

**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: Потoki в cети**  
**Вариант 4**

Студент гр. 8382

Торосян Т.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Изучить принцип работы алгоритма Форда-Фалкерсона и реализовать его на практике для нахождения 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 (взвешенный орграф) представляется в виде триплета 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

$v_i v_j w_{ij}$  - ребро графа

...

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).

Вариант 4. Поиск в глубину. Итеративная реализация.

### **Описание алгоритма**

- 1 Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной

сетью.

2 В остаточной сети находим любой путь из источника в сток. Если такого пути нет, останавливаемся.

3 Пускаем через найденный путь максимально возможный поток:

3.1 На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью  $c_{min}$ .

3.2 Для каждого ребра на найденном пути увеличиваем поток на  $c_{min}$ , а в противоположном ему — уменьшаем на то же значение.

3.3 Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной части, а если обнулилась, стираем его.

4 Возвращаемся на шаг 2.

Чтобы найти кратчайший путь в графе, используем поиск в глубину:

1 Создается стек вершин  $O$ . Вначале  $O$  состоит из единственной вершины  $s$ .

2 Вершина  $s$  отмечается как посещенная без родителя, а все остальные как непосещенные.

3 Пока стек не пуст, выполняются следующие шаги:

3.1 Удаляется вершина  $i$  на стеке

3.2 Для всех дуг  $(i, j)$ , исходящих из  $i$ , для которых вершина  $j$  ещё не посещена, выполняются следующие шаги

3.2.1 Вершина  $i$  отмечается как посещенная с родителем  $j$

3.2.2 Вершина  $v$  добавляется в стек

3.2.3 Если  $j = t$  (текущая вершина-сток), то

производится выход из всех циклов, т. к. путь был найден

- 4 Если стек пуст, возвращается ответ об отсутствии пути
- 5 Иначе производится проход от  $t$  к  $s$  с переходом к родителю. Путь возвращается в обратном порядке.

### **Хранение частичных решений**

```
struct answer {  
  
    char from;  
  
    char to;  
  
    int weight;  
  
};
```

Структура для ответа в формате стефика. Описывает ребро графа из  $from$  в  $to$  с весом  $weight$ .

`vector<vector<int>> graph` — двумерный вектор-контейнер для рёбер.

Функция `void printGraph(vector<vector<int>> graph, int V, string nodeNames)` используется для вывода графа в отладочных целях.

`Vector <answer> forStepik` — вектор рёбер в понятном степику формате.

### **Описание функций**

- `int FFA(vector<vector<int>>& graph, vector<vector<int>>& rGraph, int s, int t, int V, string nodeNames)` — основная функция программы, реализующая алгоритм Форда-Фалкерсона. Принимает на вход граф `graph`, граф смежности `rGraph`, вершины истока  $s$  и стока  $t$ , количество узлов  $V$  и строку с их названиями `nodeNames`.

На этапе инициализации алгоритма данные из `graph` перемещаются в `rGraph`, с которым и будет в дальнейшем идти работа. Освободившийся `graph` используется для записи результатов.

Далее запускается основной цикл, который работает, пока ф-ция DFS не найдет путь от истока к стоку. В этом случае путь записывается в массив

parent.

После этого пути рассматриваются ещё раз, чтобы вычесть из пропускной способности рёбер минимальную пропускную способность и прибавить её ребрам тех же вершин, идущих в противоположном направлении.

Функция возвращает значение максимального потока в сети.

- Bool DFS (vector<vector<int>> rGraph, int s, int t, vector<int>& parent, int V, string nodeNames) — имеет те же аргументы, что и FFA, но с добавлением вектора parent, в который записывается путь от истока в сток.

Эта функция итеративно ищет путь обходом в глубину и записывает его в parent. Возвращает true, если путь найден, иначе — false.

### **Сложность алгоритма по времени**

Каждый путь находится поиском в глубину за  $O(E)$ , количество таких итераций не превосходит  $O(VE)$ . Поэтому сложность по времени можно оценить как  $O(VE^2)$ .

### **Сложность алгоритма по памяти**

Т.к. Граф представлен в памяти в виде матрицы смежности, сложность по памяти можно оценить как  $O(V^2)$ .

### **Тестирование**

Пример вывода для 1-го теста степика:

```

7 a f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
Поиск пути...
Вершина a:
    Добавление смежной вершины b в стек
    Добавление смежной вершины c в стек
Завершение вычислений для вершины a
Вершина c:
    Добавление смежной вершины f в стек
Завершение вычислений для вершины c
Вершина f:
    Завершение вычислений для вершины f
Вершина b:
    Добавление смежной вершины d в стек
Завершение вычислений для вершины b
Вершина d:
    Добавление смежной вершины e в стек
Завершение вычислений для вершины d
Вершина e:
    Завершение вычислений для вершины e

Найден путь ascf
Поток на пути: 6
<)=====(>)
Поиск пути...
Вершина a:
    Добавление смежной вершины b в стек
Завершение вычислений для вершины a
Вершина b:
    Добавление смежной вершины d в стек
Завершение вычислений для вершины b
Вершина d:
    Добавление смежной вершины e в стек
    Добавление смежной вершины f в стек
Завершение вычислений для вершины d
Вершина f:
    Добавление смежной вершины c в стек
Завершение вычислений для вершины f
Вершина c:
    Завершение вычислений для вершины c
Вершина e:
    Завершение вычислений для вершины e

Найден путь abdf
Поток на пути: 4
<)=====(>)
Поиск пути...
Вершина a:
    Добавление смежной вершины b в стек
Завершение вычислений для вершины a
Вершина b:
    Добавление смежной вершины d в стек
Завершение вычислений для вершины b
Вершина d:
    Добавление смежной вершины e в стек
Завершение вычислений для вершины d
Вершина e:
    Добавление смежной вершины c в стек
Завершение вычислений для вершины e
Вершина c:
    Добавление смежной вершины f в стек
Завершение вычислений для вершины c
Вершина f:
    Завершение вычислений для вершины f

Найден путь abdescf
Поток на пути: 2
<)=====(>)
Поиск пути...
Вершина a:
    Добавление смежной вершины b в стек
Завершение вычислений для вершины a
Вершина b:
    Завершение вычислений для вершины b

Путь не был найден
Завершение алгоритма...
<)=====(>)
Ответ для стефика:
12
a b 6
a c 6
b d 6
c f 8
d e 2

```

Пример 2:

Ввод : 1 f f

Вывод: вершина не может быть истоком и стоком одновременно.

Пример 3:

Ввод:

6 a e

a b 12

b c 5

a c 13

c d 14

c e 3

d e 15

Вывод:

17

a b 4

a c 13

b c 4

c d 14

c e 3

d e 14

**Вывод**

В ходе выполнения лабораторной работы был реализован алгоритм Форда-Фалкерсона для нахождения максимального потока в сети с итеративным обходом в глубину. Была произведена оценка сложности данного алгоритма, а также протестирована практическая реализация.

# ПРИЛОЖЕНИЕ

## КОД ПРОГРАММЫ

### ФАЙЛ main.cpp

```
#include <iostream>

#include <limits.h>

#include <vector>

#include <stack>

#include <iomanip>

#include <algorithm>

using namespace std;

#define LINE "<"=====(>"

struct answer {

    char from; //вершина, из которой выходит ребро

    char to;   //вершина, в которую входит ребро

    int weight; //вес ребра

};

bool DFS(vector<vector<int> > rGraph, int s, int t, vector<int>& parent, int V, string nodeNames); //возвращаетtrue, если
существует путь от истока s к стоку t в графе rGraph, также заполняет массив parent

int FFA(vector<vector<int> >& graph, vector<vector<int> >& rGraph, int s, int t, int V, string nodeNames); //возвращает
максимальный поток от истока s к стоку t

void printGraph(vector<vector<int> > graph, int V, string nodeNames); //для печати графа

bool cmpForStepik(answer a, answer b); //компаратор для сортировки вершин

int main()

{

    setlocale(LC_ALL, "RUS");

    int N = 0;    //количество ориентированных рёбер графа

    char start; //исток

    char finish; //сток

    string nodeNames; //названия узлов

    string from;

    string to;

    vector<int> w;

    cin >> N;

    cin >> start;

    cin >> finish;
```



```

if (start == finish) {
    cout << "Сток и исток не могут быть одной вершиной" << endl;
    return 0;
}

char tmpFrom;
char tmpTo;
int tmpW;
nodeNames += start;
for (int i = 0; i < N; i++) {
    cin >> tmpFrom >> tmpTo >> tmpW;
    from += tmpFrom;
    to += tmpTo;
    w.push_back(tmpW);
    if (nodeNames.length() == 0) nodeNames += tmpTo;
    else if (nodeNames.find(tmpTo) == string::npos) nodeNames += tmpTo;
}

sort(nodeNames.begin(), nodeNames.end());

int V = nodeNames.length();

vector<vector<int> > graph(V, vector<int>(V, 0));
string findInNodeNames;

for (size_t k = 0; k < nodeNames.length(); k++) {
    vector<int> tmp;
    for (int j = 0; j < N; j++) {        //поиск всех ребер, ведущих из вершины nodeNames[k]
        if (from[j] == nodeNames[k]) {
            tmp.push_back(j);
        }
    }
}

vector<int> nodeTmp;
for (size_t i = 0; i < tmp.size(); i++) {    //поиск в строке nodeNames[k] вершины, в которую ведут ребра из вектора tmp
    for (size_t j = 0; j < nodeNames.length(); j++) {
        if (nodeNames[j] == to[tmp[i]])
            nodeTmp.push_back(j);
    }
}
}

```

```

    for (size_t i = 0; i < tmp.size(); i++) {
        graph[k][nodeTmp[i]] = w[tmp[i]];
    }
}

int startIndex = 0;
int finishIndex = 0;
for (int i = 0; i < V; i++) {
    if (nodeNames[i] == start) startIndex = i;
    else if (nodeNames[i] == finish) finishIndex = i;
}

vector<vector<int> > rGraph(V, vector<int>(V, 0));
int maxFlow = FFA(graph, rGraph, startIndex, finishIndex, V, nodeNames);

vector <answer> forStepik;

for (int i = 0; i < V; i++) {
    vector <int> indices; //индексы
    for (int j = 0; j < N; j++) {
        if (nodeNames[i] == from[j]) indices.push_back(j);
    }

    for (size_t j = 0; j < indices.size(); j++) {
        answer ans;
        ans.from = from[indices[j]];
        ans.to = to[indices[j]];

        int tmpF = 0;
        int tmpT = 0;
        for (int k = 0; k < V; k++) {
            if (nodeNames[k] == from[indices[j]]) tmpF = k;
            else if (nodeNames[k] == to[indices[j]]) tmpT = k;
        }

        if (graph[tmpT][tmpF] >= 0) ans.weight = 0;
        else ans.weight = abs(graph[tmpT][tmpF]);
        forStepik.push_back(ans);
    }
}

sort(forStepik.begin(), forStepik.end(), cmpForStepik);

```

```

        cout << "Ответ для степика:" << endl;

        cout << maxFlow << endl;

        for (size_t i = 0; i < forStepik.size(); i++) cout << forStepik[i].from << " " << forStepik[i].to << " " << forStepik[i].weight <<
endl;

        cout << LINE << endl;

        cout << "Максимальный поток в сети = " << maxFlow << endl;

        return 0;
    }

    bool DFS(vector<vector<int> > rGraph, int s, int t, vector<int>& parent, int V, string nodeNames) //функция поиска пути в
    глубину
    {
        vector<bool> visited(V, 0); //вектор посещенных вершин (если 0, то не посещена)

        stack<int> st; //создается стек, в который кладется исток, и начальная вершина помечается как посещенная
        st.push(s);
        visited[s] = true;
        parent[s] = -1;

        cout << "Поиск пути..." << endl;
        while (!st.empty()) { //обработка, пока стек не пуст
            int i = st.top(); //обработка первой вершины
            st.pop();

            cout << "Вершина" << nodeNames[i] << ":" << endl;

            for (int j = 0; j < V; j++) { //если смежная вершина не обработана и имеет ребро с обрабатываемой вершиной
                if (visited[j] == false && rGraph[i][j] > 0) {
                    st.push(j);
                    parent[j] = i;
                    visited[j] = true;
                    cout << "\tДобавление смежной вершины" << nodeNames[j] << " в стек" << endl;
                }
            }
        }

        cout << "Завершение вычислений для вершины" << nodeNames[i] << endl;
    }

    cout << endl;

    if (visited[t] == true) {
        cout << "Найден путь";
    }

```

```

        string str;

        for (int i = t; i != s; i = parent[i])

            str += nodeNames[i];

        str += nodeNames[s];

        for (int i = str.length() - 1; i >= 0; i--)

            cout << str[i];

        cout << endl;

    } else cout << "Путь не был найден" << endl;

    return (visited[t] == true); //если был достигнут сток, то возвращается true, иначе - false
}

int FFA(vector<vector<int> >& graph, vector<vector<int> >& rGraph, int s, int t, int V, string nodeNames)
{
    int u, v;

    for (u = 0; u < V; u++)

        for (v = 0; v < V; v++) {

            rGraph[u][v] = graph[u][v];

            graph[u][v] = 0;

        }

    vector <int> parent(V, 0); //этот массив заполняется функцией BFS и создан для хранения пути

    int max_flow = 0; //изначально поток = 0

    while (DFS(rGraph, s, t, parent, V, nodeNames)) { //увеличивается поток, пока есть путь от истока к стоку

        int path_flow = INT_MAX;

        for (v = t; v != s; v = parent[v]) {

            u = parent[v];

            path_flow = min(path_flow, rGraph[u][v]);

        }

        cout << "Поток на пути:" << path_flow << endl;

        cout << LINE << endl;

        for (v = t; v != s; v = parent[v]) {

            //обновление пропускной способности каждого ребра

```

```

        u = parent[v];
        rGraph[u][v] -= path_flow;
        rGraph[v][u] += path_flow;

        graph[u][v] += path_flow;
        graph[v][u] -= path_flow;
    }

    max_flow += path_flow;

}

cout << "Завершение алгоритма..." << endl;
cout << LINE << endl;
return max_flow;
}

void printGraph(vector<vector<int> > graph, int V, string nodeNames)
{
    cout << " ";
    for (int i = 0; i < V; i++) {
        cout << setw(3) << nodeNames[i] << " ";
    }
    cout << endl;
    for (int i = 0; i < V; i++) {
        cout << nodeNames[i];
        for (int j = 0; j < V; j++) cout << setw(3) << graph[i][j] << " ";
        cout << endl;
    }
    cout << endl;
}

bool cmpForStepik(answer a, answer b)
{
    if (a.from < b.from) return true;
    else if (a.from == b.from) {
        if (a.to < b.to) return true;
    }

    return false;
}

```