МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»
Тема: Потоки в сети

Студент гр. 9383 _____ Звега А.Р.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Форда-Фалкерсона для нахождения максимального потока в сети.

Задание.

Найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм ФордаФалкерсона.

Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа – пропускной способности (веса).

Входные данные:

N - количество ориентированных рёбер графа

v0 - исток

vn - сток

vi,vj,ωij - ребро графа

vi,vj,ωij - ребро графа

...

Выходные данные:

Pmax - величина максимального потока

vi,vj,ωij - ребро графа с фактической величиной протекающего потока

vi,vj,ωij - ребро графа с фактической величиной протекающего потока

...

В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все указанные входные рёбра, даже если поток в них равен 0).

Задание.(Вариант 6)

Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего к друг другу.

Выполнение работы.

Был реализован алгоритм Форда-Фалкерсона, описание алгоритма:

1) Выполняется глубокое копирование основного графа (отличается от обычного тем, что создается новый объект, который заполняется копиями объектов, а не ссылками).

2) Ищется путь от истока в сток. (исток начальная вершина, сток конечная).

3) Происходит поиск минимальной пропускной способности на этом пути.

4) Затем вычитается минимальная пропускная способность для каждого ребра на найденном пути и добавляется пропускная способность для ребер на найденном пути.

5) Затем алгоритм переходит на шаг 2, пока не найдет весь поток в сети.

Особенность реализации индивидуального задания заключается в том, что идет другой набор путей, но на корректность нахождения потока это не влияет, так как в любом случае будут перебраны все пути.

Анализ алгоритма.

Так как поиск пути в сети с n количеством ребер происходит за O(n), а сам алгоритм для сети с максимальным потоком m сходится за O(m) или меньше(т.к пропускная способность ребер на пути уменьшается), следовательно время работы алгоритма O(n*m), а для индивидуального задания сложность не меняется так как, меняется только набор путей.

Описание основных функций.

reed_graph(number_of_edges) – чтение графа.

max_flow(graph, start, end) – поиск максимального потока в графе.

find_path(graph, start, end) – поиск пути в графе и его минимальной пропускной способности.

Таблица 1. Результаты тестирования алгоритма

Ввод	Вывод
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
5 a b a b 14 b a 9 b c 2 a c 4 c b 15	18 a b 14 a c 4 b a 0 b c 0 c b 4

Выводы.

При выполнении работы был изучен и реализован алгоритм Форда-Фалкерсона для нахождения максимального потока в сети.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД.

Название файла lab3.py

```
import copy
import sys


def find_path(flow_graph, current, end, path='', visited=None,
min_weight=sys.maxsize):
    if visited is None:
        visited = []
    if current == end:
        return (path, min_weight)
    visited.append(current)
    for vertex in flow_graph[current]:
        if vertex not in visited and flow_graph[current][vertex] > 0:
            new_path, new_min_weight = find_path(flow_graph, vertex, end,
path+vertex, visited,

min(flow_graph[current][vertex], min_weight))
            if new_path:
                return (new_path, new_min_weight)
    return ('', min_weight)


def max_flow(graph, start, end):
    flow_graph = copy.deepcopy(graph)
    path, min_weight = find_path(flow_graph, start, end)
    while path:
```

```
        for u, v in zip((start+path)[:-1], (start+path)[1:]):
            flow_graph[u][v] -= min_weight
            if u not in flow_graph[v]:
                flow_graph[v][u] = min_weight
            else:
                flow_graph[v][u] += min_weight
        path, min_weight = find_path(flow_graph, start, end)
    return sum([graph[start][edge]-flow_graph[start][edge] for edge in flow_graph[start]]), flow_graph


def reed_graph(number_of_edges):
    graph = {}
    for i in range(number_of_edges):
        v1, v2, weight = input().split()
        if v1 in graph:
            graph[v1][v2] = int(weight)
        else:
            graph[v1] = {v2: int(weight)}
        if v2 not in graph:
            graph[v2] = {}

    graph = dict(sorted(graph.items()))
    for i in graph:
        graph[i] = dict(sorted(graph[i].items()))
    return graph


if __name__ == '__main__':
```

```
number_of_edges = int(input())
start = input()
end = input()
graph = reed_graph(number_of_edges)

flow_value, flow_graph = max_flow(graph, start, end)
print(flow_value)

for v1 in graph:
    for v2 in graph[v1]:
        if graph[v1][v2] - flow_graph[v1][v2] > 0:
            print(v1, v2, graph[v1][v2] - flow_graph[v1][v2])
        else:
            print(v1, v2, 0)
```