Tema 10

Spregnuta lista pomoću rekurzije i binarno stablo

Neke od rekurzivnih funkcija za rad sa jednostruko spregnutom listom

Dodavanje čvora na kraj

```
void dodaj_na_kraj(CVOR **pglava, CVOR *novi) {
    if(*pglava == NULL) {
        *pglava = novi;
        return;
    }

    dodaj_na_kraj(&(*pglava)->sledeci, novi);
}
```

Rekurzivno listanje liste (ispis svih čvorova)

```
void ispisi_listu(CVOR *glava) {
    if(glava != NULL) {
        printf("%d ", glava->el);
        ispisi_listu(glava->sledeci);
    }
}
```

Rekurzivno listanje liste unazad (ispis svih čvorova unazad)

- Funkcija za koju je moguća samo rekurzivna implementacija u slučaju jednostruko spregnute liste

```
void ispisi_listu_unazad(CVOR *glava) {
    if(glava != NULL) {
        ispisi_listu_unazad(glava->sledeci);
        printf("%d ", glava->el);
    }
}
```

Brisanje cele liste

```
void obrisi_listu(CVOR **pglava) {
    if(*pglava != NULL) {
        obrisi_listu(&(*pglava)->sledeci);
        free(*pglava);
        *pglava = NULL;
    }
}
```

Zadatak 1

Napisati program koji učitane reči ispisuje unazad. Pojedinačna reč može imati najviše 20 karaktera i podrazumevano je da sadrži samo slova. Reči se učitavaju dokle god se ne upiše tačka (`.`) kao reč.

Primer `main` funkcije:

```
int main() {
    char rec[MAX_STRING];
    CVOR *glava;

    inicijalizacija(&glava);

    printf("Uneti reci (. oznacava kraj unosa):\n");
    do {
        scanf("%s", rec);
        dodaj_na_kraj(&glava, napravi_cvor(rec));
    } while(strcmp(rec, ".") != 0);

    printf("Ispis unetih reci: ");
    ispisi_reci(glava);
    printf("\n");
    printf("Ispis unetih reci unazad: ");
```

```
    ispisi_reci_unazad(glava);
    printf("\n");

    obrisi_listu(&glava);

    return EXIT_SUCCESS;
}
```

Primer rada programa:

```
Uneti reci (. oznacava kraj unosa):
Ana
voli
Milovana
.
Ispis unetih reci: Ana voli Milovana .
Ispis unetih reci unazad:  . anavoliM ilov anA
```

Binarno stablo

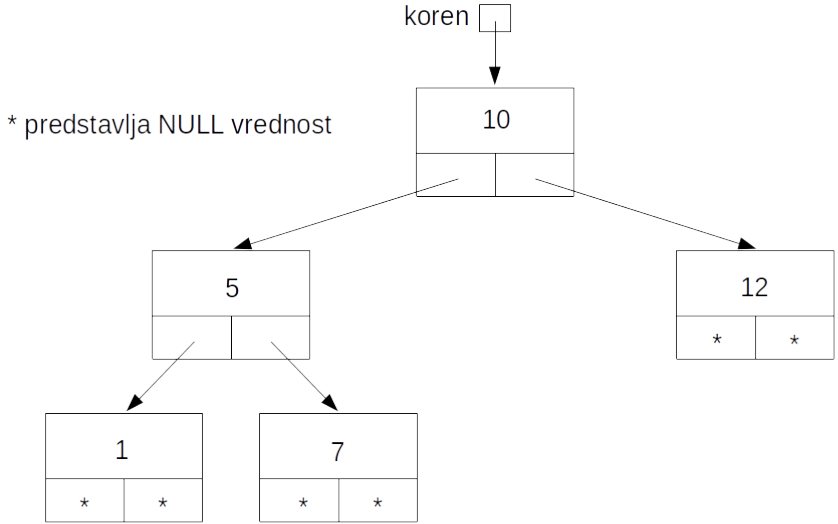
- Nelinearna struktura podataka (iz jednog čvora se može ići na dva sledeća)

- Dinamički alocirana
- Osnovni čvor - koren stabla
- Binarno stablo pretrage, binarno stablo sa dodatnom organizacijom
    - Svaki čvor sadrži informaciju i dva pokazivača
        - levi ukazuje na podstablo koje se sastoji iz čvorova manjih od tekućeg
        - desni ukazuje na podstablo koje se sastoji iz čvorova većih od tekućeg
    - Dobre osobine
        - binarno stablo očuvava elemente sortirane
        - brza pretraga

Primer čvora binarnog stabla

```
typedef struct cvor_st {
    int inf;
    struct cvor_st *levi;
    struct cvor_st *desni;
} CVOR;
```

Primer binarnog stabla pretrage



Operacije nad stablom

1. Inicijalizacija stabla
2. Ubacivanje novog čvora u stablo
3. Pronalaženje čvora
4. Ispis sadržaja stabla
5. Brisanje čvora
6. Brisanje stabla

Inicijalizacija stabla

```
void inicijalizacija(CVOR **pkoren) {
    *pkoren = NULL;
}
```

Ubacivanje novog čvora u stablo

- Kreira se novi čvor tako što se memorija zauzme dinamički
    - Popuni se informacioni deo čvora sa konkretnom vrednošću (vrednostima)
    - Pokazivači na levo i desno podstablo postave se na vrednost `NULL`
- Prođe se kroz stablo i kada se stigne do kraja stabla (naiđe se na list), umetne se ispod njega (levo ili desno, zavisi od sadržaja)

Pravljenje novog čvora

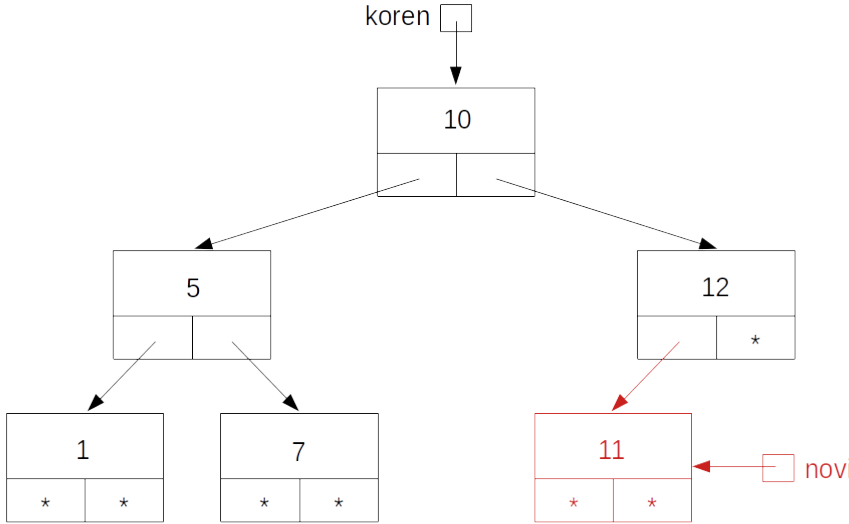
```
CVOR *napravi_cvor(int inf) {
    CVOR *novi = (CVOR *)malloc(sizeof(CVOR));

    if(novi == NULL) {
        printf("Neuspesno zauzimanje memorije!\n");
        exit(EXIT_FAILURE);
    }

    novi->inf = inf;
    novi->levi = NULL;
    novi->desni = NULL;

    return novi;
}
```

Ubacivanje novog čvora u stablo



Ubacivanje novog čvora u stablo

```
void dodaj_cvor(CVOR **pkoren, CVOR *novi) {
    if(*pkoren == NULL) {
        *pkoren = novi;
    } else {
        if((*pkoren)->inf > novi->inf) {
            dodaj_cvor(&(*pkoren)->levi, novi);
        } else {
            dodaj_cvor(&(*pkoren)->desni, novi);
        }
    }
}
```

- Pitanje: Da li dodavanje u čvor može biti realizovano i iterativno?

Pronalaženje čvora

- Krene se od korena i gleda da li je sadržaj traženog čvora manji ili veći od tekućeg
    - Ako je manji, ide se u levo podstablo
    - Ako je veći, ide se u desno podstablo
    - Ako je jednak, vrati se pokazivač na tekući element
    - Ako ne postoji, jedno od mogućih rešenja je da funkcija vrati `NULL` pokazivač

Pronalaženje čvora

```
CVOR *nadji_cvor(CVOR *koren, int vr) {
    CVOR *nadjen = NULL;

    if(koren != NULL) {
        if(koren->inf == vr) {
            nadjen = koren;
        } else {
            if(koren->inf > vr) {
                nadjen = nadji_cvor(koren->levi, vr);
            } else {
                nadjen = nadji_cvor(koren->desni, vr);
            }
        }
    }

    return nadjen;
}
```

Ispis sadržaja stabla

- Vrši se rekurzivno spuštanje do listova binarnog stabla
- Krene se od levog ili desnog čvora stabla, preko korena do desnog ili levog čvora stabla (in-order obilazak stabla)
- Zbog strukture binarnog stabla pretrage, rezultat je sortirani ispis u rastućem ili opadajućem redosledu
    - zavisi sa koje na koju stranu idemo (npr. s leva na desno)

Ispis sadržaja stabla

```
void ispisi_stablo(CVOR *koren) {
    if(koren != NULL) {
        ispisi_stablo(koren->levi);
        printf("%d ", koren->inf);
        ispisi_stablo(koren->desni);
    }
}
```

Brisanje stabla

- Vrši se rekurzivno spuštanje do listova binarnog stabla
- Krene se od korena i briše se levo i desno podstablo, pa onda koren (post-order obilazak stabla)

Brisanje stabla

```
void obrisi_stablo(CVOR **pkoren) {
    if(*pkoren != NULL) {
        obrisi_stablo(&(*pkoren)->levi);
        obrisi_stablo(&(*pkoren)->desni);
        free(*pkoren);
        *pkoren = NULL;
    }

}
```

Primer main funkcije

```
int main() {
    CVOR *koren;

    inicijalizacija(&koren);

    dodaj_cvor(&koren, napravi_cvor(10));
    dodaj_cvor(&koren, napravi_cvor(5));
    dodaj_cvor(&koren, napravi_cvor(1));
    dodaj_cvor(&koren, napravi_cvor(12));
    dodaj_cvor(&koren, napravi_cvor(7));
    dodaj_cvor(&koren, napravi_cvor(11));
    dodaj_cvor(&koren, napravi_cvor(14));

    const CVOR *nadjen = nadji_cvor(koren, 12);   // Isprobati sa razlicitim vrednostima

    if(nadjen) {
        printf("Postoji cvor sa vrednoscu %d!\n", nadjen->inf);
    } else {
        printf("Ne postoji cvor!\n");
    }
```

```
    obrisi_stablo(&koren);

    return EXIT_SUCCESS;
}
```

Zadatak 1

Napisati funkcije koje sumiraju i prebrojavaju čvorove binarnog stabla, kao dve posebne funkcije. Iskoristiti kod sa slajdova za implementaciju binarnog stabla.

- Koristiti *post-order* obilazak stabla
    - Isti raspored rekurzivnih poziva kao i kod funkcije za brisanje stabla, s tim da će funkcije iz zadatka imati povratnu vrednost (suma/broj trenutnog čvora i čvorova ispod njega)

Proširenje informacionog dela čvora binarnog stabla

- Informacioni deo čvora se može sastojati od više polja
- Prilikom dodavanja novih čvorova, bira se jedno od polja na osnovu kog će podaci biti raspoređivani u binarnom stablu pretrage
    - Često je to neki broj, mada može da bude i string (poređenje preko strcmp)

Primer čvora binarnog stabla koji opisuje automobil:

```
#define MAX_MARKA 21

typedef struct cvor_st {
    char marka[MAX_MARKA];
    unsigned godiste;
    unsigned kubikaza;
    struct cvor_st *levi;
```

```
    struct cvor_st *desni;
} CVOR;
```

Zadatak 1

Napisati program koji manipuliše strukturom sa prethodnog slajda:

- Podaci o automobilima učitavaju se iz ulazne datoteke u binarno stablo pretrage
    - ime ulazne datoteke se zadaje kao argument komandne linije
- Novi čvorovi dodaju se u odnosu na godište automobila
- U izlaznu datoteku ispisuju se automobili zadate marke
    - marka se zadaje kao argument komandne linije
    - Naziv izlazne datoteke je marka + ekstenzija ".txt"
- Na standardni izlaz treba prikazati najnoviji auto sa kubikažom ne većom od zadate
    - kubikaža se zadaje preko korisničkog unosa sa tastature u toku rada programa

Zaglavlja funkcija

```
FILE *safe_fopen(char *ime, char *rezim, int kod_greske);
void ucitaj_u_stablo(FILE *ulazna, CVOR **pkoren);
void ispisi_u_fajl(FILE *izlazna, CVOR *koren, char * marka);

void inicijalizacija(CVOR **pkoren);
CVOR *napravi_cvor(char *marka, unsigned godiste, unsigned kubikaza);
void dodaj_cvor(CVOR **pkoren, CVOR *novi);
CVOR *najnoviji_sa_kubikazom(CVOR *, unsigned kubikaza);
void obrisi_stablo(CVOR **pkoren);
```

Izgled `main` funkcije

```
int main(int argc, char **argv) {
    CVOR *koren;

    if(argc != 3) {
        printf("Primer poziva: %s ./automobili.txt Audi\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    inicijalizacija(&koren);

    FILE *ulazna = safe_fopen(argv[1], "r", 2);
    ucitaj_u_stablo(ulazna, &koren);
    fclose(ulazna);

    char naziv_izldat[MAX_NAZIV];
    strcpy(naziv_izldat, argv[2]);
    strcat(naziv_izldat, ".txt");
    FILE *izlazna = safe_fopen(naziv_izldat, "w", 3);
    ispisi_u_fajl(izlazna, koren, argv[2]);
    fclose(izlazna);
```

Izgled `main` funkcije

```
    unsigned kubikaza;
    printf("Unesite kubikazu: ");
    scanf("%u", &kubikaza);

    CVOR *najbolji = najnoviji_sa_kubikazom(koren, kubikaza);

    if(najbolji) {
        printf("Najnoviji auto sa kubikazom ne vecom od %u je: %s %u %u\n",
                kubikaza, najbolji->marka, najbolji->godiste, najbolji->kubikaza);
    } else {
        printf("Ne postoji auto sa zadatim kriterijumima!\n");
    }

    obrisi_stablo(&koren);

    return EXIT_SUCCESS;
}
```

Primer poziva i rada programa

Poziv programa:

```
./a.out automobili.txt BMW
```

Rad programa sa različitim vrednostima kubikaže:

```
Unesite kubikazu: 3000
Najnoviji auto sa kubikazom ne vecom od 3000 je: BMW 2015 2998
```

```
Unesite kubikazu: 2000
Najnoviji auto sa kubikazom ne vecom od 2000 je: Alfa-Romeo 2013 1742
```

```
Unesite kubikazu: 1700
Ne postoji auto sa zadatim kriterijumima!
```

Primer ulazne i izlazne datoteke

Primer ulazne datoteke:

```
Alfa-Romeo 2013 1742
Honda      2001 1998
Mercedes   2017 3982
KIA        2020 3342
Toyota     2005 1794
BMW        2006 3246
Alfa-Romeo 2003 3179
BMW        2015 2998
Audi       1998 1781
```

Primer izlazne datoteke `BMW.txt`:

```
BMW 2006 3246
BMW 2015 2998
```

Zadatak 2

```
.PHONY: clean

a.out: binarno-stablo.o zad1.o
	gcc binarno-stablo.o zad1.o

zad1.o: zad1.c
	gcc -c zad1.c

binarno-stablo.o: binarno-stablo.c
	gcc -c binarno-stablo.c

clean:
	rm *.o a.out
```

Slično kao kod jednostruko spregnute liste, moguće je reorganizovati kod u više datoteka i kod zadataka sa binarnim stablom. U zadatku 1, izmestiti sve funkcije koje imaju vezu isključivo sa binarnim stablom (inicijalizacija, dodavanje, brisanje binarnog stabla itd.) u par zaglavlje/implementacija `binarno-stablo.h` i `binarno-stablo.c`. Ponašanje programa nakon reorganizacije treba da ostane očuvano.