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: Потоки в сети

Студент гр. 7304 _____ Шарапенков И.И.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

Цель работы.

Изучение алгоритмов поиска максимального потока в сети (метод Форда-Фалкерсона)

Теоретические сведения.

Сеть – ориентированный взвешенный граф, имеющий один исток и один сток.

Исток – вершина, из которой рёбра только выходят.

Сток – вершина, в которую рёбра только входят.

Поток – абстрактное понятие, показывающее движение по графу.

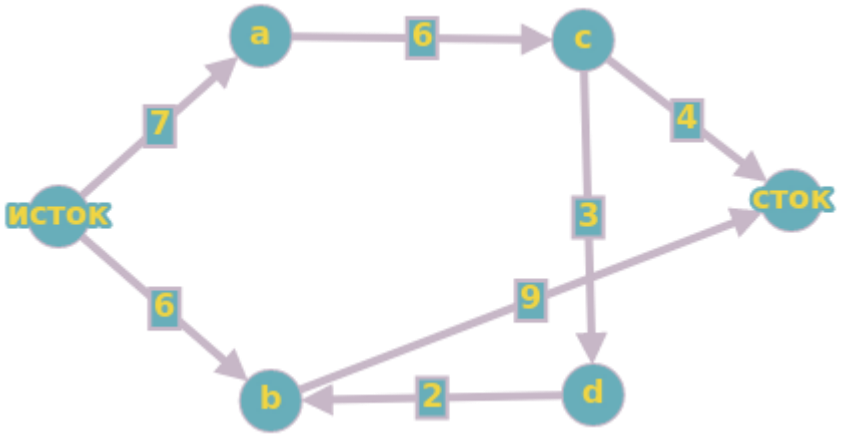
Величина потока – числовая характеристика движения по графу (сколько всего выходит из стока = сколько всего входит в сток).

Пропускная способность – свойство ребра, показывающее, какая максимальная величина потока может пройти через это ребро.

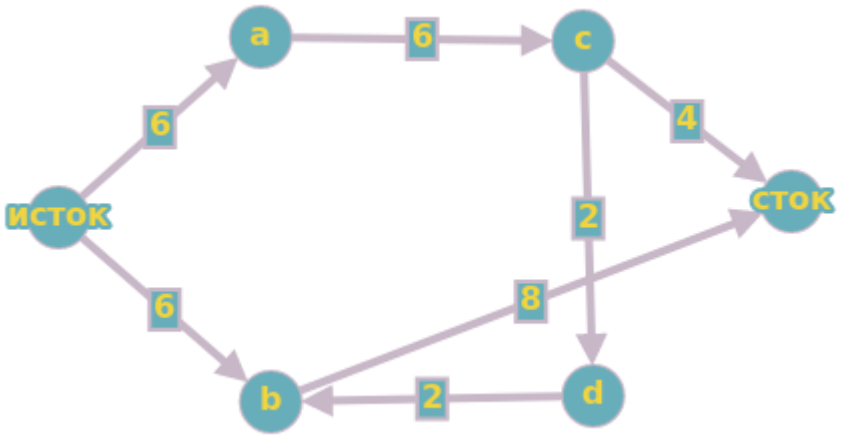
Максимальный поток (максимальная величина потока) – максимальная величина, которая может быть выпущена из стока, которая может пройти через все рёбра графа, не вызывая переполнения ни в одном ребре.

Фактическая величина потока в ребре – значение, показывающее, сколько величины потока проходит через это ребро.

Пример сети:



Фактический поток через эту сеть:



Обратите внимание, что во втором графе для каждой вершины кроме истока и стока действует правило: сколько в вершину втекает, столько и вытекает (для истока и стока – сколько из истока вытекает, столько в сток и втекает).

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 алгоритма.

1. Считываем граф в виде матрицы смежности. Значение в узлах матрицы – пропускная способность ребра.
2. Создаем новый граф – остаточный. На начальном этапе остаточный граф совпадает с исходным. Max_flow – значение максимального потока, инициализируется нулем.
3. Если не существует путей из вершины start в end алгоритм заканчивает работу.
4. Ищем произвольный путь от start поиском в ширину. Сохраняем этот путь в ассоциативный массив path.
5. Ищем ребро с минимальным весом на этом пути. Сохраняем значение минимального веса.
6. Пересчитываем значения остаточной пропускной способности для каждого ребра. При этом по направлению потока остаточная пропускная способность уменьшается на значение минимального веса, а против направления потока увеличивается.

7. Прибавляем значение максимального веса к значению максимального потока на текущем шаге.
8. Переходим к шагу 3.

Пример работы.

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

Вывод

В ходе лабораторной работы был изучен алгоритм поиска максимального потока в сети – метод Форда-Фадкерсона. Данный алгоритм был реализован на языке программирования C++. Для поиска путей использовался поиск в ширину, такая модификация метода называется алгоритм Эдмондса-Карпа.

Исходный код программы

```
#include <iostream>
#include <algorithm>
#include <map>
#include <vector>
#include <deque>
#include <climits>


using Vertex = char;
using Graph = std::map<Vertex, std::map<Vertex, long long int>>;


bool BFS(Graph &graph, Vertex start, Vertex end, std::map<Vertex, Vertex> &path)
{
    std::vector<Vertex> closed_set = {start};
    std::deque<Vertex> open_set = {start};
    std::map<Vertex, Vertex> came_from;
    Vertex current;

    while (!open_set.empty()) {
        current = open_set.front();
        open_set.pop_front();

        for (auto const &neighbor: graph[current])
            if (neighbor.second > 0 &&
                !(std::find(closed_set.begin(), closed_set.end(),
neighbor.first) != closed_set.end())) {
                open_set.push_back(neighbor.first);
                closed_set.push_back(neighbor.first);
                came_from[neighbor.first] = current;
            }

    }

    path = came_from;
    return std::find(closed_set.begin(), closed_set.end(), end) !=
closed_set.end();
}


long long int FFA(Graph &graph, Vertex start, Vertex end) {
    std::map<Vertex, Vertex> path;
    long long int max_flow = 0, path_flow;
    Vertex s, v, u;

    while (BFS(graph, start, end, path)) {
        path_flow = LLONG_MAX;

        s = end;
        while (s != start) {
            path_flow = std::min(path_flow, graph[path[s]][s]);
            s = path[s];
        }

        max_flow += path_flow;

        v = end;
        while (v != start) {
            u = path[v];
            graph[u][v] -= path_flow;
            graph[v][u] += path_flow;
```

```
            v = path[v];
        }
    }
    return max_flow;
}


int main() {
    Graph graph, residual_graph;
    Vertex start, end, v1, v2;
    long long int cost, N, max_val;

    std::cin >> N >> start >> end;

    while (N-- && std::cin >> v1 >> v2 >> cost)
        graph[v1][v2] = cost;

    residual_graph = graph;
    max_val = FFA(residual_graph, start, end);

    std::cout << max_val << std::endl;

    for (auto const &current: graph)
        for (auto const &neighbor: graph[current.first])
            std::cout << current.first << " " << neighbor.first << " "
                      << (int(neighbor.second -
residual_graph[current.first][neighbor.first]) < 0 ? 0 :
                          int(neighbor.second -
residual_graph[current.first][neighbor.first])) << std::endl;


    return 0;
}
```