МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Чувашский государственный университет имени И.Н. Ульянова

Пичугин В.Н., Фёдоров Р.В.

СТРУКТУРЫ И АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ОБРАБОТКИ ДАННЫХ

Лабораторный практикум

Чебоксары 2004

Введение.

Выполнение лабораторных работ предполагает четыре этапа: подготовка, непосредственно выполнение, оформление отчета и защита работы.

Подготовка к работе. Выполняется в часы самостоятельных занятий и заключается в следующем.

1. Изучение описания работы и других теоретических сведений, касающихся тематики выполняемой работы.
2. Анализ задач и при необходимости, их формализация. Формализация требуется в том случае, если имеется неформальная постановка задачи, и заключается в переходе от неформальной словесной постановки к математической формулировке. В результате анализа задачи определяются методы решения.
3. Выбор и описание структур данных и разработка алгоритмов. Задачи выбора структур данных и разработки алгоритмов взаимосвязаны; во многих случаях, существует прямая зависимость алгоритма от выбранной структуры данных, и наоборот, алгоритм определяет необходимые структуры данных. Выбранные структуры данных должны быть обоснованы и подробно описаны в отчете о работе. Алгоритмы могут быть представлены в произвольной форме: по шагам, в виде блок-схемы, с помощью псевдокода (используя основные конструкции языков программирования) и т. п. Для каждого алгоритма должны быть четко определены вход (исходные данные) и выход (результаты вычислений). Следует помнить, что алгоритм является формальным описанием вычислительной процедуры и в нем необходимо использовать обозначения, определенные в формальной постановке задачи. Поэтому алгоритм не должен представляться текстом программы; излишние подробности, связанные с требованиями синтаксиса языка программирования, обычно заслоняют существо дела.
4. Программная реализация. Заключается в представлении структур данных и алгоритмов соответствующими конструкциями выбранного языка программирования. При этом должны быть указаны идентификаторы переменных, подпрограмм и т. п., сопоставленные соответствующим элементам формального алгоритма. Тексты программ должны содержать подробные комментарии» поясняющие назначение процедур, их параметров, использование переменных, смысл и особенности реализации отдельных программных блоков.
5. Подготовка к экспериментальным исследованиям. Этот пункт необходим для тех работ, в которых предусмотрены экспериментальные исследования алгоритмов. Заключается в разработке алгоритмов и программ проведения исследований в соответствии с технологией, приведенной в описании работы, и подготовке специальных таблиц для записи результатов исследований. При исследовании алгоритмов подсчитываются различные характеристики, и фиксируется время вычислений. Для подсчета характеристик необходимо предусмотреть в программах соответствующие счетчики. Фиксируется же только чистое время работы алгоритмов - операции с дополнительными счетчиками не должны учитываться. Поэтому для каждого исследуемого алгоритма следует использовать две программные реализации - со счетчиками (для подсчета значений характеристик) и без них (для фиксации времени). В результате подготовки к работе оформляется соответствующий раздел отчета. Если подготовка не выполнена, студент не допускается к выполнению лабораторной работы.

Выполнение работы. Заключается в вводе текстов программ, их трансляции и отладке на тестовых примерах, а также в проведении экспериментальных исследований. В ряде работ по полученным данным и времени вычислений требуется построить аппроксимирующие функции и вычислить аналитические зависимости времени вычислений от размера входа. Для этого можно использовать любой метод решения задачи аппроксимации. Порядок выполнения приводится в описании каждой работы. Выполнение работы подтверждается подписью преподавателя в отчете, что означает допуск к защите работы.

Оформление отчета. **Отчет о работе должен содержать:**

1. Титульный лист.
2. Цель работы.
3. Информацию в соответствии с подготовкой к работе.
4. Результаты выполнения работы (результаты решения задач, таблицы, графики, аналитические и/или экспериментальные зависимости и т. д.).
5. Выводы по работе, отражающие анализ результатов и предпочтительные области применения разработанных структур данных и алгоритмов.
6. Тексты программ в виде текстовых файлов.

Защита работы. Заключается в опросе студента преподавателем. Преподаватель вправе задавать вопросы по всем разделам отчета: теоретической части, аспектам выбора структур данных и разработки алгоритмов, программной реализации, результатам исследований. По итогам опроса решается вопрос о защите лабораторной работы.

Лабораторная работа 1.

Поиск значений.

Цель работы: ознакомление с методами решения задач поиска, а именно поиска в таблице, в соответствии с данным заданием, получение навыков программирования задач поиска элементов.

Исходные теоретические сведения

Поиск в массиве иногда называют поиском в таблице, особенно если ключ сам является составным объектом, таким, как массив чисел или символов. Часто встречается именно последний случай, когда массивы символов называют строками или словами. Строковый тип определяется так:

String = array[0..M–1] of char

соответственно определяется и отношение порядка для строк x и y:

x = y, если xj = yj для 0 face="Symbol" =< j < M

x < y, если xi < yi для 0 face="Symbol" =< i < M и xj = yj для 0 face="Symbol" =< j < I

Для того чтобы установить факт совпадения, необходимо установить, что все символы сравниваемых строк соответственно равны один другому. Поэтому сравнение составных операндов сводится к поиску их несовпадающих частей, т. е. к поиску “на неравенство”. Если неравных частей не существует, то можно говорить о равенстве. Предположим, что размер слов достаточно мал, скажем, меньше 30. В этом случае можно использовать линейный поиск и поступать таким образом.

Для большинства практических приложений желательно исходить из того, что строки имеют переменный размер. Это предполагает, что размер указывается в каждой отдельной строке. Если исходить из ранее описанного типа, то размер не должен превосходить максимального размера M. Такая схема достаточно гибка и подходит для многих случаев,

в то же время она позволяет избежать сложностей динамического распределения памяти. Чаще всего используются два таких представления размера строк:

Размер неявно указывается путем добавления концевого символа, больше этот символ нигде не употребляется. Обычно для этой цели используется “непечатаемый” символ со значением 00h. (Для дальнейшего важно, что это минимальный символ из всего множества символов.)

Размер явно хранится в качестве первого элемента массива, т. е. строка s имеет следующий вид: s = s0, s1, s2, ..., sM-1. Здесь s1, ..., sM-1 – фактические символы строки, а s0 = Chr(M). Такой прием имеет то преимущество, что размер явно доступен, недостаток же в том, что этот размер ограничен размером множества символов (256).

В последующем алгоритме поиска отдается предпочтение первой схеме. В этом случае сравнение строк выполняется так:

```
i:=0;
while (x[i]=y[i]) and (x[i]<>00h) do i:=i+1
```

Концевой символ работает здесь как барьер.

Теперь вернемся к задаче поиска в таблице. Он требует “вложенных” поисков, а именно: поиска по строчкам таблицы, а для каждой строчки последовательных сравнений – между компонентами. Например, пусть таблица T и аргумент поиска x определяются таким образом:

```
T: array[0..N-1] of String;
x: String
```

Допустим, N достаточно велико, а таблица упорядочена в алфавитном порядке. При использовании алгоритма поиска делением пополам и алгоритма сравнения строк, речь о которых шла выше, получаем такой фрагмент программы:

```
L:=0; R:=N;
while L<R do begin
        m:=(L+R) div 2; i:=0;
        while (T[m,i]=x[i]) and (x[i]<>00h) do i:=i+1;
              if T[m,i]<x[i] then L:=m+1 else R:=m
end;
if R<N then begin
       i:=0;
        while (T[R,i]=x[i]) and (x[i]<>00h) do i:=i+1
end
{(R<N) and (T[R,i]=x[i]) фиксирует совпадение}
```

Но при достаточно большом значении N можно использовать алгоритм линейного поиска и алгоритм последовательного сравнения строк.

Варианты заданий.

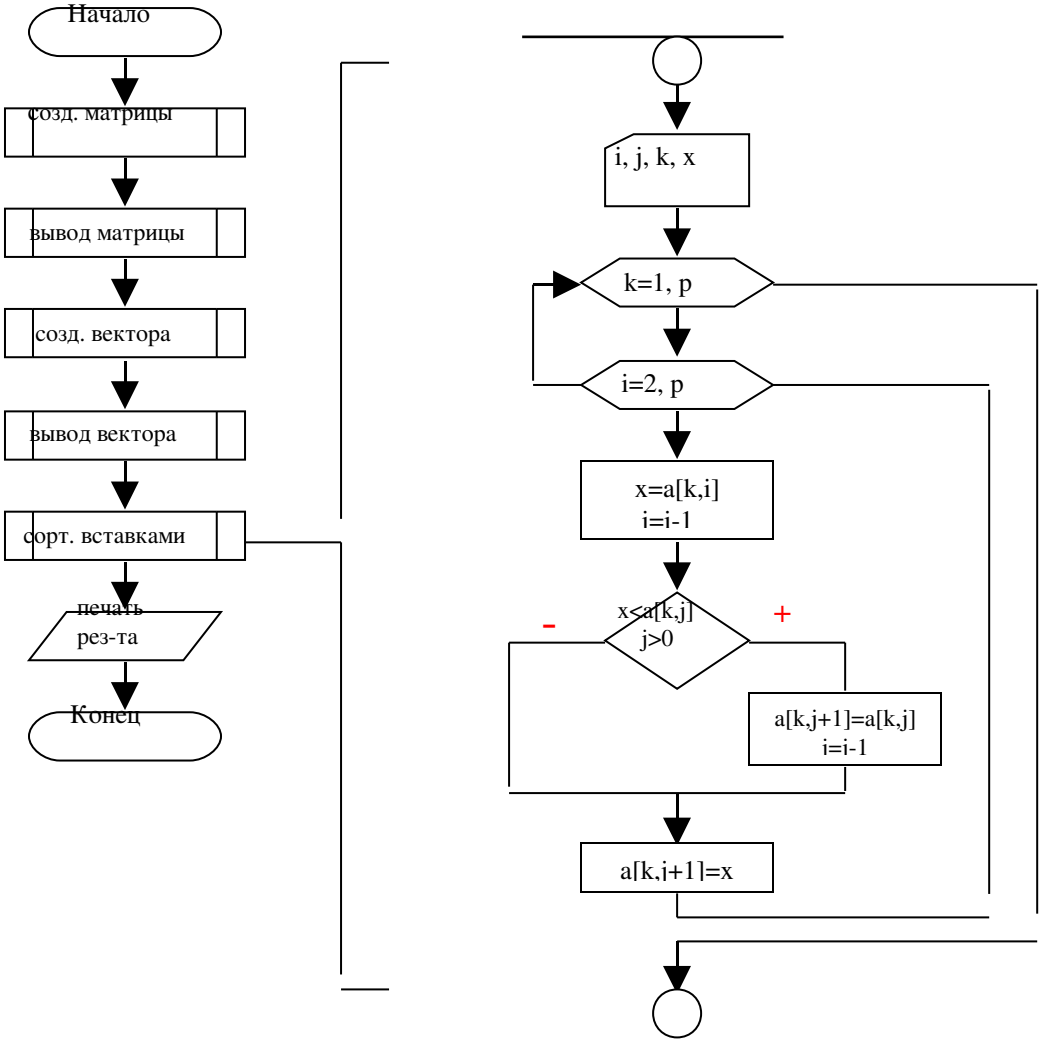
Вариант	Задание
1	Разработать алгоритм и программу последовательного поиска значений в одномерном массиве.
2	Разработать алгоритм и программу бинарного поиска значений в одномерном массиве.
3	Разработать алгоритм и программу последовательного поиска значений в таблице данных.
4	Разработать алгоритм и программу бинарного поиска значений в таблице данных.
5	Разработать алгоритм и программу поиска положительных элементов в

	каждой строке матрицы.
6	Разработать алгоритм и программу поиска положительных элементов в каждом столбце матрицы.
7	Разработать алгоритм и программу поиска элементов, кратных 2, в каждой строке матрицы.
8	Разработать алгоритм и программу поиска положительных элементов в главной диагонали матрицы.
9	Разработать алгоритм и программу поиска положительных элементов в нижней треугольной матрице.
10	Разработать алгоритм и программу поиска наименьшего элемента в матрице.

Пример выполнения.

Задание: осуществить последовательный поиск совпадений в таблице, используя сортировку вставкой.

Блок-схема алгоритма



Текст программы:

```
uses crt;
type
   data=integer;
```

```
    matrix=array[1..15,1..15] of integer;
    vector=array[1..15] of integer;
var
  a:matrix;
  x:vector;
  n,i,j,k:word;
  l,m,r:integer;
  found:boolean;

procedure create_matrix(var a:matrix;p:word);
var
  i,j:word;
begin
    writeln('Матрица создана:');
    for i:=1 to p do
    for j:=1 to p do a[i,j]:=2*(9+random(100));
end;

procedure vivod_matrix(var a:matrix;p:word);
var
  i,j:word;
begin
    for i:=1 to p do
    begin
        for j:=1 to p do write(a[i,j]:6);
        writeln;
        end;
end;
procedure create_vector(var x:vector;p:word);
var
  i:word;
begin
    writeln('Вектор создан:');
    for i:=1 to p do x[i]:=random(100);
end;

procedure vivod_vector(var x:vector;p:word);
var
  i:word;
begin
    for i:=1 to p do write(x[i]:6);
    writeln;
end;

procedure insert(var a:matrix;p:word);
var
  i,j,k:integer;
  x:data;
begin
    writeln('Сортировка вставкой:');
```

```
    for k:=1 to p do
    for i:=2 to p do
    begin
        x:=a[k,i];
        j:=i-1;
        while (x<a[k,j]) and (j>0) do
        begin
            a[k,j+1]:=a[k,j];
            j:=j-1;
        end;
        a[k,j+1]:=x;
    end;
end;

begin
    clrscr;
    write('Введите число элементов массива: n=');
    readln(n);
    create_matrix(a,n);
    vivod_matrix(a,n);
    writeln('Упорядоченная матрица в строках:');
    insert(a,n);
    vivod_matrix(a,n);
    create_vector(x,n);
    vivod_vector(x,n);
    writeln('Совпадения элементов матрицы и вектора');
    writeln('-------------------------------------');
    for i:=1 to n do
    begin
        for j:=1 to n do
        for k:=1 to n do if a[i,j]=x[k] then writeln(x[k],': совпадение в ',i,'-ой строке матрицы');
    end;
    writeln('нажмите любую клавишу');
    readkey;
end.
```

Результаты расчетов:

```
Turbo Pascal
Матрица создана:
    18    24   190    58    72
   152    80    50    92   102
    34   112    32   186    28
    76   200    90   172    82
   156   186   160    78    50
Упорядоченная матрица в строках:
Сортировка вставкой:
    18    24    58    72   190
    50    80    92   102   152
    28    32    34   112   186
    76    82    90   172   200
    50    78   156   160   186
Вектор создан:
    32    46    24    82    27
Совпадения элементов матрицы и вектора
-------------------------------------
24: совпадение в 1-ой строке матрицы
32: совпадение в 3-ой строке матрицы
82: совпадение в 4-ой строке матрицы

Нажмите любую клавишу
```

```
Turbo Pascal
Введите число элементов массива: n=7
Матрица создана:
    18    24   190    58    72   152    80
    50    92   102    34   112    32   186
    28    76   200    90   172    82   156
   186   160    78    50    82   110    66
   182    72   114    46   192    74   172
   212   116   194   182    22    46    46
   118    22   136    18   172   148   172
Упорядоченная матрица в строках:
Сортировка вставкой:
    18    24    58    72    80   152   190
    32    34    50    92   102   112   186
    28    76    82    90   156   172   200
    50    66    78    82   110   160   186
    46    72    74   114   172   182   192
    22    46    46   116   182   194   212
    18    22   118   136   148   172   172
Вектор создан:
    70    55    20    68    59    95    64
Совпадения элементов матрицы и вектора
---------------------------------------

Нажмите любую клавишу
```

Вывод: ознакомиться с методами решения задач поиска, а именно поиска в таблице, в соответствии с данным заданием, получить некоторые практические навыки программирования задач поиска элементов.

Лабораторная работа 2.

Сортировка значений в таблице.

Цель работы: изучение методов сортировки массивов.

Исходные теоретические сведения

Сортировка представляет собой процесс упорядочения множества подобных информационных объектов в порядке возрастания или убывания их значений. Например, список i из n элементов будет отсортирован в порядке возрастания значений элементов, если i <= i <= ... <= I

В общем случае при сортировке данных только часть информации используется в качестве ключа сортировки, который используется в сравнениях. Когда выполняется обмен, передается вся структура данных. Например, в списке почтовых отправлений в качестве ключа сортировки может использоваться почтовый индекс, а в операциях обмена к почтовому индексу добавляются полное имя и адрес.

Классы алгоритмов сортировки

Имеется три способа сортировки массивов:

- сортировка обменом;
- сортировка выбором;
- сортировка вставкой.

Представьте, что перед вами лежит колода карт. Для сортировки карт обменом вы должны разложить карты на столе лицевой стороной вверх и затем менять местами те карты, которые расположены в неправильном порядке, делая это до тех пор, пока колода карт не станет упорядоченной.

Для сортировки выбором вы должны разложить карты на столе, выбрать самую младшую карту и взять ее в свою руку. Затем вы должны из оставшихся на столе карт вновь выбрать наименьшую по значению карту и поместить ее позади той карты, которая уже имеется у вас в руке. Этот процесс вы должны продолжать до тех пор, пока все карты не окажутся у вас в руках. Поскольку каждый раз вы выбираете наименьшую по значению карту из оставшихся на столе, по завершению такого процесса карты у вас в руке будут отсортированы.

Для сортировки вставкой вы должны держать карты в своей руке, поочередно снимая карту с колоды. Каждая взятая вами карта помещается в новую колоду на столе, причем она ставится на соответствующее место. Колода будет отсортирована, когда у вас в руке не окажется ни одной карты.

Сортировка Шелла

Сортировка Шелла получила свое название по имени ее создателя Д.Л. Шелла. Однако, это название можно считать удачным, так как выполняемые при сортировке действия напоминают укладывание морских ракушек друг на друга. ("Ракушка" - одно из значений слова shell).

Общий метод, который использует сортировку вставкой, применяет принцип уменьшения расстояния между сравниваемыми элементами. Далее показана схема выполнения сортировки Шелла для массива "оасве". Сначала сортируются все элементы, которые смещены друг от друга на три позиции. Затем сортируются все элементы, которые смещены на две позиции. И, наконец, упорядочиваются все соседние элементы.

- проход 1 f d a c b e

- проход 2 c b a f d e
- проход 3 a b c e d f
- полученный результат a b c d e f

```
procedure shell(var a:massiv;p:word);
const
   t=5;
var
  i,j,k,s,m:integer;
  h:array[1..t] of integer;
  x:data;
begin
    writeln('Сортировка Шелла:');
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=2; h[5]:=1;
    for m:=1 to t do
    begin
        k:=h[m];
        s:=-k;
        for i:=k+1 to p do
        begin
            x:=a[i];
            j:=i-k;
            if s=0 then
            begin
                s:=-k;
                s:=s+1;
                a[s]:=x;
            end;
            while (x<a[j]) and (j<p) do
              begin
                  a[j+k]:=a[j];
                  j:=j-k;
              end;
            a[j+k]:=x;
        end;
    end;
end;
```

При поверхностном взгляде на алгоритм нельзя сказать, что он дает хороший результат и даже то, что в результате получится отсортированный массив. Однако, он дает и то и другое. Эффективность этого алгоритма объясняется тем, что при каждом проходе используется относительно небольшое число элементов или элементы массива уже находятся в относительном порядке, а упорядоченность увеличивается при каждом новом просмотре данных. Расстояния между сравниваемыми элементами могут изменяться по-разному. Обязательным является лишь то, что последний шаг должен равняться единице. Например, хорошие результаты дает последовательность шагов 9, 5, 3, 2, 1, которая использована в показанном выше примере. Следует избегать последовательностей степени двойки, которые, как показывают сложные математические выкладки, снижают эффективность алгоритма сортировки. Однако, при использовании таких последовательностей шагов между сравниваемыми элементами эта сортировка будет по-прежнему работать правильно.

Внутренний цикл имеет два условия проверки. Условие "x<a[j]" необходимо для упорядочения элементов. Условия "j>0" и "j<=count" необходимы для того, чтобы предотвратить выход за пределы массива "a". Эта дополнительная проверка в некоторой степени ухудшает сортировку Шелла. Слегка измененные версии сортировки Шелла используют специальные управляющие элементы, которые не являются в действительности частью той информации, которая должна сортироваться. Управляющие элементы имеют граничные для массива данных значения, т.е. наименьшее и наибольшее значения. В этом случае не обязательно выполнять проверку на граничные значения. Однако, применение таких управляющих элементов требует специальных знаний о той информации, которая сортируется, и это снижает универсальность процедуры сортировки. Анализ сортировки Шелла требует решения некоторых сложных математических задач

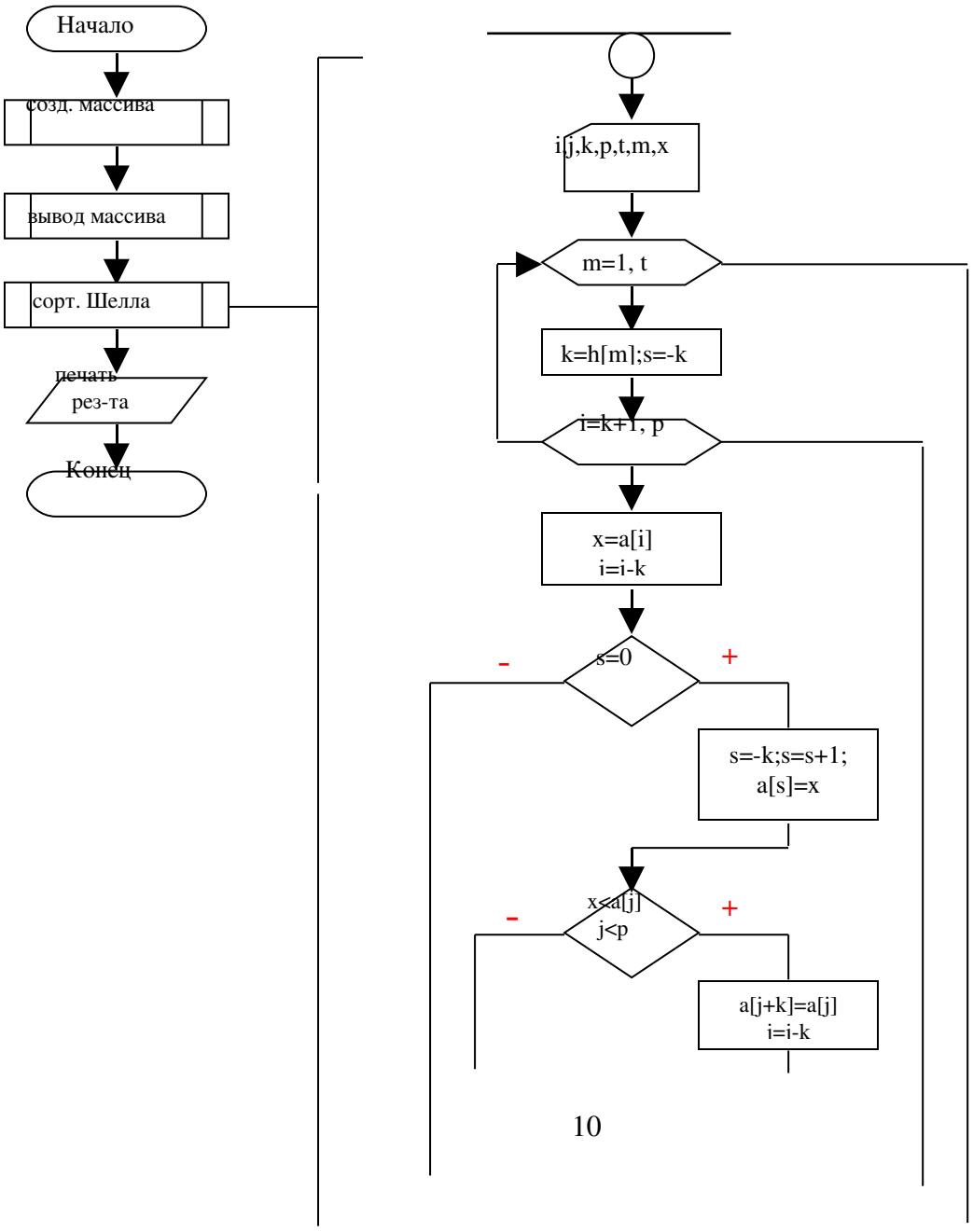
Время выполнения сортировки Шелла пропорционально n1.2. Эта зависимость значительно лучше квадратичной зависимости, которой подчиняются рассмотренные ранее алгоритмы сортировки.

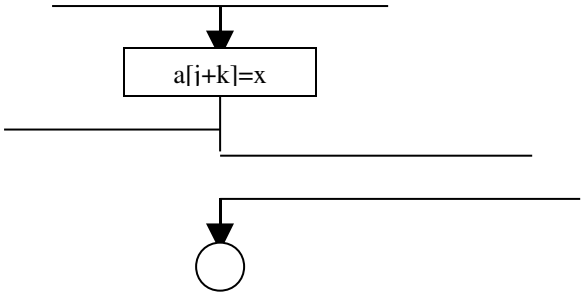
Приведем пример программы, демонстрирующей сортировку массива произвольного размера, составленного из случайных чисел.

Пример выполнения.

Задание: составить алгоритм и программу сортировки массива методом Шелла.

Блок-схема алгоритма





Текст программы:

```
uses crt;
type
   data=integer;
   massiv=array[1..15] of integer;
var
  a,b:massiv;
  n,i:word;

procedure create_massive(var a:massiv;p:word);
var
  i:word;
begin
    writeln('Массив создан:');
    for i:=1 to p do
    begin
        a[i]:=2*(9+random(100));
        write(a[i]:5);
    end;
end;

procedure vivod_massive(var a:massiv;p:word);
var
  i:word;
begin
    for i:=1 to p do write(a[i]:5);
end;

procedure shell(var a:massiv;p:word);
const
    t=5;
var
  i,j,k,s,m:integer;
  h:array[1..t] of integer;
  x:data;
begin
    writeln('Сортировка Шелла:');
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=2; h[5]:=1;
    for m:=1 to t do
    begin
        k:=h[m];
        s:=-k;
        for i:=k+1 to p do
        begin
            x:=a[i];
```

```
            j:=i-k;
            if s=0 then
            begin
                s:=-k;
                s:=s+1;
                a[s]:=x;
            end;
            while (x<a[j]) and (j<p) do
              begin
                  a[j+k]:=a[j];
                  j:=j-k;
              end;
            a[j+k]:=x;
        end;
    end;
end;

begin
    clrscr;
    write('Введите число элементов массива 2..15: ');
    readln(n);
    create_massive(b,n);
    writeln;
    shell(b,n);
    vivod_massive(b,n);
    readkey;
end.
```

Результаты расчетов:

Варианты заданий.

Вариант	Задание
1	Разработать алгоритм и программу внутренней сортировки значений в таблице (пузырьковая сортировка).
2	Разработать алгоритм и программу внутренней сортировки значений в таблице (сортировка простыми вставками).
3	Разработать алгоритм и программу внутренней сортировки значений в таблице (простая сортировка выбором).
4	Разработать алгоритм и программу внутренней сортировки значений в таблице (метод Шелла).
5	Разработать алгоритм и программу внутренней сортировки значений в таблице (бинарные вставки).
6	Разработать алгоритм и программу внутренней сортировки значений в

	таблице (быстрая сортировка).
7	Разработать алгоритм и программу внутренней сортировки значений в таблице (сортировка выбором).
8	Разработать алгоритм и программу внешней сортировки значений в таблице (простое слияние).
9	Разработать алгоритм и программу внешней сортировки значений в таблице (естественное слияние).
10	Разработать алгоритм и программу внешней сортировки значений в таблице (улучшенные методы сортировки).

Вывод: изучить методы сортировки массивов.

Лабораторная работа 3.

Стеки и очереди.

Цель работы: ознакомиться со способами реализации стеков и очередей и выполнения над ними операций. Получение практических навыков в программировании с использованием стеков и очередей.

Исходные теоретические сведения

Стеки и очереди – это динамические структуры данных с ограниченным видом операций включением новых и выключением старых переменных.

Стек представляет собой последовательность элементов с одной точкой доступа, называемой вершиной стека. Все элементы последовательности, кроме элемента, расположенного в вершине стека, недоступны. Новый элемент добавляется только в вершину стека, сдвигая остальные элементы последовательности. Исключить можно только элемент из вершины стека. Остальные элементы при этом сдвигаются в сторону вершины. Таким образом стек работает по принципу «последним пришел - первым ушел», и часто называется структурой LIFO (Last In – First Out).

В алгоритмах операцию добавления элемента в стек записывают в виде S←X (элемент X поместить в вершину стека S); операцию исключения в виде X←S (исключить элемент X из вершины стека S и присвоить его значение переменной X). В литературе эти операции традиционно называются PUSH (добавление) и POP (исключение).

Очередь представляет собой последовательность элементов с двумя точками доступа: начало (голова) и конец (хвост). Новый элемент добавляется всегда в конец очереди, исключается всегда элемент, расположенный в начале очереди. Таким образом очередь работает по принципу «первым зашел – первым ушел» и часто называется структурой FIFO (First In – First Out).

В алгоритмах операцию включения элемента в очередь записывают в виде Q←X (элемент X помещается в конец очереди Q), а операцию исключения из очереди – в виде X←Q (исключить элемент X из начала очереди Q и присвоить его значение переменной X).

Стеки и очереди удобно реализовывать с помощью указателей, т.к. указатели имеют весьма гибкую структуру.

Описание типа стек:

```
type
   ct=^celltype;
```

```
celltype=record
            element:byte;
            next:ct;
            end;
stack=record
          top:ct;
          fin:ct;
          end;
```

где top – указатель на вершину стека.

Описание типа очередь:

```
type
  ct=^celltype;
  celltype=record
            element:byte;
            next:ct;
            end;
  queue=record
          front:ct;
          rear:ct;

          end;
```

где front – указатель на начало очереди, rear – указатель на конец очереди.

Варианты заданий.

Вариант	Задание
1	Разработайте реализации для очередей, состоящих из действительных чисел, с использованием массивов и указателей.
2	Реализовать двухстороннюю очередь, если элементами являются действительных чисел.
3	Разработайте реализации для стека, состоящих из действительных чисел, с использованием массивов и указателей.
4	Напишите процедуру поиска элементов в позициях p для стека.
5	Напишите процедуру поиска элементов в позициях p для очереди.
6	Напишите процедуру сортировки элементов для очереди.
7	Напишите процедуру обмена элементами в позициях p и NEXT(p) для стека.
8	Напишите процедуру обмена элементами в позициях p и NEXT(p) для очереди.
9	Напишите программу сложения и умножения двух стеков.
10	Напишите программу сложения и умножения двух очередей.

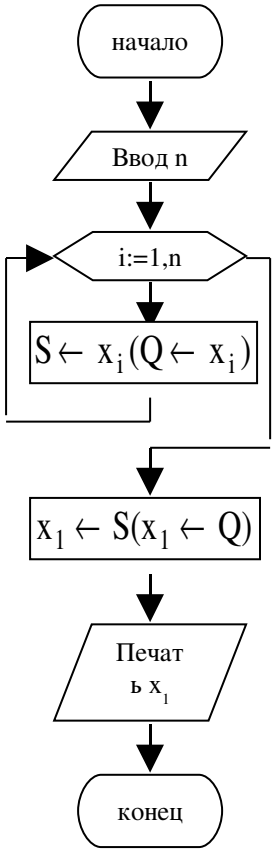
Пример выполнения.

Задание: написать программы, реализующие операции над стеками и очередями, с помощью указателей:

1. функция STACKTOP(Q), которая определяет значение элемента в вершине стека Q без его удаления (не определена для пустого стека);
2. процедура INSERT(S,X), которая помещает элемент X в конец очереди S;

3. функция REMOVE(S), которая удаляет элемент из начала очереди S, т.е. операция X:=REMOVE (S) удаляет элемент из начала очереди S и присваивает его значение переменной X (не определена для пустой очереди).

Блок-схема алгоритма



Реализация функции STACKTOP(Q):

```
function stacktop(var s:stack):byte;
begin
    stacktop:=s.top^.next^.element;
end;
```

Реализация процедуры INSERT(S,X):

```
procedure insert(var q:queue;x:byte);
begin
    new(q.rear^.next);
    q.rear:=q.rear^.next;
    q.rear^.element:=x;
    q.rear^.next:=nil;
end;
```

Реализация функции REMOVE(S):

```
function remove(var q:queue):byte;
begin
    remove:=q.front^.next^.element;
    delstart(q);
end;
```

Текст программы:

Очередь:

```
uses
   crt;
const
    n=10;
type
   ct=^celltype;
   celltype=record
               element:byte;
               next:ct;
               end;
   queue=record
            front:ct;
            rear:ct;

            end;
var
  s:queue;
  x:byte;

procedure insert(var q:queue;x:byte);
begin
    new(q.rear^.next);
    q.rear:=q.rear^.next;
    q.rear^.element:=x;
    q.rear^.next:=nil;
end;

procedure create(var q:queue);
var
  i,x:byte;
begin
    new(q.front);
    q.front^.next:=nil;
    q.rear:=q.front;
    for i:=1 to n do
    begin
        x:=round(random*100);
        insert(q,x);
    end;
    writeln('QUEUE IS CREATE.');
end;

procedure delstart(var q:queue);
begin
    q.front:=q.front^.next;
end;

function remove(var q:queue):byte;
begin
```

```
    remove:=q.front^.next^.element;
    delstart(q);
end;

procedure printqueue(var q:queue);
var
  temp:queue;
begin
    writeln('QUEUE:');
    temp:=q;
    while temp.front^.next<>nil do
    begin
        write(temp.front^.next^.element,' ');
        temp.front:=temp.front^.next;
    end;
    writeln;
end;
procedure delqueue(var q:queue);
begin
    while q.front^.next<>nil do delstart(q);
    writeln('QUEUE IS DELETE.');
end;

begin
    clrscr;
    randomize;
    create(s);
    printqueue(s);
    x:=remove(s);
    writeln('X:=',x);
    printqueue(s);
    delqueue(s);
    readkey;
end.
```

Стек:

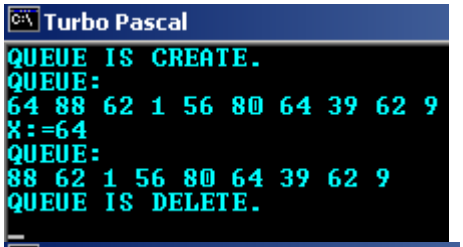
```
uses
   crt;
const
    n=10;
type
   ct=^celltype;
   celltype=record
               element:byte;
               next:ct;
               end;
   stack=record
            top:ct;
            fin:ct;
            end;
var
```

```
  q:stack;
  x:byte;

procedure push(var s:stack);
begin
    new(s.fin^.next);
    s.fin:=s.fin^.next;
    s.fin^.element:=round(random*100);
    s.fin^.next:=nil;
end;

procedure create(var s:stack);
var
  i:byte;
begin
    new(s.top);
    s.top^.next:=nil;
    s.fin:=s.top;
    for i:=1 to n do push(s);
    writeln('STACK IS CREATE.');
end;

procedure pop(var s:stack);
var
  temp:stack;
begin
    temp:=s;
    while temp.top^.next<>s.fin do temp.top:=temp.top^.next;
    s.fin:=temp.top;
end;

function stacktop(var s:stack):byte;
begin
    stacktop:=s.top^.next^.element;
end;

procedure printstack(var s:stack);
var
  temp:stack;
begin
    writeln('STACK:');
    temp:=s;
    while temp.top^.next<>nil do
    begin
        write(temp.top^.next^.element,' ');
        temp.top:=temp.top^.next;
    end;
    writeln;
end;
```

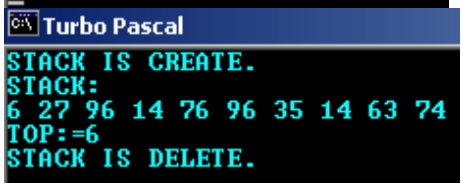
```
procedure delstack(var s:stack);
begin
    s.top^.next:=nil;
    writeln('STACK IS DELETE.');
end;

begin
    clrscr;
    randomize;
    create(q);
    printstack(q);
    x:=stacktop(q);
    writeln('TOP:=',x);
    delstack(q);
    readkey;
end.
```

Результаты расчетов.

Для очереди S: 64,88,62,1,56,80,64,39,62,9 результат работы функции REMOVE(S) будет следующим: X=64, S: 88,62,1,56,80,64,39,62,9.



Для стека Q:6,27,96,14,76,96,35,14,63,74 результат работы функции STACKTOP(Q) будет следующим: TOP=6.



Вывод: ознакомится со способами реализации стеков и очередей и выполнения над ними операций. Получить практические навыки в программировании с использованием стеков и очередей.

Лабораторная работа 4.

Бинарные деревья.

Цель работы: ознакомление со способами представления деревьев и методов их прохождения, получение практических навыков программирования задач с использованием деревьев.

Исходные теоретические сведения

В повседневной жизни мы очень часто встречаем примеры деревьев. Например, люди часто используют генеалогическое дерево для изображения структуры своего рода; как мы увидим, много терминов, связанных с деревьями, взято именно отсюда.

Второй пример - это структура большой организации; использование древообразной структуры для представления ее "иерархической структуры" ныне широко используется во многих компьютерных задачах.

Третий пример - это грамматическое дерево; изначально оно служило для грамматического анализа компьютерных программ, а ныне широко используется и для грамматического анализа литературного языка.

Мы начнем изучение деревьев с того, что определим их как некий абстрактный объект и введем всю связанную с ними терминологию.

Самые простые из деревьев считаются бинарные деревья.

Бинарное дерево - это конечное множество элементов, которое либо пусто, либо содержит один элемент, называемый корнем дерева, а остальные элементы множества делятся на два непересекающихся подмножества, каждое из которых само является бинарным деревом.

Эти подмножества называются левым и правым поддеревьями исходного дерева.

Каждый элемент бинарного дерева называется узлом дерева.

На рисунке показан общепринятый способ изображения бинарного дерева. Это дерево состоит из девяти узлов, A - корень дерева. Его левое поддерево имеет корень B, а правое-корень C. Это изображается двумя ветвями, исходящими из A: левым - к B и правым - к C. Отсутствие ветви обозначает пустое поддерево. Например, левое поддерево бинарного дерева с корнем C и правое поддерево бинарного дерева с корнем E оба пусты. Бинарные деревья с корнями D, G, H и I имеют пустые левые и правые поддеревья.

На рисунке приведены некоторые структуры, не являющиеся бинарными деревьями.

Если A - корень бинарного дерева и B - корень его левого или правого поддерева, то говорят, что A - отец B, а B - левый или правый сын A.

Узел, не имеющий сыновей (такие как узлы D, G, H и I), называется листом.

Узел nl -предок узла n2 (а n2-потомок nl), если nl-либо отец n2, либо отец некоторого предка n2. Например, в дереве из рисунке A-предок G и H-потомок C, но E не является ни предком, ни потомком C.

Узел n2-левый потомок узла n1, если n2 является либо левым сыном n1, либо потомком левого сына n1. Похожим образом может быть определен правый потомок.

Два узла являются братьями, если они сыновья одного и того же отца. Если каждый узел бинарного дерева, не являющийся листом, имеет непустые правые и левые поддеревья, то дерево называется строго бинарным деревом.

Свойство 1. Строго бинарное дерево с n листами всегда содержит 2n-1 узлов.

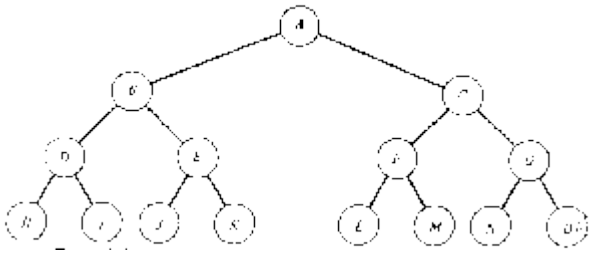
Уровень узла в бинарном дереве может быть определен следующим образом. Корень дерева имеет уровень 0, и уровень любого другого узла дерева имеет уровень на 1 больше уровня своего отца.

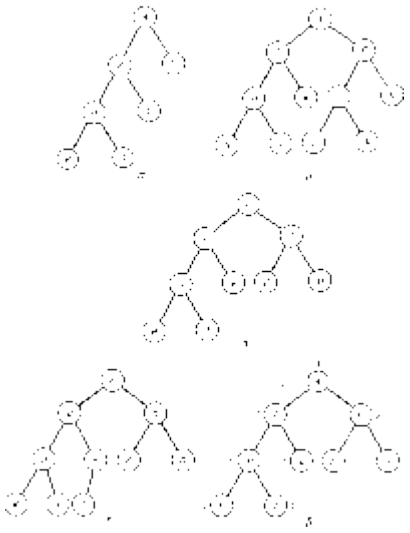
Например, в бинарном дереве на первом рисунке узел E- узел уровня 2, а узел H-уровня 3.

Глубина бинарного дерева - это максимальный уровень листа дерева, что равно длине самого длинного пути от корня к листу дерева.

Стало быть, глубина дерева на первом рисунке равна 3.

Полное бинарное дерево уровня n - это дерево, в котором каждый узел уровня n является листом и каждый узел уровня меньше n имеет непустые левое и правое поддеревья. На рисунке приведен пример полного бинарного дерева уровня 3.





Почти полное бинарное дерево - это бинарное дерево, для которого существует неотрицательное целое k такое, что

1. Каждый лист в дереве имеет уровень k или k+1.

2. Если узел дерева имеет правого потомка уровня k+1, тогда все его левые потомки, являющиеся листами, также имеют уровень k+1.

Строго бинарное дерево из рис. а не является почти полным, поскольку оно содержит листы уровней 1, 2 и 3, нарушая тем самым условие 1.

Строго бинарное дерево на рис. 6 удовлетворяет условию 1, так как каждый лист имеет уровень 2 или 3. Однако при этом нарушается условие 2, поскольку A имеет не только правого потомка уровня 3 (J), но также и левого потомка, являющегося листом уровня 2 (E).

Строго бинарное дерево на рис. в удовлетворяет обоим условиям и, следовательно, является почти полным бинарным деревом.

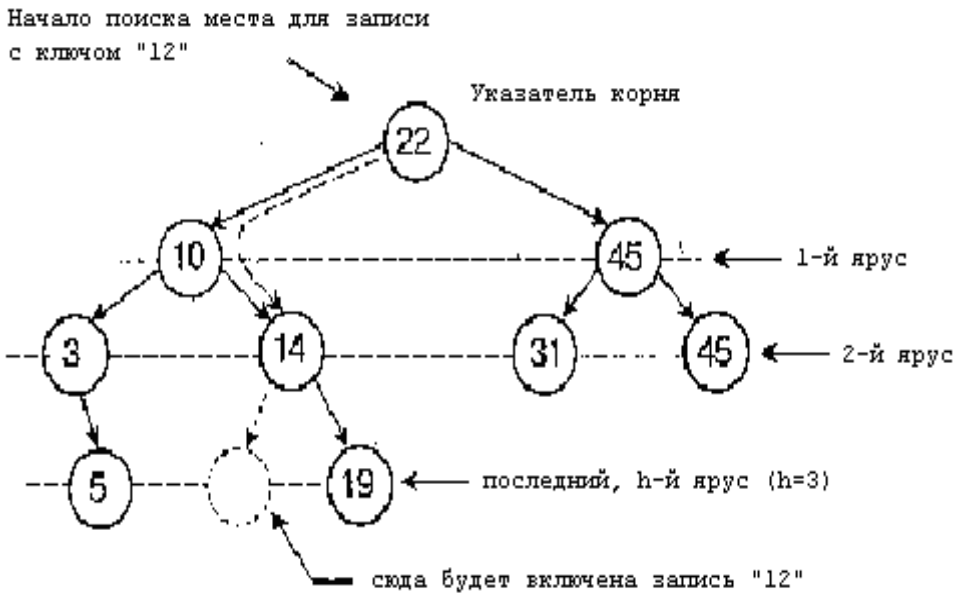
Бинарное дерево на рис. Г - также почти полное бинарное дерево, но оно не является строго бинарным, поскольку узел E имеет лишь левого сына.

Узлы почти полного бинарного дерева могут быть занумерованы так, что корню назначается номер 1, левому сыну - удвоенный номер его отца, а правому - удвоенный номер отца плюс единица (намного проще это понять, если представить эти числа в двоичной системе).

Есть еще одна разновидность бинарных деревьев, которая называется упорядоченные бинарные деревья.

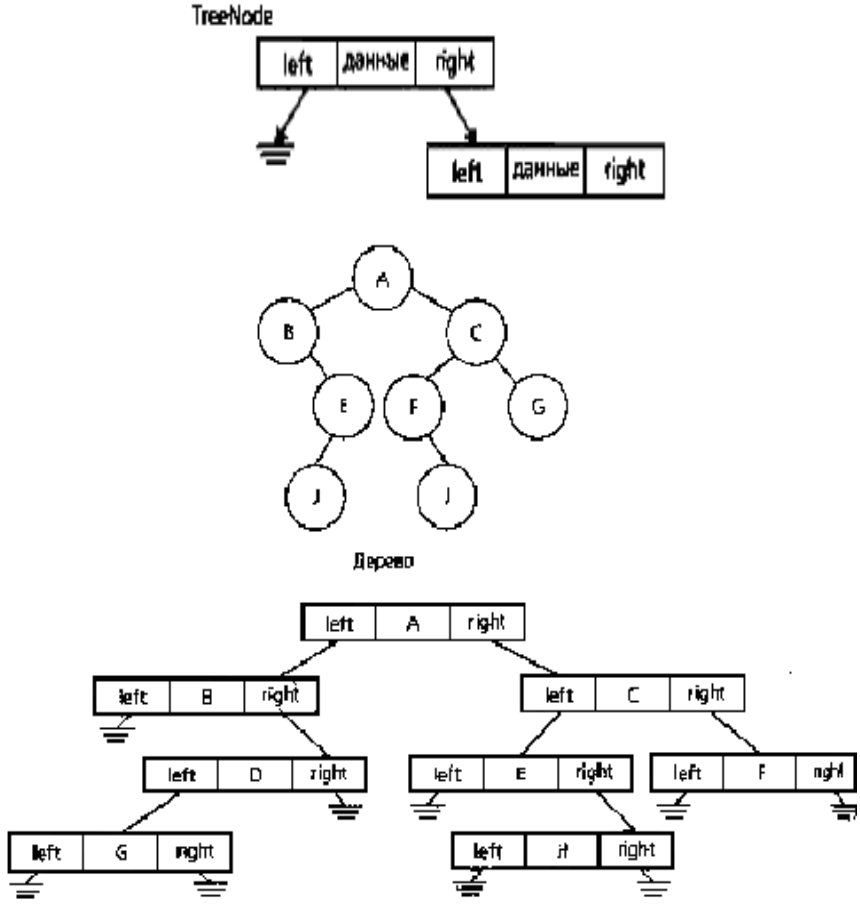
Упорядоченные бинарные деревья - это деревья, в которых для каждого узла X выполняется правило: в левом поддереве - ключи, меньшие X, в правом поддереве - большие или равные X.

Структура бинарного дерева построена из узлов. Узел дерева содержит поле данных и два поля с указателями.



Поля указателей называются левым указателем и правым указателем, поскольку они указывают на левое и правое поддерево, соответственно. Значение nil является признаком пустого дерева.

Тогда бинарное дерево можно будет представить в следующем виде.



Над бинарным деревом есть операция - его прохождение, т.е. нужно обойти все дерево, отметив каждый узел один раз.

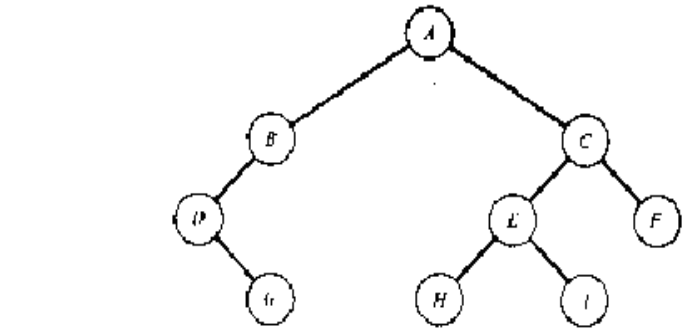
Существует 3 способа обхода бинарного дерева.

1. в прямом порядке
2. в симметричном порядке
3. в обратном порядке

В прямом порядке:
1. Попасть в корень
2. Пройти в прямом порядке левое поддерево
3. Пройти в прямом порядке правое поддерево

В симметричном порядке:
1. Пройти в симметричном порядке левое поддерево
2. Попасть в корень
3. Пройти в симметричном порядке правое поддерево

В обратном порядке:
1. Пройти в обратном порядке левое поддерево
2. Пройти в обратном порядке правое поддерево
3. Попасть в корень




```
Прямой порядок: ABDGCEHIF
Симметричный порядок: DGBAHEICF
Обратный порядок: CDBHIEFCA
```

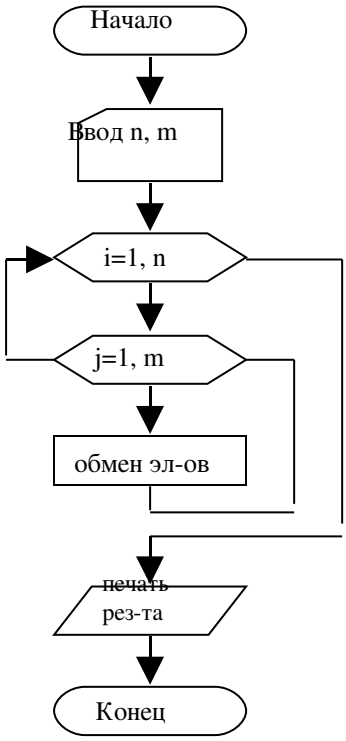
Варианты заданий.

Вариант	Задание
1	Напишите программы вычисления высоты дерева с использованием представлений деревьев.
2	Написать программу, выводящую списки узлов дерева, при обходе этого дерева в прямом порядке.
3	Напишите программы обхода двоичных деревьев в прямом порядке.
4	При прохождении дерева *в порядке уровней,* в список узлов сначала заносится корень дерева, затем все узлы глубины 1, далее все узлы глубины 2 и т.д. Узлы одной глубины заносятся в список узлов в порядке слева направо. Напишите программу обхода деревьев в порядке уровней.
5	Напишите программу поиска положительных элементов в бинарном дереве.
6	Напишите программу поиска элементов, кратных 2, в бинарном дереве.
7	Напишите программы обхода двоичных деревьев в обратном порядке.
8	Напишите программы обхода двоичных деревьев в внутреннем порядке.
9	Написать программу, выводящую списки узлов дерева, при обходе этого дерева в обратном порядке.
10	Написать программу, выводящую списки узлов дерева, при обходе этого дерева в внутреннем порядке.

Пример выполнения.

Задание: разработать программу, реализующую прямое прохождение (известное также как прохождение в глубину) бинарного дерева.

Блок-схема алгоритма



Текст программы:

```
uses
   crt;
const
    nvertex=100;
    nadj=1000;
var
  f:text;
  n,nt:integer;
  adj:array [1..nadj] of integer;
  fst:array [1..nvertex] of integer;
  nbr:array [1..nvertex] of integer;
  vtx:array [1..nvertex] of integer;
  mark:array [1..nvertex] of integer;
  t:array [1..nvertex] of integer;
  b:array [1..nvertex] of integer;

procedure init(var y:boolean);
var
  i,j,m:integer;
begin
    for i:=1 to n do
    for j:=1 to nbr[i] do
    begin
        y:=false;
        for m:=1 to n do if adj[fst[i]+j]=vtx[m] then
```

```
                    begin
                        y:=true;
                        adj[fst[i]+j]:=m;
                        break;
                        end;
        if not y then exit;
        end;
end;

procedure deep(x,u:integer;var c:integer);
var
  i,v:integer;
begin
    c:=c+1;
    mark[x]:=c;
    for i:=1 to nbr[x] do
    begin
        v:=adj[fst[x]+i];
        if mark[v]=0 then
        begin
            nt:=nt+2;
            t[nt-1]:=x;
            t[nt]:=v;
            b[nt div 2]:=1;
            deep(v,x,c);
            end
            else if (mark[v]<mark[x]) and (v<>u) then
                begin
                    nt:=nt+2;
                    t[nt-1]:=x;
                    t[nt]:=v;
                    b[nt div 2]:=0;
                    end;
    end;
end;

procedure way;
var
  v,c:integer;
begin
    nt:=0;
    c:=0;
    for v:=1 to n do mark[v]:=0;
    for v:=1 to n do if mark[v]=0 then deep(v,0,c);
end;

procedure main;
var
  i,j:integer;
  y:boolean;
```

```
begin
    writeln('Reading...');
    assign(f,'in.txt');
    reset(f);
    read(f,n);
    fst[1]:=0;
    for i:=1 to n do
    begin
        read(f,vtx[i]);
        read(f,nbr[i]);
        for j:=1 to nbr[i] do read(f,adj[fst[i]+j]);
        fst[i+1]:=fst[i]+nbr[i];
        end;
    close(f);
    writeln('Solve...');
    assign(f,'out.txt');
    rewrite(f);
    init(y);
    if not y then
    begin
        writeln(f,'BAD SEED! Hi from hell');
        writeln('BAD SEED! Hi from hell');
        close(f);
        exit;
        end;
    way;
    for i:=1 to nt div 2 do write(f,vtx[t[2*i-1]]:3);
    writeln(f);
    for i:=1 to nt div 2 do write(f,vtx[t[2*i]]:3);
    writeln(f);
    for i:=1 to nt div 2 do write(f,b[i]:3);
    writeln(f);
    close(f);
    writeln('End.');
    writeln('Press any key.');
end;
begin
    clrscr;
    main;
    readkey;
end.
```

Результаты расчетов

Структура входного файла IN.TXT:

```
19
1 3    2 3 4
2 3    1 5 6
3 2    1 7
4 4    1 8 9 10
5 2    2 11
```

6 1 2
7 3 3 12 13
8 1 4
9 3 4 14 15
10 1 4
11 3 5 16 17
12 1 7
13 1 7
14 1 9
15 3 9 18 19
16 1 11
17 1 11
18 1 15
19 1 15

Структура выходного файла OUT.TXT:

1 2 5 11 11 2 1 3 7 7 1 4 4 9 9 15 15 4
2 5 11 16 17 6 3 7 12 13 4 8 9 14 15 18 19 10
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Вывод: ознакомится со способами представления деревьев и методов их прохождения, получить практические навыки программирования задач с использованием деревьев.