*Государственное образовательное учреждение высшего профессионального образования*
*«Московский государственный технический университет имени Н.Э. Баумана»*
*(МГТУ им. Н.Э. Баумана)*

УТВЕРЖДАЮ
Заведующий кафедрой ИУ6

_____В.В. Сюзев
«___» _____ 2011г.

# МЕТОДИЧЕСКИЕ УКАЗАНИЯ

## к лабораторным работам по дисциплине «Математические основы проектирования топологии многопроцессорных вычислительных систем»

Автор: Можаров Г.П.
Факультет – ИУ (каф. ИУ6)
Кафедра ИУ6 – «Компьютерные системы и сети»

Москва
2011

# СОДЕРЖАНИЕ

# Введение

Целью лабораторного практикума по математическим основам проектирования топологии многопроцессорных вычислительных систем является знакомство студентов с разработанными методами решения ряда прикладных и математических задач. Кроме того одна из целей – привлечь студентов к тематике экстремальных задач и к комбинаторике как к предмету исследований.

Теория графов за последние десятилетия развилась в весьма обширную ветвь математики, имеющую многочисленные приложения в самых разнообразных сферах человеческой деятельности. Подбор тем, затронутых в пособии, представляет основное, устоявшееся ядро современной теории графов и сопутствующее ему семейство алгоритмов дискретной оптимизации, наиболее часто используемых программистами.

Начало систематических комбинаторных исследований положено трудами Б. Паскаля и П. Ферма. Работы Я. Бернулли и Г. Лейбница способствовали выделению комбинаторики в самостоятельный раздел. Именно Г. Лейбниц осуществил первую попытку целостного осмысления комбинаторики в своей диссертации «*Ars Combinatoria*», откуда, по-видимому, и пошел термин «комбинаторика».

Эффективные комбинаторные алгоритмы находят применение во многих областях нечисленной обработки информации, особенно в дискретной оптимизации и в исследовании операций.

Основным объектным понятием комбинаторики является понятие соответствия. Комбинирование есть перебор соответствий между свойствами объектов с целью изучения их природы. Сложность такого перебора предопределяется взаимной зависимостью этих свойств. Предмет комбинаторики состоит в изучении соответствий и комбинаций простейших математических объектов – чисел, множеств и фигур. В методологической основе комбинаторики лежит комбинирование тремя атрибутными свойствами множества – различимостью, очередностью и целостностью. Это комбинирование порождает весь простейший комбинаторный инструментарий: различимость – мультимножество, очередность – перестановку, целостность – разбиение.

Простейший количественный анализ комбинаций и соединений составляет основу традиционного проблемного направления комбинаторики – перечислительные задачи. Развитие этого направления служит главным источником построения комбинаторного анализа. Исторически первым и общим для комбинаторного анализа явился метод производящих функций.

Иное проблемное направление комбинаторики составляют структурные задачи. Наибо-

лее явственно проявилось оно в теории графов, представляющее собой раздел комбинаторики, изучающий различного рода простейшие отношения на множествах и системах множеств. Понятие графа как системы двухэлементных подмножеств (ребер) некоторого множества (вершин) возникло и изучалось на основе его топологической природы. Выведенный Куратовским критерий планарности графа расширил представление о нем: граф может быть изображен на плоскости точками и соединяющими их линиями без пересечения последних тогда и только тогда, когда он не содержит подграфов, гомеоморфных графам $K_5$ и $K_{3,3}$. Это значит, что «топологичность» графа полностью определяется его теоретико-множественной структурой.

Расширение областей применения теоретических комбинаторных результатов приводит к зарождению важного проблемного направления – комбинаторного моделирования. При этом выбор наиболее подходящей комбинаторной трактовки прикладных задач определяется конечными целями их решения. Широкая степень абстракции каждой комбинаторной модели позволяет с их помощью исследовать некоторый определенный круг процессов или явлений из различных областей знаний. Следовательно, объединение таких моделей в комплексы, чей состав будет определяться путем нахождения правил соответствия между ними, которые, в свою очередь, будут зависеть от задач, решаемых с помощью таких комплексов моделей, существенным образом расширит области их применения. Это приводит к образованию еще одного проблемного направления – изучению соответствий между различными моделями. Основная цель, которая преследуется этим проблемным направлением: создание унифицированных комплексов комбинаторных моделей, пригодных для адекватного описания не только специализированных задач практики, но и для описания процессов и явлений, принадлежащих некоторому кругу предметных областей знаний.

Комбинаторика может служить практикой и теорией. В период становления она была практикой для теории вероятностей, подтверждая и подсказывая ее методы и законы; теорией выступала, решая задачи. Эта замечательная двойственность проявляется и в экстремальных задачах, которые являются не только рабочим инструментом решения чисто практических вопросов, но сами же характеризуют эффективность этого разрешения, являясь тем самым удобным мерилом основного критерия истинности – практики.

В настоящем пособии представлены некоторые разделы комбинаторики, причем особое внимание уделено конструктивному алгоритмическому подходу – рядом с обсуждаемыми комбинаторными проблемами, как правило, приводятся алгоритмы их решения вместе с анализом их вычислительной сложности.

Задача лабораторного практикума – ознакомить студентов с некоторыми алгоритмами,

используемыми при проектировании многопроцессорных ВС.

В первой и второй лабораторных работах рассматриваются перестановки.

Третья лабораторная работа посвящена генерированию подмножеств множества; четвертая – генерированию *k*-элементных подмножеств.

**Порядок выполнения и защиты лабораторных работ.** Для проведения лабораторной работы в классе ЦВМ студент должен:

• ознакомиться с теоретическими сведениями и алгоритмами, связанными с параллельными системами;

• изучить среду программирования, используемую для решения задач построения параллельных систем;

• получить свой вариант задания к лабораторной работе у преподавателя, ответственного за проведение лабораторных работ.

Во время выполнения лабораторной работы на ЦВМ необходимо:

• в соответствии с предлагаемым алгоритмом написать и отладить программу;

• для этой программы подготовить 2-3 тестовых примера, иллюстрирующих работу программы;

• обеспечить режим пошагового выполнения алгоритмов с выдачей на дисплей получаемых промежуточных результатов;

• во всех случаях, когда результатом работы программы являются графы или временные диаграммы, необходимо обеспечить их выдачу на дисплей, используя графический метод доступа.

Самостоятельная подготовка студентов к каждой лабораторной работе требует около четырех академических часов. Каждая работа выполняется в учебном классе ЦВМ университета в присутствии преподавателя в течение четырех академических часов. После выполнения очередной лабораторной работы необходимо получить у преподавателя отметку о выполнении. Отчет в соответствии с установленной формой представляется на очередном занятии. Отсутствие отчета может служить причиной недопуска студента к следующей лабораторной работе.

Залогом успешного выполнения лабораторных работ является самостоятельная подготовка студента. Для облегчения этой подготовки в описании каждой работы есть теоретическая часть, в которой приводятся основные понятия, используемые в данной лабораторной работе, а также требуемые алгоритмы. Имеются контрольные вопросы, с помощью которых можно оценить степень готовности студента к выполнению лабораторной работы.

Работа считается выполненной и зачтенной, если она была защищена. Защита лабораторной работы заключается в демонстрации работы программы на ЦВМ, ответах на вопросы, связанные с ее теоретическими аспектами и получаемыми численными характеристиками.

# Лабораторная работа № 1

# Перестановки: разложение на циклы и знак перестановки

**Цель работы** – ознакомление с одним из разделов комбинаторики – перестановками, а именно с методами разложения на циклы и определения знака перестановки.

## Теоретическая часть

**Определение 1.1.** Перестановкой *n*-элементного множества $X$ называется произвольная взаимно однозначная функция $f : X \to X$ . Число перестановок *n*-элементного множества равно $n!$

Предположим, что мы размещаем *n* объектов по *m* ячейкам так, чтобы каждый ячейка содержала бы последовательность, а не множество, как прежде, помещенных в ней объектов. Два размещения назовем равными, если в каждом ячейке содержится одна и та же последовательность объектов. Размещения такого типа будем называть *упорядоченными размещениями n объектов по m ячейкам*. Обозначим число таких упорядочений через $[m]^n$.

**Определение 1.2.** Число упорядоченных размещений *n* объектов по *m* ячейкам равно

$$[m]^n = m(m+1) \ \ldots \ (m+n-1)$$

(полагаем $[m]^0 = 1$).

Обычно перестановка определяется с помощью таблицы с двумя строками, из которых каждая содержит все элементы множества $X$ , причем элемент $f(x)$ помещается под элементом $x$. Рассмотрим такую перестановку $f$ множества $\{a,\ b,\ c,\ d\}$, что

$$f(a) = d; \quad f(b) = a; \quad f(c) = c; \quad f(d) = b;$$

она записывается в виде

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 2 & 1 & 4 \end{pmatrix}, \quad g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix},$$

Если порядок элементов в верхней строке фиксирован, то каждой перестановке однозначно соответствует последовательность, содержащаяся в нижней строке, например для перестановки $n$ это есть $\langle d,\ a,\ c,\ b \rangle$. Поэтому будем называть иногда произвольную инъективную последовательность длины $n$ с элементами из множества $X$ перестановкой $n$-элементного множества $X$ .

Примем для простоты $X = \{1, \ldots, n\}$. Обозначим множество всех перестановок этого множества через $S_n$. Произвольная перестановка $f \in S_n$ будет обычно отождествляться с последовательностью $\langle a_1, \ldots, a_n \rangle$, где $a_i = f(i)$. Под *суперпозицией* перестановок $f$ и $g$ мы будем понимать перестановку $fg$, определяемую следующим образом:

$$fg(i) = f(g(i)).$$

Отметим, что для суперпозиции двух перестановок, скажем

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 2 & 1 & 4 \end{pmatrix}, \quad g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix},$$

достаточно изменить порядок столбцов в перестановке $f$ таким образом, чтобы в первой строке получить последовательность, имеющуюся во второй строке перестановки $g$, тогда вторая строка перестановки $f$ дает суперпозицию $fg$. В нашем случае

$$g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 3 & 1 & 4 \end{pmatrix},$$

$$f = \begin{pmatrix} 2 & 5 & 3 & 1 & 4 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix},$$

$$fg = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 5 & 1 \end{pmatrix}.$$

Перестановка

$$e = \begin{pmatrix} 1 & 2 & \cdots & n \\ 1 & 2 & \cdots & n \end{pmatrix}$$

называется *тождественной перестановкой*. Очевидно, что $ef = fe = f$ для произвольной перестановки $f \in S_n$. Легко также заметить, что каждая перестановка $f \in S_n$ однозначно определяет перестановку $f^{-1}$, такую что $ff^{-1} = f^{-1}f = e$. Будем называть ее *перестановкой, обратной к* $f$. Чтобы ее определить, достаточно поменять местами строки в записи перестановки $f$. Например, для

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix}$$

получаем

$$f^{-1} = \begin{pmatrix} 3 & 4 & 2 & 1 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 1 & 2 & 5 \end{pmatrix}.$$

Из наших рассуждений следует, что для произвольных перестановок $f$, $g$, $h \in S_n$ выполняются условия

$$(fg)h = f(gh), \tag{1.6}$$

$$fe = ef = f, \tag{1.7}$$

$$f^{-1}f = ff^{-1} = e. \tag{1.8}$$

Чтобы отразить этот факт, будем говорить, что $S_n$ образует *группу* относительно операции суперпозиции. Эту группу будем называть *симметрической группой степени* $n$. Произвольное подмножество $G \subseteq S_n$, для которого выполнены условия

$$f,\ g \in G \Rightarrow fg \in G,$$
$$f \in G \Rightarrow f^{-1} \in G,$$

называется *группой перестановок степени* $n$.

Каждую перестановку $f \in S_n$ можно представить графически с помощью ориентированного графа с множеством вершин $X = \{1,\ ...,\ n\}$, в котором $x \to y$ тогда и только тогда, когда $f(x) = y$. Из каждой вершины $x$ выходит в точности одно ребро, а именно $\langle x,\ f(x)\rangle$. Подобным же образом убеждаемся, что единственным ребром, входящим в вершину $x_0$, является $\langle f^{-1}(x),\ x\rangle$. Легко заметить, что, выходя из произвольной вершины $x_0$ и рассматривая по очереди вершины $x_1 = f(x_0),\ x_2 = f(x_1),\ ...,$ мы дойдем после конечного числа шагов до вершины $x_0$, т.е. $x_l = f(x_{l-1}) = x_0$ для некоторого $l \geq 1$. Отсюда следует, что наш граф состоит из некоторого числа элементарных циклов с различными множествами вершин, в сумме дающих все множество $X$. Предположим, что в этом разложении появляются $k$ циклов

$$a_0^{(i)} \to a_1^{(i)} \to\ ...\ \to a_{n_i-1}^{(i)} \to a_0^{(i)},\quad i = 1,\ ...,\ k.$$

Каждому такому циклу соответствует перестановка

$$f_i = \left[a_0^{(i)} a_1^{(i)}\ ...\ a_{n_i-1}^{(i)}\right],$$

называемая также *циклом* (*длины* $n_i$), которая определяется следующим образом:

$$f_i\left(a_0^{(i)}\right) = a_1^{(i)},\ \ f_i\left(a_1^{(i)}\right) = a_2^{(i)},\ \ ...,\ \ f_i\left(a_{n_i-1}^{(i)}\right) = a_0^{(i)},$$
$$f_i(x) = x \quad \text{для} \quad x \in X\ ,\ \ \left\{a_0^{(i)},\ a_1^{(i)},\ ...,\ a_{n_i-1}^{(i)}\right\}.$$

Нашу перестановку можно представить в виде суперпозиции циклов

$$f = \left[a_0^{(1)} a_1^{(1)}\ ...\ a_{n_1-1}^{(1)}\right]\left[a_0^{(2)} a_1^{(2)}\ ...\ a_{n_2-1}^{(2)}\right]\ ...\ \left[a_0^{(k)} a_1^{(k)}\ ...\ a_{n_k-1}^{(k)}\right].$$

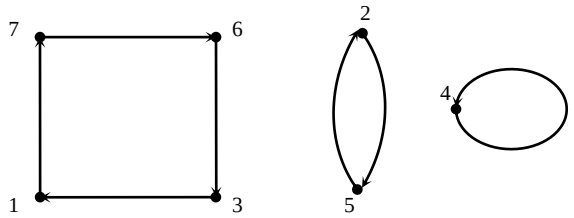


Рис. 1. 1. Разложение перестановки на циклы

Такое представление перестановки будем называть *разложением на циклы*. Будем говорить, что перестановка $f$ есть перестановка типа $\langle \lambda_1,\ ...,\ \lambda_n \rangle$, если она содержит в разложении на циклы в точности $\lambda_i$ циклов длины $i$, $i = 1, 2, ..., n$. Тип $\langle \lambda_1,\ ...,\ \lambda_n \rangle$ обычно записывается символически $1^{\lambda_1} ... n^{\lambda_n}$ (если $\lambda_i = 0$, то $i^{\lambda_i}$ опускается). Например, перестановка

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 5 & 1 & 4 & 2 & 3 & 6 \end{pmatrix}$$

имеет следующее разложение на циклы:

$$f = [1 \quad 7 \quad 6 \quad 3][2 \quad 5][4],$$

следовательно, она имеет тип $1^1 2^1 4^1$. Это проиллюстрировано на рис. 1.1. Пару $\langle a_i,\ a_j \rangle$, $i < j$, будем называть *инверсией* перестановки $\langle a_i,\ ...,\ a_n \rangle$, если $a_i > a_j$. Для произвольной перестановки $f \in S_n$ обозначим через $I(f)$ число ее инверсий, а также определим знак этой перестановки следующим образом:

$$\operatorname{sgn}(f) = (-1)^{I(f)}.$$

Перестановку $f$ назовем *четной*, если $\operatorname{sgn}(f) = 1$, и *нечетной*, если $\operatorname{sgn}(f) = -1$. Проиллюстрируем эти понятия на ряде примеров. Тождественная перестановка $\langle 1,\ ...,\ n \rangle$ не содержит никаких инверсий, и, следовательно, $I(e) = 0$, и для любого $n$ эта перестановка будет четной. Перестановка $\langle n,\ n-1,\ ...,\ 1 \rangle$ содержит $n(n-1)/2$ инверсий (число всех пар $\langle i,\ j \rangle$, $i \neq j$, равно $[n]_2 = n(n-1)$, причем каждой паре $\langle i,\ j \rangle$, $i < j$, соответствует пара $\langle j,\ i \rangle$, $j > i$). Таким образом, наша перестановка будет четной для $n$ вида $4k$ или $4k+1$ и нечетной в остальных случаях. Перестановка $\langle 2,\ 4,\ 3,\ 5,\ 1 \rangle$ имеет следующие инверсии: $\langle 2,\ 1 \rangle$, $\langle 4,\ 3 \rangle$, $\langle 4,\ 1 \rangle$, $\langle 3,\ 1 \rangle$, $\langle 5,\ 1 \rangle$, следовательно, является нечетной.

Знак перестановки можно определить с помощью непосредственного подсчета всех ин-

версий, однако такой алгоритм в общем случае требует количества шагов такого же порядка, что и число инверсий, т.е. по меньшей мере $\Omega(n^2)$. Опишем теперь алгоритм сложности $O(n)$. Для этой цели нам понадобится несколько утверждений.

Произвольную перестановку, являющуюся циклом длины 2, будем называть *транспозицией*. Важную роль в дальнейших рассуждениях будут играть транспозиции соседних элементов, т. е. транспозиции вида $[i \;\; i+1]$.

**Утверждение 1.3.** Произвольную перестановку $f \in S_n$ можно представить в виде суперпозиции $I(f)$ транспозиций соседних элементов.

Заметим, прежде всего, что если $f$ имеет вид $\langle a_1, \ldots, a_n \rangle$ и $t = [i \;\; i+1]$, то суперпозиция $ft$ имеет вид $\langle a_1, \ldots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \ldots, a_n \rangle$. Обозначим через $r_i$ число инверсий, расположенных перед элементом $i$:

$$r_i = \left| \{ j : j < i \wedge a_j > a_i \} \right|.$$

Легко заметить, что в $\langle a_1, \ldots, a_n \rangle$ мы можем переставить элемент $1 = a_{r_1} + 1$ на первую позицию, произведя $r_1$ транспозиций соседних элементов, затем элемент 2 переставить на вторую позицию, произведя $r_2$ транспозиций соседних элементов и т. д. В конце концов после $r_1 + \ldots + r_n = I(f)$ шагов получим последовательность $\langle 1, \ldots, n \rangle$. Это означает, что $ft_1 \ldots t_{I(f)} = e$, где $t_1, \ldots, t_{I(f)}$ – транспозиции соседних элементов. Итак,

$$f = \left(t_1 \ldots t_{I(f)}\right)^{-1} = t_{I(f)}^{-1} \ldots t_1^{-1},$$

что завершает доказательство, где $t^{-1} = t$ для произвольной транспозиции $t$.

**Утверждение 1.4.** Для произвольных перестановок $f$, $g \in S_n$

$$\operatorname{sgn}(fg) = \operatorname{sgn}(f)\operatorname{sgn}(g).$$

Положим сначала, что $g$ есть транспозиция вида $t = [i \;\; i+1]$. Если $f = \langle a_1, \ldots, a_n \rangle$, то

$$ft = \langle a_1, \ldots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \ldots, a_n \rangle$$

и, очевидно,

$$I(ft) = \begin{cases} I(f) + 1, & \text{если } a_i < a_{i+1}, \\ I(f) - 1, & \text{если } a_i > a_{i+1}. \end{cases}$$

В обоих случаях $\operatorname{sgn}(ft) = -(-1)^{I(f)} = -\operatorname{sgn}(f)$.. Произвольную перестановку $g$ мы можем на основе **утверждения 1.3** представить в виде $t_1 \ldots t_k$, где $t_1, \ldots, t_k$ суть транспозиции соседних элементов и $k = I(g)$. Имеем

$$\operatorname{sgn}(fg) = \operatorname{sgn}(ft_1 \ldots t_k) = -\operatorname{sgn}(ft_1 \ldots t_{k-1}) = (-1)^k \operatorname{sgn}(f) = \operatorname{sgn}(g)\operatorname{sgn}(f).$$

**Утверждение 1.5.** Каждая, транспозиция есть нечетная перестановка. Вообще знак произвольного цикла длины $k$ равен $(-1)^{k-1}$.

Первая часть вытекает из того факта, что последовательность

$$\langle 1,\ \ldots,\ i-1,\ \ j,\ i+1,\ \ldots,\ j-1,\ i,\ \ j+1,\ \ldots,\ n \rangle$$

можно преобразовать в последовательность $\langle 1,\ \ldots,\ n \rangle$, произведя сперва $j-i$ транспозиций

$$[\,j-1 \quad j\,],\ [\,j-2 \quad j-1\,],\ \ldots,\ [\,i \quad i+1\,],$$

а затем $(j-i)-1$ транспозиций

$$[\,i+1 \quad i+2\,],\ [\,i+2 \quad i+3\,],\ \ldots,\ [\,j-1 \quad j\,].$$

Это означает, что транспозиция $[ij]$ может быть представлена в виде суперпозиции $(j-i)+(j-i)-1 = 2(j-i)-1$ транспозиций соседних элементов. Согласно предыдущим результатам, знак нашей транспозиции $[ij]$ равен $(-1)^{2(j-i)-1} = -1$.

Вторая часть есть следствие первой и того факта, что произвольный цикл $[\,a_1\ \ldots\ a_k\,]$ есть суперпозиция $k-1$ транспозиций;

$$[\,a_1\ \ldots\ a_k\,] = [a_1 a_2][a_2 a_3]\ \ldots\ [a_{k-1} a_k].$$

**Утверждение 1.6.** Знак произвольной перестановки $f$ типа $1^{\lambda_1}\ldots n^{\lambda_n}$ определяется формулой

$$\operatorname{sgn}(f) = (-1)^{\sum_{j=1}^{\lfloor n/2 \rfloor} \lambda_{2j}}.$$

Если $f$ имеет в разложении на циклы точно $\lambda_i$ циклов длины $i$, то, используя предыдущие результаты, получим

$$\operatorname{sgn}(f) = \prod_{i=1}^{n} \left[(-1)^{i-1}\right]^{\lambda_i} = \prod_{j=1}^{\lfloor n/2 \rfloor} (-1)^{\lambda_{2j}} = (-1)^{\lambda_2+\lambda_4+\ldots}.$$

Заметим, что о числе инверсий перестановки множества $X$ мы можем говорить только в случае $X = \{1,\ \ldots,\ n\}$, более общо: когда на множестве $X$ определен линейный порядок. Однако знак перестановки зависит только от ее типа и, следовательно, не зависит от того, как упорядочено множество $X$.

**Алгоритм 1.1. Определение знака перестановки.**

*Данные*: Произвольная перестановка $f \in S_n$, заданная в виде последовательности $P[1],\ \ldots,\ P[n]$ $(P[i] = f(i))$.

*Результат*: По завершении работы алгоритма $s = \operatorname{sgn}(f)$.

```
1  begin
2   s:=l;
3     for i:=l to n do НОВЫЙ [i] := истина;
4     for i := 1 to n do
5        if НОВЫЙ [i] then ( * найден цикл, содержащий i * )
6          begin j:=P[i];
7              while j≠ i do
8                 begin НОВЫЙ [j]:=ложь; s:= – s;  j:= P[j]
9                 end
10         end
11 end
```

Этот алгоритм просматривает последовательно позиции $1,\ \ldots,\ n$ перестановки (цикл 4) и каждый раз, когда элемент $P[i]$ не был еще проанализирован (НОВЫЙ [$i$] := **истина**), вы-

является цикл, к которому этот элемент принадлежит (блок 6). Заметим, что если этот цикл имеет длину $k$, то цикл 7 исполняется $k-1$ раз. При последовательных исполнениях этого цикла знак переменной $s$ изменяется на противоположный тогда и только тогда, когда $k$ четно. Первоначальное значение переменной $s$ равно единице, и, следовательно, после обнаружения всех циклов $s=(-1)^p$, где $p$ есть число циклов четной длины. Согласно **утверждению 1.6** имеем $s=\mathrm{sgn}(f)$.

Легко также убедиться, что число шагов алгоритма в точности равно $n$ для произвольной перестановки $s=\mathrm{sgn}(f)$. Чтобы это доказать, заметим, что суммарное число шагов, исполняемых в цикле 4, не считая шагов во внутреннем блоке 6, есть $O(n)$. Суммарное число шагов, исполняемых в блоке 6 в процессе работы алгоритма, равно сумме длин всех циклов, что в свою очередь равно $O(n)$. Это дает общую сложность $O(n)$.

## Вопросы для самопроверки

1. Что такое перестановка и как она определяется ?
2. Какое представление перестановки называется разложением на циклы ?
3. Что понимается под инверсией перестановки ?
4. Чем симметрическая группа степени $n$ отличается от группы перестановок степени $n$.
5. Что называется транспозицией ?

## Задание на лабораторную работу

1. Для произвольной перестановки $f \in S_n$, заданной в виде последовательности $P[1], \ldots, P[n]$ $(P[i]=f(i))$, написать и отладить программу (согласно с **алгоритмом 1.1**), по завершении которой определяется знак перестановки.
2. Продемонстрировать работающую программу преподавателю и получить отметку о ее выполнении.
3. Провести анализ полученного алгоритма.
4. Оформить отчет о проделанной работе. Он должен включать в себя:

– цель работы;

– ответы на вопросы для самопроверки;

– схему обрабатывающего алгоритма и описание его работы;

– распечатки экранных форм, полученных в результате работы программы.

# Лабораторная работа № 2

# Генерирование перестановок

**Цель работы** – получить программы: генерирования перестановок в антилексикографическом порядке, генерирование всех перестановок за минимальное число транспозиций и с минимальным числом транспозиций соседних элементов.

## Теоретическая часть

Опишем алгоритмы генерирования всех $n!$ перестановок $n$-элементного множества. Они имеют давнюю историю, ее возникновение можно отнести к началу XVII века, когда в Англии зародилось особое искусство колокольного боя, основанного, если говорить упрощенно, на выбивании на $n$ разных колоколах всех $n!$ перестановок. Перестановки эти следовало выбивать «по памяти», что способствовало разработке сторонниками этого искусства первых простых методов систематического перечисления всех перестановок (без повторений). Некоторые из этих незаслуженно забытых методов были переоткрыты в настоящее время в связи с появлением цифровых машин.

Опишем три разных метода генерирования последовательности всех $n!$ перестановок $n$-элементного множества. Будем предполагать, что элементы нашего множества запоминаются в виде элементов массива $P[i],\ \ldots,\ P[n]$. Во всех трех методах элементарной операцией, которая применяется к массиву $P$, является поэлементная транспозиция, т. е. обмен значениями переменных $P[i]$ и $P[j]$, где $1 \le i, j \le n$. Эту операцию будем обозначать через $P[i] :=: P[j]$. Очевидно, что она эквивалентна последовательности команд

$$\mathrm{pom} := P[i];\ \ P[i] :=: P[j];\ \ P[j] := \mathrm{pom},$$

где $\mathrm{pom}$ есть некоторая вспомогательная переменная.

Первый из методов, который мы опишем, легче всего понять, если в качестве переставляемых элементов взять числа $1,\ 2,\ \ldots,\ n$. На множестве всех перестановок – более общо: на множестве всех последовательностей длины $n$ с элементами из множества $X = \{1,\ 2,\ \ldots,\ n\}$ – определяется *лексикографический порядок*:

$$\langle x_1,\ x_2,\ \ldots,\ x_n \rangle < \langle y_1,\ y_2,\ \ldots,\ y_n \rangle \Leftrightarrow \text{существует } k \ge 1,$$

$$\text{такое что } x_k \le y_k \text{ и } x_l = y_l \text{ для каждого } l < k.$$

Отметим, что если вместо чисел $1,\ 2,\ \ldots,\ n$ взять буквы $a,\ b,\ \ldots,\ z$ с естественным порядком $a < b < \ \ldots\ < z$, то лексикографический порядок определяет стандартную последо-

вательность, в которой слова длины $n$ появляются в словаре. Подобным же образом определяется *антилексикографический порядок*, обозначаемый через $<'$, с той разницей, что как очередность позиций в последовательности, так и упорядочение элементов множества $X$ обратны по отношению к исходным:

$$\langle x_1,\ x_2,\ ...,\ x_n \rangle <' \langle y_1,\ y_2,\ ...,\ y_n \rangle \Leftrightarrow \text{существует такое } k \le n,$$

$$\text{что } x_k > y_k \text{ и } x_l = y_l \text{ для каждого } l > k.$$

Для примера приведем перестановки множества $X = \{1,\ 2,\ 3\}$ в лексикографическом (*а*) и антилексикографическом (*б*) порядке:

| (*а*) | (*б*) |
|---|---|
| 1 2 3 | 1 2 3 |
| 1 3 2 | 2 1 3 |
| 2 1 3 | 1 3 2 |
| 2 3 1 | 3 1 2 |
| 3 1 2 | 2 3 1 |
| 3 2 1 | 3 2 1 |

Алгоритм генерирования перестановок в антилексикографическом порядке сформулировать немного удобнее. Заметим с этой целью, что последовательность перестановок множества $\{1,\ 2,\ ...,\ n\}$ в этом случае имеет следующие свойства, вытекающие непосредственно из определения:

(А1) В первой перестановке элементы идут в растущей последовательности, в последней – в убывающей; другими словами, последняя перестановка – обращение первой.

(А2) Нашу последовательность можно разделить на $n$ блоков длины $(n - 1)!$, соответствующих убывающим значениям элемента в последней позиции. Первые $n - 1$ позиций блока, содержащего элемент $p$ в последней позиции, определяют последовательность перестановок множества $\{1,\ 2,\ ...,\ n\}$, $\{\mathbb{p}\}$ в антилексикографическом порядке.

Используя эти свойства определим рекурсивный алгоритм генерирования всех последовательностей в антилексикографическом порядке.

**Алгоритм 2.1. Генерирование всех последовательностей в антилексикографическом порядке.**

*Данные*: $n$.

*Результат*: Последовательность перестановок множества $\{1,\ 2,\ ...,\ n\}$ в антилексикографическом порядке.

```
1  procedure REVERSE (m);  (* обращение последовательности P[1], ..., P [m]; массив P – глобальный*)
2  begin i:= 1; j:= m;
```

```
3     while i < j do
4       begin P[i]:=:P[j]; i:=i +1; j := j – 1
5       end
6   end; (* REVERSE*)
7   procedure ANTYLEX (m); (* массив P – глобальный *)
8   begin
9     if m=l  then  (*P[1], ..., P[n] содержит  новую  перестановку *)
10        write (P[1], .... P[n])
11    else
12      for i:= 1 to m do
13        begin ANTYLEX (m – 1);
14          if i < m then
15            begin P[i]:=:P[m]; REVERSE (m – 1)
16            end
17        end
18  end; (*ANTYLEX*)
19  begin (*главная программа*)
20    for i:=1 to n do  P[i]:=:i;
21    ANTYLEX(n)
22  end
```

Чтобы понять, как работает этот алгоритм, отметим прежде всего, что выполнение процедуры REVERSE ($m$) приводит к обращению очередности в последовательности элементов $P[1], ..., P[m]$. Чтобы доказать правильность алгоритма, достаточно показать индукцией по $m$, что если $P[1] < ... < P[m]$, то вызов ANTYLEX($m$) приводит к генерированию всех перестановок множества $\{P[1], \ldots, P[m]\}$ в антилексикографическом порядке (при неизменных значениях $P[m + 1], \ldots, P[n]$). Предположим, что $P[i] = a_i, \quad 1 \le i \le m$, $a_1 < \ldots a_m$, и рассмотрим цикл 12. Эффект выполнения первой итерации этого цикла будет следующий:

| $P[1]$ | $P[2]$ | … | $P[m-2]$ | $P[m-1]$ | $P[m]$ | |
|---|---|---|---|---|---|---|
| $a_1$ | $a_2$ | … | $a_{m-2}$ | $a_{m-1}$ | $a_m$ | |
| $a_{m-1}$ | $a_{m-2}$ | … | $a_2$ | $a_1$ | $a_m$ | (после выполнения ANTYLEX ($m-1$)) |
| $a_m$ | $a_{m-2}$ | … | $a_2$ | $a_1$ | $a_{m-1}$ | (после транспозиции $P[i]:=:P[m]$) |
| $a_1$ | $a_2$ | … | $a_{m-2}$ | $a_m$ | $a_{m-1}$ | (после выполнения *REVERSE* ($m-1$)) |

(Мы воспользовались здесь индуктивным предположением о том, что ANTYLEX($m$ – 1) корректно генерирует все перестановки элементов $a_1, ..., a_{m-1}$ в антилексикографическом порядке.) Аналогичным способом, индукцией по $i$, доказываем, что $i$-я итерация цикла 12 приводит к генерированию всех перестановок элементов $a_1, ..., a_{m-i}, a_{m-i+2}, ..., a_m$ при $P[m] = a_{m-i+1}$. Согласно свойству А2 это означает, что ANTYLEX($m$) генерирует все перестановки элементов $a_1, ..., a_m$ в антилексикографическом порядке.

Следует отметить, что условие $P[i] = i, \ 1 \le i \le n,$ в начале работы программы (см. строку 20) было существенно только для облегчения понимания работы алгоритма. Сам алго-

ритм сформулирован в терминах позиций, значения которых подвергаются изменениям, и в нем ни в коей мере не используется содержание переменных $P[1], ..., P[n]$.

Рассмотрим теперь, сколько транспозиций выполняет алгоритм генерирования всех последовательностей в антилексикографическом порядке при генерировании каждой следующей перестановки. Легко отметить, что это число есть величина переменная: каждая вторая перестановка получилась за счет одной транспозиции $P[1]:=:P[2]$; но наряду с ними имеются и такие, которые требуют $\lfloor (n-1)/2 \rfloor + 1 = \lfloor (n+1)/2 \rfloor$ транспозиций. Хотя среднее число транспозиций, приходящихся на каждую перестановку, невелико, однако в некоторых приложениях лучшим был бы алгоритм, в котором каждая следующая перестановка образуется из предыдущей с помощью выполнения только одной транспозиции. Это может оказаться существенным в ситуации, когда с каждой перестановкой связаны некоторые вычисления и когда существует возможность использования частичных результатов, полученных для предыдущей перестановки, если последовательные перестановки мало отличаются друг от друга.

Покажем сейчас, что такой алгоритм действительно возможен. Его основную схему можно описать с помощью следующей рекурсивной процедуры:

```
1  procedure PERM (m); (* массив P – глобальный *)
2  begin
3     if  m = 1  then  (*P[1], ..., P[n] содержит  новую перестановку *)
4        write (P[1], ..., P[n])
5     else
6        for i := 1 to m do
7          begin PERM(m – 1);
8              if i < m then P[B [m, i]] :=: P[m]
9           end
10 end
```

Задачей этой процедуры является генерирование всех перестановок элементов $P[1], ..., P[n]$ через последовательное генерирование всех перестановок элементов $P[1], ,.., P[n-1]$ и замену элемента $P[n]$ на один из элементов $P[1], ..., P[n-1]$, определяемый из массива $B[m, i]$, $1 \le i < m \le n$. Очевидно, что для того чтобы эта схема работала правильно, мы должны определить массив $B$ так, чтобы гарантировать, что каждая транспозиция $P[B[m, i]]: = :P[m]$ в строке 8 вводила бы новый элемент в $P[m]$. Представим теперь алгоритм, в котором значение $B[m, i]$ вычисляется динамически как функция от $m$ и $i$.

**Алгоритм 2.2. Генерирование всех перестановок за минимальное число транспозиций.**

*Данные*: $n$.

*Результат*: Последовательность перестановок множества $\{1, ..., n\}$, в котором каждая

последующая перестановка образуется из предыдущей путем выполнения одной транспозиции.

```
1   procedure B (m, i);
2   begin
3     if (m mod 2 = 0) and (m > 2) then
4        if i < m – 1 then B : = i
5       else B:= m – 2,
6     else B:= m – 1
7   end; (*B*)
8   procedure PERM(m); (* массив P – глобальный*)
9   begin
10  if  m–1   then  (*P[1], ..., P[n] – новая перестановка*)
11     write (P[l], ..., P[n])
12     else
13       for i:= l to m do
14          begin PERM (m – 1);
15            if i < m then P[B(m, i)]:=:P[m]
16          end
17   end; (*PEPM*)
18   begin (* главная программа*)
19     for i := 1 to n do   P[i] :=: i;
20     PERM (n)
21  end
```

Отметим, что для нечетного $m$ транспозиция в строке 15 сводится к $P[m-1]$: =: $P[m]$, для каждого $i < m$ для четного $m$ значение $P[m]$ меняется последовательно на значения $P[1], P[2], ..., P[m-3], P[m-2], P[m-1]$ ($P[1]$ для $m = 2$).

Для каждого $m \geq 1$ определим перестановку $\varphi_m$ следующим образом:

$\varphi_m(i)$ = индекс $j$, такой что $P[j]$ содержит начальное значение переменной $P[i]$ после выполнения $PERM(m)$.

Например, если сначала переменные $P[1], ..., P[4]$ содержат последовательность 1 2 3 4, то легко убедиться, что после выполнения $PERM(4)$ эта последовательность изменится на 4 1 2 3. Это означает, что $\varphi_4$ является циклом 1 2 3 4.

Покажем теперь, что **алгоритм 2.2** корректен; точнее, докажем для каждого $m \geq 1$ выполнимость следующего условия:

*Условие* $W_m$. Выполнение $PERM[m]$ вызывает генерирование всех перестановок элементов $P[1], .,., P[m]$, причем фот есть транспозиция [ $m$  $m-1$], если $m$ нечетное ($m > 1$), или цикл [1 2 ... $m$], если $m$ четно.

*Доказательство* осуществляется индукцией по $m$ так, как это обычно делается при доказательстве корректности рекурсивных алгоритмов. Легко убедиться непосредственно, что условия $W_1$ и $W_2$ выполняются. Предположим, что $m \geq 3$. Докажем выполнимость $W_m$, предполагая истинность $W_{m-1}$. Доказательство разобьем на две части в зависимости от того, четное $m$ или нечетное,

*Случай* 1, *m нечетное*. В силу индуктивного предположения выполнение *PERM*(*m* – 1) в строке 14 приводит каждый раз к сдвигу значений *P*[1], ..., *P*[*m* – 1] вдоль цикла [1 2 ... *m* – 1]. Таким образом, транспозиция *P*[*m*] : = : *P*[*m* – 1) в строке 15 выбирает каждый раз различные элементы в *P*[*m*]. Если вначале *P*[*i*] = $a_i$, 1 ≤ *i* ≤ *m*, то в *P*[*m*] помещаются поочередно элементы $a_m$, $a_{m-2}$, $a_{m-3}$, ..., $a_1$, $a_{m-1}$. Следовательно, *PERM*(*m*) генерирует все перестановки элементов *P*[1], ..., *P*[*m*].

Заметим, что сдвиг *P*[1], ..., *P*[*m* – 1] вдоль цикла [1, 2, ... , *m* – 1], а затем выполнение транспозиции *P*[*m*]: =: *P*[*m* – 1] эквивалентно сдвигу *P*[1], ..., *P*[*m*] вдоль цикла [1 2 ... *m* – 3 *m* – 2 *m* *m* – 1] длины *m*. Если бы транспозиция в строке 15 выполнялась для каждого *i* ≤ *m*, то выполнение цикла 13 вызвало бы возвращение всех элементов на свои исходные места. В действительности же последняя транспозиция для *i* = *n* не выполняется. Следовательно, $\varphi_m = \left[ m,\ m-1 \right]$.

*Случай* 2, *m четное*. Проследим за содержанием массива *P* во время исполнения *PERM*(*m*). Изменения, которым подвергается он, представлены в табл. 2.1.

Табл. 2.1. Изменение значений переменных *P*[1], ..., *P*[*m*] во время выполнения процедуры *PERM* (*m*) при *m* четном

| Число выполненных итераций цикла 13 | *P*[1] | *P*[2] | *P*[*m* – 3] | *P*[*m* – 2] | *P*[*m* – 1] | *P*[*m*] |
|---|---|---|---|---|---|---|
| 0 | $a_1$ | $a_2$ | $a_{m-3}$ | $a_{m-2}$ | $a_{m-1}$ | $a_m$ |
| 1 | $a_m$ | $a_2$ | $a_{m-3}$ | $a_{m-1}$ | $a_{m-2}$ | $a_1$ |
| 2 | $a_m$ | $a_1$ | $a_{m-3}$ | $a_{m-2}$ | $a_{m-1}$ | $a_2$ |
| . | . | . | . | . | . | . |
| *m* – 3 | $a_m$ | $a_1$ | $a_{m-4}$ | $a_{m-1}$ | $a_{m-2}$ | $a_{m-3}$ |
| *m* – 2 | $a_m$ | $a_1$ | $a_{m-4}$ | $a_{m-3}$ | $a_{m-1}$ | $a_{m-2}$ |
| *m* – 1 | $a_m$ | $a_1$ | $a_{m-4}$ | $a_{m-2}$ | $a_{m-3}$ | $a_{m-1}$ |
| *m* | $a_m$ | $a_1$ | $a_{m-4}$ | $a_{m-3}$ | $a_{m-2}$ | $a_{m-1}$ |

Из этой таблицы следует, что каждый элемент появляется в *P*[*m*], следовательно, *PERM*(*m*) генерирует все перестановки, и что $\varphi_m$ является циклом [1 2 … *m*]. Доказательство правильности алгоритма тем самым закончено.

**Алгоритм 2.2** обладает одним свойством, которое может оказаться полезным в некоторых приложениях. Представим себе, что мы ищем перестановку, удовлетворяющую определенным условиям. В такой ситуации, если для некоторого *m* значения *P*[*m* + 1] ... *P*[*n*] не удовлетворяют требуемым ограничениям, то нет необходимости в вызове *PERM* (*m*) и в прохождении всех *m*! перестановок элементов *P*[1], ..., *P*[*m*]. Эти перестановки мы можем опустить, выполняя перестановку элементов *P*[1], ..., *P*[*m*], обозначаемую через $\varphi_m$. Заметим, что перестановки $\varphi_m$ имеют в нашем случае очень простую форму.

Третий алгоритм генерирования перестановок строит последовательность, в которой разница между двумя последовательными перестановками еще меньше: каждая следующая образуется из предыдущей с помощью однократной транспозиции соседних элементов. Этот алгоритм обычно приписывается Джонсону [] и Троттеру []. Его идею легче всего проиллюстрировать на примере. Предположим, что уже построена последовательность перестановок элементов 2, 3, …, *n*, обладающая этим свойством, например: 2 3, 3 2 для *n* – 3. Тогда требуемую последовательность перестановок элементов 1, 2, …, *n* получим, вставляя элемент 1 всеми возможными способами в каждую перестановку элементов 2, 3, …, *n*. В нашем случае получаем

1 2 3
2 1 3
2 3 1
3 2 1
3 1 2
1 3 2

В общем виде элемент 1 перемещается между первой и последней позициями попеременно вперед и назад $(n-1)!$ раз.

На основе этой конструкции можно легко получить рекурсивный алгоритм, генерирующий требуемую последовательность перестановок для произвольного *n*. Однако этот метод, примененный непосредственно, имеет недостаток: последовательность перестановок строится «целиком» и только после окончания всего построения ее можно считывать. Очевидно, что решение такого типа потребовало бы огромного объема памяти. Поэтому мы сейчас покажем нерекурсивный вариант этого алгоритма, В этом варианте для каждого $i$, $1 \le i < n$, булева переменная PR[$i$] содержит информацию о том, переносится ли элемент $i$ вперед (PR[$i$] = истина) или же назад (PR[$i$] = ложь), переменная же C[$i$] показывает, какую из возможных $n - i + 1$ позиций элемент $i$ занимает относительно элементов $i + 1, \ldots, n$ на своем пути вперед или назад. Позицию элемента $i$ в таблице $P$ определяем на основании его позиции в блоке, содержащем $i, i + 1, \ldots, n$, а также на основании числа элементов из 1, 2, ... , $i - 1$, которые находятся слева от этого блока. Это число, будучи значением переменной $x$, вычисляется как число элементов $j < i$, которые, двигаясь назад, достигли бы своего крайнего левого положения ($C[j] = n - j + 1$, PR[$i$] = ложь). Каждая новая перестановка образуется транспозицией самого меньшего из элементов $j$, который не находится в граничном положении (т. е. $C[j] < n - j + 1$) с его левым или правым соседом.

Это реализует приведенный ниже алгоритм. Доказательство его правильности предоставляется читателю.

**Алгоритм 2.3. Генерирование всех перестановок с минимальным числом транспо-**

**зиций соседних элементов.**

*Данные*: *n*.

*Результат*: Последовательность перестановок множества {1, ..., *n*}, в которой каждая последующая образуется в результате выполнения однократной транспозиции соседних элементов.

```
1  begin
2    for i := 1 to n do
3       begin P [i]:= i; C[i]:=1; PR[i] := истина;
4       end;
5    C[n]: = 0; (*так, чтобы C[i] ≠ n – i + l в строке 10 для i = n *}
6    write (P[l], ..., P[n]);
7    i : = l;
8    while i < n do
9      begin i:= 1; x:= 0;
10         while C[i] = n –i  + 1  do
11              begin PR [i] : = not PR [i]; C [i] := 1:
12                 if PR[i] then x: = x – 1;
13         i:= i +1
14            end;
15          if i < n then
16            begin (* выполнение транспозиции *)
17               if PR [i]  then k : = C[i] + x
18              else k : = n – i + 1 – C[i] + x;
19              P [k]:=:P[k + 1];
20               write (P[l], ..., P[n];
21            C[i]: = C[i] + 1
22            end
23     end
24  end
```

Отметим, что в этом алгоритме, как и в двух предыдущих, не используется ни в какой мере знание значений переменных *P*[1], ..., *P*[*n*]. Это очевидно, так как данные переменные появляются только в строке 19 (если не считать инициализации в строке 3 и вывода результатов).

На рис. 2.1 представлены последовательности перестановок, полученные для *n* = 4 при помощи **алгоритмов 2.1**, **2.2** и **2.3**.

| (*а*) | (*б*) | (*в*) |
|---|---|---|
| 1234 | 1234 | 1234 |
| 2134 | 2134 | 2134 |
| 1324 | 2314 | 2314 |
| 3124 | 3214 | 2341 |
| 2314 | 3124 | 3241 |
| 3214 | 1324 | 3214 |
| 1243 | 4321 | 3124 |
| 2143 | 3421 | 1324 |
| 1423 | 3241 | 1342 |
| 4123 | 2341 | 3142 |
| 2413 | 2431 | 3412 |
| 4213 | 4231 | 3421 |
| 1342 | 4132 | 4321 |
| 3142 | 1432 | 4312 |
| 1432 | 1342 | 4132 |
| 4132 | 3142 | 1432 |

| | | |
|---|---|---|
| 3412 | 3412 | 1423 |
| 4312 | 4312 | 4123 |
| 2341 | 4213 | 4213 |
| 3241 | 2413 | 4231 |
| 2431 | 2143 | 2431 |
| 4231 | 1243 | 2413 |
| 3421 | 1423 | 2143 |
| 4321 | 4123 | 1243 |

Рис. 2.1. Последовательности перестановок, полученные с помощью *а*) алгоритма 2.1; *б*) алгоритма 2.2; *в*) алгоритма 2.3

## Вопросы для самопроверки

1. Что понимается под лексикографическим и антилексикографическим порядками ?

2. Сколько транспозиций выполняет алгоритм генерирования всех последовательностей в антилексикографическом порядке при генерировании каждой следующей перестановки?

3. Запишите последовательность перестановок, полученную для $n = 3$ при помощи алгоритма генерирования всех перестановок за минимальное число транспозиций.

4. Проиллюстрируйте (для $n = 3$ и $n = 4$), что последовательность перестановок, полученная при помощи алгоритма генерирования всех перестановок с минимальным числом транспозиций соседних элементов, соответствует гамильтонову пути в $G_n$, т.е. пути, содержащему каждую вершину графа в точности один раз.

## Задание на лабораторную работу

1. По заданным **алгоритмам 2.1**, **2.2** и **2.3** написать и отладить программы для генерирования всех последовательностей в антилексикографическом порядке, генерирования всех перестановок за минимальное число транспозиций и генерирования всех перестановок с минимальным числом транспозиций соседних элементов.

2. Продемонстрировать работающую программу преподавателю и получить отметку о ее выполнении.

3. Провести анализ полученного алгоритма.

4. Оформить отчет о проделанной работе. Он должен включать в себя:

– цель работы;

– ответы на вопросы для самопроверки;

– схему обрабатывающего алгоритма и описание его работы;

– распечатки экранных форм, полученных в результате работы программы.

# Лабораторная работа № 3.
# Подмножества множества, множества с повторениями, генерирование подмножеств множества

Каждое *n*-элементное множество $X = \{x_1, x_2, ..., x_n\}$ имеет в точности $2^n$ подмножеств. Чтобы убедиться в этом, достаточно каждому подмножеству $Y \subseteq X$ сопоставить бинарную последовательность (т.е. с членами 0 или 1) $b_1 b_2 ... b_n$, определяемую следующим образом:

$$b_i = \begin{cases} 0, & \text{åñëè } x_i \notin Y, \\ 1, & \text{åñëè } x_i \in Y. \end{cases}$$

Тем самым мы устанавливаем взаимно однозначное соответствие между элементами множества $\mathrm{P}(X)$ всех подмножеств множества $X$ и всеми бинарными последовательностями длины *n*. Число таких последовательностей в точности равно $2^n$, а значит в силу установленного соответствия имеет место равенство $|\mathrm{P}(X)| = 2^n$.

Определенная последовательность $b_1 b_2 ... b_n$ становится удобным машинным представлением подмножества *Y*, особенно в ситуации, когда мощность множества *X* невелика и последовательность $b_1 b_2 ... b_n$ может быть закодирована в виде одного машинного слова. Такое представление подсказывает простой метод генерирования всех подмножеств. Достаточно заметить, что каждая бинарная последовательность $b_{n-1} b_{n-2} ... b_0$ соответствует взаимно однозначному целому числу из интервала $0 \le r \le 2^n - 1$, а именно числу $r = \sum_{i=0}^{n-1} b_i 2^i$, для которого $b_{n-1} b_{n-2} ... b_0$ есть двоичное представление. Это число будем обозначать через $[b_{n-1} b_{n-2} ... b_0]$. Таким образом, мы можем последовательно получать все числа из интервала $0 \le r \le 2^n - 1$ (начиная, например, с 0 и добавляя 1 на каждом шаге), а их двоичные представления дадут все подмножества *n*-элементного множества. Этот метод особенно выгоден для реализации на внутреннем языке машины.

В некоторых ситуациях нам важно, чтобы каждое последующее полученное подмножество наименьшим образом отличалось от предыдущего. Объяснение полезности алгоритма такого типа совершенно аналогично случаям алгоритмов генерирования всех перестановок за минимальное число транспозиций и с минимальным числом транспозиций соседних элементов: при проведении реальных вычислений, связанных с каждым получен-

ным подмножеством, имеется возможность использовать частичные результаты, полученные для предыдущего подмножества. Заметим, что в случае описанного выше алгоритма, основанного на двоичном представлении чисел, последовательно получаемые подмножества могут сильно отличаться друг от друга: например, после $(n-1)$-элементного множества, отвечающего числу 011 ... 1, идет одноэлементное множество, отвечающее числу 100 ... 0.

Опишем теперь другой метод, при котором каждое последующее подмножество получается из предыдущего добавлением или удалением одного элемента. Опирается он на следующее простое наблюдение. Если последовательность $\tilde{N}_1,\ \tilde{N}_2,\ ...,\ \tilde{N}_m$ содержит все $m = 2^k$ бинарных последовательностей длины $k$, причем $\tilde{N}_i$ отличается от $\tilde{N}_{i+1}$ в точности в одной координате $\left(i = 1,\ ...,\ m-1\right)$, то последовательность

$$\tilde{N}_1 0,\ \tilde{N}_2 0,\ ...,\ \tilde{N}_m 0,\ \ \tilde{N}_m 1,\ \tilde{N}_{m-1} 1,\ ...,\ \tilde{N}_1 1$$

содержит все бинарные последовательности длины $k+1$, причем каждые две соседние последовательности будут отличаться в точности в одной координате. Этим непосредственно определяется некоторый алгоритм рекуррентного построения последовательности всех подмножеств. Получаемая этим способом последовательность $\tilde{N}_1,\ ...,\ \tilde{N}_{2n}$ называется *бинарным кодом Грэя порядка* $n$.

Необходимо описать нерекуррентную версию этого алгоритма и программы.

**Алгоритм 3.1. Генерирование всех подмножеств *n*-элементного множества**.

*Данные*: $n$.

*Результат*: Последовательность всех подмножеств $n$-элементного множества, в которой каждое последующее подмножество получается из предыдущего добавлением или удалением единственного элемента. Каждое подмножество представляется бинарной последовательностью $B[1], ..., B[n]$.

```
1   begin
2     for i: = l to n do B[i]:= 0; (*пустое множество*)
3     i:= 0;  (*i = числу  сгенерированных  до  этого  момента подмножеств *)
4     repeat
5       write (B[i], ..., B[n]);
6       i:= i + 1; p:= 1; j:=i;
7       while j mod 2 = 0 do
8         begin (*j2^(p-1) = i*)
9         j :=j/2: p:=p + l
10      end; (*p = Q(i) + 1*)
11    if p ≤ n then B[p];=l – B[p] (* смена позиции p *)
12  until p > n
13  end
```

Докажем теперь, что описанный алгоритм действительно генерирует все бинарные по-

следовательности длины $n$. Пусть $Q(i)$ обозначает наибольшую степень двойки, которая делит $i$. Если мы представим $i$ в двоичном виде как $i = [b_n b_{n-1} \ldots b_1]$, то, очевидно, имеем $Q(i) + 1 = \min\{j: b_j = 1\}$. Покажем вначале, что после выхода из внутреннего цикла 7 будем иметь $p = Q(i) + 1$. С этой целью заметим, что до входа в цикл $j2^{n-1} = i$ (где $p = 1$, $j = i$) и каждая итерация цикла не нарушает этого равенства (так как $(j/2)2^{(p+1)-1} = j2^{p-1}$). После выхода из цикла $j$ нечетно, следовательно, $Q(i) = p - 1$, т. е. действительно $p = Q(i) + 1$. Предположим теперь, что $k < n$ и что в первых $2^k$ итерациях цикла 3 были получены все $2^k$ бинарные последовательности $b_1 b_2 \ldots b_n$, такие что $b_{k+1} = \ldots = b_n = 0$ (это очевидно справедливо для $k = 0$). В последней из этих $2^k$ итераций переменная i принимает значение $2^k$ (строка 7). Переменная p получает значение $Q(2^k) + 1 = k + 1$, и, следовательно, наступает изменение значения переменной $B[k + 1]$ с 0 на 1. Рассмотрим теперь последовательность

$$Q\left(2^k + 1\right) + 1,\ Q\left(2^k + 2\right) + 1,\ \ldots,\ Q\left(2^{k+1}\right) + 1 \qquad (3.1)$$

значений переменной $p$, сгенерированных в последующих $2^k$ итерациях цикла 3. Отметим, что $Q(2^k - m) = Q(2^k + m)$ для $0 \le m \le 2^k - 1$ (этот факт становится особенно очевидным, если рассматривать сложение чисел $2^k + m$ и $2^k - m$, записанных в двоичной форме). Последовательность (3.1) является зеркальным отображением последовательности значений переменной $p$, полученных в первых $2^k$ итерациях цикла 3. Отсюда следует, что и последовательности $B[1]$, $B[2]$, ..., $B[k]$, полученные в первых $2^k$ итерациях, появляются в обратном порядке в последующих $2^k$ итерациях. Это дает все бинарные последовательности $b_1 b_2 \ldots b_n$, такие, у которых $b_{k+1} = \ldots = b_n = 0$, полученные в первых $2^{k+1}$ итерациях. Отсюда становится ясно, что алгоритм генерирует все бинарные последовательности длины $n$ в таком же порядке, что и предыдущий рекурсивный алгоритм.

Последовательность подмножеств, полученных при помощи алгоритма для $n = 4$, представлена на рис. 3.1.

Можно показать, что среднее число шагов, необходимых для генерирования каждого следующего подмножества (не считая процедуры написания этого подмножества), ограничено константой, не зависящей от $n$. Последовательность подмножеств, полученных с помощью **алгоритма 3.1**, можно так же, как мы это делали для последовательности перестановок, полученных с помощью алгоритма генерирование всех перестановок с минимальным числом транспозиций соседних элементов (см. лабораторную работу № 2), проиллюстрировать на графе, вершины которого соответствуют бинарным последовательностям длины $n$ и две вершины которого соединены ребром, если соответствующие последовательности отличаются в точности в одной позиции. Такой граф будем называть (*двоичным*) *n-мерным кубом*. Очевидно, что последовательность, полученная с помощью

нашего алгоритма, соответствует гамильтонову пути в этом графе. На рис. 3.2 это продемонстрировано для $n$ = 1, 2, 3, 4.

0000
1000
1100
0100
0110
1110
1010
0010
0011
1011
1111
0111
0101
1101
1001
0001

Рис. 3.1. Последовательность подмножеств, порожденных алгоритмом 3.1 для $n$ = 4

В некоторых приложениях более естественным, нежели понятие множества, является *множество с повторениями*. Каждый элемент множества с повторениями может появляться в этом множестве несколько раз, число этих вхождений является существенным и носит название *кратности* элемента в множестве. Множество с повторениями, содержащее, например, элемент a кратности 2, элемент $b$ кратности 3 и элемент $c$ кратности 1, будем обозначать ($a$, $a$, $b$, $b$, $b$, $c$) или (2 ∗ $a$, 3∗$b$, 1∗$c$). Порядок элементов не существен:

$$(a, a, b, b, b, c) = (b, a, b, a, c, b) = ...$$

существенна только кратность каждого элемента – имеем

$$(a, a, b, b, b, c\} \neq (a, b, c)$$

в отличие от равенства множеств

$$\{a, a, b, b, b, c\} = \{a, b, c\}.$$

Если в некотором множестве с повторениями кратность каждого элемента равна единице, то, очевидно, мы можем отождествлять его с обычным множеством. Пусть $A$, $B$ – два множества с повторениями. Будем говорить, что $A$ есть подмножество $B$ (обозначение $A \subseteq B$), если кратность каждого элемента в $A$ не больше кратности этого элемента в $B$. Пусть $X$ – множество с повторениями, содержащее $r$ разных элементов $x_1$, ..., $x_r$ с кратностями $k_1$, ..., $k_r$ соответственно. Число $|X| = k_1 + ... + k_r$ будем называть *мощностью* $X$. Каждому подмножеству $A \subseteq X$ однозначно соответствует последовательность

$$\left\langle m_1,\ ...,\ m_r \right\rangle,\ 0 \le m_1 \le k_1,\ ...,\ 0 \le m_r \le k_r, \qquad (3.2)$$

где $m_i$ обозначает кратность элемента $x_i$ в $A$. Отсюда сразу же следует, что число всех

подмножеств $A \subseteq X$ равно

$$(k_1 + 1)(k_2 + 1) \ \ldots \ (k_r + 1).$$

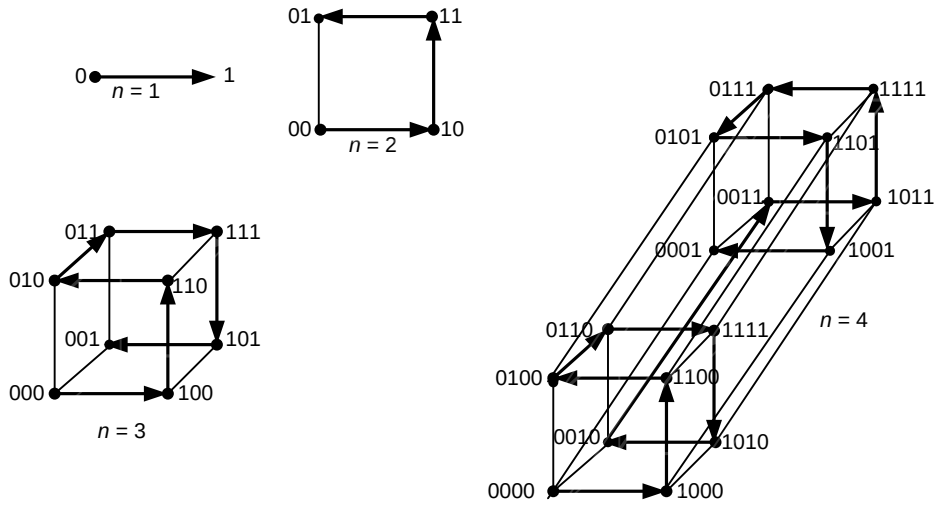


Рис. 3.2. Гамильтоновы пути в *n*-мерных кубах

Эти подмножества, а точнее соответствующие им последовательности (3.2), можно генерировать способом, подобным тому, который был использован в **алгоритме 3.1**. Для этого достаточно заметить, что если последовательность $C_1,\ C_2,\ \ldots,\ C_m$ содержит все $p = (k_1 + 1)\ \ldots\ (k_s + 1)$ последовательностей $m_1,\ \ldots,\ m_s,\ 0 \le m_1 \le k_1,\ \ldots,\ 0 \le m_s \le k_s$, то последовательность

$$C_1 0,\ C_2 0,\ \ldots,\ C_p 0,\ C_p 1,\ C_{p-1} 1,\ \ldots,\ C_1 1,\ C_1 2,\ C_2 2,\ \ldots,\ C_p 2,\ C_p 3,\ \ldots$$

длины $p(k_{s+1} + 1)$ содержит все последовательности $\langle m_1,\ \ldots,\ m_{s+1} \rangle$, $0 \le m_1 \le k_1$, ..., $0 \le m_{s+1} \le k_{s+1}$. Очевидно, что последовательность подмножеств, полученная с помощью такого построения, будет обладать таким свойством, что каждое последующее подмножество образуется из предыдущего добавлением или удалением одного элемента. Предоставляя читателю исключить рекурсию из этого алгоритма, проиллюстрируем его с помощью гамильтонова пути в некотором графе (рис. 3.3). Этот рисунок интерпретируется так же, как и рис. 3.2.

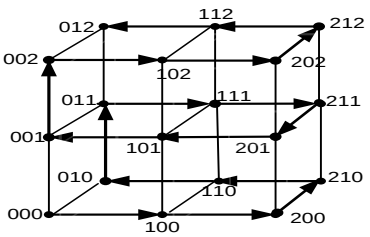


Рис. 3.3. Гамильтонов путь в графе, соответствующем подмножествам множества с повторениями $(x_1, x_1, x_2, x_2, x_3)$

### Вопросы для самопроверки

1. Что из себя представляет бинарный код Грэя порядка $n$ ?
2. Какой граф называется (двоичным) $n$-мерным кубом ?
3. Что понимается под множеством с повторениями?
4. Чему равно мощность множества с повторениями ?

### Задание на лабораторную работу

1. По заданному **алгоритму 3.1** написать и отладить программу получения подмножества наименьшим образом отличалось бы от предыдущего.
2. Продемонстрировать работающую программу преподавателю и получить отметку о ее выполнении.
3. Провести анализ алгоритма.
4. Оформить отчет о проделанной работе. Он должен включать в себя:

– цель работы;

– ответы на вопросы для самопроверки;

– схему обрабатывающего алгоритма и описание его работы;

– распечатки экранных форм, полученных в результате работы программы.

# Лабораторная работа № 4.
# *k*-Элементные подмножества, биномиальные коэффициенты. Генерирование *k*-элементных подмножеств

Число всех *k*-элементных подмножеств *n*-элементного множества будем обозначать $\binom{n}{k}$. Символ $\binom{n}{k}$ называется *биномиальным коэффициентом*, исходя из следующей формулы для *n*-й степени бинома $x + y$:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}. \qquad (4.1)$$

Чтобы убедиться в истинности этой формулы, достаточно заметить, что коэффициент при $x^k y^{n-k}$ равен числу способов, которыми из *n* сомножителей $(x+y)$ ... $(x+y)$ можно выбрать $k$ сомножителей.

Очевидно, $\binom{n}{k} = 0$ для $k > n$. Напомним, что *k*-элементные подмножества *n*-элементного множества назывались в старой литературе по комбинаторике *k*-членными комбинациями для *n*-элементного множества без повторений,

С биномиальными коэффициентами связано много интересных тождеств. Вот некоторые из них.

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n.$$

Эта формула следует из того, что сумма слева выражает число всех подмножеств *n*-элементного множества.

$$\sum_{i=0}^{n} \binom{n}{i} i = n 2^{n-1}. \qquad (4.2)$$

Для доказательства этой формулы составим список, содержащий по порядку все подмножества *n*-элементного множества $X$. Этот список содержит $\binom{n}{i}$ *i*-элементных подмножеств для $i = 1, ..., n$; таким образом, его длина (так называемое число появлений элементов множества $X$) выражается суммой в левой части формулы (4.2). С другой стороны, каждый элемент $x \in X$ появляется в списке $2^{n-1}$ раз, так как таково число подмножеств, содержащих $x$ (их столько, сколько имеется всех подмножеств множества $X \setminus \{x\}$). Поэтому длина нашего списка равна $n 2^{n-1}$. Доказательство тем самым законче-

но.

$$\binom{n}{k} = \binom{n}{n-k}.$$

Это прямое следствие того факта, что каждому *k*-элементному подмножеству $Y \subseteq X$ однозначно соответствует $(n-k)$-элементное подмножество $X \setminus Y$ множества $X$.

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}. \tag{4.3}$$

Эта формула получается из следующих рассуждений. Зафиксируем некоторый элемент $x$ *n*-элементного множества $X$. Множество всех *k*-элементных подмножеств множества $X$ распадается на два непересекающихся класса: класс подмножеств, которые не содержат элемент $x$, и класс подмножеств, которые его содержат. Мощность первого класса составляет $\binom{n-k}{k}$, а второго $\binom{n-k}{k-1}$, т.е. столько, сколько имеется $(k-1)$-элементных подмножеств у множества $X \setminus \{x\}$.

Тождество (4.3) тесно связано со следующим треугольником Паскаля:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
```

Если перенумеровать по порядку строки этого треугольника числами 0, 1, 2, ..., то *i*-я строка окажется состоящей из чисел

$$\binom{i}{0}\binom{i}{1} \cdots \binom{i}{i}.$$

В силу формулы (4.3) каждое из них, кроме крайних, равных единице, можно получить как сумму чисел, находящихся над ним в предыдущем ряду. Это дает простой метод построения треугольника Паскаля, а тем самым нахождения коэффициентов разложения $(x+y)^i$ – они задаются *i*-й строкой треугольника (ср. с (4.1)).

Докажем еще одно тождество, связанное с биномиальными коэффициентами, а именно тождество Коши:

$$\binom{m+n}{k} = \sum_{s=0}^{k} \binom{m}{s}\binom{n}{k-s}. \tag{4.4}$$

Для этого представим себе, что из группы, состоящей из *m* мужчин и *n* женщин мы хо-

тим выбрать $k$ человек. Мы можем сделать это $\binom{m+n}{k}$ способами (левая часть тождества). Все $k$-элементные подмножества нашего множества можно классифицировать по числу мужчин в подмножестве. $k$-элементное подмножество, содержащее $s$ мужчин, можно получить, выбирая сначала $s$ мужчин одним из $\binom{m}{s}$ возможных способов, а затем $(k-s)$ женщин одним из $\binom{n}{k-s}$ способов, Таким образом, число всех возможных подмножеств из $k$ человек, включающих $s$ мужчин, равно $\binom{m}{s}\binom{n}{k-s}$. Отсюда непосредственно следует формула (4.4).

Отметим, что для $m = k = n$ данная формула приобретает вид

$$\binom{2n}{n} = \sum_{s=0}^{n} \binom{n}{s}^2.$$

Мы хотим доказать еще одно тождество:

$$\binom{n}{k}\binom{k}{m} = \binom{n}{m}\binom{n-m}{n-k}. \qquad (4.5)$$

Для доказательства подсчитаем число таких пар $\langle K,\ M \rangle$ подмножеств $n$-элементного множества $X$, что $K \subseteq M$, $|K| = k$, $|M| = m$. С одной стороны, подмножество $K$ можно выбрать $\binom{n}{k}$ способами, а подмножество $M \subseteq K$ можно выбрать $\binom{k}{m}$ способами, что дает $\binom{n}{k}\binom{k}{m}$ возможных пар $\langle K,\ M \rangle$ (левая часть (4.5)). С другой стороны, для каждого из $\binom{n}{m}$ подмножеств $M \subseteq X$ можно выбрать $\binom{n-m}{n-k}$ способами подмножество $K \subseteq M$, так как такое подмножество $K$ однозначно определяется через $(k-m)$-элементное подмножество $X \setminus K$ $(n-m)$-элементного множества $X \setminus M$. Искомое число пар $\langle K,\ M \rangle$ равно, таким образом, $\binom{n}{m}\binom{n-m}{n-k}$ (правая часть (4.5)).

Приведём теперь формулу, позволяющую вычислять $\binom{n}{k}$ непосредственно.

**Утверждение 4.1.**

$$\binom{n}{k} = \frac{[n]_k}{k!} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\ \ldots\ (n-k+1)}{1\cdot 2\cdot\ \ldots\ \cdot k}. \tag{4.6}$$

*Доказательство.* Каждая из $[n]_k$ инъективных последовательностей длины $k$ определяет $k$-элементное подмножество, состоящее из элементов, появляющихся в этой последовательности, причем одно и то же подмножество $S$ получаем в точности из $k!$ последовательностей, соответствующих всем перестановкам множества $S$. Отсюда $\binom{n}{k} = \frac{[n]_k}{k!}$.

Отметим, что эта теорема легко позволяет доказать все тождества, рассмотренные в этом разделе. Например, для (4.3) имеем

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{k!(n-k)!}\left(\frac{n-k}{k} + \frac{k}{n}\right) = \frac{(n-1)!}{k!(n-1-k)!} + \frac{(n-1)!}{(k-1)!(n-k)!} = \binom{n-1}{k} + \binom{n-1}{k-1}.$$

Однако такое «алгебраическое» доказательство, основанное на механическом преобразовании выражений, гораздо менее понятно, чем «комбинаторное» доказательство, данное выше.

С другой стороны, формула (4.6) позволяет легко доказать следующее свойство биномиальных коэффициентов;

$$\binom{n}{0} < \binom{n}{1} < \ \cdots\ < \binom{n}{\lfloor n/2 \rfloor} = \binom{n}{\lceil n/2 \rceil} > \binom{n}{\lceil n/2 \rceil + 1} > \ \cdots\ > \binom{n}{n} \tag{4.7}$$

$(n > 1)$. Достаточно заметить, что

$$\binom{n}{i+1} = \frac{n(n-1)\ \ldots\ (n-i)}{1\cdot 2\cdot\ \ldots\ \cdot i(i+1)} = \frac{n-1}{i+1}\cdot\frac{n(n-1)\ \ldots\ (n-i+1)}{1\cdot 2\cdot\ \ldots\ \cdot i} = \frac{n-i}{i+1}\binom{n}{i}.$$

Имеем $(n-i)/(i+1) > 1$ для $i < (n-1)/2$, т. е. для $i < \lfloor n/2 \rfloor$, аналогично $(n-i)/(i+1) < 1$ для $i > (n-1)/2$, т. е. для $i \ge \lceil n/2 \rceil$, и $(n - \lfloor n/2 \rfloor)/(\lfloor n/2 \rfloor + 1) = \lceil n/2 \rceil / \lfloor n/2 \rfloor = 1$, если $n$ нечетно (для четного $n$ имеем $\lfloor n/2 \rfloor = \lceil n/2 \rceil$ и среднее равенство в (4.7), очевидно, выполняется).

Займемся теперь отысканием числа $k$-элементных подмножеств множества с повторениями $(k * x_1,\ \ldots,\ k * x_n)$. Другими словами, зададимся вопросом, сколькими способами можно породить $k$-элементное множество с повторениями, имея в распоряжении $n$ разных элементов, скажем $1,\ \ldots,\ n$, из которых каждый может использоваться произвольное число раз. Следует отметить, что это число равно числу неубывающих последовательностей

длины $k$ с элементами из множества $\{1, \ldots, n\}$. Это вытекает из того факта, что элементы каждого подмножества можно однозначно расставить в неубывающую последовательность. Очевидно, что подмножествам без повторений соответствуют возрастающие последовательности.

**Утверждение 4.2.** Число $k$-элементных подмножеств множества $(k * x_1, \ldots, k * x_n)$ равно

$$\binom{n+k-1}{k}.$$

*Доказательство*. Мы должны определить число убывающих последовательностей $\langle a_1, a_2, \ldots, a_k \rangle$ с элементами из множества $\{1, 2, \ldots, n\}$. Каждую такую последовательность можно разделить на $n$ частей, где $i$-я часть содержит некоторое, быть может нулевое, число следующих друг за другом элементов $i$. Если мы отделим эти части одну от другой $n-1$ нулями, использованными в качестве разделителей, то получим последовательность длины $k+n-1$. Например, для $n=5,\ k=4$ последовательности $\langle 2, 2, 4, 5 \rangle$ соответствует последовательность $\langle 0, 2, 2, 0, 0, 4, 0, 5 \rangle$. Последняя последовательность однозначно определяется размещением появляющихся в ней нулей. Действительно, мы можем восстановить её, заполняя часть до первого нуля экземплярами элемента 1, часть от первого до второго нуля – экземплярами элемента 2 и т. д. А число размещений $n-1$ нулей на $k+n-1$ позициях равно

$$\binom{n+k-1}{n-1} = \binom{n+k-1}{(n+k-1)-n-1} = \binom{n+k-1}{k}.$$

Полученное таким образом число подмножеств с повторениями можно записать несколько иначе:

$$\binom{n+k-1}{k} = \frac{n(n+1)\ \ldots\ (n+k-1)}{k!} = \frac{[n]^k}{k!}. \qquad (4.8)$$

Дадим комбинаторную интерпретацию этого тождества. Напомним с этой целью, что $[n]^k$ есть число упорядоченных размещений $k$ элементов $x_1, \ldots, x_n$ в $n$ ячейках. Отметим, что каждому такому размещению, содержащему $r_i$ элементов в $i$-й ячейке, соответствует $k$-элементное подмножество $(r_1 * y_1, \ldots, r_n * y_n)$ множества с повторениями $(k * y_1, \ldots, k * y_n)$. этом упорядочении существенно только число элементов в каждой из

ячеек. Таким образом, размещениям

| ячейка 1 | ячейка 2 | ячейка 3 |
|---|---|---|
| $x_3,\ x_2$ | | $x_5,\ x_1,\ x_4$ |
| $x_4,\ x_1$ | | $x_2,\ x_3,\ x_5$ |

соответствует одно и то же подмножество $2 * y_1,\ 3 * y_3$. Ясно, что каждые два размещения, соответствующие одному и тому же подмножеству, отличаются перестановкой объектов $x_1,\ ...,\ x_k$. В результате число всех *k*-элементных подмножеств множества с повторениями $\left(k * y_1,\ ...,\ k * y_n\right)$ равно $[n]^k \big/ k!$. Очевидно, что это рассуждение вместе с формулой (4.8) представляет собой другое доказательство **утверждения 4.2**.

Опишем сейчас два алгоритма генерирования всех *k*-элементных подмножеств *n*-элементного множества $X$. Без ограничения общности можно принять $X = \{1,\ ...,\ n\}$. Тогда каждому *k*-элементному подмножеству взаимно однозначно соответствует возрастающая последовательность длины *k* с элементами из $X$: например, подмножеству $\{3,\ 5,\ 1\}$ соответствует последовательность $\langle 1,\ 3,\ 5\rangle$.

Можно легко указать алгоритм, с помощью которого генерируются все такие последовательности в лексикографическом порядке. Достаточно с этой целью заметить, что при таком порядке последовательностью, непосредственно следующей за последовательностью $\langle a_1,\ a_2,\ ...,\ a_k\rangle$, является

$$\langle b_1,\ b_2,\ ...,\ b_k\rangle = \langle a_1,\ ...,\ a_{p-1},\ a_p + 1,\ a_p + 2,\ ...,\ a_p + k - p + 1\rangle,$$

где

$$p = \max\{i : a_i < n - k + 1\}.$$

Более того, последовательностью, непосредственно следующей за $\langle b_1,\ b_2,\ ...,\ b_k\rangle$, является

$$\langle c_1,\ c_2,\ ...,\ c_k\rangle = \langle b_1,\ ...,\ b_{p'-1},\ b_{p'} + 1,\ b_{p'} + 2,\ ...,\ b_{p'} + k - p' + 1\rangle,$$

где

$$b_{p'} = \begin{cases} p - 1, & \text{если } b_k = n, \\ k, & \text{если } b_k < n \end{cases}$$

(будем предполагать, что последовательности $\langle a_1,\ a_2,\ ...,\ a_k\rangle$ и $\langle b_1,\ b_2,\ ...,\ b_k\rangle$ отличаются от $\langle n - k + 1,\ ...,\ n\rangle$ – последней последовательности в нашем порядке). Это приводит к

следующему простому алгоритму.

**Алгоритм 4.1. Генерирование всех *k*-элементных подмножеств множества {1, ..., *n*} в лексикографическом порядке.**

*Данные*: *n*, *k*.

Результат: последовательность всех *k*-элементных подмножеств множества {1, …, *n*) в лексикографическом порядке.

```
1  begin
2    for i:= 1 to k do A [i] := i; (* первое подмножество *)
3    p: = k;
4    while p ≤ 1 do
5      begin write (A [1], ..., A[k]);
6        if A[k]= n then p:= p – 1 else p := k;
7        if p ≥ 1 then
8          for i: = k downto p do A[i] : = A[p] + i – p + 1
9      end
10 end
```

Последовательность всех 4-элементных подмножеств множества {1, ..., 6}, полученная с помощью этого алгоритма, представлена на рис. 4.1.

1 2 3 4
1 2 3 5
1 2 3 6
1 2 4 5
1 2 4 6
1 2 5 6
1 3 4 5
1 3 4 6
1 3 5 6
1 4 5 6
2 3 4 5
2 3 4 6
2 3 5 6
2 4 5 6
3 4 5 6

Рис. 4.1. Последовательность 4-элементных подмножеств множества {1, ..., 6}, построенная при помощи алгоритма 4.1

Другой алгоритм, который мы опишем ниже, генерирует все *k*-элементные подмножества таким образом, что каждое последующее подмножество образуется из предыдущего удалением одного элемента и добавлением другого. Этот алгоритм представим в рекурсивной форме. Обозначим с этой целью через $G(n,\ k)$ список, содержащий все *k*-элементные подмножества множества $\{1,\ 2,\ ...,\ n\}$, в котором первым подмножеством является $\{1,\ 2,\ ...,\ k\}$, последним – $\{1,\ 2,\ ...,\ k-1,\ n\}$ и каждое следующее подмножество образуется из предыдущего удалением некоторого элемента и добавлением другого. Отметим, что если $G(n-1,\ k)$ и $G(n-1,\ k-1)$ уже построены, то $G(n,\ k)$ можно опреде-

лить следующим образом:

$$G(n,\ k) = G(n-1,\ k),\ G^*(n-1,\ k-1) \cup \{n\}, \tag{1.25}$$

где $G^*(n-1,\ k-1) \cup \{n\}$ обозначает список, образованный из $G(n-1,\ k-1)$ изменением порядка элементов списка на обратный и последующим добавлением элемента $n$ к каждому множеству. Действительно, $G(n-1,\ k)$ содержит все *k*-элементные подмножества множества {1, ..., *n*}, не содержащие $n$, $G^*(n-1,\ k-1) \cup \{n\}$ – все *k*-элементные подмножества, содержащие $n$, причем последним подмножеством в списке $G(n-1,\ k)$ является $\{1,\ 2,\ ...,\ k-1,\ n-1\}$, а первым подмножеством в списке $G^*(n-1,\ k-1) \cup \{n\}$ является $\{1,\ 2,\ ...,\ k-1,\ n\}$. На рис. 4.2 показан процесс построения списка $G(4,\ 2)$.

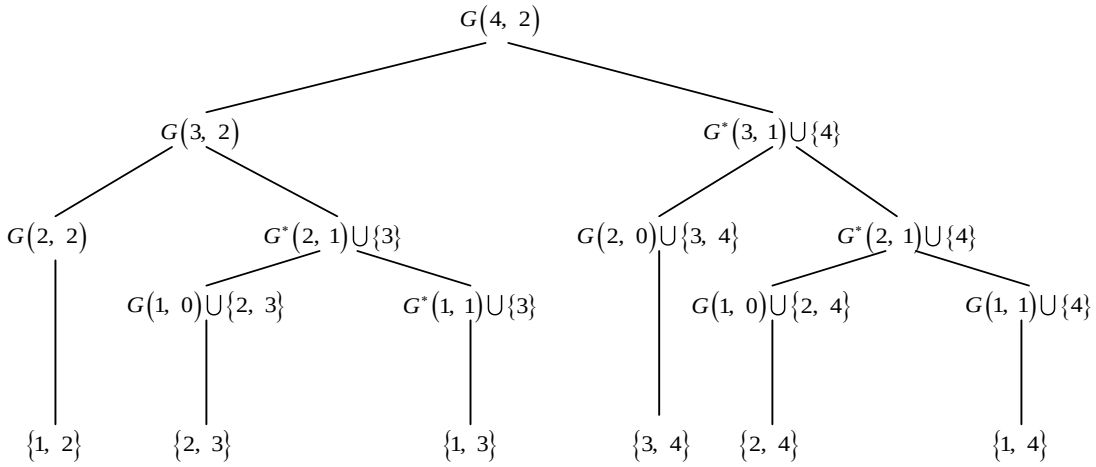


Рис. 4.2. Построение списка *G*(4, 2).

Списку $G(n,\ k)$ мы можем так же, как и в случае генерирования всех подмножеств, поставить в соответствие некоторый гамильтонов путь в графе, вершины которого соответствуют двухэлементным подмножествам множества $\{1,\ 2,\ 3,\ 4\}$, причем вершины, соответствующие подмножествам $A$ и $B$, соединены ребром тогда и только тогда, когда $|A \cap B| = k - 1 = 1$. Это проиллюстрировано на рис. 4.3.

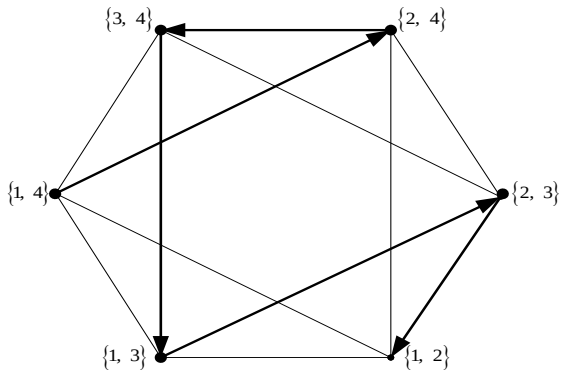


Рис. 4.3. Гамильтонов путь в графе, соответствующем всем двухэлементным подмножествам множества {1, 2, 3, 4}

## Вопросы для самопроверки

1. Что обозначает и чему равен символ $\binom{n}{k}$?

2. Докажите тождество $\sum_{k=0}^{m}\binom{m}{k}=2^m$ ?

3. Докажите тождество $\sum_{k=0}^{m}(-1)^{m-k}\binom{m}{k}=0 \quad (m\geq 1)$ ?

4. Сколькими способами можно породить $k$-элементное множество с повторениями, имея в распоряжении $n$ разных элементов, скажем 1, ..., $n$, из которых каждый может использоваться произвольное число раз ?

## Задание на лабораторную работу

1. По заданному алгоритму 4.1 написать и отладить программу генерирования всех $k$-элементных подмножеств множества $\{1, ..., n\}$ в лексикографическом порядке.

2. Продемонстрировать работающую программу преподавателю и получить отметку о ее выполнении.

3. Провести анализ полученного алгоритма.

4. Оформить отчет о проделанной работе. Он должен включать в себя:

– цель работы;

– ответы на вопросы для самопроверки;

– схему обрабатывающего алгоритма и описание его работы;

– распечатки экранных форм, полученных в результате работы программы.

## СПИСОК ЛИТЕРАТУРЫ

1. *Калужнин Л. А., Сущанский В. И.* Преобразования и перестановки: Пер. с укр. – 2-е изд., перераб. и доп. – М.: Наука, 1985. – 160 с.

2. *Кофман А.* Введение в прикладную комбинаторику. М.: Наука, 1975.

3. *Сачков В.Н.* Введение в комбинаторные методы дискретной математики. – 2-е изд., испр. и доп. – М.: МЦНМО, 2004. – 424 с.

4. *Уилсон Р.* Введение в теорию графов. М.: Мир, 1977. – 208 с.