Osnovi programiranja

Beleške sa vežbi

Školska 2005/2006 godine

Smer *Računarstvo i informatika*

Matematički fakultet, Beograd

Jelena Tomašević

May 17, 2006

Sadržaj

1 Programski jezik C **5**
1.1 Grafovi 5

1

Programski jezik C

[1]

1.1 Grafovi

Graf `G=(V,E)` sastoji se od skupa `V` čvorova i skupa `E` grana. Grane predstavljaju relacije između čvorova i odgovara paru čvorova. Graf može biti usmeren (orijentisan), ako su mu grane uređeni parovi i neusmeren (neorjentisan) ako su grane neuređeni parovi.

Uobičajena su dva načina predstavljanja grafova. To su *matrica povezanosti* grafa i *lista povezanosti*.

Matrica povezanosti je kvadratna matrica dimenzije `n`, pri čemu je `n` broj čvorova u grafu, takva da je element na preseku `i`-te vrste i `j`-te kolone jednak jedinici ukoliko postoji grana u grafu od `i`-tog do `j`-tog čvora, inače je nula.

Umesto da se i sve nepostojeće grane eksplicitno predstavljaju u matrici povezanosti, mogu se formirati povezane liste od jedinica iz `i`-te vrste za `i=1,2,...,n`. To je lista povezanosti. Svakom čvoru se pridružuje povezana lista, koja sadrži sve grane susedne tom čvoru. Graf je predstavljen vektorom lista. Svaki elemenat vektora sadrži ime (indeks) čvora i pokazivač na njegovu listu čvorova.

Prvi problem na koji se nailazi pri konstrukciji bilo kog algoritma za obradu grafa je kako pregledati ulaz. Postoje dva osnovna algoritma za obilazak grafa: pretraga u dubinu (`DFS`, skraćenica od depth-first-search) i pretraga u širinu (`BFS`, skraćenica od breadth-first-search).

Kod `DFS` algoritma, obilazak započinje iz proizvoljnog zadatog čvora `r` koji se naziva *koren pretrage u dubinu*. Koren se označava kao posećen. Zatim se bira proizvoljan neoznačen čvor `r1`, sussedan sa `r`, pa se iz čvora `r1` rekurzivno startuje pretraga u dubinu. Iz nekog nivoa rekurzije izlazi se kad se naiđe na čvor v kome su svi susedi već označeni.

Primer 1 *Primer reprezentovanja grafa preko matrice povezanosti. U programu se unosi neorijentisan graf i DFS algoritmom se utvrdjuju čvrovi koji su dostižni iz cvora 0.*

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

int** alociraj_matricu(int n)
{   int **matrica;
    int i;
    matrica=malloc(n*sizeof(int*));
```

[1] Zasnovano na materijalu `Algoritmi, Miodrag Živković` i `http://www.matf.bg.ac.yu/~filip`

```
    for (i=0; i<n; i++)
       matrica[i]=calloc(n,sizeof(int));
    return matrica;
}

void oslobodi_matricu(int** matrica, int n)
{   int i;
    for (i=0; i<n; i++)
        free(matrica[i]);
    free(matrica);
}

int* alociraj_niz(int n)
{   int* niz;
    niz=calloc(n,sizeof(int));
    return niz;
}

void oslobodi_niz(int* niz)
{   free(niz);
}

void unesi_graf(int** graf, int n)
{     int i,j;
      for (i=0; i<n; i++)
         for (j=i; j<n; j++)
         {   printf("Da li su element %d i %d povezani : ",i,j);
             do
             {  scanf("%d",&graf[i][j]);
                graf[j][i]=graf[i][j];
             } while (graf[i][j]!=0 && graf[i][j]!=1);
         }
}

void ispisi_graf(int** graf, int n)
{     int i,j;
      for (i=0; i<n; i++)
      {  for (j=0; j<n; j++)
             printf("%d",graf[i][j]);
         printf("\n");
      }
}



/* Broj cvorova grafa (dimenzija matrice) */
int n;
/* Matrica povezanosti */
int **graf;


/* Pomocni vektor koji govori o tome koji su cvorovi posecivani
```

```
   tokom DFS obilaska */
int *posecen;

/* Rekurzivna implementacija DFS algoritma */
void poseti(int i)
{    int j;
     posecen[i]=1;
     printf("Posecujem cvor %d\n",i);
     for (j=0; j<n; j++)
        if (graf[i][j] && !posecen[j])
               poseti(j);
}


main()
{    int i, j;
     printf("Unesi broj cvorova : ");
     scanf("%d",&n);

     graf=alociraj_matricu(n);
     unesi_graf(graf,n);
     ispisi_graf(graf,n);

     posecen=alociraj_niz(n);
     poseti(0);

     oslobodi_niz(posecen);
     oslobodi_matricu(graf,n);

}
```

Primer 2 *Primer predstavljanja grafa preko niza listi suseda svakog od čvorova grafa U programu se unosi graf i DFS algoritmom se utvrdjuje koji su čvorovi dostižni iz cvora 0.*

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

/* Cvor liste suseda */
typedef struct _cvor_liste
{  int broj; /* Indeks suseda */
   struct _cvor_liste* sledeci;
} cvor_liste;

/* 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;
}


/* Brisanje liste */
```

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

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

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


/* Rekurzivna implementacija DFS algoritma */
int posecen[MAX_BROJ_CVOROVA];
void poseti(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])
              poseti(sused->broj);
}


main()
{   int i;
    printf("Unesi broj cvorova grafa : ");
    scanf("%d",&broj_cvorova);
    for (i=0; i<broj_cvorova; i++)
    {    int br_suseda,j;

         graf[i]=NULL;

         printf("Koliko cvor %d ima suseda : ",i);
         scanf("%d",&br_suseda);
         for (j=0; j<br_suseda; j++)
         {    int sused;
              do
              {
                printf("Unesi broj %d.-tog suseda cvora %d : ",j,i);
                scanf("%d",&sused);
              } while (sused<1 && sused>broj_cvorova);
              graf[i]=ubaci_u_listu(graf[i],sused-1);
```

```
        }
    }

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

    poseti(0);
}
```

Primer 3 *MINESWEEPER - primer jednostavne igrice. Program demonstrira rad sa matricama, slučajnim brojevima i rekurzivnu implementaciju DFS algoritma za obilazak grafova.*

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

/* Dimenzija table */
int   n;

/* Tabla koja sadrzi 0 i 1 u zavisnosti od toga da li na polju postoji bomba */
int** bombe;

/* Tabla koja opisuje tekuce stanje igre. Moze da sadrzi sledece vrednosti :
    ZATVORENO - opisuje polje koje jos nije bilo otvarano
    PRAZNO - polje na kome ne postoji ni jedna bomba
    broj od 1-8 - polje koje je otvoreno i na kome pise koliko bombi postoji u okolini
    ZASTAVICA - polje koje je korisnik oznacio zastavicom
*/
#define PRAZNO (-1)
#define ZATVORENO 0
#define ZASTAVICA 9

int** stanje;

/* Ukupan broj bombi */
int broj_bombi;

/* Ukupan broj postavljenih zastavica */
int broj_zastavica = 0;


/* Pomocne funkcije za rad sa matricama */
int** alociraj(int n)
{  int i;
   int** m=malloc(n*sizeof(int*));
   for (i=0; i<n; i++)
       m[i]=calloc(n,sizeof(int));
   return m;
}
```

```
void obrisi(int** m, int n)
{   int i;
    for (i=0; i<n; i++)
       free(m[i]);

    free(m);
}

/* Funkcija postavlja bombe */
void postavi_bombe()
{  broj_bombi=(n*n)/6;
   int kolona;
   int vrsta;
   int i;

   /* Inicijalizujemo generator slucajnih brojeva */
   srand(time(NULL));

   for (i=0; i<broj_bombi; i++)
   {    /* Racunamo slucajni polozaj bombe */
        kolona=rand()%n;
        vrsta=rand()%n;


        /* Ukoliko bomba vec postoji tu, opet idemo u istu iteraciju */
        if (bombe[vrsta][kolona]==1)
        {   i--;
            continue;
        }

        /* Postavljamo bombu */
        bombe[vrsta][kolona]=1;
  }
}


/* Funkcija ispisuje tablu sa bombama */
void ispisi_bombe()
{   int i,j;
    for (i=0; i<n; i++)
    {   for (j=0; j<n; j++)
            printf("%d",bombe[i][j]);
        printf("\n");
    }
}

/* Funkcija ispisuje tekuce stanje */
void ispisi_stanje()
{   int i,j;

    /* Brisemo ekran pozivajuci komandu operativnog sistema */
```

```
    system("clear");


    for (i=0; i<n; i++)
    {   for (j=0; j<n; j++)
        {    if (stanje[i][j]==ZATVORENO)
                  printf(".");
             else if (stanje[i][j]==PRAZNO)
                  printf(" ");
             else if (stanje[i][j]==ZASTAVICA)
                  printf("*");
             else
                  printf("%d",stanje[i][j]);
        }
        printf("\n");
    }

}

/* Funkcija postavlja zastavicu na dato polje ili je uklanja
   ukoliko vec postoji */
void postavi_zastavicu(int i, int j)
{    if (stanje[i][j]==ZATVORENO)
     {    stanje[i][j]=ZASTAVICA;
          broj_zastavica++;
     }
     else if (stanje[i][j]==ZASTAVICA)
     {    stanje[i][j]=ZATVORENO;
          broj_zastavica--;
     }
}

/* Funkcija izracunava koliko bombi postoji u okolini date bombe */
int broj_bombi_u_okolini(int v, int k)
{    int i, j;
     int br=0;
     /* Prolazimo kroz sva okolna polja */
     for (i=-1; i<=1; i++)
         for(j=-1; j<=1; j++)
         {  /* preskacemo centralno polje */
            if (i==0 && j==0)
                continue;
            /* preskacemo polja "van table" */
            if (v+i<0 || k+j<0 || v+i>=n || k+j>=n)
                continue;
            if (bombe[v+i][k+j]==1)
                br++;
         }

   return br;

}
```

```
/* Centralna funkcija koja vrsi otvaranje polja i pritom se otvaranje "siri"
   i na polja koja su oko datog */

void otvori_polje(int v, int k)
{   /* Ukoliko smo "nagazili" bombu zavrsavamo program */
    if (bombe[v][k]==1)
    {   printf("BOOOOOOOOOOOOOOOOM!!!!\n");
        ispisi_bombe();
        exit(1);
    }
    else
    {    /* Brojimo bombe u okolini */
         int br=broj_bombi_u_okolini(v,k);

         /* Azuriramo stanje ovog polja */
         stanje[v][k]=(br==0)?PRAZNO:br;

         /* Ukoliko u okolini nema bombi, rekurzivno otvaramo
            sva polja u okolini koja su zatvorena */
         if (br==0)
         {   /* Petlje indeksiraju sva okolna polja */
             int i,j;
             for (i=-1; i<=1; i++)
                for (j=-1; j<=1; j++)
                {   /* Preskacemo centralno polje */
                    if (i==0 && j==0)
                        continue;
                    /* Preskacemo polja van table */
                    if (v+i<0 || v+i>=n || k+j<0 || k+j>=n)
                          continue;
                    /* Ukoliko je okolno polje zatvoreno, otvaramo ga */
                    if (stanje[v+i][k+j]==ZATVORENO)
                          otvori_polje(v+i, k+j);
                }
         }
    }
}


/* Funkcija utrdjuje da li je partija gotova
   Partija je gotova u trenutku kada su sve bombe pokrivene zastavicama i
   kada nijedno drugo polje nije pokriveno zastavicom
*/

int gotova_partija()
{  int i,j;
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
      {   /* Ukoliko postoji nepokrivena bomba, partija nije zavrsena */
          if (bombe[i][j]==1 && stanje[i][j]!=ZASTAVICA)
              return 0;
```

```
      }

  /* Partija je zavrsena samo ukoliko je broj zastavica jednak broj bombi */
  return broj_zastavica==broj_bombi;

}

main()
{

  /* Unosimo dimenziju table */
  printf("Unesite dimenziju table : ");
  scanf("%d",&n);


  /* Alociramo table */
  bombe=alociraj(n);
  stanje=alociraj(n);

  /* Postavljamo bombe */
  postavi_bombe();

  /* Sve dok partija nije gotova */
  while(!gotova_partija())
  {    int v,k;
       char akcija;

       /* Ispisujemo tekuce stanje */
       ispisi_stanje();

       /* Sve dok korisnik ne unese o ili z trazimo od njega da upise odgovarajucu akciju */
       do
       {  printf("Unesi akciju (o - otvaranje polja, z - postavljanje zastavice) : ");
      while(isspace(akcija = getchar()));
       }  while (akcija!='o' && akcija!='z');

       /* Trazimo od korisnika da unese koordinate polja sve dok ih ne unese ispravno
          Korisnicke koordinate krecu od 1, a interne od 0 */
       do
       {
          printf("Unesi koordinate polja : ");
          scanf("%d",&v);
          scanf("%d",&k);
       } while(v<1 || v>n || k<1 || k>n);

       /* Reagujemo na akciju */
       switch(akcija)
       {   case 'o':
               otvori_polje(v-1,k-1);
               break;
           case 'z':
               postavi_zastavicu(v-1,k-1);
```

```
        }
    }

    /* Konstatujemo pobedu */
    ispisi_stanje();
    obrisi(stanje, n);
    obrisi(bombe, n);

    printf ("Cestitam! Pobedili ste\n");

}
```