

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.

Студентка гр. 8382

Наконечная А. Ю.

Преподаватель

Фирсов М. А.

Санкт-Петербург

2020

### Цель работы.

Изучить работу и реализовать алгоритм Форда-Фалкерсона для 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). Пример входных и выходных данных представлен на Рис. 1.

Входные данные:

$N$  - количество ориентированных рёбер графа

$v_0$  - источник

$v_n$  - сток

$v_i \quad v_j \quad \omega_{ij}$  - ребро графа

$v_i \quad v_j \quad \omega_{ij}$  - ребро графа

...

Выходные данные:

$P_{max}$  - величина максимального потока

$v_i \quad v_j \quad \omega_{ij}$  - ребро графа с фактической величиной протекающего потока

$v_i \quad v_j \quad \omega_{ij}$  - ребро графа с фактической величиной протекающего потока

...

Рис. 1 — Пример входных и выходных данных

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).

**Sample Input:**

7  
a  
f  
a b 7  
a c 6  
b d 6  
c f 9  
d e 3  
d f 4  
e c 2

**Sample Output:**

12  
a b 6  
a c 6  
b d 6  
c f 8  
d e 2  
d f 4  
e c 2

**Индивидуальное задание.**

Вариант 4.

Поиск в глубину. Итеративная реализация.

### **Описание алгоритма.**

#### **Алгоритм Форда-Фалкерсона.**

1. Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью. Максимальный поток приравнивается к 0.
2. В остаточной сети находим любой путь из источника в сток с помощью итеративного поиска в глубину (алгоритм функции `dfs()` описан ниже). 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.
  - 3.2. Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность, добавляем к максимальному потоку значение потока, полученного на данной итерации.
4. Возвращаемся на шаг 2.

#### **Алгоритм итеративного поиска в глубину.**

1. Для реализации алгоритма используется структура данных стек. Поиск начинается с истока графа.
2. Далее рассматривается вершина  $j$  смежная с  $s$ .
3. Вершина выбирается, отмечается как посещённая, заносится в путь, а также заносится в стек.
4. Остальные смежные вершины (если они есть, и они не посещены) отправляются в стек и ожидают следующего захода в родительскую вершину.
5. Далее берется вершина  $q$  смежная с  $v$ . Действия повторяются. Так процесс будет продвигаться вглубь графа пока стек не останется пуст.

### Описание структур данных.

Структура Edge, описывающая ребро. Используется для вывода фактических величин потоков, проходящих через каждое ребро графа.

```
struct Edge{  
    char fromTop;//идет из вершины  
    char toTop;//идет в вершину  
    int weight; //вес ребра  
};
```

vector<vector<int>> graph — двумерный вектор, в котором хранятся рёбра графа, используется для вывода ответа.

vector<vector<int>> Graph — как и структура graph хранит в себе рёбра, используется для сохранения значений из структуры graph.

vector<int>parent — вектор, используемый для хранения пути.

vector<bool>visited — массив флагов посещаемости вершин.

stack<int>st — стек, используемый в итеративном поиске в глубину.

### Описание функций.

Функция bool compare(Edge a, Edge b) — сортирует данные в лексикографическом порядке. На вход принимает данные типа Edge.

Функция int fordFulkerson(vector<vector<int>>& graph, vector<vector<int>>& Graph, int s, int t, int vertexesNum, const string& nodes) — на вход принимает граф graph, в котором хранятся ребра; Graph — копию графа, s — исток, t — сток, vertexesNum — количество узлов, nodes — названия узлов. В начале функции значения graph копируются, при этом граф обнуляется. Работа в функции производится с Graph. Далее запускается цикл, который работает до тех пор, пока функция dfs находит путь от истока в сток в сети. Если путь найден, то он записывается в массив parent. Затем находится минимальное значение среди остаточных пропускных способностей рёбер, которые входят в текущий сквозной путь. Далее происходит обновление про-

пускных способностей каждого ребра. Функция возвращает значение максимального потока в сети.

Функция `bool dfs(const vector<vector<int>>& Graph, int s, int t, vector<int>&parent, int vertexesNum, const string& nodes)` — на вход принимает все то же самое, что и функция `fordFulkerson`, за исключением вектора `parent`, в который записывается путь от истока в сток. Эта функция ищет путь итеративным обходом в глубину в сети и записывает его в массив `parent`. Функция возвращает `true`, если путь найден, и `false`, если путь не был найден.

### **Сложность алгоритма.**

По времени.

Сложность алгоритма по времени можно оценить как  $O(VE^2)$ .

Так как каждый путь находится поиском в глубину со сложностью  $O(E)$ , общее число итераций в цикле `while` алгоритма не превосходит  $O(VE)$ , следовательно, временную сложность алгоритма можно оценить как  $O(VE^2)$ .

По памяти.

Сложность алгоритма по памяти можно оценить как  $O(V^2)$ . Такая оценка исходит из того, что программа хранит матрицу смежности графа.

### **Тестирование.**

№ теста	Тест	Результат
1	7 a f a b 7	Adding an edge which goes from: a to: b with weight: 7 Adding an edge which goes from: a to: c with weight: 6

a c 6	Adding an edge which goes from: b to: d with weight:
b d 6	6
c f 9	Adding an edge which goes from: c to: f with weight:
d e 3	9
d f 4	Adding an edge which goes from: d to: e with weight:
e c 2	3
	Adding an edge which goes from: d to: f with weight:
	4
	Adding an edge which goes from: e to: c with weight:
	2
	Line view with sorted vertexes: abcdef
	Number of nodes: 6
	Graph creation in progress . . .
	From vertex: 0 to vertex: 1 weight is: 7
	From vertex: 0 to vertex: 2 weight is: 6
	From vertex: 1 to vertex: 3 weight is: 6
	From vertex: 2 to vertex: 5 weight is: 9
	From vertex: 3 to vertex: 4 weight is: 3
	From vertex: 3 to vertex: 5 weight is: 4
	From vertex: 4 to vertex: 2 weight is: 2
	Start index is: 0 Finish index is: 5
	Begin of Ford-Fulkerson algorithm
	Zeroing a graph
	Start searching for paths from start to finish
	Current vertex is: b
	Current vertex is: c
	Current vertex is: f

		<p>Current vertex is: d</p> <p>Current vertex is: e</p> <p>Now the throughput is as follows:</p> <p>Graph[c][f] = 3</p> <p>Graph[f][c] = 6</p> <p>graph[c][f] = 6</p> <p>graph[f][c] = 6</p> <p>Graph[a][c] = 0</p> <p>Graph[c][a] = 6</p> <p>graph[a][c] = 6</p> <p>graph[c][a] = 6</p> <p>Current vertex is: b</p> <p>Current vertex is: d</p> <p>Current vertex is: e</p> <p>Current vertex is: f</p> <p>Current vertex is: c</p> <p>Now the throughput is as follows:</p> <p>Graph[d][f] = 0</p> <p>Graph[f][d] = 4</p> <p>graph[d][f] = 4</p> <p>graph[f][d] = 4</p> <p>Graph[b][d] = 2</p> <p>Graph[d][b] = 4</p> <p>graph[b][d] = 4</p> <p>graph[d][b] = 4</p> <p>Graph[a][b] = 3</p> <p>Graph[b][a] = 4</p> <p>graph[a][b] = 4</p>
--	--	--



		graph[b][a] = 4 Current vertex is: b Current vertex is: d Current vertex is: e Current vertex is: c Current vertex is: f Now the throughput is as follows: Graph[c][f] = 1 Graph[f][c] = 8 graph[c][f] = 8 graph[f][c] = 8 Graph[e][c] = 0 Graph[c][e] = 2 graph[e][c] = 2 graph[c][e] = 2 Graph[d][e] = 1 Graph[e][d] = 2 graph[d][e] = 2 graph[e][d] = 2 Graph[b][d] = 0 Graph[d][b] = 6 graph[b][d] = 6 graph[d][b] = 6 Graph[a][b] = 1 Graph[b][a] = 6 graph[a][b] = 6 graph[b][a] = 6 Current vertex is: b
--	--	--

		<p>End of Ford-Fulkerson algorithm</p> <p>Max flow is: 12</p> <p>a b 6</p> <p>a c 6</p> <p>b d 6</p> <p>c f 8</p> <p>d e 2</p> <p>d f 4</p> <p>e c 2</p>
2	<p>9</p> <p>a</p> <p>d</p> <p>a b 8</p> <p>b c 10</p> <p>c d 10</p> <p>h c 10</p> <p>e f 8</p> <p>g h 11</p> <p>b e 8</p> <p>a g 10</p> <p>f d 8</p>	<p>Adding an edge which goes from: a to: b with weight: 8</p> <p>Adding an edge which goes from: b to: c with weight: 10</p> <p>Adding an edge which goes from: c to: d with weight: 10</p> <p>Adding an edge which goes from: h to: c with weight: 10</p> <p>Adding an edge which goes from: e to: f with weight: 8</p> <p>Adding an edge which goes from: g to: h with weight: 11</p> <p>Adding an edge which goes from: b to: e with weight: 8</p> <p>Adding an edge which goes from: a to: g with weight: 10</p> <p>Adding an edge which goes from: f to: d with weight: 8</p>

		<p>Line view with sorted vertexes: abcdefgh</p> <p>Number of nodes: 8</p> <p>Graph creation in progress . . .</p> <p>From vertex: 0 to vertex: 1 weight is: 8</p> <p>From vertex: 0 to vertex: 6 weight is: 10</p> <p>From vertex: 1 to vertex: 2 weight is: 10</p> <p>From vertex: 1 to vertex: 4 weight is: 8</p> <p>From vertex: 2 to vertex: 3 weight is: 10</p> <p>From vertex: 4 to vertex: 5 weight is: 8</p> <p>From vertex: 5 to vertex: 3 weight is: 8</p> <p>From vertex: 6 to vertex: 7 weight is: 11</p> <p>From vertex: 7 to vertex: 2 weight is: 10</p> <p>Start index is: 0 Finish index is: 3</p> <p>Begin of Ford-Fulkerson algorithm</p> <p>Zeroing a graph</p> <p>Start searching for paths from start to finish</p> <p>Current vertex is: b</p> <p>Current vertex is: g</p> <p>Current vertex is: h</p> <p>Current vertex is: c</p> <p>Current vertex is: d</p> <p>Current vertex is: e</p> <p>Current vertex is: f</p> <p>Now the throughput is as follows:</p> <p>Graph[c][d] = 0</p> <p>Graph[d][c] = 10</p> <p>graph[c][d] = 10</p> <p>graph[d][c] = 10</p>
--	--	--

		<p> <math>\text{Graph}[h][c] = 0</math>  <math>\text{Graph}[c][h] = 10</math>  <math>\text{graph}[h][c] = 10</math>  <math>\text{graph}[c][h] = 10</math>  <math>\text{Graph}[g][h] = 1</math>  <math>\text{Graph}[h][g] = 10</math>  <math>\text{graph}[g][h] = 10</math>  <math>\text{graph}[h][g] = 10</math>  <math>\text{Graph}[a][g] = 0</math>  <math>\text{Graph}[g][a] = 10</math>  <math>\text{graph}[a][g] = 10</math>  <math>\text{graph}[g][a] = 10</math>  Current vertex is: b  Current vertex is: c  Current vertex is: e  Current vertex is: f  Current vertex is: d  Current vertex is: h  Current vertex is: g  Now the throughput is as follows:  <math>\text{Graph}[f][d] = 0</math>  <math>\text{Graph}[d][f] = 8</math>  <math>\text{graph}[f][d] = 8</math>  <math>\text{graph}[d][f] = 8</math>  <math>\text{Graph}[e][f] = 0</math>  <math>\text{Graph}[f][e] = 8</math>  <math>\text{graph}[e][f] = 8</math>  <math>\text{graph}[f][e] = 8</math> </p>
--	--	---

		<p>Graph[b][e] = 0</p> <p>Graph[e][b] = 8</p> <p>graph[b][e] = 8</p> <p>graph[e][b] = 8</p> <p>Graph[a][b] = 0</p> <p>Graph[b][a] = 8</p> <p>graph[a][b] = 8</p> <p>graph[b][a] = 8</p> <p>End of Ford-Fulkerson algorithm</p> <p>Max flow is: 18</p> <p>a b 8</p> <p>a g 10</p> <p>b c 0</p> <p>b e 8</p> <p>c d 10</p> <p>e f 8</p> <p>f d 8</p> <p>g h 10</p> <p>h c 10</p>
3	1 a a	0
4	8 a g a b 5 a c 8 b d 3	<p>Adding an edge which goes from: a to: b with weight: 5</p> <p>Adding an edge which goes from: a to: c with weight: 8</p> <p>Adding an edge which goes from: b to: d with weight: 3</p>

c e 6	Adding an edge which goes from: c to: e with weight:
c f 2	6
d g 2	Adding an edge which goes from: c to: f with weight:
e g 5	2
f g 2	Adding an edge which goes from: d to: g with weight:
	2
	Adding an edge which goes from: e to: g with weight:
	5
	Adding an edge which goes from: f to: g with weight:
	2
	Line view with sorted vertexes: abcdefg
	Number of nodes: 7
	Graph creation in progress . . .
	From vertex: 0 to vertex: 1 weight is: 5
	From vertex: 0 to vertex: 2 weight is: 8
	From vertex: 1 to vertex: 3 weight is: 3
	From vertex: 2 to vertex: 4 weight is: 6
	From vertex: 2 to vertex: 5 weight is: 2
	From vertex: 3 to vertex: 6 weight is: 2
	From vertex: 4 to vertex: 6 weight is: 5
	From vertex: 5 to vertex: 6 weight is: 2
	Start index is: 0 Finish index is: 6
	Begin of Ford-Fulkerson algorithm
	Zeroing a graph
	Start searching for paths from start to finish
	Current vertex is: b
	Current vertex is: c

		<p>Current vertex is: e</p> <p>Current vertex is: f</p> <p>Current vertex is: g</p> <p>Current vertex is: d</p> <p>Now the throughput is as follows:</p> <p><math>\text{Graph}[f][g] = 0</math></p> <p><math>\text{Graph}[g][f] = 2</math></p> <p><math>\text{graph}[f][g] = 2</math></p> <p><math>\text{graph}[g][f] = 2</math></p> <p><math>\text{Graph}[c][f] = 0</math></p> <p><math>\text{Graph}[f][c] = 2</math></p> <p><math>\text{graph}[c][f] = 2</math></p> <p><math>\text{graph}[f][c] = 2</math></p> <p><math>\text{Graph}[a][c] = 6</math></p> <p><math>\text{Graph}[c][a] = 2</math></p> <p><math>\text{graph}[a][c] = 2</math></p> <p><math>\text{graph}[c][a] = 2</math></p> <p>Current vertex is: b</p> <p>Current vertex is: c</p> <p>Current vertex is: e</p> <p>Current vertex is: g</p> <p>Current vertex is: f</p> <p>Current vertex is: d</p> <p>Now the throughput is as follows:</p> <p><math>\text{Graph}[e][g] = 0</math></p> <p><math>\text{Graph}[g][e] = 5</math></p> <p><math>\text{graph}[e][g] = 5</math></p> <p><math>\text{graph}[g][e] = 5</math></p>
--	--	---

		<p> <math>\text{Graph}[c][e] = 1</math>  <math>\text{Graph}[e][c] = 5</math>  <math>\text{graph}[c][e] = 5</math>  <math>\text{graph}[e][c] = 5</math>  <math>\text{Graph}[a][c] = 1</math>  <math>\text{Graph}[c][a] = 7</math>  <math>\text{graph}[a][c] = 7</math>  <math>\text{graph}[c][a] = 7</math>  Current vertex is: b  Current vertex is: c  Current vertex is: e  Current vertex is: d  Current vertex is: g  Current vertex is: f  Now the throughput is as follows:  <math>\text{Graph}[d][g] = 0</math>  <math>\text{Graph}[g][d] = 2</math>  <math>\text{graph}[d][g] = 2</math>  <math>\text{graph}[g][d] = 2</math>  <math>\text{Graph}[b][d] = 1</math>  <math>\text{Graph}[d][b] = 2</math>  <math>\text{graph}[b][d] = 2</math>  <math>\text{graph}[d][b] = 2</math>  <math>\text{Graph}[a][b] = 3</math>  <math>\text{Graph}[b][a] = 2</math>  <math>\text{graph}[a][b] = 2</math>  <math>\text{graph}[b][a] = 2</math>  Current vertex is: b </p>
--	--	--



		<p>Current vertex is: c</p> <p>Current vertex is: e</p> <p>Current vertex is: d</p> <p>End of Ford-Fulkerson algorithm</p> <p>Max flow is: 9</p> <p>a b 2</p> <p>a c 7</p> <p>b d 2</p> <p>c e 5</p> <p>c f 2</p> <p>d g 2</p> <p>e g 5</p> <p>f g 2</p>
5	<p>7</p> <p>a</p> <p>f</p> <p>a f 20</p> <p>a b 10</p> <p>b a 5</p> <p>b h 8</p> <p>h f 5</p> <p>h g 4</p> <p>g f 3</p>	<p>Adding an edge which goes from: a to: f with weight: 20</p> <p>Adding an edge which goes from: a to: b with weight: 10</p> <p>Adding an edge which goes from: b to: a with weight: 5</p> <p>Adding an edge which goes from: b to: h with weight: 8</p> <p>Adding an edge which goes from: h to: f with weight: 5</p> <p>Adding an edge which goes from: h to: g with weight: 4</p> <p>Adding an edge which goes from: g to: f with weight: 3</p>

		<p>Line view with sorted vertexes: abfgh</p> <p>Number of nodes: 5</p> <p>Graph creation in progress . . .</p> <p>From vertex: 0 to vertex: 2 weight is: 20</p> <p>From vertex: 0 to vertex: 1 weight is: 10</p> <p>From vertex: 1 to vertex: 0 weight is: 5</p> <p>From vertex: 1 to vertex: 4 weight is: 8</p> <p>From vertex: 3 to vertex: 2 weight is: 3</p> <p>From vertex: 4 to vertex: 2 weight is: 5</p> <p>From vertex: 4 to vertex: 3 weight is: 4</p> <p>Start index is: 0 Finish index is: 2</p> <p>Begin of Ford-Fulkerson algorithm</p> <p>Zeroing a graph</p> <p>Start searching for paths from start to finish</p> <p>Current vertex is: b</p> <p>Current vertex is: f</p> <p>Current vertex is: h</p> <p>Current vertex is: g</p> <p>Now the throughput is as follows:</p> <p>Graph[a][f] = 0</p> <p>Graph[f][a] = 20</p> <p>graph[a][f] = 20</p> <p>graph[f][a] = 20</p> <p>Current vertex is: b</p> <p>Current vertex is: h</p> <p>Current vertex is: f</p> <p>Current vertex is: g</p> <p>Now the throughput is as follows:</p>
--	--	---

		<p> <math>\text{Graph}[h][f] = 0</math>  <math>\text{Graph}[f][h] = 5</math>  <math>\text{graph}[h][f] = 5</math>  <math>\text{graph}[f][h] = 5</math>  <math>\text{Graph}[b][h] = 3</math>  <math>\text{Graph}[h][b] = 5</math>  <math>\text{graph}[b][h] = 5</math>  <math>\text{graph}[h][b] = 5</math>  <math>\text{Graph}[a][b] = 5</math>  <math>\text{Graph}[b][a] = 10</math>  <math>\text{graph}[a][b] = 5</math>  <math>\text{graph}[b][a] = 10</math>  Current vertex is: b  Current vertex is: h  Current vertex is: g  Current vertex is: f  Now the throughput is as follows:  <math>\text{Graph}[g][f] = 0</math>  <math>\text{Graph}[f][g] = 3</math>  <math>\text{graph}[g][f] = 3</math>  <math>\text{graph}[f][g] = 3</math>  <math>\text{Graph}[h][g] = 1</math>  <math>\text{Graph}[g][h] = 3</math>  <math>\text{graph}[h][g] = 3</math>  <math>\text{graph}[g][h] = 3</math>  <math>\text{Graph}[b][h] = 0</math>  <math>\text{Graph}[h][b] = 8</math>  <math>\text{graph}[b][h] = 8</math> </p>
--	--	---

		graph[h][b] = 8 Graph[a][b] = 2 Graph[b][a] = 13 graph[a][b] = 8 graph[b][a] = 13 Current vertex is: b End of Ford-Fulkerson algorithm Max flow is: 28 a b 8 a f 20 b a 0 b h 8 g f 3 h f 5 h g 3
--	--	---

### **Выводы.**

В ходе выполнения лабораторной работы был реализован на языке C++ алгоритм Форда-Фалкерсона. Найден максимальный поток в сети, а также фактическая величина потока, протекающего через каждое ребро.

## ПРИЛОЖЕНИЕ А

### Исходный код программы

#### риаа\_3.cpp

```
#include <iostream>
#include <vector>
#include <stack>
#include <climits>
#include <algorithm>

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;
using std::stack;
using std::min;

//структура для ребра
struct Edge{
    char fromTop;//идет из вершины
    char toTop;//идет в вершину
    int weight; //вес ребра
};

//сортировка вершин
bool compare(Edge a, Edge b){
    if (a.fromTop < b.fromTop) return true;
    else if (a.fromTop == b.fromTop) {
        if (a.toTop < b.toTop) return true;
    }
    return false;
}

//поиск в глубину
bool dfs(const vector<vector<int>>& Graph, int s, int t,
vector<int>&parent, int vertexesNum, const string& nodes){
    //массив флагов посещаемости вершин
    //создаем стек
    vector<bool>visited(vertexesNum,false);
    stack<int>st;
    //кладем исходную вершину в стек
    st.push(s);

    //посетили вершину
```

```

visited[s] = true;
//исток является начальной вершиной
parent[s] = -1;

//обработка, пока стек не пуст
while (!st.empty()) {
    //обработка первой вершины
    int i = st.top();
    st.pop();
    //если смежная вершина не обработана и имеет ребро с обрабаты-
ваемой вершиной
    for(int j = 0 ; j < vertexesNum; j++){
        if(Graph[i][j] > 0 && !visited[j]){
            cout << "Current vertex is: " << nodes[j] << endl;
            //добавляем смежную вершину
            st.push(j);
            //в пути инициализируем смежную вершину и делаем её по-
сещённой
            parent[j] = i;
            visited[j] = true;
        }
    }
}
return visited[t];
}

int fordFulkerson(vector<vector<int>>& graph, vector<vector<int>>&
Graph, int s, int t, int vertexesNum, const string& nodes) {
    int u, v;
    //graph обнуляется, в дальнейшем будет использован для ответа
    cout << "Zeroing a graph" << endl;
    for (u = 0; u < vertexesNum; u++) {
        for (v = 0; v < vertexesNum; v++) {
            Graph[u][v] = graph[u][v];
            graph[u][v] = 0;
        }
    }
    //изначально поток = 0
    int maxFlow = 0;
    // массив для хранения пути
    vector<int>parent(vertexesNum, 0);
    //увеличивается поток, пока есть путь от истока к стоку
    cout << "Start searching for paths from start to finish" << endl;
    while (dfs(Graph, s, t, parent, vertexesNum, nodes)) {

        int pathFlow = INT_MAX;
    }
}

```

```

        //выбор минимального значения пропускной способности для обнов-
ления пропускной способности каждого ребра
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            pathFlow = min(pathFlow, Graph[u][v]);
        }

        //обновление пропускной способности каждого ребра
        cout << "Now the throughput is as follows: " << endl;
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            Graph[u][v] -= pathFlow;
            Graph[v][u] += pathFlow;
            graph[u][v] += pathFlow;
            graph[v][u] -= pathFlow;
            cout << "Graph[" << nodes[u] << "][" << nodes[v] << "] = "
<< Graph[u][v] << endl;
            cout << "Graph[" << nodes[v] << "][" << nodes[u] << "] = "
<< Graph[v][u] << endl;
            cout << "graph[" << nodes[u] << "][" << nodes[v] << "] = "
<< graph[u][v] << endl;
            cout << "graph[" << nodes[v] << "][" << nodes[u] << "] = "
<< Graph[v][u] << endl;
        }
        maxFlow += pathFlow;
    }
    return maxFlow;
}

int main() {
    //исток
    char start;
    //сток
    char finish;
    //ребро графа
    char tempFrom;
    char tempTo;
    //количество ориентированных рёбер графа
    int N = 0;
    //вес графа
    int weight;
    //строки, содержащие пути
    //вершины, из которых пришли
    string from;
    //вершины, в которые вошли
    string to;
    //полный путь
    string nodes;

```

```

//ввод информации
cin >> N >> start >> finish;
if (start == finish) {
    cout << 0;
    return 0;
}
//массив с весами рёбер
vector<int> weightVector;
//добавление начального узла
nodes = nodes + start;
for (int i = 0; i < N; i++) {
    //считывание рёбер и веса
    cin >> tempFrom;
    cin >> tempTo;
    cin >> weight;
    cout << endl << "Adding an edge which goes from: " << tempFrom
<< " to: " << tempTo << " with weight: " << weight;
    //добавление вершины, из которой выходит ребро
    from += tempFrom;
    //добавление вершины, в которую входит ребро
    to += tempTo;
    //добавление веса в массив
    weightVector.push_back(weight);
    //если не будет найдена вершина, в которую идёт ребро, то
добавляем
    if (nodes.find(tempTo) == string::npos) {
        nodes += tempTo;
    }
}
//сортировка строки с вершинами
sort(nodes.begin(), nodes.end());
cout << endl;
cout << "Line view with sorted vertexes: " << nodes << endl;
//кол-во узлов
int vertexesNum = nodes.length();
cout << "Number of nodes: " << vertexesNum << endl;
//создаём граф
//двумерный вектор
vector<vector<int>> graph(vertexesNum, vector<int>(vertexesNum,
0));

cout << "Graph creation in progress . . ." << endl;
//поиск всех ребер, ведущих из вершины nodes[q]
for (int q = 0; q < nodes.length(); q++) {
    vector<int> Temp;
    for (int j = 0; j < N; j++) {
        if (from[j] == nodes[q]) {
            Temp.push_back(j);
        }
    }
}

```



```

    }
    //поиск в строке nodes[q] вершины, в которую ведут ребра из
вектора temp
    vector <int> nodesTemp;
    for (int i : Temp) {
        for (int j = 0; j < nodes.length(); j++) {
            if (nodes[j] == to[i])
                nodesTemp.push_back(j);
        }
    }
    //дополнение графа весом ребра
    for (int i = 0; i < Temp.size(); i++) {
        graph[q][nodesTemp[i]] = weightVector[Temp[i]];
    }

    //печать графа
    for (int i = 0; i < Temp.size(); i++) {
        cout << "From vertex: " << q << " to vertex: " <<
nodesTemp[i] << " weight is: " << weightVector[Temp[i]] << endl;
    }
}
//индексы стока и истока
int startIndex = 0;
int finishIndex = 0;
for (int i = 0; i < vertexesNum; i++) {
    if (nodes[i] == start)
        startIndex = i;
    else if (nodes[i] == finish)
        finishIndex = i;
}
cout << "Start index is: " << startIndex << " Finish index is: " <<
finishIndex << endl;
//граф смежности
vector<vector<int>> Graph(vertexesNum, vector<int>(vertexesNum,
0));
//нахождение максимального потока
cout << "Begin of Ford-Fulkerson algorithm" << endl;
int maxFlow = fordFulkerson(graph, Graph, startIndex, finishIndex,
vertexesNum, nodes);
cout << "End of Ford-Fulkerson algorithm" << endl;

vector <Edge> One;
//проходимся по всем вершинам
for (int i = 0; i < vertexesNum; i++){
    //составление индексов
    vector <int> pointer;
    //проходимся по всем ориентированным рёбрам
    for (int j = 0; j < N; j++) {

```

```

        //если вершина является той, из которой выходит ребро
        if (nodes[i] == from[j])
            //то добавляем новый индекс
            pointer.push_back(j);
    }
    //пробегаем по всем полученным индексам
    for (int j : pointer) {
        //находим переменный значения, чтобы получить доступ к
graph,
        //где хранятся необходимые фактические величины потока
        Edge edges{};
        edges.fromTop = from[j];
        edges.toTop = to[j];
        int tempF = 0;
        int tempT = 0;
        for (int k = 0; k < vertexesNum; k++) {
            if (nodes[k] == from[j])
                tempF = k;
            else if (nodes[k] == to[j])
                tempT = k;
        }
        if (graph[tempT][tempF] >= 0)
            edges.weight = 0;
        else
            edges.weight = abs(graph[tempT][tempF]);
        //запоминаем фактическую величину потока, необходимую для
вывода
        One.push_back(edges);
    }
}
//сортируем данные
sort(One.begin(), One.end(), compare);
cout << "Max flow is: ";
cout << maxFlow << endl;
for (auto & i : One){
    cout << i.fromTop << " " << i.toTop << " " << i.weight << endl;
}
return 0;
}

```