

**Содержание**

Реферат 3

Введение 4

1.Постановка задачи 5

2.Теоретическая часть 6

3.Описание алгоритма программы 8

4.Описание программы 9

5.Тестирование программы 14

6.Ручной расчет задачи 17

Заключение 20

Список литературы 21

Приложение А. Листинг 22

**Реферат**

Отчет 28 страниц ,14 рисунков

Граф ,теория графов, Алгоритм Флойда-Уоршелла, алгоритм поиска кратчайших путей

Цель исследования – разработка программы ,способная рассчитать кратчайший путь между вершинами графа

В работе используется алгоритм Флойда-Уоршелла, с его помощью можно найти кратчайший путь между вершинами графа

**Введение**

В математической теории графов и информатике граф — это совокупность непустого множества вершин и множества пар вершин.Объекты представляются как вершины, или узлы графа, а связи — как дуги, или рёбра. Для разных областей применения виды графов могут различаться направленностью, ограничениями на количество связей и дополнительными данными о вершинах или рёбрах.

Многие структуры, представляющие практический интерес в математике и информатике, могут быть представлены графами.

Математические модели в виде графов широко используются при моделировании разнообразных явлений, процессов и систем. Многие теоретические и реальные прикладные задачи могут быть решены при помощи тех или иных процедур анализа графовых моделей. В качестве примера можно привести задачу составления маршрута движения транспорта между различными городами при заданном расстоянии между населенными пунктами.

В данной работе рассматривается алгоритм Флойда — Уоршелла — алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа. Разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом.

**1.Постановка задачи**

Требуется разработать программу , которая реализует поиск кратчайших путей во взвешенном ориентированном графе с помощью алгоритма Флойда-Уоршелла.

Программа должна содержать в себе:

-Текстовое или графическое меню

-Возможность задания пользователем размера графа

-Возможность выбора автоматического или ручного задания графа

-Возможность сохранения результатов работы программы

**2.Теоретическая часть**

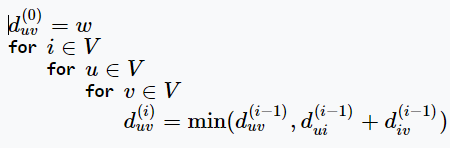
Обозначим длину кратчайшего пути между вершинами *u* и *v*, содержащего, помимо *u* и *v*, только вершины из множества :



На каждом шаге алгоритма, мы будем брать очередную вершину (пусть её номер — i) и для всех пар вершин *u* и *v* вычислять:



То есть, если кратчайший путь из *u* в *v*, содержащий только вершины из множества {1..i} , проходит через вершину *i*, то кратчайшим путем из *u* в *v* является кратчайший путь из u в *i*, объединенный с кратчайшим путем из *i* в *v*. В противном случае, когда этот путь не содержит вершины *i*, кратчайший путь из *u* в *v*, содержащий только вершины из множества {1..*i*} является кратчайшим путем из *u* в *v*, содержащим только вершины из множества {1..*i*-1}



В итоге получаем, что матрица  и является искомой матрицей кратчайших путей, поскольку содержит в себе длины кратчайших путей между всеми парами вершин, имеющих в качестве промежуточных вершин вершины из множества {1..n}, что есть попросту все вершины графа. Такая реализация работает за Θ(n3) времени и использует Θ(n3) памяти.

**3.Описание алгоритма программы**

При заполнении матрицы (с помощью файла или автоматически с выбором построения графа) используется динамический двумерный массив. Если между вершинами графа отсутствует прямой путь , то в массив записывается значение INF равное 99999.

После того как матрица выводится на экран начинает работать алгоритм Флойда-Уоршелла.

С помощью вложенных циклов производится поиск кратчайших путей. Если соблюдается условие if (dis[i][j] > dis[i][k] + dis[k][j]) {

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

записывается сумма минимальное значение.

После вышеперечисленных манипуляций измененная матрица выводится на экран.

Ниже представлен псевдокод для данного алгоритма :

1. Для k=0 пока k < n делать k++;

2.Для i=0 пока i < n делать i++;

3.Для j=0 пока j < n делать j++;

4. dis[i][j] = наименьшему значению: либо dis[i][j],

либо dis[i][k] + dis[k][j];

5. Вывод матрицы с кротчайшими расстояниями;

**4.Описание программы**

Для написания данной программы использован язык программирования С++ Язык программирования С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

При запуске программы пользователю представляется возможность выбрать количество вершин в графе.

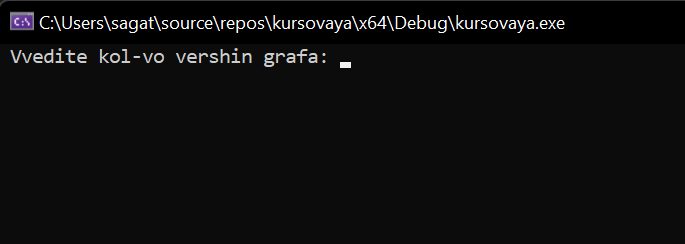


Рисунок 1-выбор количества вершин.

Далее предлагается возможность выбора способа построения графа (случайное или с файла).

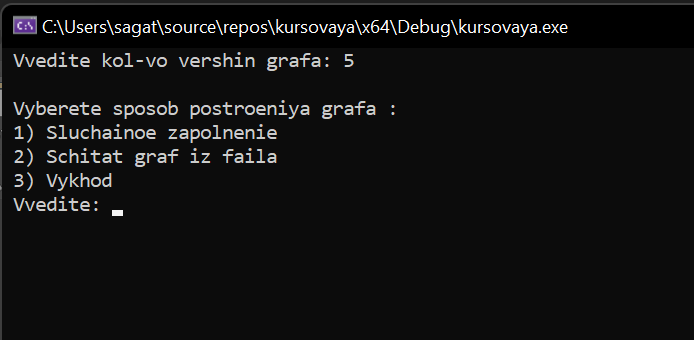


Рисунок 2-Выбор способа заполнения массива

Далее происходит вывод заполненного динамического массива при выборе «1». (Если связи между вершинами графа нет ,то пишется INF = 99999). И результат реализации алгоритма Флойда с функцией сохранения путей.

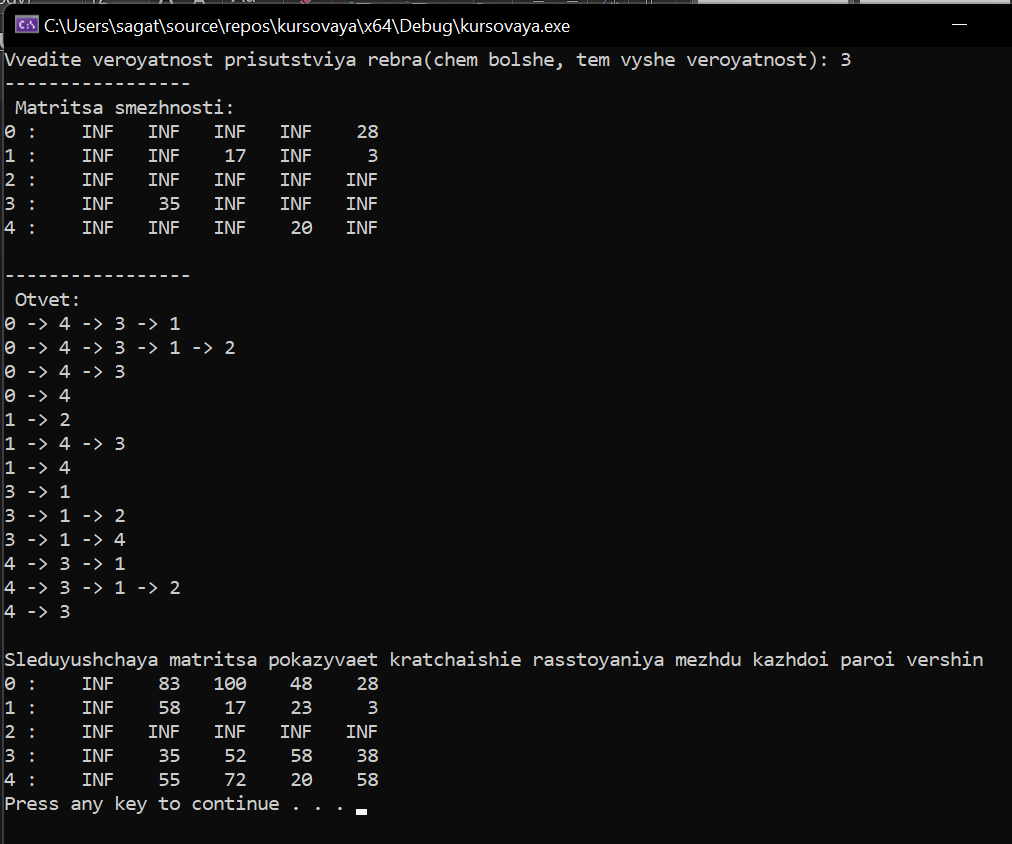


Рисунок 3-Вывод исходной и результативной матриц

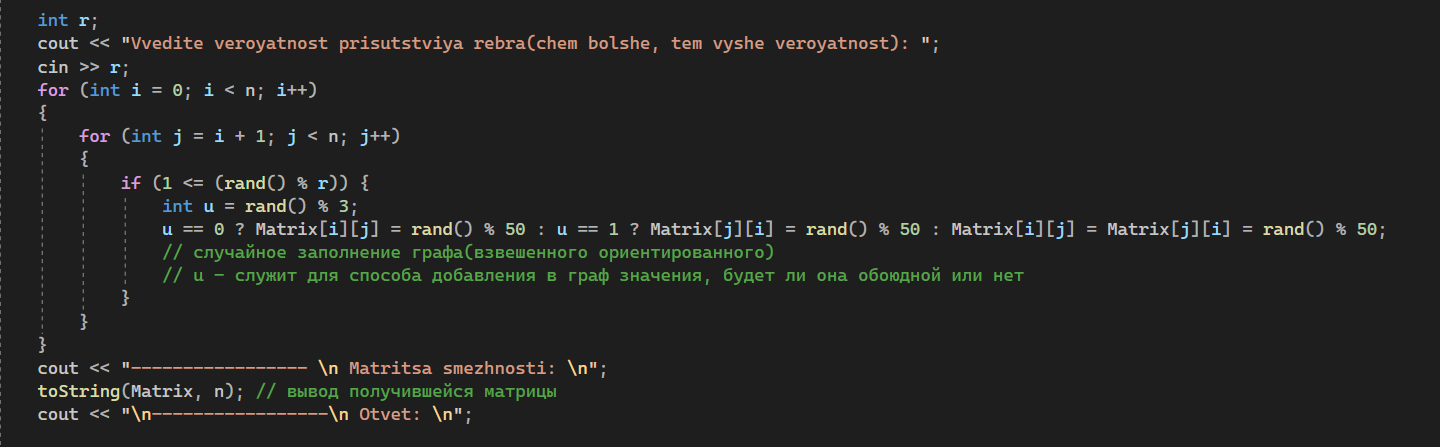


Рисунок 4- Часть кода реализующая первый способ заполнения

При выборе «2» - мы открываем файл и записываем вручную матрицу. (Если связи между вершинами графа нет ,то пишем «-1»).

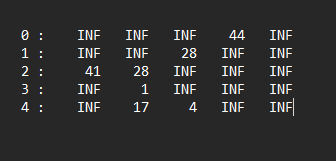


Рисунок 5 – Заполнение массива в файле

После ввода на консоль выводится наша матрица из файла и ее реализация помощью алгоритма Флойда.

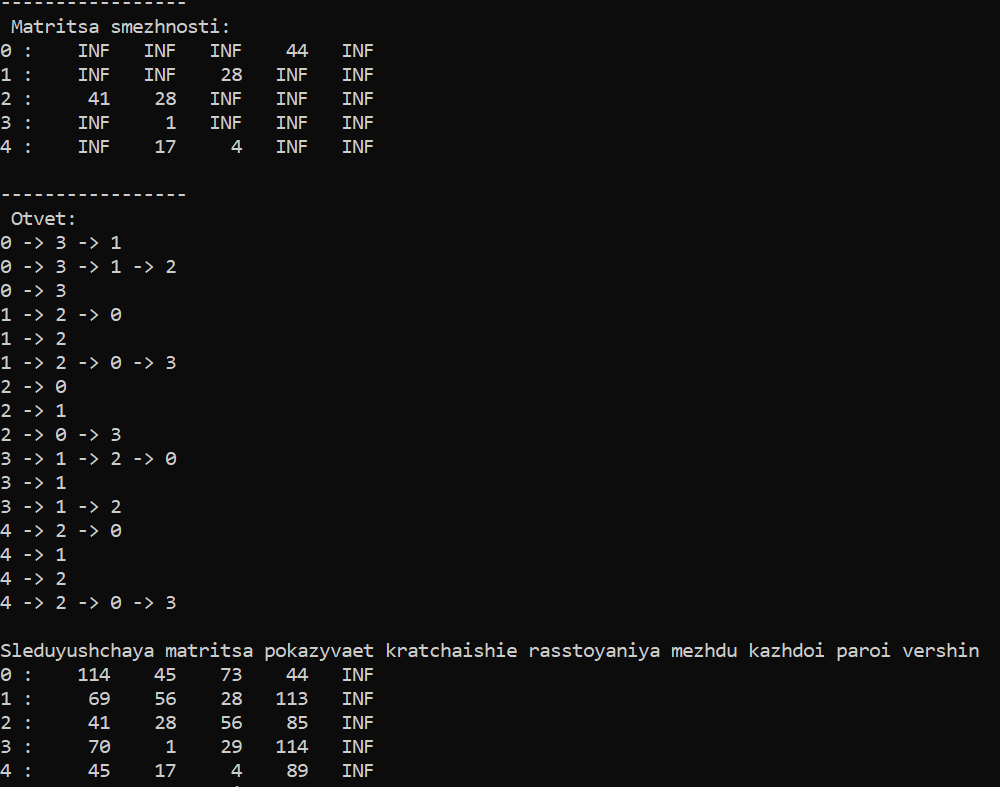


Рисунок 6 – Вывод матрицы из файла и результат программы

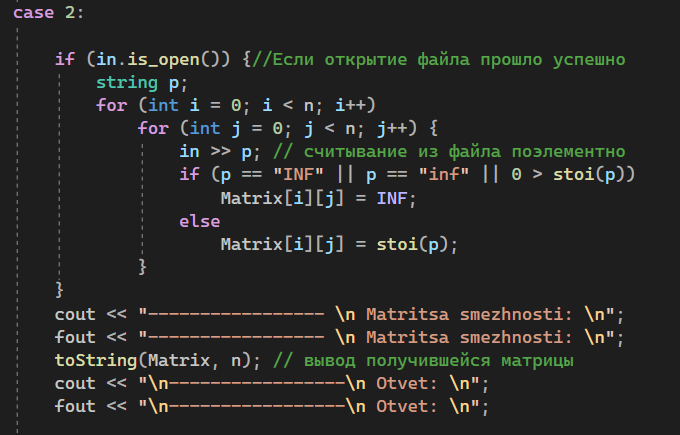


Рисунок 7 – Часть кода реализующая второй способ заполнения

Далее результаты сохраняются в файл.



Рисунок 8 – Сохранение результатов в файл

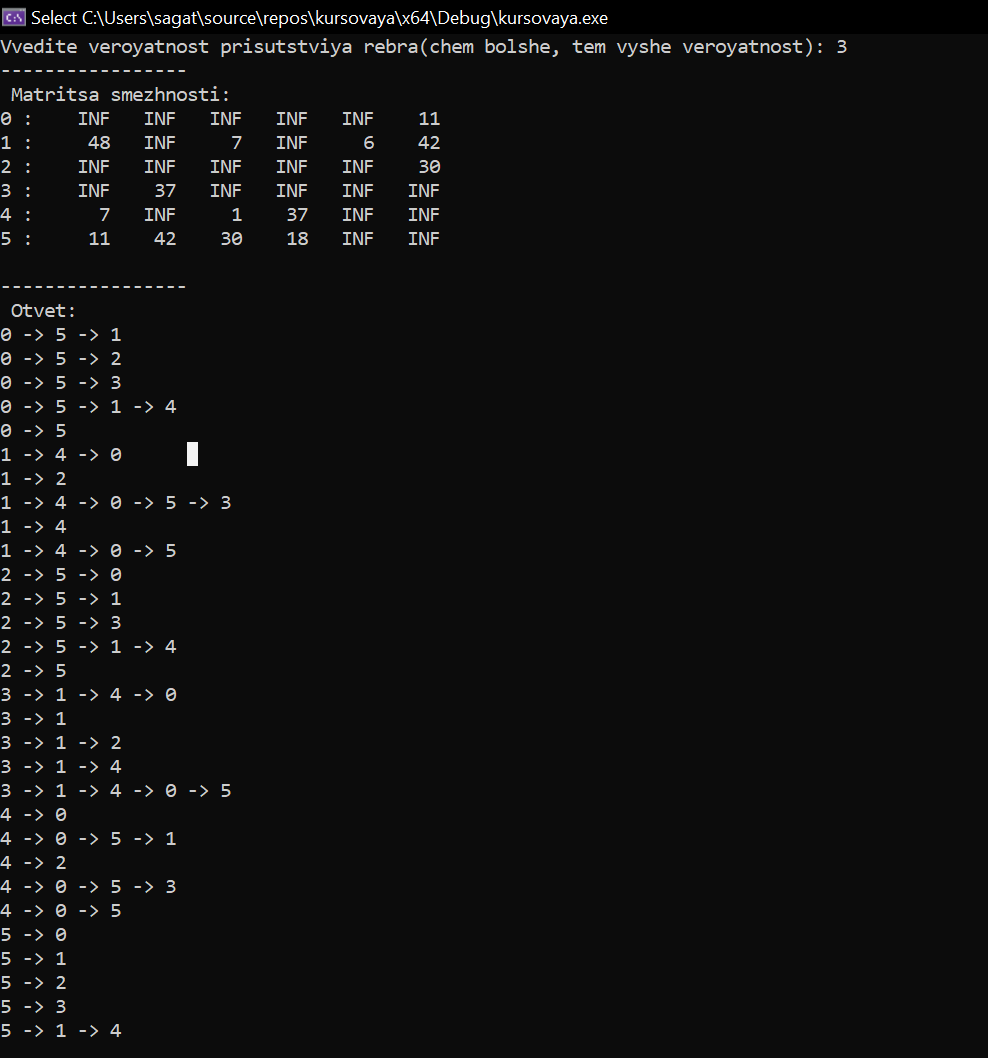
**5.Тестирование программы**

Среда разработки MicrosoftVisualStudio 2010 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки,

после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы:



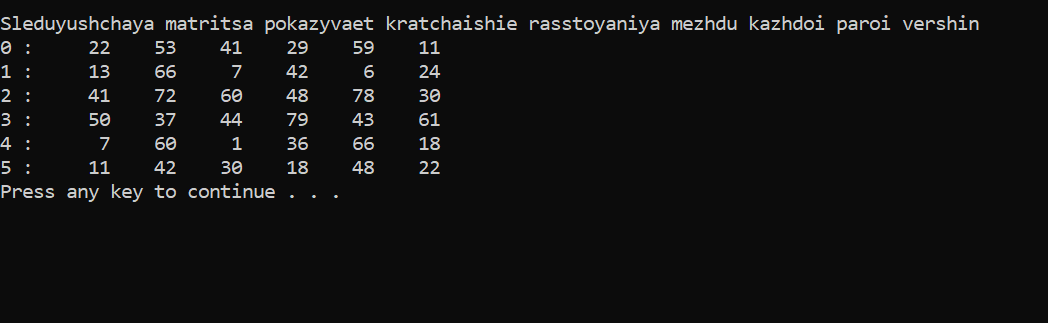
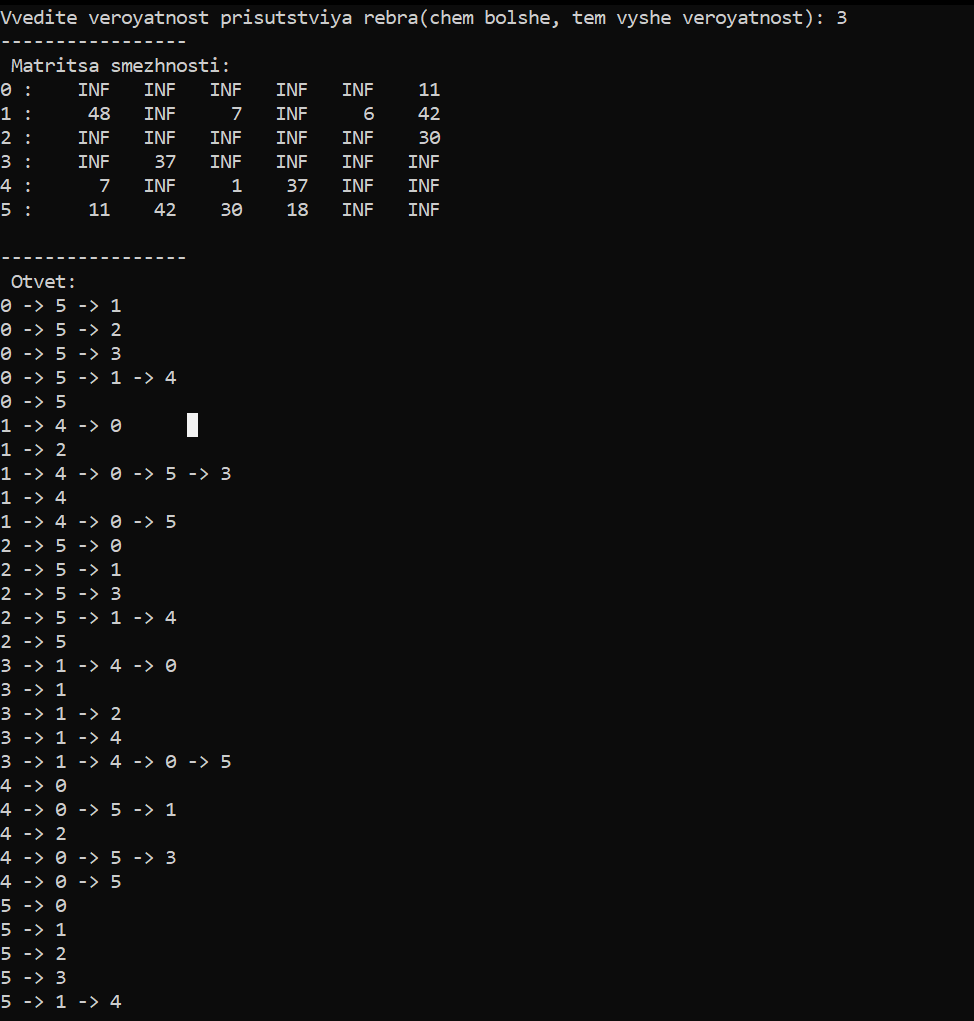


Рисунок 9 – Тестирование программы

Тестирование автоматического заполнения при N=7



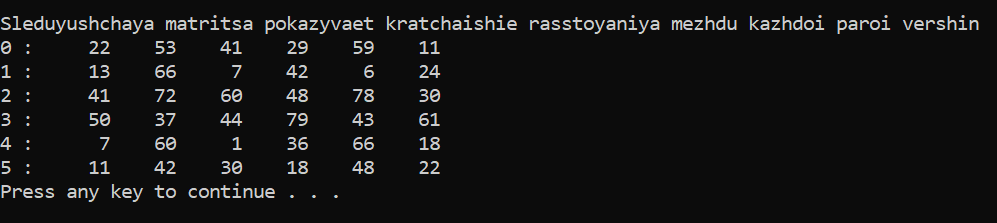


Рисунок 10 – Тестирование программы

Тестирование автоматического заполнения при N=4

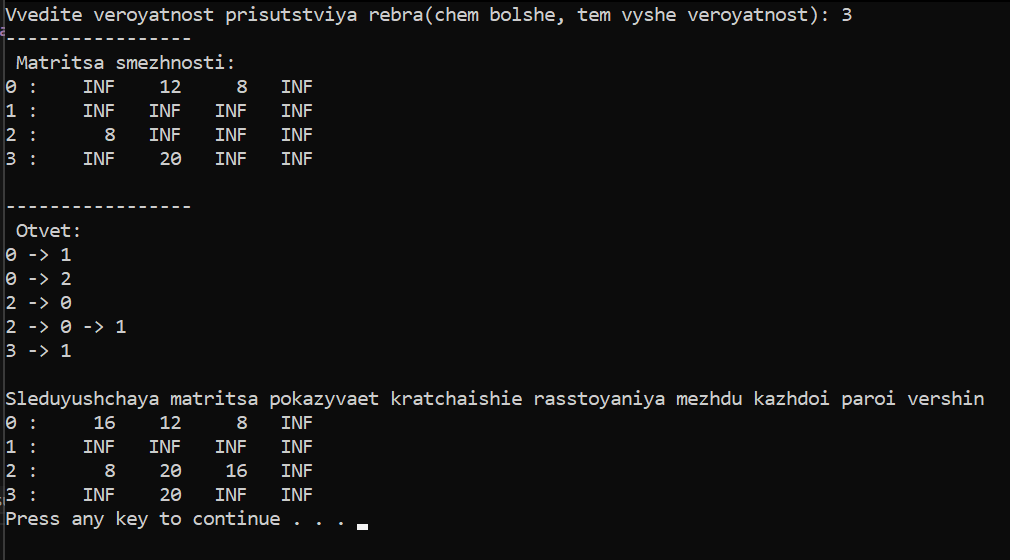


Рисунок 11 – Тестирование программы

**6.Ручной расчет задачи**

Для удобства ручного расчета ,возьмем граф с количеством вершин равным 6.

Заполняем матрицу в файле, после чего ее считывает программа и выводит результат.

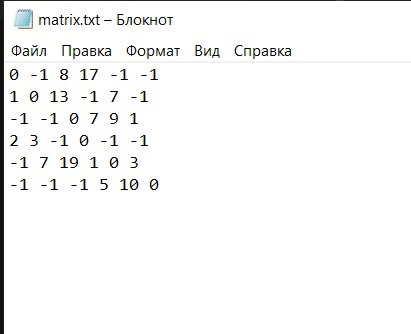


Рисунок 12-заполнение матрицы

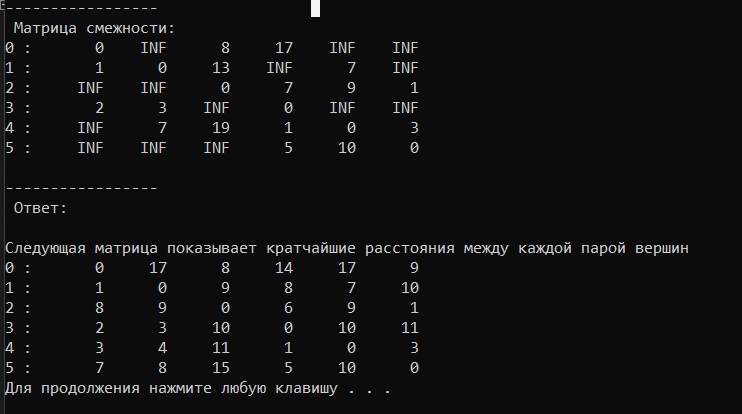


Рисунок 13-результат программы

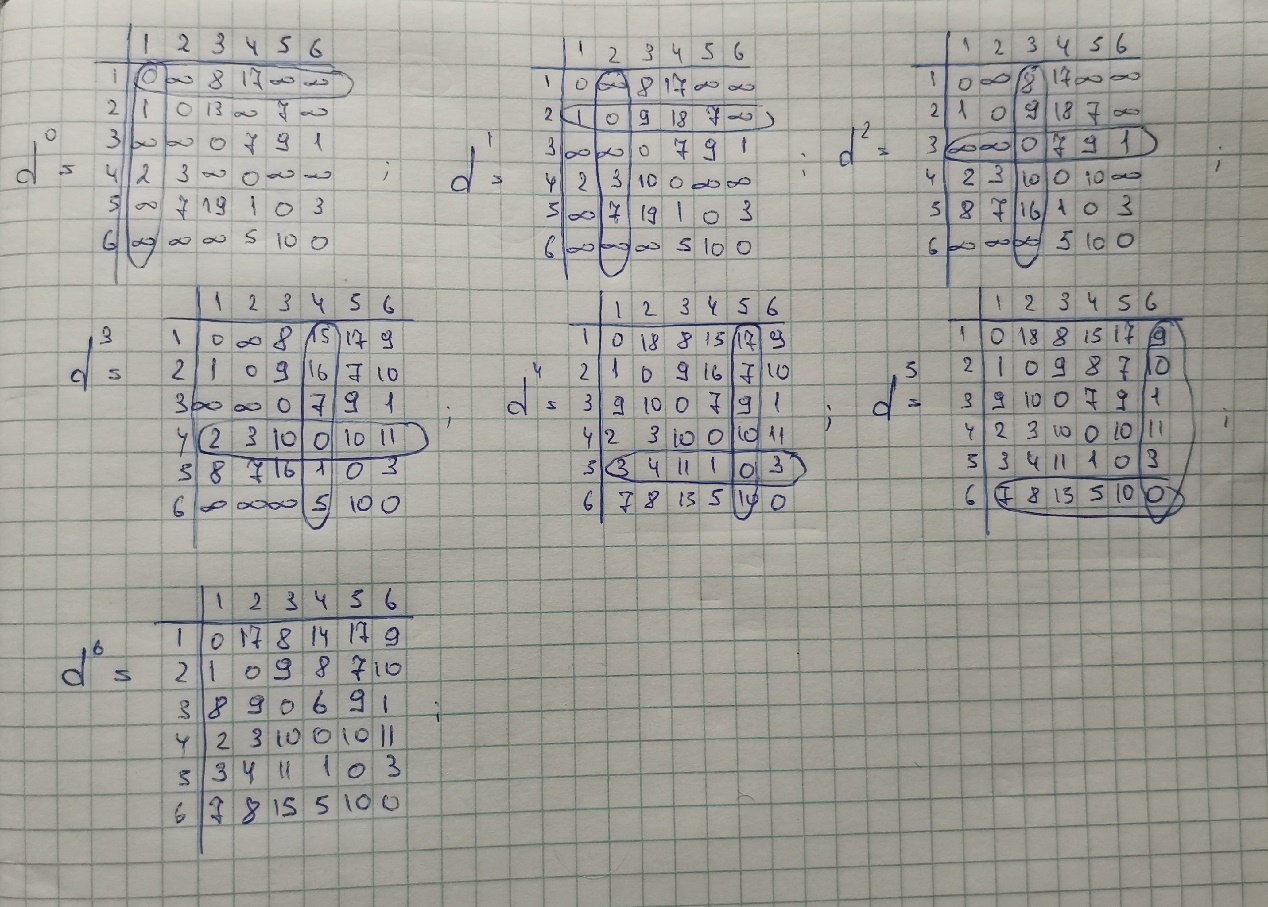


Рисунок 14- ручной просчёт программы

Результаты программы и ручного просчёта совпали, следовательно программа работает правильно.

**Заключение**

Таким образом, в процессе создания данного проекта разработана

программа, реализующая алгоритм Флойда-Уоршелла в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей,

основанных на теории орграфов. Приобретены навыки по осуществлению

алгоритма поиска кратчайших путей. Углублены знания языка программирования Cи.

Недостатком разработанной программы является примитивный

пользовательский интерфейс. Потому что программа работает в консольном

режиме, не добавляющем к сложности языка сложность программного

оконного интерфейса.

Программа имеет небольшой, но достаточный для использования

функционал возможностей.

**Список литературы**

- <https://habr.com/ru/post/105825/>

- <http://mindhalls.ru/floyd-warshall-algorithm/>

- https://www.youtube.com/watch?v=uVtDIxA9BQ4

**Приложение А. Листинг**

#include "pch.h"

#include <iostream>

#include <vector>

#include <fstream>

#include <string>

#include <iomanip>

#include <fstream>

using namespace std;

#define INF 99999

int\*\* Matrix;

int n;

ofstream fout("answer.txt");

void toString(int\*\* Matrix, int n);

void floyd()

{

// dist[][] будет выходной матрицей, которая в конечном итоге будет иметь кратчайшие

// расстояния между каждой парой вершин

//матрица предков Next[][], которая для каждой пары вершин будет содержать номер фазы,

//на которой было получено кратчайшее расстояние между ними.

int\*\* Next = new int\* [n];

int\*\* dis = new int\* [n];

for (int i = 0; i < n; i++) {

Next[i] = new int[n];

dis[i] = new int[n];

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

{

dis[i][j] = Matrix[i][j];

// Инициализирую матрицу решения так же, как входную матрицу графика.

// Нет ребра между узлами

// i и j

if (Matrix[i][j] == INF)

Next[i][j] = -1;

else

Next[i][j] = j;

}

}

// алгоритм Флойда с небольшой модификацией.

// Теперь, если мы найдем что dis[i][j] > dis[i][k] + dis[k][j]

// затем модифицируем next[i][j] = next[i][k]

for (int k = 0; k < n; k++) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

//массив dis : будем только хранить значение кратчайшего пути из i в j в dis[i][j],

//а при улучшении пути будем записать новую длину пути также в dis[i][j].

/\*Для каждой пары (i, j) исходной и целевой вершин соответственно возможны два случая.

k не является промежуточной вершиной кратчайшего пути из i в j. Мы сохраняем значение dist[i][j] как есть.

k — промежуточная вершина кратчайшего пути из i в j. Мы обновляем значение dist[i][j]

как dist[i][k] + dist[k][j], если dist[i][j] > dist[i][k] + dist[k][j]\*/

if (dis[i][j] > dis[i][k] + dis[k][j]) {

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

/\* Будем считать, что в Next[i][j] хранится номер вершины j, являющейся предшественником вершины i на кратчайшем пути из вершины.

Тогда при обновлении значения dis[i][j] нужно также обновить предшественника.

А именно, если путь i-j был обновлен на путь, проходящий через вершину k, то теперь предшественником вершины j на пути из i становится вершина,

которая была ее предшественником на пути из k\*/

Next[i][j] = Next[i][k];

// в конце цикла предпоследняя вершина в кратчайшем маршруте из вершины i в j

}

}

}

}

// сложность O(n^3);

//востановление пути

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

{

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

{

int u = i, v = j; // Если нет пути между

if (Next[u][v] == -1) // узлы u и v, просто возврат

continue; // пустой массив

// Сохранение пути в векторе

vector<int> path = { u };

while (u != v) { // аналогичен (идеи описанной выше)

u = Next[u][v];

path.push\_back(u); // записываем пусть в вектор

}

for (int k = 0; k < path.size() - 1; k++) { // вывод пути

cout << path[k] << " -> ";

fout << path[k] << " -> ";

}

if (path.size() != 1) { // если путь из одной вершины, то это не имеет смысла

cout << path[path.size() - 1] << endl;

fout << path[path.size() - 1] << endl;

}

}

}

cout << "\nSleduyushchaya matritsa pokazyvaet kratchaishie rasstoyaniya mezhdu kazhdoi paroi vershin \n";

fout << "\nSleduyushchaya matritsa pokazyvaet kratchaishie rasstoyaniya mezhdu kazhdoi paroi vershin \n";

toString(dis, n);

}

// функция вывода матрицы

void toString(int\*\* Matrix, int n) {

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

{

cout << i << " : ";

fout << i << " : ";

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

{

if (Matrix[i][j] == INF) {

cout << setw(6) << "INF";

fout << setw(6) << "INF";

}

else {

cout << setw(6) << Matrix[i][j];

fout << setw(6) << Matrix[i][j];

}

}

cout << endl;

fout << endl;

}

}

int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "RUS");

ifstream in("matrix.txt");

while (true) { // пока не выйдем сами

system("cls");// очистка консоли

cout << " Vvedite kol-vo vershin grafa: "; cin >> n;

Matrix = new int\* [n]; ;

for (int i = 0; i < n; i++) {

Matrix[i] = new int[n];

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

{

Matrix[i][j] = INF; // создание матрицы смежности( INF-бесконечность )

}

}

string s; // используем стринг для защиты от (дурака)

cout << endl << " Vyberete sposob postroeniya grafa : \n 1) Sluchainoe zapolnenie \n"

<< " 2) Schitat graf iz faila \n 3) Vihod\n Vvedite: "; cin >> s;

switch (atoi(s.c\_str()))

{

case 1:

system("cls");

int r;

cout << "Vvedite veroyatnost prisutstviya rebra(chem bolshe, tem vyshe veroyatnost): ";

cin >> r;

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

{

for (int j = i + 1; j < n; j++)

{

if (1 <= (rand() % r)) {

int u = rand() % 3;

u == 0 ? Matrix[i][j] = rand() % 50 : u == 1 ? Matrix[j][i] = rand() % 50 : Matrix[i][j] = Matrix[j][i] = rand() % 50;

// случайное заполнение графа(взвешенного ориентированного)

// u - служит для способа добавления в граф значения, будет ли она обоюдной или нет

}

}

}

cout << "----------------- \n Matritsa smezhnosti: \n";

fout << "----------------- \n Matritsa smezhnosti \n";

toString(Matrix, n); // вывод получившейся матрицы

cout << "\n-----------------\n Otvet: \n";

fout << "\n-----------------\n Otvet: \n";

floyd();

system("pause");

break;

case 2:

if (in.is\_open()) {//Если открытие файла прошло успешно

string p;

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

for (int j = 0; j < n; j++) {

in >> p; // считывание из файла поэлементно

if (p == "INF" || p == "inf" || 0 > stoi(p))

Matrix[i][j] = INF;

else

Matrix[i][j] = stoi(p);

}

}

cout << "----------------- \n Matritsa smezhnosti: \n";

fout << "----------------- \n Matritsa smezhnosti: \n";

toString(Matrix, n); // вывод получившейся матрицы

cout << "\n-----------------\n Otvet: \n";

fout << "\n-----------------\n Otvet: \n";

floyd();

system("pause");

break;

case 3:

exit(0);

break;

}

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

delete[] Matrix[i];

delete[] Matrix;

}

in.close();

fout.close();

return 0;

}