

Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и информатики

Высшая школа прикладной математики и вычислительной физики

## **Лабораторная работа №4 по дисциплине**

### **Дискретная математика**

Тема: «Циклы и раскраска»

Вариант 1 – Эйлеров цикл

Выполнил студент гр. 5030102/20202

Соколов А.Н.

Руководитель

Новиков Ф. А.

«\_\_\_» \_\_\_\_\_ 202\_\_ г.

Санкт-Петербург

2024

<b>1. Формулировка задачи и ее формализация</b>	<b>3</b>
<b>2. Используемые технологии</b>	<b>4</b>
<b>Исходные файлы программы:</b>	<b>4</b>
<b>3. Описание алгоритма</b>	<b>5</b>
<b>4. Практическая реализация</b>	<b>9</b>
<b>6. Формат входных и выходных данных</b>	<b>12</b>
<b>7. Вывод</b>	<b>13</b>

# 1. Формулировка задачи и ее формализация

## Формулировка задачи

1. Необходимо разработать консольное приложение, реализующее функции поиска в графе и вывода любого эйлеров цикла, если таковой имеется. В противном случае указать, что цикл не был найден.
2. Указать сложность алгоритма и доказать, что она именно такая.
3. Объяснить почему был выбран тот или иной способ представления графов в программе.

## 2. Используемые технологии

### Язык программирования

- C++ 23

### Система сборки

- CMake 3.27
- Ninja 1.12.1

### Исходные файлы программы:

<https://github.com/azya0/dm2025/tree/master/lab4>

### 3. Описание алгоритма

**Маршрутом** в графе называется чередующаяся последовательность вершин и рёбер, начинающаяся и кончающаяся вершиной, в которой любые два соседних элемента инцидентны, причем однородные элементы (вершины, ребра) через один смежны или совпадают.

Если **начальная** вершина маршрута является **конечной**, то **маршрут замкнут**, иначе — **открыт**.

**Если все ребра различны**, то маршрут называется **цепью**. Если все вершины (*а значит, и ребра!*) различны, то маршрут называется **простой цепью**.

**Замкнутая цепь** называется **циклом**. Замкнутая простая цепь называется **простым циклом**.

Если связный граф имеет цикл (*не обязательно простой*), содержащий **все ребра графа**, то такой цикл называется **эйлеровым** циклом, а граф называется **эйлеровым графом**.

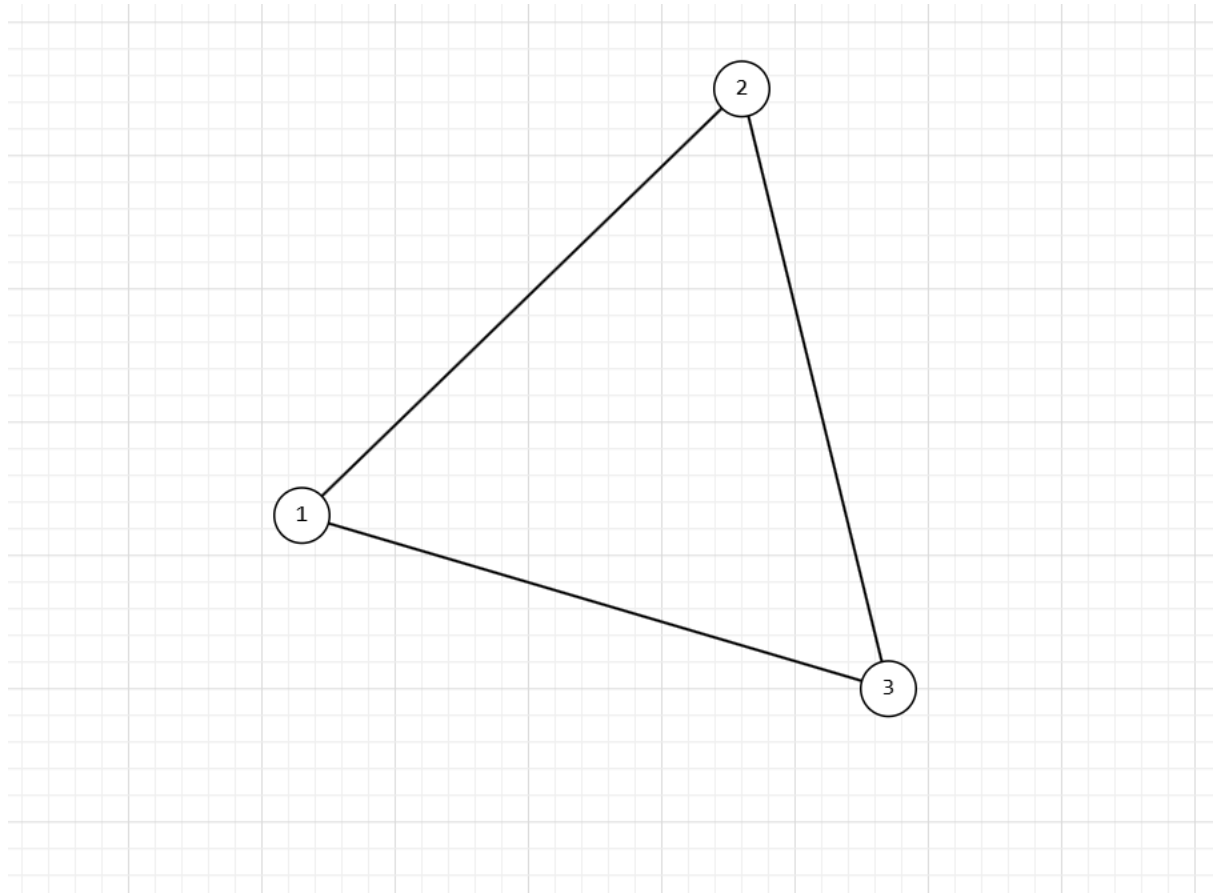
Важно обратить внимание, что граф **СВЯЗНЫЙ**.  
(*Это было сказано и написано в лекции Новикова Ф. А. - 1550 страница файла "DM2024.pdf"*)

Алгоритм будет следующим:

1. Проверим, все ли вершины имеют ненулевую, четную степень
2. Инициализируем стек с вершинами и добавим туда произвольную, а так же пустое множество
3. Пока стек не пуст, берем "верхнюю" вершину. Если у неё нет связей, удаляем её из стека и добавляем в итоговый цикл. Если связи есть, то берем первую смежную вершину, добавляем её в стек и множество и удаляем связь между ней и текущей
4. После необходимо проверить, является ли число элементов множества равным числу вершин в графе

### Пример:

Предположим, у нас есть взвешенный орграф:



Пусть первая произвольная вершина будет вершина 2.

1. Проверим степени всех вершин. Среди них все **четные и положительные**. Продолжаем.
2. **Выберем случайную вершину**, пусть это будет вершина “2”. Добавим вершину “2” в стек.
3. **Проверим стек**. Он не пуст. Продолжаем.
4. **Возьмем верхнюю вершину из стека**. Это вершина “2”.
5. **Проверим, есть ли у неё смежные вершины**. Да. Пусть первая произвольная смежная вершина - это вершина “3”. Добавим её в стек и удалим связи между неё и вершиной “2”.
6. **Проверим стек**. Он не пуст. Продолжаем.
7. **Возьмем верхнюю вершину из стека**. Это вершина “3”.

8. Проверим, есть ли у неё смежные вершины. Да.  
Единственная смежная вершина - это "1". Добавим её в стек и удалим связи между ней и вершиной "3".
9. Проверим стек. Он не пуст. Продолжаем.
10. Возьмем верхнюю вершину из стека. Это вершина "1".
11. Проверим, есть ли у неё смежные вершины. Да.  
Единственная смежная вершина "2". Добавим её в стек и удалим связи между ней и вершиной "1".
12. Проверим стек. Он не пуст. Продолжаем.
13. Возьмем верхнюю вершину из стека. Это вершина "2".
14. Проверим, есть ли у неё смежные вершины. Нет.  
Удаляем из стека, записывает в итоговый цикл и множество.
15. Возьмем верхнюю вершину из стека. Это вершина "1".
16. Проверим, есть ли у неё смежные вершины. Нет.  
Удаляем из стека, записывает в итоговый цикл и множество.
17. Проверим стек. Он не пуст. Продолжаем.
18. Возьмем верхнюю вершину из стека. Это вершина "3".
19. Проверим, есть ли у неё смежные вершины. Нет.  
Удаляем из стека, записывает в итоговый цикл и множество.
20. Проверим стек. Он не пуст. Продолжаем.
21. Возьмем верхнюю вершину из стека. Это вершина "2".
22. Проверим, есть ли у неё смежные вершины. Нет.  
Удаляем из стека, записывает в итоговый цикл и множество.
23. Проверим стек. Он пуст.
24. Проверим, столько же элементов в множестве, сколько вершин в графе? Да.
25. Вернем пользователю получившийся цикл.
26. Алгоритм завершен.

Таким образом, получившийся цикл:

2 -> 1 -> 3 -> 2

## Сложность алгоритма

$N$  — количество вершин.  $V$  — количество ребер.

### 1. Проверка

- а. Получение степени произвольной вершины занимает  $O(1)$
- б. Проверка степеней всех вершин будет требовать  $N$  итераций, поэтому сложность  $O(N * 1) \sim O(N)$

### 2. Алгоритм

- а. Стек не будет пуст, пока между вершинами есть связи, т.е.  $V$  итераций.
- б. При этом нужно потратить дополнительно  $N$  итераций, на проверку вершины, когда связей у неё не осталось. Добавление в вектор, как и добавление в множество имеет сложность  $O(1)$ , т.е.  $O(N)$ .
- с. При этом необходима ещё 1 операция, оценивающая (*уже на тот момент изолированную*) первую, добавленную в стек, вершину.

Таким образом, сложность алгоритма:  $O(2N + V + 1) \sim (N + V)$   
Линейная сложность, что является достойным результатом.



## 4. Практическая реализация

В программе граф представлен как хеш-таблица, в которой ключи - названия вершин, а значения - умные указатели на объекты класса *Node*:

```
class Node {
public:
    using Rib = std::pair<std::shared_ptr<Node>, int>;
    using RibContainer = std::unordered_map<std::shared_ptr<Node>,
int>;

    Node(std::string const & name);
    Node(std::shared_ptr<std::vector<Rib>> nodes, std::string const &
name);

    std::shared_ptr<RibContainer> Nodes();

    void addRib(std::shared_ptr<Node> rib, int weight);

    void rmRib(std::shared_ptr<Node> rib);

    std::string const & getName();

    int ribNumber() const;
private:
    std::shared_ptr<RibContainer> ribs;
    std::string name;
};
```

**Такое представление было выбрано**, т.к. с ним удобно работать и оно подразумевает использование всей выделенной под него памяти.

Так же, в отличие от прошлых реализаций, в этой объект класса *Node* хранит смежные вершины в хеш-таблице, для улучшения скорости алгоритма.

## **Получение программой исходного графа**

Программа предполагает, что вместе с исполняемым файлом пользователь будет хранить файл произвольного расширения, описывающий граф в формате:

```
A 2 B 1 D 1  
B 2 C 1 A 1  
C 2 D 1 B 1  
D 2 A 1 C 1
```

Где каждая строка - это имя вершины, количество ребер. Для каждого ребра - имя вершины, в которую ведет ребро, а также вес (цену) этого ребра (не влияет на алгоритм. Можно указать "0").

**После запуска исполняемого файла необходимо указать путь до файла.**

## 5. Область применения

### Файл с графом

- Неверный формат файла:
  - В файле содержится неправильный формат
- Количество смежных вершин у произвольной превышает размер типа данных **unsigned int**: больше, чем 4 294 967 294

Во всех остальных случаях программа будет работать корректно

## 6. Формат входных и выходных данных

На вход программа получает:

1. Название файла с графом в определенном формате:  
“Название вершины” количество ребер “Название вершины”  
“Вес ребра”...

Пример:

```
A 2 B 5 C 9
B 3 A 2 C 3 D 1
C 1 A 9
D 1 C 1
E 0
```

На выход программа выведет:

1. Эйлеров цикл, если такой существует.
2. Уведомление об отсутствии эйлеровского цикла, если такого не существует.

## **7. Вывод**

В рамках данной лабораторной работы был реализован алгоритм поиска Эйлера цикла.