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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Потоки в сети

Студент гр. 7304 _____ Давыдов А.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Задача

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).

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 работы алгоритма

1) На каждом шаге алгоритма пытаемся найти путь из истока в сток поиском в глубину, используя следующие правила:
    - Можно переходить по ребру, если его пропускная способность больше нуля

- Если путь найден, ищем минимальную пропускную способность в нем и вычитаем из пропускных способностей всех прямых ребер этого пути. К обратным ребрам наоборот прибавляем.

2) Если путь найден, то добавляем минимальную пропускную способность из этого пути к переменной максимального потока. Изначально переменная максимального потока имеет значение 0
3) Если нельзя найти путь, то алгоритм завершает работу и на экран выводится максимальный поток и фактические величины потока через каждое ребро сети.

Выводы.

Был изучен, описан и реализован алгоритм Форда-Фалкерсона в соответствии с условием задания. Данный алгоритм находит максимальный поток в сети и после его работы можно вычислить фактическую величину потока через каждое ребро сети.

ПРИЛОЖЕНИЕ А(ИСХОДНЫЙ КОД ПРОГРАММЫ)

```
#include <iostream>
#include <map>
#include <vector>
#include <set>
#include <climits>

using namespace std;
using Vertex = char;


//find path from source to end
map<Vertex, pair<Vertex, int>> Dfs(map<Vertex, map<Vertex, int>> graph, Vertex
source, Vertex end)
{
map<Vertex, pair<Vertex, int>> path;
map<Vertex, bool> visited_vertex;
vector<Vertex> stack;

for(map<Vertex, map<Vertex, int>>::iterator it = graph.begin(); it!=
graph.end(); ++it)
visited_vertex[it->first] = false;

map<Vertex, map<Vertex, int>>::iterator cur = graph.find(source);
visited_vertex[source] = true;
stack.push_back(source);

while(cur->first!= end && stack.size()!= 0)
{
bool append_new = false;
for (map<Vertex, int>::iterator it = cur->second.begin(); it!= cur-
>second.end(); ++it)
if(!visited_vertex[it->first] && it->second!= 0)
{
path[cur->first] = pair<Vertex, int>(it->first, it->second);
cur = graph.find(it->first);
```

```
visited_vertex[cur->first] = true;
stack.push_back(cur->first);
append_new = true;
break;
}

//deadlock
if(!append_new)
{
append_new = false;
path.erase(stack.back());
stack.pop_back();
cur = graph.find(stack.back());
}
}

return path;
}


int min_capacity(map<Vertex, pair<Vertex, int>> path)
{
int min = INT_MAX;

for(map<Vertex, pair<Vertex, int>>::iterator it = path.begin(); it!= path.end();
++it)
if(it->second.second < min && it->second.second > 0)
min = it->second.second;
return min;
}


void MaxFlow(map<Vertex, map<Vertex, int>> &network, int &max_flow, Vertex
source, Vertex end)
{
map<Vertex, pair<Vertex, int>> path = Dfs(network, source, end);

for(/* do nothing */; path.size()!= 0; /* do nothing */)
{
int min = min_capacity(path);
max_flow += min;
for(map<Vertex, pair<Vertex, int>>::iterator it = path.begin(); it!= path.end();
++it)
{
network[it->first][it->second.first] -= min;
network[it->second.first][it->first] += min;
}
path = Dfs(network, source, end);
}
}


int main()
{
map<Vertex, map<Vertex, int>> graph;
map<Vertex, map<Vertex, int>> straight_line_graph;
Vertex source, destination;
int count_edges;

cin >> count_edges >> source >> destination;

Vertex u, v;
int c;
```

```
for (int i = 0; i < count_edges; ++i)
{
cin >> u >> v >> c;
graph[u][v] = c; //straight line edge
straight_line_graph[u][v] = c;
graph[v][v] = 0;
//create reverse edge
if(graph[v].find(u) == graph[v].end())
graph[v][u] = 0;
}

map<Vertex, map<Vertex, int>> network = graph;
int max_flow = 0;

MaxFlow(network, max_flow, source, destination);
cout << endl;

cout << max_flow << endl;
for(map<Vertex, map<Vertex, int>>::iterator it = straight_line_graph.begin();
it!= straight_line_graph.end();++it)
for(map<Vertex, int>::iterator it2 = it->second.begin(); it2!= it->second.end();
++it2)
if(it2->second - network[it->first][it2->first] >= 0)
cout << it->first << " " << it2->first << " " << it2->second - network[it-
>first][it2->first] << endl;
else
cout << it->first << " " << it2->first << " " << 0 << endl;

return 0;
}
```