatorname{Im} f.$$

Доказательство.

[Биекция] Рассмотрим индуцированную функцию $g = f/\ker f\colon \operatorname{Dom} f/\ker f \to Y$. Функция g инъективна по построению. Функция $h\colon \operatorname{Dom} f/\ker f \to \operatorname{Im} f$, полученная из g сокращением области прибытия, сюръективна, при этом так же, как и g, инъективна, а значит, биективна.

[Гомоморфизм] Пусть R — исходное n-местное отношение на множестве X. Оно индуцирует новое n-местное отношение R_f на множестве $\operatorname{Dom} f/\ker f$ следующим образом: $R_f([x_1],\ldots,[x_n]) := R(x_1,\ldots,x_n)$. Представим функцию f в виде $f = (\operatorname{nat} f)\circ(f/\ker f) = (\operatorname{nat} f)\circ h$. Имеем: $R_f([x_1],\ldots,[x_n]) = R(x_1,\ldots,x_n) = S(f(x_1),\ldots,f(x_n)) = S(h([x_1]),\ldots,h([x_n])$. □

Пример. Рассмотрим множество слов A^+, заданных над алфавитом A, и отношение равенства длин слов на нём. Также рассмотрим множество S конечных мультимножеств букв из алфавита A и отношение равномощности на нём. Пусть $f\colon A^+ \to S$ — функция, отображающая слово в мультимножество составляющих его букв. Она является гомоморфизмом относительно описанных отношений. Тогда по теореме о гомоморфизме множество всех множеств слов, составленных из одних и тех же букв, изоморфно мультимножеству этих букв.

ЗАМЕЧАНИЕ

Мощности классов эквивалентности по ядру гомоморфизма в некотором смысле характеризуют его отклонение от инъективности. Если все мощности равны 1, то гомоморфизм инъективен.

1.8. Отношения порядка

В этом разделе определяются и описываются различные варианты отношений порядка. Отношение порядка позволяет *сравнивать* между собой различные элементы одного множества. Фактически, интуитивные представления об отношениях порядка можно считать общеизвестными, и такие представления в этом учебнике уже использованы при описании работы с упорядоченными списками. Однако при построении математических моделей интуитивных представлений может оказаться недостаточно, и необходимо более детальное описание свойств отношений порядка.

1.8.1. Предпорядок

Рефлексивное транзитивное отношение называется отношением *предпорядка* (иногда используют термин *квазипорядок*). Отношение предпорядка обычно обозначают знаком $\sqsubseteq$. Отношение предпорядка, с одной стороны, является обобщением эквивалентности, а с другой стороны, является обобщением нестрого частичного порядка (см. следующий параграф). Предпорядок, будучи транзитивным, совпадает со своим транзитивным замыканием: $\sqsubseteq^+ = \sqsubseteq$.

ТЕОРЕМА. *Если R — отношение предпорядка, то* $\forall\, n \geqslant 1\ \big(\forall k > 0\ \big((R^{n+k} \sqsubseteq R^n\big)\big)$.

Доказательство. По теореме 1.4.6 $R^2 \subset R$, и значит, $R^3 = R^2 \circ R \subset R \circ R = R^2$, и так далее, имеем $\forall\, n > 0 \ \left(R^{n+1} \subset R^n\right)$. Откуда по транзитивности отношения $\sqsubseteq$ имеем $\forall\, n \geqslant 1 \ \left(\forall\, k > 0 \ \left((R^{n+k} \sqsubseteq R^n\right)\right)$. □

Следствие. *Если R — отношение предпорядка, то $R^* = R^+ = R$.*

Доказательство. По формулам параграфа 1.5.1. □

Антирефлексивное транзитивное отношение называется отношением *строго предпорядка* и обозначается знаком $\sqsubset$.

Пример. В «Повести временных лет» сказано: «И были три брата: один по имени Кий, другой — Щек и третий — Хорив, а сестра их была Лыбедь». Отношения «x брат y» и «x сестра y» являются нестрогими предпорядками. На рис. 1.18 показаны диаграммы Хассе транзитивных редукций этих отношений.

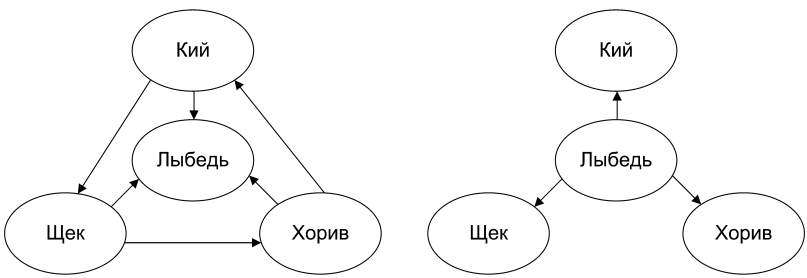


Рис. 1.18. Диаграммы Хассе транзитивных редукций отношений братства и сестринства

1.8.2. Частичный и линейный порядок

Антисимметричное транзитивное отношение называется отношением *порядка*. Отношение порядка может быть рефлексивным, и тогда оно называется отношением *нестрогого порядка*. Отношение порядка может быть антирефлексивным, и тогда оно называется отношением *строгого порядка*. Отношение порядка может быть линейным, и тогда оно называется отношением *линейного порядка*. Отношение порядка может не обладать свойством линейности, и тогда оно называется отношением *частичного порядка*. Обычно отношение строгого порядка (линейного или частичного) обозначается знаком $\prec$, а отношение нестрогого порядка — знаком $\preceq$.

Замечание

Для строгого порядка свойство антисимметричности можно определить следующим образом: $\forall\, a, b \ \ (a \prec b \iff \neg(b \prec a)) \iff \forall\, a, b \ \ (\neg(a \prec b) \vee \neg(b \prec a)) \iff \forall\, a, b \ \ (\neg(a \prec b \ \&\ b \prec a))$.

Примеры

Отношение $<$ на множестве вещественных чисел является отношением строгого линейного порядка. Отношение $\leqslant$ на множестве вещественных чисел является отношением нестрогого линейного порядка. Отношение $\subseteq$ на булеане 2^M — нестрогий частичный порядок, а отношение $\subsetneq$ — строгий частичный порядок.

Множество, на котором задано отношение частичного порядка, называется *частично упорядоченным*. Множество, на котором задано отношение линейного порядка, называется *линейно упорядоченным*.

Пример. Множество вещественных чисел упорядочено линейно, а булеан упорядочен частично.

ТЕОРЕМА 1. *Если $\prec$ — отношение частичного порядка, то обратное отношение $\succ$ также является отношением частичного порядка.*

Доказательство.

[Антисимметричность] $\forall\, a, b\ \ ((a \prec b)\ \&\ (b \prec a) \Longrightarrow a = b) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, b\ \ ((b \prec a)\ \&\ (a \prec b) \Longrightarrow a = b) \Longleftrightarrow \forall\, a, b\ \ ((a \succ b)\ \&\ (b \succ a) \Longrightarrow a = b).$

[Транзитивность] $\forall\, a, b, c\ \ ((a \prec b)\ \&\ (b \prec c) \Longrightarrow a \prec c) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, b, c\ \ ((b \succ a)\ \&\ (c \succ b) \Longrightarrow c \succ a) \Longleftrightarrow \forall\, a, b, c\ \ ((c \succ b)\ \&\ (b \succ a) \Longrightarrow c \succ a).$ □

СЛЕДСТВИЕ. *Если $\prec$ является отношением частичного порядка, то отношения $\prec \setminus I$ и $\succ \setminus I$ являются отношениями строгого частичного порядка, и отношения $\prec \cup I$ и $\succ \cup I$ являются отношениями нестрогого частичного порядка.*

Доказательство. По теореме п. 1.4.6. □

ЛЕММА. *Отношение R антисимметрично и линейно тогда и только тогда, когда дополнение отношение $\overline{R}$ совпадает с обратным отношением R^{-1} везде, за исключением диагонали.*

Доказательство. По условиям леммы, исключим диагональ, то есть рассмотрим пары *различных* элементов a и b.

[Необходимость] В указанных условиях свойство антисимметричности гласит: $\forall\, a, b\ \ (\neg\,(aRb\ \&\ bRa))$, а свойство линейности гласит: $\forall\, a, b\ \ (aRb \vee bRa)$, так что из двух пар (a, b) и (b, a) в точности одна принадлежит отношению R, а другая пара принадлежит как дополнению, так и обратному отношению.

[Достаточность] Если дополнение отношения R совпадает с обратным отношением, то это означает, что $\forall\, a, b\ \ (\neg(aRb) \Longleftrightarrow bRa)$, то есть отношение R антисимметрично. Линейность докажем от противного. Пусть $\exists\, a, b\ \ (\neg(aRb \vee bRa))$. Тогда $\exists\, a, b\ \ (\neg aRb\ \&\ \neg bRa)$, и по уже доказанному, $\exists\, a, b\ \ (bRa\ \&\ aRb)$, что противоречит антисимметричности. □

СЛЕДСТВИЕ. *Если отношение R антисимметрично и линейно, то* $\forall\, a, b\ \ (a \neq b \Longrightarrow (aRb \Longleftrightarrow \neg(bRa)))$.

ТЕОРЕМА 2. *Если $\prec$ является отношением линейного порядка, то его дополнение $\not\prec$ также является отношением линейного порядка.*

Доказательство. По лемме $\forall\, a, b\ \ (aRb \Longleftrightarrow \neg(bRa))$.

[Линейность] $\forall\, a, b\ \ (a = b \vee a \prec b \vee b \prec a) \Longleftrightarrow \forall\, a, b\ \ (a = b \vee b \not\prec a \vee a \not\prec b) \Longleftrightarrow$
$\Longleftrightarrow \forall\, a, b\ \ (a = b \vee a \not\prec b \vee b \not\prec a).$

[Антисимметричность] $\forall a, b\ (a \prec b\ \&\ b \prec a \Longrightarrow a = b) \Longleftrightarrow$
$\Longleftrightarrow \forall a, b\ (b \not\succ a\ \&\ a \not\succ b \Longrightarrow a = b) \Longleftrightarrow \forall a, b\ (a \not\succ b\ \&\ b \not\succ a \Longrightarrow a = b).$
[Транзитивность] $\forall a, b, c\ (c \prec b\ \&\ b \prec a \Longrightarrow c \prec a) \Longleftrightarrow$
$\Longleftrightarrow \forall a, b, c\ (b \not\succ c\ \&\ a \not\succ b \Longrightarrow a \not\succ c) \Longleftrightarrow \forall a, b, c\ (a \not\succ b\ \&\ b \not\succ c \Longrightarrow a \not\succ c).$ □

СЛЕДСТВИЕ. *Если $\preceq$ является отношением нестрогого линейного порядка, то дополнительное отношение $\succ$ является отношением строгого линейного порядка. Обратно, если $\prec$ является отношением строгого линейного порядка, то дополнительное отношение $\succeq$ является отношением нестрогого линейного порядка.*

Доказательство. Достаточно проверить, куда входит диагональ. □

Пример. Отношение $<$ является отношением строгого линейного порядка, а его дополнение $\geqslant$ является отношением нестрогого линейного порядка на *любом* подмножестве *числовой оси*, то есть любом подмножестве (конечном или бесконечном) множества чисел — целых, рациональных и вещественных.

ЗАМЕЧАНИЕ

Линейность существенна: дополнение частичного порядка не обязательно частичный порядок.

Пример. Отношение $\subseteq$ на булеане является отношением нестрогого частичного порядка, но его дополнение не является отношением строгого частичного порядка. Более того, оно вообще не является отношением порядка, поскольку не антисимметрично и не транзитивно. На рис. 1.19 показаны графики отношения $\subseteq$ и его дополнения на булеане $2^{\{1,2,3\}}$.

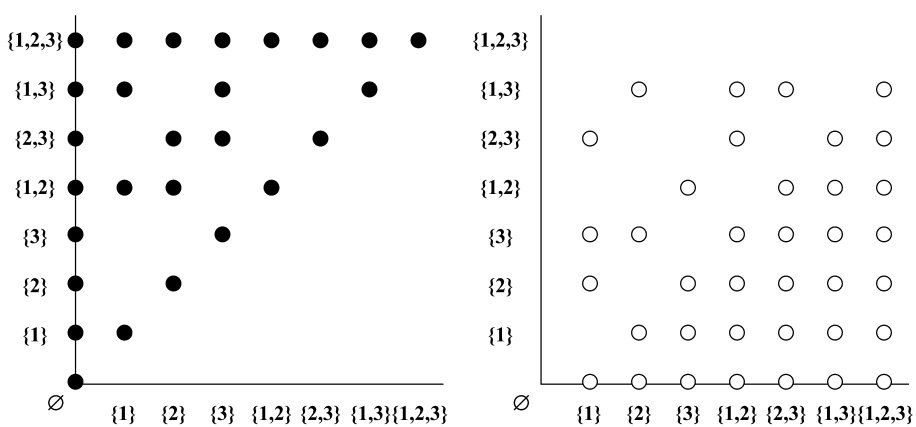


Рис. 1.19. Графики отношения $\subseteq$ и его дополнения на булеане $2^{\{1,2,3\}}$

Даже если частично упорядоченное множество не является линейно упорядоченным, в нем могут найтись линейно упорядоченные подмножества.

Пример. Семейство подмножеств $\{C_i\}$ множества A называется *цепочкой* в A, если $\forall\, i\ (C_i \subset C_{i+1}\ \&\ C_i \neq C_{i+1})$. Любая цепочка в A образует линейно упорядоченное подмножество частично упорядоченного множества 2^A.

Линейно упорядоченное подмножество $\{a_1, \ldots, a_n, \ldots\}$ можно рассматривать как последовательность $\langle a_1, \ldots, a_n, \ldots\rangle$, в которой $\forall\, i\ (a_i \prec a_{i+1}\ \&\ a_i \neq a_{i+1})$. Такие последовательности называются *строго монотонно возрастающими*, или просто *возрастающими*. Последовательности могут быть конечными или бесконечными. Если последовательность конечная, количество элементов в ней называется *длиной*.

Говорят, что частично упорядоченное множество имеет конечную *высоту* k, если любая возрастающая последовательность имеет длину не более k.

1.8.3. Минимальные и наименьшие элементы

Элемент x множества M с отношением порядка $\prec$ называется *минимальным*, если в множестве M не существует элементов, меньших, чем элемент x:
$\neg\exists\, y \in M\ \ (y \prec x\ \&\ y \neq x)$, иначе говоря, $\forall\, y \in M\ \ (\neg(y \prec x)\ \vee\ y = x)$.

Пример. Пустое множество $\varnothing$ является минимальным элементом булеана по включению.

ТЕОРЕМА 1. *Во всяком конечном непустом частично упорядоченном множестве существует минимальный элемент.*

Доказательство. От противного. Пусть $\neg(\exists x \in M\ \ (\neg\exists\, y \in M\ \ (y \prec x\ \&\ y \neq x)))$. Тогда $\forall x \in M\ \ (\exists\, y \in M\ \ (y \prec x)) \Longrightarrow \exists\, (u_i)_{i=1}^{\infty}\ \ (\forall\, i\ \ (u_{i+1} \prec u_i)\ \&\ u_{i+1} \neq u_i)$. Поскольку $|M| < \infty$, по принципу Дирихле имеем $\exists\, i, j\ \ (i < j\ \&\ u_i = u_j)$. Но по транзитивности $u_i \succ u_{i+1} \succ \cdots \succ u_j \Longrightarrow u_{i+1} \succ u_j = u_i$. Таким образом, $u_{i+1} \prec u_i\ \&\ u_{i+1} \succ u_i \Longrightarrow u_{i+1} = u_i$ — противоречие. □

ЗАМЕЧАНИЕ

Линейно упорядоченное конечное множество содержит ровно один минимальный элемент, а в произвольном конечном частично упорядоченном множестве их может быть несколько.

ТЕОРЕМА 2. *Всякий частичный порядок на конечном множестве может быть дополнен до линейного.*

Доказательство. Алгоритм 1.15. □

ЗАМЕЧАНИЕ

В данном случае слова «может быть дополнен» означают, что существует отношение линейного порядка, которое является надмножеством заданного отношения частичного порядка.

Пусть M — частично упорядоченное множество с отношением порядка $\prec$, а X — его подмножество: $X \subset M$. Элемент a называется *наименьшим* в множестве X (обозначение $a = \min X$), если $a \in X\ \&\ \forall x \in X\ \ (x \neq a \Longrightarrow a \prec x)$.

Очевидно, что наименьший элемент, если он существует, является минимальным. Но элемент может быть минимальным, не будучи наименьшим. Более того, конечное частично упорядоченное множество, имея минимальные элементы, может не иметь наименьшего элемента.

Пример. Пусть в множестве $\{a, b, c, d\}$ задано отношение порядка $\{(a, b), (c, d)\}$, то есть $a \prec b$ и $c \prec d$. Тогда элементы a и c минимальные, но наименьшего элемента не существует.

ТЕОРЕМА 3. *Если существует наименьший элемент, то он является единственным.*

Доказательство. Пусть существуют два наименьших элемента, a и b. Тогда $a \prec b$, поскольку a наименьший, и, аналогично, $b \prec a$, поскольку b также наименьший. По антисимметричности, $a = b$. □

В частично упорядоченном множестве с отношением порядка $\prec$ наряду с минимальными и наименьшими элементами можно определить *максимальные* и *наибольшие* элементы, используя обратное отношение $\succ$.

ЗАМЕЧАНИЕ

Используя теоремы п. 1.8.2, нетрудно показать, что все утверждения относительно минимальных и наименьших элементов имеют место и относительно максимальных и наибольших элементов соответственно, то есть имеет место *двойственность*.

1.8.4. Алгоритм топологической сортировки

Рассмотрим алгоритм дополнения частичного порядка до линейного на конечном множестве.

Алгоритм 1.15. Алгоритм топологической сортировки

Вход: конечное частично упорядоченное множество U.
Выход: линейно упорядоченное множество W.

```
while U ≠ ∅ do
    m := min(U)  // минимальный элемент
    yield m  // включаем элемент m в множество W
    U := U − m  // элемент m более не рассматривается
end while
```

Обоснование. Всякая процедура, генерирующая объекты с помощью оператора **yield**, определяет линейный порядок на множестве своих результатов. Действительно, последовательность, в которой объекты генерируются во время работы процедуры, — это линейный порядок. □

Дополнение частичного порядка до линейного в общем случае не единственно.

Пример. На рис. 1.20 слева показана диаграмма отношения частичного порядка, а справа показаны два дополнения из четырёх возможных дополнений этого порядка до линейного. Добавленные пары выделены.

Используемая в алгоритме функция min возвращает минимальный элемент, существование которого гарантируется теоремой 1.

Вход: конечное частично упорядоченное множество U.
Выход: минимальный элемент m.

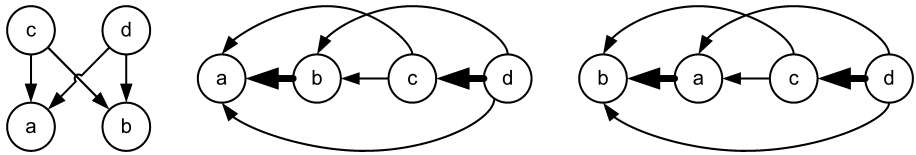


Рис. 1.20. Диаграмма отношения частичного порядка и два дополнения этого порядка до линейного

```
select m ∈ U  // выбираем любой элемент кандидатом в минимальные
for u ∈ U do
  if u ≺ m then
    m := u  // меняем кандидата в минимальные
  end if
end for
return m  // элемент m минимальный
```

Обоснование. Рассмотрим последовательность элементов, которые присваивались m во время работы min. Эта последовательность не может быть бесконечной, и она линейно упорядочена. Рассмотрим последний элемент этой последовательности. Этот элемент m — минимальный. Действительно, от противного, пусть существует элемент u такой, что $u \prec m \;\&\; u \neq m$ и u не входит в последовательность. Тогда по транзитивности u меньше любого члена последовательности, и он был бы включен в последовательность в момент своего рассмотрения. □

ЗАМЕЧАНИЕ

Если отношение порядка представлено матрицей, то функцию min можно реализовать, например, так — найти в матрице отношения первую строку, содержащую только нули.

1.8.5. Верхние и нижние границы

Пусть $X \subset M$ — подмножество упорядоченного множества M с отношением порядка $\prec$. Элемент $m \in M$ называется *нижней границей* для подмножества X, если $\forall x \in X \;\; (m \prec x)$. Элемент $m \in M$ называется *верхней границей* для подмножества X, если $\forall x \in X \;\; (x \prec m)$. Верхние и нижние границы не обязаны существовать для любого множества, и если существуют, то не всегда единственны. Если существует наибольшая нижняя граница, то она называется *инфимумом* и обозначается $\inf(X)$. Если существует наименьшая верхняя граница, то она называется *супремумом* и обозначается $\sup(X)$.

Пример. Рассмотрим множество рациональных чисел $\mathbb{Q}$ с обычным отношением порядка $<$ и его подмножество $X := \{x \in \mathbb{Q} \mid x > 0 \;\&\; x^2 > 2\}$. Тогда $\frac{17}{12} \in X$, а $\frac{17}{13}$ является одной из нижних границ множества X. Инфимума это множество не имеет.

ТЕОРЕМА. *Пусть M — частично упорядоченное множество, а X — любое его подмножество. Тогда:*

1) *если существует наименьший элемент $a = \min X$, то $a = \inf X$;*
2) *если существует инфимум $a = \inf X$ и $a \in X$, то $a = \min X$.*

Доказательство.

[1] $\forall x \in X \ (a \prec x)$, следовательно, a — нижняя граница X. Пусть b — любая нижняя граница X, тогда $b \prec a$, так как $a \in X$. Следовательно, a — наибольшая нижняя граница;

[2] $\forall x \in X \ (a \prec x)$, поскольку a — нижняя граница X, и значит, это наименьший элемент, поскольку $a \in X$. □

СЛЕДСТВИЕ. *Пусть M — частично упорядоченное множество, а X — любое его подмножество. Тогда:*

1) *если существует наибольший элемент $a = \max X$, то $a = \sup X$;*
2) *если существует супремум $a = \sup X$ и $a \in X$, то $a = \max X$.*

Говорят, что частично упорядоченное множество *линейно полно*, если любое его линейно упорядоченное подмножество имеет супремум.

Пример. Если множество X конечно, то 2^X — линейно полно относительно $\subset$.

1.8.6. Монотонные и непрерывные функции

Пусть A и B — упорядоченные множества, и $f\colon A \to B$. Тогда функция f называется *монотонно возрастающей*, если

$$a_1 \prec a_2 \ \& \ a_1 \neq a_2 \Longrightarrow f(a_1) \prec f(a_2) \ \vee \ f(a_1) = f(a_2);$$

монотонно убывающей, если

$$a_1 \prec a_2 \ \& \ a_1 \neq a_2 \Longrightarrow f(a_2) \prec f(a_1) \ \vee \ f(a_1) = f(a_2);$$

строго монотонно возрастающей, если

$$a_1 \prec a_2 \ \& \ a_1 \neq a_2 \Longrightarrow f(a_1) \prec f(a_2) \ \& \ f(a_1) \neq f(a_2);$$

строго монотонно убывающей, если

$$a_1 \prec a_2 \ \& \ a_1 \neq a_2 \Longrightarrow f(a_2) \prec f(a_1) \ \& \ f(a_1) \neq f(a_2).$$

Монотонно возрастающие и убывающие функции называются *монотонными*, а строго монотонно возрастающие и убывающие функции называются *строго монотонными*. Очевидно, что строго монотонная функция монотонна, но обратное неверно.

ЗАМЕЧАНИЕ

Монотонная функция — это гомоморфизм, уважающий порядок.

Примеры

1. Тождественная функция $y = x$ для чисел является строго монотонно возрастающей.
2. Знак числа $\operatorname{sign}(x) \stackrel{\text{Def}}{=}$ **if** $x < 0$ **then** -1 **elseif** $x > 0$ **then** 1 **else** 0 **end if** является монотонно возрастающей функцией.
3. Пусть 2^M — булеан над линейно упорядоченным конечным множеством M, а частичный порядок на булеане — это включение. Тогда функция из алгоритма топологической сортировки, доставляющая минимальный элемент, является монотонно убывающей, но не строго монотонной.

4. Пусть порядок на булеане 2^M — это собственное включение. Тогда функция, которая сопоставляет подмножеству $X \subset M$ его мощность, является строго монотонно возрастающей.
5. Пусть $f\colon M \to M$ — некоторая функция. Тогда функция перехода к образам $F\colon 2^M \to 2^M$ является монотонно возрастающей по включению.

ТЕОРЕМА 1. *Суперпозиция одинаково монотонных функций монотонна в том же смысле.*

Доказательство. Пусть $a \prec b$ и g монотонно возрастает, тогда $g(a) \prec g(b)$. Но f также монотонно возрастает, и значит, $f(g(a)) \prec f(g(b))$. Для других случаев монотонности доказательство аналогично. □

Функция $f\colon M \to M$, где M — частично упорядоченное множество, называется *непрерывной* (по Скотту), если для любой возрастающей последовательности $a_1 \prec \ldots \prec a_n \prec \ldots$, для которой существует супремум, выполнено равенство

$$f(\sup\{a_i\}) = \sup\{f(a_i)\}.$$

ЗАМЕЧАНИЕ

Учитывая теоремы п. 1.8.2 и последнее замечание п. 1.8.3, непрерывность можно определить через строго убывающие и имеющие инфимум последовательности.

ТЕОРЕМА 2. *Непрерывная функция монотонно возрастает.*

Доказательство. Пусть $a \prec b$. Тогда $\sup\{a, b\} = b$. Имеем: $f(b) = f(\sup\{a, b\}) = \sup\{f(a), f(b)\}$. Следовательно, $f(a) \prec f(b)$. □

ТЕОРЕМА 3. *Суперпозиция непрерывных функций непрерывна.*

Доказательство. Пусть строго монотонно возрастающая последовательность $\{a_n\}$ имеет супремум $a = \sup\{a_n\}$. Тогда $f(g(a)) = f(g(\sup\{a_n\})) = f(\sup\{g(a_n)\}) = \sup\{f(g(a_n))\}$. □

1.8.7. Наименьшая неподвижная точка функции

Рассмотрим функцию $f\colon M \to M$. Элемент $x \in M$ называется *неподвижной точкой* функции f, если $f(x) = x$.

ЗАМЕЧАНИЕ

Неподвижная точка функции f — это то же, что и *корень* уравнения $f(x) = x$.

Пример. Пусть $f\colon X \to X$. Рассмотрим функцию перехода к образам $F\colon 2^X \to 2^X$. $F : 2^X \to 2^X$. Тогда множество $A := \cap\{Y \subset X \mid F(Y) \subset Y\}$ (то есть пересечение всех таких подмножеств Y множества X, образы которых являются подмножествами аргументов) является неподвижной точкой функции F. Покажем, что $F(A) \subset A$. Действительно, $F(A) = F(\cap\{Y \subset X \mid F(Y) \subset Y\}) \subset \cap F(Y) = A$. Отсюда по монотонности F имеем $F(F(A)) \subset F(A)$, то есть $F(A)$ удовлетворяет условию для множеств Y, и значит, $A \subset F(A)$. Имеем $A = F(A)$, и значит, A — неподвижная точка функции перехода к образам F.

ЗАМЕЧАНИЕ

Если $A = \bigcap \{Y \subset X \mid F(Y) \subset Y\} = \varnothing$, то $F(\varnothing) = \varnothing$, и пустое множество является неподвижной точкой функции перехода к образам по определению.

Вообще говоря, функция $f\colon X \to X$ может не иметь неподвижных точек, иметь одну неподвижную точку или иметь их несколько, может быть, даже бесконечно много. Если множество X упорядочено, и функция $f\colon X \to X$ имеет неподвижные точки, то среди них можно выделить *наименьшую неподвижную точку*. Однако, даже если функция имеет неподвижные точки, среди них может не быть наименьшей просто потому, что не всякое, даже конечное частично упорядоченное множество имеет наименьший элемент.

Пример. Наименьшей неподвижной точкой для функции перехода к образам $F\colon 2^X \to 2^X$ является множество $A = \cap \{Y \subset X \mid F(Y) \subset Y\}$, введенное в предыдущем примере. Действительно, пусть $\exists B \subset X \ \ (F(B) = B \ \& \ \neg A \subset B)$. Тогда $F(B) \subset B$, и значит, множество B является одним из множеств Y. Получается, что пересечение не является подмножеством одного из пересекаемых множеств — противоречие.

ТЕОРЕМА 4 (о наименьшей неподвижной точке функции)**.** *Если множество X частично упорядочено, линейно полно, имеет наименьший элемент $\mathbf{0} = \min X$, и функция $f\colon X \to X$ непрерывна и тотальна,* $\operatorname{dom} f = X$*, то элемент $a = \sup \{f^n(\mathbf{0}) \mid n \geqslant 0\}$ является наименьшей неподвижной точкой функции f.*

Доказательство. Покажем сначала, что элемент a существует. Поскольку функция f тотальна, элементы $f^n(\mathbf{0})$ определены для любого $n \geqslant 0$. Рассмотрим подмножество $Y := \{f^n(\mathbf{0}) \mid n \geqslant 0\}$. Имеем $f^0(\mathbf{0}) = \mathbf{0} \prec f^1(\mathbf{0})$, так как $\mathbf{0}$ — наименьший элемент. Поскольку функция f непрерывна, она и монотонна, а значит, $f^1(\mathbf{0}) = f(f^0(\mathbf{0})) \prec f^2(\mathbf{0}) = f(f^1(\mathbf{0}))$, $f^2(\mathbf{0}) = f(f^1(\mathbf{0})) \prec f^3(\mathbf{0}) = f(f^2(\mathbf{0}))$ и вообще $f^n(\mathbf{0}) \prec f^{n+1}(\mathbf{0})$ для любого $n \geqslant 0$. Таким образом, подмножество Y линейно упорядочено, а значит, элемент $a = \sup \{f^n(\mathbf{0}) \mid n \geqslant 0\}$ определён по условию теоремы. Покажем теперь, что элемент a действительно является неподвижной точкой функции f, то есть $f(a) = a$. Имеем

$$\begin{aligned} f(a) = f(\sup \{f^n(\mathbf{0}) \mid n \geqslant 0\}) = \sup \{f(f^n(\mathbf{0})) \mid n \geqslant 0\} = \\ = \sup \{f^n(\mathbf{0}) \mid n \geqslant 1\} = \sup\{\mathbf{0}, a\} = a. \end{aligned}$$

Покажем теперь, что a — наименьшая неподвижная точка. Пусть b — любая неподвижная точка. Поскольку $f(b) = b$, имеем $\forall n \geqslant 0 \ \ (f^n(b) = b)$. Но $\mathbf{0} \prec b$, и функция f монотонна, следовательно, $\forall n \geqslant 0 \ \ (f^n(\mathbf{0}) \prec b)$, то есть b — верхняя граница множества $\{f^n(\mathbf{0}) \mid n \geqslant 0\}$, а значит, $a \prec b$. □

ЗАМЕЧАНИЕ

Требование непрерывности в формулировке теоремы нужно только для того, чтобы гарантировать существование супремумов. Вообще говоря, если супремумы заведомо существуют, то от функции требуется только монотонность.

ОТСТУПЛЕНИЕ

Фактически, приведённая теорема о наименьшей неподвижной точке функции служит обоснованием одного из случаев, когда для решения уравнения $x = f(x)$ правомерно применять метод итераций: $x_{i+1} := f(x_i)$.

1.8.8. Вполне упорядоченные множества

Частично упорядоченное множество X называется *вполне упорядоченным*, если любое его непустое подмножество имеет минимальный элемент. В частности, для любых двух элементов $a, b \in X$ один из них обязан быть минимальным в подмножестве $\{a, b\}$, а значит, вполне упорядоченное множество упорядочено линейно.

ЗАМЕЧАНИЕ

Не надо путать вполне упорядоченные множества и множества, на которых определён линейный (или полный) порядок. Линейно упорядоченное множество может не быть вполне упорядоченным.

Примеры

1. Всякое конечное линейно упорядоченное множество вполне упорядочено.
2. Множество натуральных чисел $\mathbb{N}$ с обычным отношением порядка вполне упорядочено.
3. Множество рациональных чисел $\mathbb{Q}$ с обычным отношением порядка не является вполне упорядоченным, так как множество $X := \{x \in \mathbb{Q} \mid x^2 > 2\}$ (равно как и само множество $\mathbb{Q}$) не имеет минимального элемента.
4. Множество вещественных чисел $\mathbb{R}$ с обычным отношением порядка не является вполне упорядоченным, так как множество $X := \{x \in \mathbb{R} \mid x > 0\}$ (равно как и само множество $\mathbb{R}$) не имеет минимального элемента.

Говорят, что два линейно упорядоченных множества A и B *изоморфны* (обозначение $A \sim B$), если между ними существует взаимно-однозначное соответствие, сохраняющее порядок:

$$A \sim B \stackrel{\text{Def}}{=} |A| = |B| \ \& \ \&(\forall\, a_1, a_2 \in A \ (a_1 \prec a_2 \ \& \ a_1 \mapsto b_1 \ \& \ a_2 \mapsto b_2 \Longrightarrow b_1 \prec b_2)).$$

ТЕОРЕМА. *Линейно упорядоченные множества A и B изоморфны тогда и только тогда, когда между ними существует строго монотонное взаимно-однозначное соответствие.*

Доказательство. Пусть A и B — линейно упорядоченные множества с отношениями порядка $<_A$ и $<_B$.

[$\Longrightarrow$] Если $A \sim B$, то $\exists f\colon A \to B \ (a_1 <_A a_2 = f(a_1) <_B f(a_2))$, причём f — биекция, то есть требуемое строго монотонно возрастающее взаимно-однозначное соответствие.

[$\Longleftarrow$] Если $f\colon A \to B$ — строго монотонно возрастающее взаимно-однозначное соответствие, то оно является требуемым изоморфизмом, и $A \sim B$. Если же $f\colon A \to B$ — строго монотонно убывающее взаимно-однозначное соответствие, то достаточно вместо отношения $<_B$ рассмотреть обратное отношение $>_B$. $\square$

ЗАМЕЧАНИЕ

Понятие *изоморфизма* применимо и к вполне упорядоченным множествам, поскольку они упорядочены линейно.

Пример. Вполне упорядоченные множества натуральных чисел и чётных чисел с обычным порядком $<$ изоморфны, поскольку соответствие $n \mapsto 2n$ является строго монотонно возрастающим.

1.8.9. Индукция

Важность вполне упорядоченных множеств определяется тем, что для них можно обобщить принцип *индукции*, применяемый обычно для натуральных чисел.

ТЕОРЕМА 1 (индукция по вполне упорядоченному множеству)**.** *Пусть X – вполне упорядоченное множество, x_0 – его минимальный элемент, а P – некоторый предикат, зависящий от элементов X. Тогда если*

$$P(x_0) \;\&\; \forall\, x_1 \in X \;\; ((\forall\, x \in X \;\; (x \prec x_1 \Longrightarrow P(x))) \Longrightarrow P(x_1))\,,$$

то

$$\forall\, x \in X \;\; (P(x))\,.$$

Доказательство. Обозначим $\overline{P} := \{x \in X \mid \neg P(x)\}$, $\overline{P} \subset X$. Далее от противного. Пусть $\overline{P} \neq \varnothing$. Поскольку X вполне упорядочено, $\overline{P}$ имеет минимальный элемент, обозначим его x_1. Тогда $\forall\, x \in X \;\; (x \prec x_1 \Longrightarrow P(x))$ по выбору x_1, и значит, $P(x_1)$ по условию теоремы, что противоречит выбору x_1. □

ЗАМЕЧАНИЕ

Обычная математическая индукция соответствует индукции по вполне упорядоченному множеству $\mathbb{N}$.

Индукция по вполне упорядоченному множеству сильнее обычного принципа математической индукции в силу следующей замечательной теоремы.

ТЕОРЕМА 2. *Любое множество может быть вполне упорядочено.*

Данное утверждение эквивалентно так называемой *аксиоме выбора* в теории множеств и само может быть принято за аксиому. Мы опускаем доказательство эквивалентности этой теоремы аксиоме выбора.

Пример. Рассмотрим множество положительных рациональных чисел $\mathbb{Q}_+$. Можно показать, что для каждого рационального числа существует единственная дробь $\frac{m}{n}$, где m и n взаимно просты: $\mathbb{Q}_+ \stackrel{\text{Def}}{=} \left\{\frac{m}{n} \mid m, n \in \mathbb{N}\right\}$. Определим отношение $\prec$ на множестве $\mathbb{Q}_+$ следующим образом:
$\frac{m_1}{n_1} \prec \frac{m_2}{n_2} \stackrel{\text{Def}}{=} m_1 < m_2 \vee (m_1 = m_2 \;\&\; n_1 < n_2)$, где $<$ – обычное отношение порядка на $\mathbb{N}$. Нетрудно видеть, что множество $\mathbb{Q}_+$ с отношением $\prec$ является вполне упорядоченным, в то время как с обычным отношением порядка $<$ оно таковым не является.

1.9. Характеристические функции

В математике нередко бывает так, что один и тот же объект описывается различными средствами, на разных языках. Например, в планиметрии окружность определяется как геометрическое место точек, равноудалённых от одной точки, называемой центром окружности, а в аналитической геометрии окружностью называется множество пар (x, y), удовлетворяющих уравнению $(x-a)^2 + (y-b)^2 = r^2$. Речь идёт об одном и том же объекте, обладающем присущими ему свойствами. Но в одном случае используется язык геометрических образов, а в другом — алгебраических формул. При решении геометрических задач на построение с помощью циркуля и линейки удобнее использовать язык планиметрии, а при решении задач на вычисление координат точек пересечения окружностей удобнее использовать язык аналитической геометрии. Подобным образом содержание наивной теории множеств может быть передано разными средствами: через рассмотренное понятие совокупности элементов или через понятие характеристической функции, рассматриваемое в этом разделе. Понятие характеристической функции наталкивает на различные полезные обобщения, наиболее заметным из которых является понятие нечёткого множества.

1.9.1. Классификация характеристических функций

Пусть задано непустое множество U, которое называется множеством элементов, и непустое частично-упорядоченное множество X, которое называется множеством характеристик. Тогда всякая функция $\mu\colon U \to \mathrm{X}$ называется *характеристической функцией*, поскольку она однозначно сопоставляет каждому элементу $u \in U$ характеристику $\chi \in \mathrm{X}$.

Характеристические функции используются в различных областях математики, и при этом применяется различная терминология, в зависимости от природы множества характеристик.

Если $|\mathrm{X}| = 1$, то все элементы имеют одну характеристику, и характеристические функции неинформативны, а потому не используются.

Наиболее употребительными являются следующие четыре случая:

1) $\mathrm{X} = \{0, 1\}$, в этом случае характеристические функции естественно называть *бинарными*;
2) $\mathrm{X} = \{\chi_1, \ldots, \chi_n\}$, такие функции называют *классификаторами*;
3) $\mathrm{X} = \mathbb{N}_0$, такие характеристические функции здесь называются *неотрицательно целочисленными*;
4) $\mathrm{X} = [0; 1]$, в этом случае характеристические функции по традиции называют *нечёткими*.

Наиболее употребительны бинарные характеристические функции.

Пример. Всякому подмножеству A множества U можно взаимно-однозначно сопоставить тотальную функцию $1_A\colon U \to 0..1$, которая определяется так:

$1_A(a) \stackrel{\text{Def}}{=}$ **if** $a \in A$ **then** 1 **else** 0 **end if** и называется *характеристической функцией множества* A.

Элементы множества U могут иметь сложную структуру. В частности, если множество $U = U_1 \times \ldots \times U_n$, то элементами U являются кортежи. Это означает, что характеристические функции (как бинарные, так и иные) не обязаны иметь только один аргумент, а могут иметь несколько аргументов.

Пример. Всякому бинарному отношению R между множествами A и B можно взаимно-однозначно сопоставить тотальную функцию $1_R\colon A \times B \to 0..1$, которая определяется так: $1_R(a,b) \stackrel{\text{Def}}{=}$ **if** aRb **then** 1 **else** 0 **end if** и называется *характеристической функцией отношения* R.

ЗАМЕЧАНИЕ

Бинарные характеристические функции активно применяют в математической логике (глава 4). В этом случае элементы множества X называют *истинностными значениями*, а характеристические функции называют *предикатами*.

В приложениях очень часто используются характеристические функции с конечным набором характеристик. Эти функции называют классификаторами, поскольку они приписывают каждому элементу определённую характеристику, то есть относят элемент к определённому классу.

Пример. Оси координат разбивают плоскость на четыре *квадранта*. Квадранты по традиции нумеруются римскими цифрами. Для каждой точки плоскости можно определить, к какому квадранту она относится, то есть задать классификатор на $\mathbb{R}^2$, по следующим правилам:

1) **if** $x>0, y>0$ **then** $\mu(x,y) := \text{I}$ **end if**; 2) **if** $x<0, y>0$ **then** $\mu(x,y) := \text{II}$ **end if**;
3) **if** $x<0, y<0$ **then** $\mu(x,y) := \text{III}$ **end if**; 4) **if** $x>0, y<0$ **then** $\mu(x,y) := \text{IV}$ **end if**.

ЗАМЕЧАНИЕ

В предыдущем примере точки на осях не отнесены ни к одному из квадрантов, то есть функция μ не тотальная. Такие классификаторы называются *неполными* и требуют оговорок или доопределения при использовании.

Пусть теперь областью значений характеристических функций являются натуральные числа и 0. Тогда имеется взаимно однозначное соответствие между множеством всех мультимножеств над множеством U (п. 1.1.4) и множеством всех характеристических функций из U в $\mathbb{N}_0$. Действительно, пусть задано мультимножество $\widehat{X} = [u_1^{a_1}, \ldots, u_n^{a_n}]$ над множеством U. Тогда мультимножеству $\widehat{X}$ взаимно однозначно соответствует характеристическая функция $\mu_{\widehat{X}}\colon U \to \mathbb{N}_0$, определяемая следующим образом: $\mu_{\widehat{X}}(u_i) := a_i$. Обозначим это взаимно однозначное соответствие f_1.

ЗАМЕЧАНИЕ

Множество индикаторов является подмножеством множества мультимножеств (п. 1.1.4), а множество бинарных характеристических функций является подмножеством множества неотрицательно целочисленных характеристических функций.

Нетрудно заметить, что сужение (п. 1.6.1) соответствия f_1 на множество индикаторов устанавливает взаимно-однозначное соответствие f_2 между множеством индикаторов и множеством бинарных характеристических функций.

В п. 1.2.1 замечено, что имеется взаимно однозначное соответствие между булеаном над множеством U и множеством индикаторов над множеством U (обозначим это соответствие g_2), а в этом параграфе установлено, что имеется взаимно однозначное соответствие между булеаном над множеством U и множеством характеристических

функций подмножеств множества U (обозначим это соответствие h_2). Таким образом, имеют место взаимно однозначные соответствия и включения, представленные на рис. 1.21, причём f_2 является сужением f_1.

ОТСТУПЛЕНИЕ

Объекты, обозначенные на рис. 1.21 пунктирными линиями и вопросительными знаками, не определяются и не используются в наивной теории множеств. Это не означает, что в этих объектах зашифрована какая-то загадка. Это означает, что язык наивной теории множеств, как и всякий формальный язык, имеет ограниченную сферу применения. В подавляющем большинстве случаев языка наивной теории множеств оказывается достаточно для построения практически удобных математических моделей, и в этих случаях всё равно, на каком языке формулировать модели: на языке множеств, на языке индикаторов или на языке характеристических функций, поскольку эти языки эквивалентны. Если же языка теории множеств оказывается недостаточно, то не следует его коверкать, изобретая доморощенные «обобщения». Лучше выбрать другой язык с более широкой сферой применения, подходящий к данной задаче.

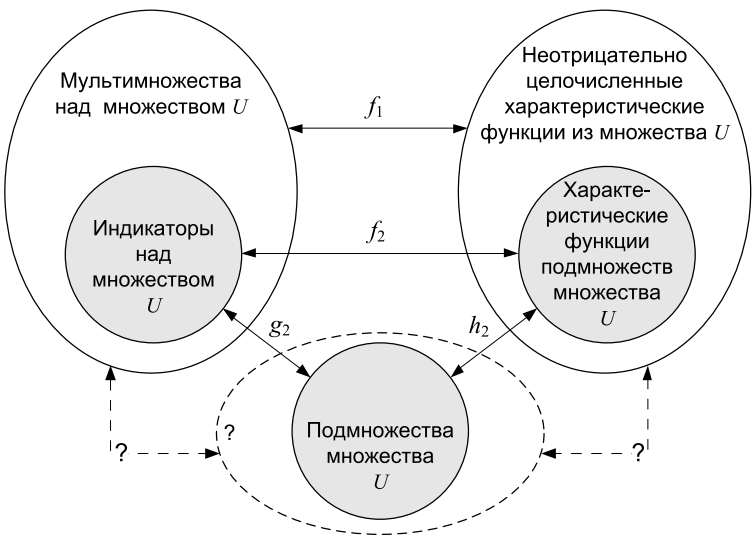


Рис. 1.21. Взаимно однозначные соответствия и включения

Отношение принадлежности элемента множеству в наивной теории множеств не имеет никакой неопределённости: либо элемент принадлежит множеству, либо не принадлежит, и третьего не дано. Но в жизни такой определённости может не быть.

Пример. Мы пользуемся понятием «высокий человек» и вправе рассматривать множество высоких людей, а также его дополнение — множество низких людей. Однако если посмотреть на рис. 1.22, становится понятно, что такой классификации недостаточно, бинарная характеристическая функция оказывается слишком грубым классификатором. Можно ввести трёхзначный классификатор и множество людей среднего роста, однако некоторая неудовлетворённость при этом остаётся, и появляются выражения «несколько ниже среднего», «чуть выше среднего», не имеющие точной интерпретации.

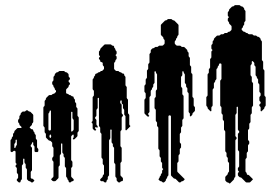


Рис. 1.22. Множество людей разного роста

В таких случаях используют *нечёткую характеристическую функцию* с множеством значений из отрезка $[0;1]$. Значение этой функции трактуется как степень принадлежности элемента множеству.

Множество, определяемое характеристической функцией с множеством значений $[0;1]$, называется *нечётким*.

ОТСТУПЛЕНИЕ

Впервые термин «нечёткая логика» (fuzzy logic) был введен Заде[1] в 1965 году. Бытует мнение, что толчком для создания новой теории послужил спор Заде со своим другом о том, чья жена привлекательнее. К единому мнению спорящие, естественно, так и не пришли. Но это спровоцировало Заде сформулировать концепцию выражения нечётких понятий («привлекательность», «молодость» и пр.) в числовой форме.

1.9.2. Характеристические функции множеств

В этом параграфе рассматриваются обычные множества и соответствующие им бинарные характеристические функции.

Используя определение $1_A(a) \stackrel{\text{Def}}{=}$ **if** $a \in A$ **then** 1 **else** 0 **end if** характеристической функции $1_A\colon U \to 0..1$ подмножества $A \subset U$ легко показать, что $1_\varnothing = 0$ и $1_U = 1$.

Характеристические функции результатов операций над множествами выражаются через характеристические функции операндов с помощью арифметических операций.

ТЕОРЕМА 1. *Пусть* $1_A\colon U \to 0..1$ *– характеристическая функция множества* $A \subset U$ *и* $1_B\colon U \to 0..1$ *– характеристическая функция множества* $B \subset U$. *Тогда:*

1) $1_{\overline{A}} = 1 - 1_A$;
2) $1_{A\cap B} = \min(1_A, 1_B) = 1_A 1_B$;
3) $1_{A\cup B} = \max(1_A, 1_B) = 1_A + 1_B - 1_A 1_B$;
4) $1_{A\setminus B} = 1_A(1 - 1_B) = 1_A - 1_A 1_B$;
5) $1_{A\triangle B} = 1_A + 1_B - 21_A 1_B$.

Доказательство. Возьмём произвольный элемент $x \in U$ и вычислим значения проверяемых выражений, рассматривая все возможные случаи принадлежности элемента x.

[1] Лотфи Заде (род. 1921).

[1] Возможны два случая, разобранные в таблице.

$1_A(x)$	$1_{\overline{A}(x)}$	$1 - 1_A(x)$
0	1	1
1	0	0

[2] Возможны четыре случая, разобранные в таблице.

$1_A(x)$	$1_B(x)$	$1_{A\cap B}(x)$	$\min(1_A(x), 1_B(x))$	$1_A(x)1_B(x)$
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0
1	1	1	1	1

Остальные утверждения доказываются аналогично. □

ТЕОРЕМА 2. 1. $A \subset B \iff \forall\, x \in U\ \ (1_A(x) \leqslant 1_B(x))$.

2. $A = B \iff \forall\, x \in U\ \ (1_A(x) = 1_B(x))$.

Доказательство.

[1] Возможны четыре случая, разобранные в таблице.

$1_A(x)$	$1_B(x)$	$x \in A \Longrightarrow x \in B$	$1_A(x) \leqslant 1_B(x)$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	1	1

[2] По определению, $A = B \iff A \subset B\ \&\ B \subset A$. Имеем: $A = B \iff$
$\iff \forall\, x \in U\ \ (1_A(x) \leqslant 1_B(x)\ \&\ 1_B(x) \leqslant 1_A(x)) \iff \forall\, x \in U\ \ (1_A(x) = 1_B(x))$. □

ЗАМЕЧАНИЕ

Нетрудно показать, что все равенства, перечисленные в п. 1.2.9, выполняются для характеристических функций. Например, выполняется свойство идемпотентности: $A \cup A = A$, так как $\max(1_A, 1_A) = 1_A$.

Характеристические функции часто оказываются полезны при решении задач в алгебре множеств.

Пример. Требуется доказать равенство $(A \triangle B) \triangle C = A \triangle (B \triangle C)$.

Выразим левую и правую часть равенства в терминах характеристических функций.

$(A \triangle B) \triangle C \mapsto 1_A + 1_B - 21_A1_B + 1_C - 21_C(1_A + 1_B - 21_A1_B) =$
$= 1_A + 1_B + 1_C - 2(1_B + 21_A1_B + 21_A1_C) + 4(1_A1_B1_C)$.

$A \triangle (B \triangle C) \mapsto 1_A + (1_B + 1_C - 21_B1_C) - 21_A(1_B + 1_C - 21_B1_C) =$
$= 1_A + 1_B + 1_C - 2(1_B + 21_A1_B + 21_A1_C) + 4(1_A1_B1_C)$.

Результаты совпали, равенство доказано.

Таким образом, доказательство получилось в один шаг, тогда как при решении задачи эквивалентными преобразованиями по равенствам п. 1.2.9 требуется больше десяти шагов.

ОТСТУПЛЕНИЕ

Арифметические вычисления обычному человеку выполнять проще, легче и привычнее всего, поэтому решение в терминах характеристических функций оказывается наглядным и коротким. На самом деле оно эквивалентно решению в терминах операций наивной теории множеств, просто многие арифметические операции выполняются «в уме» и не выписываются в выкладках явно.

1.9.3. Характеристические функции отношений

В этом параграфе рассматриваются бинарные отношения на множестве и соответствующие им бинарные характеристические функции двух аргументов, используя определение $1_R(a,b) \stackrel{\text{Def}}{=}$ **if** aRb **then** 1 **else** 0 **end if** характеристической функции $1_R\colon A \times A \to 0..1$ отношения $R \subset A^2$.

Характеристические функции результатов некоторых операций над отношениями выражаются через характеристические функции операндов.

ТЕОРЕМА. *Пусть* $1_R\colon A^2 \to 0..1$ *– характеристическая функция бинарного отношения* $R \subset A^2$, $A = \{a_1, \ldots, a_n\}$, $x, y \in A$. *Тогда:*

1) $1_{\overline{R}}(x,y) = 1 - 1_R(x,y)$;

2) $1_{R \circ R}(x,y) = \bigvee\limits_{i=1}^{n} 1_R(x,a_i)1_R(a_i,y)$;

3) $1_{\ker R}(x,y) = \bigvee\limits_{i=1}^{n} 1_R(x,a_i)1_R(y,a_i)$;

4) $1_{R^{-1}}(x,y) = 1_R(y,x)$.

Доказательство. Рассмотрим произвольную пару $(x,y) \in A^2$ и вычислим значения проверяемых выражений в различных случаях.

[1)] Возможны два случая, разобранные в таблице.

$1_R(x,y)$	$1_{\overline{R}}(x,y)$	$1 - 1_R(x,y)$
0	1	1
1	0	0

[2)] Возможны два случая, разобранные в таблице.

$\exists\, i \in 1..n\ \ (xRa_i\ \&\ a_iRy)$	$1_{R \circ R}(x,y)$	$\bigvee\limits_{i=1}^{n} 1_R(x,a_i)1_R(a_i,y)$
0	0	0
1	1	1

[3)] Возможны два случая, разобранные в таблице.

$\exists\, i \in 1..n\ \ (xRa_i\ \&\ yRa_i)$	$1_{\ker R}(x,y)$	$\bigvee\limits_{i=1}^{n} 1_R(x,a_i)1_R(y,a_i)$
0	0	0
1	1	1

[4)] В любом случае $(x,y) \in R \Longleftrightarrow (y,x) \in R^{-1}$. □

ЗАМЕЧАНИЕ

Полезно сравнить эту теорему с теоремами п. 1.4.9.

ОТСТУПЛЕНИЕ

Операции с характеристическими функциями отношений не удаётся свести к простым арифметическим манипуляциям так, как это сделано для характеристических функций множеств. Поэтому на практике характеристические функции отношений в том виде, как они представлены в этом параграфе, используются редко. Чаще характеристические функции отношений (под названием предикатов) трактуются средствами математической логики (см. главу 4).

1.9.4. Характеристические функции мультимножеств

Пусть задано мультимножество $\widehat{X} = [u_1^{a_1}, \ldots, u_n^{a_n}]$ над множеством $X = \{x_1, \ldots, x_n\}$, где a_i — показатель элемента x_i (п. 1.1.4). *Характеристической функцией мультимножества* $\widehat{X}$ называется тотальная функция $\mu_{\widehat{X}}\colon X \to \mathbb{N}_0$, которая определяется так: $\forall\, x \in X \ \left(\mu_{\widehat{X}}(x_i) \stackrel{\text{Def}}{=} a_i\right)$.

Пользуясь характеристическими функциями, для мультимножеств легко ввести те же операции, что и для множеств. Пусть A, B, C — мультимножества над множеством X, а μ_A, μ_B, μ_C — их характеристические функции, соответственно.

Мощность мультимножества: $|A| \stackrel{\text{Def}}{=} \sum\limits_{x \in X} \mu_A(x_i)$.

Включение мультимножеств: $A \subset B \stackrel{\text{Def}}{=} \forall\, x \in X \ \ (\mu_A(x) \leqslant \mu_B(x))$.

Равенство мультимножеств: $A = B \stackrel{\text{Def}}{=} \forall\, x \in X \ \ (\mu_A(x) = \mu_B(x))$.

Пересечение мультимножеств: $\mu_{A \cap B}(x) \stackrel{\text{Def}}{=} \min(\mu_A(x), \mu_B(x))$.

Объединение мультимножеств: $\mu_{A \cup B}(x) \stackrel{\text{Def}}{=} \max(\mu_A(x), \mu_B(x))$.

ЗАМЕЧАНИЕ

Учитывая теоремы п. 1.9.2, легко показать, что введённые таким образом операции над мультимножествами согласованы с одноимёнными операциями над множествами, в частности, совпадают при применении к индикаторам. Однако в общем случае операции над мультимножествами не обязаны обладать такими же свойстами, как операции над множествами. Например, равенство $\mu_{A \cap B} = \mu_A \cdot \mu_B$ всегда выполняется для индикаторов, но не всегда выполняется для мультимножеств в общем случае.

1.9.5. Классификаторы

Характеристическая функция $\mu\colon U \to \mathrm{X}$, где $\mathrm{X} = \{\chi_1, \ldots, \chi_n\}$ — конечное множество, называется *классификатором*.

Классификатор задаёт семейство подмножеств $\mathcal{E}$ множества U ($\mathcal{E} = \{E_i\}_{i=1}^n$), в котором все элементы одного подмножества объединены одной характеристикой: $\forall\, x, y \in E_i \ \ (\mu(x) = \mu(y) = \chi_i)$ (семейство множеств уровня, п. 1.7.4). Поскольку классификатор — тотальная функция, это семейство является дизъюнктным покрытием, то есть разбиением (п. 1.2.7), и задаёт отношение эквивалентности на множестве U (п. 1.7.1). Таким образом, классификаторы — ещё один способ описания часто используемой концепции разбиений, имеющий множество приложений. Например, *дискриминатный анализ* в статистике, *распознавание образов* в искусственном интеллекте, *систематика* живых организмов в биологии.

Особый интерес представляет случай, когда множество U очень велико или даже бесконечно. При этом одним классификатором трудно обойтись: либо классы оказываются слишком большими, либо их оказывается слишком много. В этом случае

на практике применяют один из двух основных методов классификации: фасетную классификацию или иерархическую классификацию.

При *фасетной классификации* на множестве U определяются независимо несколько классификаторов: $\mu_1, \ldots, \mu_k$, каждый из которых задает своё разбиение, $\mu_i \colon U \to \mathrm{X_i}$. Эти классификаторы иногда называют признаками, или свойствами. *Фасетный классификатор* — это общее измельчение всех разбиений, $\mu \colon U \to \mathrm{X}_1 \times \ldots \times \mathrm{X}_k$. На рис. 1.14 представлен пример фасетного классификатора с двумя признаками.

Пример. В системах просмотра фильмов через Интернет каталог доступных фильмов обычно устроен как фасетный классификатор: можно указать жанр фильма, страну производства фильма, год выпуска и т. д. Обычно для точного поиска оказывается достаточным иметь около десяти независимых признаков.

Иерархический классификатор также задаётся несколькими классификаторами. Опеределяется классификатор верхнего уровня. Затем на всех блоках разбиения классификатора первого уровня определяются свои классификаторы второго уровня, и так далее. Уровней измельчения разбиений может быть сколько угодно. Такие разбиения образуют иерархию, то есть неупорядоченное ориентированное дерево (см. п. 9.2.1).

При измельчении блоков используемые классификаторы и глубина измельчения могут совпадать, а могут и не совпадать с другими блоками того же уровня. В первом случае дерево классификации имеет постоянную *ширину ветвления* и является *выровненным*, а во втором случае дерево может иметь произвольную форму.

Класс каждого объекта определяется как *последовательность* уточняющих характеристик $(\chi_1, \ldots, \chi_k)$ (путь в дереве от корня), причём разные объекты могут иметь различное количество характеристик.

Пример. *Универсальная десятичная классификация* (УДК) — международная система классификации информации, используемая в России. Вся информация делится на 10 блоков, каждому из которых соответствует цифра от 0 до 9. Каждый блок в свою очередь делится ещё на 10 подблоков, каждому из которых ставится в соответствие цифра от 0 до 9 и т. д. Таким образом, каждому блоку соответствует код, составленный из кода своего предка и своей цифры. Так, например, блоку «Религии Индийского субконтинента» соответствует код 23 (рис. 1.23).

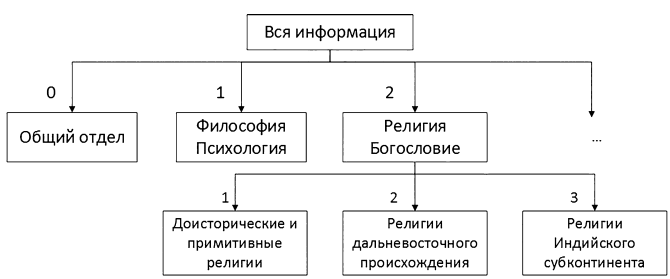


Рис. 1.23. Универсальная десятичная классификация

1.9.6. Нечёткие множества

Нечёткое множество $A \subset U$ описывается нечёткой характеристической функцией вида $\mu_A\colon U \to [0;1]$. Для нечётких множеств определены следующие операции:

1) *нечёткое дополнение*: $1_{\overline{A}} \stackrel{\text{Def}}{=} 1 - 1_A$;

2) *нечёткое пересечение*: $1_{A\cap B} \stackrel{\text{Def}}{=} \min(1_A, 1_B)$;

3) *нечёткое объединение*: $1_{A\cup B} \stackrel{\text{Def}}{=} \max(1_A, 1_B)$.

Некоторые свойства операций над обычными множествами также сохраняются и для нечётких множеств.

ТЕОРЕМА 1. *Для операций над нечёткими множествами выполняются законы де Моргана:* $\overline{A \cup B} = \overline{A} \cap \overline{B}$, $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Доказательство. В терминах характеристических функций первое утверждение может быть записано в виде $1 - \max(\mu_A, \mu_B) = \min(1 - \mu_A, 1 - \mu_B)$ и легко проверено с учётом двух случаев: $\mu_A(x) > \mu_B(x)$ и $\mu_A(x) \leqslant \mu_B(x)$. Второе утверждение доказывается аналогично. □

ТЕОРЕМА 2. *Нечёткие объединение и пересечение взаимно дистрибутивны:*

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C), \quad A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

Доказательство. В терминах характеристических функций первое утверждение имеет вид $\max(\mu_A, \min(\mu_B, \mu_C)) = \min(\max(\mu_A, \mu_B), \max(\mu_A, \mu_C))$ и может быть проверено с учётом шести случаев:

случай	$\max(\mu_A, \min(\mu_B, \mu_C))$	$\min(\max(\mu_A, \mu_B), \max(\mu_A, \mu_C))$
$\mu_A \geqslant \mu_B \geqslant \mu_C$	μ_A	μ_A
$\mu_A \geqslant \mu_C \geqslant \mu_B$	μ_A	μ_A
$\mu_B \geqslant \mu_A \geqslant \mu_C$	μ_B	μ_B
$\mu_B \geqslant \mu_C \geqslant \mu_A$	μ_B	μ_B
$\mu_C \geqslant \mu_A \geqslant \mu_B$	μ_C	μ_C
$\mu_C \geqslant \mu_B \geqslant \mu_A$	μ_C	μ_C

Второе утверждение доказывается аналогично. □

ЗАМЕЧАНИЕ

Не все свойства операций над обычными множествами сохраняются и для нечётких множеств. В частности, идемпотентность, коммутативность, ассоциативность, дистрибутивность, поглощение, инволютивность и законы де Моргана выполняются, но, например, свойство дополнения (п. 1.2.9) не работает, так как $A \cup \overline{A} = U$ в терминах характеристических функций записывается в виде $\max(\mu_A, 1 - \mu_A) = 1$, что, очевидно, неверно для всех $\mu_A\colon U \to [0;1]$.

Пример. На рис. 1.24 проиллюстрировано пересечение двух нечётких множеств, а именно множеств старых и молодых людей.

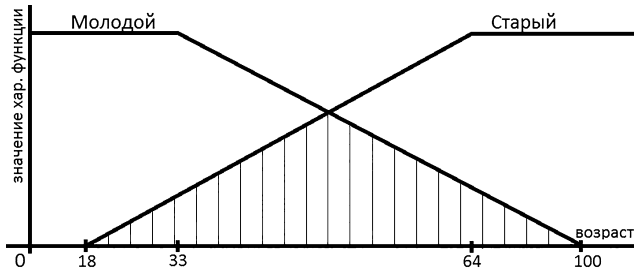


Рис. 1.24. Характеристическая функция пересечения множеств старых и молодых людей

Для нечётких множеств определяются следующие алгебраические операции, которых нет в наивной теории множеств:

алгебраическое умножение: $\mu_{AB} = \mu_A \mu_B$;
алгебраическая сумма: $\mu_{A+B} = \mu_A + \mu_B$;
абсолютная разность: $\mu_{|A-B|} = |\mu_A - \mu_B|$.

ЗАМЕЧАНИЕ

Можно заметить, что для обычных множеств пересечение и алгебраическое умножение, а также объединение и алгебраическая сумма являются эквивалентными понятиями.

ОТСТУПЛЕНИЕ

Элементами обычной логики являются простые высказывания — утверждения, которые либо истинны, либо ложны (см. п. 4.1.1). Для построения такой логики вполне достаточно наивной теории множеств с чётким понятием принадлежности. Однако, в реальных задачах часто приходится сталкиваться с ситуациями, где этого недостаточно, где невозможно однозначно ответить на вопрос принадлежит ли какой-либо объект множеству или нет, верно некоторое утверждение или нет. Возможно лишь *оценить* степень принадлежности элемента или степень достоверности утверждения. Тогда необходимо использование нечёткой логики, построенной на понятиях нечётких множеств. Нечёткая логика тесно связана с вероятностью и используется во многих приложениях, в частности, в искусственном интеллекте.

Глава 2 Алгебраические структуры

Алгебраические методы находят самое широкое применение при формализации различных предметных областей. Грубо говоря, при построении модели предметной области всё начинается с введения подходящих обозначений для операций и отношений с последующим исследованием их свойств. Владение алгебраической терминологией, таким образом, входит в арсенал средств, необходимых для абстрактного моделирования, предшествующего практическому программированию задач конкретной предметной области. Материал этой главы, помимо введения в терминологию общей алгебры, содержит некоторое количество примеров конкретных алгебраических структур. Вначале бегло рассматриваются классические структуры, которые обычно включаются в курсы общей алгебры, а также элементарное введение в теорию чисел, необходимую в некоторых современных приложениях. Затем обсуждаются некоторые более специальные структуры, наиболее часто применяемые в конкретных программных построениях. Особого внимания заслуживает понятие матроида, обсуждаемое в этой главе, поскольку матроиды тесно связаны с важнейшим классом эффективных алгоритмов, называемых *жадными*.

2.1. Алгебры и морфизмы

Словом «алгебра» называют, вообще говоря, раздел математики, посвящённый изучению систем операций над элементами некоторого множества. В такой самой общей постановке область применения алгебраических методов оказывается практически неограниченной, поскольку в любой предметной области явно выделяются определённые множества объектов и характерные операции с этими объектами. Чтобы подчеркнуть весьма общий характер алгебраических методов, слово «алгебра» в этом смысле употребляют вместе с подходящим прилагательным: «общая алгебра», «универсальная алгебра», «абстрактная алгебра».

При этом конкретные объекты, изучаемые в алгебре, также называют словосочетанием, в которое входит слово «алгебра». Например, «алгебра подмножеств» (п. 1.2.8), «булева алгебра» (см. п. 2.6.6.), «алгебра булевых функций» (см. п. 3.2.4). Для краткости и без риска возникновения недоразумений слово «алгебра» без дополнительных слов в этом учебнике используется как родовое понятие для обозначения разнообразных алгебраических структур, как конкретных, так и абстрактных.

2.1.1. Операции и их носители

Всюду определённая (тотальная) функция $\varphi\colon M^n \to M$ называется *n-арной операцией* на M (иногда говорят *n-местная операция*). Если операция φ — бинарная ($\varphi\colon M \times M \to M$), то обычно используют *инфиксную форму записи* $a\varphi b$ вместо $\varphi(a,b)$ или $a * b$, где $*$ — знак операции. Если же операция унарная $\varphi\colon M \to M$, то используют *префиксную форму записи* φa или *постфиксную форму записи* $a\varphi$.

ЗАМЕЧАНИЕ

Всякая n-арная операция $\varphi\colon M^n \to M$ задаёт $n+1$-арное отношение $\Phi \subset M^{n+1}$ на множестве M: $\Phi := \{(x_1,\ldots,x_n,y) \in M^{n+1} \mid y = \varphi(x_1,\ldots,x_n)\}$.

Множество M вместе с набором операций $\Sigma = \{\varphi_1, \ldots, \varphi_m\}$, $\varphi_i \colon M^{n_i} \to M$, где n_i — арность операции φ_i, называется *алгебраической структурой*, *универсальной алгеброй* или просто *алгеброй*. При этом множество M называется *основным (несущим)* множеством или *основой (носителем)*; вектор арностей $(n_1, \ldots, n_m)$ называется *типом*; множество операций Σ называется *сигнатурой*. Для обозначения алгебры используется запись: $\langle M; \Sigma \rangle$ или $\langle M; \varphi_1, \ldots, \varphi_m \rangle$. Операции φ_i *конечноместны*, сигнатура Σ конечна. Носитель не обязательно конечен, но не пуст.

ЗАМЕЧАНИЕ

Далее для обозначения алгебры везде, где это возможно, используется прописная рукописная буква, а для обозначения её носителя — соответствующая обычная прописная буква: $\mathcal{A} = \langle A; \Sigma \rangle$. Такое соглашение позволяет использовать вольности в обозначениях, не вводя явно две буквы для алгебры и для носителя, а подразумевая одну из них по умолчанию. Например, выражение «алгебра A» означает алгебру $\mathcal{A}$ с носителем A и сигнатурой, которая ясна из контекста, а запись $a \in \mathcal{A}$ означает элемент a из носителя A алгебры $\mathcal{A}$.

Если в качестве φ_i допускаются не только функции, но и отношения, то множество M вместе с набором операций и отношений называется *моделью*.

В приложениях обычно используется следующее обобщение понятия алгебры. Пусть $M = \{M_1, \ldots, M_n\}$ — множество *основ*, $\Sigma = \{\varphi_1, \ldots, \varphi_m\}$ — сигнатура, причём $\varphi_i \colon M_{i_1} \times \ldots \times M_{i_{n_i}} \to M_{i_0}$. Тогда $\langle M; \Sigma \rangle$ называется *многоосновной* алгеброй. Другими словами, многоосновная алгебра имеет несколько носителей, а операция сигнатуры действует из прямого произведения некоторых носителей в некоторый носитель.

2.1.2. Замыкания и подалгебры

Подмножество носителя $X \subset M$ называется *замкнутым* относительно операции φ, если $\forall\, x_1, \ldots, x_n \in X \;\; (\varphi(x_1, \ldots, x_n) \in X)$. Если X замкнуто относительно всех $\varphi \in \Sigma$, то $\langle X; \Sigma_X \rangle$ называется *подалгеброй* $\langle M; \Sigma \rangle$; операции подалгебры рассматриваются как сужения: $\Sigma_X = \{\varphi_i^X\}$, $\varphi_i^X = \varphi_i\big|_{X^k}$, $k = n_i$.

Примеры

1. Алгебра $\langle \mathbb{R}; +, \cdot \rangle$ — *поле вещественных чисел*. Тип — $(2, 2)$. Все конечные подмножества, кроме $\{0\}$, не замкнуты относительно сложения, и все конечные подмножества, кроме $\{0\}$ и $\{0, 1\}$, не замкнуты относительно умножения. *Поле рациональных чисел* $\langle \mathbb{Q}; +, \cdot \rangle$ образует подалгебру. *Кольцо целых чисел* $\langle \mathbb{Z}; +, \cdot \rangle$ образует подалгебру алгебры рациональных и, соответственно, вещественных чисел.
2. Алгебра $\left\langle 2^M; \cup, \cap, ^- \right\rangle$ — *алгебра подмножеств* над множеством M. Тип — $(2, 2, 1)$. При этом $\forall\, X \subset M \;\; \left(\left\langle 2^X; \cup, \cap, ^- \right\rangle\right)$ — её подалгебра.
3. Алгебра гладких функций $\left\langle \{f \mid f \colon \mathbb{R} \to \mathbb{R}\}; \frac{d}{dx} \right\rangle$, где $\frac{d}{dx}$ — операция дифференцирования. Множество полиномов одной переменной x образует подалгебру, которая обозначается $\mathbb{R}[x]$.

ТЕОРЕМА 1. *Непустое пересечение подалгебр некоторой алгебры образует подалгебру этой же алгебры.*

Доказательство. Пусть $\langle X_i; \Sigma_{X_i}\rangle$ — подалгебра $\langle M; \Sigma\rangle$. Обозначим $X := \bigcap X_i$. Тогда $\forall\, i\ \left(\forall\, j\ \left(\varphi_j^{X_i}(x_1, \ldots, x_{n_j}) \in X_i\right)\right) \Longrightarrow \forall\, j\ \left(\varphi_j^{X_i}(x_1, \ldots, x_{n_j}) \in \bigcap X_i\right)$. □

Замыканием множества $X \subset M$ относительно сигнатуры Σ (обозначается $[X]_\Sigma$) называется множество всех элементов (включая сами элементы множества X), которые можно получить, применяя операции из Σ. Если сигнатура подразумевается, её можно не указывать.

Пример. В кольце целых чисел $\langle \mathbb{Z}; +, \cdot\rangle$ замыканием числа 2 являются чётные числа, то есть $[\{2\}] = \{n \in \mathbb{Z} \mid n = 2k \ \&\ k \in \mathbb{Z}\}$.

ТЕОРЕМА 2. *Замыкание обладает следующими свойствами:*

1) $X \subset Y \Longrightarrow [X] \subset [Y]$;
2) $X \subset [X]$;
3) $[[X]] = [X]$;
4) $[X] \cup [Y] \subset [X \cup Y]$.

Доказательство.

[1] Очевидно.

[2] По определению.

[3] По определению.

[4] Действительно, в $[X \cup Y]$ входят не только те элементы, которые можно получить из элементов X и из элементов Y, но и те элементы, которые можно получить из комбинаций этих элементов. □

Пусть $\mathcal{A} = \langle A; \Sigma\rangle$ — некоторая алгебра и $X_1, \ldots, X_n \subset A$ — некоторые подмножества носителя, а $\varphi \in \Sigma$ — одна из операций алгебры. Тогда используется следующее соглашение об обозначениях:

$$\varphi(X_1, \ldots, X_n) \stackrel{\text{Def}}{=} \{\varphi(x_1, \ldots, x_n) \mid x_1 \in X_1 \ \&\ \ldots \ \&\ x_n \in X_n\},$$

то есть алгебраические операции можно применять не только к отдельным элементам, но и к множествам (подмножествам носителя), получая, соответственно, не отдельные элементы, а множества (подмножества носителя), подобно тому, как это определено в п. 1.6.3.

2.1.3. Система образующих

Множество $M' \subset M$ называется *системой образующих* алгебры $\langle M; \Sigma\rangle$, если $[M']_\Sigma = M$. Если алгебра имеет конечную систему образующих, то алгебра называется *конечно-порождённой*. Бесконечные алгебры могут иметь конечные системы образующих.

Примеры

1. Алгебра *натуральных чисел* — $\langle \mathbb{N}; +\rangle$ — с операцией сложения имеет конечную систему образующих $1 \in \mathbb{N}$.
2. Алгебра $\langle \mathbb{N}; *\rangle$ с тем же носителем, но с операцией умножения не является конечно-порождённой. Системой образующих этой алгебры является множество простых чисел, бесконечность которого установлена в п. 2.4.4.

Пусть заданы набор функциональных символов $\Phi = \{\varphi_1, \ldots, \varphi_m\}$, служащих обозначениями функций некоторой сигнатуры Σ типа $N = (n_1, \ldots, n_m)$, и множество переменных $V = \{x_1, x_2, \ldots\}$. Определим множество *термов* T индуктивным образом:

1) $V \subset T$;
2) $t_1, \ldots, t_{n_i} \in T \ \& \ \varphi_i \in \Phi \Longrightarrow \varphi_i(t_1, \ldots, t_{n_i}) \in T$.

Алгебра $\langle T; \Phi \rangle$ называется *свободной алгеброй термов*. Носителем этой алгебры является множество термов, то есть формальных выражений, построенных с помощью знаков операций сигнатуры Σ. Заметим, что множество V является системой образующих свободной алгебры термов.

Пример. Если $V = \{x\}$ и $\Sigma = \{+, \cdot\}$, то свободная алгебра термов — это множество всех выражений, которые можно построить из переменной x с помощью операций сложения и умножения, то есть алгебра полиномов от одной переменной с натуральными коэффициентами и без свободных членов.

2.1.4. Свойства операций

Некоторые часто встречающиеся свойства операций имеют специальные названия. Пусть задана алгебра $\langle M; \Sigma \rangle$ и $a, b, c \in M$; $\circ, \diamond \in \Sigma$; $\circ, \diamond \colon M \times M \to M$. Тогда свойства операций определяются следующим образом:

1) *ассоциативность*: $(a \circ b) \circ c = a \circ (b \circ c)$;
2) *коммутативность*: $a \circ b = b \circ a$;
3) *дистрибутивность* $\diamond$ относительно $\circ$ слева: $a \diamond (b \circ c) = (a \diamond b) \circ (a \diamond c)$;
4) *дистрибутивность* $\diamond$ относительно $\circ$ справа: $(a \circ b) \diamond c = (a \diamond c) \circ (b \diamond c)$;
5) *поглощение* ($\diamond$ поглощает $\circ$): $(a \circ b) \diamond a = a$;
6) *идемпотентность*: $a \circ a = a$.

Примеры

1. Ассоциативные операции: сложение и умножение чисел, объединение и пересечение множеств, композиция отношений. Неассоциативные операции: возведение чисел в степень, вычитание множеств.
2. Коммутативные операции: сложение и умножение чисел, объединение и пересечение множеств. Некоммутативные операции: умножение матриц, композиция отношений, возведение в степень.
3. Дистрибутивные операции: умножение относительно сложения чисел. Недистрибутивные операции: возведение в степень дистрибутивно относительно умножения справа, но не слева: $\big((ab)^c = a^c b^c, a^{bc} \neq a^b a^c\big)$.
4. Пересечение поглощает объединение, объединение поглощает пересечение. Сложение и умножение не поглощают друг друга.
5. Идемпотентные операции: наибольший общий делитель натуральных чисел, объединение и пересечение множеств. Неидемпотентные операции: сложение и умножение чисел.

2.1.5. Гомоморфизмы и изоморфизмы алгебр

Понятие гомоморфизма является одним из ключевых понятий алгебры. Каждая алгебраическая структура определённым образом выделяет класс «разумных» отображений между объектами с данной структурой, согласованных с операциями этой структуры. Поскольку операция — частный случай отношения, согласованность означает выполнение основного свойства гомоморфизма, указанного в п. 1.7.6.

Применительно к алгебраическим структурам гомоморфизм определяется для алгебр одного типа. Пусть $\mathcal{A} = \langle A; \varphi_1, \ldots, \varphi_m \rangle$ и $\mathcal{B} = \langle B; \psi_1, \ldots, \psi_m \rangle$ — две алгебры одного типа. Если функция $f\colon A \to B$ является гомоморфизмом для всех пар φ_i и ψ_i, то есть $\forall\, i \in 1..m\ \ (f(\varphi_i(a_1, \ldots, a_{n_i})) = \psi_i(f(a_1), \ldots, f(a_{n_i})))$, то говорят, что f — *гомоморфизм* из $\mathcal{A}$ в $\mathcal{B}$.

Пример. Пусть $\mathcal{A} = \langle \mathbb{N}; + \rangle$, $\mathcal{B} = \langle \mathbb{N}_{10}; +_{10} \rangle$, где $\mathbb{N}_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, а $+_{10}$ — сложение по модулю 10. Тогда $f : a \mapsto (a \mod 10)$ — гомоморфизм из $\mathcal{A}$ в $\mathcal{B}$.

Биективный гомоморфизм называется *изоморфизмом* (п. 1.7.6).

Пример. Биекция $n \mapsto 2n$ является изоморфизмом между множеством натуральных чисел и множеством чётных чисел с операцией сложения.

ЗАМЕЧАНИЕ

Сложение является операцией на множестве чётных чисел (сумма чётных чисел — чётное число), но не является операцией на множестве нечётных чисел.

Гомоморфизмы, обладающие дополнительными свойствами, имеют специальные названия. Инъективный гомоморфизм называется *мономорфизмом*. Сюръективный гомоморфизм называется *эпиморфизмом*. Другими словами, изоморфизм является одновременно мономорфизмом и эпиморфизмом. Если $A = B$, то гомоморфизм называется *эндоморфизмом*, а изоморфизм называется *автоморфизмом*. Общее соотношение между различными видами гомоморфизмов представлено на рис. 2.1.

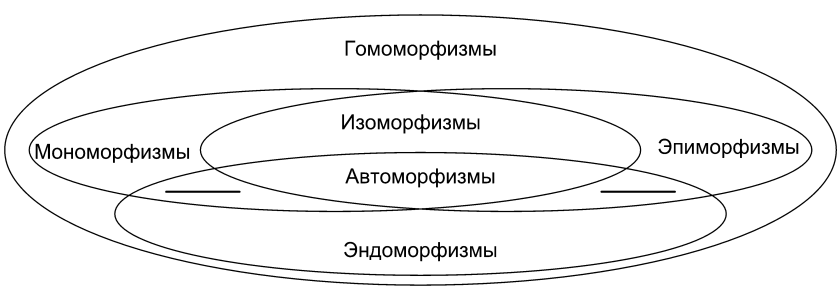


Рис. 2.1. Общее соотношение между различными видами гомоморфизмов

ЗАМЕЧАНИЕ

Прочерки на рисунке означают, что такие гомоморфизмы могут возникнуть только на бесконечных множествах, поэтому в дискретном случае им не дают специальных названий.

Примеры

1. Пусть $n \mapsto 2n$ — гомоморфизм, сохраняющий отношение порядка. Он является эндоморфизмом и мономорфизмом.
2. Пусть A^+ — множество непустых слов в алфавите A и $|\alpha|$ — длина слова α. Длина является гомоморфизмом относительно операции конкатенации на множестве A^+ и операции $+$ на множестве $\mathbb{N}$: $\forall\, \alpha, \beta \in A^+$ $(|\alpha\beta| = |\alpha| + |\beta|)$. Этот гомоморфизм является эпиморфизмом.

Пусть $\mathcal{A} = \langle A; \varphi\rangle$, $\mathcal{B} = \langle B; \psi\rangle$, тип $= (1)$ и $f\colon A \to B$. Действие функций φ, ψ, f можно изобразить с помощью диаграммы, представленной на рис. 2.2. Пусть f — гомоморфизм. Тогда если взять конкретное значение $a \in A$ и двигаться по двум различным путям на диаграмме, то получится один и тот же элемент $b \in B$ (так как $f(\varphi(a)) = \psi(f(a))$). В таком случае диаграмма называется *коммутативной*. Коммутативной диаграмма называется потому, что условие гомоморфизма можно переписать так: $f \circ \varphi = \psi \circ f$, где $\circ$ — суперпозиция функций. Коммутативные диаграммы широко используются в абстрактной алгебре и теории категорий.

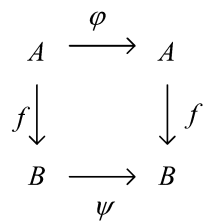


Рис. 2.2. Коммутативная диаграмма

Если $f\colon A \to B$ — изоморфизм, то алгебры $\mathcal{A}$ и $\mathcal{B}$ называют *изоморфными* и обозначают так: $\mathcal{A} \overset{f}{\sim} \mathcal{B}$. Если f ясно из контекста или просто неважно в конкретном рассуждении, то пишут $\mathcal{A} \sim \mathcal{B}$.

Из теорем п. 1.7.6 непосредственно следует, что отношение изоморфности на множестве однотипных алгебр является эквивалентностью.

ЗАМЕЧАНИЕ

Напомним, что изоморфизм алгебр уважает *все* операции.

Примеры

1. Пусть $\mathcal{A} = \langle \mathbb{N}; +\rangle$, $\mathcal{B} = \langle \{n \mid n = 2k, k \in \mathbb{N}\}; +\rangle$ — *чётные* числа. Тогда $\mathcal{A} \overset{\times 2}{\sim} \mathcal{B}$.
2. $\mathcal{A} = \langle 2^M; \cap, \cup\rangle \overset{f}{\sim} \mathcal{B} = \langle 2^M; \cup, \cap\rangle$, где $f(X) = \overline{X}$.
3. $\mathcal{A} = \langle \mathbb{R}_+; \cdot\rangle \overset{f}{\sim} \mathcal{B} = \langle \mathbb{R}; +\rangle$, где $f(x) = \log_a x$.

Понятие изоморфизма является одним из центральных понятий, оправдывающих применимость алгебраических методов в различных областях. Действительно, пусть $\mathcal{A} = \langle A; \Sigma_\varphi\rangle$, $\mathcal{B} = \langle B; \Sigma_\psi\rangle$ и $\mathcal{A} \overset{f}{\sim} \mathcal{B}$. Пусть в алгебре $\mathcal{A}$ установлено свойство $\Phi_1 = \Phi_2$, где Φ_1 и Φ_2 — некоторые формулы в сигнатуре Σ_φ. Поскольку алгебры $\mathcal{A}$ и $\mathcal{B}$ изоморфны, отсюда немедленно следует, что в алгебре $\mathcal{B}$ справедливо свойство $\Psi_1 = \Psi_2$, где Ψ_1 и Ψ_2 — формулы, полученные из формул Φ_1 и Φ_2 заменой операций из сигнатуры Σ_φ соответствующими операциями из сигнатуры Σ_ψ. Таким образом, достаточно установить некоторое свойство в одной алгебре, и оно автоматически

распространяется на все изоморфные алгебры. Алгебраические структуры принято рассматривать *с точностью до изоморфизма*.

Понятие изоморфизма применяется не только в алгебре, но и практически во всех областях математики. Например, в этом учебнике рассматривается изоморфизм множеств (п. 1.2.2), линейно упорядоченных множеств (п. 1.8.8), графов (см. п. 7.1.6).

ОТСТУПЛЕНИЕ

Термин «морфизм», вынесенный в название раздела, используется как собирательное понятие, подобно тому, как в объектно-ориентированном программировании используется термин «абстрактный класс». Напомним, что абстрактный класс — это класс, который не имеет непосредственных экземпляров, но является обобщением (предком) некоторых конкретных классов, которые могут иметь непосредственные экземпляры. Понятие морфизма является одним из ключевых понятий математики.

В этом разделе дано определение нескольких подклассов морфизмов, применимое к алгебраическим структурам, то есть множествам с операциями. В п. 1.7.6 рассмотрены подклассы морфизмов, применимых к множествам с отношениями. Более общее определение морфизмов, пригодное для самых разных математических объектов (множеств, графов, топологических пространств и т. д.) даёт *теория категорий*.

2.2. Алгебры с одной операцией

Естественно начать изучение алгебраических структур с наиболее простых. Самой простой структурой является алгебра с одной унарной операцией, или *унар*. Здесь этот случай не рассматривается, хотя он достаточно содержателен.

Следующим по порядку сложности является случай алгебры с одной бинарной операцией $*\colon M \times M \to M$, который рассматривается в этом разделе.

Операция часто обозначается знаком $*$ и называется «умножение», но это не более чем упрощение речи. Выполнение свойств арифметического умножения «по умолчанию» не предполагается.

2.2.1. Полугруппы

Полугруппа — это алгебра с одной ассоциативной бинарной операцией:

$$a * (b * c) = (a * b) * c.$$

Примеры

1. Множество непустых слов A^+ в алфавите A образует полугруппу относительно операции *конкатенации* (п. 1.1.5).
2. Всякое множество тотальных функций одного аргумента $\{f \mid f\colon M \to M\}$, замкнутое относительно суперпозиции, является полугруппой.

Если в полугруппе существует система образующих, состоящая из одного элемента, то такая полугруппа называется *циклической*.

Пример. $\langle \mathbb{N}; + \rangle$ является циклической полугруппой, поскольку $\{1\}$ является системой образующих.

Пусть $P = \langle M; * \rangle$ — полугруппа с конечной системой образующих A, $A \subset M$, $A = \{a_1, \ldots, a_n\}$. Тогда $\forall\, x \in M\ \ (\exists\, y_1, \ldots, y_k \in A\ \ (x = a_1 * \ldots * a_k))$. Если опустить обозначение операции $*$, то всякий элемент $x \in M$ можно представить как слово α в алфавите A, то есть $M \subset A^+$ (п. 1.1.5).

2.2.2. Определяющие соотношения

Пусть задана полугруппа с конечной системой образующих A. Обозначим через $\bar{\alpha}$ элемент, определяемый словом α. Слова α и β могут быть различными, но соответствующие им элементы $\bar{\alpha}$ и $\bar{\beta}$ могут быть равны: $\bar{\alpha} = \bar{\beta}$, $\alpha, \beta \in A^+$, $\bar{\alpha}, \bar{\beta} \in M$. Такие равенства называются *определяющими соотношениями*. Если в полугруппе нет определяющих соотношений и все различные слова, составленные из образующих, суть различные элементы носителя, то полугруппа называется *свободной*. Всякая конечно-порождённая полугруппа может быть получена из свободной введением определяющих соотношений. Пусть в конечно-порождённой полугруппе некоторый элемент $\bar{\alpha}$ определяется словом α, причём $\alpha = \beta\gamma\delta$ и задано определяющее соотношение $\gamma = \sigma$. Тогда вхождение слова γ в слово α можно заменить словом σ, причём полученное слово $\beta\sigma\delta$ будет определять тот же самый элемент $\bar{\alpha}$. Такое преобразование слова будем называть *применением* определяющего соотношения. Два слова в алфавите A считаются равными, если одно из другого получается применением определяющих соотношений, то есть слова равны, если равны определяемые ими элементы.

Отношение равенства слов в полугруппе с определяющими соотношениями является отношением эквивалентности. Классы эквивалентности по этому отношению соответствуют элементам полугруппы.

Примеры

1. В полугруппе $\langle \mathbb{N}; + \rangle$ имеется конечная система образующих $\{1\}$. Другими словами, каждое натуральное число можно представить как последовательность знаков 1. Очевидно, что различные слова в алфавите $\{1\}$ суть различные элементы носителя, то есть эта полугруппа свободна.
2. Пусть полугруппа $\mathcal{P}$ задана системой двух образующих $\{a, b\}$ и двумя определяющими соотношениями: $aa = a$ и $bb = b$. Следующий элементарный алгоритм определяет, равны ли два слова, α и β, в полугруппе $\mathcal{P}$.

Вход: входные слова α : **array** $[1..s]$ **of** $\{a, b\}$ и β : **array** $[1..t]$ **of** $\{a, b\}$.
Выход: значение выражения $\alpha = \beta$ в полугруппе $\mathcal{P}$.

```
if α[1] ≠ β[1] then
  return false  // первые буквы не совпадают
end if
N_α := 0  // счётчик изменений буквы в α
for i from 2 to s do
  if α[i] ≠ α[i − 1] then
    N_α := N_α + 1  // буква изменилась
  end if
end for
N_β := 0  // счётчик изменений буквы в β
for i from 2 to t do
  if β[i] ≠ β[i − 1] then
    N_β := N_β + 1  // буква изменилась
  end if
end for
return N_α = N_β  // слова равны, если значения счётчиков одинаковы
```

В предыдущем примере определяющие соотношения таковы, что равенство любых слов как элементов легко проверяется. Однако это не всегда так.

ТЕОРЕМА (Маркова[1]–Поста[2]). *Существует полугруппа, в которой проблема распознавания равенства слов алгоритмически неразрешима.*

Доказательство. Без доказательства. □

ЗАМЕЧАНИЕ

Термин «алгоритмическая неразрешимость» принадлежит теории алгоритмов, которая рассматривается в этом учебнике в главе 4, а потому строгое определение этого термина здесь не приводится. Неформально можно сказать, что проблема называется *алгоритмически неразрешимой*, если не существует программы (алгоритма) её решения (см. п. 4.5.7). Здесь словосочетание «не существует» означает не то, что программа ещё не составлена, а то, что требуемая программа не может быть составлена в принципе.

Пример. Г. С. Цейтин[3] нашел легко описываемый пример полугруппы, в которой проблема равенства слов алгоритмически неразрешима. В полугруппе с пятью образующими $\{a, b, c, d, e\}$ и семью определяющими соотношениями: $ac = ca$, $ad = da$, $bc = cb$, $bd = db$, $abac = abace$, $eca = ae$, $edb = be$ невозможно составить программу, которая бы для любых заданных слов в алфавите $\{a, b, c, d, e\}$ проверяла, можно ли преобразовать одно слово в другое применением указанных определяющих соотношений.

ОТСТУПЛЕНИЕ

Некоторые программисты полагают, что если в условиях задачи всё дискретно и конечно, то для решения такой задачи программу можно составить в любом случае (используя метод «полного перебора»). Предшествующие теорема и пример показывают, что это мнение ошибочно.

2.2.3. Системы подстановок термов

Слова в полугруппе — это выражения, построенные с помощью одной ассоциативной бинарной операции. Как указано в предыдущем параграфе, зная правила преобразований выражений (определяющие соотношения), даже в таком простом случае, как полугруппа, не всегда возможно решить проблему равенства выражений. Тем более трудной является проверка равенства более сложных выражений.

В данном параграфе рассматриваются *системы подстановок термов*, применение которых позволяет в некоторых случаях это сделать.

Системой подстановок термов (или *системой правил переписывания*) называется пара $\langle A; \rightsquigarrow \rangle$, где A — множество, элементы которого обычно называют *термами*, а $\rightsquigarrow$ — отношение на множестве A, именуемое *отношением переписывания*, или *отношением редукции*. Если $a \rightsquigarrow b$, то говорят, что терм a переписывается в терм b *за один шаг*. Говорят, что терм a *переписывается* в терм b, если существует последовательность термов $t_0, \ldots, t_n$, такая, что $a = t_0$, $t_n = b$ и $t_0 \rightsquigarrow t_1 \rightsquigarrow \cdots \rightsquigarrow t_n$.

[1] Андрей Андреевич Марков (1903–1979).

[2] Эмиль Леон Пост (1897–1954).

[3] Григорий Самуилович Цейтин (род. 1936).

ЗАМЕЧАНИЕ

Обычно считают, что терм переписывается в себя, и в качестве отношения переписывания рассматривают рефлексивное транзитивное замыкание, обозначаемое $\overset{*}{\rightsquigarrow}$.

Говорят, что терм $x \in A$ может быть *редуцирован*, если $\exists\, y \in A, y \neq x \ \left(x \overset{*}{\rightsquigarrow} y\right)$. В противном случае терм x называется *нередуцируемым*. Система подстановок термов называется *терминирующей* (или *остановочной*), если не существует *бесконечной* последовательности термов $x_1, x_2, \ldots$, такой, что $x_1 \rightsquigarrow x_2 \rightsquigarrow \ldots$.

ЗАМЕЧАНИЕ

В терминирующей системе любой терм можно переписать в некоторый нередуцируемый терм.

Пример. В примере 2 из п. 2.2.2 множество термов $A = \{a, b\}^*$ состоит из всех слов в алфавите $\{a, b\}$, а система $aa \rightsquigarrow a$ и $bb \rightsquigarrow b$ является терминирующей. Нередуцируемыми являются все слова, в которых нет двух одинаковых букв подряд.

Утверждение о том, что если любой терм можно переписать в некоторый нередуцируемый, то система терминирующая — неверно.

Пример. Рассмотрим систему $A = \{a, b, c\}, a \rightsquigarrow b, b \rightsquigarrow a, a \rightsquigarrow c, b \rightsquigarrow c$. Любой терм в такой системе можно переписать в терм c, который нередуцируем. Однако последовательность $a \rightsquigarrow b \rightsquigarrow a \rightsquigarrow b \rightsquigarrow \ldots$ очевидно бесконечна.

Система подстановок термов называется *конфлюэнтной*, если
$\forall\, x \in A \ \left(\exists\, u, v \in A \ \left(x \overset{*}{\rightsquigarrow} u \ \&\ x \overset{*}{\rightsquigarrow} v\right) \Longrightarrow \exists\, y \in A \ \left(u \overset{*}{\rightsquigarrow} y \ \&\ v \overset{*}{\rightsquigarrow} y\right)\right)$.

Пример. Система $A = \{a, b, c, d\}, a \rightsquigarrow b, b \rightsquigarrow a, a \rightsquigarrow c, b \rightsquigarrow d$ не конфлюэнтна, а система предыдущего примера — конфлюэнтна.

Терм y называется *нормальной формой* терма x, если $x \overset{*}{\rightsquigarrow} y$ и терм y не редуцируемый. В терминирующей системе все термы имеют нормальные формы, однако не обязательно единственные.

Пример. В системе $A = \{a, b, c, d\}, a \rightsquigarrow b, b \rightsquigarrow a, a \rightsquigarrow c, b \rightsquigarrow d$ оба терма a и b имеют по две нормальных формы c и d.

Говорят, что термы x и y имеют общую нормальную форму (и пишут $x \downarrow y$), если $\exists\, z \in A \ \left(x \overset{*}{\rightsquigarrow} z \ \&\ y \overset{*}{\rightsquigarrow} z\right)$, причём терм z нередуцируемый.

Пусть $\overset{*}{\leftrightarrow}$ — рефлексивное симметричное транзитивное замыкание отношения $\rightsquigarrow$. Тогда система подстановок термов обладает *свойством Чёрча–Россера*: если $x \overset{*}{\leftrightarrow} y$ тогда и только тогда, когда $x \downarrow y$.

В системе со свойством Чёрча–Россера все термы имеют единственную нормальную форму.

Другими словами, система подстановок термов обладает свойством Чёрча–Россера, если термы преобразуются друг в друга по правилам переписывания тогда и только тогда, когда у них есть общая нормальная форма.

Свойство Чёрча–Россера — очень сильное, редкое и полезное свойство. Действительно, в случае наличия этого свойства, для проверки равенства термов достаточно свести обе части проверяемого равенства к нормальной форме. Если результаты совпадут, то равенство имеет место, в противном случае не имеет места. Однако проверка свойства Чёрча–Россера является алгоритмически неразрешимой задачей.

ТЕОРЕМА. *Терминирующая система подстановок термов обладает свойством Чёрча–Россера тогда и только тогда, когда она конфлюэнтна.*

Доказательство. Без доказательства. □

Теорема имеет важное значение, поскольку конфлюэнтность системы в некотором смысле проще проверять, чем свойство Чёрча–Россера, и, что важнее, проще построить систему с таким свойством.

2.2.4. Алгоритм Кнута–Бендикса

Известен *алгоритм Кнута–Бендикса*, который по заданной системе подстановок термов (в случае успешного завершения!) строит эквивалентную терминирующую конфлюэнтную систему подстановок термов. Для простоты алгоритм излагается на примере полугрупп с конечной системой образующих (п. 2.1.3) и конечным набором определяющих соотношений (п. 2.2.2). В такой постановке термами являются слова в алфавите образующих (п. 1.1.5).

Пусть задан конечный алфавит A (образующие) и конечный набор пар слов $R = \{(\lambda, \rho) \mid \lambda \in A^* \ \& \ \rho \in A^*\}$ (определяющие соотношения). На словах определён линейный порядок (п. 1.8.2), не ограничивая общности, этот порядок можно считать лексикографическим.

Основная идея алгоритма вытекает из следующих наблюдений.

1. Если использовать определяющее соотношение «в обе стороны» $\alpha \rightsquigarrow \beta$ и $\beta \rightsquigarrow \alpha$, то система заведомо не будет терминирующей, а потому такое использование надобно запретить и «ориентировать» соотношения в одну сторону.
2. Определяющее соотношение $\alpha = \beta$ следует ориентировать так, чтобы было $|\alpha| > |\beta|$. Действительно, в таком случае в цепочке преобразований длина слова всё время уменьшается, а значит, любая цепочка обрывается и система является терминирующей.
3. Термы в обеих частях правил должны быть нередуцируемыми с помощью других правил. В противном случае надо провести редукции и упростить исходные правила.
4. Если в системе есть тривиальные правила, заданные исходно или полученные в результате редукции правил, то их следует исключить.
5. Нарушение конфлюэнтности может происходить только в случае наличия правил с пересекающимися левыми частями. Значит, такие правила необходимо устранить, заменяя эквивалентными правилами.

Важной операцией в алгоритме является выделение пересекающихся пар слов. Пересечение слов может быть двух видов: либо суффикс одного слова есть префикс другого (слова $\alpha\beta$ и $\beta\gamma$), либо одно слово — подстрока другого слова ($\alpha\beta\gamma$ и β). В обоих случаях при пересечении однозначно выделяются три слова α, β, γ, которые не могут быть одновременно все три пустыми.

Пример. В строках $abab$ и $baba$ имеем $\alpha = a$, $\beta = bab$ и $\gamma = b$.

Основной структурой данных является множество пар слов $R = \{(\lambda, \rho)\}$, то есть набор правил переписывания $\lambda \rightsquigarrow \rho$. Вначале это исходные определяющие соотношения, а в конце, возможно, конфлюэнтная система. Алгоритм может не закончить работу, если исходня система не обладает свойством Чёрча–Россера.

Вспомогательная процедура Clear проводит «чистку» множества пар слов R, а именно применяет все возможные редукции к каждому правилу, и если правило редуцируется к тривиальному, то удаляет его.

Пример. Если $R = \{aa \rightsquigarrow bb, a \rightsquigarrow b\}$, то после применения процедуры Clear имеем $R = \{a \rightsquigarrow b\}$.

Вспомогательная процедура Conf находит в R два правила, левые части которых пересекаются, и возвращает соответствующие слова α, β, γ, а также результаты редукции слова $\alpha\beta\gamma$ по первому правилу (ρ_1) и по второму правилу (ρ_2).

Пример. $\mathrm{Conf}(\{abab \rightsquigarrow c, baba \rightsquigarrow d\}) = (\alpha = a, \beta = bab, \gamma = b, \rho_1 = ca, \rho_2 = ad)$.

Алгоритм 2.1. Алгоритм Кнута–Бендикса

Вход: R — множество пар слов определяющих соотношений.
Выход: R — множество пар слов правил конфлюэнтной системы.

```
for (λ, ρ) ∈ R do
  if |λ| < |ρ| then λ :=: ρ end if    // чтобы левая часть была больше правой
end for
while true do
  Clear // чистка текущего множества правил
  (α, β, γ, ρ1, ρ2) := Conf // ищем пару пересекающихся правил
  if α = β = γ = ε then stop end if // нет конфликтующих правил
  if ρ1 ≠ ρ2 then
    R := R + (max(ρ1, ρ2), min(ρ1, ρ2)) //добавляем новую пару
  end if
end while
```

Обоснование. Построенный алгоритм строит конфлюэнтную и терминирующую систему, что следует из наблюдений, лежащих в основе алгоритма. Кроме того, построенная система R' является следствием исходного набора соотношений R. Действительно, возьмем любые α и β такие, что $\alpha \overset{*}{\rightsquigarrow}_R \beta$, и рассмотрим последовательность определяющих соотношений (правил), при применении которых $\alpha \overset{*}{\rightsquigarrow}_R \beta$. Если какое-то из правил не входит в R', то значит, оно было «удалено» в процедуре Clear, то есть при помощи применения некоторых редукций из R' данное правило сводится к тривиальному. Тогда применение этого правила можно заменить на применение некоторого количества правил из R'. Таким образом, $\alpha \overset{*}{\rightsquigarrow}_{R'} \beta$. □

Пример. Пусть $X = \{a, b\}$, $R = \{aa = \varepsilon, ab = \varepsilon\}$. В таблице отображены состояния множества R после каждой итерации основного цикла в алгоритме и результаты выполнения процедур Clear и Conf.

R	Clear	Conf
$aa \rightsquigarrow \varepsilon, ab \rightsquigarrow \varepsilon$	$\varnothing$	$b \rightsquigarrow a$
$aa \rightsquigarrow \varepsilon, ab \rightsquigarrow \varepsilon, b \rightsquigarrow a$	$ab \rightsquigarrow \varepsilon$	$\varnothing$
$aa \rightsquigarrow \varepsilon, b \rightsquigarrow a$	$\varnothing$	$\varnothing$

Пример. Пусть $X = \{a, b\}$, $R = \{abab \rightsquigarrow ab, baba \rightsquigarrow ba\}$. При применении алгоритма на каждом шаге получаем тривиальные правила переписывания $aba \rightsquigarrow aba$ или $ababa \rightsquigarrow ababa$. Соответственно, искомую конфлюэнтную систему подстановок термов не удаётся построить.

Таким образом, приведённый алгоритм не является универсальным. Если алгоритм Кнута–Бендикса успешно завершается, то в данной полугруппе можно решить проблему равенства слов. Если же алгоритм не завершается, то это ничего не означает.

ЗАМЕЧАНИЕ

Применение описанного алгоритма к полугруппе из п. 2.2.2. не приводит к построению конфлюэнтной терминирующей системы подстановок термов.

2.2.5. Моноиды

Моноид — это полугруппа с *единицей*: $\exists\, e\ \ (\forall\, a\ \ (a * e = e * a = a))$.

ЗАМЕЧАНИЕ

Единицу также называют *нейтральным элементом*.

Примеры

1. Множество слов A^* в алфавите A вместе с операцией конкатенации и пустым словом ε образует моноид.
2. Пусть T — множество термов над множеством переменных V и сигнатурой Σ. *Подстановкой*, или *заменой переменных*, называется множество пар $\sigma = [t_i/v_i]_{i \in I}$, где t_i — термы, а v_i — переменные. Результатом применения подстановки σ к терму t (обозначается $t\sigma$) называется терм, который получается заменой *всех* вхождений переменных v_i соответствующими термами t_i. *Композицией* подстановок $\sigma_1 = [t_i/v_i]_{i \in I}$ и $\sigma_2 = [t_j/v_j]_{j \in J}$ называется подстановка $\sigma_1 \circ \sigma_2 := [t_k/v_k]_{k \in K}$, где $K = I \cup J$, а $t_k =$ **if** $k \in I$ **then** $t_i\sigma_2$ **else** t_j. Множество подстановок образует моноид относительно композиции, причём тождественная подстановка $[v_i/v_i]$ является единицей.

ТЕОРЕМА 1. *Единица моноида единственна.*

Доказательство. Пусть существуют две единицы: e_1, e_2.
Тогда $e_1 * e_2 = e_1$ & $e_1 * e_2 = e_2 \Longrightarrow e_1 = e_2$. □

Функция $f\colon M \to M$ называется *преобразованием над множеством* M. Суперпозиция функций (п. 1.6.4) ассоциативна (п. 1.4.5), а тождественное преобразование является нейтральным элементом, поэтому всякое множество преобразований, замкнутое относительно суперпозиции, образует моноид.

ТЕОРЕМА 2. *Всякий моноид над M изоморфен некоторому моноиду преобразований над M.*

Доказательство. Пусть $\mathcal{M} = \langle M; * \rangle$ — моноид над M с единицей e. Построим $\mathcal{F} = \langle F; \circ \rangle$ — моноид преобразований над M, где $F := \{f_m : M \to M \mid f_m(x) := x * m\}_{m \in M}$, а $\circ$ — суперпозиция функций, $h : M \to F$, $h(m) := f_m$. Тогда $\mathcal{F}$ — моноид, поскольку суперпозиция функций ассоциативна, f_e — тождественная функция (так как $f_e(x) = x * e = x$) и F замкнуто относительно $\circ$, так как $f_a \circ f_b = f_{a*b}$: $f_{a*b}(x) = x * (a * b) = (x * a) * b = f_a(x) * b = f_b(f_a(x))$. Далее, h — гомоморфизм, так как $h(a * b) = f_{a*b} = f_a \circ f_b = h(a) \circ h(b)$. И наконец, h — биекция, поскольку h — сюръекция по построению, и h — инъекция (так как $(f_a(e) = e * a = a \;\&\; f_b(e) = e * b = b) \Longrightarrow (a \neq b \Longrightarrow f_a \neq f_b)$). □

2.2.6. Группы

Теория групп, некоторые положения которой рассматриваются в этом параграфе, является ярчайшим примером мощи алгебраических методов, позволяющих получить из немногих базовых понятий и аксиом множество важнейших следствий.

Группа — это моноид, в котором $\forall a \;\left(\exists a^{-1} \;\left(a * a^{-1} = a^{-1} * a = e\right)\right)$. Элемент a^{-1} называется *обратным* для элемента a.

ЗАМЕЧАНИЕ

Так как операция умножения в группе не обязательно коммутативна, то, вообще говоря, свойства $a^{-1} * a = e$ и $a * a^{-1} = e$ не равносильны. Можно было бы различать обратные слева и обратные справа элементы. Введенная аксиома требует существования двустороннего обратного элемента. Можно показать, что она следует из более слабых аксиом существования левой единицы и левого обратного элемента.

Мощность носителя G группы $\mathcal{G}$ называется *порядком группы* и обозначается $|G|$.

Примеры

1. Множество невырожденных квадратных матриц порядка n образует группу относительно операции умножения матриц. Единицей группы является единичная матрица. Обратным элементом является обратная матрица.
2. Множество перестановок на множестве M, то есть множество взаимно однозначных функций $f : M \to M$, является группой относительно операции суперпозиции. Единицей группы является тождественная функция, а обратным элементом — обратная функция.
3. Множество поворотов правильного треугольника относительно его центра на $0°, 120°, 240°$ образует группу поворотов, переводящих треугольник в себя. На рис. 2.3 приведена «таблица умножения» группы и геометрическая иллюстрация.

ТЕОРЕМА 1. *Обратный элемент единственен.*

Доказательство. Пусть $a * a^{-1} = a^{-1} * a = e \;\&\; a * b = b * a = e$.
Тогда $a^{-1} = a^{-1} * e = a^{-1} * (a * b) = (a^{-1} * a) * b = e * b = b$. □

	0°	120°	240°
0°	0°	120°	240°
120°	120°	240°	0°
240°	240°	0°	120°

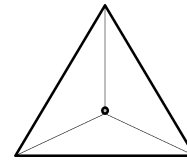


Рис. 2.3. Множество поворотов правильного треугольника относительно его центра

ТЕОРЕМА 2. *В любой группе выполняются следующие соотношения:*

1) $(a*b)^{-1} = b^{-1}*a^{-1}$;
2) $a*b = a*c \Longrightarrow b = c$;
3) $b*a = c*a \Longrightarrow b = c$;
4) $(a^{-1})^{-1} = a$.

Доказательство.

[1] $(a*b)*(b^{-1}*a^{-1}) = a*(b*b^{-1})*a^{-1} = a*e*a^{-1} = a*a^{-1} = e$.

[2] $b = e*b = (a^{-1}*a)*b = a^{-1}*(a*b) = a^{-1}*(a*c) = (a^{-1}*a)*c = = e*c = c$.

[3] $b = b*e = b*(a*a^{-1}) = (b*a)*a^{-1} = (c*a)*a^{-1} = c*(a*a^{-1}) = = c*e = c$.

[4] $a^{-1}*a = e$. □

ТЕОРЕМА 3. *В группе можно однозначно решить уравнение* $a*x = b$ *(решение:* $x = a^{-1}*b$*).*

Доказательство. $a*x = b \Longrightarrow a^{-1}*(a*x) = a^{-1}*b \Longrightarrow (a^{-1}*a)*x = a^{-1}*b \Longrightarrow e*x = a^{-1}*b \Longrightarrow x = a^{-1}*b$. □

Коммутативная группа, то есть группа, в которой $\forall\, a, b\ \ (a*b = b*a)$, называется *абелевой*. В абелевых группах обычно приняты следующие обозначения: групповая операция обозначается $+$, обратный элемент к a обозначается $-a$, единица группы обозначается 0 и называется *нулем* или *нейтральным элементом*.

Примеры

1. $\langle \mathbb{Z}; + \rangle$ — множество целых чисел образует абелеву группу относительно сложения. Нейтральным элементом группы является число 0. Обратным элементом является число с противоположным знаком: $x^{-1} := -x$.
2. $\langle \mathbb{Q}_+; \cdot \rangle$ — множество положительных рациональных чисел образует абелеву группу относительно умножения. Нейтральным элементом группы является число 1. Обратным элементом является обратное число: $(m/n)^{-1} := n/m$.
3. $\langle 2^M; \triangle \rangle$ — булеан образует абелеву группу относительно симметрической разности. Нейтральным элементом группы является пустое множество $\varnothing$. Обратным элементом к элементу x является он сам: $x^{-1} := x$.

2.2.7. Действие группы на множестве

Пусть S — множество, а G — группа с операцией $*$ и нейтральным элементом e. *Действием группы G на множестве S* называется отображение $\phi\colon G \times S \to S$ такое, что $\forall\, g_1, g_2 \in G, s \in S\ \ (\phi(g_1 * g_2, s) = \phi(g_1, \phi(g_2, s)))$ и $\forall\, s \in S\ \ (\phi(e, s) = s)$. Для краткости записи принято опускать знак операции $*$ и имя отображения ϕ, обозначая $\phi(g, s)$ просто gs. В таких обозначениях

$$\forall\, g_1, g_2 \in G, s \in S\ \ ((g_1 g_2)s = g_1(g_2 s)) \quad \text{и} \quad \forall\, s \in S\ \ (es = s)\,.$$

В этом случае также говорят, что группа G *действует* на множестве S или что S есть G-множество.

Если $s \in S$, то множество $Gs \stackrel{\text{Def}}{=} \{t \in S \mid t = gs\ \&\ g \in G\}$, то есть множества всех элементов из S вида gs, где g «пробегает» носитель группы G, называется *орбитой* (или *траекторией*) элемента s в G-множестве S.

Пример. Пусть S_1 — единичная окружность, а $\varphi \in [0; 2\pi]$ — некоторый угол. Определим действие группы $\langle \mathbb{Z}; + \rangle$ на множестве S_1 следующим образом: число $n \in Z$ действует на точку $x \in S_1$ как поворот на угол $n\varphi$. Положительные углы отсчитываются против часовой стрелки, отрицательные — по часовой стрелке. Тогда если угол φ соизмерим с числом π, то все орбиты состоят из конечного числа точек. Если же отношение φ/π иррационально, то все орбиты бесконечны.

ЛЕММА. *Если G — группа, то $\forall\, g \in G\ \ (Gg = G)$.*

Доказательство. Ясно, что $\forall\, g \in G\ \ (Gg \subset G)$. Пусть $g_1 \in G$.
Тогда $\forall\, g \in G\ \ (\exists\, g_2 \in G\ \ (g_2 g = g_1))$ по теореме п. 2.2.6, $g_1 \in Gg$ и $G \subset Gg$. □

ТЕОРЕМА. *Если группа G действует на множестве S, то орбиты не пусты и образуют разбиение множества S.*

Доказательство. Каждый элемент принадлежит своей орбите, поэтому орбиты не пусты и образуют покрытие (п. 1.2.7). Пусть две орбиты Gs_1 и Gs_2 пересекаются, то есть $\exists\, s \in S\ \ (s \in Gs_1\ \&\ s \in Gs_2)$. Тогда $\exists\, g_1 \in G\ \ (s = g_1 s_1)$ и по лемме $Gs = Gg_1 s_1 = Gs_1$. Аналогично $\exists\, g_2 \in G\ \ (s = g_2 s_2)$ и $Gs = Gg_2 s_2 = Gs_2$, откуда $Gs_1 = Gs_2$. □

Таким образом, действие группы на множестве определяет отношение эквивалентности на этом множестве (п. 1.7.1).

Пример. Каждая группа G действует на самой себе с помощью умножения. Это действие имеет всего одну орбиту.

Задать действие группы на множестве — всё равно, что рассматривать элементы группы как преобразования множества.

2.2.8. Группа перестановок

В этом параграфе рассматривается одна из важнейших групп, называемая группой *перестановок*, или *симметрической* группой.

Биективная функция $f\colon X \to X$ называется *перестановкой* множества X.

ЗАМЕЧАНИЕ

Если множество X конечно ($|X| = n$), то, не ограничивая общности, можно считать, что $X = 1..n$. В этом случае перестановку $f\colon 1..n \to 1..n$ удобно задавать таблицей из двух строк. В первой строке — значения аргументов, во второй — соответствующие значения функции. Такая таблица называется *подстановкой*. В сущности, перестановка и подстановка — синонимы.

Пример.

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 1 & 4 & 3 \end{vmatrix}, \qquad g = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 2 & 3 & 5 \end{vmatrix}.$$

Произведением перестановок f и g (обозначается fg) называется их суперпозиция $g \circ f$.

Пример.

$$fg = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 4 & 3 & 2 \end{vmatrix}.$$

Тождественная перестановка — это перестановка e, такая, что $e(x) = x$.

Пример.

$$e = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{vmatrix}.$$

Обратная перестановка — это обратная функция, которая всегда существует, поскольку перестановка является биекцией.

ЗАМЕЧАНИЕ

Таблицу обратной подстановки можно получить, если просто поменять местами строки таблицы исходной подстановки.

Пример.

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{vmatrix}, \qquad f^{-1} = \begin{vmatrix} 3 & 4 & 2 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{vmatrix} = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 1 & 2 & 5 \end{vmatrix}.$$

Таким образом, поскольку суперпозиция функций ассоциативна, а единичный и обратный элементы существуют, множество перестановок n-элементного множества образует *группу перестановок* относительно операции суперпозиции. Эта группа называется также *симметрической группой* порядка n и обозначается S_n.

ТЕОРЕМА. *Всякая конечная группа G порядка n изоморфна некоторой подгруппе симметрической группы S_n.*

Доказательство. Пусть G — группа порядка n с нейтральным элементом e. Рассмотрим действие группы G на множестве G с помощью умножения (п. 2.2.7). Каждый элемент $g \in G$ задаёт подстановку $f_g \in S_n$. Рассмотрим множество $H := \{f_g : G \to G \mid f_g(x) := x * g \ \&\ g \in G\} \subset S_n$. Ясно, что отображение $g \mapsto f_g$ биективно и уважает умножение, так что $G \sim H$. Покажем, что H — подгруппа группы S_n. Действительно, H замкнуто относительно $\circ$,
так как $f_{g*h}(x) = x * (g * h) = (x * g) * h = f_g(x) * h = f_h(f_g(x))$;
f_e — тождественная функция, так как $f_e(x) = x * e = x$,
$f_{g^{-1}}$ — обратная функция, так как $f_g\left(f_{g^{-1}}(x)\right) = (x * g^{-1}) * g = x$. □

Пример. Группа поворотов правильного треугольника (см. пример 3 в п. 2.2.6) изоморфна следующей подгруппе группы S_3:

$$e = \begin{vmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{vmatrix}, \qquad f = \begin{vmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{vmatrix}, \qquad f^{-1} = \begin{vmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{vmatrix}.$$

ЗАМЕЧАНИЕ

Не следует думать, что описание подгрупп симметрической группы является простой задачей. Это реально только для небольших n.

2.2.9. Перестановочные матрицы

Если $f : 1..n \to 1..n$ — перестановка, то *матрицей подстановки*, или *перестановочной матрицей*, называется квадратная булева матрица P_f : **array** $[1..n, 1..n]$ **of** 0..1, такая, что $P_f[i, j] := (j = f(i))$.

Нетрудно видеть, что в каждой строке и в каждом столбце перестановочной матрицы содержится ровно одна единица, остальные элементы — нули. Более того, всякой булевой матрице P, у которой в каждой строке и в каждом столбце содержится ровно одна единица, соответствует единственная перестановка f, которая получится, если матрицу P_f умножить справа на вектор-столбец $(1, \ldots, n)$.

Пример. $f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{vmatrix}, \qquad P_f = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$

Введем обозначение: δ_i^n — булевский вектор длины n, в котором в i-м разряде находится единица, а в остальных разрядах находятся нули. В этих обозначениях для каждой строки перестановочной матрицы $P_f[i, 1..n] = \delta^n_{f(i)}$, а для каждого столбца $P_f[1..n, j] = \delta^n_{f^{-1}(j)}$.

ЛЕММА. *Множество перестановочных матриц одного порядка образует некоммутативную группу относительно операции умножения матриц.*

ЗАМЕЧАНИЕ

Хотя перестановочные матрицы булевы, умножение матриц может производиться с помощью обычных арифметических, а не логических операций.

Доказательство. Произведение перестановочных матриц является перестановочной матрицей, поскольку *скалярное произведение*

$$\langle \delta_i^n \mid \delta_j^n \rangle \stackrel{\text{Def}}{=} \sum_{k=1}^{n} \delta_i^n[k] \cdot \delta_j^n[k] \stackrel{\text{Def}}{=} \textbf{ if } i = j \textbf{ then } 1 \textbf{ else } 0 \textbf{ end if},$$

то есть произведение является булевой матрицей, а также $P_f \times \delta_i^n = \delta_{f(i)}^n$, то есть каждая строка произведения содержит ровно одну единицу, и аналогично для столбцов. Далее, умножение перестановочных матриц ассоциативно в силу ассоциативности умножения квадратных матриц. Единичная матрица является перестановочной и служит нейтральным элементом. Транспонирование перестановочной матрицы даёт перестановочную матрицу, которая является обратной, то есть $P_f P_f^{\mathrm{T}} = I_n$, где I_n — единичная матрица. □

Обозначим группу перестановочных матриц порядка n как P_n.

Теорема. *Группа перестановочных матриц P_n изоморфна симметрической группе S_n.*

Доказательство. В п. 5.1.4 показано, что $|S_n| = n!$, и по правилу произведения (см. п. 5.1.1) имеем $|P_n| = n!$, поскольку единицу в первой строке можно выбрать n способами, единицу во второй строке можно выбрать $n-1$ способами и т. д. Отображение $f \mapsto P_f$, определённое в начале параграфа, является биекцией. Осталось показать, что это отображение является гомоморфизмом: $P_{fg} = P_f P_g$. Имеем: $(P_f P_g)[i,j] = 1 \iff \sum\limits_{k=1}^{n} P_f[i,k] \cdot P_g[k,j] = 1 \iff \left\langle \delta_{f(i)}^n \mid \delta_{g^{-1}(j)}^n \right\rangle = 1 \iff$
$\iff f(i) = g^{-1}(j) \iff g(f(i)) = j \iff P_{fg}[i, g(f(i))] = 1.$ □

Значение перестановочных матриц состоит в том, что это эффективное *представление* перестановки, позволяющее свести применение перестановки как функции к операциям с булевыми матрицами. В частности, если $(a_1, \ldots, a_n)$ — любой числовой n-вектор (не обязательно перестановка $1..n$), а P_f — перестановочная матрица для перестановки f, то $P_f \times (a_1, \ldots, a_n)^T = (a_{f(1)}, \ldots, a_{f(n)})^T$.

2.3. Алгебры с двумя операциями

В этом разделе мы обращаемся к объектам, знакомым читателю со школы. Среди алгебр с двумя операциями наиболее важными являются кольца и поля, а основными примерами колец и полей являются множества целых, рациональных и вещественных чисел с операциями сложения и умножения.

2.3.1. Кольца

Кольцо — это множество M с двумя бинарными операциями $+$ и $*$ (они называются сложением и умножением соответственно), в котором

1) $(a+b)+c = a+(b+c)$ — сложение ассоциативно;
2) $\exists 0 \in M \ \ (\forall a \ \ (a+0 = 0+a = a))$ — существует нуль;
3) $\forall a \ \ (\exists -a \ \ (a+(-a) = 0))$ — существует обратный элемент;
4) $a+b = b+a$ — сложение коммутативно, то есть кольцо — абелева группа по сложению;

5) $a*(b*c)=(a*b)*c$ — умножение ассоциативно, то есть кольцо — полугруппа по умножению;

6) $a*(b+c)=(a*b)+(a*c)$, $(a+b)*c=(a*c)+(b*c)$ — умножение дистрибутивно относительно сложения слева и справа.

Кольцо называется *коммутативным*, если $a*b=b*a$ — умножение коммутативно.

Кольцо называется кольцом *с единицей*, если $\exists 1 \in M \ (a*1=1*a=a)$ — существует единица, то есть кольцо с единицей — моноид по умножению.

ТЕОРЕМА. *В кольце выполняются следующие соотношения:*

1) $0*a=a*0=0$;
2) $a*(-b)=(-a)*b=-(a*b)$;
3) $(-a)*(-b)=a*b$;
4) $(-a)=a*(-1)$;
5) $-(a+b)=(-a)+(-b)$;
6) $a\neq 0 \Longrightarrow (a^{-1})^{-1}=a$.

Доказательство.

[1] $0*a=(0+0)*a=(0*a)+(0*a) \Longrightarrow$
$0=-(0*a)+(0*a)=-(0*a)+\big((0*a)+(0*a)\big)=\big(-(0*a)+(0*a)\big)+(0*a) \Longrightarrow$
$0=0+(0*a)=0*a$.

[2] $\big(a*(-b)\big)+(a*b)=a*(-b+b)=a*0=0$,
$(a*b)+\big((-a)*b\big)=\big(a+(-a)\big)*b=0*b=0$.

[3] $(-a)*(-b)=-\big(a*(-b)\big)=-\big(-(a*b)\big)=a*b$.

[4] $\big(a*(-1)\big)+a=\big(a*(-1)\big)+(a*1)=a*(-1+1)=a*0=0$.

[5] $(a+b)+\big((-a)+(-b)\big)=(a+b)+\big((-b)+(-a)\big)=$
$=a+\big(b+(-b)\big)+(-a)=a+0+(-a)=a+(-a)=0$.

[6] $a^{-1}*a=1$. □

Пример. $\langle\mathbb{Z};+,*\rangle$ — коммутативное кольцо с единицей. Кроме того, $\forall n\in\mathbb{N}\ (\langle\mathbb{Z}_n;+,*\rangle)$ — коммутативное кольцо с единицей. В частности, машинная арифметика целых чисел $\langle\mathbb{Z}_{2^{15}};+,*\rangle$ — коммутативное кольцо с единицей.

2.3.2. Области целостности

Если в кольце для некоторых ненулевых элементов x, y выполняется равенство $x*y=0$, то x называется *левым*, а y — *правым делителем нуля*.

Пример. В машинной арифметике $\langle\mathbb{Z}_{2^{15}};+,*\rangle$ имеем $256*128=2^8*2^7=2^{15}=0$.

Заметим, что если в кольце нет делителей нуля, то $\forall x\neq 0, y\neq 0\ (x*y\neq 0)$. В группе $\forall a,b,c\ \big((a*b=a*c\Longrightarrow b=c)\ \&\ (b*a=c*a\Longrightarrow b=c)\big)$, однако в произвольном кольце это не так.

Пример. В машинной арифметике $\langle \mathbb{Z}_{2^{15}}; +, * \rangle$ имеем $2^8 * 2^7 = 0$ и $0 * 2^7 = 0$, но $0 \neq 2^8 = 256$.

ТЕОРЕМА. $\forall a \neq 0 \left(a * b = a * c \Longrightarrow b = c\right) \;\&\; \left(b * a = c * a \Longrightarrow b = c\right) \Longleftrightarrow$
$\Longleftrightarrow \forall\, x{\neq}0, y \neq 0 \;\; (x * y \neq 0)$.

Доказательство.

$[\Longrightarrow]$ От противного. Пусть $\exists\, x \neq 0, y \neq 0 \;\; (x * y = 0)$.
Тогда $x \neq 0 \;\&\; x * y = 0 \;\&\; x * 0 = 0 \Longrightarrow y = 0$.

$[\Longleftarrow]$ $0 = (a*b) + \big(-(a*b)\big) = (a*b) + \big(-(a*c)\big) = (a*b) + \big(a*(-c)\big) = a*\big(b+(-c)\big)$,
$a * \big(b + (-c)\big) = 0 \;\&\; a \neq 0 \Longrightarrow b + (-c) = 0 \Longrightarrow b = c$. □

Коммутативное кольцо с единицей, не имеющее делителей нуля, называется *областью целостности*.

Пример. Целые числа $\langle \mathbb{Z}; +, * \rangle$ являются областью целостности, а машинная арифметика $\langle \mathbb{Z}_{2^{15}}; +, * \rangle$ — не является.

2.3.3. Поля

Поле — это множество M с двумя бинарными операциями $+$ и $*$, такими, что

1) $(a + b) + c = a + (b + c)$ — сложение ассоциативно;
2) $\exists 0 \in M \;\; (a + 0 = 0 + a = a)$ — существует нуль;
3) $\forall\, a \;\; (\exists\, -a \;\; (a + -a = 0))$ — существует обратный элемент по сложению;
4) $a + b = b + a$ — сложение коммутативно, то есть поле — абелева группа по сложению;
5) $a * (b * c) = (a * b) * c$ — умножение ассоциативно;
6) $\exists 1 \in M \;\; (a * 1 = 1 * a = a)$ — существует единица;
7) $\forall\, a \neq 0 \;\; \big(\exists\, a^{-1} \;\; \big(a^{-1} * a = 1\big)\big)$ — существует обратный элемент по умножению;
8) $a * b = b * a$ — умножение коммутативно, то есть ненулевые элементы образуют абелеву группу по умножению;
9) $a * (b + c) = (a * b) + (a * c)$ — умножение дистрибутивно относительно сложения.

Таким образом, поле — это коммутативное кольцо с единицей, в котором каждый элемент, кроме нуля, имеет обратный элемент по умножению.

Примеры

1. $\langle \mathbb{R}; +, * \rangle$ — поле вещественных чисел.
2. $\langle \mathbb{Q}; +, * \rangle$ — поле рациональных чисел.
3. Пусть $E_2 \stackrel{\text{Def}}{=} \{0, 1\}$. Определим операции $+, \cdot \colon E_2 \times E_2 \to E_2$ следующим образом: $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$, $1 \cdot 1 = 1$ (конъюнкция), $0 + 0 = 1 + 1 = 0$, $0 + 1 = 1 + 0 = 1$ (сложение по модулю 2). Тогда $\mathbb{Z}_2 \stackrel{\text{Def}}{=} \langle E_2; +, \cdot \rangle$ является полем и называется *двоичной арифметикой*. В двоичной арифметике нуль — это 0, единица — это 1, $-1 = 1^{-1} = 1$, а 0^{-1} — как всегда, не определён.

4. Если в условиях предыдущего примера взять в качестве сложения дизъюнкцию, то есть определить операции следующим образом: $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$, $1 \cdot 1 = 1$ (конъюнкция), $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1 + 1 = 1$ (дизъюнкция), то структура $\langle E_2; \vee, \cdot \rangle$ не является полем, поскольку у элемента 1 нет обратного по сложению.

ТЕОРЕМА 1. *Поле является областью целостности:* $a * b = 0 \Longrightarrow a = 0 \vee b = 0$.

Доказательство. $a * b = 0 \;\&\; a \neq 0 \Longrightarrow b = 1 * b = (a^{-1} * a) * b = a^{-1} * (a * b) = a^{-1} * 0 = 0$, $a * b = 0 \;\&\; b \neq 0 \Longrightarrow a = 1 * a = (b^{-1} * b) * a = b^{-1} * (b * a) = b^{-1} * (a * b) = b^{-1} * 0 = 0$. □

ТЕОРЕМА 2. *Если* $a \neq 0$*, то в поле единственным образом разрешимо уравнение* $a * x + b = 0$ *(решение:* $x = -(a^{-1}) * b$*).*

Доказательство. $a * x + b = 0 \Longrightarrow a * x + b + (-b) = 0 + (-b) \Longrightarrow$
$\Longrightarrow a * x + \big(b + (-b)\big) = -b \Longrightarrow a * x + 0 = -b \Longrightarrow a * x = -b \Longrightarrow$
$\Longrightarrow a^{-1} * (a * x) = a^{-1} * (-b) \Longrightarrow (a^{-1} * a) * x = -(a^{-1} * b) \Longrightarrow 1 * x = -(a^{-1} * b) \Longrightarrow$
$\Longrightarrow x = -(a^{-1} * b)$. □

2.4. Элементарная теория чисел

Целые числа $\mathbb{Z}$ и операции с ними (то есть, *арифметика*) используются везде, во всех предметных областях и языках программирования. Традиционно в арифметике используют по меньшей мере четыре операции: сложение, умножение, вычитание и деление.

Алгебра $\langle \mathbb{Z}; +; * \rangle$ с операциями сложения и умножения является кольцом, и свойства сложения и умножения этим обстоятельством описываются в достаточной мере.

Множество целых чисел $\mathbb{Z}$ с операцией сложения образует абелеву группу (п. 2.2.6), при этом $\forall x \;\; (\exists y \;\; (x + y = 0))$, то есть у любого элемента этого множества существует обратный по сложению элемент, что в достаточной мере описывает свойства операции вычитания.

Множество $\mathbb{Z}$ с операцией умножения образует полугруппу (моноид) (п. 2.2.5), причём обратный элемент по умножению существует лишь для чисел 1 и -1. В связи с этим операция обратная к умножению — деление — является в некотором смысле особенной и требует специфического рассмотрения.

Раздел математики, изучающий числа и операции с ними, называется *теорией чисел* или *высшей арифметикой*.

Та часть теории чисел, в которой целые числа изучаются без использования методов других разделов математики, называется *элементарной теорией чисел*. Многие задачи элементарной теории чисел, известные читателю из школьного курса математики, такие как делимость целых чисел, вычисление наибольшего общего делителя и наименьшего общего кратного, разложение числа на простые множители, вычисления по модулю, нашли в последнее время ряд важнейших приложений в информационных технологиях.

В этом разделе бегло рассматривается элементарная теория чисел, вводятся обозначения и устанавливаются основные факты, необходимые в приложениях. Сами приложения рассматриваются в других курсах по информационным технологиям.

ЗАМЕЧАНИЕ

В изложении рассматриваются натуральные числа $\mathbb{N}$ и ноль только ради упрощения формулировок и доказательств. Все определения, утверждения и алгоритмы этого раздела легко распространяются на множество целых чисел $\mathbb{Z}$ за счёт учёта знака числа.

В соответствии с традициями, в теории чисел знак операции умножения $*$ опускается, а запись $x + (-y)$ упрощается до $x - y$.

2.4.1. Делимость чисел

Число $a \in \mathbb{N}$ *делится на* число $b \in \mathbb{N}$ (обозначение $b|a$), если $\exists\, q \in \mathbb{N}\ \ (a = bq)$. Число a называется *кратным* числу b, число b называется *делителем* числа a, а число q называется *частным* от деления a на b.

Напомним обозначение $\mathbb{N}_0 \stackrel{\text{Def}}{=} \mathbb{N} + 0$.

ТЕОРЕМА. $\forall\, a, b \in \mathbb{N}\ (\exists\, q, r \in \mathbb{N}_0\ \ (a = bq + r\ \&\ 0 \leqslant r < b))$.

Доказательство. Следующий элементарный *алгоритм деления вычитанием* находит числа q и r.

$q := 0;\ r := 0$

while $a \geqslant b$ **do** $a := a - b;\ q := q + 1$ **end while**

$r := a$ □

СЛЕДСТВИЕ. *Если $a = bq + r\ \&\ 0 \leqslant r < b$, то числа q и r единственны.*

Доказательство. Пусть $a = bq_1 + r_1, 0 \leqslant r_1 < b, a = bq_2 + r_2, 0 \leqslant r_2 < b$. Тогда $0 = (bq_1 + r_1) - (bq_2 + r_2) = b(q_1 - q_2) + (r_1 - r_2)$. Если $q_1 \neq q_2$, то $|b(q_1 - q_2)| \geqslant b$, но $|r_1 - r_2| < b$, и равенство невозможно. Следовательно, $q_1 = q_2$, откуда $r_1 - r_2 = 0$ и $r_1 = r_2$. □

Число q называется *неполным частным*, а число r называется *остатком* от деления числа a на число b. Если $b|a$, то $r = 0$.

ЗАМЕЧАНИЕ

Для вычисления неполного частного и остатка (деление с остатком) обычно определяются специальные операции. В этом учебнике они обозначаются $a\ \mathbf{div}\ b$ и $a\ \mathbf{mod}\ b$, соответственно.

Делить с остатком можно отрицательные числа. Действительно, пусть $a, b, q, r \in \mathbb{N}_0$, $a = bq + r$, $0 \leqslant r < b$. Тогда если $b|a$, то $r = 0$ и $-a = b(-q)$, а в противном случае $-a = -bq - r = b(-(q + 1)) + (b - r)$, причём остаток $0 \leqslant b - r < b$.

Множество делителей числа $a \in \mathbb{N}$ обозначается $D(a) \stackrel{\text{Def}}{=} \{b \in \mathbb{N} \mid b|a\}$. Очевидно, что любое число делится на 1 и на себя. Эти делители называются *тривиальными*. Прочие делители, если они есть, называются *нетривиальными*, или *собственными*.

ОТСТУПЛЕНИЕ

Определить делители произвольного числа не так просто. Для некоторых делителей известны *признаки делимости*, то есть сравнительно простые алгоритмы, позволяющие ответить на вопрос, является ли число b делителем числа a. Читателю, несомненно, известны со школы

признаки делимости на 2, 3, 5, 10. Все признаки делимости следуют общей схеме: строится убывающая последовательность $a_0, \ldots, a_n$, такая что $a_0 = a$ и $b|a_i \iff b|a_{i+1}$. Однако все признаки делимости индивидуальны для каждого делителя, существенно зависят от системы счисления и построены далеко не для всех делителей. В реальных компьютерных вычислениях признаки делимости не используются.

В общем случае деление в компьютере является сравнительно трудоёмкой операцией. В большинстве арифметических устройств компьютеров деление выполняется примерно в два раза дольше умножения, а умножение выполняется примерно в два раза дольше сложения.

Легко видеть, что *отношение делимости* рефлексивно $\forall\, a \in \mathbb{N}\ \ (a|a)$, транзитивно $\forall\, a, b, c \in \mathbb{N}\ \ (a|b\ \&\ b|c \Longrightarrow a|c)$ и антисимметрично $\forall\, a, b \in \mathbb{N}\ \ (a|b\ \&\ b|a \Longrightarrow a = b)$. Таким образом, делимость чисел — это нестрогий частичный порядок на множестве $\mathbb{N}$.

ОТСТУПЛЕНИЕ

С отношением делимости связано множество любопытных и поучительных числовых казусов и фокусов. Например, еще в древности были обнаружены так называемые *совершенные* числа, то есть числа, равные сумме своих делителей, считая 1, но не считая само число. Первые совершенные числа: $6 = 1{+}2{+}3$, $28 = 1{+}2{+}4{+}7{+}14$, $496 = 1{+}2{+}4{+}8{+}16{+}31{+}62{+}124{+}248$, ... Все обнаруженные совершенные числа являются чётными и треугольными. До сих пор неизвестно, конечно ли множество совершенных чисел и существуют ли нечётные совершенные числа.

Пример. На рис. 2.4 представлена диаграмма Хассе отношения делимости для чисел от 1 до 12. Хорошо видно, что делимость имеет «нерегулярный» характер.

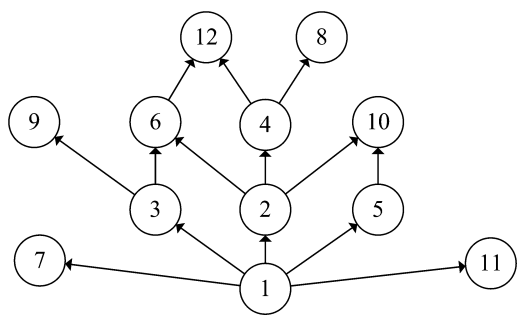


Рис. 2.4. Диаграмма Хассе отношения делимости

2.4.2. Наибольший общий делитель

Натуральное число c такое, что $c|a$ и $c|b$, называется *общим делителем* чисел a и b. Множество общих делителей чисел a и b обозначим $D(a, b)$. Наибольшее из таких чисел называется *наибольшим общим делителем* чисел a и b. Обозначение: $\gcd(a, b)$, НОД(a, b) или просто (a, b).

ЗАМЕЧАНИЕ

В литературе по теории чисел обычно используется последнее обозначение, однако оно совпадает с используемым обозначением упорядоченной пары. Поэтому для обозначения наибольшего общего делителя здесь используется функция $\gcd(a, b)$.

Если $\gcd(a, b) = 1$, то числа a и b называются *взаимно простыми*. Обозначение: $a \perp b$.

Пример. Из определения вытекает, что отношение взаимной простоты антирефлексивно (для всех чисел, кроме единицы) и симметрично. На рис. 2.5 представлена часть графика отношения взаимной простоты.

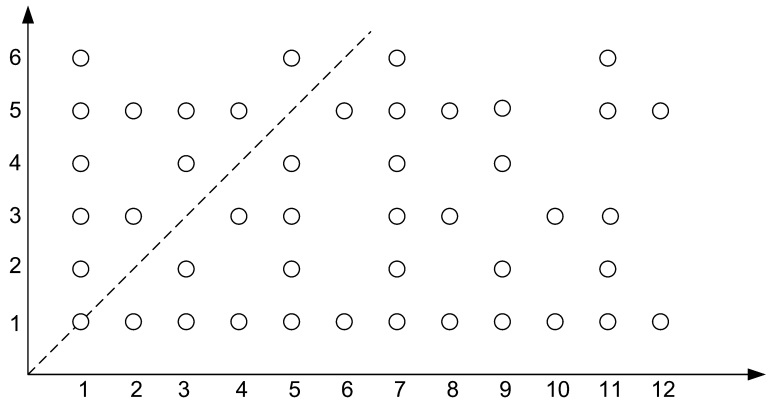


Рис. 2.5. Часть графика отношения взаимной простоты

ЛЕММА 1. *Если $b|a$, то множество общих делителей чисел a и b совпадает с множеством делителей b: $b|a \Longrightarrow D(a, b) = D(b)$.*

Доказательство. Любой общий делитель чисел a и b очевидно является делителем b. Любой делитель b делит также и a в силу транзитивности. □

СЛЕДСТВИЕ. $b|a \Longrightarrow \gcd(a, b) = b$.

ЛЕММА 2. *Если $a = bq + r$, то множество общих делителей чисел a и b совпадает с множеством общих делителей b и r: $a = bq + r \Longrightarrow D(a, b) = D(b, r)$.*

Доказательство. Пусть $x \in D(a, b)$. Тогда $\exists k, l \ \ ((a = kx, b = lx))$ и $kx = lxq + r$, $r = (k - lq)x$, следовательно, $x|r$, $x \in D(b, r)$. Пусть $y \in D(b, r)$. Тогда $\exists m, n \ \ ((b = my, r = ny))$ и $a = myq + ny = (mq + n)y$, следовательно, $y|a$, $y \in D(b, r)$. Таким образом, $D(a, b) = D(b, r)$. □

СЛЕДСТВИЕ. $a = bq + r \Longrightarrow \gcd(a, b) = \gcd(b, r)$.

Пример. $60 = 7 \cdot 8 + 4, \quad \gcd(60, 8) = \gcd(8, 4) = 4, \quad \gcd(60, 7) = \gcd(7, 4) = 1.$

Доказанные утверждения лежат в основе алгоритма нахождения наибольшего общего делителя двух натуральных чисел a и b, известного как *алгоритм Евклида*. В силу симметричности $\gcd(a, b)$, не ограничивая общности, можно считать, что $a > b$.

Обоснование. На каждом шаге алгоритма пара (a, b) замещается парой (b, r), где r — остаток от деления a на b. Согласно следствию, после каждой итерации цикла

Алгоритм 2.2. Алгоритм Евклида

Вход: натуральные числа a, b.
Выход: значение $\gcd(a, b)$.

```
if a < b then
  a :=: b  //поменяли числа местами
end if
while b > 0 do
  r := a mod b;  a := b;  b := r
end while
return  a
```

искомый наибольший общий делитель остаётся неизменным, то есть является инвариантом алгоритма. Когда a станет кратно b, r будет равно нулю. Это означает, что искомый наибольший общий делитель найден. Цикл обязательно завершится, потому что числа a и b не могут уменьшаться бесконечно. □

Пример. $\gcd(616, 286) = \gcd(286, 44) = \gcd(44, 22) = 22$.

ОТСТУПЛЕНИЕ

Алгоритм Евклида даёт повод сделать некоторые замечания относительно *стиля программирования*, то есть набора необязательных правил и соглашений, направленных на *улучшение* кода. В эту категорию входят использование *дисциплины имён*, *отступов*, *комментариев* и множества других полезных приёмов. Важно понимать, что использование того или иного стиля программирования улучшает код не абсолютно, а относительно тех целей, которые преследуют выбранные правила. Рассмотрим в качестве примера ещё одну реализацию алгоритма Евклида.

Алгоритм 2.3. «Улучшенный» алгоритм Евклида

Вход: a, b — натуральные числа.
Выход: Значение $\gcd(a, b)$ есть сумма $a + b$.

```
while a ≠ 0 & b ≠ 0 do
  if a > b then a := a mod b else b := b mod a end if
end while
```

С одной стороны, этот вариант несколько эффективнее по времени, поскольку на каждом шаге цикла выполняется одно присваивание вместо трёх, и по памяти, поскольку используются две переменные вместо трёх. С другой стороны, этот код значительно дальше от доказанных математических формул и потому требует существенного дополнительного обоснования, а самое главное, в этом варианте нарушается «правило хорошего тона», гласящее, что каждый объект (в частности, переменная) в программе при изменении состояния (значения) не должен менять свой «смысл» или свою «ответственность».

2.4.3. Наименьшее общее кратное

Натуральное число d такое, что $a|d$ и $b|d$, называется *общим кратным* чисел a и b. Множество общих кратных чисел a и b будем обозначать $K(a, b)$. Наименьшее из таких чисел называется *наименьшим общим кратным* a и b. Обозначение: $\mathrm{lcm}(a, b)$, НОК(a, b) или $[a, b]$.

ЗАМЕЧАНИЕ

В отличие от конечного множества $D(a,b)$, множество $K(a,b)$ является бесконечным. Однако $K(a,b) \subset \mathbb{N}$, поэтому $K(a,b)$ имеет наименьший элемент и определение наименьшего общего кратного корректно.

ТЕОРЕМА. *Для любых натуральных чисел a, b и m справедливо:*

1) $a|m \ \& \ b|m \Longrightarrow \operatorname{lcm}(a,b)|m$;
2) $\gcd(a,b)\operatorname{lcm}(a,b) = ab$.

Доказательство. Пусть $d := \gcd(a,b)$, тогда $a = a_1 d, b = b_1 d$, и $a_1 \perp b_1$. Пусть число k таково, что $m = ak$. Тогда $\frac{ak}{b} = \frac{a_1 k}{b_1} \in \mathbb{N}$, следовательно, $b_1|k$. Рассмотрим число t, такое что $k = b_1 t$. Имеем $m = ak = ab_1 t = \frac{ab}{d}t$. С другой стороны, очевидно, что любое число вида $\frac{ab}{d}t$ является общим кратным чисел a и b. Наименьшее общее кратное достигается при $t = 1$, то есть равно $\frac{ab}{d}$. Таким образом, оба утверждения теоремы доказаны. □

2.4.4. Простые числа

Натуральное число $a > 1$ называется *простым*, если оно не имеет собственных делителей. В противном случае a называется *составным*.

ЗАМЕЧАНИЕ

Число 1 имеет только один делитель, поэтому его не относят ни к простым, ни к составным.

Множество простых чисел обозначается $\mathbb{P}$.

ТЕОРЕМА 1. *Для любого натурального числа $a > 1$ его наименьший делитель, отличный от 1, является простым числом.*

Доказательство. От противного. Пусть этот делитель составной, тогда у него тоже есть делитель, меньший его и отличный от 1. Но тогда он делит и a, а значит, исходный делитель не был наименьшим. □

СЛЕДСТВИЕ. *Для составного числа $a > 1$ его наименьший делитель, отличный от 1, не превосходит $\sqrt{a}$.*

Доказательство. Пусть $p > 1$ — наименьший делитель a. Тогда $a = pq$, где $q \geqslant p$. Следовательно, $a \geqslant p^2$. □

ТЕОРЕМА 2. *Простых чисел бесконечно много.*

Доказательство. От противного. Пусть $|\mathbb{P}| = n$. Тогда $\mathbb{P} = \{p_1, \ldots, p_n\}$. Составим число $p := p_1 p_2 \ldots p_n + 1$. Это число не делится ни на одно из чисел $p_1, \ldots, p_n$, так как даёт остаток 1 при делении на каждое из них, и не совпадает ни с одним из них. Следовательно, число p само простое. Противоречие. □

Алгоритм 2.4. Решето Эратосфена

Вход: натуральное число $n > 1$.
Выход: множество простых чисел, не превосходящих n.

```
B := {2, 3, 4, ..., n}  // все числа 2..n
while B ≠ ∅ do
  x := min(B)  // наименьший элемент B
  yield x  // выдать x
  B := B − x  // удалить x из B
  y := x^2
  while y ⩽ n do
    B := B − y  // удалить из B все числа, кратные x
    y := y + x
  end while
end while
```

Для построения множества простых чисел, не превосходящих данного числа n, используют простой способ, называемый *решетом Эратосфена*.

Обоснование. На каждом шаге алгоритма минимальный элемент $x \in B$ — простое число. В противном случае у него был бы простой делитель $p < x$, и x был бы удалён из B ранее, как кратный p. Для простого x достаточно удалить из B все кратные ему, начиная с x^2, так как все меньшие составные, кратные x, были удалены из B ранее. □

Лемма 1. $\forall\, a, b, c \in \mathbb{N}\ (b \perp c \Longrightarrow \gcd(ac, b) = \gcd(a, b))$.

Доказательство. Рассмотрим $\gcd(ac, b)$. Так как $b \perp c$, то $\gcd(ac, b)$ не имеет общих делителей с c, отсюда $\gcd(ac, b) = \gcd(a, b)$. □

Следствие. $\forall\, a, b, c \in \mathbb{N}\ (b \perp c\ \&\ b|ac \Longrightarrow b|a)$.

Пример. Найдём с помощью алгоритма 2.4 все простые числа, не превосходящие 100. В правой нижней ячейке искомый список простых чисел.

i	Вычеркнутые составные числа, кратные i	Оставшиеся числа
2	4,6,8,10,12,14,16,18,20,22,24,26,28, 30,32,34,36,38,40,42,44,46,48,50,52, 54,56,58,60,62,64,66,68,70,72,74,76, 78,80,82,84,86,88,90,92,94,96,98,100	2,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33, 35,37,39,41,43,45,47,49,51,53,55,57,59,61,63, 65,67,69,71,73,75,77,79,81,83,85,87,89,91, 93,95,97,99
3	9,15,21,27,33,39,45,51,57,63, 69,75,81,87,93,99	2,3,5,7,11,13,17,19,23,25,29,31,35,37,41,43, 47,49,53,55,59,61,65,67,71,73,77,79,83,85, 89,91,95,97
5	25,35,55,65,85,95	2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,49,53, 59,61,67,71,73,77,79,83,89,91,97
7	49,77,91	2,3,5,7,11,13,17,19,23,29,31,37,41,43, 47,53,59,61,67,71,73,79,83,89,97

ЛЕММА 2 (лемма Евклида). $\forall a, b \in \mathbb{N}, p \in \mathbb{P}\ (p|ab \Longrightarrow p|a \vee p|b)$.

Доказательство. От противного. Пусть ни a, ни b не делятся на p. Тогда из простоты p следует, что $a \perp p$, $b \perp p$. По лемме $\gcd(ab, p) = \gcd(a, p) = 1$, $ab \perp p$ — противоречие. □

СЛЕДСТВИЕ. $\forall a_1, a_2, \ldots, a_n \in \mathbb{N}, p \in \mathbb{P}\ (p|a_1 a_2 \ldots a_n \Longrightarrow \exists i\ (p|a_i))$.

ТЕОРЕМА 3 (основная теорема арифметики). *Любое натуральное число, большее единицы, можно представить в виде произведения простых множителей, причём единственным образом.*

Доказательство.

[Существование] От противного. Пусть $1 < n \in \mathbb{N}$ — наименьшее натуральное число, не представимое в виде произведения нескольких простых чисел. Тогда если число n — простое, то оно является своим разложением. Если же число n — составное, то $n = n_1 n_2$, где $n_1 < n$, $n_2 < n$, причём каждое из n_1, n_2 имеет разложение, а значит, можно разложить и исходное число n. Противоречие.

[Единственность] Пусть n имеет два различных разложения на простые множители. Рассмотрим любое число p в одном из разложений. Тогда произведение чисел второго разложения делится на p и по следствию из леммы Евклида одно из простых чисел второго разложения совпадает с p. Сократим оба разложения на p и применим к оставшимся разложениям такое же рассуждение. Так по одному сократим все числа в первом разложении, найдя и сократив равные им во втором. Процесс закончится ввиду конечности разложений. □

Имея разложения чисел на множители, легко найти разложения наибольшего общего делителя и наименьшего общего кратного этих чисел.

СЛЕДСТВИЕ. *Пусть $p_1, p_2, \ldots, p_n \in \mathbb{P}$ — некоторый набор простых чисел. Тогда если* $a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$, $b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n}$, $\gcd(a, b) = p_1^{c_1} p_2^{c_2} \cdots p_n^{c_n}$, $\mathrm{lcm}(a, b) = p_1^{d_1} p_2^{d_2} \cdots p_n^{d_n}$, *то* $c_i = \min(a_i, b_i)$, $d_i = \max(a_i, b_i)$.

Пример. $24 = 2^3 3^1, 36 = 2^2 3^2, \gcd(24, 36) = 2^2 3^1 = 12, \mathrm{lcm}(24, 36) = 2^3 3^2 = 72$.

ОТСТУПЛЕНИЕ

Разложение натурального числа на простые множители является одной из древнейших математических задач, которая остаётся актуальной и сегодня. Для сходной задачи проверки простоты числа уже давно найдены достаточно эффективные алгоритмы (см. решето Эратосфена), а для разложения числа на множители пока не найдено достаточно эффективных с практической точки зрения алгоритмов, но при этом не доказано, что таких алгоритмов не существует. Между тем, очень многие современные информационные технологии, например, шифрование с открытым ключом, критически зависят от наличия или отсутствия этих алгоритмов.

2.4.5. Сравнения

Пусть числа $a, b, m \in \mathbb{N}$. Говорят, что число a *сравнимо* с числом b *по модулю* m, если оба числа имеют одинаковый остаток при делении на m, то есть $a = pm + r$, $b = qm + r$. Обозначение: $a \equiv b \pmod m$.

ТЕОРЕМА 1. *Пусть $a, b \in \mathbb{N}, a > b$. Число a сравнимо с числом b по модулю m тогда и только тогда, когда m является делителем $a - b$:*

$$a \equiv b \pmod m \iff m|(a-b).$$

Доказательство.

[$a \equiv b \pmod m \Longrightarrow m|(a-b)$] Пусть $a = pm + r$, $b = qm + r$.
Тогда $a - b = (p-q)m + (r-r) = km$ и $m|(a-b)$.

[$m|(a-b) \Longrightarrow a \equiv b \pmod m$] Пусть $a - b = km$.
Тогда $\exists\, p, q, r_a, r_b \in \mathbb{N}_0$ $(a = pm + r_a \;\&\; b = qm + r_b \;\&\; r_a < a \;\&\; r_b < b)$.
Имеем $a - b = (p-q)m + (r_a - r_b)$, откуда $r_a - r_b = 0$ и $r_a = r_b = r$. □

ЛЕММА 1. *Отношение $a \equiv b \pmod m$ является отношением эквивалентности на множестве $\mathbb{N}$.*

Доказательство.

[Рефлексивность] Остаток определен однозначно.

[Симметричность] Определение сравнимости не зависит от порядка.

[Транзитивность] В силу транзитивности равенства остатков. □

Рассмотрим основные свойства введенного отношения сравнимости.

ЛЕММА 2. *Пусть $a \equiv b \pmod m$ и $c \equiv d \pmod m$.*
Тогда $(a + c) \equiv (b + d) \pmod m$.

Доказательство. Пусть $a = q_a m + r_1$, $b = q_b m + r_1$, $c = q_c m + r_2$, $d = q_d m + r_2$. Тогда $a + c = q_a m + r_1 + q_c m + r_2 = q_{ac} m + r$ и $b + d = q_b m + r_1 + q_d m + r_2 = q_{bd} m + r$. Если $r = r_1 + r_2 \geqslant m$, то положим $r := r - m$, $q_{ac} := q_{ac} + 1$, $q_{bd} := q_{bd} + 1$. □

ЛЕММА 3. *Пусть $a \equiv b \pmod m$ и $c \equiv d \pmod m$. Тогда $ac \equiv bd \pmod m$.*

Доказательство. Не ограничивая общности, можно считать, что $a > b, c > d$. Так как $(a-b)|m$ и $(c-d)|m$, то $a - b = km$, $c - d = lm$ для некоторых целых k, l. Имеем $ac = (km+b)(lm+d) = (klm+bl+dk)m+bd$. Из этого равенства следует, что остатки ac и bd при делении на m равны, а значит, $ac \equiv bd \pmod m$. □

ЛЕММА 4. *Если $a \equiv b \pmod m$, $n \in \mathbb{N}$, то $a^n \equiv b^n \pmod m$.*

Доказательство. Индукция по n. База при $n = 1$ верна по условию леммы. Индукционный переход по лемме 3. □

Из доказанных лемм непосредственно следует теорема 2.

ТЕОРЕМА 2. *Пусть $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$ — многочлен с натуральными коэффициентами. Тогда если $a \equiv b \pmod m$, то $f(a) \equiv f(b) \pmod m$.*

2.4.6. Системы вычетов

Поскольку деление с остатком определено для всех целых чисел, отношение сравнимости по модулю также распространяется на всё множество целых чисел.

Классом вычетов по модулю m (обозначение $[a]_m$) называется класс эквивалентности (п. 1.7.1.) по отношению сравнимости:

$$[a]_m \stackrel{\text{Def}}{=} \{b \in \mathbb{Z} \mid b \equiv a \pmod m\}, \quad a \in \mathbb{Z}, m \in \mathbb{N}.$$

ЗАМЕЧАНИЕ

Слово «вычет» употребляется как синоним слова «остаток».

Всего классов вычетов по модулю m ровно m и один класс вычетов составляют все числа, дающие один и тот же остаток при делении на m.

Определим над классами вычетов операции сложения $+_m$ и умножения $*_m$:

$$[x]_m + [y]_m \stackrel{\text{Def}}{=} [(x+y)]_m, \ [x]_m * [y]_m \stackrel{\text{Def}}{=} [(xy)]_m.$$

Множество классов вычетов с введенными операциями обозначается $\mathbb{Z}_m$.

ЛЕММА 1. *Множество классов вычетов $\mathbb{Z}_m$ является коммутативным кольцом с единицей.*

Доказательство. В качестве нуля нужно взять $[0]_m$, а в качестве единицы — $[1]_m$. Свойства из определения кольца (п. 2.3.1.) выполнены в силу доказанных арифметических свойств сравнений. □

Полной системой вычетов по модулю m называют множество целых чисел, содержащее ровно по одному элементу из каждого класса вычетов по модулю m.

Пример. $\{7, 10, -1, 3, 101\}$ — полная система вычетов по модулю 5.

Множество $\{0, 1, 2, \ldots, m-1\}$ называется *системой наименьших неотрицательных вычетов*. По принципу Дирихле эта система полна.

ТЕОРЕМА 1 (о полных системах вычетов). *Пусть $\{a_1, \ldots, a_m\}$ — полная система вычетов по модулю m, x, y — целые числа, $x \perp m$. Тогда $\{y + xa_1, \ldots, y + xa_m\}$ — полная система вычетов по модулю m.*

Доказательство. Покажем, что числа $y + xa_1, \ldots, y + xa_m$ дают различные остатки при делении на число m. Тогда, поскольку чисел ровно m, они образуют полную систему вычетов. Предположим противное, пусть $y + xa_i$ и $y + xa_j$ дают одинаковые остатки при делении на m. Тогда $xa_i + y \equiv xa_j + y \pmod m$ или $x(a_i - a_j) | m$. Так как $x \perp m$, $(a_i - a_j) | m$ или $a_i \equiv a_j \pmod m$, что противоречит тому, что исходная система вычетов $\{a_1, \ldots, a_m\}$ является полной системой вычетов по модулю m. □

СЛЕДСТВИЕ. *Пусть $p \in \mathbb{P}$. Тогда $\mathbb{Z}_p$ является полем.*

Доказательство. Поскольку по лемме 1 рассматриваемое множество является кольцом (п. 2.4.5), достаточно установить наличие у каждого вычета $[x]_m$ обратного по введенной операции умножения. Для этого рассмотрим любую полную систему вычетов $\{a_1, \ldots, a_m\}$, умноженную на x, $\{xa_1, \ldots, xa_m\}$. По теореме это будет полная система вычетов, то есть $\exists\, z \in \{xa_1, \ldots, xa_m\}$ $(xz \equiv 1 \pmod m)$. Класс $[z]_m$ и будет искомым обратным элементом для $[x]_m$. □

ЛЕММА 2. *Если* $a \equiv b \pmod m$, *то* $\gcd(a, m) = \gcd(b, m)$.

Доказательство. $a \equiv b \pmod m \Longrightarrow b = a + mk,\ k \in \mathbb{Z} \Longrightarrow \gcd(a, m) = \gcd(b, m)$. □

Приведённой системой вычетов по модулю m называют множество целых чисел, содержащее ровно по одному элементу из каждого класса вычетов по модулю m, взаимно простого с m.

Пример. $\{7, -1, 3, 101\}$ — приведённая система вычетов по модулю 5.

ТЕОРЕМА 2. *Если* $\{a_1, \ldots, a_n\}$ — *приведённая система вычетов по модулю* m, $x \in \mathbb{Z}$, $x \perp m$, *то* $\{xa_1, \ldots, xa_n\}$ — *приведённая система вычетов по модулю* m.

Доказательство. Аналогично случаю полной системы вычетов. □

2.4.7. Китайская теорема об остатках

ЗАМЕЧАНИЕ

Китайская теорема об остатках была сформулирована в Китае, предположительно, в III веке н. э., китайским математиком Сунь Цзы, откуда и получила свое название.

ТЕОРЕМА. *Для любых попарно взаимно простых натуральных чисел* $a_1, \ldots, a_n$ *и для любых целых чисел* $r_1, \ldots, r_n$ *таких, что* $\forall\, i \in 1..n$ $(0 \leqslant r_i < a_i)$, *существует такое число* s, *что деление его на каждое из чисел* a_i *дает в остатке число* r_i.

Доказательство. Индукция по n. База: при $n = 1$ возьмём $s := r_1$. Индукционный переход. Пусть $\exists\, t \in \mathbb{N}$ $(\forall\, i \in 1..(n-1)$ $(t = q_i a_i + r_i))$. Рассмотрим систему наименьших неотрицательных вычетов $\{0, \ldots, a_n - 1\}$ по модулю a_n и применим теорему о полных системах вычетов, полагая $x := a_1 a_2 \ldots a_{n-1}$ и $y := t$. Тогда система чисел $s_1 = t$, $s_2 = t + x$, $\ldots$, $s_{a_n} = t + (a_n - 1)x$ является полной системой вычетов по модулю a_n, причём $\forall\, i \in 1..(n-1)$ $(\forall\, j \in 1..a_n$ $(s_j \equiv t \pmod{a_i}))$ по построению. Ввиду полноты системы среди чисел $s_1, \ldots, s_{a_n}$ существует такое число s, которое при делении на a_n даст в остатке r_n. □

СЛЕДСТВИЕ. *Пусть* $m := m_1 m_2 \ldots m_n$, $\forall\, i, j \in 1..n, i \neq j$ $(m_i \perp m_j)$, $x, a \in \mathbb{Z}$. *Тогда* $x \equiv a \pmod m \iff \forall\, i \in 1..n$ $(x \equiv a \pmod{m_i})$.

2.4.8. Вычисления в остаточных классах

ЗАМЕЧАНИЕ

Пусть $m = m_1 m_2 \dots m_k$, причём $m_1, m_2 \dots m_k$ — попарно взаимно просты. Тогда китайская теорема об остатках позволяет ввести взаимно однозначное соответствие между остатками при делении на m и наборами остатков от деления на $m_1, m_2 \dots m_k$.

Пусть $m := m_1 m_2 \dots m_n$, $\forall i, j \in 1..n, i \neq j \ (m_i \perp m_j)$. По следствию к китайской теореме об остатках любое число $a \in 0..(m-1)$ однозначно определяется остатками $a_1, \dots, a_n$ от деления числа a на числа $m_1, \dots, m_n$. Такая непозиционная система счисления называется *системой остаточных классов*, или *модулярной арифметикой*. Набор чисел $m_1, \dots, m_k$ называется *основанием* системы остаточных классов. Обозначение: $a = (a_1, \dots, a_n)$. В системе остаточных классов операции сложения и умножения выполняются покомпонентно, то есть если $a = (a_1, \dots, a_n)$, $b = (b_1, \dots, b_n)$, $a + b = c$, $ab = d$, то $c = (c_1, \dots, c_n)$, $d = (d_1, \dots, d_n)$, где $c_i := (a_i + b_i) \bmod m$ и $d_i := (a_i b_i) \bmod m$, соответственно.

Пример. Пусть система остаточных классов задана своим основанием $(3, 4, 5)$. Вычислим в этой системе сумму и произведение чисел 7 и 8. Имеем:
$7 = (7 \bmod 3, 7 \bmod 4, 7 \bmod 5) = (1, 3, 2)$, $8 = (2, 0, 3)$. Тогда сложение:
$7 + 8 = ((1 + 2) \bmod 3, (3 + 0) \bmod 4, (2 + 3) \bmod 5) = (0, 3, 0)$ и
$15 = (15 \bmod 3, 15 \bmod 4, 15 \bmod 5) = (0, 3, 0)$. Умножение:
$7 \cdot 8 = ((1 \cdot 2) \bmod 3, (3 \cdot 0) \bmod 4, (2 \cdot 3) \bmod 5) = (2, 0, 1)$ и
$56 = (56 \bmod 3, 56 \bmod 4, 56 \bmod 5) = (2, 0, 1)$.

ОТСТУПЛЕНИЕ

Вычисления в остаточных классах в некоторых специальных применениях используются наряду или даже вместо вычислений в обычной позиционной двоичной системе счисления.

Одним из главных преимуществ вычислений в системах остаточных классов является отсутствие переноса разрядов при сложении и умножении, поскольку достаточно сложить или умножить представляющие векторы поэлементно. В специализированном процессоре это можно сделать параллельно, а поэтому *время выполнения обеих операций не зависит от количества чисел в основании системы остаточных классов*. Сами числа в основании системы остаточных классов можно выбирать небольшими, и аппаратные операции с ними можно выполнять очень быстро.

Ограничение диапазона представления чисел не является существенным. Например, если взять основания $3, 5, 7, 11, 13$, то для хранения остатков потребуется $2+3+3+4+4 = 16$ бит. При этом диапазон представления составит 0..15014, что вполне сопоставимо с диапазоном 0..65535 для 16-битных двоичных чисел.

Действительно существенным недостатком вычислений в остаточных классах являются медленные алгоритмы перевода из позиционной системы счисления в систему остаточных классов и обратно, а также деления с остатком. Эффективные алгоритмы для этих операций неизвестны.

2.4.9. Функция Эйлера

Функцией Эйлера от натурального аргумента n называется количество чисел меньших n и взаимно простых с n. Стандартным обозначением для функции Эйлера является $\varphi(n)$. Из определения ясно, что $n > 1 \Longrightarrow 1 \leqslant \varphi(n) < n$. По определению $\varphi(1) := 1$.

Пример. $\varphi(1)=1, \varphi(2)=1, \varphi(3)=2, \varphi(4)=2, \varphi(5)=4, \varphi(6)=2.$

ЗАМЕЧАНИЕ

Приведённая система вычетов по модулю n содержит $\varphi(n)$ элементов.

Вообще говоря, числовая функция f называется *аддитивной*, если она уважает сложение, то есть $f(x+y)=f(x)+f(y)$, и называется *мультипликативной*, если она уважает умножение, то есть $f(xy)=f(x)f(y)$.

В теории чисел используются другие определения этих понятий. Функция $f\colon \mathbb{N}\to\mathbb{N}$ называется *аддитивной*, если $\forall\, x,y\in\mathbb{N}\ \ (x\perp y \Longrightarrow f(xy)=f(x)+f(y))$, и называется *мультипликативной*, если $\forall\, x,y\in\mathbb{N}\ \ (x\perp y \Longrightarrow f(xy)=f(x)f(y))$.

Примеры

Количество множителей в разложении числа n на простые без учета кратности $\Omega(n)$ — аддитивная функция. Число делителей числа n (обозначается $d(n)$) и сумма делителей числа n (обозначается $\sigma(n)$) — мультипликативные функции.

Также в теории чисел выделяют *вполне аддитивные* и *вполне мультипликативные* функции, пренебрегая взаимной простотой в соответствующих определениях.

Пример. Степенная функция $f(n)=a^n, a>0$ вполне мультипликативна.

Зададимся мультипликативной функцией f. Заметим, что для подсчета её значения для произвольного n достаточно посчитать её значения для чисел вида p^a, где $p\in\mathbb{P}$, $a\in\mathbb{N}_0$. Действительно, пусть n — произвольное натуральное число. Тогда по основной теореме арифметики (п. 2.4.4) число n можно единственным образом разложить на простые множители $n=p_1^{a_1}p_2^{a_2}\dots p_k^{a_k}$, и из мультипликативности получим $f(n)=f(p_1^{a_1}p_2^{a_2}\dots p_k^{a_k})=f(p_1^{a_1})f(p_2^{a_2})\dots f(p_k^{a_k})$.

ТЕОРЕМА 1. *Функция Эйлера мультипликативна.*

Доказательство. Пусть $a\perp b$. Рассмотрим множество

$$U:=\{ax+by \mid x\in\{1,2,\dots,b\}, y\in\{1,2,\dots,a\}\},$$

состоящее из ab чисел. Заметим, что все они дают разные остатки при делении на ab. Действительно, пусть $ax_1+by_1\equiv ax_2+by_2 \pmod{ab}$. Тогда $ax_1\equiv ax_2 \pmod b$ и $by_1\equiv by_2 \pmod a$, откуда $x_1=x_2$ и $y_1=y_2$. Далее, $ax+by\perp ab \Longleftrightarrow x\perp b \ \&\ y\perp a$. Действительно, $ax+by\perp ab \Longleftrightarrow ax+by\perp b \ \&\ ax+by\perp a \Longleftrightarrow ax\perp b \ \&\ by\perp a \Longleftrightarrow$ $\Longleftrightarrow x\perp b \ \&\ y\perp a$. Таким образом, множество элементов U, взаимно простых с ab, изоморфно множеству пар (x,y) таких, что $x\in 1..b \ \ \&\ \ x\perp b$, $y\in 1..a \ \ \&\ \ y\perp a$.
Но $|\{ax+by=u\in U \mid ax+by\perp ab\}|=\varphi(ab)$, $|\{x\in 1..b \mid x\perp a\}|=\varphi(a)$, $|\{y\in 1..a \mid y\perp b\}|=\varphi(b)$. Поэтому $\varphi(ab)=\varphi(a)\varphi(b)$. □

ТЕОРЕМА 2. *Если* $n=p_1^{a_1}p_2^{a_2}\dots p_k^{a_k}$, $p_i\in\mathbb{P}$, *то*

$$\varphi(n)=\left(p_1^{a_1}-p_1^{a_1-1}\right)\left(p_2^{a_2}-p_2^{a_2-1}\right)\dots\left(p_k^{a_k}-p_k^{a_k-1}\right)=n\prod_{i=1}^{k}\left(1-\frac{1}{p_i}\right).$$

Доказательство. Заметим, что $\varphi(p^a) = p^a - p^{a-1}$ для $\alpha > 0$. Действительно, числами, не взаимно простыми со степенью простого числа, являются те и только те, которые на это простое число делятся, а их ровно p^{a-1}. Отсюда и из мультипликативности функции Эйлера следует утверждение теоремы. □

Самое известное и важное свойство функции Эйлера выражается в теореме Эйлера.

ТЕОРЕМА 3 (Эйлера). *Если $a \perp n$, то $a^{\varphi(n)} \equiv 1 \pmod{n}$.*

Доказательство. Рассмотрим приведённую систему вычетов $\{x_1, \ldots, x_{\varphi(n)}\}$ по модулю n. В ней ровно $\varphi(n)$ элементов. Домножим каждый элемент этой системы на число a. По теореме п. 2.4.6 полученная система тоже является приведённой системой вычетов по модулю n. Таким образом, произведение всех полученных вычетов сравнимо с произведением всех исходных (поскольку новая система вычетов также приведённая, то она является некоторой перестановкой исходной системы). Отсюда $ax_1 ax_2 \ldots ax_{\varphi(n)} \equiv x_1 x_2 \ldots x_{\varphi(n)} \pmod{n}$. Поскольку $x_1 x_2 \ldots x_{\varphi(n)}$ взаимно просто с n, то на это произведение можно поделить обе части сравнения. Получим $a^{\varphi(n)} \equiv 1 \pmod{n}$. □

СЛЕДСТВИЕ (малая теорема Ферма). *Пусть p — простое число. Тогда $a^{p-1} \equiv 1 \pmod{p}$ для любого a, не делящегося на p.*

Пример. Покажем, как теорему Эйлера можно применять в задаче, на первый взгляд ничего общего с ней не имеющей. Пусть числа m и n взаимно просты, n не делится ни на 2, ни на 5. Оценить длину периода дроби $x = \frac{m}{n}$. Решение. Существует t такое, что $(10^t - 1)x = y$, где y — целое число. Отсюда $x = \frac{y}{10^t - 1}$. Поскольку $\frac{m}{n}$ — несократимая дробь, имеем $(10^t - 1) | n$. Заметим, что $t = \varphi(n)$ подходит по теореме Эйлера, а длина периода — минимальное такое t, значит, длина периода не превосходит $\varphi(n)$.

2.5. Векторные пространства и модули

Понятие векторного пространства должно быть известно читателю из курса средней школы и других математических курсов. Обычно это понятие ассоциируется с геометрической интерпретацией векторов в пространствах $\mathbb{R}^2$ и $\mathbb{R}^3$. В этом разделе даны и другие примеры векторных пространств, которые используются в последующих главах для решения задач, весьма далёких от геометрической интерпретации.

2.5.1. Векторные пространства

Пусть $\mathcal{F} = \langle F; +, \cdot \rangle$ — поле с операцией сложения $+$, операцией умножения $\cdot$, *аддитивным* нейтральным элементом (нулем) 0 и *мультипликативным* (единицей) 1. Пусть $\mathcal{V} = \langle V; + \rangle$ — абелева группа с операцией $+$ и нейтральным элементом $\mathbf{0}$. Если существует операция $F \times V \to V$ (знак этой операции опускается), такая, что для любых $a, b \in F$ и для любых $\mathbf{x}, \mathbf{y} \in V$ выполняются соотношения:

1) $(a + b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$;
2) $a(\mathbf{x} + \mathbf{y}) = a\mathbf{x} + a\mathbf{y}$;

3) $(a \cdot b)\mathbf{x} = a(b\mathbf{x})$;

4) $1\mathbf{x} = \mathbf{x}$.

то $\mathcal{V}$ называется *векторным пространством* над полем $\mathcal{F}$, элементы F называются *скалярами*, элементы V называются *векторами*, нейтральный элемент группы V называется *нуль-вектором* и обозначается $\mathbf{0}$, а необозначенная операция $F \times V \to V$ называется *умножением вектора на скаляр*.

Примеры

1. Пусть $\mathcal{F} = \langle F; +, \cdot\rangle$ — некоторое поле. Рассмотрим множество кортежей F^n. Тогда $\mathcal{F}^n = \langle F^n; +\rangle$, где $(a_1, \ldots, a_n) + (b_1, \ldots, b_n) \stackrel{\text{Def}}{=} (a_1 + b_1, \ldots, a_n + b_n)$, является абелевой группой, в которой $-(a_1, \ldots, a_n) \stackrel{\text{Def}}{=} (-a_1, \ldots, -a_n)$ и $\mathbf{0} \stackrel{\text{Def}}{=} (0, \ldots, 0)$. Положим $a(a_1, \ldots, a_n) \stackrel{\text{Def}}{=} (a \cdot a_1, \ldots, a \cdot a_n)$. Тогда $\mathcal{F}^n$ является векторным пространством над $\mathcal{F}$ для любого (конечного) n. В частности, $\mathbb{R}^n$ является векторным пространством для любого n. Векторные пространства $\mathbb{R}^2$ и $\mathbb{R}^3$ имеют естественную геометрическую интерпретацию.
2. Двоичная арифметика $\mathbb{Z}_2 = \langle E_2; +_2, \cdot\rangle$ является полем, а булеан $\langle 2^M; \triangle\rangle$ с симметрической разностью является абелевой группой. Положим $1X \stackrel{\text{Def}}{=} X$, $0X \stackrel{\text{Def}}{=} \varnothing$. Таким образом, булеан с симметрической разностью является векторным пространством над двоичной арифметикой.
3. Пусть $X = \{x_1, \ldots, x_n\}$ — произвольное конечное множество, а $\mathcal{F} = \langle F; +, \cdot\rangle$ — поле. Рассмотрим множество V_X всех формальных сумм вида $\sum\limits_{x \in X} \lambda_x x$, где $\lambda_x \in F$.

   Положим

   $$\sum_{x \in X} \lambda_x x + \sum_{x \in X} \mu_x x \stackrel{\text{Def}}{=} \sum_{x \in X} (\lambda_x + \mu_x)x, \quad \alpha \sum_{x \in X} \lambda_x x \stackrel{\text{Def}}{=} \sum_{x \in X} (\alpha \cdot \lambda_x)x,$$

   где $\alpha, \lambda_x, \mu_x \in F$. Ясно, что V_X является векторным пространством, а X является его множеством образующих. В таком случае говорят, что V_X — это векторное пространство, «натянутое» на множество X.
4. В условиях предыдущего примера ($X = \{x_1, \ldots, x_n\}$ — произвольное конечное множество, а $\mathcal{F} = \langle F; +, \cdot\rangle$ — поле) рассмотрим множество $\Phi := \{f \mid f\colon X \to F\}$. Положим

   $$(f + g)(x) \stackrel{\text{Def}}{=} f(x) + g(x), \text{ где } f, g \in \Phi, \quad (\alpha f)(x) \stackrel{\text{Def}}{=} \alpha \cdot f(x), \text{ где } \alpha \in F, f \in \Phi.$$

   Нетрудно видеть, что Φ является векторным пространством над полем F.

2.5.2. Линейные комбинации

ЛЕММА. $\forall\, \mathbf{x} \in V\ (0\mathbf{x} = \mathbf{0}), \forall\, a \in F\ (a\mathbf{0} = \mathbf{0})$.

Доказательство.

[1] $0\mathbf{x} = (1 - 1)\mathbf{x} = 1\mathbf{x} - 1\mathbf{x} = \mathbf{x} - \mathbf{x} = \mathbf{0}$.

[2] $a\mathbf{0} = a(\mathbf{0} - \mathbf{0}) = a\mathbf{0} - a\mathbf{0} = (a - a)\mathbf{0} = 0\mathbf{0} = \mathbf{0}$. □

Если $\mathcal{V}$ — векторное пространство над полем $\mathcal{F}$, S — некоторое множество векторов, $S \subset V$, то конечная сумма вида

$$\sum_{i=1}^{n} a_i \mathbf{x}_i, \quad a_i \in F, \mathbf{x}_i \in S,$$

называется *линейной комбинацией* векторов из S. Пусть $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ — конечное множество векторов. Если

$$\sum_{i=1}^{k} a_i \mathbf{x}_i = \mathbf{0} \Longrightarrow \forall\, i \in 1..k \;\; (a_i = 0)\,,$$

то множество X называется *линейно независимым*. В противном случае множество X называется *линейно зависимым*:

$$\exists\, a_1, \ldots, a_k \in F \;\; \left(\bigvee_{i=1}^{k} a_i \neq 0 \;\&\; \sum_{i=1}^{k} a_i \mathbf{x}_i = \mathbf{0} \right).$$

ТЕОРЕМА. *Линейно независимое множество векторов не содержит нуль-вектора.*

Доказательство. От противного. Пусть линейно независимое множество $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ содержит нуль-вектор и пусть для определённости $\mathbf{x}_1 = \mathbf{0}$. Положим $a_1 := 1$, $a_2 := 0, \ldots, a_k := 0$. Имеем $\sum_{i=1}^{n} a_i \mathbf{x}_i = 1\mathbf{0} + 0\mathbf{x}_2 + \cdots + 0\mathbf{x}_k\mathbf{0} = \mathbf{0}$, что противоречит линейной независимости S. □

2.5.3. Базис и размерность

Подмножество векторов $S \in V$, такое что любой элемент V может быть представлен в виде линейной комбинации элементов S, называется *порождающим* множеством пространства $\mathcal{V}$ или *множеством образующих*. Линейно независимое порождающее множество называется *базисом* векторного пространства.

ТЕОРЕМА 1. *Пусть векторное пространство $\mathcal{V}$ имеет базис $B = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$. Тогда каждый элемент векторного пространства имеет единственное представление в данном базисе.*

Доказательство. Пусть $\mathbf{x} = \sum_{i=1}^{n} a_i \mathbf{e}_i$ и $\mathbf{x} = \sum_{i=1}^{n} b_i \mathbf{e}_i$. Тогда

$$\mathbf{0} = \mathbf{x} - \mathbf{x} = \sum_{i=1}^{n} a_i \mathbf{e}_i - \sum_{i=1}^{n} b_i \mathbf{e}_i = \sum_{i=1}^{n} (a_i - b_i)\mathbf{e}_i$$

и $\forall\, i \in 1..n \;\; (a_i - b_i) = 0$, откуда $\forall\, i \;\; (a_i = b_i)$. □

Коэффициенты разложения вектора в данном базисе называются *координатами*.

ТЕОРЕМА 2. *Пусть $B = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ — базис, а $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ — линейно независимое множество векторов. Тогда $n \geqslant m$.*

Доказательство. От противного. Пусть $n < m$ и B — базис. Тогда $\exists a_1, \ldots, a_n \in F \ (\mathbf{x}_1 = a_1\mathbf{e}_1 + \ldots + a_n\mathbf{e}_n)$. Имеем $\mathbf{x}_1 \neq \mathbf{0} \Longrightarrow \bigvee_{i=1}^{n} a_i \neq 0$. Пусть для определённости $a_1 \neq 0$. Тогда $\mathbf{e}_1 = a_1^{-1}\mathbf{x}_1 - (a_1^{-1}a_2)\mathbf{e}_2 - \ldots - (a_1^{-1}a_n)\mathbf{e}_n$. Так как B порождает $\mathcal{V}$, то и $\{\mathbf{x}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\}$ тоже порождает $\mathcal{V}$. Аналогично $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{e}_3, \ldots, \mathbf{e}_n\}$ порождает $\mathcal{V}$. Продолжая процесс, получаем, что $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ порождает $\mathcal{V}$. Следовательно, $\mathbf{x}_{n+1} = \sum_{i=1}^{n} b_i\mathbf{x}_i \Longrightarrow \mathbf{x}_{n+1} - \sum_{i=1}^{n} b_i\mathbf{x}_i = \mathbf{0}$. Таким образом, X — линейно зависимое множество, что противоречит условию. □

СЛЕДСТВИЕ. *Если в векторном пространстве $\mathcal{V}$ множество векторов $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ линейно независимо и векторы множества $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_m\}$ выражаются в виде линейных комбинаций векторов множества X, другими словами, $\forall i \in 1..m \ \left(\mathbf{y}_i = \sum_{j=1}^{n} a_{ij}\mathbf{x}_j\right)$, причём $m > n$, то множество Y — линейно зависимое.*

Доказательство. Рассмотрим подпространство $\mathcal{V}_A$ векторного пространства $\mathcal{V}$, порождаемое векторами множества A. В подпространстве $\mathcal{V}_A$ множество A — базис, а векторы множества Y — элементы. Далее от противного. Пусть множество Y линейно независимое, тогда по теореме $n \geqslant m$, что противоречит условию. □

ТЕОРЕМА 3. *Если B_1 и B_2 — базисы векторного пространства $\mathcal{V}$, то $|B_1| = |B_2|$.*

Доказательство. B_1 — базис, и B_2 — линейно независимое множество. Следовательно, $|B_1| \geqslant |B_2|$. С другой стороны, B_2 — базис, и B_1 — линейно независимое множество. Следовательно, $|B_2| \geqslant |B_1|$. Имеем $|B_1| = |B_2|$. □

Если векторное пространство $\mathcal{V}$ имеет базис B, то количество элементов в базисе называется *размерностью* векторного пространства и обозначается $\dim \mathcal{V}$. Векторное пространство, имеющее конечный базис, называется *конечномерным*. Векторное пространство, не имеющее базиса (в смысле приведенного определения), называется *бесконечномерным*.

Примеры

1. Одноэлементные подмножества образуют базис булеана, $\dim 2^M = |M|$.
2. Кортежи вида $(0, \ldots, 1, 0, \ldots, 0)$ образуют базис пространства $\mathcal{F}^n$, $\dim \mathcal{F}^n = n$.

ЗАМЕЧАНИЕ

Если определить базис как максимальное линейно независимое множество (то есть множество, которое нельзя расширить, не нарушив свойства линейной независимости), то понятие базиса можно распространить и на бесконечномерные пространства.

2.5.4. Конечные поля

В этом параграфе на примере понятия конечных полей демонстрируется тесная связь различных разделов общей алгебры и теории чисел.

Поле $\mathbb{F}_n$, состоящее из конечного числа элементов n, называется *конечным полем*, или *полем Галуа*.

ЗАМЕЧАНИЕ

Иногда используется обозначение $GF(n)$ (от англ. Galois Field).

Пример. Если p — простое число, то $\mathbb{Z}_p$ — множество классов вычетов по модулю p — поле Галуа (п. 2.4.6).

Если подмножество G элементов поля F также является полем, то G называется *сужением* поля F, а F — *расширением* поля G.

Пример. Поле $\mathbb{F}_4 = \langle\{0,1,a,b\};+,*\rangle$ является расширением поля $\mathbb{Z}_2 = \langle\{0,1\};+,*\rangle$, если определить операции следующим образом:

$\mathbb{Z}_2$:

$+$	0	1
0	0	1
1	1	0

,

$*$	0	1
0	0	0
1	0	1

;

$\mathbb{F}_4$:

$+$	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

,

$*$	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

.

Введём обозначение: $qa \overset{\text{Def}}{=} \underbrace{a+a+\ldots+a}_{q \text{ раз}}$. *Порядком по сложению* элемента a из абелевой группы F называется наименьшее натуральное число q такое, что $qa = 0$. Если такого числа нет, то говорят, что порядок элемента равен бесконечности (также иногда говорят, что порядок равен нулю).

Не во всякой абелевой группе элементы имеют конечный порядок по сложению. Например, в группе $\langle\mathbb{Z};+\rangle$ все элементы, кроме 0, не имеют порядка по сложению.

ЗАМЕЧАНИЕ

Не следует путать операцию умножения элемента на число $qa = a+a+\ldots+a$ (q раз), где $a \in \mathbb{F}_n$, $q \in \mathbb{N}$, с операцией умножения в конечном поле $a * b$, где $a, b \in \mathbb{F}_n$.

ЛЕММА. *В конечном поле у любого элемента существует порядок по сложению:* $\forall\, a \in \mathbb{F}_n\ (\exists\, q \in \mathbb{N}\ (qa = 0))$.

Доказательство. Ввиду конечности поля в бесконечной последовательности $a, 2a, 3a, \ldots$ существуют повторяющиеся элементы. Пусть $q_1 a = q_2 a$, $q_2 > q_1$. Тогда $(q_2 - q_1)a = 0$, следовательно порядок элемента a не больше $q_2 - q_1$. □

ТЕОРЕМА 1. *Любое конечное поле $\mathbb{F}_n$ является расширением поля, изоморфного $\mathbb{Z}_p$, где p — простое число.*

Доказательство. Рассмотрим 1 — единицу по умножению в поле $\mathbb{F}_n$. Пусть m — порядок 1 по сложению. Если число m — составное, то есть $m = bc$ ($b > 1$, $c > 1$), то $m1 = 0 \Longrightarrow (bc)1 = 0 \Longrightarrow (b1) * (c1) = 0$. Так как $b < m$, $c < m$, имеем $b1 \neq 0$, $c1 \neq 0$. У нуля оказались ненулевые делители, что противоречит свойству полей (п. 2.3.3), значит m — простое число. Но $1 \in \mathbb{F}_n$ и $\forall\, k \in \mathbb{N}\ \ (k1 \in \mathbb{F}_n)$. Каждому элементу $k1$ ($k < m$) можно сопоставить класс вычетов k из поля вычетов $\mathbb{Z}_m$. Таким образом, в поле $\mathbb{F}_n$ можно выделить множество $G = \{0, 1, 1+1, 1+1+1, \ldots, (m-1)1\}$, изоморфное полю $\mathbb{Z}_m$. □

Число p называется *характеристикой поля* F.

ТЕОРЕМА 2. *Пусть $\mathbb{F}_n$ — конечное поле, а $\mathbb{G}$ — сужение поля $\mathbb{F}_n$ изоморфное полю $\mathbb{Z}_p$, где число p — простое. Тогда конечное поле $\mathbb{F}_n$ — векторное пространство над своим подполем $\mathbb{G}$.*

Доказательство. Элементы поля $\mathbb{F}_n$ являются векторами, сложение $+$ является векторным сложением, элементы поля $\mathbb{G}$ являются скалярами (и одновременно векторами!), а операция умножения $*$ является умножением на скаляр. Из доказательства теоремы 1 видно, что нейтральные элементы в полях $\mathbb{F}_n$ и $\mathbb{G}$ совпадают. Таким образом, все четыре аксиомы векторного пространства (п. 2.5.1) выполняются, как частные случаи свойств операций поля. □

СЛЕДСТВИЕ. *Пусть $\mathbb{F}_n$ — конечное поле, а $\mathbb{G}$ — сужение поля $\mathbb{F}_n$ изоморфное полю $\mathbb{Z}_p$, где число p — простое. Тогда существует множество $X = \{x_1, \ldots, x_m\}$ такое, что $\forall\, x \in F\ (\exists\, a_1, \ldots, a_m \in \mathbb{G}\ (x = \sum_{i=1}^{n} a_i * x_i))$.*

Доказательство. В качестве X можно взять любое порождающее линейно независимое множество, то есть базис. □

ЗАМЕЧАНИЕ

В конечном поле порядок любого элемента по сложению равен p.

ТЕОРЕМА 3. *В поле Галуа $\mathbb{F}_n$ количество элементов $n = p^k$, где $p \in \mathbb{P}$ — простое число, $k \in \mathbb{N}$ — натуральное число.*

Доказательство. Пусть p — порядок 1 по сложению. В теореме 1 показано, что в поле $\mathbb{F}_n$ существует подполе $\mathbb{G} = \{0, 1, 1{+}1, \ldots, (p{-}1)1\}$ порядка p. По теореме 2 поле $\mathbb{F}_n$ является векторным пространством над полем $\mathbb{G}$ и имеет базис $X = \{x_1, \ldots, x_k\}$. Таким образом, каждый элемент $x \in \mathbb{F}_n$ *единственным образом* представляется в виде суммы $x = \sum_{i=1}^{k} a_i{*}x_i$, и каждая сумма представляет некоторый элемент. Таких сумм ровно p^k, поскольку коэффициенты $a_1, \ldots, a_k$ принимают все p возможных значений, а k — число векторов в базисе. □

ЗАМЕЧАНИЕ

Можно показать, что верно и обратное, то есть для любого $p \in \mathbb{P}$ и $k \in \mathbb{N}$ существует поле Галуа $\mathbb{F}_{p^k}$.

2.5.5. Модули

Понятие модуля во многом аналогично понятию векторного пространства, с той лишь разницей, что векторы умножаются не на элементы из поля, а на элементы из произвольного кольца. Пусть $\mathcal{R} = \langle R; +, * \rangle$ — кольцо с операцией сложения $+$, операцией умножения $*$, нулевым элементом 0 и единичным элементом 1. Абелева группа $\mathcal{M} = \langle M; + \rangle$ называется *модулем* над кольцом $\mathcal{R}$, если задана операция умножения на скаляр $R \times M \to M$ такая, что:

1) $(a+b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$;
2) $a(\mathbf{x}+\mathbf{y}) = a\mathbf{x} + a\mathbf{y}$;
3) $(a*b)\mathbf{x} = a(b\mathbf{x})$;
4) $1\mathbf{x} = \mathbf{x}$,

где $a, b \in R$, а $\mathbf{x}, \mathbf{y} \in M$. Как и в случае векторных пространств, элементы кольца $\mathcal{R}$ называются *скалярами*, а элементы группы $\mathcal{M}$ — *векторами*.

Примеры

1. Векторное пространство $\mathcal{V}$ над полем $\mathcal{F}$ является модулем над $\mathcal{F}$, так как поле по определению является кольцом.
2. Множество векторов с целочисленными координатами в $\mathbb{R}^n$ является модулем над кольцом $\mathbb{Z}$.
3. Любая абелева группа есть модуль над кольцом $\mathbb{Z}$, если операцию умножения на скаляр $n \in \mathbb{Z}, n > 0$ определить следующим образом: $n\mathbf{v} \stackrel{\text{Def}}{=} \mathbf{v} + \mathbf{v} + \cdots + \mathbf{v}$, где в сумму входит n слагаемых.

Если кольцо некоммутативное, то различают *левые модули*, в которых аксиомы записываются так, как указано выше, и *правые модули*, в которых третья аксиома имеет вид $(a*b)\mathbf{x} = b(a\mathbf{x})$.

2.6. Решётки

Решётки иногда называют «структурами», но слово «структура» перегружено, и мы не будем использовать его в этом значении. Решётки сами по себе часто встречаются в разных программистских задачах, но еще важнее то, что понятие решётки непосредственно подводит к понятию булевой алгебры, значение которой для основ современной двоичной компьютерной техники трудно переоценить.

2.6.1. Дистрибутивные и ограниченные решётки

Решётка — это множество M с двумя бинарными операциями, $\cap$ и $\cup$, такими, что выполнены следующие условия (аксиомы решётки):

1) идемпотентность: $a \cap a = a, \quad a \cup a = a$;
2) коммутативность: $a \cap b = b \cap a, \quad a \cup b = b \cup a$;
3) ассоциативность: $(a \cap b) \cap c = a \cap (b \cap c), \quad (a \cup b) \cup c = a \cup (b \cup c)$;
4) поглощение: $(a \cap b) \cup a = a, \quad (a \cup b) \cap a = a$.

Решётка называется *дистрибутивной*, если $a \cup (b \cap c) = (a \cup b) \cap (a \cup c)$ и $a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$.

Если в решётке $\exists 0 \in M \ (\forall a \ (0 \cap a = 0))$, то 0 называется *нулем* (или *нижней гранью*) решётки. Если в решётке $\exists 1 \in M \ (\forall a \ (1 \cup a = 1))$, то 1 называется *единицей* (или *верхней гранью*) решётки. Решётка с верхней и нижней гранями называется *ограниченной*.

ТЕОРЕМА 1. *Если нижняя (верхняя) грань существует, то она единственна.*

Доказательство. Пусть $0'$ — второй нуль решётки. Тогда $0' = 0 \cap 0' = 0' \cap 0 = 0$. $\square$

Примеры

1. Булеан 2^M конечного множества M образует ограниченную дистрибутивную решётку относительно операций пересечения $\cap$ и объединения $\cup$. Пустое множество $\varnothing$ образует нижнюю грань решётки, а множество M образует верхнюю грань решётки. Выполнение аксиом решётки проверено в п. 1.2.9.
2. Множество натуральных чисел, где в качестве операции $\cap$ взят наибольший общий делитель (п. 2.4.2), а в качестве $\cup$ — наименьшее общее кратное (п. 2.4.3) является дистрибутивной решёткой с нижней гранью. Проверим аксиомы дистрибутивной решётки. Идемпотентность и коммутативность очевидны. Ассоциативность проверяется с использованием свойств делимости. Пусть $x = \gcd(\gcd(a,b),c)$, $y = \gcd(a,\gcd(b,c))$. Тогда $x|\gcd(a,b)$ & $x|c \Longrightarrow$ $\Longrightarrow x|a$ & $x|b$ & $x|c \Longrightarrow x|a$ & $x|\gcd(b,c) \Longrightarrow x|y$. Аналогично в обратную сторону и для второй операции. Поглощение вытекает из теоремы п. 2.4.3 и следствия к лемме из п. 2.4.2. Дистрибутивность проще всего показать, опираясь на следствие из основной теоремы арифметики (п. 2.4.4). В этой решётке число 1 является нижней гранью, верхней грани эта решётка не имеет.

ТЕОРЕМА 2. $a \cap b = b \Longleftrightarrow a \cup b = a$.

Доказательство.

[$\Longrightarrow$] $a \cup b = a \cup (a \cap b) = (a \cap b) \cup a = a$.

[$\Longleftarrow$] $a \cap b = (a \cup b) \cap b = (b \cup a) \cap b = b$. □

СЛЕДСТВИЕ. $0 \cup a = 1 \cap a = a$.

2.6.2. Решётки с дополнением

В ограниченной решётке элемент a' называется *дополнением* элемента a, если $a \cap a' = 0$ и $a \cup a' = 1$. Если дополнение существует у любого элемента, то ограниченная решётка называется *решёткой с дополнением*. Вообще говоря, дополнение не обязано существовать и не обязано быть единственным.

Пример. Множество всех *нечётких* подмножеств некоторого множества M образует, как нетрудно проверить, ограниченную дистрибутивную решётку относительно пересечения и объединения (п. 1.9.6). Однако, хотя для нечётких множеств дополнение определено, требуемые в решётке свойства дополнения не выполняются: $\min(\mu_A, 1-\mu_A) \neq 0$, $\max(\mu_A, 1-\mu_A) \neq 1$.

ТЕОРЕМА (о свойствах дополнения). *В ограниченной дистрибутивной решётке с дополнением выполняется следующее:*

1) *дополнение a' единственно;*
2) *дополнение инволютивно: $a'' = a$;*
3) *грани дополняют друг друга: $1' = 0, 0' = 1$;*
4) *выполняются законы де Моргана: $(a \cap b)' = a' \cup b'$, $(a \cup b)' = a' \cap b'$.*

Доказательство.

[1] Пусть x, y — дополнения a. Тогда $x = x \cap 1 = x \cap (a \cup y) = (x \cap a) \cup (x \cap y) = 0 \cup (x \cap y) = x \cap y = y \cap x = 0 \cup (y \cap x) = (y \cap a) \cup (y \cap x) = y \cap (a \cup x) = y \cap 1 = y$;

[2] $(a \cup a' = 1 \Longrightarrow a' \cup a = 1,\ a \cap a' = 0 \Longrightarrow a' \cap a = 0) \Longrightarrow a = a''$;

[3] $(1 \cap 0 = 0, 0' \cap 0 = 0) \Longrightarrow 1 = 0'$, $(1 \cup 0 = 1, 1 \cup 1' = 1) \Longrightarrow 0 = 1'$;

[4] $(a \cap b) \cap (a' \cup b') = (a \cap b \cap a') \cup (a \cap b \cap b') = (0 \cap b) \cup (a \cap 0) = 0 \cup 0 = 0$,
$(a \cap b) \cup (a' \cup b') = (a \cup a' \cup b') \cap (b \cup a' \cup b') = (1 \cup b') \cap (1 \cup a') = 1 \cap 1 = 1$,
$(a \cup b) \cap (a' \cap b') = (a \cap a' \cap b') \cup (b \cap a' \cap b') = (0 \cap b') \cup (a' \cap 0) = 0 \cup 0 = 0$,
$(a \cup b) \cup (a' \cap b') = (a \cup b \cup a') \cap (a \cup b \cup b') = (1 \cup b) \cap (1 \cup a) = 1 \cap 1 = 1$. □

2.6.3. Частичный порядок в решётке

В любой решётке можно естественным образом ввести частичный порядок, а именно: $a \prec b \overset{\text{Def}}{=} a \cap b = a$.

ТЕОРЕМА 1. *Отношение $\prec$ является отношением частичного порядка.*

Доказательство.

[Рефлексивность] $a \cap a = a \Longrightarrow a \prec a$.

[Антисимметричность] $a \prec b \ \&\ b \prec a \Longrightarrow a = a \cap b = b \cap a = b$.

[Транзитивность] $a \prec b \ \&\ b \prec c \Longrightarrow a \cap c = (a \cap b) \cap c = a \cap (b \cap c) = a \cap b = a \Longrightarrow a \prec c$. □

Наличие частичного порядка в решётке не случайно, это её характеристическое свойство. Более того, обычно решётку определяют, начиная с частичного порядка, следующим образом. Пусть M — частично упорядоченное множество с частичным порядком $\prec$. Напомним (п. 1.8.5), что элемент x называется *нижней границей* для a и b, если $x \prec a \ \&\ x \prec b$. Аналогично, y называется *верхней границей* для a и b, если $a \prec y \ \&\ b \prec y$. Элемент x называется *нижней гранью* элементов a и b, если x — нижняя граница элементов a и b и для любой другой нижней границы v элементов a и b выполняется $v \prec x$. Обозначение: $x = \inf(a, b)$.

Аналогично, y называется *верхней гранью* элементов a и b, если y — верхняя граница элементов a и b и для любой другой верхней границы u элементов a и b выполняется $y \prec u$. Обозначение: $y = \sup(a, b)$.

ЛЕММА. *Если нижняя (верхняя) грань любых двух элементов существует, то она единственна.*

Доказательство. $x = \inf(a, b) \ \&\ y = \inf(a, b) \Longrightarrow y \prec x \ \&\ x \prec y \Longrightarrow x = y$. □

ТЕОРЕМА 2. *Если в частично упорядоченном множестве для любых двух элементов существуют нижняя и верхняя грани, то это множество образует решётку относительно* inf *и* sup.

Доказательство. Пусть $x \cap y := \inf(x, y)$, $x \cup y := \sup(x, y)$.

[1] $\inf(x, x) = x \Longrightarrow x \cap x = x$, $\sup(x, x) = x \Longrightarrow x \cup x = x$.

[2] $\inf(x, y) = \inf(y, x) \Longrightarrow x \cap y = y \cap x$, $\sup(x, y) = \sup(y, x) \Longrightarrow x \cup y = y \cup x$.

[3] $\inf\bigl(x,\inf(y,z)\bigr)=\inf\bigl(\inf(x,y),z\bigr)\Longrightarrow x\cap(y\cap z)=(x\cap y)\cap z$,
$\sup\bigl(x,\sup(y,z)\bigr)=\sup\bigl(\sup(x,y),z\bigr)\Longrightarrow x\cup(y\cup z)=(x\cup y)\cup z$.

[4] $\sup\bigl(\inf(a,b),a\bigr)=a\Longrightarrow(a\cap b)\cup a=a$, $\inf\bigl(\sup(a,b),a\bigr)=a\Longrightarrow(a\cup b)\cap a=a$. □

Пример. Включение $\subset$ является естественным частичным порядком на булеане $\langle 2^M;\cap,\cup\rangle$, поскольку $A\subset B\Longleftrightarrow A\cap B=A$.

2.6.4. Полные решётки

Частичный порядок можно естественным образом ввести в любой решётке, но при этом не всякое подмножество носителя решётки обязано иметь верхнюю и нижнюю грани. Решётка называется *полной*, если любое подмножество имеет супремум и инфимум. Полная решётка L является ограниченной, если положить $\mathbf{0}:=\inf L$, $\mathbf{1}:=\sup L$. Кроме того, ясно, что полная решётка является линейно полным частично упорядоченным множеством. Свойство полноты решётки является более сильным, чем свойство линейной полноты в частично упорядоченных множествах, поэтому в полной решётке справедлива теорема о наименьшей неподвижной точке функции при более слабом предположении о свойствах функции.

Пример. Булеан является полной решеткой, поскольку если $X\subset 2^M$, то $\sup X=\bigcup\limits_{Y\in X}Y$, $\inf X=\bigcap\limits_{Y\in X}Y$.

ТЕОРЕМА. *Если L — полная решетка, а функция $f\colon L\to L$ монотонна, то функция f имеет наименьшую неподвижную точку.*

Доказательство. Рассмотрим множество $A:=\{x\in L\mid f(x)\prec x\}$ и существующий по условию теоремы элемент $a=\inf A$. Имеем $\forall\, x\in A\ \ (a\prec x)$, и значит, $\forall\, x\in A\ \ (f(a)\prec f(x)\prec x)$, следовательно, $f(a)$ — нижняя граница для множества A и $f(a)\prec a$. Далее, $f(f(a))\prec f(a)$, и значит, $f(a)\in A$, откуда по определению элемента a имеем $a\prec f(a)$ и по антисимметричности $f(a)=a$. Таким образом, a — неподвижная точка функции f. Пусть теперь b — любая неподвижная точка функции f, то есть $f(b)=b$. По рефлексивности естественного частичного порядка $f(b)\prec b$. Тем самым $b\in A$ и значит $a\prec b$. □

2.6.5. Полурешётки

Естественный частичный порядок в решётке можно определить одним из двух способов: $a\prec b:=a\cap b=a$ или $a\prec b:=a\cup b=b$. Из теоремы п. 2.5.2 следует, что это одно и то же отношение. Таким образом, для определения частичного порядка достаточно иметь только одну подходящую операцию.

Множество M с операцией $\cap\colon M\times M\to M$ называется *полурешёткой*, если выполнены следующие условия (аксиомы полурешётки):

1) идемпотентность: $a\cap a=a$;
2) коммутативность: $a\cap b=b\cap a$;
3) ассоциативность: $a\cap(b\cap c)=(a\cap b)\cap c$.

Такой набор свойств встречается достаточно часто, так что многие операции образуют полурешётку.

Пример. Операции объединения и пересечения множеств образуют полурешётки.

Как видно из доказательства теоремы п. 2.5.4, аксиом полурешетки достаточно для определения естественного частичного порядка $a \prec b := a \cap b = a$.

ЗАМЕЧАНИЕ

Для доказательства рефлексивности используется идемпотентность, для доказательства антисимметричности — коммутативность, а транзитивность следует из ассоциативности.

Функция $f\colon M \to M$, определённая на полурешётке M, называется *дистрибутивной*, если $f(x \cap y) = f(x) \cap f(y)$.

ЗАМЕЧАНИЕ

В терминах п. 2.1.5 дистрибутивная функция — это гомоморфизм.

Нетрудно видеть, что дистрибутивная функция f монотонна:

$$x \prec y \Longrightarrow x = x \cap y \Longrightarrow f(x) = f(x \cap y) = f(x) \cap f(y) \Longrightarrow f(x) \prec f(y).$$

Пример. Пересечение в булеане монотонно, а разность — нет.

Полурешётка называется *ограниченной*, если у неё существуют верхняя и нижняя грань.

Поскольку естественный частичный порядок в полурешётке определён, определены и все другие понятия, построенные на его основе:

1) верхние и нижние границы;
2) верхние и нижние грани (супремум и инфимум);
3) максимальные и минимальные элементы;
4) наибольшие и наименьшие элементы;
5) линейная полнота;
6) конечная высота.

ТЕОРЕМА. *Если L — ограниченная полурешётка конечной высоты, а $f\colon L \to L$ — монотонная функция, то функция f имеет наименьшую неподвижную точку, которая может быть получена методом итераций из нижней грани.*

Доказательство. Поскольку полурешётка ограничена, ее нижняя грань $\mathbf{0}$ является наименьшим элементом. Поскольку полурешётка имеет конечную высоту, всякое линейно упорядоченное подмножество можно рассматривать как монотонную последовательность, которая заканчивается наибольшим элементом (супремумом). В частности, последовательность $\langle f^0(\mathbf{0}), f^1(\mathbf{0}), f^2(\mathbf{0}), \dots \rangle$ монотонна в силу монотонности функции f и имеет супремум $a = \sup\{f^n(\mathbf{0}) \mid n \geqslant 0\}$, который является наименьшей неподвижной точкой (см. доказательство теоремы о наименьшей неподвижной точке в п. 1.8.7). □

ЗАМЕЧАНИЕ

Можно показать, что множество неподвижных точек монотонной функции на ограниченной полурешётке конечной высоты само образует ограниченную полурешётку конечной высоты.

Пример. Рассмотрим конечное множество A и булеан 2^A. Ясно, что 2^A является ограниченной полурешёткой конечной высоты относительно пересечения. Рассмотрим функцию $f\colon 2^A \to 2^A$, где $f(X) := X + a$. Наименьшей неподвижной точкой этой функции является $\{a\}$. Действительно, $f(\varnothing) = a$, $f(a) = a$, $f(f(a)) = a, \ldots$. Более того, все подмножества 2^A, содержащие элемент a, являются неподвижными точками функции f и образуют ограниченную полурешётку конечной высоты.

2.6.6. Булевы алгебры

Дистрибутивная ограниченная решётка, в которой для каждого элемента существует дополнение, называется *булевой алгеброй*. Свойства булевой алгебры:

1) $a \cup a = a$, $a \cap a = a$ — по определению решётки;
2) $a \cup b = b \cup a$, $a \cap b = b \cap a$ — по определению решётки;
3) $a \cup (b \cup c) = (a \cup b) \cup c$, $a \cap (b \cap c) = (a \cap b) \cap c$ — по определению решётки;
4) $(a \cap b) \cup a = a$, $(a \cup b) \cap a = a$ — по определению решётки;
5) $a \cup (b \cap c) = (a \cup b) \cap (a \cup c)$, $a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$ — по дистрибутивности;
6) $a \cup 1 = 1$, $a \cap 0 = 0$ — по ограниченности;
7) $a \cup 0 = a$, $a \cap 1 = a$ — по следствию из теоремы п. 2.5.2;
8) $a'' = a$ — по теореме п. 2.5.3;
9) $(a \cap b)' = a' \cup b'$, $(a \cup b)' = a' \cap b'$ по теореме п. 2.5.3;
10) $a \cup a' = 1$, $a \cap a' = 0$ — по определению.

Примеры

1. $\langle 2^M; \cap, \cup, {}^- \rangle$ — булева алгебра, причём M — верхняя грань, $\varnothing$ — нижняя грань, $\subset$ — естественный частичный порядок.
2. $\langle E_2; \,\&\,, \vee, \neg \rangle$ — булева алгебра, где $\&$ — конъюнкция, $\vee$ — дизъюнкция, $\neg$ — отрицание, причём 1 — верхняя грань, 0 — нижняя грань, а импликация $\Longrightarrow$ — естественный частичный порядок.
3. Пусть T — некоторое конечное множество взаимно простых чисел. Пусть P — множество всех возможных произведений различных чисел из T, включая пустое произведение, по определению равное 1. Заметим, что $P \sim 2^T$. Тогда $\langle P; \gcd, \operatorname{lcm}, ()' \rangle$ — булева алгебра, где gcd — наибольший общий делитель, lcm — наименьшее общее кратное, а $(n)' \stackrel{\text{Def}}{=} \frac{1}{n} \prod\limits_{q \in T} q$, причём $\prod\limits_{q \in T} q$ — верхняя грань, 1 — нижняя грань, отношение «$n|m$» — естественный частичный порядок.

2.7. Матроиды и жадные алгоритмы

В этом разделе рассматривается следующая простая и часто встречающаяся экстремальная задача. Задано конечное множество E, элементам которого приписаны положительные веса, и семейство $\mathcal{E} \subset 2^E$ подмножеств множества E. Требуется найти в семействе $\mathcal{E}$ элемент (подмножество E) максимального суммарного веса. Оказывается, что если семейство $\mathcal{E}$ обладает особой структурой (является матроидом), то для решения задачи достаточно применить очень простой и эффективный алгоритм (жадный алгоритм). Ясное понимание природы и области применимости жадных алгоритмов совершенно необходимо каждому программисту. Матроиды,

рассматриваемые в этом разделе, вообще говоря, не являются алгебраическими структурами в том смысле, который придан этому понятию в первом разделе данной главы. Однако матроиды имеют много общего с рассмотренными алгебраическими структурами (в частности, с линейно независимыми множествами в векторных пространствах и модулях) и изучаются сходными методами.

2.7.1. Матроиды

Матроидом $M = \langle E, \mathcal{E} \rangle$ называется пара, состоящая из конечного множества E, $|E| = n$, и семейства его подмножеств $\mathcal{E} \subset 2^E$, таких, что выполняются следующие три аксиомы:

M_1: $\varnothing \in \mathcal{E}$;
M_2: $A \in \mathcal{E}\ \&\ B \subset A \Longrightarrow B \in \mathcal{E}$;
M_3: $A, B \in \mathcal{E}\ \&\ |B| = |A| + 1 \Longrightarrow \exists\, e \in B \setminus A\ \ (A + e \in \mathcal{E})$.

Элементы множества $\mathcal{E}$ называются *независимыми*, а остальные подмножества E (то есть элементы множества $2^E \setminus \mathcal{E}$) — *зависимыми* множествами.

ЗАМЕЧАНИЕ

Аксиома M_1 исключает из рассмотрения вырожденный случай $\mathcal{E} = \varnothing$.

Пример. Семейство линейно независимых множеств векторов любого векторного пространства является матроидом. Аксиомы M_1 и M_2 выполняются. Пусть $A := \{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$ и $B := \{\mathbf{b}_1, \ldots, \mathbf{b}_{m+1}\}$ — линейно независимые множества векторов. Если бы все векторы из множества B выражались в виде линейной комбинации векторов из множества A, то множество B было бы линейно зависимым по следствию к теореме п. 2.4.3. Стало быть, среди векторов множества B есть по крайней мере один вектор $\mathbf{b}$, который не входит в множество A и не выражается в виде линейной комбинации векторов из множества A. Добавление вектора $\mathbf{b}$ к множеству A образует линейно независимое множество.

ЗАМЕЧАНИЕ

Само понятие матроида возникло в результате исследования линейной независимости в векторных пространствах.

2.7.2. Максимальные независимые подмножества

Пусть $M = \langle E, \mathcal{E} \rangle$ — матроид и $X \subset E$ — произвольное множество. *Максимальным* независимым подмножеством множества X называется множество Y, такое, что $Y \subset X\ \&\ Y \in \mathcal{E}\ \&\ \forall Z \in \mathcal{E}\ \ (Y \subset Z \subset X \Longrightarrow Z = Y)$. Множество максимальных независимых подмножеств множества X обозначим $\underline{X}$: $\underline{X} \stackrel{\text{Def}}{=} \{Y \subset E \mid Y \subset X\ \&\ Y \in \mathcal{E}\ \&\ \forall Z \in \mathcal{E}\ \ (Y \subset Z \subset X \Longrightarrow Z = Y)\}$. Рассмотрим следующее утверждение:

M_4: $\forall X\ \ (Y \in \underline{X}\ \&\ Z \in \underline{X} \Longrightarrow |Y| = |Z|)$,

то есть все максимальные независимые подмножества данного множества равномощны.

ТЕОРЕМА. *Пусть* $M = \langle E, \mathcal{E} \rangle$ *и выполнены аксиомы* M_1 *и* M_2. *Тогда аксиома* M_3

и утверждение M_4 эквивалентны, то есть

$$A, B \in \mathcal{E} \;\&\; |B| = |A| + 1 \Longrightarrow \exists e \in B \setminus A \;\; (A + e \in \mathcal{E}),$$

тогда и только тогда, когда $\forall X \;\; (Y \in \underline{X} \;\&\; Z \in \underline{X} \Longrightarrow |Y| = |Z|)$.

Доказательство.

[$\Longrightarrow$] Пусть выполнены утверждения M_1, M_2, M_3 (то есть M — матроид). Покажем от противного, что выполняется и M_4. Пусть $Y, Z \in \underline{X}$, $|Y| \neq |Z|$ и для определённости $|Y| > |Z|$. Возьмем $Y' \subset Y$, так что $|Y'| = |Z| + 1$. Тогда по свойству M_3 имеем $\exists e \in Y' \setminus Z \;\; (W := Z + e \in \mathcal{E})$. Таким образом, имеем $W \in \mathcal{E}$, $Z \subset W$, $W \subseteq X$, $Z \neq W$, что противоречит предположению $Z \in \underline{X}$.

[$\Longleftarrow$] Пусть выполнены утверждения M_1, M_2, M_4. Покажем от противного, что выполняется и M_3. Возьмем $A, B \in \mathcal{E}$, так что $|B| = |A|+1$. Допустим, что $\neg\exists e \in B \setminus A$ $(A + e \in \mathcal{E})$, то есть $\forall e \in B \setminus A \;\; (A + e \notin \mathcal{E})$. Рассмотрим $C := A \cup B$. Имеем $A \in \underline{C}$. Но $B \in \mathcal{E}$, поэтому $\exists B' \;\; (B \subset B' \;\&\; B' \in \mathcal{E} \;\&\; B' \in \underline{C})$. По условию M_4 имеем $|B'| = |A|$. Но $|A| = |B'| \geqslant |B| = |A| + 1$ — противоречие. □

ЗАМЕЧАНИЕ

Таким образом, M_1, M_2, M_4 — эквивалентная система аксиом матроида.

2.7.3. Базисы

Максимальные независимые подмножества множества E называются *базисами* матроида $M = \langle E, \mathcal{E} \rangle$. Всякий матроид имеет базисы, что следует из алгоритма 2.5.

Алгоритм 2.5. Построение базиса матроида

Вход: Матроид $M = \langle E, \mathcal{E} \rangle$.
Выход: Множество $B \subset E$ элементов, образующих базис.

```
B := ∅  // вначале базис пуст
for e ∈ E do
  if B + e ∈ ℰ then
    B := B + e  // расширяем базис допустимым образом
  end if
end for
```

Обоснование. Пусть $B_0 = \varnothing$, $B_1, ..., B_k = B$ — последовательность значений переменной B в процессе работы алгоритма 2.5. По построению $\forall i \;\; (B_i \in \mathcal{E})$. Пусть $B \notin \underline{E}$. Тогда $\exists B' \;\; (B \subset B' \;\&\; B' \neq B \;\&\; B' \in \mathcal{E})$. $\exists B' \;\; (B \subset B' \;\&\; B' \neq B \;\&\; B' \in \mathcal{E})$. Возьмем $B'' \subset B'$, так что $|B''| = |B| + 1$, $B \subset B''$ и $B'' \in \mathcal{E}$. Рассмотрим $e \in B' \setminus B$. Элемент $e \notin B$, значит $\exists i \;\; ((B_{i-1} + e) \notin \mathcal{E})$. Но $e \in B'' \;\&\; B_{i-1} \subset B'' \Longrightarrow (B_{i-1} + e) \in \mathcal{E}$. По аксиоме M_2 имеем $(B_{i-1} + e) \in \mathcal{E}$ — противоречие. □

СЛЕДСТВИЕ. *Все базисы матроида равномощны.*

2.7.4. Жадный алгоритм

Сформулируем более точно рассматриваемую экстремальную задачу. Пусть имеются конечное множество E, $|E| = n$, *весовая* функция $w\colon E \to \mathbb{R}_+$ и семейство $\mathcal{E} \subset 2^E$. Требуется найти $X \in \mathcal{E}$, такое, что

$$w(X) = \max_{Y \in \mathcal{E}} w(Y), \quad \text{где } w(Z) \stackrel{\text{Def}}{=} \sum_{e \in Z \subset E} w(e).$$

Другими словами, необходимо выбрать в указанном семействе подмножество наибольшего веса. Не ограничивая общности, можно считать, что веса упорядочены по убыванию $w(e_1) \geqslant \ldots \geqslant w(e_n) > 0$. Рассмотрим алгоритм 2.6.

Алгоритм 2.6. Жадный алгоритм

Вход: множество $E = \{e_1, \ldots, e_n\}$, семейство его подмножеств $\mathcal{E}$ и весовая функция w, $w(e_1) \geqslant \ldots \geqslant w(e_n) > 0$.

Выход: подмножество X, такое, что $X \in \mathcal{E}$ и $w(X) \leqslant \max\limits_{Y \in \mathcal{E}} w(Y)$.

```
X := ∅  // вначале множество X пусто
for i from 1 to n do
  if X + e_i ∈ E then
    X := X + e_i  // добавляем в X первый подходящий элемент
  end if
end for
```

Алгоритм такого типа называется *жадным*. Совершенно очевидно, что по построению окончательное множество $X \in \mathcal{E}$. Также очевидно, что жадный алгоритм является чрезвычайно эффективным: количество шагов составляет $O(n)$, то есть жадный алгоритм является *линейным*. (Не считая затрат на сортировку множества E и проверку независимости $X + e_i \in \mathcal{E}$.) Возникает вопрос: в каких случаях жадный алгоритм действительно решает задачу, поставленную в начале параграфа? Другими словами: когда выгодно быть жадным?

Пример. Пусть дана матрица (a). В задаче «выбрать по одному элементу *из каждого столбца* так, чтобы их сумма была максимальна» жадный алгоритм выберет элементы (b), которые действительно являются решением задачи. В то же время, в задаче «выбрать по одному элементу *из каждого столбца и из каждой строки* так, чтобы их сумма была максимальна» жадный алгоритм выберет элементы (c), которые не являются решением, потому что (d) — лучшее решение.

$$(a)\begin{pmatrix}7&5&1\\3&4&3\\2&3&1\end{pmatrix} \quad (b)\begin{pmatrix}\boxed{7}&\boxed{5}&1\\3&4&\boxed{3}\\2&3&1\end{pmatrix} \quad (c)\begin{pmatrix}\boxed{7}&5&1\\3&\boxed{4}&3\\2&3&\boxed{1}\end{pmatrix} \quad (d)\begin{pmatrix}\boxed{7}&5&1\\3&4&\boxed{3}\\2&\boxed{3}&1\end{pmatrix}$$

ТЕОРЕМА. *Если $M = \langle E, \mathcal{E}\rangle$ — матроид, то для любой функции w жадный алгоритм находит независимое множество X с наибольшим весом; обратно, если же $M = \langle E, \mathcal{E}\rangle$ не является матроидом, то существует такая функция w, что множество X, найденное жадным алгоритмом, не будет подмножеством наибольшего веса.*

Доказательство. Пусть $M = \langle E, \mathcal{E}\rangle$ — матроид и $X = \{x_1, \ldots, x_k\}$ — множество, построенное жадным алгоритмом. По построению $w(x_1) \geqslant \ldots \geqslant w(x_k) > 0$ и множество X — базис матроида M. Пусть теперь $Y = \{y_1, \ldots, y_m\} \in \mathcal{E}$ — некоторое независимое множество, $w(y_1) \geqslant \ldots \geqslant w(y_m) > 0$. Имеем $m \leqslant k$. Покажем от противного, что $\forall\, i \in 1..m\ \ (w(y_i) \leqslant w(x_i))$. Пусть $\exists\, i \in 1..m\ \ (w(y_i) > w(x_i))$. Рассмотрим независимые множества $A = \{x_1, \ldots, x_{i-1}\}$ и $B = \{y_1, \ldots, y_{i-1}, y_i\}$. Имеем $\exists\, j \leqslant i$ $(\{x_1, \ldots, x_{i-1}, y_j\})$ — независимое множество. Тогда $w(y_j) \geqslant w(y_i) > w(x_i)$, откуда следует, что $\exists\, p \leqslant i\ \ (w(x_1) \geqslant \ldots \geqslant w(x_{p-1}) \geqslant w(y_j) \geqslant w(x_p))$, что противоречит тому, что x_p — элемент с наибольшим весом, добавление которого к множеству $\{x_1, \ldots, x_{p-1}\}$ не нарушает независимости. Следовательно, $\forall\, i\ \ (w(y_i)) \leqslant w(x_i)$, и значит, $w(Y) \leqslant w(X)$. Пусть теперь $M = \langle E; \mathcal{E}\rangle$ не является матроидом. Если нарушено условие M_2, то есть $\exists\, A, B\ \ (A \subset B \in \mathcal{E})$, но $A \notin \mathcal{E}$, то определим функцию w следующим образом: $e \in A \Longrightarrow w(e) := 1$, $e \in E \setminus A \Longrightarrow w(e) := 0$. Тогда $A \not\subset X \Longrightarrow w(X) < w(A) = w(B)$. Если же условие M_2 выполнено, но нарушено условие M_3, то $\exists\, A, B\ \ (|A| = k)\ \&\ |B| = k + 1$ и для любого элемента $e \in B \setminus A$ множество $A + e$ — зависимое. Пусть $p := |A \cap B|$. Тогда $p < k$. Выберем число ε так, что $0 < \varepsilon < 1/(k - p)$. Определим функцию w: $w(e) :=$ **if** $e \in A$ **then** $1 + \varepsilon$ **else if** $e \in B \setminus A$ **then** 1 **else** 0 **end if**. Заметим, что при таких весах жадный алгоритм сначала выберет все элементы из A и отбросит элементы из $B \setminus A$. В результате будет выбрано множество X, вес которого меньше веса множества B. Действительно, $w(X) = w(A) = k(1+\varepsilon) = (k-p)(1+\varepsilon) + p(1+\varepsilon) < (k-p)(1+1/(k-p)) + p(1+\varepsilon) =$ $= (k-p+1) + p(1+\varepsilon) = w(B)$. □

ЗАМЕЧАНИЕ

Тот факт, что семейство $\mathcal{E}$ является матроидом, означает, что для решения поставленной экстремальной задачи можно применить жадный алгоритм, однако из этого не следует, что не может существовать ещё более эффективного алгоритма. С другой стороны, если семейство $\mathcal{E}$ не является матроидом, то это ещё не значит, что жадный алгоритм не найдёт правильного решения — все зависит от свойств конкретной функции w.

ОТСТУПЛЕНИЕ

Жадные алгоритмы были исследованы сравнительно недавно, но их значение в практике программирования чрезвычайно велико. Если удаётся свести конкретную экстремальную задачу к такой постановке, где множество допустимых вариантов является матроидом, то в большинстве случаев следует сразу применять жадный алгоритм, поскольку он достаточно эффективен в практическом смысле. Если же, наоборот, оказывается, что множество допустимых вариантов не образует матроида, то это «плохой признак». Скорее всего, данная задача окажется труднорешаемой. В этом случае целесообразно исследовать задачу, чтобы избежать бесплодных попыток «изобрести» эффективный алгоритм там, где это на самом деле невозможно.

2.7.5. Примеры матроидов

Ниже приведены некоторые примеры матроидов, встречающихся в рассмотренных областях дискретной математики. В последних главах книги имеются дополнительные примеры матроидов, связанных с графами.

1. *Свободные матроиды.* Если E — произвольное конечное множество, то $M = \langle E, 2^E\rangle$ — матроид. Такой матроид называется *свободным*. В свободном матроиде каждое множество независимое.

2. *Матроиды разбиений.* Пусть $\{E_1, \ldots, E_k\}$ — некоторое разбиение множества E на непустые множества. Другими словами, $\bigcup_{i=1}^{k} E_k = E$, $E_i \cap E_j = \varnothing$, $E_j \neq \varnothing$. Положим $\mathcal{E} := \{A \subseteq E \mid 1 \geqslant |A \cap E_i|\}$, то есть в независимое множество входит не более чем один элемент каждого блока разбиения. Тогда $M = \langle E, \mathcal{E}\rangle$ — матроид. Действительно, условие M_2 выполнено. Если $A, B \in \mathcal{E}$ и $|B| = |A|+1$, то $\exists\, i \;\; (|E_i \cap A| = 0 \;\&\; |E_i \cap B| = 1)$. Обозначим $e := E_i \cap B$. Тогда $A + e \in \mathcal{E}$. Значит, выполнено условие M_3.
3. *Векторные пространства.* Линейно независимые множества векторов любого векторного пространства образуют матроид. Действительно, условия M_1 и M_2, очевидно, выполнены для линейно независимых множеств векторов. Условие M_4 выполнено по теореме п. 2.4.3.
4. *Матроид трансверсалей.* Пусть E — некоторое множество и $\mathcal{E}$ — семейство подмножеств этого множества (не обязательно различных), $\mathcal{E} \subset 2^E$. Подмножество $A \subset E$ называется *частичной трансверсалью* семейства $\mathcal{E}$, если A содержит не более чем по одному элементу для каждого подмножества E_i семейства $\mathcal{E}$. Частичные трансверсали над E образуют матроид.

Глава 3 Булевы функции

Занимаясь исследованием законов мышления, английский математик Джордж Буль[1] применил в логике систему формальных обозначений и правил, близкую к математической. Так возникла алгебра логики, или булева алгебра. Значение логической алгебры долгое время игнорировалось, поскольку она не находила применения в науке и технике того времени. С появлением электронной техники операции, введенные Булем, оказались весьма полезны. Они изначально ориентированы на работу только с двумя сущностями — истина и ложь, что хорошо согласуется с двоичным кодом, который в современных компьютерах также представляется всего двумя сигналами — ноль и единица.

Данная глава имеет двойное назначение. С одной стороны, помимо основных фактов из теории булевых функций здесь затрагиваются весьма общие понятия, такие как *реализация функций формулами*, *нормальные формы*, *двойственность*, *полнота*. Эти понятия затруднительно описать исчерпывающим образом на выбранном элементарном уровне изложения, но знакомство с ними необходимо. Поэтому рассматриваются частные случаи указанных понятий на простейшем примере булевых функций. С другой стороны, материал этой главы служит для «накопления фактов» и овладения базисной логической техникой, что должно создать у читателя необходимый эффект «узнавания знакомого» при изучении довольно абстрактного и формального материала следующей главы.

3.1. Элементарные булевы функции

Подобно тому как в классической математике знакомство с основами анализа начинают с изучения *элементарных функций* (рациональные функции от вещественной переменной x, e^x, $\ln x$ и их суперпозиции), изложение теории булевых функций естественно начать с выделения, идентификации и изучения *элементарных булевых функций* (дизъюнкция, конъюнкция и т. д.).

3.1.1. Функции алгебры логики

Функции $f\colon E_2^n \to E_2$, где $E_2 \stackrel{\text{Def}}{=} \{0,1\}$, называются *функциями алгебры логики*, или *булевыми функциями* от n переменных. Элементы множества $E_2 = \{0,1\}$ называются *истинностными значениями*. Множество булевых функций от n переменных обозначим P_n, $P_n \stackrel{\text{Def}}{=} \{f \mid f\colon E_2^n \to E_2\}$.

Булеву функцию от n переменных можно задать *таблицей истинности*:

x_1	...	x_{n-1}	x_n	$f(x_1,\ldots,x_n)$
0	...	0	0	$f(0,\ldots,0,0)$
0	...	0	1	$f(0,\ldots,0,1)$
0	...	1	0	$f(0,\ldots,1,0)$
...	...	...	...	...
1	...	1	1	$f(1,\ldots,1,1)$

[1] Джордж Буль (1815–1864).

Если число переменных равно n, то в таблице истинности имеется 2^n строк, соответствующих всем различным комбинациям значений переменных. Следовательно, существует 2^{2^n} различных столбцов, каждый из которых определяет булеву функцию от n переменных. Таким образом, $|P_n| = 2^{2^n}$.

Если нужно задать несколько (например, k) булевых функций от n переменных, то это удобно сделать с помощью одной таблицы, использовав несколько столбцов:

x_1	...	x_n	$f_1(x_1,\ldots,x_n)$	...	$f_k(x_1,\ldots,x_n)$
0	...	0	$f_1(0,\ldots,0)$	...	$f_k(0,\ldots,0)$
...	...	...	...	...	...
1	...	1	$f_1(1,\ldots,1)$	...	$f_k(1,\ldots,1)$

Такая таблица будет иметь 2^n строк, n столбцов для значений переменных и ещё k столбцов для значений функций.

Вообще говоря, значения переменных можно не хранить, если принять соглашение о перечислении наборов переменных в определённом порядке. Во всех таблицах истинности в этом учебнике переменные всегда перечисляются в лексикографическом порядке, а кортежи булевых значений — в порядке возрастания целых чисел, задаваемых кортежами, как двоичными шкалами, что совпадает с лексикографическим порядком на кортежах. Такой порядок мы называем *установленным*.

Если $k > 2^n$ (что, как показывает предыдущая формула, не редкость), то таблицу истинности можно «транспонировать», выписывая наборы значений в столбцах, а значения функций — в строках:

x_1	0	...	1
...	...	...	...
x_n	0	...	1
f_1	$f_1(0,\ldots,0)$	...	$f_1(1,\ldots,1)$
...	...	...	...
f_k	$f_k(0,\ldots,0)$	...	$f_k(1,\ldots,1)$

Именно такой способ записи таблицы истинности использован в пп. 3.1.3 и 3.1.4.

3.1.2. Существенные и несущественные переменные

Булева функция $f \in P_n$ *существенно зависит* от переменной x_i, если существует такой набор значений $a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n$, что

$$f(a_1,\ldots,a_{i-1},0,a_{i+1},\ldots,a_n) \neq f(a_1,\ldots,a_{i-1},1,a_{i+1},\ldots,a_n).$$

В этом случае x_i называют *существенной переменной*, в противном случае x_i называют *несущественной* (*фиктивной*) переменной.

Пример. Пусть булевы функции $f_1(x_1,x_2)$ и $f_2(x_1,x_2)$ заданы таблицей истинности, приведённой ниже. Для этих функций переменная x_1 — существенная, а переменная x_2 — несущественная.

x_1	x_2	f_1	f_2
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0

Пусть заданы две булевы функции, $f_1(x_1, \ldots, x_{n-1})$ и $f_2(x_1, \ldots, x_{n-1}, x_n)$, и пусть переменная x_n — несущественная для функции f_2, а при одинаковых значениях остальных переменных значения функций совпадают:

$$\forall\, a_1, \ldots, a_{n-1}, a_n \;\; (f_1(a_1, \ldots, a_{n-1}) = f_2(a_1, \ldots, a_{n-1}, a_n)) .$$

В таком случае говорят, что f_2 получается из f_1 *введением* несущественной переменной x_n, а f_1 получается из f_2 *удалением* несущественной переменной x_n.

ЗАМЕЧАНИЕ

Процедурно введение и удаление несущественных переменных выполняются достаточно просто. Чтобы ввести несущественную переменную, нужно продублировать каждую строку таблицы истинности, добавить новый столбец и заполнить этот столбец чередующимися значениями 0 и 1 (или 1 и 0, что несущественно). Удаление несущественной переменной выполняется аналогично: нужно отсортировать таблицу истинности так, чтобы она состояла из пар строк, различающихся только в разряде несущественной переменной, после чего удалить столбец несущественной переменной и удалить каждую вторую строку таблицы.

Всюду в дальнейшем булевы функции рассматриваются *с точностью до несущественных переменных*. Это позволяет считать, что все булевы функции (в данной системе функций) зависят от одних и тех же переменных. Для этого в данной системе булевых функций можно сначала удалить все несущественные переменные, а потом добавить несущественные переменные так, чтобы уравнять количество переменных у всех функций.

3.1.3. Булевы функции одной переменной

В следующей таблице собраны все булевы функции одной переменной. Две из них фактически являются константами, поскольку их значения не зависят от значения аргумента.

	Переменная x	0	1	
Название	Обозначение			Несущественные
Нуль	0	0	0	x
Тождественная	x	0	1	
Отрицание	$\neg x$, $\bar{x}$, x', $\sim x$	1	0	
Единица	1	1	1	x

3.1.4. Булевы функции двух переменных

В следующей таблице собраны все булевы функции двух переменных. Из них две являются константами, четыре зависят от одной переменной и только десять существенно зависят от обеих переменных.

	Переменная x	0	0	1	1	
	Переменная y	0	1	0	1	
Название	Обозначение					Несущественные
Нуль	0	0	0	0	0	x, y
Конъюнкция	$\cdot$, $\&$, $\wedge$	0	0	0	1	
		0	0	1	0	
		0	0	1	1	y
		0	1	0	0	
		0	1	0	1	x
Сложение по модулю 2	$+$, $+_2$, $\not\equiv$, $\oplus$, $\triangle$	0	1	1	0	
Дизъюнкция	$\vee$	0	1	1	1	
Стрелка Пирса[1]	$\downarrow$	1	0	0	0	
Эквивалентность	$\equiv$	1	0	0	1	
		1	0	1	0	x
		1	0	1	1	
		1	1	0	0	y
логическое следование						
Импликация	$\rightarrow$, $\Rightarrow$, $\supset$	1	1	0	1	
Штрих Шеффера[2]	$\mid$	1	1	1	0	
Единица	1	1	1	1	1	x, y

ЗАМЕЧАНИЕ

Пустоты в столбцах «Название» и «Обозначение» в предыдущей таблице означают, что булева функция редко используется, а потому не имеет специального названия и обозначения.

3.2. Функции k-значной логики

Рассмотрим множество $E_k = \{0, 1, \ldots, k-1\}$ и множество функций $P_{k,n}$, имеющих n аргументов типа E_k и возвращающих значение типа E_k. Такие функции называются *функциями k-значной логики* и являются обобщением функций алгебры логики: $P_{2,n} = P_n$. Функции k-значной логики можно задавать таблицами, подобными таблицам истинности для обычной двузначной логики, причем в таблицах используются элементы из множества E_k, а не из множества E_2. Нетрудно видеть, что $|P_{k,n}| = k^{k^n}$. Число функций k-значной логики растет ещё быстрее, чем число булевых функций. Так, уже для трехзначной логики аналог таблицы п. 3.1.4 для всех функций двух переменных содержит $3^9 = 19\,683$ строк и не может быть помещён в книгу.

Функцию k-значной логики можно представить как систему функций обычной двузначной логики следующим образом. Функция $f(x_1, \ldots, x_n) \in P_{k,n}$ представляется системой функций

$$\{g_1(y_{11}, \ldots, y_{1m} \ldots, y_{n1}, \ldots, y_{nm}), \ldots, g_m(y_{11}, \ldots, y_{1m}, \ldots, y_{n1}, \ldots, y_{nm})\},$$

где $m = \lceil \log_2 k \rceil$. Значения функции f и переменных x_i принадлежат множеству E_k, а значения функций g_j и переменных y_{ij} принадлежат множеству E_2. При этом

[1] Чарльз Сандерс Пирс (1839–1914).
[2] Генри М. Шеффер (1882–1964).

если значение переменной x_i имеет двоичный код $a_{i1}\dots a_{im}$, то значение функции $f(x_1,\dots,x_n)$ имеет двоичный код

$$g_1(a_{11},\dots,a_{1m},\dots,a_{n1},\dots,a_{nm})\dots g_m(a_{11},\dots,a_{1m},\dots,a_{n1},\dots,a_{nm}).$$

Другими словами, функция g_j вычисляет j-ю цифру в двоичном представлении числа $f(x_1,\dots,x_n)$.

Таким образом, хотя k-значная логика может быть сведена к двузначной логике и многие полезные утверждения двузначной логики распространяются на случай k-значной логики, но вычислительные процедуры во всех случаях являются существенно более трудоёмкими для функций k-значной логики.

Пример. В трёхзначной логике $E_3=\{0,1,2\}$, $k=3$ и $m=\lceil\log_2 3\rceil=2$. Рассмотрим функцию «трёхзначная конъюнкция», которую можно определить, например, как минимум истинностных значений аргументов: $x_1\ \&\ x_2:=\min(x_1,x_2)$. В таком случае таблица истинности имеет следующий вид.

x_1	$y_{11}y_{12}$	x_2	$y_{21}y_{22}$	$x_1\ \&\ x_2$	g_1g_2
0	00	0	00	0	00
0	00	1	01	0	00
0	00	2	10	0	00
1	01	0	00	0	00
1	01	1	01	1	01
1	01	2	10	1	01
2	10	0	00	0	00
2	10	1	01	1	01
2	10	2	10	2	10

Таким образом, конъюнкцию представляет следующая система функций:
$g_1(y_{11},y_{12},y_{21},y_{22})=y_{11}\ \&\ y_{21}$,
$g_2(y_{11},y_{12},y_{21},y_{22})=y_{12}\ \&\ y_{22}\ \vee\ y_{12}\ \&\ y_{21}\ \vee\ y_{11}\ \&\ y_{22}$.

Функции k-значной логики далее в этом учебнике не рассматриваются.

3.2.1. Формулы

В этом разделе обсуждается целый ряд важных понятий, которые часто считаются самоочевидными, а потому их объяснение опускается. Такими понятиями являются, в частности, само понятие *формулы*, *реализация* функций формулами, *интерпретация* формул для вычисления значений функций, *равносильность* формул, *подстановка* и *замена* в формулах. Между тем программная реализация работы с формулами требует учёта некоторых тонкостей, связанных с данными понятиями, которые целесообразно явно указать и обсудить.

3.2.2. Реализация функций формулами

Начнём обсуждение с привычного понятия формулы.

Вообще говоря, *формула* — это цепочка символов, имеющих заранее оговорённый смысл. Обычно формулы строятся по определённым правилам из обозначений объектов и вспомогательных символов — разделителей. Применительно к рассматриваемому случаю объектами являются переменные и булевы функции, перечисленные в таблицах пп. 3.1.3 и 3.1.4, а разделителями — скобки «(», «)» и запятая «,». Мы

считаем, что обозначения всех объектов заранее определены и синтаксически отличимы друг от друга и от разделителей, а правила составления формул общеизвестны.

ЗАМЕЧАНИЕ

Для обозначения объектов используются как отдельные символы, так и группы символов, в том числе со специальным начертанием: верхние и нижние индексы, наклон шрифта и т. п. Допущение о синтаксической различимости остаётся в силе для всех таких особенностей.

Пример. Операцию сложения вещественных чисел принято обозначать знаком «+», $+\colon \mathbb{R}\times\mathbb{R}\to\mathbb{R}$, а функцию «синус» — словом «sin», которое записывается прямым шрифтом и считается одним символом, $\sin\colon \mathbb{R}\to\mathbb{R}$.

ОТСТУПЛЕНИЕ

В математических текстах часто используется *нелинейная* форма записи формул, при которой формула *не* является цепочкой символов. Примерами могут служить дроби, радикалы, суммы, интегралы. Подобные нелинейные формы записи формул привычны и удобны. Использование нелинейной записи формул не является принципиальным расширением языка формул. Можно ограничиться только линейными цепочками символов, что блестяще подтверждает система TEX, с помощью которой подготовлена эта книга.

Пусть $F=\{f_1,\ldots,f_m\}$ — некоторое множество булевых функций от n переменных. *Формулой* $\mathcal{F}$ над F называется выражение (цепочка символов) вида

$$\mathcal{F}[F]=f(t_1,\ldots,t_n),$$

где $f\in F$ и t_i — либо переменная, либо формула над F. Множество F называется *базисом*, функция f называется *главной* (*внешней*) операцией (функцией), а t_i называются *подформулами*. Если базис F ясен из контекста, то его обозначение опускают. Множество всех формул над базисом F здесь обозначается $\mathfrak{F}[F]$.

ЗАМЕЧАНИЕ

Обычно при записи формул, составленных из элементарных булевых функций, обозначения бинарных булевых функций записываются как знаки операций в инфиксной форме, отрицание записывается как знак унарной операции в префиксной форме, тождественные функции записываются как константы 0 и 1. Кроме того, устанавливается приоритет операций ($\neg$, $\&$, $\vee$, $\to$) и лишние скобки опускаются.

Всякой формуле $\mathcal{F}$ однозначно соответствует некоторая функция f. Это соответствие задается *алгоритмом интерпретации*, который позволяет вычислить значение формулы $\mathcal{F}$ при заданных значениях переменных.

ОТСТУПЛЕНИЕ

Некоторые программистские замечания по поводу процедуры Eval.

1. При программной реализации алгоритма интерпретации формул важно учитывать, что в общем случае результат вычисления значения формулы может быть не определён (**fail**). Это имеет место, если формула построена синтаксически неправильно или если в ней используются функции (операции), способ вычисления которых не задан, то есть они не входят в базис. Таким образом, необходимо либо проверять правильность формулы до начала работы алгоритма интерпретации, либо предусматривать невозможность вычисления значения в самом алгоритме.

Алгоритм 3.1. Интерпретация формул — рекурсивная функция Eval

Алгоритм *интерпретации* формулы.

Вход: формула $\mathcal{F}$, множество F функций базиса, значения переменных $x_1, \ldots, x_n$.

Выход: значение формулы $\mathcal{F}$ на значениях $x_1, \ldots, x_n$ или значение **fail**, если значение формулы не может быть определено.

```
if F = 'x_i' then
  return x_i  // значение переменной задано
end if
if F = 'f(t_1, ..., t_n)' then
  if f ∉ F then
    return fail  // функция не входит в базис
  end if
  for t ∈ {t_1, ..., t_n} do
    y_i := Eval (t_i, F, x_1, ..., x_n)  // значение i-го аргумента
    if y_i = fail then
      return fail  // аргумент не может быть вычислен
    end if
  end for
  return f(y_1, ..., y_n)  // вычисленное значение главной операции
end if
return fail  // это не формула
```

2. Это не единственный возможный алгоритм вычисления значения формулы, более того, он не самый лучший. Для конкретных классов формул, например для некоторых классов формул, реализующих булевы функции, известны более эффективные алгоритмы. (см., например, п. 3.5.3).
3. Сама идея этого алгоритма: «сначала вычисляются значения аргументов, а потом значение функции» — не является догмой.

   Например, можно построить такой алгоритм интерпретации, который вычисляет значения только *некоторых* аргументов, а потом вычисляет значение формулы, в результате чего получается новая *формула*, реализующая функцию, которая зависит от меньшего числа аргументов. Такой алгоритм интерпретации называется *смешанными вычислениями*.
4. Порядок вычисления аргументов *считается* неопределённым, то есть предполагается, что базисные функции не имеют *побочных эффектов*. Если же базисные функции имеют побочные эффекты, то результат вычисления значения функции может зависеть от порядка вычисления значений аргументов. Побочные эффекты могут иметь различные причины. Наиболее частая: использование глобальных переменных. Например, если $f(x) \stackrel{\text{Def}}{=} a := a + 1; \textbf{return } x + a$, $g(x) \stackrel{\text{Def}}{=} a := a * 2; \textbf{return } x * a$, причём переменная a глобальна, то (если $a \neq 6$) $f(2) + g(3) \neq g(3) + f(2)$.

Если формула $\mathcal{F}$ и базис F заданы (причём $\mathcal{F}$ является правильно построенной формулой над базисом F), то процедура Eval$(\mathcal{F}, F, x_1, \ldots, x_n)$ является некоторой булевой функцией f переменных $x_1, \ldots, x_n$. В этом случае говорят, что формула $\mathcal{F}$ *реализует* функцию f и записывают это так: $\operatorname{func}\mathcal{F} = f$.

ЗАМЕЧАНИЕ

Для обозначения реализуемости применяют и другие приёмы. Выбранное обозначение обладает тем достоинством, что согласовано с другими обозначениями в книге.

Зная таблицы истинности для функций базиса, можно вычислить таблицу истинности той функции, которую реализует данная формула.

Примеры

1. $F_1 := (x_1 \wedge x_2) \vee ((x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \wedge x_2))$

x_1	x_2	$x_1 \wedge \overline{x}_2$	$\overline{x}_1 \wedge x_2$	$(x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \wedge x_2)$	$x_1 \wedge x_2$	F_1
0	0	0	0	0	0	0
0	1	0	1	1	0	1
1	0	1	0	1	0	1
1	1	0	0	0	1	1

Таким образом, формула F_1 реализует дизъюнкцию.

2. $F_2 := (x_1 \wedge x_2) \to x_1$

x_1	x_2	$x_1 \wedge x_2$	F_2
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	1

Таким образом, формула F_2 реализует константу 1.

3. $F_3 := ((x_1 \wedge x_2) + x_1) + x_2$

x_1	x_2	$x_1 \wedge x_2$	$(x_1 \wedge x_2) + x_1$	$((x_1 \wedge x_2) + x_1) + x_2$
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	0	1

Таким образом, формула F_3 также реализует дизъюнкцию.

3.2.3. Равносильные формулы

Одна функция может иметь множество реализаций (над данным базисом). Формулы, реализующие одну и ту же функцию, называются *равносильными*:

$$\mathcal{F}_1 = \mathcal{F}_2 \stackrel{\text{Def}}{=} \exists f \ \left(\operatorname{func} \mathcal{F}_1 = f \ \& \ \operatorname{func} \mathcal{F}_2 = f\right).$$

Отношение равносильности формул является эквивалентностью. Имеют место, в частности, следующие равносильности.

1) $a \vee a = a, \quad a \wedge a = a$;
2) $a \vee b = b \vee a, \quad a \wedge b = b \wedge a$;
3) $a \vee (b \vee c) = (a \vee b) \vee c, \quad a \wedge (b \wedge c) = (a \wedge b) \wedge c$;
4) $(a \wedge b) \vee a = a, \quad (a \vee b) \wedge a = a$;
5) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c), \quad a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$;
6) $a \vee 1 = 1, \quad a \wedge 0 = 0$;
7) $a \vee 0 = a, \quad a \wedge 1 = a$;
8) $\neg\neg a = a$;
9) $\neg(a \wedge b) = \neg a \vee \neg b, \quad \neg(a \vee b) = \neg a \wedge \neg b$;
10) $a \vee \neg a = 1, \quad a \wedge \neg a = 0$.

Все указанные равносильности могут быть легко проверены построением соответствующих таблиц истинности. Таким образом, $\langle E_2; \vee, \wedge, \neg \rangle$ — булева алгебра (п. 2.6.6).

ЗАМЕЧАНИЕ

Считаем необходимым повторить здесь с уточнениями замечание из п. 1.2.6. Если некоторая бинарная операция ассоциативна, то в формуле, построенной с применением такой операции, порядок выполнения операций не важен и скобки можно опустить. Другими словами, ассоциативная бинарная операция определяет *групповую операцию*, применимую не только к паре, но и к любому набору аргументов. В этом случае употребляют также термины *кратная операция* и *вариаргументная операция*. Если же операция вдобавок коммутативна, то не только скобки можно опустить, но и аргументы групповой операции можно записывать в любом порядке. Ввиду выполнения равносильностей 2 и 3 можно определить кратные операции дизъюнкции и конъюнкции, для которых используются следующие обозначения:

$$\bigvee_{i=1}^{n} x_i \stackrel{\text{Def}}{=} x_1 \vee \ldots \vee x_n, \qquad \bigwedge_{i=1}^{n} x_i \stackrel{\text{Def}}{=} x_1 \wedge \cdots \wedge x_n \stackrel{\text{Def}}{=} x_1 \ldots x_n.$$

Аналогичными свойствами обладают сложение и умножение чисел, объединение и пересечение множеств.

3.2.4. Подстановка и замена

Если в формулу $\mathcal{F}$ входит переменная x, то это обстоятельство обозначается так: $\mathcal{F}(\ldots x \ldots)$. Соответственно, запись $\mathcal{F}(\ldots \mathcal{G} \ldots)$ обозначает, что в формулу $\mathcal{F}$ входит подформула $\mathcal{G}$.

Вместо подформулы (в частности, вместо переменной) в формулу можно подставить другую формулу (в частности, переменную), в результате чего получится новая правильно построенная формула.

Если подстановка формулы $\mathcal{G}$ производится вместо *всех* вхождений заменяемой переменной x (или подформулы), то результат подстановки обозначается следующим образом: $\mathcal{F}(\ldots x \ldots)[\mathcal{G}/x]$. Если же подстановка производится вместо *некоторых* вхождений (в том числе вместо одного), то результат подстановки не имеет общепринятого обозначения. В этом учебнике принято дублировать знак подстановки и писать между знаками список номеров вхождений, подлежащих замене.

Примеры

1. Замена всех вхождений переменной: $x \vee \neg x[y \wedge z/x] = (y \wedge z) \vee \neg(y \wedge z)$.
2. Замена всех вхождений подформулы: $x \vee y \vee z[\neg x/y \vee z] = x \vee \neg x$.
3. Замена первого вхождения переменной: $x \vee \neg x[y/1/x] = y \vee \neg x$.
4. Замена первого вхождения подформулы: $x \vee y \vee z[\neg x/1/y \vee z] = x \vee \neg x$.

Известны два правила: *правило подстановки* и *правило замены*, которые позволяют преобразовывать формулы с сохранением равносильности.

ТЕОРЕМА 3 (Правило подстановки). *Если в двух равносильных формулах вместо всех вхождений некоторой переменной x подставить одну и ту же формулу, то получатся равносильные формулы:*

$$\forall\, \mathcal{G}\ (\mathcal{F}_1(\ldots x \ldots) = \mathcal{F}_2(\ldots x \ldots) \Longrightarrow \mathcal{F}_1(\ldots x \ldots)[\mathcal{G}/x] = \mathcal{F}_2(\ldots x \ldots)[\mathcal{G}/x])\,.$$

Доказательство. Чтобы доказать равносильность двух формул, нужно показать, что они реализуют одну и ту же функцию. А это можно сделать, если взять произвольный набор значений переменных и убедиться, что значения, полученные при

вычислении формул, совпадают. Рассмотрим произвольный набор значений $a_1, \ldots, a_x, \ldots, a_n$ переменных $x_1, \ldots, x, \ldots, x_n$. Обозначим
$a := \mathrm{Eval}(\mathcal{G}, F, a_1, \ldots, a_x, \ldots, a_n)$. По определению алгоритма интерпретации

$$\mathrm{Eval}(\mathcal{F}_1[\mathcal{G}/x], F, a_1, \ldots, a_x, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_1, F, a_1, \ldots, a, \ldots, a_n)$$

и, аналогично,

$$\mathrm{Eval}(\mathcal{F}_2[\mathcal{G}/x], F, a_1, \ldots, a_x, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_2, F, a_1, \ldots, a, \ldots, a_n).$$

Но $\mathcal{F}_1 = \mathcal{F}_2$ и, значит,

$$\mathrm{Eval}(\mathcal{F}_1, F, a_1, \ldots, a, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_2, F, a_1, \ldots, a, \ldots, a_n),$$

откуда $\mathrm{Eval}(\mathcal{F}_1[\mathcal{G}/x], F, a_1, \ldots, a_x, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}_2[\mathcal{G}/x], F, a_1, \ldots, a_x, \ldots, a_n)$. □

ЗАМЕЧАНИЕ

В правиле подстановки условие замены *всех* вхождений существенно:
например, $x \vee \neg x = 1$ и $x \vee \neg x[y/x] = y \vee \neg y = 1$, но $x \vee \neg x[y/1/x] = y \vee \neg x \neq 1$.

ТЕОРЕМА 4 (Правило замены). *Если в формуле заменить некоторую подформулу равносильной формулой, то получится формула, равносильная исходной:*

$$\forall\, \mathcal{F}(\ldots \mathcal{G}_1 \ldots)\ (\mathcal{G}_1 = \mathcal{G}_2 \Longrightarrow \mathcal{F}(\ldots \mathcal{G}_1 \ldots) = \mathcal{F}(\ldots \mathcal{G}_1 \ldots)[\mathcal{G}_2//\mathcal{G}_1])\,.$$

Доказательство. Рассмотрим произвольный набор значений $a_1, \ldots, a_n$ переменных $x_1, \ldots, x_n$. Имеем $\mathcal{G}_1 = \mathcal{G}_2$ и, значит,

$$\mathrm{Eval}(\mathcal{G}_1, F, a_1, \ldots, a_n) = \mathrm{Eval}(\mathcal{G}_2, F, a_1, \ldots, a_n),$$

откуда $\mathrm{Eval}(\mathcal{F}[\mathcal{G}_1//\mathcal{G}_1], F, a_1, \ldots, a_n) = \mathrm{Eval}(\mathcal{F}[\mathcal{G}_2//\mathcal{G}_1], F, a_1, \ldots, a_n)$. □

Пусть $F = \{f_1, \ldots, f_m\}$ и $G = \{g_1, \ldots, g_m\}$. Тогда говорят, что формулы $\mathcal{F}[F]$ и $\mathcal{G}[G]$ имеют *одинаковое строение*, если $\mathcal{F}$ совпадает с результатами подстановки в формулу $\mathcal{G}$ функций f_i вместо функций g_i: $\mathcal{F}[F] = \mathcal{G}[G][f_i/g_i]_{i=1}^m$.

3.2.5. Алгебра булевых функций

Булевы функции $\vee$, $\wedge$, $\neg$ (и любые другие) могут рассматриваться как операции на множестве булевых функций, $\vee, \wedge\colon P_n \times P_n \to P_n$, $\neg\colon P_n \to P_n$. Действительно, пусть формулы $\mathcal{F}_1$ и $\mathcal{F}_2$ равносильны и реализуют функцию f, а формулы $\mathcal{G}_1$ и $\mathcal{G}_2$ равносильны и реализуют функцию g: $\mathrm{func}\,\mathcal{F}_1 = f$, $\mathrm{func}\,\mathcal{F}_2 = f$, $\mathrm{func}\,\mathcal{G}_1 = g$, $\mathrm{func}\,\mathcal{G}_2 = g$. Тогда, применяя правило замены нужное число раз, имеем
$\mathcal{F}_1 \vee \mathcal{G}_1 = \mathcal{F}_2 \vee \mathcal{G}_2$, $\mathcal{F}_1 \wedge \mathcal{G}_1 = \mathcal{F}_2 \wedge \mathcal{G}_2$, $\neg\mathcal{F}_1 = \neg\mathcal{F}_2$. Таким образом, если взять любые формулы $\mathcal{F}$ и $\mathcal{G}$, реализующие функции f и g соответственно, то каждая из формул $\mathcal{F} \wedge \mathcal{G}$, $\mathcal{F} \vee \mathcal{G}$ и $\neg\mathcal{F}$ реализует одну и ту же функцию, независимо от выбора реализующих формул $\mathcal{F}$ и $\mathcal{G}$. Следовательно, функции, которые реализуются соответствующими формулами, можно по определению считать результатами применения соответствующих операций. Другими словами, если $\mathrm{func}\,\mathcal{F} = f$, $\mathrm{func}\,\mathcal{G} = g$, то $f \wedge g \stackrel{\mathrm{Def}}{=} \mathrm{func}(\mathcal{F} \wedge \mathcal{G})$, $f \vee g \stackrel{\mathrm{Def}}{=} \mathrm{func}(\mathcal{F} \vee \mathcal{G})$, $\neg f \stackrel{\mathrm{Def}}{=} \mathrm{func}(\neg\mathcal{F})$. Алгебраическая структура $\langle P_n; \vee, \wedge, \neg\rangle$ называется *алгеброй булевых функций*.

ТЕОРЕМА. *Алгебра булевых функций является булевой алгеброй.*

Доказательство. Действительно, равносильности, перечисленные в п. 3.2.2, могут быть проверены путем построения таблиц истинности. Ясно, что эти таблицы не зависят от того, откуда взялись значения a, b, c. Таким образом, вместо a, b, c можно подставить любые функции, а значит, любые реализующие их формулы, если только выполнено правило подстановки. Следовательно, аксиомы булевой алгебры выполнены в алгебре $\langle P_n; \vee, \wedge, \neg\rangle$. □

Пусть $[\mathcal{F}]$ — множество формул, равносильных $\mathcal{F}$ (то есть класс эквивалентности по отношению равносильности). Рассмотрим множество $\mathcal{K}$ классов эквивалентности по отношению равносильности. Пусть операции $\vee, \wedge\colon \mathcal{K}\times\mathcal{K}\to\mathcal{K}$, $\neg\colon \mathcal{K}\to\mathcal{K}$ определены следующим образом:

$$[\mathcal{F}_1]\vee[\mathcal{F}_2]\stackrel{\text{Def}}{=}[\mathcal{F}_1\vee\mathcal{F}_2],\ [\mathcal{F}_1]\wedge[\mathcal{F}_2]\stackrel{\text{Def}}{=}[\mathcal{F}_1\wedge\mathcal{F}_2],\ \neg[\mathcal{F}_1]\stackrel{\text{Def}}{=}[\neg\mathcal{F}_1].$$

Тогда функция func, сопоставляющая формуле булевскую функцию, которую она реализует, является гомоморфизмом относительно операций $\wedge, \vee, \neg$, и к ней можно применить теорему о гомоморфизме (п. 1.7.6). В этом случае $\operatorname{Im}\text{func}\subset P_n$ и $\operatorname{Dom}\text{func}=\mathfrak{F}[\vee,\wedge,\neg]$.

ЗАМЕЧАНИЕ

В пп. 3.4.3 и 3.6.4 показано, что $\text{func}(\mathfrak{F}[\vee,\wedge,\neg])=P_n$, то есть достаточно рассматривать только формулы, построенные над базисом $\{\vee,\wedge,\neg\}$.

Применение теоремы о гомоморфизме гласит $\mathfrak{F}[\vee,\wedge,\neg]/\ker\text{func}\sim\langle P_n;\vee,\wedge,\neg\rangle$. Другими словами, алгебра классов равносильных формул (*алгебра Линденбаума–Тарского*) изоморфна алгебре булевых функций и, следовательно, является булевой алгеброй. Носитель этой алгебры — множество классов формул.

ОТСТУПЛЕНИЕ

На практике мы говорим о функциях, а пишем формулы, хотя формулы и функции — разные вещи. Например, формул бесконечно много, а функций (булевых функций n переменных) — только конечное число, и свободная алгебра формул *не изоморфна* алгебре функций. Но алгебра функций *изоморфна* алгебре классов равносильных формул, что позволяет манипулировать формулами, имея в виду функции.

3.3. Двойственность и симметрия

Понятие двойственности и симметрии с успехом применяется в самых разных областях математики. Мы рассматриваем двойственность и симметрию на простейшем примере булевых функций.

3.3.1. Двойственные и самодвойственные функции

Пусть $f(x_1,\dots,x_n)\in P_n$ — булева функция. Тогда функция $f^*(x_1,\dots,x_n)$, определённая следующим образом: $f^*(x_1,\dots,x_n)\stackrel{\text{Def}}{=}\overline{f(\overline{x}_1,\dots,\overline{x}_n)}$, называется *двойственной* к функции f.

Из определения легко видно, что двойственность инволютивна: $f^{**}=f$, и по этой причине отношение «быть двойственной к» на множестве булевых функций симметрично, то есть если $f^*=g$, то $g^*=f$.

Если в таблице истинности булевой функции f инвертировать *все* значения, то получим таблицу истинности двойственной функции f^*.

Пример.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

$\mapsto$

x_1	x_2	$(x_1 \wedge x_2)^*$
1	1	1
1	0	1
0	1	1
0	0	0

$=$

x_1	x_2	$(x_1 \wedge x_2)^*$
0	0	0
0	1	1
1	0	1
1	1	1

Таким способом можно определить двойственную функцию для любой булевой функции.

ЗАМЕЧАНИЕ

Если функции двойственны, то их отрицания также двойственны.

Пример. В таблице ниже приведены все пары двойственных функций двух переменных. Для каждой функции указан вектор значений при установленном порядке перечисления кортежей и обозначение, если оно введено в таблице п. 3.1.4.

f	0000	0001	0010	0011	0100	0101	0110	1000	1010	1100
	0	$x \,\&\, y$		x		y	$x +_2 y$	$x \downarrow y$	$\neg x$	$\neg y$
	1	$x \vee y$		x		y	$x \equiv y$	$x \mid y$	$\neg x$	$\neg y$
f^*	1111	0111	1011	0011	1101	0101	1001	1110	1010	1100

Функция называется *самодвойственной*, если $f^* = f$.

Пример. Тождественная функция и отрицание самодвойственны, все функции двух существенных переменных, а также константы не самодвойственны.

ЗАМЕЧАНИЕ

Если функция самодвойственна, то ее отрицание также самодвойственно.

ТЕОРЕМА. *Существует* $2^{2^{n-1}}$ *самодвойственных функций* n *переменных.*

Доказательство. Из алгоритма построения двойственной функции следует, что функция самодвойственна тогда и только тогда, когда вектор значений функции в таблице истинности является зеркально инверсным относительно своей середины. Получается, что первая половина вектора значений самодвойственной функции однозначно задает вторую. □

Пример. Функция «голосования» $m(x, y, z)$ (эту функцию называют также *мажоритарной*) принимает значение 1, если большинство переменных принимают значение 1. Из приведённой таблицы истинности функции голосования видно, что она самодвойственна.

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
m	0	0	0	1	0	1	1	1

3.3.2. Реализация двойственной функции

Формула, реализующая двойственную функцию, определённым образом связана с формулой, реализующей исходную функцию.

ЗАМЕЧАНИЕ

Далее используется обозначение $\overline{f}(\cdots) \stackrel{\text{Def}}{=} \overline{f(\cdots)}$.

ТЕОРЕМА. *Если функция $\varphi(x_1,\ldots,x_n)$ реализована формулой $f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n))$, то формула $f^*(f_1^*(x_1,\ldots,x_n),\ldots,f_n^*(x_1,\ldots,x_n))$ реализует функцию $\varphi^*(x_1,\ldots,x_n)$.*

Доказательство.

$$\varphi^*(x_1,\ldots,x_n) = \overline{\varphi}(\overline{x}_1,\ldots,\overline{x}_n) = \overline{f}(f_1(\overline{x}_1,\ldots,\overline{x}_n),\ldots,f_n(\overline{x}_1,\ldots,\overline{x}_n)) =$$
$$= \overline{f}(\overline{f_1}(\overline{x}_1,\ldots,\overline{x}_n),\ldots,(\overline{f_n}(\overline{x}_1,\ldots,\overline{x}_n)) = \overline{f}(\overline{f_1^*}(x_1,\ldots,x_n),\ldots,\overline{f_n^*}(x_1,\ldots,x_n)) =$$
$$= f^*(f_1^*(x_1,\ldots,x_n),\ldots,f_n^*(x_1,\ldots,x_n)).$$ □

3.3.3. Принцип двойственности

Принцип двойственности устанавливает связь между структурами формул, реализующих пару двойственных функций. Рассмотрим две системы булевых функций, $F=\{f_1,\ldots,f_m\}$ и $F^*=\{f_1^*,\ldots,f_m^*\}$, и введём обозначение

$$\mathcal{F}^*[F^*] \stackrel{\text{Def}}{=} \mathcal{F}[F][f_i^*/f_i]_{i=1}^m.$$

ТЕОРЕМА (Принцип двойственности)**.** *Пусть $F=\{f_1,\ldots,f_m\}$ — система булевых функций, а $F^*=\{f_1^*,\ldots,f_m^*\}$ — система двойственных функций. Тогда если формула $\mathcal{F}$ над базисом F реализует функцию f, то формула $\mathcal{F}^*$ над базисом F^*, полученная заменой функций f_i двойственными функциями ${f_i}^*$, реализует функцию f^*:*

$$\operatorname{func}\mathcal{F}[F]=f \Longrightarrow \operatorname{func}\mathcal{F}^*[F^*]=f^*.$$

Доказательство. Индукция по структуре формулы $\mathcal{F}$. База: если формула $\mathcal{F}$ имеет вид $f(x_1,\ldots,x_n)$, где $f\in F$, то формула $\mathcal{F}^*=f^*(x_1,\ldots,x_n)$ реализует функцию f^* по определению. Индукционный переход по теореме п. 3.3.2. □

СЛЕДСТВИЕ. $\mathcal{F}_1=\mathcal{F}_2 \Longrightarrow {\mathcal{F}_1}^*={\mathcal{F}_2}^*$.

Пример. Из $\overline{x_1\wedge x_2}=\overline{x}_1\vee\overline{x}_2$ по принципу двойственности имеем $\overline{x_1\vee x_2}=\overline{x}_1\wedge\overline{x}_2$.

ОТСТУПЛЕНИЕ

Хорошо известен принцип математической индукции для натуральных чисел:

$$(P(1)\ \&\,(P(n)\Longrightarrow P(n+1)))\Longrightarrow \forall\, n\in\mathbb{N}\ \ (P(n))\,.$$

Этот принцип является справедливым и для других множеств, упорядоченных более сложным образом, нежели натуральные числа (п. 1.8.9). Например, в доказательстве предыдущей теоремы был использован принцип индукции в следующей форме.

Пусть задана некоторая иерархия. Предположим, что: 1) некоторое утверждение P справедливо для всех узлов иерархии нижнего уровня; 2) из того, что утверждение P справедливо для всех узлов, подчиненных данному узлу, следует, что утверждение P справедливо для данного узла. Тогда утверждение P справедливо для всех узлов иерархии.

3.3.4. Симметрические функции

Булева функция n переменных называется *симметрической*, если её значение инвариантно относительно перестановок значений аргументов. Заметим, что при перестановке значений в наборе общее количество нулей и единиц сохраняется. Понятие симметрической функции применимо только к функциям двух и большего числа переменных.

Примеры

Нуль, конъюнкция, сложение по модулю 2, дизъюнкция, стрелка Пирса, эквивалентность, штрих Шеффера и единица являются симметрическими функциями двух переменных. Других симметрических функций двух переменных нет.

ТЕОРЕМА. *Существует* 2^{n+1} *симметрических функций* n *переменных.*

Доказательство. Ясно, что на любом наборе значений n переменных, в котором содержится k значений 1 (и соответственно $n-k$ значений 0), симметрическая функция принимает одно и то же значение (1 или 0). Таким образом, симметрическую функцию f от n переменных можно задать булевским вектором $(f_0, f_1, \ldots, f_n)$, где f_i — значение функции на наборах значений, содержащих i единиц. Всего существует 2^{n+1} таких наборов. □

Примеры

Функция голосования является симметрической.
Формула $\neg x \wedge y \wedge z \vee x \wedge \neg y \wedge z \vee x \wedge y \wedge \neg z$ реализует функцию, которая также является симметрической.

3.4. Нормальные формы

В данном разделе на примере булевых функций обсуждается важное понятие «нормальная форма», то есть синтаксически однозначный способ записи формулы, реализующей заданную функцию.

3.4.1. Разложение булевых функций по переменным

Положим $x^y \stackrel{\text{Def}}{=} x \wedge y \vee \overline{x} \wedge \overline{y}$. Очевидно, что
$x^y = x \equiv y =$ **if** y **then** x **else** $\overline{x}$ **end if** $=$ **if** $x = y$ **then** 1 **else** 0 **end if**.

ТЕОРЕМА (О разложении булевой функции по переменным)**.**

$$f(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n) = \bigvee_{(\sigma_1, \ldots, \sigma_m)} x_1^{\sigma_1} \wedge \ldots \wedge x_m^{\sigma_m} \wedge f(\sigma_1, \ldots, \sigma_m, x_{m+1}, \ldots, x_n),$$

где дизъюнкция берётся по всем возможным наборам значений $(\sigma_1, \ldots, \sigma_m)$.

Доказательство. Рассмотрим значение формулы в правой части на наборе значений $a_1, \ldots, a_n$. Имеем

$$\left(\bigvee_{(\sigma_1, \ldots, \sigma_m)} x_1^{\sigma_1} \wedge \ldots \wedge x_m^{\sigma_m} \wedge f(\sigma_1, \ldots, \sigma_m, x_{m+1}, \ldots, x_n) \right)(a_1, \ldots, a_n) =$$
$$= \bigvee_{(\sigma_1, \ldots, \sigma_m)} a_1^{\sigma_1} \wedge \ldots \wedge a_m^{\sigma_m} \wedge f(\sigma_1, \ldots, \sigma_m, a_{m+1}, \ldots, a_n).$$ Все конъюнкции, в которых

$\exists i\ a_i \neq \sigma_i$, равны 0, и их можно опустить, поэтому остаётся только одно слагаемое, для которого $\forall\, i \in 1..n\ \ (a_i = \sigma_i)$, значит $a_1^{a_1} \wedge \ldots \wedge a_n^{a_n} \wedge f(a_1, \ldots, a_m, a_{m+1}, \ldots, a_n) = f(a_1, \ldots, a_n)$. □

ЗАМЕЧАНИЕ

Здесь доказывается, что некоторая формула реализует заданную функцию. Для этого достаточно взять *произвольный* набор значений аргументов функции, вычислить на этом наборе значение формулы, и если оно окажется равным значению функции на этом наборе аргументов, то из этого следует доказываемое утверждение.

СЛЕДСТВИЕ 1 (Разложение булевой функции по одной переменной)**.**

$$f(x_1, \ldots, x_{n-1}, x_n) = (x_n \wedge f(x_1, \ldots, x_{n-1}, 1)) \vee (\overline{x}_n \wedge f(x_1, \ldots, x_{n-1}, 0)).$$

Пример. Разложение функции голосования по переменной x: $m(x, y, z) = (x \wedge m(1, y, z)) \vee (\overline{x} \wedge m(0, y, z)) = (x \wedge (y \vee z)) \vee (\overline{x} \wedge (y \wedge z))$.

СЛЕДСТВИЕ 2 (Разложение булевой функции по всем переменным)**.**

$$f(x_1, \ldots, x_n) = \bigvee_{\{(\sigma_1, \ldots, \sigma_n) \mid f(\sigma_1, \ldots, \sigma_n) = 1\}} x_1^{\sigma_1} \wedge \ldots \wedge x_n^{\sigma_n}.$$

3.4.2. Минимальные термы

Выражение вида $x_1^{\sigma_1} \wedge \cdots \wedge x_n^{\sigma_n}$, которое присутствует в следствии 2 теоремы о разложении булевой функции по переменным, имеет большое значение в теории булевых функций и носит специальное название: *минимальный терм*, или *совершенный одночлен*, или *конституента единицы*, или, наиболее коротко, *минтерм*. Нетрудно видеть, что минтерм — это реализация булевой функции n переменных, которая имеет значение 1 ровно на одном наборе значений переменных, а именно на наборе $(\sigma_1, \ldots, \sigma_n)$. Отсюда и из следствия 2 вытекает еще одно следствие к теореме о разложении булевой функции по переменным.

СЛЕДСТВИЕ 3. *Любую булеву функцию, кроме 0, можно представить как дизъюнкцию некоторых минтермов.*

Ясно, что при фиксированном n имеется ровно 2^n различных минтермов. Обычно минтерм обозначают m_i, где $i \in 0..2^{n-1}$, i называется номером минтерма, или прямо указывают булевский вектор $(\sigma_1, \ldots, \sigma_n)$, на котором минтерм принимает значение 1. Из структуры формулы, задающей минтерм, непосредственно следует, что двоичное представление номера минтерма задает тот набор значений (в установленном порядке), на котором минтерм принимает значение 1.

Говорят, что система булевых функций *ортогональна*, если их конъюнкция есть тождественный ноль.

Пример. Система $\{x, \overline{x}\}$ ортогональна.

ЛЕММА 1. *Любое множество из двух или более различных минтермов ортогонально.*

Доказательство. Не существует такого набора значений, на котором два или более различных минтерма принимают значение 1. □

Говорят, что система булевых функций *выполнима*, если их дизъюнкция не есть тождественный ноль.

Пример. Система $\{x, \overline{x}\}$ выполнима.

ЛЕММА 2. *Любое множество минтермов выполнимо.*

Доказательство. Дизъюнкция выполняется на всех наборах значений, на которых какой-либо из минтермов принимает значение 1. □

Двойственным к минтерму является *макстерм* — булева функция, которая принимает значение 0 на единственном наборе значений переменных. Ясно, что макстерм реализуется формулой $x_1^{\sigma_1} \vee \ldots \vee x_n^{\sigma_n}$. Используя принцип двойственности, на макстермы нетрудно распространить все наблюдения, которые сделаны для минтермов.

3.4.3. Совершенные нормальные формы

Говорят, что некоторый класс формул $\mathcal{K}$ *имеет нормальную форму*, если задан другой класс формул $\mathcal{K}'$, которые называются нормальными формами, такой, что любая формула класса $\mathcal{K}$ имеет *единственную* равносильную формулу из класса $\mathcal{K}'$. Если задан алгоритм, позволяющий для любой формулы построить её нормальную форму, то наличие у класса формул нормальной формы обеспечивает *разрешимость*, то есть наличие алгоритма проверки равносильности. Действительно, в этом случае достаточно сравнить нормальные формы двух формул. Один и тот же класс $\mathcal{K}$ может иметь несколько различных нормальных форм, то есть несколько различных классов $\mathcal{K}'$ (пп. 2.2.3 и 2.2.4).

Рассмотрим понятие нормальной формы применительно к булевым функциям. Реализация булевой функции $f(x_1, \ldots, x_n)$ в виде формулы

$$\bigvee_{\{(\sigma_1,\ldots,\sigma_n)|f(\sigma_1,\ldots,\sigma_n)=1\}} x_1^{\sigma_1} \wedge \ldots \wedge x_n^{\sigma_n}$$

называется *совершенной дизъюнктивной нормальной формой* (СДНФ).

ЗАМЕЧАНИЕ

СДНФ называется совершенной, потому что каждое слагаемое в дизъюнкции включает все переменные; дизъюнктивной, потому что главная операция — дизъюнкция; нормальной, потому что СДНФ единственна, как показано в следующей теореме.

ТЕОРЕМА 1. *Всякая булева функция имеет единственную СДНФ.*

Доказательство. По следствию 2 к теореме о разложении булевой функции по переменным имеем $f(x_1, \ldots, x_n) = \bigvee_{\{(\sigma_1,\ldots,\sigma_n)|f(\sigma_1,\ldots,\sigma_n)=1\}} x_1^{\sigma_1} \wedge \ldots \wedge x_n^{\sigma_n}$. Если при записи СДНФ используется установленный порядок, то СДНФ однозначно определяет множество наборов значений переменных, на которых функция, реализуемая

СДНФ, принимает значение 1. Тем самым СДНФ однозначно определяет таблицу истинности реализуемой функции, а значит, любые две различные СДНФ над одним и тем же набором переменных неравносильны. □

Примеры

1. СДНФ конъюнкции $x \wedge y = x^1 \wedge y^1$.
2. СДНФ дизъюнкции $x \vee y = x^0 \wedge y^1 \vee x^1 \wedge y^0 \vee x^1 \wedge y^1$.
3. СДНФ импликации $x \to y = x^0 \wedge y^0 \vee x^0 \wedge y^1 \vee x^1 \wedge y^1$.

ЗАМЕЧАНИЕ

При рассмотрении дизъюнктивных (и конъюнктивных) форм мы имеем дело с формулами вида $\bigvee\limits_{i \in I} S_i$ и $\bigwedge\limits_{i \in I} S_i$, где I — некоторое множество индексов, а S_i — некоторые формулы. Множество индексов может быть пусто. В этом случае удобно считать, что пустая дизъюнкция имеет значение 0, а пустая конъюнкция — значение 1:

$$I = \varnothing \Longrightarrow \bigvee_{i \in I} S_i = 0, \quad I = \varnothing \Longrightarrow \bigwedge_{i \in I} S_i = 1.$$

СЛЕДСТВИЕ. *Всякая булева функция может быть выражена через дизъюнкцию, конъюнкцию и отрицание:* $\forall\, f \in P_n \ \left(\exists\, \mathcal{F} \in \mathfrak{F}[\vee, \wedge, \neg] \ \left(f = \operatorname{func} \mathcal{F}\right)\right)$.

ЗАМЕЧАНИЕ

Если $f = 0$, то $\{(\sigma_1, \ldots, \sigma_n) \mid f(\sigma_1, \ldots, \sigma_n) = 1\} = \varnothing$. В соответствии с соглашением предыдущего замечания пустая формула считается СДНФ нуля.

ТЕОРЕМА 2. *Всякая булева функция имеет единственную совершенную конъюнктивную нормальную форму (СКНФ):*

$$f(x_1, \ldots, x_n) = \bigwedge_{\{(\sigma_1, \ldots, \sigma_n) \mid f^*(\sigma_1, \ldots, \sigma_n) = 1\}} x_1^{\sigma_1} \vee \ldots \vee x_n^{\sigma_n}.$$

Доказательство. По принципу двойственности из предыдущей теоремы. □

Примеры

Нормальные формы существуют у различных формул.

1. СДНФ и СКНФ являются нормальными формами для булевых формул над любым базисом.
2. Формулы вида $\sum_{i=0}^{n} a_i x^i$ и $(\ldots(a_n x + a_{n-1})x + \ldots + a_1)x + a_0$ являются нормальными формами для полиномов одной переменной степени n.
3. Множество формул, построенных суперпозицией из рациональных функций одной переменной, e^x и $\ln x$ (то есть множество формул, реализующих элементарные функции математического анализа), нормальной формы не имеет. Доказательство последнего утверждения выходит за рамки этого учебника.

3.4.4. Эквивалентные преобразования

Используя уже доказанные равносильности, можно преобразовывать по правилу замены одни формулы в другие, равносильные им. Преобразование формулы в равносильную ей называется *эквивалентным преобразованием*.

Пример. Используя равносильности из п. 3.2.2, покажем, что имеет место *правило склеивания/расщепления*: $(x \,\&\, y) \vee (x \,\&\, \neg y) = x$. Действительно,
$(x \,\&\, y) \vee (x \,\&\, \neg y) \overset{5}{=} x \,\&\, (y \vee \neg y) \overset{10}{=} x \,\&\, 1 \overset{7}{=} x$.

ЗАМЕЧАНИЕ

Если равносильность из предыдущего примера применяется для уменьшения числа операций в формуле, то говорят, что производится *склеивание*, а если наоборот, то *расщепление*.

ТЕОРЕМА. *Для любых двух равносильных формул $\mathcal{F}_1$ и $\mathcal{F}_2$ существует последовательность эквивалентных преобразований из $\mathcal{F}_1$ в $\mathcal{F}_2$, получаемая посредством равносильностей, указанных в п. 3.2.2.*

Доказательство. Любую формулу (кроме тех, которые реализуют 0) можно преобразовать в СДНФ с помощью равносильностей из п. 3.2.2 и правила расщепления из предыдущего примера по следующему алгоритму.

1. *Элиминация операций.* Любая булева операция реализуется формулой над базисом $\{\wedge, \vee, \neg\}$ (например, в виде СДНФ). Таким образом, любая присутствующая в формуле подформула с главной операцией, отличной от дизъюнкции, конъюнкции и отрицания, может быть заменена подформулой, содержащей только три базисные операции. Например, элиминация импликации выполняется с помощью равносильности $x_1 \to x_2 = \neg x_1 \vee x_2$. В результате первого шага в формуле остаются только базисные операции.
2. *Протаскивание отрицаний.* С помощью инволютивности отрицания и правил де Моргана операция отрицания «протаскивается» к переменным. В результате второго шага отрицания могут присутствовать в формуле только непосредственно перед переменными.
3. *Раскрытие скобок.* По дистрибутивности конъюнкции относительно дизъюнкции раскрываются все скобки, являющиеся операндами конъюнкции. В результате третьего шага формула приобретает вид *дизъюнктивной формы*: $\bigvee(A_i \wedge \cdots \wedge A_j)$, где A_k — это либо переменная, либо отрицание переменной.
4. *Удаление нулей.* Если в слагаемое дизъюнктивной формы входят переменная и её отрицание ($x \,\&\, \neg x$), то такое слагаемое удаляется. Если при этом формула оказывается пустой, то процесс прерывается и считается завершённым (исходная формула реализует 0).
5. *Расщепление переменных.* По правилу расщепления в каждую конъюнкцию, которая содержит не все переменные, добавляются недостающие. В результате шестого шага формула становится «совершенной», то есть в каждой конъюнкции содержатся все переменные.
6. *Приведение подобных.* С помощью идемпотентности конъюнкции удаляются повторные вхождения переменных в каждую конъюнкцию, а затем с помощью идемпотентности дизъюнкции удаляются повторные вхождения одинаковых конъюнкций в дизъюнкцию. В результате пятого шага формула не содержит «лишних» переменных и «лишних» конъюнктивных слагаемых.

7. *Сортировка*. С помощью коммутативности переменные в каждой конъюнкции, а затем конъюнкции в дизъюнкции сортируются в установленном порядке (п. 3.1.1). В результате седьмого шага формула приобретает вид СДНФ.

Заметим, что все указанные преобразования обратимы. Если теперь даны две формулы, преобразуем их в СДНФ указанным алгоритмом. Если результаты не совпали, значит, формулы не равносильны и эквивалентное преобразование одной в другую невозможно. Если же результаты совпали, то, применяя обратные преобразования в обратном порядке, преобразуем полученную СДНФ во вторую формулу. Объединяя последовательность преобразований первой формулы в СДНФ и обратных преобразований СДНФ во вторую формулу, имеем искомую последовательность преобразований. □

Пример. Покажем, что формулы $\neg((y \vee \neg z) \to (\neg y \,\&\, \neg z)) \to x$ и $(x \vee y) \to ((\neg x \to y) \,\&\, \neg(\neg x \,\&\, y))$ равносильны. Для этого приведем первую формулу к СДНФ, используя шаги 1–7 из доказательства теоремы.

1. Элиминация импликации: $\neg\neg(\neg(y \vee \neg z) \vee (\neg y \,\&\, \neg z)) \vee x$.
2. Протаскивание отрицаний: $((\neg y \,\&\, z) \vee (\neg y \,\&\, \neg z)) \vee x$.
3. Раскрытие скобок: $(\neg y \,\&\, z) \vee (\neg y \,\&\, \neg z) \vee x$.
4. Удаление нулей: нет.
5. Расщепление переменных: $(\neg y \,\&\, z \,\&\, x) \vee (\neg y \,\&\, z \,\&\, \neg x) \vee (\neg y \,\&\, \neg z \,\&\, x) \vee$ $\vee\, (\neg y \,\&\, \neg z \,\&\, \neg x) \vee (x \,\&\, \neg y \,\&\, \neg z) \vee (x \,\&\, \neg y \,\&\, z) \vee (x \,\&\, y \,\&\, \neg z) \vee (x \,\&\, y \,\&\, z)$.
6. Приведение подобных: $(\neg y \,\&\, z \,\&\, x) \vee (\neg y \,\&\, z \,\&\, \neg x) \vee (\neg y \,\&\, \neg z \,\&\, x) \vee$ $\vee\, (\neg y \,\&\, \neg z \,\&\, \neg x) \vee (x \,\&\, y \,\&\, \neg z) \vee (x \,\&\, y \,\&\, z)$.
7. Сортировка: $(\neg x \,\&\, \neg y \,\&\, \neg z) \vee (\neg x \,\&\, \neg y \,\&\, z) \vee (x \,\&\, \neg y \,\&\, \neg z) \vee (x \,\&\, \neg y \,\&\, z) \vee$ $(x \,\&\, y \,\&\, \neg z) \vee (x \,\&\, y \,\&\, z)$.

Аналогично обрабатывается вторая формула $(x \vee y) \to ((\neg x \to y) \,\&\, \neg(\neg x \,\&\, y))$.

1. Элиминация импликации: $\neg(x \vee y) \vee ((\neg\neg x \vee y) \,\&\, \neg(\neg x \,\&\, y))$.
2. Протаскивание отрицаний: $(\neg x \,\&\, \neg y) \vee ((x \vee y) \,\&\, (x \vee \neg y))$.
3. Раскрытие скобок: $(\neg x \,\&\, \neg y) \vee (x \,\&\, x) \vee (x \,\&\, \neg y) \vee (y \,\&\, x) \vee (y \,\&\, \neg y)$.
4. Удаления нулей: $(\neg x \,\&\, \neg y) \vee (x \,\&\, x) \vee (x \,\&\, \neg y) \vee (y \,\&\, x)$.
5. Расщепление переменных: $(\neg x \,\&\, \neg y \,\&\, \neg z) \vee (\neg x \,\&\, \neg y \,\&\, z) \vee$ $\vee\, (x \,\&\, x \,\&\, \neg y \,\&\, \neg z) \vee (x \,\&\, x \,\&\, \neg y \,\&\, z) \vee (x \,\&\, x \,\&\, y \,\&\, \neg z) \vee (x \,\&\, x \,\&\, y \,\&\, z) \vee$ $\vee\, (x \,\&\, \neg y \,\&\, \neg z) \vee (x \,\&\, \neg y \,\&\, z) \vee (y \,\&\, x \,\&\, \neg z) \vee (y \,\&\, x \,\&\, z)$.
6. Приведение подобных: $(\neg x \,\&\, \neg y \,\&\, \neg z) \vee (\neg x \,\&\, \neg y \,\&\, z) \vee (x \,\&\, \neg y \,\&\, \neg z) \vee$ $\vee\, (x \,\&\, \neg y \,\&\, z) \vee (x \,\&\, y \,\&\, \neg z) \vee (x \,\&\, y \,\&\, z)$.
7. Сортировка: нет.

Таким образом, формулы равносильны.

Приведенный пример показывает, что хотя построение последовательности эквивалентных преобразований за счет приведения формул к нормальной форме всегда возможно, это не всегда рационально.

Пример. В условиях предыдущего примера $\neg((y \vee \neg z) \to (\neg y \,\&\, \neg z)) \to x$ и $(x \vee y) \to ((\neg x \to y) \,\&\, \neg(\neg x \,\&\, y))$ можно провести следующие эквивалентные преобразования:

$\neg((y \vee \neg z) \to (\neg y \,\&\, \neg z)) \to x = \neg\neg(\neg(y \vee \neg z) \vee (\neg y \,\&\, \neg z)) \vee x =$
$= ((\neg y \,\&\, z) \vee (\neg y \,\&\, \neg z)) \vee x = (\neg y \vee (z \,\&\, \neg z)) \vee x = \neg y \vee x.$

С другой стороны, $(x \vee y) \to ((\neg x \to y) \,\&\, \neg(\neg x \,\&\, y)) =$
$= \neg(x \vee y) \vee ((\neg\neg x \vee y) \,\&\, \neg(\neg x \,\&\, y)) = (\neg x \,\&\, \neg y) \vee (x \,\&\, (y \vee \neg y)) =$
$= (\neg x \,\&\, \neg y) \vee x = (\neg x \vee x) \,\&\, (\neg y \vee x) = \neg y \vee x.$

ОТСТУПЛЕНИЕ

Эквивалентные преобразования, рассмотренные здесь, являются частным случаем *системы подстановок термов*, или *системы правил переписывания*, рассмотренных в п. 2.2.3. В общем случае в такой системе задается класс формул и набор правил преобразования этих формул. Правила могут быть обратимыми и всегда применимыми (как в случае эквивалентных преобразований булевых формул), но могут не иметь обратных и быть обусловленными. С этими системами связан целый ряд проблем теории алгоритмов. Например, для заданной системы правил и двух заданных формул, определить, можно ли преобразовать одну формулу в другую и найти кратчайшую последовательность преобразований. Системы подстановок термов нашли множество практически важных применений в математической логике, в теории алгоритмов и в практическом программировании. Заметим, однако, что построение эффективного алгоритма отыскания кратчайшей последовательности эквивалентных преобразований является непростой задачей даже для булевых формул, не говоря уже о других классах математических объектов и других системах переписывания.

3.4.5. Минимальные дизъюнктивные формы

Булева функция может быть задана бесконечным числом различных, но равносильных формул. Возникает естественная задача: для данной булевой функции найти реализующую формулу, обладающую теми или иными свойствами.

Практически наиболее востребованной оказалась задача минимизации: найти минимальную формулу, реализующую функцию. Но и в этой постановке задача имеет множество вариантов:

1) найти реализующую формулу, содержащую наименьшее количество переменных;
2) найти реализующую формулу, содержащую наименьшее количество определённых операций;
3) найти реализующую формулу, содержащую наименьшее количество подформул определённого вида.

Кроме того, могут быть наложены ограничения на синтаксический вид искомой формулы, набор операций, которые разрешается использовать в формуле, и т. д. Из всего этого разнообразия наиболее детально, по-видимому, изучена задача отыскания дизъюнктивных форм, минимальных по числу вхождений переменных. Эту задачу мы бегло рассматриваем в заключительных параграфах данного раздела.

Дизъюнктивной формой называется формула вида

$$\bigvee_{i=1}^{k} K_i, \quad \text{где } K_i = \bigwedge_{p=1}^{m_i} x_{j_p}^{\sigma_p}.$$

Формула K_i называется элементарной конъюнкцией, переменные в элементарную конъюнкцию входят в установленном порядке (хотя и не обязательно все n). Количество переменных в конъюнкции называется её *рангом* (обозначение $|K_i|$).

ТЕОРЕМА. *Число различных дизъюнктивных форм n переменных равно 2^{3^n}.*

Доказательство. Переменная либо не входит в элементарную конъюнкцию, либо входит с отрицанием, либо входит без отрицания. Таким образом, существует 3^n элементарных конъюнкций, а каждое подмножество множества элементарных конъюнкций даёт дизъюнктивную форму. □

Из этой теоремы можно извлечь два практических вывода, важных для рассматриваемой задачи минимизации дизъюнктивной формы. Во-первых, существует тривиальный алгоритм решения задачи: перебрать все формы (их конечное число) и выбрать минимальную. Во-вторых, количество дизъюнктивных форм с ростом n растёт очень быстро, и уже при небольших n полный перебор практически неосуществим.

3.4.6. Геометрическая интерпретация

При обсуждении задач, связанных с булевыми функциями, очень полезна следующая геометрическая интерпретация. Двоичным наборам из n элементов можно взаимно однозначно сопоставить вершины n-мерного единичного гиперкуба E^n.

Пример. На рис. 3.1 представлен гиперкуб для $n = 3$.

При этом булева функция $f(x_1, \ldots, x_n)$ задается подмножеством N вершин гиперкуба, на которых её значение равно 1: $N \stackrel{\text{Def}}{=} \{(a_1, \ldots, a_n) \mid f(a_1, \ldots, a_n) = 1\}$.

Пример. Для функции g имеем $N = \{(0,0,0), (0,0,1), (0,1,0), (1,1,0)\}$.

На рис. 3.1 выделены вершины, соответствующие функции $g(x, y, z)$, имеющей следующую таблицу истинности:

x	y	z	$g(x,y,z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

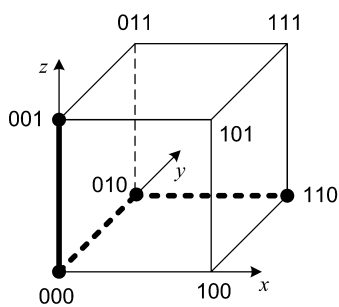


Рис. 3.1. Представление булевой функции в виде гиперкуба

Каждой элементарной конъюнкции $K = x_{i_1}^{\sigma_1} \wedge \ldots \wedge x_{i_k}^{\sigma_k}$ соответствует множество вершин гиперкуба, у которых $x_{i_1} = \sigma_1, \ldots, x_{i_k} = \sigma_k$, а значения остальных координат произвольны. Другими словами, элементарной конъюнкции K сопоставляется множество вершин $\{(a_1, \ldots, a_n) \in E^n \mid K(a_1, \ldots, a_n) = 1\}$, на которых конъюнкция

имеет значение 1. Мы обозначаем одной и той же буквой и элементарную конъюнкцию, и то множество вершин, на котором она принимает значение 1. Легко видеть, что элементарная конъюнкция k переменных образует $(n-k)$-мерную грань гиперкуба E^n.

Примеры

1. Элементарной конъюнкции $\neg x \wedge \neg y$ соответствует ребро $((0,0,0),(0,0,1))$: на рисунке это ребро выделено.
2. Элементарной конъюнкции z соответствует верхняя грань куба.

Теперь ясен геометрический смысл *задачи минимизации дизъюнктивной формы*. Дано подмножество N вершин гиперкуба E^n. Требуется найти такой набор гиперграней R_i, чтобы они в совокупности образовывали покрытие N ($N = \bigcup_{i=1}^{k} K_i$) и сумма рангов $\sum\limits_{i=1}^{k} |K_i|$ была минимальна.

3.4.7. Сокращённые дизъюнктивные формы

Известно несколько различных методов решения задачи минимизации дизъюнктивной формы. Все они используют одну и ту же идею: уменьшить по возможности множество рассматриваемых элементарных конъюнкций и затем найти минимальную дизъюнктивную форму, перебирая оставшиеся конъюнкции. Рассмотрим один из самых простых методов этого типа.

Пусть функция f задана множеством N вершин единичного гиперкуба E^n. Если $K \subset N$, то конъюнкция K называется *допустимой* для функции f. Ясно, что в минимальную (да и любую другую) дизъюнктивную форму функции f могут входить только допустимые конъюнкции. Если $K \cap (E^n \setminus N) \neq \varnothing$, то K — недопустимая конъюнкция. Поэтому для каждого набора $(a_1, \ldots, a_n)$ из множества $E^n \setminus N$ следует удалить из множества всех 3^n конъюнкций те 2^n конъюнкций, которые можно построить из сомножителей $\{x_1^{a_1}, \ldots, x_n^{a_n}\}$.

Пример. Для функции g имеем

$g(x,y,z)=0$	Недопустимые конъюнкции							
$(0,1,1)$	1	$\neg x$	y	z	$\neg x \wedge y$	$\neg x \wedge z$	$y \wedge z$	$\neg x \wedge y \wedge z$
$(1,0,0)$	1	x	$\neg y$	$\neg z$	$x \wedge \neg y$	$x \wedge \neg z$	$\neg y \wedge \neg z$	$x \wedge \neg y \wedge \neg z$
$(1,0,1)$	1	x	$\neg y$	z	$x \wedge \neg y$	$x \wedge z$	$\neg y \wedge z$	$x \wedge \neg y \wedge z$
$(1,1,1)$	1	x	y	z	$x \wedge y$	$x \wedge z$	$y \wedge z$	$x \wedge y \wedge z$

Удаляя из множества всех 27 конъюнкций те, которые не являются допустимыми, получаем следующий список допустимых конъюнкций:

$$y \wedge \neg z,\ \neg x \wedge \neg y,\ \neg x \wedge \neg z,\ x \wedge y \wedge \neg z,\ \neg x \wedge y \wedge \neg z,\ \neg x \wedge \neg y \wedge z,\ \neg x \wedge \neg y \wedge \neg z.$$

Конъюнкция K называется *максимальной* для функции f, если

$$K \subset N \ \&\ (K \subset K' \subset N \Longrightarrow K' = K)\,.$$

Очевидно, что всякая допустимая конъюнкция содержится в некоторой максимальной. Поэтому совокупность всех максимальных конъюнкций образует покрытие множества N, то есть дизъюнктивную форму, которая называется *сокращённой дизъюнктивной нормальной формой*. Сокращённая дизъюнктивная нормальная форма определена для функции f однозначно (с учётом установленного порядка переменных).

Пример. Для функции g сокращённая дизъюнктивная нормальная форма имеет вид $(\neg x \wedge \neg y) \vee (\neg x \wedge \neg z) \vee (y \wedge \neg z)$, что существенно короче, чем СДНФ этой функции: $(\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \vee (x \wedge y \wedge \neg z)$.

На рис. 3.1 выделены рёбра (000–001, 000–010, 010–110), соответствующие сокращённой дизъюнктивной нормальной форме.

ТЕОРЕМА. *Минимальная дизъюнктивная форма является подформулой сокращённой дизъюнктивной нормальной формы.*

Доказательство. От противного. Пусть минимальная форма содержит не максимальную конъюнкцию. Тогда эту конъюнкцию можно заменить соответствующей максимальной, при этом покрытие сохранится, а сумма рангов уменьшится, что противоречит минимальности формы. □

Пример. Для функции g минимальная дизъюнктивная форма имеет вид

$$\neg x \wedge \neg y \vee y \wedge \neg z.$$

Действительно, на рис. 3.1 видно, что рёбра 000–001 и 010–110 являются максимальными конъюнкциями и в совокупности покрывают множество N.

3.5. Представление булевых функций в программах

Для представления булевых функций можно использовать стандартные методы представления функций, а также некоторые специальные приёмы, и эти идеи часто оказываются применимыми для представления других, более сложных объектов.

3.5.1. Табличные представления

Область определения и область значений у булевой функции конечна, поэтому булева функция может быть представлена массивом. Самое бесхитростное представление прямо воспроизводит таблицу истинности. Если условиться, что кортежи в таблице всегда идут в установленном порядке, то для представления булевой функции $f(x_1, \ldots, x_n)$ достаточно хранить столбец значений:
F_1 : **array** $[0..(2^n - 1)]$ **of** $0..1$

Пример. В этом и последующих примерах рассматривается булева функция $g(x, y, z)$, заданная следующей таблицей истинности:

x	y	z	$g(x, y, z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$F_1(g)$: **array** $[0..7]$ **of** $0..1 := (1, 1, 1, 0, 0, 0, 1, 0)$

Данное представление — не самое эффективное. В частности, если набор аргументов задан массивом x : **array** $[1..n]$ **of** 0..1, то для вычисления значения функции f с помощью представления F_1 необходимо сначала вычислить индекс d (то есть номер кортежа в установленном порядке), чтобы затем получить значение $F_1[d]$. Индекс d нетрудно вычислить, например, с помощью следующего алгоритма.

Алгоритм 3.2. Вычисление номера кортежа в установленном порядке

Вход: кортеж x : **array** $[1..n]$ **of** 0..1 значений переменных.
Выход: номер d кортежа x при перечислении кортежей в установленном порядке.

```
d := 0  // начальное значение индекса
for i from 1 to n do
    d := d * 2 + x[i]  // сдвигаем влево и добавляем разряд
end for
```

Обоснование. Если рассматривать кортеж булевых значений как двоичную запись числа, то это число — номер кортежа в установленном порядке. Значение числа d, заданного в позиционной двоичной системе счисления цифрами $x_1 \dots x_n$, определяется следующим образом:

$$d = x_1 2^{n-1} + \dots + x_n 2^0 = \sum_{i=1}^{n} x_i 2^{n-i} = 2(2(\dots(2x_1 + x_2)\dots) + x_{n-1}) + x_n.$$

Цикл в алгоритме непосредственно вычисляет последнюю формулу, которая является частным случаем *схемы Горнера*. □

ЗАМЕЧАНИЕ

По схеме Горнера можно определить значение числа d по записи $x_1 \dots x_n$ в *любой* позиционной системе счисления с основанием b следующим образом:
$d = b(b(\dots(bx_1 + x_2)\dots) + x_{n-1}) + x_n$.

Более эффективным представлением таблицы истинности является использование n-мерного массива: F_2 : **array** $[0..1, \dots, 0..1]$ **of** 0..1 В случае использования представления F_2 значение функции $f(x_1, \dots, x_n)$ задаётся выражением $F_2[x_1, \dots, x_n]$.

Пример. Для функции g из предыдущего примера
F_2 : **array** $[0..1, 0..1, 0..1]$ **of** $0..1 = (((1,1),(1,0)),((0,0),(1,0)))$.

ЗАМЕЧАНИЕ

Мы принимаем, что многомерные массивы располагаются в линейной памяти «по строкам».

Представления булевой функции n переменных в виде массива занимают память объёмом $O(2^n)$.

3.5.2. Строковые представления

Булеву функцию можно представить с помощью реализующей её формулы. Существует множество способов представления формул, но наиболее удобной для человека является запись в виде цепочки символов на некотором формальном языке.

Как уже указывалось, для любой булевой функции существует бесконечно много различных реализующих её формул. Однако булевы функции имеют нормальные формы, в частности СДНФ, и при установленном порядке переменных СДНФ единственна.

СДНФ булевой функции может быть построена по заданной таблице истинности с помощью следующего алгоритма.

Алгоритм 3.3. Построение СДНФ

Вход: вектор x : **array** $[1..n]$ **of string** идентификаторов переменных,
вектор F_1 : **array** $[0..2^n-1]$ **of** 0..1 значений функции при установленном порядке кортежей.

Выход: последовательность символов, образующих запись формулы СДНФ для заданной функции.

```
f := false  // признак присутствия левого операнда дизъюнкции
for i from 0 to 2^n − 1 do
  if F1[i] = 1 then
    if f then yield '∨' else f := true endif  // знак дизъюнкции
    g := false  // признак присутствия левого операнда конъюнкции
    for j from 1 to n do
      if g then yield '∧' else g := true endif  // знак конъюнкции
       v := (i div 2^(j−1)) mod 2   // j-й разряд i-го кортежа
      if v = 0 then
        yield '¬'  // знак отрицания
      end if
      yield x[j]  // добавление в формулу переменной
    end for
  end if
end for
```

Обоснование. Данный алгоритм буквально воспроизводит словесную запись следующего правила: для каждой строки таблицы истинности, для которой значение функции равно 1, построить дизъюнктивное слагаемое, включающее все переменные, причём те переменные, которые имеют значение 0 в этой строке, входят со знаком отрицания. Остальное в этом алгоритме — мелкие программистские «хитрости», которые полезно один раз посмотреть, но не стоит обсуждать. □

Пример. Для функции g, используемой в примерах данного раздела, алгоритм построит строку $\neg x \wedge \neg y \wedge \neg z \vee \neg x \wedge \neg y \wedge z \vee \neg x \wedge y \wedge \neg z \vee x \wedge y \wedge \neg z$.

Представление функции в виде формулы, выраженной как строка символов, совершенно необходимо при реализации интерфейса пользователя в системах компьютерной алгебры, но крайне неудобно при выполнении других манипуляций с формулами. В следующем параграфе рассматривается представление СДНФ, более удобное, например, для вычисления значения функции.

3.5.3. Алгоритм вычисления значения булевой функции

Некоторые классы формул допускают более эффективную интерпретацию по сравнению с алгоритмом Eval (п. 3.2.1). Рассмотрим алгоритм вычисления значения

булевой функции, заданной в виде СДНФ, для заданных значений переменных $x_1, \ldots, x_n$. В этом алгоритме используется следующее представление данных. СДНФ задана массивом F_3 : **array** $[1..k, 1..n]$ **of** 0..1, где строка $F_3[i, *]$ содержит набор значений $\sigma_1, \ldots, \sigma_n$, для которого $f(\sigma_1, \ldots, \sigma_n) = 1$, $i \in 1..k$.

Пример. Для функции g

F_3 : **array** $[1..4, 1..3]$ **of** $0..1 = ((0,0,0),(0,0,1),(0,1,0),(1,1,0))$.

ОТСТУПЛЕНИЕ

Быстрое вычисление значения СДНФ имеет, помимо теоретического, большое практическое значение. Например, во многих современных программах с графическим интерфейсом для составления сложных логических условий используется наглядный бланк в виде таблицы: в клетках записываются условия, причём клетки одного столбца считаются соединёнными конъюнкцией, а столбцы — дизъюнкцией, то есть образуют ДНФ (или наоборот, в таком случае получается КНФ). В частности, так устроен графический интерфейс QBE (Query-by-Example), применяемый для формулировки логических условий при запросе к СУБД.

Алгоритм 3.4. Алгоритм вычисления значения СДНФ

Вход: массив, представляющий СДНФ: f : **array** $[1..k, 1..n]$ **of** 0..1;
множество значений переменных x : **array** $[1..n]$ **of** 0..1.
Выход: 0..1 — значение булевой функции.

```
for i from 1 to k do
  for j from 1 to n do
    if f[i,j] ≠ x[j] then
      next for i  // x_j ≠ σ_j ⟹ x_j^{σ_j} = 0 ⟹ x_1^{σ_1} ∧ ... ∧ x_n^{σ_n} = 0
    end if
  end for
  return 1  // x_1^{σ_1} & ... & x_n^{σ_n} = 1 ⟹ ⋁_{(σ_1,...,σ_m)} x_1^{σ_1} ∧ ... ∧ x_n^{σ_n} = 1
end for
return 0  // все слагаемые в дизъюнкции равны нулю
```

Обоснование. Алгоритм основан на следующих (очевидно, верных) правилах. Можно прекратить вычисление конъюнкции, как только получен конъюнктивный сомножитель, равный 0 (вся конъюнкция имеет значение 0). Можно прекратить вычисление дизъюнкции, как только получено дизъюнктивное слагаемое, равное 1 (вся дизъюнкция имеет значение 1). □

Этот алгоритм в худшем случае выполняет $k \cdot n$ сравнений, а в среднем — гораздо меньше. Таким образом, он существенно эффективнее общего алгоритма интерпретации.

3.5.4. Представление булевых функций арифметическими полиномами

Существует удобное представление булевых функций, простое в понимании и эффективное в реализации, основанное на применении обычных арифметических операций к булевским значениям 0 и 1.

Пример. Нетрудно убедиться, что $\neg x = 1 - x$.

Возникает вопрос: можно ли представить произвольную булеву функцию таким образом?

Арифметическим полиномом $P(x_1, \ldots, x_n)$ для булевой функции $f(x_1, \ldots, x_n)$ называется полином n переменных $x_1, \ldots, x_n$ с целыми коэффициентами такой, что $\forall\, x_1, \ldots, x_n \ \ (P(x_1, \ldots, x_n) = f(x_1, \ldots, x_n))$.

Пример. Рассмотрим булеву функцию, реализуемую формулой $(x \vee y)\,\&\,z$ и полином $xz + yz - xyz$. Прямое вычисление, проведенное в таблице, показывает, что этот полином является реализующим для данной функции.

x	y	z	$(x \vee y)\ \&\ z$	$xz + yz - xyz$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

Константы (булевские значения) являются своими арифметическими полиномами. Арифметический полином тождественной функции совпадает с ней, а арифметический полином отрицания построен в примере выше. Арифметические полиномы для булевых функций двух существенных переменных также построены и приведены в следующей таблице.

Переменная x		0	0	1	1	
Переменная y		0	1	0	1	
Название	Формула					Полином
Конъюнкция	$x \wedge y$	0	0	0	1	xy
	$x \wedge \neg y$	0	0	1	0	$x - xy$
	$\neg x \wedge y$	0	1	0	0	$y - xy$
Сложение по модулю 2	$x +_2 y$	0	1	1	0	$x + y - 2xy$
Дизъюнкция	$x \vee y$	0	1	1	1	$x + y - xy$
Стрелка Пирса	$x \downarrow y$	1	0	0	0	$1 - x - y + xy$
Эквивалентность	$x \equiv y$	1	0	0	1	$1 - x - y + 2xy$
	$x \vee \neg y$	1	0	1	1	$1 - y + xy$
Импликация	$x \to y$	1	1	0	1	$1 - x + xy$
Штрих Шеффера	$x \mid y$	1	1	1	0	$1 - xy$

ТЕОРЕМА. *Для любой булевой функции существует реализующий её арифметический полином.*

Доказательство. Любую булеву функцию можно задать формулой, операциями которой являются функции, для которых известны реализующие арифметические полиномы. В частности, любую булеву функцию можно выразить через дизъюнкцию, конъюнкцию и отрицание (п. 3.4.3). Тогда полином можно построить, подставляя арифметические полиномы для каждой операции формулы, начиная с внешней, а затем упростить получившийся полином, раскрыв скобки, вычислив степени и приведя подобные. □

ЗАМЕЧАНИЕ

Поскольку $\forall n > 0 \ \ (1^n = 1 \ \& \ 0^n = 0)$, вычисление степеней сводится к отбрасыванию показателей степени перед дальнейшими упрощениями.

ОТСТУПЛЕНИЕ

Представление булевых функций арифметическими полиномами может быть выгодно, так как вычисление значений линейных полиномов программно весьма просто, ибо сводится к суммированию коэффициентов при переменных, не равных нулю.

Пример. Представим арифметическим полиномом функцию g, рассматриваемую в предыдущих параграфах. Из примера в п. 3.4.6 известно, что её минимальная дизъюнктивная форма имеет вид $(\neg x \wedge \neg y) \vee (y \wedge \neg z)$. Далее имеем:

$$\begin{aligned}
&(1-x)(1-y) + y(1-z) - (1-x)(1-y)y(1-z) = \\
=&1 - x - y + xy + y - yz - (1 - x - y + xy)(y - yz) = \\
=&1 - x - y + xy + y - yz - y + xy + y^2 - xy^2 + yz - xyz - y^2z + xy^2z = \\
=&1 - x - y + xy + y - yz - y + xy + y - xy + yz - xyz - yz + xyz = \\
=&1 - x + xy - yz.
\end{aligned}$$

Рассмотрим ещё один способ реализации булевой функции, основанный на той же идее. Пусть булева функция f задана формулой F в базисе $\{\wedge, \vee, \neg\}$ некоторой дизъюнктивной формой. Выполним в формуле следующие замены логических операций арифметическими: $\neg x \mapsto 1 - x$; $x \wedge y \mapsto xy$; $x \vee y \mapsto x + y$. Такие замены уместно назвать *буквальными*. Получится арифметическое выражение $A(F)$, которое реализует некоторую функцию f_1. В общем случае функции f и f_1 различны.

Пример.

x	y	$F = x \vee y$	$A(F) = x + y$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	2

Однако, если формула F предварительно *ортогонализирована*, то есть эквивалентно преобразована к такому виду, когда все элементарные конъюнкции ортогональны, то значения F и $A(F)$ при буквальных заменах совпадают.

ЗАМЕЧАНИЕ

Ортогонализация сводится к систематическому применению тождества $a \vee b = a \wedge b \vee \neg a \wedge b \vee a \wedge \neg b$.

Пример.

x	y	$F = x \wedge y \vee \neg x \wedge y \vee x \wedge \neg y$	$A(F) = xy + (1-x)y + x(1-y)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Ортогонализация усложняет исходную формулу, но позволяет упростить замены при построении арифметического полинома.

Имеется способ избежать ортогонализации и при этом сохранить буквальную простоту преобразования логической формулы в арифметическую. Для этого достаточно заметить, что ортогонализация фактически позволяет отбросить в арифметическом полиноме дизъюнкции $x + y - xy$ слагаемое xy, поскольку оно всегда равно нулю в ортогонализированной формуле. Отбрасываемое слагаемое неотрицательное, поэтому $F = (A(F) > 0)$, то есть в качестве значения функции можно брать *знак* значения арифметического выражения, полученного буквальными заменами.

Пример. Пусть некоторая функция задана неортогонализированной дизъюнктивной формой $x \wedge y \vee y \wedge z$. Общий алгоритм даёт арифметический полином $xy+yz-xyz$. Буквальная замена даёт более простое выражение $xy + yz$, знак которого совпадает со значением арифметического полинома и функции.

3.5.5. Карты Карно

Карта Карно[1] (используется также термин *диаграмма Вейча*) — это замечательное представление булевой функции, позволяющее наглядно и эффектно описать и реализовать многие, в том числе сложные, операции с булевыми функциями и реализующими их формулами. В картах Карно применяются сразу несколько понятий, рассматриваемых в этом учебнике: код Грея (1.3.3), таблицы истинности (3.1.1), минтермы (3.4.2), единичный гиперкуб (3.4.6), табличные представления булевой функции (3.6.1).

Карта Карно для представления функций n переменных является прямоугольной таблицей, которая содержит 2^n ячеек. Если n чётное, $n = 2k$, то таблица содержит 2^k строк и 2^k столбцов, а если нечётное, $n = 2k+1$, то для удобства отображения обычно принимают, что таблица содержит 2^k строк и 2^{k+1} столбцов. Обозначим число строк в таблице $r(n) := 2^{n \,\mathbf{div}\, 2}$, число столбцов в таблице $c(n) := 2^{n \,\mathbf{div}\, 2+n \,\mathbf{mod}\, 2}$. Поскольку всего существует 2^n минтермов для n переменных, можно считать, что каждая ячейка карты Карно символизирует один минтерм, а вся карта в целом является перечислением минтермов.

Вообще говоря, существует $2^n!$ способов расставить минтермы по ячейкам карты Карно. Например, следующий прямолинейный алгоритм заполнения карты Карно минтермами в установленном порядке.

Алгоритм 3.5. Заполнение карты Карно в порядке возрастания минтермов

Вход: число переменных n.
Выход: карта Карно K : **array** $[1..r(n), 1..c(n)]$ **of array** $[1..n]$ **of** $0..1$

```
m := 0
for i from 1 to r(n) do
  for j from 1 to c(n) do
    K[i, j] := B(m)  // B(m) — двоичный код числа m
    m := m + 1
  end for
end for
```

[1] Морис Карно (род. 1924).

ЗАМЕЧАНИЕ

Полезно сопоставить этот алгоритм с алгоритмом 1.3.

Однако использование более сложных алгоритмов перечисления минтермов позволяет построить карту Карно, обладающую рядом полезных свойств. Во-первых, прямоугольную таблицу можно рассматривать как развёртку тора, то есть считать верхнюю сторону смежной с нижней, а правую — с левой. В этом случае у каждой ячейки имеются четыре соседа. Во-вторых, можно расположить минтермы таким образом, что любые два соседних отличаются ровно в одном разряде. Подобным же свойством обладают коды Грея и вершины единичного гиперкуба.

Зафиксируем один из возможных способов построения карты Карно с указанными свойствами с помощью следующего алгоритма. Разобьём множество переменных на две группы, в одной $r(n)$ переменных, а в другой $c(n)$ переменных. Множество кортежей значений переменных каждой группы перечислим не в установленном порядке, а в порядке, задаваемом алгоритмом п. 1.4.2 (код Грея). Эти коды отмечают строки и столбцы карты Карно, соответственно, а минтерм в ячейке получается конкатенацией кодов строки и столбца. Легко видеть, что полученная карта обладает требуемыми свойствами.

Алгоритм 3.6. Построение двумерного кода Грея

Вход: число переменных n.
Выход: карта Карно K : **array** $[1..r(n), 1..c(n)]$. **of array** $[1..n]$ **of** $0..1$

```
for i from 1 to r(n) do
  for j from 1 to c(n) do
    K[G(i), G(j)] := G(i)G(j)  //G(i) — левый зеркальный код Грея
  end for
end for
```

Здесь $G(i)$ — левый зеркальный код Грея (п. 1.3.4), а операция конкатенации кодов никак не обозначается (п. 1.1.5). Далее в этом учебнике данный способ заполнения карты Карно считается фиксированным.

ЗАМЕЧАНИЕ

Приведенный способ построения карты Карно далеко не единственный. Можно варьировать приведенный алгоритм: разбивать переменные на две группы произвольным образом, а не по порядку, использовать различные коды Грея, а не только левый зеркальный, различным образом сцеплять разряды кодов строки и столбца, а не только конкатенировать их. Во всех случаях получаются карты, в которых соседние минтермы отличаются ровно в одном разряде.

Ниже приведены карты Карно для $n = 2$, $n = 3$, $n = 4$ и $n = 5$, построенные приведенным алгоритмом.

$n = 2$		x_2	
		0	1
x_1	0	00	01
	1	10	11

$n = 3$		x_2, x_3			
		00	10	11	01
x_1	0	000	010	011	001
	1	100	110	111	101

$n=4$		x_3, x_4			
		00	10	11	01
x_1, x_2	00	0000	0010	0011	0001
	10	1000	1010	1011	1001
	11	1100	1110	1111	1101
	01	0100	0110	0111	0101

$n=5$		x_3, x_4, x_5							
		000	001	011	010	110	111	101	100
x_1, x_2	00	00000	00001	00011	00010	00110	00111	00101	00100
	10	10000	10001	10011	10010	10110	10111	10101	10100
	11	11000	11001	11011	11010	11110	11111	11101	11100
	01	01000	01001	01011	01010	01110	01111	01101	01100

Хранить в программе информацию, представленную в этих таблицах, совершенно не обязательно, поскольку при фиксированном способе заполнения карты легко вычислить минтерм, находящийся в клетке $K[i,j]$. Другими словами, функция $K(i,j)$, доставляющая код минтерма в ячейке $K[i,j]$, вычисляется так:

$$K(i,j) := G(i)G(j) = (B(i) +_2 (B(i) \,\mathbf{div}\, 2))(B(j) +_2 (B(j) \,\mathbf{div}\, 2)),$$

где функция $G(i)$ доставляет i-й зеркальный код Грея (см. теорему 1.3.4), функция $B(i)$ доставляет двоичный код числа i, а операция конкатенации битовых шкал никак не обозначается.

3.5.6. Представление булевой функции на карте Карно

Поскольку всякая булева функция представляется дизъюнкцией минтермов (п. 3.4.2), указание некоторого *подмножества* ячеек карты Карно является *представлением* булевой функции. Допустим, что карта Карно для заданного n, то есть распределение минтермов по ячейкам таблицы, заранее вычислена. Тогда можно построить следующее представление булевой функции, заданной таблицей истинности. Рассмотрим булеву матрицу K : **array** $[1..r(n), 1..c(n)]$ **of** 0..1 того же размера, что и карта Карно. Каждой строке таблицы истинности соответствует некоторый минтерм. Если значение в этой строке равно 1, то записываем 1 в соответствующую ячейку матрицы, в противном случае записываем 0. Фактически, элементы вектора значений таблицы истинности булевой функции раскладываются по ячейкам булевой матрицы, а правило раскладывания задается картой Карно.

Пример. Функция $g(x,y,z)$, заданная в примере п. 3.6.1, имеет следующее представление на карте Карно.

1	1	0	1
0	1	0	0

В программе такая карта представляется массивом
$F_4(g)$: **array** $[1..2, 1..4]$ **of** $0..1 := ((1,1,0,1),(0,1,0,0))$.

Карта Карно действительно является представлением булевой функции, поскольку с её помощью можно выполнять основную присущую функциям операцию: вычислять значение функции по заданным значениям аргументов. Вообще говоря, если булева функция представлена картой Карно и задан набор значений переменных

$\sigma_1, \ldots, \sigma_n$, то ответ получается за один шаг: значение функции на заданном наборе содержится в ячейке, соответствующей минтерму $(x_1^{\sigma_1} \,\&\, \ldots \,\&\, x_n^{\sigma_n})$.

Алгоритм 3.7. Вычисление значения булевой функции по карте Карно

Вход: карта Карно K : **array** $[1..r(n), 1..c(n)]$ **of** 0..1,
массив значений переменных x : **array** $[1..n]$ **of** 0..1.
Выход: 0..1 — значение булевой функции.
$i := \mathrm{Grey2Int}(r(n), x[1..r(n)])$ // номер строки
$j := \mathrm{Grey2Int}(c(n), x[(r(n)+1)..n])$ // номер столбца
return $K[i, j]$ // возвращаемое значение

Рекуррентные поразрядные вычисления выполняются в функции Grey2Int, которая вычисляет целое число — номер заданного кода Грея заданной длины.

Вход: n — длина кода Грея, G : **array** $[1..n]$ **of** 0..1 — код Грея.
Выход: целое неотрицательное число — номер кода.

```
d := 0 // вычисляемое значение числа
p := 0 // значение предыдущего разряда двоичного кода числа
for i from 1 to n do
  b := G[i] +2 p // очередной разряд двоичного кода числа
  p := b // предыдущий разряд двоичного кода числа
  d := d * 2 // сдвиг числа влево на 1 разряд
  if b = 1 then
    d := d + 1 // добавляем единицу в число
  end if
end for
return d
```

Обоснование. При выбранном способе заполнения карты Карно код Грея номера строки располагается в левых $r(n)$ разрядах, а код Грея номера столбца в правых $c(n)$ разрядах. В цикле по следствию 2 теоремы п. 1.3.4 определяется очередной разряд двоичного кода и по схеме Горнера вычисляется значение числа d (п. 3.5.1). □

Наряду с вычислением значений, карты Карно позволяют строить различные формульные представления для булевых функций и их комбинаций. Карта Карно для одной заданной функции очевидным образом задает её СДНФ — достаточно соединить их знаками дизъюнкции минтермы, указанные ячейками, содержащими 1. Столь же просто можно построить СДНФ для отрицания, дизъюнкции и конъюнкции заданных функций. Для получения СДНФ отрицания достаточно просто инвертировать матрицу. Для получения СДНФ конъюнкции или дизъюнкции произвольного числа k булевых функций можно поступить следующим образом. Взять матрицу, заполненную нулями, и добавить в каждую ячейку независимо 0 или 1 для каждой функции. При этом в некоторых ячейках окажется 0, в некоторых 1, а в некоторых какие-то числа вплоть до k. Все ненулевые ячейки образуют СДНФ дизъюнкции, все ячейки, равные k, образуют СДНФ конъюнкции.

3.5.7. Минимизация формул картами Карно

Карты Карно являются наглядным средством решения задачи минимизации дизъюнктивных форм, обсуждаемой в пп. 3.4.5–3.4.7. Пусть булева функция представлена картой Карно. Если при этом в соседних ячейках оказались 1, то по основному

свойству карт Карно соответствующие минтермы отличаются точно в одном разряде, и значит, к ним можно применить правило склеивания (п. 3.4.4). В таком случае дизъюнкцию данных минтермов можно заменить элементарной конъюнкцией (п. 3.4.5) ранга $n-1$. На карте Карно соответствующие ячейки можно объединить, показывая, что они образуют *компактную группу*. Результат склеивания двух минтермов обычно также обозначают минтермом, причём в позиции разряда, по которому проведено склеивание, поставлен прочерк.

Пример. Результат склеивания минтермов (0101) и (0111) обозначается (01-1).

Если получились две смежные непересекающиеся компактные группы, образующие прямоугольник, их можно по правилу склеивания объединить в компактную группу, соответствующую элементарной конъюнкции ранга $n-2$ и т. д. Компактные группы могут включаться друг в друга, могут пересекаться, могут примыкать другу к другу, но не образовывать новой компактной группы. Компактная группа, которая не входит ни какую другую компактную группу, называется *максимальной*.

Операция склеивания является эквивалентным преобразованием (п. 3.4.4), сохраняет дизъюнктивность формы и при этом сокращает количество вхождений переменных, поэтому операция выделения и склеивания компактных групп — основное средство минимизации дизъюнктивных форм.

Каждая прямоугольная компактная группа из 2^k смежных ячеек, содержащих 1, соответствует элементарной конъюнкции ранга $n-k$. В частности, если ячейка содержащая 1, изолированная (нет смежных ячеек, содержащих 1), то эта ячейка образует компактную группу, соответствующую минтерму, то есть элементарной конъюнкции ранга n. Таким образом, задача минимизации дизъюнктивной формы сводится к отысканию покрытия всех ячеек, содержащих 1 на карте Карно, минимальным количеством компактных групп.

Пример. На рис. 3.2 показаны три компактные группы, которые имеются на карте Карно функции $g(x,y,z)$ из примера 3.6.1. Компактные группы (00-) и (-00) не пересекаются, а компактные группы (0-0) и (-10), а также (00-) и (0-0), напротив, пересекаются. Все три компактные группы являются максимальными. Хотя все четыре единицы на этом рисунке находятся в соседних ячейках, компактную группу они не образуют, поскольку форма не прямоугольная. Минимальное покрытие на карте Карно содержит два элемента: (00-) и (-10), что соответствует дизъюнктивной форме $\neg x \,\&\, \neg y \vee y \,\&\, \neg z$.

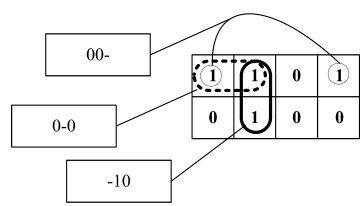


Рис. 3.2. Три компактные группы

ЗАМЕЧАНИЕ

Элементарную конъюнкцию, соответствующую компактной группе, часто называют *импликантой*, поскольку если g — элементарная конъюнкция, соответствующая некоторой компактной группе на карте Карно функции f, то имеет место импликация $g \to f$.

Пример. Рассмотрим карту Карно, приведенную на рис. 3.3 слева. На этой карте имеется 14 компактных групп ранга 3 (в центре) и еще 3 компактные группы ранга 2 (справа). Других компактных групп нет.

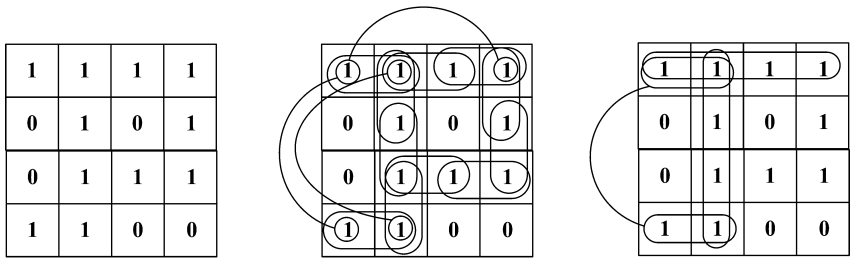


Рис. 3.3. Карта Карно и компактные группы различных рангов

Определить компактные группы, покрывающие все ячейки с 1 на карте Карно, можно многими различными способами, в этом и состоит трудность задачи минимизации дизъюнктивной формы. Известны различные алгоритмы нахождения минимальных дизъюнктивных форм с помощью карт Карно. Все эти алгоритмы переборные и используют методы сокращения перебора, основанные на фактах, устанавливаемых в следующих леммах.

ЛЕММА 1. *Достаточно рассматривать только максимальные компактные группы.*

Доказательство. Максимальные компактные группы соответствуют максимальным конъюнкциям, то есть сокращённой ДНФ, далее по теореме 3.4.7. □

Пример. Максимальные компактные группы для рассматриваемой функции приведены на рис. 3.4 в центре.

Если какая-то 1 на карте Карно покрывается только одной компактной группой, то говорят, что эта группа *входит в ядро* (данного покрытия).

ЛЕММА 2. *Всякая компактная группа, входящая в ядро сокращённой ДНФ, входит в минимальную ДНФ.*

Доказательство. По лемме 1 достаточно рассматривать сокращенную ДНФ. □

Пример. Компактные группы, показанные на рис. 3.3 справа, образуют ядро сокращенной ДНФ, показанной на рис. 3.4 в центре.

ЗАМЕЧАНИЕ

Если все максимальные компактные группы оказались в ядре, то сокращённая ДНФ является минимальной.

Компактную группу называют *избыточной* (в данном покрытии), если её можно удалить из покрытия без потери эквивалентности исходной СДНФ. Покрытие (и, соответственно, ДНФ) называется *тупиковым*, если оно не содержит избыточных компактных групп.

ЛЕММА 3. *Минимальная ДНФ является тупиковой.*

Доказательство. В противном случае ДНФ не была бы минимальной. □

Отсюда вытекает основная идея метода минимизации: построить сокращенную ДНФ, построить её ядро, перебрать все тупиковые ДНФ и найти минимальную.

Пример. Компактные группы, показанные на рис. 3.4, образуют покрытие, но соответствующая ДНФ не является минимальной (она содержит $6 \cdot 3 = 18$ литералов). Справа приведена одна из трёх минимальных ДНФ данной функции, содержащая $3 \cdot 2 + 2 \cdot 3 = 12$ литералов.

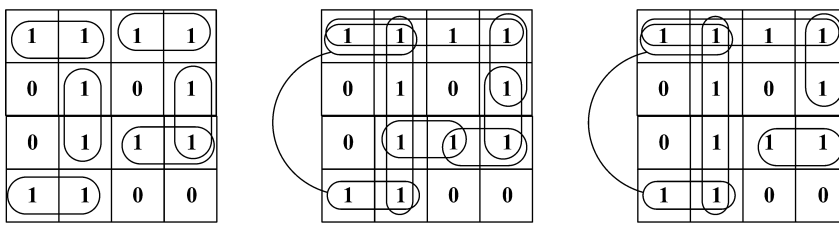


Рис. 3.4. Некоторая ДНФ, сокращённая ДНФ и минимальная ДНФ

3.5.8. Деревья решений

Наиболее эффективными с точки зрения экономии памяти и времени оказываются представления, которые не имеют прямой связи с «естественными» представлениями функции в виде графика (массива) или формулы (выражения), но специально ориентированы на выполнение операций.

Начнём с простого наблюдения. Таблицу истинности булевой функции n переменных можно представить в виде полного бинарного дерева высоты $n + 1$.

ЗАМЕЧАНИЕ

Бинарные деревья, равно как и другие виды деревьев, а также способы их представления в программах и соответствующая терминология, рассматриваются в главе 9.

Ярусы дерева соответствуют переменным, дуги дерева соответствуют значениям переменных, скажем, левая дуга — 0, а правая — 1. Листья дерева на последнем ярусе хранят значение функции на кортеже, соответствующем пути из корня в этот лист. Такое дерево называется *деревом решений* (или *семантическим деревом*).

Дерево решений можно сократить, если заменить корень каждого поддерева, все листья которого имеют одно и то же значение, этим значением. Иногда такое сокращение значительно уменьшает объём дерева.

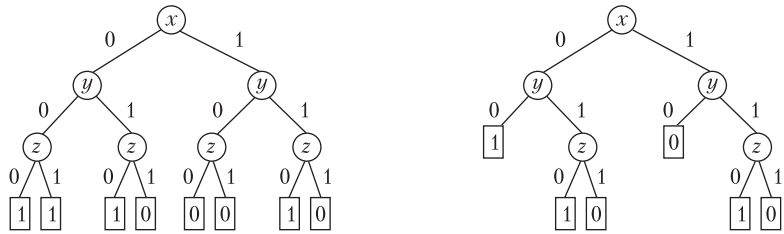


Рис. 3.5. Дерево решений и сокращённое дерево решений

Пример. На рис. 3.5 слева представлено дерево решений для функции g, а справа представлено сокращённое дерево решений для функции g.

Вычисление значения функции осуществляется проходом по дереву решений, как показано в алгоритме 3.8. При этом тип узла дерева N определён следующим образом: $N =$ **struct** $\{i : 0..1;\ l, r : \uparrow N\}$.

Алгоритм 3.8. Вычисление значения функции по дереву решений

Вход: указатель $T : \uparrow N$ на корень дерева решений,
массив x : **array** $[1..n]$ **of** 0..1 значений переменных.
Выход: 0..1 — значение булевой функции.

```
for i from 1 to n do
  if T.l = nil & T.r = nil then
    return T.i  // листовой узел — возвращаем значение
  else
    if x[i] then
      T := T.r  // 1 — переход вправо
    else
      T := T.l  // 0 — переход влево
    end if
  end if
end for
```

Дерево решений можно сделать ещё компактнее, если отказаться от древовидности связей, то есть допускать несколько дуг, входящих в узел. В таком случае мы получаем *бинарную диаграмму решений*. Бинарная диаграмма решений получается из бинарного дерева решений тремя последовательными преобразованиями.

1. Отождествляются листовые узлы, содержащие 0 и содержащие 1.

Пример. На рис. 3.6, *а* показано исходное полное дерево решений для функции g, а на рис. 3.6, *б* — результат первого преобразования.

2. В диаграмме выделяются изоморфные поддиаграммы и заменяются единственным их экземпляром.

Пример. На рис. 3.6, *б* видно, что два поддерева, корнями которых являются узлы z, находящиеся в середине, изоморфны. На рис. 3.6, *в* показан результат второго преобразования.

3. Исключаются узлы, обе исходящие дуги которых ведут в один узел.

Пример. На рис. 3.6, *в* видно, что крайние узлы z излишни — значение функции от значения z не зависит. Они удаляются, а входящие из них дуги продолжаются до тех узлов, в которые вели дуги из узлов z. На рис. 3.6, *г* показан результат третьего преобразования, который является построенной диаграммой решений.

Интерпретация бинарной диаграммы решений (вычисление значения функции) производится в точности так же, как и для дерева решений, то есть по алгоритму 3.8.

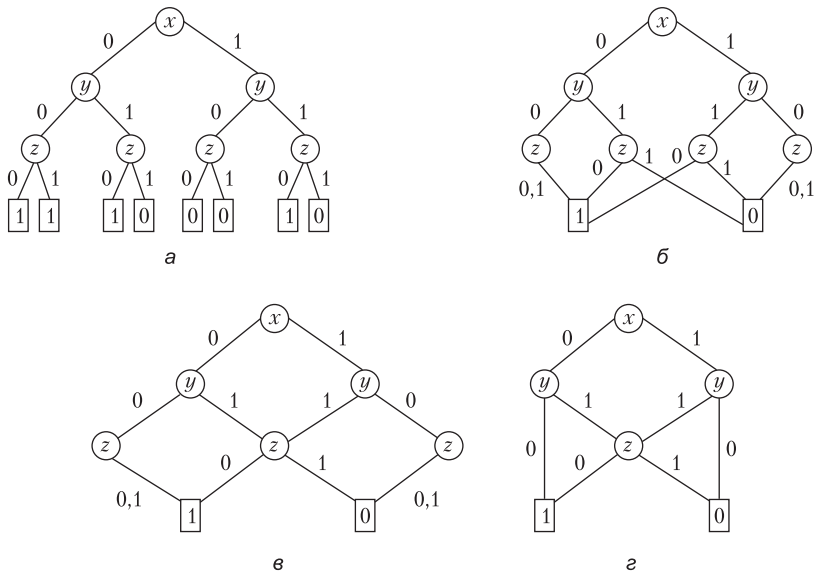


Рис. 3.6. Построение бинарной диаграммы решений

Результат преобразования дерева решений в диаграмму решений существенно зависит от того, в каком порядке рассматриваются переменные при построении исходного полного дерева решений.

Пример. На рис. 3.7 показаны последовательность преобразований и окончательная диаграмма решений для функции g в том случае, когда переменные рассматриваются в следующем порядке: y, x, z.

Полученная диаграмма компактнее: вычисление значения функции g требует всего двух операций. Фактически, диаграмма показывает, что функция g может быть реализована следующим условным выражением:

if y **then** $\neg z$ **else** $\neg x$ **end if**.

ОТСТУПЛЕНИЕ

Последний разобранный пример свидетельствует, что иногда можно построить такое специальное представление функции, которое позволяет хранить меньше информации и при этом

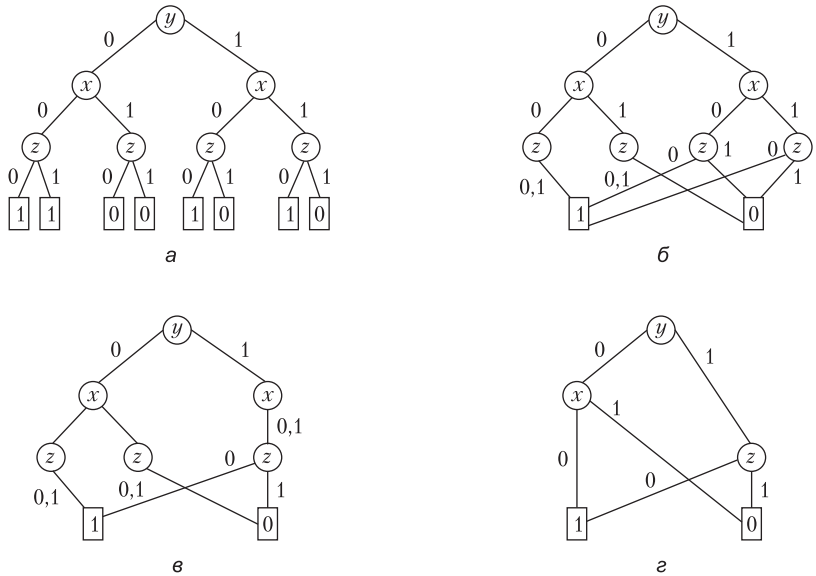


Рис. 3.7. Бинарная диаграмма решений при другом порядке переменных

производить вычисления быстрее, чем это в принципе возможно при использовании универсальных математических методов представления функций с помощью графиков (массивов) и формул (выражений). Советуем читателям обдумать этот факт.

3.6. Полные системы булевых функций

В типичной современной цифровой вычислительной машине цифрами являются 0 и 1. Следовательно, команды, которые выполняет процессор, суть булевы функции. Выше показано, что любая булева функция реализуется через конъюнкцию, дизъюнкцию и отрицание. Следовательно, можно построить нужный процессор, имея в распоряжении элементы, реализующие конъюнкцию, дизъюнкцию и отрицание. Этот раздел посвящён ответу на вопрос: существуют ли (и если существуют, то какие) иные системы булевых функций, обладающих тем свойством, что с их помощью можно выразить все другие функции.

3.6.1. Замкнутые классы

Пусть $F = \{f_1, \ldots, f_m\}$, $\forall\, i \in 1..m \;\; (f_i \in P_n)$. *Замыканием* F (обозначается $[F]$) называется множество всех булевых функций, реализуемых формулами над F:

$$[F] \stackrel{\text{Def}}{=} \{f \in P_n \mid f = \operatorname{func} \mathcal{F} \;\&\; \mathcal{F} \in \mathfrak{F}[F]\}\,.$$

Отметим следующие свойства замыкания (см. также 2.1.2).

1. $F \subset [F]$.
2. $[[F]] = [F]$.
3. $F_1 \subset F_2 \Longrightarrow [F_1] \subset [F_2]$.
4. $([F_1] \cup [F_2]) \subset [F_1 \cup F_2]$.

Класс (множество) функций F называется *замкнутым*, если $[F] = F$.
Рассмотрим следующие классы функций:

1. Класс функций, *сохраняющих* 0: $T_0 \stackrel{\text{Def}}{=} \{f \mid f(0,\ldots,0) = 0\}$.
2. Класс функций, *сохраняющих* 1: $T_1 \stackrel{\text{Def}}{=} \{f \mid f(1,\ldots,1) = 1\}$.
3. Класс *самодвойственных* функций: $T_* \stackrel{\text{Def}}{=} \{f \mid f = f^*\}$.
4. Класс *монотонных* функций: $T_\leqslant \stackrel{\text{Def}}{=} \{f \mid \alpha \leqslant \beta \Longrightarrow f(\alpha) \leqslant f(\beta)\}$,
   где $\alpha = (a_1,\ldots,a_n)$, $\beta = (b_1,\ldots,b_n)$, $a_i, b_i \in E_2$, $\alpha \leqslant \beta \stackrel{\text{Def}}{=} \forall\, i\ \ (a_i \leqslant b_i)$.
5. Класс *линейных* функций: $T_L \stackrel{\text{Def}}{=} \{f \mid f = c_0 + c_1x_1 + \cdots + c_nx_n\}$,
   где $+$ обозначает сложение по модулю 2, а знак конъюнкции опущен.

Примеры

1. Рассмотрим отрицание и введём обозначение $\varphi(x) := \overline{x}$. Имеем $\varphi \notin T_0$, так как $\varphi(0) = 1$, $\varphi \notin T_1$, так как $\varphi(1) = 0$, $\varphi \notin T_\leqslant$, так как $0 < 1$, но $\varphi(0) > \varphi(1)$. С другой стороны, $\varphi \in T_*$, так как $\varphi^*(x) = \overline{\varphi}(\overline{x}) = \neg\varphi(\neg x) = \neg\neg\overline{x} = \overline{x} = \varphi(x)$, и $\varphi \in T_L$, так как $\varphi(x) = x + 1$.
2. Рассмотрим конъюнкцию и введём обозначение $\psi(x,y) := x \wedge y$. Имеем: $\psi \in T_0$, так как $0 \wedge 0 = 0$, $\psi \in T_1$, так как $1 \wedge 1 = 1$, $\psi \in T_\leqslant$, так как $\psi(1,1) = 1$, и $\forall\,(a,b) \neq (1,1)\ \ ((a,b) \leqslant (1,1)\ \&\ \psi(a,b) = 0)$. С другой стороны, $\psi \notin T_*$, так как $\psi^*(x,y) = x \vee y$, и $\psi \notin T_L$. Действительно, от противного, пусть $\psi(x,y) = ax + by + c$. Тогда имеем: если $x = 0$ и $y = 0$, то $a0 + b0 + c = 0$, и, значит, $c = 0$; если $x = 0$ и $y = 1$, то $a0 + b1 + 0 = 0$, и, значит, $b = 0$; если $x = 1$ и $y = 0$, то $a1 + 0 \cdot 0 + 0 = 0$, и, значит, $a = 0$; если $x = 1$ и $y = 1$, то $0 \cdot 1 + 0 \cdot 1 + 0 = 1$, и, значит, $0 = 1$ — противоречие.

ТЕОРЕМА. *Классы $T_0, T_1, T_*, T_\leqslant, T_L$ замкнуты.*

Доказательство. Чтобы доказать, что некоторый класс F замкнут, достаточно показать, что если функция реализована в виде формулы над F, то она принадлежит F. Доказать, что произвольная формула обладает заданным свойством, можно с помощью индукции по структуре формулы (п. 3.3.3). База индукции очевидна: функции из F реализованы как тривиальные формулы над F. Таким образом, осталось обосновать индукционные переходы для пяти рассматриваемых классов.

[T_0] Пусть $f, f_1, \ldots, f_n \in T_0$ и $\Phi = f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n))$.
Тогда $\Phi(0,\ldots,0) = f(f_1(0,\ldots,0),\ldots,f_n(0,\ldots,0)) = f(0,\ldots,0) = 0$ и $\Phi \in T_0$.

[T_1] Пусть $f, f_1, \ldots, f_n \in T_1$ и $\Phi = f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n))$.
Тогда $\Phi(1,\ldots,1) = f(f_1(1,\ldots,1),\ldots,f_n(1,\ldots,1) = f(1,\ldots,1) = 1$ и $\Phi \in T_1$.

[T_*] Пусть $f, f_1, \ldots, f_n \in T_*$ и $\Phi = f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n))$.
Тогда $\Phi^* = f^*(f_1^*(x_1,\ldots,x_n),\ldots,f_n^*(x_1,\ldots,x_n)) =$
$= f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n)) = \Phi$ и $\Phi \in T_*$.

[$T_\leqslant$] Пусть $f, f_1, \ldots, f_n \in T_\leqslant$ и $\Phi = f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n))$.
Тогда $\alpha \leqslant \beta \Longrightarrow (f_1(\alpha),\ldots,f_n(\alpha)) \leqslant (f_1(\beta),\ldots,f_n(\beta)) \Longrightarrow$
$\Longrightarrow f(f_1(\alpha),\ldots,f_n(\alpha)) \leqslant f(f_1(\beta),\ldots,f_n(\beta)) \Longrightarrow \Phi(\alpha) \leqslant \Phi(\beta)$ и $\Phi \in T_\leqslant$.

[T_L] Пусть $f, f_1, \ldots, f_n \in T_L$ и $\Phi = f(f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n))$. Тогда
$f = c_0 + c_1x_1 + \ldots + c_nx_n$, $f_1 = c_0^1 + c_1^1x_1 + \ldots + c_n^1x_n$, $\ldots$, $f_n = c_0^n + c_1^nx_1 + \ldots + c_n^nx_n$. Подставим

эти формулы в формулу для Φ. Имеем $\Phi(x_1,\ldots,x_n)=c_0+c_1\left(c_0^1+c_1^1x_1+\ldots+c_n^1x_n\right)+\ldots+c_n\left(c_0^n+c_1^nx_1+\ldots+c_n^nx^n\right)=d_0+d_1x_1+\ldots+d_nx_n$. Следовательно, $\Phi\in T_L$. □

Пример. Таблица принадлежности булевых функций двух переменных рассмотренным выше замкнутым классам:

	T_0	T_1	T_*	$T_\leqslant$	T_L
0	+	−	−	+	+
$x\ \&\ y$	+	+	−	+	−
x	+	+	+	+	+
y	+	+	+	+	+
$x+_2 y$	+	−	−	−	+
$x\vee y$	+	+	−	+	−
$x\downarrow y$	−	−	−	−	−
$x\equiv y$	−	+	−	−	+
$\neg y$	−	−	+	−	+
$\neg x$	−	−	+	−	+
$x\to y$	−	+	−	−	−
$x\mid y$	−	−	−	−	−
1	−	+	−	+	+

Таким образом, рассмотренные классы $T_0, T_1, T_*, T_\leqslant, T_L$ попарно различны, не пусты и не совпадают с P_n.

ЗАМЕЧАНИЕ

Две функции, штрих Шеффера и стрелка Пирса, не принадлежат ни одному из рассматриваемых замкнутых классов.

3.6.2. Полные системы функций

Класс функций F называется *полным*, если его замыкание совпадает с P_n:

$$[F]=P_n.$$

Другими словами, множество функций F образует полную систему, если любая функция реализуема в виде формулы над F.

ТЕОРЕМА. *Пусть $F=\{f_1,\ldots,f_m\}$ и $G=\{g_1,\ldots,g_k\}$ — две системы функций. Тогда, если система F полна и все функции из F реализуемы формулами над G, то система G также полна:* $([F]=P_n\ \&\ \forall\, i\in 1..m\ \ (f_i=\operatorname{func}\mathcal{G}_i[G]))\Longrightarrow[G]=P_n$.

Доказательство. Пусть h — произвольная функция, $h\in P_n$. Тогда $[F]=P_n\Longrightarrow$ $\Longrightarrow h=\operatorname{func}\mathcal{F}[F]\Longrightarrow\mathcal{F}[\mathcal{G}_i/f_i]$ — формула над G. Следовательно, $h=\operatorname{func}\mathcal{G}[G]$. □

Примеры

Система $\{\vee,\wedge,\neg\}$ — полная (п. 3.4.3). Следовательно:

1) система $\{\neg,\wedge\}$ полная, так как $x_1\vee x_2=\neg(\neg x_1\wedge\neg x_2)$;
2) система $\{\neg,\vee\}$ полная, так как $x_1\wedge x_2=\neg(\neg x_1\vee\neg x_2)$;
3) система $\{|\}$ полная, так как $\neg x=x\mid x$, $x_1\wedge x_2=\neg(x_1\mid x_2)=(x_1\mid x_2)\mid(x_1\mid x_2)$.

Система $\{0, 1, \wedge, +\}$ полная, так как $\neg x = x + 1$ (здесь $+$ означает сложение по модулю 2). Представление булевой функции над базисом $\{0, 1, \wedge, +\}$ называется *полиномом Жегалкина*. Таким образом, всякая булева функция представима в виде

$$\sum_{(i_1,\ldots,i_s)} a_{i_1,\ldots,i_s} x_{i_1} \cdot \ldots \cdot x_{i_s},$$

где $\sum$ — сложение по модулю 2, знак $\cdot$ обозначает конъюнкцию и $a_{i_1,\ldots,i_s} \in E_2$.

ЗАМЕЧАНИЕ

Фактически, в полиноме Жегалкина $\sum a_{i_1,\ldots,i_s} x_{i_1} \cdot \ldots \cdot x_{i_s} = \sum x_{i_1} \cdots x_{i_s}$, поскольку если $a_{i_1,\ldots,i_s} = 1$, то этот коэффициент можно опустить, а если $a_{i_1,\ldots,i_s} = 0$, то можно опустить все слагаемое. Знак конъюнкции обычно также опускают.

Пример. $\neg x = x + 1$, $x \vee y = xy + x + y$, $x \equiv y = x + y + 1$.

ЗАМЕЧАНИЕ

Если в полиноме Жегалкина произвести следующие замены логических операций арифметическими: $x \wedge y \mapsto xy$; $x +_2 y \mapsto x + y - 2xy$, то получится арифметический полином, который можно упростить и использовать для вычисления значения булевой функции.

3.6.3. Полнота двойственной системы

ТЕОРЕМА. *Если система $F = \{f_1, \ldots, f_k\}$ полна, то система двойственных функций $F^* = \{f_1^*, \ldots, f_k^*\}$ также полна.*

Доказательство. Пусть h — произвольная функция, $h \in P_n$. Рассмотрим двойственную функцию h^*. Система F полна, так что $h^* = \operatorname{func} \mathcal{H}[F]$. По принципу двойственности $h = \operatorname{func} \mathcal{H}^*[F^*]$. □

Пример. Система $\{0, 1, \wedge, +\}$ полна, следовательно, система $\{1, 0, \vee, \equiv\}$ также полна.

3.6.4. Теорема Поста

Теорема Поста устанавливает необходимые и достаточные условия полноты системы булевых функций.

ТЕОРЕМА. *Система булевых функций F полна тогда и только тогда, когда она содержит хотя бы одну функцию, не сохраняющую нуль, хотя бы одну функцию, не сохраняющую единицу, хотя бы одну несамодвойственную функцию, хотя бы одну немонотонную функцию и хотя бы одну нелинейную функцию:*

$$[F] = P_n \iff \neg(F \subset T_0 \vee F \subset T_1 \vee F \subset T_* \vee F \subset T_\leqslant \vee F \subset T_L).$$

Доказательство.

[Необходимость] От противного. Пусть $[F] = P_n$ и

$$F \subset T_0 \vee F \subset T_1 \vee F \subset T_* \vee F \subset T_\leqslant \vee F \subset T_L.$$

Введем обозначение: i — один из индексов, 0, 1, $*$, $\leqslant$ или L.
Тогда $T_i = [T_i] \Longrightarrow [F] \subset T_i \Longrightarrow P_n \subset T_i \Longrightarrow P_n = T_i$, но $P_n \neq T_i$ по таблице из п. 3.6.1.

[Достаточность] Пусть $\neg(F \subset T_0 \vee F \subset T_1 \vee F \subset T_* \vee F \subset T_\leqslant \vee F \subset T_L)$. Тогда $\exists\, F' = \langle f_0, f_1, f_*, f_\leqslant, f_L\rangle$ $(f_0 \notin T_0 \;\&\; f_1 \notin T_1 \;\&\; f_* \notin T_* \;\&\; f_\leqslant \notin T_\leqslant \;\&\; f_L \notin T_L)$. Заметим, что функции $f_0, f_1, f_*, f_\leqslant, f_L$ не обязательно различны и не обязательно исчерпывают F. Покажем, что отрицание и конъюнкция реализуются в виде формул над F'. Тем самым теорема будет доказана (п. 3.6.2). Построение проводится в три этапа: на первом строятся формулы, реализующие константы 0 и 1, которые нужны на третьем этапе. На втором этапе строится формула, реализующая отрицание. На третьем этапе строится формула, реализующая конъюнкцию.

[Константы] Построим формулу, реализующую 1. Пусть $\varphi(x) := f_0(x, \ldots, x)$. Тогда $\varphi(0) = f_0(0, \ldots, 0) \neq 0 \Longrightarrow \varphi(0) = 1$. Возможны два случая: $\varphi(1) = 1$ или $\varphi(1) = 0$. Пусть $\varphi(1) = 1$. В этом случае формула φ реализует 1. Пусть $\varphi(1) = 0$. В этом случае формула φ реализует отрицание. Тогда рассмотрим функцию f_*. Имеем

$$f_* \notin T_* \Longrightarrow \exists\, a_1, \ldots, a_n \;\; \big(f_*(a_1, \ldots, a_n) \neq \overline{f_*}(\overline{a}_1, \ldots, \overline{a}_n)\big)\,.$$

Следовательно, $f_*(a_1, \ldots, a_n) = f_*(\overline{a}_1, \ldots, \overline{a}_n)$. Пусть теперь $\psi(x) := f_*(x^{a_1}, \ldots, x^{a_n})$. Тогда

$$\psi(0) = f_*(0^{a_1}, \ldots, 0^{a_n}) = f_*(\overline{a}_1, \ldots, \overline{a}_n) = f_*(a_1, \ldots, a_n) = f_*(1^{a_1}, \ldots, 1^{a_n}) = \psi(1).$$

Таким образом, $\psi(0) = \psi(1)$, откуда $\psi = 1$ или $\psi = 0$. Если $\psi = 1$, то требуемая константа 1 построена. В противном случае ψ реализует 0, и, значит, $\varphi(\psi(x)) = \overline{\psi}(x)$ реализует 1. Построение 0 аналогично, только вместо f_0 нужно использовать f_1.

[Отрицание] Рассмотрим функцию $f_\leqslant$. Имеем

$$f_\leqslant \notin T_\leqslant \Longrightarrow \exists\, \alpha = (a_1, \ldots, a_n), \beta = (b_1, \ldots, b_n) \;\; (\alpha \leqslant \beta \;\&\; f_\leqslant(\alpha) > f_\leqslant(\beta))\,.$$

Тогда $\alpha \leqslant \beta \Longrightarrow \forall\, i \;\; (a_i = b_i \vee a_i = 0 \;\&\; b_i = 1)$. Но

$$f_\leqslant(\alpha) \neq f_\leqslant(\beta) \Longrightarrow \alpha \neq \beta \Longrightarrow \exists\, J \subset 1..n \;\; (j \in J \Longrightarrow a_j = 0 \;\&\; b_j = 1)\,.$$

Другими словами, J — это множество индексов j, для которых $a_j \neq b_j$. Пусть $\varphi(x) := f_\leqslant(c_1, \ldots, c_n)$, где $c_j := x$, если $j \in J$, и $c_j := a_j (= b_j)$, если $j \notin J$. Тогда $\varphi(0) = f_\leqslant(c_1, \ldots, c_n)[0/x] = f_\leqslant(\alpha) > f_\leqslant(\beta) = f_\leqslant(c_1, \ldots, c_n)[1/x] = \varphi(1)$. Имеем $\varphi(0) > \varphi(1) \Longrightarrow \varphi(0) = 1 \;\&\; \varphi(1) = 0 \Longrightarrow \varphi(x) = \overline{x}$.

[Конъюнкция] Рассмотрим функцию f_L. Имеем

$$f_L \in P_n \Longrightarrow f_L = \sum_{i_1, \ldots, i_s} a_{a_{i_1}, \ldots, a_{i_s}} x_{i_1}, \ldots, x_{i_s}.$$

Но $f_L \notin T_L$, следовательно, в полиноме Жегалкина существует нелинейное слагаемое, содержащее конъюнкцию по крайней мере двух переменных. Пусть, для определённости, это x_1 и x_2. Тогда

$$f_L = x_1 \cdot x_2 \cdot f_a(x_3, \ldots, x_n) + x_1 \cdot f_b(x_3, \ldots, x_n) \;+\; + x_2 \cdot f_c(x_3, \ldots, x_n) + f_d(x_3, \ldots, x_n),$$

причём $f_a(x_3, \ldots, x_n) \neq 0$. Следовательно, $\exists\, a_3, \ldots, a_n \;\; (f_a(a_3, \ldots, a_n) = 1)$. Пусть $b := f_b(a_3, \ldots, a_n)$, $c := f_c(a_3, \ldots, a_n)$, $d := f_d(a_3, \ldots, a_n)$ и

$$\varphi(x_1, x_2) := f_L(x_1, x_2, a_3, \ldots, a_n) = x_1 \cdot x_2 + b \cdot x_1 + c \cdot x_2 + d.$$

Пусть далее $\psi(x_1, x_2) := \varphi(x_1 + c, x_2 + b) + b \cdot c + d$. Тогда

$$\begin{aligned}\psi(x_1, x_2) &= (x_1 + c) \cdot (x_2 + b) + b \cdot (x_1 + c) + c \cdot (x_2 + b) + d + b \cdot c + d = \\ &= x_1 \cdot x_2 + c \cdot x_2 + b \cdot x_1 + b \cdot c + b \cdot x_1 + b \cdot c + c \cdot x_2 + b \cdot c + d + b \cdot c + d = \\ &= x_1 \cdot x_2.\end{aligned}$$

Функции $x + a$ выразимы, так как $x + 1 = \overline{x}$, $x + 0 = x$, а константы 0, 1 и отрицание уже построены. □

Пример. В системе $\{\neg, \wedge\}$ отрицание не сохраняет констант и немонотонно, а конъюнкция несамодвойственна и нелинейна.

Глава 4 Логические исчисления

С древнейших времён человечеству известна логика, или искусство правильно рассуждать. Вообще говоря, способность к рассуждениям — это именно искусство. Имея какие-то утверждения (посылки), истинность которых проверена, скажем, на опыте, логик путём умозрительных построений приходит к другому утверждению (заключению), которое также оказывается истинным (в некоторых случаях). Опыт древних (чисто наблюдательный) был систематизирован Аристотелем. Он рассмотрел конкретные виды рассуждений, которые назвал *силлогизмами*. А именно, Аристотель рассмотрел так называемые *категорические утверждения* четырёх видов (A, B — *категории*):

1) все A обладают свойством B (все A суть B);
2) некоторые A обладают свойством B (некоторые A суть B);
3) все A не обладают свойством B (все A суть не B);
4) некоторые A не обладают свойством B (некоторые A суть не B) —

и зафиксировал все случаи, когда из посылок такого вида выводятся заключения одного из этих же видов.

Примеры

1. Все люди смертны. Сократ — человек. Следовательно, Сократ смертен. Это рассуждение правильно, потому что подходит под один из образцов силлогизмов Аристотеля.
2. Все дикари раскрашивают свои лица. Некоторые современные молодые люди раскрашивают свои лица. Следовательно, некоторые современные молодые люди — дикари. Это рассуждение неправильно, хотя, видимо, все входящие в него утверждения истинны.

Логика Аристотеля — это *классическая* логика, то есть наука, традиционно относящаяся к гуманитарному циклу и тем самым находящаяся вне рамок этого учебника.

Предметом этой главы являются некоторые элементы логики *математической*, которая соотносится с логикой классической примерно так, как язык Паскаль соотносится с английским языком. Эта аналогия довольно точна и по степени формализованности, и по широте применимости в реальной жизни, и по значимости для практического программирования. Математическая логика не является математической моделью классической логики, подобно тому как язык Паскаль не является моделью английского языка. Для математической логики характерна формализация логических операций, абстрагирование от конкретного содержания предложений, выражающих какое-либо суждение. Истинность или ложность вывода зависит только от его формы, но не от конкретного содержания составляющих его предложений.

План главы состоит в том, чтобы на основе небольшого предварительного рассмотрения ввести понятие «формальная теория», или «исчисление», в наиболее общем виде, а затем конкретизировать это понятие примерами двух наиболее часто используемых исчислений: исчисления высказываний и исчисления предикатов. Главу

завершают два раздела, демонстрирующие связь математической логики с программиированием: наивная теория алгоритмов и автоматическое доказательство теорем.

4.1. Логические связки и кванторы

Цель данного раздела — неформально ввести специфическую «логическую» терминологию и указать на её связь с материалом предшествующих глав. В последующих разделах главы определения этого раздела уточняются и конкретизируются.

4.1.1. Высказывания и связки

Элементами логических рассуждений являются утверждения, которые либо истинны, либо ложны, но не то и другое вместе. Такие утверждения называются *(простыми) высказываниями*. Простые высказывания обозначаются *пропозициональными переменными*. Пропозициональная переменная может принимать одно из двух *истинностных значений*, обозначаемых символами «T» и «F».

ЗАМЕЧАНИЕ

Истинностные значения обозначают различными способами: латинскими буквами T и F, как в этом учебнике, русскими буквами И и Л (от слов «Истина» и «Ложь»), цифрами 1 и 0, специальными значками $\top$ и $\bot$. Каждый из способов обозначения истинностных значений имеет свои достоинства и недостатки. Пара символов, T и F, выбрана нами, во-первых, для удобства чтения, во-вторых, так как эти символы не используются в учебнике в другом смысле, и, в-третьих, ради ассоциации с английскими словами «true» и «false», которые используются во многих языках программирования для обозначения истинностных значений.

Из простых высказываний с помощью *логических связок* могут быть построены составные высказывания. Обычно рассматривают следующие логические связки.

Название	Прочтение	Обозначение
Отрицание	не	$\neg$
Конъюнкция	и	$\&$
Дизъюнкция	или	$\vee$
Импликация	если... то	$\rightarrow$

4.1.2. Формулы

Правильно построенные составные высказывания называются *(пропозициональными) формулами*. Формулы имеют следующий синтаксис:

⟨формула⟩ ::= T | F |
⟨пропозициональная переменная⟩ |
(¬⟨формула⟩) |
(⟨формула⟩ & ⟨формула⟩) |
(⟨формула⟩ ∨ ⟨формула⟩) |
(⟨формула⟩ → ⟨формула⟩)

ЗАМЕЧАНИЕ

Для описания синтаксиса языка мы используем один из стандартных приёмов: контекстно-свободную порождающую грамматику в форме Бэкуса—Наура. Здесь названия синтаксических конструкций заключаются в угловые скобки, метасимвол «::=» означает «это», метасимвол «|» означает «или», все остальные символы означают сами себя. Исчерпывающее

изложение теории формальных грамматик применительно к программированию можно найти в книге [4].

Для упрощения записи вводится старшинство связок ($\neg$, $\&$, $\vee$, $\to$) и лишние скобки часто опускаются.

Истинностное значение формулы определяется через истинностные значения её составляющих в соответствии со следующей таблицей:

A	B	$\neg A$	$A \,\&\, B$	$A \vee B$	$A \to B$
F	F	T	F	F	T
F	T	T	F	T	T
T	F	F	F	T	F
T	T	F	T	T	T

4.1.3. Интерпретация

Пусть $A(x_1, \ldots, x_n)$ — пропозициональная формула, где $x_1, \ldots, x_n$ — входящие в неё пропозициональные переменные. Конкретный набор истинностных значений, приписанных переменным $x_1, \ldots, x_n$, называется *интерпретацией* формулы A. Формула может быть истинной (иметь значение T) при одной интерпретации и ложной (иметь значение F) при другой интерпретации. Значение формулы A в интерпретации I обозначим $I(A)$. Формула, истинная при некоторой интерпретации, называется *выполнимой*. Формула, истинная при всех возможных интерпретациях, называется *общезначимой* (или *тавтологией*). Формула, ложная при всех возможных интерпретациях, называется *невыполнимой* (или *противоречием*).

Примеры

$A \vee \neg A$ — тавтология, $A \,\&\, \neg A$ — противоречие, $A \to \neg A$ — выполнимая формула, она истинна при $I(A) = \text{F}$.

ТЕОРЕМА 1. *Пусть A — некоторая формула. Тогда:*

1) *если A — тавтология, то $\neg A$ — противоречие, и наоборот;*
2) *если A — противоречие, то $\neg A$ — тавтология, и наоборот;*
3) *если A — тавтология, то неверно, что A — противоречие, но не наоборот;*
4) *если A — противоречие, то неверно, что A — тавтология, но не наоборот;*
5) *если $\neg A$ выполнима, то неверно, что A — тавтология;*
6) *если A выполнима, то неверно, что A — противоречие.*

Доказательство. Очевидно из определений. □

ТЕОРЕМА 2. *Если формулы A и $A \to B$ — тавтологии, то формула B — тавтология.*

Доказательство. От противного. Пусть $I(B) = \text{F}$. Но $I(A) = \text{T}$, так как A — тавтология. Значит, $I(A \to B) = \text{F}$, что противоречит предположению о том, что $A \to B$ — тавтология. □

4.1.4. Логическое следование и логическая эквивалентность

Говорят, что формула B *логически следует* из формулы A (обозначение $A \Longrightarrow B$), если формула B имеет значение T при всех интерпретациях, при которых формула A имеет значение T. Говорят, что формулы A и B *логически эквивалентны* (обозначается $A \Longleftrightarrow B$ или просто $A = B$), если они являются логическими следствиями друг друга. Очевидно, что логически эквивалентные формулы имеют одинаковые значения при любой интерпретации.

ТЕОРЕМА 1. $P \to Q = \neg P \vee Q$.

Доказательство. Для доказательства достаточно проверить, что формулы действительно имеют одинаковые истинностные значения при всех интерпретациях.

P	Q	$P \to Q$	$\neg P$	$\neg P \vee Q$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	F
T	T	T	F	T

□

ТЕОРЕМА 2. *Если A, B, C — любые формулы, то имеют место следующие логические эквивалентности:*

1. $A \vee A = A, A \,\&\, A = A$.
2. $A \vee B = B \vee A, A \,\&\, B = B \,\&\, A$.
3. $A \vee (B \vee C) = (A \vee B) \vee C, A \,\&\, (B \,\&\, C) = (A \,\&\, B) \,\&\, C$.
4. $A \vee (B \,\&\, C) = (A \vee B) \,\&\, (A \vee C), A \,\&\, (B \vee C) = (A \,\&\, B) \vee (A \,\&\, C)$.
5. $(A \,\&\, B) \vee A = A, (A \vee B) \,\&\, A = A$.
6. $A \vee \mathrm{F} = A, A \,\&\, \mathrm{F} = \mathrm{F}$.
7. $A \vee \mathrm{T} = \mathrm{T}, A \,\&\, \mathrm{T} = A$.
8. $\neg(\neg A) = A$.
9. $\neg(A \,\&\, B) = \neg A \vee \neg B, \neg(A \vee B) = \neg A \,\&\, \neg B$.
10. $A \vee \neg A = \mathrm{T}, A \,\&\, \neg A = \mathrm{F}$.

Доказательство. Проверяется построением таблиц истинности. □

ЗАМЕЧАНИЕ

Таким образом, алгебра $\langle\{\mathrm{T,F}\}; \vee, \&, \neg\rangle$ является булевой алгеброй, которая называется *алгеброй высказываний.*

ТЕОРЕМА 3. $P_1 \,\&\, \ldots \,\&\, P_n \Longrightarrow Q$ *тогда и только тогда, когда* $(P_1 \,\&\, \ldots \,\&\, P_n) \to Q$ — *тавтология.*

Доказательство.

[$\Longrightarrow$] Пусть $I(P_1 \,\&\ldots\&\, P_n) = \mathrm{T}$. Тогда $I(Q) = \mathrm{T}$ и $I(P_1 \,\&\ldots\&\, P_n \to Q) = \mathrm{T}$. Пусть $I(P_1 \,\&\, \ldots \,\&\, P_n) = \mathrm{F}$. Тогда $I(P_1 \,\&\, \ldots \,\&\, P_n \to Q) = \mathrm{T}$ при любой интерпретации I. Таким образом, формула $P_1 \& \ldots \& P_n \to Q$ общезначима.

[$\Longleftarrow$] Пусть $I(P_1 \,\&\, \ldots \,\&\, P_n) = \mathrm{T}$. Тогда $I(Q) = \mathrm{T}$, иначе формула $P_1 \,\&\, \ldots \,\&\, P_n \to Q$ не была бы тавтологией. Таким образом, формула Q — логическое следствие формулы $P_1 \,\&\, \ldots \,\&\, P_n$. □

ТЕОРЕМА 4. *$P_1 \,\&\, \ldots \,\&\, P_n \Longrightarrow Q$ тогда и только тогда, когда $P_1 \,\&\, \ldots \,\&\, P_n \,\&\neg Q$ — противоречие.*

Доказательство. По предыдущей теореме $P_1 \,\&\, \ldots \,\&\, P_n \Longrightarrow Q$ тогда и только тогда, когда формула $P_1 \,\&\, \ldots \,\&\, P_n \to Q$ — тавтология. По первой теореме п. 4.1.3 формула $P_1 \,\&\, \ldots \,\&\, P_n \to Q$ является тавтологией тогда и только тогда, когда формула $\neg(P_1 \,\&\, \ldots \,\&\, P_n \to Q)$ является противоречием. Имеем
$\neg(P_1 \,\&\, \ldots \,\&\, P_n \to Q) = \neg(\neg(P_1 \,\&\, \ldots \,\&\, P_n) \vee Q) = \neg\neg(P_1 \,\&\, \ldots \,\&\, P_n) \,\&\, \neg Q =$
$= P_1 \,\&\, \ldots \,\&\, P_n \,\&\, \neg Q$. □

4.1.5. Подстановка и замена

Пусть A — формула, в которую входит переменная x (обозначается $A(\ldots x \ldots)$) или входит некоторая подформула B (обозначается $A(\ldots B \ldots)$), и пусть C — некоторая формула. Тогда $A(\ldots x \ldots)[C/x]$ обозначает формулу, полученную из формулы A подстановкой формулы C вместо *всех* вхождений переменной x, а $A(\ldots B \ldots)[C//B]$ обозначает любую формулу, полученную из формулы A подстановкой формулы C вместо *некоторых* (в частности, вместо одного) вхождений подформулы B.

ТЕОРЕМА 1. *Если $A(\ldots x \ldots)$ — тавтология, а B — любая формула, то формула $A(\ldots x \ldots)[B/x]$ — тавтология.*

Доказательство. Пусть $C := A(\ldots x \ldots)[B/x]$. Пусть I — интерпретация формулы C (формула уже не содержит переменной x). Построим интерпретацию I' формулы A, положив $x := I(B)$, а значения остальных переменных взяв такими же, как в интерпретации I. Тогда $I'(A) = I(C)$, но $I'(A) = \mathrm{T}$, следовательно, $I(C) = \mathrm{T}$. □

ТЕОРЕМА 2. *Если $A(\ldots B \ldots)$ и $B = C$, а $D := A(\ldots B \ldots)[C//B]$, то $A = D$.*

Доказательство. Пусть I — любая интерпретация, содержащая значения для всех переменных в формулах A, B и C. Тогда $I(B) = I(C)$, значит, $I(A) = I(D)$. □

Теоремы 1 и 2 аналогичны правилам подстановки и замены, изложенным в п. 3.2.3.

4.1.6. Кванторы

Очень важным элементом языка математической логики является понятие «предикат». *Одноместным предикатом* P (над множеством M) называется функция $P\colon M \to \{\mathrm{F}, \mathrm{T}\}$ (см. также п. 1.9.4). Если $x \in M$, то выражение $P(x)$ обладает истинностным значением, и потому может рассматриваться как простое высказывание. Аналогично *n-местным предикатом* $P(x_1, \ldots, x_n)$ называется функция $P\colon M_1 \times \cdots \times M_n \to \{\mathrm{F}, \mathrm{T}\}$, причём $x_i \in M_i$. Поскольку прямое произведение множеств также является множеством, далее в этом параграфе в примерах и определениях рассматриваются для краткости только одноместные предикаты, для многоместных предикатов всё аналогично.

Примеры

Высказывание «число пять больше числа два» имеет истинностное значение T и является простым высказыванием, но не является предикатом, поскольку не содержит переменных. Высказывание «число x больше двух» является одноместным предикатом и символически записывается $x > 2$, а высказывание «число x больше числа y» является двухместным предикатом и символически записывается $x > y$.

ЗАМЕЧАНИЕ

Одноместный предикат соответствует характеристической функции подмножества (п. 1.9.2), а n-местный предикат соответствует характеристической функции отношения.

Операция связывания квантором всеобщности сопоставляет предикату P, определённому на множестве M, высказывание, обозначаемое $\forall x \in M \ (P(x))$, истинное в том и только в том случае, когда высказывание $P(x)$ имеет значение T для всех $x \in M$, и ложное в противном случае. Высказывание $\forall x \in M \ (P(x))$ называется *универсальным высказыванием* для предиката P.

ЗАМЕЧАНИЕ

Если множество M подразумевается, то при записи его опускают: $\forall x \ (P(x))$.

Символ $\forall x$ называется *квантором всеобщности* по переменной x.

Примеры

Предикат $x < 10$, определённый на множестве $\mathbb{R}$, не является тождественно истинным, поэтому высказывание $\forall x \in \mathbb{R} \ (x < 10)$ имеет значение F. Предикат $x^2 \geqslant 0$, определённый на множестве $\mathbb{R}$, является тождественно истинным на множестве $\mathbb{R}$, поэтому высказывание $\forall x \in \mathbb{R} \ \left(x^2 \geqslant 0\right)$ имеет значение T.

Если предикат $P(x)$ задан на конечном множестве M, то операция связывания квантором всеобщности выражается через конъюнкцию:

$$M = \{a_1, \ldots, a_n\} \ \& \ \forall x \in M \ (P(x)) \Longleftrightarrow P(a_1) \ \& \ P(a_2) \ \& \ \ldots \ \& \ P(a_n).$$

Операция связывания квантором существования сопоставляет предикату P, определённому на множестве M, высказывание, обозначаемое $\exists x \in M \ (P(x))$, истинное в том и только в том случае, когда высказывание $P(x)$ имеет значение T хотя бы для одного $x \in M$, и ложное в противном случае. Высказывание $\exists x \in M \ (P(x))$ называется *экзистенциальным высказыванием* для предиката P.

ЗАМЕЧАНИЕ

Если множество M подразумевается, то при записи его опускают: $\exists x \ (P(x))$.

Символ $\exists x$ называется *квантором существования* по переменной x.

Примеры

Предикат $x = x + 1$, определенный на множестве $\mathbb{N}$, является тождественно ложным, поэтому высказывание $\exists x \in \mathbb{N} \ (x = x + 1)$ имеет значение F. В то же время высказывание $\exists x \in \mathbb{N} \ (x + 1 > 10)$ имеет значение T, так как существует значение $x \in \mathbb{N}$, например 10, на котором предикат $x + 1 > 10$, определённый на множестве $\mathbb{N}$, имеет значение T.

Если предикат $P(x)$ задан на конечном множестве M, то операция связывания квантором существования выражается через дизъюнкцию:

$$M = \{a_1, \ldots, a_n\} \;\&\; \exists\, x \in M \;\; (P(x)) \Longleftrightarrow P(a_1) \vee P(a_2) \vee \ldots \vee P(a_n).$$

Иногда используется модификация квантора существования, которая называется квантором существования и единственности. *Операция связывания квантором существования и единственности* сопоставляет предикату P, определенному на множестве M, высказывание, обозначаемое $\exists!\, x \in M\,(P(x))$, истинное в том и только в том случае, когда предикат $P(x)$ имеет истинностное значение T в точности для одного элемента $x \in M$ и ложное в противном случае.

ЗАМЕЧАНИЕ

Если множество M подразумевается, то при записи его опускают: $\exists!\, x\,(P(x))$.

Символ $\exists!\, x$ называется *квантором существования и единственности* по переменной x.

Примеры

Высказывание $\exists!\, x \in \mathbb{N}\,(x+1=10)$ имеет значение T, так как существует единственное значение $9 \in \mathbb{N}$, на котором предикат $x+1=10$, определенный на множестве $\mathbb{N}$, имеет значение T.

Квантор существования и единственности является не более чем «синтаксическим сахаром», поскольку выражается через кванторы существования, всеобщности и проверку равенства элементов:

$$\exists!\, x \in M\,(P(x)) \Longleftrightarrow \exists\, x \in M \;\; (P(x)) \;\&\; \forall\, x, y \in M \;\; (P(x) \;\&\; P(y) \to x = y)\,.$$

Взаимосвязь между универсальными и экзистенциальными высказываниями удобно представлять в виде *логического квадрата*, изображённого на рисунке 4.1.

Универсальные высказывания $\forall\, x \;\; (P(x)$ и $\forall\, x \;\; (\neg P(x))$, стоящие в двух верхних вершинах квадрата, не могут быть одновременно истинными. Такие высказывания называются *контрарными*. Экзистенциальные высказывания $\exists\, x \;\; (P(x))$ и $\exists\, x \;\; (\neg(P(x))$, стоящие в двух нижних вершинах квадрата, не могут быть одновременно ложными. Такие высказывания называются *субконтрарными*. Высказывания, соединенные диагоналями квадрата, являются отрицаниями друг друга. Такие высказывания называются *контрадикторными*. Под каждым универсальным высказыванием стоит высказывание, логически следующее из него.

ЗАМЕЧАНИЕ

Контрарные высказывания могут быть одновременно ложными, субконтрарные высказывания могут быть одновременно истинными.

ОТСТУПЛЕНИЕ

Язык логических связок и кванторов очень прост. Однако перевести текст на естественном языке в этот язык не всегда так уж просто.

Рассмотрим текст на естественном языке — начальный фрагмент из замечательной книги Д. Гильберта «Основания геометрии» (с небольшими сокращениями для экономии места и с упрощением форматирования).

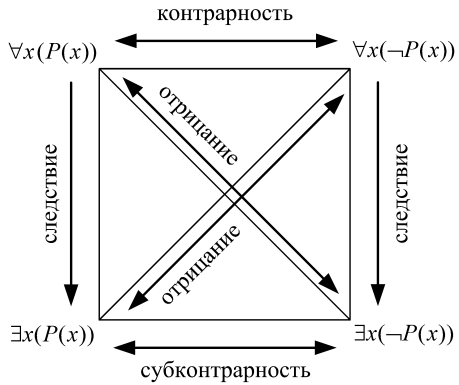


Рис. 4.1. Логический квадрат

Первая группа аксиом: аксиомы соединения (принадлежности).
Аксиомы этой группы устанавливают отношения принадлежности между введёнными выше вещами — точками, прямыми и плоскостями — и гласят следующим образом.
I_1. Для любых двух точек A, B существует прямая a, принадлежащая каждой из этих двух точек A, B.
I_2. Для двух точек A, B существует не более одной прямой, принадлежащей каждой из точек A, B.
Здесь, как и в последующем, под двумя, тремя, ... точками (прямыми, плоскостями) всегда подразумеваются различные точки (прямые, плоскости). Вместо термина «принадлежат» мы будем пользоваться также и другими формулировками. Например, вместо «прямая a принадлежит каждой из точек A и B», мы будем говорить: «прямая a проходит через точки A и B» или «прямая a соединяет точки A и B»; ...
I_3. На прямой существуют по крайней мере две точки. Существуют по крайней мере три точки, не лежащие на одной прямой.

Здесь области изменения и обозначения предметных переменных заданы. Обозначим P — множество точек, L — множество прямых. Возможны различные варианты введения предикатов. Если ввести трехместный предикат

$$R\colon P \times P \times L \to \{\mathrm{T}, \mathrm{F}\},$$

означающий принадлежность двух точек одной прямой, с ограничением, что два первых аргумента суть различные точки, то первые две аксиомы запишутся в виде одной простой формулы

$$\forall A\, \forall B\, \exists!\, a\ R(A, B, a),$$

а третья аксиома запишется в виде двух формул:

$$\forall a\, \exists A\, \exists B\ R(A, B, a) \quad \text{и} \quad \exists A\, \exists B\, \exists C\, \forall a\, (\neg\, (R(A, B, a)\ \&\ R(B, C, a)\ \&\ R(C, A, a)))\,.$$

Вторая часть третьей аксиомы позволяет определить новый предикат:

$$S(A, B, C) := \forall a\, (\neg\, (R(A, B, a)\ \&\ R(B, C, a)\ \&\ R(C, A, a)))\,,$$

означающий, что три точки не лежат на одной прямой. В таком случае запись третьей аксиомы ещё более упростится:

$$\forall a\, \exists A\, \exists B\ R(A, B, a) \quad \text{и} \quad \exists A\, \exists B\, \exists C\ S(A, B, C).$$

Такой путь сокращает количество вхождений предикатов в формулы, но может затруднить понимание формул, поскольку предикаты получаются достаточно сложные и сопровождаются оговорками про различные значения переменных.

Можно пойти по другому пути, определив двуместный предикат $\in_{PL}: P \times L \to \{\mathrm{T}, \mathrm{F}\}$, означающий, что точка лежит на прямой, явный предикат равенства точек $=_P: P \times P \to \{\mathrm{T}, \mathrm{F}\}$ и предикат различия точек: $\neq_P (A, B) := \neg(=_P (A, B))$. Аналогично можно ввести предикаты равенства и неравенства прямых: $=_L: L \times L \to \{\mathrm{T}, \mathrm{F}\}$, $\neq_L (A, B) := \neg(=_L (A, B))$. Поскольку предикаты двухместные, можно использовать инфиксную запись, а индексы у значков опускать, поскольку эти индексы указывают на типы аргументов, которые можно восстановить из контекста.

В этом случае аксиомы запишутся более длинными, но, может быть, более понятными формулами:

$$\forall A, B \ (A \neq B \to \exists a \ (A \in a \ \& \ B \in a)),$$
$$\forall A, B \ (\forall a, b \ (A \neq B \ \& \ A \in a \ \& \ B \in a \ \& \ A \in b \ \& \ B \in b \to a = b)),$$
$$\forall a \ (\exists A, B \ (A \in a \ \& \ B \in a)),$$
$$\exists A, B, C \ (\neg\exists a \ (A \in a \ \& \ B \in a \ \& \ C \in a)).$$

Таким образом, при переводе высказываний с естественного языка на язык математической логики большое значение имеет выбор системы предикатов, который ни в коей мере не является однозначным и предопределённым, а всегда зависит от целей, ради которых проводится *формализация*.

4.2. Формальные теории

Исторически понятие формальной теории было разработано в период интенсивных исследований в области оснований математики для формализации собственно логики и теории доказательства. Сейчас этот аппарат широко используется при создании специальных исчислений для решения конкретных прикладных задач. Рамки книги не позволяют привести развёрнутые примеры подобных специальных исчислений (они довольно велики), но тем не менее такие примеры существуют.

4.2.1. Определение формальной теории

Формальная теория $\mathcal{T}$ имеет следующие составляющие:

1) множество $\mathcal{A}$ символов, образующих *алфавит*;
2) множество $\mathcal{F}$ слов в алфавите $\mathcal{A}$, $\mathcal{F} \subset \mathcal{A}^*$, которые называются *формулами*;
3) подмножество $\mathcal{B}$ формул, $\mathcal{B} \subset \mathcal{F}$, которые называются *аксиомами*;
4) множество $\mathcal{R}$ отношений R на множестве формул, $R \in \mathcal{R}$, $R \subset \mathcal{F}^{n+1}$, которые называются *правилами вывода*.

Множество символов $\mathcal{A}$ может быть конечным или бесконечным. Обычно для образования множества символов $\mathcal{A}$ используют конечное множество букв, к которым, если нужно, приписываются в качестве индексов натуральные числа.

Множество формул $\mathcal{F}$ обычно задается индуктивным определением, например, с помощью порождающей формальной грамматики. Как правило, это множество бесконечно. Множества $\mathcal{A}$ и $\mathcal{F}$ в совокупности определяют *язык*, или *сигнатуру*, формальной теории.

Множество аксиом $\mathcal{B}$ может быть конечным или бесконечным. Если множество аксиом бесконечно, то обычно оно задаётся с помощью конечного множества *схем* аксиом и правил порождения конкретных аксиом из схемы аксиом. Часто аксиомы делятся на два вида: *логические* аксиомы (общие для целого класса формальных теорий) и *нелогические* (или *собственные*) аксиомы (определяющие специфику и содержание конкретной теории).

Множество правил вывода $\mathcal{R}$ чаще всего конечно.

4.2.2. Выводимость

Пусть $F_1, \ldots, F_n, G$ — формулы теории $\mathfrak{T}$, то есть $F_1, \ldots, F_n, G \in \mathcal{F}$. Если существует такое правило вывода R, $R \in \mathcal{R}$, что $(F_1, \ldots, F_n, G) \in R$, то говорят, что формула G *непосредственно выводима* из формул $F_1, \ldots, F_n$ по правилу вывода R. Обычно этот факт записывают следующим образом:

$$\frac{F_1, \ldots, F_n}{G}\,(R).$$

Здесь формулы $F_1, \ldots, F_n$ называются *посылками*, формула G — *заключением*, а R является обозначением правила вывода.

ЗАМЕЧАНИЕ

Обозначение правила вывода справа от черты, разделяющей посылки и заключение, часто опускают, если оно ясно из контекста.

Выводом формулы G из формул $F_1, \ldots, F_n$ в формальной теории $\mathfrak{T}$ называется такая последовательность формул $E_1, \ldots, E_k$, что $E_k = G$, и любая формула E_i ($i \leqslant k$) является либо аксиомой ($E_i \in \mathcal{B}$), либо исходной формулой F_j ($E_i = F_j$), либо непосредственно выводима из ранее полученных формул $E_{j_1}, \ldots, E_{j_n}$ ($j_1, \ldots, j_n < i$).

Если в теории $\mathfrak{T}$ существует вывод формулы G из формул $F_1, \ldots, F_n$, то это записывают следующим образом: $F_1, \ldots, F_n \vdash_{\mathfrak{T}} G$. Формулы $F_1, \ldots, F_n$ называются *гипотезами* вывода. Если теория $\mathfrak{T}$ подразумевается, то её обозначение обычно опускают.

Если $\vdash_{\mathfrak{T}} G$, то формула G называется *теоремой* теории $\mathfrak{T}$ (то есть теорема — это формула, выводимая только из аксиом, без гипотез).

Если $\Gamma \vdash_{\mathfrak{T}} G$, то очевидно, что $\Gamma, \Delta \vdash_{\mathfrak{T}} G$, где Γ и Δ — любые множества формул (то есть при добавлении лишних гипотез выводимость сохраняется).

ЗАМЕЧАНИЕ

При изучении формальных теорий нужно различать теоремы *самой* формальной теории и теоремы *об этой* формальной теории, или *метатеоремы*. Это различие иногда не формализуется явно, но всегда является существенным. В этой главе теоремы конкретной формальной теории, как правило, записываются в виде формул, составленных из специальных знаков, а метатеоремы формулируются на естественном языке, чтобы их легче было отличать от теорем самой формальной теории.

4.2.3. Интерпретация

В этом параграфе вводится более общее и строгое определение понятия интерпретации по сравнению с п. 4.1.2. *Интерпретацией* формальной теории $\mathfrak{T}$ в область интерпретации M называется функция $I\colon \mathcal{F} \to M$, которая каждой формуле формальной теории $\mathfrak{T}$ однозначно сопоставляет некоторое содержательное высказывание относительно объектов множества (алгебраической системы) M. Это высказывание может быть истинным или ложным (или не иметь истинностного значения). Если соответствующее высказывание является истинным, то говорят, что формула *выполняется* в данной интерпретации.

ОТСТУПЛЕНИЕ

В конечном счёте нас интересуют такие формальные теории, которые описывают какие-то реальные объекты и связи между ними. Речь идёт прежде всего о математических объектах и математических теориях, которые выбираются в качестве области интерпретации.

Интерпретация I называется *моделью* множества формул Γ, если все формулы этого множества выполняются в интерпретации I. Интерпретация I называется *моделью* формальной теории $\mathfrak{T}$, если все теоремы этой теории выполняются в интерпретации I (то есть все выводимые формулы оказываются истинными в данной интерпретации).

4.2.4. Общезначимость и непротиворечивость

Формула называется *общезначимой* (или *тавтологией*), если она истинна в любой интерпретации. Формула называется *противоречивой*, если она ложна в любой интерпретации.

Формула G называется *логическим следствием* множества формул Γ, если G выполняется в любой модели Γ.

Формальная теория $\mathfrak{T}$ называется *семантически непротиворечивой*, если ни одна её теорема не является противоречием. Таким образом, формальная теория пригодна для описания тех множеств (алгебраических систем), которые являются её моделями. Модель для формальной теории $\mathfrak{T}$ существует тогда и только тогда, когда $\mathfrak{T}$ семантически непротиворечива.

Формальная теория $\mathfrak{T}$ называется *формально непротиворечивой*, если в ней не являются выводимыми одновременно формулы F и $\neg F$. Теория $\mathfrak{T}$ формально непротиворечива тогда и только тогда, когда она семантически непротиворечива.

ЗАМЕЧАНИЕ

Последние утверждения являются доказуемыми метатеоремами, доказательства которых опускаются из-за технических сложностей.

4.2.5. Полнота, независимость и разрешимость

Пусть интерпретация $I\colon \mathfrak{T} \to M$ является моделью формальной теории $\mathfrak{T}$. Тогда формальная теория $\mathfrak{T}$ называется *полной* (или *адекватной*), если каждому истинному высказыванию области интерпретации M соответствует теорема теории $\mathfrak{T}$.

Если для множества (алгебраической системы) M существует формальная полная непротиворечивая теория $\mathfrak{T}$, то система M называется *аксиоматизируемой* (или *формализуемой*).

Система аксиом (или аксиоматизация) формально непротиворечивой теории $\mathfrak{T}$ называется *независимой*, если никакая из аксиом не выводима из остальных по правилам вывода теории $\mathfrak{T}$.

Формальная теория $\mathfrak{T}$ называется *разрешимой*, если существует алгоритм, который для любой формулы теории определяет, является ли эта формула теоремой теории. Формальная теория $\mathfrak{T}$ называется *полуразрешимой*, если существует алгоритм, который для любой формулы F теории выдает ответ «ДА», если F является теоремой теории, и, может быть, не выдает никакого ответа, если F не является теоремой (то есть алгоритм применим не ко всем формулам).

4.3. Исчисление высказываний

В этом разделе дано описание формальной теории исчисления высказываний. Поскольку исчисление высказываний уже знакомо читателю по материалам разделов 3.1 и 4.1, здесь сделан сильный акцент именно на *формальной* технике. Знакомство с чисто механическим характером формальных выводов подготавливает рассмотрение методов автоматического доказательства теорем в разделе 4.6.

4.3.1. Классическое определение исчисления высказываний

Исчисление высказываний — это формальная теория $\mathcal{L}$, в которой:

1. Алфавит: $\neg$ и $\to$ — *связки*;
   (,) — служебные символы;
   $a, b, \ldots, a_1, b_1, \ldots$ — *пропозициональные переменные*.
2. Формулы: 1) переменные суть формулы;
   2) если A, B — формулы, то $(\neg A)$ и $(A \to B)$ — формулы.
3. Аксиомы: $A_1 : (A \to (B \to A))$;
   $A_2 : ((A \to (B \to C)) \to ((A \to B) \to (A \to C)))$;
   $A_3 : ((\neg B \to \neg A) \to ((\neg B \to A) \to B))$.
4. Правило: $\dfrac{A,\ A \to B}{B}$ (Modus ponens).

Здесь A, B и C — *любые* формулы. Таким образом, множество аксиом теории $\mathcal{L}$ бесконечно, хотя задано тремя *схемами* аксиом. Множество правил вывода также бесконечно, хотя оно задано только одной *схемой*.

ЗАМЕЧАНИЕ

Латинское словосочетание *Modus ponens* обычно переводят на русский как *правило отделения* и обозначают MP.

При записи формул лишние скобки опускаются, если это не вызывает недоразумений. Другие связки вводятся определениями:

$$A \,\&\, B \stackrel{\text{Def}}{=} \neg(A \to \neg B), \qquad A \vee B \stackrel{\text{Def}}{=} \neg A \to B.$$

Любая формула, содержащая эти связки, рассматривается как синтаксическое сокращение собственной формулы теории $\mathcal{L}$.

ОТСТУПЛЕНИЕ

Классическое задание теории $\mathcal{L}$ с помощью схем аксиом над любыми формулами вполне соответствует математической традиции (мы пишем $(a+b)^2 = a^2+2ab+b^2$, подразумевая под a и b не только любые числа, но и любые выражения!). В то же время это не очень удобно для представления формальных теорий в программах. В следующих трёх параграфах вводится определение исчисления высказываний, более удобное для программной реализации.

4.3.2. Частный случай формулы

Если в формулу A вместо переменных $x_1, \ldots, x_n$ подставить формулы $B_1, \ldots, B_n$, то получится формула B, которая называется *частным случаем* формулы A:

$$B \stackrel{\text{Def}}{=} A(\ldots, x_i, \ldots)\,[B_i/x_i]_{i=1}^{n}\,.$$

Каждая формула B_i подставляется вместо *всех* вхождений переменной x_i. Набор подстановок $[B_i/x_i]_{i=1}^n$ называется *унификатором*. Формула C называется *совместным* частным случаем формул A и B, если C является частным случаем формулы A и одновременно частным случаем формулы B при одном и том же наборе подстановок, то есть $C = A(\dots, x_i, \dots)\,[X_i/x_i]_{i=1}^n$ и $C = B(\dots, x_i, \dots)\,[X_i/x_i]_{i=1}^n$. Формулы, которые имеют совместный частный случай, называются *унифицируемыми*, а набор подстановок $[X_i/x_i]_{i=1}^n$, с помощью которого получается совместный частный случай унифицируемых формул, называется *общим* унификатором. Наименьший возможный унификатор называется *наиболее общим унификатором*.

Более точно, σ называется наиболее общим унификатором, если: 1) σ является унификатором; 2) для любого другого унификатора δ существует такая подстановка λ, что унификатор δ выражается через суперпозицию σ и λ, $\delta = \sigma \circ \lambda$ (см. пример 2 в п. 2.2.3).

Примеры

1. Формулы: $\neg a \to y, x \to \neg b$. Унификатор $[\neg a/x, \neg b/y]$. Совместный частный случай: $\neg a \to \neg b$.
2. Формулы: $\neg a \to y, x \to \neg b$. Унификатор $[z/a, \neg w/y, \neg z/x, w/b]$. Совместный частный случай $\neg z \to \neg w$, но при этом $[z/a, \neg w/y, \neg z/x, w/b] =$ $= [\neg a/x, \neg b/y] \circ [z/a, w/b]$.
3. Формулы: $\neg a \to x, x \to \neg b$. Формулы не унифицируемы, так как нужно одновременно $[\neg a/x]$ и $[\neg b/x]$, что невозможно.
4. Формулы: $x, \neg x$, а также $x, x \to y$ или $x, y \to x$ и подобные им не унифицируемы, потому что конечное дерево не может быть изоморфно своему собственному поддереву.

ЗАМЕЧАНИЕ

Хотя подстановка вида $[f(x)/x]$ синтаксически допустима, она никогда не может появиться в унификаторе, поскольку конечное дерево не может быть изоморфно своему собственному поддереву, а потому такие подстановки можно не рассматривать.

Набор формул $B_1, \dots, B_n$ называется *частным случаем набора формул* $A_1, \dots, A_n$, если каждая формула B_i является частным случаем формулы A_i при одном и том же наборе подстановок. Набор формул $C_1, \dots, C_n$ называется *совместным* частным случаем наборов формул $A_1, \dots, A_n$ и $B_1, \dots, B_n$, если каждая формула C_i является частным случаем формул A_i и B_i при одном и том же наборе подстановок.

4.3.3. Алгоритмы унификации

Сначала рассмотрим общий алгоритм унификации, предложенный Робинсоном вместе с методом резолюций (см. раздел 4.6). В этом алгоритме унифицируются два терма (п. 2.1.3). При этом ничего не предполагается относительно сигнатуры функциональных символов, входящих в термы. Предполагается только, что можно отличить функциональные символы от символов переменных и символов констант. Также считается, что задана функция firstNeqSubTerms, которая отыскивает в заданных термах самые левые вхождения несовпадающих подтермов.

Алгоритм 4.1. Оригинальный алгоритм Робинсона

Вход: формулы A и B в термальной форме.
Выход: **false**, если формулы не унифицируемы; **true** и наиболее общий унификатор σ в противном случае.

$\sigma := \varnothing$
while $A \neq B$ **do**
  $[A_1, B_1] := \text{firstNeqSubTerms}\,(A, B)$
  if $\neg\,([A_1, B_1] \sim [v, t]\ \&\ v \notin t)$ **then**
    return false //формулы не унифицируемы
  end if
  $\sigma := \sigma \circ [t/v]; A := A[t/v]; B := B[t/v]$
end while
return true

ЗАМЕЧАНИЕ

Знак $\sim$ в этом алгоритме означает, что та пара подтермов, которую возвращает функция firstNeqSubTerms, образует подстановку, то есть состоит из переменной v и некоторого терма t. При этом дополнительно накладывается условие, что $v \notin t$ (см. замечание на предыдущем слайде).

Пример. В примере рассматривается унификация двух термов, при этом несовпадающие подтермы, найденные функцией firstNeqSubTerms, подчёркнуты.

$P(\underline{a}, x, f(g(y)))$	$P(\underline{z}, f(z), f(u))$	$\sigma = [a/z]$
$P(a, \underline{x}, f(g(y)))$	$P(a, \underline{f(a)}, f(u))$	$\sigma = [a/z, f(a)/x]$
$P(a, f(a), f(\underline{g(y)}))$	$P(a, \underline{f(a)}, f(\underline{u}))$	$\sigma = [a/z, f(a)/x, g(y)/u]$
$P(a, f(a), f(\underline{g(y)}))$	$P(a, f(a), f(g(y)))$	термы совпадают

Если язык формул удовлетворяет определённым условиям, то можно проверить, являются ли две заданные формулы унифицируемыми, и если это так, то найти наиболее общий унификатор. В частности, это возможно для типичного языка формул, описанного в п. 4.3.1 с помощью алгоритма 4.2.

В алгоритме 4.2 предполагается, что функция f осуществляет синтаксический разбор формулы и возвращает знак главной операции (то есть $\to$, $\neg$ или признак того, что формула является переменной), а функции l и r возвращают левый и правый операнды главной операции, то есть подформулы (считаем, что отрицание имеет только правый операнд). Набор подстановок (унификатор) представлен в виде глобального массива S, значениями индекса которого являются имена переменных, а значениями элементов — формулы, которые подставляются вместо соответствующих переменных.

Обоснование. При любых подстановках формул вместо переменных главная операция (связка) формулы остаётся неизменной. Поэтому если главные операции формул различны, то формулы заведомо не унифицируемы. В противном случае, то есть если главные операции совпадают, формулы унифицируемы тогда и только тогда, когда унифицируемы подформулы, являющиеся операндами главной операции. Эта рекурсия заканчивается, когда сопоставление доходит до переменных. Переменная унифицируется с любой формулой (в частности, с другой переменной) простой подстановкой этой формулы вместо переменной. Но подстановки для всех вхождений одной переменной должны совпадать. □

Алгоритм 4.2. Рекурсивная функция унификации Unify

Вход: формулы A и B исчисления высказываний.
Выход: **false**, если формулы не унифицируемы; **true** и наиболее общий унификатор S в противном случае.

```
a := f(A); b := f(B)
// обе формулы — переменные
if var(a) & var(b) then
  if a = b then return true end if
  if S[a] = ∅ & S[b] = ∅ then S[a] := b {или S[b] := a}; return true end if
  if S[a] = ∅ & S[b] ≠ ∅ then S[a] := S[b]; return true end if
  if S[b] = ∅ & S[a] ≠ ∅ then S[b] := S[a]; return true end if
  if S[a] ≠ ∅ & S[b] ≠ ∅ then return (S[a] = S[b]) end if
end if
// одна формула — переменная, а другая — нет
if var(a) ∨ var(b) then
  if var(a) & a ∈ B then
    return false
  else
    if S[a] = ∅ then S[a] := B; return true
    else return (S[a] = B) end if
  end if
  if var(b) & b ∈ A then
    return false
  else
    if S[b] = ∅ then S[b] := A; return true
    else return (S[b] = A) end if
  end if
end if
// обе формулы не переменные
if a ≠ b then return false end if  // главные операции различны
if a = ¬ then return Unify(r(A), r(B)) end if
if a =→ then return Unify(l(A), l(B)) & Unify(r(A), r(B)) end if
```

4.3.4. Конструктивное определение исчисления высказываний

Алфавит и множество формул — те же (см. п. 4.3.1). Аксиомы — три конкретные формулы:

A_1: $(a \to (b \to a))$;
A_2: $((a \to (b \to c)) \to ((a \to b) \to (a \to c)))$;
A_3: $((\neg b \to \neg a) \to ((\neg b \to a) \to b))$.

Правила вывода:

1. Правило подстановки: если формула B является частным случаем формулы A, то B непосредственно выводима из A.
2. Правило Modus ponens: если набор формул A, B, C является частным случаем набора формул a, $(a \to b)$, b, то формула C является непосредственно выводимой из формул A и B.

ЗАМЕЧАНИЕ

Здесь a, $(a \to b)$, b — это три конкретные формулы, построенные с помощью переменных a, b и связки $\to$.

4.3.5. Производные правила вывода

Исчисление высказываний $\mathcal{L}$ — это достаточно богатая формальная теория, в которой выводимы многие важные теоремы.

ЗАМЕЧАНИЕ

Выводимость формул в теории $\mathcal{L}$ доказывается путём предъявления конкретного вывода, то есть последовательности формул, удовлетворяющих определению, данному в п. 4.2.2. Для удобства чтения формулы последовательности вывода выписываются друг под другом в столбик, слева указываются их номера в последовательности, а справа указывается, на каком основании формула включена в вывод (то есть является ли она гипотезой, или получена из схемы аксиом указанной подстановкой, или получена из предшествующих формул по указанному правилу вывода и т. д.).

Всякую доказанную выводимость можно использовать как новое (*производное*) правило вывода. Например, следующая доказанная выводимость называется *правилом введения импликации*:

$$\frac{A}{B \to A}\ (\to^+).$$

ТЕОРЕМА 1. $A \vdash_{\mathcal{L}} B \to A$.

Доказательство. Вывод:

1. A	гипотеза.
2. $A \to (B \to A)$	A_1.
3. $B \to A$	MP; 1,2.

□

ЗАМЕЧАНИЕ

Малое количество аксиом и правил вывода не влечёт лаконичности выводов в формальной теории. Например, в используемой аксиоматике исчисления высказываний всего три схемы аксиомы и одно правило вывода, однако даже простые теоремы могут потребовать сравнительно длинных выводов.

ТЕОРЕМА 2. $\vdash_{\mathcal{L}} A \to A$.

Доказательство. Вывод:

1. $(A \to ((A \to A) \to A))$	$A_1; [A \to A/B]$.
2. $((A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A)))$	$A_2; [A \to A/B, A/C]$.
3. $((A \to (A \to A)) \to (A \to A))$	MP; 1, 2.
4. $A \to (A \to A)$	$A_1; A/B$.
5. $A \to A$	MP; 4, 3.

□

4.3.6. Дедукция

В теории $\mathcal{L}$ импликация очень тесно связана с выводимостью.

ТЕОРЕМА (о дедукции). $\Gamma, A \vdash_{\mathcal{L}} B$ *тогда и только тогда, когда* $\Gamma \vdash_{\mathcal{L}} A \to B$.

Доказательство.

[Необходимость] Пусть $E_1, \ldots, E_n$ — вывод B из Γ, A. $E_n = B$. Индукцией по i $(1 \leqslant i \leqslant n)$ покажем, что $\Gamma \vdash_{\mathcal{L}} A \to E_i$. Тем самым теорема будет доказана. База: $i = 1$. Возможны три случая.

1. Пусть E_1 — аксиома. Тогда рассмотрим вывод E_1, $E_1 \to (A \to E_1)$, $A \to E_1$. Имеем $\vdash_{\mathcal{L}} A \to E_1$.
2. Пусть $E_1 \in \Gamma$. Тогда рассмотрим вывод E_1, $E_1 \to (A \to E_1)$, $A \to E_1$. Имеем $\Gamma \vdash_{\mathcal{L}} A \to E_1$.
3. Пусть $E_1 = A$. Тогда по первой теореме предыдущего параграфа $\vdash_{\mathcal{L}} E_1 \to E_1$, а значит, $\vdash_{\mathcal{L}} A \to E_1$.

В любом случае $\Gamma \vdash_{\mathcal{L}} A \to E_1$. Таким образом, база индукции доказана. Пусть теперь $\Gamma \vdash_{\mathcal{L}} A \to E_i$ для всех $i < k$. Рассмотрим E_k. Возможны четыре случая: либо E_k — аксиома, либо $E_k \in \Gamma$, либо $E_k = A$, либо формула E_k получена по правилу Modus ponens из формул E_i и E_j, причём $i, j < k$ и $E_j = E_i \to E_k$. Для первых трёх случаев имеем $\Gamma \vdash_{\mathcal{L}} A \to E_k$ аналогичным образом. Для четвёртого случая по индукционному предположению имеется вывод $\Gamma \vdash_{\mathcal{L}} A \to E_i$ и вывод $\Gamma \vdash_{\mathcal{L}} A \to (E_i \to E_k)$. Объединим эти выводы и достроим следующий вывод:

$n.$	$(A \to (E_i \to E_k)) \to ((A \to E_i) \to (A \to E_k))$	$A_2; [E_i/B, E_k/C]$.
$n+1.$	$(A \to E_i) \to (A \to E_k)$	MP; j, n.
$n+2.$	$A \to E_k$	MP; i, $n+1$.

Таким образом, $\Gamma \vdash_{\mathcal{L}} A \to E_k$ для любого k, в частности для $k = n$. Но $E_n = B$, то есть $\Gamma \vdash_{\mathcal{L}} A \to B$.

[Достаточность] Имеем вывод $\Gamma \vdash_{\mathcal{L}} A \to B$, состоящий из n формул. Достроим его следующим образом:

$n.$	$A \to B$.	
$n+1.$	A	гипотеза.
$n+2.$	B	MP; $n+1$, n.

Таким образом, имеем $\Gamma, A \vdash_{\mathcal{L}} B$. □

ЗАМЕЧАНИЕ

Схема аксиом A_3 теории $\mathcal{L}$ не использовалась в доказательстве, поэтому теорема дедукции имеет место не только для теории $\mathcal{L}$, но и для любых теорий, в которых выполнены аксиомы A_1 и A_2 и действует правило отделения.

СЛЕДСТВИЕ 1. $A \vdash_{\mathcal{L}} B$ *тогда и только тогда, когда* $\vdash_{\mathcal{L}} A \to B$.

Доказательство. $\Gamma := \varnothing$. □

СЛЕДСТВИЕ 2. $A \to B, B \to C \vdash_{\mathcal{L}} A \to C$.

Доказательство. Вывод:

1. $A \to B$	гипотеза.
2. $B \to C$	гипотеза.
3. A	гипотеза.
4. B	MP; 3, 1.
5. C	MP; 4, 2.
6. $A \to B, B \to C, A \vdash_{\mathcal{L}} C$	1–5.
7. $A \to B, B \to C \vdash_{\mathcal{L}} A \to C$	по теореме дедукции. □

ЗАМЕЧАНИЕ

Это производное правило называется *правилом транзитивности.*

СЛЕДСТВИЕ 3. $A \to (B \to C), B \vdash_{\mathcal{L}} A \to C$.

Доказательство. Вывод:

1. $A \to (B \to C)$	гипотеза.
2. A	гипотеза.
3. $B \to C$	MP; 2, 1.
4. B	гипотеза.
5. C	MP; 4, 3.
6. $A \to (B \to C), B, A \vdash_{\mathcal{L}} C$	1–5.
7. $A \to (B \to C), B \vdash_{\mathcal{L}} A \to C$	по теореме дедукции. □

ЗАМЕЧАНИЕ

Это производное правило называется *правилом сечения.*

4.3.7. Некоторые теоремы исчисления высказываний

Множество теорем теории $\mathcal{L}$ бесконечно. Здесь приведены некоторые теоремы, которые используются в дальнейших построениях.

ЗАМЕЧАНИЕ

Каждая доказанная теорема (то есть формула теории, для которой построен вывод) может использоваться в дальнейших выводах на правах аксиомы.

ТЕОРЕМА. *В теории $\mathcal{L}$ выводимы следующие теоремы:*

а) $\vdash_{\mathcal{L}} \neg\neg A \to A$.
б) $\vdash_{\mathcal{L}} A \to \neg\neg A$.
в) $\vdash_{\mathcal{L}} \neg A \to (A \to B)$.
г) $\vdash_{\mathcal{L}} (\neg B \to \neg A) \to (A \to B)$.
д) $\vdash_{\mathcal{L}} (A \to B) \to (\neg B \to \neg A)$.
е) $\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$.
ж) $\vdash_{\mathcal{L}} (A \to B) \to ((\neg A \to B) \to B)$.

Доказательство.

[а) $\vdash_{\mathcal{L}} \neg\neg A \to A$]

1.	$(\neg A \to \neg\neg A) \to ((\neg A \to \neg A) \to A)$	A_3; $[A/B, \neg A/A]$.
2.	$\neg A \to \neg A$	первая теорема 4.3.5; $[\neg A/A]$.
3.	$(\neg A \to \neg\neg A) \to A$	Сл. 3; $[\neg A \to \neg\neg A/A, \neg A \to \neg A/B, A/C]$.
4.	$\neg\neg A \to (\neg A \to \neg\neg A)$	A_1; $[\neg\neg A/A, \neg A/B]$.
5.	$\neg\neg A \to A$	Сл. 2; $[\neg\neg A/A, \neg A \to \neg\neg A/B, A/C]$.

[б) $\vdash_{\mathcal{L}} A \to \neg\neg A$]

1.	$(\neg\neg\neg A \to \neg A) \to$ $\to ((\neg\neg\neg A \to A) \to \neg\neg A)$	A_3; $[\neg\neg A/B]$.
2.	$\neg\neg\neg A \to \neg A$	a; $[\neg A/A]$.
3.	$(\neg\neg\neg A \to A) \to \neg\neg A$	MP; 2, 1.
4.	$A \to (\neg\neg\neg A \to A)$	A_1; $[\neg\neg\neg A/B]$.
5.	$A \to \neg\neg A$	Сл. 2; $[\neg\neg\neg A/B, A/A, \neg\neg A/C]$.

[в) $\vdash_{\mathcal{L}} \neg A \to (A \to B)$]

1.	$\neg A$	гипотеза.
2.	A	гипотеза.
3.	$A \to (\neg B \to A)$	A_1; $[\neg B/B]$.
4.	$\neg A \to (\neg B \to \neg A)$	A_1; $[\neg A/A, \neg B/B]$.
5.	$\neg B \to A$	MP; 2, 3.
6.	$\neg B \to \neg A$	MP; 1, 4.
7.	$(\neg B \to \neg A) \to ((\neg B \to A) \to B)$	A_3.
8.	$(\neg B \to A) \to B$	MP; 6, 7.
9.	B	MP; 5, 8.
10.	$\neg A, A \vdash_{\mathcal{L}} B$	1–9.
11.	$\neg A \vdash_{\mathcal{L}} A \to B$	теорема дедукции.
12.	$\vdash_{\mathcal{L}} \neg A \to (A \to B)$	теорема дедукции.

[г) $\vdash_{\mathcal{L}} (\neg B \to \neg A) \to (A \to B)$]

1.	$\neg B \to \neg A$	гипотеза.
2.	A	гипотеза.
3.	$(\neg B \to \neg A) \to ((\neg B \to A) \to B)$	A_3.
4.	$A \to (\neg B \to A)$	A_1; $[\neg B/B]$.
5.	$(\neg B \to A) \to B$	MP; 1, 3.
6.	$A \to B$	Сл. 2; $[\neg B \to A/B, B/C]$.
7.	B	MP; 2, 6.
8.	$\neg B \to \neg A, A \vdash_{\mathcal{L}} B$	1–7.
9.	$\neg B \to \neg A \vdash_{\mathcal{L}} A \to B$	теорема дедукции.
10.	$\vdash_{\mathcal{L}} (\neg B \to \neg A) \to (A \to B)$	теорема дедукции.

[д) $\vdash_{\mathcal{L}} (A \to B) \to (\neg B \to \neg A)$]

1.	$A \to B$	гипотеза.
2.	$\neg\neg A \to A$	*а*.
3.	$\neg\neg A \to B$	Сл. 2; $[A/B, \neg\neg A/A, B/C]$.
4.	$B \to \neg\neg B$	*б*; $[B/A]$.
5.	$\neg\neg A \to \neg\neg B$	Сл. 2; $[\neg\neg A/A, \neg\neg B/C]$.
6.	$(\neg\neg A \to \neg\neg B) \to (\neg B \to \neg A)$	*г*; $[\neg A/B, \neg B/A]$.
7.	$\neg B \to \neg A$	MP; 5, 6.
8.	$A \to B \vdash_{\mathcal{L}} \neg B \to \neg A$	1–7.
9.	$\vdash_{\mathcal{L}} (A \to B) \to (\neg B \to \neg A)$	теорема дедукции.

[е) $\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$]

1.	A	гипотеза.
2.	$A \to B$	гипотеза.
3.	B	MP; 1, 2.
4.	$A, A \to B \vdash_{\mathcal{L}} B$	1–3.
5.	$A \vdash_{\mathcal{L}} (A \to B) \to B$	теорема дедукции.
6.	$\vdash_{\mathcal{L}} A \to ((A \to B) \to B)$	теорема дедукции.
7.	$\vdash_{\mathcal{L}} ((A \to B) \to B) \to$ $\to (\neg B \to \neg(A \to B))$	*д*; $[A \to B/A]$.
8.	$\vdash_{\mathcal{L}} A \to (\neg B \to \neg(A \to B))$	Сл. 2; $[((A \to B) \to B)/B;$ $(\neg B \to \neg(A \to B))/C]$.

[ж) $\vdash_{\mathcal{L}} (A \to B) \to ((\neg A \to B) \to B)$]

1.	$A \to B$	гипотеза.
2.	$\neg A \to B$	гипотеза.
3.	$(A \to B) \to (\neg B \to \neg A)$	*д*.
4.	$\neg B \to \neg A$	MP; 1, 3.
5.	$(\neg A \to B) \to (\neg B \to \neg\neg A)$	*д*; $[\neg A/A\}$.
6.	$\neg B \to \neg\neg A$	MP; 2, 5.
7.	$(\neg B \to \neg\neg A) \to ((\neg B \to \neg A) \to B)$	A_3; $\{[\neg A/A\}$.
8.	$(\neg B \to \neg A) \to B$	MP; 6, 7.
9.	B	MP; 4, 8.
10.	$A \to B, \neg A \to B \vdash_{\mathcal{L}} B$	1–9.
11.	$A \to B \vdash_{\mathcal{L}} (\neg A \to B) \to B$	теорема дедукции.
12.	$\vdash_{\mathcal{L}} (A \to B) \to ((\neg A \to B) \to B)$	теорема дедукции. □

4.3.8. Множество теорем исчисления высказываний

Пусть формула A содержит переменные $a_1, \ldots, a_n$, и пусть задана некоторая интерпретация I формулы A, то есть переменным $a_1, \ldots, a_n$ приписаны истинностные значения. Обозначим $A'_i \stackrel{\text{Def}}{=}$ **if** $a_i = \mathrm{T}$ **then** a_i **else** $\neg a_i$ **end if**,
$A' \stackrel{\text{Def}}{=}$ **if** $I(A) = \mathrm{T}$ **then** A **else** $\neg A$ **end if** в данной интерпретации.

ЛЕММА. $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} A'$.

Доказательство. Индукция по структуре формулы A.
[Переменная] Пусть $A = a$. Тогда $a \vdash_{\mathcal{L}} a$ и $\neg a \vdash_{\mathcal{L}} \neg a$.

[Отрицание] Пусть $A = \neg B$ и I — произвольная интерпретация формулы A. Возможны два случая: $I(B) = \mathrm{T}$ или $I(B) = \mathrm{F}$. Пусть $I(B) = \mathrm{T}$. Тогда $I(A) = \mathrm{F}$ и $A' = \neg A = \neg\neg B$. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B$. Но $\vdash_{\mathcal{L}} B \to \neg\neg B$ по теореме п. 4.3.7 *б*, следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg\neg B = A'$. Пусть теперь $I(B) = \mathrm{F}$. Тогда $I(A) = \mathrm{T}$ и $A' = A = \neg B$. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg B = A'$.

[Импликация] Пусть $A = (B \to C)$ и I — произвольная интерпретация формулы A. По индукционному предположению $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B'$ и $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} C'$. Возможны три случая: $I(B) = \mathrm{F}$, или $I(B) = \mathrm{T}$ и $I(C) = \mathrm{T}$, или же $I(B) = \mathrm{T}$ и $I(C) = \mathrm{F}$.

Пусть $I(B) = \mathrm{F}$. Тогда независимо от значения $I(C)$ имеем: $I(A) = \mathrm{T}$ и $B' = \neg B$, $A' = A$. Но $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg B, \vdash_{\mathcal{L}} \neg B \to (B \to C)$ по теореме п. 4.3.7 *в*, следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B \to C = A'$.

Пусть $I(B) = \mathrm{T}$ и $I(C) = \mathrm{T}$. Тогда $I(A) = \mathrm{T}$ и $C' = C$, $A' = A = B \to C$. Имеем $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} C$, $\vdash_{\mathcal{L}} C \to (B \to C)$ (аксиома A_1 с подстановкой $[C/A]$), следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B \to C = A'$.

Пусть теперь $I(B) = \mathrm{T}$ и $I(C) = \mathrm{F}$. Тогда $A' = \neg A = \neg(B \to C)$, $B' = B$ и $C' = \neg C$. Имеем $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} B$, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg C, \vdash_{\mathcal{L}} B \to (\neg C \to \neg(B \to C))$ по теореме п. 4.3.7 *е*. Следовательно, $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} \neg(B \to C) = A'$. □

ТЕОРЕМА. *Теоремами теории $\mathcal{L}$ являются общезначимые формулы и только они:* $\vdash_{\mathcal{L}} A \iff A$ — тавтология.

Доказательство.

[$\Longleftarrow$] Пусть A — тавтология. Тогда $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} A$ в любой интерпретации. Таким образом, имеется 2^n различных выводимостей $A'_1, \ldots, A'_n \vdash_{\mathcal{L}} A$. Среди них есть две, которые различаются в A'_n: $A'_1, \ldots, A'_{n-1}, a_n \vdash_{\mathcal{L}} A$ и $A'_1, \ldots, A'_{n-1}, \neg a_n \vdash_{\mathcal{L}} A$. По теореме дедукции $A'_1, \ldots, A'_{n-1} \vdash_{\mathcal{L}} a_n \to A$ и $A'_1, \ldots, A'_{n-1} \vdash_{\mathcal{L}} \neg a_n \to A$. Но $\vdash_{\mathcal{L}} (a_n \to A) \to ((\neg a_n \to A) \to A)$ по теореме п. 4.3.7, *ж*, с подстановкой $[a_n/A, A/B]$, следовательно, $A'_1, \ldots, A'_{n-1} \vdash_{\mathcal{L}} A$. Повторив этот процесс ещё $n-1$ раз, имеем $\vdash_{\mathcal{L}} A$.

[$\Longrightarrow$] Аксиомы A_1, A_2, A_3 суть тавтологии. Правило Modus ponens сохраняет тавтологичность, следовательно, теоремы теории $\mathcal{L}$ суть тавтологии. □

СЛЕДСТВИЕ. *Теория $\mathcal{L}$ формально непротиворечива.*

Доказательство. Все теоремы $\mathcal{L}$ суть тавтологии. Отрицание тавтологии не есть тавтология. Следовательно, в $\mathcal{L}$ нет теоремы и её отрицания. □

4.3.9. Другие аксиоматизации исчисления высказываний

Теория $\mathcal{L}$ не является единственной возможной аксиоматизацией исчисления высказываний. Её основное достоинство — лаконичность при сохранении определённой

наглядности. Действительно, в теории $\mathcal{L}$ всего две связки, три схемы аксиом и одно правило. Известны и многие другие аксиоматизации исчисления высказываний, предложенные различными авторами.

1. Гильберт[1] и Аккерман[2], 1938.

   Связки: $\vee, \neg$; $\quad (A \to B \stackrel{\text{Def}}{=} \neg A \vee B)$.

   Аксиомы:
   $(A \vee A) \to A$,
   $A \to (A \vee B)$,
   $(A \vee B) \to (B \vee A)$,
   $(B \to C) \to ((A \vee B) \to (A \vee C))$.

   Правило: Modus ponens.

2. Россер[3], 1953.

   Связки: $\&, \neg$; $\quad (A \to B \stackrel{\text{Def}}{=} \neg(A \,\&\, \neg B))$.

   Аксиомы:
   $A \to (A \,\&\, A)$,
   $(A \,\&\, B) \to A$,
   $(A \to B) \to (\neg(B \,\&\, C) \to \neg(C \,\&\, A))$.

   Правило: Modus ponens.

3. Клини[4], 1952.

   Связки: $\neg, \&, \vee, \to$.

   Аксиомы:
   $A \to (B \to A)$,
   $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$,
   $(A \,\&\, B) \to A$,
   $(A \,\&\, B) \to B$,
   $A \to (B \to (A \,\&\, B))$,
   $A \to (A \vee B)$,
   $B \to (A \vee B)$,
   $(A \to C) \to ((B \to C) \to ((A \vee B) \to C))$,
   $(A \to B) \to ((A \to \neg B) \to \neg A)$,
   $\neg\neg A \to A$.

   Правило: Modus ponens.

4. Никод[5], 1917.

   Связка: $|$; $\quad (A \mid B = \neg A \vee \neg B)$.

   Аксиома:
   $(A \mid (B \mid C)) \mid ((D \mid (D \mid D)) \mid$
   $\mid ((E \mid B) \mid ((A \mid E) \mid (A \mid E))))$.

   Правило: $\dfrac{A, A \mid (B \mid C)}{C}$.

---

[1] Давид Гильберт (1862–1943).
[2] Вильгельм Аккерман(1896–1962).
[3] Джон Беркли Россер (1907–1989).
[4] Стефан Клини (1909–1994).
[5] Жан Никод (1893–1924).

4.4. Исчисление предикатов

Описание формальной теории исчисления предикатов в этом разделе носит конспективный характер, в частности, многие технически сложные доказательства опущены. В то же время уделено внимание прагматическим аспектам теории, то есть ответу на вопрос, что выразимо и что невыразимо в исчислении предикатов.

4.4.1. Определение исчисления предикатов

(Чистое) исчисление предикатов (первого порядка) — это формальная теория $\mathcal{K}$, в которой определены следующие компоненты: алфавит, формулы, аксиомы и правила вывода.

1. Алфавит:

связки	основные	$\neg,\ \to$
	дополнительные	$\&,\ \vee$
служебные символы		$(\ ,\ )$
кванторы	*всеобщности*	$\forall$
	существования	$\exists$
предметные	*константы*	$a, b, \ldots, a_1, b_1, \ldots$
	переменные	$x, y, \ldots, x_1, y_1, \ldots$
предметные	*предикаты*	$P, Q, \ldots$
	функциональные символы	$f, g, \ldots$

С каждым предикатом и функциональным символом связано некоторое натуральное число n, которое называется *арностью*, *местностью* или *вместимостью*.

2. Формулы имеют следующий синтаксис:

⟨формула⟩ ::= ⟨атом⟩ |
$\neg$⟨формула⟩ |
(⟨формула⟩ $\to$ ⟨формула⟩) |
$\forall$ ⟨переменная⟩ (⟨формула⟩) |
$\exists$ ⟨переменная⟩ (⟨формула⟩)

⟨атом⟩ ::= ⟨предикат⟩(⟨список термов⟩)

⟨список термов⟩ ::= ⟨терм⟩ | ⟨терм⟩, ⟨список термов⟩

⟨терм⟩ ::= ⟨константа⟩ |
⟨переменная⟩ |
⟨функциональный символ⟩(⟨список термов⟩)

При этом должны быть выполнены следующие контекстные условия: в *терме* $f(t_1, \ldots, t_n)$ функциональный символ f должен быть n-местным. В *атоме* (или *атомарной* формуле) $P(t_1, \ldots, t_n)$ предикат P должен быть n-местным.

Любая переменная может входить в форму несколько раз. При этом различные вхождения одной и той же переменной в формулу имеют разный смысл, в зависмости от того, в какой синтаксической позиции рассматривается вхождение. Прежде всего, необходимо различать *свободные* и *связанные* вхождения переменной. Вхождения переменных в атомарную формулу считаются свободными. Свободные вхождения переменных в формулах A и B по определению считаются свободными в формулах $\neg A$ и $A \to B$. В формулах $\forall x\ (A)$ и $\exists x$

(A) формула A, как правило, имеет свободные вхождения переменной x. Вхождения переменной x в формулы $\forall x\ (A)$ и $\exists x\ (A)$ считаются связанными. Вхождения других переменных (отличных от x), которые были свободными в формуле A, по определению остаются свободными и в формулах $\forall x\ (A)$ и $\exists x\ (A)$. Одна и та же переменная может иметь в одной и той же формуле как свободные, так и связанные вхождения. Если все вхождения переменной в формулу связаны, то переменная называется *связанной* (в данной формуле). Формула, не содержащая свободных вхождений переменных, называется *замкнутой*. В замкнутой формуле все переменные связаны.

Пример. Рассмотрим формулу $\forall x\ (P(x) \to \exists y\ (Q(x,y)))$ и её подформулы. В подформулу $\exists y\ (Q(x,y))$ переменная x входит свободно, а все вхождения переменной y связаны (квантором существования). Таким образом, эта подформула не замкнута. С другой стороны, то же самое вхождение переменной x в подформулу $Q(x,y)$ является связанным в формуле $\forall x\ (P(x) \to \exists y\ (Q(x,y)))$. В этой формуле все вхождения всех переменных связаны, а потому формула замкнута.

ЗАМЕЧАНИЕ

Язык теории $\mathcal{L}$ не содержит кванторов, поэтому понятия свободного и связанного вхождения переменных не применимы непосредственно. Можно расширить исчисление высказываний, допуская замкнутые формулы исчисления предикатов наравне с пропозициональными переменными. В таком случае для удобства обычно полагают, что все формулы теории $\mathcal{L}$ замкнуты.

Формулы вида A и $\neg A$, где A — атом, называются *литеральными* формулами (или *литералами*). В формулах $\forall x\ (A)$ и $\exists x\ (A)$ подформула A называется *областью действия* квантора по x. Обычно связки и кванторы упорядочивают по приоритету следующим образом: $\neg, \forall, \exists, \&, \vee, \to$. Лишние скобки при этом иногда опускают.

Терм t называется *свободным* для переменной x в формуле A, если никакое свободное вхождение переменной x в формулу A не лежит в области действия никакого квантора по переменной y, входящей в терм t. В частности, терм t свободен для любой переменной в формуле A, если никакая переменная терма не является связанной переменной формулы A.

Пример. Терм y свободен для переменной x в формуле $P(x)$, но тот же терм y не свободен для переменной x в формуле $\forall y\ (P(x))$. Терм $f(x,z)$ свободен для переменной x в формуле $\forall y\ (P(x,y) \to Q(x))$, но тот же терм $f(x,z)$ не свободен для переменной x в формуле $\exists z\ (\forall y\ (P(x,y) \to Q(x)))$.

3. Аксиомы (логические): любая система аксиом исчисления высказываний плюс

$$P_1\colon \quad \forall x\ (A(x)) \to A(t),$$
$$P_2\colon \quad A(t) \to \exists x\ (A(x)),$$

где терм t свободен для переменной x в формуле A.

4. Правила вывода:

$$\frac{A,\ A \to B}{B}\ \text{(Modus ponens)}, \quad \frac{B \to A(x)}{B \to \forall x\ (A(x))}\ (\forall^+), \quad \frac{A(x) \to B}{\exists x\ (A(x)) \to B}\ (\exists^+),$$

где формула A содержит свободные вхождения переменной x, а формула B их не содержит.

Исчисление предикатов, которое не содержит предметных констант, функциональных символов, предикатов и собственных аксиом, называется *чистым*. Исчисление предикатов, которое содержит предметные константы и/или функциональные символы и/или предикаты и связывающие их собственные аксиомы, называется *прикладным*.

Исчисление предикатов, в котором кванторы могут связывать только предметные переменные, но не могут связывать функциональные символы или предикаты, называется исчислением *первого порядка*. Исчисления, в которых кванторы могут связывать не только предметные переменные, но и функциональные символы, предикаты или иные множества объектов, называются исчислениями *высших порядков*.

Практика показывает, что прикладного исчисления предикатов первого порядка оказывается достаточно для формализации содержательных теорий во всех разумных случаях.

4.4.2. Интерпретация

Интерпретация I (прикладного) исчисления предикатов $\mathcal{K}$ с областью интерпретации (или *носителем*) M — это набор функций, которые сопоставляют:

1) каждой предметной константе a элемент носителя $I(a)$, $I(a) \in M$;
2) каждому n-местному функциональному символу f операцию $I(f)$ на носителе, $I(f)\colon M^n \to M$;
3) каждому n-местному предикату P отношение $I(P)$ на носителе, $I(P) \subset M^n$.

Пусть $x = (x_1, \dots)$ — набор (последовательность) переменных (входящих в формулу), а $s = (s_1, \dots)$ — набор значений из M. Тогда всякий терм $f(t_1, \dots, t_n)$ имеет значение на s (из области M), то есть существует функция $s^*\colon \{t\} \to M$, определяемая следующим образом:

$$s^*(a) \stackrel{\text{Def}}{=} I(a), \quad s^*(x_i) \stackrel{\text{Def}}{=} s_i, \quad s^*(f(t_1, \dots, t_n)) \stackrel{\text{Def}}{=} I(f)(s^*(t_1), \dots, s^*(t_n)).$$

Всякий атом $P(t_1, \dots, t_n)$ имеет на s истинностное значение $s^*(P)$, определяемое следующим образом: $s^*(P(t_1, \dots, t_n)) \stackrel{\text{Def}}{=} (s^*(t_1), \dots, s^*(t_n)) \in I(P)$.

Если $s^*(P) = \text{T}$, то говорят, что формула P *выполнена* на s.

Формула $\neg A$ выполнена на s тогда и только тогда, когда формула A не выполнена на s. Формула $A \to B$ выполнена на s тогда и только тогда, когда формула A не выполнена на s или формула B выполнена на s. Формула $\forall x_i\ (A)$ выполнена на s тогда и только тогда, когда формула A выполнена на любом наборе s', отличающемся от s не более чем своим i-м компонентом. Формула $\exists x_i\ (A)$ выполнена на s тогда и только тогда, когда формула A выполнена на каком-либо наборе s', отличающемся от s не более чем своим i-м компонентом.

Формула называется *истинной* в данной интерпретации I, если она выполнена на любом наборе s элементов M. Формула называется *ложной* в данной интерпретации I, если она не выполнена ни на одном наборе s элементов M.

Интерпретация называется *моделью* множества формул Γ, если все формулы из Γ истинны в данной интерпретации.

Всякая замкнутая формула истинна или ложна в данной интерпретации. *Открытая* (то есть *не* замкнутая) формула $A(x, y, z, \dots)$ истинна в данной интерпретации тогда и только тогда, когда её *замыкание* $\forall x\ (\forall y\ (\forall z \cdots A(x, y, z, \dots)))$ истинно в данной интерпретации.

ЗАМЕЧАНИЕ

Это обстоятельство объясняет, почему собственные аксиомы прикладных теорий обычно пишутся в открытой форме.

4.4.3. Общезначимость

Формула (исчисления предикатов) *общезначима*, если она истинна в любой интерпретации.

ТЕОРЕМА. *Формула $\forall x\ (A(x)) \to A(t)$, где терм t свободен для переменной x в формуле A, общезначима.*

Доказательство. Рассмотрим произвольную интерпретацию I, произвольную последовательность значений из области интерпретации s и соответствующую функцию s^*. Заметим следующее. Пусть t_1 — терм и $s^*(t_1) = a_1$. Пусть $t(x)$ — некоторый другой терм, а $t' := t(\dots x \dots)\{t_1/x\}$. Тогда $s^*(t') = s_1^*(t)$, где s_1^* имеет значение a_1 на месте x. Пусть теперь $A(x)$ — формула, а терм t свободен для x в A. Положим $A(t) := A(\dots x \dots)\{t/x\}$. Имеем $s^*(A(t)) = \text{Т} \iff s_1^*(A(t)) = \text{Т}$, где s_1^* имеет значение $s^*(t)$ на месте x. Если $s^*(\forall x\ (A(x))) = \text{Т}$ и терм t свободен для x в A, то $s^*(A(t)) = \text{Т}$. Следовательно, формула $\forall x\ (A(x)) \to A(t)$ выполнена на всех последовательностях в произвольной интерпретации. □

ЗАМЕЧАНИЕ

Можно показать, что формула $A(t) \to \exists x\ (A)(x)$, где терм t свободен для переменной x в формуле A, общезначима.

4.4.4. Непротиворечивость и полнота чистого исчисления предикатов

Рассмотрим преобразование h формулы исчисления предикатов в формулу исчисления высказываний, которое «забывает» всё, что связано с исчислением предикатов, оставляя только *пропозициональную структуру*, которая является формулой исчисления высказываний.

При преобразовании h в формуле исчисления предикатов опускаются все кванторы и термы вместе с соответствующими скобками и запятыми, остаются только предикатные символы (которые рассматриваются как пропозициональные переменные) и связки.

Пример. $h(\forall x\ (P(x) \to \exists y\ (Q(x, y)))) = P \to Q$.

Нетрудно видеть, что $h(\neg A) = \neg h(A)$ и $h(A \to B) = h(A) \to h(B)$.

ТЕОРЕМА 1. *Формальная теория $\mathcal{K}$ непротиворечива.*

Доказательство. Рассмотрим пропозициональную структуру аксиом исчисления предикатов $\mathcal{K}$, то есть применим к аксиомам преобразование h. Ясно, что все полученные формулы суть тавтологии, поскольку аксиомы A_1, A_2, A_3 сами по себе совпадают со своими пропозициональными структурами, которые являются тавтологиями. Для аксиом исчисления предикатов имеем
$h(P_1) = h(\forall x\ (A(x)) \to A(t)) = A \to A$ и $h(P_2) = h(A(t) \to \exists x\ (A(x))) = A \to A$. Далее, правило Modus ponens сохраняет тавтологичность пропозициональной структуры, то есть если $h(A)$ и $h(A \to B)$ — тавтологии, то $h(B)$ — тавтология. Таким образом, если формула A является теоремой теории $\mathcal{K}$, то формула $h(A)$ является тавтологией. Далее от противного. Если $\vdash_{\mathcal{K}} A$ и $\vdash_{\mathcal{K}} \neg A$, то $h(A)$ и $\neg h(A)$ — тавтологии, что невозможно. □

Следующие две метатеоремы устанавливают свойства полноты исчисления предикатов, аналогичные тем, которые установлены для исчисления высказываний.

Теорема 2. *Всякая теорема чистого исчисления предикатов первого порядка общезначима.*

Теорема 3. *Всякая общезначимая формула является теоремой чистого исчисления предикатов первого порядка.*

Доказательство. Без доказательства. □

4.4.5. Логическое следование и логическая эквивалентность

Формула B является *логическим следствием* формулы A (обозначение $A \Longrightarrow B$), если формула B выполнена на любом наборе в любой интерпретации, на котором выполнена формула A. Формулы A и B *логически эквивалентны* (обозначение $A = B$ или $A \Leftrightarrow B$), если они являются логическим следствием друг друга. Можно показать, что имеют место следующие логические следования и эквивалентности:

1. $\neg\forall x\ (A(x)) = \exists x\ (\neg A(x))$,
2. $\neg\exists x\ (A(x)) = \forall x\ (\neg A(x))$,
3. $\forall x\ ((A(x)\ \&\ B(x))) = \forall x\ (A(x))\ \&\ \forall x\ (B(x))$,
4. $\exists x\ (A(x) \vee B(x)) = \exists x\ (A(x)) \vee \exists x\ (B(x))$,
5. $\exists x\ (A(x)\ \&\ B(x)) \Longrightarrow \exists x\ (A(x))\ \&\ \exists x\ (B(x))$,
6. $\forall x\ (A(x)) \vee \forall x\ (B(x)) \Longrightarrow \forall x\ (A(x) \vee B(x))$,
7. $\forall x\ (\forall y\ (A(x,y))) = \forall y\ (\forall x\ (A(x,y)))$,
8. $\exists x\ (\exists y\ (A(x,y))) = \exists y\ (\exists x\ (A(x,y)))$,
9. $\forall x\ ((A(x)\ \&\ C)) = \forall x\ (A(x))\ \&\ C$,
10. $\forall x\ (A(x) \vee C) = \forall x\ (A(x)) \vee C$,
11. $\exists x\ (A(x)\ \&\ C) = \exists x\ (A(x))\ \&\ C$,
12. $\exists x\ (A(x) \vee C) = \exists x\ (A(x)) \vee C$,
13. $C \to \forall x\ (A(x)) = \forall x\ (C \to A(x))$,
14. $C \to \exists x\ (A(x)) = \exists x\ (C \to A(x))$,
15. $\exists x\ (\forall y\ (A(x,y))) \Longrightarrow \forall y\ (\exists x\ (A(x,y)))$,
16. $\forall x\ (A(x) \to B(x)) \Longrightarrow \forall x\ (A(x)) \to \forall x\ (B(x))$,

где формула C не содержит никаких вхождений переменной x.

Для всякой замкнутой формулы A существует логически эквивалентная ей формула A', имеющая *предварённую форму*: $A' = Q_1\, x_1 \ldots Q_n\, x_n\, \bar{A}(x_1, \ldots, x_n)$, где $Q_1, \ldots, Q_n$ — некоторые кванторы, а $\bar{A}$ — *бесквантор*ная формула.

4.4.6. Теория равенства

Этот параграф начинает серию примеров прикладных исчислений предикатов.

ЗАМЕЧАНИЕ

Всякое прикладное исчисление предикатов содержит в себе чистое исчисление предикатов, поэтому при описании прикладного исчисления аксиомы и все теоремы чистого исчисления подразумеваются, указываются только собственные функциональные символы, предикаты и аксиомы.

Теория равенства $\mathcal{E}$ — это прикладное исчисление предикатов, в котором имеются:

1. Собственный двухместный предикат $=$ (инфиксная запись $x = y$).
2. Собственные аксиомы (схемы аксиом):

E_1: $\forall x\ (x = x)$,
E_2: $(x = y) \to (A(x) \to A(x)\{y/x\})$.

ТЕОРЕМА. *В теории* $\mathcal{E}$ *выводимы следующие теоремы:*

1. $\vdash_{\mathcal{E}} t = t$ *для любого терма* t.
2. $\vdash_{\mathcal{E}} x = y \to y = x$.
3. $\vdash_{\mathcal{E}} x = y \to (y = z \to x = z)$.

Доказательство.

[1] Из E_1 и P_1 по Modus ponens.

[2] Имеем: $(x = y) \to (x = x \to y = x)$ из E_2 при подстановке $\{x = x/A(x)\}$. Значит, $x = y, x = x \vdash_{\mathcal{E}} y = x$. Но $\vdash_{\mathcal{E}} x = x$, следовательно, $x = y \vdash_{\mathcal{E}} y = x$. По теореме о дедукции $\vdash_{\mathcal{E}} x = y \to y = x$.

[3] Имеем: $y = x \to (y = z \to x = z)$ из E_2 при подстановке $\{x = z/A(x), y/x, x/y\}$, значит, $y = x \vdash_{\mathcal{E}} (y = z \to x = z)$. Но $x = y \vdash_{\mathcal{E}} y = x$, по правилу транзитивности $x = y \vdash_{\mathcal{E}} (y = z \to x = z)$, по теореме о дедукции $\vdash_{\mathcal{E}} (x = y) \to (y = z \to x = z)$. □

Таким образом, равенство является эквивалентностью. Но равенство — это более сильное свойство, чем эквивалентность. Аксиома E_2 выражает *неотличимость* равных элементов.

4.4.7. Формальная арифметика

Формальная арифметика $\mathcal{A}$ — это прикладное исчисление предикатов, в котором имеются:

1. Предметная константа 0.
2. Двухместные функциональные символы $+$ и $\cdot$, одноместный функциональный символ $'$.
3. Двухместный предикат $=$.

4. Собственные аксиомы (схемы аксиом):

A_1: $(P(0)\ \&\ \forall x\ (P(x) \to P(x'))) \to \forall x\ (P(x))$,
A_2: $t_1' = t_2' \to t_1 = t_2$,
A_3: $\neg(t' = 0)$,
A_4: $t_1 = t_2 \to (t_1 = t_3 \to t_2 = t_3)$,
A_5: $t_1 = t_2 \to t_1' = t_2'$,
A_6: $t + 0 = t$,
A_7: $t_1 + t_2' = (t_1 + t_2)'$,
A_8: $t \cdot 0 = 0$,
A_9: $t_1 \cdot t_2' = t_1 \cdot t_2 + t_1$,

где P — любая формула, а t, t_1, t_2 — любые термы теории $\mathcal{A}$.

ЗАМЕЧАНИЕ

Впервые данная система аксиом для арифметики была предложена Пеано[1], а A_1 — это схема аксиомы математической индукции.

4.4.8. Неаксиоматизируемые теории

В этом параграфе на примере некоторых понятий теории (абелевых) групп нефор-мально устанавливаются границы применимости исчисления предикатов первого порядка.

Теория групп $\mathcal{G}$ — это прикладное исчисление предикатов *с равенством*, в котором имеются:

1. Предметная константа 0.
2. Двухместный функциональный символ $+$.
3. Собственные аксиомы (схемы аксиом):

G_1: $\forall x, y, z\ (x + (y + z) = (x + y) + z)$,
G_2: $\forall x\ (0 + x = x)$,
G_3: $\forall x\ (\exists y\ (x + y = 0))$.

ЗАМЕЧАНИЕ

Выражение «теория первого порядка с равенством» означает, что подразумевается наличие предиката $=$, аксиом E_1 и E_2 и всех их следствий.

Группа называется *абелевой*, если имеет место собственная аксиома

G_4: $\forall x, y\ (x + y = y + x)$.

Говорят, что в абелевой группе все элементы имеют *конечный порядок* n, если выполнена собственная аксиома

G_5: $\forall x\ (\exists k \leqslant n\ (kx = 0))$, где kx — это сокращение для $\underbrace{x + x + \cdots + x}_{k \text{ раз}}$.

[1] Джузеппе Пеано (1858–1932).

Эта формула не является формулой теории $\mathcal{G}$, поскольку содержит «посторонние» предметные предикаты и переменные. Однако для любого конкретного конечного n собственная аксиома может быть записана в виде допустимой формулы теории $\mathcal{G}$:

G'_5: $\forall x\ ((x = 0 \vee 2x = 0 \vee \ldots \vee nx = 0))$.

Абелева группа называется *делимой*, если выполнена собственная аксиома

G_6: $\forall n \geqslant 1\ (\forall x\ (\exists y\ (ny = x)))$.

Эта формула не является формулой теории $\mathcal{G}$, поскольку содержит «посторонние» предметные константы и переменные. Однако собственная аксиома может быть записана в виде бесконечного множества допустимых формул теории $\mathcal{G}$:

G'_6: $\forall x\ (\exists y\ (2y = x))$,
$\forall x\ (\exists y\ (3y = x))$,
$\ldots$

Но можно показать, что любое *конечное* множество формул, истинное во всех делимых абелевых группах, истинно и в некоторой неделимой абелевой группе, то есть теория делимых абелевых групп не является *конечно* аксиоматизируемой.

Абелева группа называется *периодической*, если выполнена собственная аксиома

G_7: $\forall x\ (\exists n \geqslant 1\ (nx = 0))$.

Эта формула также не является формулой теории $\mathcal{G}$, поскольку содержит «посторонние» предметные константы и переменные. Если попытаться преобразовать формулу G_7 по образцу формулы G_5, то получится бесконечная «формула»

G'_7: $\forall x\ ((x = 0 \vee 2x = 0 \vee \ldots \vee nx = 0 \vee \ldots))$,

которая не является допустимой формулой исчисления предикатов и тем более теории $\mathcal{G}$. Таким образом, теория периодических абелевых групп не является *аксиоматизируемой* (если не включать в теорию групп и всю формальную арифметику).

ОТСТУПЛЕНИЕ

Наличие неаксиоматизируемых и конечно неаксиоматизируемых формальных теорий не означает практической неприменимости аксиоматического метода на основе использования языка исчисления предикатов первого порядка. Это означает, что аксиоматический метод не применим «в чистом виде». На практике формальные теории, описывающие содержательные объекты, задаются с помощью собственных аксиом, которые наряду с собственными предикатами и функциональными символами содержат «внелогические» предикаты и функциональные символы, свойства которых аксиомами не описываются, а считаются известными (в данной теории). В рассмотренных примерах внелогическими являются натуральные числа и операции над ними. Аналогичное обстоятельство имеет место и в системах логического программирования типа Пролог. Реализация такой системы всегда снабжается обширной библиотекой внелогических (или *встроенных*) предикатов и функций, которые и обеспечивают практическую применимость системы логического программирования.

4.4.9. Теоремы Гёделя о неполноте

Давно известны некоторые важные свойства формальных теорий, в частности прикладных исчислений предикатов первого порядка, которые существенным образом влияют на практическую применимость формальных теорий (аксиоматического метода). Однако понимание границ применимости формальных методов является, по

нашему мнению, необходимым для использования таких методов. Поэтому ниже приводятся два важнейших факта, известные как теоремы Гёделя о неполноте.

Аксиоматический метод обладает множеством достоинств и с успехом применяется на практике для формализации самых разнообразных предметных областей в математике, физике и других науках. Однако этому методу присуще следующее принципиальное ограничение.

ТЕОРЕМА (Первая теорема Гёделя о неполноте). *Во всякой достаточно богатой непротиворечивой теории первого порядка (в частности, во всякой теории, включающей формальную арифметику) существует такая истинная формула F, что ни F, ни $\neg F$ не являются выводимыми в этой теории.*

Утверждения о теории первого порядка также могут быть сформулированы в виде формул теории первого порядка. В частности, утверждения о свойствах формальной арифметики (например, утверждение о непротиворечивости формальной арифметики) могут быть сформулированы как арифметические утверждения.

ТЕОРЕМА (Вторая теорема Гёделя о неполноте). *Во всякой достаточно богатой теории первого порядка (в частности, во всякой теории, включающей формальную арифметику) формула F, утверждающая непротиворечивость этой теории, не является выводимой в ней.*

ОТСТУПЛЕНИЕ

Вторая теорема Гёделя не утверждает, что арифметика противоречива. (Формальная арифметика из примера п. 4.4.7 как раз непротиворечива.) Эта теорема утверждает, что непротиворечивость достаточно богатой формальной теории не может быть установлена средствами самой этой теории. При этом вполне может статься, что непротиворечивость одной конкретной теории может быть установлена средствами другой, более мощной формальной теории. Но тогда возникает вопрос о непротиворечивости этой второй теории и т. д.

4.5. Наивная теория алгоритмов

В этом разделе изложено доказательство первой теоремы Гёделя о неполноте, основанное на интуитивном понятии алгоритма. Приводимое доказательство в своих идеях следует брошюре Успенского[1] [6] и является достаточно далёким от оригинального доказательства Гёделя. Основная цель раздела состоит в демонстрации применимости и действенности *программистских* подходов в *математическом* контексте.

4.5.1. Алгоритмы

Интуитивное понятие алгоритма как «набора инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий» или как «точно определённого предписания, позволяющего во многих случаях из исходных данных получить результат», обладает в точности теми же достоинствами и недостатками, что и определение понятия множества в п. 1.1.1. Приведённые определения понятия «алгоритм» понятны всем программистам, они вызывают верные практические ассоциации с программами, и они совершенно неполны и не обладают математической строгостью.

[1] Владимир Андреевич Успенский (род. 1930).

Тем не менее, даже этого нестрогого понятия оказывается достаточно во многих важных случаях, так же как и нестрогого понятия «множество» в наивной теории множеств оказывается достаточно для многих полезных построений, в частности тех, которые приведены в этом учебнике.

Будем считать *алгоритм* неопределяемым понятием, постулируя те или иные его свойства по мере необходимости.

ОТСТУПЛЕНИЕ

Обычно в *строгой* теории алгоритмов вводится одно или несколько «уточнений» понятия алгоритма, например: *машина Тьюринга*, *машина Поста*, *нормальный алгоритм Маркова*, и многие другие. В сущности, каждое такое уточнение понятия алгоритма определяет формальный способ записи предписаний, то есть программ и неформальный алгоритм интерпретации программ, то есть правила применения программ к аргументам для получения результатов. Все предложенные к настоящему моменту уточнения интуитивного понятия алгоритма оказались эквивалентными (в строгом математическом смысле), что дало основание выдвинуть *тезис Тьюринга–Чёрча*, неформально постулирующий, что все уточнённые понятия алгоритма равнообъёмны интуитивному понятию алгоритма. Формальные уточнения понятия «алгоритм» и построенная на их основе строгая теория алгоритмов обладают несомненными достоинствами и подлежат неукоснительному изучению. Однако теория алгоритмов имеет две имманентных особенности, которые в практическом применении проявляют себя как неудобства. Во-первых, введение формального способа записи алгоритмов резко усложняет и затрудняет программирование, поскольку программировать приходится в терминах очень низкого уровня (ячейки ленты машины Тьюринга и тому подобное). Во-вторых, способ интерпретации программ (*выполнения алгоритмов*) всё равно остаётся неформальным и потому допускает различные реализации. Поэтому затруднительно ввести независящее от реализации средство прямого анализа свойств выполнения алгоритмов, и приходится применять косвенные критерии, подобные асимптотическим оценкам трудоёмкости и тому подобное.

В этом учебнике тезис Тьюринга–Чёрча принимается безоговорочно; активно используемый способ записи программ (псевдокод) принимается как достаточно выразительный способ записи алгоритмов; убедительные рассуждения относительно текстов алгоритмов принимаются как достаточные обоснования свойств этих алгоритмов. В целом изложение можно назвать *наивной теорией алгоритмов* по аналогии с наивной теорией множеств.

Не ограничивая общности, далее считается, что зафиксирован некоторый конечный алфавит символов U, достаточный для записи всех необходимых объектов. Алгоритмы записываются с помощью последовательностей символов из этого алфавита; данные, к которым применяются алгоритмы, также записываются с помощью последовательностей символов (возможно, других символов), элементы всех множеств записываются с помощью слов в этом алфавите, функции действуют из множества слов в множество слов в этом алфавите и т. д. Например, можно считать, что в алфавит U входят кириллические, латинские и греческие буквы, прописные и строчные, в различных начертаниях, арабские цифры, знаки препинания, скобки, математические знаки и т. д.

Правила записи (кодирования) различных объектов словами в алфавите U считаются известными. В частности, считаются известными правила записи натуральных чисел (например, с помощью десятичных цифр), арифметических выражений (например, с помощью знаков операций, чисел и переменных), алгоритмов (например, с помощью псевдокода).

Таким образом, принимается, что существуют алгоритмы, которые по любому слову в данном фиксированном алфавите U позволяют надежно определить, соответствует

ли это слово правилам записи, и определить, какой именно объект записан. Доказывать существование подобных алгоритмов лексического и синтаксического анализа нет нужды — это эмпирический факт, доступный в повседневных ощущениях при работе с компьютерами.

Имена алгоритмов записываются прописными полужирными латинскими буквами в прямом начертании (например, **A**), чтобы легко отличать их от имен множеств и иных объектов. Сами алгоритмы записываются на псевдокоде. Если алгоритмы имеют входные данные и результаты, то они указываются отдельно и называются *заголовком алгоритма*, а операторы, задающие сам алгоритм, называются *телом алгоритма*.

4.5.2. Вычислимые функции

Алгоритмы бывают разные. В частности, алгоритмы могут обладать разными возможностями в части вырабатывания результата.

Рассмотрим случай, когда алгоритм **A** принимает на вход некоторые данные (*аргумент*) и перерабатывает их в некоторые другие или те же самые данные (*результат*). Тем самым алгоритм **A** определяет отношение A между множеством возможных входных данных (обозначение $\mathrm{Dom}\,\mathbf{A}$) и множеством возможных результатов (обозначение $\mathrm{Im}\,\mathbf{A}$),

$$A \subset \mathrm{Dom}\,\mathbf{A} \times \mathrm{Im}\,\mathbf{A}.$$

Если отношение A обладает свойством функциональности (п. 1.6.1), то говорят, что алгоритм **A** *реализует вычислимую функцию*. Если функция является вычислимой, то есть если предъявлен реализующий её алгоритм, то может быть построено бесконечно много других алгоритмов, также реализующих эту же функцию. Действительно, для этого достаточно вставлять в тело любого из уже построенных алгоритмов произвольные ненужные операторы, выполнение которых не влияет на результат. Все алгоритмы, реализующие одну и ту же функцию, называются *функционально эквивалентными*.

Функциональная эквивалентность, очевидно, действительно является отношением эквивалентности на классе алгоритмов и определяет фактор-множество (п. 1.7.3) классов функционально эквивалентных алгоритмов. Это фактор-множество взаимно однозначно соответствует классу вычислимых функций, поэтому можно выбрать какой-либо из функционально эквивалентных алгоритмов, объявить его *представлением* вычислимой функции и использовать в качестве функции везде, где это необходимо, в частности в других алгоритмах для вычисления значения этой функции.

Это обстоятельство оправдывает используемые далее вольности в обозначениях и в словесных оборотах, когда вычислимые функции и их представления не различаются, подобно тому, как мы говорим о функции, предъявляя запись реализующей её формулы (п. 3.2.4).

ЗАМЕЧАНИЕ

Для целей наивной теории алгоритмов совершенно не важно, какой именно из функционально эквивалентных алгоритмов представляет вычислимую функцию. Для практического программирования это, напротив, один из главных вопросов, потому что функционально эквивалентные алгоритмы могут драматически отличаться друг от друга по длине записи, по эффективности работы, по степени понятности и т. д. Для алгоритмов неизвестно нормальных форм (п. 3.4.3), к сожалению.

В используемом псевдокоде в теле вычислимой функции, как правило, присутствует оператор **return**.

Для записи вычислимых функций применяются те же обозначения, что и для записи функций:

$$\mathbf{A}\colon \operatorname{Dom}\mathbf{A} \to \operatorname{Im}\mathbf{A}, \quad \operatorname{Im}\mathbf{A} = \mathbf{A}(\operatorname{Dom}\mathbf{A}), \quad r = \mathbf{A}(a), \quad \text{где } a \in \operatorname{Dom}\mathbf{A}, r \in \operatorname{Im}\mathbf{A}.$$

ЗАМЕЧАНИЕ

Вычислимая функция может быть частичной, то есть в записи алгоритма $\mathbf{A}$ может быть указано, что алгоритм принимает на вход значения типа T_1 и выдает на выход значения типа T_2 ($\mathbf{A}\colon T_1 \to T_2$), но при этом, на самом деле, $\operatorname{Dom}\mathbf{A} \subsetneq T_1$ и $\operatorname{Im}\mathbf{A} \subsetneq T_2$, то есть область отправления и область прибытия шире области определения и области значений, соответственно. В этом случае считается, что при применении к неподходящему аргументу вычислимая функция *не вырабатывает никакого результата*. Тем самым наивная теория алгоритмов приближается к суровой программистской практике.

Примеры

1. Суперпозиция вычислимых функций, очевидно, вычислима. Тем самым любая формула над базисом вычислимых функций задает вычислимую функцию.
2. Алгоритмы унификации (п. 4.3.3) и многие другие алгоритмы в этом учебнике задают вычислимые функции.

ОТСТУПЛЕНИЕ

Понятие «тип», или «тип данных», использованное в предыдущем замечании, имеет важное значение в программировании. С точки зрения наивной теории алгоритмов *тип* — это некоторое перечислимое (и разрешимое, см. п. 4.5.4) множество слов в алфавите U. В программировании понятие типа (и родственное понятие класса) применяется, в частности, для проверки правильности применения (частичных) вычислимых функций и алгоритмов. А именно, указывая типы параметров функции, программист задаёт (до некоторой степени) область определения функции, и система программирования может проверить, принадлежит переданный аргумент указанному типу или нет (это возможно, поскольку тип — разрешимое множество). Результаты этой проверки могут быть использованы для диагностирования или предотвращения ошибки и для других полезных действий. В этом разделе нам не понадобилась теория типов, но это важное и практически полезное ответвление наивной теории алгоритмов.

4.5.3. Перечислимые множества

Рассмотрим теперь случай, когда алгоритм $\mathbf{A}$ не принимает на вход никаких данных, а будучи запущенным, вырабатывает некоторую последовательность результатов (п. 1.1.2). Тем самым алгоритм $\mathbf{A}$ определяет некоторое множество A, являющееся множеством слов в алфавите U, которое называется *перечислимым*:

$$A = \{a \in U^* \mid a := \mathbf{A}()\}.$$

Совершенно аналогично вычислимым функциям, перечислимые множества могут быть реализованы бесконечным множеством алгоритмов, которые все эквивалентны в том смысле, что перечисляют одно и то же множество. Условимся для каждого перечислимого множества выбирать *представление* в классе эквивалентных алгоритмов и не различать перечислимое множество и его представление без нужды. Алгоритм, представляющий перечислимое множество, для краткости речи называется просто *перечисляющим*.

В используемом псевдокоде в теле алгоритма, перечисляющего множество, как правило, присутствует оператор **yield** a.

Сделаем несколько замечаний относительно перечислимых множеств.

1. Иногда оператор **yield** a отсутствует в явном виде, но тогда он всегда моделируется оператором $A := A + a$ добавления элемента a в изначально пустое множество A.
2. Всякое конечное множество $\{a_1, \ldots, a_n\}$ может быть задано перечислением элементов и тем самым является перечислимым. Действительно, алгоритм **begin yield** a_1; ...; **yield** a_n **end** перечисляет это множество. В частности, алфавит U перечислим, перечисляющий его алгоритм обозначим **U**.
3. Всякое перечислимое множество A линейно упорядочено (п. 1.8.2). Действительно, порядок выполнения операторов **yield** (или оператора добавления элемента) очевидно линеен.
4. Перечислимые множества (и только они!) могут появляться в заголовке цикла по множеству **for** $a \in A$ **do** $\mathbf{B}(a)$ **end for**, где **B** — некоторый алгоритм. Действительно, чтобы реализовать этот цикл, достаточно в теле перечисляющего алгоритма **A** заменить оператор **yield** a вызовом алгоритма $\mathbf{B}(a)$.
5. Конкретные перечисляющие алгоритмы, приводимые в примерах, генерируют каждый элемент ровно один раз, как это и требуется в обычных множествах. Однако явно накладывать такое ограничение на перечисляющие алгоритмы необязательно. Действительно, если имеется перечисляющий алгоритм, который перечисляет какие-то элементы несколько раз, то в его тело можно вставить проверку, исключающую повторные добавления элементов. Перечисляющие алгоритмы, генерирующие элементы по несколько раз, можно использовать для работы с мультимножествами, которые в этом разделе не понадобились.

Примеры

1. Множество натуральных чисел перечислимо:
$$\mathbb{N} = \{n \mid n := 0;\ \textbf{while true do}\ n := n + 1; \textbf{yield}\ n\ \textbf{end while}\}.$$
2. Алгоритмы п. 1.3.2 – 1.3.4 задают перечислимые множества.
3. Пустое множество $\varnothing$ и множество всех слов U^* перечислимы. Пустое множество перечислимо пустым алгоритмом **skip**, а множество всех слов перечислимо очевидным алгоритмом, который вначале перечисляет все слова длины 1, потом все слова длины 2 и т. д.

ТЕОРЕМА 1. *Существуют невычислимые функции и неперечислимые множества.*

Доказательство. Число различных слов в непустом конечном алфавите счётно. Всякий алгоритм — это слово в конечном алфавите, следовательно, число различных алгоритмов не более чем счётно. Фактор-множество (п. 1.7.3) не может превосходить исходное множество по мощности, поэтому множество классов функционально эквивалентных алгоритмов не более чем счётно. С другой стороны, множество всех подмножеств счётного множества несчётно. Значит, число различных множеств слов и число различных функций из множеств слов в множества слов несчётно. Следовательно, невозможно взаимно-однозначное соответствие из множества алгоритмов в множество множеств и в множество функций. □

СЛЕДСТВИЕ. *Существуют перечислимые множества и вычислимые функции; их количества не более чем счётны.*

ЗАМЕЧАНИЕ

Наряду с вычислимыми функциями и перечислимыми множествами бывает полезно рассматривать и другие типы алгоритмов, например итераторы, трансформации и т. п.

Пример. Множество $\mathbb{N}$ перечислимо, множество $\mathbb{N}^2 = \mathbb{N} \times \mathbb{N}$ также перечислимо. Например, следующий алгоритм **N2** перечисляет пары натуральных чисел (a, b): $n := 0$; **while true do** $n := n + 1$; **for** b **from** 1 **to** n **do yield** $(n - b + 1, b)$ **end for end while**.

Перечислимость множеств тесно связана с вычислимыми функциями натурального ряда. Действительно, перечислимое множество $\mathbb{N}$ (и его отрезки, п. 1.2.5) — это удобное и естественное средство *нумерации*, то есть перечисления множеств слов. Пусть A — перечислимое множество, заданное алгоритмом **A**. Построим вычислимую функцию $F\colon \mathbb{N} \to A$, возвращающую элемент перечислимого множества по его номеру, и обратную функцию $F^{-1}\colon A \to \mathbb{N}$, возвращающую номер заданного элемента:

func $F(n : \mathbb{N}) : A$
$k := 0$
for $a \in A$ **do**
  $k := k + 1$
  if $k = n$ **then return** a **end if**
end for
return fail

func $F^{-1}(x : A) : \mathbb{N}$
$k := 0$
for $a \in A$ **do**
  $k := k + 1$
  if $x = a$ **then return** k **end if**
end for
return fail

Перечисляющую функцию F всегда можно и иногда удобно использовать в качестве представления перечислимого множества наряду с генерирующим алгоритмом.

ЗАМЕЧАНИЕ

В этих алгоритмах показан один из способов работы с частичными вычислимыми функциями в используемом псевдокоде. Если значение функции не определено, то возвращается специальное значение **fail**.

Пример. Формула $2^a(2b + 1) - 1$ задает номер пары в некотором перечислении натуральных пар (a, b).

ТЕОРЕМА 2. *Объединение и пересечение перечислимых множеств перечислимы.*

Доказательство. Пусть X и Y — перечислимые множества с перечисляющими функциями $\mathbf{X}\colon \mathbb{N} \to X$ и $\mathbf{Y}\colon \mathbb{N} \to Y$.

[$\cup$] Если хотя бы одно из множеств пусто, то утверждение тривиально. Иначе вычислимая функция $\mathbf{Z}\colon \mathbb{N} \to X \cup Y$ перечисляет объединение: $\mathbf{Z}(n) :=$
if $n = (n \,\mathbf{div}\, 2) * 2$ **then return** $X(n \,\mathbf{div}\, 2)$ **else return** $Y((n + 1) \,\mathbf{div}\, 2)$ **end if**.

[$\cap$] Если пересечение пусто, то оно перечислимо по определению. Иначе рассмотрим алгоритм **N2**, перечисляющий пары натуральных чисел (см. пример выше), и построим алгоритм, перечисляющий пересечение:
for $(a, b) \in$ **N2 do if** $X(a) = Y(b)$ **then yield** $X(a)$ **end if end for**. □

ЗАМЕЧАНИЕ

Другие примеры перечисляющих алгоритмов в форме итераторов приведены в п. 1.3.8.

4.5.4. Разрешимые множества

Подмножество слов S перечислимого множества M называется *разрешимым*, если существует такой алгоритм $\mathbf{S}\colon M \to \{\mathrm{F}, \mathrm{T}\}$, что $\mathbf{S}(m) = \mathrm{T}$ тогда и только тогда, когда $m \in S$. Другими словами, множество разрешимо, если существует алгоритм, который распознает принадлежность элементов этому множеству.

Ясно, что если множество S разрешимо относительно M, то множество $M \setminus S$ также разрешимо — достаточно в теле алгоритма **S** заменить оператор **return** T оператором **return** F и наоборот.

ЛЕММА 1. *Для любого множества X, $\varnothing$ и X разрешимы относительно X.*

Доказательство. Для пустого множества искомый алгоритм — константа F, а для множества X — константа T. □

ЛЕММА 2. *Разрешимое подмножество S перечислимого множества M перечислимо.*

Доказательство. Пусть **S** — разрешающий алгоритм для S, а **M** — перечисляющий алгоритм для M. Построим перечисляющий алгоритм $\mathbf{S}_1$ для S.
$\mathbf{S}_1 :=$ **for** $m \in M$ **do if** $\mathbf{S}(m)$ **then yield** m **end if end for**. □

Отсюда немедленно следует, что всякое разрешимое подмножество натурального ряда перечислимо.

ЛЕММА 3. *Подмножество S перечислимого множества M разрешимо тогда и только тогда, когда перечислимо как множество S, так и его дополнение $M \setminus S$.*

Доказательство.

[$\Rightarrow$] Если множество S разрешимо относительно M, то множество $M \setminus S$ также разрешимо, и остаётся применить лемму 2.

[$\Leftarrow$] Если S или $M \setminus S$ пусто, то по лемме 1 S разрешимо. Пусть теперь оба множества S и $M \setminus S$ не пусты и перечислимы перечисляющими алгоритмами **S** и $\mathbf{S}_1$ соответственно, а вычислимые функции $F, F_1\colon \mathbb{N} \to M$ доставляют элементы этих множеств по номерам. Тогда следующий алгоритм разрешает множество S:

```
func SS(s : M) : {F, T}
n := 0
while true do
  n := n + 1; x := F(n); y := F1(n)
  if x = s then return T end if
  if y = s then return F end if
end while
```

□

ЛЕММА 4. *Если функция вычислима, то образ перечислимого множества перечислим.*

Доказательство. Пусть функция $\mathbf{F}\colon \operatorname{Dom}\mathbf{F} \to \operatorname{Im}\mathbf{F}$ вычислима, а множество $A \subset \operatorname{Dom}\mathbf{F}$ перечислимо, тогда множество $\mathbf{F}(A) \subset \operatorname{Im}\mathbf{F}$ перечисляется следующим алгоритмом:
for $x \in A$ **do yield** $\mathbf{F}(x)$ **end for**. □

ЛЕММА 5. *Если функция вычислима и определена на перечислимом множестве, то прообраз перечислимого множества перечислим.*

Доказательство. Пусть функция $\mathbf{F}\colon \operatorname{Dom}\mathbf{F} \to \operatorname{Im}\mathbf{F}$ вычислима, а множество B перечислимо вычислимой функцией h, и множество $\operatorname{Dom}\mathbf{F}$ перечислимо вычислимой функцией g. Тогда множество $\mathbf{F}^{-1}(B)$ перечисляется следующим алгоритмом: **for** $(a, b) \in \mathbf{N2}$ **do if** $\mathbf{F}(g(a)) = h(b)$ **then yield** $g(a)$ **end if end for**. □

Пример. Конечные множества разрешимы. Бесконечные множества чётных и нечётных, а также простых и составных натуральных чисел разрешимы.

Существуют неразрешимые множества, в том числе неразрешимые подмножества натуральных чисел, поскольку множество алгоритмов счётно, а множество подмножеств натуральных чисел несчётно (п. 1.2.4).

4.5.5. Протокол выполнения алгоритма

Программистская практика свидетельствует, что программы иногда удаётся отладить, то есть найти и устранить ошибки, содержащиеся в алгоритмах. Процесс отладки основан на том, что за работой алгоритма возможно наблюдение, в частности, для любого алгоритма и для любого случая выполнения этого алгоритма можно составить *протокол выполнения*, в котором исчерпывающим образом отражается история выполнения алгоритма (для заданных входных данных). Другими словами, в самих алгоритмах и в их выполнении нет ничего «чудесного», ничего такого, чего нельзя записать словами в алфавите U.

Следующий факт мы принимаем без доказательства, как априорное свойство интуитивного понятия алгоритма.

Аксиома протокола. *Для каждого алгоритма* $\mathbf{A}$ *существует разрешимое множество* H *и существуют вычислимые функции* $a\colon H \to U^*$ *и* $z\colon H \to U^*$, *такие что* $\mathbf{A}(x) = y$ *тогда и только тогда, когда существует такое* $h \in H$, *что* $a(h) = x$ *и* $z(h) = y$.

Множество H — это множество всех возможных протоколов выполнения алгоритма $\mathbf{A}$, h — это протокол выполнения с входом x и выходом y, а функции a и z — это функции определения входов и выходов, соответственно.

ОТСТУПЛЕНИЕ

Если для каждой пары (x, y), для которой $\mathbf{A}(x) = y$ слово $h \in H$, такое что $a(h) = x$ и $z(h) = y$ не только существует, но и единственно, то алгоритм называется *детерминированным*. В протоколе детерминированного алгоритма порядок действий строгий линейный (п. 1.8.2). В сформулированной аксиоме протокола единственность не требуется, тем самым допускается, что выполнение алгоритма может быть недетерминированным. В протоколе недетерминированного алгоритма порядок действий нестрогий частичный. В недетерминированности нет ничего страшного, более того, это путь к параллельному программированию

и другим современным подходам в информационных технологиях. Поэтому накладывать требование детерминированности без необходимости не следует.

СЛЕДСТВИЕ 1. *Область применимости и множество результатов любого алгоритма перечислимы.*

Доказательство. Область применимости — это $a(H)$, а множество результатов — это $z(H)$. Оба эти множества перечислимы по лемме 4. □

СЛЕДСТВИЕ 2. *Область определения и множество значений любой вычислимой функции перечислимы.*

Доказательство. Частный случай следствия 1. □

СЛЕДСТВИЕ 3. *График любой вычислимой функции перечислим.*

Доказательство. По аксиоме протокола для функции существует множество протоколов H и вычислимые функции входа/выхода $a(h)$ и $z(h)$. Тогда следующий алгоритм перечисляет график функции: **for** $h \in H$ **do yield** $(a(h), z(h))$ **end for** □

СЛЕДСТВИЕ 4. *Функция с перечислимым графиком вычислима.*

Доказательство. Пусть алгоритм **F** перечисляет график функции f, то есть множество $\{(a, b) \mid a = f(b)\}$. Тогда следующий алгоритм вычисляет значение функции $f(x)$: **for** $(a, b) \in \mathbf{F}$ **do if** $x = a$ **then return** b **end for** □

4.5.6. Программы

Алгоритмы могут быть записаны как слова в универсальном алфавите U, причём такая запись заведомо не является единственной. Возникает естественный вопрос: если даны две записи, причём обе синтаксически распознаются как записи алгоритмов, то что можно сказать о свойствах алгоритмов по их записям? Например, это две записи одного алгоритма или разных алгоритмов? Ясно, что ответить на эти вопросы не так просто. Необходимо внести уточнения в описание способа записи алгоритмов.

Введём обозначения. Напомним, что мы рассматриваем алгоритмы над словами в алфавите U, то есть алгоритмы принимают в качестве аргументов слова (возможно, не принимают аргументов) и вырабатывают в качестве результата также слова (возможно, много слов).

Алгоритмы, в том числе вычислимые функции, не обязательно тотальны. Если аргументы принадлежат области определения алгоритма, то говорят, что алгоритм *применим*.

Говорят, что два алгоритма **A** и **B** *условно равны* для аргумента x, обозначение $\mathbf{A}(x) \simeq \mathbf{B}(x)$, если либо оба алгоритма применимы к аргументу x, и тогда их результаты совпадают, либо оба алгоритма неприменимы к x.

ЗАМЕЧАНИЕ

Алгоритмы в этом определении являются не только алгоритмами вычислимых функций, но любыми алгоритмами. Поэтому результатами работы может быть не только возвращаемое значение, но и перечислимое множество, или последовательность действий в случае итераторов, или изменение состояния объекта в случае трансформаций и т. д. Совпадение результатов для каждого типа алгоритмов определяется специфическим образом. Для вычислимых функций это равенство слов, для перечислимых множеств — равенство множеств слов.

Два алгоритма $\mathbf{A}$ и $\mathbf{B}$ называются *равносильными*, если у них совпадают области применимости, и для любого слова из этой области, взятого в качестве аргумента, совпадают результаты работы алгоритмов, то есть $\forall x \ \ (\mathbf{A}(x) \simeq \mathbf{B}(x))$. Равносильность обычно обозначается следующим образом: $\mathbf{A} \sim \mathbf{B}$. Ясно, что равносильность алгоритмов является эквивалентностью.

ЗАМЕЧАНИЕ

Все эти определения применимы к любым алгоритмам, в частности, к вычислимым функциям. Поэтому равносильность и функциональная эквивалентность — это равнообъёмные понятия.

Говорят, что два алгоритма $\mathbf{A}$ и $\mathbf{B}$ *всюду отличаются*, если $\neg\exists x \ \ (\mathbf{A}(x) \simeq \mathbf{B}(x))$. Другими словами, алгоритмы всюду отличаются, если дают разные результаты там, где применимы, и нет таких аргументов, при которых они оба неприменимы.

Пусть имеется алгоритм с двумя параметрами $\mathbf{A}\colon X \times Y \to Z$. *Проекцией* этого алгоритма на первый (второй) параметр, или *остаточной программой* по первому (второму) аргументу, называется алгоритм $\mathbf{A}_x\colon Y \to Z$ ($\mathbf{A}_y\colon X \to Z$) такой, что $\forall y \in Y \ \ (\mathbf{A}_x(y) \simeq \mathbf{A}(x, y))$ (соответственно $\forall x \in X \ \ (\mathbf{A}_y(x) \simeq \mathbf{A}(x, y))$).

ЗАМЕЧАНИЕ

Термин *остаточная программа* заимствован из теории смешанных вычислений. Концепцию смешанных вычислений предложил А. П. Ершов[1].

Пусть есть некоторый класс алгоритмов $\mathcal{A}$. Класс алгоритмов $\mathcal{B}$ называется *представительным* для класса $\mathcal{A}$, если для любого алгоритма $\mathbf{A} \in \mathcal{A}$ существует равносильный алгоритм $\mathbf{B} \in \mathcal{B}$, $\mathbf{A} \sim \mathbf{B}$.

ЗАМЕЧАНИЕ

Отношение представительности между классами алгоритмов не является эквивалентностью, но является рефлексивным и транзитивным, то есть является предпорядком (п. 1.8.1).

Идея состоит в том, чтобы для любого класса алгоритмов $\mathcal{A}$ подобрать более чётко очерченный (но, возможно, более широкий!) класс алгоритмов $\mathcal{B}$, который будет представительным по отношению к исходному классу. Элементы класса $\mathcal{B}$ называются *программами*, которые *представляют* (или *реализуют*) элементы класса $\mathcal{A}$.

Осталось определить, в каком смысле класс программ должен быть чётко очерчен. Во-первых, класс программ должен быть разрешимым — требуется надёжно отличать программы от не программ. Во-вторых, должен быть задан *алгоритм интерпретации* $\mathbf{I}$, позволяющей по программе $\mathbf{B}$ и исходным данным x определить результаты представляемого алгоритма $\mathbf{A}(x)$ — программы необходимо уметь выполнять.

[1] Андрей Петрович Ершов (1931–1988).

ЗАМЕЧАНИЕ

Этой схеме следуют все упомянутые выше теоретические уточнения понятия алгоритма. Каждое такое уточнение описывает свой класс «программ» (более или менее формально), неформально описывает алгоритм $\mathbf{I}$ и в качестве недоказываемой догмы постулирует представительность выбранного класса программ. Но этой же схеме следуют все практические системы программирования! Каждая такая система описывает свой класс программ (задает *синтаксис* языка программирования), полуформально описывает алгоритм $\mathbf{I}$ (операционная *семантика* языка программирования), а «выразительная сила» языка программирования рассматривается как предмет веры. В том, что класс программ шире класса алгоритмов (если программа синтаксически правильна, это ещё не значит, что она закончит работу и даст правильный результат), программисты очень быстро убеждаются на собственном печальном опыте.

Всё сказанное приводит к следующей аксиоме.

Аксиома программы. *Существует представительное разрешимое множество программ* $\mathcal{P}$ *такое, что для любого алгоритма* $\mathbf{A}\colon T_1 \to T_2$ *существует равносильная программа* $\mathbf{P} \in \mathcal{P}$, $\mathbf{A} \sim \mathbf{P}$ *и существует алгоритм интерпретации* $\mathbf{I}\colon \mathcal{P} \times T_1 \to T_2$, *причём* $\mathbf{A} \sim \mathbf{I_P}$.

ЗАМЕЧАНИЕ

Алгоритм $\mathbf{I}$, как и все алгоритмы, может быть запрограммирован, то есть для него может быть предъявлена равносильная программа. Полученная программа является самоприменимой. В этом нет ничего парадоксального.

4.5.7. Невычислимые функции и неразрешимые множества

Рассмотрим более подробно важное понятие *универсальной функции*, или *универсального алгоритма*. Рассмотрим произвольный класс F вычислимых функций $f\colon X \to Y$. Функция $U_F\colon F \times X \to Y$ называется универсальной для класса F, если $\forall f\ (\forall x\ (U_F(f, x) \simeq f(x)))$. Другими словами, там, где исходная функция определена, универсальная функция вычисляет значение по функции и по аргументу, а там, где функция не определена, универсальная функция также не определена.

Примеры

1. Алгоритм в п. 3.5.3 является универсальным алгоритмом в классе булевых функций n переменных.
2. Алгоритм в п. 3.2.1 является универсальным алгоритмом в классе функций, заданных формулами в базисе F.
3. Интерпретатор $\mathbf{I}$ является универсальным алгоритмом в классе программ.

Множество программ, представляющих вычислимые функции, перечислимо, и их можно перенумеровать. Интуитивно ясно, что возможно построить программу (например, в форме «очень длинного» оператора **switch**), которая по номеру функции и значению аргумента будет вычислять значение этой функции. Таким образом, из аксиомы программы имеем следующие следствия.

СЛЕДСТВИЕ 1. *Каков бы ни был класс* F *вычислимых функций* $T_1 \to T_2$, *существует вычислимая функция* $U_F\colon \mathbb{N} \times T_1 \to T_2$, *универсальная в этом классе.*

В данном разделе наибольший интерес представляют вычислимые функции с натуральными аргументами и результатами.

СЛЕДСТВИЕ 2. *Существует вычислимая функция $U_N\colon \mathbb{N}\times\mathbb{N}\to\mathbb{N}$, универсальная в классе вычислимых функций $\mathbb{N}\to\mathbb{N}$.*

Доказательство. Положим $T_1 := \mathbb{N}$ и $T_2 := \mathbb{N}$ и применим следствие 1. □

СЛЕДСТВИЕ 3. *Существует такая вычислимая функция $d\colon \mathbb{N}\to\mathbb{N}$, что никакая вычислимая функция не может отличаться от неё всюду.*

Доказательство. Рассмотрим «диагональную» функцию $d(n) := U_N(n,n)$. Пусть f_n — функция, имеющая номер n. Тогда если $f_n(n)$ определено, то $f_n(n) = d(n)$ и $f_n(n) \simeq d(n)$, если же $f_n(n)$ не определено, то и $d(n)$ не определено, и все равно $f_n(n) \simeq d(n)$. □

Предыдущее доказательство кажется парадоксом: действительно, если у нас есть функция, которая претендует на то, что, неформально говоря, её график «пересекается» с графиками всех других функций, то, казалось бы, возьмём функцию, график которой «параллелен» графику «претендента», и получится контрпример к следствию 3. Но никакого парадокса и контрпримера нет! Дело в том, что мы рассматриваем *все* вычислимые функции, в том числе и частичные, и «пересечение» получается там, где функции не определены.

Но если вычислимая функция определена частично, то можно доопределить её произвольным образом, назначив какие-то значения функции для тех значений аргументов, где функция не определена. Такое «доопределение» называется *всюду определённым продолжением*.

СЛЕДСТВИЕ 4. *Существует вычислимая функция $d_1\colon \mathbb{N}\to\mathbb{N}$, не имеющая всюду определённого вычислимого продолжения.*

Доказательство. Рассмотрим «параллельную» функцию $d_1(n) := d(n)+1$. Эта функция определена частично, потому что функция $d(n)$ определена частично. Рассмотрим $D_1(n)$ — всюду определённое продолжение $d_1(n)$. Как именно продолжается d_1 — несущественно. Функция D_1 всюду определена и всюду отличается от $d(n)$. По следствию 3 заключаем, что функция $D_1(n)$ невычислима. □

Таким образом, построена всюду определённая, но невычислимая функция.

СЛЕДСТВИЕ 5. *Существует неразрешимое перечислимое подмножество натурального ряда.*

Доказательство. Достаточно рассмотреть множество $N := \operatorname{Dom} d_1$. Множество N перечислимо по лемме 5 (п. 4.5.4). Допустим, что множество N разрешимо некоторым алгоритмом $\mathbf{N}$. Тогда построим функцию
$d_2(x) :=$ **if** $\mathbf{N}(x)$ **then** $d_1(x)$ **else** 1 **end if**. Функция $d_2(x)$ оказывается всюду определённым вычислимым продолжением функции d_1, что невозможно в силу следствия 4, а значит, алгоритма $\mathbf{N}$ не существует. □

Таким образом, построено перечислимое, но неразрешимое множество.

СЛЕДСТВИЕ 6. *Существует перечислимое подмножество натурального ряда с неперечислимым дополнением.*

Доказательство. Рассмотрим построенное перечислимое неразрешимое множество N. Его дополнение $\mathbb{N} \setminus N$ неперечислимо, в противном случае N было бы разрешимо по лемме 3 (п. 4.5.4). □

Таким образом, построено перечислимое множество с неперечислимым дополнением.

4.5.8. Истинность и доказуемость

К данному моменту накоплен достаточный запас фактов, чтобы показать применимость и полезность наивной теории алгоритмов в форме доказательства первой теоремы Гёделя о неполноте.

В первой теореме Гёделя о неполноте (п. 4.4.9) речь идет об «истинности» и «доказуемости». Чтобы доказывать какие-либо утверждения относительно этих понятий, необходимо придать им математический смысл.

Понятие *доказательства* является едва ли не самым главным в математике, но формально определить его даже труднее, чем понятия множества и алгоритма. Прекрасное определение: *доказательство* — это рассуждение, убеждающее нас настолько, что с его помощью мы готовы убеждать других. Однако это определение принадлежит скорее психологии, нежели математике. Ничуть не проще для математического определения и понятие *истины*. Попытки раскрыть это понятие во всей полноте уводят нас в философию, языкознание, теологию и в другие очень интересные и поучительные области мысли, которые, однако, далеки от изучаемого предмета: дискретной математики для программистов.

Выход состоит в том, чтобы зафиксировать неопределяемые понятия в виде несложно устроенных конструктивных объектов и далее манипулировать этими объектами средствами наивной теории множеств и наивной теории алгоритмов. То, что получится, можно назвать *наивной теорией доказательства*.

В данном случае множество истинных утверждений, или *множество истин*, рассматривается просто как некоторое множество слов T в универсальном алфавите U. Далее, множество доказательств также рассматривается просто как некоторое разрешимое множество слов D в универсальном алфавите U. Элементы множества D называются *доказательствами*. Других ограничений на множества T и D пока не накладывается. Наряду с разрешимым множеством D постулируется наличие вычислимой функции $E\colon D \to U^*$, называемой *функцией извлечения доказанного*. Если $s = E(d)$, где $d \in D$, то s называется *теоремой*. Таким образом, $S = E(D)$ — множество теорем. Пара $\langle D; E\rangle$ называется *дедуктикой*.

Пример. Формальные теории, рассматриваемые в предыдущих разделах, являются дедуктиками. Множество D в них строится как последовательности слов, получаемых из «аксиом» по «правилам вывода» (проверка корректности вывода — это и есть разрешающий алгоритм), а функция извлечения доказанного E очень проста и, очевидно, вычислима: последнее слово в доказательстве является теоремой.

Нас интересует, как соотносятся теоремы и истины в зависимости от того, какими свойствами обладают множества T и D.

Примеры

Рассмотрим некоторые примеры формулировок свойств интересующих нас множеств на естественном языке.

1. *Существуют недоказуемые истины.* Это утверждение тривиально и неинтересно. Достаточно положить $D := \varnothing$, и при любом T все истины окажутся недоказуемыми.
2. *Не существует дедуктики, в которой все истины доказуемы.* А это утверждение просто неверно. Достаточно положить $D := T$, $E(t) := t$ и при любом множестве истин T все истины окажутся легко доказуемыми.

Дедуктика $\langle D; E\rangle$ называется *непротиворечивой* относительно множества истин T, если $E(D) \subset T$ (то есть $S \subset T$, всё доказуемое истинно). Дедуктика $\langle D; E\rangle$ называется *полной* относительно множества истин T, если $T \subset E(D)$ (то есть $T \subset S$, всё истинное доказуемо).

Непротиворечивая и одновременно полная дедуктика называется *адекватной* множеству истин T, в такой дедуктике множество теорем совпадает с множеством истин, $T = S$.

ЗАМЕЧАНИЕ

Разрешимое множество доказательств D является подмножеством множества всех слов, которое перечислимо. Значит, по лемме 2 множество D перечислимо. Далее, множество теорем S является образом перечислимого множества при применении вычислимой функции. Значит, по лемме 4 множество S также перечислимо.

Пример. Адекватные дедуктики для интересных множеств истин существуют: построение таблиц истинности является адекватной дедуктикой для тавтологий в исчислении высказываний (п. 4.3.8).

Вопрос состоит в том, при каких условиях, налагаемых на множество истин T, может не найтись адекватной дедуктики?

ТЕОРЕМА. *Если множество истин T перечислимо, то существует дедуктика $\langle D; E\rangle$, адекватная T.*

Доказательство. Если $T = \varnothing$, то достаточно положить $D := \varnothing$. Пусть $T \neq \varnothing$ и $\mathbf{T}\colon \mathbb{N} \to T$ перечисляющий алгоритм в форме вычислимой функции, вычисляющей истину по её номеру. Тогда положим $D := \mathbb{N}$ и $E := \mathbf{T}$. □

Другими словами, если множество истин перечислимо, и может быть расположено в определённой последовательности, то доказательством истины является указание номера истины в этой последовательности.

СЛЕДСТВИЕ. *Если множество истин T неперечислимо, то не существует дедуктики $\langle D; E\rangle$, адекватной T.*

Доказательство. Множество теорем S любой дедуктики перечислимо, а потому не может совпадать с неперечислимым множеством истин T. □

Итак, ответ на основной вопрос получен — если множество истин можно задать алгоритмом, то адекватная дедуктика может быть построена и всё в порядке: всё истинное доказуемо и всё доказуемое истинно. Если же множество истин невозможно задать алгоритмически, то и нет надежды на построение адекватной дедуктики (системы доказательств). Но, может быть, всё не так плохо и неалгоритмические множества истин нам просто не понадобятся на практике? Увы, ответ отрицательный — неалгоритмические множества истин нам совершенно необходимы.

4.5.9. Множество истин арифметики

Сначала установим важный факт, относящийся к числовым множествам. Говорят, что множество натуральных чисел $N \subset \mathbb{N}$ *выразимо* с помощью множества истин T, если существует такая вычислимая функция $\mathbf{F}\colon \mathbb{N} \to U^*$, что $n \in N$ тогда и только тогда, когда $\mathbf{F}(n) \in T$.

ТЕОРЕМА 1. *Множество истин T выражает хотя бы одно неперечислимое множество натуральных чисел N тогда и только тогда, когда не существует дедуктики $\langle D; E\rangle$, адекватной T.*

Доказательство.

[$\Longrightarrow$] Пусть множество N неперечислимо и выразимо множеством истин T с помощью вычислимой функции $\mathbf{F}\colon \mathbb{N} \to U^*$. Имеем $N = \mathbf{F}^{-1}(\operatorname{Im}\mathbf{F} \cap T)$. Но множество значений $\operatorname{Im}\mathbf{F}$ вычислимой функции натурального ряда перечислимо по лемме 4 (п. 4.5.4). Значит, множество T неперечислимо, иначе множество $\operatorname{Im}\mathbf{F} \cap T$ было бы перечислимо и множество N было бы перечислимо по лемме 5 (п. 4.5.4). Для неперечислимого множества истин адекватной дедуктики не существует.

[$\Longleftarrow$] Пусть для множества истин T адекватной дедуктики не существует. Тогда множество T неперечислимо. Но множество U^* перечислимо, пусть оно перечисляется вычислимой функцией $\mathbf{F}\colon \mathbb{N} \to U^*$. Имеем $T = \mathbf{F}(\check{\mathbf{F}}^{-1}(T))$. Положим $N := \mathbf{F}^{-1}(T)$. Заключаем по лемме 5 (п. 4.5.4), что множество N неперечислимо и выразимо множеством истин T. □

Данный факт сводит несуществование адекватной дедуктики к выразимости неперечислимых числовых множеств. Но, может быть, ещё не всё потеряно, и такие множества истин нам просто не нужны и про них можно забыть? К сожалению, это не так. Неприятные случаи совсем рядом.

Рассмотрим в качестве примера множество истин арифметики.

Арифметика — понятие натуральных чисел, операции сложения, умножения, отношения между натуральными числами — это такой элемент общей культуры (не только раздел математики), без которого существование современного человечества трудно себе представить. Как и ранее, обсуждение вопросов о том, является ли арифметика откровением, данным Богом, или же это итог развития науки, является ли способность к счёту врожденной или благоприобретённой, увели бы нас далеко от рассматриваемого предмета: дискретной математики для программистов.

Зафиксируем язык арифметики, определённый в п. 4.4.7. В этом языке используются: цифры для записи натуральных чисел, буквы для записи переменных, операции сложения $+$ и умножения $\cdot$, отношение $=$, логические связки $\neg$, $\to$ и кванторы $\forall$, $\exists$. Далее из этих символов в языке арифметики допускается строить следующие конструкции.

1. *Термы.* Все числа и переменные суть термы. Выражения вида $(t + u)$ и $(t \cdot u)$, где t и u суть термы, также суть термы. Все переменные в терме являются его *параметрами*. Терм без параметров называется *константой*.
2. *Атомы.* Выражения вида $(t = u)$, где t и u суть термы, называются атомами, или *элементарными формулами*. Множество параметров атомов является объединение параметров термов.
3. *Формулы.* Если A и B суть формулы, а x — переменная, то $\neg A$, $A \to B$, $\forall x\ (A)$, $\exists x\ (A)$ — формулы. Кванторы исключают свои переменные из числа параметров подкванторной формулы.

Далее формулам без параметров (*суждениям*) сопоставляются истинностные значения T и F по правилам п. 4.4.2. Все суждения арифметики, имеющие истинностное значение T, образуют *множество истин арифметики* T. Например, $2 + 2 = 4$.

Охарактеризуем класс *арифметических множеств*. Рассмотрим класс формул языка арифметики не более чем с одним параметром. Будем обозначать такие формулы $\alpha(x)$. Тогда для любого конкретного $x \in \mathbb{N}$ формула $\alpha(x)$ — суждение, и имеет истинностное значение. Множество $M := \{x \in \mathbb{N} \mid \alpha(x) = \mathrm{T}\}$ называется *сопряжённым* с формулой $\alpha(x)$. Множество, сопряжённое с некоторой формулой, называется *арифметическим*. Арифметические множества обладают рядом полезных свойств, зафиксированных в виде следующих теорем.

ТЕОРЕМА 2. *Дополнение к арифметическому множеству есть арифметическое множество.*

Доказательство. Если M сопряжено с $\alpha(x)$, то $\mathbb{N} \setminus M$ сопряжено с $\neg\alpha(x)$. □

ТЕОРЕМА 3. *Объединение и пересечение арифметических множеств суть арифметические множества.*

Доказательство. Если M сопряжено с $\alpha(x)$, N сопряжено с $\beta(x)$, то $M \cup N$ сопряжено с $\alpha(x) \vee \beta(x)$, $M \cap N$ сопряжено с $\alpha(x) \wedge \beta(x)$. □

ТЕОРЕМА 4. *Арифметическое множество выразимо множеством истин арифметики.*

Доказательство. Пусть M сопряжено с $\alpha(x)$, определим $\mathbf{F}\colon \mathbb{N} \to \{\mathrm{F}, \mathrm{T}\}$ следующим образом: $\mathbf{F}(x) := \alpha(x)$. □

Арифметика, как система операций с целыми (и натуральными) числами, реализована в подавляющем большинстве компьютеров. Опыт программирования этих компьютеров приводит к убеждению, что любую вычислимую функцию натурального ряда можно запрограммировать, если отвлечься от того обстоятельства, что память компьютера конечна и слишком большие числа могут не поместиться. Впрочем, память можно увеличить и считать потенциально бесконечной. Следовательно, всякое перечислимое множество натуральных чисел (как множество значений вычислимой функции) может быть запрограммировано с помощью машинных операций. Основываясь на этих соображениях, мы принимаем, что имеет место **Аксиома арифметичности.** *Всякое перечислимое множество натуральных чисел является арифметическим.*

Принятая аксиома позволяет в несколько шагов закончить доказательство первой теоремы Гёделя о неполноте.

1. *Существует неперечислимое арифметическое множество.*

   Доказательство. Рассмотрим множество N, построенное в доказательстве следствия 5 п. 4.5.7. Это множество перечислимо и имеет неперечислимое дополнение по следствию 6. Но само множество N арифметическое по аксиоме арифметичности и его дополнение арифметично по теореме 2. □

2. *Существует неперечислимое множество, выразимое множеством истин арифметики.*
   Доказательство. По теореме 4. □

3. *Не существует дедуктики, адекватной множеству истин арифметики.*
   Доказательство. По теореме 1. □

Таким образом, для любого точно сформулированного понятия доказательства найдется либо доказуемое, но ложное утверждение, формулируемое на языке арифметики, либо истинное утверждение того же языка, не являющееся доказуемым.

4.6. Автоматическое доказательство теорем

Автоматическое доказательство теорем — это краеугольный камень логического программирования, искусственного интеллекта и других современных направлений в программировании. Здесь излагаются основы *метода резолюций* — классического (и в то же время до сих пор популярного) метода автоматического доказательства теорем, предложенного Робинсоном[1] в 1965 году.

4.6.1. Постановка задачи автоматического доказательства теорем

Алгоритм, который проверяет отношение $\Gamma \vdash_{\mathfrak{T}} S$ для формулы S, множества формул Γ и теории $\mathfrak{T}$, называется алгоритмом *автоматического доказательства теорем*. В общем случае такой алгоритм невозможен, то есть не существует алгоритма, который для любых S, Γ и $\mathfrak{T}$ выдавал бы ответ «Да», если $\Gamma \vdash_{\mathfrak{T}} S$, и ответ «Нет», если неверно, что $\Gamma \vdash_{\mathfrak{T}} S$. Более того, известно, что нельзя построить алгоритм автоматического доказательства теорем даже для большинства конкретных достаточно богатых формальных теорий $\mathfrak{T}$. В некоторых случаях удаётся построить алгоритм автоматического доказательства теорем, который применим не ко всем формулам теории (то есть частичный алгоритм, п. 4.2.5.)

Для некоторых простых формальных теорий (например, для исчисления высказываний) и некоторых простых классов формул (например, для прикладного исчисления предикатов с одним одноместным предикатом) алгоритмы автоматического доказательства теорем известны.

Пример. Поскольку для исчисления высказываний известно, что теоремами являются общезначимые формулы, то достаточно вычислить истинностное значение формулы при всех возможных интерпретациях (их конечное число). Если во всех случаях получится значение T, то проверяемая формула — тавтология и, следовательно, является теоремой теории $\mathcal{L}$. Если же хотя бы в одном случае получится значение F, то проверяемая формула не является тавтологией и, следовательно, не является теоремой теории $\mathcal{L}$.

ЗАМЕЧАНИЕ

Приведённый выше пример является алгоритмом автоматического доказательства теорем в теории $\mathcal{L}$, хотя и не является алгоритмом автоматического *поиска вывода* теорем из аксиом теории $\mathcal{L}$.

[1] Джон А. Робинсон (род. 1930).

Наиболее известный классический алгоритм автоматического доказательства теорем называется *методом резолюций*. Для любого прикладного исчисления предикатов первого порядка $\mathcal{T}$, любой формулы S и множества формул Γ теории $\mathcal{T}$ метод резолюций выдаёт ответ «Да», если $\Gamma \vdash_{\mathcal{T}} S$, и выдаёт ответ «Нет» или не выдаёт никакого ответа (то есть «зацикливается»), если неверно, что $\Gamma \vdash_{\mathcal{T}} S$.

4.6.2. Доказательство от противного

В основе метода резолюций лежит идея «доказательства от противного».

ТЕОРЕМА. *Если* $\Gamma, \neg S \vdash F$, *где* F *— любое противоречие (тождественно ложная формула), то* $\Gamma \vdash S$.

Доказательство. (для случая $\mathcal{L}$) $\Gamma, \neg S \vdash F \iff \Gamma \,\&\, \neg S \to F$ — тавтология. Но $\Gamma \,\&\, \neg S \to F = \neg(\Gamma \,\&\, \neg S) \vee F = \neg(\Gamma \,\&\, \neg S) = \neg\Gamma \vee S = \Gamma \to S$. Имеем $\Gamma \to S$ — тавтология $\iff \Gamma \vdash S$. □

Пустая формула не имеет никакого значения ни в какой интерпретации, в частности, не является истинной ни в какой интерпретации и по определению является противоречием. В качестве формулы F при доказательстве от противного по методу резолюций принято использовать пустую формулу. Полезно сравнить использование пустой формулы в методе резолюций с п. 3.4.3.

ЗАМЕЧАНИЕ

При изложении метода резолюций пустая формула традиционно обозначается □. Однако значок □ используется в книге также для обозначения конца доказательства теоремы или обоснования алгоритма. Необходимо по контексту различать, в каком смысле использован значок □ в каждом случае.

4.6.3. Сведéние к дизъюнктам

Метод резолюций работает с особой стандартной формой формул, которые называются *дизъюнктами*, или *предложениями*. Дизъюнкт — это бескванторная дизъюнкция литералов. Любая формула исчисления предикатов может быть преобразована в множество дизъюнктов следующим образом (здесь знак $\mapsto$ используется для обозначения способа преобразования формул):

1. *Элиминация импликации*. Преобразование:
   $A \to B \mapsto \neg A \vee B$.
   После первого этапа формула содержит только $\neg, \vee, \&, \forall, \exists$.
2. *Протаскивание отрицаний*. Преобразования:
   $\neg\forall x\ (A) \mapsto \exists x\ (\neg A)$, $\neg\exists x\ (A) \mapsto \forall x\ (\neg A)$, $\neg\neg A \mapsto A$,
   $\neg(A \vee B) \mapsto \neg A \,\&\, \neg B$, $\neg(A \,\&\, B) \mapsto \neg A \vee \neg B$.
   После второго этапа формула содержит отрицания только перед атомами.
3. *Разделение связанных переменных*. Преобразование:
   $Q_1 x\ A(\dots Q_2 x\ B(\dots x \dots)\dots) \mapsto Q_1 x\ A(\dots Q_2 y\ B(\dots y \dots)\dots)$,
   где Q_1 и Q_2 — любые кванторы. После третьего этапа формула не содержит случайно совпадающих связанных переменных.
4. *Приведение к предварённой форме*. Преобразования:
   $Q x\ A \vee B \mapsto Q x\ (A \vee B)$, $Q x\ A \,\&\, B \mapsto Q x\ (A \,\&\, B)$,
   где Q — любой квантор. После четвёртого этапа формула находится в предварённой форме.

5. *Элиминация кванторов существования* (*сколемизация*). Преобразования:
$\exists x_1\ (Q_2\, x_2\ \ldots Q_n\, x_n\ A(x_1, x_2, \ldots, x_n)) \mapsto Q_2\, x_2 \ldots Q_n\, x_n\ A(a, x_2, \ldots, x_n)$,
$\forall x_1\ (\ldots \forall x_{i-1}\ (\exists x_i\ (Q_{i+1}\, x_{i+1} \ldots Q_n\, x_n A(x_1, \ldots, x_i, \ldots, x_n))) \ldots) \mapsto$
$\forall x_1\ (\ldots \forall x_{i-1}\ (Q_{i+1}\, x_{i+1} \ldots Q_n\, x_n A(x_1, \ldots, f(x_1, \ldots, x_{i-1}), \ldots, x_n)) \ldots)$,
где a — новая предметная константа, f — новый функциональный символ, а $Q_1, Q_2, \ldots, Q_n$ — любые кванторы. После пятого этапа формула содержит только кванторы всеобщности.
6. *Элиминация кванторов всеобщности*. Преобразование:
$\forall x\ (A(x)) \mapsto A(x)$.
После шестого этапа формула не содержит кванторов.
7. *Приведение к конъюнктивной нормальной форме*. Преобразования:
$A \vee (B\ \&\ C) \mapsto (A \vee B)\ \&\ (A \vee C)$, $\quad (A\ \&\ B) \vee C \mapsto (A \vee C)\ \&\ (B \vee C)$.
После седьмого этапа формула находится в конъюнктивной нормальной форме.
8. *Элиминация конъюнкции*. Преобразование:
$A\ \&\ B \mapsto A, B$.
После восьмого этапа формула распадается на множество дизъюнктов.

Не все преобразования на этапах 1–8 являются логически эквивалентными.

ТЕОРЕМА. *Если* Γ *— множество дизъюнктов, полученных из формулы* S*, то* S *является противоречием тогда и только тогда, когда множество* Γ *невыполнимо.*

Доказательство. В доказательстве нуждается шаг 5 — сколемизация. Пусть $F := \forall x_1\ (\ldots \forall x_{i-1}\ (\exists x_i Q_{i+1} x_{i+1} \ldots Q_n x_n\ (A(x_1, \ldots, x_n))) \ldots)$. Положим
$F' := \forall x_1\ (\ldots \forall x_{i-1}\ (Q_{i+1} x_{i+1} \ldots Q_n x_n A(x_1, \ldots, x_{i-1}, f(x_1, \ldots, x_{i-1}), x_{i+1}, \ldots, x_n) \ldots))$
Пусть F — противоречие, а F' — не противоречие. Тогда существуют интерпретация I и набор значений $s = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$, такие, что $s^*(F') = \text{Т}$. Положим $a_i := f(a_1, \ldots, a_{i-1})$, $s_1 := (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$. Тогда $s_1^*(F) = \text{Т}$, и F — выполнимая формула. □

ЗАМЕЧАНИЕ

Невыполнимость множества формул Γ означает, что множество Γ не имеет модели, то есть не существует интерпретации, в которой все формулы Γ имели бы значение Т.

4.6.4. Правило резолюции для исчисления высказываний

Метод резолюций основан на использовании специального правила вывода.

Пусть C_1 и C_2 — два дизъюнкта в исчислении высказываний и пусть $C_1 = P \vee C_1'$, а $C_2 = \neg P \vee C_2'$, где P — пропозициональная переменная, а C_1', C_2' — любые дизъюнкты (в частности, может быть, пустые или состоящие только из одного литерала). Правило вывода

$$\frac{C_1,\ C_2}{C_1' \vee C_2'}\ (R)$$

называется *правилом резолюции*. Дизъюнкты C_1 и C_2 называются *резольвируемыми* (или *родительскими*), дизъюнкт $C_1' \vee C_2'$ — *резольвентой*, а формулы P и $\neg P$ — *контрарными* литералами.

ЗАМЕЧАНИЕ

Резольвента является дизъюнктом, то есть множество дизъюнктов замкнуто относительно правила резолюции.

ТЕОРЕМА 1. *Правило резолюции логично, то есть резольвента является логическим следствием резольвируемых дизъюнктов.*

Доказательство. Пусть $I(C_1) = \mathrm{T}$ и $I(C_2) = \mathrm{T}$. Тогда если $I(P) = \mathrm{T}$, то $C_2' \neq \varnothing$ и $I(C_2') = \mathrm{T}$, а значит, $I(C_1' \vee C_2') = \mathrm{T}$. Если же $I(P) = \mathrm{F}$, то $C_1' \neq \varnothing$ и $I(C_1') = \mathrm{T}$, а значит, $I(C_1' \vee C_2') = \mathrm{T}$. □

Правило резолюции — это очень мощное правило вывода. Многие ранее рассмотренные правила являются его частными случаями:

$$\frac{A,\ A \to B}{B}\ \text{Modus ponens} \qquad \frac{A,\ \neg A \vee B}{B}\ R,$$

$$\frac{A \to B,\ B \to C}{A \to C}\ \text{Транзитивность} \qquad \frac{\neg A \vee B,\ \neg B \vee C}{\neg A \vee C}\ R,$$

$$\frac{A \vee B,\ A \to B}{B}\ \text{Слияние} \qquad \frac{A \vee B,\ \neg A \vee B}{B}\ R.$$

ТЕОРЕМА 2. *Правило резолюции полно, то есть всякая тавтология может быть выведена с использованием только правила резолюции.*

Доказательство. Поскольку Modus ponens является частным случаем правила резолюции. □

4.6.5. Правило резолюции для исчисления предикатов

Для применения правила резолюции нужны контрарные литералы в резольвируемых дизъюнктах. В случае исчисления высказываний контрарные литералы определяются очень просто: это пропозициональная переменная и её отрицание. Для исчисления предикатов определение чуть сложнее.

Пусть C_1 и C_2 — два дизъюнкта в исчислении предикатов. Правило вывода

$$\frac{C_1,\ C_2}{(C_1' \vee C_2')\sigma}\,(R)$$

называется правилом резолюции в исчислении предикатов, если в дизъюнктах C_1 и C_2 существуют унифицируемые контрарные литералы P_1 и P_2, то есть $C_1 = P_1 \vee C_1'$ и $C_2 = \neg P_2 \vee C_2'$, причём атомарные формулы P_1 и P_2 унифицируются наиболее общим унификатором σ. В этом случае резольвентой дизъюнктов C_1 и C_2 является дизъюнкт $(C_1' \vee C_2')\sigma$, полученный из дизъюнкта $C_1' \vee C_2'$ применением унификатора σ.

Пример. Пусть заданы дизъюнкты $P(x,a) \vee Q(x,y)$ и $\neg P(b,y) \vee R(y,x)$, где P, Q, R — двухместные предикаты, а a и b — константы. Тогда применение правила резолюции имеет вид

$$\frac{P(x,a) \vee Q(x,y,) \qquad \neg P(b,y) \vee R(y,x)}{Q(b,a) \vee R(a,b)}\,(R),$$

при этом наиболее общий унификатор $\sigma = [b/x, a/y]$.

Если нет контрарных литералов, то правило резолюции неприменимо.

Пример. Пусть заданы дизъюнкты $P(x,a) \vee Q(x,y)$ и $P(b,y) \vee \neg R(y,x)$, где P, Q, R — двухместные предикаты, а a и b — константы. Тогда правило резолюции не применимо, поскольку литералы $P(x,a)$ и $P(b,y)$ хотя и унифицируемы, но не контрарны и других контрарных литералов нет.

Следует ещё раз подчеркнуть, что наличия атома и его отрицания в разных дизъюнктах ещё недостаточно: необходимо, чтобы контрарные литералы были унифицируемыми.

Пример. Пусть заданы дизъюнкты $P(x,a) \vee Q(x,y)$ и $\neg P(x,b) \vee R(y,x)$, где P, Q, R — двухместные предикаты, а a и b — константы. Тогда правило резолюции не применимо, поскольку литералы $P(x,a)$ и $P(x,b)$ не унифицируемы.

4.6.6. Опровержение методом резолюций

Опровержение методом резолюций — это алгоритм автоматического доказательства теорем в прикладном исчислении предикатов, который сводится к следующему. Пусть нужно установить выводимость $S \vdash G$. Каждая формула множества S и формула $\neg G$ (отрицание целевой теоремы) *независимо* преобразуются в множества дизъюнктов. В полученном совокупном множестве дизъюнктов C отыскиваются резольвируемые дизъюнкты, к ним применяется правило резолюции и резольвента добавляется в множество дизъюнктов до тех пор, пока не будет получен пустой дизъюнкт. При этом возможны три случая:

1. Среди текущего множества дизъюнктов нет резольвируемых, или же все резольвенты уже присутствуют в текущем множестве дизъюнктов. Это означает, что теорема опровергнута, то есть формула G не выводима из множества формул S.
2. В результате очередного применения правила резолюции получен пустой дизъюнкт. Это означает, что теорема доказана, то есть $S \vdash G$.
3. Процесс не заканчивается, то есть множество дизъюнктов пополняется всё новыми резольвентами, среди которых нет пустых. Это ничего не означает.

ЗАМЕЧАНИЕ

Таким образом, исчисление предикатов является *полуразрешимой* теорией, а метод резолюций является *частичным* алгоритмом автоматического доказательства теорем.

Пример. Проверим выводимость (п. 4.4.5): $\exists x\ (\forall y\ (A(x,y))) \vdash_{\mathcal{K}} \forall y\ (\exists x\ (A(x,y)))$. Сначала необходимо исходную формулу $\exists x\ (\forall y\ (A(x,y)))$ и отрицание целевой формулы $\neg\,(\forall y\ (\exists x\ (A(x,y))))$ преобразовать в дизъюнкты.

1. Формулы без изменений.
2. $\exists x\ (\forall y\ (A(x,y)))$, $\exists y\ (\forall x\ (\neg A(x,y))$.

3–4. Формулы без изменений.

5. $\forall y\ (A(a,y))$, $\forall x\ (\neg A(x,b)$.

6. $A(a,y)$, $\neg A(x,b)$.

7–8. Формулы без изменений.

После этого проводится резольвирование имеющихся дизъюнктов 1–2.

1. $A(a,y)$.
2. $\neg A(x,b)$.
3. $\square \qquad (1,2), \quad \sigma = [a/x, b/y]$.

Таким образом, теорема доказана.

Рассмотрим теперь обратную теорему $\forall y\ (\exists x\ (A(x,y))) \vdash_{\mathcal{K}} \exists x\ (\forall y\ (A(x,y)))$. Сначала необходимо исходную формулу $\forall y\ (\exists x\ (A(x,y)))$ и отрицание целевой формулы $\neg\,(\exists x\ (\forall y\ (A(x,y))))$ преобразовать в дизъюнкты.

1. Формулы без изменений.
2. $\forall y\ (\exists x\ (A(x,y)), \forall x\ (\exists y\ (\neg A(x,y)))$.

3–4. Формулы без изменений.

5. $\forall y\ (A(f(y),y)), \forall x\ (\neg A(x,g(x)))$.
6. $A(f(y),y)$, $\neg A(x,g(x))$.

7–8. Формулы без изменений.

Попытка резольвирования имеющихся дизъюнктов 1–2:

1. $A(f(y),y)$.
2. $\neg A(x,g(x))$.

Резольвенты нет, так как литералы $A(f(y),y)$, $A(x,g(x))$ не унифицируемы. Действительно, подстановка $[f(y)/x]$ даст $A(f(y),y)$, $A(f(y),g(f(y)))$, а подстановка $[g(f(y))/y]$ не может входить в унификатор (п. 4.3.2). Таким образом, теорема опровергнута.

4.6.7. Дерево вывода

При применении метода резолюций (и не только) часто используют графическое представление вывода, которое называется *деревом вывода*

В узлах дерева помещаются дизъюнкты. Дерево вывода растет снизу вверх. В верхних узлах (листьях) расположены исходные дизъюнкты, а в промежуточных узлах — резольвенты. В корне дерева — пустое предложение. На дугах можно показать резольвируемые литералы и используемые подстановки.

Пример. Докажем методом резолюций теорему $A \to B, B \to C \vdash_{\mathcal{L}} A \to C$.

Сначала нужно преобразовать в дизъюнкты исходные формулы $A \to B$, $B \to C$ и отрицание целевой формулы $\neg(A \to C)$.

1. $\neg A \vee B$, $\neg B \vee C$, $\neg(\neg A \vee C)$.
2. $\neg A \vee B$, $\neg B \vee C$, $A \,\&\, \neg C$.

3–7. Формулы без изменений.

8. $\neg A \vee B$, $\neg B \vee C$, A, $\neg C$.

1. $\neg A \vee B$
2. $\neg B \vee C$
3. A
4. $\neg C$
5. $\neg A \vee C$ $(1, 2)$
6. C $(3, 5)$
7. $\square$ $(4, 6)$

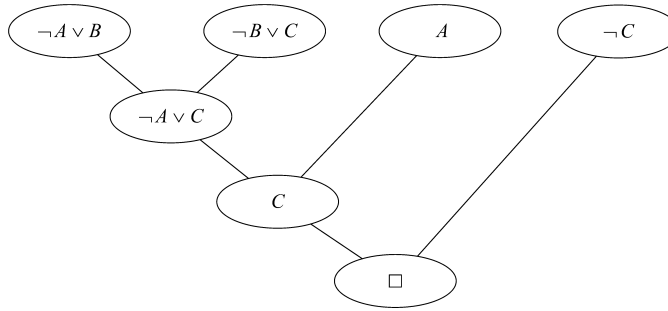


Рис. 4.2. Дерево вывода 1

После этого проводится резольвирование имеющихся дизъюнктов 1–4. Дерево вывода приведено на рис. 4.2.

Таким образом, теорема доказана.

Рассмотрим еще один пример. Докажем методом резолюций теорему $\vdash_{\mathcal{L}} (((A \to B) \to A) \to A)$. Сначала нужно преобразовать в дизъюнкты отрицание целевой формулы $\neg(((A \to B) \to A) \to A)$.

1. $\neg(\neg(\neg(\neg A \vee B) \vee A) \vee A)$.
2. $(((A \,\&\, \neg B) \vee A) \,\&\, \neg A)$.

3–6. Формула без изменений.

7. $(A \vee A) \,\&\, (\neg B \vee A) \,\&\, \neg A$.
8. $A \vee A, \neg B \vee A, \neg A$.

После этого проводится резольвирование имеющихся дизъюнктов 1–3. Дерево вывода приведено на рис. 4.3.

1. $A \vee A$
2. $\neg B \vee A$
3. $\neg A$
4. A $(1, 3)$
5. $\square$ $(3, 4)$

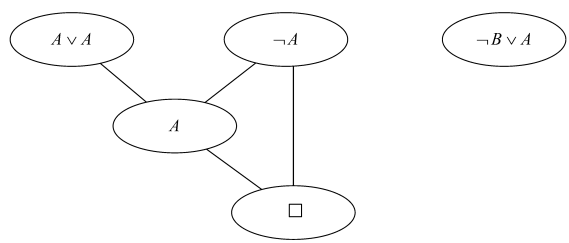


Рис. 4.3. Дерево вывода 2

ЗАМЕЧАНИЕ

Из этого примера видно, что устоявшийся термин «дерево вывода» не совсем корректен. Если дизъюнкт участвует в резольвировании несколько раз, то полученный граф может не быть деревом. Если же некоторые исходные дизъюнкты не участвуют в выводе, то граф может быть даже несвязным.

Алгоритм 4.3. Метод резолюций

Вход: множество дизъюнктов C, полученных из множества формул S и формулы $\neg G$.
Выход: **true** — если G выводимо из S, **false** — в противном случае.

```
M := C  // M — текущее множество дизъюнктов
while □ ∉ M do
  Choose(M, c1, c2, p1, p2, σ)  // выбор родительских дизъюнктов
  if c1, c2 = ∅ then
    return false  // нечего резольвировать — теорема опровергнута
  end if
  c := R(c1, c2, p1, p2, σ)  // вычисление резольвенты
  M := M + c  // пополнение текущего множества
end while
return true  // теорема доказана
```

4.6.8. Алгоритм метода резолюций

Алгоритм поиска опровержения методом резолюций проверяет выводимость формулы G из множества формул S.

Обоснование. Если алгоритм заканчивает свою работу, то правильность результата очевидным образом следует из теорем предшествующих параграфов. Вообще говоря, завершаемость этого алгоритма не гарантирована. □

ОТСТУПЛЕНИЕ

При автоматическом доказательстве теорем методом резолюций львиная доля вычислений приходится на поиск унифицируемых дизъюнктов. Таким образом, эффективная реализация алгоритма унификации критически важна для практической применимости метода резолюций.

В алгоритме поиска опровержения методом резолюций использованы две вспомогательные функции. Функция R вычисляет резольвенту двух дизъюнктов, c_1 и c_2, содержащих унифицируемые контрарные литералы p_1 и p_2 соответственно; при этом σ — наиболее общий унификатор. Результатом работы функции является резольвента.

Процедура Choose выбирает в текущем множестве дизъюнктов M два резольвируемых дизъюнкта, то есть два дизъюнкта, которые содержат унифицируемые контрарные литералы. Если таковые есть, то процедура их возвращает, в противном случае возвращается пустое множество. Конкретные реализации процедуры Choose называются *стратегиями* метода резолюций.

ЗАМЕЧАНИЕ

В настоящее время предложено множество различных стратегий метода резолюций. Среди них различаются *полные* и *неполные* стратегии. Полные стратегии — это такие, которые гарантируют нахождение доказательства теоремы, если оно вообще существует. Неполные стратегии могут в некоторых случаях не находить доказательства, зато они работают быстрее. Следует иметь в виду, что автоматическое доказательство теорем методом резолюций имеет, по существу, переборный характер, и этот перебор столь велик, что может быть практически неосуществим за приемлемое время.

Глава 5 Комбинаторный анализ

Предмету *комбинаторного анализа*, или, короче, *комбинаторики* не так просто дать краткое исчерпывающее определение. В некотором смысле слово «комбинаторика» можно понимать как синоним термина «дискретная математика», то есть исследование дискретных конечных математических структур. На школьном уровне с термином «комбинаторика» связывают просто набор известных формул, служащих для вычисления так называемых *комбинаторных чисел*, о которых идёт речь в первых разделах этой главы. Может показаться, что эти формулы полезны только для решения олимпиадных задач и не имеют отношения к практическому программированию. На самом деле это далеко не так. Вычисления на дискретных конечных математических структурах, которые часто называют *комбинаторными вычислениями*, требуют комбинаторного анализа для установления свойств и выявления оценки применимости используемых алгоритмов. Рассмотрим элементарный жизненный пример.

Пример. Предположим, что забыт пароль на каком-то сайте. Так как в базе данных он хранится в виде кодированного образа, то единственный вариант восстановления — полный перебор всех возможных вариантов. В наше время большинство сайтов требуют создать пароль из восьми символов. Если известно, что пароль состоит только из цифр и строчных символов латиницы, то необходимо перебрать $(10 + 26)^8 \approx 2,82 \cdot 10^{12}$ вариантов. Современные программы для подбора забытого пароля осуществляют порядка 200 проверок паролей в секунду. Таким образом, на подбор такого простого варианта пароля в худшем случае уйдёт около $1,41 \cdot 10^{10}$ секунд, что составляет около 160 000 дней, или же примерно 447 лет. Если же в пароле использованы прописные буквы и специальные знаки (так рекомендуется делать), то подбор пароля «в лоб» практически неосуществим.

Этот пример показывает, что практические задачи и алгоритмы требуют предварительного комбинаторного анализа применимости и количественной оценки. Задачи обычно оцениваются с точки зрения *размера*, то есть общего количества вариантов, среди которых нужно найти решение, а алгоритмы оцениваются с точки зрения *сложности*. При этом различают *сложность по времени* (или *временну́ю сложность*), то есть количество необходимых шагов алгоритма, и *сложность по памяти* (или *емкостну́ю сложность*), то есть объём памяти, необходимый для работы алгоритма.

Худшим случаем называется вариант входных данных, в котором наблюдаются максимальные затраты вычислительного ресурса (времени или памяти). *Лучшим случаем* называется вариант входных данных, в котором наблюдаются минимальные затраты вычислительного ресурса. Таким образом появляются понятия сложности в лучшем случае (оценка снизу) и сложности в худшем случае (оценка сверху).

Во всех случаях основным инструментом такого анализа оказываются формулы и методы, рассматриваемые в этой главе.

5.1. Комбинаторные задачи

Во многих практических случаях возникает необходимость подсчитать количество возможных комбинаций объектов, удовлетворяющих определённым условиям. Такие задачи называются *комбинаторными*. Разнообразие комбинаторных задач не поддаётся исчерпывающему описанию, но среди них есть целый ряд особенно часто встречающихся, для которых известны способы подсчёта.

5.1.1. Комбинаторные конфигурации

Для формулировки и решения комбинаторных задач используются различные модели *комбинаторных конфигураций*. Рассмотрим следующие две наиболее популярные:

1. Дано n мячей. Их нужно разместить по m ящикам так, чтобы выполнялись заданные ограничения. Сколькими способами это можно сделать?
2. Рассмотрим множество функций $f\colon 1..n \to 1..m$. Сколько существует функций f, удовлетворяющих заданным ограничениям?

ЗАМЕЧАНИЕ

Большей частью соответствие конфигураций, описанных на «языке мячей и ящиков» и на «языке функций», очевидно, поэтому доказательство правильности способа подсчёта (вывод формулы) можно провести на любом языке. Если сведéние одной модели к другой не очевидно, то оно включается в доказательство.

Таким образом, при решении комбинаторных задач требуется вычислять мощность некоторых множеств (множества функций, множества размещений мячей по ящикам). При этом часто оказываются полезными следующие три наблюдения из теории множеств, которым даны специальные названия.

Неформально *правило суммы* (или *правило сложения*) формулируется следующим образом: если возможности построения комбинаторной конфигурации взаимно исключают друг друга, то их количества следует складывать. На языке теории множеств правило суммы — это следствие теоремы п. 1.2.7.

Пример. Сколько возможных ходов имеет ферзь, стоящий на одном из центральный полей пустой шахматной доски? Решение: ферзь может пойти либо по горизонтали (7 ходов), либо по вертикали (7 ходов), либо по главной диагонали (7 ходов), либо по ортогональной диагонали (6 ходов). Ответ: $7 + 7 + 7 + 6 = 27$.

Неформально *правило произведения* (или *правило умножения*) формулируется следующим образом: если возможности построения комбинаторной конфигурации независимы, то их количества следует умножать. На языке теории множеств правило произведения — это следствие 1 теоремы п. 1.4.2.

Пример. Сколькими способами можно на шахматной доске поставить две ладьи так, чтобы они не били друг друга? Решение: первую ладью можно поставить на любое из 64 полей, при этом она будет бить 14 полей. Вторую ладью можно поставить на любое из оставшихся $64 - 15 = 49$ полей. Ответ: $64 \cdot 49 = 3136$ способов.

Правила суммы и произведения могут применяться совместно.

Пример. Сколькими способами можно на шахматной доске поставить два короля так, чтобы они не били друг друга? Решение: король может стоять либо в углу доски (4 занятых поля), либо на краю доски (6 занятых полей), либо не на краю (9 занятых полей). Ответ: $4 \cdot (64-4) + 24 \cdot (64-6) + 36 \cdot (64-9) = 240 + 1392 + 1980 = 3612$ способов.

Неформально *принцип Дирихле* часто излагают в следующей форме: если $n \neq m$, то n кроликов невозможно рассадить в m клеток так, чтобы в каждой клетке было по одному кролику. На языке теории множеств принцип Дирихле — это следствие теоремы 2 п. 1.2.5, о том, что различные отрезки натурального ряда неравномощны.

Примеры

1. Пусть в равностороннем треугольнике со стороной 1 расположены 5 точек. Тогда существует по крайней мере две точки, расстояние между которыми не больше $1/2$. Действительно, средние линии треугольника разбивают его на четыре равносторонних треугольника со стороной $1/2$. По принципу Дирихле две точки попадут в один из этих треугольников. Расстояние между ними будет не больше $1/2$.
2. Пусть грани куба окрашены в два цвета. Тогда найдутся соседние одноцветные грани. Действительно, рассмотрим три грани куба, имеющие общую вершину. Назовём их «кроликами», а данные цвета — «клетками». По принципу Дирихле, найдутся две соседние грани, окрашенные в один цвет.

5.1.2. Размещения

Число всех функций $f\colon 1..n \to 1..m$ (при отсутствии ограничений), или число всех возможных способов разместить n мячей по m ящикам, называется *числом размещений* и обозначается $U(m,n)$.

ТЕОРЕМА. $U(m,n) = m^n$.

Доказательство. Поскольку ограничений нет, мячи размещаются независимо друг от друга, то есть каждый из n мячей можно разместить m способами. Таким образом, по правилу произведения получаем m^n возможных размещений. □

Примеры

1. Число всевозможных бинарных последовательностей длины n равно $U(2,n) = 2^n$. Действительно, на каждую из n позиций в последовательности приходится два варианта (1 или 0), значения в разных позициях независимы.
2. У кодового замка пароль состоит из трёх позиций, на каждой из них может стоять любая из цифр от 0 до 9. Сколько возможных паролей на данном замке? Решение: $U(10,3) = 10^3 = 1000$.

5.1.3. Размещения без повторений

Число инъективных функций $f\colon 1..n \to 1..m$, или число способов разместить n мячей по m ящикам, не более чем по одному в ящик, называется *числом размещений без повторений* и обозначается $A(m,n)$, или $[m]_n$, или $(m)_n$.

ТЕОРЕМА. $A(m,n) = \dfrac{m!}{(m-n)!}$.

Доказательство. Ящик для первого мяча можно выбрать m способами, для второго — $m-1$ способами и т. д. Имеем $A(m,n) = m\cdot(m-1)\cdot\ldots\cdot(m-n+1) = \dfrac{m!}{(m-n)!}$. □

По определению считают, что $A(m,n) \stackrel{\text{Def}}{=} 0$ при $n > m$ и $A(m,0) \stackrel{\text{Def}}{=} 1$.

Пример. В некоторых видах спортивных соревнований исходом является определение участников, занявших первое, второе и третье места. Сколько возможно различных исходов, если в соревновании участвуют n участников? Каждый возможный исход соответствует функции $F\colon \{1,2,3\} \to \{1..n\}$ (аргумент — номер призового места, результат — номер участника). Таким образом, всего возможно $A(n,3) = n(n-1)(n-2)$ различных исходов.

ОТСТУПЛЕНИЕ

Простые формулы, выведенные для числа размещений без повторений, дают повод поговорить об элементарных, но весьма важных вещах. Рассмотрим две формулы:

$$A(m,n) = m\cdot(m-1)\cdot\ldots\cdot(m-n+1) \quad \text{и} \quad A(m,n) = \frac{m!}{(m-n)!}.$$

Формула слева выглядит сложной и незавершённой, формула справа — изящной и «математичной». Но формула — это частный случай алгоритма. При практическом вычислении левая формула оказывается намного предпочтительнее правой. Во-первых, для вычисления по левой формуле потребуется n умножений, а по правой — $2m-n-2$ умножений и одно деление. Поскольку $n < m$, левая формула вычисляется существенно быстрее. Во-вторых, число $A(m,n)$ растёт довольно быстро и при больших m и n может не поместиться в разрядную сетку. Левая формула работает правильно, если результат помещается в разрядную сетку. При вычислении же по правой формуле переполнение может наступить «раньше времени» (то есть промежуточные результаты не помещаются в разрядную сетку, в то время как окончательный результат мог бы поместиться), поскольку факториал — очень быстро растущая функция.

5.1.4. Перестановки

Если $|X| = |Y| = n$, то существуют взаимно-однозначные функции $f\colon X \to Y$.

Число взаимно-однозначных функций $f\colon 1..n \to 1..n$, или *число перестановок* n предметов, обозначается $P(n)$.

ТЕОРЕМА. $P(n) = n!$.

Доказательство. $P(n) = A(n,n) = n\cdot(n-1)\cdot\ldots\cdot(n-n+1) = n\cdot(n-1)\cdot\ldots\cdot 1 = n!$. □

Пример. Последовательность $\mathcal{E} = (E_1,\ldots,E_m)$ непустых подмножеств множества E ($\mathcal{E} \subset 2^E$, $E_i \subset E$, $E_i \neq \varnothing$) называется *цепочкой* в E, если

$$\forall\, i \in 1..(m-1)\ \ (E_i \subset E_{i+1}\ \&\ E_i \neq E_{i+1})\,.$$

Цепочка $\mathcal{E}$ называется *полной* цепочкой в E, если $|\mathcal{E}| = |E|$. Сколько существует полных цепочек? Очевидно, что в полной цепочке каждое следующее подмножество E_{i+1} получено из предыдущего подмножества E_i добавлением ровно одного

элемента из E и, таким образом, $|E_1| = 1, |E_2| = 2, \ldots, |E_m| = m$. Следовательно, полная цепочка определяется порядком, в котором элементы E добавляются для образования очередного элемента полной цепочки. Отсюда количество полных цепочек — это количество перестановок элементов множества E, равное $m!$.

ЗАМЕЧАНИЕ

В примере рассматриваются конечные цепочки. Понятие полной цепочки применимо и для бесконечных множеств.

5.1.5. Сочетания

Число строго монотонно возрастающих функций $f\colon 1..n \to 1..m$, или число размещений n неразличимых мячей по m ящикам не более чем по одному в ящик, то есть число способов выбрать из m ящиков n ящиков с мячами, называется *числом сочетаний* и обозначается $C(m,n)$, или C_m^n, или $\binom{n}{m}$.

По определению $C(m,n) \stackrel{\text{Def}}{=} 0$ при $n > m$.

ТЕОРЕМА. $C(m,n) = \dfrac{m!}{n!(m-n)!}$.

Доказательство.

[Обоснование формулы] Сочетание является размещением без повторений неразличимых мячей. Следовательно, число сочетаний определяется тем, какие ящики заняты мячами, поскольку перестановка мячей при тех же занятых ящиках считается одним сочетанием. Таким образом,

$$C(m,n) = \frac{A(m,n)}{P(n)} = \frac{m!}{n!(m-n)!}.$$

[Сведéние моделей] Число сочетаний является количеством строго монотонно возрастающих функций, потому что любая строго монотонно возрастающая функция $f\colon 1..n \to 1..m$ определяется набором своих значений, причём $1 \leqslant f(1) < \ldots < f(n) \leqslant m$. Другими словами, каждая строго монотонно возрастающая функция определяется выбором n чисел из диапазона $1..m$. Таким образом, число строго монотонно возрастающих функций равно числу n-элементных подмножеств m-элементного множества, которое, в свою очередь, равно числу способов выбрать n ящиков с мячами из m ящиков. □

Пример. В начале игры в домино каждому играющему выдаётся 7 костей из имеющихся 28 различных костей. Сколько существует различных комбинаций костей, которые игрок может получить в начале игры? Очевидно, что искомое число равно числу 7-элементных подмножеств 28-элементного множества. Имеем

$$C(28,7) = \frac{28!}{7!(28-7)!} = \frac{28 \cdot 27 \cdot 26 \cdot 25 \cdot 24 \cdot 23 \cdot 22}{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 1\,184\,040.$$

Пример. Сколько существует способов записать натуральное число n в виде суммы m натуральных чисел: $a_1 + a_2 + \ldots + a_m = n$? Любое натуральное число можно представить в виде суммы единиц. Тогда решение задачи сводится к разделению n выстроенных в ряд единиц на m непустых групп с помощью $m-1$ перегородки (см. рис. 5.1). Для перегородки существует $n-1$ позиция между единицами. Таким образом, число способов расставить перегородки: $C(n-1, m-1)$

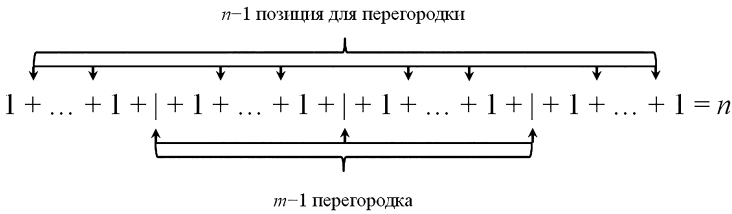


Рис. 5.1. Разделение единиц перегородками

5.1.6. Сочетания с повторениями

Число монотонно возрастающих функций $f\colon 1..n \to 1..m$, или число размещений n неразличимых мячей по m ящикам, называется *числом сочетаний с повторениями* и обозначается $V(m, n)$.

ТЕОРЕМА 1. $V(m, n) = C(n + m - 1, n)$.

Доказательство. Монотонно возрастающей функции $f\colon 1..n \to 1..m$ однозначно соответствует строго монотонно возрастающая функция $g\colon 1..n \to 1..(n + m - 1)$. Это соответствие устанавливается следующей формулой: $g(k) = f(k) + k - 1$. □

ЗАМЕЧАНИЕ

$V(0, n) = C(n-1, n) = 0$ при $n > 0$ по определению для (m, n). Однако обычно считают, что $V(0, 0) \stackrel{\text{Def}}{=} 1$.

Пример. Сколькими способами можно рассадить n вновь прибывших гостей среди m гостей, уже сидящих за круглым столом? Очевидно, что между m сидящими за круглым столом гостями имеется m промежутков, в которые можно рассаживать вновь прибывших. Таким образом, число способов равно

$$V(m, n) = C(m + n - 1, n) = \frac{(m + n - 1)!}{n!(m - 1)!}.$$

ТЕОРЕМА 2. $V(m, n) = V(n + 1, m - 1)$.

Доказательство. $V(m, n) = C(m + n - 1, n) = \dfrac{(m + n - 1)!}{n!(m - 1)!} = \dfrac{(n + m - 1)!}{(m - 1)!n!} =$
$= C(n + m - 1, m - 1) = V(n + 1, m - 1)$. □

Пример. Сколько существует способов записать натуральное число n в виде суммы m неотрицательных целых чисел: $a_1+a_2+\ldots+a_m=n$? Сведем задачу к примеру из п. 5.1.5 с натуральными числами. Пусть $y_i=a_i+1$. Тогда $y_1+y_2+\ldots+y_m=n+m$. Ответ: $C(m+n-1,m-1)=V(m,n)$.

5.1.7. Дискретная вероятность

Пусть задано конечное множество $X=\{x_1,\ldots,x_n\}$, $|X|=n$, и тотальная функция $\mathrm{P}\colon X\to[0;1]$, такая, что $\sum\limits_{i=1}^{n}\mathrm{P}(x_i)=1$. Тогда число $p_i:=\mathrm{P}(x_i)$ называется *вероятностью* элемента x_i, а набор чисел $\mathrm{P}_X=(p_1,\ldots,p_n)$ называется *распределением вероятности* на множестве X. Распределение вероятности P_X, в котором $\forall\, i\in 1..n\ \left(p_i=\frac{1}{n}\right)$, называется *равномерным*.

Элементы множества X могут иметь произвольную природу: это могут быть результаты измерения числового значения некоторой физической величины, способы разложить мячи по ящикам в комбинаторной конфигурации, альтернативные события реального мира, которые могут произойти в будущем, и т. д.

Пример. Одновременно бросаются два игральных кубика. Какова вероятность того, что на двух кубиках выпадут шестёрки? Всего возможно 36 различных комбинаций, распределение естественно считать равномерным (кубики «честные»), значит, вероятность двух шестёрок равна $\frac{1}{36}$.

ЗАМЕЧАНИЕ

Вероятность — весьма общее понятие, которое изучается в специальном разделе математики — *теории вероятности*. Теория вероятности тесно связана с другими разделами математики: математической статистикой, теорией игр, теорией меры и др. Также теория вероятности нашла множество приложений в физике, экономике и биологии. В этом учебнике используется упрощённое понятие вероятности, достаточное для проведения комбинаторных вычислений.

Рассмотрим множество $A\subseteq X$. Если на X задано распределение вероятности, то *вероятностью подмножества* A называется число, равное $\sum\limits_{a\in A}\mathrm{P}(a)$.

Из определения ясно, что $\mathrm{P}(\varnothing)=0,\quad \mathrm{P}(X)=1,\quad \forall\, A\ \ (0\leqslant\mathrm{P}(A)\leqslant 1)$.

Пример. В игральной колоде 52 карты четырёх мастей, в каждой из которых по 13 карт разного достоинства. Вероятность быть тузом равна $\frac{1}{52}+\frac{1}{52}+\frac{1}{52}+\frac{1}{52}=\frac{4}{52}=\frac{1}{13}$.

Стоит заметить, что в случае равномерного распределения вычисление вероятности подмножества A упрощается: $\mathrm{P}(A)=\dfrac{|A|}{|X|}$.

Пример. В игре «покер» комбинация из пяти карт одинаковой масти называется «флеш». Какова вероятность комбинации «флеш»? Множество X содержит в качестве элементов всевозможные комбинации из пяти карт, порядок которых значения не имеет. Количество таких элементов равно $C(52,5)$. В одной масти 13 карт, из которых в комбинацию входят пять любых. Мастей всего 4, значит, количество элементов в множестве A равно $4\cdot C(13,5)$. Таким образом, вероятность комбинации «флеш» равна $\frac{4\cdot C(13,5)}{C(52,5)}=\frac{33}{16660}\approx 0,002$.

Случайная величина — это тотальная функция $\varphi\colon X \to \mathbb{R}$, причём на множестве X заданы вероятности элементов.

Математическим ожиданием случайной величины при распределении вероятности $\mathrm{P}_X = (p_1, \ldots, p_n)$ называется число

$$\mathrm{E}_\varphi = \sum_{k=1}^{n} \varphi(x_k)p_k = \varphi(x_1)p_1 + \ldots + \varphi(x_n)p_n.$$

Легко видеть, что среднее арифметическое — частный случай математического ожидания для равномерного распределения.

Математическое ожидание можно назвать средним значением X.

ЗАМЕЧАНИЕ

Если элементы множества X являются числами, то в качестве функции φ может рассматриваться тождественная функция. Формула математического ожидания примет вид $\sum_{k=1}^{n} x_k p_k$.

Пример. Рассмотрим бросок игрального кубика: $X = \{1, 2, 3, 4, 5, 6\}$. Математическое ожидание числа очков на верхней грани

$$E = \frac{1}{6}\sum_{k=1}^{6} x_k = \frac{1}{6}(1 + 2 + 3 + 4 + 5 + 6) = 3,5.$$

В среднем при одном подбрасывании кубика выпадает 3,5 очка.

ЗАМЕЧАНИЕ

Математическое ожидание — одно из важнейших понятий в теории вероятности. Имеет важное прикладное значение при оценке рисков, при торговле на финансовых рынках и во многих других областях.

Пример. В казино предлагают сыграть в игру: игрок ставит некоторую сумму на число от 1 до 6, а потом бросает три игральных кубика. Если загаданное число не выпадет ни разу, то его ставка сгорает. Если выпадет — ставку возвращают, плюс казино платит поставленную сумму столько раз, сколько раз выпало число. Игра кажется заманчивой, однако рассмотрим средний выигрыш при ставке в 1 доллар. Игрок может потерять доллар или выиграть 1, 2 или 3 доллара. Тогда множество $X = \{-1, 1, 2, 3\}$. Рассчитаем математическое ожидание:

$$(-1)\cdot\frac{5^3}{216} + 1\cdot\frac{C(3,1)\cdot 5^2}{216} + 2\cdot\frac{C(3,2)\cdot 5}{216} + 3\cdot\frac{1}{216} = -\frac{7}{216} \approx -0,03$$

В среднем при каждой ставке в один доллар игрок потеряет три цента.

5.2. Перестановки

Для вычисления количества перестановок в п. 5.1.4 установлена очень простая формула: $P(n) = n!$. Применяя эту формулу при решении практических задач, не следует забывать, что факториал — это *очень* быстро растущая функция, в частности, факториал растёт быстрее экспоненты. График факториала показан на рис. 5.2.

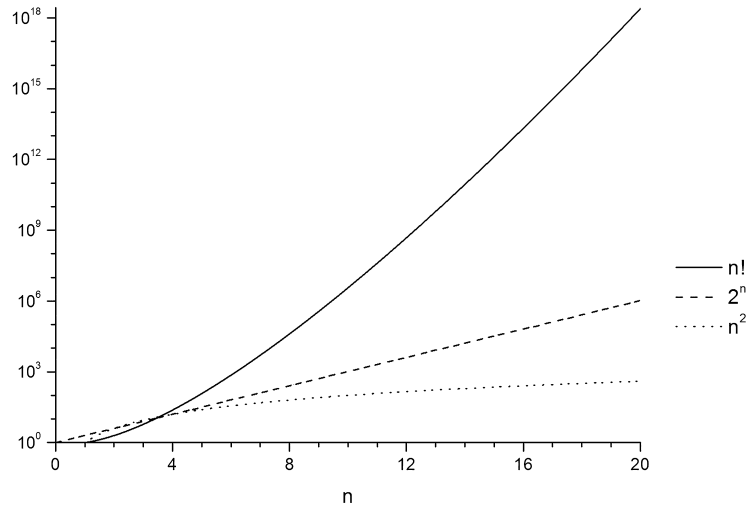


Рис. 5.2. Графики факториала, экспоненты и квадрата

5.2.1. Циклы в перестановках

В п. 2.2.7 рассматриваются взаимно-однозначные функции (перестановки), которые удобно задавать *таблицами подстановки*. В таблице подстановки нижняя строка (значения функции) является перестановкой элементов верхней строки (значения аргумента). Если принять соглашение, что элементы верхней строки (аргументы) всегда располагаются в определённом порядке (например, по возрастанию), то верхнюю строку можно не указывать — подстановка определяется одной нижней строкой. Таким образом, подстановки (таблицы) взаимно-однозначно соответствуют перестановкам (функциям). Напомним, что множество перестановок образует группу относительно суперпозиции.

Перестановку f (и соответствующую ей подстановку) элементов $1, \ldots, n$ будем обозначать $(a_1, \ldots, a_n)$, где $a_i = f(i)$ и $a_1, \ldots, a_n$ — *все* различные числа из диапазона $1..n$. Если $f(i) = i$, то i называется *неподвижной точкой* перестановки, если же $f(i) \neq i$, то i называется *подвижной точкой* перестановки.

Если задана перестановка f, то *циклом* называется *последовательность* элементов $x_1, \ldots, x_k$, такая, что $f(x_i) = x_{i+1}$ при $1 \leqslant i < k$ и $f(x_k) = x_1$. Количество элементов в цикле называется *длиной* цикла. Цикл длины 1 называется *тривиальным* циклом, или *петлёй*. Цикл длины 2 называется *транспозицией*. Иногда перестановки удобно представлять в графической форме, проводя стрелки от каждого элемента x к элементу $f(x)$.

ЗАМЕЧАНИЕ

Из графического представления перестановки наглядно видно происхождение термина «цикл».

Пример. Графическое представление перестановки (п. 5.1.4.)

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 4 & 5 \end{vmatrix}$$

показано на рис. 5.3.

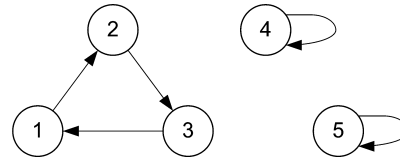


Рис. 5.3. Графическое представление перестановки

Перестановка может содержать несколько нетривиальных циклов. Перестановка длины n, которая содержит ровно один нетривиальный цикл длины k, называется *циклической* (перестановкой длины k), $k \leqslant n$. В циклической перестановке все элементы цикла — подвижные точки, а остальные элементы — неподвижные точки.

Пример. В перестановке на рис. 5.3 точки $1, 2, 3$ — подвижные точки, и образуют цикл $(1, 2, 3)$, а точки 4 и 5 — неподвижные точки. Эта перестановка циклическая.

Множество элементов, входящих в цикл, называется *орбитой*. Орбита тривиального цикла состоит из одного элемента.

ЛЕММА. *Орбиты различных циклов в одной перестановке не пересекаются.*

Доказательство. От противного. Допустим, существует элемент x, который входит в два цикла. Тогда «следующий» элемент $f(x)$ также входит в оба цикла и т. д. Ввиду конечности циклов заключаем, что циклы совпадают. □

СЛЕДСТВИЕ. *Множество орбит образует разбиение перестановки, а отношение «принадлежать одной орбите» является эквивалентностью.*

Пример. В перестановке на рис. 5.3 $\{1, 2, 3\}$ — орбита.

ТЕОРЕМА. *Всякая неединичная перестановка является суперпозицией циклических перестановок.*

Доказательство. Построим требуемое разложение следующим образом. Возьмём произвольный элемент и выделим орбиту, которой он принадлежит. Построим циклическую перестановку с циклом, соответствующим выделенной орбите. Затем возьмём любой элемент из оставшихся (не вошедших в найденные орбиты) и повторим выделение орбиты, соответствующей элементу. Ввиду конечности перестановок процесс закончится. □

ЗАМЕЧАНИЕ

Если орбита состоит из одного элемента, то цикл тривиален, а циклическая подстановка является единичной. Единичную подстановку можно не включать в композицию.

Пример. Перестановка

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 3 & 1 & 7 & 5 & 6 & 8 & 9 & 4 \end{vmatrix}$$

разлагается в два цикла: $f = (1,2,3) \circ (4,7,8,9)$. Неподвижные элементы в разложении не участвуют.

5.2.2. Порядок перестановки

Перестановка — это взаимно-однозначная функция. Что будет, если переставить элементы так, как задаёт функция, а потом переставить ещё раз и ещё раз? Другими словами, как ведёт себя степень перестановки?

ТЕОРЕМА 1. $(f^k)^{-1} = (f^{-1})^k$.

Доказательство. По индукции. База — по определению. Пусть $(f^k)^{-1} = (f^{-1})^k$. Тогда $(f^{k+1})^{-1} = (ff^k)^{-1} = (f^k)^{-1}(f)^{-1} = (f^{-1})^k f^{-1} = f^{-1}(f^{-1})^k = (f^{-1})^{k+1}$. Первый переход по определению степени (п. 1.6.5), второй — по свойству групп (п. 2.2.6), третий — по индукционному предположению, четвертый — по лемме из п. 1.6.5, а пятый, последний, снова по определению степени. □

ЛЕММА 1. *Для любой перестановки f существует такое неотрицательное целое число $k \geqslant 0$, что $f^k = e$.*

Доказательство. Поскольку различных перестановок лишь конечное число, а именно $n!$, то в бесконечной последовательности $f, f^2, f^3, f^4, \ldots$ есть совпадающие элементы. Пусть для некоторых натуральных чисел p, q $(p < q)$ выполняется равенство $f^p = f^q$. Тогда $(f^p)^{-1} = f^{-p}$, откуда $e = ((f^p)^{-1} \circ f^p = f^{-p} \circ f^q = f^{q-p}$. □

СЛЕДСТВИЕ. *Симметрическая группа S_n является периодической.*

Наименьшее число k такое, что $f^k = e$, называется порядком перестановки.

ЗАМЕЧАНИЕ

Порядок перестановки — это индивидуальная характеристика отдельной перестановки. В то же время порядок группы перестановок S_n — это общая характеристика всего множества перестановок. Эти два порядка не следует путать!

Пример. Порядок перестановки f равен 3.

$$f = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 4 & 5 \end{vmatrix}, \quad f^2 = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 2 & 4 & 5 \end{vmatrix}, \quad f^3 = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{vmatrix}.$$

ЛЕММА 2. *Порядок циклической перестановки равен длине цикла.*

Доказательство. Пусть цикл содержит k элементов. Все неподвижные точки останутся неподвижными при любом количестве применений перестановки, а подвижные точки после k применений вернутся на своё место. □

ТЕОРЕМА 2. *Порядок перестановки равен наименьшему общему кратному длин всех циклов.*

Доказательство. Пусть перестановка f разлагается в циклические перестановки по теореме п. 5.2.1: $f = f_1 \circ f_2 \circ \cdots \circ f_s$. Если перестановки $f_1, \ldots, f_s$ содержат циклы длины $k_1, \ldots, k_s$, то есть имеют порядки $k_1, \ldots, k_s$, то наименьшее число m, для которого одновременно выполняются все равенства $f_1^m = e$, $f_2^m = e$, ..., $f_s^m = e$, равняется наименьшему общему кратному чисел $k_1, k_2, \ldots, k_s$. □

5.2.3. Инверсии

Если в перестановке $f = (a_1, \ldots, a_n)$ для элементов a_i и a_j имеет место неравенство $a_i > a_j$ при $i < j$, то пара (a_i, a_j) называется *инверсией*. Обозначим $I(f)$ — *число инверсий* в перестановке f.

ТЕОРЕМА. *Произвольную перестановку f можно представить в виде суперпозиции $I(f)$ транспозиций соседних элементов.*

Доказательство. Пусть $f = (a_1, \ldots, 1, \ldots, a_n)$. Переставим 1 на первое место, меняя её местами с соседними слева элементами. Обозначим последовательность этих транспозиций через t_1. При этом все инверсии, в которых участвовало число 1, пропадут. Затем переставим 2 на второе место и т. д. Таким образом, $f \circ t_1 \circ \ldots \circ t_n = e$ и по свойству группы $f = t_n^{-1} \circ \ldots \circ t_1^{-1}$, причём $|t_1| + |t_2| + \ldots + |t_n| = I(f)$. □

СЛЕДСТВИЕ. *Всякая сортировка может быть выполнена перестановкой соседних элементов.*

ОТСТУПЛЕНИЕ

Доказанная теорема утверждает, что произвольную перестановку можно представить в виде композиции определённого числа транспозиций, но не утверждает, что такое представление является эффективным.

Рассмотрим входную последовательность, состоящую из n чисел: $(a_1, a_2, \ldots, a_n)$. *Задача сортировки* поиск такой перестановки $(b_1, b_2, \ldots, b_n)$ входной последовательности, что $\forall\, i \in 1..(n-1)\ \ (b_i \leqslant b_{i+1})$.

Метод *сортировки*, основанный на предшествующей теореме, известен как *метод пузырька*. Заметим, что при перемещении элемента на своё место транспозициями соседних элементов все элементы остаются на своих местах, кроме двух соседних элементов, которые, возможно, меняются местами. Метод пузырька может быть выражен в форме алгоритма 5.1. Этот алгоритм прост, но является далеко не самым эффективным алгоритмом сортировки.

5.2.4. Генерация перестановок

На множестве перестановок естественным образом можно определить порядок на основе упорядоченности элементов. А именно, говорят, что перестановка $(a_1, \ldots, a_n)$ *лексикографически* предшествует перестановке $(b_1, \ldots, b_n)$, если

$$\exists\, k \leqslant n\ \ (a_k < b_k\ \&\ \forall\, i < k\ \ (a_i = b_i))\,.$$

Алгоритм 5.1. Сортировка методом пузырька

Вход: массив A : **array** $[1..n]$ **of** B, где значения элементов массива расположены в произвольном порядке, для значений типа B задано отношение $<$.

Выход: массив A : **array** $[1..n]$ **of** B, в котором значения расположены в порядке возрастания.

```
for i from 2 to n do
  for j from n downto i do
    if A[j] < A[j − 1] then
      A[j] ↔ A[j − 1]  // транспозиция соседних элементов
    end if
  end for
end for
```

Аналогично, говорят, что перестановка $(a_1, \ldots, a_n)$ *антилексикографически* предшествует перестановке $(b_1, \ldots, b_n)$, если

$$\exists\, k \leqslant n \;\; (a_k > b_k \;\&\; \forall\, i > k \;\; (a_i = b_i)) .$$

Следующий алгоритм генерирует все перестановки элементов $1..n$ в антилексикографическом порядке. Массив P : **array** $[1..n]$ **of** $1..n$ является глобальным и предназначен для хранения перестановок.

Алгоритм 5.2. Генерация перестановок в антилексикографическом порядке

Вход: n — количество элементов

Выход: последовательность перестановок элементов $1..n$ в антилексикографическом порядке.

```
for i from 1 to n do
  P[i] := i  // инициализация
end for
Antilex(n)  // вызов рекурсивной процедуры Antilex
```

Основная работа по генерации перестановок выполняется рекурсивной процедурой Antilex.

Вход: m — параметр процедуры — количество первых элементов массива P, для которых генерируются перестановки.

Выход: последовательность перестановок $1, \ldots, m$ в антилексикографическом порядке.

```
if m = 1 then
  yield P  // очередная перестановка
else
  for i from 1 to m do
    Antilex(m − 1)  // рекурсивный вызов
    if i < m then
      P[i] ↔ P[m]  // следующий элемент
      Reverse(m − 1)  // изменение порядка элементов
    end if
  end for
end if
```

Вспомогательная процедура Reverse переставляет элементы заданного отрезка массива P в обратном порядке.

Вход: k — номер элемента, задающий отрезок массива P, подлежащий перестановке в обратном порядке.

Выход: первые k элементов массива P переставлены в обратном порядке

```
j := 1  // нижняя граница обращаемого диапазона
while j < k do
  P[j] ↔ P[k]  // меняем местами элементы
  j := j + 1  // увеличиваем нижнюю границу
  k := k − 1  // уменьшаем верхнюю границу
end while
```

Обоснование. Заметим, что искомую последовательность перестановок n элементов можно получить из последовательности перестановок $n-1$ элементов следующим образом. Нужно выписать n блоков по $(n-1)!$ перестановок в каждом, соответствующих последовательности перестановок $n-1$ элементов в антилексикографическом порядке. Затем ко всем перестановкам в первом блоке нужно приписать справа n, во втором — $n-1$ и так далее в убывающем порядке. Затем в каждом из блоков (кроме первого), к перестановкам которого справа приписан элемент i, нужно в перестановках блока заменить все вхождения элемента i на элемент n. В полученной последовательности все перестановки различны, и их $n!$, то есть перечислены все перестановки. При этом антилексикографический порядок соблюден для последовательностей внутри одного блока, потому что этот порядок был соблюден в исходной последовательности, а для последовательностей на границах двух блоков — потому что происходит уменьшение самого правого элемента. Обратимся к процедуре Antilex — легко видеть, что в ней реализовано указанное построение. В основном цикле сначала строится очередной блок — последовательность перестановок первых $m-1$ элементов массива P (при этом элементы $P[m], \ldots, P[n]$ остаются неизменными). Затем элемент $P[m]$ меняется местами с очередным элементом $P[i]$. Вызов вспомогательной процедуры Reverse необходим, поскольку последняя перестановка в блоке является обращением первой, а для генерации следующего блока на очередном шаге цикла нужно восстановить исходный порядок. □

Пример. Последовательность перестановок в антилексикографическом порядке для $n = 3$: $(1,2,3)$, $(2,1,3)$, $(1,3,2)$, $(3,1,2)$, $(2,3,1)$, $(3,2,1)$.

5.2.5. Двойные факториалы

Содержание этого параграфа не имеет отношения к основной теме раздела — перестановкам и включено в раздел по сходству обозначений и с целью привести две формулы, которые иногда используются в практических комбинаторных расчётах и требуются в последующих разделах.

Двойным факториалом натурального числа n (обозначается $n!!$) называется произведение числа n и всех меньших натуральных чисел той же чётности.

Пример. $10!! = 10 \cdot 8 \cdot 6 \cdot 4 \cdot 2 = 3840$.

ТЕОРЕМА 1. 1. $(2k)!! = 2^k \cdot k!$.

2. $(2k+1)!! = \dfrac{(2k+1)!}{2^k \cdot k!}$.

Доказательство.

[1] $(2k)!! = 2 \cdot 4 \cdot \dots \cdot (2k-2) \cdot 2k = (2 \cdot 1) \cdot (2 \cdot 2) \cdot \dots \cdot (2 \cdot (k-1)) \cdot (2 \cdot k) =$
$= \underbrace{(2 \cdot 2 \cdot \dots \cdot 2 \cdot 2)}_{k \text{ раз}} \cdot (1 \cdot 2 \cdot \dots \cdot (k-1) \cdot k) = 2^k \cdot k!$.

[2] Достаточно заметить, что $(2k+1)! = (2k+1)!!(2k)!!$. □

По определению $0!! \stackrel{\text{Def}}{=} 1$.

Пример. *Перестановками Стирлинга* называются перестановки набора чисел $\{1, 1, 2, 2, \dots, k, k\}$ такие, что между любыми двумя равными числами могут находиться лишь бóльшие по значению.

Пример. $(1, 2, 3, 3, 2, 1, 4, 4)$.

ТЕОРЕМА 2. *Число перестановок Стирлинга равно* $(2k-1)!!$.

Доказательство. База: при $k = 1$ существует единственная перестановка $1, 1$, откуда $(2k-1)!! = 1$. Индукционный переход: пусть число перестановок Стирлинга $1, 1, 2, 2, \dots, k-1, k-1$ равно $(2(k-1)-1)!! = (2k-3)!!$. Так как в перестановке нет чисел бóльших k, то элементы k, k могут стоять только рядом. Их можно отбросить, получив перестановку с числами $1..(k-1)$. Учитывая, что существует $2k-1$ место, куда можно поставить пару k, k, получим $(2k-3)!! \cdot (2k-1) = (2k-1)!!$. □

5.2.6. Быстрая сортировка

Быстрая сортировка является наиболее широко применяемым и одним из самых эффективных алгоритмов сортировки. Приведём неформальное описание идеи алгоритма быстрой сортировки.

1. Из массива выбирается элемент x, называемый *опорным*.
2. Массив делится на две части относительно опорного элемента: элементы левой части меньше либо равны x, элементы правой части больше x.
3. Первые два шага рекурсивно применяются к левой и правой частям массива, если они состоят более чем из одного элемента.

Алгоритм 5.3. Быстрая сортировка

Вход: числа l и r — индексы начала и конца части массива для сортировки.
Выход: отсортированный по возрастанию массив $A[a..b]$.

```
if l < r then
    i := Partition(l, r)    // Разбиение массива
    Quicksort(l, i − 1)     // Рекурсивный вызов для левой части
    Quicksort(i + 1, r)     // Рекурсивный вызов для правой части
end if
```

Массив A : **array** $[a..b]$ **of** B, подлежащий сортировке, является глобальным. Основная работа выполняется функцией Quicksort. Сортировка массива $A[a..b]$ выполняется вызовом функции Quicksort(a, b).

Функция Partition находит опорный элемент.

Вход: числа l и r — индексы начала и конца части массива для поиска.
Выход: индекс i опорного элемента.

```
x := A[r]   // опорным является последний элемент массива
i := l − 1   // инициализация за пределами массива
for j from l to r − 1 do
  if A[j] ⩽ x then
    i := i + 1; A[i] ↔ A[j]
  end if
end for
A[i] ↔ A[r] return i  // опорный элемент помещается в нужное место
return i
```

Ключевой частью алгоритма быстрой сортировки является разбиение массива. На каждой итерации цикла должны выполняться четыре условия:

1) $A[l \ldots i]$ содержит элементы, которые меньше или равны опорному;
2) $A[i+1 \ldots j-1]$ содержит элементы, которые больше опорного;
3) $A[j \ldots r-1]$ содержит и те, и другие элементы;
4) На позиции r находится опорный элемент (рис. 5.4).

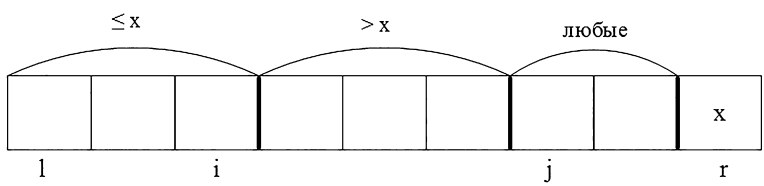


Рис. 5.4. К алгоритму быстрой сортировки

Обоснование.

[Корректность разбиения] При инициализации цикла между l и i нет элементов, а значит, первые три условия выполняются. Элемент с индексом r принят опорным. На каждой итерации цикла необходимо рассмотреть два случая, которые определяются условием $A[j] \leqslant A[r]$. Если $A[j] > A[r]$, тогда следует добавить элемент $A[j]$ в множество элементов бóльших опорного, значит, следует увеличить индекс $j := j + 1$. Если же $A[j] \leqslant x$, тогда элемент $A[j]$ подлежит добавлению в множество элементов меньших или равных опорному. Чтобы это сделать, необходимо увеличить индекс $i := i + 1$, выполнить обмен элементов $A[i] \leftrightarrow A[j]$, а затем, чтобы сохранить множество элементов бóльших опорного, следует увеличить индекс $j := j + 1$.

После выполнения цикла массив разобьётся на три непересекающихся множества. Первое множество состоит из элементов меньших или равных опорному, второе — бóльших опорного, третье — одноэлементное множество из опорного элемента.

[Корректность сортировки] Пусть n — длина входного массива. База: $n = 1$ — массив уже отсортирован. Индукционный переход: входной массив A размера n после вызова процедуры Partition разбивается на два подмассива размера меньше n (опорный элемент не рассматривается, так как уже находится на своём месте), которые корректно сортируются по индукционному предположению. □

Пример. Отсортируем массив $A = \{8, 15, 24, 18, 9, 49, 20\}$. Полужирным шрифтом отмечены опорные элементы, а в прямоугольниках находятся элементы, которые уже встали на своё место.

Опорный элемент	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$	$A[7]$
20	8	15	24	18	9	49	**20**
9, 24	8	15	18	**9**	20	49	**24**
8, 15, 49	**8**	9	18	**15**	20	24	**49**
18	8	9	15	**18**	20	24	49
	8	9	15	18	20	24	49

5.2.7. Трудоёмкость в среднем

Трудоёмкость алгоритма в среднем — математическое ожидание количества вычислительного ресурса (памяти или времени), используемого алгоритмом при заданном распределении вероятности появления тех или иных входных данных P_X (п. 5.1.7).

ЗАМЕЧАНИЕ

О трудоёмкости в среднем имеет смысл говорить только для тех алгоритмов, время работы которых зависит от P_X.

Пример. Время работы алгоритма Уоршалла (п. 1.5.2) не зависит от распределения входных данных, а зависит только от размера данных.

Рассмотрим оценки вычислительной сложности для алгоритма «бинарного поиска» (см. п. 9.4.3) на входном массиве длины n.

Оценка снизу. В лучшем случае искомый элемент находится на позиции с индексом $\frac{n}{2}$. Значит, потребуется только одно сравнение.

Оценка в среднем. Для упрощения вычислений положим $n = 2^k - 1, k \in \mathbb{N}$. Если искомый элемент имеет индекс $i = \frac{n}{2}$, то мы его найдём за одно сравнение. Если же $i = \frac{n}{4}$ или $i = \frac{3n}{4}$, то нам понадобится уже два сравнения. Таким образом, общее количество сравнений равно $\sum_{i=1}^{k} i2^{i-1}$. Так как искомый элемент может находиться на любой из позиций с вероятностью $\frac{1}{n}$, среднее количество сравнений равно $\frac{1}{2^k-1}\sum_{i=1}^{k} i2^{i-1} = \sum_{i=1}^{k} \frac{i2^i}{2(2^k-1)} = \frac{2((k-1)2^k+1)}{2(2^k-1)}$. Сокращая дробь и переходя к переменной n, имеем:

$$\frac{(\log_2(n+1)-1)(n+1)+1}{n} = \log_2(n+1) - 1 + \frac{(\log_2(n+1))}{n}.$$

Оценка сверху. Основываясь на рассуждениях для оценки в среднем, можно увидеть, что в худшем случае количество сравнений, выполненных алгоритмом бинарного поиска, равно $\log_2(n+1)$, при $n = 2^k - 1, k \in \mathbb{N}$.

Оценки в среднем и худшем случаях случаях для «бинарного поиска» имеют величины порядка $O(\log n)$. Таким образом, алгоритм бинарного поиска хорош тем, что для него нет плохих случаев.

ЗАМЕЧАНИЕ

Распределение входных данных, как и трудоёмкость в среднем, зависит от конкретной задачи, которую необходимо исследовать.

5.2.8. Гармонические числа

Гармонические числа $H(n)$ определяются следующим образом:

$$H(0) \stackrel{\text{Def}}{=} 0 \quad H(n) \stackrel{\text{Def}}{=} H(n-1) + \frac{1}{n}.$$

Применив определение $n-1$ раз, получим ($n \geqslant 1$):

$$H(n) = H(n-1) + \frac{1}{n} = H(n-2) + \frac{1}{n} + \frac{1}{n-1} = \cdots = \sum_{k=1}^{n} \frac{1}{k}.$$

По аналогии можно ввести понятие *обобщенных гармонических чисел порядка* m:

$$H(m,0) \stackrel{\text{Def}}{=} 0, \quad H(m,n) \stackrel{\text{Def}}{=} H(m,n-1) + 1/n^m \quad \text{или} \quad H(m,n) = \sum_{k=1}^{n} \frac{1}{k^m}.$$

ЗАМЕЧАНИЕ

Из курса математического анализа известна формула для приближенного вычисления гармонических чисел, которая в упрощённом виде записывается следующим образом:

$$H_n = \ln n + \gamma + \frac{1}{2n} + O\left(\frac{1}{n^2}\right), \lim_{n\to\infty}(H_n - \ln n) = \gamma \approx 0.57721.$$

Гармонические числа используются для вывода оценки в среднем некоторых алгоритмов.

5.2.9. Оценка в среднем для быстрой сортировки

ТЕОРЕМА. *Время работы алгоритма быстрой сортировки в среднем* $O(n \log n)$.

Доказательство. Пусть Q_n — среднее количество сравнений, выполняемых алгоритмом, когда входные данные — это массив размера n. Положим $n \geqslant 2$. Имеем $Q_0 = Q_1 = 0$. Известно, что после выбора опорного элемента необходимо провести $n-1$ сравнение со всеми оставшимися элементами. Опорный элемент разобьёт массив на две части. Первая часть может быть любого размера от 0 до $n-1$. Пусть первая часть имеет размер k, значит, вторая часть имеет размер $n-1-k$. Более того, $k \in \{0,\ldots,n-1\}$ равновероятны. Тогда имеем следующее равенство для математического ожидания (п. 5.1.7):

$$Q_n = (n-1) + \frac{1}{n}\sum_{k=0}^{n-1} Q_k + Q_{n-1-k}.$$

Заметим, что $\sum_{k=0}^{n-1} Q_{n-1-k} = \sum_{k=0}^{n-1} Q_k$, значит, $Q_n = (n-1) + \frac{2}{n}\sum_{k=0}^{n-1} Q_k$. Домножим на n, имеем:

$$nQ_n = n(n-1) + 2\sum_{k=0}^{n-1} Q_k.$$

Заменим $n := n-1$, что возможно, поскольку $n \geqslant 2$. Получаем

$$(n-1)Q_{n-1} = (n-1)(n-2) + 2\sum_{k=0}^{n-2} Q_k.$$

Вычтем выражения, получим:

$$nQ_n - (n-1)Q_{n-1} = n(n-1) - (n-1)(n-2) + 2Q_{n-1},$$

откуда имеем

$$nQ_n = 2(n-1) + (n+1)Q_{n-1}.$$

Разделив последнее равенство на $n(n+1)$, получим

$$\frac{Q_n}{n+1} = 2\frac{n-1}{n(n+1)} + \frac{Q_{n-1}}{n}.$$

Введём новую переменную: $F_n := \frac{Q_n}{n+1}$. Имеем

$$F_n = 2\frac{n-1}{n(n+1)} + F_{n-1}, \quad F_n = 2\sum_{k=1}^{n} \frac{k-1}{k(k+1)}.$$

Разобьём суммируемое выражение на простейшие дроби: $\frac{k-1}{k(k+1)} = \frac{2}{1+k} - \frac{1}{k}$, откуда

$$F_n = 4\sum_{k=1}^{n} \frac{1}{1+k} - 2\sum_{k=1}^{n} \frac{1}{k}$$

или

$$F_n = 4(H_{n+1} - 1) - 2H_n = 4H_{n+1} - 2H_n - 4.$$

Получили выражение в терминах гармонических чисел:

$$Q_n = (n+1)(4H_{n+1} - 2H_n - 4).$$

После разложения гармонических чисел в ряд окончательно получаем:
$Q_n = 2n(\ln n + \gamma - 2) + 2\ln n + 2\gamma + 1 + O\left(\frac{1}{n}\right)$. □

5.3. Биномиальные коэффициенты

Число сочетаний $C(m,n)$ — это число различных n-элементных подмножеств m-элементного множества. Числа $C(m,n)$ обладают целым рядом свойств, рассматриваемых в этом разделе, которые оказываются очень полезными при решении различных комбинаторных задач.

5.3.1. Элементарные тождества

Основная формула для числа сочетаний $C(m,n) = \dfrac{m!}{n!(m-n)!}$ позволяет получить следующие простые тождества.

ТЕОРЕМА. 1. $C(m,n) = C(m,m-n)$.

2. $C(m,n) = C(m-1,n) + C(m-1,n-1)$.

3. $C(n,i)C(i,m) = C(n,m)C(n-m,i-m)$.

Доказательство.

[1] $C(m,m-n) = \dfrac{m!}{(m-n)!\,(m-(m-n))!} = \dfrac{m!}{(m-n)!\,n!} = C(m,n).$

[2] $C(m-1,n) + C(m-1,n-1) = \dfrac{(m-1)!}{n!(m-n-1)!} + \dfrac{(m-1)!}{(n-1)!\,(m-1-(n-1))!} =$

$= \dfrac{(m-1)!}{n(n-1)!\,(m-n-1)!} + \dfrac{(m-1)!}{(n-1)!\,(m-n)(m-n-1)!} =$

$= \dfrac{(m-n)(m-1)! + n(m-1)!}{n(n-1)!\,(m-n)(m-n-1)!} = \dfrac{(m-n+n)(m-1)!}{n!\,(m-n)!} = \dfrac{m!}{n!\,(m-n)!} = C(m,n).$

[3] $C(n,i)C(i,m) = \dfrac{n!}{i!(n-i)!}\dfrac{i!}{m!(i-m)!} = \dfrac{n!}{m!(i-m)!(n-i)!} =$

$= \dfrac{n!(n-m)!}{m!(i-m)!(n-i)!(n-m)!} = \dfrac{n!}{m!(n-m)!}\dfrac{(n-m)!}{(i-m)!(n-i)!} =$

$= C(n,m)C(n-m,i-m).$ □

5.3.2. Бином Ньютона

Числа сочетаний $C(m,n)$ называются также *биномиальными коэффициентами*. Смысл этого названия устанавливается следующей теоремой, известной также как формула *бинома Ньютона*[1].

ТЕОРЕМА. $(x+y)^m = \sum\limits_{n=0}^{m} C(m,n)x^n y^{m-n}$.

Доказательство. По индукции. База, $m = 1$:

$(x+y)^1 = x + y = 1x^1y^0 + 1x^0y^1 = C(1,0)x^1y^0 + C(1,1)x^0y^1 = \sum\limits_{n=0}^{1} C(1,n)x^n y^{1-n}.$

[1] Исаак Ньютон (1643–1727).

Индукционный переход:

$$(x+y)^m = (x+y)(x+y)^{m-1} = (x+y)\sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n-1} =$$
$$= \sum_{n=0}^{m-1} xC(m-1,n)x^n y^{m-n-1} + \sum_{n=0}^{m-1} yC(m-1,n)x^n y^{m-n-1} =$$
$$= \sum_{n=0}^{m-1} C(m-1,n)x^{n+1} y^{m-n-1} + \sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n} =$$
$$= \sum_{n=1}^{m} C(m-1,n-1)x^n y^{m-n} + \sum_{n=0}^{m-1} C(m-1,n)x^n y^{m-n} =$$
$$= C(m-1,0)x^0 y^m + \sum_{n=1}^{m-1} (C(m-1,n-1) + C(m-1,n))x^n y^{m-n} +$$
$$+ C(m-1,m-1)x^m y^0 = \sum_{n=0}^{m} C(m,n)x^n y^{m-n}.$$

□

СЛЕДСТВИЕ 1. $\sum\limits_{n=0}^{m} C(m,n) = 2^m.$

Доказательство. $2^m = (1+1)^m = \sum\limits_{n=0}^{n} C(m,n)1^n 1^{m-n} = \sum\limits_{n=0}^{m} C(m,n).$ □

СЛЕДСТВИЕ 2. $\sum\limits_{n=0}^{m} (-1)^n C(m,n) = 0.$

Доказательство. $0 = (-1+1)^m = \sum\limits_{n=0}^{m} C(m,n)(-1)^n 1^{m-n} = \sum\limits_{n=0}^{m} (-1)^n C(m,n).$ □

5.3.3. Свойства биномиальных коэффициентов

Биномиальные коэффициенты обладают целым рядом замечательных свойств.

ТЕОРЕМА 1. $\sum\limits_{n=0}^{m} nC(m,n) = m2^{m-1}.$

Доказательство. Рассмотрим следующую последовательность, составленную из чисел $1,\ldots,m$. Сначала выписаны все подмножества длины 0, потом все подмножества длины 1 и т. д. Имеется $C(m,n)$ подмножеств мощности n, и каждое из них имеет длину n, таким образом, всего в этой последовательности $\sum\limits_{n=0}^{m} nC(m,n)$ чисел. С другой стороны, каждое число x входит в эту последовательность $2^{|\{1,\ldots,m\}\setminus\{x\}|} = 2^{m-1}$ раз, а всего чисел m. □

ТЕОРЕМА 2. $C(m+n,k) = \sum\limits_{i=0}^{k} C(m,i)C(n,k-i)$.

Доказательство. $C(n+m,k)$ — это число способов выбрать k предметов из $m+n$ предметов. Предметы можно выбирать в два приема: сначала выбрать i предметов из первых m предметов, а затем выбрать недостающие $k-i$ предметов из оставшихся n предметов. Отсюда общее число способов выбрать k предметов составляет $\sum\limits_{i=0}^{k} C(m,i)C(n,k-i)$. □

ЗАМЕЧАНИЕ

Последнее свойство известно как *тождество Коши*[1].

ОТСТУПЛЕНИЕ

Данные свойства биномиальных коэффициентов нетрудно доказать, проводя алгебраические выкладки обычным образом. Например, первое свойство можно получить так:

$$\sum_{n=0}^{m} nC(m,n) = \sum_{n=1}^{m} \frac{nm!}{(m-n)!n!} = \sum_{n=1}^{m} \frac{m(m-1)!}{(m-n)!(n-1)!} =$$
$$= m\sum_{n=0}^{m-1} \frac{(m-1)!}{(m-n-1)!n!} = m\sum_{n=0}^{m-1} C(m-1,n) = m2^{m-1}.$$

В доказательстве использованы рассуждения «в комбинаторном духе», чтобы проиллюстрировать «неалгебраические» способы решения комбинаторных задач.

5.3.4. Треугольник Паскаля

Из второй формулы теоремы п. 5.3.1 вытекает эффективный способ рекуррентного вычисления значений биномиальных коэффициентов, который можно представить в графической форме, известной как *треугольник Паскаля*[2].

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      .   .   .   .   .   .
```

В этом равнобедренном треугольнике каждое число (кроме единиц на боковых сторонах) является суммой двух чисел, стоящих над ним. Число сочетаний $C(m,n)$ находится в $(m+1)$-м ряду на $(n+1)$-м месте.

Рассмотрим вопрос реализации процесса вычислений биномиальных коэффициентов. Если по рекуррентной формуле $C(m,n) = C(m-1,n) + C(m-1,n-1)$ составить рекурсивную процедуру, то можно столкнуться с переполнением стека

[1] Огюстен Луи Коши (1789–1857).
[2] Блез Паскаль (1623–1662).

вызовов из-за большой глубины рекурсии. Кроме того, прямая рекурсивная реализация сугубо неэффективна, поскольку при рекурсивной реализации один и тот же биномиальный коэффициент будет вычисляться несколько раз. Например:

$$C(5,3) = C(4,3) + C(4,2) = C(3,3) + \underline{C(3,2)} + \underline{C(3,2)} + C(3,1) = \dots \ .$$

Для случая рекурсивного вычисления $C(m,n)$ получаем 2^n сложений, так как на каждом шаге рекурсии количество вычислений увеличивается в два раза, а всего шагов n. А для нерекурсивного метода имеем n сложений, то есть рекурсивный метод экспоненциально хуже нерекурсивного.

Замечательные свойства треугольника Паскаля позволяют организовать процесс вычислений намного эффективнее. Представим треугольник в другой форме, повернув его на 45 градусов влево. Получится бесконечная матрица, в которой верхняя строка и левый столбец содержат единицы, а каждый внутренний элемент является суммой двух элементов: соседнего слева и соседнего сверху.

1	1	1	1	1	1
1	2	3	4	5	
1	3	6	10		
1	4	10			
1	5				
1					

Алгоритм 5.4. Алгоритм рекуррентного вычисления числа сочетаний

Вход: целые неотрицательные числа m и n
Выход: число сочетаний $C(m,n)$

```
A : array[0..n] of int  // рабочий массив
if n = 1 ∨ m = n + 1 then return m end if  // по определению
if n > m then return 0 end if  // по определению
if n = m then return 1 end if  // по определению
if n > m − n then n := m − n end if  // в силу симметрии
for i from 0 to n do
   A[i] := 1  // первая строка треугольника Паскаля
end for
for i from 1 to m − n do
   for j from 1 to n do
      A[j] := A[j] + A[j − 1]  // согласно рекуррентной формуле
   end for
end for
return A[n]
```

Обоснование. Сначала в алгоритме проверяются граничные случаи, для которых значения заданы определениями. Значение (m,n) находится в $(m-n+1)$-й строке на $(n+1)$-м месте. Весь треугольник хранить в памяти нет нужды, достаточно хранить только одну строку. Первая строка содержит все единицы, а каждая следующая получается перевычислением слева направо по формуле $A[j] := A[j]+A[j-1]$. При этом в правой части оператора $A[j]$ — это «старое» значение, то есть элемент из «верхней» строки, а $A[j-1]$ — только что вычисленное значение слева. Перевычислив таким образом массив $m-n$ раз, получаем $C(m,n) = A[n]$. □

Пример. Рассмотрим вычисление $C(5,2)$. Здесь $m=5$, $n=2$, $m-n=3$.

i	A		
	1	1	1
1	1	2	3
2	1	3	6
3	1	4	10

ЗАМЕЧАНИЕ

В силу симметрии треугольника Паскаля можно было бы использовать массив большей длины $m-n+1$ и сократить количество повторений основного цикла по i. Однако это не даёт сокращения вычислений: необходимое количество сложений остаётся $n(n-m)$.

ОТСТУПЛЕНИЕ

Для доказательства применимости и полезности агоритма 5.3 оценим максимальные значения числа сочетаний, которые можно вычислить по основной формуле и по алгоритму 5.3. Предположим, что используется 32-разрядная арифметика, беззнаковые целые числа, максимально представимое число $2^{32}-1$. При фиксированном m биномиальный коэффициент $C(m,n)$ имеет максимум для $n=m/2$. Найдём максимально допустимые $C(m,n)$ и m, для которых возможно вычисление. Для формулы это $C(12,6)=924$, поскольку вычислить $m!$ уже не представляется возможным при $m>12$ для используемого представления чисел в компьютере. Использование формулы $C(m,n)=((m-n+1)\cdot(m-n+2)\cdot\ldots\cdot m)/(m-n)!$ не приведёт к существенному расширению диапазона вычисляемых значений. А для алгоритма 5.3 это максимальное $m=34$, $C(34,17)=2333606220$. Если же варьировать n от 0 до m, то получим максимальное значение $C(222,5)=4294249674$, что близко к 2^{32}, то есть к максимальному представимому числу. Значит, алгоритм 5.3 применим для гораздо более широкого набора чисел m и n по сравнению с замкнутой формулой из параграфа 5.1.5, что делает его весьма полезным.

5.3.5. Генерация подмножеств

Элементы множества $\{1,\ldots,m\}$ естественным образом упорядочены. Поэтому каждое n-элементное подмножество этого множества также можно рассматривать как упорядоченную последовательность. На множестве таких последовательностей определяется лексикографический порядок. Следующий простой алгоритм генерирует все n-элементные подмножества m-элементного множества в лексикографическом порядке.

Обоснование. Заметим, что в искомой последовательности n-элементных подмножеств (каждое из которых является возрастающей последовательностью n чисел из диапазона $1..m$) вслед за последовательностью $(a_1,\ldots,a_n)$ следует последовательность $(b_1,\ldots,b_n)=(a_1,\ldots,a_{p-1},a_p+1,a_p+2,\ldots,a_p+n-p+1)$, где p — максимальный индекс, для которого $b_n=a_p+n-p+1\leqslant m$. Другими словами, следующая последовательность получается из предыдущей заменой некоторого количества элементов в хвосте последовательности идущими подряд натуральными числами, но так, чтобы последний элемент не превосходил m, а первый изменяемый элемент был на 1 больше, чем соответствующий элемент в предыдущей последовательности. Таким образом, индекс p, начиная с которого следует изменить «хвост последовательности», определяется по значению элемента $A[n]$. Если $A[n]<m$, то следует изменять только $A[n]$, и при этом $p:=n$. Если же уже $A[n]=m$, то нужно уменьшать

Алгоритм 5.5. Генерация n-элементных подмножеств m-элементного множества

Вход: n — мощность подмножества, m — мощность множества, $m \geqslant n > 0$.
Выход: последовательность всех n-элементных подмножеств m-элементного множества в лексикографическом порядке.

```
for i from 1 to m do
    A[i] := i  // инициализация исходного множества
end for
if m = n return A[1..n] end if
if m = n then
    return A[1..n]  // единственное подмножество
end if
p := n  // p — номер первого изменяемого элемента
while p ⩾ 1 do
    yield A[1..n]  // очередное подмножество в первых n элементах массива A
    if A[n] = m then
        p := p − 1  // нельзя увеличить последний элемент
    else
        p := n  // можно увеличить последний элемент
    end if
    if p ⩾ 1 then
        for i from n downto p do
            A[i] := A[p] + i − p + 1  // увеличение элементов
        end for
    end if
end while
```

индекс $p := p - 1$, увеличивая длину изменяемого хвоста. □

Пример. Последовательность n-элементных подмножеств m-элементного множества в лексикографическом порядке для $n = 3$ и $m = 4$: $(1, 2, 3)$, $(1, 2, 4)$, $(1, 3, 4)$, $(2, 3, 4)$.

ОТСТУПЛЕНИЕ

Довольно часто встречаются задачи, в которых требуется найти в некотором множестве элемент, обладающий определёнными свойствами. При этом исследуемое множество может быть велико, а его элементы устроены достаточно сложно. Если неизвестно заранее, как именно следует искать элемент, то остаётся перебирать все элементы, пока не попадётся нужный (поэтому такие задачи называют *переборными*). При решении переборных задач совсем необязательно и, как правило, нецелесообразно строить всё исследуемое множество (*пространство поиска*) в явном виде. Достаточно иметь итератор (п. 1.3.9), перебирающий элементы множества в подходящем порядке. Фактически, многие рассмотренные алгоритмы являются примерами таких итераторов для некоторых типичных пространств поиска. Чтобы использовать эти алгоритмы в качестве итераторов при переборе, достаточно вставить вместо оператора **yield** проверку того, что очередной элемент является искомым.

5.3.6. Расширенные биномиальные коэффициенты

Сократим множитель $(m-n)!$ в формуле для числа сочетаний:

$$C(m,n)=\frac{m(m-1)\dots(m-n+1)}{n!}.$$

При определении чисел $C(m,n)$ по смыслу комбинаторной конфигурации параметры натуральны: $m,n\in\mathbb{N}$. Однако данная формула сохраняет свою вычислимость и при отрицательных значениях m, что позволяет ввести в рассмотрение *расширенные биномиальные коэффициенты* для $m\in\mathbb{Z}$ и $n\in\mathbb{N}_0$.

ЗАМЕЧАНИЕ

В качестве области определения расширенных биномиальных коэффициентов можно рассматривать также рациональные $m,n\in\mathbb{Q}$ и вещественные значения $m,n\in\mathbb{R}$.

Пусть $m,n\in\mathbb{Z}$. Положим $C(m,n)\stackrel{\text{Def}}{=}0$ при $m<n$ или $n<0$ и положим

$$C(m,n)\stackrel{\text{Def}}{=}(m(m-1)\dots(m-n+1))/n!\quad\text{при}\quad m\geqslant n,n\geqslant 0.$$

В этих определениях $\forall\, m\in\mathbb{Z}\ \ (C(m,0)=1)$ и $\forall\, n\in\mathbb{Z},n\neq 0\ \ (C(0,n)=0)$.

ТЕОРЕМА. $C(m,n)=(-1)^nC(n-m-1,n)$.

Доказательство. Заметим, что

$$\begin{aligned}m(m-1)\dots(m-n+1)&=(-1)^n(-m)(-m+1)\dots(-m+n-1)=\\&=(-1)^n(n-m-1)\dots(n-m-1-(n-1)).\end{aligned}$$

Откуда

$$\begin{aligned}C(m,n)&=m\dots(m-n+1)/n!=(-1)^n(n-m-1)\dots(-m)/n!=\\&=(-1)^nC(n-m-1,n).\end{aligned}$$

□

Доказанное тождество можно использовать для вычисления биномиальных коэффициентов с отрицательным первым параметром по уже вычисленным значениям коэффициентов с положительными параметрами.

ЗАМЕЧАНИЕ

В теореме п. 5.3.1 при доказательстве тождества $C(m,n)=C(m-1,n-1)+C(m-1,n)$ нигде не использовано то, что число $m>0$, что позволяет организовать рекуррентное вычисление биномиальных коэффициентов и для случая $m<0$.

Используя эти тождества, можно построить *расширенный треугольник Паскаля*, в котором положительные значения m распространяются вправо, отрицательные значения m распространяются влево, а n распространяется вниз. Поскольку при $m=0$ имеем $C(m,n)=0$, средний столбец, обозначенный n, содержит сами значения n, подобно тому, как верхняя строка содержит значения m.

-4	-3	-2	-1	$\frac{m}{n}$	0	1	2	3	4
1	1	1	1	0	1	1	1	1	1
-4	-3	-2	-1	1	0	1	2	3	4
10	6	3	1	2	0	0	1	3	6
-20	-10	-4	-1	3	0	0	0	1	4
35	15	5	1	4	0	0	0	0	1

На область применения биномиальных коэффициентов накладывается меньше ограничений, что может облегчить доказательство теорем и тождеств.

Пример. Докажем утверждение $C(m,n) = \frac{m}{n}C(m-1,n-1)$.

При использовании формулы $C(m,n) = m!/n!/(m-n)!$ доказательство требует разбора случаев. Первый случай. При $m \geqslant n$ имеем
$C(m,n) = \frac{m!}{n!(m-n)!} = \frac{m}{n}\frac{(m-1)!}{(n-1)!((m-1)-(n-1))!} = \frac{m}{n}C(m-1,n-1)$.
Второй случай. При $m < n$ имеем $C(m,n) = 0$ и $C(m-1,n-1) = 0$.

При использовании расширенных биномиальных коэффициентов сразу имеем $C(m,n) = m(m-1)\dots(m-n+1)/n! = (m/n)(m-1)\dots(m-n+1)/(n-1)! = (m/n)C(m-1,n-1)$.

5.3.7. Мультимножества и последовательности

В п. 5.1.5 показано, что число n-элементных подмножеств m-элементного множества равно $C(m,n)$. Поскольку n-элементные подмножества — это n-элементные индикаторы (п. 1.1.4), ясно, что число n-элементных индикаторов над m-элементным множеством также равно $C(m,n)$.

Общее число n-элементных мультимножеств $[x_1^{n_1}\dots x_m^{n_m}]$ над m-элементным множеством $X = \{x_1,\dots,x_m\}$, в которых $n_1 + \dots + n_m = n$, нетрудно определить, если заметить, что это число способов разложить n мячей по m ящикам, то есть число сочетаний с повторениями $V(m,n) = C(n+m-1,n)$. Рассмотрим последовательность $Y = (y_1,\dots,y_n)$, где $\forall\, i\ \ (y_i \in X)$, и в последовательности Y элемент x_i встречается n_i раз. Тогда мультимножество $\widehat{X} = [x_1^{n_1},\dots,x_m^{n_m}]$ называется *составом* последовательности Y. Ясно, что состав любой последовательности определяется однозначно, но разные последовательности могут иметь один и тот же состав.

Пример. Пусть $X = \{1,2,3\}$, $Y_1 = \{1,2,3,2,1\}$, $Y_2 = \{1,1,2,2,3\}$. Тогда последовательности Y_1 и Y_2 имеют один и тот же состав $\widehat{X} = \left[1^2,2^2,3^1\right]$.

ОТСТУПЛЕНИЕ

Из органической химии известно, что существуют различные вещества, имеющие один и тот же химический состав. Они называются изомерами.

Очевидно, что все последовательности над множеством $X = \{x_1,\dots,x_m\}$, имеющие один и тот же состав $\widehat{X} = [x_1^{n_1},\dots,x_m^{n_m}]$, имеют одну и ту же длину $n = n_1 + \dots + n_m$. Обозначим число последовательностей одного состава $C(n;n_1,\dots,n_m)$.

ТЕОРЕМА. $C(n;n_1,\dots,n_m) = \dfrac{n!}{n_1!\dots n_m!}$.

Доказательство. Рассмотрим мультимножество $\widehat{X} = [x_1^{n_1}, \ldots, x_m^{n_m}]$, которое можно записать в форме последовательности $x_1, \ldots, x_1, x_2, \ldots, x_2, \ldots, x_m, \ldots, x_m$, где элемент x_i встречается n_i раз. Ясно, что все последовательности одного состава $\widehat{X}$ получаются из указанной последовательности перестановкой элементов. При этом если переставляются одинаковые элементы, например x_i, то получается та же самая последовательность. Таких перестановок $n_i!$. Таким образом, общее число перестановок $n_1! \ldots n_m! C(n; n_1, \ldots, n_m)$. С другой стороны, число перестановок n элементов равно $n!$. □

5.3.8. Мультиномиальные коэффициенты

Числа $C(n; n_1, \ldots, n_m)$ встречаются в практических задачах довольно часто, поскольку обобщают случай индикаторов, которые описываются биномиальными коэффициентами, на случай произвольных мультимножеств. В частности, справедлива формула, обобщающая формулу бинома Ньютона, в которой участвуют числа $C(n; n_1, \ldots, n_m)$, поэтому их иногда называют *мультиномиальными коэффициентами*.

Теорема. $(x_1 + \ldots + x_m)^n = \sum\limits_{n_1+\ldots+n_m=n} C(n; n_1, \ldots, n_m) x_1^{n_1} \cdots x_m^{n_m}$.

Доказательство. Индукция по m. База: при $m = 2$ по формуле бинома Ньютона имеем

$$(x_1 + x_2)^n = \sum_{i=0}^{n} C(n,i) x_1^i x_2^{n-i} = \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} x_1^i x_2^{n-i} = \sum_{n_1+n_2=n} \frac{n!}{n_1! n_2!} x_1^{n_1} x_2^{n_2}.$$

Пусть теперь $(x_1 + \ldots + x_{m-1})^n = \sum\limits_{n_1+\ldots+n_{m-1}=n} C(n; n_1, \ldots, n_{m-1}) x_1^{n_1} \cdots x_{m-1}^{n_{m-1}}$.

Рассмотрим $(x_1 + \ldots + x_m)^n$. Имеем

$$\begin{aligned}
(x_1 + \ldots + x_m)^n &= ((x_1 + \ldots + x_{m-1}) + x_m)^n = \sum_{i=0}^{n} C(n,i)(x_1 + \ldots + x_{m-1})^i x_m^{n-i} = \\
&= \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} \left(\sum_{n_1+\ldots+n_{m-1}=i} \frac{i!}{n_1! \cdots n_{m-1}!} x_1^{n_1} \cdots x_{m-1}^{n_{m-1}} \right) x_m^{n-i} = \\
&= \sum_{i=0}^{n} \sum_{n_1+\ldots+n_{m-1}=i} \frac{n! i!}{i!(n-i)! n_1! \cdots n_{m-1}!} x_1^{n_1} \cdots x_{m-1}^{n_{m-1}} x_m^{n-i} = \\
&= \sum_{n_1+\ldots+n_{m-1}+n_m=n} \frac{n!}{n_1! \cdots n_{m-1}! n_m!} x_1^{n_1} \cdots x_{m-1}^{n_{m-1}} x_m^{n_m},
\end{aligned}$$

где $n_m := n - i$. □

Следствие. $\sum\limits_{n_1+\ldots+n_m=n} \frac{n!}{n_1! \cdots ! n_m!} = m^n$.

Доказательство. $m^n = \underbrace{(1+\ldots+1)}_{m \text{ раз}}^n = \sum\limits_{n_1+\ldots+n_m=n} \dfrac{n!}{n_1!\ldots!n_m!} 1^{n_1} 1^{n_m} =$

$= \sum\limits_{n_1+\cdots+n_m=n} \dfrac{n!}{n_1!\ldots!n_m!}.$ □

Пример. При игре в преферанс трём играющим сдаётся по 10 карт из 32 карт и 2 карты остаются в «прикупе». Сколько существует различных раскладов?

$$C(32; 10, 10, 10, 2) = \frac{32!}{10!10!10!2!} = 2\ 753\ 294\ 408\ 504\ 640.$$

5.3.9. Биномиальная система счисления

k-биномиальной системой счисления называется непозиционная система счисления, где каждое натуральное число n представляется в виде:

$$n = C(b_1, 1) + C(b_2, 2) + \cdots + C(b_k, k) = \sum_{i=1}^{k} C(b_i, i),$$

где $k \in \mathbb{N}$, $0 \leqslant b_1 < b_2 < \cdots < b_k$.

Пример. Пусть k = 2. Тогда
$1 = C(0,1) + C(2,2) = 0 + 1,$
$2 = C(1,1) + C(2,2) = 1 + 1,$
$3 = C(0,1) + C(3,2) = 0 + 3,$
$4 = C(1,1) + C(3,2) = 1 + 4,$
$5 = C(2,1) + C(3,2) = 2 + 3,$
$6 = C(0,1) + C(4,2) = 0 + 6$
и т. д.

ЗАМЕЧАНИЕ

По определению, $C(p, q) = 0$ при целых p и q таких, что $0 \leqslant p < q$.

Покажем состоятельность данной системы счисления.

ТЕОРЕМА. *При заданном натуральном значении k для любого натурального числа n существуют и единственны натуральные числа $b_1, \ldots, b_k$, такие что*

$$n = C(b_1, 1) + C(b_2, 2) + \cdots + C(b_k, k) = \sum_{i=1}^{k} C(b_i, i) \quad \textit{и} \quad 0 \leqslant b_1 < b_2 < \cdots < b_k.$$

Доказательство.

[Существование] Всякой паре натуральных чисел n и k сопоставим набор строго возрастающих натуральных чисел $0 \leqslant b_1 < b_2 < \cdots < b_k$ по следующему правилу. Сначала выберем $b_k \geqslant 0$ так, чтобы удовлетворялись неравенства

$$C(b_k, k) \leqslant n < C(b_k + 1, k).$$

Такой выбор однозначен, потому что по основному рекуррентному соотношению $C(b_k+1,k) = C(b_k,k) + C(b_k,k-1)$ и значит существует единственное число b_k такое что $C(b_k,k) \leqslant n < C(b_k,k) + C(b_k,k-1)$. Затем определим b_{k-1} так, чтобы

$$C(b_{k-1},k-1) \leqslant n - C(b_k,k) < C(b_{k-1}+1,k-1).$$

При таких условиях обязательно выполняется неравенство $b_k > b_{k-1}$. Действительно, $C(b_{k-1},k-1) \leqslant n - C(b_k,k) < C(b_k+1,k) - C(b_k,k) = C(b_k,k-1)$, откуда $b_{k-1} < b_k$. Продолжая действовать подобным образом, определим числа $b_{k-2}, b_{k-3}, \ldots, b_1$. Имеем $C(b_1,1) \leqslant n - C(b_k,k) - \ldots - C(b_2,2) < C(b_1+1,1)$. Откуда

$$n = C(b_1,1) + C(b_2,2) + \ldots + C(b_k,k).$$

[Единственность] Индукция по k. База. При $k=1$ имеем $C(n,1) = n$. Пусть для $k-1$ представление единственно и существует отличный от построенного набор натуральных чисел $0 \leqslant c_1 < c_2 < \cdots < c_k$ такой, что

$$n = C(c_1,1) + C(c_2,2) + \ldots + C(c_k,k).$$

В силу условия на b_k имеем $c_k \leqslant b_k$, причём равенство $c_k = b_k$ противоречит предположению индукции. Значит, $c_k \leqslant b_k - 1$, и имеем цепочку неравенств

$$n - C(c_k,k) \geqslant C(b_k,k) - C(c_k,k) \geqslant C(b_k,k) - C(b_k-1,k) = C(b_k-1,k-1) \geqslant C(c_k,k-1).$$

С другой стороны, в силу предположения индукции, $n - C(c_k,k) < C(c_{k-1}+1,k-1)$. Сравнивая два последних неравенства, приходим к противоречию $c_{k-1}+1 > c_k$. □

5.4. Разбиения

Разбиения не рассматривались среди типовых комбинаторных конфигураций в п. 5.1, потому что получить для них явную формулу не так просто, как для остальных. В этом разделе отмечаются основные свойства разбиений, а окончательные формулы приведены в п. 5.6.3.

5.4.1. Числа Стирлинга второго рода

Пусть $\mathcal{B} = \{B_1, \ldots, B_n\}$ — разбиение множества X из m элементов на n подмножеств: $B_i \subset X$, $\bigcup\limits_{i=1}^{n} B_i = X$, $B_i \neq \varnothing$, $B_i \cap B_j = \varnothing$ при $i \neq j$. Подмножества B_i называются *блоками разбиения*.

Между разбиениями с непустыми блоками и отношениями эквивалентности существует взаимно-однозначное соответствие (п. 1.7.1). Если E_1 и E_2 — два разбиения X, то говорят, что разбиение E_1 есть *измельчение разбиения* E_2, если каждый блок E_2 есть объединение блоков E_1. Измельчение является частичным порядком на множестве разбиений.

Число разбиений m-элементного множества на n блоков называется *числом Стирлинга*[1] *второго рода* и обозначается $S(m,n)$. По определению положим

$$S(m,m) \stackrel{\text{Def}}{=} 1, \quad S(m,0) \stackrel{\text{Def}}{=} 0 \quad \text{при } m > 0,$$
$$S(0,0) \stackrel{\text{Def}}{=} 1, \quad S(m,n) \stackrel{\text{Def}}{=} 0 \quad \text{при } n > m.$$

ТЕОРЕМА 1. $S(m,n) = S(m-1,n-1) + nS(m-1,n)$.

[1] Джеймс Стирлинг (1699–1770).

Доказательство. Пусть $\mathcal{B}$ — множество всех разбиений множества $M:=\{1,\ldots,m\}$ на n блоков. Положим

$$\mathcal{B}_1:=\{X\in\mathcal{B}\mid\exists\,B\in X\ \ (B=\{m\})\},\quad \mathcal{B}_2:=\{X\in\mathcal{B}\mid\neg\exists\,B\in X\ \ (B=\{m\})\},$$

то есть в $\mathcal{B}_1$ входят разбиения, в которых элемент m образует отдельный блок, а в $\mathcal{B}_2$ — все остальные разбиения. Заметим, что $\mathcal{B}_2=\{X\in\mathcal{B}\mid m\in X\Longrightarrow|X|>1\}$. Тогда $\mathcal{B}=\mathcal{B}_1\cup\mathcal{B}_2$, $\mathcal{B}_1\cap\mathcal{B}_2=\varnothing$. Имеем $|\mathcal{B}_1|=S(m-1,n-1)$, $|\mathcal{B}_2|=n\,S(m-1,n)$, так как все разбиения $\mathcal{B}_2$ получаются следующим образом: берём все разбиения множества $\{1,\ldots,m-1\}$ на n блоков (их $S(m-1,n)$) и в каждый блок по очереди помещаем элемент m. Следовательно, так как $\mathcal{B}_1\cap\mathcal{B}_2=\varnothing$,
$S(m,n)=|\mathcal{B}|=|\mathcal{B}_1|+|\mathcal{B}_2|=S(m-1,n-1)+nS(m-1,n)$. □

Теорема 2. $S(m,n)=\sum\limits_{i=n-1}^{m-1}C(m-1,i)S(i,n-1)$.

Доказательство. Пусть $\mathcal{B}$ — множество всех разбиений множества $M:=\{1,\ldots,m\}$ на n блоков. Рассмотрим семейство $\overline{B}:=\bigl\{B\subset 2^M\mid m\in B\bigr\}$. Тогда $\mathcal{B}=\bigcup_{B\in\overline{B}}\mathcal{B}_B$, где $\mathcal{B}_B:=\{X\mid X\in\mathcal{B}\ \&\ B\in X\}$, причём $\mathcal{B}_{B'}\cap\mathcal{B}_{B''}=\varnothing$, если $B'\neq B''$. Пусть $B\in\overline{B}$ и $b:=|B|$. Тогда $|\mathcal{B}_B|=S(m-b,n-1)$. Заметим, что $\bigl|\bigl\{B\in\overline{B}\mid b=|B|\bigr\}\bigr|=$
$=C(m-1,b-1)$. Имеем

$$\begin{aligned}S(m,n)=|\mathcal{B}|&=\sum_{b=1}^{m-(n-1)}\left|\bigcup_{B\in\overline{B}\ \&\ |B|=b}\mathcal{B}_B\right|=\\&=\sum_{b=1}^{m-(n-1)}C(m-1,b-1)S(m-b,n-1)=\\&=\sum_{i=m-1}^{n-1}C(m-1,m-i-1)S(i,n-1)=\sum_{i=n-1}^{m-1}C(m-1,i)S(i,n-1),\end{aligned}$$

где $i:=m-b$. □

5.4.2. Числа Стирлинга первого рода

Число сюръективных функций, то есть число размещений m мячей по n ящикам, таких, что все ящики заняты, называется *числом Стирлинга первого рода* и обозначается $s(m,n)$.

Теорема. $s(m,n)=n!\,S(m,n)$.

Доказательство. Каждое разбиение множества $\{1,\ldots,m\}$ соответствует семейству множеств уровня сюръективной функции и обратно (п. 1.7.4). Таким образом, число различных семейств множеств уровня сюръективных функций — это число Стирлинга второго рода $S(m,n)$. Всего сюръективных функций $s(m,n)=n!\,S(m,n)$, так как число сюръективных функций с заданным семейством множеств уровня равно числу перестановок множества значений функции. □

ЗАМЕЧАНИЕ

Число Стирлинга первого рода $s(m, n)$ можно определить как число перестановок длины m с n циклами. Нетрудно показать, что такое определение эквивалентно данному выше.

5.4.3. Вычисление чисел Стирлинга

В п. 5.4.1 выведены две формулы для вычисления чисел Стирлинга второго рода. Вторая формула прямая, но требует вспомогательного вычисления биномиальных коэффициентов и выполнения большого количества сложений и умножений. При использовании первой рекуррентной формулы встаёт вопрос о хранении уже посчитанных чисел Стирлинга, необходимых для промежуточных вычислений, а также многократного перевычисления одних и тех же чисел в случае прямого использования рекурсии.

Пример. Вычислим $S(7, 3)$. Для этого построим таблицу начальных значений чисел Стирлинга, пользуясь рекуррентной формулой.

$m \setminus n$	0	1	2	3	4	5	6	7
0	1							
1	0	1						
2	0	1	1					
3	0	1	3	1				
4	0	1	7	6	1			
5	0	1	15	25	10	1		
6	0	1	31	90	65	15	1	
7	0	1	63	301	350	140	21	1

Значения, которые необходимы для вычисления числа $S(7, 3)$, обведены прямоугольниками.

Алгоритм 5.6. Алгоритм вычисления чисел Стирлинга второго рода

Вход: числа m и n.
Выход: число Стирлинга $S(m, n)$.

```
D : array[1..n] of integer  // рабочая диагональ
if m = n then return 1  // можем сразу вернуть известные значения
if m > 0 & n = 0 then return 0
if n > m then return 0
if n = 1 then return 1
for i from 1 to n do D[i] := 1 end for  // инициализация
for i from 1 to m − n  do
  for  j from 2 to n  do
    D[j] := D[j − 1] + j ∗ D[j]  // перевычисление диагонали
  end for
end for
return D[n]
```

Обоснование. Хранить достаточно лишь одну «диагональ» длины n. Вначале диагональ инициализируется значениями 1. Далее на каждом шаге цикла по i вычисляется диагональ, начинающаяся с i-й строки. Таким образом, вычисляются только необходимые промежуточные элементы, причём по одному разу. □

ЗАМЕЧАНИЕ

При выборе структуры данных для хранения промежуточных значений очень важно обратить внимание на то, что числа Стирлинга второго рода для фиксированного m с ростом n сначала возрастают, а потом убывают.

5.4.4. Число Белла

Число всех разбиений m-элементного множества называется *числом Белла*[1] и обозначается $B(m)$, $B(m) \stackrel{\text{Def}}{=} \sum\limits_{n=0}^{m} S(m,n)$, $\quad B(0) \stackrel{\text{Def}}{=} 1$.

Теорема. $B(m+1) = \sum\limits_{i=0}^{m} C(m,i)B(i)$.

Доказательство. Пусть $\mathcal{B}$ — множество всех разбиений множества $M_1 = 1..(m+1)$. Рассмотрим множество подмножеств множества M_1, содержащих элемент $m+1$: $\overline{B} := \{ B \subset 2^{M_1} \,|\, m+1 \in B \}$. Тогда $\mathcal{B} = \bigcup\limits_{B \in \overline{B}} \mathcal{B}_B$, где $\mathcal{B}_B := \{X \in \mathcal{B} \mid B \in X\}$. Пусть $B \in \overline{B}$ и $b = |B|$. Тогда $|\mathcal{B}_B| = B(m+1-b)$. Заметим, что $|\{B \in \overline{B} \mid |B| = b\}| = C(m, b-1)$. Следовательно,

$$B(m+1) = |\mathcal{B}| = \sum_{b=1}^{m+1} C(m,b-1)B(m-b+1) =$$

$$= \sum_{i=m}^{0} C(m,m-i)B(i) = \sum_{i=0}^{m} C(m,i)B(i),$$

где $i := m-b+1$. □

5.4.5. Треугольник Белла

Вычисление числа Белла по рекуррентной формуле, выведенной в предыдущем параграфе, даже при условии хранения всех предыдущих вычисленных значений чисел Белла, имеет трудоёмкость $O(n^3)$. Рассмотрим алгоритм, использующий специальную структуру данных, который в неявном виде хранит все предыдущие вычисленные значения для биномиальных коэффициентов и имеет трудоёмкость $O(n^2)$.

Треугольник Белла — это бесконечная треугольная матрица, где каждое число (кроме чисел на левой боковой стороне) является суммой двух чисел: стоящего слева от него в этой же строке и слева над ним. Первое число в каждой строке (кроме первой) является последним числом из предыдущей строки. Число в первой строке — единица.

[1] Эрик Темпл Белл (1883–1960).

$$
\begin{array}{ccccccccccccc}
 & & & & & & 1 & & & & & & \\
 & & & & 1 & & & & 2 & & & & \\
 & & & 2 & & & 3 & & & & 5 & & \\
 & & 5 & & & 7 & & & 10 & & & 15 & \\
 & 15 & & & 20 & & & 27 & & & 37 & & & 52 \\
. & & . & & . & & . & & . & & . & & .
\end{array}
$$

Для вычислений треугольник Белла удобнее хранить в виде бесконечной нижней треугольной матрицы:

$$
\begin{array}{cccccc}
1 & & & & & \\
1 & 2 & & & & \\
2 & 3 & 5 & & & \\
5 & 7 & 10 & 15 & & \\
15 & 20 & 27 & 37 & 52 & \\
. & . & . & . & . & .
\end{array}
$$

Обозначим $A_{n,k}$ — k-й элемент в n-й строке треугольника Белла. По построению треугольника $A_{n,k} = A_{n,k-1} + A_{n-1,k-1}$ при $k > 1, n > 1$ и $A_{n,1} = A_{n-1,n-1}$.

ЛЕММА. $A_{n,k} = \sum\limits_{i=n-k+1}^{n} C(k-1, i-n+k-1)A_{i,1}$.

Доказательство. Индукция по k. База: $A_{n,1} = \sum\limits_{i=n}^{n} C(0, i-n)A_{i,1} = C(0,0)A_{n,1} = A_{n,1}$.
Индукционный переход:

$$
\begin{aligned}
A_{n,k} &= A_{n,k-1} + A_{n-1,k-1} = \\
&= \sum_{i=n-k+2}^{n} C(k-2, i-n+k-2)A_{i,1} + \sum_{i=n-k+1}^{n-1} C(k-2, i-n+k-1)A_{i,1} = \\
&= \sum_{i=n-k+2}^{n-1} \left(C(k-2, i-n+k-2) + C(k-2, i-n+k-1)\right) A_{i,1} + \\
&\qquad + C(k-2, k-2)A_{n,1} + C(k-2, 0)A_{n-k+1,1} = \\
&= \sum_{i=n-k+2}^{n-1} C(k-1, i-n+k-1)A_{i,1} + C(k-1, k-1)A_{n,1} + C(k-1, 0)A_{n-k+1,1} = \\
&= \sum_{i=n-k+1}^{n} C(k-1, i-n+k-1)A_{i,1}.
\end{aligned}
$$

□

ТЕОРЕМА. $B(m) = A_{m+1,1} = A_{m,m}$.

Доказательство. Индукция по m. База $B(0) = 1 = A_{1,1}$ по построению треугольника. Пусть $B(m) = A_{m+1,1}$. Тогда по лемме $A_{m+2,1} = A_{m+1,m+1} =$
$= \sum\limits_{i=0}^{m} C(m,i)A_{i+1,1} = \sum\limits_{i=0}^{m} C(m,i)B(i) = B(i+1)$. □

Из теоремы непосредственно следует алгоритм 5.7.

Алгоритм 5.7. Вычисление чисел Белла

Вход: число m.
Выход: число Белла $B(m)$.

```
A : array [1..m] of integer  // рабочий массив, A[k] хранит A_{n,k}
A[1] := 1  // A_{1,1} = B(0) = 1
for n from 2 to m do
  t := A[1]  // переменная t хранит A_{n-1,k-1}
  A[1] := A[n-1]  // A_{n,1} = A_{n-1,n-1}
  for k from 2 to n do
    s := A[k]  // переменная s хранит A_{n-1,k}
    A[k] := A[k-1] + t  // A_{n,k} = A_{n,k-1} + A_{n-1,k-1}
    t := s  // k := k + 1
  end for
end for
return A[m]
```

5.5. Включения и исключения

Приведённые в предыдущих четырёх разделах формулы и алгоритмы дают способы вычисления комбинаторных чисел для некоторых распространённых комбинаторных конфигураций. Практические задачи не всегда прямо сводятся к известным комбинаторным конфигурациям. В этом случае используются различные методы сведéния одних комбинаторных конфигураций к другим. В этом и двух следующих разделах рассматриваются три наиболее часто используемых метода. Мы начинаем с самого простого и прямолинейного, но имеющего ограниченную область применения *метода включений и исключений*.

5.5.1. Объединение конфигураций

Часто комбинаторная конфигурация является объединением других, число комбинаций в которых вычислить проще. В таком случае требуется уметь вычислять число комбинаций в объединении. В простых случаях формулы для вычисления очевидны:

$$|A \cup B| = |A| + |B| - |A \cap B|,$$
$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|.$$

Пример. Сколько существует натуральных чисел, меньших 1000, которые не делятся ни на 3, ни на 5, ни на 7? Всего чисел, меньших тысячи, 999. Из них:

$999 : 3 = 333$ делятся на 3,
$999 : 5 = 199$ делятся на 5,
$999 : 7 = 142$ делятся на 7,
$999 : (3 * 5) = 66$ делятся на 3 и на 5,
$999 : (3 * 7) = 47$ делятся на 3 и на 7,
$999 : (5 * 7) = 28$ делятся на 5 и на 7,

$999 : (3 * 5 * 7) = 9$ делятся на 3, на 5 и на 7.

Имеем: $999 - (333 + 199 + 142 - 66 - 47 - 28 + 9) = 457$.

5.5.2. Формула включений и исключений

Следующая формула, известная как *формула включений и исключений*, позволяет вычислить мощность объединения нескольких множеств, если известны их мощности и мощности всех возможных пересечений.

ТЕОРЕМА.

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1\leqslant i<j\leqslant n} |A_i \cap A_j| + \sum_{1\leqslant i<j<k\leqslant n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1}|A_1 \cap \ldots \cap A_n|.$$

Доказательство. Индукция по n. Для $n = 2, 3$ теорема проверена в предыдущем параграфе. Пусть

$$\left|\bigcup_{i=1}^{n-1} A_i\right| = \sum_{i=1}^{n-1} |A_i| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \ldots + (-1)^{n-2}|A_i \cap \ldots \cap A_{n-1}|.$$

Заметим, что $\left(\bigcup\limits_{i=1}^{n-1} A_i\right) \cap A_n = \bigcup\limits_{i=1}^{n-1} (A_i \cap A_n)$, и по индукционному предположению

$$\left|\bigcup_{i=1}^{n-1} (A_i \cap A_n)\right| = \sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j \cap A_n| + \cdots + (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1} \cap A_n|.$$

Тогда

$$\left|\bigcup_{i=1}^{n} A_i\right| = \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cup A_n\right| = \left|\bigcup_{i=1}^{n-1} A_i\right| + |A_n| - \left|\left(\bigcup_{i=1}^{n-1} A_i\right) \cap A_n\right| =$$

$$= \left(\sum_{i=1}^{n-1} |A_i| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \ldots + (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1}|\right) +$$

$$+ |A_n| - \left(\sum_{i=1}^{n-1} |A_i \cap A_n| - \sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j \cap A_n| + \ldots + (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1} \cap A_n|\right) =$$

$$= \left(\sum_{i=1}^{n-1} |A_i| + |A_n|\right) - \left(\sum_{1\leqslant i<j\leqslant n-1} |A_i \cap A_j| + \sum_{i=1}^{n-1} |A_i \cap A_n|\right) + \ldots -$$

$$- (-1)^{n-2}|A_1 \cap \ldots \cap A_{n-1} \cap A_n| =$$

$$= \sum_{i=1}^{n} |A_i| - \sum_{1 \leqslant i < j \leqslant n} |A_i \cap A_j| + \ldots + (-1)^{n-1} |A_1 \cap \ldots \cap A_n|. \qquad \square$$

ЗАМЕЧАНИЕ

Обозначения сумм с неравенствами в пределах суммирования, использованные в формулировке и доказательстве теоремы, являются не более чем сокращённой формой записи кратных сумм. Например, $\sum\limits_{1 \leqslant i < j \leqslant n}$ означает $\sum\limits_{i=1}^{n-1} \sum\limits_{j=i+1}^{n}$.

Пример. Докажем формулу для функции Эйлера $\varphi(n)$ (п. 2.4.9) с помощью метода включений и исключений. Пусть $n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ и $\varphi(n)$ — количество чисел, меньших n и взаимно простых с ним. Пусть A_i — множество чисел, не превосходящих n и делящихся на p_i. Тогда $\varphi(n) = n - |\bigcup\limits_{i=1}^{k} A_i|$. Заметим, что

$$|A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k}| = \frac{n}{p_{i_1} p_{i_2} \ldots p_{i_k}},$$

поскольку это множество чисел, не превосходящих n, делящихся на $p_{i_1} p_{i_2} \ldots p_{i_k}$. Отсюда получаем

$$\varphi(n) = n - (\frac{n}{p_1} + \frac{n}{p_2} \ldots \frac{n}{p_k}) + (\frac{n}{p_1 p_2} + \frac{n}{p_2 p_3} \ldots \frac{n}{p_k p_{k-1}}) - \ldots + (-1)^k \frac{n}{p_1 p_2 \ldots p_k} =$$
$$= n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2}) \ldots (1 - \frac{1}{p_k}),$$

что согласуется с формулой, выведенной в п. 2.4.9.

5.5.3. Число булевых функций, существенно зависящих от всех своих переменных

Рассмотрим применение формулы включений и исключений на примере следующей задачи. Пусть $p_n \stackrel{\text{Def}}{=} |P_n| = 2^{2^n}$ — число всех булевых функций n переменных, а $\widetilde{p}_n$ — число булевых функций, существенно зависящих от всех n переменных (п. 3.1.2). Пусть P_n^i — множество булевых функций, у которых переменная x_i фиктивная (кроме x_i могут быть и другие фиктивные переменные). Имеем

$$\widetilde{p}_n = |P_n \setminus (P_n^1 \cup \ldots \cup P_n^n)| = \left| P_n \setminus \bigcup_{i=1}^{n} P_n^i \right|.$$

С другой стороны, $|P_n^i| = 2^{2^{n-1}}$, более того, $|P_n^{i_1} \cap \ldots \cap P_n^{i_k}| = 2^{2^{n-k}}$. Следовательно,

$$\widetilde{p}_n = 2^{2^n} - \left(\sum_{i=1}^{n} |P_n^i| - \sum_{1 \leqslant i < j \leqslant n} |P_n^i \cap P_n^j| + \ldots + (-1)^{n-1} |P_n^1 \cap \ldots \cap P_n^n| \right) =$$
$$= 2^{2^n} - \left(C(n,1) 2^{2^{n-1}} - C(n,2) 2^{2^{n-2}} + \ldots + (-1)^{n-1} C(n,n) 2 \right) =$$
$$= \sum_{i=0}^{n} (-1)^i C(n,i) 2^{2^{n-i}}.$$

5.5.4. Комбинации свойств

Достаточно часто комбинаторная задача формулируется как задача подсчёта количества объектов, обладающих или не обладающих определёнными *свойствами*.

Уточним постановку задачи. Пусть имеется множество объектов M, $|M| = m$, и множество *свойств* $a_1, \ldots, a_n$, то есть одноместных предикатов $a_i\colon M \to 0..1$. Обозначим $N(a_{i_1}, \ldots, a_{i_s})$ — количество элементов, обладающих свойствами $a_{i_1}, \ldots, a_{i_s}$, $N(\overline{a}_{j_1}, \ldots, \overline{a}_{j_t})$ — количество элементов, не обладающих свойствами $a_{j_1}, \ldots, a_{j_t}$.

ТЕОРЕМА. $N(\overline{a}_{j_1}, \ldots, \overline{a}_{j_t}) = m - \sum\limits_{i=1}^{n} N(a_i) + \sum\limits_{1 \leqslant i < j \leqslant n} N(a_i, a_j) + \ldots + (-1)^n N(a_1, \ldots, a_n)$.

Доказательство. Каждый одноместный предикат является характеристическим предикатом подмножества $M_i := \{x \in M \mid a_i(x)\}$. Поэтому

$$N(\overline{a}_{j_1}, \ldots, \overline{a}_{j_t}) = \left| \bigcap_{i=1}^{n} \overline{M_i} \right| = |M| - \left| \bigcup_{i=1}^{n} M_i \right|,$$

и далее по формуле включений и исключений. □

Пример. Симметричные пин-коды $xyyx$ и периодические пин-коды $xyxy$ считаются ненадёжными. Сколько существует надёжных пин-кодов?
Ответ: $10000 - (100 + 100) + 10 = 9810$.

5.5.5. Беспорядки и субфакториал

Перестановка без неподвижных точек называется *беспорядком*. Количество всех беспорядков n предметов называется *субфакториалом* числа n и обозначается $!n$.

ТЕОРЕМА 1. $!n = n! \sum\limits_{k=0}^{n} \dfrac{(-1)^k}{k!}$.

Доказательство. Обозначим через $D(n, k)$ — количество перестановок из n элементов, в которых k элементов — неподвижные точки. Существует $P(n-k)$ перестановок, в которых k элементов остаются на своей позиции. Выбрать k неподвижных элементов из n можно $C(n, k)$ способами. Окончательно получаем $D(n, k) = C(n, k)P(n-k)$. Общее количество перестановок равняется $n!$. Воспользуемся формулой включений и исключений. Имеем:

$$\begin{aligned}
!n &= n! - C(n,1)P(n-1) + C(n,2)P(n-2) - \ldots \\
&\quad + (-1)^k C(n,k)P(n-k) - \ldots + (-1)^n C(n,n) = \\
&= n! - \frac{n!(n-1)!}{(n-1)!} + \frac{n!(n-2)!}{(n-2)!2!} - \ldots + (-1)^k \frac{n!(n-k)!}{(n-k)!k!} - \cdots + (-1)^n \frac{n!}{0!} = \\
&= \sum_{k=0}^{n} (-1)^k \frac{n!}{k!}.
\end{aligned}$$

□

ЗАМЕЧАНИЕ

Представим субфакториал, как $!n = n! \sum\limits_{k=0}^{n} \frac{(-1)^k}{k!}$. Тогда можно заметить, что сумма в правой части является частичной суммой разложения $e^{-1} = 1 - 1/1! + 1/2! - \ldots + (-1)^n/n! + \ldots$. Поэтому субфакториал является ближайшим целым числом к числу $n!/e$.

ТЕОРЕМА 2. $!n = (n-1)(!(n-1)+!(n-2))$.

Доказательство. $(n-1)\,(!(n-1)+!(n-2)) =$

$$= (n-1)\left((n-1)!\sum_{k=0}^{n-1}\frac{(-1)^k}{k!} + (n-2)!\sum_{k=0}^{n-2}\frac{(-1)^k}{k!}\right) =$$

$$= (n-1)(n-2)!\left((n-1)\sum_{k=0}^{n-1}\frac{(-1)^k}{k!} + \sum_{k=0}^{n-2}\frac{(-1)^k}{k!}\right) =$$

$$= (n-1)!\left(n\sum_{k=0}^{n-1}\frac{(-1)^k}{k!} - \sum_{k=0}^{n-1}\frac{(-1)^k}{k!} + \sum_{k=0}^{n-2}\frac{(-1)^k}{k!}\right) =$$

$$= (n-1)!\left(n\sum_{k=0}^{n-1}\frac{(-1)^k}{k!} - \frac{(-1)^{n-1}}{(n-1)!}\right) = (n-1)!\left(n\sum_{k=0}^{n-1}\frac{(-1)^k}{k!} + \frac{n(-1)^n}{n(n-1)!}\right) =$$

$$= n(n-1)!\sum_{k=0}^{n}\frac{(-1)^k}{k!} = !n.$$

□

СЛЕДСТВИЕ. $!n = !(n-1)\cdot n + (-1)^n$.

Доказательство. По индукции. База: $!1 = 0$.
Пусть $!(n-1) = !(n-2)\cdot(n-1) + (-1)^{n-1}$.
Тогда $!n = (n-1)\,(!(n-1)+!(n-2)) = !(n-1)\cdot n - !(n-1) + !(n-2)\cdot n - !(n-2) =$
$= !(n-1)\cdot n - (!(n-2)\cdot(n-1) + (-1)^{n-1}) + !(n-2)\cdot n - !(n-2) =$
$= !(n-1)\cdot n - !(n-2)\cdot n + !(n-2) + (-1)^n + !(n-2)\cdot n - !(n-2) =$
$= !(n-1)\cdot n + (-1)^n$.

□

Пример. Профессор собирается поручить четырем студентам (назовем их A, B, C, D) проверить друг у друга контрольную работу. Ни один студент не должен проверять свою работу. Сколько у профессора вариантов раздать работы на проверку? Из всех $P(4) = 4! = 24$ перестановок, подходят только $!4 = 9$ штук:

$$BADC, BCDA, BDAC, CADB, CDAB, CDBA, DABC, DCAB, DCBA.$$

Любая другая перестановка будет иметь как минимум одну неподвижную точку.

Субфакториал достаточно быстро растущая функция: $!1 = 0$; $!2 = 1$; $!3 = 2$; $!4 = 9$; $!5 = 44$; $!6 = 265$; $!7 = 1854$; $!8 = 14833$; $!9 = 133496$.

5.6. Формулы обращения

Полезную, но очень специфическую группу приёмов образуют различные способы преобразования уже полученных комбинаторных выражений. В этом разделе рассматривается один частный, но важный случай.

5.6.1. Теорема обращения

Пусть $a_{n,k}$ и $b_{n,k}$ — некоторые (комбинаторные) числа, зависящие от параметров n и k, причём $0 \leqslant k \leqslant n$. Если известно выражение чисел $a_{n,k}$ через числа $b_{n,k}$, то в некоторых случаях можно найти и выражение чисел $b_{n,k}$ через числа $a_{n,k}$, то есть решить комбинаторное уравнение.

ТЕОРЕМА. *Пусть* $\forall\, n \left(\forall\, k \leqslant n \left(a_{n,k} = \sum\limits_{i=0}^{n} \lambda_{n,k,i} b_{n,i}\right)\right)$ *и пусть*
$\exists\, \mu_{n,k,i} \left(\forall\, k \leqslant n \left(\forall\, m \leqslant n \left(\sum\limits_{i=0}^{n} \mu_{n,k,i}\lambda_{n,i,m} = \begin{cases}1, & m = k,\\ 0, & m \neq k\end{cases}\right)\right)\right)$.
Тогда $\forall\, k \leqslant n \ \left(b_{n,k} = \sum_{i=0}^{n} \mu_{n,k,i} a_{n,i}\right)$.

Доказательство. $\sum\limits_{i=0}^{n} \mu_{n,k,i} a_{n,i} = \sum\limits_{i=0}^{n} \mu_{n,k,i} \left(\sum\limits_{m=0}^{n} \lambda_{n,i,m} b_{n,m}\right) =$
$= \sum\limits_{m=0}^{n} \left(\sum\limits_{i=0}^{n} \mu_{n,k,i}\lambda_{n,i,m}\right) b_{n,m} = b_{n,k}.$ □

5.6.2. Формулы обращения для биномиальных коэффициентов

Применение теоремы обращения предполагает отыскание для заданных чисел $\lambda_{n,k,i}$ (коэффициентов комбинаторного уравнения) соответствующих чисел $\mu_{n,k,i}$, удовлетворяющих условию теоремы обращения. Особенно часто числами $\lambda_{n,k,i}$ являются биномиальные коэффициенты.

ЛЕММА. $\sum\limits_{i=0}^{n} (-1)^{i-m} C(n,i) C(i,m) = \begin{cases}1, & m = n,\\ 0, & m < n.\end{cases}$

Доказательство. Используя формулу 3 из п. 5.3.1 и тот факт, что $C(n-m, i-m) = 0$ при $i \leqslant m$, имеем

$$\sum_{i=0}^{n} (-1)^{i-m} C(n,i) C(i,m) = \sum_{i=0}^{n} (-1)^{i-m} C(n,m) C(n-m, i-m) =$$
$$= \sum_{i=m}^{n} (-1)^{i-m} C(n,m) C(n-m, i-m) = C(n,m) \sum_{i=m}^{n} (-1)^{i-m} C(n-m, i-m).$$

Но при $m < n$ имеем $\sum\limits_{i=m}^{n} (-1)^{i-m} C(n-m, i-m) = \sum\limits_{j=0}^{n-m} (-1)^j C(n-m, j) = 0$, где $j := i - m$. С другой стороны, при $m = n$ имеем
$C(n,m) \sum\limits_{i=m}^{n} (-1)^{i-m} C(n-m, i-m) = C(n,n)(-1)^{n-n} C(0,0) = 1.$ □

ТЕОРЕМА 1. *Если* $a_{n,k} = \sum\limits_{i=0}^{k} C(k,i) b_{n,i}$, *то* $b_{n,k} = \sum\limits_{i=0}^{k} (-1)^{k-i} C(k,i) a_{n,i}$.

Доказательство. Здесь $\lambda_{n,k,i} = C(k,i)$ и $\mu_{n,k,i} = (-1)^{k-i}C(k,i)$. При $k \leqslant n$, $m \leqslant n$ имеем: $\sum\limits_{i=0}^{n} \mu_{n,k,i}\lambda_{n,i,m} = \sum\limits_{i=0}^{n}(-1)^{k-i}C(k,i)C(i,m) =$ (так как $C(k,i) = 0$ при $i > k$) $= \sum\limits_{i=0}^{k}(-1)^{k-i+m-m}C(k,i)C(i,m) = (-1)^{k-m}\sum\limits_{i=0}^{k}(-1)^{i-m}C(k,i)C(i,m) =$ $=$ **if** $k = m$ **then** 1 **else** 0 **end if**. □

ТЕОРЕМА 2. *Если* $a_{n,k} = \sum\limits_{i=k}^{n} C(i,k)b_{n,i}$, *то* $b_{n,k} = \sum\limits_{i=k}^{n}(-1)^{i-k}C(i,k)a_{n,i}$.

Доказательство. Здесь $\lambda_{n,k,i} = C(i,k)$ и $\mu_{n,k,i} = (-1)^{i-k}C(i,k)$. При $k \leqslant n$, $m \leqslant n$ имеем: $\sum\limits_{i=0}^{n} \mu_{n,k,i}\lambda_{n,i,m} = \sum\limits_{i=0}^{n}(-1)^{i-k}C(i,k)C(m,i) =$ $= \sum\limits_{i=0}^{n}(-1)^{i-k}C(m,i)C(i,k) =$ **if** $k = n$ **then** 1 **else** 0 **end if**. □

5.6.3. Формулы для чисел Стирлинга

В качестве примера использования формул обращения рассмотрим получение явных формул для чисел Стирлинга первого и второго рода. Рассмотрим множество функций $f\colon A \to B$, где $|A| = n$ и $|B| = k$. Число всех таких функций равно k^n. С другой стороны, число функций f, таких, что $|f(A)| = i$, равно $s(n,i)$, поскольку $s(n,i)$ — это число сюръективных функций $f\colon 1..n \to 1..i$. Но множество значений функции (при заданном i) можно выбрать $C(k,i)$ способами. Поэтому

$$k^n = \sum_{i=0}^{k} C(k,i)s(n,i).$$

Обозначив $a_{n,k} := k^n$ и $b_{n,i} := s(n,i)$, имеем по теореме 1 предыдущего параграфа

$$s(n,k) = \sum_{i=0}^{k}(-1)^{k-i}C(k,i)i^n.$$

Учитывая связь чисел Стирлинга первого и второго рода, окончательно имеем

$$S(n,k) = \frac{1}{n!}\sum_{i=0}^{k}(-1)^{k-i}C(k,i)i^n.$$

5.7. Производящие функции

Для решения комбинаторных задач в некоторых случаях можно использовать методы математического анализа. Разнообразие применяемых здесь приёмов весьма велико и не может быть в полном объёме рассмотрено в рамках этой книги. В данном разделе рассматривается только основная идея метода производящих функций, применение которой иллюстрируется несколькими простыми примерами. Более детальное рассмотрение можно найти в литературе, например в [7].

5.7.1. Формальные степенные ряды

Пусть есть последовательность комбинаторных чисел a_i и последовательность функций $\varphi_i(x)$. Рассмотрим формальный ряд $\mathcal{F}(x) \stackrel{\text{Def}}{=} \sum_i a_i\varphi_i(x)$. Выражение $\mathcal{F}(x)$ называется *производящей функцией* (для заданной последовательности комбинаторных чисел a_i относительно заданной последовательности функций $\varphi_i(x)$).

Обычно используют $\varphi_i(x) \stackrel{\text{Def}}{=} x^i$ или $\varphi_i(x) \stackrel{\text{Def}}{=} x^i/i!$.

Пример. Из формулы бинома Ньютона при $y = 1$ имеем

$$(1+x)^n = \sum_{i=0}^{n} C(n,i)x^i.$$

Таким образом, $(1 + x)^n$ является производящей функцией для биномиальных коэффициентов.

ЛЕММА. $V(m,n) = V(m-1,n) + V(m,n-1)$.

Доказательство. $V(m,n) = C(n+m-1,n) = C(n+m-2,n) + C(n+m-2,n-1) = V(m-1,n) + V(m,n-1)$. □

Пример. Производящая функция для $V(m,n)$ при заданном m.

$$\begin{aligned}\varphi_m(x) &= \sum_{n=0}^{\infty} V(m,n)x^n = 1 + \sum_{n=1}^{\infty} V(m,n)x^n = \\ &= 1 + \sum_{n=1}^{\infty} V(m-1,n)x^n + \sum_{n=1}^{\infty} V(m,n-1)x^n = \\ &= 1 + \sum_{n=1}^{\infty} V(m-1,n)x^n + x\sum_{n=1}^{\infty} V(m,n-1)x^{n-1} = \\ &= \sum_{n=0}^{\infty} V(m-1,n)x^n + x\sum_{n=0}^{\infty} V(m,n)x^n = \varphi_{m-1}(x) + x\varphi_m(x).\end{aligned}$$

Получили, что $\varphi_m(x) = 1/(1-x)\varphi_{m-1}(x) = 1/(1-x)^{-2}\varphi_{m-2}(x) = \ldots$
$= 1/(1-x)^m\varphi_0(x) = 1/(1-x)^m$, так как

$$\varphi_0(x) = \sum_{n=0}^{\infty} V(0,n)x^n = V(0,0) + \sum_{n=1}^{\infty} V(0,n)x^n = 1 + 0 = 1.$$

В итоге, $\varphi_m(x) = 1/(1-x)^m$.

5.7.2. Метод неопределённых коэффициентов

Из математического анализа известно, что если

$$\mathcal{F}(x) = \sum_i a_i\varphi_i(x) \quad \text{и} \quad \mathcal{F}(x) = \sum_i b_i\varphi_i(x),$$

то $\forall\, i \;\; (a_i = b_i)$ (для рассматриваемых здесь систем функций φ_i).

В качестве примера применения производящих функций докажем следующее тождество.

ТЕОРЕМА. $C(2n,n)=\sum\limits_{k=0}^{n}C(n,k)^2$.

Доказательство. Имеем: $(1+x)^{2n}=(1+x)^n(1+x)^n$. Следовательно, $\sum\limits_{i=0}^{2n}C(2n,i)x^i = \sum\limits_{i=0}^{n}C(n,i)x^i\cdot\sum\limits_{i=0}^{n}C(n,i)x^i$. Приравняем коэффициент при x^n: $C(2n,n)=\sum\limits_{k=0}^{n}C(n,k)C(n,n-k)=\sum\limits_{k=0}^{n}C(n,k)^2$. □

5.7.3. Числа Фибоначчи

Числа Фибоначчи[1] $F(n)$ определяются следующим образом:

$$F(0)\stackrel{\text{Def}}{=}1,\ F(1)\stackrel{\text{Def}}{=}1,\ F(n+2)\stackrel{\text{Def}}{=}F(n+1)+F(n).$$

Найдём выражение для $F(n)$ через n. Для этого найдём производящую функцию $\varphi(x)$ для последовательности чисел $F(n)$. Имеем

$$\begin{aligned}\varphi(x)&=\sum_{n=0}^{\infty}F(n)x^n=F(0)x^0+F(1)x^1+\sum_{n=2}^{\infty}\left(F(n-2)+F(n-1)\right)x^n=\\&=1+x+\sum_{n=2}^{\infty}F(n-2)x^n+\sum_{n=2}^{\infty}F(n-1)x^n=\\&=1+x+x^2\sum_{n=2}^{\infty}F(n-2)x^{n-2}+x\sum_{n=2}^{\infty}F(n-1)x^{n-1}=\\&=1+x+x^2\sum_{n=0}^{\infty}F(n)x^n+x\left(\sum_{n=0}^{\infty}F(n)x^n-F(0)\right)=\\&=1+x+x^2\varphi(x)+x\left(\varphi(x)-1\right).\end{aligned}$$

Решая это функциональное уравнение относительно $\varphi(x)$, получаем, что

$$\varphi(x)=\frac{1}{1-x-x^2}.$$

Последнее выражение нетрудно разложить в ряд по степеням x. Действительно, уравнение $1-x-x^2=0$ имеет корни $x_1=-(1+\sqrt{5})/2$ и $x_2=-(1-\sqrt{5})/2$, причём, как нетрудно убедиться,

$$1-x-x^2=(1-ax)(1-bx),\quad\text{где } a=\frac{1+\sqrt{5}}{2},\quad b=\frac{1-\sqrt{5}}{2}.$$

Далее,

$$\frac{1}{(1-ax)(1-bx)}=\frac{\alpha}{1-ax}+\frac{\beta}{1-bx},\quad\text{где }\alpha=\frac{a}{a-b},\quad\beta=\frac{-b}{a-b}.$$

[1] Фибоначчи (Ленардо Пизанский, 1180–1228).

Из математического анализа известно, что для малых x

$$\frac{1}{1-\gamma x} = \sum_{n=0}^{\infty} \gamma^n x^n.$$

Таким образом,

$$\varphi(x) = \frac{\alpha}{1-ax} + \frac{\beta}{1-bx} = \alpha \sum_{n=0}^{\infty} a^n x^n + \beta \sum_{n=0}^{\infty} b^n x^n =$$
$$= \sum_{n=0}^{\infty} \frac{a}{a-b} a^n x^n - \sum_{n=0}^{\infty} \frac{b}{a-b} b^n x^n = \sum_{n=0}^{\infty} \frac{a^{n+1} - b^{n+1}}{a-b} x^n.$$

Окончательно получаем соотношение, известное как *формула Бине*[1].

$$F(n) = \frac{a^{n+1} - b^{n+1}}{a-b} = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right).$$

Пример. Число $F_{n+2} = F_{n+1} + F_n$ является количеством двоичных последовательностей длины n, в которых нет двух 1, находящихся на соседних позициях. База: $n = 1$, $F_3 = 2$, последовательности 0 и 1; $n = 2$, $F_4 = 3$, последовательности 00, 01, 10. Индукционный переход: пусть F_{n+2} — число способов построить требуемые последовательности длины n. Построим последовательность длины $n+1$: возьмем все такие последовательности длины n и допишем к ним 0, не нарушая условие. Также мы можем взять все последовательности длины $n-1$ и дописать к ним 10. Таким образом, количество способов получить последовательность длины $n+1$ будет равно $F_{n+2} + F_{n+1} = F_{n+3}$.

5.7.4. Числа Каталана

Числа Каталана $C(n)$ можно определить следующим образом:

$$C(0) \stackrel{\text{Def}}{=} 1, \quad C(n) \stackrel{\text{Def}}{=} \sum_{k=0}^{n-1} C(k) C(n-k-1).$$

Числа Каталана используются при решении различных комбинаторных задач.

Пример. Пусть $\langle S, * \rangle$ — полугруппа (п. 3.2.2) и нужно вычислить выражение $s_1 * \cdots * s_n$. Сколькими способами это можно сделать? То есть сколькими способами можно расставить скобки, определяющие порядок вычисления выражения? Обозначим число способов $C(n)$. Ясно, что $C(0) = 1$. При любой схеме вычислений на каждом шаге выполняется некоторое вхождение операции $*$ над соседними элементами, и результат ставится на их место. Пусть последним выполняется то вхождение операции $*$, которое имеет номер k в исходном выражении. При этом слева от выбранного вхождения знака $*$ находилось и было выполнено $k-1$ знаков операции $*$, а справа $(n-1)-(k-1) = n-k$ знаков операции $*$. Тогда ясно, что $C(n) = \sum\limits_{k=1}^{n} C(k-1)C(n-k) = \sum\limits_{k=0}^{n-1} C(k)C(n-k-1)$, и ответом на поставленный вопрос является число Каталана $C(n)$.

[1] Жак Филипп Мари Бине (1786–1856).

Числа Каталана выражаются через биномиальные коэффициенты. Получим это выражение, используя метод производящих функций.

ТЕОРЕМА. $C(n) = \dfrac{C(2n,n)}{n+1}$.

Доказательство. Найдём производящую функцию для чисел Каталана:

$$\varphi(x) = \sum_{n=0}^{\infty} C(n)x^n.$$

Для этого рассмотрим квадрат этой функции:

$$\varphi^2(x) = \left(\sum_{n=0}^{\infty} C(n)x^n\right)^2 = \sum_{m=0}^{\infty} C(m)x^m \cdot \sum_{n=0}^{\infty} C(n)x^n =$$

$$= \sum_{m,n=0}^{\infty} C(m)C(n)x^{m+n} = \sum_{n=0}^{\infty}\sum_{k=0}^{n} C(k)C(n-k)x^n = \sum_{n=0}^{\infty} C(n+1)x^n =$$

$$= \sum_{n=0}^{\infty} C(n+1)\frac{x^{n+1}}{x} = \frac{1}{x}\left(\sum_{n=0}^{\infty} C(n)x^n - C(0)\right) = \frac{1}{x}\left(\varphi(x) - 1\right).$$

Решая уравнение $\varphi(x) = x\varphi^2(x) + 1$ относительно функции $\varphi(x)$, имеем

$$\varphi(x) = \frac{1 \pm \sqrt{1-4x}}{2x}.$$

Обозначим $f(x) := \sqrt{1-4x}$ и разложим $f(x)$ в ряд по формуле Тейлора:

$$f(x) = f(0) + \sum_{k=1}^{\infty} \frac{f^{(k)}(0)}{k!}x^k.$$

Имеем $\frac{d^k}{dx^k}(1-4x)^{\frac{1}{2}} = \frac{1}{2}\cdot\left(\frac{1}{2}-1\right)\cdot\dots\cdot\left(\frac{1}{2}-k+1\right)\cdot(1-4x)^{\frac{1}{2}-k}\cdot(-4)^k =$
$= -2^k\cdot 1\cdot 3\cdot\dots\cdot(2k-3)\cdot(1-4x)^{\frac{1}{2}-k} = -2^k\cdot(2k-3)!!\cdot(1-4x)^{\frac{1}{2}-k} =$
$= -2^k\frac{(2k-3)!}{2^{k-2}(k-2)!}(1-4x)^{\frac{1}{2}-k} = -2^2\frac{(2k-2)!(k-1)(k-1)!}{(2k-2)(k-1)!(k-1)!}(1-4x)^{\frac{1}{2}-k} =$
$= -2(k-1)!\frac{(2k-2)!}{(k-1)!(k-1)!}(1-4x)^{\frac{1}{2}-k} = -2(k-1)!C(2k-2,k-1)(1-4x)^{\frac{1}{2}-k}.$

Таким образом, $f^{(k)}(0) = -2(k-1)!C(2k-2,k-1)$ и
$f(x) = 1 - 2\sum\limits_{k=1}^{\infty}\frac{1}{k}C(2k-2,k-1)x^k$. Подставляя выражение $f(x)$ в формулу для $\varphi(x)$, следует выбрать знак «минус» перед корнем, чтобы удовлетворить условию $C(0) = 1$. Окончательно имеем

$$\sum_{n=0}^{\infty} C(n)x^n = \varphi(x) = \frac{1-f(x)}{2x} = \frac{1-1+2\sum_{n=1}^{\infty}\frac{1}{n}C(2n-2,n-1)x^n}{2x} =$$

$$= \sum_{n=1}^{\infty}\frac{1}{n}C(2(n-1),n-1)x^{n-1} = \sum_{n=0}^{\infty}\frac{C(2n,n)}{n+1}x^n,$$

и по методу неопределённых коэффициентов $C(n) = C(2n,n)/(n+1)$. □

В примере в начале параграфа показано, что число способов составить скобочную последовательность длины $2n$ равно $C(n)$, то есть числу Каталана. Числа Каталана проявляются во многих, часто неожиданных задачах.

Пример. Сколько существует монотонных путей в квадрате — маршрутов из левого нижнего угла квадрата $n \times n$ в правый верхний, которые идут по линиям сетки вверх или вправо и не заходят выше диагонали? Проведем биекцию с правильными скобочными последовательностями длины $2n$: открывающая скобка — горизонтальная линия, закрывающая скобка — вертикальная линия. Очевидно, что соответствующий правильной скобочной последовательности путь не пересечет диагональ. На рис. 5.5 представлены возможные монотонные пути для случая $n = 3$.

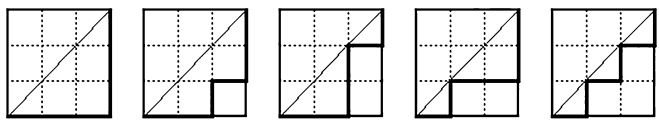


Рис. 5.5. Возможные монотонные пути для случая $n = 3$

Глава 6 Кодирование

Вопросы кодирования издавна играли заметную роль в математике.

Примеры

1. *Десятичная позиционная система счисления* — это универсальный способ кодирования чисел, в том числе натуральных.
2. *Римские цифры* — другой способ кодирования небольших натуральных чисел, причём гораздо более наглядный и естественный: палец — I, пятерня — V, две пятерни — X. Однако при этом способе кодирования трудно выполнять арифметические операции над большими числами, поэтому он был вытеснен десятичной позиционной системой.
3. *Декартовы координаты* — способ кодирования точек и других геометрических объектов числами.

Ранее средства кодирования играли вспомогательную роль и не рассматривались как отдельный предмет математического изучения, но с появлением компьютеров ситуация радикально изменилась. Кодирование буквально пронизывает информационные технологии и является центральным вопросом при решении самых разных задач программирования.

Примеры

1. Представление данных произвольной природы (чисел, текста, графики) в памяти компьютера.
2. Защита информации от несанкционированного доступа.
3. Обеспечение помехоустойчивости при передаче данных по каналам связи.
4. Сжатие информации в базах данных.

ЗАМЕЧАНИЕ

Составление текста программы часто и совершенно справедливо называют кодированием.

Не ограничивая общности, задачу кодирования можно сформулировать следующим образом. Пусть заданы алфавиты $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_m\}$ и функция $F\colon A^* \to B^*$, причём $\mathrm{Dom}\, F = S$, где S — некоторое множество слов в алфавите A, $S \subset A^*$. Тогда функция F называется *кодированием*, элементы множества S — *сообщениями*, а элементы $\beta = F(\alpha)$, $\alpha \in S$, $\beta \in B^*$ — *кодами* (соответствующих сообщений). Обратная функция F^{-1} (если она существует!) называется *декодированием*. Если $|B| = m$, то F называется *m-ичным кодированием*. Наиболее распространенный случай $B = \{0, 1\}$ — *двоичное* кодирование. Именно этот случай рассматривается в последующих разделах; слово «двоичное» опускается.

Типичная задача теории кодирования формулируется следующим образом: при заданных алфавитах A, B и множестве сообщений S найти кодирование F, которое обладает определёнными свойствами (то есть удовлетворяет заданным ограничениям) и оптимально в некотором смысле. Критерий оптимальности, как правило,

связан с минимизацией длин кодов. Свойства, которые требуются от кодирования, бывают самой разнообразной природы.

1. Существование декодирования, или *однозначность* кодирования: функция кодирования F обладает тем свойством, что $\alpha_1 \neq \alpha_2 \Longrightarrow F(\alpha_1) \neq F(\alpha_2)$. Это очень естественное свойство, однако даже оно требуется не всегда. Например, трансляция программы на языке высокого уровня в машинные команды — это кодирование, для которого не требуется однозначного декодирования.
2. *Помехоустойчивость*, или исправление ошибок: продолжение функции декодирования F^{-1} обладает тем свойством, что $F^{-1}(\beta) = F^{-1}(\beta')$, где $\beta \in \operatorname{Im} F$, $\beta' \in B^* \setminus \operatorname{Im} F$, если β' в определённом смысле близко к β (см. раздел 6.3).
3. Заданная сложность (или простота) кодирования и декодирования. Например, в криптографии изучаются такие способы кодирования, при которых функция F вычисляется просто, но определение значения функции F^{-1} требует очень сложных вычислений (см. п. 6.4.7).

Большое значение для задач кодирования имеет природа множества сообщений S. При одних и тех же алфавитах A, B и требуемых свойствах кодирования F оптимальные решения для разных S могут кардинально различаться. Для описания множества S (как правило, очень большого или бесконечного) применяются различные методы:

1) теоретико-множественное описание, например $S = \{\alpha \mid \alpha \in A^* \ \&\ |\alpha| = n\}$;
2) вероятностное описание, например $S = A^*$, и заданы вероятности p_i появления букв a_i в сообщении, $\sum_{i=1}^{n} p_i = 1$;
3) логико-комбинаторное описание, например S задано порождающей формальной грамматикой.

В этой главе рассматривается несколько наиболее важных задач теории кодирования и демонстрируется применение большей части вышеупомянутых методов.

6.1. Алфавитное кодирование

Кодирование F может сопоставлять код всему сообщению из множества S как единому целому или же строить код сообщения из кодов его частей. Элементарной частью сообщения является одна буква алфавита A. Этот простейший случай рассматривается в этом и следующих двух разделах.

6.1.1. Таблица кодов

Алфавитное (или *побуквенное*) кодирование задаётся *схемой* (или *таблицей кодов*):

$$\sigma \stackrel{\text{Def}}{=} \langle a_1 \to \beta_1, \ldots, a_n \to \beta_n \rangle, \quad a_i \in A, \beta_i \in B^*.$$

Множество кодов букв $V \stackrel{\text{Def}}{=} \{\beta_i\}$ называется множеством *элементарных кодов* (множеством *кодовых слов*). Алфавитное кодирование пригодно для любого множества сообщений, поскольку $F\colon A^* \to B^*$, и если $a_{i_1} \ldots a_{i_k} = \alpha \in A^*$, то $F(\alpha) \stackrel{\text{Def}}{=} \beta_{i_1} \ldots \beta_{i_k}$.

Пример. Рассмотрим алфавиты $A := \{0,1,2,3,4,5,6,7,8,9\}$, $B := \{0,1\}$ и схему

$$\sigma_1 := \langle 0 \to 0, 1 \to 1, 2 \to 10, 3 \to 11, 4 \to 100,$$
$$5 \to 101, 6 \to 110, 7 \to 111, 8 \to 1000, 9 \to 1001 \rangle.$$

Эта схема однозначна, но кодирование не является взаимно-однозначным: $F_{\sigma_1}(333) = 111111 = F_{\sigma_1}(77)$, а значит, декодирование невозможно. С другой стороны, схема

$$\sigma_2 := \langle 0 \to 0000, 1 \to 0001, 2 \to 0010, 3 \to 0011, 4 \to 0100,$$
$$5 \to 0101, 6 \to 0110, 7 \to 0111, 8 \to 1000, 9 \to 1001 \rangle,$$

известная под названием «*двоично-десятичное кодирование*», допускает однозначное декодирование.

6.1.2. Разделимые схемы

Рассмотрим схему алфавитного кодирования σ и различные слова, составленные из элементарных кодов. Схема σ называется *разделимой*, если

$$\beta_{i_1} \dots \beta_{i_k} = \beta_{j_1} \dots \beta_{j_l} \Longrightarrow k = l \ \&\ \forall t \in 1..k \ \ (i_t = j_t)\,,$$

то есть любое слово, составленное из элементарных кодов, единственным образом разлагается на элементарные коды. Алфавитное кодирование с разделимой схемой допускает декодирование.

Если таблица кодов содержит одинаковые элементарные коды, то есть если

$$\exists\, i, j \ \ (i \neq j \ \&\ \beta_i = \beta_j)\,,$$

где $\beta_i, \beta_j \in V$, то схема заведомо не является разделимой. Такие схемы далее не рассматриваются, то есть $\forall\, i \neq j \ \ (\beta_i, \beta_j \in V \Longrightarrow \beta_i \neq \beta_j)$.

6.1.3. Префиксные схемы

Схема σ называется *префиксной*, если элементарный код одной буквы не является префиксом элементарного кода другой буквы:

$$\forall\, i \neq j \ \ (\beta_i, \beta_j \in V) \Longrightarrow \forall\, \beta \in B^* \ \ (\beta_i \neq \beta_j \beta)\,.$$

ТЕОРЕМА. *Префиксная схема является разделимой.*

Доказательство. От противного. Пусть кодирование с префиксной схемой σ не является разделимым. Тогда существует такое слово $\beta \in F_\sigma(A^*)$, что

$$\beta = \beta_{i_1} \dots \beta_{i_k} = \beta_{j_1} \dots \beta_{j_l} \ \&\ \exists\, t \ \ (\forall\, s < t \ \ (\beta_{i_s} = \beta_{j_s} \ \&\ \beta_{i_t} \neq \beta_{j_t}))\,.$$

Поскольку $\beta_{i_t} \dots \beta_{i_k} = \beta_{jt} \dots \beta_{jl}$, значит, $\exists\, \beta' \ \ (\beta_{i_t} = \beta_{j_t}\beta' \vee \beta_{j_t} = \beta_{i_t}\beta')$, но это противоречит тому, что схема префиксная. □

ЗАМЕЧАНИЕ

Свойство быть префиксной достаточно, но не необходимо для разделимости схемы.

Пример. Разделимая, но не префиксная схема:

$$A = \{a, b\}, \quad B = \{0, 1\}, \quad \sigma = \langle a \to 0, b \to 01 \rangle.$$

6.1.4. Неравенство Крафта–Макмиллана

Чтобы схема алфавитного кодирования была разделимой, необходимо, чтобы длины элементарных кодов удовлетворяли определённому соотношению, известному как *неравенство Крафта–Макмиллана*.

ТЕОРЕМА 1. *Если схема* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ *разделима, то* $\sum\limits_{i=1}^{n} 2^{-|\beta_i|} \leqslant 1$.

Доказательство. Обозначим $l_i := |\beta_i|$, $l := \max\limits_{i \in 1..n} l_i$. Рассмотрим некоторую k-ю степень левой части неравенства $\left(\sum\limits_{i=1}^{n} 2^{-l_i}\right)^k$. Раскрывая скобки, имеем сумму $\sum\limits_{(i_1,\ldots,i_k)} (2^{l_{i_1}+\ldots+l_{i_k}})^{-1}$, где $i_1, \ldots, i_k$ — различные наборы номеров элементарных кодов. Обозначим через $\nu(k,t)$ количество входящих в эту сумму слагаемых вида $1/2^t$, где $t = l_{i_1} + \ldots + l_{i_k}$. Для некоторых t может быть, что $\nu(k,t) = 0$. Приводя подобные, имеем сумму $\sum\limits_{t=1}^{kl} \frac{\nu(k,t)}{2^t}$. Каждому слагаемому вида $(2^{l_{i_1}+\ldots+l_{i_k}})^{-1}$ можно сопоставить код (слово в алфавите B) вида $\beta_{i_1} \ldots \beta_{i_k}$. Это слово состоит из k элементарных кодов и имеет длину t. Таким образом, $\nu(k,t)$ — это число некоторых слов вида $\beta_{i_1} \ldots \beta_{i_k}$, таких, что $|\beta_{i_1} \ldots \beta_{i_k}| = t$. В силу разделимости схемы $\nu(k,t) \leqslant 2^t$, в противном случае заведомо существовали бы два одинаковых слова $\beta_{i_1} \ldots \beta_{i_k} = \beta_{j_1} \ldots \beta_{j_k}$, допускающих различное разложение. Имеем

$$\sum_{t=1}^{kl} \frac{\nu(k,t)}{2^t} \leqslant \sum_{t=1}^{kl} \frac{2^t}{2^t} = kl.$$

Следовательно, $\forall\, k \quad \left(\left(\sum\limits_{i=1}^{n} 2^{-l_i}\right)^k \leqslant kl\right)$ и, значит, $\forall\, k \quad \left(\sum\limits_{i=1}^{n} 2^{-l_i} \leqslant \sqrt[k]{kl}\right)$, откуда $\sum\limits_{i=1}^{n} 2^{-l_i} \leqslant \lim\limits_{k\to\infty} \sqrt[k]{kl} = 1$. □

Неравенство Крафта–Макмиллана является не только необходимым, но и в некотором смысле достаточным условием разделимости схемы алфавитного кодирования.

ТЕОРЕМА 2. *Если числа* $l_1, \ldots, l_n$ *удовлетворяют неравенству* $\sum\limits_{i=1}^{n} 2^{-l_i} \leqslant 1$, *то существует такая разделимая (даже префиксная) схема алфавитного кодирования* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$, *что* $\forall\, i \;\; (|\beta_i| = l_i)$.

Доказательство. Без ограничения общности можно считать, что $l_1 \leqslant l_2 \leqslant \cdots \leqslant l_n$. Разобьём множество $\{l_1, \ldots, l_n\}$ на классы эквивалентности по равенству $\{L_1, \ldots, L_m\}$, $m \leqslant n$. Пусть $\lambda_i \in L_i$, $\mu_i := |L_i|$. Тогда $\sum\limits_{i=1}^{m} \mu_i = n$, $\lambda_1 < \lambda_2 < \cdots < \lambda_m$. В этих обозначениях неравенство Крафта–Макмиллана можно записать так: $\sum\limits_{i=1}^{m} \frac{\mu_i}{2^{\lambda_i}} \leqslant 1$. Из

этого неравенства следуют m неравенств для частичных сумм:

$$\frac{\mu_1}{2^{\lambda_1}} \leqslant 1 \Longrightarrow \mu_1 \leqslant 2^{\lambda_1},$$
$$\frac{\mu_1}{2^{\lambda_1}} + \frac{\mu_2}{2^{\lambda_2}} \leqslant 1 \Longrightarrow \mu_2 \leqslant 2^{\lambda_2} - \mu_1 2^{\lambda_2-\lambda_1},$$
$$\dots$$
$$\frac{\mu_1}{2^{\lambda_1}} + \frac{\mu_2}{2^{\lambda_2}} + \dots + \frac{\mu_m}{2^{\lambda_m}} \leqslant 1 \Longrightarrow \mu_m \leqslant 2^{\lambda_m} - \mu_1 2^{\lambda_m-\lambda_1} - \mu_2 2^{\lambda_m-\lambda_2} - \dots - \mu_{m-1} 2^{\lambda_m-\lambda_{m-1}}.$$

Рассмотрим слова длины λ_1 в алфавите B. Поскольку $\mu_1 \leqslant 2^{\lambda_1}$, из этих слов можно выбрать μ_1 различных слов $\beta_1, \dots, \beta_{\mu_1}$ длины λ_1. Исключим из дальнейшего рассмотрения все слова из B^*, начинающиеся со слов $\beta_1, \dots, \beta_{\mu_1}$. Далее рассмотрим множество слов в алфавите B длиной λ_2 и не начинающихся со слов $\beta_1, \dots, \beta_{\mu_1}$. Таких слов будет $2^{\lambda_2} - \mu_1 2^{\lambda_2-\lambda_1}$. Но $\mu_2 \leqslant 2^{\lambda_2} - \mu_1 2^{\lambda_2-\lambda_1}$, значит, можно выбрать μ_2 различных слов. Обозначим их $\beta_{\mu_1+1}, \dots, \beta_{\mu_1+\mu_2}$. Исключим слова, начинающиеся с $\beta_{\mu_1+1}, \dots, \beta_{\mu_1+\mu_2}$, из дальнейшего рассмотрения. И далее, используя неравенства для частичных сумм, мы будем на i-м шаге выбирать μ_i слов длины λ_i, $\beta_{\mu_1+\mu_2+\dots+\mu_{i-1}}, \dots, \beta_{\mu_1+\mu_2+\dots+\mu_{i-1}+\mu_i}$, причём эти слова не будут начинаться с тех слов, которые были выбраны раньше. В то же время длины этих слов всё время растут (так как $\lambda_1 < \lambda_2 < \dots < \lambda_m$), поэтому они не могут быть префиксами тех слов, которые выбраны раньше. Итак, в конце имеем набор из n слов $\beta_1, \dots, \beta_{\mu_1+\dots+\mu_m} = \beta_n$, $|\beta_1| = l_1, \dots, |\beta_n| = l_n$, коды $\beta_1, \dots, \beta_n$ не являются префиксами друг друга, а значит, схема $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ будет префиксной и, по теореме предыдущего параграфа, разделимой. □

ЗАМЕЧАНИЕ

Неравенство Макмиллана справедливо не только для двоичного, но и для m-ичного кодирования.

Пример. *Азбука Морзе*[1] — это схема алфавитного кодирования

$\langle A \to 01$, $B \to 1000$, $C \to 1010$, $D \to 100$, $E \to 0$,
$F \to 0010$, $G \to 110$, $H \to 0000$, $I \to 00$, $J \to 0111$,
$K \to 101$, $L \to 0100$, $M \to 11$, $N \to 10$, $O \to 111$,
$P \to 0110$, $Q \to 1101$, $R \to 010$, $S \to 000$, $T \to 1$,
$U \to 001$, $V \to 0001$, $W \to 011$, $X \to 1001$, $Y \to 1011$,
$Z \to 1100\rangle$,

где по историческим и техническим причинам 0 называется *точкой* и обозначается знаком «·», а 1 называется *тире* и обозначается знаком «—». Имеем

$$\begin{array}{lllllllllll}
 & 1/4 & + & 1/16 & + & 1/16 & + & 1/8 & + & 1/2 & + \\
+ & 1/16 & + & 1/8 & + & 1/16 & + & 1/4 & + & 1/16 & + \\
+ & 1/8 & + & 1/16 & + & 1/4 & + & 1/4 & + & 1/8 & + \\
+ & 1/16 & + & 1/16 & + & 1/8 & + & 1/8 & + & 1/2 & + \\
+ & 1/8 & + & 1/16 & + & 1/8 & + & 1/16 & + & 1/16 & + \\
+ & 1/16 & = & & & & & & & & \\
= & 2/2 & + & 4/4 & + & 8/8 & + & 12/16 & = & & \\
= & 3 & + & 3/4 & > & 1. & & & & &
\end{array}$$

[1] Самуэль Морзе (1791–1872).

Таким образом, неравенство Макмиллана для азбуки Морзе не выполнено, и эта схема не является разделимой. Более того, в азбуке Морзе многие кодовые слова являются префиксами других слов, а потому некоторые комбинации букв неразличимы.

Пример. $AM = 0111 = J, \quad TO = 1111 = MM, \quad WMW = 01111011 = JY.$

На самом деле в азбуке Морзе имеются дополнительные элементы — *паузы* между буквами (и словами), которые позволяют декодировать сообщения. Эти дополнительные элементы определены неформально, поэтому приём и передача сообщений с помощью азбуки Морзе, особенно с высокой скоростью, являлись некоторым искусством, а не простой технической процедурой.

СЛЕДСТВИЕ. *Если схема алфавитного кодирования* $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ *разделима, то существует префиксная схема* $\sigma' = \langle a_i \to \beta'_i \rangle_{i=1}^n$, *причём* $\forall\, i \;\; (|\beta_i| = |\beta'_i|)$.

Пример. Схема $\langle a \to 0, b \to 01 \rangle$ — разделимая, но не префиксная, а схема $\langle a \to 0, b \to 10 \rangle$ — префиксная (и разделимая).

6.2. Кодирование с минимальной избыточностью

Для практики важно, чтобы коды сообщений имели по возможности наименьшую длину. Алфавитное кодирование пригодно для любых сообщений, то есть для $S = A^*$. Если больше про множество S ничего не известно, то точно сформулировать задачу оптимизации затруднительно. Однако на практике часто доступна дополнительная информация. Например, для текстов на естественных языках известно распределение вероятности появления букв в сообщении. Использование такой информации позволяет строго поставить и решить задачу построения оптимального алфавитного кодирования.

6.2.1. Минимизация длины кода сообщения

Если задана разделимая схема алфавитного кодирования $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$, то любая схема $\sigma' = \langle a_i \to {\beta_i}' \rangle_{i=1}^n$, где последовательность $\langle {\beta'}_1, \ldots, {\beta'}_n \rangle$ является перестановкой последовательности $\langle \beta_1, \ldots, \beta_n \rangle$, также будет разделимой. Если длины элементарных кодов равны, то перестановка элементарных кодов в схеме не влияет на длину кода сообщения. Но если длины элементарных кодов различны, то длина кода сообщения зависит от состава букв в сообщении и от того, какие элементарные коды каким буквам назначены. Если заданы конкретное сообщение и конкретная схема кодирования, то нетрудно подобрать такую перестановку элементарных кодов, при которой длина кода сообщения будет минимальна. Пусть $k_1, \ldots, k_n$ — количества вхождений букв $a_1, \ldots, a_n$ в сообщение $s \in S$, а $l_1, \ldots, l_n$ — длины элементарных кодов $\beta_1, \ldots, \beta_n$ соответственно. Тогда, если $k_i \leqslant k_j$ и $l_i \geqslant l_j$, то $k_i l_i + k_j l_j \leqslant k_i l_j + k_j l_i$. Действительно, пусть $k_j = k + a$, $k_i = k$ и $l_j = l$, $l_i = l + b$, где $a, b \geqslant 0$. Тогда $(k_i l_j + k_j l_i) - (k_i l_i + k_j l_j) = (kl + (k+a)(l+b)) - (k(l+b) + l(k+a)) =$
$= (kl + al + bk + ab + kl) - (kl + al + kl + bk) = ab \geqslant 0$. Отсюда вытекает алгоритм назначения элементарных кодов, при котором длина кода конкретного сообщения $s \in S$ будет минимальна: нужно отсортировать буквы сообщения s в порядке убывания количества вхождений, элементарные коды отсортировать в порядке возрастания длины и назначить коды буквам в этом порядке.

Пример. Для алфавита из трёх букв кратчайшая разделимая схема имеет вид $\langle x \to 0, y \to 10, z \to 11\rangle$ (или $\langle x \to 1, y \to 01, z \to 00\rangle$), поскольку схемы с более короткими кодовыми словами не удовлетворяют неравенству Макмиллана. Если необходимо передать сообщение «SOS SOS SOS», то самой выгодной является схема $\langle S \to 0, O \to 10, \ \to 11\rangle$.

ЗАМЕЧАНИЕ

Этот простой метод решает задачу минимизации длины кода только для фиксированного сообщения $s \in S$ и фиксированной схемы σ.

6.2.2. Цена кодирования

Пусть заданы алфавит $A = \{a_1, \ldots, a_n\}$ и вероятности появления букв в сообщении $P = \langle p_1, \ldots, p_n\rangle$ (p_i — вероятность появления буквы a_i). Не ограничивая общности, можно считать, что $p_i + \ldots + p_n = 1$ и $p_1 \geqslant \ldots \geqslant p_n > 0$ (то есть можно сразу исключить буквы, которые не могут появиться в сообщении, и упорядочить буквы по убыванию вероятности их появления). Для каждой (разделимой) схемы $\sigma = \langle a_i \to \beta_i\rangle_{i=1}^n$ алфавитного кодирования математическое ожидание коэффициента увеличения длины сообщения при кодировании σ (обозначается l_σ) определяется следующим образом: $l_\sigma(P) \stackrel{\text{Def}}{=} \sum_{i=1}^{n} p_i l_i$, где $l_i \stackrel{\text{Def}}{=} |\beta_i|$, и называется средней *ценой* (или *длиной*) кодирования σ при распределении вероятностей P (см. п. 5.1.7).

Пример. Для разделимой схемы $A = \{a, b\}$, $B = \{0, 1\}$, $\sigma = \langle a \to 0, b \to 01\rangle$ при распределении вероятностей $\langle 0,5; 0,5\rangle$ цена кодирования составляет $0,5 * 1 + 0,5 * 2 = 1,5$, а при распределении вероятностей $\langle 0,9; 0,1\rangle$ она равна $0,9 * 1 + 0,1 * 2 = 1,1$.

Введём обозначения:

$$l_*(P) \stackrel{\text{Def}}{=} \inf_\sigma l_\sigma(P), \qquad p_* \stackrel{\text{Def}}{=} \min_{i=1}^{n} p_i, \qquad L \stackrel{\text{Def}}{=} \lfloor \log_2(n-1)\rfloor + 1.$$

Очевидно, что всегда существует разделимая схема $\sigma = \langle a_i \to \beta_i\rangle_{i=1}^n$ такая, что $\forall\, i \ \ (|\beta_i| = L)$. Такая схема называется схемой *равномерного* кодирования. Следовательно, $1 \leqslant l_*(P) \leqslant L$, и достаточно учитывать только такие схемы, для которых $\forall\, i \ \ (p_i l_i \leqslant L)$, где l_i — целое и $l_i \leqslant L/p_*$. Таким образом, имеется лишь конечное число схем σ, для которых $l_*(P) \leqslant l_\sigma(P) \leqslant L$. Значит, существует схема σ_*, на которой инфимум достигается: $l_{\sigma_*}(P) = l_*(P)$.

Алфавитное (разделимое) кодирование σ_*, для которого $l_{\sigma_*}(P) = l_*(P)$, называется кодированием с *минимальной избыточностью* или *оптимальным* кодированием для распределения вероятностей P.

Пример. Пусть в алфавите 8 букв. Рассмотрим цену равномерного кодирования C_1 и «наиболее неравномерного» кодирования C_2 при равномерном P_1 и неравномерном P_2 распределении вероятностей.

C_1	l_1	P_1	L_{11}	P_2	L_{12}	C_2	l_2	P_1	L_{21}	P_2	L_{22}
000	3	0,125	0,375	0,31	0,93	0	1	0,125	0,125	0,31	0,31
001	3	0,125	0,375	0,24	0,72	10	2	0,125	0,25	0,24	0,48
010	3	0,125	0,375	0,17	0,51	110	3	0,125	0,375	0,17	0,51
011	3	0,125	0,375	0,11	0,33	1110	4	0,125	0,5	0,11	0,44
100	3	0,125	0,375	0,09	0,27	11110	5	0,125	0,625	0,09	0,45
101	3	0,125	0,375	0,05	0,15	111110	6	0,125	0,75	0,05	0,3
110	3	0,125	0,375	0,02	0,06	1111110	7	0,125	0,875	0,02	0,14
111	3	0,125	0,375	0,01	0,03	11111110	8	0,125	1	0,01	0,08
		1	3	1	3			1	4,5	1	2,71

Из таблицы видно, что при равномерном кодировании распределение вероятностей не влияет на цену кодирования, в то время как неравномерное кодирование по сравнению с равномерным кодированием существенно проигрывает при равномерном распределении вероятностей и несколько выигрывает при неравномерном распределении вероятностей.

6.2.3. Алгоритм Фано

Рекурсивный алгоритм Фано[1] строит разделимую префиксную схему алфавитного кодирования, близкого к оптимальному. Алгоритм использует две глобальные структуры данных:

P : **array** $[1..n]$ **of real** — массив вероятностей появления букв в сообщении, упорядоченный по невозрастанию; $1 \geqslant P[1] \geqslant \ldots \geqslant P[n] > 0$, $P[1] + \ldots + P[n] = 1$.

C : **array** $[1..n, 1..L]$ **of** $0..1$ — массив элементарных кодов.

Основная работа по построению элементарных кодов выполняется рекурсивной процедурой Fano. Начальный вызов процедуры имеет вид $\mathrm{Fano}(1, n, 0)$.

Алгоритм 6.1. Процедура Fano построения кодирования, близкого к оптимальному

Вход: b — индекс начала обрабатываемой части массива P, e — индекс конца обрабатываемой части массива P, k — длина уже построенных кодов в обрабатываемой части массива C.

Выход: заполненный массив C.

```
if e > b then
  k := k + 1  // место для очередного разряда в коде
  m := Med(b, e)  // деление массива на две части
  for i from b to e do
    C[i, k] := i > m  // в первой части добавляем 0, во второй — 1
  end for
  Fano(b, m, k)  // обработка первой части
  Fano(m + 1, e, k)  // обработка второй части
end if
```

Функция Med находит *медиану* указанной части массива $P[b..e]$, то есть определяет такой индекс m ($b \leqslant m < e$), что сумма элементов $P[b..m]$ наиболее близка к сумме элементов $P[(m+1)..e]$.

[1] Роберт Фано (род. 1917).

Алгоритм 6.2. Функция Med определения медианы

Вход: b — индекс начала обрабатываемой части массива P, e — индекс конца обрабатываемой части массива P.

Выход: m — индекс медианы, то есть $\min\limits_{m\in b..(e-1)} \left|\sum\limits_{i=b}^{m} P[i] - \sum\limits_{i=m+1}^{e} P[i]\right|$.

$S_b := 0$ // сумма элементов первой части
for i **from** b **to** $e-1$ **do**
  $S_b := S_b + P[i]$ // вначале все, кроме последнего
end for
$S_e := P[e]$ // сумма элементов второй части
$m := e$ // начинаем искать медиану с конца
repeat
  $d := |S_b - S_e|$ // разность сумм первой и второй частей
  $m := m - 1$ // сдвигаем границу медианы вниз
  $S_b := S_b - P[m]; S_e := S_e + P[m]$ // перевычисляем суммы
until $|S_b - S_e| \geqslant d$
return m

Обоснование. При каждом удлинении кодов в одной части коды удлиняются нулями, а в другой — единицами. Таким образом, коды одной части не могут быть префиксами другой. Удлинение кода заканчивается тогда и только тогда, когда длина части равна 1, то есть остается единственный код. Таким образом, схема по построению префиксная, а потому разделимая. □

Пример. Коды, построенные алгоритмом Фано для заданного неравномерного распределения вероятностей ($n = 8$).

p_i							l_i	$p_i l_i$
0,31	0,31					0	1	0,31
0,24	0,69		0,24			100	3	0,72
0,17		0,41	0,17			101	3	0,51
0,11		0,28	0,11			110	3	0,33
0,09			0,17	0,09		1110	4	0,36
0,05				0,08	0,05	11110	5	0,25
0,02					0,03	0,02	111110	6	0,12
0,01						0,01	111111	6	0,06
								2,66

6.2.4. Оптимальное кодирование

Оптимальные схемы алфавитного кодирования обладают определёнными структурными свойствами, устанавливаемыми в следующих леммах. Нарушение этих свойств для какой-либо схемы означает, что схема не оптимальна.

Лемма 1. *Пусть $\sigma = \langle a_i \to \beta_i\rangle_{i=1}^{n}$ — схема оптимального кодирования для распределения вероятностей $P = p_1 \geqslant \ldots \geqslant p_n > 0$. Тогда если $p_i > p_j$, то $l_i \leqslant l_j$.*

Доказательство. От противного. Пусть $i < j$, $p_i > p_j$ и $l_i > l_j$. Тогда рассмотрим схему $\sigma' = \{a_1 \to \beta_1, \ldots, a_i \to \beta_j, \ldots, a_j \to \beta_i, \ldots, a_n \to \beta_n\}$. Имеем $l_\sigma - l_{\sigma'} = (p_i l_i + p_j l_j) - (p_i l_j + p_j l_i) = (p_i - p_j)(l_i - l_j) > 0$, что противоречит тому, что схема σ — оптимальна. □

Таким образом, не ограничивая общности, можно считать, что $l_1 \leqslant \ldots \leqslant l_n$.

ЗАМЕЧАНИЕ

Фактически, лемма 1 повторяет для вероятностей то, что было обнаружено для частот в п. 6.2.1.

ЛЕММА 2. *Если $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ — схема оптимального префиксного кодирования для распределения вероятностей $P = p_1 \geqslant \ldots \geqslant p_n > 0$, то среди элементарных кодов имеются два кода максимальной длины, которые различаются только в последнем разряде.*

Доказательство. По предыдущей лемме кодовые слова максимальной длины являются последними в схеме. Далее от противного.

[1] Пусть кодовое слово максимальной длины одно и имеет вид $\beta_n = \beta b$, где $b = 0 \vee b = 1$. Имеем $\forall\, i \in 1..n-1 \;\; (l_i \leqslant |\beta|)$. Так как схема префиксная, то слова $\beta_1, \ldots, \beta_{n-1}$ не являются префиксами β. С другой стороны, β не является префиксом слов $\beta_1, \ldots, \beta_{n-1}$, иначе было бы $\beta = \beta_j$, а значит, β_j было бы префиксом β_n. Тогда схема $\sigma' := \langle a_1 \to \beta_1, \ldots, a_n \to \beta \rangle$ тоже префиксная, причём $l_{\sigma'}(P) = l_\sigma(P) - p_n$, что противоречит оптимальности σ.

[2] Пусть теперь два кодовых слова β_{n-1} и β_n максимальной длины различаются не в последнем разряде, то есть $\beta_{n-1} = \beta' b'$, $\beta_n = \beta'' b''$, $\beta' \neq \beta''$, причём β', β'' не являются префиксами для $\beta_1, \ldots, \beta_{n-2}$ и наоборот. Тогда схема $\sigma' := \langle a_1 \to \beta_1, \ldots, a_{n-2} \to \beta_{n-2}, a_{n-1} \to \beta' b', a_n \to \beta'' \rangle$ также является префиксной, причём $l_{\sigma'}(P) = l_\sigma(P) - p_n$, что противоречит оптимальности σ.

[3] Пусть кодовых слов максимальной длины больше двух. Тогда среди них найдутся два, которые различаются не только в последнем разряде, что противоречит уже доказанному пункту 2. □

ЗАМЕЧАНИЕ

В примере п. 6.2.3 имеется не два, а шесть элементарных кодов максимальной длины, что противоречит лемме 2 и говорит о том, что кодирование Фано не оптимально.

ТЕОРЕМА. *Если $\sigma_{n-1} = \langle a_i \to \beta_i \rangle_{i=1}^{n-1}$ — схема оптимального префиксного кодирования для распределения вероятностей $P = p_1 \geqslant \ldots \geqslant p_{n-1} > 0$ и $p_j = q' + q''$, причём $p_1 \geqslant \ldots \geqslant p_{j-1} \geqslant p_{j+1} \geqslant \ldots \geqslant p_{n-1} \geqslant q' \geqslant q'' > 0$, то кодирование со схемой $\sigma_n = \langle a_1 \to \beta_1, \ldots, a_{j-1} \to \beta_{j-1}, a_{j+1} \to \beta_{j+1}, \ldots, a_{n-1} \to \beta_{n-1}, a_j \to \beta_j 0, a_n \to \beta_j 1 \rangle$ является оптимальным префиксным кодированием для распределения вероятностей $P_n = p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_{n-1}, q', q''$.*

Доказательство. Заметим следующее.

[1] Если σ_{n-1} было префиксным, то σ_n тоже будет префиксным по построению.

[2] Цена кодирования для схемы σ_n больше цены кодирования для схемы σ_{n-1} на величину p_j: $l_{\sigma_n}(P_n) = p_1 l_1 + p_2 l_2 + \ldots + p_{j-1} l_{j-1} + p_{j+1} l_{j+1} + \ldots + p_{n-1} l_{n-1} +$

$+q'(l_j+1)+q''(l_j+1)=p_1l_1+\ldots+p_{n-1}l_{n-1}+l_j(q'+q'')+(q'+q'')=p_1l_1+\ldots+$
$+p_{n-1}l_{n-1}+l_jp_j+p_j=l_{\sigma_{n-1}}(P_{n-1})+p_j$.

[3] Пусть некоторая схема $\sigma'_n:=\langle a_i\to\beta_i\rangle_{i=1}^n$ оптимальна для P_n. Тогда по лемме 2 два кода максимальной длины различаются в последнем разряде: $\sigma'_n=\langle a_1\to\beta'_1,\ldots,a_{n-2}\to\beta'_{n-2},a_{n-1}\to\beta0,a_n\to\beta1\rangle$. Положим $l':=|\beta|$ и рассмотрим схему $\sigma'_{n-1}:=\langle a_1\to\beta'_1,\ldots,a_j\to\beta,\ldots,a_{n-2}\to\beta'_{n-2}\rangle$, где j определено так, чтобы $p_{j-1}\geqslant q'+q''\geqslant p_{j+1}$. Цена кодирования для схемы σ'_n больше цены кодирования для схемы σ'_{n-1} также на величину p_j: $l_{\sigma'_n}(P_n)=l_1p_1+\ldots+l_{n-2}p_{n-2}+(l'+1)q'+$
$+(l'+1)q''=l_1p_1+\ldots+l_{n-2}p_{n-2}+l'(q'+q'')+(q'+q'')=l_{\sigma'_{n-1}}(P_{n-1})+p_j$.

[4] Схема σ'_n — префиксная, значит, схема σ'_{n-1} — тоже префиксная.

[5] Схема σ_{n-1} — оптимальна, значит, $l_{\sigma'_{n-1}}(P_{n-1})\geqslant l_{\sigma_{n-1}}(P_{n-1})$.

[6] Имеем $l_{\sigma_n}(P_n)=l_{\sigma_{n-1}}(P_{n-1})+p_j\leqslant l_{\sigma'_{n-1}}(P_{n-1})+p_j=l_{\sigma'_n}(P_n)$. Но схема σ'_n оптимальна, значит, схема σ_n также оптимальна. □

6.2.5. Алгоритм Хаффмена

Алгоритм Хаффмена[1] строит схему оптимального префиксного алфавитного кодирования для заданного распределения вероятностей появления букв.

Алгоритм 6.3. Рекурсивная процедура Huffman

Вход: количество букв n, массив вероятностей букв P : **array** $[1..n]$ **of real**, упорядоченный по убыванию.
Выход: массив элементарных кодов C : **array** $[1..n,1..L]$ **of** 0..1, массив длин элементарных кодов ℓ : **array** $[1..n]$ **of** $1..L$.

```
if n = 2 then
    C[1,1] := 0; ℓ[1] := 1  // первый элемент
    C[2,1] := 1; ℓ[2] := 1  // второй элемент
else
    q := P[n−1] + P[n]  // сумма двух последних вероятностей
    j := Up(n, q)  // поиск места и вставка суммы
    Huffman(P, n−1)  // рекурсивный вызов
    Down(n, j)  // достраивание кодов
end if
```

Функция Up находит в массиве P место, в котором должно находиться число q (п. 6.2.4), и вставляет это число, сдвигая вниз остальные элементы.

Вход: n — длина обрабатываемой части массива P, q — вставляемая сумма.
Выход: измененный массив P.

```
j := 1  // место вставляемого элемента
for i from n−1 downto 2 do
    if P[i−1] < q then
        P[i] := P[i−1]  // сдвиг элемента массива
```

[1] Дэвид А. Хаффмен (1925–1999).

```
    else
      j := i  // определение места вставляемого элемента
      exit for i  // всё сделано — цикл не нужно продолжать
    end if
  end for
  P[j] := q  // запись вставляемого элемента
  return j
```

Процедура Down строит оптимальный код для n букв на основе построенного оптимального кода для $n-1$ буквы. Для этого код буквы с номером j временно исключается из массива C путем сдвига вверх кодов букв с номерами, большими j, а затем в конец обрабатываемой части массива C добавляется пара кодов, полученных из кода буквы с номером j удлинением на 0 и 1 соответственно. Здесь $C[i,*]$ означает вырезку из массива, то есть i-ю строку массива C.

Вход: n — длина обрабатываемой части массива P, j — номер «разделяемой» буквы.
Выход: оптимальные коды в первых n элементах массивов C и ℓ.

```
  c := C[j,*]  // запоминание кода буквы j
  l := ℓ[j]  // и длины этого кода
  for i from j to n − 2 do
    C[i,*] := C[i+1,*]  // сдвиг кода
    ℓ[i] := ℓ[i+1]  // и его длины
  end for
  C[n−1,*] := c; C[n,*] := c  // копирование кода буквы j
  C[n−1,l+1] := 0; C[n,l+1] := 1  // наращивание кодов
  ℓ[n−1] := l+1; ℓ[n] := l+1  // и увеличение длин
```

Обоснование. Для пары букв при любом распределении вероятностей оптимальное кодирование очевидно: первой букве нужно назначить код 0, а второй — 1. Именно это и делается в первой части оператора **if** основной процедуры Huffman. Рекурсивная часть алгоритма в точности следует доказательству теоремы предыдущего подраздела. С помощью функции Up в исходном упорядоченном массиве P отбрасываются две последние (наименьшие) вероятности, и их сумма вставляется в массив P так, чтобы массив (на единицу меньшей длины) остался упорядоченным. Заметим, что при этом место вставки сохраняется в локальной переменной j. Так происходит до тех пор, пока не останется массив из двух элементов, для которого оптимальный код известен. После этого в обратном порядке строятся оптимальные коды для трёх, четырёх и т. д. элементов. Заметим, что при этом массив вероятностей P уже не нужен — нужна только последовательность номеров кодов, которые должны быть изъяты из массива кодов и продублированы в конце с добавлением разряда. А эта последовательность хранится в экземплярах локальной переменной j, соответствующих рекурсивным вызовам процедуры Huffman. □

Пример. Построение оптимального кода Хаффмена для $n = 8$. В левой части таблицы показано изменение массива P, а в правой части — массива C. Позиция, соответствующая текущему значению переменной j, выделена полужирным начертанием шрифта.

p_i												C	l_i	$p_i l_i$
0,31	0,31	0,31	0,31	0,31	**0,41**	**0,59**	**0**	**1**	1	00	00	00	2	0,62
0,24	0,24	0,24	0,24	**0,28**	0,31	0,41	1	00	**01**	10	10	10	2	0,48
0,17	0,17	0,17	0,17	0,24	0,28		01	000	11	11	11	11	2	0,34
0,11	0,11	0,11	**0,17**	0,17			001	**010**	011	011	011	011	3	0,33
0,09	0,09	0,09	0,11				011	0100	0100	0100	0100	0100	4	0,36
0,05	0,05	**0,08**					**0101**	01010	01010	01010	01010	01010	5	0,25
0,02	**0,03**						**01011**	010110	010110	010110	010110	010110	6	0,12
0,01							010111	010111	010111	010111	010111	010111	6	0,06
														2,56

Цена кодирования составляет $2,56$, что несколько лучше, чем в кодировании, полученном алгоритмом Фано.

6.3. Помехоустойчивое кодирование

Надёжность электронных устройств по мере их совершенствования всё время возрастает, но тем не менее в их работе возможны как систематические, так и случайные ошибки. Сигнал в канале связи может быть искажён помехой, поверхность магнитного носителя может быть повреждена, в разъёме может быть потерян контакт и т. п. Ошибки аппаратуры ведут к искажению или потере передаваемых или хранимых данных. При определённых условиях, некоторые из которых рассматриваются в этом разделе, можно применять методы кодирования, позволяющие правильно декодировать исходное сообщение, несмотря на ошибки в данных кода. В качестве исследуемой модели достаточно рассмотреть *канал связи с помехами*, потому что к этому случаю легко сводятся остальные. Например, запись на диск можно рассматривать как передачу данных в канал, а чтение с диска — как приём данных из канала.

6.3.1. Кодирование с исправлением ошибок

Пусть имеется канал связи, содержащий источник помех. Помехи проявляются в том, что принятое сообщение может отличаться от переданного, то есть имеется отношение $C \subset K \times K'$, $K, K' \subset B^*$, где K — множество переданных, а K' — соответствующее множество принятых по каналу сообщений. Используя запись $s' = C(s)$, $s \in K$, $s' \in K'$, подразумеваем, что разные «вызовы» отношения C с одним и тем же аргументом могут возвращать различные результаты. Кодирование $F\colon A^* \to B^*$ (вместе с декодированием $F^{-1}\colon B^* \to A^*$), обладающее тем свойством, что

$$S \xrightarrow{F} K \xrightarrow{C} K' \xrightarrow{F^{-1}} S,\ \forall\, s \in S \subset A^*\ \left(F^{-1}(C(F(s))) = s\right),$$

называется *помехоустойчивым*, или *самокорректирующимся*, или *кодированием с исправлением ошибок*. Без ограничения общности принимаем, что $A = B = \{0,1\}$. Кроме того, естественно предположить, что содержательное кодирование (вычисление F и F^{-1}) выполняется на устройстве, свободном от помех, то есть F и F^{-1} являются функциями в обычном смысле.

ЗАМЕЧАНИЕ

Функция F^{-1} является декодированием для кодирования F, но она *не* является обратной функцией для функции F в смысле определений пп. 1.4.3, 1.6.2.

Вообще говоря, ошибки в канале могут быть следующих типов:

$0 \mapsto 1,\ 1 \mapsto 0$ — ошибка типа замещения разряда;
$0 \mapsto \varepsilon,\ 1 \mapsto \varepsilon$ — ошибка типа выпадения разряда;
$\varepsilon \mapsto 1,\ \varepsilon \mapsto 0$ — ошибка типа вставки разряда.

Если про источник помех C ничего не известно, то функции F и F^{-1} не могут быть определены, за исключением тривиального случая $S = \varnothing$.

Пример. Если канал физически разорвать, то все разряды выпадут, и никакое декодирование невозможно.

Таким образом, для решения поставленной задачи необходимо иметь описание возможных ошибок (проявлений помех).

Вообще говоря, помехи могут иметь различные характеристики в зависимости от физической природы канала:

ошибки могут зависеть от самого передаваемого сообщения;
ошибки могут случайно возникать с некоторой вероятностью;
ошибки могут определяться физическим устройством канала.

Здесь рассматривается простейшая математическая модель канала, практически не зависящая от его физической природы. Предполагается, что:

1) сообщения передаются *блоками* определённой длины n, причём ошибки в различных блоках независимы;
2) значения разрядов 0 и 1 равноправны: если возможна ошибка $0 \mapsto 1$, то возможна и ошибка $1 \mapsto 0$;
3) для каждого типа ошибок известна верхняя граница количества ошибок в блоке длины n.

Общая *характеристика* ошибок канала (то есть их количество и типы) обозначается $\delta = \langle \delta_1, \delta_2, \delta_3 \rangle$, где $\delta_1, \delta_2, \delta_3$ — верхние оценки количества ошибок замещения, выпадения, вставки разряда соответственно.

Пример. Допустим, что имеется канал с характеристикой $\delta = \langle 1, 0, 0 \rangle$, то есть в канале возможно не более одной ошибки типа замещения разряда при передаче блока длины n. Кодирование: $F(a) := aaa$ (то есть каждый разряд в сообщении утраивается) и декодирование $F^{-1}(abc) := a + b + c > 1$ (то есть разряд восстанавливается методом «голосования») кажется помехоустойчивым для данного канала. Однако метод голосования очень расточителен (потребуется передать три блока вместо одного) и не вполне надёжен — если длина блока n не кратна трём, то граница блоков пройдёт по коду одного разряда, и если ошибка произойдёт в последнем разряде первого блока и в соседнем первом разряде второго блока, то восстановление пройдёт неверно.

Таким образом, при построении помехоустойчивого кодирования необходимо найти баланс между надёжностью и эффективностью.

ОТСТУПЛЕНИЕ

Следует различать *обнаружение ошибок* и *исправление ошибок*. Например, ранее при передаче данных широко использовался *бит чётности*, то есть дополнительный разряд, который добавлялся к каждому блоку так, чтобы общее количество разрядов, имеющих значение 1, было

чётным (или нечётным), то есть сумма разрядов блока по модулю два равна 0 (или 1). Очевидно, что этот простой метод позволяет обнаружить одиночную ошибку типа замещения, но не позволяет её исправить. В настоящее время бит чётности применяется реже, поскольку имеет невысокую помехоустойчивость: если в одном блоке произойдут две ошибки типа замещения, то они не будут обнаружены.

Пример. Пусть $\delta = \langle 1, 0, 0\rangle$ для слов длины n. Метод голосования, рассмотренный в примере предыдущего подраздела, является кодированием с исправлением одной ошибки типа замещения, но не является кодированием с исправлением всех ошибок при $n > 1$.

6.3.2. Возможность исправления всех ошибок

Рассмотрим множество сообщений $S \subset \{0,1\}^*$, кодирование $F\colon \{0,1\}^* \to \{0,1\}^*$ и канал C с характеристикой δ. Пусть $E_F^\delta(s)$ — множество слов, которые могут быть получены из сообщения $s \in S$ в результате кодирования F и всех возможных комбинаций, допустимых в канале ошибок типа δ.

Пример. Пусть $S = \{00, 11\}$, кодирование F добавляет к сообщению третий разряд (бит чётности), а $\delta = \langle 1, 0, 0\rangle$ для блоков длины 3. Тогда

$$F(00) = 000,\ E_F^\delta(00) = \{000, 001, 010, 100\},\ F(11) = 110,\ E_F^\delta(11) = \{110, 111, 100, 010\}.$$

Имеем $E_F^\delta(00) \cap E_F^\delta(11) = \{100, 010\}$, и если из канала получен один из кодов 100 или 010, то однозначно определить исходное сообщение невозможно. Если же в качестве третьего разряда при кодировании взять конъюнкцию, то

$$F(00) = 000,\ E_F^\delta(00) = \{000, 001, 010, 100\},\ F(11) = 111,\ E_F^\delta(11) = \{111, 110, 101, 011\}.$$

При этом $E_F^\delta(00) \cap E_F^\delta(11) = \varnothing$ и исходное сообщение определяется однозначно.

ЗАМЕЧАНИЕ

Множество сообщений S является существенным фактором. Например, если в условиях предыдущего примера множество сообщений $S = \{01, 10\}$, то, как нетрудно проверить, никакое добавление третьего разряда не является помехоустойчивым кодированием.

ТЕОРЕМА. *Чтобы существовало помехоустойчивое кодирование F с исправлением всех ошибок типа δ, необходимо и достаточно, чтобы*

$$\forall\, s_1, s_2 \in S\ \left(E_F^\delta(s_1) \cap E_F^\delta(s_2)\right) = \varnothing.$$

Доказательство.

[$\Longrightarrow$] Если кодирование помехоустойчивое, то $E_F^\delta(s_1) \cap E_F^\delta(s_2) = \varnothing$, иначе F^{-1} невозможно было бы определить как функцию.

[$\Longleftarrow$] Из условия $\forall\, s_1, s_2 \in S\ \left(E_F^\delta(s_1) \cap E_F^\delta(s_2) = \varnothing\right)$ по теореме п. 1.7.1 следует, что существует разбиение $\mathcal{B} = \{B_s\}_{s\in S}$ множества $\{0,1\}^*$, причём $\forall\, s \in S\ \left(E_F^\delta(s) \subset B_s\right)$. По разбиению $\mathcal{B}$ требуемая функция $F^{-1}\colon \{0,1\}^* \to S$ строится следующим образом: **if** $s' \in B_s$ **then** $F^{-1}(s') := s$ **end if**. □

Таким образом, сама возможность исправления ошибок зависит от трёх факторов: множества сообщений, применяемого кодирования и характеристики канала. Как правило, множество сообщений и характеристика канала считаются заданными, и задача состоит в том, чтобы подобрать по возможности эффективное помехоустойчивое кодирование. Однако в некоторых практических случаях исправление *всех* ошибок невозможно.

Пример. Рассмотрим канал, в котором в любом передаваемом разряде может происходить ошибка типа замещения с вероятностью p, причём замещения различных разрядов статистически независимы. Такой канал называется *двоичным симметричным*. В этом случае любое слово может быть преобразовано в любое другое слово той же длины замещениями разрядов. Таким образом, исправить *все* ошибки в двоичном симметричном канале невозможно даже при сколь угодно малом $p > 0$.

ОТСТУПЛЕНИЕ

Для различных моделей со случайными ошибками (в том числе для двоичного симметричного канала) средствами теории вероятностей рассматриваются и решаются, в частности, такие задачи, как:

1) определение максимального количества информации, которая может быть передана через канал при наличии помех (*пропускная способность канала*);
2) определение математического ожидания вероятности исправления ошибок в канале (*достоверность декодирования*);
3) определение минимального количества информации, которую необходимо передать через канал с сохранением возможности последующего использования данных (*сжатие с потерями*).

Все эти вопросы являются предметом *теории информации*, которая здесь не рассматривается.

Рассмотрим возможность исправления всех ошибок с практической точки зрения. Пусть в канале возможны только ошибки замещения разрядов (выпадение и вставка разрядов исключены), сообщения передаются блоками длины n, и в каждом разряде возможна ошибка с вероятностью p. Это допущение соответствует устройству шины передачи данных в современном компьютере. Тогда вероятность того, что произойдёт ровно k ошибок, равна $v = p^k(1-p)^{n-k}$. Рассмотрим численное значение этой величины при некоторых типичных значениях величин n, k и p.

n	k	p	v	n	k	p	v
32	1	10^{-03}	$9,695 \cdot 10^{-04}$	64	1	10^{-03}	$9,389 \cdot 10^{-04}$
32	1	10^{-06}	$1,000 \cdot 10^{-06}$	64	1	10^{-06}	$9,999 \cdot 10^{-07}$
32	2	10^{-03}	$9,704 \cdot 10^{-07}$	64	2	10^{-03}	$9,399 \cdot 10^{-07}$
32	3	10^{-03}	$9,714 \cdot 10^{-10}$	64	3	10^{-03}	$9,408 \cdot 10^{-10}$
32	2	10^{-06}	$1,000 \cdot 10^{-12}$	64	2	10^{-06}	$9,999 \cdot 10^{-13}$
32	3	10^{-06}	$1,000 \cdot 10^{-18}$	64	3	10^{-06}	$9,999 \cdot 10^{-19}$

Таким образом, возможностью одновременного возникновения ошибок большой кратности можно пренебречь, и достаточно рассматривать только каналы с ограниченной характеристикой $\delta << n$, для которых задача исправления всех ошибок оказывается разрешимой.

6.3.3. Расстояния Левенштейна и Хэмминга

Неотрицательная функция $d(x,y)\colon M\times M\to\mathbb{R}_+$ называется *расстоянием* (или *метрикой*) на множестве M, если выполнены три условия, которые называются *аксиомами метрики*.

1. *Аксиома тождества*: $d(x,y)=0\iff x=y$.
2. *Аксиома симметрии*: $d(x,y)=d(y,x)$.
3. *Неравенство треугольника*: $d(x,y)\leqslant d(x,z)+d(y,z)$.

Пример. Функция $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}$ является метрикой и называется *евклидовым расстоянием* между точками (x_1,y_1,z_1) и (x_2,y_2,z_2) в трёхмерном пространстве $\mathbb{R}^3$.

Множество $\{y\mid d(x,y)\leqslant p\}$ называется *шаром*, или *метрической окрестностью*, с центром в x и радиусом p.

Ошибки замещения, выпадения и вставки символов можно рассматривать как операции с двоичными словами. Применяя эти операции, возможно одно слово $\alpha\in\{0,1\}^*$ преобразовать в другое слово $\beta\in\{0,1\}^*$.

Пример. $01\xrightarrow{0\mapsto1}11\xrightarrow{1\mapsto0}10$; $01\xrightarrow{0\mapsto\varepsilon}1\xrightarrow{\varepsilon\mapsto0}10$; $01\xrightarrow{\varepsilon\mapsto0}010\xrightarrow{0\mapsto\varepsilon}10$.

Минимальное число операций, переводящее слово α в слово β, называется *расстоянием Левенштейна* и обозначается $L(\alpha,\beta)$.

Пример. $L(01,10)=2$.

ЛЕММА. *Расстояние Левенштейна L является метрикой.*

Доказательство.

[1] $L(\alpha,\beta)=0\iff\alpha=\beta$, поскольку преобразование отсутствует тогда и только тогда, когда слова совпадают.

[2] $L(\alpha,\beta)=L(\beta,\alpha)$, поскольку ошибки симметричны и, значит, преобразования слов обратимы.

[3] $L(\alpha,\beta)\leqslant L(\alpha,\gamma)+L(\beta,\gamma)$, поскольку конкатенация последовательностей преобразования $\alpha\mapsto\gamma$ и $\gamma\mapsto\beta$ является *некоторой* последовательностью, преобразующей α в β, а $L(\alpha,\beta)$ является длиной кратчайшей из таких последовательностей. □

ЗАМЕЧАНИЕ

Расстояние Левенштейна нетрудно определить для слов в произвольном алфавите, а не только в алфавите $\{0,1\}$. В таком случае это расстояние часто называют *дистанцией редактирования*.

ОТСТУПЛЕНИЕ

На операции редактирования могут быть наложены различные ограничения. При этом, если преобразования слов остаются обратимыми, а конкатенация преобразований является преобразованием, то расстояние Левенштейна остаётся метрикой. Например, в популярной интеллектуальной игре требуется преобразовать одно слово в другое, заменяя по одной букве. Тогда $3\leqslant L(\text{коза},\text{волк})\leqslant5$, поскольку необходимо заменить три буквы, и есть решение коза↦поза↦пола↦полк↦волк.

Пусть некоторый канал имеет характеристику $\delta = \langle \delta_1, \delta_2, \delta_3 \rangle$. Пусть $E^\delta(\alpha)$ — множество слов, которые могут быть получены из слова α в результате всех возможных комбинаций допустимых в канале ошибок типа δ. Поскольку рассматриваются симметричные ошибки, ясно, что $\beta \in E^\delta(\alpha) \iff \alpha \in E^\delta(\beta)$.

Положим $H^\delta(\alpha, \beta) :=$ **if** $\alpha \in E^\delta(\beta)$ **then** $L(\alpha, \beta)$ **else** $+\infty$ **end if**.

Функция H^δ называется *расстоянием Хэмминга*[1]. Расстояние Хэмминга является метрикой по тем же причинам, что и расстояние Левенштейна, а потому множество $E^\delta(\alpha)$ уместно называть *δ-окрестностью* слова α.

Пример. Расстояние Хэмминга в E_2^n. Пусть $\delta = \langle n, 0, 0 \rangle$, то есть допускаются только ошибки типа замещения разрядов. Рассмотрим слова $\alpha = a_1 \dots a_n$ и $\beta = b_1 \dots b_n$. Тогда $H^\delta(\alpha, \beta) \stackrel{\text{Def}}{=} \sum\limits_{i=1}^{n} (a_i \neq b_i)$. То есть расстоянием Хэмминга между битовыми шкалами одной длины является число несовпадающих разрядов. Именно этот частный случай часто называют расстоянием Хэмминга в E_2^n и обозначают H^n.

ЗАМЕЧАНИЕ

В последней формуле, так же как и в других местах этой книги, используется неявное приведение **false** $\mapsto 0$, **true** $\mapsto 1$.

6.3.4. Кодовое расстояние

Рассмотрим применение введённых понятий к простейшему случаю алфавитного кодирования. Пусть $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ — схема алфавитного кодирования, а d — некоторая метрика на B^*. Тогда минимальное расстояние между элементарными кодами $d(\sigma) \stackrel{\text{Def}}{=} \min\limits_{1 \leqslant i < j \leqslant n} d(\beta_i, \beta_j)$ называется *кодовым расстоянием* схемы σ в метрике d.

ТЕОРЕМА. *Алфавитное кодирование со схемой $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^n$ и с кодовым расстоянием Хэмминга $H(\sigma) = \min\limits_{\beta', \beta'' \in V} H^\delta(\beta', \beta'')$ является кодированием с исправлением h ошибок типа δ тогда и только тогда, когда $H(\sigma) > 2h$.*

Доказательство. Если можно исправить ошибки в каждом из кодовых слов, то можно исправить ошибки и во всем сообщении. Поэтому достаточно рассматривать только кодовые слова. Заметим, что множество $E^\delta(\alpha)$ — это шар радиуса h с центром в α. Таким образом, $E^\delta(\beta_i) \cap E^\delta(\beta_j) = \varnothing \iff H(\sigma) > 2h$. По теореме п. 6.3.2 условие $E^\delta(\beta_i) \cap E^\delta(\beta_j) = \varnothing$ равносильно существованию декодирования. □

Другими словами, исправление ошибок типа δ возможно тогда и только тогда, когда кодовые слова схемы не лежат в δ-окрестностях друг друга.

СЛЕДСТВИЕ. *Схема равномерного кодирования $\sigma = \langle a_i \to \beta_i \rangle_n^{i=1}$, где $\forall\, i\ (|\beta_i| = m)$, $m \geqslant \lfloor \log_2(n-1) \rfloor + 1$, может исправлять h ошибок типа замещения в m разрядах, если и только если $\min\limits_{1 \leqslant i < j \leqslant n} \left(\sum\limits_{k=1}^{m} \beta_i[k] \neq \beta_j[k] \right) > 2h$, то есть любые два кодовых слова различаются не менее чем в $2h$ разрядах.*

[1] Ричард Весли Хэмминг (1915–1998).

Пример. На рисунке 6.1 представлен гиперкуб для пространства E_2^3 и две 1-окрестности точек 101 и 010 (то есть два шара радиуса 1 с центрами в точках 101 и 010). Очевидно, что можно взять два любых инверсных кода, и шары с центрами в этих кодах не пересекутся, но все другие шары радиуса 1 с центрами в других точках пересекутся с обоими выделенными шарами. Из этого наблюдения и теоремы следует, что трёхразрядный код может исправлять одну ошибку типа замещения для алфавитного кодирования только двухбуквенного алфавита.

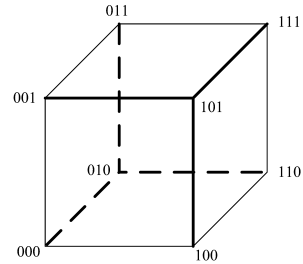


Рис. 6.1. Гиперкуб

6.3.5. Мощность кодирования

Мощностью кодирования называется мощность множества сообщений S, которые удаётся закодировать с выполнением заданных условий. Чем мощнее кодирование, тем больше различных сообщений возможно закодировать с его помощью.

Из теоремы п. 6.3.2 следует, что помехоустойчивое кодирование максимальной мощности достигается в том случае, когда разбиение множества принятых сообщений имеет максимальную мощность.

Пример. Помехоустойчивое кодирование с исправлением не более одной ошибки типа замещения в блоке длины 3 имеет мощность не более 2.

Рассмотрим случай канала, в котором возможно не более h ошибок замещения разрядов в блоках длины n. Какова максимальная возможная мощность помехоустойчивого кодирования? В рассматриваемом случае в метрическом пространстве E_2^n с расстоянием Хэмминга H^n множество возможных ошибок в сообщении s определяется как шар радиуса h с центром в коде сообщения $F(s)$:

$$E_F^{\langle h,0,0\rangle}(s) = V^n(F(s),h), \quad \text{где} \quad V^n(x,h) \stackrel{\text{Def}}{=} \{y \in E_2^n \mid H^n(x,y) \leqslant h\}.$$

ЛЕММА. $\forall\, x \in E_2^n \left(\forall\, h \geqslant 0 \left(|V^n(x,h)| = \sum\limits_{i=0}^{h} C(n,i)\right)\right)$.

Доказательство. $V^n(x,h) = \{y \in E_2^n \mid H^n(x,y) \leqslant h\} =$
$= \bigcup\limits_{i=0}^{h} \{y \in E_2^n \mid H^n(x,y) = i\}$, $|\{y \in E_2^n \mid H^n(x,y) = i\}| = C(n,i)$. □

ЗАМЕЧАНИЕ

Доказанная лемма обобщается на любое дискретное метрическое пространство, в частности, пространство слов над любым алфавитом с расстоянием Левенштейна.

ТЕОРЕМА. *Мощность помехоустойчивого кодирования для канала, в котором воз-*

можно не более h ошибок замещения разрядов в блоках длины n, не превосходит числа шаров, которые можно уложить в пространстве E_2^n без пересечений:

$$|S| \leqslant \frac{2^n}{\sum\limits_{i=0}^{h} C(n,i)}.$$

Доказательство. Для каждого сообщения множество возможных ошибок является шаром радиуса h, для помехоустойчивости шары не должны пересекаться, а всего в пространстве 2^n точек. □

Пример. В следующей таблице показаны верхние оценки для мощности помехоустойчивовго кодирования в зависимости от h и n.

$h \setminus n$	2	3	4	5	6	7	8	9	10	11	12
0	4	8	16	32	64	128	256	512	1024	2048	4096
1	1	2	3	5	9	16	28	51	93	170	315
2	1	1	1	2	2	4	6	11	18	30	51
3	1	1	1	2	2	2	3	5	7	11	17

ЗАМЕЧАНИЕ

Случай $h = 0$ соответствует максимальной мощности кодирования при отсутствии ограничений. Такое кодирование называется *безызбыточным*.

Специфические особенности множества сообщений S можно использовать при построении кодирования, однако это требует специальных ухищрений и не является универсальным решением. Для практики наиболее важен случай $S = E_2^m$, $|S| = 2^m$, то есть сообщением может быть любое слово длины m.

Следствие. *Если $S = E_2^m$ и существует помехоустойчивое кодирование для канала, в котором возможно не более одной ошибки замещения в блоке длины n, то*

$$2^m \leqslant \frac{2^n}{n+1}.$$

Доказательство. По формуле теоремы $C(n,0) + C(n,1) = 1 + n$. □

Помехоустойчивое кодирование не может быть безызбыточным.

Обозначим $k := n - m$ — минимальное количество *контрольных разрядов*, то есть избыточных разрядов, необходимых для помехоустойчивости.

Пример. В следующей таблице показано соотношение для m, n и k при $h = 1$. Здесь $n' := \lfloor 2^n/(n+1) \rfloor$. Для $n = 1, 2$ помехоустойчивого кодирования не существует.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
n'	2	3	5	9	16	28	51	93	170	315	585	1092	2048	3855
2^m	2	2	4	8	16	16	32	64	128	256	512	1024	2048	2048
m	1	1	2	3	4	4	5	6	7	8	9	10	11	11
k	2	3	3	3	3	4	4	4	4	4	4	4	4	5

Можно заметить, например, что 7-разрядные блоки передают столько же информации и с такой же надёжностью, что и 8-разрядные, аналогичная ситуация для 15-разрядных и 16-разрядных кодов и т. д.

Доля контрольных разрядов, необходимых для существования помехоустойчивого кодирования, начиная с $n = 6$, не превышает половины всех разрядов блока и с ростом n постепенно уменьшается. В частности, при $n \in 27..57$ имеем $k = 6$, а при $n = 64$ имеем $k = 7$.

6.3.6. Виды кодирования

Предложено множество разнообразных способов помехоустойчивого кодирования. Все эти способы подчиняются выведенным количественным ограничениям, но отличаются используемым математическим аппаратом, удобством применения, наглядностью и другими качественными характеристиками.

Пример. Пусть $m = 2, h = 1$. Тогда минимально $n = 5, k = 3$. Можно *подобрать* подходящие четыре пятиразрядных кода так, чтобы их 1-окрестности в метрике Хэмминга не пересекались. В частности, с помощью карты Карно (п. 3.5.5), как показано на рис. 6.2. Выбранные коды (центры шаров) выделены полужирным начертанием и подчёркнуты, сами шары выделены различной штриховкой и заливкой.

	000	001	011	010	110	111	101	100
00	00000	**00001**	00011	00010	00110	00111	00101	00100
01	01000	01001	01011	**01010**	01110	01111	01101	01100
11	11000	11001	11011	11010	11110	**11111**	11101	11100
10	10000	10001	10011	10010	10110	10111	10101	**10100**

Рис. 6.2. Карта Карно

Затем можно произвольно назначить коды сообщениям: $00 \mapsto 00001$, $01 \mapsto 01010$, $11 \mapsto 11111$, $10 \mapsto 10100$. Декодирование можно реализовать с помощью дерева решений (п. 3.5.8), как показано на рис. 6.3.

Варианты с пометкой «err» означают появление ошибки, при которой произошло более одного замещения разряда. Такую ошибку данное кодирование исправить не может, но может обнаружить.

ЗАМЕЧАНИЕ

Выбор системы непересекающихся шаров в E_2^n, назначения кодов словам из E^m является произвольным. Если сохранить этот выбор в тайне, то кодирование, наряду с помехоустойчивостью, станет обладать и свойствами тайнописи.

Среди пестроты методов помехоустойчивого кодирования своей простотой и ясностью выделяется группа методов, основанных на применении математического аппарата и обозначений линейной алгебры.

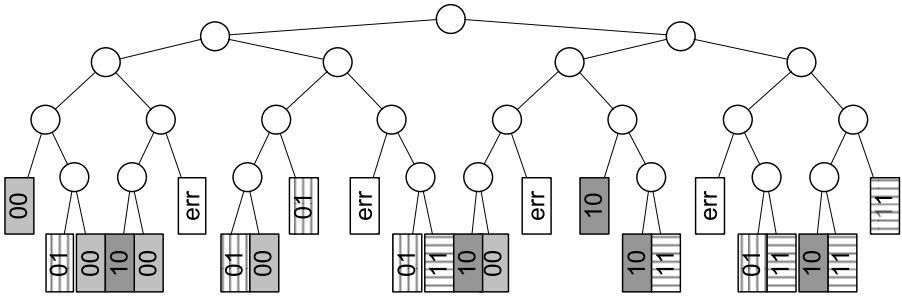


Рис. 6.3. Дерево решений

Напомним, что двоичная арифметика $\langle E_2; +_2, \& \rangle$ является полем (п. 2.3.3) относительно сложения по модулю 2 и конъюнкции, множество битовых шкал E_2^n образует абелеву группу (п. 2.2.6) относительно операции поразрядного логического сложения шкал и, как нетрудно проверить, множество битовых шкал E_2^n образует векторное пространство (п. 2.5.1) над двоичной арифетикой.

ЗАМЕЧАНИЕ

В контексте этого раздела знак + означает сложение по модулю 2, а знак умножения опускается, что позволяет использовать обычные обозначения линейной алгебы, имея в виду, что все векторы и матрицы — булевы, и операции выполняются по правилам, изложенным в п. 1.4.9. Размеры матриц и векторов, если нужно, указываются в нижнем индексе.

Введем обозначения: $\mathbf{0}_m$ означает вектор из m нулей, $\mathbf{0}_{m\times n}$ означает матрицу размера $m \times n$ из нулей, а I_m означает единичную матрицу размера $m \times m$.

Рассмотрим простейший случай кодирования, когда исходными сообщениями являются все слова длины m, а кодами являются блоки длины n, $n \geqslant m$. Пусть $\alpha = a_1 \ldots a_m \in E_2^m$ — исходное сообщение, а $\beta = b_1 \ldots b_n \in E_2^n$ — кодовый блок. Рассмотрим случай, когда каждый разряд β можно представить в виде *линейной* функции от разрядов α (п. 3.6.1). Тогда кодирование записывается следующим образом: $\forall\, j \in 1..n\ \ (b_j = \sum_{i=1}^{m} a_i g_{i,j})$. Используя обозначения линейной алгебры, код β можно представить в виде произведения вектора α и булевой матрицы $G_{m\times n}$, составленной из коэффициентов $g_{i,j}$, или $\beta = \alpha G$. Матрица G называется *порождающей матрицей* кода.

Пример. Рассмотрим сообщение $\alpha = (0 \quad 1 \quad 1)$. Пусть матрица $G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$. Тогда $\beta = (0 \quad 1 \quad 1) \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} =$ $= (0 \quad 1 \quad 1 \quad 1 \quad 0)$. Здесь $m = 3, k = 2, n = 5$.

Если $\exists\, G_{n\times m}^{-1}\ \ \big(GG^{-1} = I_m\big)$, то $\beta G^{-1} = (\alpha G)G^{-1} = \alpha(GG^{-1}) = \alpha I_m = \alpha$, и декодирование выполняется умножением кода на матрицу G^{-1}.

ТЕОРЕМА. *Если* $\exists\, H_{k\times n}\ \big(GH^{\mathrm{T}} = \mathbf{0}_{m\times k}\big)$, *то* $\beta = \alpha G \iff \beta H^{\mathrm{T}} = \mathbf{0}_k$.

Доказательство. $\beta = \alpha G \iff \beta H^{\mathrm{T}} = \alpha G H^{\mathrm{T}} = \alpha \mathbf{0}_{m\times k} = \mathbf{0}_k$. □

Матрица H называется *проверочной*, поскольку с помощью равенства $\beta H^{\mathrm{T}} = \mathbf{0}_k$ позволяет проверить, является ли принятый через канал код β действительно кодом некоторого сообщения α при кодировании G.

Пример. Матрица $H = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$ является проверочной для матрицы G предыдущего примера: $GH^{\mathrm{T}} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$.

ОТСТУПЛЕНИЕ

В терминах линейной алгебры, если строки порождающей матрицы G линейно независимы, то коды различных сообщений образуют линейное подпространство векторного пространства E_2^n. В этом случае проверочная матрица H задаёт ортогональное дополнение векторного пространства, заданного матрицей G. Все положения распространяются на случай векторных пространств над любым конечным полем, а не только E_2. Использование понятий линейной алгебры (в том числе понятий, выходящих за рамки изложенного в главе 2) позволяет сделать описание кодирования лаконичным и общим, но не меняет алгоритмической сути дела.

6.3.7. Систематическая форма кодовых сообщений

Известен очень простой способ определения матриц G, G^{-1} и H, который из-за своей формы называется *систематическим кодированием*. При систематическом кодировании блок из n разрядов состоит из m *информационных разрядов*, за которыми следуют $k = n - m$ *проверочных разрядов* (или *контрольных разрядов*), причём информационные разряды копируются в код без изменения, а проверочные разряды вычисляются как линейные функции от информационных разрядов.

В этих соглашениях порождающая матрица имеет вид $G = [I_m | P_{m\times k}]$, где I_m — единичная матрица, формирующая информационные разряды, а $P_{m\times k}$ — матрица, определяющая проверочную часть.

ТЕОРЕМА. *Если* $G = [I_m | P_{m\times k}]$, *то:*

1) $G^{-1}_{n\times m} := [I_m | \mathbf{0}_{m\times k}]^{\mathrm{T}}$ *и при этом* $GG^{-1} = I_m$;
2) $H_{k\times n} := \left[P^{\mathrm{T}}_{k\times m} | I_k\right]$ *и при этом* $GH^{\mathrm{T}} = \mathbf{0}_{m\times k}$.

Доказательство.

[1] Положим $C_{m\times m} := GG^{-1}$. Имеем $G_{i,l} =$ **if** $i = l$ **then** 1 **else** 0 **end if** или, короче, $G_{i,l} = (i = l)$ при $1 \leqslant i, l \leqslant m$, и также $G^{-1}_{l,j} = (l = j)$.

Тогда $C_{i,j} = \sum\limits_{l=1}^{n} G_{i,l} G^{-1}_{l,j} = (i = j)$, то есть $C = I_m$.

[2] Положим $C_{m\times k} := GH^{\mathrm{T}}$. Имеем $G_{i,l} = (i = l)$ при $1 \leqslant i, l \leqslant m$, $H^{\mathrm{T}}_{l,j} = (l - m = j)$ при $m + 1 \leqslant l \leqslant m + k, 1 \leqslant j \leqslant k$, а также $G_{i,l} = P_{i,l-m}$ при $l > m$

и $H^{\mathrm{T}}_{l,j} = P_{l,j}$ при $1 \leqslant l \leqslant m$.

Тогда $C_{i,j} = \sum\limits_{l=1}^{n} G_{i,l} H^{\mathrm{T}}_{l,j} = \sum\limits_{l=1}^{m} G_{i,l} H^{\mathrm{T}}_{l,j} + \sum\limits_{l=m+1}^{n} G_{i,l} H^{\mathrm{T}}_{l,j} = P_{i,j} + P_{i,j} = 0.$ □

ЗАМЕЧАНИЕ

Напомним, что матрицы булевы, и в операциях с матрицами знак + означает сложение по модулю 2.

Систематический код с параметрами n и m обозначается как (n, m)-код.

Систематическая форма кодовых сообщений позволяет удобно кодировать и декодировать сообщения, а также обнаруживать некоторые ошибки, но не гарантирует исправления ошибок.

Пример. В условиях предыдущих примеров $G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$, $H = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$. Пусть $\alpha = (0 \quad 1 \quad 1)$. Тогда $\beta = \alpha G = (0 \quad 1 \quad 1 \quad 1 \quad 0)$. Пусть теперь произошла ошибка в третьем разряде, и из канала принято слово $\beta' = (0 \quad 1 \quad 0 \quad 1 \quad 0)$. Тогда $\beta' G^{-1} = (0 \quad 1 \quad 0)$ и сообщение декодировано неверно, но $\beta' H^{\mathrm{T}} = (0 \quad 1 \quad 0 \quad 1 \quad 0) \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = (1 \quad 1)$ и ошибка обнаружена.

ЗАМЕЧАНИЕ

По теореме п. 6.3.5 кодирование с параметрами $m = 3$, $k = 2$, $n = 5$, рассматриваемое в примере, не может быть помехоустойчивым. Если $m = 3$, то необходимо $k \geqslant 3$.

6.3.8. Коды Хэмминга

При помехоустойчивом кодировании основная задача состоит в том, чтобы определить, какую дополнительную информацию надо передать, чтобы её объём был минимальным, а декодирование по возможности несложным.

Видимо, наиболее известным систематическим самокорректирующимся кодом в настоящее время является *код Хэмминга*.

Введём обозначения. Пусть $\alpha \in E_2^m$ — исходный блок, $\beta \in E_2^n$ — кодовый блок, полученный с помощью порождающей матрицы $G_{m\times n}$, для которой известна проверочная матрица $H_{k\times n}$, такая, что $GH^{\mathrm{T}} = \mathbf{0}_{m\times k}$. Пусть $\beta' \in E_2^n$ — блок с возможной ошибкой, полученный через канал, а $\epsilon \in E_2^n$ — *битовая шкала ошибок*, которая содержит 1 там, где произошла ошибка, и 0 в остальных разрядах. Результат применения проверочной матрицы к блоку с ошибкой называется *синдромом* и обозначается $\zeta := \beta' H^{\mathrm{T}}$.

ТЕОРЕМА 1. *Синдром зависит только от битовой шкалы ошибок:* $\zeta = \epsilon H^{\mathrm{T}}$.

Доказательство. По определению $\beta' = \beta + \epsilon$, где $+$ означает поразрядное сложение битовых шкал по модулю 2. Далее $\zeta = \beta' H^{\mathrm{T}} = (\beta + \epsilon) H^{\mathrm{T}} = \beta H^{\mathrm{T}} + \epsilon H^{\mathrm{T}} = \epsilon H^{\mathrm{T}}$, поскольку $\beta H^{\mathrm{T}} = \mathbf{0}$. □

Отсюда вытекает метод кодирования и декодирования с исправлением одиночной ошибки замещения разряда в блоке длины n.

1. По формулам или таблицам п. 6.3.5 подобрать число m так, чтобы $2^m \leqslant 2^n/(n+1)$ и положить $k := n - m$.
2. Определить булевы матрицы: порождающую матрицу $G_{m\times n}$, строки которой линейно независимы; матрицу декодирования $G^{-1}_{n\times m}$, такую что $GG^{-1} = I_m$; проверочную матрицу $H_{k\times n}$, такую что $GH^{\mathrm{T}} = \mathbf{0}_{m\times k}$. Все эти матрицы определяются неоднозначно.
3. Разбить исходное сообщение на блоки длины m.
4. Каждый блок α длины m независимо умножить на порождающую матрицу, получая блок $\beta := \alpha G$ длины n.
5. Блок β передать через канал, получая блок $\beta' = b_1 \ldots b_n$, возможно содержащий одну ошибку замещения.
6. Вычислить синдром $\zeta := \beta' H^{\mathrm{T}}$.
7. Если синдром $\zeta = \mathbf{0}_k$, то ошибок нет, а если $\zeta \neq \mathbf{0}_k$, то определить j — номер того столбца проверочной матрицы, который совпадает с синдромом, после чего инвертировать j-й разряд: $b_j := 1 + b_j$.
8. Декодировать блок $\alpha' := \beta' G^{-1}$.

ТЕОРЕМА 2. *В указанных обозначениях* $\alpha = \alpha'$.

Доказательство. Если ошибок не случилось и $\beta' = \beta$, то $\alpha' = \alpha$ по свойству матриц G и G^{-1}. Пусть произошла ошибка в разряде j. Известно, что $\zeta = \epsilon H^{\mathrm{T}}$. Поскольку битовая шкала ϵ содержит 1 в j-м разряде и 0 во всех остальных разрядах, синдром равен j-му столбцу матрицы H. Инверсия j-го разряда устраняет ошибку в коде и позволяет произвести обычное декодирование. □

ЗАМЕЧАНИЕ

Шаги 1 и 2 выполняются только один раз, и их трудоёмкость не критична. Прочие шаги выполняются для каждого блока, и скорость их выполнения важна. Особого внимания заслуживают шаги 6 и 7, поскольку это «накладные расходы» ради помехоустойчивости на кодирование и декодирование.

Основная идея кода Хэмминга состоит в том, чтобы в проверочной матрице H использовать к качестве j-го столбца двоичную запись номера этого столбца. Тогда вычисление синдрома сразу даёт номер разряда, в котором произошла ошибка. Если синдром равен нулю, то ошибки нет. Соотношение между n и k таково, что k разрядов как раз достаточно для записи чисел в диапазоне $1..n$.

Пример. Пусть $m = 4$, тогда $k = 3$ и $n = 7$. Проверочная матрица кода: $H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$. Пусть $\alpha = 1100$ и используется систематическое кодирование. Тогда $\beta = 1100 b_5 b_6 b_7$. Соотношения для проверочных символов получаем из

условия $\beta H^{\mathrm{T}} = \mathbf{0}_k$, то есть $\forall\, i \in 1..k \ \left(\sum_{j=1}^{n} b_j H_{j,i} = 0\right)$. Имеем $b_5 + b_6 + b_7 = 0$, $1 + b_6 + b_7 = 0$, $1 + b_5 + b_7 = 0$. Получаем $b_5 = 1$, $b_6 = 1$, $b_7 = 0$. Значит, $\beta = 1100110$. Пусть во время прохождения через канал произошла ошибка в третьем символе: $\beta' = 1110110$. Найдём синдром в данном случае:

$$\zeta = (1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0) \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} = (0 \quad 1 \quad 1)\,.$$

Имеем $j = 3$, в этом разряде как раз и произошла ошибка.

6.3.9. Исправление одного замещения

Существуют другие способы линейного кодирования, отличные от систематических кодов, при которых проверочные разряды находятся на позициях, облегчающих кодирование. Рассмотрим в качестве примера случай одного замещения. Определим последовательности натуральных чисел в соответствии с их представлениями в двоичной системе счисления следующим образом:
$V_1 := 1, 3, 5, 7, 9, 11, \ldots$ — все числа, у которых разряд 1 равен 1;
$V_2 := 2, 3, 6, 7, 10, \ldots$ — все числа, у которых разряд 2 равен 1;
$V_3 := 4, 5, 6, 7, 12, \ldots$ — все числа, у которых разряд 3 равен 1 и т. д.
По определению последовательность V_k начинается с числа 2^{k-1}. Также рассмотрим последовательности V'_k, где V'_k получается из V_k отбрасыванием первого элемента. Эти последовательности определены статически и могут быть вычислены заранее. Пусть заданы числа n, m, причём $2^m \leqslant 2^n/(n+1)$ и $k = n - m$. Рассмотрим k разрядов с номерами $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $\ldots 2^{k-1}$ в слове $b_1 \ldots b_n$. Эти разряды считаются контрольными. Остальные разряды, а их ровно m, считаются информационными. Поместим в информационные разряды все разряды слова $a_1 \ldots a_m$ как они есть. Контрольные разряды определим следующим образом:
$b_1 := b_3 + b_5 + b_7 + \ldots$, то есть сумма всех разрядов с номерами из V'_1;
$b_2 := b_3 + b_6 + b_7 + \ldots$, то есть сумма всех разрядов с номерами из V'_2;
$b_4 := b_5 + b_6 + b_7 + \ldots$, то есть сумма всех разрядов с номерами из V'_3;
и вообще, $b_{2^{j-1}} := \sum_{i \in V'_j} b_i$, причём сложение выполняется по модулю 2. Пусть после прохождения через канал получен код $c_1 \ldots c_n$ и пусть I — номер разряда, в котором, возможно, произошла ошибка замещения. Пусть это число имеет следующее двоичное представление: $I = i_l \ldots i_1$. Определим число $J = j_l \ldots j_1$ следующим образом:
$j_1 := c_1 + c_3 + c_5 + c_7 + \ldots$, т. е. сумма всех разрядов с номерами из V_1;
$j_2 := c_2 + c_3 + c_6 + c_7 + \ldots$, т. е. сумма всех разрядов с номерами из V_2;
$j_3 := c_4 + c_5 + c_6 + c_7 + \ldots$, т. е. сумма всех разрядов с номерами из V_3;
и вообще, $j_p := \sum_{q \in V_p} c_q$, где сложение выполняется по модулю 2.

ТЕОРЕМА. *В указанных обозначениях $I = J$.*

Доказательство. Эти числа равны, потому что поразрядно равны их двоичные представления. Действительно, пусть $i_1 = 0$. Тогда $I \notin V_1$ и, значит, $j_1 = c_1 + c_3 + {} + c_5 + \ldots = b_1 + b_3 + b_5 + \ldots = 0$ по определению разряда b_1. Пусть теперь $i_1 = 1$.

Тогда $I \in V_1$ и, значит, $j_1 = c_1 + c_3 + c_5 + \ldots = b_1 + b_3 + b_5 + \ldots + \overline{b_x} + \ldots = 1$, так как если в сумме по модулю 2 изменить ровно один разряд, то изменится и значение всей суммы. Итак, $i_1 = j_1$. Аналогично (используя последовательности V_2 и т. д.) имеем $i_2 = j_2$ и т. д. Таким образом, $I = J$. □

6.4. Сжатие и шифрование данных

Материал раздела 6.2 показывает, что при кодировании наблюдается некоторый баланс между временем и памятью. Затратив дополнительные усилия при кодировании и декодировании, можно сэкономить память, и наоборот, если пренебречь оптимальным использованием памяти, можно существенно выиграть во времени кодирования и декодирования. Конечно, этот баланс имеет место только в определённых пределах, и нельзя сократить расход памяти до нуля или построить мгновенно работающие алгоритмы кодирования. Для алфавитного кодирования пределы возможного установлены в разделе 6.2. Для достижения дальнейшего прогресса нужно рассмотреть неалфавитное кодирование.

6.4.1. Сжатие текстов

Допустим, что имеется некоторое сообщение, которое закодировано каким-то общепринятым способом (для текстов это, например, код ASCII) и хранится в памяти компьютера. Заметим, что равномерное кодирование (в частности, ASCII) не является для текстов оптимальным. Действительно, в текстах обычно используется существенно меньше, чем 256 символов (в зависимости от языка — примерно 60–80 с учётом знаков препинания, цифр, строчных и прописных букв). Кроме того, вероятности появления букв различны, и для каждого естественного языка известны (с некоторой точностью) частоты появления букв в тексте. Таким образом, можно задаться некоторым набором букв и частотами их появления в тексте, и с помощью алгоритма Хаффмена построить оптимальное алфавитное кодирование текстов (для заданного алфавита и языка). Простые расчёты показывают, что такое кодирование для распространенных естественных языков будет иметь цену кодирования несколько меньше 6, то есть даст выигрыш по сравнению с кодом ASCII на 25% или несколько больше.

ЗАМЕЧАНИЕ

Методы кодирования, позволяющие построить (без потери информации!) коды сообщений меньшей длины по сравнению с исходным сообщением, называются методами *сжатия* (или *упаковки*) данных. Качество сжатия определяется *коэффициентом сжатия*, который обычно измеряется в процентах и показывает, на сколько процентов кодированное сообщение короче исходного.

Известно, что практические программы сжатия (rar, zip и др.) имеют гораздо лучшие показатели, чем код Хаффмена: при сжатии текстовых файлов коэффициент сжатия достигает 70% и более. Это означает, что в таких программах используется *не* алфавитное кодирование.

6.4.2. Предварительное построение словаря

Рассмотрим следующий способ кодирования.

1. Исходное сообщение по некоторому алгоритму разбивается на последовательности букв, называемые *словами* (слово может иметь одно или несколько вхождений в исходный текст сообщения).

2. Полученное множество слов считается буквами нового алфавита. Для этого алфавита строится разделимая схема алфавитного кодирования (равномерного кодирования или оптимального кодирования, если для каждого слова подсчитать число вхождений в текст). Полученная схема обычно называется *словарем*, так как она сопоставляет слову код.
3. Далее код сообщения строится как пара — код словаря и последовательность кодов слов из данного словаря.
4. При декодировании исходное сообщение восстанавливается путем замены кодов слов на слова из словаря.

Пример. Допустим, что требуется кодировать тексты на русском языке. В качестве алгоритма деления на слова примем естественные правила языка: слова отделяются друг от друга пробелами или знаками препинания. Можно принять допущение, что в каждом конкретном тексте имеется не более 2^{16} различных слов (обычно гораздо меньше). Таким образом, каждому слову можно сопоставить номер — целое число из двух байтов (равномерное кодирование). Поскольку в среднем слова русского языка состоят более чем из двух букв, такое кодирование даёт существенное сжатие текста (около 75% для обычных текстов на русском языке). Если текст достаточно велик (сотни тысяч или миллионы букв, то есть сотни и тысячи страниц), то дополнительные затраты на хранение словаря оказываются сравнительно небольшими.

ЗАМЕЧАНИЕ

Данный метод попутно позволяет решить задачу *полнотекстового поиска*, то есть определить, содержится ли заданное слово (или слова) в данном тексте, причём для этого не нужно просматривать весь текст (достаточно просмотреть только словарь).

Указанный метод можно усовершенствовать следующим образом. На шаге 2 следует применить алгоритм Хаффмена для построения оптимального кода, а на шаге 1 — решить экстремальную задачу разбиения текста на слова таким образом, чтобы среди всех возможных разбиений выбрать то, которое даёт наименьшую цену кодирования на шаге 2. Такое кодирование будет «абсолютно» оптимальным. К сожалению, указанная экстремальная задача очень трудоёмка, поэтому на практике не используется — время на предварительную обработку большого текста оказывается чрезмерно велико даже при использовании современных быстродействующих компьютеров.

6.4.3. Алгоритм Лемпела–Зива

На практике используется следующая идея, которая известна также как *адаптивное сжатие*. За один проход по тексту одновременно динамически строится словарь и кодируется текст. При этом нет необходимости хранить словарь: за счёт того, что при декодировании используется тот же самый алгоритм построения словаря, словарь динамически восстанавливается.

Ниже приведена простейшая реализация этой идеи, известная как *алгоритм Лемпела*[1]*–Зива*[2].

[1] Абрахам Лемпел (род. 1936).
[2] Якоб Зив (род. 1931).

Алгоритм 6.4. Упаковка по методу Лемпела–Зива

Вход: исходный текст, заданный массивом кодов символов f : **array** $[1..n]$ **of char**.
Выход: сжатый текст, представленный последовательностью пар $\langle p, q\rangle$, где p — номер слова в словаре, q — код дополняющей буквы.

```
D[0] := ε; d := 0  // начальное состояние словаря
k := 1  // номер текущей буквы в исходном тексте
while k ⩽ n do
  p := FD(k)  // p — индекс найденного слова в словаре
  l := Length(D[p])  // l — длина найденного слова в словаре
  yield ⟨p, f[k + l]⟩  // код найденного слова и ещё одна буква
  d := d + 1; D[d] := D[p] + f[k + l]  // пополнение словаря
  k := k + l + 1  // продвижение вперед по исходному тексту
end while
```

Обоснование. Вначале словарь D : **array** [**int**] **of string** содержит пустое слово, имеющее код 0. Далее в тексте последовательно выделяются слова. Выделяемое слово — это максимально длинное слово из уже имеющихся в словаре плюс ещё один символ. В сжатое представление записываются найденный код слова и расширяющая буква, а словарь пополняется расширенной комбинацией. □

ЗАМЕЧАНИЕ

В алгоритмах этого раздела знак «+», применённый к целым числам, означает сложение, а применённый к словам или символам — конкатенацию.

Пример. Упаковать по методу Лемпела–Зива $abcdecf\ abcdac$

Вход		a	b	c	d	e	cf		ab	cd	ac
Код $\langle p, q\rangle$		$\langle 0, a\rangle$	$\langle 0, b\rangle$	$\langle 0, c\rangle$	$\langle 0, d\rangle$	$\langle 0, e\rangle$	$\langle 3, f\rangle$	$\langle 0, \rangle$	$\langle 1, b\rangle$	$\langle 3, d\rangle$	$\langle 1, c\rangle$
Индекс d	0	1	2	3	4	5	6	7	8	9	10

Получили сжатый текст, представленный массивом пар:
$\langle 0, a\rangle$, $\langle 0, b\rangle$, $\langle 0, c\rangle$, $\langle 0, d\rangle$, $\langle 0, e\rangle$, $\langle 3, f\rangle$, $\langle 0, \rangle$, $\langle 1, b\rangle$, $\langle 3, d\rangle$, $\langle 1, c\rangle$.

Слово в словаре ищется с помощью несложной функции FD.

Вход: k — номер символа в исходном тексте, начиная с которого нужно искать в тексте слова из словаря.
Выход: p — индекс самого длинного слова в словаре, совпадающего с символами $f[k]..f[k + l]$. Если такого слова в словаре нет, то $p = 0$.

```
l := 0; p := 0  // начальное состояние
for i from 1 to d do
  m := Length(D[i])  // длина слова в словаре
  if D[i] = f[k..k + m − 1] & m > l then
    p := i; l := m  // нашли более подходящее слово
  end if
end for
return p
```

Распаковка осуществляется следующим алгоритмом.

Алгоритм 6.5. Распаковка по методу Лемпела–Зива

Вход: сжатый текст, представленный массивом пар g : **array** $[1..m]$ **of struct {** p : **int;** q : **char }**, где p — номер слова в словаре, q — код дополняющей буквы.
Выход: исходный текст, заданный последовательностью строк и символов.

```
D[0] := ε; d := 0  // начальное состояние словаря
for k from 1 to m do
    p := g[k].p  // p — индекс слова в словаре
    q := g[k].q  // q — дополнительная буква
    yield D[p] + q  // вывод слова и ещё одной буквы
    d := d + 1; D[d] := D[p] + q  // пополнение словаря
end for
```

Пример. Распаковать по методу Лемпела–Зива
$\langle 0, a\rangle$, $\langle 0, b\rangle$, $\langle 0, c\rangle$, $\langle 0, d\rangle$, $\langle 0, e\rangle$, $\langle 3, f\rangle$, $\langle 0, \rangle$, $\langle 1, b\rangle$, $\langle 3, d\rangle$, $\langle 1, c\rangle$.

Код $\langle p, q\rangle$		$\langle 0, a\rangle$	$\langle 0, b\rangle$	$\langle 0, c\rangle$	$\langle 0, d\rangle$	$\langle 0, e\rangle$	$\langle 3, f\rangle$	$\langle 0, \rangle$	$\langle 1, b\rangle$	$\langle 3, d\rangle$	$\langle 1, c\rangle$
Словарь		a	b	c	d	e	cf		ab	cd	ac
Индекс d	0	1	2	3	4	5	6	7	8	9	10

Получили исходный текст $abcdecf\ abcdac$.

ЗАМЕЧАНИЕ

На практике применяют различные усовершенствования этой схемы:

1. Словарь можно сразу инициализировать, например, кодами символов (то есть считать, что однобуквенные слова уже известны).
2. В текстах часто встречаются регулярные последовательности: пробелы, табуляции в таблицах и т. п. Сопоставлять каждой подпоследовательности такой последовательности отдельное слово в словаре нерационально. В таких случаях лучше применить специальный приём, например, закодировать последовательность пробелов парой $\langle k, s\rangle$, где k — количество пробелов, а s — код пробела.

ЗАМЕЧАНИЕ

В данном параграфе представлен алгоритм Лемпела–Зива LZ78 (1978 г.), не следует путать его с алгоритмом LZ77 (Лемпела–Зива, 1977 г.), который имеет другой принцип работы.

ОТСТУПЛЕНИЕ

Основная идея LZ77 состоит в том, что второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на её первое вхождение. LZ77 использует уже просмотренную часть сообщения как словарь. Чтобы добиться сжатия, он пытается заменить очередной фрагмент сообщения на указатель в содержимое словаря. LZ77 использует «скользящее» по сообщению «окно», разделённое на две неравные части. Первая, большая по размеру, включает уже просмотренную часть сообщения. Вторая является буфером, содержащим ещё незакодированные символы входного потока. Алгоритм пытается найти в словаре (большей части окна) фрагмент, совпадающий с содержимым буфера. LZ77 выдаёт коды, состоящие из трёх элементов: смещение в словаре относительно его начала подстроки, совпадающей с началом содержимого буфера; длина этой подстроки; первый символ буфера, следующий за подстрокой. Недостатки: невозможность кодирования подстрок, отстоящих

друг от друга на расстоянии, большем длины словаря; длина подстроки, которую можно закодировать, ограничена размером буфера. Рассмотренный Алгоритм Лемпела–Зива LZ78 не имеет этих недостатков.

6.4.4. Криптография

Защита информации, хранящейся в компьютере, от несанкционированного доступа, искажения и уничтожения в настоящее время является серьёзной социальной проблемой. Применяются различные подходы к решению этой проблемы.

1. Поставить между злоумышленником и данными в компьютере непреодолимый «физический» барьер, то есть исключить саму возможность доступа к данным путём изоляции компьютера с данными в охраняемом помещении, применения аппаратных ключей защиты и т. п. Такой подход надёжен, но он затрудняет доступ к данным для законных пользователей, а потому постепенно уходит в прошлое.
2. Поставить между злоумышленником и данными в компьютере «логический» барьер, то есть проверять наличие прав на доступ к данным и блокировать доступ при отсутствии таких прав. Для этого применяются различные системы паролей, регистрация и идентификация пользователей, разграничение прав доступа и т. п. Практика показывает, что борьба между хакерами и разработчиками модулей защиты операционных систем идёт с переменным успехом.
3. Хранить данные таким образом, чтобы они могли «сами за себя постоять». Другими словами, так закодировать данные, чтобы, даже получив к ним доступ, злоумышленник не смог нанести ущерба.

Этот и последующие параграфы раздела посвящены обсуждению методов кодирования, применяемых в последнем случае.

Шифрование — это кодирование данных с целью защиты от несанкционированного доступа. Процесс кодирования сообщения называется *шифрованием* (или *зашифровкой*), а процесс декодирования — *расшифровыванием* (или *расшифровкой*). Само кодированное сообщение называется *шифрованным* (или просто *шифровкой*), а применяемый метод называется *шифром*.

Основное требование к шифру состоит в том, чтобы расшифровка (и, может быть, зашифровка) была возможна только при наличии санкции, то есть некоторой дополнительной информации (или устройства), которая называется *ключом* шифра. Процесс декодирования шифровки *без* ключа называется *дешифрованием* (или *дешифрацией*, или просто *раскрытием* шифра). Область знаний о методах создания и раскрытия шифров называется *криптографией* (или *тайнописью*). Свойство шифра противостоять раскрытию называется *криптостойкостью* (или *надёжностью*) и обычно оценивается сложностью алгоритма дешифрации.

ОТСТУПЛЕНИЕ

В практической криптографии криптостойкость шифра оценивается из экономических соображений. Если раскрытие шифра стоит (в денежном выражении, включая необходимые компьютерные ресурсы, специальные устройства, услуги специалистов и т. п.) больше, чем сама зашифрованная информация, то шифр считается достаточно надёжным.

Криптография известна с глубокой древности и использует самые разнообразные шифры, как чисто информационные, так и механические. В настоящее время наибольшее практическое значение имеет защита данных в компьютере, поэтому далее рассматриваются программные шифры для сообщений в алфавите $\{0, 1\}$.

6.4.5. Шифрование с помощью случайных чисел

Пусть имеется датчик *псевдослучайных* чисел, работающий по некоторому определённому алгоритму. Часто используют следующий алгоритм: $T_{i+1} := (a \cdot T_i + b) \bmod c$, где T_i — предыдущее псевдослучайное число, T_{i+1} — следующее псевдослучайное число, а коэффициенты a, b, c постоянны и хорошо известны. Обычно $c = 2^n$, где n — разрядность процессора, $a \bmod 4 = 1$, а b — нечётное. В этом случае последовательность псевдослучайных чисел имеет период c (см. [11]).

Процесс шифрования определяется следующим образом. Шифруемое сообщение представляется в виде последовательности слов $S_0, S_1, \ldots$, каждое длины n, которые поразрядно складываются по модулю 2 со словами последовательности $T_0, T_1, \ldots$, то есть $C_i := S_i +_2 T_i$.

ЗАМЕЧАНИЕ

Последовательность $T_0, T_1, \ldots$ называется *гаммой шифра*.

Процесс расшифровывания заключается в том, чтобы ещё раз сложить шифрованную последовательность с той же самой гаммой шифра: $S_i := C_i +_2 T_i$. Ключом шифра является начальное значение T_0, которое является секретным и должно быть известно только отправителю и получателю шифрованного сообщения.

ЗАМЕЧАНИЕ

Шифры, в которых для зашифровки и расшифровки используется один и тот же ключ, называются *симметричными*.

Если период последовательности псевдослучайных чисел достаточно велик, чтобы гамма шифра была длиннее сообщения, то дешифровать сообщение можно только подбором ключа. При увеличении n экспоненциально увеличивается криптостойкость шифра.

ОТСТУПЛЕНИЕ

Этот очень простой и эффективный метод часто применяют «внутри» программных систем, например для защиты данных на локальном диске. Для защиты данных, передаваемых по открытым каналам связи, особенно в случае многостороннего обмена сообщениями, этот метод применяют не так часто, поскольку возникают трудности с надёжной передачей секретного ключа многим пользователям.

6.4.6. Криптостойкость

Описанный в предыдущем подразделе метод шифрования обладает существенным недостатком. Если известна хотя бы часть исходного сообщения, то всё сообщение может быть легко дешифровано. Действительно, пусть известно одно исходное слово S_i. Тогда $T_i := C_i +_2 S_i$, и далее вся правая часть гаммы шифра определяется по указанной формуле датчика псевдослучайных чисел.

ЗАМЕЧАНИЕ

На практике часть сообщения вполне может быть известна злоумышленнику. Например, многие текстовые редакторы помещают в начало файла документа одну и ту же служебную информацию. Если злоумышленнику известно, что исходное сообщение подготовлено в данном редакторе, то он сможет легко дешифровать сообщение.

Для повышения криптостойкости симметричных шифров применяют различные приёмы:

1) вычисление гаммы шифра по ключу более сложным (или секретным) способом;
2) применение вместо $+_2$ более сложной (но обратимой) операции для вычисления шифровки;
3) предварительное перемешивание битов исходного сообщения по фиксированному алгоритму.

Наиболее надёжным симметричным шифром считается DES (Data Encryption Standard), в котором используется сразу несколько методов повышения криптостойкости.

6.4.7. Шифрование с открытым ключом

В настоящее время широкое распространение получили *шифры с открытым ключом*. Эти шифры не являются симметричными — для зашифровки и расшифровки используются разные ключи. При этом ключ, используемый для зашифровки, является открытым (не секретным) и может быть сообщен всем желающим отправить шифрованное сообщение, а ключ, используемый для расшифровки, является закрытым и хранится в секрете получателем шифрованных сообщений. Даже знание всего зашифрованного сообщения и открытого ключа, с помощью которого оно было зашифровано, не позволяет дешифровать сообщение (без знания закрытого ключа).

Шифрование с открытым ключом производится следующим образом.

1. Получателем сообщений производится генерация открытого ключа (пара чисел n и e) и закрытого ключа (число d). Для этого:
   а) выбираются два простых числа (п. 2.4.4), p и q;
   б) вычисляется первая часть открытого ключа $n := pq$;
   в) определяется вторая часть открытого ключа — выбирается небольшое нечётное число e, взаимно простое (п. 2.4.2) с числом $(p-1)(q-1)$ (заметим, что $(p-1)(q-1) = pq(1-1/p)(1-1/q) = \varphi(n)$, п. 2.4.9);
   г) определяется закрытый ключ: $d := e^{-1} \bmod (p-1)(q-1)$.

   После чего открытый ключ (числа n и e) сообщается всем отправителям сообщений.
2. Отправитель шифрует сообщение, разбивая его, если нужно, на слова S_i длиной менее $\log_2 n$ разрядов: $C_i := (S_i)^e \bmod n$, и отправляет получателю.
3. Получатель расшифровывает сообщение с помощью закрытого ключа d: $P_i := (C_i)^d \bmod n$.

ТЕОРЕМА. *Шифрование с открытым ключом корректно, то есть в предыдущих обозначениях $P_i = S_i$.*

Доказательство. Легко видеть, что $P_i = (S_i)^{ed} \bmod n$. Покажем, что

$$\forall\, M < n \ \left(M^{ed} \equiv M \bmod n\right).$$

Действительно, числа d и e взаимно обратны по модулю $(p-1)(q-1)$, то есть

$$ed = 1 + k(p-1)(q-1)$$

при некотором k. Если $M \not\equiv 0 \bmod p$, то по малой теореме Ферма (п. 2.4.9) имеем

$$M^{ed} \equiv M\left(M^{(p-1)}\right)^{k(q-1)} \equiv M \cdot 1^{k(q-1)} \equiv M \bmod p.$$

Если $M \equiv 0 \bmod p$, то сравнение $M^{ed} \equiv M \bmod p$, очевидно, выполняется. Таким образом, $\forall\, 0 \leqslant M < n\ \left(M^{ed} \equiv M \bmod p\right)$. Совершенно аналогично имеем $\forall\, 0 \leqslant M < n\ \left(M^{ed} \equiv M \bmod q\right)$ и, по следствию из китайской теоремы об остатках (п. 2.4.7), $\forall\, M < n\ \left(M^{ed} \equiv M \bmod n\right)$. Поскольку $S_i < n$ и $P_i < n$, заключаем, что $\forall\, i\ (P_i = S_i)$. □

Пример. Генерация ключей:

1. $p := 3$, $q := 11$.
2. $n := pq = 3 * 11 = 33$.
3. $(p-1)(q-1) = 2 * 10 = 20$, $e := 7$.
4. $d := 7^{-1} \bmod 20 = 3$, $(7 * 3 \bmod 20 = 1)$.

Пусть $S_1 := 3$, $S_2 := 1$, $S_3 := 2$ $(S_1, S_2, S_3 < n = 33)$. Тогда код определяется следующим образом:

1. $C_1 := 3^7 \bmod 33 = 2187 \bmod 33 = 9$.
2. $C_2 := 1^7 \bmod 33 = 1 \bmod 33 = 1$.
3. $C_3 := 2^7 \bmod 33 = 128 \bmod 33 = 29$.

При расшифровке имеем:

1. $P_1 := 9^3 \bmod 33 = 729 \bmod 33 = 3$.
2. $P_2 := 1^3 \bmod 33 = 1 \bmod 33 = 1$.
3. $P_3 := 29^3 \bmod 33 = 24389 \bmod 33 = 2$.

ОТСТУПЛЕНИЕ

Шифры с открытым ключом сравнительно просты в реализации, очень практичны (поскольку нет необходимости пересылать по каналам связи закрытый ключ и можно безопасно хранить его в одном месте) и в то же время обладают высочайшей криптостойкостью. Кажется, что дешифровать сообщение несложно: достаточно разложить открыто опубликованное число n на множители, восстановив числа p и q, и далее можно легко вычислить секретный ключ d. Однако дело заключается в следующем. В настоящее время известны эффективные алгоритмы определения простоты чисел, которые позволяют за несколько минут подобрать пару очень больших простых чисел (по 100 и больше цифр в десятичной записи). В то же время неизвестны эффективные алгоритмы разложения очень больших чисел на множители. Разложение на множители числа в 200 и больше цифр потребовало бы сотен лет работы самого лучшего суперкомпьютера. При практическом применении шифров с открытым ключом используют действительно большие простые числа (не менее 100 цифр в десятичной записи, а обычно значительно больше). В результате вскрыть этот шифр оказывается невозможно, если не существует эффективных алгоритмов разложения на множители (что очень вероятно, хотя и не доказано строго).

6.4.8. Цифровая подпись

Шифр с открытым ключом позволяет выполнять и многие другие полезные операции, помимо шифрования и посылки сообщений в одну сторону. Прежде всего, для организации многосторонней секретной связи каждому из участников достаточно сгенерировать свою пару ключей (открытый и закрытый), а затем сообщить всем партнёрам свой открытый ключ.

Заметим, что операции зашифровки и расшифровки, по существу, одинаковы и различаются только показателем степени, а потому коммутируют:

$$M = (M^e)^d \bmod n = M^{ed} \bmod n = M^{de} \bmod n = (M^d)^e \bmod n = M.$$

Это обстоятельство позволяет применять различные приёмы, известные как *цифровая* (или *электронная*) подпись.

Рассмотрим следующую схему взаимодействия корреспондентов X и Y. Отправитель X кодирует сообщение S своим закрытым ключом ($C := S^d \bmod n$) и посылает получателю Y пару $\langle S, C\rangle$, то есть подписанное сообщение. Получатель Y, получив такое сообщение, кодирует подпись сообщения открытым ключом отправителя X.

То есть вычисляет $S' := C^e \bmod n$. Если оказывается, что $S = S'$, то это означает, что (нешифрованное!) сообщение S действительно было отправлено корреспондентом X. Если же $S \neq S'$, то сообщение было искажено при передаче или фальсифицировано.

ОТСТУПЛЕНИЕ

В подобного рода схемах возможны различные проблемы, которые носят уже не математический, а социальный характер. Например, допустим, что злоумышленник Z имеет техническую возможность контролировать всю входящую корреспонденцию получателя Y незаметно для последнего. Тогда, перехватив сообщение отправителя X, в котором сообщался открытый ключ e, злоумышленник Z может подменить открытый ключ отправителя X своим собственным открытым ключом. После этого злоумышленник сможет фальсифицировать все сообщения отправителя X, подписывая их своей цифровой подписью, и таким образом действовать от имени отправителя X. Другими словами, цифровая подпись удостоверяет, что сообщение S пришло из того же источника, из которого был получен открытый ключ e, но не более того.

Можно подписывать и шифрованные сообщения. Для этого отправитель X сначала кодирует своим закрытым ключом сообщение S, получая цифровую подпись C, а затем кодирует полученную пару $\langle S, C\rangle$ открытым ключом получателя Y. Получив такое сообщение, получатель Y сначала расшифровывает сообщение своим закрытым ключом, а потом убеждается в подлинности полученного сообщения, сравнив его с результатом применения открытого ключа отправителя X к подписи C.

ЗАМЕЧАНИЕ

К сожалению, даже эти меры не смогут защитить от злоумышленника Z, сумевшего подменить открытый ключ отправителя X. Конечно, в этом случае злоумышленник не сможет дешифровать исходное сообщение, но он сможет подменить исходное сообщение фальсифицированным, а получатель не сможет обнаружить подмену сообщения.

Глава 7 Графы

Эта глава открывает заключительную часть книги, целиком посвященную графам и алгоритмам на них. Среди дисциплин и методов дискретной математики теория графов и особенно *алгоритмы на графах* находят наиболее широкое применение в программировании.

Как показано в разделе 7.5, между понятием графа и понятием бинарного отношения, рассмотренным в главе 1, имеется глубокая связь — можно сказать, что это равнообъёмные понятия. Возникает естественный вопрос, почему же тогда графам оказывается столь заметное предпочтение при изучении дискретной математики и программирования?

Дело в том, что теория графов предоставляет очень удобный *язык* для описания программных (да и многих других) моделей. Этот тезис можно пояснить следующей аналогией. Понятие отношения также можно полностью выразить через понятие множества (замечание в п. 1.5.1 и далее). Однако независимое определение понятия отношения *удобнее* — введение специальных терминов и обозначений упрощает изложение теории и делает её более понятной. То же относится и к теории графов.

Стройная система специальных терминов и обозначений теории графов позволяет просто и доступно описывать сложные и тонкие вещи. Особенно важна возможность наглядной графической интерпретации понятия графа (см. п. 7.1.4). Само название «граф» подразумевает наличие графической интерпретации. Картинки часто позволяют сразу «усмотреть» суть дела на интуитивном уровне, дополняя и украшая утомительные текстовые доказательства и сложные формулы.

Представление графов и подобных им объектов в программах даёт убедительный пример решающей роли представления данных в программировании. Для графов множественность альтернативных представлений в программах не только теоретически возможна, но и практически неизбежна. Многие алгоритмы на графах достаточно эффективны в том смысле, что имеют невысокую теоретическую оценку трудоёмкости. Такие алгоритмы бывают чувствительны к аддитивным и мультипликативным константам, которыми пренебрегают при получении теоретической оценки. Значение этих констант определяется представлением графа. На практике, особенно для больших графов, неудачно выбранное представление может воспрепятствовать решению конкретной задачи на имеющемся оборудовании.

Эта глава практически полностью посвящена описанию *языка* теории графов. Конкретные методы и алгоритмы более подробно рассматриваются далее.

7.1. Определения графов

Как это ни удивительно, но для понятия «граф» нет общепризнанного единого определения. Разные авторы, особенно применительно к разным приложениям, называют словом «граф» очень похожие, но все-таки различные объекты. Здесь используется терминология из книги Харари [9], которая была выбрана из соображений максимального упрощения определений и доказательств.

7.1.1. История теории графов

Теория графов многократно переоткрывалась разными авторами при решении различных прикладных задач.

1. *Задача о кёнигсбергских мостах.* На рис. 7.1 представлен схематический план центральной части города Кёнигсберг (ныне Калининград), включающий два берега реки Преголя, два острова на ней и семь соединяющих их мостов. Задача состоит в том, чтобы обойти все четыре участка суши (обозначены A, B, C, D), пройдя по каждому мосту один раз, и вернуться в исходную точку. В 1736 году Эйлером[1] было показано, что решения этой задачи не существует. Точнее говоря, Эйлер получил необходимое и достаточное условие существования решения для всех задач типа задачи о кёнигсбергских мостах (см. п. 10.2.1).

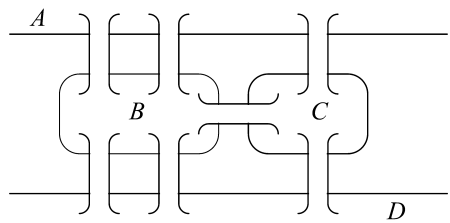


Рис. 7.1. Иллюстрация к задаче о кёнигсбергских мостах

2. *Задача о трёх домах и трёх колодцах.* Имеется три дома и три колодца, каким-то образом расположенные на плоскости. Требуется провести от каждого дома к каждому колодцу тропинку так, чтобы тропинки не пересекались. Эта задача также не имеет решения (рис. 7.2). В 1930 году Куратовским[2] было доказано намного более сильное утверждение, а именно достаточность условия существования решения для всех задач типа задачи о трёх домах и трёх колодцах (необходимость этого условия была известна и ранее, см. п. 10.8.2).

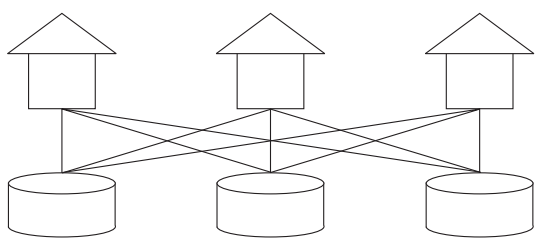


Рис. 7.2. Иллюстрация к задаче о трёх домах и трёх колодцах

3. *Задача о четырёх красках.* Разделение плоскости на неперекрывающиеся области называется *картой*. Области на карте называются соседними, если они имеют общую границу. Задача состоит в раскрашивании карты таким образом, чтобы никакие две соседние области не были закрашены одним цветом (рис. 7.3). С конца XIX века известна гипотеза, что для этого достаточно четырёх красок. В 1976 году Аппель и Хейкен опубликовали решение задачи о четырёх красках, которое базировалось на переборе вариантов с помощью компьютера.

---

[1] Леонард Эйлер (1707–1783).

[2] Казимир Куратовский (1896–1979).

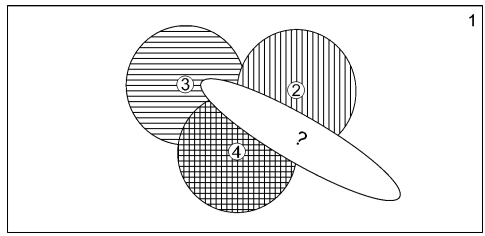


Рис. 7.3. Иллюстрация к задаче о четырёх красках

ОТСТУПЛЕНИЕ

Решение задачи о четырёх красках Аппелем и Хейкеном «программным путем» явилось прецедентом, породившим бурную дискуссию, которая отнюдь не закончена. Суть опубликованного решения состоит в том, чтобы перебрать большое, но конечное число (около 2000) типов потенциальных контрпримеров к теореме о четырёх красках и показать, что ни один случай контрпримером не является. Этот перебор был выполнен программой примерно за тысячу часов работы суперкомпьютера. Проверить «вручную» полученное решение невозможно — объём перебора выходит далеко за рамки человеческих возможностей. Многие математики ставят вопрос: можно ли считать такое «программное доказательство» действительно доказательством? Ведь в программе могут быть ошибки... Методы формального доказательства правильности программ не применимы к программам такой сложности, как обсуждаемая. Тестирование не может гарантировать отсутствие ошибок и в данном случае вообще невозможно. Таким образом, остаётся уповать на программистскую квалификацию авторов и верить, что они всё сделали правильно.

7.1.2. Основное определение

Графом $G(V,E)$ называется совокупность двух множеств — непустого множества V (множества *вершин*) и множества E двухэлементных подмножеств множества V (E — множество *рёбер*),

$$G(V,E) \stackrel{\text{Def}}{=} \langle V; E\rangle\,, \quad V \neq \varnothing, \quad E \subset 2^V \;\&\; \forall\, e \in E \;\; (|e| = 2)\,.$$

ЗАМЕЧАНИЕ

Легко видеть, что любое множество E двухэлементных подмножеств множества V определяет симметричное бинарное отношение на множестве V. Поэтому можно считать, что $E \subset V \times V$, $E = E^{-1}$ и трактовать ребро не только как множество $\{v_1, v_2\}$, но и как пару (v_1, v_2).

Число вершин графа G обозначим p, а число рёбер — q:

$$p \stackrel{\text{Def}}{=} p(G) \stackrel{\text{Def}}{=} |V|, \quad q \stackrel{\text{Def}}{=} q(G) \stackrel{\text{Def}}{=} |E|.$$

Если хотят явно упомянуть числовые характеристики графа, то говорят (p,q)-граф.

ЗАМЕЧАНИЕ

Множества вершин и рёбер считаются конечными, если явно не оговорено противное.

7.1.3. Инцидентность, смежность и дополнение

Пусть v_1, v_2 — вершины, $e = (v_1, v_2)$ — соединяющее их ребро. Тогда вершина v_1 и ребро e *инцидентны*, ребро e и вершина v_2 также инцидентны. Два ребра, инцидентные одной вершине, называются *смежными*; две вершины, инцидентные одному ребру, также называются *смежными*.

Множество вершин, смежных с вершиной v, называется *множеством смежности* (или *окрестностью*) вершины v и обозначается $\Gamma^+(v)$. По определению $\Gamma^+(v) \stackrel{\text{Def}}{=} \{u \in V \mid (u, v) \in E\}$, $\Gamma^*(v) \stackrel{\text{Def}}{=} \Gamma^+(v) + v$.

Множество рёбер, инцидентных вершине v, называется *множеством инцидентности* вершины v и обозначается $\Lambda(v)$. По определению $\Lambda(v) \stackrel{\text{Def}}{=} \{e \in E \mid v \in e\}$.

ЗАМЕЧАНИЕ

Если не оговорено противное, то символ Γ без индекса подразумевает Γ^+, то есть саму вершину в окрестность не включают.

Очевидно, что $u \in \Gamma(v) \Longleftrightarrow v \in \Gamma(u)$. Если $A \subset V$ — множество вершин, то $\Gamma(A)$ — множество всех вершин, смежных с вершинами из A:

$$\Gamma(A) \stackrel{\text{Def}}{=} \{u \in V \mid \exists\, v \in A \;\; (u \in \Gamma(v))\} = \bigcup_{v \in A} \Gamma(v).$$

Рассмотрим граф $G(V, E)$ с некоторым множеством вершин V. *Дополнением* $\overline{G}(V, \overline{E})$ называется другой граф, с тем же множеством вершин V, в котором вершины смежны тогда и только тогда, когда они не смежны в исходном графе.

Другими словами, если ребро $e = \{u, v\}$ принадлежит графу, то оно не принадлежит дополнению графа, и обратно, если ребро $e = \{u, v\}$ принадлежит дополнению графа, то оно не принадлежит графу.

ТЕОРЕМА. *Суммарное количество рёбер в графе и в его дополнении определяется числом вершин p и равно $p(p-1)/2$.*

Доказательство. Число двухэлементных подмножеств множества вершин — это число сочетаний $C(p, 2) = \frac{p!}{2!(p-2)!} = p(p-1)/2$. □

СЛЕДСТВИЕ. *Для любой вершины суммарное количество вершин смежных и не смежных с ней равно $p - 1$.*

Дополнение можно рассматривать и как операцию на графах, и как отношение между графами. Как операция, дополнение инволютивно: дополнением к дополнению графа является сам граф. Как отношение, дополнение симметрично: если $\overline{G}$ является дополнением G, то и G является дополнением $\overline{G}$.

7.1.4. Диаграмма графа

Обычно граф изображают *диаграммой*: вершины — точками (или кружками), рёбра — линиями.

Пример. На рис. 7.4 приведен пример диаграммы графа, имеющего четыре вершины и пять рёбер. В этом графе вершины v_1 и v_2, v_2 и v_3, v_3 и v_4, v_4 и v_1, v_2 и v_4 смежны, а вершины v_1 и v_3 не смежны. Смежные рёбра: e_1 и e_2, e_2 и e_3, e_3 и e_4, e_4 и e_1, e_1 и e_5, e_2 и e_5, e_3 и e_5, e_4 и e_5. Несмежные рёбра: e_1 и e_3, e_2 и e_4.

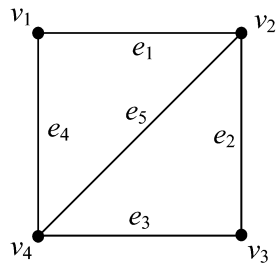


Рис. 7.4. Диаграмма графа

7.1.5. Орграфы, псевдографы, мультиграфы и гиперграфы

Часто рассматриваются и другие родственные графам объекты.

1. Если элементами множества E являются *упорядоченные* пары (то есть $E \subset V \times V$), то граф называется *ориентированным* (или *орграфом*). В этом случае элементы множества V называются *узлами*, а элементы множества E — *дугами*.
2. Если элементом множества E может быть пара *одинаковых* (не различных) элементов V, то такой элемент множества E называется *петлей*, а граф называется *графом с петлями* (или *псевдографом*).
3. Если E является не множеством, а *мультимножеством*, содержащим некоторые элементы по нескольку раз, то эти элементы называются *кратными рёбрами*, а граф называется *мультиграфом*.
4. Если элементами множества E являются не обязательно двухэлементные, а *любые* (непустые) подмножества множества V, то такие элементы множества E называются *гиперрёбрами*, а граф называется *гиперграфом*.
5. Если множество E является отношением на 2^V, то есть каждый элемент множества E действует из множества «начал» в множество «концов», то элементы множества E называются *гипердугами*, а граф называется *гиперорграфом*.
6. Если задана функция $F\colon V \to M$ и (или) $F\colon E \to M$, то множество M называется множеством *пометок*, а граф называется *помеченным* (или *нагруженным*). В качестве множества пометок обычно используются буквы или целые числа. Если функция F инъективна, то есть разные вершины (рёбра) имеют разные пометки, то граф называют *нумерованным*.

ЗАМЕЧАНИЕ

Отношение смежности является в некотором смысле определяющим для графов и подобных им объектов. При этом следует учитывать особенности каждого типа объектов. В орграфе узел v смежен с узлом u, если существует дуга (u, v). При этом узел u может быть несмежен с узлом v. Отношение смежности в графе симметрично, а в орграфе оно вовсе не обязано быть симметричным. В графе обычно считают отношение смежности рефлексивным, то есть

полагают, что вершина смежна сама с собой. В псевдографе, напротив, вершину не считают смежной с собой, если у неё нет петли. В гиперграфе две вершины считаются смежными, если они принадлежат одному гиперребру. В гипероргафе гипердуга обычно проводится из одного узла в множество узлов (возможно, пустое). В таком случае отношение смежности оказывается уже не бинарным отношением на V, а отношением из V в 2^V. Эти и подобные естественные вариации определений обычно считают ясными из контекста.

Далее выражение «граф $G(V, E)$» означает неориентированный непомеченный граф без петель и кратных рёбер с множеством вершин V и множеством рёбер E.

7.1.6. Изоморфизм графов

Говорят, что два графа, $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$, *изоморфны* (обозначается $G_1 \sim G_2$ или $G_1 = G_2$), если существует изоморфизм $h\colon V_1 \to V_2$, уважающий отношение смежности (см. п. 1.7.4): $e_1 = (u, v) \in E_1 \iff e_2 = \big(h(u), h(v)\big) \in E_2$.

ТЕОРЕМА. *Изоморфизм графов есть отношение эквивалентности.*

Доказательство.

[Рефлексивность] Имеем $G \overset{h}{\sim} G$, где h есть тождественная функция.

[Симметричность] Если $G_1 \overset{h}{\sim} G_2$, то $G_2 \overset{h^{-1}}{\sim} G_1$.

[Транзитивность] Если $G_1 \overset{h}{\sim} G_2$ и $G_2 \overset{g}{\sim} G_3$, то $G_1 \overset{g\circ h}{\sim} G_3$. □

Графы рассматриваются *с точностью до изоморфизма*, то есть рассматриваются классы эквивалентности по отношению изоморфизма. (п. 2.1.5).

Пример. Три внешне различные диаграммы, приведённые на рис. 7.5, являются диаграммами одного и того же графа $K_{3,3}$.

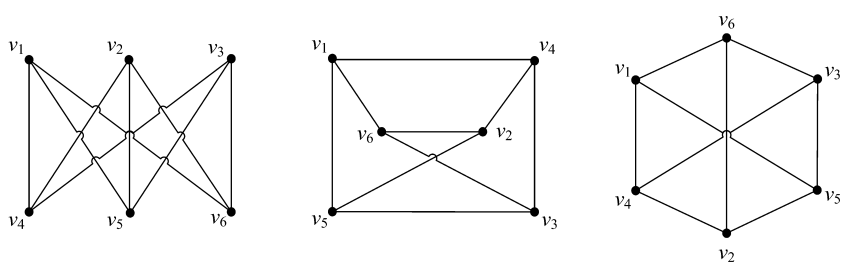


Рис. 7.5. Диаграммы изоморфных графов

Числовая характеристика, одинаковая для всех изоморфных графов, называется *инвариантом* графа. Так, $p(G)$ и $q(G)$ — инварианты графа G. Не известно никакого простого набора инвариантов, определяющих граф с точностью до изоморфизма.

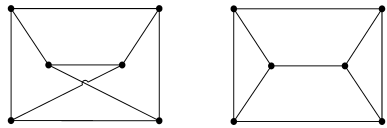


Рис. 7.6. Диаграммы неизоморфных графов с совпадающими инвариантами

Пример. Количество вершин, рёбер и количество смежных вершин для каждой вершины не определяют граф даже в простейших случаях! На рис. 7.6 представлены диаграммы графов, у которых указанные инварианты совпадают, но графы при этом не изоморфны. Представленные графы не изоморфны, потому что в графе справа есть два треугольника, а в графе слева треугольников нет.

Изморфизм графа в себя называется *автоморфизмом* графа.

Пример. Треугольник автоморфен себе при любой перестановке вершин.

7.2. Элементы графов

После рассмотрения определений, относящихся к графам как к цельным объектам, естественно дать определения различным элементам графов.

7.2.1. Подграфы

Граф $G'(V', E')$ называется *подграфом* (или *частью*) графа $G(V, E)$ (обозначается $G' \subset G$), если $V' \subset V \ \& \ E' \subset E$. Если $V' = V$, то G' называется *остовным подграфом* G. Если $V' \subset V \ \& \ E' \subset E \ \& (V' \neq V \vee E' \neq E)$, то граф G' называется *собственным* подграфом графа G. Подграф $G'(V', E')$ называется *правильным* подграфом графа $G(V, E)$, если G' содержит все возможные рёбра G: $\forall\, u, v \in V' \ \ ((u, v) \in E \Longrightarrow (u, v) \in E')$.

Правильный подграф $G'(V', E')$ графа $G(V, E)$ определяется подмножеством вершин V'.

Пример. На рис. 7.7 показаны диаграммы графа, правильного подграфа и неправильного подграфа.

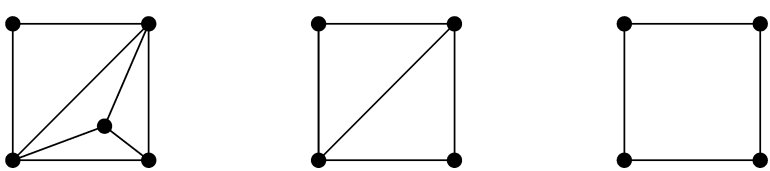


Рис. 7.7. Диаграммы графа, правильного подграфа и неправильного подграфа

ЗАМЕЧАНИЕ

Иногда подграфами называют только правильные подграфы, а неправильные подграфы называют *изграфами*.

7.2.2. Валентность

Количество рёбер, инцидентных вершине v, называется *степенью* (или *валентностью*) вершины v и обозначается $d(v)$:

$$\forall\, v \in V \;\; (0 \leqslant d(v) \leqslant p-1)\,, \qquad d(v) = |\Gamma^+(v)|.$$

Таким образом, степень $d(v)$ вершины v совпадает с количеством смежных с ней вершин. Количество вершин, не смежных с v, обозначают $\overline{d}(v)$. Ясно, что

$$\forall\, v \in V \;\; \big(d(v) + \overline{d}(v) = p-1\big)\,.$$

Обозначим *минимальную* степень вершины графа G через $\delta(G)$, а *максимальную* — через $\Delta(G)$:

$$\delta\left(G(V,E)\right) \stackrel{\text{Def}}{=} \min_{v\in V} d(v), \qquad \Delta\left(G(V,E)\right) \stackrel{\text{Def}}{=} \max_{v\in V} d(v).$$

Ясно, что $\delta(G)$ и $\Delta(G)$ являются инвариантами. Если степени всех вершин равны k, то граф называется *регулярным* степени k:

$$\delta(G) = \Delta(G) = k, \quad \forall\, v \in V \;\; (d(v) = k)\,.$$

Степень регулярности обозначается $r(G)$. Для нерегулярных графов $r(G)$ не определено.

Примеры

На рис. 7.8 приведены диаграммы трёх примечательных регулярных графов степени 3. На рис. 7.6 приведены диаграммы двух регулярных, но неизоморфных графов степени 3.

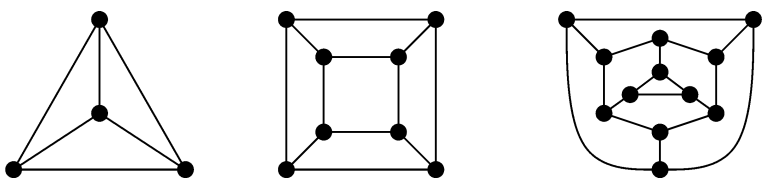


Рис. 7.8. Диаграммы регулярных графов степени 3

Если степень вершины равна нулю (то есть $d(v) = 0$), то вершина называется *изолированной*. Если степень вершины равна единице (то есть $d(v) = 1$), то вершина называется *концевой*, или *висячей*.

Для орграфа число дуг, исходящих из узла v, называется *полустепенью исхода*, а число входящих — *полустепенью захода*. Обозначаются эти числа $d^-(v)$ и $d^+(v)$.

ТЕОРЕМА (Лемма о рукопожатиях). *Сумма степеней вершин графа (мультиграфа) равна удвоенному количеству рёбер:*

$$\sum_{v\in V} d(v) = 2q.$$

Доказательство. При подсчёте суммы степеней вершин каждое ребро учитывается два раза: для одного конца ребра и для другого. □

СЛЕДСТВИЕ 1. *Число вершин нечётной степени чётно.*

Доказательство. По теореме сумма степеней всех вершин — чётное число. Сумма степеней вершин чётной степени чётна, значит, сумма степеней вершин нечётной степени также чётна, следовательно, их чётное число. □

СЛЕДСТВИЕ 2. *Сумма полустепеней узлов орграфа равна удвоенному количеству дуг:*

$$\sum_{v\in V} d^-(v) + \sum_{v\in V} d^+(v) = 2q.$$

Доказательство. Сумма полустепеней узлов орграфа равна сумме степеней вершин графа (мультиграфа), полученного из орграфа забыванием ориентации дуг. □

7.2.3. Маршруты, цепи, циклы

Маршрутом в графе называется чередующаяся последовательность вершин и рёбер, начинающаяся и кончающаяся вершиной, $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, в которой любые два соседних элемента инцидентны, причём однородные элементы (вершины, рёбра) через один смежны или совпадают.

ЗАМЕЧАНИЕ

Это определение подходит также для псевдо-, мульти- и орграфов. При этом в графе (орграфе) достаточно указать только последовательность вершин (узлов) или только последовательность рёбер (дуг).

Если $v_0 = v_k$, то маршрут *замкнут*, иначе — *открыт*. Если все рёбра различны, то маршрут называется *цепью*. Если все вершины (а значит, и рёбра) различны, то маршрут называется *простой* цепью. В цепи $v_0, e_1, \ldots, e_k, v_k$ вершины v_0 и v_k называются *концами* цепи. Говорят, что цепь с концами u и v *соединяет* вершины u и v. Цепь, соединяющая вершины u и v, обозначается $\langle u, v\rangle$. Если нужно указать граф G, которому принадлежит цепь, то добавляют индекс: $\langle u, v\rangle_G$. Нетрудно показать, что если есть какая-либо цепь, соединяющая вершины u и v, то есть и простая цепь, соединяющая эти вершины.

Замкнутая цепь называется *циклом*; замкнутая простая цепь называется *простым циклом*. Число циклов в графе G обозначается $z(G)$. Граф без циклов называется *ациклическим*. Число рёбер в цикле называется *длиной цикла*. Число циклов длины k в графе G обозначается $z(G, k)$.

ЛЕММА. *Количество простых циклов длины k одинаково для изоморфных графов.*

Доказательство. Пусть графы $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ изоморфны с биекцией $h\colon V_1 \to V_2$, сохраняющей смежность. Количество простых циклов длины k в G_2 не меньше, чем в G_1: $z(G_1, l) \leqslant z(G_2, l)$. От противного. Пусть $v_1, v_2, \ldots, v_l$ — простой цикл графа G_1, а $h(v_1), h(v_2), \ldots, h(v_l)$ не является простым циклом графа G_2. Значит, найдётся такая пара вершин $h(v_i), h(v_{i+1})$, для которой $(h(v_i), h(v_{i+1})) \notin E_2$, что противоречит тому, что h сохраняет смежность. Аналогично, количество простых циклов длины k в G_2 не больше, чем в G_1: достаточно рассмотреть $h^{-1}\colon V_2 \to V_1$. □

ЗАМЕЧАНИЕ

Для псевдографов обычно особо оговаривают, считаются ли петли циклами.

Для орграфов цепь называется *путем*, а цикл — *контуром*. Путь в орграфе из узла u в узел v обозначают $\langle \overrightarrow{u,v} \rangle$.

Пример. В графе, диаграмма которого приведена на рис. 7.9:

1) v_1, v_3, v_1, v_4 — маршрут, но не цепь;
2) $v_1, v_3, v_5, v_2, v_3, v_4$ — цепь, но не простая цепь;
3) v_1, v_4, v_3, v_2, v_5 — простая цепь;
4) $v_1, v_3, v_5, v_2, v_3, v_4, v_1$ — цикл, но не простой цикл;
5) v_1, v_3, v_4, v_1 — простой цикл.

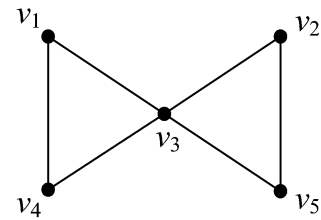


Рис. 7.9. Маршруты, цепи, циклы

7.2.4. Связные и несвязные графы

Говорят, что две вершины в графе *связаны*, если существует соединяющая их (простая) цепь. Граф, в котором все вершины связаны, называется *связным*. Нетрудно показать, что отношение связанности вершин является эквивалентностью. Классы эквивалентности по отношению связанности называются *компонентами связности* графа. Число компонент связности графа G обозначается $k(G)$. Граф G является связным тогда и только тогда, когда $k(G) = 1$. Если $k(G) > 1$, то G — *несвязный* граф. Граф, состоящий только из изолированных вершин (в котором $k(G) = p(G)$ и $r(G) = 0$), называется *вполне несвязным*.

ТЕОРЕМА. *Для любого графа либо сам граф, либо его дополнение является связным графом.*

Доказательство. Если исходный граф связен, то теорема доказана. Пусть исходный граф несвязен и имеет несколько компонент связности. Рассмотрим любые две вершины. Если они принадлежат разным компонентам связности исходного графа, то они соединены в дополнении ребром, а потому связаны. Если они принадлежат одной компоненте связности, то в дополнении они связаны маршрутом из двух рёбер, проходящим через любую вершину из другой компоненты связности. □

7.2.5. Расстояние между вершинами, ярусы и диаметр графа

Длиной маршрута называется количество рёбер в нём (с учётом повторений). Если маршрут $M = v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, то длина M равна k (обозначается $|M| = k$). *Расстоянием* между вершинами u и v (обозначается $d(u,v)$) называется длина кратчайшей цепи $\langle u, v \rangle$, а сама кратчайшая цепь называется *геодезической*, $d(u,v) \stackrel{\text{Def}}{=} \min\limits_{\{\langle u,v \rangle\}} |\langle u, v \rangle|$. Если для любых двух вершин графа существует единственная геодезическая цепь, то граф называется *геодезическим*.

ЗАМЕЧАНИЕ

Если вершины u и v не связаны, то $d(u,v) \stackrel{\text{Def}}{=} +\infty$.

Легко видеть, что расстояние между вершинами является метрикой в пространстве вершин графа. Множество вершин, находящихся на заданном расстоянии n от вершины v (обозначение $D(v,n)$), называется *ярусом*:
$D(v,n) \stackrel{\text{Def}}{=} \{u \in V \mid d(v,u) = n\}$. Ясно, что множество вершин V всякого связного графа однозначно разбивается на ярусы относительно данной вершины. *Диаметром* $D(G)$ графа G называется длиннейшая геодезическая: $D(G) \stackrel{\text{Def}}{=} \max\limits_{u,v \in V} d(u,v)$.

7.2.6. Эксцентриситет и центр

Эксцентриситетом $e(v)$ вершины v в связном графе $G(V,E)$ называется максимальное расстояние от вершины v до других вершин графа G:

$$e(v) \stackrel{\text{Def}}{=} \max_{u \in V} d(v,u).$$

Заметим, что наиболее эксцентричные вершины — это концы диаметра. *Радиусом* $R(G)$ графа G называется наименьший из эксцентриситетов вершин:

$$R(G) \stackrel{\text{Def}}{=} \min_{v \in V} e(v).$$

Вершина v называется *центральной*, если её эксцентриситет совпадает с радиусом графа, $e(v) = R(G)$. Множество центральных вершин $C(G)$ называется *центром* графа: $C(G) \stackrel{\text{Def}}{=} \{v \in V \mid e(v) = R(G)\}$.

Пример. На рис. 7.10 указаны эксцентриситеты вершин и центры двух графов. Вершины, составляющие центр, закрашены.

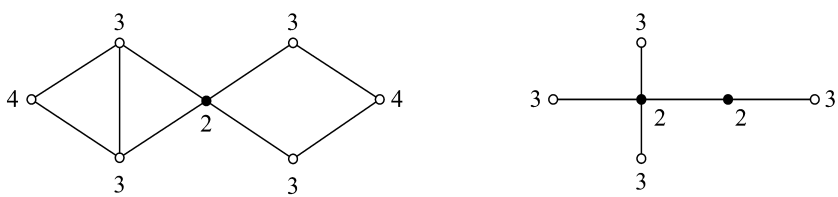


Рис. 7.10. Эксцентриситеты вершин и центры графов

7.2.7. Степенные последовательности

Последовательность степеней вершин $(d_1, \ldots, d_p)$ некоторого графа называется *степенной последовательностью графа*, или *графовой последовательностью*. Ясно, что всякая степенная последовательность является представлением мультимножества, поэтому, не ограничивая общности, можно считать, что
$d_1 \leqslant d_2 \leqslant \cdots \leqslant d_p$.

Пример. Последовательности $(2,2,2,2,3,3,4)$ и $(1,1,1,1,2,4)$ являются степенными последовательностями графов, диаграммы которых представлены на предыдущем рис..

Не всякая последовательность неотрицательных целых чисел является степенной последовательностью.

Пример. Последовательность $(0, 1, 2, \ldots, p-1)$ не является степенной ни при каком p. Действительно, первая вершина должна быть изолированной, а последняя должна быть смежна со всеми остальными, что противоречит изолированности первой вершины.

Если последовательность $D = (d_1, \ldots, d_p)$ является степенной последовательностью графа G, то говорят, что граф G *реализует* последовательность D.

Легко заметить, что, если два графа имеют различные степенные последовательности, то они не изоморфны. Но из рис. 7.6 видно, что графы с одинаковыми степенными последовательностями также могут быть не изоморфными. Таким образом, степенная последовательность может иметь множество *различных* реализаций.

Отметим несколько почти очевидных свойств степенных последовательностей.

ЛЕММА 1. *Сумма степенной последовательности — чётное число.*

Доказательство. По лемме о рукопожатиях $\sum_{i=1}^{p} d_i = 2q$. □

ЛЕММА 2. *Если последовательность $(d_1, \ldots, d_p)$ степенная, то последовательность $(0, \ldots, 0, d_1, \ldots, d_p)$ также степенная, и наоборот.*

Доказательство. Изолированные вершины можно не учитывать. □

ЛЕММА 3. *В степенной последовательности $(d_1, \ldots, d_p)$ каждый элемент меньше длины последовательности: $\forall\, i \in 1..p\ \ (d_i < p)$.*

Доказательство. Спепень любой вершины меньше числа вершин. □

Таким образом, не умаляя общности, достаточно рассматривать только последовательности $(d_1, \ldots, d_p)$, такие что $0 < d_1 \leqslant d_2 \leqslant \cdots \leqslant d_p < p$.

Пусть $D = (d_1, \ldots, d_p)$ — неубывающая последовательность натуральных чисел, причём $\forall\, i\ \ (d_i < p)$. Построим последовательность $D' = (d'_1, \ldots, d'_{p-1})$ следующим образом: $d'_{p-1} := d_{p-1} - 1$, $\ldots$, $d'_{p-d_p+1} := d_{p-d_p+1} - 1$, $d'_{p-d_p} := d_{p-d_p} - 1$, $d'_{p-d_p-1} := := d_{p-d_p-1}$, $\ldots$, $d'_1 := d_1$. Другими словами, последний (наибольший) элемент последовательности d_p удаляется, а d_p предшествующих элементов уменьшаются на единицу. Оставшиеся элементы (а они есть, поскольку $d_p < p$) остаются без изменения. Такая операция называется *откладыванием* элемента d_p, а D' называется *остаточной последовательностью*.

Пример. Отложим последний элемент в последовательности: $(2, 2, 3, 3, 3 \,|\, 3) \mapsto$ $\mapsto (2, 2, 2, 2, 2)$. При откладывании элемента упорядоченность может нарушиться: $(2, 2, 2, 2 \,|\, 2) \mapsto (2, 2, 1, 1)$.

Графически откладывание элемента можно отобразить, соединяя рёбрами вершину, соответствующую удаляемому элементу, с вершинами, соответствующими уменьшаемым элементам. Фактически, при откладывании элементов постепенно прорисовывается диаграмма графа (если это возможно), начиная с вершины с максимальной степенью.

ЗАМЕЧАНИЕ

Поскольку откладывание элемента в степенной последовательности однозначно соответствует прорисовке вершины в диаграмме графа, часто используют вольность речи и говорят «откладывание вершины».

Пример. На рис. 7.11 показано откладывание вершин для последовательности $(2, 3, 3, 3, 3)$. Откладываемые вершины выделены серым.

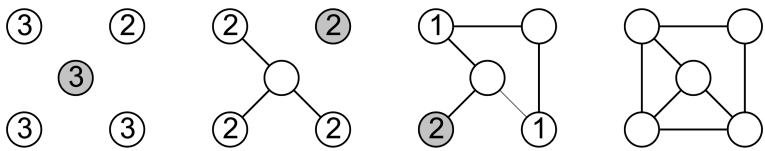


Рис. 7.11. Откладывание вершин

ТЕОРЕМА. *Последовательность $D = (d_1, \ldots, d_p)$ является степенной последовательностью тогда и только тогда, когда последовательность $D' = (d'_1, \ldots, d'_{p-1})$, полученная откладыванием последнего элемента d_p, является степенной.*

Доказательство.

[$\Longrightarrow$] Пусть последовательность $D = (d_1, \ldots, d_p)$ степенная, а G — реализующий граф. Рассмотрим в графе G вершину максимальной степени w, $d(w) = d_p$. Пусть S — множество вершин со степенями $d_{p-d_p}, \ldots, d_{p-1}$. Тогда если $\Gamma(w) = S$, то $G - w$ — искомый граф, реализующий последовательность $D' = (d'_1, \ldots, d'_{p-1})$. Пусть теперь $\Gamma(w) \neq S$, построим тогда такой граф G', в котором $\Gamma(w) = S$. Поскольку $\Gamma(w) \neq S$, существует вершина $z \in \Gamma(w)$, смежная с вершиной w, и существует вершина $x \in S$, не смежная с вершиной w. По построению S имеем $d(z) \leqslant d(x)$, следовательно, существует вершина y, смежная с x, но не смежная с z. Удалим рёбра (w, z) и (y, x), добавим рёбра (w, x) и (y, z). Такими построениями получаем граф G' такой, что $\Gamma(w) = S$, причём граф $G' - w$ реализует последовательность $D' = (d'_1, \ldots, d'_{p-1})$, рис. 7.12.

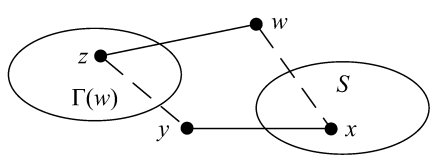


Рис. 7.12. К доказательству теоремы п. 7.2.7

[$\Longleftarrow$] Пусть последовательность $D' = (d'_1, \ldots, d'_{p-1})$ степенная, а G — реализующий граф, вершины которого перенумерованы в порядке неубывания степеней $1, \ldots, p-1$. Добавим в граф G вершину степени d_p и соединим её с вершинами с номерами $p - d_p .. p - 1$. Полученный граф реализует последовательность $D = (d_1, \ldots, d_p)$. □

Доказанная теорема позволяет проверить, является ли заданная последовательность натуральных чисел степенной последовательностью и построить граф, реализующий эту последовательность. Алгоритм использует двусвязный список для представления степенной последовательности. Процедура Ordering восстанавливает упорядоченность двунаправленного списка, используя тот факт, что отрезки слева и справа упорядоченны (рис. 7.13).

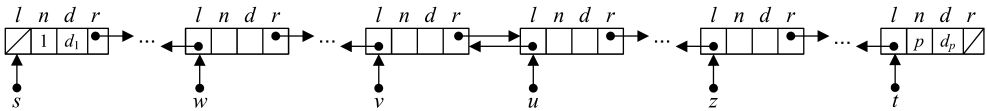


Рис. 7.13. К процедуре Ordering

Алгоритм 7.1. Процедура Ordering.

Вход: указатель на начало списка s, указатель на конец списка t, указатель на тот элемент u, где может быть нарушен порядок.

Выход: переупорядоченный список

if $u \neq s$ & $u.l.d > u.d$ **then**
  $v := u.l$ // $v.d > u.d$ — нарушение порядка
  $z := u$; **while** $v.d > z.d$ **do** $z := z.r$ **end while** // $v.d \leqslant z.r.d$
  $w := v$; **while** $w.d > u.d$ **do** $w := w.l$ **end while** // $w.l.d \leqslant u.d$
  $v.r := z.r$; $z.r.l := v$; $u.l := w.r$; $w.l.r := u$; $z.r := w$; $w.l := z$
end if

Алгоритм 7.2. Построение графа по степенной последовательности

Вход: натуральное число p и двунаправленный упорядоченный список натуральных чисел, представленный указателем t.

Выход: **false**, если последовательность D не степенная, иначе **true**;

граф G : **array** $[1..q]$ **of struct** { $b, e : 1..p$ }
$q := 0$ // счётчик рёбер
for i **from** 1 **to** p **do**
  $d := t.d$; $n := t.n$ // последний элемент
  if $d = 0$ **then return true end if** // граф уже построен
  if $d < 0$ **then return false end if** // граф нельзя построить
  $u := t.l$; $u.r :=$**nil**; $\sim t$; $t := u$ // откладываем вершину t
  for j **from** 1 **to** d **do**
    $u.d := u.d - 1$; $q := q + 1$; $G[q] := u$ // добавляем ребро
  end for
  Ordering(s, t, u) // восстанавливаем порядок, если нужно
end for

7.3. Виды графов и операции над графами

В данном разделе рассматриваются различные частные случаи графов и вводятся операции над графами и их элементами. Заметим, что не все используемые нами обозначения операций над графами являются традиционными и общепринятыми.

7.3.1. Полные и пустые графы

Граф, состоящий из одной вершины, называется *тривиальным*. Граф, состоящий из простого цикла с k вершинами, обозначается C_k.

Пример. C_3 — *треугольник.*

Граф, в котором любые две вершины смежны, называется *полным*. Полный граф с p вершинами обозначается K_p, он имеет максимально возможное число рёбер:

$$q(K_p) = p(p-1)/2.$$

Граф, в котором нет смежных вершин, называется *пустым*. Пустой граф с p вершинами обозначается $\overline{K_p}$, он имеет минимально возможное число рёбер:

$$q(\overline{K_p}) = 0.$$

Дополнением полного графа является пустой и обратно.

Максимальный полный подграф (некоторого графа) называется *кликой* (этого графа). Максимальным подграф является в том смысле, что любой другой подграф графа, включающий множество вершин клики, не является полным. Граф может иметь несколько клик. Максимальное число вершин среди клик данного графа называется *кликовым числом* графа (или *плотностью*).

Пример. На рис. 7.14 изображён граф и две его клики, рёбра клик на диаграмме отмечены утолщёнными и пунктирными линиями. Кликовое число в данном случае равно четырём.

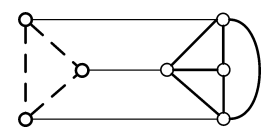


Рис. 7.14. Граф и две его клики

7.3.2. Двудольные графы

Граф $G(V, E)$ называется *двудольным* (или *биграфом*, или *чётным* графом), если множество V может быть разбито на два непересекающихся множества V_1 и V_2 ($V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$), причём всякое ребро из E инцидентно вершине из V_1 и вершине из V_2 (то есть соединяет вершину из V_1 с вершиной из V_2). Множества V_1 и V_2 называются *долями* двудольного графа. Если двудольный граф содержит все рёбра, соединяющие множества V_1 и V_2, то он называется *полным двудольным графом*. Если $|V_1| = m$ и $|V_2| = n$, то полный двудольный граф обозначается $K_{m,n}$.

Примеры

На рис. 7.5 приведены три диаграммы полного двудольного графа $K_{3,3}$. На рис. 7.15 приведена диаграмма другого двудольного графа.

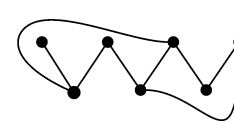


Рис. 7.15. Диаграмма двудольного графа

ТЕОРЕМА. *Граф является двудольным тогда и только тогда, когда в нём нет простых циклов нечётной длины.*

Доказательство. Достаточно рассматривать только связные графы, поскольку каждую компоненту связности можно рассматривать отдельно.

[$\Longrightarrow$] От противного. Пусть $G(V_1,V_2;E)$ — двудольный граф и $v_1, v_2, \ldots, v_{2k+1}$, v_1 — простой цикл нечётной длины. Пусть $v_1 \in V_1$, тогда $v_2 \in V_2$, $v_3 \in V_1$, $v_4 \in V_2, \ldots, v_{2k+1} \in V_1$. Имеем: $v_1, v_{2k+1} \in V_1$ и $(v_1, v_{2k+1}) \in E$, что противоречит двудольности.

[$\Longleftarrow$] Разобьём множество V на подмножества V_1 и V_2 с помощью следующей процедуры.

Вход: граф $G(V, E)$.
Выход: Множества V_1 и V_2 — доли графа.

```
select v ∈ V  // произвольная вершина
V1 := v  // в начале первая доля содержит v,...
V2 := ∅  // ...а вторая пуста
for u ∈ V − v do
  if d(v,u) — чётно then
    V1 := V1 + u  // помещаем вершину u в первую долю
  else
    V2 := V2 + u  // помещаем вершину u во вторую долю
  end if
end for
```

Далее от противного. Пусть есть две вершины в одной доле, соединённые ребром. Пусть для определённости $u, w \in V_2$ и $(u, w) \in E$. Рассмотрим геодезические $\langle v, u\rangle$ и $\langle v, w\rangle$ (здесь v — та произвольная вершина, которая использовалась в алгоритме построения долей графа). Тогда длины $|\langle v, u\rangle|$ и $|\langle v, w\rangle|$ нечётны. Эти геодезические имеют общие вершины (по крайней мере, вершину v). Рассмотрим наиболее удалённую от v общую вершину геодезических $\langle v, u\rangle$ и $\langle v, w\rangle$ и обозначим её v' (может оказаться так, что $v = v'$). Имеем $|\langle v', u\rangle| + |\langle v', w\rangle| = |\langle v, u\rangle| + |\langle v, w\rangle| - 2|\langle v, v'\rangle|$ — чётно и $v', \ldots, u, w, \ldots, v'$ — простой цикл нечётной длины, что противоречит условию. Если же $u, w \in V_1$, то длины $|\langle v, u\rangle|$ и $|\langle v, w\rangle|$ чётны, и имеем $v', \ldots, u, w, \ldots, v'$ — простой цикл нечётной длины. □

Следствие. *Ациклические графы двудольны.*

Пример. На рис. 7.16 представлены диаграммы всех связных двудольных графов с пятью вершинами.

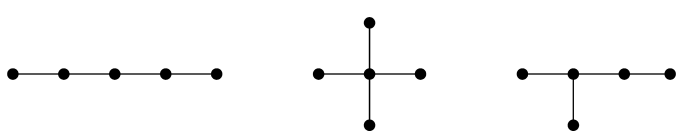


Рис. 7.16. Диаграммы связных двудольных графов

7.3.3. Направленные орграфы и сети

Если в графе ориентировать все рёбра, то получится орграф, который называется *направленным*, или *антисимметричным*. Направленный орграф, полученный из полного графа, называется *турниром*.

ЗАМЕЧАНИЕ

В антисимметричном орграфе не может быть «встречных» дуг (u, v) и (v, u), а в произвольном орграфе такое допустимо.

ОТСТУПЛЕНИЕ

Название «турнир» имеет следующее происхождение. Рассмотрим спортивное соревнование для пар участников (или пар команд), где не предусматриваются ничьи. Пометим вершины орграфа участниками и проведем дуги от победителей к побеждённым. В таком случае турнир в смысле теории графов — это как раз результат однокругового турнира в спортивном смысле.

Если в орграфе полустепень захода некоторого узла равна нулю ($d^+(v) = 0$), то такой узел называется *источником*, если же нулю равна полустепень исхода ($d^-(v) = 0$), то узел называется *стоком*. Направленный слабосвязный (см. п. 8.5.1) орграф с одним источником и одним стоком называется *сетью*.

7.3.4. Операции над графами

Введем следующие операции над графами:

1. *Дополнением графа* $G_1(V_1, E_1)$ (обозначение — $\overline{G}_1(V_1, E_1)$) называется граф $G_2(V_2, E_2)$, где $V_2 = V_1$ & $E_2 = \overline{E_1} = \{e \in V_1 \times V_1 \mid e \notin E_1\} = V \times V \setminus E$.

   Пример. $\overline{K_1} = K_1$.

2. *Объединение (дизъюнктное) графов* $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ (обозначение — $G_1(V_1, E_1) \cup G_2(V_2, E_2)$ при условии $V_1 \cap V_2 = \varnothing$) даёт граф $G(V, E)$, где $V = V_1 \cup V_2$ & $E = E_1 \cup E_2$.

   Пример. $\overline{K}_{3,3} = C_3 \cup C_3$.

3. *Соединение графов* $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ (обозначение — $G_1(V_1, E_1) + G_2(V_2, E_2)$, при условии $V_1 \cap V_2 = \varnothing$) даёт граф $G(V, E)$, где $V = V_1 \cup V_2$ & $E = E_1 \cup E_2 \cup \{e = (v_1, v_2) \mid v_1 \in V_1$ & $v_2 \in V_2\}$.

   Пример. $K_{3,3} = \overline{C}_3 + \overline{C}_3$.

4. *Удаление вершины* v из графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) - v$ при условии $v \in V_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1 - v$ & $E_2 = E_1 \setminus \{e = (v_1, v_2) \mid v_1 = v \vee v_2 = v\}$.

   Пример. $C_3 - v = K_2$.

5. *Удаление ребра* e из графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) - e$ при условии $e \in E_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1$ & $E_2 = E_1 - e$.

   Пример. $K_2 - e = \overline{K}_2$.

6. *Добавление вершины* v в граф $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) + v$ при условии $v \notin V_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1 + v$ & $E_2 = E_1$.

Пример. $K_2 + v = K_2 \cup K_1$.

7. *Добавление ребра* e в граф $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) + e$ при условии $e \notin E_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1$ & $E_2 = E_1 + e$.
8. *Стягивание правильного подграфа* с множеством вершин A графа $G_1(V_1, E_1)$ к вершине v (обозначение — $G_1(V_1, E_1)/A$ при условии $A \subset V_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = (V_1 \setminus A) + v$,
$E_2 = E_1 \setminus \{e = (u, w) \mid u \in A \vee w \in A\} \cup \{e = (u, v) \mid u \in \Gamma(A) \setminus A\}$.

Пример. $K_4/C_3 = K_2$.

9. *Размножение вершины* v графа $G_1(V_1, E_1)$ (обозначение — $G_1(V_1, E_1) \uparrow v$ при условии $v \in V_1, v' \notin V_1$) даёт граф $G_2(V_2, E_2)$, где $V_2 = V_1 + v'$,
$E_2 = E_1 \cup \{(v, v')\} \cup \{e = (u, v') \mid u \in \Gamma^+(v)\}$.

Пример. $K_2 \uparrow v = C_3$.

Некоторые из примеров, приведённых в определениях операций, нетрудно обобщить. В частности, легко показать, что имеют место следующие соотношения:

$$K_{m,n} = \overline{K}_m + \overline{K}_n, \quad K_{p-1} = K_p - v,$$
$$G + v = G \cup K_1, \quad K_{p-1} = K_p/K_2,$$
$$K_p/K_{p-1} = K_2, \quad K_p = K_{p-1} \uparrow v.$$

Введённые операции обладают рядом простых свойств, которые легко вывести из определений. В частности:

$$G_1 \cup G_2 = G_2 \cup G_1, \qquad G_1 + G_2 = G_2 + G_1,$$
$$G_1 = G_2 \Longleftrightarrow \overline{G}_1 = \overline{G}_2, \qquad \overline{G_1 \cup G_2} = \overline{G}_1 + \overline{G}_2.$$

ЗАМЕЧАНИЕ

Операции добавления и удаления ребра взаимно обратны:

$$\forall\, e \in E, e' \notin E \ \left((G - e) + e = (G + e') - e' = G\right).$$

В то же время $\forall\, v' \notin V \ ((G + v') - v' = G)$, но если вершина v графа G не изолированная, то $(G - v) + v \neq G$, потому что в этом случае $q((G - v) + v) < q(G)$.

Результат выполнения нескольких последовательных операций удаления или добавления вершин или рёбер не зависит от порядка выполнения операций:

$$(G + v_1) + v_2 = (G + v_2) + v_1, \qquad (G - v_1) - v_2 = (G - v_2) - v_1,$$
$$(G + e_1) + e_2 = (G + e_2) + e_1, \qquad (G - e_1) - e_2 = (G - e_2) - e_1.$$

Поэтому операции удаления и добавления вершин и рёбер можно обобщить и допускать в качестве второго аргумента множества вершин и рёбер.

ОТСТУПЛЕНИЕ

Приведённые определения операций над графами и примеры к ним дают повод затронуть один тонкий вопрос, связанный с различиями в традициях математических и программных обозначений. Рассмотрим пример к операции дизъюнктного объединения графов

$$\overline{K}_{3,3} = C_3 \cup C_3.$$

Если введённые обозначения понимать буквально, то этот пример кажется противоречащим определению. Действительно, определение требует, чтобы множества вершин объединяемых графов не пересекались. Если считать, что C_3 обозначает конкретный треугольник, то придётся признать, что в выражении $C_3 \cup C_3$ множества вершин не только пересекаются, но и совпадают, а значит, операция объединения неприменима. На самом деле C_3 обозначает как класс треугольников (все треугольники изоморфны как графы), так и отдельный объект — *экземпляр* этого класса. Как именно следует понимать обозначение, считается ясным из контекста. Если стремиться к (излишней в данном случае) строгости и однозначности обозначений, то приведённый пример можно было бы записать, например, так:

$$\forall\, G_1(V_1, E_1) \in C_3, G_2(V_2, E_2) \in C_3 \ \left(V_1 \cap V_2 \neq \varnothing \Longrightarrow \exists\, G_3(V_3, E_3) \in K_{3,3} \ \left(G_1 \cup G_2 \sim \overline{G}_3\right)\right).$$

Подобная неоднозначность обозначений присуща и программированию, хотя и в меньшей степени. Например, ключевое слово **int** в различных контекстах может обозначать встроенный тип данных (класс объектов), операцию порождения нового объекта этого типа (экземпляра класса), операцию преобразования другого объекта в объект типа **int** (явное приведение). Следуя стилю объектно-ориентированных языков программирования, приведённый пример можно было бы записать так:

$$\overline{\textbf{new}K_{3,3}} = \textbf{new}C_3 \cup \textbf{new}C_3.$$

Ради краткости и простоты изложения в этой книге принят значительно менее строгий стиль обозначений в надежде на то, что программистская интуиция и здравый смысл позволят читателю избежать заблуждений, несмотря на вольности в обозначениях.

7.3.5. Простые циклы

Важный и полезный подкласс графов образуют простые циклы C_n.

ТЕОРЕМА 1. *Регулярный граф степени* 2 *состоит из простых циклов.*

Доказательство. Рассмотрим сначала случай связного графа. Пусть $G(V, E)$ — связный граф и $\forall\, v \in V \ \ (d(v) = 2)$. Рассмотрим произвольную вершину $v \in V$. У неё есть две смежные вершины. Пусть одна из них — вершина u. Перейдём из вершины v в вершину u по ребру (v, u). Тогда у вершины u есть ровно одна несмежная и ранее не пройденная вершина w. Перейдем в неё по ребру (u, w). Таким образом переходим единственным образом каждый раз по новому ребру либо в ранее непройденную вершину, либо в начальную вершину v. Действительно, поскольку переходы однозначные, недостижение какой-либо вершины противоречит связности графа. Таким образом, связный регулярный граф степени 2 — это простой цикл. Рассмотрим теперь несвязный граф. Он состоит из k компонент связности, каждая из которых является простым циклом. □

ТЕОРЕМА 2. *В ациклическом графе рёбер меньше, чем вершин:* $z(G) = 0 \Longrightarrow q(G) < p(G)$.

Доказательство. Рассмотрим сначала случай связного графа. Пусть $G(V, E)$ — связный ациклический граф. Индукция по p. База: при $p = 1$ имеем $q = 0$, при $p = 2$ имеем $q = 1$. Пусть утверждение теоремы верно для всех графов с числом вершин меньше p. Рассмотрим связный ациклический граф с p вершинами. Если в графе есть висячая вершина v, то удалим её, $G := G - v$. Тогда $q - 1 < p - 1$ по индукционному предположению, и значит $q < p$. Если висячих вершин нет, то $\forall\, v \in V \;\; (d(v) \geqslant 2)$. По образцу построения в доказательстве предыдущей теоремы возьмём произвольную вершину и перейдём в смежную вершину по ранее не пройденному ребру. Это всегда возможно, поскольку степень любой вершины не меньше двух. Ввиду конечности множества вершин, рано или поздно попадём в одну из ранее пройденных вершин, то есть выявим цикл — противоречие. Рассмотрим теперь несвязный ациклический граф. Его компонентами являются связные ациклические графы, в каждом из которых число ребёр меньше числа вершин. □

7.3.6. Рёберные графы

Пусть $G(V, E)$ — граф без петель. Составим квадратную матрицу $I(G)$: **array** $[1..q, 1..q]$ **of** 0..1, в которой и строки, и столбцы отвечают рёбрам G, и определим её элементы следующим образом:
$I(G)[x, y] :=$ **if** $\exists\, v \in V \;\; (v \in x \;\&\; v \in y)$ **then** 1 **else** 0 **end if**. Другими словами, если рёбра x и y смежны и различны, то в соответствующей клетке матрицы ставим 1, иначе 0. Такую матрицу называют *матрицей смежности рёбер*.

Рёберным графом для (p, q)-графа $G(V, E)$ (обозначение $I(G)$) называется граф с q вершинами, соответствующими рёбрам исходного графа, причём вершины рёберного графа смежны тогда и только тогда, когда смежны соответствующие рёбра исходного графа. Таким образом, матрица смежности рёбер является матрицей смежности рёберного графа.

ЗАМЕЧАНИЕ

Построить диаграмму рёберного графа $I(G)$ по диаграмме графа G очень просто. Достаточно поставить по одной точке в произвольном месте каждой линии, изображающей ребро G, и затем соединить попарно все точки, которые соответствуют смежным ребрам.

Мы видим, что для всякого графа существует соответствующий ему рёберный граф. Возникает естественный вопрос: как узнать, является ли данный граф рёберным графом некоторого другого графа? Ответ на вопрос дает следующая теорема.

ТЕОРЕМА. *Граф $I(G)$ является рёберным графом для некоторого графа G тогда и только тогда, когда он представим в виде множества рёберно непересекающихся клик $K_{d(v)}$ с $d(v)$ вершинами, таких, что $K_{d(v)}$ имеет не более одной общей вершины с любой другой кликой, причём вершина может входить не более чем в две клики.*

Доказательство.

[$\Longrightarrow$] Пусть $I(G)$ — рёберный граф для графа $G(V, E)$. Рассмотрим произвольную вершину $v \in V$. Ей инцидентно $d(v)$ рёбер. Рассмотрим любую вершину $u \in \Gamma(v)$, и пусть $e = (v, u)$. Тогда в $I(G)$ вершина e, соответствующая ребру e графа G, соединяется ребрами с $d(v) - 1$ вершинами, соответствующими остальным ребрам G, инцидентным v. То есть в $I(G)$ имеем полный подграф $K_{d(v)}$, порожденный вершиной v графа G. Рассмотрим теперь вершину u. Проведя для неё аналогичные

рассуждения, приходим к тому, что e соединена ребрами также с $d(u)-1$ вершинами, которые вместе с e образуют другой полный граф $K_{d(u)}$. Два графа, $K_{d(v)}$ и $K_{d(u)}$, имеют ровно 1 общую вершину, определяемую единственным ребром e, связывающим v и u (рис. 7.17).

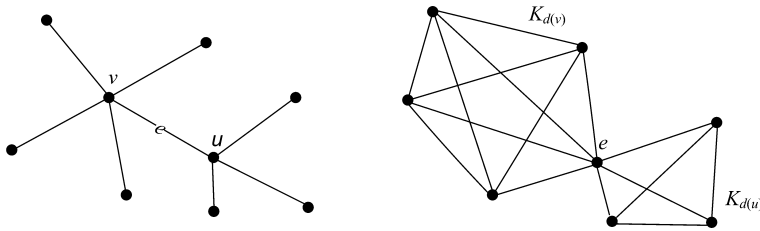


Рис. 7.17. К доказательству теоремы п. 7.3.6

[$\Longleftarrow$] Предположим, что, наоборот, граф G_1 представим в виде множества рёберно непересекающихся клик $U(e_i)$, и любая пара этих клик $(U(e_i), U(e_j))$ имеет не более одной общей вершины, причём вершина может входить не более чем в две клики. Тогда G_1 можно рассматривать как рёберный граф $I(G)$ некоторого графа G. Ясно, что каждая клика $U(e_i)$ имеет $d_i \leqslant d(e_i)+1$ общих вершин с другими кликами $U(e_j)$. Каждому $U(e_i)$ поставим в соответствие одну вершину v_i и соединим v_i и v_j ребром в G тогда и только тогда, когда $U(e_i)$ и $U(e_j)$ имеют общую вершину. К этим рёбрам добавим $d(e_i)-d_i$ рёбер (v_i, u), идущих к новым вершинам u, в которых существует только одно это ребро. □

Говорят, что два графа $G_1(E_1, V_1)$ и $G_2(E_2, V_2)$ *рёберно изоморфны*, если существует такое взаимно однозначное соответствие $E_1 \leftrightarrow E_2$ между их рёбрами, что если x_1 и y_1 — смежные рёбра в G_1, то соответствующие им рёбра x_2 и y_2 смежны в G_2, и наоборот. Заметим, что два графа рёберно изоморфны тогда и только тогда, когда их рёберные графы изоморфны. На рис. 7.18 представлены два графа, изоморфные рёберно, но не изоморфные в обычном смысле.

Как мы выяснили ранее, любой граф G определяет, причём единственным образом, соответствующий рёберный граф $I(G)$. Возникает вопрос, однозначно ли определяет рёберный граф $I(G)$ исходный граф G? Рассмотренный пример — треугольник C_3 и трёхконечная звезда S_3 (рис. 7.18) — показывает, что это не всегда так.

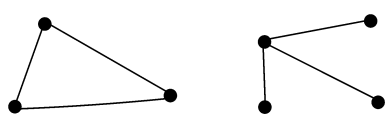


Рис. 7.18. Два графа с одинаковыми рёберными графами

ТЕОРЕМА (Уитни)**.** *Любой связный конечный граф, отличный от C_3 и S_3, однозначно задаётся своим рёберным графом.*

Доказательство. Без доказательства. □

7.4. Представление графов в программах

Следует еще раз подчеркнуть, что конструирование структур данных для представления в программе объектов математической модели — это основа искусства практического программирования. Мы приводим четыре различных базовых представления графов. Выбор наилучшего представления определяется требованиями конкретной задачи. Более того, на практике используются, как правило, некоторые комбинации или модификации указанных представлений, общее число которых необозримо. Но все они так или иначе основаны на тех базовых идеях, которые описаны в этом разделе.

7.4.1. Требования к представлению графов

Известны различные способы представления графов в памяти компьютера, которые различаются объёмом занимаемой памяти и скоростью выполнения операций над графами. Представление выбирается исходя из потребностей конкретной задачи. Далее приведены четыре наиболее часто используемых представления с указанием характеристики $n(p,q)$ — объёма памяти для каждого представления. Здесь p — число вершин, а q — число рёбер.

ЗАМЕЧАНИЕ

Значение характеристики $n(p,q)$ указывается с помощью символа O, который означает совпадение по порядку величины (или равенство с точностью до мультипликативной константы c, последнее замечание в п. 1.3.2). Применительно к измерению занимаемой памяти использование символа O связано с тем, что память может быть измерена в битах, байтах, машинных словах или иных единицах. Коэффициент c при этом меняется, а порядок величины остаётся.

Представления иллюстрируются на конкретных примерах графа G и орграфа D, диаграммы которых представлены на рис. 7.19.

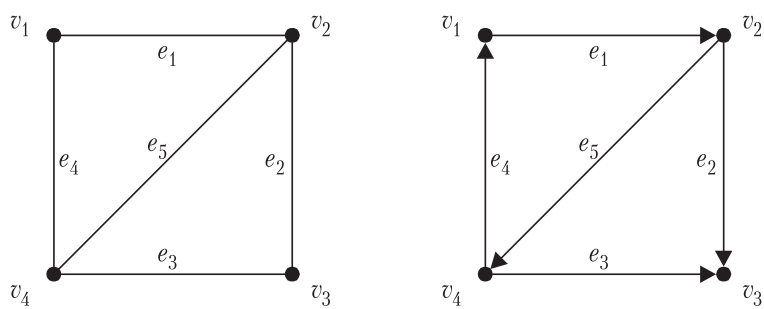


Рис. 7.19. Диаграммы графа (*слева*) и орграфа (*справа*), используемых в качестве примеров

7.4.2. Матрица смежности

Представление графа с помощью матрицы M : **array** $[1..p, 1..p]$ **of** 0..1, отражающей смежность вершин, называется *матрицей смежности*, где $M[i,j] := (v_i \in \Gamma(v_j))$, то есть вершины v_i и v_j смежны. Для матрицы смежности $n(p,q) = O(p^2)$.

Пример.

$$G: \begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix} \qquad D: \begin{vmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{vmatrix}$$

ЗАМЕЧАНИЕ

Матрица смежности графа симметрична относительно главной диагонали, поэтому достаточно хранить только верхнюю (или нижнюю) треугольную матрицу.

7.4.3. Матрица инциденций

Представление графа с помощью матрицы H : **array** $[1..p, 1..q]$ **of** 0..1, отражающей инцидентность вершин и рёбер, называется *матрицей инциденций*,
где $H[i,j] := \exists\, e_j \in E\ \ (v_i \in e_j)$, то есть вершина v_i инцидентна ребру e_j.

Представление орграфа с помощью матрицы H : **array** $[1..p, 1..q]$ **of** $-1..1$, отражающей инцидентность узлов и дуг, также называется *матрицей инциденций*, где $H[i,j] :=$ **if** $\exists\, e_j \in E\ \ (e_j = (x, v_i))$ **then** 1 **else if** $\exists\, e_j \in E\ \ (e_j = (v_i, x))$ **then** -1 **else** 0 **end if**, то есть если узел v_i инцидентен дуге e_j и является её концом, то $H[i,j] = 1$, а если узел v_i инцидентен дуге e_j и является её началом, то $H[i,j] = -1$, если же узел v_i и ребро e_j не инцидентны, то $H[i,j] = 0$. Для матрицы инциденций $n(p,q) = O(pq)$.

Пример.

$$G: \begin{vmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{vmatrix} \qquad D: \begin{vmatrix} -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 \end{vmatrix}$$

ЗАМЕЧАНИЕ

Для связных графов $q \geqslant p - 1$, поэтому матрица смежности несколько компактнее матрицы инциденций.

7.4.4. Списки смежности

Представление графа с помощью списочной структуры, отражающей смежность вершин и состоящей из массива указателей Γ : **array** $[1..p]$ **of** $\uparrow N$ на списки смежных вершин, где элемент списка является структурой $N =$ **struct { **$v : 1..p, n :\uparrow N$** }**, называется *списком смежности*. В случае представления неориентированных графов списками смежности $n(p,q) = O(p + 2q)$, а в случае ориентированных графов $n(p,q) = O(p + q)$.

ЗАМЕЧАНИЕ

Массив Γ также можно представить списком.

Пример. Списки смежности для графа G и орграфа D представлены на рис. 7.20.

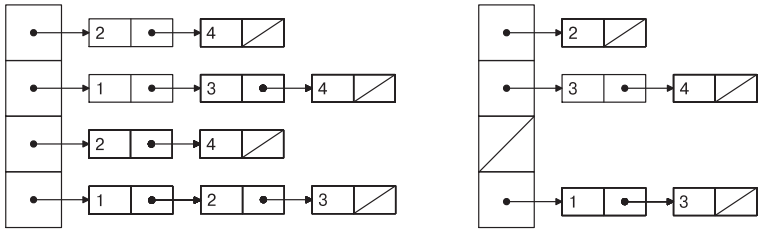


Рис. 7.20. Списки смежности для графа G (*слева*) и орграфа D (*справа*)

7.4.5. Массив дуг

Представление графа с помощью массива структур
E : **array** $[1..q]$ **of struct {** $b, e : 1..p$**}**, отражающего список пар смежных вершин (или, для орграфов, узлов), называется *массивом рёбер* (*массивом дуг*). Для массива рёбер (или дуг) $n(p, q) = O(2q)$.

ЗАМЕЧАНИЕ

Для представления графов с изолированными вершинами может понадобиться хранить ещё и число p, если только система программирования не позволяет извлечь значение p из массива структур E.

Пример. Представление с помощью массива рёбер (дуг) показано в следующей таблице (для графа G — слева, а для орграфа D — справа).

b	e	b	e
1	2	1	2
1	4	2	3
2	3	2	4
2	4	4	1
3	4	4	3

ЗАМЕЧАНИЕ

Указанные представления пригодны для графов и орграфов, а после некоторой модификации — также и для псевдографов, мультиграфов и гиперграфов.

7.4.6. Обходы графов

Обход графа — это некоторое систематическое перечисление его вершин и (или) рёбер. Наибольший интерес представляют обходы, использующие локальную информацию (списки смежности). Среди всех обходов наиболее известны поиск *в ширину* и *в глубину*. Алгоритмы поиска в ширину и в глубину лежат в основе многих конкретных алгоритмов на графах. Алгоритмы поиска в ширину и в глубину изложены во множестве источников, одно из самых подробных и детальных изложений приведено в фундаментальной книге [2]. Определение поиска в ширину и в глубину обычно даётся не постулированием требуемых свойств обходов, но алгоритмически,

предъявлением конкретных алгоритмов. Это обстоятельство, а также многочисленные технические детали изложения несколько затуманивают основную идею, и может сложиться впечатление, что поиск в ширину и в глубину — это принципиально различные алгоритмы: например, поиск в ширину — итеративный алгоритм, а поиск в глубину — рекурсивный. На самом деле, хотя поиск в ширину и в глубину действительно дают различные обходы, по существу это две реализации одного алгоритма, применяющие различные структуры данных, как показано в алгоритме 7.3.

Алгоритм 7.3. Поиск в ширину и в глубину

Вход: граф $G(V, E)$, представленный списками смежности Γ.
Выход: последовательность вершин обхода.

```
for v ∈ V do
  x[v] := 0  // вначале все вершины не отмечены
end for
select v ∈ V  // начало обхода — произвольная вершина
v → T  // помещаем v в структуру данных T...
x[v] := 1  // ...и отмечаем вершину v
repeat
  u ← T  // извлекаем вершину из структуры данных T...
  yield u  // ...и возвращаем её в качестве очередной пройденной вершины
  for w ∈ Γ(u) do
    if x[w] = 0 then
      w → T  // помещаем w в структуру данных T...
      x[w] := 1  // ...и отмечаем вершину w
    end if
  end for
until T = ∅
```

Если в алгоритме 7.3. структура данных T — это стек (LIFO — Last In First Out), то обход называется *поиском в глубину*. Если T — это очередь (FIFO — First In First Out), то обход называется *поиском в ширину*.

ЗАМЕЧАНИЕ

Это изложение алгоритмов заметно отличается от изложения в книге [2]. Здесь не используется трёх цветов вершин, и при буквальном сравнении протоколов работы алгоритмов поиска в глубину получаются различные обходы. В результате у внимательного читателя может создаваться ошибочное впечатление, что приведённый алгоритм «неправильный». На самом деле это не так. В случае поиска в ширину различий между приведённым алгоритмом 7.2 и обычным изложением алгоритма поиска в ширину нет. Различие между приведённым алгоритмом 7.2 в случае поиска в глубину и обычным рекурсивным алгоритмом поиска в глубину заключается только в порядке рассмотрения вершин w, смежных с текущей вершиной u. В алгоритме 7.2 вершины w заносятся в структуру данных T в том порядке, в каком вершины встречаются в списке смежности $\Gamma(u)$, и, соответственно, рассматриваться будут в обратном порядке, если T — это стек. В рекурсивном варианте вершины будут рассматриваться в порядке следования в списке смежности. Таким образом, различие в том, как просматривать списки смежности: от начала к концу или от конца к началу. На свойства алгоритмов и получаемых обходов это различие не влияет.

Пример. В следующей таблице показаны протоколы поиска в глубину и в ширину для графа, диаграмма которого приведена на рис. 7.21. Предполагается, что

начальной является вершина 1. В первой паре столбцов в таблице приведён протокол поиска в ширину, а во второй паре столбцов — в глубину. Кроме того, для сравнения, в третьем столбце приведен протокол поиска в глубину для случая, когда смежные вершины заносятся в стек в обратном порядке. Тем самым третий столбец показывает, как ведёт себя рекурсивный вариант алгоритма поиска в глубину. На рис. 7.21 сплошные стрелки с номерами рядом с рёбрами показывают движение по графу при поиске в глубину, а пунктирные — в ширину.

u	T	u	T	u	T
1	2,4	1	2,4	1	4,2
2	4,3	4	2,3	2	4,3
4	3	3	2	3	4
3	$\varnothing$	2	$\varnothing$	4	$\varnothing$

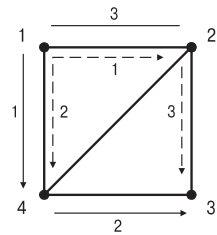


Рис. 7.21. Диаграмма графа к примеру обхода в ширину и в глубину

ТЕОРЕМА. *Если граф G связен и конечен, то поиск в ширину и поиск в глубину обходят все вершины по одному разу за время, пропорциональное не более чем суммарному числу вершин и ребер.*

Доказательство.

[Единственность обхода вершины] Обходятся только вершины, попавшие в T. В T попадают только неотмеченные вершины. При попадании в T вершина отмечается. Следовательно, любая вершина будет обойдена не более одного раза.

[Завершаемость алгоритма] Всего в T может попасть не более p вершин. На каждом шаге одна вершина удаляется из T. Следовательно, алгоритм завершит работу не более чем через p шагов.

[Обход всех вершин] От противного. Пусть алгоритм закончил работу и вершина w не обойдена. Значит, w не попала в T. Следовательно, она не была отмечена. Отсюда следует, что все вершины, смежные с w, не были обойдены и отмечены. Аналогично, любые вершины, связанные с неотмеченными, сами не отмечены (после завершения алгоритма). Но G связен, значит, существует путь $\langle v, w\rangle$. Следовательно, вершина v не отмечена. Но она была отмечена на первом шаге!

[Трудоёмкость алгоритма] Инициализация (вторая строчка) выполняется за $O(p)$. В цикле каждая вершина просматривается не более одного раза, и для каждой просматриваемой вершины её список смежности просматривается один раз. Следовательно, цикл выполняется за $O(q)$. Таким образом, трудоёмкость алгоритма $O(p+q)$. □

СЛЕДСТВИЕ 1. *Пусть $(u_1, \ldots, u_i, \ldots, u_j, \ldots, u_p)$ — обход (то есть последовательность вершин) при поиске в ширину. Тогда $\forall\, i < j\ \ (d(u_1, u_i) \leqslant d(u_1, u_j))$.*

Другими словами, расстояние текущей вершины от начальной является монотонно возрастающей функцией времени поиска в ширину, вершины обходятся в порядке возрастания расстояния от начальной вершины.

СЛЕДСТВИЕ 2. *Пусть* $(u_1, \ldots, u_i, \ldots, u_p)$ *— обход при поиске в глубину. Тогда* $\forall\, i \geqslant 1\ \ (d(u_1, u_i) < i \leqslant p)$.

Другими словами, время поиска в глубину любой вершины не менее расстояния от начальной вершины и не более общего числа вершин, причём в худшем случае время поиска в глубину может быть максимальным, независимо от расстояния до начальной вершины.

СЛЕДСТВИЕ 3. *Пусть* $(u_1, \ldots, u_i, \ldots, u_p)$ *— обход при поиске в ширину, а* $D(u_1, 1)$, $D(u_1, 2), \ldots$ *— ярусы графа относительно вершины* u_1. *Тогда*

$$\forall\, i \geqslant 1 \left(\sum_{j=0}^{d(u_1,u_i)-1} |D(u_1, j)| < i \leqslant \sum_{j=0}^{d(u_1,u_i)} |D(u_1, j)| \right).$$

Другими словами, время поиска в ширину ограничено снизу количеством вершин во всех ярусах, находящихся на расстоянии меньшем, чем расстояние от начальной вершины до текущей, и ограничено сверху количеством вершин в ярусах, начиная с яруса текущей вершины и включая все меньшие ярусы.

СЛЕДСТВИЕ 4. *Пусть* $(u_1, \ldots, u_i, \ldots, u_p)$ *— обход при поиске в ширину, а* $(v_1, \ldots, v_j, \ldots, v_p)$ *— обход при поиске в глубину, где* $u_i = v_j$. *Тогда в среднем* $i = 2j$.

Другими словами, поиск в глубину в среднем вдвое быстрее, чем поиск в ширину (см. рис. 7.22).

Пример. На рис. 7.22 в графической форме показаны протоколы поиска в ширину (слева) и в глубину (справа).

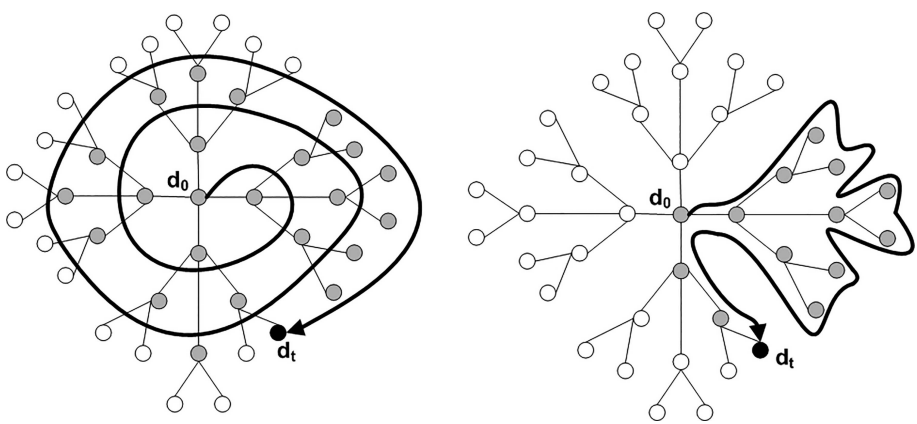


Рис. 7.22. Обходы в ширину и глубину

ЗАМЕЧАНИЕ

Алгоритм поиска в ширину и в глубину может быть применен также к несвязным графам. Действительно, для этого достаточно применить алгоритм к одной компоненте связности,

потом к следующей и т. д. Алгоритм может быть применён также к любым орграфам, в том числе к орграфам, которые не являются сильно связными (см. п. 8.5.1). Для этого достаточно применить алгоритм начиная с произвольного узла, удалить все обойдённые узлы с инцидентными дугами, а затем применить алгоритм к оставшейся части орграфа и т. д.

7.5. Графы и отношения

Целью заключительного раздела данной главы является установление связи теории графов с другими разделами дискретной математики.

7.5.1. Орграфы и бинарные отношения

Любой орграф $G(V, E)$, возможно, с петлями, но без кратных дуг, задаёт бинарное отношение E на множестве V. Обратно, любое бинарное отношение E на множестве V определяет орграф $G(V, E)$: пара элементов принадлежит отношению $(a, b) \in E \subset V \times V$ тогда и только тогда, когда в графе G есть дуга (a, b).

Пример. На рис. 7.23 представлена диаграмма нагруженного орграфа, соответствующего отношению собственной делимости (п. 2.4.1), заданному на множестве $\{2, 3, 4, 5, 6, 7, 8, 9\}$.

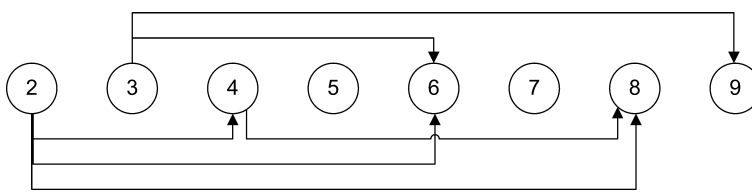


Рис. 7.23. Отношение собственной делимости чисел

Орграф, соответствующий рефлексивному отношению, в каждой вершине имеет петлю, а значит, является псевдографом. Если же отношение антирефлексивно, то орграф вовсе не имеет петель. Если отношение рефлексивно, то рисовать петлю на диаграмме у *каждой* вершины утомительно и неинформативно, поэтому петли на диаграммах графов и орграфов показывают, только если эта информация существенна.

Дополнение орграфов есть дополнение отношений. Изменение направления всех дуг соответствует обратному отношению. Степени бинарного отношения $E \subset V \times V$ соответствуют путям в орграфе. Степень E^1 соответствует путям длины 1, то есть дугам, E^2 соответствует путям длины 2 и т. д. Если отношение антисимметрично, то соответствующий ему орграф не имеет встречных дуг. Антисимметричными являются, в частности, отношения порядка, которые рассматриваются в следующем параграфе с точки зрения теории графов.

Если же отношение симметрично, то всякая дуга орграфа имеет встречную, и направление дуги не имеет значения. Таким образом, всякому симметричному отношению соответствует неориентированный граф, и обратно, граф (неориентированный) соответствует симметричному отношению. Полный граф соответствует универсальному отношению.

Пример. Рассмотрим отношение $R = \{(a, b) \in \mathbb{N} \times \mathbb{N} \mid (a + b) \mod 4 = 0\}$ на множестве $\{2, 3, 4, 5, 6, 7, 8, 9\}$ (рис. 7.24). Это отношение симметрично, поэтому ему удобнее ставить в соответствие граф, а не орграф. Петли на диаграмме этого графа являются существенными, так как отношение не рефлексивно и не антирефлексивно.

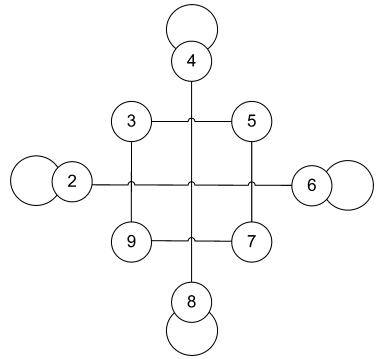


Рис. 7.24. Диаграмма отношения R

Пример. Напомним, что отношение взаимной простоты $R = \{(a, b) \in \mathbb{N} \times \mathbb{N} \mid \gcd(a, b) = 1\}$ на множестве натуральных чисел симметрично. На рис. 7.25 представлена диаграмма графа, соответствующего отношению взаимной простоты на множестве $\{2, 3, 4, 5, 6, 7, 8, 9\}$.

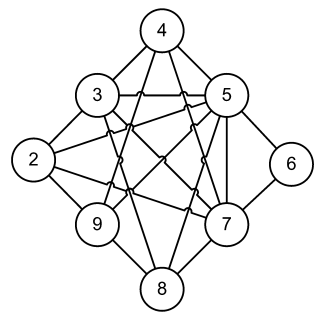


Рис. 7.25. Диаграмма отношения взаимной простоты

ЗАМЕЧАНИЕ

Исходя из названий «граф» и «ориентированный граф» в первый момент может показаться, что класс графов шире класса орграфов. Однако на самом деле, поскольку симметричные отношения являются подклассом класса всех отношений, класс графов является подклассом класса орграфов. Так исторически сложилась терминология.

В то же время следует обратить внимание, что мультиорграф не может задавать бинарное отношение, так как пары в подмножестве прямого произведения двух множеств не повторяются. Другими словами, между языком мультиграфов и языком бинарных отношений нет прямого соответствия.

Функция является частным случаем отношения и может быть представлена орграфом. Рассмотрим функцию $f\colon A \to B$, ей соответствует двудольный орграф $G(A, B, E)$, для которого $V = A \cup B$ и $E = \{(a, b) \in A \times B \mid b = f(a)\}$. Множество всех вершин орграфа G, имеющих исходящие дуги, есть область определения функции, а множество всех вершин, имеющих хотя бы одну входящую дугу, — область значений функции. При этом в силу однозначности любой узел орграфа G имеет не более одной исходящей дуги.

Если же функция является функцией на множестве V, $f\colon V \to V$, то диаграмма орграфа $G(V, E)$ может иметь довольно причудливую форму.

Примеры

На рис. 7.26 слева представлена диаграмма орграфа, соответствующего функции $f(x) = x \bmod 9 + 1$, а справа — функции $g(x) = x \bmod 4 + 1$, действующих на множестве $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Так как функции тотальные, каждый узел имеет ровно одну исходящую дугу.

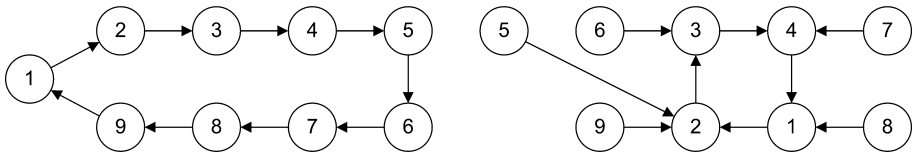


Рис. 7.26. Диаграммы орграфов, соответствующих функциям f и g

ЗАМЕЧАНИЕ

Диаграммы орграфов пригодны для представления только таких функкций, у которых область определения и область значений существенно конечны. Для бесконечно заданных функций, в частности, непрерывных функций вещественной переменной, традиционным и наиболее используемым графическим средством является представление графика функции в виде линии, которая к тому же наглядно иллюстрирует многие свойства этой функции.

7.5.2. Граф инциденций

Отношение инцидентности между вершинами и рёбрами (узлами и дугами) также может быть представлено графом (орграфом) *инциденций*. Для графов отношение инцидентности симметрично, поэтому граф инциденций является двудольным графом (а не орграфом).

Пример. На рис. 7.27 представлены диаграммы графа (слева) и его графа инциденций (справа), для примера из п. 7.4.1.

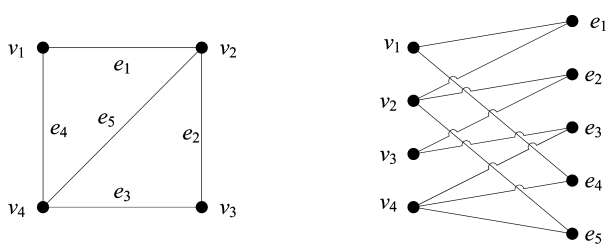


Рис. 7.27. Диаграммы графа и его графа инциденций

Чтобы не потерять информацию о направлении дуг, для орграфов обычно считают, что узел инцидентен исходящим дугам, и входящие дуги инцидентны узлу. Таким образом, для орграфов отношение инциденции задаётся орграфом.

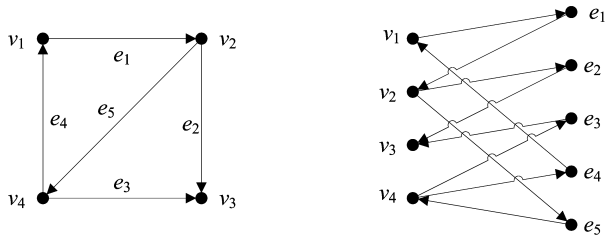


Рис. 7.28. Диаграммы орграфа и его орграфа инциденций

Пример. На рис. 7.28 представлены диаграммы орграфа (слева) и его орграфа инциденций (справа), для примера из п. 7.4.1.

ЗАМЕЧАНИЕ

В графе инциденций степени вершин сохраняются, а степени рёбер равны 2. В орграфе инциденций полустепени узлов сохраняются, а у дуг полустепени равны 1.

Граф инциденций оказывается особенно полезен при рассмотрении отношений, устроенных более сложно, нежели бинарные отношения на конечном множестве.

7.5.3. Гиперграфы и многоместные отношения

В гиперграфах рёбра связывают произвольные непустые подмножества множества вершин. Для гиперграфов нет единой общепринятой нотации диаграмм. На рис. 7.29 приведены три наиболее распространённые. Слева рёбра изображены в виде овалов, охватывающих смежные вершины. При большом количестве вершин и рёбер такая диаграмма трудно читается. Можно рисовать рёбра, имеющие несколько концов (см. рисунок в центре) с помощью специальных «ромбиков», которые считаются частью линии, а не фигурой. Фактически, этот способ задаёт диаграмму графа инциденций гиперграфа (см. рисунок справа).

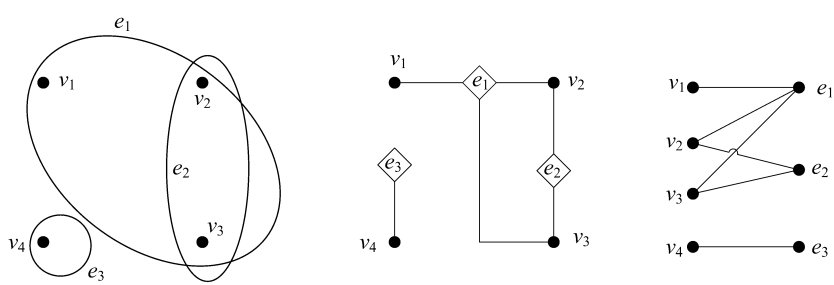


Рис. 7.29. Диаграммы гиперграфов

Заметим, что граф инциденций гиперграфа — это двудольный граф без ограничений на степени вершин.

Аналогичные построения легко провести для гипероргафов.

ОТСТУПЛЕНИЕ

Не следует думать, что гиперграфы и гиперорграфы — это какая-то экзотика. Напротив, такие конструкции часто встречаются в реальных приложениях. Например, *игровые деревья*, известные также как *деревья И/ИЛИ*, которые являются основной структурой данных в игровых программах в *символическом искусственном интеллекте*, фактически являются гиперорграфами специального вида. Другим примером являются блок-схемы, диаграммы UML, диаграммы «сущность-связь» и т. п.

Таким образом, имеется полная аналогия между орграфами и бинарными отношениями — фактически, это один и тот же класс объектов, только описанный разными средствами. Отношения (в частности, функции) являются базовым средством для построения подавляющего большинства математических моделей, используемых при решении практических задач. С другой стороны, графы допускают наглядное представление в виде диаграмм. Этим обстоятельством объясняется широкое использование графов в программировании.

7.5.4. Достижимость и частичное упорядочение

В качестве примера связи между орграфами и бинарными отношениями рассмотрим отношения частичного порядка с точки зрения теории графов. Узел u в орграфе $G(V, E)$ *достижим* из узла v, если существует путь из v в u. Путь из v в u обозначим $\langle v, \overrightarrow{u} \rangle$. Отношение достижимости можно представить матрицей T : **array** $[1..p, 1..p]$ **of** 0..1, где $T[i, j] = 1$, если узел v_j достижим из узла v_i, и $T[i, j] = 0$, если узел v_j недостижим из узла v_i. Рассмотрим отношение строгого частичного порядка $\succ$, которое характеризуется следующими аксиомами.

1. Антирефлексивность: $\forall\, v \in V \;\; (\neg(v \succ v))$.
2. Транзитивность: $\forall\, u, v, w \;\; ((v \succ w \;\&\; w \succ u \Longrightarrow v \succ u))$.
3. Антисимметричность: $\forall\, u, v \;\; (\neg(u \succ v \;\&\; v \succ u))$.

Отношению строгого частичного порядка $\succ \subset V \times V$ можно сопоставить орграф $G(V, E)$, в котором $a \succ b \Longleftrightarrow (a, b) \in E$.

ТЕОРЕМА 1. *Если отношение E есть строгое частичное упорядочение, то орграф $G(V, E)$ не имеет контуров.*

Доказательство. От противного. Пусть в G есть контур. Рассмотрим любую дугу (a, b) в этом контуре. Тогда имеем $a \succ b$, но $b \succ a$ по транзитивности, что противоречит антисимметричности упорядочения. □

ТЕОРЕМА 2. *Если орграф $G(V, E)$ не имеет контуров, то отношение достижимости есть строгое частичное упорядочение.*

Доказательство.

[Антирефлексивность] Нет контуров, следовательно, нет петель.

[Транзитивность] Если существуют пути из v в w и из w в u, то существует и путь из v в u.

[Антисимметричность] От противного. Пусть $\exists\, u, v \;\; (u \succ v \;\&\; v \succ u)$, то есть существует путь $\langle v, \overrightarrow{u} \rangle$ из v в u и путь $\langle u, \overrightarrow{v} \rangle$ из u в v. Следовательно, существует контур вида $\langle u, \overrightarrow{v} \rangle + \langle v, \overrightarrow{u} \rangle$, что противоречит условию. □

ТЕОРЕМА 3. *Если орграф не имеет контуров, то в нем есть узел, полустепень захода которого равна 0.*

Доказательство. От противного. Пусть такого узла нет, тогда для любого узла найдётся узел, из которого есть дуга в данный узел. Следовательно, имеем контур против направления стрелок. □

ЗАМЕЧАНИЕ

Эта теорема позволяет найти минимальный элемент в конечном частично упорядоченном множестве, который требуется в алгоритме топологической сортировки. А именно, минимальный элемент — это источник, то есть узел, которому в матрице смежности соответствует нулевой столбец.

7.5.5. Достижимость и транзитивное замыкание

Если E — бинарное отношение на V, то транзитивным замыканием E^+ на V будет отношение достижимости на орграфе $G(V, E)$.

ТЕОРЕМА. *Пусть M — матрица смежности орграфа $G(V, E)$. Тогда $M^k[i, j] = 1$ в том и только в том случае, если существует путь длиной k из узла v_i в узел v_j.*

Доказательство. Индукция по k. База: $k = 1$, $M^1 = M$ — пути длины 1. Пусть M^{k-1} содержит пути длины $k-1$. Тогда $M^k[i, j] = \bigvee\limits_{l=1}^{p} \left(M^{k-1}[i, l] \;\&\; M[l, j]\right)$, то есть путь длины k из узла i в узел j существует тогда и только тогда, когда найдётся узел l такой, что существует путь длины $k-1$ из i в l и дуга (l, j), то есть $\exists\, \langle \overrightarrow{i, j} \rangle \left(|\langle \overrightarrow{i, j} \rangle| = k\right) \Longleftrightarrow \exists\, l \;\left(\exists\, \langle \overrightarrow{i, l} \rangle \;\left(|\langle \overrightarrow{i, l} \rangle| = k-1 \;\&\; (l, j) \in E\right)\right)$. □

Если T — матрица достижимости, то очевидно, что $T = \bigvee\limits_{k=1}^{p-1} M^k$. Трудоёмкость прямого вычисления по этой формуле составит $O(p^4)$. Матрица достижимости T может быть вычислена по матрице смежности M алгоритмом Уоршалла (п. 1.5.2) за $O(p^3)$.

Глава 8 Связность

В этой главе обсуждается важное для приложений понятие связности, доказывается фундаментальная теорема — теорема Менгера[1] и рассматриваются наиболее популярные алгоритмы поиска кратчайших путей.

8.1. Компоненты связности

В русском языке есть как слово «компонент» мужского рода, так и слово «компонента» женского рода, оба варианта допустимы. В современной языковой практике чаще используется слово мужского рода. Однако исторически сложилось так, что «компонента связности» имеет женский род, и в данном случае мы подчиняемся традиции.

8.1.1. Объединение графов и компоненты связности

ЛЕММА. *В нетривиальном связном графе $G(V, E)$ для любого разбиения множества вершин на непустые блоки U, W существует ребро, инцидентное вершинам из разных блоков:*

$$\forall\, U, W \subset V\ (U \neq \varnothing\ \&\ W \neq \varnothing\ \&\ U \cap W = \varnothing\ \&\ U \cup W = V \Longrightarrow \\ \Longrightarrow \exists\, (u, w) \in E\ \ (u \in U\ \&\ w \in W)).$$

Доказательство. Рассмотрим вершины $u \in U$, $w \in W$ и цепь $\langle u, w\rangle$, которые существуют по условиям леммы. Если цепь $\langle u, w\rangle$ состоит из одного ребра, то это ребро — требуемое. Если цепь состоит из нескольких рёбер, то рассмотрим вершину v на цепи $\langle u, w\rangle$, смежную с вершиной u. Если $v \in W$, то ребро (u, v) — требуемое. В противном случае отщепим от цепи $\langle u, w\rangle$ ребро (u, v), положим $u := v$ и снова рассмотрим цепь $\langle u, w\rangle$, которая удовлетворяет условиям. Будем продолжать процесс отщеплений, пока не получим требуемое ребро. Процесс отщеплений закончится, потому что исходная цепь конечна. □

ТЕОРЕМА. *Граф связен тогда и только тогда, когда его нельзя представить в виде объединения двух графов.*

Доказательство.

[$\Longrightarrow$] От противного. Пусть $k(G) = 1$ и граф G состоит из двух компонент, то есть $G = G_1(V_1, E_1) \cup G_2(V_2, E_2)$, где $V_1 \cap V_2 = \varnothing$, $E_1 \cap E_2 = \varnothing$, $V_1 \neq \varnothing$, $V_2 \neq \varnothing$. Возьмём $v_1 \in V_1$, $v_2 \in V_2$. Тогда $\exists\, \langle v_1, v_2\rangle\ \ (v_1 \in V_1\ \&\ v_2 \in V_2)$. В этой цепи $\exists\, e = (a, b)\ \ (a \in V_1\ \&\ b \in V_2)$. Но тогда $e \notin E_1$, $e \notin E_2$, следовательно, $e \notin E$.

[$\Longleftarrow$] От противного. Пусть $k(G) > 1$. Тогда существуют две вершины, u и v, не соединённые цепью. Следовательно, вершины u и v принадлежат разным компонентам связности. Положим G_1 — компонента связности для u, а G_2 — все остальное. Имеем $G = G_1 \cup G_2$. □

---

[1] Карл Менгер (1902–1985).

ЗАМЕЧАНИЕ

Несвязный граф всегда можно представить как объединение связных компонент. Эти компоненты можно рассматривать независимо. Поэтому во многих случаях можно без ограничения общности предполагать, что рассматриваемый граф связен.

8.1.2. Точки сочленения, мосты и блоки

Вершина графа называется *точкой сочленения*, или *разделяющей вершиной*, если её удаление увеличивает число компонент связности. *Мостом* называется ребро, удаление которого увеличивает число компонент связности. *Блоком* называется связный граф, не имеющий точек сочленения.

Пример. В графе, диаграмма которого представлена на рис. 8.1:

1) вершины u, v — точки сочленения, и других точек сочленения нет;
2) ребро x — мост, и других мостов нет;
3) правильные подграфы $\{a, b, w\}$, $\{b, u, w\}$, $\{a, b, u, w\}$, $\{c, d, v\}$, $\{e, f, v\}$ — блоки, и других блоков нет.

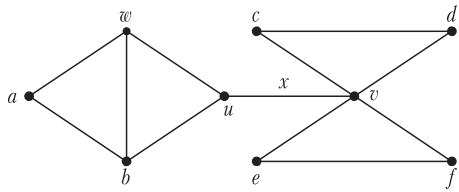


Рис. 8.1. Точки сочленения, мосты и блоки

ЛЕММА. *В любом нетривиальном графе есть по крайней мере две вершины, которые не являются точками сочленения.*

Доказательство. Например, таковыми являются концы любого диаметра. □

ТЕОРЕМА 1. *Пусть $G(V, E)$ — связный граф и $v \in V$. Тогда следующие утверждения эквивалентны:*

1) *v — точка сочленения;*
2) $\exists\, u, w \in V\ (u \neq w\ \&\ \forall\ \langle u, w\rangle_G\ (v \in \langle u, w\rangle_G))$;
3) $\exists U, W\ (U \cap W = \varnothing\ \&\ U \cup W = V - v\ \&\ \forall\, u \in U, w \in W\ (\forall\ \langle u, w\rangle_G\ (v \in \langle u, w\rangle_G)))$.

Доказательство.

[1$\Longrightarrow$3] Рассмотрим $G' := G - v$. Граф G' не связен, $k(G') > 1$, следовательно, G' имеет по крайней мере две компоненты связности, то есть $V - v = U \cup W$, где U — множество вершин одной из компонент связности, а W — все остальные вершины. Пусть $u \in U$, $w \in W$. Тогда $\neg\exists\langle u, w\rangle_{G'}$. Но $k(G) = 1$, следовательно, $\exists\langle u, w\rangle_G$ и, значит, $\forall\ \langle u, w\rangle_G\ \ (v \in \langle u, w\rangle_G)$.

[3$\Longrightarrow$2] Тривиально.

[2$\Longrightarrow$1] Рассмотрим $G' := G - v$. Имеем $\neg\exists\langle u, w\rangle_{G'}$. Следовательно, $k(G') > 1$, откуда v — точка сочленения. □

Пример. На рис. 8.1: v — точка сочленения, при этом $U = \{a, b, u, w\}$, $V = \{c, d, e, f\}$.

СЛЕДСТВИЕ 1. *Если вершина инцидентна мосту и не является висячей, то она является точкой сочленения.*

Доказательство. Пусть (u, v) — мост и $d(v) > 1$. Тогда v смежна с w, причём $w \neq u$. Далее разбор случаев. Если $\exists\, \langle u, w\rangle\ \ (v \notin \langle u, w\rangle)$, то ребро (u, v) принадлежит циклу, что противоречит тому, что (u, v) — мост. Если же $\neg\exists\, \langle u, w\rangle\ \ (v \notin \langle u, w\rangle)$, то $\forall\, \langle u, w\rangle\ \ (v \in \langle u, w\rangle)$ и, значит, v — точка сочленения по пункту 2 теоремы. □

ЗАМЕЧАНИЕ

Отнюдь не всякая точка сочленения является концом моста.

Пример. В условиях предыдущего примера при удалении моста x вершина v остаётся точкой сочленения, и не является концом моста.

СЛЕДСТВИЕ 2. *Если граф $G(V, E)$ связен, а вершина v не является точкой сочленения, то для любых двух других вершин, u и w, существует простая $\langle u, w\rangle$-цепь, не содержащая вершину v.*

Доказательство. По пункту 2 теоремы 1. □

ТЕОРЕМА 2. *Пусть $G(V, E)$ — связный граф и $x \in E$. Тогда следующие утверждения эквивалентны:*

1) *x — мост;*
2) *x не принадлежит ни одному простому циклу;*
3) $\exists\, u, w \in V\ \ (\forall\, \langle u, w\rangle_G\ \ (x \in \langle u, w\rangle_G))$;
4) $\exists\, U, W\ (U \cap W = \varnothing\ \&\ U \cup W = V\ \&\ \forall\, u \in U\, (\forall\, w \in W\, (\forall\ \langle u, w\rangle_G\ \ (x \in \langle u, w\rangle_G))))$.

Доказательство. Аналогично теореме 1. □

Графы K_1 и K_2 формально можно считать блоками, поскольку у них нет точек сочленения. Эти блоки уместно назвать *тривиальными*, поскольку они удовлетворяют определению, но не обладают замечательными свойствами, присущими нетривиальным блокам. В частности, нетрудно видеть, что в нетривиальном блоке не только нет точек сочленения, но также нет мостов, и нет висячих вершин. Более того, все вершины блока связаны неединственным образом, что устанавливает следующая теорема.

ТЕОРЕМА 3. *Если граф $G(V, E)$ связен и $p(G) > 2$, то следующие утверждения эквивалентны:*

1) *G — нетривиальный блок;*
2) *любые две вершины принадлежат одному простому циклу;*

3) *любая вершина и любое ребро принадлежат одному простому циклу;*
4) *любые два ребра принадлежат одному простому циклу;*
5) *для любых двух вершин и любого ребра существует простая цепь, соединяющая эти вершины и содержащая это ребро;*
6) *для любых трех вершин существует простая цепь, соединяющая две вершины и содержащая третью.*

Доказательство.

[1$\Longrightarrow$ 2] Пусть u и w — любые две вершины. Рассмотрим U — множество всех вершин, входящих во все простые циклы, содержащие вершину u. Ясно, что $U \neq \varnothing$, поскольку вершина u не является висячей, поэтому для любых двух вершин, u_1 и u_2, смежных с вершиной u, по следствию 2 к теореме 1 существует соединяющая их цепь, не содержащая вершины u, и, значит, существует некоторый простой цикл, содержащий вершину u. Положим $W := V \setminus U$ (рис. 8.2). Далее от противного. Пусть $W \neq \varnothing$ и $w \in W$. Граф связен, и, значит, в нем есть ребро (s,t) такое, что $s \in U$, $t \in W$ (лемма п. 8.1.1). Поскольку $u \in U$ и $s \in U$, обе эти вершины принадлежат некоторому простому циклу Z. По следствию 2 существует простая $\langle u,t\rangle$-цепь P, не содержащая вершину s. Обозначим через v первую вершину, считая от вершины t, на цепи P, которая также принадлежит циклу Z. Тогда рассмотрим следующую последовательность простых цепей: отрезок цепи P от вершины t до вершины v, отрезок цикла Z от вершины v до вершины u, отрезок цикла Z от вершины u до вершины s, ребро (s,t). Это цикл, причём простой, и вершины u и t обе принадлежат этому циклу и входят в множество U, что противоречит выбору вершины t и предположению $w \in W$.

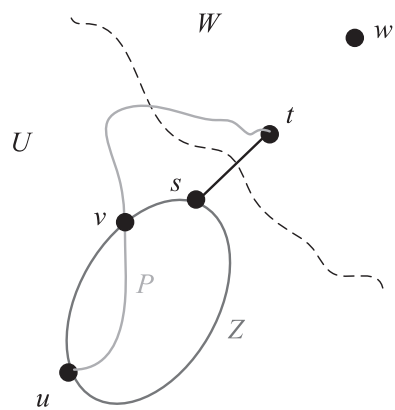


Рис. 8.2. К доказательству пункта [1$\Longrightarrow$2] теоремы о блоках

[2$\Longrightarrow$3] Пусть u — любая вершина и (v,w) — любое ребро. Вершины u и v принадлежат некоторому простому циклу Z (рис. 8.3). Если $w \in Z$, то требуемый цикл строится из ребра (v,w) и из того отрезка цикла Z от v до w, который содержит вершину u. Если же $w \notin Z$, то по следствию 2 существует простая $\langle u,w\rangle$-цепь P, не содержащая вершину v. Обозначим через s первую вершину, считая от вершины w, на цепи P, которая также принадлежит циклу Z. Тогда рассмотрим следующую последовательность простых цепей: отрезок цепи P от вершины w до вершины s,

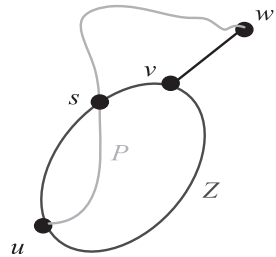


Рис. 8.3. К доказательству пункта [2⟹3] теоремы о блоках

отрезок цикла Z от вершины s до вершины v, содержащий вершину u, и ребро (v, w). Это цикл, причём простой, содержащий вершину u и ребро (v, w).

[3⟹4] Пусть (u, w) и (s, t) — два любых ребра. По условию, ребро (u, w) и вершина s принадлежат некоторому циклу Z_1, и в то же время ребро (u, w) и вершина t принадлежат некоторому циклу Z_2 (рис. 8.4). Тогда рассмотрим следующую последовательность простых цепей: ребро (u, w), отрезок цикла Z_1 от вершины u до вершины s, ребро (s, t), отрезок цикла Z_2 от вершины t до вершины w. Это цикл, причём простой, содержащий рёбра (u, w) и (s, t).

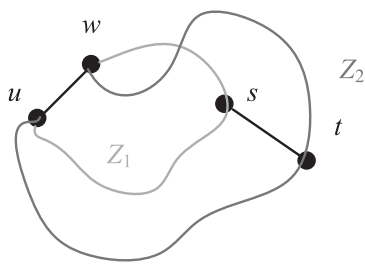


Рис. 8.4. К доказательству пункта [3⟹4] теоремы о блоках

[4⟹5] Пусть u и w — любые две вершины и (s, t) — любое ребро. Граф связен, значит, существует простая $\langle u, w\rangle$-цепь P. Пусть $P = (u, x, \ldots, w)$. По условию, рёбра (u, x) и (s, t) принадлежат некоторому простому циклу Z (рис. 8.5). Обозначим через v первую вершину, считая от вершины w, на цепи P, которая также принадлежит циклу Z. Тогда рассмотрим следующую последовательность простых цепей: отрезок цикла Z от вершины u до вершины v, проходящий через ребро (s, t), отрезок цепи P от вершины v до вершины w. Это цепь, причём простая, содержащая ребро (s, t).

[5⟹6] Пусть u, v, w — любые три вершины и (w, x) — некоторое ребро, инцидентное вершине w. По условию существует простая $\langle u, v\rangle$-цепь P, содержащая ребро (w, x) и, значит, вершину w.

[6⟹1] Пусть v — любая вершина. Покажем, что вершина v не является точкой сочленения, то есть граф $G - v$ связен. Рассмотрим u и w — любые две вершины. По условию существует простая $\langle u, v\rangle$-цепь P, проходящая через вершину u. Отрезок цепи P от u до w обеспечивает связанность вершин u и w в графе $G - v$. □

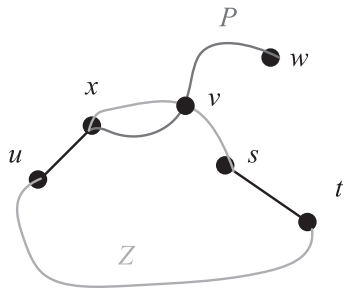


Рис. 8.5. К доказательству пункта [4⟹5] теоремы о блоках

8.1.3. Вершинная и рёберная связность

При взгляде на диаграммы связных графов (сравните, например, рис. 7.5 и 8.1) возникает естественное впечатление, что связный граф может быть «более» или «менее» связен. В этом параграфе рассматриваются некоторые количественные меры связности.

Вершинной связностью графа G (обозначается $\varkappa(G)$) называется наименьшее число вершин, удаление которых приводит к несвязному или тривиальному графу.

Примеры

$\varkappa(G) = 0$, если G несвязен;
$\varkappa(G) = 1$, если G имеет точку сочленения;
$\varkappa(K_p) = p - 1$, если K_p — полный граф.

Граф G называется *n-связным*, если $\varkappa(G) = n$.

Рёберной связностью графа G (обозначается $\lambda(G)$) называется наименьшее число рёбер, удаление которых приводит к несвязному или тривиальному графу.

Примеры

$\lambda(G) = 0$, если G несвязен;
$\lambda(G) = 1$, если G имеет мост;
$\lambda(K_p) = p - 1$, если K_p — полный граф.

ТЕОРЕМА. $\varkappa(G) \leqslant \lambda(G) \leqslant \delta(G)$.

Доказательство.

[$\varkappa \leqslant \lambda$] Если $\lambda = 0$, то $\varkappa = 0$. Если $\lambda = 1$, то есть мост, а значит, либо есть точка сочленения, либо $G = K_2$. В любом случае $\varkappa = 1$. Пусть теперь $\lambda \geqslant 2$. Тогда, удалив $\lambda - 1$ ребро, получим граф G', имеющий мост (u, v). Для каждого из удаляемых $\lambda - 1$ рёбер удалим инцидентную вершину, отличную от u и v. Если после такого удаления вершин граф несвязен, то $\varkappa \leqslant \lambda - 1 < \lambda$. Если связен, то удалим один из концов моста — u или v. Имеем $\varkappa \leqslant \lambda$.

[$\lambda \leqslant \delta$] Если $\delta = 0$, то $\lambda = 0$. Пусть вершина v имеет наименьшую степень: $d(v) = \delta$. Удалим все рёбра, инцидентные v. Тогда $\lambda \leqslant \delta$. □

Особое значение имеют *двусвязные* графы по двум причинам. Во-первых, они особенно часто встречаются в приложениях. Во-вторых, двусвязные графы являются частным случаем блоков, которые обладают многими полезными свойствами, часть из которых приведена в предыдущем параграфе.

ЗАМЕЧАНИЕ

Легко видеть, что любой двусвязный граф является блоком, поскольку не содержит точек сочленения. В то же время всякий нетривиальный блок является, по меньшей мере, двусвязным, но может быть и трёхсвязным, и вообще иметь произвольную величину вершинной связности, поскольку, например, всякий полный граф является блоком.

8.1.4. Оценка числа рёбер

Легко заметить, что инварианты $p(G)$, $q(G)$ и $k(G)$ не являются независимыми.

ТЕОРЕМА. *Пусть p — число вершин, q — число рёбер, k — число компонент связности графа. Тогда $p - k \leqslant q \leqslant (p-k)(p-k+1)/2$.*

Доказательство.

[$p - k \leqslant q$] Индукция по p. База: $p = 1$, $q = 0$, $k = 1$. Пусть оценка верна для всех графов с числом вершин меньше p. Рассмотрим граф G, $p(G) = p$. Удалим в G вершину v, которая не является точкой сочленения: $G' := G - v$. Тогда если v — изолированная вершина, то $p' = p - 1$, $k' = k - 1$, $q' = q$. Имеем $p - k = p' - k' \leqslant \leqslant q' = q$. Если v — не изолированная вершина, то $p' = p - 1$, $k' = k$, $q' < q$. Имеем $p - k = 1 + p' - k' \leqslant 1 + q' \leqslant q$.

[$q \leqslant (p-k)(p-k+1)/2$] Метод выделения «критических» графов. Число рёбер q графа с p вершинами и k компонентами связности не превосходит числа рёбер в графе с p вершинами и k компонентами связности, в котором все компоненты связности суть полные графы. Следовательно, достаточно рассматривать только графы, в которых все компоненты связности полные. Пусть есть две полные компоненты связности, G_1 и G_2, такие, что $1 < p_1 = p(G_1) \leqslant p_2 = p(G_2)$. Если перенести одну вершину из компоненты связности G_1 в компоненту связности G_2, то число рёбер изменится на $\Delta q = p_2 - (p_1 - 1) = p_2 - p_1 + 1 > 0$. Следовательно, число рёбер возросло. Таким образом, среди всех графов с p вершинами и k полными компонентами связности наибольшее число рёбер имеет граф $\widetilde{K_p} := K_{p-k+1} \cup \bigcup\limits_{i=1}^{k-1} K_1$, в котором все рёбра сосредоточены в одной компоненте связности. Прямым подсчётом получаем: $q(\widetilde{K_p}) = (p-k+1)(p-k+1-1)/2 + 0 = (p-k)(p-k+1)/2$. □

СЛЕДСТВИЕ 1. *Если $q > (p-1)(p-2)/2$, то граф связен.*

Доказательство. От противного. Пусть $q > (p-1)(p-2)/2$, но граф не связен, то есть $k \geqslant 2$. Но тогда $(p-1)(p-2)/2 < q \leqslant (p-k)(p-k+1)/2 \leqslant (p-2)(p-1)/2$ — противоречие. □

СЛЕДСТВИЕ 2. *В связном графе $p - 1 \leqslant q \leqslant p(p-1)/2$.*

Доказательство. Следует из теоремы при $k = 1$. □

8.2. Теорема Менгера

Интуитивно очевидно, что граф тем более связен (при использовании различных мер связности), чем больше существует различных цепей, соединяющих одну вершину с другой, и тем менее связен, чем меньше нужно удалить промежуточных вершин, чтобы отделить одну вершину от другой. В этом разделе устанавливается теорема Менгера, которая придаёт этим неформальным наблюдениям точный и строгий смысл.

8.2.1. Непересекающиеся цепи и разделяющие множества

Пусть $G(V, E)$ — связный граф, u и v — две его несмежные вершины. Две цепи $\langle u, v\rangle$ называются *вершинно-непересекающимися*, если у них нет общих вершин, отличных от u и v. Две цепи $\langle u, v\rangle$ называются *рёберно-непересекающимися*, если у них нет общих рёбер. Если две цепи вершинно не пересекаются, то они и рёберно не пересекаются. Обозначим через $P(u, v)$ множество вершинно-непересекающихся цепей $\langle u, v\rangle$.

Множество S вершин и (или) рёбер графа G *разделяет* две вершины, u и v, если u и v принадлежат разным компонентам связности графа $G - S$. *Разделяющее множество рёбер* называется *разрезом*. *Разделяющее множество вершин* для вершин u и v обозначим $S(u, v)$.

Из определений следует, что при $|P(u, v)| > 1$ любое множество $P(u, v)$ вершинно-непересекающихся цепей обладает тем свойством, что $\bigcap_{\langle u,v\rangle \in P(u,v)} \langle u, v\rangle = \{u, v\}$, а любое разделяющее множество вершин $S(u, v)$ обладает тем свойством, что $G - S = G_1 \cup G_2$ & $v \in G_1$ & $u \in G_2$. Для заданных вершин u и v множества $P(u, v)$ и $S(u, v)$ можно выбирать различным образом.

ЗАМЕЧАНИЕ

Если вершины u и v принадлежат разным компонентам связности графа G, то $|P(u, v)| = 0$ и $|S(u, v)| = 0$. Поэтому без ограничения общности можно считать, что G — связный граф.

Пример. В графе, диаграмма которого представлена на рис. 8.6, можно выбрать, например, множества вершинно-непересекающихся путей $P_1 = \{\langle u, a, d, v\rangle, \langle u, b, e, v\rangle\}$ и $P_2 = \{\langle u, b, d, v\rangle, \langle u, c, e, v\rangle\}$. Заметим, что путь $\langle u, c, b, d, v\rangle$ образует множество P_3 вершинно-непересекающихся путей, состоящее из одного элемента. Множества вершин $S_1 = \{a, b, c\}$ и $S_2 = \{d, e\}$ являются разделяющими для вершин u и v.

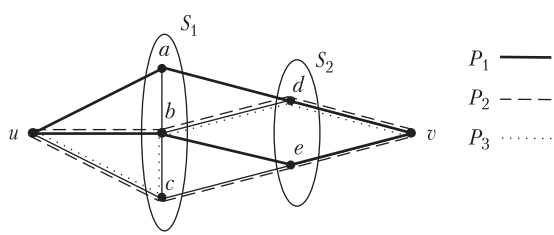


Рис. 8.6. Вершинно-непересекающиеся пути и разделяющие множества вершин

8.2.2. Теорема Менгера в «вершинной форме»

ТЕОРЕМА (Менгера). *Пусть u и v — несмежные вершины в графе G. Наименьшее число вершин в множестве, разделяющем u и v, равно наибольшему числу вершинно-непересекающихся простых $\langle u, v\rangle$-цепей:* $\max|P(u,v)| = \min|S(u,v)|$.

ЗАМЕЧАНИЕ

Пусть G — связный граф и вершины u и v несмежны. Пусть $P := P(u,v)$, $S := S(u,v)$. Легко видеть, что $|P| \leqslant |S|$. Действительно, любая $\langle u, v\rangle$-цепь проходит через S. Если бы $|P| > |S|$, то в S была бы вершина, принадлежащая более чем одной цепи из P, что противоречит выбору P. Таким образом, $\forall P\ (\forall S\ (|P| \leqslant |S|))$. Следовательно, $\max|P| \leqslant \min|S|$. Утверждение теоремы состоит в том, что в любом графе *существуют* такие P и S, что достигается равенство $|P| = |S|$.

Доказательство. Пусть G — связный граф, u и v — несмежные вершины. Совместная индукция по p и q. База: наименьший граф, удовлетворяющий условиям теоремы, состоит из трех вершин, u, w, v, и двух рёбер, (u,w) и (w,v). В нём $P(u,v) = \{\langle u,w,v\rangle\}$ и $S(u,v) = \{w\}$. Таким образом, $|P(u,v)| = |S(u,v)| = 1$. Пусть утверждение теоремы верно для всех графов с числом вершин меньше p и (или) числом рёбер меньше q. Рассмотрим граф G с p вершинами и q ребрами. Пусть $u, v \in V$, причём u, v — не смежны и S — некоторое наименьшее множество вершин, разделяющее u и v. Обозначим $n := |S|$. Рассмотрим три случая.

[1] Пусть в S есть вершины, несмежные с u и несмежные с v. Тогда граф $G - S$ состоит из двух *нетривиальных* графов, G_1 и G_2. Образуем два новых графа, G_u и G_v, стягивая нетривиальные графы G_1 и G_2 к вершинам u и v соответственно: $G_u := G/G_1$, $G_v := G/G_2$ (рис. 8.7). S по-прежнему является наименьшим разделяющим множеством для u и v как в G_u, так и в G_v. Так как G_1 и G_2 нетривиальны, то G_u и G_v имеют меньше вершин и (или) рёбер, чем G. Следовательно, по индукционному предположению в G_u и в G_v имеется n вершинно-непересекающихся простых цепей. Комбинируя отрезки этих цепей от u до S и от S до v, получаем n вершинно-непересекающихся простых цепей в G.

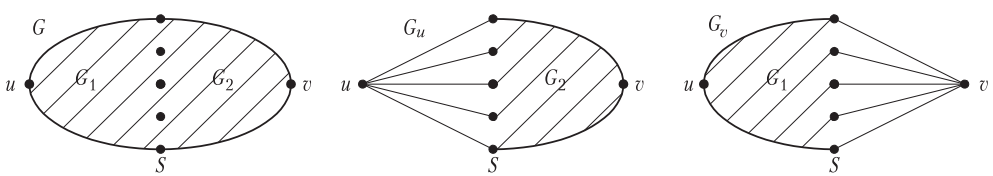


Рис. 8.7. К доказательству теоремы Менгера, случай 1

[2] Пусть все вершины S смежны с u или с v (для определённости пусть с u) и среди вершин S есть вершина w, смежная одновременно с u и с v (рис. 8.8). Рассмотрим граф $G' := G - w$. В нем $S - w$ — разделяющее множество для u и v, причём наименьшее. По индукционному предположению в G' имеется $|S - w| = n - 1$ вершинно-непересекающихся простых цепей. Добавим к ним цепь $\langle u, w, v\rangle$. Она простая и вершинно не пересекается с остальными. Таким образом, имеем n вершинно-непересекающихся простых цепей в G.

[3] Пусть теперь все вершины S смежны с u или с v (для определённости пусть с u), и среди вершин S нет вершин, смежных одновременно с вершиной u и с вершиной

v. Рассмотрим *кратчайшую* $\langle u, v\rangle$-цепь $\langle u, w_1, w_2, \ldots, v\rangle$, $w_1 \in S$, $w_2 \neq v$ (рис. 8.9). Рассмотрим граф $G' := G/\{w_1, w_2\}$, полученный из G склейкой вершин w_1 и w_2. Имеем $w_2 \notin S$, в противном случае цепь $\langle u, w_2, \ldots, v\rangle$ была бы ещё более короткой. Следовательно, в графе G' множество S по-прежнему является наименьшим, разделяющим u и v, и граф G' имеет по крайней мере на одно ребро меньше. По индукционному предположению в G' существуют n вершинно-непересекающихся простых цепей. Но цепи, которые не пересекаются в G', не пересекаются и в G. Таким образом, получаем n вершинно-непересекающихся простых цепей в G. □

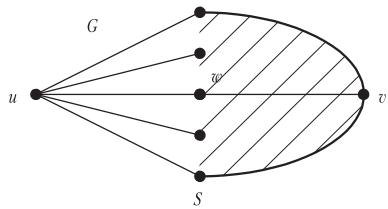


Рис. 8.8. К доказательству теоремы Менгера, случай 2

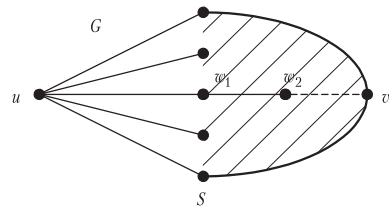


Рис. 8.9. К доказательству теоремы Менгера, случай 3

8.2.3. Варианты теоремы Менгера

Теорема Менгера представляет собой весьма общий факт, который в разных формулировках встречается в различных областях математики. Комбинируя вершинно- и рёберно-непересекающиеся цепи, разделяя не отдельные вершины, а множества вершин, используя инварианты $\varkappa$ и λ, можно сформулировать и другие утверждения, подобные теореме Менгера. Например:

ТЕОРЕМА 1. *Для любых двух несмежных вершин u и v графа G наибольшее число рёберно-непересекающихся $\langle u, v\rangle$-цепей равно наименьшему числу рёбер в (u, v)-разрезе.*

ТЕОРЕМА 2. *Чтобы граф G был n-связным, необходимо и достаточно, чтобы любые две несмежные вершины были соединены не менее чем n вершинно-непересекающимися простыми цепями.*

Другими словами, в любом графе G любые две несмежные вершины соединены не менее чем $\varkappa(G)$ вершинно-непересекающимися простыми цепями.

Доказательство подобных утверждений требует места, и мы оставляем их за пределами книги. В последующих разделах приведены и некоторые другие результаты, в которых проявляется теорема Менгера.

8.3. Паросочетания

Паросочетанием (или *независимым множеством рёбер*) называется множество рёбер, в котором никакие два ребра не смежны. Паросочетание называется *максимальным*, если никакое его надмножество не является независимым.

Задача отыскания паросочетаний имеет множество различных применений и интерпретаций, часть которых приведена в этом разделе.

Рассмотрение начинается с интерпретаций теоремы Холла[1] «о свадьбах» и продолжается другими примерами.

[1] Маршалл Холл (1910–1990).

8.3.1. Задачи о свадьбах, трансверсалях и паросочетаниях

Рассмотрим три внешне различные задачи.

1. Пусть имеется конечное множество юношей, каждый из которых знаком с некоторым подмножеством конечного множества девушек. В каком случае всех юношей можно женить так, чтобы каждый женился на знакомой девушке?
2. Пусть $S = \{S_1, \ldots, S_m\}$ — семейство подмножеств конечного множества E. Подмножества S_k не обязательно различны и могут пересекаться. *Системой различных представителей* в семействе S (или *трансверсалью* в семействе S) называется любое подмножество $C = \{c_1, \ldots, c_m\}$ из m элементов множества E таких, что $\forall\, k \in 1..m\ \ (c_k \in S_k)$. В каком случае существует трансверсаль?

ЗАМЕЧАНИЕ

Все элементы множества C различны, откуда и происходит название «система различных представителей».

3. Пусть $G(V_1, V_2, E)$ — двудольный граф. *Совершенным паросочетанием* из V_1 в V_2 называется паросочетание, покрывающее вершины V_1. В каком случае существует совершенное паросочетание из V_1 в V_2?

ЗАМЕЧАНИЕ

Совершенное паросочетание является максимальным.

Вообще говоря, указанные задачи — это одна и та же задача. Действительно, задача о свадьбах сводится к задаче о паросочетаниях следующим образом. V_1 — множество юношей, V_2 — множество девушек, рёбра — знакомства юношей с девушками. В таком случае совершенное паросочетание — искомый набор свадеб. Задача о трансверсали сводится к задаче о паросочетаниях следующим образом. Положим $V_1 := S$, $V_2 := E$, ребро (S_k, e_i) существует, если $e_i \in S_k$. В таком случае совершенное паросочетание — искомая трансверсаль. Все задачи имеют общий ответ: в том и только том случае, когда

$$\text{любые } k \left(\begin{array}{c}\text{юношей}\\ \text{подмножеств}\\ \text{вершин из } V_1\end{array}\right) \text{ в совокупности } \left(\begin{array}{c}\text{знакомы с}\\ \text{содержат}\\ \text{смежны с}\end{array}\right)$$

$$\text{не менее чем } k \left(\begin{array}{c}\text{девушками}\\ \text{элементов}\\ \text{вершинами из } V_2\end{array}\right),$$

что устанавливается теоремой Холла.

8.3.2. Теорема Холла

ТЕОРЕМА (Холла). *Пусть $G(V_1, V_2, E)$ — двудольный граф. Совершенное паросочетание из V_1 в V_2 существует тогда и только тогда, когда $\forall\, A \subset V_1\ \ (|A| \leqslant |\Gamma(A)|)$.*

Доказательство.

[$\Longrightarrow$] Пусть существует совершенное паросочетание из V_1 в V_2. Тогда в $\Gamma(A)$ входит $|A|$ вершин из V_2, парных к вершинам из множества A, и, возможно, еще что-то. Таким образом, $|A| \leqslant |\Gamma(A)|$.

[$\Longleftarrow$] Добавим в G две новые вершины, u и v, так что вершина u смежна со всеми вершинами из V_1, а вершина v смежна со всеми вершинами из V_2. Совершенное паросочетание из V_1 в V_2 существует тогда и только тогда, когда существуют $|V_1|$ вершинно-непересекающихся простых $\langle u, v\rangle$-цепей (рис. 8.10). Ясно, что $|P(u,v)| \leqslant$ $\leqslant |V_1|$ (так как V_1 разделяет вершины u и v).

По теореме Менгера $\max|P(u,v)| = \min|S(u,v)| = |S|$, где S — наименьшее множество, разделяющее вершины u и v. Имеем $|S| \leqslant |V_1|$. Покажем, что $|S| \geqslant |V_1|$. Пусть $S = A \cup B$, $A \subset V_1$, $B \subset V_2$. Тогда $\Gamma(V_1 \setminus A) \subset B$, не считая вершин u, v. Действительно, если бы $\Gamma(V_1 \setminus A) \not\subset B$, то существовал бы «обходной» путь $\langle u, v_1, v_2, v\rangle$ (рис. 8.10), и S не было бы разделяющим множеством для u и v. Имеем $|V_1 \setminus A| \leqslant |\Gamma(V_1 \setminus A)| \leqslant |B|$. Следовательно, $|S| = |A| + |B| \geqslant |A| + |V_1 \setminus A| = |V_1|$. □

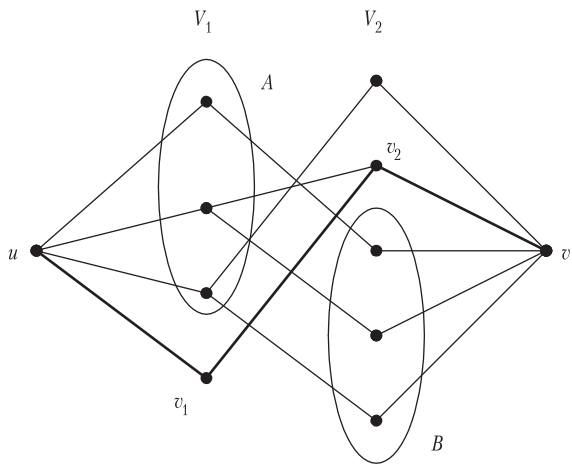


Рис. 8.10. К доказательству теоремы Холла

8.3.3. Устойчивые паросочетания

Пусть имеется конечное множество мужчин и конечное множество женщин (равномощные). Элементы множества мужчин обозначим прописными буквами, а элементы множества женщин — строчными. Пусть также задано совершенное взаимнооднозначное паросочетание из множества мужчин в множество женщин и обратно (брачное отношение): **Aa**, **Bb** и т. д. Пусть теперь у каждого мужчины есть список женщин, упорядоченный по убыванию «привлекательности» для этого мужчины. А у каждой женщины — список мужчин, также упорядоченный по убыванию «привлекательности» для неё.

Паросочетание называется *неустойчивым*, если в нём найдутся мужчина **X** (женатый на женщине **x**) и женщина **y** (замужем за мужчиной **Y**), предпочитающие друг друга своим супругам, то есть:

1) **X** считает **y** привлекательнее, чем **x**;
2) **y** считает **X** привлекательнее, чем **Y**.

Паросочетание называется *устойчивым*, если таких пар в нём нет.

Пример. Четверо мужчиин (**A**, **B**, **C**, **D**) состоят в браке с четырьмя женщинами (**a**, **b**, **c**, **d**). Порядок предпочтений представлен в таблицах.

Мужчины	Предпочтения	Женщины	Предпочтения
A	**c b d a**	**a**	**A B D C**
B	**b a c d**	**b**	**C A B D**
C	**b d a c**	**c**	**C B D A**
D	**c a d b**	**a**	**B A C D**

Паросочетание (**Aa**, **Bb**, **Cc**, **Dd**) неустойчиво, так как **A** и **b** предпочитают друг друга своим супругам.

Задача заключается в нахождении устойчивого паросочетания по данным спискам предпочтений. Иными словами, нужно в полном двудольном графе $K_{n,n}$ найти совершенное устойчивое паросочетание, если оно существует.

Решение задачи существует, как показывает алгоритм 8.1. Для представления данных задачи в алгоритме используются следующие структуры данных.

1. Матрица предпочтений мужчин M : **array** $[1..n, 1..n]$ **of** $1..n$. Элемент $M[m, p]$ означает номер женщины, стоящей на месте p в списке предпочтений мужчины m.
2. Матрица предпочтений женщин W : **array** $[1..n, 1..n]$ **of** $1..n$. Элемент $W[w, m]$ означает рейтинг мужчины m в списке предпочтений женщины w.

   Пример. Данные предыдущего примера представляются следующими матрицами:

$$M = \begin{pmatrix} 3 & 2 & 4 & 1 \\ 2 & 1 & 3 & 4 \\ 2 & 4 & 1 & 3 \\ 3 & 1 & 4 & 2 \end{pmatrix}, \quad W = \begin{pmatrix} 4 & 3 & 1 & 2 \\ 3 & 1 & 4 & 2 \\ 1 & 3 & 4 & 2 \\ 3 & 4 & 2 & 1 \end{pmatrix}.$$

3. Список женихов B : **array** $[1..n]$ **of** $0..n$, где $B[m]$ — номер невесты, помолвленной с женихом m.
4. Список предложений P : **array** $[1..n]$ **of** $0..n$, где $P[m]$ — количество предложений, уже сделанных женихом m.
5. Список невест G : **array** $[1..n]$ **of** $0..n$, где $G[w]$ — номер жениха, помолвленного с невестой w.
6. Текущий рейтинг R : **array** $[1..n]$ **of** $0..n$, где $R[w]$ — рейтинг жениха, помолвленного с невестой w.

ЗАМЕЧАНИЕ

Считаем, что в начальном состоянии списки инициализированы нулями, то есть никакие предложения не были сделаны и никакие помолвки не состоялись.

Обоснование.

[Завершаемость] На каждой итерации рассматривается одно предложение, а всего возможно не более n^2 предложений.

[Паросочетаемость] На каждой итерации для ненулевых элементов инвариантно выполняется условие: $G[w] = m \iff B[m] = w$, пары образуют паросочетание.

Алгоритм 8.1. Гейла–Шепли отложенного согласия

Вход: n число мужчин и женщин, матрицы предпочтений M, W.
Выход: n сформированных пар в списках B, G.

```
k := 0 // количество сформированных пар
while (k < n) do
  for m from 1 to n do
    if B[m] = 0 then
      // первый свободный жених m делает предложение невесте w
      P[m] := P[m] + 1; w := M[m, P[m]]
      exit for i  // цикл не нужно продолжать
    end if
  end for
  if G[w] = 0 then
    // первое предложение всегда принимается — помолвка
    k := k + 1; G[w] := m, B[m] := w; R[w] := W[w, m]
    next while k < n  // следующий жених
  end if
  if W[w, m] > R[w] then
    // новый жених лучше — прежнему отставка
    B[G[w]] := 0; B[m] := w; R[w] := W[w, m]; G[w] := m
  end if
end while
```

[Совершенность] Всего создано n пар, значит, паросочетание совершенное.

[Устойчивость] От противного. Пусть в найденном паросочетании **Xx, Yy** существует неустойчивая комбинация **X–y**. Возможны два случая. Мужчина **X** не делал предложения женщине **y**. Но это означает, что **x** предшествует **y** в списке предпочтений **X**, что противоречит неустойчивости. Мужчина **X** делал предложение женщине **y**, и она ему отказала. Но это означает, что **Y** предшествует **X** в списке предпочтений **y**, что противоречит неустойчивости. □

Пример. Для рассматриваемого примера алгоритм последовательно построит следующие паросочетания: $\varnothing \mapsto \{\mathbf{Ac}\} \mapsto \{\mathbf{Ac}, \mathbf{Bb}\} \mapsto \{\mathbf{Ac}, \mathbf{Cb}\} \mapsto \{\mathbf{Ac}, \mathbf{Cb}, \mathbf{Ba}\} \mapsto \{\mathbf{Cb}, \mathbf{Ba}, \mathbf{Dc}\} \mapsto \{\mathbf{Cb}, \mathbf{Ba}, \mathbf{Dc}, \mathbf{Ad}\}$.

ЗАМЕЧАНИЕ

Алгоритм находит одно из многих возможных решений задачи. В данном случае алгоритм строит наилучший вариант для мужчин (для каждого мужчины выбранная алгоритмом супруга наиболее привлекательна для него из всех возможных устойчивых паросочетаний) и наихудший для женщин (выбранный супруг нравится каждой женщине не больше, чем возможный супруг в любом другом устойчивом паросочетании).

ОТСТУПЛЕНИЕ

Задача об устойчивом паросочетании имеет множество приложений и интерпретаций. Например, задача о приеме n абитуриентов в m университетов. Пусть n абитуриентов поступают в m университетов, и университет с номером k может принять n_k абитуриентов. Не ограничивая общности, можно считать, что $n_1 + n_2 + \ldots + n_m = n$, рассматривая, если нужно, самый

нежелательный «никакой» университет с неограниченной ёмкостью приёма. Каждый абитуриент имеет свой рейтинг университетов, каждый университет — свой рейтинг абитуриентов. Можно говорить об «устойчивом зачислении», если никакой абитуриент **A**, не зачисленный в университет **a**, но предпочитающий его своему вузу, не окажется предпочтительнее для университета **a**, чем кто-то из его студентов.

8.4. Потоки в сетях

Рассмотрим некоторые примеры практических задач.

Примеры

1. Пусть имеется сеть автомобильных дорог, по которым можно проехать из пункта A в пункт B. Дороги могут пересекаться в промежуточных пунктах. Количество автомобилей, которые могут проехать по каждому отрезку дороги за единицу времени, не безгранично: оно определяется такими факторами, как ширина проезжей части, качество дорожного покрытия, действующие ограничения скорости движения и т. д. (обычно это называют «пропускной способностью» дороги). Каково максимальное количество автомобилей, которые могут проехать из пункта A в пункт B за единицу времени без образования пробок на дорогах (обычно это называют «автомобильным потоком»)? Можно поставить и другой вопрос: какие дороги и насколько нужно расширить или улучшить, чтобы увеличить максимальный автомобильный поток на заданную величину?
2. Пусть имеется сеть трубопроводов, соединяющих пункт A (скажем, нефтепромысел) с пунктом B (скажем, нефтеперерабатывающим заводом). Трубопроводы могут соединяться и разветвляться в промежуточных пунктах. Количество нефти, которое может быть перекачано по каждому отрезку трубопровода в единицу времени, также не безгранично и определяется такими факторами, как диаметр трубы, мощность нагнетающего насоса и т. д. (обычно это называют «пропускной способностью» или «максимальным расходом» трубопровода). Сколько нефти можно прокачать через такую сеть за единицу времени?
3. Пусть имеется система машин для производства готовых изделий из сырья и последовательность технологических операций может быть различной (то есть операции могут выполняться на разном оборудовании или в разной последовательности), но все допустимые варианты заранее строго определены. Максимальная производительность каждой единицы оборудования, естественно, также заранее известна и постоянна. Какова максимально возможная производительность всей системы в целом и как нужно организовать производство для достижения максимальной производительности?

Изучение этих и многочисленных подобных им практических задач приводит к теории потоков в сетях. В данном разделе рассматривается решение только одной (но самой существенной) задачи этой теории, а именно: задачи определения максимального потока.

8.4.1. Определение потока

Пусть $G(V, E)$ — сеть, s и t — источник и сток сети соответственно. Дуги сети нагружены неотрицательными вещественными числами, $c\colon E \to \mathbb{R}^+$. Если u и v — узлы сети, то число $c(u, v)$ называется *пропускной способностью* дуги (u, v).

ЗАМЕЧАНИЕ

Матрица пропускных способностей C : **array** $[1..p, 1..p]$ **of real** является представлением сети, аналогичным матрице смежности. Нулевое значение элемента $C[u, v]$ соответствует дуге с нулевой пропускной способностью, то есть отсутствию дуги, а положительное значение элемента $C[u, v]$ соответствует дуге с ненулевой пропускной способностью, то есть дуга присутствует.

Пусть задана функция $f\colon E \to \mathbb{R}$. *Дивергенцией* функции f в узле v называется число $\operatorname{div}(f, v)$, которое определяется следующим образом:

$$\operatorname{div}(f, u) \stackrel{\text{Def}}{=} \sum_{\{v|(u,v)\in E\}} f(u, v) - \sum_{\{v|(v,u)\in E\}} f(v, u).$$

ЗАМЕЧАНИЕ

В физике дивергенция обычно определяется наоборот: то, что пришло, минус то, что ушло. Но в данном случае удобнее, чтобы дивергенция источника была положительна.

Функция $f\colon E \to \mathbb{R}$ называется *потоком* в сети G, если:

1) $\forall\,(u, v) \in E \;\; (0 \leqslant f(u, v) \leqslant c(u, v))$, то есть поток через дугу неотрицателен и не превосходит пропускной способности дуги;
2) $\forall\, u \in V \setminus \{s, t\} \;\; (\operatorname{div}(f, u) = 0)$, то есть дивергенция потока равна нулю во всех узлах, кроме источника и стока.

Число $w(f) \stackrel{\text{Def}}{=} \operatorname{div}(f, s)$ называется *величиной* потока f. Если $f(u, v) = c(u, v)$, то дуга (u, v) называется *насыщенной*.

8.4.2. Разрезы

Пусть P — (s, t)-разрез, $P \subset E$. Всякий разрез определяется разбиением множества вершин V на два подмножества, S и T, так, что $S \subset V$, $T \subset V$, $S \cup T = V$, $S \cap T = \varnothing$, $s \in S$, $t \in T$, а в P попадают все дуги, соединяющие S и T. Тогда $P = P^+ \cup P^-$, где P^+ — дуги от S к T, P^- — дуги от T к S. Сумма потоков через дуги разреза P обозначается $F(P)$. Сумма пропускных способностей дуг разреза P называется *пропускной способностью* разреза и обозначается $C(P)$:

$$F(P) \stackrel{\text{Def}}{=} \sum_{e\in P} f(e), \qquad C(P) \stackrel{\text{Def}}{=} \sum_{e\in P} c(e).$$

Аналогично, $F(P^+)$ и $F(P^-)$ — суммы потоков через соответствующие части разрезов (**«положительную» и «отрицательную»**).

8.4.3. Теорема Форда–Фалкерсона

В этом параграфе устанавливается количественная связь между величиной максимального потока и пропускными способностями дуг сети.

ЛЕММА 1. $w(f) = F(P^+) - F(P^-)$.

Доказательство. Рассмотрим сумму $W := \sum\limits_{v \in S} \operatorname{div}(f, v)$. Пусть дуга $(u, v) \in E$. Если $u, v \in S$, то в сумму W попадают два слагаемых для этой дуги: $+f(u, v)$ при вычислении $\operatorname{div}(f, u)$ и $-f(u, v)$ при вычислении $\operatorname{div}(f, v)$, итого 0. Если $u \in S$, $v \in T$, то в сумму W попадает одно слагаемое $+f(u, v)$, все такие слагаемые дают $F(P^+)$. Если $u \in T$, $v \in S$, то в сумму W попадает одно слагаемое $-f(u, v)$, все такие слагаемые дают $F(P^-)$. Таким образом, $W = F(P^+) - F(P^-)$. С другой стороны, $W = \sum\limits_{v \in S} \operatorname{div}(f, v) = \operatorname{div}(f, s) = w(f)$. □

Лемма 2. $\operatorname{div}(f, s) = -\operatorname{div}(f, t)$.

Доказательство. Рассмотрим разрез $P := (S, T)$, где $S := V - t$, а $T := \{t\}$. Имеем

$$\operatorname{div}(f, s) = w(f) = F(P^+) - F(P^-) = F(P^+) = \sum_v f(v, t) = -\operatorname{div}(f, t).$$ □

Лемма 3. $w(f) \leqslant F(P)$.

Доказательство. $w(f) = F(P^+) - F(P^-) \leqslant F(P^+) \leqslant F(P)$. □

Лемма 4. $\max\limits_f w(f) \leqslant \min\limits_P C(P)$.

Доказательство. По предыдущей лемме $w(f) \leqslant F(P)$, следовательно, $\max\limits_f w(f) \leqslant \min\limits_P F(P)$. По определению $F(P) \leqslant C(P)$, следовательно, $\min\limits_P F(P) \leqslant \min\limits_P C(P)$. Имеем $\max\limits_f w(f) \leqslant \min\limits_P C(P)$. □

Теорема (Форда[1]–Фалкерсона[2]). *Максимальный поток в сети равен минимальной пропускной способности положительной части разреза, то есть существует поток f^* такой, что $w(f^*) = \max\limits_f w(f) = \min\limits_P C(P^+)$.*

Доказательство. Пусть f — некоторый максимальный поток. Покажем, что существует разрез P такой, что $w(f) = C(P)$. Рассмотрим граф G', полученный из сети G забыванием ориентации дуг. Построим множество вершин S следующим образом:

$$S \stackrel{\text{Def}}{=} \{u \in V \mid \exists\, \langle s, u\rangle \in G'\ \forall (u_i, u_{i+1}) \in \langle s, u\rangle \in G'$$
$$((u_i, u_{i+1}) \in E \Longrightarrow f(u_i, u_{i+1}) < C(u_i, u_{i+1})\ \&$$
$$\&\ (u_{i+1}, u_i) \in E \Longrightarrow f(u_{i+1}, u_i) > 0)\},$$

то есть вдоль пути $\langle s, u\rangle$ дуги в направлении пути не насыщены, а дуги против направления пути имеют положительный поток. Такая цепь $\langle s, u\rangle$ называется *аугментальной*. Имеем $s \in S$ по построению. Следовательно, $S \neq \varnothing$. Положим $T := V \setminus S$. Покажем, что $t \in T$. Действительно, пусть $t \in S$. Тогда существует аугментальная цепь $\langle s, t\rangle$, обозначим её R. Но тогда можно найти число $\delta > 0$:

$$\delta := \min_{e \in R} \Delta(e), \text{ где } \Delta(e) := \begin{cases} c(e) - f(e) > 0, & \text{если } e \text{ ориентировано вдоль } R, \\ f(e) > 0, & \text{если } e \text{ ориентировано против } R. \end{cases}$$

[1] Лестер Рендольф Форд мл. (род. 1927).

[2] Дальберт Рей Фалкерсон (1924–1976).

В этом случае можно увеличить величину потока на δ, изменив поток для всех дуг аугментальной цепи:

$$f(e) := \begin{cases} f(e) + \delta, & \text{если } e \text{ ориентировано вдоль } R, \\ f(e) - \delta, & \text{если } e \text{ ориентировано против } R. \end{cases}$$

При этом условия потока выполняются: $0 \leqslant f(e) \leqslant C(e)$, $\operatorname{div}(v) = 0$.

Таким образом, поток f увеличен на величину δ, что противоречит максимальности потока f. Имеем $t \in T$ и $T \neq \varnothing$. Следовательно, S и T определяют разрез $P = P^+ \cup P^-$. В этом разрезе все дуги e^+ насыщены ($f(e^+) = c(e^+)$), а все дуги e^- не нагружены ($f(e^-) = 0$), иначе S можно было бы расширить. Имеем $w(f) = F(P^+) - F(P^-) = C(P^+)$, таким образом, P — искомый разрез. □

8.4.4. Алгоритм нахождения максимального потока

Следующий алгоритм определяет максимальный поток в сети, заданной матрицей пропускных способностей дуг. Этот алгоритм использует идею доказательства теоремы Форда–Фалкерсона, а именно, задавшись начальным приближением потока, определяем множество вершин S, которые соединены аугментальными цепями с источником s. Если оказывается, что $t \in S$, то это означает, что поток не максимальный и его можно увеличить на величину δ. Для определения аугментальных цепей и одновременного подсчета величины δ в алгоритме использована вспомогательная структура данных:

P : **array** $[1..p]$ **of struct {**
  s : **enum** $(-,+)$ // «знак», то есть направление дуги
  n : $1..p$ // предшествующая вершина в аугментальной цепи
  δ : **real** // величина возможного увеличения потока
}
N : **array** $[1..p]$ **of** $0..1$ // отметка узла
S : **array** $[1..p]$ **of** $0..1$ // признак принадлежности вершины множеству S

Алгоритм 8.2. Нахождение максимального потока

Вход: сеть $G(V, E)$ с источником s и стоком t, заданная матрица пропускных способностей C : **array** $[1..p, 1..p]$ **of real**.
Выход: матрица максимального потока F : **array** $[1..p, 1..p]$ **of real**.

```
for u, v ∈ V do
  F[u, v] := 0  // вначале поток нулевой
end for
M :  // итерация увеличения потока
for v ∈ V do
  S[v] := 0; N[v] := 0; P[v] :=(+, nil, 0)  // инициализация
end for
S[s] := 1; P[s] :=(+, nil, ∞)  // так как s ∈ S
repeat
  a := 0  // признак расширения S
  for v ∈ V do
    if S[v] = 1 & N[v] = 0 then
      for u ∈ Γ(v) do
        if S[u] = 0 & F[v, u] < C[v, u] then
```

```
            S[u] := 1; P[u] := (+, v, min(P[v].δ, C[v, u] − F[v, u])); a := 1
          end if
        end for
        for u ∈ Γ^{-1}(v) do
          if S[u] = 0 & F[u, v] > 0 then
            S[u] := 1; P[u] := (−, v, min(P[v].δ, F[u, v]))); a := 1
          end if
        end for
        N[v] := 1  // узел v отмечен
      end if
    end for
    if S[t] then
      x := t  // текущий узел аугментальной цепи
      δ := P[t].δ  // величина изменения потока
      while x ≠ s do
        if P[x].s = + then
          F[P[x].n, x] := F[P[x].n, x] + δ  // увеличение потока
        else
          F[x, P[x].n] := F[x, P[x].n] − δ  // увеличение потока
        end if
        x := P[x].n  // предыдущий узел аугментальной цепи
      end while
      goto M
    end if
  until a = 0
```

Обоснование. В качестве первого приближения берётся нулевой поток, который по определению является допустимым. В основном цикле, помеченном меткой M, делается попытка увеличить поток. Для этого в цикле **repeat** расширяется, пока это возможно, множество S узлов, достижимых из источника s по аугментальным цепям. При этом если в множество S попадает сток t, то поток вдоль найденной аугментальной цепи $\langle s, t\rangle$ немедленно увеличивается на величину δ и начинается новая итерация с целью увеличить поток. Процесс расширения множества S в цикле **repeat** заканчивается, потому что множество узлов конечно, а отмеченные в массиве N узлы повторно не рассматриваются. Если процесс расширения множества S заканчивается, и при этом сток t не попадает в множество S, то по теореме Форда–Фалкерсона найденный поток F является максимальным и работа алгоритма завершается. □

ЗАМЕЧАНИЕ

Приведённый алгоритм не является самым эффективным. Более подробное изложение известных методов можно найти, например, в [14] или в [13].

8.4.5. Связь между теоремой Менгера и теоремой Форда–Фалкерсона

Теорема Менгера является фундаментальным результатом, который проявляется в различных формах. Теорема Форда–Фалкерсона также может быть получена из теоремы Менгера. Далее приведена схема доказательства теоремы Форда–Фалкерсона

на основе теоремы Менгера. Сначала нужно получить вариант теоремы Менгера в ориентированной рёберной форме: наибольшее число $\langle s,t\rangle$-путей, не пересекающихся по дугам, равно наименьшему числу дуг в (s,t)-разрезе. Это теорема Форда–Фалкерсона в том случае, когда $\forall e \in E \ \ (c(e)=1)$. Действительно, пусть Q — множество дуг из максимального набора непересекающихся $\langle s,t\rangle$-путей. Назначим поток следующим образом: $f(e):=1$, если $e \in Q$, и $f(e):=0$, если $e \notin Q$. Это поток, так как дивергенция в любом узле равна 0, и поток через дугу не превосходит пропускной способности. Величина этого потока равна $k \leqslant d^+(s)$, где k — число дуг, выходящих из s, которые начинают пути из Q. Этот поток максимальный. Действительно, если положить $f(e):=a>0$ для некоторой дуги $e \notin Q$, то:

1) если дуга e входит в узел, входящий в пути из Q, или выходит из такого узла, то нарушается условие потока (дивергенция $-a$ или $+a$ соответственно);
2) если вновь назначенные дуги с ненулевыми потоками образуют новый $\langle s,t\rangle$-путь, то это противоречит выбору Q.

Пусть теперь пропускные способности суть натуральные числа. Тогда можно провести аналогичные рассуждения для мультиграфов, считая, что вместо одной дуги с пропускной способностью n имеется n дуг с пропускной способностью 1. Если пропускные способности — рациональные числа, то их можно привести к общему знаменателю и свести задачу к предыдущему случаю.

Для вещественных пропускных способностей заключение теоремы можно получить путем перехода к пределу в условиях предыдущего случая.

ОТСТУПЛЕНИЕ

Приведенное в п. 8.4.3 доказательство теоремы Форда–Фалкерсона *конструктивно*, из него не только следует заключение о величине максимального потока, но и извлекается способ нахождения максимального потока. Набросок доказательства в этом параграфе *неконструктивен* — из него нельзя извлечь алгоритм.

8.5. Связность в орграфах

Связность является одним из немногих понятий, которые не распространяются непосредственно с графов на другие родственные объекты и требуют отдельного определения и рассмотрения в каждом случае. В данном разделе рассматривается связность в орграфах, поскольку именно это понятие чаще всего применяется в программировании.

8.5.1. Сильная, односторонняя и слабая связность

В неориентированном графе две вершины либо связаны (если существует соединяющая их цепь), либо не связаны. В ориентированном графе отношение связанности узлов несимметрично, а потому определение связности различается.

Пусть $G(V,E)$ — орграф, v_1 и v_2 — его узлы. Говорят, что два узла, v_1 и v_2, *сильно связаны* в орграфе G, если существует путь (ориентированная цепь) как из v_1 в v_2, так и из v_2 в v_1. Говорят, что два узла, v_1 и v_2, *односторонне связаны* в орграфе G, если существует путь либо из v_1 в v_2, либо из v_2 в v_1. Говорят, что два узла, v_1 и v_2, *слабо связаны* в орграфе G, если они связаны в графе G', полученном из G забыванием ориентации дуг. Если все вершины в орграфе сильно (односторонне, слабо) связаны, то орграф называется *сильно* (*односторонне*, *слабо*) *связным*. Сильная связность влечет одностороннюю связность, которая влечет слабую связность. Обратное, разумеется, неверно.

Пример. На рис. 8.11 показаны диаграммы сильно, односторонне и слабо связных орграфов.

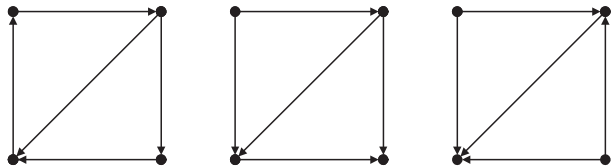


Рис. 8.11. Сильная (*слева*), односторонняя (*в центре*) и слабая (*справа*) связность

8.5.2. Компоненты сильной связности

Компонента сильной связности (употребляется аббревиатура КСС) орграфа G — это его максимальный сильно связный подграф.

Каждый узел орграфа принадлежит только одной КСС. Если узел не связан сильно с другими, то считаем, что он сам образует КСС. *Конденсацией* G^* орграфа G (или *графом Герца*, или *фактор-графом*) называется орграф, который получается стягиванием в один узел каждой КСС орграфа G.

Пример. На рис. 8.12 показаны диаграммы орграфа и его конденсации.

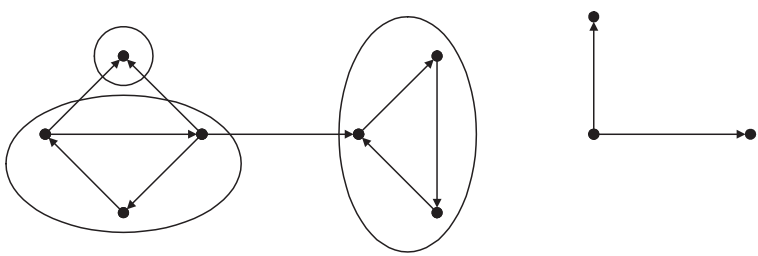


Рис. 8.12. Орграф (*слева*) и его фактор-граф (*справа*)

ЗАМЕЧАНИЕ

Фактор-граф не содержит контуров.

8.5.3. Выделение компонент сильной связности

Следующий алгоритм, основанный на методе поиска в глубину, находит все компоненты сильной связности орграфа.

Основная работа выполняется рекурсивной процедурой без параметров КСС, которая использует стек T для хранения просматриваемых узлов. Процедура КСС выделяет все компоненты сильной связности, достижимые из узла, выбранного в основном алгоритме.

Алгоритм 8.3. Выделение компонент сильной связности

Вход: орграф $G(V, E)$, заданный списками смежности $\Gamma(v)$.
Выход: список C компонент сильной связности, каждый элемент которого есть список узлов, входящих в компоненту сильной связности.

```
C := ∅
for v ∈ V do
  M[v] := {v}  // M[v] список узлов, входящих в ту же КСС, что и v
  e[v] := 0  // все узлы не рассмотрены
end for
while V ≠ ∅ do
  select v ∈ V  // взять v из V
  v → T  // положить v в стек
  e[v] := 1  // отметить v
  КСС  // вызов процедуры КСС
end while
```

Алгоритм 8.4. Рекурсивная процедура КСС

```
if T = ∅ then
  return  // негде выделять
end if
v := top T  // v — верхний элемент стека
if Γ[v] ∩ V = ∅ then
  C := C ∪ M[v]  // это КСС
  V := V − v  // удалить узел
  v ← T  // снять узел v со стека
  КСС  // возврат из тупика
else
  for u ∈ Γ[v] do
    if e[u] = 0 then
      u → T  // положить узел u в стек
      e[u] := 1  // отметить узел u
    else
      repeat
        w ← T  // w — склеиваемый узел
        V := V − w  // удалить узел
        Γ[u] := Γ[u] ∪ Γ[w]  // запомнить смежность
        M[u] := M[u] ∪ M[w]  // склеивание узлов
      until u = w
      w → T  // чтобы не убрать тот узел,
      V := V + w  // с которого начали
    end if
    КСС  // поиск в глубину
  end for
end if
```

Обоснование. Достаточно заметить, что любой контур принадлежит ровно одной КСС, и, более того, если в КСС входит несколько узлов, то они обязательно принадлежат некоторым контурам. Поэтому, если при обходе в глубину мы попадаем

в уже отмеченный узел u, то это означает, что обнаружен контур, причём предшествующие узлы найденного контура находятся на стеке, начиная от вершины стека и до рассматриваемого отмеченного узла u, который также присутствует в стеке. Все узлы найденного контура можно «склеить». При склеивании узла w его список смежности и список уже найденных узлов той КСС, которой принадлежит узел w, объединяются с соответствующими списками узла u. После этого можно продолжить поиск в глубину от узла u, который для этой цели следует оставить в стеке. □

8.6. Кратчайшие пути

Задача нахождения кратчайшего пути в графе имеет столько практических применений и интерпретаций, что читатель, без сомнения, может сам легко привести множество примеров. Здесь рассматривается несколько классических алгоритмов, которые обязан знать каждый программист.

8.6.1. Взвешенные графы

Орграф $G(V, E)$ называется *взвешенным*, если задана числовая функция $w\colon V \times V \to \mathbb{R}$, называемая *весом* или *длиной* дуги.

ЗАМЕЧАНИЕ

Если дуга $e = (u, v) \in E$ ведёт из узла u в узел v, то записи $w(e)$ и $w(u, v)$ обе считаются допустимыми и равноправными.

Если $M = \langle \overrightarrow{s,t} \rangle$ — путь из узла s в узел t, то *длиной пути* называется величина $W(M) = W(\langle \overrightarrow{s,t} \rangle) = W(s,t) = \sum\limits_{e \in M} w(e)$, а *длиной кратчайшего пути* называется величина $\Omega(s,t) = \min\limits_{\langle \overrightarrow{s,t} \rangle \subset E^*} W(s,t)$.

ЗАМЕЧАНИЕ

Поскольку неориентированные графы являются частным случаем орграфов, приведённые определения, утверждения и алгоритмы распространяются также и на графы. Поэтому изложение ведётся исключительно в терминах орграфов.

ТЕОРЕМА. *Для взвешенных орграфов выполняется неравенство треугольника:* $\Omega(s,v) \leqslant \Omega(s,u) + w(u,v)$.

Доказательство. Пусть существует кратчайший путь из s в v. Тогда его длина не превышает длины любого другого пути $\langle \overrightarrow{s,v} \rangle$ и, в частности, пути, состоящего из кратчайшего пути из s в u и дуги (u, v). □

Взвешенный орграф представим в виде *матрицы длин дуг* W : **array** $[1..p, 1..p]$ **of** $\mathbb{R}$, где

$$W[i,j] = \begin{cases} 0, & \text{для } i = j, \\ w(i,j), & \text{конечная величина, если есть дуга из узла } i \text{ в узел } j, \\ +\infty, & \text{если нет дуги из узла } i \text{ в узел } j. \end{cases}$$

Для хранения информации о последовательности узлов в кратчайших путях на орграфе используется *матрица предшествования* Π : **array** $[1..p, 1..p]$ **of** $1..p$, где

$$\Pi[i,j] = \begin{cases} k, \text{ если } k - \text{ узел, достигаемый} \\ \quad \text{на кратчайшем пути из } i \text{ непосредственно перед узлом } j; \\ 0, \text{ если из узла } i \text{ в узел } j \text{ нет пути.} \end{cases}$$

Если требуется хранить информацию только о путях из одного узла или, в частности, о единственном пути между двумя узлами, то матрицу можно сократить до вектора Π : **array** $[1..p]$ **of** $0..p$.

ОТСТУПЛЕНИЕ

Матрица Π размера $O(p^2)$ хранит информацию обо *всех* (кратчайших) путях в графе. Заметим, что всего в графе $O(p^2)$ путей, состоящих из $O(p)$ узлов. Таким образом, непосредственное представление всех путей потребовало бы памяти объема $O(p^3)$. Экономия памяти при использовании матрицы предшествования достигается за счет *интерпретации представления*, то есть динамического вычисления некоторой части информации вместо её хранения в памяти. В данном случае любой конкретный путь $\langle u, v\rangle$ легко извлекается из матрицы с помощью следующего алгоритма.

$w := v$; **yield** w // последний узел
while $w \neq u$ **do**
  $w := \Pi[u, w]$;
  yield w // предыдущий узел
end while

Полученную последовательность можно обратить, если это потребуется, и получить прямой порядок узлов пути.

Во многих алгоритмах данного раздела используются две процедуры.

Процедура инициализации $\mathrm{Init}(s)$ строит начальное состояние матрицы длин путей и матрицы предшествования.
Вход: узел s // $\mathrm{Init}(s)$
Выход: заполненные матрицы T : **array** $[1..p, 1..p]$ **of** $\mathbb{R}$ длин путей и
  Π : **array** $[1..p, 1..p]$ **of** $0..p$ самих путей.
  for v **from** 1 **to** p **do**
    $T[v] := +\infty$, $\Pi[v] := 0$
  end for
  $T[s] := 0$

Процедура *ослабления*, или *релаксации*, — $\mathrm{Relax}(s, v, u)$ проверяет, возможно ли улучшить известный путь из узла s в узел v, проведя новый путь через узел u, и обновляет пути, если это возможно.
Вход: узлы s, v, u // $\mathrm{Relax}(s, v, u)$,
  матрица W : **array** $[1..p, 1..p]$ **of** $\mathbb{R}$ длин дуг,
  матрица T : **array** $[1..p, 1..p]$ **of** $\mathbb{R}$ длин путей,
  матрица Π : **array** $[1..p, 1..p]$ **of** $0..p$ самих путей.
Выход: обновленные матрицы T, Π
  if $T[s, v] > T[s, u] + W[u, v]$ **then**
    $T[s, v] := T[s, u] + W[u, v]$; $\Pi[s, v] := u$ // новый путь короче
  end if

Процедура релаксации обладает рядом очевидных, но важных свойств.

1. **Верхняя граница (безопасность)**. Для всех пар узлов $T[s,v] \geqslant \Omega[s,v]$, причём $T[s,v]$ не изменяется после того, как станет равным $\Omega[s,v]$.
2. **Отсутствие пути**. Если из узла s в t нет пути, то $T[s,t] = \Omega(s,t) = +\infty$.
3. **Сходимость**. Если $\langle \overrightarrow{s,u} \rangle, (u,v)$ — кратчайший путь $\langle \overrightarrow{s,v} \rangle$ и до ослабления $\mathrm{Relax}(s,v,u)$ выполняется равенство $T[s,u] = \Omega(s,u)$, то после ослабления $T[s,v] = \Omega(s,v)$.
4. **Ослабление пути**. Если $M = \langle s, v_1, ..., v_k, t \rangle$ — кратчайший путь $\langle \overrightarrow{s,t} \rangle$ и ослабление производится в порядке $\mathrm{Relax}(s,s,v_1)$, $\mathrm{Relax}(s,v_1,v_2), \ldots, \mathrm{Relax}(s,v_k,t)$, то $T[s,t] = \Omega(s,t)$, и это свойство выполняется вне зависимости от других операций ослабления.

ЗАМЕЧАНИЕ

Если в графе G есть контур с отрицательным весом, то на этом контуре можно «накручивать» сколь угодно короткий путь (вплоть до $-\infty$). В этой ситуации решение задачи о поиске кратчайшего пути, вообще говоря, не определено. Однако, в зависимости от конкретной прикладной задачи, данная ситуация может быть интерпретирована по-разному: как полное отсутствие решения, или как решение, содержащее путь бесконечного отрицательного веса, или ещё как-то. В любом случае, способность предсказуемо реагировать на контуры отрицательного веса является важной характеристикой алгоритмов поиска кратчайших путей.

8.6.2. Кратчайшие пути в бесконтурном орграфе

Существует эффективный алгоритм, который позволяет найти в орграфе *дерево кратчайших путей*, растущее из заданного узла, даже если длины дуг могут быть отрицательными, но известно, что орграф не содержит контуров.

ЗАМЕЧАНИЕ

Совокупность путей от узла s к другим узлам образует корневое дерево в смысле п. 9.2.1. Действительно, никакой определённый кратчайший путь, очевидно, не содержит контуров. Всякий узел достижим из корня по построению. Более того, этот путь единственный, поскольку, даже если в узел ведет несколько кратчайших путей одинаковой длины, алгоритм выберет из них только один.

ЛЕММА. *В произвольном бесконтурном орграфе $G(V,E)$ узлы можно перенумеровать так, что $\forall\,(v_i,v_j) \in E\ \ (i<j)$.*

Доказательство. По теореме 2 п. 7.5.4 отношение достижимости в бесконтурном графе есть строгое частичное упорядочение на конечном множестве. Применяя алгоритм топологической сортировки (п. 1.8.4), получаем требуемую нумерацию узлов. □

ЗАМЕЧАНИЕ

В алгоритме используется множество «предшествования» $\Gamma^{-1}(v) \stackrel{\text{Def}}{=} \{u \mid v \in \Gamma(u)\}$, которое совпадает со списками смежности $\Gamma(v)$ для графов и не пересекается с $\Gamma(v)$ для направленных орграфов.

Алгоритм 8.5. Определение расстояний от источника в бесконтурном графе

Вход: взвешенный орграф $G(V,E)$, матрица весов W : **array** $[1..p,1..p]$ **of** $\mathbb{R}$, списки предшествующих узлов Γ^{-1}; источник $s=1$.
Выход: вектор T : **array** $[1..p]$ **of** $\mathbb{R}$ длин кратчайших путей от источника, вектор Π : **array** $[1..p]$ **of** $0..p$ самих путей.

```
Init(s)
for v from 1 to p do
  for u ∈ Γ⁻¹(v) do
    Relax(1, v, u)
  end for
end for
```

Обоснование. Докажем индукцией по v, что основной цикл имеет инвариант $\forall\, u \in 1..v \;\; (T[u]{=}\Omega(1,u))$. База: если $v=1$, то $T[1]=0=\Omega(1,1)$. Пусть теперь $\forall\, u < v \;\; (T[u]{=}\Omega(1,v))$. В кратчайшем пути $\langle 1,\overrightarrow{v}\rangle$ все промежуточные узлы имеют номера больше 1 и меньше v по построению орграфа, в частности, узел u, предшествующий v на кратчайшем пути, будет выбран во внутреннем цикле, для него по индукционному предположению $T[u]{=}\Omega(1,u)$ и, значит, $T[v]{=}\Omega(1,v)$. □

Оценка времени работы алгоритма складывается из времени топологической сортировки, которая выполняется за время $O(p+q)$, инициализации — $O(p)$ и итераций двойного цикла, которых ровно q, следовательно время имеет оценку $O(q)$. Так получаем $O(p+q)+O(p)+O(q)=O(p+q)$.

8.6.3. Алгоритм Беллмана–Форда

Алгоритм Беллмана[1]*–Форда* также решает задачу о построении дерева кратчайших путей, но на взвешенных орграфах общего вида.

Алгоритм 8.6. Беллмана–Форда

Вход: взвешенный орграф $G(V,E)$, матрица весов W : **array** $[1..p,1..p]$ **of real**, источник s.
Выход: вектор T : **array** $[1..p]$ **of real** длин кратчайших путей от источника, вектор Π : **array** $[1..p]$ **of** $0..p$ самих путей.

```
Init(s)  // Инициализация
for i from 1 to p − 1 do
  for (u, v) ∈ E do
    Relax(s, v, u)
  end for
end for
// проверка на отрицательные контуры
for (u, v) ∈ E do
  if T[v] > T[u] + W[u, v] then
    stop  // найден контур отрицательного веса
  end if
end for
```

[1] Ричард Эрнст Беллман (1920–1984).

Обоснование. Корректность алгоритма основывается на очевидном утверждении, что кратчайший путь содержит не более чем $p-1$ дугу при условии отсутствия контуров отрицательного веса, доступных из источника. Соответственно, если прорелаксировать все дуги взвешенного орграфа $p-1$ раз, то по свойствам релаксации (в особенности по свойствам ослабления пути и безопасности) результатом станет корректное дерево кратчайших путей.

Если провести дополнительную релаксацию всех дуг, и при этом дерево кратчайших путей изменится, то этот факт по свойству безопасности войдет в противоречие с отсутствием контуров отрицательного веса, и можно за счёт этого выдавать предупреждение об их наличии. □

ЗАМЕЧАНИЕ

Иногда может иметь смысл добавить в алгоритм проверку, что изменений дерева на прошедшей итерации внешнего цикла не произошло, поскольку часто количество дуг во всех кратчайших путях оказывается существенно меньше $p-1$. Выбор в пользу этого решения делается исходя из конкретной реализации и структуры задачи.

Оценка времени работы алгоритма тривиальна. Время инициализации — $O(p)$. Вызовов процедуры ослабления в точности $(p-1)\cdot q$, а значит, асимптотическая оценка времени работы составляет $O(p) + (p-1)q \cdot O(1) = O(p) + O((p-1)q) = O(pq)$. Очевидно, это медленнее, чем $O(p+q)$ алгоритма поиска на бесконтурном орграфе. Однако следует напомнить, что множество графов, на которых корректен алгоритм Беллмана–Форда, покрывает, в отличие от предыдущего алгоритма, практически весь спектр задач поиска дерева кратчайших путей.

8.6.4. Алгоритм Дейкстры

Алгоритм Дейкстры[1], безусловно, является наиболее популярным алгоритмом поиска дерева кратчайших путей. Однако он имеет как преимущества, так и недостатки относительно алгоритма Беллмана–Форда, что разобрано ниже в рамках обоснования и оценки времени работы.

ОТСТУПЛЕНИЕ

Дейкстра предложил свой алгоритм для решения задач, связанных с разводкой печатных плат. В этом случае вес дуги — это геометрическое расстояние, и оно всегда положительно. В положительно взвешенном орграфе алгоритм Дейкстры всегда применим и находит решение, поскольку контуры отрицательного веса невозможны. Поэтому в литературе часто область применимости алгоритма Дейкстры не обсуждается, и рассматривается применение только к положительно взвешенным графам и орграфам (см., например, [15], [2]). На самом деле класс орграфов, в которых применим алгоритм Дейкстры, существенно шире и заслуживает отдельного исследования.

Назовём *рекордом пути* $M = \langle \overrightarrow{s,t} \rangle$ (обозначение $R(\langle \overrightarrow{s,t} \rangle)$ максимальный вес начального отрезка этого пути, $R(\langle \overrightarrow{s,t} \rangle) \stackrel{\text{Def}}{=} \max\limits_{v \in M-s} W(s,v)$. Отдельно отметим, что $v \neq s$, то есть рекорд не может достигаться в начальном узле по определению.

Из определения следует, что $\forall\, \langle \overrightarrow{s,t} \rangle \ \Big(R(\langle \overrightarrow{s,t} \rangle) \geqslant W(\langle \overrightarrow{s,t} \rangle) \geqslant \Omega(s,t) \Big)$.

Теорема. *Результат работы алгоритма Дейкстры на взвешенном направленном*

[1] Эдсгер Вибе Дейкстра (1930–2002).

Алгоритм 8.7. Алгоритм Дейкстры поиска кратчайших путей

Вход: взвешенный орграф $G(V, E)$, матрица весов W : **array** $[1..p, 1..p]$ **of real**, источник s.
Выход: вектор T : **array** $[1..p]$ **of real** длин кратчайших путей от источника, вектор Π : **array** $[1..p]$ **of** $0..p$ самих путей.

```
Init(s) // инициализация
Q := V // контейнер (очередь с приоритетами)
while Q ≠ ∅ do
  u := ExtractMin(Q) // извлечение узла с минимальным значением T[u]
  if T[u] = ∞ then stop end if // остальные узлы недостижимы из s
  for v ∈ Γ(u) do
    if v ∈ Q then
      Relax(s, v, u) // релаксация дуги (u, v)
    end if
  end for
end while
```

орграфе без контуров отрицательного веса есть дерево кратчайших путей с корнем в узле s тогда и только тогда, когда для каждого узла $v \in G - s$, достижимого из узла s, рекорд кратчайшего пути $\langle s, \overrightarrow{v} \rangle$ меньше, чем рекорд любого не кратчайшего пути $\langle s, \overrightarrow{v} \rangle$.

Доказательство. Начнём с некоторых наблюдений относительно работы алгоритма Дейкстры. После инициализации массив T содержит оценки сверху длин кратчайших путей, ведущих в соответствующие узлы из узла s. В процессе работы алгоритма эти оценки уменьшаются, пока не станут длинами кратчайших путей. На каждой итерации алгоритм выполняет операцию ExtractMin, извлекает узел u с наименьшим накопленным весом из контейнера Q и выполняет релаксацию исходящих из этого узла дуг. Во-первых, отметим, что по крайней мере одна итерация выполняется в любом случае, поскольку при инициализации полагаем $T[s] := 0 < \infty$. Во-вторых, если $T[u] = \infty$, то и все оставшиеся узлы в Q имеют оценку ∞, а это означает, что ни разу не проводилась релаксация дуг, направленных в узлы Q из уже рассмотренных узлов, а это, в свою очередь, означает, что оставшиеся узлы в Q недостижимы из узла s, возможная часть дерева кратчайших путей уже построена, и работу алгоритма можно прервать. В-третьих, алгоритм в любом случае строит именно дерево, представленное массивом Π, с корнем в s, поскольку при первой релаксации входящей дуги, когда оценка ∞ меняется на конечную величину, к дереву добавляется лист, а при последующих релаксациях, когда оценка уменьшается, происходит переназначение предка для узла u, причём вновь назначаемый предок не является потомком узла u по построению, так что контуров появиться не может. В-четвёртых, после того как узел u извлечён из контейнера, его оценка уже не может измениться, поскольку релаксируются только дуги, ведущие в узлы контейнера Q. В-пятых, для извлекаемого узла $T[u] \leqslant R(\langle s, \overrightarrow{u} \rangle)$, поскольку все узлы на пути из s уже были извлечены ранее.

[Достаточность] Пусть для любого узла u, достижимого из узла s (недостижимые узлы можно не рассматривать), рекорд кратчайшего пути M_0 меньше рекордов альтернативных путей из s в u: $\forall \langle s, \overrightarrow{u} \rangle \ \Big(R(M_0) \leqslant R(\langle s, \overrightarrow{u} \rangle) \Big)$, причём равенство достигается, только если путь $\langle s, \overrightarrow{u} \rangle$ — кратчайший. Покажем по индукции, что на каждой

итерации алгоритма Дейкстры при извлечении узла u путь $M_0 = \langle s, \overrightarrow{u} \rangle$, определяемый значением $\Pi[u]$, имеет рекорд меньше, чем рекорды альтернативных путей. Тем самым, по условию теоремы этот путь кратчайший, а значит, после каждой итерации построено частичное дерево кратчайших путей из узла s. База: на первой итерации всегда выбирается узел s, который образует корень дерева. Длина кратчайшего пути $\Omega(s, s) = 0$, поскольку контуры отрицательного веса запрещены. Индукционное предположение: пусть на предыдущих итерациях дерево построено корректно, рассмотрим итерацию, на которой из контейнера Q выбран узел u. Случай $T[u] = \infty$ можно не рассматривать.

Тогда существует путь $M_0 = \langle s, \overrightarrow{u} \rangle$, причём узел u получил свою оценку в результате релаксации дуги (v, u), рис. 8.13. Если путь M_0 единственный, то его рекорд наименьший, индукционный переход доказан. Рассмотрим возможные альтернативные пути. Возможен путь M_1, в котором все узлы, кроме последнего, уже исключены из Q на предыдущих итерациях. Возможен путь M_2, в котором есть узлы, принадлежащие Q.

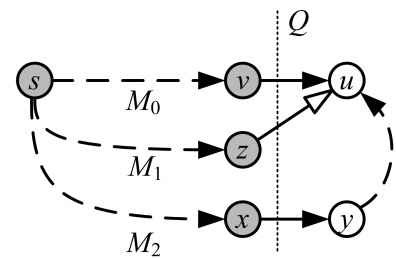


Рис. 8.13. К доказательству теоремы 8.6.4

В случае пути M_1 обе дуги, (v, u) и (z, u), уже участвовали в релаксации, и в построенном дереве сохранена та, которая даёт меньшую оценку, поэтому путь M_1 не оказывает влияния на построенное дерево, и его можно не учитывать. Остаётся рассмотреть путь M_2, состоящий из узлов, входящих и не входящих в Q. Далее от противного. Пусть $R(M_2) < R(M_0)$. Рассмотрим те узлы, на которых рекорды путей достигаются. Пусть $u_0 \in M_0$ — место достижения рекорда пути M_0. Тогда $T[u_0] = R(M_0)$, поскольку в пути M_0 все оценки — уже точные веса отрезков путей (узел u_0 может быть узлом u, v или любым промежуточным узлом). Рассмотрим ту итерацию, когда был выбран узел u_0. Пусть u_2 — первый ещё не извлечённый узел на пути M_2 в этот момент (узел u_2 может быть узлом y, x или любым промежуточным узлом). Тогда $T[u_0] = R(M_0) > R(M_2) \geqslant T[u_2]$, что противоречит тому, что узел u_0 извлечён раньше узла u_2.

[Необходимость] Пусть алгоритм Дейкстры построил дерево кратчайших путей, и пусть существует узел u такой, что в него ведут два пути: M_0 — кратчайший, построенный алгоритмом, и M_2 — не кратчайший. Тогда рассмотрим узел $u_0 \in M_0$, в котором рекорд достигается, то есть $T[u_0] = R(M)$, и узел $u_2 \in M_2$ — первый узел на пути M_2 такой, что $u_2 \in Q$. По выбору алгоритма имеем $T[u_0] \leqslant T[u_2]$, а значит $R(M_0) = T[u_0] \leqslant T[u_2] \leqslant R(M_2)$. Если $R(M_0) < R(M_2)$, то теорема доказана, а если $T[u_0] = T[u_2]$, то имеет место недетерминированный случай: алгоритм может выбрать «верный» узел u_0 или «ложный» узел u_2. Если принять допущение, что алгоритму Дейкстры может «не повезти», и может быть выбран «ложный» узел, то для гарантии правильности работы необходимо строгое неравенство рекордов. □

Пример. На рисунках приведены примеры работы алгоритма на орграфах с положительными и отрицательными весами дуг.

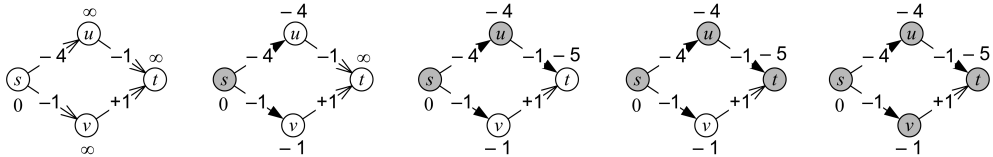


Рис. 8.14. Правильная работа алгоритма Дейкстры

На рис.8.14 рекорд кратчайшего пути s, u, t равен -4, в то время как рекорд пути s, v, t равен 0, и алгоритм работает правильно.

На рис.8.15 рекорд кратчайшего пути s, u, t равен $+1$, в то время как рекорд пути s, v, t равен 0, и алгоритм работает неправильно.

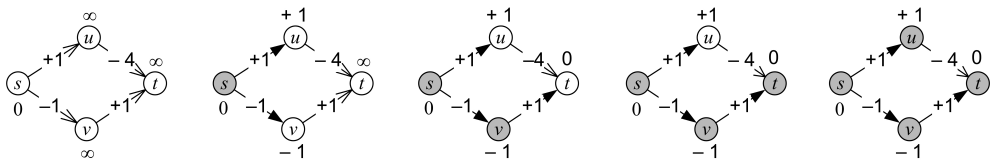


Рис. 8.15. Неправильная работа алгоритма Дейкстры

СЛЕДСТВИЕ. *Алгоритм Дейкстры корректен на классе неотрицательно взвешенных орграфов.*

Доказательство. Неотрицательно взвешенный орграф очевидно не содержит контуров отрицательного веса. Далее заметим, что из неотрицательности весов следует, что рекорд любого пути достигается в его конечном узле, соответственно, рекорд кратчайшего пути меньше рекордов альтернативных или равен им в конечном узле. □

Поскольку алгоритм Дейкстры не имеет в себе никакой разумной реакции на контуры отрицательного веса, а анализ орграфов на выполнение необходимого и достаточного условия обычно сложен, алгоритм повсеместно применяется только для неотрицательно взвешенных орграфов, что покрывает подавляющее большинство прикладных задач поиска пути.

ЗАМЕЧАНИЕ

Алгоритм Дейкстры легко применим к решению задачи о поиске кратчайшего пути между двумя узлами. Для этого достаточно после операции ExtractMin(Q) завершить работу, если извлеченный узел u является целевым, поскольку $T[u]$ для него больше не изменится.

Время работы алгоритма Дейкстры зависит от реализации контейнера Q с учётом вектора T и операций над ним: начальное заполнение $Q := V$, уменьшение $T[v]$ в процедуре релаксации и операция ExtractMin(Q). При работе алгоритма каждая дуга будет ослаблена не более чем один раз — поскольку начальные узлы дуг u не повторяются, что в сумме произойдет не более чем q раз. Пусть Q — массив, тогда одна вставка выполняется за $O(1)$, а все вставки — за $O(p)$, уменьшения значений

T займут в сумме $qO(1) = O(q)$, а извлечения минимума — $pO(p) = O(p^2)$. Итого имеем $O(p) + O(q) + O(p^2) = O(p^2 + q) = O(p^2)$. Пусть ExtractMin и уменьшение T выполняются за $O(\log p)$, а $Q := V$ — за $O(p)$. Тогда рассчёт времени работы алгоритма будет следующим: $O(p) + pO(\log p) + qO(\log p) = O((p+q)\log p)$, что будет меньше $O(p^2)$, если $q < p^2/\log p$. Такими свойствами обладают *неубывающие пирамиды*.

8.6.5. Алгоритм Флойда–Уоршалла

В отличие от предыдущих алгоритмов, алгоритм Флойда[1]–Уоршалла решает задачу поиска не дерева кратчайших путей из источника, а всех кратчайших путей в орграфе, и делает это за время $O(p^3)$.

Алгоритм 8.8. Флойда–Уоршалла поиска всех кратчайших путей

Вход: матрица $W[1..p, 1..p]$ длин дуг.
Выход: матрица $T[1..p, 1..p]$ длин путей и матрица $\Pi[1..p, 1..p]$ самих путей.

```
// инициализация
for i from 1 to p do
  for j from 1 to p do
    T[i,j] := W[i,j]
    if C[i,j] = ∞ then
      Π[i,j] := 0  // нет дуги из i в j
    else
      Π[i,j] := j  // есть дуга из i в j
    end if
  end for
end for
for i from 1 to p do
  for j from 1 to p do
    for k from 1 to p do
      if i ≠ j & T[j,i] ≠ ∞ & i ≠ k & T[i,k] ≠ ∞ & (T[j,k] = ∞ ∨ T[j,k] > T[j,i] + T[i,k]) then
        T[j,k] := T[j,i] + T[i,k]  // запомнить длину нового пути
        Π[j,k] := Π[j,i]  // и сам путь
      end if
    end for
  end for
  for j from 1 to p do
    if T[j,j] < 0 then
      stop  // Узел j входит в отрицательный контур
    end if
  end for
end for
```

Обоснование. Алгоритм Флойда–Уоршалла является обобщением алгоритма Уоршалла (п. 1.5.2). Покажем по индукции, что после выполнения i-го шага основного цикла по i элементы матриц $T[j,k]$ и $\Pi[j,k]$ содержат, соответственно, длину кратчайшего пути и первый узел на кратчайшем пути из узла j в узел k, проходящем через промежуточные узлы из диапазона $1..i$. База: $i = 0$, то есть до начала цикла

[1] Роберт Флойд (1936–2001).

элементы матриц T и Π содержат информацию о кратчайших путях (если таковые есть), не проходящих ни через какие промежуточные узлы. Пусть теперь перед началом выполнения тела цикла на i-м шаге $T[j,k]$ содержит длину кратчайшего пути от j к k, а $\Pi[j,k]$ содержит первый узел (если таковой есть) на кратчайшем пути из узла j в узел k. В таком случае, если в результате добавления узла i к диапазону промежуточных узлов находится более короткий путь (в частности, если это вообще первый найденный путь), то он записывается. Таким образом, после окончания цикла, когда $i = p$, матрицы содержат кратчайшие пути, проходящие через промежуточные узлы $1..p$, то есть искомые кратчайшие пути. Алгоритм не всегда выдаёт решение, поскольку оно не всегда определено. Дополнительный цикл по j служит для прекращения работы в случае обнаружения в орграфе контура с отрицательным весом. □

8.6.6. Алгоритмы с предобработкой

Как уже было сказано, алгоритм Дейкстры обычно применяется только к неотрицательно взвешенным орграфам. Однако в некоторых случаях можно привести исходный орграф к неотрицательно взвешеному (провести *предобработку*) и получить алгоритм поиска пути, обладающий рядом свойств как алгоритма предобработки, так и алгоритма Дейкстры.

ОТСТУПЛЕНИЕ

Предобработка — это весьма общий приём конструирования новых эффективных алгоритмов на основе имеющихся алгоритмов. Иначе говоря, на примере алгоритма Дейкстры, задаётся отображение входа нового алгоритма во вход алгоритма Дейкстры и отображение выхода алгоритма Дейкстры в выход нового алгоритма. Сам алгоритм Дейкстры обеспечивает отображение своего входа в свой выход с присущей ему эффективностью.

Предобработка имеет смысл, если предстоит решать не разовую, а массовую задачу. Скажем, искать кратчайшие пути на графе, представляющем карту автомобильных дорог. Такая задача в навигационных приложениях должна решаться многие тысячи раз в день, поэтому затрат времени на предобработку графа не жаль, если предобработка позволяет ускорить выполнение основной задачи поиска кратчайшего пути.

Рассмотрим алгоритм поиска кратчайших путей между всеми вершинами орграфа, в котором используется предобработка типа «перевзвешивание» («reweighting») — введение функции потенциала. Каждому узлу v присваивается *потенциал* $h(v)$. Веса дуг переопределяются соответственно: для каждой дуги (u,v) определяется новый вес $w'(u,v) := w(u,v) + h(u) - h(v)$. Потенциалы определяются так, чтобы новые веса были неотрицательными. Можно показать, что такое преобразование сохраняет отношение порядка весов путей между узлами u и v:

$$W'(\langle\overrightarrow{u,v}\rangle) = w(u,p_1) + h(u) - h(p_1) + w(p_1,p_2) + h(p_1) - h(p_2) + \ldots$$
$$+ w(p_n,v) + h(p_n) - h(v) = W(\langle\overrightarrow{u,v}\rangle) + h(u) - h(v) = W(\langle\overrightarrow{u,v}\rangle) + c.$$

Поскольку прибавление константы c сохраняет отношение порядка весов путей, по результатам предобработки с помощью алгоритма Дейкстры очевидным образом находятся кратчайшие пути на исходном орграфе.

Здесь необходимо поставить вопрос о поиске указанных выше потенциалов $h(v)$. Один из простейших методов приведён ниже.

1. Присоединить фиктивный узел s с дугами нулевого веса ко всем реальным узлам орграфа.

2. Воспользоваться алгоритмом Беллмана–Форда для поиска кратчайших путей из узла s во все реальные узлы орграфа. Нетрудно понять, что для неотрицательно взвешенного орграфа веса всех таких путей будут нулевыми, в то время как для других орграфов это, вообще говоря, не так.
3. Взять $\Omega(s, v)$ в качестве $h(v)$, то есть искомых потенциалов, после чего удалить фиктивный узел и ведущие из него дуги.

ЗАМЕЧАНИЕ

По наследству от алгоритма Беллмана–Форда метод будет работоспособен для всех взвешенных орграфов без циклов отрицательного веса и выдаст предупреждение в случае наличия таких циклов.

Обоснование. Предложенный способ построения потенциальной функции корректен. Действительно, если в неравенство треугольника для взвешенных орграфов $\Omega(s, v) \leqslant \Omega(s, u) + w((u, v))$ подставить $h(v) = \Omega(s, v)$, $h(u) = \Omega(s, u)$, то получаем $h(v) \leqslant h(u) + w(u, v)$, и тогда $w'(u, v) = w(u, v) + h(u) - h(v) \geqslant 0$. □

Оценим трудоёкость этого решения. Пусть предложенный алгоритм используется для поиска одного дерева кратчайших путей. Предварительная обработка алгоритмом Беллмана–Форда — $O(pq)$, применение алгоритма Дейкстры — $O(p \log_2(p) + q)$, обратное преобразование дерева — $O(p)$. В сумме имеем
$O(pq) + O(p \log_2(p) + q) + O(p) = O(pq)$, что, очевидно, не даёт никакого выигрыша в сравнении с алгоритмом Беллмана–Форда.

Однако если применить алгоритм для поиска p деревьев кратчайших путей, предварительную обработку потребуется провести только один раз. Это и есть алгоритм с предобработкой:

1) однократно провести предобработку;
2) найти p деревьев кратчайших путей алгоритмом Дейкстры;
3) сделать p обратных преобразований, выдать ответ.

Время работы алгоритма с предобработой (при оптимальной реализации алгоритма Дейкстры) составит $O(pq) + O(p \cdot (p \log_2(p) + q)) + pO(p) = O(p^2 \log_2(p) + pq)$, что лучше, чем алгоритм Флойда–Уоршалла ($O(p^3)$), если q асимптотически меньше, чем p^2.

ЗАМЕЧАНИЕ

Изложение центрального результата этой главы — теоремы Менгера и сопутствующего материала — в основном следует [9]. Алгоритм нахождения максимального потока в сети заимствован из [13] с небольшими модификациями. Доказательство необходимого и достаточного условия корректности алгоритма Дейкстры является оригинальным.

Глава 9 Деревья

Деревья заслуживают отдельного и подробного рассмотрения по двум причинам:

- Деревья являются в некотором смысле простейшим классом графов. Для них выполняются многие интересные утверждения, которые не всегда выполняются для графов в общем случае. Применительно к деревьям многие доказательства и рассуждения оказываются намного проще. Выдвигая какие-то гипотезы при решении задач теории графов, целесообразно сначала их проверять на деревьях.
- Деревья являются самым распространённым классом графов, применяемых в программировании, причём в самых разных ситуациях. Более половины объёма этой главы посвящено рассмотрению конкретных применений деревьев в программировании.

9.1. Свободные деревья

Изучение деревьев целесообразно начать с самых общих определений и установления основных свойств.

9.1.1. Свободные деревья и их элементы

Граф без циклов называется *ациклическим*, или *лесом*. Связный ациклический граф называется (*свободным*) *деревом*. Таким образом, компонентами связности леса являются деревья.

ЗАМЕЧАНИЕ

Прилагательное «свободное» употребляется в том случае, когда нужно подчеркнуть отличие деревьев от других объектов, родственных деревьям: ориентированных деревьев, упорядоченных деревьев и т. д.

В связном графе G выполняется неравенство $q(G) \geqslant p(G) - 1$ (п. 8.1.4). Граф G (не обязательно связный!), в котором $q(G) = p(G) - 1$, называется *древочисленным*.

В ациклическом графе G имеем $z(G) = 0$. Пусть u, v— несмежные вершины графа G, $x = (u, v) \notin E$. Если граф $G + x$ имеет ровно один простой цикл, $z(G + x) = 1$, то граф G (не обязательно ациклический!) называется *субциклическим*.

ЗАМЕЧАНИЕ

Графы $K_3 \cup K_1$ и $K_3 \cup K_2$ являются древочисленными и субциклическими и в то же время не являются ни связными, ни ациклическими.

Ребро, соединяющее несмежные вершины свободного дерева, называется *хордой*. Хорда дерева не принадлежит дереву!

Примеры

На рис. 9.1, 9.2, 9.3 последовательно показаны диаграммы всех различных (свободных) деревьев с 4, 5 и 6 вершинами.

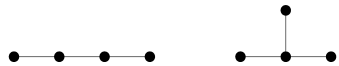


Рис. 9.1. Свободные деревья с 4 вершинами

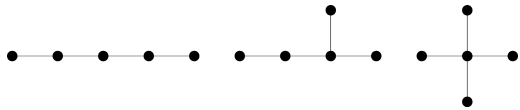


Рис. 9.2. Свободные деревья с 5 вершинами

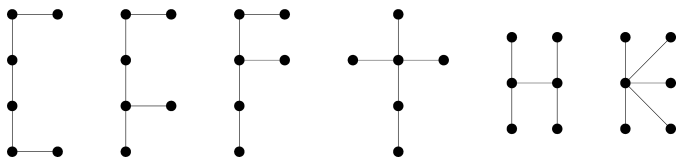


Рис. 9.3. Свободные деревья с 6 вершинами

9.1.2. Основные свойства деревьев

Следующая теорема устанавливает, что два из четырех свойств — связность, ацикличность, древочисленность и субцикличность — характеризуют граф как дерево.

ТЕОРЕМА. *Пусть $G(V, E)$ — граф с p вершинами, q рёбрами, k компонентами связности и z простыми циклами. Пусть далее x — ребро, соединяющее любую пару несмежных вершин в G. Тогда следующие утверждения эквивалентны:*

1) *G— дерево, то есть связный граф без циклов, $k(G) = 1 \ \& \ z(G) = 0$;*
2) *любые две вершины соединены в G единственной простой цепью, $\forall\, u, v \ \ (|P(u, v)| = 1)$;*
3) *G— связный граф, и любое ребро есть мост, $k(G) = 1 \ \& \ \forall\, e \in E \ \ (k(G - e) > 1)$;*
4) *G— связный и древочисленный, $k(G) = 1 \ \& \ q(G) = p(G) - 1$;*
5) *G— ациклический и древочисленный, $z(G) = 0 \ \& \ q(G) = p(G) - 1$;*
6) *G— древочисленный и субциклический (за двумя исключениями), $q(G) = p(G) - 1 \ \& \ G \neq K_1 \cup K_3 \ \& \ G \neq K_2 \cup K_3 \ \& \ z(G + x) = 1$;*
7) *G— ациклический и субциклический, $z(G) = 0 \ \& \ z(G + x) = 1$;*
8) *G— связный, субциклический и неполный, $k(G) = 1 \ \& \ G \neq K_p \ \& \ p \geqslant 3 \ \& \ z(G + x) = 1$.*

Доказательство.

[1$\Longrightarrow$2] От противного. Пусть существуют две цепи $\langle u, v\rangle$. Некоторые вершины этих цепей различны. Обозначим через w_1 первую вершину при перечислении вершин от u к v, такую, что следующие вершины в цепях различны, а через w_2 обозначим первую вершину при перечислении вершин от v к u, такую, что следующие вершины в цепях различны (рис. 9.4 слева). Рассмотрим отрезок $\langle w_1, w_2\rangle$ первой цепи при перечислении вершин от u к v и отрезок $\langle w_2, w_1\rangle$ второй цепи при

перечислении вершин от v к u. Тогда $\langle w_1, w_2\rangle + \langle w_2, w_1\rangle$— цикл, что противоречит ацикличности графа G.

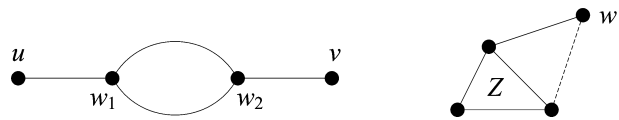


Рис. 9.4. К доказательству теоремы о свойствах деревьев $[1 \Longrightarrow 2]$, $[6 \Longrightarrow 7]$

[2⟹3] Любые две вершины соединены цепью (единственной), следовательно, $k(G) = 1$. Далее от противного. Пусть ребро x— не мост. Тогда в $G - x$ концы этого ребра связаны цепью. Само ребро x— вторая цепь.

[3⟹4] Индукция по p. База: $p = 1 \Longrightarrow q = 0$. Пусть $q(G) = p(G) - 1$ для всех связных графов G с числом вершин меньше p, у которых любое ребро является мостом. Тогда удалим из графа G некоторое ребро x (которое является мостом). Получим две компоненты, G' и G'', удовлетворяющие индукционному предположению. Имеем

$$q' = p' - 1,\ q'' = p'' - 1,\ q = q' + q'' + 1 = p' - 1 + p'' - 1 + 1 = p - 1.$$

[4⟹5] От противного. Пусть есть цикл с n вершинами и n рёбрами. Остальные $p - n$ вершин имеют инцидентные им рёбра, которые связывают их с циклом. Следовательно, $q \geqslant p$, что противоречит условию $q = p - 1$.

[5⟹6] Граф G— ациклический, следовательно, $G \neq K_2 \cup K_3$, $G \neq K_1 \cup K_3$. Далее от противного. Пусть $z(G + x) \neq 1$. Если $z(G + x) \geqslant 2$, то $x \in Z_1$, $x \in Z_2$, где Z_1 и Z_2— вновь образованные циклы в графе $G + x$. Но тогда $Z_1 \cup Z_2 \setminus \{x\}$— цикл в графе G, что противоречит ацикличности графа G.

Если же $z(G + x) = 0$, то удалим из графа $G + x$ ребро x. Получим две компоненты, G_1 и G_2. Графы G_1 и G_2— ациклические. Убедимся, что G_1 и G_2 являются связными. Если не связны они оба, то $q(G_1) < p(G_1) - 1$ и $q(G_2) < p(G_2) - 1$, то есть $q(G) = q(G_1) + q(G_2) < p(G_1) - 1 + p(G_2) - 1 = p(G) - 2$, что противоречит древочисленности G. Если несвязным является только один из них, пусть G_1, то G_2 связный ($q(G_2) \geqslant p(G_2) - 1$) и ациклический ($q(G_2) < p(G_2)$), что даёт древочисленность. Тогда $p(G_1) = p(G) - p(G_2) = q(G) + 1 - (q(G_2) + 1) = q(G_1)$, что противоречит условию ацикличности G_1 (п. 7.3.5). То есть G_1 и G_2— связные и ациклические, а значит, древочисленные. Имеем $q(G) = q(G_1) + q(G_2) = p(G_1) - 1 + p(G_2) - 1 = p(G) - 2$ и получаем $q(G) = p(G) - 2$, что противоречит условию древочисленности G.

[6⟹7] От противного. Пусть в G есть цикл $Z = C_n$. Если $n > 3$, то рассмотрим пары вершин цикла, которые не соединены рёбрами цикла. Если среди них уже есть смежные вершины, имеем три цикла. Если среди них нет смежных вершин, то, соединив несмежные вершины в Z, получим три цикла. Следовательно, в цикле нет вершин, не соединённых рёбрами цикла, значит $Z = C_3$. Этот цикл Z является компонентой связности G. Действительно, пусть это не так. Тогда существует вершина w, смежная с Z. Если w смежна более чем с одной вершиной Z, то имеем больше одного цикла. Если w смежна только с одной вершиной Z, то, соединив её с другой вершиной, получим два цикла (см. рис. 9.4 справа). Рассмотрим $G' := G - Z$. Имеем $p = p' + 3$, $q = q' + 3$. Но $q = p - 1$, следовательно, $q' = p' - 1$. Отсюда $z(G') = 0$, так как один цикл уже есть. Следовательно, компоненты G'— деревья. Пусть их k.

Имеем $q' = \sum_{i=1}^{k} q_i = \sum_{i=1}^{k} (p_i - 1) = \sum_{i=1}^{k} p_i - k = p' - k$, но $q' = p' - 1$, следовательно, $k = 1$, то есть дерево одно. Если в этом дереве соединить несмежные вершины, то получим второй цикл. Два исключения: деревья, которые не имеют несмежных вершин,— это K_1 и K_2.

[7$\Longrightarrow$8] При $p \geqslant 3$ граф K_p содержит цикл, следовательно, $G \neq K_p$. Далее от противного. Пусть G несвязен, тогда при соединении ребром двух компонент связности цикл не возникнет, что противоречит субцикличности.

[8$\Longrightarrow$1] От противного. Пусть в графе G есть единственный цикл (большее количество циклов противоречит субцикличности графа G). Если в G нет висячих вершин и $p = 3$, то $G = K_3$, что противоречит условию неполноты G. Если в связном G нет висячих вершин, $p > 3$ и есть единственный цикл, то $G = C_p$. Соединим любые две несмежные вершины u и v ребром x (рис. 9.5, слева). Имеем три цикла в $G + x$, что противоречит условию субцикличности. Если в G есть висячая вершина u, соединим её с любой несмежной с ней вершиной v ребром x. Вершина v может принадлежать (рис. 9.5, в центре) или не принадлежать циклу (рис. 9.5, справа). Теперь из u в v есть две или три простые цепи (одна или две цепи были раньше, так как G— связный с одним простым циклом, и появилась цепь из ребра x). Имеем два, три или более циклов в графе $G + x$, что противоречит условию субцикличности. □

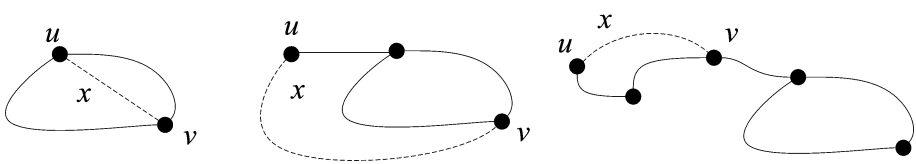


Рис. 9.5. К доказательству теоремы о свойствах деревьев [$8 \Longrightarrow 1$]

СЛЕДСТВИЕ 1. *В любом нетривиальном дереве имеются по крайней мере две висячие вершины.*

Доказательство. Рассмотрим дерево $G(V, E)$. Дерево— связный граф, следовательно, $\forall\, v_i \in V \;\; (d(v_i) \geqslant 1)$. Далее от противного. Пусть $\forall\, i \in 1..p-1 \;\; (d(v_i) > 1)$. Тогда $2q = \sum_{i=1}^{p} d(v_i) > 2(p-1) + 1 = 2p - 1$. Но $q = p - 1$, то есть $2q = 2p - 2$. Имеем противоречие: $2p - 2 > 2p - 1$. □

ЗАМЕЧАНИЕ

Легко видеть, в частности, что висячими вершинами в дереве являются концы любого диаметра.

СЛЕДСТВИЕ 2. *Каждая не висячая вершина свободного дерева является точкой сочленения.*

Доказательство. Пусть $G(V,E)$— дерево, $v \in V$ и $d(v) > 1$. Тогда по определению $\exists\, u, w \in V \;\; (u \neq w \;\&\; (u,v) \in E \;\&\; (v,w) \in E)$. Граф G связен, поэтому существует цепь $\langle u, w\rangle$. Если $v \notin \langle u, w\rangle$, то имеем цикл $v, \langle u, w\rangle, v$, что противоречит тому, что G— дерево. Следовательно, $\exists\, u, w \in V \;\; (\forall \langle u, w\rangle \;\; (v \in \langle u, w\rangle))$, и по теореме 1 п. 8.1.2 вершина v— точка сочленения. □

Следствие 3. *Если в связном графе нет висячих вершин, то в нём есть цикл.*

Доказательство. От противного. Если связный граф не имеет циклов, то он является деревом и по следствию 1 обязан иметь висячие вершины. □

9.1.3. Центр дерева

Свободные деревья выделяются из других графов тем, что их центр всегда оправдывает своё название.

Теорема. *Центр свободного дерева состоит из одной вершины или из двух смежных вершин:* $(z(G) = 0 \;\&\; k(G) = 1) \Longrightarrow (C(G) = K_1 \vee C(G) = K_2)$.

Доказательство. Для деревьев K_1 и K_2 утверждение теоремы очевидно. Пусть теперь $G(V,E)$ — некоторое свободное дерево, отличное от K_1 и K_2. Рассмотрим граф $G'(V',E')$, полученный из G удалением всех висячих вершин. Заметим, что G'— дерево, поскольку ацикличность и связность при удалении висячих вершин сохраняется. Далее, если эксцентриситет $e_G(v) = d(v,u)$, то u— висячая вершина в дереве G (иначе можно было бы продолжить цепь «за» вершину u). Поэтому $\forall\, v \in V' \;\; (e_G(v) = e_{G'}(v) + 1)$, и при удалении висячих вершин эксцентриситеты оставшихся уменьшаются на 1. Следовательно, при удалении висячих вершин центр не меняется, $C(G) = C(G')$. Поскольку дерево G конечно, то, удаляя на каждом шаге все висячие вершины, в конце концов за несколько шагов придём к K_1 или K_2. □

9.2. Ориентированные, упорядоченные и бинарные деревья

Ориентированные (упорядоченные) деревья являются абстракцией иерархических отношений, которые очень часто встречаются как в практической жизни, так и в математике и программировании. Дерево (ориентированное) и иерархия— это равнообъёмные понятия.

9.2.1. Ориентированные деревья

Ориентированным деревом (или *ордеревом*, или *корневым* деревом) называется орграф со следующими свойствами:

1) существует единственный узел r, полустепень захода которого равна 0, $d^+(r) = 0$; он называется *корнем* ордерева;
2) полустепень захода всех остальных узлов равна 1, $\forall\, v \in V - r \;\; (d^+(v) = 1)$;
3) каждый узел достижим из корня, $\forall\, v \in V - r \;\; \left(\exists \langle \overrightarrow{r, v}\rangle\right)$.

Пример. На рис. 9.6 приведены диаграммы всех различных ориентированных деревьев с 3 узлами, а на рис. 9.7 показаны диаграммы всех различных ориентированных деревьев с 4 узлами.

Рис. 9.6. Ориентированные деревья с 3 узлами

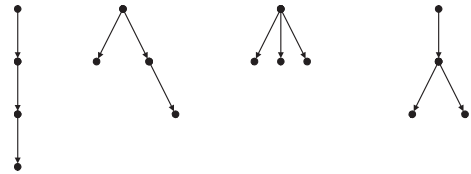


Рис. 9.7. Ориентированные деревья с 4 узлами

ТЕОРЕМА. *Ордерево обладает следующими свойствами:*

1) *ордерево древочисленно, $q = p - 1$;*
2) *если в ордереве забыть ориентацию дуг, то получится свободное дерево;*
3) *в ордереве нет контуров;*
4) *для каждого узла существует единственный путь, ведущий в этот узел из корня;*
5) *подграф, определяемый множеством узлов, достижимых из узла v, является ордеревом с корнем v (это ордерево называется поддеревом узла v);*
6) *если в свободном дереве любую вершину назначить корнем, то получится ордерево.*

Доказательство.

[1] Каждая дуга входит в какой-то узел. Из п. 2 определения этого параграфа имеем $\forall\, v \in V - r \;\; (d^+(v) = 1)$, где r— корень. Следовательно, $q = p - 1$.

[2] Пусть G— ордерево, граф G' получен из G забыванием ориентации рёбер, r— корень. Тогда $\forall\, v_1, v_2 \in V \;\; (\exists \langle v_1, r\rangle \in G' \;\&\; \exists \langle r, v_2\rangle \in G')$, следовательно, $\forall\, v_1, v_2 \;\; (\exists \langle v_1, v_2\rangle)$ и граф G' связен. Таким образом, учитывая п. 4. теоремы п. 9.1.2, G'— дерево.

[3] Следует из п. 2.

[4] От противного. Если бы в G существовали два пути из r в v, то в G' имелся бы цикл.

[5] Пусть G_v— правильный подграф, определяемый множеством узлов, достижимых из v. Тогда $d^+_{G_v}(v) = 0$, иначе узел v был бы достижим из какого-то узла $v' \in G_v$ и, таким образом, в G_v, а значит, и в G имелся бы контур, что противоречит пункту 3. Далее имеем: $\forall\, v' \in G_v - v \;\; (d^+(v') = 1)$, так как $G_v \subset G$. Все узлы G_v достижимы из v по построению. По определению получаем, что G_v— ордерево.

[6] Пусть вершина r назначена корнем и дуги последовательно ориентированы «от корня» обходом в глубину. Тогда $d^+(r) = 0$ по построению; $\forall\, v \in V - r \;\; (d^+(v) = 1)$, так как входящая дуга появляется при первом посещении узла; все узлы достижимы из корня, так как обход в глубину посещает все вершины связного графа. Таким образом, по определению получаем ордерево. □

СЛЕДСТВИЕ. *Алгоритм поиска в глубину строит ордерево с корнем в начальном узле.*

ЗАМЕЧАНИЕ

Каждую вершину в свободном дереве с p вершинами можно назначить корнем и получить ордерево. Некоторые из полученных ордеревьев могут оказаться изоморфными. Следовательно, свободное дерево определяет не более p различных ориентированных деревьев. Таким образом, общее число различных ордеревьев с p узлами не более чем в p раз превосходит общее число различных свободных деревьев с p вершинами.

Концевая вершина ордерева называется *листом*. Множество листьев называется *кроной*. Путь из корня в лист называется *ветвью*. Длина наибольшей ветви ордерева называется его *высотой*. *Уровень* узла ордерева— это расстояние от корня до узла. Сам корень имеет уровень 0. Узлы одного уровня образуют *ярус* ордерева.

ЗАМЕЧАНИЕ

Наряду с «растительной» применяется еще и «генеалогическая» терминология. Узлы, достижимые из узла u, называются *потомками* узла u (потомки одного узла образуют поддерево). Если узел v является потомком узла u, то узел u называется *предком* узла v. Если в дереве существует дуга (u, v), то узел u называется *отцом* (или *родителем*) узла v, а узел v называется *сыном* узла u. Сыновья одного отца называются *братьями*.

9.2.2. Эквивалентное определение ордерева

Ордерево T— это непустое конечное множество узлов, на котором определено разбиение, обладающее следующими свойствами:

1) имеется один выделенный одноэлементный блок $\{r\}$, называемый корнем данного ордерева;
2) остальные узлы (исключая корень) содержатся в k ($k \geqslant 0$) блоках $T_1, \ldots, T_k$, каждый из которых, в свою очередь, является ордеревом и называется *поддеревом*: $T \stackrel{\text{Def}}{=} \{\{r\}, T_1, \ldots, T_k\}$.

Нетрудно видеть, что данное определение эквивалентно определению п. 9.2.1. Достаточно построить орграф, проводя дуги от заданного корня ордерева r к корням поддеревьев $T_1, \ldots, T_k$ и далее повторяя рекурсивно этот процесс для каждого из поддеревьев.

9.2.3. Упорядоченные деревья

Если относительный порядок поддеревьев $T_1, \ldots, T_k$ в эквивалентном определении ордерева фиксирован, то ордерево называется *упорядоченным*.

Примеры

Ориентированные и упорядоченные ориентированные деревья интенсивно используются в программировании. Несколько примеров представлены на рис. 9.8.

1. Для представления выражений языков программирования, как правило, используются ориентированные упорядоченные деревья. Пример представления выражения $a + b * c$ показан на рис. 9.8, *а*.
2. Для представления блочной структуры программы и связанной с ней структуры областей определения идентификаторов часто используется ориентированное дерево (может быть, неупорядоченное, так как порядок определения переменных в

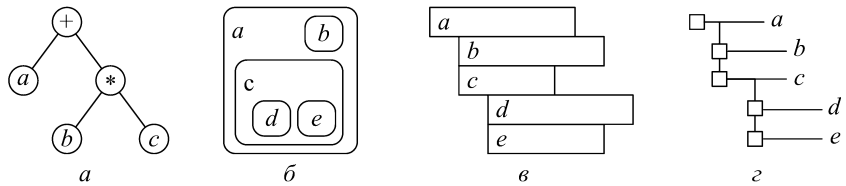


Рис. 9.8. Примеры изображения деревьев в программировании

блоке в большинстве языков программирования считается несущественным). На рис. 9.8, *б* показана структура областей определения идентификаторов a, b, c, d, e, причём для отображения иерархии использованы вложенные области.

3. Для представления иерархической структуры вложенности элементов данных и (или) операторов управления часто используется техника отступов, показанная на рис. 9.8, *в*.
4. Структура вложенности каталогов и файлов в современных операционных системах является упорядоченным ориентированным деревом. Для изображения таких деревьев часто применяется способ, показанный на рис. 9.8, *г*.
5. Различные «правильные скобочные структуры» (например, $(a(b)(c(d)(e)))$) являются ориентированными упорядоченными деревьями.

ОТСТУПЛЕНИЕ

Тот факт, что большинство систем управления файлами использует ориентированные деревья, отражается даже в терминологии, например: «корневой каталог диска».

ЗАМЕЧАНИЕ

Общепринятой практикой при изображении деревьев является соглашение о том, что корень находится наверху и все дуги ориентированы сверху вниз, поэтому стрелки можно не изображать. Таким образом, диаграммы свободных, ориентированных и упорядоченных деревьев оказываются графически неразличимыми, и требуется дополнительное уточнение, дерево какого класса изображено на диаграмме. В большинстве случаев это ясно из контекста.

Пример. На рис. 9.9 приведены три диаграммы деревьев, которые внешне выглядят различными. Как упорядоченные деревья они действительно все различны: $a \neq б$, $б \neq в$, $в \neq a$. Как ориентированные деревья $a = б$, но $б \neq в$. Как свободные деревья они все изоморфны: $a = б = в$.

Указанное в п. 9.2.2 построение позволяет определить на упорядоченном ордереве единый линейный порядок узлов, согласованный с уже заданными в дереве упорядочениями.

ТЕОРЕМА. *В упорядоченном ордереве с p узлами существует такая нумерация узлов числами из диапазона $1..p$, что номера потомков больше номеров предков и номера старших братьев больше номеров младших братьев.*

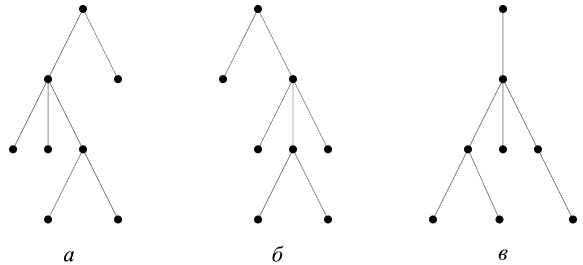


Рис. 9.9. Диаграммы деревьев

Доказательство. Применим к упорядоченному ордереву алгоритм обхода в глубину, начиная с корня, причём узлы, смежные с данным, посещаются в порядке, определяемом порядком поддеревьев. Присвоим узлам номера в порядке первого посещения. Тогда все узлы получат уникальные номера из диапазона $1..p$, и корень получит номер 1. Далее, старшие братья получат номера, бóльшие, чем номера младших братьев по условию обхода, а потомки получат номера бóльшие, чем номера предков, поскольку алгоритм обхода в глубину посещает предков раньше, чем потомков. □

ЗАМЕЧАНИЕ

Указанный порядок обхода часто называют *прямым*.

Пример. Узлы упорядоченного ордерева на рис. 9.10 при прямом обходе получат следующие номера: $a \mapsto 1$, $b \mapsto 2$, $c \mapsto 5$, $d \mapsto 3$, $e \mapsto 4$, $f \mapsto 6$, $g \mapsto 7$, $h \mapsto 8$, $i \mapsto 9$.

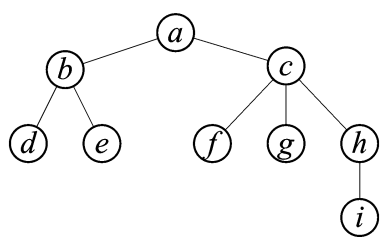


Рис. 9.10. Пример ориентированного упорядоченного дерева

ЗАМЕЧАНИЕ

Построенная в доказательстве теоремы нумерация не является единственной нумерацией, обладающей указанными свойствами.

Пример. Можно обойти упорядоченное ордерево по ярусам. При этом узлы упорядоченного ордерева на рис. 9.10 получат следующие номера: $a \mapsto 1$, $b \mapsto 2$, $c \mapsto 3$, $d \mapsto 4$, $e \mapsto 5$, $f \mapsto 6$, $g \mapsto 7$, $h \mapsto 8$, $i \mapsto 9$.

9.2.4. Бинарные деревья

Бинарное (или *двоичное*) дерево— это непустое конечное множество узлов, на котором определена структура, обладающая следующими свойствами:

1) имеется один выделенный узел r, называемый корнем данного бинарного дерева;
2) остальные узлы (исключая корень) содержатся в двух непересекающихся множествах (поддеревьях) — левом и правом, каждое из которых, в свою очередь, либо пусто, либо является бинарным деревом.

На первый взгляд может показаться, что бинарное дерево — это частный случай упорядоченного ориентированного дерева, в котором у каждого узла не более двух смежных. Но это не так, бинарное дерево *не является* упорядоченным ордеревом. Дело в том, что даже если у некоторого узла бинарного дерева имеется только одно непустое поддерево, то всё равно известно, какое именно это поддерево: левое или правое.

Пример. На рис. 9.11 приведены две диаграммы деревьев, которые изоморфны как упорядоченные, ориентированные и свободные деревья, но не изоморфны как бинарные деревья.

Рис. 9.11. Два различных бинарных дерева

ЗАМЕЧАНИЕ

Понятие двоичного дерева допускает обобщение. *m-ичным деревом* называется непустое конечное множество узлов, которое состоит из корня и m непересекающихся подмножеств, имеющих номера $1, \ldots, m$, каждое из которых, в свою очередь, либо пусто, либо является m-ичным деревом. Большая часть утверждений и алгоритмов для двоичных деревьев может быть сравнительно легко распространена и на m-ичные деревья (с соответствующими модификациями). Поэтому, хотя на практике m-ичные деревья и используются достаточно часто, здесь мы ограничиваемся только двоичными деревьями.

9.3. Представление деревьев в программах

Обсуждению представлений деревьев можно предпослать в точности те же рассуждения, что были предпосланы обсуждению представлений графов. Кроме того, следует подчеркнуть, что задача представления деревьев в программе встречается гораздо чаще, чем задача представления графов общего вида, а потому методы её решения оказывают ещё большее влияние на практику программирования.

9.3.1. Представление свободных деревьев

Для представления деревьев можно использовать те же приёмы, что и для представления графов общего вида — матрицы смежности и инциденций, списки смежности и др. Но, используя особенные свойства деревьев, можно предложить существенно более эффективные представления.

Рассмотрим следующее представление свободного дерева, известное как *код Прюфера*. Допустим, что вершины дерева $T(V, E)$ пронумерованы числами из интервала $1..p$. Построим последовательность A : **array** $[1..p-1]$ **of** $1..p$ в соответствии с алгоритмом 9.1.

Алгоритм 9.1. Построение кода Прюфера свободного дерева

Вход: Дерево $T(V, E)$ в любом представлении, вершины дерева пронумерованы числами $1..p$ произвольным образом.
Выход: Массив A : **array** $[1..p-1]$ **of** $1..p$ — код Прюфера дерева T.

```
for i from 1 to p − 1 do
    v := min {k ∈ V | d(k) = 1}  // выбираем висячую вершину v
    A[i] := Γ(v)  // заносим в код номер единственной вершины, смежной с v
    V := V − v  // удаляем вершину v из дерева
end for
```

Пример. Для дерева, представленного на рис. 9.12, код Прюфера $7, 9, 1, 7, 2, 2, 7, 1, 2, 5, 12$. На этом рисунке числа в вершинах— это их номера, а числа на рёбрах указывают порядок, в котором будут выбираться висячие вершины и удаляться рёбра при построении кода Прюфера.

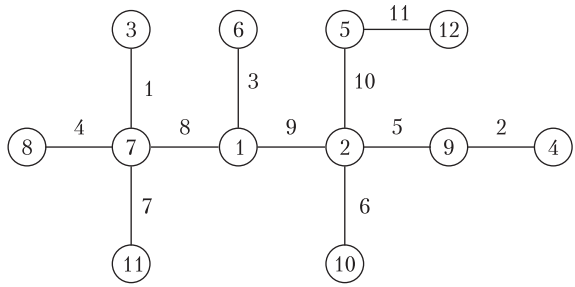


Рис. 9.12. Построение кода Прюфера

По построенному коду можно восстановить исходное дерево алгоритмом 9.2.

Обоснование. Код Прюфера действительно является представлением свободного дерева. Чтобы убедиться в этом, покажем, что если T'— дерево, построенное алгоритмом распаковки по коду A, который построен алгоритмом упаковки по дереву T, то T' изоморфно T, $T' \sim T$. Для этого установим отображение $f\colon 1..p \to 1..p$ между номерами вершин в деревьях T и T' по порядку выбора вершин в алгоритмах: если вершина v выбрана на i-м шаге алгоритма упаковки, то вершина $f(v)$ выбрана на i-м шаге алгоритма распаковки. Заметим, что $\mathrm{Dom}\, f = 1..p$, поскольку висячие вершины есть в любом дереве, и удаление висячей вершины оставляет дерево деревом. Далее, $\mathrm{Im}\, f = 1..p$, поскольку на i-м шаге алгоритма распаковки использовано $i-1$ число из p чисел и остаётся $p-i+1$ чисел, а в хвосте кода Прюфера занято не более $p-i$ чисел. Более того, $\forall\, v\ \ (f(v) = v)$. Действительно, номера вершин, которые являются висячими в исходном дереве, не появляются в коде Прюфера (кроме, может быть, одной висячей вершины с наибольшим номером), а номера вершин, которые не являются висячими, обязательно появляются. Поскольку при выборе первой вершины

Алгоритм 9.2. Распаковка кода Прюфера свободного дерева

Вход: Массив $A:$ **array** $[1..p-1]$ **of** $1..p$— код Прюфера дерева T.
Выход: Дерево $T(V,E)$, заданное множеством рёбер E, вершины дерева пронумерованы числами $1..p$.

```
E := ∅  // вначале множество рёбер пусто
B := 1..p  // множество неиспользованных номеров вершин
for i from 1 to p − 1 do
  // выбираем вершину v— неиспользованную вершину с наименьшим
  // номером, который не встречается в остатке кода Прюфера
  v := min {k ∈ B | ∀ j ⩾ i  (k ≠ A[j])}
  E := E + (v, A[i])  // добавляем ребро (v, A[i])
  B := B − v  // удаляем вершину v из списка неиспользованных
end for
```

v в алгоритме упаковки все вершины с меньшими номерами не являются висячими, их номера будут присутствовать в коде и, значит, не могут быть использованы на первом шаге алгоритма распаковки. Таким образом, на первом шаге алгоритм распаковки выберет ту же вершину v. Но после удаления вершины v на втором шаге снова имеется дерево, к которому применимы те же рассуждения. Итак, f— тождественное и, значит, взаимно-однозначное отображение. Заметим теперь, что для определения i-го элемента кода на i-м шаге алгоритма упаковки используется, а затем удаляется ребро $(v, A[i])$ и в точности то же ребро добавляется в дерево T' на i-м шаге алгоритма распаковки. Следовательно, f— взаимно-однозначное отображение, сохраняющее смежность и $T \sim T'$. □

ЗАМЕЧАНИЕ

Код Прюфера — наиболее экономное по памяти представление дерева. Его можно немного улучшить, если заметить, что существует всего одно дерево с двумя вершинами — K_2, а потому информацию о «последнем ребре» можно не хранить, она восстанавливается однозначно.

Пример. Для дерева на рис. 9.12 код Прюфера 7, 9, 1, 7, 2, 2, 7, 1, 2, 5, 12. Рассмотрим протокол выполнения распаковки кода Прюфера:

i	B	A	v	$A[i]$
1	$\{1,2,3,4,5,6,7,8,9,10,11,12\}$	$[\|7,9,1,7,2,2,7,1,2,5,12]$	3	7
2	$\{1,2,4,5,6,7,8,9,10,11,12\}$	$[7,\|9,1,7,2,2,7,1,2,5,12]$	4	9
3	$\{1,2,5,6,7,8,9,10,11,12\}$	$[7,9,\|1,7,2,2,7,1,2,5,12]$	6	1
4	$\{1,2,5,7,8,9,10,11,12\}$	$[7,9,1,\|7,2,2,7,1,2,5,12]$	8	7
5	$\{1,2,5,7,9,10,11,12\}$	$[7,9,1,7,\|2,2,7,1,2,5,12]$	9	2
6	$\{1,2,5,7,10,11,12\}$	$[7,9,1,7,2,\|2,7,1,2,5,12]$	10	2
7	$\{1,2,5,7,11,12\}$	$[7,9,1,7,2,2,\|7,1,2,5,12]$	11	7
8	$\{1,2,5,7,12\}$	$[7,9,1,7,2,2,7,\|1,2,5,12]$	7	1
9	$\{1,2,5,12\}$	$[7,9,1,7,2,2,7,1,\|2,5,12]$	1	2
10	$\{2,5,12\}$	$[7,9,1,7,2,2,7,1,2,\|5,12]$	2	5
11	$\{5,12\}$	$[7,9,1,7,2,2,7,1,2,5,\|12]$	5	12

9.3.2. Представление упорядоченных корневых деревьев

В упорядоченных ордеревьях выделен корень и поддеревья упорядочены. Использование этой информации позволяет построить компактное представление ориентированных упорядоченных деревьев. Пусть упорядоченное ордерево задано списками смежности (п. 7.4.4), причём узлы перенумерованы в соответствии с порядком прямого обхода, определённым в п. 9.2.3. Тогда следующий алгоритм упаковки построит представление упорядоченного ордерева с q дугами в виде битовой шкалы (кода) из $2q$ разрядов.

Алгоритм 9.3. Построение кода упорядоченного ордерева

Вход: списки смежности Γ : **array** $[1..p]$ **of** $\uparrow N$ упорядоченного ордерева T,
Выход: массив C : **array** $[1..2q]$ **of** 0..1, являющийся кодом ордерева T.

```
i := 0  // количество заполненных разрядов кода
TraverseTree(Γ[1])  // указатель на список смежности корня
```

Основная работа выполняется рекурсивной процедурой TraverseTree, для которой переменная i и массивы Γ и C глобальны, а элемент списка дуг определён следующим образом: $N =$ **struct {** v : $1..p$; n : $\uparrow N$ **}**.

Вход: $d : \uparrow N$ — указатель на список исходящих дуг.
Выход: заполнение двух разрядов кода C.

```
while d ≠ nil do
  i := i + 1; C[i] := 1  // отмечаем вход в узел
  TraverseTree(Γ[d.v])  // построение кода поддерева
  i := i + 1; C[i] := 0  // отмечаем выход из узла
  d := d.n  // выбор следующего узла в списке
end while
```

ЗАМЕЧАНИЕ

Представленный алгоритм, в сущности, строит протокол выполнения алгоритма обхода в глубину упорядоченного ордерева. Запись 1 отмечает вход в узел по единственной входящей дуге, запись 0 отмечает возврат из узла по этой же дуге.

По построенному коду нетрудно восстановить исходное представление упорядоченного ордерева.

В алгоритме используются вспомогательный стек S для хранения номеров узлов и две процедуры:

1) NewNode(v : $1..p, n$: $\uparrow N$) : $\uparrow N$ — функция, создающая новый элемент списка смежности с полями v и n и возвращающая указатель на него;
2) Append(L, e : $\uparrow N$) — процедура, присоединяющая элемент, указатель на который задан параметром e, к списку, указатель на первый элемент которого задан параметром L.

Обоснование. Массив C является представлением упорядоченного ордерева T, то есть если к T применить алгоритм кодирования, а затем к результату применить алгоритм декодирования, то получится то же самое упорядоченное ордерево T. Действительно, алгоритм декодирования интерпретирует протокол работы алгоритма кодирования. Каждый раз, когда алгоритм упаковки входит по дуге в некоторый узел первый раз, он записывает в протокол 1, и в точности в том же порядке

Алгоритм 9.4. Восстановление упорядоченного ордерева по коду

Вход: Массив C : **array** $[1..2q]$ **of** $0..1$— код упорядоченного ордерева T.
Выход: Массив Γ : **array** $[1..p]$ **of** $\uparrow N$, где $N =$ **struct {** $v : 1..p;\ n : \uparrow N$ **}**— списки смежности упорядоченного ордерева T.

```
p := 1  // счётчик узлов
Γ[1] := nil  // корень
n := 1  // текущий узел
for i from 1 to 2q do
  if C[i] = 1 then
    p := p + 1  // номер нового узла
    Γ[p] := nil  // новый узел пока листовой
    d := NewNode(p, nil)  // дуга от текущего узла к новому узлу
    Append(Γ[n], d)  // добавить узел в список смежности
    n → S  // положить номер текущего узла на стек
    n := p  // перейти в новый узел
  else
    n ← S  // снять со стека и перейти в новый узел
  end if
end for
```

алгоритм распаковки создаёт узлы. Каждый раз, когда алгоритм упаковки возвращается на предыдущий уровень и записывает в протокол 0, алгоритм распаковки возвращается к родительскому узлу, снимая его номер со стека. □

Пример. Применение алгоритма к упорядоченному ордереву на рис. 9.10 даст код 1101001101011000.

ЗАМЕЧАНИЕ

Наложенное здесь требование прямого порядка нумерации узлов не является ограничением и используется только для упрощения изложения. Можно показать, что при *любой* нумерации узлов ордерева алгоритм кодирования построит код, а алгоритм декодирования восстановит по этому коду изоморфное ордерево, отличающееся разве что нумерацией узлов.

9.3.3. Число упорядоченных ориентированных деревьев

Представление упорядоченных ордеревьев, построенное в предыдущем параграфе, обладает замечательным характеристическим свойством, позволяющим получить явную формулу для числа упорядоченных ордеревьев. Введём обозначения. Пусть b : **array** $[1..n]$ **of** $0..1$— любая битовая шкала. Обозначим через $N_0(b,k)$— количество нулей в отрезке шкалы $b[1..k], k \leqslant n$, и через $N_1(b,k)$— количество единиц в отрезке шкалы $b[1..k], k \leqslant n$. Пусть теперь c : **array** $[1..2q]$ **of** $0..1$— битовая шкала, построенная по упорядоченному ордереву T алгоритмом 9.3. Тогда по построению алгоритма имеем

$$N_0(c,2q) = N_1(c,2q)\ \&\ \forall\, k < 2q\ \ (N_0(c,k) \leqslant N_1(c,k))\,. \qquad (*)$$

Из алгоритма 9.4 следует, что данное свойство является характеристическим, то есть *любая* шкала, обладающая свойством $(*)$, является кодом некоторого дерева, причём

по теореме п. 9.2.3 соответствие между деревьями и кодами взаимно-однозначно. Пусть S_n — множество битовых шкал длины $2n$:

$$S_n \stackrel{\text{Def}}{=} \{c = (a_1, \ldots, a_{2n}) \mid \forall\, i \in 1..2n \;\; (a_i \in \{0,1\})\}\,.$$

Обозначим число тех битовых шкал $c \in S_n$, которые обладают свойством $(*)$, через $C(n)$:

$$C(n) \stackrel{\text{Def}}{=} |\{c \in S_n \mid N_0(c,2n) = N_1(c,2n) \;\&\; \forall\, k < 2n \;\; (N_0(c,k) \leqslant N_1(c,k))\}|\,.$$

По определению $C(0) \stackrel{\text{Def}}{=} 1$.

Пример. Ясно, что $C(1) = |\{(1,0)\}| = 1$, $C(2) = 2$ — см. рис. 9.6, $C(3) = 4$ — см. рис. 9.7.

ЛЕММА. $C(n) = \sum\limits_{k=0}^{n-1} C(k)C(n-k-1)$.

Доказательство. Рассмотрим подмножество кодов $c \in S_n$, удовлетворяющих более сильному условию

$$N_0(c,2q) = N_1(c,2q) \;\&\; \forall\, k < 2q \;\; (N_0(c,k) < N_1(c,k))\,, \qquad (*')$$

и обозначим число таких кодов через $C'(n)$. Ясно, что в *любом* коде, удовлетворяющем условию $(*')$, первый бит равен 1, а последний — 0. Если их отбросить, то оставшаяся часть кода будет удовлетворять условию $(*)$, и поэтому $C'(n) = C(n-1)$, причём по определению $C'(1) = C(0) = 1$. Пусть теперь s — наименьшее число, такое, что код $c[1..s]$ удовлетворяет условию $(*')$. Сгруппируем все коды, удовлетворяющие условию $(*)$, по значениям числа s. В группе, в которой $s = n$, имеется $C'(n)$ кодов, а в группах, в которых $1 \leqslant s \leqslant n-1$, имеется $C'(s)C(n-s)$ кодов. Имеем

$$C(n) = C'(n) + \sum_{s=1}^{n-1} C'(s)C(n-s) = C(n-1)\cdot 1 + \sum_{s=1}^{n-1} C(s-1)C(n-s) =$$

$$= C(n-1)C(0) + \sum_{k=0}^{n-2} C(k)C(n-k-1) = \sum_{k=0}^{n-1} C(k)C(n-k-1),$$

где $k := s-1$. □

Таким образом, для числа $C(n)$ битовых шкал, удовлетворяющих условию $(*)$, выполняется характеристическое рекуррентное соотношение чисел Каталана (п. 5.7.4), откуда немедленно следует теорема.

ТЕОРЕМА. *Число ориентированных упорядоченных деревьев с q дугами равно*

$$\frac{C(2q,q)}{q+1}.$$

Алгоритм 9.5. Проверка правильности скобочной структуры

Вход: строка s : **array** $[1..n]$ **of char**, возможно, содержащая скобки «(» и «)».
Выход: число 0, если скобочная структура правильна, или число в диапазоне $1..(n+1)$, указывающее на позицию в строке, где скобочная структура нарушена.

```
p := 0  // число прочитанных открывающих минус число закрывающих скобок
for i from 1 to n do
  if s[i] = "(" then
    p := p + 1  // прочли открывающую скобку
  end if
  if s[i] = ")" then
    p := p − 1  // прочли закрывающую скобку
  end if
  if p < 0 then
    return i  // лишняя закрывающая скобка
  end if
end for
if p = 0 then
  return 0  // скобочная структура правильна
else
  return n + 1  // не хватает закрывающих скобок
end if
```

9.3.4. Проверка правильности скобочной структуры

Из характеристического свойства $(*)$ рассматриваемого представления упорядоченных ордеревьев в качестве побочного, но полезного наблюдения можно извлечь эффективный алгоритм 9.5 проверки правильности скобочной структуры.

Обоснование. Всякая правильная скобочная структура взаимно-однозначно соответствует упорядоченному ордереву. Если трактовать открывающую скобку как 1, а закрывающую— как 0, то последовательность скобок, содержащихся в строке, образует код дерева. Алгоритм очевидным образом проверяет выполнение условия $(*)$, то есть проверяет допустимость этого кода. □

Пример. Строка $a(b(d)(e))(c(f)(g)(h(i)))$ является естественной записью с помощью правильной скобочной структуры дерева на рис. 9.10.

9.3.5. Представление бинарных деревьев

Всякое свободное дерево можно ориентировать, назначив один из узлов корнем. Всякое ордерево можно произвольно упорядочить. Для потомков одного узла (братьев) упорядоченного ордерева определено отношение старше–младше (левее–правее). Всякое упорядоченное дерево можно представить бинарным деревом, например, проведя правую связь к старшему брату, а левую — к младшему сыну. Таким образом, достаточно рассмотреть представление в программе бинарных деревьев. Это наблюдение объясняет, почему представлению бинарных деревьев в программах традиционно уделяется особое внимание при обучении программированию.

Пример. На рис. 9.13 приведены диаграммы упорядоченного и соответствующего ему бинарного деревьев.

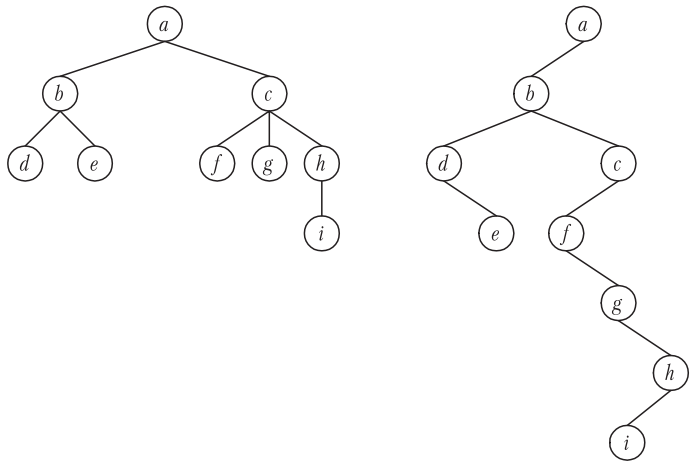


Рис. 9.13. Упорядоченное и бинарное деревья

ЗАМЕЧАНИЕ

Из данного представления следует, что множество бинарных деревьев взаимно-однозначно соответствует множеству упорядоченных лесов упорядоченных ордеревьев. Действительно, в указанном представлении одиночному упорядоченному ордереву всегда соответствует бинарное дерево, у которого правая связь корня пуста, а упорядоченному лесу — бинарное дерево, у которого правая связь корня не пуста.

Обозначим через $n(p)$ объем памяти, занимаемой представлением бинарного дерева, где p— число узлов. Наиболее часто используются следующие представления бинарных деревьев.

1. **Списочные структуры**: каждый узел представляется записью типа N, содержащей два поля (l и r) с указателями на левый и правый узлы и еще одно поле i для хранения указателя на информацию об узле. Дерево представляется указателем на корень. Тип N обычно определяется следующим образом: $N =$ **struct {** $i : info;\ l, r : \uparrow N$ **}**, где тип $info$ считается заданным. Для этого представления $n(p) = 3p$.

ЗАМЕЧАНИЕ

Поскольку в бинарном дереве, как и в любом другом, $q = p-1$, то из $2p$ указателей, отводимых для хранения дуг, $p+1$ всегда хранит значение **nil**, то есть половина связей не используется.

2. **Упакованные массивы**: все узлы располагаются в массиве, так что все узлы поддерева данного узла располагаются вслед за этим узлом. Вместе с каждым узлом хранится индекс узла, который является первым узлом правого поддерева данного узла. Дерево T обычно определяется следующим образом: $T :$ **array** $[1..p]$ **of struct {** $i : info,\ k : 1..p$ **}**, где тип $info$ считается заданным. Для этого представления $n(p) = 2p$.
3. **Польская запись**: аналогично, но вместо связей фиксируется «размеченная степень» каждого узла (например, 0 означает, что это лист, 1 — есть левая связь,

но нет правой, 2 — есть правая связь, но нет левой, 3 — есть обе связи). Дерево T определяется следующим образом: T : **array** $[1..p]$ **of struct {** $i : info$, $d : 0..3$ **}**, где тип $info$ считается заданным. Для этого представления $n(p) = 2p$. Если степень узла известна из информации, хранящейся в самом узле, то можно не хранить и степень. Такой способ представления деревьев называется *польской записью* и обычно используется для представления выражений. В этом случае представление дерева оказывается наиболее компактным: объем памяти $n(p) = p$.

Пример. Покажем, как выглядят в памяти бинарные деревья в разных представлениях. В приводимой ниже таблице первая группа столбцов соответствует полям списочных структур, вторая— упакованным массивам и третья— польской записи с явными размеченными степенями. Условные адреса— это целые числа, а пустой указатель— 0. Рассматривается бинарное дерево на рис. 9.13 справа.

Адрес	i	l	r	i	k	i	d
1	a	2	0	a	0	a	1
2	b	3	5	b	5	b	3
3	d	0	4	d	4	d	2
4	e	0	0	e	0	e	0
5	c	6	0	c	0	c	1
6	f	0	7	f	7	f	2
7	g	0	8	g	8	g	2
8	h	9	0	h	0	h	1
9	i	0	0	i	0	i	0

9.3.6. Обходы бинарных деревьев

Большинство алгоритмов работы с деревьями основаны на обходах. Возможны следующие основные обходы бинарных деревьев.

Прямой (*префиксный*, *левый*) обход: попасть в корень,
обойти левое поддерево,
обойти правое поддерево.

Внутренний (*инфиксный*, *симметричный*) обход: обойти левое поддерево,
попасть в корень,
обойти правое поддерево.

Концевой (*постфиксный*, *правый*) обход: обойти левое поддерево,
обойти правое поддерево,
попасть в корень.

Кроме трёх основных, возможны еще три соответствующих обхода, отличающихся порядком рассмотрения левых и правых поддеревьев. Этим исчерпываются обходы, если в представлении фиксированы только дуги, ведущие от отцов к сыновьям.

ЗАМЕЧАНИЕ

Если кроме связей «отец–сын» в представлении есть другие связи, то возможны и другие (более эффективные) обходы. Деревья, в которых пустые поля l и r в структуре N используются для хранения дополнительных связей, называются *прошитыми деревьями*.

Пример. Концевой обход дерева выражения $a + b * c$ дает *обратную* польскую запись этого выражения: $abc * +$.

ОТСТУПЛЕНИЕ

Польская запись выражений (прямая или обратная) применяется в некоторых языках программирования непосредственно и используется в качестве внутреннего представления программ во многих трансляторах и интерпретаторах. Причина заключается в том, что такая форма записи допускает очень эффективную интерпретацию (вычисление значения) выражений. Например, значение выражения в обратной польской записи может быть вычислено при однократном просмотре выражения слева направо с использованием одного стека. В таких языках, как Forth и PostScript, обратная польская запись используется как основная.

ЗАМЕЧАНИЕ

Всякое ордерево задаёт частичный порядок на множестве узлов, причём минимальный элемент единственный (корень). Всякий обход дерева задаёт строгий линейный порядок на множестве узлов. Прямой обход дерева задает такой линейный порядок на множестве узлов, который не противоречит исходному порядку. Некоторые другие обходы также сохраняют исходный порядок (например, поиск в ширину— обход *по уровням*). В то же время, часть обходов нарушает исходный порядок (например, концевой и симметричный обходы).

9.3.7. Алгоритмы симметричного обхода бинарного дерева

Программные реализации различных обходов бинарных деревьев однотипны. В качестве примера рассмотрим симметричный (инфиксный) обход. Реализация обхода бинарного дерева с помощью рекурсивной процедуры TraverseTree не вызывает затруднений.

Алгоритм 9.6. Рекурсивный алгоритм симметричного обхода бинарного дерева

Вход: бинарное дерево, представленное списочной структурой, p — указатель на корень.

Выход: последовательность узлов бинарного дерева в порядке симметричного обхода.

```
if p.l ≠ nil then TraverseTree(p.l) end if  // обойти левое поддерево
 yield p  // очередной узел при симметричном обходе
if p.r ≠ nil then TraverseTree(p.r) end if  // обойти правое поддерево
```

В некоторых случаях из соображений эффективности применение явной рекурсии оказывается нежелательным. Следующий очевидный алгоритм реализует наиболее популярный симметричный обход без явной рекурсии, но с использованием стека.

ЗАМЕЧАНИЕ

Добавление в списочную структуру указателя на родителя позволяет написать обходы деревьев без использования рекурсии или контейнера (стека, очереди).

Алгоритм 9.7. Алгоритм симметричного обхода бинарного дерева со стеком

Вход: бинарное дерево, представленное списочной структурой, r — указатель на корень.

Выход: последовательность узлов бинарного дерева в порядке симметричного обхода.

```
T := ∅; p := r  // вначале стек пуст и p указывает на корень дерева
M :  // анализируем узел, на который указывает p
if p = nil then
  if T = ∅ then
    stop  // обход закончен
  end if
  p ← T  // левое поддерево обойдено
  yield p  // очередной узел при симметричном обходе
  p := p.r  // начинаем обход правого поддерева
else
  p → T  // запоминаем текущий узел...
  p := p.l  // ...и начинаем обход левого поддерева
end if
goto M
```

9.4. Деревья сортировки

В этом разделе обсуждается одно конкретное применение деревьев в программировании, а именно *деревья сортировки* (также называемые *деревьями упорядочивания*). При этом рассматриваются как теоретические вопросы, связанные, например, с оценкой высоты деревьев, так и практическая реализация алгоритмов, а также целый ряд прагматических аспектов применения деревьев сортировки и некоторые смежные вопросы.

9.4.1. Ассоциативная память

В практическом программировании для организации хранения данных и доступа к ним часто используется механизм, который обычно называют *ассоциативной памятью*. При использовании ассоциативной памяти данные делятся на порции (называемые *записями*), и с каждой записью ассоциируется *ключ*. Ключ— это значение из некоторого линейно упорядоченного множества, а записи могут иметь произвольную природу и различные размеры. Доступ к данным осуществляется по значению ключа, которое обычно выбирается простым, компактным и удобным для работы.

Примеры

Ассоциативная память используется во многих областях жизни.

1. Толковый словарь или энциклопедия: записью является словарная статья, а ключом — заголовок словарной статьи.
2. Адресная книга: ключом является имя абонента, а записью — адресная информация (телефон(ы), почтовый адрес и т. д.).
3. Банковские счета: ключом является номер счета, а записью — финансовая информация (которая может быть очень сложной).

Таким образом, ассоциативная память должна поддерживать по меньшей мере три основные операции:

1) добавить (ключ, запись);
2) найти (ключ) : запись;
3) удалить (ключ).

Эффективность каждой операции зависит от структуры данных, используемой для представления ассоциативной памяти. Эффективность ассоциативной памяти в целом зависит от соотношения частоты выполнения различных операций в данной конкретной программе.

ЗАМЕЧАНИЕ

Таким образом, невозможно указать способ организации ассоциативной памяти, который оказался бы наилучшим во всех возможных случаях.

9.4.2. Способы реализации ассоциативной памяти

Для представления ассоциативной памяти используются разнообразные структуры данных:

1) неупорядоченный массив, в элементах которого хранятся значения ключей актуальных записей;
2) упорядоченный массив, в элементах которого хранятся значения ключей актуальных записей в порядке возрастания (или убывания);
3) *таблица расстановки* (или *хэш-таблица*);
4) неупорядоченный список, в элементах которого хранятся значения ключей актуальных записей;
5) упорядоченный список, в элементах которого хранятся значения ключей актуальных записей в порядке возрастания (или убывания);
6) *дерево сортировки* — бинарное дерево, каждый узел которого содержит ключ (и указатель на запись) и обладает следующим свойством: значения ключа во всех узлах левого поддерева меньше, а во всех узлах правого поддерева — больше, чем значение ключа в узле.

При использовании неупорядоченного массива алгоритмы реализации операций ассоциативной памяти очевидны.

1. Операция «добавить (ключ, запись)» реализуется добавлением записи в конец массива (трудоёмкость $O(1)$).
2. Операция «найти (ключ) : запись» реализуется проверкой в цикле всех записей в массиве (трудоёмкость $O(n)$).
3. Операция «удалить (ключ)» реализуется поиском удаляемой записи, а затем перемещением последней записи на место удаляемой (трудоёмкость $O(n)$).

Упорядоченные массивы обсуждаются в следующем параграфе.

При использовании неупорядоченного списка алгоритмы реализации операций ассоциативной памяти также очевидны и не дают особого преимущества по сравнению с массивом.

1. Операция «добавить (ключ, запись)» реализуется добавлением записи в начало списка (трудоёмкость $O(1)$).
2. Операция «найти (ключ): запись» реализуется проверкой в цикле всех записей в списке (трудоёмкость $O(n)$).

3. Операция «удалить (ключ)» реализуется поиском удаляемой записи, а затем исключением записи из списка (хотя перемещать элементы при этом нет нужды, трудоёмкость всё равно составляет $O(n)$).

ОТСТУПЛЕНИЕ

Таблицы расстановки являются чрезвычайно важным практическим приёмом программирования, подробное описание которого не включено в учебник из экономии места. Вкратце основная идея заключается в следующем. Подбирается специальная функция, которая называется *хэш-функцией*, переводящая значение ключа в адрес хранения записи (адресом может быть индекс в массиве, номер кластера на диске и т. д.). Таким образом, по значению ключа с помощью хэш-функции сразу определяется место хранения записи и открывается доступ к ней. Хэш-функция подбирается таким образом, чтобы разным ключам соответствовали, по возможности, разные адреса из диапазона возможных адресов записей. Как правило, мощность множества ключей существенно больше размера пространства адресов, которое, в свою очередь, больше количества одновременно хранимых записей. Поэтому при использовании хэширования возможны *коллизии*— ситуации, когда хэш-функция сопоставляет один и тот же адрес двум актуальным записям с различными ключами. Различные методы хэширования отличаются друг от друга способами разрешения коллизий и приёмами вычисления хэш-функций. Тщательная программная реализация и соблюдение ограничений на мощность множества ключей, адресов и записей позволяют практически избегать коллизий. Если коллизий нет, то трудоёмкость всех трёх операций составляет $O(1)$.

9.4.3. Алгоритм бинарного (двоичного) поиска

При использовании упорядоченного массива для представления самой важной оказывается операция поиска, поскольку чтобы удалить запись, её необходимо сначала найти, и при вставке записи необходимо найти место для этой записи.

В упорядоченном массиве операция поиска записи по ключу может быть выполнена за время $O(\log_2 n)$ (где n — количество записей) с помощью следующего алгоритма, известного как алгоритм *бинарного* (или *двоичного*) поиска.

Алгоритм 9.8. Бинарный поиск

Вход: ключ $a : key$, упорядоченный массив
$A :$ **array** $[1..n]$ **of struct** $\{ k : key; i : info \}$.
Выход: индекс записи с искомым ключом a в массиве A или 0, если записи с таким ключом нет.

```
b:=1  // начальный индекс части массива для поиска
e:=n  // конечный индекс части массива для поиска
while b ⩽ e do
  c:=(b+e)/2  // индекс проверяемого элемента (округленный до целого)
  if A[c].k > a then
    e:=c-1  // продолжаем поиск в первой половине
  else if A[c].k < a then
    b:=c+1  // продолжаем поиск во второй половине
  else
    return c  // нашли искомый ключ
  end if
end while
return 0  // искомого ключа нет в массиве
```

Обоснование. Достаточно заметить, что на каждом шаге основного цикла искомый элемент массива (если он есть) находится между (включительно) элементами с индексами b и e. Поскольку диапазон поиска на каждом шаге уменьшается вдвое, общая трудоёмкость не превосходит $\log_2 n$. □

ОТСТУПЛЕНИЕ

Упорядоченные списки проигрывают упорядоченным массивам при реализации операций ассоциативной памяти, поскольку для списков нет аналога алгоритма двоичного поиска. Это частное наблюдение является проявлением весьма общего факта, состоящего в том, что время доступа к элементу массива не зависит от номера элемента и от размера массива. Говорят, что массив является структурой данных *прямого доступа*, это характеристическое свойство массива. Во всех же «динамических» структурах данных, в частности, в списках, время доступа зависит от положения элемента и от размера структуры данных.

9.4.4. Алгоритм поиска в дереве сортировки

Обратимся теперь к деревьям сортировки (поиска).

ЗАМЕЧАНИЕ

Симметричный обход дерева поиска перечисляет множество узлов в порядке возрастания значения ключа, то есть определяет строгий линейный порядок на множестве узлов. Таким образом, любое конечное линейно упорядоченное множество можно представить в виде дерева поиска. Польза от такого представления в том, что это позволяет производить многие операции с множеством (в частности, поиск элемента) за время $O(h)$, где h — высота дерева. В предельном случае, когда $h = \log_2 n$, поиск с помощью дерева сортировки работает также быстро, как двоичный поиск.

Следующий алгоритм находит в дереве сортировки узел с указанным ключом, если он там есть.

Алгоритм 9.9. Поиск узла в дереве сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ $a : key$.
Выход: указатель p на найденный узел или **nil**, если в дереве нет такого ключа.

```
p:=T  // указатель на проверяемый узел
while p ≠ nil do
  if a < p.i then
    p:=p.l  // продолжаем поиск слева
  else if a > p.i then
    p:=p.r  // продолжаем поиск справа
  else
    return p  // нашли узел
  end if
end while
return nil  // искомого ключа нет в дереве
```

Обоснование. Этот алгоритм работает в точном соответствии с определением дерева сортировки: если текущий узел не искомый, то в зависимости от того, меньше или больше искомый ключ по сравнению с текущим, нужно продолжать поиск слева или справа соответственно. □

9.4.5. Алгоритм вставки в дерево сортировки

Один из возможных способов вставки в дерево сортировки узла с указанным ключом описан в алгоритме 9.10.

ЗАМЕЧАНИЕ

Необходимо отметить два важных обстоятельства. Во-первых, в дереве уже может быть узел с указанным ключом. Требуется определить, что надлежит делать в этом случае: выдавать сообщение об ошибке, менять старую запись на новую с тем же ключом и т. д. В данной реализации ничего не делается. Во-вторых, предлагаемый алгоритм хорошо работает, если дерево сортировки «динамичное», то есть в него постоянно добавляются и удаляются ключи, причем значения ключей случайны и распределены равномерно. Если же значения ключей меняются регулярным образом, например, если ключи всё время возрастают, то данный алгоритм применять не следует.

Алгоритм 9.10. Вставка узла в дерево сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ $a : key$.
Выход: модифицированное дерево сортировки T.

```
if T = nil then
  T := NewNode(a)  // первый узел в дереве
  return T
end if
p := T  // указатель на текущий узел
while true do
  if a = p.i then
    return T  // в дереве уже есть такой ключ
  end if
  if a < p.i then
    if p.l = nil then
      p.l := NewNode(a)  // создаём новый узел
      return T  // и подцепляем его к p слева
    else
      p := p.l  // продолжаем поиск места для вставки слева
    end if
  else
    if a > p.i then
      if p.r = nil then
        p.r := NewNode(a)  // создаём новый узел
        return T  // и подцепляем его к p справа
      else
        p := p.r  // продолжаем поиск места для вставки справа
      end if
    end if
  end if
end while
```

Обоснование. Алгоритм вставки, в сущности, аналогичен алгоритму поиска: в дереве ищется такой узел, имеющий свободную связь для подцепления нового узла, чтобы не нарушалось условие дерева сортировки. А именно, если новый ключ меньше текущего, то либо его можно подцепить слева (если левая связь свободна),

либо нужно найти слева подходящее место. Аналогично, если новый ключ больше текущего. □

ОТСТУПЛЕНИЕ

В последнее время в среде профессиональных программистов наблюдается противоестественное и достойное сожаления явление — *обфускация* программ, то есть искусственное составление текстов программ таким образом, чтобы сделать их как можно более непонятными для читателя. Обычно обфускация обосновывается необходимостью защиты интеллектуальной собственности, хотя на самом деле, по мнению автора, обфускация часто является проявлением комплекса неполноценности, развивающегося на фоне недостаточного уровня профессиональной подготовки. Для иллюстрации последнего тезиса приведём пример того, как *не надо* писать программы, проведя обфускацию алгоритма 9.10.

Вход: дерево сортировки, заданное указателем на корень $T : \uparrow N$,
где $N =$ **struct {** $i :$ key; $l :$ **array** $[0..1]$ **of** $\uparrow N$ **}**; ключ $a :$ key.
Выход: модифицированное дерево сортировки T.

```
if T = nil then
    T := NewNode(a)
    return T
end if
p := T
while true do
    if a = p.i then
        return T
    end if
    b := a > p.i
    if p.l[b] = nil then
        p.l[b] := NewNode(a)
        return T
    else
        p := p.l[b]
    end if
end while
```

Эта программа примерно вдвое короче, чуть медленнее (доступ к элементу массива обычно медленнее доступа к полю структуры), использует неявное приведение **false** $\mapsto 0$, **true** $\mapsto 1$, и её трудно понять без дополнительных пояснений.

9.4.6. Алгоритм удаления из дерева сортировки

Следующий алгоритм удаляет из дерева сортировки узел с указанным ключом. Если узла с указанным ключом нет в дереве, то ничего не делается. На рис. 9.14 приведена иллюстрация к этому алгоритму.

Обоснование. Удаление узла производится перестройкой дерева сортировки. При этом возможны три случая (не считая тривиального случая, когда удаляемого узла нет в дереве, и ничего делать не нужно).

[1] Правая связь удаляемого узла p пуста (рис. 9.14, *слева*). В этом случае левое поддерево 1 узла p подцепляется к родительскому узлу q с той же стороны, с которой был подцеплен узел p. Условие дерева сортировки, очевидно, выполняется.

[2] Правая связь удаляемого узла p не пуста и ведёт в узел u, левая связь которого пуста (рис. 9.14, *в центре*). В этом случае левое поддерево 1 узла p подцепляется к узлу u слева, а сам узел u подцепляется к родительскому узлу q с той же стороны, с которой был подцеплен узел p. Нетрудно проверить, что условие дерева сортировки выполняется и в этом случае.

Алгоритм 9.11. Удаление узла из дерева сортировки

Вход: дерево сортировки T, заданное указателем на корень; ключ $a: key$.
Выход: модифицированное дерево сортировки T.

```
Find(T, a, p, q, s)  // поиск удаляемого узла
if p = nil then
  return T  // нет такого узла— ничего делать не нужно
end if
if p.r = nil then
  Delete(p, q, p.l, s)  // случай 1, рис. 9.14, слева
else
  u := p.r
  if u.l = nil then
    u.l := p.l
    Delete(p, q, u, s)  // случай 2, рис. 9.14, в центре
  else
    w := u; v := u.l
    while v.l ≠ nil do
      w := v; v := v.l
    end while
    p.i := v.i
    Delete(v, w, v.r, −1)  // случай 3, рис. 9.14, справа
  end if
end if
return T
```

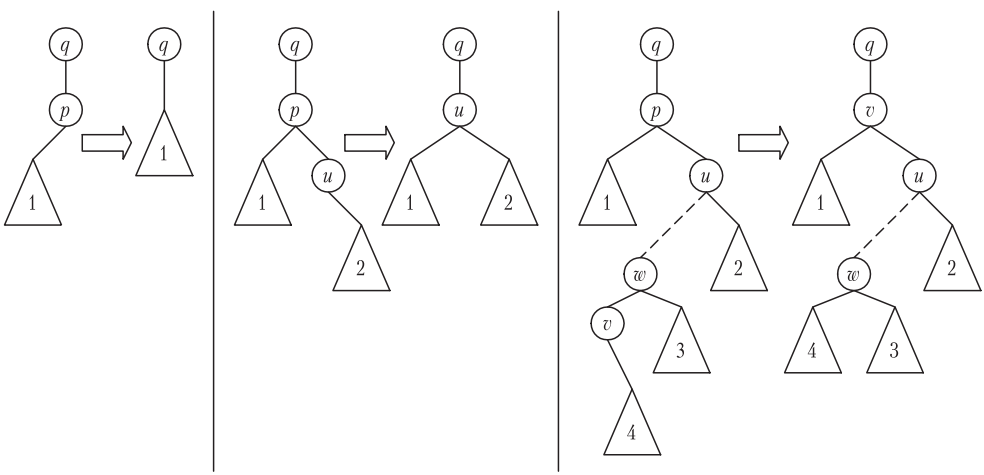


Рис. 9.14. Иллюстрация к алгоритму удаления узла из дерева сортировки

[3] Правая связь удаляемого узла p не пуста и ведёт в узел u, левая связь которого не пуста. Поскольку дерево сортировки конечно, можно спуститься от узла u до узла v, левая связь которого пуста (рис. 9.14, *справа*). В этом случае выполняются два преобразования дерева. Сначала информация в узле p заменяется информацией узла v. Поскольку узел v находится в правом поддереве узла p и в левом поддереве

узла u, имеем $p.i < v.i < u.i$. Таким образом, после этого преобразования условие дерева сортировки выполняется. Далее правое поддерево 4 узла v подцепляется слева к узлу w, а сам узел v удаляется. Поскольку поддерево 4 входило в левое поддерево узла w, условие дерева сортировки также сохраняется. □

ЗАМЕЧАНИЕ

В замечательной книге [10], из которой заимствован данный алгоритм, показано, что, хотя алгоритм «выглядит несимметричным» (правые и левые связи обрабатываются по-разному), на самом деле в среднем характеристики дерева сортировки не искажаются.

9.4.7. Вспомогательные алгоритмы для дерева сортировки

Алгоритмы трех предыдущих параграфов используют вспомогательные функции, описанные здесь.

1. Поиск узла — функция Find.

Вход: дерево сортировки T, заданное указателем на корень; ключ $a : key$.

Выход: p — указатель на найденный узел или **nil**, если в дереве нет такого ключа; q — указатель на отца узла p; s — способ подцепления узла q к узлу p ($s = -1$, если p слева от q; $s = +1$, если p справа от q; $s = 0$, если p — корень).

```
p:=T; q:=nil; s:=0  // инициализация
while p ≠ nil do
  if p.i = a then
    return p, q, s
  end if
  q:=p  // сохранение значения p
  if a < p.i then
    p:=p.l; s:=−1  // поиск слева
  else
    p:=p.r; s:=+1  // поиск справа
  end if
end while
return p, q, s
```

ОТСТУПЛЕНИЕ

В этой простой функции стóит обратить внимание на использование указателя q, который отслеживает значение указателя p «с запаздыванием», то есть указатель q «помнит» предыдущее значение указателя p. Такой приём полезен при обходе однонаправленных структур данных, в которых невозможно вернуться назад.

2. Удаление узла — процедура Delete.

Вход: $p1$ — указатель на удаляемый узел; $p2$ — указатель на подцепляющий узел; $p3$ — указатель на подцепляемый узел; s — способ подцепления.

Выход: преобразованное дерево.

```
if s = −1 then
  p2.l:=p3  // подцепляем слева
end if
if s = +1 then
  p2.r:=p3  // подцепляем справа
end if
```

dispose($p1$) // удаляем узел

3. Создание нового узла — конструктор NewNode.

Вход: ключ a.

Выход: указатель p на созданный узел.

new(p); $p.i := a$; $p.l :=$ **nil**; $p.r :=$ **nil**

return p

9.4.8. Сравнение представлений ассоциативной памяти

Пусть n — количество элементов в ассоциативной памяти. Рассмотрим следующие представления:

А) неупорядоченный массив;

Б) упорядоченный массив;

В) таблица расстановки;

Г) неупорядоченный список;

Д) упорядоченный список;

Е) дерево сортировки.

Оценки трудоёмкости операций для различных представлений указаны в таблице.

	А	Б	В	Г	Д	Е
Добавить	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(h)$
Найти	$O(n)$	$O(\log_2 n)$	$O(1)$	$O(n)$	$O(n)$	$O(h)$
Удалить	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(h)$

ЗАМЕЧАНИЕ

Необходимо сделать два существенных уточнения. Во-первых, для таблиц расстановки указаны условные оценки в среднем, которые выполняются только при определённых ограничениях. В худшем случае оценки для таблиц расстановки безнадёжно плохие. Во-вторых, трудоёмкость операций с деревом сортировки ограничена сверху высотой дерева, которая может меняться в пределах от $\log_2 n$ до n. Дерево сортировки может расти неравномерно. Например, если при загрузке дерева исходные данные уже упорядочены, то полученное дерево будет право- или леволинейным и окажется даже менее эффективным, чем неупорядоченный массив.

9.5. Специальные деревья

Высота дерева сортировки является критическим параметром эффективности. В этом разделе обсуждаются методы уменьшения высоты дерева сортировки, а также использование других структур данных, родственных бинарным деревьям, для реализации функций ассоциативной памяти.

9.5.1. Выровненные и полные деревья

Бинарное дерево называется *выровненным*, если все листья находятся на одном (последнем) уровне. В выровненном дереве все ветви имеют одну длину, равную высоте дерева, однако выровненное дерево не всегда является эффективным деревом сортировки.

Пример. Дерево, состоящее из корня и двух поддеревьев, леволинейного и праволинейного, ведущих к двум листьям, является выровненным, однако такое дерево имеет высоту $(p-1)/2$.

Бинарное дерево называется *заполненным*, если все узлы, степень которых меньше 2, располагаются на одном или двух последних уровнях. Другими словами, в заполненном дереве T все ярусы $D(r,i)$, кроме, может быть, последнего, заполнены: $\forall\, i \in 0..(h(T)-1)\ \ \left(|D(r,i)| = 2^i\right)$.

Пример. На рис. 9.15 приведены диаграммы заполненного (*слева*) и незаполненного (*справа*) деревьев.

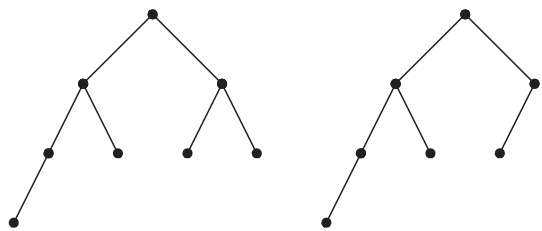


Рис. 9.15. Заполненное и незаполненное деревья

Заполненное дерево имеет наименьшую возможную для данного p высоту h.

ТЕОРЕМА. *Для заполненного бинарного дерева* $\log_2(p+1) - 1 \leqslant h < \log_2(p+1)$.

Доказательство. На i-м уровне может быть самое большее 2^i вершин, следовательно, $\sum_{i=0}^{h-1} 2^i < p \leqslant \sum_{i=0}^{h} 2^i$. Значит, $2^h < p+1 \leqslant 2^{h+1}$, и, логарифмируя, имеем $h < \log_2(p+1)$ и $h+1 \geqslant \log_2(p+1)$. Следовательно, $\log_2(p+1)-1 \leqslant h < \log_2(p+1)$. □

Выровненное заполненное бинарное дерево называется *полным*. В полном бинарном дереве все листья находятся на последнем уровне, и все узлы, кроме листьев, имеют полустепень исхода 2. Другими словами, в полном дереве *все* ярусы заполнены.

СЛЕДСТВИЕ. *Полное бинарное дерево высотой* h *имеет* $2^{h+1} - 1$ *узлов.*

ЗАМЕЧАНИЕ

Иногда полным называют бинарное дерево, в котором все нелистовые узлы имеют полустепень исхода 2, но листья могут встречаться на любом уровне. Высота дерева, полного в таком смысле, может изменяться в широких пределах.

Заполненные деревья дают наибольший возможный эффект при поиске. Однако известно, что вставка/удаление в заполненное дерево может потребовать полной перестройки всего дерева и, таким образом, трудоёмкость операции в худшем случае составит $O(p)$.

Желательно выделить такой подкласс деревьев сортировки, в котором высота ограничена, и в то же время операции по вставке и удалению элементов выполняются эффективно. Было найдено несколько таких подклассов, наиболее популярный из них описывается в следующем параграфе.

9.5.2. Сбалансированные деревья

(Бинарное) дерево называется *подровненным деревом*, или *АВЛ-деревом* (Адельсон-Вельский и Ландис, 1962), или *сбалансированным по высоте деревом*, если для любого узла высоты левого и правого поддеревьев различаются не более чем на единицу.

Пример. На рис. 9.16 приведена диаграмма максимально несимметричного сбалансированного по высоте дерева, в котором для всех узлов высота левого поддерева ровно на 1 больше высоты правого поддерева.

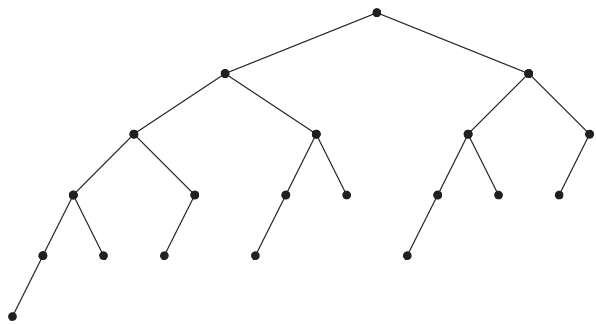


Рис. 9.16. Сбалансированное дерево

ЗАМЕЧАНИЕ

Кроме сбалансированных по высоте, рассматривают и другие классы деревьев, например *сбалансированные по весу* деревья, в которых для каждого узла количества узлов в левом и правом поддеревьях находятся в определённом соотношении. Здесь такие классы не рассматриваются, поэтому далее термин «сбалансированное дерево» означает сбалансированное по высоте бинарное дерево сортировки.

ТЕОРЕМА. *Для сбалансированного по высоте бинарного дерева* $h \leqslant 2\log_2 p$.

Доказательство. Рассмотрим наиболее несимметричные сбалансированные деревья, скажем, такие, у которых всякое левое поддерево на 1 выше правого. Такие сбалансированные деревья имеют максимальную возможную высоту среди всех сбалансированных деревьев с заданным числом вершин. Пусть P_h — число вершин в наиболее несимметричном сбалансированном дереве высоты h. По построению имеем $p = P_h = P_{h-1} + P_{h-2} + 1$. Непосредственно проверяется, что $P_0 = 1$, $P_1 = 2$, $P_2 = 4$. Покажем по индукции, что $P_h \geqslant (\sqrt{2})^h$. База: $P_0 = 1$, $(\sqrt{2})^0 = 1$, $1 \geqslant 1$; $P_1 = 2$, $(\sqrt{2})^1 = \sqrt{2}$, $2 \geqslant \sqrt{2}$. Пусть $P_h \geqslant (\sqrt{2})^h$. Тогда

$$P_{h+1} = P_h + P_{h-1} + 1 \geqslant (\sqrt{2})^h + (\sqrt{2})^{h-1} + 1 =$$
$$= (\sqrt{2})^h \left(1 + \frac{1}{(\sqrt{2})} + \frac{1}{(\sqrt{2})^h}\right) > (\sqrt{2})^h \left(1 + \frac{1}{(\sqrt{2})}\right) > (\sqrt{2})^h \sqrt{2} = (\sqrt{2})^{h+1}.$$

Имеем $(\sqrt{2})^h = 2^{h/2} \leqslant P_h = p$, следовательно, $\frac{h}{2} \leqslant \log_2 p$ и $h \leqslant 2\log_2 p$. □

Можно заметить, что рекуррентное соотношение для P_h похоже на определение чисел Фибоначчи $F(n)$ (п. 5.7.3), откуда легко выводимо следствие.

СЛЕДСТВИЕ. $P_h = F(h+2) - 1$.

Доказательство. По индукции. База: $P_0 = 1$, $F(0) = 1$, $F(1) = 1$, $F(2) = 2$, $1 = 2 - 1$. Пусть $P_h = F(h+2) - 1$. Рассмотрим P_{h+1}. Имеем $P_{h+1} = P_h + P_{h-1} = F(h+2) - 1 + F(h-1+2) - 1 + 1 = F(h+2) + F(h+1) - 1 = F(h+3) - 1 = F((h+1)+2) - 1$. □

ЗАМЕЧАНИЕ

Известна более точная оценка высоты сбалансированного дерева: $h < \log_{(\sqrt{5}+1)/2} 2 \log_2 p$.

Сбалансированные деревья уступают заполненным деревьям по скорости поиска (менее чем в два раза), однако их преимущество состоит в том, что известны алгоритмы вставки узлов в сбалансированное дерево и удаления их из него, которые сохраняют сбалансированность и в то же время при перестройке дерева затрагивают только конечное число узлов (см. следующий параграф). Поэтому во многих случаях сбалансированное дерево оказывается наилучшим вариантом представления дерева сортировки.

9.5.3. Балансировка деревьев

При добавлении (или при удалении) узла в сбалансированном дереве сортировки может возникнуть ситуация, в которой баланс нарушается. В этом случае необходимо перестроить дерево, чтобы восстановить баланс. Алгоритм балансировки дерева представлен на рисунках, при этом использованы следующие обозначения: x — добавляемый узел, v — первый узел, в котором нарушен баланс на пути от корня r к новому узлу x. Тогда возможны два варианта. Путь от v к x состоит либо из дуг одной ориентации (скажем, только из левых дуг), либо из левых и правых дуг. Варианты, в которых путь состоит только из правых дуг либо из правых и левых дуг, зеркально симметричны. В первом варианте дерево перестраивается в соответствии с *правилом простого вращения*, представленным на рис. 9.17.

Простое вращение восстанавливает баланс и сохраняет условие дерева сортировки. Действительно, до преобразования имеем $u < v, p < v, q < v, p < u, q > u, w > v$. Следовательно, $p < u, u < v, u < q, u < w, q < v, w > v$ и преобразование не нарушает условия дерева сортировки.

Во втором варианте дерево перестраивается в соответствии с *правилом двойного вращения* (рис. 9.18). Двойное вращение восстанавливает баланс и сохраняет условие дерева сортировки. Действительно, до преобразования имеем $u < v$, $w > v$, $p > v$, $q > v$, $p < w$, $q > w$, $s > v$, $t > v$, $s < w$, $t < w$, $s < p$, $t > p$. Следовательно, $v < p$, $u < p$, $s < p$, $u < v$, $s > v$, $w > p$, $t > p$, $q > p$, $t < w$, $q > w$ и преобразование не нарушает условия дерева сортировки.

ЗАМЕЧАНИЕ

Детальное обсуждение алгоритмов работы с АВЛ-деревьями, включая оценки трудоёмкости в среднем и в худшем случае, можно найти в [3] и [12].

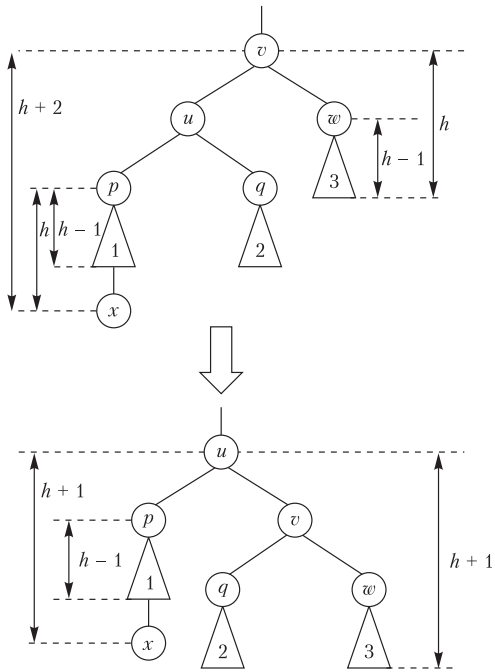


Рис. 9.17. Простое вращение

9.5.4. Двоичные кучи

Довольно часто в приложениях оказывается востребованной модификация ассоциативной памяти, в которой записи добавляются в произвольном порядке, а ищется и извлекается всегда только запись с максимальным ключом из хранящихся в данный момент в памяти.

Пример. Выбор для обслуживания процесса с наивысшим приоритетом.

ЗАМЕЧАНИЕ

Случай минимального ключа совершенно аналогичен.

Такая ассоциативная память предполагает наличие следующих базовых операций:

1) добавить (ключ, запись);
2) извлечь по максимальному ключу (): запись;
3) изменить (ключ старый, ключ новый, запись).

ЗАМЕЧАНИЕ

В некоторых случаях рассматривают несколько иной набор операций. Вторую операцию делят на две: только найти и просмотреть запись с максимальным ключом и удалить запись с максимальным ключом. Также иногда не используют третью операцию или не разрешают задавать новое значение записи при изменении ключа. Такие изменённые операции легко выражаются через базовые.

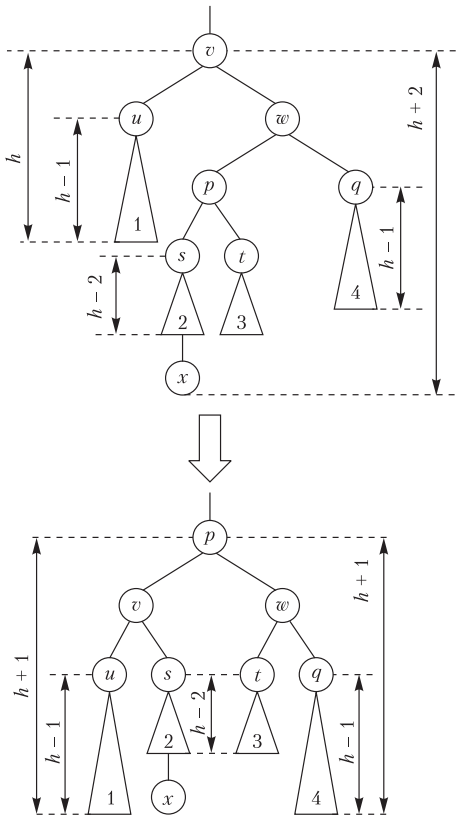


Рис. 9.18. Двойное вращение

Двоичная куча — это нагруженное бинарное дерево со следующими свойствами:

1) значение в любом узле не меньше, чем значения в узлах поддеревьев;
2) глубина листьев (расстояние до корня) отличается не более чем на 1;
3) последний уровень заполняется слева направо.

Удобная структура данных для двоичной кучи — представление бинарного дерева массивом A, у которого первый элемент $A[1]$ — значение в корне, а потомками элемента $A[i]$ являются элементы $A[2i]$ и $A[2i + 1]$. При таком способе хранения свойства двоичной кучи 2 и 3 выполнены автоматически.

Пример. На рис. 9.19 представлена двоичная куча, которая хранится в виде массива

10	8	5	7	3	1

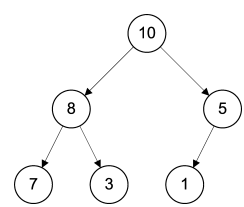


Рис. 9.19. Двоичная куча

Двоичная куча интересна тем, что при её использовании в качестве представления ассоциативной памяти реализация операций оказывается весьма эффективной.

9.5.5. Восстановление свойства двоичной кучи

Если в куче изменяется один из элементов, то она может перестать удовлетворять свойству 1 двоичной кучи. Для восстановления этого свойства служит процедура Heapify. Эта процедура принимает на вход массив элементов A и индекс i. Она восстанавливает свойство кучи во всём поддереве, корнем которого является элемент $A[i]$ в случае, когда поддеревья обладают свойством кучи.

Алгоритм 9.12. Процедура Heapify восстановления свойства кучи

Вход: массив A, индекс i.
Выход: восстанавливается свойство кучи в поддереве с корнем $A[i]$, причём левое и правое поддеревья удовлетворяют свойствам кучи.

```
l := 2 * i; r := 2 * i + 1; m := i   // инициализация
if l ⩽ |A| & A[l] > A[i] then m := l end if  // нарушение слева
if r ⩽ |A| & A[r] > A[i] then m := r end if  // нарушение справа
if i ≠ m then
   A[i] ↔ A[m]  // транспозиция элементов массива
   Heapify (A, m)
end if
```

Обоснование. В случае, если корень поддерева меньше хотя бы одного из сыновей, нужно поменять местами значения корня с максимальным значением сыновей. Тогда достаточно спуска по дереву с рекурсивным вызовом функции Heapify для гарантирования восстановления свойств двоичной кучи. Время работы процедуры прямо пропорционально высоте дерева, то есть $O(\log n)$. □

Пример. На рис. 9.20 слева выделен элемент, в котором нарушено условие кучи, а правее показан ход выполнения процедуры Heapify.

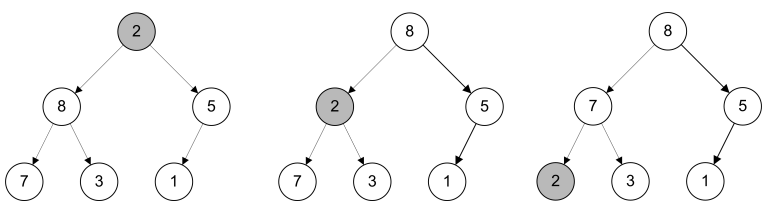


Рис. 9.20. Восстановление свойства кучи

Лемма. *Если выполнить* Heapify *для всех элементов массива A, начиная с последнего и кончая первым, то он станет кучей.*

Доказательство. Заметим, что Heapify(A, i) не делает ничего, если $i > \lfloor |A|/2 \rfloor$, так как узлы с такими индексами являются листьями и изначально имеют свойства двоичной кучи. Далее по индукции, к моменту выполнения Heapify(A, i) все поддеревья, чьи корни имеют индекс больше i, уже кучи, и, следовательно, после выполнения Heapify(A, i) кучей будет вся часть массива A, начиная с i. □

Алгоритм 9.13. Процедура MakeHeap построения кучи

Вход: массив A
Выход: устанавливается свойство кучи в дереве, заданным массивом A
for i **from** $\lfloor |A|/2 \rfloor$ **downto** 1 **do** Heapify(A, i) **end for**

Время работы равно $O(n \log n)$, так как функция Heapify(A, i) вызывается $O(n)$ раз.

Пример. В таблице показано изменение массива A при построении кучи.

i	1	3	5	7	8	10
3	1	3	10	7	8	5
2	1	8	10	7	3	5
1	10	8	5	7	3	1

9.5.6. Реализация операций двоичной кучи

Если при изменения значения элемента с индексом i новый ключ k меньше, то достаточно вызвать процедуру Heapify, разобранную в п. 9.5.5. При изменении значения элемента $A[i]$ на большее также могут нарушиться свойства кучи. В этом случае необходимо выполнить следующий алгоритм IncElt(A, i, k).

Алгоритм 9.14. Процедура IncElt увеличения элемента в куче

Вход: массив A, индекс изменяемого элемента i, новое значение k.
Выход: значение элемента с индексом i изменяется на k без нарушения свойств двоичной кучи.
if $k < A[i]$ **then**
  return fail // новое значение должно быть не меньше предыдущего
end if
$A[i] := k$
while $i > 1$ & $A[\lfloor i/2 \rfloor] < A[i]$ **do**
  $A[i] \leftrightarrow A[\lfloor i/2 \rfloor]$ // транспозиция элементов
  $i := \lfloor i/2 \rfloor$
end while

Время работы алгоритма IncElt составляет $O(\log n)$, так как количество транспозиций прямо пропорционально высоте дерева.

Добавить элемент в двоичную кучу можно так: в конец массива добавляется новый элемент с заведомо минимальным значением, затем его значение увеличивается с помощью только что описанной процедуры. Время работы получается такое же — $O(\log n)$.

Максимальный элемент двоичной кучи находится в корне, то есть в первом элементе массива, поэтому его поиск не является проблемой. Нужно только правильно перестроить кучу. Для этого достаточно поменять местами значения в корне и в последней вершине дерева, после чего удалить последнюю вершину. Для правильного перестроения кучи нужно «протащить» новое значение в корне вниз, что и делает процедура Heapify.

Время работы прямо пропорционально высоте дерева — $O(\log n)$.

Алгоритм 9.15. Извлечение максимального элемента из кучи

Вход: массив A

Выход: извлечение максимального элемента из кучи с сохранением свойств двоичной кучи.

if $|A| = 0$ **then return fail end if** // куча пуста
$m := A[1]$ // сохранение значения корневого элемента
$A[1] := A[|A|]$ // копирование последнего элемента в корень дерева
$|A| := |A| - 1$ // уменьшение длины массива
Heapify(A, 1)
return m

9.5.7. Построение графа по степенному ряду

В качестве примера использования двоичной кучи рассмотрим задачу построения графа по заданным степеням вершин, если такой граф существует. В п. 7.2.7 показано, что невозрастающая последовательность $D = (d_1, \ldots, d_p)$ является степенной последовательностью графа тогда и только тогда, когда последовательность $D' = (d'_1, \ldots, d'_{p-1})$, полученная откладыванием последнего элемента d_p, является степенной. Эта теорема определяет конструктивный способ ответа на вопрос, является ли данная поледовательность степенной.

Для решения этой задачи подходит структура данных *очередь с приоритетом*, реализуемая на основе двоичной кучи.

ОТСТУПЛЕНИЕ

Приоритет — это ключ, который ассоциируется с записью. Понятие «приоритета» прижилось, так как изначально эта структура данных была придумана для распределения вычислительных ресурсов в многозадачных операционных системах.

Очередь с приоритетом позволяет хранить пары (ключ, значение) и поддерживает две операции:

1) добавить (ключ, значение);
2) извлечь запись с максимальным ключом : (ключ, значение).

Идея реализации очереди с приоритетом состоит в построении двоичной кучи, ключами в которой являются приоритеты. Как хранить значения — не имеет принципиального значения, можно считать, что элементами кучи являются пары (ключ, значение).

Добавление ключа k со значением v в очередь Q записывается $(k, v) \to Q$, а извлечение, соответственно, $(k, v) \leftarrow Q$.

В таком случае граф по степенной последовательности строится с помощью очереди с приоритетом следующим алгоритмом. В этой очереди ключом является степень вершины, а значением — её номер.

ЗАМЕЧАНИЕ

Для заданной последовательности чисел граф существует не всегда. Если вершину наибольшей степени не удаётся *отложить*, то граф построить невозможно.

Алгоритм 9.16. Построение графа по степенному ряду

Вход: A — массив степеней вершин.
Выход: граф с заданными степенями, если такой существует, представленный массивом рёбер E.

```
Q := ∅  // инициализация очереди с приоритетом
 for i from 1 to |A| do (A[i], i) → Q end for
// номера вершин— индексы в начальном массиве
E := ∅  // массив рёбер
while |Q| > 0 do
  (d, n) ← Q  // извлечение из очереди откладываемой вершины
   if d > |Q| then return fail end if  // невозможно отложить вершину
  S := ∅  // инициализация временного контейнера
  for k from 1 to d do
    (d', n') ← Q  // извлечение вершины, смежной с откладываемой
    (n, n') → E  // добавление ребра в граф
     if d' > 1 then (d' − 1, n') → S end if  // убрать в контейнер
  end for
  for (d', n') ∈ S do S(d', n') → Q end for  // достать из контейнера
end while
return E
```

9.5.8. Дерево отрезков

Пусть задано конечное линейно упорядоченное множество (то есть массив) некоторых элементов, и требуется производить различные *групповые* операции над *отрезками* этого множества (то есть операции с *отрезками массива*). Например: найти минимум, максимум или сумму элементов отрезка массива; присвоить заданную величину всем элементам отрезка или увеличить все элементы отрезка на заданную величину и т. д.

ЗАМЕЧАНИЕ

В языке Algol-68 часть массива называется *вырезкой из массива*, но словосочетание «дерево вырезок» неблагозвучно, поэтому здесь используется термин *отрезок массива*, означающий часть массива, определяемую двумя индексами — начало отрезка и конец отрезка.

Если требуется только один раз выполнить операцию над одним отрезком, то дополнительные ухищрения ничего не дадут, и разумнее всего просто выполнить требуемую операцию в цикле. Однако если операцию требуется выполнять много раз над различными отрезками, то можно провести *предобработку* заданного массива, построив дополнительную структуру данных, которая обеспечит более эффективное выполнение операций.

Для реализации многократно повторяемых групповых операций на линейно упорядоченном множестве используется структура данных, которая называется *дерево отрезков*.

Пусть задан массив A : **array** $[1..n]$ **of** M, причём M является моноидом (п. 2.2.3) с ассоциативной операцией $\varphi\colon M \times M \to M$ и нейтральным элементом e. Также на множестве M задана произвольная функция преобразования $f\colon M \to M$. Дерево отрезков позволяет выполнять, по крайней мере, три основные операции:

1) построить дерево отрезков (массив A);

2) изменить в массиве элементы (индекс l, индекс r, функция f);
3) применить операцию (индекс l, индекс r) : значение $\varphi(A[l..r])$.

Дерево отрезков позволяет реализовать *любую* ассоциативную операцию над отрезком, то есть групповую операцию, результат которой не зависит от порядка применения операции к индивидуальным элементам отрезка, и позволяет использовать *любую* функцию независимого преобразования элементов отрезка. Не ограничивая общности, здесь в примерах в качестве операции рассматривается суммирование элементов, то есть $\varphi(A[l..r]) := \sum_{i=l}^{r} A[i]$, а в качестве функции преобразования рассматривается присваивание конкретного значения x, то есть $\forall\, i \in 1..n\ \ (A[i] := x)$.

Дерево отрезков является нагруженным бинарным деревом, в котором во всех листовых узлах хранятся элементы массива A (или нейтральные элементы, если нужно), а все нелистовые узлы хранят результат применения операции φ к своим сыновьям.

Представление дерева отрезков — массив T : **array** $[1..(2n-1)]$ **of** M, в котором потомками элемента $T[i]$ являются элементы $T[2i]$ и $T[2i+1]$, причём $T[i] = \varphi(T[2i], T[2i+1])$. Если $T[i] = \varphi(A[l..r])$ — значение операции на отрезке $A[l..r]$, то $T[2i] = \varphi(A[l..(l+r)\ \mathbf{div}\ 2])$ — значение операции на левой «половине» отрезка, а $T[2i+1] = \varphi(A[(l+r)\ \mathbf{div}\ 2+1..r])$ — значение операции на правой «половине» отрезка. Значением операции на отдельном элементе (когда $l = r$) является сам этот элемент $A[l]$.

Построение дерева отрезков производится следующим образом: $T[1]$ — значение операции на отрезке $A[1..n]$. Для вычисления значения $T[1]$ вычисляются значения в потомках корня, то есть $T[2]$ и $T[3]$, где $T[2]$ — значение операции на отрезке $A[1..n\ \mathbf{div}\ 2]$, а $T[3]$ — на отрезке $A[n\ \mathbf{div}\ 2+1..n]$. Далее отрезки делятся рекурсивно. Окончание рекурсии: $T[i]$ — значение операции на отрезке $A[j..j] = A[j]$. В результате добавленные элементы находятся в массиве T слева, а исходные элементы массива A — справа.

Пример. Пусть исходный массив $A = [3, 12, 6, 8, 4, 3, 5]$. Тогда построенное дерево (для операции сложения) имеет вид $T = [41, 29, 12, 15, 14, 7, 5, 3, 12, 6, 8, 4, 3, 5, 0]$, рис. 9.21, исходные элементы закрашены, добавленные элементы не закрашены.

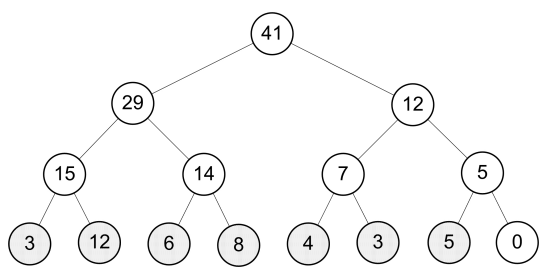


Рис. 9.21. Дерево отрезков

ЗАМЕЧАНИЕ

Нелистовые элементы с суммами находятся в массиве T левее элементов исходного массива A. Дополнительные элементы с нулями (нейтральными элементами моноида) находятся

в массиве T правее элементов исходного массива A. Вообще говоря, они излишни для вычислений и вставлены только для соблюдения формального правила, что каждый нелистовой узел имеет ровно двух сыновей.

Построение дерева отрезков можно выполнить с помощью рекурсивной процедуры Build. При этом массив $A[1..n]$ считается глобальным, и тогда построение массива T выполняется вызовом $\mathrm{Build}(1, 1, n)$.

Алгоритм 9.17. Построение дерева отрезков

Вход: номер узла v, границы l и r отрезка, соответствующего узлу.
Выход: построенное дерево T.

```
if l = r then
  T[v] := A[l]   // отрезок является листом
else
  m := (l + r) div 2   // середина отрезка
  Build(A, 2v, l, m)   // левое поддерево
  Build(A, 2v + 1, m + 1, r)   // правое поддерево
  T[v] := φ(T[2v], T[2v + 1])   // выполняем операцию в данном узле
end if
```

Обоснование. Построенный массив T действительно является представлением дерева отрезков, поскольку порядок элементов в этом массиве в точности совпадает с обходом бинарного дерева по уровням, а узлы нагружены нужными значениями по построению. □

Если дерево отрезков T для данного массива A и для заданной операции φ построено (и считается глобальным в следующих процедурах), то вычислить $\varphi(A[l..r])$, где $1 \leqslant l \leqslant r \leqslant n$, можно алгоритмом 9.18.

Алгоритм 9.18. Вычисление значения функции на отрезке

Вход: границы l и r запроса.
Выход: значение операции на отрезке $\varphi(A[l..r])$.

```
return Φ(1, 1, n, l, r)
```

Рекурсивная процедура Φ вычисления суммы на отрезке — алгоритм 9.19.

Алгоритм 9.19. Рекурсивная процедура Φ вычисления значения функции на отрезке

Вход: индекс узла v, границы p и q отрезка, соответствующего узлу v, границы l и r запроса на вычисление значения $\varphi(A[l..r])$.
Выход: сумма на отрезке $[l..r]$

```
if l > r then return e end if   // пустой запрос
if l = p & r = q return T[v] end if   // знаем ответ
m := (p + q) div 2   // середина отрезка
g := Φ(2v, p, m, l, min(r, m))   // в левом поддереве
h := Φ(2v + 1, m + 1, q, max(l, m + 1), r)   // в правом поддереве
return φ(g, h)   // ответ
```

Обоснование. При первом вызове функции Ф имеем $p \leqslant l \leqslant r \leqslant q$, поскольку $p = 1 \leqslant l \leqslant r \leqslant n = q$. Далее либо границы запроса совпали с границей ответственности узла, и тогда можно сразу вернуть ответ, либо нет, и тогда возможны три случая.

[$p \leqslant l \leqslant r \leqslant m$] В этом случае ответ надо искать в левом поддереве, и для него выполнено требуемое предусловие, поскольку $p \leqslant l \leqslant \min(r, m) \leqslant m$, а в правом поддереве запрос заведомо вернёт нейтральный элемент, поскольку $\max(l, m+1) > r$.

[$m+1 \leqslant l \leqslant r \leqslant q$] В этом случае ответ надо искать в правом поддереве, и для него выполнено требуемое предусловие, поскольку $m+1 \leqslant \max(l, m+1) \leqslant r \leqslant q$, а в левом поддереве запрос заведомо вернёт нейтральный элемент, поскольку $l > \min(r, m)$.

[$p \leqslant l \leqslant m \leqslant r \leqslant q$] В этом случае ответ надо искать в обоих поддеревьях, и для них выполнены требуемые предусловия, поскольку $p \leqslant l \leqslant \min(r, m) \leqslant m$ и $m + 1 \leqslant \max(l, m + 1) \leqslant r \leqslant q$. □

Изменение значения элемента в массиве и перестраивание дерева отрезков.

Алгоритм 9.20. Изменение значения в дереве отрезков

Вход: i — индекс изменяемого элемента в исходном массиве A, f — функция изменения значения.
Выход: измененное дерево T
Modify$(1, 1, n, i, f)$

Алгоритм 9.21. Рекурсивная процедура Modify изменения значения элемента

Вход: индекс узла v, границы p и q отрезка, соответствующего узлу v, индекс i элемента в массиве A, f — функция изменения значения.
Выход: измененное дерево T.

```
if p = q then T[v] := f(); return end if   // отрезок является листом
m := (p + q) div 2   // середина отрезка
if p ⩽ m then Modify(2v, p, m, i, f) else Modify(2v+1, m+1, q, i, f) end if
T[v] = φ(T[2v], T[2v + 1])   // пересчитываем узел v
```

Обоснование. Необходимо рекурсивно спуститься по дереву отрезков до листа, изменить его, и, возвращаясь из рекурсии, пересчитать всех предков. □

9.5.9. Оценки эффективности дерева отрезков

ЗАМЕЧАНИЕ

Операция φ не может являться параметром запроса, поскольку узлы дерева хранят конкретные значения, и при изменении операции все значения необходимо пересчитать. Функция преобразования f, напротив, может быть параметром запроса. Более того, эта функция может не зависеть от параметров и быть константой, может зависеть от прежнего значения элемента или может зависеть от каких-то внешних параметров. Таким образом, дерево отрезков можно эффективно использовать и в том случае, когда массив A динамически меняет свои элементы (но не длину).

ТЕОРЕМА 1. *Количество узлов в дереве отрезков не превосходит* $2n - 1$.

Доказательство. Если $n = 2^k$, то дерево отрезков является полным и количество узлов в точности равно $2n - 1$, поскольку крона полного дерева содержит на 1 узел больше, чем всё остальное дерево. В неполном дереве отрезков узлов разве что меньше. □

ТЕОРЕМА 2. *Время работы процедуры построения дерева отрезков $O(n)$.*

Доказательство. Процедура построения дерева отрезков посещает каждый узел один раз, а количество узлов не более $2n - 1$. □

ТЕОРЕМА 3. *Процедура запроса на каждом уровне дерева отрезков посещает не более четырёх узлов.*

Доказательство. Для удобства введём обозначение запроса следующим образом: (l, r, l', r'), где l, r — границы запроса, а l', r' — границы отрезка массива для данного узла. Для краткости записи знак / здесь означает деление с остатком. Проведём разбор случаев.

[$l = l'$ & $r = r'$] Запрос такого типа по алгоритму отработает мгновенно.

[$r' \leqslant (l+r)/2 \vee l' > (l+r)/2$] В этом случае произойдет всего один рекурсивный вызов: либо от левого поддерева, либо от правого.

[$r' > (l+r)/2$ & $l' \leqslant (l+r)/2$] В этом случае запрос разделится на два запроса: $(l, (l+r)/2, l', (l+r)/2)$ и $((l+r)/2+1, r, (l+r)/2+1, r')$. Теперь, если $l' > (3l+r)/4$, либо $r' \leqslant (3l+r)/4$, то имеем случай 2. В противном случае, если $l' \leqslant (3l+r)/4$, то левый запрос разделится на два запроса: $(l, (3l+r)/4, l', (3l+4)/4)$ и $((3l+r)/4+1, (l+r)/2, (3l+r)/4+1, (l+r)/2)$. Видно, что второй из них соответствует пункту 1 и отработает мгновенно. Аналогично для правого запроса. Таким образом, в любой момент времени запрос посещает не более четырёх вершин на каждом уровне и имеется не более двух реально работающих ветвей рекурсии. □

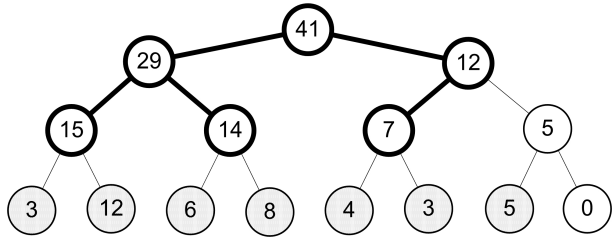
ТЕОРЕМА 4. *Время работы процедуры запроса $O(\log n)$.*

Доказательство. Процедура запроса посещает не более четырёх вершин на каждом уровне, а количество уровней в дереве отрезков $O(\log n)$. □

ТЕОРЕМА 5. *Время работы процедуры изменения элемента в дереве отрезков $O(\log n)$.*

Доказательство. Элемент $A[i]$ участвует в $O(\log n)$ узлах дерева отрезков — по одному на каждом уровне. □

Пример. Работа процедуры запроса суммы на отрезке $A[1..6]$. Процедура посещает только выделенные узлы.



9.6. Кратчайший остов

Задача отыскания кратчайшего остова графа является классической задачей теории графов. Методы решения этой задачи послужили основой для многих других важных результатов. В частности, исследования алгоритма Краскала[1], описанного в п. 9.6.3, привели в конечном счёте к теории жадных алгоритмов, изложенной в п. 2.7.4.

9.6.1. Стягивающие деревья и остовы

Пусть $G(V, E)$ — граф. *Остовный подграф* графа $G(V, E)$ — это подграф, содержащий все вершины. Остовный подграф, являющийся деревом, называется *остовом*, или *каркасом*. Если подграф, являющийся деревом, в то же время является остовным, то такое дерево называется *стягивающим деревом*. Таким образом, стягивающее дерево и остов — это одно и то же.

ЗАМЕЧАНИЕ

Остов определяется множеством рёбер, поскольку вершины остова суть вершины графа. Таким образом, всякий остов определяет разбиение множества рёбер графа на два множества: множество рёбер остова и множество хорд остова.

Несвязный граф не имеет остова. Связный граф может иметь много остовов. Но несвязный граф имеет *остовный лес* — множество остовов компонент связности.

Если задать длины рёбер, то можно поставить задачу нахождения *кратчайшего* остова.

ЗАМЕЧАНИЕ

Существует множество различных способов найти какой-то остов графа. Например, алгоритм поиска в глубину строит остов (по рёбрам возврата).

ЗАМЕЧАНИЕ

Множество кратчайших путей из заданной вершины ко всем остальным также образует остов. Однако этот остов может не быть кратчайшим.

Пример. На рис. 9.22 показаны диаграммы (*слева направо*) графа, дерева кратчайших путей из вершины 1 с суммарным весом 5 и два кратчайших остова этого графа.

[1] Джозеф Бернард Краскал (1928–2010).

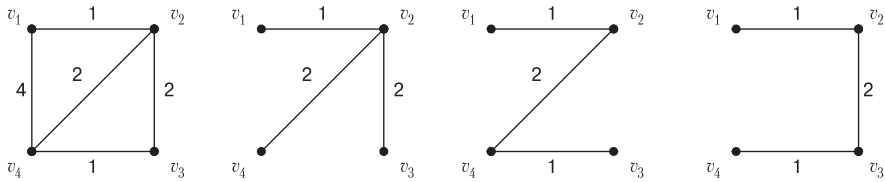


Рис. 9.22. Граф, дерево кратчайших путей и два кратчайших остова

ОТСТУПЛЕНИЕ

Задача нахождения кратчайшего остова имеет множество практических интерпретаций. Например, пусть задано множество аэродромов и нужно определить минимальный (по сумме расстояний) набор авиарейсов, который позволил бы перелететь с любого аэродрома на любой другой. Решением этой задачи будет кратчайший остов полного графа расстояний между аэродромами.

9.6.2. Схема алгоритма построения кратчайшего остова

Рассмотрим следующую схему алгоритма построения кратчайшего остова. Пусть T — множество непересекающихся деревьев, являющихся подграфами графа G. Вначале T состоит из отдельных вершин графа G, в конце T содержит единственный элемент — кратчайший остов графа G.

Алгоритм 9.22. Построение кратчайшего остова

Вход: граф $G(V, E)$, заданный матрицей длин рёбер C.
Выход: кратчайший остов T.

```
T := V
while в T больше одного элемента do
  взять любое поддерево из T
  найти к нему ближайшее
  соединить эти деревья в T
end while
```

Обоснование. Возьмём произвольное дерево T_i и выберем ближайшее к нему дерево T_j в T, $T_i \cap T_j = \varnothing$. Положим $\Delta_{i,j} := \min\limits_{t_i \in T_i, t_j \in T_j} d(t_i, t_j)$. Так как с самого начала все вершины G покрыты деревьями из T, а T_j выбирается ближайшим к T_i, то $\Delta_{i,j}$ всегда реализуется на некотором ребре с длиной $c_{i,j}$. Далее индукцией по шагам алгоритма 9.22. покажем, что все рёбра, включенные в деревья из T, принадлежат кратчайшему остову — SST[1]. Вначале выбранных рёбер нет, поэтому их множество включается в кратчайший остов. Пусть теперь все рёбра, добавленные в T, принадлежат SST. Рассмотрим очередной шаг алгоритма. Пусть на этом шаге добавлено ребро (i, j), соединяющее поддерево T_i с поддеревом T_j. Если $(i, j) \notin SST$, то, поскольку SST является деревом и, стало быть, связен, $\exists (i^*, j^*) \in SST$, соединяющее T_i с остальной частью SST. Тогда удалим из SST ребро (i^*, j^*) и добавим ребро (i, j): $SST' := SST - (i^*, j^*) + (i, j)$. Полученный подграф SST' является остовом, причём более коротким, чем SST, что противоречит выбору SST. □

[1] *SST — Shortest Spanning Tree* — стандартное обозначение для кратчайшего остова.

ЗАМЕЧАНИЕ

Различные способы выбора поддерева для наращивания на первом шаге тела цикла дают различные конкретные варианты алгоритма построения SST.

9.6.3. Алгоритм Краскала

Следующий алгоритм, известный как *алгоритм Краскала*, находит кратчайший остов в связном графе.

Алгоритм 9.23. Алгоритм Краскала

Вход: список E рёбер графа G с длинами, упорядоченный в порядке возрастания длин.
Выход: множество T рёбер кратчайшего остова.

```
T := ∅
k := 1  // номер рассматриваемого ребра
for i from 1 to p − 1 do
  while z(T + E[k]) > 0 do
    k := k + 1  // пропустить это ребро
  end while
  T := T + E[k]  // добавить это ребро в SST
  k := k + 1  // и исключить его из рассмотрения
end for
```

Обоснование. Для обоснования алгоритма Краскала достаточно показать, что выдерживается схема алгоритма предыдущего параграфа. Действительно, поскольку в множестве рёбер T нет циклов по построению, это множество суть совокупность рёбер некоторого множества деревьев. Если множество T содержит более одного дерева, то существует ребро, при добавлении которого не возникает цикла — оно соединяет два дерева в T. Добавляемое ребро — кратчайшее возможное, значит, на нём реализуется расстояние между некоторыми деревьями в T. По построению в конце работы алгоритма множество рёбер T содержит $p-1$ элемент, а значит, является искомым остовом. □

ТЕОРЕМА. *Семейство всех таких подмножеств множества рёбер графа, которые не содержат циклов, является матроидом.*

Доказательство. Пустое множество рёбер не содержит циклов, и аксиома M_1 выполнена. Далее, если множество рёбер не содержит циклов, то любое его подмножество также не содержит циклов, и аксиома M_2 выполнена. Пусть теперь $E' \subset E$ — произвольное множество рёбер, а G' — правильный подграф графа G, образованный этими рёбрами. Очевидно, что любое максимальное не содержащее циклов подмножество множества E' является объединением остовов компонент связности графа G' и по теореме п. 9.1.2 содержит $p(G') - k(G')$ элементов. Таким образом, все максимальные не содержащие циклов подмножества произвольного множества рёбер содержат одинаковое количество элементов, и по теореме п. 2.7.3 семейство ациклических подмножеств рёбер образует матроид. □

Таким образом, алгоритм Краскала, как жадный алгоритм применённый к матроиду, находит ациклическое подмножество рёбер наименьшего веса. По построению алгоритма (основной цикл до $p-1$) это подмножество рёбер древочисленно, а значит, является кратчайшим остовом.

9.6.4. Алгоритм Прима

В алгоритме Прима[1] кратчайший остов порождается в процессе разрастания одного дерева, к которому присоединяются одиночные вершины. При этом для каждой вершины v, кроме начальной, используются две пометки: $\alpha[v]$ — это ближайшая к v вершина, уже включённая в остов, а $\beta[v]$ — это длина ребра, соединяющего v с остовом. Если вершину v ещё нельзя соединить с остовом одним ребром, то $\alpha[v] := 0, \beta[v] := \infty$.

Алгоритм 9.24. Алгоритм Прима

Вход: граф $G(V, E)$, заданный матрицей длин рёбер C.
Выход: множество T рёбер кратчайшего остова.

```
select u ∈ V   // выбираем произвольную вершину
S := {u}  // S — множество вершин, включённых в кратчайший остов
T := ∅  // T — множество рёбер, включённых в кратчайший остов
for v ∈ V − u do
  if v ∈ Γ(u) then
    α[v] := u  // u — ближайшая вершина остова
    β[v] := C[u, v]  // C[u, v] — длина соответствующего ребра
  else
    α[v] := 0  // ближайшая вершина остова неизвестна
    β[v] := ∞  // и расстояние также неизвестно
  end if
end for
for i from 1 to p − 1 do
  x := ∞  // начальное значение для поиска ближайшей вершины
  for v ∈ V \ S do
    if β[v] < x then
      w := v  // нашли более близкую вершину
      x := β[v]  // и расстояние до неё
    end if
  end for
  S := S + w  // добавляем найденную вершину в остов
  T := T + (α[w], w)  // добавляем найденное ребро в остов
  for v ∈ Γ(w) do
    if v ∉ S then
      if β[v] > C[v, w] then
        α[v] := w  // изменяем ближайшую вершину остова
        β[v] := C[v, w]  // и длину ведущего к ней ребра
      end if
    end if
  end for
end for
```

Обоснование. Алгоритм Прима буквально следует схеме алгоритма 9.22. В качестве первого из соединяемых деревьев используется одно и то же разрастающееся дерево T, а в качестве второго — ближайшая одиночная вершина. □

[1] Роберт Клей Прим (род. 1921).

ОТСТУПЛЕНИЕ

Задача о нахождении кратчайшего остова принадлежит к числу немногих задач теории графов, которые можно считать полностью решёнными. Между тем, если изменить условия задачи, на первый взгляд даже незначительно, то она оказывается гораздо более трудной. Рассмотрим следующую задачу. Пусть задано множество городов на плоскости и нужно определить минимальный (по сумме расстояний) набор железнодорожных линий, который позволил бы переехать из любого города в любой другой. Кратчайший остов полного графа расстояний между городами не будет являться решением этой (практически, очевидно, очень важной) задачи, известной как *задача Штейнера*. В задаче Штейнера допускается введение дополнительных вершин, называемых *точками Штейнера*. На рис. 9.23 приведены, соответственно, диаграммы кратчайшего остова, наивного «решения» задачи Штейнера и правильного решения для случая, когда города расположены в вершинах квадрата. Задача Штейнера на плоскости хорошо изучена, в частности, известно много важных свойств точного решения. Например, известно, что каждая точка Штейнера имеет степень 3 и отрезки в ней встречаются под углом 120°. Тем не менее до сих пор неизвестно достаточно эффективных алгоритмов решения данной задачи, и она остаётся предметом интенсивных исследований.

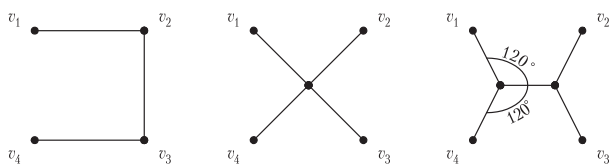


Рис. 9.23. Кратчайший остов, приближённое и точное решения задачи Штейнера

ЗАМЕЧАНИЕ

Материал этой главы затрагивает вопросы, которые очень часто возникают в практическом программировании. Поэтому различные сведения о деревьях можно найти не только в специальных учебниках по теории графов, но и в книгах по программированию и конструированию эффективных алгоритмов. В качестве рекомендуемых для программистов источников назовём [1], [10], [3]. Алгоритмы 9.1. и 9.2. заимствованы из редкой книги [8], в которой приведено большое число алгоритмов на графах, в том числе не очень широко известных. Подробное изложение и анализ алгоритмов работы с различными классами деревьев имеется в прекрасном учебнике [2]. Алгоритмы поиска кратчайшего остова (алгоритмы 9.1–9.2) принадлежат к числу известных классических алгоритмов, их описание можно найти во многих источниках, например в [13].

Глава 10 Циклы, независимость и раскраска

После рассмотрения ациклических связных графов, то есть деревьев, естественно перейти к рассмотрению графов с циклами. Некоторые задачи, связанные с циклами, в частности с гамильтоновыми циклами, оказываются труднорешаемыми, так называемыми переборными задачами. В этой главе на примере задачи отыскания наибольшего независимого множества вершин рассматриваются подходы к программному решению таких задач. К числу переборных задач относится и задача раскрашивания графов, которая имеет много приложений в программировании. С раскраской связана задача об укладывании графа на заданной поверхности, имеющая приложения в электронике.

Таким образом, в этой заключительной главе рассматриваются некоторые известные задачи на графах и указываются связи между ними. В частности, приводятся решения тех исторических задач, с которых мы начали изложение теории графов в главе 7.

10.1. Фундаментальные циклы и разрезы

Первый раздел главы посвящён установлению структуры множеств циклов и разрезов в графе с точки зрения векторных пространств.

10.1.1. Циклы и разрезы

Цикл может входить только в одну компоненту связности графа $G(V, E)$, а в несвязном графе понятие разреза является вырожденным, поэтому без ограничения общности в этом разделе граф $G(V, E)$ считается связным. Цикл не может содержать одно ребро более одного раза, поэтому в этом разделе цикл рассматривается как множество рёбер. В связи с этим можно дать эквивалентное определение простого цикла: *простым* называется цикл, никакое собственное подмножество которого циклом не является. Напомним, что *разрезом* связного графа называется множество рёбер, удаление которых делает граф несвязным. Заметим, что любое разбиение множества вершин V на два непустых подмножества, $V_1 \neq \varnothing$ и $V_2 \neq \varnothing$, $V_1 \cap V_2 = \varnothing$, $V_1 \cup V_2 = V$, определяет разрез $S := \{(v_1, v_2) \in E \mid v_1 \in V_1 \ \& \ v_2 \in V_2\}$, поскольку правильные подграфы G_1 и G_2, определяемые подмножествами V_1 и V_2 соответственно, являются, очевидно, компонентами связности графа $G - S$. Заметим далее, что множества V_1 и V_2 определяют друг друга: $V_1 = V \setminus V_2$, $V_2 = V \setminus V_1$, поэтому достаточно задать только одно из них. Естественно ввести обозначение $\forall U \subset V \ \left(\overline{U} \stackrel{\text{Def}}{=} V \setminus U\right)$.

Введём обозначение $E(V_1, V_2)$ для множества рёбер, соединяющих два дизъюнктных непустых подмножества вершин графа $G(V, E)$:

$$E(V_1, V_2) \stackrel{\text{Def}}{=} \{(v_1, v_2) \in E \mid v_1 \in V_1 \ \& \ v_2 \in V_2\},$$

где $V_1 \subset V$, $V_2 \subset V$, $V_1 \neq \varnothing$, $V_2 \neq \varnothing$, $V_1 \cap V_2 = \varnothing$. Заметим, что $E(V_1, V_2) = E(V_2, V_1)$.

Разрез связного графа $G(V, E)$, определяемый непустым подмножеством U множества вершин V, называется *правильным* разрезом и обозначается $S(U)$:

$$S(U) \stackrel{\text{Def}}{=} \left\{(v_1, v_2) \in E \mid v_1 \in U \ \& \ v_2 \in \overline{U}\right\} = E(U, \overline{U}).$$

Правильный разрез не содержит «лишних» рёбер, то есть таких рёбер, включение или исключение которых не меняет компонент связности, получаемых при удалении рёбер разреза. Ясно, что всякий разрез содержит некоторый правильный разрез. В этом разделе разрез считается правильным, если не оговорено обратное.

ЛЕММА. *Симметрическая разность двух различных правильных разрезов, определяемых множествами V_1 и V_2, является правильным разрезом, определяемым симметрической разностью множеств V_1 и V_2:*

$$V_1 \neq V_2 \Longrightarrow S(V_1) \triangle S(V_2) = S(V_1 \triangle V_2).$$

Доказательство. «Пересечение» правильных разрезов $S(V_1)$ и $S(V_2)$ образует разбиение множества вершин V на четыре подмножества:

$$V_{11} := V_1 \cap V_2, \quad V_{10} := V_1 \cap \overline{V_2}, \quad V_{01} := \overline{V_1} \cap V_2, \quad V_{00} := \overline{V_1} \cap \overline{V_2}.$$

(рис. 10.1). В этих обозначениях

$$S(V_1) = E(V_{11}, V_{01}) \cup E(V_{10}, V_{01}) \cup E(V_{11}, V_{00}) \cup E(V_{10}, V_{00}),$$
$$S(V_2) = E(V_{11}, V_{10}) \cup E(V_{01}, V_{10}) \cup E(V_{11}, V_{00}) \cup E(V_{01}, V_{00}),$$

откуда, учитывая, что $E(V_{10}, V_{01}) = E(V_{01}, V_{10})$, имеем

$$S(V_1) \triangle S(V_2) = E(V_{11}, V_{01}) \cup E(V_{10}, V_{00}) \cup E(V_{11}, V_{10}) \cup E(V_{01}, V_{00}).$$

Заметим, что

$$V_1 \triangle V_2 = (V_1 \cap \overline{V_2}) \cup (\overline{V_1} \cap V_2) = V_{10} \cup V_{01},$$
$$\overline{V_1 \triangle V_2} = (V_1 \cap V_2) \cup (\overline{V_1} \cap \overline{V_2}) = V_{11} \cup V_{00}.$$

Поэтому

$$S(V_1 \triangle V_2) = E(V_{10}, V_{11}) \cup E(V_{10}, V_{00}) \cup E(V_{01}, V_{11}) \cup E(V_{01}, V_{00}),$$

и, учитывая, что $E(V_{10}, V_{11}) = E(V_{11}, V_{10})$ и $E(V_{01}, V_{11}) = E(V_{11}, V_{01})$, окончательно имеем $S(V_1) \triangle S(V_2) = S(V_1 \triangle V_2)$. □

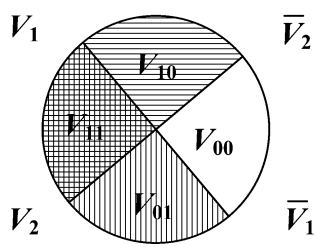


Рис. 10.1. К доказательству леммы

Простым разрезом называется минимальный разрез, то есть такой разрез, никакое собственное подмножество которого разрезом не является.

ЗАМЕЧАНИЕ

Всякий простой разрез является правильным, но не всякий правильный разрез является простым.

Пример. Рассмотрим граф G, представленный на рис. 10.2, *а*. Разрез $S = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_4, v_5), (v_2, v_3)\}$ графа G не является правильным разрезом (рис. 10.2, *б*). Исключив ребро (v_2, v_3), получим $S1 = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_5, v_4)\}$ — правильный разрез графа G (рис. 10.2, *в*), причём $S1$ не является простым разрезом. Правильный разрез $S1$ содержит в себе два простых разреза: $S1_1 = \{(v_1, v_2), (v_1, v_3), (v_1, v_4)\}$ и $S1_2 = \{(v_5, v_4)\}$, (рис. 10.2, *г* и *д*).

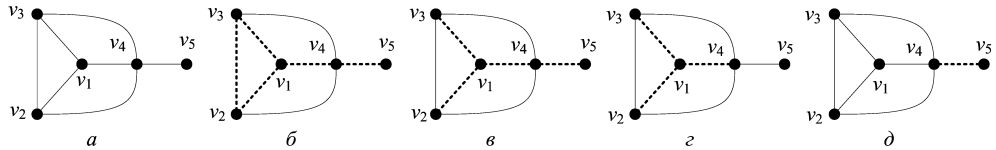


Рис. 10.2. Правильные и простые разрезы

ЗАМЕЧАНИЕ

Чем больше в графе циклов, тем труднее его разре́зать. В дереве, напротив, каждое ребро само по себе является простым разрезом.

10.1.2. Фундаментальная система циклов и циклический ранг

Пусть $T(V, E_T)$ — некоторый остов графа $G(V, E)$. *Кодеревом* $T^*(V, E_T^*)$ остова T называется остовный подграф, такой, что $E_T^* = E \setminus E_T$. (Кодерево не является деревом!) Рёбра кодерева являются *хордами* остова. По теореме п. 9.1.2 об основных свойствах деревьев каждая хорда $e \in T^*$ остова T порождает ровно один простой цикл, обозначим его Z_e. Таким образом, имеем систему простых циклов $\mathfrak{Z} \stackrel{\text{Def}}{=} \{Z_e\}_{e \in T^*}$, определяемых выбранным остовом T, которая называется *фундаментальной системой циклов*. Циклы фундаментальной системы называются *фундаментальными*, а количество циклов в (данной) фундаментальной системе называется *циклическим рангом* (или *цикломатическим числом*) графа G и обозначается $m(G)$.

ТЕОРЕМА. *Любой цикл в связном графе $G(V, E)$ можно представить как симметрическую разность нескольких фундаментальных циклов из системы $\mathfrak{Z}$, определяемой произвольным остовом T.*

Доказательство. Если в графе G нет циклов, то это дерево, $G = T$, $T^* = \varnothing$, $\mathfrak{Z} = \varnothing$, и утверждение теоремы тривиально. Рассмотрим цикл Z в графе G. Этот цикл содержит хорды $e_1, \ldots, e_n \in T^*$ (рис. 10.3). Такие хорды в Z обязательно есть, в противном случае $Z \subset T$, что невозможно, поскольку T — дерево. Докажем индукцией по n, что $Z = Z_{e_1} \triangle \ldots \triangle Z_{e_n}$. База: пусть $n = 1$, тогда $e_1 \notin T$, $Z - e_1 \subset T$ и $Z = Z_{e_1}$, так как если бы $Z \neq Z_{e_1}$, то концы e_1 были бы соединены в T двумя цепями, что невозможно по теореме п. 9.1.2. Пусть (индукционное предположение) $Z = Z_{e_1} \triangle \ldots \triangle Z_{e_m}$ для всех циклов Z с числом хорд $m < n$. Рассмотрим цикл Z с n хордами $e_1, \ldots, e_n \in T^*$ и цикл Z_{e_n}. Имеем $Z' := Z \triangle Z_{e_n} = (Z - e_n) \cup (Z_{e_n} - e_n)$ — тоже цикл (возможно, не простой). Но Z' содержит только $n - 1$ хорд $e_1, \ldots, e_{n-1}$. По индукционному предположению $Z' = Z_{e_1} \triangle \ldots \triangle Z_{e_{n-1}}$. Имеем $Z = Z' \triangle Z_{e_n} =$
$= \left(Z_{e_1} \triangle \ldots \triangle Z_{e_{n-1}}\right) \triangle Z_{e_n} = Z_{e_1} \triangle \ldots \triangle Z_{e_{n-1}} \triangle Z_{e_n}$. □

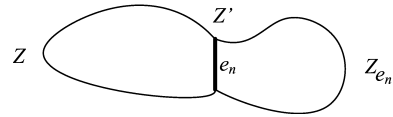


Рис. 10.3. К доказательству теоремы п. 10.1.2

СЛЕДСТВИЕ. *Количество циклов в фундаментальной системе равно числу хорд остова:* $m(G) = q - p + 1$.

Доказательство. $m(G) = q(T^*) = q(G) - q(T) = q - (p-1) = q - p + 1$. □

Пример. На рис. 10.4 представлена система фундаментальных циклов, определяемых некоторым остовом. Рёбра остова выделены жирными линиями, а соответствующие фундаментальные циклы, их четыре — $Z_1 = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$, $Z_2 = \{(v_2, v_5), (v_5, v_3), (v_3, v_2)\}$, $Z_3 = \{(v_2, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_2)\}$, $Z_4 = \{(v_2, v_4), (v_4, v_5), (v_5, v_2)\}$, — пунктирными контурами.

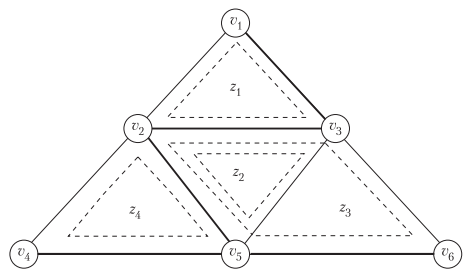


Рис. 10.4. Фундаментальные циклы

ЗАМЕЧАНИЕ

Совокупность всевозможных симметрических разностей фундаментальных циклов содержит множеств больше, чем нужно: в неё входят не только все циклы графа G, но и некоторые объединения таких циклов. В частности, симметрическая разность двух (фундаментальных) циклов, которые не имеют общих вершин, снова даёт два этих же цикла. Иногда множество объектов, определяемое всевозможными циклическими разностями фундаментальных циклов, называют множеством *циклических векторов*.

Пример. На рис. 10.4 нет фундаментальных циклов, не имеющих общих вершин. Симметрическая разность трёх фундаментальных циклов, $Z_1 \triangle Z_3 \triangle Z_4$, даёт цикл $(v_1, v_2), (v_2, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_1)$ — «внешнюю границу» графа. Симметрическая разность всех фундаментальных циклов $Z_1 \triangle Z_2 \triangle Z_3 \triangle Z_4$ даёт эйлеров цикл данного графа (см. раздел 10.2).

10.1.3. Фундаментальная система разрезов и коциклический ранг

Пусть опять $T(V, E_T)$ — некоторый остов графа $G(V, E)$. Рассмотрим ребро остова $e \in E_T$ и определим разрез S_e следующим образом. Так как ребро e — разрез дерева

T, оно разбивает множество V вершин T на два непустых подмножества, V_1 и V_2, так что $V_1 \subset V$, $V_1 \neq \varnothing$, $V_2 \subset V$, $V_2 \neq \varnothing$, $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$. Включим в разрез S_e ребро e и все хорды остова T, соединяющие вершины множества V_1 с вершинами множества V_2:

$$S_e \stackrel{\text{Def}}{=} e + \{(v_1, v_2) \in T^* \mid v_1 \in V_1 \ \& \ v_2 \in V_2\} = E(V_1, V_2).$$

Тогда S_e — это простой разрез. Система разрезов $\mathcal{S} \stackrel{\text{Def}}{=} \{S_e\}_{e \in T}$ называется *фундаментальной системой разрезов*. Разрезы фундаментальной системы называются *фундаментальными*, а количество разрезов в (данной) фундаментальной системе называется *коциклическим рангом* (или *коцикломатическим числом*) графа G и обозначается $m^*(G)$.

ЗАМЕЧАНИЕ

Между циклами и правильными разрезами существует определённая двойственность, поэтому правильные разрезы иногда называют *коциклами*. Отсюда название «фундаментальная система коциклов» и «коциклический ранг». Детальное рассмотрение этой двойственности выходит за рамки данной книги.

ТЕОРЕМА. *Любой правильный разрез в связном графе $G(V, E)$ можно представить как симметрическую разность некоторых фундаментальных разрезов из системы $\mathcal{S}$, определяемой произвольным остовом T.*

Доказательство. Действительно, любой разрез S содержит хотя бы одно ребро из остова T, так как T — связный и содержит все вершины G. Пусть S — правильный разрез, который содержит рёбра $e_1, \ldots, e_n \in T$. Докажем индукцией по n, что $S = S_{e_1} \triangle \ldots \triangle S_{e_n}$. База: пусть $n = 1$, тогда $T - e_1 = T_1 \cup T_2$, где T_1 и T_2 — компоненты, получаемые из остова T удалением ребра e_1. Имеем $S_{e_1} \subset S$, иначе S не был бы разрезом, и $S \subset S_{e_1}$, иначе S не был бы правильным разрезом. Таким образом, $S = S_{e_1}$. Пусть теперь $S = S_{e_1} \triangle \ldots \triangle S_{e_m}$ для всех правильных разрезов S с числом рёбер остова $m < n$. Рассмотрим правильный разрез S с n рёбрами $e_1, \ldots, e_n \in T$. Положим $S' := S \triangle S_{e_n}$, то есть исключим из разреза S все рёбра фундаментального разреза S_{e_n}. Разрез S' — правильный по лемме п. 10.1.1 и содержит рёбра $e_1, \ldots, e_{n-1}$, а значит, по индукционному предположению $S' = S_{e_1} \triangle \ldots \triangle S_{e_{n-1}}$. Имеем $S = S' \triangle S_{e_n}$ и окончательно $S = S_{e_1} \triangle \ldots \triangle S_{n-1} \triangle S_{e_n}$. □

СЛЕДСТВИЕ. *Количество разрезов в фундаментальной системе равно числу рёбер остова: $m^*(G) = p - 1$.*

Доказательство. По определению. □

Пример. На рис. 10.5 представлены все пять фундаментальных разрезов, соответствующих графу и его остову, представленным на рис. 10.4.

ЗАМЕЧАНИЕ

Совокупность всевозможных симметрических разностей фундаментальных разрезов содержит множеств меньше, чем может понадобиться: в неё входят только правильные разрезы, разрезы, не являющиеся правильными, в ней не содержатся. В частности, разрез может содержать циклы, в правильном разрезе или симметрической разности правильных разрезов циклов быть не может, поэтому разрезов с циклами нам не получить с помощью фундаментальной системы разрезов.

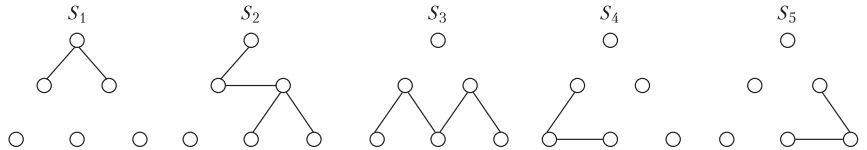


Рис. 10.5. Фундаментальные разрезы

Пример. Обратимся к приведённому на рис. 10.4 графу. Рассмотрим разрез $S = (v_1, v_2), (v_2, v_3), (v_3, v_5), (v_5, v_6), (v_6, v_3), (v_3, v_1)$. Этот разрез графа не является правильным и его нельзя представить в виде симметрической разности фундаментальных разрезов.

10.1.4. Подпространства циклов и коциклов

Рассмотрим векторное пространство (п. 2.4.1) над двоичной арифметикой (п. 2.3.3), натянутое на множество рёбер E графа $G(V, E)$. Элемент этого векторного пространства (линейную комбинацию, п. 2.4.2) можно отождествлять с подмножеством рёбер: если коэффициент в линейной комбинации равен 1, то ребро входит в подмножество, если коэффициент равен 0, то не входит. При такой интерпретации сложению векторов соответствует симметрическая разность множеств рёбер. Множество циклических векторов и множество правильных разрезов замкнуты относительно симметрической разности, а потому являются подпространствами общего пространства подмножеств рёбер графа.

ЗАМЕЧАНИЕ

Строго говоря, множество циклов, равно как и множество разрезов, не образует подпространства, поскольку не замкнуто относительно симметрической разности. Множество циклических векторов шире множества циклов, а множество правильных разрезов у́же множества разрезов. Однако интуитивно ясно, что «по потребительским свойствам» циклические векторы и циклы, равно как разрезы и правильные разрезы, весьма близки, поэтому далее рассматриваются только циклические векторы и правильные разрезы, которые для краткости называются циклами и разрезами (коциклами).

Множество циклов $\{Z_i\}_{i=1}^n$ называется *независимым*, если ни один из циклов Z_i не является линейной комбинацией остальных.

Множество разрезов $\{S_i\}_{i=1}^n$ называется *независимым*, если ни один из разрезов S_i не является линейной комбинацией остальных.

Максимальное независимое множество циклов (или минимальное множество циклов, от которых зависят все остальные, или независимое порождающее множество циклов) является *базисом* пространства циклов. Максимальное независимое множество разрезов (или минимальное множество разрезов, от которых зависят все остальные, или независимое порождающее множество разрезов) является *базисом* пространства разрезов.

Введенная фундаментальная система циклов является базисом подпространства циклов. Действительно, циклы любой фундаментальной системы $\mathcal{Z} = \{Z_e\}_{e \in T^*}$ независимы, поскольку каждый из них содержит индивидуальную хорду $e \in T^*$, и по теореме п. 10.1.2 фундаментальная система порождает множество циклов. Таким образом, циклический ранг — это размерность пространства циклов.

ЗАМЕЧАНИЕ

Фундаментальные системы циклов, порождаемые остовами, — это не единственные базисы пространства циклов. Например, четыре «естественных» треугольника на рисунке в п. 10.1.2 — $\{(v_1,v_2),(v_2,v_3),(v_3,v_1)\}$, $\{(v_2,v_4),(v_4,v_5),(v_5,v_2)\}$, $\{(v_2,v_5),(v_5,v_3),(v_3,v_2)\}$, $\{(v_3,v_5),(v_5,v_6),(v_6,v_3)\}$ — являются базисом подпространства циклов, но не являются фундаментальной системой ни для какого остова.

Введённая фундаментальная система разрезов является базисом подпространства разрезов. Действительно, разрезы любой фундаментальной системы $\mathcal{S}=\{S_e\}_{e\in T}$ независимы, поскольку каждый из них содержит индивидуальное ребро $e\in T$, и по теореме 10.1.3 фундаментальная система порождает множество разрезов. Таким образом, коциклический ранг — это размерность пространства разрезов.

ЗАМЕЧАНИЕ

Все утверждения этого раздела распространяются на случай мультиграфов.

10.2. Эйлеровы циклы

Здесь приведено исчерпывающее решение задачи о кёнигсбергских мостах (п. 7.1.1), приведшей к исторически первой успешной попытке развития теории графов как самостоятельного предмета.

10.2.1. Эйлеровы графы

Если связный граф имеет цикл (не обязательно простой), содержащий все рёбра графа, то такой цикл называется *эйлеровым* циклом, а граф называется *эйлеровым* графом. Если связный граф имеет цепь (не обязательно простую), содержащую все рёбра, то такая цепь называется *эйлеровой* цепью, а граф называется *полуэйлеровым* графом. И эйлеров цикл и эйлерова цепь содержат не только все рёбра, но и все вершины графа (возможно, по нескольку раз).

ТЕОРЕМА. *Если граф G связен и нетривиален, то следующие утверждения эквивалентны:*

1) *G — эйлеров граф;*
2) *каждая вершина G имеет чётную степень;*
3) *множество рёбер G можно разбить на простые циклы.*

Доказательство.

[1$\Longrightarrow$ 2] Пусть Z — эйлеров цикл в G. Двигаясь по Z, подсчитаем степени вершин, полагая их до начала подсчёта нулевыми. Прохождение каждой вершины вносит 2 в степень этой вершины. Поскольку Z содержит все рёбра, то, когда обход Z будет закончен, будут учтены все рёбра, а степени всех вершин — чётные.

[2$\Longrightarrow$ 3] G — связный и нетривиальный граф, следовательно, $\forall\, v_i\ \ (d(v_i)>0)$. Степени вершин чётные, следовательно, $\forall\, v_i\ \ (d(v_i)\geqslant 2)$. Имеем $2q=\sum_{i=1}^{p} d(v_i)\geqslant 2p \Longrightarrow q\geqslant p \Longrightarrow q>p-1$. Следовательно, граф G — не дерево, а значит, граф G содержит (хотя бы один) простой цикл Z_1. (Z_1 — множество рёбер.) Тогда $G-Z_1$ — остовный подграф, в котором опять все степени вершин чётные. Исключим из рассмотрения изолированные вершины. Таким образом, $G-Z_1$ тоже удовлетворяет условию 2, следовательно, существует простой цикл $Z_2\subset(G-Z_1)$.

Далее выделяем циклы Z_i, пока граф не будет пуст (рис. 10.6). Имеем $E = \bigcup Z_i$ и $\bigcap Z_i = \varnothing$.

[3$\Longrightarrow$1] Возьмем какой-либо цикл Z_1 из данного разбиения. Если $Z_1 = E$, то теорема доказана. Если нет, то существует цикл Z_2, не имеющий общих рёбер с Z_1 (рис. 10.6), такой, что $\exists\, v_1\ ((v_1 \in Z_1\ \&\ v_1 \in Z_2))$, так как G связен. Маршрут $Z_1 \cup Z_2$ является циклом и содержит все свои рёбра по одному разу. Если $Z_1 \cup Z_2 = E$, то теорема доказана. Если нет, то существует цикл Z_3, такой, что $\exists\, v_2\ \ (v_2 \in Z_1 \cup Z_2\ \&\ v_2 \in Z_3)$. Далее будем наращивать эйлеров цикл, пока он не исчерпает разбиения. □

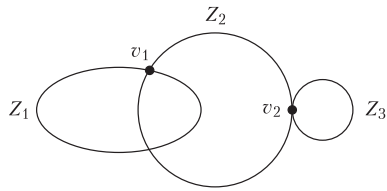


Рис. 10.6. К доказательству теоремы об эйлеровых циклах

10.2.2. Алгоритм построения эйлерова цикла в эйлеровом графе

Алгоритм 10.1. Построение эйлерова цикла

Вход: эйлеров граф $G(V, E)$, заданный списками смежности ($\Gamma[v]$ — список вершин, смежных с вершиной v).
Выход: последовательность вершин эйлерова цикла.

```
S := ∅  // стек для хранения вершин
select v ∈ V  // произвольная вершина
v → S  // положить v в стек S
while S ≠ ∅ do
  v := top S  // v — верхний элемент стека
  if Γ[v] = ∅ then
    v ← S; yield v  // очередная вершина эйлерова цикла
  else
    select u ∈ Γ[v]  // взять первую вершину из списка смежности
    u → S  // положить u в стек
    Γ[v] := Γ[v] − u; Γ[u] := Γ[u] − v  // удалить ребро (v, u)
  end if
end while
```

Обоснование. Принцип действия этого алгоритма заключается в следующем. Начав с произвольной вершины v, строим путь, удаляя рёбра и запоминая вершины в стеке, до тех пор пока множество смежности очередной вершины не окажется пустым, что означает, что путь удлинить нельзя. Заметим, что при этом мы с необходимостью придём в ту вершину, с которой начали. В противном случае вершина v имеет нечётную степень, что невозможно по условию. Таким образом, из графа были удалены рёбра цикла, а вершины цикла были сохранены в стеке S. Заметим, что при этом степени всех вершин остались чётными. Далее вершина v выводится в качестве первой вершины эйлерова цикла, а процесс продолжается с вершины, стоящей на вершине стека. □

ЗАМЕЧАНИЕ

В предыдущем разделе был установлен эффективный способ проверки наличия эйлерова цикла в графе. А именно для этого достаточно убедиться, что степени всех вершин чётные, что нетрудно сделать при любом представлении графа. Приведённый *алгоритм* находит эйлеров цикл в графе, если известно, что граф эйлеров, то есть цикл заведомо существует. Если же граф не эйлеров и цикла не существует, то алгоритм не применим.

10.2.3. Оценка числа эйлеровых графов

Пусть $\mathcal{G}(p)$ — множество всех графов с p вершинами, а $\mathcal{E}(p)$ — множество эйлеровых графов с p вершинами.

ЗАМЕЧАНИЕ

В этом параграфе речь идёт о числе *нумерованных* графов, то есть о числе графов, в которых вершины перенумерованы. Считается, что если перенумеровать вершины в другом порядке, то это будет другой граф. Очевидно, что нумерованных графов (любого типа) больше, чем графов, определяемых как классы эквивалентности по отношению изоморфизма, поэтому приводимые здесь оценки являются достаточно грубыми.

ТЕОРЕМА. *Эйлеровых графов почти нет, то есть* $\lim\limits_{p\to\infty} \frac{|\mathcal{E}(p)|}{|\mathcal{G}(p)|} = 0$.

Доказательство. Пусть $\mathcal{E}'(p)$ — множество графов с p вершинами и чётными степенями. Тогда по предыдущей теореме $\mathcal{E}(p) \subset \mathcal{E}'(p)$ и $|\mathcal{E}(p)| \leqslant |\mathcal{E}'(p)|$. В любом графе число вершин нечётной степени чётно, следовательно, любой граф из $\mathcal{E}'(p)$ можно получить из некоторого графа $\mathcal{G}(p-1)$, если добавить новую вершину и соединить её со всеми старыми вершинами нечётной степени. Следовательно, $|\mathcal{E}'(p)| \leqslant |\mathcal{G}(p-1)|$. Но $|\mathcal{G}(p)| = 2^{C(p,2)}$, поскольку (нумерованный) граф с p вершинами определяется подмножеством включённых в него рёбер, выбранных из множества всех возможных рёбер, а их $C(p,2)$. Заметим, что $C(k,2) - C(k-1,2) = \frac{k(k-1)}{2} - \frac{(k-1)(k-2)}{2} = k-1$. Далее имеем

$$|\mathcal{E}(p)| \leqslant |\mathcal{E}'(p)| \leqslant |\mathcal{G}(p-1)| = 2^{C(p-1,2)} = 2^{C(p,2)-(p-1)} = |\mathcal{G}(p)|2^{-(p-1)}$$

и $\dfrac{|\mathcal{E}(p)|}{|\mathcal{G}(p)|} \leqslant \dfrac{1}{2^{p-1}}$, откуда $\lim\limits_{p\to\infty} \dfrac{|\mathcal{E}(p)|}{|\mathcal{G}(p)|} = 0$. □

10.3. Гамильтоновы циклы

Название «гамильтонов цикл» произошло от задачи «Кругосветное путешествие» (рис. 10.7), придуманной Гамильтоном[1] в XIX веке: нужно обойти все вершины графа, диаграмма которого показана на рисунке (в исходной формулировке вершины были помечены названиями столиц различных стран), по одному разу и вернуться в исходную точку. Этот граф представляет собой укладку додекаэдра.

[1] Уильям Гамильтон (1805–1856).

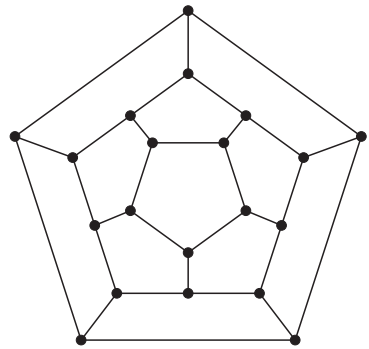


Рис. 10.7. Задача «Кругосветное путешествие»

10.3.1. Гамильтоновы графы

Если граф имеет простой цикл, содержащий все вершины графа (по одному разу), то такой цикл называется *гамильтоновым* циклом, а граф называется *гамильтоновым* графом. Если граф имеет простую цепь, содержащую все вершины графа (по одному разу), то такая цепь называется *гамильтоновой* цепью, а граф называется *полугамильтоновым* графом.

Гамильтонов цикл не обязательно содержит все рёбра графа. Ясно, что гамильтоновым может быть только связный граф.

ЗАМЕЧАНИЕ

Любой граф G можно превратить в гамильтонов, добавив достаточное количество новых вершин и инцидентных им рёбер и не добавляя рёбер, инцидентных только старым вершинам. Для этого, например, достаточно к вершинам $v_1, \ldots, v_p$ графа G добавить вершины $u_1, \ldots, u_p$ и множество рёбер $\{(v_i, u_i)\} \cup \{(u_i, v_{i+1})\}$, считаем $v_{p+1} = v_1$.

Простые необходимые и достаточные условия гамильтоновости графа неизвестны. Известны только некоторые достаточные условия, одно из которых приведено в следующей теореме.

ТЕОРЕМА (Дирак). *Если $p(G) \geqslant 3$ и $\delta(G) \geqslant p/2$, то граф G является гамильтоновым.*

Доказательство. От противного. Пусть G — не гамильтонов. Добавим к G *минимальное* количество новых вершин $u_1, \ldots, u_n$, соединяя их со *всеми* вершинами G так, чтобы граф $G' := G + u_1 + \cdots + u_n$ был гамильтоновым. Пусть $v, u_1, w, \ldots, v$ — гамильтонов цикл в графе G', причём $v \in G$, $u_1 \in G'$, $u_1 \notin G$. Такая пара вершин, v и u_1, в гамильтоновом цикле обязательно найдется, иначе граф G был бы гамильтоновым (рис. 10.8). Тогда $w \in G$, $w \notin \{u_1, \ldots, u_n\}$, иначе вершина u_1 была бы не нужна. Более того, вершина v не смежна с вершиной w, иначе вершина u_1 была бы не нужна. Далее, если в цикле $v, u_1, w, \ldots, v', w', \ldots, v$ есть вершина w', смежная с вершиной w, то вершина v' не смежна с вершиной v, так как иначе можно было бы построить гамильтонов цикл $v, v', \ldots, w, w' \ldots v$ без вершины u_1, взяв последовательность вершин $w, \ldots, v'$ в обратном порядке.

Отсюда следует, что число вершин графа G', не смежных с v, не менее числа вершин, смежных с w. Но для любой вершины w графа G в графе G' имеем $d(w) \geqslant p/2 + n$ по построению, в том числе $d(v) \geqslant p/2 + n$. Общее число вершин (смежных и не

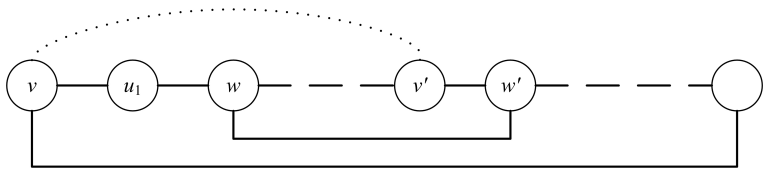


Рис. 10.8. К доказательству теоремы п. 10.3.1

смежных с v, за исключением самой вершины v) составляет $n+p-1$. Таким образом, имеем

$$n+p-1=\overline{d}(v)+d(v)\geqslant d(w)+d(v)\geqslant \frac{p}{2}+n+\frac{p}{2}+n=2n+p.$$

Следовательно, $0\geqslant n+1$, что противоречит тому, что $n>0$. □

СЛЕДСТВИЕ. *При $p\geqslant 3$ полный граф K_p гамильтонов.*

10.3.2. Задача коммивояжёра

Рассмотрим следующую задачу, известную как *задача коммивояжёра*. Имеется p городов, расстояния между которыми известны. Коммивояжёр должен посетить все p городов по одному разу, вернувшись в тот, с которого начал. Требуется найти такой маршрут движения, при котором суммарное пройденное расстояние будет минимальным. Очевидно, что задача коммивояжёра — это задача отыскания кратчайшего гамильтонова цикла в нагруженном полном графе. Можно предложить следующую простую схему решения задачи коммивояжёра: сгенерировать все $p!$ возможных перестановок вершин полного графа, подсчитать для каждой перестановки длину маршрута и выбрать из них кратчайший. Очевидно, такое вычисление потребует не менее $O(p!)$ шагов. Как известно, $p!$ — быстро растущая функция. Таким образом, решение задачи коммивояжёра описанным методом *полного перебора* оказывается практически неосуществимым даже для сравнительно небольших p. Более того, известно, что задача коммивояжёра принадлежит к числу так называемых *NP-полных* задач, подробное обсуждение которых выходит за рамки этого учебника.

Вкратце суть проблемы *NP*-полноты сводится к следующему. В различных областях дискретной математики, комбинаторики, логики и т. п. известно множество задач, принадлежащих к числу наиболее фундаментальных, для которых, несмотря на все усилия, не удалось найти алгоритмов решения, имеющих полиномиальную сложность. Более того, если бы удалось отыскать эффективный алгоритм решения хотя бы одной из этих задач, то из этого немедленно следовало бы существование эффективных алгоритмов для всех остальных задач данного класса. На этом основано общепринятое мнение, что таких алгоритмов не существует.

ОТСТУПЛЕНИЕ

Полезно сопоставить задачи отыскания эйлеровых и гамильтоновых циклов, рассмотренные в этом и предыдущем разделах. Внешне формулировки этих задач очень похожи, однако они оказываются принципиально различными с точки зрения практического применения. Уже давно Эйлером получено просто проверяемое необходимое и достаточное условие существования в графе эйлерова цикла. Что касается гамильтоновых графов, то для них неизвестны простые необходимые и достаточные условия. На основе необходимого и достаточного условия существования эйлерова цикла можно построить эффективные алгоритмы отыскания

такого цикла. В то же время задача проверки существования гамильтонова цикла оказывается *NP*-полной (так же, как и задача коммивояжёра). Далее, известно, что почти нет эйлеровых графов, и эффективный алгоритм отыскания эйлеровых циклов редко оказывается применимым на практике.

С другой стороны, можно показать, что почти все графы — гамильтоновы, то есть

$$\lim_{p\to\infty} \frac{|\mathcal{H}(p)|}{|\mathcal{G}(p)|} = 1,$$

где $\mathcal{H}(p)$ — множество гамильтоновых графов с p вершинами, а $\mathcal{G}(p)$ — множество всех графов с p вершинами. Таким образом, задача отыскания гамильтонова цикла или задача коммивояжёра являются практически востребованными, но эффективный алгоритм решения для них неизвестен (и, скорее всего, не существует).

10.3.3. Теорема Поша

В степенной последовательности гамильтонова графа не может быть много вершин с малыми степенями. Введём обозначение $N_=(G,n) \stackrel{\text{Def}}{=} |\{v \in V \mid d(v) = n\}|$ числа вершин графа G степени n, $0 \leqslant n < p$. В этих обозначениях

$$N_\leqslant(G,n) \stackrel{\text{Def}}{=} |\{v \in V \mid d(v) \leqslant n\}| = \sum_{i=0}^{n} N_=(G,i),$$

$$N_>(G,n) \stackrel{\text{Def}}{=} |\{v \in V \mid d(v) > n\}| = \sum_{i=n+1}^{p-1} N_=(G,i), \quad N_\leqslant(G,n) + N_>(G,n) = p.$$

ЛЕММА. $\forall\, G(V,E)\ (\forall\, e \notin E\ (\forall\, 0 \leqslant k < p\ (N_\leqslant(G+e,n) \leqslant N_\leqslant(G,n))))\,.$

Другими словами, добавление рёбер разве что уменьшает число вершин ограниченной степени.

Следующая теорема даёт достаточное условие того, что граф является гамильтоновым. Она обобщает результат, полученный Дираком.

ТЕОРЕМА (Поша). *Если $G(V,E)$ — связный граф, $p \geqslant 3$, $\forall\, n < (p-1)/2\ (N_\leqslant(G,n) < n)$, причём $N_\leqslant(G,(p-1)/2) \leqslant (p-1)/2$ для нечётных p, то граф G гамильтонов.*

Доказательство. От противного. Пусть G — не гамильтонов граф с p вершинами, удовлетворяющий условиям теоремы. Доказательство содержит три шага.

[1] По лемме добавление рёбер не нарушает неравенств в условии теоремы. Добавим в G все возможные рёбра так, чтобы он оставался не гамильтоновым, то есть рассмотрим *максимальный* не гамильтонов надграф G. Тогда, поскольку добавление к перестроенному максимальному графу G произвольного ребра приводит к гамильтонову графу, любые две несмежные вершины соединимы простой гамильтоновой (остовной, то есть содержащей все вершины графа) цепью.

[2] Покажем, что всякая вершина, степень которой *не меньше* $(p-1)/2$, смежна с каждой вершиной со степенью, *большей чем* $(p-1)/2$. Допустим (не теряя общности), что $d(v_1) \geqslant (p-1)/2$ и $d(v_p) \geqslant p/2 > (p-1)/2$, но вершины v_1 и v_p не смежны. Тогда существует гамильтонова цепь $\langle v_1, v_2, \ldots, v_p\rangle$, соединяющая v_1 и v_p (рис. 10.9). Все вершины, смежные с v_1, находятся в этой же цепи, поскольку цепь гамильтонова. Пусть v_1 смежна с v_j. Тогда вершина v_p не смежна с вершиной v_{j-1}, поскольку иначе в G был бы гамильтонов цикл $v_1, v_2, \ldots, v_{j-1}, v_p, v_{p-1}, \ldots, v_j, v_1$.

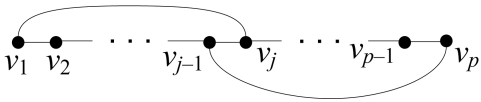


Рис. 10.9. К доказательству теоремы Поша

Положим $n := d(v_1) \leqslant (p-1)/2$. Тогда среди всех p вершин по меньшей мере n вершин, предшествующих на гамильтоновом пути вершинам, смежным с v_1, не смежны с v_p. Имеем $p/2 \leqslant d(v_p) \leqslant p-1-n \leqslant p-1-(p-1)/2 = (p-1)/2$. Противоречие.

[3] По теореме п. 10.3.1, если $\forall\, v \in V \;\; (d(v) \geqslant p/2)$, то граф гамильтонов. Таким образом, в G есть вершины со степенями меньше, чем $p/2$. Обозначим $U := \{v \in V \mid d(v) < p/2\} \neq \varnothing$ и $W := V \setminus U$. Заметим, что $\forall\, w \in W \;\; (d(w) \geqslant p/2)$. Выберем вершину $v_1 \in U$ так, что $m := d(v_1) = \max_{u \in U} d(u)$. По условию теоремы $|U| = N_{\leqslant}(G, m) < m < p/2$, и значит $|W| \geqslant p/2$ и среди вершин W есть вершина v_p не смежная с v_1. Тогда существует гамильтонова цепь $\langle v_1, v_2, \ldots, v_p \rangle$, соединяющая v_1 и v_p (рис. 10.9). Как показано в пункте 2, вершина $v_{j-1} \notin \Gamma(v_p)$, в противном случае нашёлся бы гамильтонов цикл. Но тогда $m < (p-1)/2$, потому что если $m \geqslant (p-1)/2$, то $p/2 \leqslant d(v_p) \leqslant p-1-m \leqslant p-1-(p-1)/2 = (p-1)/2$. Противоречие. По условию теоремы $N_{\leqslant}(G, m) < m$ и значит хотя бы одна из вершин v_{j-1}, предшествующих вершинам, смежным с v_1, имеет степень больше $p/2$, $d(v_{j-1}) \geqslant p/2$. Получили пару несмежных вершин, v_{j-1} и v_p, таких, что $d(v_{j-1}) \geqslant p/2$ и $d(v_p) \geqslant p/2$. Это противоречит пункту 2. □

СЛЕДСТВИЕ. *Если $p \geqslant 3$ и $d(u) + d(v) \geqslant p$ для любой пары u и v несмежных вершин графа G, то G — гамильтонов граф.*

10.4. Независимые и покрывающие множества

В этом разделе вводятся определения и рассматриваются основные свойства независимых и покрывающих множеств вершин и рёбер. Эти определения и свойства используются в последующих разделах.

Значительное количество экстремальных задач в различных предметных областях сводится к отысканию наибольших независимых или наименьших покрывающих множеств вершин и рёбер в графах, входящих в модели этих предметных областей. Поэтому данный раздел теории графов имеет множество практических приложений.

Обсуждение переборных алгоритмов решения экстремальных задач представляет особый интерес.

10.4.1. Независимые и покрывающие множества вершин и рёбер

Говорят, что вершина *покрывает* инцидентные ей рёбра, а ребро *покрывает* инцидентные ему вершины. Множество вершин, которые в совокупности покрывают все рёбра, называется *вершинным покрытием*. Наименьшее число вершин во всех вершинных покрытиях называется *числом вершинного покрытия* и обозначается α_0.

Примеры

1. $\alpha_0(K_p) = p-1$, если K_p — полный граф.

2. $\alpha_0(K_{m,n}) = \min(m, n)$, если $K_{m,n}$ — полный двудольный граф.
3. $\alpha_0(\overline{K}_p) = 0$, если $\overline{K}_p$ — вполне несвязный граф.
4. $\alpha_0(G_1 \cup G_2) = \alpha_0(G_1) + \alpha_0(G_2)$, если G_1 и G_2 — компоненты несвязного графа.

Множество рёбер, которые в совокупности покрывают все вершины, называется *рёберным покрытием*. Наименьшее число рёбер во всех рёберных покрытиях называется *числом рёберного покрытия* и обозначается α_1.

ЗАМЕЧАНИЕ

Для графа с изолированными вершинами α_1 не определено.

Примеры

1. $\alpha_1(C_{2n}) = n$, если C_{2n} — чётный цикл.
2. $\alpha_1(C_{2n+1}) = n + 1$, если C_{2n+1} — нечётный цикл.
3. $\alpha_1(K_{2n}) = n$, если K_{2n} — полный граф с чётным числом вершин.
4. $\alpha_1(K_{2n+1}) = n + 1$, если K_{2n+1} — полный граф с нечётным числом вершин.
5. $\alpha_1(K_{m,n}) = \max(m, n)$, если $K_{m,n}$ — полный двудольный граф.

Множество вершин называется *независимым* (или *внутренне устойчивым*), если никакие две из них не смежны. Наибольшее число вершин в независимом множестве вершин называется *вершинным числом независимости* и обозначается β_0.

Примеры

1. $\beta_0(K_p) = 1$, если K_p — полный граф.
2. $\beta_0(K_{m,n}) = \max(m, n)$, если $K_{m,n}$ — полный двудольный граф.
3. $\beta_0(\overline{K}_p) = p$, если $\overline{K}_p$ — вполне несвязный граф.
4. $\beta_0(G_1 \cup G_2) = \beta_0(G_1) + \beta_0(G_2)$, если G_1 и G_2 — компоненты несвязного графа.

Ясно, что множество вершин S является независимым тогда и только тогда, когда $\forall\, v \in S \;\; (\Gamma(v) \cap S = \varnothing)$.

Множество рёбер называется *независимым* (или *паросочетанием*), если никакие два из них не смежны. Наибольшее число рёбер в независимом множестве рёбер называется *рёберным числом независимости* и обозначается β_1.

Примеры

1. $\beta_1(C_{2n}) = n$, если C_{2n} — чётный цикл.
2. $\beta_1(C_{2n+1}) = n$, если C_{2n+1} — нечётный цикл.
3. $\beta_1(K_{2n}) = n$, если K_{2n} — полный граф с чётным числом вершин.
4. $\beta_1(K_{2n+1}) = n$, если K_{2n+1} — полный граф с нечётным числом вершин.
5. $\beta_1(K_{m,n}) = \min(m, n)$, если $K_{m,n}$ — полный двудольный граф.

ЗАМЕЧАНИЕ

Индексы 0 и 1 в обозначениях α_0, α_1, β_0, β_1 выбраны из следующих мнемонических соображений. Индекс 0 соответствует вершинам, так как вершина нульмерна, а 1 — рёбрам, так как ребро одномерно.

10.4.2. Связь чисел независимости и покрытий

Приведённые примеры наводят на мысль, что числа независимости и покрытия связаны друг с другом и с количеством вершин p.

ТЕОРЕМА. *Для любого нетривиального связного графа* $\alpha_0 + \beta_0 = p = \alpha_1 + \beta_1$.

Доказательство. Докажем четыре неравенства, из которых следуют два требуемых равенства.

[$\alpha_0+\beta_0 \geqslant p$] Пусть M_0 — наименьшее вершинное покрытие, то есть $|M_0| = \alpha_0$. Рассмотрим $V \setminus M_0$. Тогда $V \setminus M_0$ — независимое множество, так как если бы в множестве $V \setminus M_0$ были смежные вершины, то M_0 не было бы покрытием. Имеем $|V \setminus M_0| \leqslant \beta_0$, следовательно, $p = |M_0| + |V \setminus M_0| \leqslant \alpha_0 + \beta_0$.

[$\alpha_0+\beta_0 \leqslant p$] Пусть N_0 — наибольшее независимое множество вершин, то есть $|N_0| = \beta_0$. Рассмотрим $V \setminus N_0$. Тогда $V \setminus N_0$ — вершинное покрытие, так как нет рёбер, инцидентных только вершинам из N_0, стало быть, любое ребро инцидентно вершине (или вершинам) из $V \setminus N_0$. Имеем $|V \setminus N_0| \geqslant \alpha_0$, следовательно, $p = |N_0| + |V \setminus N_0| \geqslant \beta_0 + \alpha_0$.

[$\alpha_1 + \beta_1 \geqslant p$] Пусть M_1 — наименьшее рёберное покрытие, то есть $|M_1| = \alpha_1$. Множество M_1 не содержит цепей длиной больше 2. Действительно, если бы в M_1 была цепь длиной 3, то среднее ребро этой цепи можно было бы удалить из M_1 и это множество все равно осталось бы покрытием. Следовательно, M_1 состоит из звёзд. (*Звездой* называется граф, диаметр которого не превосходит двух, и имеется одна центральная вершина, смежная с остальными, а остальные не смежны между собой. Другими словами, звезда — это полный двудольный граф $K_{1,n}$.) Пусть этих звёзд m. Имеем $|M_1| = \sum_{i=1}^{m} n_i$, где n_i — число рёбер в i-й звезде. Заметим, что звезда из n_i рёбер покрывает $n_i + 1$ вершину. Имеем $p = \sum_{i=1}^{m}(n_i+1) = m+\sum_{i=1}^{m} n_i = m+|M_1|$. Возьмём по одному ребру из каждой звезды и составим из них множество X. Тогда $|X| = m$, множество X — независимое, то есть $|X| \leqslant \beta_1$. Следовательно, $p = |M_1| + m = |M_1| + |X| \leqslant \alpha_1 + \beta_1$.

[$\alpha_1 + \beta_1 \leqslant p$] Пусть N_1 — наибольшее независимое множество рёбер, то есть $|N_1| = \beta_1$. Построим рёберное покрытие Y следующим образом. Множество N_1 покрывает $2|N_1|$ вершин. Добавим по одному ребру, инцидентному непокрытым $p-2|N_1|$ вершинам, таких рёбер $p-2|N_1|$. Тогда множество Y — рёберное покрытие, $|Y| = |N_1| + p - 2|N_1| = p - |N_1|$ и $|Y| \geqslant \alpha_1$. Имеем $p = p - |N_1| + |N_1| = |Y| + |N_1| \geqslant \alpha_1 + \beta_1$. □

ЗАМЕЧАНИЕ

Условия связности и нетривиальности гарантируют, что все четыре инварианта определены. Хотя это условие является достаточным, оно не является необходимым. Например, для графа $K_n \cup K_n$ заключение теоремы остается справедливым, хотя условие не выполнено.

ОТСТУПЛЕНИЕ

Задача отыскания экстремальных независимых и покрывающих множеств возникает во многих практических случаях. Например, пусть дано множество процессов, использующих неразделяемые ресурсы. Соединим рёбрами вершины, соответствующие процессам, которым требуется один и тот же ресурс. Тогда β_0 будет определять количество возможных параллельных процессов.

10.4.3. Оценка числа вершинной независимости

ТЕОРЕМА. *Для любого графа $G(V,E)$ справедливо неравенство:*

$$\beta_0(G) \geqslant \sum_{v\in V} \frac{1}{1+d(v)}.$$

Доказательство. Заметим, что если $G = K_p$, то $\beta_0(K_p) = 1$ и

$$\sum_{v\in V} \frac{1}{1+d(v)} = \sum_{v\in V} \frac{1}{1+p-1} = 1,$$

то есть достигается равенство. Далее индукция по p. База для $p = 1$ проверена. Ввиду того, что для полных графов достигается равенство, не ограничивая общности, можно считать, что $G \neq K_p$. Пусть теперь $p > 2$ и для всех графов с числом вершин меньше p неравенство выполнено. Рассмотрим в графе G вершину u минимальной степени: $d(u) = \delta(G)$. Так как $G \neq K_p$, имеем $\Gamma^*(u) \neq V$. Удалим вершину u и её окрестность из множества вершин $V' := V \setminus \Gamma^*(u)$, и рассмотрим граф G', который является правильным подграфом графа G, определяемым множеством V'. Степень вершины в графе G' обозначим d'. Заметим, что $\forall\, v \in V'\ \ (d'(v) \leqslant d(v))$, и поэтому $\forall\, v \in V'\ \ \big((1+d'(v))^{-1} \geqslant (1+d(v))^{-1}\big)$. Далее, если M — независимое множество вершин в графе G', то $M + u$ — независимое множество вершин в графе G, а потому $\beta_0(G) \geqslant \beta_0(G') + 1$. Заметим теперь, что в силу выбора вершины u имеем $\sum_{v\in\Gamma^*(u)}(1+d(v))^{-1} \leqslant \sum_{v\in\Gamma^*(u)}(1+d(u))^{-1} = 1$. Получаем

$$\beta_0(G) \geqslant \beta_0(G') + 1 \geqslant \sum_{v\in V'} (1+d'(v))^{-1} + 1 \geqslant \sum_{v\in V'} (1+d(v))^{-1} + 1 \geqslant$$
$$\geqslant \sum_{v\in V'} (1+d(v))^{-1} + \sum_{v\in\Gamma^*(u)} (1+d(v))^{-1} \geqslant \sum_{v\in V} (1+d(v))^{-1}.$$

□

Рассмотрим *среднюю степень* графа: $\bar{d}(G) \overset{\text{Def}}{=} \dfrac{1}{p}\displaystyle\sum_{v\in V} d(v)$.

СЛЕДСТВИЕ. *Для любого графа $G(V,E)$ справедливо неравенство:*

$$\beta_0(G) \geqslant \frac{p}{1+\bar{d}}.$$

Доказательство. Известно *неравенство Коши–Буняковского* $\left(\displaystyle\sum_{i=1}^{n} a_i b_i\right)^2 \leqslant \left(\sum_{i=1}^{n} a_i^2\right)\left(\sum_{i=1}^{n} b_i^2\right)$. Положим $a_i := \sqrt{(1+d(v_i))^{-1}}$, $b_i := \sqrt{1+d(v_i)}$.
Тогда $p^2 = \left(\displaystyle\sum_{i=1}^{p} \sqrt{\frac{1+d(v_i)}{1+d(v_i)}}\right)^2 \leqslant \beta_0 \left(\sum_{i=1}^{n}(1+d(v_i))\right)$, и $p^2 \leqslant \beta_0(p + p\bar{d})$,
то есть $p \leqslant \beta_0(1+\bar{d})$. □

10.5. Построение независимых множеств вершин

Этот раздел фактически является вводным обзором методов решения переборных задач. Методы рассматриваются на примере задачи отыскания максимального независимого множества вершин графа. Наш обзор не претендует на полноту, но описание основополагающих идей и терминов в нём присутствует.

10.5.1. Постановка задачи отыскания наибольшего независимого множества вершин

Задача отыскания наибольшего независимого множества вершин (и тем самым определения вершинного числа независимости β_0) принадлежит к числу трудоёмких. Эту задачу можно поставить следующим образом. Пусть задан граф $G(V, E)$. Найти такое множество вершин X, $X \subset V$, что

$$|X| = \max_{Y \in \mathcal{E}} |Y|, \quad \mathcal{E} \stackrel{\text{Def}}{=} \{Y \subset V \mid \forall\, u, v \in Y \;\; ((u, v) \notin E)\}.$$

Если бы семейство $\mathcal{E}$ независимых множеств оказалось матроидом, то поставленную задачу можно было бы решить жадным алгоритмом. Однако семейство $\mathcal{E}$ матроидом не является. Действительно, хотя аксиомы M_1 и M_2 (п. 2.6.1) выполнены, так как пустое множество вершин независимо и всякое подмножество независимого множества независимо, но аксиома M_3 может быть не выполнена, как видно из следующего примера.

Пример. Рассмотрим полный двудольный граф $K_{n,n+1}$. Доли этого графа образуют независимые множества, и их мощности различаются на 1. Однако никакая вершина из большей доли не может быть добавлена к вершинам меньшей доли с сохранением независимости.

Нижняя оценка числа вершинной независимости β_0, полученная в п. 10.4.3 оказывается точным значением для крайних случаев — для полных и для пустых графов. На первый взгляд кажется разумным предположить, что эта оценка «достаточно точна» и для других графов. В таком случае, на основе конструктивного доказательства нижней оценки, возникает соблазн предложить эффективный *эвристический* алгоритм поиска независимого множества, «близкого» к максимальному.

Вход: граф $G(V, E)$.
Выход: наибольшее независимое множество X.

```
X := ∅  // вначале независимое множество пусто.
while G ≠ ∅ do
  select u ∈ V & d(u) = δ(G)  // u — вершина минимальной степени
  X := X + u  // помещаем её в независимое множество
  G := G \ Γ*(u)  // удаляем вершину u вместе с её окрестностью
end while
```

Пример. На рис. 10.10 слева приведена диаграмма графа, для которого эвристический алгоритм находит точное решение $\beta_0 = 2$, а справа приведена диаграмма графа, для которого алгоритм даёт ответ 2, в то время, как максимальным независимым множеством является доля в центре и $\beta_0 = 3$.

Более того, приведённый пример легко обобщить. Рассмотрим граф $\overline{K}_m + K_n$ и добавим к нему одну вершину, соединив её с вершинами доли $\overline{K}_m$. Тогда при $n \geqslant m > 2$

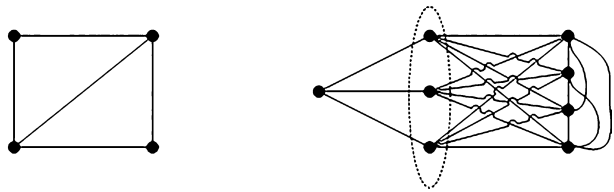


Рис. 10.10. Диаграммы некоторых графов

наибольшим независимым множеством является средняя доля $\overline{K}_m$ и $\beta_0 = m$, в то время как эвристический алгоритм по-прежнему выдаёт ответ 2. Таким образом, эвристический алгоритм может дать ответ, сколь угодно далёкий от точного решения.

10.5.2. Поиск с возвратами

Даже если для решения задач, подобных поставленной в предыдущем параграфе, не удаётся найти эффективного алгоритма, остаётся возможность попробовать найти решение «полным перебором» всех возможных вариантов, просто в силу конечности числа возможностей. Например, наибольшее независимое множество можно найти по следующей схеме.

Вход: граф $G(V, E)$.
Выход: наибольшее независимое множество X.

```
m := 0  // наилучшее известное значение β0
for Y ∈ 2^V do
  if Y ∈ 𝓔 & |Y| > m then
    m := |Y|; X := Y  // наилучшее известное значение X
  end if
end for
```

ЗАМЕЧАНИЕ

Для выполнения этого алгоритма потребуется $O(2^p)$ шагов.

ОТСТУПЛЕНИЕ

Алгоритм, трудоёмкость которого (число шагов) ограничена полиномом от характерного размера задачи, принято называть *эффективным*, в противоположность *неэффективным* алгоритмам, трудоёмкость которых ограничена функцией, растущей быстрее, например, экспонентой. Таким образом, жадный алгоритм эффективен, а полный перебор — нет.

При решении переборных задач большое значение имеет способ организации перебора (в нашем случае — способ построения и последовательность перечисления множеств Y). Наиболее популярным является следующий способ организации перебора, основанный на идее поиска в глубину и называемый *поиском с возвратами*.

ЗАМЕЧАНИЕ

Иногда употребляется термин «*бэктрекинг*» (транслитерация английского названия этого метода — *backtracking*). Буквальный перевод английского названия — обратное прослеживание — явно неудачен, поскольку в методе нет ничего «обратного» и непонятно, что «прослеживается».

Идея поиска с возвратами состоит в следующем. Находясь в некоторой ситуации, пробуем изменить её допустимым образом в надежде найти решение. Если изменение не привело к успеху, то возвращаемся в исходную ситуацию (отсюда название «поиск с возвратами») и пробуем изменить её другим образом, и так до тех пор, пока не будут перебраны все возможности.

Для рассматриваемой задачи отыскания наибольшего независимого множества вершин метод поиска с возвратами может быть реализован с помощью следующего рекурсивного алгоритма.

Алгоритм 10.2. Поиск с возвратами

Вход: граф $G(V, E)$.
Выход: наибольшее независимое множество X.

```
m := 0  // наилучшее известное значение β0
X := ∅  // наибольшее известное независимое множество X
BT(∅, V)  // вызов рекурсивной процедуры BT
```

Алгоритм 10.3. Рекурсивная процедура BT

Вход: S — текущее независимое множество вершин, T — оставшиеся вершины графа.
Выход: изменение глобальной переменной X, если текущее множество не может быть расширено (является максимальным).

```
e := false  // признак расширяемости множества
if |S| > m then
   X := S; m := |S|  // наибольшее известное независимое множество
end if
for v ∈ T do
   if S + v ∈ E then
      BT(S + v, T \ Γ*(v))  // пробуем добавить v
   end if
end for
```

Обоснование. По построению вершина v добавляется в множество S только при сохранении независимости расширенного множества. В алгоритме это обстоятельство указано в форме условия $S + v \in \mathcal{E}$. Проверить сохранение условия независимости нетрудно, например, с помощью следующей функции.

Вход: независимое множество S и проверяемая вершина v.
Выход: **true**, если множество $S + v$ независимое, **false** — в противном случае.

```
for u ∈ S do
   if (u, v) ∈ E then
      return false  // множество S + v зависимое
   end if
end for
return true  // множество S + v независимое
```

Этот цикл не включен в явном виде в рекурсивную процедуру BT, чтобы не загромождать основной текст и не затуманивать идею поиска с возвратами. Таким образом, множество S, а следовательно, и множество X — независимые. В тот

момент, когда множество S нельзя расширить, оно максимально по определению. Переменная m глобальна, поэтому среди всех максимальных независимых множеств в конце работы алгоритма построенное множество X является наибольшим независимым множеством вершин. □

10.5.3. Улучшенный перебор

Применение метода поиска с возвратами не гарантирует эффективности — трудоёмкость поиска с возвратами имеет тот же порядок, что и другие способы перебора (в худшем случае).

Используя конкретную информацию о задаче, в некоторых случаях можно существенно сократить трудоёмкость выполнения каждого шага перебора или уменьшить количество перебираемых возможностей в среднем (при сохранении оценки количества шагов в худшем случае). Такие приёмы называются методами *улучшения перебора*. Например, может оказаться, что некоторые варианты заведомо не могут привести к решению, а потому их можно не рассматривать.

ЗАМЕЧАНИЕ

Рекурсивная форма метода поиска с возвратами удобна для понимания, но не является самой эффективной. По сути, рекурсия здесь используется для сохранения *контекста* — то есть информации, характеризующей текущий рассматриваемый вариант. Если использовать другие методы сохранения контекста, то поиск с возвратами может быть реализован без явной рекурсии, а значит, более эффективно.

Рассмотрим методы улучшения перебора на примере задачи отыскания всех максимальных независимых множеств вершин. Идея: начинаем с пустого множества и пополняем его вершинами с сохранением независимости (пока возможно).

Пусть S_k — уже полученное множество из k вершин, Q_k — множество вершин, которое можно добавить к S_k, то есть $S_k \cap \Gamma(Q_k) = \varnothing$. Среди вершин Q_k будем различать те, которые уже использовались для расширения S_k (обозначим их множество Q_k^-), и те, которые еще не использовались (Q_k^+). Тогда общая схема нерекурсивной реализации поиска с возвратами будет состоять из следующих шагов.

Шаг вперед от k к $k+1$ состоит в выборе вершины $x \in Q_k^+$:

$$\begin{aligned} S_{k+1} &:= S_k + x, \\ Q_{k+1}^- &:= Q_k^- \setminus \Gamma^+(x), \\ Q_{k+1}^+ &:= Q_k^+ \setminus \Gamma^*(x). \end{aligned}$$

Шаг назад от $k+1$ к k:

$$\begin{aligned} S_k &:= S_{k+1} - x, \\ Q_k^+ &:= Q_k^+ - x, \\ Q_k^- &:= Q_k^- + x. \end{aligned}$$

Если S_k — максимальное, то $Q_k^+ = \varnothing$. Если $Q_k^- \neq \varnothing$, то S_k было расширено раньше и не является максимальным. Таким образом, проверка максимальности задаётся следующим условием: $Q_k^+ = Q_k^- = \varnothing$.

Перебор можно улучшить, если заметить следующее.

Пусть $x \in Q_k^-$ и $\Gamma(x) \cap Q_k^+ = \varnothing$. Эту вершину x никогда не удалить из Q_k^-, так как из Q_k^- удаляются только вершины, смежные с Q_k^+. Таким образом, существование x, такого, что $x \in Q_k^-$ и $\Gamma(x) \cap Q_k^+ = \varnothing$, является достаточным условием для возвращения. Кроме того, $k \leqslant p - 1$.

10.5.4. Алгоритм построения независимых множеств

Приведённый алгоритм 10.4, обоснование которого дано в предыдущем параграфе, строит все максимальные независимые множества вершин заданного графа.

Алгоритм 10.4. Построение максимальных независимых множеств

Вход: граф $G(V, E)$, заданный списками смежности $\Gamma[v]$.
Выход: последовательность максимальных независимых множеств.

$k := 0$ // количество элементов в текущем независимом множестве
$S[k] := \varnothing$ // $S[k]$ — независимое множество из k вершин
$Q^-[k] := \varnothing$ //$Q^-[k]$ — вершины, уже использованные для расширения $S[k]$
$Q^+[k] := V$ // $Q^+[k]$ — вершины, ещё не использованные для расширения $S[k]$
$M1:$ // шаг вперед
select $v \in Q^+[k]$ // расширяющая вершина
$X[k] := v$ // запоминаем расширяющую вершину
$S[k+1] := S[k] \cup \{v\}$ // расширенное множество
$Q^-[k+1] := Q^-[k] \setminus \Gamma[v]$ // вершина v использована для расширения
$Q^+[k+1] := Q^+[k] \setminus (\Gamma[v] \cup \{v\})$ // не могут быть использованы для расширения
$k := k + 1$
$M2:$ // проверка
for $u \in Q^-[k]$ **do**
  if $\Gamma[u] \cap Q^+[k] = \varnothing$ **then goto** $M3$ **end if** // можно возвращаться
end for
if $Q^+[k] = \varnothing$ **then**
  if $Q^-[k] = \varnothing$ **then yield** $S[k]$ **end if** // множество $S[k]$ максимально
  goto $M3$ // можно возвращаться
else
  goto $M1$ // можно идти вперед
end if
$M3:$ // шаг назад
$k := k - 1$
$S[k] := S[k+1] - X[k+1]$
$Q^-[k] := Q^-[k] + X[k+1]$ // вершина v уже добавлялась
$Q^+[k] := Q^+[k] - X[k+1]$
if $k = 0$ & $Q^+[k] = \varnothing$ **then**
  stop // перебор завершён
else
  goto $M2$ // переход на проверку
end if

Пример. Известная *задача о восьми ферзях* (расставить на шахматной доске 8 ферзей так, чтобы они не били друг друга) является задачей об отыскании максимальных независимых множеств. Действительно, достаточно представить доску в виде графа с 64 вершинами (соответствующими клеткам доски), которые смежны, если клетки находятся на одной вертикали, горизонтали или диагонали.

10.6. Доминирующие множества

Задача о наименьшем покрытии (сокращённо ЗНП) является примером общей экстремальной задачи, к которой прямо или косвенно сводятся многие практические задачи. Эта задача является классической, хорошо изучена и часто используется в качестве теста для сравнения и оценки различных общих методов решения трудоёмких задач.

В этом разделе на основе рассмотрения понятия доминирующего множества ЗНП формулируются и приводятся сведения о связи ЗНП с другими задачами.

10.6.1. Минимальное и наименьшее доминирующее множество

Множество вершин $S \subset V$ графа $G(V, E)$ называется *доминирующим множеством* (или *внешне устойчивым*), если $S \cup \Gamma(S) = V$, то есть

$$\forall\, v \in V \;\; (v \in S \vee \exists\, s \in S \;\; ((s, v) \in E)) .$$

Очевидно, что множество S доминирует тогда и только тогда, когда

$$\forall\, v \notin S \;\; (\Gamma(v) \cap S \neq \varnothing) ,$$

что равносильно утверждению $\forall\, v \in V \;\; (\exists\, s \in S \;\; (d(v, s) \leqslant 1))$.

Доминирующее множество называется *минимальным*, если никакое его подмножество не является доминирующим. Доминирующее множество называется *наименьшим*, если число элементов в нём наименьшее возможное.

Пример. Известная *задача о пяти ферзях* (расставить на шахматной доске 5 ферзей так, чтобы они били всю доску) является задачей об отыскании наименьших доминирующих множеств.

10.6.2. Доминирование и независимость

Доминирование тесно связано с вершинной независимостью.

ТЕОРЕМА. *Независимое множество вершин является максимальным тогда и только тогда, когда оно является доминирующим.*

Доказательство.

[$\Longrightarrow$] Пусть множество вершин $S \subset V$ — максимальное независимое. Допустим (от противного), что оно не доминирующее. Тогда существует вершина v, находящаяся на расстоянии больше 1 от всех вершин множества S. Эту вершину можно добавить к S с сохранением независимости, что противоречит максимальности.

[$\Longleftarrow$] Пусть S — независимое доминирующее множество. Допустим (от противного), что оно не максимальное. Тогда существует вершина v, не смежная ни с одной из вершин множества S, то есть находящаяся на расстоянии больше 1 от всех вершин множества S. Это противоречит тому, что множество S — доминирующее. □

Независимое доминирующее множество вершин называется *ядром* графа.

Пример. В полном графе K_p каждая из p вершин является ядром и других ядер нет.

СЛЕДСТВИЕ. *Любой граф имеет ядро.*

Доказательство. Следующий простой алгоритм, основанный на доказательстве предыдущей теоремы, строит некоторое ядро S.

$S := \varnothing$ // ядро
while $V \neq \varnothing$ **do**
  select $v \in V$ // любая нерассмотренная вершина
  $S := S + v$ // расширяем множество S
  $V := V \setminus \Gamma^*(v)$ // удаляем вершины, которые не могут быть использованы для расширения
end while □

Понятия независимости, доминирования и ядра применимы как к графам, так и к орграфам. Множество узлов S орграфа называется *независимым*, если $\forall\, v \in S \;\; (\Gamma(v) \cap S = \varnothing)$, и называется *доминирующим*, если $\forall\, v \notin S \;\; (\Gamma(v) \cap S \neq \varnothing)$. Независимое доминирующее множество узлов в орграфе называется *ядром*.

ЗАМЕЧАНИЕ

Существуют орграфы, не имеющее ядра. Например, контур C_3 не имеет ядра. Более того, задача выделения ядра в произвольном орграфе оказывается *NP*-полной.

10.6.3. Задача о наименьшем покрытии

Рассмотрим следующую задачу. Пусть каждой вершине сопоставлена некоторая цена. Требуется выбрать доминирующее множество с наименьшей суммарной ценой. Эта задача называется *задачей о наименьшем покрытии*. ЗНП является весьма общей задачей, к которой сводятся многие другие задачи.

Задача о выборе переводчиков. Организации нужно нанять переводчиков для перевода с определённого множества языков. Каждый из имеющихся переводчиков владеет некоторыми иностранными языками и требует определённую зарплату. Требуется определить, каких переводчиков следует нанять, чтобы сумма расходов на зарплату была минимальной. Задача о выборе переводчиков сводится к ЗНП следующим образом. Рассмотрим двудольный граф с долями V_1 и V_2, где доля V_1 соответствует множеству переводчиков, а доля V_2 — множеству языков. Вершины $v_1 \in V_1$ и $v_2 \in V_2$ смежны, если переводчик v_1 владеет языком v_2. Требуется найти наименьшее доминирующее множество такое, что $S \subset V_1$.

Задача о развозке. Поставщику нужно доставить товары своим потребителям. Имеется множество возможных маршрутов, каждый из которых позволяет обслужить определённое подмножество потребителей и требует определённых расходов. Требуется определить, какие маршруты следует использовать, чтобы все потребители были обслужены, а сумма транспортных расходов была минимальной. Задача о развозке сводится к ЗНП аналогичным образом. Здесь V_1 — множество маршрутов, V_2 — множество потребителей и вершины $v_1 \in V_1$ и $v_2 \in V_2$ смежны, если маршрут v_1 обслуживает потребителя v_2.

10.6.4. Связь задачи о наименьшем покрытии с другими задачами

ЗНП может быть сформулирована многими разными способами.

Пример. Пусть имеется конечное множество $V = \{v_1, \ldots, v_p\}$ и семейство подмножеств этого множества $E = \{E_1, \ldots, E_p\}$. Каждому подмножеству E_i приписан вес. Найти покрытие E' ($E' \subset E$) наименьшего веса. На языке графов v_i — это вершина, а E_i — множество смежности вершины, то есть $E_i = \Gamma^*(v_i)$.

ЗАМЕЧАНИЕ

Из этой формулировки происходит название «задача о наименьшем покрытии».

Известно, что ЗНП относится к числу трудоёмких задач и для её решения применяются переборные алгоритмы с теми или иными улучшениями.

На рис. 10.11 приведена схема (заимствованная из книги [13]), показывающая связь ЗНП и некоторых других задач. На этой схеме стрелка от задачи A к задаче B означает, что решение задачи A влечет за собой решение задачи B.

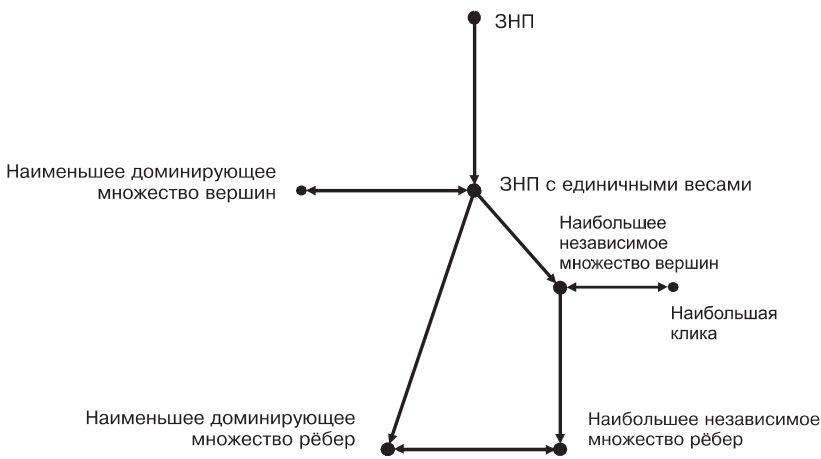


Рис. 10.11. Связь различных задач

10.7. Раскраска графов

Задача раскрашивания графов, которая на первый взгляд кажется просто праздной головоломкой, имеет неожиданно широкое применение в программировании, особенно при решении фундаментальных теоретических проблем (см., например, книгу [5]).

10.7.1. Оценки хроматического числа

Раскраской графа G называется такое приписывание цветов (натуральных чисел) его вершинам, что никакие две смежные вершины не получают одинаковый цвет. Если задана допустимая раскраска графа, использующая m цветов, то говорят, что граф m-раскрашиваемый. Наименьшее возможное количество цветов в раскраске называется *хроматическим числом* и обозначается $\chi(G)$.

Примеры

$\chi(\overline{K}_p) = 1$, $\chi(K_p) = p$, $\chi(K_{m,n}) = 2$, $\chi(C_{2n}) = 2$, $\chi(C_{2n+1}) = 3$, $\chi(T) = 2$, где T — свободное дерево.

Очевидно, что существует m-раскраска графа G для любого m в диапазоне $\chi(G) \leqslant m \leqslant p$. Множество вершин, покрашенных в один цвет, называется *одноцветным классом*. Одноцветные классы образуют независимые множества вершин, то есть никакие две вершины в одноцветном классе не смежны.

Способ явного выражения хроматического числа через другие стандартные инварианты графа неизвестен. Известны только некоторые оценки, часть из которых приведена ниже.

ТЕОРЕМА 1. $\chi(G) \leqslant 1 + \Delta(G)$.

Доказательство. Индукция по p. База: $p = 1 \Longrightarrow \chi(G) = 1 \;\&\; \Delta(G) = 0$. Пусть $\forall G \;\; (p(G) < p \Longrightarrow \chi(G) \leqslant \Delta(G) + 1)$. Рассмотрим граф G, такой, что $p(G) = p$. Тогда $\forall v \in V \;\; (\chi(G - v) \leqslant \Delta(G - v) + 1 \leqslant \Delta(G) + 1)$. Но $d(v) \leqslant \Delta(G)$, значит, хотя бы один цвет в $(\Delta(G) + 1)$-раскраске графа $G - v$ свободен для v. Покрасив вершину v в этот цвет, получаем $(\Delta(G) + 1)$-раскраску графа G. □

ТЕОРЕМА 2. $p/\beta_0(G) \leqslant \chi(G) \leqslant p - \beta_0(G) + 1$.

Доказательство.

[$p/\beta_0(G) \leqslant \chi(G)$] Пусть $\chi(G) = n$ и $V = V_1 \cup \cdots \cup V_n$, где V_i — одноцветные классы. V_i — независимое множество вершин, следовательно, $|V_i| \leqslant \beta_0(G)$. Имеем $p = \sum_{i=1}^{n} |V_i| \leqslant n\beta_0(G) \Longrightarrow p/\beta_0 \leqslant \chi$.

[$\chi(G) \leqslant p - \beta_0(G) + 1$] Пусть $S \subset V$ — наибольшее независимое множество, $|S| = \beta_0(G)$. Тогда $\chi(G - S) \leqslant |V - S| = p - \beta_0(G)$. Из n-раскраски графа $G - S$ можно получить $(n+1)$-раскраску графа G, так как все вершины из S можно покрасить в один новый цвет. Следовательно, $\chi(G) = \chi(G - S) + 1 \leqslant p - \beta_0 + 1$. □

10.7.2. Хроматические числа графа и его дополнения

Хроматические числа графа G и его дополнения $\overline{G}$ связаны: если в G сравнительно мало рёбер, то и $\chi(G)$ будет невелико, но тогда в $\overline{G}$ — много рёбер и $\chi(\overline{G})$ будет близко к p.

ТЕОРЕМА. *Пусть* $\chi := \chi(G)$, $\overline{\chi} := \chi(\overline{G})$. *Тогда*

$$2\sqrt{p} \leqslant \chi + \overline{\chi} \leqslant p + 1, \quad p \leqslant \chi\overline{\chi} \leqslant \left(\frac{p+1}{2}\right)^2.$$

Доказательство.

[$p \leqslant \chi\overline{\chi}$] Пусть $\chi(G) = n$, $V_1, \ldots, V_n$ — одноцветные классы, $p_i := |V_i|$. Тогда $\sum\limits_{i=1}^{n} p_i = p$, следовательно, $\max\limits_{i=1}^{n} p_i \geqslant p/n$. Но V_i — независимые множества в G, следовательно, V_i — клики в $\overline{G}$. Значит, $\overline{\chi} \geqslant \max\limits_{i=1}^{n} p_i \geqslant p/n$. Имеем $\chi\overline{\chi} \geqslant n \cdot p/n = p$.

[$2\sqrt{p} \leqslant \chi + \overline{\chi}$] Известно, что среднее геометрическое двух чисел не превосходит среднего арифметического: $(a + b)/2 \geqslant \sqrt{ab}$. Следовательно, $\chi + \overline{\chi} \geqslant 2\sqrt{\chi\overline{\chi}} \geqslant 2\sqrt{p}$.

[$\chi + \overline{\chi} \leqslant p + 1$] Индукция по p. База: $p = 1 \Longrightarrow \chi = 1 \;\&\; \overline{\chi} = 1$. Пусть $\chi + \overline{\chi} \leqslant p$ для всех графов с $p - 1$ вершинами. Рассмотрим граф G с p вершинами и вершину $v \in V$. Тогда, очевидно, $\chi(G) \leqslant \chi(G - v) + 1$ и $\chi(\overline{G}) \leqslant \chi(\overline{G} - v) + 1$.

Если $\chi(G) < \chi(G-v)+1$ или $\chi(\overline{G}) < \chi(\overline{G}-v)+1$, то $\chi+\overline{\chi} = \chi(G)+\chi(\overline{G}) < \chi(G-v)+1+\chi(\overline{G}-v)+1 < p+2 \leqslant p+1$. Следовательно, $\chi+\overline{\chi} \leqslant p+1$. Пусть теперь $\chi(G)=\chi(G-v)+1$ и $\chi(\overline{G})=\chi(\overline{G}-v)+1$. Положим $d:=d(v)$ в графе G, тогда $\overline{d} = p-d-1$ — степень вершины v в графе $\overline{G}$. Имеем $d \geqslant \chi(G-v)$. Действительно, $\chi(G)=\chi(G-v)+1$, и если бы $d<\chi(G-v)$, то вершину v можно было бы покрасить в любой из свободных $\chi(G-v)-d$ цветов и получить $\chi(G-v)$-раскраску графа G. Аналогично, $\overline{d} = p-d-1 \geqslant \chi(\overline{G}-v)$. Таким образом,

$$\chi+\overline{\chi} = \chi(G)+\chi(\overline{G}) = \chi(G-v)+1+\chi(\overline{G}-v)+1 \leqslant d+1+p-d-1+1 = p+1.$$

[$\chi\overline{\chi} \leqslant ((p+1)/2)^2$] Имеем $2\sqrt{\chi\overline{\chi}} \leqslant \chi+\overline{\chi} \leqslant p+1 \Longrightarrow ((p+1)/2)^2 \geqslant \chi\overline{\chi}$. □

10.7.3. Точный алгоритм раскрашивания

Поскольку точная формула для хроматического числа неизвестна, задача нахождения наилучшей раскраски графа оказывается, как и следовало ожидать, труднорешаемой.

Рассмотрим следующую схему рекурсивной процедуры P:

1. Выбрать в графе G некоторое максимальное независимое множество вершин S.
2. Покрасить вершины множества S в очередной цвет.
3. Применить процедуру P к графу $G-S$.

На псевдокоде процедура P может быть записана следующим образом.

Вход: граф $G(V,E)$, номер свободного цвета i.

Выход: раскраска, заданная массивом $C[V]$, — номера цветов, приписанные вершинам.

if $V=\varnothing$ **then return end if** // раскраска закончена
$S:=\mathrm{Selectmax}(G)$ //S — максимальное независимое множество
$C[S]:=i$ // раскрашиваем вершины множества S в цвет i
$P(G-S,i+1)$ // рекурсивный вызов

ЗАМЕЧАНИЕ

Функция Selectmax может быть реализована, например, на основе алгоритма 10.5.4.

ТЕОРЕМА. *Если граф G — k-раскрашиваемый, то существует такая последовательность выборов множества S на шаге 1 процедуры P, что применение процедуры P к графу G построит не более чем k-раскраску графа G.*

Доказательство. Пусть имеется некоторая k-раскраска графа $G(V,E)$. Перестроим её в такую не более чем k-раскраску, которая может быть получена процедурой P. Пусть $V_1 \subset V$ — множество вершин в данной k-раскраске, покрашенных в цвет 1. Множество V_1 — независимое, но, может быть, не максимальное. Рассмотрим множество $V_1{}'$, такое, что $V_1 \cup V_1{}'$ — максимальное независимое множество (может оказаться, что $V_1{}'=\varnothing$). Вершины из $V_1{}'$ не смежны с V_1, значит, вершины из $V_1{}'$ можно перекрасить в цвет 1. Пусть далее $V_2 \subset V \setminus (V_1 \cup V_1{}')$ — множество вершин, покрашенных в цвет 2. Аналогично рассмотрим множество $V_2{}'$, такое, что $V_2 \cup V_2{}'$ — максимальное независимое в $G \setminus (V_1 \cup V_1{}')$, покрасим вершины $V_2 \cup V_2{}'$ в цвет 2 и т. д. Всего в исходной раскраске было k независимых множеств. При перекраске их число не возрастёт (но может уменьшиться, если $\chi(G)<k$). На каждом шаге

алгоритма рассматривается одно из множеств, следовательно, процесс закончится. □

10.7.4. Приближённые алгоритмы раскрашивания

В предыдущем параграфе некоторый алгоритм точного раскрашивания был построен на основе алгоритма выделения максимальных независимых множеств вершин, который имеет переборный характер. Таким образом, предложенный алгоритм точного раскрашивания также имеет переборный характер. Можно показать, что это не случайно и задача построения минимальной раскраски является NP-полной.

При практическом решении NP-полных задач целесообразно рассматривать *приближённые* алгоритмы, которые не всегда находят точное решение задачи (иногда они находят только приближение к нему, и мы не можем знать этого заранее), но зато достаточно эффективны.

Здесь понятие приближённого алгоритма имеет более широкий смысл по сравнению с тем, который обычно подразумевается в вычислительной математике и при проведении численных расчётов на компьютере. В вычислительной математике приближённый алгоритм, например, численное решение уравнения, находит приближение к корню с заданной точностью: $x \pm \delta$. Увеличивая число итераций, можно уменьшать δ.

Приближённые алгоритмы решения переборных задач находят решение с той точностью, с какой умеют — дополнительные вычисления не улучшают решения. Поэтому приближенные алгоритмы решения переборных задач характеризуются двумя параметрами: насколько далеко найденное решение может отстоять от точного решения, и насколько меньшие трудозатраты по сравнению с полным перебором при этом требуются.

Рассмотрим следующий *алгоритм последовательного раскрашивания*.

Алгоритм 10.5. Алгоритм последовательного раскрашивания

Вход: граф G.
Выход: раскраска графа — массив C : **array** $[1..p]$ **of** $1..p$.

```
for v ∈ V do
    C[v] := 0  // все вершины не раскрашены
end for
for v ∈ V do
    A := {1, ..., p}  // все цвета
    for u ∈ Γ⁺(v) do
        A := A \ {C[u]}  // занятые для вершины v цвета
    end for
    C[v] := min A  // минимальный свободный цвет
end for
```

ЗАМЕЧАНИЕ

Таким образом, красить вершины необходимо последовательно, выбирая среди допустимых цветов минимальный.

Обоснование. В основном цикле рассматриваются все вершины, и каждая из них получает допустимую раскраску. Таким образом, процедура строит допустимую раскраску. □

Улучшенный алгоритм последовательного раскрашивания также строит допустимую раскраску, применяя такую эвристику: начинать раскрашивать следует с вершин наибольшей степени, поскольку если их раскрашивать в конце процесса, то более вероятно, что для них не найдётся свободного цвета и придётся использовать еще один цвет.

Алгоритм 10.6. Улучшенный алгоритм последовательного раскрашивания

Вход: граф G.
Выход: раскраска графа — массив C : **array** $[1..p]$ **of** $1..p$.

```
Sort(V)  // упорядочить вершины по невозрастанию степени
c := 1  // первый цвет
for v ∈ V do
  C[v] := 0  // все не раскрашены
end for
while V ≠ ∅ do
  for v ∈ V do
    for u ∈ Γ+(v) do
      if C[u] = c then
        next for v  // вершину v нельзя покрасить в цвет c
      end if
    end for
    C[v] := c  // красим вершину v в цвет c
    V := V \ {v}  // и удаляем её из рассмотрения
  end for
  c := c + 1  // следующий цвет
end while
```

Обоснование. Заметим, что данный алгоритм отличается от предыдущего тем, что основной цикл идет не по вершинам, а по цветам: сначала всё, что можно, красим в цвет 1, затем в оставшемся красим всё, что можно, в цвет 2 и т. д. В остальном алгоритмы аналогичны, и данный алгоритм заканчивает свою работу построением допустимой раскраски по тем же причинам, что и предыдущий. □

ОТСТУПЛЕНИЕ

Улучшенный алгоритм последовательного раскрашивания несколько сложнее первого алгоритма и основан на остроумной эвристике. Можно было бы ожидать, что он даст существенно лучшие результаты. Однако прямые вычислительные эксперименты показывают, что эти алгоритмы почти во всех случаях работают одинаково хорошо (или плохо). Таким образом, программистские «хитрости» далеко не всегда дают практически значимые результаты.

10.8. Планарность

Обсуждение планарности в этом разделе позволяет решить вторую историческую задачу из перечисленных в п. 7.1.1, а также подготавливает результаты, необходимые для доказательства теоремы о пяти красках.

10.8.1. Укладка графов

Граф *укладывается* на некоторой поверхности, если его можно нарисовать на этой поверхности так, чтобы рёбра графа при этом не пересекались. Граф называется *планарным*, если его можно уложить на плоскости. *Плоский* граф — это граф, уже уложенный на плоскости.

Область, ограниченная ребрами в плоском графе, называется *гранью*. Грань не содержит других граней. Число граней плоского графа G обозначается $f(G)$.

ЗАМЕЧАНИЕ

Внешняя часть плоскости также образует грань.

Пример. На рис. 10.12 показаны диаграммы планарного графа K_4 и две его укладки на плоскости. Этот граф имеет 4 грани.

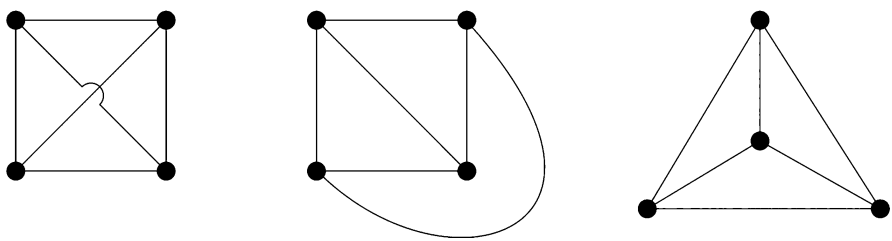


Рис. 10.12. Планарный граф и его укладка

ОТСТУПЛЕНИЕ

Точное определение некоторых понятий, используемых в этом разделе, в частности, таких понятий, как «поверхность», «область», «граница», выходит далеко за рамки этого учебника и стандартного курса дискретной математики. Мы полагаемся на геометрическую интуицию читателя и не даём никаких определений. Для понимания простейших рассуждений этого раздела достаточно неформальных интуитивных представлений. Однако читателю следует иметь в виду, что при решении сложных практических задач, например из вычислительной геометрии, интуитивных представлений может оказаться недостаточно.

10.8.2. Эйлерова характеристика

Для графов, уложенных на некоторой поверхности, справедливо определённое соотношение между числом вершин, рёбер и граней графов, которые укладываются на этой поверхности.

ТЕОРЕМА (формула Эйлера). *Для связного планарного графа справедливо следующее соотношение:* $p - q + f = 2$.

ЗАМЕЧАНИЕ

Число в правой части этого соотношения называется *эйлеровой характеристикой* поверхности.

Доказательство. Индукция по q. База: $q = 0 \Longrightarrow p = 1 \;\&\; f = 1$. Пусть теорема верна для всех графов с q рёбрами: $p - q + f = 2$. Добавим еще одно ребро. Если добавляемое ребро соединяет существующие вершины, то легко видеть, что $q' = q+1$, $p' = p$, $f' = f + 1$ и $p' - q' + f' = p - q - 1 + f + 1 = p - q + f = 2$. Если добавляемое ребро соединяет существующую вершину с новой, то $p' = p + 1$, $q' = q + 1$, $f' = f$ и $p' - q' + f' = p + 1 - q - 1 + r = p - q + f = 2$. □

СЛЕДСТВИЕ 1. *Если G — связный планарный граф ($p > 3$), то $q \leqslant 3p - 6$.*

Доказательство. Каждая грань ограничена по крайней мере тремя ребрами, каждое ребро ограничивает не более двух граней, отсюда $3f \leqslant 2q$. Имеем
$2 = p - q + f \leqslant p - q + \frac{2q}{3} \implies 3p - 3q + 2q \geqslant 6 \implies q \leqslant 3p - 6$. □

СЛЕДСТВИЕ 2. *Графы K_5 и $K_{3,3}$ не планарны.*

Доказательство.

[K_5 не планарен] Имеем $p = 5$, $q = p(p-1)/2 = 10$. Если K_5 планарен, то по предыдущему следствию $q = 10 \leqslant 3p - 6 = 3 \cdot 5 - 6 = 9$ — противоречие.

[$K_{3,3}$ не планарен] Имеем $p = 6$, $q = 9$. В этом графе нет треугольников, значит, если этот граф планарен, то в его плоской укладке каждая грань ограничена не менее чем четырьмя ребрами и, следовательно, $4f \leqslant 2q$ или $2f \leqslant q$. По формуле Эйлера $6-9+f=2$, откуда $f=5$. Имеем $2f=2 \cdot 5=10 \leqslant q=9$ — противоречие. □

ЗАМЕЧАНИЕ

Операция *подразбиения* ребра $x = (u, v)$ состоит в замене его двумя рёбрами, (u, w) и (w, v), где w — новая вершина. Два графа называются *гомеоморфными*, если они могут быть получены из одного графа подразбиением рёбер. Граф планарен тогда и только тогда, когда он не содержит подграфов, гомеоморфных K_5 или $K_{3,3}$. Доказательство достаточности этого утверждения (*теоремы Куратовского*) выходит за рамки данного курса.

СЛЕДСТВИЕ 3. *В любом планарном графе существует вершина, степень которой не больше 5.*

Доказательство. От противного. Пусть $\forall\, v \in V \;\; (d(v) \geqslant 6)$. Тогда

$$6p \leqslant \sum_{v \in V} d(v) = 2q \Longrightarrow 3p \leqslant q,$$

но $q \leqslant 3p - 6$. Имеем $3p \leqslant 3p - 6$ — противоречие. □

ОТСТУПЛЕНИЕ

Для случая многогранников формулу Эйлера вывел Декарт. Действительно, каркас многогранника — это плоский граф, и обратно, связному плоскому графу (при $f > 3$) соответствует многогранник.

10.8.3. Теорема о пяти красках

ТЕОРЕМА. *Всякий планарный граф можно раскрасить пятью красками.*

Доказательство. Достаточно рассматривать связные графы, потому что компоненты можно раскрашивать независимо: $\chi\left(\bigcup\limits_{i=1}^{n} G_i\right) = \max\limits_{i=1}^{n} \chi\left(G_i\right)$. Индукция по p. База: если $p \leqslant 5$, то $\chi \leqslant p \leqslant 5$. Пусть теорема верна для всех связных планарных графов с p вершинами. Рассмотрим граф G с $p+1$ вершинами. По третьему следствию к формуле Эйлера $\exists\, v \in V \;\; (d(v) \leqslant 5)$. По индукционному предположению $\chi(G - v) \leqslant 5$. Нужно раскрасить вершину v. Если $d(v) < 5$, то в 5-раскраске графа $G - v$ существует цвет, свободный для вершины v. Если $d(v) = 5$ и для $\Gamma^{+}(v)$ использованы не все пять цветов, то в 5-раскраске графа $G - v$ существует цвет, свободный для вершины v. Остался случай, когда $d(v) = 5$ и все пять цветов использованы. Пусть G_{13} — правильный подграф графа $G - v$, порожденный всеми вершинами, покрашенными в цвета 1 или 3 в 5-раскраске графа $G - v$. Если v_1 и v_3 принадлежат разным компонентам связности графа G_{13}, то в той компоненте, в которой находится вершина v_1, произведем перекраску $1 \leftrightarrow 3$. При этом получится 5-раскраска графа $G - v$, но цвет 1 будет свободен для вершины v. В противном случае существует простая цепь, соединяющая v_1 и v_3 и состоящая из вершин, покрашенных в цвета 1 и 3. Тогда вершины v_2 и v_4 принадлежат разным компонентам связности подграфа G_{24} (так как граф G — планарный). Перекрасим вершины $2 \leftrightarrow 4$ в той компоненте связности графа G_{24}, которой принадлежит v_2, и получим 5-раскраску графа $G - v$, в которой цвет 2 свободен для вершины v (см. рис. 10.13). □

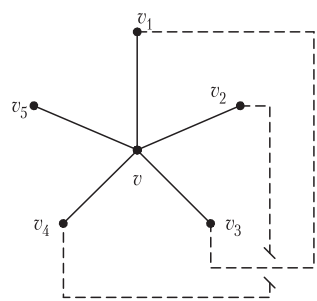


Рис. 10.13. К доказательству теоремы о пяти красках

Указатель основных обозначений

Метаобозначения

Обозначение	Смысл	Пример
$\stackrel{\text{Def}}{=}$	По определению есть	$\lvert\varnothing\rvert \stackrel{\text{Def}}{=} 0$
$:=$	Положим равным	$c:=(a+b)/2$
$=$	Равно	$1=1, 2\times 2=4$

Числовые множества

Обозначение	Название	Примеры
$\mathbb{N}$	Натуральные числа	$1, 2, 3$
$\mathbb{Z}$	Целые числа	$0, 1, -1, 2, -2$
$\mathbb{Q}$	Рациональные числа	$\frac{1}{2}, \frac{1}{3}, \frac{553}{113}, 2,718281828$
$\mathbb{R}$	Вещественные числа	$1, 1.5, \sqrt{2}, \pi, e$

Совокупности

Обозначение	Применение	Примеры
$\{a_1, \ldots, a_n\}$	Неупорядоченное множество различных элементов	$\{1, 2, 3\}$
$(a_1, \ldots, a_n)$	Упорядоченная последовательность однородных элементов	$(0, 1, 0, 1)$
$\langle A; B, C\rangle$	Упорядоченная последовательность разнородных элементов	$\langle\mathbb{R}; +, *\rangle$
$\langle a_1, \ldots, a_n\rangle$	Упорядоченная последовательность составных элементов	$\langle a\to 01, b\to 10\rangle$

Операции с множествами

Обозначение	Прочтение	Примеры
$a \in A$	Элемент a принадлежит множеству A	$1 \in \{1, 2, 3\}$
$a \notin A$	Элемент a не принадлежит множеству A	$4 \notin \{1, 2, 3\}, \sqrt{2} \notin \mathbb{Q}$
$A \subset B$	Множество A является подмножеством множества B	$\{2, 3\} \subset \{1, 2, 3\}$
$A \cup B$	Объединение множеств A и B	$\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$
$A \cap B$	Пересечение множеств A и B	$\{1, 2\} \cap \{2, 3\} = \{2\}$
$A \setminus B$	Разность множеств A и B	$\{1, 2\} \setminus \{2, 3\} = \{1\}$
$A \triangle B$	Симметрическая разность множеств A и B	$\{1, 2\} \triangle \{2, 3\} = \{1, 3\}$
$A \times B$	Прямое произведение множеств A и B	$\{1, 2\} \times \{2, 3\} = = \{(1, 2), (1, 3), (2, 2), (2, 3)\}$
$\varnothing$	Пустое множество	$\{\ \}$
$\lvert A \rvert$	Мощность множества A	$\lvert\{1, 2\}\rvert = 2$

Логические обозначения

Обозначение	Название	Прочтение
$\neg P$	Отрицание	Не P
$P \vee Q$	Дизъюнкция	P или Q
$P \,\&\, Q$	Конъюнкция	P и Q
$P \Longrightarrow Q$	Импликация	Если P, то Q
$\forall x\ (P(x))$	Квантор всеобщности	Для всех x выполнено $P(x)$
$\exists x\ (P(x))$	Квантор существования	Существует x, такой, что выполнено $P(x)$

Отношения

Обозначение	Название
$R \circ S$	Композиция отношений
R^n	Степень отношения
$\boxed{R}$	Матрица отношения
$\equiv$	Отношение эквивалентности
$\prec$	Отношение порядка
$<$	Отношение строгого линейного порядка
$\leqslant$	Отношение нестрогого линейного порядка

Функции и отображения

Обозначение	Прочтение	Примечание
$f\colon A \to B$	Функция f из A в B	$f \subset A \times B$
$b = f(a)$	b является значением функции f для аргумента a	$a \mapsto b$
$f \circ g$	Суперпозиция функций f и g	$(f \circ g)(x) = f(g(x))$
f^{-1}	Обратная функция к f	$f^{-1}\colon B \to A$
$\mathrm{Dom}\, f$	Множество определения функции $f\colon A \to B$	$\forall\, a \in \mathrm{Dom}\, f \;\; (\exists\, b \in B \;\; (b = f(a)))$
$\mathrm{Im}\, f$	Множество значений функции $f\colon A \to B$	$\forall\, b \in \mathrm{Im}\, f \;\; (\exists\, a \in A \;\; (b = f(a)))$

Стандартные функции

Обозначение	Прочтение	Примечание
$\lfloor x \rfloor$	Целая часть числа	$\forall\, x \in \mathbb{R} \;\; (\lfloor x \rfloor \leqslant x)$
$\log_a x$	Логарифм по основанию a	$\log x := \log_2 x$; $\ln x := \log_e x$
$m \,\mathbf{div}\, n$	Неполное частное	$m = (m \,\mathbf{div}\, n) * n + m \,\mathbf{mod}\, n$
$m \,\mathbf{mod}\, n$	Остаток от деления	$m = (m \,\mathbf{div}\, n) * n + m \,\mathbf{mod}\, n$
$n!$	Факториал	$n! := \prod_{i=1}^{n} i$

Групповые операции

Обозначение	Прочтение	Примечание
$\sum_{i=1}^{n} a_i$	Сумма элементов $a_1, \ldots, a_n$	$\sum_{i=1}^{n} a_i = a_1 + \ldots + a_n$
$\prod_{i=1}^{n} a_i$	Произведение элементов $a_1, \ldots, a_n$	$\prod_{i=1}^{n} a_i = a_1 \times \cdots \times a_n$
$\min(a_1, \ldots, a_n)$	Минимальный из элементов $a_1, \ldots, a_n$	$\min(a, b) \leqslant a \;\&\; \min(a, b) \leqslant b$
$\max(a_1, \ldots, a_n)$	Максимальный из элементов $a_1, \ldots, a_n$	$\max(a, b) \geqslant a \;\&\; \max(a, b) \geqslant b$

Список литературы

1. Ахо А., Хопкрофт Дж., Ульман Дж. *Построение и анализ вычислительных алгоритмов*. М.: Мир, 1979.
2. Кормен Т., Лейзерсон Ч., Ривест Р. *Алгоритмы: построение и анализ*. М.: МЦНМО, 1999.
3. Лавров С. С., Гончарова Л. И. *Автоматическая обработка данных, хранение информации в памяти ЭВМ*. М.: Наука, 1971.
4. Ахо А., Ульман Дж. *Теория синтаксического анализа, перевода и компиляции*. М.: Мир, 1978.
5. Ершов А. П. *Введение в теоретическое программирование*. М.: Наука, 1977.
6. Успенский В. А. *Теорема Геделя о неполноте*. М.: Наука, 1982.
7. Сачков В. Н. *Введение в комбинаторные методы дискретной математики*. М.: Наука, 1982.
8. Зыков А. А. *Основы теории графов*. М.: Наука, 1987.
9. Харари Ф. *Теория графов*. М.: Мир, 1973.
10. Кнут Д. *Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы*. М.: Мир, 1977.
11. Кнут Д. *Искусство программирования для ЭВМ. Т. 2. Получисленные алгоритмы*. М.: Мир, 1977.
12. Кнут Д. *Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск*. М.: Мир, 1977.
13. Кристофидес Н. *Теория графов. Алгоритмический подход*. М.: Мир, 1978.
14. Липский В. *Комбинаторика для программистов*. М.: Мир, 1988.
15. Андерсон Д. *Дискретная математика и комбинаторика*. М.: Вильямс, 2003.

Предметный указатель

A

Автоматическое доказательство теорем (automatic theorem proving), 244
Автоморфизм (automorphism), 108, 339
Адекватность (adequacy), 208
  дедуктики, 241
Азбука Морзе (Morse alphabet), 302
Аксиома (axiom), 21
  выбора (of choice), 93
  логическая (logical axiom), 206
  метрики (of a metric, 314
  регулярности (of regularity), 24
  симметрии (of symmetry, 314
  собственная (proper axiom), 206
  тождества (of identity, 314
  формальной теории (of formal theory), 206
Аксиоматизируемость (axiomatizability), 208, 227
  конечная (finite), 227
Алгебра (algebra), 27, 105
  Линденбаума–Тарского (Lindenbaum–Tara), 165
  булева (boolean algebra), 149
  булевых функций (of boolean functions), 164
  высказываний (propositional algebra), 201
  конечно-порожденная (finitely generated algebra), 106
  многоосновная (multi-based algebra), 105
  подмножеств (of subsets), 35, 105
  термов свободная (free algebra of terms), 107
  универсальная (universal algebra), 105
Алгоритм (algorithm), 229
  Беллмана–Форда (Bellman–Ford) , 391
  Дейкстры (Dijkstra), 392
  Евклида (Euclid), 128
  Кнута–Бендикса (Knuth–Bendix), 114
  Краскала (Joseph Kruskal), 442
  Лемпела–Зива (Lempel–Ziv), 325
  Прима (Robert Clay Prim), 443
  Уоршалла (Stephen Warshall), 64
  Фано (Robert Mario Fano), 305
  Хаффмена (David Albert Huffman), 308
  восстановления упорядоченного ордерева по коду (ordered directed tree recovery), 412
  вставки узла в дерево сортировки (node insertion in sorting tree), 422
всюду отличающийся (everywhere different), 237
выделения компонент сильной связности (strong connectivity component), 386
вычисления значения функции по дереву решений (evaluation of function on the base of decision tree), 190
вычисления номера кортежа в установленном порядке (evaluation of tuple's number in established order), 178
вычисления объединения слиянием (union by merge), 48
вычисления пересечения слиянием (intersection by merge), 50
вычитания итераторов (substraction of iterators), 51
генерации всех подмножеств (generation of all subsets), 42
генерации перестановок (generation of permutations), 264
генерации подмножеств (generation of subsets), 275
деления вычитанием (of division by subtraction), 126
детерминированный (determined), 235
жадный (greedy), 104, 152
интерпретации (interpretation), 160, 161, 237
линейный (linear), 152
метода резолюций (resolution method), 251
нахождения максимального потока (maximal flow), 383
неэффективный (inefficient), 462
нормальный Маркова(Markov normal), 229
обхода бинарного дерева (binary tree traverse), 417
объединения дизъюнктных итераторов (union of disjunctive iterators), 51
определения расстояний от источника (distances from source), 391
пересечения итераторов (intersection of iterators), 51
перечисляющий (enumeration), 231
поиска (search)
  бинарного (binary), 420
  в глубину (depth first), 356
  в дереве сортировки (in sorting tree), 421

в ширину (breadth first), 356
с возвратами (backtracking), 462
последовательного раскрашивания (sequential coloring), 471
последовательного раскрашивания (улучшенный) (improved sequential coloring), 472
построения СДНФ (construction of perfect DNF), 179
построения кода Прюфера (Prüfer's code), 409
построения кода упорядоченного ордерева (ordered directed tree code), 411
построения кратчайшего остова (shortest spanning tree), 441
построения максимальных независимых множеств (maximal independent sets), 465
построения эйлерова цикла (of Euler cycle), 453
приближённый (approximate), 471
применимый (applicable), 236
проверки включения слиянием (checking inclusion by merge), 48
проверки правильности скобочной структуры (checking parenthesis structure accuracy), 414
равносильный (equivalent), 237
распаковки кода Прюфера (unpacking Prüfer's code), 410
слияния (merge), 47
топологической сортировки (topological sorting), 87
удаления узла из дерева сортировки (removal of node from sorting tree), 424
универсальный (universal), 238
унификации (unification), 210
условно равный (conventionally equal), 236
эффективный (efficient), 462
Алфавит (alphabet), 26
формальной теории (of formal theory), 206
Аргумент (argument), 230
функции (of function), 70
Арифметика (arithmetic), 27, 125, 242
высшая (higher), 125
двоичная (binary), 124
модулярная (modular), 136
Арифметический полином (arithmetic polynom, 181
Ассоциативность (associativity)
объединения (of union), 36
пересечения (of intersection), 36
Атом (atom), 220, 242

Б

Базис (base), 160
векторного пространства (of vector space), 140
матроида (of matroid), 151
пространства коциклов (of cocyclic space), 450
пространства циклов (of cycle space), 450
Беспорядок (disorder), 289
Биграф (bigraph), 347
Биекция (bijection), 70
Бином Ньютона (binomial theorem), 271
Бит (bit), 37
чётности (parity), 311
Битовая шкала (bit scale), 37
ошибок (of errors), 321
Блок (block)
графа (of graph), 367
кода (code block), 311
разбиения (of partition), 34, 281
тривиальный (trivial), 368
Брат узла (sibling node), 405
Буква (letter), 26
Булеан (boolean), 35
Бэктрекинг (backtracking), 462

В

Валентность (valence)
вершины (of vertex), 340
Вектор (vector), 26, 76, 139
двоичный (binary), 37
циклический (cyclic), 448
Векторное пространство (vector space), 138
бесконечномерное (infinite-dimensional), 141
конечномерное (finite-dimensional), 141
Величина потока (value of flow), 381
Вероятность (probability), 258
подмножества (of subset), 258
Вершина (vertex)
висячая (dangling), 340
графа (of graph), 335
достижимая (accessible), 364
изолированная (isolated), 340
концевая (leaf), 340
покрывающая (covering), 457
разделяющая (separate), 367
связанная (connected), 342
центральная (central), 343
Ветвь ордерева (branch of directed tree), 405
Включение (inclusion)
множеств (of sets), 27
мультимножеств (of multisets), 100
Вместимость отношения (arity of relation), 57
Всюду определённое продолжение (everywhere defined continuation), 239
Вхождение (occurrence), 26
определяющее (defining), 18
переменной (variable)
свободное (free), 220
связанное (bound), 220
Выводимость (inference), 207
непосредственная (direct), 207
Выполнение алгоритма (execution of algorithm), 229
Выполнимость (satisfiability)
системы булевых функций (of Boolean functions system), 170
формулы (of formula), 207, 222
Выразимость (expressibility), 242
Вырезка из массива (slice of array), 435
Высказывание (proposition)
контрадикторное (contradictory), 204
контрарное (contrary), 204

простое (atomic), 199
субконтрарное (subcontrary), 204
универсальное (universal), 203
экзистенциальное (existential), 203
Высота (height)
дерева (of tree), 405
множества (of set), 86
Вычитание (subtraction)
матриц (of matrix), 62

Г

Гамма шифра (cypher gamma), 329
Геодезическая цепь (geodesic line), 342
Гиперграф (hypergraph), 337
Гипердуга (hyperedge), 337
Гипероргаф, 337
Гипотеза (hypothesis), 207
Гомоморфизм (homomorphism), 80, 108
Граница (bound)
верхняя (upper), 88, 146
наименьшая (least), 146
нижняя (low), 88, 146
наибольшая (greatest), 146
Грань (bound)
графа (of graph), 473
решётки (of lattice)
верхняя (upper), 144
нижняя (low), 144
Граф (graph), 335
n-связный (n-connected), 371
Герца (Herz), 386
ациклический (acyclic), 341, 399
гамильтонов (Hamilton), 454
геодезический (geodesic), 342
гомеоморфный (homeomorphic), 474
двудольный (bipartite), 347
двусвязный (biconnected), 372
древочисленный (tree-numerical), 399
инциденций (of incidences), 362
нагруженный (weighted), 337
несвязный (disconnected), 342
вполне (totally), 342
нумерованный (numbered), 337, 453
ориентированный (directed), 337
планарный (planar), 473
плоский (flat), 473
полный (complete), 347
двудольный (bipartite), 347
полугамильтонов (semi-Hamilton), 454
полуэйлеров (semi-Euler), 451
помеченный (labeled), 337
пустой (empty), 347
рёберный (line), 352
регулярный (regular), 340
с петлями (with loops), 337
связный (connected), 342
субциклический (subcyclic), 399
тривиальный (trivial), 347
чётный (even), 347
эйлеров (Euler), 451
График (graph)
отношения (of relation), 55
Группа (group), 117
абелева (abelian), 118, 226
делимая (divisible), 227
избыточная (excees), 189
коммутативная (commutative), 118
максимальная (maximum), 187
перестановок (of permutations), 119
перестановок (of substitutions), 120
периодическая (periodic), 227
порядка n (n-order), 226
симметрическая (symmetric), 119, 120

Д

Двоичное представление (binary representation), 40
Двойственность (duality), 87
Дедуктика, 240
Действие (operation), 119
Действие группы на множестве (operation of group on set), 119
Декодирование (decoding), 298
Делимость (divisibility), 126
Делитель (divisor), 126
нетривиальный (non-trivial), 126
нуля (of zero), 123
левый (left), 123
правый (right), 123
общий (common), 127
наибольший (greatest), 127
собственный (proper), 126
тривиальный (trivial), 126
Дерево (tree), 399
m-ичное (m-ary), 408
АВЛ-дерево (AVL-tree), 428
И/ИЛИ (and/or), 364
бинарное (binary), 408
полное (full), 427
вывода (of derivation), 249
выровненное (aligned tree), 426
двоичное (binary), 408
заполненное (thick), 427
игровое (game), 364
корневое (rooted), 403
кратчайших путей (of shortest paths), 390
ориентированное (directed), 403
отрезков (segment), 435
подровненное (balanced), 428
прошитое (threaded), 416
решений (decision), 189
сбалансированное по весу (weight balanced), 428
сбалансированное по высоте (height balanced), 428
свободное (free), 399
семантическое (semantic), 189
сортировки (sorting), 418, 419
упорядоченное (ordered), 405
упорядочивания (sorting), 418
Дешифрация (decoding), 328
Дешифрование (decoding), 328

Диагональ (diagonal)
прямого произведения (of direct product), 56
Диаграмма (diagram)
Вейча (Veitch), 183
Венна (Venn), 34
Хассе (Hasse), 67
бинарная решений (binary of solutions), 190
графа (of graph), 336
коммутативная (commutative), 109
Диаметр (diameter)
графа (of graph), 343
Дивергенция (divergence), 381
Дизъюнкт (clause), 245
резольвируемый (resolvable), 246
родительский (parent), 246
Дизъюнкция (disjunction), 37
матриц (of matrix), 62
Дискриминантный анализ (discriminant analysis), 100
Дистанция редактирования (edit distance), 314
Дистрибутивность (distributivity)
объединения относительно пересечения (of union with respect to intersection), 36
пересечения относительно объединения (of intersection with respect to union), 36
Дисциплина имён (naming convention), 129
Длина (length)
маршрута (of route), 342
набора (of tuple), 53
последовательности (of sequence), 86
слова (of word), 26
цикла (of cycle), 60, 260, 341
Добавление (adding)
вершины (of vertex), 349
ребра (of edge), 350
Доказательство (proof), 240
Доля (part), 347
Дополнение (complement), 145
графа (of graph), 336, 349
множества (of set), 33
нечёткое (fuzzy), 102
Достоверность декодирования (accuracy of decoding), 313
Дуга (arc), 337
вес (weight), 388
длина (length), 388
насыщенная (saturated), 381

Е

Единица (identity)
моноида (of monoid), 116
решётки (of lattice), 144

З

Заголовок алгоритма (algorithm header), 230
Задача (problem)
NP-полная (*NP*-complete), 455
Штейнера (Jacob Steiner), 444
комбинаторная (combinatorial), 253
коммивояжёра (of travelling salesman), 455
минимизации дизъюнктивной формы (of minimum disjunctive form), 176
о восьми ферзях (of eight queens), 465
о выборе переводчиков (translator choice), 467
о кёнигсбергских мостах (of Königsberg bridges), 334
о наименьшем покрытии (of least covering), 466, 467
о пяти ферзях (five queens), 466
о развозке (of transshipment), 467
о свадьбах (marriage), 376
о трёх домах и трёх колодцах (of three houses and three utilities), 334
о четырёх красках (of four colours), 334
переборная (of search), 276
сортировки (of sorting), 263
Заключение правила вывода (conclusion of inference rule), 207
Законы де Моргана (de Morgan laws), 36
Замена буквальная (literal replacement), 182
Замена переменных (substitution of variables), 116
Замыкание (closure), 63, 192
множества (of set), 106
отношения (of relation), 64
формулы (of formula), 223
Запись (notation)
префиксная (prefix), 70
Запись (record), 418
Зашифровка (enciphering), 328
Звезда (star), 459
Значение (value)
истинностное (truth value), 199
функции (of function), 70

И

Идемпотентность (idempotency)
объединения (of union), 36
пересечения (of intersection), 36
Изграф (subgraph), 339
Измельчение разбиения (refinement of partition), 281
Изоморфизм (isomorphism), 80, 108
алгебр (of algebras), 109, 165
графов (of graphs), 338
множеств (of sets), 28
вполне упорядоченных (well-ordered), 93
линейно упорядоченных (linear ordered), 92
Изоморфность (isomorphism)
рёберная (edge), 353
Импликанта (implicant), 187
Инвариант (invariant)
графа (of graph), 338
Инверсия (inversion), 263
матрицы (of matrix), 62
Инвертирование (invertion), 37
Инволютивность (involutivity)
дополнения (of complement), 36
Индикатор (indicator), 25
Интерпретация (interpretation)
исчисления предикатов (of predicate calculus), 222
формальной теории (of formal theory), 207
формулы (of formula), 200

Интерпретация (interpretation) представления (of representation), 389
Инфиксная форма записи (infix notation), 56
Инфимум (infimum), 88
Инцидентность (incidence), 336
Инъекция (injection), 70
Исключающее или (exclusive or), 37
Искусственный интеллект (artificial intelligence)
  символический (symbolic), 364
Исправление ошибок (error correction), 311
Истина (truth), 240
Истинностное значение (truth value), 95, 155
Источник (source), 349
Исчисление (calculus)
  высказываний (propositions calculus), 209
  предикатов (predicates calculus), 220
    высшего порядка (higher order), 222
    первого порядка (first order), 220, 222
    прикладное (applied), 222
    с равенством (with equality), 226
    чистое (pure), 220, 222
Итератор (iterator), 50

К

Канал (channel)
  двоичный симметричный (binary symmetric), 313
  связи (communication)
    с помехами (noisy), 310
Каркас (skeleton), 440
Карта (map), 334
Карта Карно (Karnaugh map), 183
Квадрант (quadrant), 95
Квазипорядок (quasi–order), 82
Квантор (quantifier)
  всеобщности (universal), 203, 220
  существования (existential), 203, 220
  существования и единственности (existential and uniqueness), 204
Класс (class), 22
  арифметических множеств (of arithmetic sets), 243
  вычетов (residue)
    по модулю m (modulo m), 134
  замкнутый (closed), 193
  одноцветный (unicolored), 469
  полный (complete), 194
  представительный (representative), 237
  толерантности (of tolerance), 79
  эквивалентности (of equivalence), 75
Классификатор (classifier), 94, 100
  иерархический (hierarchical), 101
  неполный (incomplete), 95
  универсальнвй десятичный (universal decimal), 101
  фасетный (faceted), 101
Клика (clique), 347
Ключ (key)
  ассоциативной памяти (content-addressable), 418
  шифра (of encryption), 328
Код (code)
  Прюфера (Prüfer), 409
  Хэмминга (Hamming), 321
  бинарный (binary), 40
  бинарный Грея (binary Frank Gray), 44
    левый зеркальный (left mirror), 45
  двоичный (binary), 37
  множества (of set), 38
  сообщения (of message), 298
  элементарный (elementary), 299
Кодерево (cotree), 447
Кодирование (coding), 298
  m-ичное (m-ary), 298
  систематическое (systematic), 320
  алфавитное (alphabetic), 299
  безизбыточное (breakeven), 317
  двоично-десятичное (binary coded decimal), 300
  двоичное (binary), 298
  длина (length), 304
  однозначное (unique), 299
  оптимальное (optimal), 304
  побуквенное (alphabetic), 299
  помехоустойчивое (antinoise), 310
  равномерное (uniform), 304
  с исправлением ошибок (error correction), 310
  с минимальной избыточностью (optimal), 304
  самокорректирующееся (self-correcting), 310
  цена (price), 304
Кодовое слово (codeword), 299
Коллизия (collision), 420
Коллинеарность (collinearity), 76
Кольцо (ring), 122
  коммутативное (commutative), 123
  с единицей (with identity), 123
  целых чисел (of integer numbers), 105
Комбинаторика (combinatorics), 252
Комбинаторный анализ (combinatorial analysis), 252
Комментарий (comment), 129
Коммутативность (commutativity)
  объединения (of union), 36
  пересечения (of intersection), 36
Компактная группа ячеек (compact group of cells), 187
Композиция (composition)
  отношений (of relations), 57
  подстановок (of substitutions), 116
Компонента (component)
  связности (of connectivity), 342
  сильной связности (of strong connectivity), 386
Конденсация орграфа (condensation of digraph), 386
Конец цепи (circuit end), 341
Константа (constant), 242
  предметная (object), 220
Конституета единицы (unit constutuent), 169
Конструктивизм (constructivism), 24
Контекст (context), 464
Контур (contour), 342

Конфигурация комбинаторная (combinatorial configuration), 253
Конъюнкция (conjunction), 37
  допустимая (admissible), 176
  максимальная (maximal), 176
Координаты (coordinates), 140
  Декартовы (Cartesian), 298
Корень (root)
  ордерева (of directed tree), 403
  уравнения (of equation), 90
Кортеж (tuple), 53, 57
Коцикл (cocycle), 449
Коэффициент (coefficient)
  биномиальный (binomial), 271
    расширенный (extended), 277
  мультиномиальный (multinomial), 279
  сжатия (of compression), 324
Кратное (multiple), 126
  общее (common), 129
    наименьшее (least), 129
Криптография (cryptography), 328
Криптостойкость (cryptostability), 328
Крона ордерева (tree top), 405

Л

Лес (forest), 399
Линейная комбинация (linear combination), 140
Лист ордерева (leaf of directed tree), 405
Литерал (literal), 221
  контрарный (contrary), 246
Логическая связка (logical connective), 199
Логический квадрат (logical square), 204
Логическое отрицание (logical negation), 37
Логическое сложение (logical addition), 37
Логическое умножение (logical multiplication), 37

М

Макстерм (maxterm), 170
Маршрут (route), 341
  замкнутый (closed), 341
  открытый (opened), 341
Массив (array)
  дуг (of arcs), 356
  рёбер (of edges), 356
Математическое ожидание (expectation), 259
Матрица (matrix)
  порождающая (generating)), 319
  проверочная (control), 320
  булева (boolean), 62
  инциденций (incidence), 355
  перестановочная (permutable), 121
  подстановки (of substitutions), 121
  предшествования (precedence), 389
  смежности (adjacency), 354
  смежности (of adjacency)
    ребёр (of edges), 352
Матроид (matroid), 150
  разбиений (of partitions), 154
  свободный (free), 153
  трансверсалей (of transversals), 154
Машина (machine)
  Поста (of Post), 229
  Тьюринга (of Turing), 229
Машинное слово (machine word), 37
Медиана (median)
  множества (of set), 305
Метатеорема (metatheorem), 207
Метод (method), 50
  аксиоматический (axiomatic), 21
  включений и исключений (inclusion-exclusions-), 286
  резолюций (of resolutions), 244, 245
Метрика (metric), 314
Минтерм (minterm), 169
Множество (set), 21
  аксиом (of axiom)
    независимое (independent), 208
  арифметическое (arithmetic), 243
  бесконечное (infinite), 23, 29
  вершин (of vertex)
    внешне устойчивое (outer stable), 466
    внутренне устойчивое (internally stable), 458
    доминирующее (dominating), 466
    независимое (independent), 458
    разделяющее (dividing), 373
  вполне упорядоченное (well-ordered), 92
  зависимое (dependent), 150
  заданное (defined by)
    перечислением элементов (enumeration), 23
    порождающей процедурой (generating procedure), 23
    характеристическим предикатом (characteristic predicate), 23
  замкнутое (closed), 105
  изоморфное (isomorphic), 81
  инцидентности (of incidence), 336
  истин (of truths), 240
  истин арифметики (of truths of arithmetic), 243
  конечное (finite), 23, 29
  линейно зависимое (linearly dependent), 140
  линейно независимое (linearly independent), 140
  линейно полное (completely ordered), 89
  минимальное (minimal), 466
  наименьшее (least), 466
  независимое (independent), 150
    максимальное (maximal), 150
  несущее (support), 105
  несчётное (non–enumerable, 30
  нечёткое (fuzzy), 97
  образующих (of generators), 140
  перечислимое (enumerable), 231
  пометок (of labels), 337
  порождающее (generating), 140
  пустое (empty), 22
  рёбер (of edges)
    независимое (independent), 375, 458
    разделяющее (separate), 373
  разрезов (of cuts)
    независимое (independent), 450
  смежности (of adjacency), 336
  сопряжённое (conjugate), 243

счётное (enumerable), 29
узлов (of nodes)
доминирующее (dominating), 467
независимое (independent), 467
универсальное (universal), 23
упорядоченное (ordered)
линейно (linearly), 84
частично (partially), 84
уровня (level), 79
циклов (of cycles)
независимое (independent), 450
Модель (model), 105
множества формул (of set of formulas), 208, 223
формальной теории (of formal theory), 208
Модуль (module), 143
Моноид (monoid), 116
Мономорфизм (monomorphism), 108
Мост (bridge), 367
Мощность (power)
множества (of set), 32
Мощность (power, cardinal number)
кодирования (of coding), 316
множества (of set), 28
мультимножества (of multiset), 25, 100
Мультиграф (multigraph), 337
Мультимножество (multiset), 25

H

Набор (tuple), 53
Надмножество (superset), 27
собственное (proper), 27
Направленный отрезок (directed line segment), 76
Начало слова (prefix), 26
Непротиворечивость (consistency)
дедуктики, 241
семантическая (semantic), 208
формальная (formal), 208
Неравенство (inequality)
Коши–Буняковского (Cauchy–Schwarz), 460
Крафта–МакМиллана (Kraft–MacMillan), 301
треугольника (triangle), 314
Носитель (carrier), 27
Носитель (support), 105
интерпретации (of interpretation), 222
мультимножества (of multiset), 25
Нуль (identity)
группы (of group), 118
Нуль (zero)
решётки (of lattice), 144
Нуль-вектор (nill vector), 139
Нумерация множества (numbering of set), 33

O

Область
определения (domain)
отношения (of relation), 56
прибытия (target range)
функции (of function), 70
действия квантора (scope of quantifier), 221
значений (codomain, value range)
отношения (of relation), 56
функции (of function), 70
интерпретации (interpretation domain), 207
определения (domain)
функции (of function), 70
отправления (source range)
отношения (of relation), 55
функции (of function), 70
прибытия (target range)
отношения (of relation), 55
целостности (integral domain), 124
Обнаружение ошибок (diagnosis), 311
Образ (image), 71
Обфускация (obfuscation), 423
Обход дерева (traverse of tree)
внутренний (inorder), 416
инфиксный (inorder), 416
концевой (postorder), 416
левый (preorder), 416
постфиксный (postorder), 416
правый (postorder), 416
прямой (preorder), 407, 416
симметричный (inorder), 416
Объединение (union)
графов (of graphs)
дизъюнктное (disjunctive), 349
множеств (of sets), 33
дизъюнктное (disjunctive), 54
мультимножеств (of multisets), 100
нечёткое (fuzzy), 102
размеченное (marked), 54, 55
Окончание слова (postfix), 26
Окрестность (neighborhood)
δ (δ), 315
вершины (of vertex), 336
метрическая (metric), 314
Оператор (statement)
возврата (return), 74
структурного перехода (structural go to), 51
Операция (operation), 27
n-арная (n-ary), 104
n-местная (n-ary), 104
ассоциативная (associative), 107
вариаргументная, 163
внешняя (outermost), 160
главная (principal), 160
групповая (group), 34, 163
дистрибутивная (distributive), 107
добавления элемента (of element adding), 24
идемпотентная (idempotent), 107
коммутативная (commutative), 107
конечноместная (finitude), 105
конкатенации (of concatenation), 26, 110
кратная (multiple), 163
первичная (primary), 50
подразбиения (of subdivision), 474
связывания (of linking)
квантором всеобщности (linking with universal quantifier), 203
квантором существования (with existential quantifier), 203

квантором существования и единственности (with existential and uniqueness quantifier), 204
сравнения (of comparison), 27
сцепления (of concatenation), 26
удаления элемента (of element removal), 24
Опровержение методом резолюций (refutation with resolution method), 248
Орбита (orbit), 119, 261
Орграф (digraph), 337
антисимметричный (antisymmetric), 348
взвешенный (weighted), 388
направленный (directed), 348
Ордерево (directed tree), 403
Ортогонализация (orthogonalization), 182
Ортогональная система булевых функций (orthogonal system of Boolean functions), 169
Ослабление (relaxation), 389
Основа (base), 105
Основание (base), 136
системы счисления (number system), 41
Остаток (remainder), 126
Остов (skeleton), 440
кратчайший (shortest), 440
Ось (axis)
числовая (numeric), 85
Отец узла (node parent), 405
Откладывание (laying off), 344
Отношение (relation)
n-арное (n-ary), 57
n-местное (n-ary), 57
ациклическое (acyclic), 60
антирефлексивное (antireflexive), 58
антисимметричное (antisymmetric), 58
бинарное (binary), 55
делимости (of divisibility), 127
дополнительное (complement), 56
изоморфности (of isomorfism, 81
линейное (linear), 59
на множестве (on set), 56
обратное (converse), 56
однозначное (single-value), 70
переписывания (of rewriting), 112
полное (complete), 59
порядка (of order), 83
антилексикографического (antilexicographical), 264
лексикографического (lexicographical), 263
линейного (linear), 83
нестрогого (non–strict), 83
строгого (strict), 83
частичного (partial), 83
предпорядка (of preorder), 82
равенства (equality), 76
равномощности (of equivalence), 28
редукции (of reduction), 112
рефлексивное (reflexive), 58
симметричное (symmentic), 58
тождественное (identical), 56
транзитивное (transitive), 59
универсальное (universal), 56
функциональное (functional), 70
частичное (partial), 59
эквивалентности (of equivalence), 75
Отображение (map)
натуральное (natural), 78
Отступ (tabbing), 129
Очередь с приоритетом, 434

П

Память ассоциативная (content-addressable memory), 418
Парадокс Кантора (Cantor paradox), 36
Парадокс Рассела (Bertrand Russell's paradox), 24
Параметр (parameter), 242
Паросочетание (matching), 375, 458
неустойчивое (unstable), 377
совершенное (perfect), 376
устойчивое (stable), 377
Переменная (variable)
несущественная (inessential), 156
предметная (object), 220
пропозициональная (propositional), 199, 209
связанная (bounded), 221
существенная (essential), 156
фиктивная (inessential), 156
Переписывание (rewriting), 112
за один шаг (one step rewriting), 112
Пересечение (intersection), 26
множеств (of sets), 33
мультимножеств (of multisets), 100
нечёткое (fuzzy), 102
Перестановка (permutation), 119
Стирлинга (of Stirling), 266
обратная (inverse), 120
тождественная (identity), 120
циклическая (cyclic), 261
Период (period), 74
Петля (loop), 60, 260, 337
Плотность (density), 347
Побочный эффект (side-effect), 161
Поглощение (absorbancy), 36, 107
Подалгебра (subalgebra), 105
Подграф (subgraph), 339
остовный (spanning), 339, 440
правильный (regular), 339
собственный (proper), 339
Поддерево (subtree), 404
Подмножество (subset), 27
разрешимое (solvable), 234
собственное (proper), 27
Подстановка (substitution), 116, 120
Подформула (subformula), 160
Поиск (search)
бинарный (binary), 420
в глубину (depth first), 357
в ширину (breadth first), 357
двоичный (binary), 420
полнотекстовый (full text), 325
Показатель элемента (index of element), 25
Покрытие (covering), 34
вершинное (vertex), 457

рёберное (edge), 458
тупиковое (deadlock), 189
Поле (field), 124
Галуа (Galois), 141
вещественных чисел (of real numbers), 105
конечное (finite), 141
рациональных чисел (of rational numbers), 105
Полином (polynom)
Жегалкина (of Zhegalkin), 195
Полнота (completeness)
дедуктики, 241
формальной теории (of formal theory), 208
Полный перебор (exhaustive search), 455
Полугруппа (semigroup), 110
свободная (free), 111
циклическая (cyclic), 110
Полуразрешимость формальной теории (semisolveability of formal theory), 208
Полурешётка (semilattice), 147
ограниченная (bounded), 148
Полустепень
захода (in-degree), 340
исхода (out-degree), 340
Польская запись (polish notation), 416
обратная (reverse), 417
Помехоустойчивость (noise-immunity), 299
Порядок (order)
группы (of group), 117
по сложению (additive), 142
установленный (established), 156
Последовательность (sequence), 25
возрастающая (increasing), 86
строго монотонно (strict monotone), 86
графовая (graph), 343
заключительно периодическая (eventially periodic), 74
конечная (finite), 53
остаточная (residual), 344
периодическая (periodic), 74
степенная (degree)
графа (of graph), 343
Постфикс (postfix), 26
Посылка правила вывода (hypothesis of rule), 207
Потенциал (potential), 397
Поток (flow), 381
Потомок узла (descendant node), 405
Правило (rule)
введения импликации (introduction of implication), 213
вывода (inference)
производное (derived), 213
формальной теории (of formal theory), 206
двойного вращения (double rotation), 429
замены (of replacement), 163
отделения (of detachment), 209
подстановки (of substitution), 163
произведения (of product), 253
простого вращения (single rotation), 429
резолюции (resolution), 246
сечения (cut), 215
склеивания/расщепления (clutching/splitting), 172
сложения (of addition), 253
суммы (of sum), 253
транзитивности (of transitivity), 215
умножения (of multiplication), 253
Предикат (predicate), 95, 220
n-арный (n-ary), 220
n-местный (n-ary), 202, 220
одноместный (unary), 202
характеристический (characteristic), 23
Предложение (clause), 245
Предобработка (preprocessing), 397, 435
Предок узла (descendant node), 405
Представление (representation)
алгоритма (of algorithm), 237
минимальное (minimal), 65
множества (of set), 231
функции (of function), 230
Представления чисел (representation of numbers), 41
Преобразование (transformation), 70
над множеством (on set), 116
эквивалентное (equivalent), 172
Префикс (prefix), 26
Приведение подобных (collecting terms), 172
Признак (attribute)
делимости (of divisibility), 126
Принцип (principle)
Дирихле (Dirichlet), 33, 254
двойственности (of duality), 167
индукции (of induction), 93
Проблема (problem)
алгоритмически неразрешимая (algorithmically unsolvable), 112
Программа (program), 237
остаточная (residual), 237
Продолжение функции (extension of function), 70
Проекция (projection)
каноническая (canonical), 78
Проекция алгоритма (algorithm projection), 237
Произведение (product)
декартово (Cartesian), 53
перестановок (of permutations), 120
прямое (direct), 53
скалярное (scalar), 122
Прообраз (preimage), 71
Пропускная способность (capacity)
дуги (of arc), 380
канала (of chanal), 313
разреза (of cut), 381
Пространство (space)
векторное (vector), 139
поиска (search), 276
Протаскивание отрицаний (dragging of negations), 172, 245
Противоречие (contradiction), 200
Протокол выполнения (execution log), 235
Прямая сумма множеств (direct sum of sets), 54
Прямой доступ (direct access), 421
Псевдограф (pseudograph), 337

Путь (path), 342
кратчайший (shortest), 388

P

Равенство (equality)
множеств (of sets), 28
мультимножеств (of multisets), 100
упорядоченных пар (of ordered couples), 52
Радиус (radius)
графа (of graph), 343
Разбиение (separation), 34
Разделение связанных переменных (splitting of bounded variables), 245
Разложение (resolution)
функции (of function), 78
Размерность векторного пространства (dimension of vector space), 141
Размножение (duplication)
вершины (of vertex), 350
Разность (difference)
алгебраическая (algebraic)
нечётких множеств (of fuzzy sets), 103
множеств (of sets), 33
симметрическая (symmetric), 33
Разрез (cut), 373, 445
правильный (regular), 445
простой (simple), 446
фундаментальный (fundamental), 449
Разрешимость (solveability)
класса формул (formulas class), 170
формальной теории (of formal theory), 208
Разряд (bit), 37
информационный (of data, 320
контрольный (of check), 317, 320
проверочный (of check), 320
Ранг (rank)
конъюнкции (of conjunction), 175
коциклический (cocyclomatic), 449
циклический (cyclomatic), 447
Раскраска графа (coloring of graph), 468
Раскрытие скобок (removal of brackets), 172
Распознавание образов (pattern recognition), 100
Распределение вероятности (probability distribution), 258
равномерное (uniform), 258
Расстояние (distance), 314, 342
Евклидовое (Euclid), 314
Левенштейна (Lowenstein), 314
Хэмминга (Hamming), 315
кодовое (code), 315
Расширение (expansion), 142
Расшифровка (deciphering), 328
Расшифровывание (deciphering), 328
Расщепление (splitting), 172
переменных (variables), 172
Реализация (realization), 161, 230, 237, 344
Ребро (edge)
графа (of graph), 335
кратное (multiple), 337
покрывающее (covering), 457
Редукция (reduction), 64
транзитивная (transitive), 65
Резольвента (resolvent), 246
Результат (result), 230
Рекорд пути (path record), 392
Рекуррентность (recurrence), 46
Релаксация (relaxation), 389
Решётка (lattice), 144
дистрибутивная (distributive), 144
ограниченная (bounded), 144
полная (complete), 147
с дополнением (with complement), 145
Решето (sieve)
Эратосфена (of Eratosthenes), 131
Родитель узла (parent node), 405

C

Свойство (property), 50, 289
Чёрча–Россера (of Church–Rosser), 113
Связанность узлов (connectivity of nodes)
односторонняя (one-way), 385
сильная (strong), 385
слабая (weak), 385
Связка логическая (logical connective), 209
Связность (connectivity)
вершинная (vertex), 371
односторонняя (one-way), 385
рёберная (edge), 371
сильная (strong), 385
слабая (weak), 385
Семантика (semantics), 238
Семейство (family)
дизъюнктное (disjunct), 34
множеств (of sets), 22
равномощных подмножеств (of equipotent sets), 35
Сеть (network), 349
Сжатие (compression), 324
адаптивное (adaptive), 325
с потерями (lossy), 313
Сигнатура (signature), 105
формальной теории (of formal theory), 206
Символ (symbol), 26
Синдром (syndrome), 321
Синтаксис (syntax), 238
Система (system)
вычетов по модулю m (group of integers modulo m), 134
наименьших неотрицательных (least non-negative), 134
приведённая (reduced), 135
образующих (generator set), 106
остаточных классов (of residue classes), 136
подстановок термов (of rewriting rules), 174
подстановок термов (of term substitutions), 112, 174
конфлюэнтная (confluent), 113
остановочная (stopping), 113
терминирующая (terminating), 113
правил переписывания (of rewriting rules), 112
различных представителей (of different representatives), 376

разрезов (of cuts)
фундаментальная (fundamental), 449
счисления (number), 41
k-биномиальной (k-binomial), 280
биномиальная (binomial), 41
остаточных классов (of residual classes), 41
позиционная (positional), 41
смешанная (mixed), 41
факториальная (factorial), 41
фибоначчиева (Fibonacci), 41
циклов (of cycles)
фундаментальная (fundamental), 447
Систематика (systematics), 100
Скаляр (scalar), 139, 144
Склеивание (clutching), 172
Сколемизация (skölemising), 246
Следствие логическое (logical consequence), 201, 208, 224
Словарь (dictionary), 325
Слово (word), 26, 324
машинное (machine), 38
пустое (empty), 26
Сложение по модулю 2 (addition modulo 2), 37
Сложность (complexity)
временна́я (time), 252
емкостна́я (space), 252
по времени (time), 252
по памяти (space), 252
Случай (case)
лучший (best), 252
худший (worst), 252
Случайная величина (random value), 259
Смежность (adjacency)
вершин (vertex), 336
рёбер (edge), 336
Смешанные вычисления (mixed computations), 161
Совершенный одночлен (perfect monomial), 169
Содержаться (be contained), 27
Соединение (join)
графов of graphs), 349
Сокращение (reduction), 64
транзитивное (transitive), 65
Сообщение (message), 298
шифрованное (coded), 328
Соответствие взаимно-однозначное (one-to-one correspondence), 28
Соотношение (relationship)
определяющее (defining), 111
Сортировка (sorting), 263
быстрая (fast), 266
пузырьком (bubble), 263
формулы (formula), 173
Состав (compound)
мультимножества (of multiset), 25
последовательности (of sequence), 278
Список (list)
смежности (of adjacency), 355
элементов (of elements), 47
Сравнимость (congruence), 133
Сравнимость по модулю (parity), 133
Средняя степень (average degree), 460
Степень (degree)
вершины (of vertex), 340
максимальная (maximal), 340
минимальная (minimal), 340
множества (of set), 54
отношения (of relation), 60
функции (of function), 73
Стиль программирования (coding standard), 129
Сток (sink), 349
Стратегия (strategy)
метода резолюций (of resolution method), 251
Стратегия метода резолюций (strategy of resolution method)
неполная (incomplete), 251
полная (complete), 251
Структура алгебраическая (algebraic structure), 105
Стягивание (shrinkage)
подграфа (of subgraph)
правильного (proper), 350
Субфакториал (subfactorial), 289
Суждение (proposition), 243
Сужение (reduction), 142
Сужение функции (restriction of function), 70
Сумма (sum)
алгебраическая (algebraic)
нечётких множеств (of fuzzy sets), 103
Суперпозиция (superposition), 72
Супремум (supremum), 88
Схема (scheme)
аксиомы (of axiom), 206, 209
кодирования (coding), 299
правила вывода (of inference rule), 209
префиксная (prefix), 300
разделимая (separable), 300
Схема Горнера (Horner's method), 178
Сын узла (child node), 405
Сюръекция(surjection), 70

T

Таблица (table)
истинности (of truth values), 155
кодов (of codes), 299
подстановки (of substitution)), 260
расстановки (hash), 419
хэш (hash), 419
Тавтология (tautology), 200, 208
Тайнопись (cryptogram), 328
Тег (tag), 55
Тезис Тьюринга–Чёрча (Turing–Church thesis), 229
Тело алгоритма (algorithm body), 230
Теорема (theorem), 240
Гёделя (Kurt Friedrich Gödel), 228
Куратовского (Kazimierz Kuratowski), 474
Маркова–Поста (Markov–Post), 111
Менгера (Karl Menger), 374
Поста (Emil Leon Post), 195
Форда–Фалкерсона (Ford–Fulkerson), 382
Холла (Philip Hall), 376

китайская об остатках (chinese theorem of remainders, 135
формальной теории (of formal theory), 207
Теория (theory)
наивная алгоритмов (naive of algorithms), 229
вероятности (of probability), 258
групп (group), 226
информации (of information), 313
категорий (of categories), 110
наивная доказательства (naive of proof), 240
полуразрешимая (semisolvable), 248
равенства (of equality), 225
формальная (formal), 206
формальной арифметики (formal arithmetic), 225
чисел (of numbers), 125
элементарная (elementary), 125
Терм (term), 107, 112, 220, 242
минимальный (minimal), 169
нередуцированный (unreducted), 113
редуцированный (reducted), 113
свободный для переменной в формуле (free for variable in formula), 221
Тип (type), 26, 105, 231
Толерантность (tolerance), 79
Точка (point)
Штейнера (Steiner), 444
неподвижная (fixed), 90, 260
наименьшая (least), 91
подвижная (movable), 260
сочленения (junction), 367
Траектория (trajectory), 119
Трансверсаль (transversal), 376
частичная (partial), 154
Транспозиция (transposition), 260
Транспонирование матрицы (transposition of matrix), 62
Треугольник (triangle), 60
Белла (Eric Temple Bell), 284
Паскаля (Blaise Paskal), 273
расширенный (extended), 277
Трудоёмкость алгоритма в среднем (complexity of algorithm on average), 268
Турнир (tournament), 348

У

Удаление (removal)
вершины (of vertex), 349
нулей (of zeros), 172
ребра (of edge), 349
Узел (node), 337
Укладка графа (graph flattering), 473
Улучшение перебора (improvement of exhaustive search), 464
Умножение (multiplication)
алгебраическое (algebraic)
нечётких множеств (of fuzzy sets), 103
вектора на скаляр (of vector on scalar), 139
матриц (of matrix), 62
Унар (unar), 110
Универсум (universe), 23, 24
Унификатор (unifier), 210
наиболее общий (most common), 210
общий (common), 210
Упаковка (packing), 324
Упорядоченная пара (ordered couple), 52
Уровень узла (level of node), 405

Ф

Фактор-граф (quotient graph), 386
Факториал (factorial)
двойной (double), 265
Факторизация (factorization), 78
функции (of function), 78
Фактормножество (factor set), 78
Форма (form)
дизъюнктивная (disjunctive), 172, 174
нормальная (normal), 113, 170
нормальная сокращённая (normal reduced)
дизъюнктивная (disjunctive), 176
совершенная нормальная (perfect normal)
дизъюнктивная (disjunctive), 170
конъюнктивная (conjunctive), 171
Форма записи (notation)
инфиксная (infix), 104
постфиксная (postfix), 104
префиксная (prefix), 104
Формализация (formalization), 206
Формализуемость (formalizability), 208
Формула (formula), 159, 160, 242
Бине (Binet), 295
Коши (Augustin Louis Cauchy), 273
Эйлера (Leonard Euler), 473
атомарная (atomic), 220
бескванторная (quantifier-free), 225
в предварённой форме (prenex form), 225
включений и исключений (inclusion-exclusion), 287
выполнимая (satisfiable), 200
замкнутая (closed), 221
истинная (true), 222
ложная (false), 222
невыполнимая (unsatisfiable), 200
общезначимая (valid), 200, 208, 223
ортогональная (orthogonal), 182
открытая (open), 223
пропозициональная (propositional), 199
противоречивая (contradictory), 208
пустая (empty), 245
равносильная (equivalent), 162
унифицируемая (unificated), 210
формальной теории (of formal theory), 206
элементарная (elementary), 242
Функционально эквивалентные (functionally equivalent), 230
Функциональный символ (functional symbol), 220
n-арный (n-ary), 220
n-местный (n-ary), 220
Функция (function), 70
k-значной логики (k-valued logic), 158
n аргументов (n arguments), 70
n-местная (n-ary), 70

вычислимая (computable), 230
Эйлера (Euler), 136
аддитивная (additive), 137
алгебры логики (of boolean algebra), 155
арифметическая (arithmetic)
вполне (totally), 137
биективная (bijective), 70
булева (boolean), 155
весовая (weight), 152
взаимно-однозначная (one-to-one), 71
возрастающая (increasing)
монотонно (monotone), 89
строго монотонно (strict monotone), 89
двойственная (dual), 165
дистрибутивная (distributive), 148
извлечения доказанного (of extraction proven), 240
индуцированная (inducted), 78
инъективная(injective), 70
линейная (linear), 193
мажоритарная (majority), 166
монотонная (monotone), 89, 193
строго (strict), 89
мультипликативная (multiplicative), 137
вполне (totally), 137
непрерывная по Скотту (Scott-continuous), 90
обратная (inverse), 71
отождествления (identification), 78
перехода к образам (image function), 72
перехода к прообразам (preimage function), 72
производящая (course-of-value), 293
самодвойственная (self–dual), 193
самодвойственная (self-daul), 166
симметрическая (symmetric), 168
сохраняющая 0 (keeping 0), 193
сохраняющая 1 (keeping 1), 193
сюръективная(surjective), 70
тотальная (total), 70
убывающая (decreasing)
монотонно (monotone), 89
строго монотонно (strict monotone), 89
универсальная (universal), 238
характеристическая (characteristic), 94
бинарная (binary), 94
множества (of set), 94
мультимножества (of multiset), 100
неотрицательно целочисленная (integer non-negative), 94
нечёткая (fuzzy), 94, 97
отношения (of relation), 95
хэш (hash), 420
частичная (partial), 70

Х

Характеристика канала (channel characteristic), 311
Характеристика(characteristic)
поля (of field), 143
Хорда (chord), 447

Ц

Центр графа (center of graph), 343
Цепочка (chain), 86
множеств (of sets), 255
полная (complete), 255
Цепочка (string), 26
Цепь (circuit), 341
аугментальная (augmenting), 382
вершинно-непересекающаяся (pairwise vertex-independent), 373
гамильтонва (hamilton), 454
простая (simple), 341
рёберно-непересекающаяся (pairwise edge-independent), 373
эйлерова (euler), 451
Цикл (cycle), 60, 260, 341
гамильтонов (hamilton), 454
простой (simple), 341, 445
тривиальный (trivial), 260
фундаментальный (fundamental), 447
эйлеров (euler), 451
Цифра (digit), 41
Цифровая подпись (digital signature), 332
Цифры (numerals)
римские (roman), 298

Ч

Частное (quotient), 126
неполное (partial), 126
Частный случай (instance)
набора формул (set of formulas), 210
наборов формул (set of formulas)
совместный (shared), 210
совместный (shared), 210
формулы (formula), 209
Часть графа (section of graph), 339
Числа (numbers)
взаимно простые (coprime), 128
Число (number)
Белла (Eric Temple Bell), 284
Каталана (Catalan), 295
Стирлинга (of Stirling)
первого рода (of first kind), 282
второго рода (of second kind), 281
Фибоначчи (Fibonacci), 294
вершинного покрытия (of vertex covering), 457
вершинное независимости (of vertex independence), 458
вещественное (real), 22
гармоническое (harmonic), 269
обобщённое (generalized), 269
инверсий (of inversions), 263
кликовое (clique), 347
коцикломатическое (cocyclomatic), 449
натуральное (natural), 21, 106
перестановок (of permutations), 255
простое (prime), 21, 130
псевдослучайное (pseudorandom), 329
рёберного покрытия (of edge covering), 458
рёберное независимости (of edge independence), 458
размещений (of partial permutations), 254
без повторений (without repetitions), 254

совершенное (perfect), 127
составное (composite), 130
сочетаний (of combinations), 256
с повторениями (of complete combinations), 257
треугольное (triangle), 30
хроматическое (chromatic), 468
цикломатическое (cyclomatic), 447
чётное (even), 109

Ш

Шар (ball), 314
Шифр (cipher), 328
надёжный (robust), 328
раскрытие (decoding), 328
с открытым ключом (public key), 330
симметричный (symmetric), 329
Шифрование (enciphering), 328
Шифровка (coded message), 328
Шкала битовая (bit scale), 38

Э

Эвристический (heuristic), 461
Эйлерова характеристика (Euler characteristic), 473
Эквивалентность логическая (logical equivalence), 201, 224
Экземпляр класса (instance of class), 351
Экономичность системы счисления (radix efficiency), 41
Эксцентриситет (eccentricity)
вершины (of vertex), 343
Электронная подпись (digital signature), 332
Элемент (element)
максимальный (maximal), 87
минимальный (minimal), 86
множества (of set), 21
наибольший (greatest), 87
наименьший (least), 86
нейтральный (neutral), 116, 118
аддитивный (additive), 138
мультипликативный (multiplicative), 138
обратный (inverse), 117
опорный (reference), 266
принадлежность множеству (membership), 22
Элиминация (elimination)
импликации (of implication), 245
квантора всеобщности (of universal quantifier), 246
квантора существования (of existential quantifier), 246
конъюнкции (of conjunction), 246
операции (of operation), 172
Эндоморфизм (endomorphism), 108
Эпиморфизм (epimorphism), 108

Я

Ядро (kernel), 188
графа (of graph), 466
орграфа (of digraph), 467
отношения (of relation), 61
Язык (language), 26
формальной теории (of formal theory), 206
Ярус (tier), 343
дерева (of tree), 405

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» предлагает профессиональную, популярную и детскую развивающую литературу

Заказать книги оптом можно в наших представительствах

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-83, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1, 6 этаж
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: hitsenko@piter.com

Екатеринбург: ул. Толедова, д. 43а; тел./факс: (343) 378-98-41, 378-98-42;
e-mail: office@ekat.piter.com; skype: ekat.manager2

Нижний Новгород: тел.: 8 930 712-75-13; e-mail: yashny@yandex.ru; skype: yashny1

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 277-89-66; e-mail: pitvolga@mail.ru,
pitvolga@samara-ttk.ru

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163; тел./факс: +37 517 208-80-01, 208-81-25;
e-mail: og@minsk.piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов:
тел./факс: (812) 703-73-72, (495) 234-38-15; e-mail: ivanovaa@piter.com
Подробная информация здесь: http://www.piter.com/page/avtoru

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок: *тел./факс: (812) 703-73-73; e-mail: sales@piter.com*

Заказ книг для вузов и библиотек:
тел./факс: (812) 703-73-73, доб. 6243; e-mail: uchebnik@piter.com

---

Заказ книг по почте: *на сайте www.piter.com; тел.: (812) 703-73-74, доб. 6216;*
e-mail: books@piter.com

Вопросы по продаже электронных книг: *тел.: (812) 703-73-74, доб. 6217;*
e-mail: kuznetsov@piter.com

Федор Александрович Новиков

Дискретная математика:
Учебник для вузов. 3-е издание
Стандарт третьего поколения

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Римицан*
Художник	*С. Маликова*
Корректор	*С. Беляева*
Верстка	*О. Орлов*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), д. 3, литер А, пом. 7Н.
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.11.000 — Учебники печатные общеобразовательного назначения.
Подписано в печать 05.10.16. Формат 70x100/16. Бумага писчая.
Усл. п. л. 39,990. Тираж 500. Заказ

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор» .
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
Факс: 8(496) 726-54-10, телефон: (495) 988-63-87