

Министерство 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»

Выполнили студенты группы 24ВВВ4:
Чернышевский Е.И. ____

 Хорошо
30.12.2025

Принял к.э.н., доцент:
Акифьев И.В. ____

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

« » 20

ЗАДАНИЕ

на курсовое проектирование по курсу

«Безопасность и основы алгоритмизации в инж. задачах»
Студенту Чернышова Ивана Игоревича Группа 21 ВРВЧ
Тема проекта Задача нахождения парами
чисел

Исходные данные (технические требования) на проектирование

Разработать алгоритм и программное обеспечение
в соответствии с данными задания курсового проекта
Краткое содержание задания:

1. Изучить задание;
2. Изучить задачу;
3. Описать алгоритм решения задачи;
4. Привести пример расчета задачи и вычисления;
5. Описать способ программирования;
6. Исходные;
7. Список литературы;
8. Исходные программы;
9. Результаты работы программы.

Объем работы по курсу

1. Расчетная часть

Этот расчет работы алгоритма

2. Графическая часть

Схема алгоритма в форме блок-схемы

3. Экспериментальная часть

Тестирование программы;
результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

- 1 Изучение предметной области
- 2 Разработка алгоритма программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление итоговой работы
- 6
- 7
- 8

Дата выдачи задания " " "

Дата защиты проекта " " "

Руководитель Андреев Илья Владимирович

Задание получил " 12 " сентября 2025 г.

Студент Чернышевский Евгений Игоревич

Содержание

Реферат	5
Введение	6
1. Постановка задачи.....	7
2. Теоретическая часть задания	8
3. Описание алгоритма программы	11
4. Описание программы	12
5. Тестирование	16
6. Ручной расчет задачи	22
Заключение	24
Список литературы.....	25
Приложение А	26

Реферат

Отчет 30 стр, 16 рисунков, 1 таблица, 1 приложение.

ЭЙЛЕРОВ ЦИКЛ, ЭЙЛЕРОВ ПУТЬ, ГРАФ, ОБХОД В ГЛУБИНУ, СТЕК

Цель исследования – разработка программы, способная выделить эйлеров цикл (путь) в графе, используя стек и основные правила обхода графа.

В работе рассмотрены правила и принципы нахождения Эйлера цикла (пути).

Реализованы основные требования для курсового проекта.

Введение

В современном мире существуют множество задач, которые требуют рационального подхода к решению, например прорисовка фигур и диаграмм, в которых прохождение соединения не будет повторяться, не отрывая карандаша. Таким, представляющим собой важную концепцию в теории графов, является алгоритм нахождения Эйлеровых циклов (путей).

В рамках данной курсовой работы будет рассмотрено нахождение Эйлерова цикла (пути) в графе с использованием языков программирования C/C++. Такой цикл (путь) назван в честь Леонарда Эйлера – математика, который описал решение задачи о 7 мостах Кёнигсберга. Открытие Эйлера нашло применение в различных областях: математика, информатика, инженерия, маршрутизация, проектирование сетей, логистика, игровой дизайн и т.д.

В процессе нахождения Эйлерова цикла (пути) будут использованы основные принципы программирования, такие как работа с графами, динамическими массивами (векторами), циклами и функциями.

Целью данной работы является углубленное изучение Эйлеровых циклов и путей, а также развитие навыков программирования.

Таким образом, изучение нахождения Эйлерова цикла позволит углубить свои знания в программировании, а также научиться применять их на практике для разработки полезных программных инструментов.

1. Постановка задачи

Необходимо разработать программу для работы с графом. Программа должна иметь: задание размерности, в которой должны быть определены граничные значения, случайную генерацию, ручной ввод графа. Необходимо построить способ при котором возможна корректировка графа для формирования цикла. Также результат текущих вычислений программа должна вычислять в файл. Необходимо сделать обработку и проверку программы на наличие ошибок.

Многомодульность программы. Программа должна быть поделена на логические модули. Это упростит поиск ошибок при отладке и тестировании программы, а также позволит легко расширять функционал программы.

Режим работы – текстовый.

Устройство ввода-вывода – клавиатура и мышь. Необходимо различать и идентифицировать действия, произведенные с их помощью, это облегчит использование программы.

Интерфейс должен быть построен на основе меню. Это необходимо для создания интуитивно понятного интерфейса для пользователя. Заставка необходима для того, чтобы пользователь, запустивший программу, смог получить достаточную информацию о ней.

Таким образом требуется реализовать следующие возможности при работе с графом:

1. Задание размерности
2. Случайная генерация
3. Ручной ввод из файла
4. Ориентированный/неориентированный граф
5. Сохранение в файл результатов.
6. Корректировка графа для нахождения Эйлера цикла

2. Теоретическая часть задания

Цикл в графе называется эйлеровым, если он содержит все ребра графа. Связный граф, в котором есть эйлеров цикл, называется эйлеровым графом. Такой граф можно нарисовать, не отрывая карандаша от бумаги и не повторяя линий.

При анализе можно рассмотреть задачу о мостах Кёнигсберга. Основной загадкой для жителей был способ прохождения всех мостов, не проходя ни по одному из них дважды. На упрощённой схеме части города (рисунок 1) мостам соответствуют линии, а частям города — точки соединения линий.

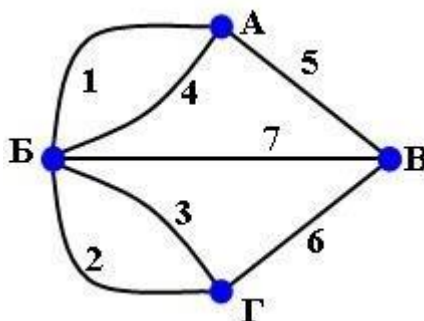


Рисунок 1—Задача о Кёнигсбергских мостах

В ходе рассуждений Эйлер пришёл к следующим выводам: Число нечётных вершин (вершин, к которым ведёт нечётное число рёбер) графа всегда чётно. Невозможно начертить граф, который имел бы нечётное число нечётных вершин. Граф кёнигсбергских мостов имел четыре нечётные вершины, следовательно невозможно пройти по всем мостам, не проходя ни по одному из них дважды.

Теорема. Связный неориентированный граф является эйлеровым тогда и только тогда, когда степени всех его вершин четны.

Справедлива и обратная теорема:

Теорема. Связный неориентированный граф содержит эйлеров цикл тогда и только тогда, когда число вершин нечётной степени равно нулю.

Рассмотрим один из алгоритмов проверки графа на возможность построения алгоритма Эйлера. Этот алгоритм был основан на обходе графа в глубину, используя стек.

Пусть дан неориентированный граф G , удовлетворяющий условию теоремы Эйлера. Рассмотрим алгоритм нахождения эйлера цикла в этом графе. Этот алгоритм основан на обходе графа в глубину. При переходе в очередную вершину будем удалять соответствующее, пройденное ребро. При обнаружении вершины, из которой не выходят рёбра (мы их удалили ранее при обходе в глубину), будем записывать её номер в стек. Обнаружение вершины с нулевым числом рёбер говорит о том, что найден цикл. Его можно удалить, при этом чётность степеней вершин не изменится. Процесс продолжается до тех пор, пока есть непройденные рёбра. После окончания обхода в глубину всего графа в стеке будут записаны номера вершин графа в порядке, соответствующем эйлерову циклу (Примеры Эйлеровых циклов приведены на рисунке 2).

1. DFS
2. Вход: граф $Graph$, представленный матрицей смежности; стек вершин эйлера цикла $Stack$, номер стартовой вершины U .
3. Начало. Для всех вершин V графа $Graph$, смежных с U : $Graph[U, V] := 0$, $Graph[V, U] := 0$; DFS($Graph$, $Stack$, V).
Push($Stack$, U).
4. Конец.

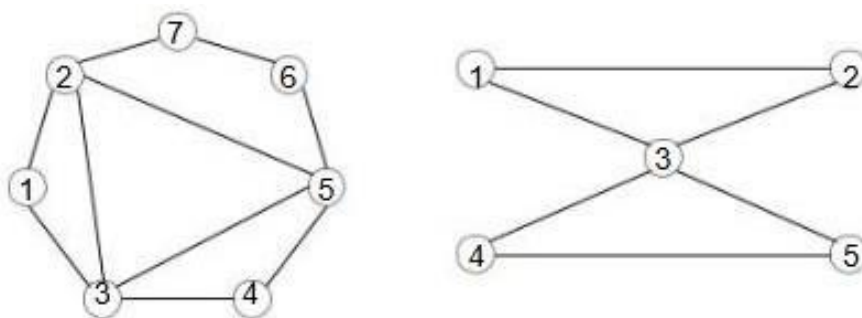


Рисунок 2 – примеры Эйлеровых графов

Можно выделить ещё несколько способов решения задачи Эйлера:

- Алгоритм Флери:

1) С произвольной вершины u присваиваем ребру $\{u, v\}$ номер 1, вычеркиваем его и переходим в вершину v .

2) Вершина w , в которую мы пришли, имеет номер k для предыдущего ребра. Выбираем любое инцидентное ребро к w , но оно выбирается только при отсутствии других вариантов. Номер этому ребру присваиваем $k + 1$ и вычеркиваем его. Процесс продолжается, пока все ребра графа не будут занумерованы.

- Алгоритм Тэрри:

1) Начинаем движение с произвольной вершины графа.

2) Находясь в произвольной вершине, выбираем направление дальнейшего движения, исходя из двух правил запрета: нельзя проходить по ребру дважды в одном направлении, если есть другие возможности, нельзя выбирать ребро, которое привело в данную вершину в первый раз.

3) Проходя выбранное ребро, фиксируем (например, стрелкой) направление, по которому движемся, и заносим его в список.

4) Попадая в вершину впервые, отмечаем (например, крестом) ребро, по которому пришли.

5) Если пройдены дважды не все ребра, возвращаемся к п.2. Практически это означает, что построение будет продолжаться до тех пор, пока имеется возможность движения, и закончится в начальной вершине.

3. Описание алгоритма программы.

Рассмотрим порядок основных операций для проверки и поиска Эйлера цикла в программе:

1. Функция `ziklkarochiliput`:

- Принимает граф (в виде матрицы смежности) и стартовую вершину.
- Подсчитывает степень каждой вершины (количество смежных рёбер).
- Определяет количество вершин с нечётной степенью (переменная `oddCount`).
- Запоминает последнюю найденную вершину с нечётной степенью в переменной `start`.
- Возвращает `true`, если: все вершины имеют чётную степень (граф имеет Эйлера цикл), или ровно две вершины имеют нечётную степень (граф имеет Эйлера путь).

2. Функция `poiskziclaeler`

- Принимает граф и начальную вершину для поиска.
- Инициализирует стек для хранения вершин и вектор для хранения найденного цикла.
- Начинает добавления стартовой вершины в стек.

3. Поиск Эйлера цикла/пути

Цикл продолжается, пока стек не станет пустым:

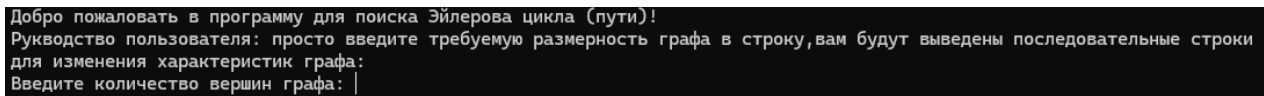
- Извлекает вершину из стека.
- Проверяет наличие смежных рёбер:
- Если рёбер нет, добавляет вершину в цикл и удаляет её из стека.
- Если рёбра есть, выбирает одно, уменьшает его количество в графе, и добавляет смежную вершину в стек.

4. Вывод результата

4. Описание программы

Программа предназначена для определения наличия Эйлера цикла и возможного вывода пути Эйлера при наличии

После запуска будет выведена приветственная заставка и руководство пользователя (рисунок 3).



Добро пожаловать в программу для поиска Эйлера цикла (пути)!
Руководство пользователя: просто введите требуемую размерность графа в строку, вам будут выведены последовательные строки для изменения характеристик графа:
Введите количество вершин графа: |

Рисунок 3 – ручной ввод матрицы смежности и вывод цикла Эйлера

Присутствует возможность случайного формирования и последующего вывод матрицы смежности неориентированного графа и цикла Эйлера в случае нахождения.

Принцип взаимодействия пользователем выглядит следующим образом

- 1) Для начала работы вас попросят ввести количество нужных вам вершин в графе.
- 2) Далее вы выбираете способ ввода инцидентности для вершин, доступны случайная генерация и ручной ввод пользователем. В случае ручного ввода, вам потребуется ввести вершины в формате («вершина 1» «вершина 2») для окончания ввода введите «-1 -1» (рисунок 4)
- 3) Выбор ориентированности графа

```

Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 1
1 – Ориентированный, 2 – Не ориентированный граф2
Введите связь (формат: вершина1 вершина2). Введите '-1 -1' для завершения:
1 2
2 3
3 4
-1 -1
Матрица смежности:
  1 2 3 4 5
1 0 1 0 0 0
2 1 0 1 0 0
3 0 1 0 1 0
4 0 0 1 0 0
5 0 0 0 0 0
Матрица успешно записана в graph.txt

Эйлеров цикл: 1 2 3 4

Введите у для продолжения, n – для выхода

```

Рисунок4–ручнойвводматрицысмежностиивыводциклаЭйлера

Все выведенные данные будут последовательно сохраняться в файл.

Далее представлен пример случайной генерации в неориентированном графе. В случае невозможности построения цикла Эйлера, программа пробует добавить связь между нечетными вершинами и таким образом сформировать цикл (рисунок 5).

```

Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 2
1 – Ориентированный, 2 – Не ориентированный граф2
Матрица смежности:
  1 2 3 4 5
1 0 0 1 0 1
2 0 0 0 1 0
3 1 0 0 0 1
4 0 1 0 0 1
5 1 0 1 1 0
Матрица успешно записана в graph.txt

Эйлеров цикл: 2 4 5 3 1 5

Введите у для продолжения, n – для выхода
у
Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 2
1 – Ориентированный, 2 – Не ориентированный граф2
Матрица смежности:
  1 2 3 4 5
1 0 0 1 0 1
2 0 0 0 0 1
3 1 0 0 1 1
4 0 0 1 0 0
5 1 1 1 0 0
Матрица успешно записана в graph.txt

Невозможно построить Эйлеров цикл или цепь!
Добавлено ребро между вершинами 5 и 4
Добавлено ребро между вершинами 3 и 2

Эйлеров цикл: 5 4 3 5 2 3 1 5

Введите у для продолжения, n – для выхода

```

Рисунок5–автоматическаягенерацияматрицысмежности, вывод цикла Эйлера/автокорректировка

В случае если автоматическая корректировка не смогла найти Эйлеров путь, то пользователю будет предложено самостоятельно ввести недостающие связи (рисунок 6).

```
Введите количество вершин графа: 5
1 - ручной ввод, 2 - случайная генерация: 2
1 - Ориентированный, 2 - Не ориентированный граф2
Матрица смежности:
  1  2  3  4  5
1  0  1  1  1  0
2  1  0  1  0  1
3  1  1  0  1  0
4  1  0  1  0  1
5  0  1  0  1  0
Матрица успешно записана в graph.txt

Невозможно построить Эйлеров цикл или цепь!
Добавлено ребро между вершинами 4 и 3
Добавлено ребро между вершинами 2 и 1

Все еще невозможно построить Эйлеров цикл, попробуйте добавить связь самостоятельно
Введите связь (формат: вершина1 вершина2) для добавления. Введите '-1 -1' для завершения:
2 4
Связь добавлена!
-1 -1

Эйлеров цикл: 1 4 5 2 4 3 2 1 3
Введите у для продолжения, n - для выхода
```

Рисунок6–автоматическаягенерацияматрицысмежности, вывод

цикла Эйлера/ ручная корректировка

В случае с ориентированным графом взаимодействие проходит похожим образом

Рассмотрим задачи основных функций программы, описывающих принцип работы (полный код программы – приложение А):

1. `filmoymatmam()` – данная функция необходима для ручного задания связей в графе(после ввода размерности графа в основной функции), ввод организован построчным указанием связей между вершинами (u,v). Ввод происходит до тех пор, пока пользователем не будут введены значения (-1, -1).
2. `filrandommam()`–такжефункциядлязаполненияграфа,нослучайными связями. Принцип работы схож с заполнением обычной матрицы массива `graph[i/j][j/i]`.
3. `printgraphmama()`–функциядлявыводаматрицысмежности.
4. `sohranigrafplz()`–функциядлясохранениявыведенныхданныхвфайл «graph.txt».Вслучае,еслипокакой-топричинеоткрытиефайла невозможно, выводится соответствующее сообщение об ошибке.

5. `poiskziclaeler()` – необходима, для определения Эйлера цикла. Алгоритм в данном случае перемещается по доступным ребрам, в то время как текущая вершина помещается в стек. В случае если на вершине не осталось ребер, она добавляется в формирующийся цикл и удаляется из стека.
6. `zikkaroehiliput()` – работает для определения существования цикла или пути Эйлера в графе. Если количество вершин с нечетными связями равно 0 или 2 ($\text{oddCount} == 0 \text{ И } \text{oddCount} == 2$), то такой граф имеет такой цикл.
7. `main()` – объединяет предыдущие функции, дает выбор нужного варианта работы программы и обеспечивает передачу данных.

5. Тестирование

Тестирование проводилось в среде разработки Microsoft Visual Studio 2022, в процессе было выявлено множество ошибок, которые в итоге были исправлены. Ниже представлен результат тестирования программы в различных вариациях работы программы: случайная генерация/ручной ввод связей в ориентированном/неориентированном графе, вывод матрицы смежности, корректировка графа для создания Эйлера цикла, вывод итогового результата (рис. 7, 8, 9, 10, 11, 12, 13).

```
Введите количество вершин графа: 5
1 - ручной ввод, 2 - случайная генерация: 2
1 - Ориентированный, 2 - Не ориентированный граф: 2
Матрица смежности:
  1  2  3  4  5
1  0  1  1  0  0
2  1  0  0  0  0
3  1  0  0  0  1
4  0  0  0  0  1
5  0  0  1  1  0
Матрица успешно записана в graph.txt
Эйлеров путь: 2 1 3 5 4 2
Введите у для продолжения, n - для выхода
|
```

Рисунок 7 – случайная генерация матрицы смежности в неориент. графе

```

Введите у для продолжения, п – для выхода
у
Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 1
1 – Ориентированный , 2 – Не ориентированный граф2
Введите связь (формат: вершина1 вершина2). Введите '-1 -1' для завершения:
1 2
2 3
3 4
4 1
-1 -1
Матрица смежности:
  1  2  3  4  5
1  0  1  0  1  0
2  1  0  1  0  0
3  0  1  0  1  0
4  1  0  1  0  0
5  0  0  0  0  0
Матрица успешно записана в graph.txt

Эйлеров цикл: 1 4 3 2 1

Введите у для продолжения, п – для выхода
|

```

Рисунок8–ручнойвводматрицысмежностивнеориент.графе

```

Введите у для продолжения, п – для выхода
у
Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 2
1 – Ориентированный , 2 – Не ориентированный граф1
Матрица смежности:
  1  2  3  4  5
1  0  1  0  0  0
2 -1  0  1  0 -1
3  0 -1  0  0  0
4  0  0  0  0  0
5  0  1  0  0  0
Матрица успешно записана в graph.txt

Граф не имеет цикл Эйлера

Введите у для продолжения, п – для выхода
|

```

Рисунок9–случайнаягенерацияматрицысмежностивориент.графе

```

Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 1
1 – Ориентированный , 2 – Не ориентированный граф1
Введите связь (формат: вершина1 вершина2). Введите '-1 -1' для завершения:
1 2
2 3
3 4
4 1
-1 -1
Матрица смежности:
  1  2  3  4  5
1  0  1  0 -1  0
2 -1  0  1  0  0
3  0 -1  0  1  0
4  1  0 -1  0  0
5  0  0  0  0  0
Матрица успешно записана в graph.txt

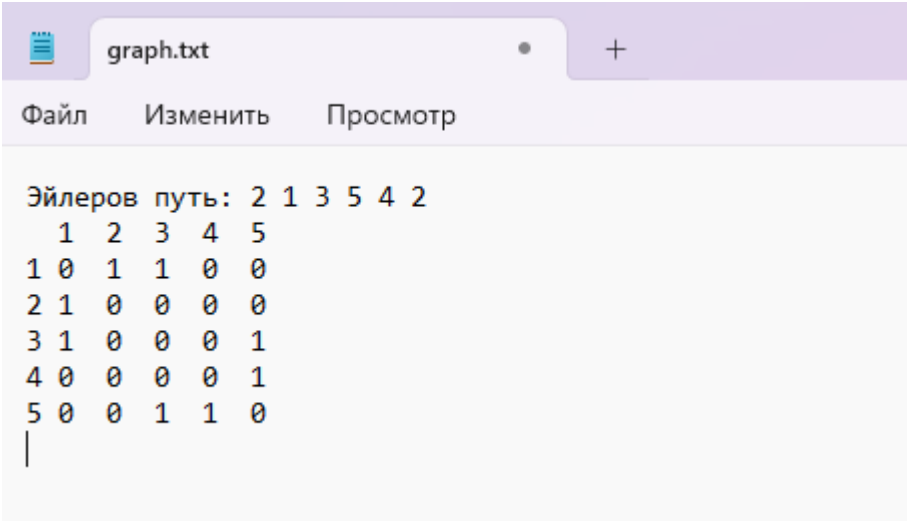
Граф имеет цикл Эйлера
Эйлеров цикл: 1 2 3 4 1

Введите у для продолжения, п – для выхода
|

```

Рисунок10–ручнойвводматрицысмежностивориент.Графе

Пример сохранения в файл на примере случайной генерации в неориентированном графе (см. рисунок 7) представлено ниже (рисунок 11).



The screenshot shows a text editor window with a single tab titled 'graph.txt'. The window has a menu bar with 'Файл', 'Изменить', and 'Просмотр'. The text content is as follows:

```
Эйлеров путь: 2 1 3 5 4 2
  1  2  3  4  5
1 0  1  1  0  0
2 1  0  0  0  0
3 1  0  0  0  1
4 0  0  0  0  1
5 0  0  1  1  0
|
```

Рисунок11–Сохранениевфайл

```

Введите у для продолжения, n – для выхода
у
Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 2
1 – Ориентированный, 2 – Не ориентированный граф2
Матрица смежности:
  1  2  3  4  5
1  0  1  0  1  1
2  1  0  1  0  0
3  0  1  0  0  0
4  1  0  0  0  0
5  1  0  0  0  0
Матрица успешно записана в graph.txt

Невозможно построить Эйлеров цикл или цепь!
Добавлено ребро между вершинами 5 и 4
Добавлено ребро между вершинами 3 и 1

Эйлеров цикл: 5 4 1 3 2 1 5

Введите у для продолжения, n – для выхода
^A\

```

Рисунок12–автокорректировкаграфадлясозданияЭйлеровацикла(пути)

```

у
Введите количество вершин графа: 5
1 – ручной ввод, 2 – случайная генерация: 2
1 – Ориентированный, 2 – Не ориентированный граф2
Матрица смежности:
  1  2  3  4  5
1  0  1  0  0  0
2  1  0  1  1  0
3  0  1  0  1  1
4  0  1  1  0  1
5  0  0  1  1  0
Матрица успешно записана в graph.txt

Невозможно построить Эйлеров цикл или цепь!
Добавлено ребро между вершинами 4 и 3
Добавлено ребро между вершинами 2 и 1

Все еще невозможно построить Эйлеров цикл, попробуйте добавить связь самостоятельно
Введите связь (формат: вершина1 вершина2) для добавления. Введите '-1 -1' для завершения:
4 1
Связь добавлена!
1 3
Связь добавлена!
-1 -1

Эйлеров путь: 1 4 5 3 4 2 3 1 2

Введите у для продолжения, n – для выхода

```

Рисунок13–ручнойвводоп.связейграфадлясозданияЭйлеровацикла(пути)

Результатыприведенныхвышереультатовтестированияприведены ниже (таблица 1).

Таблица 1–Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Открытие окна с полем для ввода количества вершин и дальнейшем взаимодействии с графом	Верно
Автоматическая генерация графа	После ввода в поле количества вершин графа, выборе варианта генерации/ориентированности в диалоговом окне должен появиться случайно сгенерированная матрица смежности	Верно
Генерация графа с использованием пользовательского ввода	При выборе соответствующей позиции в диалоговом окне должна вывестись информация о ручном вводе связей, в котором пользователь сможет заполнить матрицу смежности	Верно
Выбор ориентированности графа	Программа предложит выбор варианта графа (ориентированного или неориентированного) и сгенерирует его	Верно
Вывод Эйлера цикла (пути) при наличии	Если граф имеет эйлеров цикл (путь), то будет выведена последовательность цикла (пути).	Верно
Сохранение сгенерированного графа и вывода	Программа должна сохранить результаты работы программы	Верно

Продолжение таблицы 1

Описание теста	Ожидаемый результат	Полученный результат
Корректировка графа	В случае невозможности построения эйлера цикла, программа попытается автоматически сгенерировать недостающие связи, при невозможности будет выведен соответствующее сообщение и просьба о построении связей пользователем	Верно
Сохранение результатов в файл	Все результаты программы должны сохраняться в файл	Верно

В результате тестирования было выявлено, что программа успешно соответствует необходимым требованиям.

6. Ручной расчет задачи

Проведем проверку программы с помощью ручных вычислений на примере случайно сгенерированного графа с соответствующей матрицей смежности (рисунок 14)

```
у
Введите количество вершин графа: 5
1 - ручной ввод, 2 - случайная генерация: 2
1 - Ориентированный, 2 - Не ориентированный граф2
Матрица смежности:
  1 2 3 4 5
1 0 0 1 1 1
2 0 0 0 1 0
3 1 0 0 0 0
4 1 1 0 0 0
5 1 0 0 0 0
Матрица успешно записана в graph.txt

Невозможно построить Эйлеров цикл или цепь!
Добавлено ребро между вершинами 5 и 3
Добавлено ребро между вершинами 2 и 1

Эйлеров цикл: 5 3 1 4 2 1 5

Введите у для продолжения, n - для выхода
```

Рисунок14–случайнаягенерацияматрицы

Мы видим, что сгенерированная матрица смежности не имеет эйлера цикла (рисунок 15)

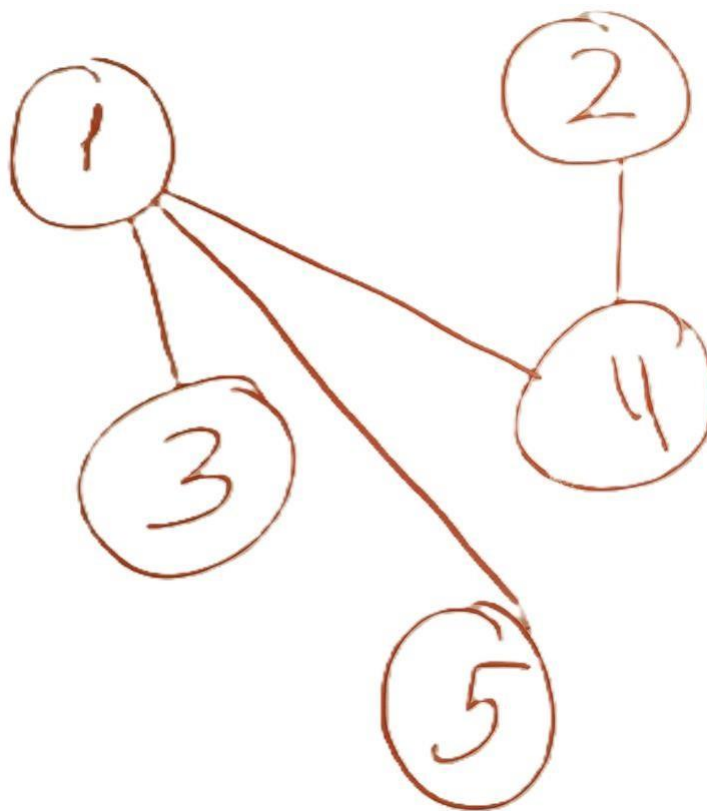


Рисунок15–построенныйграф

Так как 1, 3, 5 имеют нечетное количество связей, построить такой цикл или путь невозможно (начальная вершина не равна конечной, любая комбинация обхода всех связей предполагает прохождение хотя бы по одному ребру дважды: 2-4-1-5-1-3, 2-4-1-3-1-5 и т.д.). Для построения цикла программа построила связи между нечетными вершинами 3-5 и 1-2 (рисунок 16).

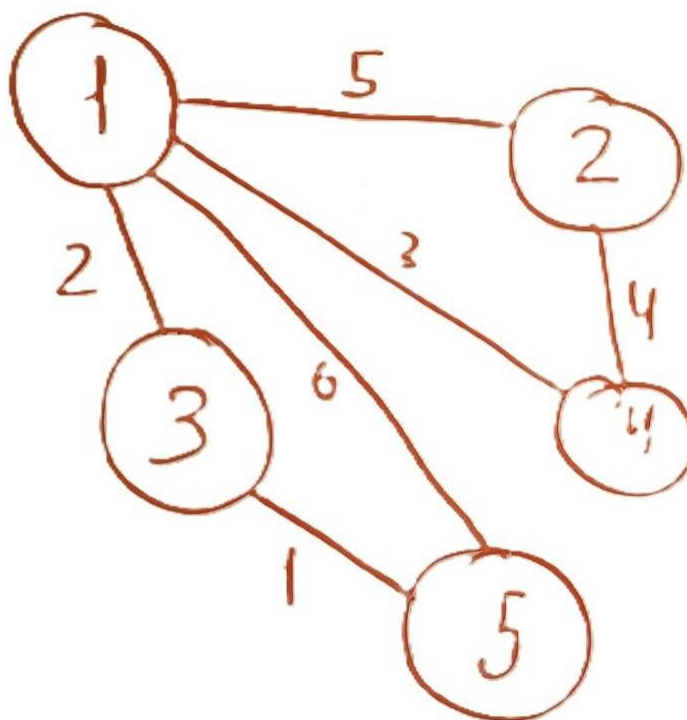


Рисунок 16 – отредактированный граф

После редактирования графа, можно сделать вывод о том, что он имеет цикл Эйлера: 5-3-1-4-2-1-5 (начальная вершина совпадает с конечной, все вершины имеют четную степень смежности).

В результате можно сделать вывод, что программа работает полностью исправно.

Заключение

При выполнении курсовой работы были получены навыки разработки многомодульных программ. Были освоены способы обработки событий при вводе с клавиатуры. Усвоены базовые навыки программирования на языке C/C++. Изучены основные возможности среды программирования VisualStudio 2022. Получены навыки тестирования и отладки программ.

В рамках выполнения курсовой работы была написана программа, необходимая для определения Эйлера цикла в графе. Программа предоставляет небольшой, но достаточный для использования список возможностей.

В дальнейшем программу можно усовершенствовать, путём добавления графического интерфейса, что позволит улучшить навигацию по меню программы, путём добавления возможности использования мыши.

Списокиспользуемыхисточников

1. ЯзыкC++ипрограммированиенанём:учебноепособие/В.И.Рейзлин;
Томский политехнический университет. – 3-е изд., перераб. – Томск : Изд-во
Томского политехнического университета, 2021. – 208 с.
2. КерниганБ.РитчиД.ЯзыкпрограммированияС.1985г.
3. Дольников, В. Л. Основные алгоритмы на графах : текст лекций / В. Л.
Дольников, О. П. Якимова; Яросл. гос. ун-т им. П. Г. Демидова. – Ярославль
:ЯрГУ, 2011. – 80 с. ISBN978-5-8397-0855-6

Приложение А Листинг

гпрограммы

Файлlalala.cpp

```
#include <iostream>
#include <vector>
#include <fstream>
#include <stack>
#include <cstdlib>
#include <ctime>
#include <windows.h>

using namespace std;

void filmoygrafmam(vector<vector<int>>& graph, int orient) {
    int u, v;
    cout<< "Введите связь (формат: вершина1 вершина2). Введите '-1 -1' для завершения:\n";
    while (true) {
        if (orient == 1) {
            cin>> u >> v;
            if (u == -1 && v == -1) break;
            if (u > 0 && (size_t)u<= graph.size() && v > 0 && (size_t)v <= graph.size()) {
                graph[u - 1][v - 1] = 1;
                graph[v - 1][u - 1] = -1;
            }
        } else {
            cout<< "Неверный ввод! Используйте вершины от 1 до " <<
                graph.size() <<endl;
        }
        if (orient == 2) {
            cin>> u >> v;
            if (u == -1 && v == -1) break;
            if (u > 0 && (size_t)u<= graph.size() && v > 0 && (size_t)v <= graph.size()) {
                graph[u - 1][v - 1] = graph[v - 1][u - 1] = 1;
            }
        } else {
            cout<< "Неверный ввод! Используйте вершины от 1 до " <<
                graph.size() <<endl;
        }
    }
}

void asm_funk() {
    for (inti = 0; i< 5; ++i)
    {
        string s;
        for (int j = 0; j < 50; ++j)
        {
            s.push_back(j <i % 50 ? '#' : ' ');
            printf("_|_|_|_|_|_|_|_|_\\n");
```

```

    }
s.push_back('\r');
Sleep(150);
}
printf("Добро пожаловать в программу для поиска Эйлера цикла (пути)!\nРуководство пользователя : просто
введите требуемую размерность графа в строку, вам будут выведены последовательные строки для изменения
характеристик графа :\n");
}

```

```

void filrandommam(vector<vector<int>>& graph, int orient) {
    for (size_t i = 0; i < graph.size(); ++i) {
        for (size_t j = i + 1; j < graph.size(); ++j) {
            graph[i][j] = graph[j][i] = rand() % 2;
        }
    }
    if (orient == 1) {
        for (size_t i = 0; i < graph.size(); ++i) {
            for (size_t j = i + 1; j < graph.size(); ++j) {
                if (graph[i][j] == 1)
                    if (rand() % 2 == 1) graph[i][j] *= -1;
                    else graph[j][i] *= -1;
            }
        }
    }
}

```

```

bool checkEulerian(const vector<vector<int>>& graph) {
    for (size_t i = 0; i < graph.size(); i++) {
        int inDegree = 0, outDegree = 0;
        for (size_t j = 0; j < graph.size(); j++) {
            if (graph[i][j] == -1) inDegree++;
            if (graph[i][j] == 1) outDegree++;
        }
        if (inDegree != outDegree) return false;
    }
    return true;
}

```

```

void findEulerianCycleUtil(vector<vector<int>>& graph, int v, stack<int>& path) {
    for (size_t u = 0; u < graph.size(); u++) {
        if (graph[v][u] == 1) {
            graph[v][u] = 0; // Удаляем ребро
            graph[u][v] = 0;
            findEulerianCycleUtil(graph, u, path);
        }
    }
    path.push(v);
}

```

```

void findEulerianCycle(vector<vector<int>>& graph) {
    stack<int> path;
    findEulerianCycleUtil(graph, 0, path); // Начинаем с вершины 0
    ofstream outFile;
    outFile.open("graph.txt", ios::out | ios::in);
    cout << "Эйлеров цикл: ";
    outFile << "Эйлеров цикл: ";
    while (!path.empty()) {
        cout << path.top() + 1 << " ";
        if (outFile.is_open()) {
            outFile << path.top() + 1 << " ";
        }
        path.pop();
    }
    cout << endl;
    outFile << std::endl;
}

```

```

void correctOrient(vector<vector<int>>& graph) {

```

```

int u, v;
cout<< "Введите связь (формат: вершина1 вершина2) для добавления. Введите '-1 1' для завершения:\n";

while (true) {
    cin>>u>>v;
    if (u == -1 && v == -1) break;

    if (u > 0 && (size_t)u<= graph.size() && v > 0 && (size_t)v <= graph.size()) {
        graph[u - 1][v - 1] = 1;
        graph[v - 1][u - 1] = -1;
        cout<< "Связь добавлена!\n";
    }
    else {
        cout<< "Неверный ввод! Используйте вершины от 1 до " <<graph.size() <<
        endl;
    }
}

void printgraphmama(const vector<vector<int>>& graph) {
    cout<< "Матрица смежности:\n ";
    for (size_t j = 0; j <graph.size(); j++) {
        cout<< j + 1 <<" ";
    }
    cout<<endl;

    for (size_t i = 0; i<graph.size(); i++) {
        cout<<i + 1 << " ";
        for (size_t j = 0; j <graph.size(); j++) {
            printf("%2i ", graph[i][j]);
        }
        cout<<endl;
    }
}

void sohranigrafplz(const vector<vector<int>>& graph) {
    ofstream outFile("graph.txt");
    if (outFile.is_open()) {
        outFile<< "Матрица смежности:\n ";
        for (size_t j = 0; j <graph.size(); j++) {
            outFile<< j + 1 <<" ";
        }
        outFile<<endl;

        for (size_t i = 0; i<graph.size(); i++) {
            outFile<<i + 1 << " ";
            for (size_t j = 0; j <graph.size(); j++) {
                outFile<< graph[i][j] <<" ";
            }
            outFile<<endl;
        }
        cout<< "Матрица успешно записана в graph.txt\n";
    }
    else {
        cout<< "Не удалось открыть файл для записи\n";
    }
}

bool ziklkarochiliput(const vector<vector<int>>& graph, int& start) {
    int oddCount = 0;
    for (size_t i = 0; i<graph.size(); i++) {
        int degree = 0;
        for (size_t j = 0; j <graph.size(); j++) {
            degree += graph[i][j];
        }
        if (degree % 2 != 0) {
            oddCount++;
            start = i; // Запоминаем стартовую вершину
        }
    }
}

```

```

    }
    return (oddCount == 0 || oddCount == 2);
}

void poiskziclaeler(vector<vector<int>>& graph, int start) {
    stack<int> st;
    vector<int> cycle;
    st.push(start);

    while (!st.empty()) {
        int v = st.top();
        bool hasEdge = false;

        for (size_t i = 0; i < graph.size(); i++) {
            if (graph[v][i]) {
                st.push(i);
                graph[v][i]--;
                graph[i][v]--;
                hasEdge = true;
                break;
            }
        }

        if (!hasEdge) {
            cycle.push_back(v);
            st.pop();
        }

        if (zikkaroehiliput(graph, start) && !cycle.empty() && cycle.front() ==
            cycle.back()) {
            cout<< "\nЭйлеровцикл: ";
        }
        else {
            cout<< "\nЭйлеровпуть: ";
        }

        fstream outFile;
        outFile.open("graph.txt", ios::out | ios::in);
        outFile<< "Эйлеровцикл: ";
        for (size_t i = 0; i < cycle.size(); i++) {
            int v = cycle[i];
            cout<< (v + 1) << " ";
            if (outFile.is_open()) {
                outFile<< (v + 1) << " ";
            }
        }
        outFile<<std::endl;
        cout<<endl;
    }

    void correct(vector<vector<int>>& graph) {
        int u, v;
        cout<< "Введите связь (формат: вершина1 вершина2) для добавления. Введите '-1 1' для завершения:\n";

        while (true) {
            cin>>u>>v;
            if (u == -1 && v == -1) break;

            if (u > 0 && (size_t)u <= graph.size() && v > 0 && (size_t)v <= graph.size()) {
                graph[u - 1][v - 1] = graph[v - 1][u - 1] = 1;
                cout<< "Связь добавлена!\n";
            }
            else {
                cout<< "Неверный ввод! Используйте вершины от 1 до " <<graph.size() <<
                    endl;
            }
        }
    }
}

```

```

void autocorrect(vector<vector<int>>& graph) {
    vector<int>oddVertices;
    for (size_t i = 0; i < graph.size(); i++) {
        int degree = 0;
        for (size_t j = 0; j < graph.size(); j++) {
            degree += graph[i][j];
        }
        if (degree % 2 != 0) {
            oddVertices.push_back((int)i);
        }
    }
    ofstream outFile;
    outFile.open("graph.txt", ios::out | ios::in);
    while (oddVertices.size() > 1) {
        int u = oddVertices.back(); oddVertices.pop_back();
        int v = oddVertices.back(); oddVertices.pop_back();

        // Добавляем ребро между u и v
        graph[u][v] = graph[v][u] = 1;
        cout<< "Добавлено ребро между вершинами " << (u + 1) << " и " << (v + 1) <<
        endl;
        if (outFile.is_open()) {
            outFile<< "Добавлено ребро между вершинами " << (u + 1) << " и " << (v
            + 1) << endl;
        }
        if (graph[u].size() % 2 == 0) oddVertices.push_back(u);
        if (graph[v].size() % 2 == 0) oddVertices.push_back(v);
    }
    cout<<endl;
}

int main() {
    setlocale(LC_ALL, "Russian");
    srand(static_cast<unsigned int>(time(0)));
    __asm {
        call asm_funk
    }
    while (true) {
        int size;
        cout<< "Введите количество вершин графа: ";
        cin>>size;

        while (size < 1) {
            cout<< "Некорректное значение! Введите количество вершин графа: ";
            cin>> size;
        }

        vector<vector<int>>graph(size, vector<int>(size, 0));
        int option, orient;

        cout<< "1 - ручной ввод, 2 - случайная генерация: ";
        cin>> option;

        cout<< "1 - Ориентированный, 2 - Не ориентированный граф: ";
        cin>> orient;

        if (option == 1) {
            filmoygrafmam(graph, orient);
        }
        else {
            filrandommam(graph, orient);
        }

        printgraphmama(graph);
        sohranigrafplz(graph);

        int start = 0;
        if (orient == 2) {

```

```

        if (ziklkarochiliput(graph, start)) {
    poiskziclaeler(graph, start);
    }
    else {
    cout<< "\nНевозможно построить Эйлеров цикл или цепь!\n";

    autocorrect(graph);
        if (ziklkarochiliput(graph, start)) {
    poiskziclaeler(graph, start);
    }
    else {
    cout<< "\nВсе еще невозможно построить Эйлеров цикл, попробуйте добавить связь самостоятельно\n";
    correct(graph);
        if (ziklkarochiliput(graph, start)) {
    poiskziclaeler(graph, start);
    }
    else {
    cout<< "\nВсе еще невозможно построить Эйлеров цикл!\n";
    }
    }
    }
    if (orient == 1) {
        if (checkEulerian(graph)) {
    cout<< "\nГраф имеет цикл Эйлера\n";
    findEulerianCycle(graph);
    }
    else
    cout<< "\nГраф не имеет цикл Эйлера\n";
    }

    char papa;
    cout<< "\nВведите у для продолжения, n - для выхода\n";
    cin>> papa;
    if (papa != 'y') {
        break;
    }
    else
        continue;
    }
    return 0

```