Санкт-Петербургский Национальный
Исследовательский Университет Информационных Технологий,
Механики и Оптики

Алгоритмы и структуры данных

Лабораторная работа №2.3

Выполнил:
Кононов С. В.
Группа:
К3140
Преподаватель:
Харьковская Т. А.

Санкт-Петербург,
21 апреля 2022

Задача № 1

Описание задания:

Вам дан неориентированный граф и две различные вершины u и v. Проверьте, есть ли путь между u и v.

Решение:

```
import sys

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

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

graph = {str(i): set() for i in range(1, n + 1)}

for i in range(m):
    verge_1, verge_2 = input().split()
    graph[verge_1].add(verge_2)
    graph[verge_2].add(verge_1)
u, v = input().split()


def DFS(start_vertex, graph, used):
    used.add(start_vertex)
    for neighbor in graph[start_vertex]:
        if neighbor not in used:
            DFS(neighbor, graph, used)


n = 0
used = set()

DFS(u, graph, used)

if v not in used:
    print(0)
else:
    print(1)
```

Вывод:

Результаты всех тестов, совпадают с ответами.

Задача № 2

Описание задания:

Дан неориентированный граф с n вершинами и m ребрами. Нужно посчитать количество компонент свзяности в нем.

Решение:

```
import sys

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

n, m = map(int, input().split())
graph = {str(i): set() for i in range(1, n + 1)}

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


def DFS(start_vertex, graph, used):
    used.add(start_vertex)
    for neighbor in graph[start_vertex]:
        if neighbor not in used:
            DFS(neighbor, graph, used)


used = set()
N = 0
for vertex in graph:
    if vertex not in used:
        DFS(vertex, graph, used)
        N += 1
print(N)
```

Вывод:

Для решения задачи я использовал DFS, обходя все вершины графа и вычеркиваем их. Если существует вершина которую не посетили, то увеличиваем счётчик и запускаем обход.

Задача № 3

Описание задания:

Проверьте, содержит ли данный граф циклы.

Решение:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.colour = 0
        self.node_list = []


def main():
    input_file = open("INPUT.TXT")
    arr = []
    n, m = map(int, input_file.readline().split())
    for i in range(1, n + 1):
        arr.append(Node(i))
    for s in input_file.readlines():
        a, b = map(int, s.split())
        arr[a - 1].node_list.append(arr[b - 1])
    dfs(arr[0])
    print(0)


def dfs(vertex):
    vertex.colour = 1
    for cur in vertex.node_list:
        if cur.colour == 1:
            print(1)
            exit()
        if cur.colour == 0:
            dfs(cur)
    vertex.colour = 2


main()
```

Вывод:

Результаты всех тестов, совпадают с ответами.

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 № 6

Описание задания:

Дан неориентированный граф с n вершинами и m ребрами, а также две вершины u и v, нужно посчитать длину кратчайшего пути между u и v (то есть, минимальное количество ребер в пути из u в v).

Решение:

```
from collections import deque
import sys

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

n, m = map(int, input().split())
graph = {str(i): set() for i in range(1, n + 1)}

for i in range(m):
    verge_1, verge_2 = input().split()
    graph[verge_1].add(verge_2)
    graph[verge_2].add(verge_1)
u, v = input().split()


def BFS(start_vertex, graph):
    distances = dict()
    parents = dict()
    for vertex in graph:
        distances[vertex] = None
        parents[vertex] = []
    distances[start_vertex] = 0
    queue = deque([start_vertex])
    while queue:
        cur_v = queue.popleft()
        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)
    #print(parents)
    return parents


parents = BFS(v, graph)

end_vertex = u
path = [end_vertex]

if parents[end_vertex]:
    cur_parent = parents[end_vertex][0]

    while parents[cur_parent]:
        path.append(cur_parent)
        cur_parent = parents[cur_parent][0]
    path.append(v)
    print(len(path)-1)
else:
    print(-1)
```

Вывод:

Используем простой BFS, сохраняя предков, для последующего восстановления пути.

Задача № 7

Описание задания:

Дан неориентированный граф с n вершинами и m ребрами, проверьте, является ли он двудольным.

Решение:

```
import sys

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

n, m = map(int, input().split())
graph = {str(i): set() for i in range(1, n + 1)}

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


used = dict()
colors = [1, 0]
def bipartite_graph_check(start_vertex, graph, used, color):
    used[start_vertex] = colors[color]
    for neighbor in graph[start_vertex]:
        if neighbor in used:
            if colors[color] == used[neighbor]:
                return False
        else:
            bipartite_graph_check(neighbor, graph, used, colors[color])
    return True


all_bipartite = True
for vertex in graph:
    if vertex not in used:
        all_bipartite = all_bipartite and bipartite_graph_check(vertex,
graph, used, 0)
if all_bipartite:
    print(1)
else:
    print(0)
```

Вывод:

Используем DFS с раскраской.

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(u-1)
```

Вывод:

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

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 № 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.

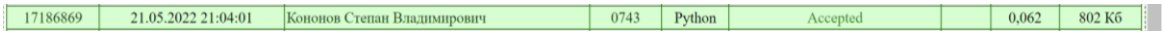
Решение:

```
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:

По сути нам задан граф и просят найти кратчайший маршрут (если он существует).



17186869	21.05.2022 21:04:01	Кононов Степан Владимирович	0743	Python	Accepted		0,062	802 Кб

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 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.

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)
```

```
#include <stdio.h>
#include <vector>

int main(){
    int n, m;
    scanf("%d %d", &n, &m);
    const int colors = 100;
    std:: vector<std::vector<int>> matrix(1+n, std::vector<int>(1+colors,
0));
    for (int i = 0; i < m; i++){
        int v1,v2, color;
        scanf("%d %d %d", &v1, &v2, &color);
        matrix[v1][color] = v2;
        matrix[v2][color] = v1;
    }
    int steps;
    scanf("%d", &steps);
    int cur = 1;
    for(int i = 0; i<steps; i++){
        int color;
        scanf("%d", &color);
        cur = matrix[cur][color];
    }
    if (cur == 0){
        printf("INCORRECT");
    }
    else{
        printf("%d", cur);
    }

}
```

Вывод:

Решение на Python выходит за пределы памяти

Гостевая	17186912	21.05.2022 21:16:22	Кононов Степан Владимирович	0601	Python	Memory limit exceeded	21	0,187	17 Мб

Решение на C++

Мои задачи Баланс	17186916	21.05.2022 21:17:47	Кононов Степан Владимирович	0601	C++	Accepted		0,186	5016 Кб

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).

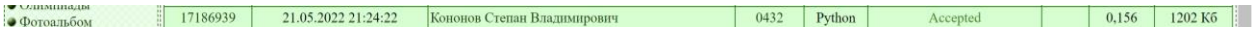
Подсчитайте количество грядок на садовом участке.

Решение:

```
from collections import deque
import sys

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

check = "#"
uncheck = "."

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] = uncheck
                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()
```

Вывод:



Задача № 15

Описание задания:

Коварный кардинал Ришелье вновь организовал похищение подвесок королевы Анны; вновь спасать 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, exrtra = 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 += exrtra
print(ans)
```

Вывод:

Паспорт	17186974	21.05.2022 21:33:05	Кононов Степан Владимирович	0846	Python	Accepted	0,046	794 Кб

Задача № 16

Описание задания:

Рассмотрим программу, состоящую из n процедур P1, P2, . . . , Pn. Пусть для каждой процедуры известны процедуры, которые она может вызывать. Процедура P называется потенциально рекурсивной, если существует такая последовательность процедур Q0, Q1, . . . , Qk, что Q0 = Qk = P и для i = 1...k процедура Qi−1 может вызвать процедуру Qi. В этом случае задача будет заключаться в определении для каждой из заданных процедур, является лиона 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")
```

Вывод:

- Правила
- Олимпиады
- Фотоальбом

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
17187001	21.05.2022 21:39:11	Кононов Степан Владимирович	0345	Python	Accepted		0,062	3310 Кб

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.

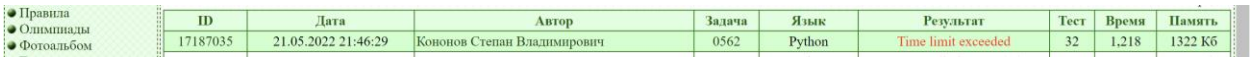
Решение:

```
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 j in range(n)] for i 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()
```

```
#include <stdio.h>
#include <vector>
#include <algorithm>

int main(){
    int n,m;
    scanf("%d %d", &n, &m);
    const int INF = 1000*1000*1000;
    std::vector<std::vector<int>> graph (n, std::vector<int> (n, INF));
    for (int i = 0; i < n; ++i) {
        graph[i][i] =0;
    }

    for (int i = 0; i < m; ++i) {
        int to_1, to_2;
        scanf("%d %d", &to_1, &to_2);
        to_1--;
        to_2--;
        graph[to_1][to_2] = 0;
        graph[to_2][to_1] = std::min(graph[to_2][to_1], 1);
    }

    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                graph[i][j] = std::min(graph[i][j], graph[i][k] +
graph[k][j]);
            }
        }
    }

    int max = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            max = std::max(max, graph[i][j]);
        }
    }
    printf("%d", max);
    return 0;
}
```

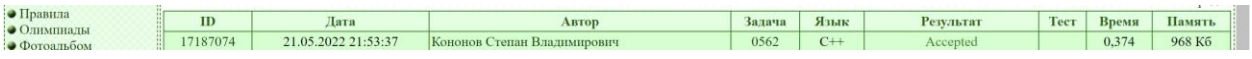
Вывод:

Решение на Python



ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
17187035	21.05.2022 21:46:29	Кононов Степан Владимирович	0562	Python	Time limit exceeded	32	1,218	1322 Кб

Решение на C++



ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
17187074	21.05.2022 21:53:37	Кононов Степан Владимирович	0562	C++	Accepted		0,374	968 Кб