Obilazak grafa

1. Ako su dati brojevi a,b,c, d (1<=a,b,c,d<=8) koji predstavljaju dve lokacije konja (skakača u šahu koji se kreće u obliku slova L) na šahovskoj tabli i ako je dat broj g (1<=g<10), proverite može li skakač doći na polje c-d u ne više od g poteza.
ULAZ
1 1 8 8 5
IZLAZ
Ne moze

Ideja rešenja:
DFS (depth first search, pretraga u dubinu) je algoritam pretrage koji potencijalna rešenja obilazi grananjem u dubinu. DFS pretražuje sve situacije (stanja) koja su potencijalno rešenje problema.

Zadatak ćemo rešiti DFS-om tj. rekurzivnom funkcijom void obidji(int x, int y, int potez)
Funkcija obidji kao argument ima poziciju polja na tabli (x,y) i broj poteza da bismo došli do tog polja na tabli.

Konj se kreće u obliku slova L, tj. dva polja u proizvoljnom smeru (horizontalnom i vertikalnom) i jedno polje u normalnom pravcu u odnosu na prethodni pravac.

Analiza zadatka i primera:

Dakle, skakač sa polja 4-4 može doći na polje 2 3, 2 5, 3 2, 3 6, 5 2, 5 6, 6 3, 6 5

			X		X		
		X				X	
				K			
		X				X	
			X		X		

Skakač sa polja x-y može doći na polje
x-1,y-2
x-1,y+2
x+1,y-2
x+1,y+2
x-2,y-1
x-2,y+1
x+2,y-1
x+2,y+1

Za ulaz 1 1 8 8 5 možemo DFSom upisati u koliko poteza smo stigli do neke pozicije na tabli

0	3	2	3	2	3	4	5
3	4	1	2	3	4	3	4
2	1	4	3	2	3	4	5
3	2	3	2	3	4	3	4
2	3	2	3	4	3	4	5
3	4	3	4	3	4	5	4
4	3	4	3	4	5	4	5
5	4	5	4	5	4	5	**-1**

Implementacija

```
#include <stdio.h>

int tabla[8][8];
int granica;

void obidji (int x,int y,int potez)
{ if(x<0 || x>7 || y<0 || y>7) return; //lose polje
if(tabla[x][y]!=-1 &&tabla[x][y]<=potez) return;
//tu smo vec bili u isto ili manje poteza
if(potez>granica) return; //potrosili smo poteze
tabla[x][y]=potez;
 obidji(x-1,y-2,potez+1);
 obidji(x-1,y+2,potez+1);
 obidji(x+1,y-2,potez+1);
 obidji(x+1,y+2,potez+1);
 obidji(x-2,y-1,potez+1);
 obidji(x-2,y+1,potez+1);
 obidji(x+2,y-1,potez+1);
 obidji(x+2,y+1,potez+1);
}

int main()
{  int a,b,c,d,i,j;
   for(i=0;i<8;i++)
    for(j=0;j<8;j++) tabla[i][j]=-1;
   scanf("%d%d%d%d%d", &a,&b,&c,&d,&granica);
    a--;b--;c--;d--; //pomicanje polja
    //tako da gornje levo polje bude 0-0
    obidji(a,b,0);
    if(tabla[c][d]!=-1) printf("Moze\n");
    else printf("Ne moze!\n");
    return 0;
}
```

Diskusija:
Promenljiva granica opisuje koliko najviše poteza skakač sme napraviti.

U glavnoj funkciji main postavimo sva polja 8*8 table na -1 kako bismo označili da su sva polja neposećena na početku rada programa.

U svako polje matrice tabla upisaćemo u koliko poteza smo stigli do neke pozicije na tabli. Kako sa 0 poteza dolazimo do startne pozicije, zato neposećena mesta postavljamo na početku na -1, a ne na 0.

U glavnoj funkciji, naredbama

```
    a--;b--;c--;d--; //pomicanje polja
```

postavljamo da gornje levo polje table ima poziciju 00, a donje desno polje da ima poziciju 77.

Funkcija obidji kao argument ima poziciju polja na tabli i broj poteza da bismo došli do tog polja na tabli. Rekurziju zaustavljamo:

1. Ako smo izašli van polja šahovske table
   if(x<0 || x>7 || y<0 || y>7) return; //lose polje
2. Ako smo potrošili dozvoljen broj poteza
   if(potez>granica) return; //potrosili smo poteze
3. Ako smo na polje xy došli sa ranije, sa ne više od *potez* poteza

if(tabla[x][y]!=-1 &&tabla[x][y]<=potez) return;

U funkciji obidji, u liniji tabla[x][y]=potez;
Zapisujemo u koliko poteza je skakač došao na polje xy.

Da li je ovaj uslov izlaska bio nužan, tj. da li smo mogli izbaciti ovaj uslov

3. Ako smo na polje xy došli sa ranije, sa ne više od *potez* poteza

if(tabla[x][y]!=-1 &&tabla[x][y]<=potez) return;

ODGOVOR: Da, ali bi program bio veoma spor, jer bi se rekurzija ponavljala sa istim ili lošijim parametrima
Na primer, za ulaz 1 1 8 8 5 program sa linojom
if(tabla[x][y]!=-1 &&tabla[x][y]<=potez) return;
napravi 729 poziva rekurzijem a bez tog poziva 22681 poziva rekurzije.

Program bez linije
if(tabla[x][y]!=-1 &&tabla[x][y]<=potez) return;
faktor DFS grananja programa je 8, te je složenost programa $O(8^g)$

Program sa linijom
if(tabla[x][y]!=-1 &&tabla[x][y]<=potez) return;
može najviše g=granica puta promeniti vrednost nekog polja matrice tabla, te je složenost programa O(64*g), jer tabla ima 64 polja

2. Napisati program koji unosi sa standardnog ulaza cele brojeva (1<=n,m<=50), a zatim n redova sa po m znakova
(samo 'x' i ' .'). Uneseni znakovi opisuju zemljiste cije ivice moraju biti oivicene znakovima 'x', tako da 'x' predstavlja kamen, a '.' predstavlja plodno tlo. Nakon unesenog opisa zemljista unose se dva broja koja predstavljaju red i kolone polja (nije kamen) gde se seje trava. Vas program mora da sve znakove '.' zameni sa '+' ako se na odgovarajuce mesto moze prosiriti trava. Trava se siri samo na susedna polja na kojima je plodno tlo. Susedna polja dele zajednicku ivicu. Ispisite izgled zemljista nakon sto se trava prosiri koliko god moze.

ULAZ	IZLAZ
4 10 xxxxxxxxxx x.....x.xx xxx...x.xx x..xxx..xx 2 2	xxxxxxxxxx x++++++x.xx xxx+++x.xx x..xxx..xx

Koristicemo rekurzivni DFS, koji nece raditi nikakvu pretragu, nego ce se jednostavno siriti.
Koristicemo za memoizaciju (obelezavanje obilaska plusicem) matricu zemlja i obeleziti znakom + svako polje matrice u koje dodjemo 1. put, a da to polje nije kamen, tj. jednako '.'

```
#include <stdio.h>

char zemlja[50][51];
void DFS(int x, int y)
{
   if(zemlja[x][y]!='.') return;
   zemlja[x][y]='+';
   DFS(x-1,y);
   DFS(x+1,y);
   DFS(x,y-1);
   DFS(x,y+1);
}
int main()
{
   int n,m,i,x,y;
   scanf("%d%d", &n,&m);
   for(i=0;i<n;i++) scanf("%s", zemlja[i]);
   scanf("%d%d", &x,&y);
   DFS(x-1,y-1);
   for(i=0;i<n;i++)
   printf("%s\n",zemlja[i]);
   return 0;
}
```

Kad zadatak ne bi jos u formulaciji ogranicio da zemljiste bude omedjeno kamenom, morali bismo u rekurziju dodati ogranicenje koje vodi racuna da ne istupimo van polja.
Vremenska slozenost ovog programa zbog memoizacije je O(m*n).
Slicno ovom zadatku se resavaju zadaci koji proveravaju moze li se od nekog mesta na zemljistu doci do nekog drugog mesta.

3. Resite zadatak 1, odnosno problem pronalaska minimalnog broj poteza da se skakacem od polja a-b dodje do polja c-d, ali tako na ne koristite matrice.

Ideja resenja: Koristicemo IDDFS tj. iterative deeping DFS kao dopunu tradicionalnoj DFS strategiji. Ideja se sastoji od uzastopnog pozivanja DFSa sa sve vecom granicom dubina do kojih se sme vrsiti grananje.

Ideja IDDFS je da prvo ogranicimo dubinu grananja na 1 i kad rekurzija zavrsi, pozovemo je ponovo ali sa granicom dubine 2, a zatim sa granicom 3,... sve dok ne uspemo doci do ciljne tacke pretrage. Tada znamo da je pronadjeno resenje na najmanjoj dubini, jer prethodni poziv koji je bio za jedan nivo manje dubine nije uspeo doci do resenja.

Ako je faktor grananja manji od 2, nije efikasno koristiti IDDFS.
U zadataku sa skakacem, faktor grananja je 8.

```
#include <stdio.h>
```

```
int granica,c,d;
int nasao=0;

void obidji (int x,int y,int potez)
{ if (x==c && y==d) nasao=1;
  if(x<0 || x>7 || y<0 || y>7) return; //lose polje
  if(potez>granica) return; //potrosili smo poteze
 obidji(x-1,y-2,potez+1);
 obidji(x-1,y+2,potez+1);
 obidji(x+1,y-2,potez+1);
 obidji(x+1,y+2,potez+1);
 obidji(x-2,y-1,potez+1);
 obidji(x-2,y+1,potez+1);
 obidji(x+2,y-1,potez+1);
 obidji(x+2,y+1,potez+1);
}

int main()
{  int a,b;
   scanf("%d%d%d%d", &a,&b,&c,&d);
    a--;b--;c--;d--; //pomicanje polja
    //tako da gornje levo polje bude 0-0
    for(granica=0;nasao==0;granica++)
    obidji(a,b,0);

    printf("Potrebno je minimalno %d poteza\n", granica);
    return 0;
}
```

Ovaj program za unos
1 1 8 8
otkriva da je potrebno minimalno 6 poteza.
U IDDFS verzijo ovaj program koristi 27334 puta rekurziju.
Da smo program prekinuli u 7. redu tako sto smo odmah nakon if selekcije pozvali exit(0), broj rekurzivnih poziva bi bio i do 8 puta manji, tj. 5799

4. Ilija je počeo da radi za veliku softversku kompanije. Hijerhija kompanije je u obliku stabla, gde svaka osoba (osim šefa) ima tačno jednog direktnog menadžera. Kompanija je podeljena u timove, tako da svaki programer i svi njegovi direktni i indirektni podređeni (ako postoje) formiraju tim. To znači da tim može biti sačinjen od drugih timova. Na primer u kompaniji poput Microsoft-a postoji tim koji radi na softverskom paketu Office, koji se sastoji od podtimova koji rade na softverskim alatima Word, Excel, itd.
U Ilijinom slučaju Stanko je šef kompanije, a njemu direktno su potčinjeni Ilija i Petar. Ilija je menadžer Kristijanu, Petar je menadžer Goranu i Tomi. Prema gore navedenim pravilima možemo zaključiti da Stanko, Ilija, Petar, Kristijan, Goran i Toma obrazuju tim. Ilija i Kristijan takođe formiraju tim, i Petar, Goran i Toma formiraju drugi tim.
Kompanija se upravo premestila u novu, veoma dugačku, ali, na žalost, usku kancelariju u kojoj postoji prostor za
samo jedan red kompjuterskih stolova. Šef je već zauzeo krajnje levo mesto, i želi da svakom svom radniku napravi

raspored sedenja tako da:
a) Direktni menadžer svakog programera sedi levo od programera.
b) Svi članovi tima zauzimaju uzastopne stolove (npr. sede jedan do drugog).
Ako razmotrimo navedeni primer programera Ilije i kolega, jedan moguć raspoed sedenja je, redom:
Stanko, Petar,
Goran, Toma, Ilija, Kristijan.
Pomozite Iliji da ostavi dobar utisak na svog šefa pisanjem programa, koji pronalazi moguć raspored sedenja
programera (za datu hijerarhiju kompanije u obliku stabla).

Ulaz
U prvoj liniji standardnog ulaza biće dat ceo broj N ($1 \leq N \leq 200000$) - broj programera u kompaniji. Neka su programeri predstavljeni brojevima od 1 do N, gde je sa 1 označen šef kompanije (koji nema direktnog menadžera). U sledećiht N – 1 linija dati su parovi celih brojeva W1 W2, koji označavaju da programer sa brojem W1 je direktan menadžer programera W2.

Izlaz
Na standardni izlaz štampati jednu liniju koja predstavlja raspored sedenja programera i koja sadrži N celih brojeva
između 1 i N koji su razdvojeni blankom. Ako postoji više od jednog rasporeda sedenja, ispisati onaj koji je
leksikografski najmanji. Raspored A je leksikografski manji od rasporeda B, ako prvi (najlevlji) broj po kom se oni
razlikuju je manji i nalazi se u rasporedu A. Na primer, raspored {1, 3, 4, 6, 5, 2, 7} je manji od rasporeda {1, 3, 5,
2, 4, 7, 6}.

Test primeri:
1.

Ulaz	Izlaz
6 1 2 2 5 4 3 1 4 4 6	1 2 5 4 3 6

2.

Ulaz	Izlaz
14 9 11 1 9 1 3 3 8 1 2 2 4 2 5 10 13 3 7 9 10 2 6	1 2 4 5 6 3 7 8 9 10 12 13 11 14

10 12
11 14
Obrazloženje: U prvom test primeru Stanko je označen brojem 1, Ilija je 2, Kristijan je 5, Petar je 4, Goran je 3,
Toma je 6.

Ideja:
Razmotrite drvo koje predstavlja hijererhiju kompanije. Možemo uočiti da direktni menadžer svakog programera će
biti levo od njega, kao i indirektni supervizori. Ovo nas dovodi do zaključka da je ovo DFS pretraga grafa
(DFS=pretraga grafa u dubinu) u svom tradicionalnom obliku. Dalje, kako nam je potrebno štampanje redosleda u
leksikografskom poretku, izvršićemo sortiranje čvorova grafa.
Pošto je u nekoliko test primera N>=10000, MORA se voditi računa o dubini rekurzije tj.o veličinu steka.
Ukupna vremenska složenost je:
O(n) za učitavanje ulaza + O(n log n) za sortiranje naslednika + O(n) za rekurziju = O(n log n)

Rešenje 1:

```
#include <cstdio>
#include <vector>
#include <algorithm>
#define MAX 1048576
//2^20=1048576
int main(void)
{
 FILE* in = stdin; FILE* out = stdout;
// in = fopen("Kompanija.in", "rt"); out = fopen("Kompanija.out", "wt");
 int numNodes;
 static std::vector <int> v[MAX];
 fscanf(in, "%d", &numNodes);
 for (int edge = 0; edge < numNodes - 1; edge++)
 {
    int from, to;
    fscanf(in, "%d %d", &from, &to);
    v[from].push_back(to);
 }
 for (int i = 1; i <= numNodes; i++)
   std::sort(v[i].rbegin(), v[i].rend());
 static int ans[MAX], ansSize = 0;
 static int index[MAX] = {0};
 static int stack[MAX], stackSize = 0;  stack[stackSize++] = 1;
 while (stackSize)
 {
   int node = stack[stackSize - 1];
   if (index[node] < (int)v[node].size())
     stack[stackSize++] = v[node][index[node]++];
   else ans[ansSize++] = stack[--stackSize];
 }
```

```
 for (int i = ansSize - 1; i >= 0; i--)
 fprintf(out, "%d%c", ans[i], i == 0 ? '\n' : ' ');
 return 0;
}
```

Rešenje 2:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> naslednici[200004];
bool d[200004];
int n,a,b,c,p[200004],shef;
void end(int tek){
   cout<<tek<<" ";
  for (int i=0;i<naslednici[tek].size();++i){
     end(naslednici[tek][i]);
 }
}

int main(){
 cin>>n;
 for (int i=0;i<n-1;++i){
   cin>>a>>b;
   d[b]=true;
   naslednici[a].push_back(b);
 }

 for (int i=1;i<=n;++i){
    for (int j=0;j<naslednici[i].size();++j){
        p[j]=naslednici[i][j];
     }
 sort(p,p+naslednici[i].size());

for (int j=0;j<naslednici[i].size();++j)
    naslednici[i][j]=p[j];
 }


 for (int i=1;i<=n;++i){
 if (!d[i]){shef=i;break;}
 }

 end(shef);cout<<endl;

 return 0;
}
```

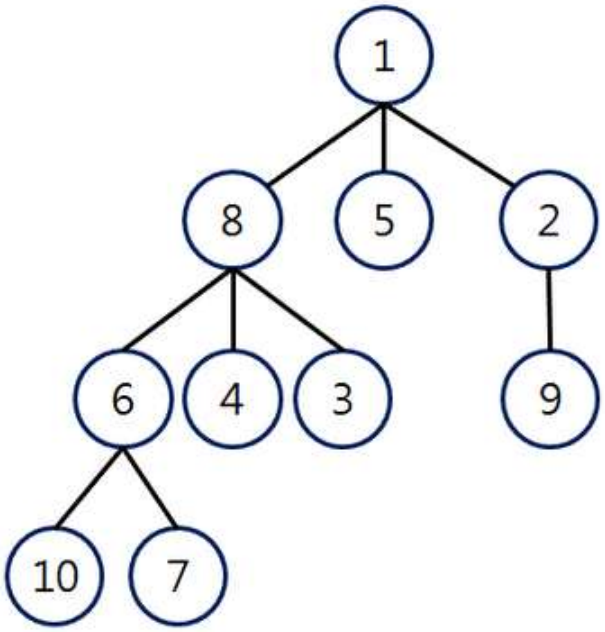
BFS

Opšti oblik BFS-a

Postavi početni čvor u red (queue)
Dok se ne nađe rešenje
{ Uzmi čvor iz reda i izvršiti obradu čvora
Čvorove nastale obradom postaviti u red
}

Ako se koristi programski jezik C, red (queue) koji je potreban BFS-u se mora isprogramirati od strane korisnika dok u C++ se moze koristiti vec implementirani *queue*.

Primer



Redosled:

$1 \rightarrow 8 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 9 \rightarrow 10 \rightarrow 7$

Ako je faktor grananja manji od 2, prikladnije je koristiti BFS umesto IDDFS. IDDFS je podesniji u slucajevima kad imamo malo raspolozive memorije, jer BFS sve cvorove postavlja u red, te mu je često potrebno više memorije.

5. Unose se dva broja n i m (1<=n,m<=1000) koji predstavljaju dimenzije plana grada koji se ucitava sa standardnog ulaza. Potom se ucitava n linija sa po m karaktera koji predstavljaju izgled grada. Svaki uneseni znak moze biti '.', 'X', 'o' ili 'G' tako da '.' predstavlja prostor za prolaz (npr. prohodna ulica), 'X' predstavlja prostor kojim se ne moze prolaziti (npr. zgrada), 'o' je krofna, 'G' je gladni čiča Gliša. Postoji samo jedan čiča Gliša. Čiča Gliša se može kretati gore, dole, levo i desno kroz grad. Napisati program koji će na standardni izlaz ispisati koliko najviše krofni može pojesti čiča Gliša i koliko najmanje koraka treba napraviti tako da može od početne lokacije doći do svake krofne koju može pojesti.

Primer
ULAZ

5 10
...XXXX.o.
...XXX..o.
o.oXXX..o.
......G...
.........X

IZLAZ
Moze pojesti 5 krofni!
Najdalji je 7 koraka!

Ideja resenja:

U funkciji main pronadjemo pojavljivanje znaka 'G' u matrici plana grada i pozovemo funkciju BFS

```
 for(i=0;i<n;i++)
   for(j=0;j<m;j++)
     if(plan[i][j]=='G') BFS(i,j);
```

Funkciju BFS(i,j) ćemo pozvati samo jednom, jer postoji samo jedan gladni čiča Gliša.

U funkciji BFS(x,y) najpre ispraznimo, tj. inicijalizujemo red za BFS, tj.
pocetak=kraj=0; //praznimo red

U red uvek ubacujemo tri podatka; lokacije x,y i udaljenost od polaznog mesta, tj. u funkciji BFS pozovemo uvek istim redom sledeće potprograme

```
 staviUred(x); staviUred(y);
 staviUred(0);//udaljenost od pocetne pozicije
```

Tim istim redom i uzimamo te elemente iz reda.

```
 x=uzmiIzReda(); y=uzmiIzReda(); koraka=uzmiIzReda();
```

U funkciji BFS, dokle god ima neobrađenih mesta u redu, vršimo obradu čvorova, tj. lokacija grada

```
 while(pocetak!=kraj) {
 //dok ne ispraznimo red
```

Pri toj obradi, može da se desi da izažemo van plana grda, i to je situacija lošeg polja, čiju obradu ignorišemo

```
 if (x<0 || x>=n || y<0 || y>=m) continue;  //lose polje
```

Pri toj obradi, ignorišemo i lokacije na planu na kom smo vec bili ili lokacije na kom je zgrada X

```
 else continue;  //preskoci mesto na kom si vec bio ili mesto na kom je zgrada
```

Pomoću znaka Q obeležavamo već obrađeno mesto na planu grada

```
      plan[x][y]='Q';
```

Kada naiđemo na lokaciju o, tj. lokaciju krofne, onda proveramo da li je pronađena krofna najdalja do sada i shodno tome popravljamo rešenje (nije nužna provera, razmislite o strategiji BFS-a)

```
    if(brKoraka<koraka)
     brKoraka=koraka;
```

Trenutna 4 susedna mesta za lokaciju x,y na planu grada, potom ubacujemo u red

```
 for(i=0;i<4;i++)
 {
    staviUred(x+smerX[i]);
    staviUred(y+smerY[i]);
    staviUred(koraka+1);
 }
```

```
 #include <stdio.h>
#define NMAX 1000
char plan[NMAX][NMAX+1];
int brKrofni, brKoraka, n, m;
int red[100*NMAX], pocetak, kraj;

void staviUred(int x)
{
 red[kraj]=x;
 kraj=(kraj+1)%(100*NMAX);
}

int uzmiIzReda()
{ int x;
   x=red[pocetak];
   pocetak=(pocetak+1)%(100*NMAX);
   return x;
}

void BFS(int x,int y)
{ int i, koraka;
  const int smerX[4]={0,0,-1,1};
  const int smerY[4]={1,-1,0,0};
  pocetak=kraj=0; //praznimo red
  staviUred(x); staviUred(y);
  staviUred(0);//udaljenost od pocetne pozicije
  while(pocetak!=kraj) {
  //dok ne ispraznimo red
  x=uzmiIzReda(); y=uzmiIzReda(); koraka=uzmiIzReda();
  if (x<0 || x>=n || y<0 || y>=m) continue;  //lose polje
  if (plan[x][y]=='.' || plan[x][y]=='G')
   plan[x][y]='Q';
  else if (plan[x][y]=='o')
  {
     plan[x][y]='Q';
     brKrofni++;
     if(brKoraka<koraka)
      brKoraka=koraka;
  }
  else continue;  //preskoci mesto na kom si vec bio ili mesto na kom je zgrada
  for(i=0;i<4;i++)
  {
```

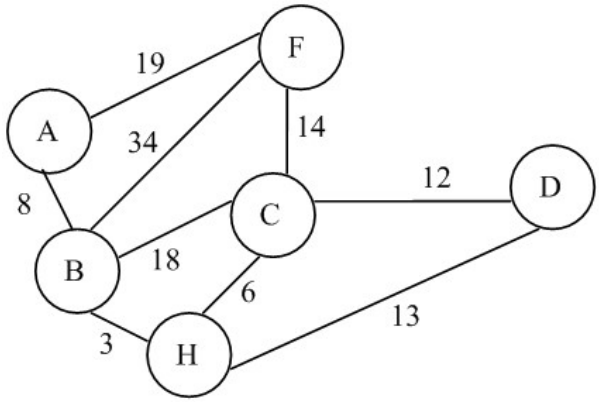
```
     staviUred(x+smerX[i]);
     staviUred(y+smerY[i]);
     staviUred(koraka+1);
  }
  }

}
int main()
{   int i,j;
    scanf("%d%d", &n,&m);
    for(i=0;i<n;i++) scanf("%s", plan[i]);
    for(i=0;i<n;i++)
     for(j=0;j<m;j++)
       if(plan[i][j]=='G') BFS(i,j);

    printf("Moze pojesti %d krofni!\n", brKrofni);
    printf("Najdalji je %d koraka!\n", brKoraka);
    return 0;
}
```

Složenost navedenog rešenja je O(n*m), jer BFS pređe svako polje samo jednom.

Svi ranije navedeni zadaci rešeni DFSom ili IDDFSom mogu se rešiti (to ne mora uvek biti slučaj) i pomoću BFSa u istoj ili manjoj složenosti.

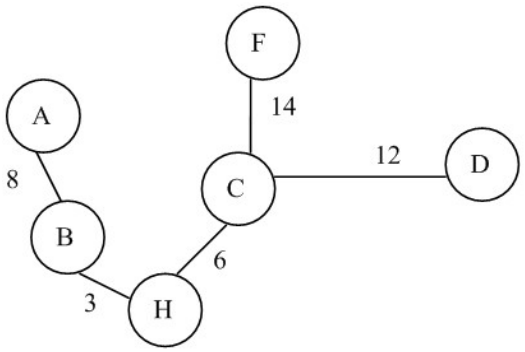
MCST

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



Rešenje:

PRIMov algoritam:
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
Cena MCST-a je 8+3+6+12+14=43

Prim-ov algoritam:
PRIM(G, s)
$U = \{s\}$
$E' = \varnothing$
while ($U \neq V$) **do**
    find $(u, v) \Rightarrow \min \{w(u, v) : (u \in U) \text{ and } (v \in (V - U))\}$
    $U = U + \{v\}$
    $E' = E' + \{(u, v)\}$
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)
Kompleksnost Kruskalovog algoritama je O(m·ln(n)).

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

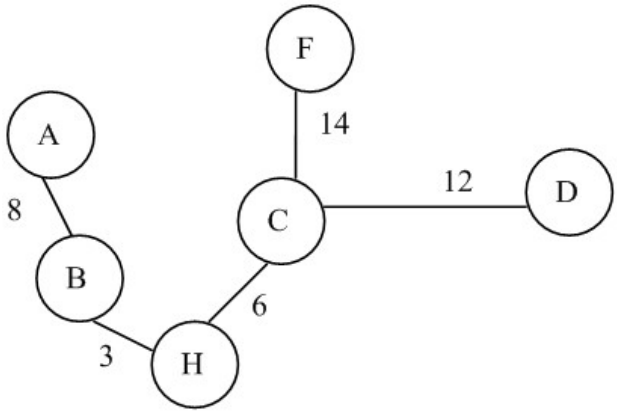
$MCST = (V, E')$

Sortirane grane {BH=3, CH=6, AB=8, CD=12, DH=13, CF=14, BC=18, AF=19, BF=34}

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

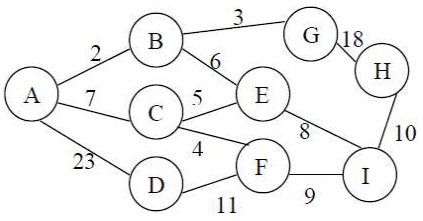
c)



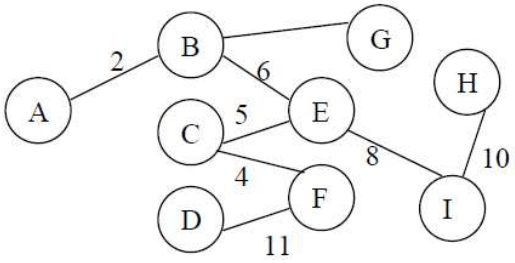
Poredjenje tri algoritma za konstrukciju MCST
Osnovne razlike između tri algoritama su:
• Prim-Jarnikov algoritam u svakom koraku proširuje označeno stablo sa najbližim čvorom,
• Kruskalov algoritam u svakom koraku spaja dva najbliža stabla sa novom granom,
• Borůvkin algoritam u svakom koraku spaja sva najbliža stabla sa novom granom.

Kruskalov i Borůvkin algoritam se mogu unaprediti tako da njihova kompleksnost bude $O(m \cdot \alpha(n))$, gdje je α inverzna Ackermanova funkcija.

8. 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

9. 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 <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();
}
```

10. 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* ≤ 1000, *M* ≤ 100000, *K* ≤ 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 ≤ *i*, *j* ≤ *N, Wi,j* ≤ 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();
}
```