Stabla

-***Stabla*** su nelinearne strukture podataka

-Predstavljaju ***najvažnije nelinearne strukture*** koje se vrlo često koriste u računarstvu

-Odnos između elemenata ***nije linearan***

-Imaju razgranatu ili ***hijerarhijsku strukturu elemenata***

-Njihov naziv implicira vezu sa stablima (drvećem) u prirodi ili porodičnim stablima

-Većina terminologije potiče iz ovih izvora

Definicija 1:

-Struktura podataka B=(K,R); K-skup čvorova i R - binarna relacija prethođenja nad skupom K, predstavlja ***stablo*** ako R zadovoljava sledeće uslove:

(1) Postoji samo jedan čvor r, koga nazivamo koren, kome ne prethodi ni jedan drugi čvor

(2) Svaki čvor, izuzev korena, ima samo jednog prethodnika

(3) Za svaki čvor k, $k \neq r$, postoji niz čvorova $k_0, k_1, k_2, \ldots k_n = k$ ($n \geq 1$ i=1,n) koji predstavljaju listu, tj. (k_{i-1}, k_i) € R i=1,n

Definicija 2:

-Stablo se može definisati kao poseban oblik ***nelinearnog grafa***:

-***Linearni graf*** je skup čvorova i skup relacija (nazivaju se linije grafa) koji opisuju veze između čvorova. Linearni graf je povezan ako je svaki par čvorova u grafu povezan linijom.

-Stablo se onda može definisati kao ***povezan graf*** koji ne sadrži petlje, odnosno ako važi:

-Bilo koja dva čvora u stablu su povezana linijom

-Stablo sa n čvorova ima n-1 linija

Definicija 3:

-Stablo se može ***rekurzivno*** definisati kao:

-***Stablo*** sa korenom je konačan skup K od jednog ili više čvorova takvih da je:

(1) Postoji jedan specijalni čvor koji se naziva koren stabla

(2) Preostali čvorovi (isključujući koren) se mogu podeliti u $m \geq 0$ skupova $K_1, K_2, \ldots K_m$, čiji je presek prazvan skup, a koji svaki predstavlja stablo za sebe. Stabla $K_1, K_2, \ldots K_m$ se nazivaju podtsbla korena

-Ova rekurzivna definicija je veoma pogodna za predstavljanje stabala u memoriji računara i obavljanje operacija nad stablima

-***Slična stabla***: Za dva stabla se kaže da su slična ako imaju istu strukturu, tj. tačnije ako su oba prazna ili su sva njihova podstabla slična

-***Ekvivalentna stabla***: Dva stabla su ekvivalentna ako su slična i imaju identičan informacioni sadržaj u odgovarajućim čvorovima

Termini vezani za stabla

-***Terminalni čvor*** ili list je čvor koji nema podstabla

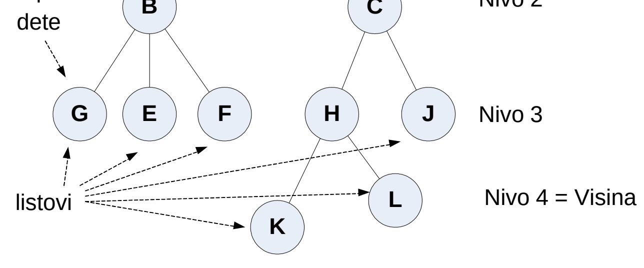
-Čvor d je ***dete*** čvoru k ako je d koren podstabla od čvora k. Čvor k se naziva ***roditelj***.

-***Stepen*** čvora je broj podstabala datog čvora

-***Šuma*** je skup stabala koja ne preklapaju

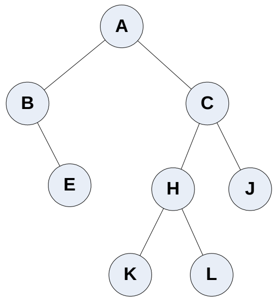
-***Nivo čvora*** je 1 ako je koren ili jednak broju čvorova koji se prodju na putu od korena do datog čvora.

-***Visina*** (ili dubina) stabla je maksimalni nivo na kome se nalazi neki čvor stabla

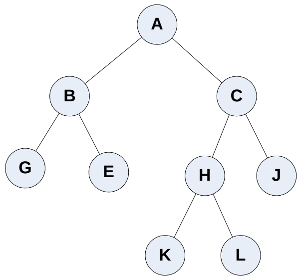


<u>*Binarna stabla*</u>

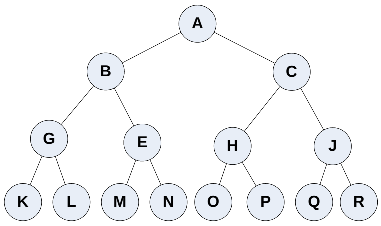
-***Binarno stablo*** se rekurzivno definiše kao konačan skup elemenata koji je ili prazan ili se sastoji od korena i dva binarna podstabla koja se ne preklapaju (tzv. levo i desno podstablo).



-***Striktno binarno stablo*** je binarno stablo kod koga svaki čvor nema nijedno podstabla ima tačno dva

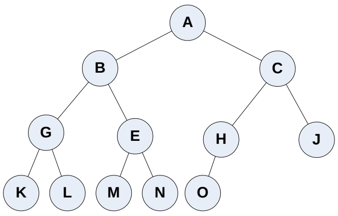


-***Kompletno binarno stablo*** je striktno binarno stablo kod koga su svi listovi na istom nivou



-*Broj čvorova* u kompletnom stablu je: $\mathbf{n = 2^h - 1}$, gde je h visina stabla

-***Skoro kompletno binarno stablo*** je binarno stablo za koga važi da su svi nivoi u stablu popunjeni, osim eventualno zadnjeg, i to tako da se popunjavanje vrši s leva u desno



Prolazi kroz stablo

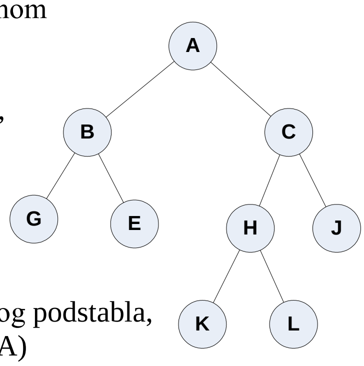
-Način kako se mogu obići čvorovi datog stabla a da se pri tome posete tačno jednom

-Tri su standardna načina

(1) **Prefiks prolaz (K-L-D)**: Prvo se poseti koren, zatim prefiks prolazom svi čvorovi levog podstabla, a zatim svi čvorovi desnog podstabla (A, B, G, E, C, H, K, L, J)

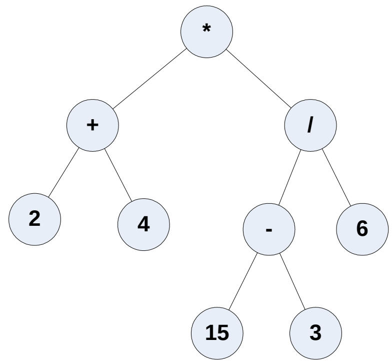
(2) **Infiks prolaz (L-K-D)**: Prvo se posete infiks prolazom svi čvorovi levog podstabla,
zatim se poseti koren, a zatim svi čvorovi desnog podstabla (G, B, E, A, K, H, L, C, J)

(3) **Postfiks prolaz (L-D-K)**: Prvo se posete postfiks prolazom svi čvorovi levog podstabla, zatim svi čvorovi desnog podstabla, a zatim se poseti koren (G, E, B, K, L, H, J, C, A)



Predstavljanje izraza preko stabla

-Matematički izrazi sa binarnim operatorima se mogu predstaviti preko striktnog binarnog stabla i to: listovi su operandi, unutrašnji čvorovi su binarni operatori (operacije).



Definicija stabla kao ATP

-Binarno stablo se može definisati kao **apstraktni tip** na sledeći način:

```
public interface binStablo {
    void ubaci(int a);
    void izbaci(int a);
    int visina();
    void prefiks();
    void postfiks();
    void infiks();
}
```

-Binarna stablo se mogu **implementirati na dva načina**:

(1) ***Dinamička implementacija*** (preko dinamičko spregnute strukture)
- -Vrlo fleksibilna implementacija, lako dodavanje i izbacivanje
- -Nema ograničenja na broj čvorova i broj nivoa stabla
- -Moguća primena za sve slučajeve i oblike binarnih stabala

(2) ***Implementacija preko niza***
- -Efikasna implementacija moguća samo za specijalni slučaj – skoro kompletno binarno stablo
- -Broj čvorova i nivo stabla ograničen dimenzijom niza

Statička implementacija

-Efikasna implementacija preko niza moguća samo za skoro kompletno binarno stablo (SKBS)

-Kod SKBS je moguće numerisati čvorove da odgovaraju indeksima u nizu i to tako da za svaki čvor važi:

*-Indeks levog deteta = Indeks * 2*

*-Indeks desnog deteta = Indeks * 2 + 1*

-Indeks roditelja = Indeks / 2

Knutova transformacija

-Transformiše šumu stabala u jedno binarno stablo

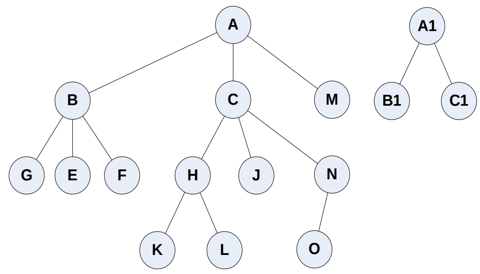
-Koraci:

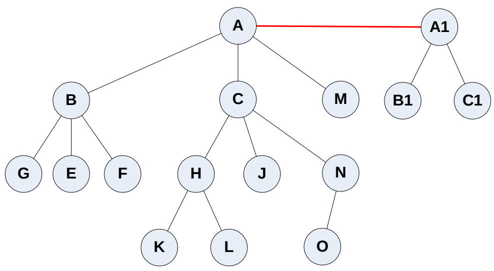
(1) Povežu se koreni svih stabala

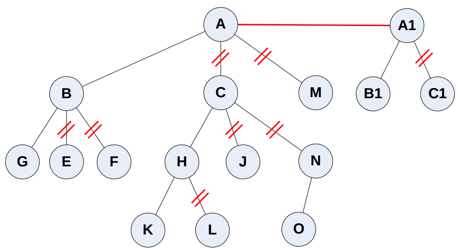
(2) Prekinu se sve veze između roditelja i dece izuzev krajnje leve veze

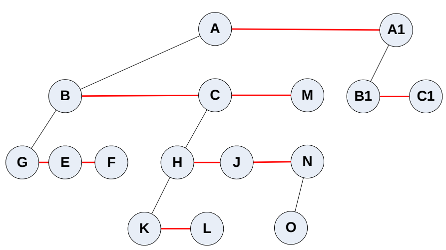
(3) Povežu se sva deca istog roditelja

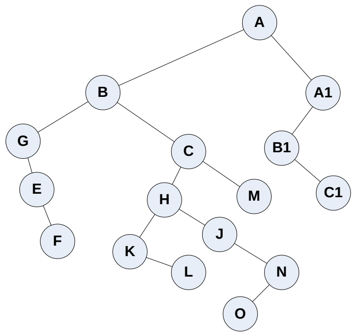
(4) Dobijena slika se rotira za 45 stepeni udesno











-Rekonstrukcija stabla originalnog stabla

-pokazivač nalevo je pokazivač na prvo dete

-pokazivač na desno je pokazivač na brata

Transformacija u striktno binarno stablo

-Transformiše bilo ***koje stablo*** u striktno binarno stablo

-Koraci:

(1) Prekinu se sve veze između roditelja i dece izuzev krajnje leve veze

(2) Povežu se sva deca ista roditelja

(3)Dobijena slika se zarotira za 45st udesno

<u>Stablo za binarno pretraživanje (BST)</u>

-Kod običnog binarnog stabla pretraživanje se zasniva na nekom od prolaza i ima **O(n) efikasnost**.

-BST se zasniva se na ideji da se binarno stablo organizuje tako da omogući pretraživanje slično binarnom pretraživanju niza

-BST je binarno stablo kod koga **za <u>svaki</u> čvor važe sledeća 2 uslova**:

(1) Svi čvorovi ***u levom*** podstablu imaju sadržaj koji je ***manji ili jednak*** sadžaju datog čvora

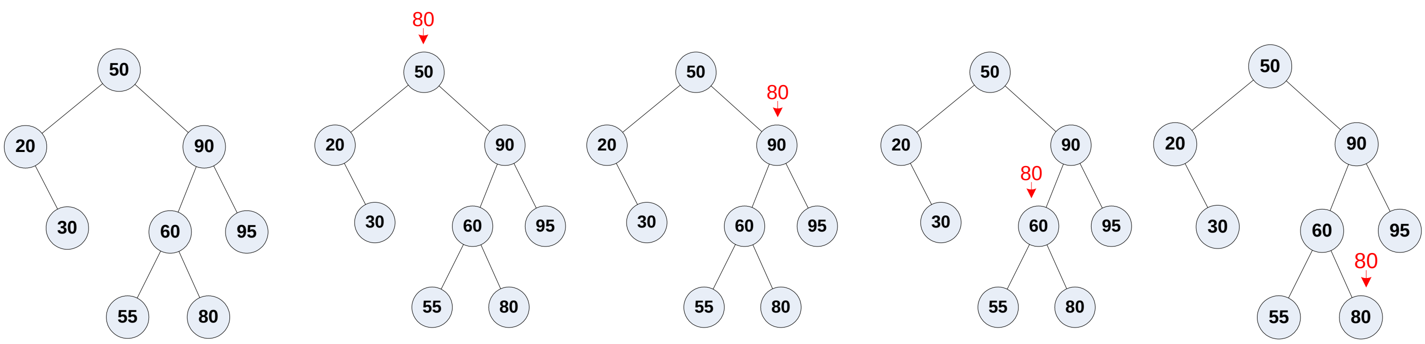
(2) Svi čvorovi ***u desnom*** podstablu imaju sadržaj koji je ***veći ili jednak*** sadžaju datog čvora

Postupak pretraživanja BST

-Ako je sadžaj korena ***veći*** od traženog ključa onda se pretražuje ***levo*** podstablo rekurzivno

-Ako je sadžaj korena ***manji*** od traženog ključa onda se pretražuje ***desno*** podstablo rekurzivno

-Ako je ***jednak*** po sadržaju onda je ***koren*** traženi čvor

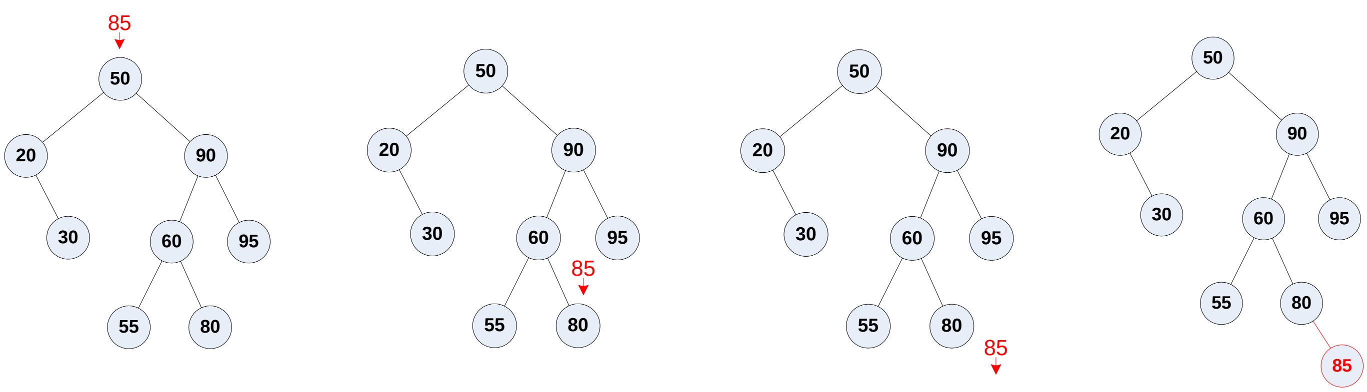


Ubacivanje u BST

-Postupak ubacivanja u BST

(1) Prvo se ***pretraži*** BST

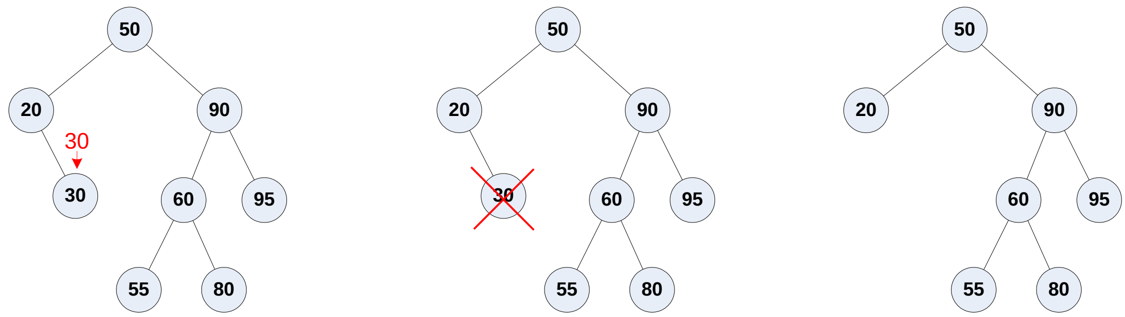
(2) Ubaci se novi čvor ***na mesto nula pokazivača*** gde se pretraživanje završilo



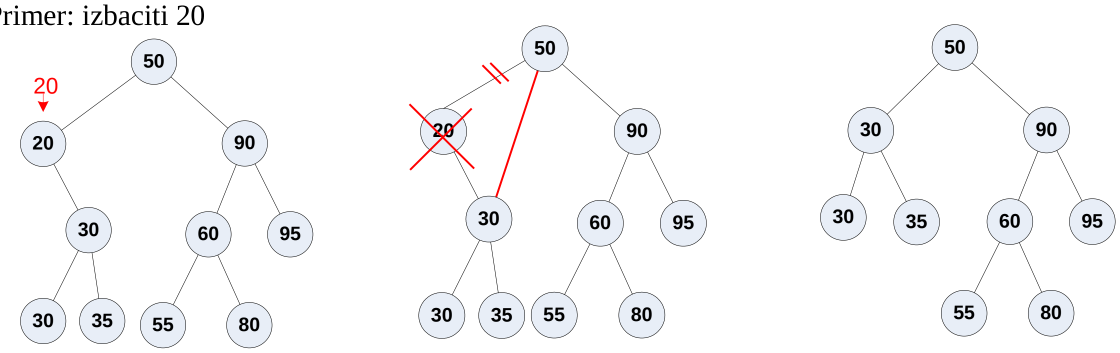
Izbacivanje iz BST

-Postupak izbacivanja u BST
(1) Prvo se ***pretraži*** BST da bi se našao traženi čvor za izabacivanje
(2) zatim se ***nadjeni čvor izbucuje*** na nači koji zavisi od njegove pozicije
(3) Moguće su ***tri situacije*** gde se nalazi čvor:
a) *Čvor je list*
b) *Čvor je polu-list* (Ima samo jedno dete)
c) *Čvor je unutrašnji* (Ima oba deteta)

-Situacija 1: Čvor je list - prosto se izbaci čvor iz stabla i ažurira pokazivač njegovog roditelja da sadrži null vrednost
-Primer: izbaciti 30

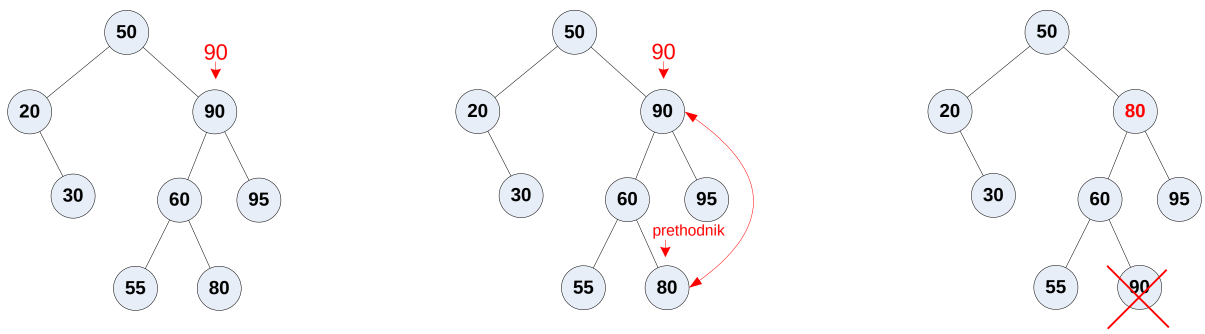


-Situacija 2: Čvor je polu-list - izbaci čvor iz stabla, a pokazivač njegovog roditelja se ažurira da sadrži pokazivač na dete čvora koji se izbacuje
-Primer: izbaciti 20



-Situacija 3: Čvor je unutrašnji
(1) Pronađe se njegov prvi sledbenik (ili prethodnik) koji mora biti na krajnjoj levoj (tj. desnoj) poziciji u desnom (tj. levom) podstablu. Sledbenik (tj. prethodnik) mora biti ili list ili polu-list
(2) Zameni se sadržaj sledbenik (tj. prethodnik) sa sadržajem traženog čvora.
(3) Izbacuje se čvor sa pozicije sledbenika (tj. prethodnika). Ovo se svodi na slučajeve 1. i 2. izbacivanja

-Primer: Izbaciti 90

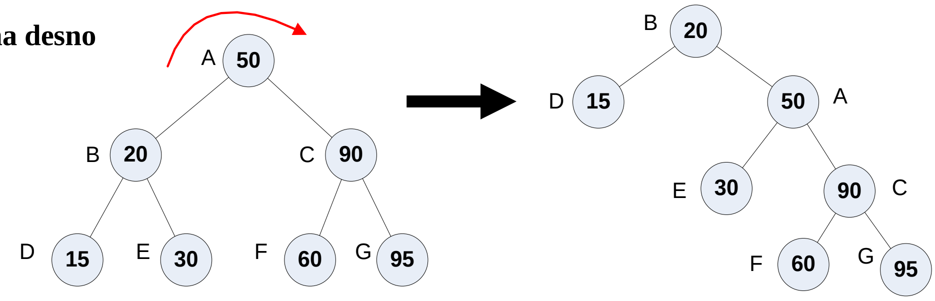


-Ako je stablo **dobro balansirano *efikasnost je O(log n)***
-Ako stablo **nije balansirano**, tada je ***efikasnost lošija***.
-Najgori slučaj je O(n) - kada se ubacuje sortirani niz brojeva, BST se degeneriše u listu

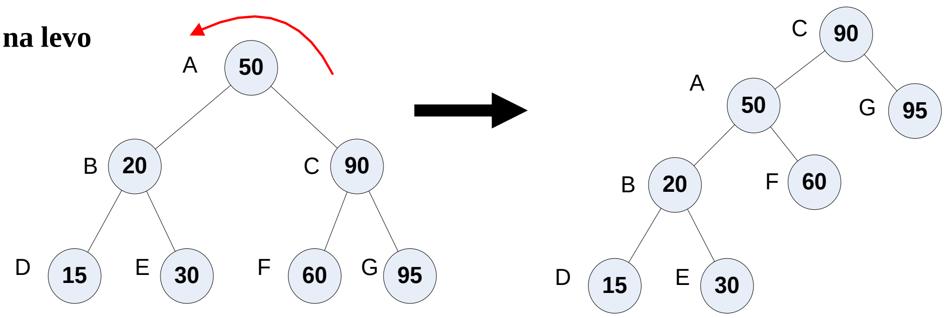
Visinski balansirana stablo – AVL

-BST stablo kod koga za svaki čvor važi da se visina njegovog levog i desnog podstabla ne razlikuje za više od 1 se naziva ***visinski balansirano stablo*** ili ***AVL stablo***
-AVL po imenima naučnika ***Addison, Velsky i Landin*** koji su predložili ovakvo stablo
-Rešava problem najgoreg slučaja kod BST
-Pretraživanje garantovano ima ***efikasnost O(log n)***
-U odnosu na BST ima ***modifikovani algoritme*** za ubacivanje i izbacivanje
   -Zasniva se na rotacijama
   -Rotacije mogu biti na levo i na desno

-Rotacije na desno



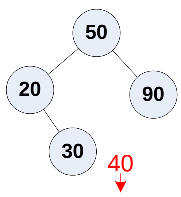
-Rotacije na levo



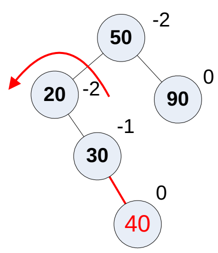
Balansiranje stabla
-Rotacije menjaju visinski balans stabla
   -Rotacija na desno povećava debalans u korist desnog podstabla
   -Rotacija na levo povećava debalans u korist levog podstabla
-Ovo se koristi kod algoritama za ubacivanje i izbacivanje
   -Kada operacija naruši visinski balans, tj. stvori se debalans na jednoj strani, primenjuje se odgovarajuća rotacija da koriguje debalans
   -Rotacija je uvek suprotna debalansu kako bi izjednačila balans. Npr, ako je debalans na desno, koristi se leva rotacija i obrnuto

Ubacivanje u AVL
   (1) Ubacuje se čvor na isti način kao u slučaju BST
Proveri se za svaki čvor u stablu njegov debalans, tj. razlika visina između levog i desnog podstabla
   (2) Ako postoji debalns, onda se vrši odgovarjuća rotacija oko čvora k koji je najbliži mestu ubacivanja
   (3) Ako je oznaka debalansa deteta čvora k suprotna, onda se pre rotacije iz koraka 3, vrši suprotna rotacija oko tog deteta čvora k. U ovom slučaju su potrebne dve rotacije!
      -DEBALANS = Visina Levo – Visina Desno

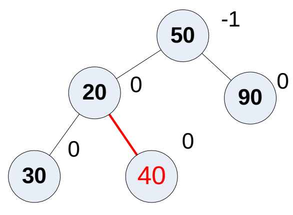
-Ubacuje se broj **40**:



Debalans = Visina Levo – Visina Desno

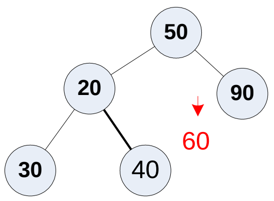


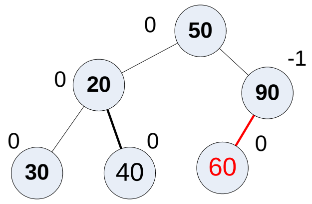
Potrebna jedna rotacija u levo oko 20



Rezultat nakon rotacije

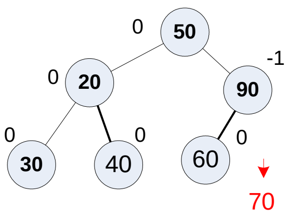
-Ubacuje se broj **60**:

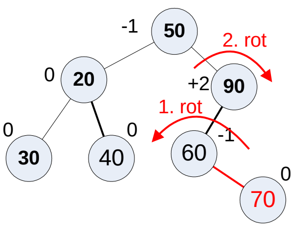




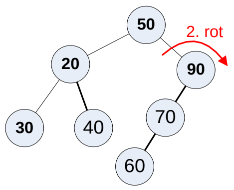
Nema debalansa, nisu potrebne rotacije

-Ubacuje se broj **70**:

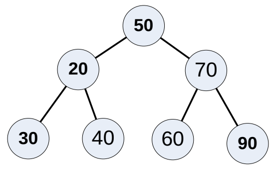




Postoji debalans, suprotni znaci, potrebne 2 rotacije



Nakon 1. rotacije



Nakon 2. rotacije

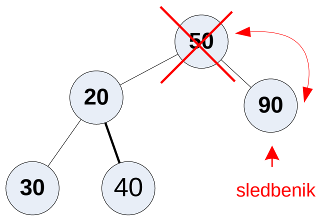
Izbacivanje iz AVL stabla

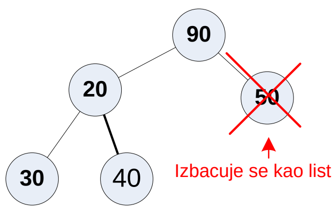
-Izbacivanje iz AVL stabla se vrši na sledeći način:

(1) Izbaci se čvor na isti način kao kod BST. 3 slučaja:
- a) Čvor list – samo se izbaci
- b) Čvor polu-list – preveže se dete na roditelja
- c) Čvor unutrašnji – vrši se zamena sa prethodnikom (sledbenikom) i svodi se na prva dva slučaja

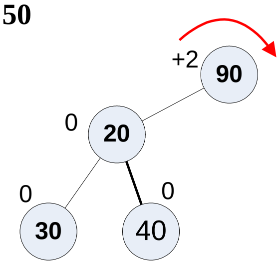
(2) Provri se debalans i vrše potrebne (jedna ili dve) rotacije kao i kod ubacivanja

-Izbacuje se broj **50**:



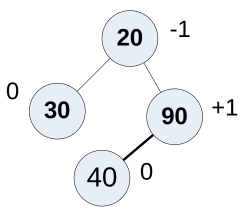


-Nakon izbacivanja **50**



Postoji debalans, potrebna 1 rotacija

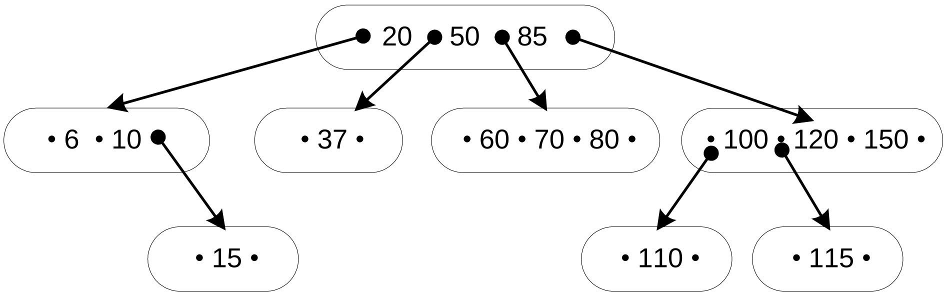
Nakon rotacije:



B, B*, B+ stabla

Višegranska stabla
-Uopštenje binarnih stabala pretraživanja
-Postoji ***više ključeva*** u čvoru
-Svakom ključu pridružena ***dva podstabla*** **-** jedno podstablo je istovremeno levo desno za dva susedna ključa
-Svi ključevi u levom podstablu su manji, a u desnom podstablu veći od datog ključa.
-Red stabla određuje koliko može svaki čvor imati maksimalno ključeva. ***Maximalni broj podstabala je n+1***

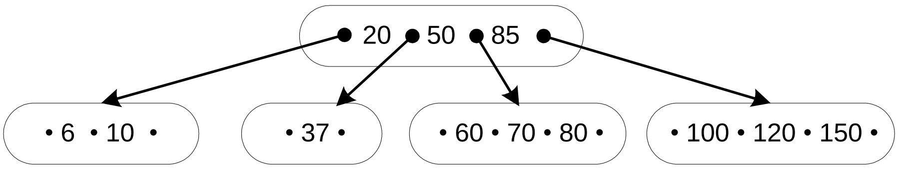
-Stablo reda 3:



B-stabla

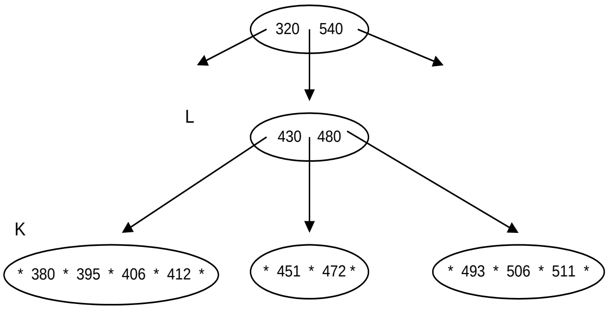
-B-Stablo reda n je **višegransko stablo reda n kod koga važi**:
(1) ***Svi listovi su na istoj visini***
(2) ***Svaki čvor, izuzev korena, ima minimalno n/2 ključeva***

-B-Stablo reda 3:

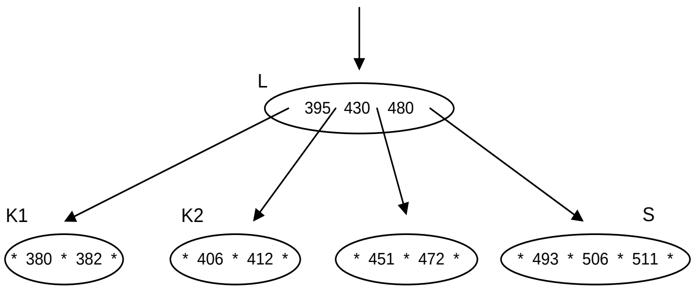


Ubacivanje u B-Stablo
-Pronaći čvor gde se vrši ubacivanje (Koristi se algoritam za pretraživanje)
-Ubaciti novi ključ
(1) Ako **ima mesta**, ključ se prosto ubaci
(2) Ako **nema mesta**, onda:
-Pocepa se dati čvor na dva, i sortirani niz postojećih ključeva (uključujući i novi), osim srednjeg ključa (sredina sortiranog niza) se raspodeljuje u nove čvorove
-Srednji ključ se ubacuje u nadređeni čvor
-Postupak ubacivanja se rekurzivno ponavlja

-Primer ubacivanja **382**



Posle ubacivanja 382 (čvor K se pocepao)



Izbacivanje iz B-Stabla

-**Dva slučaja** izbacivanja:

(1) ***Izbacivanje iz čvora koji nije list***

-Svodi se na izbacivanje iz čvora koji jeste list

-Pronalazi se prethodnik (ili sledbenik) datog ključa

-Prethodnik (sledbenik) se nalazi na krajnjoj desnoj (levoj) poziciji u krajnjem desnom (levom) čvoru u levom (desnom) podstablu datog ključa

(2) ***Izbacivanje iz čvora koji je list***

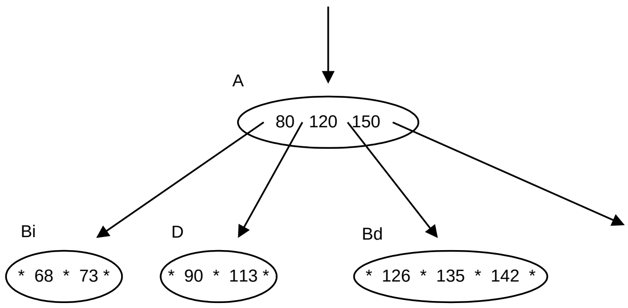
-Problem se javlja ako nakon izbacivanja u listu ostaje manje od n/2 ključeva. U tom slučaju:

-Pozajmjuje se od levog ili desnog brata ključevi

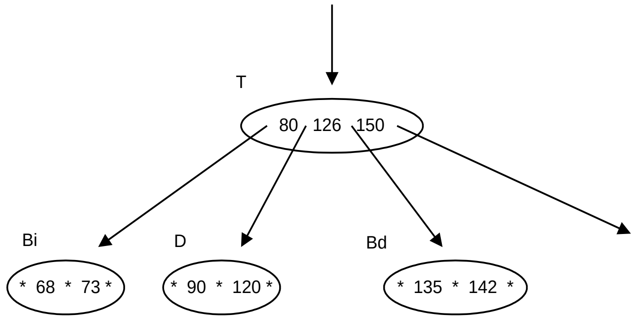
-Ako nijedan brat nema dovoljno ključeva tada se dati čvor spaja sa nekim od svoje braće. Tom prilikom se spušta njihov nadređeni ključ iz nadređenog čvora

-Ako spuštanje nadređenog ključa narušava integritet onda se postupak izbacivanja rekurzivno ponavlja

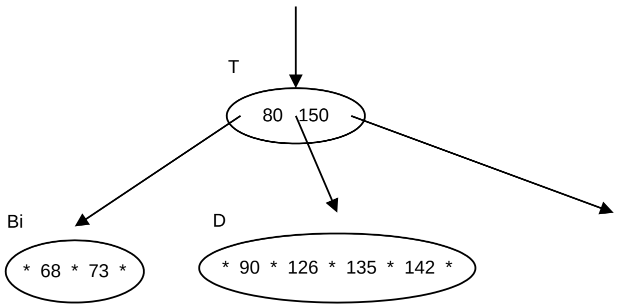
-Primer izbacivanja ključa **113**



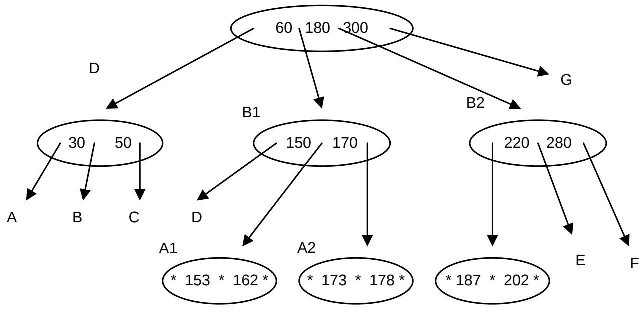
Nakon izbacivanja ključa 113

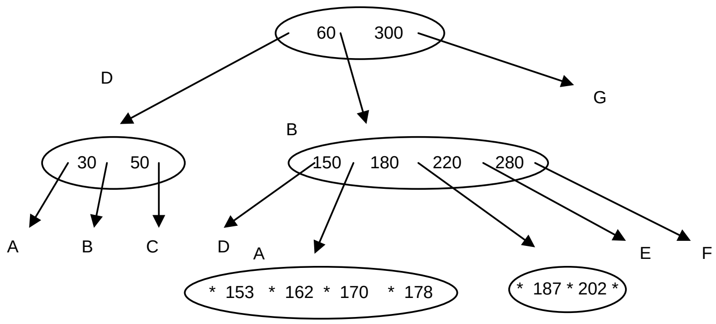


Nakon izbacivanja ključa 120



Nakon izbacivanja ključa 173





Efikasnost B-stabala
-Neka je red stabla **m-1**
-n je ukupan broj ključeva u stablu.
-Svaki čvor sadrži najmanje
q = (m-1) div 2 ključeva.

Visina B-stabla	Minimum		Maksimum	
	čvorova	ključeva	čvorova	ključeva
1	1	1	1	m-1
2	2	2q	m	(m -1)m
3	2(q+1)	2q(q+l)	m^2	(m -l)m^2
⋮	⋮	⋮	⋮	*. ⋮
i	$2(q+1)^{i-2}$	$2q(q+1)^{i-2}$	m^{i-1}	$(m-i)m^{i-1}$
⋮	⋮	⋮	⋮	⋮
ukupno za visinu d	$\frac{2(q+1)^{d-1}-1}{q}+1$	$2(q+1)^{d-1}$	$\frac{m^d-1}{m-1}$	m^d-1

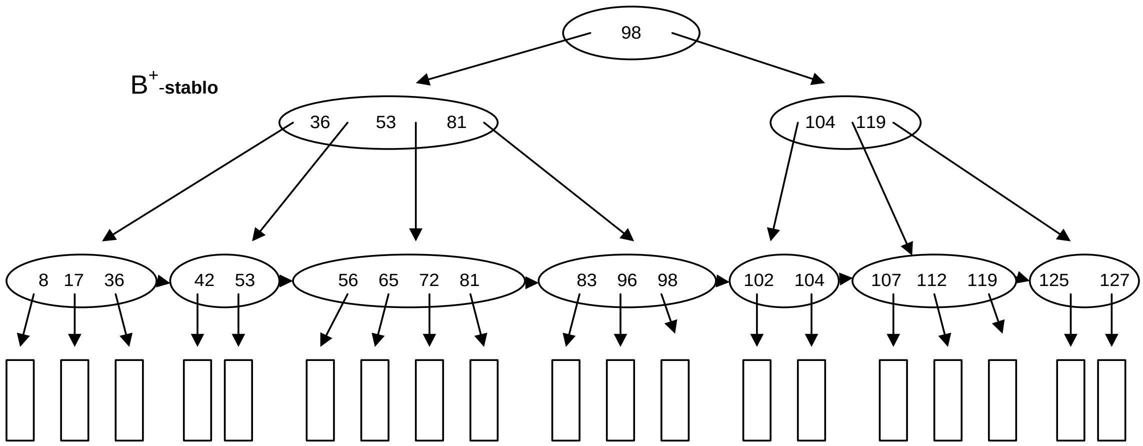
-Iz minimalnog broja ključeva pronalazimo da je **maksimalna visina** (d) B-stabla: $\mathbf{n = 2(q+1)^{d-1}}$.
-Logaritmujući levu i desnu stranu dobijamo: $\mathbf{\log_{q+1} n = \log_{q+1} 2(q+1)^{d-1}}$.
-Sređivanjem dobijamo: $\mathbf{\log_{q+1} n - \log_{q+1} 2 = (d-1) \log_{q+1}(q+1)}$,
-a iz toga sledi

$$\boxed{\mathbf{d = \log_{q+1}(n/2) + 1}}$$

B stablo*

-B* - Poboljšan algoritam ubacivanja
-Umesto da se čvor u kome nema mesta za novi ključ odmah cepa, vrše se prethodno **pozajmica** viška ključeva njegovoj braći
-Tek ako pozajmica nije moguća (ako su braća popunjena) se vrši cepanje

-Omogućava i **sekvencijalni pristup**



Hashing – funkcija transformacije

-Skupovi kao strukture podataka

-Skup kao struktura podataka

(1) Ne postoji uređenje elemenata u skupu, tj.operacije *sledeći*, *prethodni*, *prvi*, *zadnji* nemaju smisla

(2) Samo se zna da je dati element član skupa, tj. *nadjiPoKljuču* jedino ima smisla

(3) Potrebne operacije *ubaci* i *izbaci*

Specifikacija skupa - Definicija preko ATP

```
public interface ISkup
{
  void izbaci(int obj);
  void ubaci(int obj);
  int nadjiPoKljucu(int k);
...
}
```

Implementacija skupa pomoću transformacije ključa u adresu (hashing-a)

-Niz (tabela) može da pamti elemente skupa

-Transformacijom ključa u adresu (indeks) se određuje pozicija elementa u nizu

-Funkcija koja vrši transformaciju se haziva **hash** funkcija, a ceo ovaj pristup **hashing**

-Potrebno je da hash funkcija generiše jedinstvene adrese

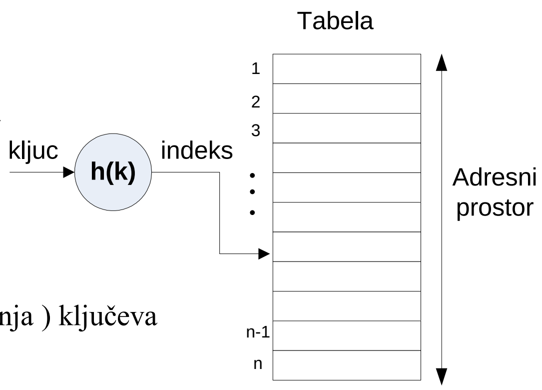
-Prilikom ubacivanje se pomoću hash funkcije odredi adresa i element se smešta na datu adresu

-Prilikom pretraživanja po ključu hash funkcijom se određuje adresa gde se on nalazi

Organizacija zasnovana na hashingu

-Teoretski dovoljan samo 1 pristup

-Efikasnost O(1), potencijalno brže od svih drugih metoda pretraživanja



Problemi hashing organizacije

(1) Kako pronaći idealnu hash funkciju h(k)

-Treba da bude jednoznačna transformacija, tj. bez kolizije (sudaranja) ključeva

-Za svako k1 ≠ k2 → h(k1) ≠ h(k2)

(2) Ako se dogodi kolizija kako je rešiti

-Problem ubacivanja: šta uraditi sa elementom koji dobiju adresu koja je već zauzeta

-Problem pretraživanja: kako naći traženi element ako nije na adresi dobijenoj pomoću hash funkcije

Hashing funkcija h(k)

-Idealna (perfektna) h(k)

-Moguće je konstruisati samo ako se unapred znaju svi mogući ključevi i adrese

-U praksi ovaj uslov često nije ispunjen

-Dobre osobine h(k)

-Da se brzo računa

-Da ima skoro slučajnu (uniformnu) raspodelu adresa, tj. da nema tačke nagomilavanja (sudaranja)

Metode za realizaciju neperfektne h(k)

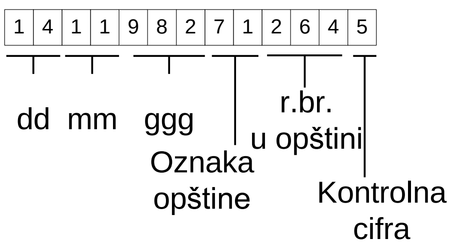
-Selekcija cifara

-Obično je ključ numerički, pa se mogu neke cifre iz ključa uzeti kao adresa (Važan odabir cifara)

-Npr. skup od 50.000 studenata. Adresni prostor je 0-50.000, dakle, potrebno je da indeks ima pet cifara.

-Ključ studenta je MLB – 13 cifara. Npr. 1411981712645. Odabrati pet cifara od 13

Struktura MLB



-Ako se uzmu dd,mm,g svi rodjeni istog dana i meseca u istom veku će imati istu adresu
-Bolje je ako se uzmu 2 cifre opštine i tri cifre rednog broja

-Ostatak od celobrojnog deljenja

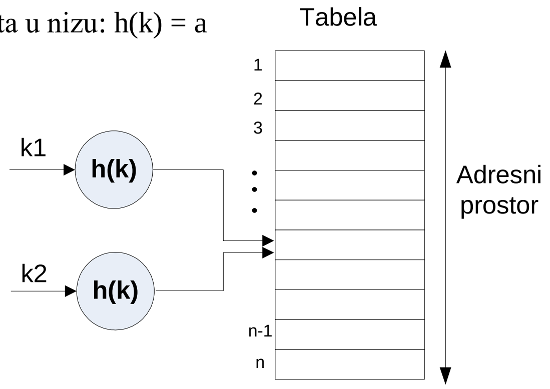
$h(k) = k \bmod m = b$; $0 <= b <= m$

-Adresni prostor od 0 – m generiše iste adrese ako je k faktor od m (Npr. ako je m = 25, svi ključevi koji su deljivi sa 5 će se preslikavati na adrese 0, 5, 10, 15 i 20)

-Bolje ako k i m nemaju zajednički faktor. Stoga za m treba uzeti prost broj blizak veličini adresnog prostora (Npr. ako je n = 100, m = 103)

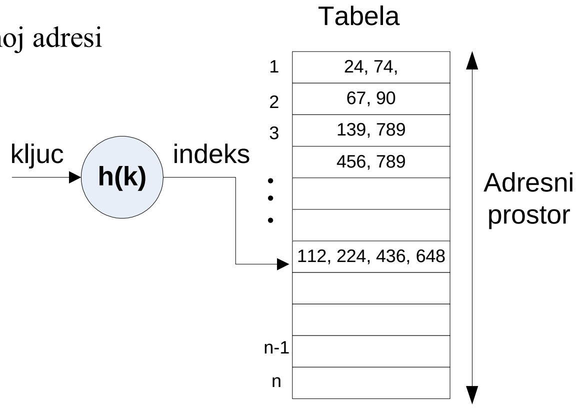
Problem kolizija

-Transformacijom ključa u adresu (indeks) se određuje pozicija elementa u nizu: h(k) = a
-Problem kolizije: **$k1 \neq k2$, $h(k1) = h(k2)$**



Prihvatanje više zapisa na jednu adresu

-Jedna adresa može da prihvati više
-Problem smeštanja ne postoji sve dok se ne popuni prostor na jednoj adresi



-Ako se dogodi kolizija kako je rešiti
(1) Problem ubacivanja: šta uraditi sa elementom koji dobiju adresu koja je već zauzeta
(2) Problem pretraživanja: kako naći traženi element ako nije na adresi dobijenoj pomoću hash funkcije
-Prihvatanje više zapisa na jednu adresu
-Ublažava se problem kolizije, tj. problem smeštanja

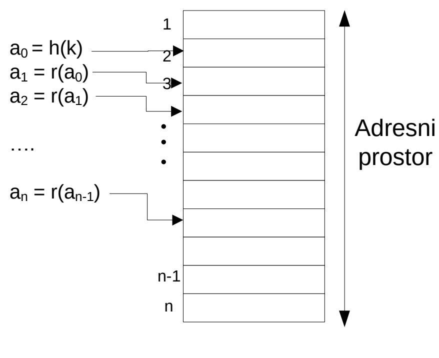
-Dva standardna načina rešavanja kolizije

(1) <u>**Otvoreno adresiranje**</u>

-Otvoreno adresiranje se bazira na ideji da se primeni druga hash funkcija r(k) na adresu dobijenu originalnom h(k)

-Uzastopna primena r(k) dok se ne dobije slobodna adresa (n – veličina tabele):

$a_0 = h(k)$
$a_1 = r(a_0) \bmod n$ //da ne da veću adresu od n
$a_2 = r(a_1) \bmod n$
….
$a_n = r(a_{n-1}) \bmod n$



a) <u>Linearno probanje</u>

-Problem izbora r(k)
-Uvećati prethodno dobijenu adresu za 1: **$r(a_{i+1}) = r(a_i) \bmod n$**

b) Sekundarna kolizija

-Primarna kolizija je kada različiti ključevi imaju istu adresu

-Sekundarna kolizija se javlja kada se sudare ključevi koji imaju <u>različite h(k)</u>

-Otvoreno adresiranje generiše sekundarnu koliziju - Uvećava se broj kolizija

c) Kvadratno probanje

-Problem izbora r(k)

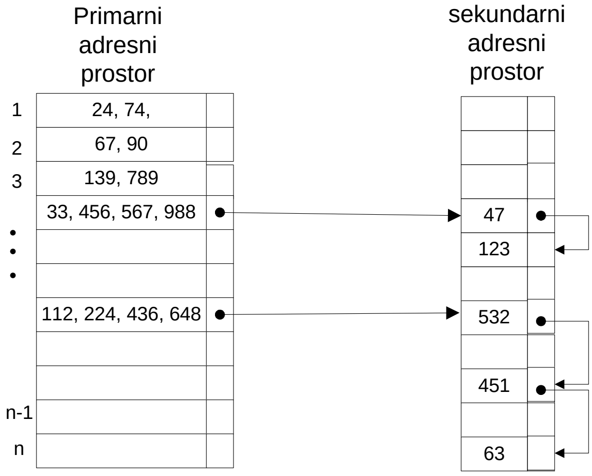
-Sukcesivno uvećati prethodno dobijenu adresu za kvadrat prethodnog rastojanj: $r(a_{i+1}) = r(a_i) \pm j^2$; j=1,2,…n

-Delimično se rešava problem sekundarne kolizije

(2) <u>Metoda olančavanja</u>

-Problem primarne kolizije se rešava uvođenjem posebne zone za zapise koji dobiju istu adresu

-Zapisi koji su u koliziji se olančavaju



Poređenje metoda rešavanja kolizije

Metoda	Prednost	Nedostatak
Otvoreno adresiranje	Nema dodatnog memorijskog prostora	Problem sekundarne kolizije
Metoda olančavanja	Nema sekundarne kolizije	-Potreban dodatan memorijski prostor -Sekvencijalno pretraživanje jednostruko spregnute liste

<u>*Grafovi i mreže*</u>

-Grafovi su nelinearne strukture podataka

-Predstavljaju najopštije strukture podataka

-Odnos između elemenata nije linearan; jedna element može imati više sledbenika i više prethodnika

-Nema ograničenja u pogledu povezivanja elemenata

-Elementi grafa se nazivaju **čvorovi**

-Veze između čvorova se nazivaju **lukovi**

Definicija orjentisanih grafova

-Struktura podataka B=(K,R); K-skup čvorova i R - binarna relacija nad skupom K i to tako da svaki par čvorova $(k_1, k_2) \in R$ kažemo da obrazuje luk grafa

-Luk $l = (k_1, k_2)$ povezuje čvor k_1 sa čvorom k_2 i to tako da čvor k_1 prethodi čvoru k_2 binarna;

-Za graf kažemo da je ***neorjentisan*** ukoliko važi $(k_1, k_2) \in R \rightarrow (k_2, k_1) \in R$

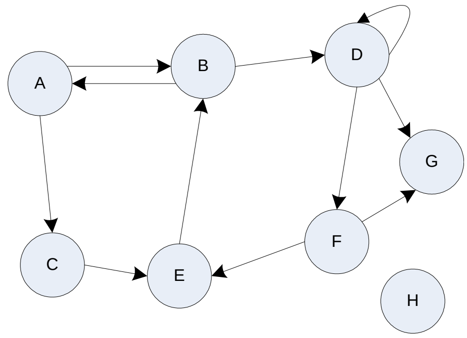
-Za luk $l = (k_1, k_2)$ kažemo da je k_1 ***početni čvor*** luka l, a za k_2 da je završni čvor luka (Kaže se i da luk izlazi iz čvora k_1 i da ulazi u čvor k_2).

-Za dva čvora se kaže da su susedni ukoliko postoji luk koji ih spaja

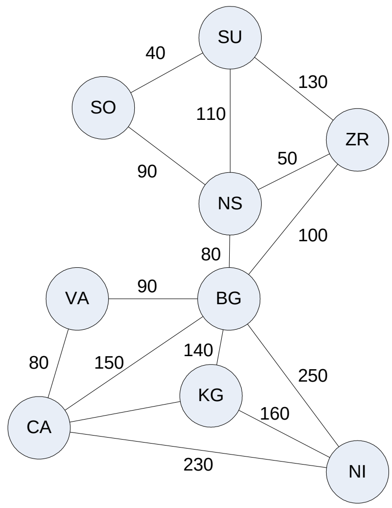
-Za dva luka se kaže da su susedni ukoliko imaju jedan zajednički čvor

-Čvor koji nema susednih čvorova se naziva ***izolovani čvor***.
-Luk kod koga su identični završni i početni čvor se naziva ***petljom***

-Lukovi čiji završni čvor jednog je identičan početnom čvoru drugog se naziva ***put grafa***
-Put kod koga su početni čvor prvog luka i završni čvor poslednjeg luka identični se naziva ***ciklus***
-Broj lukova koji obrazuje put se naziva ***dužinom puta*** (Kod petlje dužina puta je 1!)
-Kod neorjentisanog grafa se umesto termina put koristi termin ***lanac***
-Za graf kažemo da je ***povezan*** ukoliko između svaka dva čvora postoji lanac
- Za graf kažemo da je ***stogo povezan*** ukoliko između svaka dva čvora postoji put

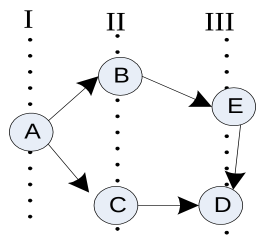


Mreže
-Grafovi kod kojih se likovima pridružuje neki podatak se naziva mreža
-Primer: *putna mreža* (neorijentisan graf)

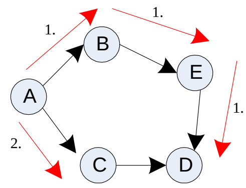


Prolazi kroz graf
-Obilazak svih čvorova grafa tačno jednom
-Potrebni za rešavanje raznih problema:
   -Da li su dva čvora povezana?
   -Nalaženje najkraćeg rastojanja
-Postoji više algoritama za obilazak grafa

(1) Prolaz po širini
-Čvorovi se obilaze po slojevima (nivoima)
-Kada se jedna čvor obiđe, onda se obilaze svi njegovi susedi, pa tek onda susedi njegovih suseda, itd.
-<u>Primer</u>:   A, B, C, E, D
-Algoritam za prolaz po širini
-zahteva korišćenje ATP Red (queue)
-Svaki čvor ima status: Čeka, spreman i obrađen
-Obilaze se svi čvorovi grafa i ubacuju u red ako već nisu posećeni ili ubačeni u red (tj. one sa statusom da čekaju)
-Uzima čvorove iz reda, posećuje ih i menja status da su posećeni, a zatim ubacuje sve njihove susede u red ako već nisu posećeni ili ubačeni u red i menja im status u speman (tj. da su ubačeni u red)



(2) Prolaz po dubini
-Kada se jedna čvor obiđe, onda se obilazi <u>jedan</u> njegov sused, pa tek kad se on i njegovi susedi obiđu rekurzivno po dubini, prelazi se na obilazak ostalih suseda
-<u>Primer</u>:   A, B, E, D, C
-Algoritam za prolaz po dubini
-zahteva korišćenje ATP Stek (stack)
-Svaki čvor ima status: Čeka, spreman i obrađen
-Obilaze se svi čvorovi grafa i ubacuju u stak ako već nisu posećeni ili ubačeni u red (tj. one sa statusom da čekaju)
-Uzima čvorove iz staka, posećuje ih i menja status da su posećeni, a zatim ubacuje sve njihove susede u stak ako već nisu posećeni ili ubačeni u stak i menja im status u speman (tj. da su ubačeni u red)



Implementacija grafova

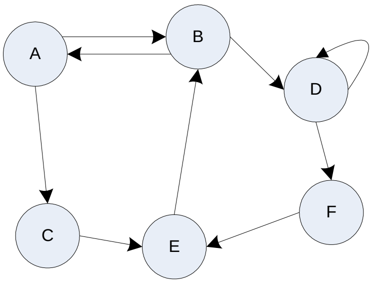
(1) Matrice susedstva

-Matrica susedstva grafa G sa n čvorova je kvadratna matrica A [i, j] reda n čiji elementi:

a) 1, ako je čvor i susedan čvoru j

b) 0, ako čvor i i j nisu susedni

-Ako je graf neorijentisan tada je njegova matrica susedstva simetrična: $\mathbf{A = A^T}$



	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	0	0	1	0	0
C	0	0	0	0	1	0
D	0	0	0	1	0	1
E	0	1	0	0	0	0
F	0	0	0	0	1	0

-Implementacija matrice susedstva

-Matrica je dvodimenzioni niz

-Prostorna kompleksnost $O(n^2)$

-Ako je broj čvorova grafa veliki, a sadrži mali broj lukova, onda se dobija tzv. <u>retko posednuta</u> matrica, tj. matrica sa mnogo 0

-Neefikasno korišćenje memorije

-Koristi se bit-matrice

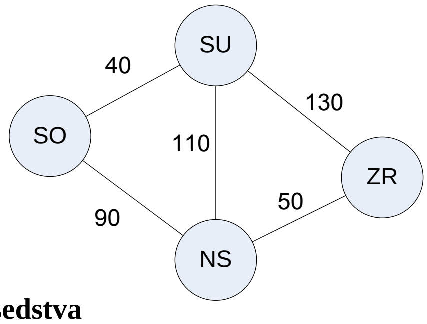
-Za pamćenje elementa Aij se koristi samo jedan bit

-Potrebno je manipulisati bitovima

-Predstavljanje mreže

-Matrica susedstva se može iskoristiti za predstavljanje mreža

-Elementi a_{ij} matrice A sadrže vrednost luka



	SO	SU	NS	ZR
SO	0	40	90	
SU	40	0	110	130
NS	90	110	0	50
ZR		130	50	0

(2) Liste susedstva

-Lista susedstva je multi-lista – tj. lista listi, tj. lista čiji elementi su liste

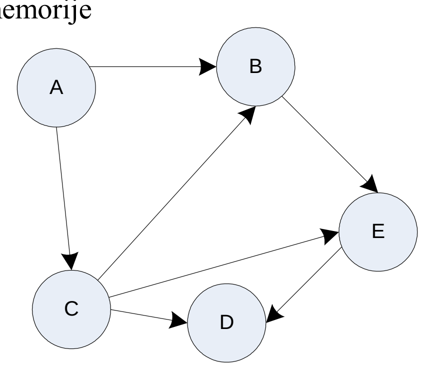
-Lista čvorova grafa od kojih svaki sadrži listu svojih suseda

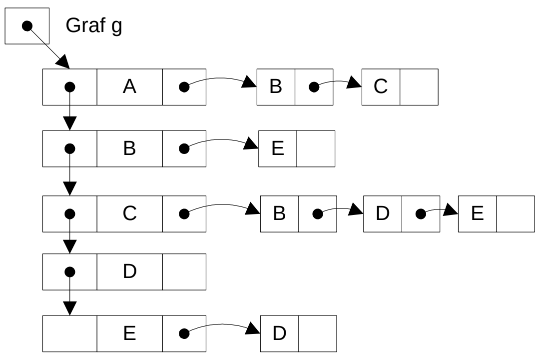
-Prostorna kompleksnost O(n)

-Za broj čvorova O(n)

-Za broj lukova O(n)

-Efikasno korišćenje memorije





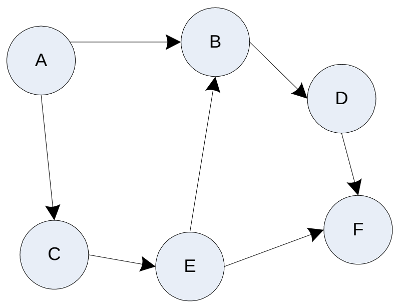
-Poređenje metoda:
(1) Matrice susedstva
-Jednostavnije za manipulaciju
-Prostorna kompleksnost $O(n^2)$
-Neefikasno korišćenje memorije kod retko posednutih matrica
(2)Liste susedstva
-Složenije za manipulaciju
-Prostorna kompleksnost O(n)
-Efikasnije korišćenje memorije

Algoritmi nad grafovima

(1) Topologijski sort

-Linearno uređenje čvorova grafa tako da nijedan čvor i koji prethodi nekom čvoru j ne bude u lineranoj listi posle čvora j
-Primenljivo samo na orijentisane aciklične grafove

-Topologijski sort:
(1) Nadji čvor bez sledbenika
(2) Ubaci ga na listu
(3) Izbaci iz grafa nađeni čvor i sve njegove lukove
(4) Ponovi korake 1-3 sve dok graf ne ostane prazan



-Primer: Toplogijski sort: A, C, E, B, D, F

-Topologijski sort zahteva dve osnovne procedure:
-Nalaženje čvora koji nema sledbenika
-Izbacivanje čvora i svih njegovih lukova

(2) Tranzitivni zatvarač
-Graf T je tranzitivni zatvarač grafa G akko postoji luk (i,j) u grafu T ako postoji puitanja bilo koje dužine u grafu G između čvorova i i j.
-P – matrica susedstva ---- Putanje duzine 1
-P^2=PxP ---- Putanje dužine 2
-P^n=PxPx ..P ---- Putanje dužine n
-Algoritam koji stepenuje matricu susedstva na n-ti stepen nije efikasan
-Warshall-ov algoritam
-Pk(i,j) = 1 ako postoji putanja od i do j koja prolazi kroz čvorove koji su numerisani do k
-Pk+1(i,j) = 1 akko
-P(i,j) = 1
-Pk(i,k+1) and Pk(k+1,j)
-Efikasnost je $O(n^3)$

```
void Warshal() {
   for(int k = 0; k<= numVertices-1; k++) {
      for(int i = 0; i<= numVertices-1; i++) {
         if (adjMatrix[i, k]) {
            for(int j= 0; j<= numVertices-1; j++)
             adjMatrix[i, j] = adjMatrix[i, k] || adjMatrix[i, k]
         }
      }
   }
}
```