

**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: Максимальный поток**

Студент гр. 9383

\_\_\_\_\_

Крейсманн К.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

**2021**

### **Цель работы.**

Изучить алгоритм Форда-Фалкерсона, написать программу, реализующую его.

### **Задание.**

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).

Входные данные:

$N$ -количество ориентированных ребер графа.

$v_0$  — источник

$v_n$  — 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

...

Пример входные данных:

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

Пример выходных данных:

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

**Вариант 1.** Поиск в ширину. Поочередная обработка вершин текущего фронта, перебор вершин в алфавитном порядке.

### Теория

**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.

### **Описание алгоритма**

1) Происходит поиск пути в графе с помощью обхода в ширину от истока к стоку.

2) Если путь есть, высчитывается поток через него, и всем ребрам в этом пути уменьшается пропускная способность на эту величину, а обратным ребрам увеличивается на эту величину.

3) К максимальному потоку прибавляется поток через найденный путь.

4) Пока есть путь выполняется 1-3.

5) Если пути нет, алгоритм завершает работу.

Сложность алгоритма по памяти –  $O(E)$ .

На каждом шаге ищется путь от истока к стоку, сложность поиска пути  $O(E)$ . В худшем случае каждый раз максимальный поток будет увеличиваться на единицу, следовательно сложность алгоритма по операциям:  $O(E \cdot F)$ .

( $E$  – количество ребер,  $F$  – максимальный поток).

### **Описание основных функций и структур данных**

**Vertex** – структура данных для хранения вершины. Хранит информацию о вершине: имя, посещенная или нет, предыдущая вершина в пути.

`Graph (std::map<Vertex*<Vertex*,std::map<Vertex*,int>>>)`– структура данных для хранения графа. Представляет собой ассоциативный контейнер вершин и их соседей.

`findPath()` – ищет путь от истока к стоку с помощью обхода в ширину. Сначала в очередь помещается вершина-исток. Пока очередь не пуста обходятся все не посещенные соседи текущей вершины, они заносятся в очередь. Когда текущей вершиной является сток, алгоритм завершает работу.

`FordFulkerson()` – функция, реализующая алгоритм Форда-Фалкерсона.

`initGraph()` – инициализация графа.

`printResult()` – вывод результата.

## Тестирование

### ***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:***

*Просмотренные ребра при поиске нового пути:*

*Ребро:(a,b), пропускная способность : 7*

*Ребро:(a,c), пропускная способность : 6*

*Ребро:(b,d), пропускная способность : 6*

*Ребро:(c,f), пропускная способность : 9*

*Найденный путь: a c f*

*Поток пути: 6*

*Просмотренные ребра при поиске нового пути:*

*Ребро:(a,b), пропускная способность : 7*

*Ребро:(b,d), пропускная способность : 6*

*Ребро:(d,e), пропускная способность : 3*

*Ребро:(d,f), пропускная способность : 4*

*Найденный путь: a b d f*

*Поток пути: 4*

*Просмотренные ребра при поиске нового пути:*

*Ребро: (a,b), пропускная способность : 3*

*Ребро: (b,a), пропускная способность : 4*

*Ребро: (b,d), пропускная способность : 2*

*Ребро: (d,e), пропускная способность : 3*

*Ребро: (e,c), пропускная способность : 2*

*Ребро: (c,f), пропускная способность : 3*

*Найденный путь: a b d e c f*

*Поток пути: 2*

*Просмотренные ребра при поиске нового пути:*

*Ребро: (a,b), пропускная способность : 1*

*Ребро: (b,a), пропускная способность : 6*

*Результат:*

*Максимальный поток: 12*

*Потоки через ребра графа:*

*a b 6*

*a c 6*

*b d 6*

*c f 8*

*d e 2*

*d f 4*

*e c 2*

**Входные данные:**

*11*

*a*

*f*

*a b 3*

*a c 2*

*d a 3*

*b d 2*

*c d 1*

*b e 8*

*e b 6*

*b f 80*

*e f 1*

*d f 5*

*a e 7*

**Выходные данные:**

*Просмотренные ребра при поиске нового пути:*

*Ребро: (a,b), пропускная способность : 3*

*Ребро: (a,c), пропускная способность : 2*

*Ребро: (a,e), пропускная способность : 7*

*Ребро: (b,d), пропускная способность : 2*

*Ребро: (b,f), пропускная способность : 80*

*Найденный путь: a b f*

*Поток пути: 3*

*Просмотренные ребра при поиске нового пути:*

Ребро: (a,c), пропускная способность : 2  
Ребро: (a,e), пропускная способность : 7  
Ребро: (c,d), пропускная способность : 1  
Ребро: (e,b), пропускная способность : 6  
Ребро: (e,f), пропускная способность : 1  
Найденный путь: a e f  
Поток пути: 1

Просмотренные ребра при поиске нового пути:

Ребро: (a,c), пропускная способность : 2  
Ребро: (a,e), пропускная способность : 6  
Ребро: (c,d), пропускная способность : 1  
Ребро: (e,a), пропускная способность : 1  
Ребро: (e,b), пропускная способность : 6  
Ребро: (d,f), пропускная способность : 5  
Найденный путь: a c d f  
Поток пути: 1

Просмотренные ребра при поиске нового пути:

Ребро: (a,c), пропускная способность : 1  
Ребро: (a,e), пропускная способность : 6  
Ребро: (c,a), пропускная способность : 1  
Ребро: (e,b), пропускная способность : 6  
Ребро: (b,d), пропускная способность : 2  
Ребро: (b,f), пропускная способность : 77  
Найденный путь: a e b f  
Поток пути: 6

Просмотренные ребра при поиске нового пути:

Ребро: (a,c), пропускная способность : 1  
Ребро: (c,a), пропускная способность : 1

Результат:

Максимальный поток: 11

Потоки через ребра графа:

a b 3

a c 1

a e 7

b d 0

b e 0

b f 9

c d 1

d a 0

d f 1

e b 6

e f 1

**Входные данные:**

12

a

f

a b 3

a k 3

a e 3  
k l 1  
k c 5  
b c 5  
e c 5  
e g 1  
l m 1  
m f 1  
g f 1  
c f 100

### **Выходные данные:**

*Просмотренные ребра при поиске нового пути:*

Ребро:(a,b), пропускная способность : 3  
Ребро:(a,e), пропускная способность : 3  
Ребро:(a,k), пропускная способность : 3  
Ребро:(b,c), пропускная способность : 5  
Ребро:(e,g), пропускная способность : 1  
Ребро:(k,l), пропускная способность : 1  
Ребро:(c,f), пропускная способность : 100  
Найденный путь: a b c f  
Поток пути: 3

*Просмотренные ребра при поиске нового пути:*

Ребро:(a,e), пропускная способность : 3  
Ребро:(a,k), пропускная способность : 3  
Ребро:(e,c), пропускная способность : 5  
Ребро:(e,g), пропускная способность : 1  
Ребро:(k,l), пропускная способность : 1  
Ребро:(c,b), пропускная способность : 3  
Ребро:(c,f), пропускная способность : 97  
Найденный путь: a e c f  
Поток пути: 3

*Просмотренные ребра при поиске нового пути:*

Ребро:(a,k), пропускная способность : 3  
Ребро:(k,c), пропускная способность : 5  
Ребро:(k,l), пропускная способность : 1  
Ребро:(c,b), пропускная способность : 3  
Ребро:(c,e), пропускная способность : 3  
Ребро:(c,f), пропускная способность : 94  
Найденный путь: a k c f  
Поток пути: 3

*Просмотренные ребра при поиске нового пути:*

*Результат:*

Максимальный поток: 9  
Потоки через ребра графа:  
a b 3



*a e 3*  
*a k 3*  
*b c 3*  
*c f 9*  
*e c 3*  
*e g 0*  
*g f 0*  
*k c 3*  
*k l 0*  
*l m 0*  
*m f 0*

### **Вывод.**

Произведено знакомство с алгоритмом Форда-Фалкерсона поиска максимального потока в графе, написана программа реализующая его. Поиск пути в графе, использующийся в алгоритме, осуществлялся с помощью обхода в ширину.

## Приложение А

### main.cpp

```
#include "astar.hpp"

int main()
{
    Vertex* start, * finish1, * finish2;
    std::ifstream file("Input.txt");
    Graph graph = initGraph(file, &start, &finish1, &finish2);

    file.close();

    PairPaths paths = findTwoPaths(graph, start, finish1, finish2);

    std::optional<Path> path1to2;
    std::optional<Path> path2to1;
    if (paths.first && paths.second)
    {
        clearMarksAndPrev(graph);
        path1to2 = findPathWithoutOneVertex(graph, finish1, finish2, start);
        clearMarksAndPrev(graph);
        path2to1 = findPathWithoutOneVertex(graph, finish2, finish1, start);
    }

    printAnswer(paths.first, paths.second, path1to2, path2to1, std::cout);
    freeMemory(graph);
    return 0;
}
```

### Ford\_fulkerson.hpp

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#include <map>
#include <fstream>

struct Vertex
{
    Vertex(char name = ' ') :name(name) {}
    char name;
    bool visited = false;
    Vertex* prev = nullptr;
};

struct cmpVertexes
{
    bool operator()(Vertex* v1, Vertex* v2)
```

```

    {
        return v1->name<v2->name;
    }
};
using Graph = std::map<Vertex*,std::map<Vertex*,int,cmpVertexes>,cmpVertexes>;
using Path = std::vector<Vertex*>;
void currentResult(Graph flowGraph,Graph graph,Path path,int maxFlow, int pathFlow);
Path findPath(Vertex *start, Vertex *finish,Graph& graph);
int FordFulkerson(Graph &graph,Graph &originGraph,Vertex*start,Vertex*finish,std::vector<Vertex*> vertexes);
int indexByName(std::vector<Vertex*> vec,char name);
Graph initGraph(std::istream& in, Vertex** start, Vertex** end,std::vector<Vertex*> &vertexes);
void printResult(Graph &graph,Graph &flowGraph,int maxFlow,std::ostream& out);

```

ford\_fulkerson.cpp

```
#include "ford_fulkerson.hpp"
```

```

void currentResult(Graph flowGraph,Graph graph,Path path,int maxFlow, int pathFlow)
{
    std::cout << "Найденный путь: ";
    for(auto i : path)
    {
        std::cout<<i->name<<' ';
    }
    std::cout<<"\n";

    std::cout<<"Поток пути: "<<pathFlow<<"\n";
    std::cout<<"Текущий максимальный поток:"<<maxFlow<<"\n";
    std::cout<<"Поток через ребра графа:"<<"\n";
    for(auto i:graph)
    {
        for (auto j:i.second)
        {
            int flow = std::max(0,j.second - flowGraph[i.first][j.first]);
            std::cout << i.first->name <<' ' << j.first->name <<' ' << flow <<"\n";
        }
    }
}

```

```

Path findPath(Vertex *start, Vertex *finish,Graph& graph)
{
    std::cout<<"\nПросмотренные ребра при поиске нового пути:"<<"\n";

    Path path;
    std::queue<Vertex*> q;
    q.push(start);
    while(!q.empty())
    {
        Vertex *temp = q.front();
        q.pop();
    }
}

```

```

for(auto i:graph[temp])
{
    if(!i.first->visited && graph[temp][i.first]>0)
    {
        q.push(i.first);
        i.first->visited=true;
        i.first->prev=temp;

        std::cout<<"Ребро:("&<<temp->name<<","<<i.first-
>name<<"), пропускная способность : "<<graph[temp][i.first]<<'\n';

        if(i.first->name==finish->name)
        {
            Vertex* t = i.first;
            while(t->name!=start->name)
            {
                path.push_back(t);
                t=t->prev;
            }
            path.push_back(t);
            std::reverse(path.begin(),path.end());
            return path;
        }
    }
}
return path;
}

```

```

int FordFulkerson(Graph &graph,Graph &originGraph,Vertex*start,Vertex*finish,std::vector<Vert
ex*> vertexes)
{
    Path path;
    int maxFlow = 0;
    while(!((path=findPath(start,finish,graph)).empty()))
    {
        for(auto i:vertexes)
        {
            i->visited=false;
        }

        int min=-1;
        for(int i = 0 ;i<path.size()-1;i++)
        {
            if(graph[path[i]][path[i+1]]<min || min==-1)
            {
                min = graph[path[i]][path[i+1]];
            }
        }
        maxFlow+=min;

        for(int i = 0 ;i<path.size()-1;i++)

```

```

    {
        graph[path[i]][path[i+1]]-=min;
        graph[path[i+1]][path[i]]+=min;
    }

    currentResult(graph,originGraph,path,maxFlow,min);
}
return maxFlow;
}

int indexByName(std::vector<Vertex*> vec,char name)
{
    for(int i = 0 ;i<vec.size();i++)
    {
        if(vec[i]->name==name)
        {
            return i;
        }
    }
    return -1;
}

Graph initGraph(std::istream& in, Vertex** start, Vertex** end,std::vector<Vertex*> &vertexes)
{
    Graph graph;
    int weight,size;
    char nameStart, nameEnd;
    in >> size >> nameStart >> nameEnd;

    *start = new Vertex(nameStart);
    *end = new Vertex(nameEnd);
    vertexes.push_back(*start);
    vertexes.push_back(*end);

    for (int i = 0 ;i<size;i++)
    {
        char nameVertexOut,nameVertexIn;
        in >> nameVertexOut >> nameVertexIn >> weight;

        int indStart=indexByName(vertexes,nameVertexOut);
        int indFinish=indexByName(vertexes,nameVertexIn);

        Vertex *vertexOut = indStart== -1 ? new Vertex(nameVertexOut) : vertexes[indStart];
        Vertex *vertexIn = indFinish== -1 ? new Vertex(nameVertexIn) : vertexes[indFinish];
        if(indStart== -1)
        {
            vertexes.push_back(vertexOut);
        }
        if(indFinish== -1)
        {
            vertexes.push_back(vertexIn);
        }
    }
}

```

```

        graph[vertexOut][vertexIn]=weight;
    }
    return graph;
}

void printResult(Graph &graph, Graph &flowGraph, int maxFlow, std::ostream& out)
{
    out<<"\nРезультат:\n";
    out<<"Максимальный поток: "<<maxFlow<<"\n";
    out<<"Потоки через ребра графа:\n";
    for(auto i:graph)
    {
        for (auto j:i.second)
        {
            int flow = std::max(0,j.second - flowGraph[i.first][j.first]);
            out << i.first->name <<' ' <<j.first->name <<' ' << flow <<"\n";
        }
    }
}

```