**15. Максимальный поток. Алгоритм Форда Фалкерсона.**

***Максимальный поток***

Представим себе ветвящийся водопровод. Если понимать его как сеть в теории графов, то ребра – это трубы, а числа на них – пропускные способности.

Вершины бывают трех сортов: источники, стоки, просто соединения труб.

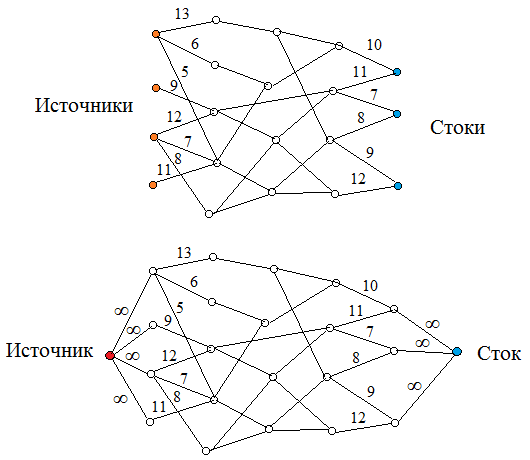
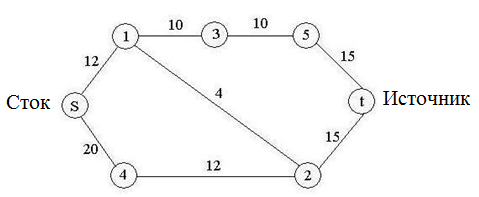
Про источники (истоки) известно, сколько килограммов в секунду жидкости они подают (отводят).

Задача – наладить максимальный стационарный поток из источников в стоки.

Рассмотрим два примера:

первый – с одним источником и одним стоком;

второй – с несколькими источниками и стоками.

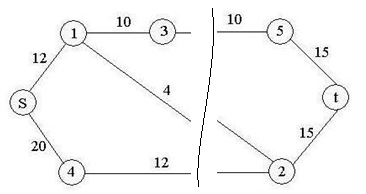


Если много источников и стоков, не теряя общности, можно свести задачу к одному источнику и одному стоку. Вводится фиктивный источник бесконечной мощности и от него протягиваются трубы в настоящие источники с пропускной способностью равной мощности источника.

Назовем разрезом такое множество ребер, что если их разрезать, то сеть распадается на две компоненты связности, в одной из которых будет источник, а в другой - сток. Пропускной способностью разреза (сечения) назовем сумму пропускных способностей его ребер.

Очевидно, что мощность максимального потока не превосходит пропускной способности минимального разреза.

Разрез с минимальной пропускной способностью называется ***минимальным сечением***.



Пропускная способность данного разреза равна **26**.

**Теорема**. Мощность максимального потока равна пропускной способности минимального сечения.

В данной сети видно, что максимальный поток равен 24. Ясно, что по мере роста сложности сети это далеко не очевидно.

***Метод Форда-Фалкерсона***

**C[i,j]** – матрица смежности;

**Use[i]** – отметка о посещенности;

**W[i]** – количество воды полученной вершиной **i;**

**Way[i]** – для запоминания номера вершины, откуда получена вода;

**f0[i,k]** – поток из вершины **i** в вершину **k**;

**f[i,k]** – текущий поток из вершины **i** в вершину **k**;

**L[i]** – очередь просмотра вершин.

Назовем ребро (i,j) ***насыщенным***, если поток по нему равен пропускной способности: **fij = сij.** Путь, состоящий из насыщенных ребер будем называть ***насыщенным путем***.

***Алгоритм общего шага таков***:

Дана вершина **i** с одинарной меткой и **Wi**.

1. В цикле рассмотреть каждую вершину **k**, связанную с **i** ребром:

1.1. Если вершина **k** имеет любую метку, - пропустить **k**;

1.2. Если ребро (**i**,**k**) насыщено и поток направлен из i в **k**, - пропустить **k**;

1.3. Если ребро (**i**,**k**) ненасыщенное и поток направлен из **i** в **k**, -пустить по нему поток **fik (нов)=min(Wi, cik-fik(стар))**;

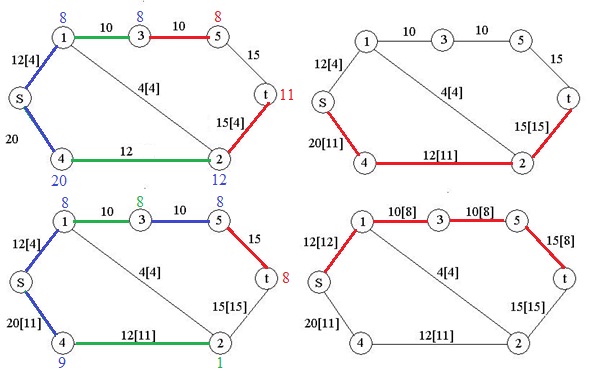
1.4. Если поток идет из **k** в **i**, то **fki(нов)=fki(стар) - min(Wi, fki(стар))**; В отметке о посещении, в **Use[k]:=-1**, что поток нужно будет вычитать.



Поиском в ширину ищем сток. В зависимости от пропускных способностей труб, на вершинах ставим количество перекаченной жидкости **W[i]**.

Как только найдём сток, обратным ходом находим путь и определяем текущий поток жидкости (значения указаны в квадратных скобках).

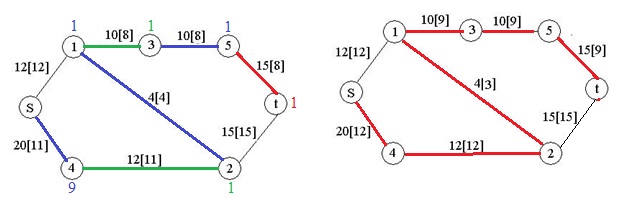
Повторяем поиск в ширину. В двух последующих поисках используются пункты п.1 – п.3



После последнего прохода в поиска в ширину, в новом поиске

на вершине ***2*** возникает ситуация ***пункта 4***. Здесь, мы по ребру

***<2, t>*** пускаем пришедший в вершину ***2*** поток в вершину ***t***. Как видно из рисунка, это поток равен ***1***. Тогда, нам надо будет из потока по ребру ***<1, 2>*** вычесть ***1***. И этот поток постараться пустить по ребру ***<1, 3>***. Как видно из рисунка – это возможно, т.к. пропускные способности рёбер это позволяют.



#include <iostream>

#include <conio.h>

#include <algorithm>

using namespace std;

float a[101][101], c[101][101], f[101][101], f0[101][101];

float W[101]; int L[101], p[101], color[101];

int i,j,k,m,n,s,t,it,kol; bool Ok; float d, ans;

**void flow(int v)** {

if (v>0){

f0[p[v]][v]=f0[p[v]][v]+color[v]\*W[t]; flow(p[v]);

}

}

**void MaxFlow**(int s, int t){

int i,j,u,v;

for (i=1; i<=100;i++){

color[i]=0; p[i]=0; L[i]=0;

for (j=1; j<=n; j++)f[i][j]=0;

}

i=1; k=1; L[k]=s; W[s]=1000000; p[s]=0; color[s]=1;

do {

v=L[i];

for (u=1; u<=n;u++)

if (color[u]==0 && (c[v][u]>f0[v][u]||f0[u][v]>0)) {

if ( c[v][u]>f0[v][u] && f0[u][v]==0){

f[v][u]=min(W[v], c[v][u]-f0[v][u]);

W[u]=f[v][u]; color[u]=1;

}

else {

f[u][v]=f0[u][v] - min(W[v],f0[u][v]);

W[u]=min(W[v], f0[u][v]); color[u]=-1;

}

p[u]=v; k=k+1; L[k]=u;

if (u==t) { Ok=true; return; }

}

i++;

} while (i<=k);

}

**void main**(){

 freopen("maxflow.in","r",stdin);

freopen("maxflow.out","w",stdout);

cin>>n>>m;

for (k=1;k<=m;k++) {

cin>>i>>j>>d; c[i][j]=d; c[j][i]=d;

}

cin>>s>>t;

do {

Ok=false;

MaxFlow(s,t);

if (Ok) {

flow(t); //writeln('->',W[t]:0:0);

ans=ans+W[t]; kol++;

}

}while (Ok);

cout<<ans;

}