Programiranje II

Beleške sa vežbi

Smer *Informatika*

Matematički fakultet, Beograd

Sana Stojanović

22.05.08.

Sadržaj

1 (Binarno) Pretraživačko drvo

1. Napisati program koji kreira uređeno binarno drvo (čiji se elementi dobijaju sa standardnog ulaza dok se ne unese nula kao oznaka za kraj). Napisati funkcije za kreiranje čvora drveta, ubacivanje elementa u uređeno stablo, ispisivanje stabla i oslobađanje stabla. Funkciju za ubacivanje elementa u stablo napisati tako da ima sledeću deklaraciju:

```
void ubaci_u_drvo(cvor** pdrvo, int b)
```

```
/* Binarno pretrazivacko drvo - drvo sadrzi cele brojeve.
   Ubacivanje realizovano preko dvostrukog pokazivaca */

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

/* Struktura jednog cvora drveta */
typedef struct _cvor
{
    /* Podatak */
    int broj;
    /* Pokazivac na levo i desno podstablo */
    struct _cvor *l, *d;
} cvor;


/* Pomocna funkcija koja kreira novi cvor na osnovu datog broja */
cvor* napravi_cvor(int b)
{
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije");
        exit(1);
    }

    /* Postavljamo brojnu vrednost */
    novi->broj = b;

    /* Novi cvor se kreira kao list */
    novi->l = NULL;
    novi->d = NULL;
    return novi;
}


/* Rekurzivna funkcija koja ubacuje dati broj u dato drvo (koje je
```

```
   dato preko pokazivaca da bi mu se mogla promeniti vrednost) */
void ubaci_u_drvo(cvor** pdrvo, int b)
{
    /* Ukoliko je drvo prazno kreira se novi cvor */
    if (*pdrvo == NULL)
    {
        *pdrvo = napravi_cvor(b);
        return;
    }

    /* Ukoliko je broj koji se ubacuje manji od broja u korenu,
       rekurzivno ga ubacujemo u levo podstablo.
       Ukoliko je broj koji se ubacuje veci od broja u korenu,
       rekurzivno ga ubacujemo u desno podstablo. */

    if (b < (*pdrvo)->broj)
        ubaci_u_drvo(&((*pdrvo)->l), b);
    else if (b > (*pdrvo)->broj)
        ubaci_u_drvo(&((*pdrvo)->d), b);
}


/* Rekurzivna funkcija koja ispisuje drvo u inorder redosledu */
void ispisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


/* Rekurzivna funkcija koja uklanja drvo. Obilazak mora biti
   postorder. */
void obrisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}
```

```
main()
{
    cvor* drvo = NULL;
    int br;

    while(1)
    {
        scanf("%d", &br);
        if (br == 0)
            break;
        ubaci_u_drvo(&drvo, br);
    }

    ispisi_drvo(drvo);
    putchar('\n');

    obrisi_drvo(drvo);
}
```

2. Napisati program koji formira uređeno binarno stablo na način opisan u prethodnom zadatku i nakon toga unosi ceo broj sa standardnog ulaza i proverava da li se taj broj nalazi u stablu pozivom funkcije:

```
int pronadji(cvor* drvo, int br)

/* Binarno pretrazivacko drvo - drvo sadrzi cele brojeve.
   Ubacivanje realizovano preko dvostrukog pokazivaca */

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

/* Struktura jednog cvora drveta */
typedef struct _cvor
{
    /* Podatak */
    int broj;
    /* Pokazivac na levo i desno podstablo */
    struct _cvor *l, *d;
} cvor;

/* Pomocna funkcija koja kreira novi cvor na osnovu datog broja */
cvor* napravi_cvor(int b)
{
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
```

```
    {
        fprintf(stderr, "Greska prilikom alokacije memorije");
        exit(1);
    }

    /* Postavljamo brojnu vrednost */
    novi->broj = b;

    /* Novi cvor se kreira kao list */
    novi->l = NULL;
    novi->d = NULL;
    return novi;
}

/* Rekurzivna funkcija koja ubacuje dati broj u dato drvo (koje je
   dato preko pokazivaca da bi mu se mogla promeniti vrednost) */
void ubaci_u_drvo(cvor** pdrvo, int b)
{
    /* Ukoliko je drvo prazno kreira se novi cvor */
    if (*pdrvo == NULL)
    {
        *pdrvo = napravi_cvor(b);
        return;
    }

    /* Ukoliko je broj koji se ubacuje manji od broja u korenu,
       rekurzivno ga ubacujemo u levo podstablo.
       Ukoliko je broj koji se ubacuje veci od broja u korenu,
       rekurzivno ga ubacujemo u desno podstablo. */

    if (b < (*pdrvo)->broj)
        ubaci_u_drvo(&((*pdrvo)->l), b);
    else if (b > (*pdrvo)->broj)
        ubaci_u_drvo(&((*pdrvo)->d), b);
}


/* Rekurzivna funkcija koja proverava da li dati broj postoji u
   drvetu */
int pronadji(cvor* drvo, int b)
{
    /* U praznom drvetu ne postoji broj */
    if (drvo == NULL)
        return 0;

    /* Ukoliko je jednak vrednosti u korenu, onda postoji */
```

```
    if (drvo->broj == b)
        return 1;

    /* Ukoliko je broj koji trazimo manji od vrednosti u korenu,
       trazimo ga samo u levom podstablu, a inace ga trazimo
       samo u desnom podstablu */
    if (b < drvo->broj)
        return pronadji(drvo->l, b);
    else
        return pronadji(drvo->d, b);
}

/* Rekurzivna funkcija koja ispisuje drvo u inorder redosledu */
void ispisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


/* Rekurzivna funkcija koja uklanja drvo. Obilazak mora biti
   postorder. */
void obrisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}

main()
{
    cvor* drvo = NULL;
    int br;

    while(1)
    {
        scanf("%d", &br);
        if (br == 0)
            break;
```

```
        ubaci_u_drvo(&drvo, br);
    }

    printf("Unesite ceo broj: ");
    scanf("%d", &br);
    if (pronadji(drvo, br))
        printf("Broj %d se nalazi u drvetu.\n", br);
    else
        printf("Broj %d se ne nalazi u drvetu.\n", br);

    ispisi_drvo(drvo);
    putchar('\n');

    obrisi_drvo(drvo);
}
```

3. Napisati program koji kreira uređeno binarno drvo. Napisati funkcije za kreiranje čvora drveta, ubacivanje elementa u uređeno stablo, ispisivanje stabla i oslobađanje stabla. Ubacivanje elementa u stablo realizovati preko funkcije koja ima sledeću deklaraciju:

```
cvor* ubaci_u_drvo(cvor* drvo, int b)

/* Binarno pretrazivacko drvo - drvo sadrzi cele brojeve.
   Ubacivanje realizovano preko eksplicitnog vracanja korena
   rezultujuceg drveta*/

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

typedef struct _cvor
{
    int broj;
    struct _cvor *l, *d;
} cvor;

cvor* napravi_cvor(int b)
{
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije");
        exit(1);
    }
    novi->broj = b;
```

```
    novi->l = NULL;
    novi->d = NULL;
    return novi;
}

/* Funkcija koja ubacuje element u drvo kao argument prima drvo i
   novi element a novo drvo vraca kao povratnu vrednost. */
cvor* ubaci_u_drvo(cvor* drvo, int b)
{
    if (drvo == NULL)
        return napravi_cvor(b);

    if (b < drvo->broj)
        drvo->l = ubaci_u_drvo(drvo->l, b);
    else
        drvo->d = ubaci_u_drvo(drvo->d, b);

    return drvo;
}


void ispisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


void obrisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}

main()
{
    cvor* drvo = NULL;
    int br;
```

```
    while(1)
    {
        scanf("%d", &br);
        if (br == 0)
            break;
        drvo = ubaci_u_drvo(drvo, br);
    }

    ispisi_drvo(drvo);
    putchar('\n');

    obrisi_drvo(drvo);
}
```

4. Napisati rekurzivne funkcije za rad sa celobrojnim stablima. Napisati funkcije za računanje broja čvorova drveta, broja listova drveta, sumu čvorova, sumu listova, ispisivanje listova, izračunavanje dubine i izračunavanje najvećeg čvora (u slučaju neuređenog stabla).

```
/* Rekurzivne funkcije za rad sa celobrojnim stablima (ne obavezno
   pretrazivackim):
   broj_cvorova, broj_listova, suma_cvorova, dubina, najveci_cvor, ... */

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

typedef struct _cvor
{
    int broj;
    struct _cvor *l, *d;
} cvor;

cvor* napravi_cvor(int b)
{
    cvor* novi = (cvor*)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "Greska prilikom alokacije memorije");
        exit(1);
    }
    novi->broj = b;
    novi->l = NULL;
    novi->d = NULL;
```

```
    return novi;
}

void ubaci_u_drvo(cvor** drvo, int b)
{
    if (*drvo == NULL)
    {
        *drvo = napravi_cvor(b);
        return;
    }

    if (b < (*drvo)->broj)
        ubaci_u_drvo(&((*drvo)->l), b);
    else
        ubaci_u_drvo(&((*drvo)->d), b);
}

void ispisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        ispisi_drvo(drvo->l);
        printf("%d ", drvo->broj);
        ispisi_drvo(drvo->d);
    }
}


void obrisi_drvo(cvor* drvo)
{
    if (drvo != NULL)
    {
        obrisi_drvo(drvo->l);
        obrisi_drvo(drvo->d);
        free(drvo);
    }
}

/* Izracunava broj cvorova datog drveta */
int broj_cvorova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    return  broj_cvorova(drvo->l) +
            1 +
            broj_cvorova(drvo->d);
```

```
}

/* Izracunava sumu svih elemenata u cvorovima drveta */
int suma_cvorova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    return  suma_cvorova(drvo->l) +
            drvo->broj +
            suma_cvorova(drvo->d);
}

/* Izracunava broj listova datog drveta */
int broj_listova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;

    /* Cvor je list ukoliko nema ni jednog naslednika */
    if (drvo->l == NULL && drvo->d == NULL)
        return 1;

    return  broj_listova(drvo->l) + broj_listova(drvo->d);
}

/* Izracunava sumu svih listova */
int suma_listova(cvor* drvo)
{
    if (drvo == NULL)
        return 0;

    if (drvo->l == NULL && drvo->d == NULL)
        return drvo->broj;

    return  suma_listova(drvo->l) + suma_listova(drvo->d);
}

/* Ispisuje sve elemente u listovima drveta */
void ispisi_listove(cvor* drvo)
{
    if (drvo == NULL)
        return;

    ispisi_listove(drvo->l);

    if (drvo->l == NULL && drvo->d == NULL)
```

```
        printf("%d ", drvo->broj);

    ispisi_listove(drvo->d);
}

/* Izracunava dubinu (broj nivoa drveta) */
int dubina(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    else
    {
        int dl = dubina(drvo->l);
        int dd = dubina(drvo->d);

        return dl<dd ? dd + 1 : dl + 1;
    }
}

/* Izracunava vrednost najveceg cvora u proizvoljnom drvetu.
   Vraca 0 ukoliko je drvo prazno */
int najveci_cvor(cvor* drvo)
{
    if (drvo == NULL)
        return 0;
    else
    {
        int max_l = najveci_cvor(drvo->l);
        int max_d = najveci_cvor(drvo->d);

        return max_l<max_d ?
            (max_d<drvo->broj?drvo->broj:max_d) :
            (max_l<drvo->broj?drvo->broj:max_l);
    }
}

main()
{
    cvor* drvo = NULL;
    int br;

    while(1)
    {
        scanf("%d", &br);
        if (br == 0)
            break;
```

```
        ubaci_u_drvo(&drvo, br);
    }

    printf("Suma cvorova : %d\n", suma_cvorova(drvo));
    printf("Broj cvorova : %d\n", broj_cvorova(drvo));
    printf("Broj listova : %d\n", broj_listova(drvo));
    printf("Suma listova : %d\n", suma_listova(drvo));
    printf("Najveci cvor : %d\n", najveci_cvor(drvo));
    printf("Dubina : %d\n", dubina(drvo));
    printf("Listovi : ");
    ispisi_listove(drvo);
    putchar('\n');

    obrisi_drvo(drvo);
}
```