

Dvočas XIII – Rešenja

Milan Banković
Matematički Fakultet

June 10, 2007

Sadržaj

1	Grafovi	2
1.1	Uvod	2
1.2	DFS numeracija	2
1.3	Acikličnost usmerenog grafa	2
1.4	DFS stablo	3
1.5	Topološko sortiranje	3
1.6	BFS numeracija	3
1.7	BFS stablo	4
2	Grafovi – Rešenja	5

1 Grafovi

1.1 Uvod

Ovaj dvočas biće posvećen osnovnim tehnikama za rad sa grafovima. Tu se pre svega misli na algoritme obilaska grafa kao i razne primene ovih algoritama. Biće spomenuti i neki važni pojmovi kao što su *topološko sortiranje*, *acikličnost*, *BFS* i *DFS stablo* itd.

Kako je akcenat na algoritmima a ne na implementaciji, koristićemo najjednostavniji mogući oblik predstavljanja grafova u računarima – *matricu povezanosti*. Ova matrica ima elemente 1 ili 0 u zavisnosti od toga da li između odgovarajućih čvorova postoji grana ili ne. Takođe ćemo pretpostaviti da postoji neko razumno ograničenje po pitanju broja čvorova tako da ćemo moći da statički alociramo kvadratnu matricu (bez korišćenja `malloc()` i `free()`) Takođe se zbog jednostavnijeg kodiranja ovoga puta ohrabruje korišćenje globalnih promenljivih (što inače nije baš dobra praksa u programiranju).

1.2 DFS numeracija

Napisati funkciju koja demonstrira obilazak grafa metodom pretrage u dubinu, i pri tom vrši *dolaznu* i *odlaznu* numeraciju čvorova. Dolazna numeracija je u skladu sa redosledom posete čvorova. Odlazna numeracija je u skladu sa redosledom završetaka rekurzivnih poziva za odgovarajuće čvorove. Funkcija treba da ima sledeći prototip:

```
void DFS_numeracija(int polazni_cvor);
```

tj. treba da prihvata samo redni broj čvora od koga počinje obilazak. Obrada se vrši nad globalnim podacima, koje treba propisno inicijalizovati pre korišćenja funkcije. Nakon toga napisati i `main()` program koji testira ovu funkciju. Prilikom testiranja treba imati u vidu da se kod usmerenih grafova možda ne mogu dostići svi čvorovi iz jednog polaznog čvora, čak i da je graf povezan, te da se možda algoritam mora pozivati više puta iz različitih polaznih čvorova.

1.3 Acikličnost usmerenog grafa

Za usmeren graf kažemo da je *acikličan* ako u njemu ne postoji usmeren ciklus. Napisati funkciju koja ispituje da li je graf acikličan. Funkcija treba da ima prototip:

```
int DFS_aciklican(int polazni_cvor);
```

i da vrati jedan ako jeste acikličan, a nulu u suprotnom. Funkcija treba da koristi *DFS* obilazak, pa zato ima sličan prototip kao i prethodna funkcija.

1.4 DFS stablo

Napisati funkciju koja na osnovu datog grafa, polazeći iz datog čvora kreira novi graf koji sadrži samo one grane polaznog grafa koje se koriste prilikom *DFS* obilaska za prelazak na nove neposećene susede tekućeg čvora. Ovakav graf inače nema cikluse (ni usmerene, ni neusmerene) i naziva se *DFS stablo*. Uopšte, graf bez ciklusa se naziva *šuma*, a ako je pri tom još i povezan, tada ga zovemo *stablom* (za graf kažemo da je povezan ako u njegovoj neusmerenoj varijanti iz svakog čvora možemo doći do svakog drugog čvora). *DFS* stablo je dakle stablo koje se generiše pretragom u dubinu. Funkcija treba da ima prototip:

```
void DFS_stablo(int polazni_cvor);
```

Rezultat svog rada funkcija treba da sačuva u odgovarajućim globalnim podacima.

1.5 Topološko sortiranje

Zadatak topološkog sortiranja je da se čvorovi acikličnog usmerenog grafa postave u niz tako da važi sledeće: za svaka dva čvora u i v , ako postoji put od u do v u grafu, tada se u nalazi levo od v u nizu (tj. za svaki čvor znamo da su svi čvorovi koji su dostizni iz njega obavezno desno od njega u poretku). Ovakav poredak naravno nije jednoznačan. Nalaženje topološkog poretka u grafu ima mnogo praktičnih primena (npr. nalaženje redosleda obavljanja poslova, kada se zna da neki poslovi moraju biti obavljeni pre nekih drugih).

Napisati funkciju koja za dati aciklični graf daje jedan od mogućih poredaka. Funkcija treba da ima prototip:

```
void topolosko_sortiranje();
```

Funkcija barata sa globalnim podacima. Napisati i `main()` program koji testira napisanu funkciju.

1.6 BFS numeracija

Napisati funkciju koja obilazi graf pretragom u širinu i pri tom vrši numeraciju čvorova grafa. Primetimo da kod pretrage u širinu nema rekurzije, pa samim tim ni odlazne numeracije. Funkcija treba da ima prototip:

```
void BFS_numeracija(int polazni_cvor);
```

gde je `polazni_cvor` čvor iz koga kreće pretraga. Kao i kod pretrage u dubinu i ovde treba imati u vidu da se kod usmerenog grafa možda ne može iz svakog čvora doći do svakog drugog, te da će možda morati da se isti algoritam poziva više puta, iz raznih polaznih čvorova, kako bismo obeležili sve čvorove grafa. Funkcija barata sa globalnim podacima.

1.7 BFS stablo

Napisati funkciju koja na osnovu datog grafa, polazeći od datog čvora, kreira novi graf koji sadrži samo one grane polaznog grafa koje se koriste prilikom BFS obilaska grafa iz datog čvora za prelazak na obradu novih neposećenih čvorova. Ovakav graf je stablo, i nazivamo ga *BFS stablo*. Ono ima jednu lepu osobinu: put od korena stabla do proizvoljnog čvora u stablu je najkraći put (u smislu broja grana) između ta dva čvora u polaznom grafu. Ovo tvrdjenje naravno nećemo dokazivati. Funkcija treba da ima prototip:

```
void BFS_stablo(int polazni_cvor);
```

Funkcija barata sa globalnim podacima. Napisati i funkciju `main()` za testiranje date funkcije.

2 Grafovi – Rešenja

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

#define MAX_CVOROVA 100

/* Graf cemo, jednostavnosti radi, predstavljati staticki
   alociranom matricom povezanosti. Pretpostavka je da broj
   cvorova grafa nece biti veca od MAX_CVOROVA */
int graf[MAX_CVOROVA][MAX_CVOROVA];
int broj_cvorova;

/* Niz 0/1 vrednosti kojim se obelezavaju poseceni cvorovi */
int posecen[MAX_CVOROVA];

/* Pomocna funkcija koja utvrđuje da li postoji neposecen cvor.
   Funkcija vraca indeks prvog takvog cvora, ili -1 u slucaju da
   su svi cvorovi vec poseceni */
int postoji_neposecen()
{
    int i;

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

    return -1;
}

/* Pomocna funkcija koja služi za inicijalizaciju raznih nizova
   (poput posecen[], dolazna_numeracija[] itd.) Za sve nizove
   se pretpostavlja da su duzine broj_cvorova. */
void inicijalizuj(int niz[])
{
    int i;

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

```

}

/* Staticki podaci koji se koriste za smestanje rezultata
   prilikom numeracije cvorova (i za DFS i za BFS). */
int dolazna_numeracija[MAX_CVOROVA];
int odlazna_numeracija[MAX_CVOROVA];

/* Donji brojac treba da se postave na nulu pre pozivanja
   funkcija za numeraciju */
int brojac_dolazna=0;
int brojac_odlazna=0;

/* Funkcija obilazi graf DFS algoritmom, i tom prilikom
   vrsi ulaznu i izlaznu numeraciju cvorova. Pre poziva
   ove funkcije obavezno postaviti brojace dolazne i
   odlazne numeracije na 0 */
void DFS_numeracija(int polazni_cvor)
{
    int i;

    /* Obelezavanje */
    posecen[polazni_cvor] = 1;

    /* Ulazna obrada */
    dolazna_numeracija[polazni_cvor] = ++brojac_dolazna;

    /* Rekurzija (za sve cvorove ka kojima postoji grana
       i koji jos nisu obelezeni) */
    for(i = 0 ; i < broj_cvorova; i++)
        if(graf[polazni_cvor][i] && !posecen[i])
            DFS_numeracija(i);

    /* Izlazna obrada */
    odlazna_numeracija[polazni_cvor] = ++brojac_odlazna;
}

/* Staticki niz koji predstavlja indikator da li je
   odgovarajuci cvor na putu od korena do tekuceg cvora */
int na_putu[MAX_CVOROVA];

```

```

/* Funkcija proverava da li u grafu postoji usmeren
   ciklus. Ako ne postoji tada funkcija vraca 1 (tada
   kazemo da je graf aciklican). U suprotnom vraca 0.
   Pre koriscenja ove funkcije obavezno treba inicijalizovati
   posecen[], i na_putu[]. Funkcija koristi DFS obilazak */
int DFS_aciklican(int polazni_cvor)
{
    int i;

    /* Obelezavanje */
    posecen[polazni_cvor] = 1;

    /* Ulazna obrada */
    na_putu[polazni_cvor] = 1;

    /* Rekurzija */
    for(i = 0 ; i < broj_cvorova; i++)
        if(graf[polazni_cvor][i])
        {
            /* Ako je cvor neposecen posetiti ga */
            if(posecen[i] == 0)
            {
                if(DFS_aciklican(i) == 0)
                    return 0;
            }
            /* Ako je posecen, i na putu tada postoji ciklus */
            else if(na_putu[i])
                return 0;
        }

    /* Izlazna obrada */
    na_putu[polazni_cvor] = 0;

    return 1;
}

/* Graf koji ce predstavljati stablo koje se generise
   obilaskom grafa */

```



```

int stablo[MAX_CVOROVA][MAX_CVOROVA];

/* Kreiramo DFS stablo. Prethodno treba inicijalizovati
   posecen. */
void DFS_stablo(int polazni_cvor)
{
    int i;

    /* Obelezavanje */
    posecen[polazni_cvor] = 1;

    /* Rekurzija */
    for(i = 0 ; i < broj_cvorova; i++)
        if(graf[polazni_cvor][i] && !posecen[i])
        {
            /* Obelezavamo odgovarajucu granu */
            stablo[polazni_cvor][i] = 1;
            DFS_stablo(i);
        }
        else stablo[polazni_cvor][i] = 0;
}

/* Niz sadrzi ulazne stepene cvorova grafa */
int ulazni_stepen[MAX_CVOROVA];

/* Funkcija racuna ulazne stepene cvorova grafa,
   koristeci DFS pretragu. Prethodno se mora
   inicijalizovati ulazni_stepen[], kao i posecen[] */
void izracunaj_ulazne_stepene(int polazni_cvor)
{
    int i;

    /* Obelezavanje */
    posecen[polazni_cvor] = 1;

    /* Rekurzija */
    for(i = 0 ; i < broj_cvorova; i++)
        if(graf[polazni_cvor][i])
        {
            if(!posecen[i])

```

```

        izracunaj_ulazne_stepene(i);
    /* Izlazna obrada */
    ulazni_stepen[i]++;
}

}

/* Staticki niz u koji smestamo rezultat topoloskog sortiranja */
int poredak[MAX_CVOROVA];

/* Funkcija vrši topolosko sortiranje datog acikličnog grafa. */
void topolosko_sortiranje()
{
    int i,k;

    /* Improvizovani stek na koji stavljamo svaki novi cvor čiji
       ulazni stepen u toku postupka postane nula. Velicina steka
       je MAX_CVOROVA, što će biti dovoljno s obzirom na to da se
       na stek stavljaju cvorovi kojih ima najviše toliko */
    int stek[MAX_CVOROVA];
    int stek_pok = -1; /* Pokazivac na vrh steka */

    /* Inicijalizujemo posecen[] i ulazni_stepen[] */
    inicijalizuj(posecen);
    inicijalizuj(ulazni_stepen);

    /* Pozivamo funkciju koja izracunava ulazne stepene */
    while((i = postoji_neposecen()) >= 0)
        izracunaj_ulazne_stepene(i);

    /* Stavljamo na stek sve elemente čiji je ulazni stepen nula */
    for(i = 0; i < broj_cvorova; i++)
        if(ulazni_stepen[i] == 0)
            stek[++stek_pok] = i;

    /* Dokle god ima elemenata na steku, uzimamo element sa vrha
       i postavljamo ga za naredni element u poretku, a zatim
       umanjujemo ulazne stepene svim cvorovima ka kojima postoji
       grana iz tog cvora. */
    k = -1;

```

```

while(stek_pok >= 0)
{
/* Skidamo element sa steka i stavljamo ga u poredak */
poredak[++k] = stek[(stek_pok--)];

/* Umanjujemo za jedan ulazne stepene svih cvorova
   ka kojima postoji grana iz datog cvora. Ukoliko
   se nakon tog umanjjenja stepen nekog cvora svede
   na nulu, tada se isti dodaje na stek. */
for(i = 0; i < broj_cvorova; i++)
    if(graf[poredak[k]] [i])
        if(--ulazni_stepen[i] == 0)
            stek[++stek_pok] = i;
}

}

/* Funkcija vrši BFS obilazak i numerise cvorove u poretku
   obilaska. Pre poziva ove funkcije treba inicijalizovati
   posecen[] i brojac_dolazna */
void BFS_numeracija(int polazni_cvor)
{
/* Improvizovani red koji koristimo za smestanje
   cvorova koji cekaju da budu obradjeni. Cvorovi
   se smestaju na kraj reda (kraj niza), a uzimaju
   sa pocetka reda (niza). Zato imamo dva indeksa
   koji pokazuju na pocetak i kraj reda. Duzina
   ovog reda je MAX_CVOROVA sto je dovoljno jer
   ce svaki cvor na red biti postavljen tacno jednom */
int red[MAX_CVOROVA];
int sledeci_na_redu = 0;
int poslednji_na_redu = -1;
int i;

/* Inicijalno se na redu nalazi samo polazni cvor */
posecen[polazni_cvor] = 1;
red[++poslednji_na_redu] = polazni_cvor;

/* Dokle god imamo cvorove u redu...*/

```

```

while(sledeci_na_redu <= poslednji_na_redu)
{
    /* Vrsimo obradu cvora (numeracija) */
    dolazna_numeracija[red[sledeci_na_redu]] = ++brojac_dolazna;

    /* Za sve cvorove ka kojima postoji grana i koji nisu
       poseceni vrsimo obelezavanje i dodavanje u red */
    for(i = 0; i < broj_cvorova; i++)
        if(graf[red[sledeci_na_redu]][i] && !posecen[i])
        {
            red[++poslednji_na_redu] = i;
            posecen[i] = 1;
        }
    /* Prelazimo na sledeci u redu */
    sledeci_na_redu++;
}

}

/* Funkcija obilazi stablo BFS algoritmom i kreira BFS stablo.
   Funkcija ocekuje da se pre toga inicijalizuj posecen[]. Ova
   funkcija je potpuno analogna prethodnoj s tim sto je ovde
   obrada drugacija: postavljaju se grane prema novoposecenim
   cvorovima, a "iskljucuju" se grane u suprotnom */
void BFS_stablo(int polazni_cvor)
{
    int red[MAX_CVOROVA];
    int sledeci_na_redu = 0;
    int poslednji_na_redu = -1;
    int i;

    posecen[polazni_cvor] = 1;
    red[++poslednji_na_redu] = polazni_cvor;

    while(sledeci_na_redu <= poslednji_na_redu)
    {
        for(i = 0; i < broj_cvorova; i++)
            if(graf[red[sledeci_na_redu]][i] && !posecen[i])
            {
                /* Uz obelezavanje i stavljanje na red vrsi se

```

```

        i ukljucivanje odgovarajuce grane u stablo */
        stablo[red[sledeci_na_redu]][i] = 1;
        red[++poslednji_na_redu] = i;
        posecen[i] = 1;
    }
    else /* Iskljucuje se grana u suprotnom */
        stablo[red[sledeci_na_redu]][i] = 0;

    sledeci_na_redu++;
}

}

/* Test program */
int main()
{
    int i,j;
    int aciklican;

    /* Unos matrice povezanosti grafa */
    printf("Unesite broj cvorova grafa (<=%d): ", MAX_CVOROVA);
    scanf("%d",&broj_cvorova);

    for(i = 0; i < broj_cvorova; i++)
        for(j = 0; j < broj_cvorova; j++)
        {
            if(i == j) continue;

            printf("Da li postoji grana (%d,%d) (0/1): ", i, j);
            scanf("%d", &graf[i][j]);
        }

    /* Testiramo DFS numeraciju */
    inicijalizuj(posecen);

    brojac_dolazna = 0;
    brojac_odlazna = 0;

    while((i = postoji_neposecen()) >= 0)
        DFS_numeracija(i);

```

```

printf("Dolazna DFS numeracija: ");
for(i = 0; i < broj_cvorova; i++)
    printf("%d ",dolazna_numeracija[i]);

putchar('\n');
printf("Odlazna DFS numeracija: ");
for(i = 0; i < broj_cvorova; i++)
    printf("%d ", odlazna_numeracija[i]);
putchar('\n');

/* Testiramo aciklicnost grafa */
inicijalizuj(posecen);
inicijalizuj(na_putu);

while((i = postoji_neposecen()) >= 0)
    aciklican = DFS_aciklican(i);

if(aciklican)
    printf("U grafu ne postoji usmeren ciklus\n");
else
    printf("U grafu postoji usmeren ciklus\n");

/* Testiramo DFS stablo */
inicijalizuj(posecen);
while((i = postoji_neposecen()) >= 0)
    DFS_stablo(i);

printf("DFS stablo (matrica povezanosti):\n");
for(i = 0; i < broj_cvorova; i++)
{
    for(j = 0; j < broj_cvorova; j++)
        printf("%d",stablo[i][j]);
    printf("\n");
}

/* Testiramo topolosko sortiranje. Ovo je moguće
samo ako je graf aciklican */
if(aciklican)
{

```

```

topolosko_sortiranje();
printf("Topoloski poredak: ");
for(i = 0 ; i < broj_cvorova ; i++)
    printf("%d ",poredak[i]);

putchar('\n');

}

/* Testiramo BFS numeraciju */
inicijalizuj(posecen);
brojac_dolazna = 0;
while((i = postoji_neposecen()) >= 0)
    BFS_numeracija(i);

printf("Dolazna BFS numeracija: ");
for(i = 0; i < broj_cvorova; i++)
    printf("%d ",dolazna_numeracija[i]);
putchar('\n');

/* Testiramo BFS stablo */
inicijalizuj(posecen);
while((i = postoji_neposecen()) >= 0)
    BFS_stablo(i);

printf("BFS stablo (matrica povezanosti):\n");
for(i = 0; i < broj_cvorova; i++)
{
    for(j = 0; j < broj_cvorova; j++)
        printf("%d",stablo[i][j]);
    printf("\n");
}

return EXIT_SUCCESS;
}

```