Санкт-Петербургский Политехнический Университет

Высшая школа прикладной математики и вычислительной физики, ФизМех

01.03.02 Прикладная математика и информатика

# Лабораторная работа №3

## Дисциплина “Дискретная математика”

## Тема “ Деревья ”

## Вариант “Проверка свойства древочисленности (субцикличность)”

Выполнил студент гр. 5030102/20201 Мелко Тимофей Андреевич

## Поставленная задача

Проверить является ли граф деревом, ацикличность, субцикличность, древочисленность.

## Используемый язык программирования

Python 3.12.6

## Описание проверки свойств

Проверка ацикличности:

Функция is_acyclic(graph):

  visited ← пустое множество

  Для каждой вершины v в графе:

    Если v не посещена:

      Если find_cycle_DFS(v, -1, graph, visited) = TRUE:

        Возвратить FALSE  # Граф содержит цикл

  Возвратить TRUE  # Граф ацикличен

Функция find_cycle_DFS(vertex, parent, graph, visited):

  Добавить vertex в visited

  Для каждой соседней вершины neighbor из graph[vertex]:

    Если neighbor не посещена:

      Если find_cycle_DFS(neighbor, vertex, graph, visited) = TRUE:

        Возвратить TRUE  # Цикл найден

    Иначе если neighbor ≠ parent:

      Возвратить TRUE  # Цикл найден

  Возвратить FALSE  # Циклов не найдено

Проверка субцикличности:
Функция is_subcyclic(graph):

  Если count_cycles(graph) ≠ 1:

    Возвратить (FALSE, NULL)  # Граф не субциклический

  Для каждой пары вершин (u, v), где (u, v) не являются соединенными в graph:

    Временно добавить ребро (u, v) в graph

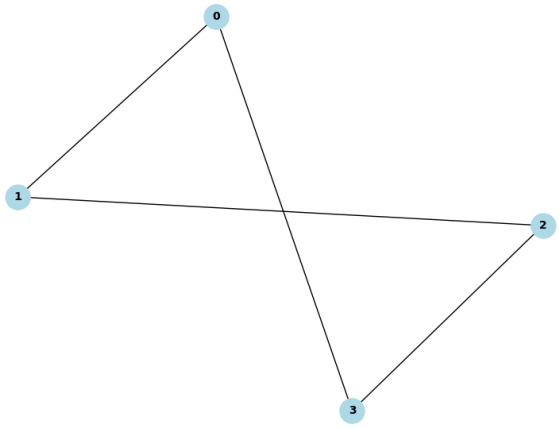
    Если count_cycles(graph) > 1:

```
            Удалить ребро (u, v) из graph
        Иначе:
            Удалить ребро (u, v) из graph
            Возвратить (FALSE, (u, v))  # Добавление ребра не увеличивает количество циклов
    Возвратить (TRUE, NULL)  # Граф является субциклическим

Функция count_cycles(graph):
    visited ← пустое множество
    cycle_count ← 0
    Для каждой вершины v в графе:
        Если v не посещена:
            Выполнить DFS из v
            Для каждого найденного цикла увеличить cycle_count на 1
    Возвратить cycle_count // 2  # Учитываем, что каждый цикл подсчитывается дважды

Проверка древочисленности:
Функция is_drevocislen(graph):
    p ← количество вершин в graph
    q ← 0

    Для каждой вершины v в graph:
        Для каждого соседа neighbor вершины v:
            Если ребро между v и neighbor существует:
                q ← q + 1

    q ← q // 2  # Каждое ребро подсчитано дважды

    Если q = p - 1:
        Возвратить TRUE  # Граф древочисленный
    Иначе:
        Возвратить FALSE  # Граф не древочисленный
```

## Пример работы

Рассмотрим граф для примера

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 1 | 0 | 1 | 0 |




Количество вершин p = 4, ребер q = 4

Древочисленность. q = p – 1. 4 != 4 – 1. Следовательно граф не является древочисленным.

Ацикличность z(G) = 0. С помощью алгоритма DFS обнаруживаем цикл 0 – 1 – 2 - 3 – 0, следовательно граф не является ацикличным.

Субцикличность z(G+x) = 1. Граф уже имеет цикл следовательно z(G+x) = 1 не выполняется и граф не субцикличный.

## Сложность

Ацикличность. Алгоритм проверки ацикличности основывается на поиске цикла через DFS. Алгоритм обхода графа с использованием DFS посещает каждую вершину и каждое ребро один раз. Для поиска цикла используется множество посещенных вершин и структура родителя для отслеживания предков. Это имеет сложность O(V+E).

Субцикличность. Алгоритм проверки ацикличности основывается на подсчете циклов через DFS. Основной цикл проходит по всем парам вершин, а для каждой пары вызывается подсчет циклов, дающий итоговую сложность O(V^2 (V + E)). В худшем случае $E \approx V^2$, тогда сложность составляет O(V^3).

## Входные и выходные данные

Входные данные. Квадратная матрица n×n, где n — количество вершин в графе. Каждая строка матрицы представляет связи (ребра) для одной вершины.

## Область применимости

Проверка графа на свойства и дальнейшая работа с ним особенно полезен в области сетей, алгоритмической оптимизации, системного проектирования и научных исследований.

## Представление графов в программе

Для представления графа в программе я буду использовать матрицу смежности. Доступ к данным в матрице занимает O(1) что делает возможным прямую и быструю работу с каждой парой вершин. Кроме того, добавление или удаление ребра также выполняется за O(1), так как достаточно изменить один элемент матрицы.

## Вывод

Данный код предоставляет универсальный инструментарий для анализа свойств графов, включая проверку их ацикличности, связности, субцикличности, древовидности и других характеристик. Он эффективно реализует алгоритмы на основе матрицы смежности, что делает его подходящим для решения задач в широком спектре областей, таких как теория графов, сетевой анализ, оптимизация маршрутов, проектирование систем и научные исследования.