

Algoritmi i strukture podataka

vežbe 6

Mirko Stojadinović

12. novembar 2015

1

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.

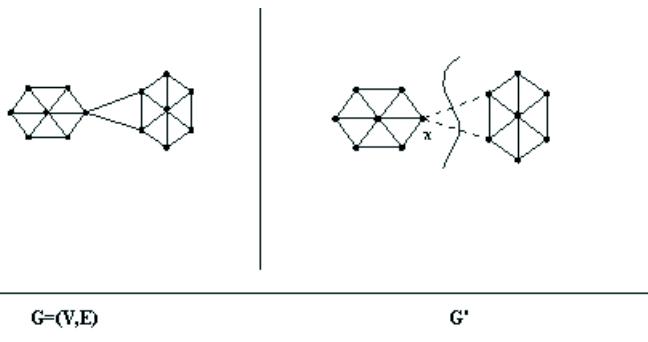
Rešenje: Zadatak prestavlja lemu iz profesorove knjige, pogledati knjigu za rešenje.

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

¹Materijali velikim delom preuzeti od Jelene Hadži-Purić: www.math.rs/~jelenagr

```

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

    za svaku granu (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 $|V|$.

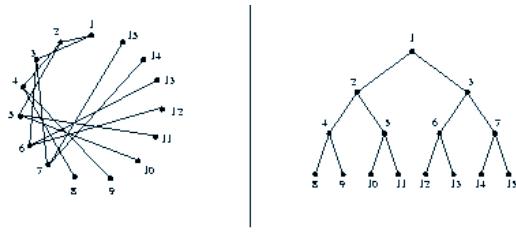
Napomena. Prethodni kod samo označava čvorove koje je moguće ukloniti, a ne uklanja te čvorove.

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 može 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 svaku granu (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. Čvorove treba uneti u obrnutom redosledu, pošto se smještaju na početak liste.

Rešenje:Naredni kod u .c fajlu

```

#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 suseda */

/* Ubacivanje na pocetak liste */
cvor_liste* ubaci_u_listu (cvor_liste* lista, int broj)
{
    cvor_liste* novi=malloc (sizeof(cvor_liste));
    if (novi == NULL){
        fprintf (stderr, "Neuspela alokacija.\n");
        exit (EXIT_FAILURE);
    }
    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, int posecen[])
{
    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, posecen);
}

int main()
{
    int i, j, br_suseda, sused, broj_cvorova;
    int posecen[MAX_BROJ_CVOROVA];

    printf ("Unesi broj cvorova grafa : ");
    scanf("%d",&broj_cvorova);

    for (i=0; i<broj_cvorova; i++)
        posecen[i] = 0;

    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 %.suseda cvora %. : ",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);
        }
    }
    printf ("\n");
    for (i=0; i<broj_cvorova; i++)
    {
        printf("%d - ",i);
        ispisi_listu(graf[i]);
        printf("\n");
    }
    DFS(0, posecen);
    return 0;
}

```

Složenost DFS algoritma koji koristi liste povezanosti je $O(|V| + |E|)$. 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 čvorova 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;
    }

    /* Prvi deo algoritma: nalazenje kandidata. */
    do
    {
        next = next + 1;
        if (A[i][j] == 1)
        {
            kandidat = j; // čvor i nije ponor
            i = next;
        }
        else
        {
            kandidat = i; // čvor j nije ponor
            j = next;
        }
    } while (next < n);

    /* Drugi deo algoritma: provera da li je kandidat ponor. */
    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;
}

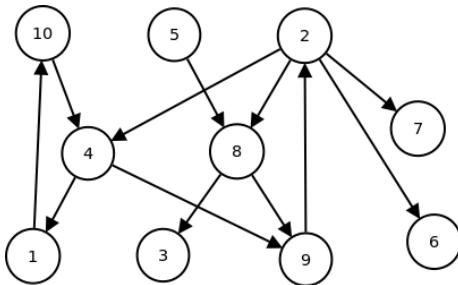
```

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 označi kao posećenog i dodati ga na kraj reda.

7. Navesti čvorove u redosledu kojim će biti posećeni BFS obilaskom (obilaskom u širinu) 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

```

Rešiti prethodni zadatak sa razlikom da se u obilazak kreće iz čvora 2.

```

2
4 6 7 8
1 9 3
10

```

Čvor 5 ostaje neposećen u obilasku iz čvora 2, da bi bio posećen mora se posebno pokrenuti BFS iz 5.

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

IDEJA: Kod BFS pretrage kada se stigne do nekog čvora v u grafu 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 v u grafa G
 - (a) smesti se u red
 - (b) poseti se
 - (c) markira kao posećen
 - (d) upišu se susedi čvora v na kraj reda i označe kao posećeni
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

Rešenje:Naredni kod u .c fajlu

```
/*graf = matrica susedstva, n=broj cvorova grafa */
void poseti_sve(int graf[][] [MAX], int n)
{
    int v; /* cvor grafa */
    int posecen[MAX]; /* posecen je niz markiranih cvorova grafa */

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

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

/*v = polazni cvor pretrage po sirini, graf = matrica susedstva,
 n = broj cvorova grafa, posecen = niz posecenih cvorova*/
void BFS(int v, int graf[][] [MAX], int n, int posecen[])
{
    int p,k; /*brojac u ciklusu */
    int w; /* cvor grafa */
    int red [MAX]; /* niz cvorova grafa poredjanih u poretku BFS pretrage */

    /*inicijalizacije niza posecen, u red se smesta polazni cvor
     pretrage v*/
    posecen[v]=1;
    red[0]=v;
    k=1;

    /*obilazak neposrednih suseda cvora v, razlika od DFSa.
     p označava pocetak reda, k kraj reda. */
    for(p=0; p<k; p++)
    {
        v=red[p];
        for(w=0; w<n; w++)
            if(!posecen[w] && graf[v][w])
            {
                posecen[w]=1;
                red[k++]=w;
            }
    }

    /*ispis BFS poretku*/
    for(p=0; p<k; p++)
        printf(" %d ", red[p]);
    printf ("\n");
}
```

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

9. Implementirati algoritam povezan(a,n) kojim se u grafu $G=(V,E)$ sa n cvorova (gradova) proverava da li su svih n cvorova povezani, tj. da li za svaki koja dva grada postoji put koji ih povezuje. Prepostaviti da su veze između gradova zadate simetričnom matricom a dimenzije $n \times n$ takvom da $a[i][j]=1$, ako postoji dvostruka 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 sime-tričnom 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 .