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 21

Выполнила:
Савченко А.С.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

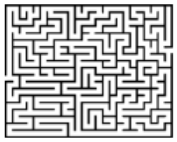
Задачи по варианту

Задача №1. Лабиринт [1 баллов]

Текст задачи:

1 Задача. Лабиринт [5 s, 512 Mb, 1 балл]

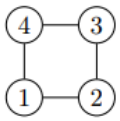
Лабиринт представляет собой прямоугольную сетку ячеек со стенками между некоторыми соседними ячейками. Вы хотите проверить, существует ли путь от данной ячейки к данному выходу из лабиринта, где выходом также является ячейка, лежащая на границе лабиринта (в примере, показанном на рисунке, есть два выхода: один на левой границе и один на правой границе). Для этого вы представляете лабиринт в виде неориентированного графа: вершины графа являются ячейками лабиринта, две вершины соединены неориентированным ребром, если они смежные и между ними нет стены. Тогда, чтобы проверить, существует ли путь между двумя заданными ячейками лабиринта, достаточно проверить, что существует путь между соответствующими двумя вершинами в графе.



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

- **Формат ввода / входного файла (input.txt).** Неориентированный граф с n вершинами и m ребрами по формату 1. Следующая строка после ввода всго графа содержит две вершины u и v.
- **Ограничения на входные данные.** $2 \le n \le 10^3, 1 \le m \le 10^3, 1 \le u, v \le n, u \ne v$.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если есть путь между вершинами u и v; выведите 0, если пути нет.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

input	output
4 4 1 2 3 2 4 3 1 4 1 4	1



Листинг кода. (именно листинг, а не скрины)

```
def dfs(graph, u, v, visited):
    if u == v:
        return True
    visited[u] = True
    for node in range(len(graph)):
```

```
        if graph[u][node] and not visited[node]:
            if dfs(graph, node, v, visited):
                return True
    return False


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 = [[False for _ in range(n)] for _ in range(n)]

    for _ in range(m):
        u, v = map(int, input_file.readline().split())
        graph[u - 1][v - 1] = True
        graph[v - 1][u - 1] = True

    visited = [False for _ in range(n)]

    output_file.write('1' if dfs(graph, u - 1, v - 1, visited) else '0')
```

Текстовое объяснение решения.

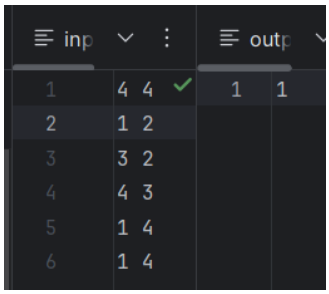
Читаем из файла кол-во вершин и кол-во ребер. Инициализируем граф в виде матрицы смежности заполненной False. В цикле по кол-ву ребер(длинной) заполняем матрицу смежности, так как граф не ориентированной заполняем как (v, u), так и (u, v).

Инициализируем список посещенных вершин значениями False. Если функция обхода возвращает True, печатаем 1 (значит есть путь между вершинами u и v) , иначе печатаем в файл 0.

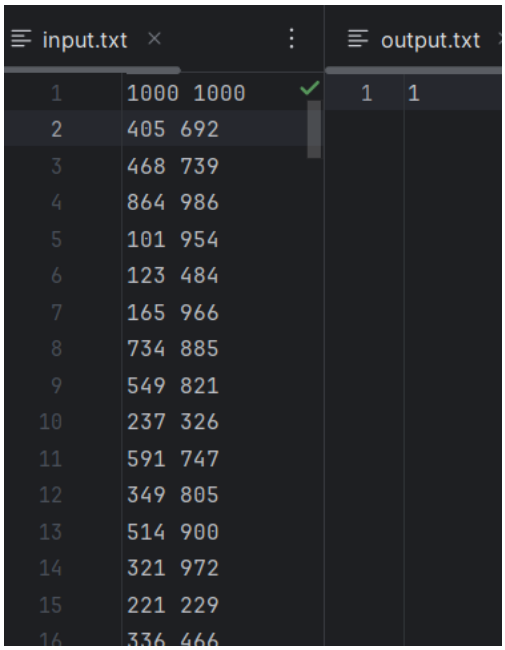
В функции обхода в глубину: Проверяем, если узел ,из которого мы начинаем путь, совпадает с тем, в который нам нужно добраться, возвращаем True. В остальных случаях отмечаем (узел) вершину u как True т.е. посещенную, и дальше в цикле по всем узлам проверяем, если существует

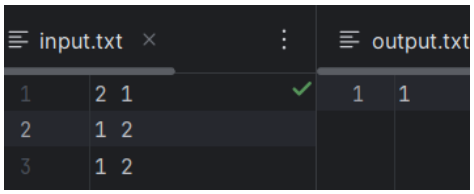
ребро, но его нет в списке посещенных рекурсивно вызываем функцию dfs, куда передаем переменную итератор в качестве u. Проверяем, если рекурсивная функция возвращает тру,путь найден возвращаем тру. Если за все итерации путь не был, найден возвр. false.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





Проверка задачи на (acmp и тд при наличии в задаче). (скрин)

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.00046170002315	14.75390625

входных данных из текста задачи		
Пример из задачи	0.0004351000534370	14.66796875
Верхняя граница диапазона значений входных данных из текста задачи	0.0355778000084	15.63671875

Вывод по задаче:

Задачка очень простая на первую отработку графов. Для реализации алгоритма для ее решения я использовала матрицу смежности, в обосновании решения я постарался подробно описать обход в глубин в первый раз, чтобы в следующих задачах не заострять на этом внимание.

Задача №7. Двудольный граф [1.5 баллов]

Текст задачи:

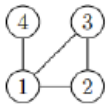
7 Задача. Двудольный граф [10 s, 512 Mb, 1.5 балла]

Неориентированный граф называется **двудольным**, если его вершины можно разбить на две части так, что каждое ребро графа соединяет вершины из разных частей, то есть не существует рёбер между вершинами одной и той же части графа. Двудольные графы естественным образом возникают в задачах, где граф используется для моделирования связей между объектами двух разных типов (например, мальчиками и девочками, или студентами и общежитиями). Альтернативное определение таково: граф двудольный, если его вершины можно раскрасить двумя цветами (например, черным и белым) так, что концы каждого ребра окрашены в разные цвета.

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

- **Формат ввода / входного файла (input.txt).** Неориентированный граф задан по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^5, 0 \le m \le 10^5$.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если граф двудольный; и 0 в противном случае.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

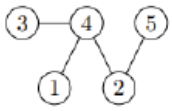
input	output
4 4 1 2 4 1 2 3 3 1	0



Этот граф не является двудольным. Чтобы убедиться в этом, предположим, что вершина 1 окрашена в белый цвет. Тогда вершины 2 и 3 нужно покрасить в черный цвет, так как граф содержит ребра 1, 2 и 1, 3. Но тогда ребро 2, 3 имеет оба конца одного цвета.

- Пример 2:

input	output
5 4 5 2 4 2 3 4 1 4	1



Этот граф двудольный: вершины 4 и 5 покрасим в белый цвет, все остальные вершины – в черный цвет.

- Что делать? Адаптируйте поиск в ширину (BFS), чтобы решить эту проблему.

Листинг кода. (именно листинг, а не скрины)

```
from collections import deque


def bipartite_graph(graph, n):
    colors = [-1] * n

    for start in range(n):
        if colors[start] == -1:
            queue = deque([start])
```

```
            colors[start] = 0

            while queue:
                node = queue.popleft()
                for i in graph[node]:
                    if colors[i] == -1:
                        colors[i] = 1 - colors[node]
                        queue.append(i)
                    elif colors[i] == colors[node]:
                        return False
    return True


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(n)}

    for _ in range(m):
        u, v = map(int, input_file.readline().split())
        graph[u - 1].append(v - 1)
        graph[v - 1].append(u - 1)

        output_file.write(str(int(bipartite_graph(graph,
n))))
```

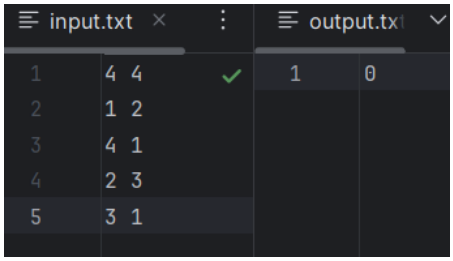
Текстовое объяснение решения.

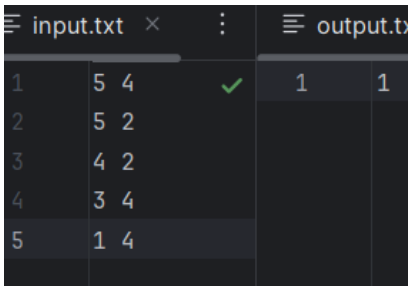
Считываем неориентированный граф.

Инициализируем список цветов значениями -1, что значит вершины не окрашены. В цикле идем по всем вершинам графа, если вершина не окрашена(т.е. =-1), начинаем обход в ширину от этой вершины: создаем очередь поместив туда эту вершину(в очереди будем хранить вершины которые собираемся посетить). Окрашиваем стартовую вершину в цвет

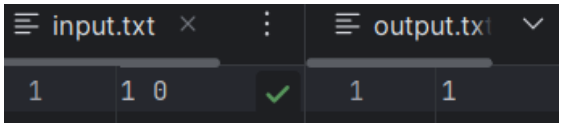
0(условно белый). В цикле пока очередь не пуста, будем проходить по соседним вершинам, если соседняя вершина еще не окрашена, окрасим ее в противоположный цвет, добавим ее в очередь в конец.
Если же у текущей и соседней вершины одинаковый цвет - граф не двудольный -> возвращаем false.
Если удалось окрасить все вершины - граф двудольный -> возвращаем true.
Для вывода 1 или 0 интуем(int) булевое значение которое вернула функция.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)





Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004452000139	14.8515625
Пример из задачи	0.00046540005	14.796875
Пример из задачи	0.0004441000055	14.89453125
Верхняя граница диапазона значений входных данных из	0.10623709997162	17.5625

текста задачи		

Вывод по задаче:

Наконец-то от обхода в глубину перешли к обходу в ширину) Мне понравилась сама идея определения двудольности графа по цвету вершин. Задачу удалось решить корректно для всех тестов.

Задача №13. Грядки[3 баллов]

Текст задачи:

13 Задача. 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.

- **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.
- Примеры:

input.txt	output.txt
5 10 ##.....#. .#..#...#. .###....#. ..##....#. ....#.	3

input.txt	output.txt
5 10 ##..#####. .#.#.#.... ###..##.#. ..##.....# .###.#####	5

- Проверяем **обязательно** – на acmp.

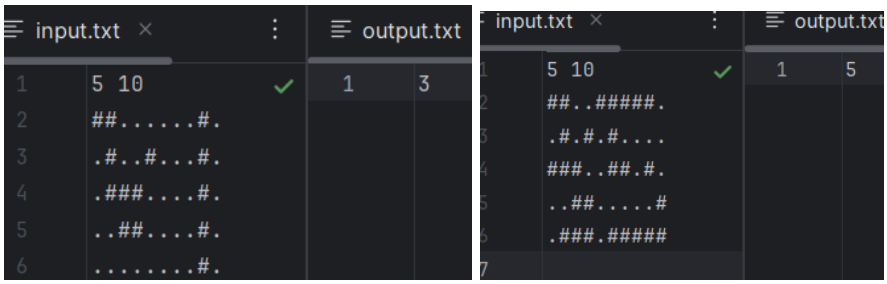
Листинг кода. (именно листинг, а не скрины)

Текстовое объяснение решения.

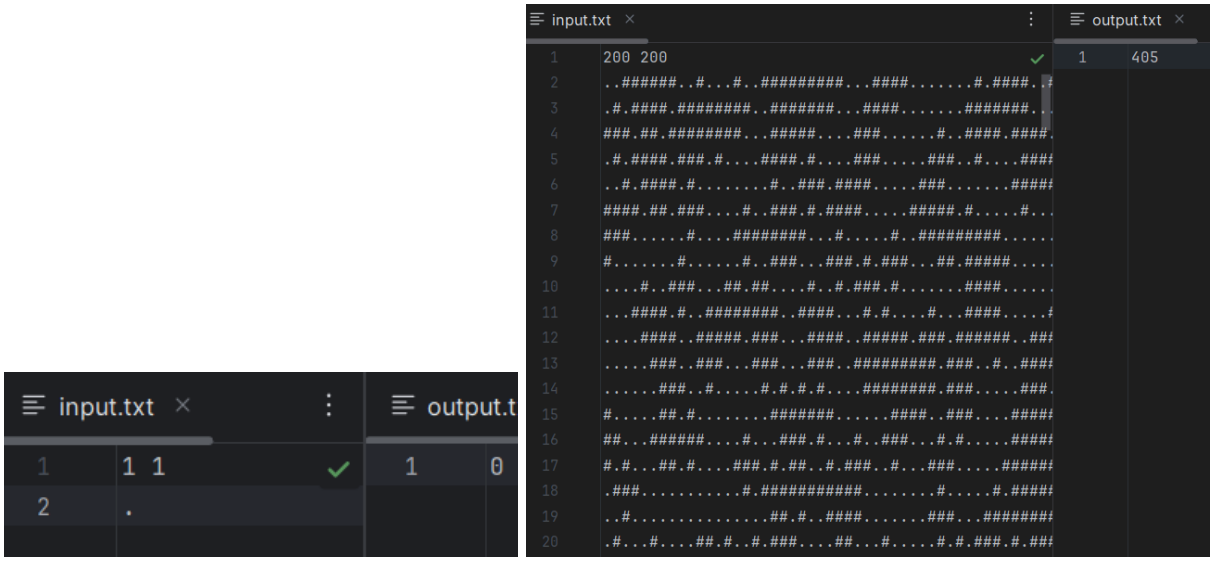
Считываем сад из решеток и точек как матрицу по строкам.

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 зависмоти от расположения слева, сверху,справа, снизу).

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)



Проверка задачи на (acmp и тд при наличии в задаче). (скрин)

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
21873556	08.09.2024 15:22:45	Савченко Анастасия Сергеевна	0432	Python	Accepted		0.109	3626 Кб

	Время выполнения	Затраты памяти, Мб

Нижняя граница диапазона значений входных данных из текста задачи	0.00048110005	14.4179687
Пример из задачи	0.00050540000665	14.460937
Пример из задачи	0.000464200042	14.45703125
Верхняя граница диапазона значений входных данных из текста задачи	0.012188800	14.8671875

Вывод по задаче:

Задача относительно простая, но с интересной идей. Я также не сразу поняла пример, но когда разобралась задача стала понятна. Начала я решать ее через подсчет компонент связности, но на больших входных начали появляться проблемы с рекурсией, так что я решила ее через обход в ширину, такое решение вполне подходит, так как нам нужны не сами грядки, а только их кол-во.

Дополнительные задачи

Задача №2. Компоненты [1 баллов]

Текст задачи.

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.

- **Формат ввода / входного файла (input.txt).** Неориентированный граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^3$, $0 \le m \le 10^3$.
- **Формат вывода / выходного файла (output.txt).** Выведите количество компонент связности.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input	output
4 2 1 2 3 2	2

В этом графе есть два компонента связности: 1, 2, 3 и 4.

Листинг кода. (именно листинг, а не скрины)

```
def dfs(graph, u, visited):
    visited[u] = True
    for node in graph[u]:
        if not visited[node]:
            dfs(graph, node, visited)


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 = [[] for _ in range(n)]

    for _ in range(m):
        u, v = map(int, input_file.readline().split())
```

```
        graph[u - 1].append(v - 1)
        graph[v - 1].append(u - 1)

    visited = [False for _ in range(n)]

    counter = 0

    for node in range(n):
        if not visited[node]:
            dfs(graph, node, visited)
            counter += 1

    output_file.write(str(counter))
```

Текстовое объяснение решения.

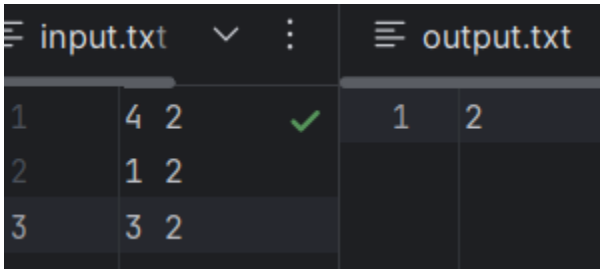
В функции обхода в глубину отмечаем текущую вершину как посещенную. Перебираем соседние вершины, если вершина не была посещена,то рекурсивно вызываем функцию.

читаем входные данные, создаем пустой список смежности. Заполняем список смежности для неориентированного графа. (в нем для каждой вершины будет сохранен список его соседей, индексация с нуля)

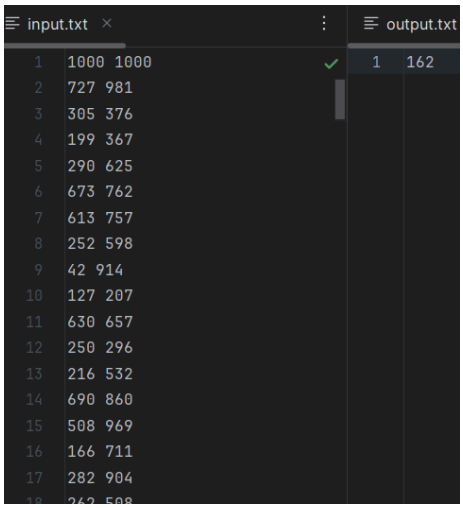
Инициализируем список посещенных вершин значениями False.

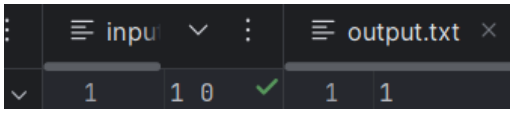
Заметим, если посетить один узел в компоненте, то можно посетить и все остальные узлы в этой компоненте. Перебираем все узлы в графе, если узел не посещен, вызываем функцию обхода для связной компоненты и увеличиваем счетчик на 1.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





	Время выполнения, с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0004588000010699034	14.83203125
Пример из задачи	0.00043929996	14.87109375
Верхняя граница диапазона значений входных данных из текста задачи	0.00154560001101344	15.19140625

Вывод по задаче:

Для этой задачи я использовала список смежности, для него в отличии от матрицы не придется перебирать всех возможных соседей, что делает код более эффективным. Так, на макс. значения время (с 0.0637288000 улучшилось до 0.00154560001101344 сек.)

Задача №3. Циклы [1 баллов]

Текст задачи.

3 Задача. 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.

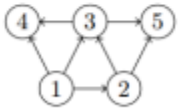
- **Формат ввода / входного файла (input.txt).** Ориентированный граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^3, 0 \le m \le 10^3$.
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если данный граф содержит цикл; выведите 0, если не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

input	output
4 4 1 2 4 1 2 3 3 1	1

Этот граф содержит цикл: $3 \to 1 \to 2 \to 3$.

- Пример 2:

input	output
5 7 1 2 2 3 1 3 3 4 1 4 2 5 3 5	0



В этом графе нет циклов. Это можно увидеть, например, отметив, что все ребра в этом графе идут от вершины с меньшим номером к вершине с большим номером.

Листинг кода. (именно листинг, а не скрины)

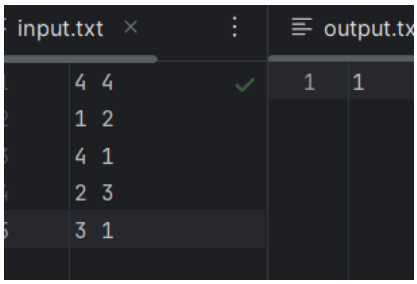
```
def dfs_cycle(graph, u, visited, visited_stack):
    visited[u] = True
    visited_stack[u] = True

    for node in graph[u]:
```

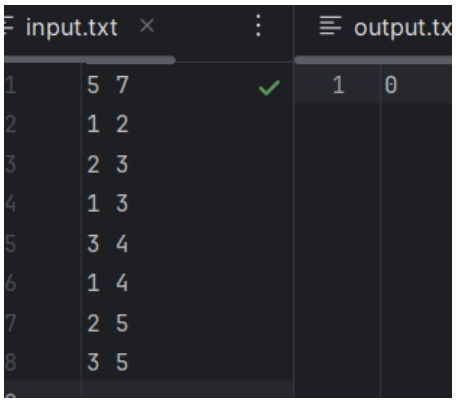
```
        if not visited[node]:
                    if  dfs_cycle(graph,  node,  visited,
visited_stack):
                return True
        elif visited_stack[node]:
            return True

    visited_stack[u] = False
    return False


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 = [[] for _ in range(n)]

    for _ in range(m):
        u, v = map(int, input_file.readline().split())
        graph[u - 1].append(v - 1)

    visited = [False for _ in range(n)]
    visited_stack = [False for _ in range(n)]

    for node in range(n):
         if not visited[node] and dfs_cycle(graph, node,
visited, visited_stack):
            output_file.write('1')
            break
    else:
        output_file.write('0')
```

Текстовое объяснение решения.

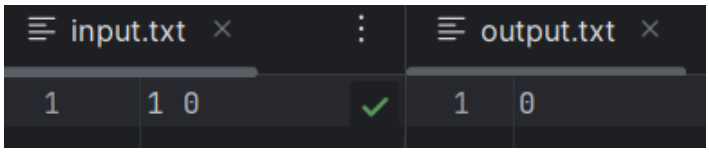
Читаем входные данные создаем пустой список смежности. Заполняем список смежности для ориентированного графа.

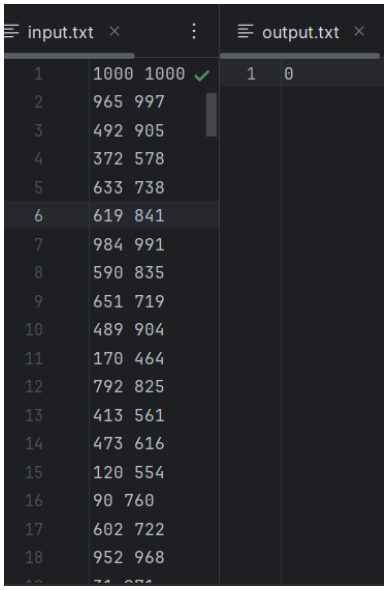
Инициализируем список посещенных вершин и список стек для поиска циклов.
В функции обхода в глубину отмечаем текущую вершину как посещенную и тоже с текущей вершиной в стеке. В цикле проходим по всем соседним вершинам. Если сосед не был посещен, рекурсивно вызываем функцию обхода для этой вершины. Если сосед находится в стеке, значит найден цикл - возвращаем true. Если был завершен проход, но цикл не найден, возвращаем false.
Если цикл обнаружен, записываем в выходной файл 1 иначе 0.
Результат работы кода на примерах из текста задачи:(скрины input output файлов)



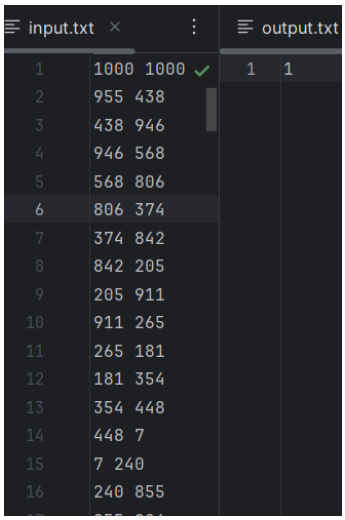


Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





(без цикла)



(с циклом)

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004021000349894	14.87109375
Пример из задачи	0.0004063999513164	14.8046875
Пример из задачи	0.000396999996155	14.86328125
Верхняя граница	0.0015979999443516	15.14453125

диапазона значений входных данных из текста задачи БЕЗ ЦИКЛА		
Верхняя граница диапазона значений входных данных из текста задачи С ЦИКЛом	0.00116450001951	15.140625

Вывод по задаче:

В этой задаче в отличии от предыдущей работа идет уже с ориентированными графами). Для этой задаче я снова использовала обход в глубину и список смежности, а для поиска цикла в графе использовала стек рекурсии.

Задача №4. Порядок курсов [1 баллов]

Текст задачи.

4 Задача. Порядок курсов [10 s, 512 Mb, 1 балл]

Теперь, когда вы уверены, что в данном учебном плане нет циклических зависимостей, вам нужно найти порядок всех курсов, соответствующий всем зависимостям. Для этого нужно сделать топологическую сортировку соответствующего ориентированного графа.

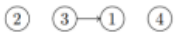
Дан ориентированный ациклический граф (DAG) с n вершинами и m ребрами. Выполните топологическую сортировку.

- **Формат ввода / входного файла (input.txt).** Ориентированный ациклический граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \leq n \leq 10^5$, $0 \leq m \leq 10^5$. Графы во входных файлах гарантированно ациклические.
- **Формат вывода / выходного файла (output.txt).** Выведите ***любое*** линейное упорядочение данного графа (Многие ациклические графы имеют более одного варианта упорядочения, вы можете вывести любой из них).
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

input	output
4 3 1 2 4 1 3 1	4 3 1 2

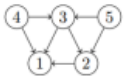

- Пример 2:

input	output
4 1 3 1	2 3 1 4

- Пример 3:

input	output
5 7 2 1 3 2 3 1 4 3 4 1 5 2 5 3	5 4 3 2 1




Листинг кода. (именно листинг, а не скрины)

```
def topological_sort(n, graph):
    visited = [False for _ in range(n)]
    topo_sort_s = []

    def dfs(node):
        visited[node] = True
        for i in range(n):
            if graph[node][i] and not visited[i]:
                dfs(i)
        topo_sort_s.append(node + 1)
```

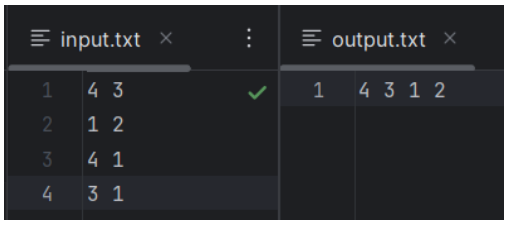
```
    for i in range(n):
        if not visited[i]:
            dfs(i)

    return topo_sort_s[::-1]


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 = [[False for _ in range(n)] for _ in range(n)]

    for _ in range(m):
        u, v = map(int, input_file.readline().split())
        graph[u - 1][v - 1] = True

    result = topological_sort(n, graph)
    output_file.write(" ".join(map(str, result)))
```

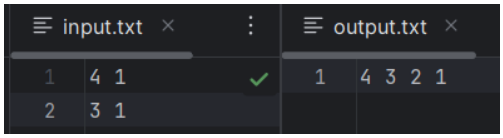
Текстовое объяснение решения.

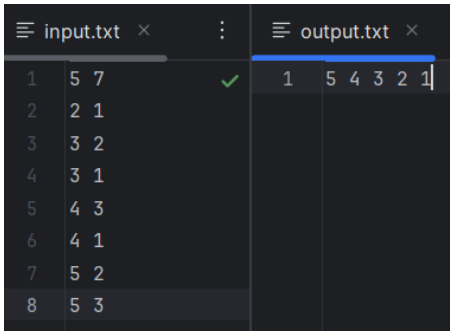
Считываем данные, заполняем ориентированный граф используя матрицу смежности.

Проходимся по непосещенным вершинам поиском в глубину и добавляем вершину в список после обхода ее соседей. Выводим массив топологической сортировки в обратном порядке.

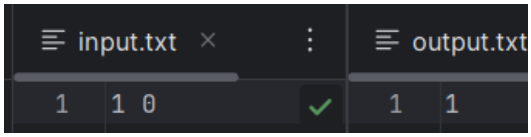
Результат работы кода на примерах из текста задачи:(скрины input output файлов)







Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)



	Время выполнения,с	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи	0.00043619994539	14.71875
Пример из задачи	0.000423700083047	14.640625
Пример из задачи	0.00048489996697	14.6328125
Пример из задачи	0.000467799	14.68359375
Верхняя граница диапазона значений входных данных из текста задачи	6.028809000039473	820.40625

Вывод по задаче:

Задача мне ооочень не понравилась.

Сначала я реализовала алгоритм классической топологической сортировки через обход в глубину, он хорошо работал но на тесте 2 показывал неправильный ответ, т.к. Не учитывал вершин без входящих ребер.Поэтому я начала использовать алгоритм Кана(с очередью).
Однако, что новый что прежний алгоритм не осиливает максимальные входные данные n=m=10**5, поэтому я провела тесты на n=m=10**4:
без алгоритма Кана на входных n=m=10**4:
(хотя что обход в глубину, что топологическая сортировка работают за линейное время)
Время работы: 6.028809000039473 секунд
Затраты памяти: 820.40625 Мб

с алгоритмов Кана n=m=10**4 (справляется лучше но я не совсем уверена в его корректности на малых тестов, хотя мб это иза того что топологической сортировки может быть несколько вариантов):
Время работы: 5.831121800001711 секунд
Затраты памяти: 18.89453125 Мб
Таким образом, можно сделать вывод что python не подходит для решения данной задачи, когда кол-во ребер и кол-во вершин больше 10**4.

Однако позже я заменима матрицу смежности на список и наконец решение получилось эффективным для n=m=10**5, но на малых тестах можно увидеть что все же алгорит иногда работает неправильно так, что это решение нельзя считать верным.

Вообщем в листинге я привела самый первый алгоритм хоть он и не эффективен, думаю с учетом того, что за эту задачу 1 балл, примерно такое решение подразумевалось.

Задача №8. Стоимость полета [1,5 баллов]

Текст задачи.

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).

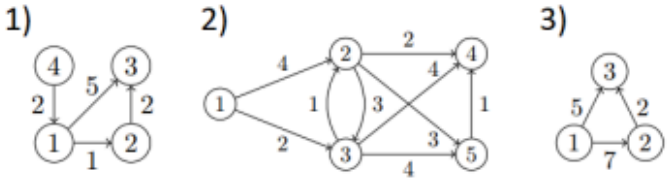
- **Формат ввода / входного файла (input.txt).** Ориентированный взвешенный граф задан по формату 1. Следующая строка содержит две вершины u и v.
- **Ограничения на входные данные.** $1 \le n \le 10^4$, $0 \le m \le 10^5$, $1 \le u, v \le n$, $u \ne v$, вес каждого ребра – неотрицательное целое число, не превосходящее 10^8.
- **Формат вывода / выходного файла (output.txt).** Выведите минимальный вес пути из u в v. Веведите -1, если пути нет.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Примеры:

input	output
4 4 1 2 1 4 1 2 2 3 2 1 3 5 1 3	3

input	output
5 9 1 2 4 1 3 2 2 3 2 3 2 1 2 4 2 3 5 4 5 4 1 2 5 3 3 4 4 1 5	6

input	output
3 3 1 2 7 1 3 5 2 3 2 3 2	-1

- Объяснения:



Пример 1 – В этом графе существует единственный кратчайший путь из вершины 1 в вершину 3 ($1 \to 2 \to 3$), и он имеет вес 3. Пример 2 – Есть два пути от 1 до 5 общего веса 6: $1 \to 3 \to 5$ и $1 \to 3 \to 2 \to 5$. Пример 3 – Нет пути от вершины 3 до 2.

Листинг кода. (именно листинг, а не скрины)

```
import heapq


def dijkstra(graph, start, end):
    dist = {i: float('inf') for i in graph}
    dist[start] = 0
    prior_queue = [(0, start)]
```

```
    while prior_queue:
        curr_dist, curr_v = heapq.heappop(prior_queue)

        if curr_dist > dist[curr_v]:
            continue

        for n, w in graph[curr_v].items():
            distance = curr_dist + w

            if distance < dist[n]:
                dist[n] = distance
                  heapq.heappush(prior_queue, (distance,
n))

    return dist[end] if dist[end] != float('inf') else
-1


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

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

                    start,      end      =      map(int,
file_input.readline().split())

    result = dijkstra(graph, start, end)
    file_output.write(str(result))
```

Текстовое объяснение решения.

Считаем граф в виде словаря, у которого ключи - вершины графа, значения - словари, у которых ключ - сосед, значение - соотв. вес ребра.

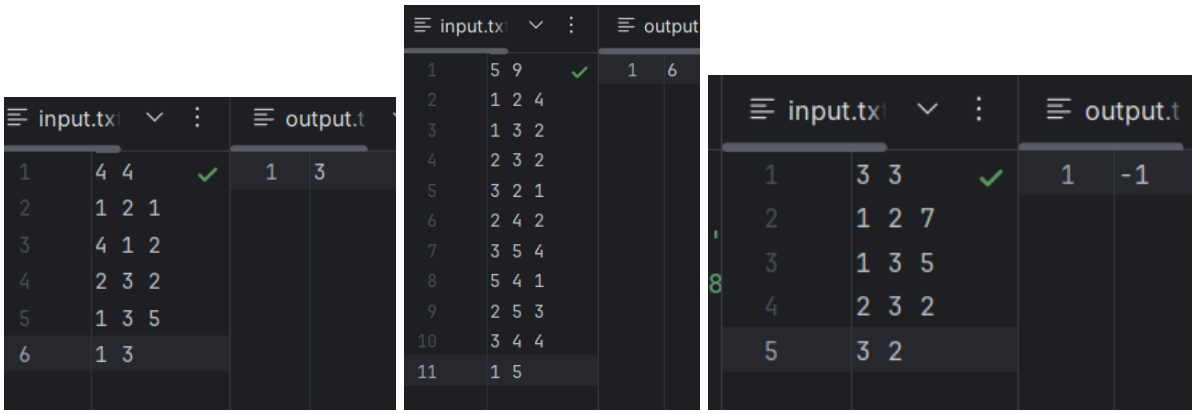
В функции Дейкстры создаем словарь dict для хранения расстояния от начальной вершины до других вершин, для начала заполним его значениями бесконечности. Расстояние от начальной вершины до самой себя = 0. Очередь с приоритетом будет хранить вершины графа в зависимости от расстояния до них от начальной вершины.

Пока очередь с приоритетом не пуста: достаем из нее вершину с наименьшим расстоянием, используя распаковку кортежа, присваиваем значения переменным curr_dist и curr_v. Проверяем, если текущее расстояние до вершины curr_v больше ранее известного, пропускаем вершину и идем дальше т.к. это не самый короткий путь.

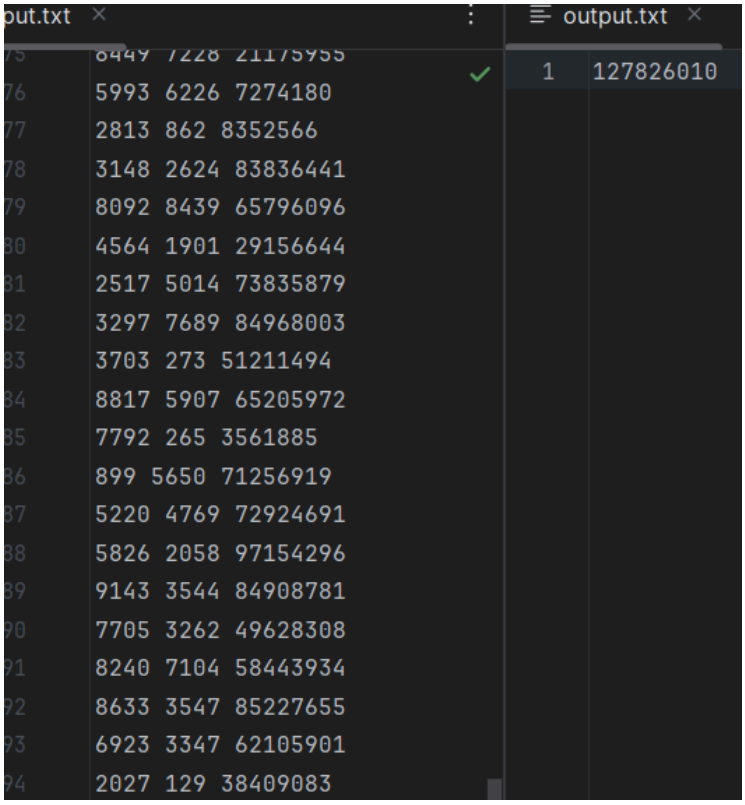
В цикле for идем по всем соседям и соотв. им весам, считаем расстояние, добавляя вес ребра к текущему расстоянию, если новое расстояние меньше ранее известного, то обновляем расстояние до этой вершины и добавляем соседа в очередь.

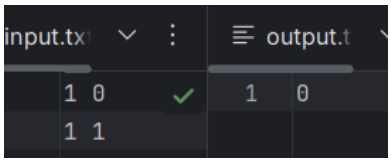
Возвращаем расстояние от начальной до конечной вершины, если его удалось найти(т.е. оно != бесконечности), иначе возвращаем -1.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.004146100021	14.73828125 Мб
Пример из задачи	0.004419700009748	14.76953125 Мб
Пример из задачи	0.0051978999981	14.828125 Мб
Пример из задачи	0.0011545000597	14.8125
Верхняя граница диапазона значений входных данных из	0.1125836999854	18.6875

текста задачи		

Вывод по задаче:

Для этой задачи я, реализовала алгоритм Дейкстры с приоритетной очередью(вроде она основана на бинарной куче) для нахождения кратчайших путей от одной вершины до всех остальных в ориентированном взвешенном графе(все ребра имеют неотрицательный вес). Алгоритм корректно работает на всех тестах.

Задача №9. Аномалии курсов валют [2 баллов]

Текст задачи.

9 Задача. Аномалии курсов валют [10 s, 512 Mb, 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 $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot \ldots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$.

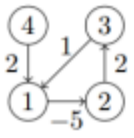
Для этого построим следующий граф: вершинами являются валюты $c_1, c_2, \ldots, c_n$, вес ребра из c_i в c_j равен $-\log r_{ij}$. Тогда достаточно проверить, есть ли в этом графе отрицательный цикл. Пусть цикл $c_i \to c_j \to c_k \to c_i$ имеет отрицательный вес. Это означает, что $-(\log c_{ij} +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

input	output
4 4 1 2 -5 4 1 2 2 3 2 3 1 1	1



Вес цикла $1 \to 2 \to 3$ равен -2, то есть отрицателен.

Листинг кода. (именно листинг, а не скрины)

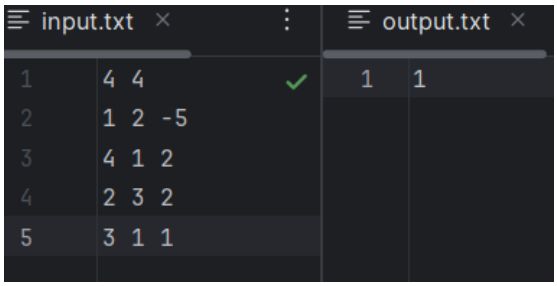
```
def is_cycle_negative_weight(graph, n):
    dist = [float('inf')] * n
    dist[0] = 0

    for _ in range(n - 1):
        for u, v, w in graph:
```

```
            if dist[u] < float('inf') and dist[v] >
dist[u] + w:
                dist[v] = dist[u] + w

    for u, v, w in graph:
        if dist[u] < float('inf') and dist[v] > dist[u]
+ w:
            return 1

    return 0


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 = []
    for _ in range(m):
                            u, v, w = map(int,
input_file.readline().split())
        graph.append((u - 1, v - 1, w))

output_file.write(str(is_cycle_negative_weight(graph,
n)))
```

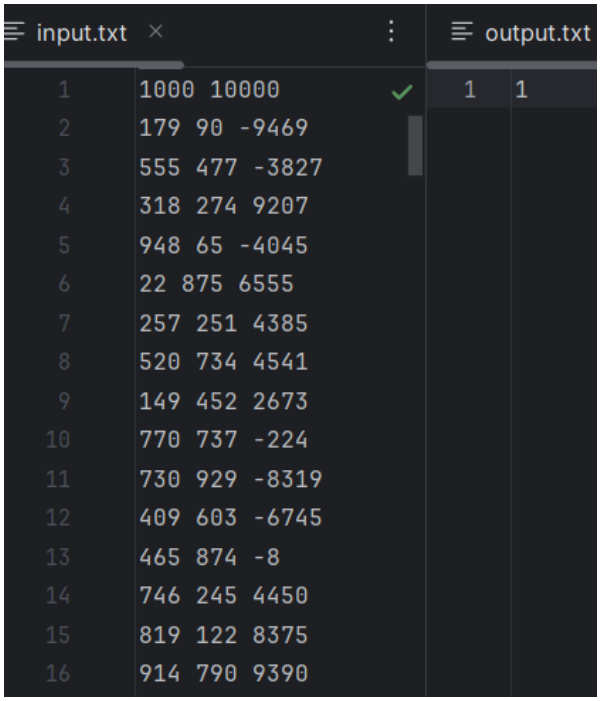
Текстовое объяснение решения.

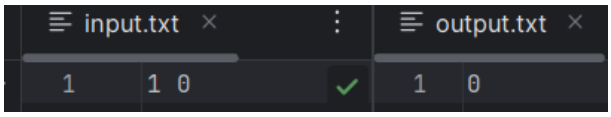
Нам дан ориентированный граф(в том числе и с отриц.. ребрами) Воспользуемся алгоритмом Беллмана-Форда, чтобы найти отриц. цикл в графе. Инициализируем список расстояний dist. Расстояние от начальной вершины до самой себя = 0.в первом цикле заполняем список кратчайших расстояний от начальной вершины до всех вершин в графе. В цикле(расслабляем ребра) смотрим для каждого ребра в графе можно ли найти путь короче, если да, то обновляем расстояние до узла v как расстояние до u + вес ребра w.

Во втором цикле будем искать отрицательный цикл. Если можно еще раз расслабить ребро, то значит есть отрицательный цикл, вывод 1, если цикл завершился и отрицательных ребер так и не было найдено возвращаем 0.
Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





	Время выполнения, сек	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из	0.0004894000012427	14.890625

текста задачи		
Пример из задачи	0.00044289999641	14.6953125
Верхняя граница диапазона значений входных данных из текста задачи	2.280299899983	16.42578125

Вывод по задаче:

Это очень важная задача на применение алгоритма Беллмана-Форда, для поиска цикла с отрицательным весом в ориентированным графе с возможными отрицательными ребрами. Задача мне понравилась)

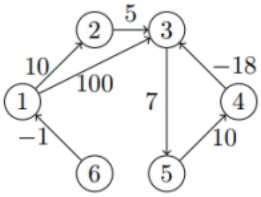
Задача №10. Оптимальный обмен валюты [2 баллов]

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.
- Пример 1:

input	output
6 7 1 2 10 2 3 5 1 3 100 3 5 7 5 4 10 4 3 -18 6 1 -1 1	0 10 - - - *



Первая строка вывода утверждает, что расстояние от вершины 1 до 1 равно 0. Вторая показывает, что расстояние от 1 до 2 равно 10 (соответствующий путь $1 \to 2$). Следующие три строки показывают, что расстояние от 1 до вершин 3, 4 и 5 равно $-\infty$: действительно, сначала можно достичь вершины 3 через ребра $1 \to 2 \to 3$, а затем сделать длину пути произвольно малой, совершая достаточно много обходов по циклу $3 \to 5 \to 4 \to 3$ с отрицательным весом. Последняя строка вывода показывает, что в этом графе нет пути от 1 до 6.

Листинг кода. (именно листинг, а не скрины)

```
def bellman_ford(graph, n, s):
    dist = [float('inf')] * n
    dist[s] = 0

    for _ in range(n - 1):
        for u, v, w in graph:
            if dist[u] < float('inf') and dist[v] >
dist[u] + w:
                dist[v] = dist[u] + w

    for _ in range(n - 1):
        for u, v, w in graph:
            if dist[u] < float('inf') and dist[v] >
dist[u] + w:
                dist[v] = float('-inf')

    return dist


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 = []
    for _ in range(m):
                            u, v, w = map(int,
input_file.readline().split())
        graph.append((u - 1, v - 1, w))

    s = int(input_file.readline().strip()) - 1

    dist = bellman_ford(graph, n, s)

    result = []
```

```
    for d in dist:
        if d == float('inf'):
            result.append("*")
        elif d == float('-inf'):
            result.append("-")
        else:
            result.append(str(d))
    output_file.write("\n".join(result))
```

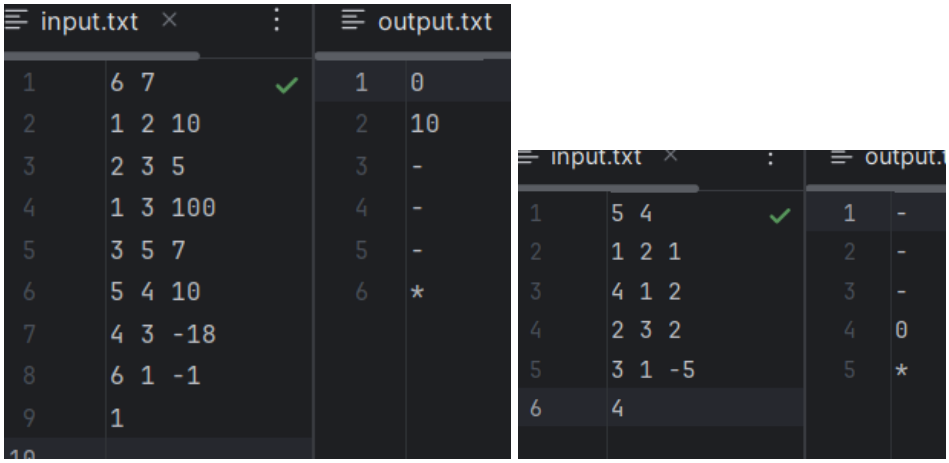
Текстовое объяснение решения.

Нам снова нужно найти кратчайшие пути из начальной вершины во все остальные в графе(могут быть отрицательные ребра) следовательно исп. алгоритм Беллмана-Форда.

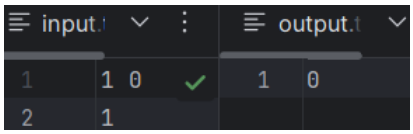
Код почти такой же как в предыдущей задаче. В первом цикле релаксируем ребра. Во втором цикле ищем отрицательный цикл, если расстояние до вершины можно сократить обновляем его до минус бесконечности(т.к. его можно уменьшать сколько угодно из-за отрицательного цикла). Возвращаем массив расстояний.

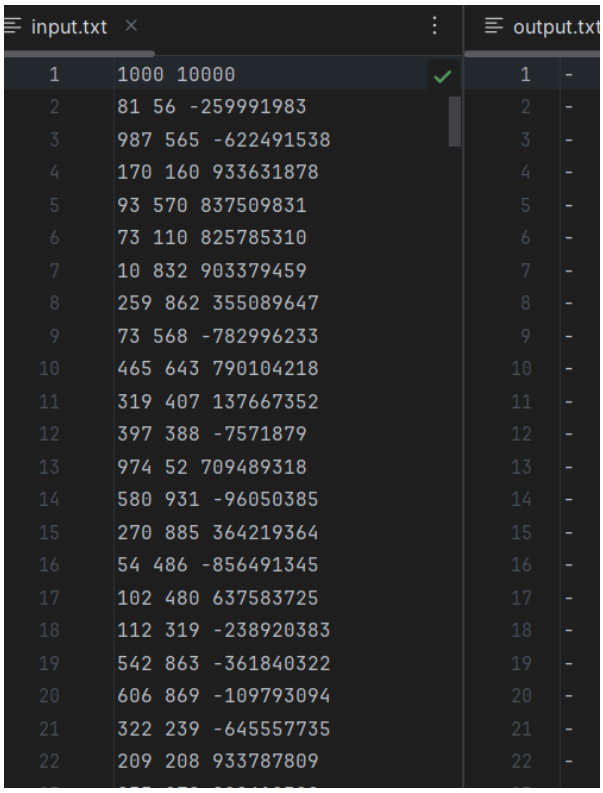
Для корректного вывода: если расстояние = бесконечности, значит пути нет выводим *, если путь есть и он может быть сколь угодно малым(т.е. расстояние = - беск.) возвр. -. В остальных случаях возвращаем длину пути, ну и как всегда путь от начальной вершины до самой себя = 0.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





	Время выполнения, сек	Затраты памяти, Мб
Нижняя граница диапазона значений входных данных из текста задачи	0.000453799962997	14.90234375
Пример из задачи	0.0004695998504757	14.83984375
Пример из задачи	0.000468700192	14.78125
Верхняя граница диапазона значений входных данных из текста задачи	4.267980800010264	16.33984375

Вывод по задаче:

Задача строится на алгоритме из предыдущей задачи. С помощью Беллмана-Форда находим расстояния до всех вершин в графе и делаем вывод в формате, как это требует задача. Еще в этой задаче я заметила, что если float('inf') заменить на math.inf или просто заранее заданное больше число программа будет работать быстрее(~=2,5 сек), но и вариант с float('inf') вполне укладывается во временные ограничения задачи.

Задача №12. Цветной лабиринт [2 баллов]

Текст задачи.

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.
- Примеры:

input.txt	output.txt
3 2 1 2 10 1 3 5 5 10 10 10 10 5	3

input.txt	output.txt
3 2 1 2 10 2 3 5 5 5 10 10 10 10	INCORRECT

input.txt	output.txt
3 2 1 2 10 1 3 5 4 10 10 10 5	INCORRECT

- Проверяем **обязательно** – на acmp.

Листинг кода. (именно листинг, а не скрины)

```
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

    print(graph)

    input_file.readline()

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

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

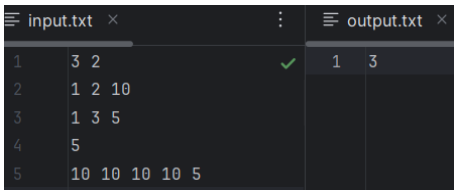
Текстовое объяснение решения.

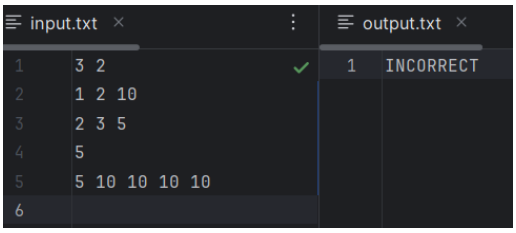
Считываем входные данные: словарь graph, такой что ключи - это номера вершин, значения - это словари, где ключи - цвета коридоров,

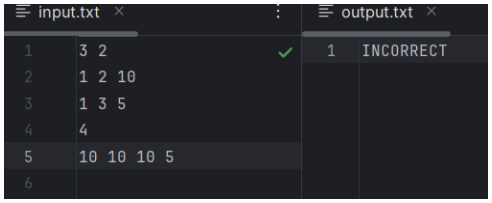
значения - номера комнат(вершин), соединенных с текущей этим коридором. Считываем описание пути colors.

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.

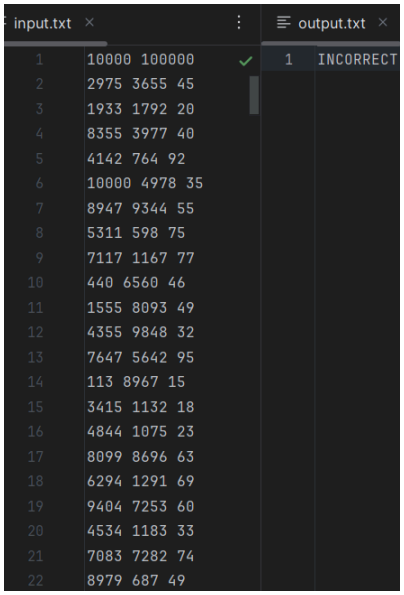
Результат работы кода на примерах из текста задачи:(скрины input output файлов)

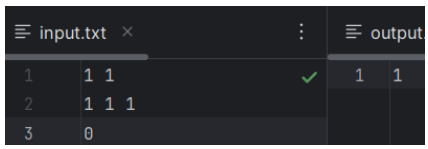






Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
21879741	09.09.2024 21:58:04	Иванова Дарья Сергеевна	0015	C++	Accepted		0,03	432 Кб
21879740	09.09.2024 21:57:58	Савченко Анастасия Сергеевна	0601	PyPy	Accepted		0,281	6,5 Мб

	Время выполнения, с	Затраты памяти , Мб
Нижняя граница диапазона значений входных данных из текста задачи	0.000490700011	14.68359375
Пример из задачи	0.0004649999318	14.85546875
Пример из задачи	0.000465799937956	14.703125
Пример из задачи	0.000498699955642	14.88671875
Верхняя граница диапазона значений входных данных из текста задачи	0.1024510000133	17.51171875

Вывод по задаче:

Классная задачка, мне понравилась, сначала прикинула граф, чтобы понять пример, а потом написала простой алгоритм для решения задачи.

Вывод

В ходе лабораторной работы я научилась решать задачи на графы. Узнала о списке и матрице смежности. Научилась реализовать алгоритмы поиска в глубину и в ширину для графов, а также алгоритмы для поиска кратчайших путей в графе с неотрицательными ребрами(алгоритм Дейкстры) и с возможными отрицательными весами ребер (алгоритм Беллмана-Форда).

После лабораторной на бинарные деревья, графы, конечно, дались проще. Но даже на тех задачах, которые приведены этой лабораторной можно заметить какие разнообразные задачи на графы бывают и сколько много алгоритмов для них существует. Складывается ощущение, что на самом деле(а это так и есть) тема очень обширная, и существует еще много разных задач на графы.