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 23

Выполнил:
Тарасов А.Н.
К3244

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Лабораторная работа №3.

Задача №3. Циклы [5 s, 512 Mb, 1 балл]

Учебная программа по инфокоммуникационным технологиям определяет пререквизиты для каждого курса в виде списка курсов, которые необходимо пройти перед тем, как начать этот курс. Вы хотите выполнить проверку согласованности учебного плана, то есть проверить отсутствие циклических зависимостей. Для этого строится следующий ориентированный граф: вершины соответствуют курсам, есть направленное ребро (u, v) – курс u следует пройти перед курсом v. Затем достаточно проверить, содержит ли полученный граф цикл. Проверьте, содержит ли данный граф циклы.

```
import time
nach = time.time()

def solve():
    f = open('input.txt')
    n, m = map(int, f.readline().split())
    sides = {}
    for i in range(1, n+1):
        sides[i] = []
    for i in range(m):
        v1, v2 = map(int, f.readline().split())
        sides[v1].append(v2)

    for u in sides:
        visited = []
        parent = {}
        cur_node = u
        node_found = 1
        node_completed = 0
        while True:
            visited.append(cur_node)
            flag = False
            for i in sides[cur_node]:
                if i == u:
                    return 1
                if i not in visited:
                    parent[i] = cur_node
                    cur_node = i
                    node_found += 1
                    flag = True
                    break
            if not flag:
                node_completed += 1
                if node_found == node_completed:
                    break
                cur_node = parent[cur_node]
    return 0
```

```
def main():
    with open('output.txt', 'w') as f:
        f.write(str(solve()))

if __name__ == "__main__":
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

Основная идея — попытаться пройти по всем рёбрам, начиная с каждой вершины, и проверять, возвращаемся ли мы в начальную точку (образуется ли цикл)

Задача №9. Аномалии курсов валют [10 s, 512 Mb, 2 балла]

Для заданного ориентированного графа с возможными отрицательными весами ребер, у которого n вершин и m ребер, проверьте, содержит ли он цикл с отрицательным суммарным весом.

```
import time

nach = time.time()


def main():
    def has_negative_cycle(graph, n):
        distances = [float('inf')] * n
        distances[0] = 0

        for _ in range(n - 1):
            for u in range(n):
                for v, weight in graph[u]:
                    if distances[u] + weight < distances[v]:
                        distances[v] = distances[u] + weight

        for u in range(n):
            for v, weight in graph[u]:
                if distances[u] + weight < distances[v]:
                    return 1

        return 0

    with open("input.txt", 'r') as file:
        n, m = map(int, file.readline().split())
        graph = [[] for _ in range(n)]
        for _ in range(m):
            u, v, weight = map(int, file.readline().split())
            graph[u - 1].append((v - 1, weight))
```

```
    result = has_negative_cycle(graph, n)
    with open("output.txt", "w") as f:
        f.write(str(result))


if __name__ == "__main__":
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

Задача №13. Грядки [1 s, 16 Mb, 3 балла]

Прямоугольный садовый участок шириной N и длиной M метров разбит на квадраты со стороной 1 метр. На этом участке вскопаны грядки. Грядкой называется совокупность квадратов, удовлетворяющая таким условиям: • из любого квадрата этой грядки можно попасть в любой другой квадрат этой же грядки, последовательно переходя по грядке из квадрата в квадрат через их общую сторону; • никакие две грядки не пересекаются и не касаются друг друга ни по вертикальной, ни по горизонтальной сторонам квадратов (касание грядок углами квадратов допускается). Подсчитайте количество грядок на садовом участке.

```
import time

nach = time.time()

def main():
    def garden_count(graph):
        counter = 0

        for i in range(n):
            for j in range(m):
                if graph[i][j] == '#':
                    counter += 1
                    stack = [(i, j)]
                    while stack:
                        x, y = stack.pop()
                        if graph[x][y] == '#':
                            graph[x][y] = '.'
                            if x > 0:
                                stack.append((x - 1, y))
                            if y > 0:
                                stack.append((x, y - 1))
                            if x < n - 1:
```

```
                                stack.append((x + 1, y))
                            if y < m - 1:
                                stack.append((x, y + 1))
        return counter


    with open("input.txt", "r", encoding='utf-8') as input_file,
open("output.txt", "w", encoding='utf-8') as output_file:
        n, m = map(int, input_file.readline().split())
        garden = [list(input_file.readline().strip()) for _ in range(n)]
        output_file.write(str(garden_count(garden)))

if __name__ == "__main__":
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

В функции проходи по элементам матрицы, если элемент это # т.е. Часть грядки - начинаем для него обход в ширину, исследуя соседние квадраты, в стек добавляем соседние квадраты, в цикле пока мы извлекаем элемент из стека и проверяем является ли он частью грядки #, если да помечаем как пустой “ . ”(посещенный), добавляем соседние необработанные элементы в стек(в зависимости от расположения слева, сверху, справа, снизу).

Дополнительные задачи

Задача №12. Цветной лабиринт [1 s, 16 Mb, 2 балла]

В одном из парков одного большого города недавно был организован новый аттракцион Цветной лабиринт. Он состоит из n комнат, соединенных m двунаправленными коридорами. Каждый из коридоров покрашен в один из ста цветов, при этом от каждой комнаты отходит не более одного коридора каждого цвета. При этом две комнаты могут быть соединены любым количеством коридоров. Человек, купивший билет на аттракцион, оказывается в комнате номер один. Кроме билета, он также получает описание пути, по которому он может выбраться из лабиринта. Это описание представляет собой последовательность цветов c1 . . . ck. Пользоваться ей надо так: находясь в комнате, надо посмотреть на очередной цвет в этой последовательности, выбрать коридор такого цвета и пойти по нему. При этом если из комнаты нельзя пойти по коридору

соответствующего цвета, то человеку приходится дальше самому выбирать, куда идти. В последнее время в администрацию парка стали часто поступать жалобы от заблудившихся в лабиринте людей. В связи с этим, возникла необходимость написания программы, проверяющей корректность описания и пути, и, в случае ее корректности, сообщающей номер комнаты, в которую ведет путь. Описание пути некорректно, если на пути, который оно описывает, возникает ситуация, когда из комнаты нельзя пойти по коридору соответствующего цвета.

```
import time

nach = time.time()

def main():
    def labyrinth(graph, colors):
        node = 1
        for color in colors:
            if color in graph[node]:
                node = graph[node][color]
            else:
                return 'INCORRECT'
        return node


    with open("input.txt", "r", encoding='utf-8') as input_file,
open("output.txt", "w", encoding='utf-8') as output_file:
        n, m = map(int, input_file.readline().split())
        graph = {i: {} for i in range(1, n + 1)}

        for _ in range(m):
            u, v, c = map(int, input_file.readline().split())
            graph[u][c] = v
            graph[v][c] = u


        input_file.readline()

        colors = [int(i) for i in input_file.readline().split()]

        output_file.write(str(labyrinth(graph, colors)))

if __name__ == "__main__":
    main()

kon = time.time()
c = kon - nach
print('Время :', c)

import os, psutil; print(psutil.Process(os.getpid()).memory_info().rss /
1024 ** 2)
```

Считываем входные данные: словарь graph, такой что ключи - это номера вершин, значения - это словари, где ключи - цвета коридоров, значения -

номера комнат(вершин), соединенных с текущей этим коридором. Считываем описание пути colors. Всегда начинаем с комнаты номер 1, как это сказано в условии. Перебираем все цвета в коридоров в маршруте: если такой цвет есть, берем из словаря новую вершину, если нет коридора данного цвета выходим и возвращаемся incorrect. Если цикл завершился, вернется вершина в которую ведет путь по цветам.

Вывод

В ходе лабораторной работы я научился решать задачи на графы.