Strukture podataka

Stabla: BSP, 2-3-4

Predmet: Uvod u Algoritme 17 - ESI053
Studijski program: Primenjeno softversko inženjerstvo

DEPARTMAN ZA RAČUNARSTVO I AUTOMATIKU
ccd
DEPARTMAN ZA ENERGETIKU, ELEKTRONIKU I KOMUNIKACIJE
DEET

Stablo

Tipovi:

- Binarno stablo pretrage
- 2-3-4 stablo
- Crveno-Crno stablo
- AVL stablo
- B-stablo
- ...

Element stabla sadrži:

- ključ
  - neka stabla mogu da sadrže elemente sa istim ključem
- satelitske podatke

Omogućava efikasne operacije:

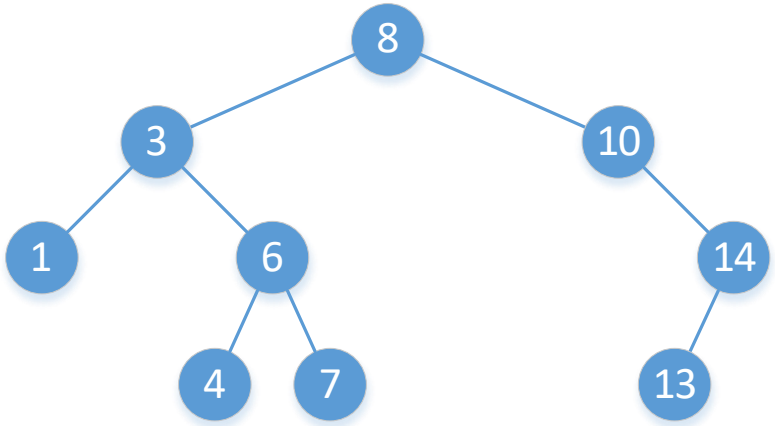
1. Upiti nad skupom
   - pretraga skupa S po ključu k – PRETRAŽI(S,k) - *Search*
   - vraća elemenat sa najmanjim ključem – MINIMUM(S) - *Minimum*
   - ... sa najvećim ... – MAKSIMUM(S) - *Maximum*
   - vraća sledeći elemenat od x – NAREDNI(S,x) - *Successor*
   - ... prethodni elemenat ... – PRETHODNI(S,x) - *Predecessor*
2. Operacije izmena skupa
   - dodaje elemenat x u skup S – DODAJ(S,x) – *Insert*
   - briše elemenat x – OBRIŠI(S,x) - *Delete*

Binarno stablo pretrage (BSP)

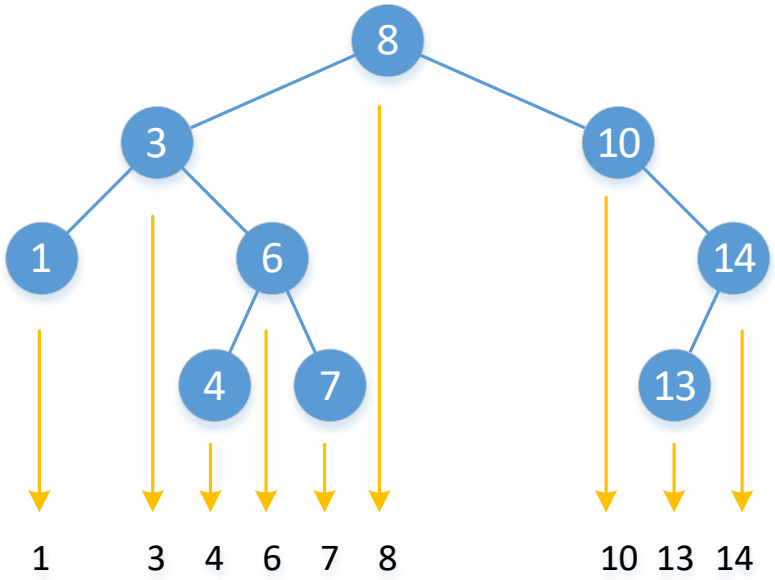
- ili sortirano binarno stablo,
  engl. *Binary Search Tree* (BST)

- Svaki čvor u stablu ima ključ čija se vrednost može porediti
  - veća je od ključeva u levom podstablu (zapravo: ≥)
  - manja je od ključeva u desnom podstablu (zapravo: ≤)

  Posledica: kada se posmatra podstablo, svi ostali ključevi se nalaze van intervala vrednosti ključeva u podstablu.



- Osobine
  - (dobro) pretraga, ubacivanje i brisanje elemenata može uraditi efikasno
  - (loše) stablo može biti degenerisano



Osobine i operacije

- Svaki čvor x ima:
  - Ključ $ključ(x)$ – BSP osobina
  - Roditelja $r(x)$ - pokazivač
  - Levo dete $levo(x)$ - pokazivač
  - Desno dete $desno(x)$ – pokazivač

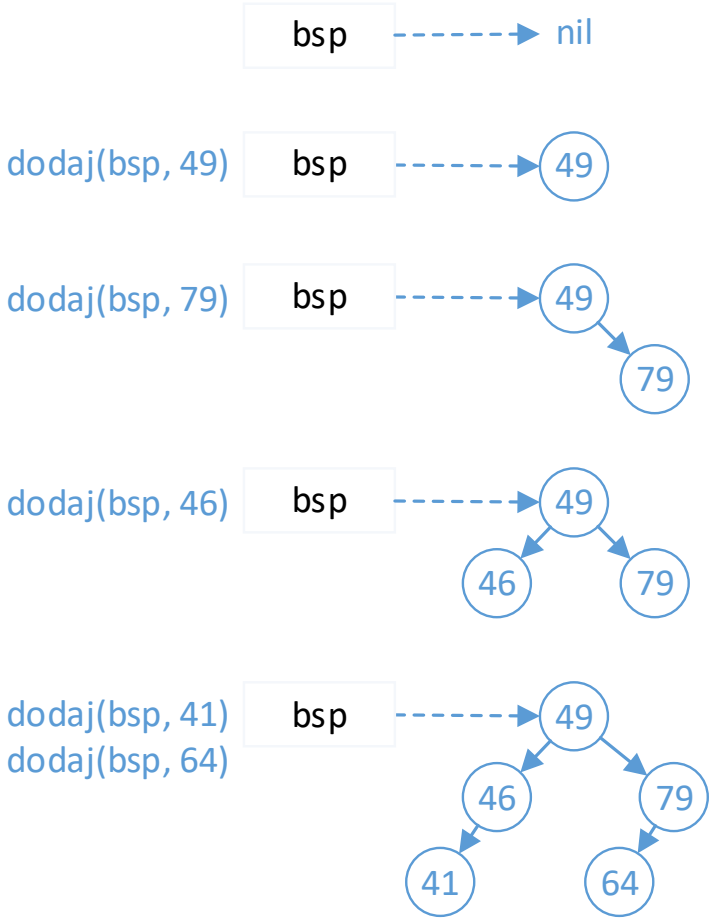
- Operacije
  - Dodaj: DODAJ(x)
  - Obriši: OBRIŠI(x)
  - Pronađi po ključu: PRONAĐI($ključ$)
  - Pronađi min/max: NAJMANJI(), NAJVEĆI()
  - Pronađi sledeći manji/veći elemenat: PRETHODNI(x), NAREDNI(x)
  - Promeni vrednost udruženu sa ključem
- Ako je h visina stabla, onda su sve operacije složenosti $O(h)$

Vežba: napisati pseudokod za navedene operacije i brisanje elementa i ispisivanje svih elemenata u sortiranom poretku.

Dodavanje u BSP

- Dodaje se čvor z sa postavljenim ključem i Nil pointerima za decu.

```
DODAJ(T, z)
1   y = Nil                 // čvor u koji se dodaje
2   x = T.koren             // gde se dodaje
3   while x ≠ Nil
4       y = x
5       if z.ključ < x.ključ
6               x = x.levo
7       else    x = x.desno
8   z.r = y
9   if y == Nil
10      T.koren = z         // bilo prazno
11  elseif z.ključ < y.ključ
12          y.levo = z
13  else    y.desno = z
```



Može se uraditi i rekurzijom.

Pretraga BSP

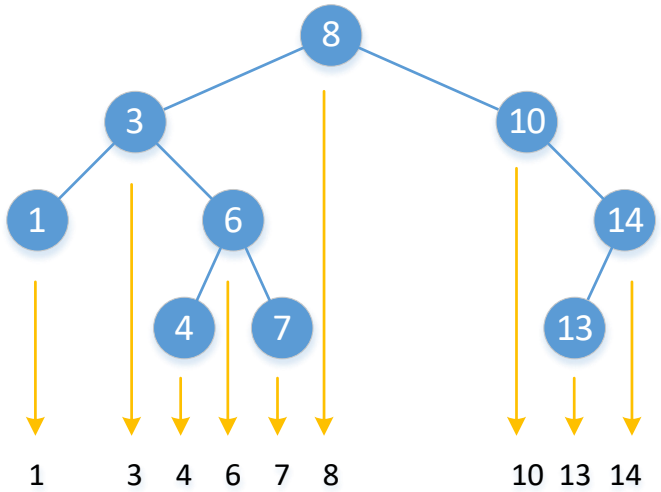
- Ideja: posmatramo elemenat x
  - Ako je njegov ključ jednak traženom onda je gotovo
  - Ako je traženi ključ manji onda se možda nalazi u levom podstablu, pa pretragu tamo nastavljamo
  - Ako je traženi ključ veći onda se možda nalazi u desnom podstablu, pa pretragu tamo nastavljamo

```
PRONAĐI(x, k)
1  if x == Nil | x.ključ == k
2     return x
3  if k < x.ključ
4     return PRONAĐI(x.levo, k)
5  else
6     return PRONAĐI(x.desno, k)
```

```
PRONAĐI(x, k)
1  while x ≠ Nil & x.ključ ≠ k
2     if k < x.ključ
3        x = x.levo
4     else
5        x = x.desno
6  return x
```

Minimum i maksimum

- Elemenat sa najmanjim ključem se nalazi sasvim levo, a sa najvećim sasvim desno.
- Ideja je da se krećemo ulevo (udesno) dokle god je moguće. Poslednji posećen čvor sadrži minimum (maksimum).




```
MINIMUM(x)
1  while x.levo ≠ Nil
2      x = x.levo
3  return x
```

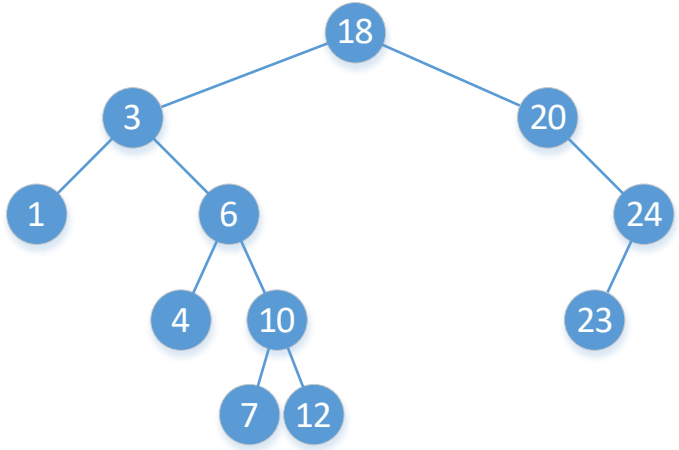
Prikazani pseudokod se može primeniti na podstablo čvora x. Minimum celog stabla se dobija primenom metode na koren stabla.

Slično je za maksimum.

Naredni/prethodni elemenat

Posmatra se dati čvor i tražimo naredni (sledećeg veći):

- Traženi čvor je sa desne strane (ako postoji).
- Ako dati čvor sadrži desno podstablo onda je traženi čvor minimum tog podstabla
  npr. 6, 7
- Ako dati čvor ne sadrži desno podstablo onda je traženi čvor prvi roditelj koji se nalazi sa desne strane
  npr. 12, 18 (roditelji 10, 6, 3 sa putanje od 12 do 18 su sa leve strane).
  Ovo se može obrnuto posmatrati: 18 ima prethodni elemenat koji se nalazi kao maksimum levog podstabla (=12).




```
NAREDNI(x)
1  if x.desno ≠ Nil
2     return MINIMUM(x.desno)
3  y = x.r
4  while y ≠ Nil & y.desno == x
5       x = y
6       y = x.r
7  return y
```

Slično je za prethodni elemenat.

Zamena čvora drugim čvorom

- Čvor x se zamenjuje čvorom y. Nakon toga je x nebitan.
- Potrebna ažuriranja:
  - roditelj od x treba da ukazuje na y (dilema: sa leve ili desne strane?)
  - postaviti novog roditelja za y

```
ZAMENA(T, x, y)
1  if x.r == Nil
2     T.koren = y
3  elseif x == x.r.levo
4        x.r.levo = y
5  else x.r.desno = y
6  if y ≠ Nil
7     y.r = x.r
```

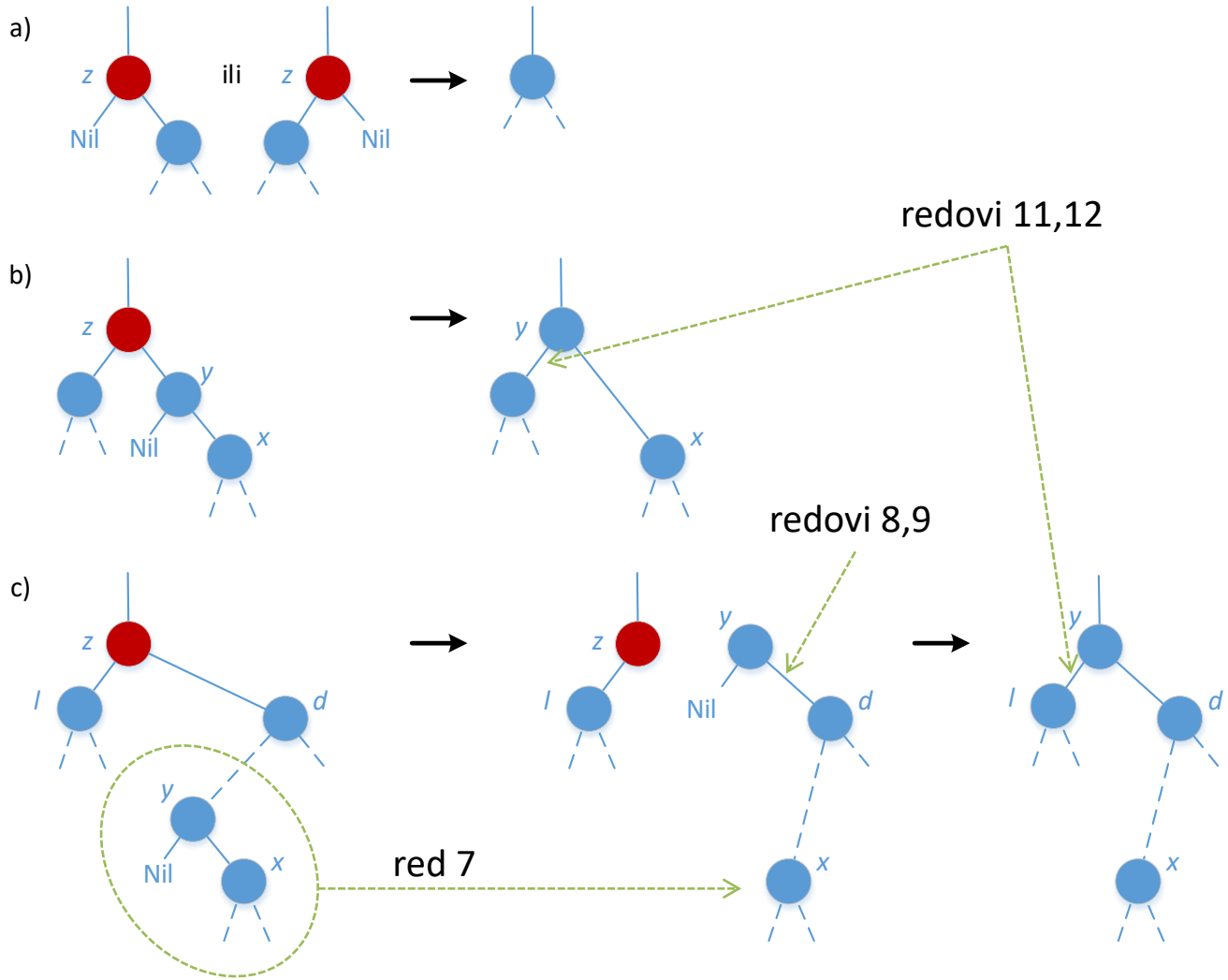
Brisanje čvora

- Posmatra se čvor z
- Lako je kada z nema više od jednog podstabla (nijedno ili levo ili desno). Tada se z zameni tim podstablom (ili sa Nil)
  Podsećanje: ključ roditelja je van opsega ključeva tog podstabla tako da se ne narušava osnovna osobina BSP.
- Ako z ima oba podstabla, onda tražimo npr. sledeći veći elemenat y.
  Podsećanje: y nema levo podstablo, dok može imati desno podstablo x.
  Ideja je da se zameni z sa y, ali se prvo zamenjuje y sa x.
  Ako je y dete od z tada je to jednostavniji slučaj.

Alternativno rešenje može tražiti sledeći manji od z.

Brisanje čvora (2)

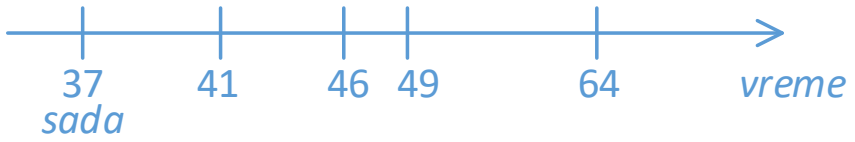
```
OBRIŠI(T, z)
1   if z.levo == Nil
2       ZAMENI(T, z, z.desno)    // a)
3   elseif z.desno == Nil
4       ZAMENI(T, z, z.levo)     // a)
5   else y = MINIMUM(z.desno)
6      if y.r ≠ z
7         ZAMENI(T, y, y.desno) // c)
8         y.desno = z.desno
9         y.desno.r = y
10     ZAMENI(T, z, y)           // b,c)
11     y.levo = z.levo
12     y.levo.r = y
```



Primer

Posmatra se sletanje aviona na jednu pistu i održavaju se planirana vremena sletanja $t_i, i = 1,2, \ldots$

- Operacije:
  - $NajavaSletanja(t)$ – dodaje buduće sletanje najavljeno za trenutak t pod uslovom da je odobreno (povratna vrednost može biti odobreno/neodobreno)
  - $Sleteo(t)$ – avion je sleteo i uklanja se ranije najavljeni trenutak t
- Komplikacija (ograničenje): sva sletanja moraju biti planirana u razmacima od najmanje k minuta.
- Cilj: projektovati efikasno rešenje gde se sve operacije izvršavaju u složenosti $O(\log_2 n)$ gde je n broj najava u sistemu.



$k = 3$

$NajavaSletanja(42)$	✗
$NajavaSletanja(43)$	✗
$NajavaSletanja(20)$	✗
$NajavaSletanja(46)$	✗
$NajavaSletanja(55)$	✓

Primer: Analiza mogućih rešenja

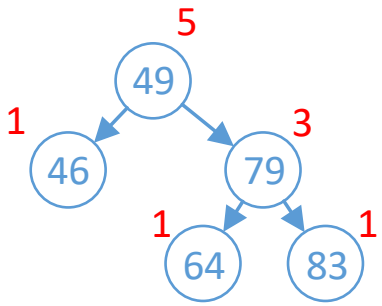
Trajanje operacija:

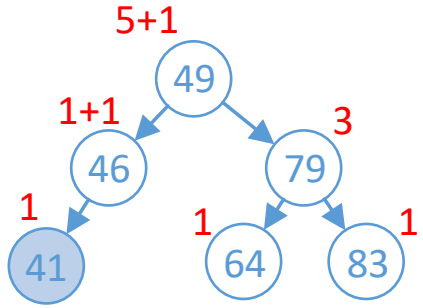
- Nesortirana lista/niz
  - dodavanje $O(1)$; provera ograničenja $O(n)$;
- Sortirani niz
  - binarna pretraga $O(\log_2 n)$ + provera ograničenja $O(1)$, ali ubacivanje $O(n)$;
- Sortirana lista
  - pretraga $O(n)$ + provera ograničenja $O(1)$; ubacivanje $O(n)$;
- min/max-Heap
  - ubacivanje $O(\log_2 n)$; provera ograničenja $O(n)$;
- Rečnik (ili mapa)
  - ubacivanje $O(1)$; provera ograničenja $\Omega(n)$;
- BSP
  - ubacivanje sa proverom $O(h)$; dodavanje $O(h)$
- ...

Uopšteno posmatrano (za rešavanje programerskih problema) neophodno je poznavanje algoritama i struktura podataka da bi se napravio pravi izbor.

Proširenje primera

- Koliko aviona treba da sleti do datog vremena T?
- Rešenje: u svaki čvor dodati broj čvorova ispod njega (crveni brojevi)
  - Dodavanje i brisanje elemenata treba da modifikuje te brojeve





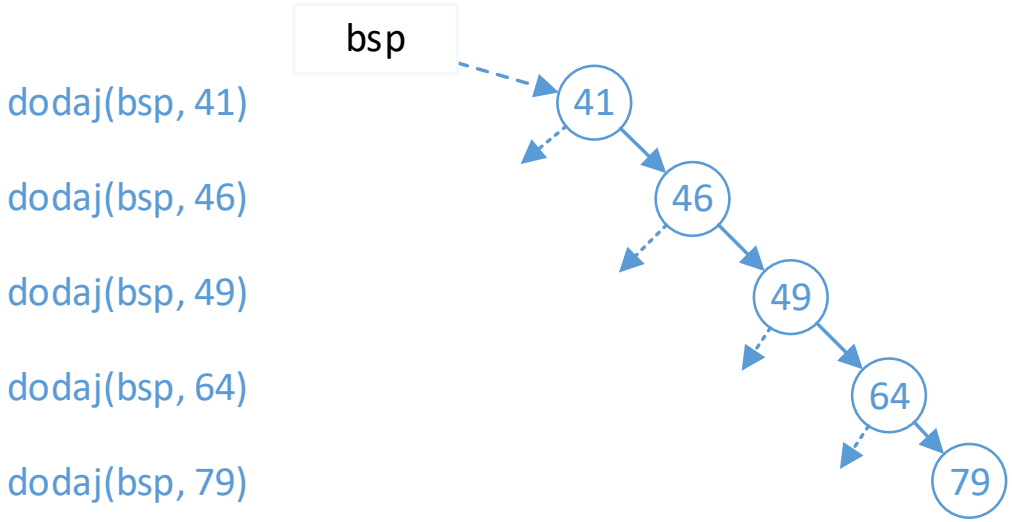
- Algoritam: „silazimo“ niz stablo tako da je $ključ \leq T$
  - Dodajemo 1 za čvor koji testiramo, ako je $ključ \leq T$
  - Kada siđemo desno dodamo veličinu stabla sa leva

Razumevanje algoritama i struktura podataka je osnova za njihovo prilagođavanje praktičnim problemima.

Visina binarnog stabla

- Optimalna visina stabla je $O(\log_2 n)$ i tada kažemo da je stablo **balansirano**
- Međutim, složenost većine operacija sa BSP je $O(h) \geq O(\log_2 n)$
- Nažalost, visina BSP zavisi od redosleda dodavanja elemenata!
  Posledica: Dobija se **nebalansirano** ili **degenerisano stablo**

- Npr. dodavanje sortiranih elemenata dovodi do visine $O(n)$
  - Ovo je najgori slučaj „biranog stabla“ gde ono postaje lista

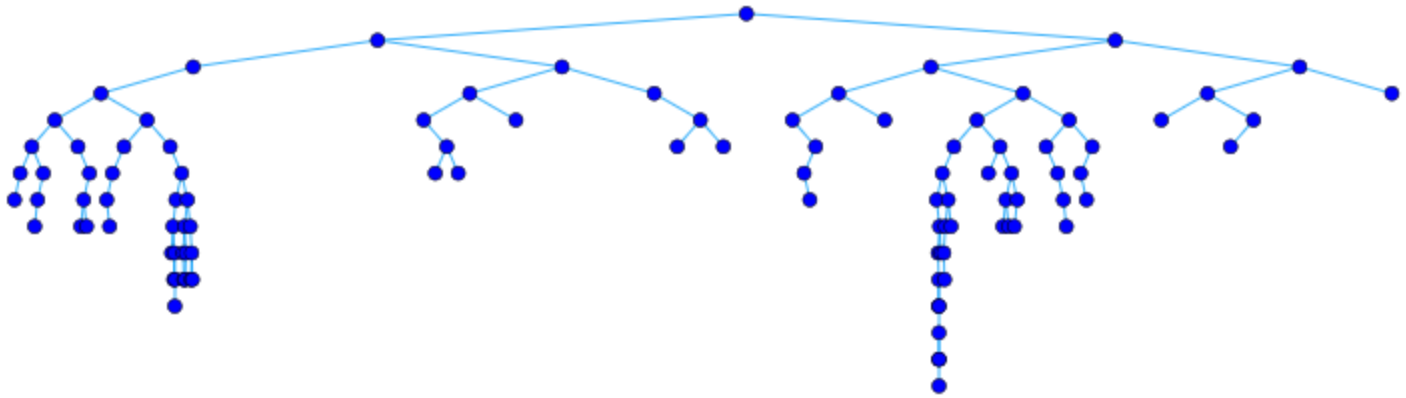


BSP sa jednakim ključevima

- Jednaki ključevi komplikuju implementaciju BSP
  - ako su česti elementi sa jednakim ključevima dobija se degenerisano stablo
- Treba modifikovati metod za dodavanje elementa tako što se posebno proverava jednakost ključeva. Ima nekoliko rešenja:
  1. U svaki čvor se doda bulova promenljiva koja menja vrednost samo kada se doda isti ključ. Npr. kada je neistina novi čvor se doda levo i vrednost se promeni na istinu, i obrnuto.
  2. Umesto dodate bulove promenljive slučajno se bira strana na koju se dodaje elemenat
  3. Čvorovi sa istim ključem se organizuju u listu

BSP izgrađeno od slučajno generisanih ključeva

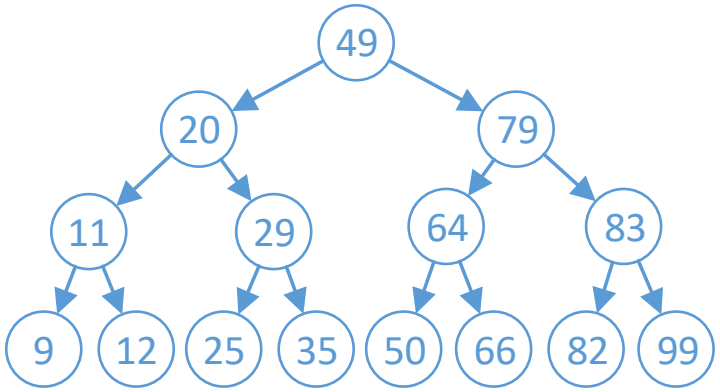
- Visina BSP varira od slučaja do slučaja.
- Može se pokazati da je očekivana visina BSP izgrađenog od slučajno generisanih ključeva $O(\log_2 n)$
  - Ne dobija se idealno balansirano stablo



Balansirano stablo

- Sve operacije sa BSP su trajanja $O(h)$ gde je h visina stabla
  - h je između $\log_2 n$ i n
- Motiv: balansirano stablo održava $h = O(\log_2 n)$
- Posledica: Sve operacije sa BSP su trajanja $O(\log_2 n)$

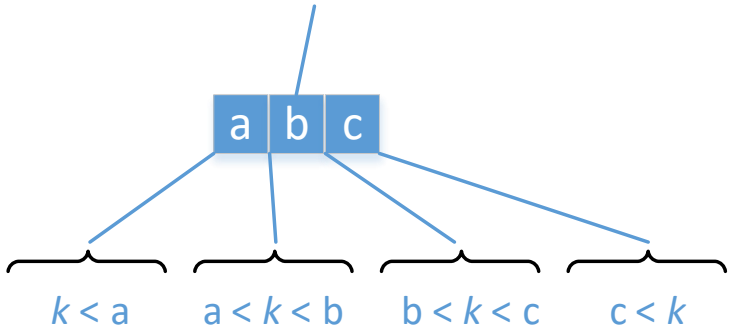
Idealno balansirano stablo

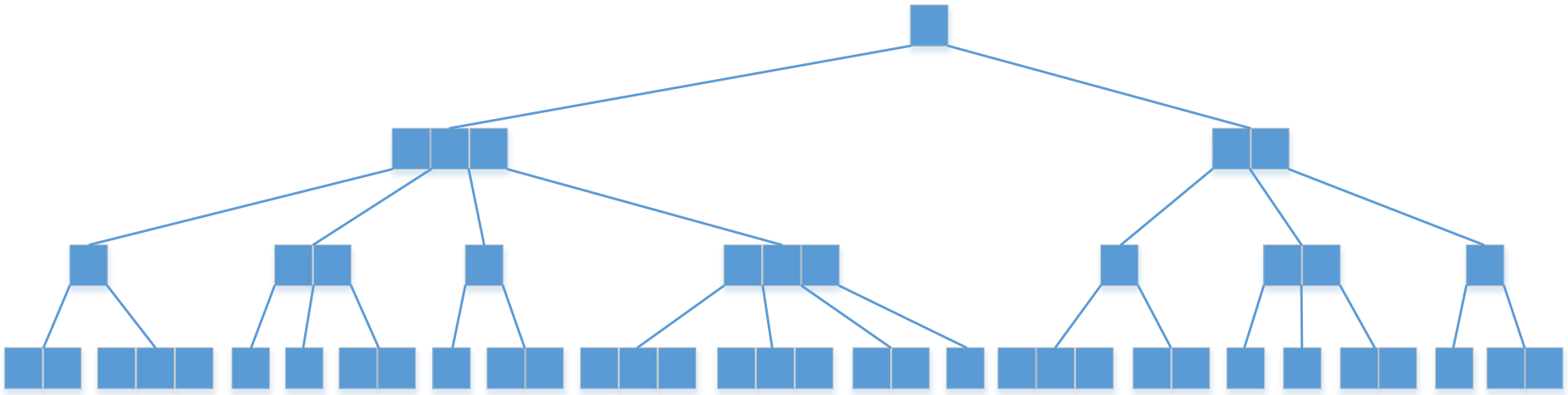


Crveno-crno stablo je binarno balansirano stablo.
Da bi se lakše razumelo, prvo će se posmatrati 2-3-4 stablo.

2-3-4 stablo

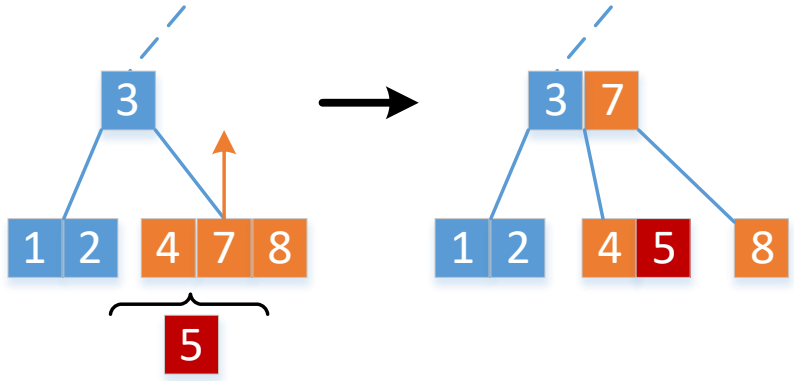
- Više se ne upotrebljava, ali ima istorijsku važnost.
- Sastoji se od čvorova sa po 2 ili 3 ili 4 deteta.
  Znači, svaki čvor ima 1 ili 2 ili 3 ključa.
- Stablo je idealno balansirano (svaki list je podjednako udaljen od korena)
- Pretraga je slična kao kod BSP, ali se kod čvorova sa više ključeva poredi sa svima njima i ako traženi ključ nije među njima prelazi se na odgovarajuće dete u zavisnosti od pridruženog intervala vrednosti ključeva.





Dodavanje u 2-3-4 stablo

- Dodavanje je dozvoljeno samo na poslednji nivo (u lišće).
  - Iako postoji prostor u nekom čvoru na višem nivou ne može se dodati novi ključ jer bi se narušila osobina da je putanja do svakog lista jednaka.
- Dodavanje u čvorove sa 1 ili 2 ključa je trivijalno.
- Ako treba dodati element u čvor sa postojeća 3 ključa, onda se taj čvor „cepa“ na dva čvora, a srednji ključ (element) se prebacuje u roditeljski čvor (u viši sloj).
  - Npr. dodaje se 5 u čvor [4,7,8] pa se 7 prebacuje u roditeljski čvor.
- Da bi se izbeglo naknadno proširivanje roditeljskog čvora (i potencijalno rekurzivno proširivanje sve do korena stabla) u postupku dodavanja, koji kreće od korena, na putanji koja vodi do lišća se pri nailasku na svaki čvor sa 3 ključa vrši njegovo „cepanje“.
  - Na ovaj način se potencijalno pravi prostor koji neće biti popunjen, ali je cilj da se svakako obezbedi prostor za dodavanje elementa na poslednji nivo bez potrebe da se naknadno koriguju gornji novoi.
    (znači, postoji samo jedan prolaz na dole).



Primer dodavanja u 2-3-4 stablo