Algoritmi i strukture podataka
vežbe 7

Mirko Stojadinović

14. novembar 2013

1 Algoritmi za rad sa grafovima - nastavak

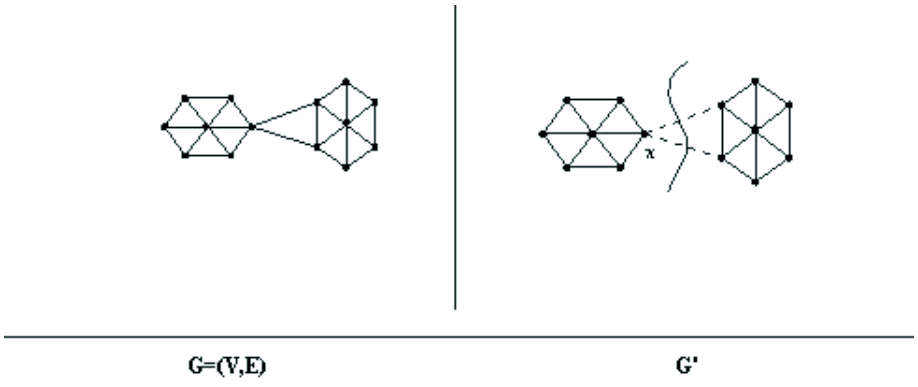
1. Ako je graf G povezan, onda će algoritmom pretrage u dubinu svi njegovi čvorovi biti označeni, a sve njegove grane biće u toku izvršavanja algoritma pregledane bar po jednom.

Rešenje: Zadatak prestavlja lemu iz profesorove knjige, pogledati knjigu za rešenje.

2. Konstruisati algoritam linearne složenosti koji u povezanom neusmerenom grafu G=(V,E) pronalazi čvor takav da i nakon izbacivanja tog čvora iz grafa, graf i dalje ostaje povezan. **Napomena:** Ako nije drugačije rečeno smatraće se da je graf predstavljen listom povezanosti u svim narednim zadacima. Linearna složenost kod grafova je O(|V| +|E|).

Rešenje:

Primer izbacivanja čvora kada graf postaje nepovezan:



Graf je povezan ako (u njegovom neusmerenom) obliku postoji put između ma

koja dva čvora. Neusmereni oblik usmerenog grafa G=(V,E) jeste graf G bez orijentacije nad granama. Kao rezultat primene algoritma DFS dobija se DFS stablo. Koren pretrage grafa u dubinu, tj. čvor v jeste koren DFS stabla.
Obilazeći graf G pretragom u dubinu (DFS) gradi se stablo i označavaju čvorovi.

1. Pri tome neki čvorovi će biti listovi DFS stabla.
2. Na početku se stavi da svaki v iz V jeste list, tj. v.indikator=1
3. Ako je čvor x iz V povezan sa bar jednim neoznačenim čvorom, onda je to dovoljno da odbacimo x kao list, tj. x.indikator=0 (jer se nalazi nivo iznad lista)
4. Na kraju, ako se izbaci neki čvor koji jeste list DFS stabla, novi graf G' (dobijen iz G bez x) će ostati povezan

```
Algoritam Izbaciti(G,s)
ulaz: G , x (cvor neusmerenog i povezanog grafa G=(V,E) )
izlaz: za svaki v vazi da v.indikator=1 ako cvor v moze biti izbacen iz G
{
 oznaiti x;
 x.indikator=1;

 for each (x,v)
    if not oznacen(v)
    {
       Izbaciti(G,v);
       x.indikator=0;
    }
}
```

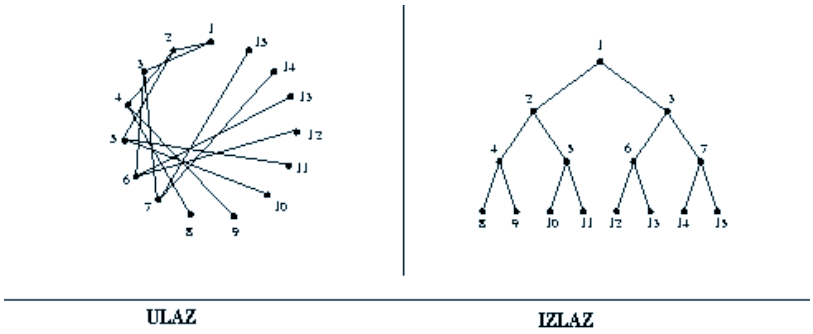
Vremenska složenost ovog algoritma (koji koristi DFS pretragu) odgovara vremenskoj složenosti DFS algoritma za povezane neusmerene grafove O(|V| +|E|) (videti zadatak 3), dok je broj rekurzivnih pokretanja |V| .

Da li izbacivanje čvora može da se obavi za svaki povezan graf G? Može, jer za svaki povezan graf postoji odgovarajuće DFS stablo. Svako stablo ima bar jedan list i to stablo ostaje povezano kada se list izbaci, te povezani neusmeren graf zaista sadrži čvor koji se moze izbaciti, a da novi graf ostane povezan.

3. Dat je usmereni graf G=(V,E) i čvor v skupa V. Konstruisati algoritam složenosti O(|E| + |V|) koji ispituje da li G jeste korensko stablo sa korenom v. Smatrati da su u korenskom stablu grane usmerene od korena ka listovima.

Rešenje:

Korensko stablo jeste usmereno stablo koje poseduje posebno izdvojen čvor koji se zove koren, a sve grane su usmerene od korena.



Na zadati usmeren graf G=(V,E) treba primeniti algoritam za obilazak grafa smatrajući pri tome da je čvor v njegov koren i brojeći čvorove kroz koje se prolazi. Ukoliko pri obilasku dođemo do grane koja vodi do čvora koji je već posećen, graf G ima ciklus(e), te nije stablo; ukoliko se obilazak završi sa brojem čvorova kroz koje se prošlo koji je manji od |V|, graf G nije povezan, te nije stablo; inače graf G jeste stablo.

```
Algoritam obilazak1(G,v,broj)
ulaz: G=(V,E,broj,indikator) (usmeren graf), v (cvor grafa G),
      broj (adresa brojaca), indikator (adresa indikatora
      da li graf ima cikluse)
izlaz: brojac je uvecan za broj cvorova kroz koje se proslo
{
   markiraj v;
   *broj=*broj+1 ;
   za sve grane (v,w) do
      if markiran(w)
         *indikator = -1;
      else
         obilazak1(G,w);
}

Algoritam  provera1(G,v)
ulaz: G=(V,E) (usmeren graf), v(cvor grafa G)
izlaz: odgovor na pitanje da li je graf G sa korenom v stablo
{
   broj = 0;
   indikator = 0;
   obilazak1(G,v,&broj,&indikator);
   if indikator == -1
      write ('graf G ima cikluse (u neusmerenom obliku)');
   else if broj < |V|
      write ('graf G nije povezan');
   else
      write ('graf G je stablo');
```

```
}
```

4. Primer predstavljanja grafa preko niza listi povezanosti (jednostruko povezanih listi) svakog od čvorova grafa. U programu se unosi graf i DFS algoritmom se za svaki čvor i u promenljivoj *posecen[i]* čuva indikator da li je dostižan iz čvora 0.

Rešenje:

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

typedef struct _cvor_liste
{
   int broj; /* indeks suseda */
   struct _cvor_liste* sledeci;
} cvor_liste;

#define MAX_BROJ_CVOROVA 100
cvor_liste* graf[MAX_BROJ_CVOROVA]; /* Graf predstavlja niz pokazivaca na pocetke listi suse

int broj_cvorova;
int posecen[MAX_BROJ_CVOROVA]; /* niz markiranih \v cvorova u DFS algoritmu */

/* Ubacivanje na pocetak liste */
cvor_liste* ubaci_u_listu (cvor_liste* lista, int broj)
{
  cvor_liste* novi=malloc (sizeof(cvor_liste));
  novi->broj=broj;
  novi->sledeci=lista;
  return novi;
}

void obrisi_listu(cvor_liste* lista)
{
  if (lista)
  {
    obrisi_listu(lista->sledeci);
    free(lista);
  }
}

void ispisi_listu(cvor_liste* lista)
{
  if (lista) { printf("%d ",lista->broj);
  ispisi_listu(lista->sledeci); }
}
```

```
/* Rekurzivna implementacija DFS algoritma */
void DFS(int i)
{
  cvor_liste* sused;
  printf("Posecujem cvor %d\n",i);
  posecen[i]=1;

  for(sused=graf[i]; sused!=NULL; sused=sused->sledeci)
    if (!posecen[sused->broj])
      DFS(sused->broj);
}

int main()
{
  int i, j, br_suseda, sused;
  printf ("Unesi broj cvorova grafa : ");
  scanf("%d",&broj_cvorova);

  for (i=0; i<broj_cvorova; i++)
  {
    graf[i]=NULL;
    printf ("Koliko cvor %d ima suseda : ",i);
    scanf("%d",&br_suseda);

    for (j=0; j<br_suseda; j++)
    {
      do
      {
        printf ("Unesi broj %d. suseda cvora %d : ",j+1,i);
        scanf("%d",&sused);
      }
      while (sused<0 || sused>=broj_cvorova); // u petlji je dok se ne unese ispravan sused

      graf[i]=ubaci_u_listu (graf[i],sused);
    }
  }
  for (i=0; i<broj_cvorova; i++)
  {
    printf("%d - ",i);
    ispisi_listu(graf[i]);
    printf("\n");
  }
  DFS(0);
  return 0;
```

```
}
```

Složenost DFS algoritma koji koristi liste povezanosti je $O(\mid V \mid + \mid E \mid)$. Zašto? Pogledati profesorovu knjigu za objašnjenje.

Zadatak: Napisati prethodni program tako da se umesto lista za predstavljanje suseda koristi niz (kao na prethodnom času).

5. Napisati program koji učitava grane usmerenog grafa od n čvorova i za par datih čvorova (čvorovi su zadati rednim brojevima) ispisuje da li je jedan čvor dostupan iz drugog. Graf se predstavlja kao u zadatku 3.

6. Čvor s usmerenog grafa G = (V,E) sa n čvorova se naziva ponor ako je ulazni stepen čvora s jednak n-1 i njegov izlazni stepen jednak 0. Drugim rečima, za svaki čvor x iz skupa V različit od s postoji grana (x, s) i ne postoji grana (s, x). Precizno opišite algoritam vremenske složenosti $O(n)$ koji određuje da li usmereni graf G predstavljen matricom susedstva sadrži čvor koji je ponor.

Rešenje:

Primetimo da ako postoji grana između čvorova v i u, tada v ne može da bude ponor, a ako grana ne postoji tada u ne može da bude ponor. Ako je A matrica susedstva tada važi:

1. Ako je A[v][u] = 1 tada čvor v nije ponor.
2. Ako je A[v][u] = 0 tada čvor u nije ponor.

Pretraživanjem matrice susedstva na linearan način korišćenjem prethodne činjenice možemo da eliminišemo po jedan čvor.

Kada sa n-1 pitanja eliminišemo n-1 čvorova, ostaće samo čvor kandidat. Za njega je potrebno utvrditi da li je zaista ponor, npr. kao u sledećem algoritmu.

```
Algoritam Ponor(A, n)
ulaz: A, n /* A je matrica susedstva grafa, n je broj cvorova u grafu*/
izlaz: ponor ili -1 ako ponor ne postoji
{
  i = 0;
  j = 1;
  next = 1;

  if (n == 1){
    if (A[0][0] = 0)
       return 0;
    else
       return -1;
  }
```

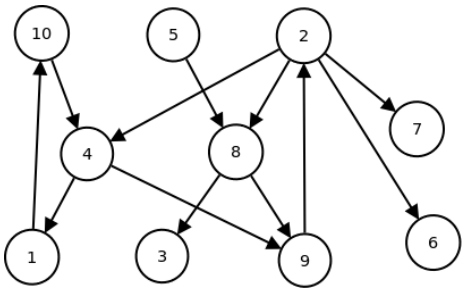
```
  do
  {
    next = next + 1;
    if (A[i][j] == 1)
    {
      kandidat = j; // cvor i nije ponor
      i = next;
    }
    else
    {
      kandidat = i; // cvor j nije ponor
      j = next;
    }
  } while (next < n);

  k = 0;
  while (k < n)
  {
    if (A[kandidat][k] ==1)
      return -1; // nema ponora
    if ((A[k][kandidat] == 0) && (kandidat!= k))
      return -1; // nema ponora
    k = k + 1;
  }

  return kandidat;
}
```

Obrazložiti zašto je vremenska složenost algoritma $O(n)$.

1.1 Obilazak grafa u širinu (BFS - Breadth-first search)

Obilazak u širinu počinje iz proizvoljnog zadatog čvora r koji se označava kao posećen i dodaje kao jedini element reda. Potom se ponavljaju sledeći koraci, dok god red ne postane prazan: obriši čvor sa početka reda i svakog neposećenog komšiju ovog čvora označiti kao posećenog i dodati ga na kraj reda.

7. Navesti čvorove u redosledu kojim će biti posećeni BFS obilaskom (obilaskom u dubinu) iz čvora 5. Od narednih čvorova koje je moguće posetiti bira se onaj koji ima najmanju vrednost.



Rešenje (brojevi prikazani po nivoima obilaska):

```
5
8
3 9
2
4 6 7
1
10
```

8. Implementirati BFS algoritam za obilazak grafa G=(V,E) koji je zadat matricom povezanosti tako da ispise čvorove u redosledu obilaska.

Rešenje:

IDEJA: Kod BFS pretrage kada se stigne do nekog čvora x grafa G, najpre se obilaze njegovi neposećeni i neposredni susedi. Nakon toga se nastavlja BFS algoritam, tj. posećuju se svi neposećeni susedi iz prethodnog nivoa pretrage. Zbog toga se koristi FIFO (First InFirst Out) red.

SKICA rešenja:

1. Polazni čvor x grafa G
   (a) smesti se u red
   (b) poseti se
   (c) markira kao posećen
   (d) upišu se susedi čvora x na kraj reda
2. Poseti se sledeći čvor iz reda, markira kao posećen, njegovi neposećeni susedi se upišu na kraj reda
3. Ponavlja se korak 2 dok ima neposećenih čvorova u redu

```
/*a = matrica susedstva, n=broj cvorova grafa G */
void poseti_sve(int a[][50], int n)
{
  int x;      /* cvor grafa */
  int m[50];  /* m je niz markiranih cvorova grafa */

  /* inicijalizacija niza markiranih cvorova na 0, jer su na pocetku svi neposeceni */
  for (x=0; x<n; x++)
     m[x]=0;

  /* poseta grafa pocev od prvog neposecenog cvora */
  for (x=0 ;x<n ;x++ )
     if (!m[x])
       BFS (x, a, n, m)
}

/*s = polazni cvor pretrage po sirini, a = matrica susedstva,
  n = broj cvorova grafa, m = niz posecenih cvorova*/
void BFS(int s, int a[][50], int n, int m[])
{
  int i,k; /*brojaci u ciklusu /
  int v; /* cvor grafa */
  int red [50]; /* niz cvorova grafa poredjanih u poretku BFS pretrage */

  /*inicijalizacije niza m, red u odnosu na polazni cvor pretrage s*/
  m[s]=1;
  red[0]=s;
  k=1;

  /*obilazak neposrednih suseda cvora s, razlika od DFSa*/
  for(i=0;i<k;i++)
  {
     s=red[i];
     for(v=0; v<n; v++)
       if( !m[v] && a[s][v])
       {
          m[v]=1;
          red[k++]=v;
       }
  }

  /*ispis BFS poretka*/
  for(i=0;i<k;i++)
     printf( %d , red[i]);
}
```

Složenost BFS algoritma ista je kao i složenost DFS algoritma, tj. u slučaju korišćenja matrica je $O(\mid V \mid^2)$ a u slučaju korišćenja lista povezanosti $O(\mid V \mid + \mid E \mid)$.

9. Implementirati algoritam povezan(a,n) kojim se u grafu G=(V,E) sa n čvorova (gradova) proverava da li su svih n čvorova povezani, tj. da li za ma koja dva grada postoji put koji ih povezuje. Pretpostaviti da su veze između gradova zadate simetričnom matricom a dimenzije n*n takvom da a[i][j]=1, ako postoji dvosmerna grana od čvora i do čvora j, a[i][j]=0, ako ne postoji grana od čvora i do čvora j.

10. Implementirati algoritam postojiPut(g, a, n) kojim se u grafu G=(V,E) sa n čvorova ispisuje na standardni izlaz do kojih se čvorova može doći polazeći od zadatog čvora g. Pretpostaviti da su veze između gradova zadate simetričnom matricom a dimenzije n*n takvom da a[i][j]=1, ako postoji dvosmerna grana od čvora i do čvora j, a[i][j]=0, ako ne postoji grana od čvora i do čvora j.