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 _____ Бергалиев М.

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 матрицу смежности. В матрице, помимо наличия ребер, хранится их пропускная способность и фактический поток. Также в классе хранится матрица обратных ребер. Конструктор принимает массив ребер с их пропускными способностями и строит матрицы. Содержит метод bfs, ищущий путь из одной вершины в другую поиском в ширину.

Метод fold_fulkerson на каждом шаге находит путь из истока в сток и увеличивает потоки в ребрах данного пути на минимальную остаточную пропускную способность ребер этого пути. Алгоритм продолжает работу, пока есть путь из истока в сток.

Тестирование

Тестирование проводилось в Ubuntu 16.04 LTS. По результатам тестирования были выявлены ошибки в коде. Тестовые случаи представлены в приложении Б.

Исследование алгоритма

Исследование проводилось по количеству просмотренных вершин.

Поскольку на каждом шаге увеличиваем поток как минимум на 1, а поиск пути в графе происходит за O(|E|) операций, то алгоритм находит решение за O(F|E|) операций, где F — максимальный поток в сети. Полученный график показан на рис. 1.

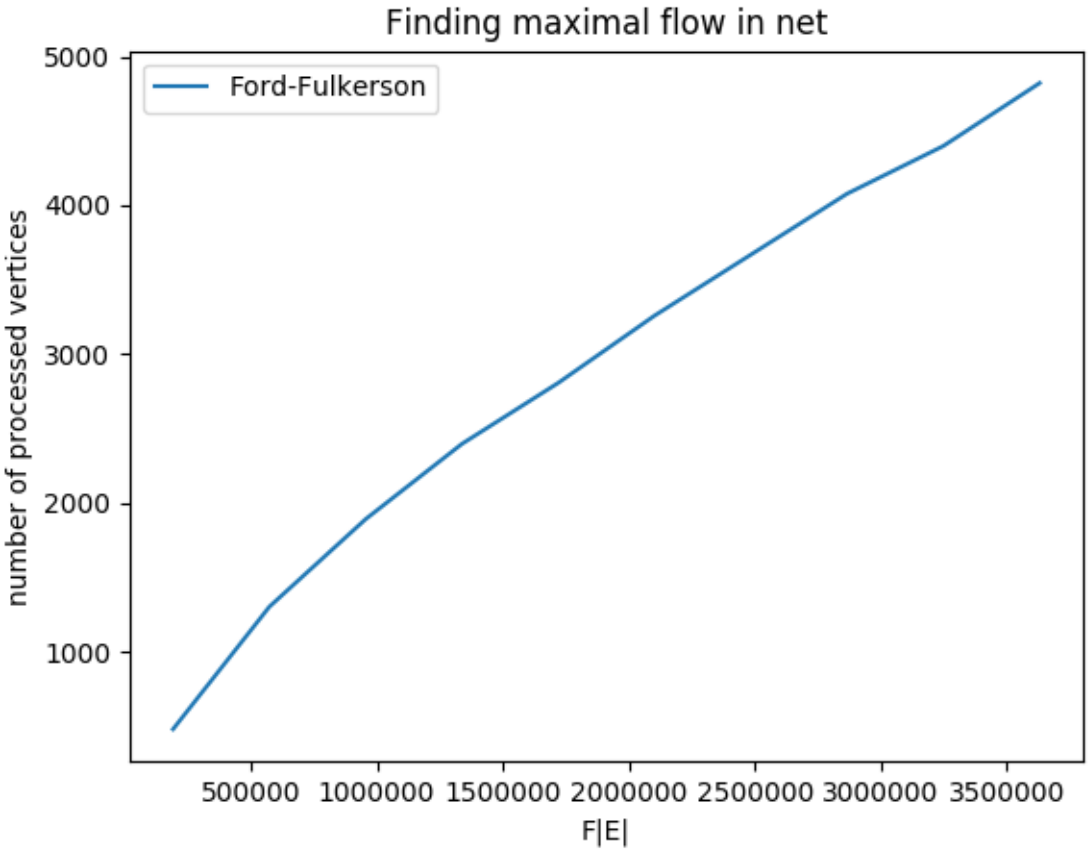

Рисунок 1 — Сложность алгоритма через максимальный поток.

Однако данная оценка требует знать максимальный поток. Так как максимальный поток не может превышать сумму пропускных способностей истока и сумму пропускных способностей стока, то можно заменить F на максимальную из этих двух сумм M. Тогда сложность алгоритма O(M|E|). Полученный график показан на рис. 2.

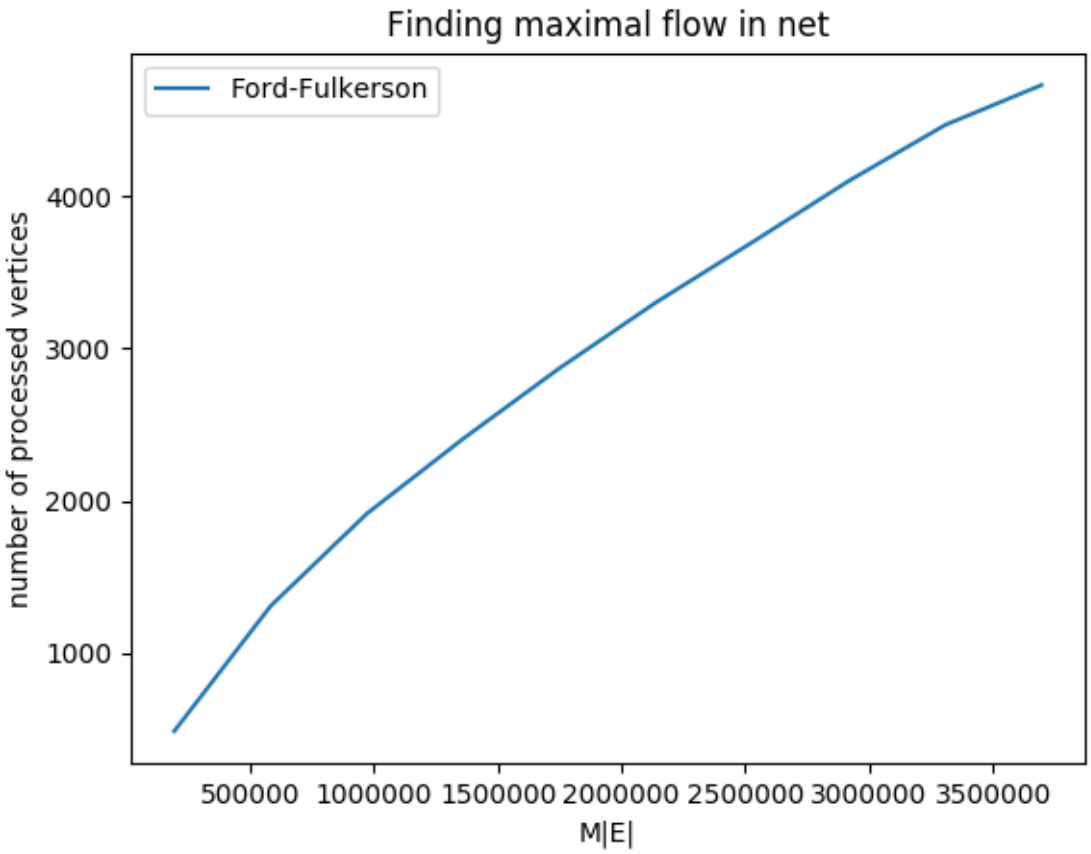

Рисунок 2 — Сложность алгоритма.

Вывод

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(M|E|).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

class Adj_Matrix{
private:
	struct Edge{
		int bandwidth;
		int flow;
		Edge(int band, int flow) : bandwidth(band) , flow(flow) {}
	};
public:
	Adj_Matrix(std::vector<std::tuple<char, char, int>> & v){
		for(auto i : v){
			if(find(indexes.begin(), indexes.end(), std::get<0>(i))
== indexes.end())
				indexes.push_back(std::get<0>(i));
			if(find(indexes.begin(), indexes.end(), std::get<1>(i))
== indexes.end())
				indexes.push_back(std::get<1>(i));
		}
		int n = indexes.size();
		matrix = new Edge**[n];
		for(int i=0; i<n; ++i){
			matrix[i] = new Edge*[n];
		}
		rest_net = new Edge**[n];
		for(int i=0; i<n; ++i){
			rest_net[i] = new Edge*[n];
		}
		char from, to;
		int bandwidth;
		int index_from, index_to;
		std::vector<char>::iterator it;
		for(auto i : v){
			std::tie(from, to, bandwidth) = i;
			it = find(indexes.begin(), indexes.end(), from);
			index_from = it - indexes.begin();
			it = find(indexes.begin(), indexes.end(), to);
			index_to = it - indexes.begin();
```

```
                        matrix[index_from][index_to] = new Edge(bandwidth,
0);
                        rest_net[index_to][index_from] = new Edge(0, 0);
                }
        }

        std::string bfs(char from, char to){
                std::queue<std::string> queue;
                std::vector<int> checked;
                std::string cur(1, from);
                int n = indexes.size();
                int i;
                while(true){
                        i = find(indexes.begin(), indexes.end(), cur.back()) -
indexes.begin();
                        checked.push_back(i);
                        for(int j=0; j<n; ++j){
                                if(find(checked.begin(), checked.end(), j) !=
checked.end())
                                        continue;
                                if((matrix[i][j] != nullptr) && (matrix[i][j]-
>flow != matrix[i][j]->bandwidth)){
                                        if(indexes[j] == to)
                                                return cur + '+' + to;
                                        queue.push(cur + '+' + indexes[j]);
                                        checked.push_back(j);
                                }
                                if((rest_net[i][j] != nullptr) && (rest_net[i][j]-
>flow != rest_net[i][j]->bandwidth)){
                                        queue.push(cur + '-' + indexes[j]);
                                        checked.push_back(j);
                                }
                        }
                        if(queue.empty())
                                return "";
                        cur = queue.front();
                        queue.pop();
                }
        }

        std::vector<std::tuple<char, char, int>> ford_fulkerson(char
sourse, char stock){
                std::string path;
                int min;
                int i, j;
```

```
                while(true){
                        path = bfs(sourse, stock);
                        if(path == "")
                                break;;
                        min = 0;
                        i = find(indexes.begin(), indexes.end(), path.front()) -
indexes.begin();
                        for(int k=1; k<path.length(); k += 2){
                                j     =     find(indexes.begin(),     indexes.end(),
path[k+1]) - indexes.begin();
                                if(path[k] == '+'){
                                        if((matrix[i][j]->bandwidth   -   matrix[i][j]-
>flow < min) && (matrix[i][j]->bandwidth - matrix[i][j]->flow != 0) || (min
== 0))
                                                min   =   matrix[i][j]->bandwidth   -
matrix[i][j]->flow;
                                }
                                else if((rest_net[i][j]->bandwidth - rest_net[i][j]-
>flow < min) && (rest_net[i][j]->bandwidth - rest_net[i][j]->flow != 0) || (min
== 0))
                                                min   =   rest_net[i][j]->bandwidth   -
rest_net[i][j]->flow;
                                i = j;
                        }
                        i = find(indexes.begin(), indexes.end(), path.front()) -
indexes.begin();
                        for(int k=1; k<path.length(); k+=2){
                                j     =     find(indexes.begin(),     indexes.end(),
path[k+1]) - indexes.begin();
                                if(path[k] == '+'){
                                        matrix[i][j]->flow += min;
                                        rest_net[j][i]->flow -= min;
                                }
                                else{
                                        rest_net[i][j]->flow += min;
                                        matrix[j][i]->flow -= min;
                                }
                                i = j;
                        }
                }
                int n=indexes.size();
                std::vector<std::tuple<char, char, int>> ans;
                for(i=0; i<n; ++i)
                        for(j=0; j<n; ++j)
                                if(matrix[i][j] != nullptr)
```

```
        ans.push_back(std::make_tuple(indexes[i], indexes[j], matrix[i][j]->flow));
                return ans;
        }

    private:
        std::vector<char> indexes;
        Edge* **matrix;
        Edge* **rest_net;
    };

    int main(){
        std::vector<std::tuple<char, char, int>> net;
        int n;
        std::cin >> n;
        char sourse, stock;
        std::cin >> sourse;
        std::cin >> stock;
        char from, to;
        int w;
        for(int i=0; i<n; ++i){
            std::cin >> from;
            std::cin >> to;
            std::cin >> w;
            net.push_back(std::make_tuple(from, to, w));
        }
        Adj_Matrix matr(net);
        auto res = matr.ford_fulkerson(sourse, stock);
        std::sort(res.begin(), res.end());
        int sum = 0;
        for(auto i : res)
            if(std::get<0>(i) == sourse)
                sum += std::get<2>(i);
        std::cout << sum << std::endl;
        for(auto i : res)
            std::cout << std::get<0>(i) << " " << std::get<1>(i) << " " << std::get<2>(i) << std::endl;
        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