Московский Государственный Технический Университет имени Н. Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Иванова Г.С., Пугачев Е.К.

УТВЕРЖДАЮ
Зав. кафедрой ИУ6

д.т.н., проф. _____ Сюзев В.В.
"____"__2013 г.

Исследование структур и методов обработки данных

Методические указания по выполнению лабораторной работы

по дисциплине «Технология разработки программных систем»

Москва 2013

Введение

При разработке алгоритмов программ часто возникает задача выбора структур данных и методов их обработки. Исходными составляющими для решения этой задачи являются описание набора и типов хранимых данных, а также перечень операций, выполняемых над ними.

Можно выделить следующие основные вопросы, на которые необходимо ответить при решении поставленной задачи:

- Как логически организовать структуру данных? Как ее реализовать?
- Как осуществлять поиск информации? Как упорядочить данные?
- Как выполнить функции корректировки данных?

Цель работы – исследование структур данных, методов их обработки и оценки.

Продолжительность работы – 9 часов.

1. Краткие теоретические сведения

1.1 Классификация абстрактных структур данных

Под с*труктурой данных* понимают совокупность правил и ограничений, которые отражают связи, существующие между отдельными частями (элементами) данных.

В процессе проектирования программного продукта разработчик обычно создает модель представления данных, не зависящую от реализации, используя при этом абстрактные структуры данных. Классификация таких структур приведена на рисунке 1. Модель представления данных может быть как простой, так и комбинированной (включающей различные абстрактные структуры).

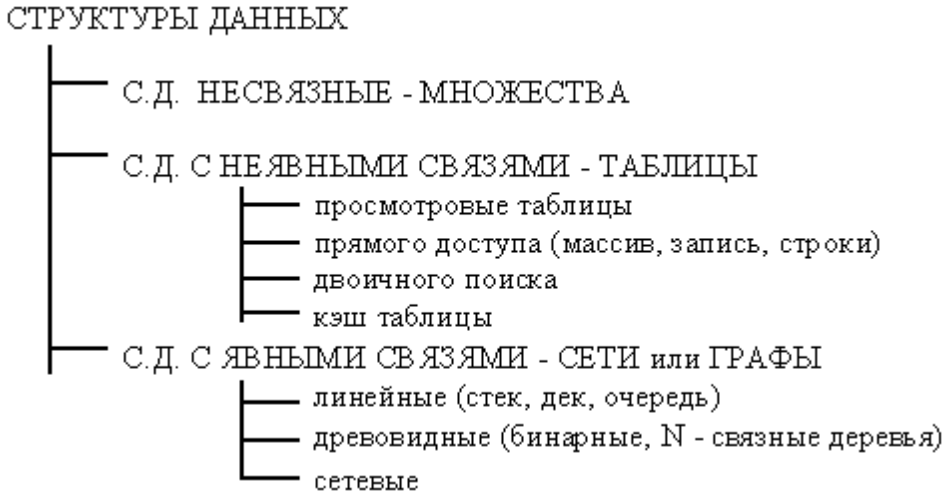

Рисунок 1 – Классификация абстрактных структур данных

Множество – неупорядоченная совокупность взаимно независимых (несвязанных) элементов. Используется если необходимо определять принадлежность какого-либо элемента множеству аналогичных элементов.

Структуры данных с неявными (статическими) связями элементов – таблицы – множество элементов, с каждым из которых связан ключ. Используются обычно для хранения данных. Основная операция – поиск информации по ключу. Различают четыре типа таблиц (просмотровые, прямого доступа, двоичного поиска и таблицы с перемешиванием) с различной *эффективностью* поиска данных. Эффективность поиска в таблицах различных типов оценивают, используя две характеристики: S – длина поиска – количество записей, которые необходимо просмотреть, чтобы найти запись с заданным ключом; L – средняя длина поиска при постоянной частоте обращения.

1. ***Просмотровые таблицы*** – таблицы, в которых записи просматривают последовательно $S_i = i$; $L = (N+1)/2$, где N – количество записей.

Недостаток: очень велико среднее время поиска. Достоинство: простота создания и добавления данных.

2. ***Таблицы прямого доступа*** – таблицы, в которых ключ – однозначно определяет адрес. Чаще всего ключом является натуральное число или ключ приводится к натуральному числу – номеру элемента. В таких таблицах ключ можно не хранить. Длина поиска и средняя длина поиска таблиц прямого доступа соответственно равны: S = 1, L = 1.

Недостаток: подобная организация таблицы редко применима, т. к. ключи часто бывают непоследовательными или изменяются в недопустимо больших диапазонах. Достоинство: быстрый поиск.

Частными случаями таблиц прямого доступа являются:

1) *Массив* – множество элементов, расположенных таким образом, что упорядоченное множество целых чисел (ключ) однозначно определяет позицию каждого элемента, а также обеспечивает возможность организации прямого доступа к каждому элементу. Если упорядоченное множество содержит N чисел, то массив N-мерный. Числа, входящие в упорядоченное множество, называются индексами. Каждый индекс имеет свой диапазон изменения.

2) *Строка* – одномерный массив символов.

3) *Запись* – конечное упорядоченное множество элементов, в общем случае различных типов. Ключом является порядковый номер поля или его имя.

Логический порядок элементов в просмотровых таблицах и таблицах прямого доступа при реализации обычно совпадает с физическим порядком расположения элементов. Элементами являются записи. Записи могут быть фиксированной, переменной и неопределенной длины. В зависимости от типа записи таблицы реализуются массивом записей или массивом векторов (статических или динамических), в котором основной массив содержит адреса векторов записей.

Для массива записей фиксированной длины адреса промежуточных записей задаются формулой:

$A_i = A_1 + (i-1)*l$, где A_i – адрес i-ой записи; A_1 – адрес первой записи; l – длина записи.

Достоинство: экономное использование памяти, т. к. записи располагаются одна за другой без промежутков. Недостаток: необходимо заранее знать общее число записей, их длину и диапазон изменения ключей.

3. ***Таблицы с перемешиванием*** *(кэш-таблицы)* – попытка построить таблицу с $L \cong 1$ при нерегулярных ключах. Для построения кэш-таблицы выбирается функция перемешивания (кэш-функция), *определенная на множестве значений ключа*, которая разбивает весь диапазон ключей на эквивалентные классы максимально равномерно. Далее для класса ключей адрес определяется как при прямом доступе, а внутри класса – последовательно (см. пример на рисунке 2).

Недостаток: часто сложно построить кэш-функцию, так как при слишком большом количестве классов часть из них может оказаться пустыми, что увеличивает объем используемой памяти. Оптимальным считается заполнение таблицы на 60-70%. Достоинство: среднее время поиска близко к 1.

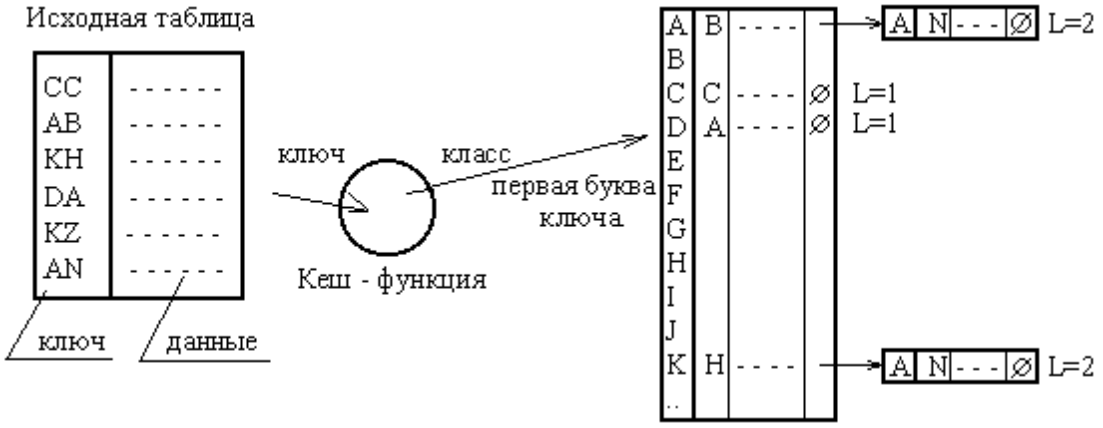

Рисунок 2 - Пример построения кэш-таблицы

4. ***Таблицы двоичного поиска*** – записи в таблице располагают по возрастанию или убыванию ключей, что делает применимым двоичный (дихотомический) поиск (см. далее), при котором $L = (\log_2 N)/2$.

Недостаток: добавляя записи необходимо сохранять сортировку. Достоинство: среднее время поиска существенно меньше, чем при последовательном просмотре.

Реализуются массивом или бинарным деревом.

Структуры с явными (динамическими) связями – структуры, каждый элемент которых содержит указатели на следующие элементы (связи представлены в явном виде). В основном используются при моделировании данных с изменяемыми связями между элементами, т.к. несут в явном виде информацию о связях.

1. ***Линейные структуры с явными связями –*** элемент расположен между двумя другими (если он не первый и не последний) – изменяемая последовательность элементов. Такие структуры могут быть реализованы в виде изменяемых векторов или с использованием списков (одно-, двухсвязных или кольцевых).

Частные случаи:

Очередь – последовательность элементов, добавление элементов к которой осуществляется в конец, а удаление – из начала.

Стек – последовательность элементов, добавление и удаление элементов в которую осуществляется с одного конца.

Дек – последовательность элементов, добавление и удаление элементов в которую возможны с любого конца.

2. ***Древовидные структуры с явными связями*** – записи располагаются по уровням следующим образом:

- на 1-м уровне расположена только одна запись (корень дерева);
- любая запись *i*-го уровня связана (адресуется) только с одной записью (*i*-1)-го уровня.

Записи *i*-го уровня, адресованные общей записью (*i*-1)-го уровня, образуют группу этой записи. Максимальное число записей в группе называют порядком (степенью) дерева. Число занятых уровней называют рангом.

Обычно реализуются на двусвязных или n-связных списках. Сбалансированные деревья могут реализовываться массивом. Возможна реализация с использованием матрицы связности, как сетевой структуры.

Двоичное дерево – каждая вершина адресует 0-2 вершин следующего уровня. N-мерное дерево – каждая вершина указывает на 0-n вершин более низкого уровня.

3. ***Сетевые структуры с явными связями*** – сети или графы. Граф – совокупность конечного множества вершин и конечного множества неупорядоченных пар вершин,

называемых ребрами. В ориентированном графе пары вершин упорядочены. В этом случае вершины называются узлами, а ребра дугами.

Реализуются различными способами: от матриц связности или инцидентности до n-связных списков (см. рисунок 8).

1.2 Методы обработки структурированных данных

Методы упорядочения (сортировки по возрастанию/убыванию) записей таблиц

*1.**Методы упорядочения обменом:***

а) Метод, базирующийся на попарном сравнении соседних элементов, заключается в следующем. Сравнивается первый элемент со вторым и, если он больше (меньше), то элементы меняются местами. Далее второй элемент сравнивается с третьим и т. д. После анализа последнего элемента процесс повторяется с начала. Количество сравнений определяется выражением: C = N (N-1), где N – количество элементов.

б) Метод, базирующийся на поиске минимального (максимального) элемента, заключается в следующем. Определяется минимальный (максимальный) элемент всего набора и осуществляется обмен с первым элементом. Затем определяется наименьший (наибольший) элемент из оставшегося набора и меняется местами со вторым и т. д. Количество сравнений по данному методу определяется по формуле: C = N(N-1)/2.

*2. **Метод вставки:***

Элемент массива a_i (начиная со второго) сравнивается последовательно с предшествующими a_j, где j = i-1, i-2, … до тех пока не будет найден элемент с меньшим значением, чем a_i. Пусть этот элемент с номером j, где j < i. Тогда все элементы с номерами j+1, …i-1 сдвигаются на одну позицию, а i-й элемент ставится на место (j+1)-го элемента. Если все впереди стоящие элементы больше *i*-го, то они сдвигаются на одну позицию, а *i*-й элемент ставится на первое место. Количество сравнений определяется по формуле: C = N(N-1)/4.

*3. **Метод Шелла**.*

Метод состоит в том, что упорядочиваемый массив делится на ряд групп, каждая из которых упорядочивается методом вставки. Количество операций сравнений оценивается следующим образом: $C \le 0{,}5\ N^{3/2}$.

*4. **Метод квадратичной выборки***

Упорядочиваемый массив, состоящий из N элементов, делится на $\sqrt{N}$ групп по $\sqrt{N}$ элементов в каждой. Для каждой группы необходимо выделить поле памяти, достаточное для размещения значения ключевого признака. Совокупность таких полей образует зону линейного накопления S_N. В результате просмотра элементов каждой группы определяется наименьший элемент, который заносится в соответствующее поле зоны линейного накопления S_N. Затем, просматриваются элементы в S_N, и наименьший из элементов заносится в зону формирования S_R. Далее осуществляется просмотр в группе, элемент которой записан в зону S_R. Наименьший элемент из оставшихся в группе заносится в зону накопления S_N на место перенесенного в зону формирования S_R. После этого снова просматриваются элементы в S_N, и наименьший из элементов заносится в зону формирования S_R. Процесс повторяется, пока в зоне S_N не останется элементов. На рисунке 3 приведен пример схемы сортировки по методу квадратичной выборки. Количество сравнений в процессе сортировки: $C = (N-1)\sqrt{N}/2$.

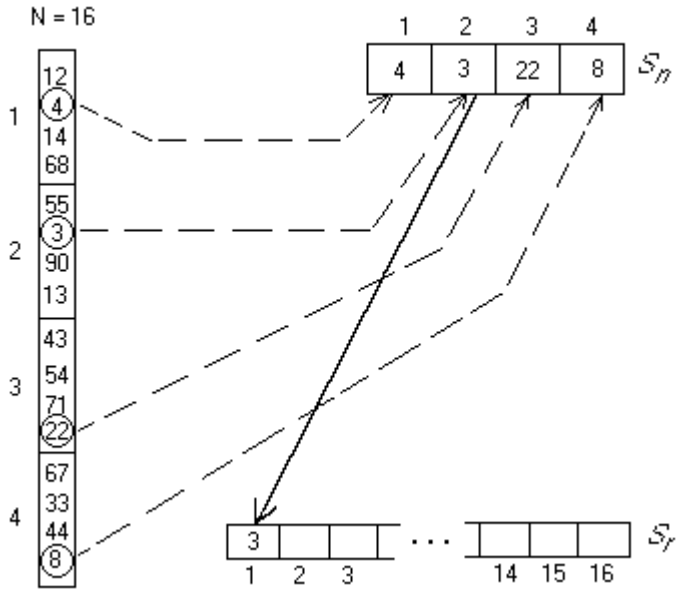

Рисунок 3 - Схема сортировки по методу квадратичной выборки

5. ***Метод слияния***

В основе данного метода лежит процедура слияния двух сортированных массивов A и B. Слияние заключается в поочередной пересылке элементов из массивов A и B в аналогично сортированный массив C. Порядок пересылки зависит от результата сравнения очередных элементов массивов A и B.

Упорядочение слиянием состоит в том, что обрабатываемый массив разделяется на равные группы элементов. Группы упорядочиваются, а затем попарно сливаются, образуя новые группы. Элементы внутри групп упорядочивают методом вставки.

Методы поиска в последовательных структурах данных

Поиском называется процедура выделения из множества записей подмножества, элементы которого удовлетворяют заранее поставленному условию. Поисковым признаком может служить номер записи в массиве или значение ключа. Условия поиска могут быть различными: по совпадению, по попаданию в интервал, по удовлетворению арифметическому условию, по удовлетворению семантическому условию, по нескольким условиям. Если задано только одно значение признака поиска, то такой поиск называется единичным; если же задано множество признаков поиска, то это групповой поиск.

Эффективность различных алгоритмов поиска оценивается количеством сравнений пар признаков, необходимым для выполнения условия поиска.

1. ***Метод последовательного поиска***

В основе метода последовательный просмотр записей массива с целью отыскания записей, для которых значение ключевого признака p совпадает со значением признака поиска q. Метод является универсальным в том смысле, что применим как к упорядоченным, так и к неупорядоченным массивам данных.

При реализации метода необходимо учитывать дополнительные условия:

* массив неупорядочен и неизвестно число элементов, удовлетворяющих поисковому признаку;
* массив неупорядочен, но известно число элементов, удовлетворяющих поисковому признаку;

* массив упорядочен (если в массиве имеется несколько записей с одинаковыми значениями ключевых признаков, то упорядоченность массива имеет существенное значение).

Среднее число сравнений для реализации данного метода вычисляется следующим образом

$$C = (N+1)/2, \text{ где } N \text{ – число записей в массиве.}$$

2. Метод двоичного поиска

Существует группа методов поиска, в основе которых лежит деление массива на части. Метод деления на две части называется двоичным (дихотомическим) поиском.

Пусть массив упорядочен и длина его известна. На первом шаге сравнивается значение ключевого признака средней записи p_i. Если $p_i = q$, то после проверки соседних элементов (они могут иметь тоже значение ключевого признака) поиск прекращается. Если $p_i > q$, то дальнейший поиск будет осуществляться в 1-й половине массива. Если $p_i < q$, то поиск будет продолжаться во 2-й половине массива. Затем в выделенном подмассиве определяется средняя запись и производится сравнение p_i с q и т. д.

Метод называют двоичным в связи с тем, что после каждого сравнения принимается одно из двух альтернативных решений.

Среднее число операций сравнения при поиске в массиве равно

$$C_{ср} = [(N+1)\log_2 (N+1)]/N - 1$$

3. Метод вычисления адреса (адресный поиск)

Применяется к упорядоченным массивам, если значения ключевых признаков не повторяются и их число не превышает числа записей в массиве. При этом записи в массиве должны быть фиксированной длины. Номера записей и числовые значения ключевых признаков связаны между собой адресной функцией: i = f(p), где i – номер записи (или адрес элемента в памяти); p – значение ключевого признака.

Вид функции может быть любым и выбирается при разработке системы поиска. Простейшая адресная функция имеет вид: $i = kp – a$, где a – константа и для простоты обработки берется равной $a = kP_{min} - 1$, где P_{min}- минимальное значение ключевого признака; k - целочисленная константа.

На рисунке 4 представлен пример использования данного метода для быстрого поиска нужного гнезда сотрудников определенного возраста.

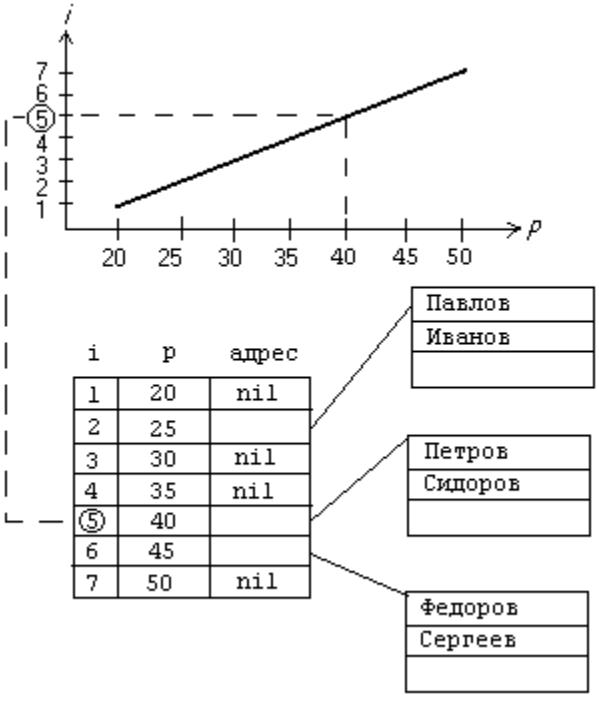

Рисунок 4 - Пример использования метода вычисления адреса

Корректировка структур данных, реализованных в виде массивов

Корректировка представляет собой процедуру внесения изменений в структуру данных. Различают следующие виды корректировки: вставка (добавление), аннулирование (удаление), замена элементов.

Опишем некоторые особенности корректировки последовательных структур данных, реализованных в виде сортированных массивов. При необходимости сохранения сортировки для вставки требуется наличие свободных участков памяти между записями массива для размещения вставляемых записей. Заранее спланировать необходимый объем памяти и зарезервировать место (адреса) невозможно, поэтому реализация вставки невозможна без реорганизации массива. Для вставки новой записи требуется сдвиг всех последующих записей (в среднем – половины элементов массива).

При удалении записей также необходим сдвиг всех последующих записей (также в среднем половины элементов массива). Аннулирование записей возможно без реорганизации массива. Появляющиеся при этом свободные участки памяти могут быть отмечены как свободные в специальном массиве, тогда каждый раз при выполнении операции над данными придется проверять, не удалены ли данные.

Операция замены возможна без реорганизации массива.

Корректировка может выполняться двумя способами. В первом случае изменения вносятся в основной массив сразу. Во втором случае, применяемом при очень больших размерах массивов, записи с измененными данными накапливаются в специальном массиве изменений. При достижении определенного объема массив изменений объединяется с основным массивом.

Методы обработки списковых структур данных

Способы ускорения поиска в последовательных структурах данных, реализованных в виде списков

Основным преимуществом списковой структуры является гибкость и высокая скорость корректировки (добавления и удаления элементов). Однако поиск в них выполняется только последовательно. Имеются способы, позволяющие ускорить поиск в списковых структурах. К ним относятся: использование адресной функции (см. ранее), К- и А-индексов; гнездовой способ организации.

1. ***Метод К- и А- индексов***

Значения ключей записей, номера которых образуют арифметическую прогрессию, называют К-индексами. Адреса записей, значения ключей которых образуют арифметическую прогрессию, называют А-индексами.

В списковой структуре, снабженной теми или иными индексами, поиск ведется в два этапа: в массиве индексов и в основной структуре.

При поиске элемента с ключом q в списковой структуре, имеющей массив К-индексов, находится индекс K_i, такой, что $K_i \le q < K_{i+1}$ (i – номер индекса в массиве индексов). Далее организуется вход в список по адресу, записанному в K_i, и поиск в этой структуре методом перебора.

В случае А-индексов для искомого q вычисляется $i = q/z$, где z – интервал значения ключа для А- индекса. Далее элементы списка просматриваются последовательно, начиная с адреса, записанного в i-м А-индексе.

2. ***Гнездовой способа организации***

Способ заключается в том, что множество элементов разбивается на группы по определенному принципу. Затем эти группы объединяют в отдельные гнезда с одинаковой структурой. Далее строят дополнительную структуру, в которой все элементы

связываются. Каждый элемент содержит ключевой признак и адрес только одного гнезда.

На рисунке 5 представлен пример организации гнездового способа с помощью списка гнезд. В этом примере записи базы данных делятся на группы по принципу принадлежности первой буквы фамилии. Для доступа к гнезду организован список гнезд, в котором каждый элемент имеет три поля: ключевой признак гнезда (первая буква фамилии), адрес гнезда и адрес следующего элемента списка гнезд. Для хранения самих гнезд используется структура с неявными связями – массив.

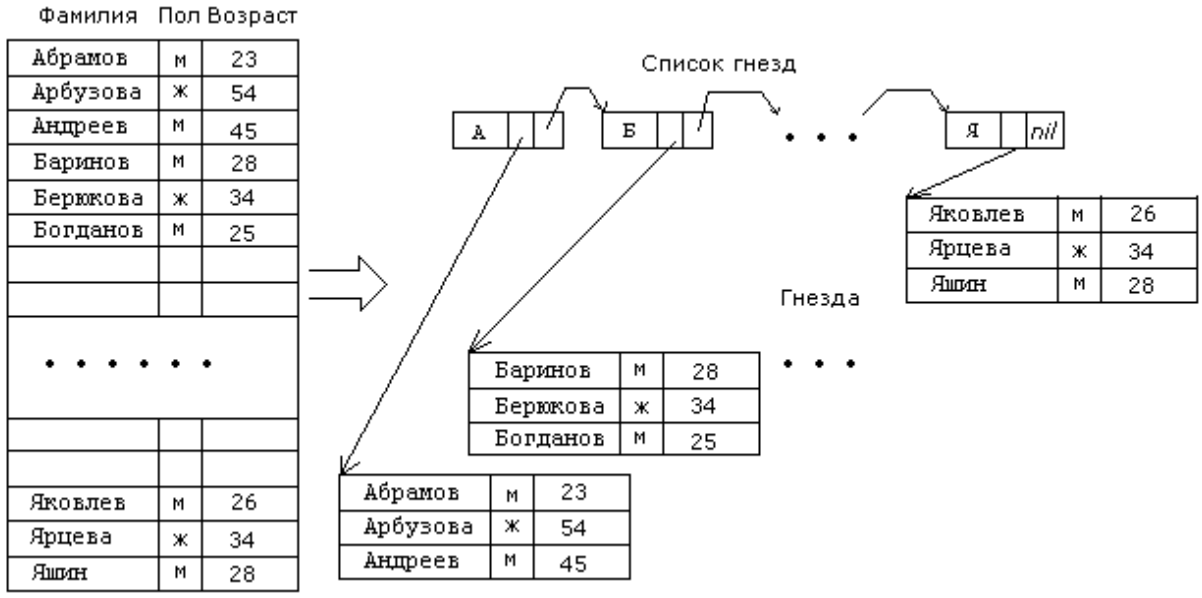

Рисунок 5 - Пример организации данных с помощью списка гнезд

Учитывая особенности данной задачи, можно предложить другую организацию гнездового способа. На рисунке 6 представлена структура, где для списка гнезд используется массив, а элементы каждого гнезда объединены в отдельном списке. Такой подход учитывает, что количество букв в алфавите фиксированное число, а количество фамилий может изменяться.

Рисунок 6 - Пример организации данных с помощью массива гнезд

Для числовых полей также можно формировать гнезда, например, разбив все множество записей по количеству значащих цифр и др.

Обработка древовидных списков

Обработка бинарных деревьев. Бинарные деревья имеют порядок 2. Подчиненность вершин бинарного дерева задается адресами связи. Каждая вершина i-го уровня содержит два адреса вершин i+1-го уровня, которые делятся на правый и левый, и один адрес вершины i-1-го уровня, который называется обратным.

В упорядоченном бинарном дереве значение ключа каждого элемента больше, чем значение ключа у любого элемента его левой ветви, и не больше, чем значение ключа любого элемента его правой ветви.

Формирование упорядоченного бинарного дерева.

Алгоритм А.

Первый элемент массива следует поместить в корень дерева. Значение ключа второго элемента p_2 сравниваем с ключом p_1 корня дерева. Если $p_2 < p_1$, то добавляем p_2 к левой от корня ветви. Если $p_2 \geq p_1$, то p_2 добавляем к правой от корня ветви. Для добавления следующих элементов осуществляют поиск первого свободного места, продвигаясь по дереву следующим образом: если $p_i < p_k$, то необходимо перейти к вершине, находящейся на левой ветви от p_k., иначе необходимо перейти к вершине, находящейся на правой ветви от p_k . Если такой вершины нет, то она достраивается и в нее помещается i-й элемент массива.

Алгоритм Б.

Исходные записи должны иметь вид упорядоченного массива. За корень дерева принимается средняя запись массива, которая разделяет массив на примерно равные части. Средние записи в обеих частях массива образуют вершины второго уровня, а массив оказывается разделенным на 4 части. Средние части каждой из четырех частей помещаются на третьем уровне бинарного дерева и т. д.

Включение новой записи в бинарное дерево происходит так же, как и при формировании дерева по алгоритму А.

Исключение концевой вершины сводится к удалению ее адреса из предшествующей вершины. При исключении вершины, имеющей одно поддерево, бинарное дерево распадается на части. Для восстановления связности дерева необходимо корень поддерева присоединить к основной части дерева, используя алгоритм А. При исключении вершины с двумя поддеревьями бинарное дерево распадается на три части. Корни поддеревьев в любом порядке присоединяются к основной части также согласно алгоритму А.

Древовидные структуры данных часто используются для реализации структур данных, элементы которых необходимо искать по составному ключу, включающему несколько признаков. В этом случае каждый признак соответствует вершине дерева: первый – вершине 1-го уровня; второй – вершине 2-го уровня и т. д. Если у двух составных ключей несколько начальных признаков совпадают, то они имеют общие вершины в дереве.

1.3 Выбор способа реализации абстрактных структур данных

Способ реализации абстрактных структур данных зависит от: частоты изменения данных, их структуры и ограничений на время доступа. Реализация включает физическую структуру данных, а также процедуры и функции, выполняющие операции над ними.

Физическая структура данных помимо информационных полей может содержать служебные поля, в которых хранится информация, необходимая для работы со структурой. Например, физическая структура односвязного списка представляет собой совокупность дескриптора и одинаковых по формату и размеру записей, размещенных произвольно в некоторой области памяти и связанных друг с другом в линейно упорядо-

ченную цепочку с помощью указателей. Дескриптор может содержать информацию: код структуры, имя списка, адрес начала, текущее число элементов и т. п. В линейном двусвязном списке имеется два указателя: прямой (указывает на следующий элемент) и обратный (указывает на предыдущий элемент).

Односвязный список всегда линейный. Двусвязный может быть нелинейным, если второй указатель каждого элемента в списке задает порядок произвольного типа. На рисунке 7 представлен двусвязный список, в котором первый указатель осуществляет связь между элементами какой-либо группы, а второй осуществляет связь внутри подгруппы. S – указатель начала всего списка.

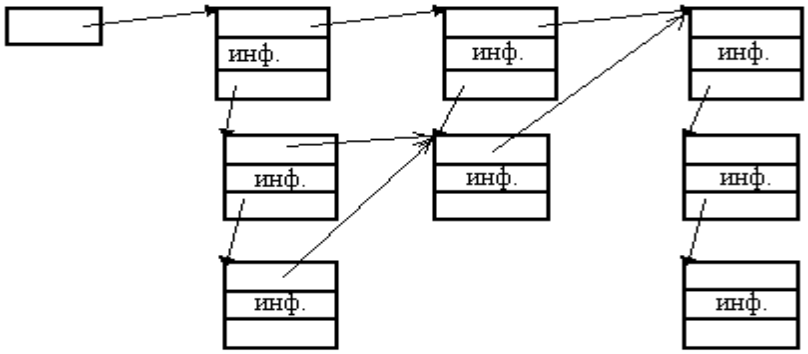

Рисунок 7 - Пример двусвязного нелинейного списка

Для конкретной задачи обычно можно предложить несколько реализаций выбранной структуры, которые отличаются временами выполнения отдельных операций (например, см. рисунок 8 – девять способов реализации графа).

Сравнение вариантов выполняется по требуемым объемам памяти (*емкостной сложности*) и времени выполнения операций (*временной сложности*).

Емкостная сложность оценивается по физической структуре с учетом используемых типов данных.

Для оценки временной сложности можно использовать следующую методику: 1) разрабатывается фрагмент алгоритма и по нему примерно определяется последовательность выполнения команд, реализующих ту или иную операцию над данными; 2) полученный результат преобразуется по таблице соответствий (см. таблицу 1) к времени выполнения фрагмента алгоритма в машинных тактах.

Таблица 1 - Коэффициенты приведения некоторых команд Borland Pascal (для Pentium)

Операция	Обозначение	Время, тактов	Операции	Обозначение	Время, тактов
a:=b	$t_{=}$	2	a/b	$t_{/}$	28
c:=a±b, a>b	t_{+}	2	a:=b^.num	$t_{=\wedge}$	2
A:=a±b	$t_{=+}$	2*	b:=b^.p	$t_{\wedge}$	2*
a:=a±const	t_{++}	1	сдвиги	$t_{\rightarrow}$	1
A±const	t_{+c}	1	a[i]	t_i	2
a*b	t_{*}	20	a[i,j]	$t_{i,j}$	26
if	t_{if}	$t_{пр}$+1+p1×t1+ +p2×t2 **	for	t_{for}	$t_{уст}+t_{пр}+1+$ $+n(t_{тела}+t_{пр}+2)$ ***

* – операции накопления/переадресации (при использовании в цикле коэффициент равен 1);

**- $t_{пр}$ – время проверки условия, p1,p2 – вероятности выбора соответствующих ветвей, t1,t2 – времена выполнения ветвей;

***- $t_{уст}$ – время установки начального значения индекса, $t_{пр}$ – время проверки условия, $t_{тела}$ – время выполнения тела цикла; для цикла вида **for i:=1 to n do** … получаем: $t=2+2+1+n(t_{тела}+2+2)= 5+n(t_{тела}+4)$.

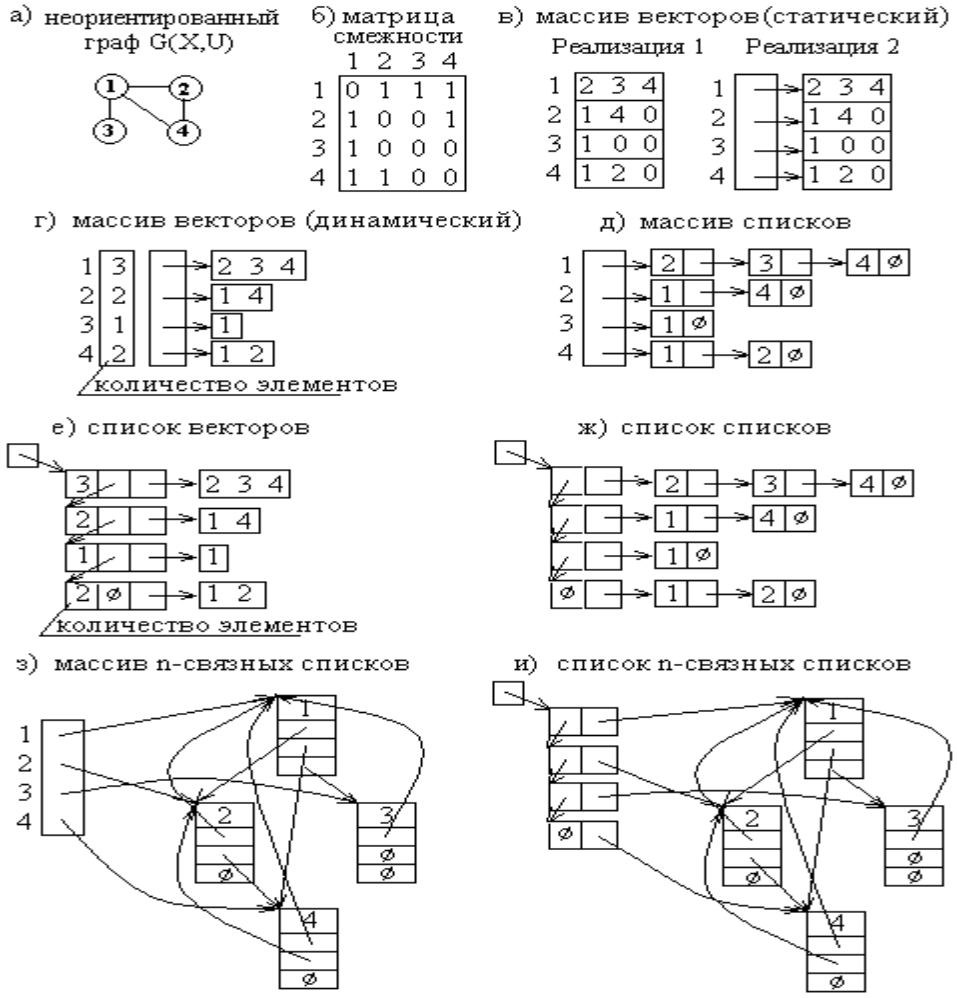

Рисунок 8 - Различные способы реализации графов

Пример. Выполнить оценки памяти и вычислительной сложности выполнения операций поиска и удаления элемента для двух реализаций хранения последовательности n целых чисел: в виде массива и в виде линейного односвязного списка (см. рис. 9).

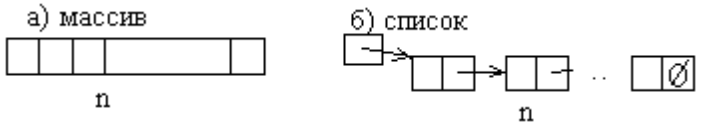

Рисунок 9 – Массив (а) и односвязный список (б)

Решение.

1. *Оценка памяти*:

а) для массива: $V= l_{эл}*n$, если в массиве хранятся целые числа, то $V=2n$ (байт)

б) для списка: $V=(lэл+lук)*n+lук$,

если в списке хранятся целые числа, а указатели имеют длину 4 байта, то $V=6n+4$ (байт)

2. *Оценка времени поиска i-го элемента данных:*

а) для массива:

поиск выполняется по индексу $A_i =A_б+(i-1)*2=A_б-2+i*2=const+i*2$,

операция умножения на 2 выполняется посредством сдвига на 1 разряд влево,

откуда $t= t_{++} + t \rightarrow = 2$ (такта)

б) для списка:

доступ к элементу выполняется последовательно **f:=first; for j:=1 to i do f:=f^.p;**
определяем $t = 2+2+1+(i-1)*(t_{\wedge}+2+2)=5+5(i-1)$,
минимальное время – первого элемента $t_{min}=5$,
максимальное время поиска – последнего элемента $t_{max} =5+(n-1)(t_{\wedge}+4)=5+5n-5=5n$,
в среднем $t= (t_{max}+t_{min})/2= 2.5+2.5n$ (тактов).

3. *Оценка времени удаления i-го элемента:*

а) для массива:

при удалении *i*-го элемента выполняется сдвиг остальной части массива: **for j:=i to n-1 do a[j] := a[j+1];**
откуда $t=t_{+c}+2+2+(n-i)\ (t_{+c}+t_i+t_i+t_{=})=5+7(n-i)$,
минимальное время работы цикла (при $i=n-1$) $t_{min}=12$,
максимальное время работы цикла (при $i=1$) $t_{max}=5+7(n-1)=-2+7n$,
соответственно среднее время работы цикла $t=5+3,5n$ (тактов).

При $i=n$ сдвиг не выполняется, а только значение n уменьшается на единицу (t_{++} =1). Тогда среднее время удаления равно: $t = t_{пр} + (t_{++} + t_{max})/2 = 2+(1-2+7n)/2 =1,5 + 3,5n$ (таков)

б) для списка:

при удалении элемента достаточно перезаписать адресное поле: **f^.p:=r^.p** (без учета времени освобождения памяти и времени поиска элемента), следовательно, $t=t_{\wedge}+t_{=\wedge}=3$ (такта).

При выборе структуры учитывают удельный вес операций анализируемых операций.

2. Порядок выполнения работы

1. Ознакомиться с теоретическими сведениями по абстрактным структурам данных и методами их обработки.

2. Для указанной задачи и типа данных (см. таблицу 2) предложить способ реализации и определить требуемый объем памяти.

3. Провести анализ заданных методов поиска, упорядочения и корректировки. Оценить время выполнения соответствующих операций.

Таблица 2 - Варианты заданий.

Вариант	Задача	Структура	Поиск	Упорядочение	Корректировка
1	1	таблица	последоват.	пузырьком	удаление сдвигом
2	1	таблица	дихотомич.	вставкой	удаление маркир.
3	1	таблица	вычисл. адр.	Шелла	замена данных
4	1	таблица	последоват.	кв. выборки	удаление сдвигом
5	2	таблица	дихотомич.	слиянием	вставка записи
6	2	таблица	последоват.	пузырьком	удаление маркир.
7	2	таблица	дихотомич.	вставкой	удаление сдвигом
8	2	таблица	вычисл. адр.	Шелла	замена данных
9	2	таблица	последоват.	кв. выборки	удаление сдвигом
10	2	таблица	дихотомич.	слиянием	удаление маркир.
11	2	таблица	вычисл. адр.	пузырьком	удаление сдвигом
12	3	таблица	дихотомич.	вставкой	удаление маркир.
13	3	таблица	вычисл. адр.	Шелла	замена данных
14	3	таблица	последоват.	кв. выборки	удаление сдвигом
15	3	таблица	дихотомич.	слиянием	удаление маркир.

16	4	список	гнездовой	любой	удаление записи
17	4	список	последоват.	любой	удаление записи
18	4	кольц.сп.	последоват.	любой	удаление записи
19	4	$2^{св}$ список	последоват.	любой	замена данных
20	5	список	гнездовой	любой	удаление записи
21	5	список	последоват.	любой	удаление записи
22	5	кольц.сп.	последоват.	любой	удаление записи
23	5	$2^{св}$ список	последоват.	любой	удаление записи
24	6	список	гнездовой	любой	удаление записи
25	6	список	последоват.	любой	удаление записи
26	6	кольц.сп.	последоват.	любой	удаление записи
27	6	$2^{св}$ список	последоват.	любой	удаление записи
28	7	нелин. $2^{св}$ список	последоват.	любой	удаление записи
29	7	дерево	гнездовой	любой	удаление записи
30	6	дерево	гнездовой	любой	удаление записи
31	8	бин. упор. дерево (алгоритм А)	последоват.	нет	исключения неполной вершины
32	8	"-" (Б)	последоват.	нет	исключения неполной вершины
33	9	"-" (А)	последоват.	нет	исключения неполной вершины
34	9	"-" (Б)	последоват.	нет	исключения полной вершины
35	10	"-" (А)	последоват.	нет	исключения неполной и полной вершин

4. Предложить альтернативный вариант решения задачи, в котором должно быть минимум одно улучшение. Улучшения могут касаться как структуры данных, так и основных операций. Обосновать новые решения, используя количественные и качественные критерии. Количественными критериями являются: объем памяти, среднее количество сравнений и количество тактов. Качественные критерии определяют возможность использования того или иного метода применительно к разработанной структуре. К ним можно отнести: применимость операции только к упорядоченным данным; необходимость знать количество элементов; наличие признака разбивки на гнезда; необходимость в прямом доступе к элементам; знание граничных значений; невозможность создать структуру в соответствии с арифметической прогрессией и др.

Отчет должен включать:

- название работы и ее цель;
- конкретный вариант задания;
- рисунок физической структуры данных в соответствии с вариантом задания;
- оценку требуемого объема памяти в зависимости от количества элементов;
- описания или схемы алгоритмов оцениваемых операций;
- расчеты оценок вычислительной сложности в зависимости от количества элементов для указанных методов обработки;

- описание альтернативного варианта и его оценки;
- выводы о том, в каких случаях должен использоваться основной, а в каких – альтернативный вариант реализации.

Примечания

1. Если структура данных в альтернативном варианте заменена, а метод реализующий операцию остался прежним, то для обоснования необходимо определить количество тактов.

2. Результаты выполнения лабораторной работы свести в таблицу.

Задачи

Задача 1. Даны N записей вида: код материала; дата поступления; номер склада; количество; сумма.

Задача 2. Дана таблица материальных нормативов, состоящая из K записей фиксированной длины вида: код детали; код материала; единица измерения; номер цеха; норма расхода.

Задача 3. Даны M записей вида: код группы; ФИО; дата рождения.

Задача 4. Дано N элементов, где каждый элемент является предложением на естественном языке.

Задача 5. Даны элементы: 230, 150, 100, 85, 77, 33, 93, 200, 57, 137, 205.

Задача 6. Даны элементы: 130, 50, 120, 185, 27, 43, 913, 210, 5, 17, 245.

Задача 7. «Поток ИУ-6». В потоке имеется 6 групп, в каждой группе не более 25 человек. Необходимо хранить следующую информацию о каждом студенте: ФИО, дата рождения, пол.

Задача 8. Даны ключи: 61, 36, 50, 42, 17, 75, 54, 14, 90, 254, 134, 248, 123.

Задача 9. Даны ключи: 81, 122, 51, 3, 52, 116, 79, 46,178, 44, 124, 45, 165, 38, 198, 9, 7.

*Задача 10.*Даны ключи: 88, 36, 57, 47, 57, 85, 343, 64,72, 638, 596, 94, 7, 24, 514, 36, 173, 89, 673,524.

Список литературы

1. Диго С.М., Клешко Г.Н., Мишенин А.И., Петров Е.А. Сборник задач по курсу «Информационные системы и структуры данных». – М.:Статистика,1981.

2. Костин А.Е., Шаньгин В.Ф. Организация и обработка структур данных в вычислительных системах. -М.: Высшая школа,1987.

3. Лэнгсам Й., Огенстайн М., Тененбаум А. Структуры данных для персональных ЭВМ. – М: Мир,1989.

4. Вирт Н. Алгоритмы и структуры данных. -М: Мир,1989.

5. Иванова Г.С. . Иванова Г.С. Технология программирования: Учебник. - М: Кнорус, 2011 – 333 с.

6. Иванова Г.С. Основы программирования. Учеб. для ВУЗов. М: Из-во МГТУ им. Баумана, 2003.