# САНКТ – ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3 По курсу «Алгоритмы и структуры данных» Тема: Графы Вариант 13

> Выполнила: Мкртчян А.А. K3242

Проверил: Афанасьев А.В.

Санкт-Петербург 2024 г.

# Задача N3

#### Описание задачи

**Условие**: Направленный граф задан в виде списка рёбер. Нужно определить, существует ли цикл в этом графе.

## Исходный код

```
def has cycle(n, edges):
    for u, v in edges:
        graph[u - 1].append(v - 1)
             if not visited[neighbor]:
    if dfs(neighbor):
         recursion stack[v] = False
result = has cycle(n, edges)
```

```
with open('output.txt', 'w') as f: f.write(str(result) + '\n')
Входные данные (input.txt):
```

5 7

12

23

13

3 4

14

25

35

Выходные данные (output.txt):

#### 0

#### Пояснение:

- Граф содержит 5 вершин и 7 рёбер.
- Программа проверяет наличие циклов в графе с помощью поиска в глубину (DFS).
- В данном графе нет циклов, что подтверждается результатом 0.

#### Алгоритм

- 1. Построение графа: Граф строится как список смежности, где каждая вершина хранит список всех своих соседей.
- 2. Поиск в глубину (DFS):
  - 。 Используются два массива:

- visited[]: хранит информацию о том, была ли вершина посещена.
- recursion\_stack[]: отслеживает вершины, находящиеся в текущем рекурсивном вызове, чтобы обнаружить цикл.
- Если в процессе DFS обнаруживается вершина, которая уже находится в рекурсивном стеке, это означает, что найден цикл.
- 3. **Запуск DFS для каждой вершины**: Если вершина не посещена, запускается DFS. Если цикл обнаружен, программа возвращает 1. В противном случае возвращается 0.

#### Вывод

Данный алгоритм эффективно решает задачу поиска цикла в ориентированном графе с помощью поиска в глубину, возвращая результат в виде 0 при отсутствии цикла или 1 при его наличии.

# Задача N12

#### Условие:

У нас есть набор комнат, соединённых коридорами, каждый из которых имеет свой цвет. Путь задан последовательностью цветов, и нужно определить, в какой комнате окажется человек,

следуя этому пути, начиная из комнаты 1. Если путь не может быть пройден, программа должна вернуть "INCORRECT".

## Алгоритм

- 1. **Граф:** Построим граф с использованием списка смежности, где каждая комната хранит список других комнат, доступных через коридор определённого цвета.
- 2. **Путь:** Итерируем по цветам в пути и проверяем, существует ли коридор этого цвета из текущей комнаты. Если коридора нет, возвращаем "INCORRECT". Если есть, переходим в соответствующую комнату.
- 3. **Выходные данные:** Если путь завершён корректно, возвращаем номер комнаты. Если путь не может быть пройден, выводим "INCORRECT".

## Исходный код:

```
def check_path(n, corridors, path):
# Проверка: путь должен содержать хотя бы две комнаты
if len(path) < 1: # Путь может быть пустым
    return 1 # Если путь пустой, остается в комнате 1

# Строим граф как список смежности
graph = [{} for _ in range(n)]
for u, v, color in corridors:
    if color not in graph[u - 1]:
        graph[u - 1][color] = []
        if color not in graph[v - 1]:
            graph[v - 1][color] = []
        graph[u - 1][color].append(v - 1)
        graph[v - 1][color].append(u - 1)

current_room = 0 # Havunaem c комнаты 1 (индекс 0)

# Проверяем путь
for color in path:
    if color in graph[current_room]: # Если есть коридор нужного

цвета

# Переходим в первую комнату по этому цвету (можно улучшить
для выбора конкретной комнаты)
        current_room = graph[current_room][color][0] # Переход в
первую подходящую комнату
else:
        return "INCORRECT"

return current_room + 1 # Возвращаем номер комнаты (индекс + 1)

# Чтение данных
with open('input.txt', 'r') as f:
        n, m = map(int, f.readline().split())
        corridors = [tuple(map(int, f.readline().split())) for in range(m)]
```

```
k = int(f.readline().strip())
    path = list(map(int, f.readline().split())) if k > 0 else []

# Проверка пути
result = check_path(n, corridors, path)

# Запись в файл
with open('output.txt', 'w') as f:
    if result == "INCORRECT":
        f.write("INCORRECT\n")
    else:
        f.write(str(result) + '\n')
```

#### Пояснение

- 1. **Структура графа:** Каждый элемент списка graph это словарь, где ключ цвет коридора, а значение список комнат, куда можно попасть через этот коридор.
- 2. **Алгоритм поиска пути:** Для каждого цвета из пути проверяем, существует ли переход из текущей комнаты по коридору этого цвета. Если коридор найден, мы переходим в первую доступную комнату, иначе возвращаем "INCORRECT".
- 3. Граничный случай: Если путь пустой, человек остаётся в комнате 1.

## Пример

```
Входные данные (input.txt):
3 2
1 2 10
1 3 5
5
10 10 10 10 5
Выходные данные (output.txt):
3
Объяснение:
```

- Из комнаты 1 можно пойти в комнату 2 через коридор цвета 10.
- Следуя цветам коридоров 10 четыре раза, человек остаётся в комнате 2.
- Последний цвет 5 ведёт из комнаты 2 в комнату 3.

# Задача N13

Описание: Дано поле с грядками и пустыми участками, представленные символами:

- # грядка
- . пустая земля

Требуется посчитать количество отдельных грядок. Грядка — это непрерывная область клеток с символом #, которая может распространяться по четырём направлениям: вверх, вниз, влево и вправо.

#### Алгоритм:

- 1. **Обход поля:** Используем алгоритм поиска в глубину (DFS) для нахождения всех клеток, связанных с одной грядкой. Начинаем DFS, если встречаем клетку с грядкой, которую еще не посещали.
- 2. **DFS:** После нахождения новой грядки (клетки с #), с помощью DFS посещаем все связанные клетки этой грядки, помечая их как посещенные.
- 3. Счетчик: Для каждой новой найденной грядки увеличиваем счетчик.

### Реализация:

```
with open('input.txt', 'r') as f:
    n, m = map(int, f.readline().split())
    grid = [list(f.readline().strip()) for _ in range(n)]

visited = [[False] * m for _ in range(n)]

beds_count = 0

# Обход по участку
for i in range(n):
    if grid[i][j] == '#' and not visited[i][j]: # Если нашли грядку
        visited[i][j] = True
        dfs(grid, visited, i, j) # Запускаем DFS
        beds_count += 1 # Увеличиваем счётчик грядок

# Запись результата
with open('output.txt', 'w') as f:
    f.write(str(beds_count) + '\n')
```

#### Пояснение:

- 1. **Стек:** Используем стек для хранения координат клеток грядок, которые нужно посетить. Это позволяет использовать DFS итеративно.
- 2. **Обход:** Начинаем обход поля с каждой клетки. Если клетка является частью грядки и еще не была посещена, запускаем DFS, который пометит все клетки этой грядки как посещенные.
- 3. Счетчик грядок: Для каждой новой грядки увеличиваем счетчик.

### Пример:

Входные данные (input.txt):

```
5 10

##..####.

.#.#.#...

###..##.#.

..##...#

..##.####

Выходные данные (output.txt):
```

# Вывод по проделанной работе

В рамках задачи была реализована программа для подсчета количества грядок на участке, представленном в виде сетки символов. Используя поиск в глубину (DFS), программа эффективно находит и подсчитывает все изолированные грядки (обозначенные символом '#').