Redukcije nekih problema na već rešavane

1. Algoritam za proveru da li postoji trougao u zadatom neusmerenom grafu

Problem: Neka je G = (V, E) neusmeren povezan graf sa n čvorova i m grana. Potrebno je ustanoviti da li u grafu postoji trougao, tj. da li postoje takva tri čvora da između svaka dva od njih postoji grana.

Direktno rešavanje obuhvata proveru svih tročlanih podskupova skupa svih čvorova grafa. Tih podskupova je $\frac{n(n-1)(n-2)}{6}$, a svaki od njih se može proveriti za konstantno vreme, stoga bi složenost ovog algoritma bila $O(n^3)$.

Neka je matrica A matrica povezanosti grafa G. Pošto je graf neusmeren, matrica A je simetrična. Posmatrajmo matricu B koja je jednaka A^2

–B[i, j] je tada jednako $\sum_{k=1}^{n}$ A[i, k] * A[k, j].

Iz ove jednakosti sledi uslov da je B[i, j] > 0 akko postoji čvor v_k takav da je k različito i od i i od j, i da su oba čvora v_i i v_j povezana sa v_k. Prema tome u grafu postoji trougao akko postoje takvi indeksi i i j da je A[i, j] = 1 i B[i, j] > 0. Stoga, treba izračunati matricu V i proveriti ispunjenost uslova A[i, j] = 1 i B[i, j] > 0 za svih n^2 parova čvorova. Za množenje matrice se može iskoristiti Štrasenov algoritam i tako se dobija algoritam složenosti $O(n^{2,81})$.

2. Dokazati da ako postoji algoritam složenosti *O(T(n))* za množenje kvadratne *n*×*n* donje trougaone matrice kvadratnom *n*×*n* gornje trougaonom matricom, onda postoji algoritam složenosti $O(T(n)+n^2)$ za množenje dve proizvoljne *n*×*n* matrice. Pretpostaviti da *T(cn)=O(T(n))* za proizvoljnu konstantu *c* > 0.

REŠENJE:

Neka su A i B dve proizvoljne kvadratne matrice reda n .

Svaka od njih se moze predstaviti kao zbir po jedne gornje i po donje trougaone matrice (T_A, B_A, T_B, B_B, redom):

$A= T_A + B_A$

$B= T_B + B_B$

Dalje je : $AB=(T_A + B_A) (T_B + B_B)=T_AT_B + B_AB_B + B_AT_B + T_AB_B$

Ako se upotrebi algoritam iz formulacije zadataka moguce je izracunati proizvod :

$$\begin{vmatrix} B_A & 0 \\ T_A & B_A \end{vmatrix} \times \begin{vmatrix} T_B & B_B \\ 0 & T_B \end{vmatrix} = \begin{vmatrix} B_AT_B & B_AB_B \\ T_AT_B & B_AT_B + T_AB_B \end{vmatrix}$$

Kao rezultat dobija se matrica koja sadrzi blokove: B_AB_B , T_AT_B , $B_AT_B + T_AB_B$

Ovi blokovi ucestvuju u izracunavanju proizvoda A × B.

Na taj nacin se problem izracunavanja proizvoda dve proizvoljne matrice svodi na problem izracunavanja proizvoda kvadratne donje trougaone matrice kvadratnom gornjom trougaonom matricom.

Ukupno vreme izvršavanja : $O(T(2n)+n^2)=O(T(n)+n^2)$

3. Ako je dat algoritam za mnozenje dve n * n donje trougaone matrice čije vreme izvrsavanja je O(T(n)), dokazati da postoji algoritam za mnozenje dve proizvoljne n *n matrice cije vreme izvrsavanja je $O(T(n)+n^2)$. (Moze se pretpostaviti da je T(cn)=O(T(n)) za svaku konstantu c)

REŠENJE: Neka su A i B dve proizvoljne kvadratne matrice reda n .

Svaka od njih se moze predstaviti kao zbir po jedne gornje i po jednedonje trougaone matrice(T_A, B_A, T_B, B_B, redom):

$A = T_A + B_A$

$B = T_B + B_B$

Dalje je : $AB = (T_A + B_A)(T_B + B_B) = T_A T_B + B_A B_B + B_A T_B + T_A B_B$

Ako se upotrebi algoritam iz formulacije zadataka moguce je izracunati proizvod :

$$\begin{bmatrix} B_A & 0 \\ T_A & B_A \end{bmatrix} \times \begin{bmatrix} B_B & 0 \\ T_B & B_B \end{bmatrix} = \begin{bmatrix} B_A B_B & 0 \\ T_A B_B + B_A T_B & B_A B_B \end{bmatrix}$$

Kao rezultat dobija se matrica koja sadrzi blokove: $B_A B_B$, $T_A T_B$, $B_A T_B + T_A B_B$

Ovi blokovi ucestvuju u izracunavanju proizvoda $A \times B$.
Matrice $(T_A)^T$ i $(T_B)^T$ su donje trougaone matrice, te se upotrebom datog algoritma moze izracunati blok $T_A T_B$ na sledeci nacin:

$T_A T_B = ((T_B)^T (T_A)^T)^T$
Ukupno vreme izvršavanja je $O(T(2n)+n^2) = O(T(n)+n^2)$.
Na taj nacin se problem izracunavanja proizvoda proizvoljne dve matrice svodi na problem izracunavanja proizvoda dve kvadratne donje trougaone matrice.

4. Zadat je niz rekurzivno tako da je: **F[n] = A*F[n-1] + B*F[n-2] + C*F[n-3],**. Date su početne vrednosti niza **F[1], F[2], F[3]** i konstante **A, B, C. Svaki od učitanih brojeva je iz intervala [0, 10000]. Konstruisati algoritam složenosti O(log n) koji će pronaći vrednost niza (F[n], F[n] mod 10000)** za zadato veliko *n* iz intervala **[1, 2000000000].**.

Rešenje:
Jasno je da postoji algoritam reda O(n).
Na primer, ако je n>4 onda: **for (i=4;i <n;i++) f[i]=a*f[i-1]+b*f[i-2]+c*f[i-3].**

Vremensko poboljšanje je moguće ako se problem svede na množenje matrica 4x4.

$$\begin{bmatrix} f4 & f3 & f2 & f1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} A & 1 & 0 & 0 \\ B & 0 & 1 & 0 \\ C & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} f5 & f4 & f3 & f2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Analogno,

$$\begin{bmatrix} f5 & f4 & f3 & f2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} A & 1 & 0 & 0 \\ B & 0 & 1 & 0 \\ C & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} f6 & f5 & f4 & f3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Dakle, $$\begin{bmatrix} f4 & f3 & f2 & f1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} A & 1 & 0 & 0 \\ B & 0 & 1 & 0 \\ C & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{n-4} = \begin{bmatrix} f_n & f_{n-1} & f_{n-2} & f_{n-3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Komapktnije to možemo zapisati: $F_4 \times M^{n-4} = F_n$

Dakle, algoritam se sastoji od stepenovanja matrice M^{n-4}, šo je reda $O(\log_2 n)$. Da li se slažete sa tom složenošću? Obrazložite!!!
Nakon toga se rezultujuća matrica množi sa leve strane vektorom F_4. Rezultat koji se traži bio bi prva koordinaa vektora F_n.

```
#include <cstdio>
#include <iostream>

using namespace std;

struct matrix {
    int p[5][5];

    void init (){
      for ( int i=0;i<4;++i )
        for ( int j=0;j<4;++j )
          p[i][j] = 0;
    }

    matrix operator * ( matrix x ){
        matrix ret;
        ret.init();
        for ( int i=0;i<4;++i ){
          for ( int j=0;j<4;++j ){
            int z=0;
          for ( int k=0;k<4;++k )
              z = ( z + (p[i][k]*x.p[k][j])%10000 ) % 10000;
        ret.p[i][j] = z%10000;
        }
       }
      return ret;
     }


    };

void print ( matrix x ){
       for ( int i=0;i<4;++i,printf("\n") )
       for ( int j=0;j<4;++j )
         printf ("%d ",x.p[i][j]);
         return;
    }

matrix power ( matrix mat, int exp ){
    if ( exp == 1 ) return mat;
    if ( !(exp%2) ){
      matrix tmp = power ( mat,exp/2 );
     return tmp*tmp;
      }
    else return ( mat * power(mat,exp-1) );
    }

int main(){
  matrix m,ret,f4;
  int b[4],f[5];
  int n,sol;
```

```
  ret.init();

  scanf("%d%d%d",&f[0],&f[1],&f[2]);
  scanf("%d%d%d",&b[2],&b[1],&b[0]);
  scanf("%d",&n);

  if ( n < 4 ) printf ("%d\n",f[n-1]);
  else {
     f4.init();
     m.init();
     f4.p[0][0] = ( f[0]*b[2] + f[1]*b[1] + f[2]*b[0] ) % 10000;
     for ( int j=0;j<3;++j ) f4.p[0][j+1] = f[2-j];
      for ( int i=0;i<3;++i ) { m.p[i][0] = b[i] % 10000; m.p[i][i+1] = 1; }
     m = power(m,n-4);
     ret = f4 * m;
     sol = ret.p[0][0]%10000;

     printf ("%d\n",sol);
    }

  return 0;
  }
```

Zaključak: Linearne rekurentne formule sa **κ** nezavisnih početnih koeficijenata i **κ** zadatih početnih vrednosti niza mogu da se svedu na $F_{k+1} \times M^{n-k-1} = F_n$, gde su sve matrice F **(k+1)x(k+1).**

5. Ukupno 2n studenata konkurisalo je na n fakulteta. Razmotrimo bipartitni graf formiran od dva skupa čvorova - studenata i fakulteta, pri čemu grana izmedju nekog studenta i nekog fakulteta postoji akko student zadovoljava uslove za prijem na taj fakultet. Konstruisati algoritam za maksimiziranje broja primljenih studenata, uz uslov da nijedan fakultet ne primi više od dva studenta. Problem rešiti svodjenjem na običan problem bipartitnog uparivanja.

Rešenje:
Redukciju izvodimo na sledeći način:
1. svaki čvor koji odgovara fakultetu zamenjujemo sa dva čvora, povezana granama sa svim studentima koji su se kvalifikovali za prijem na taj fakultet.

Ovim je inicijalni problem sveden na obični problem bipartitnog uparivanja.

6. U školi ima n kurseva i n nastavnika. Posmatra se bipartitni graf sa dva skupa od po n čvorova (kursevi i nastavnici), pri čemu grana izmedju kursa i nastavnika postoji ako je nastavnik kvalifikovan za održavanje kursa. Svaki kurs mogu da drže najviše dva nastavnika, a svaki nastavnik može da drži samo dva kursa. Konstruisati algoritam (svođenjem na neki poznati problem) za maksimiziranje ukupnog broja kursevima koji mogu da se održe.

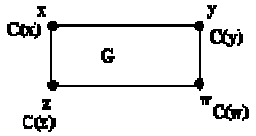
Rešenje:
Problem svodimo na problem bipartitnog uparivanja. Polazni graf je G=(U,V,E), U skup nastavnika, V skup kurseva, E skup grana.
Na osnovu grafa G formiramo novi graf G'=(U', V', E') tako da svakom nastavniku $u \in U$ odgovaraju dva čvora u0, $u1 \in U'$

Svakom kursu $v \in V$ odgovaraju dva kursa $v0,v1 \in V'$, a svakoj grani $(u,v) \in E$ odgovaraju 4 grane (u0,v0), (u0,v1), (u1,v0), $(u1,v1) \in E'$

Svakom bipartitnom uparivanju u G'odgovara takvo dodeljivanje kurseva nastavnicima u G da svaki nastavnik drži najviše dva kursa i svaki kurs drži najviše 2 nastavnika. Pri tom je ukupan broj kurseva jednak veličini odgovarajućeg uparivanja u G'.

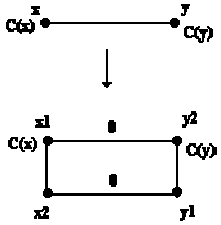
7. Dat je usmeren graf G=(V,E) sa izabranim čvorom v, takav da je svakom čvoru x pridružen pozitivan broj c(x). Cena usmerenog puta v, x1, x2, . . ., xk, u je definisana kao c(x1) + c(x2) + . . . +c(xk). Cene krajnjih tačaka puta, čvorova v, u se ignorišu(tj.ako grana (v,u) pripada grafu, cena puta od v do u je 0). Opisati algoritam za određivanje najjeftinijeg puta od v do svih ostalih čvorova.

REŠENJE:



Da li se moze primeniti algoritam za određivanje najkraćih puteva od v do ostalih čvorova u grafu? Napomena: težine su pridružene čvorovima (ne granama).

Konstruisati novi graf F sa dva čvora w1 i w2 za svaki čvor w grafa G.
Skup grana odredjen je sa:
1. za svaku granu (w,u) grafa G, grafu F dodati granu (w2,u 1) i pridruziti joj cenu 0.
2. za svaki cvor u grafa G, dodati grafu F granu (u1,u2) i pridruziti joj cenu c(u).



Takvim nacinom formiranja grafa F se dati problem svodi na određivanje najkraćih puteva od čvora v2 do svih ostalih čvorova u grafu F.

Pretraga i grananje sa odsecanjem

Problem (bojenja grafa): Potrebno je dodeliti svakom čvoru grafa jednu boju tako da nijedan sused tog čvora nije obojen tom istom bojom. Problem se može rešiti primenom algoritma za obilazak stabla, što predstavlja suštinu pretrage i grananja sa odsecanjem.

3 – bojenje grafa: Na početku obojimo dva susedna čvora grafa proizvoljnim bojama, ovo je početno stanje i predstavlja koren stabla. Stablo konstruišemo tokom samog obilaska. Zatim za svaki čvor stabla biramo sledeći čvor koji bojimo i dodajemo u stablo kao sinove one boje kojima se taj čvor može obojiti. Ako su obojeni svi čvorovi grafa, problem je rešen, a ako naiđemo na čvor koji ne možemo da obojimo onda "odsecamo" tu granu i vraćamo se unazad u stablu i tražimo grane koje nismo obišli.

1. Datim algoritmom obojiti zadate grafove, ako ih je moguće obojiti sa tri boje.

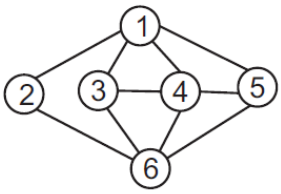
Algoritam 3-bojenje(G,U)
Ulaz G = (V,E) (neusmereni graf)
U (skup čvorova koji su već obojeni, inicijalno je prazan)
Izlaz validno 3-bojenje, ako takvo postoji
begin
if U = V then printf "bojenje je zavrseno";
  halt;
else
  izaberi čvor v koji nije u U;
  for c := 1 to 3 do
    if (ne postoji sused čvora v obojen bojom c) then dodaj čvor v obojen bojom c u U;

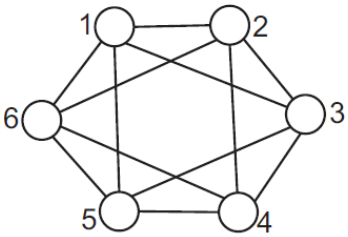
3-bojenje(G, U);
end
Zadatak nam je da svakom čvoru dodelimo jednu od 3 boje, tako da susedni čvorovi budu obojeni različitom bojom. Bojimo neki od čvorova, a zatim se pokušava sa svim bojama za sledeće čvorove, tako da bojenje ostane konzistentno. Ovaj proces se može obaviti algoritmom za obilaženje stabla koji i predstavlja suštinu pretrage i grananja sa odsecanjem.
Na početku biramo neka dva susedna čvora i bojimo ih nekim dvema različitim bojama. Ovo je početno stanje problema i njega pridružujemo korenu. Zatim za svaki čvor stabla biramo sledeći čvor grafa za bojenje i dodajemo 1,2 ili 3 sina čvoru u zavisnosti od toga kolikim brojem boja se može konzistentno obojiti.

U slučaju da su obojeni svi čvorovi grafa, rešili smo problem. Ako, međutim, naiđemo na čvor koji se ne može obojiti, vraćamo se nazad i pokušavamo sa drugim sinovima.
Algoritam za obilazak stabla konstruišemo induktivno:
Induktivna Hipoteza: Umemo da završimo 3-bojenje grafa sa manje od n neobojenih čvorova ili da ustanovimo da se ne mogu obojiti sa 3 boje.

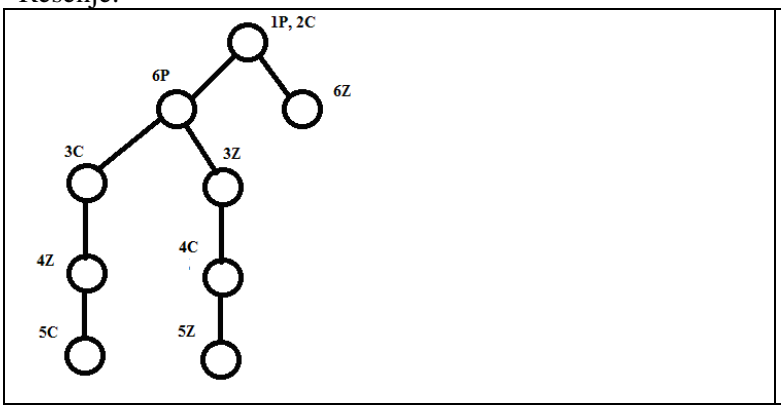
Induktivni Korak: Imamo graf sa n neobojenih čvorova, biramo proizvoljno jedan i dodajemo onoliko sinova sa koliko se boja može obojiti - ako je 0, započeto bojenje se ne može kompletirati. Inače čvor bojimo 1 po 1 mogućom bojom i kompletiramo bojenje (to su sad problemi sa po n-1 neobojenim čvorom koje po induktivnoj hipotezi znamo da rešimo).

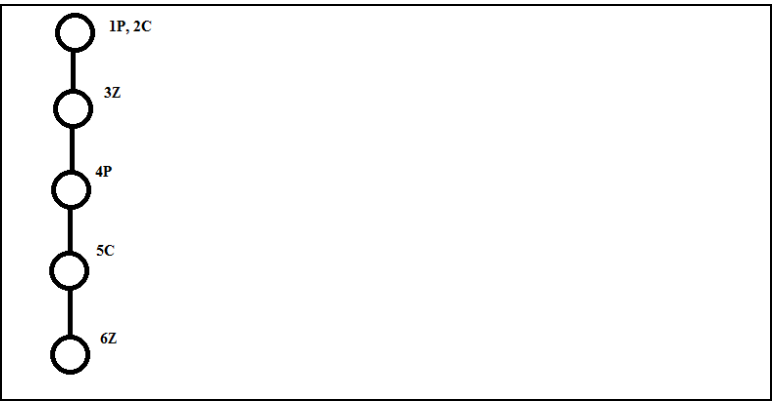
2. Datim algoritmom obojiti zadati graf. Ako je moguće obojiti ga sa tri boje.



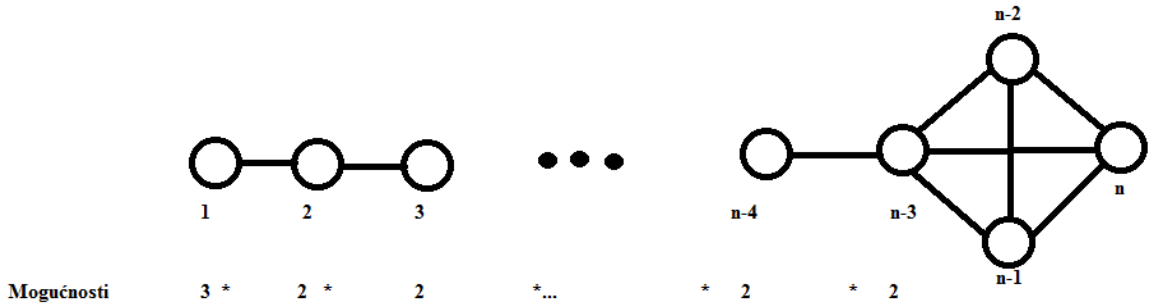


Rešenje:



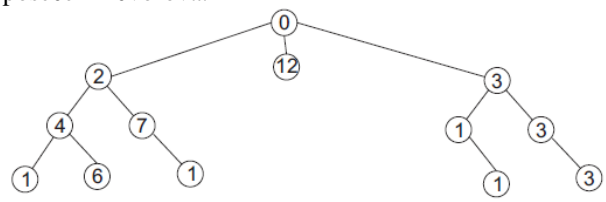


3. Navesti primer familije grafova, za koju trajanje bojenja sa tri boje algoritmom pretrage raste eksponencijalno sa brojem čvorova n.
Rešenje:
Posmatrajmo familiju grafova sa slike:

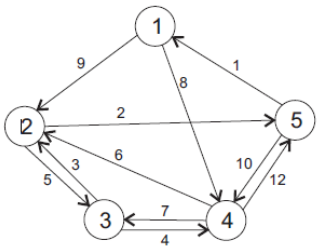


Ako su čvorovi ovako numerisani, onda algoritam bojenja pretragom mora da redom proveri svih $3*2^{n-4}$ ispravnih bojenja podgrafa koji čine čvorovi 1, 2,..., n-3
Svaki taj pokušaj završava se neuspehom kada se dođe do čvora n koji ima 3 susedna čvora: n-3, n-2, n-1, a svi oni moraju biti obojeni različitim bojama.
U ovom algoritmu bitna je numeracija čvorova: ako bi čvorovi n, n-1, n-2, n-3 bili numerisani sa 1,2,3,4, onda bi se posle O(1) koraka ustanovilo da traženo bojenje ne postoji.

4. Za dato stablo grananja sa odsecanjem, gde brojevi u čvorovima označavaju inkrementalne cene posećivanja tog čvora, a listovi kompletna rešenja, pri problemu minimizacije ukupne cene puta, obeležiti redosled izvršavanja pretrage u dubinu sa odsecanjem redosledom posećenih čvorova.



5. Rešiti problem trgovačkog putnika za zadati graf, tehnikom grananja sa odsecanjem.



Resenja 1-4-3-2-5-2 8+7+3+2+1
1-2-3-4-5 9+5+4+12+1

```
The Cost Matrix is:

        0       9       0       8       0
        0       0       5       0       2
        0       3       0       4       0
        0       6       7       0       12
        1       0       0       10      0

The Path is:

1 ===> 4 ===> 3 ===> 2 ===> 5===> 1

Minimum cost:21
```

```
#include<stdio.h>
#include<stdlib.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
int i,j;
//Number of Cities:
scanf("%d",&n);

//Cost Matrix
for( i=0;i<n;i++)
    {
    for( j=0;j<n;j++)
    scanf("%d",&a[i][j]);
    visited[i]=0;
    }
 printf("\n\nThe Cost Matrix is:\n");
 for( i=0;i<n;i++)
    {    printf("\n\n");
    for( j=0;j<n;j++)
        printf("\t%d",a[i][j]);
        }
```

```
 }
 void mincost(int city)
{int i,ncity;
visited[city]=1;
printf("%d ===> ",city+1);
ncity=least(city);

if(ncity==999)
    {
     ncity=0;
     printf("%d",ncity+1);
     cost+=a[city][ncity];
     return;
     }
 mincost(ncity);
}

int least(int c)
{
 int i,nc=999;
 int min=999,kmin;
 for(i=0;i<n;i++)
    {
     if((a[c][i]!=0)&&(visited[i]==0))
     if(a[c][i]<min)
     {
      min=a[i][0]+a[c][i];
      kmin=a[c][i];
      nc=i;
      }
     }

if(min!=999)
cost+=kmin;
return nc;
}
void put()
{printf("\n\nMinimum cost:"); printf("%d",cost);}

int main()
{get(); printf("\n\nThe Path is:\n\n"); mincost(0);put();
}
```

6. Dat je neusmeren acklički graf. Opisati algoritam za nalaženje maksimalnog nezavisnog skupa u datom grafu, čija je složenost i dalje eksponencijalna, ali manja od 2^n.