

# https://sun9-36.userapi.com/impg/-d-h_6NMQ7scvWNlyXtEP_dzzGEKir4Bo_Xepw/ibOzCnzaKvA.jpg?size=1401x2160&quality=95&sign=6a8bac7612a03efc90107a43d19c61ce&type=album

Содержание

[Реферат 5](#_Toc154528602)

[Введение 6](#_Toc154528603)

[Постановка задачи 8](#_Toc154528604)

[Теоретическая часть задания 9](#_Toc154528605)

[Описание алгоритма программы 12](#_Toc154528606)

[Описание программы 13](#_Toc154528607)

[Тестирование 20](#_Toc154528608)

[Заключение 24](#_Toc154528609)

[Список литературы 25](#_Toc154528610)

[Приложение А 26](#_Toc154528611)

[Приложение В 29](#_Toc154528612)

# Реферат

Отчет 40 стр, 19 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, ПОИСК КРАТЧАЙШИХ ПУТЕЙ.

Реализация алгоритма Флойда-Уоршелла.

Цель исследования – разработка программы, выполняющей алгоритм Флойда-Уоршелла.

В работе рассмотрен способ нахождения кратчайших расстояний между всеми вершинами взвешенного графа без циклов с отрицательными весами.

# Введение

Курсовая работа посвящена одному из фундаментальных вопросов теории графов и вычислительной практики, а именно — задаче поиска кратчайших путей между всеми парами вершин графа. Эта проблема имеет не только теоретическое значение, пронизывающее глубокие слои математики и информатики, но и широкое практическое применение, начиная от маршрутизации в компьютерных сетях и заканчивая планированием логистики и расписаний транспортных средств. Среди разнообразия алгоритмов, применяемых для решения этой задачи, алгоритм Флойда-Уоршелла занимает особое место. Он представляет собой простой и эффективный метод расчёта кратчайших путей во взвешенном графе, идеально подходящий для плотных графов, где требуется вычислить расстояния между всеми парами вершин. Особенностью данного алгоритма является его универсальность и простота реализации на различных программных платформах.

В рамках данной курсовой работы будет произведен анализ теоретических основ алгоритма Флойда-Уоршелла, особенности его функционирования и этапы реализации. Также будет проведено исследование существующих модификаций и улучшений алгоритма, направленных на повышение его эффективности. Особое внимание будет уделено аспектам оптимизации работы данного алгоритма для обработки больших наборов данных, что актуально для современных приложений. Целью работы является изучение принципов работы алгоритма Флойда-Уоршелла, разработка программного кода для его реализации и экспериментальная проверка эффективности на различных входных данных. В ходе работы будут поставлены и решены следующие задачи: описание теоретической базы алгоритма, разработка программной реализации алгоритма, проведение тестирования программы, анализ результатов и формирование выводов по работе. Таким образом, данная курсовая работа позволит глубже понять аспекты проектирования алгоритмов на графах и освоить практические навыки разработки и оптимизации алгоритмических процедур, что имеет важное значение в рамках подготовки специалиста в области информационных технологий и вычислительной математики.

# 

# Постановка задачи

Требуется разработать программу, которая реализует алгоритм Флойда.

Условный граф должен задаваться матрицей смежности. Программа должна работать так, чтобы у пользователя был выбор самому вводить данные, случайно сгенерировать или же загружать из файла, после обработки данных на экран должна выводиться матрица смежности и результат работы алгоритма Флойда.

Устройство ввода- клавиатура.

Задания выполняются в соответствии с вариантом №7.

# 

# Теоретическая часть задания

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.

Основная идея алгоритма. Пусть есть три вершины i, j, k и заданы расстояния между ними. Если выполняется неравенство A[i,k]+A[k,j]<A[i,j], то целесообразно заменить путь i->j путем i->k->j. Такая замена выполняется систематически в процессе выполнения данного алгоритма.

Шаг 0. Определяем начальную матрицу расстояния A0 и матрицу последовательности вершин S0. Каждый диагональный элемент обеих матриц равен 0, таким образом, показывая, что эти элементы в вычислениях не участвуют. Полагаем k = 1.

Основной шаг k. Задаем строку k и столбец k как ведущую строку и ведущий столбец. Рассматриваем возможность применения замены описанной выше, ко всем элементам A[i,j] матрицы Ak-1. Если выполняется неравенство A[i,k] + A[k, j] < A[i,j], ( i != k, j != k, i != j), тогда выполняем следующие действия:

1. создаем матрицу Ak путем замены в матрице Ak-1 элемента A[i,j] на сумму A[i,k]+A[k,j] ;
2. создаем матрицу Sk путем замены в матрице Sk-1 элемента S[i,j] на k. Полагаем k = k + 1 и повторяем шаг k.

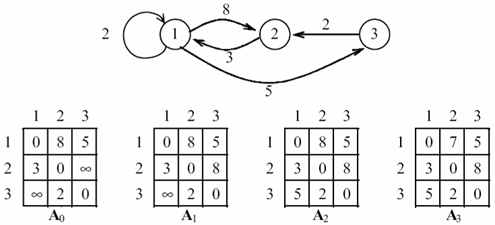
Таким образом, алгоритм Флойда делает n итераций, после i -й итерации матрица А будет содержать длины кратчайших путей между любыми двумя парами вершин при условии, что эти пути проходят через вершины от первой до i -й. На каждой итерации перебираются все пары вершин и путь между ними сокращается при помощи i -й вершины.

Рисунок 1

**Пример работы**

1.Дан граф

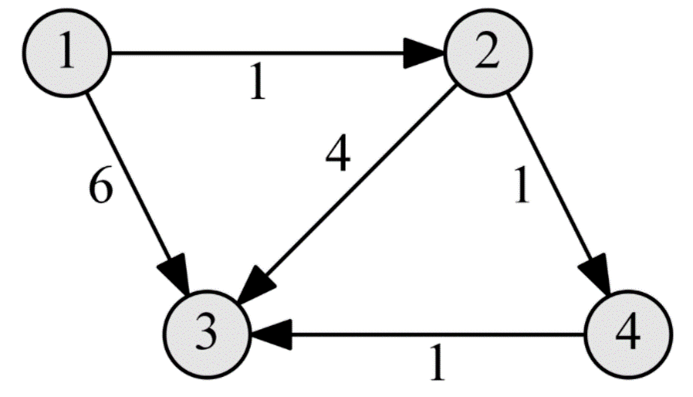


Рисунок 2

2.Выберем вершину

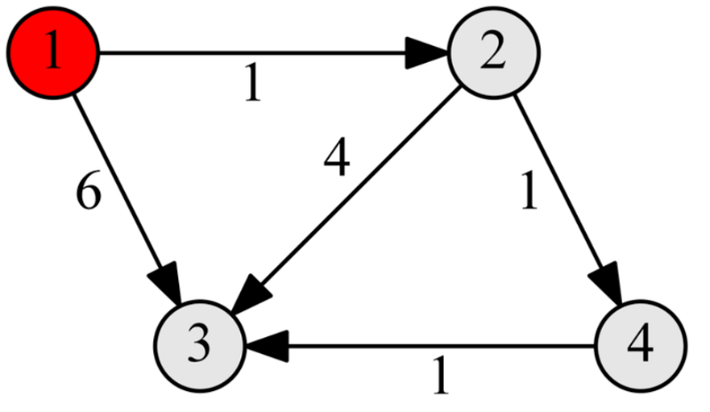


Рисунок 3

3.Выберем потенциально более короткий путь

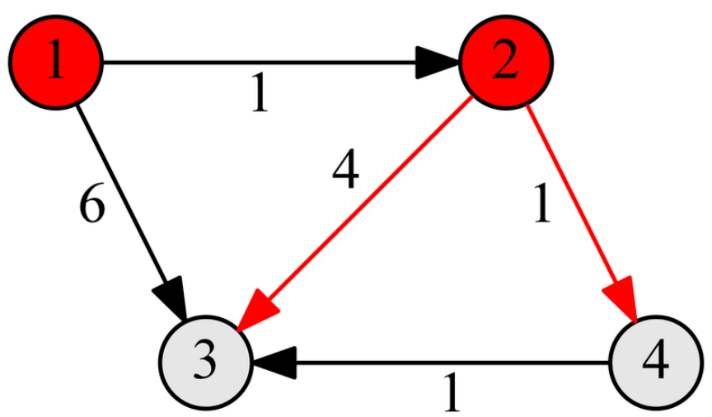


Рисунок 4

4.Выберем потенциально более короткий путь с новой вершины

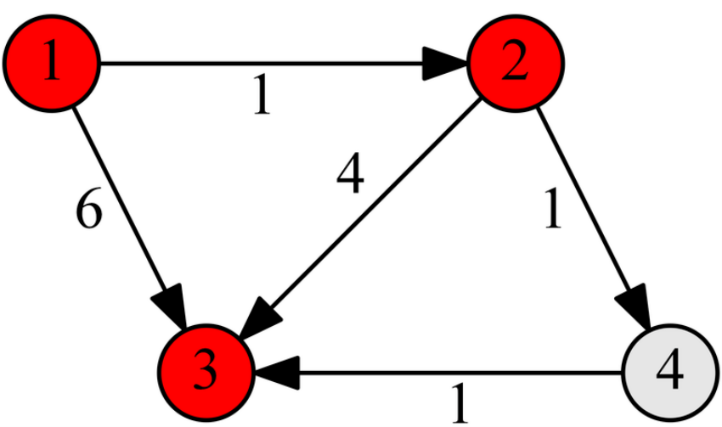


Рисунок 5

5.Самый короткий путь найден

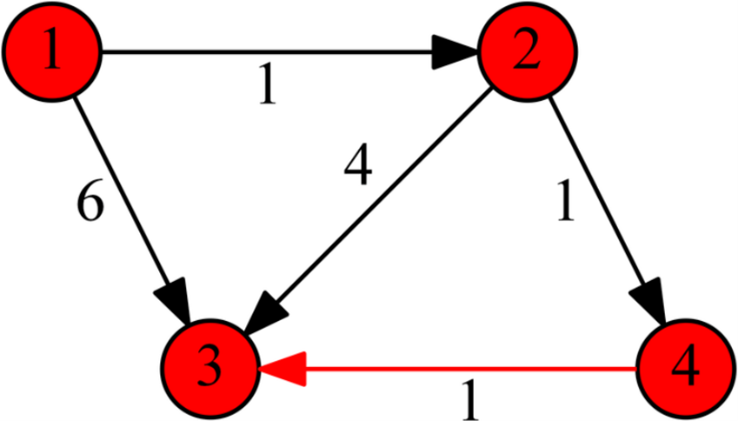


Рисунок 6

# Описание алгоритма программы

Для программной реализации алгоритма понадобиться один массив: A, в котором будут хранится величины вершин ориентированного взвешенного графа.

Создаем матрицу предшествования `p` размерности SIZE x SIZE. В этой матрице будем хранить информацию о предшествующих вершинах на кратчайших путях (строка 1 псевдокода)

Инициализируем матрицу `p`. Если элемент матрицы `a[i][j]` меньше "бесконечности" (999) и `i` не равно `j`, то указываем что предшествующим узлом для `j` является узел `i`. В противном случае ставим -1 (строки 2-8 псевдокода).

Внешний цикл (`k`) перебирает все вершины в качестве "промежуточных" точек(строка 9).

Два внутренних цикла (`i` и `j`) перебирают все возможные пары вершин(строки 10-11).

Во внутренних циклах, для каждой пары вершин (i, j) алгоритм проверяет, сможет ли он улучшить текущее известное расстояние между этими вершинами, используя вершину k как промежуточную. Если новый путь короче, он обновляет значение a[i][j] и p[i][j](строки 12-14).

Проверяем матрицу расстояний `a` на наличие отрицательных циклов, что делается проверкой диагонали матрицы на наличие отрицательных значений. Если они есть – граф содержит отрицательный цикл, и алгоритм возвращает nullptr (строки 15-18 псевдокода).  
Выводим матрицу кратчайших путей в консоль (строки 19-30 псевдокода).  
Выводим все маршруты, используя матрицу предшествования `p`. Вывод осуществляется в обратном порядке: начиная от конечной вершины и двигаемся в направлении начальной вершины, пока не достигнем ее или не столкнемся с недостижимостью вершины (`у == -1`) (строки 31-47 псевдокода).

Функция возвращает указатель на матрицу, которая теперь содержит кратчайшии пути.

Ниже представлен псевдокод функций программы.

**FloydWarshall()**

1. p = новое целое[размер][размер]

// основной цикл алгоритма, обработка всех вершин графа как промежуточные

2.для i от 0 до размер:

3.p[i] = новое целое[размер]

4.для j от 0 до размер:

5.если a[i][k] < 999 и i не равно j:

6.p[i][j] = i

7.иначе:

8. p[i][j] = -1

9.для k от 0 до размер:

10.для i от 0 до размер:

11.для j от 0 до размер

// если найден более короткий путь между i и j через k и через вершину k существует путь между i и j

12.если (a[i][k] < 999 и a[k][j] < 999) и (a[i][j] > a[i][k] + a[k][j]):

// обновляем значение кратчайшего пути между i и j

13.a[i][j] = a[i][k] + a[k][j]

14.p[i][j] = p[k][j]

// проверка наличия отрицательных циклов

15.для i от 0 до размер:

16. если a[i][i] < 0:

17.вывести "Обнаружен отрицательный цикл в вершине: ", i + 1

18.вернуть nullptr

// вывод матрицы кратчайших путей на экран

19.вывод "матрица кратчайших путей в графе:"

20.вывод пустую строку

21.для j от 0 до размер:

22.вывод "\t" + "r" + (j + 1)

23.конец цикла j

24.вывод пустую строку

25.для i от 0 до размер:

26.вывод "r" + (i + 1) + "\t|"

27.для j от 0 до размер:

28.вывод a[i][j] + "\t|"

29.вывод пустую строку

30.конец цикла i.

31.Вывод "Маршруты между вершинами:",

32.для i от 0 до размер:

33.для j от 0 до размер:

34.если i не равно j:

35. вывести "Кратчайший путь из ", i + 1, " в ", j + 1, ": "

36. целое u = j

37.пока u не равно i

38.Если u равно -1

39.Вывести недостижимо

40.Прерывание циклы

41.Вывести u+1, “<-“

42.u = p[i][u]

43.конец цикла пока

44.Вывести i+1

45.конец условия

46.конец кикла j

47.конец цикла i

46.вернуть p

# Описание программы

Для написания данной программы был использован язык программирование СИ++.

Проект был создан в виде консольного приложения в программе VS 2022.

Данная программа является многомодульной, поскольку состоит из нескольких функций: main, gen\_matrix, write\_matrix, get\_matrix, FloydWarshall, output, file\_output.

Работа программы начинается с запроса генерации матрицы в цикле. Если пользователь выберет пункт под номером 1, то ему предложат ввести путь к файлу из которого будет записываться матрица. Если выбрать пункт 2, то пользователь введёт размер матрицы и запишет матрицу с клавиатуры. Если выбрав пункт 3, то ему предложат ввести размер матрицы, затем сгенерируют матрицу со случайными значениями заданного размера. Пункт 0 завершает работу программы.

Функция main

int main()

{

setlocale(LC\_ALL, "rus");

int SIZE;

int b;

int\*\* A;

string path;

cout << "\t\t\tКурсовая работа\n\t\tПо дисциплине Л и ОА\n\t\tНа тему: \"Реализация алгоритма флойда-уоршелла поиска кратчайших путей в графе\"\n\n\tВыполнил студент группы 22ВВВ3: Гурьянов Данил\n\tПринял: Акифьев И.В";

getchar();

system("cls");

while (true) {

cout << "--->1. Загрузить из файла" << endl;

cout << "--->2. Записать через консоль" << endl;

cout << "--->3. Сгенерировать случайно" << endl;

cout << "--->0. Выход" << endl;

b = \_getch();

if (b == '0')

{

return 0;

}

if (b == '1')

{

system("cls");

cout << "--->Введите путь к файлу: "; cin >> path; cout << endl;

A = get\_matrix(path, SIZE);

output(SIZE, A);

FloydWarshall(SIZE, A);

cout << "--->1.Сохранить в файл" << endl;

cout << "--->2.Вывести в консоль" << endl;

b = \_getch();

if (b == '1')

{

cout << "Введите путь к файлу: "; cin >> path; cout << endl;

file\_output(path, SIZE, A);

}

if (b == '2')

{

output(SIZE, A);

}

}

if (b == '2')

{

system("cls");

cout << "\n--->Введите кол-во вершин: "; SIZE = getIntegerFromUser(); cout << endl;

A = write\_matrix(SIZE);

output(SIZE, A);

FloydWarshall(SIZE, A);

cout << "--->1.Сохранить в файл" << endl;

cout << "--->2.Вывести в консоль" << endl;

b = \_getch();

if (b == '1')

{

cout << "Введите путь к файлу: "; cin >> path; cout << endl;

file\_output(path, SIZE, A);

}

if (b == '2')

{

output(SIZE, A);

}

}

if (b == '3')

{

system("cls");

cout << "\n--->Введите кол-во вершин: "; SIZE = getIntegerFromUser(); cout << endl;

A = gen\_matrix(SIZE);

output(SIZE, A);

FloydWarshall(SIZE, A);

cout << "--->1.Сохранить в файл" << endl;

cout << "--->2.Вывести в консоль" << endl;

b = \_getch();

if (b == '1')

{

cout << "Введите путь к файлу: "; cin >> path; cout << endl;

file\_output(path, SIZE, A);

}

if (b == '2')

{

output(SIZE, A);

}

}

}

return 0;

}

После чего выводятся элементы массива, а именно матрица смежности орграфа c помощью функции:

void output(const int SIZE, int\*\* a)

{

//вывод матрицы в консоль

cout << "Ваша матрица: " << endl;

cout << endl;

for (size\_t j = 0; j < SIZE; j++) cout << "\t" << " R" << j + 1;

cout << endl;

for (size\_t i = 0; i < SIZE; i++)

{

cout << "R" << i + 1;

for (size\_t j = 0; j < SIZE; j++)

{

cout << "\t| " << a[i][j];

}

cout << "\t|" << endl;

}

}

Далее происходит вызов функции алгоритма и выводится матрица кратчайших расстояний и кратчайшие пути, после чего пользователь может сохранить матрицу в файл или еще раз вывести в консоль.

Потом начинается новый цикл и снова выводится меню, чтобы пользователю не было необходимо перезапускать программу.

# 

# 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.

Таблица 1

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод меню в консоль | Верно |
| Генерация матрицы случайным образом и ручным ввод размера | Вывод сообщения о вводе размера матрицы  И выводе результатов программы в консоль | Верно |
| Генерация матрицы из файла | Вывод сообщения о выборе генерации и выводе результатов программы | Верно |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Сохранение в файл | Создание файла, содержащего результаты программы | Верно |
| Выход из программы | Завершение работы программы | Верно |

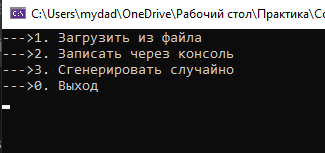


Рисунок 7 - Вывод меню в консоль

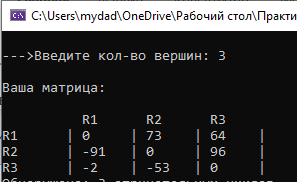


Рисунок 8 - Случайная генерация матрицы

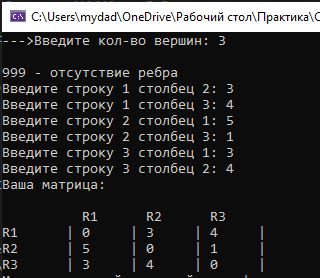


Рисунок 9 - Ручной ввод матрицы

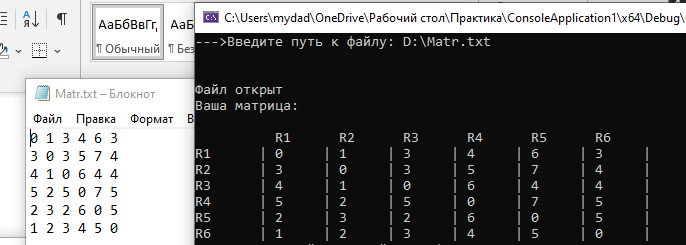


Рисунок 10 - Генерация матрицы из файла

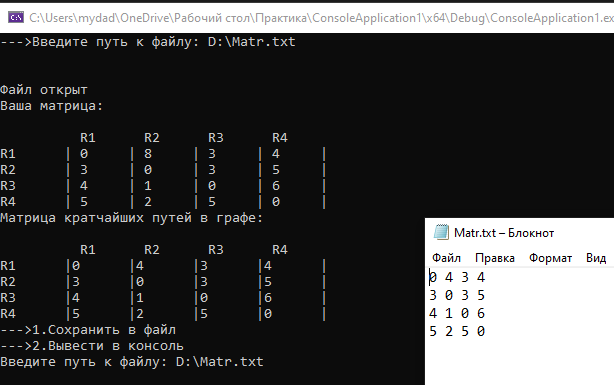


Рисунок 11 - Сохранение результата в файл

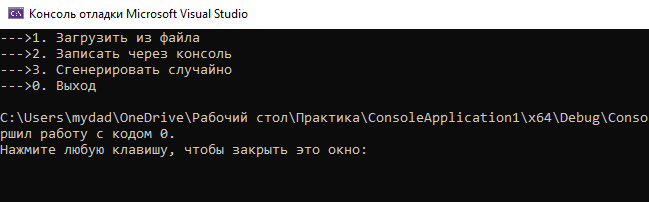


Рисунок 12 - Выход из программы

# Заключение

В процессе создания данного проекта разработана программа, реализующая алгоритм Флойда для поиска кратчайшего пути в графе в среде MicrosoftVisualStudio 2022.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма Флойда. Углублены знания языка программирования C++.

# Список литературы

1. Керниган Б. Ритчи Д. Язык программирования С. 1985 г.
2. К. Джамса. Учимся программировать на языке C++. 1997 г
3. В. Г. Давыдов. Программирование и основы алгоритмизации. 2003

# Приложение А

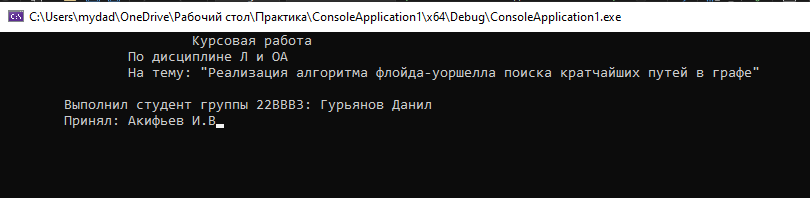


Рисунок 13 - Вывод титульника в консоли

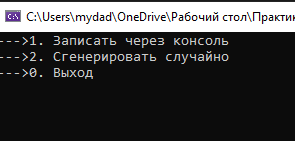


Рисунок 14 – Меню

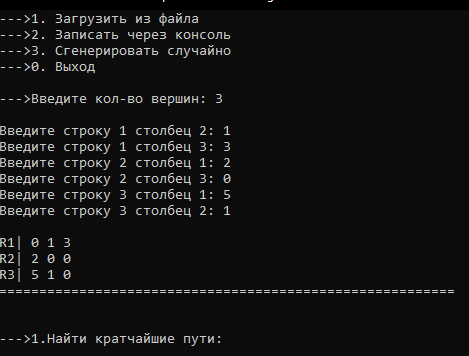


Рисунок 15 - Запись матрицы через консоль

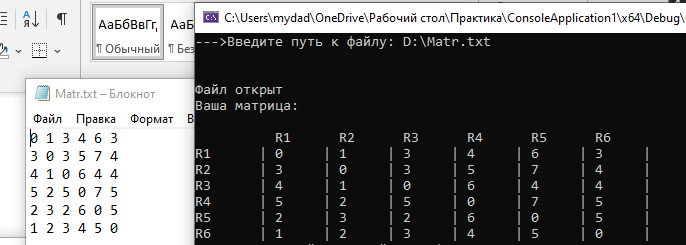


Рисунок 16 - Генерация матрицы из файла

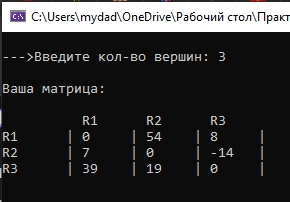


Рисунок 17 - Случайная генерация матрицы

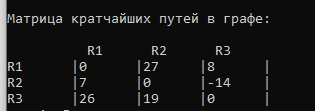


Рисунок 18 - Вывод результата работы в консоль

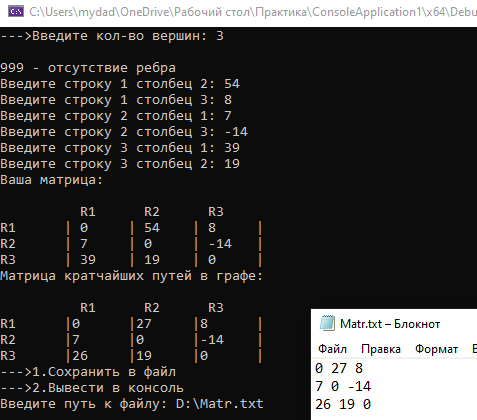


Рисунок 19 - Выгрузка результата работы в файл

# Приложение В

**Листинг программы.**

#include <iostream>

#include <locale>

#include <time.h>

#include <fstream>

#include <conio.h>

#include <string>

using namespace std;

int getIntegerFromUser() {

int value;

while (true) {

cin >> value;

if (cin.fail() || cin.peek() != '\n') { // провал чтения означает не тот тип данных или наличие лишних символов

cin.clear(); // Сброс состояния потока

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Игнорируем оставшийся ввод до перевода строки

cout << "Введено некорректное значение. Попробуйте еще раз: ";

}

else {

return value;

}

}

}

int\*\* FloydWarshall(const int SIZE, int\*\* a) {

int\*\* p = new int\* [SIZE];

// Инициализация матрицы предшествования

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

p[i] = new int[SIZE];

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

if (a[i][j] < 999 && i != j)

p[i][j] = i;

else

p[i][j] = -1;

}

}

// Модификация алгоритма Флойда-Уоршалла для отслеживания предшественников

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

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

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

if (a[i][k] < 999 && a[k][j] < 999 && a[i][j] > a[i][k] + a[k][j]) {

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

p[i][j] = p[k][j]; // Обновление предшествования

}

}

}

}

// Проверка на наличие отрицательных циклов

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

if (a[i][i] < 0) {

cout << "Обнаружен отрицательный цикл в вершине: "<< i + 1 << endl;

return nullptr;

}

}

// Вывод матрицы кратчайших путей

cout << "Матрица кратчайших путей в графе:" << endl;

cout << endl;

for (size\_t j = 0; j < SIZE; j++) cout << "\t" << " R" << j + 1;

cout << endl;

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

{

cout << "R" << i + 1 << "\t|";

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

cout << a[i][j] << "\t|";

}

cout << endl;

}

cout << "Маршруты между вершинами:" << endl;

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

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

if (i != j) {

cout <<"Кратчайший путь из " << i + 1 << " в " << j + 1 << ": ";

int u = j;

while (u != i) {

if (u == -1) {

cout << "Недостижимо";

break;

}

cout << u + 1 << " <- ";

u = p[i][u];

}

cout << i + 1 << endl;

}

}

}

return p;

}

int\*\* gen\_matrix(const int SIZE)

{

int ka = 80;

int kq = 0;

int\*\* a = new int\* [SIZE];

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

{

a[i] = new int[SIZE];

}

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

{

int co = 0;

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

{

if (!(i == j))

{

kq = rand() % 100;

if (kq < ka) {

a[i][j] = (rand() % 200);

}

else {

a[i][j] = 999;

co++;

}

if (co == SIZE - 1)

a[i][j] = (rand() % 200);

}

else {

a[i][j] = 0;

}

}

}

return a;

}

int\*\* write\_matrix(const int SIZE) {

int\*\* a = new int\* [SIZE];

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

a[i] = new int[SIZE];

}

cout << "999 - отсутствие ребра" << endl;

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

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

if (!(i == j)) {

cout << "Введите строку " << i + 1 << " столбец " << j + 1 << ": ";

// Убедимся, что сообщение было выведено

a[i][j] = getIntegerFromUser(); // Используем функцию для ввода числа

}

else {

a[i][j] = 0;

}

}

}

return a;

}

void output(const int SIZE, int\*\* a)

{

//вывод матрицы в консоль

cout << "Ваша матрица: " << endl;

cout << endl;

for (size\_t j = 0; j < SIZE; j++) cout << "\t" << " R" << j + 1;

cout << endl;

for (size\_t i = 0; i < SIZE; i++)

{

cout << "R" << i + 1;

for (size\_t j = 0; j < SIZE; j++)

{

cout << "\t| " << a[i][j];

}

cout << "\t|" << endl;

}

}

int\*\* get\_matrix(string path, int& SIZE)

{

int counter = 0;

ifstream fin;

fin.open(path);

if (!fin.is\_open())

{

cout << "\nОшибка открытия файла" << endl;

}

else

{

cout << "\nФайл открыт" << endl;

int temp;

while (fin >> temp)

{

counter++;

}

}

fin.close();

SIZE = sqrt(counter);

int\*\* A;

A = new int\* [SIZE];

for (size\_t i = 0; i < SIZE; i++)

{

A[i] = new int[SIZE];

}

fin.open(path);

if (!fin.is\_open())

{

cout << "\nОшибка отрытия файла" << endl;

}

else

{

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

{

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

{

fin >> A[i][j];

}

}

}

return A;

}

void file\_output(string path, const int SIZE, int\*\* a)

{

ofstream fout;

fout.open(path);

if (!fout.is\_open())

{

cout << "Ошибка отрытия файла" << endl;

}

else

{

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

{

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

{

fout << a[i][j] << " ";

}

fout << "\n";

}

}

fout.close();

}

int main()

{

setlocale(LC\_ALL, "rus");

int SIZE;

int b;

int\*\* A;

string path;

cout << "\t\t\tКурсовая работа\n\t\tПо дисциплине Л и ОА\n\t\tНа тему: \"Реализация алгоритма Флойда-Уоршелла поиска кратчайших путей в графе\"\n\n\tВыполнил студент группы 22ВВВ3: Гурьянов Данил\n\tПринял: Акифьев И.В";

getchar();

system("cls");

while (true) {

cout << "--->1. Загрузить из файла" << endl;

cout << "--->2. Записать через консоль" << endl;

cout << "--->3. Сгенерировать случайно" << endl;

cout << "--->0. Выход" << endl;

b = \_getch();

if (b == '0')

{

return 0;

}

if (b == '1')

{

system("cls");

cout << "--->Введите путь к файлу: "; cin >> path; cout << endl;

A = get\_matrix(path, SIZE);

output(SIZE, A);

FloydWarshall(SIZE, A);

cout << "--->1.Сохранить в файл" << endl;

cout << "--->2.Вывести в консоль" << endl;

b = \_getch();

if (b == '1')

{

cout << "Введите путь к файлу: "; cin >> path; cout << endl;

file\_output(path, SIZE, A);

}

if (b == '2')

{

output(SIZE, A);

}

}

if (b == '2')

{

system("cls");

cout << "\n--->Введите кол-во вершин: "; SIZE = getIntegerFromUser(); cout << endl;

A = write\_matrix(SIZE);

output(SIZE, A);

FloydWarshall(SIZE, A);

cout << "--->1.Сохранить в файл" << endl;

cout << "--->2.Вывести в консоль" << endl;

b = \_getch();

if (b == '1')

{

cout << "Введите путь к файлу: "; cin >> path; cout << endl;

file\_output(path, SIZE, A);

}

if (b == '2')

{

output(SIZE, A);

}

}

if (b == '3')

{

system("cls");

cout << "\n--->Введите кол-во вершин: "; SIZE = getIntegerFromUser(); cout << endl;

A = gen\_matrix(SIZE);

output(SIZE, A);

FloydWarshall(SIZE, A);

cout << "--->1.Сохранить в файл" << endl;

cout << "--->2.Вывести в консоль" << endl;

b = \_getch();

if (b == '1')

{

cout << "Введите путь к файлу: "; cin >> path; cout << endl;

file\_output(path, SIZE, A);

}

if (b == '2')

{

output(SIZE, A);

}

}

}

return 0;

}