Санкт-Петербургский государственный
политехнический университет

Кафедра “Прикладная математика”

Отчёт по лабораторной работе 4 – Циклы и раскраска. Дисциплина “Дискретная математика”

Выполнл студент группы 5030102/20201 Сивошенко А.Ю

27 ноября 2024г.

Оглавление

Введение

В данной лабораторной работе требуется реализовать приближённый алгоритм для нахождения хроматического числа графа, используя жадную стратегию. Графы задаются в формате DOT, и результаты работы программы записываются в выходные файлы.

Язык программирования и его версия

Реализация выполнена на языке программирования C++ с использованием стандарта C++17.

Описание алгоритма

Алгоритм раскраски графа:

1. Сортировка вершин: Вершины графа упорядочиваются по невозрастанию степени.
2. Инициализация раскраски: Все вершины инициализируются неопределённым цветом (0).
3. Процесс раскраски:
   - Устанавливаем начальный цвет, начинаем красить вершины по очереди, избегая одинаковых цветов у соседних вершин.
   - Если вершину можно покрасить в текущий цвет, ей присваивается этот цвет, и она исключается из рассмотрения.
4. Переход к следующему цвету, если текущий исчерпан.

Псевдокод алгоритма:

```
функция ПриближённаяЖаднаяРаскраска(Граф):
    Сортировать вершины по невозрастанию степени

    Инициализировать Цвет текущим числом 1
```

```
Для каждой вершины v в отсортированном списке:
    Цвет[v] := 0 имеющие цвета

Пока существуют не покрашенные вершины:
    Для каждой вершины v из списка:
        можетБытьПокрашена := true
        Для каждого соседа u вершины v:
            Если Цвет[u] равен текущему Цвету:
                можетБытьПокрашена := false
                Выходим из внутреннего цикла

        Если можетБытьПокрашена:
            Цвет[v] := текущий Цвет
            Удалить v из списка вершин

    Увеличить текущий Цвет на 1
```

Сложность алгоритма и её обоснование

1. Сортировка вершин
   - Heap sort сортировка списка из (n) элементов имеет временную сложность (O(n log n)).
   - Здесь (n) – это количество вершин, обозначаемое как (V). Таким образом, сортировка вершин занимает (O(V log V)) времени.
2. Инициализация и раскраска
   - Каждую вершину мы обрабатываем, проверяя её соседей, что в сумме для всех вершин охватывает все рёбра. Каждое ребро (или дуга) просматривается один раз, что даёт временную сложность (O(E)), где (E) – количество рёбер.

С учётом этих двух основных частей, наибольшая временная сложность определяется как (O(V log V + E)).

Выбор представления графа

Графы представлены в виде списка смежности, это экономичныю по памяти для разреженных графов и удобно для итерации по соседям вершины.

Демонстрация работы алгоритма

Пример графа

Рассмотрим граф:

```
graph G {
    0 -- 1;
    0 -- 2;
    1 -- 3;
    2 -- 3;
    3 -- 4;
}
```

Шаги выполнения алгоритма:

1. Сортировка вершин

Сначала определим степени вершин, чтобы отсортировать их по невозрастанию степени:

- Вершина 0 имеет степень 2 (соседние вершины: 1, 2)
- Вершина 1 имеет степень 2 (соседние вершины: 0, 3)
- Вершина 2 имеет степень 2 (соседние вершины: 0, 3)
- Вершина 3 имеет степень 3 (соседние вершины: 1, 2, 4)
- Вершина 4 имеет степень 1 (соседняя вершина: 3)

Отсортируем вершины по невозрастанию степени:

1. Вершина 3 (степень 3)
2. Вершина 0 (степень 2)
3. Вершина 1 (степень 2)
4. Вершина 2 (степень 2)
5. Вершина 4 (степень 1)

    2. Инициализация раскраски

Каждой вершине изначально присваивается цвет 0, что означает, что они не раскрашены.

3. Присвоение цветов

Теперь алгоритм проходит по вершинам в отсортированном порядке и пытается покрасить каждую, следя за тем, чтобы соседние вершины не имели такого же цвета:

1. Вершина 3
   - Не имеет покрашенных соседей на момент раскраски, поэтому присваиваем ей цвет 1.
2. Вершина 0
   - Соседями являются вершины 1 и 2, которые ещё не окрашены. Присваиваем цвет 1.
3. Вершина 1
   - Сосед 0 имеет цвет 1, поэтому присваиваем вершине 1 цвет 2.
4. Вершина 2
   - Сосед 0 имеет цвет 1, поэтому присваиваем вершине 2 цвет 2.
5. Вершина 4
   - Сосед 3 имеет цвет 1, поэтому вершине 4 можно присвоить любой другой цвет, но в пределах используемых, например, цвет 2.

   Результат

После выполнения алгоритма, имеем следующую цветовую раскраску:

- Вершина 0: цвет 1
- Вершина 1: цвет 2
- Вершина 2: цвет 2
- Вершина 3: цвет 1
- Вершина 4: цвет 2

Полученное хроматическое число равно 2, так как мы использовали два цвета для раскраски всего графа, соблюдая правила, что никакие две смежные вершины не имеют одинаковый цвет.

Область применения и возможные ошибки

Алгоритм приближённой жадной раскраски работает корректно для графов с количеством вершин, меньшим, чем `INT_MAX`. Это следует из использования в программе соответствующих типов данных.

Алгоритм не гарантирует нахождение минимального хроматического числа.

Например, полный двудольный граф Kn,n с удалёнными рёбрами совершенного паросочетания будет окрашиваться в N цветов. Рассмотрим это в примере 4.

Формат входных и выходных данных

- Входные данные: Граф задаётся в формате DOT, имеющий следующий вид:

  Пример входного формата:

  ```
  graph G {
      0 -- 1;
      0 -- 2;
      1 -- 3;
      2 -- 4;
  }
  ```

- Выходные данные: Результаты раскраски графа в формате DOT, имеющий следующий вид:

  ```
  graph G {
      3 [style=filled, fillcolor=thistle];
      4 [style=filled, fillcolor=thistle];
      0 [style=filled, fillcolor=thistle];
      1 [style=filled, fillcolor=lightsteelblue];
      2 [style=filled, fillcolor=lightsteelblue];
      2 -- 4;
      1 -- 3;
      0 -- 1;
      0 -- 2;
  }
  ```

Примеры работы

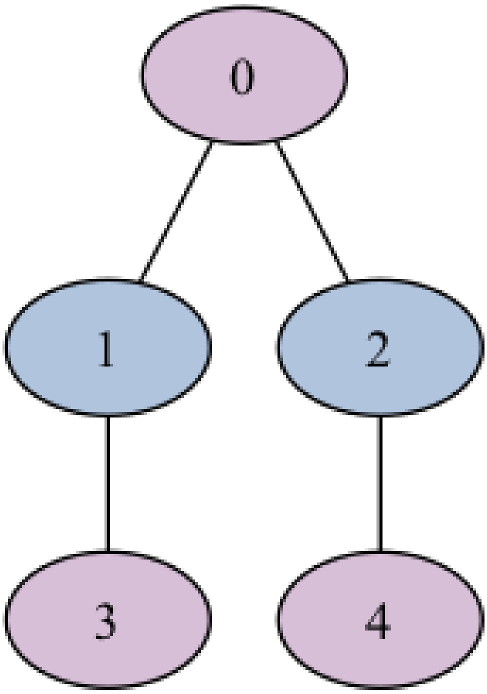
Пример 1:
Одно дерево с оптимальной раскраской.

Входной граф:

graph G { 0 – 1; 0 – 2; 1 – 3; 2 – 4; }

Вывод программы:

graph G { 3 [style=filled, fillcolor=thistle]; 4 [style=filled, fillcolor=thistle]; 0 [style=filled, fillcolor=thistle]; 1 [style=filled, fillcolor=lightsteelblue]; 2 [style=filled, fillcolor=lightsteelblue]; 2 – 4; 1 – 3; 0 – 1; 0 – 2; }



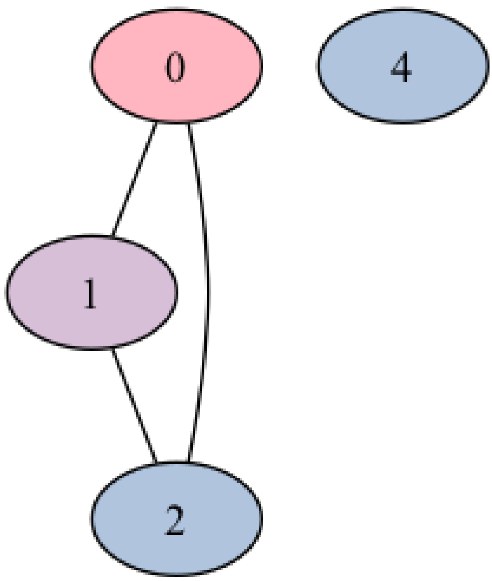
Пример 2:

Входной граф:

graph G { 0 – 1; 1 – 2; 2 – 0; 4 – 4; }

Вывод программы:

graph G { 0 [style=filled, fillcolor=lightpink]; 1 [style=filled, fillcolor=thistle]; 2 [style=filled, fillcolor=lightsteelblue]; 4 [style=filled, fillcolor=lightsteelblue]; 1 – 2; 0 – 1; 0 – 2; }



Пример 3:

Входной граф:

graph G {

1 – 6; 1 – 7; 1 – 8;

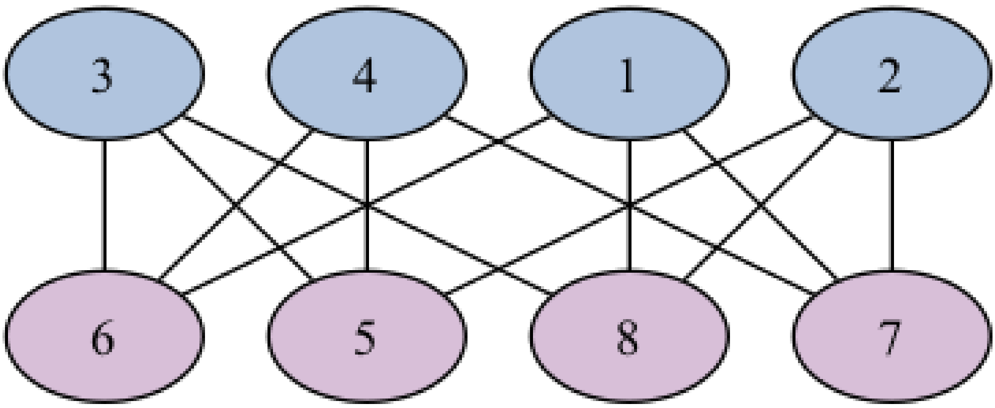
2 – 5; 2 – 7; 2 – 8;

3 – 5; 3 – 6; 3 – 8;

4 – 5; 4 – 6; 4 – 7; }

Вывод программы:

graph G { 1 [style=filled, fillcolor=lightsteelblue]; 6 [style=filled, fillcolor=thistle]; 7 [style=filled, fillcolor=thistle]; 2 [style=filled, fillcolor=lightsteelblue]; 5 [style=filled, fillcolor=thistle]; 8 [style=filled, fillcolor=thistle]; 3 [style=filled, fillcolor=lightsteelblue]; 4 [style=filled, fillcolor=lightsteelblue]; 4 – 5; 4 – 6; 4 – 7; 3 – 5; 3 – 6; 3 – 8; 2 – 5; 2 – 7; 2 – 8; 1 – 6; 1 – 7; 1 – 8; }



Раскраска оптимальна.

Пример 4:

Входной граф:

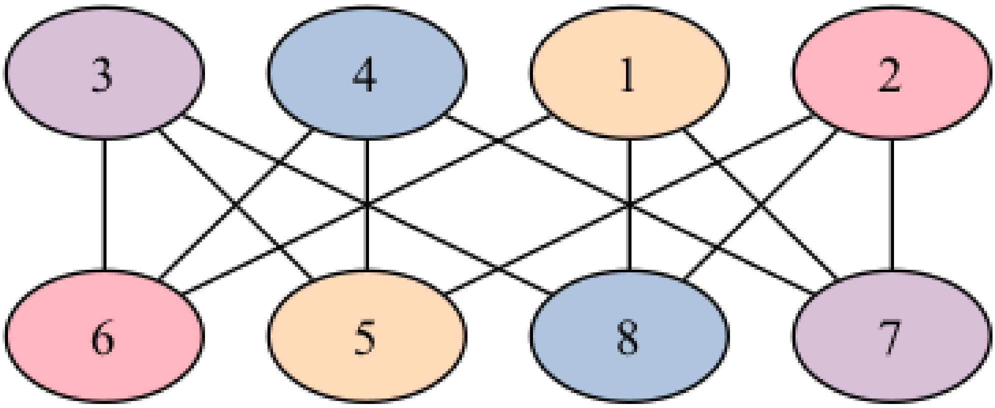
graph G { 1 – 6; 5 – 2; 3 – 8; 7 – 4;

1 – 7; 1 – 8; 2 – 7; 2 – 8;

3 – 6; 3 – 5; 4 – 5; 4 – 6; }

Вывод программы:

graph G { 1 [style=filled, fillcolor=peachpuff]; 6 [style=filled, fillcolor=lightpink]; 5 [style=filled, fillcolor=peachpuff]; 3 [style=filled, fillcolor=thistle]; 8 [style=filled, fillcolor=lightsteelblue]; 2 [style=filled, fillcolor=lightpink]; 7 [style=filled, fillcolor=thistle]; 4 [style=filled, fillcolor=lightsteelblue]; 4 – 7; 4 – 5; 4 – 6; 3 – 8; 3 – 6; 3 – 5; 2 – 5; 2 – 7; 2 – 8; 1 – 6; 1 – 7; 1 – 8; }



Пример неоптимальной работы алгоритма.

Заключение

В ходе работы был реализован алгоритм для нахождения хроматического числа графа с итоговой сложностью ($O(V \log V + E)$). Алгоритм оказался применим для быстрой оценки хроматического числа на практике.