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 5

Выполнил:
Егоров Алексей Алексеевич
К3141

Проверила:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

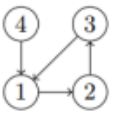
Задача №5. Город с односторонним движением [1.5 баллов]

Департамент полиции города сделал все улицы односторонними. Вы хотели бы проверить, можно ли законно проехать с любого перекрестка на какой-либо другой перекресток. Для этого строится ориентированный граф: вершины – это перекрестки, существует ребро (u, v) всякий раз, когда в городе есть улица (с односторонним движением) из u в v. Тогда достаточно проверить, все ли вершины графа лежат в одном компоненте сильной связности.

Нужно вычислить количество компонентов сильной связности заданного ориентированного графа с n вершинами и m ребрами.

- **Формат ввода / входного файла (input.txt).** Ориентированный граф с n вершинами и m ребрами по формату 1.
- **Ограничения на входные данные.** $1 \le n \le 10^4, 0 \le m \le 10^4$.
- **Формат вывода / выходного файла (output.txt).** Выведите число – количество компонентов сильной связности.
- Ограничение по времени. 5 сек.
- Ограничение по памяти. 512 мб.
- Пример 1:

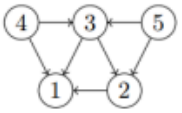
input	output
4 4 1 2 4 1 2 3 3 1	2



В этом графе есть два компонента сильной связности: $\{1, 3, 2\}$ и $\{4\}$.

- Пример 2:

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



В этом графе пять компонента сильной связности: $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$ и $\{5\}$.

Листинг кода.

```
import time
import tracemalloc


def first_dfs(graph, v, visited, attended):
    visited[v] = True
    for i in graph[v]:
        if not visited[i]:
```

```
            first_dfs(graph, i, visited, attended)
    attended.append(v)

def second_dfs(graph, v, visited):
    visited[v] = True
    for i in graph[v]:
        if not visited[i]:
            second_dfs(graph, i, visited)

# ============================

def transpose_graph(graph, n):
    new_graph = dict({i: [] for i in range(1, n + 1)})
    for i in graph:
        for j in graph[i]:
            new_graph[j].append(i)
    return new_graph

# ============================

def count_components(n, Graph):
    attended = []
    visited = [False] * (n + 1)

    for i in range(1, n + 1):
        if not visited[i]:
            first_dfs(Graph, i, visited, attended)

    gr = transpose_graph(Graph, n)

    visited = [False] * (n + 1) #очищаем visited
    count = 0
    while attended:
        i = attended.pop()
        if not visited[i]:
            second_dfs(gr, i, visited)
            count += 1
```

```
    return count


def main():
    with open("input.txt") as f:
        n, m = map(int, f.readline().split())
        Graph = dict({i: [] for i in range(1, n + 1)})
        for _ in range(m):
            u, v = map(int, f.readline().split())
            Graph[u].append(v)

    with open("output.txt", "w") as f:
        f.write(str(count_components(n, Graph)))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения

Граф представлен в виде списка смежностей, где узлы – перекрёстки.

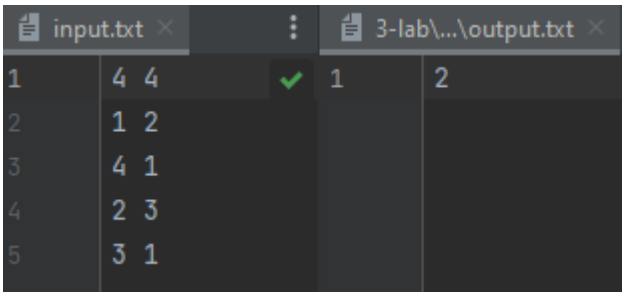
Функция first_dfs реализует поиск в глубину в графе. Она принимает на вход граф, стартовую вершину v, список посещённых вершин и стек для хранения вершин в том порядке, в котором они были обнаружены. Функция transpose создаёт транспонированный граф, в котором каждое ребро (u, v) исходного графа становится (v, u) в новом.

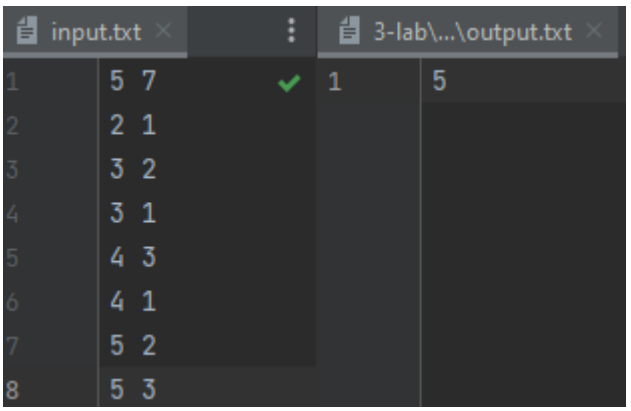
Функция second_dfs аналогична функции first_dfs, но не добавляет вершины в стек. Она используется для транспонированного графа.

Функция count_components принимает на вход количество узлов и сам граф. Инициализируется стек и список посещенных узлов. Выполняется DFS на исходном графе, чтобы заполнить стек вершинами в порядке их

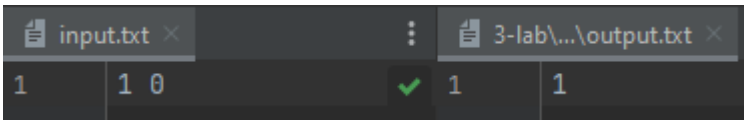
обнаружения. Далее происходит транспонирование графа с помощью функции transpose. Сбрасывается список посещённых узлов. Пока стек не пуст, из него извлекается номер узла. Если этот узел ещё не был посещён, то запускается функция second_dfs. Когда функция second_dfs прекращает работу, это означает, что были пройдены все вершины в одной компоненте сильной связности. Счётчик count увеличивается на единицу. Функция возвращает количество найденных компонентов сильной связности.

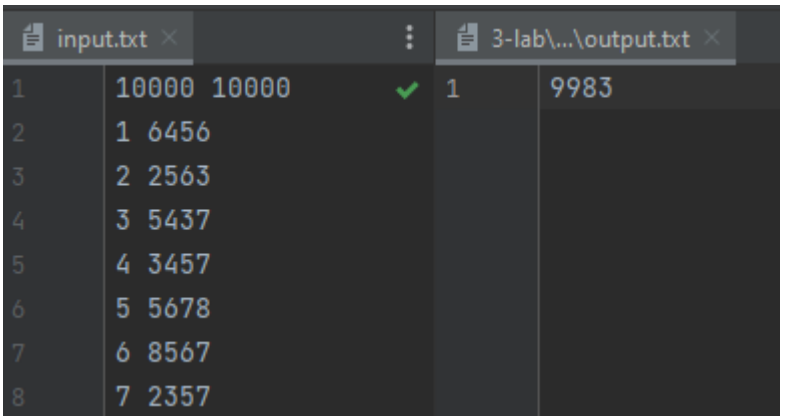
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ

Нижняя граница диапазона значений входных данных из текста задачи	0.0003508999943733215 3	18,61
Пример из задачи	0.0003815999953076243 4	19,15
Пример из задачи	0.0004084000829607248 3	19,20
Верхняя граница диапазона значений входных данных из текста задачи	0.0794565656463853676	3187,32

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

Задача №9. Аномалии курсов валют [2 баллов]

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 $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot \ldots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$.

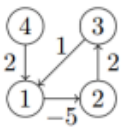
Для этого построим следующий граф: вершинами являются валюты $c_1, c_2, \ldots, c_n$, вес ребра из c_i в c_j равен $-\log r_{ij}$. Тогда достаточно проверить, есть ли в этом графе отрицательный цикл. Пусть цикл $c_i \to c_j \to c_k \to c_i$ имеет отрицательный вес. Это означает, что $-(\log c_{ij} +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

```
import time
import tracemalloc


def bellman_ford(graph, n):
    distance = [float("inf")] * (n + 1)
    distance[1] = 0
```

```
    for _ in range(n - 1):
        for u, v, value in graph:
            if distance[u] != float("inf") and distance[u] + value < distance[v]:
                distance[v] = distance[u] + value

    for u, v, value in graph:
        if distance[u] != float("inf") and distance[u] + value < distance[v]:
            return False
    return True


def main():
    with open("input.txt", "r") as f:
        n, m = map(int, f.readline().split())
        graph = []
        for _ in range(m):
            graph.append(tuple(map(int, f.readline().split())))

    with open("output.txt", "w") as f:
        if bellman_ford(graph, n):  # используем алгоритм Беллмана-Форда
            f.write('0')
        else:
            f.write('1')


if __name__ == '__main__':
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения

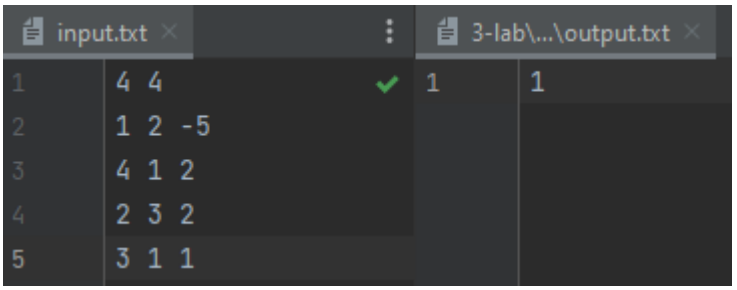
Функция bellman_ford реализует алгоритм Беллмана-Форда. Она на вход принимает граф и число вершин.

Инициализируется список distance для хранения кратчайшего расстояния от исходной вершины до всех остальных вершин. Затем выполняется релаксация всех рёбер n - 1 раз, где n - количество вершин. Этот шаг гарантирует нахождение кратчайшего пути.

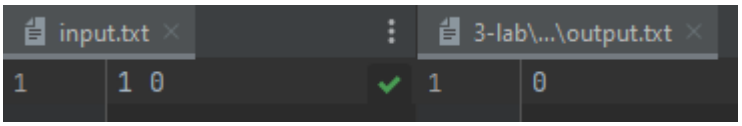
Далее проверяется наличие циклов с отрицательным весом. Если в ходе повторного перебора был найден более короткий путь, это указывает на наличие цикла с отрицательным весом, и функция возвращает значение False.

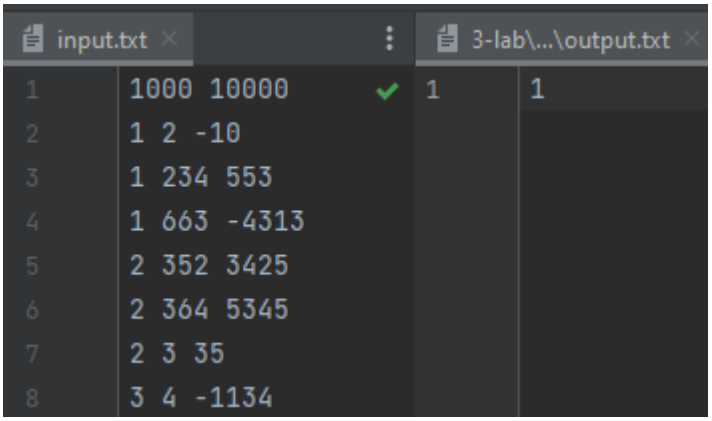
Если циклы с отрицательным весом не обнаружены, функция возвращает значение True.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003665999975055456	18,60
Пример из задачи	0.0003535000141710004	18,61

	3	
Верхняя граница диапазона значений входных данных из текста задачи	9.8945624566784675657	1410,54

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти.
Был использован алгоритм Беллмана-Форда

Задача №16. Рекурсия [3 баллов]

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 ($1 \leq n \leq 100$). 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 ($k \leq n$) – 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.

- Пример:

input.txt	output.txt
3 p1 2 p1 p2 ***** p2 1 p1 ***** p3 1 p1 *****	YES YES NO

- Проверяем **обязательно** – на acmp.

Листинг кода.

```
import time
import tracemalloc
```

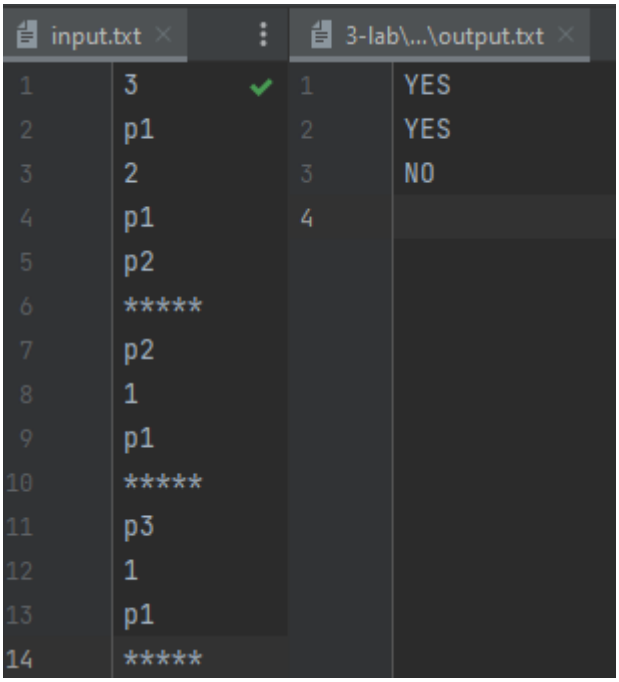
```
def find_cycle(pos, end, ways, visited):
    if pos in visited and visited[pos] == end:
        return pos == end
    visited[pos] = end
    for new_pos in ways[pos]:
        if find_cycle(new_pos, new_pos, ways, visited):
            return True
    return False


def main():
    with open("input.txt", "r") as f:
        n = int(f.readline().strip())

        ways = {}
        identifiers = []

        for _ in range(n):
            identifier = f.readline().strip()
            k = int(f.readline().strip())
            identifiers.append(identifier)
            ways[identifier] = []
            for _ in range(k):
                id_process = f.readline().strip()
                ways[identifier].append(id_process)
            f.readline()

    visited = {}
    with open("output.txt", "w") as f:
        for id in identifiers:
            visited.clear()
            if find_cycle(id, id, ways, visited):
                f.write("YES\n")
            else:
                f.write("NO\n")
```

```
if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

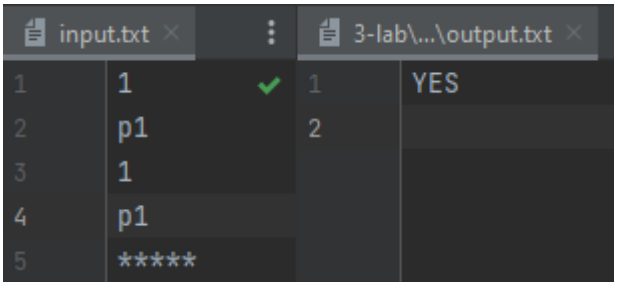
Текстовое объяснение решения

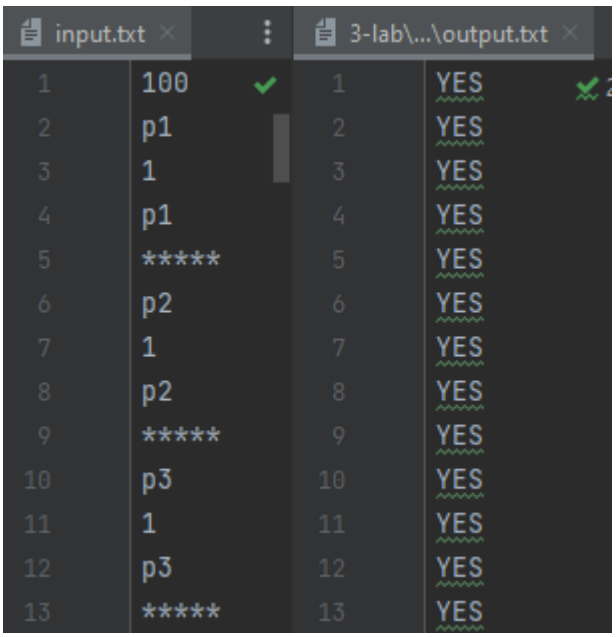
После считывания графа в виде списка смежности, запускается рекурсивный поиск цикла от каждой вершины. Для этого используется алгоритм dfs. В ходе работы каждая посещённая вершина окрашивается. В случае, если находится цикл, выводится YES, иначе NO.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





Проверка задачи на acmp.

Тест	Результат	Время	Память
1	Accepted	0,015	2062 Кб
2	Accepted	0,109	4046 Кб
3	Accepted	0,125	4514 Кб
4	Accepted	0,046	2162 Кб
5	Accepted	0,046	2130 Кб
6	Accepted	0,046	2150 Кб
7	Accepted	0,062	2118 Кб
8	Accepted	0,062	2166 Кб
9	Accepted	0,046	2118 Кб
10	Accepted	0,031	2114 Кб
11	Accepted	0,062	2158 Кб
12	Accepted	0,093	2174 Кб
13	Accepted	0,046	2122 Кб
14	Accepted	0,046	2174 Кб
15	Accepted	0,031	2122 Кб
16	Accepted	0,062	2146 Кб
17	Accepted	0,046	2126 Кб
18	Accepted	0,031	2114 Кб
19	Accepted	0,078	2614 Кб
20	Accepted	0,125	4578 Кб

	Время выполнения, с	Затраты памяти, КБ

Нижняя граница диапазона значений входных данных из текста задачи	0.0003688998986035585 4	19,01
Пример из задачи	0.0003785999724641442 3	19,37
Верхняя граница диапазона значений входных данных из текста задачи	0.0012389000039547682	40,59

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти.

На acmp работает корректно.

Дополнительные задачи

Задача №8. Стоимость полета [1.5 баллов]

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).

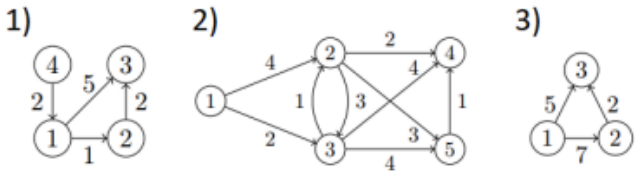
- **Формат ввода / входного файла (input.txt).** Ориентированный взвешенный граф задан по формату 1. Следующая строка содержит две вершины u и v.
- **Ограничения на входные данные.** $1 \leq n \leq 10^4$, $0 \leq m \leq 10^5$, $1 \leq u, v \leq n$, $u \neq v$, вес каждого ребра – неотрицательное целое 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.

```
import time
import tracemalloc


def find_short_dist(graph, fly_start, fly_end):
    costs = {i: float('inf') for i in graph}
    costs[fly_start] = 0
```

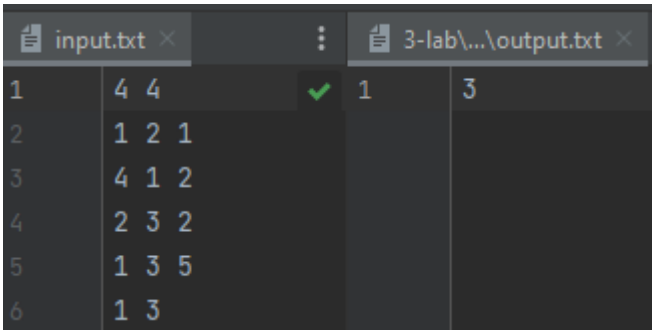
```
    visited = set()

    while True:
        min_cost = float('inf')
        current_node = None
        for node in graph:
            if costs[node] < min_cost and node not in visited:
                min_cost = costs[node]
                current_node = node
        if  current_node is None or current_node == fly_end:
            break
        visited.add(current_node)
        for fly_out, fly_in, cost in graph[current_node]:
            if cost + costs[current_node] < costs[fly_in]:
                costs[fly_in] = cost + costs[current_node]
    return costs[fly_end] if costs[fly_end] != float('inf') else -1


def main():
    with open("input.txt", "r") as f:
        n, m = map(int, f.readline().split())
        Graph = dict({i: [] for i in range(1, n + 1)})
        for i in range(m):
            fly_out, fly_in, cost = map(int, f.readline().split())
            Graph[fly_out].append((fly_out, fly_in, cost))

        fly_out, fly_in = map(int, f.readline().split())

    with open("output.txt", "w") as f:
                f.write(str(find_short_dist(Graph,  fly_out,  fly_in))) # Используем
алгоритм дийкстры



if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
```

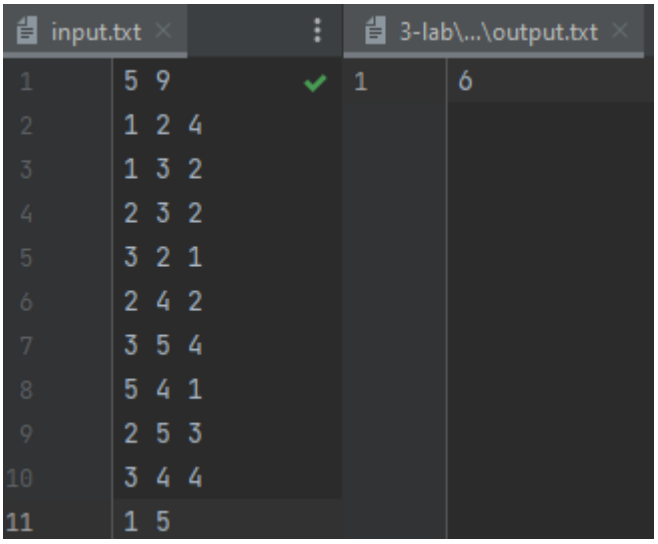
```
main()
t_end = time.perf_counter()
print(f"Время: {t_end - t_start} секунд")
print(f"Память: {tracemalloc.get_traced_memory()}")
tracemalloc.stop()
```

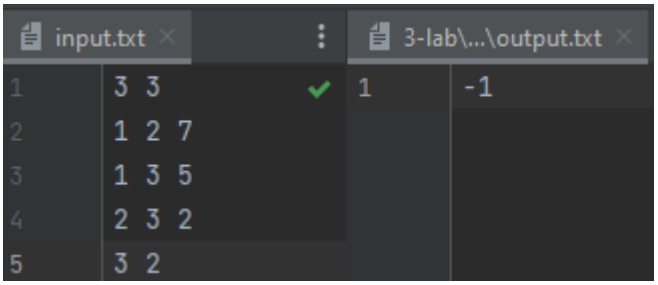
Текстовое объяснение решения.

Функция find_short_dist принимает на вход список смежностей, стартовый узел и конечный узел. Инициализирует словарь costs для хранения наименьшей стоимости перелёта, а также множество visited для отслеживания посещенных узлов. Итеративно находит непосещённый узел с наименьшей стоимостью. Затем добавляет его в visited и обновляет стоимость для соседних вершин. Это продолжается до тех пор, пока не будут посещены все узлы или не будет достигнут конечный узел. Функция возвращает наименьшую стоимость, если она существует, в противном случае возвращает значение -1.

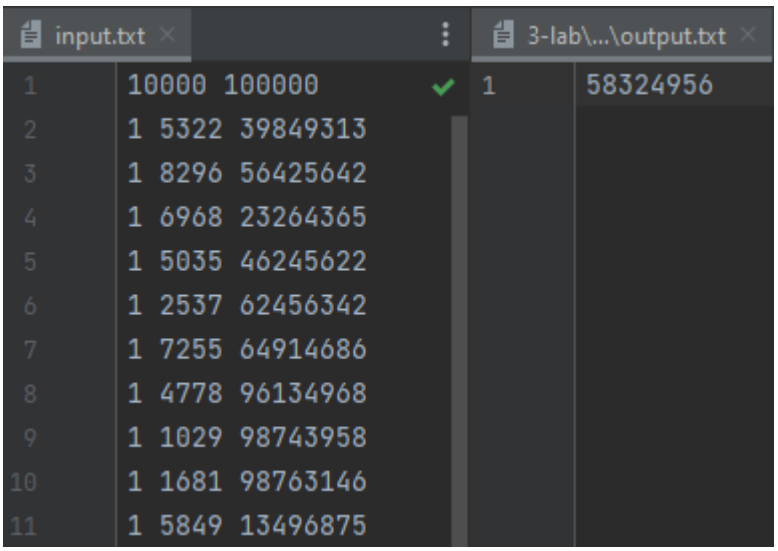
Результат работы кода на примерах из текста задачи:

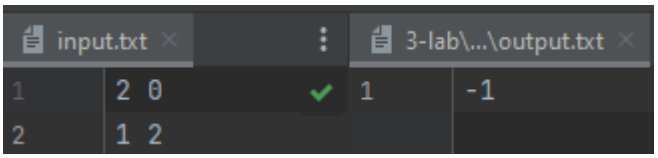






Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003908999497070908 5	19,02
Пример из задачи	0.005433000042103231	19,21
Пример из задачи	0.0004009000258520245 6	19,59
Пример из задачи	0.000382099999114871	19,10
Верхняя граница диапазона значений входных данных из текста задачи	0.0024356456354645546	16,95

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти.

Задача №11. Алхимия [3 баллов]

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 ($0 \le m \le 1000$) – 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.

- Примеры:

input.txt	output.txt
5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Aurum	2

input.txt	output.txt
5 Aqua -> AquaVita AquaVita -> PhilosopherStone AquaVita -> Argentum Argentum -> Aurum AquaVita -> Aurum Aqua Osmium	-1

- Проверяем **обязательно** – на acmp.

Листинг кода.

```
import time
import tracemalloc


def alchemy(all_reactions, start_reaction, end_reaction):
    len = {}
```

```
    len[start_reaction] = 0

    incidents = []
    incidents.append(start_reaction)

    while incidents:
        current = incidents.pop(0)

        if current == end_reaction:
            return len[current]
        if current in all_reactions:
            for next in all_reactions[current]:
                if next not in len:
                    len[next] = len[current] + 1
                    incidents.append(next)
    return -1


def main():
    with open("input.txt", "r") as f:
        m = int(f.readline().strip())
        all_reactions = {}

        for _ in range(m):
            reaction = f.readline().split()
            if reaction[0] in all_reactions:
                all_reactions[reaction[0]].append(reaction[2])
            else:
                all_reactions[reaction[0]] = []
                all_reactions[reaction[0]].append(reaction[2])
        start_reaction = f.readline().strip()
        end_reaction = f.readline().strip()

    with open("output.txt", "w") as f:
        f.write(str(alchemy(all_reactions, start_reaction, end_reaction)))
```

```
if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения.

Граф представлен в виде списка смежностей incidents. Узлы – названия химических веществ, которые являются исходными в описанных химических реакциях, рёбра – сами химические реакции.

Функция alchemy принимает на вход список реакций, название исходного вещества и название требуемого вещества.

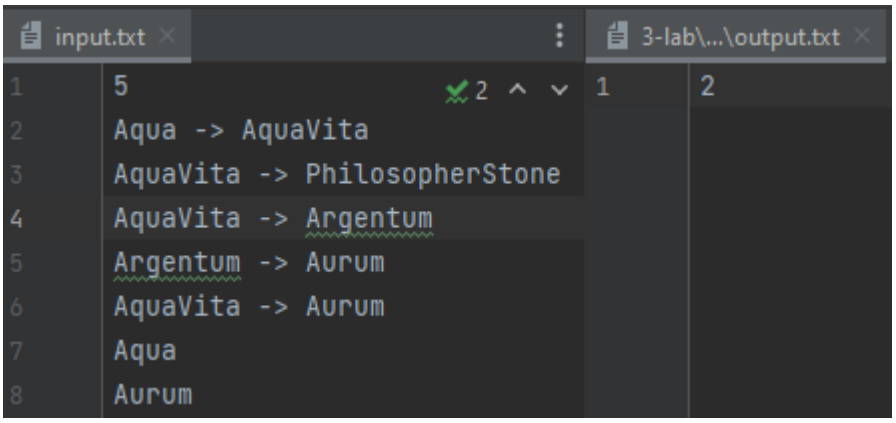
Инициализируется словарь len, ключами которого являются названия веществ, описываемых в реакциях. Он хранит количество химических реакций, которые требуется провести, чтобы получить текущее вещество из исходного.

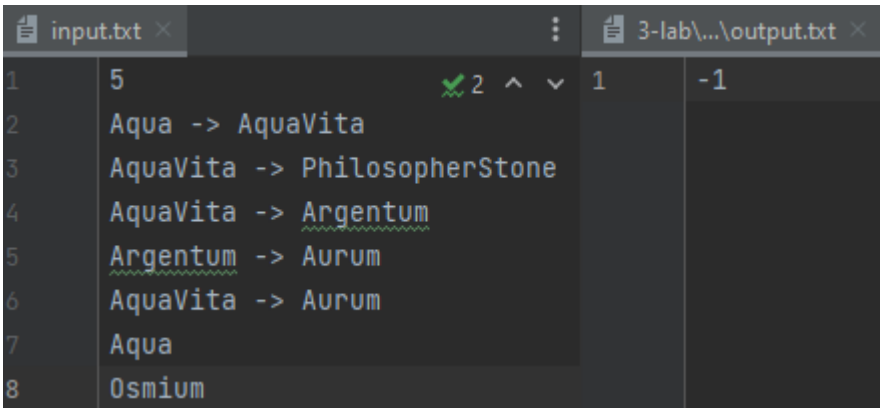
Затем инициализируется очередь, и в неё добавляется название исходного вещества.

Далее в цикле пока очередь не будет пуста, извлекается название химического вещества из неё. Если оно совпадает с названием требуемого вещества, то функция возвращает значение из словаря len для данного вещества. Иначе если это вещество является исходным в какой-либо реакции, то перебираются все химические реакции, где он является исходным веществом. Если получаемого в ходе такой реакции вещества нет в len (значит этот узел в графе ещё не был посещён), то оно туда заносится со значением из len текущего исходного вещества плюс один, а также оно добавляется в очередь.

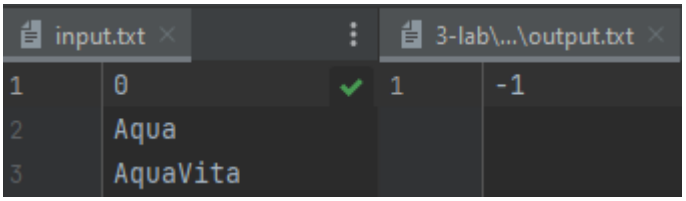
Если цикл не был прерван, то это значит, что невозможно из описанных реакций нельзя получить требуемое вещество, и функция возвращает значение -1.

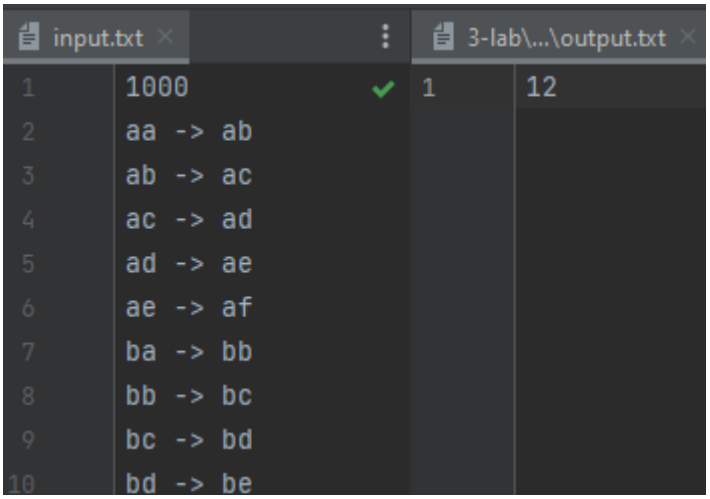
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





Проверка задачи на acmp

Тест	Результат	Время	Память
1	Accepted	0,046	2050 Кб
2	Accepted	0,031	2042 Кб
3	Accepted	0,062	2050 Кб
4	Accepted	0,046	2046 Кб
5	Accepted	0,031	2042 Кб
6	Accepted	0,062	2046 Кб
7	Accepted	0,015	2046 Кб
8	Accepted	0,046	2046 Кб
9	Accepted	0,046	2050 Кб
10	Accepted	0,062	2082 Кб
11	Accepted	0,046	2086 Кб
12	Accepted	0,031	2078 Кб
13	Accepted	0,015	2078 Кб
14	Accepted	0,046	2090 Кб
15	Accepted	0,015	2086 Кб
16	Accepted	0,046	2094 Кб
17	Accepted	0,031	2102 Кб
18	Accepted	0,046	2226 Кб
19	Accepted	0,031	2046 Кб
20	Accepted	0,015	2054 Кб
21	Accepted	0,078	2090 Кб
22	Accepted	0,046	2082 Кб
23	Accepted	0,046	2118 Кб
24	Accepted	0,046	2078 Кб
25	Accepted	0,062	2086 Кб
26	Accepted	0,046	2094 Кб
27	Accepted	0,031	2094 Кб
28	Accepted	0,031	2126 Кб
29	Accepted	0,031	2226 Кб
30	Accepted	0,062	2114 Кб
31	Accepted	0,062	2234 Кб
32	Accepted	0,078	2230 Кб
33	Accepted	0,046	2226 Кб
34	Accepted	0,031	2226 Кб
35	Accepted	0,046	2174 Кб
36	Accepted	0,046	2186 Кб
37	Accepted	0,046	2162 Кб
38	Accepted	0,078	2174 Кб

	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00032049999572336674	18,49

Пример из задачи	0.0004118999931961298	19,70
Пример из задачи	0.0003702000249177215	19,70
Верхняя граница диапазона значений входных данных из текста задачи	0.0051314509654890567	93,64

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти.

На acmp работает корректно.

Задача №12. Цветной лабиринт [2 баллов]

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.

```
import time
import tracemalloc


def passage_of_the_maze(Data, way):
    room = 1
    for color in way:
        room = Data[room][color]
        if room == 0:
            return 'INCORRECT'
    return str(room)
```

```
def main():
    with open('input.txt') as f:
        n, m = map(int, f.readline().split())
        Data = [[0]*101 for _ in range(n + 1)]
        for _ in range(m):
            first_room, second_room, color = map(int, f.readline().split())
            Data[first_room][color] = second_room
            Data[second_room][color] = first_room
        f.readline()
        way = map(int, f.readline().split())

    with open("output.txt", "w") as f:
        f.write(passage_of_the_maze(Data, way))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

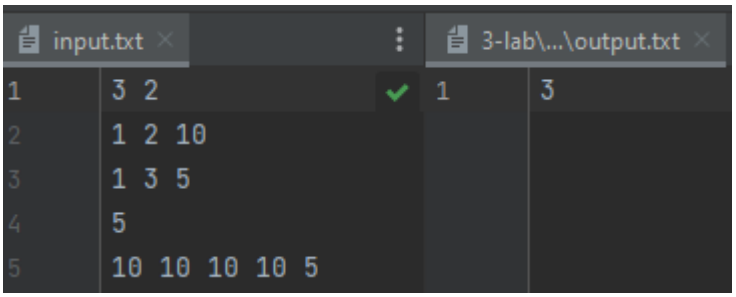
Текстовое объяснение решения.

Граф представлен в виде таблицы, где столбцы — это значения цветов, а строки - номера комнат. Если какие-то две комнаты соединены коридором какого-то цвета, то на пересечении строки этой комнаты и столбца этого цвета в ячейку заносится номер этой соседней комнаты.

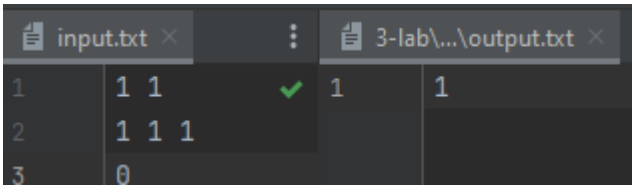
Функция passage_of_the_maze принимает на вход таблицу A и описание пути way. Осуществляется проход в цикле по всему пути, переменной room на каждой итерации присваивается номер комнаты, куда ведёт коридор текущего цвета из описания пути. Если переменной room на каком-то шаге присваивается значение ноль, то это означает, что нет

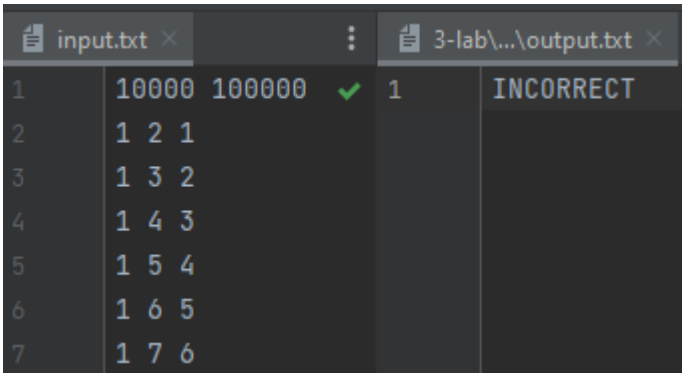
коридора такого цвета, ведущего из этой комнаты. В таком случае возвращается «INCORRECT». Если этого не произошло, и цикл не прерывался, то это значит, что путь корректен. Функция возвращает номер последней комнаты.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





Проверка задачи на acmp

Тест	Результат	Время	Память
1	Accepted	0,015	502 Кб
2	Accepted	0,046	506 Кб
3	Accepted	0,031	1954 Кб
4	Accepted	0,015	3022 Кб
5	Accepted	0,015	3210 Кб
6	Accepted	0,015	2286 Кб
7	Accepted	0,031	2626 Кб
8	Accepted	0,046	9,7 Мб
9	Accepted	0,015	3310 Кб
10	Accepted	0,062	4922 Кб
11	Accepted	0,062	4078 Кб
12	Accepted	0,062	12 Мб
13	Accepted	0,062	7,2 Мб
14	Accepted	0,062	9,5 Мб
15	Accepted	0,062	7,7 Мб
16	Accepted	0,062	10 Мб
17	Accepted	0,078	8,9 Мб
18	Accepted	0,062	12 Мб
19	Accepted	0,078	8,6 Мб
20	Accepted	0,078	9,2 Мб
21	Accepted	0,078	15 Мб
22	Accepted	0,078	12 Мб

	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00036860001273453236	20,23
Пример из задачи	0.00486389989964664	21,93
Пример из задачи	0.00042729999404400587	21,93
Пример из задачи	0.00036900001578032897	21,92

Верхняя граница диапазона значений входных данных из текста задачи	0.4493246984587349856	13965,49

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти.

На acmp работает корректно.

Задача №14. Автобусы [3 баллов]

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.
- Пример:

input.txt	output.txt
3 1 3 4 1 0 2 5 1 1 2 3 2 3 3 5 1 1 3 10	5

- Проверяем **обязательно** – на acmp.

Листинг кода.

```
import time
import tracemalloc


def find_short_time(graph, start_pos, end_pos):
    distances = {i: float('inf') for i in graph}
```

```
    distances[start_pos] = 0
    visited = set()

    while True:
        min_distance = float('inf')
        current_node = None
        for node in graph:
            if distances[node] < min_distance and node not in visited:
                min_distance = distances[node]
                current_node = node
        if current_node is None or current_node == end_pos:
            break
        visited.add(current_node)
                    for start_place, leaving_time, end_position, coming_time in
graph[current_node]:
                    if leaving_time >= distances[current_node] and coming_time <
distances[end_position]:
                distances[end_position] = coming_time
    return distances[end_pos] if distances[end_pos] != float('inf') else -1


def main():
    with open('input.txt') as f:
        N = int(f.readline().strip())
        d, v = map(int, f.readline().strip().split())
        R = int(f.readline().strip())
        bus_schedule = {i: [] for i in range(1, N + 1)}
        for _ in range(R):
                    start_place, leaving_time, finish_place, coming_time = map(int,
f.readline().split())
                        bus_schedule[start_place].append((start_place, leaving_time,
finish_place, coming_time))

    with open("output.txt", "w") as f:
        f.write(str(find_short_time(bus_schedule, d, v)))
```

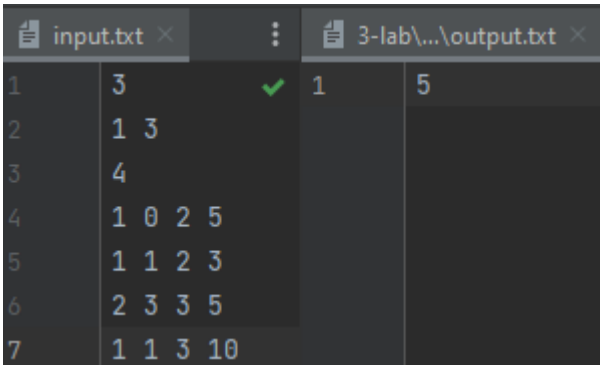
```
if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения.

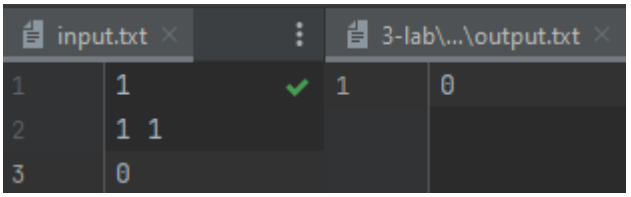
Граф представлен в виде словаря, где каждый ключ является узлом графа (деревней), а значение - списком кортежей, содержащих деревню, откуда идёт автобусный рейс, время отправления, время прибытия и деревню, куда автобус прибывает.

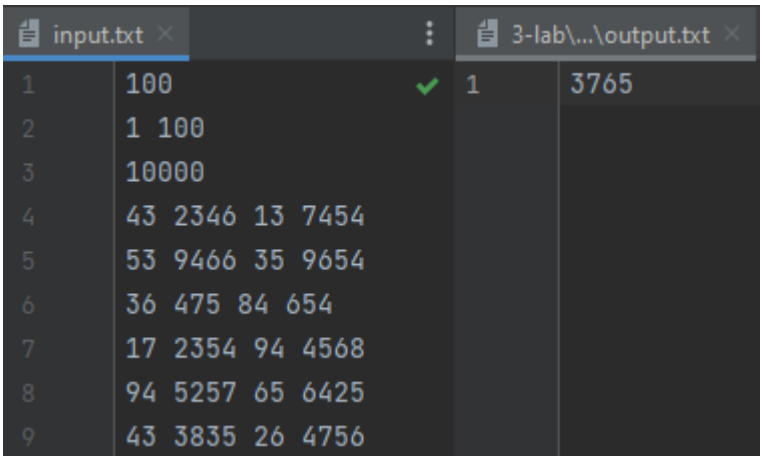
Функция find_short_time принимает на вход граф, стартовый узел и конечный узел. Инициализирует словарь distances для хранения наименьшего времени, требуемого для того, чтобы добраться из одной деревни в другую, а также множество visited для отслеживания посещенных узлов. Итеративно находит непосещённый узел с наименьшим временем. Затем добавляет его в visited и обновляет время, требуемое, чтобы добраться до его соседей. Это продолжается до тех пор, пока не будут посещены все узлы или не будет достигнут конечный узел. Функция возвращает кратчайшее расстояние до конечного узла, если оно существует, в противном случае возвращает значение -1.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





Проверка задачи на acmp

Тест	Результат	Время	Память
1	Accepted	0,031	2034 Кб
2	Accepted	0,031	2070 Кб
3	Accepted	0,078	2070 Кб
4	Accepted	0,031	2078 Кб
5	Accepted	0,046	2098 Кб
6	Accepted	0,031	2058 Кб
7	Accepted	0,093	2254 Кб
8	Accepted	0,093	2362 Кб
9	Accepted	0,343	6,2 Мб
10	Accepted	0,39	3510 Кб

	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0004422999918460846	18,59
Пример из задачи	0.006995600066147745	18,95
Верхняя граница диапазона значений входных данных из текста задачи	0.05435236528945363 16	1254,68

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти.

На acmp работает корректно.

Вывод

В ходе данной лабораторной работы я научился решать задачи. Написанные программы были протестированы, а также были измерены потребляемый ими объём памяти и время работы. Все программы работаю корректно и укладываются в установленные ограничения по времени и памяти на примерах из задач.