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 полустепений захода: для всех остальных вершин полустепень входа должна быть равна 1.
3. Проверка достижимости: каждая вершина должна быть достижима из корня.

Псевдокод алгоритма:

```
функция являетсяОрдеревом(Граф):
    полустепеньЗахода := словарь, где ключ - вершина, значение - 0 для
всех вершин в Графе

    для каждой вершины u в Графе:
        для каждой соседней вершины v, доступной из u:
            полустепеньЗахода[v] := полустепеньЗахода[v] + 1

    корни := список

    для каждой вершины v в Графе:
        если полустепеньЗахода[v] == 0:
            добавить v в корни
```

```
если размер корни ≠ 1:
    вернуть "Некорректный корень или несколько корней"

корень := корни[0]

для каждой вершины v в Графе:
    если v ≠ корень и полустепеньЗахода[v] ≠ 1:
        вернуть "Некорректная полустепень захода для вершины " + v

достижимыеВершины := множество, содержащее корень
очередьДляОбхода := очередь, содержащая корень

пока очередьДляОбхода не пуста:
    текущаяВершина := извлечьПервый(очередьДляОбхода)

    для каждой соседней вершины сосед в Графе[текущаяВершина]:
        если сосед не принадлежит достижимыеВершины:
            добавить сосед в достижимыеВершины
            добавить сосед в очередьДляОбхода

если размер достижимыеВершины ≠ количество вершин в Графе:
    вернуть "Не все вершины достижимы из корня"

вернуть "Граф является ордеревом"
```

Сложность алгоритма и её обоснование

1. Инициализация и подсчёт полустепеней захода:
   - Инициализация словаря полустепеней захода: Пробегаем по всем вершинам графа, чтобы установить начальные значения, что занимает (O(V)) времени, где (V) — количество вершин.
   - Подсчёт полустепеней захода: Мы перебираем все дуги в графе, чтобы подсчитать количество входящих дуг для каждой вершины, что требует (O(E)) времени, где (E) — количество дуг (ребер) в графе.
2. Определение корня:
   - Для каждой вершины мы проверяем её полустепень захода, чтобы определить корень, что занимает (O(V)) времени.
3. Проверка полустепеней захода:

- Снова нужна одна итерация по всем вершинам для проверки их полустепеней захода, что занимает (O(V)) времени.

4. Проверка достижимости всех вершин из корня:
   - Эта часть использует обход в ширину (BFS). BFS проходит по всем вершинам и всем дугам в графе, что требует (O(V + E)) времени.

Итоговая сложность:

Каждая из перечисленных операций выполняется последовательно, и общая сложность алгоритма определяется самой “дорогостоящей” частью, что приводит к следующей суммарной временной сложности:

- Общая сложность: (O(V + E))

Алгоритм работает за линейное время от размера графа, учитывая как количество его вершин, так и количество дуг.

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

Для демонстрации возьмём граф с пятью вершинами:

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

Исходные условия

Корень графа определяется как вершина 0. Все полустепени инициализируются:

- (text{in-degree}[0] = 0)
- (text{in-degree}[1] = 1)
- (text{in-degree}[2] = 1)
- (text{in-degree}[3] = 1)
- (text{in-degree}[4] = 1)

Шаги алгоритма

1. Проверка наличия единственного корня.
2. Полустепени других вершин успешно проверены.
3. Все вершины достижимы из корня.

Алгоритм подтверждает, что граф — ордерево.

Область применения и возможные ошибки

Ордерева часто применяются в иерархических структурах данных. Возможные ошибки:

- Неверное определение корня при множественных перекрёстках дуг.
- Ошибки при проверке достижения всех узлов из корня.

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:

```
Graph is an arborescence
```

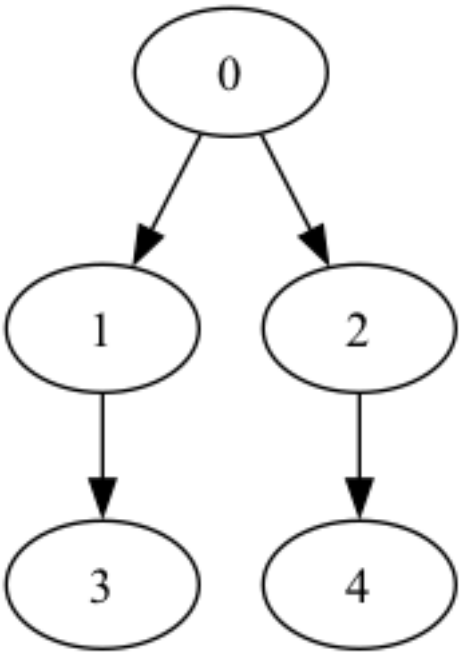
Примеры работы

Пример работы алгоритма

Пример 1: Одно дерево с корректным ордеревом.

Входной граф: Файл: `graph1.dot`

Графическое представление:



Вывод программы:

```
Graph analysis successful. Граф является ордеревом.
Result saved to output/output1.txt
```

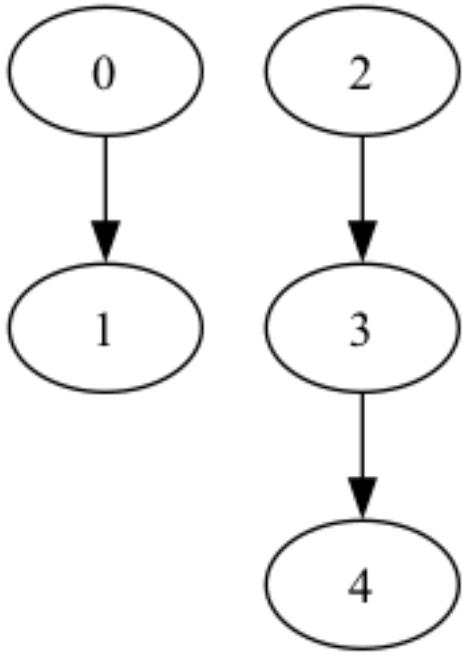
Описание работы алгоритма:

1. Инициализация: Определяется полустепень захода для всех вершин.
2. Определение корня: Вершина 0 является единственным корнем, так как у неё полустепень захода 0.
3. Проверка достижимости: Алгоритм успешно проверяет достижимость всех вершин из корня 0.
4. Результат: Все условия ордерева соблюдены, граф является ордеревом.

Пример 2: Несколько корней.

Входной граф: Файл: `graph2.dot`

Графическое представление:



Вывод программы:

```
Graph analysis failed. Некорректный корень или несколько корней.
Result saved to output/output2.txt
```

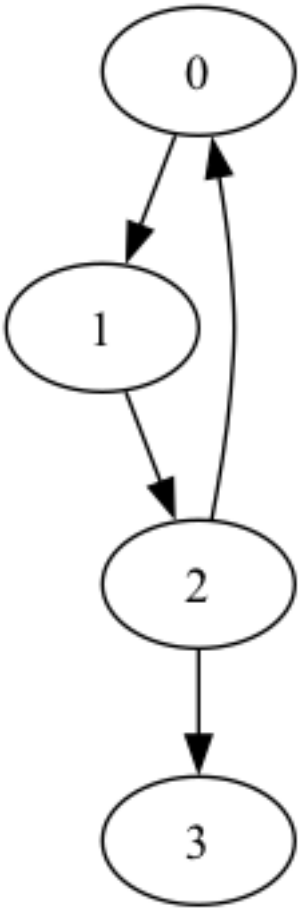
Описание работы алгоритма:

1. Инициализация: Полустепень захода подсчитывается для всех вершин.
2. Определение корня: Выявлено более одного корня (вершины 0 и 2), что нарушает условия ордерева.
3. Результат: Граф не может считаться ордеревом из-за множественных корней.

Пример 3: Граф с циклом.

Входной граф: Файл: `graph3.dot`

Графическое представление:



Вывод программы:

```
Graph analysis failed. Некорректный корень или несколько корней.
Result saved to output/output3.txt
```

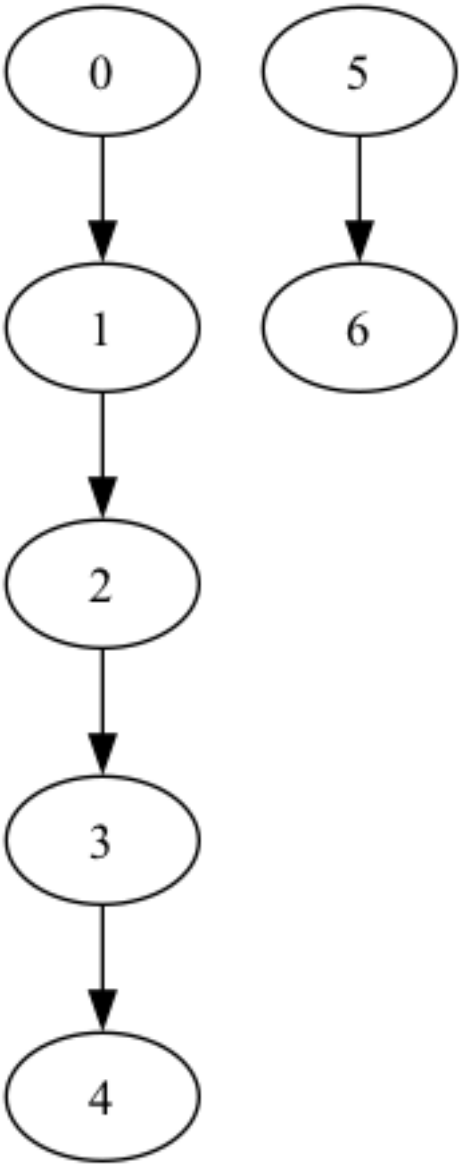
Описание работы алгоритма:

1. Инициализация: Подсчитываются полустепени захода.
2. Определение корня: Мультикорневое состояние наблюдается из-за цикла.
3. Обработка цикла: Наличие цикла (0 -> 1 -> 2 -> 0) не позволяет графу быть ордеревом.

Пример 4: Нерасширяемый граф.

Входной граф: Файл: `graph4.dot`

Графическое представление:



Вывод программы:

```
Graph analysis failed. Некорректный корень или несколько корней.
Result saved to output/output4.txt
```

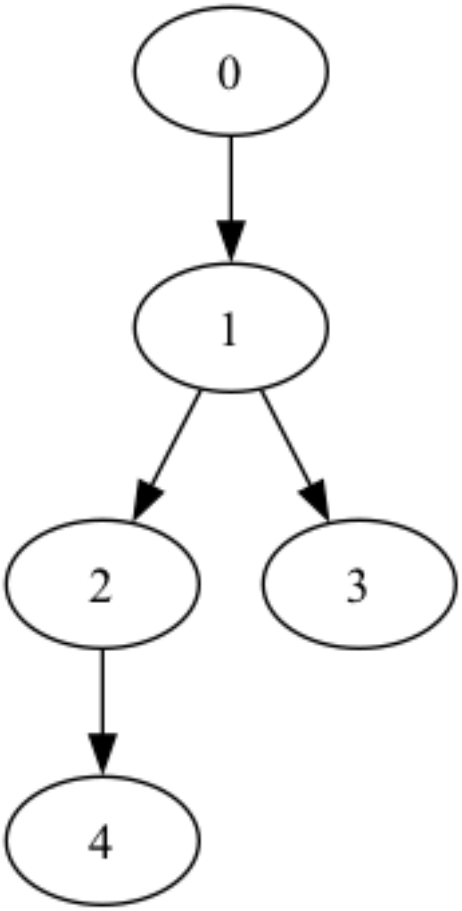
Описание работы алгоритма:

1. Инициализация: Определяется полустепень захода.
2. Достижимость: Обнаружено, что вершина 5 изолирована и недостижима из корня 0.
3. Результат: Граф не является ордеревом.

Пример 5: Свободное дерево.

Входной граф: Файл: `graph5.dot`

Графическое представление:



Вывод программы:

```
Graph analysis successful. Граф является ордеревом.
Result saved to output/output5.txt
```

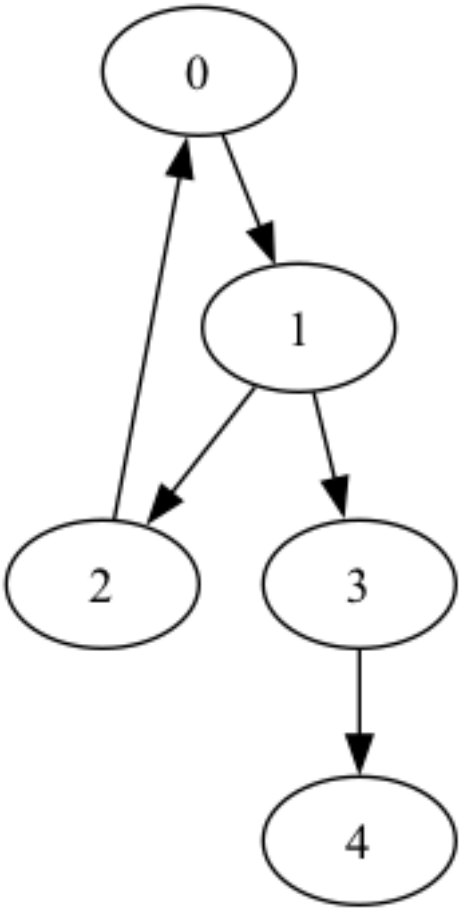
Описание работы алгоритма:

1. Инициализация: Подсчитываются полустепени захода для всех вершин.
2. Определение корня: Вершина 0 является единственным корнем.
3. Проверка достижимости: Все вершины доступны из корня.
4. Результат: Граф проходит все проверки, подтверждая свою структуру как ордерево.

Пример 6: Граф без единственного корня.

Входной граф: Файл: `graph6.dot`

Графическое представление:



Вывод программы:

```
Graph analysis failed. Некорректный корень или несколько корней.
Result saved to output/output6.txt
```

Описание работы алгоритма:

1. Инициализация: Вычисляется полустепень захода для всех узлов.
2. Обработка циклов: Найдены циклы, усложняющие структуру графа.
3. Результат: Граф не соответствует условиям ордерева из-за отсутствия единственного корня и присутствующих циклов.

Заключение

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