6. Osnovne strukture podataka

Strukture podataka se bave organizacijom podataka i načinom njihovog memorisanja u kompjuterima.

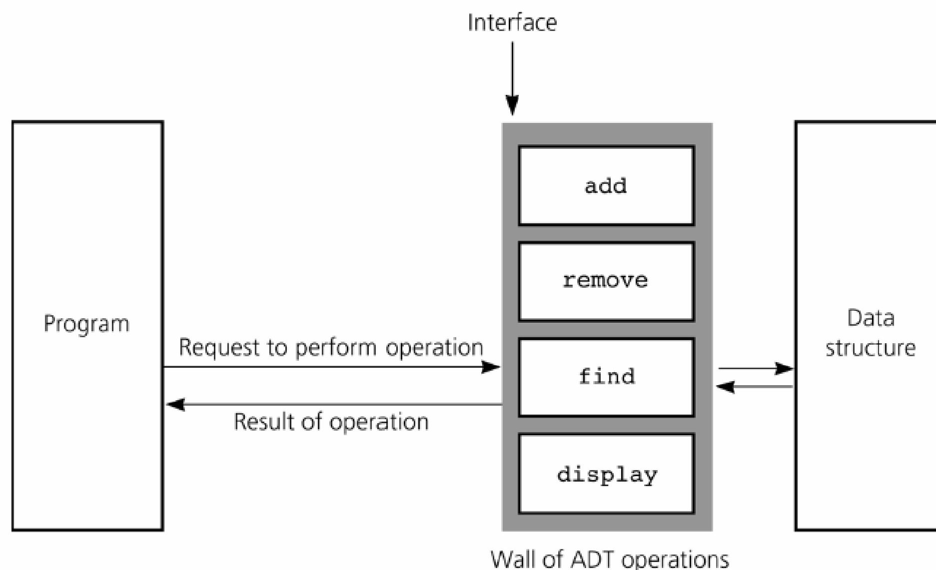
Struktura podataka je takva konstrukcija koja se može izraziti programskim jezikom za memorisanje skupa međusobno povezanih podataka - npr. niza celih brojeva, niza objekata, niza nizova, itd. Često se kompjuterske strukture podataka nazivaju i apstraktnim strukturama, jer se podaci modeliraju tako da vrše apstrakcije koje na najbolji način odgovaraju problemu koji se rešava. Kaže se da se u načinu predstavljanja problema podacima zapravo krije i njegovo rešenja.

Apstrakna struktura podataka (Abstract Data Structure-Type) - ADT

Skup (kolekcija) povezanih podataka sa skupom operacija nad tim podacima naziva se apstraktnom strukturom podataka. Apstrakna struktura pokazuje koje su operacije definisane nad podacima, ali ne i kako se te operacije implementiraju. Programer, znači, može da koristi apstaraktnu strukturu bez znanja kako je ona implementirana.

Operacije nad strukturama podataka

Svaka (apstraktna) struktura podataka ima sa njom povezan skup operacija kojima se uspostavlja sama struktura (konstruktorske operacije), kao i niz operacija za dodavanje, brisanje, sortiranje i pretraživanje podataka memorisanih unutar strukture, kao što je ilustrovano su na sledećem dijagramu.



Slika 6.1 Upotreba apstarktnih struktura podataka

Gornja slika slikovito predstavlja poznatu “formulu”:

PROGRAM = ALGORITAM + STRUKTURA PODATAKA

Gornjom formulom se izražava povezanost strukture podataka sa rešenjem (algoritmom) problema.

Slika prikazuje kako program preko interfejsa (operacije, funkcije, procedure, metoda) koristi strukturu podataka. U objektno orjentisanoj metodologiji zapravo struktura podataka zajedno sa interfejsom predstavlja osnovni element OO pristupa. Tako se definiše klasa objekata koji su modelirani strukturom podataka i metodama njihove manipulacije (dinamičkog kreiranja objekta, modifikacije objekta, i sl). OO pristup se može smatrati generalizacijom gornje formule o programima.

O Objektno orjentisanom programiranju biće više reči kasnije.

Apstrakne strukture podataka imaju niz prednosti nad klasičnim pristupom, jer s jedne starne precizno definišu objekte, a s druge strane olakšavaju programiranje jer:

- Sakrivaju nepotrebne detalje (implementaciju)
- Olakšavaju upravljanje razvojem kompleksnog softvera
- Olakšavaju održavanje softvera
- Smanjuju potrebu za funkcionalnim promenama softvera
- Omogućavaju lokalne a ne globalne promene

Nizovi i indeksi (pointeri)

Niz predstavlja kolekciju objekata (podataka) istog tipa smeštenih u memoriji u uzastopnim memorijskim lokacijama kao što prikazuje sledeća slika.

P[0]	P[1]	...	P[n-2]	P[n-1]

U gornjem primeru prikazan je slučaj jenog niza od n elemenata (zapazite da brojanje počinje od 0), gde čitav niz ima zajedničko ime **p**.

Pojedinačnom elementu niza pristupa se tako što se pored imena niza u zagradi navede i redni broj (indeks) tog elementa u nizu. Tako je p[14] podatak koji se nalazi u 15-tom elementu niza p.

Nizovi su standardan element u programskim jezicima i obično se definišu (deklarišu) na početku programa kao u sledećem primeru iz jezika C.

int kolicina[100];

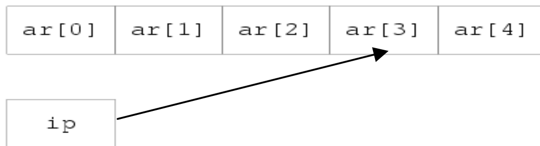
Ovde je deklarisan niz sa imenom od 100 elemenata, celobrojnog tipa.

POINTERI

Pointer predstavlja specijalnu vrstu podataka koji se interpretiraju kao adrese na podatke, a nisu podaci sa nekim drugim značenjem.

```
int ar[5], *ip;
```

Gornjom deklaracijom definisan je niz p i pointer pp



ip = &ar[3];

Ovde je ip pointer na četvrti element niza ar.

Všedimenzionalni nizovi

Možemo zamisliti i nizove sa više dimenzija (indeksa). Tako bi dvodimenzioni niz deklarisan sa:

```
int t_d[2][3]
```

mogao biti predstavljen kao dvodimenziona šema (matrica)

t_d[0][0]	T_d[0][1]	t_d[0][2]
t_d[1][0]	T_d[1][1]	t_d[1][2]

dok bi mogao da ima sledeće preslikavanje u memoriji računara.

Trodimenzioni niz:

```
int three_dee[5][4][2];
```

ima 5x4x2=40 elemenata, gde se svakom od elemenata pristupa preko tri indeksa.

Ovaj niz bi slikovito mogao da bude prikazan kao paralelopiped (trodimenzionalno telo) izdeljeno na 40 "kućica", gde svaka kućica ima svoju adresu izraženu sa tri indeksa, kao npr. three_dee[3][2][1].

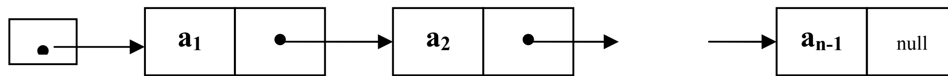
Iako je moguće zamisliti i nizove sa više od tri indeksa, njihova vizuelizacija nije moguća.

Treba ovde još istaći da elementi niza mogu biti proizvoljni objekti, pa čak i nizovi. Tako nizovi mogu sadržati kao elemente brojeve (cele i decimalne), slova, slike, itd.

Liste

Lista je konačan skup podataka $a_1,a_2,a_3,...,a_n$ sličan nizu koji smo ranije definisali. Međutim, za razliku od niza liste mogu biti realizovane kao dinamičke strukture kod kojih se unapred ne zna broj elemenata.

Sledeća slika ilustruje jednu jednostruko povezanu listu.



Slika 6.2 Lista

Liste su vrlo rasprostranjene u programiranju – lista događaja, lista poruka, lista klasa itd.

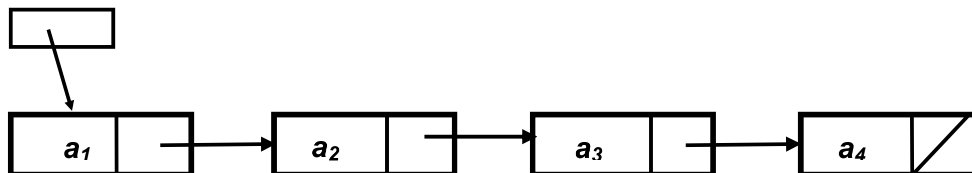
Tipične operacije sa listama:

- Kreiranje liste
- Dodavanje (Insert) elementa u listu
- Brisanje (Remove) elementa liste
- Test da li je lista prazna (bez članova)
- Traženje elmenta u listi
- Trenutni element/ sledeći/ prethodni
- Nađi k-ti element
- Štampaj celu listu
- Itd.

Implementacija lista POINTERIMA

Sada ćemo prikazati jedan efikasniji način implementacije liste pomoću pointera, dinamički.

Glava liste – pointer na prvi element

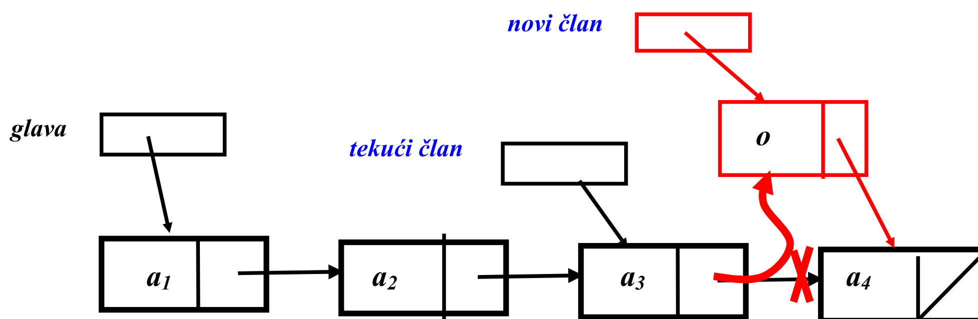


Slika 6.3 Implementacija liste pointerima

Dakle, kod implementacije pointerima lista se sastoji od jednog posebnog pointera koji ukazuje na početak liste, a svaki element u listi se sastoji iz dva dela (za podatke i pointera koji ukazuje na adresu sledećeg elementa liste), kako je to prikazano na gornjoj slici.

Sada ćemo ilustrovati kako se operacije insert i delete sa lakoćom realizuju u ovom slučaju, kada je lista implementirana pointerima.

Operacija INSERT



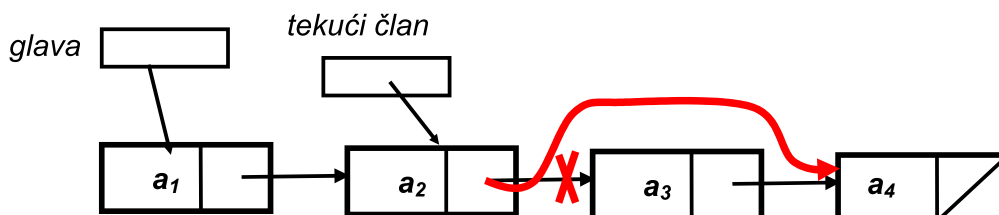
Slika 6.4 Dodavanje novog člana u listu

Ako novog člana liste treba dodati posle tekućeg člana (a_3), onda se takva operacija može opisati na sledeći način.

Jednostavno se pointer člana a_3 promeni da pokazuje na novi član, a pointer novog člana uzme predhodnu vrednost pointera člana a_3. Znači, nema nikakvog pomeranja članova niza, već se prostom zamenom pointera izvrši dodavanje novog člana.

Operacija DELETE

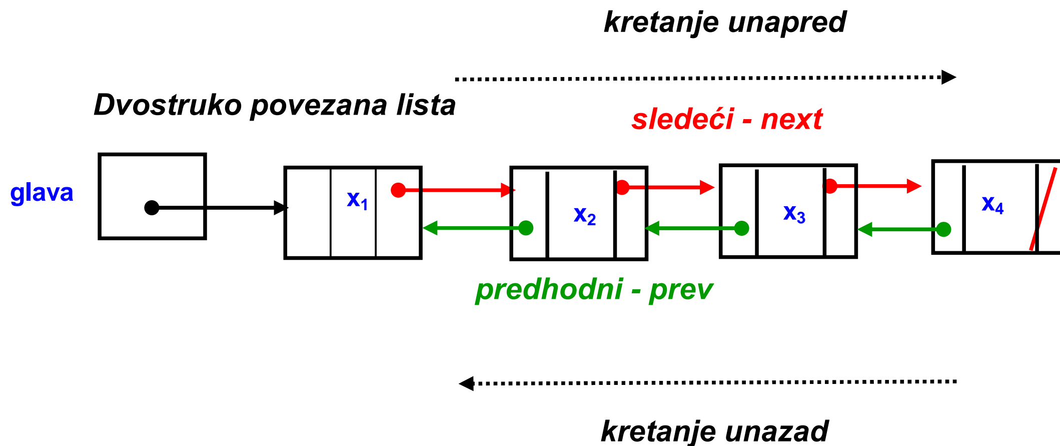
Kod brisanja je stvar još jednostavnija. Prosto se članu koji predhodi brisanom članu promeni pointer da ukazuje na član koji sledi posle brisanog člana, kako to prikazuje sledeća slika.



Slika 6.5 Brisanje člana iz liste

Dvostruko povezane liste

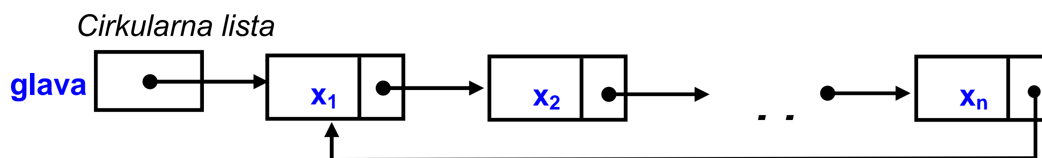
Ponekad je pogodno da se kretanje kroz listu, od elementa do elementa vrši u oba smera (napred i nazad). U takvim slučajevima organizuju se liste sa dva pointera, jedan za kretanje unapred kao kod obične liste, i dodatni pointer za kretanje unazad, što je ilustrovano na sledećoj slici.



Slika 6.6 Dvostruko povezana lista

Cirkularne liste

Cirkularna lista nastaje kada umesto da poslednji član liste ima NULL pointer (NULL pointer je pointer koji ne pokazuje ni na jedan element) on sadrži adresu prvog člana. Time se omogućava kružno-cirkularno kretanje kroz listu, kao što se možete uveriti posmatranjem sledeće slike.



Slika 6.7 Cirkularna lista

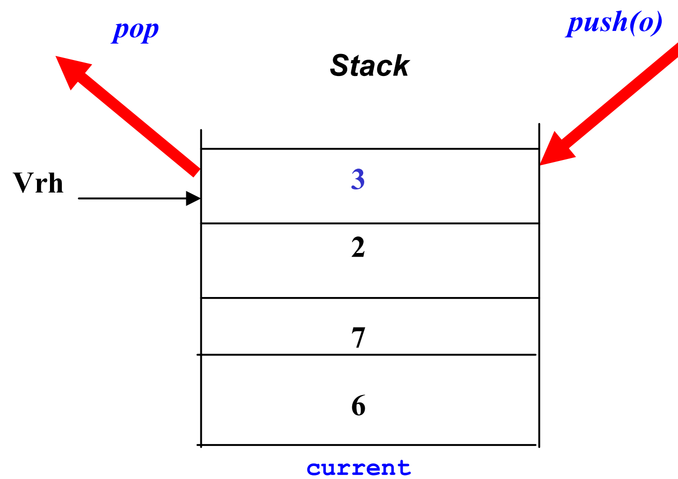
Generalno govoreći, liste su jedna od najčešće korišćenih apstraktnih struktura i služe za implementacijo mnogih drugih i složenijih struktura. Posebno je pogodna kada se implementira pointerima i dinamički. Objektno orjentisani jezici obično sadrže u biblioteci klasu koja odgovara napred izloženom konceptu liste i iz nje izvedenih drugih struktura o kojima će sada biti reči.

Stekovi (LIFO lista, Stack)

Stek je lista kod koje se dodavanje i brisanje člana uvek vrši na jednom mestu, tj. na vrhu steka (početku liste).

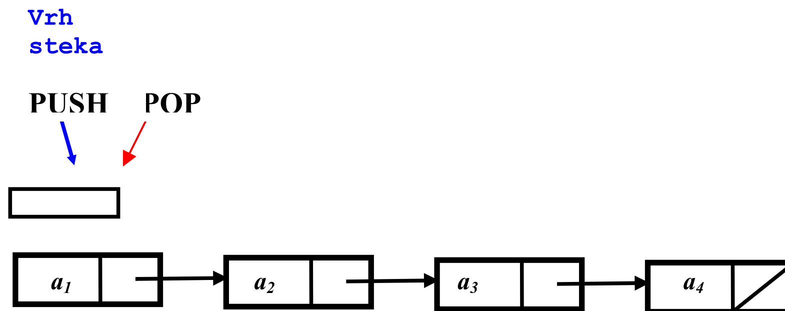
Operacije dodavanja i brisanja u steku imaju posebne nazive: push (dodaj-insert) and pop (obriši-delete).

Stek se može slikovito prikazati na sledeći način:



Slika 6.8 Primer steka

Implementacija pomoću liste



Slika 6.9 Implementacija steka pomoću liste

Kod operacije POP pointer vrha uzima vrednost koju je imao pointer prvog elementa steka. Operacijom PUSH pointer vrh pokazuje na novi element a novi element na prvi element pre izvođenja PUSH operacije.

Stek ima raznovsne primene kao što su:

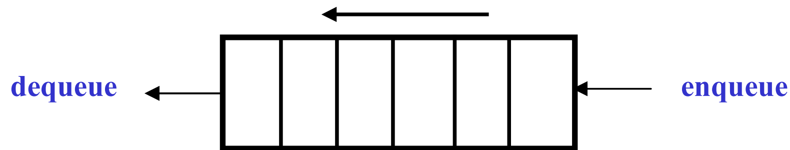
- *kod editora teksta (line editing)*
- *uparivanje zagrada (leksička analiza)*
- *izračunavanje u postfiks notaciji*
- *rekurziji i pozivu potprograma (funkcije)*

Redovi (FIFO liste, Queue)

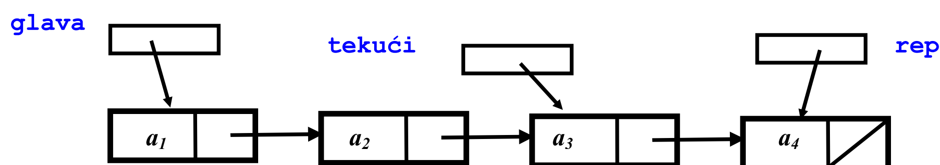
Kao i stekovi, redovi se takođe mogu implemetirati kao liste. Kod redova se, međutim, dodavanje vrši na jednom kraju a brisanje na drugom kraju liste.

Redovima se implementira FIFO (first-in first-out) princip. Npr., printer/job red.

Dve osnovne operacije sa redovima:
–dequeue: brisanje elementa sa čela reda
–enqueue: dodavanje člana na kraj reda



Implementacija pomoću liste

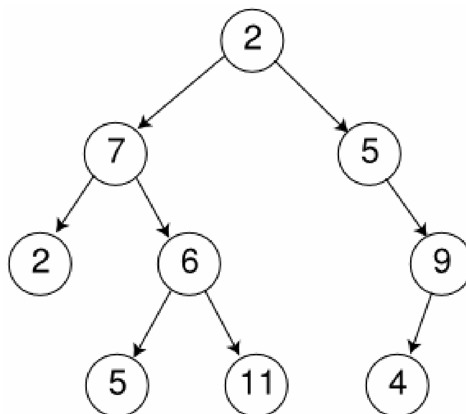


Slika 6.10 Implementacija reda pomoću liste

Primene redova
- Printer redovi za čekanje na čtampu,
- Telekomunikacioni redovi ,
- Simulacije,
- Itd.

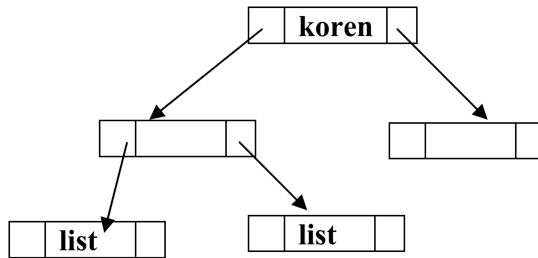
Binarno stablo (Binary Tree)

Binarno stablo je struktura podataka koji su povezani tako da svaki od podataka (čvorova stabla) može imati najviše dva sukcesora. Elementi binarnog stabla su: Koren (root), unutrašnji elementi i listovi (leaf) stabla. Koren (root) je čvor koji se nalazi na vrhu stabla



Slika 6.11 Primer binarnog stabla

Implementacija listama



Slika 6.12 Implementacija binarnog stabla listom

Svaki element u stablu ima pored podataka i dva pointera koji pokazuju na levo i desno podstablo (ako ih ima, ili NULL ako ih nema).

Operacije nad binarnim stablima

- dodavanje novog elementa
- brisanje postojećeg elementa
- prolazak: preorder, inorder, postorder

Dodavanje i brisanje elenata stabla vrši se promenom pointera slično kao kod lista (ako su stabla implementirana pomoću liste), tako da se nećemo posebno baviti ovim operacijama. Samo ćemo, nešto kasnije, pokazati jednu strategiju za dodavanje elemenata u stablo koja će rezultirati jednom specijalnom vrstom stabla – takozvanim sortnim binarnim stablom.

Malo detaljnije ćemo se baviti operacijama sistematskog obilaska svih elemenata stabla – takozvanim algoritmima za prolazak kroz stablo. Ovo iz razloga što ovi algoritmi dobro ilustruju jedan važan koncept u programiranja – takozbanu rekurziju.

Algoritmi prolaska kroz binarno stablo.

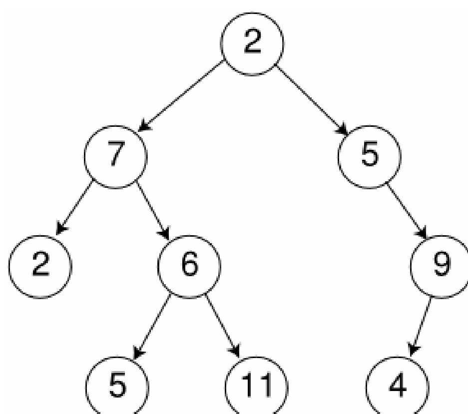
Preorder algoritam

Preorder algoritam se može iskazati na sledeći načina, sa tri koraka.

Preorder: 1. Poseti koren
2. Poseti levi deo stabla (levo podstablo)
3. Poseti desni deo stabla (desno podstablo)

Korak 1. je eksplicitan. U njemu se posećuje koren, tj jest čvor koji je na vrhu stabla (ili podstabla, kada se posećuje podstablo). Koraci 2 i 3 su implicitni (rekurzivni), jer nam ne kažu koji čvor treba posetiti, već nam daju uputstvo kako nastaviti sa prolazon – kroz levo ili desno podstablo. Zapravo poseta levom (ili desnom) podstablu će se vršiti istim ovim **preorder** algoritmom, ali sada primenjenim na podstablu. I tako redom dok god ima novih podstabala.

Za primer preorder prolaska kroz stablo uzmimo stablo sa sledeće slike.



Ako primenimo preorder algoritam imamo:

Korak 1. Posetimo koren stabla. Dakle posetimo vrh koji sadrži podatak 2 i označimo ga posećenim.

Korak 2. Sada treba da posetimo levo podstablo, tj stablo koje ima čvorove 7,2,6,5,11. Kako da ga posetimo. Pa opet istim algoritmom, ali sada primenjenim na ovo podstablo. A to znači:

- Korak 1. Poseti koren. Znači poseti i označi posećenim čvor 7 (koji je koren podstabla koji upravo posećujemo).
- Korak 2. Poseti levo podstablo tekućeg podstabla. To je podstablo koje se sastoji od samo jednog čvora – čvora 2. Tako imamo:
  - Korak 1. Poseti koren – označi 2 posećenim.
  - Korak 2. Nema levog podstabla.
  - Korak 3. Nema desnog podstabla.
  - Znači nastavljamo sa korakom 3 za levo podstablo originalnog stabla.
- Korak 3. Poseti desno podstablo. To je podstablo sa čvorovima 6,5,11
  - Korak1. Poseti koren – to je čvor 6.
  - Korak 2. Poseti levo podstablo. To je samo jedan čvor – 5.
  - Korak 3. Poseti desno podstablo. To je samo jedan čvor – 11.
  - Sada se vraćamo na Korak 3 za pocetno stablo.

Korak 3 Sada treba da posetimo desno podtablo, tj stablo koje ima čvorove 5,9,i 4.

- Korak1. Poseti koren – to je čvor 5 oynačimo ga posećenim.
- Korak 2. Poseti levo podstablo – nema
- Korak 3. Poseti desno podstablo – to je stablo sa čvorovima 9 i 4.
  - Korak 1. Poseti koren – to je čvor 9, označi ga posećenim.
  - Korak 2. Poseti levo podstablo – to je samo jedan čvor, 4, označi ga posećenim.
  - Korak3. Poseti desno podstablo – nema.

Time je algoritam završen. Redosled posete čvorova je sledeći: 2,7,2,6,5,11,5,9,4.

Sličan postupak se sprovodi i za druga dva načina prolsaka kroz stablo. Za vežbu odredite redosled prolaska kroz gornje stablo za inorder I postorder redolsede.

Inorder: 1. Poseti levi deo
2. Poseti root
3. Poseti desni deo

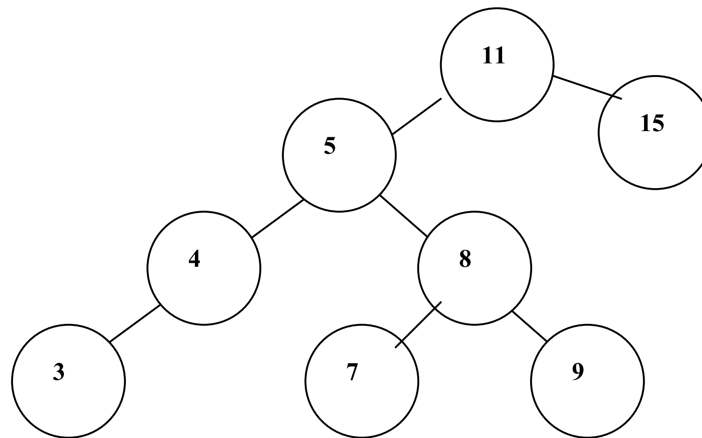
Postorder: 1. Poseti levi deo
2. Poseti desni deo
3. Poseti root

Sortno binarno stablo

Sortno binarno stablo je stablo koje se dobija kada se niz brojeva (ili nekih drugih objekata) unose u stablo po sledećem algoritmu.

1. Prvi pristigli broj (objekat) se unosi u koren stabla.
2. Svaki sledeći se rekurzivno unosi u stablo tako da ako je manji od broja (objekta) u korenu ide u levo podstablo, a ako je veću u desno.

Tako na primer niz brojeva 11,5,8,4,3,9,7,15 biće smešten u sortno binarno stablo kao što je prikazano na sledećoj slici.



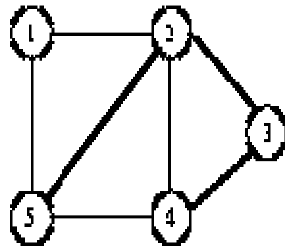
Slika 6.13 Sortno binarno stablo

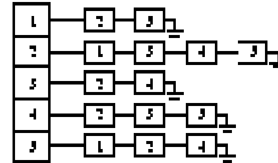
Ako se sada kroz ovo stablo prođe inorder redosledom dobićemo niz: 3,4,5,7,8,9,11,15. Zapravo dobićemo sortiran ulazni (početni niz). Dakle, ovaj postupak se može koristiti za sortiranje (veoma efikasno) nizova brojeva (objekata).

Grafovi G(nodes, vertices) – čvorovi i lukovi.

Grafovi su posebne mrežne strukture koje nalaze primenu u raznim algoritmima optimizacije, teoriji igara, telekomunikacijama, itd.

Na sledećoj slici prikazan je jedan graf i njegova implementacija kombinacijom nizova i lista. Kružići u grfau predstavljaju takozvane čvorove (engleski nodes) , a linije lukove (engleski arcs).





INPUT OUTPUT

Slika 6.14 Graf

Graf se može implementirati kao niz lkoji sadrži čvorove uz koje je pridružena lista susednih čvorova sa kojima je čvor povezan. Tako se na predhodnoj slici nalazi lista sa 5 elemenata koja sadrži čvorove 1 do 5. Svaki element iz niza ima pointer na listu. Tako čvor 1 ima pointer na čvor 2 a ovaj na čvor 5, jers su čvorovi 2 i 5 susedni čvoru 1. Slično tome, čvor 2 ima pointer na listu čvorova 1,3,4 i 5 jer je čvor 2 povezan sa svim tim čvorovima.

Algoritmi sa grafovima su složeniji i izlaze van okvira ovog predemeta. Ovde ćemo samo napomenuti da se za sistematski prolaz kroz graf mogu koristiti dve osnovne startegije: prvo u dubinu (depth first) ili prvo u širinu (breadth first). Kod prve se najpre ide po dubini grafa, tj. ide se sve dalje od početnog čvora, a kod druge se najpre posećuju susedni čvorovi.

Heš tabele

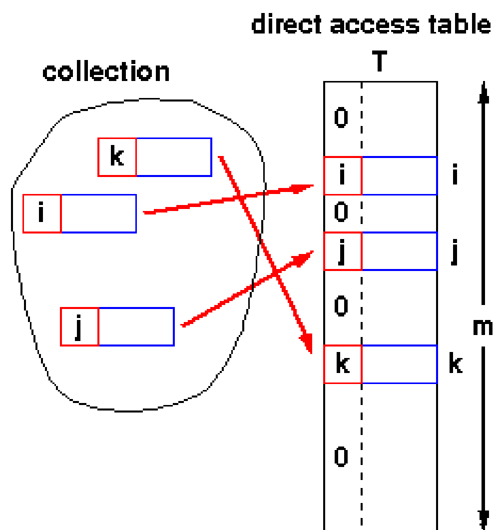
Heš tabele je posebna struktura podataka koja podseća na asocijativnu memoriju kod čoveka. U toj strukturi sam podatak koji se traži ukazuje na moguću adresu na kojoj se on nalazi u memoriji. To se postiže takozvanom transformacijom podatka (ključa) u adresu na sledeći način:

adresa= f(ključ), gde je f neka matematička funkcija.

Najčešće se primenjuju funkcije opisane metodama 1,2 i 3.
Metod 1: adresa = (ključ) MOD N + 1
Metod 2: promena baze (radix conversion)
Metod 3: kvadriranje ključa i uyimanje dela rezultata

Implementacija heš tabele nizovima

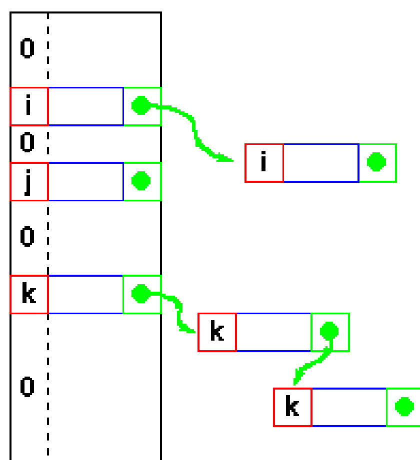
Podaci iz neke kolekcije podataka imaju poseban deo koji se zove ključ i koji služi za izračunavanje adrese na kojoj će podataka (zajedno sa ključem) biti smešten. Taj postupak je ilustrovan na sledećoj slici.



Slika 6.15 Heš tabela

Međutim, postoji verovatnoća da će transformacija dva različita ključa dati istu adresu, tako da dolazi do takozvane kolizije ključeva. Tada bi oba ključa trebalo smestiti u istu adresu, što je naravno nemoguće. U takvim slučajevima se pribegava posebnom postupku razrešavanja kolizije, što je moguće postići na više načina.

Na sledećoj slici je prikazano rešavanje kolizije olančavanjem (chaining).



Samo ćemo navesti još neke metode rešavanja kolizije bez dalje elaboracije:

1. Posebno područje memorrije za koliziju (overflow areas),
2. Primena alternativne heš funkcije (re-hashing),
3. Smeštanje na slučajno odabranu adresu (random probing)

Pitanja

1. Navedite bar tri osnovne strukture podatak.
2. Šta je ADS (ADT)?
3. Navedite neke od operacija nad strukturom podataka.
4. Koje su osnovne karakteristike ADT-a?
5. Definišite poiter (pokazivač).
6. Koliko elemenata sadrži niz **triD** definisan sa : int triD [10][3][5]? A koliko bitova, ako int zauzima 4 bajta?
7. Koliki je maksimalan i minimalan indeks niza deklarisanog sa: char niz [100]?
8. Koje podatke o listi (linked list) najčešće moramo da imamo?
9. Koje su najčešće operacije nad elementima liste?
10. Navedite način na koje se liste mogu implementirati?
11. Kakva je prednost dvostruko povezanih lista u odnosu na obične (jednostruko povezane) liste?
12. Definišite cirkularnu listu?
13. Opišite FIFO i LIFO principe.
14. Koje podatke o steku najčešće moramo da imamo?
15. Koje su najčešće operacije nad stekom?
16. Navedite bar dve primene steka.
17. Koje podatke o redovima najčešće moramo da imamo?
18. Koje su najčešće operacije nad redovima?
19. Navedite bar dve primene redova.
20. Nacrtajte primer nekog binarnog stabla i označite koren (root) i lišće (leafs) stabla.
21. Na primeru nekog stabla (nacrtati stablo i njegove čvorove označiti slovima a,b,c,d,...) ispisati redosled posete čvorova za:
    A. preorder
    B. inorder
    C. postorder

    prolaske kroz stablo.
22. Šta je sortno binarno stablo?
23. Definisati hešing postupak za memorisanje podataka.
24. Na kojoj adresi će se naći podatak 248 ako se za hešing koristi sledeća transformacija ključa u adresu:
    a) adresa = (kljuc) MOD 300 + 1
    b) kvadriranje ključa i uzimanje prve tri cifre.