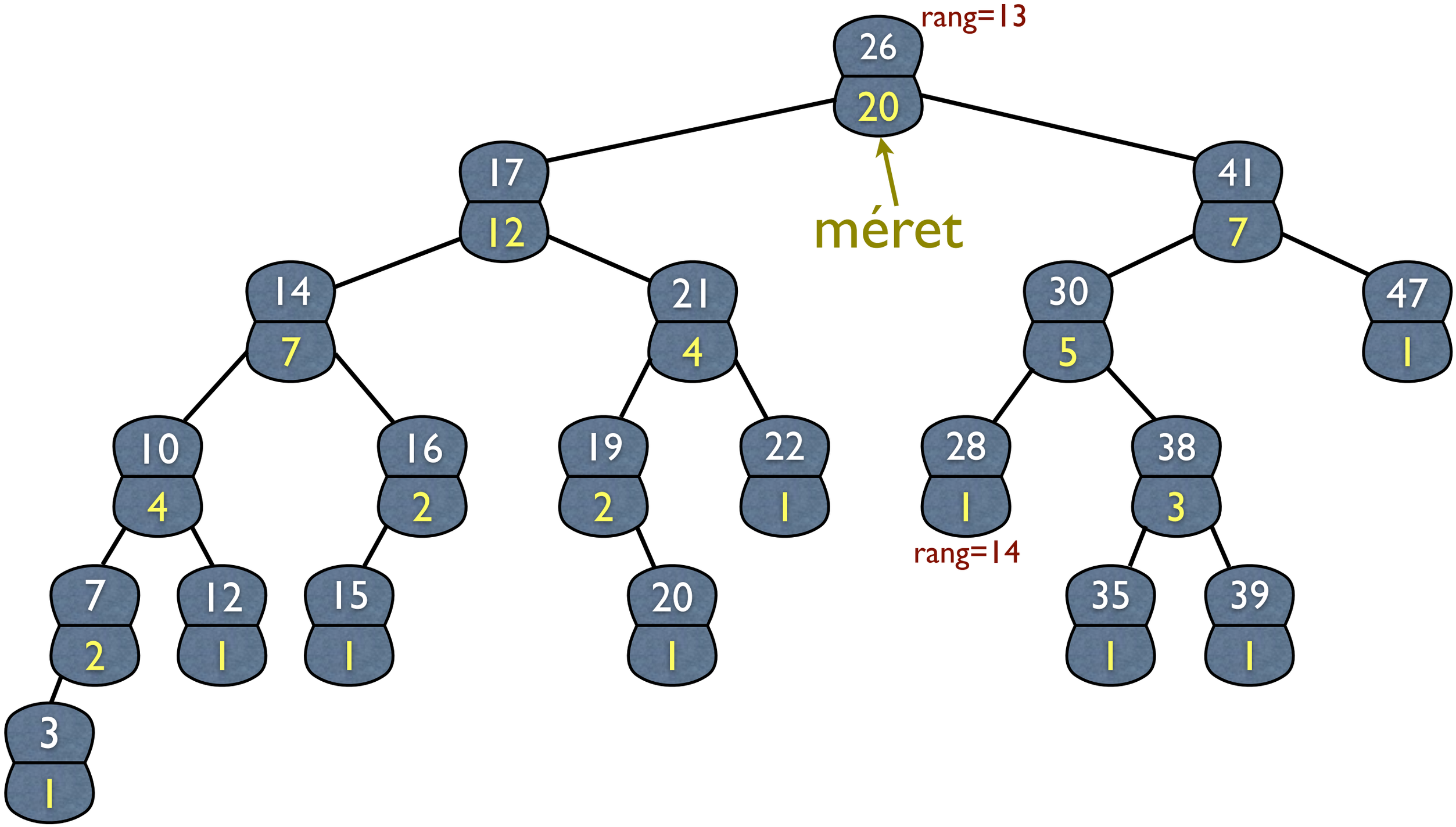
Rendezettminta-fa

- Minden p ponthoz tároljuk a p gyökerű fa belső pontjainak számát (méretét)
- Adott elem rangja: az elem sorszáma (sorrendben hányadik az adatszekezetben)
- Adott rangú elem keresése - T[r]
- Adott elem rangjának meghatározása **[2] 272-277**
- Egyéb bővítési lehetőségek (pl. intervallumfák) **[2] 279-285**

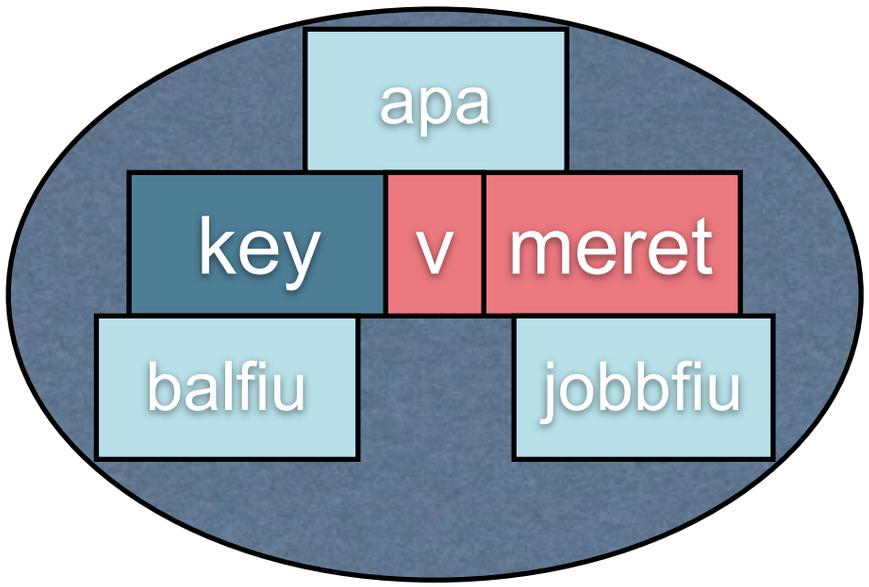
Példa


rang=13
26
20
méret
17
12
41
7
14
7
21
4
30
5
47
1
10
4
16
2
19
2
22
1
28
1
38
3
rang=14
7
2
12
1
15
1
20
1
35
1
39
1
3
1

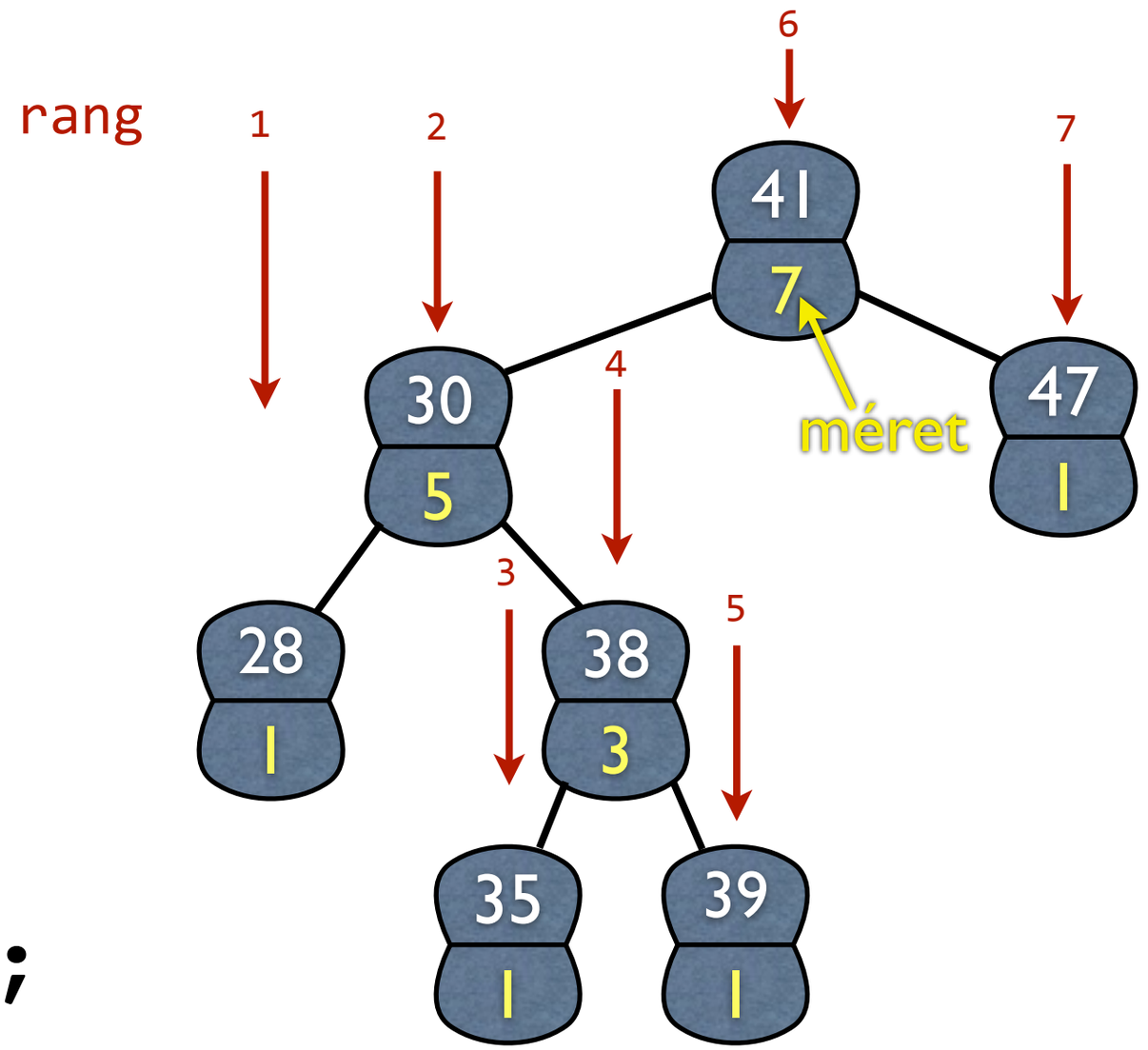
Fapont tárolása

első megközelítés, a gyakorlatban nem így tároljuk!

```
struct fapont{
    int key, v, meret ;
    fapont bal, jobb, apa ;
}
```



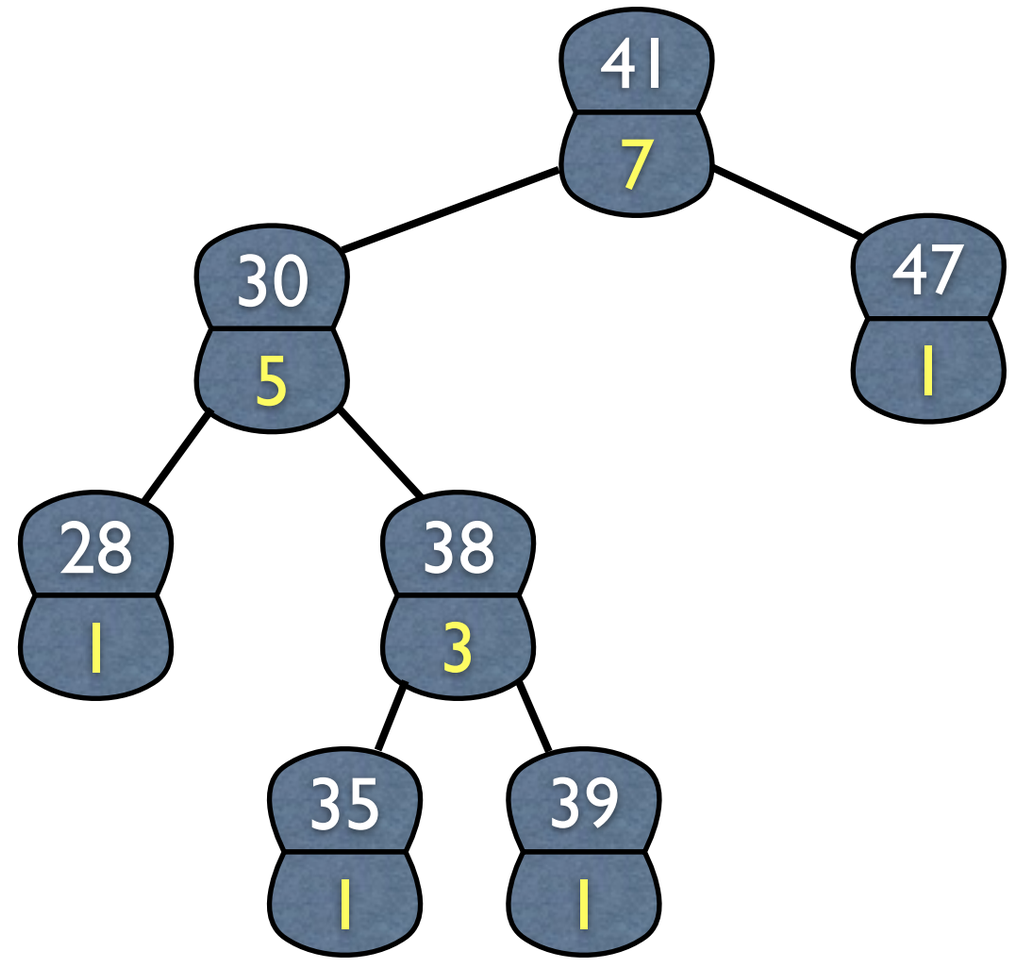
Adott rangú elem keresése




```
rmkeres(p,x) {
  r=p.bal.meret+1 ;
  if (x==r) return p ;
  if (x<r) return rmkeres(p.bal,x) ;
  else return rmkeres(p.jobb,x-r) ;
}
```

O(h)

Elem rangjának meghatározása



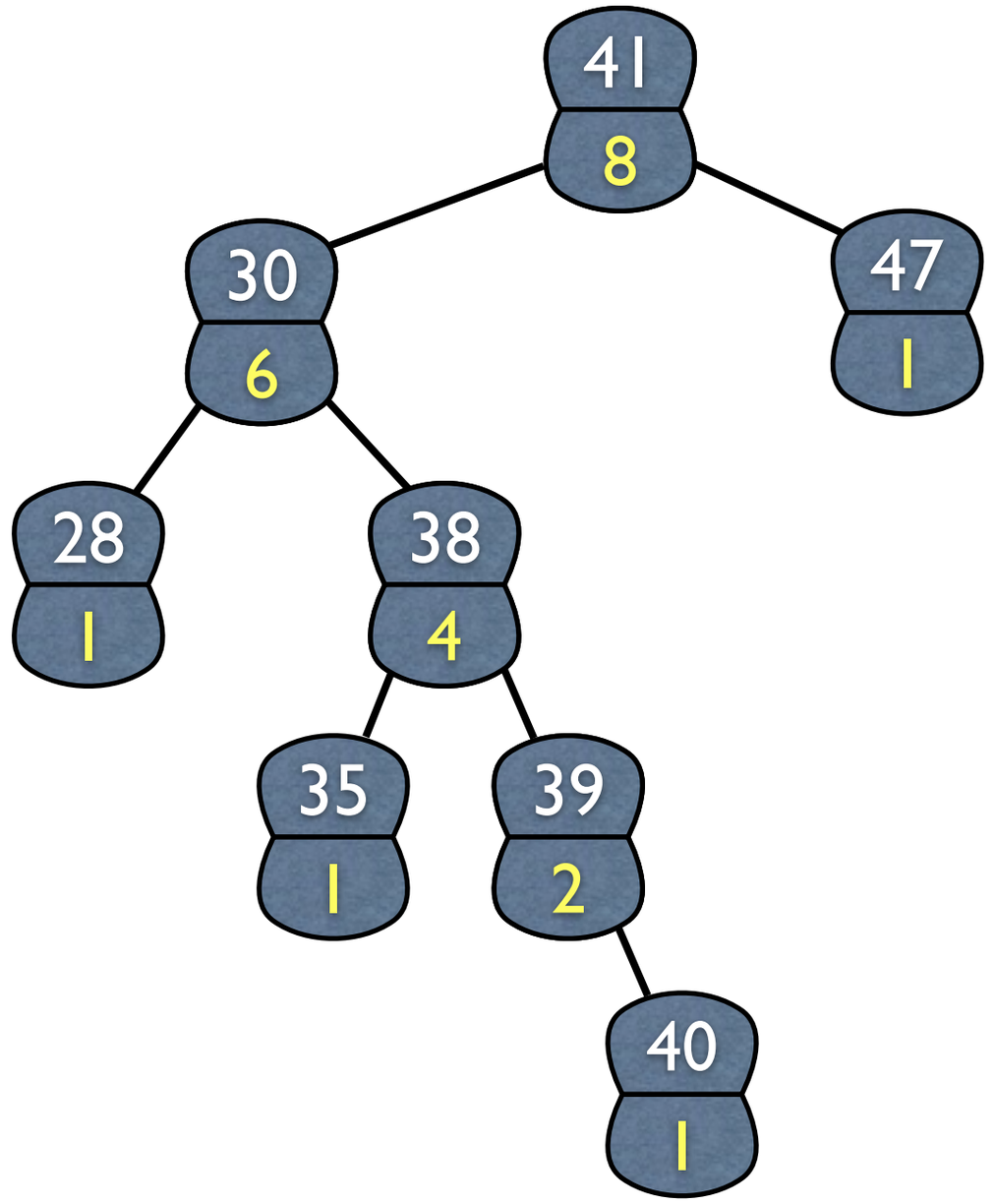

```
rmrang(p) {
  r=p.bal.meret+1 ;
  q=p ;
  while (q!=gyoker) {
    if (q==q.apa.jobb) r+=q.apa.bal.meret+1;
    q=q.apa ;
  }
  return r ;
}
```

O(h)

Méret információ fenntartása

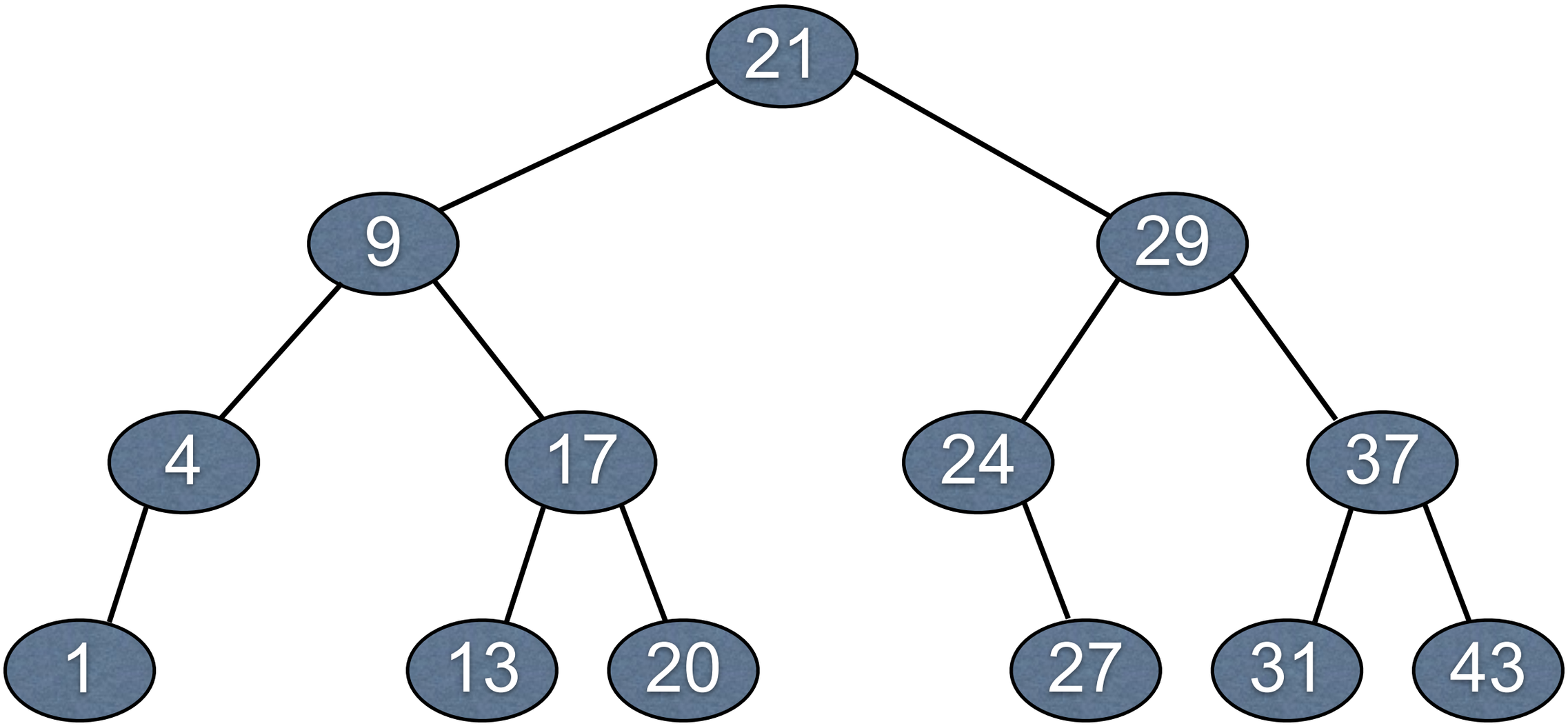
```
beszjav(p) {
  while (p!=gyoker) {
    p=p.apa;
    p.meret++ ;
  }
}

torljav(p) {
  while (p!=gyoker) {
    p=p.apa;
    p.meret-- ;
  }
}
```

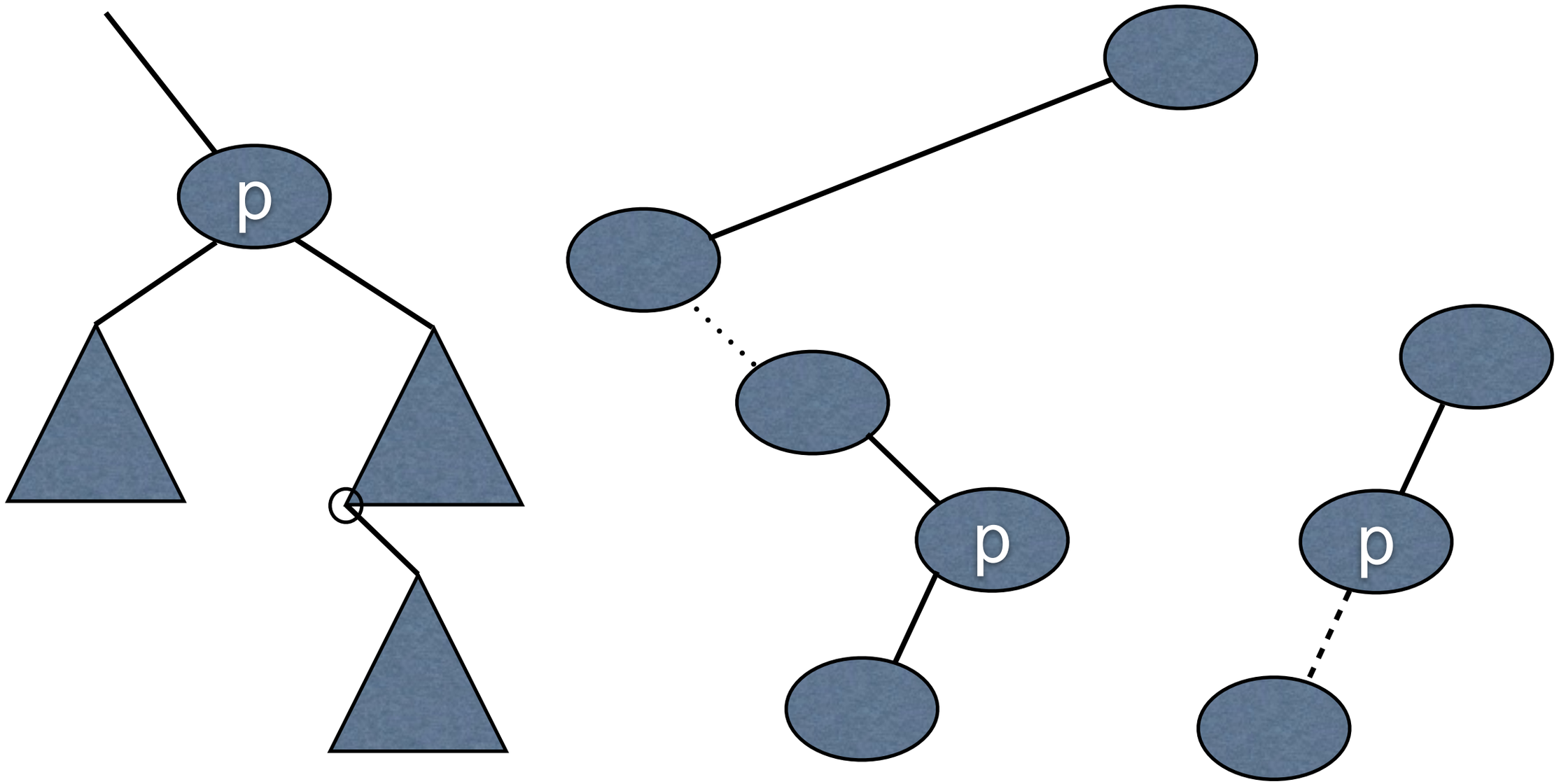


O(h)

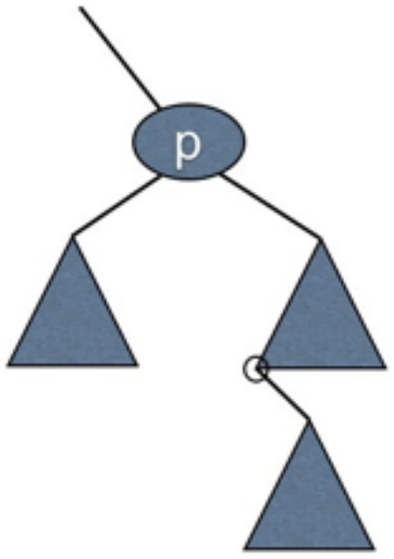
Elem rákövetkezője



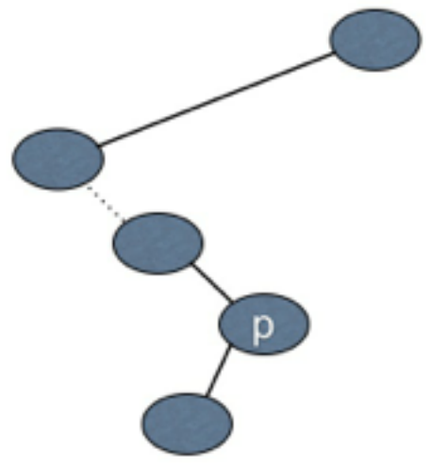
p rákövetkezője



O(h)



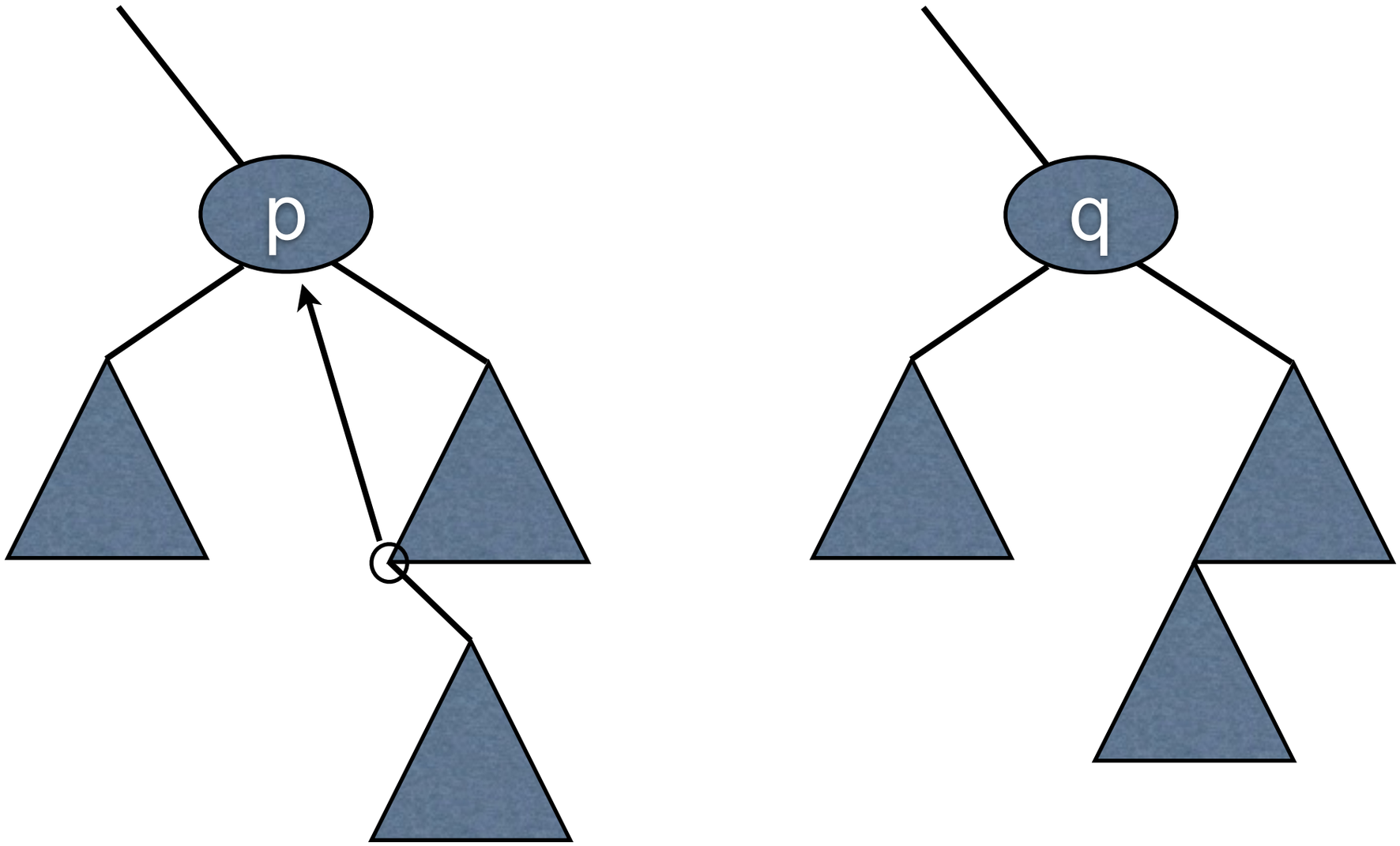
p rákövetkezője




```
fapont kovetkezo(fapont p) {
     if (p.jobb!=nil) {                  //ha van jobb fiú
       p=p.jobb ;                        //akkor a jobb részfa
       while (p.bal!=nil) p=p.bal;       //bal szélső eleme
       return p;
     } else {                            // ha nincs jobb fiú
       while (p.apa!=nil) {              // amíg van apa
          if (p.apa.bal==p)              // ha p bal gyerek
            return p.apa;                // p apja a rákövetkező
          p=p.apa ;                      // megyünk felfelé p-vel
       };
       return nil ;                      // nincs rákövetkező
     }
}
```

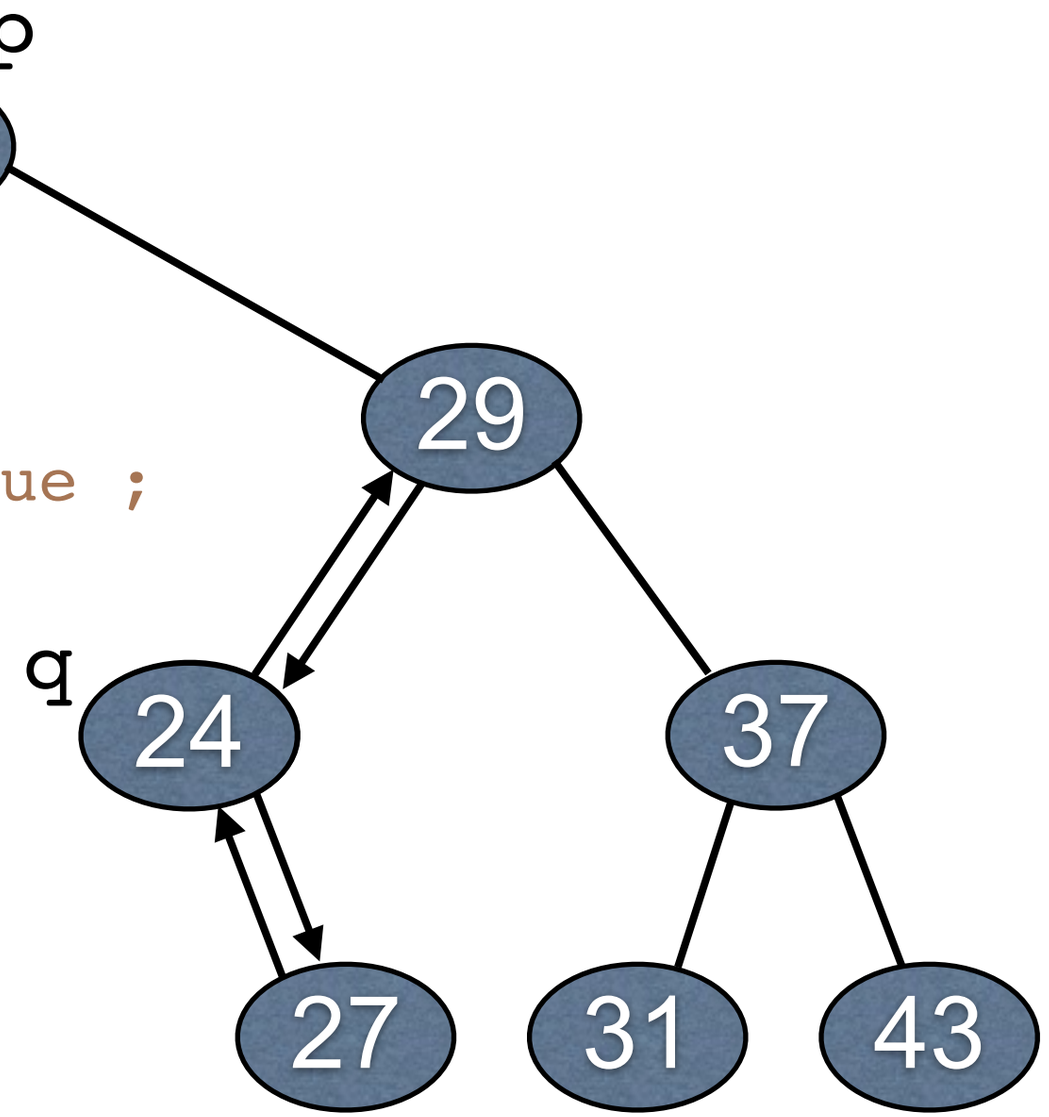
O(h)

Törlés bináris keresőfából



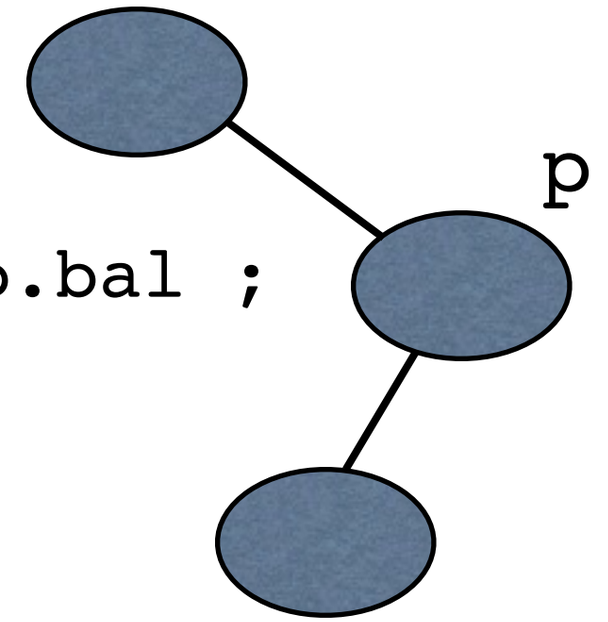
O(h)

```
void torol(fapont p) {
   if (p.jobb!=nil) {
      fapont q=kovetkezo(p) ;
      p.key=q.key ; p.value=q.value ;
      q.apa.bal=q.jobb ;
      q.jobb.apa=q.apa ;
      delete q ;
   } else {
      ...
   }
}
```



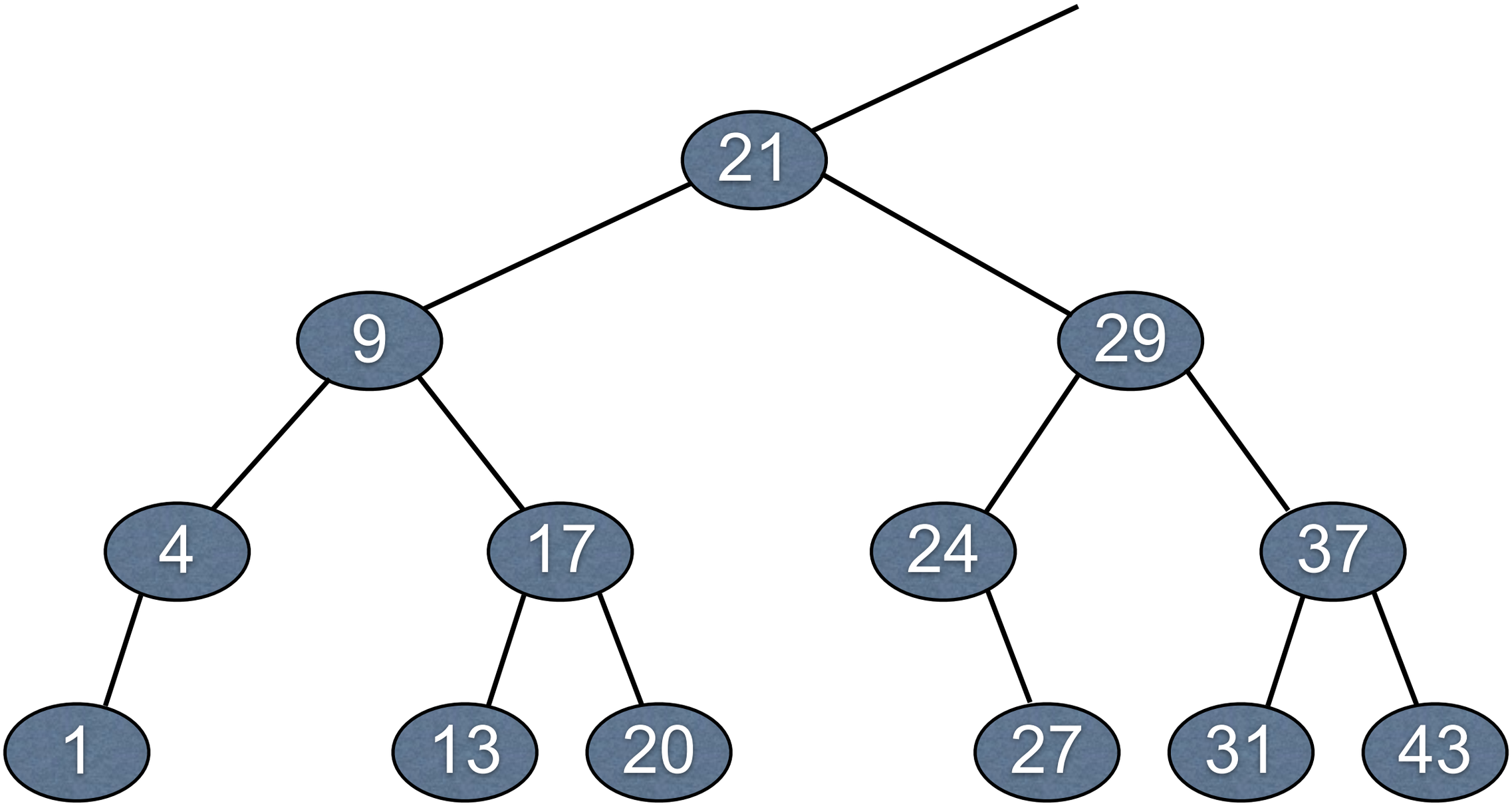
O(h)

```
void torol(fapont p) {
    if (p.jobb!=nil) {
      fapont q=kovetkezo(p) ;
      p.key=q.key ; p.value=q.value ;
      q.apa.bal=q.jobb ;
      q.jobb.apa=q.apa ;
      delete q ;
    } else {
      if (p.apa.jobb==p) p.apa.jobb=p.bal ;
      else p.apa.bal=p.bal ;
      p.bal.apa=p.apa ;
      delete p ;
    }
}
```



O(h)

Elem törlése példa

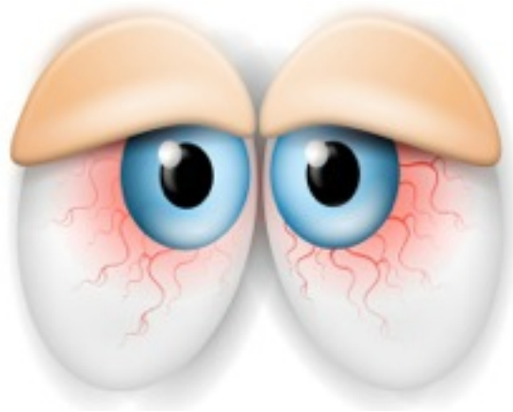


Futási idők elemzése

- Keresés
- Beszúrás
- Törlés

Mennyi a futási idő?

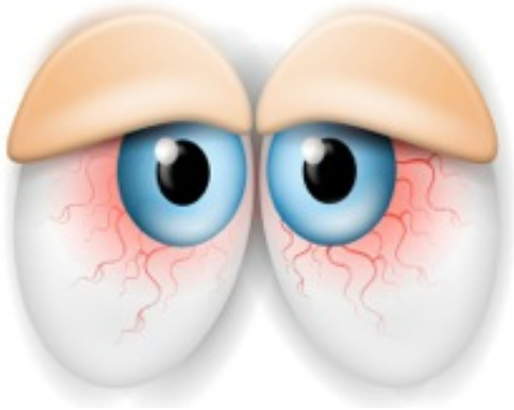
$O(h)$

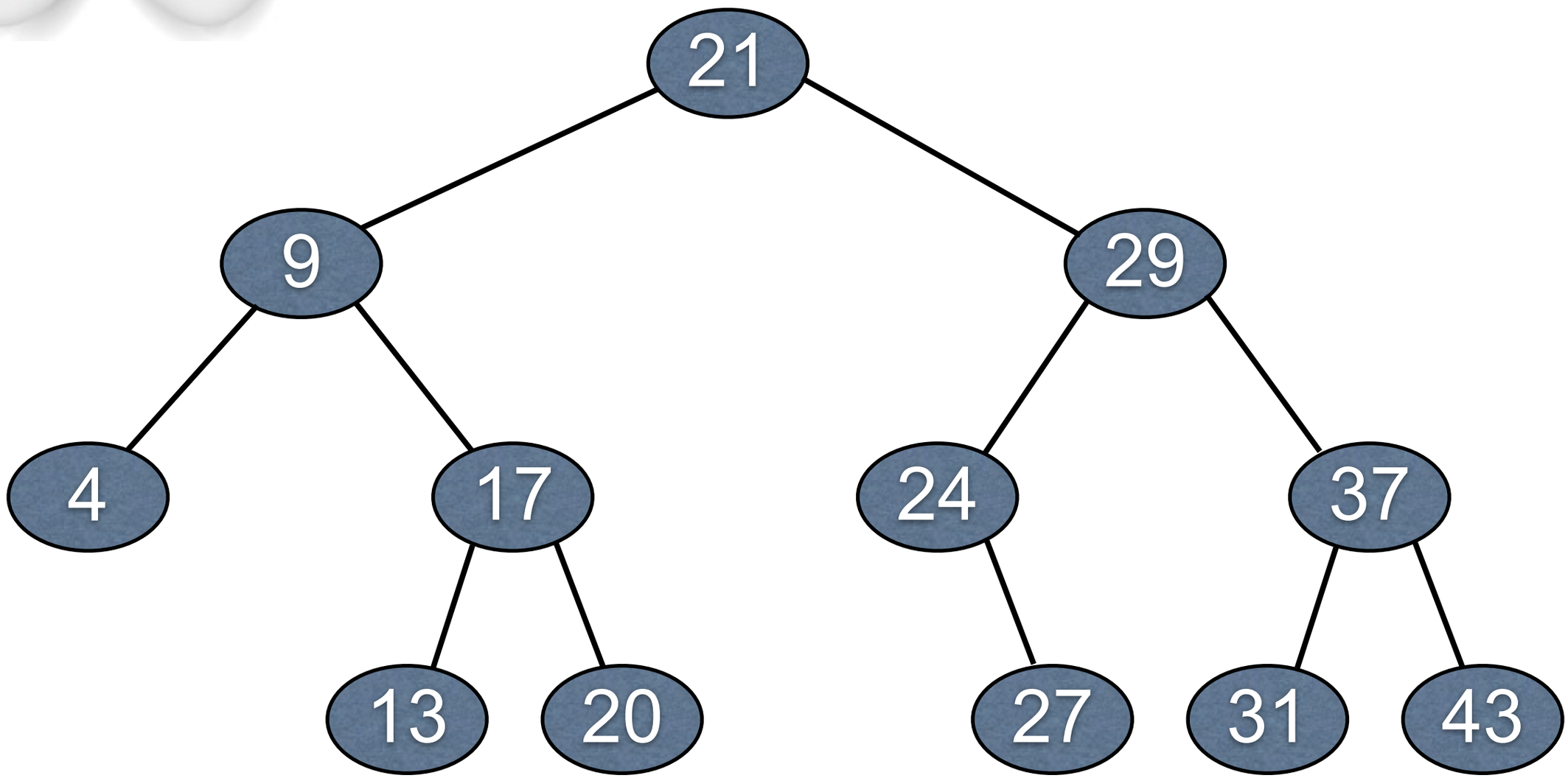


Rendezettmintafa

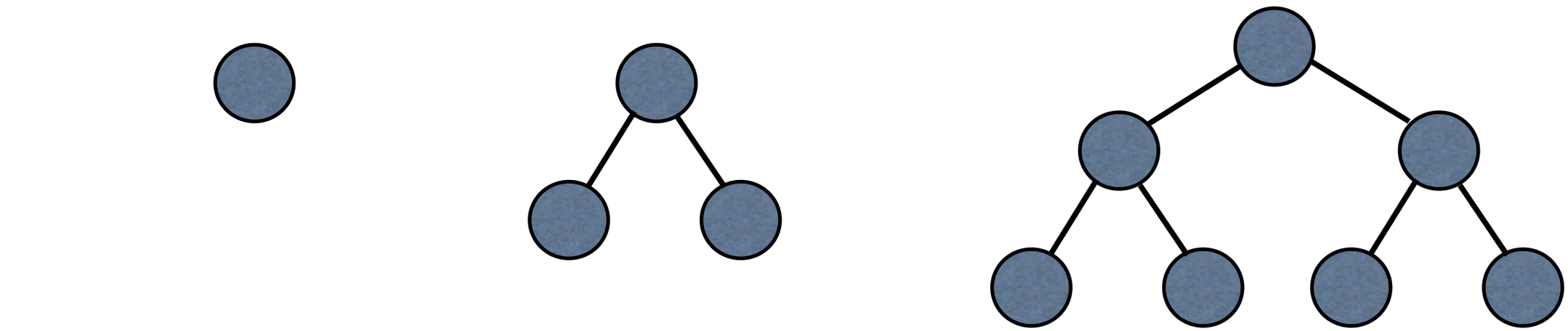
Műveletek:

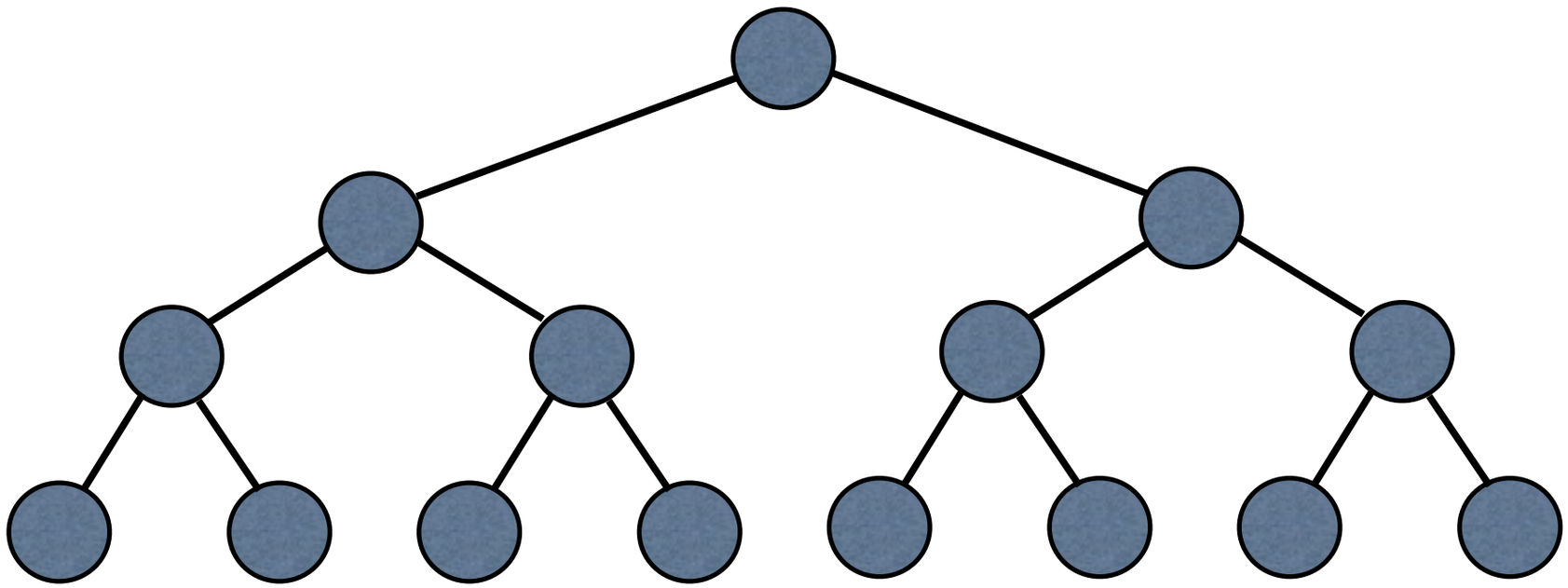
bool *beszúr*(<K> key, <V> value)	O(*h*)
fapont *keres*(<K> x)	O(*h*)
bool *töröl*(fapont p)	O(*h*)
fapont *rákövetkező*(fapont p)	O(*h*)
***mindetkiir(*fapont p)**	O(n)
mindenttöröl(fapont p)	O(*h*)
int *elemszám*(fapont p)	O(*1*)
int *rang*(fapont p)	O(*h*)
fapont *rangkeres*(int x)	O(*h*)




21
9
29
4
17
24
37
13
20
27
31
43

Teljes bináris keresőfa





h	n(h)	2
1	**1**	2
2	**3**	4
3	**7**	8
4	**15**	16
5	**31**	32
k	**2**	2

n pontot tartalmazó teljes bináris keresőfa magassága

h	n(h)	2
1	1	2
2	3	4
3	7	8
4	15	16
5	31	32
h	2	2

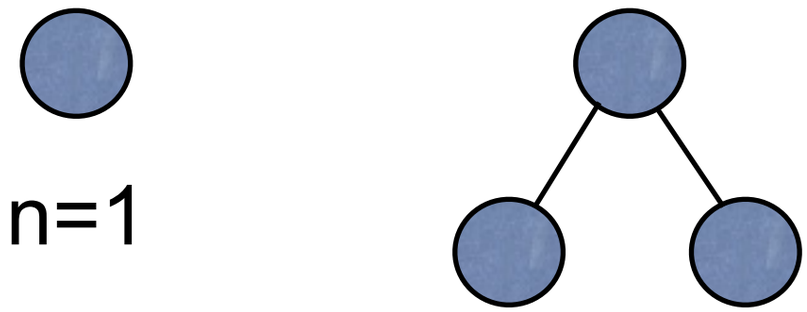
$$n = 2^{\mathrm{h}} - 1$$

$$n + 1 = 2^{\mathrm{h}}$$

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

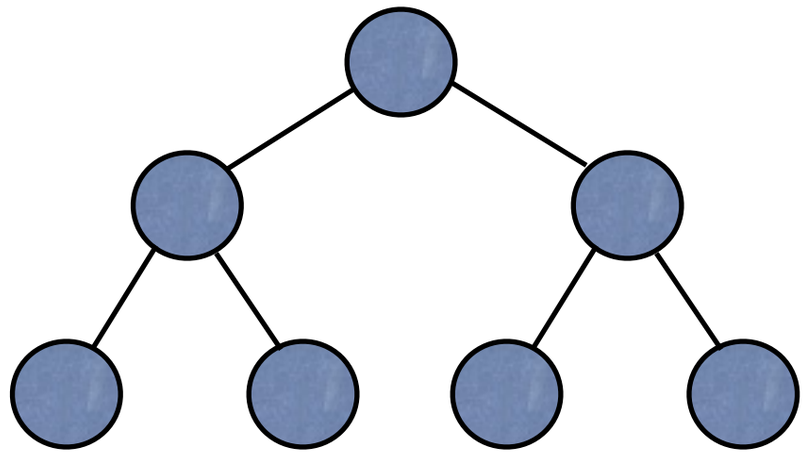
$$\mathrm{h} = \mathbf{O}(logn)$$

Lehet-e egy bináris keresőfa “teljesen kiegyensúlyozott” ???



n=1

n=3



n=7

n=2 ?

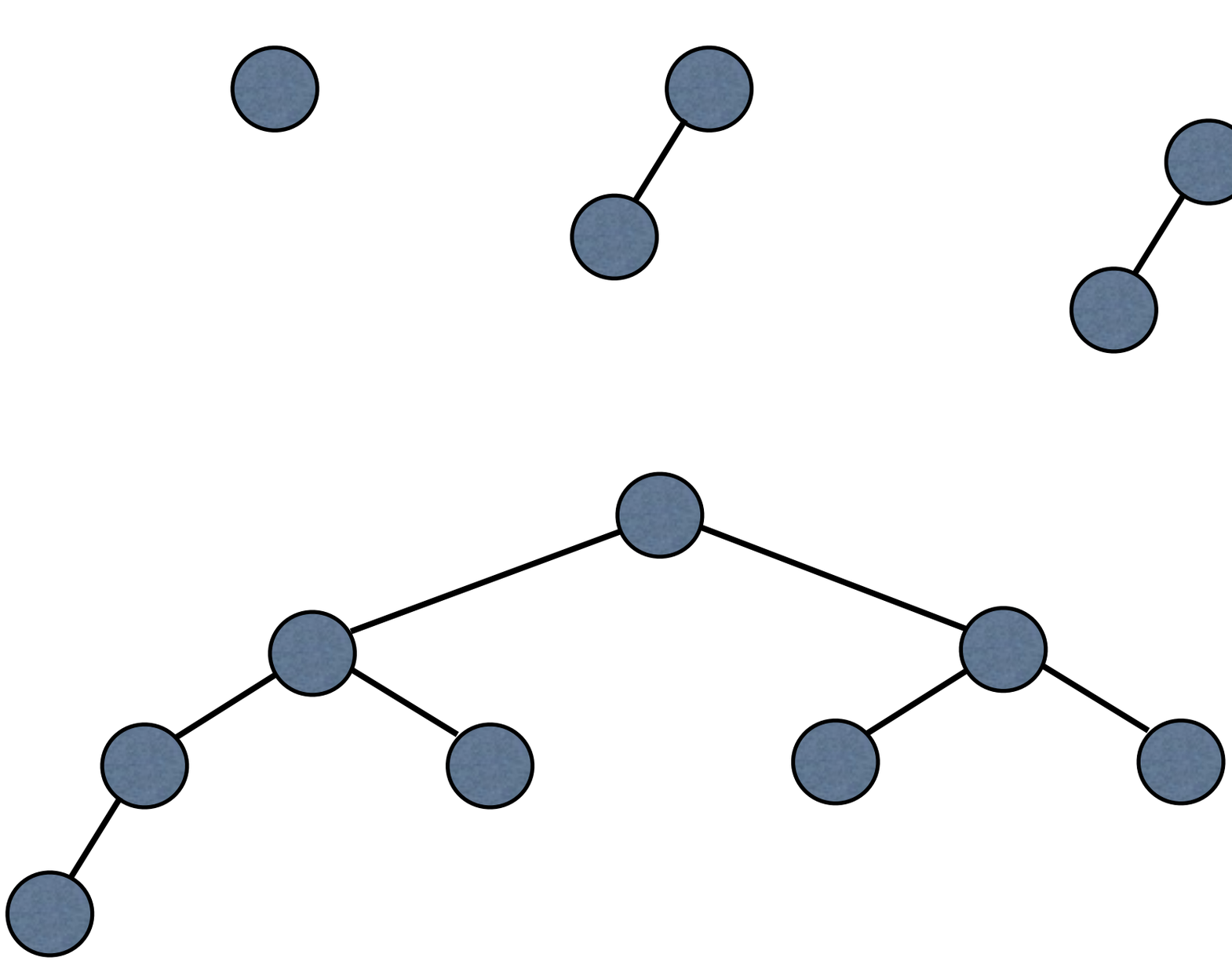
n=4

n=5

n=6 ?

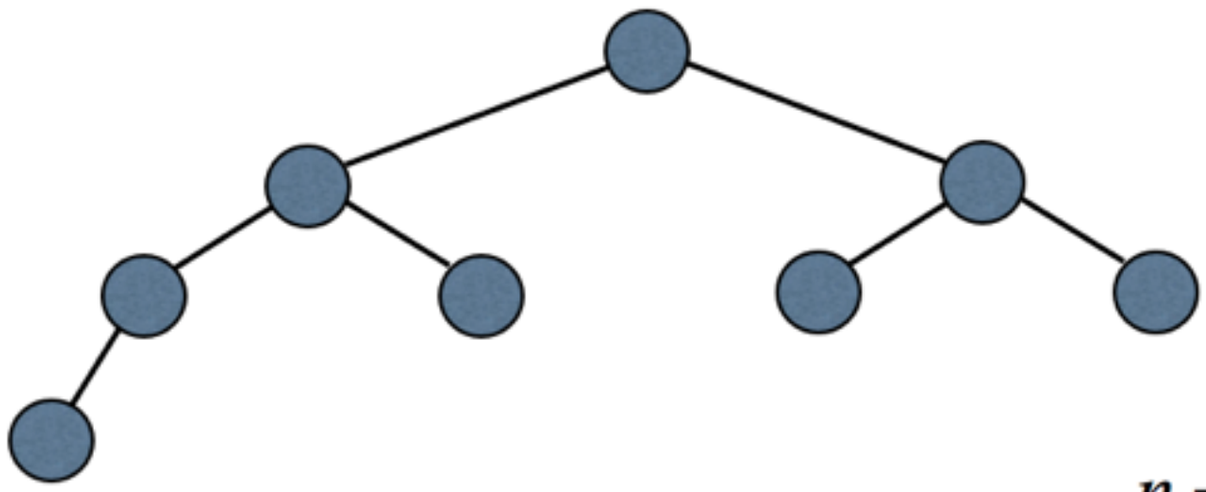
n=8 ... ?

Majdnem teljes bináris keresőfa



h	n(h)	2
1	1	1
2	2	2
3	4	4
4	8	8
5	16	16
k	2	2

n pontot tartalmazó majdnem teljes bináris keresőfa magassága



h	n(h)	2
1	1	1
2	2	2
3	4	4
4	8	8
5	16	16
k	2	2

$$n = 2^{h-1}$$

$$\log_2 n = \log_2 2^{h-1} = (h-1)\log_2 2 = h-1$$

$$\log_2 n + 1 = h$$

$$\mathrm{h} = \mathrm{O}(\log n)$$

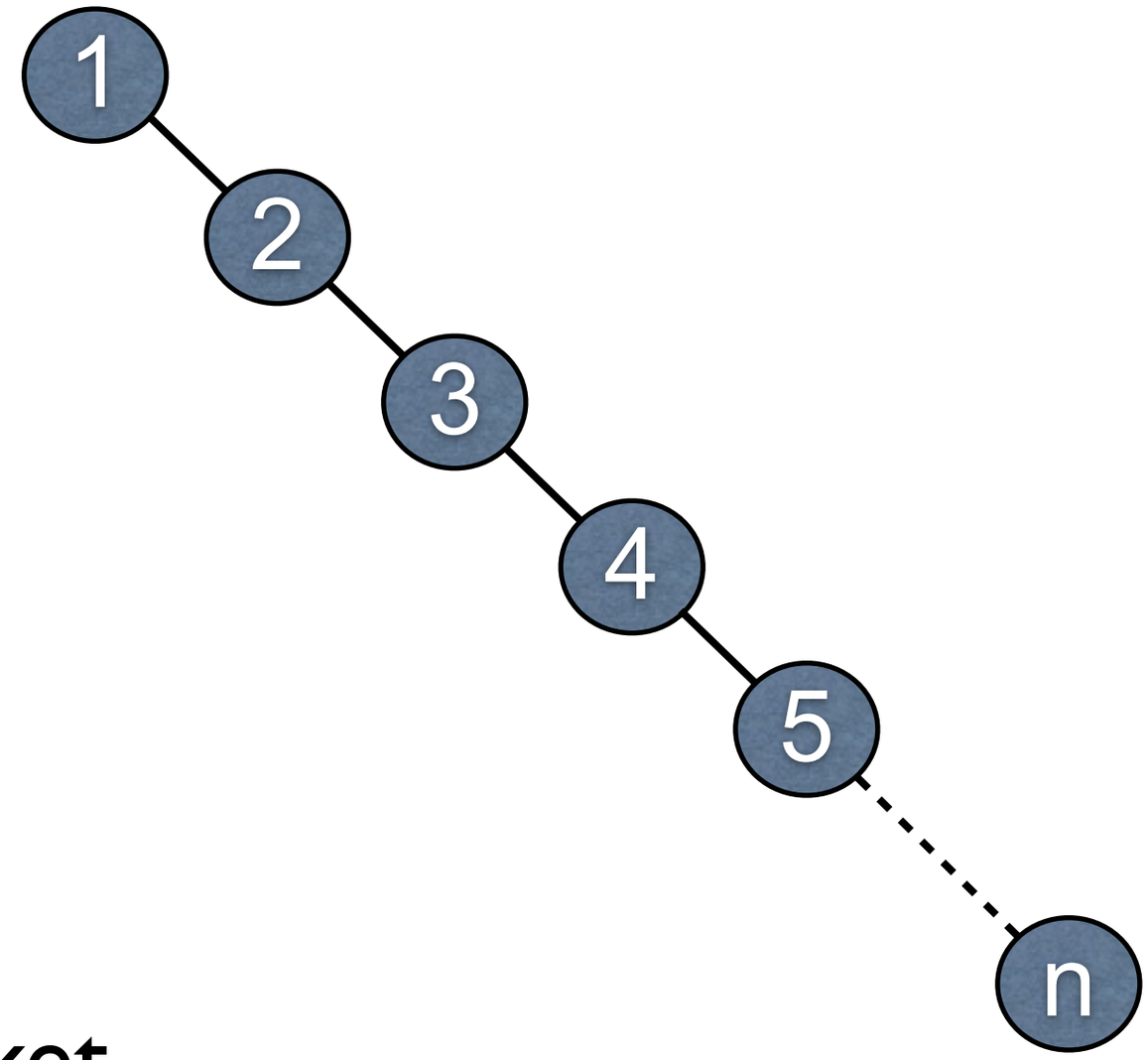
Véletlen építésű bináris keresőfa

- Adott n egymástól különböző kulcs, melyekből bináris keresőfát építünk úgy, hogy a kulcsokat valamilyen sorrendben egymás után beszúrjuk a kezdetben üres fába.
- Ha itt minden sorrend, vagyis az n kulcsnak mind az n! permutációja egyformán valószínű, akkor a kapott fát véletlen építésű bináris keresőfának nevezzük.
- **Tétel:** Egy n különböző kulcsot tartalmazó véletlen építésű bináris keresőfa várható magassága: $O(\log \mathbf{n})$.

[2] 241-244

Szúrjuk be rendre az

- 1
- 2
- 3
- 4
- 5
- ...
- n elemeket



Legrosszabb eset

h=n

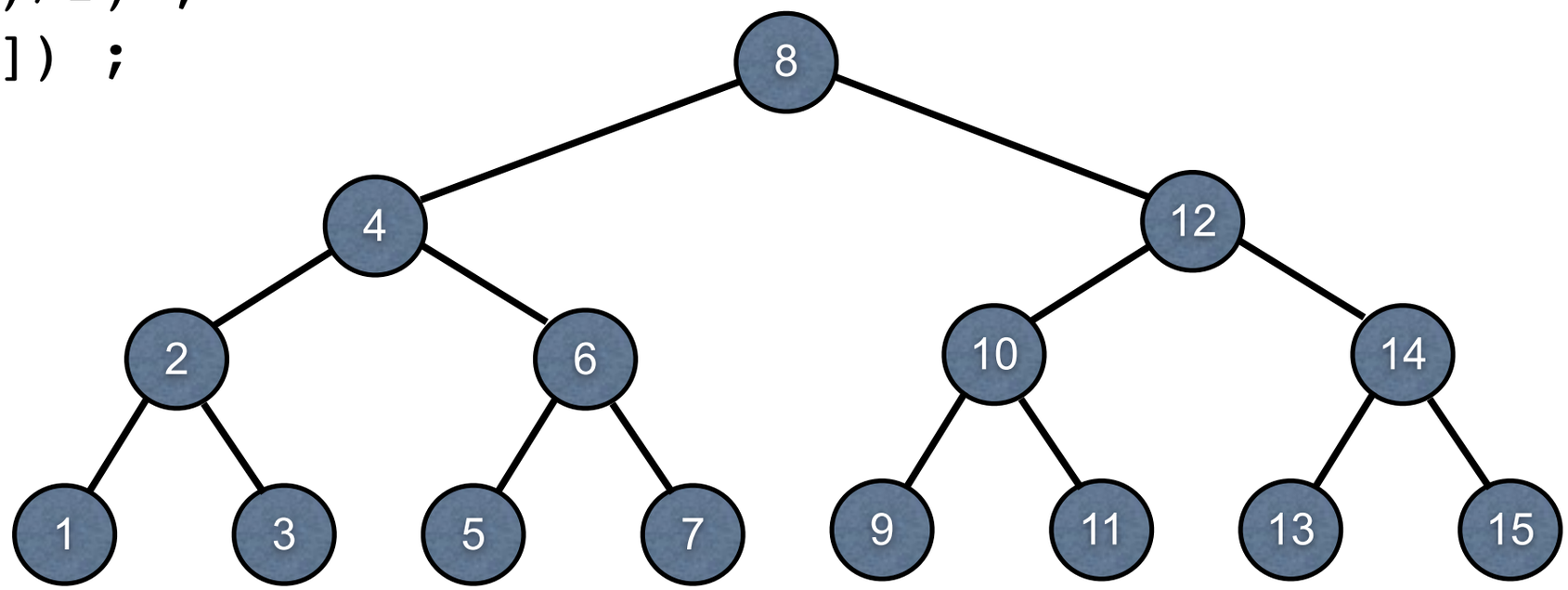
Kiegyensúlyozott bináris keresőfák

Hogy lehetne javítani?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

más sorrendben kell beszúrni:

```
f(i,j) {
   if (i<=j) {
      f=int((i+j)/2) ;
      beszur(T[f]) ;
      f(i,f-1) ;
      f(f+1,j) ;
   }
}
```



Optimális bináris keresőfa

Adott kulcsoknak egy $K=(k_1,...,k_n)$ sorozata

Minden k_i kulcshoz ismert annak p_i előfordulási valószínűsége és

Minden $d_i=(k_i,k_{i+1})$ intervallumhoz ismert annak q_i előfordulási valószínűsége, hogy arra az intervallumra keresünk, $d_0= (-\infty,k_i)$, $d_n= (k_n,\infty)$

Optimális bináris keresőfa építhető
(pl. dinamikus programozás módszerével)

Építés: $O(n^2)$

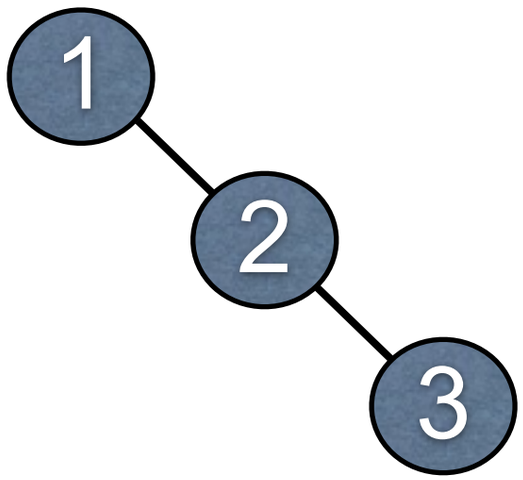
[2] 314-321

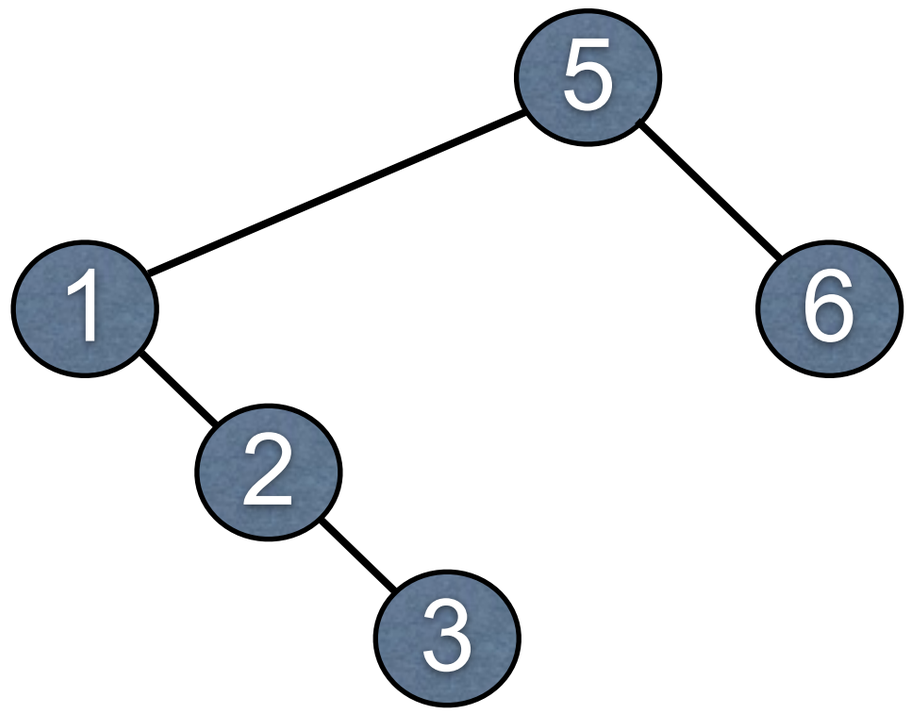
De az elemek
egyesével is jöhetnek!

Hogy lehetne javítani?

Mivel az elemek egyesével jönnek

Minden beszúrás (és törlés) után rögtön javítani kellene





AVL fák

Az **AVL-fa** alatt egy ön-kiegyensúlyozó bináris keresőfát értünk.

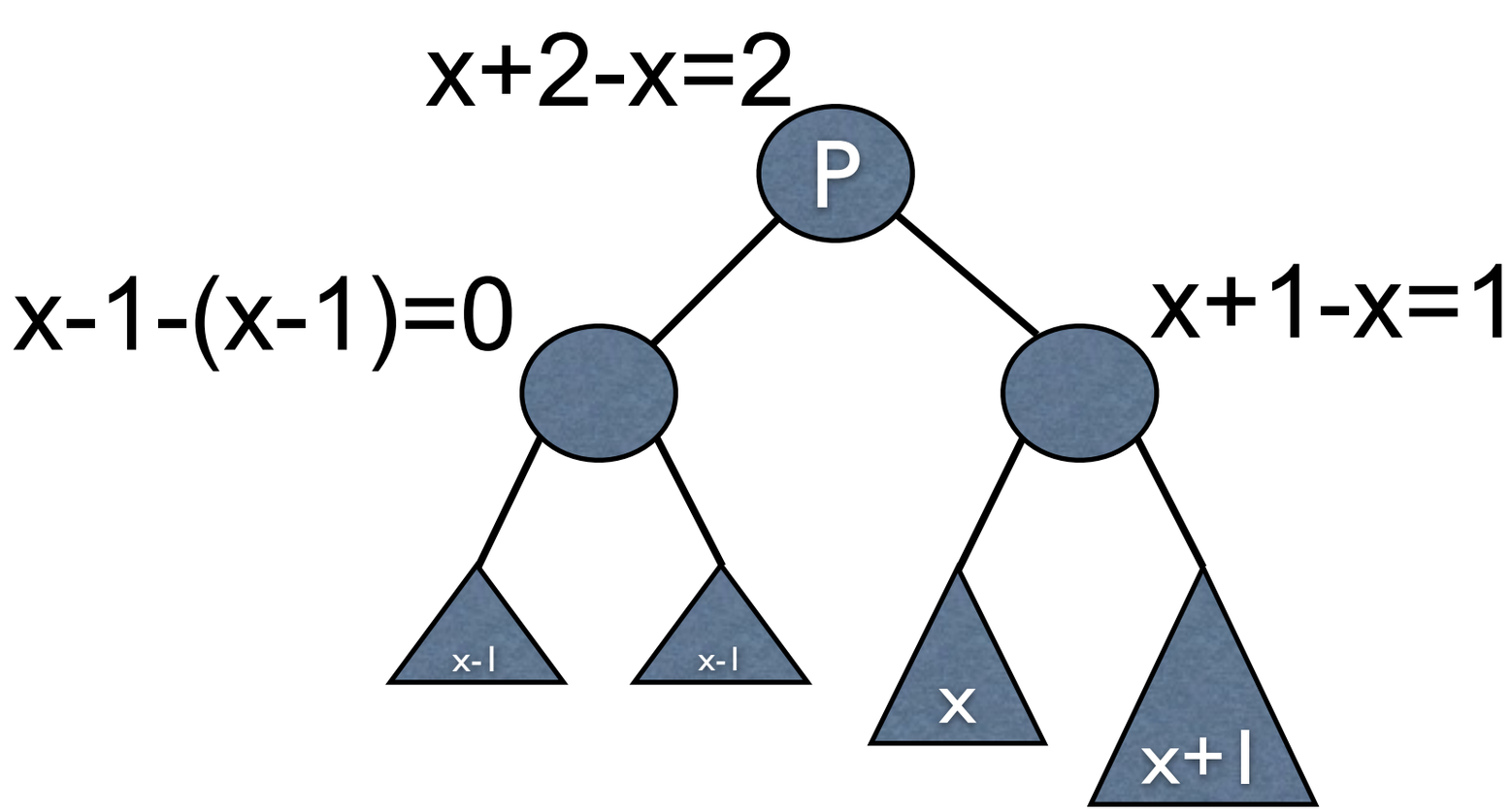
Egy AVL-fában bármely csúcspont két részfájának magassága közti különbség legfeljebb egy.

Az AVL-fa nevét két feltalálójáról G. M. Adelson-Velsky-ről és E. M. Landis -ról kapta, akik 1963 -ban publikálták [4] *An algorithm for the organization of information (Egy algoritmus az információ szervezettségéhez)* című cikkükben.

[3]

p pont egyensúlyfaktora

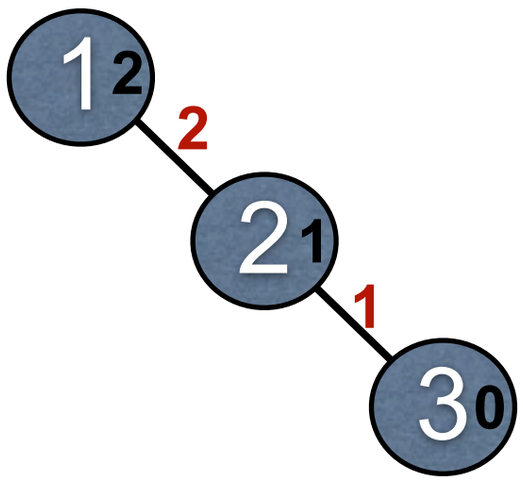
egy(p)=h(p.jobb)-h(p.bal)



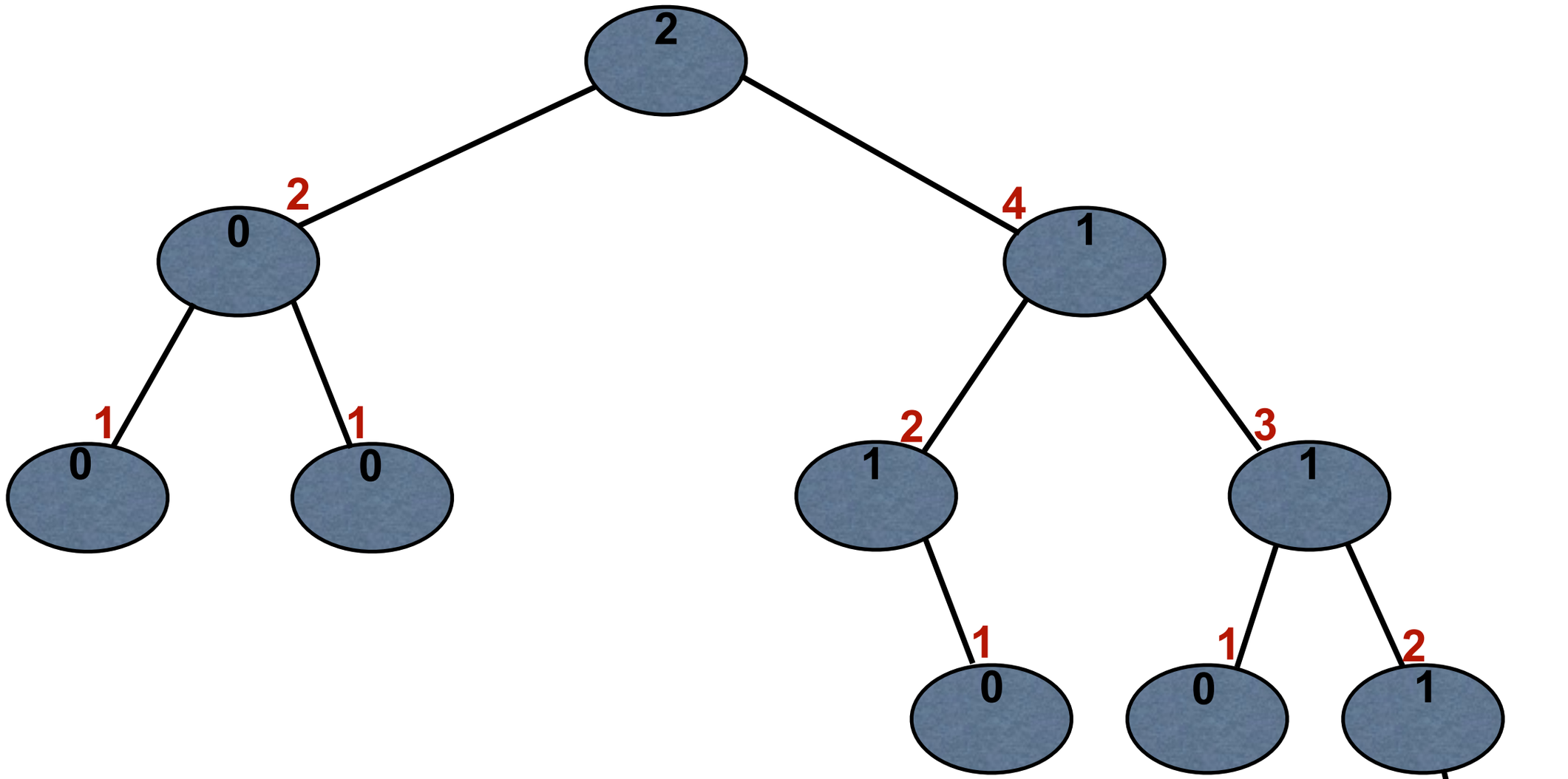
Példa

h(p)=1+max{h(p.jobb),h(p.bal)}

egy(p)=h(p.jobb)-h(p.bal)



Példa



egy(p)=h(p.jobb)-h(p.bal)

h(p)=1+max{h(p.jobb),h(p.bal)}

AVL fa

Def A $P \in F$ pont (magasság-)egyensúlya:

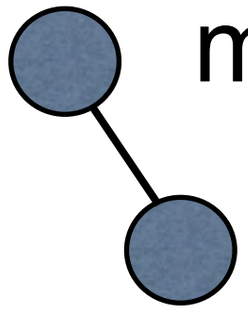
$$Egy(P) = h(Job(P)) - h(Bal(P))$$

Def Az F binfa AVL-fa, ha $(\forall P \in F)(-1 \leq Egy(P) \leq 1)$

tétel Ha F AVL-fa, akkor $h(F) \leq 1.44 \cdot lg(n+1)$, ahol n az F fa pontjainak számát jelöli.

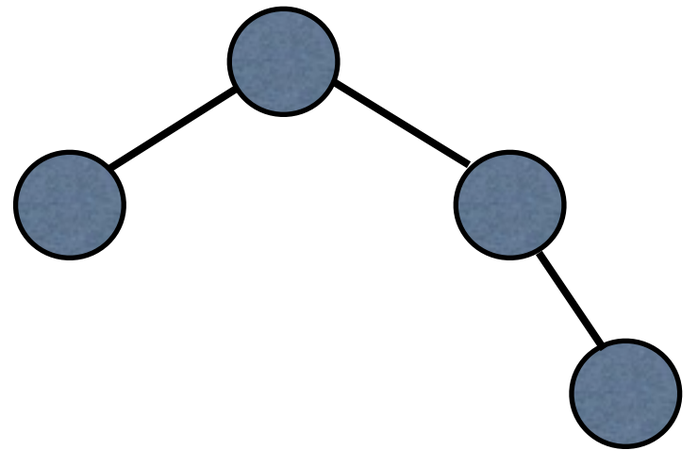
Legkevesebb pontot tartalmazó h magasságú AVL fa

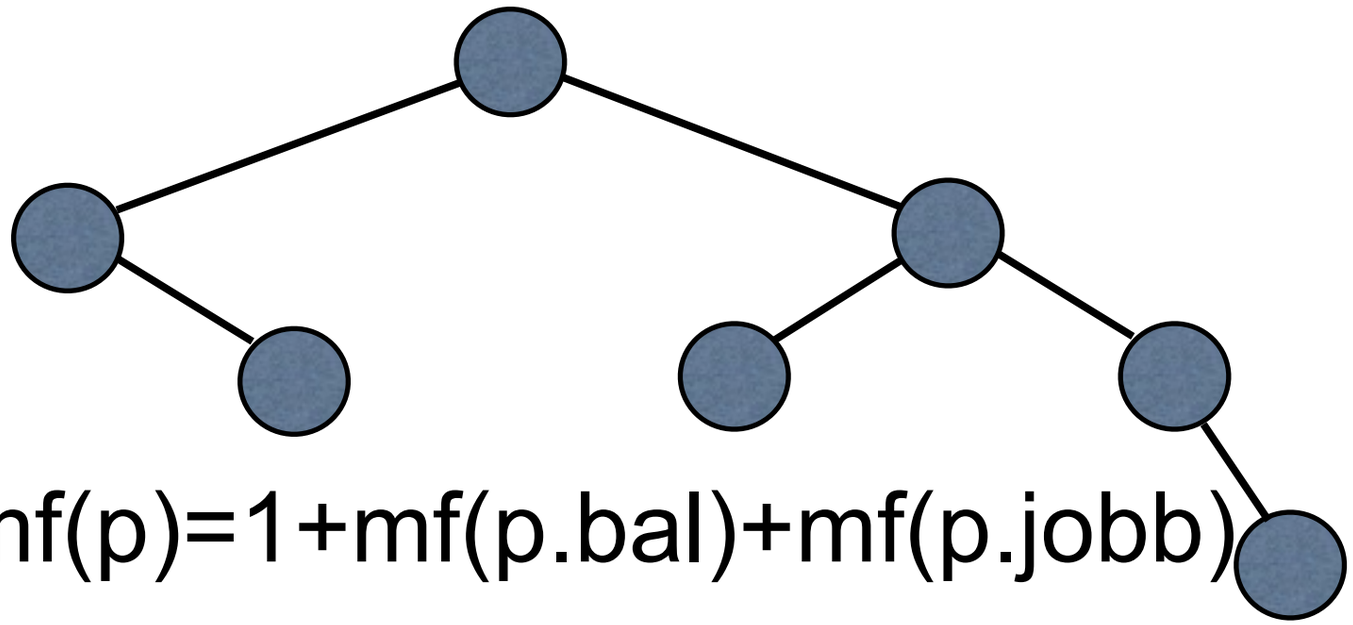
mf(1)=1



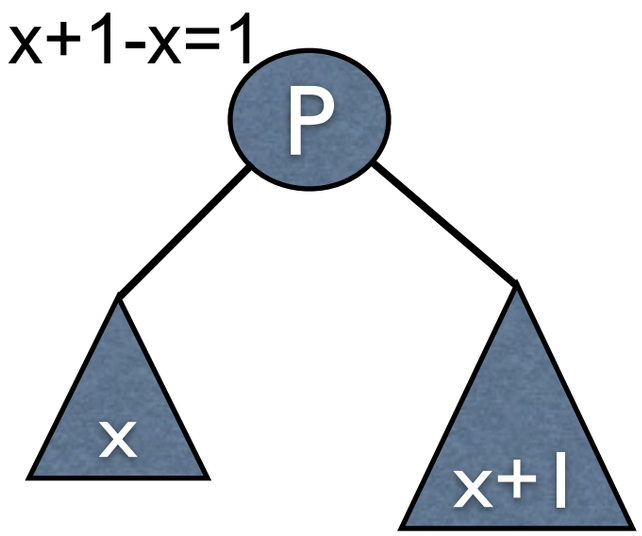
mf(2)=2

mf(3)=1+mf(1)+mf(2)=4





mf(p)=1+mf(p.bal)+mf(p.jobb)



mf(h)=fib(h)+1

fib(n)=fib(n-2)+fib(n-1)

[1] 276

Def A $P \in F$ pont (magasság-)egyensúlya:

$$Egy(P) = h(Job(P)) - h(Bal(P))$$

Def Az F binfa AVL-fa, ha $(\forall P \in F)(-1 \leq Egy(P) \leq 1)$

tétel Ha F AVL-fa, akkor $h(F) \leq 1.44 \cdot lg(n+1)$, ahol n az F fa pontjainak számát jelöli.

Biz Legyen N_m az m magasságú, legkevesebb pontot tartalmazó AVL-fa pontjainak száma.
Ekkor $N_m \leq |F|$, ha F AVL-fa és $h(F) = m$.
$N_0 = 0$, $N_1 = 1$, $N_2 = 2$ és $N_m = 1 + N_{m-2} + N_{m-1}$, ha m > 1.
Legyen $B_i = N_i + 1$. Ekkor $B_0 = 1$,$B_1 = 2$ és $B_m = B_{m-2} + B_{m-1}$ ha (m > 1).
lemma $\Phi^m \leq B_m$, ahol $\Phi = (1+\sqrt{5})/2$.
$1 = \Phi^0 \leq B_0$, $\Phi \leq B_1$. Teljes indukció alapján, ha $2, \ldots, m-1$-re áll az egyenlőtlenség
$B_m = B_{m-2} + B_{m-1} \leq \Phi^{m-2} + \Phi^{m-1} = \Phi^{m-2}(1+\Phi)$.
Viszont $(1+\Phi) = \Phi^2$, így a lemmát igazoltuk.

Tehát $\Phi^m \leq B_m = N_m + 1 \leq n+1$, azaz
$m \cdot \log \Phi \leq \log(n+1)$.
Következésképpen $h(F) = m \leq \frac{1}{\log \Phi} \log(n+1) = 1.44 \cdot \log(n+1)$. [2]

□

Műveletek

Mind a beszúrás mind pedig a törlés esetén az algoritmus elsőként a bináris keresőfáknál tanultaknak megfelelően beszúrja illetve törli az elemet, majd a beszúráshoz illetve törléshez tartozó bővítőúton illetve törlőúton felfele haladva lokális forgatásokkal helyreállítja a tulajdonságot.

Def $U = [P_0, \ldots, P_m]$ X-bővítőút, ha

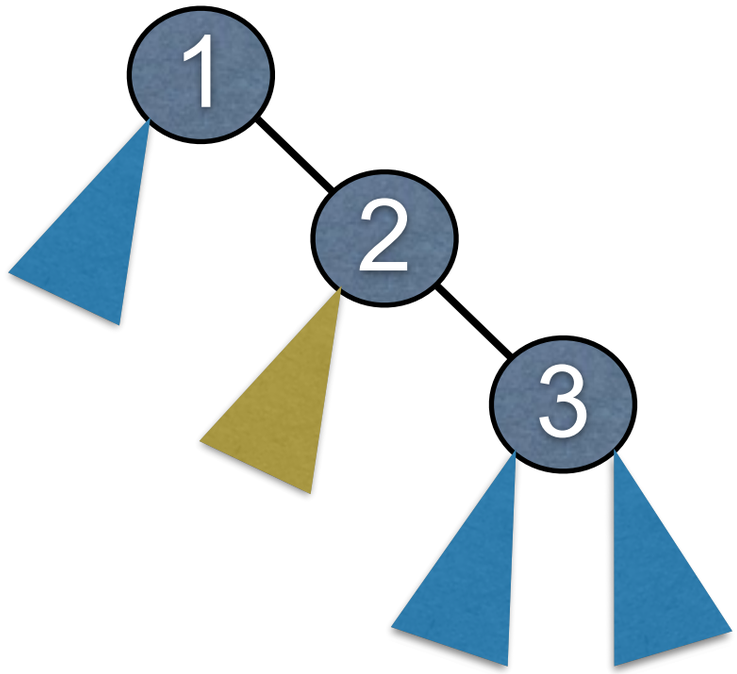
- ha $X < Adat(P_i)$ akkor $P_{i+1} = Bal(P_i)$ és ha $X > Adat(P_i)$ akkor $P_{i+1} = Jobb(P_i)$ $i < m$ esetén
- ha $X < Adat(P_m)$ akkor $Bal(P_m) = Nil$ és ha $X > Adat(P_m)$ akkor $Jobb(P_m) = Nil$

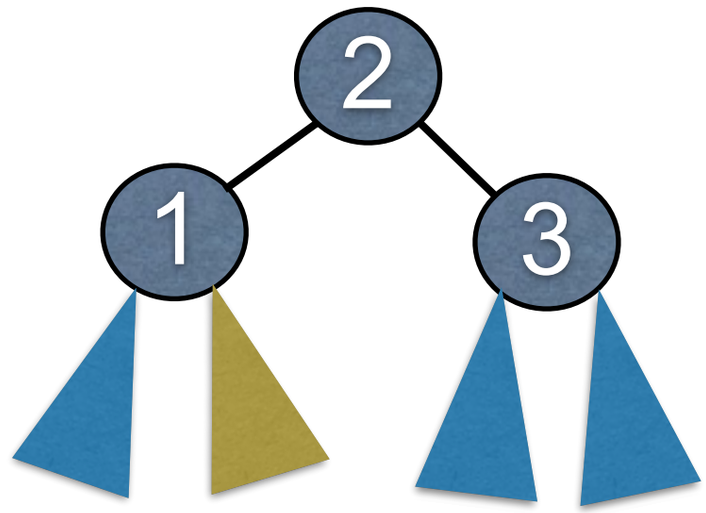
Def $U = [P_0, \ldots, P_m]$ X-törlőút, ha

- $\exists 0 \leq t \leq m$ amelyre $Adat(P_t) = X$ és ha $X < Adat(P_i)$ akkor $P_{i+1} = Bal(P_i)$ és ha $X > Adat(P_i)$ akkor $P_{i+1} = Jobb(P_i)$ $i < t$ esetén
- t = m és $Bal(P_t) = Nil$ vagy $Jobb(P_t) = Nil$ vagy $t < m$ és $Bal(P_t) \neq Nil \wedge Jobb(P_t) \neq Nil$ és $P_{t+1} = Jobb(P_t)$,$P_{i+1} = Bal(P_i)$ ha $(t < i < m)$

[2]

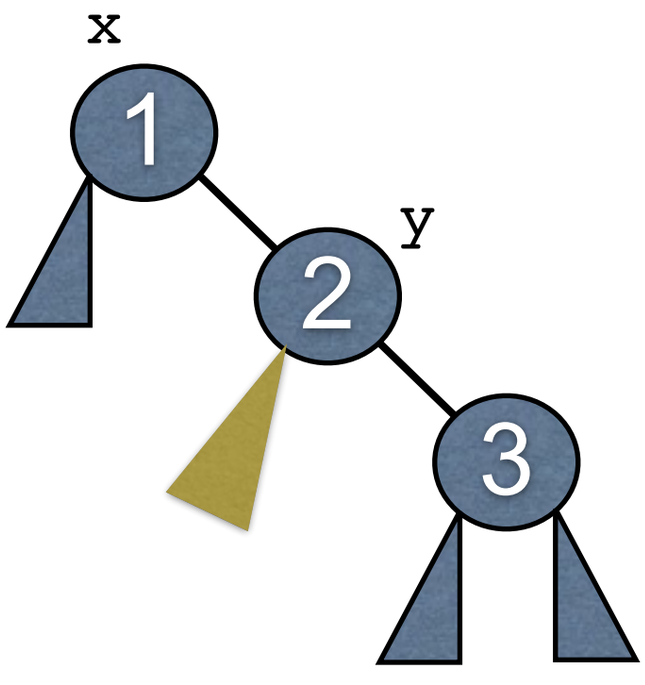
Forgatás

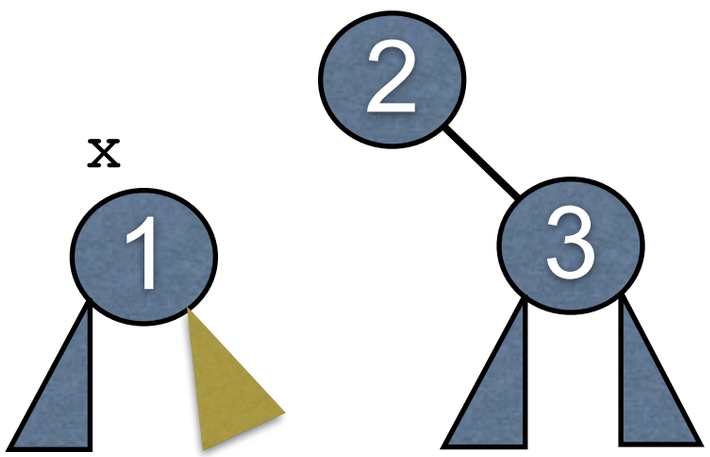




[1] 279-280

Forgatás



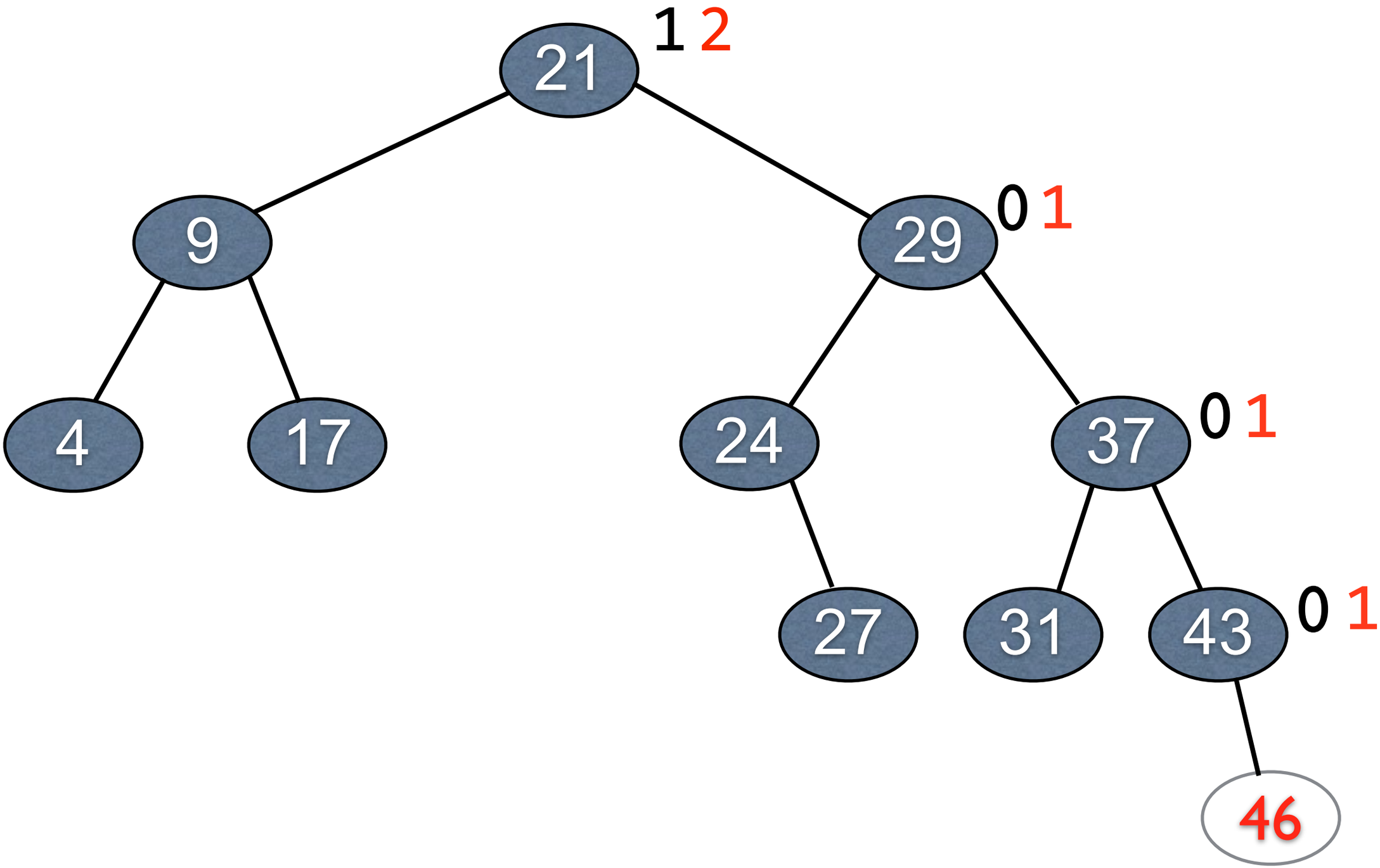



```
void balraforgat(fapont x) {
     fapont y ;
     y=x.jobb ;
     x.jobb=y.bal ;
     if (x.jobb!=nil) x.jobb.apa=x;
     y.apa=x.apa;
     if (y.apa==nil) gyoker=y ;
     else {
        if (x==y.apa.bal) y.apa.bal=y ;
        else y.apa.jobb=y ;
     }
     y.bal=x ;
     x.apa=y ;
}
```

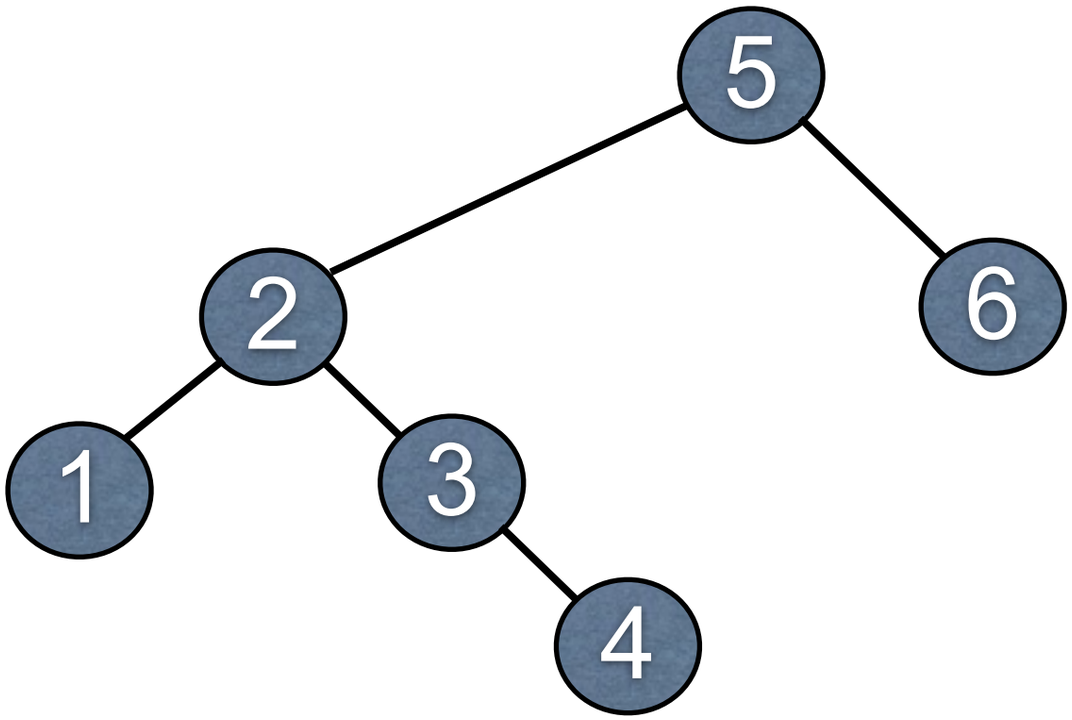
$\mathrm{O}(1)$

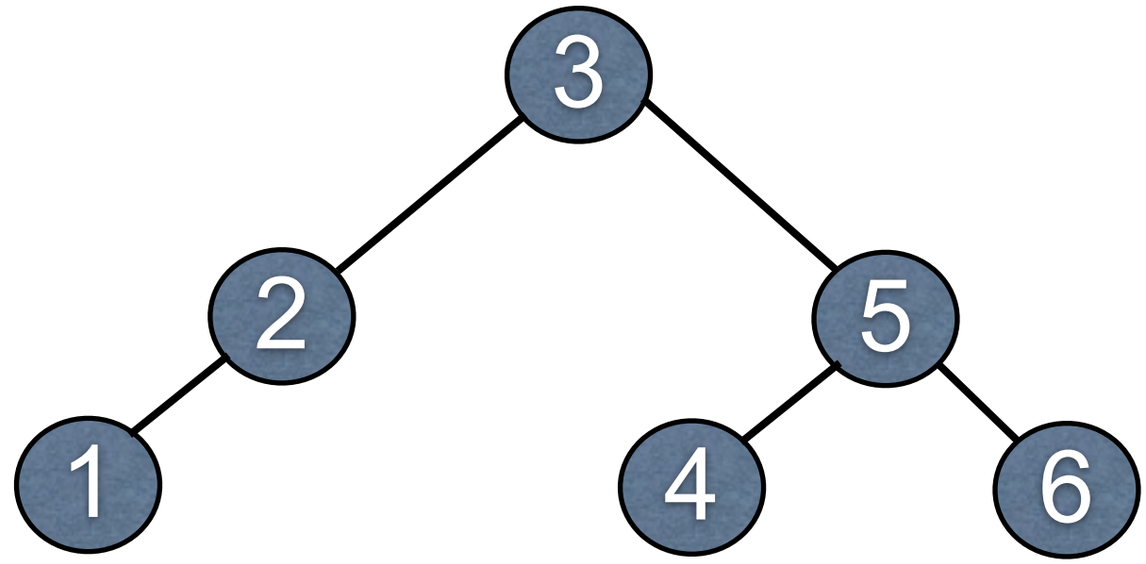
[1] 279-282

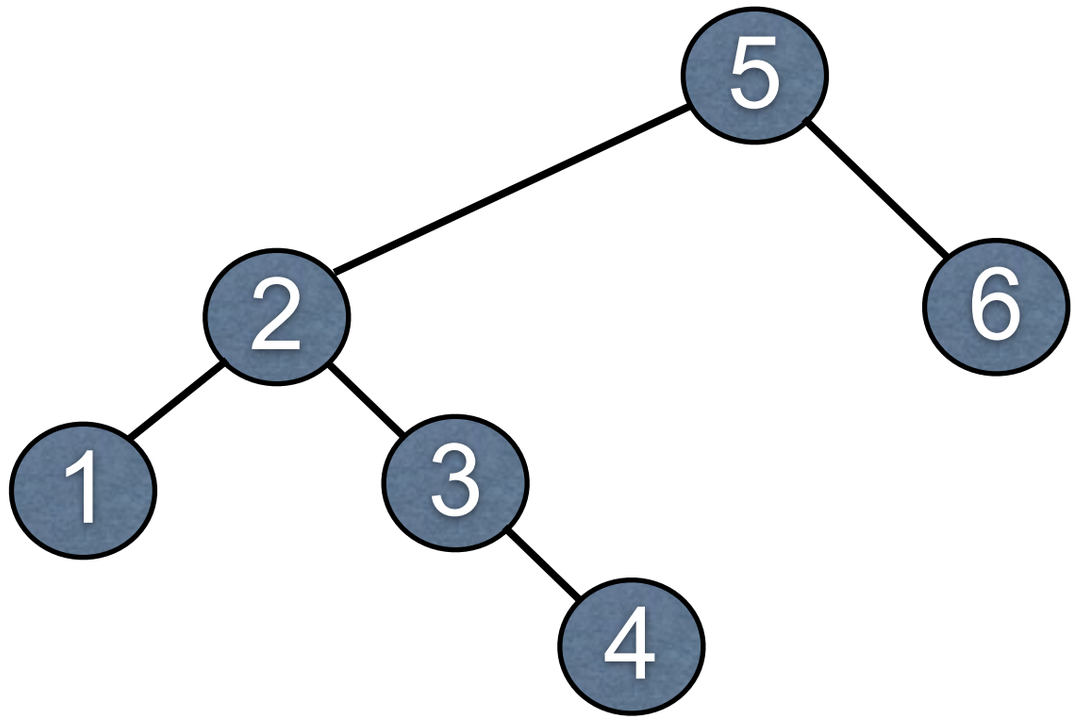
Példa

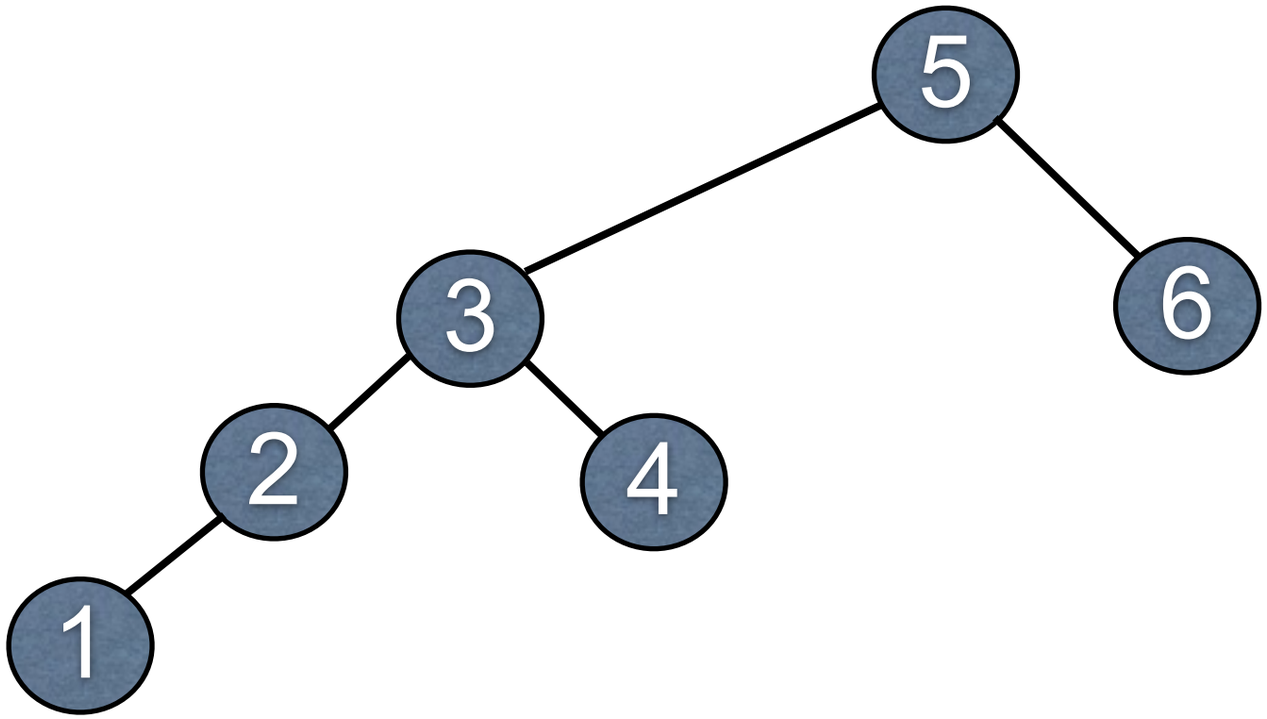


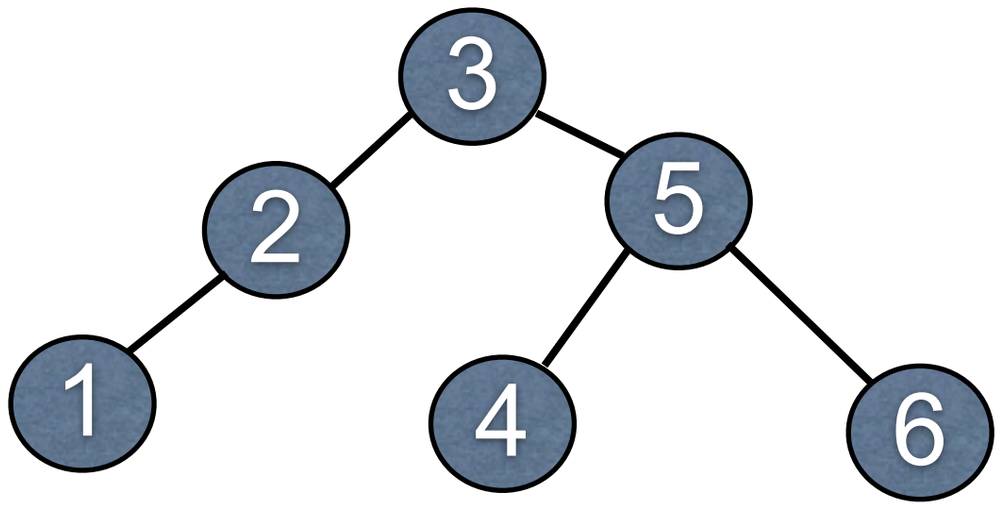
Egy érdekes eset



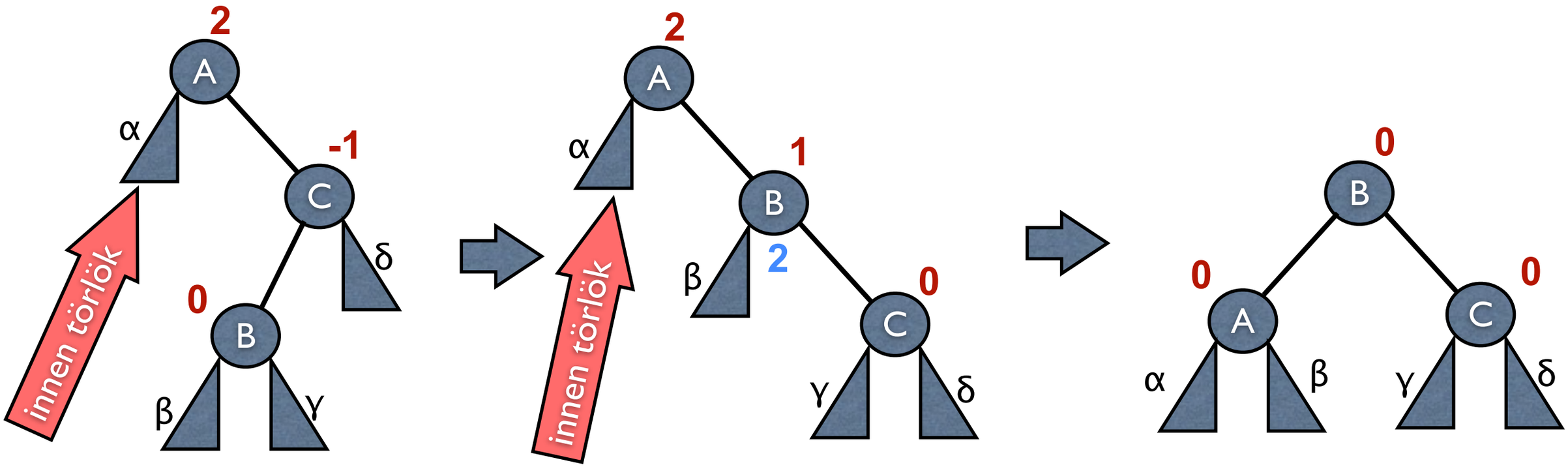



5
2
6
1
3
4


5
3
6
2
4
1

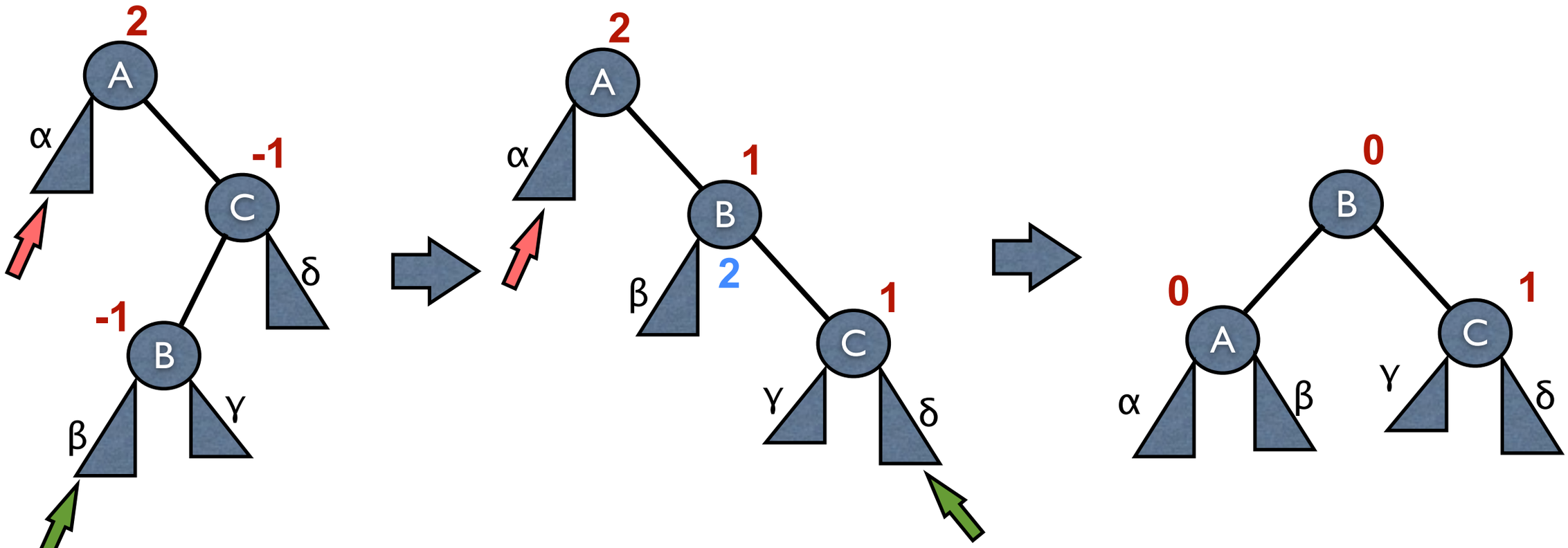

3
2
5
1
4
6

1. eset



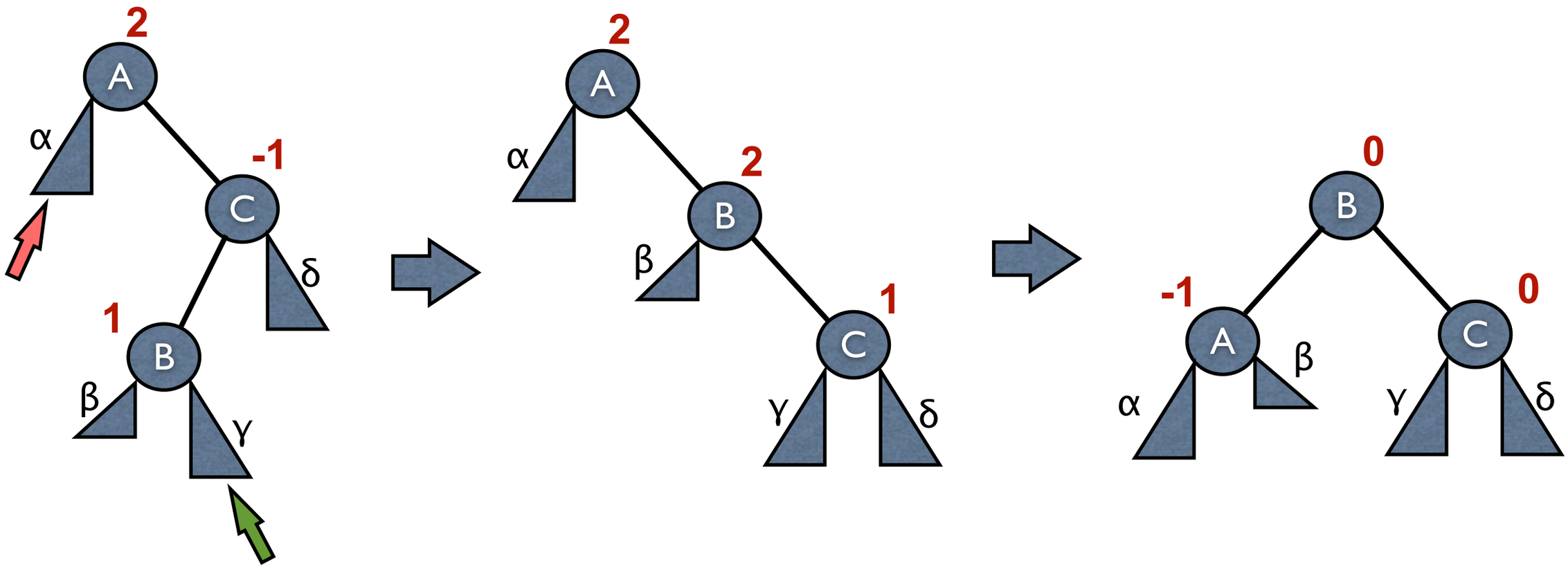
$$\alpha=\beta=\gamma=\delta$$

2. eset



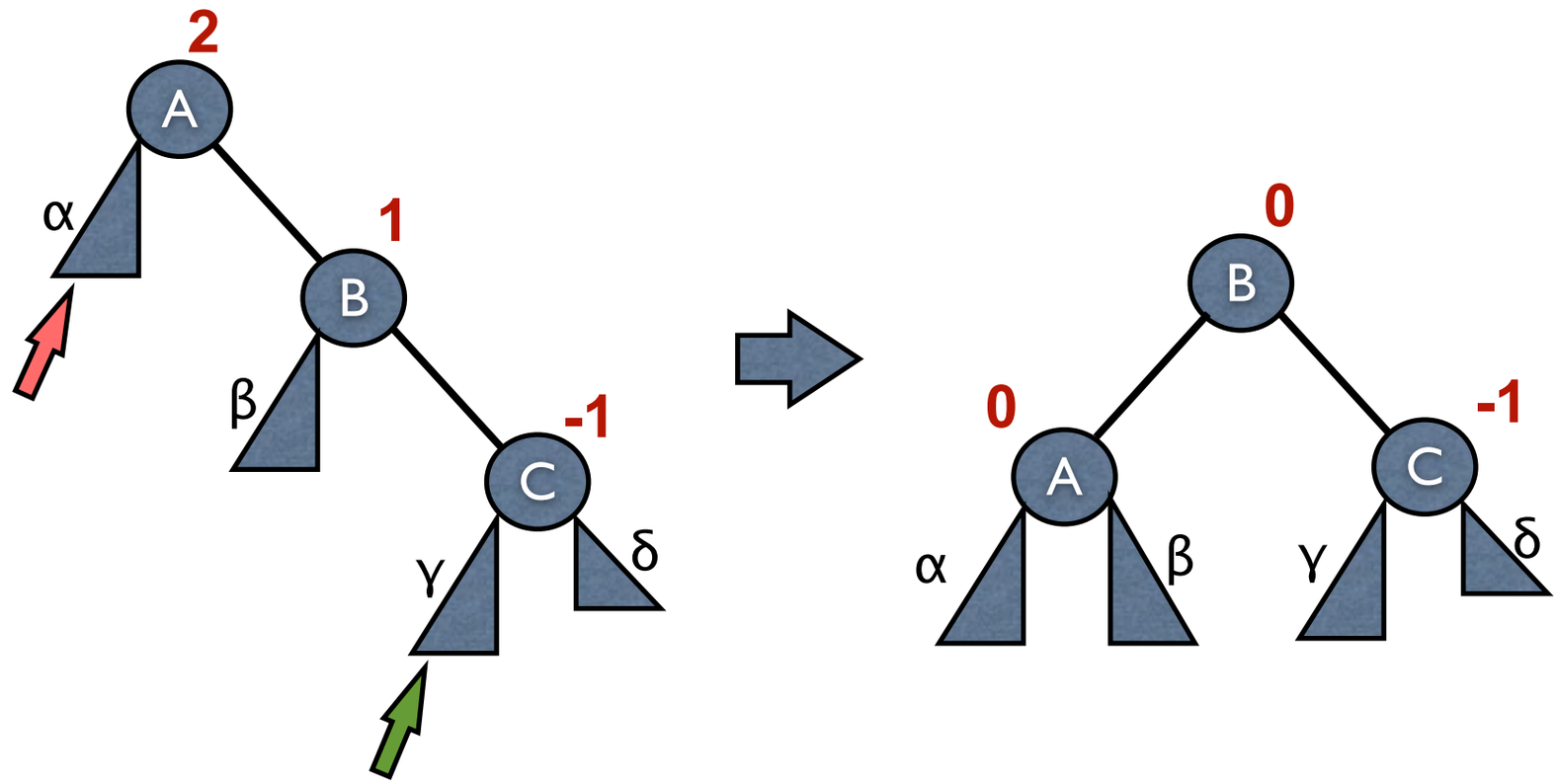
$$\alpha=\beta=(\gamma+1)=\delta$$

3. eset



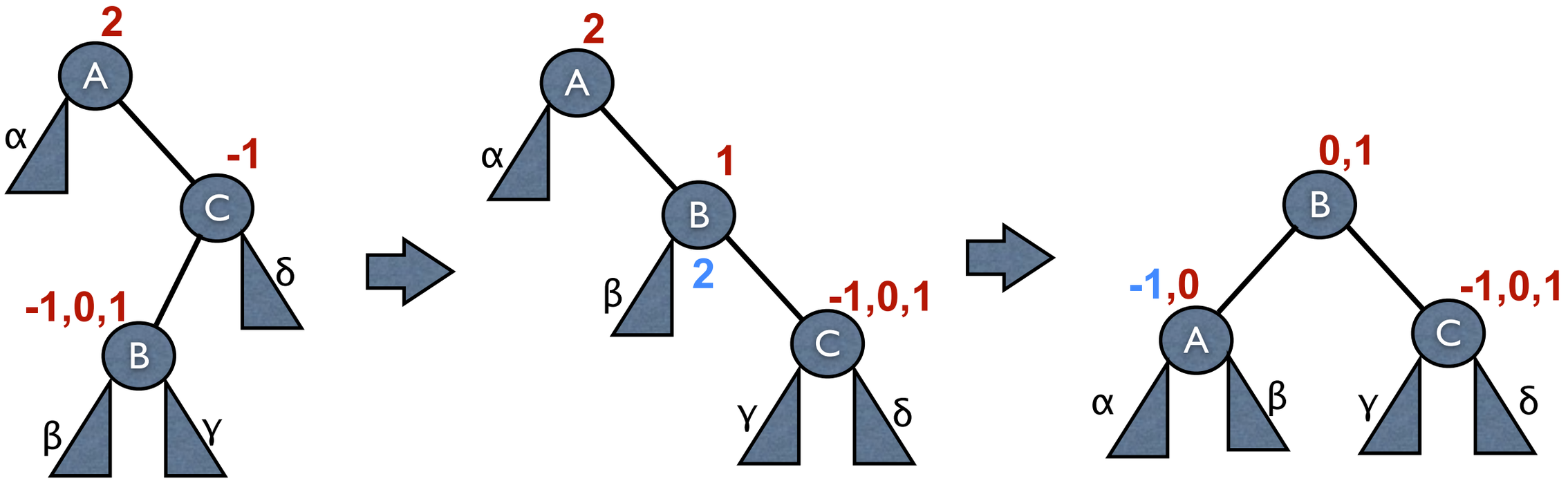
$\alpha = (\beta + 1) = \gamma = \delta$

4. eset



$$\alpha=\beta=\gamma=(\delta+1)$$

Összes eset



$\alpha=\beta=\gamma=\delta \longrightarrow \alpha=\beta=\gamma=\delta$

$\alpha=\beta=(\gamma+1)=\delta \longrightarrow \alpha=\beta=(\gamma+1)=\delta$

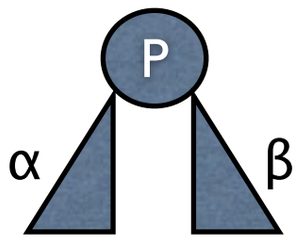
$\alpha=\beta=\gamma=(\delta+1)$

fj(p,q)

$\alpha=(\beta+1)=\gamma=\delta \longrightarrow \alpha=(\beta-1)=\gamma=\delta$

És ezek tükörképei

Egyensúlyfaktorok javítása beszúrás

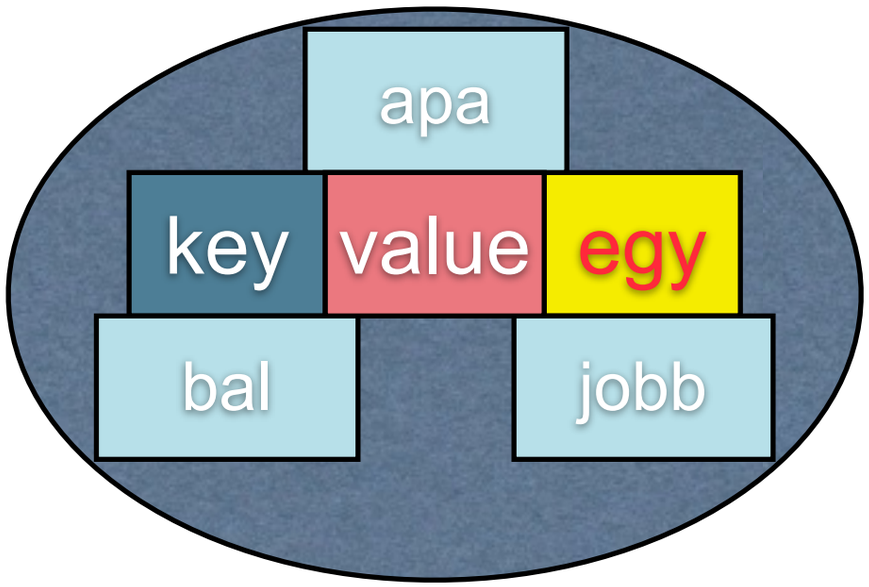


		α=β	α<β	α>β	
	x	**0**	**1**	**-1**	beszúrás előtt
α	**bal: eb(x)**	**-1**	**0**	**-2**	beszúrás után
β	**jobb: ej(x)**	**1**	**2**	**0**	beszúrás után

Fapont tárolása

második megközelítés, a gyakorlatban nem így tároljuk!

```
struct fapont{
    int key, v, egy ;
    fapont bal, jobb, apa ;
}
```



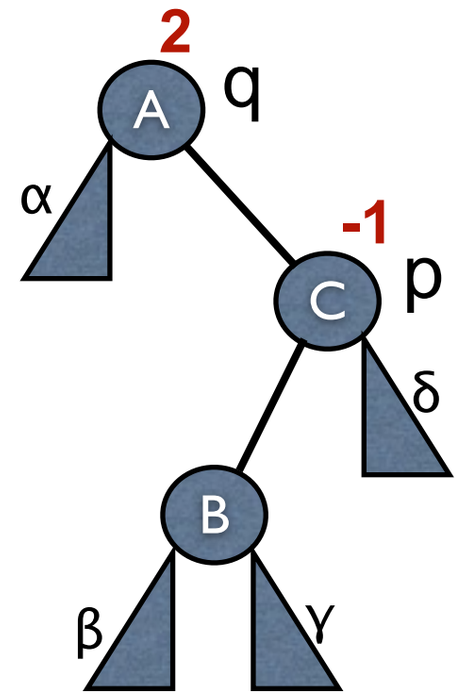
Egyensúlyfaktorok javítása

kezdetben p a beszúrt vagy törölt elem

```
while (p!=gyoker && cv) {
   q=p.apa ;
   if (p==q.jobb) {
      egyq=++q.egy ;
      if (egyq==2) {
         if (p.egy==-1) {
            jobbraforgat(p) ;
         }
         balraforgat(q) ;
         fj(p,q) ;        //egyensúlyfaktorokat rendbetesz
         cv=false ;       //kilép
      }
   } else ...             // bal eset
   p=p.apa ;
}
```

x	0	1	-1
eb(x)	-1	0	-2
ej(x)	1	2	0

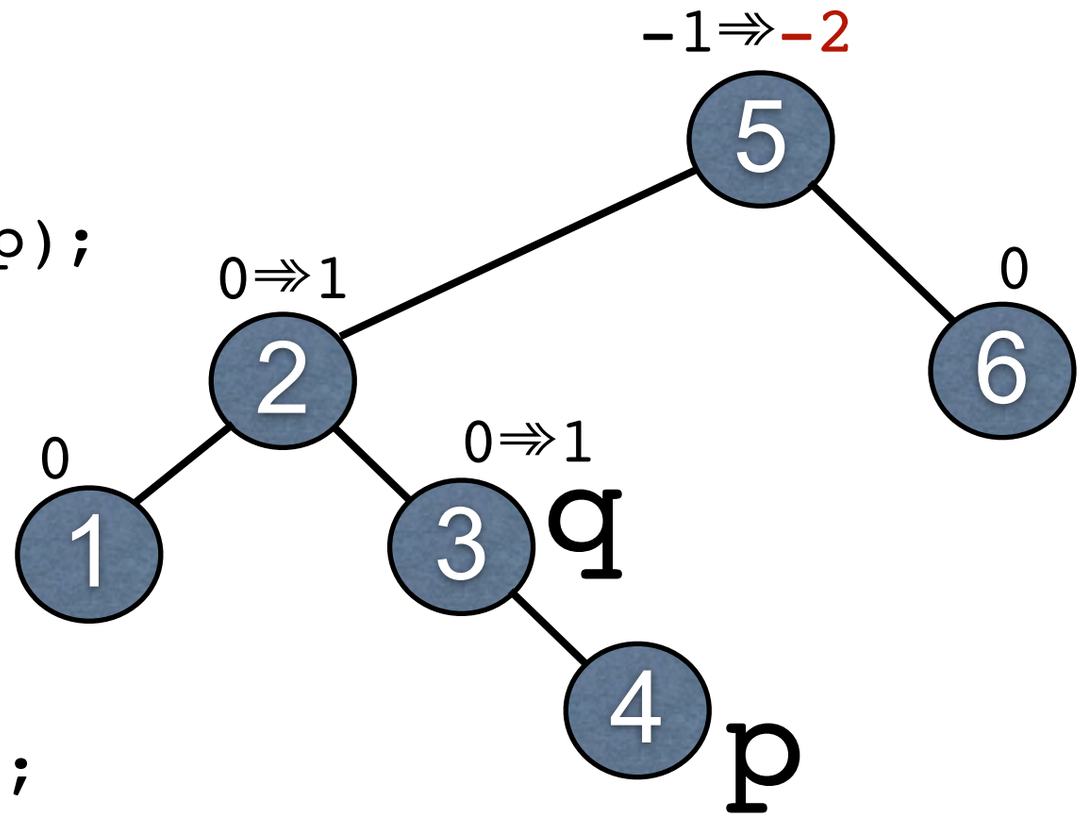
törlésnél (testver(p)==-1)

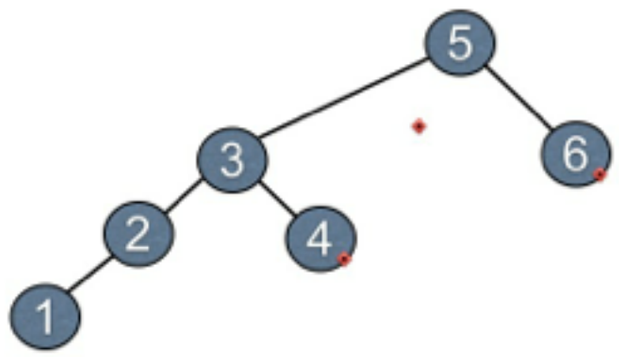


$$O(log\mathbf{n})$$

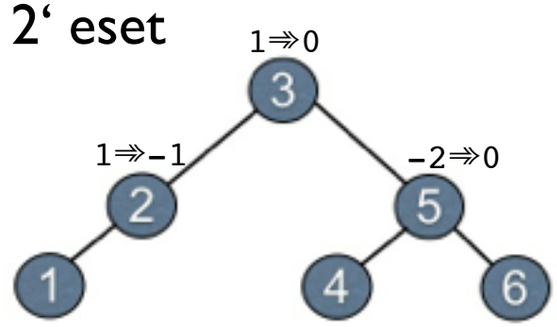
Példa

```
while (p!=gyoker && cv) {
⇛  q=p.apa ;
   if (p==q.jobb) {
      egyq=++q.egy ;
      if (egyq==2) {
         if (p.egy==-1) jobbraforgat(p);
         balraforgat(q);
         cv=false;  fj(p,q) ;
      }
   } else {
      egyq=--q.egy;
      if (egyq==-2) {
         if (p.egy==1)  balraforgat(p);
         jobbraforgat(q);
         cv=false;
         fj(p,q) ;
      }
   }
   p=p.apa;
}
```

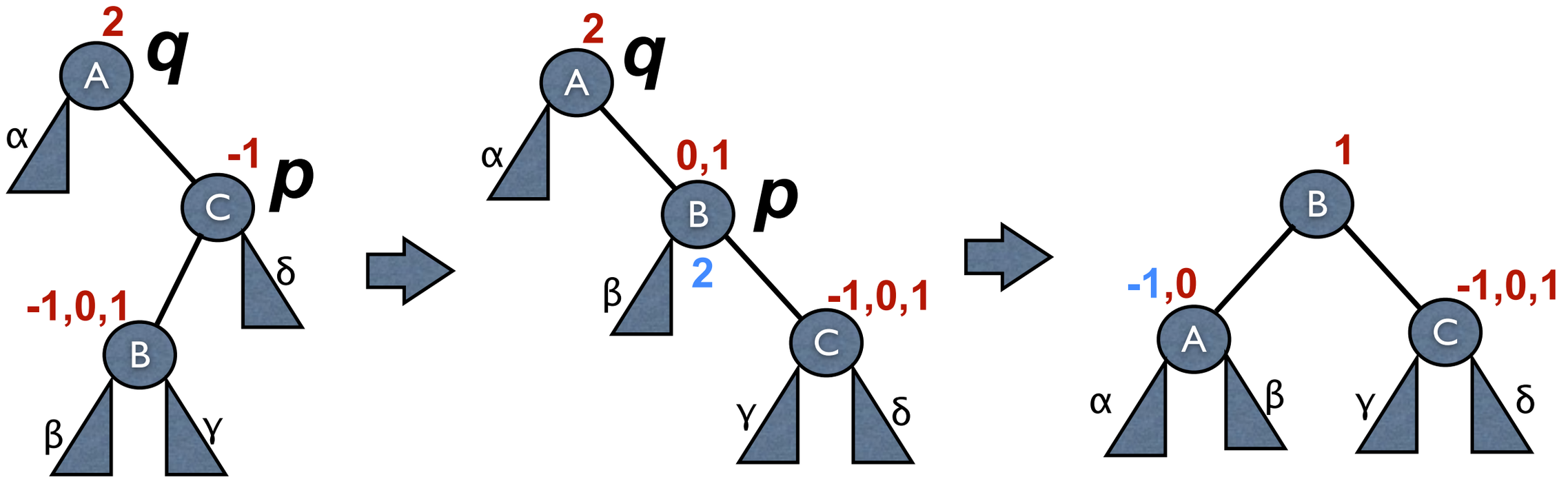




2‘ eset



fj(p,q)



$\alpha=\beta=\gamma=\delta \longrightarrow \alpha=\beta=\gamma=\delta$

$\alpha=\beta=(\gamma+1)=\delta \longrightarrow \alpha=\beta=(\gamma+1)=\delta$

$\alpha=\beta=\gamma=(\delta+1)$

$\alpha=(\beta+1)=\gamma=\delta \longrightarrow \alpha=(\beta-1)=\gamma=\delta$

És ezek tükörképei