



STRUKTURE PODATAKA
LETNJI SEMESTAR

STABLA BINARNOG TRAŽENJA
SORTIRANA/UREĐENA BINARNA STABLA

Prof. Dr Leonid Stoimenov
Katedra za računarstvo
Elektronski fakultet u Nišu

STABLA BINARNOG TRAŽENJA - PREGLED

- Definicija
- Operacije
- Primeri
- Gomila (Heap)
- Operacije

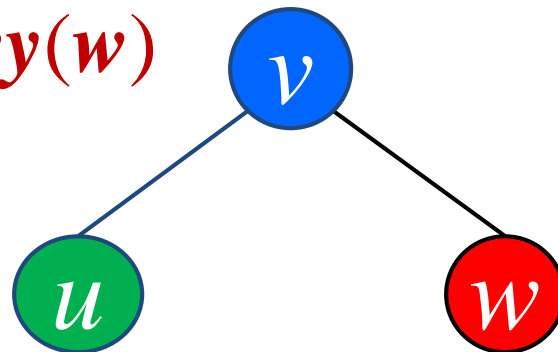
POJAM STABLA BINARNOG TRAŽENJA

- Stablo binarnog traženja (SBT) je binarno stablo koje čuva ključeve (ili parove ključ-element) u čvorovima



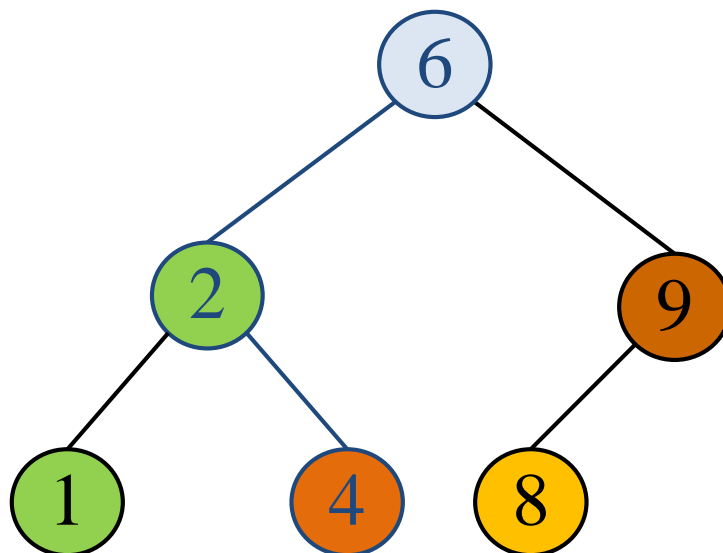
- i za koje važi:
 - Neka su u , v , i w tri čvora, takva da je u u levom podstablu čvora v , a w je u desnom podstablu v . Tada važi da je

$$\textit{key}(u) < \textit{key}(v) \leq \textit{key}(w)$$



POJAM STABLA BINARNOG TRAŽENJA (2)

- **Stablo binarnog traženja** (SBT) - Navedeni uslov važi za sve čvorove u stablu, odnosno za celo levo i desno podstablo:
- Vrednost čvora je
 - **veća** od svih vrednosti *levog podstabla* i
 - **manja ili jednaka** od vrednosti *desnog podstabla*

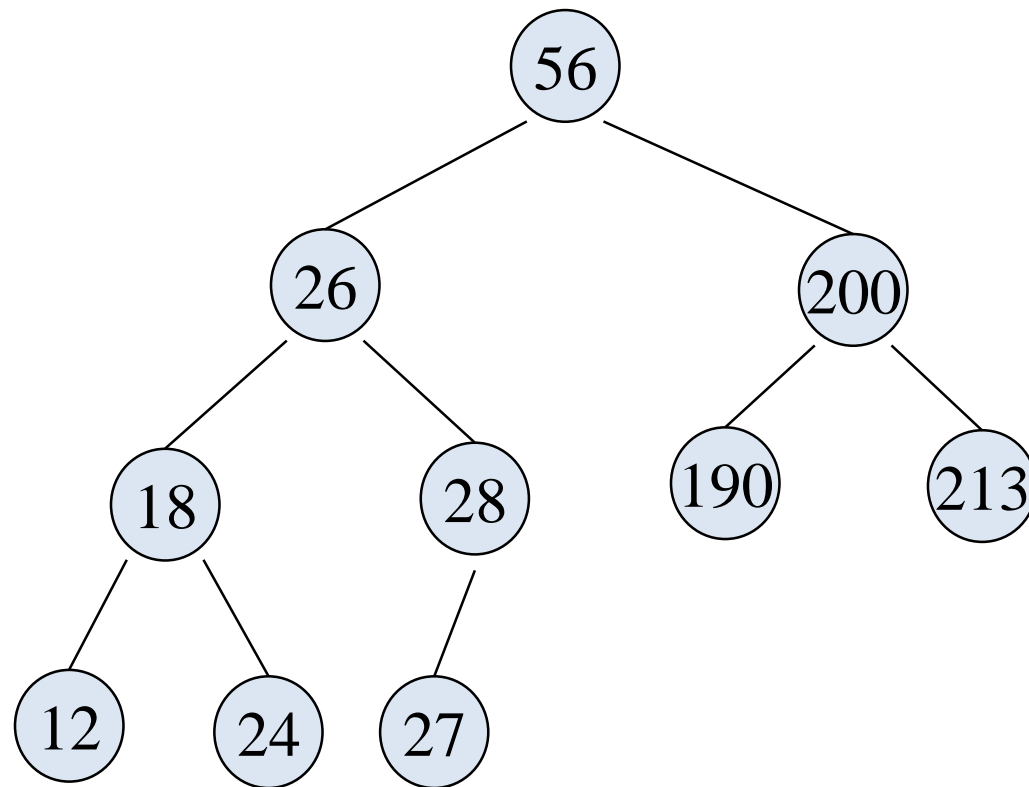


PRIMER

STABLA BINARNOG TRAŽENJA

- Na osnovu niza formirati stablo bin. traženja:

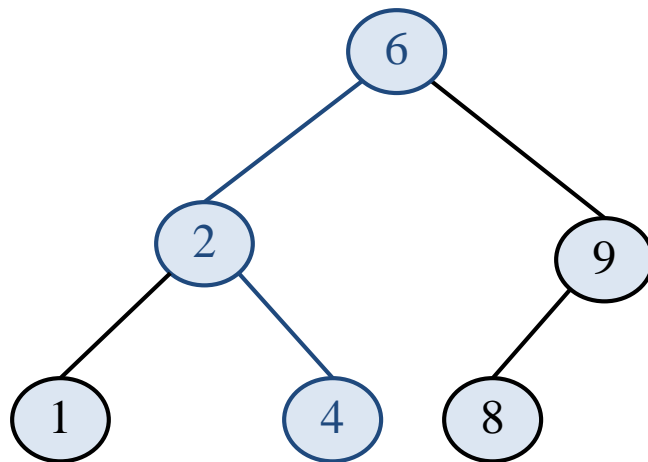
56, 26, 28, 200, 18, 24, 190, 213, 27, 12



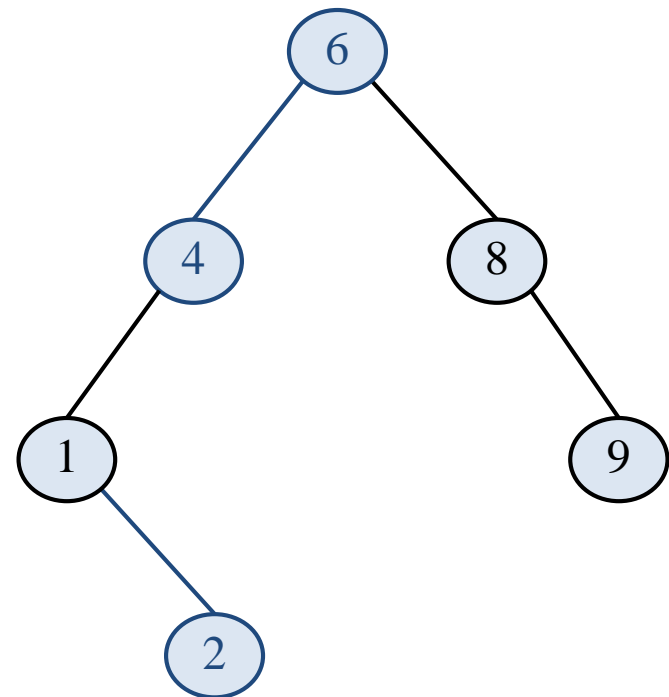
PRIMER

STABLA BINARNOG TRAŽENJA

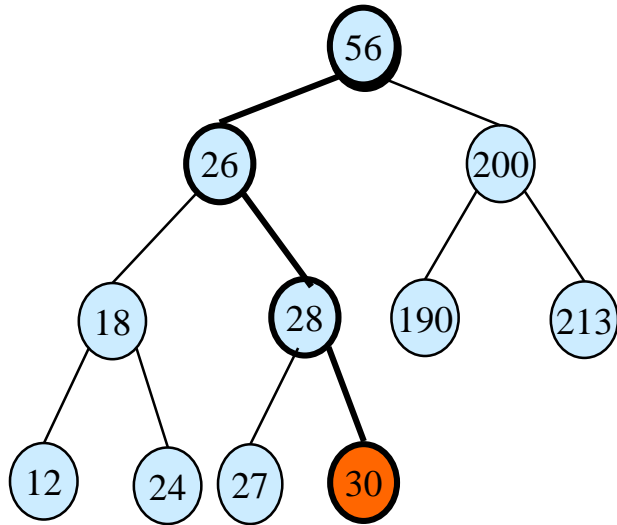
Niz: 6, 9, 2, 1, 4, 8



Niz: 6, 8, 4, 1, 2, 9



OPERACIJA DODAVANJA



- Inicijalizacija: $O(1)$
- While petlja u linijama 3-8 traži mesto za ubacivanje z , vodeći računa o roditelju y
Zahteva vreme $O(h)$
- Linije 9-14 dodaju vrednost: $O(1)$

⇒ UKUPNO: $O(h)$

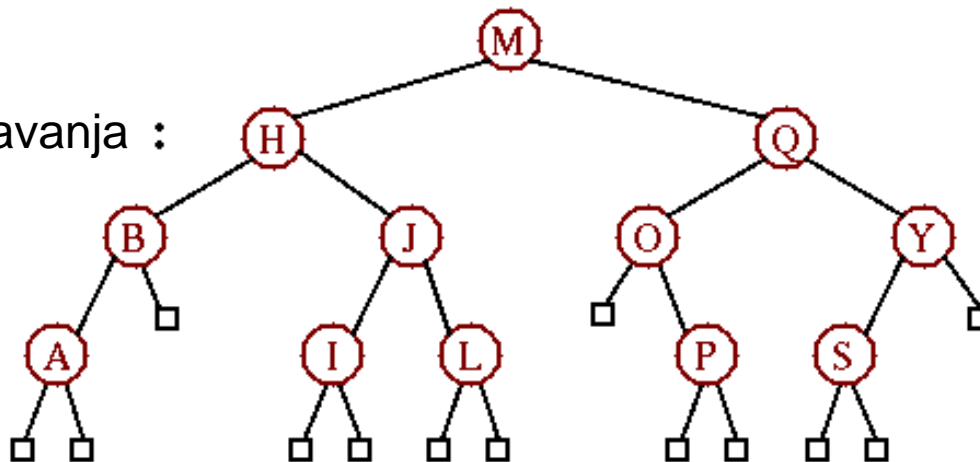
Algoritam SBT.3. Dodavanje Tree-Insert(T, z)

1. $y \leftarrow \text{null}$
2. $x \leftarrow \text{root}(T)$
3. **while** ($x \neq \text{null}$)
4. $y \leftarrow x$
5. **if** ($\text{key}(z) < \text{key}(x)$)
6. **then** $x \leftarrow \text{left}(x)$
7. **else** $x \leftarrow \text{right}(x)$
8. **end while**
9. $p(z) \leftarrow y$ // p -roditeljski čvor
10. **if** ($y = \text{null}$)
11. **then** $\text{root}(T) \leftarrow z$
12. **else if** $\text{key}(z) < \text{key}(y)$
13. **then** $\text{left}(y) \leftarrow z$
14. **else** $\text{right}(y) \leftarrow z$

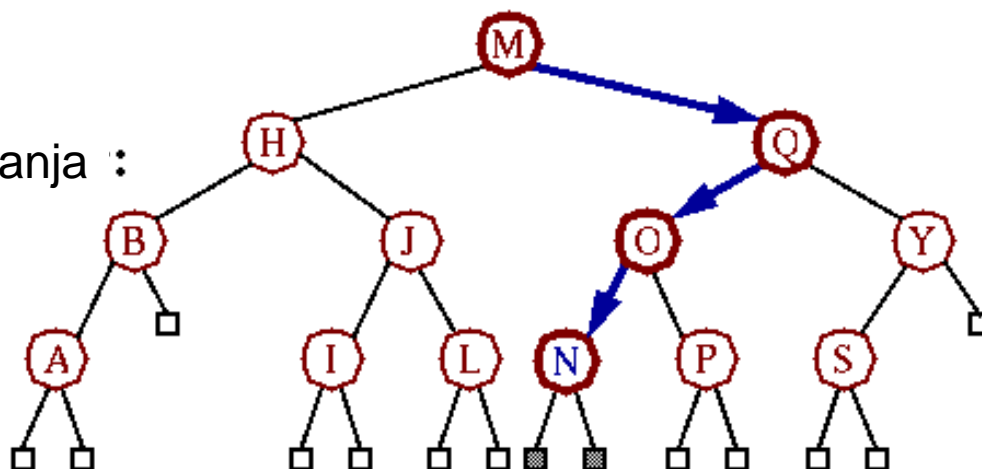
ILUSTRACIJA DODAVANJA

○ Dodavanje elementa N

Pre dodavanja :



Posle dodavanja :

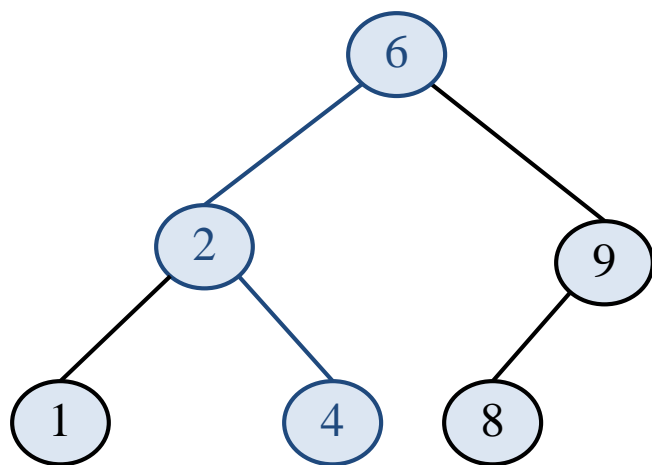


STABLA BINARNOG TRAŽENJA – OPERACIJE

- Osnovne operacije sa stablom:
obilazak stabla
- Dodatne operacije u odnosu na binarna stabla
 - **Traženje** čvora sa ključem k – operacija *find*
 - **Umetanje** – operacija *insertItem*
 - **Brisanje** čvora – operacija *removeElement*

OBILAZAK STABLA BINARNOG TRAŽENJA

- **Inorder** obilazak stabla binarnog traženja
rezultat je **uređeni niz u rastućem poretku**:

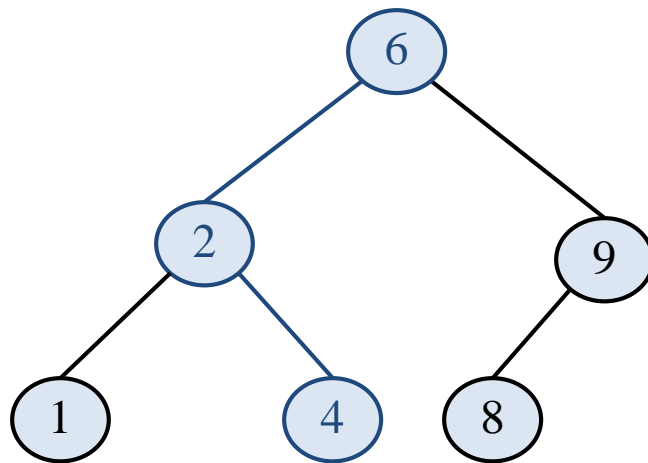


1 2 4 6 8 9

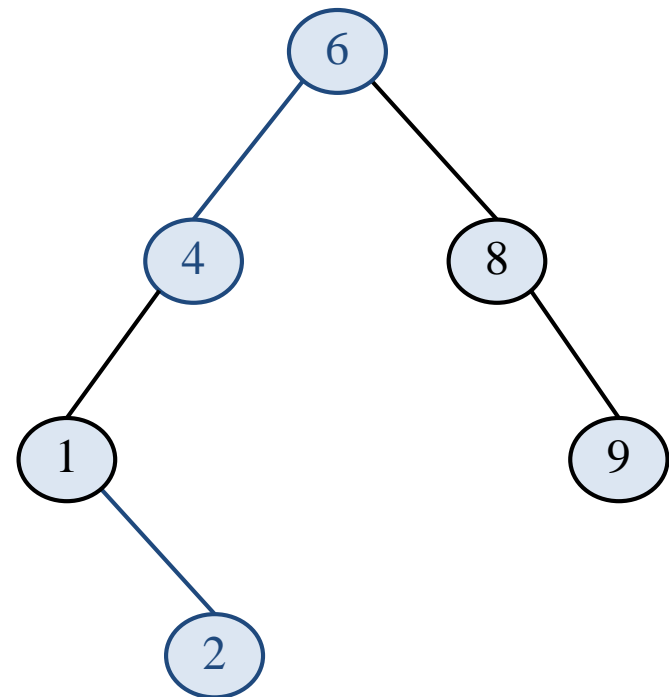
PRIMER

OBILAZAK STABLA BINARNOG TRAŽENJA

Niz: 6, 9, 2, 1, 4, 8



Niz: 6, 8, 4, 1, 2, 9



Obilazak: 1 2 4 6 8 9

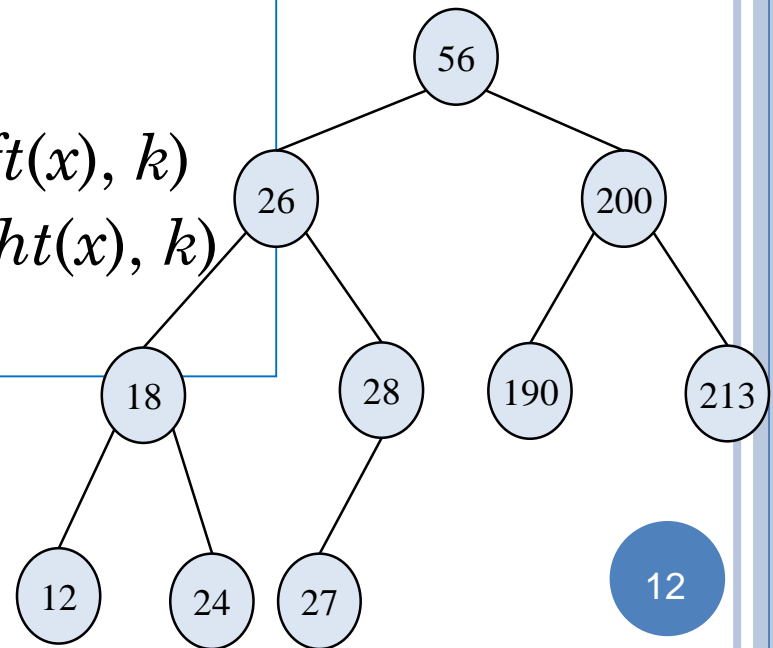
OPERACIJA TRAŽENJA – REKURZIVNA

Algoritam SBT.1. Rekurzivno traženje
Tree-Search(x, k)

1. **if** ($x = \text{null}$ or $k = \text{key}(x)$)
2. **then** return x
3. **if** ($k < \text{key}(x)$)
4. **then** return Tree-Search($\text{left}(x), k$)
5. **else** return Tree-Search($\text{right}(x), k$)

Složenost: $O(h)$

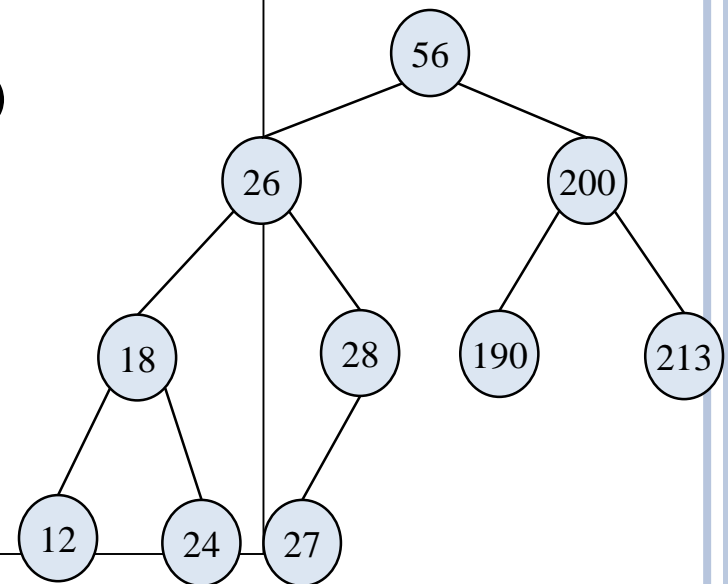
$O(\log n)$, n broj čvorova u stablu



OPERACIJA TRAŽENJA – ITERATIVNA

Algoritam SBT.2. Iterativno traženje
[Iterative-Tree-Search\(\$x, k\$ \)](#)

1. **while** ($x \neq null$ **and** $k \neq key(x)$)
2. **if** ($k < key(x)$)
3. **then** $x \leftarrow left(x)$
4. **else** $x \leftarrow right(x)$
5. **return** x

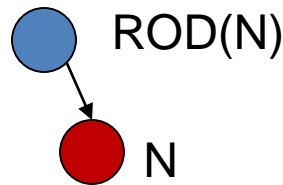


Uglavnom [efikasnija](#) od rekurzivne.

OPERACIJA BRISANJA ČVORA

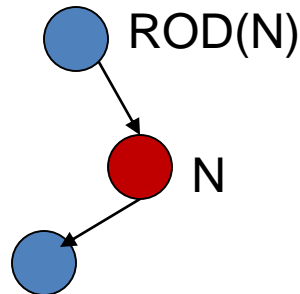
- Brisanje čvora je složeno zato što treba paziti da nakon brisanja ostane zadovoljen **uslov uređenja stabla**
- Čvor N koji se briše treba zameniti čvorom X koji treba pronaći u stablu, tako da se održi uređenje stabla
- **Postoje tri slučaja:**

1. Čvor je list

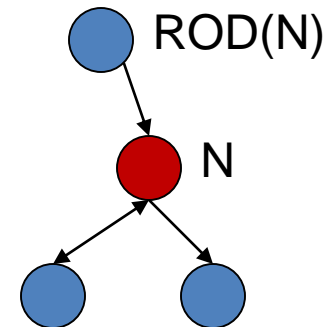


N je čvor koji se briše
ROD(N) je roditelj čvora N

2. Čvor je stepena 1



3. Čvor je stepena 2



OPERACIJA BRISANJA ČVORA

1. Naći čvor N i čvor ROD
2. Obrisati čvor N na sledeći način:

Slučaj 1 (Brisanje iz lista):

null → ROD.link(N)

// link na N roditelja se postavlja na null

Slučaj 2 (brisanje čvora koji ima 1 dete):

N.link(D) → ROD.link(N)

// u roditeljski čvor čvora N se postavlja

// link na dete čvora N

Slučaj 3 (brisanje čvora sa 2 deteta):

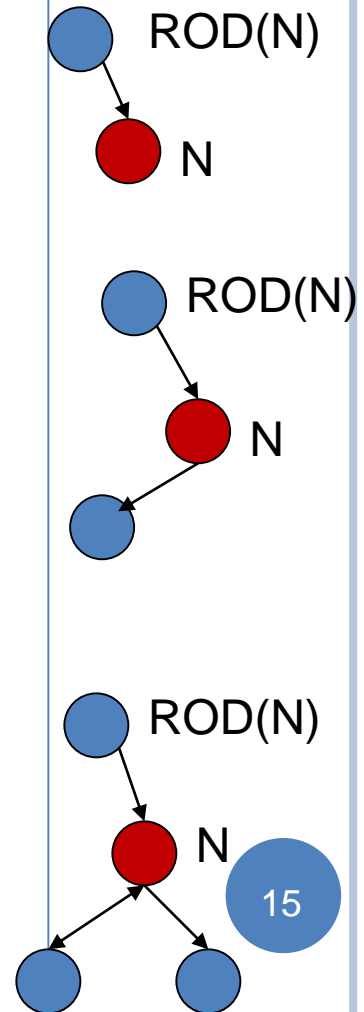
Naći čvor S (**inorder** sledbenik čvora N)

Obrisati čvor S korišćenjem slučaja 1 ili 2

S → ROD.link(N)

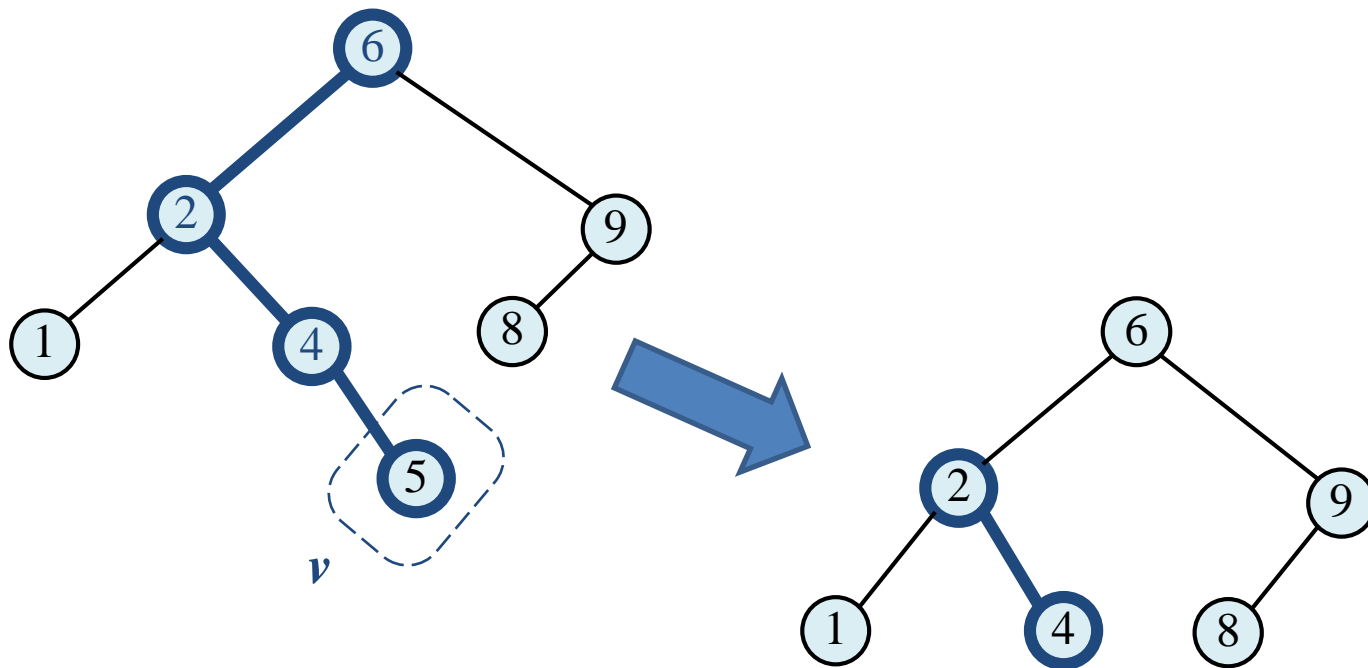
// u roditeljski čvor čvora N se postavlja

// link na inorder sledbenika čvora N



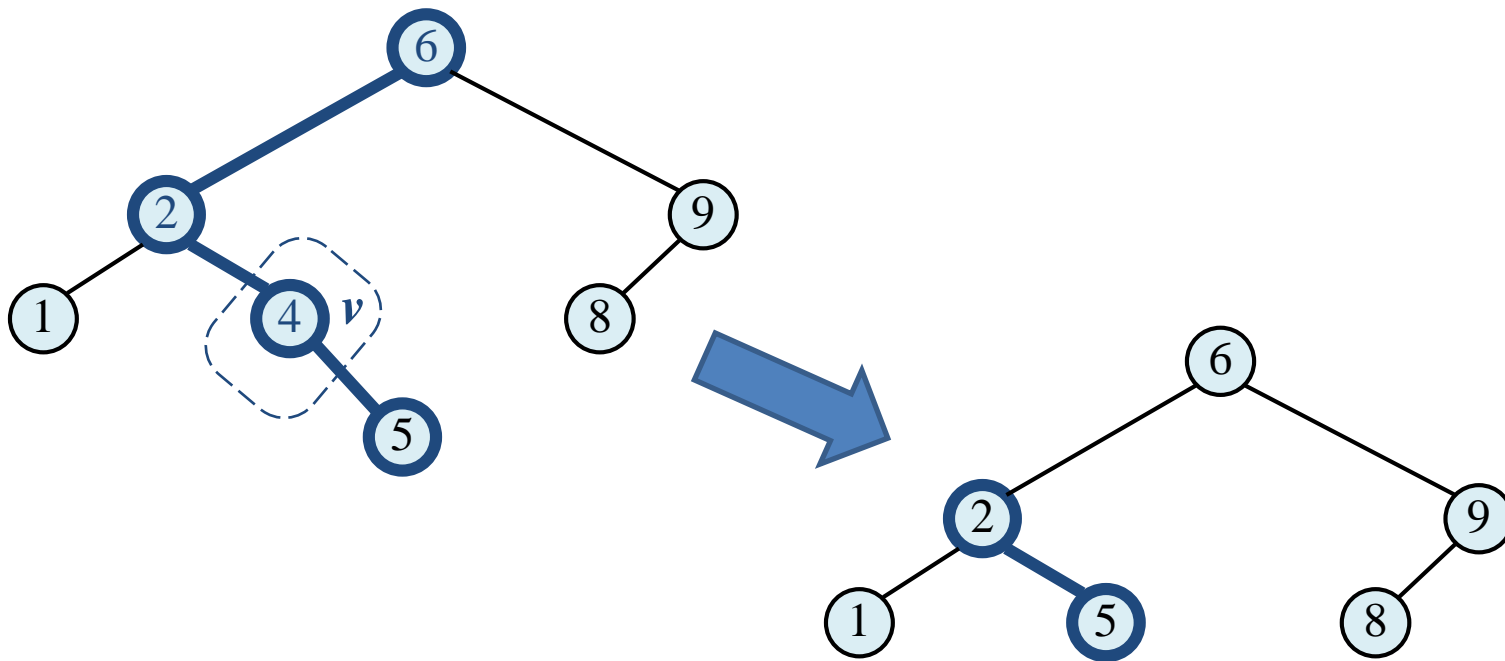
ILUSTRACIJA BRISANJA

- Brisanje lista (čvor 5)



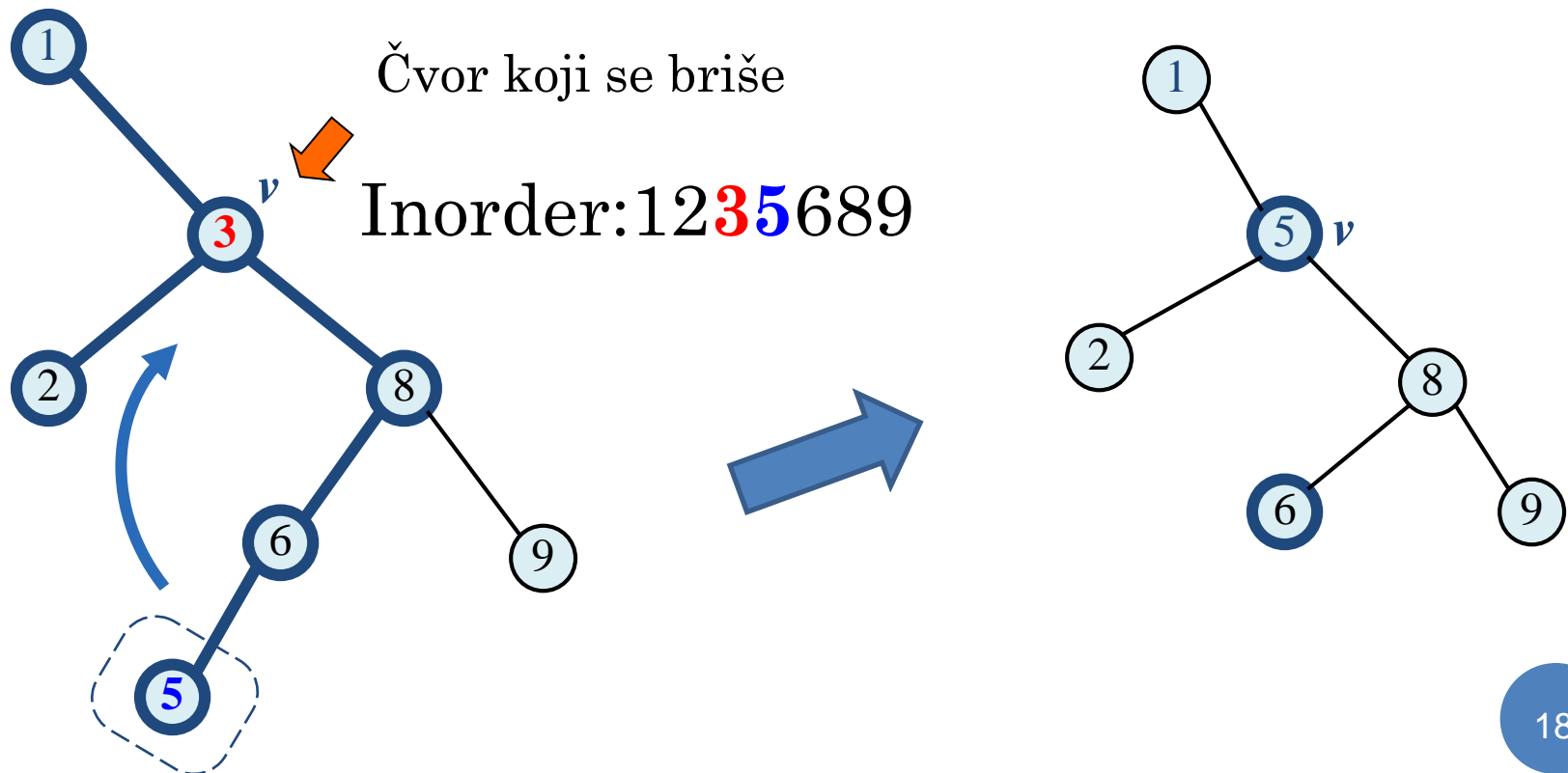
ILUSTRACIJA BRISANJA (2)

- Brisanje čvora stepena 1 (čvor 4)



ILUSTRACIJA BRISANJA (3)

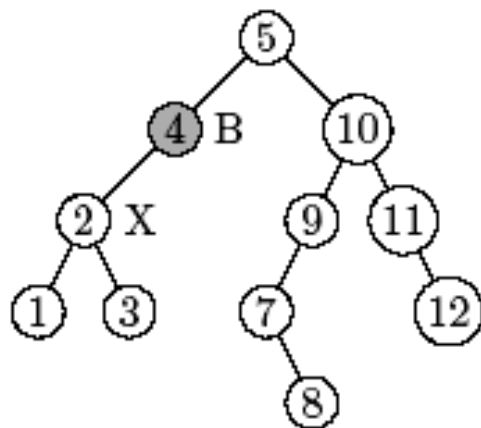
- Brisanje čvora stepena 2 (čvor 3)



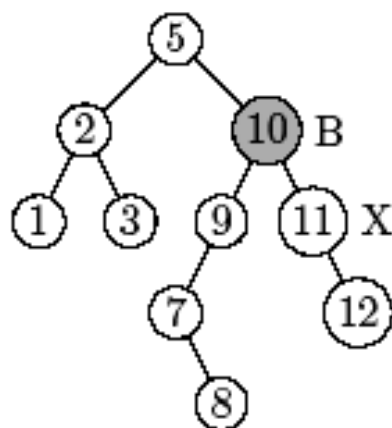
BRISANJE ČVORA

- PRIMERI

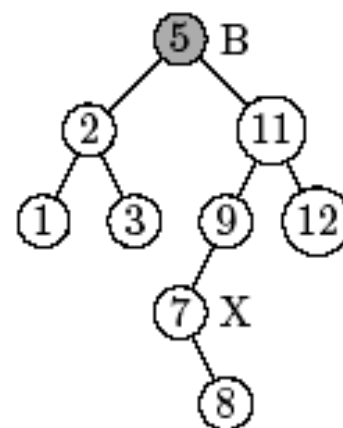
- Prikaz prethodno opisanih slučajeva je na slici.
- Sa B je označen čvor koji se briše (siva boja), a sa X čvor koji ga zamenjuje.
- Svako iduće stablo ujedno je i rezultat brisanja čvora iz prethodnog stabla.



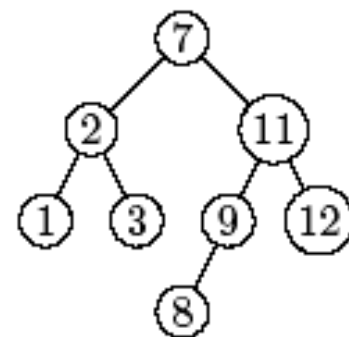
a



b

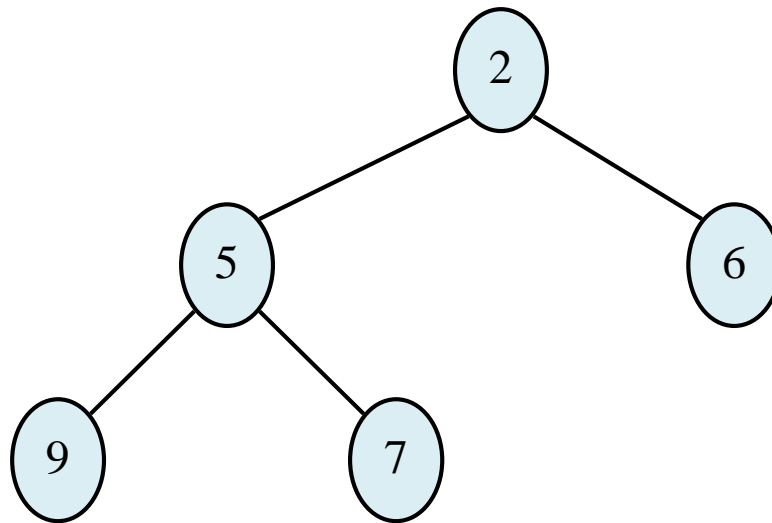


c



d

GOMILA HEAP



ŠTA JE HEAP?

- Heap je **gotovo kompletno binarno stablo** koje čuva vednosti u čvorovima i koje zadovoljava osobine tzv Heap-order.

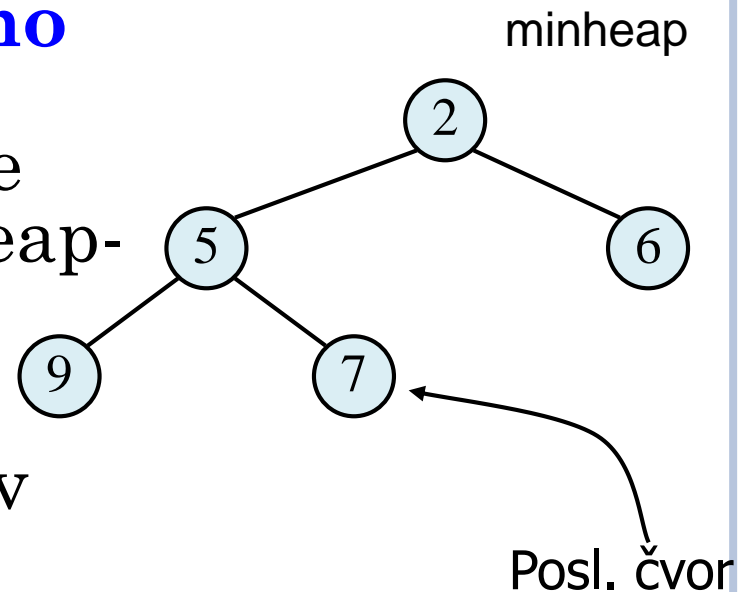
- **Heap-order**: za svaki čvor v izuzev korena, važi:

Minheap

$$key(v) \geq key(parent(v))$$

Maxheap

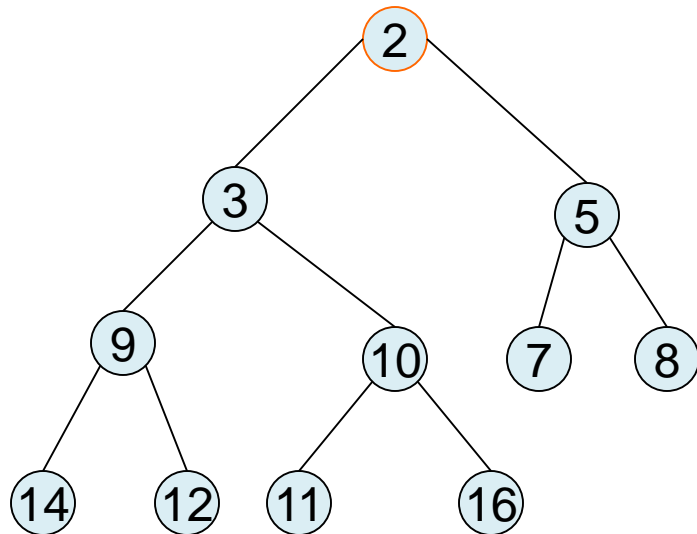
$$key(v) \leq key(parent(v))$$



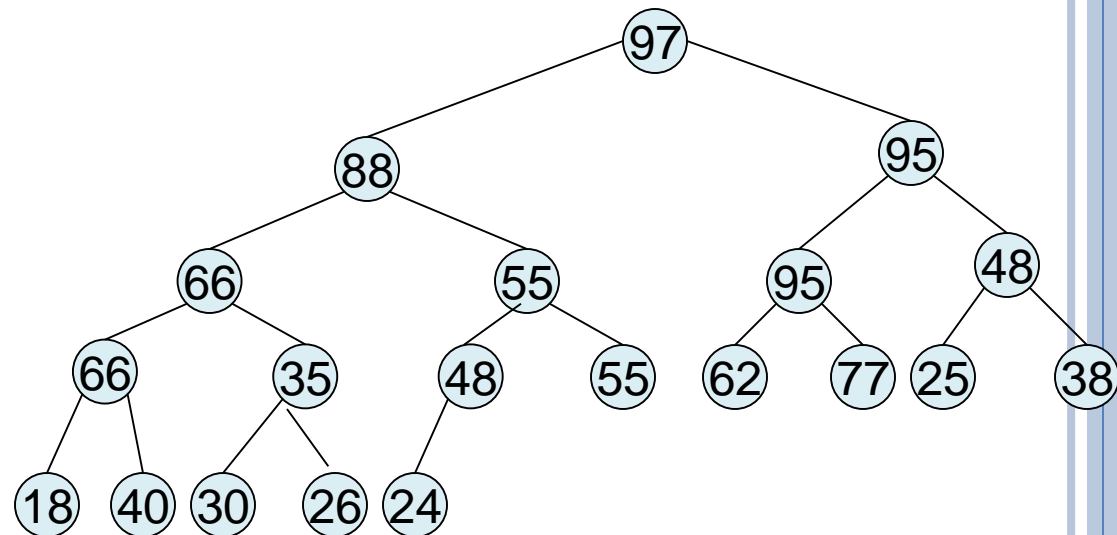
MIN I MAX HEAP

- Svaki koren nekog podstabla mora da ima **maksimalnu vrednost (MAXHEAP)** ili **minimalnu vrednost (MINHEAP)** u odnosu na sve ostale čvorove

Min Heap

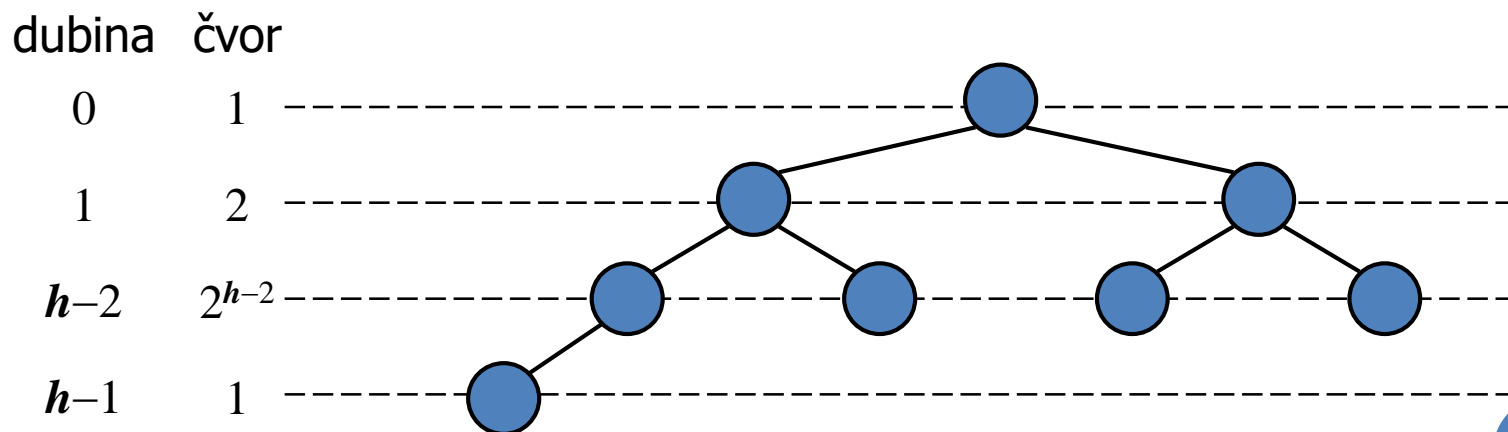


Max Heap



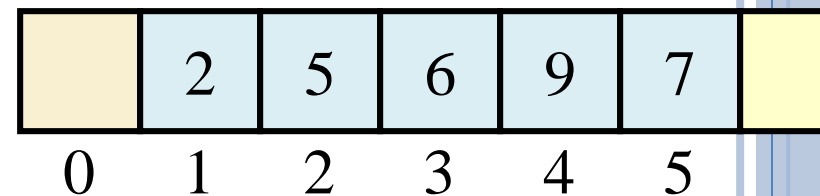
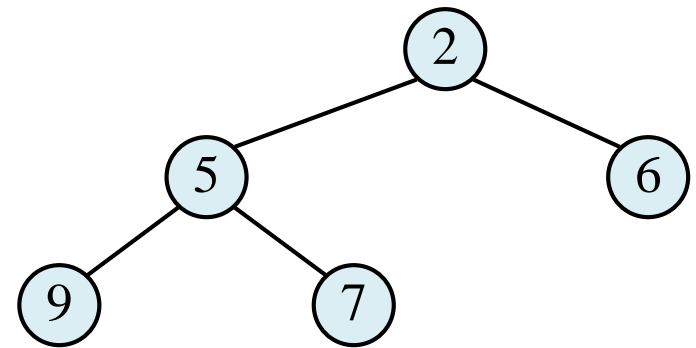
VISINA HEAP-A

- Heap koji čuva n vrednosti ima visinu $O(\log n)$
 - Neka je h visina heap-a koji čuva n vrednosti
 - Pošto ima 2^i čvorova/vrednosti na nivou $i = 0, \dots, h - 2$ i na nivou $h - 1$ je poslednji čvor



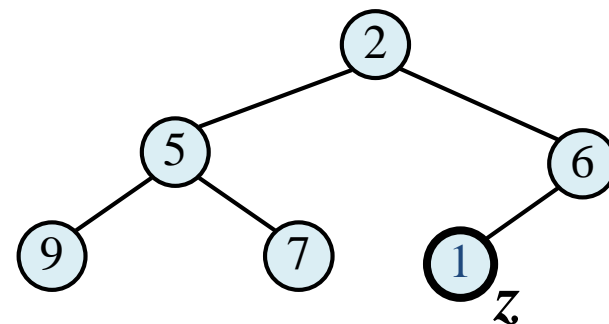
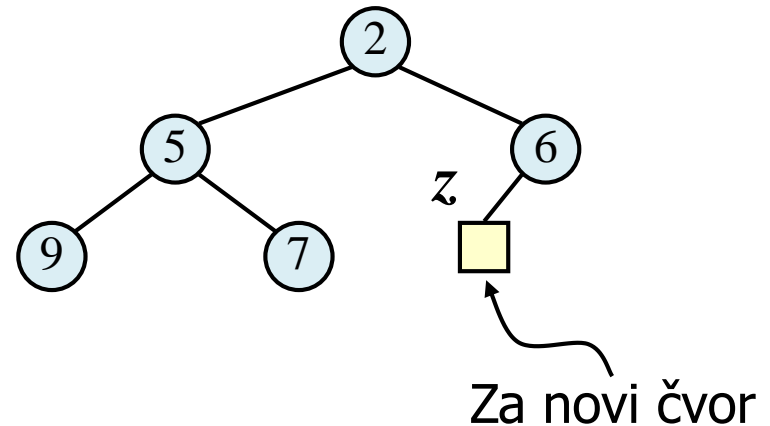
HEAP IMPLEMENTACIJA PREKO POLJA

- Za heap sa n vrednosti koristi se **polje** veličine $n + 1$
- Za čvor koji se nalazi na poziciji i
 - Levi potomak je na $2i$
 - Desni potomak je na $2i + 1$
- Čelija sa indeksom 0 se ne koristi
- Operacija *insertItem* odgovara dodavanju u polje na poziciji $n + 1$



DODAVANJE U HEAP

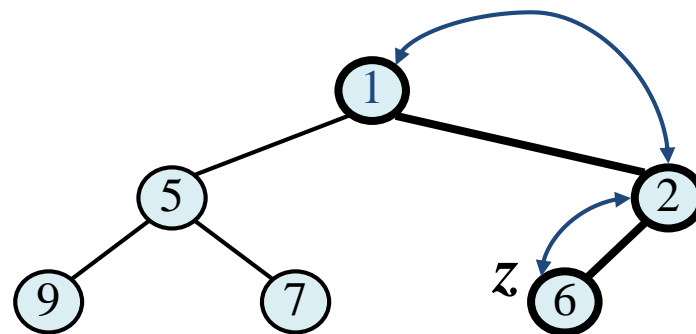
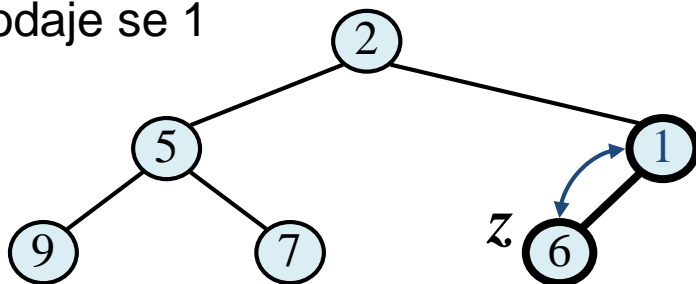
- Operacija **insertItem** – dodavanje ključa k u heap
- Algoritam u tri koraka:
 - Pronaći poslednji novi čvor z
 - Smesti k u z
 - Preuredi heap da poštuje uređenost - **upheap**



OPERACIJA UPHEAP

- Algoritam *upheap* ponovo uspostavlja heap-order osobinu premeštanjem k do prave pozicije
- upheap* se završava kada k dostigne koren ili čvor čiji roditelj ima manju ili jednaku vrednost u odnosu na k (minheap)
- Pošto je visina heap-a $O(\log n)$, *upheap* zahteva $O(\log n)$

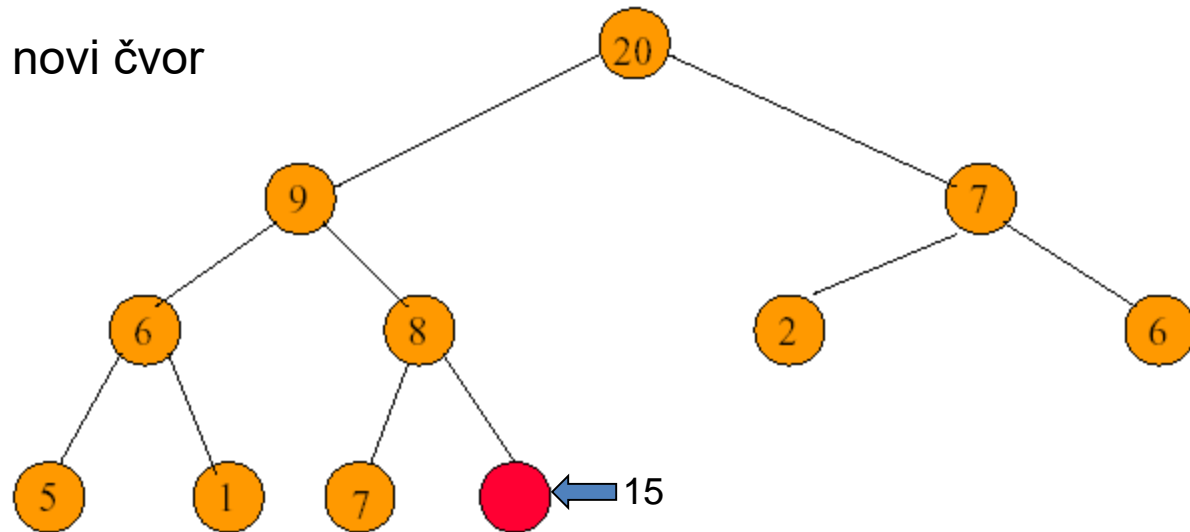
Dodaje se 1



PRIMER: DODAVANJE U HEAP

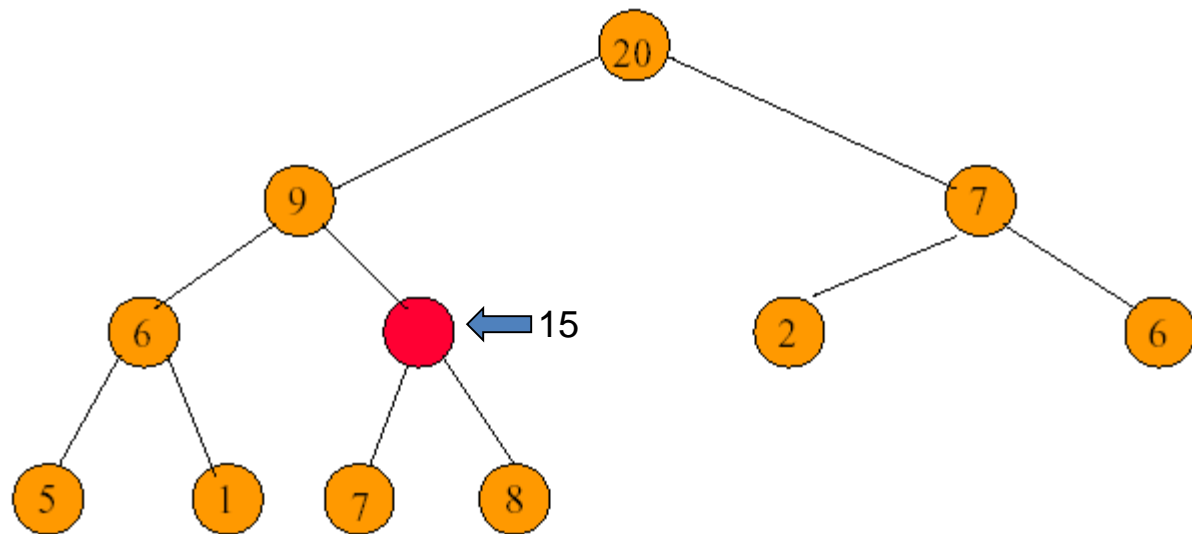
- Dodati element 15 u maxheap

Početno stablo + novi čvor



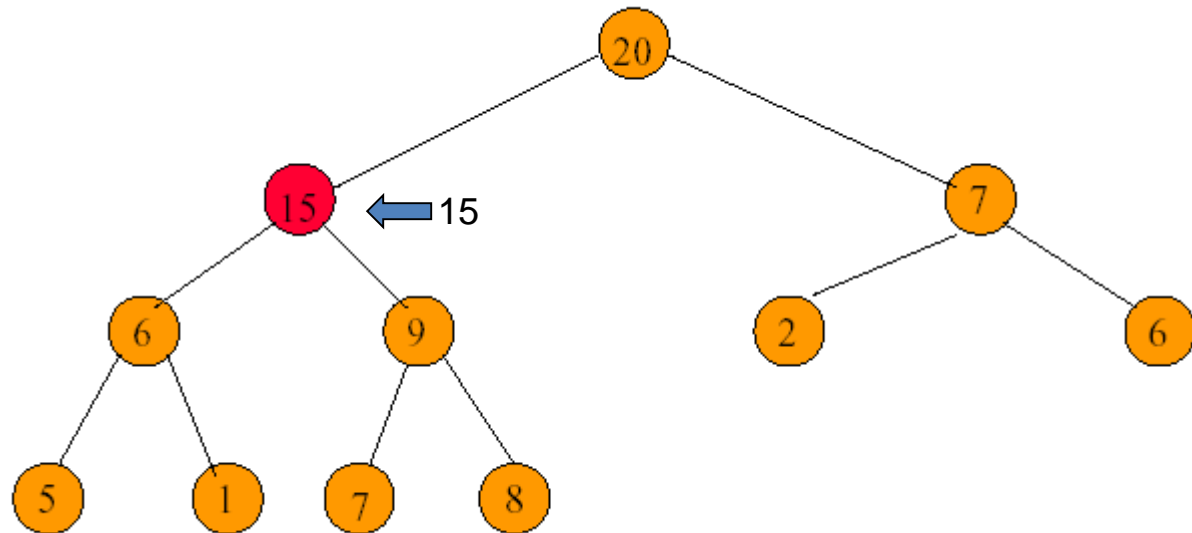
PRIMER: DODAVANJE U HEAP (2)

- Traženje pozicije za novu vrednost uz stablo (*upheap*)



PRIMER: DODAVANJE U HEAP (3)

- Upis 15 u novi čvor na pravom mestu



OPERACIJA DODAVANJA - PSEUDOKOD

Algoritam SBT.4 Dodavanje elementa min-gomili

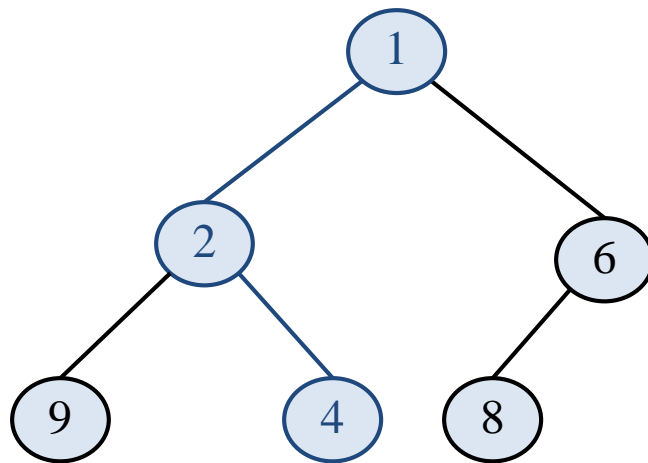
InsertHeap(h,n,e)

//h – gomila (heap), n – broj elemenata gomile, e – novi element

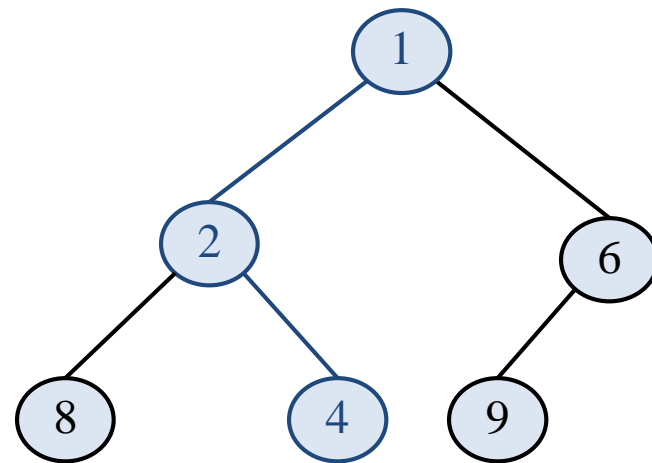
```
1.  n ← n+1, ptr ← n // lokacija novog elementa
2.  repeat while (ptr>1) {
3.    //nalaženje lokacije elementa e
4.    rod ← ⌊ptr/2⌋      // lokacija roditelja čvora
5.    if (e≤h(rod)) then
6.      h(ptr) ← e
7.      return
8.    h(ptr) ← h(rod)    // pomeranje čvora naniže
9.    ptr ← rod          // ažuriranje ptr
10. end repeat
11. h(1) ← e            // smeštanje e u koren
12. return
```

PRIMER KREIRANJE MIN HEAP-A

Niz: 6, 9, 2, 1, 4, 8



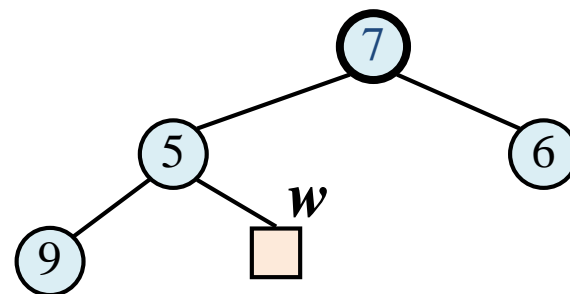
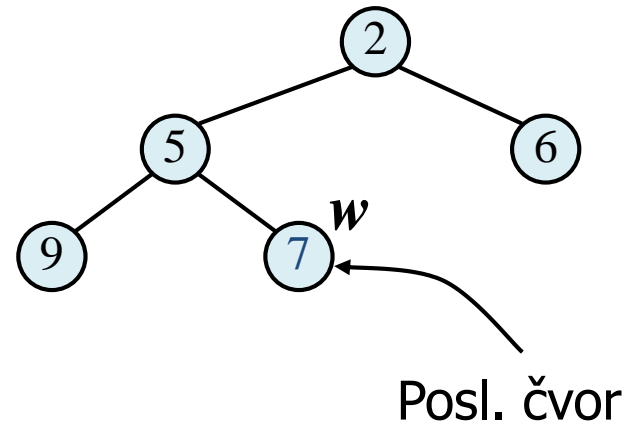
Niz: 6, 8, 4, 1, 2, 9



BRISANJE KORENA HEAP-A

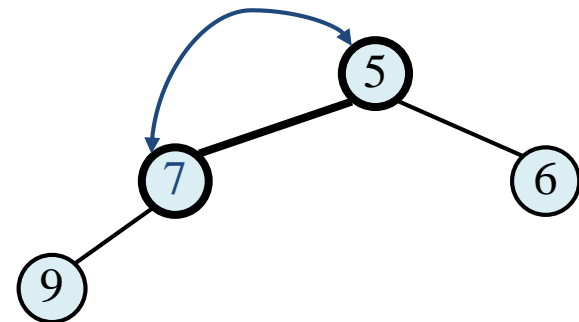
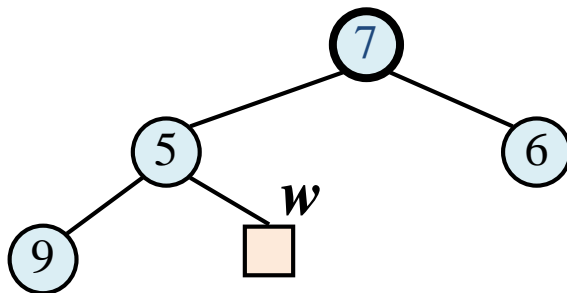
- Operacija **removeMin**
– odnosi se na brisanje korena minheap-a

- Algoritam:
 - Zameni koren vrednošću koja se nalazi u poslednjem čvoru w
 - Povrati uređenost heap-a odnosno heap-order osobinu - **downheap**



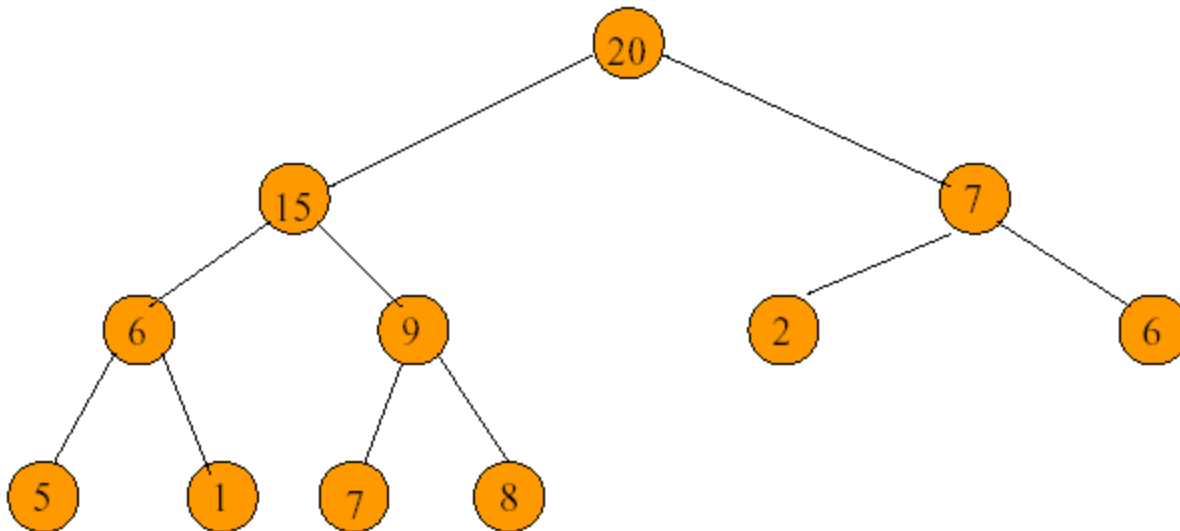
OPERACIJA DOWNHEAP

- Algoritam **downheap** ponovo uspostavlja heap-order zamenom ključa k sa čvorovima nadole od korena
- **Downheap** se završava kada ključ k dostigne list ili kada dođe do čvora čija deca imaju ključeve veće ili jednake k
- Visina za heap je reda $O(\log n)$, za **downheap** treba $O(\log n)$



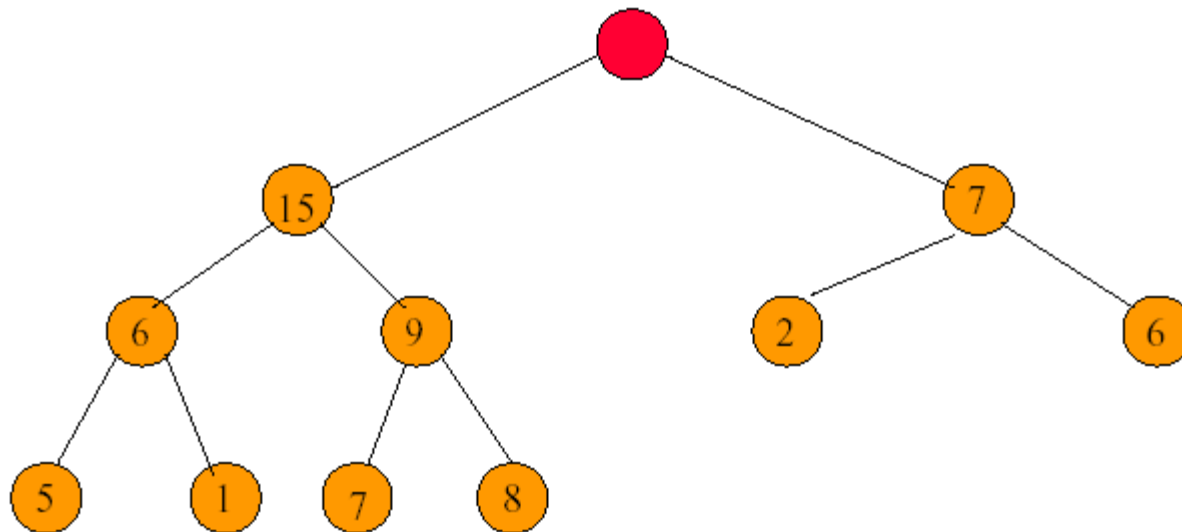
PRIMER BRISANJA KORENA GOMILE (1)

- Koren sadrži max element



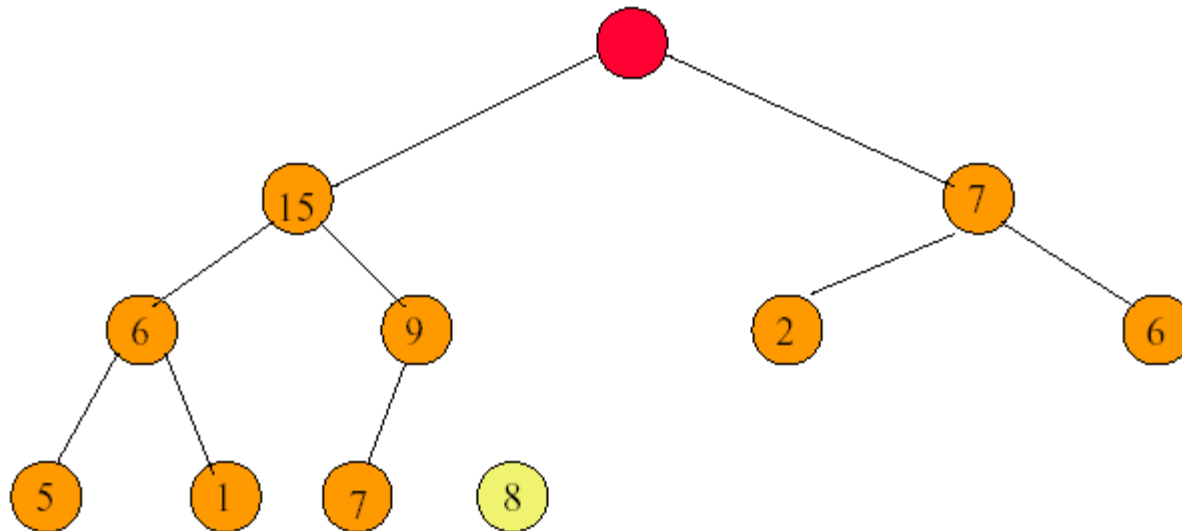
PRIMER BRISANJA KORENA GOMILE (2)

- Brisanje vrednosti iz korena



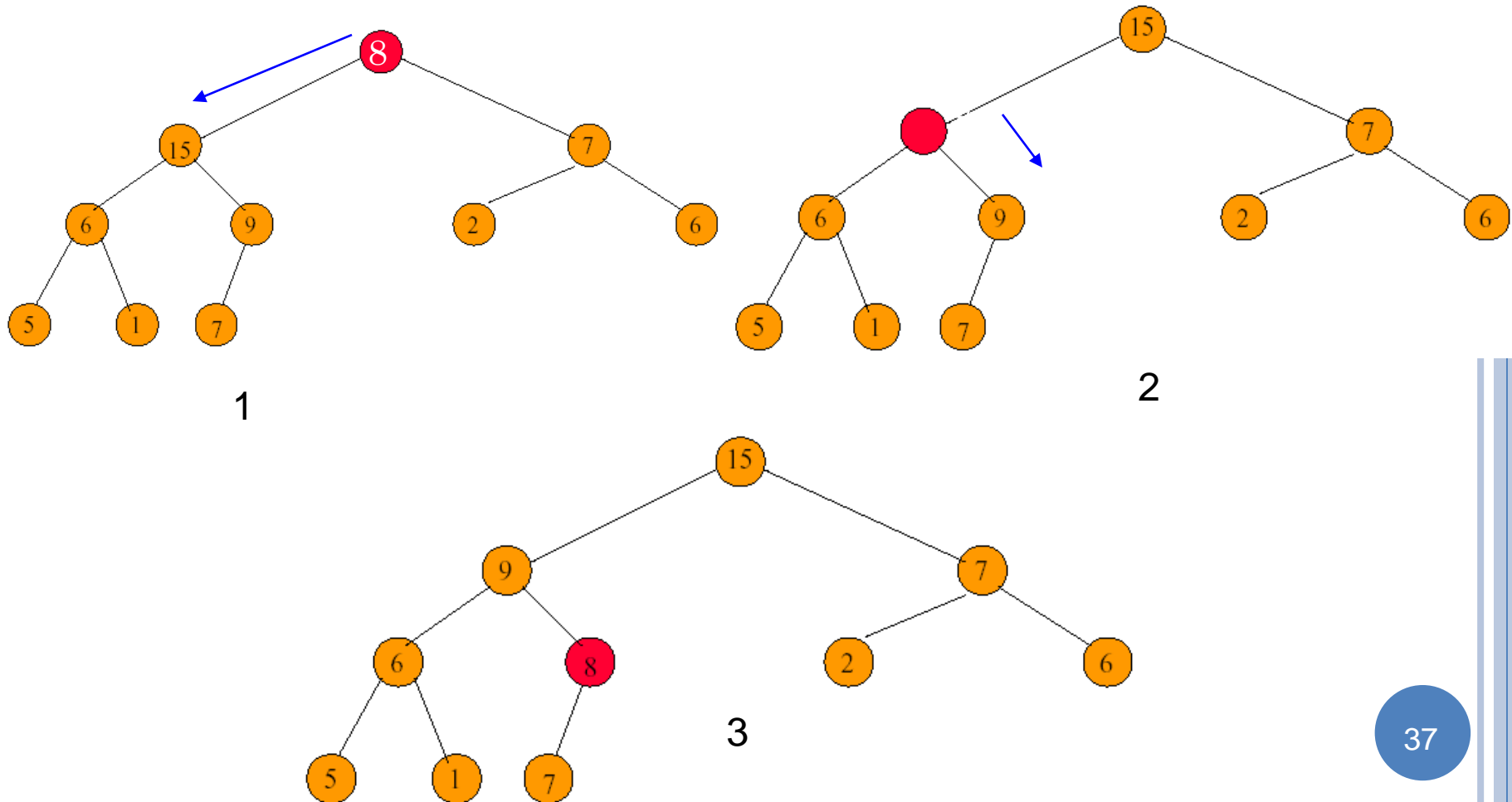
PRIMER BRISANJA KORENA GOMILE (3)

- Uzimanje poslednjeg čvora 8 iz heap-a



PRIMER BRISANJA KORENA GOMILE (4)

- Traženje mesta za vrednost 8



BRISANJE KORENA GOMILE

Algoritam SBT.5 Brisanje korena gomile

DeleteHeap(h,n,e)

/ h – heap, n – broj el. gomile, e – obrisani el., posl - vrednost iz poslednjeg elem. Gomile, ptr - lokacija poslednjeg elementa, levo i desno – deca posl. elementa gomile */*

```
1.  e ← h[1] // uklanjanje korena gomile
2.  posl ← h[n]; n ← n-1 // brisanje posl. elementa
3.  if (n=0) then return //brisanje jedinog elem. gomile
4.  ptr ← 1; levo ← 2; desno ← 3 // inicijalizacija
5.  repeat while (desno<=n)
6.  //nalazenje prave lokacije elementa
7.    if (posl>h[levo] and posl>= h[desno])
8.      then {h[ptr] ← posl; return}
9.    if (h[desno]<=h[levo])
10.     then {h[ptr] ← h[levo]; ptr ← levo}
11.     else {h[ptr] ← h[desno]; ptr ← desno }
12.    levo ← 2*ptr; desno ← levo + 1
13.  end repeat
14.  if (levo = n) and if(posl < h[levo]) then ptr ← levo
15.  h[ptr] ← posl
16.  return
```

- Petlja 5-13 se ponavlja sve dok čvor **posl** ima desno dete
- Korak 14 obrađuje granični slučaj kada **posl** nema desno dete, ali ima levo dete
- U koraku 14 dve if naredbe su stavljene jer **h[levo]** ne mora biti definisan kada je **levo>n**

HEAP-SORT

- Neka heap sadrži n elemenata
 - Zahtevani prostor je $O(n)$
 - Operacije `insertItem` i `removeMin` zahtevaju $O(\log n)$
 - Operacije `size`, `isEmpty`, `minKey`, i `minElement` zahtevaju $O(1)$
- Korišćenjem heap-a možemo sortirati sekvencu od n elemenata za $O(n \log n)$
- Takvo sortiranje se zove **Heap-sort**
- Heap-sort je mnogo brži od algoritama kao što su insertion-sort i selection-sort

HEAPSORT – PSEUDOKOD

Algoritam SBT.6. Heap sort

heapSort(a,n)

// sortira polje *a* od *n* elemenata

// formiranje min ili max gomile od elemenata polja
a

1. **repeat for** $j=1, n-1$

2. **call** insertHeap($a, j, a[j+1]$)

// sortiranje brisanjem korena gomile dok se gomila
ne isprazni

1. **repeat while** $n > 1$

2. **call** deleteHeap(a, n, e)

3. $a[n+1] = e$

4. **exit**

PITANJA, IDEJE, KOMENTARI

