ALGORITMI I STRUKTURE PODATAKA

STUDIJSKI PROGRAMI:

SOFTVERSKO INŽENJERSTVO, RAČUNARSKA TEHNIKA, INFORMATIKA I MATEMATIKA

NASTAVNIK: DOC. DR ULFETA MAROVAC, UMAROVAC@NP.AC.RS

PRETRAŽIVANJE

- Lociranje željenog podatka u cilju pristupa na osnovu neke identifikacije
- Veoma česta aktivnost – utiče na složenost
- Podaci se nalaze u:
  - tabelama
  - datotekama
- Zapisi identifikovani ključem
- Ključevi:
  - unutrašnji i spoljašnji
  - primarni i sekundarni

PRETRAŽIVANJE

- Po ishodu operacije pretraživanje:
  - uspešno ili
  - neuspešno
- Po mestu pretraživanja:
  - unutrašnje
  - spoljašnje
- Skup koji se pretražuje:
  - statičan ili dinamičan
  - uređen ilineuređen
- Složenija pretraživanja po pripadnosti intervalu ili složenim uslovima (konjuktivnim ilidisjunktivnim)

SEKVENCIJALNO PRETRAŽIVANJE

- Jednostavnost
- Jedini mogući način kod lista ili neuređenih skupova
- Složenost O(n)

SEKVENCIJALNO PRETRAŽIVANJE

- Prolazimo redom kroz članove sve članove niza i ukolko nađemo traženu vrednost vraćamo njen indeks

```
SEQ-SEARCH(K, key)
i = 1
while (i ≤ n) do
    if (key = K[i]) then
        return i
    else
        i = i + 1
    end_if
end_while
return 0
```

SEKVENCIJALNO PRETRAŽIVANJE

- Trženi ključ se postavi na poziciji n+1
- Idemo kroz niz dok ne naiđemo na ključ
- Ako je pozicija pronađenog ključa n+1(pronasli smo kljuc koji smo dodali kao graničnik) onda se u nizu ne nalazi ključ

```
SEQ-SEARCH-SENT(K, key)
K[n + 1] = key
i = 1
while (key ≠ K[i]) do
        i = i + 1
end_while
if (i = n + 1) then
        i = 0
end_if
return i
```

OPTIMIZACIJE

- Ukoliko se ključevi kao argumenti pretraživanja javljaju sa nepoznatim verovatnoćama.
- Najčešće tražene ključeve traba staviti na početak niza
- Za neunifromne verovatnoće pretraživanja pi dobijamo prosečan broj pretraživanja min $\sum$ (i pi) $\Rightarrow$ p1$\geq$ p2 $\geq$ ... $\geq$ pn
- Problemi sa verovatnoćama
  - nisu unapred poznate
  - menjanju se
- Prebacivanje na početak
  - nađeni ključ na prvo mesto
  - oscilacije performansi
- Transpozicija
  - nađeni ključ se pomera za jedno mesto unapred
  - stabilnije performanse

PRETRAŽIVANJE NA VIŠE KLJUČEVA

- Uređena tabela K
- Uređen niz ključeva S
- Složenost O(n)
- P[i] čuva pozicije ključeva
- Tražimo do prvog elemnata koji je veći od ključa jer je K uređena po rastućem poretku

```
SEQ-SEARCH-MUL(K, S)
for i = 1 to m do
    P[i] = 0
end_for
i = j = 1
while (i ≤ n) and (j ≤ m) do
    while (i < n) and (S[j] > K[i]) do
        i = i + 1
    end_while
    if (S[j] = K[i]) then
        P[j] = i
    end_if
    j = j + 1
end_while
return
```

BINARNO PRETRAŽIVANJE

- Sekvencijalno pretraživanje sporo konvergira čak i u uređenim nizovima
- Za uređene nizove rekurzivna taktika “podeli-pobedi”
- Binarno odlučivanje polovi interval pretraživanja
- Rekuzivna i iterativna realizacija
- Neprimenljivo za ulančane liste (čak i uređene)
- Vremenska složenost O(log n)

BINARNO PRETRAŽIVANJE

low,high donja i gornja granica u nizu

Pronalazimo središnji element *K*[*mid*] ako je jednak traženom ključu vraćamo njegovu poziciju mid

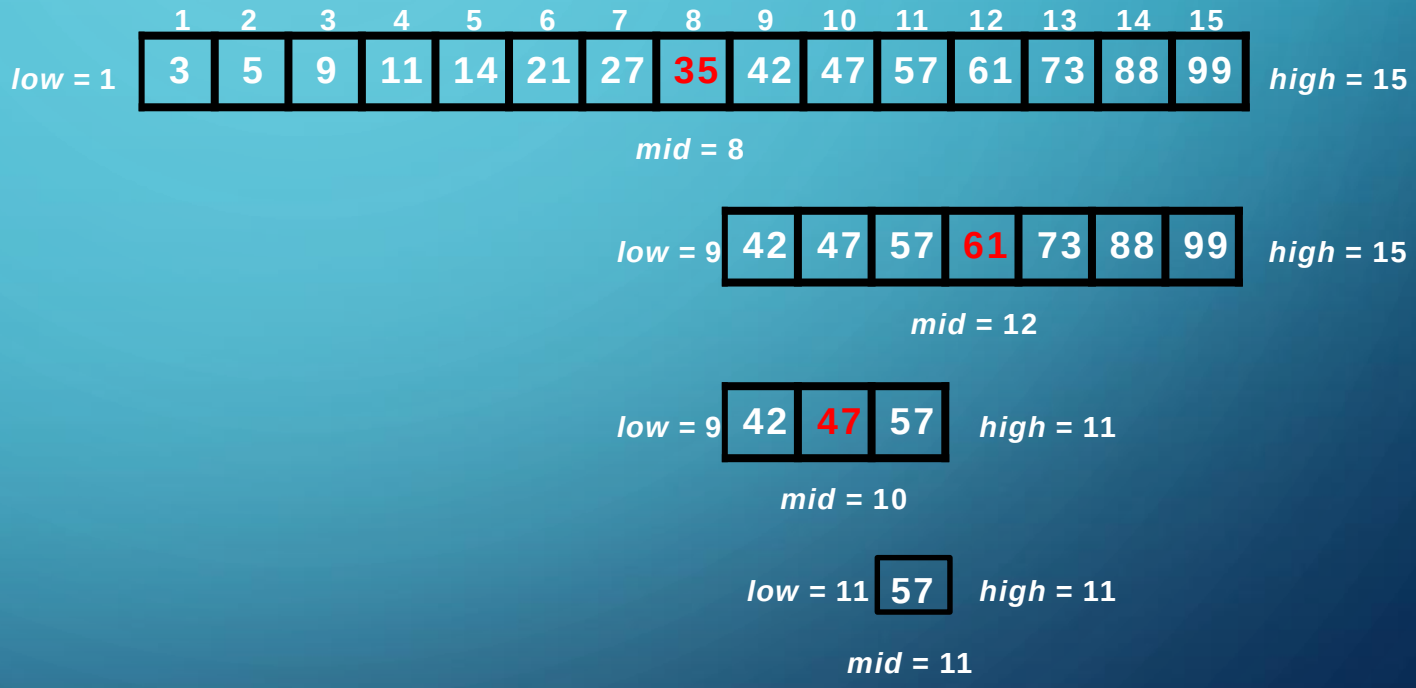
key < *K*[*mid*] element se nalazi u prvoj polovini niza => *high* = *mid* - 1

key > *K*[*mid*] element se nalazi u drugoj polovini niza =>*low* = *mid* + 1

```
BIN-SEARCH(K, key)

low =1

high = n
 while (low ≤ high) do  mid = (low +
 high)/2  if (key = K[mid]) then

          return mid

    else if (key < K[mid]) then

                high = mid - 1

          else

                low = mid + 1

        end_if  end_if

end_while
return 0
```

PRIMER

- Pretraživanje niza za vrednost 57

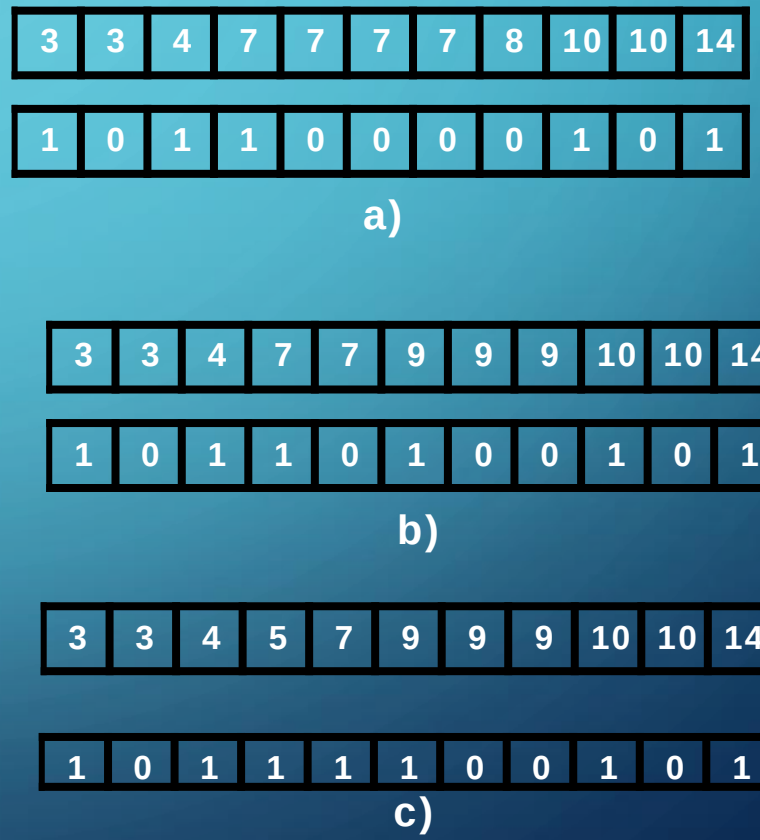


VARIJANTE BINARNOG PRETRAŽIVANJA

- Binarno pretraživanje u tabeli nepoznate veličine
  - ne zna se sredina
  - binarno se odredi interval (i...2i) - O(log i)
  - standardno pretraživanje u intervalu - O(log i)
  - efikasno za pretraživanje na početku i kraju
- Binarno pretraživanje u povećanoj tabeli
  - problem umetanja i brisanja ključeva
  - maksimizacija veličine tabele
  - vektor validnosti
  - “prividni ključevi” na slobodnim pozicijama
  - (= ili >od prvog prethodnog ključa, a < od prvog narednog validnog ključa)

VARIJANTE BINARNOG PRETRAŽIVANJA

- Operacije
  - pretraživanje (konsultovanje bita validnosti)
  - umetanje (moguća replikacija i pomeranja)
  - brisanje (resetovanje bita validnosti)



VARIJANTE BINARNOG PRETRAŽIVANJA

- Interpolaciono pretraživanje
  - binarno pretraživanje neosetljivo na sadržaj
  - za uniformnu raspodelu ključeva može i bolje
  - pozicija poređenja
  - low + (high - low)(key – K[low])/(K[high] - K[low])
  - performanse i do O(log log n)
  - može biti pogodno za spoljašnje pretraživanje
  - za neuniformne raspodele nije dobro

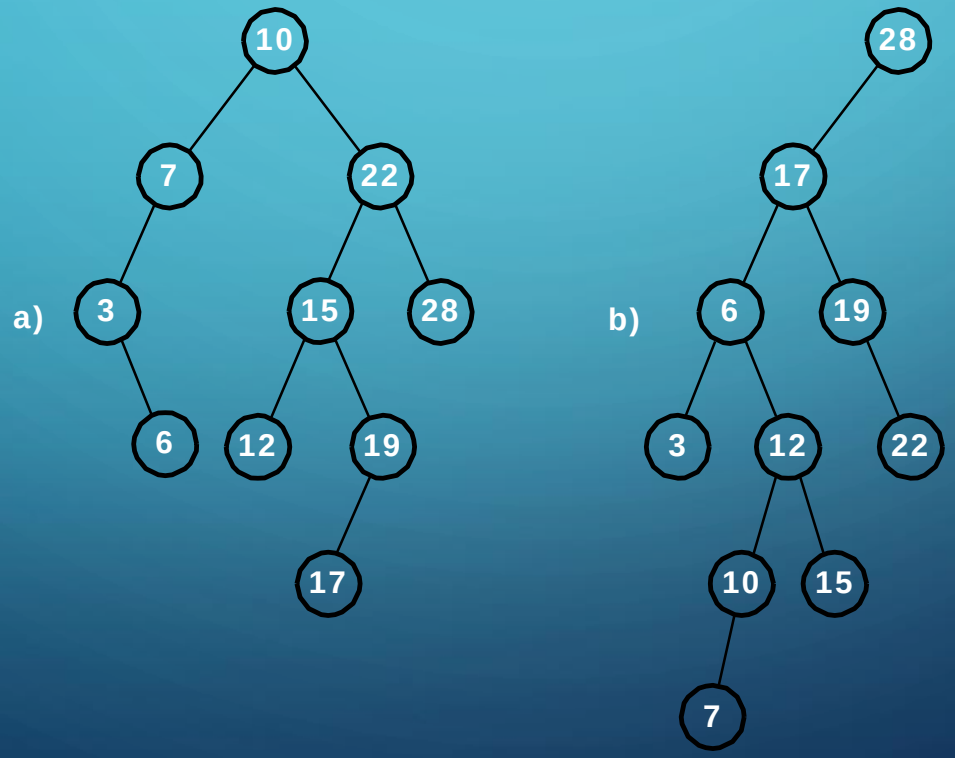
- Fibonacci-jevo pretraživanje
  - asimetrična deoba intervala
  - u odnosu Fibonacci-jevih brojeva
  - performanse O(log n), ali nešto lošije

STABLO BINARNOG PRETRAŽIVANJA

- BP nepogno za dinamičke tabele
  - Rešenje – binarno stablo
  - dinamička struktura
  - uređenje po sadržaju
- Stablo binarnog pretraživanja (BST)
  - za svaki ključ K najviše jedan čvor key(p) = K
  - u levom podstablu važi key(pl) < K
  - u desnom podstablu važi key(pr) > K
- Inorder daje sortirani poredak

STABLO BINARNOG PRETRAŽIVANJA

- Isti *inorder* poredak, različita stabla



PRETRAŽIVANJE BST

- Pronalaženje najmanjeg ključa u stablu (najlevlji čvor)
- Nalaženje najvećeg ključa u stablu (najdesniji čvor)

```
BST-MIN(root)
p = root
while (left(p) ≠ nil) do
        p = left(p)
end_while
return p
```

```
BST-MAX(root)
p = root
while (right(p) ≠ nil ) do
        p = right(p)
end_while
return p
```

ISPITIVANJE BST

- Nalaženje sledbenika po inorderu
- Pokazivač na oca - parent

```
BST-SUCC(r)
p = r
if (right(p) ≠ nil) then
      return BST-MIN(right(p))
else
      q = parent(p)
      while (q ≠ nil and p = right(q)) do
            p = q
            q = parent(q)
      end_while
      return q
end_if
```

UMETANJE U BST

```
BST-INSERT(new, root)
p = root
if (p = nil) then
        root = new
else
        if (key(new) = key(p)) then
                ERROR(Postoji ključ)
        else if (key(new) < key(p)) then
                        BST_INSERT(new, left(p))
                else
                        BST_INSERT(new, right(p))
                end_if
        end_if
end_if
```

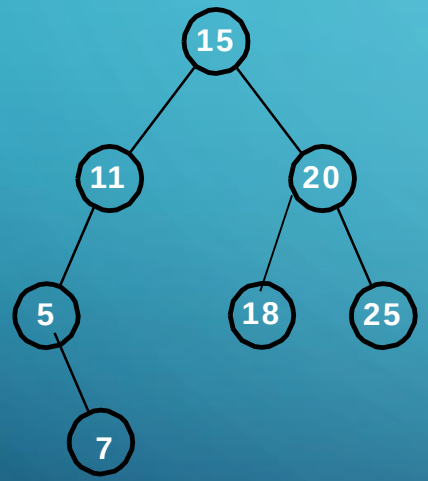
UMETANJE U BST

```
BST-INSERT-I(new, root)
p = root
q = nil
while (p ≠ nil) do
    q = p
    if (key(new) = key(p)) then
        ERROR(Postoji ključ)
    else if (key(new) < key(p)) then
        p = left(p)
    else
        p = right(p)
    end_if
end_if
end_while
```

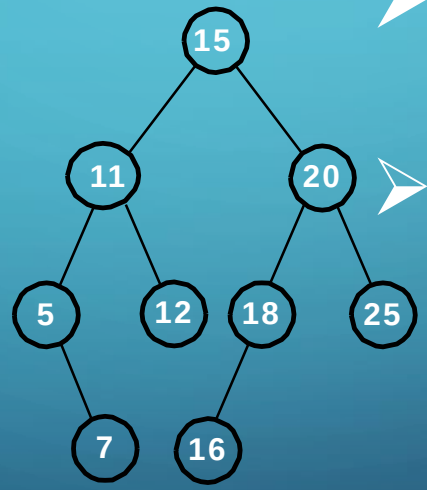
```
if (q = nil) then
    root = new
else
    if (key(new) < key(q))
then
        left(q) =new
    else
        right(q) = new
    end_if
end_if
```

- Složenost O(*h*)

UMETANJE U BST



a)



b)

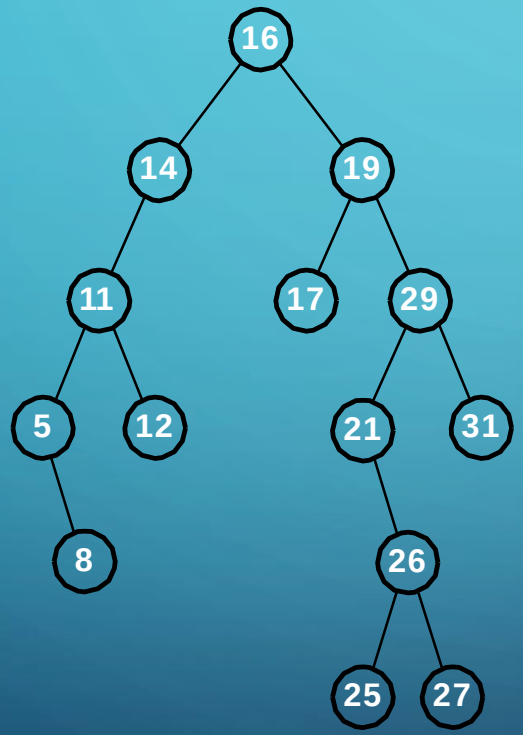
- Primer – umetanje ključeva 12 i 16
- Sa mogućnošću dodavanja istih ključeva,
- može se koristiti i kao efikasan algoritam sortiranja

BRISANJE IZ BST

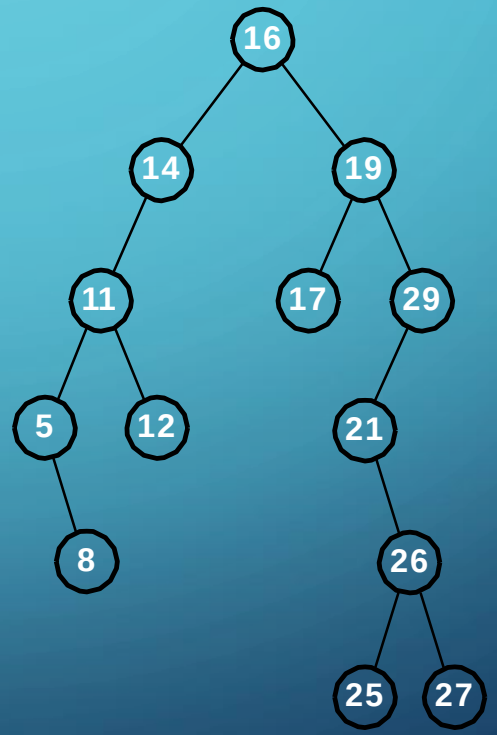
- Brisanje lista
- Brisanje čvora sa jednim podstablom
  - ✓ preuzima ga otac
- Brisanje čvora sa dva podstabla
  - ✓ zamenjuje ga sledbenik (ili prethodnik)
- Složenost O(h)

BRISANJE IZ BST

- Brisanje(31)

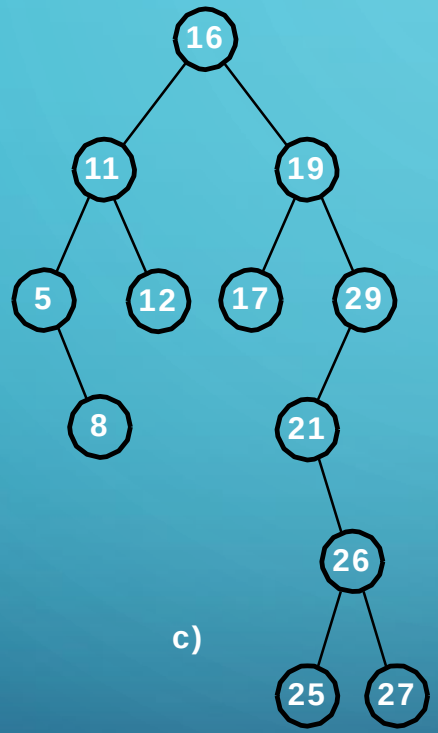


a)

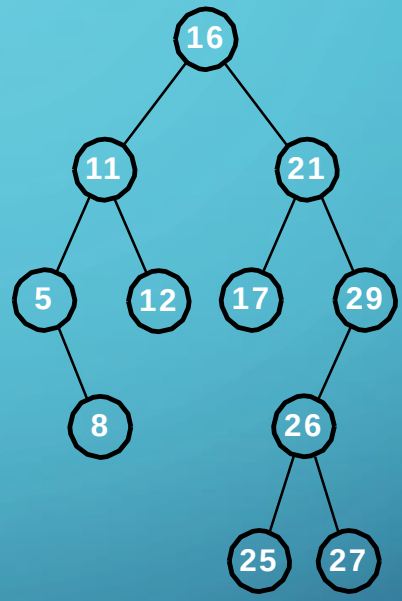


b)

STABLA



c)

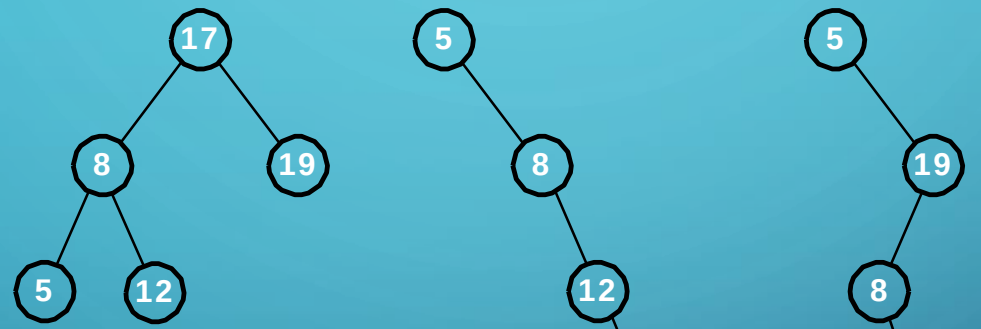


d)

- 14

- 19

ANALIZA PERFORMANSI BST



a)

b)

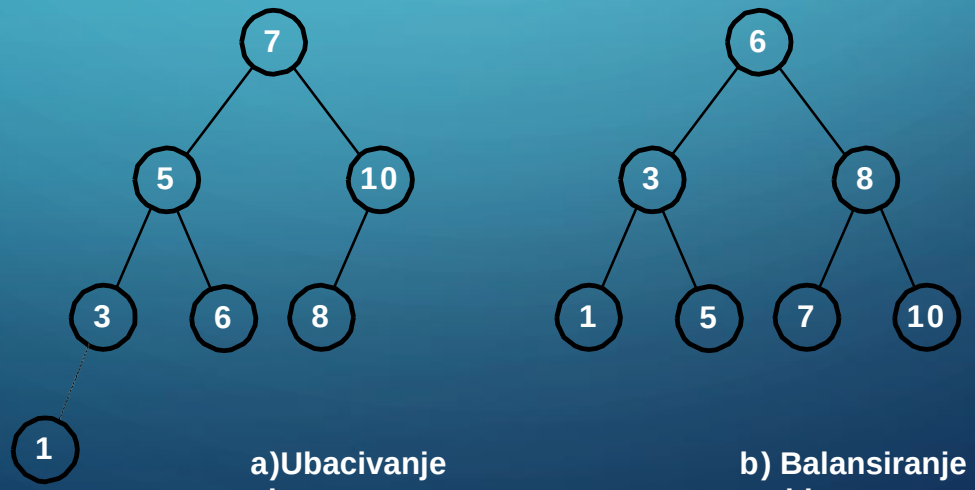
c)

ANALIZA PERFORMANSI BST

- Pretpostavke
  - ✓ *n* ključeva sa jednakom verovatnoćom pristupa
  - ✓ "slučajno" BST nastalo umetanjima
    - u proizvoljnom poretku
- Zaključak
  - ✓ prosečna performansa pretraživanja
    - u slučajnom stablu proizvoljne topologije istog reda (O(log *n*)) kao i u optimalnom
    - i lošija samo za konstantan faktor < 40% !

BALANSIRANJE STABLA

- Balansirano stablo
  - ✓ optimalno za pretraživanje
  - ✓ održavanje balansa može biti skupo
- Suboptimalno rešenje – “skoro balansirana” stabla



a)Ubacivanje elementa

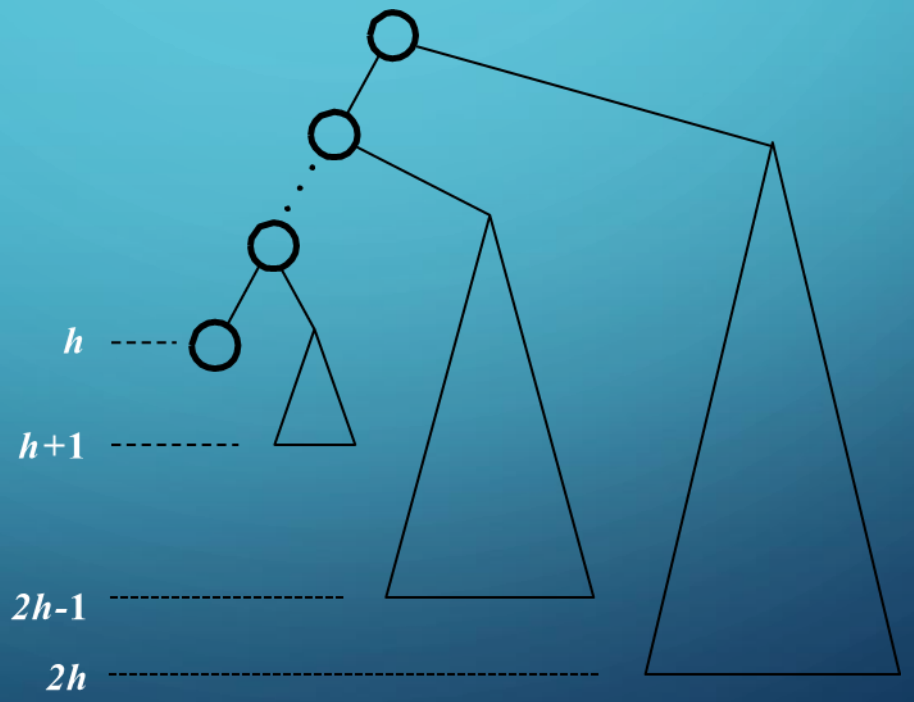
b) Balansiranje stabla

AVL STABLA

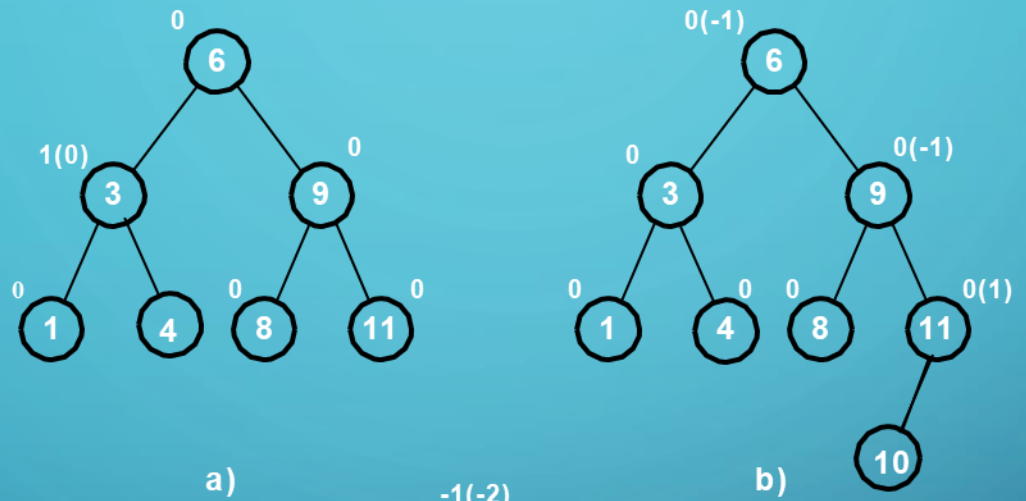
- **A**delson-**V**elski, **L**andis
- AVL je stablo binarnog pretrazivanja kod koga za svaki cvor važi da se visina levog od desnog podtabla razlikuje najvise za jedan
- Balans čvora – $b = h_l - h_r$
- Visinski balansirano stablo (AVL)
  - stablo binarnog pretraživanja
  - za svaki čvor b može biti samo –1, 0 ili 1

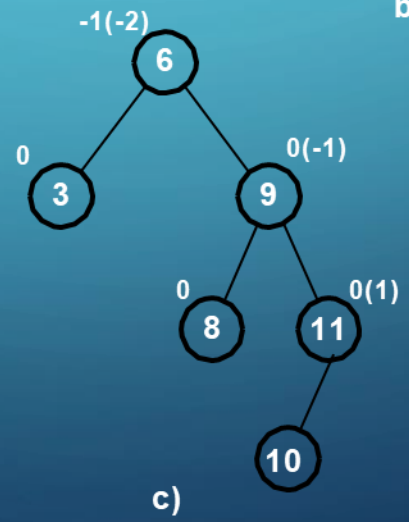
AVL STABLA

- AVL stablo sa najvećim rasponom nivoa



UMETANJE U AVL STABLO



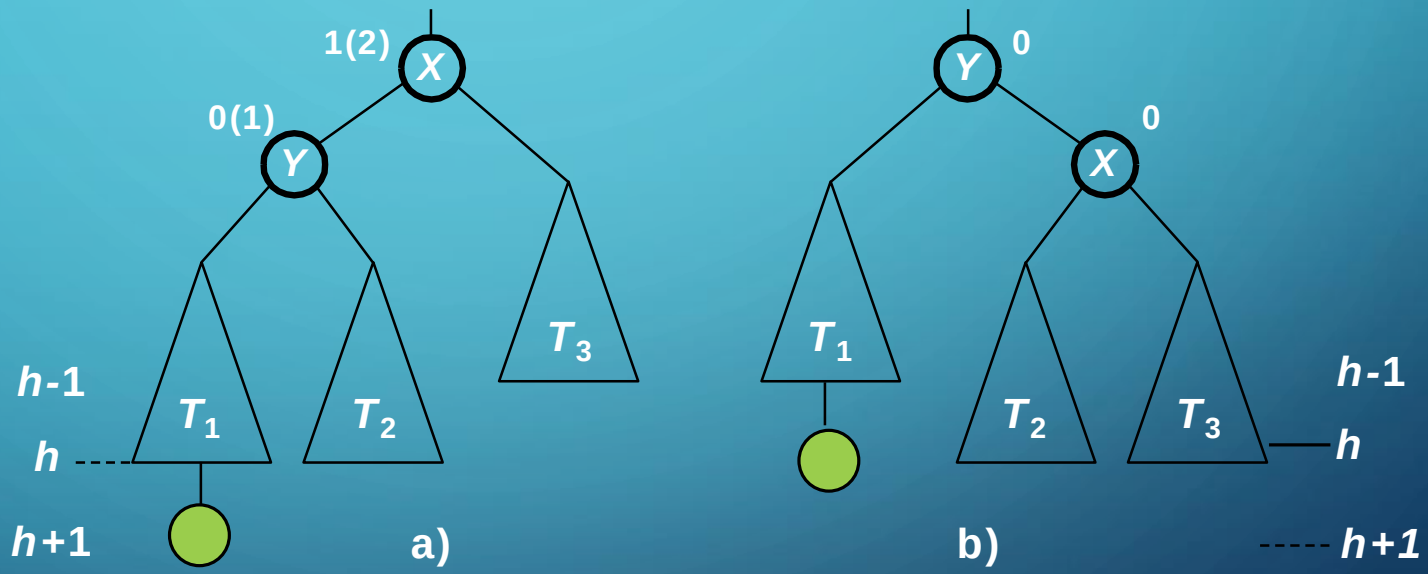


UMETANJE U AVL STABLO

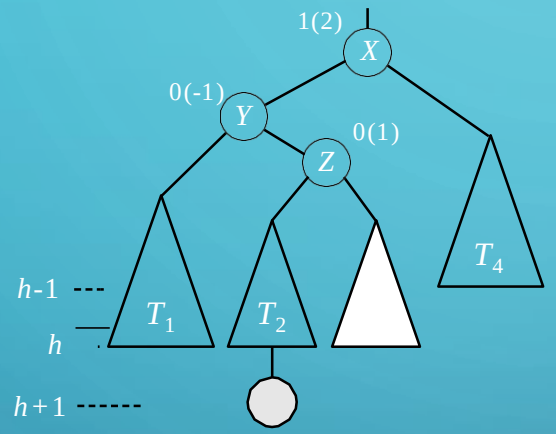
- Umetanje kao u BST
- Kritični čvor
- Dva karakteristična slučajeva
  - ✓ kritični čvor i njegov sin na strani umetanja se naginju na istu stranu
  - ✓ kritični čvor i njegov sin na strani umetanja se naginju na suprotnu stranu
- Rotacije (jednostruke ili dvostruke) oko kritičnog čvora
  - ✓ ispravljaju balans
  - ✓ održavaju *inorder* poredak

JEDNOSTRUKA ROTACIJA

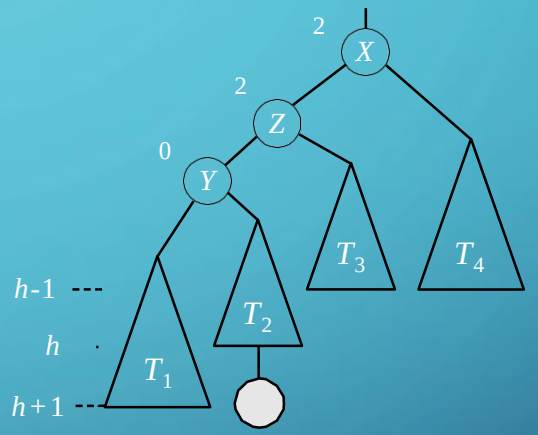
RIGHT-ROTATION(*x*)
y = *left*(*x*)
temp = *right*(*y*)
right(*y*) = *x*
left(*x*) = *temp*



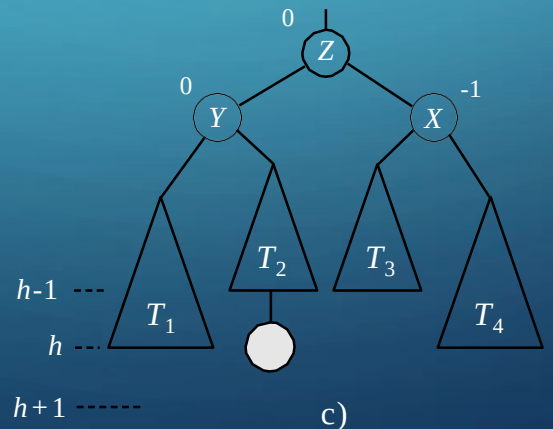
DVOSTRUKA ROTACIJA



a)



b)



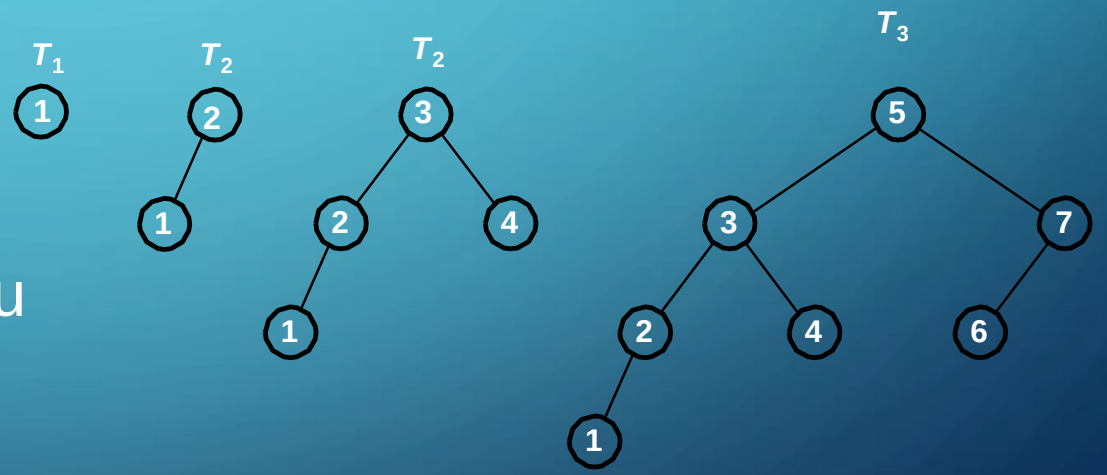
c)

PERFORMANSE AVL STABLA

- Održavanje relativno jednostavno
  - ✓ najviše jedna dvostruka rotacija kod umetanja
  - ✓ najviše jedna rotacija po nivou kod brisanja
- Performansa pretraživanja
  - $\log (n + 1) < h_{AVL}(n) < 1.4404 \log (n + 2) - 0.328$
- Zaključak
  - ✓ Garantovana performansa pretraživanja malo ispod optimalne
  - ✓ Jeftine i ne mnogo česte operacije balansiranja

FIBONACCI-JEVA STABLA

- Najveća visina za dati broj čvorova *n*
- $n_0 = 1, n_1 = 2, \ldots, n_h = n_{h-1} + 1 + n_{h-2}$
- Balans čvorova grananja
- Svako brisanje smanjuje visinu



TEŽINSKI BALANSIRANA STABLA

- Skoro” balansirana stabla
- Kriterijum - težina podstabla (broj eksternih čvorova)
- Za svaki čvor odnos težina podstabala u opsegu a i $1-a$
- Balansiranje preko rotacija
- Za $a \approx 0.5$ dobar balans, ali zahtevnije održavanje
- Performanse - $O(\log n)$

PRIMENE BST

- Predstavljanje prioritetnog reda
- Vektorska i ulančana reprezentacija neefikasne - O(n)
- Realizacija preko stabla binarnog pretraživanja uz omogućavanje postojanja istih ključeva
- Performanse operacija umetanja i brisanja - O(log n) ako se stablo održava balansiranim
- Fleksibilnije od *heap*-a:
  - ✓ istovremeno i rastući i opadajući red
  - ✓ efikasno brisanje niza ključeva iz zadatog opsega

TEST PITANJA

1. Šta je pretraživanje? Kakva pretraživanja razlikujemo?
2. Kako se može optimizovati sekvencijalno pretraživanje?
3. Dajte primer zasto je SEQ-SEARCH-SENT(K, key) efikasnija od SEQ-SEARCH(K, key)
4. Na primeru objasnite kako se izvršava binarno pretraživanje
5. Kako dolazimo do minimalnog a kako do maksimalnog elementa stabla
6. Na primeru pokazite kako se vrši ubacivanje elementa u binarno stablo
7. Pokazite na primeru razlicite slucajeve izbacivanja elementa iz stabla
8. Šta je balansirano stablo? Zašto je ovo bitno kod balansiranja
9. Da li je AVL stablo balansirano?
10. Dajte primer AVL stable I dodajte cvor pa izvrsite odgovarajucu rotacu. Pokozite primerom I jednostruku I dvostruku rotaciju