

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

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проектированию
по курсу «Логика и основы алгоритмизации в инженерных задачах»
на тему «Реализация алгоритма нахождения Эйлеровых циклов»

24.12.24
отлично
Фед

Выполнил:

Студент группы 23BBB2

Стрельцов А.П.

Приняли:

Юрова О. В.

Митрохин М.А.

Пенза 2024

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ _____

« ____ » _____ 20 ____

ЗАДАНИЕ

на курсовое проектирование по курсу

«Логика и основы алгоритмизации в инженерных задачах»
Студенту Стрепылову Александру Павловичу Группа 23ВВВ2
Тема проекта Реализация алгоритма нахождения Эйлерова цикла

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии с данным заданием курсового проекта.

Пояснительная записка должна содержать:

1. Постановку задачи;
2. Теоретическую часть задания;
3. Описание алгоритма поставленной задачи;
4. Пример ручного расчета задачи и вычислений (на небольшом участке работы алгоритма);
5. Описание самой программы;
6. Тесты;
7. Список литературы;
8. Листы программы;
9. Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схема

3. Экспериментальная часть

Тестирование программы;
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

- 1 Изучение тематической части курса
- 2 Разработка алгоритма программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания " 6 " сентября

Дата защиты проекта " " "

Руководитель Юрова О.В.

Задание получил " 6 " сентября 20 24 г.

Студент Стрелцов А.И.

Содержание

Реферат.....	5
Введение	6
1. Постановка задачи	7
2. Теоретическая часть задания.....	8
3. Описание алгоритма программы.....	11
4. Описание программы.....	20
5. Тестирование.....	22
6. Ручной расчет задачи.....	27
Заключение.....	29
Список литературы	30
Приложение А. Листинг программы.	31

Реферат

Отчет 38 стр, 12 рисунков.
ЭЙЛЕРОВЫЙ ЦИКЛ, РЕАЛИЗАЦИЯ ЭЙЛЕРОВА ЦИКЛА В ОРИЕНТИРОВАННЫХ И
НЕОРИЕНТИРОВАННЫХ ГРАФАХ.

Цель исследования – разработка программы, которая будет реализовывать алгоритм Эйлера цикла в ориентированных и неориентированных графах.

В работе рассмотрены правила и условия нахождения Эйлера цикла.

Определены основные свойства графов, в которых существует Эйлеров цикл:

- Для неориентированных графов необходимо, чтобы граф был связным и степени всех вершин были четными.
- Для ориентированных графов необходимо, чтобы граф был сильно связным, а для каждой вершины количество входящих ребер равнялось количеству исходящих.

Изучены алгоритмы нахождения Эйлера цикла, алгоритм на основе поиска в глубину (DFS).

Реализация алгоритма была выполнена с использованием языка программирования Си. Рассмотрены особенности работы алгоритма для графов различного размера.

Программа была протестирована на различных примерах, чтобы подтвердить её корректность и эффективность.

Введение

Эйлеров цикл – одно из ключевых понятий теории графов, раздела математики, изучающего свойства и структуры графов. Понятие было введено швейцарским математиком Леонардом Эйлером в 1736 году при решении знаменитой задачи о Кенигсбергских мостах. Эта задача стала первым историческим примером использования графов для анализа реальной проблемы.

Эйлеров цикл – это путь в графе, который проходит через каждое ребро ровно один раз и возвращается в начальную вершину. Граф, в котором существует Эйлеров цикл, называется Эйлеровым. Определение таких графов и изучение их свойств играют важную роль как в теоретической математике, так и в прикладных задачах, включая логистику, проектирование сетей, анализ маршрутов и моделирование биологических процессов.

Для определения наличия Эйлерова цикла в графе достаточно проверить два условия:

1. Граф должен быть связным (все вершины связаны между собой, если игнорировать направления рёбер в ориентированном графе).
2. Степень каждой вершины (число рёбер, соединённых с вершиной) должна быть чётной.

Понимание и анализ Эйлеровых циклов позволяют решать множество практических задач, от оптимизации маршрутов в транспортных сетях до анализа молекулярных структур в химии.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, 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

1. Постановка задачи

Требуется разработать программу для нахождения Эйлеровых циклов в ориентированных и неориентированных графах, используя алгоритм поиска в глубину.

Программа должна принимать от пользователя количество вершин для генерации матрицы смежности графа, автоматически создавать эту матрицу с учетом граничных условий, таких как отсутствие рёбер или наличие изолированных вершин, и выводить сформированную матрицу, визуальное представление графа и все компоненты сильной связности. При этом необходимо реализовать алгоритм поиска в глубину, который будет корректно обрабатывать различные исходы поиска, избегая ошибок и обеспечивая стабильную работу программы, а также предусмотреть удобный интерфейс для пользователя с возможностью ввода данных с клавиатуры и мыши.

2. Теоретическая часть задания

Эйлеров цикл — это путь в графе, который проходит по всем рёбрам ровно один раз и возвращается в начальную вершину. Для нахождения Эйлерова цикла в графах необходимо учитывать их тип: ориентированные или неориентированные.

Условия существования Эйлерова цикла:

1. Неориентированные графы:

Граф имеет Эйлеров цикл, если все вершины имеют чётную степень. Это условие гарантирует, что для каждой вершины можно войти и выйти из неё, не оставляя рёбер неиспользованными.

2. Ориентированные графы:

Граф имеет Эйлеров цикл, если для каждой вершины количество входящих рёбер равно количеству исходящих рёбер. Это означает, что для каждой вершины можно войти и выйти из неё, что также позволяет пройти по всем рёбрам.

Алгоритм нахождения Эйлерова цикла для неориентированных графов:

1. Проверка условий

Убедитесь, что все вершины имеют чётную степень.

2. Выбор начальной вершины

Начните с любой вершины.

3. Поиск цикла

Используйте алгоритм обхода (например, DFS) для нахождения цикла, удаляя рёбра по мере их использования.

Если в процессе обхода вы достигли вершины, из которой можно продолжить, продолжайте до тех пор, пока не будут использованы все рёбра.

Граф содержит Эйлеров цикл: $1 \Rightarrow 4 \Rightarrow 3 \Rightarrow 5 \Rightarrow 2 \Rightarrow 1$

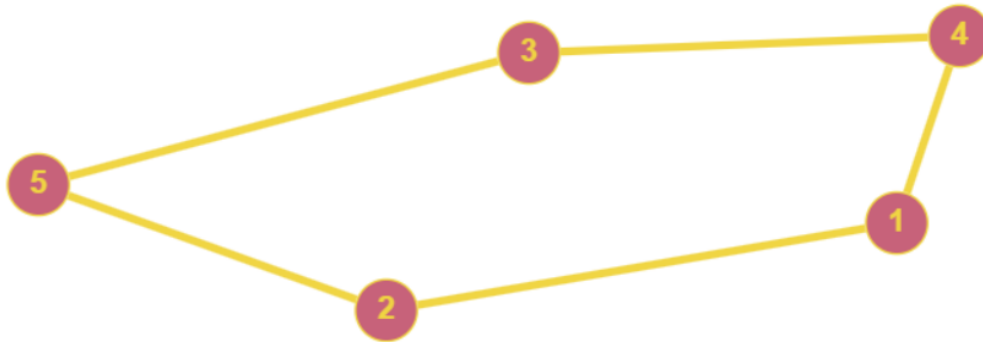


Рисунок 1 – ЭЦ в неорграфе.

Для ориентированных графов:

1. Проверка условий

Убедитесь, что для каждой вершины количество входящих рёбер равно количеству исходящих.

2. Выбор начальной вершины

Начните с любой вершины, у которой есть исходящие рёбра.

3. Поиск цикла

Используйте алгоритм обхода (например, DFS) для нахождения цикла, удаляя рёбра по мере их использования.

Если вы достигли вершины, из которой можно продолжить, продолжайте до тех пор, пока не будут использованы все рёбра.

Граф содержит Эйлеров цикл: $1 \Rightarrow 4 \Rightarrow 3 \Rightarrow 5 \Rightarrow 2 \Rightarrow 1$

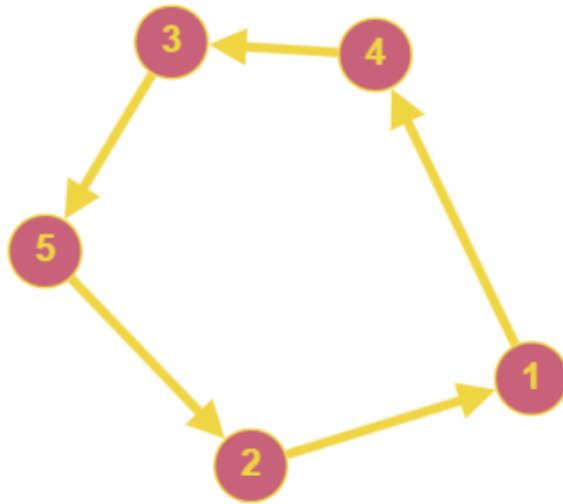


Рисунок 2 – ЭЦ в орграфе.

3. Описание алгоритма программы

Проверка наличия Эйлера цикла:

Алгоритм выполняет следующие шаги для ориентированных и неориентированных графов.

Неориентированный граф

1. Проверка четности степеней вершин

У каждой вершины степень (число смежных рёбер) должна быть четной. Если хотя бы у одной вершины степень нечетная, цикл невозможен.

2. Проверка связности

Для графа необходимо проверить, связаны ли его компоненты. Для этого используется обход в глубину (DFS). Если существует хотя бы одна вершина с ненулевой степенью, но она недостижима из начальной вершины, граф не связан.

Ориентированный граф

1. Равенство входящих и исходящих степеней

Для каждой вершины проверяется, чтобы количество входящих рёбер (in-degree) совпадало с количеством исходящих рёбер (out-degree). Если это условие нарушается, цикл невозможен.

2. Проверка связности в двух направлениях

Для сохранения исходной структуры графа создается временная копия матрицы смежности.

Если оба условия выполняются, граф содержит Эйлеров цикл.

Нахождение Эйлера цикла:

Если Эйлеров цикл существует, программа его строит с помощью алгоритма, основанного на методе Флери или итеративном обходе с использованием стека:

1. Копирование графа

Для сохранения исходной структуры графа создается временная копия матрицы смежности.

2. Поиск цикла

Используется стек для хранения пути. Начальная вершина помещается в стек. Алгоритм проходит по рёбрам графа, удаляя их после использования. Если из текущей вершины нельзя продолжать путь, она извлекается из стека и добавляется в результат.

3. Вывод результата

После завершения обхода в массиве хранится порядок вершин, образующих Эйлеров цикл.

Структура программы:

1. Функция dfs

Выполняет обход графа в глубину, проверяя достижимость вершин.

2. Функция hasEulerianCycle

Определяет, содержит ли граф Эйлеров цикл. Учитываются различия для ориентированных и неориентированных графов.

3. Функция findEulerianCycle

Осуществляет построение самого Эйлера цикла с использованием временной матрицы смежности и стека.

Программа сначала проверяет существование Эйлера цикла, а затем выводит его, если он есть.

Ниже представлен псевдокод функций `dfs`, `hasEulerianCycle`, `findEulerianCycle`:

`_dfs`

1. Функция `DFS`(вершина `v`, массив посещенных `visited`, матрица смежности `matrix`, размер `size`)
 2. установить `visited[v] = 1`
 3. для `i = 0` пока `i < size` делать
 4. если `matrix[v][i] > 0` И `visited[i] == 0` тогда
 5. вызвать `DFS(i, visited, matrix, size)`
 6. конец условия
 7. конец цикла
8. конец функции

`_hasEulerianCycle`

1. Функция `hasEulerianCycle`(матрица `matrix`, размер `size`, направленный `directed`)
 2. создать массив `visited` размером `size` и инициализировать его нулями
 3. если `directed` тогда
 4. создать массив `inDegree` размером `size` и инициализировать его нулями

5. создать массив outDegree размером size и инициализировать его нулями

6. для i от 0 до size - 1 делать

7. для j от 0 до size - 1 делать

8. если matrix[i][j] > 0 тогда

9. outDegree[i] = outDegree[i] + 1

10. inDegree[j] = inDegree[j] + 1

11. конец условия

12. конец цикла

13. конец цикла

14. для i от 0 до size - 1 делать

15. если inDegree[i] != outDegree[i] тогда

16. освободить память visited, inDegree, outDegree

17. вернуть 0

18. конец условия

19. конец цикла

20. освободить память inDegree, outDegree

21. найти startVertex с ненулевой степенью

22. если startVertex == -1 тогда

23. освободить память visited
24. вернуть 1
25. конец условия
26. вызвать DFS(startVertex, visited, matrix, size)
27. для i от 0 до size делать
 28. если degree > 0 И !visited[i] тогда
 29. освободить память visited
 30. вернуть 0 // Граф не связан
 31. конец условия
32. конец цикла
33. сбросить массив visited
34. создать обратную матрицу reverseMatrix размером size
35. для i от 0 до size - 1 делать
 36. для j от 0 до size делать
 37. если matrix[i][j] > 0 тогда
 38. reverseMatrix[j][i] = 1
 39. конец условия
 40. конец цикла
41. конец цикла
42. вызвать DFS(startVertex, visited, reverseMatrix, size)
43. для i от 0 до size - 1 делать

44. если $\text{degree} > 0$ И $!\text{visited}[i]$ тогда
 45. освободить память `reverseMatrix`, `visited`
 46. вернуть 0
 47. конец условия
48. конец цикла
49. освободить память `reverseMatrix`
50. иначе
 51. для i от 0 до $\text{size} - 1$ делать
 52. если $\text{degree} \% 2 \neq 0$ тогда
 53. освободить память `visited`
 54. вернуть 0
 55. конец условия
 56. конец цикла
 57. найти `startVertex` с ненулевой степенью
 58. если $\text{startVertex} == -1$ тогда
 59. освободить память `visited`
 60. вернуть 1
 61. конец условия
 62. вызвать `DFS(startVertex, visited, matrix, size)`
 63. для i от 0 до size делать
 64. если $\text{degree} > 0$ И $!\text{visited}[i]$ тогда

65. освободить память visited

66. вернуть 0 // Граф не связан

67. конец условия

68. конец цикла

69. освободить память visited

70. вернуть 1

71. конец функции

_findEulerianCycle

1. Функция findEulerianCycle(матрица matrix, размер size, направленный directed)

2. создать стек stack размером size * size

3. создать массив cycle размером size * size

4. установить top = -1, cycleIndex = 0, current = 0

5. если stack или cycle == NULL тогда

6. вывести "Ошибка выделения памяти."

7. завершить программу с кодом ошибки

8. конец условия

9. создать временную матрицу tempMatrix размером size

10. для i от 0 до size - 1 делать

11. выделить память для tempMatrix[i] размером size

12. скопировать данные из matrix[i] в tempMatrix[i]

13. конец цикла

14. `stack[++top] = current`

15. пока `top >= 0` делать

16. `current = stack[top]`

17. установить `found = 0`

18. для `i` от 0 до `size` делать

19. если `tempMatrix[current][i] > 0` тогда

20. `stack[++top] = i`

21. если `directed` тогда

22. `tempMatrix[current][i]--`

23. иначе

24. `tempMatrix[current][i]--`

25. `tempMatrix[i][current]--`

26. конец условия

27. `found = 1`

28. прервать цикл

29. конец условия

30. конец цикла

31. если `!found` тогда

32. `cycle[cycleIndex++] = current`

33. top--

34. конец условия

35. конец цикла

36. вывести "Эйлеров цикл: "

37. для i от cycleIndex - 1 до 0 делать

38. вывести cycle[i]

39. конец цикла

40. вывести новую строку

41. освободить память stack

42. освободить память cycle

43. для i от 0 до size - 1 делать

44. освободить память tempMatrix[i]

45. конец цикла

46. освободить память tempMatrix

47. Конец функции

Полный код программы можно увидеть в Приложении А.

4. Описание программы

курсач.sln представляет собой консольное приложение на языке C, предназначенное для работы с графами, представленными в виде матриц смежности. Она включает в себя функционал для создания, редактирования, анализа и сохранения графов с проверкой наличия Эйлера цикла.

Данная программа является многомодульной, поскольку состоит из нескольких функций:

1. **generateRandomMatrix** — Создает случайный граф (ориентированный или неориентированный), заполняя его матрицу смежности случайными связями между вершинами.
2. **manualInputMatrix** — Позволяет пользователю вручную ввести матрицу смежности графа.
3. **writeMatrixToFile** — Сохраняет матрицу смежности графа в файл.
4. **readMatrixFromFile** — Считывает матрицу смежности из файла и проверяет её на наличие Эйлера цикла.
5. **hasEulerianCycle** — Проверяет, содержит ли граф Эйлеров цикл.
6. **findEulerianCycle** — Реализует алгоритм нахождения Эйлера цикла (обход графа с использованием стека).
7. **promptMatrixEdit** — Позволяет пользователю вручную отредактировать матрицу смежности.
8. **dfs** — Реализует обход графа в глубину (DFS) для проверки связности.
9. **freeMatrix** — Освобождает память, выделенную для матрицы графа.

Работа программы начинается с вывода меню. Дальнейшее описание состояний программы реализовано в таблице (Таблица 1).

Таблица 1. – Описание состояний программы

Клавиши, вызывающее событие	Действие пользователя	Действие программы
1,Enter	Генерация рандомной матрицы G графа	Запускается диалог выбора типа графа(ориентированный или неориентированный), после чего предлагается выбрать количество вершин в генерируемой матрице смежности
2,Enter	Ручной ввод матрицы G графа	Запускается диалог “Введите размер матрицы”, далее необходимо ввести матрицу смежности вручную
3,Enter	Запись матрицы в файл	Запускается диалог, где необходимо ввести имя файла, в котором будет храниться последняя сгенерированная матрица
4,Enter	Получение данных из файла	Запускается диалог, где необходимо ввести имя файла, из которого будет извлечена матрица смежности
5,Enter	Выход	Выход из программы

5. Тестирование

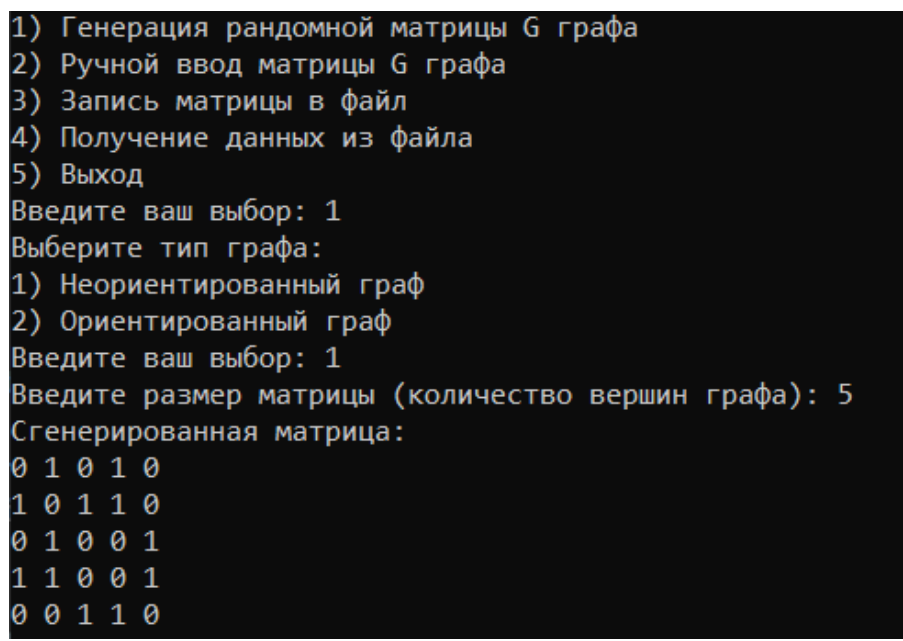
Для разработки программы была выбрана среда Microsoft Visual Studio 2022, которая предоставляет все необходимые инструменты для написания, отладки и тестирования кода на языке C. В процессе разработки активно использовались такие возможности Visual Studio, как точки останова, трассировка выполнения программы, анализ значений переменных и управление памятью.

Процесс тестирования программы

Тестирование программы проводилось на всех этапах разработки, начиная с реализации отдельных функций и заканчивая проверкой работы программы в целом. Основное внимание уделялось следующим аспектам:

1. Работа с матрицами смежности:

Проверялась корректность генерации случайных матриц для ориентированных и неориентированных графов.



```
1) Генерация случайной матрицы G графа
2) Ручной ввод матрицы G графа
3) Запись матрицы в файл
4) Получение данных из файла
5) Выход
Введите ваш выбор: 1
Выберите тип графа:
1) Неориентированный граф
2) Ориентированный граф
Введите ваш выбор: 1
Введите размер матрицы (количество вершин графа): 5
Сгенерированная матрица:
0 1 0 1 0
1 0 1 1 0
0 1 0 0 1
1 1 0 0 1
0 0 1 1 0
```

Рисунок 3 – тестирование создания случайной ориентированной матрицы.

```

1) Генерация рандомной матрицы G графа
2) Ручной ввод матрицы G графа
3) Запись матрицы в файл
4) Получение данных из файла
5) Выход
Введите ваш выбор: 1
Выберите тип графа:
1) Неориентированный граф
2) Ориентированный граф
Введите ваш выбор: 2
Введите размер матрицы (количество вершин графа): 5
Сгенерированная матрица:
0 1 0 0 1
0 0 1 0 1
1 0 0 1 0
0 1 0 0 0
0 0 1 1 0

```

Рисунок 4 – тестирование создания случайной неориентированной матрицы.

2. Работа с файлами:

Проверялась корректность записи и чтения матриц из файлов.

```

Обновленная матрица:
0 0 1 0 1
0 0 0 1 1
1 0 0 1 0
0 1 1 0 0
1 1 0 0 0
Матрица теперь содержит Эйлеров цикл.
Граф содержит Эйлеров цикл. Поиск цикла:
Эйлеров цикл: 0 2 3 1 4 0
Меню:
1) Генерация рандомной матрицы G графа
2) Ручной ввод матрицы G графа
3) Запись матрицы в файл
4) Получение данных из файла
5) Выход
Введите ваш выбор: 3
Введите имя файла: graph.txt
Матрица записана в файл graph.txt.

```

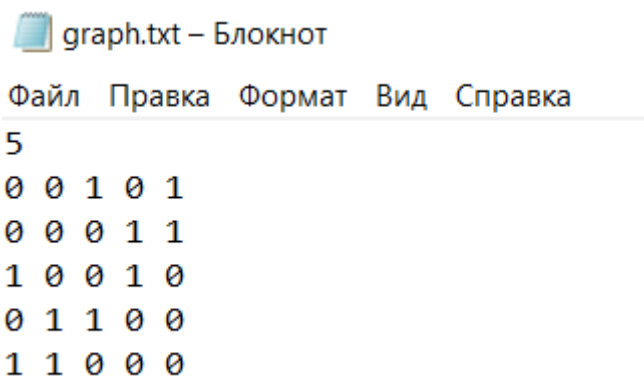
Рисунок 5 – тестирование записи матрицы смежности в файл.

```

Меню:
1) Генерация случайной матрицы G графа
2) Ручной ввод матрицы G графа
3) Запись матрицы в файл
4) Получение данных из файла
5) Выход
Введите ваш выбор: 4
Введите имя файла: graph.txt
Матрица считана из файла graph.txt.
Считанная матрица:
0 0 1 0 1
0 0 0 1 1
1 0 0 1 0
0 1 1 0 0
1 1 0 0 0
Граф содержит Эйлеров цикл. Поиск цикла:
Эйлеров цикл: 0 2 3 1 4 0

```

Рисунок 6 – тестирование чтения матрицы смежности из файла.



graph.txt – Блокнот

Файл Правка Формат Вид Справка

```

5
0 0 1 0 1
0 0 0 1 1
1 0 0 1 0
0 1 1 0 0
1 1 0 0 0

```

Рисунок 7 – файл, с помощью которого производится запись/чтение матриц смежности.

3. Проверка наличия и нахождения Эйлера цикла:

Тестировались графы с заранее известным наличием или отсутствием Эйлера цикла. Проверялась корректность алгоритмов для ориентированных и неориентированных графов.

```
Сгенерированная матрица:
0 0 1 0 1
0 0 1 1 1
1 1 0 0 1
0 1 0 0 0
1 1 1 0 0
Граф не содержит Эйлеров цикл. Хотите отредактировать матрицу вручную? (1 - Да, 0 - Нет): 1
Редактирование матрицы вручную.
Введите новую матрицу размером 5 x 5:
0 0 0 1 1
0 0 1 0 1
0 1 0 0 1
1 0 0 0 1
1 1 1 1 0
Обновленная матрица:
0 0 0 1 1
0 0 1 0 1
0 1 0 0 1
1 0 0 0 1
1 1 1 1 0
Матрица теперь содержит Эйлеров цикл.
Граф содержит Эйлеров цикл. Поиск цикла:
Эйлеров цикл: 0 3 4 1 2 4 0
```

Рисунок 8 – тестирование корректности алгоритма проверки и нахождения алгоритма Эйлера цикла в неориентированном графе.

```
2) Ориентированный граф
Введите ваш выбор: 2
Введите размер матрицы (количество вершин графа): 5
Сгенерированная матрица:
0 1 0 0 0
0 0 0 1 1
0 1 0 0 0
0 0 0 0 0
0 0 1 0 0
Граф не содержит Эйлеров цикл. Хотите отредактировать матрицу вручную? (1 - Да, 0 - Нет): 1
Редактирование матрицы вручную.
Введите новую матрицу размером 5 x 5:
0 1 0 0 0
0 0 0 1 1
0 1 0 0 0
1 0 0 0 0
0 0 1 0 0
Обновленная матрица:
0 1 0 0 0
0 0 0 1 1
0 1 0 0 0
1 0 0 0 0
0 0 1 0 0
Матрица теперь содержит Эйлеров цикл.
Граф содержит Эйлеров цикл. Поиск цикла:
Эйлеров цикл: 0 1 4 2 1 3 0
```

Рисунок 9 – тестирование корректности алгоритма проверки и нахождения алгоритма Эйлера цикла в ориентированном графе.

Результаты тестирования

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям (Таблица 2).

Таблица 2 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Генерация рандомной матрицы G графа	Запускается диалог выбора типа графа(ориентированный или неориентированный), после чего предлагается выбрать количество вершин в генерируемой матрице смежности.	Верно
Запись матрицы в файл	Запускается диалог, где необходимо ввести имя файла, в котором будет храниться последняя сгенерированная матрица.	Верно
Получение данных из файла	Запускается диалог, где необходимо ввести имя файла, из которого будет извлечена матрица смежности.	Верно
Проверка наличия и нахождения Эйлера цикла	Корректный вывод Эйлера цикла в ориентированных и неориентированных графах.	Верно

6. Ручной расчет задачи

Проведем проверку, правильно ли алгоритм находит Эйлеров цикл посредством ручных вычислений на примере матрицы смежности, сгенерированной программой. При генерации случайной матрицы смежности вероятность того, что она подходит под условие Эйлера цикла, крайне мала, поэтому мы будем редактировать её вручную. На отредактированной матрице проверим правильность алгоритма нахождения Эйлера цикла (рис.10).

```
Введите ваш выбор: 1
Выберите тип графа:
1) Неориентированный граф
2) Ориентированный граф
Введите ваш выбор: 1
Введите размер матрицы (количество вершин графа): 5
Сгенерированная матрица:
0 0 1 1 0
0 0 0 1 0
1 0 0 1 0
1 1 1 0 0
0 0 0 0 0
Граф не содержит Эйлеров цикл. Хотите отредактировать матрицу вручную? (1 - Да, 0 - Нет): 1
Редактирование матрицы вручную.
Введите новую матрицу размером 5 x 5:
0 0 1 0 1
0 0 0 1 1
1 0 0 1 0
0 1 1 0 0
1 1 0 0 0
Обновленная матрица:
0 0 1 0 1
0 0 0 1 1
1 0 0 1 0
0 1 1 0 0
1 1 0 0 0
Матрица теперь содержит Эйлеров цикл.
Граф содержит Эйлеров цикл. Поиск цикла:
Эйлеров цикл: 0 2 3 1 4 0
```

Рисунок 10 – матрица для тестирования.

Отобразили в графическом редакторе Paint нашу матрицу и представили её в виде графа (рис.11). Совершили обход графа, пронумеровав шаги.

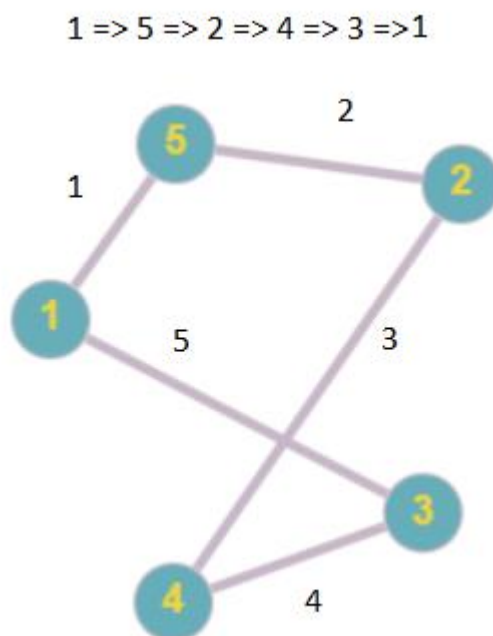


Рисунок 11 – расчет работы алгоритма вручную.

После сравнения результатов нахождения Эйлера цикла вручную и с помощью программы мы получили идентичные результаты. Можно сделать вывод, что работа алгоритма совершена правильно (рис.12).

Эйлеров цикл: 0 2 3 1 4 0

=

$1 \Rightarrow 5 \Rightarrow 2 \Rightarrow 4 \Rightarrow 3 \Rightarrow 1$

Рисунок 12 – сравнения результатов.

Заключение

Таким образом, в процессе выполнения данной курсовой работы была разработана программа, реализующая алгоритм нахождения Эйлеровых циклов в графах. В ходе работы над проектом были получены навыки программирования и освоены методы работы с графами, включая создание матриц смежности и проверку графа на эйлеровость.

При реализации алгоритма были углублены знания о теории графов и алгоритмах, что позволило не только понять принципы работы Эйлеровых циклов, но и применить их на практике. Программа демонстрирует основные функции, необходимые для поиска Эйлеровых циклов как в ориентированных, так и в неориентированных графах.

Однако, стоит отметить, что разработанная программа имеет некоторые ограничения, такие как примитивный пользовательский интерфейс, работающий в консольном режиме. Это упрощает процесс разработки, но не добавляет удобства для конечного пользователя. Тем не менее, функционал программы является достаточным для выполнения поставленных задач и может быть использован в учебных целях.

В будущем возможно улучшение интерфейса и расширение функциональности программы, что сделает её более удобной и многофункциональной для пользователей.

Список литературы

1. Язык Си: Б.В. Керниган, Д.М. Ричи - Санкт-Петербург, Невский диалект, 2003г. - 355 с.
2. Как программировать на С: Харви Дейтел, Пол Дейтел - Москва, Бином-пресс, 2006 г. - 512 с.
3. Ресурсы электронной библиотеки <http://msdn.microsoft.com/>
4. Лекции по теории графов / Под ред. В.А. Емеличева., О.Н. Мельникова, В.И. Сарванова, Р.И. Тышкевич. - Москва, Наука, Гл. ред. физ.-мат. лит., 1990г. - 384 с.
5. Основы теории графов: Зыков А.А. - Москва, Наука, 1987г. - 381 с.

Приложение А.

Листинг программы.

```
define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>
#include <string.h>

// Объявление функций
void generateRandomMatrix(int*** matrix, int* size, int directed);
void manualInputMatrix(int*** matrix, int* size);
void writeMatrixToFile(int** matrix, int size, const char* filename);
void readMatrixFromFile(int*** matrix, int* size, const char* filename);
int hasEulerianCycle(int** matrix, int size, int directed);
void findEulerianCycle(int** matrix, int size, int directed);
void freeMatrix(int** matrix, int size);
void promptMatrixEdit(int*** matrix, int* size, int directed);

int main() {
    setlocale(LC_ALL, "RUS");
    int** matrix = NULL;
    int directed = 0;
    int size = 0;
    int choice;
    char filename[50];

    do {
        printf("Меню:\n");
        printf("1) Генерация случайной матрицы G графа\n");
        printf("2) Ручной ввод матрицы G графа\n");
        printf("3) Запись матрицы в файл\n");
        printf("4) Получение данных из файла\n");
        printf("5) Выход\n");
        printf("Введите ваш выбор: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                freeMatrix(matrix, size);
                printf("Выберите тип графа:\n");
                printf("1) Неориентированный граф\n");
                printf("2) Ориентированный граф\n");
                printf("Введите ваш выбор: ");
                int graphType;
                scanf("%d", &graphType);
                directed = (graphType == 2);
                generateRandomMatrix(&matrix, &size, directed);

                if (!hasEulerianCycle(matrix, size, directed)) {
                    printf("Граф не содержит Эйлера цикл. Хотите отредактировать матрицу вручную? (1 - Да, 0 - Нет): ");
                    int editChoice;
                    scanf("%d", &editChoice);
                    if (editChoice == 1) {
                        promptMatrixEdit(&matrix, &size, directed);
                    }
                }
                break;
            }
            case 2:
                freeMatrix(matrix, size);
```

```

    manualInputMatrix(&matrix, &size);
    break;

case 3:
    if (matrix) {
        printf("Введите имя файла: ");
        scanf("%s", filename);
        writeMatrixToFile(matrix, size, filename);
    }
    else {
        printf("Матрица не задана!\n");
    }
    break;

case 4:
    printf("Введите имя файла: ");
    scanf("%s", filename);
    freeMatrix(matrix, size);
    readMatrixFromFile(&matrix, &size, filename);
    if (matrix) {
        printf("Считанная матрица:\n");
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                printf("%d ", matrix[i][j]);
            }
            printf("\n");
        }
        if (hasEulerianCycle(matrix, size, directed)) {
            printf("Граф содержит Эйлеров цикл. Поиск цикла:\n");
            findEulerianCycle(matrix, size, directed);
        }
        else {
            printf("Граф не содержит Эйлеров цикл.\n");
        }
    }
    break;

case 5:
    printf("Выход из программы.\n");
    break;

default:
    printf("Неверный выбор, попробуйте снова.\n");
}

if ((choice == 1 || choice == 2) && matrix) {
    if (hasEulerianCycle(matrix, size, directed)) {
        printf("Граф содержит Эйлеров цикл. Поиск цикла:\n");
        findEulerianCycle(matrix, size, directed);
    }
    else {
        printf("Граф не содержит Эйлеров цикл.\n");
    }
}
} while (choice != 5);

freeMatrix(matrix, size);
return 0;
}

// Функция, позволяющая редактировать матрицу вручную
void promptMatrixEdit(int*** matrix, int* size, int directed) {
    printf("Редактирование матрицы вручную.\n");
    printf("Введите новую матрицу размером %d x %d:\n", *size, *size);
    for (int i = 0; i < *size; i++) {

```



```

        for (int j = 0; j < *size; j++) {
            scanf("%d", &(*matrix)[i][j]);
        }
    }

    printf("Обновленная матрица:\n");
    for (int i = 0; i < *size; i++) {
        for (int j = 0; j < *size; j++) {
            printf("%d ", (*matrix)[i][j]);
        }
        printf("\n");
    }

    if (hasEulerianCycle(*matrix, *size, directed)) {
        printf("Матрица теперь содержит Эйлеров цикл.\n");
    }
    else {
        printf("Матрица все еще не содержит Эйлеров цикл.\n");
    }
}

void generateRandomMatrix(int*** matrix, int* size, int directed) {
    printf("Введите размер матрицы (количество вершин графа): ");
    scanf("%d", size);

    // Выделяем память под матрицу
    *matrix = (int**)malloc(*size * sizeof(int*));
    for (int i = 0; i < *size; i++) {
        (*matrix)[i] = (int*)calloc(*size, sizeof(int));
    }

    srand((unsigned int)time(NULL));

    // Генерация случайного графа
    if (directed) {
        for (int i = 0; i < *size; i++) {
            for (int j = 0; j < *size; j++) {
                if (i != j && rand() % 2) {
                    // Добавляем ребро только в одном направлении
                    if ((*matrix)[i][j] == 0 && (*matrix)[j][i] == 0) {
                        (*matrix)[i][j] = 1; // Ребро из i в j
                    }
                }
            }
        }
    }
    else {
        // Для неориентированного графа
        for (int i = 0; i < *size; i++) {
            for (int j = i + 1; j < *size; j++) {
                if (rand() % 2) {
                    (*matrix)[i][j] = 1;
                    (*matrix)[j][i] = 1;
                }
            }
        }
    }

    // Вывод матрицы
    printf("Сгенерированная матрица:\n");
    for (int i = 0; i < *size; i++) {
        for (int j = 0; j < *size; j++) {
            printf("%d ", (*matrix)[i][j]);
        }
        printf("\n");
    }
}

```

```

    }
}

// Ввод матрицы вручную
void manualInputMatrix(int*** matrix, int* size) {
    printf("Введите размер матрицы (количество вершин графа): ");
    scanf("%d", size);

    *matrix = (int**)malloc(*size * sizeof(int*));
    for (int i = 0; i < *size; i++) {
        (*matrix)[i] = (int*)malloc(*size * sizeof(int));
    }

    printf("Введите матрицу смежности:\n");
    for (int i = 0; i < *size; i++) {
        for (int j = 0; j < *size; j++) {
            scanf("%d", &(*matrix)[i][j]);
        }
    }
}

void writeMatrixToFile(int** matrix, int size, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (!file) {
        printf("Ошибка открытия файла для записи.\n");
        return;
    }

    fprintf(file, "%d\n", size);
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            fprintf(file, "%d ", matrix[i][j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
    printf("Матрица записана в файл %s.\n", filename);
}

void readMatrixFromFile(int*** matrix, int* size, const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        printf("Ошибка открытия файла для чтения.\n");
        return;
    }

    fscanf(file, "%d", size);
    *matrix = (int**)malloc(*size * sizeof(int*));
    for (int i = 0; i < *size; i++) {
        (*matrix)[i] = (int*)malloc(*size * sizeof(int));
    }

    for (int i = 0; i < *size; i++) {
        for (int j = 0; j < *size; j++) {
            fscanf(file, "%d", &(*matrix)[i][j]);
        }
    }
    fclose(file);
    printf("Матрица считана из файла %s.\n", filename);
}

// Функция для обхода графа в глубину (DFS)
void dfs(int v, int* visited, int** matrix, int size) {
    visited[v] = 1;

```

```

for (int i = 0; i < size; i++) {
    if (matrix[v][i] > 0 && !visited[i]) {
        dfs(i, visited, matrix, size);
    }
}
}

// Проверка наличия Эйлера цикла
int hasEulerianCycle(int** matrix, int size, int directed) {
    int* visited = (int*)calloc(size, sizeof(int));

    if (directed) {
        // Для ориентированного графа проверяем равенство входящих и исходящих степеней
        int* inDegree = (int*)calloc(size, sizeof(int));
        int* outDegree = (int*)calloc(size, sizeof(int));

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (matrix[i][j] > 0) {
                    outDegree[i]++;
                    inDegree[j]++;
                }
            }
        }

        for (int i = 0; i < size; i++) {
            if (inDegree[i] != outDegree[i]) {
                free(visited);
                free(inDegree);
                free(outDegree);
                return 0; // Неравенство степеней исключает Эйлеров цикл
            }
        }

        free(inDegree);
        free(outDegree);

        // Проверяем связность графа
        int startVertex = -1;
        for (int i = 0; i < size; i++) {
            int degree = 0;
            for (int j = 0; j < size; j++) {
                degree += matrix[i][j];
            }
            if (degree > 0) { // Ищем вершину с ненулевой степенью
                startVertex = i;
                break;
            }
        }

        if (startVertex == -1) {
            free(visited);
            return 1; // Граф пуст, но считается, что Эйлеров цикл есть
        }

        // Проверяем связность
        dfs(startVertex, visited, matrix, size);
        for (int i = 0; i < size; i++) {
            int degree = 0;
            for (int j = 0; j < size; j++) {
                degree += matrix[i][j];
            }
        }
        if (degree > 0 && !visited[i]) {
            free(visited);
            return 0; // Граф не связан
        }
    }
}

```

```

    }
}

// Проверяем связность по входящим рёбрам
memset(visited, 0, size * sizeof(int)); // Сбрасываем массив visited
int** reverseMatrix = (int**)malloc(size * sizeof(int*));
for (int i = 0; i < size; i++) {
    reverseMatrix[i] = (int*)calloc(size, sizeof(int));
}
for (int i = 0; i < size; i++) {
    for (int j = 0; j < size; j++) {
        if (matrix[i][j] > 0) {
            reverseMatrix[j][i] = 1; // Обратное направление
        }
    }
}

dfs(startVertex, visited, reverseMatrix, size);
for (int i = 0; i < size; i++) {
    int degree = 0;
    for (int j = 0; j < size; j++) {
        degree += matrix[i][j];
    }
    if (degree > 0 && !visited[i]) {
        for (int k = 0; k < size; k++) {
            free(reverseMatrix[k]);
        }
        free(reverseMatrix);
        free(visited);
        return 0; // Граф не связан в обратном направлении
    }
}

for (int k = 0; k < size; k++) {
    free(reverseMatrix[k]);
}
free(reverseMatrix);
}
else {
    // Для неориентированного графа проверяем чётность степеней всех вершин
    for (int i = 0; i < size; i++) {
        int degree = 0;
        for (int j = 0; j < size; j++) {
            degree += matrix[i][j];
        }
        if (degree % 2 != 0) {
            free(visited);
            return 0; // Нечётная степень исключает Эйлеров цикл
        }
    }
}

// Проверяем связность графа
int startVertex = -1;
for (int i = 0; i < size; i++) {
    int degree = 0;
    for (int j = 0; j < size; j++) {
        degree += matrix[i][j];
    }
    if (degree > 0) { // Ищем вершину с ненулевой степенью
        startVertex = i;
        break;
    }
}

if (startVertex == -1) {

```

```

    free(visited);
    return 1; // Граф пуст, но считается, что Эйлеров цикл есть
}

dfs(startVertex, visited, matrix, size);
for (int i = 0; i < size; i++) {
    int degree = 0;
    for (int j = 0; j < size; j++) {
        degree += matrix[i][j];
    }
    if (degree > 0 && !visited[i]) {
        free(visited);
        return 0; // Граф не связан
    }
}

free(visited);
return 1; // Граф связан, и условия для Эйлера цикла выполнены
}

// Нахождение Эйлера цикла
void findEulerianCycle(int** matrix, int size, int directed) {
    int* stack = (int*)malloc((size_t)size * (size_t)size * sizeof(int)); // Стек для пути
    int* cycle = (int*)malloc((size_t)size * (size_t)size * sizeof(int)); // Массив для хранения цикла
    int top = -1, cycleIndex = 0, current = 0;

    if (!stack || !cycle) {
        fprintf(stderr, "Ошибка выделения памяти.\n");
        exit(EXIT_FAILURE);
    }

    // Копируем матрицу
    int** tempMatrix = (int**)malloc(size * sizeof(int*));
    for (int i = 0; i < size; i++) {
        tempMatrix[i] = (int*)malloc(size * sizeof(int));
        memcpy(tempMatrix[i], matrix[i], size * sizeof(int));
    }

    stack[++top] = current;
    while (top >= 0) {
        current = stack[top];
        int found = 0;
        for (int i = 0; i < size; i++) {
            if (tempMatrix[current][i] > 0) {
                // Проходим по ребру
                stack[++top] = i;
                if (directed) {
                    tempMatrix[current][i]--;
                }
                else {
                    tempMatrix[current][i]--;
                    tempMatrix[i][current]--;
                }
                found = 1;
                break;
            }
        }
        if (!found) {
            // Если нет соседей, добавляем вершину в цикл
            cycle[cycleIndex++] = current;
            top--;
        }
    }
}

```

```

// Выводим Эйлеров цикл
printf("Эйлеров цикл: ");
for (int i = cycleIndex - 1; i >= 0; i--) {
    printf("%d ", cycle[i]);
}
printf("\n");

// Освобождаем память
free(stack);
free(cycle);
for (int i = 0; i < size; i++) {
    free(tempMatrix[i]);
}
free(tempMatrix);
}

// Освобождение памяти
void freeMatrix(int** matrix, int size) {
    if (matrix) {
        for (int i = 0; i < size; i++) {
            free(matrix[i]);
        }
        free(matrix);
    }
}

```