

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

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»

Тема: Графы

Вариант 16

Выполнила:

Бархатова Н.А.

К3139

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Лабиринт [1 балл]	3
Задача №9. Аномалии курсов валют [2 балла]	5
Задача №16. Рекурсия [3 балла]	7
Дополнительные задачи	8
Задача №2. Компоненты [1 балл]	8
Задача №3. Циклы [1 балл]	10
Задача №4. Порядок курсов [1 балл]	13
Задача №11. Алхимия [3 балла]	15
Задача №12. Цветной лабиринт [2 балла]	17
Задача №13. Грядки [3 балла]	20
Задача №14. Автобусы [3 балла]	22
Вывод	24

Задачи по варианту

Задача №1. Лабиринт [1 балл]

Текст задачи.

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.

Листинг кода.

```
import time
import tracemalloc

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
vertices, edges = map(int, input_file.readline().split())
graph = {}
for i in range(edges):
    from_vertex, to_vertex = map(int, input_file.readline().split())
    if from_vertex not in graph.keys():
        graph[from_vertex] = []
    if to_vertex not in graph.keys():
        graph[to_vertex] = []
    graph[from_vertex].append(to_vertex)
    graph[to_vertex].append(from_vertex)
u, v = map(int, input_file.readline().split())
visited = [False] * (vertices+1)
order = []
def explore(v):
    order.append(v)
    visited[v] = True
    for w in graph[v]:
        if not visited[w]:
            explore(w)
def DFS(graph, start):
    for v in list(graph.keys())[start-1:]:
```

```

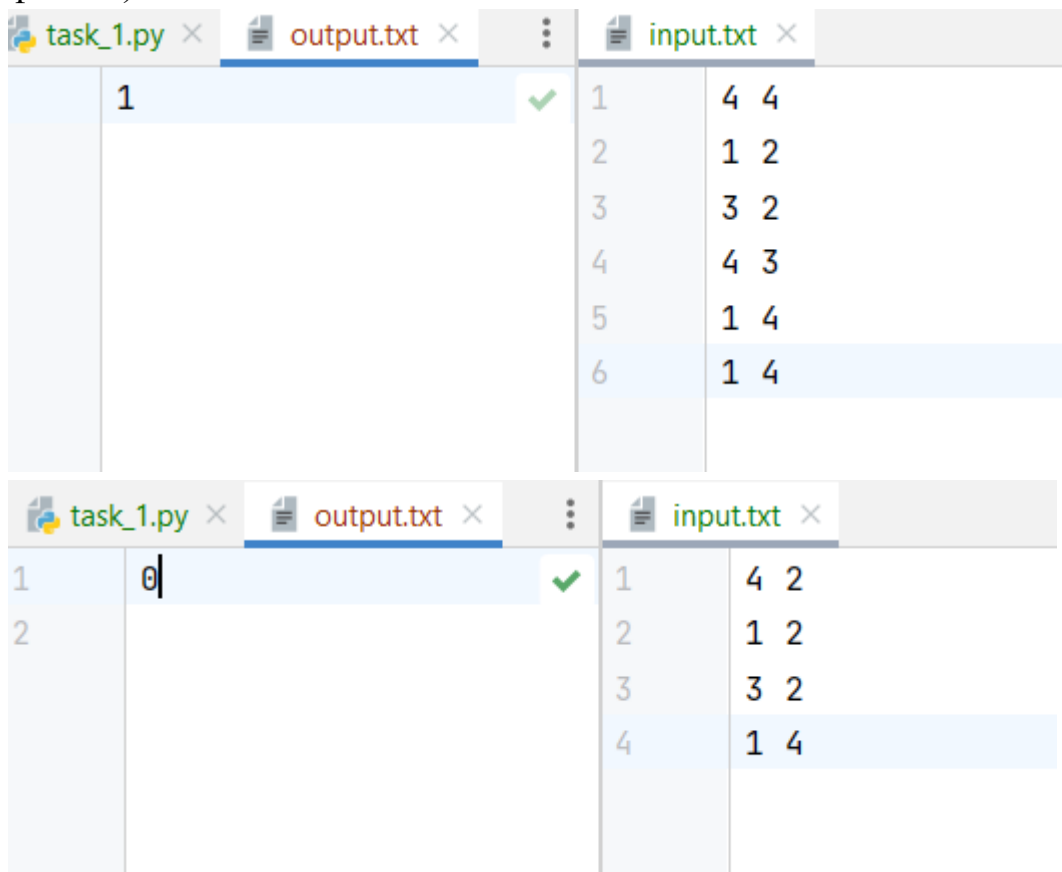
        if not visited[v]:
            explore(v)
        return order
with open('output.txt', 'w') as output_file:
    print(int(v in DFS(graph, u)), file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()

```

Текстовое объяснение решения.

Считываю input.txt и задаю граф как список смежностей. Создаю функцию обхода графа. Если во время обхода графа с первой вершины попадаетея вторая, то значит, они соединены между собой.

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



	Время выполнения	Затраты памяти
Пример из задачи	0.0010	0.0037
Пример из задачи	0.0008	0.0037

Вывод по задаче: легко и просто

Задача №9. Аномалии курсов валют [2 балла]

Текст задачи.

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.

Листинг кода.

```
import time
import tracemalloc

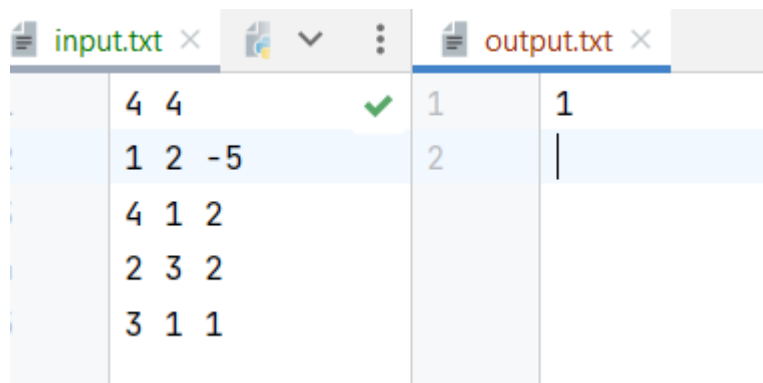
def bellman_ford(n, S, edges):
    dist = [float('inf')] * (n + 1)
    dist[S] = 0
    for _ in range(n - 1):
        for u, v, w in edges:
            if dist[u] != float('inf') and dist[v] > dist[u] + w:
                dist[v] = dist[u] + w
        no_inf_dist = [i for i in dist if isinstance(i, int)]
        if sum(no_inf_dist) < 0:
            return 1
    return 0

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
vertices, edges = map(int, input_file.readline().split())
values = [tuple(map(int, input_file.readline().split())) for _ in
range(edges)]
with open('output.txt', 'w') as output_file:
    print(bellman_ford(vertices, 1, values), file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()
```

Текстовое объяснение решения:

Реализуем алгоритм Беллмана-Форда

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



	Время выполнения	Затраты памяти
Пример из задачи	0.0008	0.0034

Вывод по задаче: выглядит сложнее, чем есть на самом деле

Задача №16. Рекурсия [3 балла]

Текст задачи.

Рассмотрим программу, состоящую из n процедур P_1, P_2, \dots, 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, \dots, Q_k , что $Q_0 = Q_k = P$ и для $i = 1 \dots k$ 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.

Листинг кода.

```
import time
import tracemalloc

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
n = int(input_file.readline().strip())
graph = {}
for _ in range(n):
    p_id = input_file.readline().strip()
    p_calls = int(input_file.readline().strip())
    graph[p_id] = []
    if p_calls != 0:
        for _ in range(p_calls):
            called_prog = input_file.readline().strip()
            graph[p_id].append(called_prog)
        input_file.readline()

def dfs(v, visited, start_v):
    if v in visited:
        return v == start_v
    visited.append(v)
    for neighbor in graph[v]:
        if dfs(neighbor, visited, start_v):
            return True
    return False

result = ''

for p_id in list(graph.keys()):
    visited = []
    if dfs(p_id, visited, p_id) and graph[p_id] is not None:
        result += 'YES\n'
    else:
        result += 'NO\n'

with open('output.txt', 'w') as output_file:
    print(result, file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()
```

Текстовое объяснение решения.

Считываем данные в ориентированный граф. Определяем для каждой вершины её достижимость из самой же себя.

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

```

input.txt
1 3
2 p1
3 2
4 p1
5 p2
6 *****
7 p2
8 1
9 p1
10 *****
11 p3
12 1
13 p1
14 *****

task_16.py
acmp_check.py
output.txt
1 YES
2 YES
3 NO
4
5

```

Проверка задачи на (openedu, астр и тд при наличии в задаче). (скрин)

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19533913	13.06.2023 0:20:26	Бархатова Наталья Александровна	0345	Python	Accepted		0.062	1694 Kb
19533912	13.06.2023 0:20:21	Арслан Гаит	1418	Python	Wrong answer	13	0.046	466 Kb
19533911	13.06.2023 0:18:36	Беляков Семён Валерьевич	0295	Python	Wrong answer	1		494 Kb
19533910	13.06.2023 0:16:01	Кооп Максим	0867	Python	Wrong answer	3	0.031	486 Kb

	Время выполнения	Затраты памяти
Пример из задачи	0.0013	0.0042

Вывод по задаче: при решении используется алгоритм из первых задач этой лабораторной работы.

Дополнительные задачи

Задача №2. Компоненты [1 балл]

Текст задачи.

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.

Листинг кода.

```
import time
import tracemalloc

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
vertices, edges = map(int, input_file.readline().split())
graph = {}
for i in range(edges):
    from_vertex, to_vertex = map(int, input_file.readline().split())
    if from_vertex not in graph.keys():
        graph[from_vertex] = []
    if to_vertex not in graph.keys():
        graph[to_vertex] = []
    graph[from_vertex].append(to_vertex)
    graph[to_vertex].append(from_vertex)
while len(graph.keys()) != vertices:
    index = 1
    for i in range(1, len(graph.keys()) + 2):
        if i not in graph.keys():
            index = i
            break
    graph[index] = []
visited = [False] * (vertices + 1)
order = []
CC_nums = [0] * (vertices + 1)
cc_number = 1

def explore(v):
    CC_nums[v] = cc_number
    order.append(v)
    visited[v] = True
    for w in graph[v]:
        if not visited[w]:
            explore(w)

def DFS(graph, start):
    global cc_number
    for v in list(graph.keys())[start - 1:]:
        if not visited[v]:
            explore(v)
```

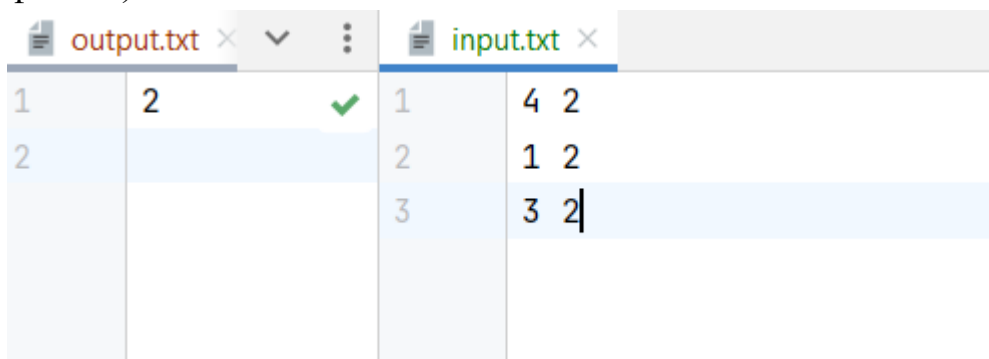
```
cc_number += 1
```

```
DFS(graph, 1)
with open('output.txt', 'w') as output_file:
    print(max(CC_nums), file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()
```

Текстовое объяснение решения.

Запускаю обход графа от вершины с номером 1. Если обход прервался, но при этом не посещенные вершины всё ещё есть, значит, при прерывании был завершен обход компоненты.

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



	Время выполнения	Затраты памяти
Пример из задачи	0.0023	0.0040

Вывод по задаче: весело

Задача №3. Циклы [1 балл]

Текст задачи.

Проверьте, содержит ли данный граф циклы.

Листинг кода.

```
import time
import tracemalloc

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
vertices, edges = map(int, input_file.readline().split())
```

```

graph = {}
for i in range(edges):
    from_vertex, to_vertex = map(int, input_file.readline().split())
    if from_vertex not in graph.keys():
        graph[from_vertex] = []
    if to_vertex not in graph.keys():
        graph[to_vertex] = []
    graph[from_vertex].append(to_vertex)
queue = []
visited = [False] * (vertices + 1)
colors = ['white' for _ in range(vertices + 1)]

def dfs(start, visited, graph):
    visited[start] = True
    colors[start] = 'grey'
    for v in graph[start]:
        if colors[v] != 'grey':
            dfs(v, visited, graph)
        else:
            raise Exception
    colors[start] = 'black'

with open('output.txt', 'w') as output_file:
    try:
        dfs(1, visited, graph)
        print(0, file=output_file)
    except:
        print(1, file=output_file)

print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()

```

Текстовое объяснение решения.

В случае ориентированного графа произведём серию обходов. То есть из каждой вершины, в которую мы ещё ни разу не приходили, запустим поиск в глубину, который при входе в вершину будет красить её в серый цвет, а при выходе из нее — в чёрный. И, если алгоритм пытается пойти в серую вершину, то это означает, что цикл найден.

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

input.txt	task_3.py	output.txt
1 5 7 ✓	1 0	
2 1 2	2	
3 2 3		
4 1 3		
5 3 4		
6 1 4		
7 2 5		
8 3 5		

input.txt	task_3.py	output.txt
1 4 4 ✓	1 1	
2 1 2	2	
3 4 1		
4 2 3		
5 3 1		

	Время выполнения	Затраты памяти
Пример из задачи	0.0013	0.0038

Вывод по задаче: весело

Задача №4. Порядок курсов [1 балл]

Текст задачи.

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

Листинг кода.

```
import time
import tracemalloc

start_time = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
vertices, edges = map(int, input_file.readline().split())
graph = {}
for i in range(edges):
    from_vertex, to_vertex = map(int, input_file.readline().split())
    if from_vertex not in graph.keys():
        graph[from_vertex] = []
    if to_vertex not in graph.keys():
        graph[to_vertex] = []
    graph[from_vertex].append(to_vertex)
graph = dict(sorted(graph.items()))
order = []
pre = [0 for _ in range(vertices + 1)]
post = [0 for _ in range(vertices + 1)]
visited = [False for _ in range(vertices + 1)]
clock = 1

def explore(v):
    order.append(v)
    visited[v] = True
    global clock
    pre[v] = clock
    clock += 1
    for w in graph[v]:
        if not visited[w]:
            explore(w)
    post[v] = clock
    clock += 1

def DFS(graph):
    for v in graph.keys():
        if not visited[v]:
            explore(v)

DFS(graph)
node_and_post = list(zip(order, [post[i] for i in order]))
topological_order = [i[0] for i in sorted(node_and_post, reverse=True,
key=lambda x: x[1])]
with open('output.txt', 'w') as output_file:
    print(*topological_order, file=output_file)
print("Время: ", time.perf_counter() - start_time)
print("Память: ", float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))
tracemalloc.stop()
```

Текстовое объяснение решения.

Вводим метки времени. Ответом будет являться порядок вершин, отсортированный по массиву post в порядке убывания.

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

The image shows three screenshots of a code editor with three tabs: `input.txt`, `task_4.py`, and `output.txt`.

First Screenshot:

- `input.txt`:


```
1 4 3
2 1 2
3 4 1
4 3 1
```
- `task_4.py`: A green checkmark is visible next to the first line.
- `output.txt`:


```
1 4 3 1 2
2 |
```

Second Screenshot:

- `input.txt`:


```
1 4 1
2 3 1
```
- `task_4.py`: A green checkmark is visible next to the first line.
- `output.txt`:


```
1 4 2 3 1
2 |
```

Third Screenshot:

- `input.txt`:


```
1 5 7
2 2 1
3 3 2
4 3 1
5 4 3
6 4 1
7 5 2
8 5 3
```
- `task_4.py`: A green checkmark is visible next to the first line.
- `output.txt`:


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

	Время выполнения	Затраты памяти
Пример из задачи	0.0028	0.0049
Пример из задачи	0.0012	0.0062

Вывод по задаче: класс

Задача №11. Алхимия [3 балла]

Текст задачи.

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.

Листинг кода.

```
from collections import defaultdict, deque
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
m = int(input_file.readline())
reactions = [input_file.readline() for _ in range(m)]
source = input_file.readline().strip()
target = input_file.readline().strip()

def set_graph(reactions):
    graph = defaultdict(list)
    for reaction in reactions:
        substance1, substance2 = reaction.split("->")
        graph[substance1.strip()].append(substance2.strip())
    return graph

def find_target(graph, source, target):
    visited = set()
    queue = deque()
    queue.append((source, 0))
    while queue:
        current, count = queue.popleft()
        if current == target:
            return count
```

```

        visited.add(current)
        for neighbor in graph[current]:
            if neighbor not in visited:
                queue.append((neighbor, count + 1))
    return -1

graph = set_graph(reactions)
min_reactions = find_target(graph, source, target)

with open("output.txt", "w") as file:
    file.write(str(min_reactions))

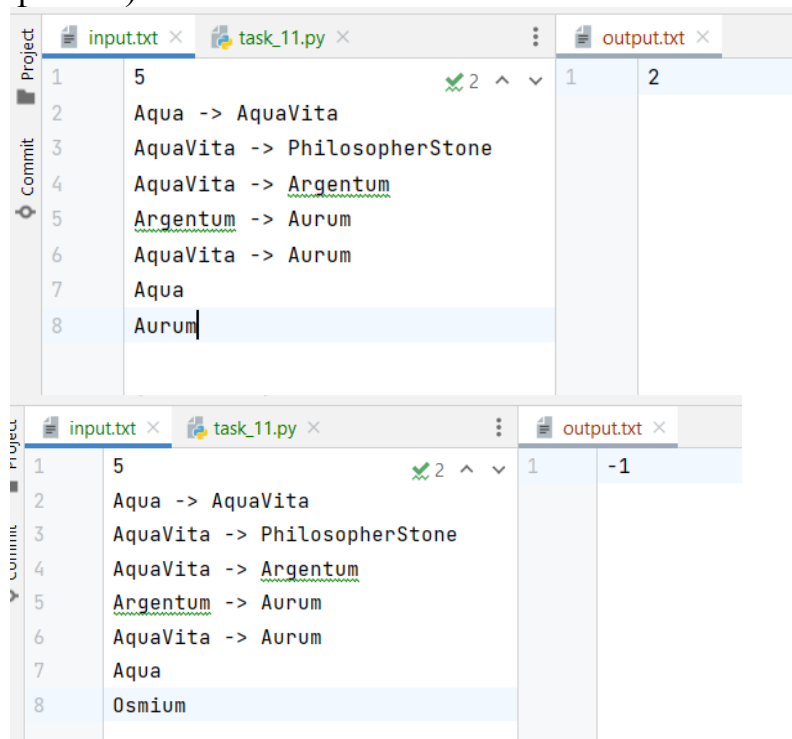
print(time.perf_counter() - t_start)
print(float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))

```

Текстовое объяснение решения.

Строим ориентированный граф. Начинаем обход графа, для этого создаем очередь. В очередь помещается название элемента и его «уровень» (у источника уровень = 0). Берем следующий элемент очереди. Если он – цель, то возвращаем его уровень, если нет, то добавляем в очередь его соседей следующего уровня. Если ни один не подошел, то возвращаем -1.

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



Проверка задачи на (openedu, астр и тд при наличии в задаче).

19571587	22.06.2023 14:04:24	Инженер Илья Ноутбуквич	0005	C#	Compiling			
19571586	22.06.2023 14:04:22	Шарифова Аише Алиевна	0368	Delphi	Compilation error			
19571585	22.06.2023 14:04:20	Бархатова Наталья Александровна	0743	Python	Accepted	0,062	2114 K6	
19571584	22.06.2023 14:04:16	Тен Илья Леонидович	1859	Python	Accepted	0,406	16 M6	
19571583	22.06.2023 14:03:51	Тен Илья Леонидович	1859	Python	Time limit exceeded	13	2,015	12 M6

	Время выполнения	Затраты памяти
--	------------------	----------------

Пример из задачи	0.0009	0.0044
Пример из задачи	0.0008	0.0045

Вывод по задаче: вспомнила те самые игры про алхимию, прослезилась.

Задача №12. Цветной лабиринт [2 балла]

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 $s_1 \dots s_k$. 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.

Листинг кода.

```
from collections import defaultdict
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
rooms, corridor = map(int, input_file.readline().split())
graph = defaultdict(list)
for i in range(corridor):
    from_room, to_room, color = input_file.readline().split()
    graph[from_room].append([to_room, color])
    graph[to_room].append([from_room, color])
moves_number = input_file.readline()
```

```

moves = list(map(str, input_file.readline().split()))
current = '1'
way = [current]
answer = "CORRECT"

for move in moves:
    for neighbor in graph[current]:
        if neighbor[1] == move:
            current = neighbor[0]
            way.append(current)
            break
    else:
        answer = "INCORRECT"
if answer == "CORRECT":
    answer = way[-1]
with open("output.txt", "w") as output_file:
    output_file.write(answer)

print(time.perf_counter() - t_start)
print(float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))

```

Текстовое объяснение решения.

Составляем граф. Проходимся по всем соседям и, если цвет коридора совпадает с нужным, совершаем переход. В список way записываются все комнаты на пути посетителя. Если в соседях ни нашлось ни одного подходящего коридора, то выводим ошибку.

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

Project		input.txt ×
1	3 2	
2	1 2 10	
3	1 3 5	
4	5	
5	10 10 10 10 5	
Commit		
		task_12.py × output.txt ×
1	3	

input.txt x

1

3 2

2

1 2 10

3

2 3 5

4

5

5

5 10 10 10 10

task_12.py x

output.txt x

1

INCORRECT

input.txt x

1

3 2

2

1 2 10

3

1 3 5

4

4

5

10 10 10 5

task_12.py x

output.txt x

1

INCORRECT

Проверка задачи на (openedu, астр и тд при наличии в задаче).

№1 - №20								Вперед »
ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19571809	22.06.2023 14:58:18	Бархатова Наталья Александровна	0601	Python	Accepted		0.171	14 Мб
19571808	22.06.2023 14:57:58	Зоинцов Аъзам Нуруллоевич	1088	C++	Accepted		0.03	432 Кб

	Время выполнения	Затраты памяти
Пример из задачи	0.0010	0.0039
Пример из задачи	0.0008	0.0039

Вывод по задаче: проверка с первого раза прошла, я в шоке

Задача №13. Грядки [3 балла]

Текст задачи.

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.

Листинг кода.

```
from collections import defaultdict
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()
input_file = open('input.txt')
N, M = map(int, input_file.readline().split())
graph = defaultdict(list)
field = []
for i in range(N):
    line = input_file.readline().strip()
    line = [c for c in line]
    field.append(line)

# for line in field:
#     print(line, end='\n')

counter = 1
new_field = []
for i in range(N):
    new_field.append([0] * M)

for line in range(N):
    for square in range(M):
        if field[line][square] == "#":
            graph[counter] = []
            new_field[line][square] = counter
            counter += 1
        if field[line][square] == "#":
            if new_field[line][square - 1] and square > 0:
```

```

graph[new_field[line][square -
1]].append(new_field[line][square])

graph[new_field[line][square]].append(new_field[line][square - 1])
    if new_field[line - 1][square] and line > 0:
        graph[new_field[line -
1][square]].append(new_field[line][square])
        graph[new_field[line][square]].append(new_field[line -
1][square])
for line in new_field:
    print(line, end='\n')
visited = [False] * (len(graph.keys()) + 1)
order = []
CC_nums = [0] * (len(graph.keys()) + 1)
cc_number = 1

def explore(v):
    CC_nums[v] = cc_number
    order.append(v)
    visited[v] = True
    for w in graph[v]:
        if not visited[w]:
            explore(w)

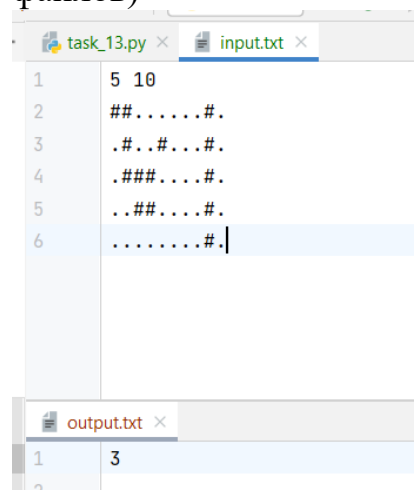
def DFS(graph, start):
    global cc_number
    for v in list(graph.keys())[start - 1:]:
        if not visited[v]:
            explore(v)
            cc_number += 1

DFS(graph, 1)
with open('output.txt', 'w') as output_file:
    print(max(CC_nums), file=output_file)
print(time.perf_counter() - t_start)
print(float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))

```

Текстовое объяснение решения.

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



```

task_13.py x input.txt x
1 5 10
2 ##.....#
3 .#..#...#
4 .###.....#
5 ..##.....#
6 .....#.|

output.txt x
1 3
2

```

```

task_13.py x input.txt x acmp_check.py x
1 5 10
2 ##.#####.
3 .#.#.#....
4 ###.##.##.
5 ..##.....#
6 .###.#####

output.txt x
1 5
2

```

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

Проверка задачи на (openedu, астр и тд при наличии в задаче).

№1 - №20									Вперед »
ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память	
19572332	22.06.2023 16:28:41	Бархатова Наталья Александровна	0432	Python	Runtime error	5	0.093	5042 Kб	
19572331	22.06.2023 16:28:33	Галимзянов Алмаз	0755	Python	Accepted		0.031	470 Kб	
19572330	22.06.2023 16:28:26	Шай-Тимур Рашидович	0713	C++	Accepted		0.02	422 Kб	

	Время выполнения	Затраты памяти
Пример из задачи	0.0013	0.0085
Пример из задачи	0.0012	0.0075

Вывод по задаче: Runtime error ☹️

Задача №14. Автобусы [3 балла]

Текст задачи.

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).

Листинг кода. (именно листинг, а не скрины)

```

import time
import tracemalloc

```

```

t_start = time.perf_counter()
tracemalloc.start()

input_file = open('input.txt')
village_count = int(input_file.readline())
buses = [[] for _ in range(village_count + 1)]
start, finish = map(int, input_file.readline().split())
trip_count = int(input_file.readline())
for i in range(trip_count):
    start_village, t1, finish_village, t2 = map(int,
input_file.readline().split())
    buses[start_village].append([t1, finish_village, t2])

inf = float('inf')
times = [inf] * (village_count + 1)
times[start] = 0
visited = [False] * (village_count + 1)
while True:
    min_time = inf
    for i in range(1, village_count + 1):
        if not visited[i] and times[i] < min_time:
            min_time = times[i]
            min_village = i
    if min_time == inf:
        break
    start_village = min_village
    visited[start_village] = True
    for t1, finish_village, t2 in buses[start_village]:
        if times[start_village] <= t1 and t2 <= times[finish_village]:
            times[finish_village] = t2

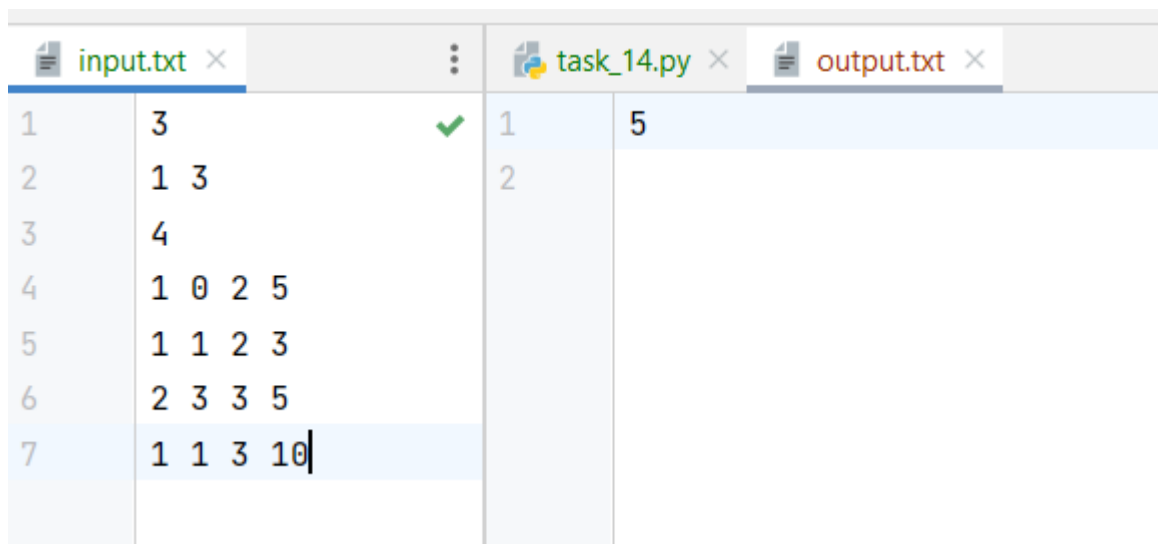
with open('output.txt', 'w') as output_file:
    if times[finish] == inf:
        print('-1', file=output_file)
    else:
        print(times[finish], file=output_file)
print(time.perf_counter() - t_start)
print(float(tracemalloc.get_tracemalloc_memory()) / (2 ** 20))

```

Текстовое объяснение решения.

Обозначаем стартовую деревню как `min_village`. Для каждого из этой деревни находим наименьшее время финиша. Затем смотрим на следующие деревни и находим минимальное время для них.

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



Проверка задачи на (openedu, астр и тд при наличии в задаче).

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19572460	22.06.2023 16:52:19	Зойцов Аязм Нуруллоевич	0970	C++	Compiling			
19572459	22.06.2023 16:52:17	Бархатова Наталья Александровна	0134	Python	Accepted		0.062	1690 Kб
19572458	22.06.2023 16:51:58	Голованов Алексей Михайлович	0777	Visual C++	Accepted		0.03	300 Kб
19572457	22.06.2023 16:51:51	Неизвестный	0148	C#	Runtime error	1		771 Kб

	Время выполнения	Затраты памяти
Пример из задачи	0.0008	0.0036

Вывод по задаче: я помогла Марии Ивановне доехать!

Вывод

В результате выполнения данной лабораторной работы я ознакомилась с основными принципами работы с графами, алгоритмами поиска путей в графах. Таким образом, выполнение данной лабораторной работы позволило получить полное представление об основных алгоритмах и структурах данных, используемых в работе с графами, что будет полезно при дальнейшем изучении и применении этой темы в практических задачах.