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 17

Выполнил:

Прокопец Семен Романович 3w

K3139

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

Содержание отчета

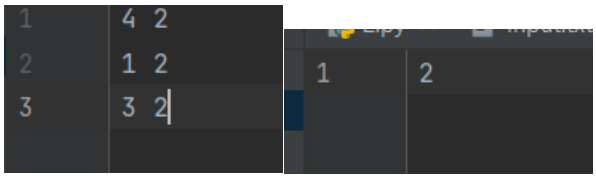
Задачи по варианту

2 Задача. Компоненты [5 s, 512 Mb, 1 балл]

Теперь вы решаете сделать так, чтобы в лабиринте не было мертвых зон, то есть чтобы из каждой клетки был
доступен хотя бы один выход. Для этого вы находите связные компоненты соответствующего неориентированного
графа и следите за тем, чтобы каждый компонент содержал выходную ячейку.
Дан неориентированный граф с n вершинами и m ребрами. Нужно посчитать количество компонент связности в
нем.

```
import time
start_time = time.time()
inp = open("input.txt")
out = open("output.txt", "w")
n, m = map(int, inp.readline().split())
graph = {str(i): set() for i in range(1, n + 1)}
for i in range(m):
    edge1, edge2 = inp.readline().split()
    graph[edge1].add(edge2)
    graph[edge2].add(edge1)
def dfs(start, graph, bypassed):
    bypassed.add(start)
    for neighbor in graph[start]:
```

```
        if neighbor not in bypassed:
            dfs(neighbor, graph, bypassed)
bypassed = set()
amount = 0
for vertex in graph:
    if vertex not in bypassed:
        dfs(vertex, graph, bypassed)
        amount += 1
out.write(str(amount))
print(time.time() - start_time)
```



8 Задача. Стоимость полета [10 s, 512 Mb, 1.5 балла]

Теперь вас интересует минимизация не количества пересадок, а общей стоимости полета. Для этого строится
взвешенный граф: вес ребра из одного города в другой – это стоимость соответствующего перелета.
Дан ориентированный граф с положительными весами ребер, n - количество вершин и m - количество ребер, а
также даны две вершины u и v. Вычислить вес кратчайшего пути между u и v (то есть минимальный общий вес пути
из u в v).

```
from collections import defaultdict


with open("input.txt", "r") as inp:
    n, m = map(int, inp.readline().split())
    list1 = []
    for i in range(m):
        d1, d2, d3 = map(int, inp.readline().split())
        list1.append([d1 - 1, d2 - 1, d3])

    a, b = map(int, inp.readline().split())
    a -= 1
    b -= 1


class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = defaultdict(list)

    def addEdge(self, u, v, w):
        self.graph[u].append((v, w))

    def topologicalSortUtil(self, v, visited, stack):
```

```
        visited[v] = True
        if v in self.graph.keys():
            for node, weight in self.graph[v]:
                if not visited[node]:
                    self.topologicalSortUtil(node, visited, stack)
        stack.append(v)

    def shortestPath(self, s):
        visited = [False] * self.V
        stack = []
        for i in range(self.V):
            if not visited[i]:
                self.topologicalSortUtil(s, visited, stack)
        dist = [10 ** 18] * self.V
        dist[s] = 0

        while stack:
            i = stack.pop()
            for node, weight in self.graph[i]:
                if dist[node] > dist[i] + weight:
                    dist[node] = dist[i] + weight

        l = []
        for i in range(self.V):
            c = dist[i] if dist[i] != 10 ** 18 else 10 ** 18
            l.append(c)
        return l


g = Graph(n)
for i in list1:
    g.addEdge(i[0], i[1], i[2])

result = g.shortestPath(a)
answ = result[b]
with open("output.txt", "w") as outp:
    if answ == 10 ** 18:
        print(-1)
        outp.write(str(-1))
    else:
        print(answ)
        outp.write(str(answ))
```

```
4 4
1 2 1
4 1 2
2 3 2
1 3 5
1 3
```

```
1
3
```

Задача №7. Двудольный граф [1.5 баллов]

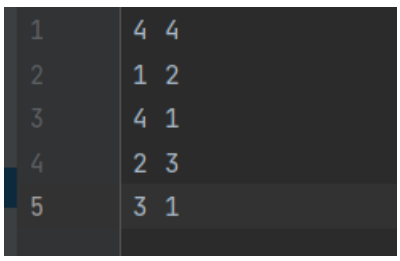
Текст задачи.

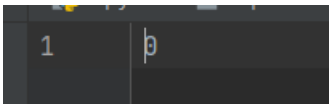
```
from collections import deque

def halfGraph(u):
    global sides, total_colours
    search_queue = deque()
    search_queue.append((u, 0))
    visited = []
    while search_queue:
        cur_node, colour = search_queue.popleft()
        if cur_node not in visited:
            total_colours[cur_node] = colour
            visited.append(cur_node)
            for node in sides[cur_node]:
                if colour == 0:
                    search_queue.append((node, 1))
                else:
                    search_queue.append((node, 0))
        elif total_colours[cur_node] != colour:
            return 0
    return 1

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

total_colours = [None] * (n+1)
result = halfGraph(1)
print(result)
```





Задача №14. Оптимальный обмен валюты [2 баллов]

Марии Ивановне требуется добраться из деревни d в деревню v как можно быстрее (считается, что в момент
времени 0 она находится в деревне d).

```
import math

file = open("input.txt")
```

```
c_villages = int(file.readline())
start, end = map(int, file.readline().split())
c_races = int(file.readline())

buses = [[] for _ in range(c_races)]

for _ in range(c_races):
    src, src_time, dst, dst_time = map(int, file.readline().split())
    buses[src - 1].append((src_time, dst - 1, dst_time))

times = [math.inf] * (c_villages)
times[start - 1] = 0

used = [False] * (c_villages)
while True:
    min_time = math.inf
    for i in range(c_villages):
        if not used[i] and times[i] < min_time:
            min_time = times[i]
            min_village = i
    if min_time == math.inf:
        break
    src = min_village
    used[src] = True
    for src_time, dst, dst_time in buses[src]:
        if times[src] <= src_time and dst_time < times[dst]:
            times[dst] = dst_time

f = open("output.txt", "w")
if times[end - 1] == math.inf:
    f.write("-1")
else:
    f.write(str(times[end - 1]))
```

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19070110	22.03.2023 14:42:35	Прокопец Семен Романовичч	0134	Python	Accepted		0,046	1782 Кб

Задача №3. Циклы [1 баллов]

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.

```
from collections import defaultdict

def dfs(v, p=-1):
    used[v] = True
    for u in graph[v]:
```

```
        if not used[u]:
            dfs(u, v)
        elif u != p:
            f.write("1")
            exit()
    used[v] = False

graph = {}

with open("input.txt") as file:
    n, m = map(int, file.readline().split())
    for i in range(n):
        graph[i + 1] = []
    for i in range(m):
        u, v = map(int, file.readline().split())
        graph[u].append(v)

f = open("output.txt", "w")

used = defaultdict(lambda: False)

for i in range(n):
    if not used[i + 1]:
        dfs(i + 1)

f.write("0")
```