Санкт-Петербургский государственный
политехнический университет

Кафедра “Прикладная математика”

Отчёт по лабораторной работе 3 – Деревья. Проверка свойства древочисленности. Дисциплина “Дискретная математика”

Выполнл студент группы 5030102/20201 Сивошенко А.Ю

27 ноября 2024г.

Оглавление

Введение

В данной лабораторной работе требуется реализовать алгоритм для проверки, является ли ориентированный граф ордеревом.

Язык программирования и его версия

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

Описание алгоритма

Алгоритм проверки ордерева включает в себя следующие шаги:

1. Идентификация корня: вершина с полустепенью захода 0.
2. Проверка полустепеней захода: для всех остальных вершин полустепень захода должна быть равна 1.
3. Проверка достижимости: каждая вершина должна быть достижима из корня.

Псевдокод алгоритма проверки на ордерево:

```
Функция isArborescence():
     Создать inDegree как пустую карту для отслеживания степеней захода
вершин
    Создать allVertices как множество для всех уникальных вершин

    Для каждой пары (u, список смежных вершин) в списке смежности:
        Добавить u в allVertices
        Для каждой вершины v в списке смежных вершин:
            Увеличить степень захода v на 1 в inDegree
            Добавить v в allVertices

    Создать roots как список для хранения возможных корневых вершин

    Для каждой вершины в allVertices:
```

```
        Если степень захода вершины равна 0:
            Добавить вершину в roots

    Если размер roots не равен 1:
        Вернуть ложь (обнаружено несколько корней) и список корней

    Отметить root как первую (и единственную) вершину в roots

    Для каждой вершины в allVertices, кроме root:
        Если степень захода вершины не равна 1:
            Вывести сообщение о некорректной полустепени захода

    Использовать поиск в ширину или глубину, чтобы проверить, достижимы
ли все вершины из root:
        С помощью очереди (или стека), запустить поиск из root
        Если обнаруживается недостижимость:
            Вывести сообщение о недостижимости

    Если все условия выполнены:
        Вернуть истину и сообщение о том, что граф является ордеревом
```

Псевдокод алгоритма проверки на древочисленность:

```
Функция isDrevochislenny():
    Установить переменную vertexCount равной количеству вершин в списке
смежности
    Установить переменную edgeCount равной 0

    Для каждой пары (u, список смежных вершин) в списке смежности:
        Увеличить edgeCount на количество рёбер из u (размер списка
смежных вершин)

    Если vertexCount равно edgeCount + 1:
        Вернуть истину и сообщение о том, что граф является древочисленным
    Иначе:
        Вернуть ложь и сообщение "Граф не является древочисленным (V !=
E + 1)"
```

Сложность алгоритма и её обоснование

Функция проверки на ордерево.

1. Инициализация и подсчёт полустепеней захода:

- Инициализация множеств и подсчёт степеней: Перебор всех рёбер и построение списка смежности и полустепеней захода требуют (O(V + E)).

2. Определение корня:
   - Поиск вершины с нулевой полустепенью захода: Требуется один проход по всем вершинам, дающий сложность (O(V)).
3. Проверка полустепеней захода:
   - Подтверждение корректности полустепеней захода: Еще один проход по всем вершинам требует (O(V)).
4. Проверка достижимости всех вершин из корня:
   - Обход в ширину (BFS): Посещение всех вершин и рёбер с использованием BFS занимает (O(V + E)).

Общая сложность для функции проверки на ордерево:

- Общая сложность: (O(V + E))

Функция проверки древочисленности

1. Подсчёт количества вершин:
   - Подсчёт из списка смежности: Проходим по всем вершинам в списке смежности, давая (O(V)).
2. Подсчёт количества рёбер:
   - Суммируем количество исходящих рёбер: Для каждой вершины, подсчитываем количество исходящих рёбер, что требует общего времени (O(E)).
3. Проверка условия древочисленности:
   - Сравнение количества вершин и рёбер: Простое сравнение, которое занимает (O(1)).

Общая сложность для функции проверки древочисленности:

- Общая сложность: (O(V + E))

Итоговая сложность программного кода:

При раздельной реализации этих функций, каждая из них работает с временной сложностью (O(V + E)).

Выбор представления графа

Графы представлены в виде списка смежности, так как это:

- Экономично по памяти для разреженных графов.
- Удобно для итерации по соседям вершины и подходит для обходов, как в BFS.

Демонстрация работы алгоритма

Давайте переделаем описание, чтобы оно было более подробным и ясным в контексте выполнения алгоритма:

Пример графа

Рассмотрим граф, заданный следующим образом:

```
digraph G {
    0 -> 1;
    0 -> 2;
    1 -> 3;
    2 -> 4;
}
```

Исходные условия

1. Корень графа: Вершина 0 идентифицируется как корень графа, поскольку её полустепень захода равна 0.
2. Инициализация полустепеней захода:
   - degree[0] = 0 (корень)
   - degree[1] = 1
   - degree[2] = 1
   - degree[3] = 1
   - degree[4] = 1

Шаги выполнения алгоритма

1. Определение корня:
   - Проверяется наличие единственной вершины с полустепенью захода 0. В графе это вершина 0, что выделяет её как единственный корень.
2. Проверка полустепеней захода каждой вершины:

- Убедимся, что все вершины, за исключением корня, имеют полустепень захода 1. В этом графе вершины 1, 2, 3 и 4 удовлетворяют этому условию.

3. Проверка достижимости всех вершин из корня:
   - Выполняем обход в ширину (BFS) начиная с корня (вершина 0):
     - От корня 0 достижимы вершины 1 и 2.
     - От вершины 1 достижима вершина 3.
     - От вершины 2 достижима вершина 4.
   - Все вершины графа (0, 1, 2, 3, 4) оказываются достижимыми из корня, это подтверждает связность графа.

Область применения и возможные ошибки

Использование int для подсчета вершин и рёбер может вызвать ошибки в случае их большого количества. Программа корректна для графов где количество дуг и количество вершин < INT_MAX.

Формат входных и выходных данных

- Входные данные: граф задаётся в формате списков смежности DOT.

  Пример входного формата:

  ```
  digraph G {
      0 -> 1;
      0 -> 2;
      1 -> 3;
      2 -> 4;
  }
  ```

- Выходные данные: в текстовом файле указывается, является ли граф ордеревом.

  Пример выходного файла:

  ```
  Граф является ордеревом.
  Граф является древочисленным.
  ```

Примеры работы

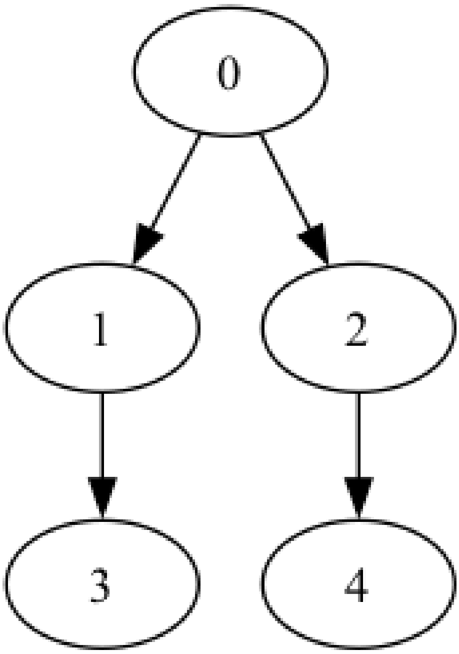
Пример работы алгоритма

Пример 1: Одно ордерево.

Входной граф: Файл:

```
digraph G {
  0 -> 1;
  0 -> 2;
  1 -> 3;
  2 -> 4;
}
```

Графическое представление:



Вывод программы:

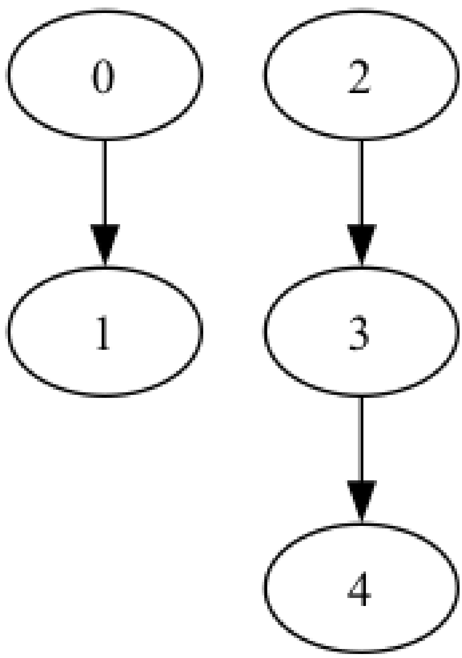
Граф является ордеревом. Граф является древочисленным.

Пример 2: Несколько корней.

Входной граф: Файл:

```
digraph G {
  0 -> 1;
  2 -> 3;
  3 -> 4;
}
```

Графическое представление:



Вывод программы:

Некорректный корень или несколько корней. Найдено корней: 2. Корни: 0 2

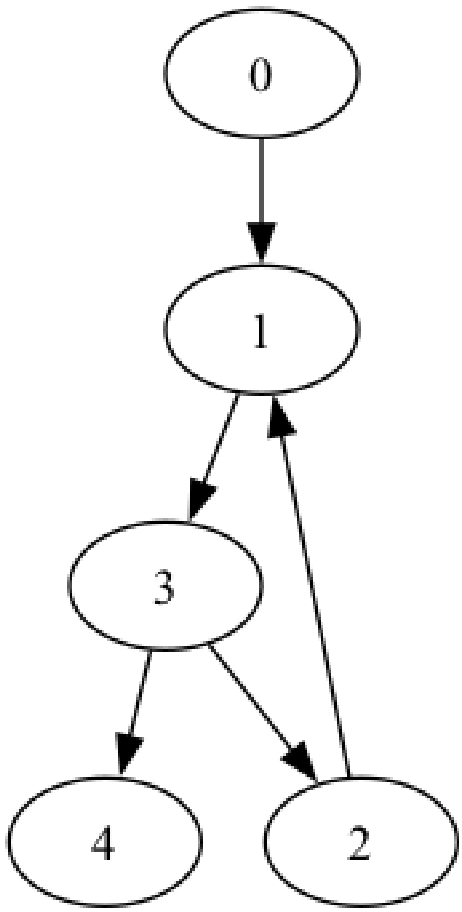
Граф не является древочисленным (V != E + 1).

Пример 3: Граф с циклом.

Входной граф: Файл:

```
digraph G {
  0 -> 1;
  1 -> 3;
  3 -> 4;
  3 -> 2;
  2 -> 1;
}
```

Графическое представление:



Вывод программы:

Вершина 1 имеет некорректную полустепень захода.

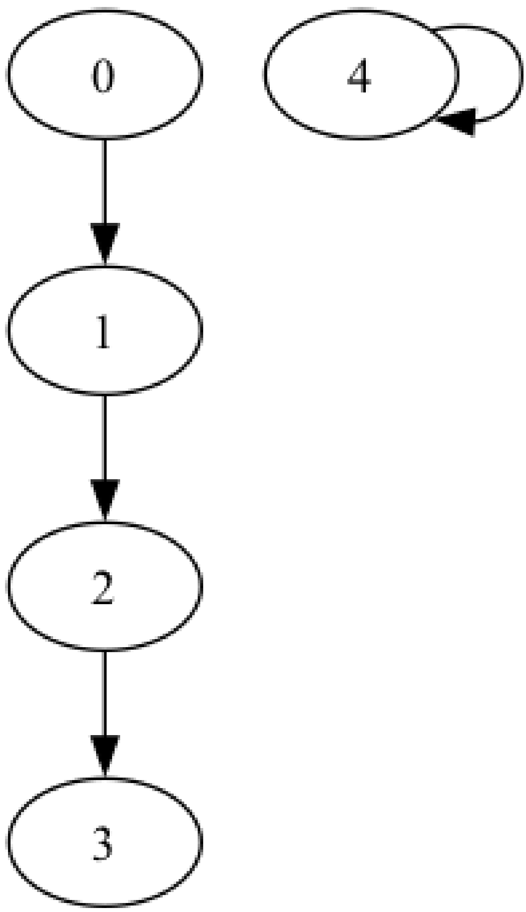
Граф не является древочисленным (V != E + 1).

Пример 4:

Входной граф: Файл:

```
digraph G {
  0 -> 1;
  1 -> 2;
  2 -> 3;
  4 -> 4;
}
```

Графическое представление:



Вывод программы:

Вершина 4 недостижима из корня.

Граф является древочисленным.

Отметим особенность, что в этом случае граф является древочисленным.

Заключение

В ходе работы был реализован алгоритм для проверки ордеревьев с итоговой сложность O(E+V).