

Министерство науки и высшего образования   
Российской Федерации  
Пензенский государственный университет  
Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**к курсовой работе  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Реализация алгоритма Флойда»

Выполнил:

Студент группы 23ВВВ2:

Мадамкин В.В

Принял:

Юрова О.В.

Митрохин М.А.

Пенза 2024

**Содержание**

[Реферат 5](#_Toc187598081)

[Введение 6](#_Toc187598082)

[1. Постановка задачи. 7](#_Toc187598083)

[2. Описания алгоритма программы. 8](#_Toc187598084)

[3. Описание программы 12](#_Toc187598085)

[4.Тестирование 17](#_Toc187598086)

[Заключение 26](#_Toc187598087)

[Список литературы 27](#_Toc187598088)

[Приложение А. Листинг программы. 28](#_Toc187598089)

# Реферат

Отчет 32 стр, 11 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ПУТЕЙ В ГРАФЕ.

Цель исследования – разработка программы для поиска кратчайших путей.

В работе рассмотрена реализация алгоритм Флойда для поиска кратчайших путей в графе. Установлено, что с помощью данного алгоритма можно найти кратчайший путь.

# Введение

Алгоритм **Флойда-Уоршелла** — это алгоритм поиска кратчайших путей во взвешенном графе с положительным или отрицательным весом рёбер (но без отрицательных циклов).

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2022, язык программирования – С.

Целью данной курсовой работы является разработка программы на языке С, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Флойда.

# 1. Постановка задачи.

Требуется разработать программу, на практике показывает реализацию алгоритма Флойда.

Исходный код должен в себе содержать ориентированную и неориентированную матрицу, ввод матрицы вручную или из файла а так же ее сохранение матрицы в файл. Необходимо предусмотреть различные исходы, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода клавиатура и мышь.

Задания выполняются в соответствии с вариантом №15.

# 2. Описания алгоритма программы.

Программа предназначена для работы с графами и вычисления кратчайших путей между всеми парами вершин с использованием алгоритма Флойда-Уоршелла. Она предоставляет пользователю удобный интерфейс через текстовое меню. Рассмотрим шаги выполнения программы:

**Меню программы**

Программа запускает текстовое меню, позволяющее пользователю выбирать из следующих пунктов:

1.Ввести граф вручную.

2.Сгенерировать ориентированную матрицу.

3.Сгенерировать неориентированную матрицу.

4.Ввести матрицу из файла.

5.Выполнить алгоритм Флойда.

6.Вывести матрицу кратчайших путей.

7.Сохранить матрицу в файл.

8.Выход.

Каждый пункт меню выполняет определенную функцию, описанную ниже.

**Функции программы**

2.1. Ввод графа вручную

Пользователь задает количество вершин графа n.

Вводится матрица смежности, где значения INF (максимальное значение для int) обозначают отсутствие пути между вершинами, а нулевые значения автоматически устанавливаются на диагонали.

**Генерация ориентированного графа**

Пользователь задает количество вершин n.

Генерируется случайная матрица смежности с весами от 1 до 10, где диагональные элементы устанавливаются в 0, а остальные заполняются случайными значениями.

**Генерация неориентированного графа**

Пользователь задает количество вершин n.

Создается случайная матрица смежности с симметричными значениями (вес ребра между вершинами i и j равен весу между j и i).

Вес также случайно выбирается от 1 до 10.

**Загрузка графа из файла**

Пользователь вводит имя файла.

Из файла считываются количество вершин n и значения матрицы смежности. Если файл не удается открыть, выводится сообщение об ошибке.

**Выполнение алгоритма Флойда-Уоршелла**

Реализуется алгоритм для нахождения кратчайших путей между всеми парами вершин:

Матрица расстояний dist инициализируется исходной матрицей смежности.

Используется промежуточная вершина k, для которой проверяется возможность уменьшения пути между i и j через k.

Итоговая матрица кратчайших путей выводится на экран.

**Вывод матрицы кратчайших путей**

Если алгоритм Флойда-Уоршелла уже выполнен, выводится результат (матрица кратчайших путей). В противном случае программа выдает предупреждение.

**Сохранение матрицы в файл**

Пользователь вводит имя файла.

Матрица сохраняется в файл, где каждый элемент матрицы выводится в строку. Если файл не удается открыть, выводится сообщение об ошибке.

**Выход**

Завершение работы программы.

**Основной алгоритм**

Вход в цикл **do-while**, где отображается меню и выполняется обработка выбора пользователя.

Каждый пункт меню вызывает соответствующую функцию.

Программа позволяет последовательно вводить или генерировать граф, обрабатывать его алгоритмом Флойда-Уоршелла и сохранять результат

**Основные элементы алгоритма**

**Алгоритм Флойда-Уоршелла**

Для каждой вершины k проверяется, можно ли улучшить путь между вершинами i и j через k.

Если новый путь короче, значение в dist[i][j] обновляется.

**Работа с файлами**

Программа поддерживает ввод и сохранение графов из/в файл. Это упрощает повторное использование данных.

**Генерация графов**

Поддерживается случайная генерация ориентированных и неориентированных графов, что полезно для тестирования.

**Особенности реализации**

Константа INF используется для обозначения отсутствия пути между вершинами.

Для локализации (поддержки русского языка в консоли) используется функция setlocale(LC\_ALL, "Russian").

В программе реализованы проверки корректности входных данных (например, проверка на пустую матрицу перед выполнением алгоритма).

# 

# 3. Описание программы

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

**Подключение библиотек**

#include <iostream> // Для ввода/вывода в консоли.

#include <vector> // Для работы с динамическими массивами (векторами).

#include <limits> // Для работы с константой INF (бесконечность).

#include <locale> // Для локализации (русский язык).

#include <fstream> // Для работы с файлами (чтение и запись).

#include <cstdlib> // Для генерации случайных чисел.

#include <ctime> // Для инициализации генератора случайных чисел.

const int INF = numeric\_limits<int>::max(); // Константа INF обозначает "бесконечность".

Константа INF используется для обозначения отсутствия пути между вершинами в графе.

Она равна максимальному значению для типа int.

**Функция printMenu**

void printMenu() {

cout << "===К=У=Р=С=О=В=О=Й==========\n";

cout << "1. Ввести граф вручную\n";

cout << "2. Сгенерировать ориентированную матрицу\n";

cout << "3. Сгенерировать неориентированную матрицу\n";

cout << "4. Ввести матрицу из файла\n";

cout << "5. Выполнить алгоритм Флойда\n";

cout << "6. Вывести матрицу кратчайших путей\n";

cout << "7. Сохранить матрицу в файл\n";

cout << "8. Выход\n";

cout << "============================\n";

cout << "Выберите пункт меню: ";

}//функция меню которая выдает действия которые будут доступны пользователю

 Отображает пользователю текстовое меню программы.

 Пользователь вводит номер действия для выполнения.

**Функция printMatrix**

void printMatrix(const vector<vector<int>>& matrix, int n) {

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

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

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

cout << "ИНФА";

}

else {

cout << matrix[i][j] << " ";

}

}

cout << endl;

}

}

* Выводит матрицу в удобном для чтения формате.
* Циклы for проходят по строкам и столбцам матрицы.

**Функция inputGraph**

void inputGraph(vector<vector<int>>& graph, int& n) {

cout << "Введите количество вершин графа: ";

cin >> n;

graph.assign(n, vector<int>(n, INF));

cout << "Введите матрицу смежности графа (используйте " << INF << " для бесконечности):\n";

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

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

cin >> graph[i][j];

if (i == j) graph[i][j] = 0; // Нулевая длина пути для петли

}

}

 Пользователь вручную вводит матрицу смежности.

 Если i == j, устанавливается нулевой вес для петли.

**Функция generateDirectedGraph**

void generateDirectedGraph(vector<vector<int>>& graph, int& n) {

cout << "Введите количество вершин графа: ";

cin >> n;

graph.assign(n, vector<int>(n, INF));

srand(time(0));

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

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

if (i != j) {

graph[i][j] = rand() % 10 + 1; // Генерация случайного веса от 1 до 10

}

else {

graph[i][j] = 0; // Нулевая длина пути для петли

}

}

}

cout << "Ориентированная матрица сгенерирована:\n";

printMatrix(graph, n);

 Генерируется ориентированный граф со случайными весами.

**Функция generateUndirectedGraph**

void generateUndirectedGraph(vector<vector<int>>& graph, int& n) {

cout << "Введите количество вершин графа: ";

cin >> n;

graph.assign(n, vector<int>(n, INF));

srand(time(0));

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

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

int weight = rand() % 10 + 1; // Генерация от 1 до 10

graph[i][j] = weight;

graph[j][i] = weight; // Для неориентированного графа

}

graph[i][i] = 0; // Нулевая длина пути для петли

}

cout << "Неориентированная матрица сгенерирована:\n";

printMatrix(graph, n);

 Генерируется неориентированный граф.

 Вес задаётся симметрично: graph[i][j] == graph[j][i].

void inputGraphFromFile(vector<vector<int>>& graph, int& n) {

string filename;

cout << "Введите имя файла: ";

cin >> filename;

ifstream file(filename);

if (!file.is\_open()) {

cout << "Ошибка открытия файла!\n";

return;

}

file >> n;

graph.assign(n, vector<int>(n));

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

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

file >> graph[i][j];

if (i == j) graph[i][j] = 0; // Нулевая длина пути для петли

}

}

file.close();

cout << "Матрица загружена из файла:\n";

printMatrix(graph, n);

}

//Функция загружает матрицу смежности из файла.

//Проверяет, удалось ли открыть файл, и если нет, выводит сообщение об ошибке.

//Читает количество вершин и значения матрицы из файла, устанавливая нулевые значения для диагонали.

void floydWarshall(const vector<vector<int>>& graph, vector<vector<int>>& dist, int n) {

dist = graph;

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

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

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

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

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

}

}

}

}

cout << "Алгоритм Флойда выполнен. Матрица кратчайших путей:\n";

printMatrix(dist, n);

}

//Эта функция реализует алгоритм Флойда-Уоршелла для нахождения кратчайших путей между всеми парами вершин.

//dist инициализируется значениями из graph.

//Три вложенных цикла проверяют, можно ли улучшить кратчайший путь между вершинами i и j через вершину k.

void saveMatrixToFile(const vector<vector<int>>& matrix, int n) {

string filename;

cout << "Введите имя файла для сохранения: ";

cin >> filename;

ofstream file(filename);

if (!file.is\_open()) {

cout << "Ошибка открытия файла для записи!\n";

return;

}

file << n << endl;

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

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

file << (matrix[i][j] == INF ? INF : matrix[i][j]) << " ";

}

file << endl;

}

file.close();

cout << "Матрица сохранена в файл.\n";

}

//Функция сохраняет матрицу в файл.

//Проверяет, удалось ли открыть файл, и если нет, выводит сообщение об ошибке.

//Записывает размер матрицы и ее значения в файл.

int main() {

setlocale(LC\_ALL, "Russian");// локализуем что бы компилятор и консоль выдавала русские символы в командной строке.

vector<vector<int>> graph;

vector<vector<int>> dist;

int n = 0;

int choice;

do {

printMenu();

cin >> choice;

switch (choice) {

case 1:

inputGraph(graph, n);

break;

case 2:

generateDirectedGraph(graph, n);

break;

case 3:

generateUndirectedGraph(graph, n);

break;

case 4:

inputGraphFromFile(graph, n);

break;

case 5:

if (graph.empty()) {

cout << "Сначала введите граф!\n";

}

else {

floydWarshall(graph, dist, n);

}

break;

case 6:

if (dist.empty()) {

cout << "Сначала выполните алгоритм Флойда!\n";

}

else {

cout << "Матрица кратчайших путей:\n";

printMatrix(dist, n);

}

break;

case 7:

if (dist.empty()) {

cout << "Сначала выполните алгоритм Флойда!\n";

}

else {

saveMatrixToFile(dist, n);

}

break;

case 8:

cout << "Выход из программы.\n";

break;

default:

cout << "Некорректный выбор. Попробуйте снова.\n";

}

} while (choice != 8);

return 0;

}

//Объявляются переменные graph (матрица смежности), dist (матрица кратчайших путей), n (количество вершин) и choice (выбор пользователя).

//Цикл do - while отображает меню и обрабатывает выбор пользователя, вызывая соответствующие функции.

//В зависимости от выбора выполняются различные действия, такие как ввод графа, генерация матриц, выполнение алгоритма Флойда и сохранение матрицы в файл.

Конец программы.

# 4.Тестирование

Среда разработки MicrosoftVisualStudio 2022 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. Для эффективного тестирования была создана таблица 1.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Описание | Предусловие | Тестирование | Ожидаемый результат |
| 1 | Работа меню | Программа запущена | Запускаем программу с помощью Visual Studio | Вывод в консоли меню |
| 2 | Ввести граф вручную | Программа запущена | Вводим номер функции | Переход к выполнению функции |
| 3 | Создание ориентированного графа | Ввод 1 в меню | Ввод размера матрицы | Вывод матрицы ориентированного графа |
| 4 | Создание неориентированного графа | Ввод 2 в меню | Ввод размера матрицы | Вывод матрицы неориентированного графа |
| 5 | Загрузка матрицы из файла | Ввод 3 в меню | Ввод названия файла | Вывод матрицы |

Продолжение таблицы 1 – Описание поведения программы при тестировании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Описание | Предусловие | Тестирование | Ожидаемый результат |
| 6 | Работа алгоритма Флойда | Выполнения условия Флойда | Ввод стартовой вершины | Вывод в консоли результата алгоритма |
| 7 | Сохранение в файл результата | Создание матрицы смежности | Ввод названия файла сохранения | Появление в файле сохранения результатов |

При выборе 1, пользователь может ввести граф вручную. При выборе 2, генерируется ориентированная матрица (рисунок 1).

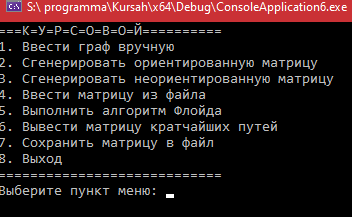


Рисунок 1 - Меню

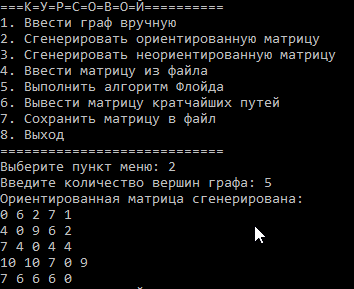


Рисунок 2 – Ориентированная матрица

При выборе пункта 3, генерируется неориентированная матрица (рисунок 2).

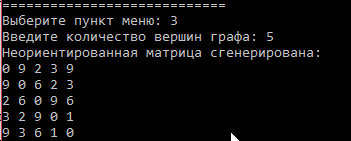


Рисунок 3 – Неориентированная матрица

При выборе пункта 4, запросится название файла, и оттуда скопируется матрица (рисунок 3).

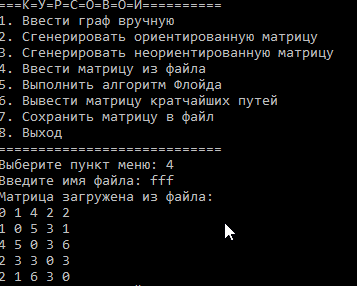


Рисунок 4 – Вывод матрицы с файла

При выборе пункта 5, происходит выполнение алгоритма Флойда.

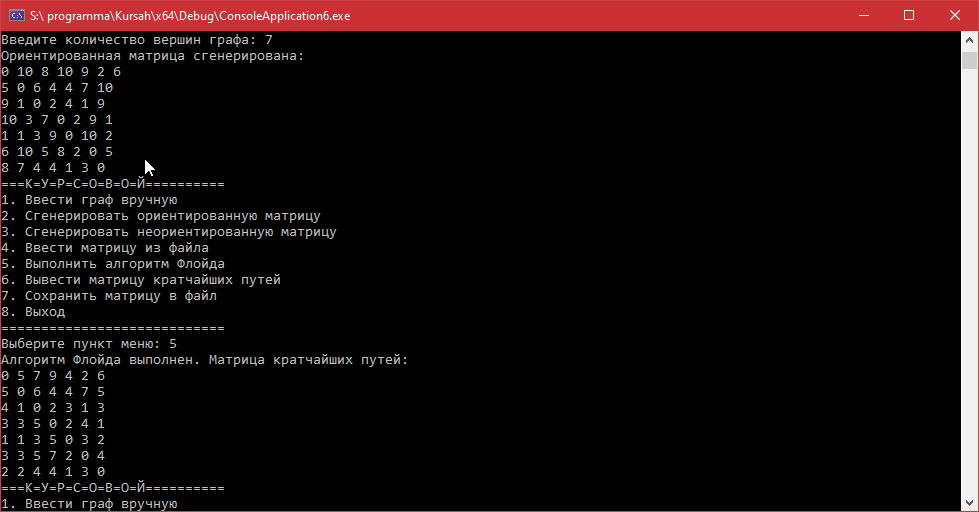


Рисунок 5 – Алгоритм Флойда

При выборе пункта 6, программа выводит матрицу кратчайших путей.

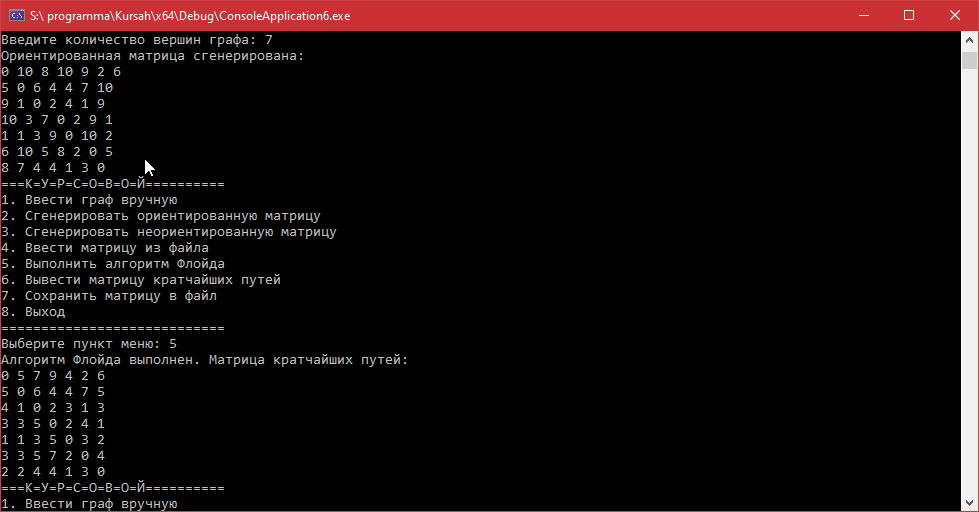


Рисунок 6 – Матрица кратчайших путей

Если вводим отрицательное число, получаем сообщение об ошибке.

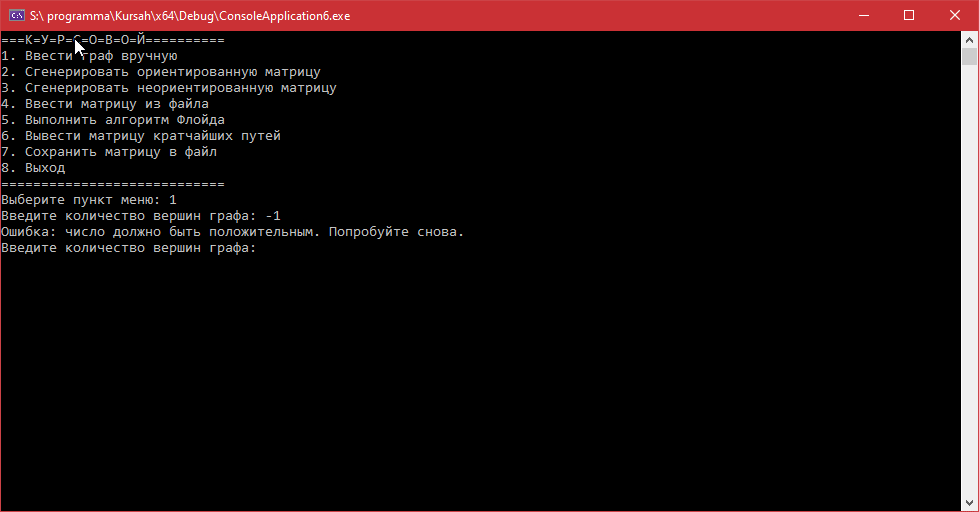


Рисунок 7 – отрицательный граф

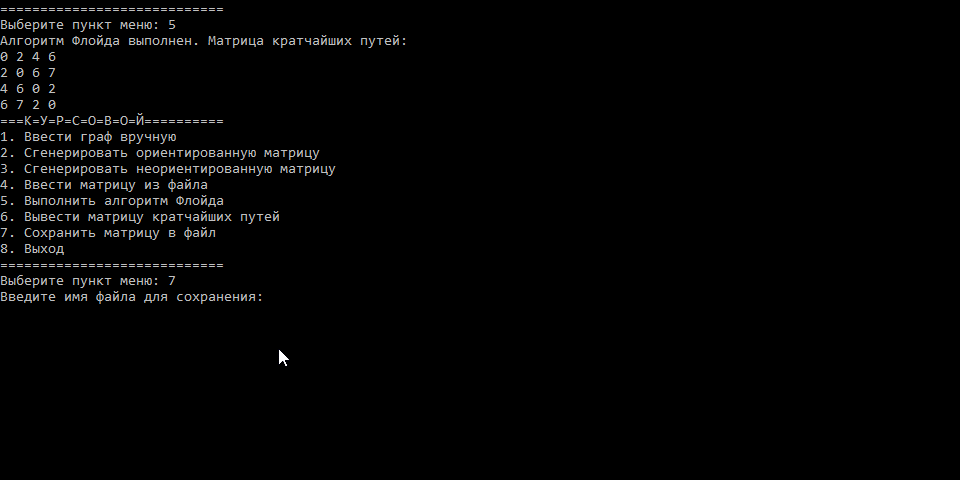
При выборе пункта 7, пользователю необходимо ввести название файла, в который будет занесен результат. 

Рисунок 8 – сохранение файла

Если мы хотим сохранить результат, не выполнив алгоритм Флойда, то программа выдаст ошибку и попросит выполнить алгоритм.

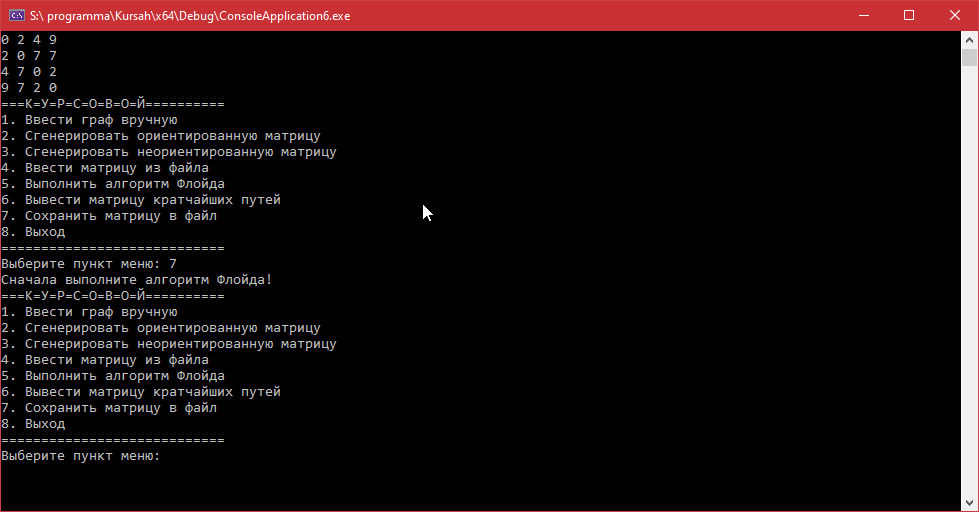
.

Рисунок 9 – просит выполнить агоритм Флойда

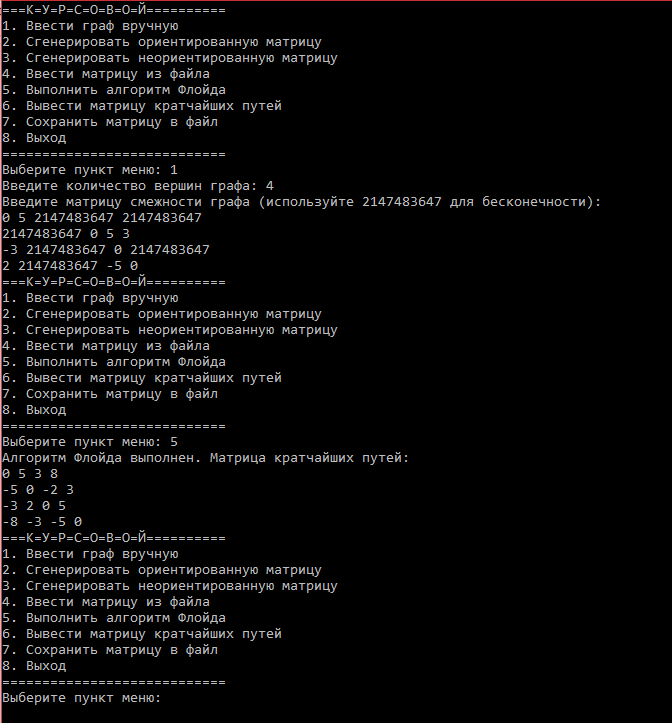


Рисунок 9- алгоритм Флойда

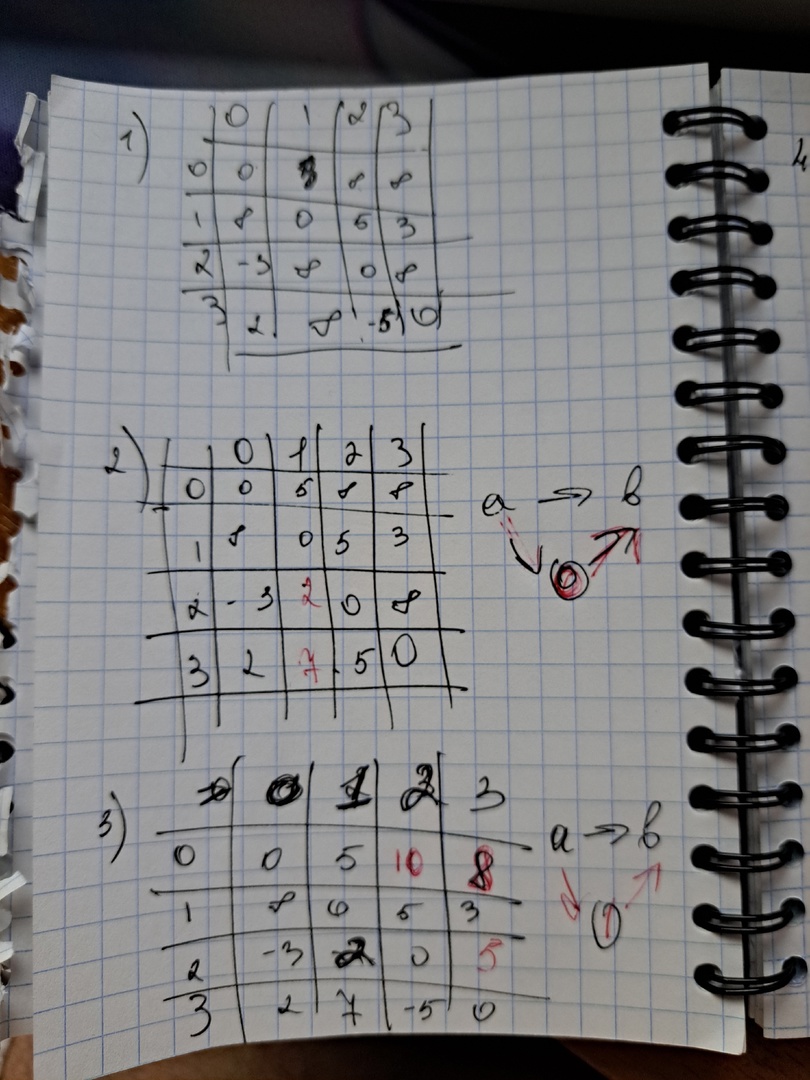


Рисунок 10 – проверка программы

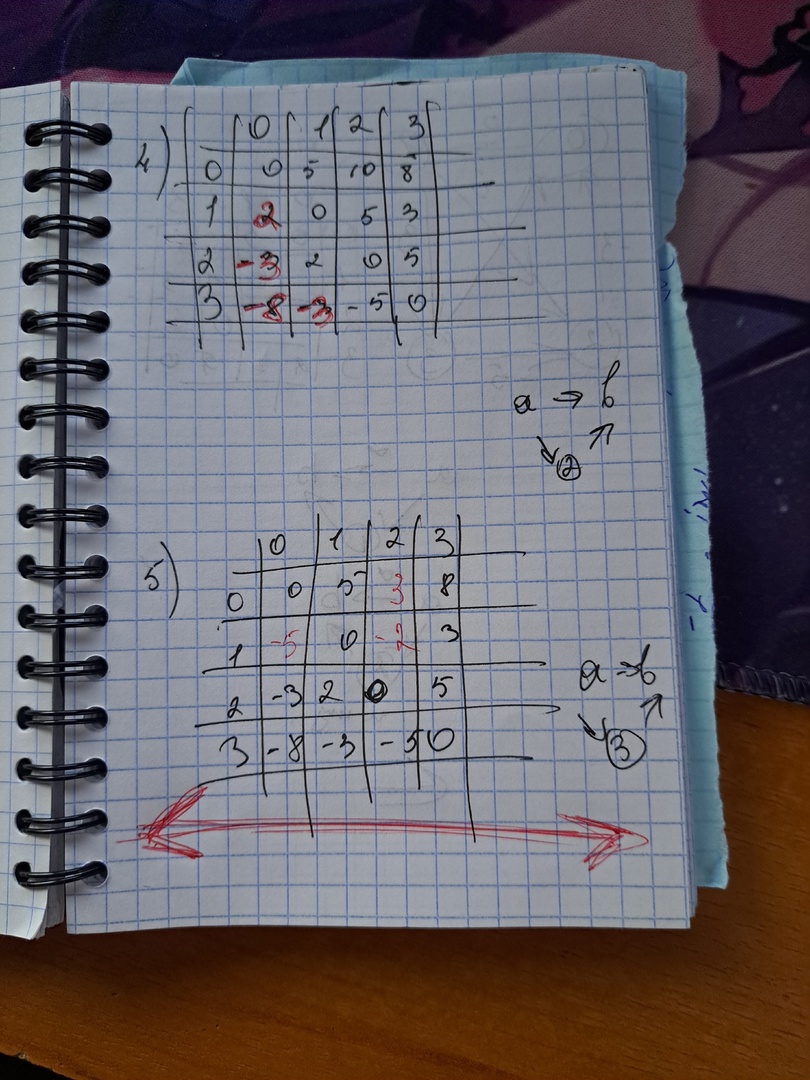


Рисунок 11- проверка программы

# Заключение

В процессе создания проекта разработана программа, реализующая алгоритм нахождения кратчайших путей между всеми парами вершин графа с использованием алгоритма Флойда-Уоршелла в Microsoft Visual Studio 2022.

При выполнении данной курсовой работы были приобретены навыки разработки программного обеспечения и освоены методы работы с матрицами смежностей, а также углублены знания языка программирования C. Реализация алгоритма Флойда позволила изучить эффективные приемы обработки графов и оптимизации вычислений.

В работе рассмотрены правила создания и заполнения матрицы смежности графа, а также изучены основы алгоритмов динамического программирования, на которых базируется алгоритм Флойда-Уоршелла. Установлено, что с помощью данного алгоритма можно эффективно вычислять кратчайшие пути в графе, содержащем любые значения весов ребер, включая бесконечные.

Недостатком разработанной программы является отсутствие удобного графического интерфейса, так как программа работает в консольном режиме, что может затруднить ее использование для неподготовленного пользователя.

Тем не менее, программа имеет необходимый функционал, включая ввод графа вручную, генерацию матриц смежности, выполнение алгоритма Флойда и сохранение результатов в файл, что делает ее подходящей для учебных и исследовательских целей.

# 

# Список литературы

Баранов, С. В. Алгоритмы и структуры данных. — М.: Наука, 2020. — 384 с.

Гусев, В. Л. Основы программирования на языке C. — М.: ДМК Пресс, 2021. — 400 с.

Кузнецов, С. В. Графы и алгоритмы. Алгоритм Флойда-Уоршелла. — Самара: Самарский гос. университет, 2019. — 450 с.

Лебедев, А. П. Программирование на C для начинающих. — М.: Велигор, 2021. — 350 с.

Николаев, И. В. Введение в теорию графов. — Екатеринбург: Урал. университет, 2021. — 300 с.

Павлов, Р. И. Алгоритмы поиска кратчайших путей в графах. — Ростов: ЮФУ, 2018. — 150 с.

# Приложение А. Листинг программы.

#include <iostream>

#include <vector>//библиотека для работы с динамическими массивами (векторами).

#include <limits>

#include <locale>

#include <fstream>//библиотека для работы с файлами (чтение и запись в файл и из файла).

#include <cstdlib>//библиотека генерации чисел

#include <ctime>//используем для иницилизации гсч

using namespace std;

const int INF = numeric\_limits<int>::max();//константа для бесконечности+ инф используем как обозначения отсутствия пути между вершинами в графе.МАКС знач тип инт

void printMenu() {

cout << "===К=У=Р=С=О=В=О=Й==========\n";

cout << "1. Ввести граф вручную\n";

cout << "2. Сгенерировать ориентированную матрицу\n";

cout << "3. Сгенерировать неориентированную матрицу\n";

cout << "4. Ввести матрицу из файла\n";

cout << "5. Выполнить алгоритм Флойда\n";

cout << "6. Вывести матрицу кратчайших путей\n";

cout << "7. Сохранить матрицу в файл\n";

cout << "8. Выход\n";

cout << "============================\n";

cout << "Выберите пункт меню: ";

}//функция меню которая выдает действия которые будут доступны пользователю

void printMatrix(const vector<vector<int>>& matrix, int n) {

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

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

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

cout << "ИНФА ";

}

else {

cout << matrix[i][j] << " ";

}

}

cout << endl;

}

}//matrix = двумерный вектор, представляющий матрицу смежности.

//n = количество вершин-размер матрицы.

//Функция проходит по всем элементам матрицы и выводит их на экран.Если элемент равен INF, выводится "ИНФA".

int inputPositiveInteger(const string& prompt) {

int value;

do {

cout << prompt;

cin >> value;

if (value <= 0) {

cout << "Ошибка: число должно быть положительным. Попробуйте снова.\n";

}

} while (value <= 0);

return value;

}

void inputGraph(vector<vector<int>>& graph, int& n) {

n = inputPositiveInteger("Введите количество вершин графа: ");

graph.assign(n, vector<int>(n, INF));

cout << "Введите матрицу смежности графа (используйте " << INF << " для бесконечности):\n";

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

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

cin >> graph[i][j];

if (i == j) graph[i][j] = 0;// Нулевая длина пути для петли

}

}

}//graph = ссылка на двумерный вектор, который будет хранить матрицу смежности.

//n = ссылка на переменную, которая будет хранить количество вершин.

//Функция запрашивает у пользователя количество вершин, инициализирует матрицу graph значениями INF, а затем запрашивает значения для матрицы смежности, устанавливая нулевые значения для диагонали.

void generateDirectedGraph(vector<vector<int>>& graph, int& n) {

n = inputPositiveInteger("Введите количество вершин графа: ");

graph.assign(n, vector<int>(n, INF));

srand(time(0));

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

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

if (i != j) {

graph[i][j] = rand() % 10 + 1;

}

else {

graph[i][j] = 0;

}

}

}

cout << "Ориентированная матрица сгенерирована:\n";

printMatrix(graph, n);

}//Эта функция генерирует случайную ориентированную матрицу смежности.

//srand(time(0)) = инициализация генератора случайных чисел на основе текущего времени.

//Внутренние циклы заполняют матрицу случайными значениями от 1 до 10 для всех пар вершин,кроме друг друга

void generateUndirectedGraph(vector<vector<int>>& graph, int& n) {

n = inputPositiveInteger("Введите количество вершин графа: ");

graph.assign(n, vector<int>(n, INF));

srand(time(0));

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

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

int weight = rand() % 10 + 1;// Генерация случайного веса от 1 до 10

graph[i][j] = weight;

graph[j][i] = weight;

}

graph[i][i] = 0;// Нулевая длина пути для петли

}

cout << "Неориентированная матрица сгенерирована:\n";

printMatrix(graph, n);

}//Эта функция генерирует случайную неориентированную матрицу смежности.

//Здесь веса задаются симметрично = если i связано с j, то и j связано с i.

void inputGraphFromFile(vector<vector<int>>& graph, int& n) {

string filename;

cout << "Введите имя файла: ";

cin >> filename;

ifstream file(filename);

if (!file.is\_open()) {

cout << "Ошибка открытия файла!\n";

return;

}

file >> n;

if (n <= 0) {

cout << "Ошибка: количество вершин в файле должно быть положительным.\n";

file.close();

return;

}

graph.assign(n, vector<int>(n));

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

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

file >> graph[i][j];

if (i == j) graph[i][j] = 0;// Нулевая длина пути для петли

}

}

file.close();

cout << "Матрица загружена из файла:\n";

printMatrix(graph, n);

}//Функция загружает матрицу смежности из файла.

//Проверяет, удалось ли открыть файл, и если нет, выводит сообщение об ошибке.

//Читает количество вершин и значения матрицы из файла, устанавливая нулевые значения для диагонали.

void floydWarshall(const vector<vector<int>>& graph, vector<vector<int>>& dist, int n) {

dist = graph;

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

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

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

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

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

}

}

}

}

cout << "Алгоритм Флойда выполнен. Матрица кратчайших путей:\n";

printMatrix(dist, n);

}//Эта функция реализует алгоритм Флойда-Уоршелла для нахождения кратчайших путей между всеми парами вершин.

//dist инициализируется значениями из graph.

//Три вложенных цикла проверяют, можно ли улучшить кратчайший путь между вершинами i и j через вершину k.

void saveMatrixToFile(const vector<vector<int>>& matrix, int n) {

string filename;

cout << "Введите имя файла для сохранения: ";

cin >> filename;

ofstream file(filename);

if (!file.is\_open()) {

cout << "Ошибка открытия файла для записи!\n";

return;

}

file << n << endl;

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

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

file << (matrix[i][j] == INF ? INF : matrix[i][j]) << " ";

}

file << endl;

}

file.close();

cout << "Матрица сохранена в файл.\n";

}//Функция сохраняет матрицу в файл.

//Проверяет, удалось ли открыть файл, и если нет, выводит сообщение об ошибке.

//Записывает размер матрицы и ее значения в файл.

int main() {

setlocale(LC\_ALL, "Russian");// локализуем что бы компилятор и консоль выдавала русские символы в командной строке.

vector<vector<int>> graph;

vector<vector<int>> dist;

int n = 0;

int choice;

do {

printMenu();

cin >> choice;

switch (choice) {

case 1:

inputGraph(graph, n);

break;

case 2:

generateDirectedGraph(graph, n);

break;

case 3:

generateUndirectedGraph(graph, n);

break;

case 4:

inputGraphFromFile(graph, n);

break;

case 5:

if (graph.empty()) {

cout << "Сначала введите граф!\n";

}

else {

floydWarshall(graph, dist, n);

}

break;

case 6:

if (dist.empty()) {

cout << "Сначала выполните алгоритм Флойда!\n";

}

else {

cout << "Матрица кратчайших путей:\n";

printMatrix(dist, n);

}

break;

case 7:

if (dist.empty()) {

cout << "Сначала выполните алгоритм Флойда!\n";

}

else {

saveMatrixToFile(dist, n);

}

break;

case 8:

cout << "Выход из программы.\n";

break;

default:

cout << "Некорректный выбор. Попробуйте снова.\n";

}

} while (choice != 8);

return 0;

}//Объявляются переменные graph (матрица смежности), dist (матрица кратчайших путей), n (количество вершин) и choice (выбор пользователя).

//Цикл do - while отображает меню и обрабатывает выбор пользователя, вызывая соответствующие функции.

//В зависимости от выбора выполняются различные действия, такие как ввод графа, генерация матриц, выполнение алгоритма Флойда и сохранение матрицы в файл.