ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ**

**БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**‹‹МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ›› (МГТУ ГА)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра Прикладной математики

Курсовая работа

Защищена с оценкой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись преподавателя, дата)

КУРСОВАЯ РАБОТА

по дисциплине ‹‹Программирование для ЭВМ››

Вариант № 4л

Тема: Разработка алгоритмов и программ реализации

нелинейной динамической структуры

Выполнил студент группы ПМ-2–1

Ерастов Иван Сергеевич л

(Ф.И.О.)

Руководитель:

Профессор кафедры ПМ Егорова А. А.

(звание, степень, Ф.И.О.)

Оглавление

[Аннотация 2](#_Toc92997530)

[Цель и задачи курсовой работы 4](#_Toc92997531)

[Задание на выполнение курсовой работы 5](#_Toc92997532)

[Вариант № 4 6](#_Toc92997533)

[1. Задача №1 7](#_Toc92997534)

[1.1 Таблицы подпрограмм и глобальных переменных 7](#_Toc92997535)

[1.2 Структура первой программы 8](#_Toc92997536)

[1.3 Алгоритмы функций и процедур 9](#_Toc92997537)

[1.4 Руководство пользователя 16](#_Toc92997538)

[1.5 Листинг задачи №1 17](#_Toc92997539)

[2. Задача №2 21](#_Toc92997540)

[2.1 Таблицы подпрограмм и глобальных переменных 21](#_Toc92997541)

[2.2 Структура программы 22](#_Toc92997542)

[2.3 Алгоритмы функций и процедур 23](#_Toc92997543)

[2.4 Руководство пользователя 26](#_Toc92997544)

[2.5 Листинг задачи №2 27](#_Toc92997545)

[3. Задача №3 30](#_Toc92997546)

[3.1 Таблицы подпрограмм и глобальных переменных 30](#_Toc92997547)

[3.2 Алгоритмы функций и процедур 31](#_Toc92997548)

[3.3 Структура программы 33](#_Toc92997549)

[3.4 Руководство пользователя 33](#_Toc92997550)

[3.5 Листинг программы №3 34](#_Toc92997551)

[4. Задача №4 35](#_Toc92997552)

[4.1 Таблицы подпрограмм и глобальных переменных 35](#_Toc92997553)

[4.2 Алгоритмы функций и процедур 36](#_Toc92997554)

[4.3 Листинг программы №3 38](#_Toc92997555)

[Список использованной литературы 40](#_Toc92997556)

[Приложение А. исходные данные и задача № 1 41](#_Toc92997557)

[Приложение B. исходные данные и задача № 2 43](#_Toc92997558)

[Приложение C. исходные данные и задача № 3 44](#_Toc92997559)

[Приложение D. исходные данные и задача № 4 45](#_Toc92997560)

# 

# Аннотация

В курсовой работе рассматривается решение практических задач на языке PASCAL. Задачи затрагивают разделы, посвященные работе со структурами данных (деревья, графы), алгоритмами, работе с модулями. Программы имеют пользовательский интерфейс, для выбора функции в программах используется текстовое меню, ввод данных осуществляется из файлов и с консоли.

Основная часть работы посвящена описанию переменных и функций программ. В приложениях приводится исходный код и результаты работы.

# Цель и задачи курсовой работы

Целью выполнения курсовой работы является освоение основных приемов работы с нелинейной динамической структурой типа «дерево» и «граф», алгоритмов поиска данных, а также освоения основ оформления программной документации. Приемы работы с нелинейной динамической структурой:

* создание нелинейной динамической структуры типа «дерево»;
* обход нелинейной динамической структурой типа «дерево»;
* удаление «дерева» и из «дерева»;
* поиск в структуре типа «дерево»;
* представление графа в виде массива.

Задачи, решаемые при выполнении курсовой работы:

* разработка структуры программы;
* разработка алгоритмов процедур;
* формирование модуля, содержащего необходимые процедуры обработки исходных данных;
* оформление программы и формирование пользовательского интерфейса;
* отладка программы;
* оформление пояснительной записки, включающей всю необходимую информацию для программиста.

Для решения поставленных задач необходимо разработать алгоритм и программу на языке программирования высокого уровня в соответствии со своим вариантом, проверить ее работоспособность и оформить сопроводительную документацию (пояснительную записку).

Разработанная программа должна быть хорошо структурирована, разработана с использованием модульной структуры, оформлена с комментариями, проясняющими работу программы, реализована с использованием языка высокого уровня. Пользовательский интерфейс должен быть дружественным.

# Задание на выполнение курсовой работы

1. Разработать алгоритмы и программу в соответствии со своим вариантом, реализующую структуру типа «дерево» на языке Паскаль (любая модификация), включающую создание и обработку.

Программа должна содержать процедуры:

* создания структуры типа «дерево»;
* проверки пустоты «дерева»;
* добавления элемента в «дерево»;
* чтения «дерева»;
* удаления из дерева (вне зависимости от варианта);
* обработки «дерева» в соответствии с вариантом.

В программе необходимо использовать динамическое выделение памяти для формирования древовидной структуры. Процедура чтения дерева должна осуществлять вывод дерева на экран или в файл в виде, отображающем его структуру.

2. Разработать алгоритм и программу поиска в соответствии со своим вариантом.

Программа должна содержать процедуры:

* поиска данных;
* печати результатов.

3. Разработать алгоритм и программу, использующую результаты двух первых программ (использовать модуль).

4. Разработать программу, выполняющую обработку графа. При этом:

* отображать вводимые исходные данные;
* при необходимости обеспечивать интерактивную обработку (диалог с пользователем), например, для выбора вершины;
* использовать треугольные и диагональные массивы, если матрицы симметричны.

Вариант № 4

1. Написать программу, которая создает бинарное дерево, состоящее из целых чисел, печатает все числа бинарного дерева, печатает числа бинарного дерева, используемые больше одного раза и количество их повторений. Определить количество чисел, используемых в дереве только один раз.

2. Дан текстовый файл, содержащий информацию о достопримечательностях городов, в составе: названия городов и их основные достопримечательности. Определить, в каком количестве городов встречаются заданные достопримечательности (например, Кремль, краеведческий музей и т. п.), и сколько раз встречаются они упоминаются в тексте. Для поиска использовать алгоритм Боуера и Мура.

3. Определить самый часто используемый символ (п.1) и определить, входит ли он в названия городов в п.2 и сколько раз.

4. Найти кратчайшие пути в орграфе от первой вершины ко всем остальным, используя алгоритм Дейкстры. Постройте дерево кратчайших путей.

# Задача №1

# Таблицы подпрограмм и глобальных переменных

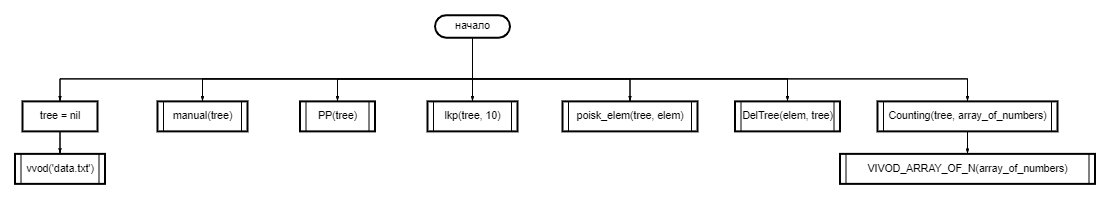
1. Таблица глобальных переменных

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Назначение |
| tree | Node | Указатель на корень дерева |
| fl | text | Файл для исходных данных |
| array\_of\_numbers | array\_of\_array\_of\_integer | Массив массивов для вывода данных |
| ST | string | Путь к файлу с исходными данными |

1. Таблица подпрограмм

|  |  |
| --- | --- |
| Описание | Назначение |
| procedureP(Z: Bder) | процедура для вывода одной записи |
| procedureFORM(**var** TR: Node; Z: Bder) | рекурсивная процедура, предназначенная для формирования нового узла с данными, расположенными в записи Z |
| procedure VVOD(NF: string) | Процедура, предназначенная для формирования дерева. Исходные данные вводятся из файла fl. Формирование узла для каждой записи производится посредством вызова процедуры FORM |
| procedure DelTree(x: Integer; Var s: Node); | Удаление заданного узла из бинарного дерева |
| procedure PrintDer(tr: Node) | Симметричный вывод дерева |
| procedure Counting(tr: Node; var arr: array of array of integer); | Процедура, которая считает числа бинарного дерева, используемые больше одного раза и количество их повторений, а также считает количество чисел, используемых в дереве только один раз. |
| procedure VIVOD\_ARR(const arr: array of array of integer); | Выводит результат процедуры Counting |

# Структура первой программы



# Алгоритмы функций и процедур

procedureFORM(**var** TR: Node; Z: Bder):

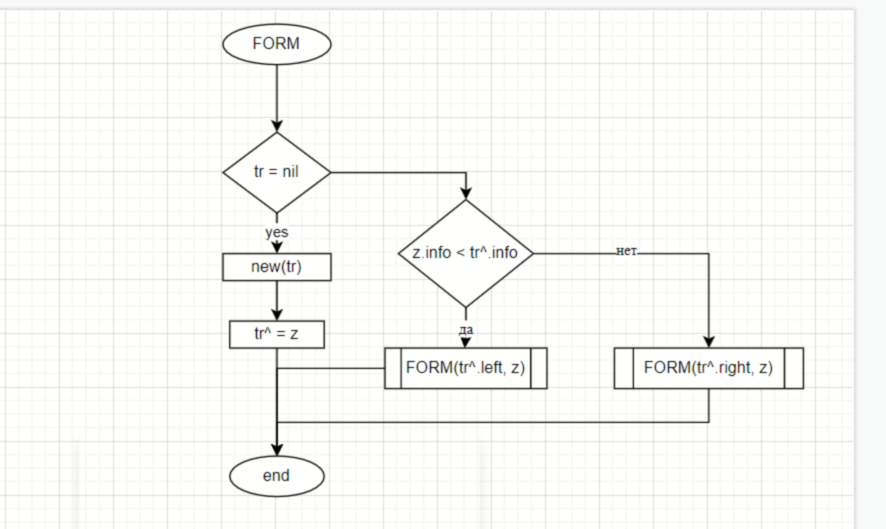
****

Рисунок 1 - формирования нового узла с данными

procedure VVOD(NF: string):

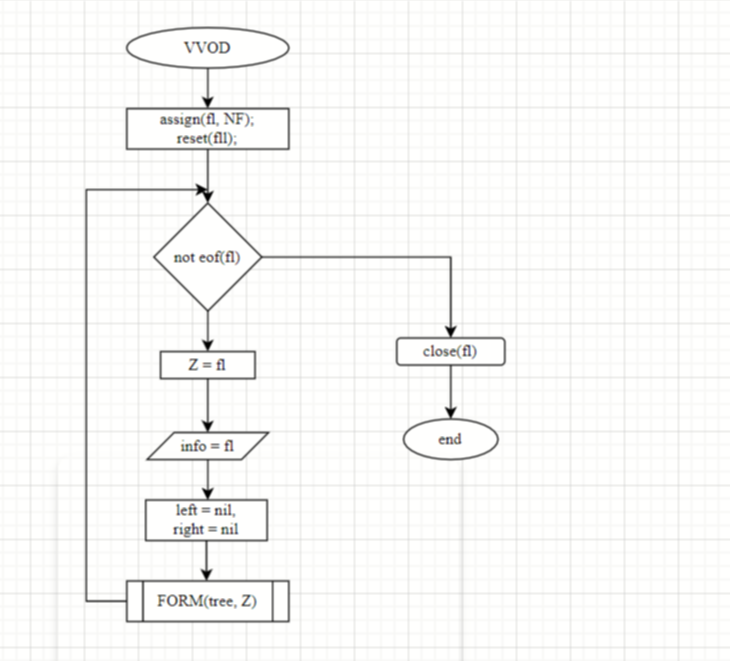
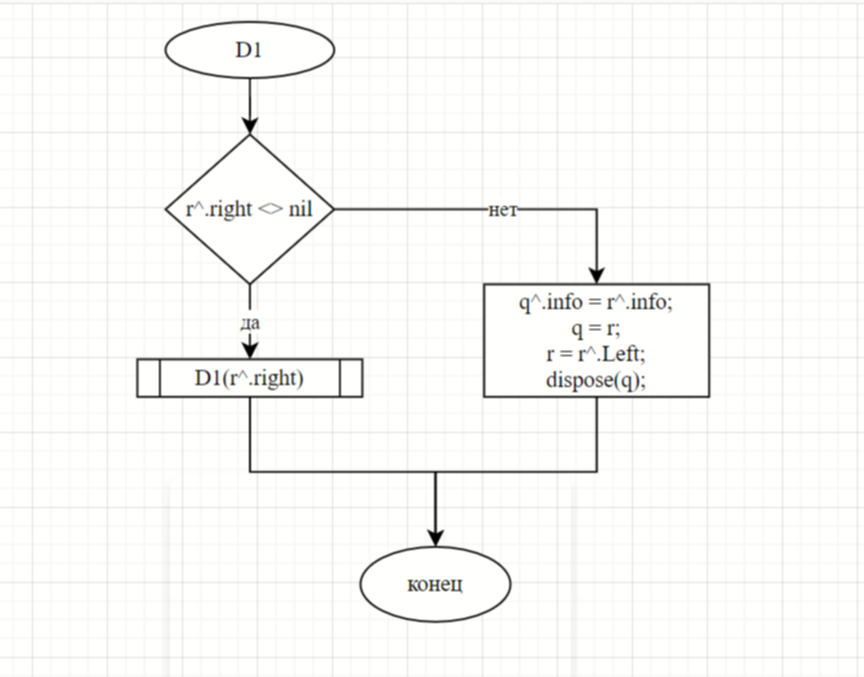


Рисунок 2 – ввод значений в дерево

procedure DelTree(x: Integer; Var s: Node):



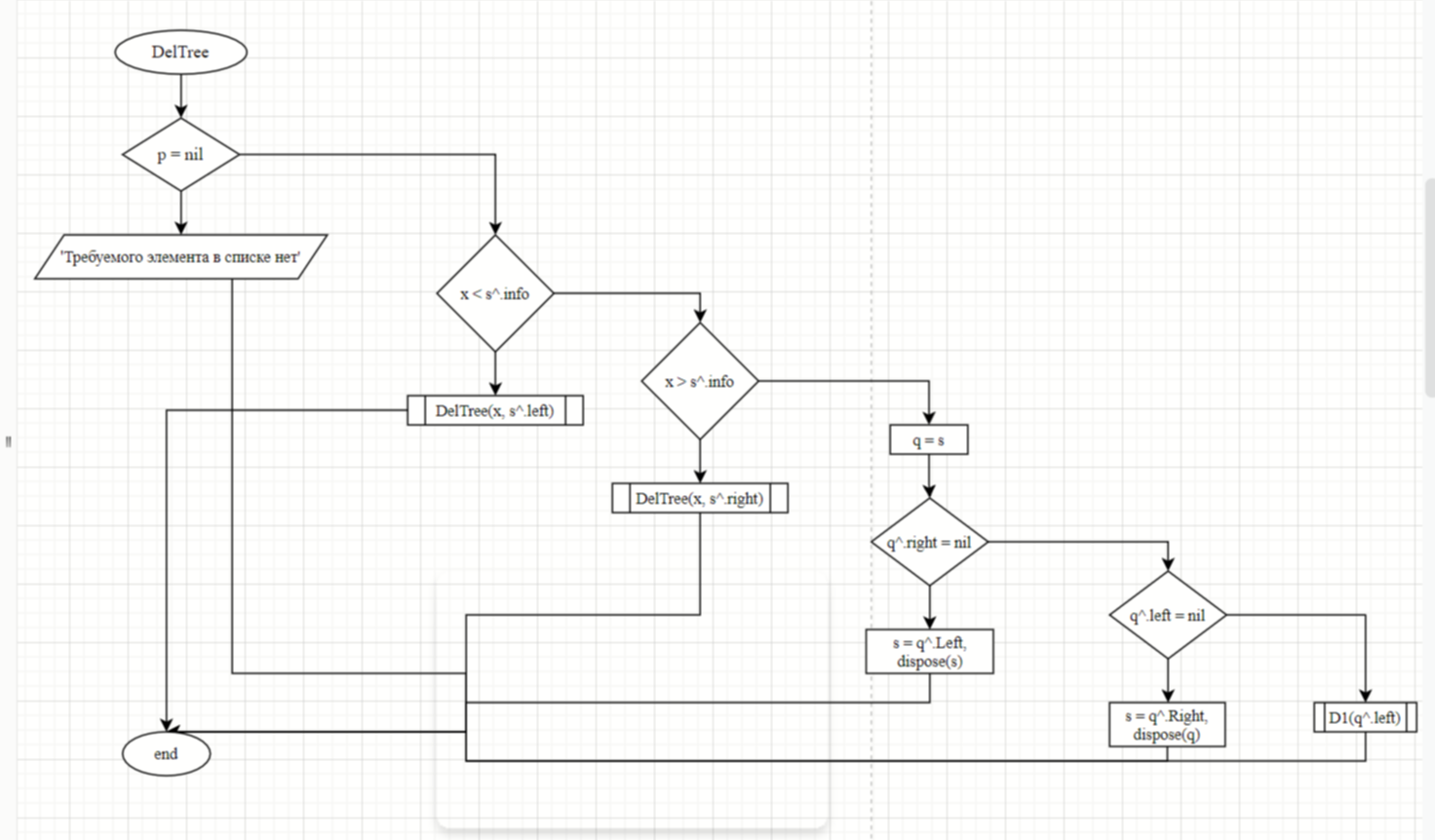


Рисунок 3 – удаление узла из дерева

procedure PrintDer(tr: Node):

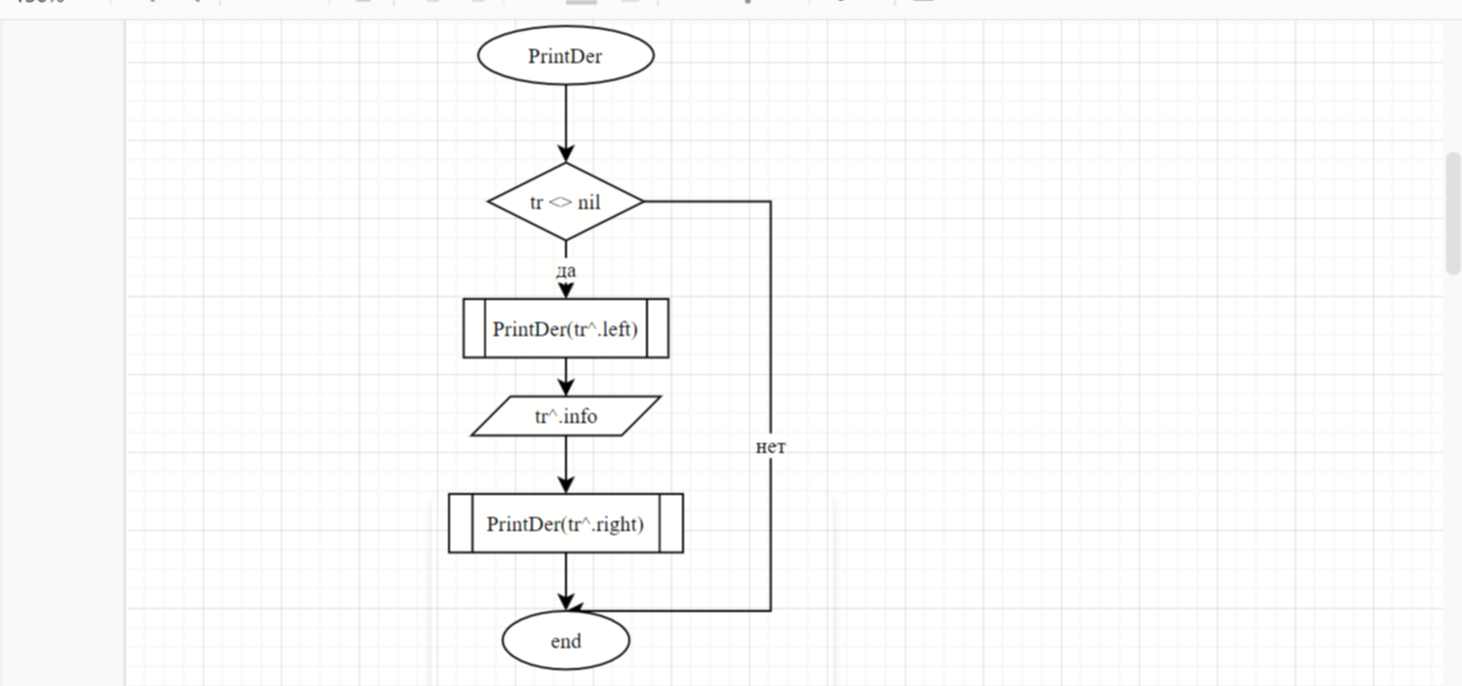


Рисунок 4 – вывод значений дерева

procedure Counting(tr: Node; var arr: array of array of integer):

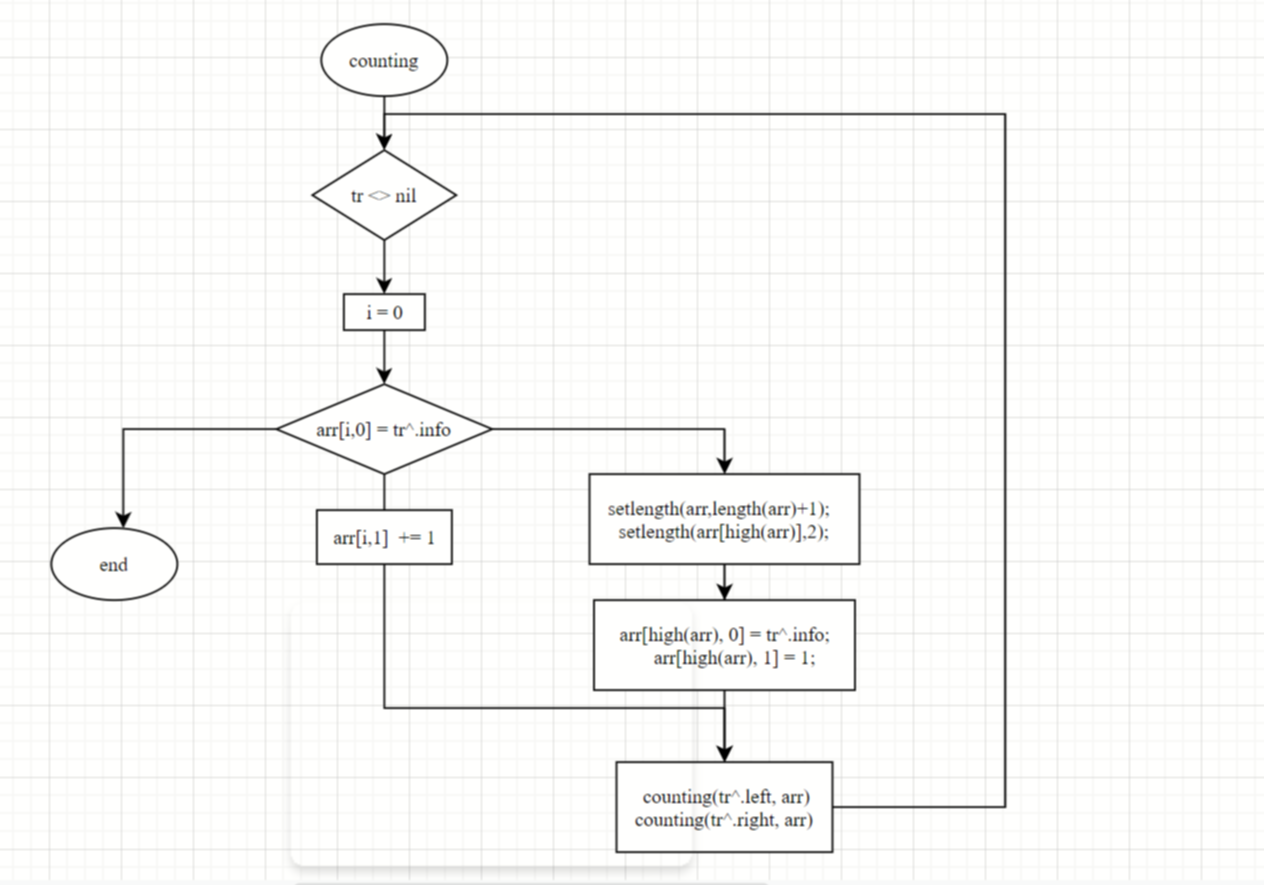


Рисунок 5 - считает числа бинарного дерева, используемые больше одного раза и количество их повторений, а также считает количество чисел, используемых в дереве только один раз

procedure VIVOD\_ARR(const arr: array of array of integer):

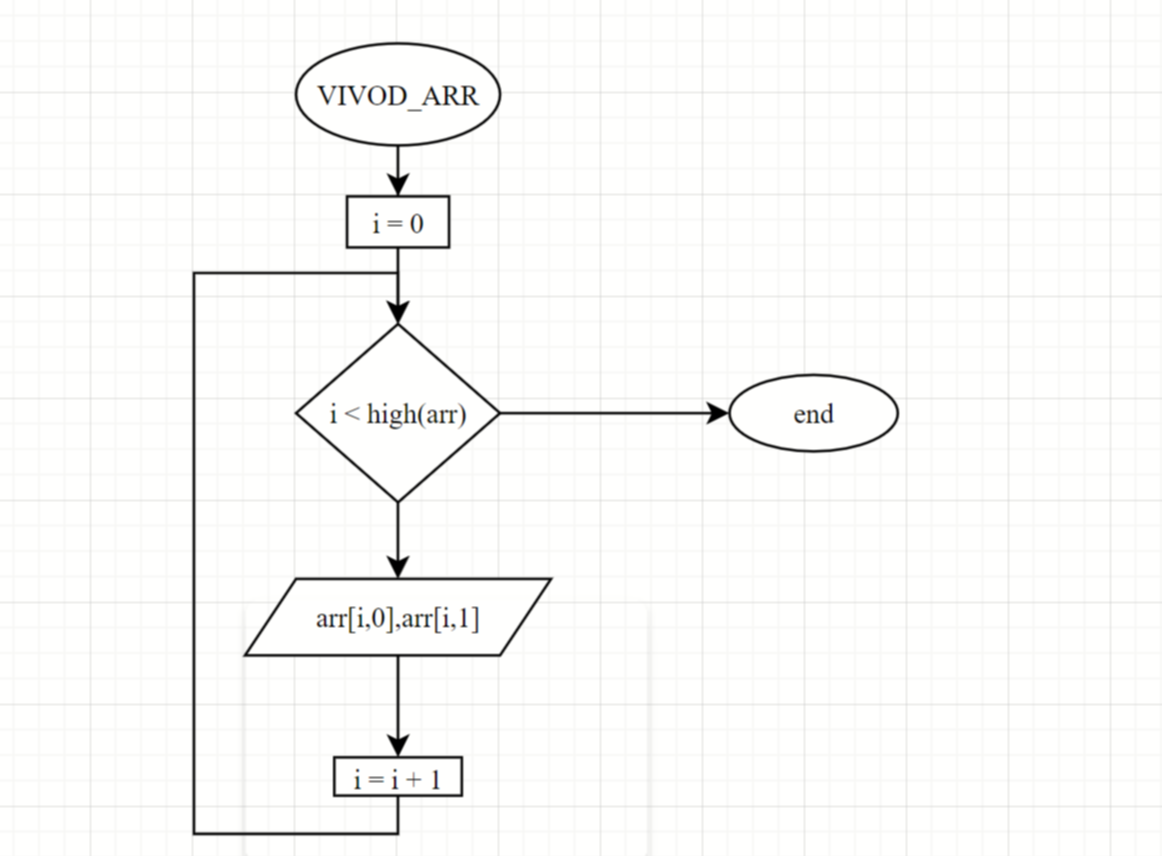


Рисунок 6 – выводит результат процедуры counting

Основная программа:

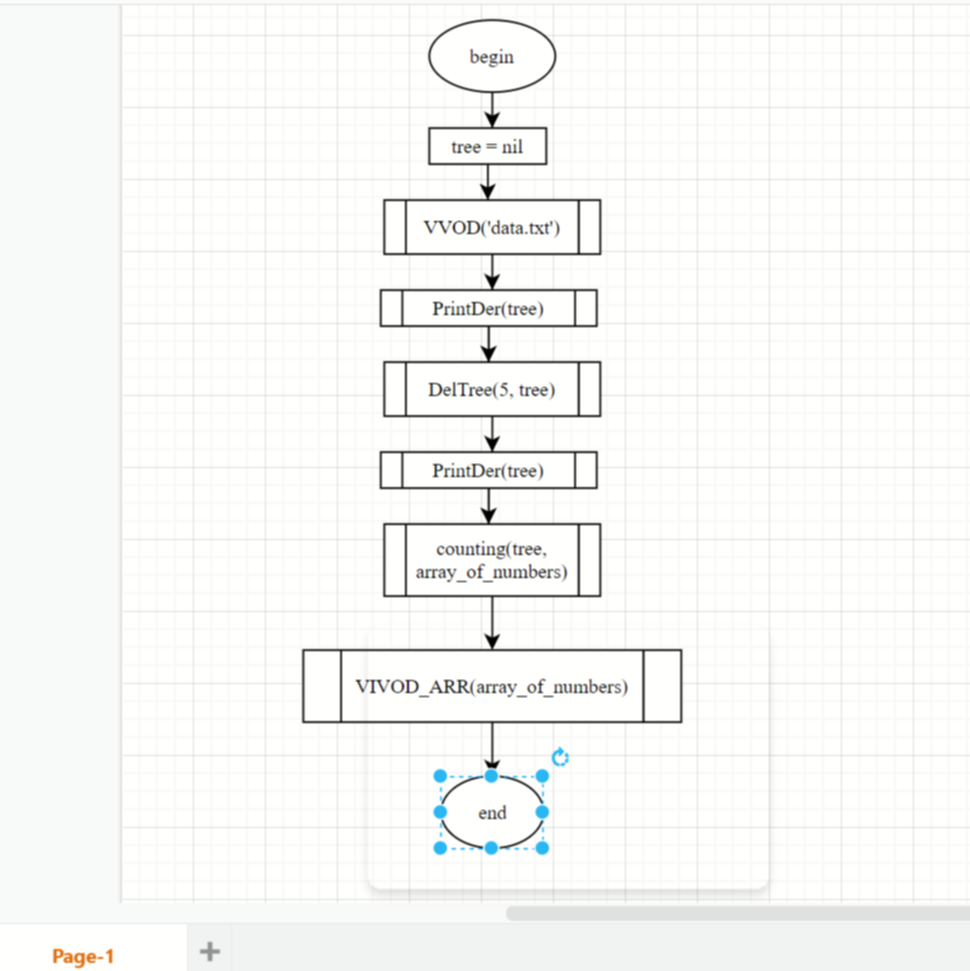


Рисунок 6 – основная программа

# Руководство пользователя

Для выбора функции программы используется текстовое меню.

Программа позволяет:

1. Добавить элементы в дерево из файла

2. Добавить элемент в дерево вручную

3. Произвести проверку пустоты дерева

4. Произвести печать в виде «Дерева»

5. Произвести поиск элемента

6. Произвести удаление узла дерева вручную

7. Посчитать сколько раз элементы дерева встречаются в дереве

0. Выйти из программы

# Листинг задачи №1

program kr;

type Node = ^Bder;

Bder = record

info: integer;

key: integer;

left, right: Node;

end;

var tree, prev: Node;

fl, fr: text;

array\_of\_numbers: array of array of integer;

ST: string;

{-----------------------------вывод записи--------------------------------------}

procedure P(Z: Bder);

begin

write(Z.info);

end;

{-----------------------------создание нового узла------------------------------}

procedure FORM(var TR: Node; Z: Bder);

begin

if tr = nil then begin //формирование новой записи

new(tr);

tr^:= z; //в данном узле

end

else if z.info < tr^.info then FORM(tr^.left, z) // в левой ветви

else FORM(tr^.right, z); // в правой

end;

{--------------------------------вывод всех для дерева--------------------}

procedure VVOD(NF: string);

var Z: Bder;

begin

assign(fl, NF); reset(fl);

writeln('---------Заполнение дерева новыми записями---------');

while not seekeof(fl) do with Z do begin

read(fl, info); //P(Z);//ввод данных и вывод

left:= nil; right:= nil;

FORM(tree, z); //вызов для формирования нового узла

end;

writeln; close(fl);

end;

{--------------------------------удаление из дерева----------------------------}

Procedure DelTree(x: Integer; Var s: Node);

Var q: Node;

{ процедура поиска подходящего узла и вставка его на место удаляемого }

Procedure D1(Var r: Node);

Begin

if r^.Right <> Nil Then D1(r^.Right)

else begin

q^.info:=r^.info;

q:=r;

r:=r^.Left;

dispose(q);

end;

end;

begin { поиск узла для удаления}

if p = Nil then Writeln('Требуемого элемента в списке нет')

else If x < s^.info Then DelTree (x, s^.Left)

else if x > s^.info Then DelTree (x, s^.Right)

else { узел найден }

begin { удаление узлов? имеющих одного сына}

q:= s;

if q^.Right = Nil then begin

s:=q^.Left;

dispose(s);

end

else

if q^.Left = Nil then begin

s:=q^.Right;

dispose(q);

end

else D1(q^.Left) { у узла два сына, удаление постепенно}

end;

end;

{-------------------------------проверка пустоты дерева------------------------}

function PP(s: Node): boolean;

begin

if s = nil then begin

writeln('Пусто');

PP:= true;

end

else PP:= False;

end;

{----------------------------вывод дерева--------------------------------------}

Procedure PrintDer(tr: Node);

begin

if tr <> Nil then begin

PrintDer(tr^.left); { <- переход к левому сыну при его наличии }

writeln('-->', tr^.info); { <- печать узла }

PrintDer(tr^.right); { <- переход к правому сыну при наличии }

end;

end;

Procedure Counting (tr: Node; var arr: array of array of integer);

var

i: integer;

label

next\_number;

begin

if tr <> Nil then begin

for i := 0 to high(arr) do

if arr[i,0] = tr^.info then begin

arr[i,1] += 1;

goto next\_number;

end;

setlength(arr, length(arr)+1);

setlength(arr[high(arr)], 2);

// Присваиваем значения элементам нового двухэлементного массива в массиве arr

arr[ high(arr), 0 ] := tr^.info;

arr[ high(arr), 1 ] := 1;

next\_number:

// Переход к сыновьям текущего узла tr:

Counting(tr^.left, arr); { <- переход к левому сыну при его наличии }

Counting(tr^.right, arr); { <- переход к правому сыну при наличии }

end;

end;

procedure VIVOD\_ARRAY\_OF\_N (const arr: array of array of integer);

var

i: integer;

begin

writeln('Число Кол-во повторений');

for i := 0 to high(arr) do

writeln(arr[i,0]:5,arr[i,1]:20);

end;

begin

PP(tree);

writeln('Создание дерева!');

tree:= nil;

VVOD('data.txt');

writeln('Вывод узлов дерева');

PrintDer(tree);

DelTree(5, tree);

writeln('Вывод узлов дерева');

PrintDer(tree);

Counting(tree, array\_of\_numbers);

VIVOD\_ARRAY\_OF\_N(array\_of\_numbers);

end.

# Задача №2

# Таблицы подпрограмм и глобальных переменных

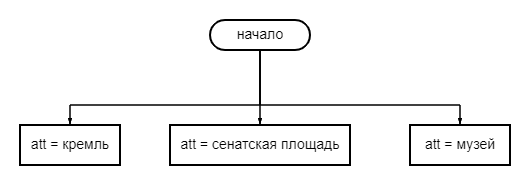
Таблица глобальных переменных

|  |  |  |
| --- | --- | --- |
| имя | тип | назначение |
| cities | city | Множество столиц |
| Simvol | char | Множество все русских букв |
| fl | text | Файл исходных данных |
| rat | string | Переменная хранит исходный текст |
| att | string | Переменная, которая хранит заданную достопримечательность |
| counter | string | Переменная, которая считает количество появлений переменной att |
| ans | byte | Переменная для взаимодействия с пользователем |

Таблица процедур

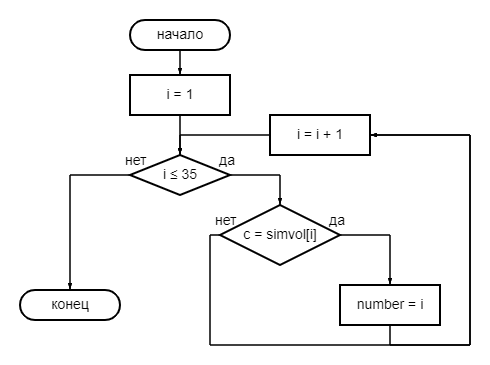
|  |  |
| --- | --- |
| описание | назначение |
| Function number(c:char) : byte; | Функция для формирования индекса в массиве. Если символ алфавита в подстроке не используется, то в  массив заносится длина подстроки. Если же символ встречается в подстроке, то  нужно вычислить расстояние от него до конца строки и занести это  расстояние на место того элемента в массиве. |
| function Poisk(S,P : ST): array of integer; | Функция для сравнения с концом подстроки, но от начала заданной строки |

# Структура программы

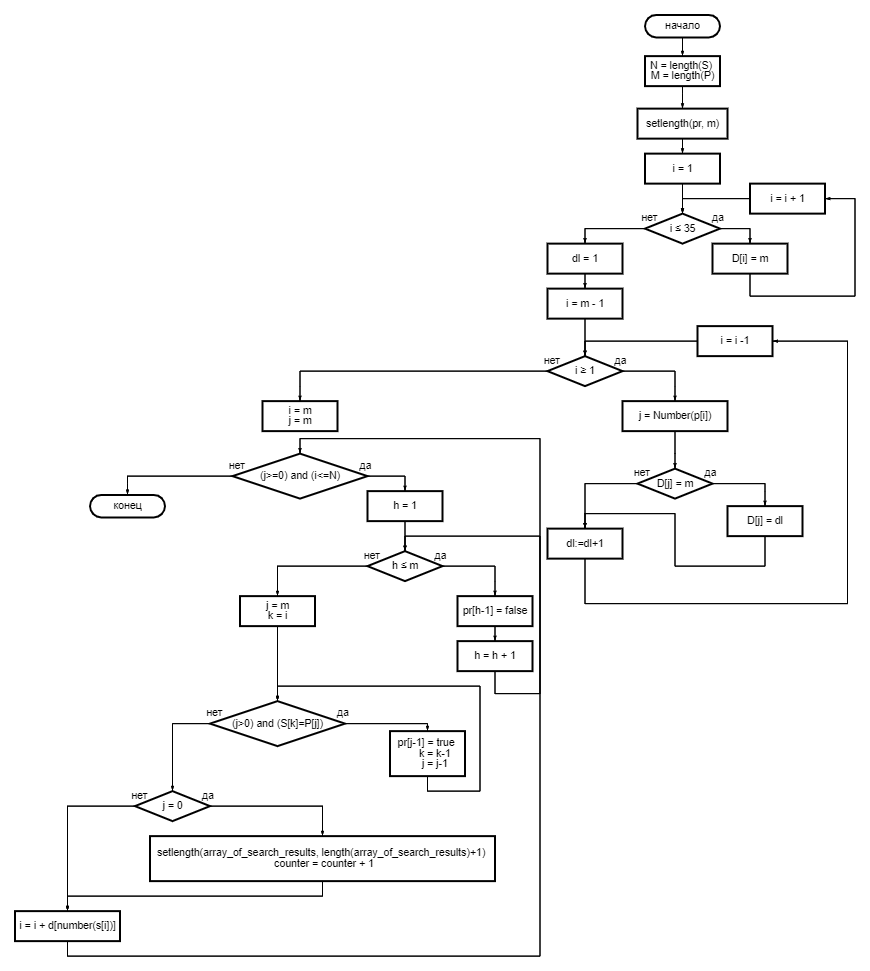


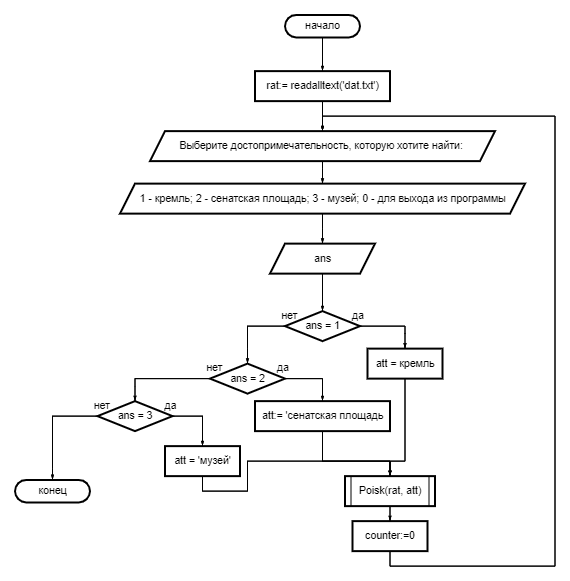
# Алгоритмы функций и процедур

Function number (c:char): byte



function Poisk(S,P : ST): array of integer



Основная программа:

# Руководство пользователя

Для выбора функции программы используется текстовое меню.

Программа позволяет:

1. Загрузить из файла текст
2. Выбор слова, по которому будет происходить поиск
3. Завершить работу

# Листинг задачи №2

**type**

ST = string;

city = **set of** string;

**var**

cities: city := ['москва', 'санкт-петербург', 'новосибирск'];

Simvol: **array**[1..35] **of** char = ('а', 'б', 'в', 'г', 'д', 'е', 'ж', 'з', 'и',

'й', 'к', 'л', 'м', 'н', 'о', 'п', 'р', 'с',

'т', 'у', 'ф', 'х', 'ц', 'ч', 'ш', 'щ', 'ъ',

'ы', 'ь', 'э', 'ю', 'я', ' ', '-', #13);

//array\_of\_search\_results: array of integer;

fl: text;

rat: string;

att: string;

counter: integer;

ans: byte;

**label** met1;

**Function** number (c:char) : byte;

**Var**

i: byte;

**Begin**

**for** i:=1 **to** 35 **do**

**if** c = Simvol[i] **then**

number:=i;

**End**;

**function** Poisk(S,P : ST): **array of** integer;

**Var**

i, j, k, dl, h, N, M: integer;

D : **array**[1..35] **of** byte;

pr : **array of** boolean;

array\_of\_search\_results: **array of** integer;

**Begin**

N:= length(S);

M:= length(P);

setlength(pr, m);

{--Заполняем матрицу расстояний--}

**for** i:=1 **to** 35 **do** D[i]:=m;

dl:=1;

**for** i:=m-1 **downto** 1 **do begin**

j:=Number(p[i]);

**if** D[j]=m **then** D[j]:=dl;

dl:=dl+1;

**end**;

{--Непосредственно поиск--}

i:=m;

j:=m;

**while** (j>=0) **and** (i<=N) **do begin**

**for** h:=1 **to** m **do** pr[h-1]:=false;

j:=m;

k:=i;

**while** (j>0) **and** (S[k]=P[j]) **do begin**

pr[j-1]:=true;

k:=k-1;

j:=j-1;

**end**;

**if** j = 0 **then begin**

setlength(array\_of\_search\_results, length(array\_of\_search\_results)+1);

//array\_of\_search\_results[high(array\_of\_search\_results)] := i-m+1;

inc(counter);

**end**;

i:=i+d[number(s[i])];

**end**;

Poisk := array\_of\_search\_results;

**End**;

**procedure** Convert(**var** s:string);

{ANSI -> ASCII}

**var** i:integer;

**begin**

**for** i:=1 **to** length(s) **do**

**if** ord(s[i]) **in** [192..239] **then** s[i]:=chr(ord(s[i])-64)

**else if** ord(s[i]) **in** [240..255] **then** s[i]:=chr(ord(s[i])-16)

**else if** ord(s[i])=168 **then** s[i]:=chr(ord(240))

**else if** ord(s[i])=184 **then** s[i]:=chr(ord(241));

**end**;

**begin**

rat:= readalltext('dat.txt');

writeln('Выберите достопримечательность, которую хотите найти: ');

met1:

writeln('1 - кремль; 2 - сенатская площадь; 3 - музей; 0 - для выхода из программы');

write('ваш ввод --> ');

read(ans);

**case** ans **of**

1: att:= 'кремль';

2: att:= 'сенатская площадь';

3: att:= 'музей';

0: **exit**;

**end**;

Poisk(rat, att);

//array\_of\_search\_results := Poisk(rat, 'кремль');

writeln('Количество раз, когда появилось слово ', '"',att, '"',': ', counter);

writeln;

//write(array\_of\_search\_results);

counter:=0;

**goto** met1;

**end**.

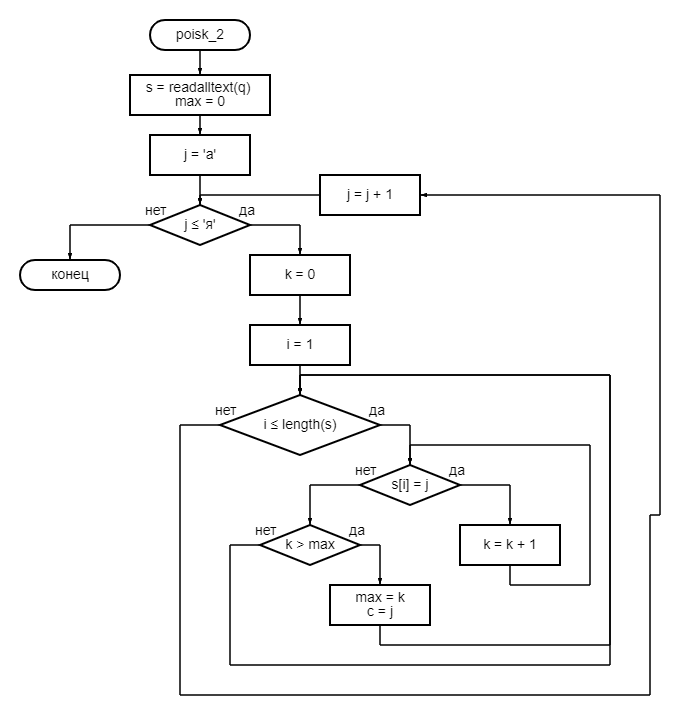
# Задача №3

# Таблицы подпрограмм и глобальных переменных

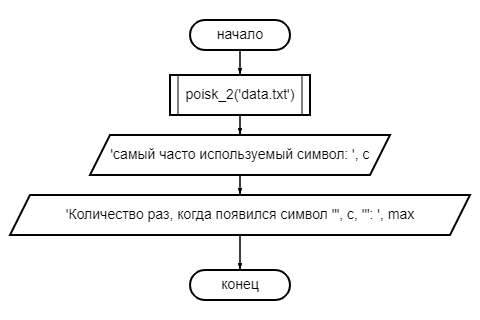
|  |  |  |
| --- | --- | --- |
| имя | тип | назначение |
| с | char | Это переменная для хранения символа |
| Max | integer | Переменная для хранения количества раз появления символа в тексте |
| k | integer | Промежуточная переменная для хранения значения символа в тексте |
| s | string | Переменная для хранения всего текста |
| q | string | Переменная для хранения ссылки на текст |

# Алгоритмы функций и процедур

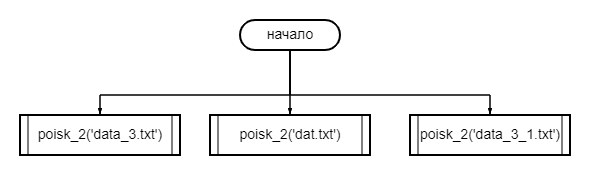
procedure poisk\_2(q: string):



Основная программа:



# Структура программы



# Руководство пользователя

Для выбора функции программы используется текстовое меню.

Программа позволяет:

1. Загрузить из файла текст на выбор для поиска символа
2. Завершить работу

# Листинг программы №3

**program** kr3;

**uses** modul;

**var** k, max: integer;

c: char;

s, q: string;

ans: byte;

**procedure** poisk\_2(q: string);

**var** j: char;

i: integer;

**begin**

s:= readalltext(q);

max:= 0;

**for** j:='а' **to** 'я' **do begin**

k:=0;

**for** i:=1 **to** length(s) **do**

**if** s[i] = j **then** k:= k + 1;

**if** k > max **then begin**

max:= k;

c:= j;

**end**;

**end**;

**end**;

**begin**

**repeat**

writeln('Выберите вариант для текста, для поиска наиболее частого символа');

writeln('1 - текст № 1, 2 - текст №2, 3 - текст №3');

write('--> '); read(ans);

**case** ans **of**

1: **begin** poisk\_2('data\_3.txt');

writeln('самый часто используемый символ: ', c);

writeln('Количество раз, когда появился символ "', c, '": ', max); **end**;

2: **begin** poisk\_2('dat.txt'); writeln('самый часто используемый символ: ', c);

writeln('Количество раз, когда появился символ "', c, '": ', max); **end**;

3: **begin** poisk\_2('data\_3\_1.txt'); writeln('самый часто используемый символ: ', c);

writeln('Количество раз, когда появился символ "', c, '": ', max); **end**;

**end**;

**until** ans = 0;

**end**.

# Задача №4

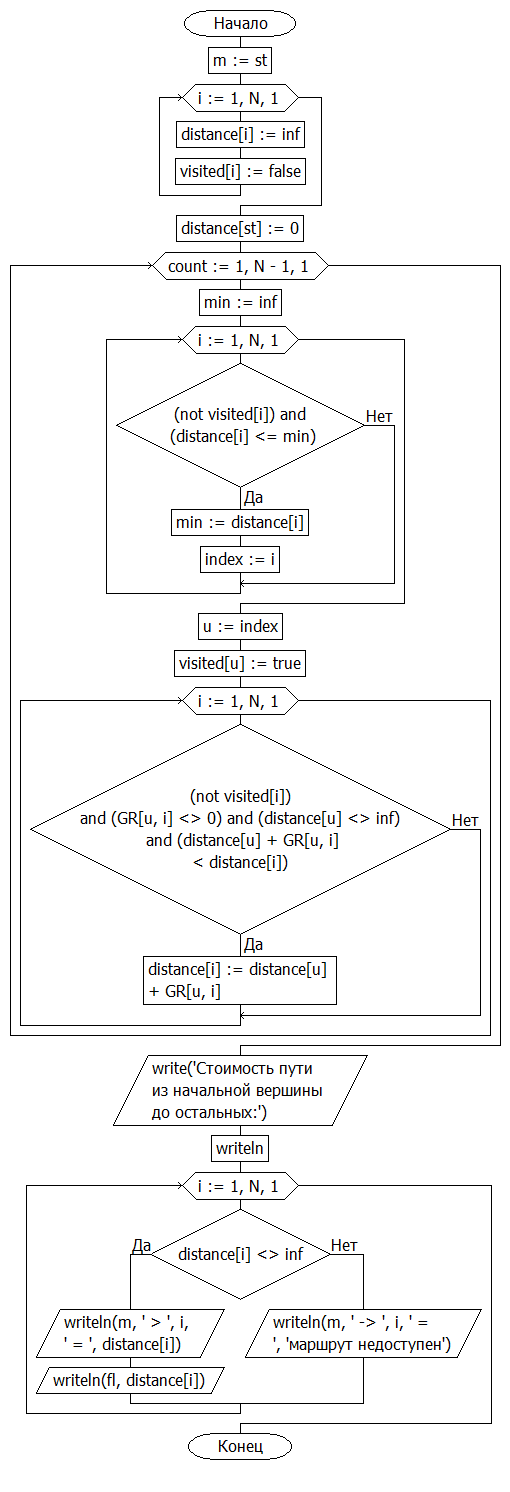
# Таблицы подпрограмм и глобальных переменных

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Назначение |
| GR | const | Переменная для хранения вершин графа |
| vector | type | Тип для хранения дистанции |
| tm | type | Тип для хранения матрицы смежности |
| start | start | Переменная для задания начальной вершины |

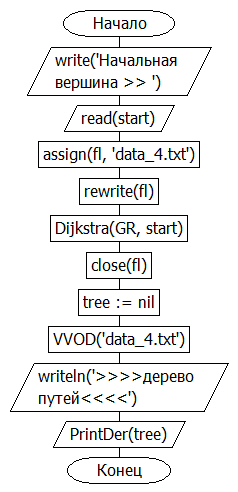
|  |  |
| --- | --- |
| описание | назначение |
| procedure Dijkstra(GR: tm; st: integer) | Процедура для поиска кратчайших путей между узлами графа |

# Алгоритмы функций и процедур

procedure Dijkstra(GR: tm; st: integer)



Основная программа:



# Листинг программы №3

**program** Dijkstraall;

**uses** modul;

**const** N = 6; inf = 100000;

**type** vektor = **array**[1..N] **of** integer;

tm = **array**[1..N, 1..N] **of** integer;

**var** start: integer;

// задается матрица начальных значений размера NxN.

**const** GR: tm = (

(0, 9, 1, 4, 0, 0), //1

(0, 0, 0, 0, 0, 0), //2

(1, 0, 0, 3, 2, 4), //3

(4, 0, 3, 0, 0, 1), //4

(0, 0, 2, 0, 0, 2), //5

(0, 0, 4, 1, 2, 0) //6

);

**procedure** Dijkstra(GR: tm; st: integer);

**var** count, index, i, u, m, min: integer;

distance: vektor;

visited: **array**[1..N] **of** boolean;

**begin**

m:=st;

**for** i:=1 **to** N **do begin** distance[i]:=inf; visited[i]:=false; **end**;

distance[st]:=0;

**for** count:=1 **to** N-1 **do begin**

min:=inf;

**for** i:=1 **to** N **do**

**if** (**not** visited[i]) **and** (distance[i]<=min) **then begin**

min:=distance[i]; index:=i; **end**;

u:=index;

visited[u]:=true;

**for** i:=1 **to** N **do if** (**not** visited[i]) **and** (GR[u, i]<>0) **and** (distance[u]<>inf) **and**

(distance[u]+GR[u, i]<distance[i]) **then** distance[i]:=distance[u]+GR[u, i];

**end**;

write('Стоимость пути из начальной вершины до остальных:'); writeln;

**for** i:=1 **to** N **do if** distance[i]<>inf **then begin** writeln(m,' > ', i,' = ', distance[i]); writeln(fl, distance[i]); **end**

**else** writeln(m,' -> ', i,' = ', 'маршрут недоступен');

**end**;

{основной блок программы}

**begin**

write('Начальная вершина >> '); read(start);

assign(fl, 'data\_4.txt'); rewrite(fl);

Dijkstra(GR, start);

close(fl);

tree:= nil;

VVOD('data\_4.txt');

writeln('>>>>дерево путей<<<<');

PrintDer(tree);

**end**.

# Список использованной литературы

1. Егорова А.А. Учебно-методическое пособие по выполнению курсовой работы «Программирование для ЭВМ» для студентов II курса направления 01.03.04 МГТУ ГА, 2019г. – 40с.[1]
2. Климова Л.М. Pascal 7.0. Практическое программирование. Решение типовых задач - М.: КУДИЦ-ОБРАЗ, 2003. [2].
3. Егорова А.А. Пособие по дисциплине «Структуры данных и методы обработки информации» для студентов IV курса специальности 073000 – М.: МГТУ ГА, 2011г.

# Приложение А. исходные данные и задача № 1

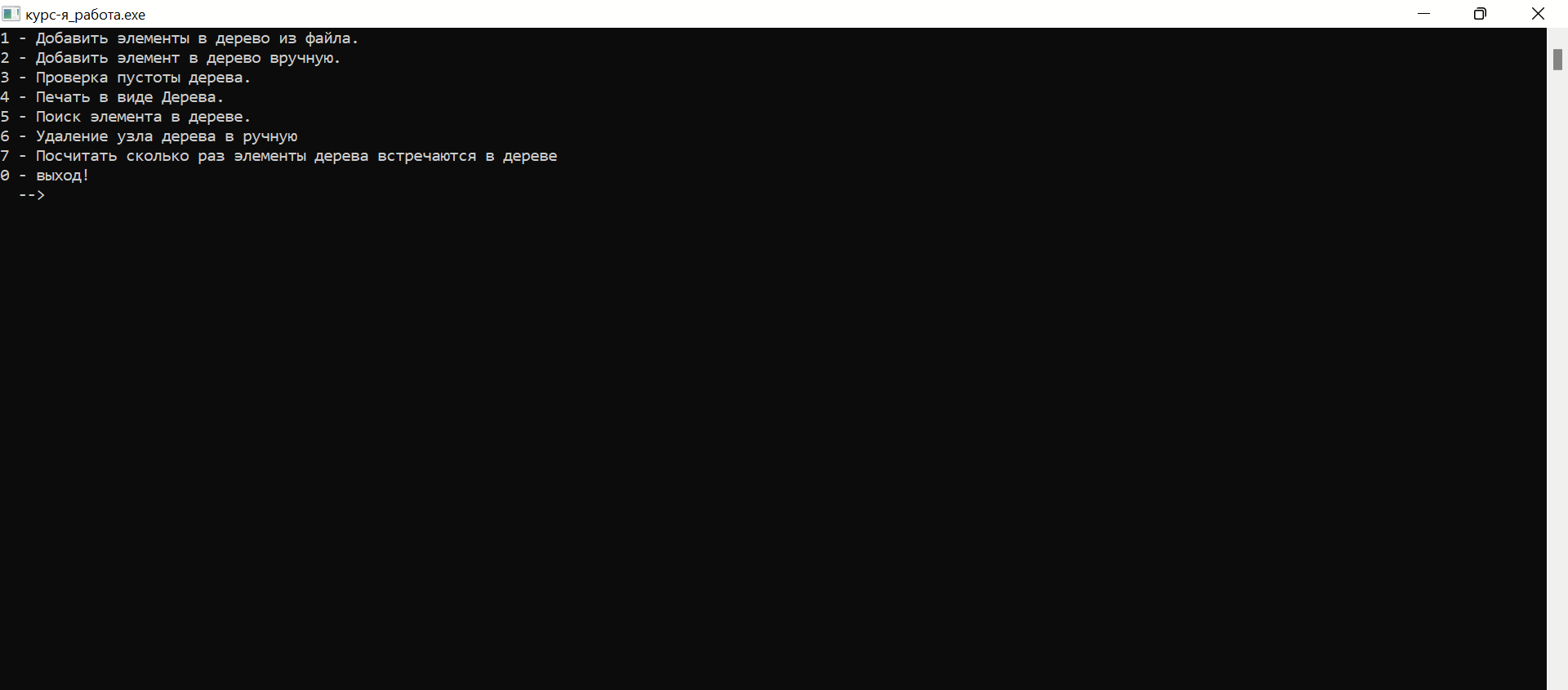


Рисунок А. 1 – Начало

Изображение выглядит как текст

Автоматически созданное описание

Рисунок А. 2 – ввод элементов дерева и вывод

Изображение выглядит как текст

Автоматически созданное описание

Рисунок А. 3 – Проверка пустоты и удаление узла

Изображение выглядит как текст

Автоматически созданное описание

Рисунок А. 4 – Удаление из дерева и выход

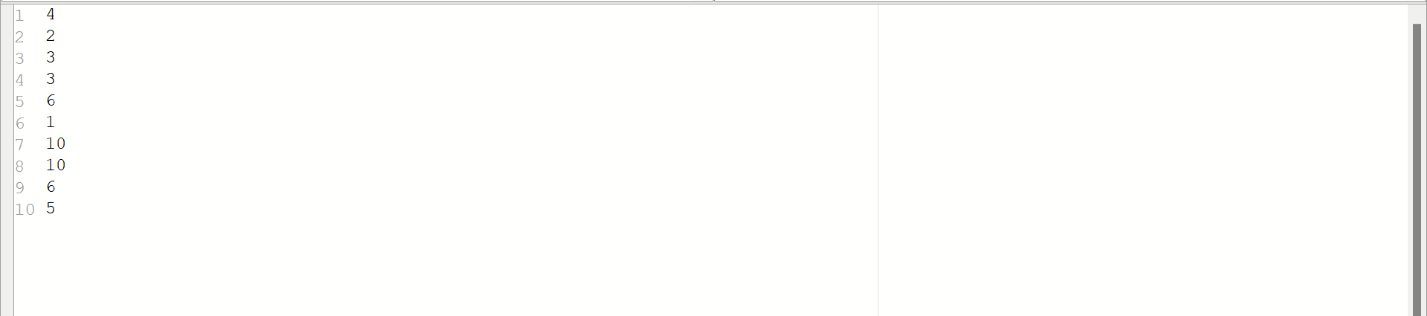


Рисунок А. 5 – Исходные данные файла

# Приложение B. исходные данные и задача № 2

Изображение выглядит как текст

Автоматически созданное описание

Рисунок В.1 – выбор всех вариантов и результат

Изображение выглядит как текст

Автоматически созданное описание

Рисунок В.2 – исходнные данные

# Приложение C. исходные данные и задача № 3

Изображение выглядит как текст

Автоматически созданное описание

Рисунок C. 1 – выбор всех вариантов и результаты

Изображение выглядит как текст

Автоматически созданное описание

Рисунок С.2 – исходные данные файла 1

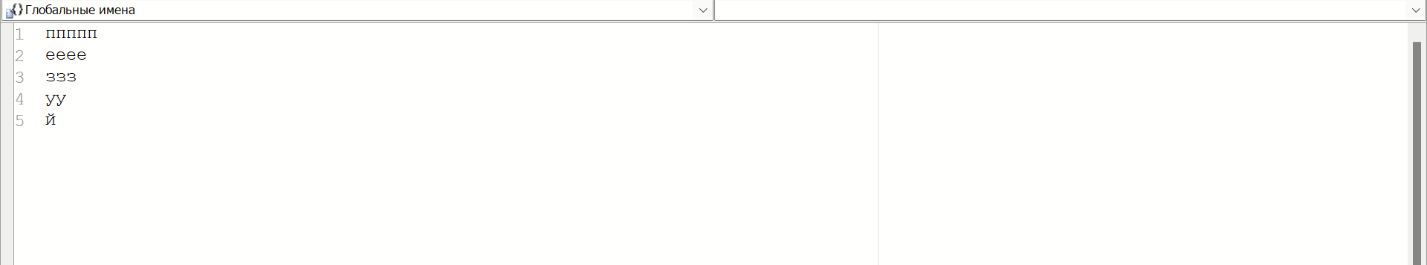


Рисунок С.3 – исходные данные файла 2

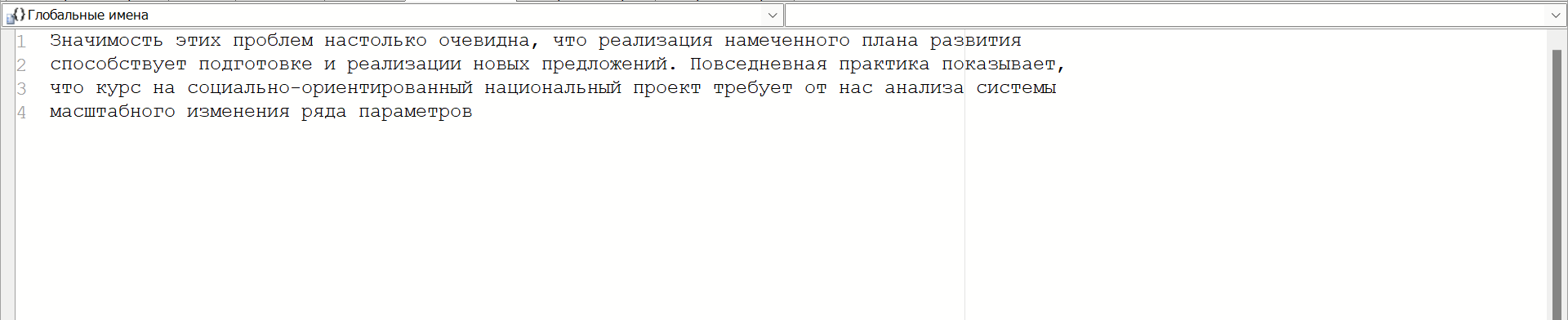


Рисунок С.4 – исходные данные файла 3

# Приложение D. исходные данные и задача № 4

Изображение выглядит как текст

Автоматически созданное описание

Рисунок D.1 – выбор всех вариантов и результаты

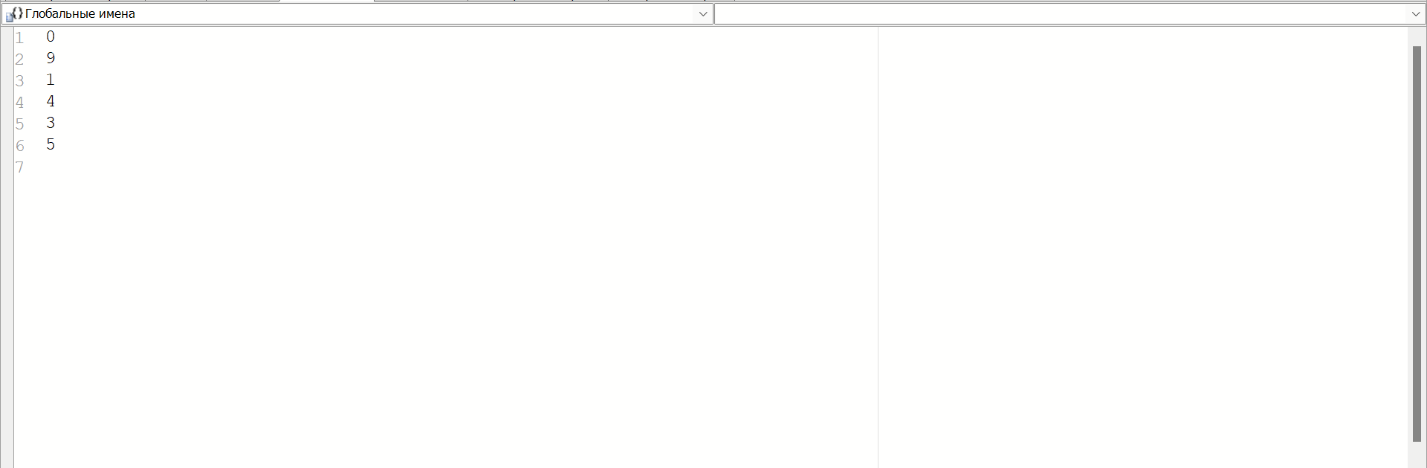


Рисунок D.2 – исходные данные файла 1