СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Методическое пособие для заочного обучения

Часть 1
Методы сортировки и поиска

Новосибирск 2006

ЦЕЛЬ КУРСА

В результате изучения курса студент должен

знать о методах сортировки массивов и последовательностей, а также о трудоемкости рассматриваемых алгоритмов;

уметь разрабатывать программы для решения задач сортировки и поиска информации;

иметь навыки применения методов сортировки массивов и последовательностей.

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

1. Основные структуры данных.
1.1 Основные структуры данных. 1.2 Задача сортировки массивов. 1.3 Трудоемкость методов сортировки массивов. 1.4 Задача сортировки последовательностей. 1.5 Теорема о сложности сортировки. 1.6 Задача поиска элементов с заданным ключом.

2. Методы сортировки с квадратичной трудоемкостью.
2.1 Метод прямого выбора. 2.2 Пузырьковая сортировка 2.3 Шейкерная сортировка.

3. Быстрые методы сортировки массивов.
3.1 Метод Шелла 3.2 Пирамидальная сортировка. 3.3 Метод Хоара. 3.4 Проблема глубины рекурсии.

4. Работа с линейными списками
4.1 Указатели. Основные операции с указателями. 4.2 Основные операции с линейными списками.

5. Методы сортировки последовательностей
5.1 Метод прямого слияния 5.2 Цифровая сортировка.

6. Двоичный поиск в упорядоченном массиве.
6.1 Алгоритм двоичного поиска 6.2 Две версии поиска

7. Сортировка данных с произвольной структурой
7.1. Сравнение данных произвольной структуры 7.2 Сортировка по множеству ключей. Индексация 7.3 Индексация через массив указателей

8. Хэширование и поиск
8.1. Понятие хэш-функции 8.2 Метод прямого связывания 8.3 Метод открытой адресации. Линейные и квадратичные пробы.

ВВЕДЕНИЕ

Изучение дисциплины «Структуры и алгоритмы обработки данных» является одним из основных моментов в процессе подготовки специалистов по разработке программного обеспечения для компьютерных систем. Это связано с тем, что первичная задача программиста заключается в применении решения о форме представления данных и выборе алгоритмов, применяемых к этим данным. И лишь затем выбранная структура программы и данных реализуется на конкретном языке программирования. В связи с этим знание классических методов и приемов обработки данных позволяет избежать ошибок, которые могут возникать при чисто интуитивной разработке программ.

Данное пособие содержит необходимый теоретический материал по трем основным разделам курса «Структуры и алгоритмы обработки данных»: методы сортировки данных, древовидные структуры данных и хеширование информации. Все рассмотренные методы проиллюстрированы наглядными примерами. Также для каждого метода приведен конкретный алгоритм, позволяющий легко программировать данный метод. Кроме того, в каждой главе даны контрольные вопросы для самопроверки.

1. НЕОБХОДИМЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

1.1 Основные структуры данных

Любая программа в процессе работы обрабатывает некоторые данные. По способу представления в памяти компьютера данные можно разделить на две группы: статические и динамические. Статические данные имеют фиксированную структуру, поэтому размер выделенной для них памяти постоянен. Динамические данные изменяют свою структуру в процессе работы программы, при этом объём памяти изменяется. Основные структуры данных представлены в следующей таблице.

Таблица 1 Различные типы данных

Статические	Простые	Перечислимые
		Целые
		Вещественные
		Логические
		Символьные
	Структурированные	Массивы
		Записи
		Объединения
Динамические	Списки	Стек
		Очередь
	Деревья	АВЛ-деревья
		Б-деревья

Основной характеристикой данных в программировании является тип

данных. Любая константа, переменная, выражение, функция всегда относятся к определенному типу. Тип характеризует множество значений, которые может принимать переменная. К простым типам языка программирования относятся целые, вещественные, логические, символьные. Целые типы в языках программирования различаются количеством байт, отведённых в памяти (диапазоном значений) и наличием знака. Вещественные типы характеризуются точностью представления числа. Перечислимые типы образуются в процессе перечисления всех возможных значений. Логический тип является частным случаем перечислимого типа с двумя возможными значениями ИСТИНА и ЛОЖЬ.

Структурированные (составные) типы отличаются от простых тем, что состоят из набора компонент одинакового или разного типа. К структурированным типам относят массивы, записи (структуры), объединения (записи с вариантами).

Массивы – это наиболее известная и распространённая структура данных. Массив представляет собой фиксированное количество элементов одного типа. Всем элементам присваивается одно имя, а к отдельному элементу обращаются по его номеру (индексу). Тип элементов массива может быть любым, но тип индексов должен быть скалярным. Массив – структура данных со случайным (прямым) доступом, т.е. в любой момент времени доступен любой элемент массива, при этом индекс элемента можно вычислять. Обычный способ работы с массивом – выбор определённых элементов и их изменение. Также часто используется перебор элементов в цикле. В памяти элементы массива располагаются последовательно, без разрывов, по возрастанию адресов памяти.

Другой вид структурированных типов – запись состоит из фиксированного числа компонент называемых **полями**, которые в отличие от элементов массива могут быть разных типов. **Запись** также является структурой данных со случайным доступом.

Объединения или (записи с вариантами) используются для размещения в одной и той же области памяти данных различного типа. В один и тот же момент времени в памяти находятся данные только одного типа.

Динамические структуры данных позволяют работать с большими объемами данных. Наиболее используемые динамические структуры данных – списки и деревья.

1.2 Задача сортировки массивов

Пусть дан массив A=(a_1, a_2, …, a_n) и для всех его элементов определены операции отношения: меньше, больше, равно (<, >, =). Необходимо отсортировать массив, т.е. переставить элементы массива A таким образом, что выполняется одно из следующих неравенств:

$$a_1 \le a_2 \le a_3 \le \ldots \le a_n$$
$$a_1 \ge a_2 \ge a_3 \ge \ldots \ge a_n$$

Если выполняется первое неравенство, то массив сортируется по возрастанию и такой порядок элементов будем называть *прямым*. Если выполняется второе неравенство, то массив отсортирован по убыванию и такой

порядок элементов будем называть *обратным*. В дальнейшем, если не оговорено особо, используется прямой порядок сортировки.

При сортировке массивов элементов со сложной структурой возникает задача определить отношение порядка между элементами. Та часть информации, которая учитывается при определении отношения порядка называется *ключом сортировки.*

Сортировка называется *устойчивой*, если после её проведения в массиве не меняется относительный порядок элементов с одинаковыми ключами.

Для проверки правильности сортировки массива могут использоваться следующие приемы. Вычисление контрольной суммы элементов массива до и после сортировки дает возможность проверить потерю элементов массива во время процесса сортировки. Определение количества серий элементов массива, т.е. неубывающих последовательностей из элементов массива, позволяет проверить правильность упорядочивания массива, поскольку массив, отсортированный по возрастанию, состоит из одной серии, а в массиве, отсортированном по убыванию, количество серий равно количеству элементов в массиве.

1.3 Трудоемкость методов сортировки массивов

Существует много способов или методов сортировки массивов. Для того, чтобы оценить насколько один метод сортировки лучше другого необходимо каким-то образом оценивать эффективность метода сортировки. Естественно отличать методы сортировки по времени, затрачиваемому на реализацию сортировки. Для сортировок основными считаются две операции: операция сравнения элементов и операция пересылки элемента. Будем считать, что в единицу времени выполняется одна операция сравнения или пересылки. Таким образом, время или трудоемкость метода имеет две составляющие M и C, где

M – количество операций пересылки.

C– количество операций сравнения.

Нетрудно видеть, что M и C – зависят от количества элементов в массиве, т.е. являются функциями от длины массива. Часто бывает трудно определить точное выражение для трудоемкости алгоритма. В этом случае пользуются асимптотической оценкой времени работы.

Будем говорить, что функция $g(x)$ *асимптотически доминирует* на f(x) или $g(x)=O(f(x))$, если $|g(x)| \le const|f(x)|$ при $x \to \infty$. В дальнейшем будем рассматривать асимптотическое поведение величин M и C в зависимости от числа элементов в массиве n, при $n \to \infty$.

Для функций f, f_1, f_2 и константы k справедливы свойства:

1. $f = O(f)$
2. $O(k*f) = O(f)$
3. $O(f_1+f_2) = O(f_1) + O(f_2)$

Пример. $T = 10n + 20$

$T = O(10n+20) = O(10n) + O(20) = O(n) + O(1) = O(n)$, при $n \to \infty$.

Приведем ряд доминирования основных функций

$O(1)<O(\log n)<O(n)<O(n \log n)<O(n^a)<O(a^n)<O(n!)<O(n^n)$, при $n \to \infty$, $a>1$

Поскольку для различных массивов один и тот же метод сортировки может иметь различную трудоемкость, то необходимо знать в каких пределах могут изменяться величины характеризующие трудоемкость, т.е. определить минимальное и максимальное значения трудоемкости и массивы, на которых достигаются эти значения, а также средние значения величин M и C.

1.4 Задача сортировки последовательностей

Пусть дана последовательность $S=S_1S_2 \ldots S_n$, т.е. совокупность данных с последовательным доступом к элементам. Примерами такой последовательности могут служить файл на магнитной ленте, линейный список:

Необходимо переставить элементы последовательности так, чтобы выполнялись неравенства:$S_1 \leq S_2 \leq \ldots \leq S_n$ Последовательный доступ означает, что любой элемент списка может быть получен только путём просмотра предыдущих элементов, причём просмотр возможен только в одном направлении. Это является существенным ограничением по сравнению с массивом, где можно было обратиться к любому элементу массиву, используя индекс. Поэтому методы сортировки, разработанные специально для массивов, не годятся для последовательностей, в то время как методы сортировки последовательностей используются и для сортировки массивов. Трудоемкость методов сортировки последовательностей измеряется количеством операций, затрачиваемых на сортировку. Характерными операциями при сортировке последовательностей являются операция сравнения элементов и операция пересылки элемента одной последовательности в другую. Как и прежде будем обозначать количества операций сравнения и пересылки C и M соответственно.

1.5 Теорема о сложности сортировки

При изучении различных методов сортировок возникает закономерный вопрос о построении метода сортировки с минимально возможной трудоемкостью. Следующая теорема устанавливает нижнюю границу трудоемкости для сортировки массива из *n* элементов.

Теорема. *Если все перестановки из n элементов равновероятны, то любое дерево решений, сортирующее последовательность из n элементов имеет среднюю высоту не менее log(n!).*

Приведем нестрогое доказательство. Рассмотрим дерево решений для трех элементов a, b, c.

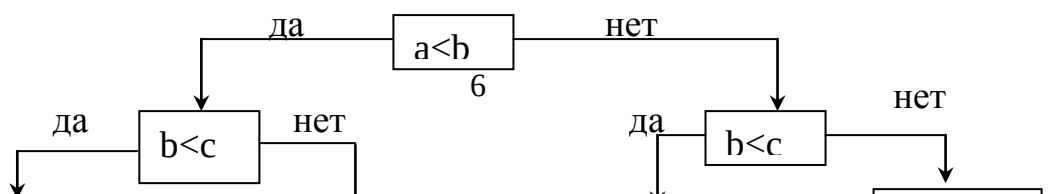


Рисунок 1 Дерево решений для 6 элементов

Все возможные перестановки – это листья дерева (6 вариантов). Чтобы получить конкретную перестановку нужно сделать два или три сравнения. Оценим среднее количество сравнений, необходимых для упорядочивания массива или среднюю длину пути от корня дерева до листьев. Для этого посчитаем сумму длин всевозможных путей от корня до листьев (длина внешнего пути двоичного дерева) и поделим ее на количество листьев $C_{ср}$= (2+3+3+3+3+2)/6=2,6.

Из теории графов известно, что длина внешнего пути двоичного дерева с m листьями $D(m) \geq m \log m$. Поскольку в общем случае на дереве имеется $n!$ листьев. Тогда $C_{ср} \geq n! \log(n!)/n! = \log n! > n \log n - n \log e$. Последнее неравенство является известной нижней оценкой для значения факториала. Таким образом, не существует алгоритма сортировки n элементов, использующего в среднем меньше чем ($n \log n - \log e$) операций сравнения. Класс сложности $n \log n$ является предельно достижимым для алгоритмов сортировки с использованием операций сравнения. Что касается количества пересылок, то если мы определим требуемую перестановку, и имеем память для второй копии массива, то достаточно сделать n пересылок. На сегодняшний день алгоритм, требующий $n \log n$ сравнений и n пересылок, неизвестен.

1.6 Задача поиска элементов с заданным ключом

Пусть имеется массив $A = (a_1, a_2, \ldots a_n)$ и задан ключ поиска X. Необходимо найти элемент (элементы) массива с ключом X, или определить, что элемента с заданным ключом в массиве нет. Если массив не упорядочен, то единственный путь поиска – просмотр всех элементов массива (линейный поиск) до тех пор, пока не будет найден этот элемент, или просмотрен весь массив. Трудоёмкость поиска в этом случае равна $O(n)$, $n \to \infty$. Более эффективные алгоритмы поиска можно построить, если предполагать, что массив отсортирован, т.е. $a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_n$. Трудоемкость метода поиска будем оценивать по количеству сравнений, требуемых для поиска нужного элемента, при этом нас интересует только асимптотическая оценка трудоемкости.

1.7 Контрольные вопросы

1. Перечислите основные типы данных
2. Сформулируйте задачу сортировки массивов.
3. Как измеряется трудоемкость сортировки массивов?
4. Сформулируйте задачу сортировки последовательностей
5. Сформулируйте задачу поиска заданного элемента в массиве
6. Назовите предельную сложность для задачи сортировки массивов.

2. МЕТОДЫ СОРТИРОВКИ С КВАДРАТИЧНОЙ ТРУДОЕМКОСТЬЮ

2.1 Метод прямого выбора

Один из самых простых методов сортировки, метод прямого выбора, заключается в следующем. Находим наименьший элемент массива и обмениваем его с первым элементом массива. Таким образом, первый элемент можно больше не рассматривать. Среди оставшихся элементов находим наименьший элемент и обмениваем его со вторым элементом массива. Среди оставшихся элементов находим наименьший и переставляем его на третье место и т. д.

Пример. Упорядочим слово методом прямого выбора.

Условные обозначения

X сравнение элемента X с текущим максимальным элементом

Ẋ текущий максимальный элемент

Перестановка элементов

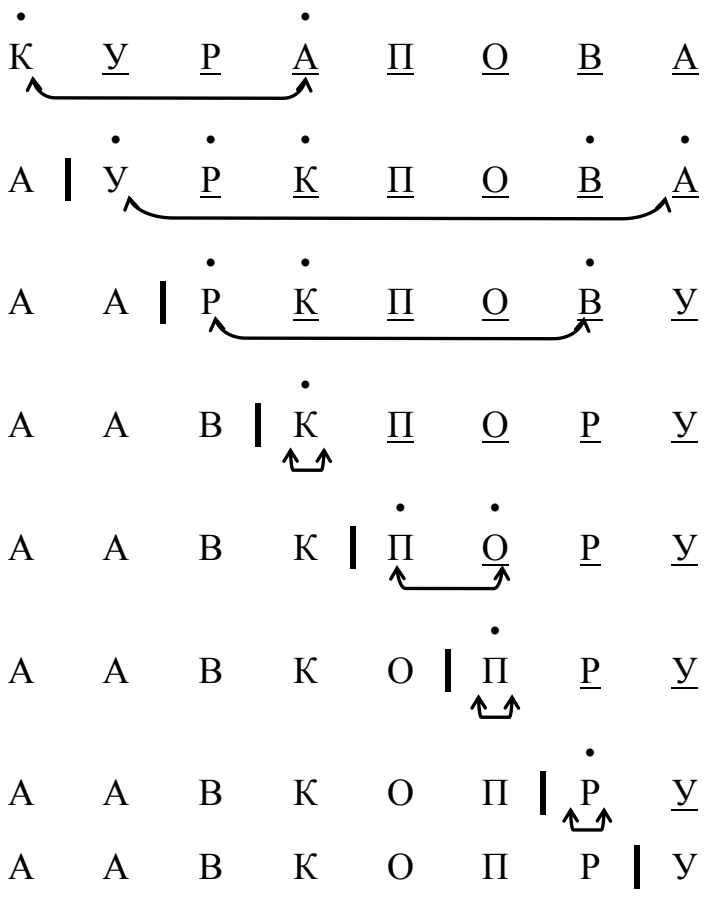


Рисунок 2 Метод прямого выбора

Алгоритм на псевдокоде

Метод прямого выбора

DO (i = 1, 2, ... n-1)

k:=<номер наименьшего элемента из a_i,… a_n>

$a_i \leftrightarrow a_k$

OD

Дадим оценку трудоёмкости метода прямого выбора. В данном случае можно найти точные выражения для M и C. Поскольку на каждой итерации происходит точно один обмен, то M = *3(n-1).* Определим теперь количество сравнений. На первом этапе имеем *(n-1)* сравнений, на втором – *(n-2)* сравнений, на i-том этапе происходит *(n- i)* сравнений и т.д. Суммируя, получим C = $\frac{n^2 - n}{2}$

Отметим важные особенности метода прямого выбора. Метод не зависит от исходной отсортированности массива, т.е. значения M и C не меняются, даже если сортируется уже отсортированный массив. Сортировка методом прямого выбора неустойчива.

2.2 Пузырьковая сортировка

Популярный метод пузырьковой сортировки заключается в следующем. Двигаясь от конца массива к его началу, будем сравнивать между собой соседние элементы. При этом если правый элемент a_j будет меньше чем левый a_{j-1}, j=*n, n-1,* … ,2, то поменяем их местами. Таким образом, при первом проходе наименьший элемент переместится на первое место и можно не учитывать его при сортировке оставшейся части массива. При втором проходе наименьший элемент из оставшихся “всплывёт” на второе место. Через (*n-1*) итераций массив будет отсортирован.

Пример. Отсортировать слово методом пузырьковой сортировки. Подчёркнуты те пары, в которых произошёл обмен. Вертикальной чертой ограничена отсортированная часть массива.

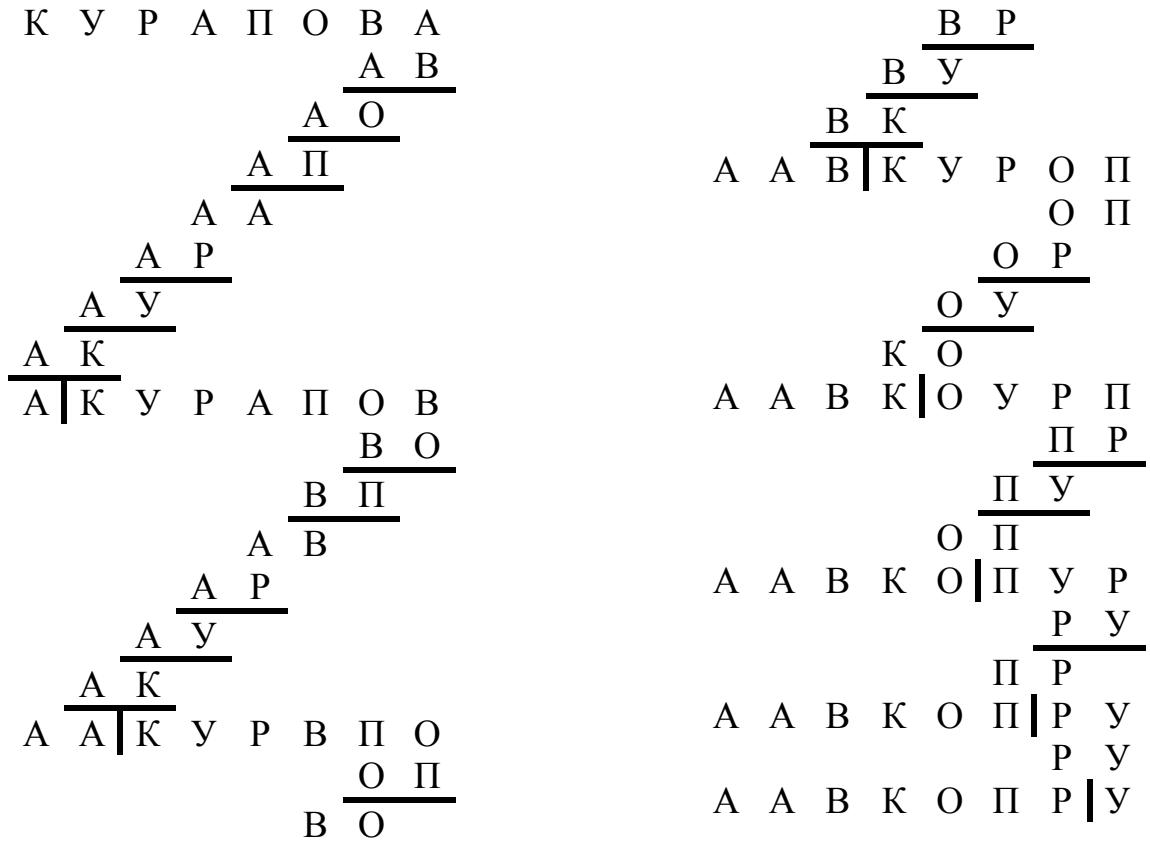


Рисунок 3 Пузырьковая сортировка

Алгоритм на псевдокоде

Пузырьковая сортировка

Обозначим i – номер итерации,

j – индекс правого элемента в текущей паре.

```
DO (i= 1, 2, ... n-1)
    DO (j= n, n-1, ... i+1)
        IF (a_j < a_j-1) a_j ↔ a_j-1 FI
    OD
OD
```

Проанализируем сложность пузырьковой сортировки. Количество сравнений в методе прямого выбора и в методе пузырьковой сортировки одинаково и не зависит от исходной отсортированности массива. Однако количество пересылок М зависит от того, как часто выполняется условие $a_j < a_{j-1}$. Можно определить максимальное и минимальное значения $M_{min} \le M_{сред} \le M_{max}$,. где

$M_{min} = 0$, при сортировке упорядоченного по возрастанию массива.

$M_{max} = ЗС = \frac{3(n^2 - n)}{2}$, при сортировке упорядоченного по убыванию массива.

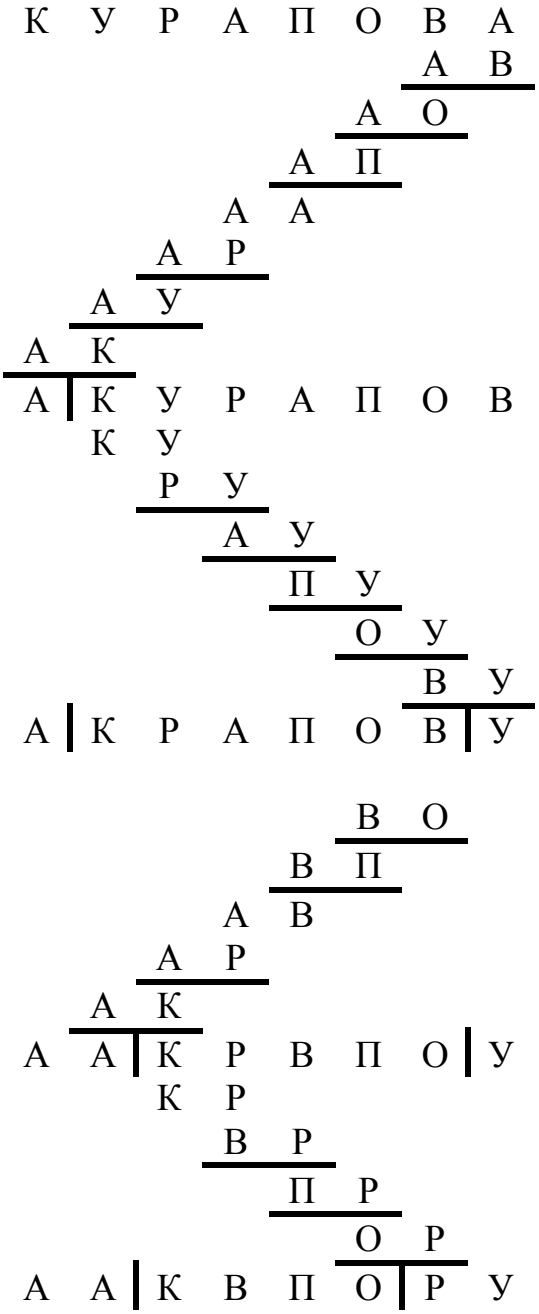
Отсюда следует, что $M_{сред}=O(n^2)$, при $n \to \infty$

Таким образом, пузырьковая сортировка сильно зависит от исходной упорядоченности массива по количеству сравнений. Метод обеспечивает устойчивую сортировку.

2.3 Шейкерная сортировка

Анализируя метод пузырьковой сортировки можно отметить два обстоятельства. Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, ее можно исключить из рассмотрения. Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо. Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (т.е. части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

Пример. Отсортировать слово методом шейкерной сортировки. Подчёркнуты те пары, в которых произошёл обмен. Вертикальной чертой ограничена рабочая часть массива.



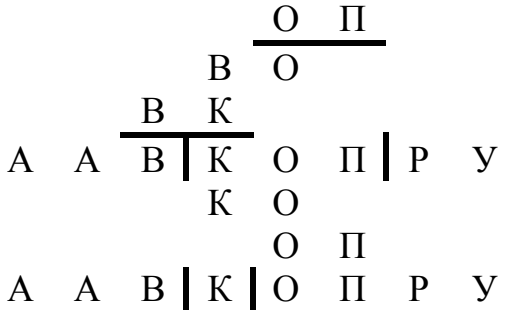


Рисунок 4 Шейкерная сортировка

Алгоритм на псевдокоде

Метод шейкерной сортировки

Обозначим

L – левая граница рабочей части массива.

R – правая граница рабочей части массива.

n – количество элементов в массиве

```
L: = 1, R: = n, k: = n,
DO
    DO (j =R, R-1, ... L+1)
        IF (a_j < a_j-1) a_j ↔ a_j-1, k: = j FI
    OD
    L: = k
    DO (j: = L, L+1, ... R-1)
        IF (a_j > a_j+1) a_j ↔ a_j+1, k: = j, FI
    OD
    R: = k
OD (L < R)
```

Оценим трудоемкость метода. Количество пересылок такое же, как и в методе пузырьковой сортировки $M_{сред}=O(n^2)$, при $n\to\infty$. Улучшения в методе шейкерной сортировки приводят к снижению количества сравнений. Точное выражение для величины С получить не удается, поэтому определим границы, в которых изменяется С. Если сортируется массив, в котором элементы расположены в порядке возрастания, то в методе шейкерной сортировки достаточно один раз просмотреть массив. Тогда $C_{min} = n-1$, где n – количество элементов в массиве. Если массив отсортирован в обратном порядке, то на каждой итерации границы слева и справа сдвигаются на одну позицию и $C_{max} = \frac{(n^2-n)}{2}$. Следовательно, $C_{сред}=O(n^2)$, при $n\to\infty$. Таким образом, как и пузырьковая сортировка, метод шейкерной сортировки сильно зависит от исходной упорядоченности массива по количеству сравнений. Метод обеспечивает устойчивую сортировку.

2.4 Контрольные вопросы

1. Сформулируйте основную идею метода прямого выбора.

2. Каковы теоретические оценки сложности метода пузырьковой сортировки?
3. Как метод пузырьковой сортировки зависит от начальной отсортированности массива?
4. Какова трудоемкость метода шейкерной сортировки?
5. Является ли метод шейкерной сортировки устойчивым?

3. МЕТОД ШЕЛЛА

3.1 Метод прямого включения

Сначала рассмотрим метод сортировки, который является базовым для метода Шелла. Метод прямого включения заключается в следующем. Начиная с $i = 2$, i=2,… n, берём очередной i–й элемент массива и включаем его на нужное место среди первых (i-1) элементов, при этом все элементы, которые больше a_i сдвигаются на одну позицию вправо.

Пример. Отсортировать слово методом прямого включения.

Условные обозначения

[X] i-тый элемент

X сравнение элемента X с i-тым элементом

↘ сдвиг элемента на одну позицию вправо

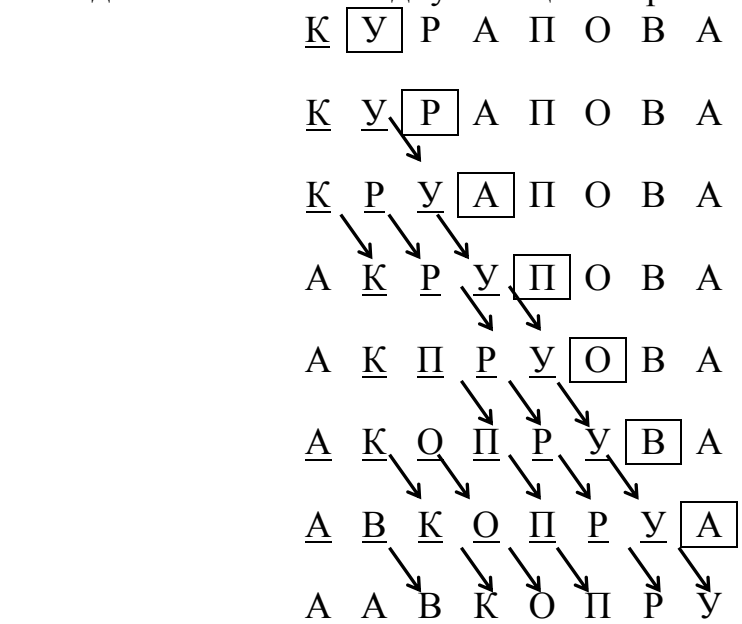


Рисунок 5 Метод прямого включения

Алгоритм на псевдокоде

Метод прямого включения

```
DO (i: = 2, 3, … n)
        t: = a_i, j: =i-1
        DO (j > 0 и t < a_j)
                a_{j+1}:= a_j
                j: = j-1
        OD
        a_{j+1}:= t
```

OD

Для метода прямого включения справедливы следующие оценки величин M и C.

$C_{min} \le C_{сред} \le C_{max}$, где $C_{min} = n-1$, $C_{max} = \frac{n^2 - n}{2}$,

$M_{min} \le M_{сред} \le M_{max}$, где $M_{min} = 2(n-1)$, $M_{max} = \frac{n^2 - n}{2} + 2n - 2$.

Минимальные и максимальные значения величин C и M достигаются на прямо отсортированном и обратно отсортированном массивах соответственно. Таким образом, средняя трудоемкость этого метода имеет квадратичный порядок, т.е. $C = O(n^2)$ $M = O(n^2)$, при $n \to \infty$.

Метод прямого включения устойчивый.

3.2 Метод Шелла

На базе метода прямого включения разработан алгоритм, обеспечивающий значительную производительность сортировки. Мы заметили, что в методе прямого включения, чем больше упорядочен массив, тем меньше операций требуется для его сортировки. При сортировке уже упорядоченного массива трудоемкость имеет линейный порядок. Поэтому имеет смысл попытаться предварительно несколько улучшить порядок элементов в массиве, а затем отсортировать массив методом прямого включения.

Предварительное упорядочивание будем проводить с помощью k – сортировок. Суть k – сортировки заключается в следующем. Массив разбивается на последовательности с шагом k

$a_i, a_{k+i}, a_{2k+i}, \ldots, a_{[n/k]k+i}$, $i = 1, 2, \ldots, k$

и сортировка происходит только внутри этих последовательностей.

Обозначим через H последовательность из m возрастающих шагов

$$H=(h_1, h_2, \ldots h_m), \text{ где } h_1=1, h_1 < h_2 < h_3 < \ldots < h_m$$

Метод Шелла состоит в последовательном проведении h_i-сортировки, $i=m, m-1,\ldots, 1$, причем $h_1=1$ гарантирует, что массив будет полностью отсортирован, поскольку 1-сортировка является методом прямого включения.

Пример. Отсортировать слово методом Шелла. Последовательность шагов выберем следующим образом $h_1=1$, $h_2=2$.

2-сортировка

К У Р А П О В А

К У Р А П О В А

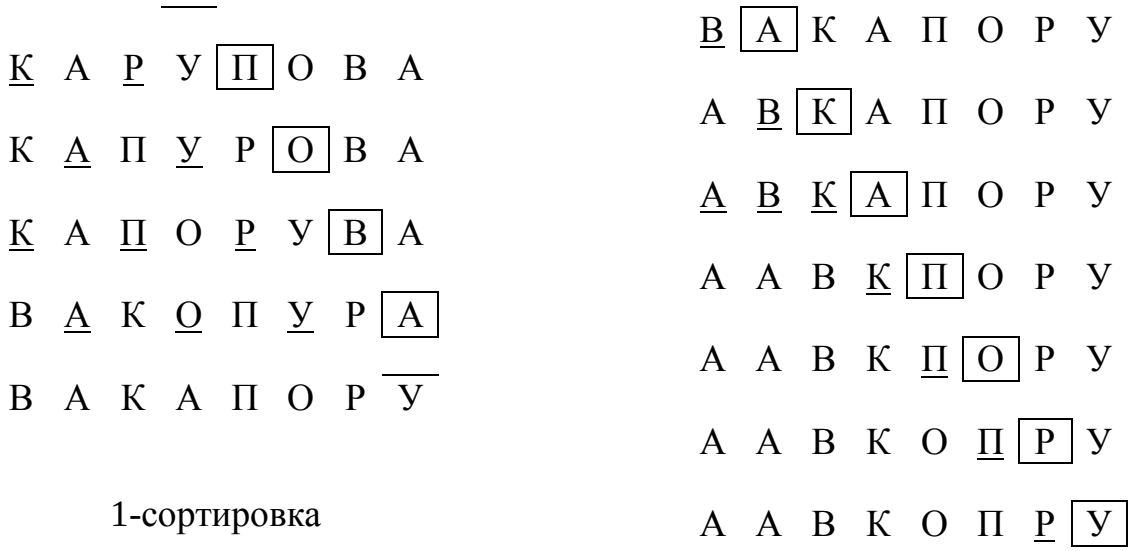


Рисунок 6 Метод Шелла

Алгоритм на псевдокоде

Сортировка методом Шелла

```
DO (k=h_m, h_m-1, … 1)
    DO (i=k+1, k+2, … n)
        t: = a_i, j: =i-k
        DO (j>0 и t < a_j)
            a_j+k: = a_j
            j: = j-k
        OD
        a_j+k: = t
    OD
OD
```

Эффективность метода зависит от выбора значений шагов. Последовательность значений шагов, которая дает наилучшую трудоемкость, пока неизвестна, метод продолжает исследоваться, но существует и часто используется следующая последовательность шагов, предложенная Кнутом.

$h_1=1,\ h_i=2h_{i-1}+1,\ i=2,\dots\ m,\ m=\lfloor \log_2 n \rfloor - 1$

При такой последовательности шагов средний порядок трудоёмкости $O(n^{1.2})$, $n \to \infty$. Таким образом, метод Шелла существенно быстрее методов с квадратичной трудоемкостью. Метод Шелла не устойчив.

3.3 Контрольные вопросы

1. Сформулируйте основную идею метода прямого включения.
2. Каковы теоретические оценки сложности метода прямого включения?
3. Что такое *n*-сортировка?
4. Как метод Шелла зависит от начальной отсортированности массива?
5. Какова трудоемкость метода Шелла?
6. Является ли метод Шелла устойчивым?
7. Каким образом выбирается последовательность шагов в методе Шелла?

4. БЫСТРЫЕ МЕТОДЫ СОРТИРОВКИ МАССИВОВ

4.1 Пирамидальная сортировка

Пирамидальная сортировка основана на алгоритме построения пирамиды. Последовательность $a_i, a_{i+1},\ldots,a_k$ называется *(i,k)-пирамидой*, если неравенство

$$a_j \le \min(a_{2j}, a_{2j+1}) \qquad (*)$$

выполняется для каждого j, j=i,…,k для которого хотя бы один из элементов a_{2j}, a_{2j+1} существует.

Например, массив A является пирамидой, а массив B — не является.

A=(a_2 , a_3 , a_4 , a_5 , a_6 a_7 , a_8)=(3, 2, 6, 4, 5, 7)

B=(b_1, b_2, b_3, b_4, b_5, b_6, b_7)=(3, 2, 6, 4, 5, 7)

Свойства пирамиды

1. Если последовательность $a_i, a_{i+1},\ldots,a_{k-1}, a_k$ является (*i, k*)-пирамидой, то последовательность $a_{i+1},\ldots,a_{k-1}$, полученная усечением элементов с обоих концов последовательности, является (*i+1, k-1*)пирамидой.
2. Если последовательность $a_1\ldots a_n$ – (*1, n*)-пирамида, то a_1 – минимальный элемент последовательности.
3. Если a_1, $a_2\ldots,a_{n/2},a_{n/2+1},\ldots a_n$-произвольная последовательность, то последовательность $a_{n/2+1},\ldots,a_n$ является (*n/2+1, n*)-пирамидой.

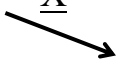
Процесс построения пирамиды выглядит следующим образом. Дана последовательность $a_{s+1},\ldots,a_k$, которая является (*s+1, k*)-пирамидой. Добавим новый элемент x и поставим его на *s*-тую позицию в последовательности, т.е. пирамида всегда будет расширяться влево. Если выполняется (*), то полученная последовательность – (*s, k*)-пирамида. Иначе найдутся элементы a_{2s+1},a_{2s} такие, что либо $a_{2s} < a_s$ либо $a_{2s+1} < a_s$.

Пусть имеет место первый случай, второй случай рассматривается аналогично. Поменяем местами элементы a_s и a_{2s}. В результате получим новую последовательность a_s',$a_{s+1},\ldots$, a_{2s}',…, a_k. Повторяем все действия для элемента a_{2s}' и т.д. пока не получим (*s, k*)-пирамиду.

Пример. Добавим в (2, 8)-пирамиду новый элемент x=6.

Условные обозначения: [X] новый элемент

X сравнение с включаемым элементом

 обмен элементов

1	2	3	4	5	6	7	8	
	3	2	6	3	4	5	7	пирамида

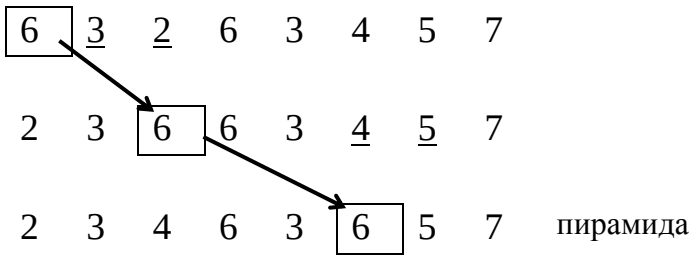


Рисунок 7 Добавление в пирамиду нового элемента

Алгоритм на псевдокоде

Построение (L,R)-пирамиды

$a_{L+1},\ldots,a_R$ – на входе пирамида (L+1,R)
a_L –новый элемент
x:= a_L, i:=L
DO
    j:=2i
    IF (j>R) OD
    IF((j<R) и ($a_{j+1}\le a_j$)) j=j+1 FI
    IF ($x\le a_j$) OD
    $a_i= a_j$
    i:=j
OD
a_i:=x

Величины M и C в процессе построения (L, R)-пирамиды имеют следующие оценки $M_{пир}\le\log (R/L)+2$, $C_{пир}\le 2\log (R/L)$

Пирамидальная сортировка производится в два этапа. Сначала строится пирамида из элементов массива. По свойству (3) правая часть массива является (*n/2+1, n*)-пирамидой. Будем добавлять по одному элементу слева, расширяя пирамиду, пока в неё не войдут все элементы массива. Тогда по свойству (2) первый элемент последовательности – минимальный.

Произведём двустороннее усечение: уберём элементы a_1,a_n. По свойству (1) оставшаяся последовательность является (2, n-1)-пирамидой. Элемент a_1 поставим на последнее место, а элемент a_n добавим к пирамиде $a_2,\ldots,a_{n-1}$ слева. Получим новую (1, *n-1*)-пирамиду. В ней первый элемент является минимальным. Поставим первый элемент пирамиды на позицию *n-1*, а элемент a_{n-1} добавим к пирамиде $a_2,\ldots,a_{n-1}$, и т.д. В результате получим обратно отсортированный массив.

Пример. Отсортировать слово методом пирамидальной сортировки.

Построение (1, 8)-пирамиды

1 2 3 4 5 6 7 8

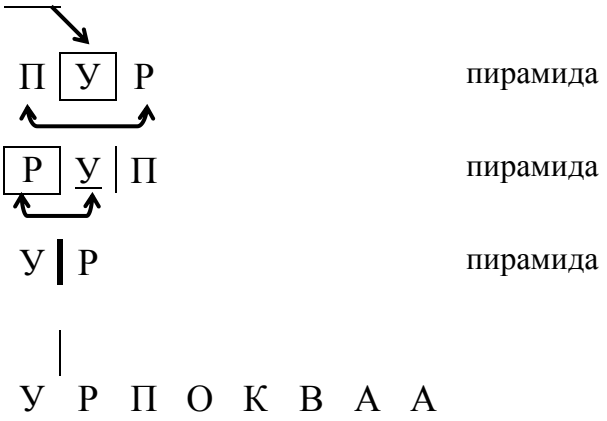


Рисунок 8 Пирамидальная сортировка

Алгоритм на псевдокоде

Пирамидальная сортировка

```
L:=⌊n/2⌋
DO (L>0)
        <Построение (L,n) пирамиды>
        L:=L-1
OD
R:=n
DO (R>1)
        a1↔aR
        R:=R-1
        <Построение (1,R) пирамиды >
OD
```

Общее количество операций сравнений и пересылок для пирамидальной сортировки: $C \le 2n \log n + n + 2$, $M \le n \log n + 6.5n - 4$. Таким образом, $C=O(n \log n)$, $M=O(n \log n)$ при $n \to \infty$.

Отметим некоторые свойства пирамидальной сортировки. Метод пирамидальной сортировки неустойчив и не зависит от исходной отсортированности массива.

4.2 Метод Хоара

Метод Хоара или метод быстрой сортировки заключается в следующем. Возьмём произвольный элемент массива x. Просматривая массив слева, найдём элемент $a_i \ge x$. Просматривая массив справа, найдём $a_j \le x$. Поменяем местами a_i и a_J . Будем продолжать процесс просмотра и обмена, до тех пор пока i не станет больше j. Тогда массив можно разбить на две части: в левой части все элементы не больше x, в правой части массива не меньше x. Затем к каждой части массива применяется тот же алгоритм.

Пример: Отсортировать слово методом быстрой сортировки.

Условные обозначения: Ⓧ ведущий элемент

$\underline{X}$ сравнение с ведущим элементом при просмотре справа

$\overline{X}$ сравнение с ведущим элементом при просмотре слева

| разделение массива на части  обмен элементов

$\dot{X}$ индекс i X индекс j

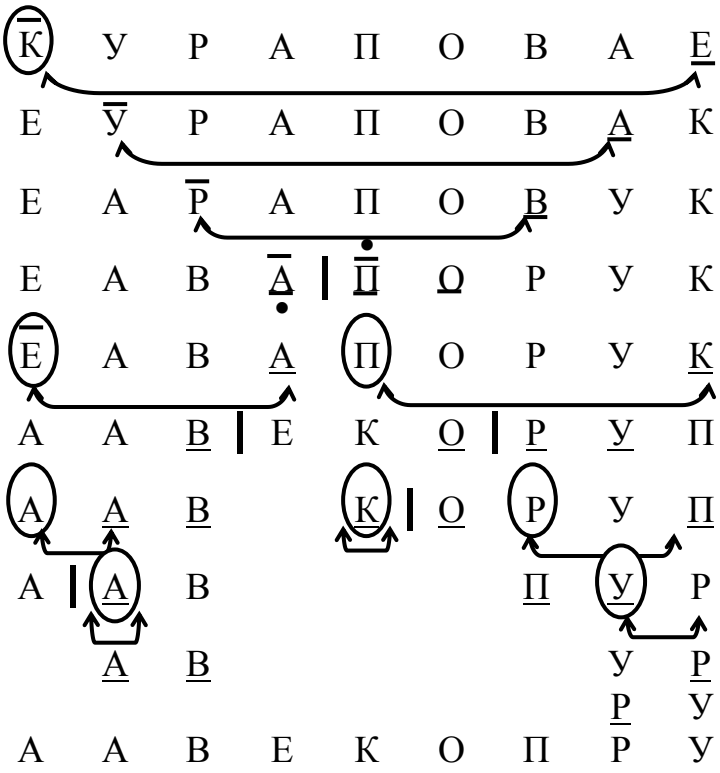


Рисунок 9 Метод Хоара

Алгоритм на псевдокоде

Сортировка части массива с границами (L,R).

Обозначим: L-левую границу рабочей части массива
R-правую границу рабочей части массива

```
x:=a_L, i:=L, j:=R,
DO (i≤j)
        DO (a_i<x) i:=i+1 OD
        DO (a_j>x) j:=j-1 OD
        IF (i<=j)
                a_i↔ a_j,, i:=i+1, j:=j-1
        FI
OD
IF (L<j)
        <Сортировка части массива с границами (L,j)>
FI
IF (i<R)
        <Сортировка части массива с границами (i,R)>
FI
```

Очевидно, трудоёмкость метода существенно зависит от выбора элемента x, который влияет на разделение массива. Максимальные значения M и C для метода быстрой сортировки достигаются при сортировке упорядоченных массивов (в прямом и обратном порядке). Тогда в этом случае в одной части остаётся только один элемент (минимальный или максимальный), а во второй – все остальные элементы. Выражения для M и C

имеют следующий вид

$$M=3(n-1),\ C=(n^2+5n+4)/2$$

Таким образом, в случае упорядоченных массивов трудоёмкость сортировки имеет квадратичный порядок.

Элемент a_m называется *медианой* для элементов $a_L \ldots a_R$, если количество элементов меньших a_m равно количеству элементов больших a_m с точностью до одного элемента (если количество элементов нечётно). В примере буква К- медиана для КУРАПОВАЕ.

Минимальная трудоемкость метода Хоара достигается в случае, когда на каждом шаге алгоритма в качестве ведущего элемента выбирается медиана массива. Количество сравнений в этом случае $C=(n+1)\log(n+1)-(n+1)$. Количество пересылок зависит от положения элементов, но не может быть больше одного обмена на два сравнения. Поэтому количество пересылок – величина того же порядка, что и число сравнений. Асимптотические оценки для средних значений M и C имеют следующий вид

$$C=O(n \log n),\ M=O(n \log n) \text{ при } n \to \infty.$$

Метод Хоара неустойчив.

4.3 Проблема глубины рекурсии.

В теле подпрограммы доступны все объекты, описанные в основной программе, в том числе и имя самой подпрограммы. Это позволяет внутри тела подпрограммы осуществлять её вызов. Процедуры и функции, организующие вызовы «самих себя» называются *рекурсивными*. Рекурсия широко используется в программирование, потому что многие математические алгоритмы имеют рекурсивную природу.

В качестве примера приведём известный алгоритм вычисления факториала неотрицательного целого числа:

$0!=1$

$1!=1$

$n!=(n-1)!*n$

```
function fact (n:word):longint;
    begin
        if (n=0) or (n=1) then fact:=1
        else fact:=fact(n-1)*n;
    end;
```

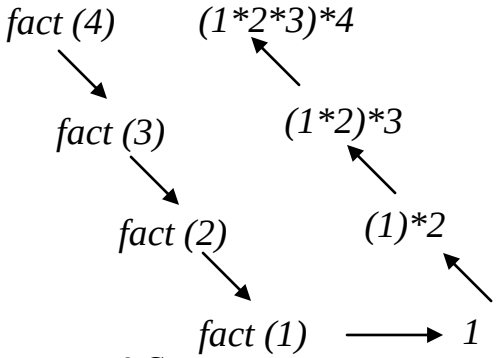


Рисунок 10 Схема вызовов при вычислении 4!

Рекурсивное оформление программы более компактно, наглядно и эффективно. Но существует опасность переполнения стека. Каждый вызов подпрограммы требует специально отведённой области памяти, называемой *фреймом*. В ней хранятся фактические параметры, адреса возврата, локальные переменные и регистры УП.

Фрейм

Практический параметр
Адрес возврата
Регистры из программы
Локальные переменные

Рисунок 11 Структура фрейма

При выходе из программы эта память освобождается. Но если подпрограмма вызывает другую подпрограмму или саму себя, то в дополнение к существующему фрейму создаётся новый, т.е. *n* вложенных вызовов требуют выделения *n* фреймов в памяти.

Рассмотренный алгоритм Хоара может потребовать *n* вложенных вызовов (*n* – размер массива), т.е. глубина рекурсии достигает *n*. Это большой недостаток предложенного алгоритма. Попробуем уменьшить глубину рекурсии до log *n*. В рассмотренном алгоритме производится 2 рекурсивных вызова. Но один из них можно заменить простой итерацией, т.е. для одной части массива будем применять рекурсию, а для другого – простую итерацию. Чтобы уменьшить глубину рекурсии нужно делать рекурсивный вызов для меньшей по размеру части массива. Тогда в худшем случае, когда размеры правой и левой частей будут одинаковые, максимальная глубина рекурсии будет не больше log *n*. Например, для массива из 1 млн. элементов понадобиться одновременно менее 20 фреймов в памяти. Запишем новую версию алгоритма:

Алгоритм на псевдокоде

Сортировка части массива (L,R)

```
DO (есть хотя бы 2 элемента, т.е. L<R)
        <разделение> (как в 1 версии)
        IF (левая часть длиннее правой, т.е.j-L>R-i)
Сортировка части массива (i,R)
        R:=j
        Else
                Сортировка части массива (L,j)
                L:=i;
        FI
OD
```

4.4 Контрольные вопросы

1. Дайте определение пирамиды.
2. Назовите основные свойства пирамиды
3. Какова сложность пирамидальной сортировки?
4. Сформулируйте основную идею метода Хоара.
5. Какова сложность метода Хоара?
6. Как зависит метод Хоара от начальной отсортированности массива?

5. РАБОТА С ЛИНЕЙНЫМИ СПИСКАМИ

5.1 Указатели. Основные операции с указателями

Каждый элемент данных, хранящихся в памяти компьютера, имеет свой адрес. Адреса могут находиться в специальных переменных, называемых указателями. Мы будем рассматривать типизированные указатели, которые могут хранить адреса только объектов определенного типа.

Пусть указатели p и q содержат адрес объекта x некоторого типа tData. Графически будем изображать это следующим образом:

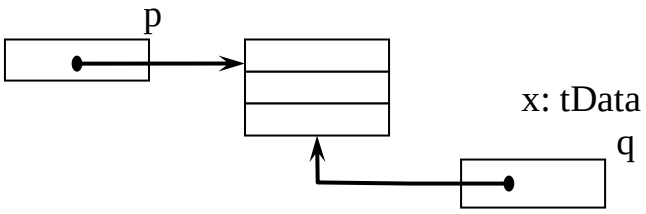


Рисунок 12 Равенство указателей

Стрелка начинается в указателе и указывает на объект в целом, а не на отдельную компоненту, поэтому указатели p и q равны. Заметим, что обычно адрес объекта совпадает с адресом его первой компоненты. Можно обращаться к переменной по имени, а можно по адресу. При обращении по адресу не нужно знать имени переменной.

Основные операции с указателями приведены в следующей таблице.

Таблица 2 Основные операции с указателями

Операция	*Псевдокод*	*Комментарии*
1. Присваивание	p:=q	Указателю p присвоить значение указателя q
2. Сравнение	p=q, p≠q	Сравнение значений указателей p и q
3. Получение адреса	p:=@x	Указателю p присвоить адрес переменной x
4.Доступ по адресу	*p=y *p=*q	Переменной по адресу p присвоить значение переменной по адресу q
5. Доступ к отдельной компоненте	p→comp:=x	Компоненте comp переменной по адресу p присвоить значение переменной x
6. Отсутствие адреса	NIL	Значение указателя p не равно никакому адресу

5.2 Основные операции с линейными списками

Списком называется последовательность однотипных элементов, связанных между собой указателями. Будем считать, что элементы списка имеют тип tLE, указатели на элементы списка имеют тип pLE.

X: tLE p: pLE

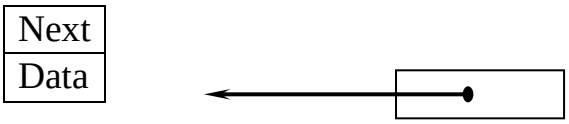


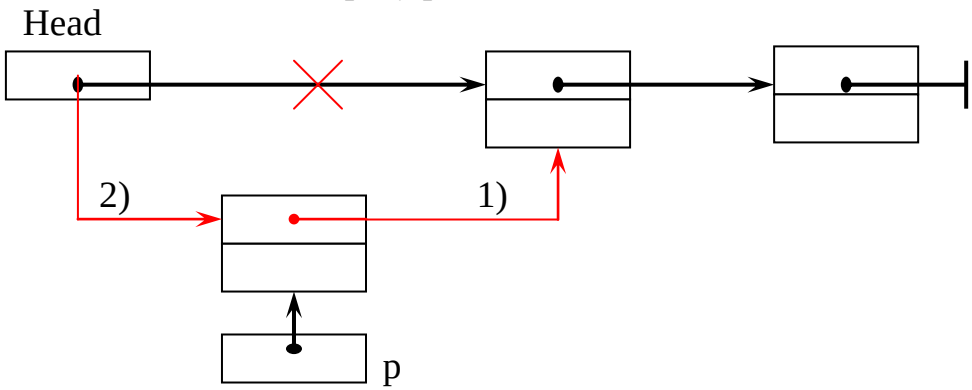
Рисунок 13 Указатель на элемент списка

Поле Next является указателем на элемент списка и может занимать произвольное место в структуре элемента. Однако если оно является первым элементом структуры, то его адрес совпадает с адресом элемента списка, и это позволяет оптимизировать многие операции со списками. Поле Data содержит информацию, которая будет учитываться при сортировке.

Рассмотрим два вида списков: *стек* и *очередь*. *Стек* характеризуется тем, что новый элемент добавляется в начало последовательности, а удаляться может только первый элемент списка. При добавлении в *очередь* новый элемент ставится в конец списка, удаляется первый элемент последовательности.

Рассмотрим основные операции со стеком и очередью. Для работы со стеком необходимо иметь указатель на начало списка. Обозначим его Head. При работе с очередью требуется дополнительный указатель на конец очереди. Обозначим его Tail. Иногда при работе с очередью удобно объединять указатели Head и Tail в виде полей некоторой переменной Queue.

Добавление элемента по адресу p в стек.



1) p→Next:=Head
2) Head:=p

Рисунок 14 Добавление в стек

Удаление из стека или очереди (при условии, что список не пуст, т.е. Head≠NIL*)*

1) p:=Head
2) Head:=p→Next
<освободить память по адресу p>

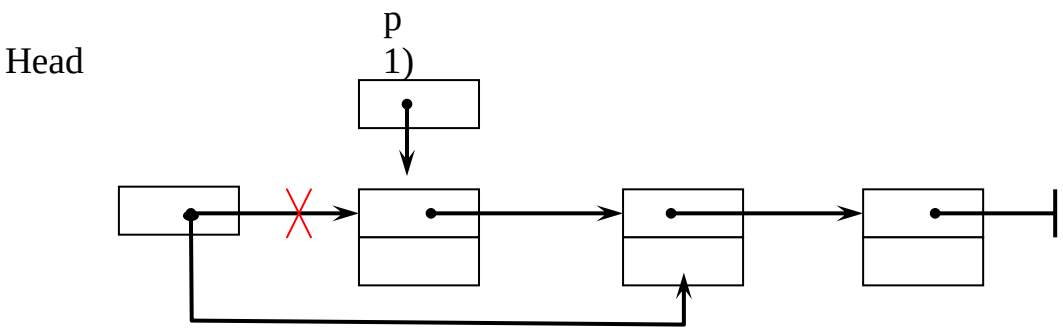


Рисунок 15 Удаление из стека

Добавление элемента по адресу p в очередь.

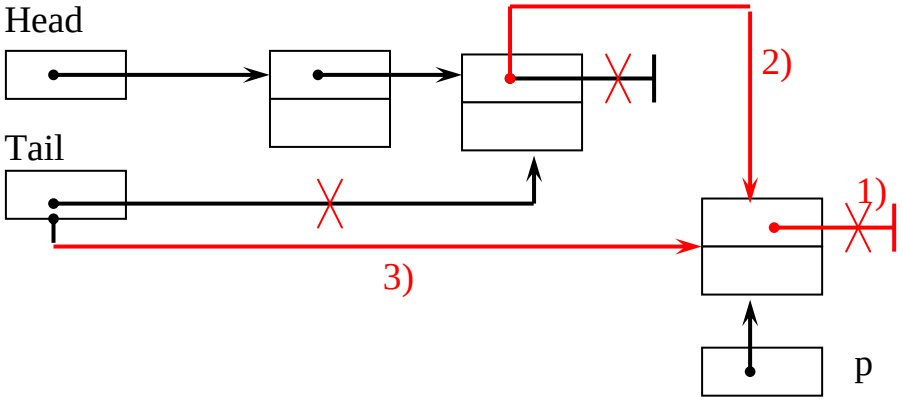


Рисунок 16 Добавление в очередь

1) p→Next:=NIL
2) IF (Head≠NIL) Tail→Next:=p
   ELSE Head:=p
   FI
3) Tail:=p

Операцию добавления элемента в очередь можно оптимизировать в случае, если поле Next является первой компонентой элемента очереди и его адрес совпадает с адресом всего элемента. Зададим пустую очередь следующим образом:

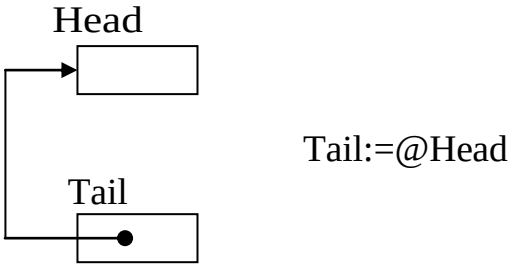


Рисунок 17 Структура очереди

Эту операцию назовем инициализацией очереди. Тогда добавление элемента в очередь будет происходить в два раза быстрее:

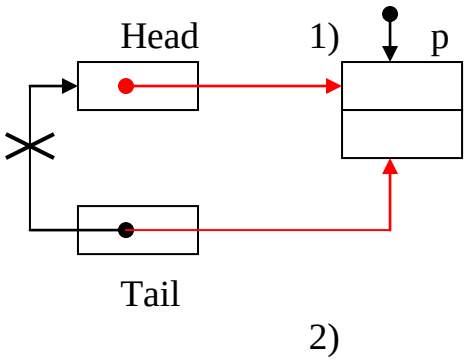


Рисунок 18 Добавление в очередь

1) Tail->Next:=p
2) Tail:=p

Перемещение элемента из начала списка List в конец очереди Queue.

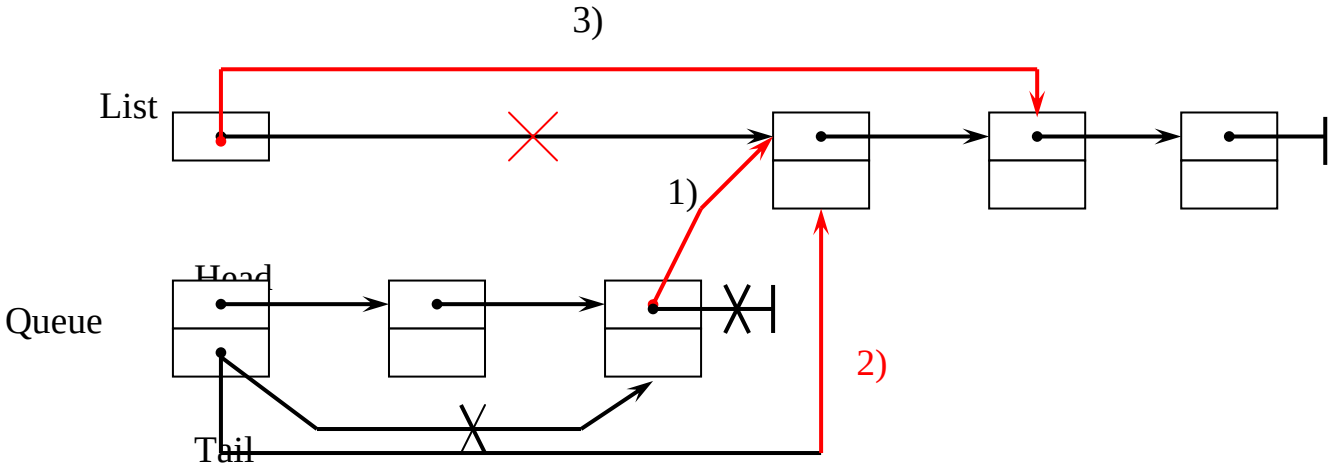


Рисунок 19 Перемещение элемента

1) Queue.Tail→Next:=List
2) Queue.Tail:=List
3) List:=List→Next

6. МЕТОДЫ СОРТИРОВКИ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

6.1 Метод прямого слияния

В основе метода прямого слияния лежит операция слияния серий. *p-серией* называется упорядоченная последовательность из *p* элементов.

Пусть имеются две упорядоченные серии *a* и *b* длины *q* и *r* соответственно. Необходимо получить упорядоченную последовательность *c*, которая состоит из элементов серий *a* и *b*. Сначала сравниваем первые

элементы последовательностей *a* и *b*. Минимальный элемент перемещаем в последовательность *c*. Повторяем действия до тех пор, пока одна из последовательностей *a* и *b* не станет пустой, оставшиеся элементы из другой последовательности переносим в последовательность *c*. В результате получим (*q*+*r*)-серию.

Пример. Слить две серии a=(1, 4, 5, 6′) и b=(2, 3, 6″, 7, 8)

Условные обозначения | операция сравнения первых элементов списков.

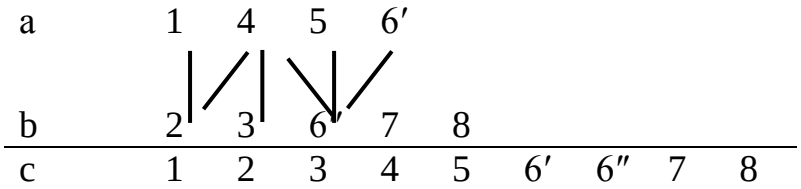


Рисунок 20 Слияние серий

Алгоритм на псевдокоде

Слияние q – серии из списка a с r – серией из списка b,
запись результата в очередь c
Слияние (a, q, b, r, c)

```
DO (q ≠ 0 и r ≠ 0)
        IF (a → Data ≤ b → Data)
                <Переместить элемент из списка a в очередь c>
                q:=q-1
        ELSE
                <Переместить элемент из списка b в очередь c>
                r:=r-1
        FI
OD
DO (q > 0)
        <переместить элемент из списка a в очередь c>
        q:=q-1
OD
DO (r > 0)
        <Переместить элемент из списка b в очередь c>
        r:=r-1
OD
```

Для алгоритма слияния серий с длинами q и r необходимое количество сравнений и перемещений оценивается следующим образом

$$\min(q, r) \le C \le q+r-1,\ M=q+r$$

Пусть длина списка S равна степени двойки, т.е. 2^k, для некоторого натурального *k*. Разобьем последовательность S на два списка *a* и *b*, записывая поочередно элементы S в списки *a* и *b*. Сливаем списки a и b с образованием двойных серий, то есть одиночные элементы сливаются в упорядоченные пары, которые записываются попеременно в очереди c_0 и c_1.

Переписываем очередь c_0 в список a, очередь c_1 – в список b. Вновь сливаем a и b с образованием серий длины 4 и т. д. На каждом итерации размер серий увеличивается вдвое. Сортировка заканчивается, когда длина серии превысит общее количество элементов в обоих списках. Если длина списка S не является степенью двойки, то некоторые серии в процессе сортировки могут быть короче.

Пример. Отсортировать слово методом прямого слияния.

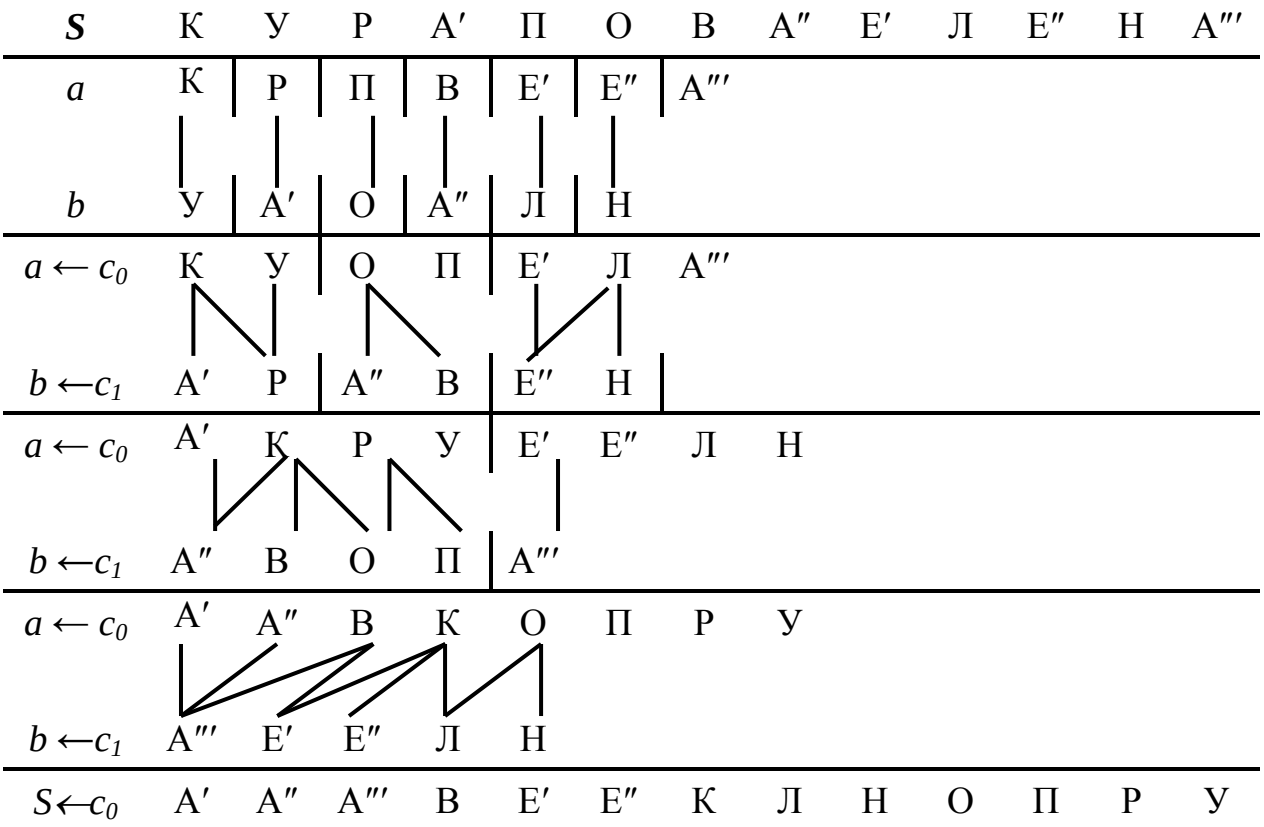


Рисунок 21 Метод прямого слияния

Схематически начальное расщепление последовательности S на списки a и b можно изобразить следующим образом. Ниже приведен алгоритм расщепление на псевдокоде при условии, что список S не пуст.

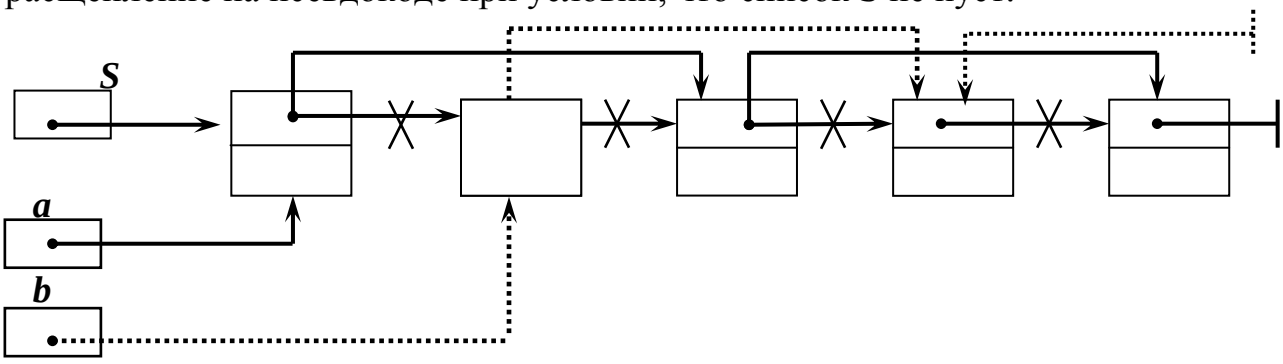


Рисунок 22 Начальное расщепление

Расщепление (S, a, b, n)

Обозначим

n - количество элементов в S

k, p - рабочие указатели

a:=S, b:=S → Next, n:=1
k:=a, p:=b
DO (p ≠ NIL)
n:=n+1
k → next:=p → next
k:=p
p:=p → next
OD

Алгоритм на псевдокоде

Обозначим n – количество элементов в S
a, b – рабочие списки
$c=(c_0, c_1)$ – массив из двух очередей
p – предполагаемый размер серии
q – фактический размер серии в списке a
r – фактический размер серии в списке b
m – текущее количество элементов в списках a и b
i – номер активной очереди

<Расщепление (S, a, b, n)>
p:= 1
DO (p < n)
<инициализация очередей c_0, c_1>
i:=0, m:=n
DO (m > 0)
IF (m ≥ p) q:=p ELSE q:=m FI
m:= m – q
IF (m ≥ p) r:=p ELSE r:=m FI
m:= m – r
<слияние(a, q, b, r, c_i)>
i:=1–i
OD
a:=c_0.Head, b:=c_1.Head
p:=2p
OD
c_0.Tail → next:=NIL
S:=c_0.Head

При инициализации очереди обнуляются указатели, указывающие на начало и на конец очереди, т.е. очередь становится пустой.

Трудоёмкость метода прямого слияния определяется сложностью операции слияния серий. На каждой итерации происходит ровно n перемещений элементов списка и не более *n* сравнений. Как нетрудно видеть,

количество итераций равно $\lceil \log n \rceil$. Тогда

$$C < n\lceil \log n \rceil,\ M = n\lceil \log n \rceil + n.$$

Дополнительные n перемещений происходят во время начального расщепления исходного списка. Асимптотические оценки для M и C имеют следующий вид

$$C = O(n \log n),\ M = O(n \log n) \text{ при } n \to \infty.$$

Метод обеспечивает устойчивую сортировку. При реализации для массивов, метод требует наличия второго вспомогательного массива, равного по размеру исходному массиву. При реализации со списками дополнительной памяти не требуется.

6.2 Цифровая сортировка

Другим методом сортировки последовательностей является цифровая сортировка. Пусть дана последовательность из S чисел, представленных в m – ичной системе счисления. Каждое число состоит из L цифр $d_1 d_2 \ldots d_L$, $0 \le d_i \le m - 1$, i=1..L. Сначала числа из списка S распределяются по m очередям, причём номер очереди определяется последней цифрой каждого числа. Затем полученные очереди соединяются в список, для которого все действия повторяются, но распределение по очередям производится в соответствии со следующей цифрой и т.д.

Пример. Отсортировать последовательность 31 03′ 20 02 03″ 33 30 21 методом цифровой сортировки. Числа представлены в четверичной системе счисления.

S :	3<u>1</u>	0<u>3</u>′	2<u>0</u>	0<u>2</u>	0<u>3</u>″	3<u>3</u>	3<u>0</u>	2<u>1</u>
Q_0 :	20	30						
Q_1 :	31	21						
Q_2 :	02							
Q_3 :	03′	03″	33					
S :	<u>2</u>0	<u>3</u>0	<u>3</u>1	<u>2</u>1	<u>0</u>2	<u>0</u>3′	<u>0</u>3″	<u>3</u>3
Q_0 :	02	03′	03″					
Q_1 :								
Q_2 :	20	21						
Q_3 :	30	31	33					
S :	02	03′	03″	20	21	30	31	33

Рисунок 23 Цифровая сортировка

Алгоритм на псевдокоде

Цифровая сортировка

```
DO (j=L, L–1, … , 1)
    <инициализация очередей Q>
    <расстановка элементов из списка S в очереди Q по j – ой цифре >
    <соединение очередей Q в список S >
OD
```

Цифровой метод может успешно использоваться не только для

сортировки чисел, но и для сортировки любой информации, представленной в памяти компьютера. Необходимо лишь рассматривать каждый байт ключа сортировки как цифру, принимающую значения от 0 до 255. Тогда для сортировки потребуется m=256 очередей. Для выделения каждого байта ключа сортировки можно использовать массив Digit, наложенный в памяти компьютера на поле элемента последовательности, по которому происходит сортировка. Приведем более детальный алгоритм цифровой сортировки.

Алгоритм на псевдокоде

Цифровая сортировка

```
DO (j=L, L-1, … 1)
    DO (i=0, 1, … 255)
        Qi.Tail:=@ Qi.Head
    OD
    p:=S
    DO (p ≠ NIL)
        d:=p → Digit[j]
        Qd.Tail → Next:=p
        Qd.Tail:=p
        p:=p → Next
    OD
    p:=@ S
    DO (i=0, 1, … 255)
        IF (Qi.Tail ≠ @ Qi.Head)
            p → Next:=Qi.Head
            p:=Qi.Tail
        FI
    OD
    p → Next:=NIL
    OD
```

Для цифровой сортировки M<const L(m+n). При фиксированных m и L M=O(n) при $n \to \infty$, что значительно быстрее остальных рассмотренных методов. Однако если длина чисел L велика, то метод может проигрывать обычным методам сортировки. Кроме того, Метод применим только, если задача сортировки сводится к задаче упорядочивания чисел, что не всегда возможно.

Метод обеспечивает устойчивую сортировку. Чтобы изменить направление сортировки на обратное, очереди нужно присоединять в обратном порядке.

6.3 Контрольные вопросы

1. В чем смысл операции слияния серий?
2. Какова сложность метода прямого слияния?
3. В чем основная идея метода цифровой сортировки?

4. Является ли метод цифровой сортировки устойчивым?
5. Можно ли применять методы сортировки последовательностей для упорядочивания массивов?

7. ДВОИЧНЫЙ ПОИСК В УПОРЯДОЧЕННОМ МАССИВЕ

7.1 Алгоритм двоичного поиска

Алгоритм двоичного поиска в упорядоченном массиве сводится к следующему. Берём средний элемент отсортированного массива и сравниваем с ключом X. Возможны три варианта:

1. Выбранный элемент равен X. Поиск завершён.
2. Выбранный элемент меньше X. Продолжаем поиск в правой половине массива.
3. Выбранный элемент больше X. Продолжаем поиск в левой половине массива.

Далее рассмотрим две версии реализации двоичного поиска в упорядоченном массиве.

Версия 1

Пример. Найти в отсортированном массиве элемент с ключом X = б.

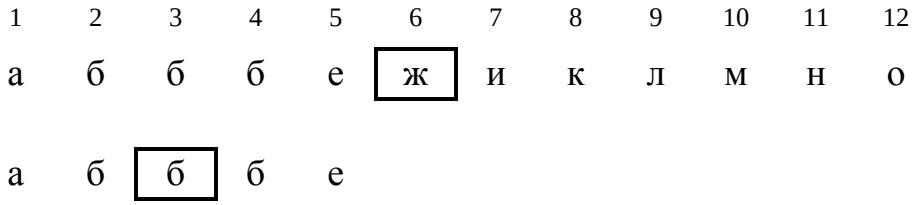


Рисунок 24 Первая версия поиска

Алгоритм на псевдокоде

Поиск элемента с ключом X

Обозначим

L, R – правая и левая границы рабочей части массива.

Найден – логическая переменная, в которой будем отмечать факт успешного завершения поиска.

```
L: = 1, R: =n, Найден: = нет
    DO (L≤R)
        m: = ⌊(L + R) / 2⌋
        IF (a_m=X) Найден: =да OD FI
        IF (a_m < X) L: = m+1
              ELSE R: = m-1
        FI
    OD
```

Отметим недостаток этой версии алгоритма. На каждой итерации совершается два сравнения, но можно обойтись одним сравнением. Если в

массиве несколько элементов с одинаковым ключом, то эта версия находит один из них. Чтобы найти все элементы с одинаковыми ключами, необходимо просмотреть массив влево и вправо от найденного элемента.

Версия 2

Пример. Найти в отсортированном массиве элемент с ключом X = б.

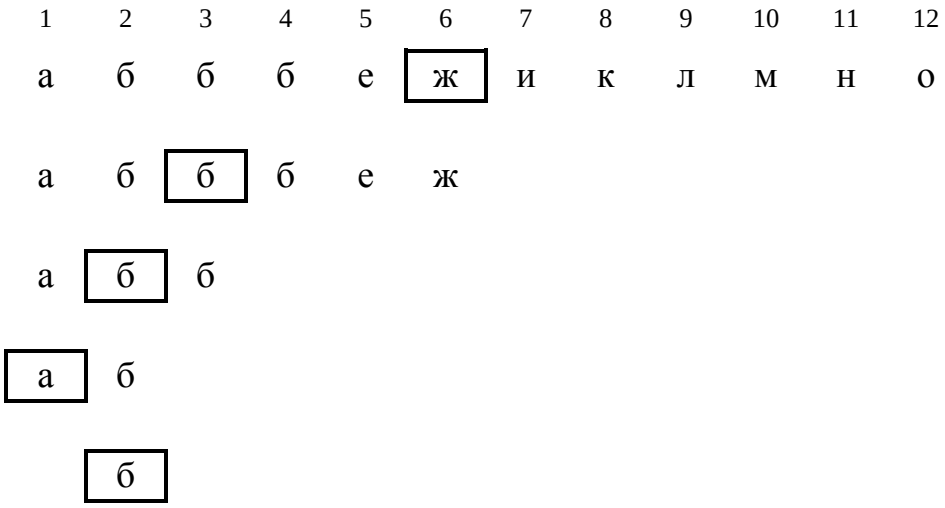


Рисунок 25 Вторая версия поиска

Алгоритм на псевдокоде

Поиск элемента с ключом X

```
L: = 1, R: =n
DO (L<R)
        m: = ⌊(L + R) / 2⌋
        IF (a_m < X) L: = m+1
                ELSE R: = m
        FI
OD
IF (a_r=X) Найден: =да
        ELSE Найден: =нет
FI
```

Нетрудно заметить, что после выхода из цикла L = R. Если в массиве несколько элементов с одинаковым ключом, то эта версия алгоритма находит самый левый из них. Для поиска остальных элементов с заданным ключом требуется просмотреть массив только в одном направлении – вправо от найденного элемента.

Дадим верхнюю оценку трудоёмкости алгоритма двоичного поиска. На каждой итерации поиска необходимо два сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем $\lceil \log_2 n \rceil$. Таким образом, трудоёмкость двоичного поиска в обоих случаях

$$C=O(\log n),\ n \to \infty.$$

7.2 Контрольные вопросы

1. Сформулируйте основную идею быстрого поиска.
2. Чем отличаются две версии быстрого поиска друг от друга?
3. Сравните средние трудоемкости различных версий быстрого поиска.

8. СОРТИРОВКА ДАННЫХ С ПРОИЗВОЛЬНОЙ СТРУКТУРОЙ

8.1 Сравнение данных произвольной структуры

Основная проблема, возникающая при сортировке данных произвольной структуры — неопределенность операции сравнения. Если исходный массив A заполнен числами, то в качестве операции сравнения могут быть использованы стандартные операции сравнения. Если структура сортируемых данных не соответствует простым встроенным типам языка, то необходимо переопределить операции сравнения с помощью логических функций. Например, пусть массив A — телефонный справочник, каждый элемент которого является записью с полями Name (фамилия абонента) и Phone (номер телефона):

A:

Иванов	*Петров*	....	*Егоров*
223455	*452185*		*454455*

Рисунок 26 Список абонентов

Если необходимо отсортировать телефонный справочник по фамилиям абонентов, то логическая функция Less (меньше) может выглядеть следующим образом:

```
function less ( x,y: <тип записи>): boolean;
    begin
        less:=x.Name <y.Name;
    end;
```

Такой подход позволяет путем изменения функции Less учитывать любые сложные условия упорядочивания массива элементов произвольной структуры. Например, если необходимо упорядочить телефонный справочник по номерам телефонов дополнительно по фамилиям абонентов, то функцию Less можно записать так:

```
function less ( x,y: <тип записи>): boolean;
    begin

        IF (x.phone<y.phone)
            less:=true
        ELSEIF (x.phone>y.phone )
            less:=false
```

```
        ELSE less:=(x.name<y.name)
        FI
        FI
end;
```

Кроме того, нужно переопределить операцию сравнения и при организации поиска элементов в отсортированном массиве. Изменение направления упорядочивания массива достигается путем замены операций сравнения на противоположные, т. е. в самой функции меняем «<» на «>».Операция пересылки не требует переопределения и выполняется путем побитового копирования.

8.2 Сортировка по множеству ключей. Индексация

Пусть рассмотренный выше телефонный справочник необходимо использовать для быстрого поочередного поиска абонентов или по номеру телефона, или по фамилии абонента. Пересортировка массива то по одному, то по другому ключу требует значительных затрат времени. Для эффективного решения подобной задачи используется прием, называемый индексацией, или созданием индексного массива.

Вначале построения индексный массив B заполняется целыми числами от 1 до n. Затем производится сортировка, но при условии, что в операциях сравнения элементы массива A индексируются через массив B. Перестановки делаются только в массиве B. Тогда при доступе к элементам массива A через индексный массив B A[B[i]] можно работать с массивом A как с упорядоченным по возрастанию (например, производить быстрый поиск элементов), в то время как сами элементы A физически не переставляются.

Пример. Рассмотрим создание индексного массива, который упорядочивает массив целых чисел A=(7,1,6,3,2,8,5).Чтобы упорядочивать массив A по возрастанию, пронумеруем его элементы. Введем новый массив B и запишем в него номера массива A в последовательности, соответствующей условию упорядочиванию (по возрастанию). Получим следующий индексный массив B=(2,5,4,7,3,1,6)

Алгоритм на псевдокоде

(на примере пузырьковой сортировки)

```
B:=(1,2,…,n)
DO (i=1,2,…,n-1)
        DO(j=n,n-1,…,i+1)
                IF(a[b_j]< a[b_{j-1}]) b_i↔b_{j-1} FI
        OD
OD
```

Отметим ряд положительных свойств индексации.

1. Индексация дает возможность построения нескольких различных индексов, которые можно использовать по мере необходимости.
2. Исключается копирование больших массивов данных (физический

массив остаётся на месте, а индексы занимают мало места).

3. Имеется возможность фильтрации данных. Фильтрация означает, что при работе с базами данных используются не все элементы, а только те, которые отвечают определённым условиям. В индекс включаются физические номера тех элементов, которые удовлетворяют условию фильтра.

8.3 Индексация через массив указателей

Индексация через массив указателей отличается от обычной индексации тем, что вместо номеров элементов в индексный массив записываются адреса сортируемых элементов. К достоинствам такой индексации можно отнести то, что исходные данные могут располагаться не только в массиве, а произвольным образом в динамической памяти.

8.4 Контрольные вопросы

1. Что такое индексный массив?
2. Назовите основные достоинства индексный массивов.
3. Что такое фильтрация данных?
4. Каким образом строится индексный массив?

9. ХЭШИРОВАНИЕ И ПОИСК

9.1 Понятие хэш-функции

Все рассмотренные ранее алгоритмы были связаны с задачей поиска, которую можно сформулировать следующим образом: задано множество ключей, необходимо так организовать это множество ключей, чтобы поиск элемента с заданным ключом потребовал как можно меньше затрат времени. Поскольку доступ к элементу осуществляется через его адрес в памяти, то задача сводится к определению подходящего отображения H множества ключей K во множество адресов элементов A.

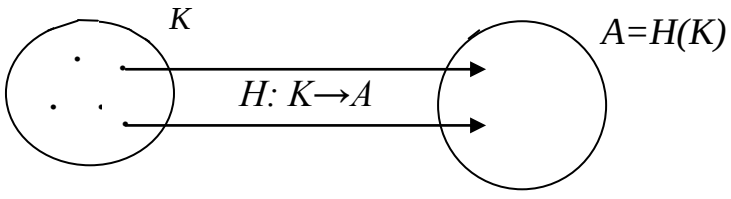


Рисунок 27 Отображение $H: K \rightarrow A$

В предыдущих главах такое отображение получалось путем различного размещения ключей (в отсортированном порядке, в виде деревьев поиска), т.е. каждому ключу соответствовал свой адрес в памяти. Теперь рассмотрим задачу построения отображения $H: K \rightarrow A$ при условии, что количество

всевозможных ключей существенно больше количества адресов. Будем обозначать это так: $|K| >> |A|$. Например, в качестве множества ключей можно взять всевозможные фамилии студентов до 15 букв ($|K|= 32^{15}$), а в качестве множества адресов – 100 мест в аудитории ($|A|=100$). Функция $H: K \rightarrow A$, определенная на конечном множестве K, называется *хеш-функцией*, если $|K| >> |A|$. Таким образом, хеш-функция допускает, что нескольким ключам может соответствовать один адрес. *Хеширование* – один из способов поиска элементов по ключу, при этом над ключом k производят некоторые арифметические действия и получают значение функции $h=H(k)$, которое указывает адрес, где хранится ключ k и связанная с ним информация. Если найдутся ключи $k_i \neq k_j$, для которых $H(k_i)=H(k_j)$, т.е. несколько ключей отображаются в один адрес, то такая ситуация называется *коллизией (конфликтом)*.

Если данные организованы как обычный массив, то H – отображение ключей в индексы массива. Процесс поиска происходит следующим образом:

1) для ключа k вычисляем индекс $h=H(k)$
2) проверяем, действительно ли h определяет в массиве T элемент с ключом k, т. е. верно ли соотношение T[$H(k)$].data = k. Если равенство верно, то элемент найден. Если неверно, то возникла коллизия.

Для эффективной реализации поиска с помощью хеш-функций необходимо определить какого вида функцию H нужно использовать и что делать в случае коллизии (конфликта). Хорошая хеш-функция должна удовлетворять двум условиям:

1) её вычисление должно быть очень быстрым
2) она должна минимизировать число коллизий, т.е. как можно равномернее распределять ключи по всему диапазону индекса.

Для разрешения коллизий нужно использовать какой-нибудь способ, указывающий альтернативное местоположение искомого элемента. Выбор хеш-функции и выбор метода разрешения коллизий – два независимых решения.

Функции, дающие неповторяющиеся значения, достаточно редки даже в случае довольно большой таблицы. Например, знаменитый парадокс дней рождений утверждает, что если в комнате присутствует не менее 23 человек, имеется хороший шанс, что у двух из них совпадет день рождения. Т.е., если мы выбираем функцию, отображающую 23 ключа в таблицу из 365 элементов, то с вероятностью 0,4927 все ключи попадут в разные места.

Теоретически невозможно так определить хеш-функцию, чтобы она создавала случайные данные из неслучайных реальных ключей. Но на практике нетрудно сделать достаточно хорошую имитацию случайности, используя простые арифметические действия.

Будем предполагать, что хеш-функция имеет не более m различных значений: $0 \leq H(k) < m$ для любого значения ключа. Например, если ключи десятичные, то возможен следующий способ. Пусть m=1000, в качестве $H(k)$ можно взять три цифры из середины двадцатизначного произведения $k \bullet k$.

Этот метод «середины квадрата», казалось бы, должен давать довольно равномерное распределение между 000 и 999. но на практике такой метод не плох, если ключи не содержат много левых или правых нулей подряд.

Исследования выявили хорошую работу двух типов хеш-функций: один основан на умножении, другой на делении.

1) *метод деления* особенно прост: используется остаток от деления на m $H(K)=K$ mod m. При этом желательно m брать простым числом.
2) *метод умножения* $H(K)=2^m(A\cdot K \bmod w)$, где A и w взаимно простые числа.

Далее будем использовать функцию $H(k)=ORD(k) \bmod m$, где ORD(k) – порядковый номер ключа, m – размер массива (таблицы), причем m рекомендуется брать простым числом.

Если ключ поиска является строкой, то для вычисления ее хеш-номера будем рассматривать её как большое целое число, записанное в 256-ичной системе счисления (каждый символ строки является цифрой), т.е.

$$H(S_1S_2S_3\ldots S_t)=(S_1\cdot 256^{t-1}+S_2\cdot 256^{t-2}+\ldots+S_{t-1}\ 256+S_t) \bmod m\ .$$

Используя свойства остатка от деления можно легко вычислить подобные выражения: $(a+b)\cdot \bmod m=(a \bmod m + b \bmod m) \bmod m$. Например, (47+56) mod 10 = (7+6) mod 10 = 3

Алгоритм на псевдокоде

Вычисление хеш-функции для строки S

Обозначим t – длина строки S

```
h:=0
DO (i=1,2,…,t)
      h:=(h·256+Si) mod m
OD
```

9.2 Метод прямого связывания

Рассмотрим метод устранения коллизий путем связывания в список всех элементов с одинаковыми значениями хеш-функции, при этом необходимо m списков. Включение элемента в хэш-таблицу осуществляется в два действия*:*

1) вычисление $i=H(k)$
2) добавление элемента k в конец i-того списка

Поиск элемента также требует два действия:

1) вычисление $i=H(k)$
2) последовательный просмотр i-того списка.

Пример. Составить хеш-таблицу для строки КУРАПОВА ЕЛЕНА. Будем использовать номера символов в алфавитном порядке. Пусть m=5,

$H(k)=ORD$ (k mod 5)

Вычислим значения хэш-функции для символов строки

H(К)=11 mod 5=1

H(У)=20 mod 5=0
H(Р)=17 mod 5=2
H(А)=1 mod 5=1
H(П)=16 mod 5=1
H(О)=15 mod 5=0
H(В)=3 mod 5=3
H(Е)=6 mod 5=1
H(Л)=12 mod 5=2
H(Н)=14 mod 5=4
Объединим символы с одинаковыми хеш-номерами в один список

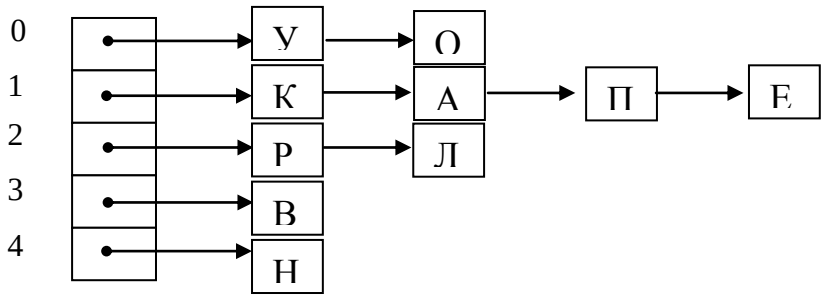


Рисунок 28 Хеш-таблица, построенная методом прямого связывания

Оценим трудоемкость поиска в хеш-таблице, построенной методом прямого связывания. Пусть n – количество элементов данных, m – размер хеш-таблицы. Если все ключи равновероятны и равномерно распределены по хеш-таблице, то средняя длина списка будет $\frac{n}{m}$. При поиске в среднем нужно просмотреть половину списка. Поэтому $C_{ср}=\frac{n}{2m}$. Если $n<m$, то $C_{ср}<2$, т. е. в большинстве случаев достаточно одного сравнения. Объем дополнительной памяти определяется объемом памяти, необходимой для хранения $(m+n)$ указателей. Известно, что трудоемкость поиска с помощью двоичного дерева: $C_{ср}=\log n$, объем дополнительной памяти – $2n$ указателей. Метод прямого связывания становится более эффективным, чем дерево поиска, когда

$$\frac{n}{2m} < \log n,\ m > \frac{n}{2\log n}$$

Если n=1000, то при m>50 (m=53) метод прямого связывания более эффективен, чем дерево поиска, причем экономия памяти составит около 4 Кбайт. Можно сэкономить еще больше памяти, если отказаться от списков и размещать данные в самой хеш-таблице.

9.3 Метод открытой адресации

Рассмотрим метод открытой адресации, который применяется для разрешения коллизий при поиске с использованием хеш-функций. Суть метода заключается в последовательном просмотре различных элементов таблицы, пока не будет найден искомый ключ k или свободная позиция.

Очевидно, необходимо иметь правило, по которому каждый ключ k определяет последовательность проб, т.е. последовательность позиций в таблице, которые нужно просматривать при вставке или поиске ключа k. Если мы произвели пробы и обнаружили свободную позицию, то ключа k нет в таблице. Таким образом, коллизия устраняется путем вычисления последовательности *вторичных хеш-функций:*

$h_0=h(x)$

$h_1=h(x)+g(1)$ (mod m)

$h_2=h(x)+g(2)$ (mod m)

$h_i=h(x)+g(i)$ (mod m)

Самое простое правило для просмотра – просматривать подряд все следующие элементы таблицы. Этот прием называется *методом линейных проб*, при этом $g(i)=i$, $i=1,2,\ldots,m-1$. Недостаток данного метода – плохое рассеивание ключей (ключи группируются вокруг первичных ключей, которые были вычислены без конфликта), хотя и используется вся хеш-таблица.

Если в качестве вспомогательных функций использовать квадратичные, т.е. $g(i)=i^2$, $i=1,2,\ldots,m-1$, то такой способ просмотра элементов называется *методом квадратичных проб*. Достоинство этого метода – хорошее рассеивание ключей, хотя хеш-таблица используется не полностью.

Утверждение. *Если m – простое число, то при квадратичных пробах просматривается по крайней мере половина хеш-таблицы.*

Доказательство. Пусть i-ая и j-ая пробы, $i<j$, приводят к одному значению h, т.е. $h_i=h_j$. Тогда $i^2 \bmod m=j^2 \bmod m$

$$(j^2 - i^2) \bmod m=0$$
$$(j+i)(j-i) \bmod m=0$$
$$(j+i)(j-i)=km$$
$$i+j=km/(j-i)$$

Если m – простое число, то $k/(j-i)$ – целое число больше нуля. В худшем случае $k/(j-i)=1$, тогда $i+j=m$ и $j>m/2$. (Если m – не простое число, то $k/(j-i)$ не обязательно должно быть целым).

На практике этот недостаток не столь существенен, т.к. $m/2$ вторичных попыток при разрешении конфликтов встречаются очень редко, главным образом в тех случаях, когда таблица почти заполнена.

Итак, нам нужно вычислять

$$h_0=h(x)$$
$$h_i=(h_0+i^2) \bmod m,\ i>0$$

Вычисление h_i требует одного умножения и деления. Покажем, как можно избавиться от этих операций. Произведем несколько первых шагов при вычислении h_i.

$h_1=h_0+1$

$h_2=h_0+4=h_0+1+3=h_1+3 \pmod m$

$h_3=h_0+9=h_0+4+5=h_2+5 \pmod m$

…

Нетрудно видеть, что возникает рекуррентное соотношение:

d_0=1, h_0=$h(x)$

h_{i+1}=h_i+d_i (mod m)

d_{i+1}=d_i+2

Поскольку h_i<m, d_i<m, то можно избавиться от деления, заменив его вычитанием h=h-m (см. алгоритм).

Алгоритм на псевдокоде

Поиск и вставка элемента с ключом x

Пусть хеш-таблица является массивом A=(a_0, a_1, … , a_{m-1}), сначала заполненный нулями. Пусть $x\neq0$.

```
h:=x mod m
d:=1
DO
    IF (a_h=x) элемент найден OD
    IF (a_h=0) ячейка пуста, a_h:=x OD
    IF (d≥m) переполнение таблицы OD
    h:=h+d
    IF (h≥m) h:=h-m FI
    d:=d+2
OD
```

Заметим, что если нужен только поиск, то необходимо исключить операцию a_h:=x.

Пример построения хеш-таблицы методом квадратичных проб (m=11) для строки ВЛАДИМИР ПУТИН. Номера символов данной строки приведены в таблице.

Таблица 3 Номера символов строки

В	Л	А	Д	И	М	И	Р	П	У	Т	И	Н
3	12	1	5	9	13	9	17	16	20	19	9	14

Для каждого символа вычисляем его хеш-номер (или последовательность хеш-номеров, если потребуется) и в соответствии с вычисленным номером заносим символ в хеш-таблицу.

В: h_0=3 mod 11=3

Л: h_0=12 mod 11=1

А: h_0=1 mod 11=1

h_1=1+1=2

Д: h_0=5

И: h_0=9

М: h_0=13 mod 11=2

h_1=2+1=3

h_2=3+3=6

Р: h_0=17 mod 11=6

h_1=6+1=7

П: h_0=16 mod 11=5

h_1=5+1=6

h_2=6+3=9
h_3=9+5=3
h_4=3+7=10

У: h_0=20 mod 11=9
h_1=9+1=10
h_2=10+3=13 mod 11=2
h_3=2+5=7
h_4=7+7=14 mod 11=3
h_5=3+9=12 mod 11=1

Просмотр элементов хеш-таблицы на этом заканчивается несмотря на то, что в таблице еще имеются незаполненные ячейки, поскольку следующее значение d уже не будет строго меньше m=11. Таким образом, для данной строки не удается построить хеш-таблицу с m=11. Заполненная часть хеш-таблицы выглядит следующим образом

0	1	2	3	4	5	6	7	8	9	10
	Л	А	В		Д	М	Р		И	П

Рисунок 29 Использование квадратичных проб

9.4 Контрольные вопросы

1. Что такое хэш-функция?
2. Что такое коллизия?
3. Как осуществляется поиск с помощью хэш-таблицы?
4. Какие способы построения хэш-таблиц Вы знаете?

ПРАВИЛА ВЫПОЛНЕНИЯ ЗАДАНИЙ НА ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Программы выполняются на языках высокого уровня (Паскаль, Си). По согласованию с преподавателем допускается работа в средах Delphi, Builder C++, Visual C++. Для защиты задания студенту необходимо представить

- Исходные тексты программ;
- Исполняемые файлы;
- Результаты работы программы

Защита задания включает в себя следующие разделы

- Формулировку задания;
- Описание основных методов и алгоритмов, используемых при выполнении задания;
- Анализ результаты работы программы.

Тестирование программ должно проводиться для различных случаев: упорядоченный массив (прямой и обратный порядок), случайный массив.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 1

Тема: Методы сортировки массивов с квадратичной трудоемкостью.

Цель работы: Освоить методы сортировки массивов с квадратичной трудоемкостью.

Порядок выполнения работы:

1. Разработать процедуры сортировки массива целых чисел методом прямого выбора, методом пузырьковой сортировки и методом шейкерной сортировки (язык программирования Паскаль или Си).
2. Правильность сортировки проверить путем подсчета контрольной суммы и числа серий в массиве.
3. Во время сортировки предусмотреть подсчет количества пересылок и сравнений (М и С), сравнить их с теоретическими оценками.
4. Составить таблицу следующего вида (данные получить экспериментально) для n= 10, 50, 100, 200. (n – количество элементов в массиве)

метод	М для упорядоченного массива	С для упорядоченного массива	М для случайного массива	С для случайного массива
Прямой выбор				
Пузырьковая				
Шейкерная				

5. Проанализировать полученные результаты. (Какой из методов самый быстрый? Самый медленный? Как сложность зависит от начальной отсортированности?)

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 2

Тема: Быстрые методы сортировки массивов

Цель работы: Освоить быстрые методы сортировки массивов

Порядок выполнения работы:

1. Разработать процедуры сортировки массива целых чисел методом Шелла, методом пирамидальной сортировки и методом Хоара (язык программирования Паскаль или Си).
2. Правильность сортировки проверить путем подсчета контрольной суммы и числа серий в массиве.
3. Во время сортировки предусмотреть подсчет количества пересылок и сравнений (М и С), сравнить их с теоретическими оценками.

4. Составить таблицу следующего вида (данные получить экспериментально) для n= 10, 50, 100, 200. (n – количество элементов в массиве)

метод	М для упорядоченного массива	С для упорядоченного массива	М для случайного массива	С для случайного массива
Метод Шелла				
Пирамидальная сортировка				
Метод Хоара				

5. Проанализировать полученные результаты. (Какой из методов самый быстрый? Самый медленный? Как сложность зависит от начальной отсортированности?)
6. Сравните трудоемкости методов быстрой сортировки и трудоемкости методов с квадратичной трудоемкости (использовать результаты практического занятия 1)

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 3

Тема: Работа с линейными списками

Цель работы: Освоить основные операции с линейными сприсками

Порядок выполнения работы:

1. Разработать процедуры для работы со списками (язык программирования Паскаль или Си):
   - заполнение стека и очереди возрастающими числами;
   - заполнение стека и очереди убывающими числами;
   - заполнение стека и очереди случайными числами;
   - печать элементов списка;
   - подсчет контрольной суммы элементов списка;
   - подсчет количества серий в списке.
2. Применить разработанные процедуры для n= 20 (n – количество элементов в списке).
3. Проанализировать полученные результаты. (Какой порядок элементов в стеке? В очереди? Зависит ли количество серий от вида списка?)

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 4

Тема: Быстрые методы сортировки последовательностей

Цель работы: Освоить быстрые методы сортировки последовательностей

Порядок выполнения работы:

4. Разработать процедуры сортировки последовательности целых чисел методом прямого слияния и методом цифровой сортировки (язык программирования Паскаль или Си).
5. Во время сортировки предусмотреть подсчет количества пересылок элементов в очередь и сравнений (М и С), сравнить их с теоретическими оценками.
6. Составить таблицу следующего вида (данные получить экспериментально) для n= 10, 50, 100, 200. (n – количество элементов в массиве)

метод	М для упорядоченной последовательности	С для упорядоченной последовательности	М для случайной последовательности	С для случайной последовательности
Прямое слияние				
цифорвая				

7. Проанализировать полученные результаты. (Какой из методов самый быстрый? Самый медленный? Как сложность зависит от начальной отсортированности?)

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 5

Тема: Индексация и быстрый поиск

Цель работы: Освоить методы построения индексных массивов и быстрого поиска в массиве.

Порядок выполнения работы:

Написать программу «Телефонный справочник», которая обрабатывает данные об абонентах телефонной станции. Каждый абонент имеет имя, адрес, телефонный номер. В программе описать массив абонентов (назовем его справочник). В справочнике должно быть не менее 20 элементов, которые заполняются либо программно, либо считываются из файла.

С помощью индексов и фильтров (номер задания выбирается по последней цифре шифра)

0. упорядочить справочник по имени по возрастанию
1. упорядочить справочник по телефонному номеру по возрастанию
2. упорядочить справочник по адресу по убыванию
3. выбрать тех абонентов, которые имеют номер в заданном диапазоне
4. упорядочить справочник по имени и телефонному номеру по возрастанию
5. выбрать тех абонентов, которые имеют имя в заданном диапазоне
6. выбрать абонентов, которые имеют имя и адрес в заданном диапазоне
7. упорядочить справочник по телефонному номеру по убыванию
8. упорядочить справочник по адресу по убыванию

9. выбрать абонентов, которые имеют адрес в заданном диапазоне

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 6

Тема: Хэширование и поиск

Цель работы: Освоить методы построения хэш-таблиц и поиска с помощью хэш-таблиц.

Порядок выполнения работы:

1. Построить хэш-таблицу методом линейных проб для слов заданного текста. Текст находится в некотором файле (примерно 200 слов). Экспериментально определить минимально необходимый объем хэш-таблицы и число коллизий при построении.
2. Построить хэш-таблицу методом квадратичных проб для слов заданного текста. Файл с текстом должен быть тот же, что и п.1. Экспериментально определить минимально необходимый объем хэш-таблицы и число коллизий при построении.
3. Заполнить следующую таблицу полученными данными. Проанализировать результаты. (Какой метод требует большего объема памяти? Для какого из методов меньшее число коллизий?).

метод	Объем хэш-таблицы	Число коллизий
Линейные пробы		
Квадратичные пробы		

КОНТРОЛЬНАЯ РАБОТА

ПРАВИЛА ВЫПОЛНЕНИЯ И ОФОРМЛЕНИЯ КОНТРОЛЬНОЙ РАБОТЫ

При выполнении контрольной работы необходимо строго придерживаться указанных ниже правил. Работы, выполненные без соблюдения этих правил, не засчитываются и возвращаются студенту для переработки.

1. Контрольная работа состоит из восьми заданий, которые одинаковы для всех студентов, однако входные данные выбираются индивидуально.
2. В работу должны быть включены все задачи, указанные в задании, строго по своему варианту. Контрольные работы, содержащие не все задачи или задачи не своего варианта, не засчитываются.
3. Решения задач необходимо располагать в порядке номеров, указанных в заданиях, сохраняя номера задач. Перед решением каждой задачи необходимо выписать полностью ее условие. Работа выполняется аккуратно в тетради и присылается в деканат для проверки. В качестве образцов оформления обязательно использовать примеры из

теоретической части пособия, для наглядности желательно использовать вспомогательные цвета (черный, зеленый).

4. После получения прорецензированной работы, как недопущенной, так и допущенной к защите, студент должен исправить все отмеченные рецензентом ошибки и недочеты и выполнить все рекомендации. Если работа не допущена к защите, то после исправления указанных рецензентом ошибок работу следует прислать для повторной проверки в короткий срок. При высылаемых исправлениях должны обязательно находиться прорецензированная работа и рецензия к ней. Без выполненной контрольной работы студент к экзамену не допускается.

ПРАВИЛА ВЫБОРА ВАРИАНТА

Задания для контрольной работы одинаковы для всех студентов. Начальные данные выбираются индивидуально в зависимости от задания в контрольной работе.

1. Используя в качестве массива набор из 8 букв своих фамилии, имени, отчества, определить на каждом шаге в методе прямого выбора номера перемещаемых элементов.
2. Используя в качестве массива набор из 8 букв своих фамилии, имени, отчества, определить на каждом шаге в методе шейкерной сортировки левую и правую границы сортируемой части массива (L и R).
3. Используя в качестве массива набор из 8 букв своих фамилии, имени, отчества провести сортировку массива методом Шелла. Последовательность шагов h_1=1, h_2=2.
4. Используя в качестве массива набор из 10 букв своих фамилии, имени, отчества, построить пирамиду и отсортировать массив.
5. Провести сортировку последовательности из 15 букв своих фамилии, имени, отчества методом прямого слияния.
6. Составить произвольную последовательность из 12 трехзначных чисел в четверичной системе счисления и отсортировать ее с помощью цифровой сортировки.
7. Провести быстрый поиск (2 версии) буквы «Е» (русс.) в массиве из 15 букв своих фамилии, имени, отчества.
8. Построить хэш-таблицу методом квадратичных проб для всех букв своих фамилии, имени, отчества.

ВОПРОСЫ К ЭКЗАМЕНУ

Для допуска к экзамену необходимо выполнить и защитить все задания на практические занятия и контрольную работу.

ПЕРЕЧЕНЬ ВОПРОСОВ НА ЭКЗАМЕН (СЕССИЯ 5)

1. Основные структуры данных
2. Постановка задачи сортировки
3. Алгоритм сортировки методом прямого выбора
4. Классы сложности алгоритмов
5. Алгоритмы пузырьковой и шейкерной сортировок
6. Алгоритм сортировки методом прямого включения
7. Алгоритм сортировки методом Шелла
8. Алгоритм двоичного поиска в упорядоченном массиве
9. Алгоритм сортировки данных с произвольной структурой
10. Алгоритмы индексации данных
11. Алгоритм пирамидальной сортировки
12. Теорема о сложности сортировки
13. Алгоритм быстрой сортировки Хоара
14. Динамические структуры данных. Адреса и указатели
15. Динамически распределяемая память
16. Алгоритм индексации через массив указателей
17. Алгоритмы работы с линейными списками
18. Задача сортировки последовательностей
19. Алгоритм прямого слияния
20. Алгоритм цифровой сортировки
21. Хеш-функции и их приложение к задаче поиска
22. Хеширование методом прямого связывания
23. Алгоритм открытой адресации: линейные и квадратичные пробы

ЗАДАЧИ

1. Привести пример массивов, в которых имеется 2 и 3 серии.
2. В массиве (А,Л,Р,П,Д,К,Я,З) определить медиану.
3. Являются ли данные последовательности пирамидами?
   $a_1=2, a_2=6, a_3=5, a_4=7, a_5=2, a_6=2, a_7=12, a_8=10$
   $a_3=2, a_4=6, a_5=5, a_6=7, a_7=2, a_8=2, a_9=12, a_{10}=10$
4. Какова глубина рекурсии в методе Хоара при сортировке данного массива? (1,2,3,4,5,6,7,8)
5. Методом цифровой сортировки отсортировать массив
   (41, 73, 90, 52, 93, 53, 31, 41)
6. Построить индексный массив, сортирующий массив
   (71, 93, 30, 152, 53, 23, 39, 101)
7. Построить хэш-таблицу методом прямой адресации, используя все буквы фамилии, имени, отчества.
8. Построить хэш-таблицу методом линейных проб, используя все буквы фамилии, имени, отчества.
9. Отсортировать методом пузырьковой сортировки 8 букв своих фамилии, имени, отчества.
10. Привести пример массивов, в которых имеется 4 и 1 серии.

11. В массиве (Р,Л,Р,П,Л,К,Ф,З) определить медиану.
12. Являются ли данные последовательности пирамидами?

$a_1=3, a_2=7, a_3=9, a_4=17, a_5=2, a_6=2, a_7=2, a_8=1$

$a_3=2, a_4=6, a_5=5, a_6=17, a_7=22, a_8=32, a_9=52, a_{10}=100$

13. Какова глубина рекурсии в методе Хоара при сортировке данного массива? (8,7,6,5,4,3,2,1)
14. Методом цифровой сортировки отсортировать массив

(71, 43, 190, 82, 3, 23, 1, 4)

15. Построить индексный массив, сортирующий массив

(11, 63, 38, 15, 513, 3, 79, 10)

16. Определить последовательность шагов в методе Шелла для массива с 20 элементами.
17. Построить индексный массив, сортирующий массив в обратном порядке (71, 93, 30, 152, 53, 23, 39, 101)
18. Построить хэш-таблицу методом прямой адресации, используя все буквы фамилии, имени, отчества.
19. Построить хэш-таблицу методом линейных проб, используя все буквы фамилии, имени, отчества.
20. Отсортировать методом пузырьковой сортировки 8 букв своих фамилии, имени, отчества.
21. Привести пример массивов, в которых имеется 2 и 5 серий.
22. В массиве (Л,Д,Ж,Э,Л,К,С,З) определить медиану.
23. Являются ли данные последовательности пирамидами?

$a_1=3, a_2=7, a_3=9, a_4=17, a_5=2, a_6=2, a_7=2, a_8=1$

$a_3=2, a_4=6, a_5=5, a_6=17, a_7=22, a_8=32, a_9=52, a_{10}=100$

24. Построить хэш-таблицу методом прямой адресации, используя все буквы фамилии, имени, отчества.
25. Построить хэш-таблицу методом линейных проб, используя все буквы фамилии, имени, отчества.

ПРИЛОЖЕНИЕ А

Псевдокод для записи алгоритмов

Для записи алгоритма будем использовать специальный язык – псевдокод. Алгоритм на псевдокоде записывается на естественном языке с использованием двух конструкций: ветвления и повтора. В круглых скобках будем писать комментарии. В треугольных скобках будем описывать действия, алгоритм выполнения которых не требует детализации, например, <обнулить массив>.

: = Операция присваивания значений.

⟷ Операция обмена значениями.

Конструкции ветвления.

1. IF (условие) Если выполняется условие,
<действие> то выполнить действие
FI FI указывает на конец этих действий.

2. IF (условие)
<действия 1>
ELSE <действия 2> Действия 2 выполняются,
FI если неверно условие.

3. IF (условие1)
<действия1>
ELSEIF (условие2) Действия 2 выполняются,
<действия2> если неверно условие1 и верно условие 2
…FI

Конструкции повтора.

1. Цикл с предусловием.
DO (условие) Действия повторяются
<действия> пока условие истинно.
OD OD указывает на конец цикла.

2. Цикл с постусловием.
DO <действия>
OD (условие выполнения)

3. Цикл с параметром.
DO (i=1, 2, ... n) Действия выполняются для значений
<действия> параметра из списка
OD

4. Бесконечный цикл.
DO
<действия>
OD

5. Принудительный выход из цикла.
DO
...IF (условие) OD Если условие истинно, то выйти из цикла.
OD