

Содержание

[Реферат 5](#_Toc185602012)

[Введение 6](#_Toc185602013)

[1. Постановка задачи 7](#_Toc185602014)

[2. Описание алгоритма программы. 8](#_Toc185602015)

[3. Описание программы 11](#_Toc185602016)

[4. Тестирование 15](#_Toc185602017)

[Заключение 21](#_Toc185602018)

[Списоклитературы 22](#_Toc185602019)

[Приложение А. Листинг программы. 23](#_Toc185602020)

# Реферат

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, ЭЙЛЕРОВ ЦИКЛ, ЧЕТНОСТЬ СТЕПЕНЕЙ ВЕРШИН, СВЯЗНОСТЬ.

Отчет 28 страниц, 11 рисунков, 2 таблицы.

Цель исследования - разработать эффективную программу, способную подсчитывать степени вершин, для проверки графа на связность и четность, для нахождения Эйлерова цикла.

В работе необходимо рассмотреть правила заполнения матрицы. Выполнить проверку степеней вершин на четность и на отсутствие нулевых степеней вершин. Алгоритм выполнить через меню и функцию switch. Установить, что с помощью данного алгоритма можно проверить любой граф на наличие Эйлерова цикла в нем.

# Введение

В современном мире информационные технологии играют ключевую роль во многих сферах деятельности. Разработка и анализ алгоритмов являются основой для создания эффективных и надёжных программных решений. В данном курсовом проекте рассматривается реализация алгоритма нахождения эйлерова цикла, который является одним из фундаментальных алгоритмов в теории графов. Эйлеров цикл — это путь, проходящий через все рёбра графа ровно один раз. Этот алгоритм имеет множество применений в различных областях, таких как планирование маршрутов, оптимизация транспортных сетей и анализ социальных сетей.

В ходе выполнения курсового проекта будет создан код на выбранном языке программирования, который позволит находить эйлеровы циклы в графах различной структуры и размера. Результаты тестирования покажут эффективность предложенного алгоритма и его применимость в реальных задачах.

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

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым.

# Постановка задачи

Требуется разработать программу, которая реализует Эйлеров цикл, проверку на связность и четность степеней.

Исходный граф в программе должен задаваться матрицей смежности. Пользователь при генерации должен выбрать матрица будет ориентированная или не ориентированная. Так же граф может быть введен вручную или же скопирован из файла.

Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. Необходимо предусмотреть различные исходы обхода, чтобы программа не выдавала ошибок и работала эффективно.

Устройство ввода – клавиатура и мышь.

# Описание алгоритма программы.

Для реализации программы использовалось 5 функций: createG1 – для генерации ориентированной матрицы, create G2 – для генерации неориентированной матрицы, printG – для вывода матрицы, Euler – для самого Эйлерового цикла, dfs – для проверки на связность граф.

В начале программы пользователь встречает заставку с полезной информацией о курсовом проекте. Далее появляется меню, состоящее из 8 пунктов (где 0 – выход из программы, а 7 – сохранение результатов в файл).

Итак, после заполнения графа любым способом из предложенных (а именно: генерация ориентированного графа, генерация неориентированного графа, ручной ввод графа, ввод из файла) пользователь выбирает 5 пункт – непосредственно обход графа.

Вводит номер вершины, с которой начнется обход. Значение заносится в переменную start.

Инициализируется массив степеней. Далее выполняется проверка на четность всех вершин графа. Если хоть одна степень будет нечетная, то обход не начнется.

Если проверка пройдена, проверяется условие связности графа. Если будет хоть одна не посещенная вершина, то обход не начнется.

И после этого уже выполняется сам обход графа.

Ниже представлен псевдокод функции Euler(), частично main().

**Euler(graph, v, n, degree, result, index):**

1.Для u от 0 до n-1:

2. Если (graph[v][u] == 1):

3.Удалить ребро между v и u

4.Уменьшить степень вершин v и u

5.Euler(graph, u, n, degree, result, index)

6.конец условия

7. конец цикла

8. Добавить вершину v + 1 в результат

9.Увеличить индекс

10.Конец функции

**Main():**

1.Пока пользователь не выбрал "0" (выход):

2.Вывести меню:

0. Выход

1. Сгенерировать ориентированную матрицу

2. Сгенерировать неориентированную матрицу

3. Ввести матрицу из файла

4. Ввести матрицу вручную

5. Обход

6. Изменить матрицу

7. Сохранить результат в файл

3.Считать выбор пользователя (search)

4.Переключатель (switch) по значению search:

5.Случай 0:

6.Вывести сообщение "До свидания!" и выйти из цикла

7.Случай 1:

8. Сгенерировать ориентированную матрицу

9.Считать размер матрицы (n)

10.Если размер некорректный (n < 0), вывести ошибку и выйти

11.Создать ориентированную матрицу

12.Вывести матрицу

13.Случай 3:

14.Считать матрицу из файла

15.Считать имя файла

16.Открыть файл

17.Если файл не удалось открыть, вывести ошибку и выйти

18.Считать размер матрицы (n)

19.Считать матрицу из файла в graph

20.Закрыть файл

21.Вывести матрицу (printG(graph, n))

22.Случай 5:

23.Если матрица не существует, вывести ошибку и выйти

24.Скопировать матрицу в graph2

25.Считать начальную вершину (start)

26.Если начальная вершина некорректная, вывести ошибку и выйти

27.Проверить, что все вершины имеют чётные степени

28.Если хотя бы одна вершина имеет нечётную степень, вывести ошибку

29.Если граф Эйлеров:

30.Выполнить обход (Euler или Euler2)

31.Вывести результат обхода

32.Иначе:

33.Вывести сообщение, что граф не имеет Эйлерова цикла

# 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++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: createG1, createG2, printG, Euler, Euler2.

Подключаем библиотеки, первые две отвечают за стандартные функции, <time.h> отвечает за генерацию случайных чисел, <locale.h> отвечает за русский язык в программе.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

Создаем матрицу для ориентированного графа:

int\*\* createG1(int size) {

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

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

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

}

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

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

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

if (i == j) {

G[i][j] = 0;

}

}

}

return G;

}

Создаем матрицу для неориентированного графа:

int\*\* createG2(int size) {

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

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

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

}

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

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

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

if (i == j) {

G[i][j] = 0;

}

}

}

return G;

}

Выводим матрицу:

void printG(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

return;

}

Делаем обход матрицы для нахождения Эйлерова Цикла в неориентированном графе:

void Euler(int\*\* graph, int v, int n, int\* degree, int\* result, int\* index) {

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

if (graph[v][u] == 1) {

graph[v][u] = graph[u][v] = 0;

degree[v]--;

degree[u]--;

Euler(graph, u, n, degree, result, index);

}

}

result[(\*index)++] = v + 1; // Сохраняем вершину в результат

}

Делаем обход матрицы для нахождения Эйлерова Цикла в ориентированном графе:

void Euler2(int\*\* graph, int v, int n, int\* degree, int\* result, int\* index) {

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

if (graph[v][u] == 1) {

graph[v][u] = 0; // Удаляем ребро

degree[v]--; // Уменьшаем степень текущей вершины

Euler2(graph, u, n, degree, result, index); // Рекурсивно продолжаем обход

}

}

result[(\*index)++] = v + 1; // Сохраняем вершину в результат

}

В основной части алгоритма реализован цикл, который будет выводить меню и реализовывать функцию switch до тех пор, пока пользователь не выберет 0.

В case 1 вводится размер матрицы, генерируется ориентированная матрица, выводится матрица и флаг number устанавливается в 1.

В case 2 вводится размер матрицы, генерируется неориентированная матрица, выводится матрица и флаг number устанавливается в 0.

В case 3 вводится название файла, проверяется условие отсутствия такого файла, если условие выполнено, то матрица не будет создана, иначе вводится размер матрицы и ребра между вершинами, выводится матрица на экран и флаг устанавливается в 0.

В case 4 вводится размер матрицы, пользователь вводит 1 или 0 для каждого ребра, если будет введен иной символ, он начнет вводить символы заново, затем выводится матрица на экран и флаг устанавливается в 0.

В case 5 проверяется наличие матрицы, затем вводится стартовая вершина. Инициализируем флаг isEulerian равный 1, если флаг установлен в 1, то подсчитываем степень для вершин ориентированного графа, иначе для неориентированного. Далее условие четности вершин, если условие выполняется, то флаг isEulerian устанавливается в 0. Далее условие наличия нулевых вершин, если выполняется, то флаг isEulerian устанавливается в 0. Если флаг isEulerian установлен в 1, то проверяем флаг, если 1, то выполняем обход для ориентированного графа, иначе для неориентированного.

В case 6 проверяется наличие матрицы, затем вводится две вершины. Если флаг установлен в 1, то выполняем удаление или добавление ребра для ориентированного графа, иначе удаляем или добавляем два ребра в неориентированном графе. Далее пересчитываем степени вершин. Если остались нечетные, то выводим у каких вершин.

В case 7 проверяется наличие матрицы, вводится название файла. Затем открываем файл, сохраняем размер, далее матрицу, после результат обхода и закрываем файл.

Конец программы.

# 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. Для эффективного тестирования была создана таблица 1.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Описание | Предусловие | Тестирование | Ожидаемый результат |
| 1 | Работа меню | Программа запущена | Запускаем программу с помощью Visual Studio | Вывод в консоли меню |
| 2 | Выбор функции | Программа запущена | Вводим номер функции | Переход к выполнению функции |
| 3 | Создание неориентированного графа | Ввод 1 в меню | Ввод размера матрицы | Вывод матрицы неориентированного графа |
| 4 | Создание ориентированного графа | Ввод 2 в меню | Ввод размера матрицы | Вывод матрицы ориентированного графа |
| 5 | Загрузка графа из файла | Ввод 3 в меню | Ввод названия файла | Вывод матрицы |

Продолжение таблицы 1 – Описание поведения программы при тестировании

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Описание | Предусловие | Тестирование | Ожидаемый результат |
| 6 | Работа алгоритма | Создание матрицы смежности, ввод 4 в меню | Ввод стартовой вершины | Вывод в консоли результата алгоритма |
| 7 | Сохранение в файл результата | Создание матрицы смежности | Ввод названия файла сохранения | Появление в файле сохранения результатов |

При выборе 0, программа завершает свою работу. При выборе 1, генерируется ориентированная матрица (рисунок 1).

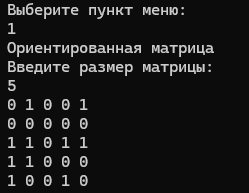


Рисунок 1 – пункт 1

При выборе пункта 2, генерируется неориентированная матрица (рисунок 2).

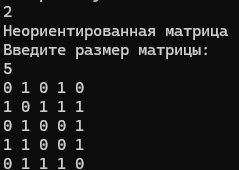


Рисунок 2 – пункт 2.

При выборе пункта 3, запросится название файла, и оттуда скопируется матрица (рисунок 3).

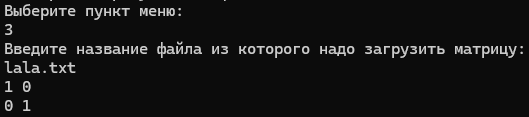


Рисунок 3 – пункт 3

При выборе пункта 4, пользователь вводит размер матрицы и затем поэлементно вводит 1/0 (рисунок 4).

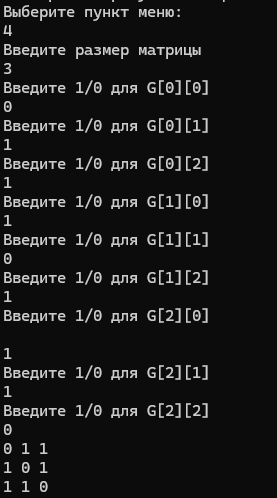


Рисунок 4 – пункт 4

При выборе пункта 5, пользователь вводит номер вершины, с которой начнется обход (рисунок 5).

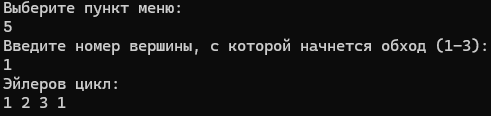


Рисунок 5- пункт 5

При выборе пункта 6, пользователю будет необходимо самостоятельно определить, как необходимо исправить граф, чтобы он имел Эйлеров цикл (рисунок 6).

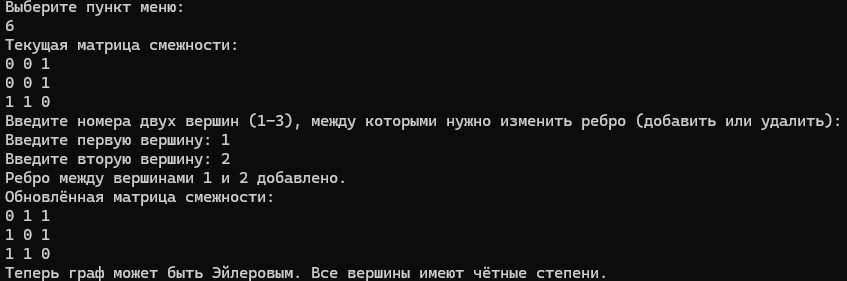
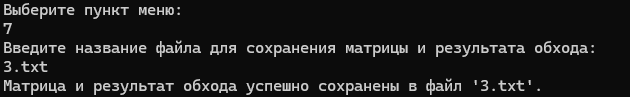


Рисунок 6 – пункт 6

При выборе пункта 7, пользователю необходимо ввести название файла, в который необходимо занести результат обхода (рисунок 7).

Рисунок 7 – пункт 7

Введем данные, неудовлетворяющие условиям существования графа. (рисунки 8 – 11)

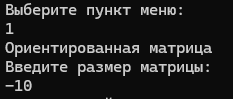


Рисунок 8 – Тестирование при вводе отрицательного размера.

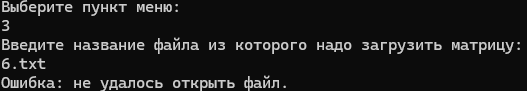


Рисунок 9 – Тестирование при вводе несуществующего файла.

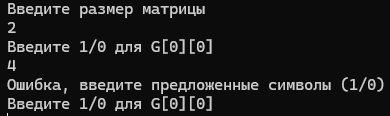


Рисунок 10 – Тестирование при вводе недопустимого символа

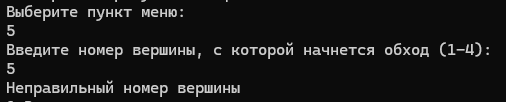
.

Рисунок 11 – Тестирование при вводе недопустимого номера вершин

Ниже можно ознакомиться с описанием поведения программы при тестировании (таблица 2).

Таблица 2 – Результаты поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Приветственная заставка, вывод меню | Верно |
| Выбор генерации ориентированной матрицы | Ввод размера, вывод матрицы | Верно |
| Выбор генерации неориентированной матрицы | Ввод размера, вывод матрицы | Верно |
| Выбор ввода матрицы из файла | Ввод названия файла, вывод матрицы | Верно |
| Выбор ввода матрицы вручную | Ввод ребер вручную для каждых вершин, вывод | Верно |

Продолжение таблицы 2 – Результаты поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Выбор обхода | Ввод стартовой вершины, вывод Эйлерова цикла | Верно |
| Выбор изменения матрицы | Ввод двух вершин, вывод изменённой матрицы | Верно |
| Выбор сохранения в файл | Ввод названия файла, ввод матрицы и результата | Верно |

# Заключение

В процессе создания проекта разработана программа, реализующая алгоритм нахождения Эйлерова цикла в MicrosoftVisualStudio 2022.

При выполнении данной курсовой работы были получены навыки разработки программы и освоены приемы создания матриц смежностей, обхода в глубины и реализации нахождения эйлерова цикла. Углублены знания языка программирования Си.

В работе рассмотрены правила заполнения матрицы. Выполнена проверка степеней вершин на четность и на отсутствие нулевых степеней вершин. Алгоритм выполнен через меню и функцию switch. Установлено, что с помощью данного алгоритма можно проверить любой граф на наличие Эйлерова цикла в нем.

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.

# Списоклитературы

1. Баранов, С. В. Алгоритмы и структуры данных. — М.: Наука, 2020. — 384 с.
2. Гусев, В. Л. Основы программирования на языке C. — М.: ДМК Пресс, 2021. — 400 с.
3. Кузнецов, С. В. Графы и алгоритмы. — Самара: Самарский гос. университет, 2019. — 450 с.
4. Лебедев, А. П. Программирование на C для начинающих. — М.: Велигор, 2021. — 350 с.
5. Николаев, И. В. Введение в теорию графов. — Екатеринбург: Урал. университет, 2021. — 300 с.
6. Павлов, Р. И. Эйлеровы циклы. Алгоритмическое исследование. — Ростов: ЮФУ, 2018. — 150 с.
7. Петров, А. Н. Алгоритмы для обработки графов. — М.: Высшая школа, 2019. — 275 с.
8. Смирнов, И. Н. Основы создания программ. — М.: Гуманитарный издательский центр, 2021. — 500 с.
9. Тихонов, А. В. Современное программирование на C. — М.: БХВ-Петербург, 2022. — 480 с.

# Приложение А. Листинг программы.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int\*\* createG1(int size) {

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

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

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

}

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

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

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

if (i == j) {

G[i][j] = 0;

}

}

}

return G;

}

int\*\* createG2(int size) {

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

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

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

}

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

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

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

if (i == j) {

G[i][j] = 0;

}

}

}

return G;

}

void printG(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

return;

}

void Euler(int\*\* graph, int v, int n, int\* degree, int\* result, int\* index) {

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

if (graph[v][u] == 1) {

graph[v][u] = graph[u][v] = 0;

degree[v]--;

degree[u]--;

Euler(graph, u, n, degree, result, index);

}

}

result[(\*index)++] = v + 1; // Сохраняем вершину в результат

}

void Euler2(int\*\* graph, int v, int n, int\* degree, int\* result, int\* index) {

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

if (graph[v][u] == 1) {

graph[v][u] = 0; // Удаляем ребро

degree[v]--; // Уменьшаем степень текущей вершины

Euler2(graph, u, n, degree, result, index); // Рекурсивно продолжаем обход

}

}

result[(\*index)++] = v + 1; // Сохраняем вершину в результат

}

int main() {

setlocale(LC\_ALL, "Rus");

printf("Курсовая работа\nПо дисциплине ЛиОАВИЗ\nНа тему: Реализация Эйлерова цикла\nВыполнил студент 23ВВВ2: Герасимов В. Р.\nПриняли: Юрова О. В., Митрихин М. А.\n\n");

system("PAUSE");

system("cls");

int n = 5, start, search = 1, number = 0;

int\* degree = (int\*)malloc(n \* sizeof(int));

int\*\* graph = NULL;

int\*\* graph2 = NULL;

int\* result = (int\*)malloc(n \* n \* sizeof(int)); // Массив для хранения результата обхода

int index = 0; // Индекс для записи в массив результата

while (search != 0) {

printf("0.Выход\n");

printf("1.Сгенерировать ориентированную матрицу\n");

printf("2.Сгенерировать неориентированную матрицу\n");

printf("3.Ввести матрицу из файла\n");

printf("4.Ввести матрицу вручную\n");

printf("5.Обход\n");

printf("6.Изменить матрицу\n");

printf("7.Сохранить результат в файл\n");

printf("Выберите пункт меню:\n");

scanf\_s("%d", &search);

switch (search) {

case 0: {

printf("До свидания! Я рассчитываю на оценку 5! \n");

break;

}

case 1: {

printf("Ориентированная матрица\n");

printf("Введите размер матрицы: \n");

scanf\_s("%d", &n);

if (n < 0) {

printf("Недопустимый размер\n");

break;

}

graph = (int\*\*)malloc(n \* sizeof(int\*));

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

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

}

graph = createG1(n);

printG(graph, n);

number = 1;

break;

}

case 2: {

printf("Неориентированная матрица\n");

printf("Введите размер матрицы: \n");

scanf\_s("%d", &n);

if (n < 0) {

printf("Недопустимый размер\n");

break;

}

graph = (int\*\*)malloc(n \* sizeof(int\*));

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

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

}

graph = createG2(n);

printG(graph, n);

number = 0;

break;

}

case 3: {

printf("Введите название файла из которого надо загрузить матрицу:\n");

char filename[100];

scanf\_s("%s", filename, (unsigned)\_countof(filename));

FILE\* file;

fopen\_s(&file, filename, "r");

if (file == NULL) {

printf("Ошибка: не удалось открыть файл.\n");

break;

}

else {

fscanf\_s(file, "%d", &n);

graph = (int\*\*)malloc(n \* sizeof(int\*));

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

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

}

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

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

fscanf\_s(file, "%d", &graph[i][j]);

}

}

}

fclose(file);

printG(graph, n);

number = 0;

break;

}

case 4: {

int timevar;

printf("Введите размер матрицы\n");

scanf\_s("%d", &n);

if (n < 0) {

printf("Недопустимый размер\n");

break;

}

graph = (int\*\*)malloc(n \* sizeof(int\*));

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

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

}

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

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

printf("Введите 1/0 для G[%d][%d]\n", i, j);

scanf\_s("%d", &timevar);

if (timevar == 1 || timevar == 0) {

graph[i][j] = timevar;

}

else {

printf("Ошибка, введите предложенные символы (1/0)\n");

i = 0;

j = -1;

}

}

}

printG(graph, n);

number = 2;

break;

}

case 5: {

if (graph == NULL) {

printf("Ошибка: матрица не существует.\n");

break;

}

printf("Введите номер вершины, с которой начнется обход (1-%d): \n", n);

scanf\_s("%d", &start);

start -= 1;

int isEulerian = 1;

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

degree[i] = 0;

if (number == 1) {

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

degree[i] += graph[i][j] + graph[j][i];

}

}

else {

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

degree[i] += graph[i][j];

}

}

if (degree[i] % 2 != 0) {

isEulerian = 0;

printf("Вершина %d имеет нечётную степень.\n", i + 1);

break;

}

}

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

if (degree[i] == 0) {

printf("Вершина %d имеет нулевую степень.\n", i + 1);

isEulerian = 0;

break;

}

}

if (isEulerian) {

printf("Эйлеров цикл:\n");

if (number == 1) {

Euler2(graph, start, n, degree, result, &index);

}

else {

Euler(graph, start, n, degree, result, &index);

}

for (int i = index - 1; i >= 0; i--) {

printf("%d ", result[i]);

}

printf("\n");

}

else {

printf("Граф не имеет Эйлерова цикла.\n");

}

break;

}

case 6: {

if (graph == NULL) {

printf("Ошибка: матрица не существует. Сначала создайте или загрузите матрицу.\n");

break;

}

int v1, v2;

printf("Текущая матрица смежности:\n");

printG(graph, n);

printf("Введите номера двух вершин (1-%d), между которыми нужно изменить ребро (добавить или удалить):\n", n);

printf("Введите первую вершину: ");

scanf\_s("%d", &v1);

printf("Введите вторую вершину: ");

scanf\_s("%d", &v2);

// Преобразуем номера вершин в индексы массива

v1 -= 1;

v2 -= 1;

// Проверяем корректность ввода

if (v1 < 0 || v1 >= n || v2 < 0 || v2 >= n) {

printf("Ошибка: некорректные номера вершин.\n");

break;

}

if (number == 1) {

if (graph[v1][v2] == 1) {

graph[v1][v2] = 0;

printf("Ребро между вершинами %d и %d удалено.\n", v1 + 1, v2 + 1);

}

else {

graph[v1][v2] = 1;

printf("Ребро между вершинами %d и %d добавлено.\n", v1 + 1, v2 + 1);

}

}

else {

// Изменяем ребро: если оно есть, удаляем; если его нет, добавляем

if (graph[v1][v2] == 1) {

graph[v1][v2] = 0;

graph[v2][v1] = 0;

printf("Ребро между вершинами %d и %d удалено.\n", v1 + 1, v2 + 1);

}

else {

graph[v1][v2] = 1;

graph[v2][v1] = 1;

printf("Ребро между вершинами %d и %d добавлено.\n", v1 + 1, v2 + 1);

}

}

if (number == 1) {

// Пересчитываем степени вершин

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

degree[i] = 0;

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

degree[i] += graph[i][j] + graph[j][i];

}

}

}

else {

// Пересчитываем степени вершин

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

degree[i] = 0;

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

degree[i] += graph[i][j];

}

}

}

// Выводим обновлённую матрицу смежности

printf("Обновлённая матрица смежности:\n");

printG(graph, n);

// Проверяем, стали ли все степени чётными

int isEulerian = 1;

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

if (degree[i] % 2 != 0) {

isEulerian = 0;

printf("Вершина %d имеет нечётную степень: %d\n", i + 1, degree[i]);

}

}

if (isEulerian) {

printf("Теперь граф может быть Эйлеровым. Все вершины имеют чётные степени.\n");

}

else {

printf("Граф всё ещё не является Эйлеровым. Исправьте степени вершин.\n");

}

break;

}

case 7: {

if (graph == NULL) {

printf("Ошибка: матрица не существует. Сначала создайте или загрузите матрицу.\n");

break;

}

printf("Введите название файла для сохранения матрицы и результата обхода:\n");

char filename[100];

scanf\_s("%s", filename, (unsigned)\_countof(filename));

FILE\* file;

fopen\_s(&file, filename, "w");

if (file == NULL) {

printf("Ошибка: не удалось открыть файл для записи.\n");

break;

}

else {

// Сохраняем размер матрицы

fprintf(file, "Размер матрицы: %d\n", n);

// Сохраняем матрицу

fprintf(file, "Матрица:\n");

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

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

fprintf(file, "%d ", graph2[i][j]);

}

fprintf(file, "\n");

}

if (result == NULL) {

printf("Ошибка: не был совершен обход.\n");

break;

}

// Сохраняем результат обхода

fprintf(file, "Результат обхода:\n");

for (int i = index - 1; i >= 0; i--) { // Записываем результат в обратном порядке

fprintf(file, "%d ", result[i]);

}

fprintf(file, "\n");

fclose(file);

}

printf("Матрица и результат обхода успешно сохранены в файл '%s'.\n", filename);

break;

}

default: {

printf("Неправильный выбор:\n");

break;

}

}

}

return 0;

}