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: Алгоритм Форда-Фалкерсона

Студент гр. 7304 _____ Есиков О.И.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

Цель работы.

Изучить и реализовать на языке программирования c++ алгоритм Форда-Фалкерсона, который позволяет найти максимальный поток в сети.

Формулировка задания.

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).

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.

Алгоритм.

Шаг 1: строится остаточная сеть, в которой изначально поток через каждое ребро равен 0, максимальный поток в сети равен 0.

Шаг 2: ищется путь от истока к стоку через рёбра, которые имеют не нулевой вес (разрешается переход от конца ребра к его началу с уменьшением потока через него), если путь не найден, то переход на шаг 4.

Шаг 3: в найденном пути ищется ребро с минимальным весом, величина этого ребра добавляется к максимальному потоку в графе, его величина вычитается из весов всех рёбер и прибавляется к величине потока, после чего переход на шаг 2.

Шаг 4: вывод максимального потока и потока через каждое ребро.

Пример работы программы.

Входные данные:

```
7
a
f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
```

Выходные данные:

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

Выводы.

В ходе выполнения данной лабораторной работы были изучен и реализован на языке программирования c++ алгоритм Форда-Фалкерсона, который ищет максимальный поток через сеть. Для поиска очередного пути от истока к стоку использовался алгоритм поиска в глубину (DFS), сложность которого составляет O (|E| + |V|), где E – множество всех рёбер в графе, V – множество всех вершин. В результате общая сложность алгоритма Форда-Фалкерсона составляет O ((|E| + |V|)*F), где F – максимальный поток в сети. Также, стоит отметить особенность данного алгоритма, которая заключается в том, что он гарантированно сходится только для целых пропускных способностей рёбер, в случае вещественных весов алгоритм может работать бесконечно долго, даже не сходясь к правильному результату.

Приложение А.

Исходный код.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct edge
{
    char beg;
    char end;
    int heft;
    int forward;
    int back;
    bool doubl;
};

bool compare(edge first, edge second)
{
    if(first.beg == second.beg)
        return first.end < second.end;
    return first.beg < second.beg;
}

class Graph
{
private:
    vector <edge> graph;
    char source;
    char estuary;
    int N;
    vector <char> viewingpoint;
    vector <char> result;

public:
    Graph()
    {
        cin >> N;
        cin >> source >> estuary;
        for(int i = 0; i < N; i++)
        {
            edge element;
            cin >> element.beg >> element.end >> element.heft;
            element.forward = element.heft;
            element.back = 0;
            element.doubl = false;
            bool flag = true;
            for(int i = 0; i < graph.size(); i++)
            {
                if(graph.at(i).beg == element.end && graph.at(i).end ==
element.beg)
                {
                    graph.at(i).back += element.forward;
                    flag = false;
                    graph.at(i).doubl = true;
                    break;
                }
            }
            if(!flag)
                continue;
```

```
        graph.push_back(element);
    }
}

bool isViewing(char value)
{
    for(size_t i = 0; i < viewingpoint.size(); i++)
        if(viewingpoint.at(i) == value)
            return true;
    return false;
}

bool Search(char value, int& min)
{
    if(value == estuary)
    {
        result.push_back(value);
        return true;
    }
    viewingpoint.push_back(value);
    for(size_t i(0); i < graph.size(); i++)
    {
        if(value == graph.at(i).beg)
        {
            if(isViewing(graph.at(i).end) || graph.at(i).forward == 0)
                continue;
            result.push_back(graph.at(i).beg);
            bool flag = Search(graph.at(i).end, min);
            if(flag)
            {
               if(graph.at(i).forward < min)
                   min = graph.at(i).forward;
               return true;
            }
            result.pop_back();
        }
        if(value == graph.at(i).end)
        {
            if(isViewing(graph.at(i).beg) || graph.at(i).back == 0)
                continue;
            result.push_back(graph.at(i).end);
            bool flag = Search(graph.at(i).beg, min);
            if(flag)
            {
               if(graph.at(i).back < min)
                    min = graph.at(i).back;
               return true;
            }
            result.pop_back();
        }
    }
    return false;
}


void FordFalk()
{
    int res = 0;
    int min = 9999;

    while(Search(source, min))
    {
        for(int i = 1; i < result.size(); i++)
```

```
            {
                for(int j = 0; j < graph.size(); j++)
                {
                    if(graph.at(j).beg == result.at(i-1) && graph.at(j).end ==
result.at(i))
                    {
                        graph.at(j).forward -= min;
                        graph.at(j).back += min;
                    }
                    if(graph.at(j).end == result.at(i-1) && graph.at(j).beg ==
result.at(i))
                    {
                        graph.at(j).forward += min;
                        graph.at(j).back -= min;
                    }
                }
            }
            res += min;
            viewingpoint.clear();
            result.clear();
            min = 9999;
        }

        sort(graph.begin(), graph.end(), compare);
        cout << res << endl;
        for(int i = 0; i < graph.size(); i++)
        {
            int value = max(graph.at(i).heft - graph.at(i).forward, 0 -
graph.at(i).back);
            if(graph.at(i).doubl == true)
            {
                if(value < 0)
                    value = 0;
                cout << graph.at(i).beg << " " << graph.at(i).end << " " <<
value << endl;
                swap(graph.at(i).beg, graph.at(i).end);
                swap(graph.at(i).back, graph.at(i).forward);
                graph.at(i).doubl = false;
                sort(graph.begin(), graph.end(), compare);
                i--;
            }
            else
            {
                cout << graph.at(i).beg << " " << graph.at(i).end << " " <<
value << endl;
            }
        }
    }
};

int main()
{
    Graph element;
    element.FordFalk();
    return 0;
}
```