



УНИВЕРСИТЕТ ИТМО

Факультет инфокоммуникационных технологий

Мобильные и сетевые технологии

Лабораторная работа №2.3

по дисциплине

«Алгоритмы и структуры данных»

Выполнила:

Юсипов Артем Александрович

К3140

Преподаватель:

Харьковская Т. А.

Санкт-Петербург

2022

Задача №4

Описание задания:

Дан ориентированный ациклический граф (DAG) с n вершинами и m ребрами. Выполните топологическую сортировку.

Решение:

```
import sys

sys.stdin = open("INPUT.TXT")
n, m = map(int, input().split())
G = {i: set() for i in range(1, n + 1)}
visired = [False] * (n + 1)
ans = []

for i in range(m):
    verge_1, verge_2 = map(int, input().split())
    G[verge_1].add(verge_2)

def dfs(start, G, visited, ans):
    visited[start] = True

    for u in G[start]:
        if not visited[u]:
            dfs(u, G, visited, ans)
    ans.append(start)

for i in range(1, n + 1):
    if not visired[i]:
        dfs(i, G, visired, ans)
print(ans[::-1])
```

Вывод:

Используем DFS. Так как данные добавляются на обратном ходе рекурсии, разворачиваем массив.

Задача №10

Описание задания:

Дан ориентированный граф с возможными отрицательными весами ребер, у которого n вершин и m ребер, а также задана одна его вершина s . Вычислите длину кратчайших путей из s во все остальные вершины графа.

Решение:

```
import sys

sys.stdin = open("INPUT.TXT")

class Graph:

    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = []

    def bellman_ford(self, src):
        dist = [float("Inf")] * self.vertices
        dist[src] = 0
        for _ in range(self.vertices - 1):
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w
        for u, v, w in self.graph:
            if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                dist[u] = 0
                dist[v] = 0
        g.print_array(dist, src)

    def print_array(self, dist, start):
        for i in range(self.vertices):
            if dist[i] != float("inf"):
                if i == start:
                    print(0)
                elif dist[i] == 0:
                    print("-")
                else:
                    print(dist[i])
            else:
                print("*")

n, m = map(int, input().split())
g = Graph(n)
for i in range(m):
    u, v, w = map(int, input().split())
    g.graph.append([u-1, v-1, w])
start = int(input())-1
g.bellman_ford(start)
```

Вывод:

Используем алгоритм Беллмана – Форда. С его помощью находим пути, с учетом отрицательных рёбер.

Задача №15

Описание задания:

Коварный кардинал Ришелье вновь организовал похищение подвесок королевы Анны; вновь спасти королеву приходится героическим мушкетерам. Атос, Портос, Арамис и д'Артаньян уже перехватили агентов кардинала и вернули украденное; осталось лишь передать подвески королеве Анне. Королева ждет мушкетеров в дворцовом саду. Дворцовый сад имеет форму прямоугольника и разбит на участки, представляющие собой небольшие садики, содержащие коллекции растений из разных климатических зон. К сожалению, на некоторых участках, в том числе на всех участках, расположенных на границах сада, уже притаились в засаде гвардейцы кардинала; на бой с ними времени у мушкетеров нет. Мушкетерам удалось добыть карту сада с отмеченными местами засад; теперь им предстоит выбрать оптимальные пути к королеве. Для надежности друзья разделили между собой спасенные подвески и проникли в сад поодиночке, поэтому начинают свой путь к королеве с разных участков сада. Двигаются герои по максимально короткой возможной траектории. Марлезонский балет вот-вот начнется; королева не в состоянии ждать героев больше L минут; ровно в начале $L+1$ -ой минуты королева покинет парк, и те мушкетеры, что не успеют к этому времени до нее добраться, не смогут передать ей подвески. На преодоление одного участка у мушкетеров уйдет ровно по минуте. С каждого участка мушкетеры могут перейти на 4 соседние. Требуется выяснить, сколько подвесок будет красоваться на платье королевы, когда она придет на бал.

Решение:

```
from collections import deque
import sys

sys.stdin = open("INPUT.TXT")
m, n = map(int, input().split())
graph = [[char for char in input()] for i in range(m)]
q_x, q_y, max_len = map(int, input().split())
l = [[-1 for i in range(n)] for j in range(m)]
q_x, q_y = q_x - 1, q_y - 1
q = deque()
l[q_x][q_y] = 0
q.append([q_x, q_y])
while q:
    cur = q.popleft()
    for elem_i in range(-1, 2):
        for elem_j in range(-1, 2):
            if elem_i * elem_i + elem_j * elem_j == 1:
                ni = cur[0] + elem_i
                nj = cur[1] + elem_j
                if l[ni][nj] == -1 and graph[ni][nj] == '0':
                    l[ni][nj] = l[cur[0]][cur[1]] + 1
                    q.append([ni, nj])
ans = 0
for i in range(4):
    m_x, m_y, extra = map(int, input().split())
    m_x, m_y = m_x - 1, m_y - 1
    if l[m_x][m_y] <= max_len and l[m_x][m_y] != -1:
        ans += extra
print(ans)
```

Задача №16

Описание задания:

Рассмотрим программу, состоящую из n процедур P_1, P_2, \dots, P_n . Пусть для каждой процедуры известны процедуры, которые она может вызывать. Процедура P называется потенциально рекурсивной, если существует такая последовательность процедур Q_0, Q_1, \dots, Q_k , что $Q_0 = Q_k = P$ и для $i = 1 \dots k$ процедура Q_{i-1} может вызвать процедуру Q_i . В этом случае задача будет заключаться в определении для каждой из заданных процедур, является ли она потенциально рекурсивной. Требуется написать программу, которая позволит решить названную задачу.

Решение:

```
from collections import deque
import sys

sys.stdin = open("INPUT.TXT")
sys.stdout = open("OUTPUT.TXT", 'w')
n = int(input())
main_graph = dict()
for i in range(n):
    main_p = input()
    main_graph[main_p] = set()
    m = int(input())
    for j in range(m):
        main_graph[main_p].add(input())
    input()

def is_recursive_function(graph, function):
    queue = deque()
    visited = set()
    inqueue = set()
    queue.append(function)
    inqueue.add(function)
    while queue:
        c = queue.popleft()
        inqueue.remove(c)
        visited.add(c)
        for vertex in graph[c]:
            if vertex == function:
                return True
            if vertex not in visited and vertex not in inqueue:
                queue.append(vertex)
                inqueue.add(vertex)
    return False

for function in main_graph:
    if is_recursive_function(main_graph, function):
        print("YES")
    else:
        print("NO")
```

Задача №17

Описание задания:

Ане, как будущей чемпионке мира по программированию, поручили очень ответственное задание. Правительство вручает ей план постройки дорог между N городами. По плану все дороги односторонние, но между двумя городами может быть больше одной дороги, возможно, в разных направлениях. Ане необходимо вычислить минимальное такое K , что данный ей план является слабо K -связным. Правительство называет план слабо K -связным, если выполнено следующее условие: для любых двух различных городов можно проехать от одного до другого, нарушая правила движения не более K раз. Нарушение правил – это проезд по существующей дороге в обратном направлении. Гарантируется, что между любыми двумя городами можно проехать, возможно, несколько раз нарушив правила.

Решение:

```
import sys

def main():
    sys.stdin = open("INPUT.TXT")
    sys.stdout = open("OUTPUT.TXT", 'w')
    n, m = map(int, input().split())
    inf = float('inf')
    graph = [[inf for _ in range(n)] for _ in range(n)]
    for i in range(m):
        town_1, town_2 = map(int, input().split())
        town_1 -= 1
        town_2 -= 1
        graph[town_1][town_2] = 0
        graph[town_2][town_1] = min(graph[town_2][town_1], 1)
    for k in range(n):
        for i in range(n):
            for j in range(n):
                graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j])
    ans = 0
    for i in range(n):
        for j in range(n):
            ans = max(ans, graph[i][j])
    print(ans)

main()
```

Задача №13

Описание задания:

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.

Решение:

```
from collections import deque
import sys

sys.stdin = open("INPUT.TXT")
sys.stdout = open("OUTPUT.TXT", 'w')
check = "#"
unchecked = "."
n, m = map(int, input().split())
graph = [[elem for elem in input()] for i in range(n)]
move_x = [-1, 0, 1, 0]
move_y = [0, -1, 0, 1]

def correct(x, y):
    if x < 0 or y < 0:
        return False
    if x >= n or y >= m:
        return False
    return True

def DFS(fx, fy):
    s = deque()
    s.append([fx, fy])
    while s:
        cur = s.popleft()
        for i in range(4):
            x = cur[0] + move_x[i]
            y = cur[1] + move_y[i]
            if correct(x, y) and graph[x][y] == check:
                graph[x][y] = unchecked
                s.append([x, y])

def main():
    amount = 0
    for i in range(n):
        for j in range(m):
            if graph[i][j] == check:
                DFS(i, j)
                amount += 1
    print(amount)

main()
```

Задача №12

Описание задания:

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 s 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 $s_1 \dots s_k$. 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.

Решение:

```
import sys

sys.stdin = open("INPUT.TXT")
sys.stdout = open("OUTPUT.TXT", 'w')
n, m = map(int, input().split())
maze = [[0 for j in range(100 + 1)] for i in range(n+1)]
for i in range(m):
    room1, room2, color = map(int, input().split())
    maze[room1][color] = room2
    maze[room2][color] = room1
l = input()
cur = 1
path = list(map(int, input().split()))
for color in path:
    cur = maze[cur][color]
if cur == 0:
    print("INCORRECT")
else:
    print(cur)
```

Задача №11

Описание задания:

Результатом алхимической реакции является превращение одного вещества в другое. Задан набор алхимических реакций, описанных на найденных глиняных табличках, исходное вещество и требуемое вещество. Необходимо выяснить: возможно ли преобразовать исходное вещество в требуемое с помощью этого набора реакций, а в случае положительного ответа на этот вопрос – найти минимальное количество реакций, необходимое для осуществления такого преобразования.

Решение:

```
from collections import deque
import sys

sys.stdin = open("INPUT.TXT")
sys.stdout = open("OUTPUT.TXT", 'w')

n = int(input())
graph = dict()
elem_set = set()
for i in range(n):
    elem_from, temp, elem_to = input().split()
    elem_set.add(elem_from)
    elem_set.add(elem_to)
    if elem_from not in graph:
        graph[elem_from] = set()
        graph[elem_from].add(elem_to)
    else:
        graph[elem_from].add(elem_to)
start_elem = input()
final_elem = input()
if start_elem == final_elem:
    print(0)
else:
    def BFS(start_vertex, graph, set_el):
        distances = dict()
        parents = dict()
        for vertex in set_el:
            distances[vertex] = None
            parents[vertex] = []
        distances[start_vertex] = 0
        queue = deque([start_vertex])
        while queue:
            cur_v = queue.popleft()
            if cur_v in graph:
                for neigh_v in graph[cur_v]:
                    if distances[neigh_v] is None:
                        distances[neigh_v] = distances[cur_v] + 1
                        parents[neigh_v].append(cur_v)
                        queue.append(neigh_v)
        return parents
    parents = BFS(start_elem, graph, elem_set)
    path = [final_elem]
    if final_elem in parents and parents[final_elem]:
        cur_parent = parents[final_elem][0]
        while parents[cur_parent]:
            path.append(cur_parent)
            cur_parent = parents[cur_parent][0]
        path.append(start_elem)
        print(len(path)-1)
    else:
        print(-1)
```

Вывод:

Нам задан граф и просят найти кратчайший маршрут (если он существует).

Задача №9

Описание задания:

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.

Решение:

```
import sys

sys.stdin = open("INPUT.TXT")

class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = []

    def bellman_ford(self, src):
        dist = [float("Inf")] * self.vertices
        dist[src] = 0
        for _ in range(self.vertices - 1):
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                    print(1)
                    return
        print(0)

n, m = map(int, input().split())
g = Graph(n)

for i in range(m):
    u, v, w = map(int, input().split())
    g.graph.append([u - 1, v - 1, w])

g.bellman_ford(0)
```

Вывод:

Используем алгоритм Беллмана – Форда. С его помощью находим отрицательный цикл.