

**Содержание**

[Реферат 5](#_Toc122073280)

[Введение 6](#_Toc122073281)

[1. Постановка задачи 7](#_Toc122073282)

[2. Теоретическая часть задания 8](#_Toc122073283)

[3. Описание алгоритма программы 12](#_Toc122073284)

[4. Описание программы 13](#_Toc122073285)

[5. Тестирование 18](#_Toc122073286)

[6. Ручной расчет задачи 22](#_Toc122073287)

[Заключение 24](#_Toc122073288)

[Список литературы 25](#_Toc122073289)

[Приложение А. Листинг программы 26](#_Toc122073290)

# **Реферат**

Отчет 27 страниц, 9 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ФЛОЙДА

Цель исследования – разработка программы, способная находить наименьшее расстояние последовательным перебором всех путей, то есть реализация алгоритма Флойда.

В работе рассмотрены правила перебора всех путей между всеми вершинами взвешенного ориентированного графа. Рассмотрена суть алгоритма Флойда.

# **Введение**

Алгоритм Флойда (алгоритм Флойда–Уоршелла) — алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе. Работает корректно, если в графе нет циклов отрицательной величины. Он представляет собой простой перебор всех путей и выбор из них наименьшего.

В качестве среды разработки мною была выбрана среда разработки Microsoft Visual Studio 2019, язык программирования Си.

Целью данной курсовой работы является разработка программы на языке Си, в которой реализуется поиск наименьшего расстояния последовательным перебором всех путей, то есть алгоритм Флойда

# **Постановка задачи**

Необходимо осуществить программную реализацию алгоритма поиска кратчайших расстояний между всеми вершинами взвешенного ориентированного графа (Алгоритм Флойда).

Программа должна работать так, чтобы пользователь мог самостоятельно выбрать порядок случайно созданного ориентированного графа. После обработки этих данных на экран должна выводиться матрица весов ребер графа и матрица кратчайших путей, затем будет возможность сохранить полученные результаты в файл. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Для выбора задания матрицы реализуется текстовое меню. Граф задается матрицей смежности либо с клавиатуры (пользователь сам вводит значения матрицы), либо считывается из файла, либо задается случайно.

Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

# **Теоретическая часть задания**

Граф — это абстрактное представление множества объектов и связей между ними. Графом называют пару (V, E) где V это множество вершин, а E множество пар, каждая из которых представляет собой связь (эти пары называют рёбрами).

Ориентированный граф — граф, рёбрам которого присвоено направление. Орграф, полученный из простого графа ориентацией ребер, называется направленным. В произвольном простом орграфе две вершины могут соединяться двумя разнонаправленными дугами.

Изображение выглядит как текст, часы, коллекция картинок

Автоматически созданное описание

Рисунок 1-Орграф

Алгоритм Флойда – Уоршелла.

Наиболее часто используемое название, метод получил в честь двух американских исследователей Роберта Флойда и Стивена Уоршелла, одновременно открывших его в 1962 году. Реже встречаются другие варианты наименований: алгоритм Рой – Уоршелла или алгоритм Рой – Флойда. Рой – фамилия профессора, который разработал аналогичный алгоритм на 3 года раньше коллег (в 1959 г.), но это его открытие осталось безвестным.

Алгоритм Флойда – Уоршелла – динамический алгоритм вычисления значений кратчайших путей для каждой из вершин графа. Метод работает на взвешенных графах, с положительными и отрицательными весами ребер, но без отрицательных циклов, являясь, таким образом, более общим в сравнении с алгоритмом Дейкстры, т. к. последний не работает с отрицательными весами ребер, и к тому же классическая его реализация подразумевает определение оптимальных расстояний от одной вершины до всех остальных. Перебор осуществляется по так называемой «матрице смежности» размера n x n, где n — количество вершин графа. Ключевая идея алгоритма — разбиение процесса поиска кратчайших путей на фазы.

На пересечении i-ой строки и j-го столбца матрицы стоит значение веса ребра из вершины i в вершину j. Главная диагональ матрицы смежности — это всегда нули, потому что ребер из i-ой вершины в i-ую в графе быть не должно.

Перед k-ой фазой (k = 1 \ldots n) считается, что в матрице расстояний d[][] сохранены длины таких кратчайших путей, которые содержат в качестве внутренних вершин только вершины из множества \{ 1, 2, \ldots, k-1 \} (вершины графа мы нумеруем, начиная с единицы).

Иными словами, перед k-ой фазой величина d[i][j] равна длине кратчайшего пути из вершины i в вершину j, если этому пути разрешается заходить только в вершины с номерами, меньшими k (начало и конец пути не считаются).

Легко убедиться, что чтобы это свойство выполнилось для первой фазы, достаточно в матрицу расстояний d[][] записать матрицу смежности графа: d[i][j] = g[i][j] — стоимости ребра из вершины i в вершину j. При этом, если между какими-то вершинами ребра нет, то записать следует величину "бесконечность" \infty. Из вершины в саму себя всегда следует записывать величину 0, это критично для алгоритма.

Пусть теперь мы находимся на k-ой фазе, и хотим пересчитать матрицу d[][] таким образом, чтобы она соответствовала требованиям уже для k+1-ой фазы. Зафиксируем какие-то вершины i и j. У нас возникает два принципиально разных случая:

* Кратчайший путь из вершины i в вершину j, которому разрешено дополнительно проходить через вершины \{ 1, 2, \ldots, k \}, совпадает с кратчайшим путём, которому разрешено проходить через вершины множества \{ 1, 2, \ldots, k-1 \}.

В этом случае величина d[i][j] не изменится при переходе с k-ой на k+1-ую фазу.

* "Новый" кратчайший путь стал лучше "старого" пути.

Это означает, что "новый" кратчайший путь проходит через вершину k. Сразу отметим, что мы не потеряем общности, рассматривая далее только простые пути (т.е. пути, не проходящие по какой-то вершине дважды).

Тогда заметим, что если мы разобьём этот "новый" путь вершиной k на две половинки (одна идущая i \Rightarrow k, а другая — k \Rightarrow j), то каждая из этих половинок уже не заходит в вершину k. Но тогда получается, что длина каждой из этих половинок была посчитана ещё на k-1-ой фазе или ещё раньше, и нам достаточно взять просто сумму d[i][k] + d[k][j], она и даст длину "нового" кратчайшего пути.

Объединяя эти два случая, получаем, что на k-ой фазе требуется пересчитать длины кратчайших путей между всеми парами вершин i и j следующим образом:

new\_d[i][j] = min (d[i][j], d[i][k] + d[k][j]);

Таким образом, вся работа, которую требуется произвести на k-ой фазе — это перебрать все пары вершин и пересчитать длину кратчайшего пути между ними. В результате после выполнения n-ой фазы в матрице расстояний d[i][j] будет записана длина кратчайшего пути между i и j, либо \infty, если пути между этими вершинами не существует.

Последнее замечание, которое следует сделать, — то, что можно не создавать отдельную матрицу \rm new\_d[][] для временной матрицы кратчайших путей на k-ой фазе: все изменения можно делать сразу в матрице d[][]. В самом деле, если мы улучшили (уменьшили) какое-то значение в матрице расстояний, мы не могли ухудшить тем самым длину кратчайшего пути для каких-то других пар вершин, обработанных позднее.

# **Описание алгоритма программы**

На вход программе подаётся граф, заданный в виде матрицы смежности — двумерного массива d[][] размера n \times n, в котором каждый элемент задаёт длину ребра между соответствующими вершинами.

Требуется, чтобы выполнялось d[i][i] = 0 для любых i.

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

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

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

d[i][j] = min (d[i][j], d[i][k] + d[k][j]);

Предполагается, что если между двумя какими-то вершинами **нет ребра**, то в матрице смежности было записано какое-то большое число (достаточно большое, чтобы оно было больше длины любого пути в этом графе); тогда это ребро всегда будет невыгодно брать, и алгоритм сработает правильно.

# **Описание программы**

Для написания данной программы использован язык программирования Си. Данная программа является многомодульной, поскольку состоит из нескольких функций: main, orgraf, printM, FU, Save1/2, vvod, vvesti, Menu.

Работа программы начинается с запроса о выборе инициализации матрицы.

int main()

{

setlocale(LC\_ALL, "Rus");

do {

Menu();

vari = get\_vari(5);

switch (vari) {

case 1:

printf("Алгоритм Флойда \n");

printf("\n");

printf("Создание матрицы динамически \n");

orgraf(GR);

std::cout << "Матрица весов ребер:\n";

printM(GR);

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

FU(GR, n);

printM(D);

break;

case 2:

printf("Соранить? Ваш ответ (1 - да, 2 - нет):");

std::cin >> variant;

if (variant != 1 && variant != 2) {

printf("Введен недопустимый символ. Введите ваш ответ: ");

std::cin >> variant;

}

else {

if (variant == 1) {

Save1(GR);

Save2(D);

std::cout << "Сохранение выполнено успешно.\n";

}

break;

}

case 3:

printf("Алгоритм Флойда \n");

printf("\n");

printf("Создание матрицы динамически \n");

vvod(GR);

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

FU(GR,n);

printM(D);

break;

case 4:

printf("Алгоритм Флойда \n");

printf("\n");

printf("В файле должна быть информация о порядке орграфа, ниже о матрице смежности!!Пример ввода: privet.txt\nВведите название файла: ");

vvesti(GR);

std::cout << "Матрица весов ребер:\n";

printM(GR);

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

FU(GR,n);

printM(D);

break;

}

system("pause");

system("cls");

} while (vari != 5);

//Clear(root);

return 0;

}

void Menu() {

system("cls");

printf("Что вы хотите сделать?\n");

printf("1. Начать алгоритм\n");

printf("2. Сохранить в файл\n");

printf("3. Ввести матрицу самостоятельно\n");

printf("4. Считать с файла\n");

printf("5. Выход\n");

printf(">");

}

Если пользователь выбрал начать алгоритм(случайно сгенерированная матрица), то будет предложено ввести количество порядок орграфа.

void orgraf(int R[][maxV]) {

printf("Введите порядок орграфа:");

std::cin >> n;

if (n < 0)

{

printf("\n Введен недопустимый символ\n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

if (n > 50)

{

printf("\n Введенное число превышающее допустимое значение(50) \n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

srand(time(NULL));

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

{

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

{

R[i][j] = rand() % 10;

}

}

}

Если пользователь выбрал ввести матрицу самостоятельно, то будет предложено ввести порядок орграфа, а далее вводить саму матрицу.

void vvod(int R[][maxV])

{

printf("Введите порядок орграфа:");

std::cin >> n;

if (n < 0)

{

printf("\n Введен недопустимый символ\n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

if (n > 50)

{

printf("\n Введенное число превышающее допустимое значение(50) \n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

printf("\nВведите матрицу смежности:\n");

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

R[i][j] = (int)malloc(n \* sizeof(int)); }

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

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

scanf\_s("%d", &R[i][j]);

}

}

}

Если пользователь выбрал считать с файла, то необходимо, чтобы существовал текстовый файл формата txt, в нем содержалось количество вершин графа, ниже матрица смежности.

void vvesti(int R[][maxV]) {

char vvesti[50];

FILE\* fil;

scanf("%s", &vvesti);

fil = fopen(vvesti, "r");

if (fil == NULL)

{

printf("Не удалось открыть файл!\n");

return;

}

fscanf(fil, "%d", &n);

R[i][j] = (int)malloc(n \* sizeof(int));

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

R[i][j] = (int)malloc(sizeof(int) \* (n));

}

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

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

{

fscanf(fil, "%d", &R[i][j]);

}

}

fclose(fil);

return;

}

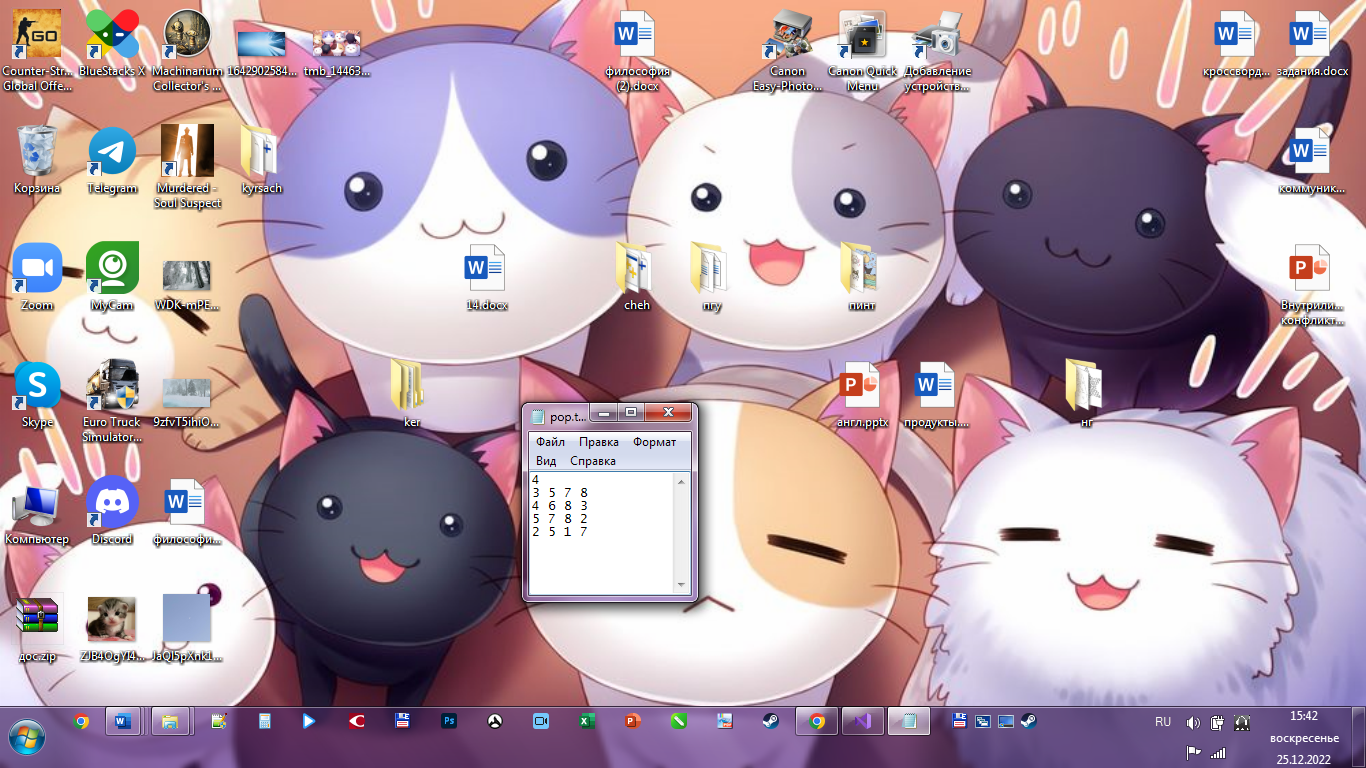


Рисунок 2- Пример файла с матрицей смежности

Если пользователь выбрал сохранить файл, то нет необходимости создавать файл для данной операции, он создается сам. В функции Save1 происходит сохранение матрицы весов ребер. В функции Save2 происходит сохранение матрицы кратчайших путей.

void Save1(int R[][maxV]) {

FILE\* fl;

fl = fopen("file.txt", "w");

if ((fl = fopen("file.txt", "w")) == NULL)

printf("Невозможно открыть/создать для записи файл 'file.txt'\n");

else

{

fprintf(fl, "Матрица весов ребер: \n");

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

{

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

{

fprintf(fl, " %d ", R[i][j]);

}

fprintf(fl, "\n ");

}

}

fclose(fl);

}

void Save2(int R[][maxV]) {

FILE\* fl;

fl = fopen("file.txt", "a");

if ((fl = fopen("file.txt", "a")) == NULL)

printf("Невозможно открыть/создать для записи файл 'file.txt'\n");

else

{

fprintf(fl, "Матрица кратчайших путей: \n");

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

{

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

{

fprintf(fl, " %d ", R[i][j]);

}

fprintf(fl, "\n ");

}

}

fclose(fl);

}

# **Тестирование**

Средой разработки была выбрана программа Visual Studio 2019. Программа содержит в себе все необходимые средства для успешной отладки и тестирования модулей программы, а также всей программы в целом. Для написания данной программы использован язык программирования Си. Данный язык программирования очень известен своей эффективностью, экономичностью ресурсов и переносимостью на других программные платформы. Тестирование позволило выявить множество проблем и недочетов по части внешнего вида матрицы, соответствия пунктов меню функциям.

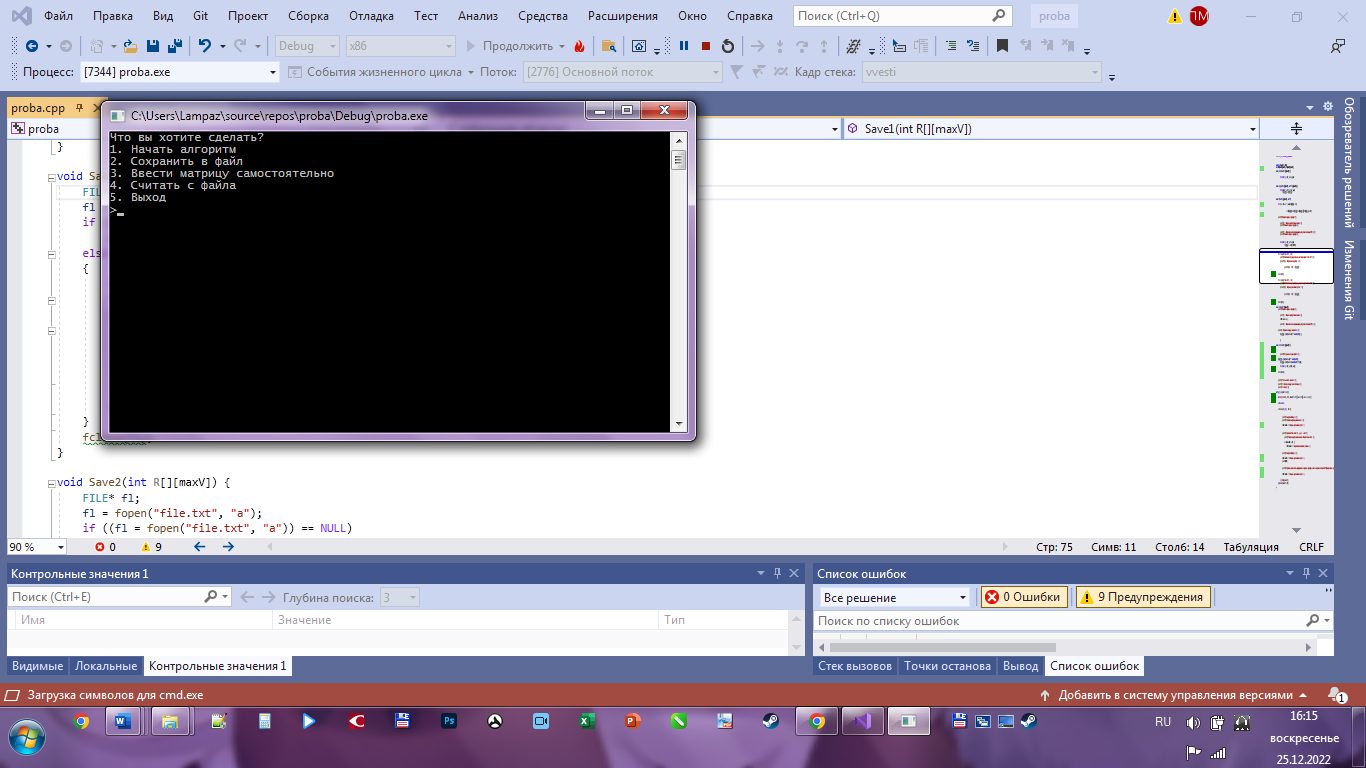


Рисунок 3- Меню

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных данных.

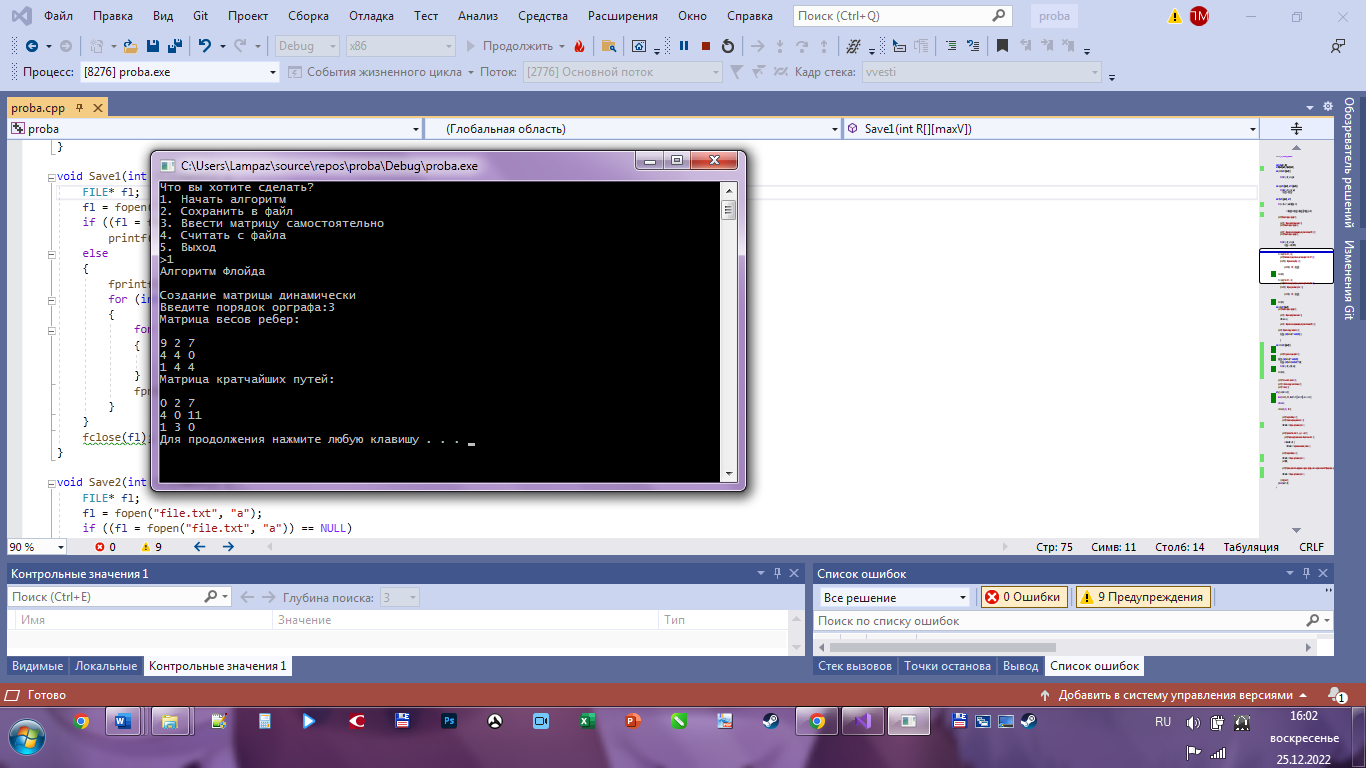


Рисунок 4- Тестирование при случайной генерации данных

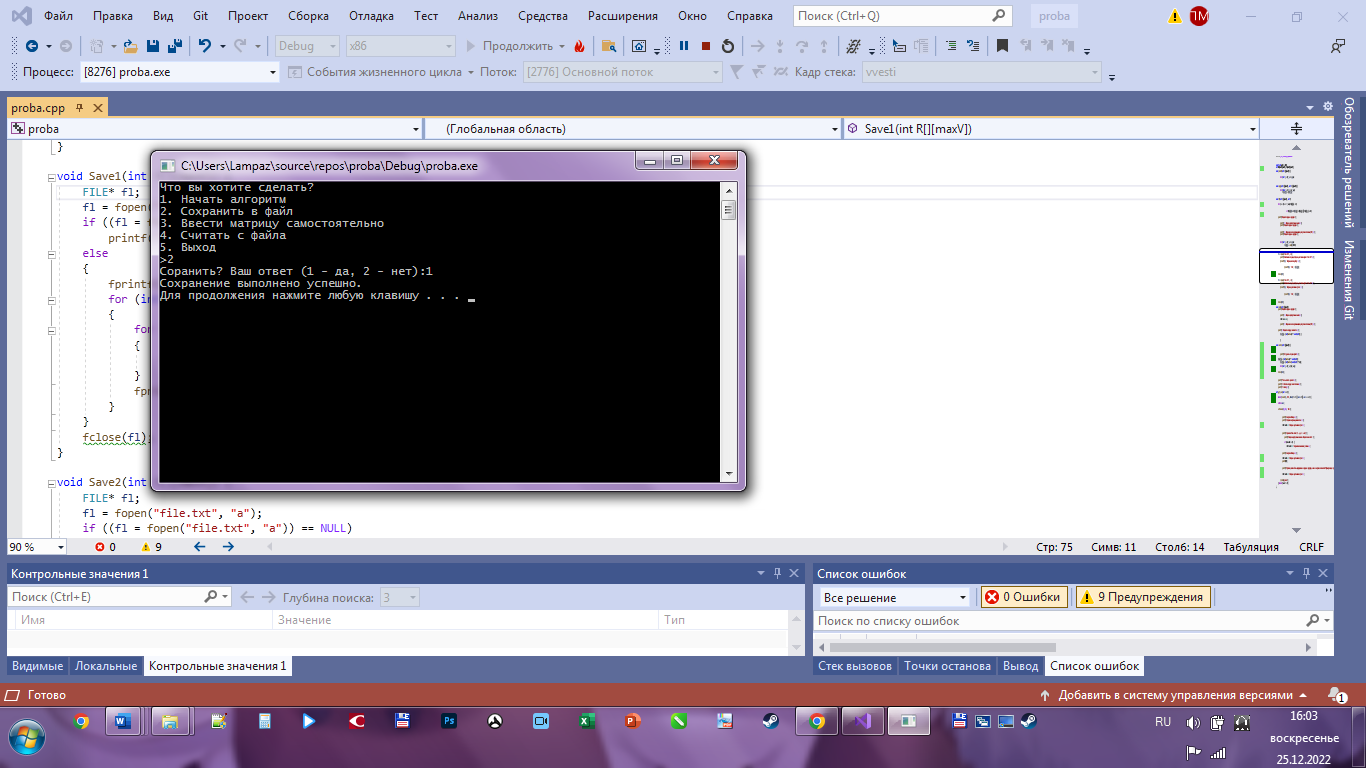


Рисунок 5- Тестирование при сохранении данных в файл

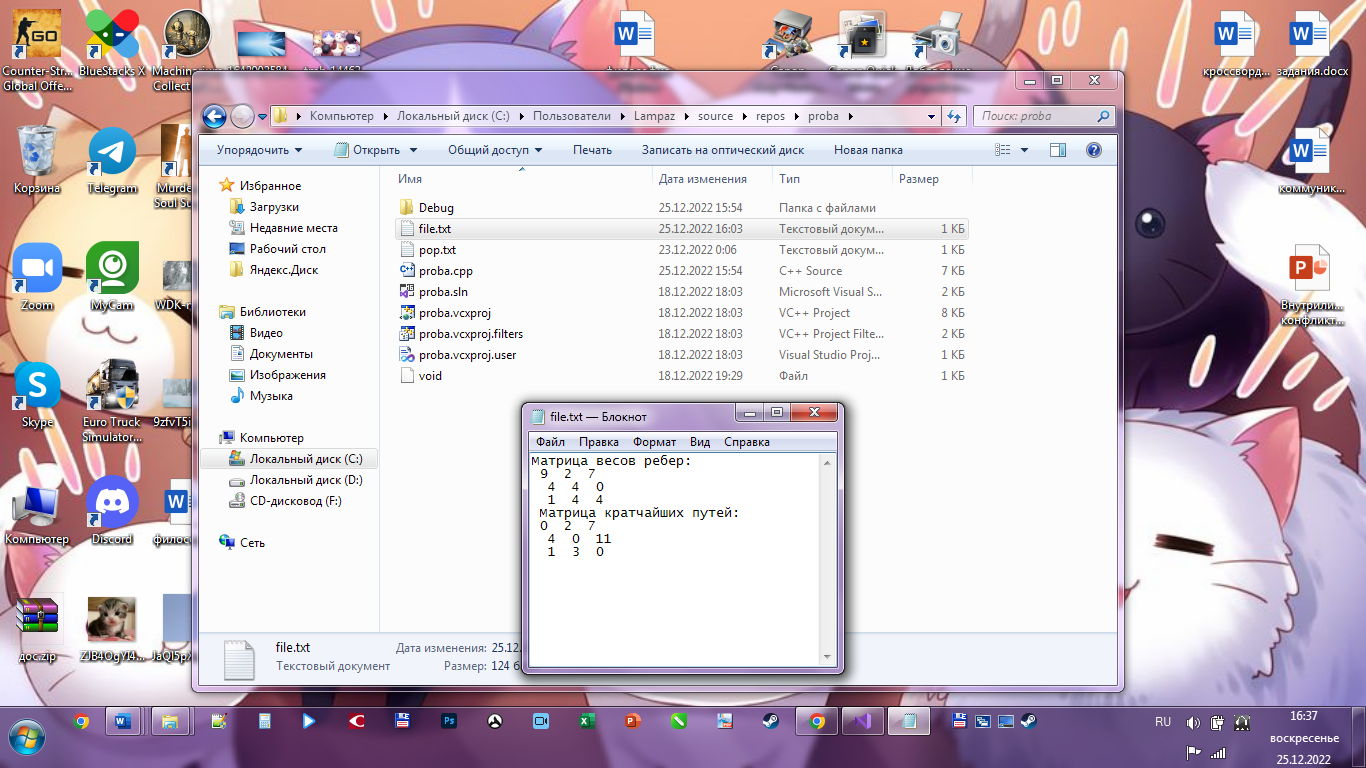


Рисунок 6- Записанный результат в файл

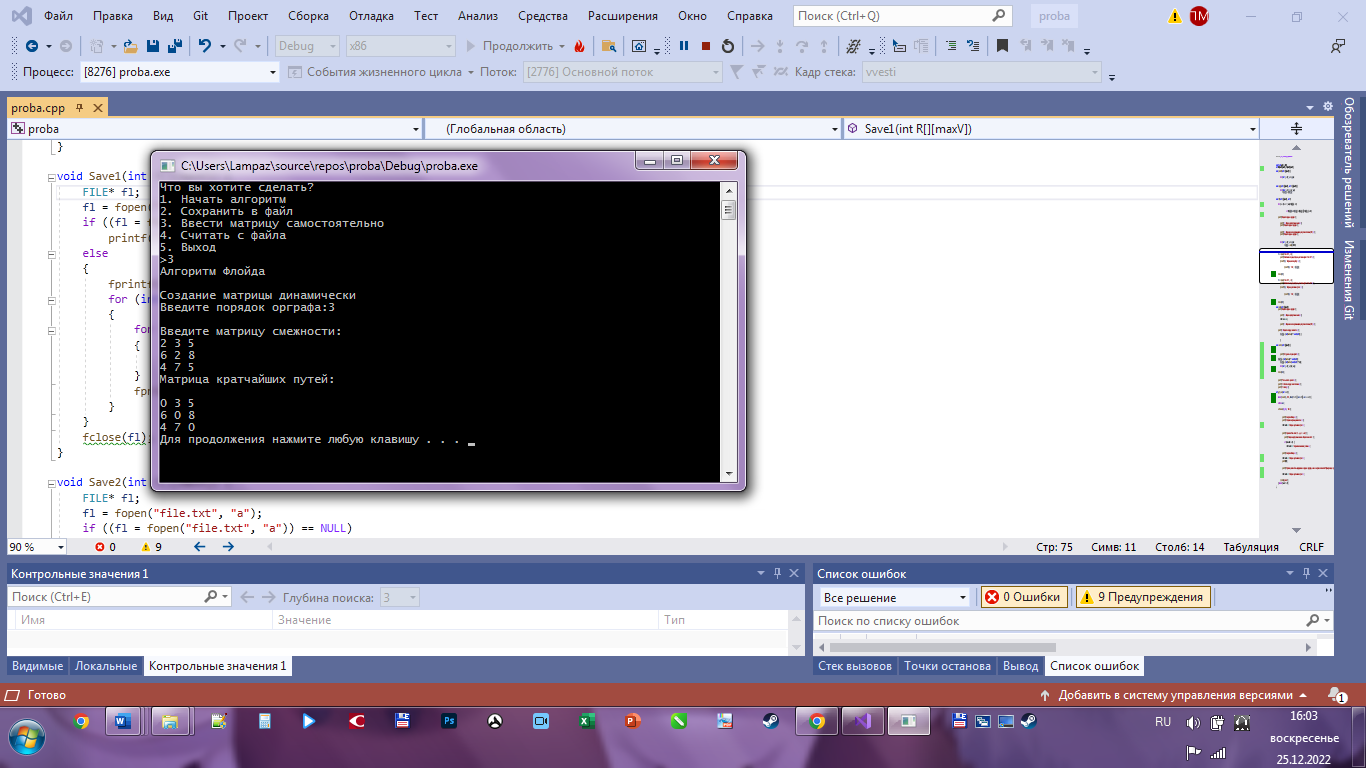


Рисунок 7-Тестирование при ручном вводе

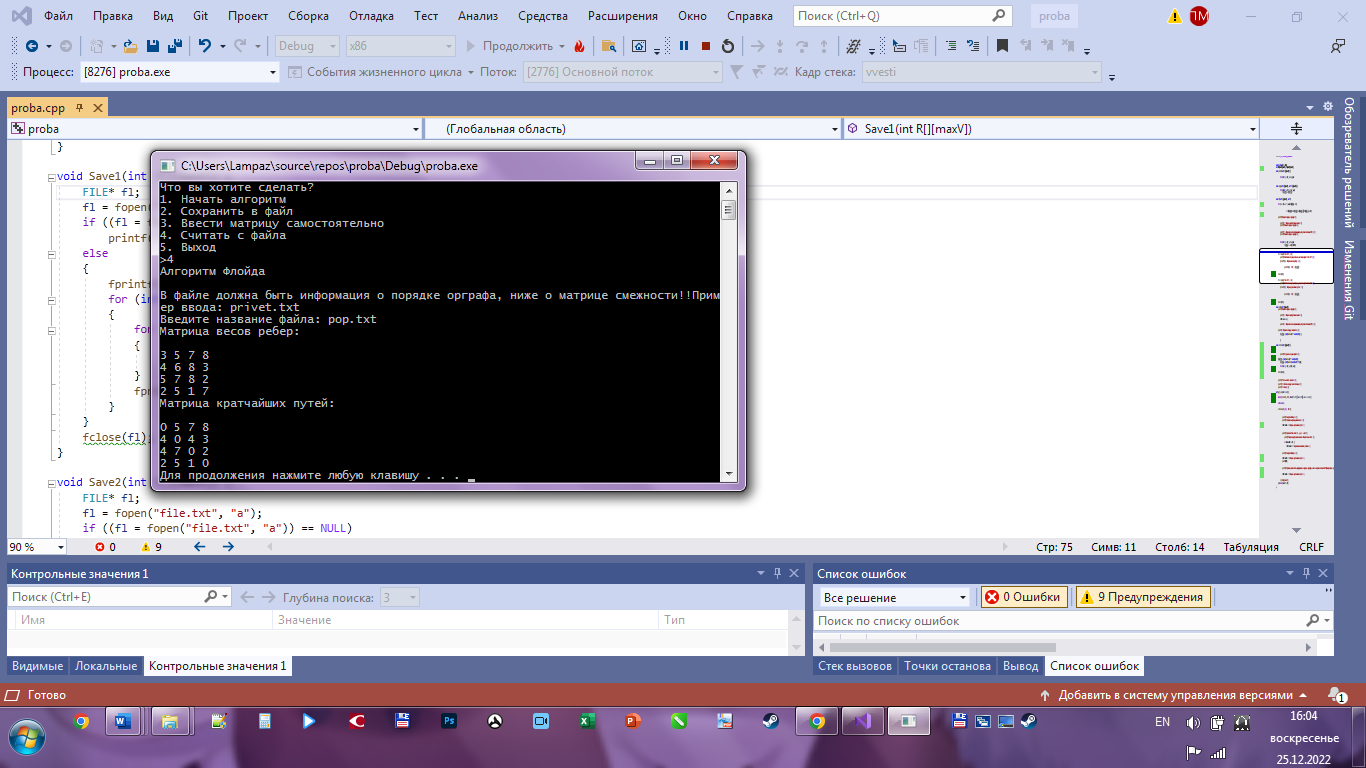


Рисунок 8- Тестирование при считывании значений из файла

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о выборе: начать алгоритм, ввести матрицу самостоятельно, сохранить в файл и считать с файла, выход | Верно |
| Выбор начать алгоритм | Вывод сообщения о порядке орграфа и генерация случайной матрицы, вывод матрицы кратчайших путей | Верно |
| Выбор ввести матрицу самостоятельно | Вывод сообщения о порядке орграфа и ввод матрицы с клавиатуры, вывод матрицы кратчайших путей | Верно |
| Выбор сохранить файл | Вывод сообщения о точности решения, успешное сохранение | Верно |
| Выбор считать с файла | Вывод сообщения о названии файла, ввод имени файла, вывод матрицы смежности, вывод матрицы кратчайших путей | Верно |
| Выбор выход | Выход из приложения | Верно |

Таблица 1 – Описание поведения программы при тестировании

В результате тестирования было выявлено, что программа корректно отрабатывает и выводит верные результаты.

# **Ручной расчет задачи**

Проведем проверку работы программы посредством ручных вычислений на примере графа с 4 вершинами (рис. 8). Приведем матрицу смежности.

Таблица 2 – матрица смежности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | [0] | [1] | [2] | [3] |
| [0] | 3 | 5 | 7 | 8 |
| [1] | 4 | 6 | 8 | 3 |
| [2] | 5 | 7 | 8 | 2 |
| [3] | 2 | 5 | 1 | 7 |

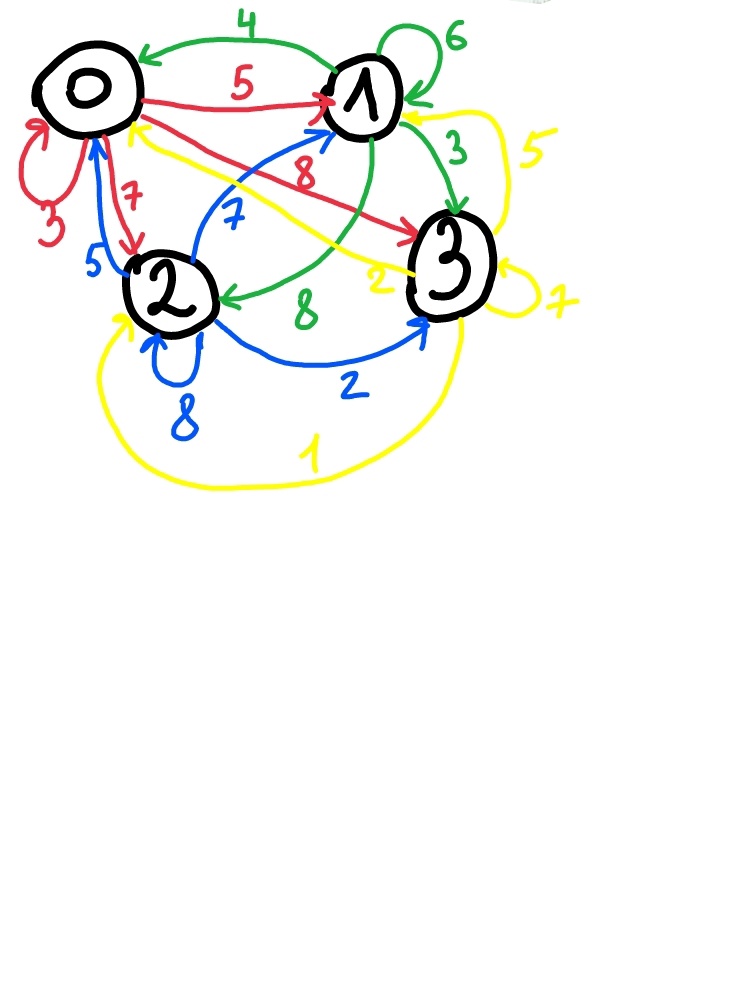


Рисунок 9- представление матрицы смежности в другом виде

Ищем наименьший пути из вершины 0.

1) От 0 до 0 будет 0, так как алгоритм Флойда игнорирует петли.

2) От 0 до 1 будет 0-1= 5 , 0-2-1= 15 , 0-2-3-1= 14 , 0-3-1= 13. Самый наименьший путь является 0-1=5. По такому же принципу проверяются все остальные пути и выбирается, где самый наименьший путь.

3)От 0 до 2 будет 7. 4)От 0 до 3 будет 8.

Ищем наименьшие пути из вершины 1.

1) От 1 до 0 будет 4. 2) От 1 до 1 будет 0, так как это петля.

3) От 1 до 2 будет 1-3-2= 4. 4) От 1 до 3 будет 3.

Ищем наименьшие пути из вершины 2.

1) От 2 до 0 будет 2-3-0= 4. 2) От 2 до 1 будет 7.

3) От 2 до 2 будет 0, так как это петля. 4) От 2 до 3 будет 2.

Ищем наименьшие пути из вершины 3.

1) От 3 до 0 будет 2. 2) От 3 до 1 будет 5.

3) От 3 до 2 будет 1. 4) От 3 до 3 будет 0, так как петля.

В итоге на выходе получается такая матрица:

Таблица 2 – матрица кратчайших путей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | [0] | [1] | [2] | [3] |
| [0] | 0 | 5 | 7 | 8 |
| [1] | 4 | 0 | 4 | 3 |
| [2] | 4 | 7 | 0 | 2 |
| [3] | 2 | 5 | 1 | 0 |

Таким образом, сравнивая полученный результат посредством ручного расчета с вычисленным результатом программы (рис. 7) можно сделать вывод, что программа работает верно.

# **Заключение**

В процессе создания проекта была разработана программа, реализующая алгоритм поиска кратчайших расстояний между всеми вершинами взвешенного ориентированного графа (Алгоритм Флойда).

В ходе выполнения курсовой работы были получены и расширены навыки разработки многомодульных программ на языке C, освоены приемы создания и редактирования матрицы смежности, ее интерпретации. Повторен и изучен расширенно алгоритм поиска в глубину. Закреплены приемы работы в компиляторе Visual Studio 2019.

Недостатком разработанной программы можно считать примитивный интерфейс, однообразное представление графа в виде матрицы смежности (а не в виде самого графа или списка смежности). Программа имеет небольшой, но достаточный для использования функционал возможностей.

В дальнейшем программу можно улучшить, добавив псевдографический интерфейс, вывод сообщений обо всех ошибках и подсказок при неверном действии пользователя, оптимизировав код.

# **Список литературы**

1. <https://habr.com/ru/post/105825/>
2. <https://e-maxx.ru/algo/floyd_warshall_algorithm>
3. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006

# **Приложение А. Листинг программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <stdio.h>

#include <iostream>

const int maxV = 100;

int i, j, n, variant, vari;

int GR[maxV][maxV], D[maxV][maxV];

void printM(int M[][maxV]) {

printf("\n");

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

{

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

{

printf("%d ", M[i][j]);

}

printf("\n");

}

}

void copy(int M[][maxV], int M1[][maxV]) {

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

{

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

{

M1[i][j] = M[i][j];

}

}

}

void FU(int R[][maxV], int V)

{

int k;

copy(R, D);

for (i = 0; i < V; i++) D[i][i] = 0;

for (k = 0; k < V; k++)

for (i = 0; i < V; i++)

for (j = 0; j < V; j++)

if (D[i][k] && D[k][j] && i != j)

if (D[i][k] + D[k][j] < D[i][j] || D[i][j] == 0)

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

}

void orgraf(int R[][maxV]) {

printf("Введите порядок орграфа:");

std::cin >> n;

if (n < 0)

{

printf("\n Введен недопустимый символ\n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

if (n > 50)

{

printf("\n Введенное число превышающее допустимое значение(50) \n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

srand(time(NULL));

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

{

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

{

R[i][j] = rand() % 10;

}

}

}

void Save1(int R[][maxV]) {

FILE\* fl;

fl = fopen("file.txt", "w");

if ((fl = fopen("file.txt", "w")) == NULL)

printf("Невозможно открыть/создать для записи файл 'file.txt'\n");

else

{

fprintf(fl, "Матрица весов ребер: \n");

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

{

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

{

fprintf(fl, " %d ", R[i][j]);

}

fprintf(fl, "\n ");

}

}

fclose(fl);

}

void Save2(int R[][maxV]) {

FILE\* fl;

fl = fopen("file.txt", "a");

if ((fl = fopen("file.txt", "a")) == NULL)

printf("Невозможно открыть/создать для записи файл 'file.txt'\n");

else

{

fprintf(fl, "Матрица кратчайших путей: \n");

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

{

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

{

fprintf(fl, " %d ", R[i][j]);

}

fprintf(fl, "\n ");

}

}

fclose(fl);

}

void vvod(int R[][maxV])

{

printf("Введите порядок орграфа:");

std::cin >> n;

if (n < 0)

{

printf("\n Введен недопустимый символ\n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

if (n > 50)

{

printf("\n Введенное число превышающее допустимое значение(50) \n");

printf("\n");

printf("Введите порядок орграфа:");

std::cin >> n;

}

printf("\nВведите матрицу смежности:\n");

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

R[i][j] = (int)malloc(n \* sizeof(int)); }

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

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

scanf\_s("%d", &R[i][j]);

}

}

}

void vvesti(int R[][maxV]) {

char vvesti[50];

FILE\* fil;

scanf("%s", &vvesti);

fil = fopen(vvesti, "r");

if (fil == NULL)

{

printf("Не удалось открыть файл!\n");

return;

}

fscanf(fil, "%d", &n);

R[i][j] = (int)malloc(n \* sizeof(int));

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

R[i][j] = (int)malloc(sizeof(int) \* (n));

}

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

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

{

fscanf(fil, "%d", &R[i][j]);

}

}

fclose(fil);

return;

}

void Menu() {

system("cls");

printf("Что вы хотите сделать?\n");

printf("1. Начать алгоритм\n");

printf("2. Сохранить в файл\n");

printf("3. Ввести матрицу самостоятельно\n");

printf("4. Считать с файла\n");

printf("5. Выход\n");

printf(">");

}

int get\_vari(int count) {

int vari;

char s[100];

scanf("%s", s);

while (sscanf(s, "%d", &vari) != 1 || vari < 1|| vari > count) {

printf("Введен недопустимый символ. Ваш ответ: ");

scanf("%s", s);

}

return vari;

}

int main()

{

setlocale(LC\_ALL, "Rus");

do {

Menu();

vari = get\_vari(5);

switch (vari) {

case 1:

printf("Алгоритм Флойда \n");

printf("\n");

printf("Создание матрицы динамически \n");

orgraf(GR);

std::cout << "Матрица весов ребер:\n";

printM(GR);

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

FU(GR, n);

printM(D);

break;

case 2:

printf("Соранить? Ваш ответ (1 - да, 2 - нет):");

std::cin >> variant;

if (variant != 1 && variant != 2) {

printf("Введен недопустимый символ. Введите ваш ответ: ");

std::cin >> variant;

}

else {

if (variant == 1) {

Save1(GR);

Save2(D);

std::cout << "Сохранение выполнено успешно.\n";

}

break;

}

case 3:

printf("Алгоритм Флойда \n");

printf("\n");

printf("Создание матрицы динамически \n");

vvod(GR);

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

FU(GR,n);

printM(D);

break;

case 4:

printf("Алгоритм Флойда \n");

printf("\n");

printf("В файле должна быть информация о порядке орграфа, ниже о матрице смежности!!Пример ввода: privet.txt\nВведите название файла: ");

vvesti(GR);

std::cout << "Матрица весов ребер:\n";

printM(GR);

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

FU(GR,n);

printM(D);

break;

}

system("pause");

system("cls");

} while (vari != 5);

//Clear(root);

return 0;

}