УДК 681.31 (031)

Л - 38

Лойко В.И. Структуры и алгоритмы обработки данных. Учебное пособие для вузов.- Краснодар: КубГАУ. 2004. - 261 с., ил.

Учебное пособие разработано на основе лекций по курсу "Структуры и алгоритмы обработки данных в ЭВМ", преподаваемых автором студентам различных специальностей. В теоретической части пособия изложены основные положения теории алгоритмов и структур данных для персональных ЭВМ. Главное внимание в пособии уделено оперативным структурам.

Рассмотрены простые типы данных и такие структуры, как статические, полустатические и динамические. В динамических структурах данных выделены линейные и нелинейные связные списки.

Изложены и проанализированы основные алгоритмы сортировки и поиска данных в различных структурах.

В практической части учебного пособия приведены методические указания к лабораторным работам и курсовому проектированию.

Учебное пособие предназначено для студентов специальности 351400 – «Прикладная информатика (по областям)» и других экономических специальностей, изучающих информатику и информационные технологии.

Ил. 64. Библиогр.: 6 назв.

Рецензенты: проф., д-р техн. наук В. И. Ключко
(зав. кафедрой ВТ и АСУ, КубГТУ)
проф., д-р экон. наук М.И. Семенов
(зав. кафедрой АИТ, КубГАУ)

© Кубанский государственный
аграрный университет

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ** ..... **8**

**ЧАСТЬ 1. ВВЕДЕНИЕ В ТЕОРИЮ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ИХ ОБРАБОТКИ** ..... **10**

**1.ТИПЫ ДАННЫХ** ..... **11**

1.1 Целый тип - INTEGER ..... 12
1.2 Вещественный тип - REAL ..... 13
1.3 Логический тип - BOOLEAN ..... 14
1.4 Символьный тип - CHAR ..... 14
1.5 Указательный тип - POINTER ..... 15
1.6 Стандартные типы пользователя ..... 16
*1.6.1 Перечисляемый* ..... *16*
*1.6.2 Диапазонный или интервальный* ..... *17*

**2. СТАТИЧЕСКИЕ И ПОЛУСТАТИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ** ..... **19**

2.1 Уровни представления данных ..... 20
2.2 Классификация структур данных ..... 21
2.3 Статические структуры данных ..... 22
*2.3.1 Векторы* ..... *22*
*2.3.2 Массивы* ..... *23*
*2.3.3 Записи* ..... *23*
*2.3.4 Таблицы* ..... *26*
2.4 Полустатические структуры данных ..... 27
*2.4.1 Стеки* ..... *28*
*2.4.2 Очередь* ..... *30*
*2.4.3 Дек* ..... *39*

**3. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ** ..... **41**

3.1 Связные списки ..... 42
*3.1.1 Односвязные списки* ..... *42*
*3.1.2 Кольцевой односвязный список* ..... *43*
*3.1.3 Двусвязный список* ..... *44*
*3.1.4 Кольцевой двусвязный список* ..... *45*
3.2 Реализация стеков с помощью односвязных списков ..... 46

3.3 Организация операций GetNode, FreeNode и утилизация освободившихся элементов ........ 49
*3.3.1 Операция GetNode* ........ *50*
*3.3.2 Операция FreeNode* ........ *51*
*3.3.3 Утилизация освободившихся элементов в многосвязных списках* ........ *51*
3.4 Односвязный список, как самостоятельная структура данных ........ 51
*3.4.1 Вставка и извлечение элементов из списка* ........ *53*
*3.4.2 Примеры типичных операций над списками* ........ *55*
*3.4.3 Элементы заголовков в списках* ........ *58*
3.5 Нелинейные связанные структуры ........ 59
**4. РЕКУРСИВНЫЕ СТРУКТУРЫ ДАННЫХ ........ 63**
4.1 Деревья ........ 63
*4.1.1 Представление деревьев* ........ *65*
4.2 Бинарные деревья ........ 65
*4.2.1 Сведение m-арного дерева к бинарному* ........ *67*
*4.2.2 Основные операции с деревьями* ........ *69*
*4.2.3 Алгоритм создания дерева бинарного поиска* ........ *70*
*4.2.4 Прохождение бинарных деревьев* ........ *72*
**5. ПОИСК ........ 75**
5.1. Последовательный поиск ........ 76
5.2. Индексно-последовательный поиск ........ 78
5.3. Эффективность последовательного поиска ........ 80
5.4. Эффективность индексно-последовательного поиска ........ 81
5.5. Методы оптимизации поиска ........ 82
*5.5.1. Переупорядочивание таблицы поиска путем перестановки найденного элемента в начало списка* ........ *83*
*5.5.2. Метод транспозиции* ........ *84*
*5.5.3. Дерево оптимального поиска* ........ *85*
5.6 Бинарный поиск (метод деления пополам) ........ 87
5.7. Поиск по бинарному дереву ........ 90
5.8 Поиск со вставкой (с включением) ........ 91
5.9 Поиск с удалением ........ 92
**6. СОРТИРОВКА ........ 97**
6.1. Сортировка методом прямого включения ........ 98
6.2 Сортировка методом прямого выбора ........ 101
6.3. Сортировка с помощью прямого обмена (пузырьковая сортировка) ........ 102
1.4. Улучшенные методы сортировки ........ 105

*6.4.1. Быстрая сортировка (Quick Sort) .................................. 105*
*6.4.2 Сортировка Шелла (сортировка с уменьшающимся шагом) ........................................................................................ 106*

**7. ПРЕОБРАЗОВАНИЕ КЛЮЧЕЙ (РАССТАНОВКА) .......................... 110**

7.1. Выбор функции преобразования ..................................... 110
7.2. Алгоритм ........................................................................ 112

**ЧАСТЬ 2. ПРАКТИКУМ ПО СРУКТУРАМ И АЛГОРИТМАМ ОБРАБОТКИ ДАННЫХ В ЭВМ ....................................................... 116**

**МЕТОДИЧЕСКОЕ РУКОВОДСТВО К ЛАБОРАТОРНЫМ РАБОТАМ ............................................................................................ 117**

Организационно-методические указания ............................ 117
**Лабораторная работа № 1.** "ПОЛУСТАТИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ" ............................................................................. 119
Краткая теория ....................................................................... 119
Алгоритм ................................................................................ 121
Задания .................................................................................... 123
**Лабораторная работа № 2.** "СПИСКОВЫЕ СТРУКТУРЫ ДАННЫХ" ..................................................................................... 124
Краткая теория ....................................................................... 124
*Линейные однонаправленные списки ..................................... 126*
Алгоритм ................................................................................ 127
*Удаление элемента из начала односвязного списка ................. 128*
*Вставка элемента в список ...................................................... 129*
*Удаление элемента из односвязного списка ............................ 130*
Задания .................................................................................... 131
**Лабораторная работа № 3.** "СПИСКОВЫЕ СТРУКТУРЫ ДАННЫХ" ..................................................................................... 132
Краткая теория ....................................................................... 132
Алгоритм ................................................................................ 133
*Вставка элемента в кольцевой список ..................................... 133*
*Удаление элемента из кольцевого списка ................................ 134*
Задания .................................................................................... 135
**Лабораторная работа № 4.** "МОДЕЛЬ МАССОВОГО ОБСЛУЖИВАНИЯ" ................................................................... 137
Краткая теория ....................................................................... 137
Алгоритм ................................................................................ 139
*Процедура прибавления элемента в начало списка. ................. 139*
*Процедура удаления из начала списка. ..................................... 139*
*Процедура прибавления элемента в список. ............................. 139*
*Процедура удаления из списка ................................................. 140*

Задания ........................................................................ 140
**Лабораторная работа № 5.** "БИНАРНЫЕ ДЕРЕВЬЯ(основные процедуры)" ........................................................................ 142
Краткая теория ........................................................................ 142
Алгоритм ........................................................................ 145
*Процедура создания бинарного дерева ........................................ 145*
*Процедуры "обхода" дерева ........................................ 147*
*Процедура поиска по бинарному дереву ........................................ 148*
*Процедура включения элемента в дерево ........................................ 149*
*Процедура удаления элемента из бинарного дерева ........................................ 151*
Задания ........................................................................ 153
**Лабораторная работа № 6 .** "СОРТИРОВКА МЕТОДОМ ПРЯМОГО ВКЛЮЧЕНИЯ" ........................................................................ 156
Краткая теория ........................................................................ 156
Алгоритм ........................................................................ 158
Задания ........................................................................ 159
**Лабораторная работа № 7.** "СОРТИРОВКА МЕТОДОМ ПРЯМОГО ВЫБОРА" ........................................................................ 161
Краткая теория ........................................................................ 161
Алгоритм ........................................................................ 165
Задания ........................................................................ 167
**Лабораторная работа № 8."**СОРТИРОВКА С ПОМОЩЬЮ ПРЯМОГО ОБМЕНА" ........................................................................ 168
Краткая теория ........................................................................ 168
Алгоритм ........................................................................ 170
*Алгоритм пузырькового метода ........................................ 170*
*Алгоритм метода Quiksort ........................................ 170*
Задания ........................................................................ 171
**Лабораторная работа № 9.** "СОРТИРОВКА С ПОМОЩЬЮ ДЕРЕВА" ........................................................................ 174
Краткая теория ........................................................................ 174
Алгоритм ........................................................................ 176
*Создание дерева бинарного поиска : ........................................ 177*
*Обход дерева слева - направо ........................................ 178*
Задания ........................................................................ 179
**Лабораторная работа № 10.** "ИССЛЕДОВАНИЕ МЕТОДОВ ЛИНЕЙНОГО И БИНАРНОГО ПОИСКА" ........................................................................ 182
Краткая теория ........................................................................ 182
Алгоритм ........................................................................ 183
*Линейный поиск ........................................ 183*
*Поиск делением пополам (двоичный поиск). ........................................ 185*
Задания ........................................................................ 188

**Лабораторная работа №11.** "ИССЛЕДОВАНИЕ МЕТОДОВ ОПТИМИЗАЦИИ ПОИСКА " ........ 189
Краткая теория ........ 189
Алгоритм ........ 191
*Переупорядочение путем перестановки в начало списка ........ 191*
*Метод транспозиции ........ 192*
Задания ........ 193
**Лабораторная работа № 12.** "ПОИСК ПО ДЕРЕВУ С ВКЛЮЧЕНИЕМ" ........ 196
Краткая теория ........ 196
Алгоритм ........ 197
Задания ........ 199
**Лабораторная работа № 13.** "ПОИСК ПО ДЕРЕВУ С ИСКЛЮЧЕНИЕМ" ........ 201
Краткая теория ........ 201
Алгоритм ........ 202
Задания ........ 205

**ТЕСТЫ К ЛАБОРАТОРНЫМ РАБОТАМ ........ 207**

**МЕТОДИЧЕСКОЕ РУКОВОДСТВО К КУРСОВОЙ РАБОТЕ ........ 222**

1 Требования к курсовой работе ........ 222
2. Примерный перечень курсовых работ ........ 223
3. Пример выполнения курсовой работы ........ 224
*3.1 Постановка задачи ........ 224*
*3.2 Краткая теория ........ 224*
*3.3 Метод исследования ........ 228*
*3.4 Результаты исследования ........ 229*
*3.5 Контрольный пример ........ 230*
*3.6 Выводы ........ 231*
*3.7 Описание процедур, используемых в программе ........ 232*

**ЗАКЛЮЧЕНИЕ ........ 244**

**ЛИТЕРАТУРА ........ 246**

**ПРИЛОЖЕНИЕ. ТЕСТЫ С ОТВЕТАМИ ........ 247**

# ВВЕДЕНИЕ

Компьютер - это машина, которая обрабатывает информацию. Изучение науки об ЭВМ предполагает изучение того, каким образом эта информация организована внутри ЭВМ, как она обрабатывается и как может быть использована. Следовательно, для изучения предмета студенту особенно важно понять концепции организации информации и работы с ней.

Так как вычислительная техника базируется на изучении информации, то первый возникающий вопрос заключается в том, что такое информация. К сожалению, несмотря на то , что концепция информации является краеугольным камнем всей науки о вычислительной технике, на этот вопрос не может быть дано однозначного ответа. В этом контексте понятие "информация" в вычислительной технике сходно с понятием "точка", "прямая" и "плоскость" в геометрии - все это неопределенные термины, о которых могут быть сделаны некоторые утверждения и выводы, но которые не могут быть объяснены в терминах более элементарных понятий.

Базовой единицей информации является бит, который может принимать два взаимоисключающих значения. Если устройство может находиться более чем в двух состояниях, то тот факт, что оно находится в одном из этих состояний, уже требует нескольких битов информации.

Для представления двух возможных состояний некоторого бита используются двоичные цифры - нуль и единица.

Число битов, необходимых для кодирования символа в конкретной вычислительной машине, называется размером байта, а группа битов в этом числе называется байтом. Размер байта в большинстве ЭВМ равен 8.

Память вычислительной машины представляет собой совокупность битов. в любой момент функционирования в

ЭВМ каждый из битов памяти имеет значение 0 или 1 (сброшен или установлен). Состояние бита называется его значением или содержимым.

Биты в памяти ЭВМ группируются в элементы большего размера, например в байты. В некоторых ЭВМ несколько байтов объединяются в группы, называемые словами. Каждому такому элементу назначается адрес, который представляет собой имя, идентифицирующее конкретный элемент памяти среди аналогичных элементов. Этот адрес обычно числовой. Он называется ячейкой, а содержимое ячейки есть значение битов, которые ее составляют.

Итак, мы видим, что информация в ЭВМ сама по себе не имеет конкретного смысла. С некоторой конкретной битовой комбинацией может быть связано любое смысловое значение. Именно интерпретация битовой комбинации придает ей заданный смысл.

Метод интерпретации битовой информации часто называется типом данных. Каждая ЭВМ имеет свой набор типов данных.

Важно осознавать роль, выполняемую спецификацией типа в языках высокого уровня. Именно посредством подобных объявлений программист указывает на то, каким образом содержимое памяти ЭВМ интерпретируется программой. Эти объявления детерминируют объем памяти, необходимый для размещения отдельных элементов, способ интерпретации этих элементов и другие важные детали. Объявления также сообщают интерпретатору точное значение используемых символов операций.

# ЧАСТЬ 1.
# ВВЕДЕНИЕ В ТЕОРИЮ СТРУКТУР ДАННЫХ И АЛГОРИТМОВ ИХ ОБРАБОТКИ

# 1.ТИПЫ ДАННЫХ

В математике принято классифицировать переменные в соответствие с некоторыми важными характеристиками. Мы различаем вещественные, комплексные и логические переменные ,переменные ,представляющие собой отдельные значения, множества значений или множества множеств. В обработке данных понятие классификации играет такую же, если не большую роль. Мы будем придерживаться того принципа, что любая константа, переменная, выражение или функция относятся к некоторому типу.

Фактически тип характеризует множество значений, которые может принимать некоторая переменная или выражение и которые может формировать функция.

В большинстве языков программирования различают стандартные типы данных и типы, заданные пользователем. К стандартным относят 5 типов:

***a) целый (INTEGER);***
***b) вещественный (REAL) ;***
***c) логический (BOOLEAN);***
***d) символьный (CHAR);***
***e) указательный (POINTER).***

К пользовательским относят 2 типа:
***a) перечисляемый;***
***b) диапазонный.***

Любой тип данных должен быть охарактеризован областью значений и допустимыми операциями над этим типом данных.

## 1.1 Целый тип - INTEGER

Этот тип включает некоторое подмножество целых, размер которого варьируется от машины к машине. Если для представления целых чисел в машине используется n разрядов, причем используется дополнительный код, то допустимые числа должны удовлетворять условию $-2^{n-1} <= x < 2^{n-1}$.

Считается, что все операции над данными этого типа выполняются точно и соответствуют обычным правилам арифметики. Если результат выходит за пределы представимого множества, то вычисления будут прерваны. Такое событие называется переполнением.

Числа делятся на знаковые и беззнаковые. Для каждого из них имеется свой диапазон значений:

a)$(0..2^{n}-1)$ для беззнаковых чисел

b) $(-2^{N-1}..\ 2^{N-1}-1)$ для знаковых.

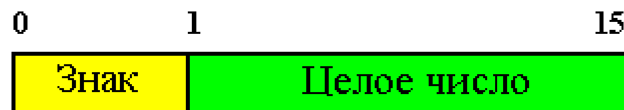



Рис. 1.1

При обработке машиной чисел, используется формат со знаком. Если же машинное слово используется для записи и обработки команд и указателей, то в этом случае используется формат без знака.

Операции над целым типом:

a) Сложение.

b) Вычитание.

c) Умножение.

d) Целочисленное деление.

e) Нахождение остатка по модулю.

f) Нахождение экстремума числа (минимума и максимума)

g) Реляционные операции (операции сравнения) (<,>,<=, >=,=,<>)

Примеры:
A div B = C
A mod B = D
C * B + D = A
7 div 3 = 2
7 mod 3 = 1

Во всех операциях, кроме реляционных, в результате получается целое число.

## 1.2 Вещественный тип - REAL

Вещественные типы образуют ряд подмножеств вещественных чисел, которые представлены в машинных форматах с плавающей точкой. Числа

в формате с плавающей точкой характеризуются целочисленными значениями мантиссы и порядка, которые определяют диапазон изменения

и количество верных знаков в представлении чисел вещественного типа.

$X = +/- M * q^{(+/-P)}$ - полулогарифмическая форма представления числа, показана на рисунке 2.

$937{,}56 = 93756 * 10^{-2} = 0{,}93756 * 10^{3}$

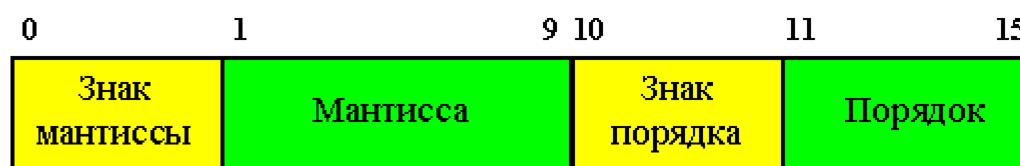



Рис. 1.2

Удвоенная точность необходима для того, чтобы увеличить точность мантиссы.

## 1.3 Логический тип - BOOLEAN

Стандартный логический тип Boolean (размер-1 байт) представляет собой тип данных, любой элемент которого может принимать лишь 2 значения: True и False.

Над логическими элементами данных выполняются логические операции. Основные из них:

a) Отрицание (NOT)

b) Конъюнкция (AND)

c) Дизъюнкция (OR)

**Таблица истинности основных логических функций.**

| A | B | not A | A or B | A and B |
|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |

Рис. 1.3

Логические значения получаются также при реляционных операциях с целыми числами.

## 1.4 Символьный тип - CHAR

Тип CHAR содержит 26 прописных латинских букв и 26 строчных, 10 арабских цифр и некоторое число других графических символов, например, знаки пунктуации.

Подмножества букв и цифр упорядочены и "соприкасаются", т.е.

("A"<= x)&(x <= "Z") - x представляет собой прописную букву

("0"<= x)&(x <= "9") - x представляет собой цифру

Тип CHAR содержит некоторый непечатаемый символ, пробел, его можно

использовать как разделитель.

Операции:

a) Присваивания

b) Сравнения

c) Определения номера данной литеры в системе кодирования. ORD($W_i$)

d) Нахождение литеры по номеру. CHR(i)

e) Вызов следующей литеры. SUCC($W_i$)

f) Вызов предыдущей литеры. PRED($W_i$)

## 1.5 Указательный тип - POINTER

Переменная типа указатель является физическим носителем адреса величины базового типа. Стандартный тип-указатель Pointer дает указатель, не связанный ни с каким конкретным базовым типом. Этот тип совместим с любым другим типом-указателем.

Операции:

a) Присваивания

b) Операции с беззнаковыми целыми числами.

При помощи этих операций можно вычислить адрес данных. В машинном виде эти типы занимают максимально возможную длину.

**Например:**

ABCD:1234 - значение указателя в шестнадцатеричной системе счисления - относительный адрес.

Первое число (ABCD) - адрес сегмента

Второе число (1234) - адрес внутри сегмента.

***Получение абсолютного адреса из относительного:***

Для получения абсолютного адреса необходимо произвести сдвиг адреса сегмента влево, и к полученному числу прибавить адрес внутреннего сегмента.

**Например:**

1) Сдвигаем ABCD на один разряд влево. Получаем ABCD0.
2) Прибавляем 1234. Полученный результат и является абсолютным адресом.

ABCD0
  12 3 4
----------
ACF04 - абсолютный адрес данного числа.

## 1.6 Стандартные типы пользователя

### 1.6.1 Перечисляемый

Перечисляемый тип определяется конечным набором значений, представленных списком идентификаторов в объявлении типа. Значениям из этого набора присваиваются номера в соответствии с той последовательностью, в которой перечислены идентификаторы. Формат
объявления перечисляемого типа таков:

TYPE<имя> = (<список>);
<список>:= <идентификатор>,[<список>]

Если идентификатор указан в списке значений перечисляемого типа, он считается именем константы, определенной в том блоке, где объявлен перечисляемый тип. Порядковые номера значений в объявлении перечисляемого типа определяются их позициями в списке идентификаторов, причем у первой константы в списке порядковый номер равен нулю. К

данным перечисляемого типа относится, например, набор цветов:

TYPE <Цвет> = (Красный, Зеленый, Синий)

Операции те же, что и для символьного типа.

### 1.6.2 Диапазонный или интервальный

В любом порядковом типе можно выделить подмножество значений, определяемое минимальным и максимальным значениями, в которое входят все значения исходного типа, находящиеся в этих границах, включая сами границы. Такое подмножество определяет диапазонный тип. Он задаётся указанием минимального и максимального значений, разделенных двумя точками.

TYPE T=[ MIN..MAX ]
TYPE <Час>=[1..60]

Минимальное значение при определении такого типа не должно быть больше максимального.

**Контрольные вопросы:**

1. Каковы основные характеристики структур данных?
2. Какие типы данных вы знаете ?
3. Какие из них относятся к стандартным, а какие к пользовательским ?
4. Как представляются вещественные числа ?
5. Что представляют собой данные логического типа ?
6. Какие типы данных относятся к стандартным пользовательским ?
7. Какому условию должны удовлетворять допустимые числа типа INTEGER ?
8. Какие операции можно производить над целыми числами ?

9. Перечислите булевские операции.
10. Какова структура типа CHAR ?
11. Какие операции возможны над данными этого типа ?
12. Что можно вычислить с помощью данных указательного типа ?
13. Что представляет собой перечисляемый тип данных?
14. Как задается диапазонный тип ?

# 2. СТАТИЧЕСКИЕ И ПОЛУСТАТИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Структуры данных - это совокупность элементов данных и отношений между ними. При этом под элементами данных может подразумеваться как простое данное так и структура данных. Под отношениями между данными понимают функциональные связи между ними и указатели на то, где находятся эти данные.

Графическое представление элемента структуры данных.

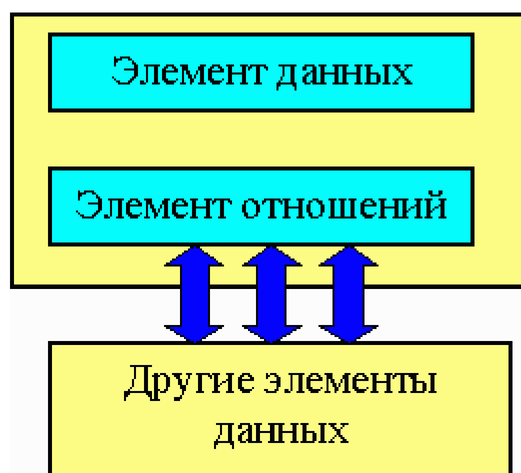



Рис.2.1

Элемент отношений - это совокупность всех связей элемента с другими элементами данных, рассматриваемой структуры.

S:=(D,R)

Где S - структура данных, D - данные и R - отношения.

Как бы сложна ни была структура данных, в конечном итоге она состоит из простых данных (см. рис. 2.2, 2.3).

***Одномерный массив***

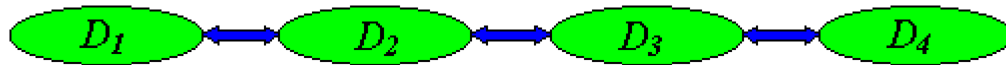



Рис.2.2

***Двумерный массив***

| j \ i | 1 | 2 | … | n |
|---|---|---|---|---|
| 1 | элемент данных | элемент данных | … | элемент данных |
| 2 | элемент данных | элемент данных | … | элемент данных |
| … | … | … | … | |
| n | элемент данных | элемент данных | … | элемент данных |

Рис.2.3

## 2.1 Уровни представления данных

Внутренний мир ЭВМ далеко не так прост, как мы думаем. Память машины состоит из миллионов триггеров, которые обрабатывают поступающую информацию

Мы, занося информацию в компьютер, представляем ее в каком-то виде, который на наш взгляд упорядочивает данные и придает им смысл. Машина отводит поле для поступающей информации и задает ей какой-то адрес. Таким образом получается, что мы обрабатываем данные на логическом уровне, как бы абстрактно, а машина делает это на физическом уровне.

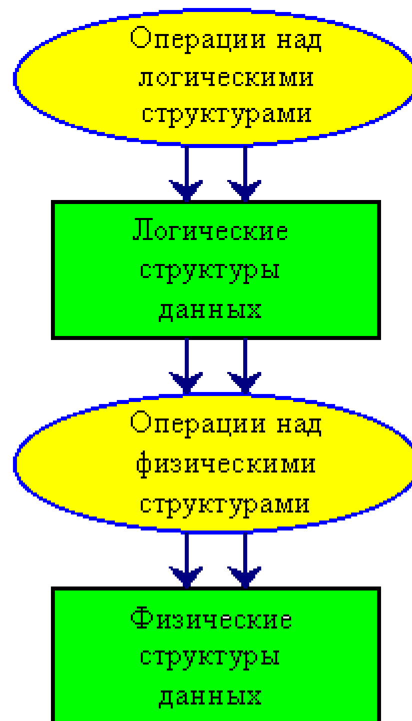



**Рис. 2.4**

Последовательность переходов от логической организации к физической показана на рис. 2.4.

## 2.2 Классификация структур данных

Структуры данных классифицируются:

1. По связанности данных в структуре:

- если данные в структуре связаны очень слабо, то такие структуры называются несвязанными (вектор, массив, строки, стеки)

- если данные в структуре связаны, то такие структуры называются связанными (связанные списки)

2. По изменчивости структуры во времени или в процессе выполнения программы:

- статические структуры - структуры, неменяющиеся до конца выполнения программы (записи, массивы, строки, вектора)

- полустатические структуры (стеки, деки, очереди)

- динамические структуры - происходит полное изменение при выполнении программы

3. По упорядоченности структуры:

- линейные (вектора, массивы, стеки, деки, записи)

- нелинейные (многосвязные списки, древовидные структуры, графы)

Наиболее важной характеристикой является изменчивость структуры во времени.

## 2.3 Статические структуры данных

### 2.3.1 Векторы

Самая простая статическая структура - это вектор. Вектор - это чисто линейная упорядоченная структура, где отношение между ее элементами есть строго выраженная последовательность элементов структуры (рис. 2.5).

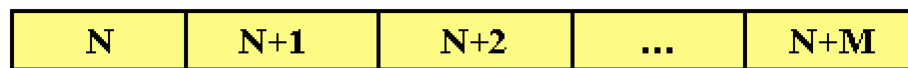



Рис. 2.5

Каждый элемент вектора имеет свой индекс, определяющий положение данного элемента в векторе. Поскольку индексы являются целыми числами, над ними можно производить операции и, таким образом, вычислять положение элемента в структуре на логическом уровне доступа. Для доступа к элементу вектора, достаточно просто указать имя вектора (элемента) и его индекс .

Для доступа к этому элементу используется функция адресации, которая формирует из значения индекса адрес слота, где находится значение исходного элемента. Для объявления в программе вектора необходимо указать его имя, количество элементов и их тип (тип данных)

Пример:

```
var
  M1: Array [1..100] of integer;
  M2: Array [1..10] of real;
```

Вектор состоит из совершенно однотипных данных и количество их строго определено.

### 2.3.2 Массивы

В общем случае элемент массива - это есть элемент вектора, который сам по себе тоже является элементом структуры (рис. 2.6).

| $B_{11}$ | $B_{12}$ | ... | $B_{1N}$ |
|---|---|---|---|
| $B_{21}$ | $B_{22}$ | ... | $B_{2N}$ |
| $B_{31}$ | $B_{32}$ | ... | $B_{3N}$ |
| $B_{41}$ | $B_{42}$ | ... | $B_{4N}$ |

Рис. 2.6

Для доступа к элементу двумерного массива необходимы значения пары индексов (номер строки и номер столбца, на пересечении которых находится элемент). На физическом уровне двумерный массив выглядит также, как и одномерный (вектор), причем трансляторы представляют массивы либо в виде строк, либо в виде столбцов.

### 2.3.3 Записи

Запись представляет из себя структуру данных последовательного типа, где элементы структуры расположены один за другим как в логическом, так и в физическом представлении. Запись предполагает множество элементов разного типа. Элементы данных в записи часто называют полями записи.

Пример:

***Запись студентов***

| 621 | Иванов И.И. | ОАПП | 95-ОА-21 |
|---|---|---|---|

Рис. 2.7

Логическая структура записи может быть представлена как в графическом виде, так и в табличном.

***Логическая структура***

| Номер | Фамилия | Факультет | Группа |
|---|---|---|---|

Рис. 2.8

***Графическая структура***

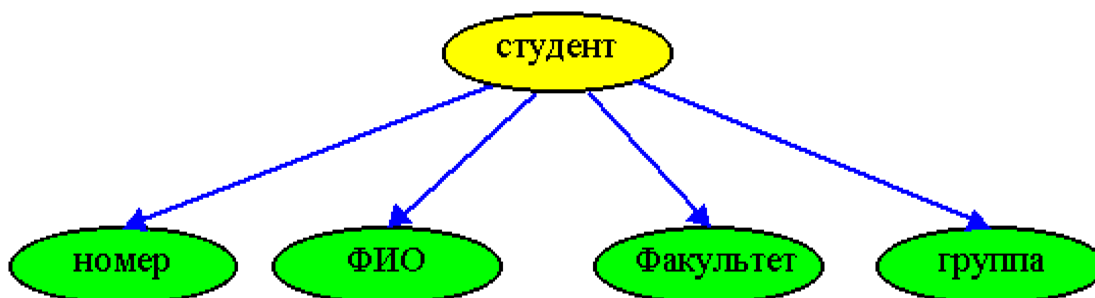



Рис.2.9

Элемент записи может включать в себя записи. В этом случае возникает сложная иерархическая структура данных.

***Пример:***

Необходимо заполнить запись о студенте, содержащую следующую информацию: N - порядковый номер студента; Имя студента, в составе которого должны быть: Фамилия, Имя, Отчество; Анкетные данные студента: год рождения, место рождения, родители: мать, отец; Факультет; Группа;

Оценки, полученные в сессию: по английскому языку и микропроцессорам.

Ниже приведены два логических представления структуры этой записи.

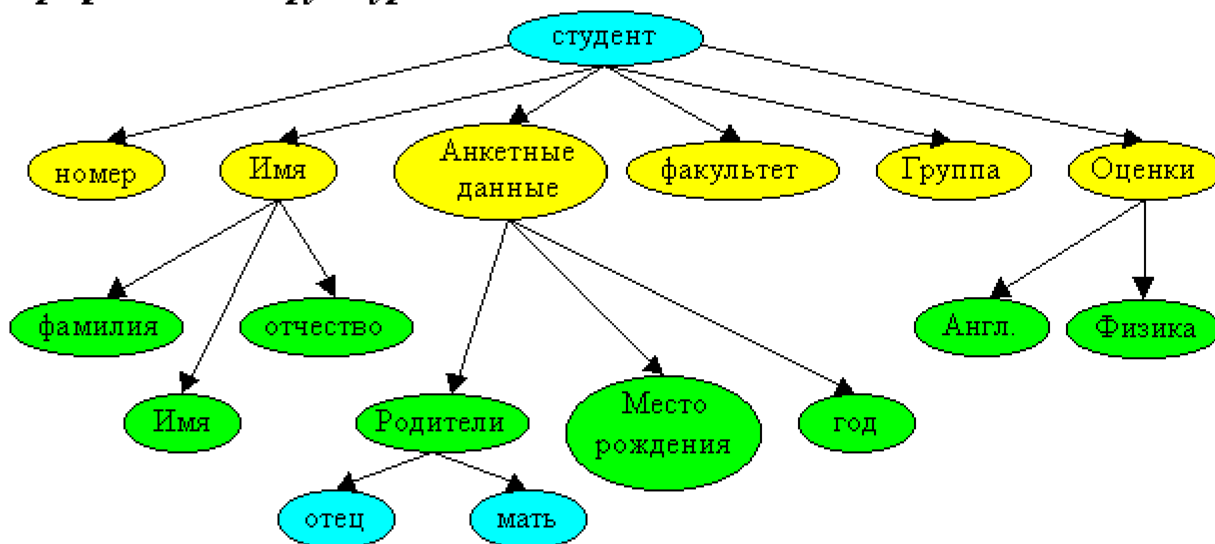



Рис.2.10

Получена четырехуровневая иерархическая структура данных. Информация содержится в листьях, остальные узлы служат для указания пути к листьям.

| | |
|---|---|
| 1-ый уровень | Студент = запись |
| 2-ой уровень | Номер |
| 2-ой уровень | Имя = запись |
| 3-ий уровень | Фамилия |
| 3-ий уровень | Имя |
| 3-ий уровень | Отчество |
| 2-ой уровень | Анкетные данные = запись |
| 3-ий уровень | Место рождения |
| 3-ий уровень | Год рождения |
| 3-ий уровень | Родители = запись |
| 4-ый уровень | Мать |
| 4-ый уровень | Отец |
| 2-ой уровень | Факультет |

2-ой уровень Группа
2-ой уровень Оценки = запись
3-ий уровень Английский
3-ий уровень Физика

Эта структура называется вложенной записью.

***Операции над записями:***

1. Прочтение содержимого поля записи.
2. Занесение информации в поле записи.
3. Все операции, которые разрешаются над полем записи, соответствующего типа.

### 2.3.4 Таблицы

Таблица - это конечный набор записей (рис. 2.11).

| N | ФИО | ГРУППА | ФАКУЛЬТЕТ | АНГЛ. | ФИЗИКА |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | | | | |
| … | | | | | |
| n | | | | | |

Рис. 2.11.

При задании таблицы указывается количество содержащихся в ней записей.

Пример:

```
Type ST = Record
        Num: Integer;
        Name: String[15];
        Fak: String[5];
```

```
        Group: String[10];
        Angl: Integer;
        Physic: Integer;

var
  Table: Array [1..19] of St;
```

Элементом данных таблицы является запись. Поэтому операции, которые производятся с таблицей - это операции, производимые с записью.

***Операции с таблицами:***

1. Поиск записи по заданному ключу.
2. Занесение новой записи в таблицу.

Ключ - это идентификатор записи. Для хранения этого идентификатора отводится специальное поле.

Составной ключ - ключ, содержащий более двух полей.

## 2.4 Полустатические структуры данных

К полустатическим структурам данных относят стеки, деки и очереди.

**Списки**

Это набор связанных элементов данных, которые в общем случае могут быть разного типа.

Пример списка:

$E_1$, $E_2$, ........, $E_n$,...   n > 1 и не зафиксировано.

Количество элементов списка может меняться в процессе выполнения программы. Различают 2 вида списков:

1) Несвязные
2) Связные

В несвязных списках связь между элементами данных выражена неявно. В связных списках в элемент данных зано-

сится указатель связи с последующим или предыдущим элементом списка.

Стеки, деки и очереди - это несвязные списки. Кроме того они относятся к последовательным спискам, в которых неявная связь отображается их последовательностью.

### 2.4.1 Стеки

Очередь вида LIFO (Last In First Out - Последним пришел, первым ушел ), при которой на обслуживание первым выбирается тот элемент очереди, который поступил в нее последним, называется стеком. Это одна из наиболее употребляемых структур данных, которая оказывается весьма удобной при решении различных задач.

В силу указанной дисциплины обслуживания, в стеке доступна единственая его позиция, которая называется вершиной стека- это позиция, в которой находится последний по времени поступления в стек элемент. Когда мы заносим новый элемент в стек, то он помещается поверх прежней вершины и теперь уже сам находится в вершине стека. Выбрать элемент можно только из вершины стека; при этом выбранный элемент исключается из стека, а в его вершине оказывается элемент, который был занесен в стек перед выбранным из него элементом (структура с ограниченным доступом к данным).

Графически стек можно представить следующим образом:

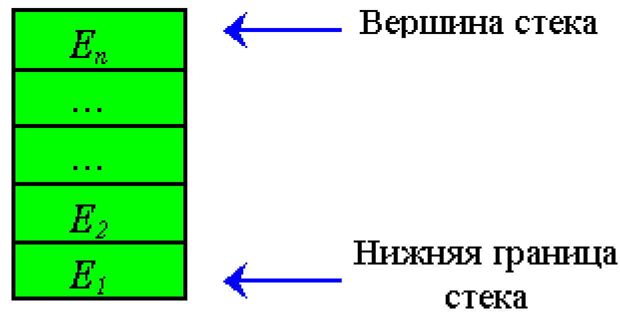



Рис. 2.12

Первый элемент заносится вниз стека . Выборка из стека осуществляется с вершины, поэтому стек является структурой с ограниченным доступом.

***Операции, производимые над стеками:***

1. Занесение элемента в стек.

Push(S,I), где S - идентификатор стека, I - заносимый элемент.

2. Выборка элемента из стека.

Pop(S)

3. Определение пустоты стека.

Empty(S)

4. Прочтение элемента без его выборки из стека.

StackTop(S)

**Пример реализации стека на Паскале с использованием одномерного массива:**

```
type
  Stack = Array[1..10] of Integer; {стек вместимостью 10 элементов типа Integer}

var
  StackCount: Integer; {Переменная - указатель на вершину стека, ее начальное  значение должно быть равно 0}
```

```
  S: Stack; {Объявление стека}

Procedure Push(I: Integer; var S: Stack);
begin
  Inc(StackCount);
S[StackCount]:=I;
end;

Procedure Pop(var I: Integer; S: Stack);
begin
  I:=S[StackCount];
Dec(StackCount);
end;

Function Empty: Boolean;
begin
  If  I = 0 then Empty:=True Else Empty:=False;
end;
```

При выполнении операции выборки из стека сначала необходимо осуществить проверку на пустоту стека. Если он пуст, то Empty возвращает значение True. Если Empty возвращает False, то это означает, что стек не пуст и из него еще можно выбирать элементы.

```
Procedure StackTop(var I: Integer; S: Stack);
begin
  I:=S[StackCount];
end;
```

### 2.4.2 Очередь

Понятие очереди всем хорошо известно из повседневной жизни. Элементами очереди в общем случае являются заказы на то или иное обслуживание.

В программировании имеется структура данных, которая называется очередью. Эта структура данных используется, например, для моделирования реальных очередей с целью определения их характеристик при данном законе поступления заказов и дисциплине их обслуживания. По своему существу очередь является полустатической структурой- с течением времени и длина очереди, и состав могут изменяться.

На рис. 2. 13 приведена очередь, содержащая четыре элемента — А, В, С и D. Элемент А расположен в начале очереди, а элемент D — в ее конце. Элементы могут удаляться только из начала очереди, то есть первый помещаемый в очередь элемент удаляется первым. По этой причине очередь часто называют списком, организованным по принципу «первый размещенный первым удаляется» в противоположность принципу стековой организации — «последний размещенный первым удаляется».

Дисциплину обслуживания, в которой заказ, поступивший в очередь первым, выбирается первым для обслуживания (и удаляется из очереди), называется FIFO (First In First Out - Первым пришел, первым ушел). Очередь открыта с обеих сторон.

Таким образом, изъятие компонент происходит из начала очереди, а запись- в конец. В этом случае вводят два указателя: один - на начало очереди, второй- на ее конец.

Реальная очередь создается в памяти ЭВМ в виде одномерного массива с конечным числом элементов, при этом необходимо указать тип элементов очереди, а также необходима переменная в работе с очередью.

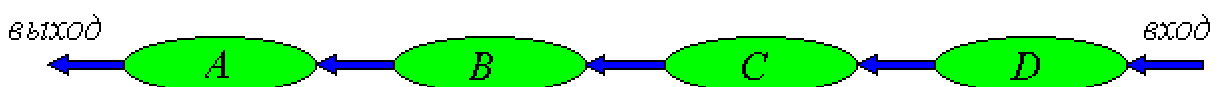



Рис. 2.13.

Физически очередь занимает сплошную область памяти и элементы следуют друг за другом, как в последовательном списке.

***Операции, производимые над очередью:***

Для очереди определены три примитивные операции. Операция insert (q,x) помещает элемент x в конец очереди q. Операция remove(q) удаляет элемент из начала очереди q и присваивает его значение переменной x. Третья операция, empty (q), возвращает значение true или false в зависимости от того, является ли данная очередь пустой или нет. Кроме того. Учитывая то, что полустатическая очередь реализуется на одномерном массиве, необходимо следить за возможностью его переполнения. Сэтой целью вводится опнрация full(q).

Операция insert может быть выполнена всегда, поскольку на количество элементов, которые может содержать очередь, никаких ограничений не накладывается. Операция remove, однако, применима только к непустой очереди, поскольку невозможно удалить элемент из очереди, не содержащей элементов. Результатом попытки удалить элемент из пустой очереди является возникновение исключительной ситуации **потеря значимости.** Операция empty, разумеется, выполнима всегда.

**Пример реализации очереди в виде одномерного массива на Паскале:**

```
const
  MaxQ = ...
type
  E = ...
  Queue = Array [1..MaxQ] of E;
var
  Q: Queue;
  F, R: Integer;
```

{Указатель F указывает на начало очереди. Указатель R указывает на конец очереди. Если очередь пуста, то F = 1, а R = 0 (То есть **R < F – условие пустоты очереди**).}

```
Procedure Insert(I: Integer; var Q: Queue);
begin
 Inc(R);
 Q[R]:=I;
end;

Function Empty(Q: Queue): Boolean;
begin
  If R < F then Empty:=True Else Empty:=False;
end;

Procedure Remove(var I: Integer; Q: Queue);
begin
 If Not Empty(Q) then
   begin
    I:=Q[F];
    Inc(F);
   end;
end;
```

**Пример работы с очередью с использованием стандартных процедур.**

MaxQ = 5

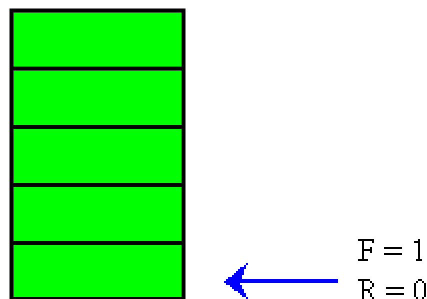



Рис. 2.14.

Производим вставку элементов А, В и С в очередь.

Insert(q, A);
Insert(q, B);
Insert(q, C);

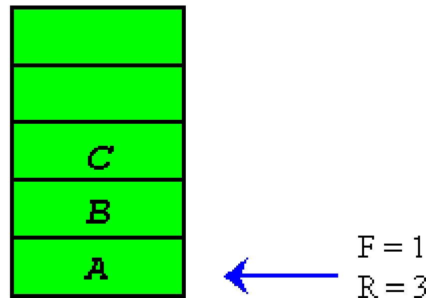



Рис. 2.15.

Убираем элементы А и В из очереди.
Remove(q);
Remove(q);

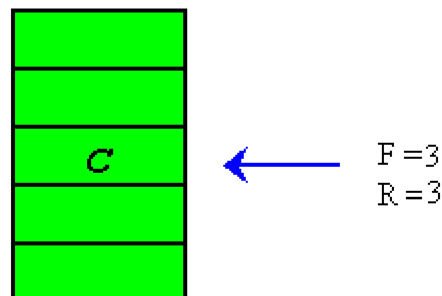



Рис. 2.16.

К сожалению, при подобном представлении очереди, возможно возникновение абсурдной ситуации, при которой очередь является пустой, однако новый элемент разместить в ней нельзя (рассмотрите последовательность операций удаления и вставки, приводящую к такой ситуации). Ясно, что реализация очереди при помощи массива является неприемлемой.

Одним из решений возникшей проблемы может быть модификация операции remove таким образом, что при удалении очередного элемента вся очередь смещается к началу массива. Операция remove (q) может быть в этом случае реализована следующим образом.

```
X = q[1]
For I =1 to R-1
        q[I] =q[I+1]
next I
R =R-1
```

Переменная F больше не требуется, поскольку первый элемент массива всегда является началом очереди. Пустая очередь представлена очередью, для которой значение R равно нулю.

Однако этот метод весьма непроизводителен. Каждое удаление требует перемещения всех оставшихся в очереди элементов. Если очередь содержит 500 или 1000 элементов, то очевидно, что это весьма неэффективный способ. Далее, операция удаления элемента из очереди логически предполагает манипулирование только с одним элементом, т. е. с тем, который расположен в начале очереди. Реализация данной операции должна отражать именно этот факт, не производя при этом множества дополнительных действий.

Другой способ предполагает рассматривать массив, который содержит очередь в виде замкнутого кольца, а не линейной последовательности, имеющей начало и конец. Это означает, что первый элемент очереди следует сразу же за последним. Это означает, что даже в том случае, если последний элемент занят, новое значение может быть размещено сразу же за ним на месте первого элемента, если этот первый элемент пуст.

***Организация кольцевой очереди.***

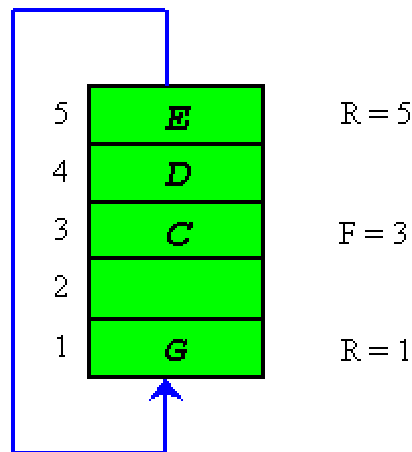



**Рис. 2.17**

Рассмотрим пример. Предположим, что очередь содержит три элемента - в позициях 3, 4 и 5 пятиэлементного массива. Этот случай, показанный на рис. 2.17. Хотя массив и не заполнен, последний элемент очереди занят.

Если теперь делается попытка поместить в очередь элемент G, то он будет записан в первую позицию массива, как это показано на рис. 2.17. Первый элемент очереди есть Q(3), за которым следуют элементы Q (4), Q(5) и Q(l).

К сожалению, при таком представлении довольно трудно определить, когда очередь пуста. Условие R<F больше не годится для такой проверки, поскольку на рис. 2. 17 показан случай, при котором данное условие выполняется, но очередь при этом не является пустой.

Одним из способов решения этой проблемы является введение соглашения, при котором значение **F есть индекс элемента массива, немедленно предшествующего первому элементу очереди**, а не индексу самого первого элемента. В этом случае, поскольку R содержит индекс последнего элемента очереди, условие F = R подразумевает, что очередь пуста.

Отметим, что перед началом работы с очередью, в F и R устанавливается значение последнего индекса массива, а не 0 и 1, поскольку при таком представлении очереди последний

элемент массива немедленно предшествует первому элементу. Поскольку R = F, то очередь изначально пуста.

***Основные операции с кольцевой очередью:***

1. Вставка элемента q в очередь x.

Insert(q,x)

2. Извлечение элемента из очереди x.

Remove(q)

3. Проверка очереди на пустоту.

Empty(q)

***Операция empty (q)*** может быть записана следующим образом:

```
if F = R
    then empty = true
    else empty = false
endif
return
```

***Операция remove (q)*** может быть записана следующим образом:

```
empty (q)
if empty = true
      then print «выборка из пустой очереди»
      stop
endif
if F =maxQ
      then F =1
      else F = F+1
endif
x = q(F)
return
```

Отметим, что значение F должно быть модифицировано до момента извлечения элемента.

***Операция вставки insert (q,x).***

Для того чтобы запрограммировать операцию вставки, должна быть проанализирована ситуация, при которой возникает переполнение. Переполнение происходит в том случае, если весь массив уже занят элементами очереди и при этом делается попытка разместить в ней еще один элемент. Рассмотрим, например, очередь на рис. 2. 17. В ней находятся три элемента — C, D и E, соответственно расположенные в Q (3), Q (4) и Q (5). Поскольку последний элемент в очереди занимает позицию Q (5), значение R равно 5. Так как первый элемент в очереди находится в Q (3), значение F равно 2. На рис. 2. 17 в очередь помещается элемент G, что приводит к соответствующему изменению значения R. Если произвести следующую вставку, то массив становится целиком заполненным, и попытка произвести еще одну вставку приводит к переполнению. Это регистрируется тем фактом, что F = R, а это как раз и указывает на переполнение. Очевидно, что при такой реализации нет возможности сделать различие между пустой и заполненной очередью Разумеется, такая ситуация удовлетворить нас не может

Одно из решений состоит в том, чтобы пожертвовать одним элементом массива и позволить очереди расти до объема на единицу меньшего максимального. Так, если массив из 100 элементов объявлен как очередь, то очередь может содержать до 99 элементов. Попытка разместить в очереди 100-й элемент приведет к переполнению. Подпрограмма insert может быть записана следующим образом:

```
if R = max(q)
    then R = 1
    else R = R+1
endif
```

```
'проверка на переполнение
if R = F
    then print «переполнение очереди»
    stop
endif
q (r) =x
return
```

Проверка на переполнение в подпрограмме insert производится после установления нового значения для R, в то время как проверка на потерю значимости в подпрограмме remove производится сразу же после входа в подпрограмму до момента обновления значения F.

### 2.4.3 Дек

От английского DEQ - Double Ended Queue (Очередь с двумя концами)

Особенностью деков является то, что запись и считывание элементов может производиться с двух концов (рис. 2.18).

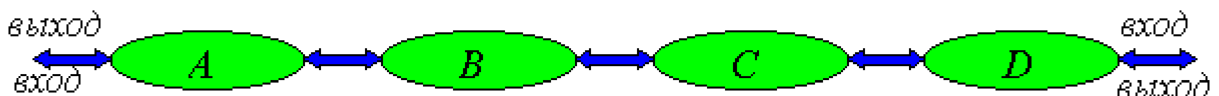



Рис. 2.18.

Дек можно рассматривать и в виде двух стеков, соединенных нижними границами. Возьмем пример, иллюстрирующий принцип построения дека. Рассмотрим состав на железнодорожной станции. Новые вагоны к составу можно добавлять либо к его концу, либо к началу. Аналогично, чтобы отсоединить от состава вагон, находящийся в середине, нужно сначала отсоединить все вагоны или вначале, или в конце

состава, отсоединить нужный вагон, а затем присоединить их снова.

***Операции над деками:***

1. Insert - вставка элемента.
2. Remove - извлечение элемента из дека.
3. Empty - проверка на пустоту.
4. Full - проверка на переполнение.

**Контрольные вопросы**

1. Что такое структуры данных?
2. Назовите уровни представления данных?
3. Какова классификация структур данных?
4. Какая статическая структура является самой простой?
5. Перечислите основные элементы таблицы.
6. Назовите их основные особенности.
7. Что такое вектор?
8. Что представляет из себя запись?
9. Какова структура записи?
10. К каким структурам данных относятся очереди и стеки ?
11. Каково правило выборки элемента из стека ?
12. Какая операция читает верхний элемент стека без его удаления ?
13. Какую дисциплину обслуживания принято называть FIFO, а какую - LIFO ?
14. Признак заполнения кольцевой очереди ?
15. Признак пустой очереди ?
16. Что называется списком?
17. Перечислите виды списков.
18. Назовите элементы очереди.
19. Как организуется кольцевая очередь?
20. Какова особенность деков?

# 3. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

До сих пор мы рассматривали только статические программные объекты. Однако использование при программировании только статических объектов может вызвать определенные трудности, особенно с точки зрения получения эффективной машинной программы. Дело в том, что иногда мы заранее не знаем не только размера значения того или иного программного объекта, но также и того, будет ли существовать этот объект или нет. Такого рода программные объекты, которые возникают уже в процессе выполнения программы или размер значений которых определяется при выполнении программы, называются динамическими объектами.

Динамические структуры данных имеют 2 особенности:

1) Заранее не определено количество элементов в структуре.

2) Элементы динамических структур не имеют жесткой линейной упорядоченности. Они могут быть разбросаны по памяти.

Чтобы связать элементы динамических структур между собой в состав элементов помимо информационных полей входят поля указателей (рис. 3.1) (связок с другими элементами структуры).

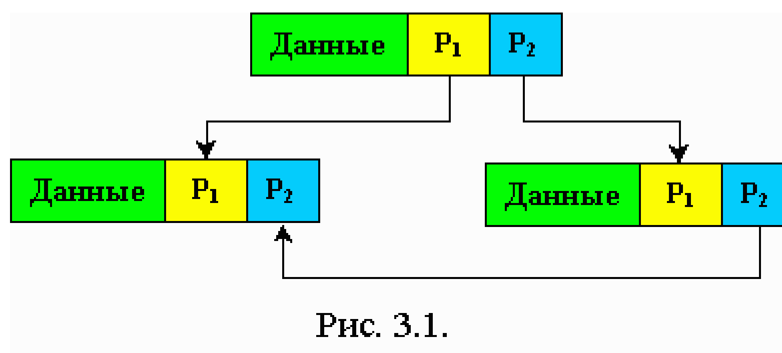



Рис. 3.1.

$P_1$ и $P_2$ это указатели, содержащие адреса элементов, с которыми они связаны. Указатели содержат номер слота.

## 3.1 Связные списки

Наиболее распространенными динамическими структурами являются связанные списки. С точки зрения логического представления различают линейные и нелинейные списки.

В линейных списках связи строго упорядочены: указатель предыдущего элемента содержит адрес последующего элемента или наоборот.

К линейным спискам относятся односвязные и двусвязные списки. К нелинейным - многосвязные.

Элемент списка в общем случае представляет собой поле записи и одного или нескольких указателей.

### 3.1.1 Односвязные списки

Элемент односвязного списка содержит два поля (рис. 3.2): информационное поле (INFO) и поле указателя (PTR).

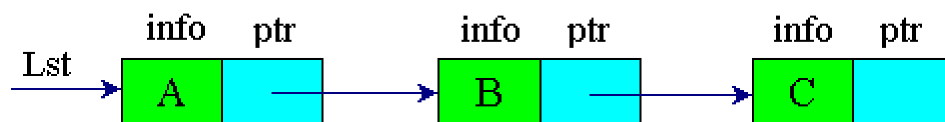



Рис. 3.2

Особенностью указателя является то, что он дает только адрес последующего элемента списка. Поле указателя последнего элемента в списке является пустым (NIL). LST - указатель на начало списка. Список может быть пустым, тогда LST будет равен NIL.

Доступ к элементу списка осуществляется только от его начала, то есть обратной связи в этом списке нет.

### 3.1.2 Кольцевой односвязный список

Кольцевой односвязный список получается из обычного односвязного списка путем присваивания указателю последнего элемента списка значение указателя начала списка (рис 3.3).

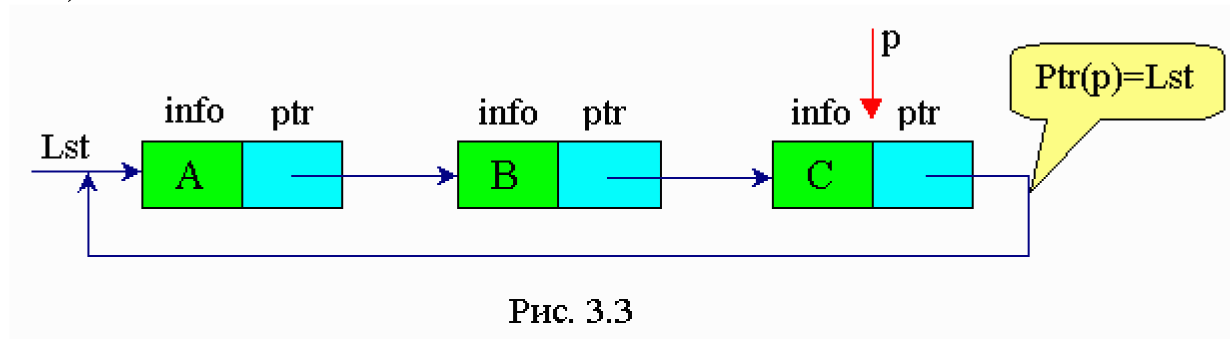



Рис. 3.3

***Простейшие операции, производимые над односвязными списками***

*1) Вставка в начало односвязного списка.*

Надо вставить в начало односвязного списка элемент с информационным полем D. Для этого необходимо сгенерировать пустой элемент (P=GetNode). Информационному полю этого элемента присвоить значение D (INFO(P)=D), значению указателя на этот элемент присвоить значение указателя на начало списка (Ptr(P) = Lst), значению указателя начала списка присвоить значение указателя P (Lst = P).

*Реализация на Паскале:*

```
type
  PNode = ^TNode;
  TNode = record
    Info: Integer;  {тип элементов списка - может быть любым}
    Next: PNode;
  end;
var
  Lst: PNode;  {указатель на начало списка}
  P: PNode;
```

Вставка в начало

New(P); {создание нового элемента}

P^.Info:=D;

P^.Next:=Lst; {P указывает на начало списка, но Lst не указывает на P - новое начало}

Lst:=P; {Lst указывает на новое начало списка}

*2) Удаление элемента из начала односвязного списка.*

Надо удалить первый элемент списка, но запомнить информацию, содержащуюся в поле Info этого элемента. Для этого введем указатель P, который будет указывать на удаляемый элемент (P = Lst). В переменную X занесем значение информационного поля Info удаляемого элемента (X=Info(P)). Значению указателя на начало списка присвоим значение указателя следующего за удаляемым элемента (Lst = Ptr(P)). Удалим элемент (FreeNode(P)).

*Реализация на Паскале:*

Удаление из начала

P:=Lst;

X:=P^.Info;

Lst:=P^.Next;

Dispose(P); {Удаляет элемент из динамической памяти}

### 3.1.3 Двусвязный список

Использование однонаправленных списков при решении ряда задач может вызвать определенные трудности. Дело в том, что по однонаправленному списку можно двигаться только в одном направлении, от заглавного звена к последнему звену списка. Между тем нередко возникает необходимость произвести какую-либо обработку элементов, предшествующих элементу с заданным свойством. Однако после нахождения элемента с этим свойством в односвязном списке у нас нет возможности получить достаточно удобный и

быстрый доступ к предыдущим элементам- для достижения этой цели придется усложнить алгоритм, что неудобно и нерационально.

Для устранения этого неудобства добавим в каждое звено списка еще одно поле, значением которого будет ссылка на предыдущее звено списка. Динамическая структура, состоящая из элементов такого типа, называется двунаправленным или двусвязным списком.

Двусвязный список характеризуется тем, что у любого элемента есть два указателя. Один указывает на предыдущий элемент (обратный), другой указывает на следующий элемент (прямой) (рис. 3.4).

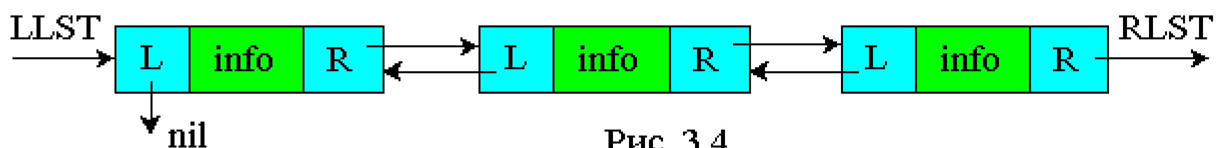



Рис. 3.4

Фактически двусвязный список это два односвязных списка с одинаковыми элементами, записанных в противоположной последовательности.

### 3.1.4 Кольцевой двусвязный список

В программировании двусвязные списки часто обобщают следующим образом: в качестве значения поля Rptr последнего звена принимают ссылку на заглавное звено, а в качестве значения поля Lptr заглавного звена- ссылку на последнее звено. Список замыкается в своеобразное кольцо: двигаясь по ссылкам, можно от последнего звена переходить к заглавному и наоборот.

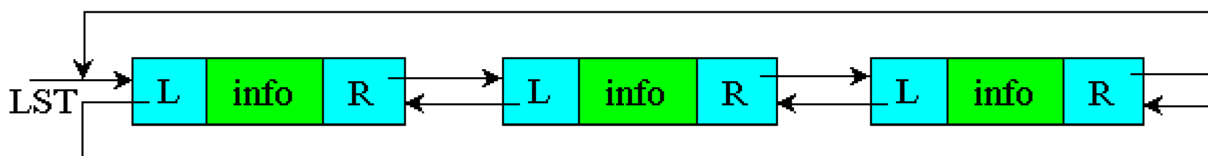



Рис. 3.5

*Операции над двусвязными списками:*

- создание элемента списка;
- поиск элемента в списке;
- вставка элемента в указанное место списка;
- удаление из списка заданного элемента.

## 3.2 Реализация стеков с помощью односвязных списков

Любой односвязный список может рассматриваться в виде стека. Однако список по сравнению со стеком в виде одномерного массива имеет преимущество, так как заранее не задан его размер.

*Стековые операции, применимые к спискам*

1). Чтобы добавить элемент в стек, надо в алгоритме заменить указатель Lst на указатель Stack (операция Push(S, X)).

```
P = GetNode
Info(P) = X
Ptr(P) = Stack
Stack = P
```

2) Проверка стека на пустоту (Empty(S))

```
If Stack = Nil then Print “Стек пуст”
Stop
```

3) Выборка элемента из стека (POP(S))

```
P = Stack
X = Info(P)
Stack = Ptr(P)
FreeNode(P)
```

*Реализация на Паскале:*
***Стек***

Вместо указателя Lst используется указатель Stack
Процедура Push (S, X)

```
procedure Push(var Stack: PNode; X: Integer);
var
  P: PNode;
begin
  New(P);
  P^.Info:=X;
  P^.Next:=Stack;
  Stack:=P;
end;
```

Проверка на пустоту (Empty)

```
function Empty(Stack: PNode): Boolean;
begin
  If Stack = nil then Empty:=True Else Empty:=False;
end;
```

Процедура Pop

```
procedure Pop(var X: Integer; var Stack: PNode);
var
  P: PNode;
begin
  P:=Stack;
  X:=P^.Info;
  Stack:=P^.Next;
  Dispose(P);
end;
```

### *Операции с очередью, применимые к спискам.*

Указатель начала списка принимаем за указатель начала очереди F, а указатель R, указывающий на последний элемент списка - за указатель конца очереди.

1) Операция удаления из очереди (Remove(Q, X)).

Операция удаления из очереди должна проходить из ее начала.

```
P = F
F = Ptr(P)
X = Info(P)
FreeNode(P)
```

2) Проверка очереди на пустоту. (Empty(Q))

```
If F = Nil then Print "Очередь пуста"
Stop
```

3) Операция вставки в очередь. (Insert(Q, X))

Операция вставки в очередь должна осуществляться к ее концу.

```
P = GetNode
Info(P) = X
Ptr(P) = Nil
Ptr(R)= P
R = P
```

*Реализация на Паскале:*

```
procedure Remove(var X: Integer; Fr: PNode);
var
  P: PNode;
begin
  New(P);
  P:=Fr;
  X:=P^.Info;
  Fr:=P^.Next;
  Dispose(P);
```

```
end;

function Empty(Fr: PNode): Boolean;
begin
  If Fr = nil then Empty:=True Else Empty:=False;
end;

procedure Insert(X: Insert; var Re: PNode);
var
  P: PNode;
begin
  New(P);
  P^.Info:=X;
  P^.Next:=nil;
  Re^.Next:=P;
end;
```

### 3.3 Организация операций Getnode, Freenode и утилизация освободившихся элементов

Для более эффективного использования памяти компьютера (для исключения *мусора*, то есть неиспользуемых элементов) при работе его со списками создается свободный список, имеющий тот же формат полей, что и у функциональных списков.

Если у функциональных списков разный формат, то надо создавать свободный список каждого функционального списка.

Количество элементов в свободном списке должно быть определено задачей, которую решает программа. Как правило, свободный список создается в памяти машины как стек. При этом создание (*GetNode*) нового элемента эквивалентно выборке элемента свободного стека, а операция *FreeNode* - добавлению в свободный стек освободившегося элемента.

Пусть нам необходимо создать пустой список по типу стека (рис. 3.6) с указателем начала списка - AVAIL. Разработаем процедуры, которые позволят нам создавать пустой элемент списка и освобождать элемент из списка.

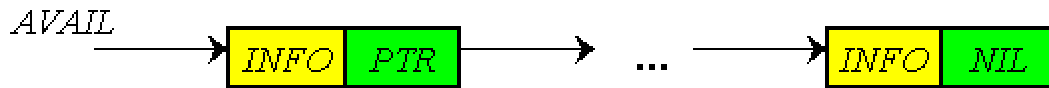



Рис. 3.6

### 3.3.1 Операция GetNode

Разработаем процедуру, которая будет создавать пустой элемент списка с указателем P.

Для реализации операции GetNode необходимо указателю сгенерированного элемента присвоить значение указателя начала свободного списка, а указатель начала перенести к следующему элементу.

```
P = Avail
Avail = Ptr(Avail)
```

Перед этим надо проверить, есть ли элементы в списке. Пустота свободного списка (Avail = Nil), эквивалентна переполнению функционального списка.

```
If Avail = Nil then Print “Переполнение”  Stop
Else
  P = Avail
  Avail = Ptr(Avail)
Endif
```

### 3.3.2 Операция FreeNode

При освобождении элемента Nod(P) из функционального списка операцией FreeNode(P), он заносится в свободный список.

Ptr(P) = Avail

Avail = P

### 3.3.3 Утилизация освободившихся элементов в многосвязных списках

Стандартные операции возвращения освободившегося элемента списка в пул свободных элементов не всегда дают эффект, если используются нелинейные многосвязные списки.

Первый способ утилизации - метод счетчиков. В каждый элемент многосвязного списка вставляется поле счетчика, который считает количество ссылок на данный элемент. Когда счетчик элемента оказывается в нулевом состоянии, а поля указателей элемента находятся в состоянии nil, этот элемент может быть возвращен в пул свободных элементов.

Второй способ - метод сборки мусора (метод маркера). Если с каким-то элементом установлена связь, то однобитовое поле элемента (маркер) устанавливается в “1”, иначе - в “0”. По сигналу переполнения ищутся элементы, у которых маркер установлен в ноль, т. е. включается программа сборки мусора, которая просматривает всю отведенную память и возвращает в список свободных элементов все элементы, не помеченные маркером.

## 3.4 Односвязный список, как самостоятельная структура данных

Просмотр односвязного списка может производиться только последовательно, начиная с головы (с начала) списка.

Если необходимо просмотреть предыдущий элемент, то надо снова возвращаться к началу списка. Это - недостаток односвязных списков по сравнению со статическими структурами типа массива. Списковая структура проявляет свои достоинства, когда число элементов списка велико, а вставку или удаление необходимо произвести внутри списка.

Пример:

Необходимо вставить в существующий массив элемент X между 5 и 6 элементами.

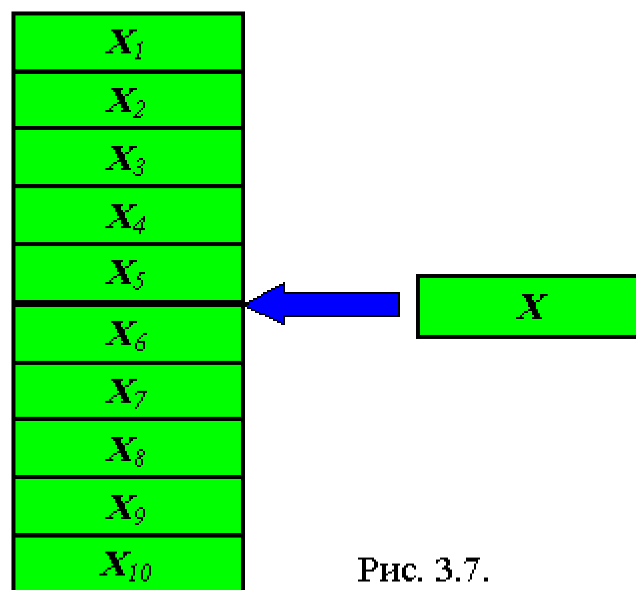



Рис. 3.7.

Для проведения данной операции в массиве нужно “опустить” все элементы, начиная с X6 - увеличить их индексы на единицу. В результате вставки получаем следующий массив (рис. 3.7, 3.8):

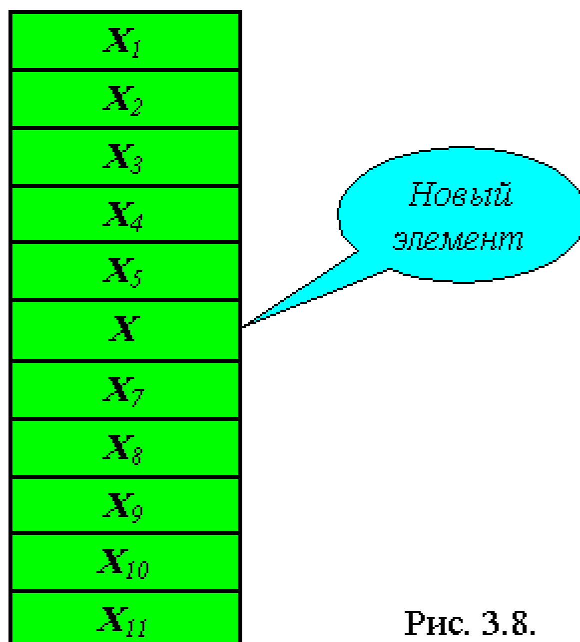



Рис. 3.8.

Данная процедура может занимать очень значительное время. В противоположность этому, в связанном списке операция вставки состоит в изменении значения 2-х указателей и генерации свободного элемента. Причём время, затраченное на выполнение этой операции, является постоянным и не зависит от количества элементов в списке.

### 3.4.1 Вставка и извлечение элементов из списка

Определяем элемент, после которого необходимо вставить элемент в список. Вставка производится с помощью процедуры InsAfter(Q, X), а удаление - DelAfter(Q, X).

При этом рабочий указатель P будет указывать на элемент, после которого необходимо произвести вставку или удаление (рис 29).

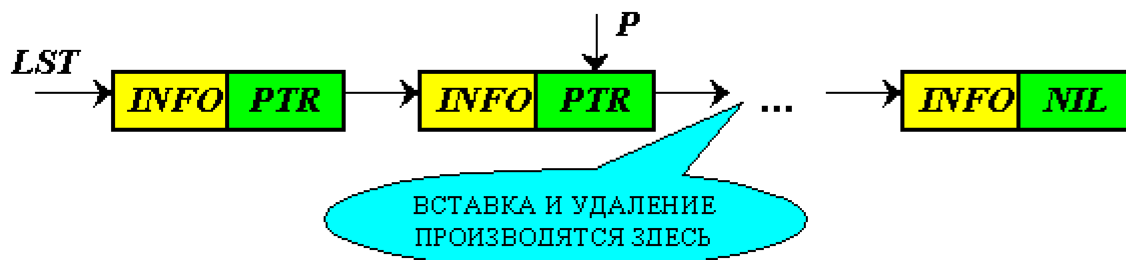



Рис. 3.9.

Пример:

Пусть необходимо вставить новый элемент с информационным полем X после элемента, на который указывает рабочий указатель P.

Для этого:

1) Необходимо сгенерировать новый элемент.

Q = GetNode

2) Информационному полю этого элемента присвоить значение X.

Info(Q) = X

3) Вставить полученный элемент.

Ptr(Q) = Ptr(P)
Ptr(P) = Q

Это и есть процедура InsAfter(Q, X).

Пусть необходимо удалить элемент списка, который следует после элемента, на который указывает рабочий указатель P.

Для этого:

1) Присваиваем Q значение указателя на удаляемый элемент.

Q = Ptr(P)

2) В переменную X сохраняем значение информационного поля удаляемого элемента.

X = Info(Q)

3) Меняем значение указателя на удаляемый элемент на значение указателя на следующий элемент и производим удаление .

Ptr(P) = Ptr(Q)
FreeNode(Q)

Это и есть процедура DelAfter(P, X).

На языке Паскаль вышеописанные процедуры будут выглядеть следующим образом:

**procedure** InsAfter(var Q: PNode; X: Integer);
var

```
  Q: PNode;
begin
  New(Q);
  Q^.Info:=X;
  Q^.Next:=P^.Next;
  P^.Next:=Q;
procedure DelAfter(var Q: PNode; var X: Integer);
var
  Q: PNode;
begin
  Q:=P^.Next;
  P^.Next:=Q^.Next;
  X:=Q^.Info;
  Dispose(Q);
end;
```

### 3.4.2 Примеры типичных операций над списками

***Задача 1.***

Требуется просмотреть список и удалить элементы, у которых информационные поля равны 4.

Обозначим P - рабочий указатель, в начале процедуры P = Lst. Введем также указатель Q, который отстает на один элемент от P. Когда указатель P найдет элемент, последний будет удален относительно указателя Q как последующий элемент.

```
Q = Nil
P = Lst
While P <> Nil do
  If Info(P) = 4 then
    If Q = Nil then
      Pop(Lst)
      FreeNode(P)
      P = Lst
    Else
```

```
    DelAfter(Q, X)
   EndIf
  Else
   Q = P
   P = Ptr(P)
  EndIf
EndWhile
```

***Задача 2.***

Дан упорядоченный по возрастанию Info - полей список. Необходимо вставить в этот список элемент со значением X, не нарушив упорядоченности списка.

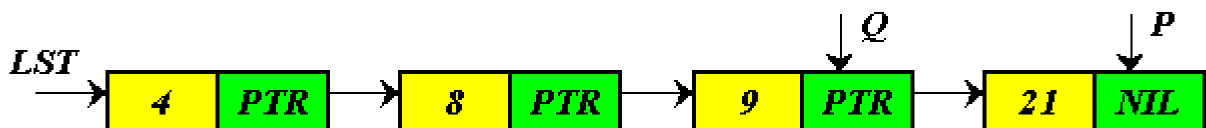



Рис. 3.10.

Пусть X = 16. Начальное условие: Q = Nil, P = Lst. Вставка элемента должна произойти между 3 и 4 элементом (рис.3.10).

```
Q =Nil
P = Lst
While (P <> Nil) and (X > Info(P)) do
  Q = P
  P = Ptr(P)
EndWhile
if Q = Nil then
  Push(Lst, X)
EndIf
InsAfter(Q, X)
```

Реализация примеров на языке Паскаль:

***Задача 1:***

```
Q:=nil;
```

```
P:=Lst;
While P <> nil do
  If P^.Info = 4 then
    begin
      If Q = nil then
        begin
          Pop(Lst);
          P:=Lst;
        end
      Else Delafter(Q,X);
    End;
  Else
    begin
      Q:=P;
      P:=P^.Next;
    end;
  end;
```

***Задача 2:***

```
Q:=nil;
P:=Lst;
While (P <> nil) and (X > P^.Info) do
  begin
    Q:=P;
    P:=P^.Next;
  end;
{В эту точку производится вставка}
If Q = nil then Push(Lst, X);
InsAfter(Q, X);
End;
```

### 3.4.3 Элементы заголовков в списках

Для создания списка с заголовком в начало списка вводится дополнительный элемент, который может содержать информацию о списке (рис. 3.11).

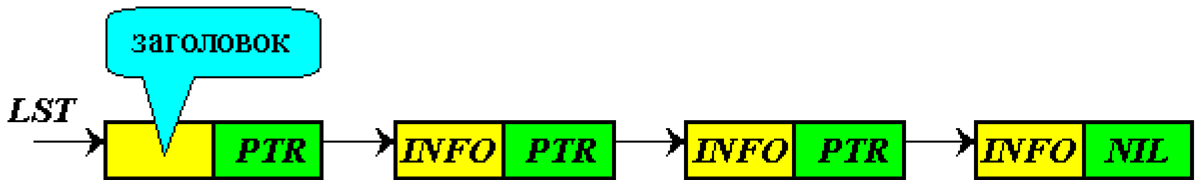



Рис. 3.11.

В заголовок списка часто помещают динамическую переменную, содержащую количество элементов в списке (не считая самого заголовка).

Если список пуст, то остается только заголовок списка (рис. 3.12).

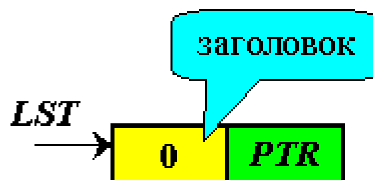



Рис. 3.12.

Также удобно занести в информационное поле заголовка значение указателя конца списка. Тогда, если список используется как очередь, то Fr = Lst, а Re = Info(Lst).

Информационное поле заголовка можно использовать для хранения рабочего указателя при просмотре списка P = Info(Lst). То есть заголовок - это дескриптор структуры данных.

## 3.5 Нелинейные связанные структуры

Двусвязный список может быть нелинейной структурой данных, если вторые указатели задают произвольный порядок следования элементов (рис.3.13).

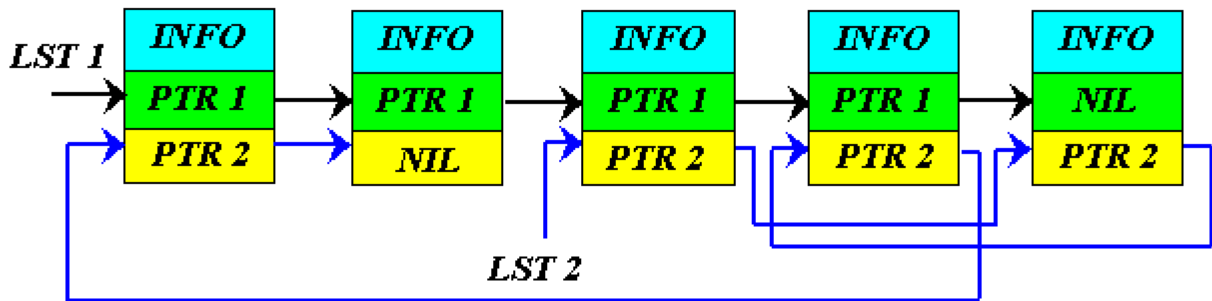



Рис. 3.13

LST1 - указатель на начало 1 - ого списка (ориентированного указателем P1). Он линейный и состоит из 5-и элементов.

2-ой список образован из этих же самых элементов, но в произвольной последовательности. Началом 2-ого списка является 3-ий элемент, а концом 2-ой элемент.

В общем случае элемент списочной структуры может содержать сколь угодно много указателей, то есть может указывать на любое количество элементов.

Можно выделить 3 признака отличия нелинейной структуры:

1) Любой элемент структуры может ссылаться на любое число других элементов структуры, то есть может иметь любое число полей-указателей.

2) На данный элемент структуры может ссылаться любое число других элементов этой структуры.

3) Ссылки могут иметь вес, то есть подразумевается иерархия ссылок.

Представим, что имеется дискретная система, в графе состояния которой узлы - это состояния, а ребра - переходы из состояния в состояние (рис. 3.14).

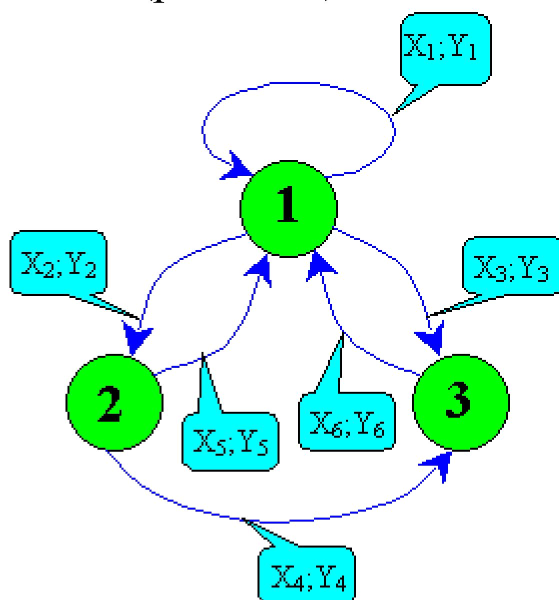



**Рис. 3.14.**

Входной сигнал в систему это X. Реакцией на входной сигнал является выработка выходного сигнала Y и переход в соответствующее состояние.

Граф состояния дискретной системы можно представит в виде двусвязного нелинейного списка. При этом в информационных полях должна записываться информация о состояниях системы и ребрах. Указатели элементов должны формировать логические ребра системы (рис. 3.15).

Для реализации вышесказанного:

1) должен быть создан список, отображающий состояния системы (1, 2, 3);

2) должны быть созданы списки, отображающие переходы по ребрам из соответствующих состояний .

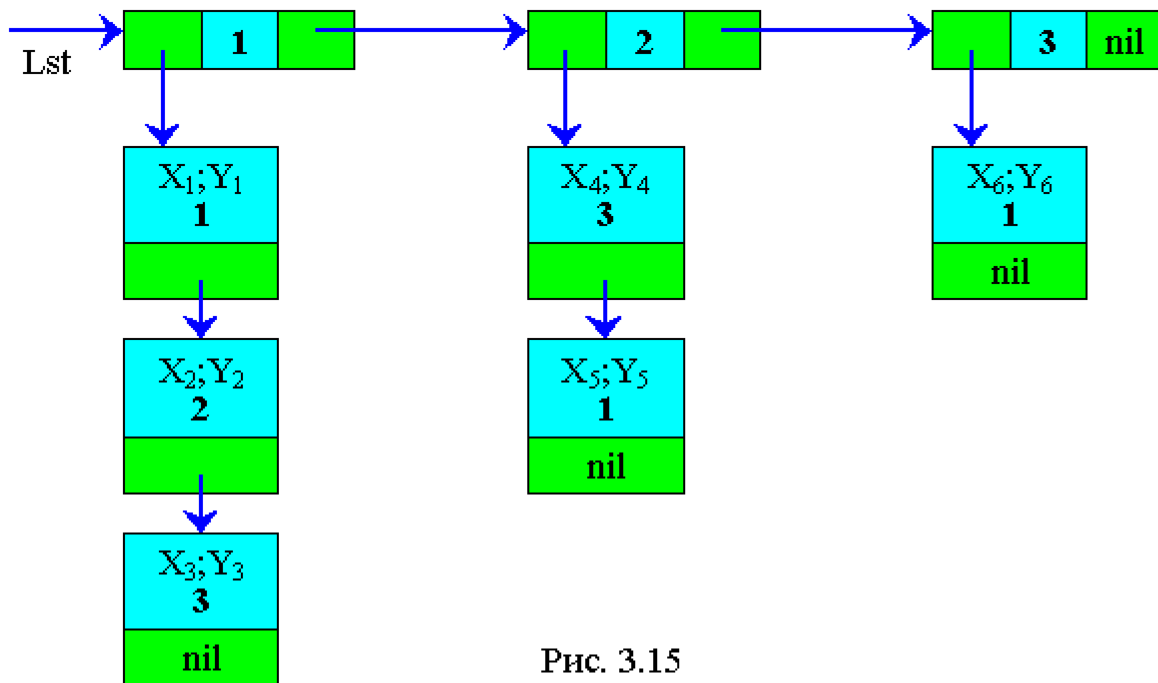


Рис. 3.15

В общем случае при реализации многосвязной структуры получается сеть.

**Контрольные вопросы**

1. Какие динамические структуры Вам известны?
2. В чем отличительная особенность динамических объектов?
3. Какой тип представлен для работы с динамическими объектами?
4. Как связаны элементы в динамической структуре ?
5. Назовите основные особенности односвязного списка..
6. В чем отличие линейных списков от кольцевых ?
7. Зачем были введены двусвязные списки ?
8. В чем разница в операциях, производимых над односвязными и двусвязными списками ?
9. Какой список является более удобным в обращении, односвязный или двусвязный ?
10. Что такое указатель?
11. Какие стековые операции можно производить над списками ?

12. Какие операции, производимые над очередью, можно производить над списками ?
13. Почему можно производить все эти операции над списками ?
14. Для чего предназначены операции Getnode и Freenode?
15. Какие методы утилизации вы знаете ?
16. Перечислите элементы заголовков в списках.
17. Зависит ли время, затраченное на вставку элемента в односвязный список, от количества элементов в списке ?
18. Где процесс вставки и удаления эффективнее, в списке или в массиве ?
19. Как можно производить просмотр односвязного списка?
20. Что означает AVAIL?
21. Какой недостаток односвязных списков по сравнению с
22. массивом?
23. Какие структуры являются нелинейными ?
24. Каковы признаки отличия нелинейных структур?
25. Как можно создать нелинейную связную структуру?
26. Что такое граф состояния ?

# 4. РЕКУРСИВНЫЕ СТРУКТУРЫ ДАННЫХ

Рассмотрим рекурсивные алгоритмы и рекурсивные структуры данных.

Рекурсия - процесс, протекание которого связано с обращением к самому себе (к этому же процессу).

Пример рекурсивной структуры данных - структура данных, элементы которой являются такими же структурами данных (рис. 4.1).

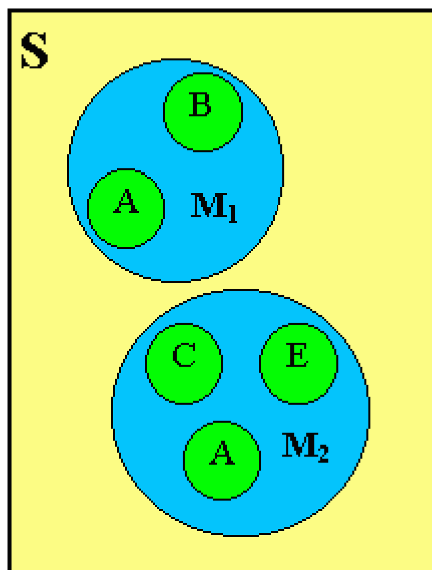



**Рис. 4.1**

## 4.1 Деревья

Дерево - нелинейная связанная структура данных (рис. 4.2).

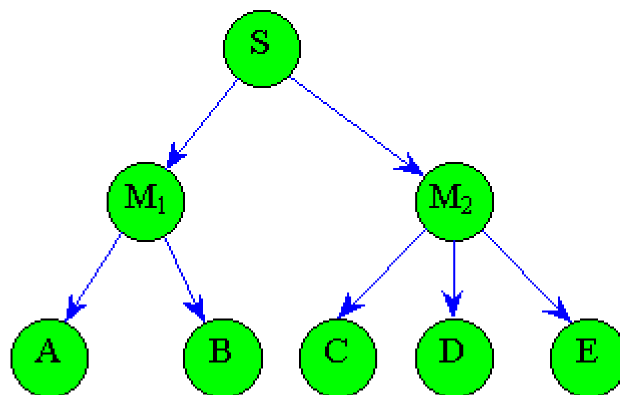



**Рис. 4.2**

Дерево характеризуется следующими признаками:

- дерево имеет 1 элемент, на которого нет ссылок от других элементов. Этот элемент называется корнем дерева;
- в дереве можно обратиться к любому элементу путем прохождения конечного числа ссылок (указателей);
- каждый элемент дерева связан только с одним предыдущим элементом, Любой узел дерева может быть промежуточным либо терминальным (листом). На рис. 4.2 промежуточными являются элементы M1, M2, листьями - A, B, C, D, E. Характерной особенностью терминального узла является отсутствие ветвей.

Высота - это количество уровней дерева. У дерева на рис. 4.2 высота равна двум.

Количество ветвей, растущих из узла дерева, называется степенью исхода узла (на рис. 4.2 для M1 степень исхода 2, для M2 - 3). По степени исхода классифицируются деревья:

1) если максимальная степень исхода равна m, то это - m-арное дерево;

2) если степень исхода равна либо 0, либо m, то это - полное m-арное дерево;

3) если максимальная степень исхода равна 2, то это - бинарное дерево;

4) если степень исхода равна либо 0, либо 2, то это - полное бинарное дерево.

Для описание связей между узлами дерева применяют также следующую терминологию: М1 - “отец” для элементов А и В. А и В - “сыновья” узла М1.

### 4.1.1 Представление деревьев

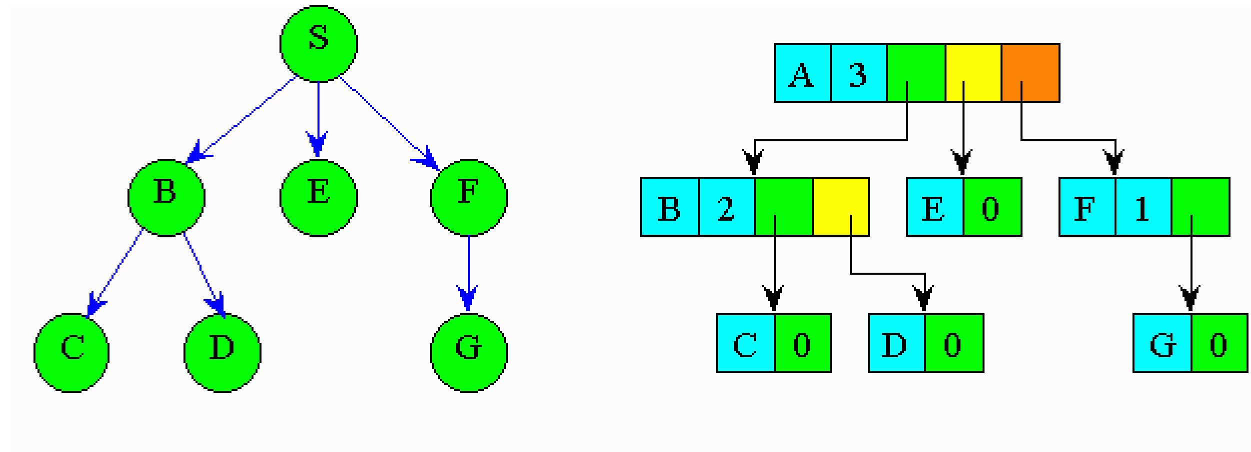



Рис. 4.3. Графическая форма представления дерева и форма представления в виде нелинейного списка.

Наиболее удобно деревья представлять в памяти ЭВМ в виде связанных списков. Элемент списка должен содержать информационные поля, в которых содержится значение узла и степень исхода, а также - поля-указатели, число которых равно степени исхода (рис4.3). То есть, любой указатель элемента ориентирует данный элемент-узел с сыновьями этого узла.

## 4.2 Бинарные деревья

Бинарные деревья являются наиболее используемой разновидностью деревьев.

Согласно представлению деревьев в памяти ЭВМ каждый элемент будет записью, содержащей 4 поля. Значения этих полей будут соответственно ключ записи, ссылка на

элемент влево-вниз, на элемент вправо-вниз и на текст записи.

При построении необходимо помнить, что левый сын имеет ключ меньший, чем у отца, а значение ключа правого сына больше значения ключа отца. Например, построим бинарное дерево из следующих элементов: 50, 46, 61, 48, 29, 55, 79. Оно имеет следующий вид:

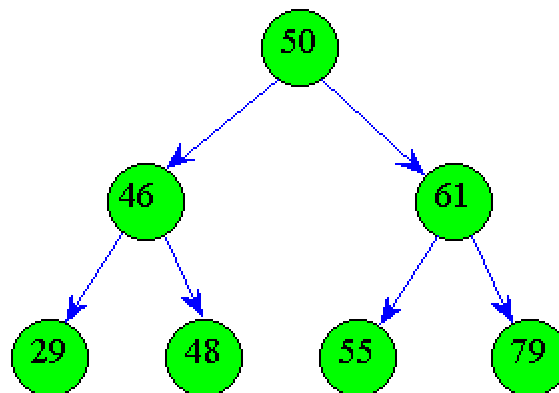



Рис. 4.4 .

Получили упорядоченное бинарное дерево с одинаковым числом уровней в левом и правом поддеревьях. Это - идеально сбалансированное дерево, то есть дерево, в котором левое и правое поддеревья имеют уровни, отличающиеся не более чем на единицу.

Для создания бинарного дерева надо создавать в памяти элементы типа (рис. 4.5):

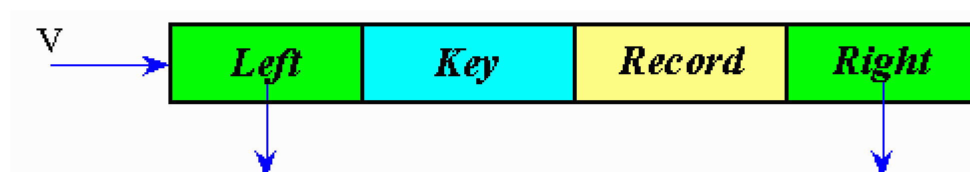



Рис. 4.5

Операция V = MakeTree(Key, Rec) - создает элемент (узел дерева) с двумя указателями и двумя полями (ключевым и информационным) .

Вид процедуры MakeTree:

| *Псевдокод* | *Паскаль* |
|---|---|
| p = getnode | New(p); |
| r(p) = rec | p^.r := rec; |
| k(p) = key | p^.k := key; |
| v = p | v := p; |
| left(p) = nil | p^.left := nil; |
| right(p) = nil | p^.right := nil; |

### 4.2.1 Сведение m-арного дерева к бинарному

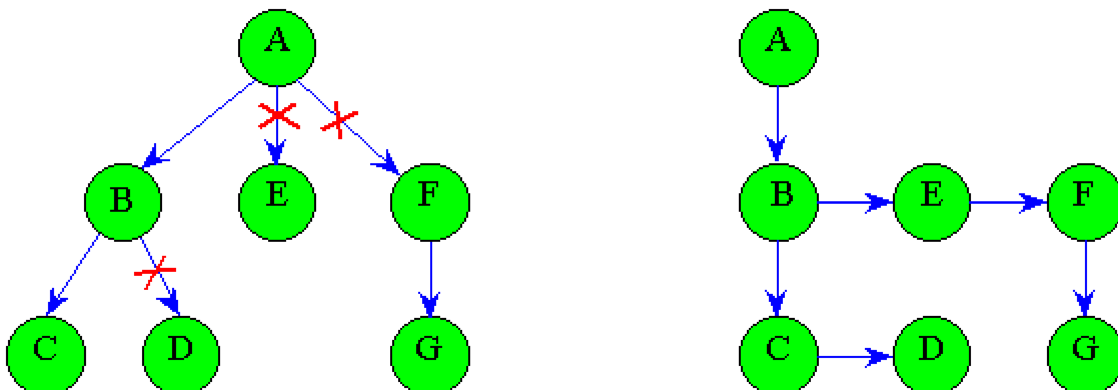
Неформальный алгоритм:

1.В любом узле дерева отсекаются все ветви, кроме крайней левой, соответствующей старшим сыновьям.

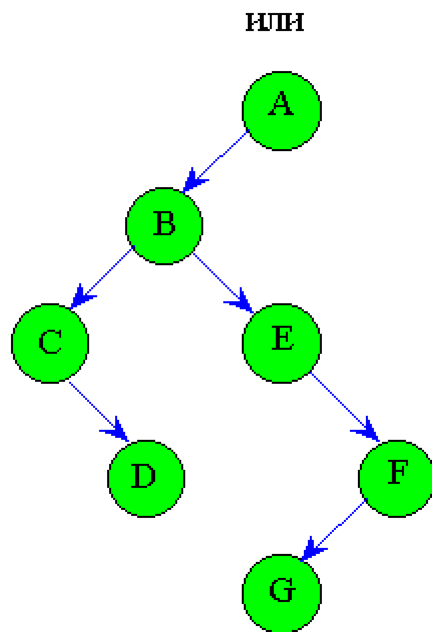
2.Соединяется горизонтальными линиями все сыновья одного родителя.

3. Старшим сыном в любом узле полученной структуры будет узел, находящийся под данным узлом (если он есть).

Последовательность действий алгоритма приведена на рис. 4.6.



или



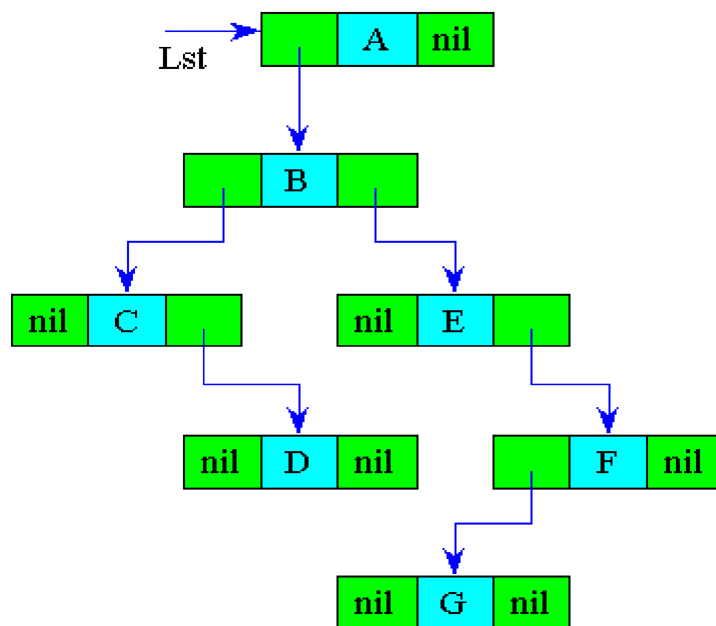




Рис. 4.6. Сведение m-арного дерева к бинарному.

### 4.2.2 Основные операции с деревьями

1. Обход дерева.
2. Удаление поддерева.
3. Вставка поддерева.

Для выполнения *обхода* дерева необходимо выполнить три процедуры:

1.Обработка корня.
2.Обработка левой ветви.
3.Обработка правой ветви.

В зависимости от того, в какой последовательности выполняются эти три процедуры, различают три вида обхода.

1.Обход сверху вниз. Процедуры выполняются в последовательности

1- 2 - 3.

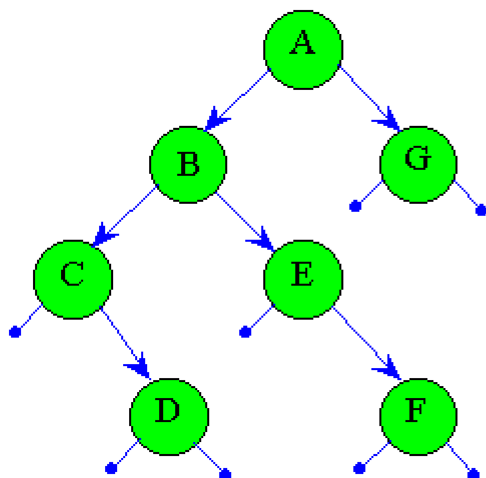
2.Обход слева направо. Процедуры выполняются в последовательности

2 - 1- 3.

3.Обход снизу вверх. Процедуры выполняются в последовательности

2 - 3 -1.

В зависимости от того, какой по счету заход в узел приводит к обработке узла, получается реализация одного из трех видов обхода. Если обработка идет после первого захода в узел, то сверху вниз, если после второго, то слева направо, если после третьего, то снизу вверх (см. рис. 4. 7).




Порядок обхода дерева:
A, B, C, D, E, F, G - сверху вниз;
C, D, B, E, F, A, G - слева направо;
D, C, F, E, B, G, A - снизу вверх.

Точно так же можно совершить обход , начав с отростка дерева .

Рис. 4.7. Различные направления обхода дерева.

*Операция исключения поддерева.* Необходимо указать узел, к которому подсоединяется исключаемое поддерево и индекс этого поддерева. Исключение поддерева состоит в том, что разрывается связь с исключаемым поддеревом, т. е. указатель элемента устанавливается в nil, а степень исхода данного узла уменьшается на единицу.

*Вставка поддерева* - операция, обратная исключению. Надо знать индекс включаемого поддерева, узел, к которому подвешивается дерево, установить указатель этого узла на корень поддерева, а степень исхода данного узла увеличивается на единицу. При этом в общем случае необходимо произвести перенумерацию сыновей узла, к которому подвешивается поддерево.

Алгоритм вставки и удаления рассмотрен в главе 5.

### 4.2.3 Алгоритм создания дерева бинарного поиска

Пусть заданы элементы с ключами: 14, 18, 6, 21, 1, 13, 15. После выполнения нижеприведенного алгоритма получится дерево, изображенное на рис.4.6. Если обойти полу-

ченное дерево слева направо, то получим упорядочивание: 1, 6, 13, 14, 15, 18, 21.

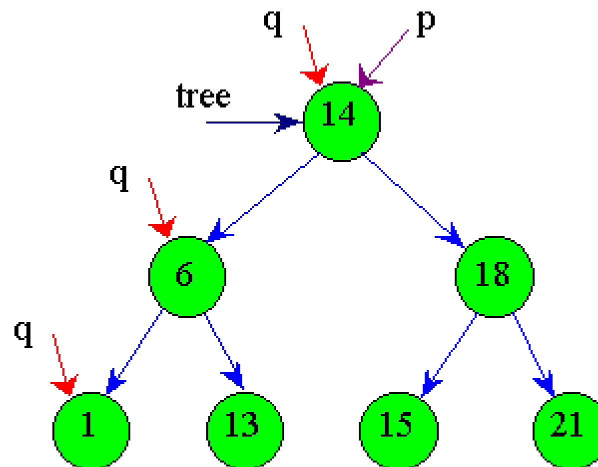



Рис. 4.8.

*Псевдокод*

```
read (key, rec)
tree = maketree(key rec)
p = tree
q = tree
while not eof do
  read (key, rec)
  v = maketree(key, rec)
  while p <> nil do
    q = p
    if key < k(p) then
      p = left(p)
    else
      p = right(p)
    endif
  endwhile
    if key < k(q) then
      left(p) = v
    else
      right(p) = v
```

*Паскаль*

```
read (key, rec);
tree := maketree(key rec);
p := tree;
q := tree;
while not eof do
  begin
    read (key, rec);
    v := maketree(key, rec);
    while p <> nil do
      begin
        q := p;
        if key < p^.k then
          p := p^.left
        else
          p := p^.right;
      end;
      if key < q^.k then
        p^.left := v
      else
```

```
  endif
   if q=tree then
       ' только корень'
   endif
return
```

```
        p^.right := v;
    end
     if q=tree
      then   writeln('Только   ко-
рень');
    exit
```

### 4.2.4 Прохождение бинарных деревьев

В зависимости от последовательности обхода поддеревьев различают три вида обхода (прохождения) деревьев (рис.4.9):

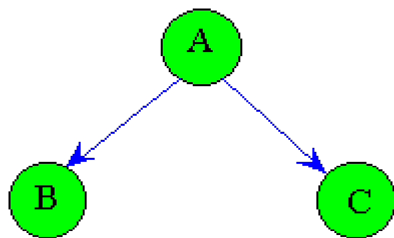



Рис. 4.9. Прохождение бинарных деревьев

1. Сверху вниз **A, B, C.**
2. Слева направо или симметричное прохождение **B, A, C.**
3. Снизу вверх **B, C, A.**

Наиболее часто применяется второй способ.

Ниже приведены рекурсивные алгоритмы прохождения бинарных деревьев.

```
subroutine pretrave (tree) 'сверху вниз
if tree <> nil then
  print info(tree)
  pretrave(left(tree))
```

```
Procedure pretrave (tree: tnode)
Begin
 if tree <> nil then
  begin
```

```
  pretrave(right(tree))
endif
return
```

```
      WriteLn(Tree^.Info);
      Pretrave(Tree^.left);
      Pretrave(Tree^.right);
    End;
end;
```

```
subroutine intrave (tree) 'симметричный
if tree <> nil then
  intrave(left(tree))
  print info(tree)
  intrave(right(tree))
endif
return
```

```
procedure intrave (tree: tnode)
begin
  if tree <> nil then
    begin
      intrave(Tree^.left);
      writeLn(Tree^.info);
       intrave(Tree^.right);
    end;
end;
```

**Обход дерева А, В, С**

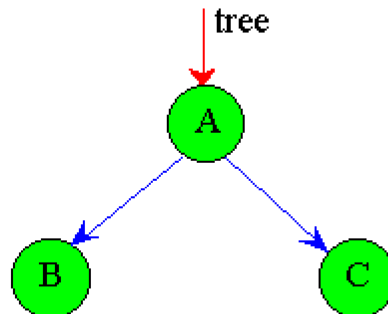



Рис. 4.10

Поясним подробнее рекурсию алгоритма обхода дерева слева направо.

Пронумеруем строки алгоритма ***intrave (tree):***

***1*** *if tree <> nil*
***2*** *then intrave (left(tree))*
***3*** *print info (tree)*
***4*** *intrave (right (tree))*
***5*** *endif*
***6*** *return*

Обозначим указатели: **t → tree; l → left; r → right**

На приведенном рис. 4.11 проиллюстрирована последовательность вызова процедуры ***intrave (tree)*** по мере обхода узлов простейшего дерева, изображенного на рис. 4. 10.

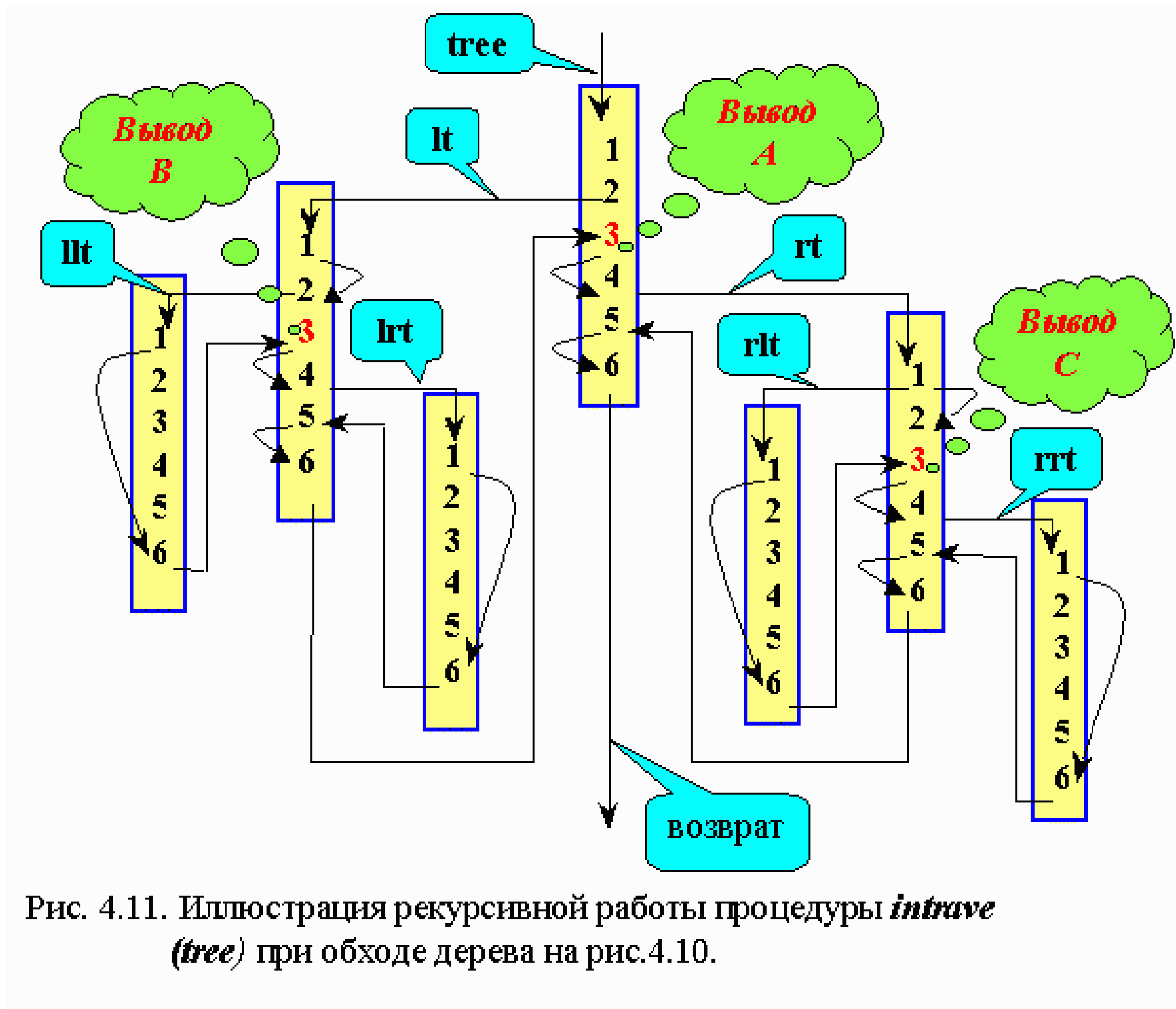



Рис. 4.11. Иллюстрация рекурсивной работы процедуры ***intrave (tree)*** при обходе дерева на рис.4.10.

# 5. ПОИСК

Поиск является одной из основных операций при обработке информации в ЭВМ. Ее назначение - по заданному аргументу найти среди массива данных те данные, которые соответствуют этому аргументу.

Набор данных (любых) будем называть таблицей или файлом. Любое данное (или элемент структуры) отличается каким-то признаком от других данных. Этот признак называется ключом. Ключ может быть уникальным, т. е. в таблице существует только одно данное с этим ключом. Такой уникальный ключ называется первичным. Вторичный ключ в одной таблице может повторяться, но по нему тоже можно организовать поиск. Ключи данных могут быть собраны в одном месте (в другой таблице) или представлять собой запись, в которой одно из полей - это ключ. Ключи, которые выделены из таблицы данных и организованы в свой файл, называются внешними ключами. Если ключ находится в записи, то он называется внутренним.

Поиском по заданному аргументу называется алгоритм, определяющий соответствие ключа с заданным аргументом. Результатом работы алгоритма поиска может быть нахождение этого данного или отсутствие его в таблице. В случае отсутствия данного возможны две операции:

1. индикация того, что данного нет
2. вставка данного в таблицу.

Пусть k - массив ключей. Для каждого k(i) существует r(i) - данное. Key - аргумент поиска. Ему соответствует информационная запись rec. В зависимости от того, какова структура данных в таблице, различают несколько видов поиска.

## 5.1 Последовательный поиск

Применяется в том случае, если неизвестна организация данных или данные неупорядочены. Тогда производится последовательный просмотр по всей таблице начиная от младшего адреса в оперативной памяти и кончая самым старшим.

Последовательный поиск в массиве (переменная search хранит номер найденного элемента).

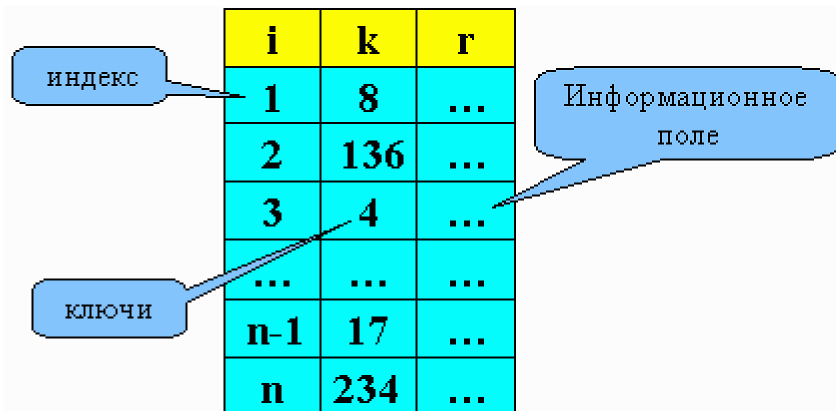


Рис. 5.1

```
for i:=1 to n
  if k(i) = key then
                       search = i
                        return
  endif
next i
search = 0
return
```

На Паскале программа будет выглядеть следующим образом:

```
for i:=1 to n do
  if k[i] = key then
    begin
      search = i;
      exit;
```

```
      end;
    search = 0;
    exit;
```

Эффективность последовательного поиска в массиве можно определить как количество производимых сравнений M. $M_{min} = 1$, $M_{max} = n$. Если данные расположены равновероятно во всех ячейках массива, то $M_{ср} \approx (n + 1)/2$.

Если элемент не найден в таблице и необходимо произвести вставку, то последние 2 оператора заменяются на

| | *на Паскале* |
|---|---|
| n = n + 1 | n:=n+1; |
| k(n) = key | k[n]:=key; |
| r(n) = rec | r[n]:=rec; |
| search = n | search:=n; |
| return | exit; |

Если таблица данных задана в виде односвязного списка, то производится последовательный поиск в списке (рис. 5.2).

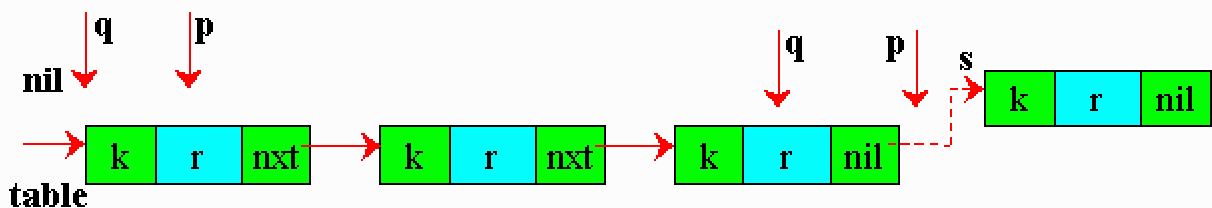



Рис. 5.2 Поиск в списке

Варианты алгоритмов:

| *На псевдокоде*: | *На Паскале*: |
|---|---|
| q=nil | q:=nil; |
| p=table | p:=table; |
| while (p <> nil) do | while (p <> nil) do |
| if k(p) = key then | begin |
| search = p | if p^.k = key then |
| return | begin |

```
  endif                                search = p;
  q = p                                exit;
  p = nxt(p)                         end;
endwhile                            q := p;
s = getnode                         p := p^.nxt;
k(s) = key                        end;
r(s) = rec                        New(s);
nxt(s) = nil                      s^.k:=key;
if q = nil then                   s^.r:=rec;
  table = s                       s.^nxt:= nil;
  else nxt(q) = s                 if q = nil then table = s
endif                                        else q.^nxt = s;
search = s                        search:= s;
return                            exit
```

Достоинством списковой структуры является ускоренный алгоритм удаления или вставки элемента в список, причем время вставки или удаления не зависит от количества элементов, а в массиве каждая вставка или удаление требуют передвижения примерно половины элементов. Эффективность поиска в списке примерно такая же, как и в массиве.

Эффективность последовательного поиска можно увеличить.

Пусть имеется возможность накапливать запросы за день, а ночью их обрабатывать. Когда запросы собраны, происходит их сортировка.

## 5.2.Индексно-последовательный поиск

При таком поиске организуется две таблицы: таблица данных со своими ключами (упорядоченная по возрастанию) и таблица индексов, которая тоже состоит из ключей данных, но эти ключи взяты из основной таблицы через определенный интервал (рис. 5.3).

Сначала производится последовательный поиск в таблице индексов по заданному аргументу поиска. Как только мы проходим ключ, который оказался меньше заданного, то этим мы устанавливаем нижнюю границу поиска в основной таблице - low, а затем - верхнюю - hi , на которой ( kind > key ).

Например, key = 101.

Поиск идет не по всей таблице, а от low  до hi.

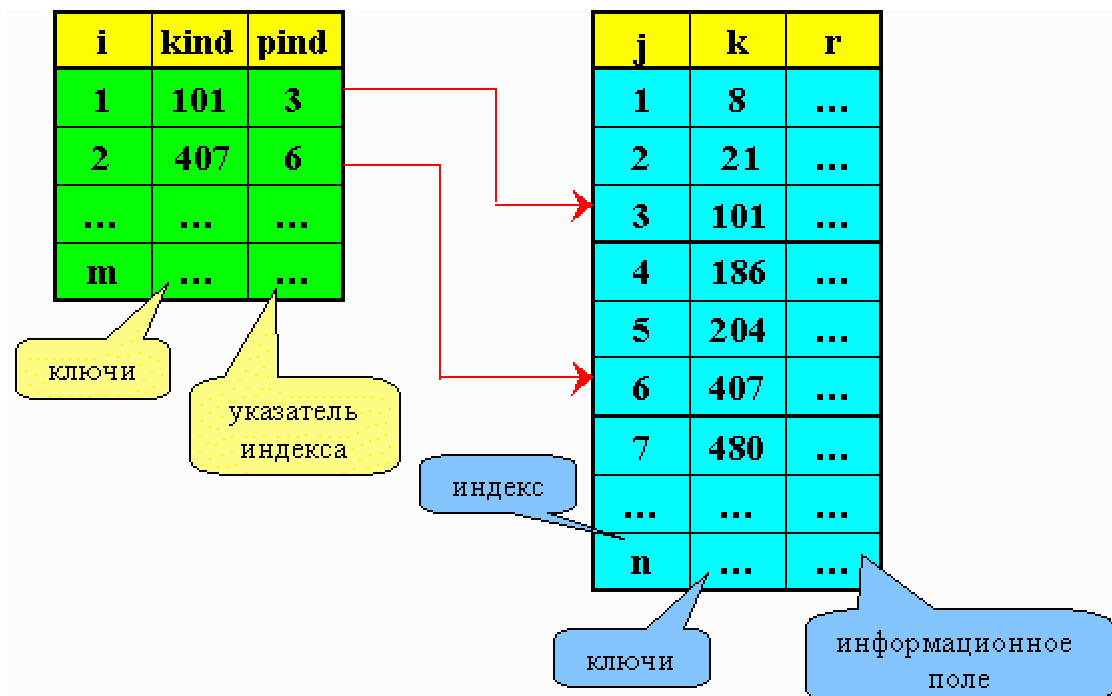



Рис. 5.3

Примеры программ:

*Псевдокод:*

```
i = 1
while (i <= m) and (kind(i) <= key) do
  i=i+1
endwhile
if i = 1 then low = 1
  else low = pind(i-1)
endif
```

*Паскаль:*

```
i:=1;
while (i <= m) and (kind[i] <= key) do
  i:=i+1;
if i = 1 then low:=1 else low:=pind[i-1];
if i = m+1 then hi:=n else hi:=pind[i]-1;
```

```
if i = m+1 then hi = n
  else hi = pind(i)-1
endif
for j=low to hi
  if key = k(j) then
    search=j
    return
  endif
next j
search=0
return
```

```
for j:=low to hi do
  if key = k[j] then
    begin
      search:=j;
      exit;
    end;
search:=0;
exit;
```

## 5.3. Эффективность последовательного поиска

Эффективность любого поиска может оцениваться по количеству сравнений $C$ аргумента поиска с ключами таблицы данных. Чем меньше количество сравнений, тем эффективнее алгоритм поиска.

Эффективность последовательного поиска в массиве:

$$C = 1 \div n,\ C = (n + 1)/2.$$

Эффективность последовательного поиска в списке - то же самое. Хотя по количеству сравнений поиск в связанном списке имеет ту же эффективность, что и поиск в массиве, организация данных в виде массива и списка имеет свои достоинства и недостатки. Целью поиска является выполнение следующих процедур:

1) Найденную запись считать.
2) При отсутствии записи произвести ее вставку в таблицу.
3) Найденную запись удалить.

Первая процедура (собственно поиск) занимает для них одно время. Вторая и третья процедуры для списочной струк-

туры более эффективна (т. к. у массивов надо сдвигать элементы).

Если k - число передвижений элементов в массиве, то $k = (n + 1)/2$.

### 5.4. Эффективность индексно-последовательного поиска

Если считать равновероятным появление всех случаев,то эффективность поиска можно рассчитать следующим образом:

Введем обозначения: $m$ - размер индекса; $m = n / p$;
$p$ - размер шага

$$Q = (m+1)/2 + (p+1)/2 = (n/p+1)/2 + (p+1)/2 = n/2p+p/2+1$$

Продифференцируем Q по p и приравняем производную нулю:

$$dQ/dp=(d/dp)\ (n/2p+p/2+1)= - n / 2\, p^2 + 1/2 = 0$$

Отсюда

$$p^2=n\ ;$$

$$p_{опт} = \sqrt{n}$$

Подставив $p_{опт}$ в выражение для Q, получим следующее количество сравнений:

$$Q = \sqrt{n} + 1$$

Порядок эффективности индексно-последовательного поиска $\boldsymbol{O}\ (\sqrt{\boldsymbol{n}}\ )$

## 5.5 Методы оптимизации поиска

Всегда можно говорить о некотором значении вероятности поиска того или иного элемента в таблице. Считаем, что в таблице данный элемент существует. Тогда вся таблица поиска может быть представлена как система с дискретными состояниями, а вероятность нахождения там искомого элемента - это вероятность *p(i) i* - го состояния системы.

$$\sum_{i=1}^{n} p(i) = 1$$

Количество сравнений при поиске в таблице, представленной как дискретная система, представляет собой математическое ожидание значения дискретной случайной величины, определяемой вероятностями состояний и номерами состояний системы.

$$Z=Q=1p(1)+2p(2)+3p(3)+\ldots+np(n)$$

Желательно, чтобы $p(1)\geq p(2) \geq p(3) \geq\ldots\geq p(n)$.

Это минимизирует количество сравнений, то есть увеличивает эффективность. Так как последовательный поиск начинается с первого элемента, то на это место надо поставить элемент, к которому чаще всего обращаются (с наибольшей вероятностью поиска).

Наиболее часто используются два основных способа автономного переупорядочивания таблиц поиска. Рассмотрим их на примере односвязных списков.

### 5.5.1 Переупорядочивание таблицы поиска путем перестановки найденного элемента в начало списка

Суть этого метода заключается в том, что элемент списка с ключом, равным заданному, становится первым элементом списка, а все остальные сдвигаются.

Рис. 5.4 . Перестановка первого и второго элементов списка .

Этот алгоритм применим как для списков, так и для массивов. Однако не рекомендуется применять его для массивов, так как на перестановку элементов массива затрачивается гораздо больше времени, чем на перестановку указателей.

Алгоритм переупорядочивания в начало списка:

*Псевдокод:*

```
q=nil
p=table
while (p <> nil) do
  if key = k(p) then
      if q = nil then    'перестановка не нужна'
              search = p
           return
      endif
    nxt(q) = nxt(p)
    nxt(p) = table
    table = p
    search = p
```

*Паскаль:*

```
q:=nil;
p:=table;
while (p <> nil) do
  begin
    if key = p^.k then
      begin
       if q = nil
             then 'перестановка не нужна'
          search := p;
          exit;
          end;
          q^.nxt := p^.nxt;
```

```
    return
  endif
  q = p
  p = nxt(p)
endwhile
search = nil
return
```

```
      p^.nxt := table;
      table := p;
      exit;
    end;
  q := p;
  p := p^.nxt;
 end;
search := nil;
exit;
```

### 5.5.2. Метод транспозиции

В данном методе найденный элемент переставляется на один элемент к голове списка. И если к этому элементу обращаются часто, то, перемещаясь к голове списка, он скоро окажется на первом месте.

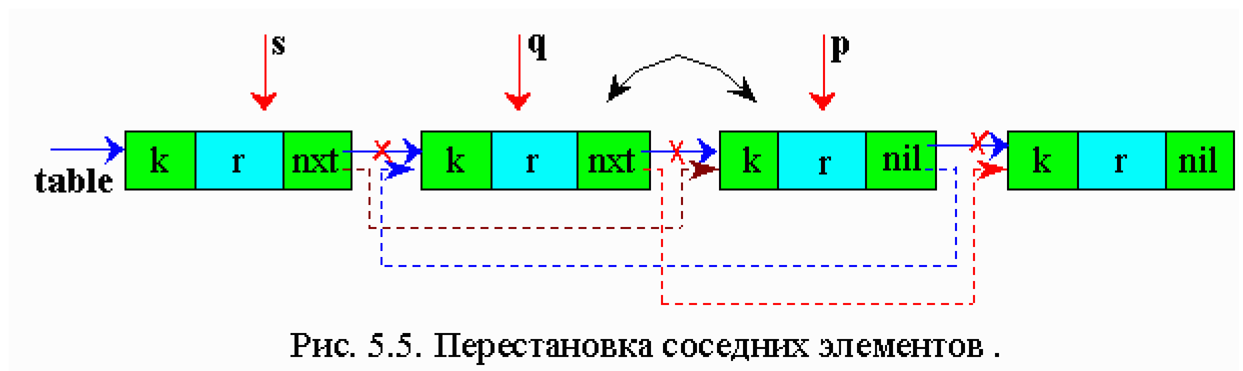



Рис. 5.5. Перестановка соседних элементов .

p - рабочий указатель
q - вспомогательный указатель, отстает на один шаг от p
s - вспомогательный указатель, отстает на два шага от q

Алгоритм метода транспозиции:

*Псевдокод:*

```
s=nil
q=nil
p=table
```

*Паскаль:*

```
s:=nil;
q:=nil;
p:=table;
```

```
while (p <> nil) do
  if key = k(p) then
    ‘ нашли, транспонируем
    if q = nil then
      ‘ переставлять не надо
      search=p
      return
    endif
    nxt(q)=nxt(p)
    nxt(p)=q
    if s = nil then
      table = p
      else nxt(s)=p
    endif
    search=p
    return
  endif
endwhile
search=nil
return
```

```
while (p <> nil) do
  begin
    if key = p^.k then
      ‘ нашли, транспонируем
      begin
        if q = nil then
          begin
          ‘переставлять не на-
            до  search:=p;
          exit;
        end;
        q^.nxt:=p^.nxt;
        p^.nxt:=q;
        if s = nil then
          table := p;
        else
          begin
            s^.nxt := p;
        end;
        search:=p;
        exit;
  end;
end;
search:=nil;
exit;
```

Этот метод удобен при поиске не только в списках, но и в массивах (так как меняются только два стоящих рядом элемента).

### 5.5.3. Дерево оптимального поиска

Если извлекаемые элементы сформировали некоторое постоянное множество, то может быть выгодным настроить

дерево бинарного поиска для большей эффективности последующего поиска.

Рассмотрим деревья бинарного поиска, приведенные на рисунках a и b.Оба дерева содержат три элемента - *к1, к2, к3,* где *к1<к2<к3*. Поиск элемента *к3* требует двух сравнений для рисунка 5.6 а), и только одного - для рисунка 5.6 б).

Число сравнений ключей, которые необходимо сделать для извлечения некоторой записи, равно уровню этой записи в дереве бинарного поиска плюс 1.

Предположим, что:

*p1* - вероятность того, что аргумент поиска *key = к1*

*p2* - вероятность того, что аргумент поиска *key = к2*

*p3* - вероятность того, что аргумент поиска *key = к3*

*q0* - вероятность того, что *key < к1*

*q1* - вероятность того, что *к2 > key > к1*

*q2* - вероятность того, что *к3 > key > к2*

*q3* - вероятность того, что *key > к3*

*C1* - число сравнений в первом дереве рисунка 5.6 а)

*C2* - число сравнений во втором дереве рисунка 5.6 б)

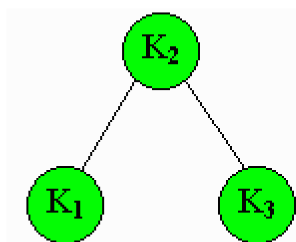



Рис. 5.6 а)

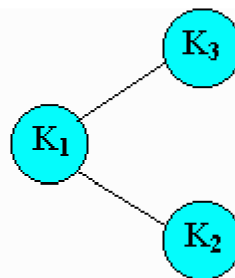



Рис. 5.6 б)

Тогда $p1+p2+p3+q0+q1+q2+q3 = 1$

Ожидаемое число сравнений в некотором поиске есть сумма произведений вероятности того, что данный аргумент имеет некоторое заданное значение, на число сравнений, необходимых для извлечения этого значения, где сумма берется по всем возможным значениям аргумента поиска. Поэтому

$$C1 = 2 \cdot p1 + 1 \cdot p2 + 2 \cdot p3 + 2 \cdot q0 + 2 \cdot q1 + 2 \cdot q2 + 2 \cdot q3$$
$$C2 = 2 \cdot p1 + 3 \cdot p2 + 1 \cdot p3 + 2 \cdot q0 + 3 \cdot q1 + 3 \cdot q2 + 1 \cdot q3$$

Это ожидаемое число сравнений может быть использовано как некоторая мера того, насколько "хорошо" конкретное дерево бинарного поиска подходит для некоторого данного множества ключей и некоторого заданного множества вероятностей. Так, для вероятностей, приведенных далее слева, дерево из а) является более эффективным, а для вероятностей, приведенных справа, дерево из б) является более эффективным:

| | |
|---|---|
| $P1 = 0.1$ | $P1 = 0.1$ |
| $P2 = 0.3$ | $P2 = 0.1$ |
| $P3 = 0.1$ | $P3 = 0.3$ |
| $q0 = 0.1$ | $q0 = 0.1$ |
| $q1 = 0.2$ | $q1 = 0.1$ |
| $q2 = 0.1$ | $q2 = 0.1$ |
| $q3 = 0.1$ | $q3 = 0.2$ |
| $C1 = 1.7$ | $C1 = 1.9$ |
| $C2 = 2.4$ | $C2 = 1.8$ |

Дерево бинарного поиска, которое минимизирует ожидаемое число сравнений некоторого заданного множества ключей и вероятностей, называется оптимальным. Хотя алгоритм создания дерева может быть очень трудоемким, дерево, которое он создает, будет работать эффективно во всех последующих поисках. К сожалению, однако, заранее вероятности аргументов поиска редко известны.

## 5.6 Бинарный поиск (метод деления пополам)

Будем предполагать, что имеем упорядоченный по возрастанию массив чисел. Основная идея - выбрать случайно некоторый элемент $A_M$ и сравнить его с аргументом поиска

X. Если $A_M$=X, то поиск закончен; если $A_M$ <X, то мы заключаем, что все элементы с индексами, меньшими или равными M, можно исключить из дальнейшего поиска. Аналогично, если $A_M$ >X.

Выбор M совершенно произволен в том смысле, что корректность алгоритма от него не зависит. Однако на его эффективность выбор влияет. Ясно, что наша задача- исключить как можно больше элементов из дальнейшего поиска. Оптимальным решением будет выбор среднего элемента, т.е. середины массива.

Рассмотрим пример, представленный на рис. 5.7. Допустим нам необходимо найти элемент с ключом 52. Первым сравниваемым элементом будет 49. Так как 49<52, то ищем следующую середину среди элементов, расположенных выше 49. Это число 86. 86>52, поэтому теперь ищем 52 среди элементов, расположенных ниже 86, но выше 49. На следующем шаге обнаруживаем, что очередное значение середины равно 52. Мы нашли элемент в массиве с заданным ключом.

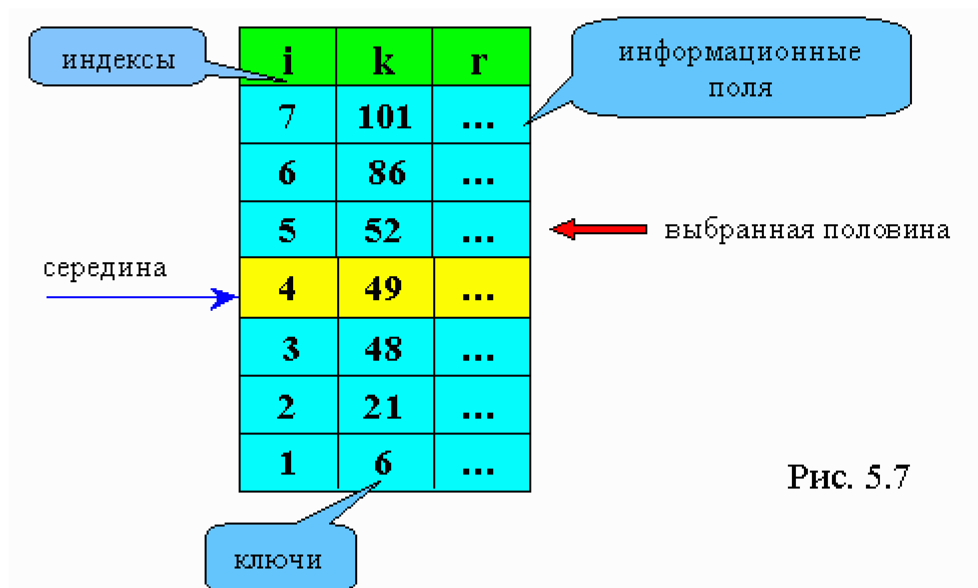



Рис. 5.7

Программы на псевдокоде и Паскале:

```
Low = 1
hi = n
while (low <= hi) do
  mid = (low + hi) div 2
  if key = k(mid) then
    search = mid
    return
  endif
  if key < k(mid) then
    hi = mid - 1
    else low = mid + 1
  endif
endwhile
search = 0
return
```

```
low := 1;
hi := n;
while (low <= hi) do
  begin
    mid := (low + hi) div 2;
    if key = k[mid] then
      begin
        search := mid;
        exit;
      end;
    if key < k[mid]
      then hi := mid - 1
      else low := mid + 1;
  end;
search := 0;
exit
```

При key = 101 запись будет найдена за три сравнения (в последовательном поиске понадобилось бы семь сравнений).

Если $C$ - количество сравнений, а $n$ - число элементов в таблице, то

$C = log_2 n.$

Например, $n = 1024$.

При последовательном поиске $C = 512$, а при бинарном $C = 10$.

Можно совместить бинарный и индексно-последовательный поиск (при больших объемах данных), учитывая, что последний также используется при отсортированном массиве.

## 5.7. Поиск по бинарному дереву

Назначение его в том, чтобы по заданному ключу осуществить поиск узла дерева. Длительность операции зависит от структуры дерева. Действительно, дерево может быть вырождено в однонаправленный список, как правое на рис. 5.8.

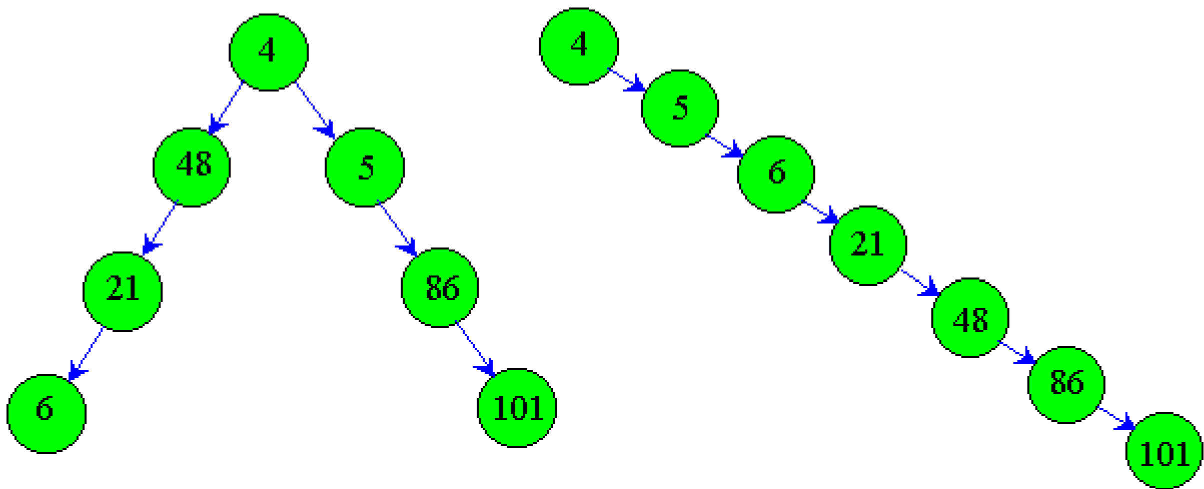



Рис. 5.8. Бинарные деревья.

В этом случае время поиска будет такое же, как и в однонаправленном списке, т.е. придется перебирать N/2 элементов.

Наибольшего эффекта использования дерева достигается в том случае, когда дерево сбалансировано.В этом случае для поиска придотся перебрать не больше $log_2N$ элементов.

Строго сбалансированное дерево - это дерево, в котором каждый узел имеет левое и правое поддеревья, отличающиеся по уровню не более, чем на единицу.

Поиск элемента в бинарном дереве называется бинарным поиском по дереву.

Такое дерево называют деревом бинарного поиска.

Суть поиска заключается в следующем. Анализируем вершину очередного поддерева. Если ключ меньше инфор-

мационного поля вершины, то анализируем левое поддерево, больше - правое.

Пусть задан аргумент key

```
p = tree
whlie p <> nil do
 if key = k(p) then
  search = p
  return
 endif
 if key < k(p) then
  p = left(p)
 else
  p = right(p)
 endif
endwhile
search = nil
return
```

```
p := tree;
whlie p <> nil do
 begin
  if key = p^.k then
   begin
    search := p;
    exit;
   end;
  if key < p^.k then
   p := p^.left
  else
   p := p^.right;
 end;
search := nil;
```

## 5.8 Поиск со вставкой (с включением)

Для вставки элемента в дерево, сначало нужно осуществить поиск в дереве по заданному ключу. Если такой ключ имеется, то программа завершается, если нет, то происходит вставка.

Для включения новой записи в дерево, прежде всего нужно найти тот узел, к которому можно присоединить новый элемент. Алгоритм поиска нужного узла тот же самый, что и при поиске узла с заданным ключом. Однако непосредственно использовать процедуру поиска нельзя, потому что по ее окончании не фиксирует тот узел, из которого была выбрана ссылка NIL (search = nil).

Модифицируем описание процедуры поиска так, чтобы в качестве ее побочного эффекта фиксировалась ссылка на узел, в котором был найден заданный ключ (в случае успеш-

ного поиска), или ссылка на узел, после обработки которого поиск прекращается (в случае неуспешного поиска).

*Псевдокод:*

```
P = tree
Q = nil
Whlie p <> nil do
  q = p
  if key = k(p) then
    search = p
    return
  endif
 if key < k(p) then
    p = left(p)
  else
    p = right(p)
  endif
endwhile
v = maketree(key, rec)
if q = nil then
  tree = v
 else
  if key < k(q) then
    left(q) = v
   else
    right(q) = v
  endif
endif
   search = v
   return
```

*Паскаль:*

```
p := tree;
q := nil;
whlie p <> nil do
  begin
    q := p;
    if key = p^.k then
      begin
        search := p;
        exit;
      end;
    if key < p^.k then
      p := p^.left
    else
      p := p^.right;
  end;
          v :=
maketree(key, rec)
  if q=nil then
    tree:=v
   else
    if key < q^.k then
     q^.left := v
    else
     q^.right := v;
    search := v;
```

## 5.9 Поиск с удалением

Удаление узла не должно нарушить упорядоченности дерева. Возможны три варианта:

1) Найденный узел является листом. Тогда он просто удаляется и все.

2) Найденный узел имеет только одного сына. Тогда сын перемещается на место отца.

3) У удаляемого узла два сына. В этом случае нужно найти подходящее звено поддерева, которое можно было бы вставить на место удаляемого. Такое звено всегда существует:

- это либо самый правый элемент левого поддерева (для достижения этого звена необходимо перейти в следующую вершину по левой ветви, а затем переходить в очередные вершины только по правой ветви до тех пор, пока очередная ссылка не будет равна NIL.

- либо самый левый элемент правого поддерева (для достижения этого звена необходимо перейти в следующую вершину по правой ветви, а затем переходить в очередные вершины только по левой ветви до тех пор, пока очередная ссылка не будет равна NIL

Предшественником удаляемого узла называют самый правый узел левого поддерева (для 12 - 11). Преемником - самый левый узел правого поддерева (для 12 - 13).

Будем разрабатывать алгоритм для преемника (рис.5.9).

p - рабочий указатель;

q - отстает от p на один шаг;

v - указывает на приемника удаляемого узла;

t - отстает от v на один шаг;

s - на один шаг впереди v (указывает на левого сына или пустое место).

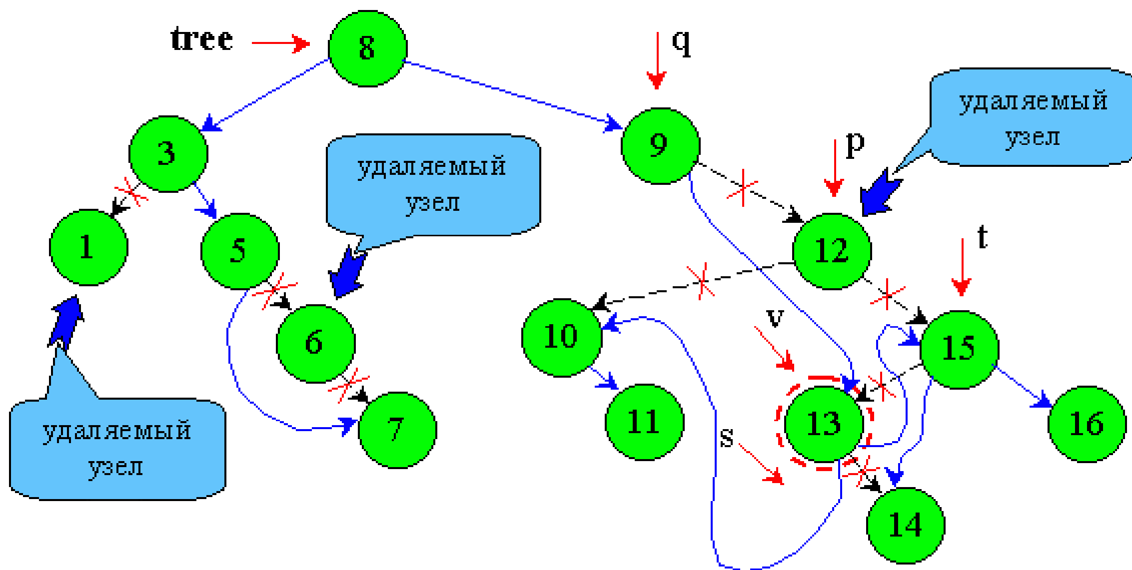



Рис. 5.9. Удаление узлов дерева

На узел 13 в конечном итоге должен указывать v, а указатель s - на пустое место (как показано на рис. 5.9).

*Псевдокод:*

```
 q = nil
 p = tree
while (p <> nil) and (k(p) <> key)
do
  q = p
  if key < k(p) then
    p = left(p)
  else
    p = right(p)
  endif
endwhile
if p = nil then 'Ключ не найден'
                      return
endif
if left(p) = nil then  v = right(p)
                     else if right(p) = nil
                          then v = left(p)
```

*Паскаль:*

```
 q := nil;
  p := tree;
while (p <> nil) and (p^.k <>
key) do
  begin
    q := p;
    if key < p^.k then
      p := p^.left
    else
p := p^.right;
  end;
if p = nil then
  WriteLn('Ключ не
найден');
               exit;
end;
if p^.left = nil then
```

```
                  else
'У nod(p) - два сына'
'Введем два указателя (t отстает
от v на 1 'шаг, s - опережает)
     t = p
     v = right(p)
     s = left(v)
     while s <> nil do
       t = v
       v = s
       s = left(v)
     endwhile
     if t <> p then
       'v не является сыном p'
       left(t) = right(v)
       right(v) = right(p)
     endif
     left(v) = left(p)
  endif
endif
if q = nil then 'p - корень'
                 tree = v
          else  if p = left(q)
                  then left(q) = v
                  else right(q) = v
                 endif
endif
freenode(p)
return
```

```
  v := p^.right
else
  if p^.right = nil then
    v := p^.left
  else
    begin
{Введем два указателя (t
отстает от v на 1 шаг, s -
опережает)}
      t := p;
      v := p^.right;
      s := v^.left;
      while s <> nil do
        begin
          t := v;
          v := s;
          s := v^.left;
        end;
      if t <> p then
        begin
    WriteLn('v не является
сыном p');
          t^.left := v^.right;
          v^.right := p^.right;
        end;
      v^.left := p^.left;
      end;
      end;
      if p = q^.left then
        q^.left := v
      else
        q^.right := v;
     end;
dispose(p);
  end;
```

**Контрольные вопросы**

1. В чем состоит назначение поиска ?
2. Что такое уникальный ключ ?
3. Какая операция производится в случае отсутствия заданного ключа в списке ?
4. В чем разница между последовательным и индексно-последовательным поиском ?
5. Какой из них более эффективный и почему ?
6. Какие способы переупорядочивания таблицы вы знаете ?
7. Основные отличия метода перестановки в начало от метода транспозиции .
8. Где они будут работать быстрее, в массиве или списке ?
9. В каких списках они работают, упорядоченных или нет ?
10. В чем суть бинарного поиска?
11. Как можно обойти бинарное дерево?
12. Можно ли применять бинарный поиск к массивам ?
13. Если удалить корень в непустом бинарном дереве, какой элемент станет на его место ?

# 6. СОРТИРОВКА

При обработке данных важно знать информационное поле данных и размещение их в машине.

Различают внутреннюю и внешнюю сортировку:

- внутренняя сортировка - сортировка в оперативной памяти;

- внешняя сортировка - сортировка во внешней памяти.

Сортировка - это расположение данных в памяти в регулярном виде по их ключам. Регулярность рассматривают как возрастание (убывание) значения ключа от начала к концу в массиве.

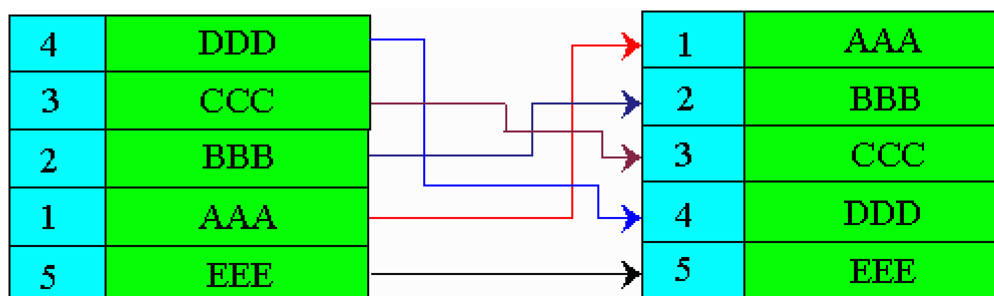



Рис. 6.1. Сортировка.

Если сортируемые записи занимают большой объем памяти, то их перемещение требует больших затрат. Для того, чтобы их уменьшить, сортировку производят в таблице адресов ключей, делают перестановку указателей, т.е. сам массив не перемещается. Это метод сортировки таблицы адресов.

При сортировке могут встретиться одинаковые ключи. В этом случае при сортировке желательно расположить после сортировки одинаковые ключи в том же порядке, что и в исходном файле. Это - устойчивая сортировка.

Эффективность сортировки можно рассматривать с нескольких критериев:

– время, затрачиваемое на сортировку;

– объем оперативной памяти, требуемой для сортировки;

– время, затраченное программистом на написание программы.

Выделяем первый критерий, поскольку будем рассматривать только методы сортировки «на том же месте», то есть не резервируя для процесса сортировки дополнительную память. Эквивалентом затраченного на сортировку времени можно считать количество сравнений при выполнении сортировки и количество перемещений.

Порядок числа сравнения при сортировке лежит в пределах от O (n log n) до O ($n^2$); O (n) - идеальный и недостижимый случай.

Различают следующие методы сортировки:

– строгие (прямые) методы;

– улучшенные методы.

Строгие методы:

1) метод прямого включения;
2) метод прямого выбора;
3) метод прямого обмена.

Эффективность этих трех методов примерно одинакова.

## 6.1. Сортировка методом прямого включения

Такой метод широко используется при игре в карты. Элементы мысленно делятся на уже готовую последовательность $a_1,...,a_{i-1}$ и исходную последовательность. При каждом шаге, начиная с i = 2 и увеличивая i каждый раз на единицу, из исхожной последовательности извлекается i-й элемент и перекладывается в готовую последовательность, при этом он вставляется на нужное место.На рис. 6.2 показан в качестве примера процесс сортировки с помощью включения шести случайно выбранных чисел. Алгоритм этой сортировки таков:

```
for i = 2 to n
  x = a(i)
  находим место среди a(1)…a(i) для включения x
next i
```

| i = 2 | 4 | 5 | 3 | 8 | 1 | 7 |
|---|---|---|---|---|---|---|
| i = 3 | 4 | 3 | 5 | 8 | 1 | 7 |
| | 3 | 4 | 5 | 8 | 1 | 7 |
| i = 4 | 3 | 4 | 5 | 8 | 1 | 7 |
| i = 5 | 1 | 3 | 4 | 5 | 8 | 7 |
| i = 6 | 1 | 3 | 4 | 5 | 7 | 8 |

Рис. 6.2.Сортировка методом прямого включения .

Для сортировки методом прямого включения пользуются следующими приемами:

*Псевдокод:*

*Без барьера*:

```
for i = 2 to n
    x = a(i)
    for j = i - 1 downto 1
      if x < a(j )
         then  a( j + 1) = a(j )
         else  go to  L
      endif
     next j
 L:  a( j + 1) = x
 next i
 return
```

*Паскаль:*

*Без барьера*:

```
for i:= 2 to n do
  begin
    x:= a(i);
    for j:= i - 1downto 1 do
      if x < a(j ) then
         a(j +1):= a(j )
       else  goto  1;
      end;
     end;
1:   a(j + 1):= x;
  end;
 end;
```

*С барьером*:

```
for i = 2 to n
```

*С барьером*:

```
  for i := 2 to n do
```

```
  x = a(i)
  a(0) = x {a(0) - барьер}
  j = i - 1
  while  x < a(j )  do
    a( j +1) = a(j )
    j = j - 1
  endwhile
  a( j +1) = x
next i
return
```

```
begin
  x:= a(i);
  a(0):=  x;  {a(0)  - барьер}
  j:= i - 1;
  while  x < a(j ) do
    begin
      a(j +1):= a(j );
      j:=j - 1;
    end;
  a(j +1):= x;
end;
```

***Эффективность алгоритма***

Число сравнений ключей Ci при i- м просеивании самое большее равно i-1, самое меньшее - 1; если предположить, что все перестановки из N ключей равновероятны, то среднее число сравнений = i/2. Число же пересылок Mi=Ci+3 (включая барьер). Минимальные оценки встречаются в случае уже упорядоченной исходной последовательности элементов, наихудшие же оценки - когда они первоначально расположены в обратном порядке. В некотором смысле сортировка с помощью включения демонстрирует истинно естественное поведение. Ясно, что приведенный алгоритм описывает процесс устойчивой сортировки: порядок элементов с равными ключами при нем остается неизменным.

Количество сравнений в худшем случае, когда массив отсортирован противоположным образом, $C_{max} = n(n - 1)/2$, т. е. - O $(n^2)$. Количество перестановок $M_{max} = C_{max} + 3(n-1)$, т.е. - O $(n^2)$. Если же массив уже отсортирован, то число сравнений и перестановок минимально: $C_{min} = n-1$; $M_{min} = =3(n-1)$.

## 6.2 Сортировка методом прямого выбора

Этот прием основан на следующих принципах:

1. Выбирается элемент с наименьшим ключом.
2. Он меняется местами с первым элементом $a_1$.
3. Затем этот процесс повторяется с оставшимися n-1 элементами, n-2 элементами и т.д. до тех пор, пока не останется один, самый "большой" элемент.

Алгоритм формулируется так:

```
for i = 1 to n - 1
  x = a(i)
  k = i
  for j = i + 1 to n
    if  a(j) < x  then
      k = j
      x = a(k)
    endif
  next j
  a(k) = a(i)
  a(i) = x
next i
return
```

```
for i := 1 to n - 1 do
  begin
    x := a[i];
    k := i;
    for j := i + 1 to n do
      if a[j] < x then
        begin
          k := j;
          x := a[k];
        end;
    a[k] := a[i];
    a[i] := x;
  end;
```

***Эффективность алгоритма:***

Число сравнений ключей C, очевидно, не зависит от начального порядка ключей. Можно сказать, что в этом смысле поведение этого метода менее естественно, чем поведение прямого включения. Для C при любом расположении ключей имеем:

$C = n(n-1)/2$

Порядок числа сравнений, таким образом, $O(n^2)$

Число перестановок минимально $M_{min} = 3(n - 1)$ в случае изначально упорядоченных ключей и максимально, $M_{max}$ =

$3(n - 1) + C$, т.е. порядок $O(n^2)$, если первоначально ключи располагались в обратном порядке.

В худшем случае сортировка прямым выбором дает порядок $n^2$, как для числа сравнений, так и для числа перемещений.

## 6.3. Сортировка с помощью прямого обмена (пузырьковая сортировка)

Классификация методов сортировки редко бывает осмысленной. Оба разбиравшихся до этого метода можно тоже рассматривать как "обменные" сортировки. В данном же, однако, разделе описан метод, где обмен местами двух элементов представляет собой характернейшую особенность процесса. Изложенный ниже алгоритм прямого обмена основывается на сравнении и смене мест для пары соседних элементов и продолжении этого процесса до тех пор, пока не будут упорядочены все элементы.

Как и в упоминавшемся методе прямого выбора, мы повторяем проходы по массиву, сдвигая каждый раз наименьший элемент оставшейся последовательности к левому концу массива. Если мы будем рассматривать массивы как вертикальные, а не горизонтальные построения, то элементы можно интерпретировать как пузырьки в чане с водой, причем вес каждого соответствует его ключу. В этом случае при каждом проходе один пузырек как бы поднимается до уровня, соответствующего его весу (см. рис.6.3).

Такой метод широко известен под именем "пузырьковая сортировка". В своем простейшем виде он представлен ниже.

Номер прохода

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 4 | 1 | 1 | 1 | 1 |
| 3 | 4 | 2 | 2 | 2 |
| 7 | 3 | 4 | 3 | 3 |
| 2 | 7 | 3 | 4 | 4 |
| 1 | 2 | 7 | 5 | 5 |
| 6 | 5 | 5 | 6 | 6 |
| 5 | 6 | 6 | 7 | 7 |

Рис. 6.3.

Программы на псевдокоде и Паскале:

```
for i = 2 to n
 for j = n to i step -1
  if a(j) < a(j - 1) then
   x = a(j - 1)
   a(j - 1) = a(j)
   a(j) = x
  endif
 next j
next i
return
```

```
for i := 2 to n do
 for j := n downto i do
  if a[j] < a[j - 1]then
   begin
    x := a[j - 1];
    a[j - 1] := a[j];
    a[j] := x;
   end;
 end;
end;
```

В нашем случае получился один проход “вхолостую”. Чтобы лишний раз не переставлять элементы, можно ввести флажок *fl*, который остается в значении *false* , если при очередном проходе не будет произведено ни одного обмена. На нижеприведенном алгоритме добавления отмечены курсивом.

```
fl = true
for i = 2 to n
 if fl = false then return
 endif
 fl = false
```

```
      for j = n to i step -1
        if a(j) < a(j - 1) then
         fl = true
          x = a(j - 1)
          a(j - 1) = a(j)
          a(j) = x
        endif
      next j
next i
return
```

Улучшением пузырькового метода является шейкерная сортировка, где после каждого прохода меняют направление во внутреннем цикле.

***Эффективность алгоритма:***

Число сравнений $C_{max} = n(n-1)/2$ ,  $O(n^2)$.

Число перемещений $M_{max}=3C_{max}=3n(n-1)/2$, $O(n^2)$.

Если массив уже отсортирован и применяется алгоритм с флажком, то достаточно всего одного прохода, и тогда получаем минимальное число сравнений

$$C_{min} = n - 1, \qquad O(n),$$

а перемещения вообще отсутствуют

Сравнительный анализ прямых сортировок показывает, что обменная "сортировка" в классическом виде представляет собой нечто среднее между сортировками с помощью включений и с помощью выбора. Если же в нее внесены приведенные выше усовершенствования, то для достаточно упорядоченных массивов пузырьковая сортировка даже имеет преимущество.

## 6.4. Улучшенные методы сортировки

### 6.4.1. Быстрая сортировка (Quick Sort)

Относится к методам обменной сортировки. В основе лежит методика разделения ключей по отношению к выбранному.

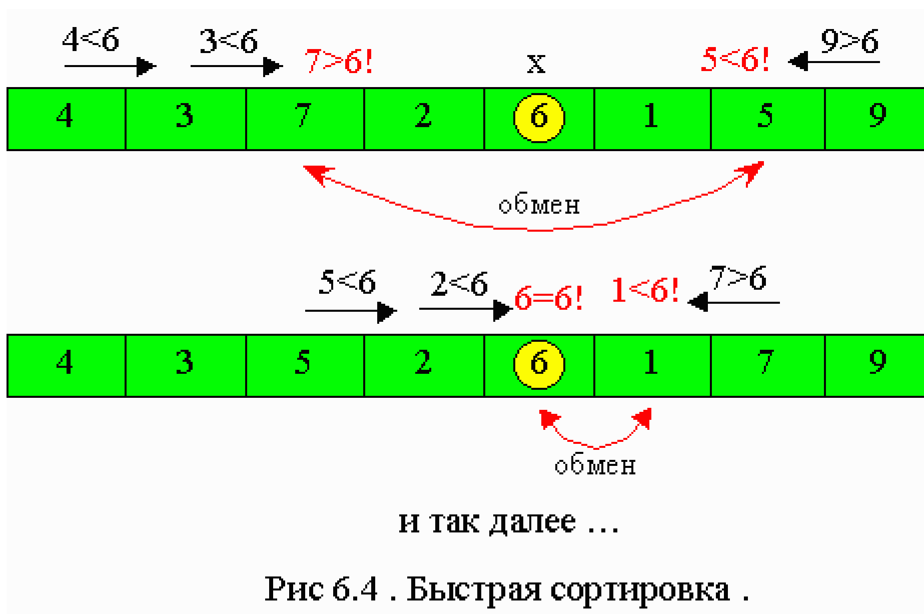



Рис 6.4 . Быстрая сортировка .

Слева от 6 располагают все ключи с меньшими, а справа - с большими или равными 6 (рис. 6.4).

```
Sub Sort (L, R)
i = L
j = R
x = a((L + R) div 2)
repeat
  while a(i) < x do
    i = i + 1
  endwhile
  while a(j) > x do
    j = j - 1
```

```
procedure Sort (L, R: in-
teger);
begin
  i := L;
  j := r;
  x := a[(L + r) div 2];
  repeat
    while a[i] < x do
      i := i + 1;
    while a[j] > x do
```

```
  endwhile
  if i <= j then
    y = a(i)
    a(i) = a(j)
    a(j) = y
    i = i + 1
    j = j - 1
  endif
until i > j
if L < j then
  sort (L, j)
endif
if i < R then
  sort (i, R)
endif
return

sub QuickSort
sort (1, n)
return
```

```
      j := j - 1;
    if i <= j then
      begin
        y := a[i];
        a[i] := a[j];
        a[j] := y;
        i := i + 1;
        j := j - 1
      end;
  until i > j;
  if L < j then sort (L, j);
  if i < r then sort (i, r);
end;

procedure QuickSort;
begin
  sort (1, n);
end;
```

***Эффективность алгоритма:***
O(n log n) - самый эффективный метод.

### 6.4.2 Сортировка Шелла (сортировка с уменьшающимся шагом)

В 1959 году Д. Шеллом было предложено усовершенствование сортировки с помощью метода прямого включения. Работа его представлена на рис. 6.5:

| сортировка | 44 | 55 | 12 | 42 | 94 | 18 | 6 | 67 |
|---|---|---|---|---|---|---|---|---|
| после четверной | 44 | 18 | 6 | 42 | 94 | 55 | 12 | 67 |
| двойной | 6 | 18 | 12 | 42 | 44 | 55 | 94 | 67 |
| одинарной | 6 | 12 | 18 | 42 | 44 | 55 | 67 | 94 |

Рис. 6.5.Сортировка Шелла . .

Сначала отдельно группируются и сортируются элементы, отстоящие друг от друга на расстоянии 4. Такой процесс называется четверной сортировкой. После первого прохода элементы перегруппировываются - теперь каждый элемент группы отстоит от другого на 2 позиции - и вновь сортируются. Это называется двойной сортировкой. И, наконец, на третьем проходе идет обычная или одиночная сортировка.

На первый взгляд можно засомневаться: если необходимо несколько процессов сортировки, причем в каждый включаются все элементы, то не добавят ли они больше работы, чем сэкономят? Однако, на каждом этапе либо сортируется относительно мало элементов, либо элементы уже довольно хорошо упорядочены и требуют сравнительно немного перестановок.

Ясно, что такой метод в результате дает упорядоченный массив, и, конечно, сразу же видно, что каждый проход от предыдущих только выигрывает; также очевидно, что расстояния в группах можно уменьшать по разному, лишь бы последнее было единичным, ведь в самом плохом случае последний проход и сделает всю работу.

Приводимый алгоритм основан на методе прямой вставки без барьера и не ориентирована на некую определенную последовательность расстояний, хотя в нем для конкретности заданы шаги 5, 3 и 1.

При использовании метода барьера  каждая из сортировок нуждается в постановке своего собственного барьера, и программу для определения его местонахождения необходи-

мо делать насколько возможно простой. Поэтому приходится расширять массив до [-h1..N].

h[1..t] - массив размеров шагов
a[1..n] - сортируемый массив
k - шаг сортировки
x - значение вставляемого элемента

***Subroutine ShellSort***

*Псевдокод:*

```
const t = 3
        h(1) = 5
        h(2) = 3
        h(3) = 1
  for m = 1 to t
    k = h(m)
    for i = k + 1 to n
      x = a(i)
      for j = i - k to 1 step -k
        if  x < a(j) then
          a( j+k) = a(j)
        else
           goto L
         endif
       next j
  L:  a(j+k) = x
      next i
     next m
    return
```

*Паскаль:*

```
const   t = 3;
           h(1) = 5;
           h(2) = 3;
           h(3) = 1;
var
  h: array [1..t] of integer;
  a: array [1..n] of integer;
   k, x, m, t, i, j: integer;
 begin
   for  m = 1 to t do
     begin
       k:= h(m);
       for i = k + 1 to n do
         begin
          x:= a(i);
          for j = i-k  downto k   do
             begin
              if  x < a(j ) then
                a(j+k):= a(j);
               else
                 goto 1;
              end ;
             end;
```

```
            end;
1:          a(j+1) = x ;
          end;
        end .
```

Не доказано, какие расстояния дают наилучший результат, но они не должны быть множителями один другого. Д. Кнут предлагает такую последовательность шагов (в обратном порядке): 1, 3, 7, 15, 31, … То есть: $h_{m-1} = h_m + 1$, $t = \log_2 n - 1$. При такой организации алгоритма его эффективность имеет порядок **O** ( $n^{1.2}$)

**Контрольные вопросы**

1. Что такое сортировка?
2. Назовите основные методы сортировки.
3. Какие методы сортировки относятся к строгим?
4. Какие методы сортировки относятся к улучшенным?
5. Какая сортировка называется устойчивой?
6. В чем состоит суть метода прямого включения?
7. В чем состоит суть метода прямого выбора?
8. В чем состоит суть метода прямого обмена?
9. Назовите разницу между этими тремя методами.
10. Какой метод сортировки является самым эффективным?
11. К какому из основных методов относится метод Шелла?

# 7. ПРЕОБРАЗОВАНИЕ КЛЮЧЕЙ (РАССТАНОВКА)

Метод расстановок (**хеширования**) направлен на быстрое решение задачи о местоположении элемента в структуре данных. В методе расстановок данные организованы как обычный массив. Поэтому H - отображение, преобразующее ключи в индексы массива, отсюда и появилось название *«преобразование ключей»,* обычно даваемое этому методу. Следует заметить, что нет никакой нужды обращаться к каким-либо процедурам динамического размещения, ведь массив — это один из основных, статических объектов. Метод преобразования ключей часто используется для таких задач, где можно с успехом воспользоваться и деревьями.

Основная трудность, связанная с преобразованием ключей, заключается в том, что множество возможных значений значительно шире множества допустимых адресов памяти (индексов массива). Возьмем в качестве примера имена, включающие до 16 букв и представляющие собой ключи, идентифицирующие отдельные индивиды из множества в тысячу персон. Следовательно, мы имеем дело с $26^{16}$ возможными ключами, которые нужно отобразить в $10^3$ возможных индексов. Поэтому функция H будет функцией класса «много значений в одно значение». Если дан некоторый ключ k, то первый шаг операции поиска — вычисление связанного с ним индекса h = H(k), а второй (совершенно необходимый) — проверка, действительно ли h идентифицирует в массиве (таблице) T элемент с ключом k.

## 7.1. Выбор функции преобразования

Само собой разумеется, что любая хорошая функция преобразования должна как можно равномернее распределять

ключи по всему диапазону значений индекса. Если это требование удовлетворяется, то других ограничений уже нет, и даже хорошо, если преобразование будет выглядеть как совершенно случайное. Такая особенность объясняет несколько ненаучное название этого метода — «перемалывание» (хеширование), т. е. дробление аргумента, превращение его в какое-то «месиво». Функция же H будет называться «функцией расстановки». Ясно, что H должно вычисляться достаточно эффективно, т. е. состоять из очень небольшого числа основных арифметических операций.

Представим себе, что есть функция преобразования ORD(k), обозначающая порядковый номер ключа k в множестве всех возможных ключей. Кроме того, будем считать, что индекс массива i лежит в диапазоне 0 **...** N — 1, где N — размер массива. В этом случае ясно, что нужно выбрать следующую функцию:

$$H(k) = ORD(k) \ \ MOD\ N$$

Для нее характерно равномерное отображение значений ключа на весь диапазон изменения индексов, поэтому ее кладут в основу большинства преобразований ключей. Кроме того, при N, равном степени двух, эта функция эффективно вычисляется. Однако если ключ представляет собой последовательность букв, то именно от такой функции и следует отказаться. Дело в том, что в этом случае допущение о равновероятности всех ключей ошибочно. В результате слова, отличающиеся только несколькими символами, с большой вероятностью будут отображаться в один и тот же индекс, что приведет к очень неравномерному распределению. Поэтому на практике рекомендуется в качестве N брать простое число. Следствием такого выбора будет необходимость использования полной операции деления, которую уже нельзя заменить выделением нескольких двоичных цифр.

### 7.2. Алгоритм

Если обнаруживается, что строка таблицы, соответствующая заданному ключу, не содержит желаемого элемента, то, значит, произошел конфликт, т. е. два элемента имеют такие ключи, которые отображаются в один и тот же индекс. В этой ситуации нужна вторая попытка с индексом, вполне определенным образом получаемым из того же заданного ключа. Существует несколько методов формирования вторичного индекса. Очевидный прием — связывать вместе все строки с идентичным первичным индексом H(k). Превращая их в связанный список. Такой прием называется *прямым связыванием* (direct chaining). Элементы получающегося списка могут либо помещаться в основную таблицу, либо нет; в этом случае память, где они размещаются, обычно называется *областью переполнения*. Недостаток такого приема в том, что нужно следить за такими вторичными списками и в каждой строке отводить место для ссылки (или индекса) на соответствующий список конфликтующих элементов.

Другой прием разрешения конфликтов состоит в том, что мы совсем отказываемся от ссылок и вместо этого просматриваем другие строки той же таблицы — до тех пор, пока не обнаружим желаемый элемент или же пустую строку. В последнем случае мы считаем, что указанного ключа в таблице нет. Такой прием называется *открытой адресацией*. Естественно, что во второй попытке последовательность индексов должна быть всегда одной и той же для любого заданного ключа. В этом случае алгоритм просмотра строится по такой схеме:

```
h = H(k)
i = 0
repeat
  if T(h)  =  k
```

```
          then  элемент найден
          else if T(h) = free
                    then элемента в таблице нет
                    else {конфликт}
                    i := i + 1
                    h := H(k) + G(i)
               endif
      endif
   until  либо найден, либо нет в таблице (либо она полна)
```

Предлагались самые разные функции G(i) для разрешения конфликтов. Приведенный в работе Морриса (1968) обзор стимулировал активную деятельность в этом направлении. Самый простой прием — посмотреть следующую строку таблицы (будем считать ее круговой), и так до тех пор, пока либо будет найден элемент с указанным ключом, либо встретится пустая строка. Следовательно, в этом случае G(i)==i, а индексы $h_i$, употребляемые при последующих попытках, таковы:

$h_0$ := H(k)
$h_i$ := ($h_i$ + i) MOD N,        i = 1 … N - 1

Такой прием называется *линейными пробами,* его недостаток заключается в том, что строки имеют тенденцию группироваться вокруг первичных ключей (т. е. ключей, для которых при включении конфликта не возникало). Конечно, хотелось бы выбрать такую функцию G, которая вновь равномерно рассеивала бы ключи по оставшимся строкам. Однако на практике это приводит к слишком большим затратам, потому предпочтительнее некоторые компромиссные методы; будучи достаточно простыми с точки зрения вычислений, они все же лучше линейной функции. Один из них — использование квадратичной функции, в этом случае последовательность пробуемых индексов такова:

$h_0 := H(k)$
$h_i := (h_i + i^2)$ MOD N, $\quad i > 0$

Если воспользоваться рекуррентными соотношениями для $h_i = i^2$ и $d_i = 2i + 1$ при $h_0 = 0$ и $d_0 = 1$, то при вычислении очередного индекса можно обойтись без операции возведения в квадрат.

$h_{i+1} = h_i + d_i$
$d_{i+1} = d_i + 2$ $(i > 0)$

Такой прием называется *квадратичными пробами,* существенно, что он позволяет избежать группирования, присущего линейным пробам, не приводя практически к дополнительным вычислениям. Небольшой же его недостаток заключается в том, что при поиске пробуются не все строки таблицы, т. е. при включении элемента может не найтись свободного места, хотя на самом деле оно есть. Если размер N — простое число, то при квадратичных пробах просматривается по крайней мере половина таблицы. Такое утверждение можно вывести из следующих рассуждений. Если i-я и j-я пробы приводят к одной и той же строке таблицы, то справедливо равенство

$i^2$ MOD N $= j^2$ MOD N

или

$(i^2 - j^2) = 0$ ( MOD N )

Разлагая разность на два множителя, получаем

$(i + j)(i - j) = 0$ ( MOD N )

Поскольку $i \neq j$, то либо i, либо j должны быть больше N/2, чтобы было справедливо равенство $i + j = cN$, где c — некоторое целое число. На практике упомянутый недостаток не столь существен, так как N/2 вторичных попыток при разрешении конфликтов встречаются очень редко, главным образом в тех случаях, когда таблица почти заполнена.

**Контрольные вопросы**

1. Для чего предназначен метод расстановок ?
2. От чего зависит положение элемента в массива в методе расстановок ?
3. Какую функцию возможно использовать в качестве хэш-функции:
4. Что называется конфликтом ?
5. Какой из методов является методом разрешения конфликтов.
6. Какой из методов разрешения конфликтов позволяет более равномерно распределить элементы по массиву.
7. Почему невозможно применять в качестве H(k) функцию Trunc(sqrt($k^5$- i*tan(k))) MOD N ?

# ЧАСТЬ 2.
# ПРАКТИКУМ ПО СРУКТУРАМ И АЛГОРИТМАМ ОБРАБОТКИ ДАН-НЫХ В ЭВМ

# МЕТОДИЧЕСКОЕ РУКОВОДСТВО К ЛАБОРАТОРНЫМ РАБОТАМ

## Организационно-методические указания

1. Перед началом лабораторной работы проводится консультация по методике выполнения лабораторных работ по данной дисциплине.
2. Объем каждой лабораторной работы, подготовка и порядок выполнения построены таким образом, чтобы все студенты выполнили работу и сдали отчеты.
3. Студенты готовятся к выполнению очередной работы заблаговременно.
4. Студенты обязаны изучить технику безопасности при работе на лабораторных установках до 1000 В.
5. Готовясь к лабораторному занятию, студент обязан изучить необходимый теоретический материал, пользуясь настоящими указаниями и рекомендованной литературой, произвести необходимые расчеты, заполнить соответствующую часть отчета и дать ответы на контрольные вопросы.
6. Неподготовленные студенты к выполнению лабораторной работы не допускаются.
7. Студенты, не сдавшие отчет во время занятия, сдают его в назначенное преподавателем время.
8. Студент, не выполнивший лабораторную работу, выполняет ее в согласованное с преподавателем время.
9. Каждая лабораторная работа выполняется студентами самостоятельно. Все студенты предъявляют индивидуальные отчеты. Допускается предъявление отчета в виде электронного документа.

10. Проверка знаний студентов производится преподавателем во время лабораторного занятия и при сдаче отчета.
11. При сдаче отчета студент должен показать знание теоретического материала в объеме, определяемом контрольными вопросами, а также пониманием физической сущности выполняемой работы.

## Лабораторная работа № 1. "ПОЛУСТАТИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ"

Цель работы: исследовать и изучить стеки.

Задача работы: овладеть навыками написания программ по исследованию стеков на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

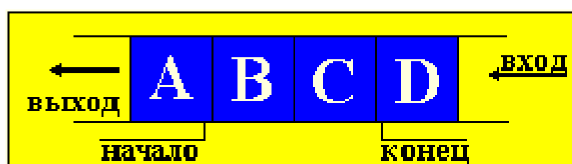
Понятие очереди всем хорошо известно из повседневной жизни. Элементами очереди в общем случае являются заказы на то или иное обслуживание: выбить чек на нужную сумму в кассе магазина, получить нужную информацию в справочном бюро, выполнить очередную операцию по обработке детали на данном станке в автоматической линии и т.д.

В программировании имеется структура данных, которая называется очередь. Эта структура данных используется, например, для моделирования реальных очередей с целью определения их характеристик (средняя длина очереди, время пребывания заказа в очереди и т.п.) при данном законе поступления заказов и дисциплине их обслуживания.

По своему существу очередь является полустатической структурой - с течением времени и длина очереди, и набор образующих ее элементов могут изменяться.

Различают два основных вида очередей, отличающихся по дисциплине обслуживания находящихся в них элементов :

1. При первой из дисциплин заказ, поступивший в очередь первым, выбирается первым для обслуживания (и удаляется из очереди). Эту дисциплину обслуживания принято называть **FIFO** (*First input-First output*, т.е. первый пришел - первый ушел). Очередь открыта с обеих сторон.




2. Вторую дисциплину принято называть **LIFO** (*Last input - First output*, т.е. последний пришел - первый ушел), при которой на обслуживание первым выбирается тот элемент очереди, который поступил в нее последним. Очередь такого вида в программировании принято называть СТЕКОМ (магазином) - это одна из наиболее употребительных структур данных, которая оказывается весьма удобной при решении различных задач.

В силу указанной дисциплины обслуживания, в стеке доступна единственная его позиция, которая называется ВЕРШИНОЙ стека - эта позиция, в которой находится последний по времени поступления в стек элемент. Когда мы заносим новый элемент в стек, то он помещается поверх вершины и теперь уже сам находится в вершине стека. Выбрать элемент можно только из вершины стека; при этом выбранный элемент исключается из стека, а в его вершине оказывается элемент, который был занесен в стек перед выбранным из него элементом (структура с ограниченным доступом к данным).

**Алгоритм**

ОПЕРАЦИИ НАД СТЕКАМИ:

- PUSH ( s , i ) - занесение элемента в стек, где s - название стека, i - элемент, который заносится в стек;
- POP ( s ) - выборка элемента из стека. При выборке элемент помещается в рабочую область памяти, где он используется;
- EMPTY ( s ) - проверка стека на пустоту (true - пуст, false - не пуст);
- STACKTOP ( s ) - чтение верхнего элемента без его удаления.

Фрагмент программы создания стека (необходимые процедуры)

```
Program STACK;
  const
     max_st=50;
const
    max_st=50;
  var
    st,st2: array[1..max_st] of integer;
    n:integer;
```

```
function empty:boolean;  {Проверка стека на наличие элементов в нем}
  begin
    empty:=n=0
  end;

procedure push(a:char);  {Поместить элемент в стек}
  begin
   inc(n);
   st[n]:=a;
  end;

procedure pop(var a:char);  {Извлечь элемент из стека}
  begin
   a:=st[n];
   dec(n);
  end;

function full:boolean;   {Проверка на переполнение}

  begin
   Full:=n=max_st
  end;

procedure stacktop(var a:char);   {Узнать верхний элемент}
  begin
   a:=st[n];
  end;

begin  {Основная программа}
   .
   .
end.
```

**Задания**

- Ввести символы, формируя из них стек.

  Варианты :

  1.Поменять местами первый и последний элементы стека.

  2.Развернуть стек, т.е. "дно" стека сделать вершиной, а вершину - "дном".

  3.Удалить элемент, который находится в середине стека, если нечетное число элементов, а если четное, то два средних.

  4.Удалить каждый второй элемент стека.

  5.Вставить символ '*' в середину стека, если четное число элементов, а если нечетное, то после среднего элемента.

  6.Найти минимальный элемент и вставить после него 0.

  7.Найти максимальный элемент и вставить после него 0.

  8.Удалить минимальный элемент.

  9.Удалить все элементы, равные первому.

  10.Удалить все элементы, равные последнему.

  11.Удалить максимальный элемент.

  12.Найти минимальный элемент и вставить на его место 0.
- Вывести полученный стек на экран.
- Распечатать полученный стек.

## Лабораторная работа № 2. "СПИСКОВЫЕ СТРУКТУРЫ ДАННЫХ"

Цель работы: исследовать и изучить списковые структуры данных.

Задача работы: овладеть навыками написания программ по исследованию списковых структур на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

До сих пор мы рассматривали только статические программные объекты. Этим термином обозначаются объекты, которые порождаются непосредственно перед выполнением программы, существуют в течение всего времени ее выполнения и размер значений которых не изменяется по ходу выполнения программы.

Поскольку статические объекты порождаются до выполнения программы и размер их значений можно выделить еще на этапе трансляции исходного текста программы на языке машины.

Однако использование при программировании только статических объектов может вызвать определенные трудности, особенно с точки зрения получения эффективной машинной программы, а такая эффективность бывает чрезвычайно важной при решении ряда задач. Дело в том, что иногда мы заранее не знаем не только размера значения того или иного программного объекта, но даже и того, будет существовать этот объект или нет. Такого рода программные объекты , которые возникают уже в процессе выполнения программы, называют динамическими объектами.

В языке ПАСКАЛЬ для работы с динамическими объектами предусматривается специальный тип данных - так называемый ссылочный тип. Значением этого типа является ссылка на какой - либо программный объект, по которой осуществляется непосредственный доступ к этому объекту. На машинном языке такая ссылка как раз и представляется указанием места в памяти соответствующего объекта.

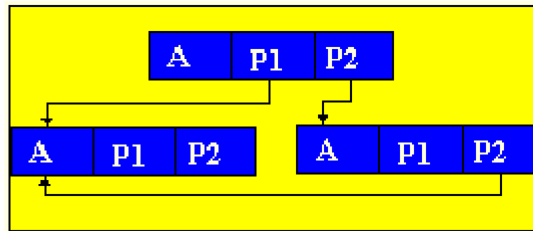
В этом случае для описания действий над динамическими объектами каждому такому объекту в программе сопоставляется статическая переменная ссылочного типа - в терминах этих ссылочных переменных и описываются действия над соответствующими динамическими объектами.

Итак, динамические структуры данных имеют две особенности :

1. Заранее не определимо количество элементов в структуре;

2. Элементы динамической структуры не имеют жесткой линейной упорядоченности. Они могут быть разбросаны по памяти.

Чтобы связать элементы динамической структуры между собой, в состав элемента помимо информационного поля входят поля указателей (связок с другими элементами структуры) .




p1, p2 - указатели, содержащие адреса элементов, с которыми данный элемент связан.

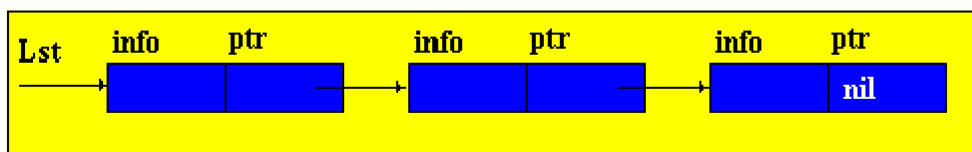
Наиболее распространенными динамическими структурами являются связанные списки. С точки зрения логического представления различают линейные и нелинейные списки. В линейных списках связи строго упорядочены. Указатель предыдущего элемента дает адрес последующего элемента или наоборот.

К линейным спискам относятся односвязные и двусвязные списки.

К нелинейным - многосвязные списки.

Элемент списка в общем случае представляет собой комбинацию поля записи (информационного поля) и одного или нескольких указателей.

## Линейные однонаправленные списки




Под односвязными списками понимают упорядоченную последовательность элементов, каждый из которых имеет 2 поля : информационное поле *info* и поле указателя *ptr* .

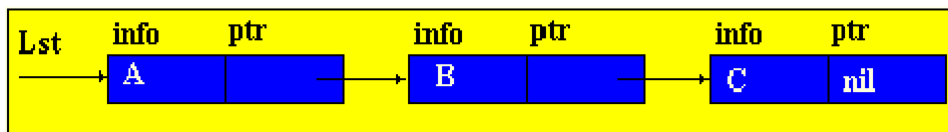
Особенностью указателя является то, что он дает только адрес последующего элемента списка. У однонаправленных списков поле указателя последнего элемента в списке является пустым nil.

*Lst* - указатель начала списка. Он представляет список как единое целое. Иногда список может быть пустым, т.е. в данном списке нет ни одного элемента. В этом случае *lst* = nil.

**Алгоритм**

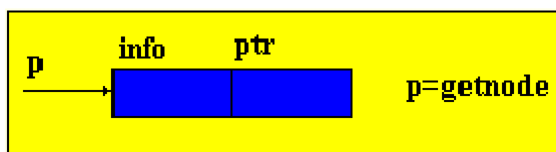
Операции с односвязными списками.

1. Вставка элемента в начало односвязного списка.

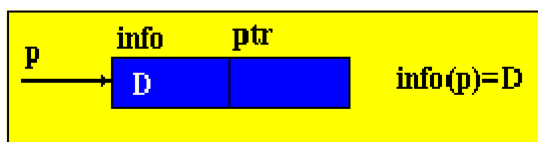



Вставим в начало данного списка элемент, информационное поле которого содержит переменную D. Чтобы это осуществить, необходимо произвести следующие действия :

a) Создать пустой элемент, на который указывает указатель p.




b) Информационному полю созданного элемента присвоить значение D.

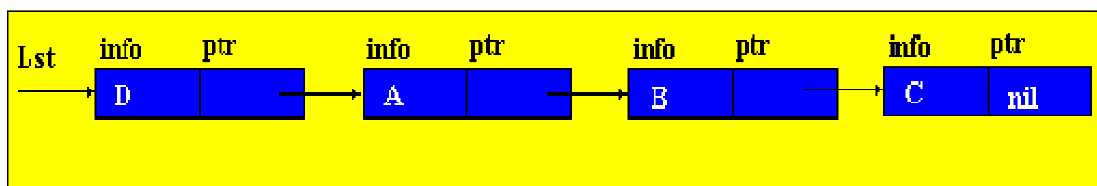



c) Связать новый элемент со списком.

Ptr(p)=lst (lst - уже не указывает на начало списка)

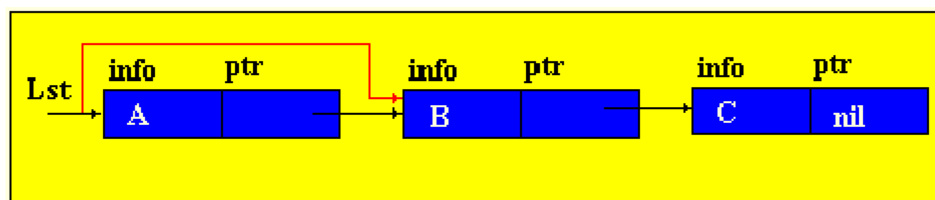
d) Перенести указатель lst на начало списка.

Окончательно:




**Удаление элемента из начала односвязного списка**

Удалим 1-й элемент списка, но при этом запомним информацию содержащиеся в поле info этого элемента.




Чтобы это осуществить, необходимо произвести следующие действия :

a) Ввести указатель p, который будет указывать на удаляемый элемент.

P=lst

b) Запомнить поле info элемента, на который ссылается указатель p, в некоторую переменную (x).

X=info( P )

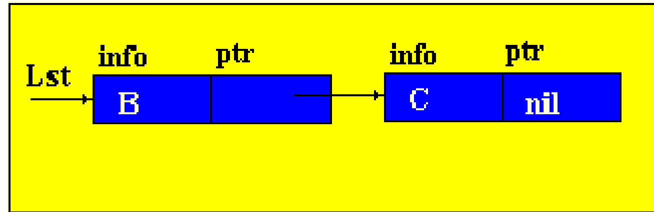
c) Перенести указатель lst на новое начало списка.

lst=ptr( P )

d) Удалить элемент на который указывает указатель p.

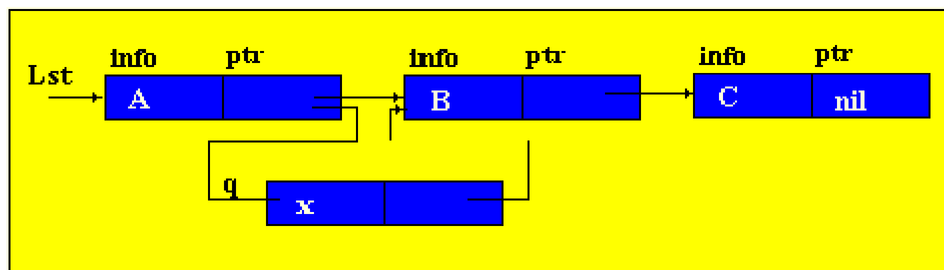
Freenode( P )

Окончательно:




### Вставка элемента в список

Вставим в данный список перед элементом на который указывает указатель p, элемент с информационным полем x.




Чтобы это осуществить необходимо произвести следующие действия :

a) Создать пустой элемент на который указывает указатель q.

Q=getnode

b) Внести x в информационное поле созданного элемента.

Info(Q)=x

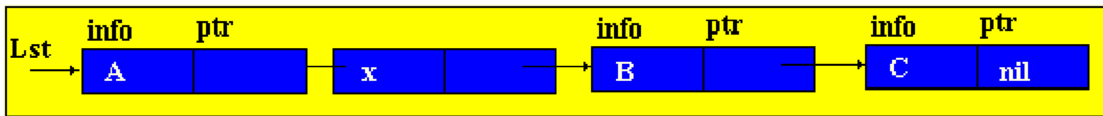
c) Связать элемент x с элементом B.

Ptr(Q)=Ptr(P) - это значит, что указателю созданного элемента присваивается значение указателя элемента p.

d) Связать элемент A с элементом x.

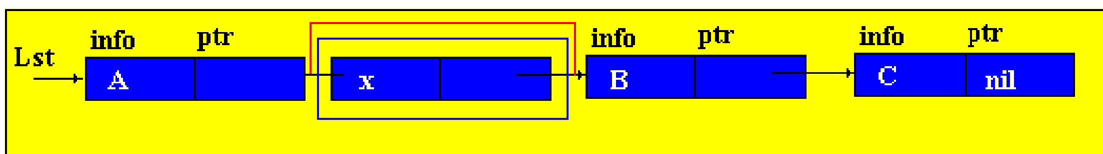
Ptr(p)=Q - это значит, что следующим за элементом A будет элемент на который указывает указатель Q.

Окончательно :




## Удаление элемента из односвязного списка

Удалим из списка элемент, который следует за элементом с рабочим указателем p.




Чтобы это осуществить необходимо произвести следующие действия :

a) Ввести указатель Q, который будет указывать на удаляемый элемент.

Q=ptr(p)

b) Поставить за элементом A элемент B.

Ptr(p)=Ptr(Q)

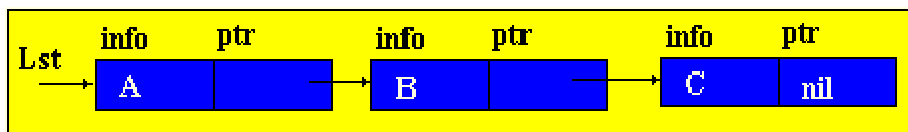
c) Запомнить информацию, которая содержится в поле info удаляемого элемента.

K=info(Q)

d) Удалим элемент с указателем Q.

Freenode(Q)

Окончательно :



**Задания**

Варианты :

1. Написать программу передвижения элемента на n позиций.
2. Создать копию списка.
3. Добавить элемент в начало списка.
4. Склеить два списка.
5. Удалить n-ый элемент из списка.
6. Вставить элемент после n-го элемента списка.
7. Создать список содержащий элементы общие для двух списков.
8. Упорядочить элементы в списке по возрастанию.
9. Удалить каждый второй элемент списка.
10. Удалить каждый третий элемент списка.
11. Упорядочить элементы списка по убыванию.
12. Очистить список.

## Лабораторная работа № 3. "КОЛЬЦЕВЫЕ СПИСКИ"

Цель работы: исследовать и изучить кольцевые списки на примере основных процедур.

Задача работы: овладеть навыками написания программ по исследованию основных процедур списковых структур на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

О том, что из себя представляют списки говорилось в предыдущей работе. Мы рассматривали однонаправленные списки, теперь мы рассмотрим кольцевые списки.

Как видно на рисунке, список замыкается в своеобразное "кольцо": двигаясь по ссылкам, можно от последнего элемен-

та списка переходить к заглавному элементу. В связи с этим списки подобного рода называют <u>кольцевыми списками</u>.

Чтобы закольцевать список необходимо присвоить указателю последнего элемента указатель начала списка (Ptr(p)=lst).

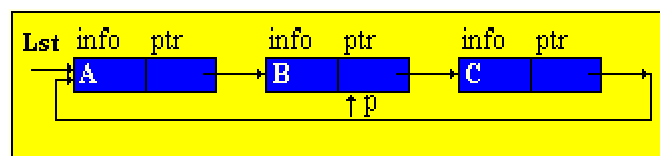
Ptr(p) - указатель последнего элемента;

Lst - указатель начала списка.

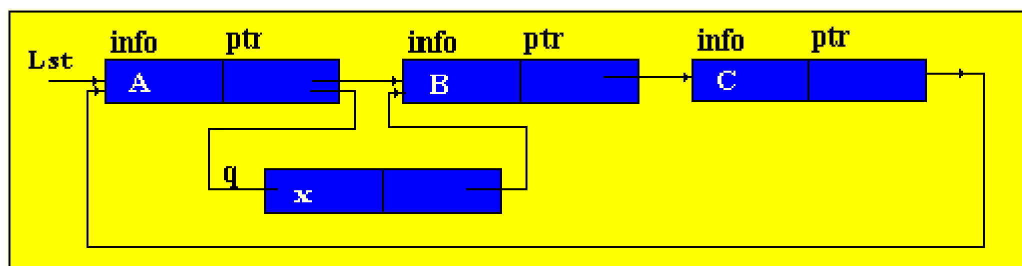
**Алгоритм**

Операции с кольцевыми списками:

**Вставка элемента в кольцевой список**




Чтобы это осуществить необходимо произвести следующие действия:




a) Создать пустой элемент на который указывает указатель q

q=getnode

b) Внести x в информационное поле созданного элемента

info(q)=x

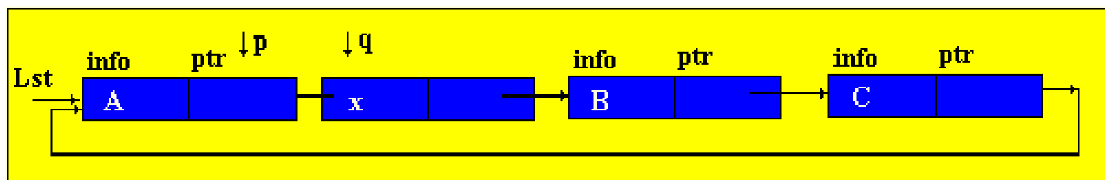
c) Связать элемент X с элементом B

ptr(q)=ptr(p) - это означает, что указателю созданного элемента присваивается значение указателя элемента p.

d) Связать элемент A с элементом X

ptr(p)=q - это означает, что следующим за элементом A будет элемент на который указывает указатель q.

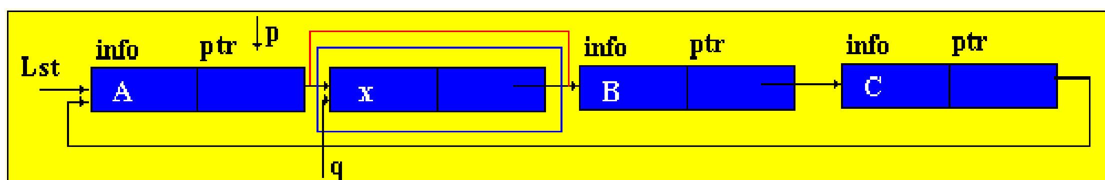
Окончательно:




Детально процесс вставки был проиллюстрирован в предыдущей работе.

## Удаление элемента из кольцевого списка

Удалим из списка элемент, который следует за элементом с рабочим указателем p.




Чтобы это осуществить, необходимо произвести следующие действия:

a) Ввести указатель q, который будет указывать на удаляемый элемент.

q=ptr(p)

b) Поставить за элементом A элемент B

ptr(p)=ptr(q)

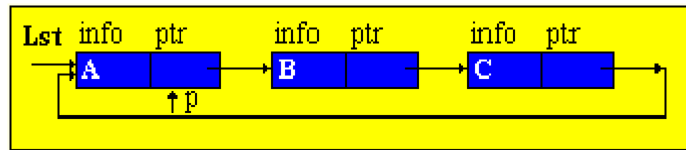
c) Запомнить информацию, которая содержится в поле info удаляемого элемента.

k=info(q)

d) Удалить элемент с указателем q.

Freenode(q)

Окончательно:




**Задания**

Варианты:

1) Дан кольцевой список, содержащий 20 фамилий игроков футбольной команды. Разбить игроков на 2 группы по 10 человек. Во вторую группу попадает каждый 12-й человек.

2) Даны 2 кольцевых списка, содержащие фамилии спортсменов 2-х фехтовальных команд. Произвести жеребьевку. В первой команде выбирается каждый n-й игрок, а во второй - каждый m-й.

3) Задача Джозефуса.

4) Даны 2 кольцевых списка, содержащие фамилии участников лотереи и наименования призов. Выиграет N человек (каждый К-й). Число для пересчета призов - t.

5) Даны 2 списка, содержащих фамилии учащихся и номера экзаменационных билетов. Число пересчета для билетов - Е, для учащихся - К. Определить номера билетов, вытащенных учащимися.

6) Дан список содержащий перечень товаров. Из элементов 1-го списка (товары изготовленные фирмой SONY) создать новый список.

7) Даны 2 списка, содержащие фамилии студентов 2-х групп. Перевести L студентов из 1-й группы во вторую. Число пересчета -К.

8) Даны 2 списка, содержащие перечень товаров, производимых Концернами BOSH и FILIPS. Создать список товаров, выпускаемых как одной так и другой фирмой.

9) Даны 2 списка, содержащие фамилии футболистов основного состава команды и запасного. Произвести К замен.

10) Даны 2 списка, содержащие фамилии солдат 1-го и 2-го взводов. Во время атаки М человек из 1-го взвода погибли. Произвести пополнение солдатами 2-го взвода.

11) Даны 2 списка, содержащие перечень товаров и фамилии покупателей. Каждый N-й покупатель покупает М-й товар. Вывести список покупок.

12) Даны 2 списка, содержащие наименования товаров, выпускаемых фирмами SONY и SHARP. Создать список товаров, конкурирующих между собой товаров.

## Лабораторная работа № 4. "МОДЕЛЬ МАССОВОГО ОБСЛУЖИВАНИЯ"

Цель работы: исследовать и изучить применение списковых структур данных при моделировании систем массового обслуживания.

Задача работы: овладеть навыками написания программ для теории массового обслуживания на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

Формулировка задачи.

Пусть обслуживающая система состоит из конечного числа обслуживающих аппаратов. Система относится к числу систем с ожиданием. Каждый аппарат может обслуживать одновременно только одно требование. Если в момент поступления очередного требования имеются свободные аппараты, то один из них немедленно приступает к обслуживанию, если свободных аппаратов нет, то требование ждет обслуживания. Естественно, если требований больше, чем обслуживающих аппаратов, то образуется очередь.

Время обслуживания одного требования есть случайная величина, подчиненная показательному закону распределения:

$$F(t)=1-e^{-\nu t}$$

Поток поступающих требований ограничен, то есть одновременно в системе обслуживания не может находиться больше m требований, где m - конечное число. Это дает право считать, что требования на обслуживание поступают от m обслуживаемых объектов, которые время от времени нуждаются в обслуживании. Пусть вероятность того, что поступит заявка на обслуживания на данном такте равна P(A) и вероятность того, что требование из очереди поступит на обслуживание равно P(B) ( на каждом такте может поступить не более одной заявки на обслуживание). Число обслуживающих аппаратов равно N. Допустим, требование дождалось своей очереди и оно начало обслуживаться. Обслуживание может длиться в течении не более 3-х тактов. Заявки могут быть двух приоритетов:

Заявка первого приоритета:

Это обычная заявка, она не обладает ни какими привилегиями. Она может покинуть систему через определенное число тактов T. При приходе в обслуживающую систему заявка первого приоритета становится в конец очереди.

Заявка второго приоритета:

Эта заявка отличается только тем, что она при поступлении в обслуживающую систему становится в начало очереди, то есть как только освобождается аппарат, то она поступает на обслуживание с вероятностью P(B).

Заявка второго приоритета , как и заявка первого приоритета, покидает систему через T тактов. Естественно, что появление заявок второго приоритета достаточно мало ( хотя эта вероятность задается пользователем и может быть любой ). Теперь об обслуживании: Количество тактов, в течение

которых будет обслуживаться та или иная заявка выбирается случайно (эта величина в данной задаче не должна быть больше 3). Если заявка обслужилась положенное ей число тактов, то она покидает систему. Если заявка находится в очереди больше Т тактов, то она с некоторой вероятностью покидает систему.

**Алгоритм**

Наиболее рациональна реализация очередей в виде списковых структур данных. Данная задача при ее реализации на ЭВМ дает большие навыки при работе со списковыми структурами и при написании алгоритма программы следует воспользоваться стандартными процедурами, такими как присоединение элемента в начало и конец списка, удаление произвольного элемента из списка. Эти процедуры имеют следующий вид:

**Процедура прибавления элемента в начало списка.**

p = getnode Node(p) - элемент, на который указывает
info(p) = d указатель P
ptr(p) = lst info(ptr(p)) - информационное поле следующего
lst = b элемента списка

**Процедура удаления из начала списка.**

p = lst
x = info(p)
lst = ptr(p)
freenode(p) - делает свободный узел с указателем

**Процедура прибавления элемента в список.**

p = getnode q - вставляемый

```
info(p) = x
ptr(q) = ptr(p)
ptr(p) = q
```

**Процедура удаления из списка**

```
q = ptr(p)        q - удаляемый
ptr(p) = ptr(q)
x = info(p)
freenode(q)
```

**Задания**

Общее задание для всех.

Пусть имеется обслуживающая система из n обслуживающих аппаратов. Работа этой системы разбита на такты. В течение одного такта может одна заявка стать в очередь и одна заявка приступить к обслуживанию, (разумеется, если аппарат свободен). Вероятность заявки поступить на обслуживание P(A), вероятность обслужить заявку P(B), вероятность заявки покинуть очередь после T тактов P(C). После каждых L тактов давать информацию о длине очереди и число тактов, в течении которых обслуживающий аппарат простаивал. Четным вариантам реализовать обслуживающую систему с неограниченной очередью, нечетным вариантам с конечной очередью (т.е. если в очереди будет стоять K заявок, то следующая заявка получает отказ в обслуживании).

Варианты:

1) L=50, после окончания работы системы выдать информацию, сколько заявок покинуло систему без обслуживания.

2) L=55, после окончания работы системы выдать информацию, сколько заявок обслуживалось больше 2 тактов.

3) L=100, после окончания работы системы выдать информацию, сколько тактов очередь была пустой.

4) L=75 , после окончания работы системы выдать информацию, сколько заявок обслуживалось один такт.

5) L=25 , после окончания работы системы выдать информацию, сколько заявок первого приоритета приступили к обслуживанию.

6) L=40 , после окончания работы системы выдать информацию о среднем приращении очереди.

7) L=80 , после окончания работы системы выдать информацию, сколько заявок обслуживалось 2 такта.

8) L=100, после окончания работы системы выдать информацию, заявок обслужилось.

9) L=70 , после окончания работы системы выдать информацию, на каком такте была самая длинная очередь.

10) L=50, после окончания работы системы посчитать практическую вероятность простоя аппарата по формуле s/n, где s- число тактов простоя аппарат, n- общее число тактов.

11) L=65, после окончания работы системы выдать информацию, сколько заявок второго приоритета поступили на обслуживания.

12) L=30, после окончания работы системы выдать информацию, сколько заявок обслуживалось 2 или 3 такта.

## Лабораторная работа № 5. "БИНАРНЫЕ ДЕРЕВЬЯ(основные процедуры)"

Цель работы: исследовать и изучить процедуры, используемые при работе с бинарными (двоичными) деревьями.

Задача работы: овладеть навыками написания программ по исследованию бинарных деревьев .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

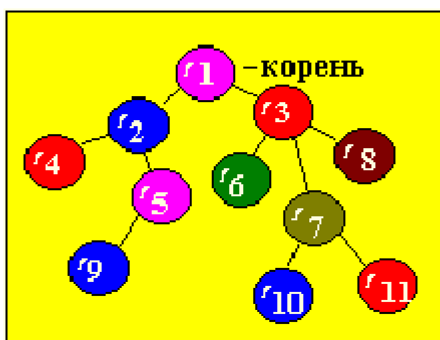
### Краткая теория

Общие сведения о структуре данных – дерево.

Дерево - это нелинейная связанная структура данных, характеризуемая следующими признаками :

- дерево имеет один элемент, на который нет ссылок от других элементов. Этот элемент, или "узел", называется корнем дерева;
- в дереве можно обратиться к любому элементу путем прохождения конечного числа ссылок (указателей) ;
- каждый элемент дерева связан только с одним предыдущим элементом.

Каждый узел дерева может быть промежуточным (элементы 2,3,5,7) либо терминальным ("листом" дерева) (элементы 4,9,10,11,8,6). Характерной особенностью терминальных узлов является отсутствие ветвей.




Элемент Y, находящийся непосредственно ниже элемента X, называется непосредственным потомком X, если X находится на уровне i , то говорят, что Y лежит на уровне i+1.

Считается, что корень лежит на уровне 0.

Число непосредственных потомков элемента называется его степенью исхода, в зависимости от степени исхода узлов деревья классифицируют :

A) Если степень исхода узлов - M, то дерево называется M-арным ;

B) Если степень исхода узлов - M или 0, то - полное M-арное дерево;

C) Если степень исхода узлов дерева равна 2, то дерево называется бинарным ;

D) Если степень исхода равна 2 или 0, то - полное бинарное дерево.

Особенно важную роль играют бинарные деревья, далее мы будем рассматривать их более подробно.

## Представление деревьев в памяти ЭВМ

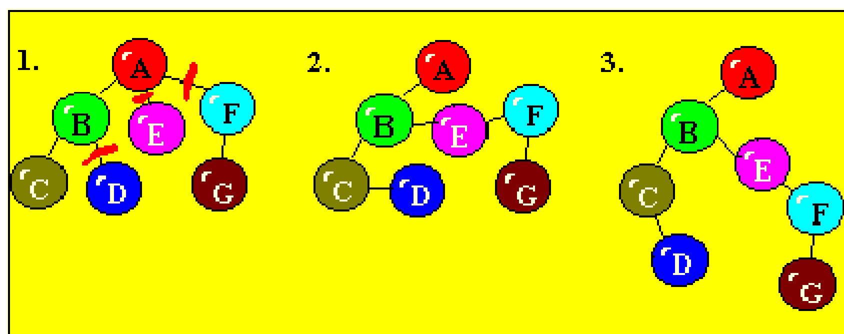
Деревья наиболее удобно представлять в памяти ЭВМ в виде связанных нелинейных списков. Элемент должен со-

держать INFO-поле, где содержится характеристика узла. Следующее поле определяет степень исхода узла и количество полей указателей равное степени исхода. Каждый указатель элемента ориентирует данный элемент-узел с его сыновьями. Узлы, в которые входят стрелки от исходного элемента, называются его сыновьями.

INFO-поле содержит два поля : поле записи (rec) и поле ключа (key). Ключ задается числом, по ключу определяют место элемента в дереве.

## Сведение М-арного дерева к бинарному

1. В каждом узле дерева отсекают все ветви, кроме крайних левых, соответствующих старшим сыновьям;
2. Соединяют горизонтальными линиями сыновей одного родителя (узла);
2. Старшим сыном в каждом узле полученной структуры будет узел под обрабатываемым узлом.



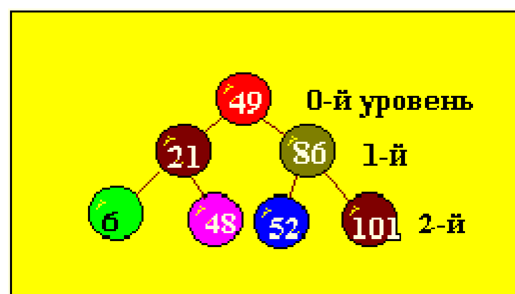
## Построение бинарных деревьев

Согласно представлению деревьев в памяти ЭВМ каждый элемент (узел бинарного дерева) будет записью, содержащей четыре поля. Значением этих полей будут соответственно ключ записи, ссылка на элемент влево-вниз, на элемент вправо-вниз и на текст записи.

При построении необходимо помнить, что левый сын имеет ключ меньше чем отец (родитель). Значение ключа правого сына больше значения ключа отца.

Например, узлы дерева имеют значения 6, 21, 48, 49, 52, 86, 101.

Бинарное дерево будет иметь вид:



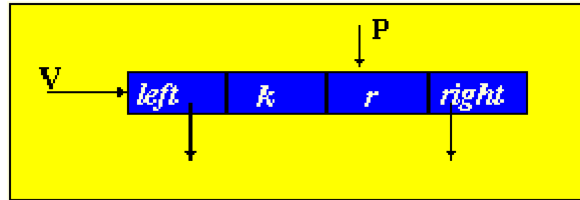

Получили упорядоченное бинарное дерево с минимальным числом уровней.

Идеально сбалансированное дерево - это дерево, в котором левое и правое поддеревья имеют уровни, отличающиеся не больше, чем на единицу.

### Алгоритм

### Процедура создания бинарного дерева

Для построения дерева необходимо создать в памяти ЭВМ элемент следующего типа:




P, Q - рабочие указатели

V=maketree(key,rec) - процедура, которая создает сегмент ключа и записи

P=getnode - генерация нового элемента

r=rec

k=key

V=P

left=nil

right=nil

tree - указатель на корень дерева

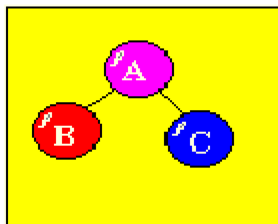
Введем сначала первое значение ключа, потом сгенерируем сам элемент узла дерева процедурой maketree. Далее идем по циклу до тех пор, пока указатель не передвинется на нулевое значение.

```
READ(key,rec)
tree=maketree(key,rec)
WHILE not eof DO
  READ(key,rec)
  V=maketree(key,rec)
  WHILE P<>nil DO
     Q=P
     IF key=k(P)
       THEN P=left(P)
       ELSE P=right(P)
    END IF
 END WHILE
 IF P=nil
  THEN WRITELN(' Это корень');
  tree=V
```

```
  ELSE IF key<k(q)
           THEN left(P)=V
           ELSE right(P)=V
          END IF
   END IF
END WHILE
```

**Процедуры "обхода" дерева**

Пусть задано бинарное дерево :




Существуют три принципа обхода бинарных деревьев. Рассмотрим их на примере заданного дерева :

1) Обход сверху вниз (корень посещается ранее, чем поддеревья) : A, B, C ;

2) Слева направо : B, A, C ;

3) Снизу вверх (корень посещается после поддеревьев) : B, C, A .

Наиболее часто применяется 2-ой способ, узлы посещаются в порядке возрастания значения их ключа.

<u>Рекурсивные процедуры обхода дерева :</u>

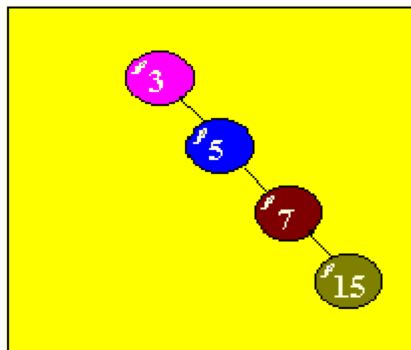
```
1) PROCEDURE pretrave (tree)
         IF tree<>nil
           THEN PRINT info (tree)
                      pretrave (left (tree))
                      pretrave (right (tree))
         END IF
      RETURN
2) PROCEDURE intrave (tree)
```

```
        IF tree<>nil
          THEN intrave (left (tree))
               PRINT info (tree)
               intrave (right (tree))
        END IF
        RETURN
3) PROCEDURE postrave (tree)

        IF tree<>nil
          THEN postrave (left (tree))
               postrave (right (tree))
               PRINT info (tree)
        END IF
        RETURN
```

**Процедура поиска по бинарному дереву**

Назначение этой процедуры в том, чтобы по заданному ключу осуществить поиск узла дерева. Длительность операции поиска (число узлов, которое надо перебрать для этой цели) зависит от структуры дерева. Действительно, дерево может быть вырождено в однонаправленный список (иметь единственную ветвь) - такое дерево может возникнуть, если элементы поступали в дерево в порядке возрастания (убывания) их ключей, например :



В этом случае время поиска будет такое же, как и в однонаправленном списке, т.е. в среднем придется перебрать N/2 элементов.

Наибольший эффект использования дерева достигается в том случае, когда дерево "сбалансировано". В этом случае для поиска придется перебрать не более LOG2N элементов.

Опишем процедуру поиска. Переменной search будет присваиваться указатель на найденное звено :

```
p=tree
    WHILE p<>nil DO
        IF key=k (p)
          THEN search=p
         RETURN
        END IF
        IF key<>k (p)
          THEN p=left (p)
          ELSE p=right (p)
        END IF
     END WHILE
     search=nil
     RETURN
```

**Процедура включения элемента в дерево**

Для включения новой записи в дерево прежде всего нужно найти тот его узел, к которому можно "подвесить" (присоединить) новый элемент. Алгоритм поиска нужного узла тот же самый, что и при поиске узла с заданным ключом. Этот узел будет найден в тот момент, когда в качестве очередной ссылки, определяющей ветвь дерева, в которой надо продолжить поиск, окажется ссылка NIL.

Однако непосредственно использовать описанную выше процедуру поиска нельзя, потому что по окончании вычисления ее значения не фиксируется тот узел, из которого была выбрана ссылка NIL.

Модифицируем описание процедуры поиска так, чтобы в качестве ее побочного эффекта фиксировалась ссылка на узел, в котором был найден заданный ключ (в случае успешного поиска), или ссылка на узел, после обработки которого поиск прекращен (в случае неуспешного поиска). И опишем процедуру включения элемента в дерево, учитывая, что в дереве нет узла с тем же ключом, что и у включаемого элемента.

```
q=nil
p=tree
WHILE p<>nil DO
      q=p
   IF key=k (p)
    THEN search=p
    RETURN
   END IF
   IF key<k (p)
    THEN p=left (p)
    ELSE p=right (p)
   END IF
END WHILE
{Узел с заданным ключом не найден, требуется вставка элемента. На узел предполагаемого отца указывает q.}
V=maketree (key, rec)
{Нужно выяснить, левым или правым сыном будет вставляемый элемент V.}
 IF key<k (q)
   THEN left (q)=V
   ELSE right (q)=V
 END IF
 search=V
RETURN
```

## Процедура удаления элемента из бинарного дерева

Удаление узла не должно нарушать упорядоченность дерева. При удалении элемента из бинарного дерева с заданным ключом различают три случая :

1) удаляемый узел является листом, в этом случае он просто удаляется, не нарушая этим упорядоченность дерева ;

2) если удаляемый узел имеет только одного сына, то сын перемещается на место удаляемого отца ;

4) если у удаляемого узла 2 сына. В этом случае на место удаляемого отца помещается либо его предшественник в симметричном обходе (обход слева направо), либо его последователь.

Разработаем алгоритм для 3-его случая (вершина-узел имеет 2-х сыновей), вместо удаляемого узла поставим его приемника в симметричном обходе. Предшественником удаляемого узла 12 является самый правый узел левого поддерева - узел 11, а приемником или последователем при симметричном обходе (обход слева направо) является самый левый узел правого поддерева - узел 13.

Будем использовать следующие указатели :

p - рабочий указатель;

q - отстает на один шаг от p;

v - указывает приемника;

t - отстает от v на один шаг;

s - опережает v на один шаг.

1) Процедурой поиска элемента находим удаляемый элемент. Указатель p указывает на удаляемый элемент.

2) Установим указатель v на узел, который будет замещать удаляемый элемент.

```
      IF left (p)=nil
       THEN v=right (p)
       ELSE IF right (p)=nil
                THEN v=left (p)
                ELSE t=p
                        v=right (p)
                        s=left (v)
WHILE s<>nil
          t=v
          v=s
          s=left (v)
END WHILE
      IF t<>p
       THEN WRITE (v не является сыном p)
                left (t)=right (v)
               right (v)=right (p)
      END IF
left (v)=left (p)
       IF q=nil
        THEN WRITE (v корень)
                tree=v
        RETURN
       END IF
       IF p=left (q)
        THEN left (q)=v
        ELSE right (q)=v
       END IF
    END IF
END IF
```

FREENODE (p) (процедура создает свободный узел, т.е. очищает ячейку памяти, в которой находится удаленный элемент)

RETURN

**Задания**

Варианты:

1.Описать процедуру или функцию, которая:

a) присваивает параметру E запись из самого левого листа непустого дерева T (лист-вершина, из которого не выходит ни одной ветви);

b) определяет число вхождений записи E в дерево T.

2. Вершины дерева вещественные числа. Описать процедуру или функцию, которая:

a) вычисляет среднее арифметическое всех вершин дерева;

b) добавляет в дерево вершину со значением, вычисленным в предыдущей процедуре (функции).

3. Записи вершин дерева - вещественные числа. Описать процедуру, которая удаляет все вершины с отрицательными записями.

4. Записи вершин дерева - вещественные числа. Описать процедуру или функцию, которая:

a) находит максимальное или минимальное значение записей вершин непустого дерева;

b) печатает записи из всех листьев дерева.

5. Описать процедуру или функцию, которая:

a) определяет, входит ли вершина с записью E в дерево T;

b) если такая запись не найдена, то она добавляется.

6. Описать процедуру или функцию, которая:

a) находит в непустом дереве T длину (число ветвей) пути от корня до ближайшей вершины с записью E; если E не входит в T, то за ответ принять -1.

b) определяет максимальную глубину непустого дерева T, т.е. число ветвей в самом длинном из путей от корня дерева до листьев.

7. Описать процедуру COPY(T,T1), которая строит T1 - копию дерева T.

8. Описать процедуру EQUAL(T1,T2), проверяющую на равенство деревья T1 и T2 (деревья равны, если ключи и записи вершин одного дерева равны соответственно ключам и записям другого дерева).

9. Описать процедуру или функцию, которая:

a) печатает узлы непустого дерева при обходе слева направо;

b) удаляет все листья исходного дерева;

c) печатает модифицированное дерево.

10. Описать процедуру, которая:

a) присваивает переменной b типа char значение:

К - если вершина - корень,

П - если вершина - промежуточная вершина,

Л - если вершина - лист;

b) распечатывает атрибуты всех вершин дерева.

11. Описать процедуру или функцию, которая:

a) вставляет узел с записью E в дерево, если ранее такой не было;

b) удалить ее, если она уже существует.

12. Описать процедуру или функцию, которая:

a) печатает дерево, встречающееся в дереве один раз;

b) печатает запись, встречающееся в дереве максимальное число раз.

## Лабораторная работа № 6 . "СОРТИРОВКА МЕТОДОМ ПРЯМОГО ВКЛЮЧЕНИЯ"

<u>Цель работы:</u> исследовать и изучить методы сортировки включением.

<u>Задача работы:</u> овладеть навыками написания программ для сортировки методом прямого включения на языке программирования ПАСКАЛЬ .

<u>Порядок работы :</u>

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

При обработке данных в ЭВМ важно знать и информационное поле элемента, и его размещение в памяти машины. Для этих целей используется сортировка. Итак, сортировка- - это расположение данных в памяти в регулярном виде по их ключам. Регулярность рассматривают как возрастание значения ключа от начала к концу в массиве.

Различают следующие типы сортировок:

- внутренняя сортировка - это сортировка, происходящая в оперативной памяти машины
- внешняя сортировка - сортировка во внешней памяти.

Если сортируемые записи занимают большой объем памяти, то их перемещение требует больших затрат. Для того, чтобы их уменьшить, сортировку производят в таблице адресов ключей, делают перестановку указателей, т.е. сам массив не перемещается. Это метод сортировки таблицы адресов.

При сортировке могут встретиться одинаковые ключи. В этом случае при сортировке желательно расположить после сортировки одинаковые ключи в том же расположении, что и в исходном файле. Это устойчивая сортировка.

Эффективность сортировки можно рассмотреть с нескольких критериев:

- время, затрачиваемое на сортировку
- объем оперативной памяти, требуемой для сортировки
- время, затраченное программистом на написание программы

Рассмотрим второй критерий подробнее: мы можем подсчитать количество сравнений при выполнении сортировки или количество перемещений.

Предположим, что число сравнений определяется формулой:

$$\mathbf{C = 0{,}01n^2 + 10n}$$

Если $n<1000$, то второе слагаемое больше.

Если $n>1000$, то первое слагаемое больше.

То есть, при малых n порядок сравнения равен , а при больших n он равен n.

Различают следующие методы сортировки:

- строгие (прямые) методы
- улучшенные методы

Рассмотрим преимущества прямых методов:

1. Программы этих методов легко понимать, и они коротки. Напомним, что сами программы также занимают память.

2. Прямые методы особенно удобны для объяснения характерных черт основных принципов большинства сортировок.

3. Усложненные методы требуют небольшого числа операций, но эти операции обычно сами более сложны, и поэтому для достаточно малых n прямые методы оказываются быстрее, хотя при больших n их использовать, конечно, не следует.

Методы сортировки "на том же месте" можно разбить в соответствии с определяющими их принципами на 3 категории:

1. Сортировки с помощью включения (by insertion)
2. Сортировки с помощью выделения (by selection)
3. Сортировки с помощью обменов (by exchange)

Сортировка методом прямого включения

Такой метод широко используется при игре в карты. Элементы (карты) мысленно делятся на уже "готовую" последовательность a(1),...,a(i-1) и исходную последовательность. При каждом шаге, начиная с i=2 и увеличивая i каждый раз на единицу, из исходной последовательности извлекается i-й элемент и перекладывается в готовую последовательность, при этом он вставляется на нужное место.

**Алгоритм**

Алгоритм сортировки прямым включением таков:

```
for x:=2 to n do
  x:=a[i]
  включение x на соответствующее место среди a[1]...a[i]
```

```
end
```

В реальном процессе поиска подходящего места удобно, чередуя сравнения

сравнивается с очередным элементом a( j ), а затем либо x вставляется на свободное место, либо a( j ) сдвигается (передается) вправо и процесс "уходит" влево. Обратите внимание, что процесс просеивания может закончиться при выполнении одного из двух следующих различных условий:

1. Найден элемент a( j ) с ключом, меньшим, чем ключ у x.
2. Достигнут левый конец готовой последовательности.

```
Procedure StraightInsertion
 Var
   i,j:index; x:item;
 begin
   for i:=2 to n do
     x:=a[i]; a[0]:=x; j:=1;
     while x<a[j-1] do a[j-1]:=a[j-1]; j:=j-1; end;
     a[j]:=x
     end
 end StraightInsertion
```

**Задания**

В ремонтной мастерской находятся несколько (N) машин. О них имеются следующие сведения: номер, марка, имя владельца, дата последнего ремонта (число, месяц, год), день, к которому машина должна быть отремонтирована (число, месяц, год).

Требуется (согласно варианту) :

1.Расположить по алфавиту имена владельцев и, соответственно, вывести информацию об их машинах.

2.Исходя из того, что машина, дата окончания ремонта которой раньше, должна ремонтироваться в первую очередь, вывести порядок ремонта автомобилей.

3.Вывести по убыванию количество всех предыдущих ремонтов машин марки "Жигули".

4.Вывести по убыванию номера тех машин, количество предыдущих ремонтов которых равно 2.

5.Вывести по возрастанию даты конца ремонта всех машин, которые ранее не ремонтировались.

6.Вывести по алфавиту в обратном порядке владельцев автомобилей марки "Мерседес"

7.Вывести по алфавиту марки машин, которые должны быть отремонтированы раньше всех (дата конца ремонта меньше 01.08.96).

8.Вывести по возрастанию номера машин марки "Жигули".

9.Вывести по алфавиту имена владельцев, чьи машины не ремонтировались с прошлого года.

10.Вывести машины, которые надо отремонтировать к следующему месяцу по возрастанию даты их последнего ремонта.

11. Вывести по алфавиту в обратном порядке имена владельцев, количество предыдущих ремонтов машин которых больше трех.

12. Вывести по убыванию номера машин марки "Мерседес".

## Лабораторная работа № 7. "СОРТИРОВКА МЕТОДОМ ПРЯМОГО ВЫБОРА"

<u>Цель работы:</u> исследовать и изучить сортировки методом прямого выбора.

<u>Задача работы:</u> овладеть навыками написания программ для сортировки методом прямого выбора на языке программирования ПАСКАЛЬ .

<u>Порядок работы :</u>

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

В общем сортировку следует понимать как процесс перегруппировки заданного множества объектов в некотором определенном порядке. Цель сортировки облегчить последующий  поиск элементов в таком отсортированном множестве. Это почти универсальная, фундаментальная деятельность. Мы встречаемся с отсортированными объектами в телефонных книгах, в списках подоходных налогов, в оглавлениях книг, в библиотеках, в словарях, на складах почти везде, где нужно искать хранимые объекты. Даже малышей учат держать свои вещи "в порядке", и они уже сталкиваются с неко-

торыми видами сортировок задолго до того, как познакомятся с азами арифметики.

Таким образом, разговор о сортировке вполне уместен и важен, если речь идет об обработке данных. Что может легче сортироваться, чем данные? Тем не менее наш первоначальный интерес к сортировке основывается на том, что при построении алгоритмов мы сталкиваемся со многими весьма фундаментальными приемами. Почти не существует методов, с которыми не приходится встречаться при обсуждении этой задачи. В частности, сортировка это идеальный объект для демонстрации огромного разнообразия алгоритмов, все они изобретены для одной и той же задачи, многие в некотором смысле оптимальны, большинство имеет свои достоинства. Поэтому это еще и идеальный объект, демонстрирующий необходимость анализа производительности алгоритмов. К тому же на примерах сортировок можно показать, как путем усложнения алгоритма, хотя под рукой и есть уже очевидные методы, можно добиться значительного выигрыша в эффективности.

Выбор алгоритма зависит от структуры обрабатываемых данных это почти закон, но в случае сортировки такая зависимость столь глубока, что соответствующие методы были даже разбиты на два класса сортировку массивов и сортировку файлов последовательностей . Иногда их называют внутренней и внешней сортировкой, поскольку массивы хранятся в быстрой оперативной, внутренней памяти машины со случайным доступом, а файлы обычно размещаются в более медленной, но и более емкой внешней памяти, на устройствах, основанных на механических перемещениях дисках или лентах . Уже на примере сортировки пронумерованных карточек становится очевидным существенное различие в этих подходах. Если карты "выстроены" в виде массива, то они как бы лежат перед сортирующим, он видит каждую из них и имеет к ней доступ. Если же карты образуют файл, то это предполагает, что видна только верхняя карта в каждой

из стопок. Такое ограничение, конечно же, серьезно повлияет на метод сортировки, но ничего не поделаешь: ведь карточек может быть так много, что все они на столе не поместятся.

Прежде чем идти дальше, введем некоторые понятия и обозначения. Ими мы будем пользоваться далее. Если у нас есть элементы a1, a2,..., an, то сортировка есть перестановка этих элементов массив ak1, ak2, ..., akn, где при некоторой упорядочивающей функции f выполняются отношения f ak1 <= f ak2 <= ... <= f akn.

Обычно упорядочивающая функция не вычисляется по какому-либо правилу, а хранится как явная компонента поля каждого элемента. Ее значение называется ключом key элемента. Поэтому для представления элементов хорошо подходят такие образования, как запись, а графически это представляется так (рис. 1):

Говоря об алгоритмах сортировки, мы будем обращать внимание лишь на компоненту - ключ, другие же компоненты можно даже и не определять (рис.2). Поэтому из наших дальнейших рассмотрений вся сопутствующая информация. Чтобы уменьшить эти затраты, сортировку производят в таблице адресов ключей. После сортировки перестанавливают указатели. Это метод сортировки таблицы адресов (рис.3) . Метод сортировки называется устойчивым, если в процессе сортировки относительное расположение элементов с равными ключами не изменяется. Устойчивость сортировки часто бывает желательной, если речь идет об элементах, уже упорядоченных (отсортированных) по некоторым вторичным ключам (т.е. свойствам) , не влияющим на основной ключ.

Отсортированный массив (рис. 2):

Массив отсортированный другим методом (рис. 3):

Основное условие: выбранный метод сортировки массивов должен экономно использовать доступную память. Это предполагает, что перестановки, приводящие элементы в порядок, должны выполняться на том же месте, т.е. методы, в

которых элементы из массива А передаются в результирующий массив H, представляют существенно меньший интерес. Ограничив критерием экономии памяти наш выбор нужного метода среди многих возможных, мы будем сначала классифицировать методы по их экономичности, т.е. по времени их работы. Хорошей мерой эффективности может быть С - число необходимых сравнений ключей и М - число пересылок (перестановок) элементов.

Эти числа - функции от n- числа сортируемых элементов. Хотя улучшенные алгоритмы сортировки требуют порядка n*log n сравнений, мы разберем простой метод, который причисляется к так называемым прямым, где требуется порядка n^2 сравнений ключей. К достоинствам прямых методов относятся :

1.Прямые методы особенно удобны для объяснения характерных черт основных принципов большинства сортировок.

2.Программы этих методов легко понимать, и они коротки. Напомним, что сами программы также занимают память.

3.Улучшенные методы требуют небольшого числа операций, но эти операции обычно сами более сложны, и поэтому для достаточно малых n прямые методы оказываются быстрее, хотя при больших n их использовать, конечно, не следует.

Методы сортировки "на том же месте" можно разбить в соответствии с определяющими их принципами на три основные категории:

1.Сортировки прямым включением By insertion .

2.Сортировки прямым выделением By selection .

3.Сортировки прямым обменом By exchange .

**Алгоритм**

К прямым методам относится метод прямого выбора.

Рассматриваем весь ряд массива и выбираем элемент меньший или больший элемента a(i), определяем его место в массиве - k, и затем меняем местами элемент a(i) и элемент a(k).

```
for i = 1 to n - 1
  x = a(i)
  k = i
  for j = i + 1 to n
    if x > a(j) then
      k = j
      x = a(k)
    endif
  next j
  a(k) = a(i)
  a(i) = x
next i
return
for i := 1 to n - 1 do
  begin
    x := a[i];
    k := i;
    for j := i + 1 to n do
      if x > a[j] then
        begin
          k := j;
          x := a[k];
        end;
    a[k] := a[i];
    a[i] := x;
  end;
```

Эффективность алгоритма:

- Количество сравнений

$$M = \frac{N}{2} \cdot (N-1) = \frac{N^2 - N}{2}$$

- Количество перемещений, когда массив упорядочен

$$C_{\min} = 3 \cdot (N-1)$$

- Количество перемещений когда массив обратно отсортирован

$$C_{\max} = C_{\min} \cdot \frac{N}{2} = 3 \cdot (N-1) \cdot \frac{N}{2}$$

В худшем случае сортировка прямым выбором дает порядок $n^2$, как и для числа сравнений, так и для числа перемещений.

**Задания**

Создать группу из N студентов. Ввести их: фамилия, имя, год рождения, оценки по предметам: стр. и алг.данных, высш. математика, физика, программирование, общий балл сдачи сессии; Разработать программу с использованием метода "прямого выбора", которая бы осуществляла сортировку (согласно варианту) :

1. Фамилий студентов по алфавиту.
2. Фамилий студентов по алфавиту в обратном порядке.
3. Студентов по старшинству (начиная со старшего).
4. Студентов по старшинству (начиная с младшего).
5. Студентов по общему баллу (по возрастанию).
6. Студентов по общему баллу (по убыванию).
7. Студентов по результатам 1-го экзамена (по возрастанию).
8. Студентов по результатам 2-го экзамена (по убыванию).
9. Студентов по результатам 3-го экзамена (по возрастанию).
10. Студентов по результатам 4-го экзамена (по убыванию).
11. Имен в алфавитном порядке.
12. Имен в обратном алфавитном порядке.

## Лабораторная работа № 8."СОРТИРОВКА С ПОМОЩЬЮ ПРЯМОГО ОБМЕНА"

Цель работы: исследовать и изучить методы сортировки с помощью прямого обмена.

Задача работы: овладеть навыками написания программ для сортировки с помощью прямого обмена на языке программирования ПАСКАЛЬ .

Порядок работы :

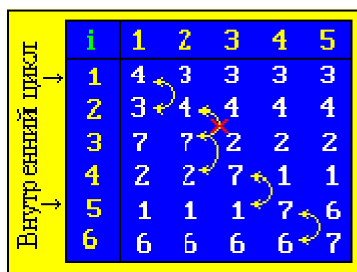
- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

Пузырьковая сортировка

**Идея :** (N-1) раз массив проходится снизу вверх (сверху вниз), при этом элементы попарно сравниваются, если нижний элемент меньше верхнего, то элементы переставляются.

**Пример :** массив - 4, 3, 7, 2, 1, 6.



Можно улучшить "пузырьковый" метод, если проходить массив элементов и вверх и вниз одновременно.

Количество сравнений :

$$M = \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$$

Количество перемещений :

$$C_{mzx} = 3 \cdot \frac{n^2}{4}$$

Пример вычисления "пузырьковым" методом

| 1 | | | | | 2 | | | | 3 | | | | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 4 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | -//- |
| 5 | 5 | 2 | 2 | 2 | 2 | 4 | 1 | 1 | 1 | 2 | 2 | 2 | -//- |
| 2 | 2 | 5 | 1 | 1 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | -//- |
| 1 | 1 | 1 | 5 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | -//- |
| 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | -//- |

На примере представлен массив из пяти элементов, это означает, что массив проходится снизу вверх (сверху вниз) 5-1=4 раза. Так же на примере видно, что алгоритм "в пустую" обрабатывает массив начиная уже с третьего шага внутреннего цикла, а 4-й шаг можно уже не выполнять.

Преимущества данного метода :

1) Самый простой алгоритм
2) Простой в реализации
3) Не нужно дополнительных переменных

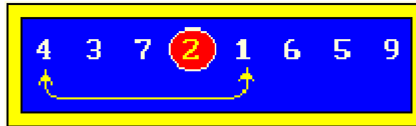
Недостатки:

1) Долго обрабатывает большие массивы
2) В любом случае количество проходов не уменьшается

Quiksort - метод быстрой сортировки

**Идея** : В основе данного метода положен метод разделение ключей по отношению к выбранному.

**Пример :**




После первого прохода выбранный элемент становится на свое место.

Улучшение "пузырькового" метода.

1) Если проходить массив не только сверху вниз, но и снизу вверх одновременно, то не только "легкие" элементы будут "всплывать", но и "тяжелые" "тонуть".

2) Для отмены прохода массива "впустую" нужно в во внешний цикл вставить проверку на отсортированность массива.

**Алгоритм**

**Алгоритм пузырькового метода**

(ПСЕВДОКОД)

```
for i=2 to n
  for j=n to i step -1
    if A( j-1 ) > A( j ) then
      x=A( j-1 )
      A( j-1 )=A( j )
      A( j )=x
    end if
next j
next i
```

**Алгоритм метода Quiksort**

(ПСЕВДОКОД)

```
Sub Sort(L,R)
  i=l
```

```
 j=r
 x=A(( l+r ) div 2 )
      while A( i ) < x do
        i=i+1
    end while
    while A( j )>x do
        j=j-1
    end while
    if i<=j then
       Y=A( i )
       A( i )=A( j )
       A( j )=Y
     while i>j do
         if l<j then
           sort( l,j )
         end if
         if i<r then
           sort( i,r )
         end if
 return
sub Quiksort
   sort ( 1, n )
return
```

Число перестановок и сравнений: (n* log( n )).

**Задания**

Варианты:

1. Составить программу вывода на экран самого большого (самого малого) элемента массива А.

2. Составить программу сортировки массива А по убыванию величин его элементов.

3. В массиве А находятся элементы. Составить программу, которая сформирует массив В, в котором расположить элементы масива В в порядке убывания.

4. Дан упорядоченный массив А - числа, расположенные в порядке возрастания, и число а, которое необходимо вставить в массив А, так, чтобы упорядоченность массива сохранилась.

5. Составить программу для быстрой перестройки данного массива А, в котором элементы расположены в порядке возрастания, так, чтобы после перестройки эти же элементы оказались расположенными в порядке убывания.

6. Дан массив А, содержащий как отрицательные, так и положительные числа. Составить программу исключения из него всех отрицательных чисел, а оставшиеся положительные расположить в порядке их возрастания.

7. Составить программу, которая будет из массива А брать одно число за другим и формировать из них массив В, располагая числа в порядке возрастания.

8. Дан список авторов в форме массива А. Составить программу формирования указателя авторов в алфавитном порядке и вывести его на экран.

9. Имеется n абонентов телефонной станции. Составить программу, в которой формируется список по форме: номер телефона, фамилия (номера идут в порядке возрастания).

10. Имеется n слов различной длины, все они занесены в массив А. Составить программу упорядочения слов по возрастанию их длин.

11. Составить программу для сортировки массива А по возрастанию и убыванию модулей его элементов.

12. В отсортированный массив А между каждой соседней парой элементов вставить число больше левого и меньше правого элемента в массиве и вывести полученный массив на экран.

## Лабораторная работа № 9. "СОРТИРОВКА С ПОМОЩЬЮ ДЕРЕВА"

Цель работы: исследовать и изучить методы сортировки с помощью дерева.

Задача работы: овладеть навыками написания программ для сортировки с помощью бинарного дерева на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

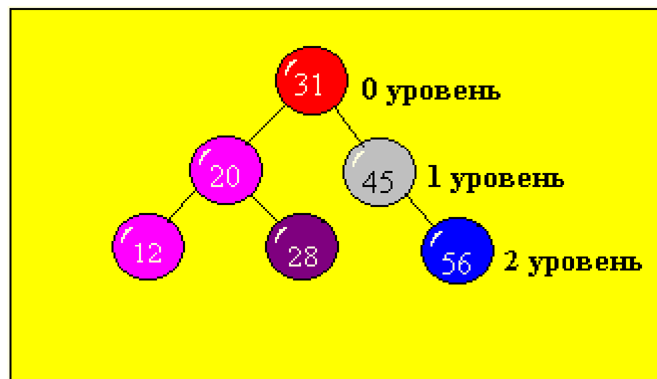
### Краткая теория

При обработке данных в ЭВМ важно знать и информационное поле элемента, и его размещение в памяти машины. Для этих целей используется сортировка. Итак, сортировка - это расположение данных в памяти в регулярном виде по их ключам. Регулярность рассматривают как возрастание значения ключа от начала к концу в массиве.

Различают следующие типы сортировок:

- внутренняя сортировка - это сортировка, происходящая в оперативной памяти машины
- внешняя сортировка - сортировка во внешней памяти.

Схематичное представление двоичного дерева : имеется набор вершин, соединенных стрелками. Из каждой вершины выходит не более двух стрелок (ветвей ), направленных влево - вниз или вправо - вниз. Должна существовать единственная вершина, в которую не входит ни одна стрелка - эта вершина называется корнем дерева. В каждую из оставшихся вершин входит одна стрелка.




Существует множество способов упорядочивания дерева. Рассмотрим один из них :

" Левый " сын имеет ключ меньше, чем ключ " Отца "

" Правый " сын имеет ключ больше, чем ключ " Отца "

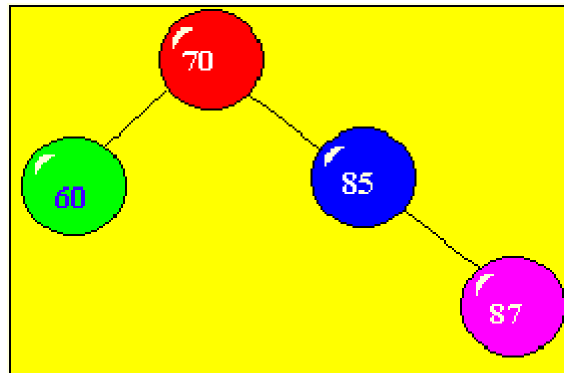
Получили бинарное упорядоченное дерево с минимальным числом уровней.

Строго сбалансированное дерево : дерево, в котором левое и правое поддерево имеют уровни , отличающиеся не более, чем на единицу.

Рассмотрим принцип построения дерева при занесении элементов в массив :

Пусть в первоначально пустой массив заносятся последовательно поступающие элементы : 70, 60, 85, 87, 90, 45, 30, 88, 35, 20, 86 ; Т.к. это еще пустое дерево, то ссылки left и right равны nil( left - указатель на левого сына (элемент), right - указатель на правого сына (элемент)).

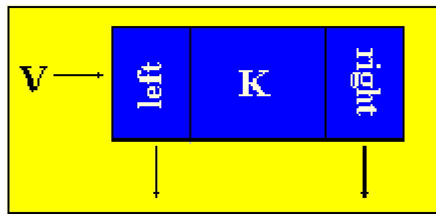
Четвертый из поступающих элементов 87. Т.к. этот ключ больше ключа 70 ( корень ), то новая вершина должна размещаться на правой ветви дерева. Чтобы определить ее место, спускаемся по правой стрелке к очередной вершине ( с ключом 85 ), и если поступивший ключ больше 85, то новую вершину делаем правым сыном вершины с ключом 85 .




В рассмотренном примере ПРИНЦИП ПОСТРОЕНИЯ ДЕРЕВА имеет следующий вид : если поступает очередной элемент массива, то начиная с корня дерева (в зависимости от сравнения текущего элемента с очередной вершиной) идем влево или вправо от нее до тех пор, пока не найдем подходящую вершину, к которой можно подключить новую вершину с текущим ключом. В зависимости от результата сравнения ключей, новую вершину делаем левой или правой для найденной вершины.

**Алгоритм**

Для создания дерева необходимо создать в памяти элемент следующего типа :




Тип на ПАСКАЛе :

```
type
pelem = ^elem;
elem  = record
          left : pointer;
          right : pointer;
          K : integer;
end;
```

K - элемент массива, V - указатель на созданный элемент.

В процедуре создания дерева бинарного поиска будут использованы следующие указатели :

tree - указатель на корень дерева;

p - рабочий указатель;

q - указатель отстающий на шаг от p;

key - новый элемент массива;

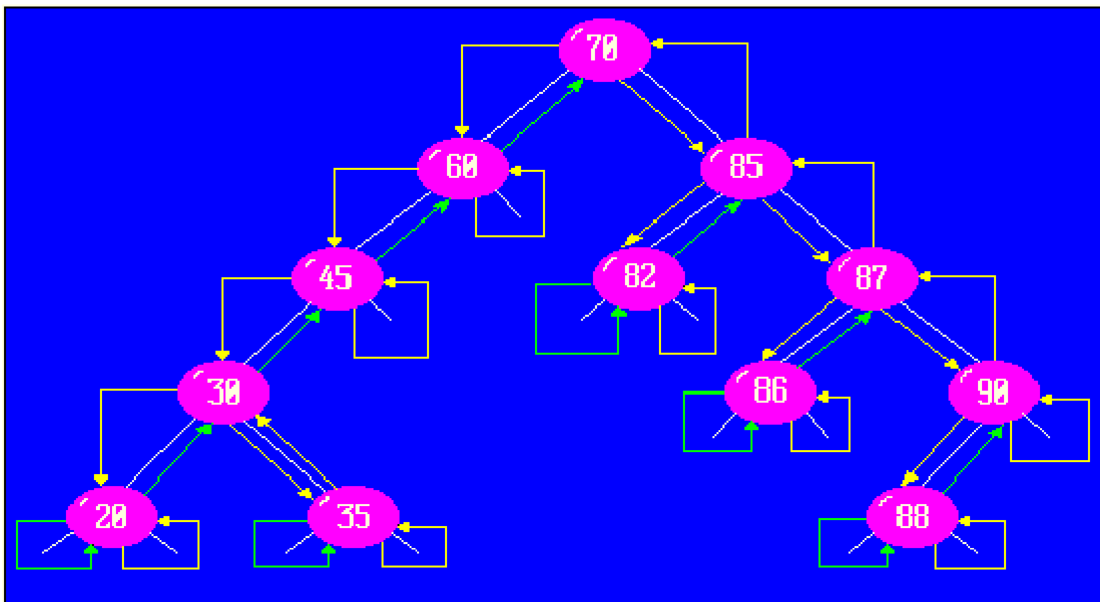
**Создание дерева бинарного поиска :**

(псевдокод)

```
writeln(' Введите элемент массива ');
readln(key);
tree:=maketree(key);
p:=tree;
while not eof do
  begin
    readln(key);
    v:=maketree(key);
    while p<>nil do
       begin
```

```
            q:=p;
            if key<k(p) then p:=left(p)
            else p:=right(p);
        end;
        if p=nil then
          begin
             writeln('Это корень');
             tree:=v;
          end
          else
             if key<k(q) then left(p):=v
             else right(p):=v;
end;
```

При обходе дерева слева - направо получаем отсортированный массив 20, 30, 35, 45, 60, 70, 82, 85, 86, 87, 88, 90. Элемент дерева заносится в массив при втором заходе в него (на рисунке вторые заходы показаны зелеными стрелками).




**Обход дерева слева - направо**

```
procedure InTree(tree);
 begin
```

```
        if Tree = nil then write('Дерево пусто')
        else
           with Tree^ do
             begin
                if left<>nil then InTree(left);
                if right<>nil then InTree(right);
             end;
  end;
```

**Задания**

Вариант 1: На заводе выпустили детали со следующими серийными номерами : 45, 56, 13, 75, 14, 18, 43, 11, 52, 12, 10, 36, 47, 9. Детали с четными номерами поступают на склад №1, а с нечетными на слад №2. Требуется отсортировать детали на складе №1.

Вариант 2 : Угнали автомобиль. Свидетель запомнил, что первой цифрой номера была 4. В базе угнанных автомобилей в этот день были следующие номера : 456, 124, 786, 435, 788, 444, 565, 127, 458, 322, 411, 531, 400, 546, 410. Нужно составить список номеров начинающихся на 4 и упорядочить его по возрастанию.

Вариант 3 : За неделю езды в транспорте накопились билеты с номерами 124512, 342351, 765891, 453122, 431350, 876432, 734626, 238651, 455734, 234987. Нужно отобрать "счастливые" билеты и расположить их по возрастанию.

Вариант 4 : Дан список людей с указанием их возраста. Для составления графика ухода сотрудников на пенсию требуется составить новый список новый список в том порядке, в каком они будут уходить на пенсию.

Вариант 5 : Студенты сдали пять экзаменов. Нужно отсортировать список студентов по возрастанию общего балла по результатам сданных экзаменов.

Вариант 6 : В городе был один автобусный парк, куда приезжали автобусы с номерами : 11, 32, 23, 12, 6, 52, 47, 63, 69, 50, 43, 28, 35, 33, 42, 56, 55, 101. После строительства второго автопарка решили перевести туда автобусы с нечетными номерами. Для того чтобы составить расписание их движения нужно организовать список номеров автобусов второго парка, упорядочив их по убыванию.

Вариант 7 : Была составлена ведомость по зарплате, представленная в виде : Иванов - 166000, Сидоров - 180000, ... Требуется упорядочить этот список таким образом, чтобы размер зарплаты уменьшался.

Вариант 8 : На стоянке стоят автомобили со следующими номерами : 1212, 3451, 7694, 4512, 4352, 8732, 7326, 2350, 4536, 2387, 5746, 6776, 4316, 1324. Для статистики необходимо составить список автомобилей с такими номерами, сумма первых двух цифр которых равна сумме двух последних цифр, так чтобы каждый следующий номер был меньше предыдущего.

Вариант 9 : Выпустили лотерейные билеты с четырехзначными номерами. Выигрышными считаются те билеты, сумма цифр которых делится на 4. Составить список выигрышных билетов, упорядоченных по убыванию.

Вариант 10 : Молодой человек взял номер телефона у своей знакомой, но забыл его. Он смог вспомнить только первые три цифры : 469***. В его записной книжке были следующие номера телефонов : 456765, 469465, 469321, 616312, 576567, 469563, 567564, 469129, 675665, 469873,

569090, 469999, 564321, 469010. Составить список номеров начинающихся с цифр 469 и упорядочить их по убыванию.

Вариант 11 : Студенты сдали пять экзаменов. Нужно отсортировать список студентов по убыванию общего балла по результатам сданных экзаменов.

Вариант 12 : Выпустили лотерейные билеты с четырехзначными номерами. Выигрышными считаются те билеты, сумма первых трех цифр которых равна 8. Составить список выигрышных билетов, упорядоченных по возрастанию.

## Лабораторная работа № 10. "ИССЛЕДОВАНИЕ МЕТОДОВ ЛИНЕЙНОГО И БИНАРНОГО ПОИСКА"

Цель работы: изучить методы линейного и бинарного поиска.

Задача работы: овладеть навыками написания программ для методов линейного и бинарного поиска на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

### ПОИСК

Одно из наиболее часто встречающихся в программировании действий - поиск. Он же представляет собой идеальную задачу, на которой можно испытывать различные структуры данных по мере их появления. Существует несколько основных "вариаций этой темы", и для них создано много различных алгоритмов. При дальнейшем рассмотрении мы исходим из такого принципиального допущения: группа данных, в которой необходимо отыскать заданный элемент, фиксирована. Будем считать, что множество из N элементов задано, скажем, в виде такого массива

a: ARRAY[0..N-1] OF item

Обычно тип item описывает запись с некоторым полем, выполняющим роль ключа. Задача заключается в поиске элемента, ключ которого равен заданному "аргументу поиска" x. Полученный в результате индекс i, удовлетворяющий условию a[i].key=x, обеспечивает доступ к другим полям обнаруженного элемента. Так как нас интересует в первую очередь сам процесс поиска, а не обнаруженные данные, то мы будем считать, что тип item включает только ключ, т.е. он есть ключ (key).

### Алгоритм

### Линейный поиск

Если нет никакой дополнительной информации о разыскиваемых данных, то очевидный подход - простой последовательный просмотр массива с увеличением шаг за шагом той его части, где желаемого элемента не обнаружено. Такой метод называется линейным поиском. Условия окончания поиска таковы:

1. Элемент найден, т.е. ai = x.
2. Весь массив просмотрен и совпадения не обнаружено.

Это дает нам линейный алгоритм:

```
i := 0;
WHILE (i < N) AND (a[i] <> x) DO
  i := i+1 ;
END;
```

Обратите внимание, что порядок элементов в логическом выражении имеет существенное значение. Инвариант цикла, т.е. условие, выполняющееся перед каждым увеличением индекса i, выглядит так:

$(0 \le i < N)$ AND $(A_k : 0 \le k < i : a_k \ne x)$

Он говорит, что для всех значений k, меньших чем i, совпадения не было. Отсюда и из того факта, что поиск заканчивается только в случае ложности условия в заголовке цикла, можно вывести окончательное условие:

$((i = N) \text{ OR } (a_i = x)) \text{ AND } (A_k : 0 \le k < i : a_k \ne x)$

Это условие не только указывает на желаемый результат, но из него же следует, что если элемент найден, то он найден вместе с минимально возможным индексом, т.е. это первый из таких элементов. Равенство i = N свидетельствует, что совпадения не существует.

Совершенно очевидно, что окончание цикла гарантировано, поскольку на каждом шаге значение i увеличивается, и, следовательно, оно, конечно же, достигнет за конечное число шагов предела N; фактически же, если совпадения не было, это произойдет после N шагов.

Ясно, что на каждом шаге требуется увеличивать индекс и вычислять логическое выражение. А можно ли эту работу упростить и таким образом убыстрить поиск ?

Единственная возможность - попытаться упростить само логическое выражение, ведь оно состоит из двух членов. Следовательно, единственный шанс на пути к более простому решению - сформулировать простое условие, эквивалентное нашему сложному. Это можно сделать, если мы гарантируем, что совпадение всегда произойдет. Для этого достаточно в конец массива поместить дополнительный элемент со значением x. Назовем такой вспомогательный элемент "барьером", ведь он охраняет нас от перехода за пределы массива. Теперь массив будет описан так:

```
a: ARRAY[0..N] OF INTEGER
```

и алгоритм линейного поиска с барьером выглядит следующим образом:

```
a[N] := x;
i := 0;
WHILE a[i] <> x DO
  i := i+1;
```

```
END;
```

Результирующее условие, выведенное из того же инварианта, что и прежде:

$(a_i=x)$ AND $(A_k : 0 \le k < i : a_k \ne x)$

Ясно, что равенство i = N свидетельствует о том, что совпадения (если не считать совпадения с барьером) не было.

**Поиск делением пополам (двоичный поиск).**

Совершенно очевидно, что других способов убыстрения поиска не существует, если, конечно, нет еще какой-либо информации о данных, среди которых идет поиск. Хорошо известно, что поиск можно сделать значительно более эффективным, если данные будут упорядочены. Вообразите себе телефонный справочник, в котором фамилии не будут расположены по порядку. Это нечто совершенно бесполезное! Поэтому мы приводим алгоритм, основанный на знании того, что массив a упорядочен, т.е. удовлетворяет условию

$$A_k : 1 \le k < N : a_{k-1} \ne a_k$$

Основная идея - выбрать случайно некоторый элемент, предположим am, и сравнить его с аргументом поиска x. Если он равен x, то поиск заканчивается, если он меньше x, то мы заключаем, что все элементы с индексами, меньшими или равными m, можно исключить из дальнейшего поиска; если же он больше x, то исключаются индексы больше и равные m. Это соображение приводит нас к следующему алгоритму (он называется "поиском делением пополам"). Здесь две индексные переменные L и R отмечают соответственно левый и правый конец секции массива a, где еще может быть обнаружен требуемый элемент.

```
L := 0;
R := N-1;
found := FALSE;
WHILE (L J R) AND NOT found DO
   m := любое значение между L и R;
```

```
      IF a[m] = x THEN found := TRUE;
      IF a[m] < x THEN L := m+1
      ELSE R := m-1;
      END;
   END;
```

Инвариант цикла, т.е. условие, выполняющееся перед каждым шагом, таков:

$$(L \le R) \text{ AND } (A_k : 0 \le k < L : a_k < x) \text{ AND } (A_k : R < k < N : a_k > x)$$

из чего выводится результат

$$\text{found OR } ((L > R) \text{ AND } (A_k : 0 \le k < L : a_k < x) \text{ AND } (A_k : R < k < N : a_k > x))$$

откуда следует

$$(a_m = x) \text{ OR } (A_k : 0 \le k < N : ak \ne x)$$

Выбор m совершенно произволен в том смысле, что корректность алгоритма от него не зависит. Однако на его эффективность выбор влияет. Ясно, что наша задача - исключить на каждом шагу из дальнейшего поиска, каким бы ни был результат сравнения, как можно больше элементов. Оптимальным решением будет выбор среднего элемента, так как при этом в любом случае будет исключаться половина массива. В результате максимальное число сравнений равно log N, округленному до ближайшего целого. Таким образом, приведенный алгоритм существенно выигрывает по сравнению с линейным поиском, ведь там ожидаемое число сравнений - N/2.

Эффективность можно несколько улучшить, поменяв местами заголовки условных операторов. Проверку на равенство можно выполнять во вторую очередь, так как она встречается лишь единожды и приводит к окончанию работы. Но более существенно следующее соображение: нельзя ли, как и при линейном поиске, отыскать такое решение, которое опять бы упростило условие окончания. И мы действительно находим такой быстрый алгоритм, как только отказываемся

от наивного желания кончить поиск при фиксации совпадения. На первый взгляд это кажется странным, однако при внимательном рассмотрении обнаруживается, что выигрыш в эффективности на каждом шаге превосходит потери от сравнения с несколькими дополнительными элементами. Напомним, что число шагов в худшем случае - log N. Быстрый алгоритм основан на следующем инварианте:

$(A_k : 0 \le k < L : a_k < x)$ AND $(A_k : R \le k < N : a_k \ge x)$

причем поиск продолжается до тех пор, пока обе секции не "накроют" массив целиком.

```
L := 0;
R := N;
  WHILE L < R DO
    m := (L+R) DIV 2;
    IF a[k] < x THEN L := m+1
    ELSE R := m ;
  END
END
```

Условие окончания - L i R, но достижимо ли оно? Для доказательства этого нам необходимо показать, что при всех обстоятельствах разность R-L на каждом шаге убывает. В начале каждого шага L < R. Для среднего арифметического m справедливо условие L J m < R. Следовательно, разность действительно убывает, ведь либо L увеличивается при присваивании ему значения m+1, либо R уменьшается при присваивании значения m. При L = R повторение цикла заканчивается. Однако наш инвариант и условие L = R еще не свидетельствуют о совпадении. Конечно, при R = N никаких совпадений нет. В других же случаях мы должны учитывать, что элемент a[R] в сравнениях никогда не участвует. Следовательно, необходима дополнительная проверка на равенство a[R] = x. В отличие от первого нашего решения приведенный алгоритм, как и в случае линейного поиска, находит совпадающий элемент с наименьшим индексом.

**Задания**

Варианты:

1.Найти наименьший элемент в массиве А с помощью линейного поиска.

2.Поиск элементов в массиве А, которые больше 30.

3.Вывести на экран все числа массива А кратные 3 (3,6,9,...) с помощью линейного поиска.

4.Найти все элементы, модуль которых больше 20 и меньше 50, с помощью линейного поиска.

5.Вывести на экран все числа массива А кратные 4 (4,8,...) с помощью линейного поиска.

6.Вывести на экран сообщение, каких чисел больше относительно 50, с помощью линейного поиска.

7.Найти элемент в массиве А и найти число сравнений с помощью линейного поиска.

8.Поиск элементов случайным образом с помощью бинарного поиска.

9.Дан список номеров машин (345, 368, 876, 945, 564, 387, 230), найти, на каком месте стоит машина с заданным номером, бинарный поиск.

10.Поиск каждого второго элемента в списке и число сравнений.

11.Найти элемент с заданным ключом с помощью бинарного поиска.

## Лабораторная работа №11. "ИССЛЕДОВАНИЕ МЕТОДОВ ОПТИМИЗАЦИИ ПОИСКА "

Цель работы: исследовать и изучить методы поиска с перемещением в начало и транспозицией.

Задача работы: овладеть навыками написания программ по исследованию методов поиска с перемещением в начало и транспозицией на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

ПОИСК (SEARCH) является одной из основ обработки данных в ЭВМ. Его назначение состоит в том, чтобы по заданному аргументу найти среди массива данных те данные, которые соответствуют этому аргументу или определить, что этих данных нет. Если этих данных нет, добавить их в массив данных.

Набор любых данных будем называть ТАБЛИЦЕЙ или ФАЙЛОМ. Данное или элемент структуры отличается каким-то признаком от других данных. Этот признак называется КЛЮЧОМ. Ключ может быть УНИКАЛЬНЫМ, т.е. в табли-

це только одно данное с таким ключом, иначе уникальные ключи называются ПЕРВИЧНЫМИ КЛЮЧАМИ.

ВТОРИЧНЫЙ КЛЮЧ в одной таблице может повторяться, но по нему тоже можно организовать поиск. Ключи данных собраны в одном месте (таблице).

Ключи, которые выделены из таблицы данных и организованы в свой файл, называются ВНЕШНИМИ КЛЮЧАМИ. Если ключ в записи, то он называется ВНУТРЕННИМ КЛЮЧОМ.

ПОИСКОМ ПО ЗАДАННОМУ АРГУМЕНТУ называется алгоритм, определяющий соответствие ключа данного с заданным аргументом. Результатом работы алгоритма поиска может быть нахождение этого данного или отсутствие данного в таблице. В случае отсутствия данного возможны две операции:

1. Индикация того, что данного нет.
2. Вставка данного в таблицу.

Пусть K - массив ключей, тогда для всех K(i) существует R(i) - данное. KEY - аргумент поиска. Ему соответствует информационная запись REC. В зависимости от того, какова структура данных в таблице, различают несколько видов поиска.

## ПЕРЕУПОРЯДОЧЕНИЕ ТАБЛИЦЫ ПОИСКА

Всегда можно говорить о некотором значении вероятности нахождения того или иного элемента.

P(i) - вероятность нахождения элемента.

В этом случае таблица поиска представляется как система с дискретными состояниями, а искомый элемент характеризуется i-ым состоянием системы, вероятность которого P(i).

$$P(1) + P(2) + ... + P(n) = 1$$

Количество сравнений Z при поиске в таблице, т.е. количество сравнений, необходимых для поиска по заданному аргументу, представляет собой значение дискретной случайной

величины, определяемой номерами состояний и вероятностями состояний системы.

$$Z=1*P(1) + 2*P(2) + 3*P(3) + ... + n*P(n)$$

ТАБЛИЦА ДАННЫХ ДОЛЖНА БЫТЬ УПОРЯДОЧЕНА ТАКИМ ОБРАЗОМ , чтобы в начале таблицы были элементы с большими вероятностями поиска или элементы , к которым чаще всего обращаются в таблице.

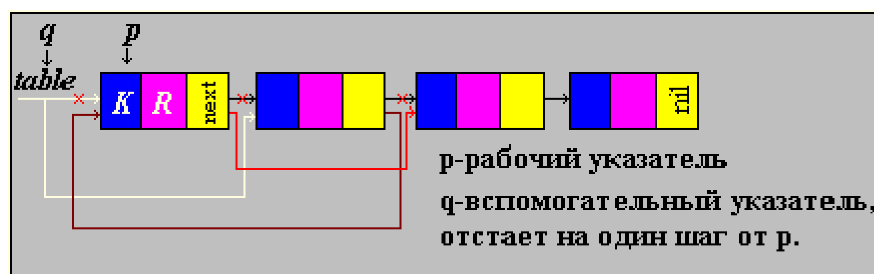
$$P(1) >= P(2) >= P(3) >= ... >= P(n)$$

ИМЕЕТСЯ ДВА ОСНОВНЫХ МЕТОДА ПЕРЕУПОРЯДОЧЕНИЯ ТАБЛИЦ ПОИСКА:

1. Переупорядочение путем перестановки найденного элемента в начало списка.
2. Метод транспозиции.

**Алгоритм**

**Переупорядочение путем перестановки в начало списка**




Найденный элемент сразу оказывается в голове списка.

Первоначально указатель q нулевой, указатель p указывает на начало списка; p перемещается на второй элемент, а q на первый. Указатель начала списка (table) перемещается на второй элемент, а указатель второго элемента на третий. Таким образом второй элемент перемещается на первое место.

```
(ПСЕВДОКОД)
  q=nil
  p=table
  WHILE (p <> nil) do
    IF  key=k(p)
          then  SEARCH=p
          next(q)=next(p)
          next(p)=q
          table=p
          return
    end  IF
    q=p
    p=next(p)
  end WHILE
  SEARCH=0
  return
```

**Метод транспозиции**

В данном методе найденный элемент переставляется на один элемент к голове списка. Если к этому элементу обращаются часто, то постепенно перемещаясь к началу списка, он скоро окажется в его голове.

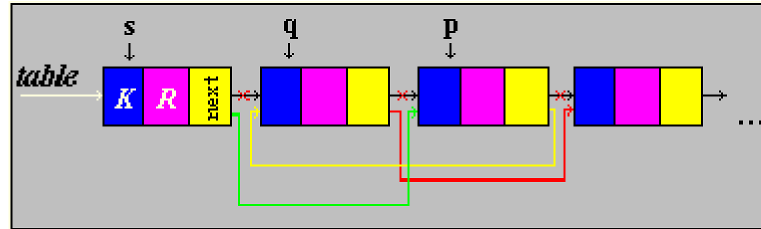
Этот метод удобен тем, что он эффективен не только в списковых структурах, но и в неупорядоченных массивах, т.к. меняются местами только два рядом стоящих элемента.

Будем использовать три указателя:

p - рабочий указатель, указывает на элемент.

q - вспомогательный указатель, отстает на один шаг от p.

s - вспомогательный указатель, отстает на два шага от p.




Найденный нами третий элемент передвигается на один шаг к голове списка (т.е. становится вторым). Указатель первого элемента перемещается на третий элемент, указатель второго элемента на четвертый, таким образом третий элемент перемещается на второе место. Если к данному элементу обратиться еще раз, то он окажется в голове списка.

```
s=nil
q=nil
p=nil
while (p<>nil) do
 if key=k(p) then search=p
 if q=nil then return
 else next(q)=next(p)
       next(p)=q
        if s=nil then table
        else next(s)=p
       end if
       search=p
 end if
 end if
 return
end while
search=nil return
```

**Задания**

Дан массив целых чисел. Составить процедуры метода транспозиции и перестановки в начало. Решить заданную за-

дачу и переставить найденный в задаче элемент обоими методами в начало списка.

ВАРИАНТ 1. Найти максимальный элемент массива.

ВАРИАНТ 2. Найти минимальный элемент массива.

ВАРИАНТ 3. Найти число, нацело делящееся на 11 ( если таких чисел несколько, то найти максимальное; если таких чисел нет - выдать сообщение ).

ВАРИАНТ 4. Найти число, нацело делящееся на 11 ( если таких чисел несколько, то найти максимальное; если таких чисел нет - выдать сообщение ).

ВАРИАНТ 5. Найти элемент, разность соседних элементов которого не меньше 72. Если таких элементов несколько, выбрать максимальный. Если таких элементов нет, выдать сообщение.

ВАРИАНТ 6. Найти элемент, частное соседних элементов которого четное число. Если таких элементов несколько, выбрать максимальный или минимальный элемент. Если таких элементов нет, выдать сообщение.

ВАРИАНТ 7. Найти элемент, разность соседних элементов которого четное число. Если таких элементов несколько, выбрать максимальный или минимальный элемент. Если такого элемента нет, выдать сообщение.

ВАРИАНТ 8. Найти элемент, среднее арифметическое элементов, находящихся до этого элемента равно 12. Если таких элементов нет, выдать сообщение.

ВАРИАНТ 9. Найти максимальный элемент, делящийся на 10. Если такого элемента нет, выдать сообщение.

ВАРИАНТ 10. Найти элемент, разность соседних элементов которого четное число и делится на 3. Если такого элемента нет, выдать сообщение.

ВАРИАНТ 11. Найти элемент, среднее квадратичное элементов, находящихся после этого элемента меньше 10. Если таких элементов несколько, выбрать максимальный элемент. Если таких элементов нет, выдать сообщение.

ВАРИАНТ 12. Найти значение tg (x) от каждого элемента и переставить на 1 место элемент, значение функции от которого максимально.

## Лабораторная работа № 12. "ПОИСК ПО ДЕРЕВУ С ВКЛЮЧЕНИЕМ"

Цель работы: освоить алгоритм и метод вставки элементов бинарного дерева.

Задача работы: овладеть навыками написания программ по исследованию методов вставки элементов бинарного дерева на языке программирования ПАСКАЛЬ .

Порядок работы :

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

### Краткая теория

ПОИСК ПО БИНАРНОМУ ДЕРЕВУ

Для вставки элемента в дерево сначала нужно осуществить поиск в дереве по заданному ключу. Если такой ключ имеется, то программа завершается, если нет то происходит вставка. Длительность операции поиска (число узлов, которые надо перебрать для этой цели) зависит от структуры дерева. Действительно, деревом может быть вырождено в однонаправленный список (иметь единственную ветвь) - такое дерево может возникнуть, если элементы поступали в дерево в порядке возрастания (убывания) их, ключей. В этом случае

время поиска будет такое же, как и однонаправленном списке, т.е. в среднем придется перебрать N/2 элементов. Наибольший эффект использования дерева достигается в том случае, когда дерево "сбалансировано". В этом случае для поиска придется перебрать не более $\log_2 N$.

ВКЛЮЧЕНИЕ ЭЛЕМЕНТА В ДЕРЕВО

Для включения новой записи в дерево, прежде всего нужно найти тот узел, к которому можно "подвесить"(присоединить) новый элемент. Алгоритм поиска нужного узла тот же самый, что и при поиске узла с заданным ключом. Этот узел будет найден в тот момент, когда в качестве очередной ссылки, определяющий ветвь дерева, в которое надо продолжить поиск, окажется ссылка NIL.

Однако, непосредственно использовать процедуру поиска нельзя, потому что по окончанию вычисления ее значение не фиксирует тот узел, из которого была выбрана ссылка NIL. Модифицируем описание процедуры поиска так, чтобы в качестве ее побочного эффекта фиксировалась ссылка на узел, в котором был найден заданный ключ (в случае успешного поиска), или ссылка на узел, после обработки которого поиск прекращен (в случае неуспешного поиска).

**Алгоритм**

Рассмотрим алгоритм вставки узла в бинарное дерево.

Вставим узел с номером 150, тогда он станет правым сыном узла с номером 120, т.к. он является большим по значению узла с номером 120, но меньше значения узла головы дерева.

P - рабочий указатель

Q - указатель отстающий от P на один шаг

V - указатель на элемент, который будет вставлен в бинарное дерево .

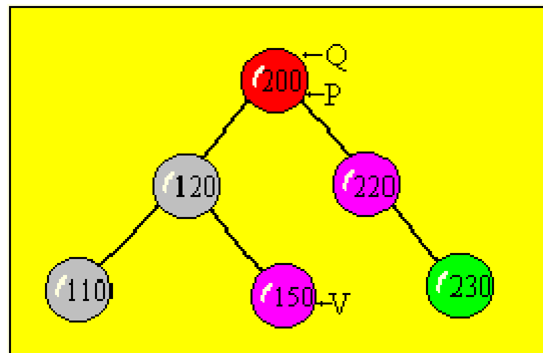
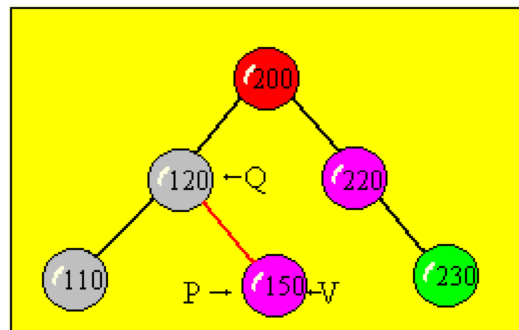
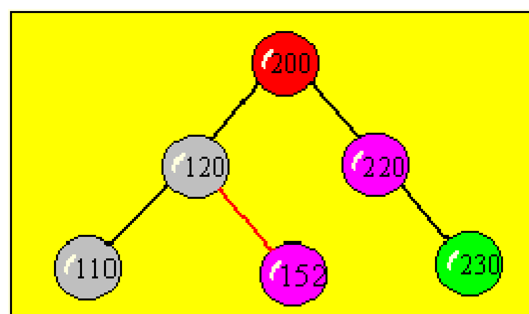



Иллюстрация процесса вставки узла 150, в соответствии с вышеприведенным алгоритмом (красным цветом выделены новые связи в дереве).




Конечный вариант дерева после вставки :




Программа

| Псевдокод | Паскаль |
|---|---|
| Q=nil | Q=nil |
| P=Tree | P=Tree |

```
While (P<>nil) do
    Q=P
    If key=k(P) then
        search=P
        return
    EndIf
    If key<k(P)  then
        P=left(P)
    else
        P=right(P)
    EndIf
EndWhile
V=maketree(key,rec)
If key<k(Q) then
else
    Right(Q)=V
EndIf
search=V
Return
```

```
While (P<>nil) do
Begin
    Q=P;
    If key=P^.k then
    Begin
        search:=P;
        exit;
    End;
    If key<P^.k then
        P:=P^.left;
    else
        P:=P^.right;
End;
V=maketree(key,rec)
If key<Q^.k then
    Q^.left:=V
else
    Q^.right:=V;
search:=V
```

### Задания

Используя генератор случайных чисел сформировать бинарное дерево, состоящее из 5 элементов (предусмотреть ручной ввод элементов). Причем числа должны лежать в диапазоне от -99 до 99. Произвести поиск с вставкой элементов в соответствии со следующими вариантами заданий:

1. Числа кратные N.
2. Нечетные числа.
3. Числа > N.
4. Простые числа.
5. Числа по выбору.
6. Случайное число.

7. Составные числа.

8. Числа в интервале от X до Y.

9. Числа, сумма цифр (по модулю) которого > N.

10. Числа, сумма цифр (по модулю) которого < N.

11. Числа, сумма цифр (по модулю) которого лежит в интервале от X до Y.

12. Числа, взятые по модулю, квадратный корень которых целое число.

где: N, X, Y - задается преподавателем.

## Лабораторная работа № 13. "ПОИСК ПО ДЕРЕВУ С ИСКЛЮЧЕНИЕМ"

<u>Цель работы:</u> исследовать и изучить методы поиска с помощью дерева.

<u>Задача работы:</u> овладеть навыками написания программ для поиска с помощью бинарного дерева на языке программирования ПАСКАЛЬ .

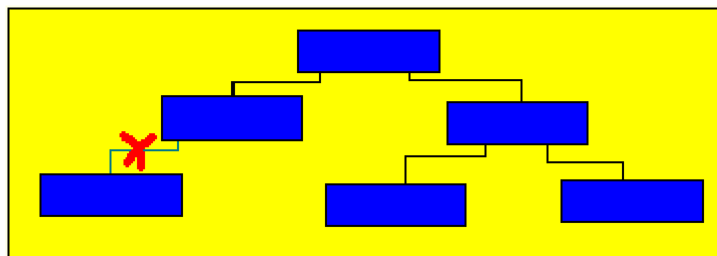
<u>Порядок работы :</u>

- изучить описание лабораторной работы;
- по заданию, данному преподавателем, разработать алгоритм программы решения задачи;
- написать программу на языке ПАСКАЛЬ;
- отладить программу;
- решить задачу;
- оформить отчет.

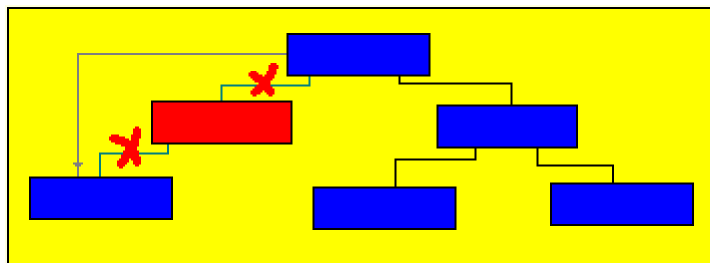
### Краткая теория

Удаление узла дерева не должно нарушать упорядоченность дерева. При удалении возможны следующие варианты расположения узлов:

1) найденный узел является листом - он просто удаляется (рис.1);



2) найденный узел имеет только сына - в этом случае сын перемещается на место удаленного отца (рис.2);



3) у удаляемого отца два сына - в этом случае основная трудность состоит в удалении отца, поскольку в удаляемую вершину входит одна стрелка, а выходит две.

В этом случае нужно найти подходящее звено поддерева, которое можно было бы вставить на место удаляемого, причем это подходящее звено должно просто перемещаться. Такое звено всегда существует:

- это самый правый элемент левого поддерева (для достижения этого звена необходимо перейти в следующую вершину по левой ветви, а затем переходить в очередные вершины только по правой ветви до тех пор, пока очередная ссылка не будет равна nil);

- это самый левый элемент правого поддерева (для достижения этого звена необходимо перейти в следующую вершину по правой ветви, а затем переходить в очередные вершины только по левой ветви до тех пор, пока очередная такая ссылка не будет равна nil).

Очевидно, что такие подходящие звенья могут иметь не более одной ветви.

**Алгоритм**

Рассмотрим алгоритм в котором вместо удаляемого узла ставится самый левый узел правого поддерева. Удалим узел с номером 150, тогда на его место станет элемент под номе-

ром 152, т.к. он является самым левым из правого поддерева.

Введем следующие обозначения:

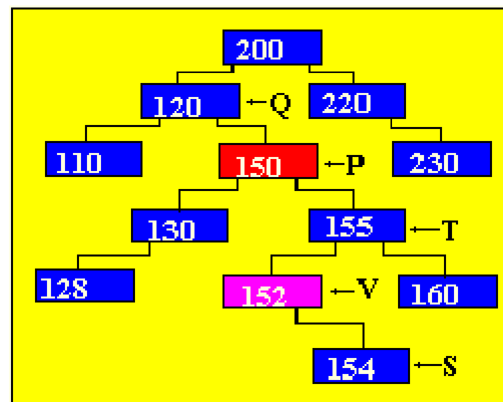
P - рабочий указатель

Q - указатель отстающий от P на один шаг

V - указатель на элемент, который будет замещать удаляемый узел

T - указатель, отстает от V на один шаг

S - указатель, опережает V на один шаг




Пример программы удаления элементов бинарного дерева

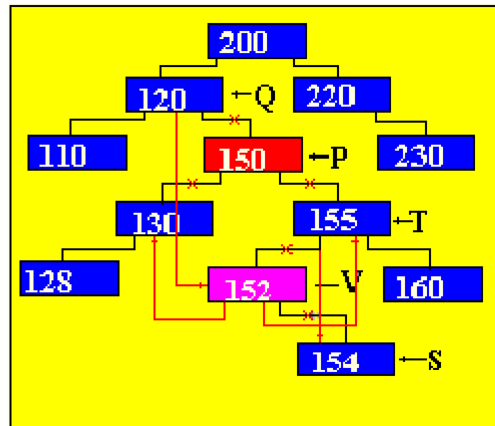
(Псевдокод)

```
Q=nil
 P=Tree
 While (P<>nil)and(K(P)<>key)do {поиск узла с нужным ключем}
   If key<K(P) then P=Left(P)
     Else P=Right(P)
   EndIf
 EndWhile
 If P=nil then "Ключ не найден" {проверка если такого элемента нет}
               Return        {выход}
```

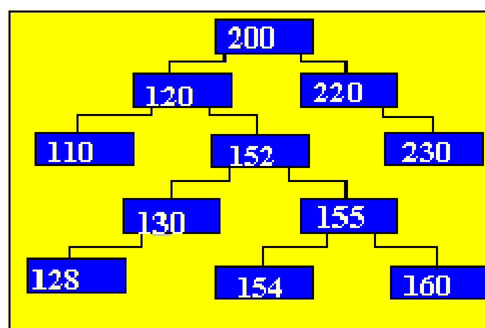
```
    EndIf
    If Left(P)=nil then V=Right(P) {если левая ссылка равна
nil
     Else
      If Right(P)=nil then V=Left(P)
        Else
          GetNode(P)
          T=P
          V=Right(P) S = Left(V)
    While S <> nil  {поиск самого }

     T = V        {левого эл-нта}
     V = S     {правого поддерева}
     S = Left(V)
    EndWhile
    If T <> P then
     "V-не сын"
     Left(T) = Right(V)
     Right(V)= Right(P)
    EndIf
    Left(V) = Left(P)
    If Q = nil then
     "p - корень"
     Tree = V
     Return
    EndIf
        If P = Left(Q) then
          Left(Q) = V
        Else
          Right(Q)= V
        EndIf
      EndIf
     EndIf
    FreeNode(P)
   Return
```

Иллюстрация процесса удаления узла 150, в соответствии с вышеприведенным алгоритмом (красным цветом выделены новые связи в дереве).




Конечный вариант дерева после удаления




## Задания

Используя генератор случайных чисел сформировать бинарное дерево, состояшее из 15 элементов (предусмотреть ручной ввод элементов). Причем числа должны лежать в диапазоне от -99 до 99. Произвести поиск с удалением элементов в соответствии со следую-щими вариантами заданий:

1. Числа кратные N.
2. Нечетные числа.
3. Числа > N.
4. Числа < N.

5. Числа по выбору.

6. Простые числа.

7. Составные числа.

8. Числа в интервале от X до Y.

9. Числа, сумма цифр (по модулю) которого > N.

10. Числа, сумма цифр (по модулю) которого < N.

11. Числа, сумма цифр (по модулю) которого лежит в интервале от X до Y.

12. Числа, взятые по модулю, квадратный корень которых целое число.

где: N, X, Y - задается преподавателем.

## ТЕСТЫ К ЛАБОРАТОРНЫМ РАБОТАМ

Лабораторная работа 1. Полустатические структуры данных (стеки).

1. В чём особенности очереди ?
   - открыта с обеих сторон ;
   - открыта с одной стороны на вставку и удаление;
   - доступен любой элемент.
2. В чём сосбенности стека ?
   - открыт с обеих сторон на вставку и удаление;
   - доступен любой элемент;
   - открыт с одной стороны на вставку и удаление .
3. Какую дисциплину обслуживания принято называть FIFO?
   - стек;
   - очередь ;
   - дек.
4. Какая операция читает верхний элемент стека без удаления?
   - pop;
   - push;
   - stackpop .
5. Какого правило выборки элемента из стека ?
   - первый элемент;
   - последний элемент ;
   - любой элемент.

Лабораторная работа 2.Списковые структуры данных (односвязные очереди).

1. Как освободить память от удаленного из списка элемента ?
   - p=getnode;
   - ptr(p)=nil;

- freenode(p) ;
- p=lst.

2. Как создать новый элемент списка с информационным полем D ?
   - p=getnode;
   - p=getnode; info(p)=D ;
   - p=getnode; ptr(D)=lst.
3. Как создать пустой элемент с указателем p ?
   - p=getnode ;
   - info(p);
   - freenode(p);
   - ptr(p)=lst.
4. Сколько указателей используется в односвязных списках ?
   - 1 ;
   - 2;
   - сколько угодно.
5. В чём отличительная особенность динамических объектов?
   - порождаются непосредственно перед выполнением программы;
   - возникают уже в процессе выполнения программы ;
   - задаются в процессе выполнения программы.

Лабораторная работа 3.Списковые структуры данных.

1. При удалении элемента из кольцевого списка…
   - список разрывается;
   - в списке образуется дыра;
   - список становится короче на один элемент .
2. Для чего используется указатель в кольцевых списках ?
   - для ссылки на следующий элемент;
   - для запоминания номера сегмента расположения элемента;
   - для ссылки на предыдущий элемент ;
   - для расположения элемента в списке памяти.
3. Чем отличается кольцевой список от линейного ?

- в кольцевом списке последний элемент является одновременно и первым;
- в кольцевом списке указатель последнего элемента пустой;
- в кольцевых списках последнего элемента нет ;
- в кольцевом списке указатель последнего элемента не пустой.

4. Сколько указателей используется в односвязном кольцевом списке ?
   - 1;
   - 2;
   - сколько угодно.
5. В каких направлениях можно перемещаться в кольцевом двунаправленном списке ?
   - в обоих ;
   - влево;
   - вправо.

Лабораторная работа 4. Модель массового обслуживания.

1. Чем отличается заявка первого приоритета от заявки второго приоритета ?
   - тем, что заявка второго приоритета обслуживается с вероятностью P=1, а заявка первого приоритета обслуживается с вероятностью P(B);
   - тем, что заявка второго приоритета становится в начало очереди, а первого приоритета становится в конец очереди ;
   - ничем, если есть очередь.
2. Может ли заявка первого приоритета вытеснить из очереди заявку второго приоритета ?
   - да, если P(B)=1;
   - да;
   - нет .

3. Может ли на обслуживании находится заявка первого приоритета, если в очереди находится заявка второго приоритета ?
   - да, если P(B)=1;
   - да ;
   - нет.
4. С помощью какой структуры данных наиболее рационально реализовать очередь ?
   - стек;
   - список ;
   - дек.
5. Когда заявка покидает систему. Найдите ошибку.
   - если заявка обслужилась подложенное ей число тактов;
   - если заявка находится в очереди больше T тактов;
   - если заявок второго приоритета стало больше, чем заявок первого приоритета .

<u>Лабораторная работа 5.</u> Бинарные деревья (основные процедуры).

1. Для включения новой вершины в дерево нужно найти узел, к которому её можно присоединить. Узел будет найден, если очередной ссылкой, определяющей ветвь дерева, в которой надо продолжать поиск, окажется ссылка:
   - p=right(p);
   - p=nil ;
   - p=left(p).
2. Для написания процедуры над двумя деревьями необходимо описать элемент типа запись, который содержит поля:
   - Element=Запись
     Left,Right : Указатели
     Rec : Запись;

   - Element=Запись
     Left : Указатель

Key : Ключ
Rec : Запись;

- Element=Запись
  Left, Right : Указатели
  Key : Ключ
  Rec : Запись.

3. В памяти ЭВМ бинарное дерево удобно представлять в виде:
   - связанных линейных списков;
   - массивов;
   - связанных нелинейных списков .
4. Элемент t, на котрый нет ссылок:
   - корнем ;
   - промежуточным;
   - терминальным (лист).
5. Дерево называется полным бинарным, если степень исходов вершин равна:
   - 2 или 0 ;
   - 2;
   - M или 0;
   - M.

Лабораторная работа 6. Сортировка методом прямого включения.

1. Даны три условия окончания просеивания при сортировке прямым включением. Найдите среди них лишнее.
   - найден элемент a(i) с ключом, меньшим чем ключ у x;
   - найден элемент a(i) с ключом, большим чем ключ у x ;
   - достигнут левый конец готовой последовательности.
2. Какой из критериев эффективности сортировки определяется формулой M=0,01*n*n+10*n ?
   - число сравнений ;

- время, затраченное на написание программы;
- количество перемещений;
- время, затраченное на сортировку.

3. Как называется сортировка, происходящая в оперативной памяти ?
   - сортировка таблицы адресов;
   - полная сортировка;
   - сортировка прямым включением;
   - внутренняя сортировка ;
   - внешняя сортировка.
4. Как можно сократить затраты машинного времени при сортировке большого объёма данных ?
   - производить сортировку в таблице адресов ключей ;
   - производить сортировку на более мощном компьютере;
   - разбить данные на более мелкие порции и сортировать их.
5. Существуют следующие методы сортировки. Найдите ошибку.
   - строгие;
   - улудшенные;
   - динамические .

Лабораторная работа 7. Сортировка методом прямого выбора.

1. Метод сортировки называется устойчивым, если в процессе сортировки…
   - относительное расположенние элементов безразлично;
   - относительное расположение элементов с равными ключами не меняется ;
   - относительное расположение элементов с равными ключами изменяется;
   - относительное расположение элементов не определено.
2. Улучшенные методы имеют значительное преимущество:
   - при большом количестве сортируемых элементов ;

- когда массив обратно упорядочен;
- при малых количествах сортируемых элементов;
- во всех случаях.

3. Что из перечисленных ниже понятий является одним из типов сортировки ?
   - внутренняя сортировка ;
   - сортировка по убыванию;
   - сортировка данных;
   - сортировка по возрастанию.
4. Сколько сравнений требует улучшенный алгоритм сортировки ?
   - n*log(n) ;
   - $e^n$;
   - n*n/4.
5. К какому методу относится сортировка, требующая n*n сравнений ключей ?
   - прямому ;
   - бинарному;
   - простейшему;
   - обратному.

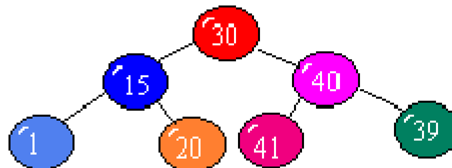
Лабораторная работа 8. Сортировка с помощью прямого обмена.

1. Сколько сравнений и перестановок элементов требуется в пузырьковой сортировке ?
   - n*lon(n);
   - (n*n)/4 ;
   - (n*n-n)/2.
2. Сколько дополнительных переменных нужно в пузырьковой сортировке помимо массива, содержащего элементы ?
   - 0 (не нужно);
   - всего 1 элемент ;
   - n переменных (ровно столько, сколько элементов в массиве).

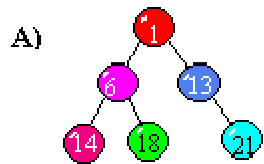
3. Как рассортировать массив быстрее, пользуясь пузырьковым методом ?
   - одинаково ;
   - по возрачстанию элементов;
   - по убыванию элементов.
4. В чём заключается идея метода QuickSort ?
   - выбор 1,2,…n – го элемента для сравнения с остальными;
   - разделение ключей по отношению к выбранному ;
   - обмен местами между соседними элементами.
5. Массив сортируется “пузырьковым” методом. За сколько проходов по массиву самый “лёгкий” элемент в массиве окажется вверху ?
   - за 1 проход ;
   - за n-1 проходов;
   - за n проходов, где n – число элементов массива.

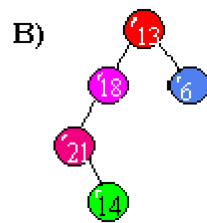
Лабораторная работа 9. Сортировка с помощью дерева.

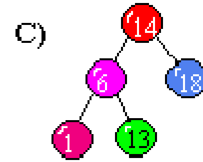
1. При обходе дерева




слева направо получаем последовательность…
   - отсортированную по убыванию;
   - неотсортированную ;
   - отсортированную по возрастанию.
2. Какое из трёх деревьев не является строго сбалансированным ?

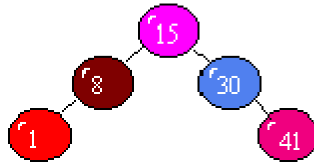




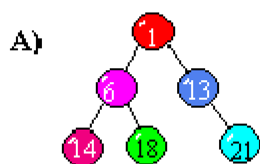


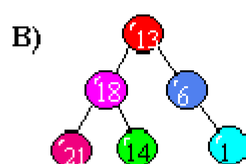

- A;
- B;
- C.

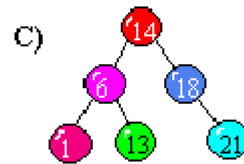
3. При обходе дерева слева направо его элемент заносится в массив…
   - при втором заходе в элемент ;
   - при первом заходе в элемент;
   - при третьем заходе в элемент.
4. Элемент массива с ключом k=20 необходимо вставить в изображённое дерево так, чтобы дерево осталось отсортированным. Куда его нужно вставить ?




   - левым сыном элемента 30 ;
   - левым сыном элемента 41;
   - левым сыном элемента 8.
   -
5. При обходе какого дерева слева направо получается отсортированный по возрастанию массив ?







- A;
- B;
- C .

Лабораторная работа 10. Исследование методов линейного и бинарного поиска.

1. Где эффективен линейный поиск ?
   - в списке;
   - в массиве;
   - в массиве и в списке .
2. Какой поиск эффективнее ?
   - линейный;
   - бинарный ;
   - без разницы.
3. В чём суть бинарного поиска ?
   - нахожденние элемента массива х путём деления массива пополам каждый раз, пока элемент не найден ;
   - нахождение элемента х путём обхода массива;
   - нахождение элемента массива х путём деления массива.
4. Как расположены элементы в массиве бинарного поиска ?
   - по возрастанию ;
   - хаотично;
   - по убыванию.
5. В чём суть линейного поиска ?
   - производится последовательный просмотр от начала до конца и обратно через 2 элемента;
   - производится последовательный просмотр элементов от середины таблицы;
   - производится последовательный просмотр каждого элемента .

Лабораторная работа 11. Исследование методов поиска с перемещением в начало и транспозицией.

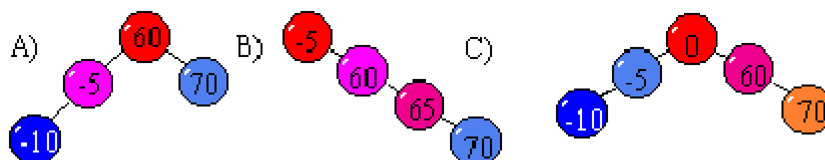
1. Где наиболее эффективен метод транспозиций ?
   - в массивах и в списках ;

- только в массивах;
- только в списках.

2. В чём суть метода перестановки ?
   - найденный элемент помещается в голову списка ;
   - найденный элемент помещается в конец списка;
   - найденный элемент меняется местами с последующим.
3. В чём суть метода транспозиции ?
   - перестановка местами соседних элементов;
   - нахождение одинаковых элементов;
   - перестановка найденного элемента на одну позицию в сторону начала списка .
4. Что такое уникальный ключ ?
   - если разность значений двух данных равна ключу;
   - если сумма значений двух данных равна ключу;
   - если в таблице есть только одно данное с таким ключом .
5. В чём состоит назначение поиска ?
   - среди массива данных найти те данные, которые соответствуют заданному аргументу ;
   - определить, что данных в массиве нет;
   - с помощью данных найти аргумент.

<u>Лабораторная работа 12.</u> Поиск по дереву с включением.

1. В каком дереве при бинарном поике нужно перебрать в среднем N/2 элементов ?




- A;
- B;
- C.

2. Сколько нужно перебрать элементов в сбалансированном дереве ?

A)N/2;

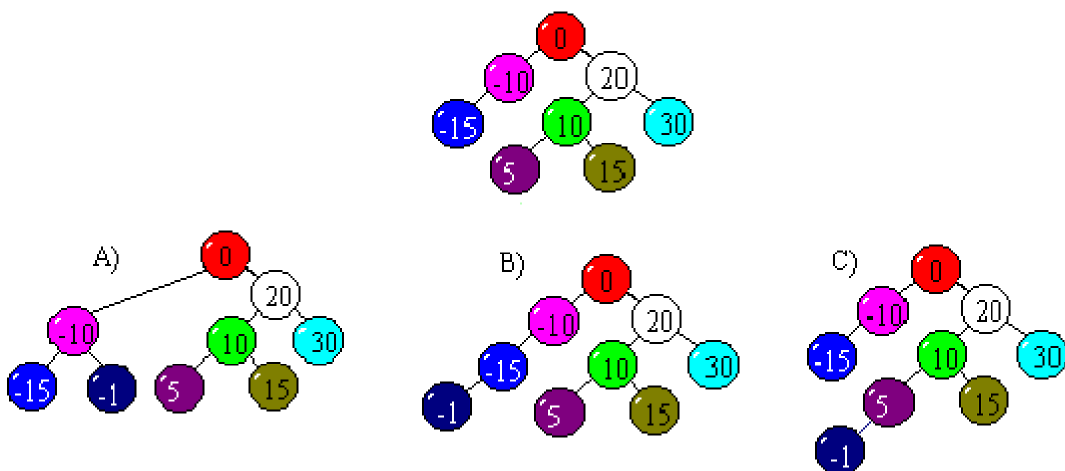
B)Ln(N);

C)$Log_2(N)$;

D)$e^N$.

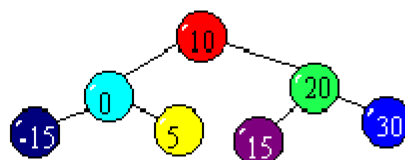
- A;
- B;
- C ;
- D.

3. Выберете вариант дерева, полученного после вставки узла -1.



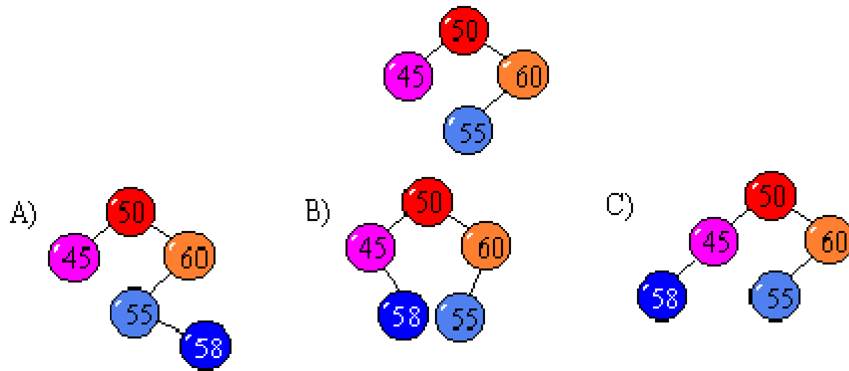

- A ;
- B;
- C.
-

4. К какому элементу присоединить элемент 40 для вставки его в данное дерево ?



- к 30-му ;
- к 15-му;
- к –15-му;
- к 5-му.

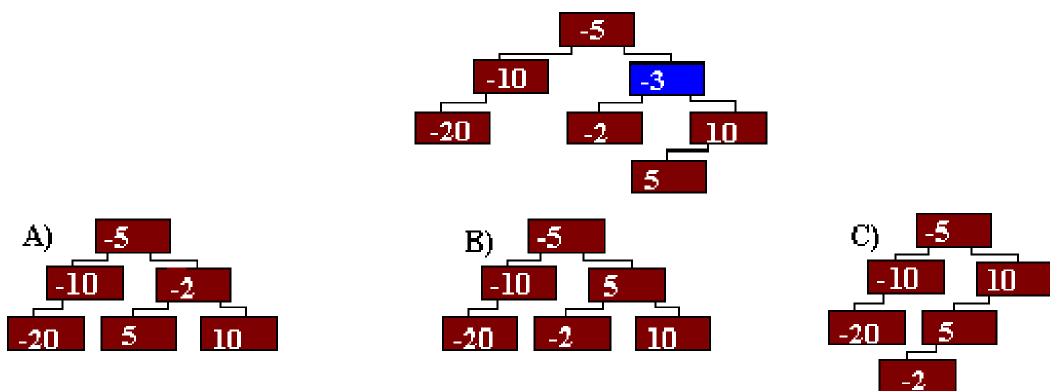
5. Какой вид примет дерево после встаки элемента с ключом 58 ?




- A ;
- B;
- C.

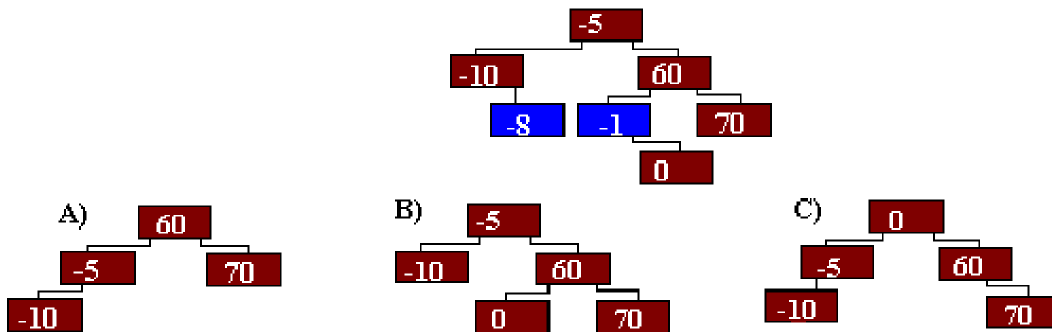
Лабораторная работа 13. Поиск по дереву с исключением.

1. Выберете вариант дерева, полученного после удаления узла –3.



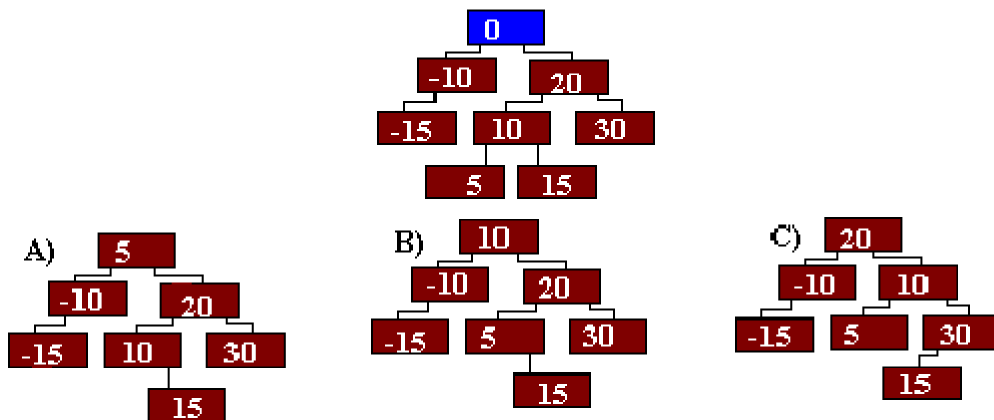
- A;
- B ;
- C.

2. Какой вариант дерева получится после удаления элемента –1, а затем –8?



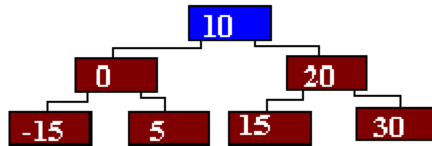

- A;
- B ;
- C.

3. Выберете вариант дерева, полученного после удаления узла с индексом 0.



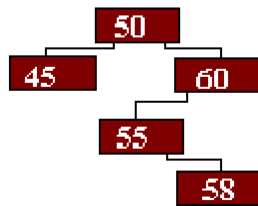

- A ;
- B;
- C.

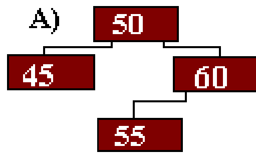
4. Какие из следующих пар чисел могут стать корнями дерева после удаления элемента 10 в соответсвии с двумя способами удаления узла, имеющего двух сыновей ?

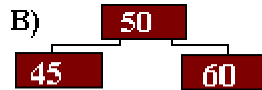


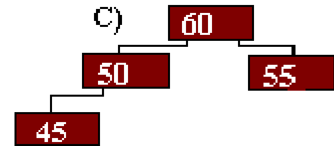

- 0 или 15;
- 0 или 20;
- 5 или 30;
- 5 или 15 .

5. Какой вид примет дерево после удаления элемента с ключом 58 ?




- A ;
- B;
- C.

# МЕТОДИЧЕСКОЕ РУКОВОДСТВО К КУРСОВОЙ РАБОТЕ

## Введение

Курсовая работа выполняется студентами специальности 22.04 в четвертом семестре.

Целью курсовой работы является закрепление основ и углубление знаний в области структур и алгоритмов обработки данных в ЭВМ.

Тематика заданий на курсовую работу, приведенных в данных методических указаниях, может быть дополнена, расширена, увязана с решением актуальных научно-исследовательских задач, выполняемых на кафедре.

## 1 Требования к курсовой работе

**1.1** Тема курсовой работы выдается каждому студенту индивидуально. В коллективных работах, в которых принимают участие два и более студентов, четко определяются объем и характер работы каждого студента. В задании формулируется задача, метод её решения.

**1.2** Курсовая работа состоит из пояснительной записки, к которой прилагается дискета с отлаженными программами (пояснительная записка может быть выполнена в виде текстового файла в формате Microsoft Word).

**1.3** В пояснительную записку должны входить:

- титульный лист (приложение Б);

- задание на курсовое проектирование (приложение А);

- реферат (ПЗ, количество таблиц, рисунков, схем, программ приложений, краткая характеристика и результаты работы);

- содержание:

а) постановка задачи исследования;

б) краткая теория по теме курсовой работы;

в) программная реализация исследуемых алгоритмов;

г) программа, с помощью которой проводилось исследование;

д) результаты проведенного исследования:

е) выводы;

- список использованной литературы;

- подпись, дата.

**1.4** Пояснительная записка должна быть оформлена на листах формата А4, имеющих поля. Все листы следует сброшюровать и пронумеровать.

**1.5** Исследование алгоритмов операций над структурами данных и методов сортировок и поиска проводить при следующих фиксированных количествах элементов в структурах: 10, 100, 1000, 10000.

**1.6** Дополнительные условия выполнения курсовой работы выдаются руководителем работы.

## 2. Примерный перечень курсовых работ

1) Исследование стеков.
2) Исследование очередей.
3) Исследование кольцевых структур.

4) Исследование полустатических структур.
5) Исследование линейных одно- и двусвязных списков.
6) Исследование деревьев бинарного поиска.
7) Исследование методов сортировки включением.
8) Исследование методов сортировки выбором.
9) Исследование методов сортировки обменом.
10) Исследование методов сортировки с помощью деревьев.
11) Исследование улучшенных методов сортировки.
12) Исследование линейного, индексного и бинарного поисков.
13) Исследование методов оптимизации поиска.
14) Исследование задач поиска по дереву.

## 3. Пример выполнения курсовой работы

### 3.1 Постановка задачи

Осуществить исследование прямых методов сортировки:
- метод прямого выбора;
- метод прямой вставки;
- метод прямого обмена.

Исследование осуществить, используя массивы упорядоченных и неупорядоченных чисел по 10,100,1000 и 10000 элементов.

### 3.2 Краткая теория

При обработке данных важно знать и информационное поле данных, и размещение их в машине.

Различают внутреннюю и внешнюю сортировки:

– внутренняя сортировка - сортировка в оперативной памяти;

– внешняя сортировка - сортировка во внешней памяти.

Сортировка - это расположение данных в памяти в регулярном виде по их ключам. Регулярность рассматривают как возрастание (убывание) значения ключа от начала к концу в массиве.

Если сортируемые записи занимают большой объем памяти, то их перемещение требует больших затрат. Для того, чтобы их уменьшить, сортировку производят в *таблице адресов ключей*, делают перестановку указателей, т.е. сам массив не перемещается. Это *метод сортировки таблицы адресов.*

При сортировке могут встретиться одинаковые ключи. В этом случае при сортировке желательно расположить после сортировки одинаковые ключи в том же порядке, что и в исходном файле. Это *устойчивая сортировка*.

Эффективность сортировки можно рассматривать с нескольких критериев:

– время, затрачиваемое на сортировку;

– объем оперативной памяти, требуемой для сортировки;

– время, затраченное программистом на написание программы.

Выделяем первый критерий. Можно подсчитать количество сравнений при выполнении сортировки или количество перемещений.

Пусть $H = 0{,}01n^2 + 10n$ - число сравнений. Если $n < 1000$, то второе слагаемое больше, если $n > 1000$, то больше первое слагаемое.

Т. е. при малых n порядок сравнения будет равен $n^2$, при больших n порядок сравнения - n.

Порядок сравнения при сортировке лежит в пределах
от 0 (n log n) до 0 ($n^2$); 0 (n) - идеальный случай.

Различают следующие методы сортировки:

– строгие (прямые) методы;

– улучшенные методы.

Строгие методы:
1) метод прямого включения;
2) метод прямого выбора;
3) метод прямого обмена.

Количество перемещений в этих трех методах примерно одинаково.

### 3.2.1 Сортировка методом прямого включения

Неформальный алгоритм

```
for i = 2 to n
  x = a(i)
  находим место среди a(1)…a(i) для включения x
next i
```

Программа на Паскале:

```
for i := 2 to n do
  begin
    x := a[i];
    a[0] = x;
    for j := i - 1 downto 1 do
      if x < a[j]
          then a[j + 1] := a[j]
          else a[j + 1] := x;
  end;
```

***Эффективность алгоритма:***

Количество сравнений в худшем случае будет равно (n - 1)(n - 1).

### 3.2.2 Сортировка методом прямого выбора

Рассматриваем весь ряд массива и выбираем элемент, меньший или больший элемента a(i), определяем его место в массиве - k, и затем меняем местами элемент a(i) и элемент a(k).

Программа на Паскале:

```
for i := 1 to n - 1 do
  begin
    x := a[i];
    k := i;
    for j := i + 1 to n do
      if x > a[j] then
        begin
          k := j;
          x := a[k];
        end;
    a[k] := a[i];
    a[i] := x;
  end;
```

***Эффективность алгоритма:***

Число сравнений M = $\frac{n}{2}(n-1) = \frac{n^2 - n}{2}$.

Число перемещений $C_{min}$ = 3(n - 1), $C_{max}$ = 3(n - 1)$\frac{n}{2}$ (порядок $n^2$).

В худшем случае сортировка прямым выбором дает порядок $n^2$, как и для числа сравнений, так и для числа перемещений.

### 3.2.3 Сортировка с помощью прямого обмена (пузырьковая)

Идея: n - 1 раз проходят массив снизу вверх. При этом ключи попарно сравниваются. Если нижний ключ в паре меньше верхнего, то их меняют местами.

Программа на Паскале:

```
for i := 2 to n do
```

```
for j := n downto i do
  if a[j - 1] > a[j] then
    begin
      x := a[j - 1];
      a[j - 1] := a[j];
      a[j] := x;
    end;
```

В нашем случае получился один проход “вхолостую”. Чтобы лишний раз не переставлять элементы, можно ввести флажок.

Улучшением пузырькового метода является шейкерная сортировка, где после каждого прохода меняют направление внутри цикла.

***Эффективность алгоритма:***

число сравнений $M = \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$,

число перемещений $C_{max} = 3\frac{n^2}{4}$.

**3.3 Метод исследования**

Данная курсовая работа заключается в исследовании прямых методов сортировки:

- метода прямого выбора;
- метода прямого включения;
- метода прямого обмена.

Исследование заключается в следующем:

берут три массива с одинаковым количеством элементов, но один из них упорядоченный по возрастанию, второй - по убыванию, а третий - случайный. Осуществляется сортировка данных массивов и сравнивается количество перемещений элементов при сортировке первого, второго и третьего

массивов, а также сравнивается количество сравнений при сортировке.

Вышеописанный способ применяется для массивов, состоящих из 10, 100, 1000 и 10000 упорядоченных и неупорядоченных элементов для всех методов сортировки.

### 3.4 Результаты исследования

***Сортировка 10 элементов:***

- упорядоченных по возрастанию

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 45 | 45 | 45 |
| перемещений | 11 | 33 | 33 |

- неупорядоченных (случайных)

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 45 | 45 | 45 |
| перемещений | 27 | 27 | 27 |

- упорядоченных по убыванию

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 45 | 45 | 45 |
| перемещений | 0 | 0 | 0 |

***Сортировка 100 элементов:***

- упорядоченных по возрастанию

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 4950 | 4950 | 4950 |
| перемещений | 2643 | 4862 | 4862 |

- неупорядоченных (случайных)

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 4950 | 4950 | 4950 |
| перемещений | 2558 | 2558 | 2558 |

- упорядоченных по убыванию

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|

| | | | |
|---|---|---|---|
| сравнений | 4950 | 4950 | 4950 |
| перемещений | 0 | 0 | 0 |

***Сортировка 1000 элементов:***

- упорядоченных по возрастанию

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 499500 | 499500 | 499500 |
| перемещений | 241901 | 498442 | 498442 |

- неупорядоченных (случайных)

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 499500 | 499500 | 499500 |
| перемещений | 244009 | 250366 | 250366 |

- упорядоченных по убыванию

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 499500 | 499500 | 499500 |
| перемещений | 0 | 0 | 0 |

***Сортировка 10000 элементов:***

- упорядоченных по возрастанию

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 49995000 | 49995000 | 49995000 |
| перемещений | 25003189 | 49984768 | 49984768 |

- неупорядоченных (случайных)

| метод | прямого выбора | прямой вставки | прямого обмена |
|---|---|---|---|
| сравнений | 49995000 | 49995000 | 49995000 |
| перемещений | 18392665 | 24986578 | 24986578 |

- упорядоченных по убыванию

| метод | прямого выбора | прямой ставки | прямого обмена |
|---|---|---|---|
| сравнений | 49995000 | 49995000 | 49995000 |
| перемещений | 0 | 0 | 0 |

## 3.5 Контрольный пример

***Задание:***

Дан список, содержащий имена студентов и соответствующие им баллы рейтинга. Необходимо отсортировать данный список по убыванию баллов рейтинга.

Сортировка методом прямого включения:

| ***До сортировки*** | | ***После сортировки*** | |
|---|---|---|---|
| Аркадий | 19 | Артур | 20 |
| *До сортировки* | | *После сортировки* | |
| Мурат | 17 | Аркадий | 19 |
| Руслан | 9 | Александр | 18 |
| Артур | 20 | Владимир | 18 |
| Евгений | 7 | Мурат | 17 |
| Михаил | 15 | Казбек | 16 |
| Александр | 18 | Михаил | 15 |
| Виталий | 14 | Борис | 15 |
| Сидор | 8 | Денис | 14 |
| Владимир | 18 | Виталий | 14 |
| Алексей | 6 | Василий | 13 |
| Казбек | 16 | Петр | 10 |
| Марат | 5 | Руслан | 9 |
| Борис | 15 | Иван | 8 |
| Геннадий | 2 | Сидор | 8 |
| Денис | 14 | Евгений | 7 |
| Василий | 13 | Алексей | 6 |
| Сидор | 3 | Марат | 5 |
| Петр | 10 | Сидор | 3 |
| Иван | 8 | Геннадий | 2 |

## 3.6 Выводы

По результатам исследования можно утверждать, что лучшим из прямых методов сортировки является метод прямого выбора, так как он дает наименьшее количество сравнений и перемещений независимо от количества сортируемых элементов и их взаимного расположения в массиве.

## 3.7 Описание процедур, используемых в программе

| | |
|---|---|
| UPOR | процедура генерирует упорядоченный по возрастанию массив чисел |
| NOTUPOR | процедура генерирует неупорядоченный (случайный) массив чисел |
| PR_CHOOSE | процедура осуществляет сортировку методом прямого выбора |
| PR_INS | процедура осуществляет сортировку методом прямой вставки |
| PR_OBM | процедура осуществляет сортировку методом прямого обмена |
| MAKE | процедура осуществляет исследование прямых методов сортировки |
| EXAMPLE | процедура выполняет контрольный пример (сортировку методом прямого включения) |

*Текст программы*

```
{$M 65000,65000,65000}
{Выделение памяти осуществляется для того, чтобы было возможно осу-
ществлять исследование  массива, содержащего 10000 элементов
***********************************************************
       Данная программа является курсовой работой по дисциплине
            'Структуры и алгоритмы обработки данных'
               на тему 'Прямые методы сортировки'
                В работе исследуются методы:
                      - прямого выбора;
                      - прямого обмена;
                      - прямой вставки.
   Для исследования используются массивы из 10,100,100,10000 элементов.
```

```
********************************************************}
{ использование модулей для осуществления вывода на экран }
uses
crt,crtext,dcrt;*************************************************}
{** процедура, генерирующая упорядоченный по возрастанию массив чи-
сел**}
{*******************************************************}
procedure upor(a:array of integer;var a1:array of integer);
var
{i - счетчик в циклах}
 i:integer;
begin
{первый элемент принимает значение 1}
a[0]:=1;
 for i:=1 to high(a) do
  begin
{каждый последующий элемент принимает значение,
равное значению предыдущего элемента + случайное число}
   a[i]:=a[i-1]+random(2);
  end;
 for i:=0 to high(a) do
   a1[i]:=a[i];
end;
{*******************************************************}
{** процедура, генерирующая  не упорядоченный (случайный) массив чи-
сел**}
{*******************************************************}
procedure notupor(a:array of integer;var a1:array of integer);
var
{i - счетчик в циклах}
 i:integer;
begin
 for i:=0 to high(a) do
  begin {элемент массива принимает случайное значение из диапазона 0 <=
a[i] < 9999}
   a[i]:=random(9999);
  end;
 for i:=0 to high(a) do
   a1[i]:=a[i];
end;
{*******************************************************}
```

```
{***** процедура, осуществляющая сортировку методом прямого выбо-
ра***}
{********************************************************}
procedure pr_choose(a:array of integer;var a1:array of integer;var k,k1:longint);
var
{i,j - счетчики в циклах}
 i,j:integer;
{x - переменная для осуществления обмена между a[i] и a[j]}
 x:integer;
begin
{k1  -  переменная для подсчета количества сравнений
 k   -  переменная для подсчета количества перемещений}
          k:=0;k1:=0;
{high(a) - функция, возвращающая номер последнего элемента массива}
for i := 0 to high(a) - 1 do
  begin
    for j := i + 1 to high(a) do
     begin
      k1:=k1+1;
      if a[i] < a[j] then
        begin
         k:=k+1;
  {перестановка i-го с j-м элементом с помощью переменной x}
         x:=a[i];
         a[i]:=a[j];
         a[j]:=x;
        end;
     end;
  end;
for i:=0 to high(a) do
 a1[i]:=a[i];
end;
{********************************************************
 ***** процедура, осуществляющая сортировку методом прямого *
 *************** обмена(метод пузырька) ********************
*********************************************************}
procedure pr_obm(a:array of integer;var a1:array of integer;var k,k1:longint);
var
{i,j - счетчики в циклах}
 i,j:integer;
{x - переменная для осуществления обмена между a[j-1] и a[j]}
 x:integer;
```

```
begin
{k1 -  переменная для подсчета количества сравнений
 k  -  переменная для подсчета количества перемещений}
k:=0;k1:=0;
for i := 1 to high(a) do
 begin
  for j := high(a) downto i do
   begin
    k1:=k1+1;
    if a[j - 1] < a[j] then
      begin
        k:=k+1;
  {обмена содержимым элементов массива a[j-1] и a[j]
   с помощью переменной x}
        x := a[j - 1];
        a[j - 1] := a[j];
        a[j] := x;
      end;
    end;
 end;
 for i:=1 to high(a) do
  a1[i]:=a[i];
end;
{*******************************************************}
{*** процедура, осуществляющая сортировку методом прямого включе-
ния **}
{*******************************************************}
procedure pr_ins(a:array of integer;var a1:array of integer;var k,k1:longint);
var
{i,j - счетчики в циклах}
 i,j:integer;
{x - переменная для осуществления обмена между a[i] и a[j]}
 x:integer;
begin
{k1 -  переменная для подсчета количества сравнений
 k  -  переменная для подсчета количества перемещений}
k:=0;k1:=0;
for i := 1 to high(a) do
  begin
    x := a[i];
    for j := i - 1 downto 0 do
     begin
```

```
     k1:=k1+1;
     if x > a[j] then
                begin
                 k:=k+1;
  {обмена содержимым элементов массива a[j+1] и a[j]
   с помощью переменной x}
                 a[j + 1] := a[j];
                 a[j]:=x;
                end;
    end;
  end;
  for i:=0 to high(a) do
   a1[i]:=a[i];
end;
{************************************************************
      процедура, осуществляющая исследование методов сортировок  для
 ***********  заданного массива чисел  **********************
************************************************************}
procedure make(x1,n:integer;a,a1:array of integer;k:byte);
var
{количество перестановок}
 kol_pr_ins, kol_pr_obm,kol_pr_choose:longint;
{количество сравнений}
 kol_sr_ins, kol_sr_obm,kol_sr_choose:longint;
  s:string;
begin
case k of
1:s:='упорядоченных по возрастанию';
2:s:='неупорядоченных (случайных)';
3:s:='упорядоченных по убыванию';
end;
{--------метод прямого включения---------}
pr_ins(a,a1,kol_pr_ins,kol_sr_ins);
{--------метод прямого обмена (метод пузырька)--------}
pr_obm(a,a1,kol_pr_obm,kol_sr_obm);
{---------метод прямого выбора----------}
pr_choose(a,a1,kol_pr_choose,kol_sr_choose);
{** вывод результата исследования **}
{вывод шапки таблицы}
gotoxy(3,x1);textcolor(cyan);textbackground(1);
writeln('Для ',high(a)+1,' ',s,' элементов:');
gotoxy(3,x1+1);textcolor(lightgreen);textbackground(1);
```

```
writeln('Методы:    прямого включения    прямого обмена     прямого вы-
бора');
{вывод полученных при исследовании данных}
gotoxy(3,x1+2);textcolor(white);write('перест.');
gotoxy(17,wherey);write(kol_pr_ins);
gotoxy(37,wherey);write(kol_pr_obm);
gotoxy(58,wherey);writeln(kol_pr_choose);
gotoxy(3,x1+3);write('сравн.');
gotoxy(17,wherey);write(kol_sr_ins);
gotoxy(37,wherey);write(kol_sr_obm);
gotoxy(58,wherey);writeln(kol_sr_choose);
str(high(a)+1,s);box(1,19,80,24,1,15,double,s+' элементов');
 gotoxy(4,20);write('Сортировка ',s,' элементов по убыванию');
 gotoxy(4,21);write('Сортируются ',s,' упорядоченных(по возрастанию) эле-
ментов');
 gotoxy(4,22);write('Сортируются ',s,' неупорядоченных(случайных) эле-
ментов');
 textbackground(lightgray);
 textcolor(red);gotoxy(3,25);write('Esc - главное меню');
end;
{********************************************
 Пример сортировки методом прямого включения
 Дан массив записей, содержащий:
         -имя студента;
         -кол-во баллов (рейтинг).
   Необходимо отсортировать данный массив по
   убыванию  количества  баллов  у  студента.
********************************************}
procedure example;
type
{rec - запись, содержащая:
              name - имя студента;
              num  - кол-во баллов (рейтинг).}
 rec=record
     name:string;
     num:byte;
    end;
var
{mas - массив записей rec}
 mas:array[1..20] of rec;
{счетчики в циклах}
 i,j:integer;
```

```
 x:rec;
{переменные для подсчета количества сравнений и перемещений
 во время сортировки}
 k_sr,k_p:integer;
 key:char;
begin
{переменные для подсчета количества сравнений и перемещений
 во время сортировки}
 k_sr:=0;k_p:=0;
randomize;
{Данный массив, содержащий имена студентов}
mas[1].name:='Иван';mas[2].name:='Петр';mas[3].name:='Сидор';
mas[4].name:='Василий';mas[5].name:='Денис';mas[6].name:='Геннадий';
mas[7].name:='Борис';mas[8].name:='Марат';mas[9].name:='Казбек';
mas[10].name:='Алексей';mas[11].name:='Владимир';mas[12].name:='Сидор';
mas[13].name:='Виталий';mas[14].name:='Александр';mas[15].name:='Михаи
л';
mas[16].name:='Евгений';mas[17].name:='Артур';mas[18].name:='Руслан';
mas[19].name:='Мурат';mas[20].name:='Аркадий';
{задание количества баллов у студента случайным образом}
for i:=1 to 20 do
 mas[i].num:=random(19)+1;
{вывод пояснений}
getshadow;
textbackground(lightgray);
textcolor(red);gotoxy(15,1);write('Пример сортировки по убыванию методом
прямого включения');
textbackground(lightgray);
textcolor(red); gotoxy(3,25);  write('Esc - главное меню');
textcolor(red); gotoxy(65,25); write('F1 - задание');
box(58,0,80,25,1,15,double,'Статистика');
textcolor(lightmagenta);
gotoxy(59,3); write(' Сортировка  методом ');
gotoxy(61,4); write('прямого включения.');
textcolor(white); gotoxy(59,5); write('---------------------');
box(31,0,57,25,1,15,double,'После сортировки');
textcolor(lightmagenta); gotoxy(32,3); write(' N   Имя          балл');
box(1,0,30,25,1,15,double,'До сортировки');
textcolor(lightmagenta); gotoxy(3,3);  write('N   Имя            балл');
{вывод на экран исходного массива}
for i:=1 to 20 do
 begin
```

```
textcolor(lightcyan); gotoxy(3,i+3);  write(i);
textcolor(yellow);    gotoxy(7,i+3);  write(mas[i].name);
textcolor(lightred);  gotoxy(25,i+3); writeln(mas[i].num);
 end;
{сортировка по убыванию количества баллов}
for i := 2 to 20 do
  begin
    x := mas[i];
    for j := i - 1 downto 1 do
     begin
  {k_sr - счетчик количества сравнений}
   k_sr:= k_sr+1;
        if x.num > mas[j].num then
                  begin
  {k_p - счетчик количества перемещений}
                  k_p:=k_p+1;
  {обмена содержимым элементов массива mas[j+1] и mas[j]
   с помощью переменной x}
                   mas[j + 1] := mas[j];
                  mas[j]:=x;
                  end;
     end;
  end;

{вывод на экран отсортированного массива}
for i:=1 to 20 do
 begin
textcolor(lightcyan); gotoxy(33,i+3);  write(i);
textcolor(yellow);    gotoxy(37,i+3);  write(mas[i].name);
textcolor(lightred);  gotoxy(52,i+3);  writeln(mas[i].num);
 end;
{вывод на экран количества сравнений и перестановок
 в массиве, осуществленных во время сортировки}
 textcolor(lightgreen); gotoxy(61,6); write('Сравнений:');
 textcolor(lightgray);  gotoxy(75,6); write(k_sr);
 textcolor(lightgreen); gotoxy(61,8); write('Перестановок:');
 textcolor(lightgray);  gotoxy(75,8); write(k_p);
   repeat
   key:=' ';   if keypressed then  key:=readkey;
    case key of
    {#59 - код клавиши F1}
    #59:begin
```

```
   {вывод окна с заданием для контрольного примера}
   {putshadow+box - вывод окна с тенью}
    putshadow(15,7,65,15);box(15,7,65,15,lightgray,0,double,'Задание');
       textcolor(red);
       gotoxy(21,9);  write('Дан список, содержащий фамилии студентов');
       gotoxy(21,10); write('и их рейтинговые  баллы. Необходимо от-');
       gotoxy(21,11); write('сортировать данный список  по убыванию ');
       gotoxy(21,12); write('рейтинговых баллов.');
       textcolor(yellow);gotoxy(50,15);write('Enter - отмена');
      end;
   {#13 - код клавиши Enter}
   #13:getshadow;
   end;
   {#27 - код клавиши Esc}
  until key=#27;
end;
{*********************************************************
*********************** Основная программа **************
*********************************************************}
const
{массив строк - пунктов главного меню}
 menu:array[1..7] of string=(' Пример сортировки  ',' Исследование (10 эл-
тов)',
                    ' Исследование (100 эл-тов) ',
                    ' Исследование (1000 эл-тов) ',
                    ' Исследование (10000 эл-тов)  ',
                    ' О программе '
                    ,' Выход ');
var
{массивы чисел, используемые для исследования}
 a,a1:array[0..9] of integer;    {10    - чисел}
 b,b1:array[0..99] of integer;   {100   - чисел}
 c,c1:array[0..999] of integer;  {1000  - чисел}
 d,d1:array[0..9999] of integer; {10000 - чисел}
 f:byte;                    {показатель того , какой массив
                           поступает в процедуру для упо-
                           рядочивания по убыванию:
                            1    - неупорядоченный (слу-
                                  чайный);
                            2    - упорядоченный по воз-
                                  растанию;
                            3    - упорядоченный по убы-
```

```
                                        ванию}.
 k:char;
 item:byte;
begin
clrscr;cursoroff; {гашение курсора}
repeat
textbackground(0);clrscr;
 {fill - процедура, заполняющая заданную область  экрана заданными  сим-
                                 волами заданного  цвета}
fill(lightgray,1,1,2,80,25,' ');
{menuv - процедура, выводящая на экран вертикальное меню}
menuv(25,10,menu,lightgray,black,red,lightgreen,yellow,'Сортировка',item,do
uble);
{item - переменная, содержащая номер выбранного пункта меню}
case item of
1:example;
2:begin
      {getshadow - процедура, убирающая тень от меню}
        getshadow;
      {** исследуются 10 упорядоченных по возрастанию элементов **}
      {box - процедура, выводящая на экран окно}
         box(1,0,80,18,1,15,double,'10 элементов');
      {вызов процедуры upor, генерирующей упорядоченный по возраста-
нию
       массив чисел}
         upor(a,a);
       {вызов процедуры make, осуществляющей исследование методов сор-
                                                тировки}
         make(3,10,a,a1,1);
      {** исследуются 10 неупорядоченных (случайных) элементов **}
      {вызов процедуры notupor, генерирующей неупорядочен-
ный(случайный)  массив чисел}
        notupor(a,a);
       {вызов процедуры make, осуществляющей исследование методов сор-
                                                тировки}
        make(8,10,a,a1,2);
      {** исследуются 10 упорядоченных по убыванию элементов **}
       {вызов процедуры make, осуществляющей исследование методов сор-
                                                тировки}
        make(13,10,a1,a1,3);
     {ожидание нажатия клавиши Esc}
    repeat
```

```
    k:=readkey;
   until k=#27;
 end;
3:begin
     {getshadow - процедура, убирающая тень от меню}
     getshadow;
     {box - процедура, выводящая на экран окно}
     box(1,0,80,18,1,15,double,'100 элементов');
    {исследуются 100 упорядоченных по возрастанию элементов}
      upor(b,b);
      make(3,100,b,b1,1);
    {исследуются 100 неупорядоченных (случайных) элементов}
       notupor(b,b);
       make(8,100,b,b1,2);
     {исследуются 100 упорядоченных по убыванию элементов}
       make(13,100,b1,b1,3);
   repeat   k:=readkey;    until k=#27;
 end;
4:begin
     {getshadow - процедура, убирающая тень от меню}
     getshadow;
     box(1,0,80,18,1,15,double,'1000 элементов');
 {исследуется 1000 упорядоченных по возрастанию элементов}
  upor(c,c);
  make(3,1000,c,c1,1);
 {исследуется 1000 неупорядоченных (случайных) элементов}
  notupor(c,c);
  make(8,1000,c,c1,2);
 {исследуется 1000 упорядоченных по убыванию элементов}
  make(13,1000,c1,c,3);
   repeat
    k:=readkey;
   until k=#27;
 end;
5:begin
     getshadow;
     box(1,0,80,18,1,15,double,'10000 элементов');
 {исследуются 10000 упорядоченных по возрастанию элементов}
  upor(d,d);
  make(3,10000,d,d1,1);
 {исследуются 10000 неупорядоченных (случайных) элементов}
  notupor(d,d);
```

```
  make(8,10000,d,d1,2);
 {исследуются 10000 упорядоченных по убыванию элементов}
  make(13,10000,d1,d,3);
    repeat
     k:=readkey;
    until k=#27;
  end;
6:begin
      {getshadow - процедура, убирающая тень от меню}
      getshadow;
      {ввод окна с темой курсовой работы}
      box(10,5,70,15,lightgray,0,double,'О программе');
      putshadow(10,5,70,15);
      textcolor(brown);
        gotoxy(12,7);write('Данная программа является курсовой работой по
                                              дисциплине');
      gotoxy(21,8);write('"Алгоритмы и структуры обработки данных"');
      gotoxy(30,9);write('Тема курсовой работы: ');
      gotoxy(18,10);write('    "Исследование прямых методов сортировки"');
       gotoxy(17,11);write('Курсовую работу выполнили студенты группы 95-
                                              ОА-21');
        textcolor(red);gotoxy(3,25);write('Esc - главное меню');
    repeat
     k:=readkey;
    until k=#27;
  end;
end;
until item=7;
end.
{********************конец программы*******************}
```

# ЗАКЛЮЧЕНИЕ

В учебном пособии были рассмотрены наиболее распространенные оперативные структуры данных и алгоритмы их обработки, которые традиционно применяются при создании программных систем и комплексов. В силу ограниченности объемом курса не было уделено внимания таким структурам, как B - деревья и графы, в разделе поиска опущен раздел хеширования. Однако, на базе уже рассмотренного материала эти разделы могут быть легко изучены самостоятельно.

Современное состояние и тенденции развития вычислительной техники как основного инструмента информатики таковы, что наряду с увеличением функциональности вычислительная техника приобретает свойства, позволяющие работать на ней пользователю, не разбирающемуся в программировании. Бурно развиваются в последнее время локальные, корпоративные и глобальные вычислительные сети. Создаются мощные накопители данных. Другими словами, основные процессы информационных технологий (обработка, обмен и накопление данных) поднялись на следующую ступень, что, естественно, требует новых подходов к организации данных в ЭВМ и созданию соответствующих систем программирования. Определяющими факторами к этому являются современные требования к пользовательскому интерфейсу и мультимедийные системы. Появились структкры графических данных и более крупные, интегральные информационные единицы - объекты. Следствием явилось бурное развитие объектно-ориентированных систем программирования: Visual BASIC, Visual PASCAL, Visual C $^{++}$ и т.д., используемых для создания программ, в основе которых лежит обработка объектных структур данных. Обмен объектными структурами в сетях вызван развитием сетевых операционных систем: Intranetware, Solaris, Windows NT и т.д. Обработка данных на

многопроцессорных вычислительных системах потребовала создания новых структур данных, основанных на абстрактных представлениях и новых языков программирования: Modula 2, ADA, OCCAM.

Таким образом, развитие информационных технологий, их проникновение во все области жизнедеятельности человека требуют компьютерного отображения информации в виде соответствующих структкр данных и, естественно, каждый новый поступаельный шаг информатики будет сопровождаться соответсвующим шагом в области структур данных.

# ЛИТЕРАТУРА


1. **Бертисс А.Т.** Структуры данных./Пер.с англ.- М.:Статистика,1974.
2. **Вирт Н.** Алгоритмы и структуры данных.- М.: Мир,1989.
3. **Д. Райли.** Абстракция и структуры данных. Вводный курс. М.: Мир, 1993.
4. **Костин А.Е.,Шаньгин В.Ф.** Организация и обработка структур данных в вычислительных системах.- М.: Высшая школа,1987.
5. **Ленгсам и др.** Структуры данных для персональных ЭВМ. - М.: Мир, 1989.
6. **Трамбле Ж., Соренсон П.** Введение в структуры данных. - М.: Машиностроение, 1982.

# ПРИЛОЖЕНИЕ.
# ТЕСТЫ С ОТВЕТАМИ

Лабораторная работа 1. Полустатические структуры данных (стеки).

6. В чём особенности очереди ?
   - открыта с обеих сторон (верный);
   - открыта с одной стороны на вставку и удаление;
   - доступен любой элемент.
7. В чём сосбенности стека ?
   - открыт с обеих сторон на вставку и удаление;
   - доступен любой элемент;
   - открыт с одной стороны на вставку и удаление (верный).
8. Какую дисциплину обслуживания принято называть FIFO ?
   - стек;
   - очередь (верный);
   - дек.
9. Какая операция читает верхний элемент стека без удаления ?
   - pop;
   - push;
   - stackpop (верный).
10. Каково правило выборки элемента из стека ?
   - первый элемент;
   - последний элемент (верный);
   - любой элемент.

Лабораторная работа 2.Списковые структуры данных (односвязные очереди).

6. Как освободить память от удаленного из списка элемента ?

- p=getnode;
- ptr(p)=nil;
- freenode(p) (верный);
- p=lst.

7. Как создать новый элемент списка с информационным полем D ?
   - p=getnode;
   - p=getnode; info(p)=D (верный);
   - p=getnode; ptr(D)=lst.
8. Как создать пустой элемент с указателем p ?
   - p=getnode (верный);
   - info(p);
   - freenode(p);
   - ptr(p)=lst.
9. Сколько указателей используется в односвязных списках ?
   - 1 (верный);
   - 2;
   - сколько угодно.
10. В чём отличительная особенность динамических объектов ?
    - порождаются непосредственно перед выполнением программы;
    - возникают уже в процессе выполнения программы (верный);
    - задаются в процессе выполнения программы.

<u>Лабораторная работа 3.</u>Списковые структуры данных.

6. При удалении элемента из кольцевого списка…
   - список разрывается;
   - в списке образуется дыра;
   - список становится короче на один элемент (верный).
7. Для чего используется указатель в кольцевых списках ?
   - для ссылки на следующий элемент;

- для запоминания номера сегмента расположения элемента;
- для ссылки на предыдущий элемент (верный);
- для расположения элемента в списке памяти.

8. Чем отличается кольцевой список от линейного ?
   - в кольцевом списке последний элемент является одновременно и первым;
   - в кольцевом списке указатель последнего элемента пустой;
   - в кольцевых списках последнего элемента нет (верный);
   - в кольцевом списке указатель последнего элемента не пустой.
9. Сколько указателей используется в односвязном кольцевом списке ?
   - 1(верный);
   - 2;
   - сколько угодно.
10. В каких направлениях можно перемещаться в кольцевом двунаправленном списке ?
   - в обоих (верный);
   - влево;
   - вправо.

Лабораторная работа 4. Модель массового обслуживания.

6. Чем отличается заявка первого приоритета от заявки второго приоритета ?
   - тем, что заявка второго приоритета обслуживается с вероятностью P=1, а заявка первого приоритета обслуживается с вероятностью P(B);
   - тем, что заявка второго приоритета становится в начало очереди, а первого приоритета становится в конец очереди (верный);
   - ничем, если есть очередь.

7. Может ли заявка первого приоритета вытеснить из очереди заявку второго приоритета ?
   - да, если P(B)=1;
   - да;
   - нет (верный).
8. Может ли на обслуживании находится заявка первого приоритета, если в очереди находится заявка второго приоритета ?
   - да, если P(B)=1;
   - да (верный);
   - нет.
9. С помощью какой структуры данных наиболее рационально реализовать очередь ?
   - стек;
   - список (верный);
   - дек.
10. Когда заявка покидает систему. Найдите ошибку.
   - если заявка обслужилась подложенное ей число тактов;
   - если заявка находится в очереди больше T тактов;
   - если заявок второго приоритета стало больше, чем заявок первого приоритета (верный).

<u>Лабораторная работа 5.</u> Бинарные деревья (основные процедуры).

6. Для включения новой вершины в дерево нужно найти узел, к которому её можно присоединить. Узел будет найден, если очередной ссылкой, определяющей ветвь дерева, в которой надо продолжать поиск, окажется ссылка:
   - p=right(p);
   - p=nil (верный);
   - p=left(p).
7. Для написания процедуры над двумя деревьями необходимо описать элемент типа запись, который содержит поля:
   - Element=Запись

Left,Right : Указатели
Rec : Запись;

- Element=Запись
  Left : Указатель
  Key : Ключ
  Rec : Запись;

- Element=Запись (верный)
  Left, Right : Указатели
  Key : Ключ
  Rec : Запись.

8. В памяти ЭВМ бинарное дерево удобно представлять в виде:
   - связанных линейных списков;
   - массивов;
   - связанных нелинейных списков (верный).
9. Элемент t, на котрый нет ссылок:
   - корнем (верный);
   - промежуточным;
   - терминальным (лист).
10. Дерево называется полным бинарным, если степень исходов вершин равна:
    - 2 или 0 (верный);
    - 2;
    - M или 0;
    - M.

Лабораторная работа 6. Сортировка методом прямого включения.

6. Даны три условия окончания просеивания при сортировке прямым включением. Найдите среди них лишнее.
   - найден элемент a(i) с ключом, меньшим чем ключ у x;

- найден элемент a(i) с ключом, большим чем ключ у x (верный);
- достигнут левый конец готовой последовательности.

7. Какой из критериев эффективности сортировки определяется формулой M=0,01*n*n+10*n ?
   - число сравнений (верный);
   - время, затраченное на написание программы;
   - количество перемещений;
   - время, затраченное на сортировку.
8. Как называется сортировка, происходящая в оперативной памяти ?
   - сортировка таблицы адресов;
   - полная сортировка;
   - сортировка прямым включением;
   - внутренняя сортировка (верный);
   - внешняя сортировка.
9. Как можно сократить затраты машинного времени при сортировке большого объёма данных ?
   - производить сортировку в таблице адресов ключей (верный);
   - производить сортировку на более мощном компьютере;
   - разбить данные на более мелкие порции и сортировать их.
10. Существуют следующие методы сортировки. Найдите ошибку.
    - строгие;
    - улудшенные;
    - динамические (верный).

Лабораторная работа 7. Сортировка методом прямого выбора.

6. Метод сортировки называется устойчивым, если в процессе сортировки…
   - относительное расположение элементов безразлично;

- относительное расположение элементов с равными ключами не меняется (верный);
- относительное расположение элементов с равными ключами изменяется;
- относительное расположение элементов не определено.

7. Улучшенные методы имеют значительное преимущество:
   - при большом количестве сортируемых элементов (верный);
   - когда массив обратно упорядочен;
   - при малых количествах сортируемых элементов;
   - во всех случаях.
8. Что из перечисленных ниже понятий является одним из типов сортировки ?
   - внутренняя сортировка (верный);
   - сортировка по убыванию;
   - сортировка данных;
   - сортировка по возрастанию.
9. Сколько сравнений требует улучшенный алгоритм сортировки ?
   - n*log(n) (верный);
   - $e^n$;
   - n*n/4.
10. К какому методу относится сортировка, требующая n*n сравнений ключей ?
    - прямому (верный);
    - бинарному;
    - простейшему;
    - обратному.

Лабораторная работа 8. Сортировка с помощью прямого обмена.

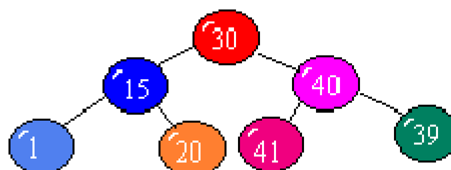
6. Сколько сравнений и перестановок элементов требуется в пузырьковой сортировке ?
   - n*lon(n);

- (n*n)/4 (верный);
- (n*n-n)/2.

7. Сколько дополнительных переменных нужно в пузырьковой сортировке помимо массива, содержащего элементы ?
   - 0 (не нужно);
   - всего 1 элемент (верный);
   - n переменных (ровно столько, сколько элементов в массиве).
8. Как рассортировать массив быстрее, пользуясь пузырьковым методом ?
   - одинаково (верный);
   - по возрачстанию элементов;
   - по убыванию элементов.
9. В чём заключается идея метода QuickSort ?
   - выбор 1,2,…n – го элемента для сравнения с остальными;
   - разделение ключей по отношению к выбранному (верный);
   - обмен местами между соседними элементами.
10. Массив сортируется “пузырьковым” методом. За сколько проходов по массиву самый “лёгкий” элемент в массиве окажется вверху ?
   - за 1 проход (верный);
   - за n-1 проходов;
   - за n проходов, где n – число элементов массива.

Лабораторная работа 9. Сортировка с помощью дерева.

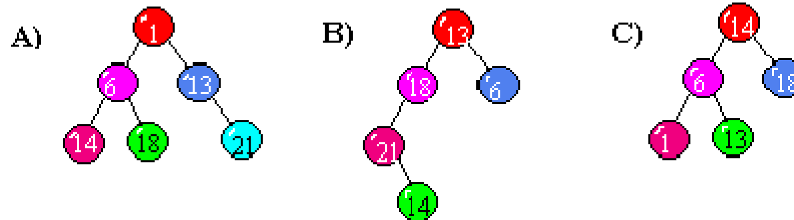
6. При обходе дерева




слева направо получаем последовательность…

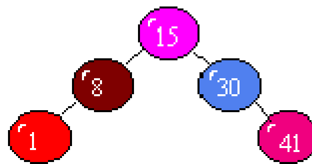
- отсортированную по убыванию;
- неотсортированную (верный);
- отсортированную по возрастанию.

7. Какое из трёх деревьев не является строго сбалансированным ?



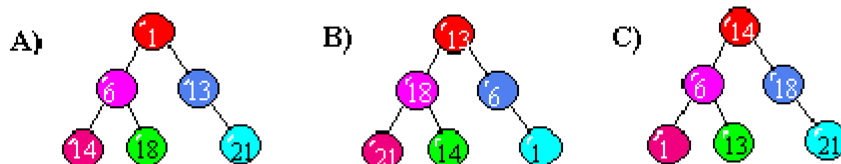

- A;
- B(верный);
- C.

8. При обходе дерева слева направо его элемент заносится в массив…
   - при втором заходе в элемент (верный);
   - при первом заходе в элемент;
   - при третьем заходе в элемент.
9. Элемент массива с ключом k=20 необходимо вставить в изображённое дерево так, чтобы дерево осталось отсортированным. Куда его нужно вставить ?




- левым сыном элемента 30 (верный);
- левым сыном элемента 41;
- левым сыном элемента 8.

10. При обходе какого дерева слева направо получается отсортированный по возрастанию массив ?




- A;
- B;

- С (верный).

Лабораторная работа 10. Исследование методов линейного и бинарного поиска.

6. Где эффективен линейный поиск ?
   - в списке;
   - в массиве;
   - в массиве и в списке (верный).
7. Какой поиск эффективнее ?
   - линейный;
   - бинарный (верный);
   - без разницы.
8. В чём суть бинарного поиска ?
   - нахожденние элемента массива x путём деления массива пополам каждый раз, пока элемент не найден (верный);
   - нахождение элемента x путём обхода массива;
   - нахождение элемента массива x путём деления массива.
9. Как расположены элементы в массиве бинарного поиска ?
   - по возрастанию (верный);
   - хаотично;
   - по убыванию.
10. В чём суть линейного поиска ?
   - производится последовательный просмотр от начала до конца и обратно через 2 элемента;
   - производится последовательный просмотр элементов от середины таблицы;
   - производится последовательный просмотр каждого элемента (верный).

Лабораторная работа 11. Исследование методов поиска с перемещением в начало и транспозицией.

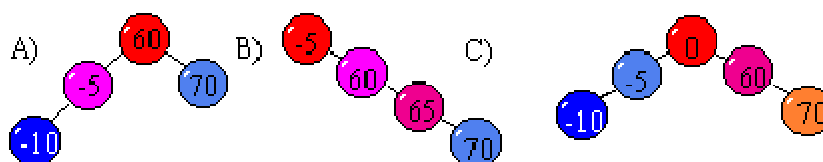
6. Где наиболее эффективен метод транспозиций ?
   - в массивах и в списках (верный);

- только в массивах;
- только в списках.

7. В чём суть метода перестановки ?
   - найденный элемент помещается в голову списка (верный);
   - найденный элемент помещается в конец списка;
   - найденный элемент меняется местами с последующим.
8. В чём суть метода транспозиции ?
   - перестановка местами соседних элементов;
   - нахождение одинаковых элементов;
   - перестановка найденного элемента на одну позицию в сторону начала списка (верный).
9. Что такое уникальный ключ ?
   - если разность значений двух данных равна ключу;
   - если сумма значений двух данных равна ключу;
   - если в таблице есть только одно данное с таким ключом (верный).
10. В чём состоит назначение поиска ?
   - среди массива данных найти те данные, которые соответствуют заданному аргументу (верный);
   - определить, что данных в массиве нет;
   - с помощью данных найти аргумент.

Лабораторная работа 12. Поиск по дереву с включением.

6. В каком дереве при бинарном поике нужно перебрать в среднем N/2 элементов ?




- A;
- B (верный);

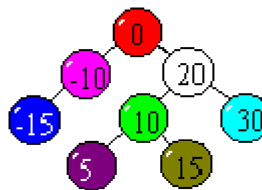
- C.

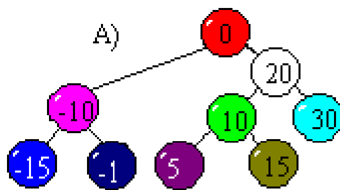
7. Сколько нужно перебрать элементов в сбалансированном дереве ?

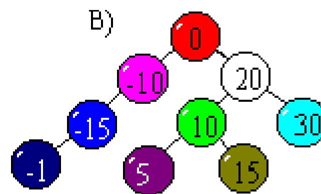
E) N/2;
F) Ln(N);
G) $Log_2(N)$;
H) $e^N$.

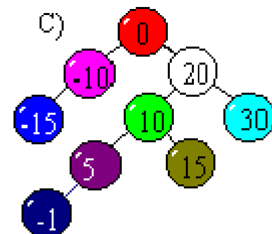
- A;
- B;
- C (верный);
- D.

8. Выберете вариант дерева, полученного после вставки узла -1.



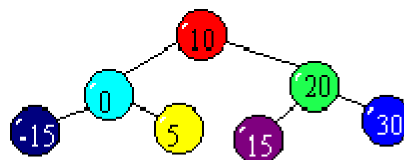




- A (верный);
- B;
- C.

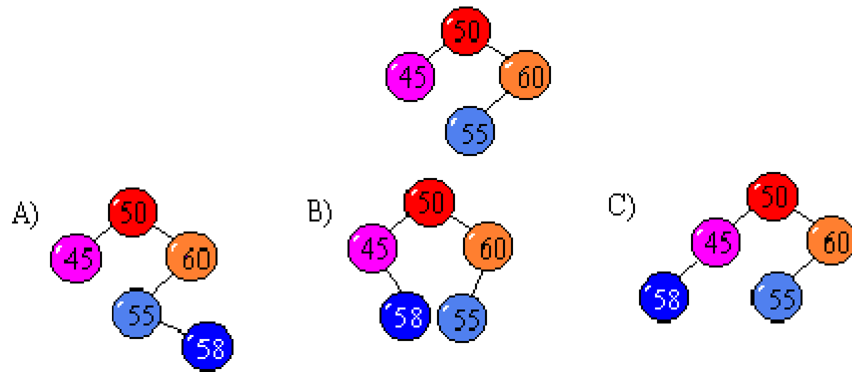
9. К какому элементу присоединить элемент 40 для вставки его в данное дерево ?




- к 30-му (верный);

- к 15-му;
- к –15-му;
- к 5-му.

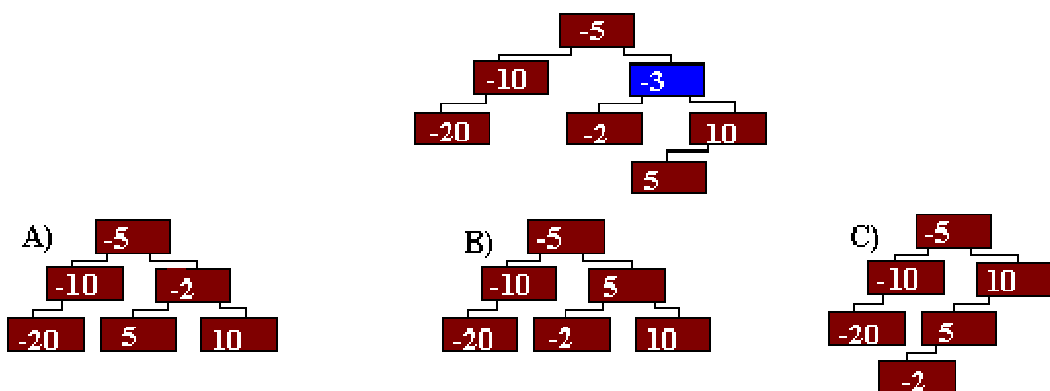
10. Какой вид примет дерево после встаки элемента с ключом 58 ?




- A (верный);
- B;
- C.

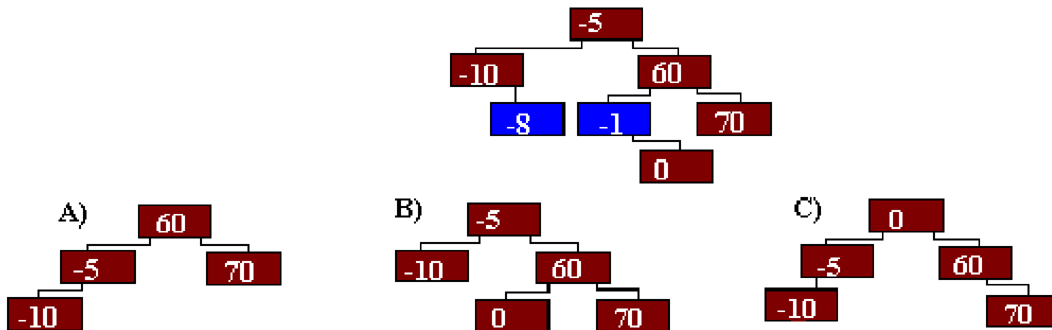
Лабораторная работа 13. Поиск по дереву с исключением.

6. Выберете вариант дерева, полученного после удаления узла –3.



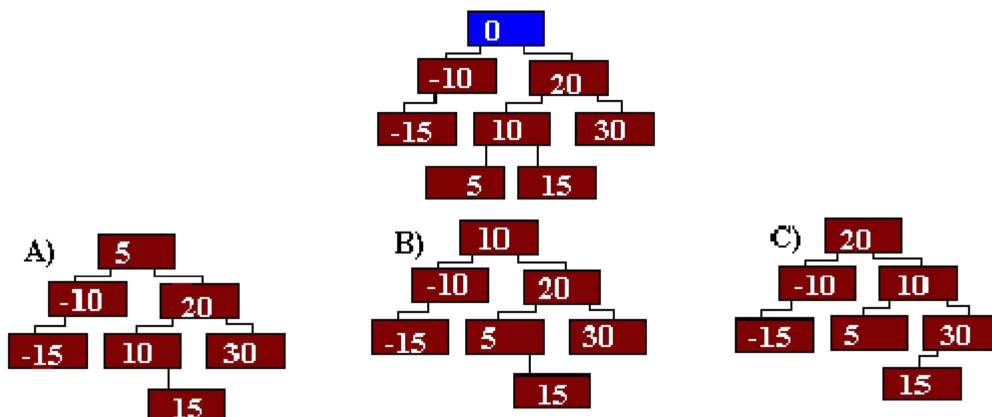

- A;
- B (верный);
- C.

7. Какой вариант дерева получится после удаления элемента –1, а затем –8 ?



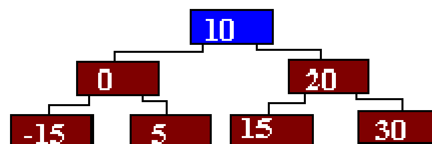

- A;
- B (верный);
- C.

8. Выберете вариант дерева, полученного после удаления узла с индексом 0.




- A (верный);
- B;
- C.

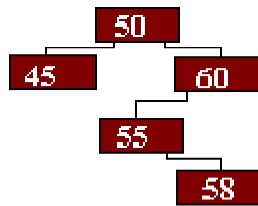
9. Какие из следующих пар чисел могут стать корнями дерева после удаления элемента 10 в соответсвии с двумя способами удаления узла, имеющего двух сыновей ?

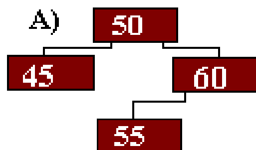


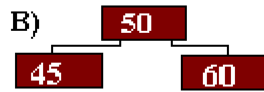

- 0 или 15;
- 0 или 20;
- 5 или 30;

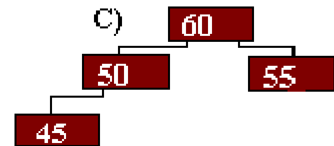
- 5 или 15 (верный).

10. Какой вид примет дерево после удаления элемента с ключом 58 ?




- A (верный);
- B;
- C.

**Лойко** Валерий Иванович

Структуры и алгоритмы
обработки данных

Учебное пособие для вузов

Авторская правка

ЛР № 02334 от 14.07.2004
Подписано в печать 2.11.2000
Бумага Типографская
Печ. л. 13,5

Формат 60 х 84
Офсетная печать
Заказ № 618
Тираж 500

350044, Краснодар, Калинина, 13

Отпечатано в типографии КубГАУ