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 14

Выполнила:

Рудникова В.О.

K3143

Проверила:

Артамонова В.Е.

Санкт-Петербург 2022 г.

Содержание отчета

| Содержание отчета | 2 |
|--|----|
| Задачи по варианту | 3 |
| Задача №4. Порядок курсов [1 балл] | 3 |
| Задача №7. Двудольный граф [1.5 балла] | 6 |
| Задача №10. Оптимальный обмен валюты [2 балла] | 7 |
| Вывод: | 11 |

Задачи по варианту

Задача №4. Порядок курсов [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 п вершинами и m ребрами по формату 1.
- Ограничения на входные данные. $1 \le n \le 105, \ 0 \le m \le 105$. 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.

```
from time import process_time
from tracemalloc import start, get_traced_memory

def depth_first_search(u, visited, graph):
    visited[u] = 1
    for v in graph[u]:
    if not visited[v]:
    depth_first_search(v, visited, graph)
    stack.append(u)

def topological_sort(graph, visited):
    for v in range(len(graph)):
    if not visited[v]:
    depth_first_search(v, visited, graph)
```

```
if name == ' main ':
start()
with open('input.txt') as f:
n, m = f.readline().split()
graph = {key: set() for key in [i for i in
range(int(n))]}
for _ in range(int(m)):
u, v = map(int, f.readline().split())
graph[u - 1].add(v - 1)
visited = [0 for x in range(len(graph))]
stack = []
topological sort(graph, visited)
res = [str(x + 1) for x in stack]
with open('output.txt', 'w') as g:
g.write(' '.join(map(str, res[::-1])))
print('Time:', str(process time()), 'sec')
print('Memory usage:', str(get traced memory()[1]
1024), 'KB')
```

Текстовое объяснение решения: в задаче реализована топологическая сортировка графа с помощью поиска в глубину.

Результат работы кода на примерах из текста задачи:

Вывод по задаче: я научилась реализовывать топологическую сортировку.

Задача №7. Двудольный граф [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

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. $1 \le n \le 105, 0 \le m \le 105$.
- 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.

Текстовое объяснение решения: я обхожу граф поиском в ширину и присваиваю вершинам цвета, а затем проверяю, что на двух концах ребра вершины разного цвета.

Результат работы кода на примерах из текста задачи:

Вывод по задаче: я научилась проверять граф на двудольность.

Задача №10. Оптимальный обмен валюты [2 балла]

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

Дан ориентированный граф с возможными отрицательными весами ребер, у которого п 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. $1 \le n \le 103, \ 0 \le m \le 104, \ 1 \le s \le n$, вес каждого ребра целое число, не превосходящее по модулю 109.
- Формат вывода / выходного файла (output.txt). Для каждой вершины і графа от 1 до n выведите в каждой отдельной строке следующее:
- «*», если пути из s в і 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.

```
class Graph:

def __init__(self, vertices):
    self.v = vertices
    self.graph = []

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

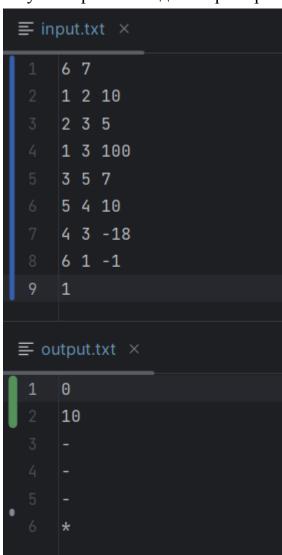
def output(self, lst, root):
    out = []
    for i in range(self.v):
    if lst[i] != float("inf"):
    if i == root:
    out.append('0')
    elif lst[i] == 0:
```

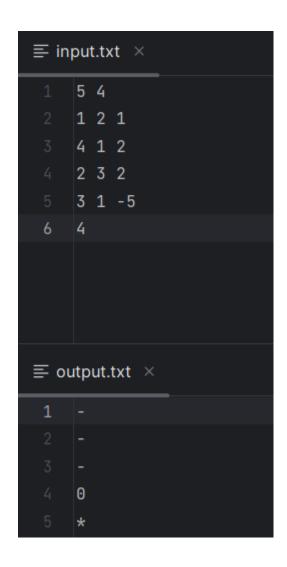
```
out.append("-")
else:
out.append(str(lst[i]))
else:
out.append("*")
return out
def BellmanFord(self, root):
lst = [float("inf")] * self.v
lst[root] = 0
for i in range(self.v - 1):
for u, v, w in self.graph:
if lst[u - 1] != float("inf") and lst[u - 1] + w <</pre>
lst[v - 1]:
lst[v - 1] = lst[u - 1] + w
for u, v, w in self.graph:
if lst[u - 1] != float("inf") and lst[u - 1] + w <</pre>
lst[v - 1]:
lst[u - 1] = 0
lst[v - 1] = 0
return self.output(lst, root)
if name == " main ":
with open("input.txt") as f:
inn = f.readlines()
n, m = map(int, inn[0].split())
gr = Graph(n)
root = int(inn[-1]) - 1
for i in range (1, m + 1):
u, v, w = map(int, inn[i].split())
gr.add(u, v, w)
with open('output.txt', 'w+') as g:
```

```
g.write('\n'.join([str(i) for i in
gr.BellmanFord(root)]))
```

Текстовое объяснение решения: в задаче я использую алгоритм Беллмана-Форда. Выбирается стартовая вершина, с которой начинается поиск (она называется root). Для каждой вершины проверяется, что расстояние до неё не бесконечно (то есть вершина достижима) и что оно не больше, чем кратчайшее (в этом случае присваивается новое кратчайшее расстояние).

Результат работы кода на примерах из текста задачи:





Вывод по задаче: я научилась искать кратчайшие расстояния до вершин с помощью алгоритма Беллмана-Форда.

Вывод:

Я поработала с графами, научилась обходить их в длину и ширину, а также узнала о некоторых интересных алгоритмах (топологическая сортировка, проверка на двудольность, поиск кратчайшего расстояния между вершинами).