# Bináris keresési fák, keresés

5. előadás

# Bináris keresőfák

# Rendezési (kereső) fák

- A rendezési fa (vagy keresőfa) olyan bináris fa adatszerkezet, amelynek kialakítása a különböző adatelemek között meglévő rendezési relációt követi
- A fa felépítése olyan, hogy minden csúcsra igaz az, hogy
  - a csúcs értéke nagyobb, mint tetszőleges csúcsé a tőle balra lévő leszálló ágon és
  - a csúcs értéke kisebb minden, a tőle jobbra lévő leszálló ágon található csúcs értékénél
- A T fa bármely $x$ csúcsára és $\mathrm{bal}(x)$ bármely $y$ csúcsára és $\mathrm{jobb}(x)$ bármely $z$ csúcsára:

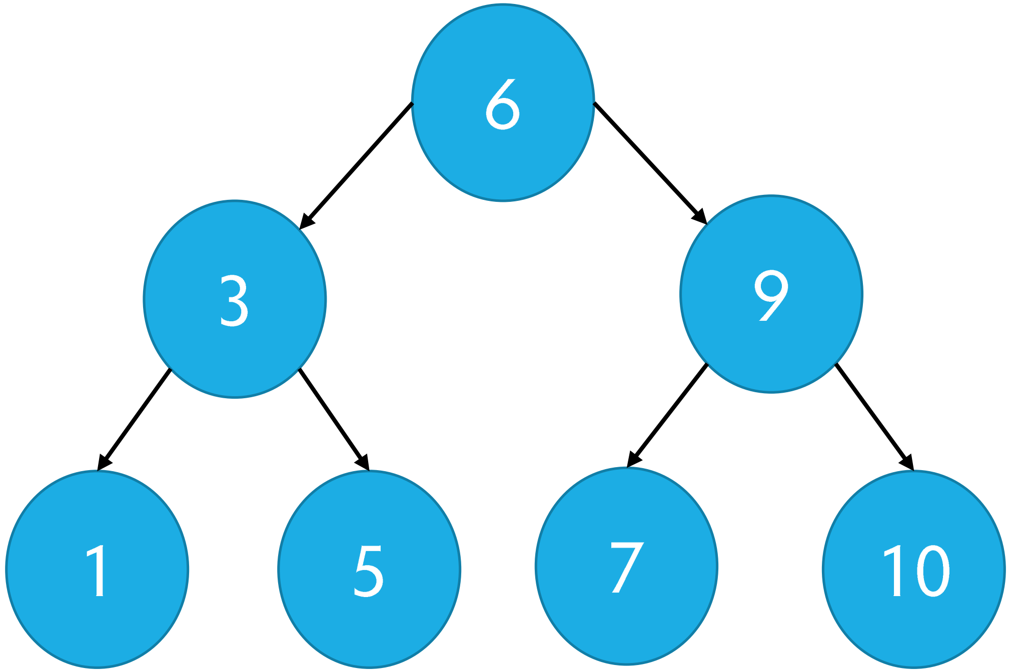
$$y < x < z$$

# Rendezési (kereső) fák

- A rendezési fa az őt tartalmazó elemek beviteli sorrendjét is visszatükrözi.
- Ugyanazokból az elemekből különböző rendezési fák építhetők fel.

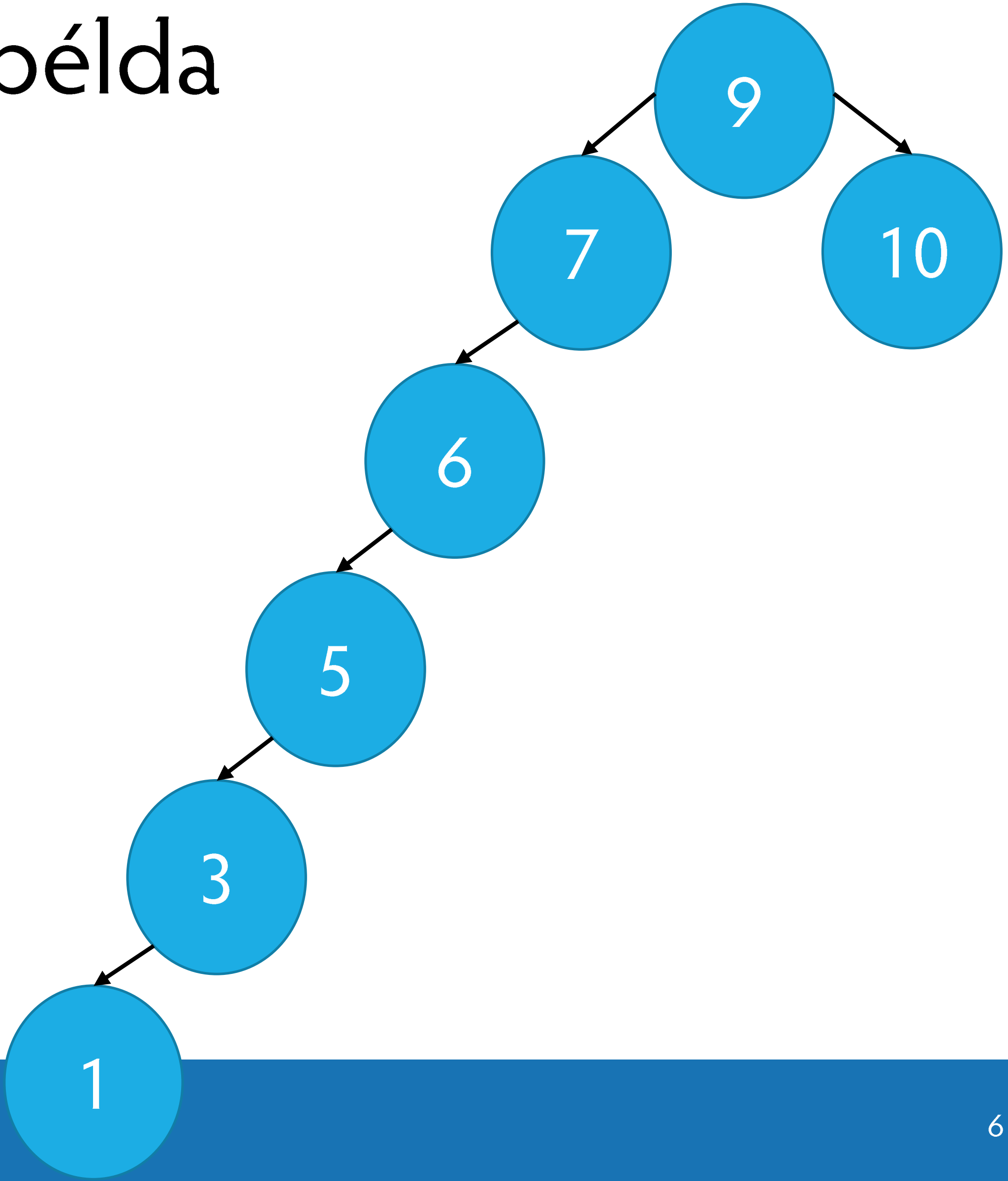
# Elemek beszúrása példa

- 6,3,1,9,7,5,10



# Elemek beszúrása példa

- 9,7,6,5,10,3,1



# Rendezési (kereső) fák

- Fontos tulajdonság
  - inorder bejárással a kulcsok rendezett sorozatát kapjuk
- Az algoritmus pszeudokódja:

```
Inorder-fa-bejárás(x)
if x≠NIL
  then Inorder-fa-bejárás(bal[x])
    print(kulcs[x])
  Inorder-fa-bejárás(jobb[x])
```

- Egy T bináris keresőfa összes értékének kiíratásához

```
Inorder-fa-bejárás(gyökér[T])
```

# Rendezési (kereső) fák

- Az algoritmus helyessége a bináris-kereső-fa tulajdonságból indukcióval adódik.

- Egy $n$ csúcsú bináris kereső fa bejárása $O(n)$ ideig tart
  - A kezdőhívás után a fa minden csúcspontja esetében pontosan kétszer (rekurzívan) meghívja önmagát
    - egyszer a baloldali részfára
    - egyszer a jobboldali részfára

# Műveletek

- Keresés
  - A T fában keressük a k kulcsú elemet (csúcsot)
  - ha ez létezik, akkor visszaadja az elem címét, egyébként NIL-t.
- Az algoritmust megadjuk rekurzív és iteratív megoldásban is, ez utóbbi a legtöbb számítógépen hatékonyabb.

# Műveletek

- Keresés
  - A rekurzív algoritmus pszeudokódja

```
Fában-keres(x, k)
    if x = NIL or k = kulcs[x]
      then return x
    if k < kulcs[x]
      then return Fában-keres(bal[x], k)
      else return Fában-keres(jobb[x], k)
```

# Műveletek

- Keresés:
  - Az iteratív algoritmus pszeudokódja

```
Fában-iteratívan-keres(x, k)
    while x ≠ NIL and k  ≠ kulcs[x] do
      if k < kulcs[x]
        then x ← bal[x]
        else x ← jobb[x]
    return x
```

# Műveletek

- Minimum keresés
  - Tegyük fel, hogy T ≠ NIL. Addig követjük a baloldali mutatókat, amíg NIL mutatót nem találunk
  - Az iteratív algoritmus pszeudokódja:

```
Fában-minimum (T)
  x ← gyökér[T]
  while bal[x] ≠ NIL
    do  x ← bal[x]
  return x
```

  - Helyessége a bináris-kereső-fa tulajdonságból következik
  - Lefut $\mathcal{O}(h)$ idő alatt, ahol $h$ a fa magassága

# Műveletek

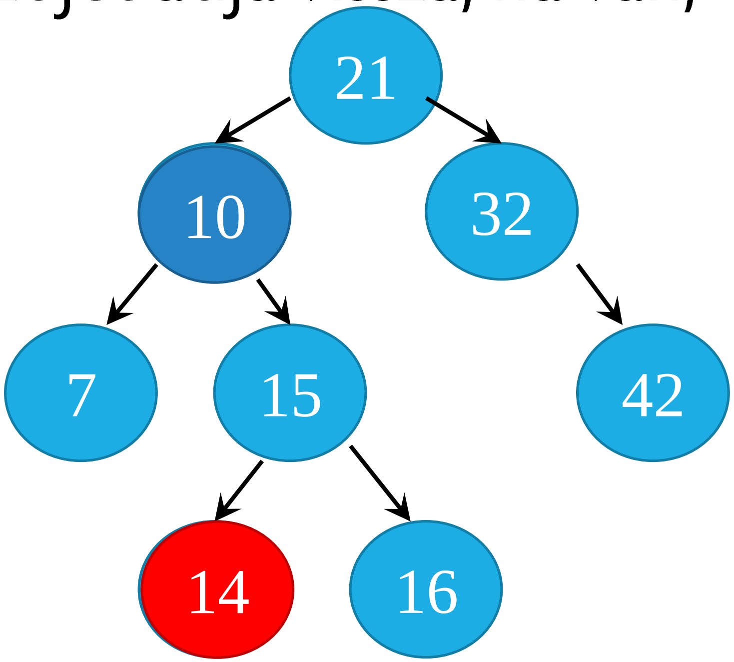
- Maximum keresés
  - Tegyük fel, hogy T ≠ NIL. Addig követjük a jobboldali mutatókat, amíg NIL mutatót nem találunk
  - Az iteratív algoritmus pszeudokódja:

```
Fában-maximum (T)
  x ← gyökér[T]
  while jobb[x] ≠ NIL
    do  x ← jobb[x]
  return x
```

  - Helyessége a bináris-kereső-fa tulajdonságból következik
  - Lefut $\mathcal{O}(h)$ idő alatt, ahol $h$ a fa magassága

# Műveletek

- Következő elem: x csúcs rákövetkezőjét adja vissza, ha van, NIL különben
  - Több eset lehetséges
  - Például a 10 rákövetkezője a 14
    - Létezik a megfelelő jobb részfa



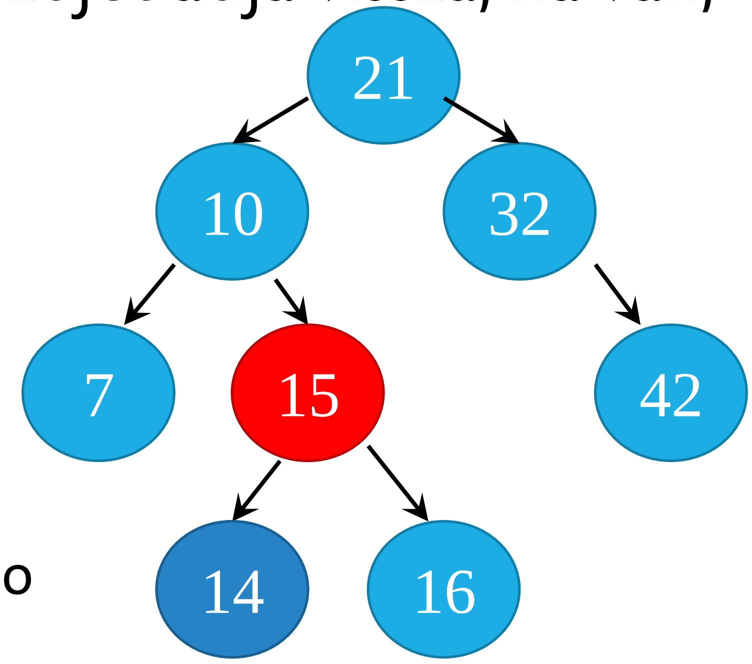

  - Algoritmusa

```
if jobb[x] ≠ NIL
   then return Fában-minimum (jobb[x])
```

# Műveletek

- Következő elem: x csúcs rákövetkezőjét adja vissza, ha van, NIL különben
  - A 14 rákövetkezője a 15
    - Nem létezik a jobb részfa
      - Felfelé kell keresni
  - Algoritmusa

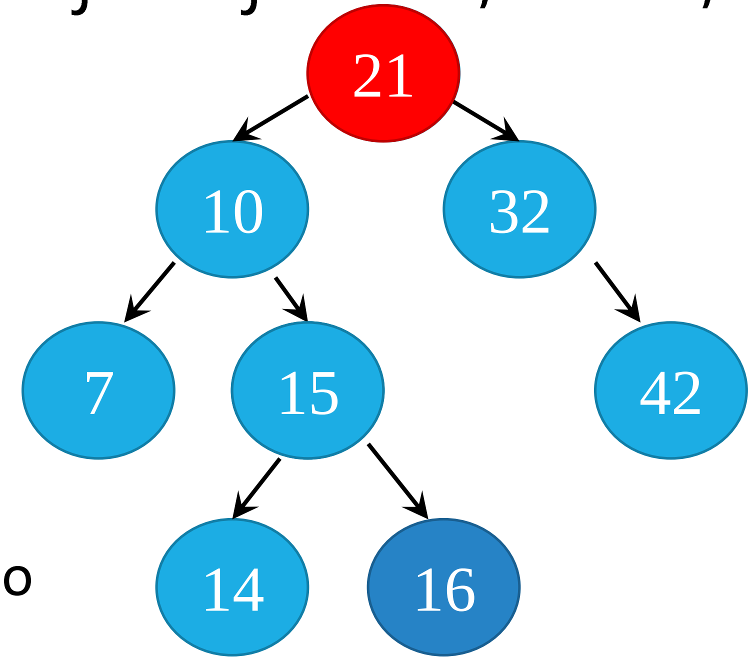
```
y ← szülő[x]
while y ≠ NIL és x = jobb[y] do
   x ← y
   y ← szülő[x]
return y
```



# Műveletek

- Következő elem: x csúcs rákövetkezőjét adja vissza, ha van, NIL különben
  - A 16 rákövetkezője a 21
    - Nem biztos, hogy mindig a gyökér!
  - Algoritmusa

```
y ← szülő[x]
while y ≠ NIL és x = jobb[y] do
  x ← y
  y ← szülő[x]
return y
```



# Műveletek

- Következő elem: x csúcs rákövetkezőjét adja vissza, ha van, NIL különben
- Teljes algoritmus

```
Fában-következő(T, x)
if jobb[x] ≠ NIL
   then return Fában-minimum (jobb[x])
y ← szülő[x]
while y ≠ NIL és x = jobb[y] do
  x ← y
  y ← szülő[x]
return y
```

# Műveletek

- Fában-következő(T, x) futási ideje $h$ magasságú fák esetén $\mathcal{O}(h)$.
- Megelőző elem: $x$ csúcs megelőzőjét adja vissza, ha van, NIL különben.
  - `Fában-megelőző(T, x)`
  - Házi feladat

# Műveletek

- Tétel: A dinamikus halmazokra vonatkozóKeres, Minimum, Maximum, Következő és Előző műveletek $h$ magasságú bináris keresőfában $O(h)$ idő alatt végezhetők el
  - Bizonyítás: az előzőekből következik

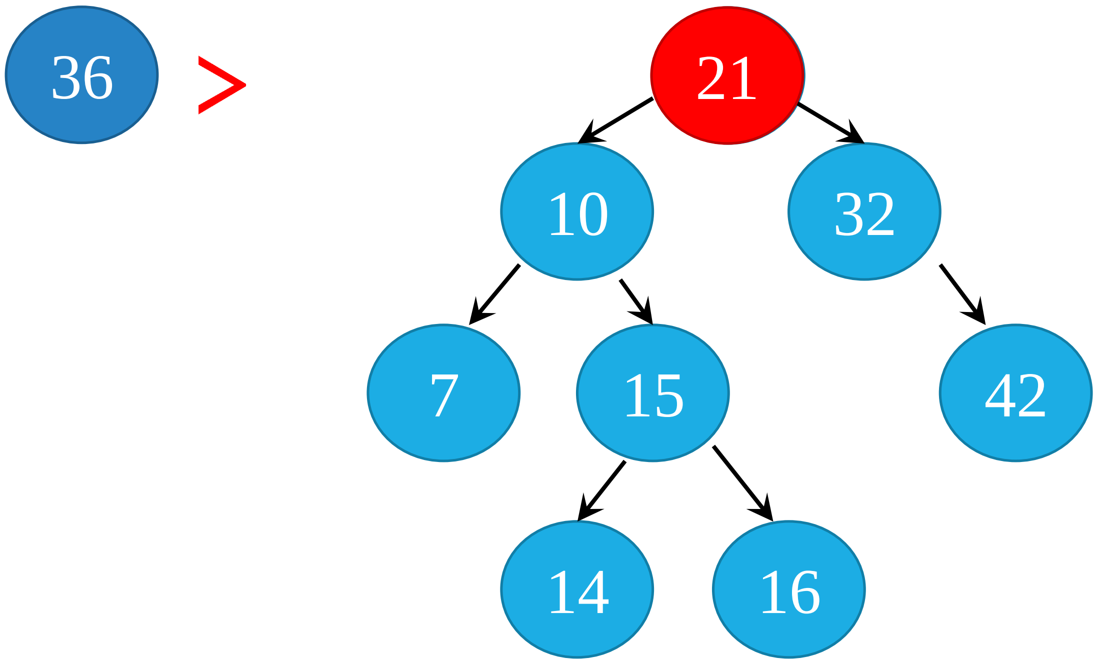
# Műveletek

- Beszúrás
  - A T bináris keresőfába a p csúcsot szúrjuk be.
  - Kezdetben:
    - `kulcs[p]=k`
    - `bal[p] = NIL`
    - `jobb[p] = NIL`
    - `szülő[p] = NIL`

- Feltételezzük, hogy a fában még nincs k kulcsú csúcs!
  - Otthoni feladat megnézni, hogyan változik az algoritmus, ha ez a feltételezés nem igaz

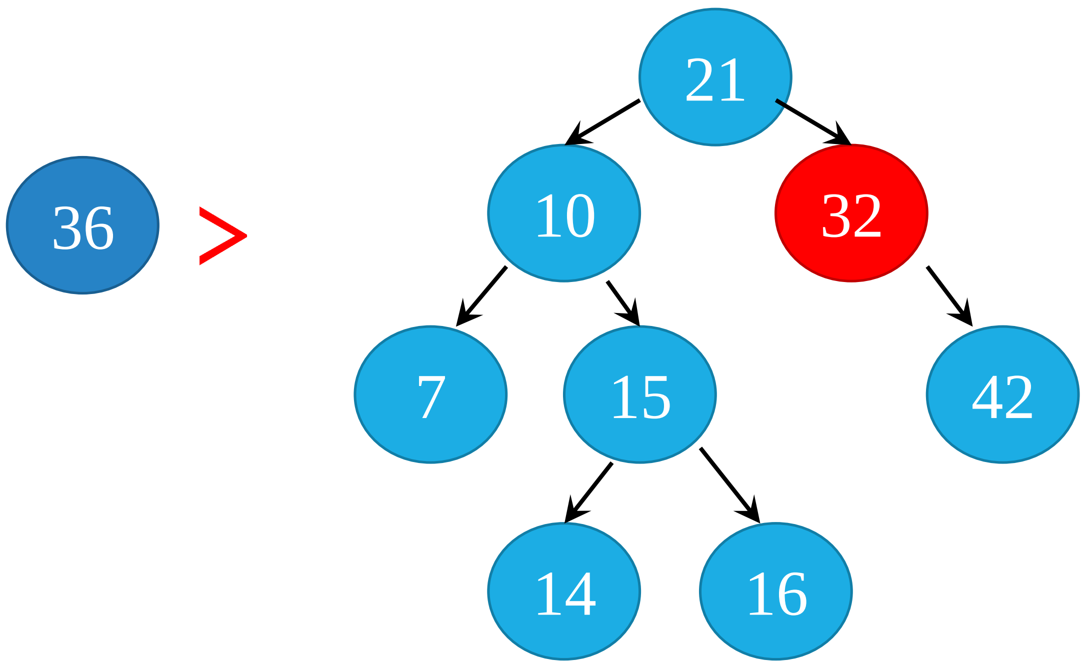
# Műveletek

- Fába beszúr: szúrjuk be például a 36-t!
  - 1. megkeressük a helyét



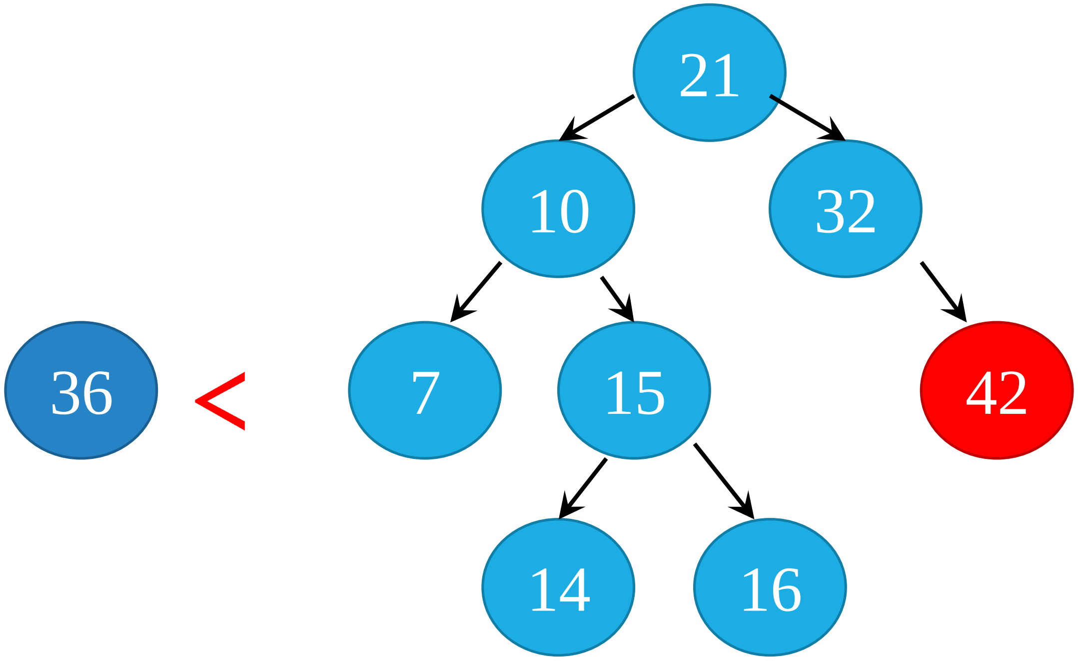
# Műveletek

- Fába beszúr: szúrjuk be például a 36-t!
  - 1. megkeressük a helyét



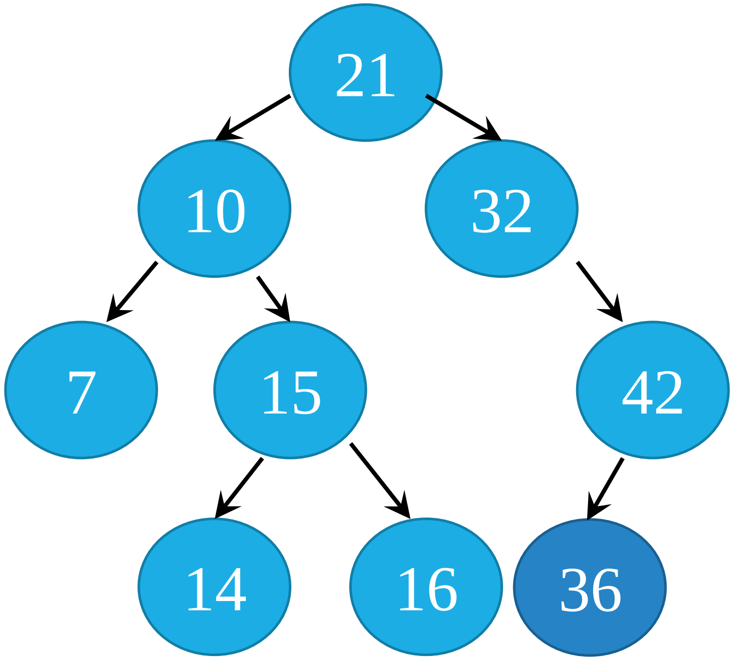
# Műveletek

- Fába beszúr: szúrjuk be például a 36-t!
  - 1. megkeressük a helyét



# Műveletek

- Fába beszúr: szúrjuk be például a 36-t!
  - 1. megkeressük a helyét
  - 2. beláncoljuk



# Műveletek

- Algoritmus

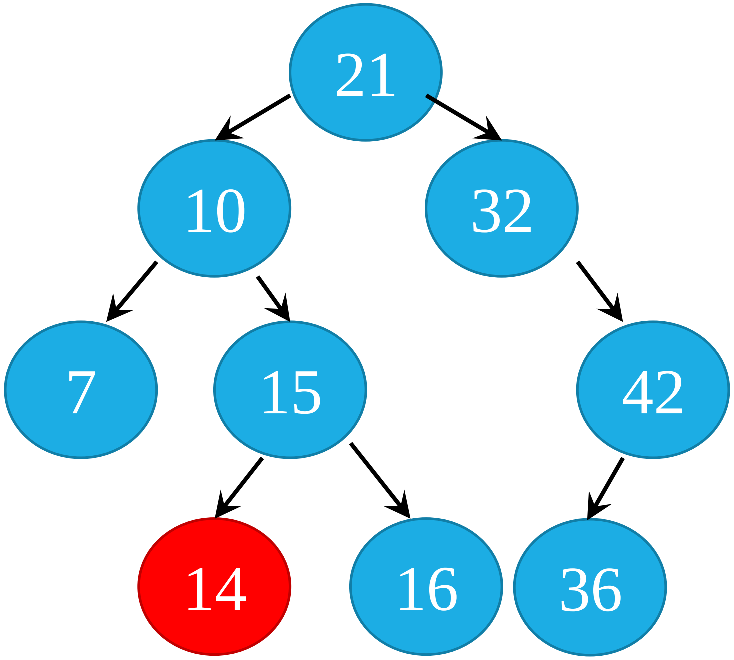
```
Fába-beszúr (T,p)
y ←NIL; x ← gyökér[T]
while x  ≠ NIL
  do y ← x
    if kulcs[p] < kulcs[x]
      then x ← bal[x]
      else x ← jobb[x]
szülő[p] ← y
if y = NIL
  then gyökér[T] ← p
  else if kulcs[p] < kulcs[y]
    then bal[y] ←p
    else jobb[y] ←p
```

# Műveletek

- Törlés:
  - A T bináris keresőfából a p csúcsot töröljük
- Lehetőségek:
  1. p-nek még nincs gyereke: szülőjének mutatóját NIL-re állítjuk
  2. p-nek egy gyereke van: a szülője és a gyermeke között építünk ki kapcsolatot
  3. p-nek két gyereke van: átszervezzük a fát: kivágjuk azt a legközelebbi rákövetkezőjét, aminek nincs balgyereke így 1., vagy 2. típusú törlés, majd ennek tartalmát beírjuk p-be

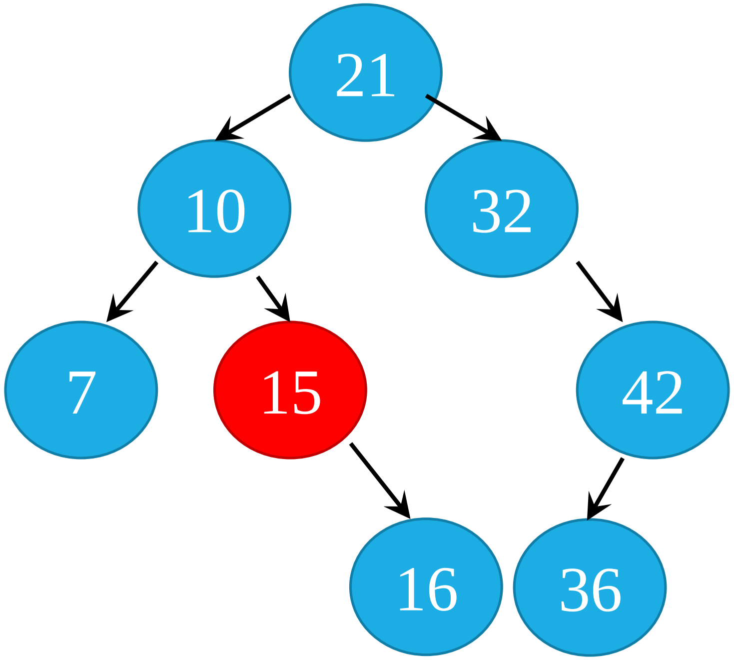
# Műveletek

- Törlés: töröljük ki például a 14-t!
  - Ez a legegyszerűbb eset, alkalmazzuk az 1. szabály szerinti teendőket



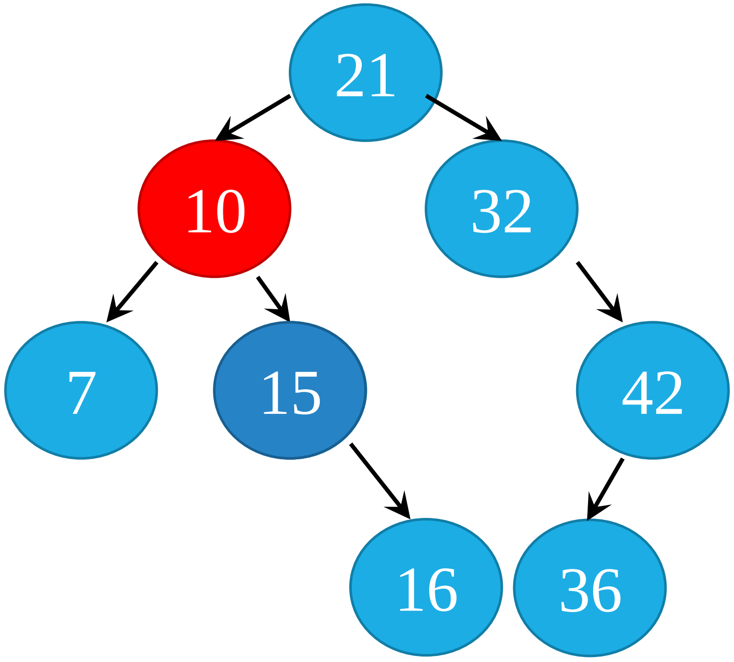
# Műveletek

- Törlés: töröljük ki például a 15-t!
  - Ebben az esetben egy gyereke van a törlendőnek, tehát a 2. szabályt alkalmazzuk



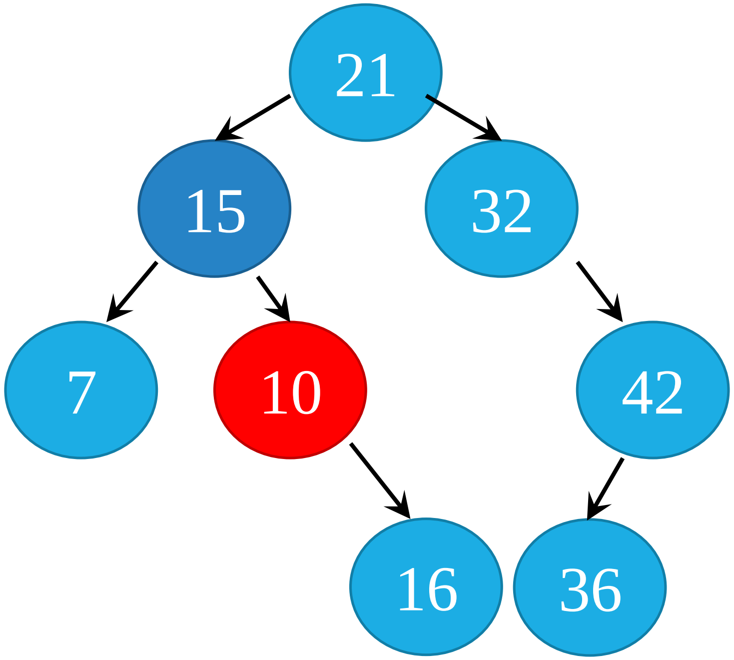
# Műveletek

- Törlés: töröljük ki például a 10-et!
  - Ebben az esetben két gyereke van a törlendőnek, tehát a 3. szabályt alkalmazzuk
    - A megfelelő, rákövetkező elem a 15.
    - Ennek nincsen balgyereke
      - Ha lenne nem az lenne a rákövetkező
    - Előfordulhat, hogy jobbgyereke sincs



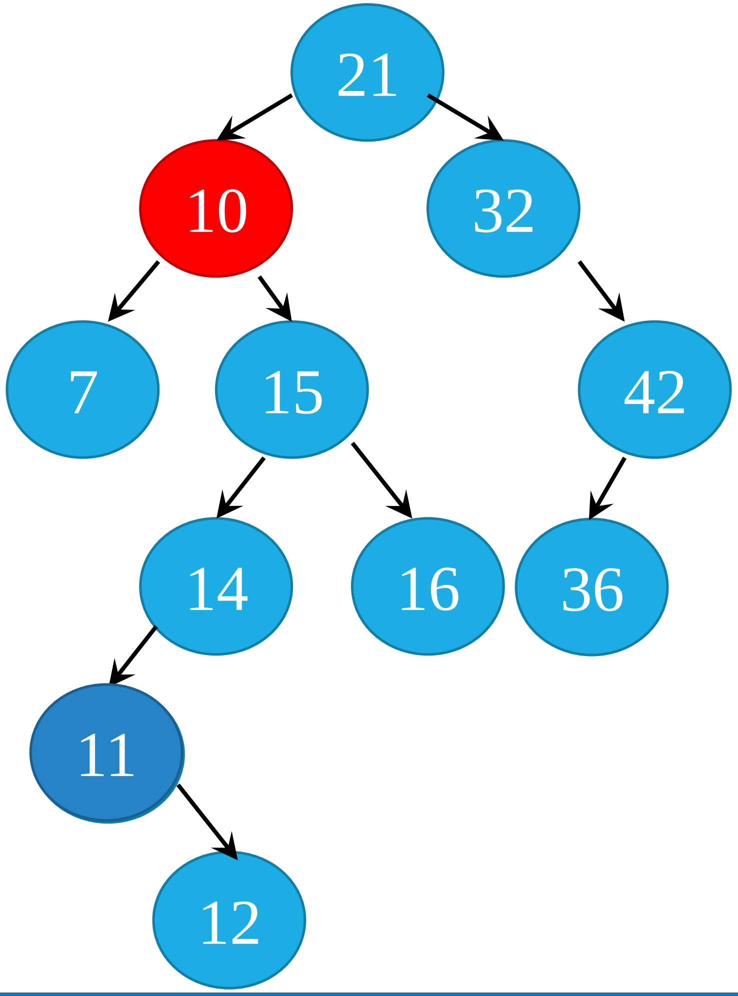
# Műveletek

- Törlés: töröljük ki például a 10-et!
  - Ebben az esetben két gyereke van a törlendőnek, tehát a 3. szabályt alkalmazzuk
    - A megfelelő, rákövetkező elem a 15.
    - Ennek nincsen balgyereke
      - Ha lenne nem az lenne a rákövetkező
    - Előfordulhat, hogy jobbgyereke sincs
  - Helyet cserél a törlendő és a rákövetkező
    - Majd végrehajtjuk a törlést az eddigiek szerint



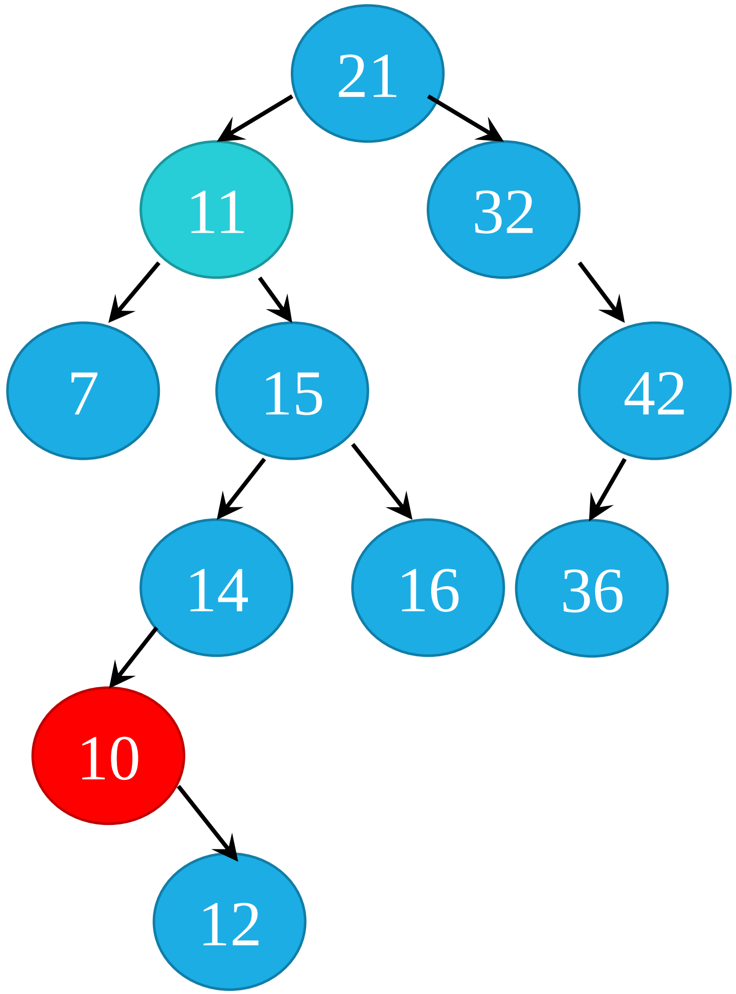
# Műveletek

- Törlés: töröljük ki például a 10-et!
  - Ugyanez komplikáltabb példán



# Műveletek

- Törlés: töröljük ki például a 10-et!
  - Ugyanez komplikáltabb példán



# Műveletek

- Feltesszük, hogy p a T-ben létezik
- Fából-töröl (T,p)

```
if bal[p] = NIL vagy jobb[p] = NIL
  then y ←p                                -- 0, vagy 1 gyerek
  else y ← Fában-következő(T, p)           -- 2 gyerek
if bal[y]  ≠ NIL
  then x ← bal[y]                          -- x az y 0, vagy 1
  else x ← jobb[y]                            gyerekére mutat
if x  ≠ NIL                                -- ha volt gyereke, akkor
  then szülő[x] ← szülő[y]                    befűzi az új szülőhöz
…
```

# Műveletek

- Fából-töröl (T,p)

```
if szülő[y] =NIL
  then gyökér[T ] ←x
  else  if  y = bal[szülő[y]]
    then bal[szülő[y]] ← x
    else  jobb[szülő[y]] ← x
if  y  ≠ p
  then kulcs[p] ← kulcs[y]
return y
```

-- ha a gyökeret töröltük akkor be kell állítani az újat
-- különben a szülőnek megfelelő oldalhoz tartozó mutatót kell az x-re állítani
-- amennyiben a ténylegesen törlendő csúcs nem azonos azzal, amit kiláncolunk át kell írni az adatot is

# Keresések

# Keresések

- Sok adat esetén milyen adatszerkezetben lehet hatékonyan
  - keresni,
  - módosítani,
  - beszúrni és
  - törölni?
- A gyakorlat azt mutatja, hogy fákban és táblázatokban

# Keresések

- Szekvenciális keresések
  - a keresési idő n-nel arányos: $\mathcal{O}(n)$
  - rendezetlen tömbök
  - láncolt listák
- Bináris keresés
  - rendezett tömbök
  - a keresési idő $\log_2 n$-nel arányos: $\mathcal{O}(\log_2 n)$

# Keresések

- Rendezett tömb létrehozása
  - elem hozzáadása megfelelő helyre
    - megkeresem a pozícióját: $c_1 \log_2 n$
    - eltolom: $\boxed{c_2 n}$
    - összesen: $c_1 \log_2 n + c_2 n$
    - vagyis: $c_2 n$

domináns

  - Minden elem hozzáadása a rendezett tömbhöz: $\mathcal{O}(n)$
- Nem lehetne-e a rendezett tömböt olcsóbb beszúrásokkal karbantartani?

# Keresések

- Szótár (dictionary) egy adatszerkezet, ha értelmezve vannak a következő műveletek:
  - Keres
  - Beszúr
  - Töröl
  - (Tól-ig)

# Keresések

- Prioritásos sor egy adatszerkezet, ha az előzőeken kívül értelmezve vannak a következők is:
  - Minimum
  - Maximum
  - Előző
  - Rákövetkező
- Ezzel lehet rendezni

# Keresések

- Adatok a struktúrában
  - kulcs + mezők (rekordok)
- Lehetőségek
  - Kulcsegyezőség
    I. minden kulcs különböző
    II. lehetnek azonos kulcsok
  - Adatstruktúra
    A. rekordok: $(k, m_1, m_2, \ldots, m_n)$
    B. csak a kulcsokat nézzük: $k$
  - Mi most az I. és B.-t választjuk.

# Bináris keresőfák

- Mennyire hatékony a bináris keresőfa?
  - Minden művelet egy útvonal bejárását jelenti a fában
  - A $h$ magasságú fákban $\mathcal{O}(h)$ idő alatt hajtódnak végre
  - Ahogy elemeket beszúrunk és törlünk, változik a fa magassága és ezzel a műveletek ideje is
    - Itt ugyanis nem tudom a fa átlagos magasságát
    - Ha csak beszúrások vesszük a felépítés során, akkor könnyebben elemezhető

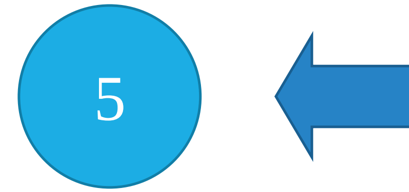
# Bináris keresőfák

- Legyen adva $n$ különböző kulcs, ebből bináris keresőfát építünk. Ha itt minden sorrend egyformán valószínű, akkor a kapott fát véletlen építésű bináris keresőfának nevezzük.
- Bizonyítható, hogy egy $n$ kulcsból **véletlen módon** épített bináris keresőfa átlagos magassága $O(\log_2 n)$.

# Bináris keresőfák

- Tegyük fel, hogy a véletlen sorrendű 1,2, … $n$ adatokból építjük fel a t keresőfát.
- Mennyi az átlagos csúcsmagasság?
  - Ennek megválaszolása megadja a következő kérdésre is a megoldást:
- Hány összehasonlítással lehet felépíteni a t keresőfát átlagosan?
- A meghatározáshoz vegyünk egy keresőfát

# Bináris keresőfák

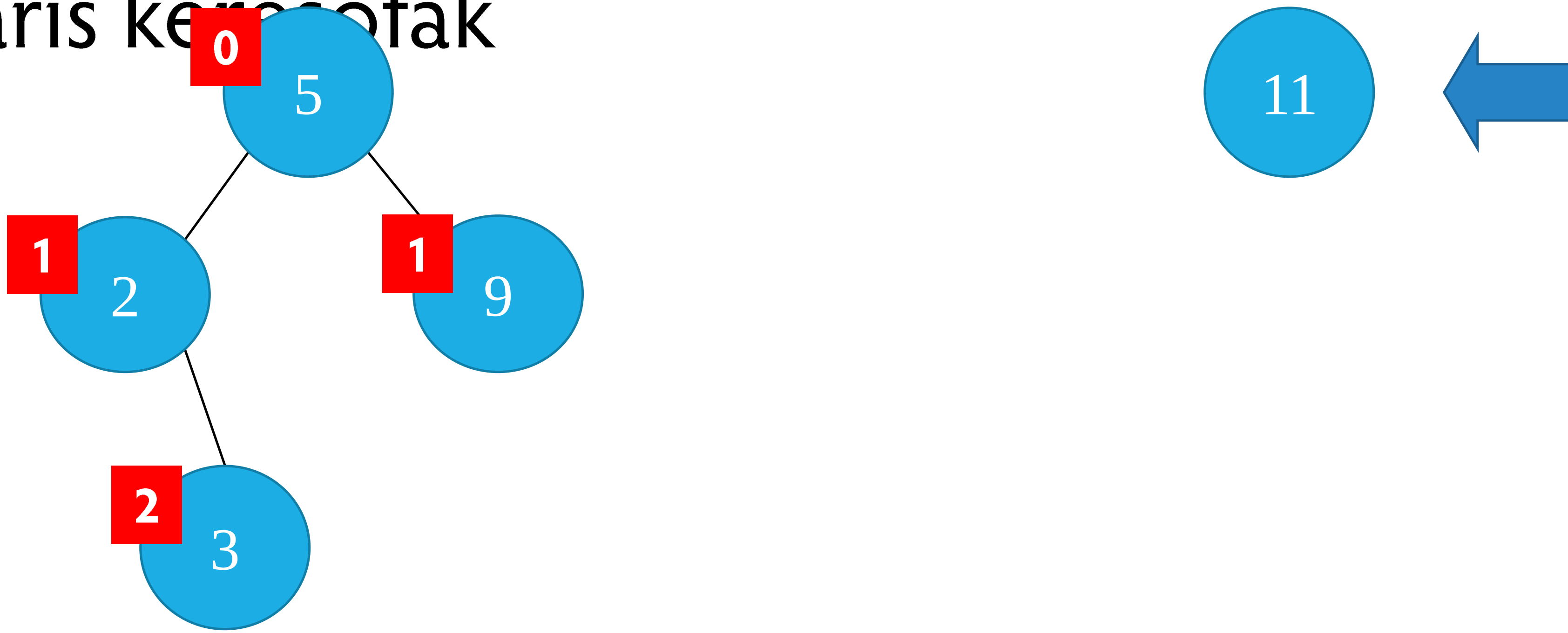


# Bináris keresőfák

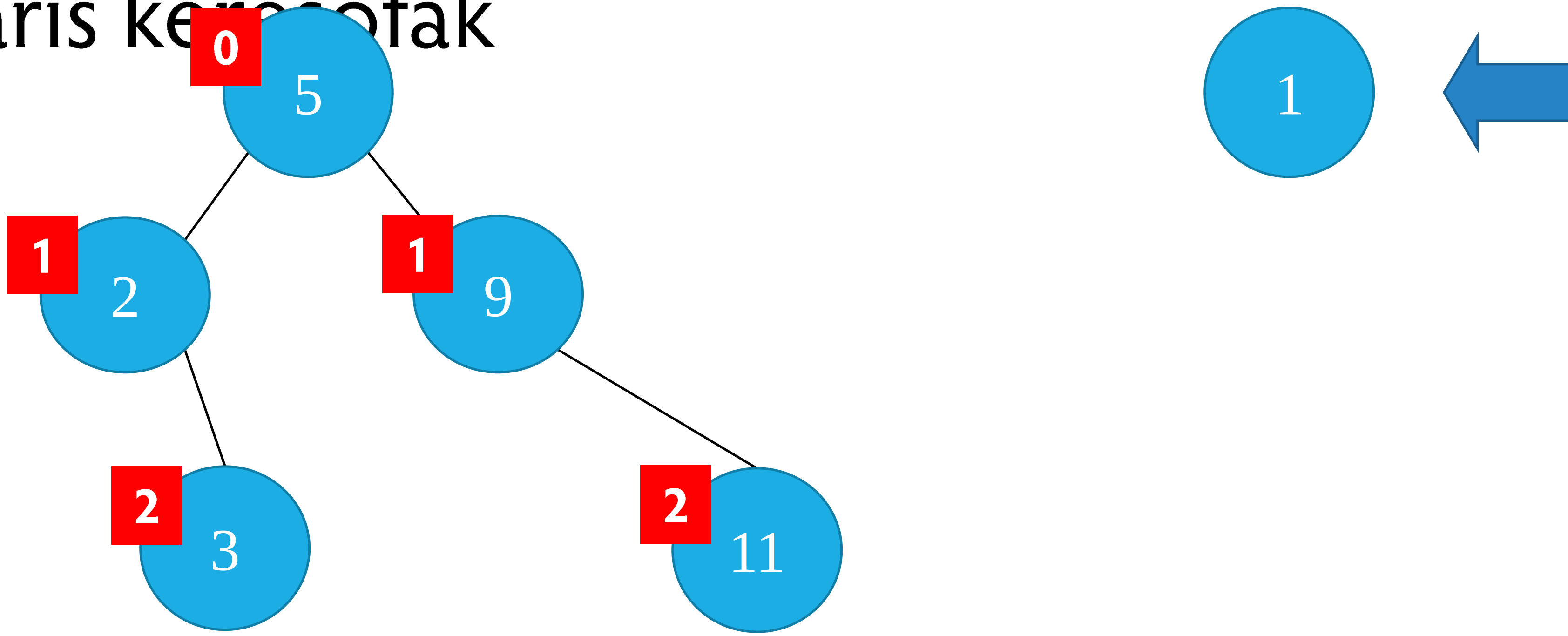
# Bináris keresőfák

# Bináris keresőfák

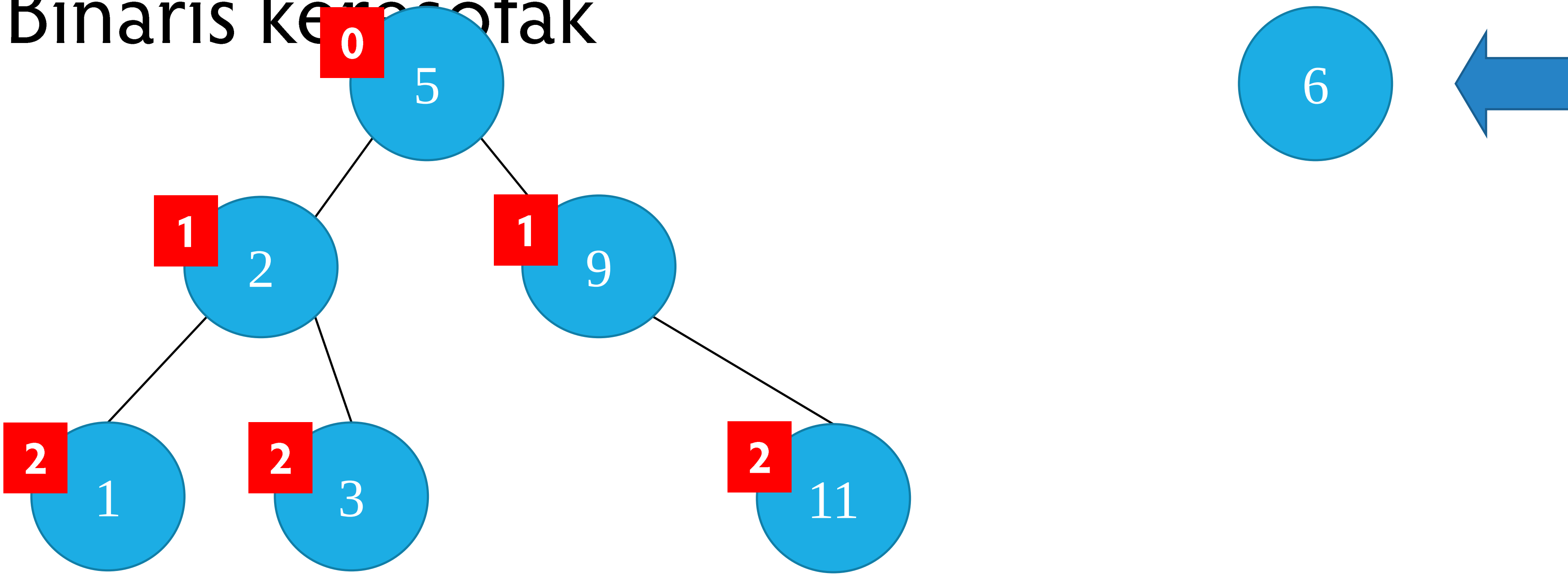
# Bináris keresőfák



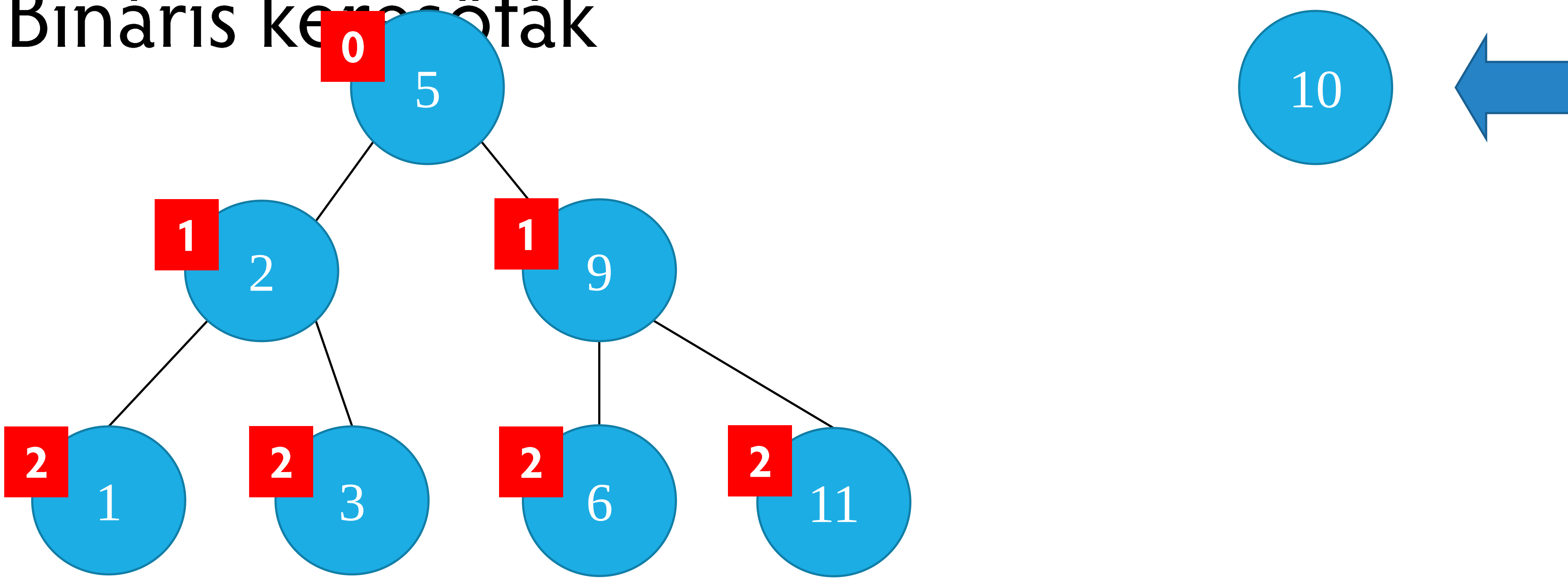
# Bináris keresőfák



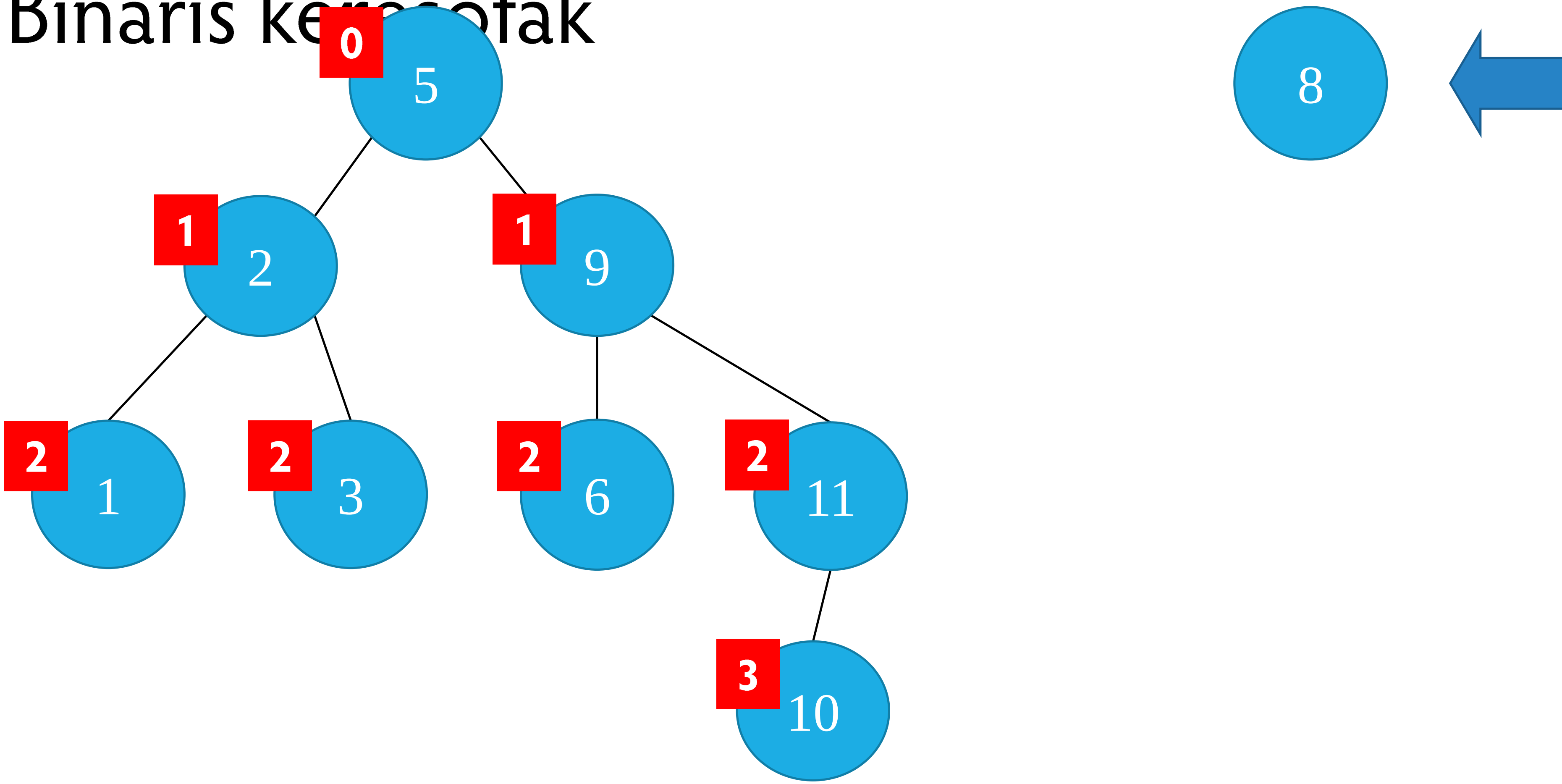
# Bináris keresőfák



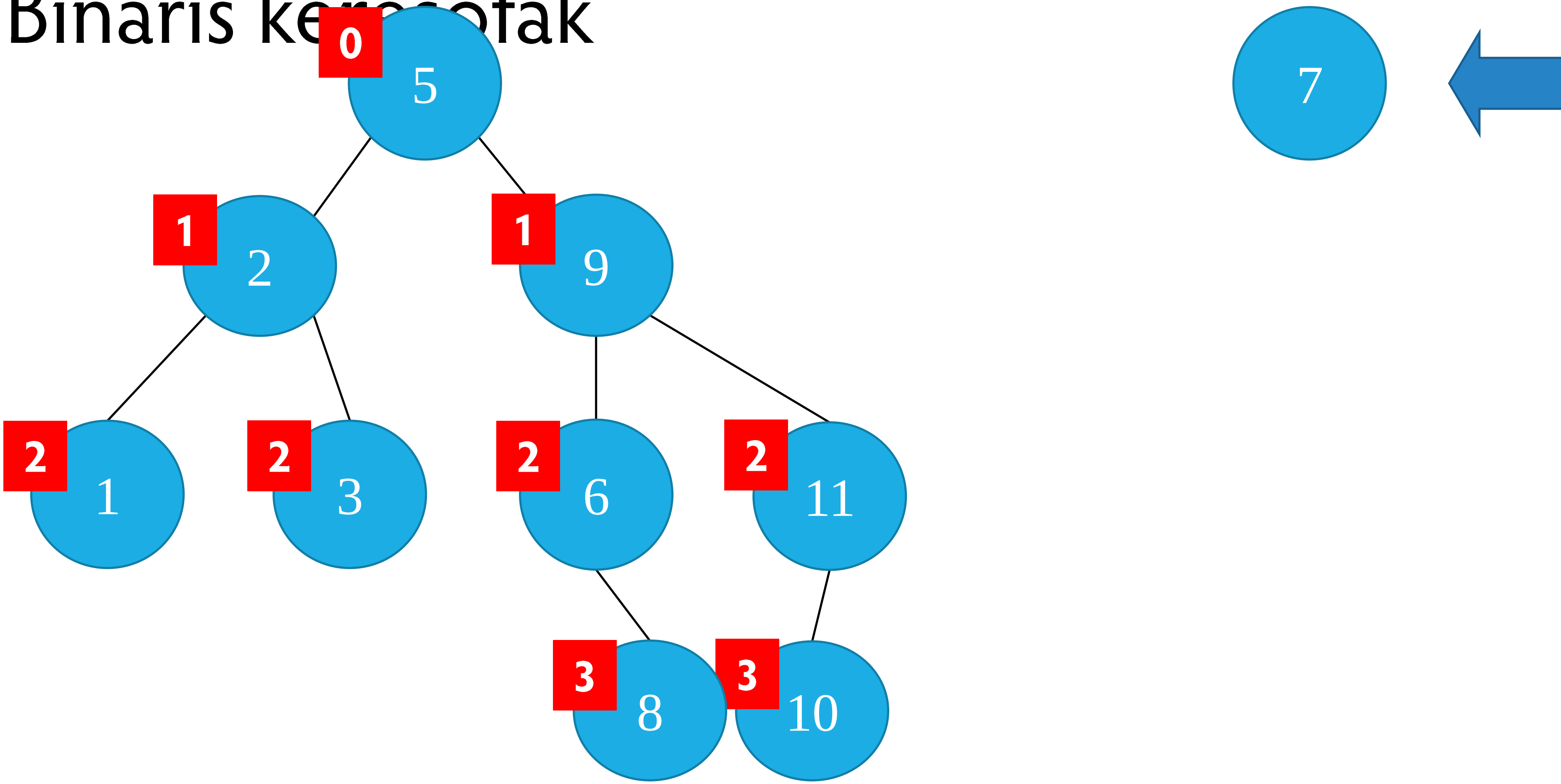
# Bináris keresőfák



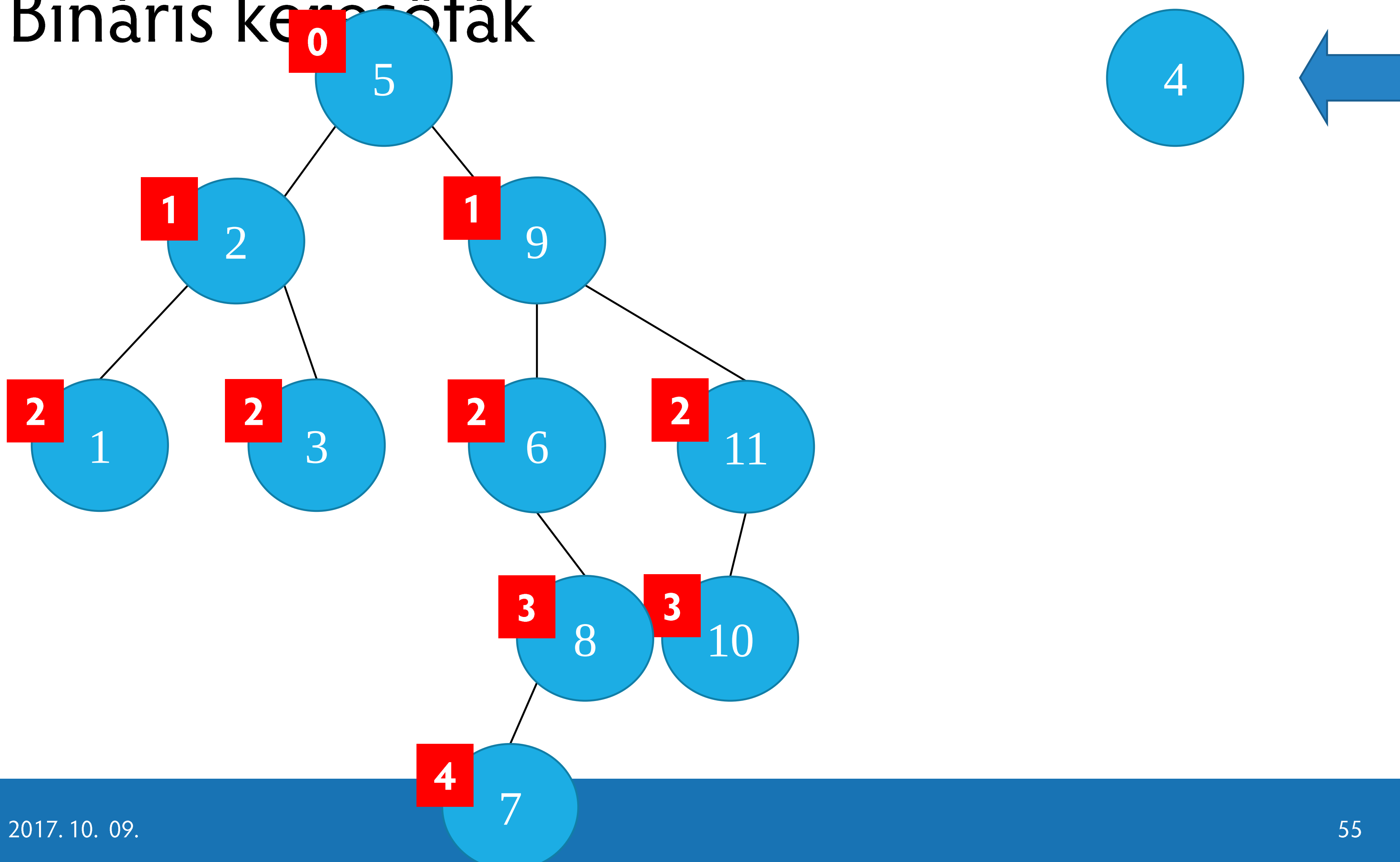
# Bináris keresőfák



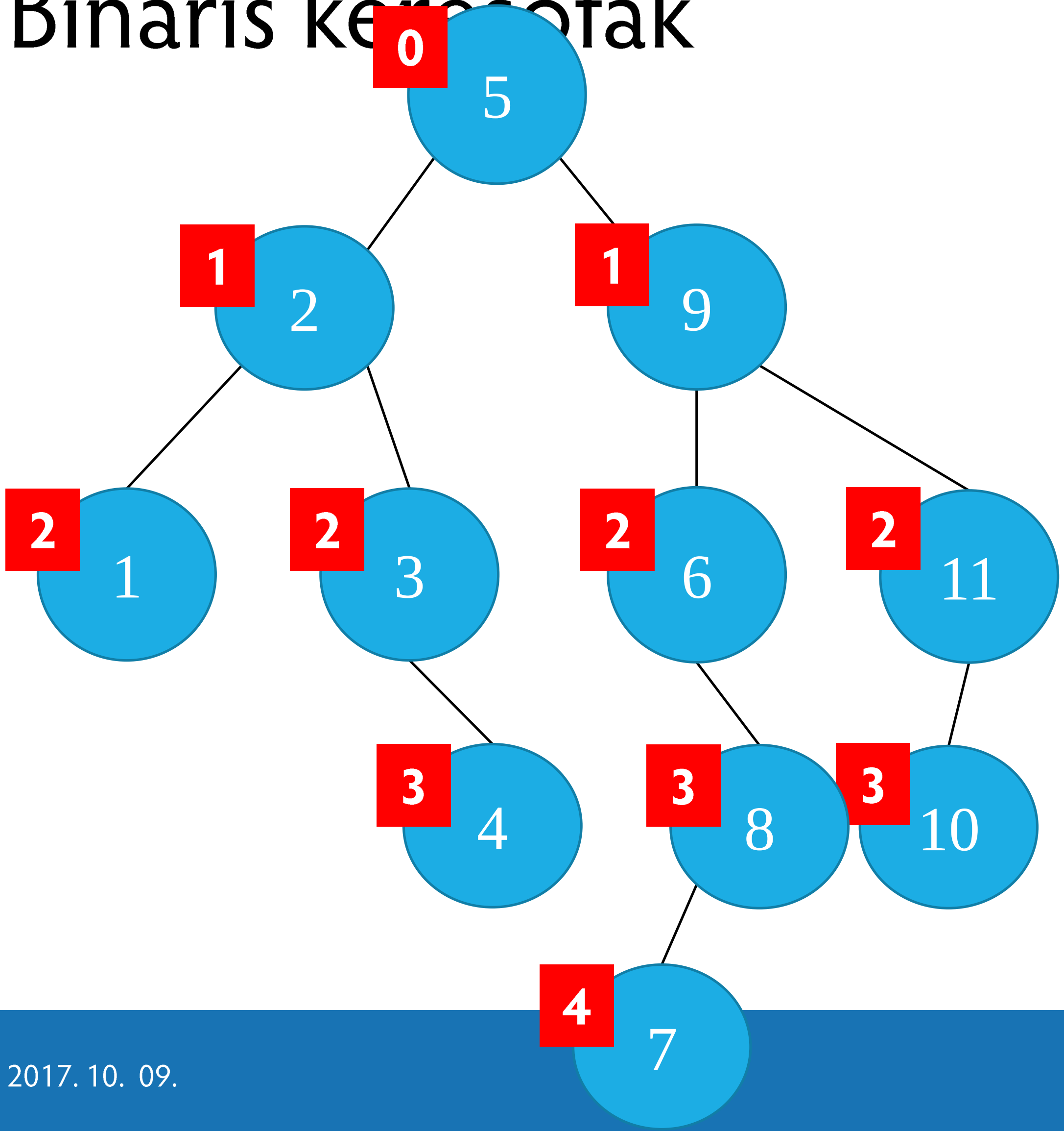
# Bináris keresőfák



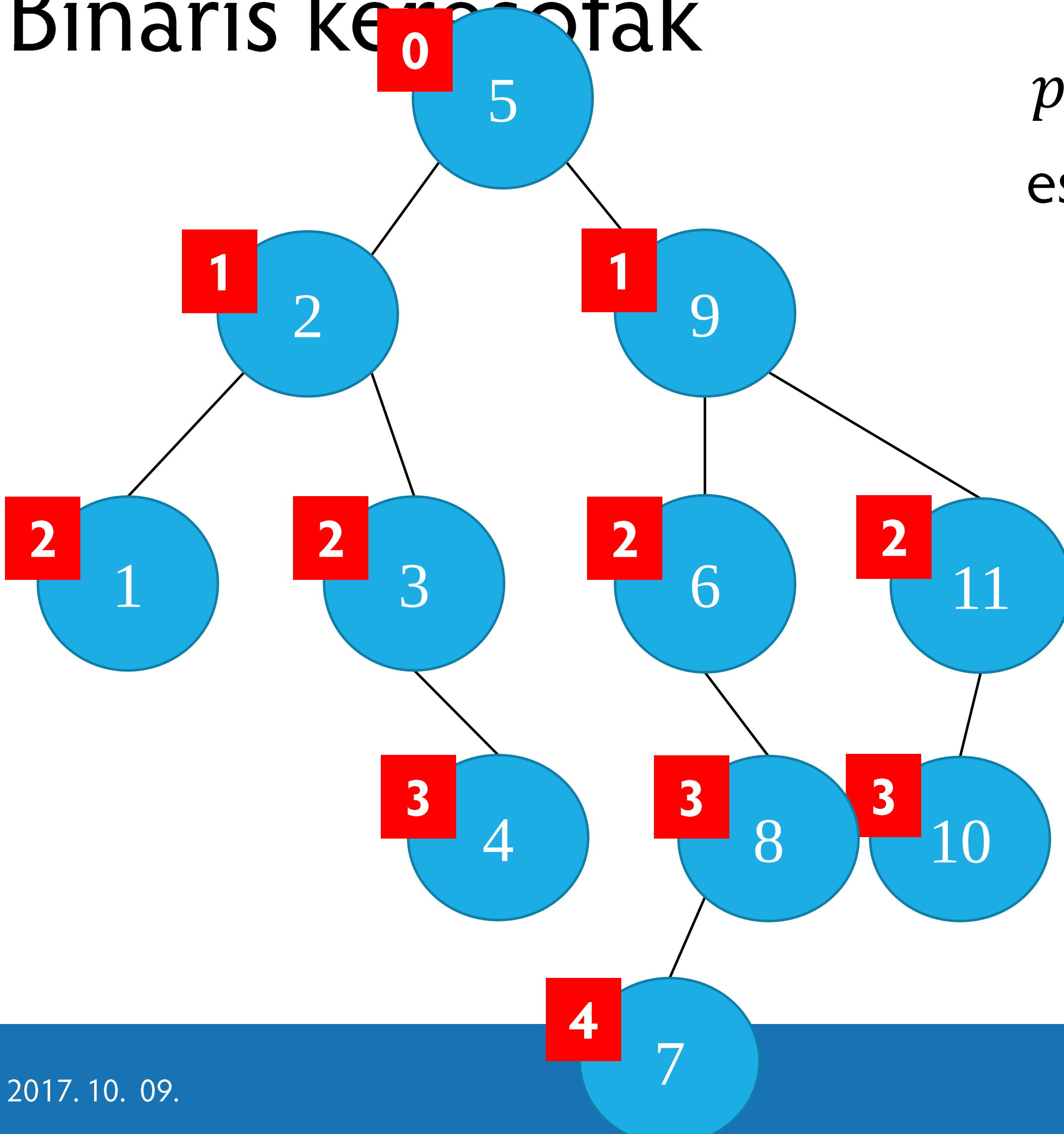
# Bináris keresőfák



# Bináris keresőfák



# Bináris keresőfák




$p = \; 5,2,9,3,11,1,6,10,8,7,4$

esetén

$$
\begin{aligned}
&\ddot{\mathrm{O}}(p) = \\
&(1+1) + \\
&(2+2+2+2) + \\
&(3+3+3) + 4 \\
&= 23
\end{aligned}
$$

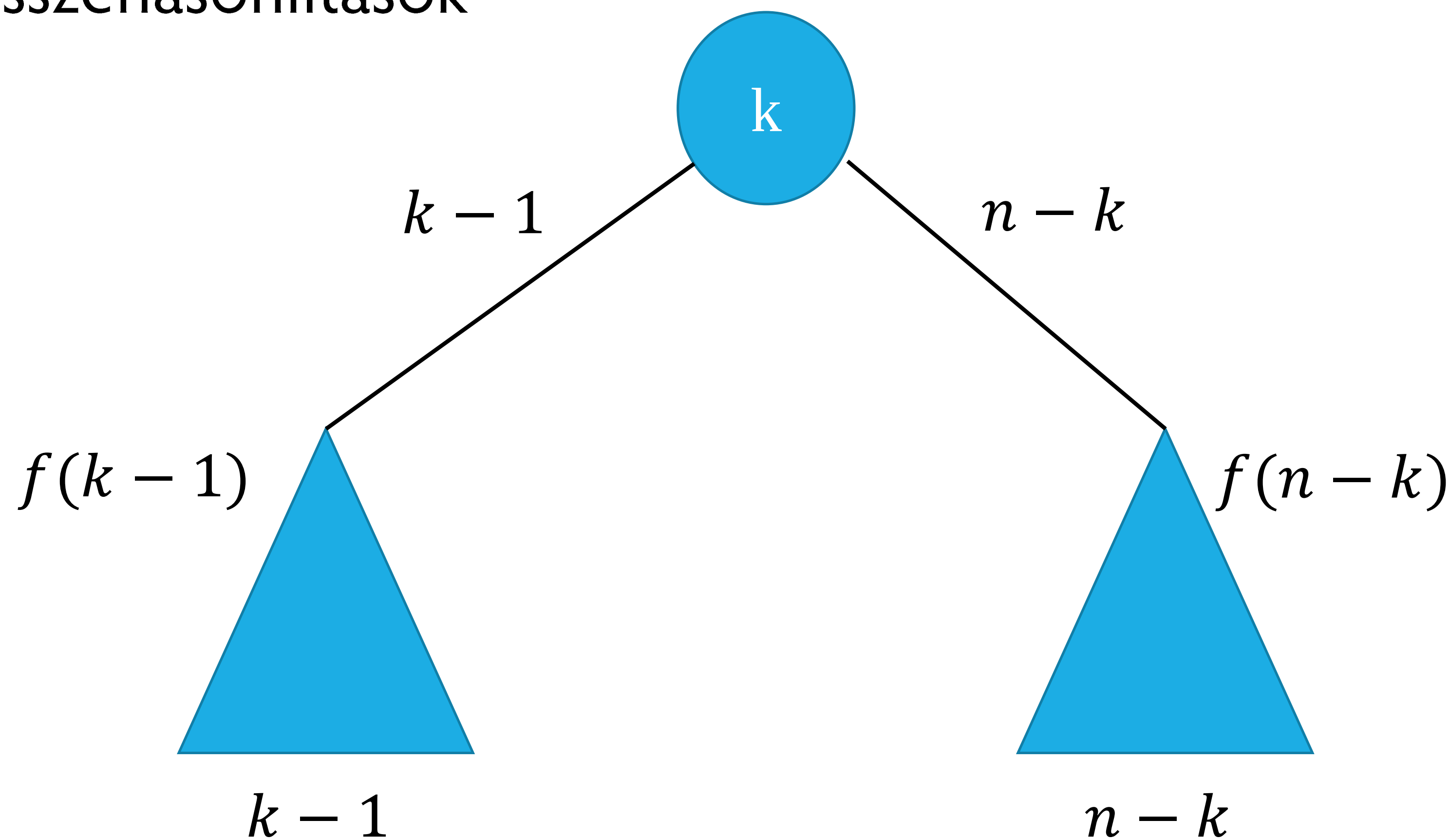
# Bináris keresőfák

- Határozzuk meg ennek az átlagát!
- Először meghatározzuk a csúcsmagasság összeget.
- Jelölés:
  - $f(n)$: $n$ adatból hány összehasonlítással lehet keresőfát építeni
  - $f(n|k)$: először a k érték jön ($k$ az input sorozat első eleme)
  - Tegyük fel, hogy minden sorozat egyforma valószínűségű

$$f(n) = \frac{1}{n}\sum_{k=1}^{n} f(n|k)$$

# Bináris keresőfák

- Összehasonlítások



# Bináris keresőfák

$$f(n) = \frac{1}{n}\sum_{k=1}^{n}\Big(k - 1 + f(k-1) + (n-k) + f(n-k)\Big)$$

$$f(0) = 0$$

$$f(n) = (n-1) + \frac{2}{n}\sum_{k=1}^{n-1} f(k)$$

Belátható, hogy $f(n) < 2n * \ln n \approx 1{,}39n \log_2 \mathrm{n}$

Tehát a fa csúcsmagasság összege $\approx 1{,}39n \log_2 n$

# Bináris keresőfák

- Tehát a fa csúcsmagasság összege $\approx 1{,}39n \log_2 n$
- Ebből következik, hogy a véletlen kulcssorozatból készítette bináris keresési fa várható csúcsmagasság-átlaga $\approx 1{,}39 \log_2 n$
  - Ez jó eredmény, mivel $1{,}39 \log_2 n = \mathcal{O}(\log_2 n)$.
    - Rosszabb, mint az optimális $h = \log_2 n$
    - Jobb, mint a $h = n$ szélsőséges eset
- Fontos, hogy ez csak akkor érvényes, ha a kulcsok véletlen sorrendben érkeznek, ami a valóságban általában nem teljesül.

# AVL Fa

Következő alkalommal