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

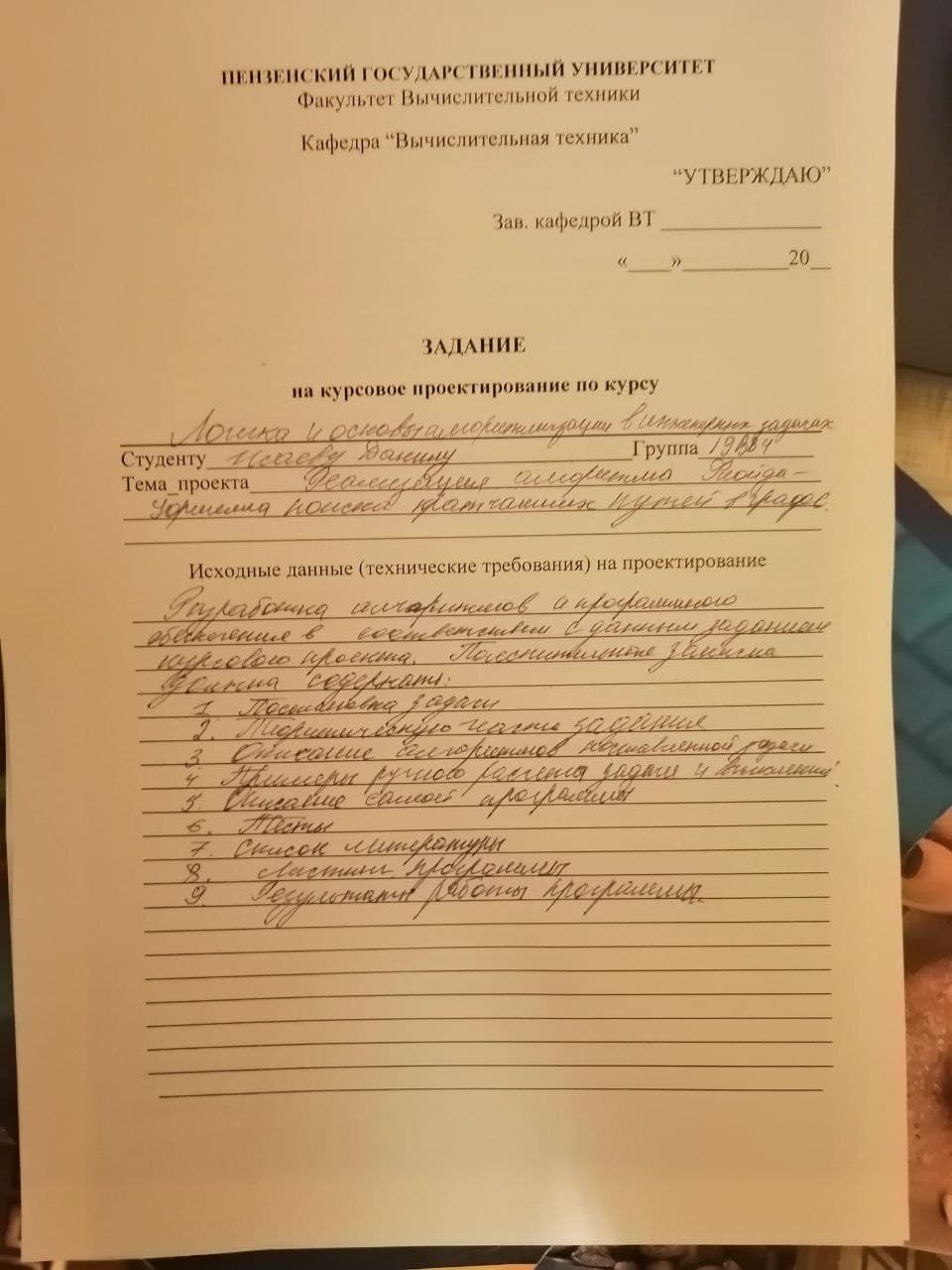
**Пояснительная записка**

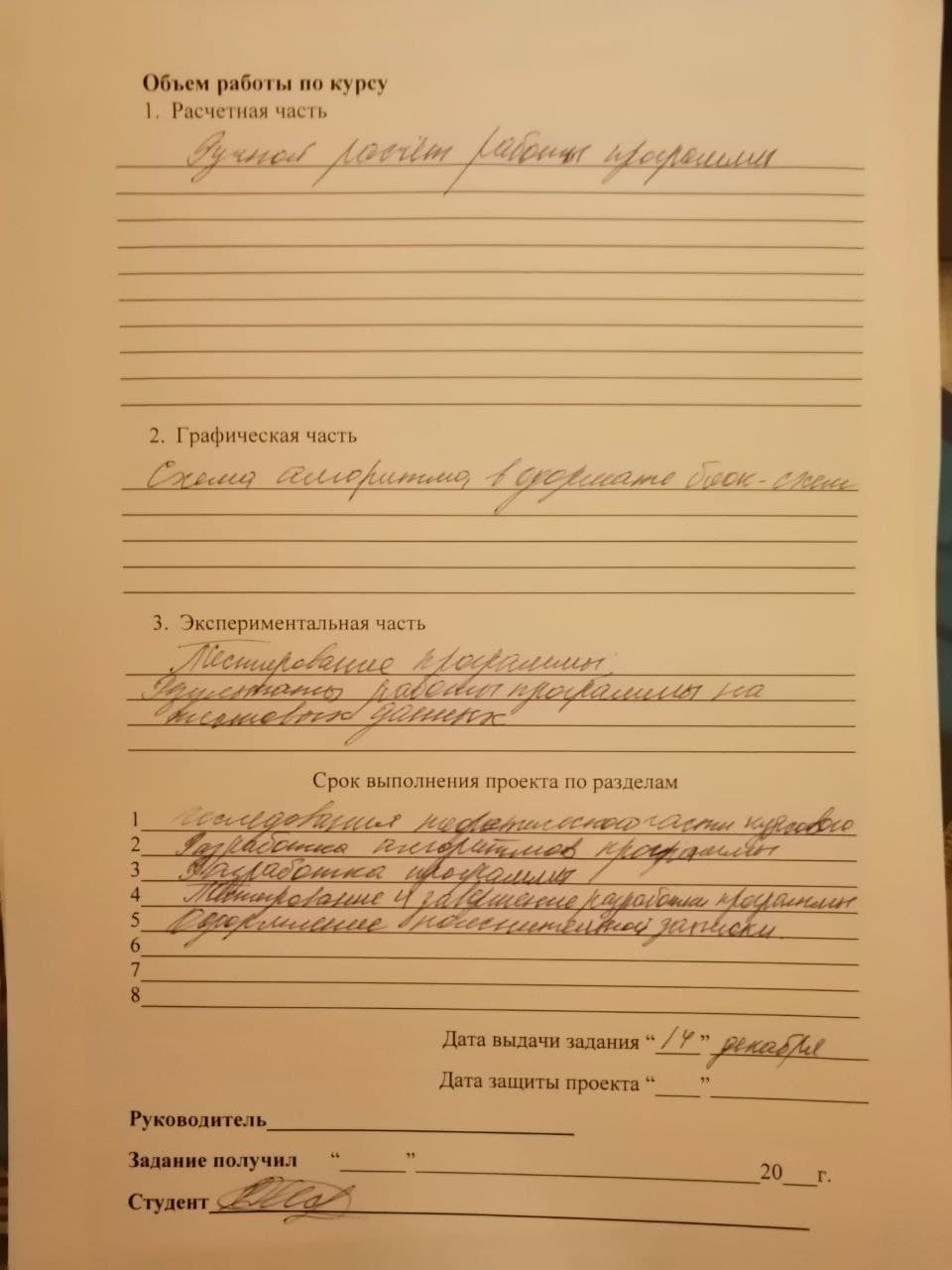
к курсовой работе  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Реализация алгоритма Флойда — Уоршелла поиска кратчайших путей в графе»

**Выполнил:**Студент группы 19ВВ4  
Исаев Данил

**Приняли:**Митрохин М.А.  
Юрова О.В.

Пенза 2020





# Содержание

[Содержание 4](#_Toc60002217)

[Реферат 5](#_Toc60002218)

[Введение 6](#_Toc60002219)

[1.Постановка задачи 7](#_Toc60002220)

[2.Теоретическая часть задания 8](#_Toc60002221)

[3.Описание алгоритма программы 9](#_Toc60002222)

[4.Описание программы 10](#_Toc60002223)

[5.Результат работы программы 15](#_Toc60002224)

[6.Тестирование 17](#_Toc60002225)

[7.Ручной расчёт задачи 20](#_Toc60002226)

[Заключение 22](#_Toc60002227)

[Список литературы. 23](#_Toc60002228)

[Приложение А. 24](#_Toc60002229)

# Реферат

Отчёт 26 стр, 16 рисунков.

Цель исследования – разработка программы, способная находить кратчайшее расстояния между вершинами графа, используя алгоритм Флойда-Уоршелла.

В работе рассмотрены принципы работы алгоритма Флойда-Уоршелла, нахождение кратчайших путей в взвешенном графе.

# Введение

Алгоритм Флойда поиска кратчайших путей между всеми парами вершин.

Граф — это совокупность множества вершин и множества пар вершин (связей между вершинами, дуг).

Алгоритм Флойда - Уоршелла - алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного графа без циклов с отрицательными весами с использованием метода динамического программирования.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2010, язык программирования – Си.

Целью данной курсовой работы является разработка программы на языке Си, именно с его помощью в данном курсовом проекте реализуется алгоритм Флорида – Уоршелла.

## Постановка задачи

Требуется разработать программу, которая будет находить кратчайшее расстояния между вершинами графа, используя алгоритм Флойда-Уоршелла.

Взвешенный граф в программе задается таблицей расстояний (расстояние 1000 – это очень большое расстояние, приближенное к бесконечности, оно значит, что связи между вершинами нет), причем в программе расстояния между вершинами генерируются, либо случайно, либо задаются с клавиатуры. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации таблицы расстояний с клавиатуры. После обработки этих данных на экран должна выводиться таблица, в которой отображаются кратчайшие расстояния между вершинами. Результат работы программы должен дополнительно выводиться в файл. Устройство ввода – клавиатура и мышь.

## Теоретическая часть задания

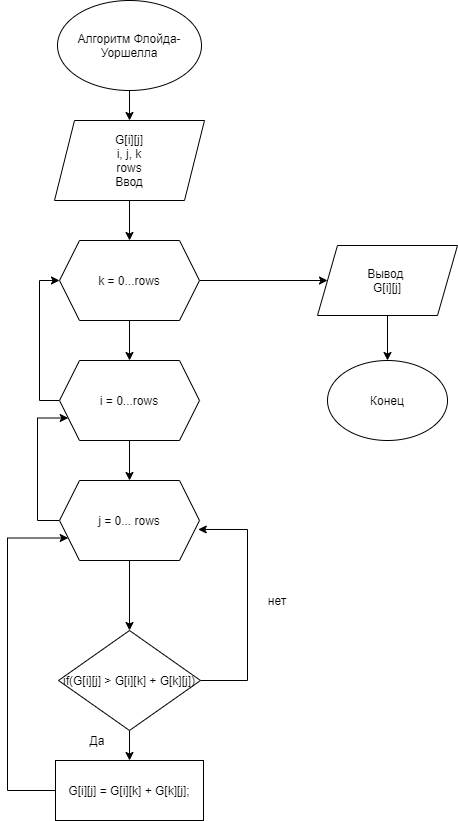
Имеем граф G=(V,E), в котором каждая вершина пронумерована от 1 до |V|. Сформируем матрицу смежности D. Эта матрица имеет размер |V|∗|V| и каждому ее элементу Dij присвоен вес ребра, соединяющего вершину i с вершиной j.

По мере выполнения алгоритма данная матрица будет перезаписываться: в каждую из ее ячеек внесется значение, определяющее оптимальную длину пути из вершины i в вершину j.

Полагаем диагональные элементы Dii равными нулю, а недиагональные элементы, соответствующие не инцидентным вершинам (не имеющим общего ребра), положим равными бесконечности или числу, заведомо большему возможного расстояния между ребрами.

Ключевая часть алгоритма состоит из трех циклов:  
Для k от 1 до |V| выполнять  
Для i от 1 до |V| выполнять  
Для j от 1 до |V| выполнять  
Если Dik+Dkj<Dij, то Dij:=Dik+Dkj.

## 3. Описание алгоритма программы



## 4. Описание программы

Для написания данной программы использован язык программирования Си.

Проект был создан в виде консольного приложения Win32.

Для данной программы нам понадобится массив G, в котором мы будем хранить расстояния между вершинами, а после обработки данного массива алгоритмом, он будет хранить в себя кратчайшие расстояния между вершинами.

В начале программа просит пользователя указать количество вершин в графе.



Рисунок 1 - Указание программе, сколько будет вершин у графа

Для генерации динамичного двумерного массива заданного размера используется:

srand(time(NULL));

G = (int\*\*)malloc(rows \* sizeof(int\*));

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

G[i] = (int\*)malloc(rows \* sizeof(int));

}

После нужно указать в какой файл нужно будет сохранить результаты (если файла не существует, программа создаст новый)

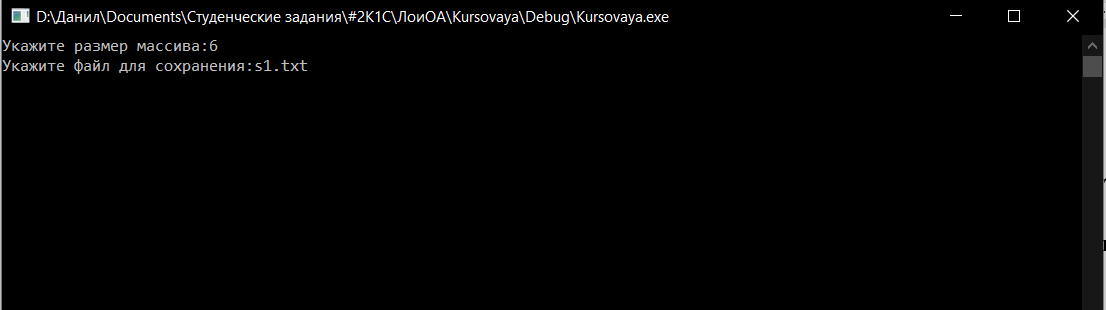


Рисунок 2 -Выбор файла сохранения результата

После появится меню выбора заполнения матрицы, оно создано с помощью menu();

void Menu(int rows, int\*\* G){

int menu;

int f = 0;

while(f != 1){

printf("Укажите как будет происходить заполнение:\n");

printf("1.Случайное\n");

printf("2.Ручное\n");

scanf("%d", &menu);

if(menu > 2 || menu < 1)

f = 0;

else

f = 1;

}

switch(menu){

case 1:

RandomInput(rows, G);//Случайное заполнение матрицы

break;

case 2:

ManualInput(rows, G);//Ручной ввод

break;

}

}

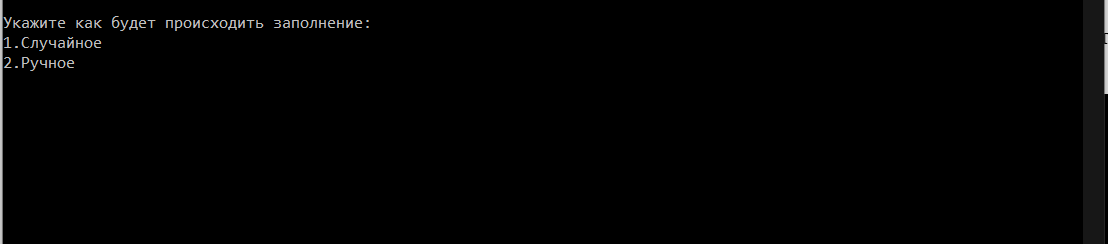


Рисунок 3 - Выбор способа заполнения

При выборе «1. Случайное» запускается функция RandonInput():

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

{

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

{

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

if (i == j)

G[i][j] = 0;

if(G[i][j] > 100)

G[i][j] = 1000;//очень большое число

G[j][i] = G[i][j];

}

}

}

G[i][j] = 1000; - значит, что между вершинами нет связей (1000 – взято, как число стремящееся к бесконечности)

При выборе «2. Ручное» запускается функция ManualInput():

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

{

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

{

if (i == j) {

G[i][j] = 0;

break;

} //Растояние от элемента до самого себя

printf("Введите растояние от %d до %d: ",i,j);

scanf("%d",&G[i][j]);

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

break;

G[j][i] = G[i][j];// Растояние от A до B = от B до A

}

}

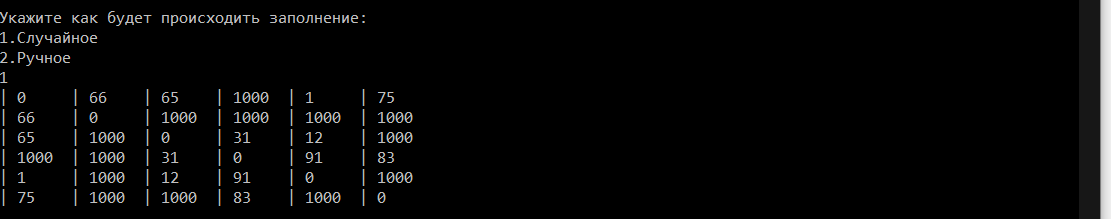


Рисунок 4 - Выбор заполнения массива псевдослучайными числами

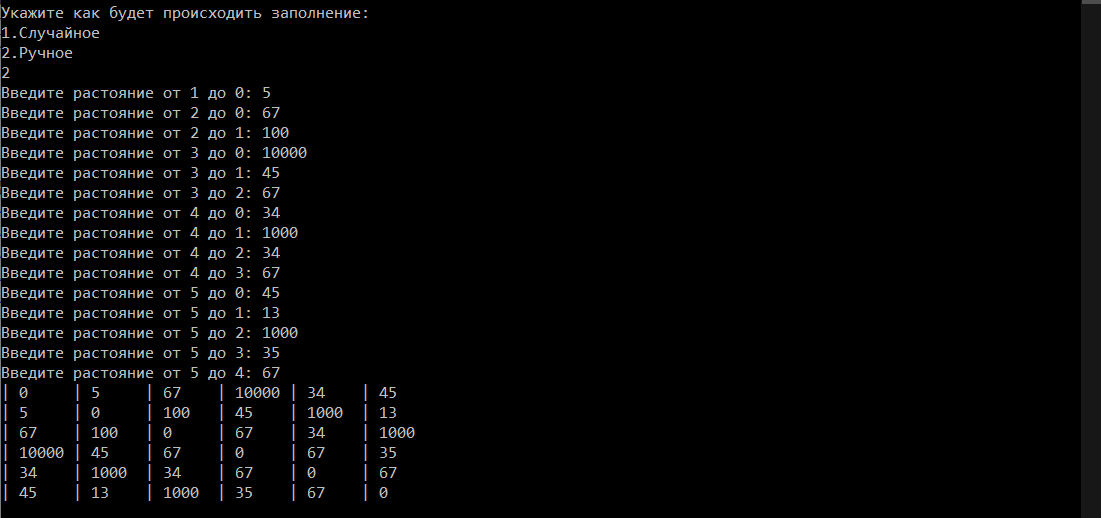


Рисунок 5 - Выбор ручного заполнения массива

После заполнения запускается алгоритм Флойда-Уоршелла:

//Алгоритм Флойда-Уоршалла

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

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

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

//Новое значение ребра равно минимальному между старым

//и суммой ребер i <-> k + k <-> j (если через k пройти быстрее)

if(G[i][j] > G[i][k] + G[k][j])

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

}

}

}

Данный алгоритм проверяет, если путь через соседнюю вершину и её ребра является короче, чем нынешнее ребро, то он заменят расстояние на более короткое, все это происходит путем перебора и сравнения величин.

После работы алгоритма выводится матрица кратчайший путей и данные сохраняются в файл.

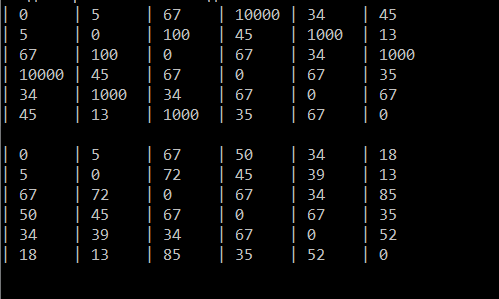


Рисунок 6 - Результат работы алгоритма

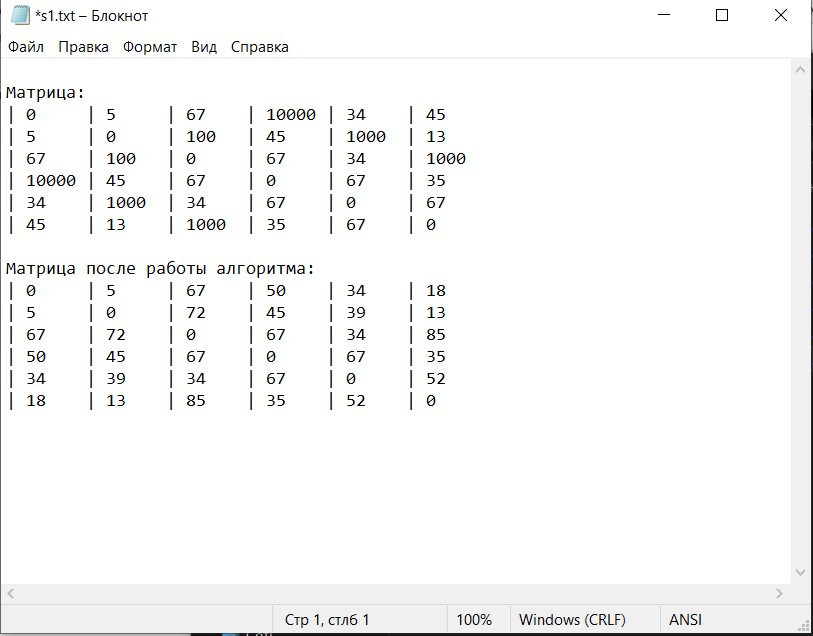


Рисунок 7 - Результат сохранен в файле

## 5. Результат работы программы

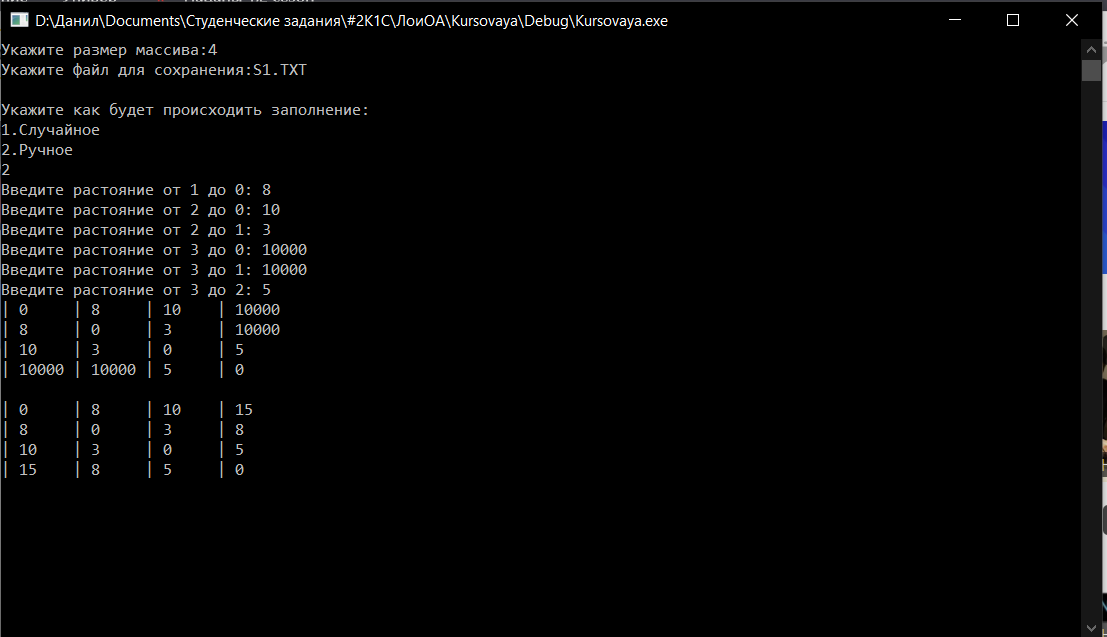


Рисунок 8 - Вывод результата работы программы

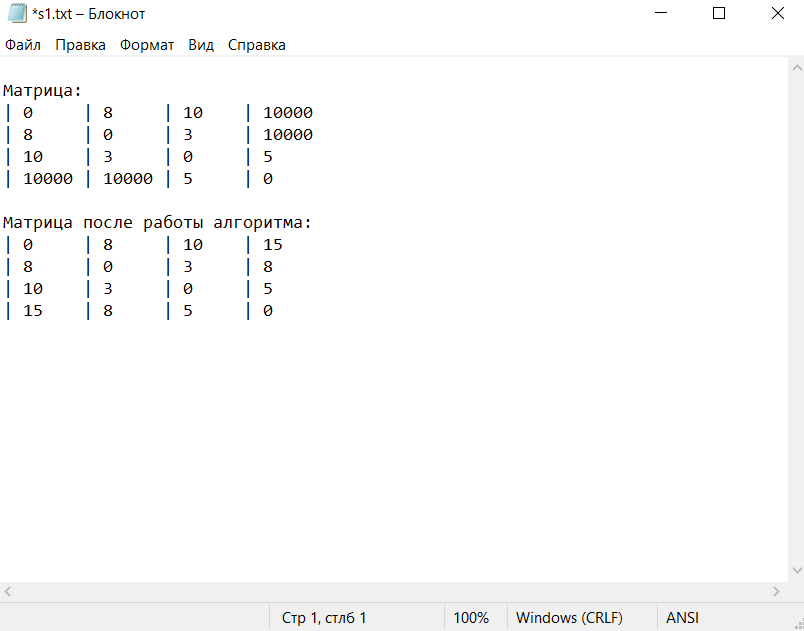


Рисунок 9 — Файл с результатом работы программы

## 6. Тестирование

Среда разработки MicrosoftVisualStudio 2010 предоставляет все средства, необходимые при разработке и отладке 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 и выводом данных, алгоритмами заполнения матриц, сохранением в файл. Ниже продемонстрирован результат тестирования программы:

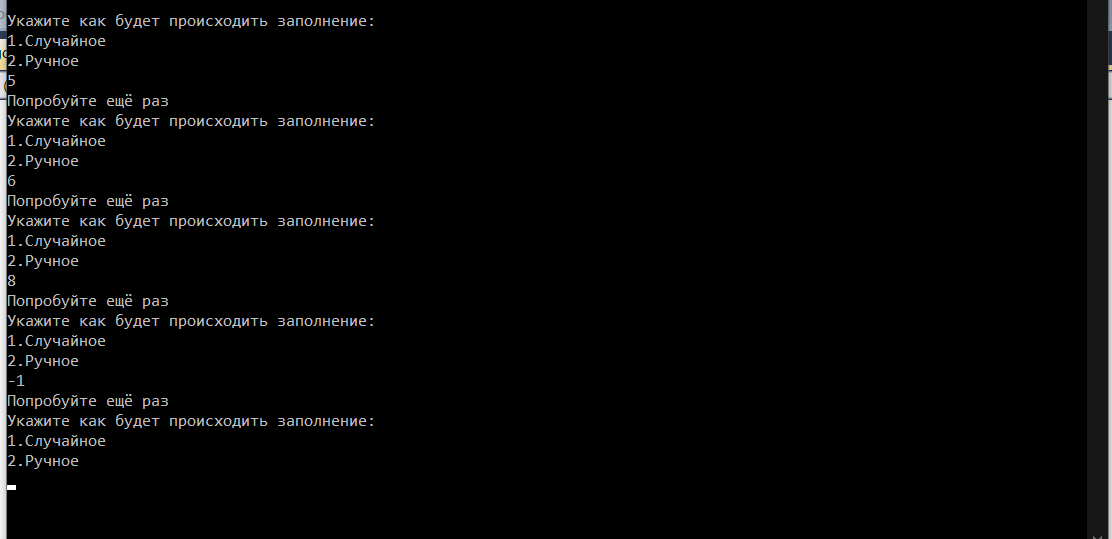


Рисунок 10 - Пользователь пытается ввести значение, которого нет в меню

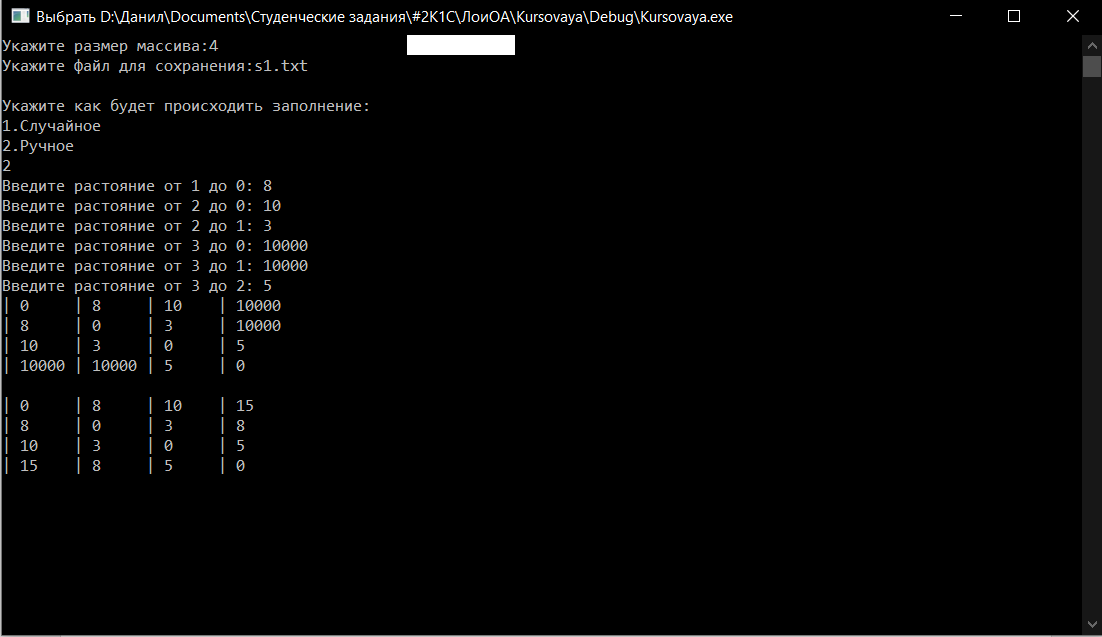


Рисунок 11 - Ручной ввод

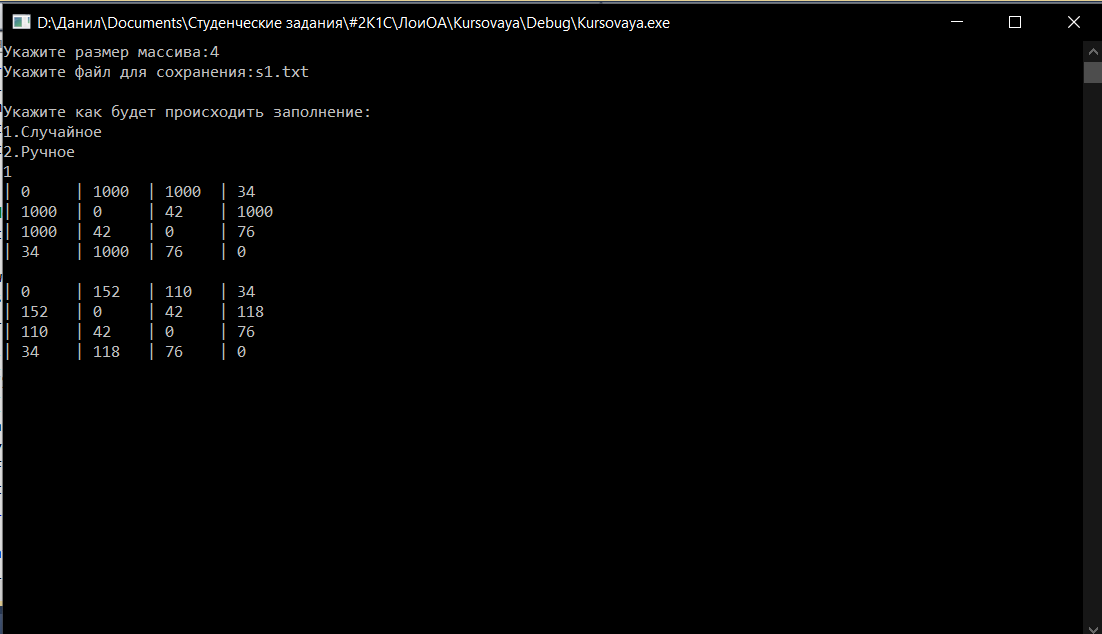


Рисунок 12 - Заполнение случайными числами

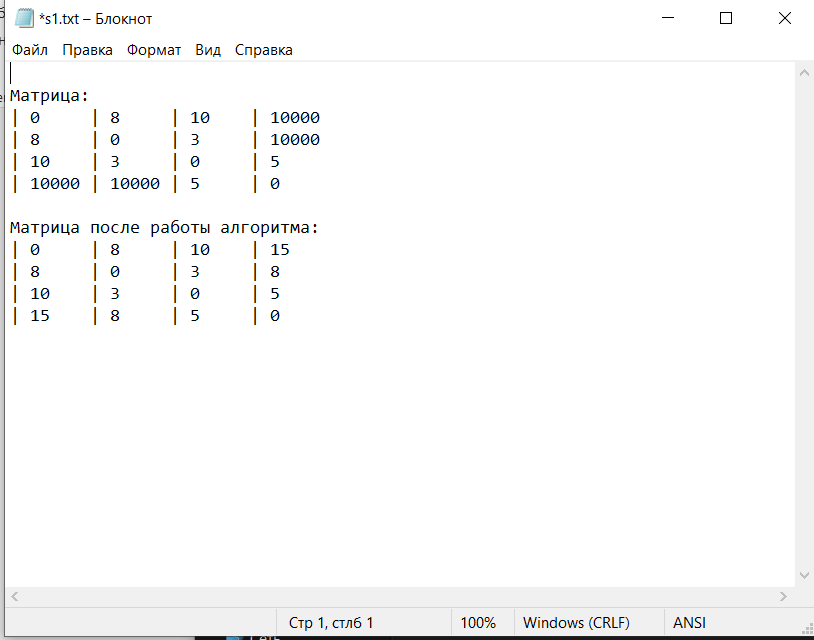


Рисунок 13 - Сохранение в файл при ручном вводе

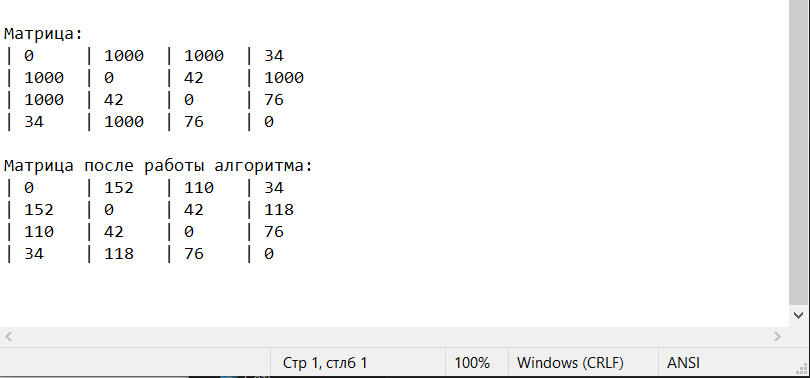


Рисунок 14 - Сохранение в файле при вводе случайных значений

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о выборе количества вершин | Верно |
| Неправильно введённые данные | Выводится сообщение о неправильном выводе данных и повторный запрос данных | Верно |
| Расчёт кратчайших расстояний | На выходе Получены корректные значения | Верно |
| Вывод матрицы с кратчайшими путями | Данные выводятся корректно. | Верно |
| Вывод результатов работы сохраняется в файл | В файле корректно отображаются исходные данные и результат работы алгоритма | Верно |

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

## 7. Ручной расчёт задачи

Возьмем граф, который мы заранее использовали в программе:

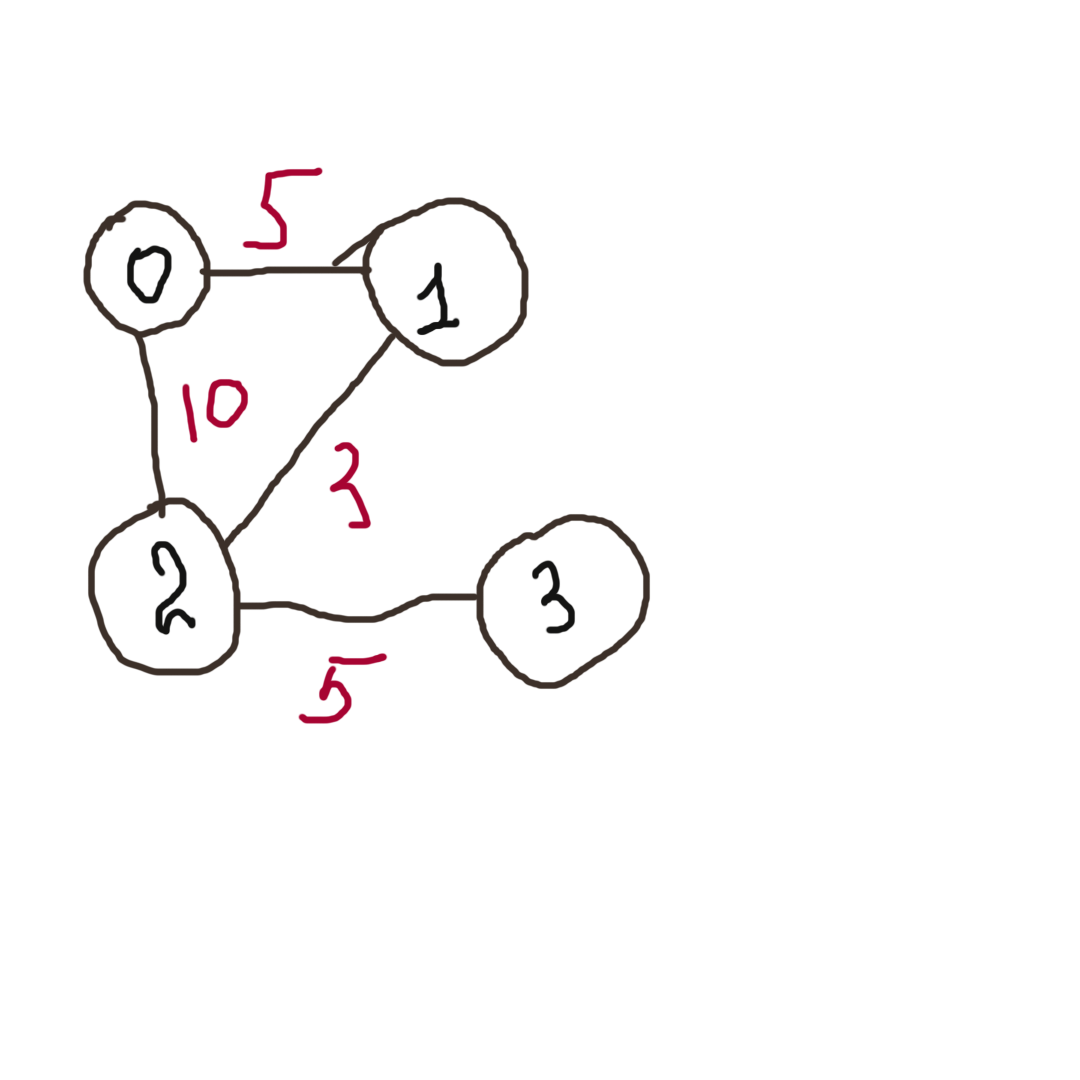


Рисунок 15 - Проверяемый граф

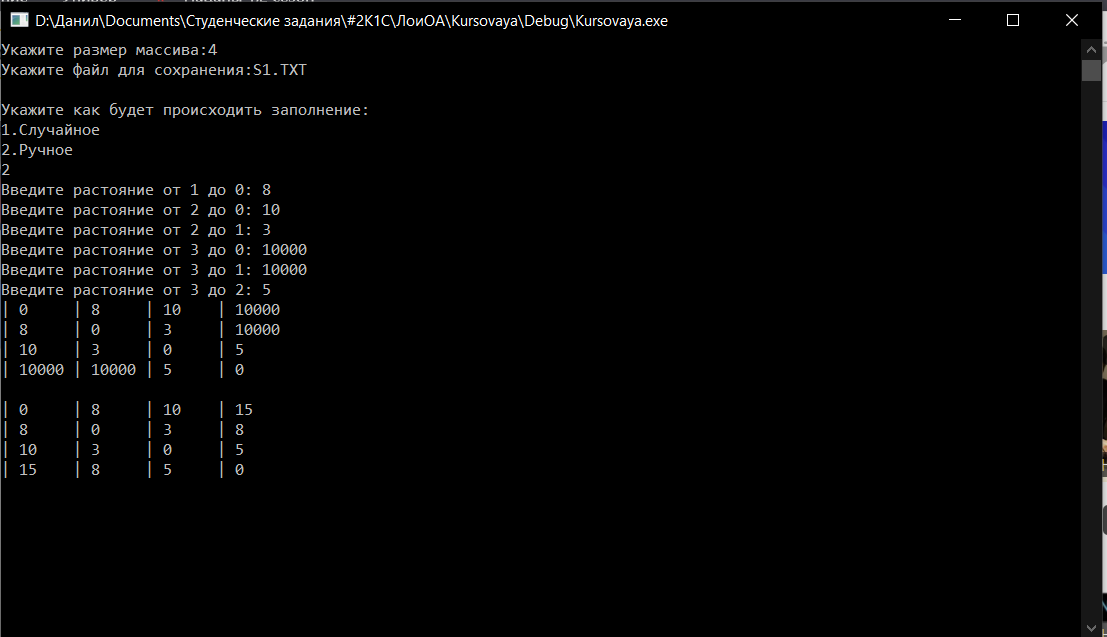


Рисунок 16 - Результат программы при работе с данным графом

По рисунку 15 мы видим, что от 0 вершины до 3 есть 2 пути:

1 путь через вершины 1 и 2 – 5+3+5 = 16

2 путь через вершину 2 – 10 + 5 = 15

Также мы видим, что от вершины 0 до вершины 2, есть путь меньше, через точку 1:

5 + 3 = 8 <10

Результаты работы алгоритма и ручной расчёт одинаковые, следует алгоритм работает - верно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Флойда-Уоршелла для поиска кратчайшего пути между вершинами графов в MicrosoftVisualStudio 2010.

Приобретены навыки: по осуществлению алгоритма Флойда-Уоршелла, углублены знания языка программирования Cи, были получены навыки разработки программ и освоены приёмы создания матриц смежностей.

# Список литературы.

<https://algowiki-project.org/ru/Алгоритм_Флойда-Уоршелла#.D0.9E.D0.B1.D1.89.D0.B5.D0.B5_.D0.BE.D0.BF.D0.B8.D1.81.D0.B0.D0.BD.D0.B8.D0.B5_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B0>

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

*Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн.* **Алгоритмы: построение и анализ**

*Белоусов А. И., Ткачев С. Б.* Дискретная математика.

<https://ru.wikipedia.org/wiki/Алгоритм_Флойда_—_Уоршелла>

# Приложение А.

Листинг программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

#include <queue>

#include <locale.h>

//Глобальные переменные

using namespace std;

int\*\* G;

//Объявление функций

void Menu(int rows, int\*\* G);

void ManualInput(int rows, int\*\* G);

void RandomInput(int rows, int\*\* G);

void OutputMatrix(int rows, int\*\* G);

void OutputMatrixFile(int rows, int\*\* G,FILE \*save);

int main()

{

//Подключение кириллицы

setlocale(LC\_ALL, "Rus");

int rows; //размер матрицы

FILE \*save;

char filename[16];

printf("Укажите размер массива:");

scanf("%d", &rows);

printf("Укажите файл для сохранения:");//открытие файла сохранения

scanf("%s",&filename);

save = fopen(filename,"a");

printf("\n");

srand(time(NULL));

G = (int\*\*)malloc(rows \* sizeof(int\*));

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

G[i] = (int\*)malloc(rows \* sizeof(int));

}

Menu(rows, G);//меню выбора ввода

OutputMatrix(rows, G);//Вывод матрицы

fprintf(save ,"Матрица:\n");

OutputMatrixFile(rows, G, save);

//Алгоритм Флойда-Уоршалла

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

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

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

//Новое значение ребра равно минимальному между старым

//и суммой ребер i <-> k + k <-> j (если через k пройти быстрее)

if(G[i][j] > G[i][k] + G[k][j])

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

}

}

}

OutputMatrix(rows, G);// ВЫВОД МАТРИЦЫ после обработки алгоритмом

fprintf(save ,"Матрица после работы алгоритма:\n");

OutputMatrixFile(rows, G, save);

//Конец программы

getch();

return 0;

}

void ManualInput(int rows, int\*\* G){

// Ручной ввод

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

{

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

{

if (i == j) {

G[i][j] = 0;

break;

} //Растояние от элемента до самого себя

printf("Введите растояние от %d до %d: ",i,j);

scanf("%d",&G[i][j]);

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

break;

G[j][i] = G[i][j];// Растояние от A до B = от B до A

}

}

}

void RandomInput(int rows, int\*\* G){

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

{

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

{

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

if (i == j)

G[i][j] = 0;

if(G[i][j] > 100)

G[i][j] = 1000;//очень большое число

G[j][i] = G[i][j];

}

}

}

void Menu(int rows, int\*\* G){

int menu;

int f = 0;

while(f != 1){

printf("Укажите как будет происходить заполнение:\n");

printf("1.Случайное\n");

printf("2.Ручное\n");

scanf("%d", &menu);

if(menu > 2 || menu < 1){

f = 0;

printf("Попробуйте ещё раз\n");

}

else

f = 1;

}

switch(menu){

case 1:

RandomInput(rows, G);//Случайное заполнение матрицы

break;

case 2:

ManualInput(rows, G);//Ручной ввод

break;

}

}

void OutputMatrix(int rows, int\*\* G){

//вывод матрицы

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

{

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

{

printf("| %d\t", G[i][j]);

}

printf("\n");

}

printf("\n");

}

void OutputMatrixFile(int rows, int\*\* G,FILE \*save){

//вывод матрицы

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

{

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

{

fprintf(save ,"| %d\t", G[i][j]);

}

fprintf(save,"\n");

}

fprintf(save, "\n");

}