Федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский Политехнический университет Петра Великого Физико-Механический институт

Лабораторная 4 – Циклы и раскраска

«Вариант 4 – Хроматическое число графа»

|  |  |  |
| --- | --- | --- |
| Выполнил студент гр. 5030102/20101: | Комаров Е.Р. |  |
| Преподаватель: | Новиков Ф. А. |  |
| Работа принята: | Дата |  |

Санкт-Петербург 2024

**Содержание**

1. [Введение](#_bookmark0) 2
   1. [Цели и задачи лабораторной работы:](#_bookmark1) . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
2. [Теоретическая часть](#_bookmark2) 3
   1. [Основные понятия графов:](#_bookmark3) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   2. [Описание алгоритма:](#_bookmark4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   3. [Сферы применения:](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
3. [Практическая часть](#_bookmark6) 4
   1. [Реализация графа](#_bookmark7) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
      1. [Список смежности](#_bookmark8) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   2. [Структура JSON файла](#_bookmark9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
4. [Сложности операций в классе Graph](#_bookmark10) 5
   1. [Инициализация графа ( init )](#_bookmark11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   2. [Добавление ребра (add\_edge)](#_bookmark12) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   3. [Добавление узла (add\_node)](#_bookmark13) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   4. [Удаление ребра (remove\_edge)](#_bookmark14) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   5. [Загрузка графа из JSON (load\_tree\_from\_json)](#_bookmark15) . . . . . . . . . . . . . . . . . . . . 5
   6. [Жадная раскраска графа (greedy\_coloring)](#_bookmark16) . . . . . . . . . . . . . . . . . . . . . . . 6
   7. [Основной метод раскраски графа (graph\_coloring)](#_bookmark17) . . . . . . . . . . . . . . . . . . . 6
      1. [Проверка возможность раскраски (\_is\_safe)](#_bookmark18) . . . . . . . . . . . . . . . . . . 6
      2. [Алгоритм раскраски графа (\_graph\_coloring)](#_bookmark19) . . . . . . . . . . . . . . . . . 7
5. [Примеры использования](#_bookmark20) 8
   1. [Неоптимальный результат жадной раскраски](#_bookmark21) . . . . . . . . . . . . . . . . . . . . . . 8
   2. [Работа алгоритма на полном графе](#_bookmark23) . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   3. [Работа алгоритма на изоморфном графе](#_bookmark24) 10
6. [Интерфейс приложения](#_bookmark25) 13
   1. [Пример запуска](#_bookmark26) 13
      1. [Задание рёбер графа](#_bookmark27) 13
      2. [Задание узлов и рёбер графа](#_bookmark28) 13
      3. [Загрузка графа из JSON файла](#_bookmark29) 13
7. [Заключение](#_bookmark30) 14
8. [Приложение](#_bookmark31) 15

# Введение

## Цели и задачи лабораторной работы:

В программе подаётся граф, и необходимо найти его хроматическое число с помощью приближённого алгоритма. Хроматическое число — это минимальное количество цветов, которое нужно для раскраски вершин графа так, чтобы смежные вершины не имели одинаковый цвет.

Цель лабораторной работы — изучить методы приближённого решения задачи раскраски графов и оценить их эффективность. Для этого будет использован жадный алгоритм для вычисления хроматического числа, а также будет проведён анализ его точности.

Ключевым вопросом является то, всегда ли выбранный алгоритм даёт точный результат. Если да, необходимо объяснить причины. Если нет, следует привести контрпример, который демонстрирует ситуации, в которых алгоритм может дать ошибочный ответ.

В рамках работы решаются следующие задачи:

Изучение ключевых понятий, связанных с графами и их раскраской.

Описание алгоритма, используемого для нахождения хроматического числа.

Реализация графа и его методов в программе.

Проведение экспериментов с различными типами графов для проверки работы алгоритма.

Анализ результатов и оценка точности приближённого алгоритма.

# Теоретическая часть

## Основные понятия графов:

Граф — это структура, состоящая из двух компонентов: множества вершин V и множества рёбер E, которые соединяют пары этих вершин. Одним из ключевых понятий в теории графов является хроматическое число графа, обозначаемое как χ(G). Оно представляет собой минимальное количество цветов, которые нужно использовать для раскраски вершин графа так, чтобы никакие две вершины, соединённые рёбром, не имели одинаковый цвет.

## Описание алгоритма:

Для нахождения хроматического числа графа можно применить приближённый алгоритм, основанный на жадном методе. Этапы работы алгоритма, следующие:

Начинаем с пустого набора цветов.

Для каждой вершины графа:

Определяем цвета, которые уже использованы соседними вершинами.

Присваиваем текущей вершине минимальный доступный цвет, который не используется её соседями.

По завершении обработки всех вершин количество использованных цветов будет приблизительным значением хроматического числа графа.

Этот алгоритм не всегда даёт точное значение хроматического числа, так как он не учитывает все возможные раскраски и может не выбрать оптимальную.

## Сферы применения:

Хроматическое число и методы его вычисления имеют множество практических применений в различных областях:

* **Теория графов**: Используется для анализа свойств графов и их структуры.
* **Компьютерные науки**: Применяется в задачах, связанных с распределением ресурсов, таких как раскраска графов, оптимизация сетей и планирование.
* **Операционные исследования**: Используется для решения оптимизационных задач, например, для распределения задач или маршрутизации.
* **Социальные сети**: Применяется для анализа взаимодействий между участниками, где вершины представляют пользователей, а рёбра — их связи.
* **Биология**: Используется для моделирования взаимодействий в биологических системах, таких как экосистемы или сети метаболизма.
* **Телекоммуникации**: Применяется для оптимизации частотного спектра и управления сетями связи.

# Практическая часть

## Реализация графа

Граф можно реализовать в виде класса, который будет содержать методы для работы с вершинами и рёбрами, а также для различных операций на графах.

### Список смежности

В реализации графа может быть использован список смежности для хранения связей между вершинами. В таком случае каждая вершина будет хранить список соседей, с которыми она соединена рёбрами.

## Структура Excel файла

Структура Excel для загрузки графа предполагает наличие следующих данных:

1. **Структура Excel**:
   * **Первый столбец**: Узел 1 (например, название вершины, например, "A").
   * **Второй столбец**: Узел 2 (связанный с первым узлом, например, "B").

В каждой строке содержатся два значения: пара рёбер графа, соединяющая узлы. Например:

| **Узел 1** | **Узел 2** |
| --- | --- |
| A | B |
| B | C |
| C | D |

1. **Алгоритм работы**:
   * Считывание данных из Excel начинается с второго ряда, где хранятся рёбра.
   * Узлы добавляются автоматически, а рёбра между ними создаются в процессе.

Таким образом, структура Excel должна содержать только два столбца, в которых указаны пары вершин, соединённых рёбрами.

# Сложности операций в классе Graph

* 1. **Инициализация графа (** init **)**
* Инициализация пустого графа с использованием defaultdict(set) — O(1).
* Если переданы узлы, то для каждого узла вызывается метод add\_node(node). Сложность этого шага — O(N), где N — количество узлов.
* Если переданы рёбра, то для каждого рёбер вызывается метод add\_edge(u, v). Сложность этого шага — O(M), где M — количество рёбер.
* Общая сложность: O(N + M).
  1. **Добавление ребра (**add\_edge**)**
* Операции добавления элемента в set — **O(1)** в среднем, так как это хеш-таблица.
* Метод add\_edge делает две операции добавления: одну для u и одну для v. Каждая операция занимает **O(1)** в среднем.
* Общая сложность: **O(1)**.
  1. **Добавление узла (**add\_node**)**

 Операция проверки наличия узла в графе с использованием in для хеш-таблицы defaultdict — **O(1)**.

 Операция добавления узла в defaultdict (если узел ещё не существует) — **O(1)**.

 Общая сложность: **O(1)**.

* 1. **Удаление ребра (**remove\_edge**)**

 Операция удаления элемента из множества set с использованием discard — **O(1)** в среднем.

 Метод remove\_edge делает две операции удаления: одну для u и одну для v. Каждая операция занимает **O(1)**.

 Общая сложность: **O(1)**.

* 1. **Загрузка графа из Excel (**load\_tree\_from\_excel**)**
* Открытие Excel файла и получение активного листа — **O(1)**.
* Чтение данных из листа с использованием iter\_rows — **O(N)**, где N — количество строк в листе (где каждая строка представляет одно ребро).
* Операция обновления множества узлов — **O(1)** для каждой строки, так как добавление в множество занимает **O(1)** в среднем.
* Операция добавления рёбер в список — **O(1)** для каждой строки.
* После сбора узлов и рёбер, добавление узлов и рёбер в граф с использованием методов add\_node и add\_edge — **O(N)** для узлов и **O(M)** для рёбер.
* Общая сложность: **O(N + M)**, где N — количество узлов, а M — количество рёбер.
  1. **Жадная раскраска графа (**greedy\_coloring**)**

 Для каждой вершины графа мы ищем все её соседей, у которых уже есть назначенный цвет.

 Операция поиска цветов соседей требует **O(deg(v))** времени для вершины с индексом v, где deg(v) — степень вершины.

 В худшем случае для каждой вершины придется выполнить цикл, проходя по всем её соседям.

 Если граф содержит N вершин и M рёбер, то общая сложность для жадной раскраски будет **O(N + M)**.

* 1. **Основной метод раскраски графа (**graph\_coloring**)**

 Метод пытается раскрасить граф с использованием от 1 до N цветов (где N — количество вершин).

 Для каждого количества цветов вызывается метод \_graph\_coloring, который будет проверять все возможные раскраски.

 В худшем случае метод выполняет N попыток, каждая из которых имеет сложность **O(N + M)**, так как метод \_graph\_coloring выполняется для каждого количества цветов.

 Общая сложность: **O(N^2 + NM)** в худшем случае.

### Проверка возможность раскраски (\_is\_safe)

 Для каждой вершины проверяется её соседей, чтобы убедиться, что назначенный цвет не конфликтует с соседними вершинами.

 В худшем случае сложность этой операции будет **O(deg(v))**, где deg(v) — степень вершины.

 Для одного вызова \_is\_safe сложность: **O(deg(v))**.

* + 1. **Алгоритм раскраски графа (**\_graph\_coloring**)**

 Алгоритм пытается раскрасить граф с использованием от 1 до max\_colors цветов.

 Для каждой вершины проверяется возможность присвоения каждого цвета (в худшем случае для всех цветов от 1 до max\_colors).

 Сложность поиска возможного цвета для каждой вершины — **O(deg(v))**, где deg(v) — степень вершины.

 В худшем случае алгоритм будет проверять все вершины, что даёт сложность **O(N × max\_colors × deg(v))**.

 Общая сложность: **O(N × max\_colors × deg(v))**.

# Примеры использования

## Граф с 4 вершинами

**Граф с рёбрами: 1-2, 1-3, 2-4, 3-4**

**Выполнение жадной раскраски**

* Узел 1 — цвет 0
* Узел 2 — цвет 1
* Узел 3 — цвет 2
* Узел 4 — цвет 3

**Выполнение оптимальной раскраски**

* Узел 1 — цвет 0
* Узел 2 — цвет 1
* Узел 3 — цвет 1
* Узел 4 — цвет 0

**Результаты:** Хроматическое число (жадная раскраска): 4 Хроматическое число (оптимальная раскраска): 2

## Граф с 5 вершинами

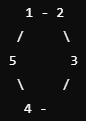
**Граф с рёбрами: 1-2, 2-3, 3-4, 4-5, 5-1 (цикл)**

Рис. 1: Цикл

**Выполнение жадной раскраски**

 Узел 1 — цвет 0

 Узел 2 — цвет 1 (так как сосед 1 имеет цвет 0)

 Узел 3 — цвет 0 (так как сосед 2 имеет цвет 1)

 Узел 4 — цвет 1 (так как сосед 3 имеет цвет 0)

 Узел 5 — цвет 0 (так как сосед 4 имеет цвет 1)

**Выполнение оптимальной раскраски**

* Узел 1 — цвет 0
* Узел 2 — цвет 1
* Узел 3 — цвет 0
* Узел 4 — цвет 1
* Узел 5 — цвет 0

**Результаты:** Хроматическое число (жадная раскраска): 2 Хроматическое число (оптимальная раскраска): 2

## Полный граф с 6 вершинами Граф с рёбрами: 1-2, 1-3, 1-4, 1-5, 1-6, 2-3, 2-4, 2-5, 2-6, 3-4, 3-5, 3-6, 4-5, 4-6, 5-6 (полный граф)

Рис. 2: Полный граф с 6 вершинами (жадная раскраска)

**Выполнение жадной раскраски**

* Узел 1 — цвет 0
* Узел 2 — цвет 1 (так как сосед 1 имеет цвет 0)
* Узел 3 — цвет 2 (так как соседи 1 и 2 имеют цвета 0 и 1)
* Узел 4 — цвет 3 (так как соседи 1, 2 и 3 имеют цвета 0, 1 и 2)
* Узел 5 — цвет 4 (так как соседи 1, 2, 3 и 4 имеют цвета 0, 1, 2 и 3)
* Узел 6 — цвет 5 (так как соседи 1, 2, 3, 4 и 5 имеют цвета 0, 1, 2, 3 и 4)

**Выполнение оптимальной раскраски**

Для полного графа с 6 вершинами минимальное количество цветов будет равно 6

**Результаты:** Хроматическое число (жадная раскраска): 6 Хроматическое число (оптимальная раскраска): 6

* 1. **Изоморфный граф**

Вершины 1, 3, 4, 6, 8 образуют один компонент

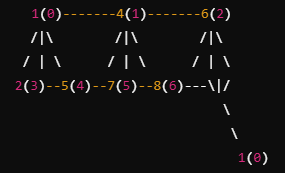
Вершины 2, 5, 7 соединены между собой и с вершинами 1, 4, 6, 8

Рис. 4: изоморфный граф

**Выполнение жадной раскраски**

* Вершина 1 — цвет 0
* Вершина 2 — цвет 1 (не может быть того же цвета, что и вершина 1)
* Вершина 3 — цвет 0 (не может быть того же цвета, что и вершина 2)
* Вершина 4 — цвет 1 (не может быть того же цвета, что и вершина 1)
* Вершина 5 — цвет 2 (не может быть того же цвета, что и вершины 2 и 4)
* Вершина 6 — цвет 2 (не может быть того же цвета, что и вершины 4 и 5)
* Вершина 7 — цвет 3 (не может быть того же цвета, что и вершины 5, 6)
* Вершина 8 — цвет 3 (не может быть того же цвета, что и вершина 7)

**Выполнение оптимальной раскраски**

* Вершина 1 — цвет 0
* Вершина 2 — цвет 1
* Вершина 3 — цвет 0
* Вершина 4 — цвет 1
* Вершина 5 — цвет 0
* Вершина 6 — цвет 1
* Вершина 7 — цвет 0
* Вершина 8 — цвет 1

**Результаты:** Хроматическое число (жадная раскраска): 4

Хроматическое число (оптимальная раскраска): 2

# Интерфейс приложения

## Пример запуска

Приложение можно запустить из командной строки с использованием различных аргументов. Все параметры указываются при запуске программы через командную строку.

### Задание рёбер графа

Если необходимо задать только рёбра графа, можно использовать параметр --edges. В этом случае узлы графа будут автоматически добавлены на основе указанных рёбер.

Пример команды:

python main.py --edges A B C D E F

### Задание узлов и рёбер графа

Если необходимо явно указать и узлы, и рёбра графа, можно использовать параметры --nodes и --edges одновременно. В этом случае граф будет строиться с указанными узлами и рёбрами.  
Пример команды:  
python main.py --nodes A B C D --edges A B C D

### Загрузка графа из Excel файла

Если граф уже сохранён в Excel файле, его можно загрузить с помощью параметра --excel-load. В файле должны быть указаны рёбра графа, где каждый ряд представляет пару соединённых узлов.

Пример команды:

python main.py --excel-load graph.xlsx

# Заключение

В ходе выполнения лабораторной работы были исследованы основные методы приближённого решения задачи раскраски графов, а также реализован жадный алгоритм для вычисления хроматического числа. Эксперименты показали, что данный алгоритм может давать приближённые результаты, которые не всегда точны.

Особенно стоит отметить, что жадный алгоритм может приводить к разным результатам для изоморфных графов. Это объясняется тем, что выбор цвета для каждой вершины зависит от порядка их обхода. Поэтому, несмотря на одинаковую структуру изоморфных графов, жадный алгоритм может назначить им разные хроматические числа, что подчеркивает его приближённый характер и чувствительность к порядку обработки вершин.

# Приложение

from collections import defaultdict, deque  
import argparse  
import openpyxl  
import sys  
  
  
class RedirectPrint:  
 def \_\_init\_\_(self, filename: str):  
 self.filename = filename  
 self.original\_stdout = sys.stdout  
  
 def \_\_enter\_\_(self):  
 self.file = open(self.filename, 'w')  
 sys.stdout = self.file  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):  
 sys.stdout = self.original\_stdout  
 self.file.close()  
 with open(self.filename, 'r') as f:  
 print(f.read())  
  
  
class Graph:  
 def \_\_init\_\_(self, nodes=None, edges=None) -> None:  
 self.graph = defaultdict(set)  
 if nodes is not None:  
 for node in nodes:  
 self.add\_node(node)  
 if edges is not None:  
 for u, v in edges:  
 self.add\_edge(u, v)  
  
 def add\_edge(self, u, v) -> None:  
 self.graph[u].add(v)  
 self.graph[v].add(u)  
  
 def add\_node(self, node) -> None:  
 if node not in self.graph:  
 self.graph[node] = set()  
  
 def remove\_edge(self, u, v) -> None:  
 self.graph[u].discard(v)  
 self.graph[v].discard(u)  
  
 def greedy\_coloring(self) -> dict:  
 color\_map = {}  
 for node in self.graph:  
 neighbor\_colors = {  
 color\_map[neighbor]  
 for neighbor in self.graph[node]  
 if neighbor in color\_map  
 }  
 color = 0  
 while color in neighbor\_colors:  
 color += 1  
 color\_map[node] = color  
 return color\_map  
  
 def \_is\_safe(self, node, color, color\_map):  
 for neighbor in self.graph[node]:  
 if neighbor in color\_map and color\_map[neighbor] == color:  
 return False  
 return True  
  
 def \_graph\_coloring(self, max\_colors):  
 color\_map = {}  
 nodes = list(self.graph.keys())  
 stack = deque([(0, 0)])  
 while stack:  
 node\_index, color\_processed = stack.pop()  
 if node\_index == 0 and color\_processed == 1:  
 return None  
 if node\_index == len(nodes):  
 return color\_map  
 node\_name = nodes[node\_index]  
 if nodes[node\_index] in color\_map:  
 del color\_map[node\_name]  
 for color in range(color\_processed, max\_colors):  
 if self.\_is\_safe(node\_name, color, color\_map):  
 color\_map[node\_name] = color  
 stack.append((node\_index, color + 1))  
 stack.append((node\_index + 1, 0))  
 break  
 return None  
  
 def graph\_coloring(self):  
 for num\_colors in range(1, len(self.graph) + 1):  
 color\_map = self.\_graph\_coloring(num\_colors)  
 if color\_map:  
 return color\_map  
 return None  
  
 def load\_tree\_from\_excel(self, filename: str):  
 wb = openpyxl.load\_workbook(filename)  
 sheet = wb.active  
 nodes = set()  
 edges = []  
 for row in sheet.iter\_rows(min\_row=2, values\_only=True):  
 u, v = row  
 nodes.update([u, v])  
 edges.append((u, v))  
 for node in nodes:  
 self.add\_node(node)  
 for u, v in edges:  
 self.add\_edge(u, v)  
  
  
def main():  
 parser = argparse.ArgumentParser(description="Хроматическое число графа")  
 graph\_group = parser.add\_argument\_group('Граф', 'Способы задания графа')  
 graph\_group.add\_argument(  
 '--edges',  
 nargs='+',  
 help='Список рёбер графа в формате: узел1 узел2, например: A B C D (для рёбер A-B и C-D)',  
 required=False)  
 graph\_group.add\_argument('--nodes', nargs='+', help='Список узлов графа, например: A B C D', required=False)  
 graph\_group.add\_argument(  
 '--excel-load',  
 help='Загрузить граф из Excel файла, например: graph.xlsx',  
 default='graph.xlsx',  
 required=False)  
 args = parser.parse\_args()  
 g = Graph()  
 with RedirectPrint('output.txt'):  
 if args.edges:  
 print("Загрузка графа из рёбер.")  
 for node in args.nodes:  
 g.add\_node(node)  
 for i in range(0, len(args.edges), 2):  
 g.add\_edge(args.edges[i], args.edges[i + 1])  
 print(  
 f"Граф загружен с {len(args.nodes)} узлами и {len(args.edges) // 2} рёбрами.")  
 elif args.excel\_load:  
 print(f"Загрузка графа из файла {args.excel\_load}.")  
 g.load\_tree\_from\_excel(args.excel\_load)  
 print("Граф успешно загружен из Excel.")  
 else:  
 raise ValueError(  
 "Неизвестный алгоритм. Укажите рёбра или файл Excel.")  
 print("Выполнение жадной раскраски.")  
 greedy\_coloring = g.greedy\_coloring()  
 print(  
 "Раскраска (жадная): ",  
 greedy\_coloring)  
 print("Выполнение оптимальной раскраски.")  
 best\_coloring = g.graph\_coloring()  
 print(  
 "Раскраска (оптимальная): ",  
 best\_coloring)  
 print(  
 f"Хроматическое число (жадная раскраска): {max(greedy\_coloring.values()) + 1}")  
 print(  
 f"Хроматическое число (оптимальная раскраска): {max(best\_coloring.values()) + 1}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Листинг 1: Код программы на Python