UNIVERZITET U BEOGRADU

FAKULTET ORGANIZACIONIH NAUKA

Seminarski rad iz predmeta:

Programiranje 1

Strukture podataka-Liste i binarna stabla

Mentor:

Prof. dr Saša D. Lazarević

Student:

Matea Lukić 410/2020

Beograd,

Jul 2021.

Sadržaj

Uvod 3
Odnos medju podacima 4
Statičke strukture podataka 5
Polustatičke strukture podataka 5
Stek 6
Red 6
Dinamičke strukture podataka 7
Lista 8
Jednostruko ulančana lista 8
Operacije 9
Dvostruko ulančana lista 11
Operacije 12
Ciklična lista 13
Binarna stabla 14
Osnovni pojmovi i vrste 15
Binarno stablo pretrage 16
Operacije 17
Zakljucak 19
Literatura 20

Uvod

Ovaj rad je napisan u cilju utvrdjivanja znanja iz oblasti struktura podataka, predmeta Programiranje 1 koji se sluša u drugom semestru na Fakultetu organizacionih nauka u Beogradu.

Sadržaj se može podeliti na tri dela, gde prvi čini glava u kojoj se objašnjavanju pojmovi koji će se koristiti u daljoj izradi rada.

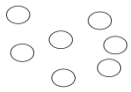
Drugi deo rada se bavi svim tipovima lista i osnovnim operacijama nad njima, pri čemu se ističe grafički prikaz istih.

Treci deo rada se odnosi na binarna stabla, vrste i neke operacije nad njima.

Odnos medju podacima

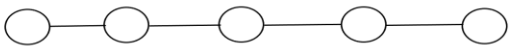
Kada govorimo o odnosu medju podacima neke strukture podataka mislimo na relacije ili osobine koje uticu na njihov raspored ili odnos unutar nje.

-Podaci sa zajedničkim svojstvima čine skup. Odnos izmedju elemenata u skupu je 0:0 odnosno ne postoji relacija njihovog rasporeda, važno je samo da su povezani postojanjem nekog zajednickog svojstva.



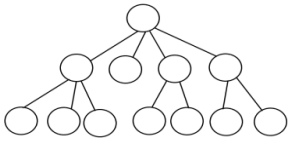
Slika 1-skup

-Svi elementi unutar liste imaju svog prethodnika i sledbenika, i to tacno po jednog pa je odnos 1:1



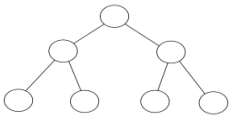
Slika 2-lista

-Ukoliko jedan element može da ima i nijednog i više sledbenika odnos je 1:M i to predstavlja stablo



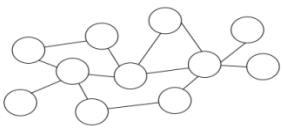
Slika 3-stablo

-Poseban vid stabla je stablo u kome je broj sledbenika uvek dva i takvo stablo se naziva binarno stablo



Slika 4-binarno stablo

-elementi mogu da budu i u uzajamnoj relaciji, tako da više ne postoji hijerarhijska veza, pa je odnos M:M i to predstavlja mrežu(graf)



Slika 5-mreža

Najčesca podela struktura podataka je po načinu izmene sadržaja i dele se na: statičke, polustatičke i dinamičke.

Statičke strukture podataka

Kod ovog tipa strukture podataka ograničen(predefinisan) je broj i tip elemenata, koji se ne može menjati. Statičke strukture imaju još mana od kojih je najvažnija niska iskorišćenost memorije koja je za nju alocirana.

Prednosti statičkih struktura:

1. Omogućeno pristupanje članovima preko indeksa
2. Jednostavni algoritmi
3. Jednostavno upravljanje memorijskim prostorom

Najpoznatije statičke strukture su vektori, nizovi i matrice.

Primer implementacije statičkih struktura:

`int niz[3] = {1, 2, 3};`-implementacija vektora(1D niza)

`int matrica[3][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };` - implementacija matrice dimenzije 3x3

`int niz[3][3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27};`-implementacija 3D niza

Polustatičke strukture podataka

Opšta karakteristika polustatičkih strukura ogleda se u tome da je dodavanje i uklanjanje elemenata kao i pristup elementima moguć samo na tačno odredjenim mestima u strukturi.

Najpoznatije polustatičke strukture su:

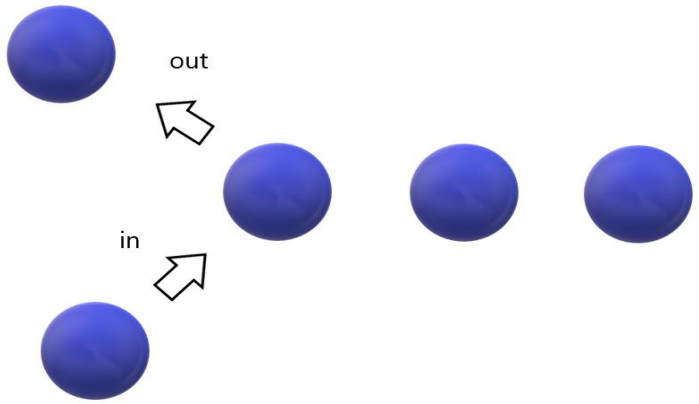
- Stek(LIFO red)
- Red(FIFO red)

Ove strukture su u osnovi linearne, a razlikuju se jedino po mestu pristupa elementima, odnosno po mestu gde je moguće dodati i ukoniti element.

Stek

Pristup je dozvoljen samo elementu na vrhu steka. Operacija pristupa ostvaruje se funkcijom koja ima opšte prihvaćeni naziv TOP, istim principom se dodavanje novog elementa vrši na vrh čime on postaje element na vrhu funkcijom PUSH. Takodje uklanjanje elementa se vrši na vrhu steka.

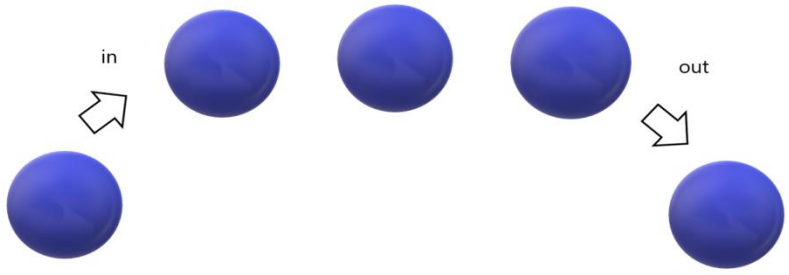
Dakle kod steka je moguće ukloniti samo onaj element koji je poslednji dodat u stek, zbog čega se stek naziva i LIFO red(Last In First Out).



Slika 6-Stek

Red

Ova struktura je takodje linearna ali se razlikuje od steka po mestu dodavanja odnosno uklanjanja elemenata. Kod reda se elementi dodaju na jednom kraju, a uklanjaju na drugom kraju strukture otkuda i naziv FIFO(First In First Out).



Slika 7-red

Dinamičke strukture podataka

Dinamičke strukture podataka zasnivaju se na korišćenju dinamičke memorije koja se dodeljuje u toku izvršavanja programa, a podaci koji se čuvaju u njoj zovu se dinamički podaci. Karakteristika rada sa dinamički dodeljenom memorijom je da ti podaci nemaju imena već im se pristupa preko adresa i nestaju automatski po završetku programa.

Deo memorije koji se dinamički alocira i dealocira u toku izvršavanja programa se naziva hip(Heap).

Prednosti korišćenja dinamičkih struktura je u tome sto je broj elemenata u toj strukturi teorijski neograničen, iskorišćenost memorije je 100% i omogućena je velika fleksibilnost u odnosu na druge oblike.

Mane se ogledaju u komplikovanijem pristupanju podacima bez direktnog pristupa i upravljanjem memorijom.

U C-u postoje 4 funkcije iz biblioteke `stdlib. h` koje omogućavaju dinamičko upravljanje memorijom:

1. malloc()-ova funkcija vraća pokazivač tipa void čija vrednost je adresa zauzetog memorijskog bloka
   -void*malloc(int n)
2. realloc()-vrši realokaciju prostora na koji pokazuje pokazivač p, a koji je prethodno bio dinamički alociran, alocira se n bajtova, kopira se stari sadržaj u novi prostor, a zatim se stari briše
   -void*realloc(void*p, int n)
3. calloc()-alocira prostor za n susednih elemenata veličine s bajtova(odnosno n*s bajtova) i inicijalizuje prostor na 0
   -void*calloc(int n, int s)
4. free()-prostor alociran prethodnim funkcijama se može dealocirati pozivom ove funkcije
   -void free(void *p)

Lista

Lista predstavlja potpuno uredjenu linearnu strukturu podataka. To znači da u listi od n elemenata:e_1, e_2, …, e_{i-1}, e_i, e_{i+1}, …, e_n gde i, n ∈ ℕ , za svaki element e važi:

1. prethodi elementu e_{i+1} i=1, …, n-1
2. sledi element e_{i-1} i=2, …, n

Drugim rečima, svaki element liste ima po jednog neposrednog prethodnika i sledbenika, sem prvog elementa koji nema prethodnika i poslednjeg elementa koji nema sledbenika. Lista bez elemenata je takodje lista. Za označavanje prazne liste se koristi konstanta Null. Vrednost elemenata liste može biti bilo kog tipa: prostog ili složenog, predefinisanog ili programerski definisanog(npr. moguće je da element liste bude celobrojni broj, znakovnog tipa ili da je tipa lista).

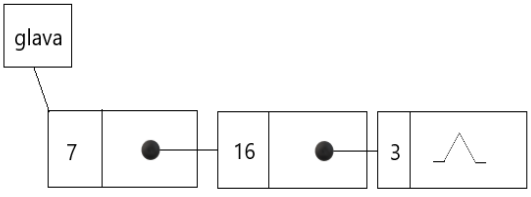
Jednostruko ulančana lista

Jednostruko ulančana lista je vrsta ulančane liste koja je jednosmerna, odnosno može se proći kroz nju samo u jednom smeru od glave do poslednjeg čvora(repa). Svaki element ulančane liste se naziva čvor. Svaki od njih sadrži podatke i pokazivač na sledeći čvor koji pomaže u održavanju strukture liste. Što nam govori da se za realizaciju koncepta ulančanih lista u programskom jeziku C koriste strukture i pokazivači.

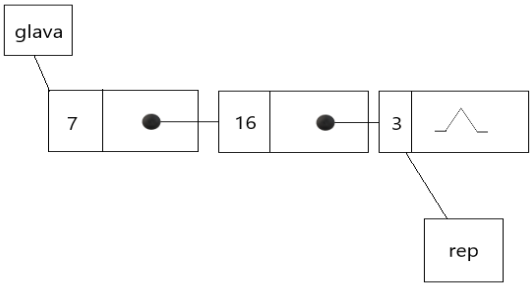
Elementi liste koji ce sadržati celobrojne podatke se definiše na sledeći način:

```
struct cvor {
	int podatak;
	struct cvor* sledeci;
};
```

Ovim kodom je definisan strukturni tip cvor koji ce predstavljati element liste. Pokazivač na prvi element se naziva glava liste, dok se pokazivač na poslednji element(ako postoji) naziva rep.



Slika 8 -jednostruko ulančana lista



Slika 9-jednostruko ulančana lista sa repom

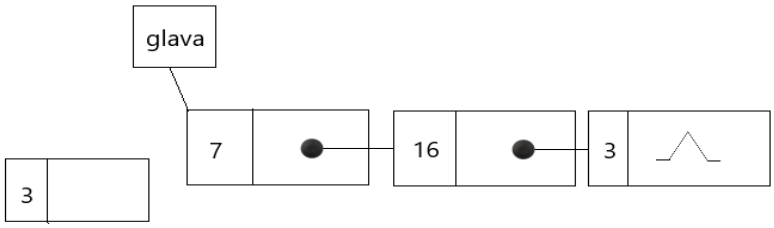
Operacije

1. Uključivanje novog elementa u listu na početak liste

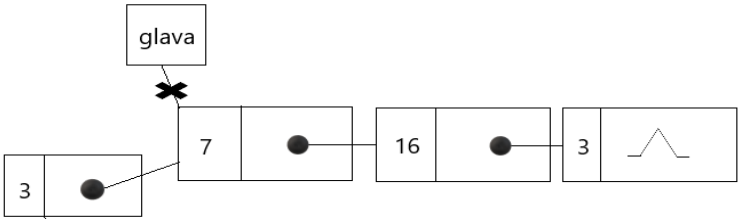
Pri dodavanju novog elementa na početak liste moraju se razlikovati dva slučaja

1. kada je lista prazna
2. kada lista nije prazna

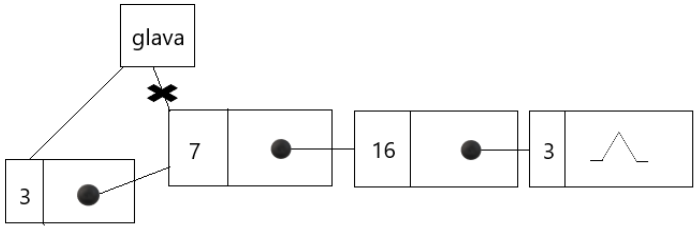
U slučaju da je lista prazna pokazivaču na prvi element(glavi), čija je vrednost NULL, se odredjuje element na koji ce pokazivati, koji ce biti taj element koji želimo da dodamo u listu. Dok u slučaju da lista nije prazna moraju se uvesti dodatne promene. Nakon kreiranja novog elementa njegov pokazivač mora pokazivati na element na koji pokazuje glava liste čime se stvara mogućnost promene vrednosti glave usled koje se ne gubi lista. Ukoliko bi prvo promenili vrednost glave lista bi bila izgubljena i jedini element u njoj bi bio novododati element.



Slika 10-dodavanje elementa na početak liste I



Slika 11-dodavanje elementa na početak liste II



Slika 12-dodavanje elementa na početak liste III

2. Uključivanje novog elementa na kraj liste

Za ovu operaciju nam je potreban način da pronadjemo poslednji element u listi zbog čega uvodimo novi pokazivač(pom) koji ce u početku pokazivati na prvi element. Ukoliko je pokazivački segment elementa liste na koji pokazuje pom jednak NULL, odnosno on je poslednji element, tada ce pokazivački segment pokazivati na novi element.

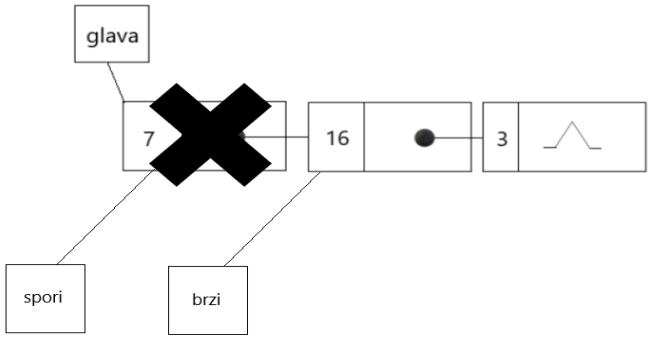
Pre prolaza kroz listu se ispituje da li je ona prazna i u slučaju da jeste novi element se dodaje na početak liste.

```
struct cvor* tekuci = glava;
while (tekuci->sledeci != NULL) {
	tekuci = tekuci->sledeci;
}
tekuci->sledeci = novielement;
```

3. Uništavanje liste

Za brisanje liste su nam potrebna dva nova pokazivača koji ce biti udaljeni za jedan element jedan od drugog pa ćemo ih zvati brzi i spori. Brzi pokazivač ćemo pomerati za jedno mesto kroz listu, a pomoću drugog pokazivača i funkcije free() ćemo brisati elemente preko kojih je brzi vec prošao.

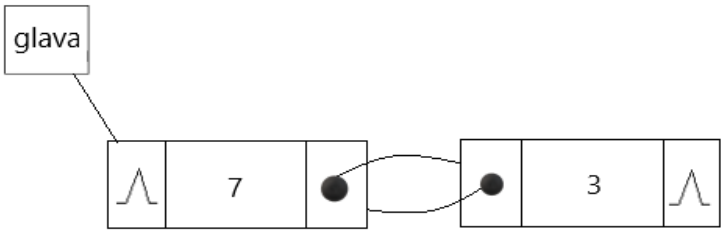
```
struct cvor* brzi = glava;
struct cvor* spori;
while (brzi != NULL) {
	spori = brzi;
	brzi = brzi->sledeci;
	free(spori);
}
glava = NULL;
```



Slika 13-brisanje liste

Dvostruko ulančana lista

Dvostruko ulančana lista je tip ulančanih listi kod kojih svaki čvor sadrži pokazivač na prethodni kao i pokazivač na naredni element. Pa se čvor sastoji od tri dela dva pokazivača i podatka koji treba biti memorisan.



Slika 14-Dvostruko ulančana lista sa dva elementa

U C-u struktura koja označava čvor bi bila

```
struct cvor
{
    struct cvor* prethodni;
    int podatak;
    struct cvor* sledeci;
};
```

Ovakva lista, za razliku od jednostruko ulančanih, omogućava prolazak kroz listu u dva pravca i može se koristiti za implementaciju binarnih stabala. Takodje ona koristi više memorije po čvoru od 1UL zbog dva pokazivača.

Iako je kod za implementaciju 2UL duži od istog za 1UL on je uglavnom intuitivniji i lakše je naslutiti rezultat različitih funkcija, što je još jedna prednost ovih listi.

Operacije

1. Ubacivanje novog elementa na početak liste

Ubacivanje elementa na početak se izvršava promenom pokazivača na prethodni element prvog člana liste, a kasnije i same glave liste. Ukoliko je lista prazna menja se samo glava.

Ta funkcija bi izgledala ovako:

```
void ubacinapocetak(struct cvor** glava,  int podatak) {
    struct cvor* novielement = malloc(sizeof(CVOR));
    novielement->podatak = podatak;
    novielement->sledeci = *glava;
    novielement->prethodni = NULL;
    if (*glava != NULL)
        (*glava)->prethodni = novielement;

    *glava = novielement;
}
```

2. Izbacivanje elementa sa kraja liste

Moramo razlikovati tri slučaja

1. Kada je lista prazna nemamo šta da izbacimo iz liste pa se funkcija završava
2. Kada lista ima samo jedan element, u tom slučaju vrednost glave ce biti NULL i pomoću funkcije free() dealociramo memoriju koja je zauzeta za prvi element
3. Kada lista ima više od jednog elementa, u ovom slučaju pomoćni pokazivač prolazi kroz listu dok ne dodje do poslednjeg elementa kada pomoću osobine 2UL možemo podesiti pokazivač pretposlednjeg elementa na NULL i dealocirati memoriju potrebnu za poslednji element

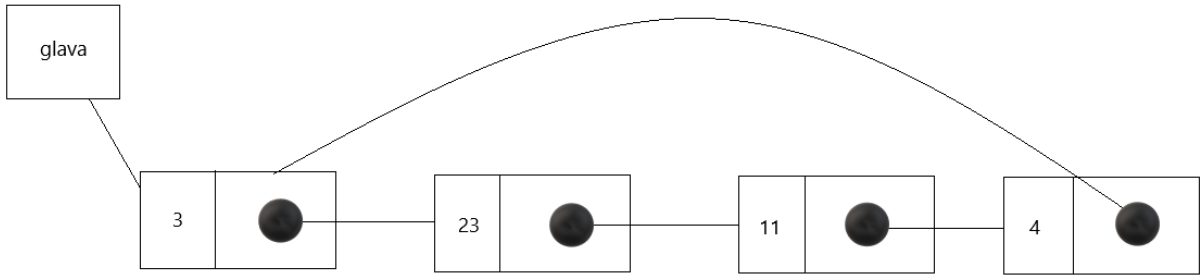
Ako je PCVOR definisan sa typedef struct cvor*PCVOR; onda važi:

```
int uklonisakraja(PCVOR*glava) {
    int uklonjen;
    PCVOR tekuci = *glava;
    if (*glava == NULL) {
        return;
    }
    else if ((*glava)->sledeci == NULL) {
        *glava = NULL;
        uklonjen = tekuci->podatak;
        free(tekuci);
        return uklonjen;
    }
    while (tekuci->sledeci != NULL) {
        tekuci = tekuci->sledeci;
    }
    uklonjen = tekuci->podatak;
    tekuci->prethodni->sledeci = NULL;
    free(tekuci);

    return uklonjen;}
```

Ciklična lista

Ciklična lista je lista u kojoj poslednji član pokazuje na prvi element i time čini zatvorenu strukturu obrazujući kružno povezanu grupu elemenata. Prolazak kroz ovakvu listu se odvija kretanjem od bilo kog elementa sve dok se ne dodje ponovo do početnog elementa, pošto ovakva lista nema kraj ni pocetak, takodje nijedan pokazivač elementa ovakve liste neće biti NULL. Umesto glava, za pokazivač na listu se ponekad koristi naziv pristupni pokazivač.

Ciklična lista može imati i oblik jednostruko ulančane liste i oblik dvostruko ulančane liste.



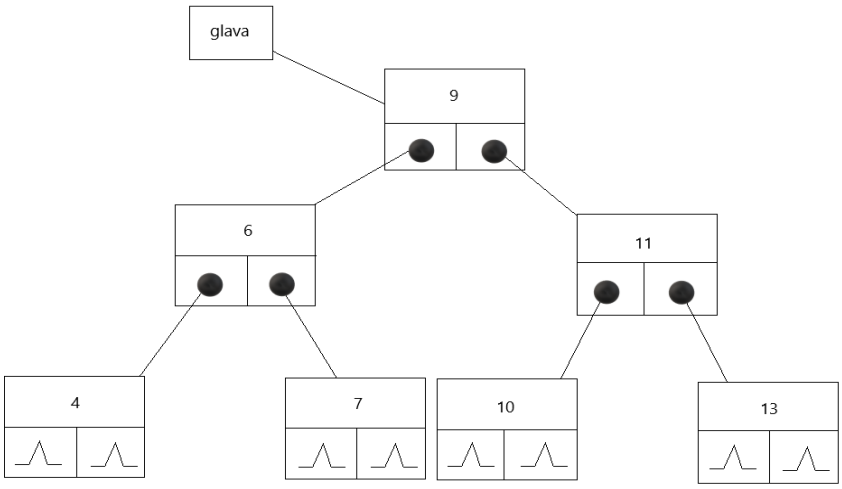
Slika 15-ciklična lista u obliku 1UL

S obzirom da nema elemenata sa NULL pokazivačem u listi za njen prikaz možemo iskoristiti do-while petlju i takva funkcija bi izgledala ovako:

```
void printujlistu(PCVOR glava) {
	PCVOR tekuci = glava;

	do {
		printf("%d->",  tekuci->podatak);
		tekuci = tekuci->sledeci;

	} while (tekuci != glava);
	printf("NULL");

}
```

Binarna stabla

Binarno stablo je struktura koja je sačinjena od čvorova koji su medjusobno povezani po principu piramide, namenjena čuvanju podataka.



Slika 16-binarno stablo

Svaki čvor binarnog stabla može da pokazuje na najviše dva elementa dok stablo ima samo jedan element na koji ne pokazuje nijedan drugi koji se naziva koren stabla(čvor sa brojem 9 u našem primeru). Od korena se može doći do bilo kog drugog elementa stabla. Dok podgrana označava skup svih čvorova koji se nalaze levo ili desno od odgovarajućeg čvora.

- Čvor stabla je jedna memorijska ćelija stabla
- List je čvor stabla koji nema nijedan podčvor
- Roditelj nekog čvora je čvor koji pokazuje na njega
- Dete čvora je čvor na koji on pokazuje

Kako je binarno stablo najjednostavniji oblik stabla ono se u programskom jeziku C može definisati pomoću:

```
typedef struct cvor {
        int podatak;
        CVOR* levi;
        CVOR* desni;
}CVOR;
```

Osnovni pojmovi i vrste

Stepen čvora stabla je broj podstabala kojima je ovaj čvor koren, odnosno stepen je broj naslednika čvora. Ukoliko je stepen čvora nula to znači da je on list stabla. Stepen stabla je maksimalan stepen svih čvorova stabla.

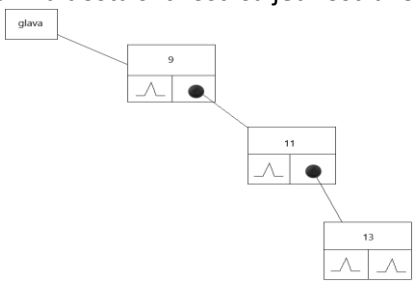
Nivo čvora definišemo tako što uzimamo da je nivo korena stabla jednak 1, a zatim uvećavamo ovaj broj za jedan pri svakom skoku od korena ka podstablima. Dubinu stabla zatim definišemo kao maksimalan nivo čvora u stablu.

Za binarno stablo važi:

1. Maksimalan broj čvorova na i-tom nivou je 2^{i-1}
2. Ako je k dubina binarnog stabla onda je maksimalan broj čvorova koje stablo moze da ima je 2^k-1

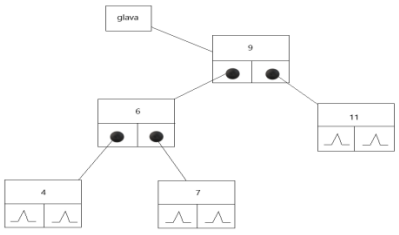
Postoje različiti tipovi binarnih stabala od kojih su neka:

1. Iskošena binarna stabla-u ovim stablima svaki čvor ima samo jedno dete, drvo može biti leve ili desne orijentacije. Ovakvo drvo ima dosta sličnosti sa jednostruko ulančanim listama.



Slika 17-iskošeno binarno stablo

2. Puno binarno stablo-ovakav tip stabla je dat na slici 21. , osobina ovog tipa je da stablo dubine k sadrži maksimalan broj čvorova odnosno 2^k-1. Kada je broj čvorova manji od maksimalnog takvo binarno stablo se naziva nepotpunim.
3. Kompletno binarno stablo-binarno stablo koje je skroz puno, sa mogućnošću izuzetka donjeg nivoa, koje je napunjeno sa leva na desno. U kompletnom binarnom stablu, svi čvorovi su najlevlje moguće.



Slika 18-kompletno binarno stablo

Stabla kao sto su iskošena stabla, odnosno ona kod kojih roditeljski čvor ima samo jedno dete se nazivaju degerisana stabla . Kod ovih stabala složenost operacija u najgorem slučaju je O(n). Ovaj slučaj se izbegava balansiranjem stabla, za šta je najpoznatija metoda samobalansirajućih stabala čiji su primeri AVL stabla i Crveno-crna stabla.

Binarno stablo pretrage

Binarno stablo za pretraživanje je binarno stablo koje ima sledeće karakteristike:

1. Čvorovima binarnog stabla su pridružene vrednosti po kojima se obavlja pretraživanje
2. Za bilo koji čvor važi da svi čvorovi u levom podstablu imaju vrednost manju od njega, a svi čvorovi u desnom podstablu imaju vrednost veću od njega. I levo i desno podstablo su takodje binarna stabla pretrage(kao na slici 23).
3. Ne smeju postojati duplikati čvorova

Ovo stablo je poznato i kao sortirano binarno stablo. Može se koristiti za pretragu u okviru O(log(n)) vremena. Prednosti ovog stabla su:

- Pretraga može biti veoma efikasna
- Veoma jednostavna implementacija
- Čvorovi su dinamički po prirodi

Neki od nedostataka su:

- Oblik binarnog stabla zavisi samo od reda ubacivanja zbog cega stablo može biti degenerisano
- Ključevi u binarnom stablu pretrage mogu biti dugački, što može uticati na vremensku složenost.

Algoritam za pretragu bi izgledao ovako:

```
If koren == NULL
       return NULL;
If broj == koren->podatak
       return koren->podatak;
If broj < koren->podatak
       return search(koren->levi)
If broj > koren->podatak
       return search(koren->desni)
```

Ovaj algoritam se zasniva na osobini 2. Odnosno na ispravnom rasporedu elemenata koji uslovljava postojanje binarnog stabla pretrage. Ako je broj manji od korena on sigurno neće biti u desnom podstablu u suprotnom možemo eliminisati levo podstablo. Ponovnom primenom iznova sužavamo količinu ostalih čvorova dok ne nadjemo odgovarajući.

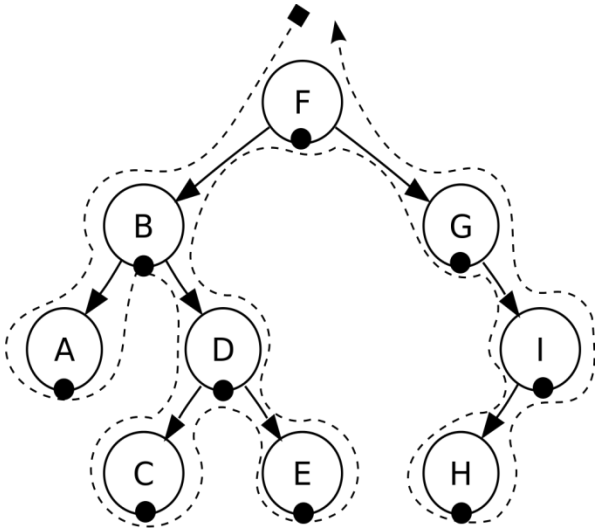
Operacije

Prikaz i ubacivanje novog elementa u binarno stablo pretrage

Prikaz stabla se može implementirati na tri načina:

1. Preorder, prvo se obilazi koren, levo podstablo pa desno
2. Inorder, prvo se obilazi levo podstablo, koren pa desno podstablo
3. Postorder, prvo se obilazi levo podstablo, desno pa koren

Primer inorder obilaska gde je redosled obilaska: A, B, C, D, E, F, G, H, I



Slika 19-inorder obilazak stabla

Funkcija za inorder prikaz stabla u C-u:

```
void inorder(struct cvor* koren)
{
    if (koren == NULL)
        return;
    inorder(koren->levi);
    printf("%d ",  koren->podatak);
    inorder(koren->desni);
}
```

Ukoliko bi za koren imali broj 53, a za elemente: 50, 55, 60, 30 ispis bi bio 30, 50, 53, 55, 60

Za dodavanje novog elementa u binarno stablo pretrage moramo razlikovati dva slučaja od kojih zavisi postupak:

1. Ako je ispitivani koren prazan (odnosno njegova vrednost je NULL) pozivamo funkciju koja će taj element tretirati kao koren stabla ili odgovarajućeg podstabla
2. Ako stablo nije prazno imamo dva podslučaja u zavisnosti od toga da li je element veći od korena ili manji, u zavisnosti od čega se element ubacuje na odgovarajuće mesto

Funkciju za ubacivanje ćemo implementirati rekurzivno:

```
struct cvor* ubaci(struct cvor* koren,  int podatak)
{

    if (koren == NULL)

    if (koren->podatak < podatak)
        koren->desni = ubaci(koren->desni,  podatak);

    else if (koren->podatak > podatak)
        koren->levi = ubaci(koren->levi,  podatak);

    return koren;
}
```

Pri čemu bi funkcija za ubacivanje korena bila:

```
struct cvor* ubacikoren(int podatak)
{
    struct cvor* novi = malloc(sizeof(struct cvor));
    novi->podatak = podatak;
    novi->levi = NULL;
    novi->desni = NULL;

    return novi;
}
```

Zakljucak

U velikom broju realnih aplikacija količina memorije za njen ispravan rad zavisi od interakcije sa korisnikom, njegovih zahteva i želja ishoda procesa obavljenih od strane aplikacije. Kako se tek u fazi izvršavanja programa odredjuje potrebna količina memorije nije nam dovoljno statičko dodeljivanje memorije. Iako je moguće predvideti gornje ograničenje, loša pretpostavka može doprineti velikom smanjenju efikasnosti rada programa. Rešenje ovih problema je dinamička alokacija memorije. Ovaj metod nam omogućava da alociramo i dealociramo memoriju na heap-u u toku izvršavanja programa koja nam je neophodna za veću iskorišćenost memorije.

Literatura

M. Jurak(2004), Programski jezik C 196-202

dr Predrag Janičić, dr Filip Marić(2021), Programiranje 2 143-183

Mateamatički fakultet u Beogradu, Predavanja iz predmeta Strukture podataka i algoritmi 2, dostupno na: https://imi.pmf.kg.ac.rs/moodle/course/view.php?id=37

Računarski fakultet u Beogradu, Predavanja iz predmeta Uvod u programiranje, dostupno na: https://petlja. org/biblioteka/r/kursevi/uvod-u-programiranje

Elektrotehnički fakultet u Beogradu, Predavanja iz predmeta, Programiranje 2, dostupno na: https://rti. etf. bg. ac. rs/rti/ir1p2/materijal. html