Санкт-Петербургский политехнический университет Петра Великого

Кафедра прикладной математики и информатики

**Дискретная математика**

Отчет по лабораторной работе № 4

Тема "Циклы и раскраска"

**Вариант 4 – «Хроматическое число графа»**

Выполнил студент гр. 5030102/20202 Тишковец С. Е.

Санкт-Петербург

2024

**Оглавление**

[**Постановка задачи** 3](#_Toc185089585)

[**Язык программирования** 3](#_Toc185089586)

[**Описание алгоритма** 3](#_Toc185089587)

[**Демонстрация алгоритма** 5](#_Toc185089588)

[**Формат данных** 7](#_Toc185089589)

[**Дополнительные требования** 7](#_Toc185089590)

[**Область применимости алгоритма** 8](#_Toc185089591)

[**Источники информации** 8](#_Toc185089592)

# **Постановка задачи**

*Вариант 4 – Хроматическое число графа*

На вход программе подаётся граф. Требуется найти, используя приближённый алгоритм, и вывести хроматическое число данного графа. Всегда ли выбранный алгоритм будет давать точный ответ? Если да, то почему, а если нет, то предъявить контрпример.

Дополнительные требования:

* указать сложность алгоритма и доказать, что она именно такая;
* объяснить почему был выбран тот или иной способ представления графов в программе.

# **Язык программирования**

Алгоритм был реализован с помощью **C++ 14**

# **Описание алгоритма**

Алгоритм нахождения хроматического числа графа:

Хроматическое число графа — минимальное число цветов, в которые можно раскрасить вершины графа так, чтобы концы любого ребра имели разные цвета. Для нахождения хроматического числа графа используется усовершенствованный приближенный алгоритм:

1. На вход функции ***welshPowellAlgorithm*** подается граф в виде списка смежности и вектор ***colors***, который сохраняет цвет для каждой вершины. Изначально все элементы вектора ***colors*** равны **-1**, что означает отсутствие цвета у каждой вершины. В начале алгоритма вершины графа сортируются по убыванию степени (то есть числа рёбер, инцидентных данной вершине).

Стоит отметить, что сортируется именно вектор пар ***vector<pair<int, int>> degreeAndIndex,*** где первый элемент пары – это степень вершины, второй – номер вершины.Компаратор ***std::sort*** сортирует пары на основании первого значения пары. Если первое значение двух пар равно, они будут сравниваться на основе их второго значения, то есть основе номера вершины графа.

1. Инициализируется переменная ***currentColor***, которая будет содержать текущий цвет, который мы присваиваем вершинам. Начинаем с 0.
2. Запускается цикл, пробегающий по всем вершинам графа. Проверяется, окрашена ли вершина ***v***: если вершина не имеет цвета (значение равно -1), тогда мы окрашиваем её в ***currentColor***.
   1. Затем мы проходим по всем вершинам в отсортированном массиве, чтобы покрасить всех соседей. Если вершина ***u*** ещё не окрашена, мы проверяем, можно ли её покрасить в текущий цвет.
   2. Для каждого соседа вершины ***u***, мы проверяем: если сосед уже окрашен в ***currentColor***, то мы не можем раскрасить ***u*** в этот цвет.
   3. Если ни один из соседей вершины ***u*** не окрашен в ***currentColor,*** то мы раскрашиваем вершину ***u*** в этот цвет.
3. После того, как все вершины ***u*** проверены и, по возможности, окрашены в ***currentColor,*** цикл переходит к следующему цвету, то есть значение ***currentColor*** увеличивается на 1. Аналогичный алгоритм повторяется уже для нового цвета.
4. После того, как все вершины графа окажутся окрашены, алгоритм возвращает полученное хроматическое число графа.

# **Демонстрация алгоритма**

Изображение выглядит как круг

Автоматически созданное описаниеПример 1. Рассмотрим следующий двудольный граф

Записать граф можно в виде списка пар ребер между вершинами:

**(1, 6) (3, 5)**

**(1, 7)** **(3, 6)**

**(1, 8) (3, 8)**

**(2, 5) (4, 5)**

**(2, 7) (4, 6)**

**(2, 8)** **(4, 7)**

Шаг 1. Сортировка вершин графа.

Поскольку все вершины имеют одинаковую степень (3), то вершины будут отсортированы в обратном порядке, то есть **8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1**

Почему так происходит, объяснено в разделе ***«Описание алгоритма».***

Шаг 2. Раскраска графа.

Начиная с цвета **0** и с вершины **8** алгоритм присваивает допустимые цвета всем вершинам графа. Вершина **8** будет окрашена в цвет **0.** Вершина **7** не смежна с вершиной **8,** поэтому вершина **7** также будет окрашена в цвет **0**. Аналогично для вершин **6** и **5**.

Рассмотрим вершину **4.** Вершина **4** все еще не окрашена, однако среди ее соседей есть вершина **7**, которая окрашена в цвет **0.** Значит, окрасить вершину **4** в цвет **0** мы не можем – увеличиваем индекс цвета на единицу и окрашиваем вершину **4** уже цветом **1.** Для вершин **3, 2** и **1** аналогичная ситуация – например, вершина **3** смежна с вершиной **8,** которая окрашена цветом **0.** Значит, вершина **3** так же будет окрашена цветом **1**.

Хроматическое число – 2. Получим следующую раскраску графа:

**Вершина 1: Цвет 1**

**Вершина 2: Цвет 1**

**Вершина 3: Цвет 1**

**Вершина 4: Цвет 1**

**Вершина 5: Цвет 0**

**Вершина 6: Цвет 0**

**Вершина 7: Цвет 0**

**Вершина 8: Цвет 0**

Изображение выглядит как круг, графическая вставка

Автоматически созданное описаниеПример 2. Рассмотрим следующий двудольный граф

Записать граф можно в виде списка пар ребер между вершинами:

**(1, 4) (5, 2)**

**(1, 6)** **(5, 4)**

**(1, 8) (5, 8)**

**(3, 2) (7, 2)**

**(3, 6) (7, 4)**

**(3, 8)** **(7, 6)**

Данный граф отличается от предыдущего лишь нумерацией вершин, однако результат будет не точным. Покажем, почему данный граф является контрпримером.

Шаг 1. Сортировка вершин графа.

Поскольку все вершины имеют одинаковую степень (3), то вершины будут отсортированы в обратном порядке, то есть **8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1**

Почему так происходит, объяснено в разделе ***«Описание алгоритма».***

Шаг 2. Раскраска графа.

Начиная с цвета **0** и с вершины **8** алгоритм присваивает допустимые цвета всем вершинам графа**.** Вершина **8** будет окрашена в цвет **0.** Вершина **7** не смежна с вершиной **8,** поэтому вершина **7** также будет окрашена в цвет **0.**

**Переходим к следующей вершине – 6. Вершина 6 смежна с вершиной 7, которая окрашена в цвет 0.** Значит увеличиваем индекс цвета на единицу и окрашиваем вершину **6** уже цветом **1.** Аналогично для вершины **5** (она смежна с вершиной **8**)**.**

**Переходим к следующей вершине – 4. Она по-прежнему не окрашена. Вершина 4 смежна с вершиной 7, которая окрашена в цвет 0, и с вершиной 5, которая окрашена в цвет 1.** Значит увеличиваем индекс цвета на единицу и окрашиваем вершину **4** уже цветом **2.** Аналогично, вершина **3** окрашивается цветом **2**.

**Переходим к следующей вершине – 2. Она по-прежнему не окрашена. Вершина 2 смежна с вершиной 7, которая окрашена в цвет 0, с вершиной 5, которая окрашена в цвет 1, и с вершиной 3, которая окрашена в цвет 2.** Значит увеличиваем индекс цвета еще на единицу и окрашиваем вершину **4** уже цветом **3.** Аналогично, вершина **1** окрашивается цветом **3**.

Хроматическое число – 4. Получим следующую раскраску графа:

**Вершина 1: Цвет 3**

**Вершина 2: Цвет 3**

**Вершина 3: Цвет 2**

**Вершина 4: Цвет 2**

**Вершина 5: Цвет 1**

**Вершина 6: Цвет 1**

**Вершина 7: Цвет 0**

**Вершина 8: Цвет 0**

**Получили, что один и тот же (визуально) граф имеет хроматическое число сначала 2, потом 4. Все дело в нумерации вершин графа.**

# **Формат данных**

Входные данные: список смежности считывается из текстового файла. В первой строке указано количество вершин в графе, далее построчно, начиная со второй строки, в файле записаны ребра графа, соединяющие 2 вершины. Формат записи ребра в строке: **x\_y**, где **x** и **y** – смежные вершины в графе, \_ - означает пробел. Все символы текстового файла являются целыми числами.

Выходные данные: результатом выполнения программы является файл, в котором записано найденное хроматическое число графа, а также цвета каждой вершины графа в формате: **Вершина i: Цвет j, где i – номер вершины, j – номер цвета данной вершины.** Сообщение дублируются в консоль в таком же формате.

# **Дополнительные требования**

* **Временная сложность:** , где V – количество вершин, E – количество рёбер. Сортировка вершин с помощью функции ***std::sort*** занимает времени. В функции запускается 2 вложенных цикла ***for***, каждый из которых перебирает V вершин. Получаем . В ходе выполнения алгоритма проверяются соседи всех вершин, то есть перебираются рёбер. В итоге для всего алгоритма получим сложность .
* **Способ представления графа** в данной программе – список смежности – был выбран не спроста. Список смежности занимает  памяти, что может быть значительно меньше (особенно если граф слабо связный), чем , требуемое для матрицы смежности, или для матрицы инцидентности. Также, доступ к соседям в списке смежности требует времени (для графов без кратных рёбер) в отличие от для матрицы смежности.

# **Область применимости алгоритма**

1. Область применения:

Алгоритм предназначен для нахождения приближённого хроматического числа графа, что делает его полезным для задач распределения ресурсов или временных интервалов.

Например, алгоритм находит применение при построении маршрутов для доставки или общественного транспорта, когда нужно избегать конфликтов между различными транспортными средствами.

1. Случаи сбоя:

- Нарушение формата входных данных: например, если в файле не все элементы списка смежности (смежные вершины) будут заполнены, то алгоритм выдаст непредсказуемый результат.

- Алгоритм является жадным и не всегда дает корректный результат. Доказано, что если задача является матроидом, то жадный алгоритм выдаст правильный результат. Если же, например, нумерация вершин в двудольном графе будет такой, как показано во втором примере раздела «***Демонстрация алгоритма***», найдет не минимальное хроматическое число графа.

# **Источники информации**

* [https://disk.yandex.ru/i/g6JvWFBBmfvr0g](https://vk.com/away.php?to=https%3A%2F%2Fdisk.yandex.ru%2Fi%2Fg6JvWFBBmfvr0g&cc_key=)
* **https://www.geeksforgeeks.org/welsh-powell-graph-colouring-algorithm/**