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: Алгоритмы поиска максимального потока в графе

Студент гр. 9303 _____ Павлов Д.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2021

Цель работы.

Ознакомиться с алгоритмами поиска максимального в ориентированном графе. Реализовать алгоритм Форда-Фалкерсона поиска максимального потока с выбранной индивидуализацией.

Задание.

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

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

$v_i\ v_j\ w_{ij}$- ребро графа

…

Выходные данные:

P_{max} – величина максимального потока

$v_i\ v_j\ w_{ij}$- 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).

Выполнение работы.

Описание классов и функций:

`Edge` – класс ребра

Описание методов класса `Edge`:

`__init__(self, u, v, w)` – конструктор класса.

`__repr__(self)` – для отображения объекта.

`FlowNetwork` – класс потока сети

Описание методов класса `FlowNetwork`:

`__init__(self)` – конструктор класса.

`add_vertex(self, vertex)` – добавление вершины.

`add_edge(self, u, v, w=0)` – добавление ребра.

`get_edges(self, v)` – получить ребра.

`find_path(self, source, sink, path)` – поиск пути.

`max_flow(self, source, sink)` – подсчет максимального потока.

`get_result(self)` – получить список вершин по которым был посчитан максимальный поток.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Был успешно реализован алгоритм Форда-Фалкерсона для нахождения максимального потока в графе.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from sys import argv, stdin

from icecream import ic


class Edge(object):
    def __init__(self, u, v, w):
        self.source = u
        self.sink = v
        self.capacity = w

    def __repr__(self):
        return "%s %s:%s" % (self.source, self.sink, self.capacity)


class FlowNetwork(object):
    def __init__(self):
        self.founded = []
        self.adj = {}
        self.flow = {}

    def add_vertex(self, vertex):
        self.adj[vertex] = []

    def get_edges(self, v):
        ic(v)
        return self.adj[v]

    def add_edge(self, u, v, w=0):
        if u == v:
            raise ValueError("u == v")
        edge = Edge(u, v, w)
        redge = Edge(v, u, 0)
        edge.redge = redge
        redge.redge = edge
        self.adj[u].append(edge)
        self.adj[v].append(redge)
        self.flow[edge] = 0
        self.flow[redge] = 0

    def find_path(self, source, sink, path):
        if source == sink:
            return path
        for edge in self.get_edges(source):
            ic(edge)
            residual = edge.capacity - self.flow[edge]
            if residual > 0 and edge not in path:
                result = self.find_path(edge.sink, sink, path + [edge])
                if result is not None:
                    return result

    def max_flow(self, source, sink):
        path = self.find_path(source, sink, [])
```

```
        while path is not None:
            residuals = [edge.capacity - self.flow[edge] for edge in path]
            flow = min(residuals)
            for edge in path:
                self.flow[edge] += flow
                self.flow[edge.redge] -= flow
            path = self.find_path(source, sink, [])
        return sum(self.flow[edge] for edge in self.get_edges(source))

    def get_result(self):
        new_dict = {}
        for i in self.flow:
            if self.flow[i] >= 0:
                name = i.__repr__().split(":")[0]
                value = self.flow[i]
                if (name in new_dict and value > new_dict[name]) or (name not in
new_dict) and name in self.founded:
                    new_dict[name] = value

        res = []
        for i in new_dict.keys():
            res.append(f'{i} {new_dict[i]}')
        return sorted(res)


def main():
    g = FlowNetwork()
    vers = []
    N = int(input())
    v0 = input()
    vn = input()
    strlist = []
    [strlist.append(line) for line in stdin]
    s = "".join(strlist).splitlines()
    for i in s:
        vers.append(i[0])
        vers.append(i[2])
        g.founded.append("".join(i[0] + ' ' + i[2]))
    vers = list(set(vers))
    [g.add_vertex(v) for v in vers]
    for i in s:
        g.add_edge(i.split(' ')[0], i.split(' ')[1], int(i.split(' ')[2]))

    print(g.max_flow(v0, vn))
    res = g.get_result()
    for i in res:
        print(i)


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

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица 1 - Примеры тестовых случаев.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	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	Программа работает корректно
2.	8 1 4 1 2 7 1 3 6 2 4 8 2 5 1 3 5 2 3 6 4 6 5 7 5 4 6	13 1 2 7 1 3 6 2 4 7 2 5 0 3 5 2 3 6 4 5 4 6 6 5 4	Программа работает корректно