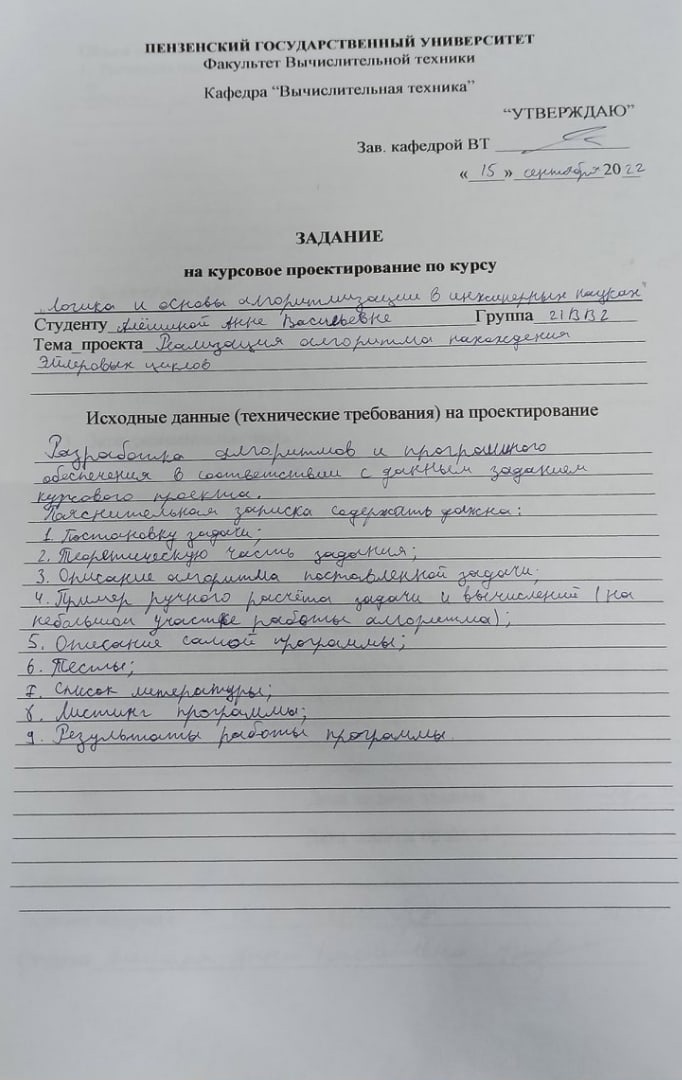
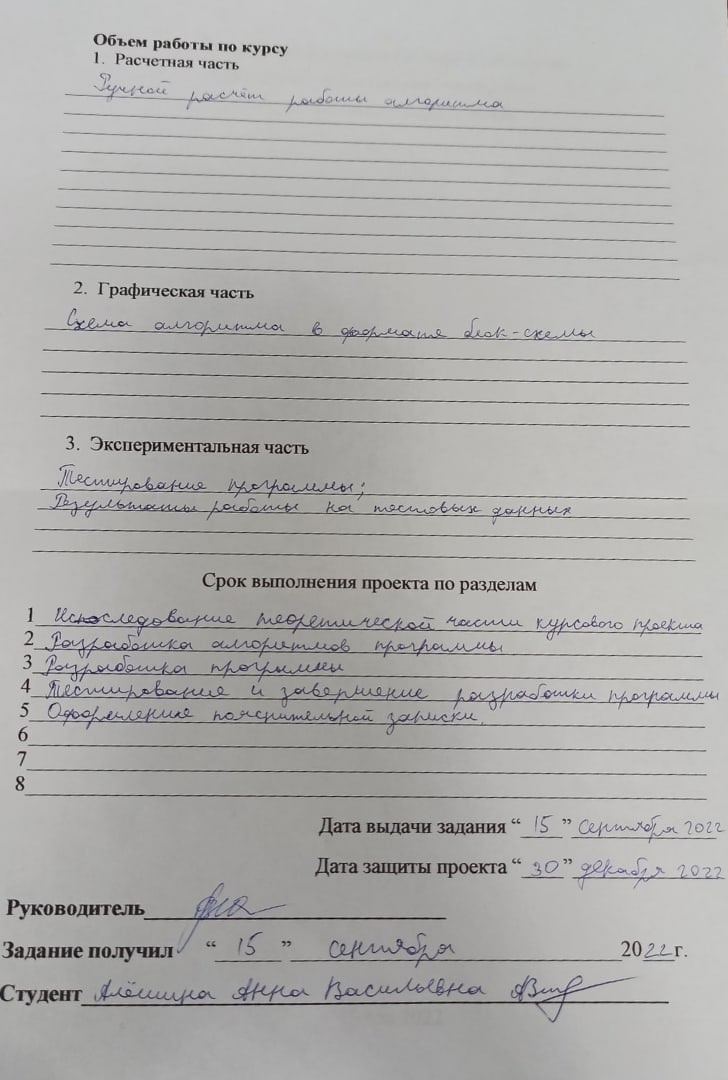


**Содержание**



[**Реферат** 3](#_Toc122913562)



**Содержание**

[**Введение** 4](#_Toc122913563)

[**Постановка задачи** 5](#_Toc122913564)

[**Теоретическая часть задания** 6](#_Toc122913565)

[**Описание алгоритма программы** 7](#_Toc122913566)

[**Описание программы** 9](#_Toc122913567)

[**Тестирование** 12](#_Toc122913568)

[**Ручной расчёт задачи** 15](#_Toc122913569)

[**Заключение** 16](#_Toc122913570)

[**Список литературы** 17](#_Toc122913571)

[**Приложение А.** 18](#_Toc122913572)

## **Реферат**

Отчет 22 стр, 11 рисунков.

ТЕОРИЯ ГРАФОВ, ОРГРАФ, ПОИСК В ГЛУБИНУ, ЭЙЛЕРОВ ЦИКЛ.

Цель исследования – разработка программы поиска путей орграфа,  проходящей по всем рёбрам [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и притом только по одному разу, используя алгоритм поиска в глубину.

## **Введение**

Алгоритм поиска (или обхода) в глубину (англ. depth-firstsearch, DFS) позволяет построить обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины.

Отличие поиска в глубину от поиска в ширину заключается в том, что (в случае неориентированного графа) результатом алгоритма поиска в глубину является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины. Этим он принципиально отличается от поиска в ширину, где одновременно обрабатывается множество вершин, в поиске в глубину в каждый момент исполнения алгоритма обрабатывается только одна вершина.

С другой стороны, поиск в глубину не находит кратчайших путей, зато он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.

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

Целью данной курсовой работы является разработка программы на языке Си++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм нахождения Эйлеровых циклов путём поиска в глубину.

## **Постановка задачи**

Требуется разработать программу, которая реализует алгоритм нахождения Эйлеровых циклов, используя алгоритм поиска в глубину.

Исходный граф в программе должен задаваться матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности или же чтобы программа сама генерировала матрицу. После обработки этих данных на экран должна выводиться матрица смежности орграфа и Эйлеров цикл. Необходимо предусмотреть, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

## **Теоретическая часть задания**

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, переход от одной вершины к другой, осуществляется по ребрам графа.

Однажды математики задались одной интересной задачкой, в которой спрашивалось, как можно пройти по семи мостам центра старого Кёнигсберга, не проходя ни по одному из них дважды. Впервые была решена в 1736 году, математиком [Леонардом Эйлером](https://ru.wikipedia.org/wiki/%D0%AD%D0%B9%D0%BB%D0%B5%D1%80,_%D0%9B%D0%B5%D0%BE%D0%BD%D0%B0%D1%80%D0%B4), который доказал, что это невозможно, и по ходу доказательства изобрёл Эйлеровы циклы.

Пусть задан граф *{\displaystyle G=(V,E)}G = (V, E)*. (рисунок 1) задается множеством вершин и множеством ребер, соединяющих между собой определенные вершины.

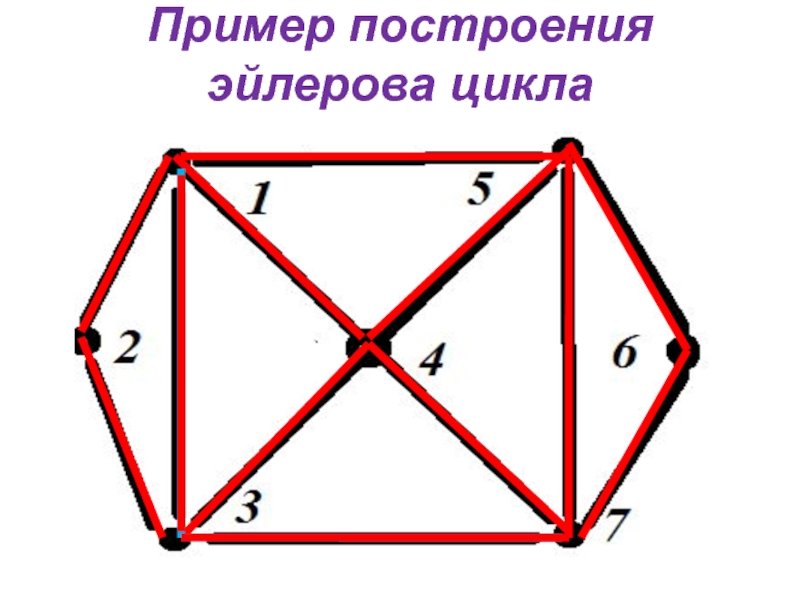


Рисунок 1 – Пример Эйлерова цикла

Начинаем с некоторой вершины {\displaystyle p\in V} и каждый раз вычеркиваем пройденное ребро. Не проходим по ребру, если удаление этого ребра приводит к разбиению графа на две связные компоненты (не считая изолированных вершин), т.е. необходимо проверять, является ли ребро [мостом](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D1%81%D1%82_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)) или нет.

## **Описание алгоритма программы**

Надо реализовать алгоритм в основе которого лежит перебор вершин графа, переход от одной вершины к другой по ребрам графа, но каждое ребро можно пройти только один раз. В настоящем разделе дается краткое описание одной из таких процедур.

Для программной реализации алгоритма понадобиться проверить можно ли использовать рёбра для данного цикла. Для того чтобы ребро подходило для программы, оно должно выполнять одно из условий. Первое – если вершина является единственной смежной вершиной, то такое ребро можно использовать. Второе условие – надо проверить является ли ребро мостом. Если данный мост подходит под определение смежности, то мы выводим получившееся ребро. Алгоритм продолжается до тех пор, пока не будут пройдены все рёбра.

Ниже представлен псевдокод функции isValidNextEdge().

**isValidNextEdge()**

1. count = 0
2. Создаётся итератор i динамического списка list
3. Для итератора i=начало списка пока i!= началу списка делать i=i+1
4. Если [итератор i] ! = -1
5. count = count+1
6. Конец условия
7. Конец цикла
8. Если count = 1
9. Вернуть true
10. Конец условия
11. Вершина Vertex посещена
12. count1 = DFS(u)
13. Вызов функции rmvEdge
14. Массив visited заполняется false
15. count2 = DFS(u)
16. Вызов функции addEdge
17. Вернуть true если count1 > count2

Полный код программы можно увидеть в Приложении А.

## **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.

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

Работа программы начинается с запроса на количество вершин в орграфе. Далее у пользователя запрашивается какой вариант дальнейших действий надо выполнить.

cout << "Введите кол-во вершин графа: "; cin >> z;

Graph c(z);

cout << endl

<< "|------------------------|" << endl

<< "| 1)Случайное заполнение |" << endl

<< "|------------------------|" << endl

<< "| 2)Заполнить самому |" << endl

<< "|------------------------|" << endl

<< "| 3)Выход |" << endl

<< "|------------------------|" << endl;

cout << " Введите: ";

cin >> s;

string q1, q2, q3;

Первый вариант – случайное заполнение графа. Для этого варианта генерируется рандомное заполнение матрицы смежности. После чего она выводится на экран и результат является ли этот цикл Эйлеровым.

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

{

case 1:

system("cls");

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

{

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

{

if (bool(rand() % 2)) {

c.addEdge(i, j);

}

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

if (c.proverka())

c.print();

else

cout << "Граф является не эйлеровым" << endl;

system("pause");

break;

Второй вариант действий – пользователь сам вводит значения для генерации графа. Вводятся две вершины, между которыми образуются рёбра, до тех пор, пока не будет введён «\*». После чего завершиться цикл, система выведет получившуюся матрицу и определит является ли она Эйлеровым циклом.

case 2:

system("cls");

while (1) {

system("cls");

cout << "Для выхода введите \*" << endl;

cout << "Введите 1-вершину: "; cin >> q1;

if (q1 == "\*") break;

cout << "Введите 2-вершину: "; cin >> q2;

if (q1 == "\*") break;

c.addEdge(atoi(q1.c\_str()), atoi(q2.c\_str()));

cout << endl;

c.toString();

system("pause");

}

system("cls");

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

if (c.proverka())

c.print();

else

cout << "Граф является не элеровым" << endl;

system("pause");

break;

И третий вариант – если вводится значение «3», то программа заканчивает свою работу.

case 3:

exit(0);

break;

}

Ниже можно увидеть оформление начального запроса и дальнейшие действия с ним.

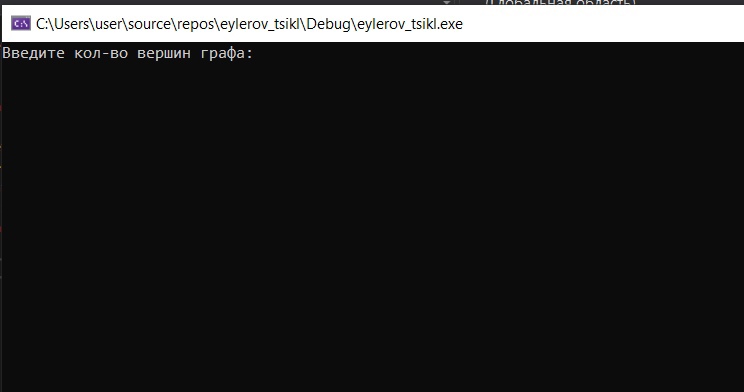


Рисунок 2 – Ввод размера графа

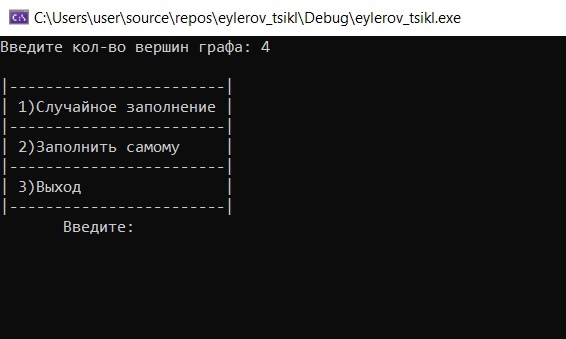


Рисунок 3 – Меню выбора действий

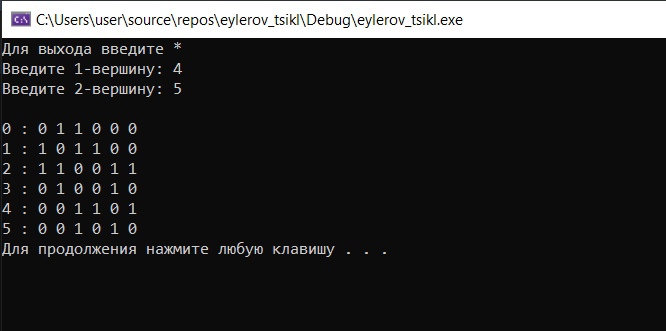


Рисунок 4 – Заполнение матрицы с клавиатуры

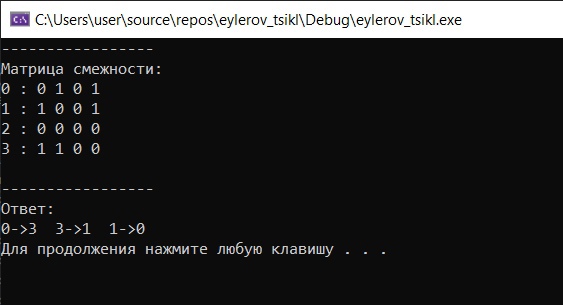


Рисунок 5 – Вывод матрицы и тропы Эйлера

## **Тестирование**

Среда разработки Microsoft Visual Studio 2019 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 данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем количества вершин, различных выборах действий и вывод получившейся матрицы и тропы Эйлера.

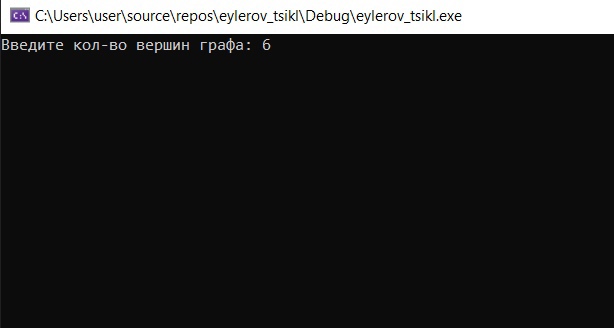


Рисунок 6 – Тестирование при вводе количество вершин = 6

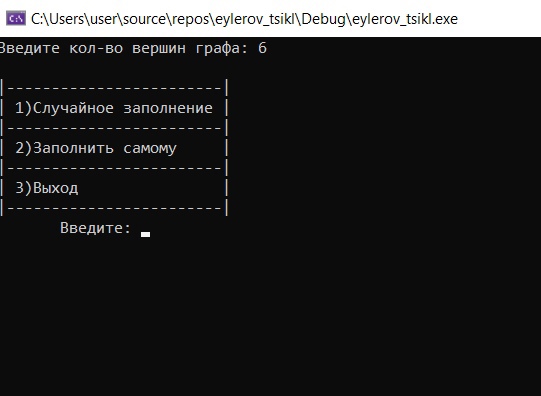


Рисунок 7 – Тестирование при вводе количество вершин = 6, выбор действия

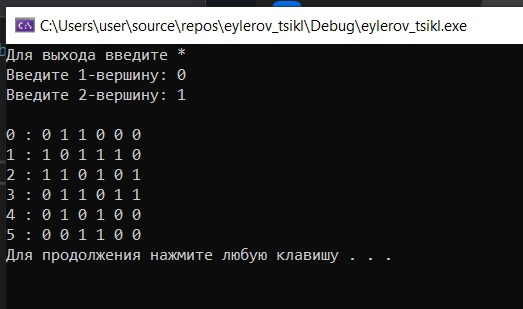


Рисунок 8 – Тестирование при вводе связей между вершинами

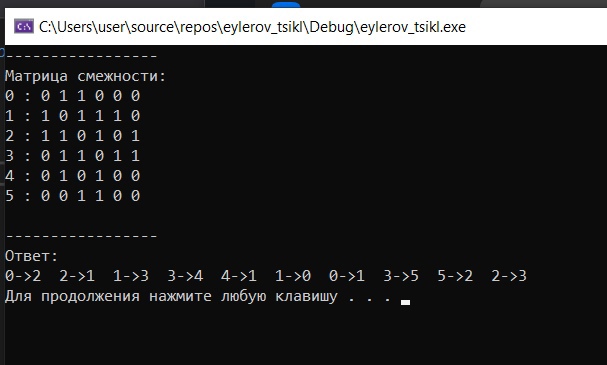


Рисунок 9 – Тестирование Эйлерового тура при количестве вершин = 6

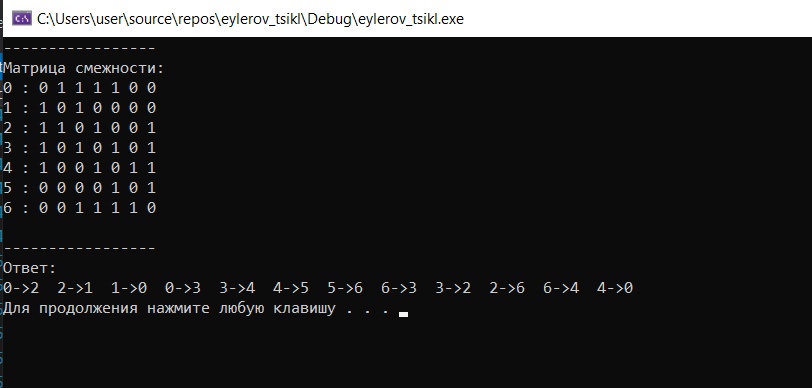


Рисунок 10 – Тестирование программы при вводе количество вершин = 7

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Выбор ввода количества вершин | Вывод сообщения о количестве вершин в графе | Верно (рисунок 6) |
| Выбор генерации матрицы | Вывод меню с выбор генерации случайным образом, ввод произвольных значений матрицы или завершение программы | Верно (рисунок 7) |
| Ввод рёбер графа | Ввод связей между вершинами графа | Верно (рисунок 8) |
| Выбор Эйлерового цикла | Вывод получившейся матрицы и Эйлерова цикла | Верно (рисунок 9) |
| Генерация матрицы с 10 вершинами | Вывод матрицы смежности с 10 вершинами. Вывод матрицы и Эйлерова цикла | Верно (рисунок 10) |

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

## **Ручной расчёт задачи**

Проведем проверку программы посредством ручных вычислений на примере графа с 6 вершинами (рисунок 8). Начинаем обход из 0 вершины в последующие. Проверяем, если есть путь из 0 в другие вершины, то идём дальше. В нашем случае идем в вершину 2. Аналогично проверяем и в вершине 2. Из вершины 2 есть путь в вершину 1. Из неё можно попасть в вершину 3. Также из 3 идём в вершину 4. Из 4 остаётся путь только в вершину 1. Далее из 1 в 0. Из вершины 0 есть путь в 1, идем туда. Из 3 идем в 5 вершину. Из вершины 5 можно пойти в 2. Из вершины 2 идём в 3. Из 3 вершины путей не пройденными нами нет. Поэтому возвращаемся в предыдущие и проверяем. Проверка показала, что все вершины пройдены. В итоге у нас получилось Эйлеров цикл {0; 2; 1; 3; 4; 1; 0; 1; 3; 5; 2; 3}. Из этого делаем вывод, что наш граф является Эйлеровым.

Таким образом, можно сделать вывод, что программа работает верно.

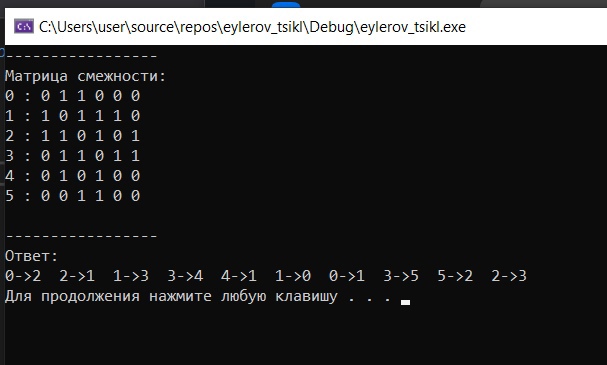


Рисунок 11 – Тестирование работы программы

## **Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм нахождения Эйлеровых циклов в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания языка программирования Cи++.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

## **Список литературы**

1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978

2. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006

3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.

4. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.

5. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

## **Приложение А.**

**Листинг программы**

#include <iostream>

#include <iomanip>

#include <algorithm>

#include <fstream>

#include <list>

using namespace std;

ofstream fout("cppstudio.txt");

// Класс, представляющий неориентированный граф

class Graph // непоределенный граф

{

bool\*\* G; // матрица смежности(для вида)

int Vertex; // Количество вершин

bool\* visited; //посещенные вершины

list<int>\* adj; // Динамический массив списков смежности

public:

// Конструктор и деструктор

Graph(int Vertex)

{

this->Vertex = Vertex;

adj = new list<int>[Vertex];

visited = new bool[Vertex];

memset(visited, false, Vertex);

G = new bool\* [Vertex];

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

G[i] = new bool[Vertex];

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

G[i][j] = 0;

}

}

~Graph() {

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

delete[] G[i];

delete[] G;

delete[] adj;

delete[]visited;

}

//вывод на экран матрицы

void toString() {

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

cout << i << " : ";

fout << i << " : ";

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

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

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

}

cout << endl;

fout << endl;

}

}

// функция для добавления

void addEdge(int u, int v) {

G[u][v] = G[v][u] = true;

adj[u].push\_back(v);

adj[v].push\_back(u);

}

//удаление ребёр

void rmvEdge(int u, int Vertex);

// Способы печати эйлерова тура

void print();

void printEuler(int s);

// Эта функция возвращает количество вершин, доступных из вершины. Это делает DFS

int DFS(int Vertex);

//элеров граф

bool isValidNextEdge(int u, int Vertex);

//проверка на эйлеровость

bool proverka();

};

/\* Основная функция, которая выводит эйлерову тропу. Сначала он находит

вершину нечетной степени (если таковая имеется), а затем вызывает printEulerUtil()

чтобы напечатать путь \*/

void Graph::print()

{

// Найти вершину с нечетной степенью

int u = 0; // временная переменная

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

if (adj[i].size() & 1) {

u = i; break;

}

printEuler(u);

cout << endl;

fout << endl;

}

// Вывести тур Эйлера, начинающийся с вершины u

void Graph::printEuler(int u)

{

// Повторяется для всех вершин, смежных с этой вершиной

for (auto i = adj[u].begin(); i != adj[u].end(); ++i)

{

int Vertex = \*i;

// Если u-вершина ребра не удалена и это допустимое следующее ребро

if (Vertex != -1 && isValidNextEdge(u, Vertex))

{

cout << u << "->" << Vertex << " ";

fout << u << "->" << Vertex << " ";

rmvEdge(u, Vertex);

printEuler(Vertex);

}

}

}

// Функция для проверки, может ли ребро u-вершина рассматриваться как следующее ребро

// Эйлер Зазывается

bool Graph::isValidNextEdge(int u, int Vertex)

{

// U-вершина ребра допустима в одном из следующих двух случаев:

// 1) Если вершина является единственной смежной вершиной u

int count = 0; // Для сохранения количества смежных вершин

list<int>::iterator i;

for (i = adj[u].begin(); i != adj[u].end(); ++i)

if (\*i != -1)

count++;

if (count == 1)

return true;

// 2) Если существует несколько смежных, то u-вершина не является мостом

// Выполните следующие действия, чтобы проверить, является ли u-вершина мостом

// 2.а) количество вершин, достижимых из u

visited[Vertex];

int count1 = DFS(u);

// 2.b) Удалите ребро (u, v) и после удаления ребра подсчитайте вершины, доступные из u

rmvEdge(u, Vertex);

memset(visited, false, Vertex);

int count2 = DFS(u);

// 2.c) Добавьте ребро обратно к графику

addEdge(u, Vertex);

// 2.d) Если count1 больше, то ребро (u, v) является мостом

return (count1 > count2) ? false : true;

}

bool Graph::proverka()

{

int sum = 0;

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

sum = 0;

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

if (G[i][j] == 1) sum++;

}

if (sum % 2) return false; // если степень вершины(хоть одной) нечетная, то граф не считается элеровым

}

return true;

}

// Эта функция удаляет u-вершину ребра из графика. Он удаляет ребро, заменяя значение смежной вершины на -1.

void Graph::rmvEdge(int u, int Vertex)

{

// Найдите вершину в списке смежности u и замените ее на -1

list<int>::iterator iv = find(adj[u].begin(), adj[u].end(), Vertex);

\*iv = -1;

// Найдите u в списке смежности вершины и замените его на -1

list<int>::iterator iu = find(adj[Vertex].begin(), adj[Vertex].end(), u);

\*iu = -1;

}

// Функция на основе DFS для подсчета достижимых вершин из вершины

int Graph::DFS(int start)

{

// Отметить текущий узел как посещенный

visited[start] = true;

int count = 1;

// Повторяется для всех вершин, смежных с этой вершиной

for (auto i = adj[start].begin(); i != adj[start].end(); ++i)

if (\*i != -1 && !visited[\*i])

count += DFS(\*i);

return count;

}

int main()

{

setlocale(0, "");

srand(time(0));

int z;

string s;

while (true) {

system("cls");

cout << "Введите кол-во вершин графа: "; cin >> z;

Graph c(z);

cout << endl

<< "|------------------------|" << endl

<< "| 1)Случайное заполнение |" << endl

<< "|------------------------|" << endl

<< "| 2)Заполнить самому |" << endl

<< "|------------------------|" << endl

<< "| 3)Выход |" << endl

<< "|------------------------|" << endl;

cout << " Введите: ";

cin >> s;

string q1, q2, q3;

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

{

case 1:

system("cls");

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

{

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

{

if (bool(rand() % 2)) {

c.addEdge(i, j);

}

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

if (c.proverka())

c.print();

else

cout << "Граф является не элеровым" << endl;

system("pause");

break;

case 2:

system("cls");

while (1) {

system("cls");

cout << "Для выхода введите \*" << endl;

cout << "Введите 1-вершину: "; cin >> q1;

if (q1 == "\*") break;

cout << "Введите 2-вершину: "; cin >> q2;

if (q1 == "\*") break;

c.addEdge(atoi(q1.c\_str()), atoi(q2.c\_str()));

cout << endl;

c.toString();

system("pause");

}

system("cls");

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << "-----------------" << endl << "Матрица смежности: \n";

c.toString();

cout << endl << "-----------------" << endl << "Ответ: \n";

fout << endl << "-----------------" << endl << "Ответ: \n";

if (c.proverka())

c.print();

else

cout << "Граф является не элеровым" << endl;

system("pause");

break;

case 3:

exit(0);

break;

}

system("cls");

}

fout.close();

return 0;

}