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»

Студентка гр.7304 _____ Каляева А.В.

Преподаватель _____ Филатов А.Ю

г. Санкт-Петербург

г. 2019

Цель работы:

Изучить алгоритм Форда-Фалкерсона поиска максимального потока в сети, а так же реализовать данный алгоритм на языке программирования C++.

Задание:

Найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда-Фалкерсона.

Теоретический материал:

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.

Ход работы:

Был реализован алгоритм поиска максимального потока в сети на языке программирования C++.

1) Создается граф, заданный матрицей смежности. Изначально все вершины отмечены как не просмотренные, все потоки изначально равны 0.

2) Для нахождения пути в графе выполняется поиск в глубину.

3) Если удалось найти путь из истока в сток, то для данного пути выполняется поиск максимального потока.

4) Для прямых ребер поток увеличивается на найденную величину, для обратных ребер поток уменьшается на найденную величину.

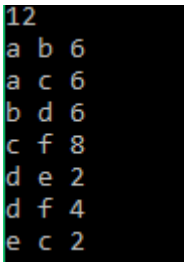
5) Значение максимального потока, найденного для данного пути, прибавляется к конечному значению максимального потока для всего графа.

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
```

Результат работы программы:



Заключение:

В ходе выполнения лабораторной работы по теме «Потоки в сети» были изучены такие понятия, как сеть, исток, сток, поток, величина потока, пропускная способность. Был изучен и реализован на языке программирования C++ алгоритм Форда-Фалкерсона нахождения максимального потока в сети. Для корректной работы алгоритма был использован алгоритм поиска в глубину (DFS), время работы которого оценивается как O(V+E). Алгоритм Форда - Фалкерсона гарантированно сходится только для целых пропускных способностей, но даже для них при больших значениях пропускных способностей он может работать очень долго. Если пропускные способности вещественны, алгоритм может работать бесконечно долго, не сходясь к оптимальному решения\

Приложение А
Исходный код программы lr3.cpp

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

#define MAX 10000000
#define COUNT 26

using namespace std;

class Ford_Falkerson {
	int size;
	char source;
	char stock;
	char from;
	char to;
	int cost;
	int delta = 0;
	vector<vector<int>> graph;
	vector<vector<int>> flows;
	vector<bool>visited;
	vector<int> way;
public:
	Ford_Falkerson(int digit, char symbol) :size(digit), source(symbol), stock(symbol),
from(symbol), to(symbol), cost(digit), delta(digit), graph(COUNT, vector<int>(COUNT, 0)),
flows(COUNT, vector<int>(COUNT, 0)),
                                        visited(COUNT, false), way(COUNT, 0){}
	void print_graph() {
		for (int i = 0; i < COUNT; i++) {
			for (int j = 0; j < COUNT; j++) {
				cout << graph[i][j] << " ";
			}
			cout << endl;
		}
	}
	void creation_graph() {
		cin >> size >> source >> stock;
		if (isalpha(source)) {
			delta = 97;
		}
		else {
			delta = 49;
		}
		for (int i = 0; i < size; i++)
		{
			cin >> from >> to >> cost;
			graph[from - delta][to - delta] = cost;
		}
	}

	void clear() {
		for (size_t i = 0; i < COUNT; i++) {
			way[i] = 0;
		}
	}

	void DFS(int vertex) {
		visited[vertex - delta] = true;
		for (size_t i = 0; i < visited.size(); i++) {
```

```
                        if (!visited[i] && (graph[vertex - delta][i] - flows[vertex - delta][i]
> 0 && graph[vertex - delta][i] != 0 || flows[vertex - delta][i] < 0 && graph[i][vertex -
delta] != 0)) {
                                way[i] = vertex;
                                DFS(i + delta);
                        }
                }
        }

        bool get_way() {
                DFS(source);
                for (size_t i = 0; i < visited.size(); i++) {
                        visited[i] = false;
                }
                return (way[stock - delta] != 0);
        }

        int FF() {
                int max_flow = 0;
                while (get_way()) {
                        int tmp = MAX;
                        for (int v = stock - delta; 0 <= way[v] - delta; v = way[v] - delta) {
                                tmp = min(tmp, graph[way[v] - delta][v] - flows[way[v] -
delta][v]);
                        }
                        for (int v = stock - delta; 0 <= way[v] - delta; v = way[v] - delta) {
                                flows[way[v] - delta][v] += tmp;
                                flows[v][way[v] - delta] -= tmp;
                        }
                        max_flow += tmp;
                        clear();
                }
                return max_flow;
        }

        void print_result() {
                for (int i = 0; i < COUNT; i++) {
                        for (int j = 0; j < COUNT; j++) {
                                if (flows[i][j] != 0 && flows[i][j] < 0)
                                        flows[i][j] = 0;
                                if (graph[i][j] > 0)
                                        cout << (char)(i + delta) << " " << (char)(j + delta) << "
" << flows[i][j] << endl;
                        }
                }
        }
};


int main() {
        Ford_Falkerson A(0, '\0');
        A.creation_graph();
        cout <<A.FF()<< endl;
        A.print_result();
        system("pause");
        return 0;
        }
```