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 _____ Корякин М.П.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

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

Формулировка задания.

Сеть (ориентированный взвешенный граф) представляется в видетриплета из имён вершин и целого неотрицательного числа - пропускной способности (веса).

Входные данные:

N - количество ориентированных рёбер графа

v_0 - исток

v_n - сток

v_i v_j w_{ij} - ребро графа

v_i v_j w_{ij} - ребро графа

...

Выходные данные:

P_{max} - величина максимального потока

v_i v_j w_{ij} - ребро графа с фактической величиной протекающего потока.

v_i v_j w_{ij} - ребро графа с фактической величиной протекающего потока

...

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).

Описание работы алгоритма.

Изначально инициализируется матрица путей графа, которая состоит из 26(т.к. 26 букв в английском алфавите) ячеек, все буквы идут по порядку. Ненужные ячейки остаются нулями.

Далее, алгоритм заходит в функцию поиска возможного пути из истока в сток. Если такой поток существует, то мы его записываем в вектор way, учитывая условия проходимости этого пути, т.е. учитываем есть ли туда путь, который еще не «иссяк». Если мы дошли до стока, то мы записываем в новую матрицу вес минимального ребра в этом пути во все ячейки графа, которые соответствовали этому пути. Минимальное значение ребра в пути сохраняется в переменную, далее он будет суммироваться с последующими найденными потоками. После чего мы вновь ищем доступный поток в графе, учитывая то, что один поток уже создан. Так будет продолжаться, пока алгоритм не сможет найти доступный путь до стока.

Индивидуальное задание выполняется по определению, заданному в инициализации матрицы. Так как матрица соответствует алфавитному порядку, то и пути будут находится непосредственно ближайшие по коду ASCII.

Результат 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 результате выполнения лабораторной работы был изучен и реализован на языке C++ алгоритм Форда – Фалкерсона. Были найдены максимальный поток в сети, а также фактические потоки всех ребер в графе.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
#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 )) {
                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)
                    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();

    return 0;
}
```