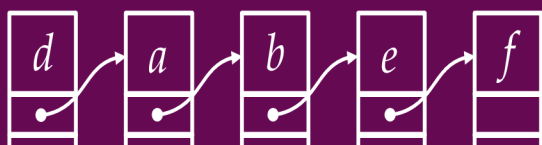
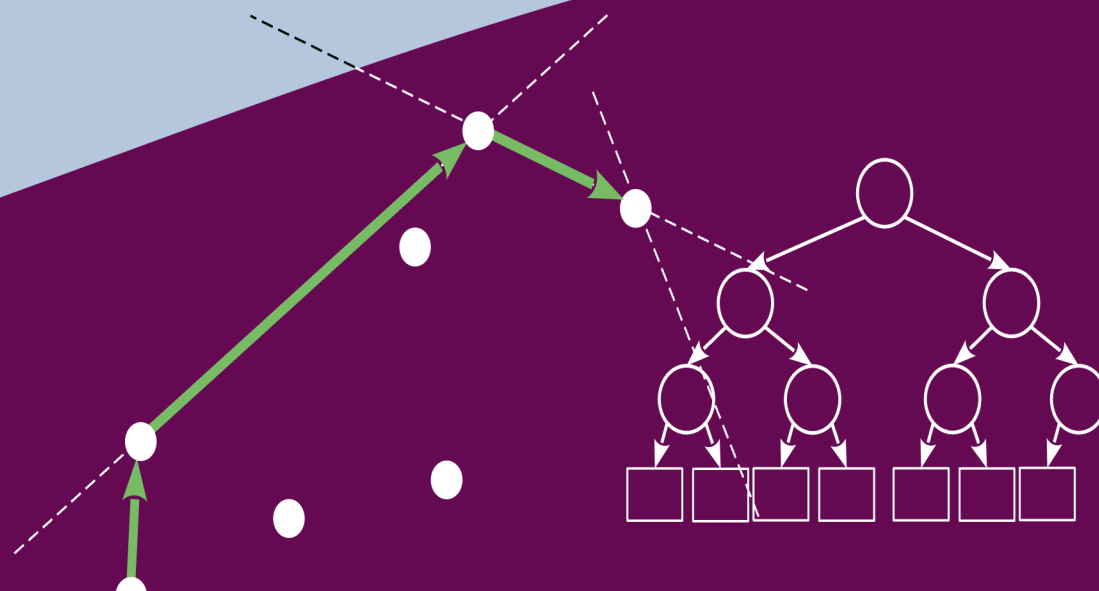
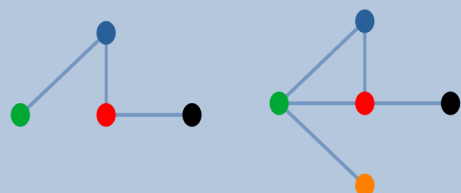


# Napredni algoritmi i strukture podataka

Tjedan 1: Napredne strukture podataka



# Creative Commons



- slobodno smijete:

- dijeliti — umnožavati, distribuirati i javnosti priopćavati djelo
- prerađivati djelo



- pod sljedećim uvjetima:

- imenovanje: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- nekomercijalno: ovo djelo ne smijete koristiti u komercijalne svrhe.
- dijeli pod istim uvjetima: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



*U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

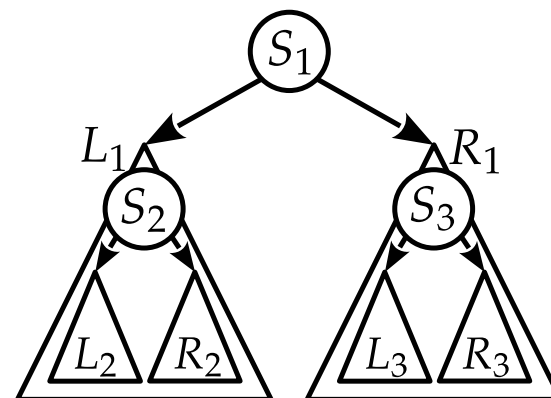
*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

*Tekst licence preuzet je s <http://creativecommons.org/>*

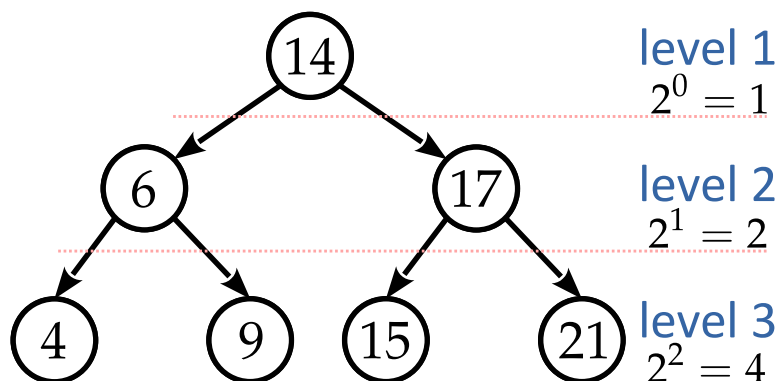
# Binarna stabla (1)

- Stabla su usmjereni aciklički grafovi
- Binarna stabla su specifična – svaki čvor može imati najviše dva djeteta
  - Binarno stablo se može opisati uređenom trojkom
$$B = (L, S, R)$$
    - gdje je L lijevo podstablo, S je korijen binarnog stabla B, a R je desno podstablo
  - Rekurzivna definicija



# Binarna stabla (2)

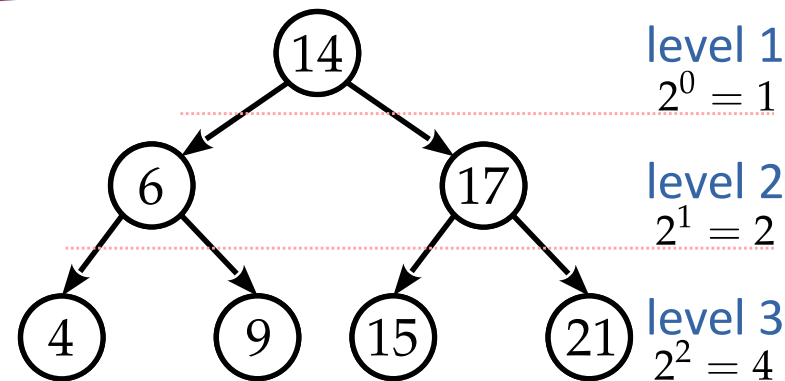
- Za opis odnosa između čvorova binarnog stabla koristimo rodoslovne izraze
  - Roditelj, dijete, blizanci
  - Mogu se koristiti izrazi kao : pradjed (roditelj roditelja), ujak (blizanac roditelja)
- **Savršeno binarno stablo** (*perfect*)
  - Binarno stablo kojem su sve razine do kraja popunjene čvorovima



# Binarna stabla (3)

- Svojstva savršenog binarnog stabla

- Broj čvorova  $n = 2^h - 1$
- Broj listova  $l = 2^{h-1}$
- Broj unutarnjih čvorova  $i = 2^{h-1} - 1$
- Visina  $h = \log_2(n + 1) = \log_2 2^h$



- **Kompletno binarno stablo** (*complete*) – Binarno stablo koje ima sve razine, osim najdonje, potpuno popunjene čvorovima. U najdonjoj razini listovi se popunjavaju s lijeve strane.

- Broj čvorova  $n \leq 2^h - 1$
- Broj listova  $l \leq 2^{h-1}$
- Broj unutarnjih čvorova  $i \leq 2^{h-1} - 1$

# Binarna stabla (4)

- Visina kompletnog binarnog stabla – direktno vezano uz kompleksnost pretraživanja

$$h = \lceil \log_2(n + 1) \rceil$$

- Vrijedi i

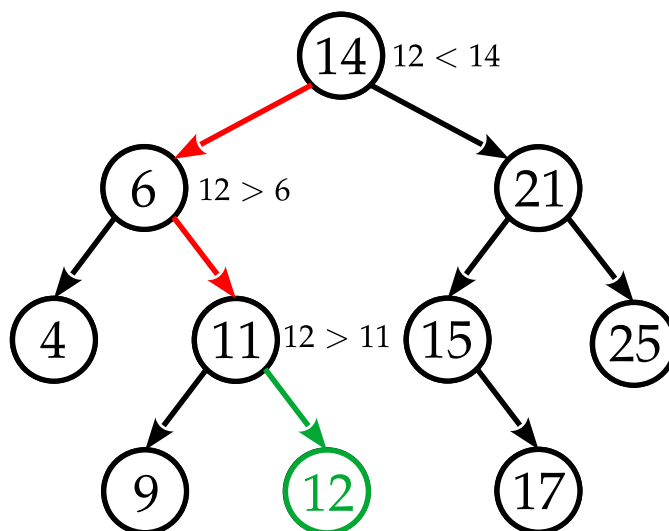
$$n = i + l \leq 2^h - 1$$

- **Puno binarno stablo** (*full*) – Binarno stablo čiji unutarnji čvorovi imaju točno dva djeteta.
- Organizacija **sortiranog** binarnog stabla  $B = (L, S, R)$ 
  - Bez duplikata  $v(S(L)) < v(S) < v(S(R))$
  - Sa duplikatima

$$\begin{aligned} v(S(L)) &\leq v(S) < v(S(R)) \\ v(S(L)) &< v(S) \leq v(S(R)) \end{aligned}$$

# Operacije nad binarnim stablom (1)

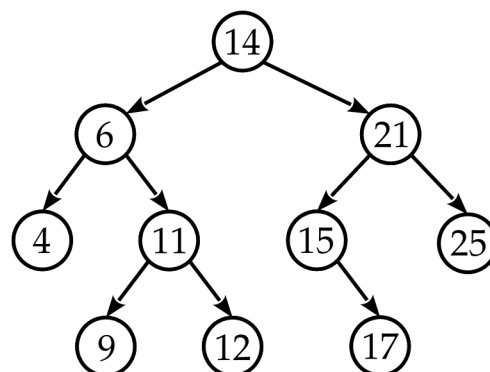
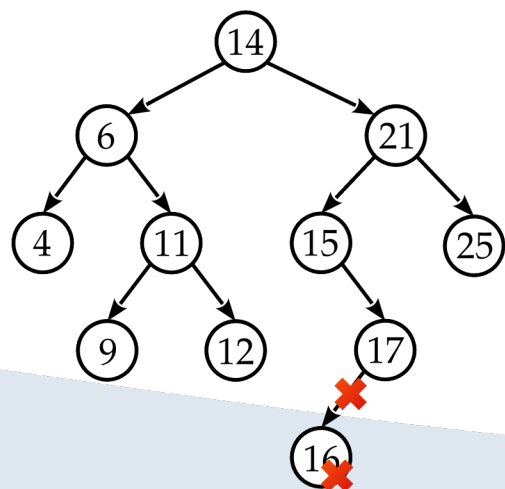
- **Ponovimo** dodavanje vrijednosti u binarno stablo
  - Dodavanju prethodi pretraživanje
  - Nailaskom na slobodno mjesto, dodajemo novi čvor prema organizaciji binarnog stabla:
    - Lijevo ako je manji
    - Desno ako je veći



# Brisanje čvorova (1)

- Prvo pronademo čvor koji brišemo
- Tri slučaja:
  1. Čvor koji se briše je list
  2. Čvor koji se briše ima jedno dijete
  3. Čvor koji se briše ima oba djeteta

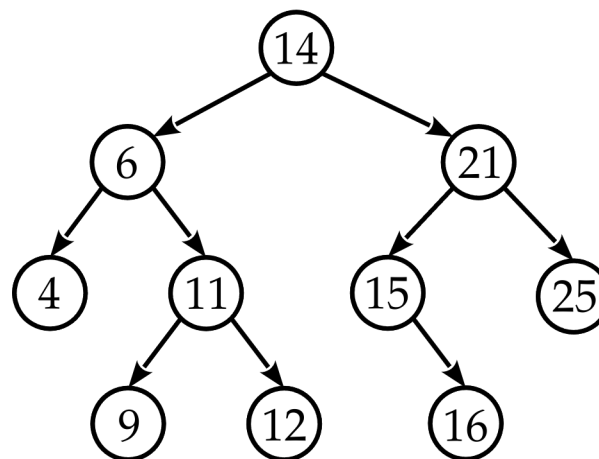
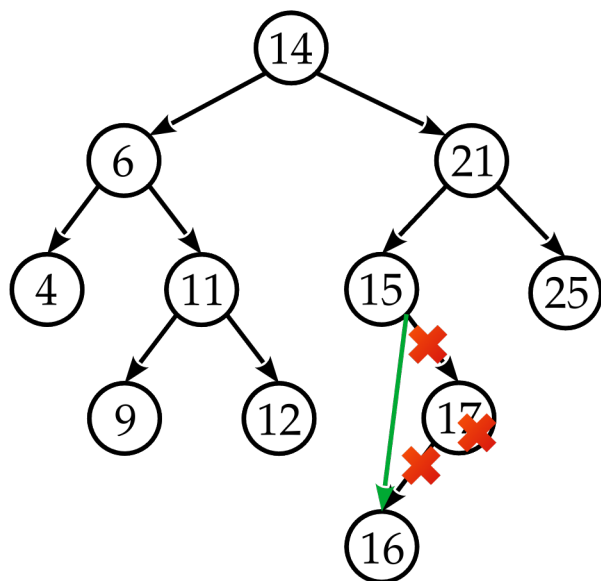
1. Čvor koji se briše je list – samo obrišemo čvor





# Brisanje čvorova (2)

2. Čvor ima jedno dijete – To dijete postaje novo dijete roditelja čvora koji se briše



# Brisanje čvorova (3)

## 3. Čvor ima dva djeteta – opcije

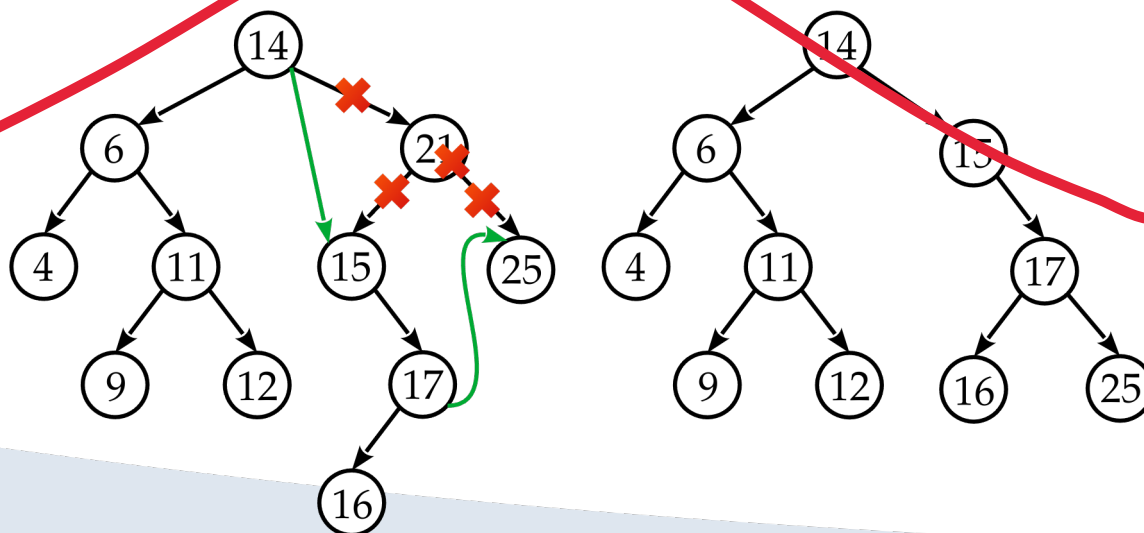
- Brisanje sjedinjenjem (delete by merging) – znatno mijenja strukturu
- Brisanje kopiranjem (delete by copy) – struktura se minimalno mijenja

- Brisanje sjedinjenjem

- Naći čvor koji se briše i njegovog roditelja (ako se ne briše korijenski čvor)
- Utvrdimo na kojoj strani je čvor koji se briše u odnosu na svojeg roditelja
  - Ako je korijenski čvor u pitanju, onda je svejedno koje podstablo uzimamo za podstablo sjedinjenja (*merging subtree*)
  - Inače imamo dvije opcije:
    - Ako je čvor koji se briše u **lijevom** podstablu roditelja, tada je postablo sjedinjenja njegovo desno podstablo
    - Ako je čvor koji se briše u **desnom** podstablu roditelja, tada je postablo sjedinjenja njegovo lijevo podstablo

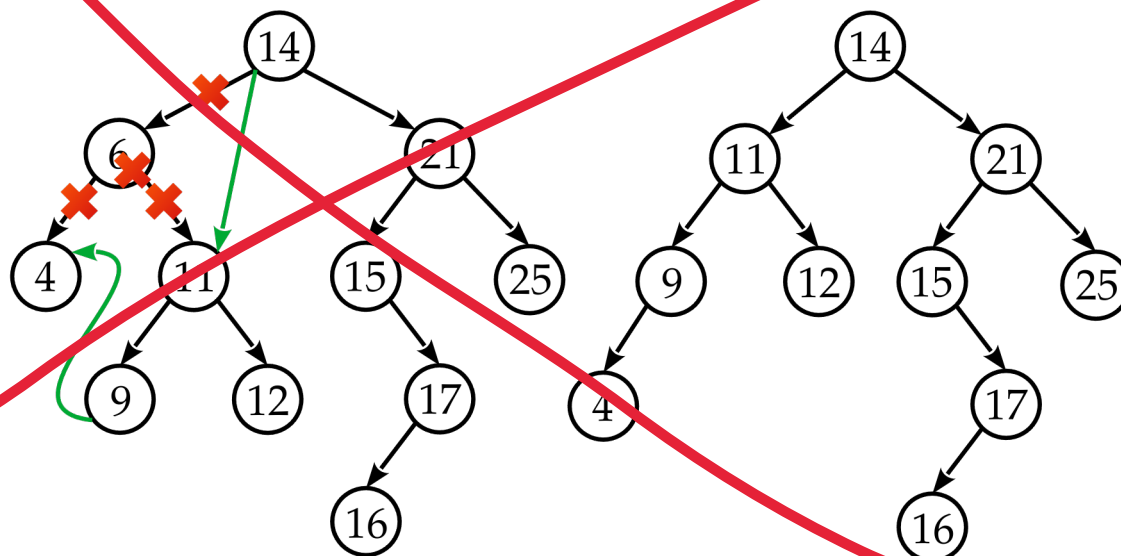
# Brisanje čvorova (4)

- Brisanje čvora A sjedinjenjem
  - Čvor blizanac korijenskom čvoru podstabla sjedinjenja spajamo na
    - Čvor **sljedbenik** ako je podstablo sjedinjenja bilo desno podstablo čvora A
    - Čvor **prethodnik** ako je podstablo sjedinjenja bilo lijevo podstablo čvora A
  - Korijen podstabla sjedinjenja spajamo s roditeljem čvora koji se obrisao
    - Osim u slučaju kada se briše korijenski čvor



# Brisanje čvorova (5)

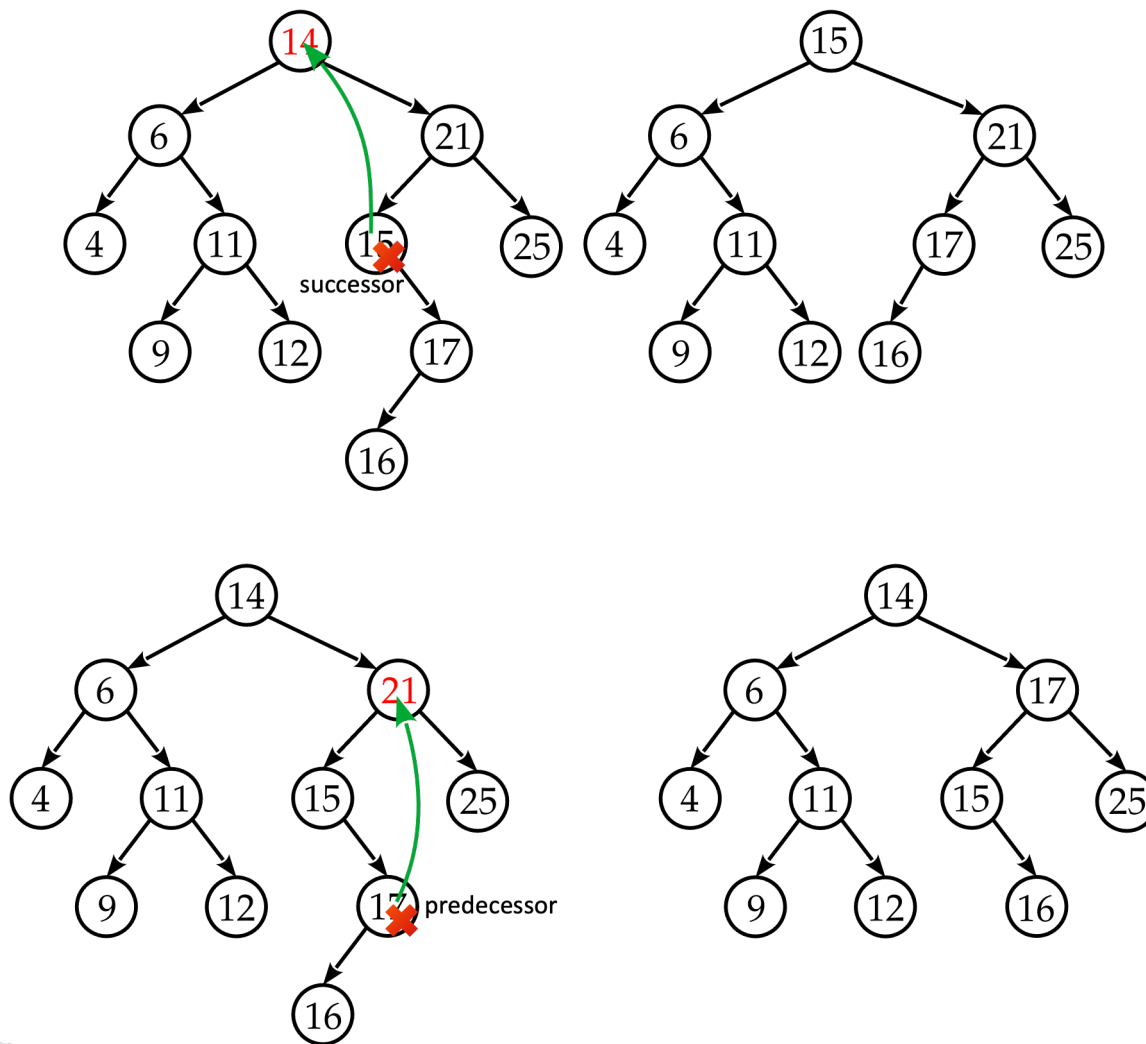
- Brisanje sjedinjenjem



# Brisanje čvorova (6)

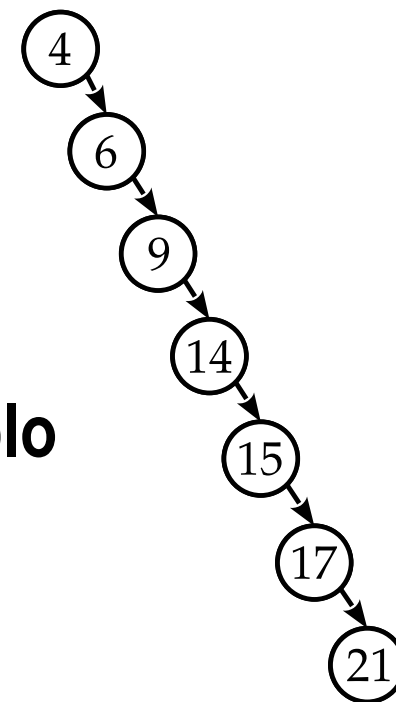
- **Brisanje kopiranjem**
    - Svodi se na brisanje čvora bez djece ili s jednim djetetom
    - Pronađe se zamjenski čvor koji se obriše - minimalno se mijenja struktura binarnog stabla
1. Pronađemo čvor A koji brišemo
  2. Pronađemo čvor X koji sadrži direktnog prethodnika ili sljedbenika – zamjenski čvor
    - Prethodnik je najdesniji čvor u lijevom podstablu A
    - Sljedbenik je najlijeviji čvor u desnom podstablu A
  3. Prekopiramo vrijednost zamjenskog čvora X u čvor A koji se briše
  4. Zamjenski čvor X se ukloni

# Brisanje čvorova (7)



# Uravnoteženo binarno stablo (1)

- Zašto nam je uravnoteženo binarno stablo zanimljivo?
- Želimo postići složenost pretraživanja od  $O(\log_2 n)$
- Druga krajnost – **degenerirano (koso) binarno stablo** (*degenerate*) – primjer desno
  - Složenost je  $O(n)$

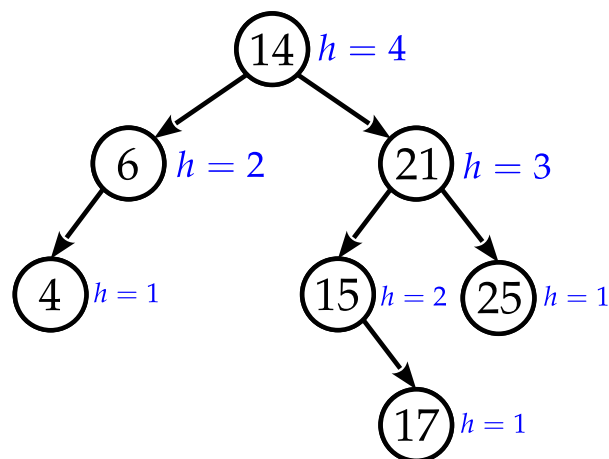
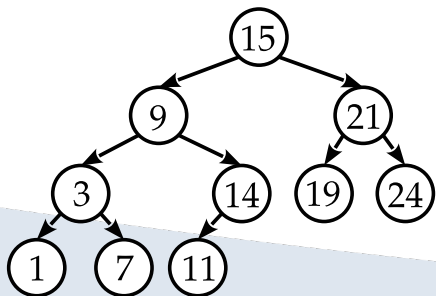


# Uravnoteženo binarno stablo (2)

- Za binarno stablo  $B = (L, S, R)$  definicija uravnoteženog stabla je
$$\forall S \quad |h(L) - h(R)| \leq 1$$
  - Razlika visina lijevog i desnog podstabla **svakog čvora** smije biti najviše 1

- **Savršeno uravnoteženo binarno stablo** (*perfectly balanced*) – uravnoteženo i kompletno

- Sve razine osim zadnje su posve popunjene čvorovima





# Stvaranje binarnog stabla (1)

(iz sortiranog polja vrijednosti)

- Na raspolaganju imamo sortirano polje (*array*) vrijednosti

$$V_S = \langle 1, 3, 7, 9, 11, 14, 15, 21, 24 \rangle$$

1. Pronađemo pozicijski srednju vrijednost  $v$  u polju
2. Stvorimo korijenski čvor vrijednosti  $v$  trenutnog binarnog stabla
3. Za podpolje lijevo od  $v$  se rekurzivno stvara lijevo podstablo
4. Za podpolje desno od  $v$  se rekurzivno stvara desno podstablo
5. Ponavljamo dok možemo stvoriti lijevo ili desno podstablo

```
function CREATEBALANCEDTREE( $V_S$ )
```

```
 $n \leftarrow |S|$ 
```

```
if  $n > 0$  then
```

```
   $i \leftarrow (n \div 2) + (n \% 2)$ 
```

```
   $root \leftarrow$  create node having value  $V_S[i - 1]$ 
```

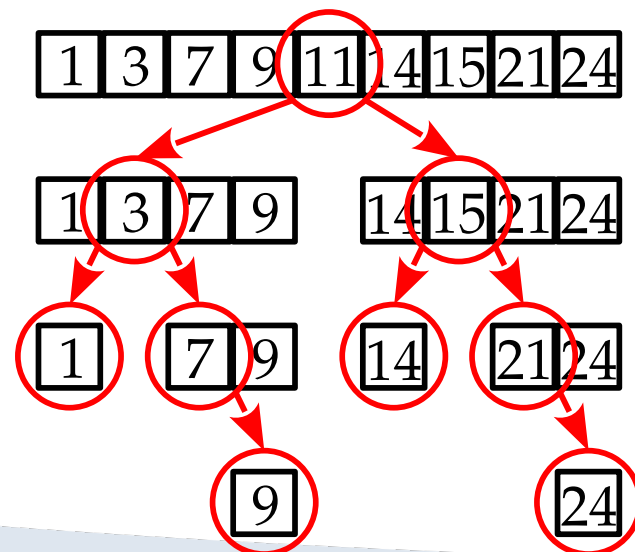
```
   $leftChild(root) \leftarrow$  CREATEBALANCEDTREE( $V_S[0, i - 1]$ )
```

```
   $rightChild(root) \leftarrow$  CREATEBALANCEDTREE( $V_S[i, n]$ )
```

```
  return  $root$ 
```

```
else
```

```
  return nil
```



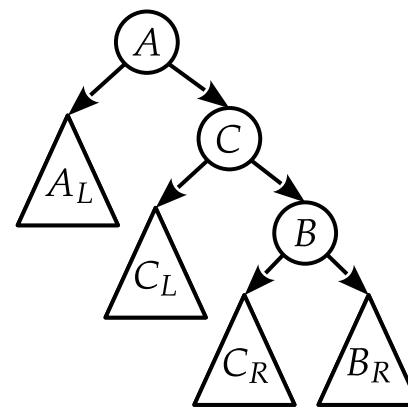
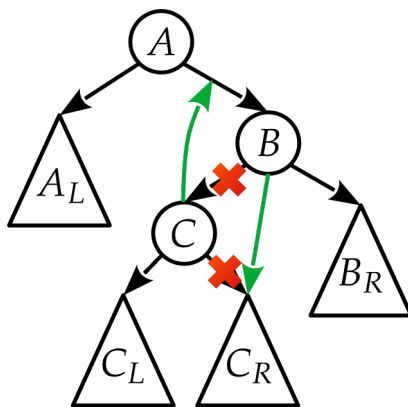
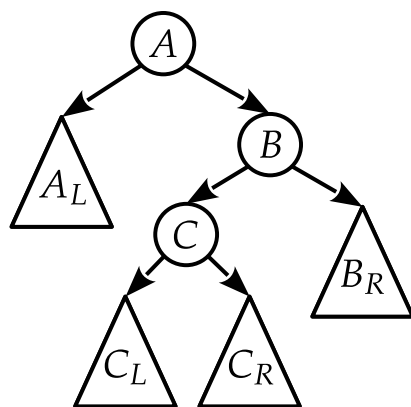
# Stvaranje binarnog stabla (2)

(iz sortiranog polja vrijednosti)

- Ovaj algoritam ima kompleksnost  $O(n \log_2 n + n)$ 
  - Sortiranje polja + prolaz po svim elementima polja
- Algoritam se može upotrijebiti samo u specijalnim situacijama
  - Kada imamo polje vrijednosti i od njega želimo stvoriti uravnoteženo binarno stablo
  - Koristi se relativno često, konkretni primjeri će biti kasnije u predavanjima
- Ovako stvoreno binarno stablo je uravnoteženo

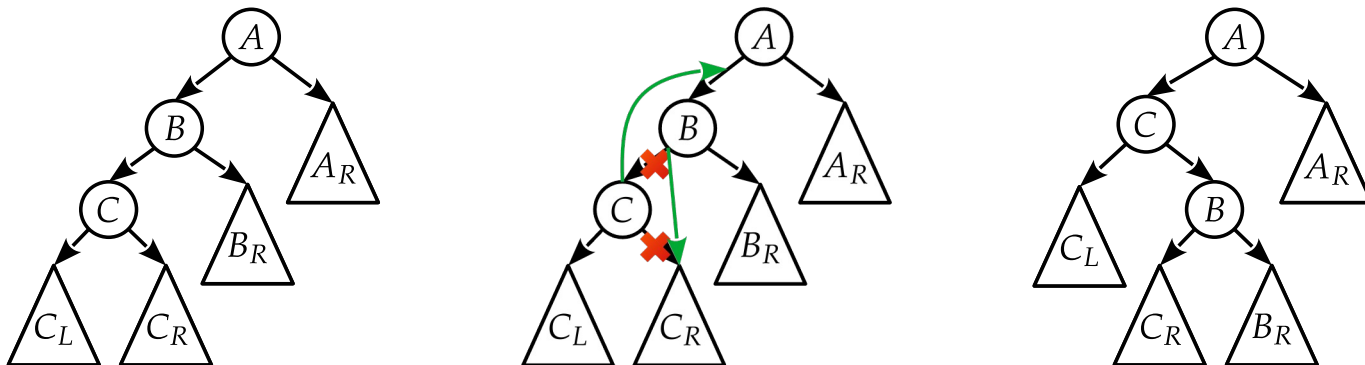
# Rotacije u stablu (1) - desna

- Za uravnotežavanje stabala trebamo dvije operacije (rotacije) nad binarnim stablima (analogija koloture)
- Desna rotacija C oko B
  - Kako rotirati stablo da C bude između A i B, a da se sačuva poredak  $v(A_L) < v(A) < v(C_L) < v(C) < v(C_R) < v(B) < v(B_R)$

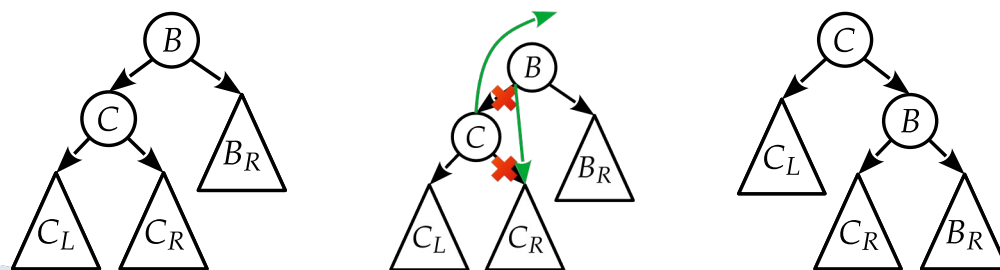


# Rotacije u stablu (2) - desna

- Još jedna desna rotacija C oko B

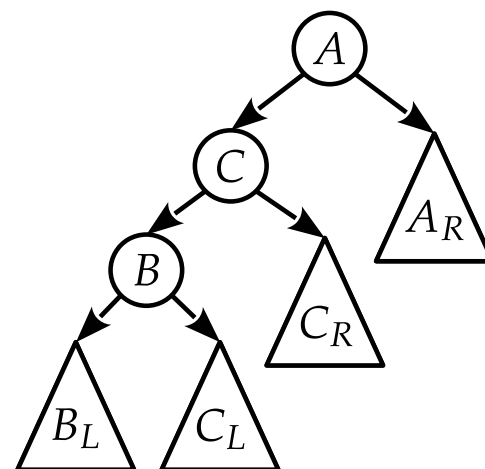
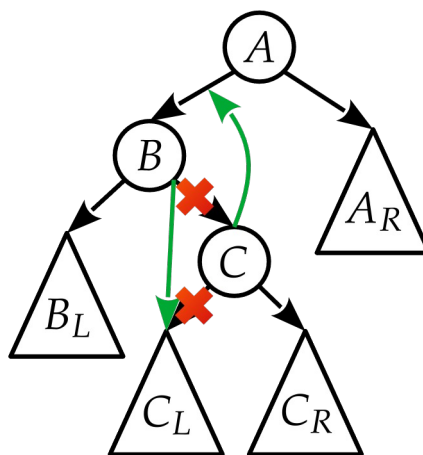
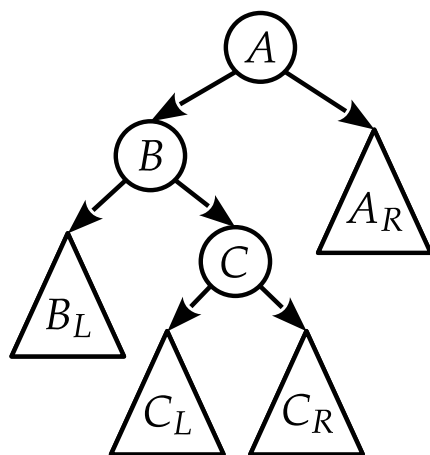


- Desno dijete od C postaje lijevo dijete od B
- B postaje desno dijete od C
- C postaje dijete od bivšeg roditelja čvora B (ako postoji)



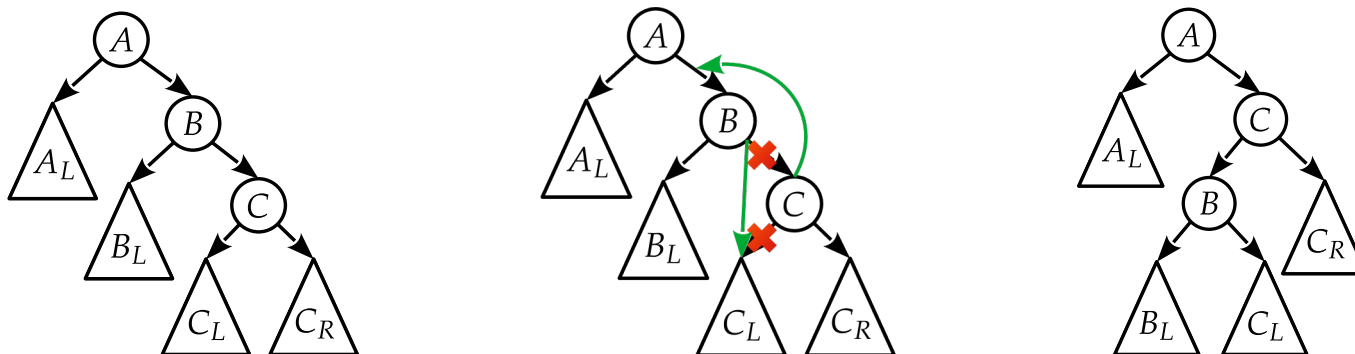
# Rotacije u stablu (3) - lijeva

- Lijeva rotacija C oko B
  - Kako rotirati stablo da C bude između A i B, a da se sačuva poredak  
 $v(B_L) < v(B) < v(C_L) < v(C) < v(C_R) < v(A) < v(A_R)$

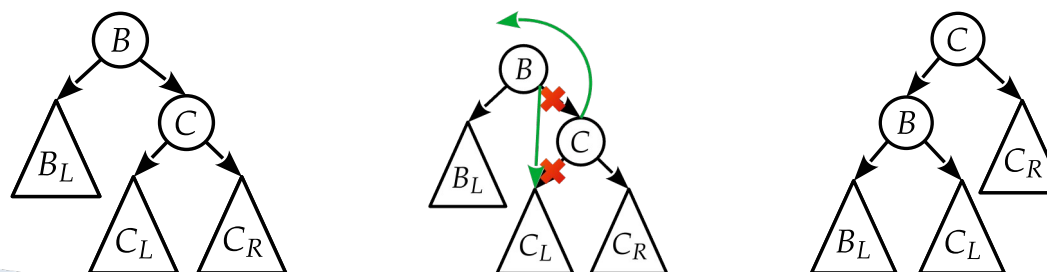


# Rotacije u stablu (4) - lijeva

- Još jedna lijeva rotacija C oko B



- Lijevo dijete od C postaje desno dijete od B
- B postaje lijevo dijete od C
- C postaje dijete od bivšeg roditelja čvora B (ako postoji)



# Day-Stout-Warren algoritam (DSW)

- Dvije faze algoritma

1. Izrada kralježnice (koso stablo)
2. Rekurzivno lomljenje kralježnice nazad u kompletno stablo

# DSW – izrada kralježnice

```
procedure RIGHTBACKBONE(root)
```

```
  B  $\leftarrow$  root
```

```
  A  $\leftarrow$  nil
```

```
  while B  $\neq$  nil do
```

```
    C  $\leftarrow$  leftChild(B)
```

```
    if C  $\neq$  nil then
```

```
      RIGHTROTATE(A, B)
```

```
      if A = nil then
```

```
        root  $\leftarrow$  C
```

```
      B  $\leftarrow$  C
```

```
    else
```

```
       $\triangleright$  Descending right to the first node that has the left child
```

```
      A  $\leftarrow$  B
```

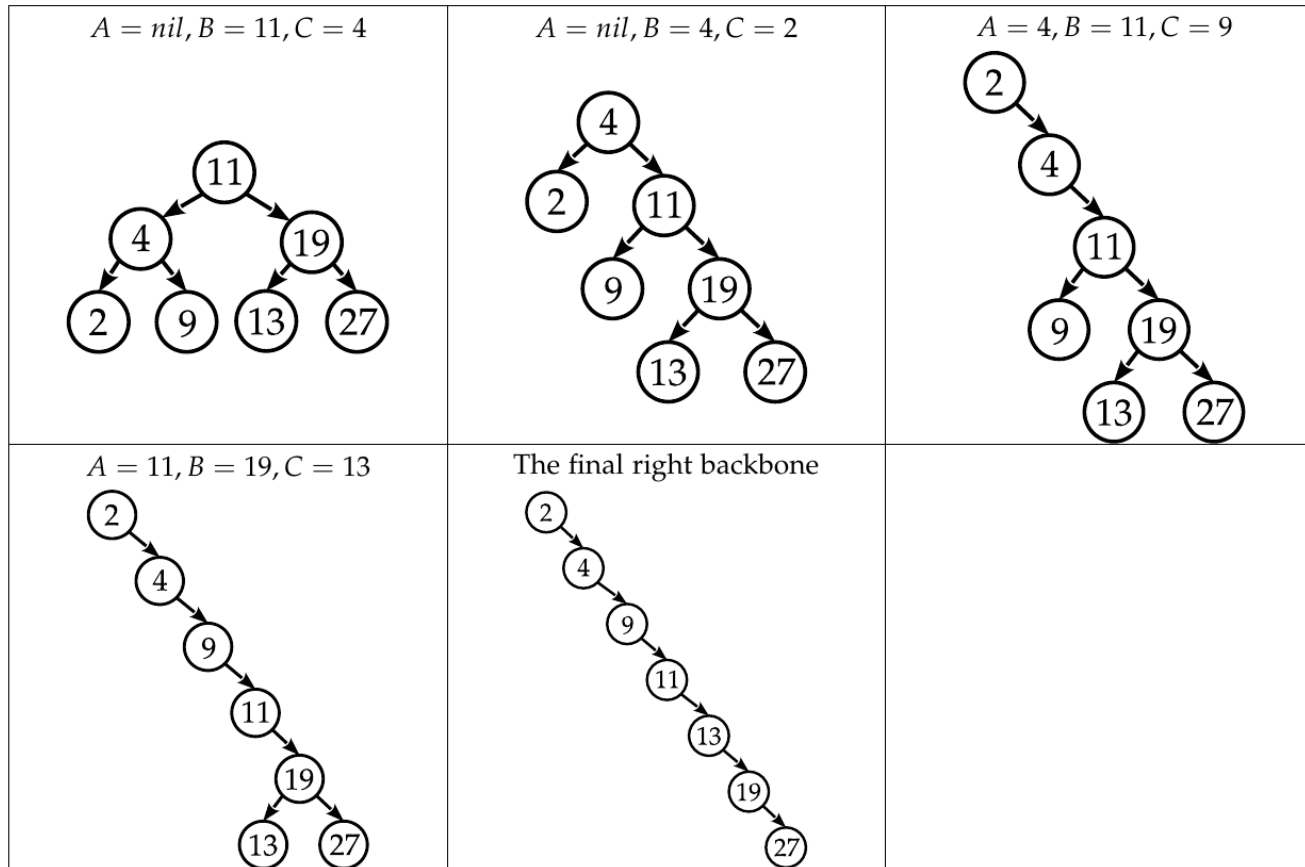
```
      B  $\leftarrow$  rightChild(B)
```

- Prvi korak je stvaranje kralježnice od binarnog stabla – npr. desna kralježnica
- Lijevu djecu čvorova rotiramo desno
- Ponavljamo desne rotacije dok nema lijeve djece



# DSW – izrada kralježnice

- Primjer



# DSW - lomljenje

- Strateški pozicionirane lijeve rotacije za savršeno uravnoteženo binarno stablo

```
procedure DSW(tree, n)  
   $h \leftarrow \lceil \log_2(n + 1) \rceil$   
   $i \leftarrow 2^{h-1} - 1$   
  perform  $n - i$  rotations of every second node from the root  
  while  $i > 1$  do  
     $i \leftarrow \lfloor i/2 \rfloor$   
    perform  $i$  rotations of every second node from the root
```

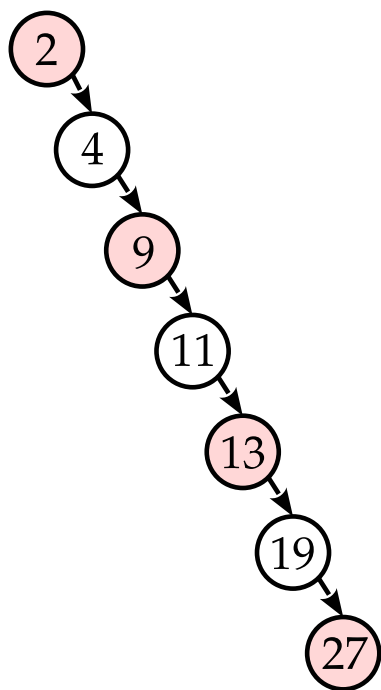
- $h$  – visina binarnog stabla za  $n$  čvorova
- $i$  – broj unutarnjih čvorova
- Za desnu kralježnicu radimo lijeve rotacije kako bismo čvorove vratili u strukturu binarnog stabla

# DSW – lomljenje, primjer

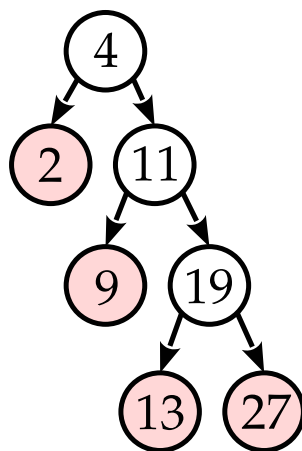
$$n = 7$$

$$h = 3$$

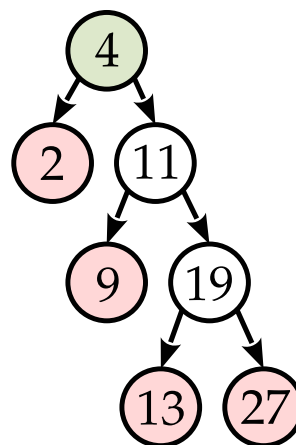
$$i = 3$$



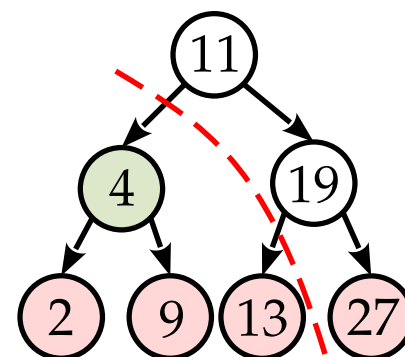
$n - i = 4$   
lijevih rotacija  
za listove



$\lfloor i/2 \rfloor = \lfloor 3/2 \rfloor = 1$   
lijevih rotacija za  
interne čvorove



dobije se  
savršeno binarno  
stablo

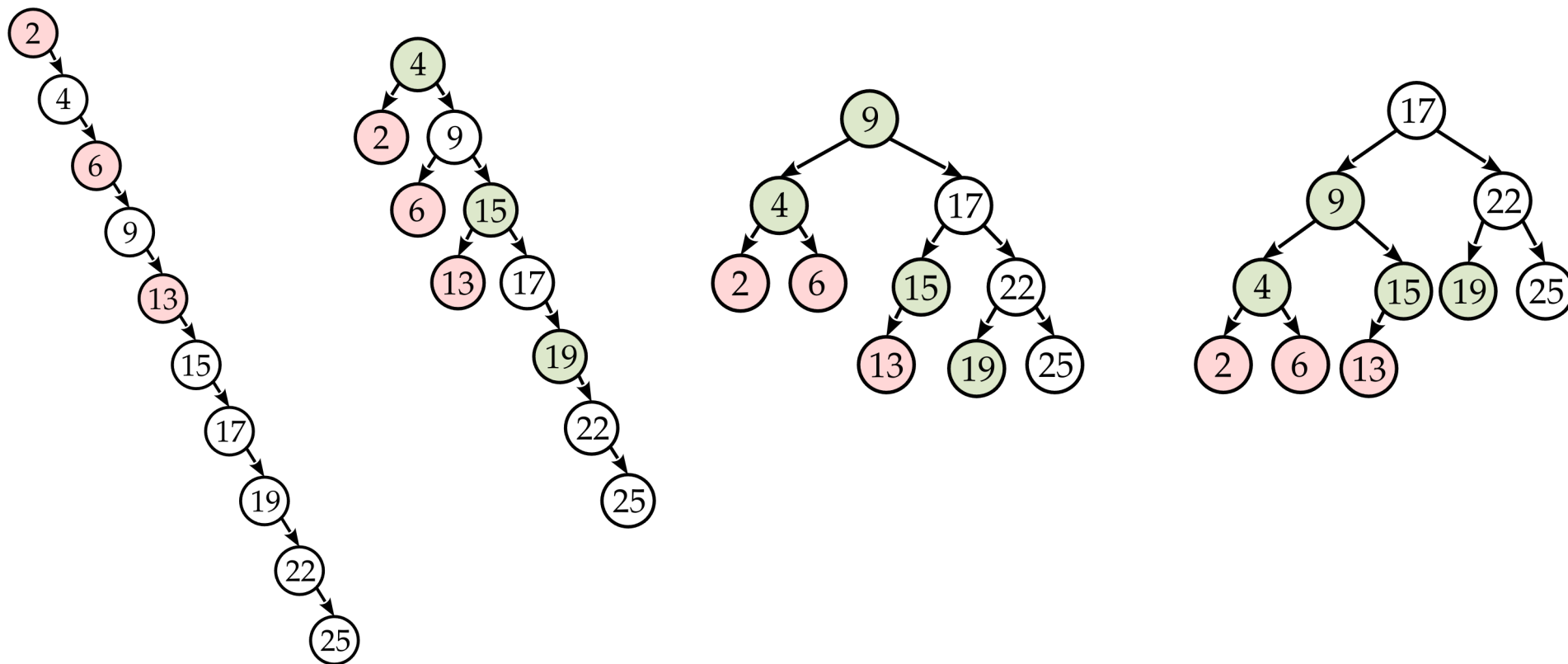


# DSW – lomljenje, primjer

- Primjer

- $n = 10$
- To je  $h = 4$  razine
- Ukupni broj internih čvorova je  $i = 7$
- Čvorova u najdonjoj razini je  $10 - 7 = 3$  – prvo radimo 3 lijeve rotacije svakog drugog čvora desne kralježnice, počevši od korijenskog čvora
- Zatim radimo  $\lfloor 7/2 \rfloor = 3$  lijeve rotacije svakog drugog čvora ostatka desne kralježnice, počevši od korijenskog čvora, što nam daje najdonju razinu internih čvorova
- Zatim radimo  $\lfloor 3/2 \rfloor = 1$  lijeve rotacije svakog drugog čvora ostatka desne kralježnice, počevši od korijenskog čvora, što nam daje najdonju razinu internih čvorova

# DSW – lomljenje, primjer



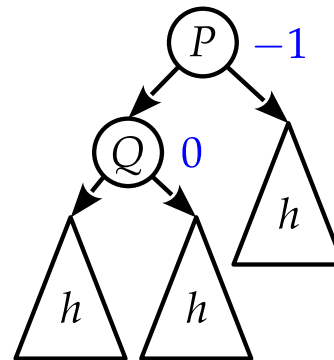
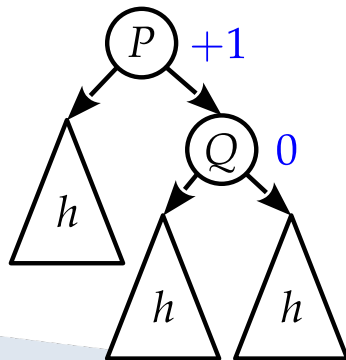
- Ovo je globalno uravnoteživanje binarnog stabla
  - Strukturu stabla prvo uništimo da bismo ga uravnotežili
  - Složenost DSW algoritma je  $O(n)$

# Adelson-Velski-Landis binarno stablo (AVL)

- Prethodni primjeri su **offline** uravnotežavanje
- **Online** uravnotežavanje – dodavanje novih vrijednosti
  - Provjeriti da li je binarno stablo uravnoteženo?
  - Ako nije, uravnotežiti to stablo s minimalnim zahvatom u njegovu strukturu
- To se zove lokalno uravnoteživanje

# AVL (2)

- Dodavanjem novog lista možemo se kretati po putanji do korijenskog čvora, te provjeravati uravnoteženost u svakom čvoru
- Za binarno stablo  $B = (L, S, R)$  definiramo faktor ravnoteže (*balance factor*) kao
$$BF(S) = h(R) - h(L)$$
- Za sve čvorove koji imaju  $-1 \leq BF(S) \leq 1$  smatramo da je njihovo podstablo uravnoteženo



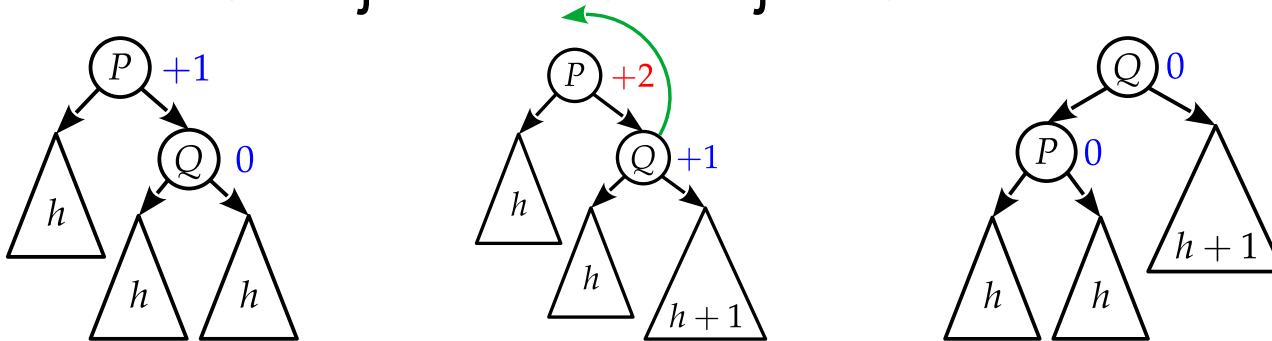
# AVL (3)

- Nakon dodavanja ažuriramo faktore ravnoteže na vertikalnoj putanji. Ako postoji čvor  $S$  sa  $BF(S) = -2$  ili  $BF(S) = 2$ , potrebno je lokalno uravnotežavanje
- Dva slučaja – čvor i njegovo dijete (ovisno o predznaku čvora):
  - **Izravnati** slučaj, identični predznaci BF:
    - Faktor ravnoteže čvora je +2 i desno dijete ima faktor ravnoteže 0 ili +1 – desni izravnati slučaj
    - Faktor ravnoteže čvora je -2 i lijevo dijete ima faktor ravnoteže 0 ili -1 – lijevi izravnati slučaj
  - **Izlomljeni** slučaj, strogo suprotni predznaci BF:
    - Faktor ravnoteže čvora je +2 i desno dijete ima faktor ravnoteže -1 – desni izlomljeni slučaj
    - Faktor ravnoteže čvora je -2 i lijevo dijete ima faktor ravnoteže +1 – lijevi izlomljeni slučaj

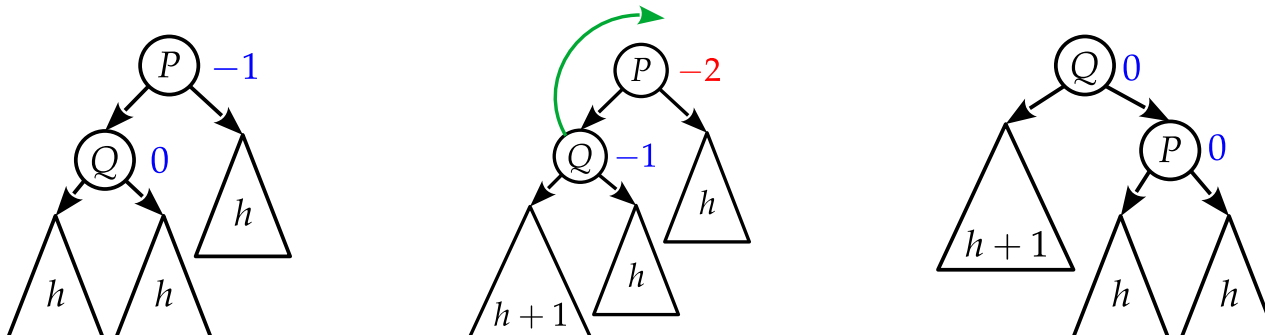


# AVL (4)

- Desni izravnati slučaj: +2 i desno dijete 0 ili +1



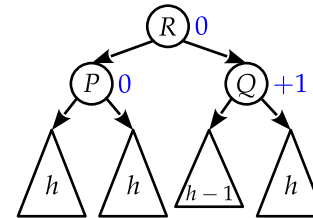
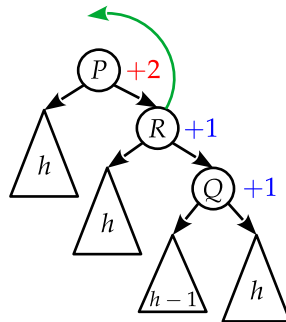
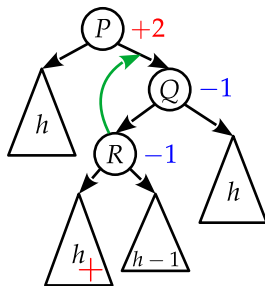
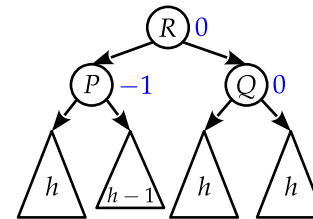
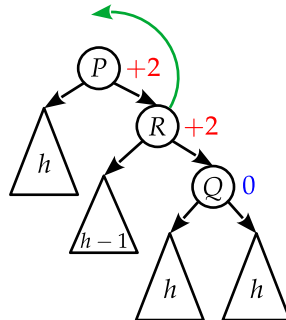
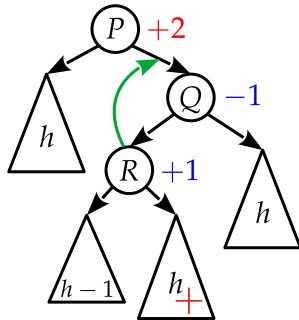
- Novu vrijednost dodajemo u desno podstablo čvora Q
- Radi se lijeva rotacija Q oko P
- Lijevi izravnati slučaj: -2 i lijevo dijete 0 ili -1



- Radi se desna rotacija Q oko P

# AVL (5)

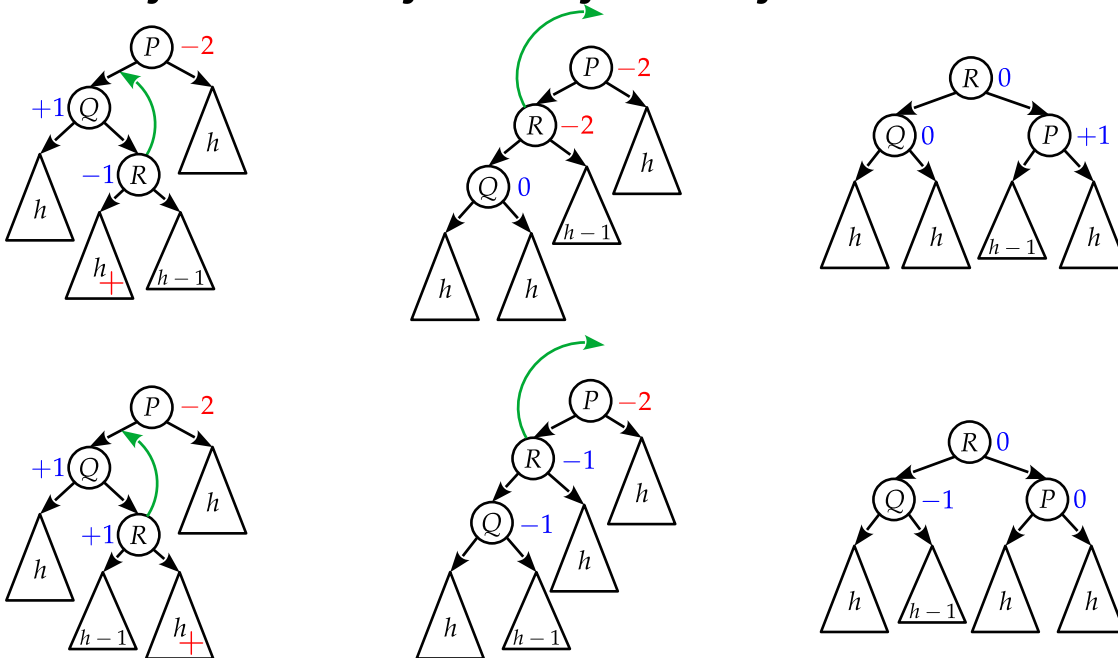
- Desni izlomljeni slučaj: +2 i desno dijete -1



- Prvo desna rotacija R oko Q
- Zatim lijeva rotacija R oko P

# AVL (6)

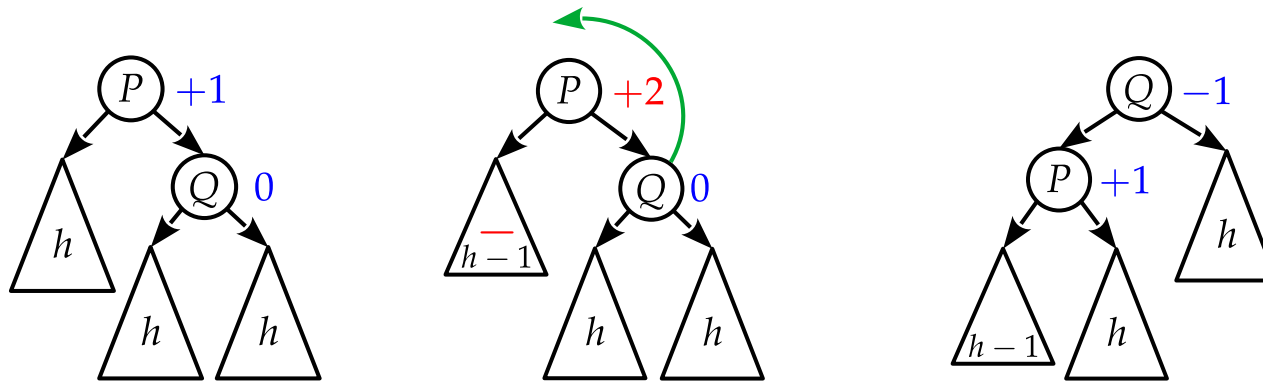
- Lijevi izlomljeni slučaj: -2 i lijevo dijete +1



- Prvo lijeva rotacija R oko Q
- Zatim desna rotacija R oko P

# AVL (7)

- Brisanje vrijednosti – uvijek brisanje kopiranjem
  - Kod takvog brisanja jednom od podstabala se može smanjiti visina



- Vidimo da je brisanje čvora u lijevom podstablu čvora  $P$  uzrokovalo disbalans
- To je desni izravnati slučaj

# AVL (8)

```
function AVLDetectRotate(n)
  if balanceFactor(n) is +2 then
     $n_1 \leftarrow \text{rightChild}(n)$ 
    if balanceFactor( $n_1$ ) is 0 or +1 then
      left rotate  $n_1$  around  $n$ 
    if balanceFactor( $n_1$ ) is -1 then
       $n_2 \leftarrow \text{leftChild}(n_1)$ 
      right rotate  $n_2$  around  $n_1$ 
      left rotate  $n_2$  around  $n$ 
  else
     $n_1 \leftarrow \text{leftChild}(n)$ 
    if balanceFactor( $n_1$ ) is 0 or -1 then
      right rotate  $n_1$  around  $n$ 
    if balanceFactor( $n_1$ ) is +1 then
       $n_2 \leftarrow \text{rightChild}(n_1)$ 
      left rotate  $n_2$  around  $n_1$ 
      right rotate  $n_2$  around  $n$ 

procedure AVLBALANCE(n)
   $p \leftarrow \text{parent}(n)$ 
  if balanceFactor(n) is -2 or +2 then
    AVLDetectRotate(n)
  if  $p$  is not nil then
    AVLBALANCE( $p$ )
```

- Složenost pretraživanja je kao i kod klasičnog binarnog stabla  $O(\log_2 n)$
- Kod upisa ili brisanja vrijednosti, vraćamo se po vertikalnoj putanji natrag to korijenskog čvora što daje kompleksnost  $O(2\log_2 n)$
- Teoretska visina AVL stabla je  $\log_2(n+1) \leq h \leq 1.44 \log_2(n+2) - 0.328$ 
  - Dokaz u Drozdeku

**Pitanja ?**