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. 9382 _____ Павлов Р.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы

Ознакомление с алгоритмом поиска максимального потока в сети, реализация алгоритма.

Задание

Вар. 5. Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность. Если таких дуг несколько, то выбрать ту, которая была обнаружена раньше в текущем поиске пути.

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
v_0 - исток
v_n - сток
v_i v_j ω_{ij} - ребро графа
v_i v_j ω_{ij} - ребро графа
...

Выходные данные:
P_{max} - величина максимального потока
v_i v_j ω_{ij} - ребро графа с фактической величиной протекающего потока
v_i v_j ω_{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).

Sample Input:

```
7
a
f
a b 7
a c 6
b d 6
```

```
c f 9
d e 3
d f 4
e c 2
```

Sample Output:

```
12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2
```

Описание алгоритма

Алгоритм Форда-Фалкерсона.

1. Находится начальная вершина (исток), она помечается как посещённая и выбирается в качестве текущей.

2. Если текущая вершина – сток, то максимальный в ведущем к нему пути поток вычитается из всех пропускных способностей рёбер, из которых этот путь состоит. При этом та же самая величина прибавляется к пропускным способностям противоположно направленных рёбер (возникает обратная пропускная способность). Выполняется переход к шагу 1.

Иначе рассматриваются инцидентные текущей вершины, из них выбираются те, пропускная способность в направлении которых положительна и которые ещё не были посещены в уже пройденном пути (или от которых не был выполнен возврат), они добавляются в множество вершин-кандидатов на посещение. Иными словами, в список кандидатов попадают те вершины, к которым возможно перейти по тем рёбрам, которые ещё не были рассмотрены на текущей итерации.

3. Из множества кандидатов по определённым критериям (зависит от порядка рассмотрения) выбирается наиболее приоритетная вершина. Если поток, который можно пропустить через выбранную вершину, меньше, чем поток, пропущенный через пройденный путь, то величина потока всего пути

уменьшается до значения первого. На этом шаге возможны три варианта действий:

a. Кандидатов нет, вершина – исток: завершить работу алгоритма, не существует увеличивающего пути.

b. Кандидатов нет, вершина – не исток: вернуться к предыдущей вершине, заблокировать последнее пройденное ребро, перейти к шагу 2.

c. Выбран приоритетный кандидат в качестве текущей вершины: перейти к шагу 2.

Сложность алгоритма

Сложность по памяти.

Алгоритм во время выполнения запоминает рёбра, по которым уже проходил, и количество хранящихся в памяти рёбер не превысит **e**, где **e** – количество всех рёбер в графе. Поэтому сложность алгоритма по памяти составляет O(**e**).

Сложность по времени.

Количество рассматриваемых на каждой итерации рёбер также не превысит **e**, а количество итераций не превысит **f** – максимальный поток через сеть. Если всегда увеличивать максимальный поток на единицу, то максимальное количество итераций составит **f**. Таким образом, сложность алгоритма по времени выполнения – O(**f*e**).

Описание функций и СД

Класс, хранящий граф:

class Graph:

numberOfEdges – число рёбер, подаваемых на вход

source – имя вершины-источника

drain – имя вершины-стока

edges – словарь рёбер с ключами $\mathbf{v_1}$, значение также представляет собой словарь, ключами $\mathbf{v_2}$ которого являются смежные вершины, а значениями **w** –

пропускные способности рёбер **(v_1, v_2).**

edgeList – список исходных рёбер для вывода

accFlow – суммарный (максимальный) поток в графе

Методы класса Graph:

__init__(self) – конструктор класса, создание графа происходит в нём.

out(self) – вывод максимального потока и фактических величин потоков изначально указанных рёбер.

solve(self) – итеративная реализация алгоритма Форда-Фалкерсона. Записывает результаты в поля класса графа.

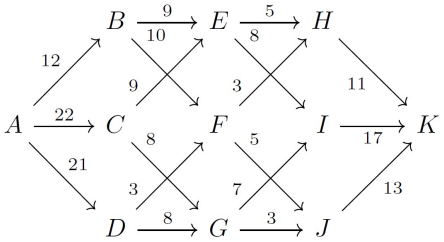
Тестирование

Результаты тестирования представлены в таблице 1.

Таблица 1

Входные данные	**Выходные данные**
8 a f a b 3 a c 1 b d 2 b e 5 c d 5 d e 2 d f 2 e f 1	3 a b 2 a c 1 b d 2 b e 0 c d 1 d e 1 d f 2 e f 1
5 a d a b 3 a c 4 b c 2 b d 3 c d 1	4 a b 3 a c 1 b c 0 b d 3 c d 1
8 a g a b 3 a d 5 b c 4 c d 3 d e 4 d f 3 e g 2 f g 2	4 a b 0 a d 4 b c 0 c d 0 d e 2 d f 2 e g 2 f g 2

10. Найдите максимальный поток через данную сеть:



18
a
k
a b 12
a c 22
a d 21
b e 9
b f 10
c e 9
c g 8
d f 3
d g 8
e h 5
e i 8
f h 3
f j 5
g i 7
g j 3
h k 11
i k 17
j k 13

31
a b 10
a c 11
a d 10
b e 5
b f 5
c e 8
c g 3
d f 3
d g 7
e h 5
e i 8
f h 3
f j 5
g i 7
g j 3
h k 8
i k 15
j k 8

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

Имя файла: main.py

```
from math import inf

class Graph:
    def __init__(self):
        self.numberOfEdges = int(input())
        self.source = input()
        self.drain = input()
        self.edges = {} # рёбра
        self.edgeList = [] # для вывода
        self.accFlow = 0 # суммарный поток
        for i in range(self.numberOfEdges):
            edge = input().split()
            key, target, capacity = edge[0], edge[1], int(edge[2])
            print("В граф добавлено ребро ", key, target, capacity)
            self.edgeList.append((key, target, capacity))

            # Рёбра в исток и из стока не добавляются
            if key != self.drain and target != self.source:
                if key not in self.edges.keys():
                    self.edges[key] = {}
                if target not in self.edges.keys():
                    self.edges[target] = {}
                if key not in self.edges[target].keys():
                    self.edges[target][key] = 0
                if target not in self.edges[key].keys():
                    self.edges[key][target] = capacity
                else:
                    self.edges[key][target] += capacity
        self.edgeList.sort(key=lambda x: (x[0], x[1]))
        print ("\nСостояние исходных рёбер в начале выполнения:")
        for i in self.edgeList:
            print(i[0], i[1], i[2])
        print("\n")

    def out(self):
        print(self.accFlow)
        for e in self.edgeList:
            finalFlow = e[2] - self.edges[e[0]][e[1]]
            if finalFlow < 0:
                finalFlow = 0
            print(e[0], e[1], finalFlow)

    def solve(self):
        openList, closedList = [[0, self.source, "", inf, 0]], []

        # Когда из истока уже некуда идти, цикл заканчивается
        while openList:
            print("Из открытого списка:", openList, "была взята вершина",
end=" ")
            cur = openList.pop()
            print(cur)
            print("Её дистанция: ", cur[0], "; возраст: ", cur[4],
```

```
                  ", а также она является самой старшей по алфавиту при
прочих подобных вершинах")

            # Если дошли до стока, строим путь по строке с предыдущими
вершинами, возвращаемся в начальное состояние
            # При этом к суммарному потоку прибавляется максимально
возможный поток в этом пути
            if cur[1] == self.drain:
                print(cur[1], "- это сток, запись результата и сброс в
начальное состояние")
                path = cur[2] + cur[1]
                print("Поток в пути", " -> ".join(path), "уменьшен на",
cur[3], end = "\n\n")
                for i in range(len(path) - 1):
                    self.edges[path[i]][path[i+1]] -= cur[3]
                    self.edges[path[i+1]][path[i]] += cur[3]
                self.accFlow += cur[3]
                closedList.clear()
                openList = [[0, self.source, "", inf, 0]]
                continue
            closedList.append(cur[1])

            # Если вершина уже была пройдена на этом пути, её не посещаем
            toOpen = list(filter(lambda x: x[0] not in closedList and
x[1] > 0, self.edges[cur[1]].items()))
            maxFlow = cur[3]

            # Добавляем все доступные вершины в открытый список
            print("Вершины, доступные для посещения:")
            for i in toOpen:
                print("\t", i[0], ", расстояние: ", i[1], sep="")
                if i[1] < maxFlow:
                    maxFlow = i[1]
                openList.append([i[1], i[0], cur[2]+cur[1], maxFlow, 0])
                maxFlow = cur[3]
            # Увеличиваем "возраст" вершин в списке, сортируем (подобно
очереди с приоритетом)
            for i in openList:
                i[4] += 1
            openList.sort(key=lambda x: (x[0], x[4], x[2]))
            print("Вершины были добавлены в открытый список\n")
        print("\nПоиск закончен, завершение работы\n")

gra = Graph()
gra.solve()
gra.out()
print("\nВыполнение завершено. Введите любой символ, чтобы выйти из
программы.")
input()
```