Hálózat

Egy számítógépes hálózat csomópontokból és bizonyos csomópont-párokat összekötő, kétirányú adatátvitelt biztosító közvetlen vonalakból épül fel. Minden közvetlen vonal adott átviteli sebességet biztosít. Adatátvitel természetesen közvetett módon is lehet, több közbülső csomóponton keresztül. Ekkor az átvitel sebessége az útvonalba eső közvetlen útvonalak átviteli sebességének minimuma lesz.

Feladat

Ijunk olyan programot, amely kiszámítja, hogy adott két csomópont között mekkora a lehető legnagyobb átviteli sebesség, és meg is ad egy megfelelő útvonalat!

Bemenet

A `standard bemenet` első sora két egész számot tartalmaz, a csomópontok n számát ($1 \leq n \leq 100\,000$) és a közvetlen vonalak m számát ($1 \leq m \leq 1\,000\,000$). A csomópontokat az $1, \ldots, n$ számokkal azonosítjuk. A második sor két csomópont sorszámot tartalmaz egy szóközzel elválasztva; `A B`, azon két csomópont sorszámát, amelyek közötti legnagyobb lehetséges átvitelt keressük. A következő m sor mindegyike három egész számot tartalmaz, egy-egy szóközzel elválasztva, `u v s`, ami azt jelenti, hogy közvetlen kétirányú átviteli vonal van kiépitve az u es a v csomópont között, aminek átviteli sebessége s ($1 \leq s \leq 1\,000$). Két csomópont között több közvetlen vonal is lehet.

Kimenet

A `standard kimenet` első sora azt a legnagyobb lehetséges átviteli sebességet tartalmazza, amelyen átvitel lehetséges a bemenetben megadott A és B csomópont között. A második sor egy olyan útvonalat tartalmazzon, amely a lehető legnagyobb átvitelt adja A és B között. A sorban az első a A, az utolsó a B pont legyen. Több megoldás esetén bármelyik megadható. Ha nem lehetséges átvitel A és B között, akkor az első és egyetlen sorba a 0 számot kell kiírni.

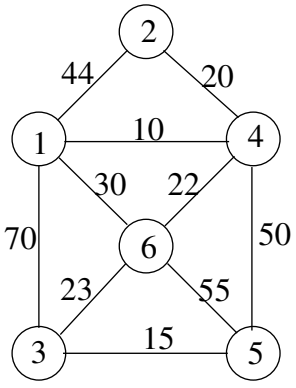
Példa

Bemenet

```
6 10
2 5
1 2 44
2 4 20
1 4 10
1 6 30
6 4 22
4 5 50
5 3 15
6 3 23
1 3 70
6 5 55
```

Kimenet

```
30
2 1 6 5
```



Korlátok

Időlimit: 0.5 mp.
Memórilimit: 32 MiB
Pontozás: a tesztesetek 40%-ában $n < 1000$

Megoldás

A probléma egy lehetséges megoldását az alábbi módosított Dijsktra algoritmus adja.

```
Dijkstra(G,r)
ciklus p:=1-töl n-ig//inicializálás
    Kesz[p]:=hamis
    Tav[p]:=0
ciklus vége
Tav[r]:=Végtelen;
Sorba(S,r)
ciklus amíg Elemszam(S)>0
    u:=Sorbol(S)
    Kesz[u]:=igaz
    ciklus minden u-v élre
        ujtav:=min(Tav[u],suly(u,v));
        ha nem Kesz[v] és ujtav>Tav[v]  akkor
            Tav[v]:=ujtav
            Apa[v]:=u;
            Modosit(S,v,ujtav)
        elágazás vége
    ciklus vége
ciklus vége
```

Megvalósítás C++ nyelven

A C++ közvetlenül nem biztosít módosítható prioritási sort. Ha nem akarjuk magunk megvalósítani a módosítható prioritási sort, akkor vagy az alap prioritási sort, vagy set-et használhatunk.
Prioritási sort (priority_queue) úgy használhatunk, hogy a `Modosit(S,v,ujtav)` művelet helyett ismét berakjuk a v pontot $ujtav$ értékkel a prioritási sorba. Ekkor egy pont többször is benne lehet a prioritási sorban, így amikor kiveszünk egy pontot, ellenőrizni kell, hogy már korábban (nagyobb érték miatt) kivettük-e, azaz $Kesz[v] = igaz$-e?
Ha set-et használunk, akkor előbb törölni kell a v pontot, majd az ujtav értékkel ismét betenni a halmazba.

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <algorithm>
5
6  #define maxN 100001
7  #define Inf 200001
8  using namespace std;
9
10 struct El{
11    int pont, suly;
12    El(int u, int s){
13       pont=u; suly=s;
14    };
15    bool operator<(const El& jobb) const{
16       return suly < jobb.suly || (suly==jobb.suly && pont>jobb.pont);
17    }
18 };
19 vector<El> G[maxN];
20 priority_queue<El> S;
21 int n,A,B;
22
23 void Beolvas(){
24    int m,u,v,s;
25    cin>>n>>m;
26    cin>>A>>B;
27    for(int i=0; i<m; i++){
28       scanf("%d␣%d␣%d", &u, &v, &s); // cin>>u>>v>>s;
29       G[u].push_back(El(v,s));
30       G[v].push_back(El(u,s));
31    }
32 }
33
34 int Apa[maxN];
35 int Tav[maxN];
36 bool Kesz[maxN];
```

```
37 void Dijkstra(int r){
38 //Globális: G,Kesz,Tav,Apa
39    int ujtav;
40    El e=El(0,0); El u=El(0,0);
41    for (int v = 1; v <= n; ++v){//inicializálás
42       Kesz[v]=false; Tav[v]=0;
43    }
44    Tav[r]=Inf;
45    Apa[r]=0;
46    S.push(El(r,Tav[r]));
47    while (S.size() > 0){
48       u=S.top(); S.pop();
49       if(Kesz[u.pont]) continue;
50       Kesz[u.pont]=true;
51       for(El v:G[u.pont]){
52          ujtav=min(u.suly, v.suly);
53          if (!Kesz[v.pont] && ujtav>Tav[v.pont]){
54             e.pont=v.pont; e.suly=ujtav;
55             S.push(e);
56             Tav[v.pont]=ujtav;
57             Apa[v.pont]=u.pont;
58          }
59       }
60    }
61 }
62
63 int main(){
64    Beolvas();
65    Dijkstra(A);
66    int Ut[maxN];
67    int x=B;
68    int m=0;
69    while (x!=A){
70       Ut[m++]=x;
71       x=Apa[x];
72    }
73    Ut[m]=A;
74    cout<<Tav[B]<<endl;
75    for (int i=m; i>=0; i--)
76       cout<<Ut[i]<<"␣";
77    cout<<endl;
78 }
```