Programski jezici i strukture podataka

11

RED

Red

- Kod steka poruke se čitaju obrnutim redosledom u odnosu na redosled dolaska.
- Ako upotrebimo red umesto steka, poruke mogu da se čitju onim redom kojim su primane.
- Baferi koji omogućavaju rad sa uređajima kod kojih se brzine transfera podataka jako razlikuju, kao što su CPU i diskovi, obično se implementiraju u obliku redova.

RED (QUEUE)

- Red je po svojoj organizaciji implementira situaciju u kojoj više korisnika čeka servis sa jednog izvora: red automobila na benzinskoj pumpi, red klijenata pred šalterom...
- Red je linearna struktura podataka čiji se terminal gde se vrši dodavanje novog elementa naziva **kraj reda**, a suprotni terminal je **početak reda** tamo se obavljaju operacije pristupa i uklanjanja elementa.

Operacije nad redom

- **FRONT(prvi)**, pristupa elementu na početku reda i vraća njegovu vrednost bez destrukcije sadržaja reda.
- **OUTQUEUE(iz_reda)**, pristupa elementu na početku reda i vraća njegovu vrednost uklanjajući ga iz reda. Operacija je destruktivna jer se njenom primenom smanjuje broj elemenata reda pri čemu element koji je bio drugi postaje element na početku reda.
- **INQUEUE(u_red)**, smešta na kraj reda novi element pri čemu do tada poslednji element postaje pretposlednji. Operacija je destruktivna jer se njenom primenom povećava broj elemenata reda pri čemu se mora voditi računa o memorijskom prostoru koji je dodeljen redu.

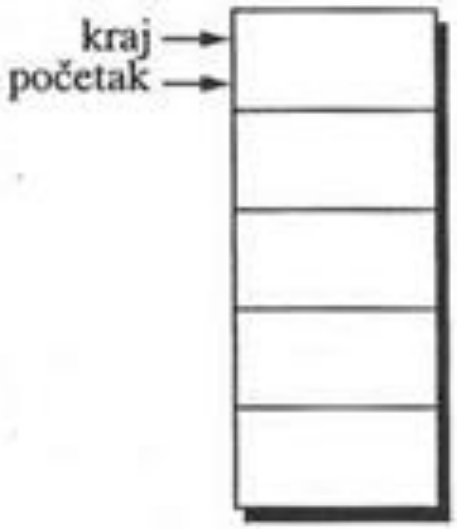
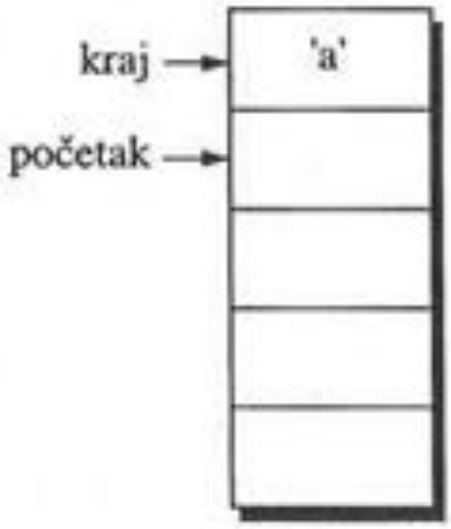
Tip podataka - red

- Red je mehanizam smeštanja u memoriju tipa prvi-u, prvi-iz (na engleskom "first-in, first-out" - FIFO).
- Element koji je prvi smešten u red, prvi se uzima iz reda. Red možemo implementirati pomoću niza znakova isto kao kod steka.
- Indeks početka pokazuje gde treba da se obavi sledeca operacija upiši
- Indeks kraja pokazuje odakle će se obaviti sledeća operacija pročitaj.

Tip podataka - red

- Za manipulaciju redom se realizuju dve funkcije **ured**() za upis elementa u red i **izreda**() za čitanje elementa iz reda.
- Kada se pokuša upis u red, proverava se najpre indeks početka.
- Ako lokacija na koju indeks početka ukazuje nije iza kraja reda, element se upisuje na tu lokaciju, zatim se inkrementira početak.
- Ako je početak već iza kraja reda, ništa ne može da se upiše.
- Na element koji se čita ukazuje indeks kraja.
- Ako je indeks kraja jednak indeksu početka, red je prazan i nema šta da se pročita.
- U suprotnom, obavlja se operacija pročitaj i indeks kraja se inkrementira.

RED (prvi u, prvi iz)

Prazan red	Posle ured(a)	Posle izreda()
		

Logička struktura **reda**

F=(S(F),r(F))

pri čemu se skup elemenata može urediti tako da da bude predstavljen sa:

$$S(F)=\{x_1,x_2,\ldots,x_n\}$$

realizacija **r** definiše se sa:

$$r(F) = \begin{cases} \{(x_i, x_{i+1}) \mid i = 1,\ldots,n-1\} & \text{za } n > 1 \\ 0 & \text{za } n \leq 1 \end{cases}$$

Za element x_1 se kaže da je na početku reda, a za element x_n da je na kraju reda.

Logička struktura reda

Da bi se ostvarila logička struktura reda potrebno je definisati dve primitivne funkcije:

1. Funkcija **prvi** koja daje element na početku (dakle x_1).
2. Funkcija **poslednji** za izdvajanje elementa x_n na kraju reda.

Izvedene, operacije nad redom:

1. provera da li je red prazan
2. uklanjanje svih elemenata
3. određivanje broja elemenata

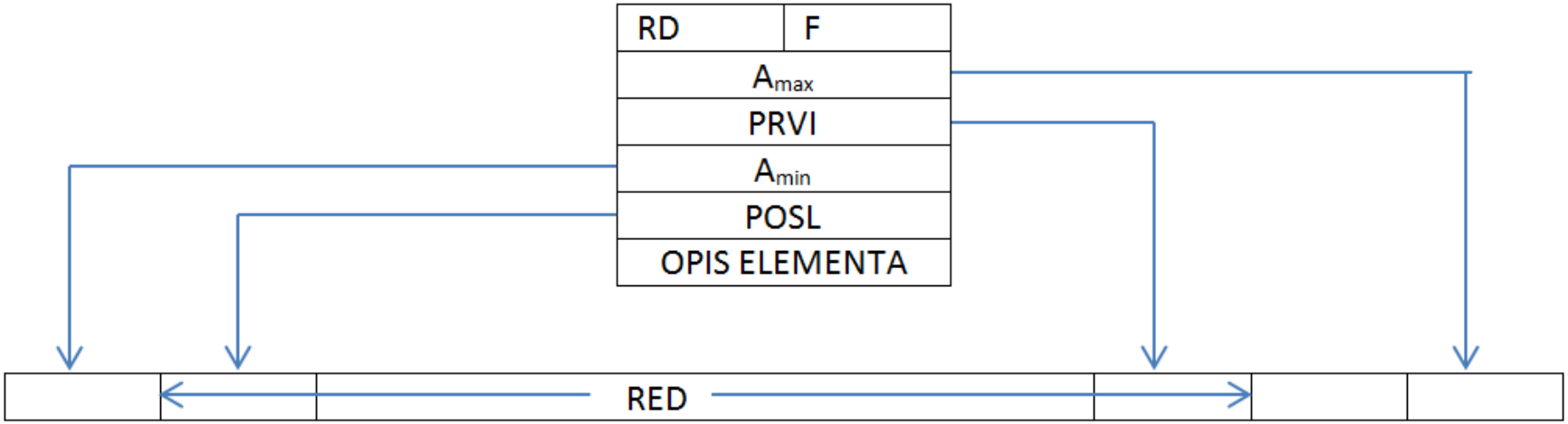
Operacije se realizuju na sličan način kao kod steka.

- Funkcija **poslednji**, sama po sebi nije kompleksna, takođe fizička realizacija je jednostavna.
- Problem sa ovom funkcijom se odnosi na samu definiciju reda, naime,red je definisan kao struktura podataka kod koje se pristup ostvaruje samo na njegovom početku.
- Nasuprot tome, funkcija **rear** nije ništa drugo no pristup elementu na kraju strukture i kao takva nije primenljiva na red.
- Daljom analizom vidimo da ova operacija ne koristi niti jednu drugu osnovnu operaciju (koristi samo primitivnu funkciju) i samim time je osnovna, što usložnjava analizu prava njene egzistencije.
- No, rešenje postoji, i vezano je za kontekst. Odnosno ako je kontekst u kojem se pojavljuje struktura takav da se predviđa pristup na oba kraja, onda to treba i omogućiti, uz ogradu da se u tom slučaju ne radi o redu već o strukturi koja je slična redu (ali i steku).

Fizička struktura reda

- Kao i kod steka, realizacija je sekvencijalna ili spregnuta.
- Sekvencijalna fizička realizacija podrazumeva postojanje deskriptora sličnog deskriptoru steka, s tom razlikom da ovde postoji osim adrese prvog, i adresa poslednjeg koja je takođe promenljiva.
- Putem ove dve adrese se definišu dve primitivne funkcije prvi i poslednji.

Sekvencijalna fizička realizacija reda

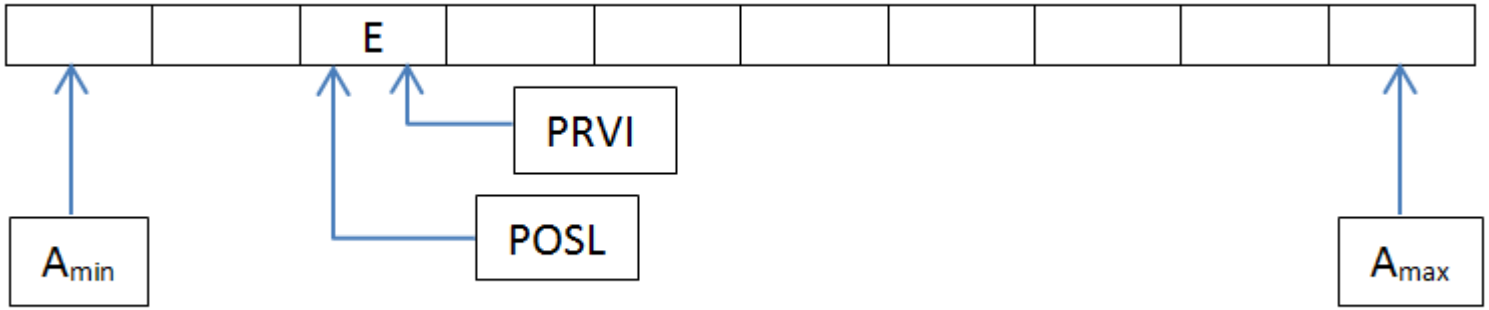


Deskriptor reda obuhvata:

- indikator strukture reda, RD
- ime reda, F
- najveću adresu memorijskog prostora, A_{max}
- adresu početka reda, PRVI
- najmanju adresu memorijskog prostora, A_{min}
- adresu elementa na kraju reda, POSL
- opis elemenata reda.

- Na osnovu prethodnog prikaza se može primetiti da se fizička struktura **steka**, može smatrati **posebnim** slučajem fizičke strukture **reda** kod kojeg je POSL=A_{min}=const.
- Sekvencijalna struktura reda karakteristična je po jednoj pojavi: Stanje prepunjenosti reda čiji memorijski prostor nije do kraja zauzet.

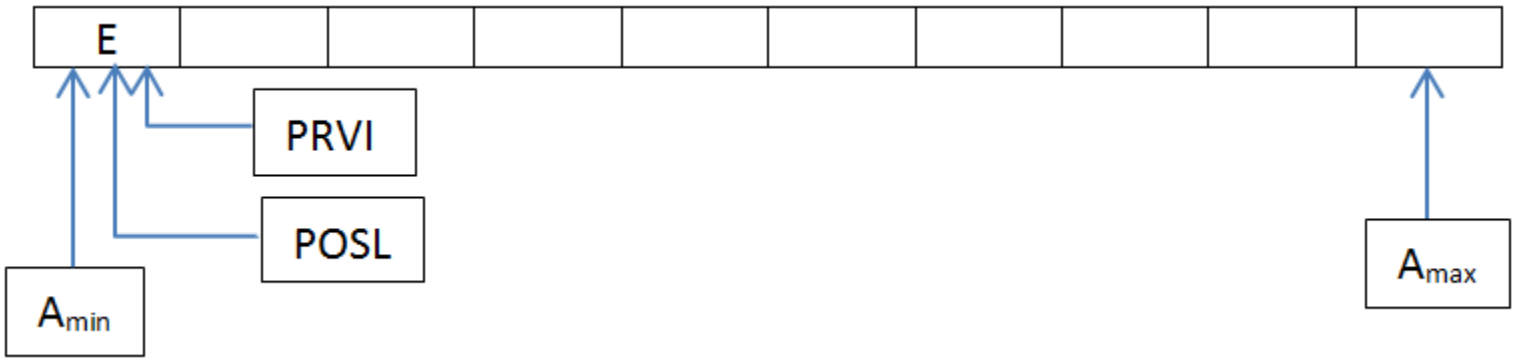
Posmatrajmo red R sa jednim elementom koji je smešten u memorijski prostor koji može da primi 10 elemenata, neka je stanje reda u nekom trenutku sledeće:



Nakon primene sledeće sekvence:

InQueue(F,a), OutQueue(F), InQueue(F,b), OutQueue(F),

stanje reda će biti:



- Kao što se može videti prva primena operacije **InQueue** izazvaće prepunjenost reda, iako u njemu postoji samo jedan element.
- Zapravo ova ilustracija opisuje sledeći paradoks: usled promenljivosti obe granične adrese red kao da se pomera po memoriji ka minimalnoj adresi i kada je dostigne ulazi u stanje prepunjenosti, ne zato što nema mesta za upis novog elementa, nego zato što nema mesta za pomeranje reda.

Da bi se prevazišao paradoks, potrebno je obezbediti mehanizam za pomeranje reda, ovo se postiže cirkularnom strukturom.

Neka je l(F) broj memorijskih lokacija određen za smeštanje jednog elementa.

Prilikom realizacije operacije InQueue adresa za upis novog elementa izračunava se na sledeći način:

$$\text{POSL}=\begin{cases} POSL - l(F) & POSL > Amin \\ Amax & POSL = Amin \end{cases}$$

odnosno, ako je najniža aresa popunjena, novi element se kružno upisuje na najvišu adresu Amax.

Slično prilikom uklanjanja elementa pomoću operacije OutQueue adresa se preračunava na sledeći način:

$$\text{PRVI}=\begin{cases} PRVI - l(F) & PRVI > Amin \\ Amax & PRVI = Amin \end{cases}$$

Na taj način, sukcesivna primena operacija InQueue i OutQueue izaziva svojevrsno kruženje reda po memorijskom prostoru bez opasnosti da dođe do lažne prepunjenosti, izuzev slučaja kada je memorija zaista popunjena.

Tehnika cirkularne realizacije stvara novi negativni efekat, koji se mora uzeti u obzir.

Definišimo funkciju:

$$\text{prethodni}(x,F)=\begin{cases} x-l(F) & x > Amin \\ Amax & x = Amin \end{cases}$$

gde je x element reda F.

Pretpostavimo da red ima jedan jedini element, te operacija OutQueue može da se izvrši još samo jednom.

Neposredno pre izvršavanja je PRVI=POSLEDNJI, tako da će posle toga red biti prazan i važiće:

prethodni(POSL,F)=PRVI

Lako je uočiti da ista realizacija važi i za slučaj da su elementi reda poređani u redosledu (po rastućim adresama)

POSL,..., Amax,Amin,...,PRVI,

a to je očigledno red koji je pun.

Odavde sledi da ako se cirkularna tehnika primeni bez dodatnih mehanizama, nema mogućnosti da se utvrdi da li je memorijski prostor potpuno zauzet ili potpuno slobodan.

Jedan od načina za prevazilaženje ovog problema jeste mehanizam kojim se lokacija koja je iza (u smislu funkcije prethodni) aktuelnog poslednjeg tretira drugačije od ostalih i to tako što se u nju ne upisuje element.

Između adrese te lokacije (nazovimo je PRAZNA) i adrese kraja reda po definiciji se uspostavlja veza:

```
prethodni(POSL,F)=PRAZNA
```

Sada je uslov za prazan red da bude:

$$PRVI=PRAZNA$$

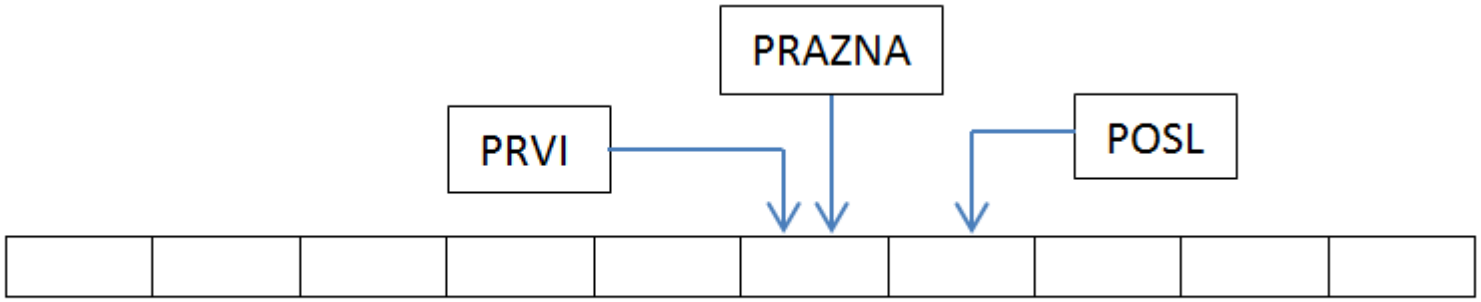
ili posredstvom funkcije prethodni:

$$prethodni(POSL,F)=PRAZNA$$

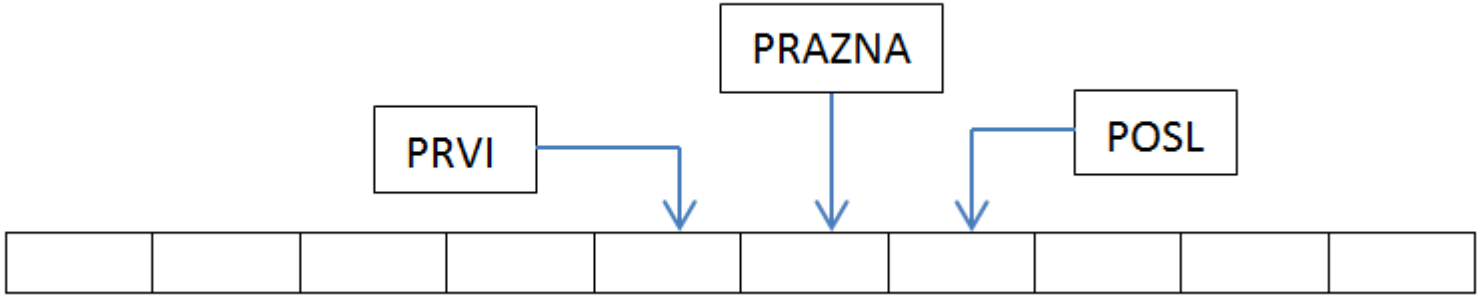
a za pun red:

$$prethodni(PRAZNA,F)=PRVI$$

Prazan red:



Pun red:



Kružni red

- Linearni redovi ograničeni su dužinom reda. Zbog toga se ne koriste često.
- Jedna varijacija linearnih redova - kružni red koristi se često, naročito za baferovanje pri prenosu podataka.
- Kružni red je takođe pravolinijski deo memorije, ali se prilikom pristupa vraćamo nazad na početak kada se dosegne kraj reda.

Kružni red

- Dobar primer kružnog reda je bafer za tastaturu personalnog računara.
- Bafer tastature prima znakove brzinom kojom ih kucate, a procesu ih predaje na zahtev.
- Ako je proces zauzet nekim drugim poslom, ovi baferi omogućuju vam da nastavite sa kucanjem, a da se ništa od toga što otkucate ne izgubi.

Kružni red

- Ako su indeksi početka i kraja jednaki, da li je red pun ili prazan? To se ne može znati.
- Za razlikovanje ove dve situacije žrtvujemo jedno mesto u redu.
- Indeks kraja se inicijalizuje na jedno mesto iza indeksa početka i nije mu dozvoljeno da ga stigne. (**z41.c**)

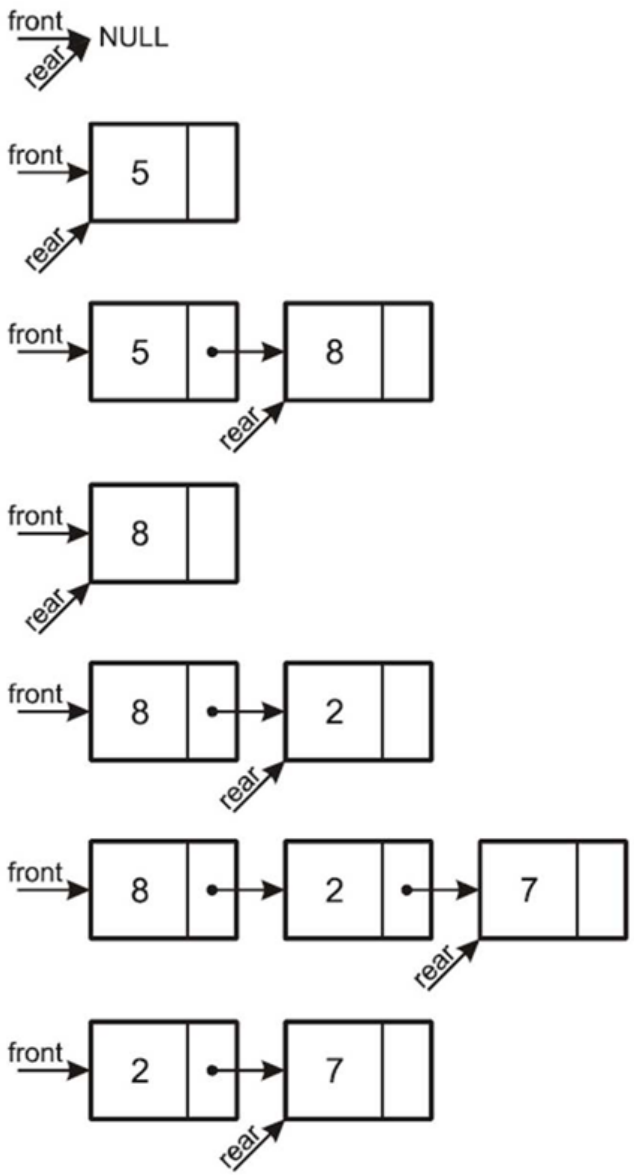
Spregnuta realizacija reda

- Red se može veoma efikasno implementirati korišćenjem spregnutih lista tako što bi se novi element dodavao uvek na kraj liste.
- Takode, u slučaju brisanja elementa iz reda skidao bi se uvek prvi element u listi, odnosno onaj koji je prvi dodat.

Spregnuta realizacija reda

- Na početku, lista je prazna, pa su i pokazivači na početak i kraj liste *(**front*** i ***rear**)* jednaki NULL.
- Funkcija ***insert*** kreira novi element i dodaje ga na kraj liste.
- Funkcija ***delete*** vraća vrednost prvog elementa u listi i pomera pokazivač *front* na sledeći element u listi.
- Na kraju se uništava element koji je bio na početku liste.

- Prazan red
- Insert(5)
- Insert(8)
- Delete()
- Insert(2)
- Insert(7)
- Delete()



Z42.c Spregnuta realizacija reda

DEK

DEK (DEQUE)

- Ime je akronim od Double Ended Queue, predstavlja uopštenje steka i reda u smislu načina pristupa, dodavanja i uklanjanja elemenata.
- Dek zapravo predstavlja, po osobinama, sintezu osobina reda i steka.

DEK (DEQUE)

- Struktura je linearna
- Pristup, uklanjanje i dodavanje dozvoljeno na oba kraja.
- Nema prvog niti poslednjeg, ravnopravni su, stoga govorimo o krajnjem desnom i krajnjem levom elementu.

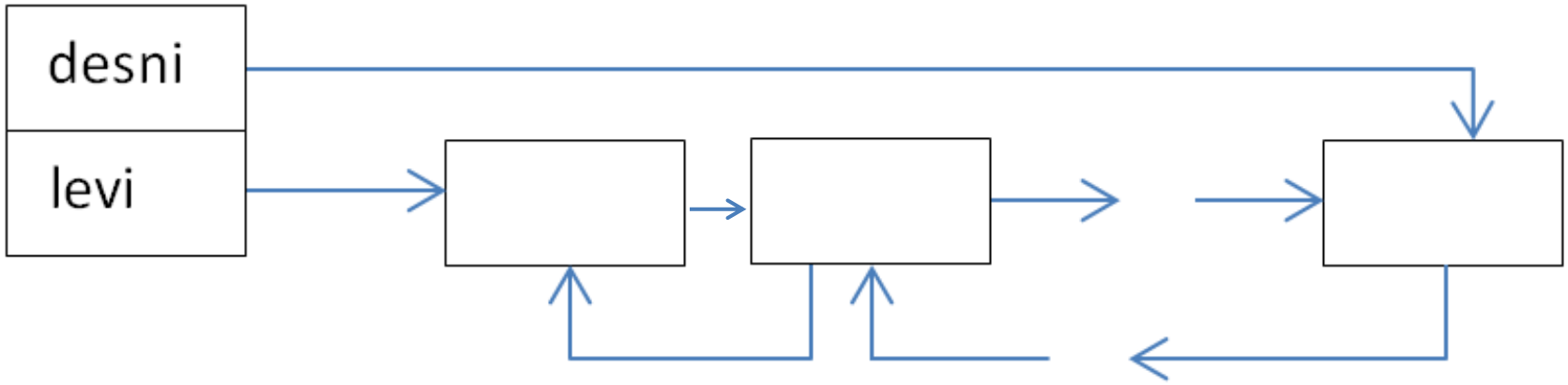
DEK(DEQUE)

- Struktura podataka klase dek nadograđuje strukturu reda mogućnošću obavljanja svake operacije na oba kraja deka.
- Time se dovodi u pitanje gde je početak, a gde kraj. Nad dekom su definisane operacije:
  - pristup elementu na levom, odnosno desnom kraju deka,
  - dodavanja elementa na levom, odnosno desnom kraju deka,
  - uklanjanja elementa sa levog, odnosno desnog kraja deka.

Fizička realizacija deka

- Sekvencijalna realizacija slična realizaciji reda, javlja se pseudopopunjenost koja se rešava cikličnom strukturom kao kod reda.
- Spregnuta (potrebno dvostruko sprezanje)

- Spregnuta realizacija je drugačija u odnosu na spregnutu realizaciju reda, zbog neophodnosti pristupanja, dodavanja i uklanjanja elemenata na oba kraja.
- Prva razlika je u deskriptoru, on mora da obezbedi pokazivače na levi i desni.
- Druga razlika je u sprezanju, mora postojati propagacija svesnosti u oba smera, to podrazumeva dvostruko sprezanje.



STABLO

NELINEARNE STRUKTURE

Stablo

- Stablo je nelinearna dinamička struktura podataka gde svaki čvor ima jednog i samo jednog prethodnika (u svetu stabla se naziva nadređeni), sem najvišeg čvora koji nema nadređenog i naziva se koren stabla, odnosno nema ili ima jednog ili više sledbenika (podređenih čvorova).
- Čvorovi stabla koji nemaju podređenih čvorova nazivaju se listovi stabla.
- Naziv ove strukture ilustruje njen oblik koji je pogodan za čuvanje pojava entiteta koji su po svojoj prirodi hijerarhijski, dekompozicioni ili analitičko-sintetički.

- Stablo je nelinearna struktura podataka koja predstavlja hijerarhijski odnos između elemenata.
- Primeri strukture stabla:
  - izgled sadržaja neke knjige,
  - rodoslovi familija,
  - organizacione šeme.
- Stabla se u raznim pojavnim oblicima koriste u računarstvu:
  - operativni sistemi organizuju sistem datoteka i foldera na način koji odgovara strukturi stabla
  - predstavljanje izraza u programima prilikom njihovog prevodenja od strane prevodilaca
  - sintaksne kategorije se kod prevodilaca interno predstavljaju u obliku stabla izraza
- Indirektna primena stabla je za predstavljanje drugih struktura podataka:
  - disjunktni skupovi
  - redovi sa prioritetima

Formalna definicija korenskog stabla

Stablo T je skup čvorova i skup grana izmedu
njih (tj. uređenih parova čvorova) sa sledećim osobinama:

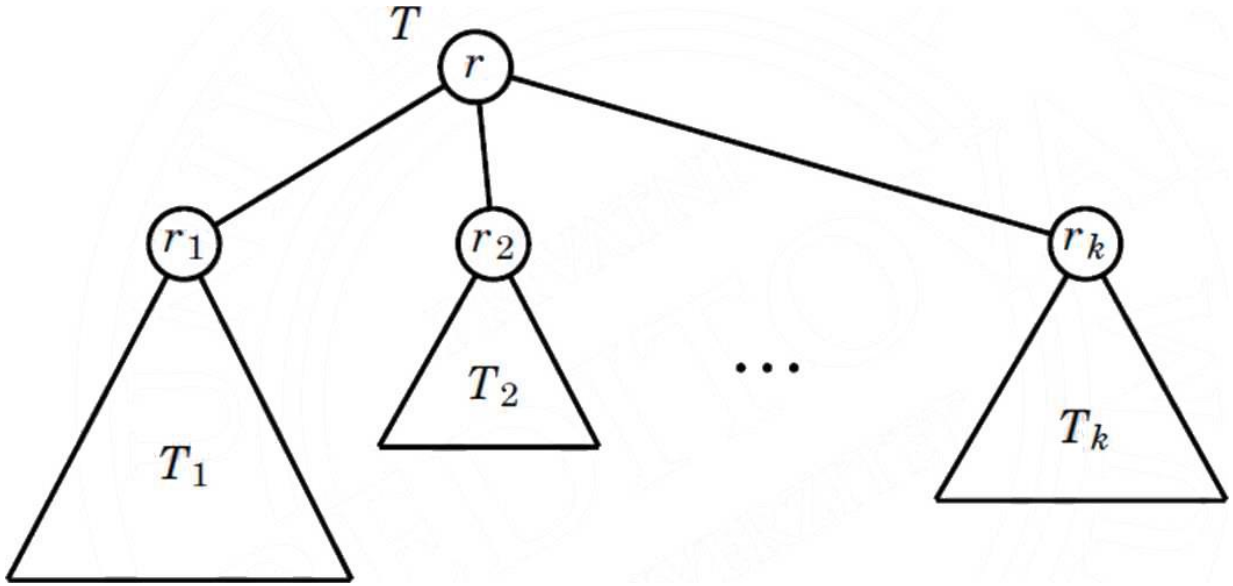
1. Ako je **T** neprazno stablo, ono ima jedinstven poseban čvor (koren) koji nema roditeljski čvor.
2. Svaki čvor **u** u**T** osim korena ima jedinstven roditeljski čvor **v**.

Na osnovu ove formalne definicije stablo može biti prazno, što znači da takvo stablo nema nijedan čvor (i nijednu granu).

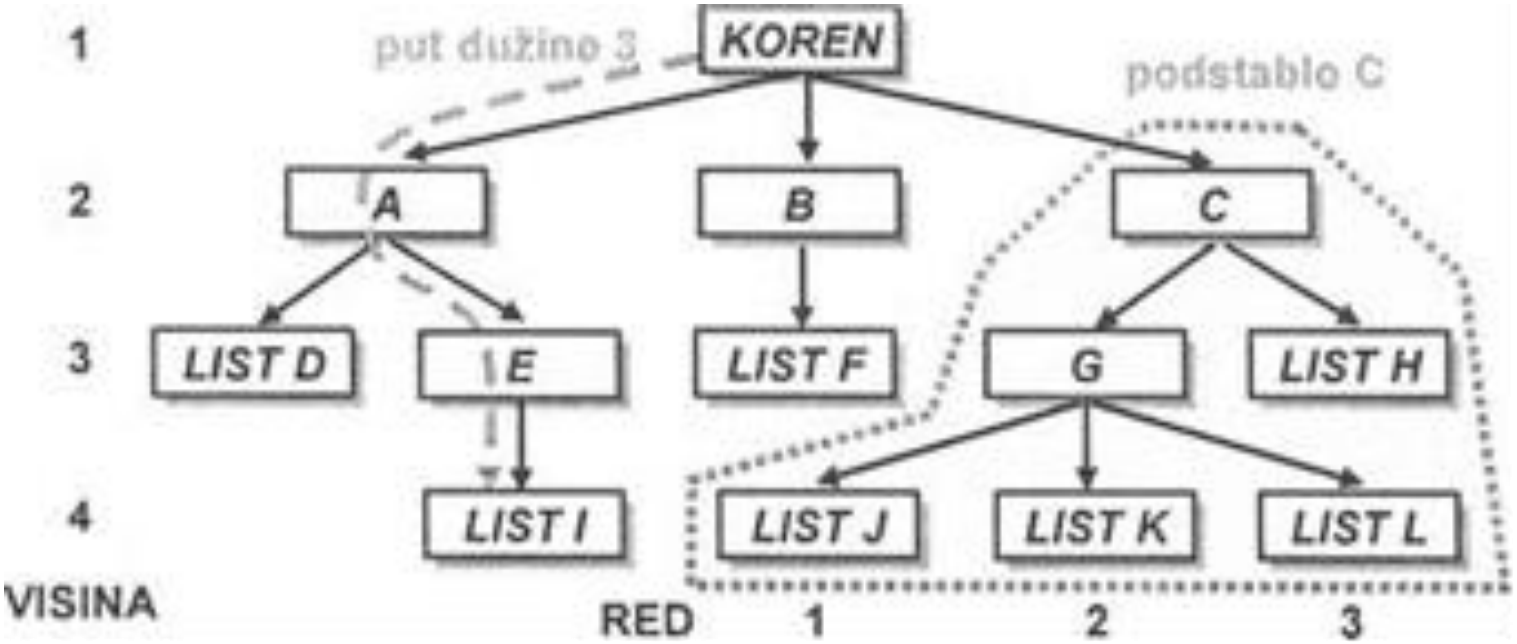
Rekurzivna definicija korenskog stabla

Na osnovu prethodnog stablo se definiše i rekurzivno na način kojim se veća stabla konstruišu od manjih:

Stablo T je prazno ili se sastoji od čvora r, koji se naziva koren stabla *T,* i (moguće praznog) skupa stabala $T_1,T_2,\ldots,T_k$ čiji su koreni deca čvora r.



Uopšteno stablo



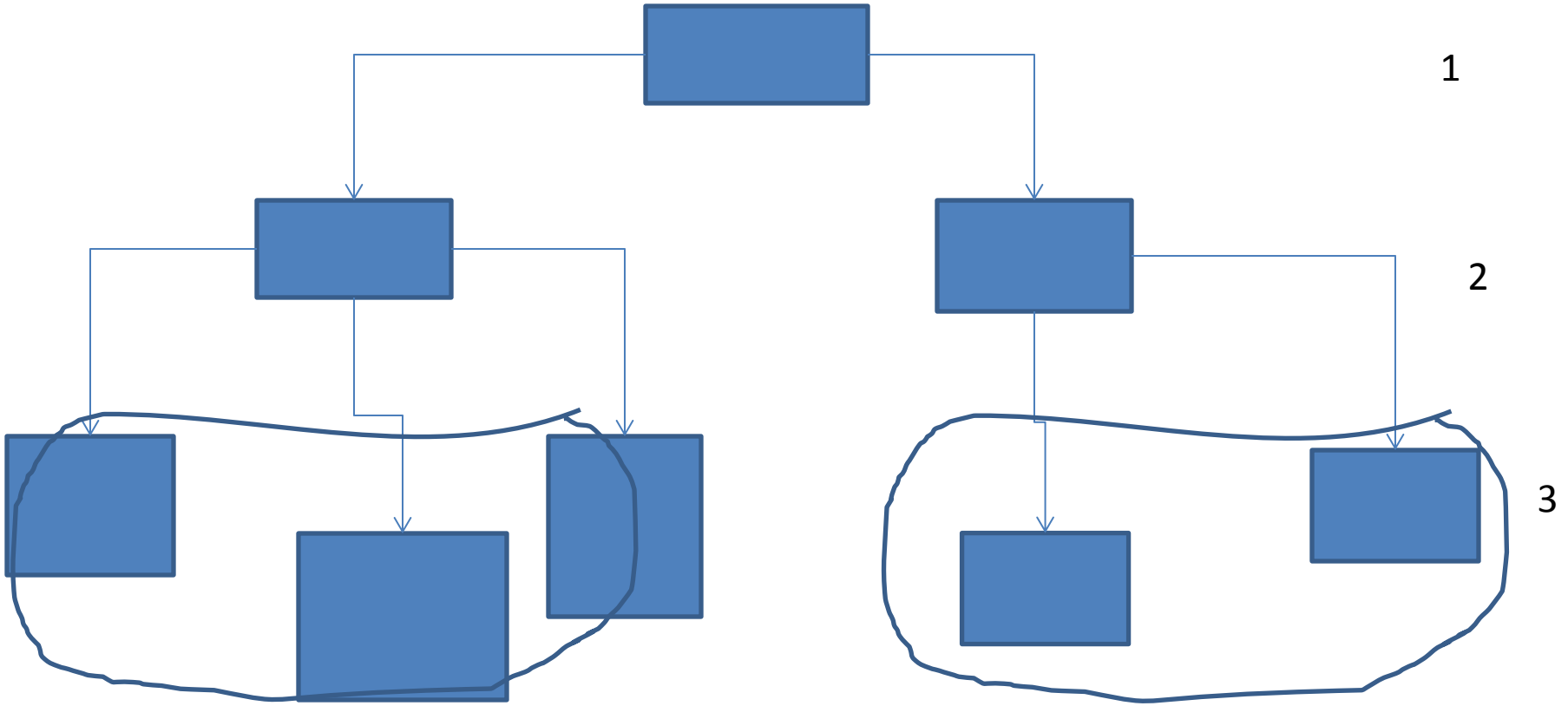
Stablo

- Stablo obuhvata sve strukture podataka kojima pripada digraf tipa stablo.
- Ključna reč koja opisuje stablo je hijerarhija.
- Elementi stabla su u odnosu:
  - podređeni
  - nadređeni
- Skup čvorova koji su povezani granama se naziva **lanac**.
- Orjentisani lanac se naziva put.
- Ako su svaka dva čvora povezana jednim lancem kažemo da je digraf **slabo povezan**.

Orjentisano stablo

- Postoji tačno jedan čvor u koji ne ulazi ni jedna grana (ulazni stepen je nula).
- U sve ostale čvorove ulazi tačno po jedna grana (ulazni stepen je jedan).
- Digraf je povezan, odnosno svaka dva čvora mogu se spojiti nizom uzastopnih grana ne uzimajući u obzir njihovu orjentaciju (digraf je slabo povezan).
- Visina stabla je jednaka je broju čvorova na najdužem putu u stablu.

- **Kompletno stablo** (reda n): svi elementi osim listova imaju izlazni stepen n.
- **Puno stablo**: svi putevi od korena do listova su iste dužine.
- Svaki element strukture je dostupan preko korena.
- **Balansirano stablo**: ono stablo u kojem se broj elemenata podstabla na istom hijerarhijskom nivou razlikuje najviše za 1.



Balansirano stablo

Struktura tipa stablo

- Digraf je tipa stablo
- Dozvoljen pristup svakom elementu
- Dodavanje i uklanjanje u domenu dozvoljenog (n-arno stablo, uopšteno-generalizovano stablo)

Struktura tipa stablo

- Slično kao u slučaju osnovnih struktura podataka (nizova, listi, stekova, redova), da bismo stabla koristili u programima za organizaciju kolekcije podataka, moramo imati način za predstavljanje stabala u računaru i implementirati skup korisnih operacija nad stablima.

- U radu sa stablima se može pojaviti potreba za vrlo raznolikim operacijama:
  - dodavanje,
  - uklanjanje
  - odredivanje deteta datog čvora,
  - operacije koje kao rezultat daju roditelja,
  - sasvim levog deteta
  - desnog brata datog čvora...

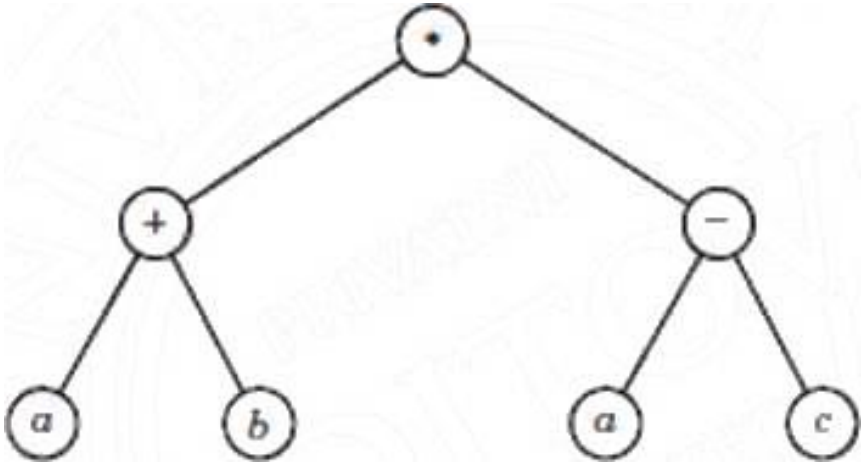
- Radi lakšeg programiranja korisno bi bilo realizovati i upitne operacije koje određuju da li je dato stablo prazno, ili da li je dati čvor koren, interni, eksterni i slično.
- Takođe i pomoćne operacije koje obuhvataju:
  - postupke konstruisanja novog stabla,
  - određivanja broja čvorova (veličine) stabla,
  - odredivanje visine stabla i tako dalje.
- Međutim, ukoliko se koristi odgovarajuća reprezentacija stabla, sve ove operacije se mogu relativno lako implementirati

- Stablo predstavlja kolekciju podataka tako što se pojedinačni podaci u kolekciji nalaze u čvorovima stabla.
- stablo na slici predstavija aritmetički izraz:

$$(a + b) * (a - c)$$

- koren tog stabla sadrži operator * za množenje, a njegova deca su koreni dva podstabla koja predstavijaju podizraze:

$a + b$ i $a - c$

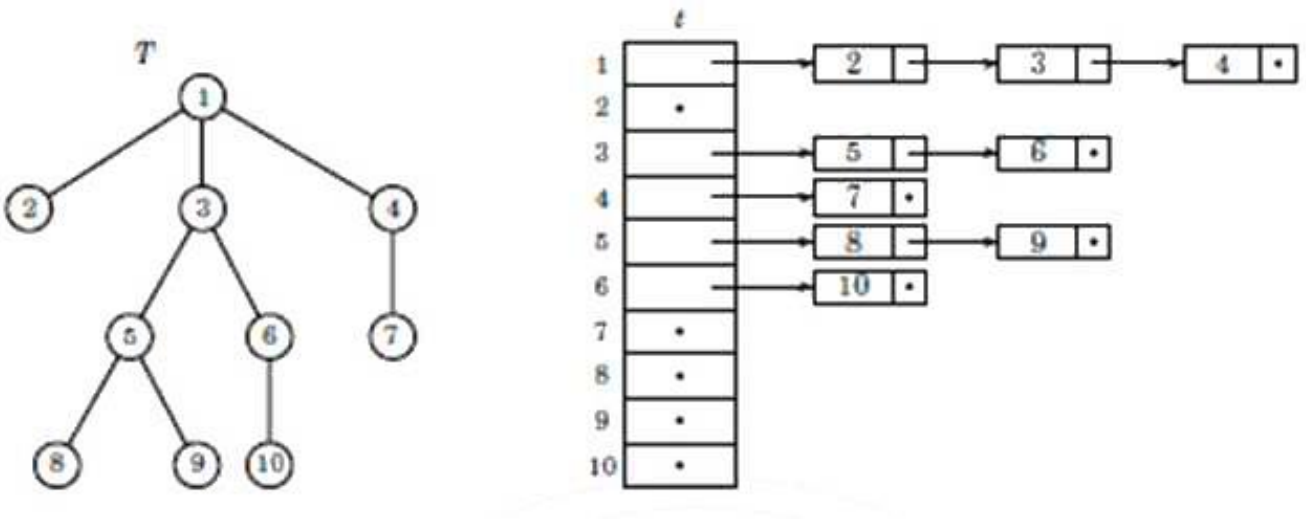


- Slično kao kod osnovnih struktura podataka, pojedinačni podaci u čvorovima stabla se na opštem nivou poistovećuju sa jednoznačnom vrednošću koja se naziva njihov *ključ*.
- Prirodna reprezentacija stabla bila bi da se svaki čvor stabla predstavlja:
  - objektom koji obuhvata polje ključa.
  - više polja u kojima se nalaze pokazivači na svako dete tog čvora i
  - polje za pokazivač na roditelja čvora.

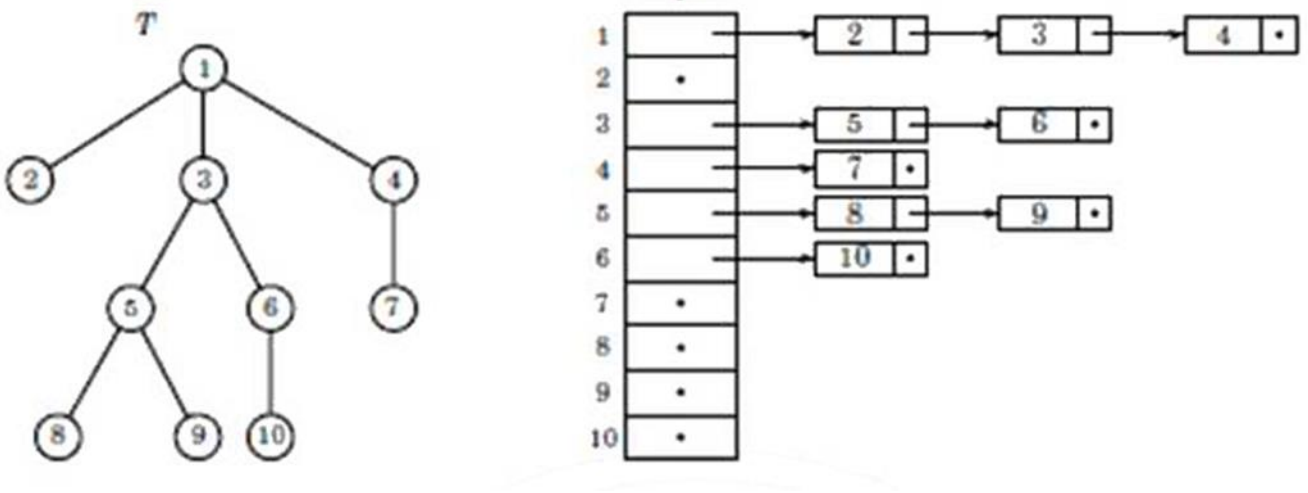
- Reč je o opštim stablima, odnosno broj dece po čvoru može biti vrlo različit i nije unapred ograničen, ovaj pristup nije najbolji jer proizvodi loše iskorišćenje memorijskog prostora.
- Zato se obično koriste dva druga načina za predstavijanje opštih stabala:
  - predstavljanje stabla listama dece čvorova
  - predstavljanje stabla pomoću sasvim levog deteta i desnog brata čvorova

Predstavljanje stabla listama dece čvorova

- Jedan način za prevazilaženje problema nepoznatog broja dece nekog čvora u opštem stablu je formiranje povezane liste dece svakog čvora.
- Može se lako generalizovati za predstavljanje složenijih struktura kao što su grafovi.



- Imamo jedan niz pokazivača ***t*** čiji elementi predstavljaju čvorove i zato je njegova veličina jednaka ukupnom broju čvorova u stablu.
- Pored toga, svaki element niza ***t*** pokazuje na početak povezane liste čvorova stabla.
- Pri tome, elementi liste na koju pokazuje element niza t_i su čvorovi-deca čvora ***i*** tako da poredak elemenata u listi odgovara poretku dece sleva na desno u stablu.
- Na primer, element t_5 pokazuje na listu 8->9 pošto su čvorovi 8 i 9 redom deca čvora 5.

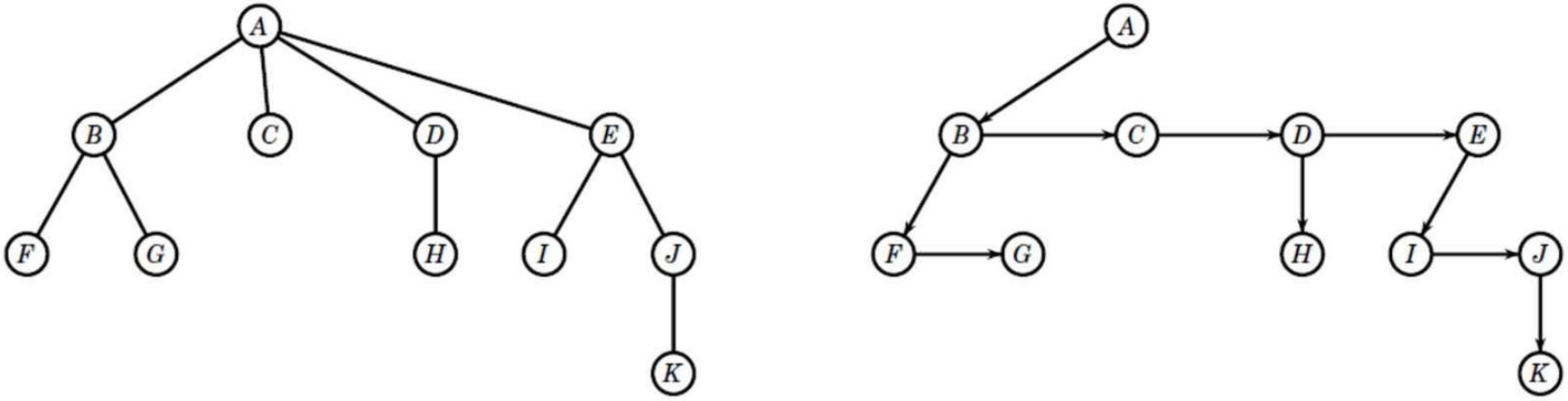


- Jedan nedostatak ovog načina predstavljanja stabla je to što ne možemo lako da konstruišemo veća stabla od manjih.
- To je posledica toga što svako stablo ima svoj niz pokazivača za svoje čvorove.
- Da bismo napravili novo stablo, recimo, od datog čvora kao korena i dva data podstabla T_1 i T_2, morali bismo da kopiramo T_1 i T_2 u treće stablo i da dodamo listu za novi koren čija su deca koreni za T_1 i T_2.

Predstavljanje stabla pomoću sasvim levog deteta i desnog brata čvorova

- Drugo rešenje za problem predstavljanja opštih stabala je *predstavljanje sasvim levog deteta i desnog brata* svakog čvora stabla.
- U ovoj reprezentaciji svaki čvor pored ključa sadrži samo još dva pokazivača:
  - jedan pokazuje na njegovog sasvim levog deteta
  - drugi na njegovog desnog brata
- Ovi pokazivači imaju specijalnu vrednost **null** ukoliko čvor nema dece ili nema desnog brata.

- Na slici je za svaki čvor stabla strelicom nadole označen pokazivač na sasvim levog deteta, a strelicom nadesno je označen pokazivač na desnog brata.

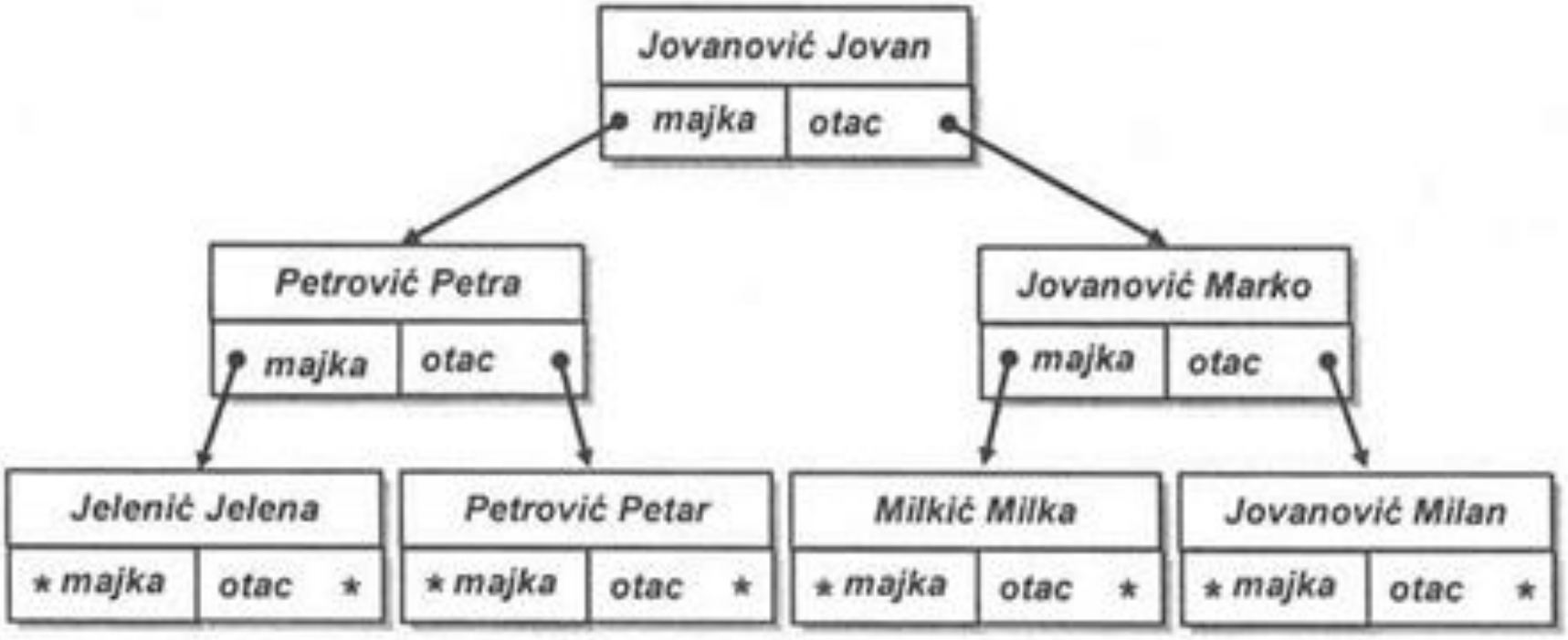


- Sve operacije nad stablima, osim određivanja roditelja datog čvora, mogu se efikasno realizovati ako je stablo predstavljeno pomoću sasvim levog deteta i desnog brata svih čvorova.
- Ako je potrebno i da operacija određivanja roditelja bude efikasna, to se može lako rešiti dodavanjem još jednog pokazivača svakom čvoru koji direktno pokazuje na njegovog roditelja.

n-arno stablo

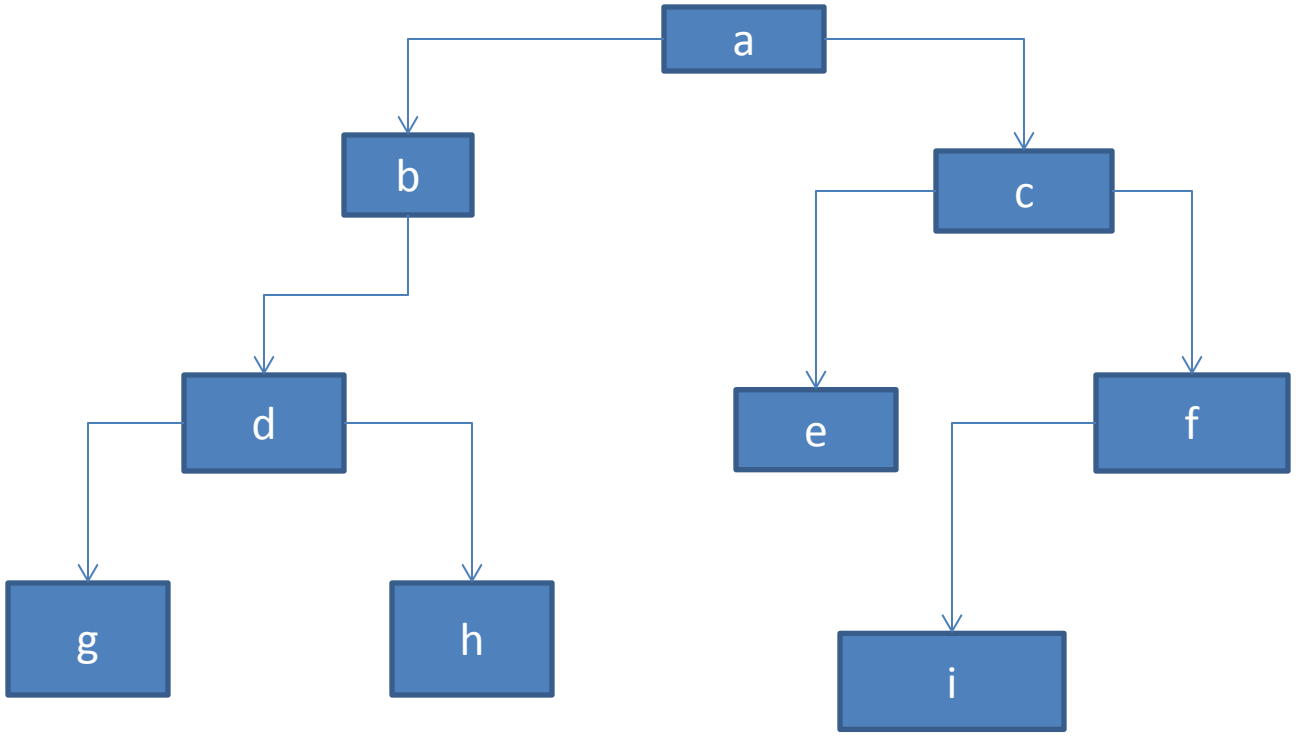
- je stablo reda n, **n** je propisan i ograničen
- u skupu podređenih svakog elementa postoji eksplicitno uređenje
- najčešće korištena su binarna stabla, n=2

N-arno (binarno) stablo



Obilazak binarnog stabla

- Obilazak s vrha ka dnu
  - pristupi nadređenom
  - pristupi levom podstablu
  - pristupi desnom podstablu
- Obilazak s leva u desno
  - pristupi levom podstablu
  - pristupi nadređenom
  - pristupi desnom podstablu
- Obilazak sa dna ka vrhu
  - pristupi levom podstablu
  - pristupi desnom podstablu
  - pristupi nadređenom
- Sve tri metode počinju od korena pošto se samo za njega zna gde je.



Obilazak s vrha ka dnu: a,b,d,g,h,c,e,f,i

Obilazak s leva u desno: g,d,h,b,a,e,c,i,f

Obilazak sa dna ka vrhu: g,h,d,b,e,i,f,c,a

Obilazak s vrha ka dnu
- pristupi nadređenom
- pristupi levom podstablu
- pristupi desnom podstablu

Obilazak s leva u desno
- pristupi levom podstablu
- pristupi nadređenom
- pristupi desnom podstablu

Obilazak sa dna ka vrhu
- pristupi levom podstablu
- pristupi desnom podstablu
- pristupi nadređenom