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 8

Выполнил:
Макунина А.А.
К32421

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №3. Циклы (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.

- **Формат ввода / входного файла (input.txt).** Ориентированный граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \leq n \leq 10^3, 0 \leq m \leq 10^3$.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если данный граф содержит цикл; выведите 0, если не содержит.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.

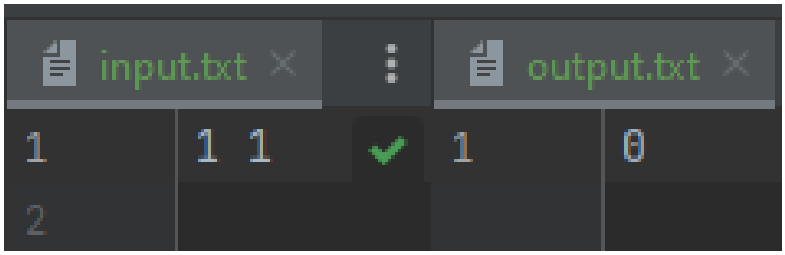
Как это работает – все просто – если ребро ведет уже в посещенную вершину, то вауля – мы нашли цикл.

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


def dfs(v: Node):
    v.colour = 1
    for u in v.family:
        if u.colour == 1:
            w.write('1')
            exit()
        if u.colour == 0:
            dfs(u)
    v.colour = 2


f = open('input.txt')
w = open('output.txt', 'w')
t_start = time.perf_counter()
tracemalloc.start()
arr = []
n, m = map(int, f.readline().split())
```

```
for i in range(1, n + 1):
    arr.append(Node(i))
for s in f.readlines():
    a, b = map(int, s.split())
    arr[a - 1].family.append(arr[b - 1])
dfs(arr[0])
w.write('0')
```

Результат работы кода на примере из текста задачи:

Результат работы кода при минимальном и максимальном значениях:



	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0006845000000000045	0.008440971374511719
Пример из задачи	0.0002445999999999976	0.009097099304199219
Пример из задачи	0.0004687000000000245	0.009466171264648438

Вывод по задаче: я не придумала как сгенерировать файл на такое количество строк, чтобы граф был простым. Сама задача простенькая.

Задача №10. Оптимальный обмен валюты (2 балла)

Теперь вы хотите вычислить оптимальный способ обмена данной вам валюты c_i на *все* другие валюты. Для этого вы находите кратчайшие пути из вершины c_i во все остальные вершины.

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.

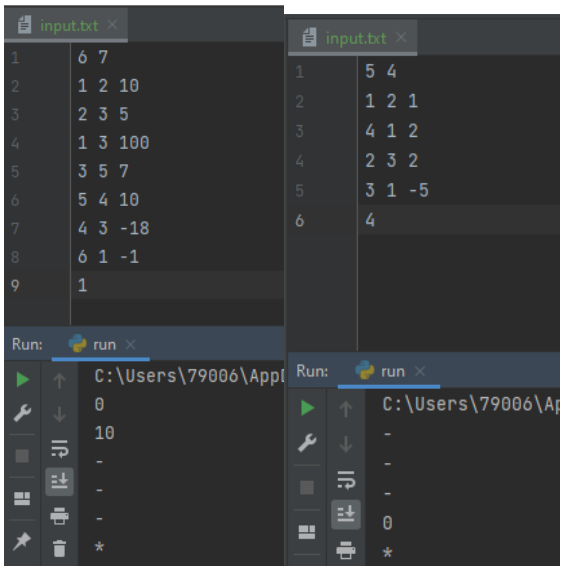
- **Формат ввода / входного файла (input.txt).** Ориентированный взвешенный граф задан по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^3$, $0 \le m \le 10^4$, $1 \le s \le n$, вес каждого ребра – целое число, не превосходящее ***по модулю*** 10^9.
- **Формат вывода / выходного файла (output.txt).** Для каждой вершины i графа от 1 до n выведите в каждой отдельной строке следующее:
  - «*», если пути из s в i нет;
  - «-», если существует путь из s в i, но нет кратчайшего пути из s в i (то есть расстояние от s до i равно $-\infty$);
  - длину кратчайшего пути в остальных случаях.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.

Используется алгоритм Беллмана-Форда. Как и в алгоритме Дейкстры, создается таблица, в которой записывается расстояние до каждой вершины и предыдущей вершины. Если есть отрицательный цикл, то мы присвоим длине пути этих вершин что-то отличное от обычной длины (тогда это поможет нам определить, когда мы будем выводить его, куда ставить «-»).

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

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

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

    def BellmanFord(self, src):
```

```
        dist = [float("inf")] * self.V
        dist[src] = 0
        for i in range(self.V - 1):
            for u, v, w in self.graph:
                if dist[u - 1] != float("inf") and
dist[u - 1] + w < dist[v - 1]:
                    dist[v - 1] = dist[u - 1] + w
        for u, v, w in self.graph:
            if dist[u - 1] != float("inf") and dist[u
- 1] + w < dist[v - 1]:
                dist[u - 1] = 0
                dist[v - 1] = 0
        # вывод всего расстояние
        self.printArr(dist, src)


f = open('input.txt')
t_start = time.perf_counter()
tracemalloc.start()
s = f.readlines()
n, m = map(int, s[0].split())
f.close()
g = Graph(n)
root = int(s[-1]) - 1
for i in range(1, m + 1):
    u, v, w = map(int, s[i].split())
    g.addEdge(u, v, w)
g.BellmanFord(root)
```

Результат работы кода на примере из текста задачи:



	Время выполнения, с	Затраты памяти, мб
Пример 1 из задачи	0.0004388999999999921	0.008714675903320312
Пример 2 из задачи	0.00029830000000000134	0.008521080017089844

Вывод по задаче: тесты выполнены успешно!

Задача №13. Грядки (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.

- **Формат входных данных (input.txt) и ограничения.** В первой строке входного файла INPUT.TXT находятся числа N и M через пробел, далее идут N строк по M символов. Символ # обозначает территорию грядки, точка соответствует незанятой территории. Других символов в исходном файле нет ($1 \le N, M \le 200$).
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите количество грядок на садовом участке.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.

Происходит считывание исходного поля в массив строк. Создание двумерного массива проходящих ячеек поля. Далее пройдите по всем ячейкам. Если мы встречаем грядку, мы увеличиваем счетчик грядок, ищем и добавляем все ячейки этой грядки. В конце мы выводим количество грядок.

```
f = open('input.txt')
n, m = map(int, f.readline().split())
array = []
for i in range(n):
    array.append(f.readline()[:m])
f.close()
visited = [[False for i in range(m)] for j in
range(n)]
count = 0
for i in range(n):
    for j in range(m):
```

```
        if not visited[i][j]:
            if array[i][j] == '#':
                count += 1
                a = [(i, j)]
                while len(a) != 0:
                    x, y = a.pop()
                    if array[x][y] == '#' and not
visited[x][y]:
                        visited[x][y] = True
                        if x != 0:
                            a.append((x - 1, y))
                        if x != n - 1:
                            a.append((x + 1, y))
                        if y != 0:
                            a.append((x, y - 1))
                        if y != m - 1:
                            a.append((x, y + 1))

w = open('output.txt', 'w')
w.write(str(count))
w.close()
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,015	354 Кб
2	Accepted	0,031	362 Кб
3	Accepted	0,015	350 Кб
4	Accepted	0,093	782 Кб
5	Accepted	0,062	2778 Кб
6	Accepted	0,046	1674 Кб
7	Accepted	0,14	4614 Кб
8	Accepted	0,125	1626 Кб
9	Accepted	0,125	3318 Кб
10	Accepted	0,125	734 Кб
11	Accepted	0,109	734 Кб
12	Accepted	0,031	362 Кб

Вывод по задаче: задача реализована успешно, что меня несомненно радует!

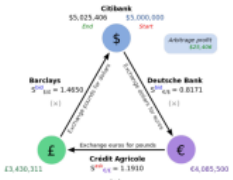
Дополнительные задачи

Задача №9. Аномалии курсов валют (2 балла)

Вам дан список валют $c_1, c_2, \ldots, c_n$ вместе со списком обменных курсов: r_{ij} – количество единиц валюты c_j, которое можно получить за одну единицу c_i. Вы хотите проверить, можно ли начать делать обмен с одной единицы какой-либо валюты, выполнить последовательность обменов и получить более одной единицы той же валюты, с которой вы начали обмен. Другими словами, вы хотите найти валюты $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ такие, что $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot \cdots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$.

Для этого построим следующий граф: вершинами являются валюты $c_1, c_2, \ldots, c_n$, вес ребра из c_i в c_j равен $-\log r_{ij}$. Тогда достаточно проверить, есть ли в этом графе отрицательный цикл. Пусть цикл $c_i \to c_j \to c_k \to c_i$ имеет отрицательный вес. Это означает, что $-(\log c_{ij} + \log c_{jk} + \log c_{ki}) < 0$ и, следовательно, $\log c_{ij} + \log c_{jk} + \log c_{ki} > 0$. Это, в свою очередь, означает, что

$$r_{ij} r_{jk} r_{ki} = 2^{\log c_{ij}} 2^{\log c_{jk}} 2^{\log c_{ki}} = 2^{\log c_{ij} + \log c_{jk} + \log c_{ki}} > 1.$$



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.

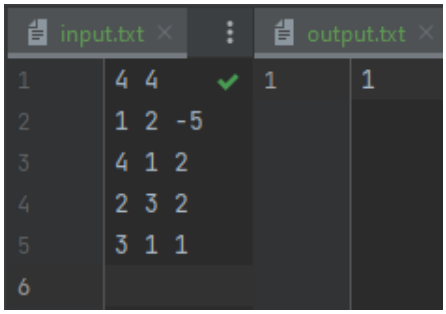
- **Формат ввода / входного файла (input.txt).** Ориентированный взвешенный граф задан по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^3$, $0 \le m \le 10^4$, вес каждого ребра – целое число, не превосходящее ***по модулю*** 10^4.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если граф содержит цикл с отрицательным суммарным весом. Выведите 0 в противном случае.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.

Я уже использовала этот алгоритм в задаче № 10, только здесь проще – если мы встречаем граф, содержащий цикл с отрицательным общим весом, мы заканчиваем работу и пишем «1» в ответ, в противном случае мы пишем «0».

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

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

    def bellman_ford(self, src):
        dist = [float("Inf")] * self.V
        dist[src] = 0
```

```
        for i in range(self.V - 1):
            for u, v, w in self.graph:
                if dist[u - 1] != float("inf") and
dist[u - 1] + w < dist[v - 1]:
                    dist[v - 1] = dist[u - 1] + w
        for u, v, w in self.graph:
            if dist[u - 1] != float("inf") and dist[u
- 1] + w < dist[v - 1]:
                return '1'
        return '0'


f = open('input.txt')
vertices, edges = f.readline().split()
graph = Graph(int(vertices))
for line in f:
    currency_i, currency_j, cost = map(int,
line.split())
    graph.add_edge(currency_i, currency_j, cost)
f.close()
result = graph.bellman_ford(1)
w = open('output.txt', 'w')
w.write(result)
w.close()
```

Результат работы коды на примерах из задачи:



	Время выполнения, с	Затраты памяти, мб
Пример из задачи	0.0010809999999999986	0.011953353881835938

Вывод по задаче: хорошая задача, потому что не пришлось переписывать её с нуля – ура!

Задача №11. Алхимия (3 балла)

Алхимики средневековья владели знаниями о превращении различных химических веществ друг в друга. Это подтверждают и недавние исследования археологов.

В ходе археологических раскопок было обнаружено m глиняных табличек, каждая из которых была покрыта непонятными на первый взгляд символами. В результате расшифровки выяснилось, что каждая из табличек описывает одну алхимическую реакцию, которую умели проводить алхимики.

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.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит целое число m ($0 \leq m \leq 1000$) – количество записей в книге. Каждая из последующих m строк описывает одну алхимическую реакцию и имеет формат «вещество1 -> вещество2», где «вещество1» – название исходного вещества, «вещество2» – название продукта алхимической реакции. $m + 2$-ая строка входного файла содержит название вещества, которое имеется исходно, $m + 3$-ая – название вещества, которое требуется получить.

  Во входном файле упоминается не более 100 различных веществ. Название каждого из веществ состоит из строчных и заглавных английских букв и имеет длину не более 20 символов. Строчные и заглавные буквы различаются.

- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите минимальное количество алхимических реакций, которое требуется для получения требуемого вещества из исходного, или -1, если требуемое вещество невозможно получить.

- Ограничение по времени. 1 сек.

- Ограничение по памяти. 16 мб.

Мы строим кратчайший возможный путь в графе, вершины которого являются элементами, а ребра – возможностью получить другое из одного вещества. Если путь найден, мы выводим его (он всегда будет минимальным, так как мы находим минимальный путь на каждом шаге), если нет, мы выводим -1.

```
file = open('input.txt')
m = int(file.readline())
arr, d = [], {}
for _ in range(m):
    first, arrow, second = file.readline().split()
    arr.append((first, second, 1))
    if first not in d.keys():
        d[first] = None
    if second not in d.keys():
        d[second] = None
f, s = file.readline().split()[0],
file.readline().split()[0]
file.close()
d[f] = 0
for i in range(len(d.keys())):
    for first, second, w in arr:
        if d[first] is not None:
```

```
            d[second] = min(10000000 if (d[second] is
None) else d[second], d[first] + w)
wr = open('output.txt', 'w')
if s not in d.keys() or d[s] is None:
    wr.write('-1')
else:
    wr.write(str(d[s]))
wr.close()
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,015	350 Кб
2	Accepted	0,015	358 Кб
3	Accepted	0,031	362 Кб
4	Accepted	0,015	358 Кб
5	Accepted	0,031	362 Кб
6	Accepted	0,015	350 Кб
7	Accepted	0,015	362 Кб
8	Accepted	0,015	362 Кб
9	Accepted	0,015	362 Кб
10	Accepted	0,015	366 Кб
11	Accepted	0,015	362 Кб
12	Accepted	0,031	366 Кб
13	Accepted	0,031	358 Кб
14	Accepted	0,046	358 Кб
15	Accepted	0,031	346 Кб
16	Accepted	0,062	362 Кб
17	Accepted	0,031	358 Кб
18	Accepted	0,109	646 Кб
19	Accepted	0,015	362 Кб
20	Accepted	0,031	362 Кб
21	Accepted	0,031	358 Кб
22	Accepted	0,031	366 Кб
23	Accepted	0,015	354 Кб
24	Accepted	0,015	366 Кб
25	Accepted	0,031	362 Кб
26	Accepted	0,046	362 Кб
27	Accepted	0,031	366 Кб
28	Accepted	0,062	362 Кб
29	Accepted	0,125	646 Кб
30	Accepted	0,046	366 Кб
31	Accepted	0,046	646 Кб
32	Accepted	0,109	646 Кб
33	Accepted	0,031	638 Кб
34	Accepted	0,062	642 Кб
35	Accepted	0,031	630 Кб
36	Accepted	0,062	634 Кб
37	Accepted	0,031	634 Кб
38	Accepted	0,031	642 Кб

Вывод по задаче: реализовала алгоритм поиска кратчайшего пути в графе, тесты успешны.

Задача №12. Цветной лабиринт (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 $c_1 \dots c_k$. 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 цвета.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит два целых числа n ($1 \le n \le 10000$) и m ($1 \le m \le 100000$) - соответственно количество комнат и коридоров в лабиринте. Следующие m строк содержат описания коридоров. Каждое описание содержит три числа u ($1 \le u \le n$), v ($1 \le v \le n$), c ($1 \le c \le 100$) - соответственно номера комнат, соединенных этим коридором, и цвет коридора. Следующая, $(m+2)$-ая строка входного файла содержит длину описания пути - целое число k ($0 \le k \le 100000$). Последняя строка входного файла содержит k целых чисел, разделенных пробелами, - описание пути по лабиринту.
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите строку INCORRECT, если описание пути некорректно, иначе выведите номер комнаты, в которую ведет описанный путь. Помните, что путь начинается в комнате номер один.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.

Решение сводится к следующему: мы находимся в комнате, если из нее нет выхода через цветной лабиринт, то мы завершаем алгоритм, в противном случае мы следуем по карте, пока не достигнем финиша или пока не зайдем в тупик.

```
file = open('input.txt')
n, m = map(int, file.readline().split())
arr = [{} for _ in range(n)]
for _ in range(m):
    f, s, c = map(int, file.readline().split())
    arr[f - 1][c - 1] = s - 1
    arr[s - 1][c - 1] = f - 1
k = int(file.readline())
path = list(map(int, file.readline().split()))
file.close()
p = 0
f = False
for i in range(k):
    c = path[i] - 1
    if c not in arr[p].keys():
```

```
            f = True
            break
        p2 = arr[p][c]
        if p2 is not None:
            p = p2
        else:
            f = True
            break
w = open('output.txt', 'w')
if f:
    w.write('INCORRECT')
else:
    w.write(str(p + 1))
w.close()
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,031	354 Кб
2	Accepted	0,015	366 Кб
3	Accepted	0,078	1854 Кб
4	Accepted	0,093	2990 Кб
5	Accepted	0,046	2094 Кб
6	Accepted	0,031	1286 Кб
7	Accepted	0,046	1818 Кб
8	Accepted	0,062	9 Мб
9	Accepted	0,031	1578 Кб
10	Accepted	0,078	3542 Кб
11	Accepted	0,031	2234 Кб
12	Accepted	0,062	11 Мб
13	Accepted	0,109	4910 Кб
14	Accepted	0,031	6,8 Мб
15	Accepted	0,093	4982 Кб
16	Accepted	0,062	6,7 Мб
17	Accepted	0,046	4862 Кб
18	Accepted	0,093	8,3 Мб
19	Accepted	0,062	3966 Кб
20	Accepted	0,093	4838 Кб
21	Accepted	0,078	11 Мб
22	Accepted	0,093	7,5 Мб

Вывод по задаче: веселая, интересная!

Задача №14. Автобусы (3 балла)

Между некоторыми деревнями края Власюки ходят автобусы. Поскольку пассажиропотоки здесь не очень большие, то автобусы ходят всего несколько раз в день.

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

- **Формат входных данных (input.txt) и ограничения.** Во входном файле INPUT.TXT записано число N - общее число деревень ($1 \leq N \leq 100$), номера деревень d и v, затем количество автобусных рейсов R ($0 \leq R \leq 10000$). Затем идут описания автобусных рейсов. Каждый рейс задается номером деревни отправления, временем отправления, деревней назначения и временем прибытия (все времена - целые от 0 до 10000). Если в момент t пассажир приезжает в деревню, то уехать из нее он может в любой момент времени, начиная с t.
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT вывести минимальное время, когда Мария Ивановна может оказаться в деревне v. Если она не сможет с помощью указанных автобусных рейсов добраться из d в v, вывести -1.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.

В этом задании необходимо было найти минимальное время, за которое вы можете прибыть из заданной деревни в заданную. Сначала я подсчитала все значения, затем, используя цикл while, я искала минимальное затраченное время.

```
f = open("input.txt")
cn = f.readline().split()
start, finish = map(int, f.readline().split())
n = f.readline().split()
graph = dict()
for i in range(int(cn[0])):
    graph[i + 1] = list()
for i in range(int(n[0])):
    a, b, c, d = map(int, f.readline().split())
    graph[a].append((c, b, d))

time = dict()
ready = dict()
time[start] = 0

while len(time):
    t = min(time, key=lambda x: time[x])
    ready[t] = time[t]
    for i in graph[t]:
        if i[0] not in ready and i[1] >= time[t]:
            if i[0] not in time:
                time[i[0]] = i[2]
            else:
                time[i[0]] = min(time[i[0]], i[2])
```

```
    time.pop(t)

if finish in ready:
    w = open('output.txt', 'w')
    w.write(str(ready[finish]))
else:
    w = open('output.txt', 'w')
    w.write('-1')
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,015	362 Кб
2	Accepted	0,031	350 Кб
3	Accepted	0,031	362 Кб
4	Accepted	0,015	350 Кб
5	Accepted	0,015	362 Кб
6	Accepted	0,015	366 Кб
7	Accepted	0,031	370 Кб
8	Accepted	0,031	378 Кб
9	Accepted	0,062	1558 Кб
10	Accepted	0,062	1062 Кб

Вывод по задаче: автобусы едут, задание решено.

Задача №15. Герои (3 балла)

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 наиболее 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.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит целые числа N и M ($1 \le N, M \le 20$) – размеры сада. Далее идут N строк по M символов в каждом; символы '0' соответствуют участкам, на которых нет засады, символы '1' – участкам, на которых разместились гвардейцы. В $N+2$-ой строке теста записано три целых числа: координаты участка, на котором королева будет ждать мушкетёров (Q_x, Q_y) ($1 < Q_x < N$, $1 < Q_y < M$) и время в минутах до начала балета ($1 \le L \le 1000$). В $N+3$-ей строке записаны через пробел целые числа координаты участка, с которого стартует Атос (A_x, A_y) ($1 < A_x < N$, $1 < A_y < M$) и количество подвесок, хранящихся у него ($1 \le P_a \le 1000$). В $N+4$, $N+5$ и $N+6$-ой строках аналогично записаны стартовые координаты и количество подвесок у Портоса, Арамиса и д'Артаньяна.
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите количество подвесок, которое королева успеет получить у мушкетеров до начала балета.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.

Происходит считывание и преобразование входных данных. Переменная field хранит информацию о карте, где 1 - есть гвардейцы, 0 - пустой сегмент сада. Переменные q_x и q_y - координаты королевы от верхнего левого угла, который равен (0; 0). L - ограничение по времени. В 12-й строке кода уменьшаем полученные координаты на 1, так как индексы в field с 0, а не с 1, как написано в задании. Переменная length хранит двумерный список (length[x][y]), где по координатам x и y хранится время, необходимое королеве пройти до них (просто в алгоритме поиска в ширину мы берем начало от королевы). Далее инициализируется очередь queue с координатами королевы. Далее используется алгоритм поиска в ширину, итерируя очередь, пока она не закончится. В цикле достаем координаты x и y. Далее, по условию задачи, мушкетеры могут перейти только на четыре соседние клетки, которые мы исследуем. При исследовании, проверяем, чтобы мушкетеры могли попадать только на правую, левую, верхнюю или нижнюю клетки через проверку $(dx)^2 + (dy)^2 = 1$. Находим новые координаты. Если новые координаты нас устраивают (еще не были посещены - значение inf и на карте не стоит там гвардеец), то тогда

записываем туда значение и добавляем в queue новую исследуемую координату. Потом происходит подсчет и вывод результата, где мы проходимся по каждому мушкетеру и сверяемся по времени с L. Если удовлетворяет, то прибавляем количество подвесок к результирующему значению.

```
n, m = map(int, input().split())
field = []
for i in range(n):
    field.append(list(input().strip()))
# координаты x и y королевы вверху слева (x:0; y:0)
q_x, q_y, L = map(int, input().split())
q_x -= 1
q_y -= 1

length = [[float("+inf") for _ in range(m)] for _ in
range(n)]
# королева имеет 0, потому что это начальная и
конечная точки
length[q_x][q_y] = 0
queue = [(q_x, q_y)]
# BFS
while queue:
    # current_pos[0] - это координаты x,
current_pos[1] - координаты y
    current_x, current_y = queue.pop(0)
    # прохождение через 4 соседних элемента
    for dx in range(-1, 2):
        for dy in range(-1, 2):
            if (dx ** 2 + dy ** 2) == 1:
                new_x = current_x + dx
                new_y = current_y + dy
                if length[new_x][new_y] ==
float('+inf') and field[new_x][new_y] == '0':
                    length[new_x][new_y] =
length[current_x][current_y] + 1
                    queue.append((new_x, new_y))
result = 0
for _ in range(4):
    m_x, m_y, suspensions = map(int, input().split())
    m_x -= 1
    m_y -= 1
```

```
    if length[m_x][m_y] <= L:
        result += suspensions
print(result)
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,031	362 Кб
2	Accepted	0,031	366 Кб
3	Accepted	0,031	366 Кб
4	Accepted	0,046	366 Кб
5	Accepted	0,031	354 Кб
6	Accepted	0,015	358 Кб
7	Accepted	0,015	362 Кб
8	Accepted	0,031	354 Кб
9	Accepted	0,015	358 Кб
10	Accepted	0,015	346 Кб
11	Accepted	0,046	358 Кб
12	Accepted	0,015	362 Кб
13	Accepted	0,015	358 Кб
14	Accepted	0,031	362 Кб
15	Accepted	0,031	362 Кб
16	Accepted	0,015	358 Кб
17	Accepted	0,031	366 Кб
18	Accepted	0,015	362 Кб
19	Accepted	0,015	358 Кб
20	Accepted	0,015	358 Кб
21	Accepted	0,031	358 Кб
22	Accepted	0,046	362 Кб
23	Accepted	0,031	366 Кб
24	Accepted	0,015	358 Кб
25	Accepted	0,031	366 Кб
26	Accepted	0,015	366 Кб
27	Accepted	0,015	366 Кб
28	Accepted	0,015	350 Кб
29	Accepted	0,015	366 Кб
30	Accepted	0,031	366 Кб

Вывод по задаче: тесты успешны, задача сложная.

Задача №16. Рекурсия (3 балла)

Одним из важных понятий, используемых в теории алгоритмов, является рекурсия. Неформально ее можно определить как использование в описании объекта самого себя. Если речь идет о процедуре, то в процессе исполнении эта процедура напрямую или косвенно (через другие процедуры) вызывает сама себя.

Рекурсия является очень «мощным» методом построения алгоритмов, но таит в себе некоторые опасности. Например, неаккуратно написанная рекурсивная процедура может войти в бесконечную рекурсию, то есть, никогда не закончить свое выполнение (на самом деле, выполнение закончится с переполнением стека).

Поскольку рекурсия может быть косвенной (процедура вызывает сама себя через другие процедуры), то задача определения того факта, является ли данная процедура рекурсивной, достаточно сложна. Попробуем решить более простую задачу.

Рассмотрим программу, состоящую из n процедур $P_1, P_2, \ldots, P_n$. Пусть для каждой процедуры известны процедуры, которые она может вызывать. Процедура P называется потенциально рекурсивной, если существует такая последовательность процедур $Q_0, Q_1, \ldots, Q_k$, что $Q_0 = Q_k = P$ и для $i = 1...k$ 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.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит целое число n – количество процедур в программе ($1 \le n \le 100$). Далее следуют n блоков, описывающих процедуры. После каждого блока следует строка, которая содержит 5 символов «*».

  Описание процедуры начинается со строки, содержащий ее идентификатор, состоящий только из маленьких букв английского алфавита и цифр. Идентификатор непуст, и его длина не превосходит 100 символов. Далее идет строка, содержащая число k ($k \le n$) – количество процедур, которые могут быть вызваны описываемой процедурой. Последующие k строк содержат идентификаторы этих процедур – по одному идентификатору на строке.

  Различные процедуры имеют различные идентификаторы. При этом ни одна процедура не может вызвать процедуру, которая не описана во входном файле.

- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT для каждой процедуры, присутствующей во входных данных, необходимо вывести слово YES, если она является потенциально рекурсивной, и слово NO – в противном случае, в том же порядке, в каком они перечислены во входных данных.

- Ограничение по времени. 1 сек.

- Ограничение по памяти. 16 мб.

Проверяем, является ли программа рекурсивной, если она вызывает предыдущие, проверяем его на цикл методом «покраски». Нашли цикл – завершение программы. выводим ответ.

```
from collections import deque


def is_recursive(graph, procedure):
    queue = deque()
    visited = set()
    inqueue = set()
    queue.append(procedure)
    inqueue.add(procedure)
    while len(queue) > 0:
        c = queue.popleft()
        inqueue.remove(c)
        visited.add(c)
```

```
        for v in graph[c]:
            if v == procedure:
                return True
            if v not in visited and v not in inqueue:
                queue.append(v)
                inqueue.add(v)
    return False


f = open('input.txt')
N = int(f.readline())
graph, res, pcs = {}, [], []
for _ in range(N):
    procedure = f.readline().strip()
    pcs.append(procedure)
    n = int(f.readline())
    calls = [f.readline().strip() for _ in range(n)]
    graph[procedure] = calls
    f.readline()
f.close()
for procedure in pcs:
    res.append("YES" if is_recursive(graph,
procedure) else "NO")
w = open('output.txt', 'w')
w.write("\n".join(res))
w.close()
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,046	802 Кб
2	Accepted	0,062	1866 Кб
3	Accepted	0,062	2126 Кб
4	Accepted	0,046	806 Кб
5	Accepted	0,031	802 Кб
6	Accepted	0,031	802 Кб
7	Accepted	0,046	806 Кб
8	Accepted	0,015	806 Кб
9	Accepted	0,015	798 Кб
10	Accepted	0,015	806 Кб
11	Accepted	0,031	806 Кб
12	Accepted	0,015	802 Кб
13	Accepted	0,031	802 Кб
14	Accepted	0,031	806 Кб
15	Accepted	0,015	802 Кб
16	Accepted	0,062	810 Кб
17	Accepted	0,015	806 Кб
18	Accepted	0,031	802 Кб
19	Accepted	0,031	1318 Кб
20	Accepted	0,062	2430 Кб

Вывод по задаче: снова работа с графами, удобно.

Задача №17. Слабая K-связность (4 балла)

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.

- **Формат входных данных (input.txt) и ограничения.** В первой строке входного файла INPUT.TXT записаны два числа $2 \leq N \leq 300$ и $1 \leq M \leq 10^5$ - количество городов и дорог в плане. В последующих M строках даны по два числа - номера городов, в которых начинается и заканчивается соответствующая дорога.
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите минимальное K, такое, что данный во входном файле план является слабо K-связным.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 16 мб.

Чтобы вычислить минимальное значение K, мы сделаем так: если во входных данных указана дорога из города A в B, то мы помещаем 0 в двумерный список (поскольку мы не нарушаем правила); если не была указана дорога из B в A, то мы ставим 1. Мы используем алгоритм Флойда-Уоршелла, который находит длины кратчайших путей между всеми парами вершин во взвешенном графе. Мы формируем результат, в котором ищем максимум (переменная D) для количества нарушений, за которые можно проехать из одного города в другой. Затем мы выводим результат K.

```
n, m = map(int, input().split())
D = [[float('+inf') for _ in range(n)] for _ in
range(n)]
# cost of A to A = 0
for i in range(n):
    D[i][i] = 0
for _ in range(m):
    a, b = map(int, input().split())
    a -= 1
    b -= 1
    D[a][b] = 0
    D[b][a] = min(D[b][a], 1)
# алгоритм Флойда
for k in range(n):
    for i in range(n):
        if D[i][k] != float('+inf'):
            for j in range(n):
```

```
                D[i][j] = min(D[i][j], D[i][k] +
D[k][j])
result = 0
for i in range(n):
    for j in range(n):
        result = max(result, D[i][j])
print(result)
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,031	362 Кб
2	Accepted	0,015	362 Кб
3	Accepted	0,031	362 Кб
4	Accepted	0,015	366 Кб
5	Accepted	0,015	370 Кб
6	Accepted	0,015	362 Кб
7	Accepted	0,015	370 Кб
8	Accepted	0,015	362 Кб
9	Accepted	0,031	362 Кб
10	Accepted	0,015	370 Кб
11	Accepted	0,015	362 Кб
12	Accepted	0,031	366 Кб
13	Accepted	0,031	358 Кб
14	Accepted	0,031	370 Кб
15	Accepted	0,015	366 Кб
16	Accepted	0,015	366 Кб
17	Accepted	0,015	366 Кб
18	Accepted	0,015	366 Кб
19	Accepted	0,031	362 Кб
20	Accepted	0,015	366 Кб
21	Accepted	0,015	358 Кб
22	Accepted	0,015	362 Кб
23	Accepted	0,015	362 Кб
24	Accepted	0,031	366 Кб
25	Accepted	0,015	362 Кб
26	Accepted	0,031	366 Кб
27	Accepted	0,031	354 Кб
28	Accepted	0,625	682 Кб
29	Accepted	0,484	694 Кб
30	Accepted	0,718	710 Кб
31	Accepted	0,468	694 Кб
32	Time limit exceeded	1,234	4162 Кб

Вывод по задаче: не уверена, что лимит этой задачи 1 секунда…простите…

Вывод

Красивая лабораторная работа с интересными заданиями на реализацию граф, хотя и было непросто.