

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3
по курсу «Алгоритмы и структуры данных»

Вариант 14

Выполнила:

Лаузер Я.П. К3144

Проверил:

Санкт-Петербург

2024 г.

Содержание отчета

Лабораторная №3

Задача №3. Циклы

Задача 7. Двудольный граф

Задача №12. Цветной лабиринт

2 лаба “Двоичные деревья поиска”.

Задача №3.Циклы

Текст задачи

Учебная программа по инфокоммуникационным технологиям определяет пререквизиты для каждого курса в виде списка курсов, которые необходимо пройти перед тем, как начать этот курс. Вы хотите выполнить проверку согласованности учебного плана, то есть проверить отсутствие циклических зависимостей. Для этого строится следующий ориентированный граф: вершины соответствуют курсам, есть направленное ребро (u, v) – курс u следует пройти перед курсом v . Затем достаточно проверить, содержит ли полученный граф цикл.

Проверьте, содержит ли данный граф циклы.

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

Листинг кода.

```
def has_cycle(graph):
    visited = set()
    rec_stack = set()

    def dfs(vertex):
        if vertex in rec_stack:
            return True
        if vertex in visited:
            return False

        visited.add(vertex)
        rec_stack.add(vertex)

        for neighbor in graph.get(vertex, []):
            if dfs(neighbor):
                return True

        rec_stack.remove(vertex)
        return False
```

```

    for vertex in graph:
        if dfs(vertex):
            return True

    return False

filename = "input.txt"
graph = {}
with open(filename, 'r') as file:
    vertices, edges = map(int, file.readline().split())
    for _ in range(edges):
        u, v = map(int, file.readline().split())
        if u not in graph:
            graph[u] = []
        graph[u].append(v)

with open('output.txt', 'w') as file:
    if has_cycle(graph):
        file.write('1')
    else:
        file.write('0')

```

Текстовое объяснение решения.

Код работает следующим образом: 1) Чтение графа: Считывает количество вершин и ребер из файла. 2) Создание графа: Создает словарь "graph", где ключи - вершины, а значения - списки смежных вершин. 3) Поиск циклов: Использует алгоритм DFS (поиск в глубину, алгоритм обхода графа, который посещает все вершины, связанные с текущей вершиной, перед переходом к следующей) с дополнительным множеством "rec_stack", чтобы обнаружить

циклы. 4) Запись результата: Записывает 1 в файл "output.txt", если цикл найден, иначе 0.

Результат работы кода на примерах из текста задачи:

3.py	input.txt	output.txt
1	4 4	
2	1 2	
3	4 1	
4	2 3	
5	3 1	

3.py	input.txt	output.txt
1	1	

Задача №7. Двудольный граф

Текст задачи

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.** $1 \leq n \leq 10^5$, $0 \leq m \leq 10^5$.

Листинг кода

```
def is_bipartite(graph):
    color = {}

    def dfs(vertex, current_color):
        color[vertex] = current_color
        next_color = 1 - current_color
        for neighbor in graph.get(vertex, []):
            if neighbor in color:
                if color[neighbor] == current_color:
                    return False
            else:
                if not dfs(neighbor, next_color):
                    return False
        return True

    for vertex in graph:
        if vertex not in color:
            if not dfs(vertex, 0):
                return 0
    return 1

filename = "input.txt"
graph = {}
with open(filename, 'r') as file:
    num_vertices, num_edges = map(int, file.readline().split())
```

```

for _ in range(num_edges):
    u, v = map(int, file.readline().split())
    if u not in graph:
        graph[u] = []
    if v not in graph:
        graph[v] = []
    graph[u].append(v)
    graph[v].append(u)

result = is_bipartite(graph)
print(result)

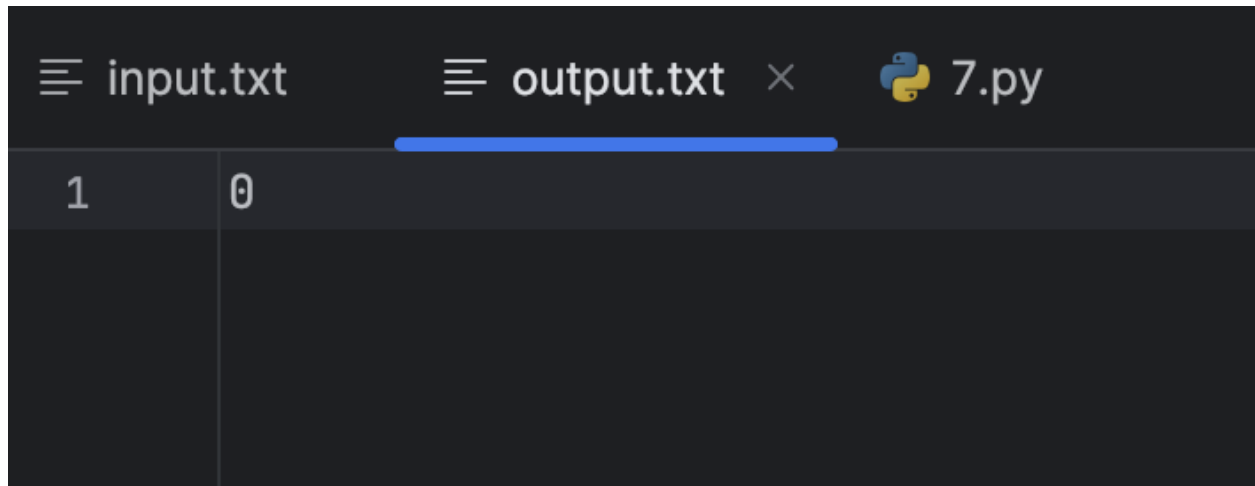
```

Объяснение кода:

Этот код проверяет, является ли данный граф двудольным. Он делает это, используя алгоритм поиска в глубину (DFS) для окраски вершин в графе в два цвета таким образом, чтобы никакие две смежные вершины не были одного цвета. Если алгоритм DFS может завершиться, не обнаружив двух смежных вершин одного цвета, граф является двудольным, иначе нет. Результат (1 или 0) записывается в файл 'output.txt'.

Результат работы кода на примерах из текста задачи:

input.txt		output.txt		7.py
1	4 4			
2	1 2			
3	4 1			
4	2 3			
5	3 1			



Задача №12. Цветной лабиринт

Текст задачи

В одном из парков одного большого города недавно был организован новый аттракцион Цветной лабиринт. Он состоит из n комнат, соединенных m двусторонними коридорами. Каждый из коридоров покрашен в один из s 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, сообщаящей 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 ($1 \leq n \leq 10000$) и m ($1 \leq m \leq 100000$) - соответственно количество комнат и коридоров в лабиринте. Следующие m строк содержат описания коридоров. Каждое описание содержит три числа u ($1 \leq u \leq n$), v ($1 \leq v \leq n$), c ($1 \leq c \leq 100$) - соответственно номера комнат, соединенных этим коридором, и цвет коридора. Следующая, $(m + 2)$ -ая строка входного файла содержит длину описания пути - целое число k ($0 \leq k \leq 100000$). 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.

Листинг кода:

```
from collections import defaultdict
```



```
def check_path_correctness( corridors, path):
    graph = defaultdict(list)
    for u, v, c in corridors:
        graph[u].append((v, c))
        graph[v].append((u, c))

    current_room = 1

    for color in path:
        next_room = None

        for room, corridor_color in
graph[current_room]:
            if corridor_color == color:
                next_room = room
                break

        if next_room is None:
            return "INCORRECT"
        current_room = next_room
    return current_room

with open("input.txt", 'r') as file:
    n, m = map(int, file.readline().split())
    corridors = [tuple(map(int,
file.readline().split())) for _ in range(m)]
    path_length = int(file.readline())
    path = list(map(int, file.readline().split()))

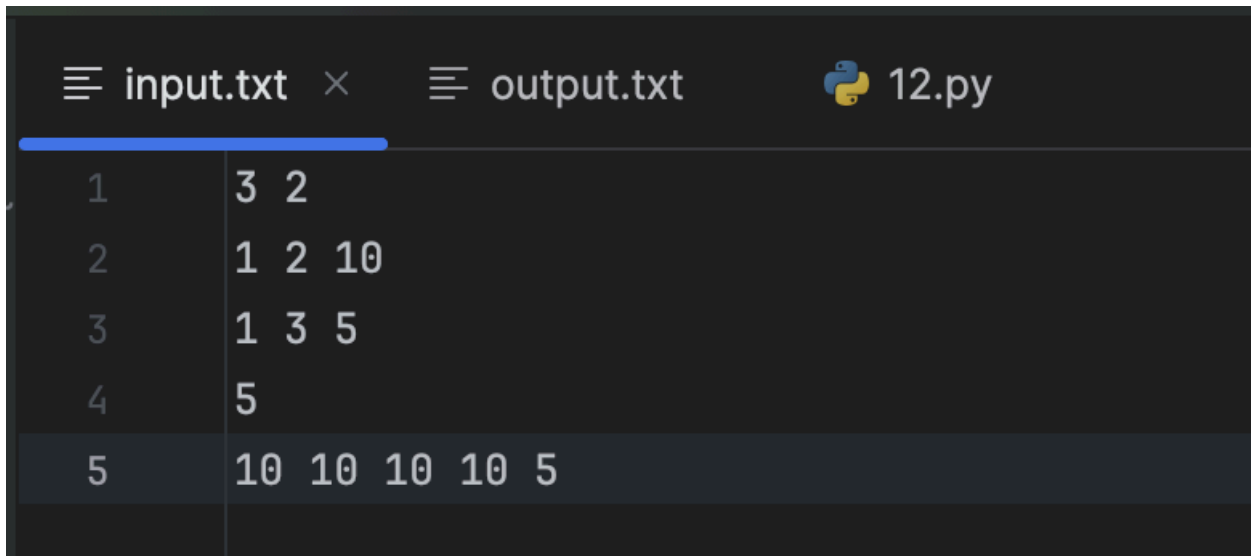
result = check_path_correctness(corridores, path)

with open("output.txt", 'w') as file:
    file.write(str(result))
```

Объяснение кода:

Этот код принимает в качестве входных данных количество комнат, количество коридоров, описания коридоров (какие комнаты они соединяют и какого цвета), длину пути и последовательность цветов коридоров пути. Он строит граф комнат и коридоров, а затем проходит по пути, проверяя, есть ли на каждом шаге коридор нужного цвета, чтобы перейти в следующую комнату. Если в какой-то момент пути такого коридора нет, то путь считается некорректным, и код выводит "INCORRECT". В противном случае код выводит номер комнаты, в которую приводит этот путь.

Результат работы кода на примерах из текста задачи:



The screenshot shows a code editor with two tabs: 'input.txt' and 'output.txt'. The 'input.txt' tab is active and displays the following content:

Line	Content
1	3 2
2	1 2 10
3	1 3 5
4	5
5	10 10 10 10 5

The 'output.txt' tab is empty. The code file is named '12.py'.

≡ input.txt	≡ output.txt ×	🐍 12.py
1	3	

≡ input.txt ×	≡ output.txt	🐍 12.py
1	3 2	
2	1 2 10	
3	2 3 5	
4	5	
5	10 10 10 10 5	

≡ input.txt	≡ output.txt ×	🐍 12.py
1	INCORRECT	

