

**Содержание**

[Реферат 5](#__RefHeading___Toc2814_1099486129)

[Введение 6](#__RefHeading___Toc2816_1099486129)

[1 Постановка задачи 7](#__RefHeading___Toc2818_1099486129)

[2 Теоретическая часть 8](#__RefHeading___Toc2820_1099486129)

[3 Описание алгоритма программы 11](#__RefHeading___Toc2822_1099486129)

[4 Ручной расчет задачи 14](#__RefHeading___Toc2824_1099486129)

[5 Описание программы 16](#__RefHeading___Toc2826_1099486129)

[6 Тестирование 20](#__RefHeading___Toc2828_1099486129)

[Заключение 24](#__RefHeading___Toc2830_1099486129)

[Список литературы 25](#__RefHeading___Toc2832_1099486129)

[Приложение A. Листинг программы 26](#__RefHeading___Toc2834_1099486129)

# Реферат

Отчет 37 страниц, 19 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ЭЙЛЕРОВ ЦИКЛ, ЭЙЛЕРОВ ПУТЬ, ПОИСК В ГЛУБИНУ, РЕКУРСИЯ.

Цель исследования – разработка программы, способная находить Эйлеров цикл, используя алгоритм поиска в глубину.

В работе рассмотрены правила нахождения Эйлерова цикла, также присутствует возможность определения отсутствия данного цикла.

Добавлена функция записи в файл всех конечных значений.

# Введение

Эйлеров путь в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу. Эйлеров цикл – замкнутый Эйлеров путь.

Согласно теореме, доказанной [Эйлером](https://ru.wikipedia.org/wiki/Эйлер,_Леонард), эйлеров цикл в неориентированном графе существует [тогда и только тогда](https://ru.wikipedia.org/wiki/Тогда_и_только_тогда), когда граф связный или будет являться связным, если удалить из него все изолированные вершины, и в нём отсутствуют вершины нечётной [степени](https://ru.wikipedia.org/wiki/Степень_вершины_(теория_графов)).

Эйлеров путь в неориентированном графе существует тогда и только тогда, когда граф связный и число вершин с нечетной степенью равно нулю или двум. Причём когда оно равно нулю, эйлеров путь вырождается в эйлеров цикл.

В качестве среды для разработки программы была выбрана Microsoft Visual Studio 2022, язык программирования C++.

Целью данной курсовой работы является разработка программы на языке C++, которая определяет наличие эйлерова цикла в неориентированном графе.

# 1 Постановка задачи

Требуется разработать программу, которая находит эйлеров цикл в неориентированном графе.

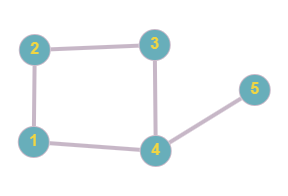
Исходный граф в программе должен задаваться матрицей смежности. Исходный граф в программе должен задаваться матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности неориентированного графа, вид графа и степени вершин. Необходимо предусмотреть проверку на корректность ввода данных, реализовать функцию вывода данных в файл.

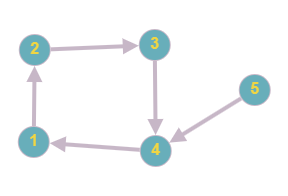
Устройство ввода – клавиатура и мышь.

# 2 Теоретическая часть

Граф – совокупность точек, соединенных линиями. Точки называются вершинами, а линии ребрами. Степенью входа вершины – количество входящих в нее ребер, а степень выхода – количество исходящих ребер.

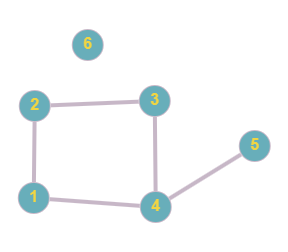
Графы можно разделить на ориентированные (рис. 1) и неориентированные (рис. 2). Отличие отображения ориентированного и неориентированного графов – обозначение ребер. В ориентированном графе ребро может иметь направление движения, которое обозначается стрелкой.

Рисунок 1 - Пример неориентированного графа

Рисунок 2 - Пример орграфа

Графы также можно разделить на связные (рис. 1 также является связным) и несвязные (рис. 3).

Граф может быть представлен несколькими способами:

Рисунок 3 - Пример несвязного графа

1. Матрица смежности;

2. Матрица инцидентности;

3. Список смежности или инцидентности;

4. Список ребер.

При представлении графа в виде матрицы смежности – информация о ребрах хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Эйлеров путь в графе – это путь, проходящий по всем ребрам графа и притом только по одному разу. Эйлеров цикл – Эйлеров путь, являющимся циклом, то есть замкнутый путь, проходящий через каждое ребро графа ровно по одному разу. Эйлеров граф – граф содержащий Эйлеров путь.

Согласно теореме, доказанной Эйлером, Эйлеров цикл существует в неориентированном графе тогда и только тогда, когда граф связный или будет являться связным, если удалить из него все изолированные вершины, и в нем отсутствуют вершины нечетной степени.

Для поиска Эйлерова цикла (пути) можно использовать алгоритм обхода графа – поиска в глубину. Поиск в глубину предполагает продвижение в глубь до тех пор, пока это возможно. Невозможность продвижения означает, что следующим шагом будет переход на последнюю, имеющую несколько вариантов движения, ранее посещенную вершину.

# 3 Описание алгоритма программы

Для начала программа проверяет степени вершин: если вершин с нечётной степенью нет, то в графе есть эйлеров цикл, если есть 2 вершины с нечётной степенью, то в графе есть только эйлеров путь (эйлерова цикла нет), если же таких вершин больше 2, то в графе нет ни эйлерова цикла, ни эйлерова пути.

Для нахождения эйлерова пути, поступим таким образом: если V1 и V2 - это две вершины нечётной степени, то просто добавим ребро (V1,V2), в полученном графе найдём эйлеров цикл (он, очевидно, будет существовать), а затем удалим из ответа "фиктивное" ребро (V1,V2).

Эйлеров цикл будем искать без добавления ребра.

Программа учитывает, что в графе могут быть изолированные вершины.

Ниже представлен псевдокод функции для нахождения эйлерова цикла и эйлерова пути FindWay.

**FindWay**

1. создать переменную first = 0;

2. **ПОКА** в векторе deg элемент с номером first будет равен 0

3. ++first;

4. создать переменные v1 = -1, v2 = -1;

5. создать переменную bad = false;

6. **ДЛЯ** i = 0 пока i < числа вершин в графе делать ++i

7. **ЕСЛИ** deg[i] & 1

8. **ЕСЛИ** v1 == -1

9. v1 = i;

7. **ИНАЧЕ ЕСЛИ** v2 == -1

8. v2 = i;

9. **ИНАЧЕ**

10. bad = true;

11. **ЕСЛИ** v1 != -1

12. в матрице смежности графа ++graph[v1][v2], ++graph[v2][v1];

13. создать структуру стэк st;

14. добавить в стэк st переменную first;

15.  **ПОКА** st не пустой

16. создать переменную v = самому верхнему элементу в стеке;

17. создать переменную i;

18. **ДЛЯ** i = 0 пока i < числа вершин в графе делать ++i

19. **ЕСЛИ** элемент матрицы смежности graph[v][i] == 1

20. break;

21. **ЕСЛИ** i == числу вершин в графе

22. добавить в вектор res переменную v;

23. удалить самый верхний элемент в st;

24. **ИНАЧЕ**

25. --graph[v][i];

26. --graph[i][v];

27. добавить в стэк st переменную i;

28. **ПОКА** v1 != -1

29. **ДЛЯ** i = 0 пока i + 1 < размер вектора res делать ++i

30. **ЕСЛИ** res[i] == v1 && res[i + 1] == v2 || res[i] == v2 && res[i + 31. 1] == v1)

32. создать вектор res2;

33. **ДЛЯ** j = i + 1 пока j < размер вектора res делать ++j

34. добавить в res2 элемент res[j];

35. **ДЛЯ** j = 1 пока j <= i делать ++j

36. добавить в res2 элемент res[j];

37. res = res2;

38. break;

39. вывод "Путь:";

40.  **ДЛЯ** i = 0 пока i + 1 < размер вектора res делать ++i

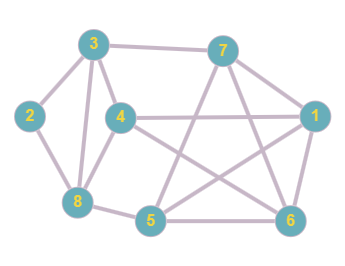
41. вывод " ";

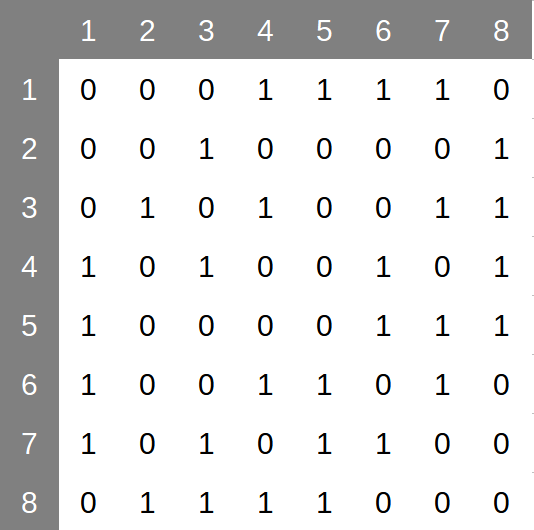
42. вывод res[i] + 1;

# 4 Ручной расчет задачи

Проведем проверку работы программы ручным расчетом.

C помощью программы был найден эйлеров цикл для графа с 8 вершинами (рис. 4). Матрица смежности данного графа представлена на рисунке 5.

Рисунок 4 - Графическое представление графа для ручного расчета

Рисунок 5 - Матрица смежности графа для ручного расчета

Программа обнаружила следующий эйлеров цикл:

{1, 6, 7, 5, 8, 4, 6, 5, 1, 7, 3, 8, 2, 3, 4, 1}.

Чтобы проверить корректность работы программы, сперва необходимо проверить условия существования Эйлерова цикла. Для этого нужно, чтобы каждая вершина имела четную степень и граф был связанным. Из рисунка 4 видно, что из каждой вершины выходит четное количество ребер и граф связан, а это значит, что условие выполняется и можно продолжать последующее вычисление Эйлерова цикла.

Обход начинается с вершины 1. Затем осуществляется переход в вершину 4 и удаление пути между 1 и 4 вершинами. Удаление пути происходит после каждого перехода из одной вершины в другую. После этого посещается 3, из нее переходим во 2, затем в 8. Из последней снова возвращаемся в 3, посещаем 7, из нее возвращаемся в 1. Переходим в 5, из нее в 6, затем в 4. Отправляемся в 8, после нее в 5, затем в 7. Из 7 переходим в 6 и, перейдя вновь в 1, замыкаем цикл. Эйлеров путь есть. Записав последовательность в обратном порядке, получим следующий путь:

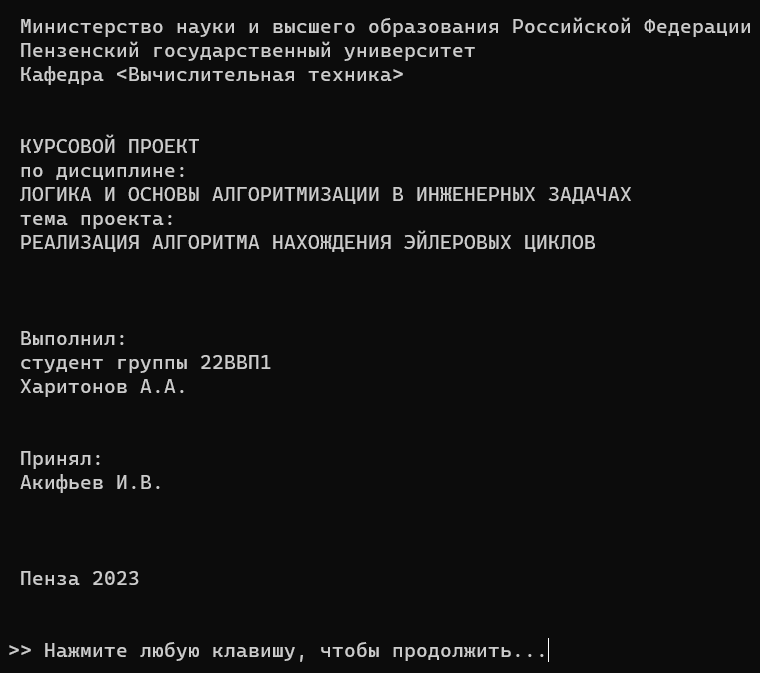
{1, 6, 7, 5, 8, 4, 6, 5, 1, 7, 3, 8, 2, 3, 4, 1}.

Ручной расчет совпал с результатом работы программы.

# 5 Описание программы

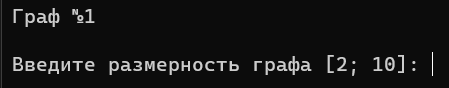
Для реализации программы использован язык программирования С++. Проект создан в виде консольного приложения.

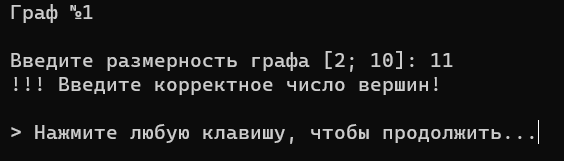
Работа программы начинается с вывода ознакомительной информации, после чего программа ожидает действия от пользователя (рис. 6).

Рисунок 6 - Начало работы

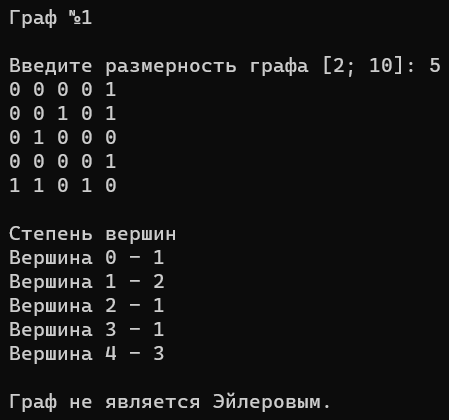
Затем программа просит ввести пользователя количество вершин для графа на отрезке от 1 до 10 (рис. 7).

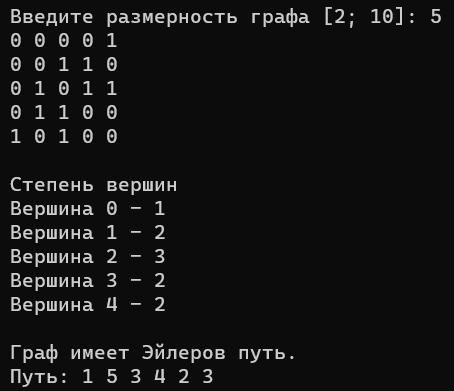
При попытке некорректного ввода, появляется сообщение об ошибке (рис. 8). После нажатия любой клавиши, программа снова попросит ввести число вершин.

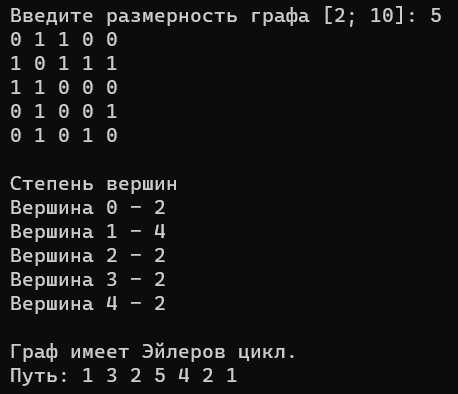
Рисунок 7 - Ввод размерности графа

Рисунок 8 - Сообщение об ошибке

После ввода количества вершин, программа выводит информацию о созданном графе, о наличии или отсутствии эйлерова цикла (или пути). Если цикл (или путь) есть, то программа выведет его.

Рисунок 9 - Граф не является эйлеровым

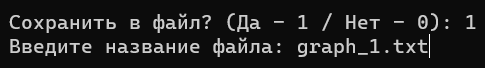
Рисунок 10 - Граф имеет эйлеров путь

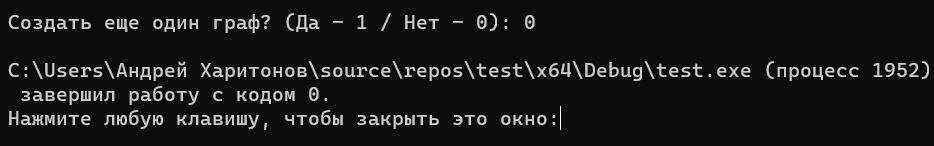
Рисунок 11 - Граф имеет эйлеров цикл

Затем программа предлагает сохранить информацию о графе в файл.

Если пользователь соглашается, то программа запрашивает имя файла, в который будет сохранена информация (рис. 12).

Затем программа предложит создать ещё один граф. Если пользователь согласится, то цикл работы программы повториться. Иначе работа будет завершена (рис. 13).

Рисунок 12 - Сохранение в файл

Рисунок 13 - Завершение работы

# 

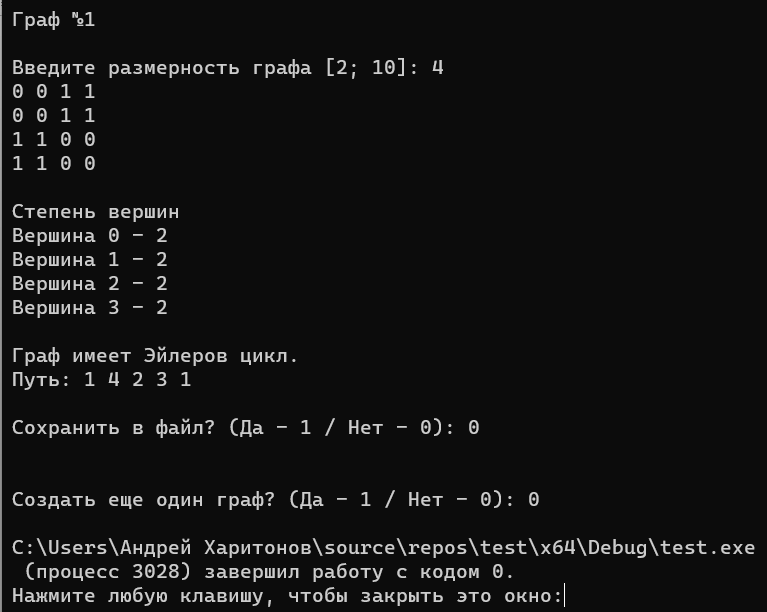
# 6 Тестирование

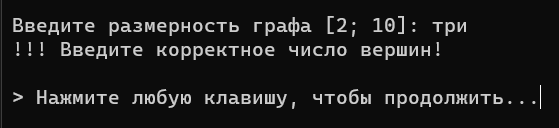
Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

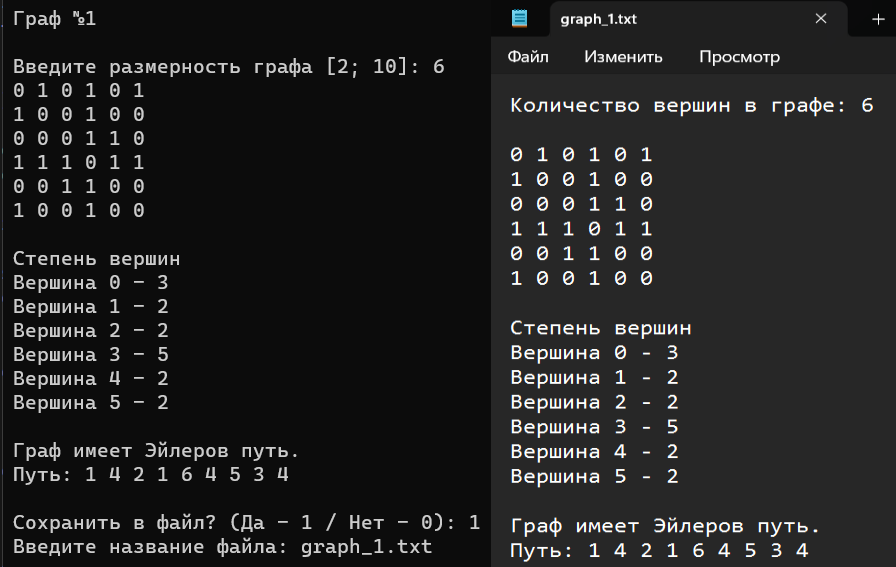
Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы.

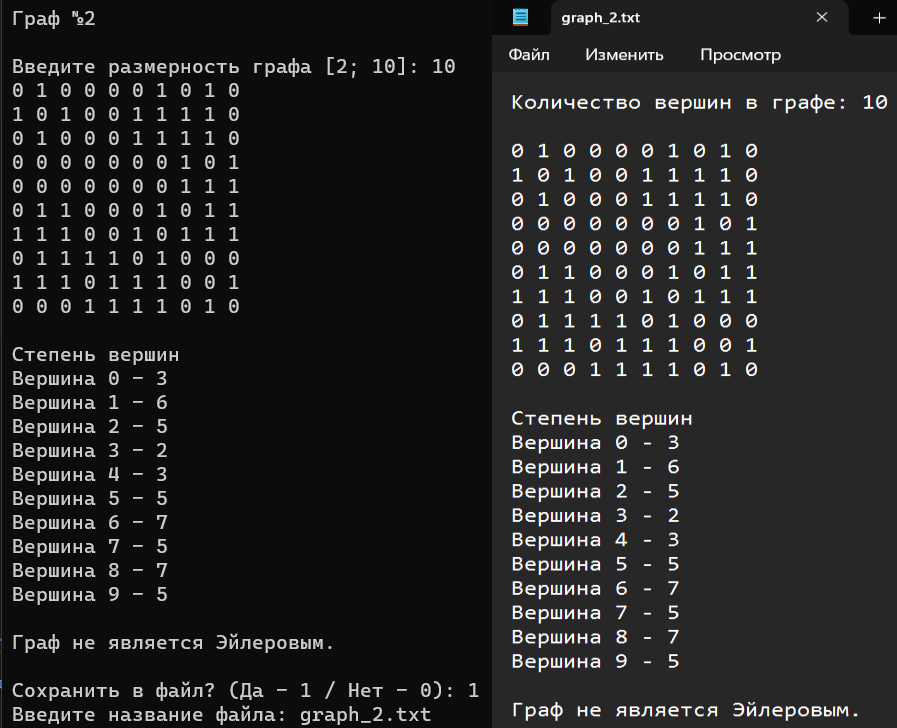
Ниже продемонстрирован результат тестирования программы при

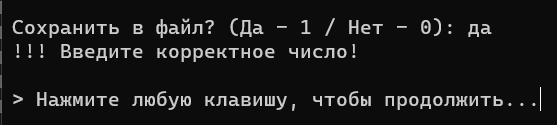
вводе пользователем различных количеств вершин.

Рисунок 14 - Тестирование при вводе 4 вершин

Рисунок 15 - Попытка некорректного ввода числа вершин

Рисунок 16 - Вывод матрицы из 6 вершин и сохранение в файл

Рисунок 17 - Вывод матрицы из 10 вершин и сохранение в файл

Рисунок 18 - Попытка некорректного ввода при сохранении в файл

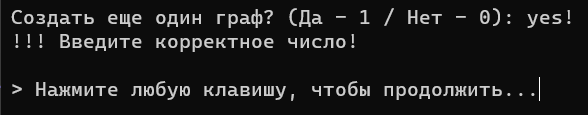
Рисунок 19 - Попытка некорректного ввода при перезапуске

Таблица 1 — Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о вводе количества вершин | Верно |
| Попытка некорректного ввода числа вершин | Сообщение об ошибке | Верно |
| Вывод матрицы из 4 вершин | Построение матрицы смежности и вывод информации о графе | Верно |
| Вывод матрицы из 6 вершин и сохранение в файл | Построение матрицы смежности и вывод информации о графе. Сохранение в указанный файл | Верно |

Продолжение таблицы 1

| Описание теста | Ожидаемый результат | Полученный результат |
| --- | --- | --- |
| Вывод матрицы из 10 вершин и сохранение в файл | Построение матрицы смежности и вывод информации о графе. Сохранение в указанный файл | Верно |
| Попытка некорректного ввода при сохранении в файл | Сообщение об ошибке | Верно |
| Попытка некорректного ввода при перезапуске | Сообщение об ошибке | Верно |

В результате тестирования было выявлено, что программа успешно

проверяет данные на соответствие необходимым требованиям.

# Заключение

В процессе выполнения курсового проекта была разработана программа для поиска эйлерова цикла в неориентированном графе.

При выполнении курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания языка программирования С+ +.

Недостатком разработанной программы можно считать интерфейс, который не отображает интуитивно понятного представления графа из-за того.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Академия Яндекса. Основы C++ — [Электронный ресурс]. — Режим доступа: <https://academy.yandex.ru/handbook/cpp> (Дата обращения: 17.12.23)

2. CodeLessons — [Электронный ресурс]. — Режим доступа: [CodeLessons: уроки программирования](https://codelessons.dev/ru/) (Дата обращения: 17.12.23)

3. Нахождение Эйлерова пути за O (M) — [Электронный ресурс]. — Режим доступа: [MAXimal :: algo :: Нахождение Эйлерова пути или Эйлерова цикла (e-maxx.ru)](https://e-maxx.ru/algo/euler_path) (Дата обращения: 17.12.23)

# Приложение A. Листинг программы

#include<iostream>

#include<vector>

#include<conio.h>

#include<time.h>

#include<locale.h>

#include <stack>

#include <string>

#include <iomanip>

#include <fstream>

int num = 1;

bool tit = 0;

using namespace std;

void traverse(int u, bool visited[], const int VALUE, int\* graph[]);

bool isConnected(int\* graph[], const int VALUE);

int isEulerian(int\* graph[], vector<int>& degree, const int VALUE);

void create(int\* graph[], const int VALUE);

void createCopy(int\* graph[], int\* graphCopy[], int const VALUE);

void FindWay(vector<int>& deg, vector<int>& res, int\* graph[], const int VALUE);

void titul();

void count\_vertex(int \*count);

bool choiceFunc(int message);

void outputFile(int\* graph[], vector<int>& degree, vector<int>& res, int const VALUE, int check);

int main() {

system("cls");

setlocale(LC\_ALL, "RUS");

if (!tit)

titul();

srand((unsigned)time(0));

int VALUE = 0;

count\_vertex(&VALUE);

vector<int> deg(VALUE); // для хранения степени вершин

vector<int> res;

int\*\* graph = new int\* [VALUE];

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

{

graph[i] = new int[VALUE];

}

int\*\* graphCopy = new int\* [VALUE];

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

{

graphCopy[i] = new int[VALUE];

}

create(graph, VALUE);

createCopy(graph, graphCopy, VALUE);

int check = isEulerian(graph, deg, VALUE);

switch (check) {

case 0:

cout << "Граф не является Эйлеровым.";

break;

case 1:

cout << "Граф имеет Эйлеров цикл.\n";

FindWay(deg, res, graph, VALUE);

break;

case 2:

cout << "Граф имеет Эйлеров путь.\n";

FindWay(deg, res, graph, VALUE);

break;

default:

break;

}

if (choiceFunc(1))

{

outputFile(graphCopy, deg, res, VALUE, check);

}

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

{

delete[] graph[i];

delete[] graphCopy[i];

}

delete graph;

delete graphCopy;

if (choiceFunc(2))

{

num++;

main();

}

else

return 0;

}

void titul() {

printf("\n");

printf(" Министерство науки и высшего образования Российской Федерации\n");

printf(" Пензенский государственный университет\n");

printf(" Кафедра «Вычислительная техника»\n");

printf("\n\n");

printf(" КУРСОВОЙ ПРОЕКТ\n");

printf(" по дисциплине:\n ЛОГИКА И ОСНОВЫ АЛГОРИТМИЗАЦИИ В ИНЖЕНЕРНЫХ ЗАДАЧАХ\n");

printf(" тема проекта:\n РЕАЛИЗАЦИЯ АЛГОРИТМА НАХОЖДЕНИЯ ЭЙЛЕРОВЫХ ЦИКЛОВ\n ");

printf("\n\n\n");

printf(" Выполнил:\n");

printf(" студент группы 22ВВП1\n");

printf(" Харитонов А.А.\n");

printf("\n\n");

printf(" Принял:\n");

printf(" Акифьев И.В.\n");

printf("\n\n\n");

printf(" Пенза 2023\n\n\n");

cout << ">> Нажмите любую клавишу, чтобы продолжить...";

\_getch();

system("cls");

tit = 1;

}

void count\_vertex(int \*count)

{

bool flag = 0;

do

{

string value;

value.resize(5);

cout << "Граф №" << num << endl << endl;

cout << "Введите размерность графа [2; 10]: ";

getline(cin, value);

flag = 0;

if (value[0] == '0')

flag = 1;

for (int i = 0; i < value.size(); i++)

{

if (!isdigit(value[i]) || isspace(value[i]))

flag = 1;

}

if (!flag)

{

if (value.size() == 1 && std::atoi(value.c\_str()) == 1)

flag = 1;

if (std::atoi(value.c\_str()) > 10)

flag = 1;

if (value.size() == 0)

flag = 1;

}

if (flag == 1)

{

cout << "!!! Введите корректное число вершин!" << endl << endl;

cout << "> Нажмите любую клавишу, чтобы продолжить...";

\_getch();

system("cls");

}

else

{

\*count = std::atoi(value.c\_str());

break;

}

} while (flag);

}

bool choiceFunc(int message)

{

bool flag = 0;

do

{

string str;

str.resize(5);

switch (message)

{

case 1:

cout << "\n\nСохранить в файл? (Да - 1 / Нет - 0): ";

break;

case 2:

cout << "\n\nСоздать еще один граф? (Да - 1 / Нет - 0): ";

break;

default:

break;

}

getline(cin, str);

flag = 0;

for (int i = 0; i < str.size(); i++)

{

if (!isdigit(str[i]) || isspace(str[i]))

flag = 1;

}

if (!flag)

{

if (std::atoi(str.c\_str()) > 1)

flag = 1;

if (str.size() > 1 && str[0] == '0')

flag = 1;

if (str.size() == 0)

flag = 1;

}

if (flag)

{

cout << "!!! Введите корректное число!" << endl << endl;

cout << "> Нажмите любую клавишу, чтобы продолжить...";

\_getch();

system("cls");

}

else if (std::atoi(str.c\_str()) == 1)

return 1;

else

{

return 0;

}

} while (flag);

}

// проверка на связность

void traverse(int u, bool visited[], const int VALUE, int\* graph[]) {

visited[u] = true; //отмечаем вершины как посещенные

for (int v = 0; v < VALUE; v++) {

if (graph[u][v]) // проверка наличия путя

{

if (!visited[v]) traverse(v, visited, VALUE, graph); // если не посещалась, то переходим

}

}

}

bool isConnected(int\* graph[], const int VALUE) {

bool\* vis = new bool[VALUE];

//для всех вершин u в качестве начальной точки проверим, видны ли все узлы

for (int u = 0; u < VALUE; u++)

{

for (int i = 0; i < VALUE; i++) vis[i] = false; // инициализируем, поскольку узел не посещался

traverse(u, vis, VALUE, graph);

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

{

if (!vis[i]) //если есть узел, не посещенный обходом, граф не связан

return false;

}

}

return true;

}

// проверка на наличие Эйлерова пути или цикла

int isEulerian(int\* graph[], vector<int>& degree, const int VALUE) {

if (isConnected(graph, VALUE) == false) //когда граф не связан

return 0;

int oddDegree = 0;

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

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

if (graph[i][j])

degree[i]++; // увеличение степени, когда соединенный край найден

}

if (degree[i] % 2 != 0) //когда степень вершин нечетна

oddDegree++; // подсчет вершин нечетной степени

}

cout << "Степень вершин\n";

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

{

cout << "Вершина " << i << " - " << degree[i] << endl;

}cout << endl;

if (oddDegree > 2) // если > 2, то нет ни цикла, ни пути

return 0;

else if (oddDegree == 0) // если = 0, то цикл

return 1;

else // если = 2, то путь

return 2;

}

// создание графа

void create(int\* graph[], const int VALUE)

{

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

{

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

{

if (i == j) graph[i][j] = 0;

else if (j < i)

{

graph[i][j] = rand() % 2;

graph[j][i] = graph[i][j];

}

}

}

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

{

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

{

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

}cout << endl;

}cout << "\n";

}

void createCopy(int\* graph[], int\* graphCopy[], int const VALUE)

{

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

{

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

{

graphCopy[i][j] = graph[i][j];

}

}

}

// поиск пути

void FindWay(vector<int>& deg, vector<int>& res, int\* graph[], const int VALUE)

{

int first = 0;

while (!deg[first]) ++first;

int v1 = -1, v2 = -1;

bool bad = false;

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

if (deg[i] & 1)

{

if (v1 == -1)

v1 = i;

else if (v2 == -1)

v2 = i;

else

bad = true;

}

if (v1 != -1)

++graph[v1][v2], ++graph[v2][v1];

stack <int> st;

st.push(first);

while (!st.empty()) // пока st не пустой

{

int v = st.top(); // v = самому верхнему элементу в стеке

int i;

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

{

if (graph[v][i]) break;

}

if (i == VALUE)

{

res.push\_back(v);

st.pop();

}

else

{

--graph[v][i];

--graph[i][v];

st.push(i);

}

}

// для цикла не используется

if (v1 != -1)

{

for (size\_t i = 0; i + 1 < res.size(); ++i)

{

if (res[i] == v1 && res[i + 1] == v2 || res[i] == v2 && res[i + 1] == v1)

{

vector<int> res2;

for (size\_t j = i + 1; j < res.size(); ++j)

res2.push\_back(res[j]);

for (size\_t j = 1; j <= i; ++j)

res2.push\_back(res[j]);

res = res2;

break;

}

}

}

cout << "Путь:";

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

cout << " " << res[i] + 1;

}

void outputFile(int\* graph[], vector<int>& degree, vector<int>& res, const int VALUE, int check)

{

string fileName;

cout << "Введите название файла: ";

getline(cin, fileName);

ofstream fout;

fout.open(fileName);

if (!fout.is\_open())

{

cout << "Ошибка открытия файла!" << endl;

}

else

{

fout << "Количество вершин в графе: " << VALUE << endl << endl;

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

{

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

{

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

}fout << endl;

}fout << endl;

fout << "Степень вершин\n";

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

{

fout << "Вершина " << i << " - " << degree[i] << endl;

}fout << endl;

switch (check)

{

case 0:

fout << "Граф не является Эйлеровым.";

break;

case 1:

fout << "Граф имеет Эйлеров цикл." << endl;

fout << "Путь:";

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

fout << " " << res[i] + 1;

break;

case 2:

fout << "Граф имеет Эйлеров путь.\n";

fout << "Путь:";

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

fout << " " << res[i] + 1;

break;

default:

break;

}

}

fout.close();

}