МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «ПиАА»

Тема: Потоки в сети

Студент(ка) гр. 0000 _____ Ивченко А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с алгоритмом Форда-Фалкерсона нахождения максимального потока 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).

Вар. 6. Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего друг к другу. Если таких дуг несколько, то выбрать ту, имя конца которой в алфавите ближайшее к началу алфавита.

Структуры данных.

Ориентированный взвешенный граф представляется в виде словаря, в котором по названию вершины можно определить список смежных дуг для этой вершины. Дуга представляется в виде структуры с полями: конечная вершина дуги, остаточная пропускная способность, поток.

```
struct  Edge {
char end_v;
int capacity;
int flow;
};
std::map<char, vector<Edge>> graph;
```

Описание алгоритма.

В теле главного цикла проверяется наличие смежных дуг у текущей вершины в остаточной пропускной способностью > 0. Если такие есть, то переход к следующей вершине осуществляется согласно правилу, описанному в

варианте №6 до тех пор пока не будет найден дополняющий путь от начальной вершины к конечной. В противном случае, если текущая вершина не является стартовой, то дуга, ведущая к ней, удаляется из множества дуг ее родителя.

В качестве промежуточной информации выводятся значения следующих вершин и значение остатоточной пропускной способности дуги, ведущей к этой вершине.

Тестирование

Рисунок 1 - 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
```

Рисунок 2 - результат работы программы

```
b:7
d:6
e:3
c:2
f:9
b:5
d:4
e:1
f:4
b:1
c:6
f:7
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 был разобран алгоритм Форда-Фалкерсона нахождения максимального потока в ориентированном взвешенном графе.

Исходный код

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#include <math.h>
#include <fstream>

using namespace std;

struct  Edge {
	char end_v;
	int capacity;
	int flow;
};

void FordFulkersonAlgorithm(std::map<char, vector<Edge>>& graph, char start, char  end,
std::ostream& out) {

	char i = start;
	int minElement = 0;
	std::vector<int> maxflowlist;
	std::vector<char> visited;
	std::map<char, char> parent;
	std::vector<int> flowlist;
	std::vector<char> path;
	visited.push_back(i);
	path.push_back(i);
	char next;

	while (i) {

		int empty = 1;
		for (int q = 0; q < graph[i].size(); q++) {
			if ((graph[i])[q].capacity > 0) { empty = 0; break; }
		}
		if (empty) {

			if (i == start) {
				int sum = 0;
				for (int q = 0; q < maxflowlist.size(); q++) {
					sum += maxflowlist[q];

				}cout << sum << endl;
				for (char i = start; i < end; i++) {
					for (int j = 0; j < graph[i].size(); j++) {
						out << i << ' ' << graph[i][j].end_v << ' ' <<
graph[i][j].flow;
						out << endl;
					}
				}return;
			}
			else {
				char par = parent[i];
				for (int k = 0; k < graph[par].size(); k++) {
					if ((graph[par])[k].end_v == i)
						graph[par][k].capacity = 0;
				}i = par;
				path.pop_back();
				flowlist.pop_back();
```

```
                        continue;
                }
        }
        int cap = 0;
        int mindiff = 1000;

        for (auto j : graph[i]) {

                int vis = 0;
                for (int k = 0; k < visited.size(); k++) {
                        if (j.end_v == visited[k])
                                vis = 1;
                }
                if (vis) continue;

                int diff = abs(j.end_v - i);
                if (diff < mindiff) {

                        mindiff = diff;
                        next = j.end_v;
                        cap = j.capacity;
                }
                else if (diff == mindiff) {
                        if (j.end_v < next) {
                                next = j.end_v;
                                cap = j.capacity;
                        }
                }

        }cout << next << ':' << cap << endl;
        flowlist.push_back(cap);
        parent[next] = i;
        path.push_back(next);
        visited.push_back(next);

        if (next == end) {
                int minElement = 1000;
                for (auto a : flowlist) {
                        if (a < minElement) minElement = a;
                }
                for (int k = 0; k < path.size() - 1; k++) {
                        for (int j = 0; j < graph[path[k]].size(); j++) {
                                if (graph[path[k]][j].end_v == path[k + 1]) {
                                        graph[path[k]][j].capacity = graph[path[k]]
[j].capacity - minElement;
                                        graph[path[k]][j].flow = graph[path[k]][j].flow +
minElement;
                                }
                        }
                }
                i = start;
                maxflowlist.push_back(minElement);
                flowlist.clear();
                path.clear();
                path.push_back(start);
                visited.clear();
                visited.push_back(start);
        }
        else { i = next; }
    }
}
int main() {

        int chose;
```

```
std::cout << "console: 0, file :1" << std::endl;
std::cin >> chose;

char u, v, source, destination;
int cap, n;
std::map<char, vector<Edge>> graph;
std::cin >> n;
std::cin >> source;
std::cin >> destination;

for (int i = 0; i < n; i++) {
	cin >> u >> v >> cap;
	Edge edge{ v, cap, 0 };
	graph[u].push_back(edge);
}

if (chose == 0) {

	FordFulkersonAlgorithm(graph, source, destination, std::cout);
}
else {
	std::ofstream file;
	file.open("result.txt");

	if (!file.is_open()) {
		std::cout << "Incorrect!\n";
		return 0;
	}
	else {

		FordFulkersonAlgorithm(graph, source, destination, file);
	}
}

return 0;
}
```