

**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. 7304

\_\_\_\_\_

Овчинников Н.В.

Преподаватель

\_\_\_\_\_

Филатов А.Ю.

Санкт-Петербург

2019

## Цель работы

Изучить алгоритм Форда-Фалкерсона поиска максимального потока в сети и 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.

## Основные теоретические положения

**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 истока = 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 истока, 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.

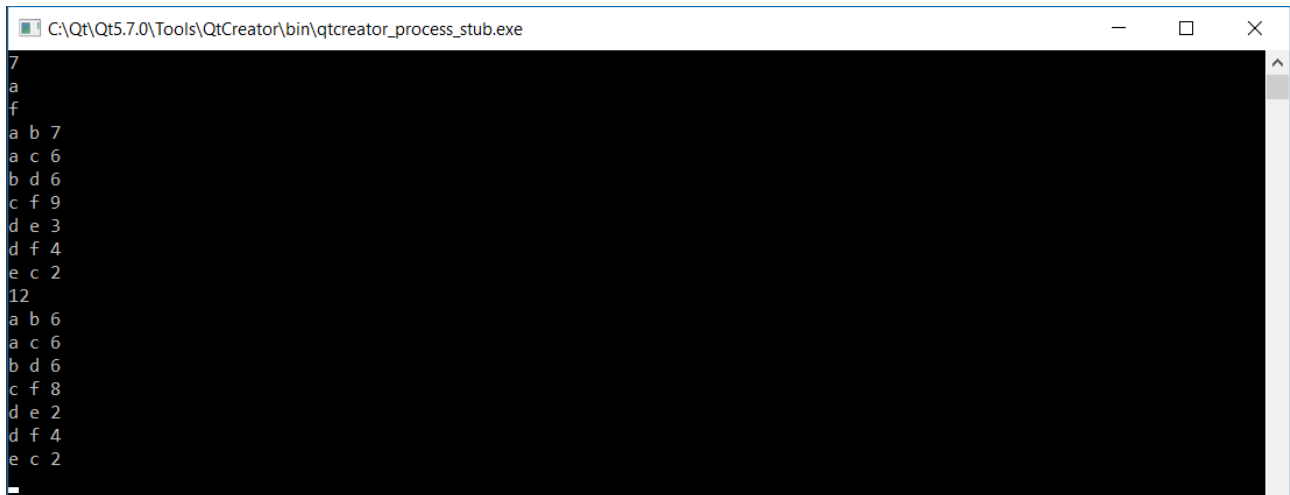
## Ход работы

Был реализован алгоритм поиска максимального потока в сети следующим образом:

1. Создается граф, заданный матрицей смежности, которая хранит содержит структуры, включающие в себя значения пропускной способности и потока. Инициализируется входными значениями.
2. Поиском в глубину от истока ищется путь. Если путь не найден, то переход на шаг 5.
3. В найденном пути вычисляется максимально возможный поток. Он увеличивает поток прямых ребер и уменьшает поток обратных.

4. Найденное значение максимального потока складывается с предыдущим (до начала алгоритма общий максимальный поток равен нулю). Переход на шаг 2.
5. Вывод результатов на экран.

## Пример работы



```
C:\Qt\Qt5.7.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
7
a
f
a b 7
a c 6
b d 6
c f 9
d e 3
d f 4
e c 2
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 выполнения лабораторной работы были изучены такие понятия, как сеть, исток, сток, поток, величина потока, пропускная способность. Был изучен алгоритм Форда-Фалкерсона нахождения максимального потока в сети и реализован на языке программирования C++. Для работы алгоритма использовался обход графа в глубину, для нахождения пути от истока к стоку. При такой реализации время работы алгоритма 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 решению.

## Приложение: исходный код программы

```
#include <QCoreApplication>
#include <iostream>
#include <vector>
#include <limits.h>
#include <algorithm>
#include <stdio.h>
#include <cstdlib>

#define SIZE 128
#define DIGIT 0

using namespace std;

int comp(const void *aa, const void *bb);

class Pair
{
public:
    unsigned char from;
    unsigned char to;
};

class Edge
{
public:
    int capacity = 0;
    int stream = 0;
};

class FordFulkerson
{
private:
    Edge **graph;
    unsigned char source;
    unsigned char stock;
    unsigned int edgeCount;
    vector<unsigned char> tmpForWays;
    vector<unsigned char> blocked;
    int maxStream = 0;
    int minStreamInWay;
    Pair *result;

public:
    FordFulkerson()
    {
        graph = new Edge* [SIZE];
        for(size_t i=0; i<SIZE; i++)
            graph[i] = new Edge [SIZE];

        cin >> edgeCount >> source >> stock;

        result = new Pair[edgeCount];

        Pair tmp;
        for(size_t i=0; i<edgeCount; i++)
        {
            cin >> tmp.from >> tmp.to;
            cin >> graph[tmp.from-DIGIT][tmp.to-DIGIT].capacity;
            result[i] = tmp;
        }

        qsort(result, edgeCount, sizeof(Pair), comp);
    }

    ~FordFulkerson()
    {

```

```

        delete result;

        for(size_t i = 0; i<SIZE; i++)
            delete graph[i];
        delete graph;
    }

    bool inVector(unsigned char &v, vector<unsigned char> &vec)
    {
        for(vector<unsigned char>::iterator it = vec.begin(); it != vec.end(); it++)
        {
            if(*it == v)
                return true;
        }
        return false;
    }

    bool searchOneWay(unsigned char v)
    {
        if(v == stock)
        {
            tmpForWays.push_back(v);
            return true;
        }

        unsigned char next = 0;
        for(unsigned char i=DIGIT; i<SIZE+DIGIT; i++)
        {
            if(((graph[v-DIGIT][i-DIGIT].capacity != 0 && graph[v-DIGIT][i-DIGIT].capacity >
graph[v-DIGIT][i-DIGIT].stream)
|| (graph[i-DIGIT][v-DIGIT].capacity != 0 && graph[i-DIGIT][v-DIGIT].stream > 0
&& graph[i-DIGIT][v-DIGIT].capacity > graph[i-DIGIT][v-DIGIT].stream))
&& !inVector(i, blocked) && !inVector(i, tmpForWays))
            {
                next = i;
                break;
            }
        }

        if(next == 0)
        {
            if(v == source)
                return false;
            blocked.push_back(v);
            return searchOneWay(tmpForWays.back());
        }

        if(!inVector(v, tmpForWays))
            tmpForWays.push_back(v);

        return searchOneWay(next);
    }

    void changeStream()
    {
        minStreamInWay = INT_MAX;
        for(vector<unsigned char>::iterator it = tmpForWays.begin()+1; it != tmpForWays.end();
it++)
        {
            if(graph[*it - DIGIT][*it - DIGIT].capacity != 0)
                minStreamInWay = min(minStreamInWay, graph[*it - DIGIT][*it - DIGIT].capacity
- graph[*it - DIGIT][*it - DIGIT].stream);
            else
                minStreamInWay = min(minStreamInWay, min(graph[*it - DIGIT][*it - 1 -
DIGIT].capacity - graph[*it - DIGIT][*it - 1 - DIGIT].stream, graph[*it - DIGIT][*it - 1 -
DIGIT].stream));
        }
        for(vector<unsigned char>::iterator it = tmpForWays.begin()+1; it != tmpForWays.end();
it++)

```

```

        {
            graph[* (it-1) - DIGIT][*it - DIGIT].stream += minStreamInWay;
            graph[*it - DIGIT][* (it-1) - DIGIT].stream -= minStreamInWay;
        }
        maxStream += minStreamInWay;
    }

void setStreams()
{
    if(source == stock)
    {
        printResults();
        return;
    }
    while(searchOneWay(source))
    {
        changeStream();
        tmpForWays.clear();
        blocked.clear();
    }
    printResults();
}

void printResults()
{
    cout << maxStream << endl;
    for(size_t i=0; i<edgeCount; i++)
    {
        printf("%c %c %d\n", result[i].from, result[i].to, graph[result[i].from-
DIGIT][result[i].to-DIGIT].stream);
    }
}

};

int comp(const void *aa, const void *bb)
{
    Pair a = *(Pair*)aa;
    Pair b = *(Pair*)bb;
    if(a.from == b.from)
    {
        return a.to - b.to;
    }
    return a.from - b.from;
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    FordFulkerson G;
    G.setStreams();

    return a.exec();
}

```