МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Потоки в сети

Студент гр. 8382 _____ Щеглов А.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение работы алгоритма Форда-Фалкерсона для нахождения максимального потока в сети.

Задание.

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

Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа – пропускной способности (веса).

Входные данные:

N - количество ориентированных рёбер графа

v0 - исток

vn - сток

vi vj ωij - ребро графа

vi vj ωij - ребро графа

...

Выходные данные:

vi vj ωij - ребро графа с фактической величиной протекающего потока

vi vj ωij - 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).

Вариант дополнительного задания.

Вар. 5. Поиск не в глубину и не в ширину, а по правилу: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность. Если таких дуг несколько, то выбрать ту, которая была обнаружена раньше в текущем поиске пути.

Описание алгоритма

Остаточная сеть — это граф с множеством ребер с положительной остаточной пропускной способностью. В остаточной сети может быть путь из u в v, даже если его нет в исходном графе (если в исходной сети есть путь (v, u) с положительным потоком).

Дополняющий путь — это путь в остаточной сети от истока до стока.

Идея алгоритма заключается в том, чтобы запускать поиск в глубину (в индивидуализации по правилу максимальной остаточной пропускной способности) в остаточной сети до тех пор, пока возможно найти новый путь от истока до стока.

Вначале алгоритма остаточная сеть — это исходный граф. Алгоритм ищет дополняющий путь в остаточной сети по следующему алгоритму:

• Находим все смежные вершины к текущей рассматриваемой

• Переходим к вершине с максимальной текущей остаточной пропускной способностью

• Повторяем шаг 1-2 для новой рассматриваемой вершины (алгоритм итеративный)

• Продолжаем, пока не дойдем до стока.

Если путь был найден, то остаточная сеть перестраивается, а к максимальному потоку прибавляется величина максимальной пропускной способности дополняющего пути.

Если путь от истока к стоку не был получен, то максимальный поток найден и алгоритм завершает свою работу.

Очевидно, что максимальный поток в сети является суммой всех максимальных пропускных способностей дополняющих путей.

Описание функций и методов.

Struct Node - структура хранит метку вершины, вектор ребер исходящих из нее, а так же флаг(была ли посещена) и имя предыдущей вершины

Struct Pair – структура хранит в себе имя начальной вершины - u, конечной - v, пропускную способность (u,v) и (v,u)

Тестирование

Ввод	**Вывод**
6 k k k c 10 c d 10 c b 1 b c 1 k b 10 b d 10	0 k b 0 k c 0 b c 0 b d 0 c b 0 c d 0
10 a f a b 16 a c 13 c b 4 b c 10 b d 12	23 a b 12 a c 11 b c 0 b d 12 c b 0 c e 11

c e 14 d c 9 d f 20 e d 7 e f 4	d c 0 d f 19 e d 7 e f 4
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

```
10
a f
a b 16
a c 13
c b 4
b c 10
b d 12
c e 14
d c 9
d f 20
e d 7
e f 4

For =
Versh = a Flag = 0
vec (a:b) = 16/0
vec (a:c) = 13/0


Versh = b Flag = 0
vec (b:a) = 0/16
vec (b:c) = 10/4
vec (b:d) = 12/0


Versh = c Flag = 0
vec (c:a) = 0/13
vec (c:b) = 4/10
vec (c:e) = 14/0
vec (c:d) = 0/9


Versh = d Flag = 0
vec (d:b) = 0/12
vec (d:c) = 9/0
vec (d:f) = 20/0
vec (d:e) = 0/7


Versh = e Flag = 0
vec (e:c) = 0/14
vec (e:d) = 7/0
vec (e:f) = 4/0


Versh = f Flag = 0
vec (f:d) = 0/20
vec (f:e) = 0/4
```

```
Versh = f Flag = 0
vec (f:d) = 0/20
vec (f:e) = 0/4


 rebro vibrano (a.b) ves = 16
 rebro vibrano (b.d) ves = 12
 rebro vibrano (d.f) ves = 20
 rebro vibrano (a.c) ves = 13
 rebro vibrano (c.e) ves = 14
 rebro vibrano (e.d) ves = 7
 rebro vibrano (d.b) ves = 12
 rebro vibrano (d.b) ves = -1
 rebro vibrano (d.f) ves = 8
 rebro vibrano (a.c) ves = 6
 rebro vibrano (c.e) ves = 7
 rebro vibrano (e.f) ves = 4
 rebro vibrano (a.c) ves = 2
 rebro vibrano (c.e) ves = 3
 rebro vibrano (c.e) ves = -1
 rebro vibrano (c.e) ves = -1
 rebro vibrano (c.e) ves = -1
23
a b 12
a c 11
c b 0
b c 0
b d 12
c e 11
d c 0
d f 19
e d 7
e f 4

--------------------------------
Process exited after 65.65 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

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

For =
Versh = a Flag = 0
vec (a:b) = 7/0
vec (a:c) = 6/0


Versh = b Flag = 0
vec (b:a) = 0/7
vec (b:d) = 6/0


Versh = d Flag = 0
vec (d:b) = 0/6
vec (d:e) = 3/0
vec (d:f) = 4/0


Versh = f Flag = 0
vec (f:c) = 0/9
vec (f:d) = 0/4


Versh = e Flag = 0
vec (e:d) = 0/3
vec (e:c) = 2/0


Versh = c Flag = 0
vec (c:a) = 0/6
vec (c:f) = 9/0
vec (c:e) = 0/2
```

```
Versh = c Flag = 0
vec (c:a) = 0/6
vec (c:f) = 9/0
vec (c:e) = 0/2


 rebro vibrano (a.b) ves = 7
 rebro vibrano (b.d) ves = 6
 rebro vibrano (d.f) ves = 4
 rebro vibrano (a.c) ves = 6
 rebro vibrano (c.f) ves = 9
 rebro vibrano (a.b) ves = 3
 rebro vibrano (b.d) ves = 2
 rebro vibrano (d.e) ves = 3
 rebro vibrano (e.c) ves = 2
 rebro vibrano (c.f) ves = 5
 rebro vibrano (a.c) ves = 2
 rebro vibrano (c.f) ves = 3
 rebro vibrano (a.b) ves = 1
 rebro vibrano (a.b) ves = -1
 rebro vibrano (a.b) ves = -1
12
a b 6
a c 6
b d 6
c f 8
d e 2
d f 4
e c 2

--------------------------------
Process exited after 20.04 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Сложность алгоритма

E – множество ребер графа.

V – множество вершин графа.

F – величина максимальной пропускной способности графа.

По времени.

На каждом шаге мы ищем путь от стока к истоку, поиском в глубину с модификацией: каждый раз выполняется переход по дуге, имеющей максимальную остаточную пропускную способность.

Так как просматривать ребра нужно в порядке уменьшения пропускной способности, для этого все ребра вершины сортируются, на это приходится тратить |E| * log(|E|) операций. Помимо этого, алгоритм представляет собой обычный поиск в глубину, поэтому поиск нового дополняющего пути в сети происходит за O(|E| * log|E| * |V|).

В худшем случае, на каждом шаге мы будем находить дополняющий путь с пропускной способностью 1, тогда получим сложность по времени O(F* |E| * log|E| * |V|)

По памяти.

Сложность по памяти O(|E|).

Выводы.

В ходе лабораторной работы была изучена работа алгоритма поиска максимального потока в сети - метод Форда-Фалкерсона, способы хранения графа и остаточной сети и сложности по времени и памяти.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <fstream>
#include <map>
#include <set>

using namespace std;

struct Pair{//Struct of edges
  char u;
  char v;
  double co;
  double ci;
  int pot;
  Pair(char a,char b,double x,double y)
  {
            u=a;
            v=b;
            co=x;
            ci=y;
            pot=0;
  }
  /*Pair(Pair &pair)
  {
            u=pair.u;
            v=pair.v;
            ci=pair.ci;
            co=pair.co;
  }*/
};

struct Node {//Struct of vertex
   char name;
   char prev;
   vector<Pair> E;
   int flag;

   Node(char c)
  {
      name = c;
      flag=0;
   }

};



int main()
{
  int n, fl=0,im,im1,skip=0,uuu=0;
  char a, b,s,e, x;
   double c;
   vector<Node> Nodes;
   vector<Pair> P2;
   vector<Pair> P21;
```

```
Pair P3(a,b,0,0);
Pair P22(a,b,0,0);
Pair P23(a,b,0,0);
double max=0,sum=0,cm=-1;
cin>>n;
cin>>a;
cin>>b;

if(a==b)
{
	cout<<"0"<<endl;//Special occasion

for(int i=0;i<n;i++)
{
	cin>>a>>b>>c;
	cout<<a<<" "<<b<<" 0"<<endl;

}	return 0;}
Node ch=Node('0');
Node chd=Node('0');
Pair ch2=Pair(a,b,0,0);
s=a;
e=b;
Nodes.push_back(Node(a));
Nodes.push_back(Node(b));
for(int i=0;i<n;i++)
{
	cin>>a>>b>>c;//Input edges
	P22.u=a;
	P22.v=b;
	P22.co=c;
	P2.push_back(P22);
	for(int j=0;j<Nodes.size();j++)
	{
		if(Nodes[j].name==a)
		{
			fl=1;
			im=j;
		}
	}
	if(fl==0)
	{
		ch=Node(a);
		ch.E.push_back(Pair(a,b,c,0));//adding edges and vertex
		Nodes.push_back(ch);
	}
	else
	{
		ch=Nodes[im];
		for(int k=0;k<ch.E.size();k++)
		{
			ch2=ch.E[k];
			if(ch2.u==a&&ch2.v==b)
			{
				skip =1;
				im1=k;
			}
		}
		if(skip==1)
```

```
		{
			ch2=ch.E[im1];
			ch2.ci+=0;
			ch2.co+=c;
			ch.E.erase(ch.E.begin()+im1);
			ch.E.push_back(ch2);
		}
		else
		{
		ch.E.push_back(Pair(a,b,c,0));
		}
		Nodes.erase(Nodes.begin()+im);
		Nodes.push_back(ch);
}
fl=0;
skip=0;
for(int j=0;j<Nodes.size();j++)
{
		if(Nodes[j].name==b)
		{
			fl=1;
			im=j;
		}
}
if(fl==0)
{
		ch=Node(b);
		ch.E.push_back(Pair(b,a,0,c));//adding edges and vertex
		Nodes.push_back(ch);
}
else
{
		ch=Nodes[im];
		for(int k=0;k<ch.E.size();k++)
		{
			ch2=ch.E[k];
			if(ch2.u==b&&ch2.v==a)
			{
				skip =1;
				im1=k;
			}
		}
		if(skip==1)
		{
			ch2=ch.E[im1];
			ch2.ci+=c;
			ch2.co+=0;
			ch.E.erase(ch.E.begin()+im1);
			ch.E.push_back(ch2);
		}
		else
		{
		ch.E.push_back(Pair(b,a,0,c));
		}
		Nodes.erase(Nodes.begin()+im);
		Nodes.push_back(ch);
		skip=0;
}
fl=0;
```

```
  }
cout<<endl<<"For = "<<x<<endl;
  for(int i=0;i<Nodes.size();i++)
  {
            ch=Nodes[i];
            cout<<"Versh = "<<ch.name<<" Flag = "<<ch.flag<<endl;
            for(int j=0;j<ch.E.size();j++)
            {
                        ch2=ch.E[j];
                        cout<<"vec ("<<ch2.u<<":"<<ch2.v<<") = "<<ch2.co<<"/"<<ch2.ci<<endl;
            }
            cout<<endl<<endl;
  }//thats what we get the vertexes with their edges

while(1)
{

  x=s;//Starting from the start
  im=0;
  for(int l=0;l<Nodes.size();l++)
            {
                        ch=Nodes[l];
                        if(s==ch.name)
                        {
                                    im=l;
                        }
            }
  ch=Nodes[im];
//          cout<<"Nachinaem s "<<ch.name<<endl;
  ch.prev='0';
  while(x!=e)//if not in the end
  {
            for(int j=0;j<ch.E.size();j++)
            {
                        ch2=ch.E[j];
                        for(int l=0;l<Nodes.size();l++)
                        {
                                    chd=Nodes[l];
                                    if(ch2.v==chd.name&&chd.flag==1)
                                    {
                                                skip=1;//we cant go in this vertex
                                    }
                        }
                        if(skip!=1)
                        {
                                    if(ch2.co>0)//compare the values
                                    {
                                                if(max==0)
                                                {
                                                            max=ch2.co;
                                                            P3=ch2;
                                                }
                                                else
                                                {
                                                            if(max<ch2.co)
                                                            {
                                                                        max=ch2.co;
                                                                        P3=ch2;
                                                            }
```

```
                                        }
                                }
                        }
                        skip=0;
                }
                max=0;
                cout<<" rebro vibrano ("<<P3.u<<"."<<P3.v<<") ves = "<<P3.co<<endl;
                if(P3.co==-1)//output the answer
                {
                        ch.flag=1;
                        Nodes.erase(Nodes.begin()+im);
                        Nodes.push_back(ch);
                        x=ch.prev;
                        if(x=='0')
                        {
                        /*                      cout<<endl<<"For = "<<x<<endl;
for(int i=0;i<Nodes.size();i++)
{
        ch=Nodes[i];
cout<<"Versh = "<<ch.name<<" Flag = "<<ch.flag<<endl;
        for(int j=0;j<ch.E.size();j++)
        {
                ch2=ch.E[j];
                cout<<"vec ("<<ch2.u<<":"<<ch2.v<<") = "<<ch2.co<<"/"<<ch2.ci<<endl;
        }
        cout<<endl<<endl;
}*/
                                cout<<sum<<endl;
                        for(int p =0;p<P2.size();p++)
                        {
                                P22=P2[p];
                                for(int i=0;i<Nodes.size();i++)
                                {
                                        ch=Nodes[i];
                                        for(int j=0;j<ch.E.size();j++)
                                        {
                                                ch2=ch.E[j];
                                                if(ch2.u==P22.u&&ch2.v==P22.v)
                                                {
                        if(ch2.pot==0)
                        {
                         cout<<ch2.u<<" "<<ch2.v<<" 0"<<endl;
                        }
                        else
                                                                cout<<ch2.u<<" "<<ch2.v<<" "<<ch2.pot<<endl;
                                                }
                                        }
                                }
                        }

                                return 0;
                        }
                        for(int l=0;l<Nodes.size();l++)
                        {
                                ch=Nodes[l];
                                if(x==ch.name)
                                {
                                        im=l;
                                }
```

```
                }
                ch=Nodes[im];
                continue;
        }
//max flow in this path
        if(cm==-1)
                cm=P3.co;
        if(cm>P3.co)
                cm=P3.co;
        ch.flag=1;
        Nodes.erase(Nodes.begin()+im);
        Nodes.push_back(ch);
        for(int l=0;l<Nodes.size();l++)
        {
                ch=Nodes[l];
                if(P3.v==ch.name)
                {
                        im=l;
                }
        }
        ch=Nodes[im];
//      cout<<"Teper v "<<ch.name<<endl;
        ch.prev=P3.u;
        x=ch.name;
        P3.co=-1;
}
sum+=cm;
im1=im;
x=e;
while(x!=s)//From the end to the start and change the values of bandwidth

{
        for(int j=0;j<ch.E.size();j++)
        {
                ch2=ch.E[j];
                if(ch2.v==ch.prev)
                {
                        im=j;
                }
        }
        ch2=ch.E[im];
        ch2.ci-=cm;//change value of bandwidth
        ch2.co+=cm;//change value of bandwidth
        if(ch2.pot-cm>=0)
                ch2.pot+=cm;//true flow
        ch.E.erase(ch.E.begin()+im);
        ch.E.push_back(ch2);
        ch.flag=0;
        Nodes.erase(Nodes.begin()+im1);
        Nodes.push_back(ch);
        //Changing for the last vertex almost complete
        a=ch.prev;
        b=ch.name;
        for(int l=0;l<Nodes.size();l++)
        {
                ch=Nodes[l];
                if(a==ch.name)
                {
                        im1=l;
```

```
                }
        }
        ch=Nodes[im1];
        for(int j=0;j<ch.E.size();j++)
        {
                ch2=ch.E[j];
                if(ch2.v==b)
                {
                        im=j;
                }
        }

        ch2=ch.E[im];
        ch2.ci+=cm;//change value of bandwidth
        ch2.co-=cm;//change value of bandwidth
        ch2.pot+=cm;//true flow
        ch.E.erase(ch.E.begin()+im);
        ch.E.push_back(ch2);
        ch.flag=0;
        Nodes.erase(Nodes.begin()+im1);
        Nodes.push_back(ch);
        b=ch.prev;
        x=ch.name;
        //changing for the prev to last complete for half and for the last is complete
        for(int l=0;l<Nodes.size();l++)
        {
                chd=Nodes[l];
                if(x==chd.name)
                {
                        im1=l;
                }
        }
        ch=Nodes[im1];//next vertex to handle

  //        cout<<" PREV else = "<<b<<endl;
  //        cout<<" PREV = "<<ch.prev<<endl;
  //        cout<<" at the same moment = "<<ch.name<<endl;
  //        cout<<" now = "<<x<<endl;
  }

//          cout<<" func = "<<cm<<endl;
}


  return 0;
}
```