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: Максимальный поток

Студент гр. 0382	____________________	Кондратов Ю.А.
Преподаватель	____________________	Шевская Н.В.

Санкт-Петербург
2022

Цель работы.

Написать программу, реализовывающую поиск максимального потока во взвешенном графе, используя жадный алгоритм Форда-Фалкерсона.

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).

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 друг к другу. Если таких дуг несколько, то выбрать ту, имя конца которой в алфавите ближайшее к началу алфавита.

Выполнение работы.

Исходный код модуля представлен в приложении А.

При выполнении данной работы было принято решение хранить граф в виде списка инцидентности, реализованного при помощи словарей.

Алгоритм Форда-Фалкерсона использует функцию поиска пути find_path. В данной функции сначала при помощи deepcopy() копируется переданный граф, далее инициализируется очередь с приоритетом p_queue, в которою будут записываться рёбра, а точнее кортеж из 4 значений: (разности ascii кодов обозначающих вершины ребра символов, обозначающий конец ребра символ, пройденный до начала ребра путь, минимальный вес ребра в этом пути).

Таким образом, при вставке очередного элемента в кучу, происходит сравнение сначала по расстоянию между acii кодами, потом по близости конца ребра к началу алфавита, что соответствует требованиям задания.

Сначала в кучу вставляются все рёбра, инцидентные начальной вершине, далее, в цикле while, до тех пор, пока куча не пуста происходит следующее: достаётся очередное ребро из кучи, если вес ребра не строго положителен, то ребра рассматривать не имеет смысла – продолжаем цикл, если конец ребра является стоком, то путь найден – возвращаем путь и вес минимального ребра в нём. Далее рассматриваются все ребра, инцидентные концу текущего ребра, они добавляются в кучу вместе с вычисленными дополнительными параметрами. В конце каждого шага ребра, рассмотренные на этом шаге, удаляются из графа во избежание повторений.

Сам алгоритм Форда-Фалкерсона реализован следующим образом: находим путь при помощи функции find_path, из весов всех ребер, через которые пущен поток, вычитает величину потока, к весам всех ребер обратных к тем, через которые пущен поток, прибавляет величину потока. К величине максимального потока прибавляем величину найденного на этом шаге, ищем новый путь и новый поток. Если пути нет, значит максимальный поток найден – возвращаем остаточную сеть и величину максимального потока.

Оценка сложности алгоритмов.

1. Поиск пути.

В данной реализации поиска, несмотря на цикл for, вложенный в цикл while, производится в общей сложности O(E) итераций (E – количество рёбер), так как на каждой итерации цикла while из кучи достаётся ровно одно ребро, а никакое из ребер в кучу дважды не добавляется, так как сразу после добавления удаляется из графа. Учитывая сложность вставки и удаление ребра из кучи, получаем сложность O(logE*E).

2.Форда-Фалкерсона.

Так как все числа целые, то алгоритм Форда-Фалкерсона сходится не более чем за f шагов, где f – величина максимального потока, на каждом шаге выполняется поиск пути (за O(logE*E)) и проход по всем ребрам за O(E). В итоге получаем сложность O(logE*E*f).

Тестирование.

Тестирование производилось при помощи библиотеки pytest.

Рассмотренные в тестировании случаи представлены в таблице 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные	Описание	Вердикт
7 a f a b 7 a c 6 b d 6 c f 9 d e 3	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2	Тест из задания.	passed

d f 4 e c 2			
4 b c a b 5 b a 7 a c 3 a c 9	7 a b 0 a c 7 b a 7	Проверка корректности работы алгоритма при наличии двойных ребер.	passed
7 a f a b 7 a c 6 b d 6 c f 6 d e 3 d f 4 e c 2	10 a b 4 a c 6 b d 4 c f 6 d e 0 d f 4 e c 0	Тест для проверки корректности работы алгоритма при наличии ребёр, по которым пущен нулевой поток.	passed
7 a d a b 4 a e 7 b f 4 b c 5 c d 4 e c 3 f d 9	7 a b 4 a e 3 b c 1 b f 3 c d 4 e c 3 f d 3	Тест для проверки правильной обработки остаточных ребер. В данном случае по ребру bc пропускается сначала поток 4 в одну сторону, а потом поток 3 в другую сторону. В итоге остаётся 1.	passed

8 a f a b 1000 a c 1000 b d 1 b e 1 c d 1 c e 1 e f 1 d f 1	2 a b 0 a c 2 b d 0 b e 0 c d 1 c e 1 d f 1 e f 1	Тест для проверки корректной работы алгоритма при наличие тяжёлых ребер, но маленьком максимальном потоке.	passed

Протокол тестирования представлен на рисунке 1.

```
C:\LETI\DAA\lab3>pytest -v
================================== test session starts ===================================
platform win32 -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0 -- C:\Users\Юрий\AppData\Loc
al\Programs\Python\Python310\python.exe
cachedir: .pytest_cache
rootdir: C:\LETI\DAA\lab3
collected 5 items

src/test_ford_fulkerson.py::test_ford_fulkerson[test_data0-result0] PASSED       [ 20%]
src/test_ford_fulkerson.py::test_ford_fulkerson[test_data1-result1] PASSED       [ 40%]
src/test_ford_fulkerson.py::test_ford_fulkerson[test_data2-result2] PASSED       [ 60%]
src/test_ford_fulkerson.py::test_ford_fulkerson[test_data3-result3] PASSED       [ 80%]
src/test_ford_fulkerson.py::test_ford_fulkerson[test_data4-result4] PASSED       [100%]

================================== 5 passed in 0.03s ===================================
```

Рисунок 1 – Протокол тестирования

Выводы.

В результате выполнения работы был изучен алгоритм поиска максимального поток в ориентированном взвешенном графе с целыми неотрицательными весами рёбер – алгоритм Форад-Фалкерсона. Был использован особый алгоритм поиска, в связи с чем сложность алгоритма увеличилась и из O(E*f) превратилась в O(logE*E*f).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: ford_fulkerson.py

```
from heapq import *
from copy import deepcopy


def read():
    graph = {}
    n = int(input())
    start = input()
    end = input()
    for i in range(n):
        s, e, w = input().split()
        w = int(w)
        if s in graph:
            graph[s][e] = w
        else:
            graph[s] = {e: w}
        if e not in graph:
            graph[e] = {}
    return graph, start, end


def find_path(graph, start, end):
    graph = deepcopy(graph)
    p_queue = []

    for adj in graph[start]:
        dist = abs(ord(start) - ord(adj))
        heappush(p_queue, (dist, adj, start, graph[start][adj]))
    graph[start] = {}

    while p_queue:
        edge = heappop(p_queue)
        if edge[3] <= 0:
            continue
        if edge[1] == end:
            return edge[2] + edge[1], edge[3]
        for adj in graph[edge[1]]:
            dist = abs(ord(adj) - ord(edge[1]))
            path = edge[2] + edge[1]
            min_weight = min(edge[3], graph[edge[1]][adj])
            heappush(p_queue, (dist, adj, path, min_weight))
        graph[edge[1]] = {}

    return "", 0


def ford_fulkerson(graph, start, end):
    graph = deepcopy(graph)
    path, flow = find_path(graph, start, end)
    max_flow = flow
    while path:
        for s, e in zip(path[:-1], path[1:]):
```

```
            graph[s][e] -= flow
            if s in graph[e]:
                graph[e][s] += flow
            else:
                graph[e][s] = flow
        path, flow = find_path(graph, start, end)
        max_flow += flow
    return max_flow, graph


def main():
    graph, start, end = read()
    flow, residual_graph = ford_fulkerson(graph, start, end)

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

    print(flow)
    for s in graph:
        for e in graph[s]:
            cur_flow = graph[s][e] - residual_graph[s][e]
            print(s, e, cur_flow if cur_flow > 0 else 0)


if __name__ == "__main__":
    main()
```