Osnovni algoritmi pretraživanja

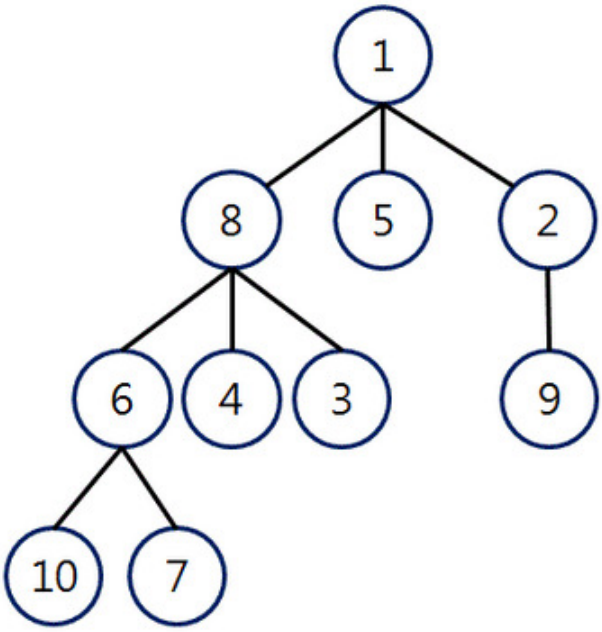
BFS (eng. breadth-first-search) je algoritam pretrage koji pretražuje nivo po nivo u stablu pretraživanja. BFS će najpre potražiti rešenje na dubini 1, potom na dubini 2, potom na dubini 3,...

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.

1. 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.

2.
U ravni je dato N pravougaonika sa dva naspramna temena. Poznato je da su stranice prevougaonika paralelne koordinatnim osama. Svi pravougaonici su zatvorene oblasti (dakle, sadrže svoju granicu). Smatra se da se pravougaonici seku ako imaju bar jednu zajedničku tačku (prema tome i na granici). Odrediti koliko likova nastaje spajanjem povezanih pravougaonika u jedan lik.

Rešenje. U rešenju se koristi funkcija Presek koja proverava da li se dva pravougaonika seku da bi se primenom DFS algoritma (pretrage po dubini) svi povezani pravougaonici spojili u jedan lik.

```
#include <iostream.h>
const int n=10;

struct TRect
{
double x1,y1,x2,y2;
}Pravougaonik[n]={
{10,5,20,20},{12,12,15,15},{8,20,21,21},{5,5,2,2},{4,4,1,1},{5,5,2,1},{12,12,10,10},{12,12,13,13},{1,3,3,5},{2,2,5
,5} };

bool used[n]; /* niz markera koji se koristi u DFS algoritmu*/

double min(double a, double b)
{return (a>b)? b:a; }
double max(double a, double b)
{re turn (a>b)? a:b; }

void Uredi(TRect &A)//(x1,y1)-donji levi ugao, (x2,y2)-gornji desni ugao
{
  double t;
       if (A.x1>A.x2) {t=A.x1; A.x1=A.x2; A.x2=t;}
       if (A.y1>A.y2) {t=A.y1; A.y1=A.y2; A.y2=t;}
}

bool Presek(TRect A, TRect B) // da li se pravougaonici A, B seku
{
  Uredi(A);
  Uredi(B);
   return (max(A.x1,B.x1)<=min(A.x2,B.x2)) && (max(A.y1,B.y1)<=min(A.y2,B.y2));
}

void DFS(int i) // pretraga po dubini
{
   used[i]=true;
    cout << i << " ";
   for (int k=0;k<n; k++)
```

```
        if (Presek(Pravougaonik [i], Pravougaonik [k]) && !used[k]) DFS(k);
}
void Ispis() // ispis likova
{
for (int i=0; i<n; i++) used[i]=false;
int k=0;
for (int i=0; i<n; i++)
if (!used[i])
{
  cout << "Lik " << ++k << "-> ";
   DFS(i);
   cout << endl;
}
}
void main()
{Ispis();}
```

3. Zadat je aciklički usmeren graf *G=(V,E).* Konstruisati algoritam linearne složenosti koji utvrđuje da li u G postoji neki prost put koji sadrži sve čvorove grafa.

REŠENJE:

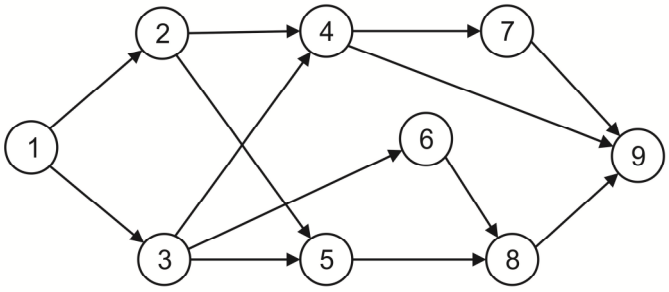
Put je prost, ako se svaki čvor pojavljuje u njemu samo jednom.
Hamiltonov put je prosti put u kom se svaki čvor grafa pojavljuje tačno jednom.
U ovom zadataku zahtev je da se konstruiše algoritam za pronalazak Hamiltonovog puta u grafu G.

Ideja je da se konstruiše topološki redosled čvorova grafa G, a potom se proveri da li postoji put tako što se proveri da li susedni čvorovi u topološkom redosledu jesu putno povezani. Ukoliko su povezani, onda Hamiltonov put čine topološki uređeni čvorovi.

Algoritam topološkog sortiranja acikličkog usmerenog grafa G je složenosti $O(|E| + |V|)$, tj. linearne složenosti kod grafova.
S druge strane, ako postoji Hamiltonov put, onda topološko sortiranje mora da dovede upravo do onog redosleda čvorova kojim se oni nižu u putu!!!

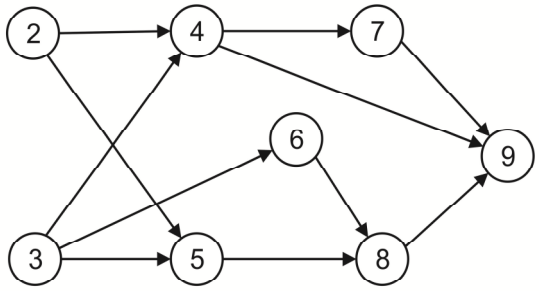
Algoritam topološkog sortiranja sastoji se u sledećem postupku. Pronađe se čvor bez ulaznih grana, iz sekvencijalnog reda čvorova u skupu V, i zapiše se čvor u niz. Potom se iz grafa ukloni taj čvor i obrišu sve grane koje polaze iz tog čvora. Za novodobijeni graf se ponovo nađe čvor bez ulaznih grana i zapiše u niz, zatim se iz grafa ukloni taj čvor i pobrišu sve grane koje polaze iz tog čvora itd.

Čvor bez ulaznih grana: 1 (postoji u usmerenom acikličkom grafu, zašto?)



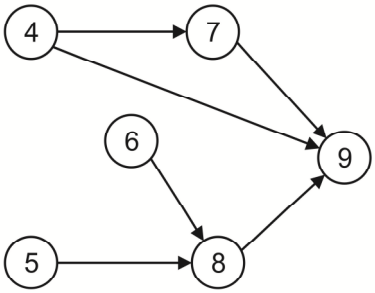
1								

Čvor bez ulaznih grana u novom grafu: 2, 3

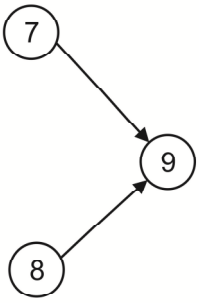


1	2	3						

Čvor bez ulaznih grana u novom grafu: 4, 5, 6

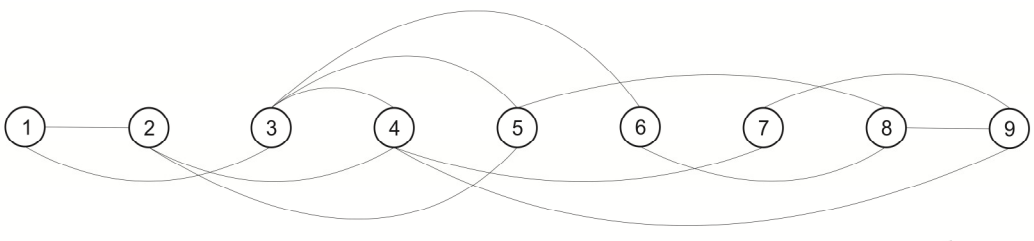


1	2	3	4	5	6			



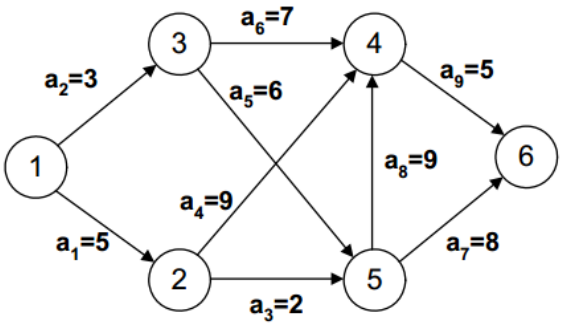
1	2	3	4	5	6	7	8	

1	2	3	4	5	6	7	8	9

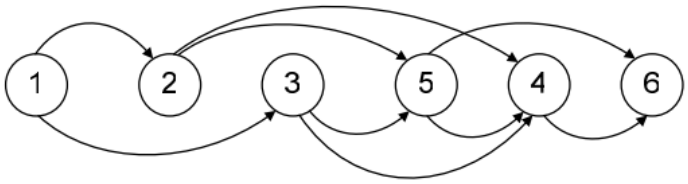


Da li su putno povezani 1,2? Jesu.
Da li su putno povezani 2,3? Nisu.

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

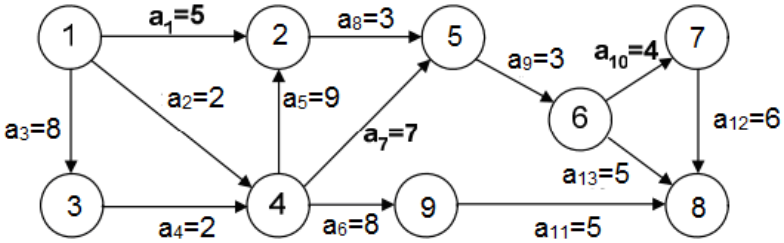


Rešenje:
Topološki redosled čvorova grafa je

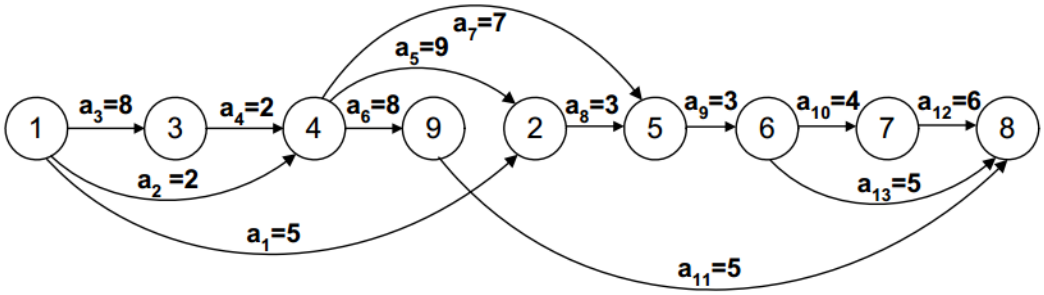


U ovom grafu topološki redosled čvorova nam ne daje Hamiltonov put, jer nisu svaka dva uzastopna čvora u topološkom redosledu putno povezani.

5. Pronaći topološki redosled čvorova za graf sa slike.



Rešenje:



6.

Student da bi mogao izaći na ispite koje je planirao da polaže mora obezbediti potpise profesora da je pratio njihova predavanja. Međutim, neki profesori su spremni da daju potpis za svoj ispit samo ako student ima potpis od njegovih kolega čija su predavanja bitna za razumevanje predmeta koji on drži.

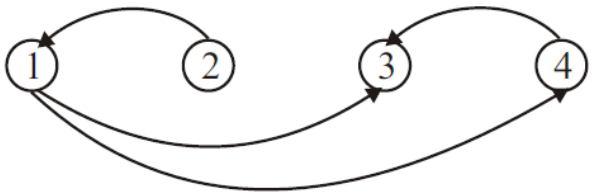
U prvoj liniji ulaznog fajla (in.txt) je broj profesora (1 <= N <= 100) od kojih student traži potpise. U sledećih N redova su potrebne informacije. Prvi broj K u liniji predstavlja broj profesora čiji su potpisi preduslov za potpis profesora I. Sledećih K brojeva označavaju te profesore.

Ako se ne mogu prikupiti potpisi svih profesora u izlazni fajl (out.txt) treba ispisati "NE". Ako se mogu prikupiti svi potpisi u prvom redu izlaznog fajla treba ispisati "DA", a u sledećih N redova - redosled prikupljanja potpisa.

(Objašnjenje: linija 1 2 ukazuje da je potpis profesora 1 preduslov za profesora 2, linija 0 ukazuje da za profesora 2 nema preduslova, linija 2 1 4 ukazuje da za profesora 3 postoje 2 preduslova: potpisi profesora 1 4, linija 1 1 ukazuje da za profesor 4 je uslov 1 potpis i to od profesora 1).

Na primer:

in.txt	out.txt
4 1 2 0 2 1 4 1 1	DA 2 1 4 3



Problem se može predstaviti orijentisanim grafom u kome čvorovi predstavljaju redne brojeve profesora, a grana od čvora J ka čvoru I izražava da je uslov za potpis profesora I potpis profesora J.

Čvor koji nema dolazne grane može se ukloniti – to je profesor kome ne trebaju potpisi drugih profesora. Ako je to I-ti čvor tada I-ta kolona ima sve nule. Pošto I-ti profesor daje potpis bezuslovno, tada se uklanjaju i sve grane od njega ka drugim profesorima (postavljaju se nule u I-tu vrstu matrice). Ovaj postupak ponavljati dok ima čvorova za uklanjanje. Ako je svih N čvorova uklonjeno – student je dobio sve potpise, a ako nije, a ne može se nastaviti uklanjanje to znači da postoji ciklus, pa se ne mogu se dobiti svi potpisi.

```
#include <iostream.h>
#include <fstream.h>
int a[101][101];
bool KolonaNula(int a[][101], int n, int j)
   {
   for (int i=1; i<=n; i++)
   if (a[i][j]) return false;
   return true;
   }
void main()
   {
   int n, i,j, k, p, b[101];
   bool Dalje, mark[101];
   ifstream f("in.txt");
   ofstream g("out.txt");
   f >>n;
   for(i=1; i<=n; i++)
     {
     f >> k; mark[i]=false;
     for (j=1; j<=k; j++)
       {
       f >> p;
       a[p][i]=1;
       }
     }
   Dalje=true; p=0;
while (Dalje) // dok ima cvorova za uklanjanje
  {
  Dalje=false;
  for (int i=1; i<=n; i++)
    if (!mark[i] && KolonaNula(a,n,i))
      {
      b[++p]=i;
      mark[i]=true;
      for (j=1; j<=n; j++) a[i][j]=0;// i-ti profesor daje potpise
      Dalje=true;
      }
  }
if (p==n)
  {
  g << "DA" << endl;
  for (i=1; i<=n; i++)g << b[i] << endl;
  }
else g << "NE";
f.close(); g.close();
}
```

7. 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;
}
```