

**МИНОБРНАУКИ РОССИИ**  
**Санкт-Петербургский государственный**  
**электротехнический университет**  
**«ЛЭТИ» им. В.И. Ульянова (Ленина)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №3**  
**по дисциплине «Построение и анализ алгоритмов»**  
**Тема: Потоки в сети.**

Студент гр. 8382

Терехов А.Е.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Изучить алгоритм поиска максимального потока в сети Форда-Фалкерсона.

### **Задание.**

Индивидуализация задания: Вариант 4. Поиск в глубину. Итеративная реализация.

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

$v_i v_j w_{ij}$  – 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).

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

### **Ход работы.**

### **Описание структур.**

Для алгоритма было выбрано хранение графа с использованием списков смежности. Списки смежности представляют собой словарь, ключи которого это вершины, из которых существует хоть один путь. Значение – это тоже словарь, но в нем ключи – это концы дуг, а значение – пропускная способность. То есть это выглядит следующим образом:

{ начало\_дуги : { конец\_дуги : стоимость } }.

Также для хранения потоков был использован инвертированный граф, который устроен схожим образом.

Листинг 1 – чтение графа в словарь.

```
graph = dict()
flows = dict()
for _ in range(n):
    u, v, c = input().split()
    if u in graph.keys():
        graph[u[0]][v[0]] = int(c)
```

```

else:
    graph[u[0]] = {v[0]: int(c)}
if v in flows.keys():
    flows[v[0]][u[0]] = 0
else:
    flows[v[0]] = {u[0]: 0}

```

### **Описание алгоритма.**

Алгоритм Форда-Фалкерсона решает задачу поиска максимального потока в сети.

Изначально считается, что по всем дугам поток нулевой. На каждом шаге ищется путь из стока в сток, способный увеличить поток. На найденном пути выбирается дуга с минимальной пропускной способностью, и пропускные способности каждой дуги этого пути уменьшается на данную величину, а поток каждой дуги увеличивается на данную величину. Естественно дуги, у которых нулевая пропускная способность не могут увеличить искомый поток, и находимые пути их не содержат.

В случае если невозможно найти естественный путь, то имеет место поиск чередующегося пути. Это последовательность вершин, в которой не обязательно вершина, стоящая на  $i$ -ом месте, является началом дуги, а вершина на  $(i+1)$ -ом месте концом этой же дуги. То есть, продвигаясь по пути, движение происходит не только в направлении дуги, но и в противоположном. Пропускные способности уменьшаются на минимум пропускной способности, а потоки дуг увеличиваются для дуг, движение по которым происходит в естественном направлении, и наоборот для дуг, движение по которым происходит в противоположном направлении, то есть пропускные способности увеличиваются, а потоки уменьшаются.

После основного цикла при существовании двунаправленных ребер возможны ситуации, когда существуют два ребра  $(u \rightarrow v)$  и  $(v \rightarrow u)$  с ненулевыми потоками. Это, с одной стороны, допустимо так как максимальный поток найден верно, но на степике данный ответ не проходил, поэтому в коде добавлен дополнительный цикл, работающий за квадрат обновляя эти потоки так, чтобы

меньшая из этих дуг стала иметь нулевой поток, а другая поток, уменьшенный на поток меньшей.

Для того, чтобы получить величину максимального потока сети достаточно сложить все потоки входящие в сток, то есть сделать разрез между стоком и остальным графом и посчитать его пропускную способность.

### Сложность алгоритма.

Программа требует памяти памяти  $2(|E| + |V|)$ , так как хранятся два графа – один с пропускными способностями, другой инвертированный с потоками.

На каждом шаге алгоритм добавляет поток увеличивающего пути к уже имеющемуся потоку. Так как пропускные способности ребер – целые числа, на каждом шаге алгоритм увеличивает поток по крайней мере на единицу, следовательно, он сойдётся не более чем за  $O(f)$  шагов, где  $f$  — максимальный поток в графе. Можно выполнить каждый шаг за время  $O(E)$ , где  $E$  — число рёбер в графе, тогда общее время работы алгоритма ограничено  $O(Ef)$ .

### Тестирование.

Тестирование программы представлено в таблице 1.

Таблица 1. Тестирование.

Input	Output
3 a c a b 700 a c 600 b c 400	Graph: {'a': {'b': 700, 'c': 600}, 'b': {'c': 400}} Flows: {'b': {'a': 0}, 'c': {'a': 0, 'b': 0}} Neighbours of a : ['b', 'c'] Choose node b Path: ['a', 'b'] Neighbours of b : ['c'] Changeable neighbours of b : ['c'] Choose node c Path: ['a', 'b', 'c'] Find path: ['a', 'b', 'c'] Minimal flow: 400 Bandwidth <a b> increased by 400 Bandwidth <b c> increased by 400 Update Graph: {'a': {'b': 300, 'c': 600}, 'b': {'c': 0}} Update Flows: {'b': {'a': 400}, 'c': {'a': 0, 'b': 400}} Neighbours of a : ['b', 'c']

Таблица 1. Тестирование.

	<p>Choose node b  Path: ['a', 'b']  Neighbours of b : []  Changeable neighbours of b : []  b is dead end  Neighbours of a : ['c']  Choose node c  Path: ['a', 'c']  Find path: ['a', 'c']  Minimal flow: 600  Bandwidth &lt;a c&gt; increased by 600  Update Graph: {'a': {'b': 300, 'c': 0}, 'b': {'c': 0}}  Update Flows: {'b': {'a': 400}, 'c': {'a': 600, 'b': 400}}  Neighbours of a : ['b']  Choose node b  Path: ['a', 'b']  Neighbours of b : []  Changeable neighbours of b : []  b is dead end  Neighbours of a : []  Can't find changeable path  Find path: []  1000  a b 400  a c 600  b c 400</p>
8 A D A B 7 A C 4 B D 8 B E 2 C E 5 C F 3 F E 7 E D 5	<p>Graph: {'A': {'B': 7, 'C': 4}, 'B': {'D': 8, 'E': 2}, 'C': {'E': 5, 'F': 3}, 'F': {'E': 7}, 'E': {'D': 5}}  Flows: {'B': {'A': 0}, 'C': {'A': 0}, 'D': {'B': 0, 'E': 0}, 'E': {'B': 0, 'C': 0, 'F': 0}, 'F': {'C': 0}}  Neighbours of A : ['B', 'C']  Choose node B  Path: ['A', 'B']  Neighbours of B : ['D', 'E']  Changeable neighbours of B : ['D', 'E']  Choose node D  Path: ['A', 'B', 'D']  Find path: ['A', 'B', 'D']  Minimal flow: 7  Bandwidth &lt;A B&gt; increased by 7  Bandwidth &lt;B D&gt; increased by 7  Update Graph: {'A': {'B': 0, 'C': 4}, 'B': {'D': 1, 'E': 2}, 'C': {'E': 5, 'F': 3}, 'F': {'E': 7}, 'E': {'D': 5}}  Update Flows: {'B': {'A': 7}, 'C': {'A': 0}, 'D': {'B': 7, 'E': 0}, 'E': {'B': 0, 'C': 0, 'F': 0}, 'F': {'C': 0}}  Neighbours of A : ['C']  Choose node C</p>

Таблица 1. Тестирование.

	Path: ['A', 'C'] Neighbours of C : ['E', 'F'] Changeable neighbours of C : ['E', 'F'] Choose node E Path: ['A', 'C', 'E'] Neighbours of E : ['D'] Changeable neighbours of E : ['D'] Choose node D Path: ['A', 'C', 'E', 'D'] Find path: ['A', 'C', 'E', 'D'] Minimal flow: 4 Bandwidth <A C> increased by 4 Bandwidth <C E> increased by 4 Bandwidth <E D> increased by 4 Update Graph: {'A': {'B': 0, 'C': 0}, 'B': {'D': 1, 'E': 2}, 'C': {'E': 1, 'F': 3}, 'F': {'E': 7}, 'E': {'D': 1}} Update Flows: {'B': {'A': 7}, 'C': {'A': 4}, 'D': {'B': 7, 'E': 4}, 'E': {'B': 0, 'C': 4, 'F': 0}, 'F': {'C': 0}} Neighbours of A : [] Can't find changeable path Find path: [] 11 A B 7 A C 4 B D 7 B E 0 C E 4 C F 0 E D 4 F E 0
7 A F A B 14 D E 6 A C 12 D F 8 B D 12 E C 4 C F 18	Graph: {'A': {'B': 14, 'C': 12}, 'D': {'E': 6, 'F': 8}, 'B': {'D': 12}, 'E': {'C': 4}, 'C': {'F': 18}} Flows: {'B': {'A': 0}, 'E': {'D': 0}, 'C': {'A': 0, 'E': 0}, 'F': {'D': 0, 'C': 0}, 'D': {'B': 0}} Neighbours of A : ['B', 'C'] Choose node B Path: ['A', 'B'] Neighbours of B : ['D'] Changeable neighbours of B : ['D'] Choose node D Path: ['A', 'B', 'D'] Neighbours of D : ['E', 'F'] Changeable neighbours of D : ['E', 'F'] Choose node E Path: ['A', 'B', 'D', 'E'] Neighbours of E : ['C'] Changeable neighbours of E : ['C'] Choose node C

Таблица 1. Тестирование.

```

Path: ['A', 'B', 'D', 'E', 'C']
Neighbours of C : ['F']
Changeable neighbours of C : ['F']
Choose node F
Path: ['A', 'B', 'D', 'E', 'C', 'F']
Find path: ['A', 'B', 'D', 'E', 'C', 'F']
Minimal flow: 4
Bandwidth <A B> increased by 4
Bandwidth <B D> increased by 4
Bandwidth <D E> increased by 4
Bandwidth <E C> increased by 4
Bandwidth <C F> increased by 4
Update Graph: {'A': {'B': 10, 'C': 12}, 'D': {'E': 2, 'F': 8}, 'B': {'D': 8}, 'E': {'C': 0}, 'C': {'F': 14}}
Update Flows: {'B': {'A': 4}, 'E': {'D': 4}, 'C': {'A': 0, 'E': 4}, 'F': {'D': 0, 'C': 4}, 'D': {'B': 4}}
Neighbours of A : ['B', 'C']
Choose node B
Path: ['A', 'B']
Neighbours of B : ['D']
Changeable neighbours of B : ['D']
Choose node D
Path: ['A', 'B', 'D']
Neighbours of D : ['E', 'F']
Changeable neighbours of D : ['E', 'F']
Choose node E
Path: ['A', 'B', 'D', 'E']
Neighbours of E : []
Changeable neighbours of E : []
E is dead end
Neighbours of D : ['F']
Changeable neighbours of D : ['F']
Choose node F
Path: ['A', 'B', 'D', 'F']
Find path: ['A', 'B', 'D', 'F']
Minimal flow: 8
Bandwidth <A B> increased by 8
Bandwidth <B D> increased by 8
Bandwidth <D F> increased by 8
Update Graph: {'A': {'B': 2, 'C': 12}, 'D': {'E': 2, 'F': 0}, 'B': {'D': 0}, 'E': {'C': 0}, 'C': {'F': 14}}
Update Flows: {'B': {'A': 12}, 'E': {'D': 4}, 'C': {'A': 0, 'E': 4}, 'F': {'D': 8, 'C': 4}, 'D': {'B': 12}}
Neighbours of A : ['B', 'C']
Choose node B
Path: ['A', 'B']
Neighbours of B : []
Changeable neighbours of B : []
B is dead end
Neighbours of A : ['C']

```



Таблица 1. Тестирование.

	<p>Choose node C  Path: ['A', 'C']  Neighbours of C : ['F']  Changeable neighbours of C : ['F', 'E']  Choose node E  Path: ['A', 'C', 'E']  Neighbours of E : []  Changeable neighbours of E : ['D']  Choose node D  Path: ['A', 'C', 'E', 'D']  Neighbours of D : []  Changeable neighbours of D : []  D is dead end  Neighbours of E : []  Changeable neighbours of E : []  E is dead end  Neighbours of C : ['F']  Changeable neighbours of C : ['F']  Choose node F  Path: ['A', 'C', 'F']  Find path: ['A', 'C', 'F']  Minimal flow: 12  Bandwidth &lt;A C&gt; increased by 12  Bandwidth &lt;C F&gt; increased by 12  Update Graph: {'A': {'B': 2, 'C': 0}, 'D': {'E': 2, 'F': 0}, 'B': {'D': 0}, 'E': {'C': 0}, 'C': {'F': 2}}  Update Flows: {'B': {'A': 12}, 'E': {'D': 4}, 'C': {'A': 12, 'E': 4}, 'F': {'D': 8, 'C': 16}, 'D': {'B': 12}}  Neighbours of A : ['B']  Choose node B  Path: ['A', 'B']  Neighbours of B : []  Changeable neighbours of B : []  B is dead end  Neighbours of A : []  Can't find changeable path  Find path: []  24  A B 12  A C 12  B D 12  C F 16  D E 4  D F 8  E C 4</p>
20 a e a b 22	<p>Graph: {'a': {'b': 22, 'd': 11, 'c': 34, 'f': 24}, 'b': {'a': 22, 'c': 54, 'e': 21, 'f': 36}, 'd': {'a': 11, 'c': 10, 'e': 13}, 'c': {'a': 34, 'b': 54, 'd': 10, 'e': 20}, 'e': {'c': 20, 'b': 21, 'd': 13}, 'f': {'a': 24,</p>

Таблица 1. Тестирование.

b a 22	'b': 36}}
a d 11	Flows: {'b': {'a': 0, 'c': 0, 'e': 0, 'f': 0}, 'a':
d a 11	{'b': 0, 'd': 0, 'c': 0, 'f': 0}, 'd': {'a': 0, 'c': 0,
a c 34	'e': 0}, 'c': {'a': 0, 'b': 0, 'd': 0, 'e': 0}, 'e':
c a 34	{'c': 0, 'b': 0, 'd': 0}, 'f': {'a': 0, 'b': 0}}
b c 54	Neighbours of a : ['b', 'd', 'c', 'f']
c b 54	Choose node b
c d 10	Path: ['a', 'b']
d c 10	Neighbours of b : ['c', 'e', 'f']
c e 20	Changeable neighbours of b : ['c', 'e', 'f']
e c 20	Choose node c
b e 21	Path: ['a', 'b', 'c']
e b 21	Neighbours of c : ['d', 'e']
d e 13	Changeable neighbours of c : ['d', 'e']
e d 13	Choose node d
a f 24	Path: ['a', 'b', 'c', 'd']
f a 24	Neighbours of d : ['e']
b f 36	Changeable neighbours of d : ['e']
f b 36	Choose node e
	Path: ['a', 'b', 'c', 'd', 'e']
	Find path: ['a', 'b', 'c', 'd', 'e']
	Minimal flow: 10
	Bandwidth <a b> increased by 10
	Bandwidth <b c> increased by 10
	Bandwidth <c d> increased by 10
	Bandwidth <d e> increased by 10
	Update Graph: {'a': {'b': 12, 'd': 11, 'c': 34, 'f':
	24}, 'b': {'a': 22, 'c': 44, 'e': 21, 'f': 36}, 'd':
	{'a': 11, 'c': 10, 'e': 3}, 'c': {'a': 34, 'b': 54, 'd':
	0, 'e': 20}, 'e': {'c': 20, 'b': 21, 'd': 13}, 'f':
	{'a': 24, 'b': 36}}
	Update Flows: {'b': {'a': 10, 'c': 0, 'e': 0, 'f': 0},
	'a': {'b': 0, 'd': 0, 'c': 0, 'f': 0}, 'd': {'a': 0,
	'c': 10, 'e': 0}, 'c': {'a': 0, 'b': 10, 'd': 0, 'e':
	0}, 'e': {'c': 0, 'b': 0, 'd': 10}, 'f': {'a': 0, 'b':
	0}}
	Neighbours of a : ['b', 'd', 'c', 'f']
	Choose node b
	Path: ['a', 'b']
	Neighbours of b : ['c', 'e', 'f']
	Changeable neighbours of b : ['c', 'e', 'f']
	Choose node c
	Path: ['a', 'b', 'c']
	Neighbours of c : ['e']
	Changeable neighbours of c : ['e']
	Choose node e
	Path: ['a', 'b', 'c', 'e']
	Find path: ['a', 'b', 'c', 'e']
	Minimal flow: 12
	Bandwidth <a b> increased by 12

Таблица 1. Тестирование.

```

Bandwidth <b c> increased by 12
Bandwidth <c e> increased by 12
Update Graph: {'a': {'b': 0, 'd': 11, 'c': 34, 'f': 24}, 'b': {'a': 22, 'c': 32, 'e': 21, 'f': 36}, 'd': {'a': 11, 'c': 10, 'e': 3}, 'c': {'a': 34, 'b': 54, 'd': 0, 'e': 8}, 'e': {'c': 20, 'b': 21, 'd': 13}, 'f': {'a': 24, 'b': 36}}
Update Flows: {'b': {'a': 22, 'c': 0, 'e': 0, 'f': 0}, 'a': {'b': 0, 'd': 0, 'c': 0, 'f': 0}, 'd': {'a': 0, 'c': 10, 'e': 0}, 'c': {'a': 0, 'b': 22, 'd': 0, 'e': 0}, 'e': {'c': 12, 'b': 0, 'd': 10}, 'f': {'a': 0, 'b': 0}}
Neighbours of a : ['d', 'c', 'f']
Choose node c
Path: ['a', 'c']
Neighbours of c : ['b', 'e']
Changeable neighbours of c : ['b', 'e', 'b']
Choose node b
Path: ['a', 'c', 'b']
Neighbours of b : ['e', 'f']
Changeable neighbours of b : ['e', 'f']
Choose node e
Path: ['a', 'c', 'b', 'e']
Find path: ['a', 'c', 'b', 'e']
Minimal flow: 21
Bandwidth <a c> increased by 21
Bandwidth <c b> increased by 21
Bandwidth <b e> increased by 21
Update Graph: {'a': {'b': 0, 'd': 11, 'c': 13, 'f': 24}, 'b': {'a': 22, 'c': 32, 'e': 0, 'f': 36}, 'd': {'a': 11, 'c': 10, 'e': 3}, 'c': {'a': 34, 'b': 33, 'd': 0, 'e': 8}, 'e': {'c': 20, 'b': 21, 'd': 13}, 'f': {'a': 24, 'b': 36}}
Update Flows: {'b': {'a': 22, 'c': 21, 'e': 0, 'f': 0}, 'a': {'b': 0, 'd': 0, 'c': 0, 'f': 0}, 'd': {'a': 0, 'c': 10, 'e': 0}, 'c': {'a': 21, 'b': 22, 'd': 0, 'e': 0}, 'e': {'c': 12, 'b': 21, 'd': 10}, 'f': {'a': 0, 'b': 0}}
Neighbours of a : ['d', 'c', 'f']
Choose node c
Path: ['a', 'c']
Neighbours of c : ['b', 'e']
Changeable neighbours of c : ['b', 'e', 'b']
Choose node b
Path: ['a', 'c', 'b']
Neighbours of b : ['f']
Changeable neighbours of b : ['f']
Choose node f
Path: ['a', 'c', 'b', 'f']
Neighbours of f : []

```

Таблица 1. Тестирование.

```

Changeable neighbours of f : []
f is dead end
Neighbours of b : []
Changeable neighbours of b : []
b is dead end
Neighbours of c : ['e']
Changeable neighbours of c : ['e']
Choose node e
Path: ['a', 'c', 'e']
Find path: ['a', 'c', 'e']
Minimal flow: 8
Bandwidth <a c> increased by 8
Bandwidth <c e> increased by 8
Update Graph: {'a': {'b': 0, 'd': 11, 'c': 5, 'f': 24},
'b': {'a': 22, 'c': 32, 'e': 0, 'f': 36}, 'd': {'a': 11,
'c': 10, 'e': 3}, 'c': {'a': 34, 'b': 33, 'd': 0, 'e':
0}, 'e': {'c': 20, 'b': 21, 'd': 13}, 'f': {'a': 24,
'b': 36}}
Update Flows: {'b': {'a': 22, 'c': 21, 'e': 0, 'f': 0},
'a': {'b': 0, 'd': 0, 'c': 0, 'f': 0}, 'd': {'a': 0,
'c': 10, 'e': 0}, 'c': {'a': 29, 'b': 22, 'd': 0, 'e':
0}, 'e': {'c': 20, 'b': 21, 'd': 10}, 'f': {'a': 0, 'b':
0}}
Neighbours of a : ['d', 'c', 'f']
Choose node c
Path: ['a', 'c']
Neighbours of c : ['b']
Changeable neighbours of c : ['b', 'b']
Choose node b
Path: ['a', 'c', 'b']
Neighbours of b : ['f']
Changeable neighbours of b : ['f']
Choose node f
Path: ['a', 'c', 'b', 'f']
Neighbours of f : []
Changeable neighbours of f : []
f is dead end
Neighbours of b : []
Changeable neighbours of b : []
b is dead end
Neighbours of c : []
Changeable neighbours of c : []
c is dead end
Neighbours of a : ['d']
Choose node d
Path: ['a', 'd']
Neighbours of d : ['e']
Changeable neighbours of d : ['e']
Choose node e
Path: ['a', 'd', 'e']

```

Таблица 1. Тестирование.

```

Find path: ['a', 'd', 'e']
Minimal flow: 3
Bandwidth <a d> increased by 3
Bandwidth <d e> increased by 3
Update Graph: {'a': {'b': 0, 'd': 8, 'c': 5, 'f': 24},
'b': {'a': 22, 'c': 32, 'e': 0, 'f': 36}, 'd': {'a': 11,
'c': 10, 'e': 0}, 'c': {'a': 34, 'b': 33, 'd': 0, 'e':
0}, 'e': {'c': 20, 'b': 21, 'd': 13}, 'f': {'a': 24,
'b': 36}}
Update Flows: {'b': {'a': 22, 'c': 21, 'e': 0, 'f': 0},
'a': {'b': 0, 'd': 0, 'c': 0, 'f': 0}, 'd': {'a': 3,
'c': 10, 'e': 0}, 'c': {'a': 29, 'b': 22, 'd': 0, 'e':
0}, 'e': {'c': 20, 'b': 21, 'd': 13}, 'f': {'a': 0, 'b':
0}}
Neighbours of a : ['d', 'c', 'f']
Choose node c
Path: ['a', 'c']
Neighbours of c : ['b']
Changeable neighbours of c : ['b', 'b']
Choose node b
Path: ['a', 'c', 'b']
Neighbours of b : ['f']
Changeable neighbours of b : ['f']
Choose node f
Path: ['a', 'c', 'b', 'f']
Neighbours of f : []
Changeable neighbours of f : []
f is dead end
Neighbours of b : []
Changeable neighbours of b : []
b is dead end
Neighbours of c : []
Changeable neighbours of c : []
c is dead end
Neighbours of a : ['d']
Choose node d
Path: ['a', 'd']
Neighbours of d : []
Changeable neighbours of d : []
d is dead end
Neighbours of a : []
Can't find changeable path
Find path: []
Flow <c b> increased by 21
54
a b 22
a c 29
a d 3
a f 0
b a 0

```

Таблица 1. Тестирование.

b	c	1
b	e	21
b	f	0
c	a	0
c	b	0
c	d	10
c	e	20
d	a	0
d	c	0
d	e	13
e	b	0
e	c	0
e	d	0
f	a	0
f	b	0

**Вывод.**

В ходе работы был реализован алгоритм поиска максимального потока в сети Форда-Фалкерсона. Алгоритм использует поиск в глубину для поиска увеличивающей цепи. Данная программа прошла все тесты на степике. Для запуска были написаны Bash-скрипты расположенные в корневой директории. Один из них просто запускает программу, второй тестирует программу используя тесты из папки Tests.

## ПРИЛОЖЕНИЕ А

```
from math import inf
import copy

stepik = False

def changeable_path(graph, reversed_graph, start, finish):
    path = [start]
    bad_nodes = []
    while path[-1] != finish:
        neighbours = [e for e in graph[path[-1]] if graph[path[-1]][e] > 0
and e not in bad_nodes and e not in path]
        if not stepik:
            print("Neighbours of", path[-1], ":", neighbours)
        if path[-1] != start:
            neighbours.extend([e for e in reversed_graph[path[-1]] if
reversed_graph[path[-1]][e] > 0 and
                                e not in bad_nodes and e not in path])
        if not stepik:
            print("Changeable neighbours of", path[-1], ":",
neighbours)
        if not neighbours:
            if path[-1] == start:
                if not stepik:
                    print("Can't find changeable path")
                    return []
            else:
                if not stepik:
                    print(path[-1], "is dead end")
                    bad_nodes.append(path[-1])
                    path.pop()
        else:
            for u in sorted(neighbours):
                if path[-1] in graph or path[-1] in reversed_graph:
                    if not stepik:
                        print("Choose node", u)
                        path.append(u)
                    if not stepik:
                        print("Path:", path)
                        break
    return path

def main():
    n = int(input()) # количество ребер
    start = input()[0] # исток
    finish = input()[0] # сток
    graph = dict() # сам граф
    flows = dict()
    for _ in range(n):
        u, v, c = input().split() # чтение триплета
        if u in graph.keys():
            graph[u[0]][v[0]] = int(c) # если из данной вершины уже был
какой-либо путь то расширяем словарь
        else:
            graph[u[0]] = {v[0]: int(c)} # иначе создаем первый путь
        if v in flows.keys():
            flows[v[0]][u[0]] = 0 # если из данной вершины уже был какой-
либо путь то расширяем словарь
        else:
            flows[v[0]] = {u[0]: 0} # иначе создаем первый путь
```

```

if not stepik:
    print("Graph: ", graph)
    print("Flows: ", flows)
while True:
    # копии для того чтобы сделать вывод об увеличении максимального
    потока
    new_graph = copy.deepcopy(graph)
    new_flows = copy.deepcopy(flows)
    # ищем любую цепь из истока в сток не обращая внимание на
    направление дуг
    path = changeable_path(new_graph, new_flows, start, finish)
    if not stepik:
        print("Find path: ", path)
    if not path:
        break
    # поиск дуги с минимальной пропускной способностью или потоком
    flow = inf
    for i in range(len(path) - 1):
        # если движение в естественном направлении, то сравнение с
        пропускной способностью
        try:
            if new_graph[path[i]][path[i + 1]] < flow:
                flow = new_graph[path[i]][path[i + 1]]
        # иначе с потоком
        except KeyError:
            if new_flows[path[i]][path[i + 1]] < flow:
                flow = new_flows[path[i]][path[i + 1]]
    if not stepik:
        print("Minimal flow: ", flow)
    for i in range(len(path) - 1):
        # уменьшаем пропускные способности и увеличиваем потоки всех
        дуг чередующейся цепи, если по дуге
        try:
            new_graph[path[i]][path[i + 1]] -= flow
            new_flows[path[i + 1]][path[i]] += flow
            if not stepik:
                print("Bandwidth <{} {}> increased by
                {}".format(path[i], path[i + 1], flow))
            # увеличиваем пропускные способности и уменьшаем потоки всех
            дуг чередующейся цепи, если против дуги
            except KeyError:
                new_graph[path[i + 1]][path[i]] += flow
                new_flows[path[i]][path[i + 1]] -= flow
                if not stepik:
                    print("Bandwidth <{} {}> reduced by
                    {}".format(path[i], path[i + 1], flow))
            # проверка увеличился ли поток если нет то выходим из цикла
            if sum(flows[finish].values()) != sum(new_flows[finish].values()):
                graph = copy.deepcopy(new_graph)
                flows = copy.deepcopy(new_flows)
                if not stepik:
                    print("Update Graph: ", graph)
                    print("Update Flows: ", flows)
            # иначе обновляем графы
            else:
                break
    # ОПТИМИЗАЦИЯ ДВУНАПРАВЛЕННЫХ ДУГ
    for u in flows:
        for v in flows[u]:
            if v in flows and u in flows[v] and flows[u][v] and
            flows[v][u]:
                if flows[u][v] > flows[v][u]:
                    if not stepik:

```



```

        print("Flow <{} {}> increased by {}".format(u, v,
flows[v][u]))
        flows[u][v] -= flows[v][u]
        flows[v][u] = 0
    else:
        if not stepik:
            print("Flow <{} {}> increased by {}".format(v, u,
flows[u][v]))
            flows[v][u] -= flows[u][v]
            flows[u][v] = 0
    # составление ответа
    list_answer = []
    for u in graph:
        for v in graph[u]:
            list_answer.append(str(u) + " " + str(v) + " " +
str(flows[v][u]))
    list_answer = sorted(list_answer)
    print(sum(flows[finish].values()))
    [print(ans) for ans in list_answer]

if __name__ == "__main__":
    main()

```