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. 7383 _____ Левкович Д.В.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Реализовать и исследовать алгоритм Форда-Фалкерсона поиска максимального потока в сети.

Постановка задачи

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).

Ход работы

Была написана программа на языке программирования C++. Код представлен в приложении А.

Для представления сети используется класс, представляющий из себя список смежности. В списке хранятся ребра и поток через эти ребра. Метод Init осуществляет ввод данных с клавиатуры и заполняет список смежности. Далее вызывается метод maxFlow, который в свою очередь вызывает методы find_way и reconstructionNetwork. Метод find_way находит пусть в сети по правилу: каждый раз берется дуга, имеющая максимальную остаточную пропускную способность. Когда найден путь, вызываем метод reconstuctionNetwork, который находит с помощью метода get_min_flow находит ребро на данном пути с минимальной остаточной пропускной способностью, которое в данном случае будет максимальным потоком для пути. Далее на всем данном пути пропускная способность дуг уменьшается на это значение, а также создается обратное ребро, оно направлено в другую сторону и на нем остаточная пропускная способность увеличивается. Если остаточная пропускная способность ребра равна максимальному потоку на пути, то ребро удаляется, так как через него уже никакой поток пройти не сможет. Алгоритм заканчивает работу, когда невозможно найти путь из истока в сток.

Тестирование

Тестирование проводилось в Windows 10. По результатам тестирования были выявлены ошибки в коде. Тестовые случаи представлены в приложении Б.

Исследование алгоритма

На каждом шаге алгоритм добавляет поток увеличивающего пути к уже имеющемуся потоку. Следовательно, 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(f|E|).

Выводы

Был изучен алгоритм Форда-Фалкерсона поиска максимального потока в сети. Была реализована версия алгоритма на языке C++, исследована сложность алгоритма, по результатам сложность равна O(f|E|).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <map>
#include <iterator>
#include <algorithm>
#include <limits>
using namespace std;

template <typename type, typename T>
class Flow{
private:
    map<type, map<type, T>> graph;
    type u;
    type v;
    T capacity;
public:

    Flow(){
        capacity = 0;
        u = 0;
        v = 0;
    }

    void Init(){
        type a, b;
        size_t N;
        T w;
        cin>>N>>u>>v;
        for(size_t i = 0;i<N;i++){
            cin>>a>>b>>w;
            graph[a][b] = w;
        }
        maxFlow(u, v);
    }

    map<type, pair<type, T>> find_way(map<type, map<type, T>> m, type
start, type finish){
        map<type, pair<type, T>> way;
        map<type, bool> visited;
        vector<type> stack;
        typename map<type, map<type, T>>::iterator it;
        typename map<type, map<type, T>>::iterator t;
        for (it = m.begin(); it != m.end(); it++)
            visited[it->first] = false;
        stack.push_back(start);
        it = graph.begin();
```

```
        for (   ; it != m.end(); it++) {
            if(stack[stack.size()-1] == finish)
                break;
            visited[stack[stack.size()-1]] = true;
            vector<pair<type, T>> neighbors;
            for(pair<type, T> neighbor:m.find(stack[stack.size()-1])-
>second)
                if(!visited[neighbor.first])
                    neighbors.push_back(neighbor);
            if(neighbors.empty()){
                if(way.size()==1){
                    way.clear();
                    return way;
                }
                way.erase(stack[stack.size()-1]);
                stack.pop_back();
                continue;
            }
            pair<type, T> max =
*max_element(neighbors.begin(),neighbors.end(),[](pair<type, T>&
n1,pair<type, T>& n2){
                    return  n1.second < n2.second;
            });
            if(max.second != 0){
                way[stack[stack.size()-1]] = pair<type, T>(max.first,
max.second);
                stack.push_back(max.first);
            }
        }
        return way;
    }

    int get_min_flow(map<type,pair<type,T>> way){
        auto t = way.begin();
        T min_flow = t->second.second;
        for(    ;t!=way.end();t++) {
            if(min_flow > t->second.second)
                min_flow = t->second.second;
        }
        return min_flow;
    }

    void reconstructionNetwork(map<type,map<type,T>> &
network,map<type,pair<type,T>> way){
        int min_flow=get_min_flow(way);
        for (auto t = way.begin();t!=way.end();t++) {
            if(t->second.second - min_flow == 0)
                network[t->first].erase(t->second.first);
            else
                network[t->first][t->second.first] -= min_flow;
            network[t->second.first][t->first] += min_flow;
```

```
        }
    }

    void maxFlow(type source,type stock){
        map<type,map<type,T>> network=graph;
        map<type,pair<type,T>> way=find_way(network,source,stock);
        while(!way.empty()){
            reconstructionNetwork(network,way);
            way = find_way(network,source,stock);
        }
        T max_flow=0;

for_each(graph[source].begin(),graph[source].end(),[&max_flow,&network,so
urce](pair<type,T> neighbor){
            max_flow+=neighbor.second-network[source][neighbor.first];
        });
        cout<<max_flow<<endl;

for_each(graph.begin(),graph.end(),[&network](pair<type,map<type,T>>
neighbors){

for_each(neighbors.second.begin(),neighbors.second.end(),[&network,neighb
ors](pair<type,T> ver){
                cout<<neighbors.first<<" "<<ver.first<<" ";
                if(ver.second-network[neighbors.first][ver.first]>=0)
                   cout<<ver.second-
network[neighbors.first][ver.first]<<endl;
                else
                   cout<<0<<endl;
            });
        });
    }
};

int main()
{
    Flow<char, int> a;
    a.Init();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

Входные данные	Результат
7 a f a b 8 b c 4 c d 3 d e 2 e b 1 e f 1 b f 6	7 a f a b 8 b c 4 c d 3 d e 2 e b 1 e f 1 b f 6
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
16 a e a b 2 b a 2 a d 1 d a 1 a c 3 c a 3 b c 4 c b 4 c d 1	6 a b 2 a c 3 a d 1 b a 0 b c 0 b e 3 c a 0 c b 1 c d 0 c e 2 d a 0

d c 1 c e 2 e c 2 b e 3 e b 3 d e 1 e d 1	d c 0 d e 1 e b 0 e c 0 e d 0