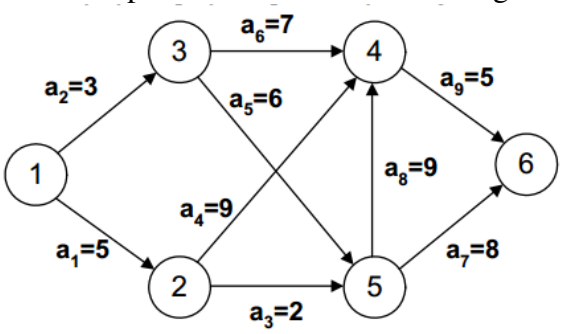
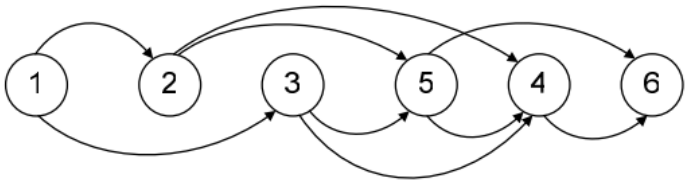
Grafovi

1. Pronaći topološki redosled čvorova za graf sa slike. Da li postoji neki Hamiltonov put u grafu?

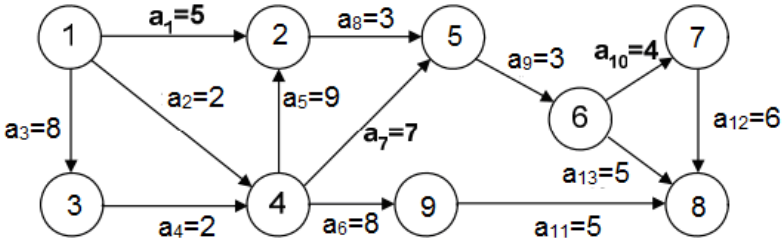


Rešenje:
Topološki redosled čvorova grafa je

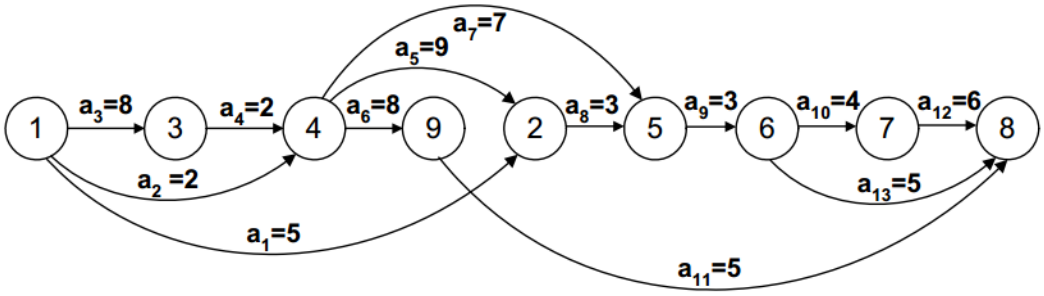


U ovom grafu topološki redosled čvorova nam ne daje Hamiltonov put, jer nisu svaka dva uzastopna čvora u topološkom redosledu putno povezani.

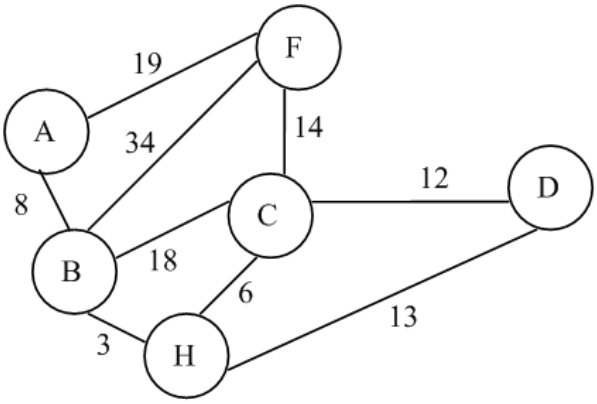
2. Pronaći topološki redosled čvorova za graf sa slike.



Rešenje:



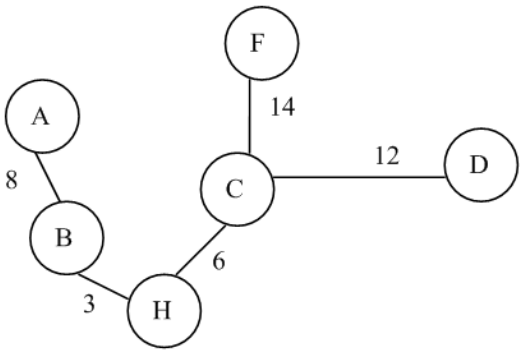
3. Za dati graf na slici, konstruišite razapinjuće stablo minimalne cene (MCST) upotrebom Prim-ovog i Kruskal-ovog algoritma.



Rešenje:
Stablo koje dobijamo uklanjanjem određenog broja grana grafa, a da pritom dobijeni graf ostane povezan, zovemo razapinjajuće ili obuhvatno stablo (spanning tree). Generalno, konstruisanje razapinjajućeg stabla jednostavan je postupak, ali je često u primenama poželjno da takvo razapinjajuće stablo ima i neka dodatna svojstva, kao što je minimalna/maksimalna suma težina grana. Obuhvatna stabla se generišu algoritmima za obilazak grafa po širini ili dubini. Za isti graf može postojati više obuhvatnih stabala.

Minimalno obuhvatno stablo ima najmanju cenu (suma cena grana). Može biti više takvih stabala (ista cena).

Čvor A je odabran za koren stabla



A-B
A-B, B-H
A-B, B-H, H-C
A-B, B-H, H-C, C-D
A-B, B-H, H-C, C-D, C-F

Prim-ov algoritam:
PRIM(G, s)
$U = \{s\}$
$E' = \varnothing$
while ($U \neq V$) **do**
    find $(u, v) \Rightarrow \min \{w(u, v) : (u \in U)$ and $(v \in (V- U))\}$
    $U = U + \{v\}$
    $E' = E' + \{(u, v)\}$
end_while
$MCST = (U, E')$

Kruskal-ov algoritam
1. Inicijalno, graf se posmatra kao potpuno nepovezan (nepovezane komponente)
2. Skup grana E se uređuje po neopadajućoj težini (prioritetan red)
3. Nova grana se dodaje samo ako spaja dve odvojene komponente (T)

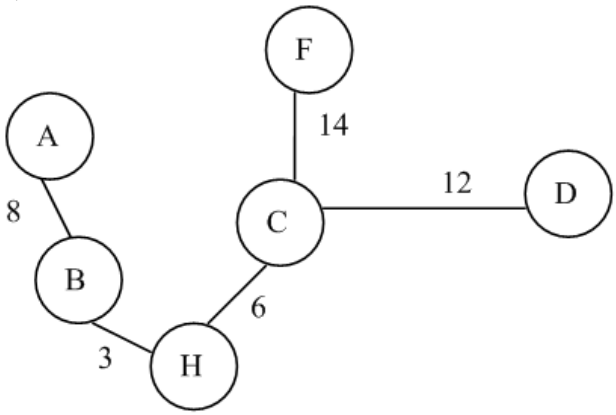
KRUSKAL(G)
$E' = \varnothing$
for each $(u, v) \in E$ **do**
    PQ-INSERT(PQ, $w(u, v)$)
end_for
$num = 0$
while ($num < n$ - 1) **do**
    $w(u,v)$ = PQ-MIN-DELETE(PQ)
    if (($u \in Ti$) and ($v \in Tj$) and ($i \neq j$)) **then**
        $E' = E' + \{(u, v)\}$
        $Tk = Ti + Tj$

num = *num* + 1
end_if
end_while
MCST = (*V*, *E*′)

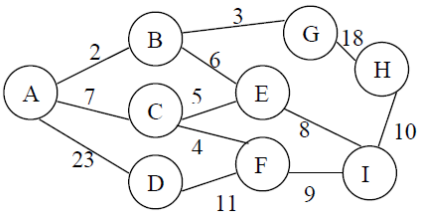
a) Od korena stabla mora da polazi grana najmanje težine. Ovde se bira čvor H.

b)
H-B
H-B, H-C
H-B, H-C, B-A
H-B, H-C, B-A, C-D
H-B, H-C, B-A, C-D, C-F

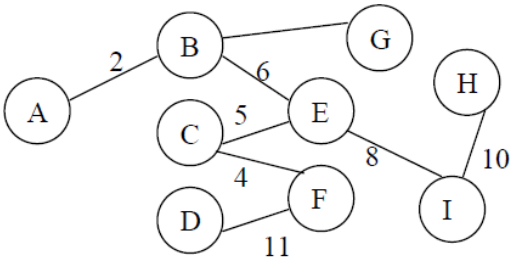
c)



4. Za dati graf na slici, konstruišite razapinjuće stablo minimalne cene (MCST) upotrebom Prim-ovog i Kruskal-ovog algoritma.



Rešenje:
Obuhvatno stablo



Prim-ov algoritam
A-B, B-G, B-E, E-C, C-F, E-I, I-H, F-D

Kruskal-ov algoritam
A-B, B-G, C-F, E-C, B-E, E-I, I-H, F-D

5. U zemlji S, postoji n jezera (numerisanih od 1 do n) i m kanala između njih. Poznata je širina svakog kanala (u metrima). Kretanje kanalima se može izvesti u oba smera. Poznato je da čamac širine jedan metar može dospeti do ma kog jezera, počevši od jezera sa rednim brojem 1.

Napisati program koji izračunava minimalni broj kanala koje treba proširiti, tako da čamac širine k metara može putovati između svaka dva jezera (čamac se može kretati od jednog jezera do drugog, ako je njegova širina manja ili jednaka od širine kanala koji povezuje jezera).

Ulaz

U prvoj liniji standardnog ulaza su dati celi brojevi n i m ($1 < n \leq 1000$, $1 < m \leq 100000$).

U narednih m linija su data tri cela broja, i, j i w, koji ukazuju da postoji kanal širine w ($1 \leq w \leq 200$) između jezera, i i j ($1 \leq i, j \leq n$).

U poslednjoj liniji je dat ceo broj k ($1 \leq k \leq 200$).

Izlaz

U jedinom redu standardnog izlaza ispišite jedan ceo broj: minimalni broj kanala koji treba proširiti.

Primer

Ulaz

```
6 9
1 6 1
1 2 2
1 4 3
2 3 3
2 5 2
3 4 4
3 6 2
4 5 5
5 6 4
4
```

Izlaz

```
2
```

Rešenje;

1 način

Nađimo maksimalno obuhvatno stablo T_{max} grafa jezera G i povežimo ga kanalima. Posle izračunavamo koliko (grana) kanala iz T_{max} su uži od date vrednosti K. Neka je broj takvih kanala q. Ako se ovi kanali prošire, čamac širine K može da prođe između svaka dva jezera. Štaviše, ako uklonimo sve kanale uže od K, graf će se razbiti na $q+1$ komponenti povezanosti i minimalni broj kanala koji će povezati sve komponente jeste q.

2 način

Tražimo broj komponenti povezanosti grafa jezera G_K i kanala širine barem K.

Neka je taj broj $q+1$. Pošto je grad G povezan, postojaće q tesnih kanala, koji će pri proširivanju do širina K i dodavanjem u G_K povezati komponente.

Pronalaženje komponenti povezanosti u G_K može da se obavi nekim algoritmom za obilazak grafa – *BFS* ili *DFS*.

```
#include <algorithm>
#include <cstdio>
#include <map>
#include <vector>
#include <queue>
#include <time.h>
#include <limits.h>
using namespace std;

const int MaxVertex=1001;
int E[MaxVertex][MaxVertex], D[MaxVertex], Pi[MaxVertex];
bool Marked[MaxVertex];
int N,M,K;
```

```
void input()
{
  int u,v,w;
  scanf("%d %d", &N, &M);
  for(int i=1; i<=N; i++)
   for (int j=1; j<=N; j++)
    E[i][j]=0;
  for(int i=1; i<=M; i++)
  {
   scanf("%d %d %d", &u, &v, &w);
   E[u][v]=w;
   E[v][u]=w;
  }
  scanf("%d", &K);
}

void CreateMST(int s)
{
  int u,v,w,cnt;
  for(int i=1; i<=N; i++)
   { D[i]=0; Pi[i]=-1; Marked[i]=false; }
  D[s]=INT_MAX; // niz D čuva težine grana u MaxSpaningTree !!
                          // niz Pi čuva prethodnike čvorova
  cnt=1; // koliko cvorova je u T, koren s je u stablu T
  while (cnt<=N) {
   w=0; u=0;
   for(int i=1; i<=N; i++)
     if ((D[i]>w) && (!Marked[i]))
       {w=D[i]; u=i;}
   Marked[u]=true;
//   if (u != s) printf("%d %d %d\n",u, Pi[u], D[u]);
   for(int v=1; v<=N; v++)
     if ((D[v]<E[u][v]) && (!Marked[v]))
       {D[v]=E[u][v]; Pi[v]=u;}
   cnt++;
  }
// uredjeni par <v,Pi(v)> je u T,  sa tezinom d[v], za v != s
}

void SolveP2()
{
  int l;
  l=0;
  for (int v=1; v<=N; v++)
   if ((Pi[v]>0) && (D[v]<K))
    l++;
//  printf("Kratki_kanali = %d\n",l);
  printf("%d\n",l);
}

main()
{
  input();
  CreateMST(1);
  SolveP2();
```

}

6. U zemlji S, postoji N jezera (numerisanih od 1 do n) i M kanala između njih. Poznata je širina svakog kanala (u metrima). Kretanje kanalima se može izvesti u oba smera.

Transportno preduzeće *Jezero* je pripremilo listu od K parova jezera između kojih će se obavljati redovni transport robe i ljudi putem čamaca.

Napišite program koji izračunava maksimalnu širinu čamaca, koji mogu proći između parova jezera koji se nalaze na listi jezera (čamac se može kretati od jednog jezera do drugog, ako je njegova širina manja ili jednaka od širine kanala koji povezuje jezera).

Ulaz

U prvoj liniji standardnog ulaza su dati celi brojevi N, M, K ($N \leq 1000$, $M \leq 100000$, $K \leq 10000$).

U narednih m linija su data tri cela broja, i, j i w, koji ukazuju da postoji kanal širine w između jezera, i i j ($1 \leq i, j \leq N$, $Wi,j \leq 200$).

Potom sledi K linija, tako da svaka sadrži brojeve dva jezera i, j, između kojih se obavlja transport ljudi i robe.

Izlaz

Program treba da ispiše K redova na standardnom izlazu, od kojih svaki sadrži jedan ceo broj, jednak maksimalnoj širini čamca, koji može putovati između odgovarajuća dva jezera.

Primer

Ulaz

```
6 9 4
1 2 2
1 4 3
1 6 1
2 3 3
2 5 2
3 4 4
3 6 2
4 5 5
5 6 4
2 6
3 5
1 2
4 6
```

Izlaz

```
3
4
3
4
```

Rešenje:

Pronađimo maksimalno obuhvatno stablo T grafa jezera G i povežimo ga kanalima. Sve grane ovog stabla su grane sa najvećim težinama u G, odnosno najširi kanali među čvorovima grafa G. Prim-ovim algoritmom se može generisati ovo stablo. Možemo ovo stablo predstaviti kao koreno stablo tako da niz D čuva i reguliše širinu kanala (iz stabla) dok niz Pi čuva prethodnike čvorova. Uz malo analize obilaska stabla možemo izračunati na kom nivou u odnosu na koren je svaki čvor (vrednosti niza *Depth*). Maksimalno obuhvatno stablo koje je predstavljen na takav način omogućuje da se u vremenu $O(p)$ (gde p je dužina puta) izračuna širina puta između svakog para čvorova u datoj listi sa K čvorova.

```
#include <algorithm>
#include <cstdio>
#include <map>
#include <vector>
#include <queue>
#include <time.h>
#include <limits.h>
using namespace std;
```

```
const int MaxVertex=1001, MaxPairs=10001;
int E[MaxVertex][MaxVertex], Pairs[MaxPairs][2], D[MaxVertex], Pi[MaxVertex], Depth[MaxVertex];
bool Marked[MaxVertex];
int N,M,K,w;


void input()
{
  int u,v,w;
  scanf("%d %d %d", &N, &M, &K);
  for(int i=1; i<=N; i++)
   for (int j=1; j<=N; j++)
    E[i][j]=0;
  for(int i=1; i<=M; i++)
  {
   scanf("%d %d %d", &u, &v, &w);
   E[u][v]=w;
   E[v][u]=w;
  }
  for(int i=1; i<=K; i++)
   scanf("%d %d\n",&Pairs[i][1],&Pairs[i][2]);
}

void CreateMST(int s)
{
  int u,v,w,cnt;
  for(int i=1; i<=N; i++)
   { D[i]=0; Pi[i]=-1; Marked[i]=false; Depth[i]=0;}
  D[s]=INT_MAX; // MaxSpaningTree !!
  cnt=1; // broj cvorova u T (koren je s)
  while (cnt<=N) {
   w=0; u=0;
   for(int i=1; i<=N; i++)
     if ((D[i]>w) && (!Marked[i]))
       {w=D[i]; u=i;}
   Marked[u]=true;
//   if (u != s) printf("%d %d %d %d\n",u, Pi[u], D[u], Depth[u]);
   for(int v=1; v<=N; v++)
     if ((D[v]<E[u][v]) && (!Marked[v]))
       {D[v]=E[u][v]; Pi[v]=u; Depth[v]=Depth[u]+1;}
   cnt++;
  }
// uredjeni par <v,Pi(v)> je u T,  sa tezinom d[v], za v != s
}

int WP(int u, int v)
{
  int w;
  w=INT_MAX;
  while (Depth[u]>Depth[v])
  {
   if (D[u]<w) w=D[u];
   u=Pi[u];
  }
  while (Depth[v]>Depth[u])
  {
```

```
   if (D[v]<w) w=D[v];
   v=Pi[v];
  }
  while (v!=u)
  {
   if (D[v]<w) w=D[v];
   if (D[u]<w) w=D[u];
   u=Pi[u];
   v=Pi[v];
  }
  return w;
}

void SolveP1()
{
  int w,u,v;
//  w=INT_MAX;
//  for (int v=1; v<=N; v++)
//   if ((Pi[v]>0) && (D[v]<w))
//    w=D[v];
//  printf("%d\n",w);

  for(int i=1; i<=K; i++)
  {
   u=Pairs[i][1];
   v=Pairs[i][2];
//   printf("%d %d %d\n",u,v,WP(u,v));
   printf("%d\n",WP(u,v));
  }
}

main()
{
  input();
  CreateMST(1);
  SolveP1();
}
```

7. Detektiv Deki je odlučio da iskoristi sav svoj šarm i diplomatske sposobnosti (manipulacija i intrige) i da omogući da za svakog studenta na fakultetu važi sledeće tvrđenje: broj studenata koje student A smatra za prijatelje se razlikuje najviše za 1 od broja studenata koji studenta A smatraju za prijatelja. Dakle, ako zamislimo da svaki student je čvor grafa i da činjenica da *A smatra da mu je B prijatelj* se može predstaviti usmerenom granom od A do B u grafu, onda Dekijeva želja je zapravo da se za svaki čvor broj ulaznih grana razlikuje od broja izlaznih grana najviše za 1.
Naš detektiv Deki zna sve parove studenata na fakultetu koji su poznanici (tj. u grafu između njih postoji neusmerna grana). Deki može šarmom i diplomatijom da primora bilo koji par poznanika da izmene svoj odnos, tako da je jedan od njih počinje da smatra drugog studenta za prijatelja ili obrnuto, ali ne i da oba studenta u isto vreme se uzajamno smatraju za prijatelja, jer Deki je protivnik kreiranja klanova i uzajamnih prijateljstava. On želi da izvrši svoj pokvareni plan tako što će"usmeriti" sve grane grafa - odnosno svako poznanstvo će pretvoriti u jednosmerno prijateljstvo. Proverite da li Deki može to da uradi i ako može prikažite neku moguću orijentaciju grana grafa da bi se postigao Dekijev plan.

Ulaz
U prvoj liniji standardnog ulaza se zadaju dva cela broja N, M ($1 \leq N \leq 1000$,$1 \leq M \leq 100000$) koji redom predstavljaju broj studenata na fakultetu i broj poznanstava. U narednih M linija su dati parovi brojeva P1 i P2, ($1 \leq P1, P2 \leq N$), koji označavaju da studenti P1 i P2 su poznanici. Svako poznanstvo se kao ulazni podatak zadaje tačno jednom. Svi brojčani ulazni podaci su pozitivni celi brojevi.

Izlaz
U prvoj linji standardnog izlaza napišite “Da” ako je moguća takva orijentacija grana grafa, ili “Ne” ako nije moguće orijentisati graf na gore opisani način. Ako odgovor jeste “Da” onda na svkoj od narednih M linija je potrebno štampati par celih brojeva P1 P2, koji označavaju da P1 smatra da mu je P2 prijatelj.
Ako postoji više načina za orijentaciju grana grafa, odšampajte bilo koji način.

Primer

Ulaz	Izlaz
5 7 1 2 1 3 4 1 1 5 3 2 4 5 3 5	Da 1 2 3 1 4 1 1 5 2 3 5 4 3 5

Objašnjenje: Postoji 5 studenta i 7 poznanstava među njima. Nakon orijentacija grana grafa, svaki student (osim studenata 5 i 3) ima broj prijatelja koji je jednak broju studenata koji njega smatraju za prijatelja. Student broj 5 ima za 1 veći broj studenata koji njega smatraju za prijtelja. Student broj 3 ima za 1 veći broj studenata koje on smatra za prijatelje.

Rešenje: Moramo polazni graf dopuniti fiktivnim neorijentisanim granama sa ciljem da dobijemo Ojlerov graf (tj. da stepen svakog čvora bude paran). To možemo **uvek** uraditi, jer u polaznom grafu uvek postoji paran broj čvorova sa neparnim stepenom (suma stepena svih čvorova je paran broj jer je ta suma jednaka dvostrukom broju grana grafa). Potom, pokrenimo algoritam da u novom grafu pronađe Ojlerov ciklus. Time se proširen graf orijentiše tako da ulazni stepen svakog čvora je jednak izlaznom stepenu. Kako smo svakom čvoru dodali najviše jednu fiktivnu granu, onda nakon njihovog uklanjanja, dobijamo da se ulazni i izlazni stepen razlikuju najviše za jedan (što smo i želeli).

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>

#define MAX_NODES 1024
#define MAX_EDGES 131072

using namespace std;

struct Edge
{
	int next;
	int node1, node2;
	int fictive, dir;
};

int n, m;
int start[MAX_NODES];
Edge edges[MAX_EDGES << 1]; int numEdges;

void addEdge(int node1, int node2, int fictive)
{
	for (int i = 0; i < 2; i++)
	{
		edges[numEdges].node1 = node1;
		edges[numEdges].node2 = node2;
```

```
		edges[numEdges].fictive = fictive;
		edges[numEdges].dir = -1;
		edges[numEdges].next = start[node1];
		start[node1] = numEdges++;
		node1 ^= node2 ^= node1 ^= node2;
	}
}

void euler(int node)
{
	for (int idx = start[node]; idx != -1; idx = edges[idx].next)
	{
		if (edges[idx].dir == -1)
		{
			edges[idx ^ 0].dir = 0;
			edges[idx ^ 1].dir = 1;
			euler(edges[idx].node2);
		}
	}
}

int main(void)
{
	// Inicijalizacija
	numEdges = 0;
	memset(start, -1, sizeof(start));

	// Ocitavanje ulaza i formiranje grafa u nizu grana(slogova) edges
	fscanf(stdin, "%d %d", &n, &m);
	for (int i = 0; i < m; i++)
	{
		int n1, n2;
		fscanf(stdin, "%d %d", &n1, &n2);
		addEdge(n1 - 1, n2 - 1, 0);
	}

	// dodavanje fiktivnih grana
	int last = -1;
	for (int i = 0; i < n; i++)
	{
		int parity = 0;
		for (int idx = start[i]; idx != -1; idx = edges[idx].next)
			parity ^= 1;

		if (parity == 1)
		{
			if (last == -1) last = i;
			else
			{
				addEdge(last, i, 1);
				last = -1;
			}
		}
	}

	// za svaku povezanu komponentu grafa, pokrenuti
//algoritam za trazenje Ojlerovog ciklusa i orijentisanje grana
```

```
//tako da da ulazni stepen svakog čvora je jednak izlaznom stepenu
        for (int i = 0; i < numEdges; i++)
                if (edges[i].dir == -1) euler(edges[i].node1);

        fprintf(stdout, "Da\n");
        for (int i = 0; i < numEdges; i++)
                if (!edges[i].fictive && edges[i].dir == 0)
                        fprintf(stdout, "%d %d\n", edges[i].node1 + 1, edges[i].node2 + 1);
        cout<<endl;
        return 0;
}
```

8. Ako je ekscentricitet čvora **i** grafa G(V,E) maksimum najkraćih rastojanja od svih ostalih čvorova, napisati C program koji će odrediti čvor najmanjeg ekscentriciteta – središte grafa. Primena: određivanje optimalne lokacije, na primer, za stanicu hitne pomoći.
Uputstvo. Korišćenjem Flojdovog algoritma odrediti najkraća rastojanja između svih čvorova. Zatim, odrediti min ekscentriciteta svih čvorova grafa.

9. Dato je **n** kutija dimenzija (x_i, y_i, z_i). Odrediti maksimalni broj kutija koje se mogu staviti jedna u drugu.
Uputstvo:
Formirati matricu veze A u kojoj je $a_{i,j} = -1$, ako se u i-tu kutiju može smestiti kutija j,
u protivnom $a_{i,j} = \infty$ (MAX).
Zatim, primenom Flojdovog algoritma odrediti matricu dužina "najkraćih puteva". Traženi maksimalni broj kutija je $abs(\min(a_{ij})) + 1$, $_{i\,j \in 1,..,n}$

10. Iz datog niza reči izdvojiti reči koje obrazuju najduži podniz reči takvih da svake dve susedne imaju zajednički segment od najmanje dva znaka. Na primer, 'NEVEN' i 'VENAC' mogu biti dve susedne reči.
Uputstvo:
Formirati matricu veze A u kojoj je $a_{i,j} = -1$, ako se na i-tu reč može nadovezati j-ta reč, u protivnom $a_{i,j} = \infty$ (MAX).
Zatim, primenom Flojdovog algoritma odrediti matricu dužina "najkraćih puteva". Traženi maksimalni broj reči je $abs(\min(a_{ij})) + 1$, $_{i\,j \in 1,..,n}$

11. Država alhemičara ima N naseljenih punktova, numerisanih od 1 do N, i M puteva. Naseljeni punktovi su dva tipa: sela i gradovi. Osim toga u državi je jedna prestonica (ona može biti u selu ili u gradu). Putovanje između dva naseljena punkta (ako postoji put) traje T_i minuta. Nakon odluke da se u prestonici organizuje olimpijada alhemičara u svaki grad su upućeni kuriri sa informacijom o olimpijadi.
Napišite pro gram koji određuje u kom poretku i za koje vreme svaki od kurira stiže do svog grada. Pretpostavlja se da se kuriri u toku puta nigde ne zadržavaju.
U prvoj liniji ulaznog fajlu su zapisana 3 broja: N, M, K – broj naseljenih punktova, broj puteva i broj gradova $(2 \le N \le 1000, 1 \le M \le 10000, 1 \le K \le N)$.
Dalje je zapisan broj prestonice C $(1 \le C \le N)$. U sledećem redu su K brojeva gradova. Dalje sledi M trojki brojeva $S_i\ E_i\ T_i$, koji opisuju puteve: S_i i E_i – brojevi naseljenih punktova koje povezuje put, a T_i – trajanje putovanja tim putem $(1 \le T_i \le 100)$. Garantuje se da se iz prestonice može stići do svakog grada.

test.txt	izlaz
5 4 5 1 1 2 3 4 5 1 2 1 2 3 10 3 4 100 4 5 100	1 0 2 1 3 11 4 111 5 211

Rešenje. Za svako naselje korišćenjem Dijkstrinog algoritma nađemo najmanje vreme, za koje kurir do njega stiže. Nakon toga ispisati naselja koja su gradovi sortirane po vremenu u neopadajućem poretku.

```
#include <iostream.h>
#include <limits.h> // Zbog INT_MAX
#include <fstream.h>
const int maxN=150; // Maksimalni broj cvorova u grafu
const int max=INT_MAX/2;
int n,m,k,c; // Startni cvor
int a[maxN][maxN], d[maxN], pos[maxN];
bool mark[maxN],grad[maxN];

// medju nemarkiranim cvorovima nalazi najblizi startnom cvoru s.
// tj. cvor j za koji je d[j] minimalno
int ExtractMin(bool mark[], int d[], int n)
{ int minD=max;
   int i,j=-1;
for (i=1; i<=n; i++)
    if (!mark[i] && d[i]<minD){minD=d[i];j=i;}
return j;
}

void Dijkstra(int s)
{ int i, j, k;
for (i=1; i<=n; i++) // Inicijalizacija: d[i]=a[s][i]
{
    d[i]=a[s][i];
   mark[i]=false;//mark[i]=false oznacava pripadnost cvora skupu T
}

d[s]=0; // Rastojanje od s do s je 0
mark[s]=true; // Cvor s ulazi u skup S - postojana rastojanja

// Ciklus se prekida ako ne postoji cvor i iz T takav da je: d[i]<MAX.
for (k=1; k<=n-1; k++) // Postoji n-1 kandidata za S
{// Izbor nemarkiranog cvora j (iz T), cije je d[j] minimalno
   j=ExtractMin(mark,d,n);
    if (j==-1)break;// Medju nemarkiranim cvorovima,
    // nema dostiznih iz s:d[i]=max (nedostizan)- izlaz

   mark[j]=true; // Cvor j ulazi u skup postojanih cvorova S

// Za sve cvorove koji su u T poboljsava se ocena, ako je moguce:
// d[i]=min(d[i],d[j]+a[j][i])

for (i=1; i<=n; i++)
if (!mark[i])
if (d[i]>d[j]+a[j][i])d[i]=d[j]+a[j][i];
}
}


void main()
{int g, s, e, t, i, j;
   bool temp;
ifstream f("test.txt");
f >> n >> m >> k >> c;

   for (i=1; i<=n; i++) {grad[i]=false;pos[i]=i;}
   for (i=1; i<=k; i++) {f >> g; grad[g]=true; }

for (i=1; i<=n; i++)
for (j=1; j<=m; j++)
    if (i==j) a[i][j]=0; else a[i][j]=max;

for (i=1; i<=m; i++) { f >> s >> e >> t; a[s][e]=t; a[e][s]=t; }
Dijkstra(c);

// sortira niz d[] tako sto razmenu prate grad[] i pos[]
for (i=1; i<=n-1; i++)
for (j=i+1; j<=n; j++)
  if (d[i]>d[j])
    {t=d[i]; d[i]=d[j]; d[j]=t;
     temp=grad[i]; grad[i]=grad[j]; grad[j]=temp;
      t=pos[i]; pos[i]=pos[j]; pos[j]=t;}

for (i=1; i<=n; i++)
if (grad[i]) cout << pos[i] << ' ' << d[i]<< endl;
}
```