

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)”

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)`:

1. Создаём пустое множество `visited`.
2. Создаём пустой словарь `parent`.
3. Для каждой вершины `v` от 0 до `V - 1`:
 - 3.1. Если вершина `v` не была посещена:
 - Создаём стек `stack` и кладём в него пару `(v, -1)`.
 - 3.2. Пока стек не пуст:
 - Извлекаем из стека пару `(current, prev)`.
 - Если `current` не в `visited`:
 - Добавляем `current` в `visited`.
 - Записываем `parent[current] = prev`.
 - Для каждого соседа `neighbor` вершины `current`:
 - Если `adj_matrix[current][neighbor] == 1` (есть ребро):
 - Если `neighbor` ещё не посещён:
 - Кладём `(neighbor, current)` в стек.
 - Иначе, если `neighbor` уже посещён и не равен `prev`:
 - Найден цикл. Возвращаем `False`.
 4. Если цикл не найден после обхода всех вершин, возвращаем `True`.

Проверка субцикличности:

Функция `is_subcyclic(graph)`:

`max_cycles` – общее количество циклов

1. Для каждой пары вершин `(u, v)`, где `u != v`:

1.1. Если между u и v нет ребра ($\text{adj_matrix}[u][v] == 0$):

- Временно добавляем ребро:

$\text{adj_matrix}[u][v] = 1$

$\text{adj_matrix}[v][u] = 1$

- Подсчитываем количество циклов в графе:

$\text{num_cycles} = \text{count_cycles}()$

$\text{max_cycles} = \max(\text{max_cycles}, \text{num_cycles})$

- Если $\text{num_cycles} > 1$:

- Удаляем добавленное ребро:

$\text{adj_matrix}[u][v] = 0$

$\text{adj_matrix}[v][u] = 0$

- Возвращаем False и пару (u, v) .

- Иначе:

- Удаляем добавленное ребро:

$\text{adj_matrix}[u][v] = 0$

$\text{adj_matrix}[v][u] = 0$

Проверка древочисленности:

Функция $\text{is_drevocislen}(\text{graph})$:

1. Инициализируем переменную $\text{edge_count} = 0$ для подсчёта количества рёбер.

2. Для каждой вершины u от 0 до $V - 1$:

2.1. Для каждой вершины v от $u + 1$ до $V - 1$:

- Если $\text{adj_matrix}[u][v] == 1$ (есть ребро между u и v):

- Увеличиваем edge_count на 1.

3. После завершения подсчёта рёбер:

- Если $\text{edge_count} == V - 1$:

- Возвращаем True (граф древочисленный).

- Иначе:

- Возвращаем False (граф не древочисленный).

Подсчет циклов

- Создаем пустое множество `all_cycles` для хранения уникальных циклов.
- Определяем `V` как размерность матрицы смежности (число вершин).

Для каждой вершины `start_vertex` в диапазоне от 0 до `V`:

- Инициализируем стек `stack` с элементом `(start_vertex, [start_vertex], [False] * V)`
(текущая вершина, путь до неё, массив посещённых вершин).

Пока стек не пуст:

- Извлекаем `current_vertex`, `path`, `visited` из стека.
- Помечаем `current_vertex` как посещённую: `visited[current_vertex] = True`.

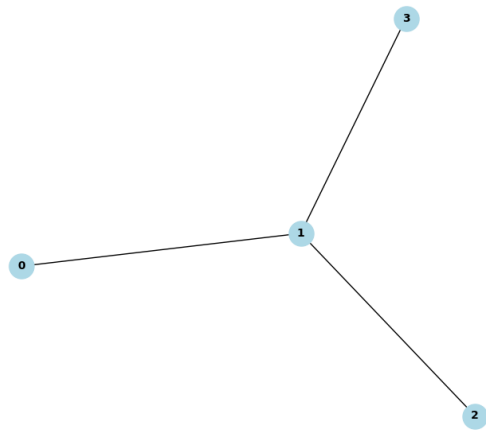
Для каждого соседа `neighbor` вершины `current_vertex`:

- Если `adj_matrix[current_vertex][neighbor] == 1` (есть ребро между вершинами):
 - Если `neighbor == start_vertex` и длина `path > 2`:
 - Найден цикл (возврат в стартовую вершину).
 - Сортируем путь `path` и добавляем его в `all_cycles`.
 - Иначе, если `neighbor` ещё не посещён:
 - Кладём `(neighbor, path + [neighbor], copy(visited))` в стек.
- Сбрасываем посещённость `current_vertex`: `visited[current_vertex] = False`.
- Возвращаем размер `all_cycles` как количество уникальных циклов.

Пример работы

Рассмотрим графы для примеров проверки свойств

	0	1	2	3
0	0	1	0	0
1	1	0	1	1
2	0	1	0	0
3	0	1	0	0



Граф является ациклическим.

Старт из вершины 0:

Стек: $[(0, -1)]$.

Из 0 идём к 1 (ребро 0-1).

Из 1 идём к 2 (ребро 1-2), затем возвращаемся.

Из 1 идём к 3 (ребро 1-3), затем возвращаемся.

Нет повторного посещения уже пройденной вершины, нет цикла.

Старт из вершин 1, 2, 3 также не выявляет циклов.

Вывод: Граф ациклический.

Граф является древочисленным ($q = p - 1$).

Количество вершин $p=4$.

Сумма всех элементов матрицы = $1+3+1+1=6$. Делим на 2: $q=3$.

$p-1=3$.

Граф является субциклическим.

Пары несвязанных вершин: (0-2), (0-3), (2-3).

Для пары (0-2):

Добавляем ребро 0-2.

Проверяем наличие циклов: один цикл (0-1-2-0).

Количество циклов = 1, граф остаётся субциклическим.

Для пары (0-3):

Добавляем ребро 0-3.

Проверяем наличие циклов: один цикл (0-1-3-0).

Количество циклов = 1, граф остаётся субциклическим.

Для пары (2-3):

Добавляем ребро 2-3.

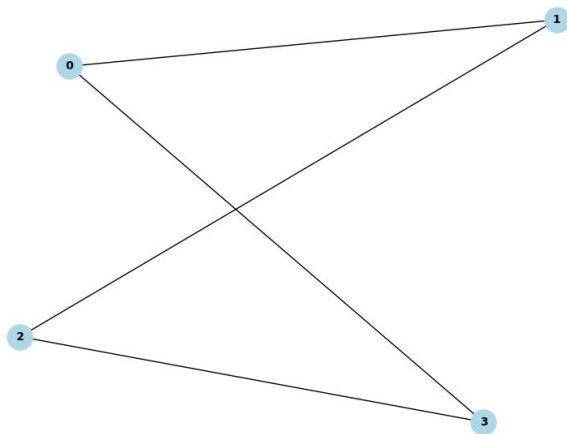
Проверяем наличие циклов: один цикл (1-2-3-1).

Количество циклов = 1, граф остаётся субциклическим.

Вывод: Граф является субциклическим.

Граф является деревом.

	0	1	2	3
0	0	1	0	1
1	1	0	1	0
2	0	1	0	1
3	1	0	1	0



Граф содержит цикл: [1, 0, 3, 2, 1]

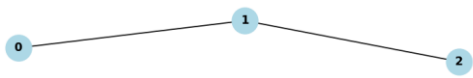
Граф не является древочисленным ($q \neq p - 1$).

Субциклическость нарушена при добавлении ребра (0, 2).

Граф не является деревом.

	0	1	2	3
0	0	1	0	0
1	1	0	1	0
2	0	1	0	0
3	0	0	0	0

3



Граф является ацикличным.

Граф не является древочисленным ($q \neq p - 1$).

Граф является субциклическим.

Граф является деревом.

	0	1	2
0	0	1	0
1	1	0	0
2	0	0	0



2

Граф является ациклическим.

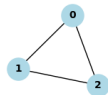
Граф не является древочисленным ($q \neq p - 1$).

Субцикличность нарушена при добавлении ребра Любого

Граф не является деревом.

	0	1	2	3
0	0	1	1	0
1	1	0	1	0
2	1	1	0	0
3	0	0	0	0

3



Граф содержит цикл: [1, 0, 2, 1]

Граф является древочисленным ($q = p - 1$).

Граф является субциклическим.

Граф не является деревом.

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 $O(V^2 (V + E))$.

Входные и выходные данные

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

Выходные данные записываем в файл в виде

if граф ацикличный:

```
f.write("Граф является ацикличным.")
```

else:

```
f.write (f"Граф содержит цикл: {найденный цикл}")
```

if граф древочисленный:

```
f.write ("Граф является древочисленным (q = p - 1).")
```

else:

```
f.write ("Граф не является древочисленным (q != p - 1).")
```

if граф субцикличный:

```
f.write ("Граф является субциклическим.")
```

else:

```
f.write (f"Субцикличность нарушена при добавлении ребра {найденное ребро}.")
```

if граф дерево ():

```
f.write ("Граф является деревом.")
```

else:

```
f.write ("Граф не является деревом.")
```

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.