Московский технический университет связи и информатики

Кафедра «Информатика»

Лабораторная работа по дисциплине «Теория и методы программирования» № 4

Тема: «*РАЗРАБОТКА И ИССЛЕДОВАНИЕ АЛГОРИТМОВ НА ДЕРЕВЬЯХ НА ОСНОВЕ ДИНАМИЧЕСКИХ СТРУКТУР ДАННЫХ»*

Выполнил:

Блёсткин Михаил

Группа БИБ2203

Вариант 4

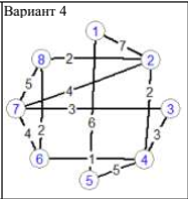
Москва 2023 г.

**Цель:**

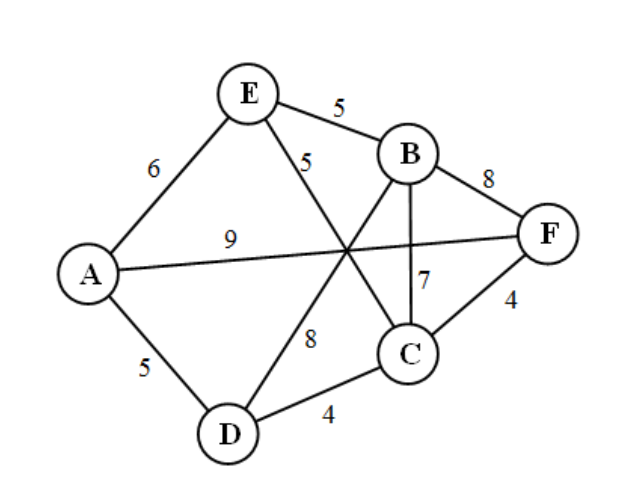
Изучить понятия, формирования, особенности доступа к данным и работы с памятью в деревьях, разрабатывать программы с использованием алгоритмов рекурсии, алгоритмов обхода деревьев и алгоритмов нахождения кратчайшего пути.

**Постановка задачи:**

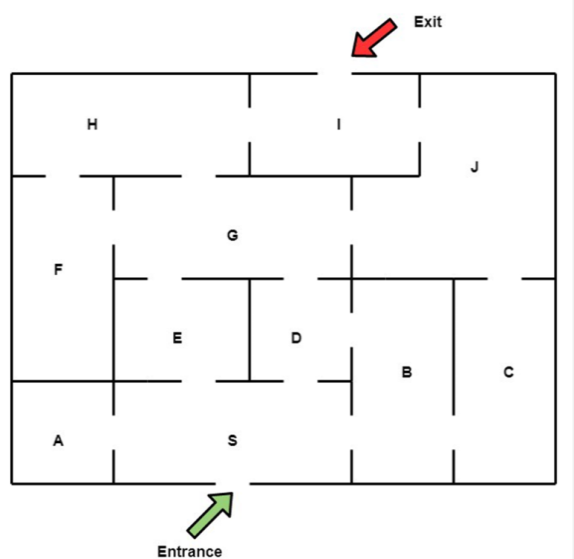
1. Написать программу поиска кратчайшие расстояния от 1-й вершины до всех остальных для графа согласно своего варианта (см. Рис. 11), используя алгоритмы *Дейкстры* и *Флойда*. Составьте таблицу сравнения двух алгоритмов по времени или количеству шагов.



1. Оля (A), Маша (B), Витя (C), Дима (D), Ваня (E) и Катя (F) живут в разных городах. Стоимость билетов из разных городов известна (рис. 10). Добраться до городов можно разными способами. Определить наименьшую сумму, которую нужно потратить, чтобы Оля могла навестить каждого из своих друзей.



1. Квадратное озеро задается матрицей MxN и покрыто мелкими островками. В левом верхнем углу находится плот размером *mxm*. За один шаг плот может передвигаться на одну клетку по вертикали или горизонтали. Требуется определить *кратчайший путь* плота до правого нижнего угла.
2. Составьте алгоритм и напишите программу, находящую строку длиной 100 символов, состоящую только из букв "A", "B", "C", такую, что в ней никакие две соседние подстроки не равны друг другу. Воспользуйтесь перебором с возвратом.
3. Напишите программу, которая находит путь (кратчайший, если есть), в сгенерированном с помощью датчика случайных чисел лабиринте. Примерный лабиринт изображен на рис. 12. Для нахождения пути используйте алгоритм *волновой алгоритм или алгоритм с возвратом*.



Пример лабиринта

1. Работу оформить в виде отчета. Отчет по работе должен соответствовать следующей структуре.

Титульный лист.

Постановка задачи.

Алгоритм решения задачи.

Листинг программы.

Контрольный тест. Результаты работы программы.

Выводы по работе.

**Алгоритм решения задачи:**

**Алгоритм Дейкстры**

Данный *алгоритм* является алгоритмом на графах, который изобретен нидерландским ученым Э. Дейкстрой в 1959 году. *Алгоритм* находит кратчайшее *расстояние* от одной из *вершин графа* до всех остальных и работает только для графов без ребер отрицательного веса.

Каждой вершине приписывается *вес* – это *вес* пути от начальной вершины до данной. Также каждая *вершина* может быть выделена. Если *вершина* выделена, то *путь* от нее до начальной вершины кратчайший, если нет – то временный. *Обходя граф*, *алгоритм* считает для каждой вершины *маршрут*, и, если он оказывается кратчайшим, выделяет вершину. Весом данной вершины становится *вес* пути. Для всех соседей данной вершины *алгоритм* также рассчитывает *вес*, при этом ни при каких условиях не выделяя их. *Алгоритм* заканчивает свою работу, дойдя до конечной вершины, и весом *кратчайшего пути* становится *вес* конечной вершины.

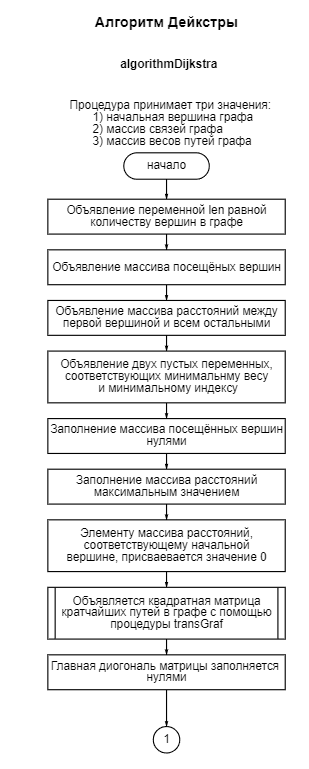
### Алгоритм Флойда

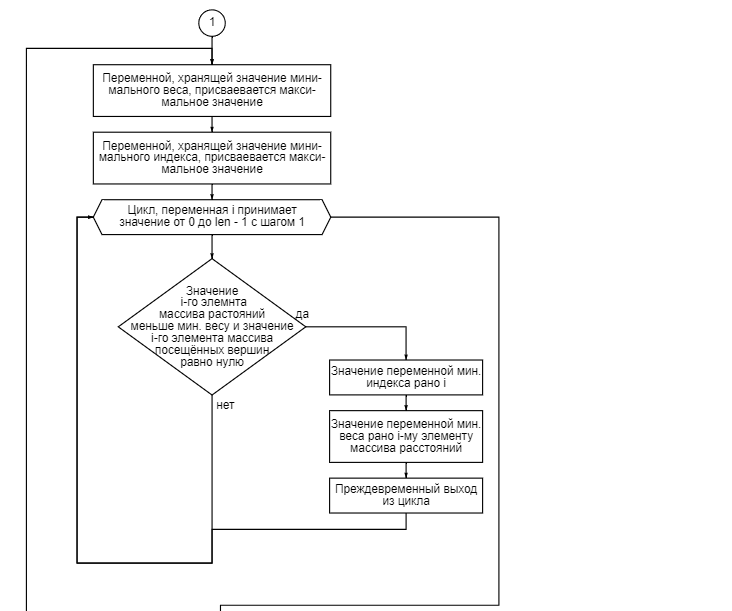
Рассматриваемый *алгоритм* иногда называют *алгоритмом Флойда*-Уоршелла. *Алгоритм Флойда*-Уоршелла является алгоритмом на графах, который разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом. Он служит для нахождения *кратчайших путей* между всеми парами *вершин графа*.

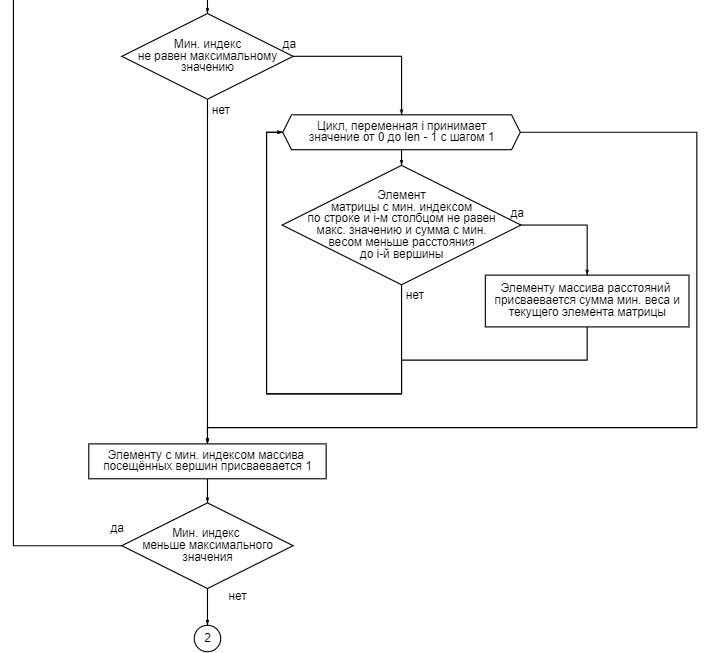
*Метод Флойда* непосредственно основывается на том факте, что в графе с положительными весами ребер всякий неэлементарный (содержащий более 1 *ребра*) *кратчайший путь* состоит из других *кратчайших путей*.

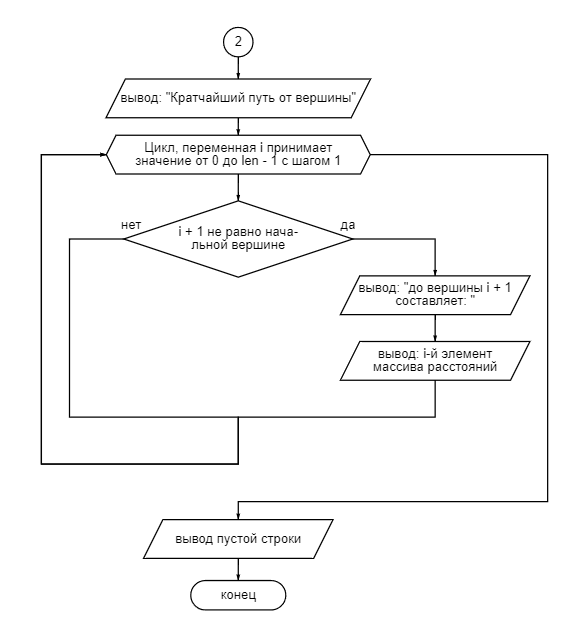
Этот *алгоритм* более общий по сравнению с алгоритмом Дейкстры, так как он находит кратчайшие пути между любыми двумя *вершинами графа*.

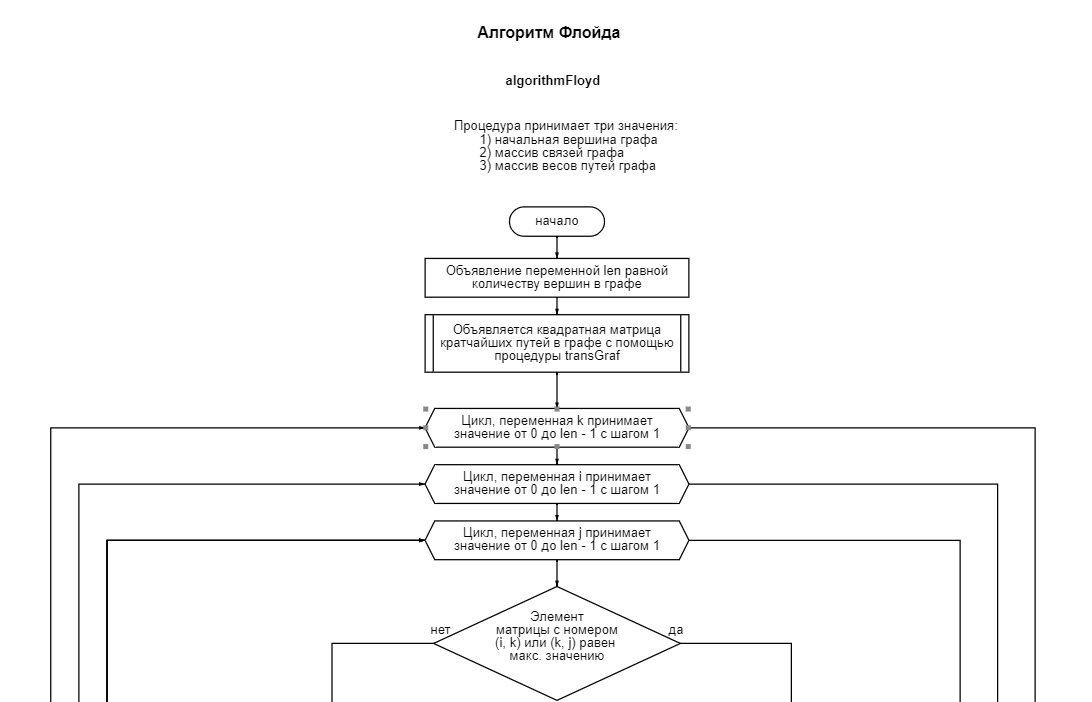
В алгоритме Флойда используется *матрица* A размером nxn, в которой вычисляются длины *кратчайших путей*. Элемент A[i,j] равен расстоянию от вершины i к вершине j, которое имеет конечное *значение*, если существует *ребро* (i,j), и равен бесконечности в противном случае.

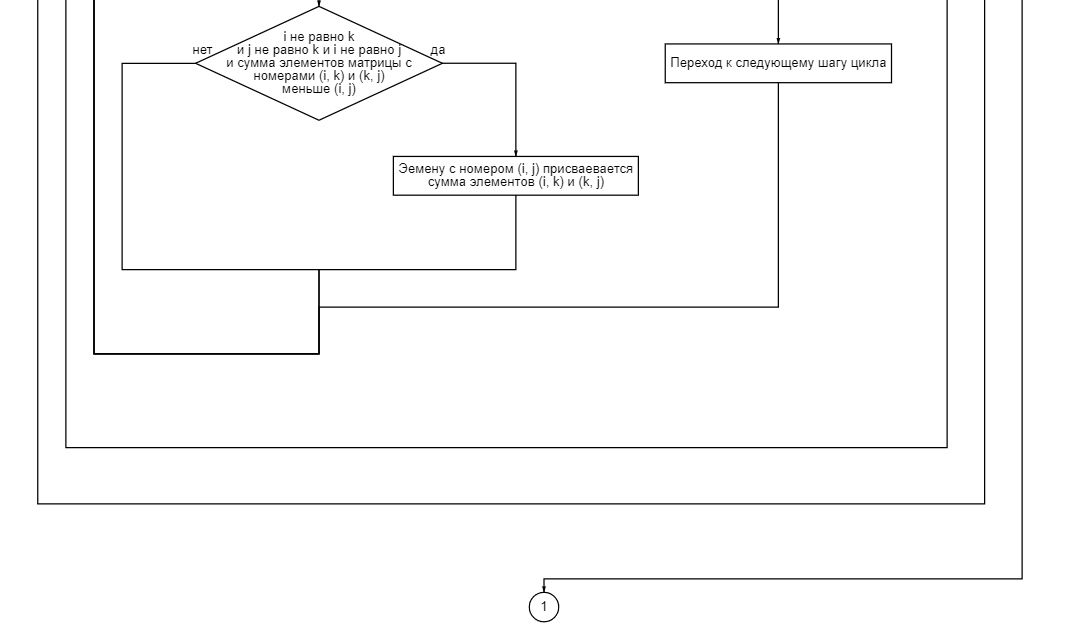
**Блок схемы алгоритмов:**

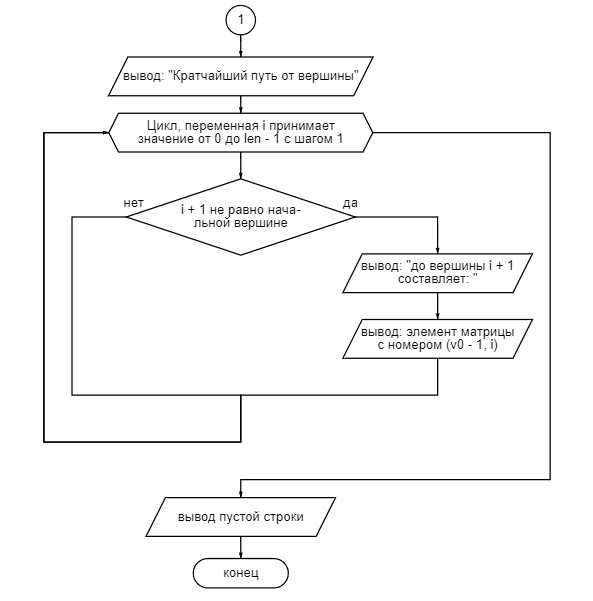


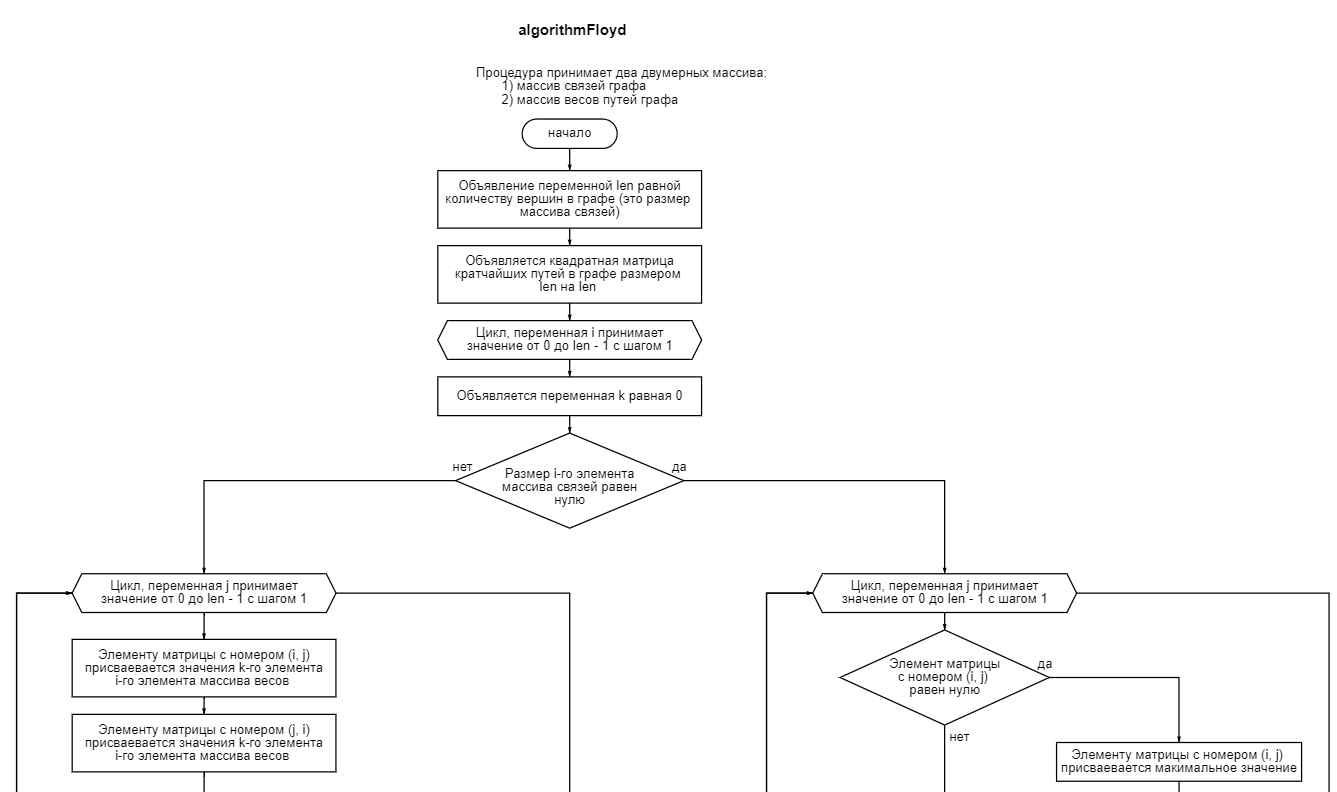
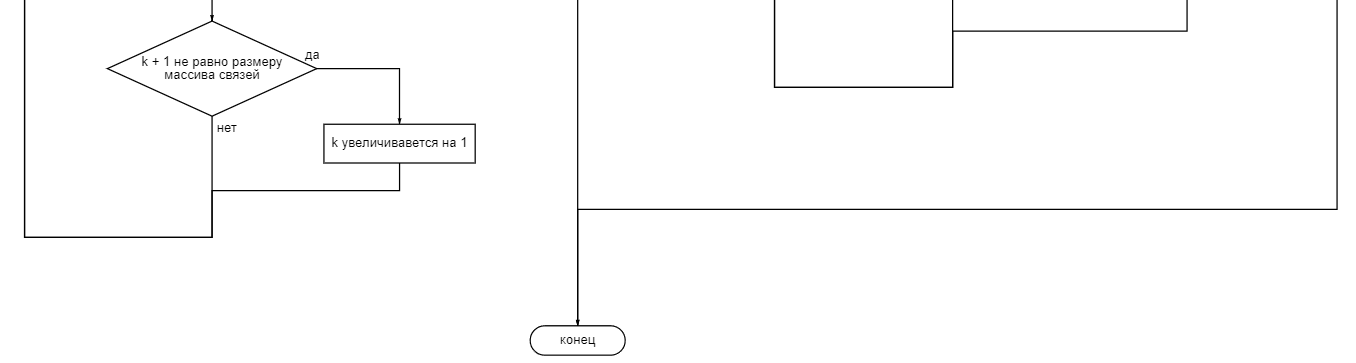


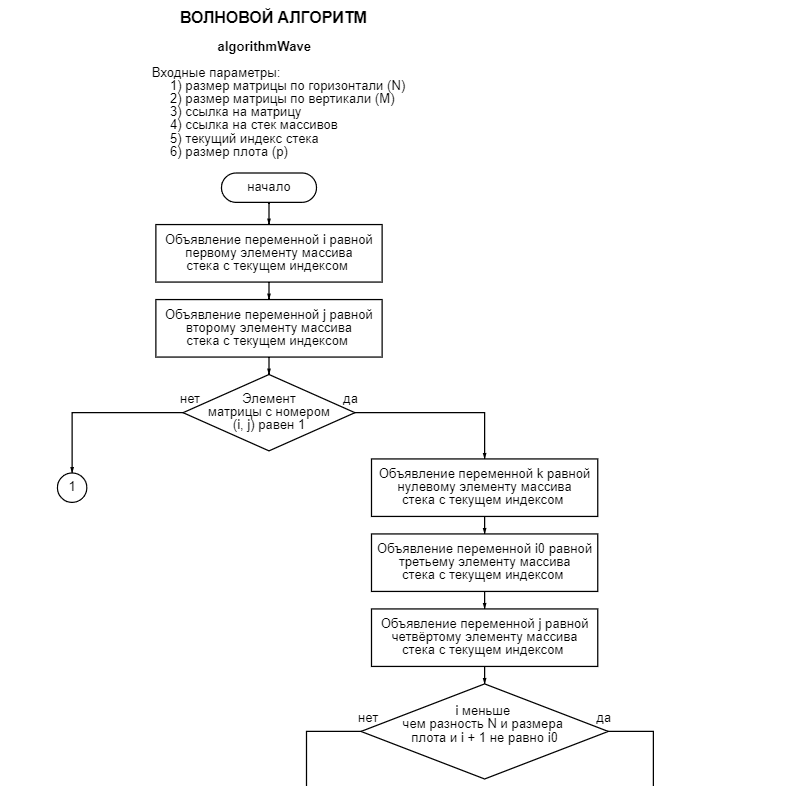
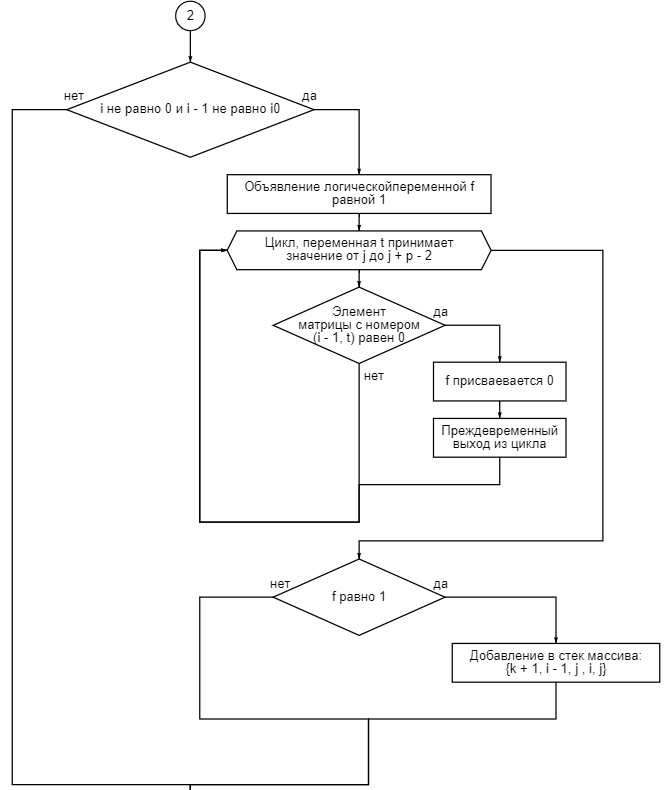


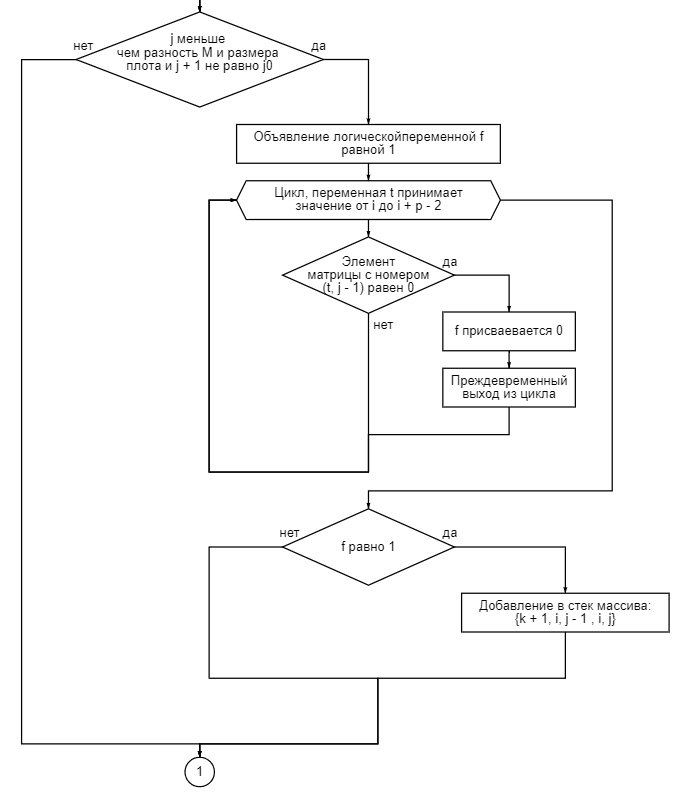


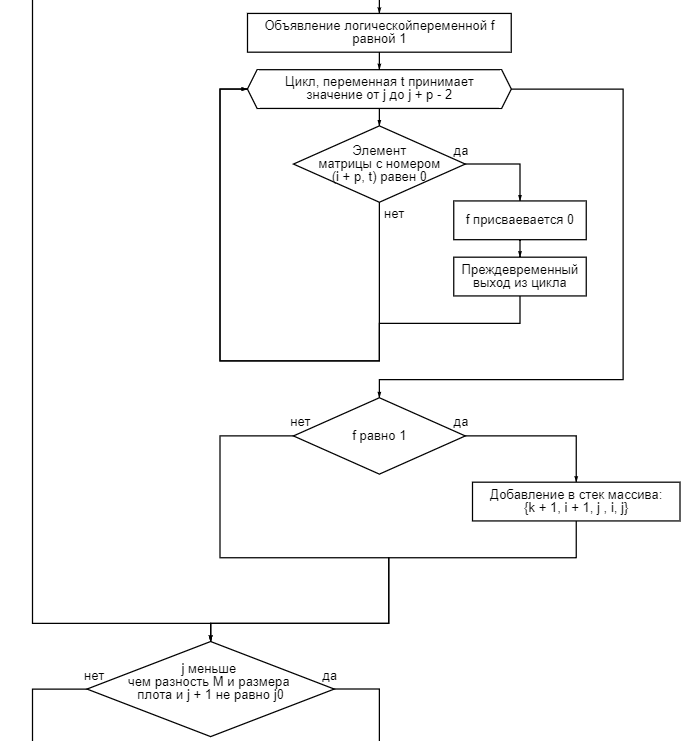


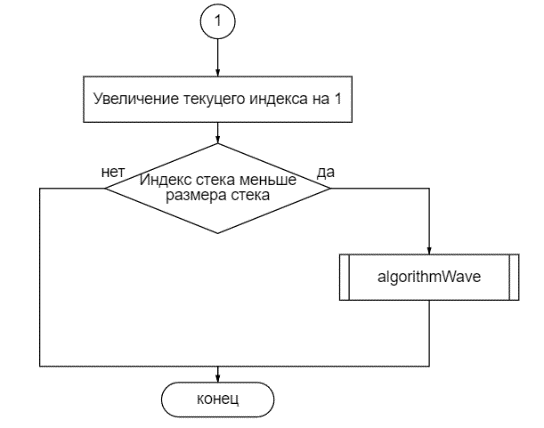
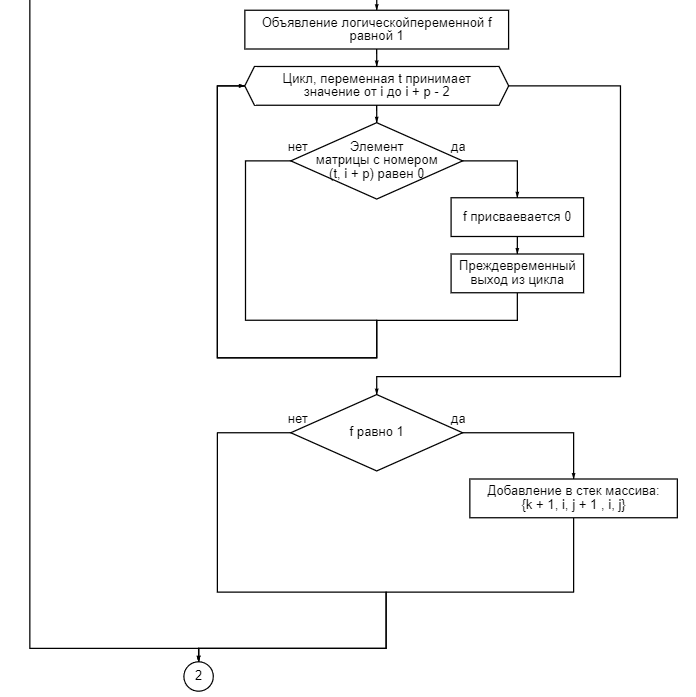


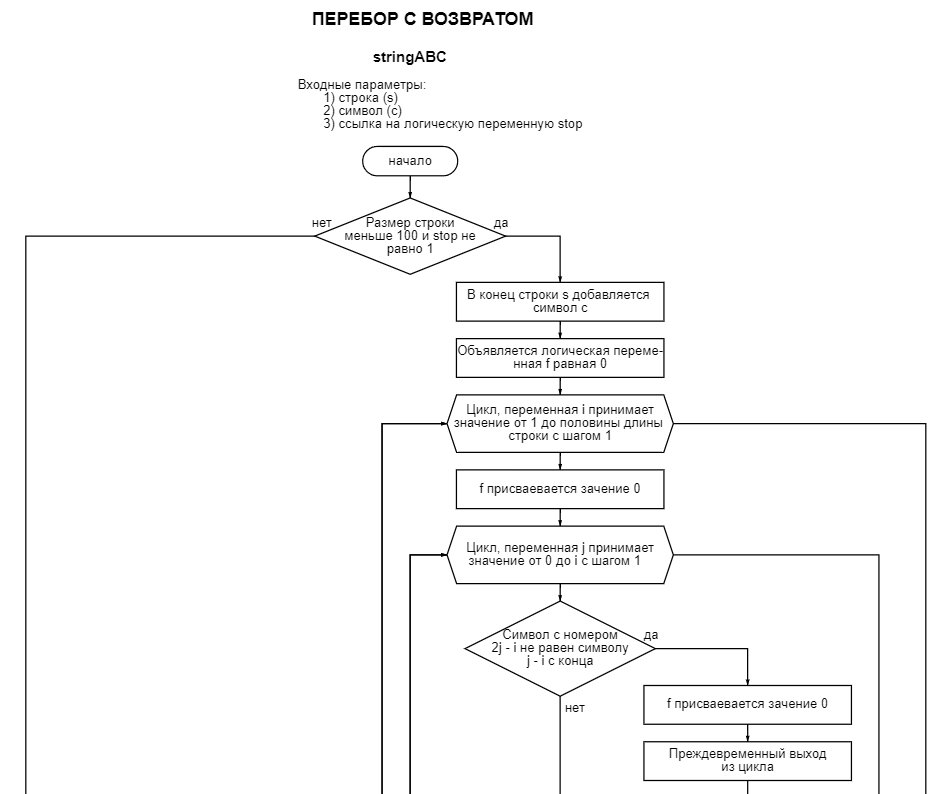


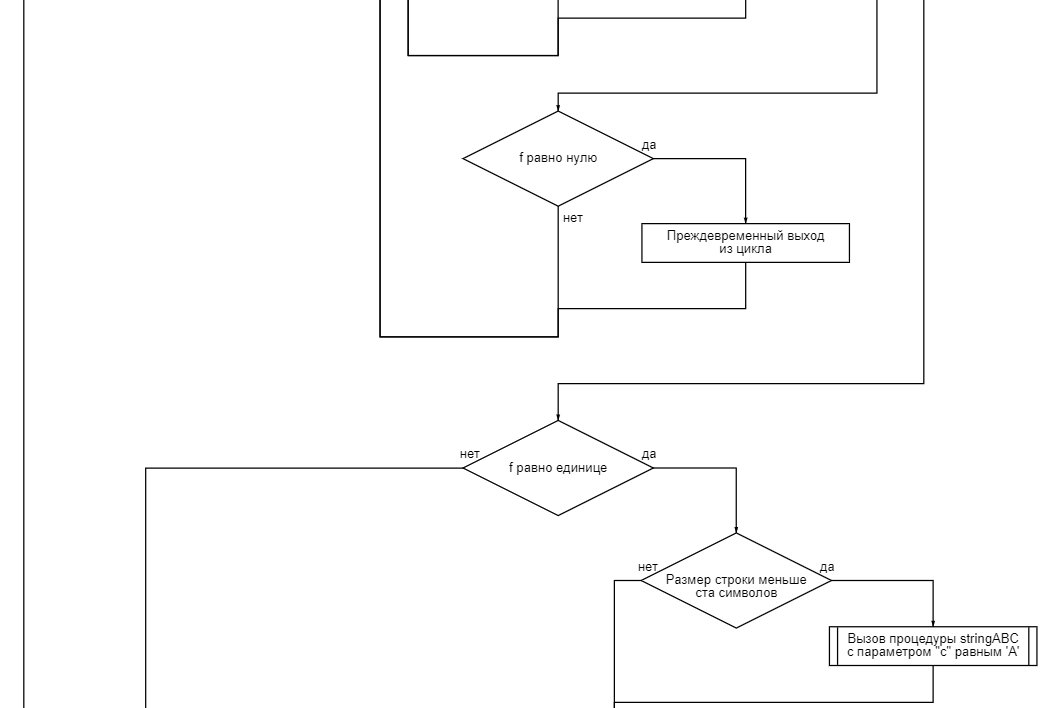


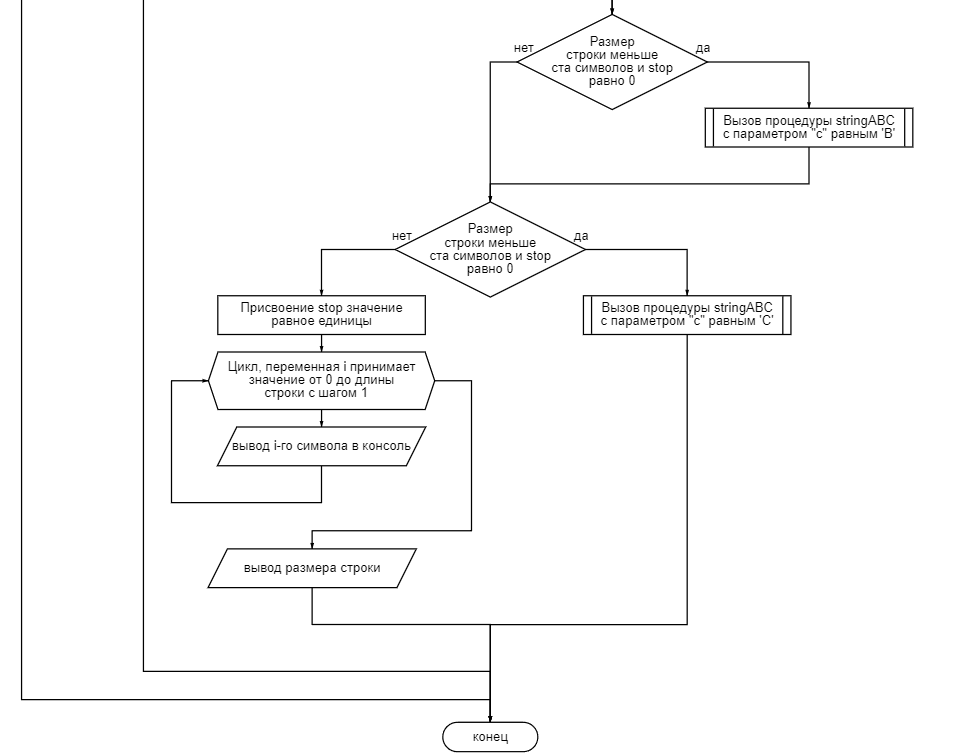


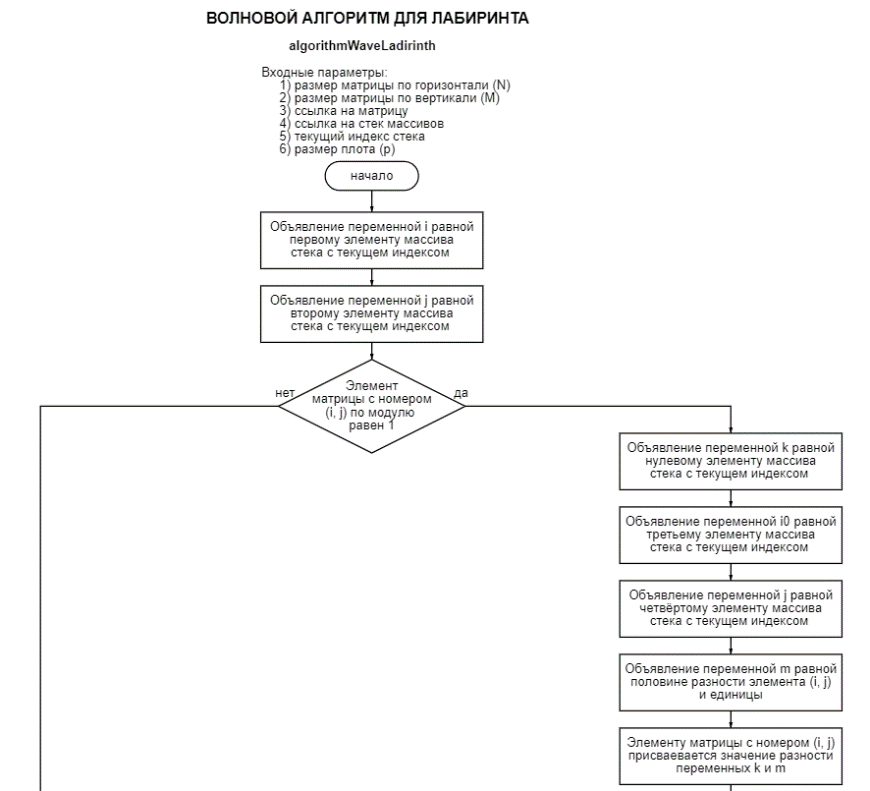


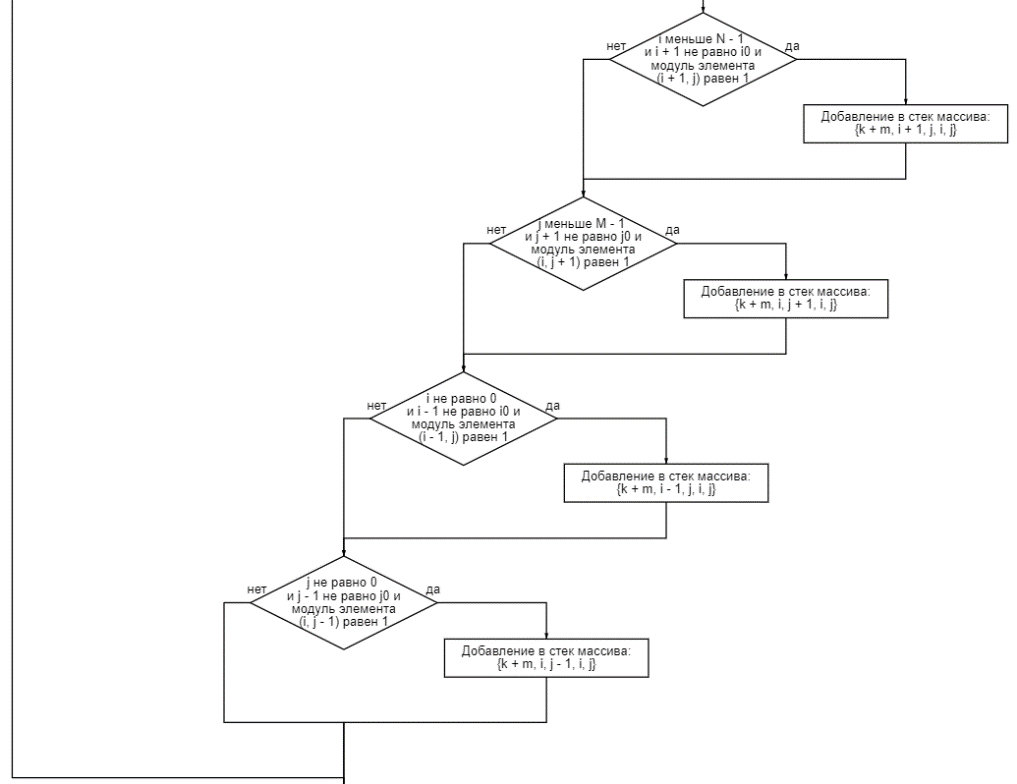


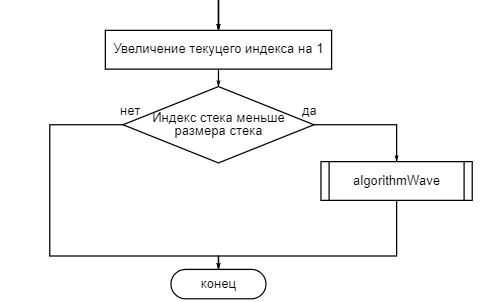


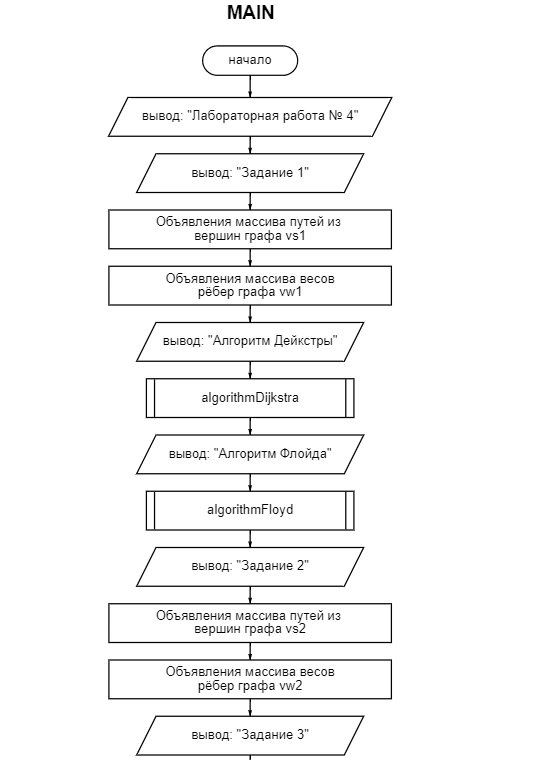
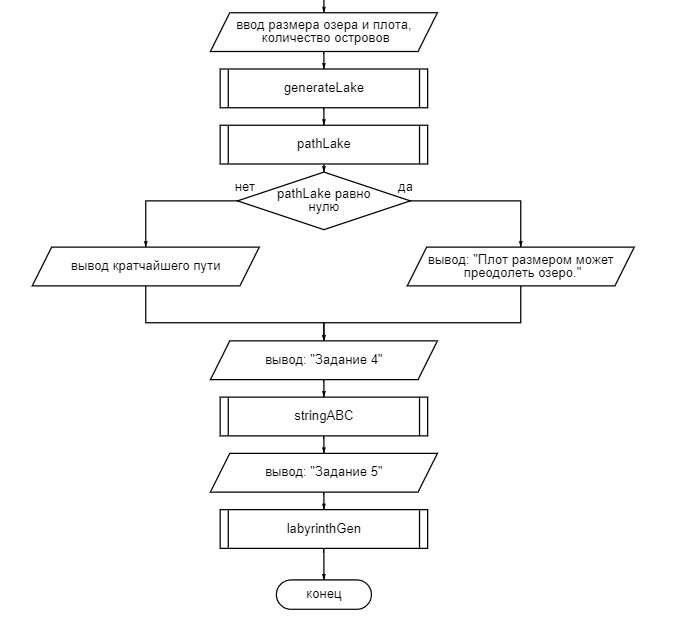












**Листинг программы:**

#include <iostream>

#include <vector>

using namespace std;

//-----------------------------------------------------------------ФУНКЦИИ К 1-Й и 2-Й ЗАДАЧАМ--------------------------------------------------------------------

/\*

-------ФУНКЦИЯ ИНИЦИАЛИЗАЦИИ МАТРИЦЫ ГРАФА-------

Трансформирует граф, представленный в виде двух

массивов: массив связей vs и массив весов vw,

в квадратную матрицу, каждый элемент которой

равен весу пути между вершинами двумя вершинами.

\*/

vector<vector<int>> transGraf(vector<vector<int>> vs, vector<vector<int>> vw, bool flag)

{

int len = vs.size();

if (flag)

len++;

vector<vector<int>> G (len, vector <int>(len));

for (int i = 0; i < len - 1; i++)

{

int k = 0;

if (vs[i].size() == 0)

{

for (int j = 0; j < len; j++)

{

if (G[i][j] == 0)

{

G[i][j] = INT\_MAX;

}

}

}

else

{

for (int j = 0; j < len; j++)

{

if (vs[i][k] == j + 1)

{

G[i][j] = vw[i][k];

G[j][i] = vw[i][k];

if (k != vs[i].size() - 1)

{

k++;

}

}

else if (G[j][i] == 0)

{

G[i][j] = INT\_MAX;

G[j][i] = INT\_MAX;

}

}

}

}

return G;

}

/\*

---------------АЛГОРИТМ ДЕЙКСТРЫ-----------------

v0 - начальная вершина

vs - массив связей графа

vw - массив весов путей графа

flag = true если ориентированнованый граф

\*/

void algorithmDijkstra(int v0, vector<vector<int>> vs, vector<vector<int>> vw, bool flag)

{

int len = vs.size();

if (flag)

len++;

vector<bool> visited (len); // массив посещёных вершин

vector<int> distances (len); // массив расстояний от v0 до других вершин

int minW, mimI; // минимальный вес и минимальный индекс

for (int i = 0; i < len; i++)

{

visited[i] = false;

distances[i] = INT\_MAX;

}

distances[v0 - 1] = 0;

vector<vector<int>> A = transGraf(vs, vw, flag); // инициализация матрицы кратчайших путей

for (int i = 0; i < len; i++)

{

A[i][i] = 0;

}

do

{

mimI = INT\_MAX;

minW = INT\_MAX;

for (int i = 0; i < len; ++i)

{

if (!visited[i] && distances[i] < minW)

{

mimI = i;

minW = distances[i];

break;

}

}

if (mimI != INT\_MAX)

{

for (int i = 0; i < len; ++i)

{

if (A[mimI][i] != INT\_MAX && minW + A[mimI][i] < distances[i])

{

distances[i] = minW + A[mimI][i];

}

}

visited[mimI] = true;

}

} while (mimI < INT\_MAX);

// Вывод результата

cout << "\tКратчайший путь от вершины " << v0 << ":\n";

for (int i = 0; i < len; i++)

{

if (i + 1 != v0)

{

cout << "\t\tдо вершины " << i + 1 << " составляет: " << distances[i] << '\n';

}

}

cout << '\n';

}

/\*

---------------АЛГОРИТМ ФЛОЙДА--------------------

v0 - начальная вершина

vs - массив связей графа

vw - массив весов путей графа

flag = true если ориентированнованый граф

\*/

void algorithmFloyd(int v0, vector<vector<int>> vs, vector<vector<int>> vw, bool flag)

{

int len = vs.size();

if (flag)

len++;

vector<vector<int>> A = transGraf(vs, vw, flag); // инициализация матрицы кратчайших путей

for (int i = 0; i < len; i++)

A[i][i] = 0;

for (int k = 0; k < len; k++)

{

for (int i = 0; i < len; i++)

{

for (int j = 0; j < len; j++)

{

if (A[i][k] == INT\_MAX || A[k][j] == INT\_MAX)

continue;

if (i != k && j != k && i != j && A[i][k] + A[k][j] < A[i][j])

A[i][j] = A[i][k] + A[k][j];

}

}

}

// Вывод результата

cout << "\tКратчайший путь от вершины " << v0 << ":\n";

for (int i = 0; i < len; i++)

{

if (i + 1 != v0)

{

cout << "\t\tдо вершины " << i + 1 << " составляет: " << A[v0 - 1][i] << '\n';

}

}

cout << '\n';

}

//-----------------------------------------------------------------ФУНКЦИИ ДЛЯ 3-Й ЗАДАЧИ-------------------------------------------------------------------------

/\*

---------------ВОЛНОВОЙ АЛГОРИТМ------------------

Это рекурсивный алгоритм. Если данная клетка

ещё не посещена, то она принимает значение

k + 1, где k - значение клетки, из которой

пришла "волна". Далее волна распространяется

в соседние клетки по горизонтали и вертикали.

list - стек, представляющий список клеток,

куда должна распространяться волна. Волна

распространяется последовательно в те клетки,

которые указаны в списке.

Элемент стека - массив вида: {k, i, j, i0, j0}

i, j - координаты текущей клетки

i0, j0 - координаты клетки, из которой пришла волна

k - количество шагов

pSize - размер плота

\*/

void algorithmWave(int N, int M, vector<vector<int>>& A, vector<vector<int>>& list, int indexList, int pSize)

{

// координаты верхей левой клетки плота плота:

int i = list[indexList][1],

j = list[indexList][2];

// проверка, что текущая клетка - не остров:

if (A[i][j] == 1)

{

int k = list[indexList][0], // вычесление значение k для текущей клетки

i0 = list[indexList][3], // координаты клетки, из которой запустили волну

j0 = list[indexList][4];

A[i][j] = k + 1;

// проверка существование соседий, чтобы не выйти за границы массива,

// и что соседняя клетка не является той, из которой пришла волна.

// Т.к. соседних клеток всего 4, то каждая из них проверяется:

if (i < N - pSize && i + 1 != i0)

{

bool f = true;

for (int t = j; t < j + pSize - 1; t++)

{

if (A[i + pSize][t] == 0)

{

f = false;

break;

}

}

if (f)

{

list.push\_back({ k + 1, i + 1, j, i, j }); // добавляем координаты клетки в список для проверки

}

}

if (j < M - pSize && j + 1 != j0)

{

bool f = true;

for (int t = i; t < i + pSize - 1; t++)

{

if (A[t][j + pSize] == 0)

{

f = false;

break;

}

}

if (f)

{

list.push\_back({ k + 1, i, j + 1, i, j });

}

}

if (i != 0 && i - 1 != i0)

{

bool f = true;

for (int t = j; t < j + pSize - 1; t++)

{

if (A[i - 1][t] == 0)

{

f = false;

break;

}

}

if (f)

{

list.push\_back({ k + 1, i - 1, j, i, j });

}

}

if (j != 0 && j - 1 != j0)

{

bool f = true;

for (int t = i; t < i + pSize - 1; t++)

{

if (A[t][j - 1] == 0)

{

f = false;

break;

}

}

if (f)

{

list.push\_back({ k + 1, i, j - 1, i, j });

}

}

}

indexList++;

if (indexList < list.size())

{

// вычисление координат следубщей клетки

algorithmWave(N, M, A, list, indexList, pSize); // переход волны в следующую клетку

}

}

/\*

-----------------ГЕНЕРАЦИЯ ОЗЕРА------------------

функция создаёт двумерный массив "озеро"

размером N на M клеток со случайными островами.

Если от верхней левой клетки нельзя добраться

до правой нижней, то озеро пересоздаётся.

\*/

void generateLake(int N, int M, vector<vector<int>> &A, int x)

{

// значение 0 означает наличие острова

// значение 1 означает отсутствие острова

// в данной клетке.

srand(time(NULL)); // инициализация rand для текущего времени

do {

// заполнение озеро еденицами:

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

A[i][j] = 1;

}

}

// генерация координат для x островов:

int t = 0;

while (t != x)

{

// случайная генерация координат:

int i = rand() % N,

j = rand() % M;

if (A[i][j] == 1 && !(i == 0 && j == 0) && !(i == N - 1 && j == M - 1)) // если сгенерированные координаты не являются координатоми начала и конца

{

A[i][j] = 0; // помечаем клетку как остров

t++;

}

}

// проверка, что озеро преодолимо:

vector<vector<int>> list = {{0, 0, 0, 0, 0}};

algorithmWave(N, M, A, list, 0, 1);

}

while (A[N - 1][M - 1] == 1); // генерация повторяется до тех пор, пока до левой нижней клетки не дойдёт волна

// Вывод матрицы озера:

cout << "\n\t";

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

if (A[i][j] != 0)

{

cout << ". ";

}

else

{

cout << "O ";

}

}

cout << "\n\t";

}

cout << '\n';

// Повторная генерация (если необходимо):

cout << "\tСгенерировать озеро заного?[Y/N]\n\t";

char c;

cin >> c;

if (c == 'Y' || c == 'y')

{

cout << "\tВведите количество островов:\n\t";

cin >> x;

generateLake(N, M, A, x);

}

}

/\*

-------ФУНКЦИЯ ВЫЧИСЛЕНИЯ МИНИМАЛЬНОГО ПУТИ--------

Определяет минимальный путь для квадратного плота

с длиною pSize, а также может ли плот преодолеть озеро.

\*/

int pathLake(int N, int M, vector<vector<int>> A, int pSize)

{

if (pSize == 1)

{

return A[N - 1][M - 1] - 1;

}

else

{

// проверка, что плот можно разместить на начальной позиции:

for (int i = 0; i < pSize; i++)

{

for (int j = 0; j < pSize; j++)

{

if (A[i][j] == 0)

{

return 0;

}

}

}

// проверка, что плот можно разместить на конечной позиции:

for (int i = N - 1 - pSize; i < N - 1; i++)

{

for (int j = M - 1 - pSize; j < M - 1; j++)

{

if (A[i][j] == 0)

{

return 0;

}

}

}

// возвращение озера в исходное состояние:

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

if (A[i][j] != 0)

{

A[i][j] = 1;

}

}

}

// исполнение волнового алгоритма

vector<vector<int>> list = { {0, 0, 0, 0, 0} };

algorithmWave(N, M, A, list, 0, pSize);

return A[N - pSize][M - pSize] - 1;

}

}

//-----------------------------------------------------------------ФУНКЦИИ ДЛЯ 4-Й ЗАДАЧИ-------------------------------------------------------------------------

/\*

----------------ПЕРЕБОР С ВОЗВРАТОМ-----------------

Рекурсивная функция. К концу строки добавляется

буква А, В или С и и проверяется, что никакая

подстрока, заканчивающаяся в конце строки,

не равна соседней строке. Если это условие

выполняется, то функция вызывает сама себя,

но искомая строка становится длинее на одну букву.

Условие выхода из рекурсии: бинарный параметр

stop, который становися положительным, когда

строка достигает длины в 100 символов.

\*/

void stringABC(vector<char> s, char c, bool &stop)

{

if (s.size() < 100 && !stop)

{

s.push\_back(c);

bool f = false; // дополнительная переменная для выхода из цикла

// перебор всех крайних подстрок с длиной до половины искомой строки:

for (int i = 1; i <= s.size() / 2; i++) // i - длина подстроки

{

f = false;

for (int j = 0; j < i; j++)

{

// если хотя бы одна буква не совпадает, то подстроки не равны:

if (s[s.size() - 2 \* i + j] != s[s.size() - i + j])

{

f = true;

break;

}

}

// если все буквы совпали, то подстроки равны. Функция прерывается:

if (!f)

{

break;

}

}

// если никакая крайняя подстрока не равна соседней подстроке, то подбирается слудующая буква:

if (f)

{

if (s.size() < 100)

{

stringABC(s, 'A', stop);

}

if (s.size() < 100 && !stop)

{

stringABC(s, 'B', stop);

}

if (s.size() < 100 && !stop)

{

stringABC(s, 'C', stop);

}

if (s.size() == 100)

{

// Вывод строки и её размера:

stop = true;

for (int i = 0; i < s.size(); i++)

{

cout << s[i];

}

cout << ' ' << s.size() << "\n\n";

}

}

}

}

//-----------------------------------------------------------------ФУНКЦИИ ДЛЯ 5-Й ЗАДАЧИ-------------------------------------------------------------------------

/\*

------ФУНКЦИЯ ПРИСВОЕНИЯ ИМЕНИ КОМНАТЕ ЛАБИРИНТА------

\*/

void roomName(vector<vector<int>>& A, int i, int j, vector<vector<int>>& roomCords, int& ind)

{

int x0, y0;

if (A[i + 1][j] == 1)

{

x0 = i + 1;

y0 = j - 1;

}

else if (A[i - 1][j] == 1)

{

x0 = i - 3;

y0 = j - 1;

}

else if (A[i][j - 1] == 1)

{

x0 = i - 1;

y0 = j - 3;

}

else if (A[i][j + 1] == 1)

{

x0 = i - 1;

y0 = j + 1;

}

while (A[x0 - 1][y0] != 0 || A[x0][y0 - 1] != 0)

{

if (A[x0 - 1][y0] != 0)

{

x0 -= 4;

}

else

{

y0 -= 4;

}

}

bool f = true;

for (int t = 0; t < roomCords.size(); t++)

{

if (x0 == roomCords[t][1] && y0 == roomCords[t][2])

{

f = false;

break;

}

}

if (f)

{

roomCords.push\_back({ ind, x0, y0 });

ind++;

}

}

/\*

----------ВОЛНОВОЙ АЛГОРИТМ ДЛЯ ЛАБИРИНТА-------------

Функция схожа с algorithmWave, но волна при переходе

к соседней клетке присваевает значение k, а не k + 1.

При переходе волны в новую комнату лабиринта значение

k увеличивается на 1.

\*/

void algorithmWaveLadirinth(int N, int M, vector<vector<int>>& A, vector<vector<int>>& list, int indexList, int x, int y, vector<vector<int>>& roomCords, int ind)

{

int i = list[indexList][1],

j = list[indexList][2];

if (abs(A[i][j]) == 1)

{

int k = list[indexList][0], // вычесление значение k для текущей клетки

i0 = list[indexList][3], // координаты клетки, из которой запустили волну

j0 = list[indexList][4];

int m = (A[i][j] - 1) / 2;

A[i][j] = k - m;

if (i < N - 1 && i + 1 != i0)

{

if (abs(A[i + 1][j]) == 1)

{

list.push\_back({ k - m, i + 1, j, i, j });

}

}

if (j < M - 1 && j + 1 != j0)

{

if (abs(A[i][j + 1]) == 1)

{

list.push\_back({ k - m, i, j + 1, i, j });

}

}

if (i != 0 && i - 1 != i0)

{

if (abs(A[i - 1][j]) == 1)

{

list.push\_back({ k - m, i - 1, j, i, j });

}

}

if (j != 0 && j - 1 != j0)

{

if (abs(A[i][j - 1]) == 1)

{

list.push\_back({ k - m, i, j - 1, i, j });

}

}

if (m == -1)

{

roomName(A, i, j, roomCords, ind);

}

}

indexList++;

if (indexList < list.size() && A[x][y] == 1)

{

// вычисление координат следубщей клетки

algorithmWaveLadirinth(N, M, A, list, indexList, x, y, roomCords, ind); // переход волны в следующую клетку

}

}

/\*

---------ФУНКЦИЯ ГЕНЕРАЦИИ КОМНАТ В ЛАБИРИНТЕ---------

\*/

void roomGeneration(int N, int M, vector<vector<int>>& A)

{

for (int i = 0; i < 20; i += 2)

{

int r = rand() % 4 + 1;

//cout << r << ' ';

switch (r)

{

case 1:

if (int(i % 5 + 1) \* 4 != 20)

{

A[int(i / 5) \* 4 + 1][int(i % 5 + 1) \* 4] = 1;

A[int(i / 5) \* 4 + 2][int(i % 5 + 1) \* 4] = 1;

A[int(i / 5) \* 4 + 3][int(i % 5 + 1) \* 4] = 1;

}

break;

case 2:

if (int(i % 5) \* 4 != 0)

{

A[int(i / 5) \* 4 + 2][int(i % 5) \* 4] = 1;

A[int(i / 5) \* 4 + 3][int(i % 5) \* 4] = 1;

A[int(i / 5) \* 4 + 1][int(i % 5) \* 4] = 1;

}

break;

case 3:

if (int(i / 5 + 1) \* 4 != 16)

{

A[int(i / 5 + 1) \* 4][int(i % 5) \* 4 + 1] = 1;

A[int(i / 5 + 1) \* 4][int(i % 5) \* 4 + 2] = 1;

A[int(i / 5 + 1) \* 4][int(i % 5) \* 4 + 3] = 1;

}

break;

case 4:

if (int(i / 5) \* 4 != 0)

{

A[int(i / 5) \* 4][int(i % 5) \* 4 + 1] = 1;

A[int(i / 5) \* 4][int(i % 5) \* 4 + 2] = 1;

A[int(i / 5) \* 4][int(i % 5) \* 4 + 3] = 1;

}

break;

default:

break;

}

}

}

/\*

----------ФУНКЦИЯ ГЕНЕРАЦИИ ВЫХОДОВ ИЗ КОМНАТ---------

\*/

void exitGeneration(int N, int M, vector<vector<int>>& A)

{

for (int i = 0; i < 20; i++)

{

int r = rand() % 4 + 1;

switch (r)

{

case 1:

if (int(i % 5 + 1) \* 4 != 20 && A[int(i / 5) \* 4 + 2][int(i % 5 + 1) \* 4] == 0)

{

A[int(i / 5) \* 4 + 2][int(i % 5 + 1) \* 4] = -1;

}

break;

case 2:

if (int(i % 5) \* 4 != 0 && A[int(i / 5) \* 4 + 2][int(i % 5) \* 4] == 0)

{

A[int(i / 5) \* 4 + 2][int(i % 5) \* 4] = -1;

}

break;

case 3:

if (int(i / 5 + 1) \* 4 != 16 && A[int(i / 5 + 1) \* 4][int(i % 5) \* 4 + 2] == 0)

{

A[int(i / 5 + 1) \* 4][int(i % 5) \* 4 + 2] = -1;

}

break;

case 4:

if (int(i / 5) \* 4 != 0 && A[int(i / 5) \* 4][int(i % 5) \* 4 + 2] == 0)

{

A[int(i / 5) \* 4][int(i % 5) \* 4 + 2] = -1;

}

break;

default:

break;

}

}

}

/\*

----------ФУНКЦИЯ ВЫВОДА ЛАБИРИНТА В КОНСОЛЬ----------

\*/

void output(int N, int M, vector<vector<int>>& A, vector<vector<int>> &roomCoords)

{

const char\* names = "SABCDEFGHIJKLMNOPQRTU";

int len = roomCoords.size() + 1;

cout << '\n';

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

if (A[i][j] == 0)

{

if (i % 4 == 0 && j % 4 != 0)

{

cout << "--";

}

else if (!(i % 4 == 0 && j % 4 == 0))

{

cout << '|';

}

else

{

if (i > 0 && i < N - 1 && j > 0 && j < M - 1)

{

if (A[i - 1][j] == 0 && A[i + 1][j] == 0 && A[i][j - 1] != 0 && A[i][j - 1] != 0)

cout << '|';

else if (A[i - 1][j] != 0 && A[i + 1][j] != 0 && A[i][j - 1] == 0 && A[i][j + 1] == 0)

cout << '-';

else

cout << '+';

}

else if (i > 0 && i < N - 1)

{

if (A[i - 1][j] == 0 && A[i + 1][j] == 0)

cout << '|';

}

else if (j > 0 && j < M - 1)

{

if (A[i][j - 1] == 0 && A[i][j + 1] == 0)

cout << '-';

}

else

{

cout << '+';

}

}

}

else

{

if (i % 4 == 1 && j % 4 == 1)

{

bool f = true;

for (int t = 0; t < roomCoords.size(); t++)

{

if (i == roomCoords[t][1] && j == roomCoords[t][2])

{

if (A[roomCoords[t][1]][roomCoords[t][2]] == 2)

{

roomCoords[t][0] = 0;

}

cout << names[roomCoords[t][0]] << ' ';

f = false;

break;

}

}

if (f)

{

if (A[i][j] == 2)

{

bool f2 = true;

for (int t = 0; t < roomCoords.size(); t++)

{

if (roomCoords[t][0] == 0)

{

cout << " ";

f2 = false;

break;

}

}

if (f2)

{

roomCoords.push\_back({ 0, i, j });

cout << names[0] << ' ';

}

}

else

{

if (A[i][j - 1] == 0 && A[i - 1][j] == 0)

{

if (j == 17)

{

cout << names[len] << ' ';

len++;

}

else if ((A[i - 1][j + 4] == 0) || (A[i - 1][j + 4] != 0 && A[i][j + 3] == 0))

{

cout << names[len] << ' ';

len++;

}

else

{

cout << " ";

}

}

else

{

cout << " ";

}

}

}

}

else

{

if (j % 4 != 0)

{

cout << " ";

}

else

{

cout << ' ';

}

}

}

}

cout << '\n';

}

}

/\*

-------------ФУНКЦИЯ ГЕНЕРАЦИИ ЛАБИРИНТА--------------

\*/

void labyrinthGen()

{

const char\* names = "SABCDEFGHIJKLMNOPQRTU";

// Начальные размеры лабиринта и координаты выхода:

int N = 17, M = 21, x = 0, y = 10;

vector<vector<int>> A(N, vector<int>(M)); // матрица лабиринта

char ans;

do {

srand(time(NULL)); // инициализация rand для текущего времени

vector<vector<int>> roomCoords; // массив координат верхнего левого угла комнат комнат

// создание первоначальной матрицы лабиринта:

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

if (i % 4 == 0 || j % 4 == 0)

{

A[i][j] = 0;

}

else

{

A[i][j] = 1;

}

}

}

A[16][10] = 1;

A[x][y] = 1;

// генерация комнат:

roomGeneration(N, M, A);

// генерация выходов из комнат:

exitGeneration(N, M, A);

vector<vector<int>> list = { {2, 15, 10, 16, 10} };

algorithmWaveLadirinth(N, M, A, list, 0, x, y, roomCoords, 1);

// Вывод лабиринта:

output(N, M, A, roomCoords);

if (A[x][y] - 1 == 0)

{

cout << "Из лабиринта нельзя выйти\n\n";

}

else {

if (A[x][y] - 1 < 5)

{

cout << "Минимальный путь составляет: " << A[x][y] - 1 << " комнаты\n\n";

}

else

{

cout << "Минимальный путь составляет: " << A[x][y] - 1 << " комнат\n\n";

}

}

cout << "Сгенерировать новый лабиринт?[Y/N]\n\n";

cin >> ans;

}

while (ans == 'Y' || ans == 'y');

}

Main:

#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

using std::chrono::steady\_clock;

using std::chrono::microseconds;

void algorithmFloyd(int v0, vector<vector<int>> vs, vector<vector<int>> vw, bool flag);

void algorithmDijkstra(int v0, vector<vector<int>> vs, vector<vector<int>> vw, bool flag);

void generateLake(int N, int M, vector<vector<int>> &A, int x);

int pathLake(int N, int M, vector<vector<int>> A, int p);

void stringABC(vector<char> s, char c, bool &stop);

void labyrinthGen();

using namespace std;

int main()

{

steady\_clock::time\_point timeStart, timeStop;

double timeResult;

setlocale(LC\_ALL, "russian");

cout << "\t\tЛабораторная работа № 4\n\n";

//------------------------------------------ЗАДАНИЕ 1------------------------------------------------

cout << "Задание 1\n\n";

vector<vector<int>> vs1 = {

{2, 5},

{4, 7, 8},

{4, 7},

{5, 6},

{},

{7, 8},

{8}

};

vector<vector<int>> vw1 = {

{7, 6},

{2, 4, 2},

{4, 3},

{5, 1},

{},

{4, 2},

{5}

};

cout << "\tАлгоритм Дейкстры:\n\n";

timeStart = steady\_clock::now();

algorithmDijkstra(1, vs1, vw1, true);

timeStop = steady\_clock::now();

timeResult = std::chrono::duration\_cast<microseconds>(timeStop - timeStart).count() / 1000.0;

cout << "\n\tВремя выполнения алгоритма: " << timeResult << " мс\n\n";

cout << "\tАлгоритм Флойда:\n\n";

timeStart = steady\_clock::now();

algorithmFloyd(1, vs1, vw1, true);

timeStop = steady\_clock::now();

timeResult = std::chrono::duration\_cast<microseconds>(timeStop - timeStart).count() / 1000.0;

cout << "\n\tВремя выполнения алгоритма: " << timeResult << " мс\n\n";

//------------------------------------------ЗАДАНИЕ 2------------------------------------------------

cout << "Задание 2\n\n";

vector<vector<int>> vs2 = {

{4, 5, 6},

{3, 4, 5, 6},

{4, 5, 6},

{},

{}

};

vector<vector<int>> vw2 = {

{5, 6, 9},

{7, 8, 5, 8},

{4, 5, 4},

{},

{}

};

algorithmFloyd(1, vs2, vw2, true);

//------------------------------------------ЗАДАНИЕ 3------------------------------------------------

cout << "Задание 3\n\n";

int M, N, x, pSize;

cout << "\tВведите размер озера по горизонтали:\n\t";

cin >> M;

cout << "\tВведите размер озера по вертикали:\n\t";

cin >> N;

vector<vector<int>> A(N, vector<int>(M));

cout << "\tВведите количество островов:\n\t";

cin >> x;

generateLake(N, M, A, x);

cout << "\tВведите размер плота:\n\t";

cin >> pSize;

int p = pathLake(N, M, A, pSize);

if (p == 0)

{

cout << "\tПлот назмером " << pSize << " на " << pSize << " не может преодолеть озеро.\n";

}

else

{

cout << "\tКратчайший путь: " << p << "\n\n";

}

//------------------------------------------ЗАДАНИЕ 4------------------------------------------------

cout << "Задание 4\n\n";

vector<char> s;

s.push\_back('A');

bool stop = false;

stringABC(s, 'B', stop);

//------------------------------------------ЗАДАНИЕ 5------------------------------------------------

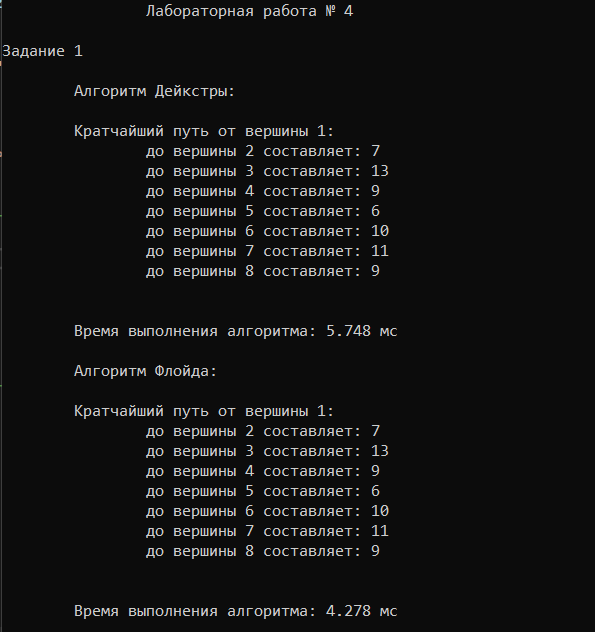
cout << "Задание 5\n\n";

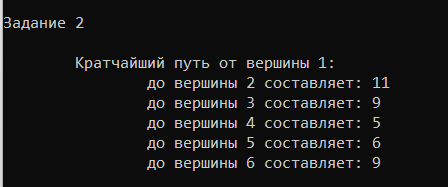
labyrinthGen();

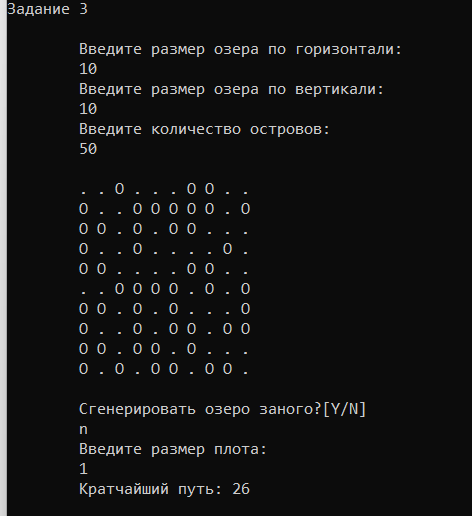
return 0;

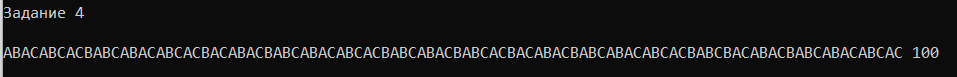
}

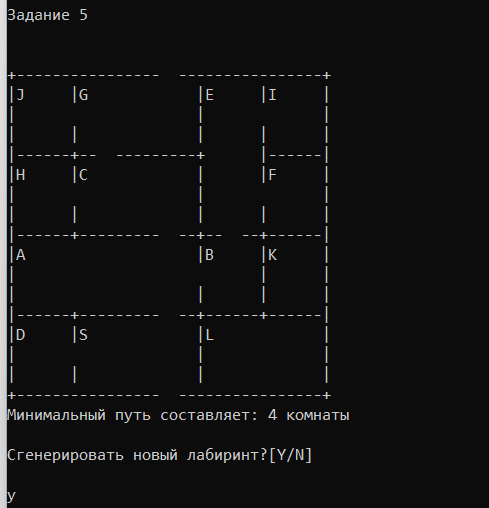
**Контрольный тест и результаты работы программы:**

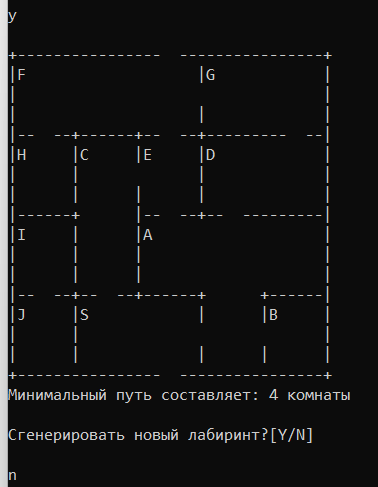
****

****

****

****

****

****

**Выводы:**

В данной лабораторной работе я изучил понятия, формирования, особенности доступа к данным и работы с памятью в деревьях, разработал программы с использованием алгоритмов рекурсии, алгоритмов обхода деревьев и алгоритмов нахождения кратчайшего пути.

Мной было выполнено 5 практических задач, для каждой я разработал и описал алгоритм в виде блок-схем, а также реализовал код на языке C++.

Таблица сравнения алгоритмов Дейкстры и Флойда:

|  |  |  |
| --- | --- | --- |
|  | Время выполнения (мс) | Количество шагов |
| Алгоритм Дейкстры | 4.177 | 255 |
| 4.287 | 255 |
| Алгоритм Флойда | 4.328 | 1042 |
| 3.987 | 1042 |