Adatszerkezetek
Vekov Géza
Bináris keresőfa
Adatok
Műveletek
Tökéletesen egyensúlyozott
Kiegyensúlyozott bináris fák
AVL fák
Műveletek
Forgatás

# Adatszerkezetek

## 07. Keresőfák

Vekov Géza

2023. április 5.

BABEŞ-BOLYAI TUDOMÁNYEGYETEM
Matematika és Informatika Kar

# Bináris keresőfa

# Ismétlés



## Bináris fa

- Olyan fa, melyben minden adatelemnek legtöbb két rákövetkezője van

## Bináris keresőfa

Olyan rendezett bináris fa, melyben:

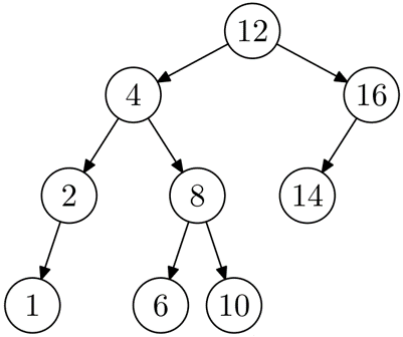
- Az adatelemek mindegyike rendelkezik egy kulccsal
- Minden adatelemre igaz, hogy:
  - Az adatelem bal oldali részfájában levő elemek kulcsai kisebbek az elem kulcsánál
  - Az adatelem jobb oldali részfájában levő elemek kulcsai nagyobbak az elem kulcsánál

# Bináris keresőfa: adatok

Adatszerkezetek
Vekov Géza
Bináris keresőfa
Adatok
Műveletek
Tökéletesen egyensúlyozott
Kiegyensúlyozott bináris fák
AVL fák
Műveletek
Forgatás

## Bináris keresőfa elemei

- **Kulcsok** (által jellemzett objektumok)
- Feltételezzük, hogy minden kulcs **egyedi**.
- Nem egyedi kulcsok esetén szükséges egy egyértelmű szabály az azonos kulccsal rendelkező elemek kezelésére



# Bináris keresőfa: műveletek



## Műveletek

- **Keresés(x)** - az x értékű elem keresése
- **Minimum/Maximum** - a bináris fa legkisebb/legnagyobb elemének meghatározása
- **Előző/Következő(x)** - az x értékű elem előtti/utáni elem a rendezett sorozatban
- **i. Elem(i)** - a bináris fa i. legkisebb elemének meghatározása
- **Rang(x)** - az x értékű elem rangjának meghatározása (az x értékű elem "sorszáma" a rendezett sorozatban)
- **Beszúrás(x)** - az x értékű elem beszúrása a bináris fába *
- **Törlés(x)** - az x értékű elem törlése a bináris fából *
- **Kiírás** - a bináris fa rendezett sorrendben való kiírása

* fontos, hogy a keresőfa tulajdonság megmaradjon

# Bináris keresőfa: műveletek



## Keresés

**x** - keresett elem

- **előfeltétel**: -
- **utófeltétel**: *igazat* térít ha megtalálta az elemet, *hamisat* ha nem (alternatíva: egy *mutatót* térít, ha megtalálta, *NULL*-t ha nem)*

**Algoritmus**:

- Ha a fa üres → a keresett elem nem található a fában
- Összehasonlítjuk a gyökérben található kulcsot ($c$) a keresett elemmel
  - Ha $x = c$: keresés vége
  - Ha $x < c$: megkeressük $x$-et a gyökérelem bal alfájában
  - Különben: megkeressük $x$-et a gyökérelem jobb alfájában

* - függ a feladattól

# Bináris keresőfa: műveletek

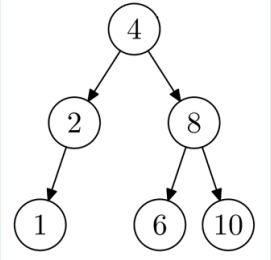


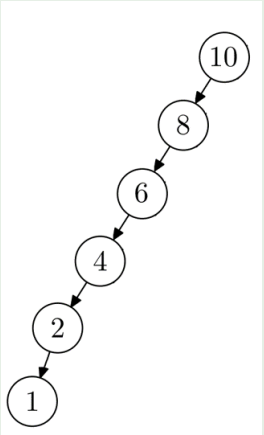
## Keresés *(keresett elem x)*

- A keresőfa **formája** nem egyértelmű
- A műveletek futási ideje függ a fa magasságától
  - Keresés időbonyolultsága:

$O(h)$
ahol $h$ a magasság

$O(\log n)$ vs $O(n)$



# Bináris keresőfa: műveletek

## Minimum meghatározása

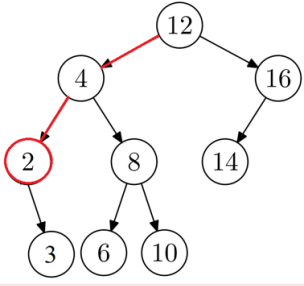
**x** - keresett elem

- **előfeltétel**: -
- **utófeltétel**: visszatéríti a minimális kulcsú elemet / elem címét. Meghatározott értéket térít, ha üres a fa.

**Algoritmus**:

- Ha a fa üres → a keresett elem nem található a fában
- Amíg az aktuális csomópontnak van bal gyereke → balra lépünk
- Ha nincs bal gyereke → visszatérítjük az aktuális elem értékét a fa minimumaként

## Példa



# Bináris keresőfa: műveletek



## Minimum meghatározása

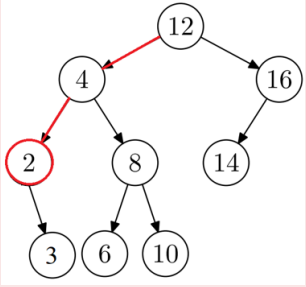
**x** - keresett elem

- **előfeltétel**: -
- **utófeltétel**: visszatéríti a minimális kulcsú elemet / elem címét. Meghatározott értéket térít, ha üres a fa.

**Algoritmus**:

- Ha a fa üres → a keresett elem nem található a fában
- Amíg az aktuális csomópontnak van bal gyereke → balra lépünk
- Ha nincs bal gyereke → visszatérítjük az aktuális elem értékét a fa minimumaként

## Példa




## Maximum meghatározása

Hasonlóan

# Bináris keresőfa: műveletek



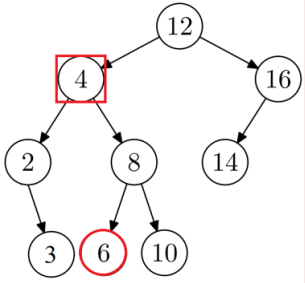
## Következő elem meghatározása (*a* csomópont)

**a** - csomópont

- **előfeltétel**: -
- **utófeltétel**: visszatéríti a következő csomópont címét, NULL-t, ha nincs ilyen.
- **1. eset**: *a*-nak van jobb gyereke:
  - Visszatérítjük az *a* csomópont jobb részfájának a minimumát

## Példa

4-nek a következője



# Bináris keresőfa: műveletek

## Következő elem meghatározása (*a* csomópont)

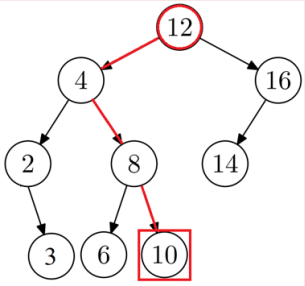
**a** - csomópont

- **előfeltétel**: -
- **utófeltétel**: visszatéríti a következő csomópont címét, *NULL*-t, ha nincs ilyen.

- **1. eset**: *a*-nak van jobb gyereke:
  - Visszatérítjük az *a* csomópont jobb részfájának a minimumát
- **2. eset**: *a*-nak nincs jobb gyereke:
  - Addig lépünk felfele, amíg egy x-nél nagyobb értéket nem találunk
  - Visszatérítjük az első nagyobb elemet (vagy *NULL*-t, ha nincs ilyen elem)

## Példa

10-nek a következője



# Bináris keresőfa: műveletek



## Következő elem meghatározása (*a* csomópont)

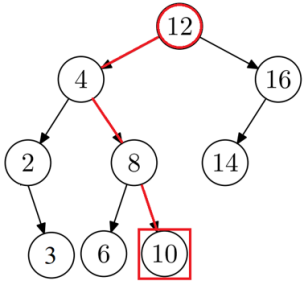
**a** - csomópont

- **előfeltétel**: -
- **utófeltétel**: visszatéríti a következő csomópont címét, *NULL*-t, ha nincs ilyen.

- **1. eset**: *a*-nak van jobb gyereke:
  - Visszatérítjük az *a* csomópont jobb részfájának a minimumát
- **2. eset**: *a*-nak nincs jobb gyereke:
  - Addig lépünk felfele, amíg egy x-nél nagyobb értéket nem találunk
  - Visszatérítjük az első nagyobb elemet (vagy *NULL*-t, ha nincs ilyen elem)

## Példa

10-nek a következője




## Előző elem meghatározása

Hasonlóan

# Bináris keresőfa: műveletek



## Beszúrás

- **x** - beszúrandó elem
- **előfeltétel**: -
- **utófeltétel**: $x$ szabályos helyet foglal el a bináris fában (a bináris fának megmaradnak a jellemző tulajdonságai). Jelezzük, hogyha nem sikerült beszúrni az elemet*.

**Algoritmus**:

- Ha a fa üres → a beszúrandó elem lesz a fa egyetlen eleme (levélelem és gyökér), algoritmus vége
- Összehasonlítjuk a gyökérben található kulcsot ($c$) a keresett elemmel
  - Ha $x < c$: beszúrjuk az $x$-et a gyökérelem bal oldali részfájába
  - Ha $x > c$: beszúrjuk az $x$-et a gyökérelem jobb oldali részfájába
  - Ha $x = c$: $x$ nem szúrható be a fába, mert nem szerepelhet két azonos kulcsú elem a keresőfában*

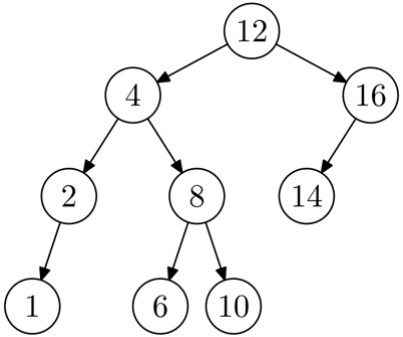
* - függ a feladattól

# Bináris keresőfa: műveletek



## Feladat

- Szúrjuk be a 3-ast az ábrán látható bináris fába!

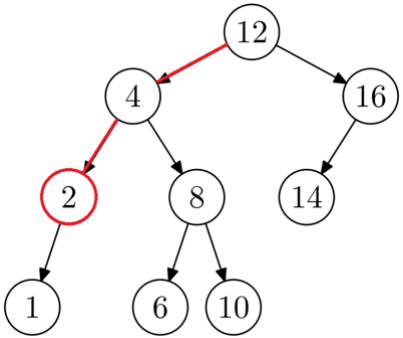


# Bináris keresőfa: műveletek



## Feladat

- Szúrjuk be a 3-ast az ábrán látható bináris fába!

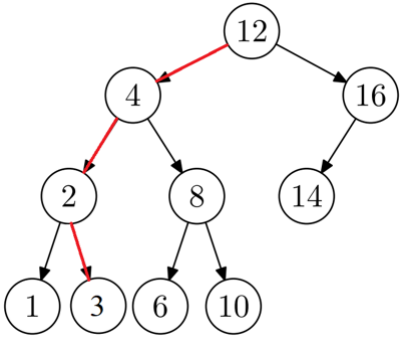


# Bináris keresőfa: műveletek



## Feladat

- Szúrjuk be a 3-ast az ábrán látható bináris fába!



# Bináris keresőfa: műveletek



## Törlés

- **x** - törlendő elem
- **előfeltétel**: -
- **utófeltétel**: *x* törlődik a bináris keresőfa elemei közül*. A fa tulajdonságai megmaradnak.

* - függ a feladattól

# Bináris keresőfa: műveletek



## Törlés

**Algoritmus**:

- Ha a fa üres → nem tudunk törölni, algoritmus vége
- Összehasonlítjuk a gyökérben található kulcsot ($c$) a keresett elemmel
  - Ha $x < c$: töröljük az $x$-et a gyökérelem bal oldali részfájából
  - Ha $x > c$: töröljük az $x$-et a gyökérelem jobb oldali részfájából
  - Ha $x = c$: megnézzük, hogy a gyökérelemnek hány rákövetkezője van ($k$):
    - $k = 0$: töröljük a csomópontot
    - $k = 1$: felülírjuk a gyökérelemet a rákövetkező elemmel (egy szinttel fennebb csúsztatjuk a györkérelem nem üres részfáját)
    - $k = 2$: a gyökérelemet felülírjuk a bal oldali részfája legjobboldalibb elemének az értékével, majd a bal oldali részfából töröljük a legjobboldalibb elemet

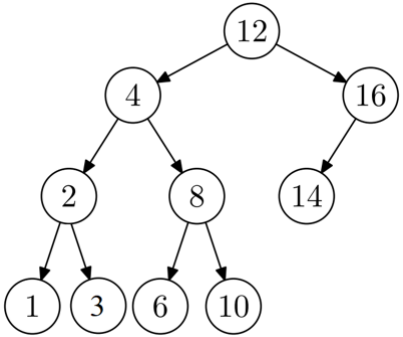
      *vagy*
    - $k = 2$: a gyökérelemet felülírjuk a jobb oldali részfája legbaloldalibb elemének az értékével, majd a jobb oldali részfából töröljük a legbaloldalibb elemet

# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 4-est az ábrán látható bináris fából!

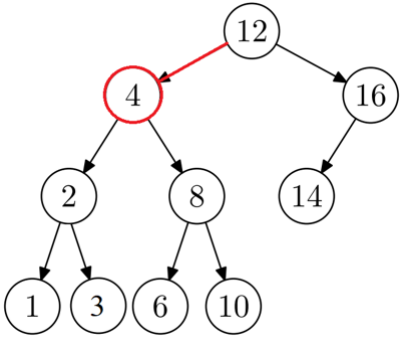


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 4-est az ábrán látható bináris fából!

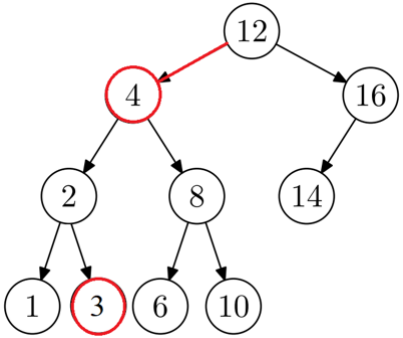


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 4-est az ábrán látható bináris fából!

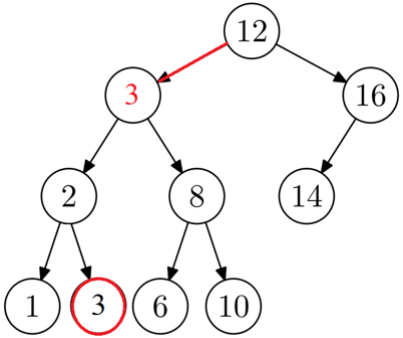


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 4-est az ábrán látható bináris fából!

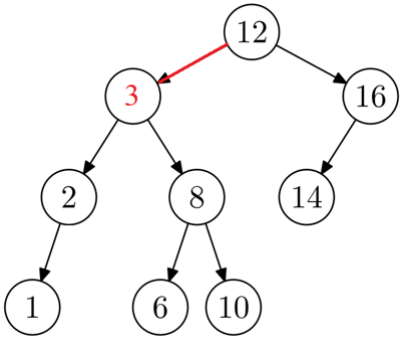


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 4-est az ábrán látható bináris fából!

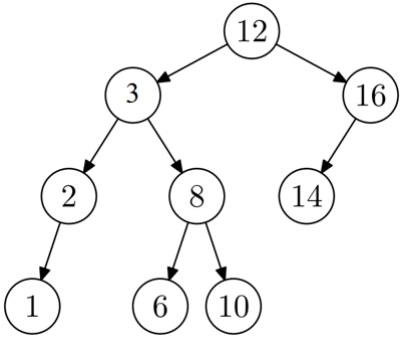


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 3-ast az ábrán látható bináris fából!

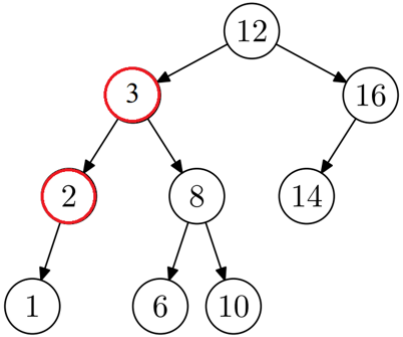


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 3-ast az ábrán látható bináris fából!

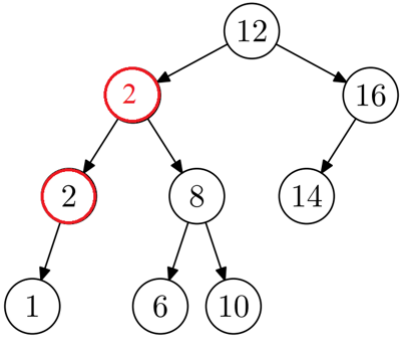


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 3-ast az ábrán látható bináris fából!

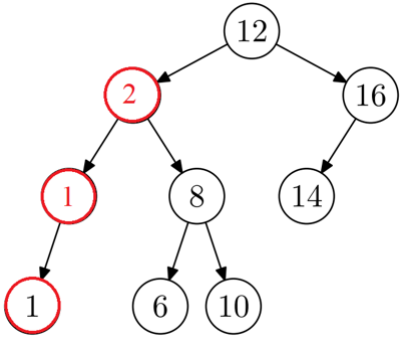


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 3-ast az ábrán látható bináris fából!

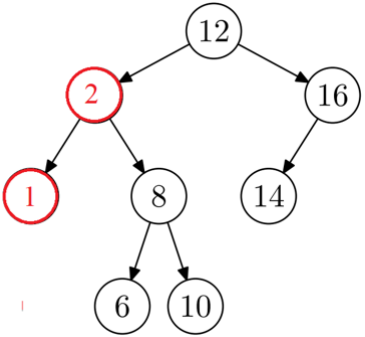


# Bináris keresőfa: műveletek



## Feladat

- Töröljük a 3-ast az ábrán látható bináris fából!



# Bináris keresőfa: műveletek



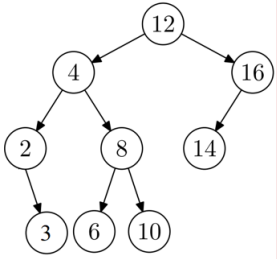
## i. elem keresése

**i** - sorszám a rendezett sorozatban

- **előfeltétel**: a fának van legalább *i* eleme
- **utófeltétel**: visszatéríti az *i.* csomópont címét, *NULL*-t, ha nincs ilyen.

## Példa

Keressük meg a 4. elemet.



# Bináris keresőfa: műveletek



## i. elem keresése

**i** - sorszám a rendezett sorozatban

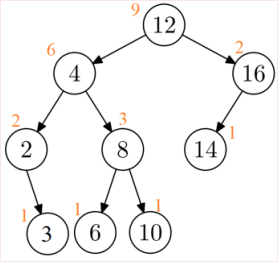
- **előfeltétel**: a fának van legalább *i* eleme
- **utófeltétel**: visszatéríti az *i.* csomópont címét, *NULL*-t, ha nincs ilyen.

**Ötlet**: tároljunk el minden csomópont esetén egy plusz információt ami segíti a keresést:

- **méret(x)**: az x gyökerű részfában levő csomópontok száma
- **Megjegyzés**: legyen *x* gyereke *y* és *z*:
  - méret(x) = méret(y) + méret(z) + 1
- **Hátrány**: minden módosításkor (törlés, beszúrás) frissíteni kell a méret információt

## Példa

Keressük meg a 4. elemet.



# Bináris keresőfa: műveletek



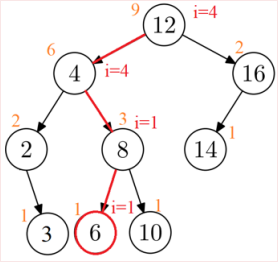
## i. elem keresése

**Algoritmus**

- legyen: $x$ a fa gyökere,
- $y$ és $z$ az $x$ bal és jobb gyereke
- legyen $a = méret(y)$
- ha $a = i - 1$: a keresett elem az $x$
- ha $a \geq i$:
  - keressük meg az $i$-dik elemet az $x$ bal részfájában
- ha $a < i - 1$:
  - keressük meg az *(i-a-1)*-dik elemet az $x$ jobb részfájában

## Példa

Keressük meg a 4. elemet.



# Bináris keresőfa: műveletek



## Kiírás növekvő sorrendben

- Inorder bejárás

# Bináris keresőfa - műveletek



## Összehasonlítás - átlagos esetben vett műveletigények

| | Rendezett tömb | Bináris keresőfa |
|---|---|---|
| **Keresés** + | $O(\log n)$ | $O(\log n)$ |
| **Minimum / Maximum** * | $O(1)$ | $O(\log n)$ |
| **Előző / Következő** | $O(1)$ | $O(\log n)$ |
| **i. elem** | $O(1)$ | $O(\log n)$ |
| **Rang** | O(log n) | $O(\log n)$ |
| **Beszúrás** | $O(n)$ | $O(\log n)$ |
| **Törlés** | $O(n)$ | $O(\log n)$ |
| **Kiírás** | $O(n)$ | $O(n)$ |

\+ hatékonyabb hash táblával

$O(1)$

* hatékonyabb kupaccal

$O(1)$

# Tökéletesen egyensúlyozott

# Tökéletesen kiegyensúlyozott bináris fa

## Minimális magasságú fa

- Egy fa minimális magasságú, ha adott számú elemet nem lehetne kisebb magasságú bináris fában elhelyezni.

## Tökéletesen kiegyensúlyozott bináris fa

- Egy bináris fa tökéletesen kiegyensúlyozott, ha bármely elemének bal és jobb oldali részfájában az elemek darabszáma legfennebb 1-gyel tér el

## Megjegyzés

- Megjegyzés: minden tökéletesen kiegyensúlyozott fa minimális magasságú

**Kérdés**

Hogyan lehetne kiegyensúlyozni egy keresőfát egy elem beszúrása után (újraépítés nélkül)?

`www.menti.com` - *1551 9871*

# Kiegyensúlyozott bináris fák

# Kiegyensúlyozott bináris fák

Adatszerkezetek
Vekov Géza
Bináris keresőfa
Adatok
Műveletek
Tökéletesen egyensúlyozott
Kiegyensúlyozott bináris fák
AVL fák
Műveletek
Forgatás

## Műveletek időbonyolultsága

- Bináris fák esetén, a műveletek futási ideje a fa magasságától függ.

## Cél

- Minél gyorsabb műveletek.

## Megoldás

- kiegyensúlyozott bináris fa

# Kiegyensúlyozott bináris fák



## Példák

Kiegyensúlyozott fák a következő adatszerkezetek:

- AVL fa
- Piros-fekete fa
- Splay fa

# AVL fák

# AVL fák

Adatszerkezetek
Vekov Géza
Bináris keresőfa
Adatok
Műveletek
Tökéletesen egyensúlyozott
Kiegyensúlyozott bináris fák
AVL fák
Műveletek
Forgatás

## Megfontolások

- A tökéletesen kiegyensúlyozott keresőfák egyensúlyának a megőrzése és helyreállítása minden módosítás (beszúrás, törlés) után egy rendkívűl bonyolult feladat
- **AVL fák**:
  - Lazítjuk a megszorítást ami a részfák elemeinek a számát illeti
  - *Megelégszünk* a magasság szempontjából egyensúlyozott fákkal

## Megjegyzés

- Georgy **A**delson-**V**elsky és Evgenii **L**andis szovjet feltalálók (1962)

# AVL fák

## Definíció

- **magasság szempontjából kiegyensúlyozott** bináris keresőfa
- azaz: bármely csomópont két részfájának a mélysége közötti különbség legtöbb 1

## Tulajdonságok

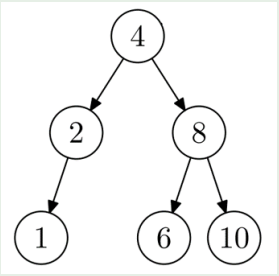
- Egy AVL fa bármely részfája szintén AVL fa.
- Bármely tökéletesen kiegyensúlyozott fa egyben AVL fa (fordítva nem igaz).
- Minden levél az utolsó két szinten található.

# AVL fák



## Példa

- Kérdés: AVL fa-e az alábbi fa?
- Válasz:

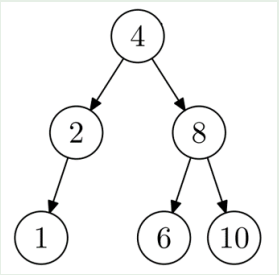


# AVL fák



## Példa

- Kérdés: AVL fa-e az alábbi fa?
- Válasz: **igen**

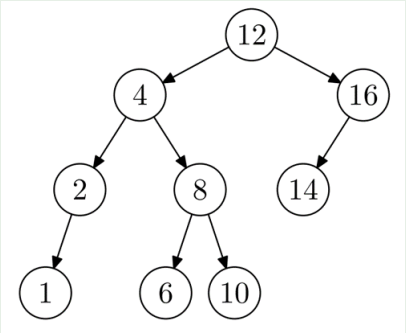


# AVL fák



## Példa

- Kérdés: AVL fa-e az alábbi fa?
- Válasz:

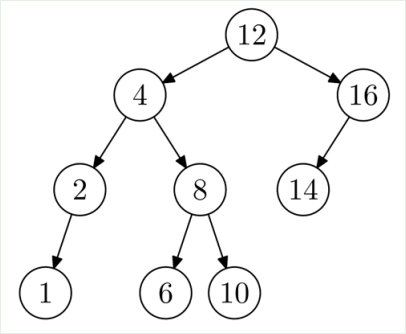


# AVL fák



## Példa

- Kérdés: AVL fa-e az alábbi fa?
- Válasz: **igen**



# AVL fa: műveletek



## AVL fa: Műveletek

- **Keresés(x)** - az x értékű elem keresése
- **Minimum/Maximum** - az AVL fa legkisebb/legnagyobb elemének meghatározása
- **Előző/Következő(x)** - az x értékű elem előtti/utáni elem a rendezett sorozatban
- **i. Elem(i)** - az AVL fa i. dik legkisebb elemének meghatározása
- **Rang(x)** - az x értékű elem rangjának meghatározása (az x értékű elem "sorszáma" a rendezett sorozatban)
- **Beszúrás(x)** - az x értékű elem beszúrása az AVL fába *
- **Törlés(x)** - az x értékű elem törlése az AVL fából *
- **Kiírás** - az AVL fa rendezett sorrendben való kiírása

* fontos, hogy a keresőfa tulajdonság és a kiegyensúlyozottság megmaradjon

# AVL fák: Forgatás



## Megfontolások

- Beszúrás / törlés esetén **sérülhet a kiegyensúlyozottság**:
  - egy csomópont bal és a jobb részfáinak magassága közötti különbség $> 1$

**Ötlet**: állítsuk vissza lokálisan az egyensúlyt egy csomópont esetén.

**Módszer**: forgatás

## Megjegyzés

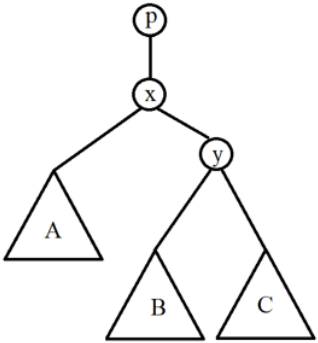
- A forgatásokat, mint alapműveleteket, valamennyi kiegyensúlyozott fa esetén használjuk.

# AVL fák: Forgatás



## Bal forgatás

- Szülő: **x**
- Jobb gyerek: **y**



# AVL fák: Forgatás

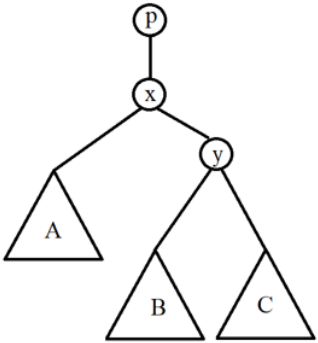


## Bal forgatás

- Szülő: **x**
- Jobb gyerek: **y**

**Tudjuk, hogy**:

- $x < y$



# AVL fák: Forgatás

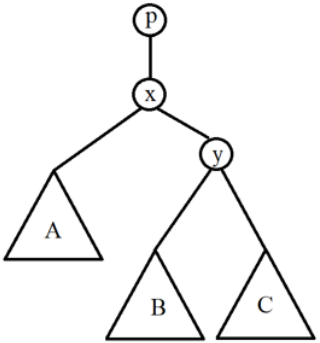


## Bal forgatás

- Szülő: **x**
- Jobb gyerek: **y**

**Tudjuk, hogy**:

- $x < y$
- $\forall e \in A, e < x < y$
- $\forall e \in B, x < e < y$
- $\forall e \in C, x < y < e$



# AVL fák: Forgatás

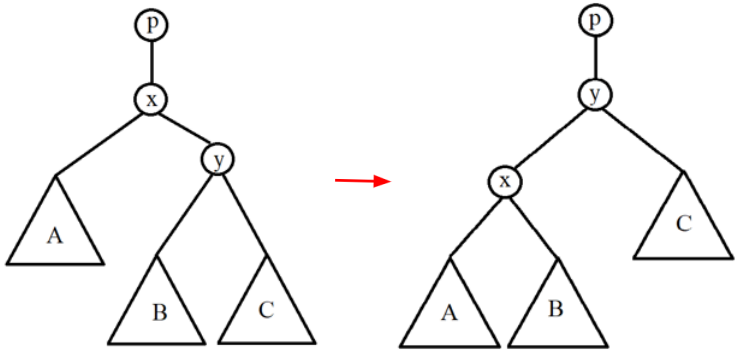


## Bal forgatás

- Szülő: **x**
- Jobb gyerek: **y**

**Tudjuk, hogy**:

- $x < y$
- $\forall e \in A, e < x < y$
- $\forall e \in B, x < e < y$
- $\forall e \in C, x < y < e$

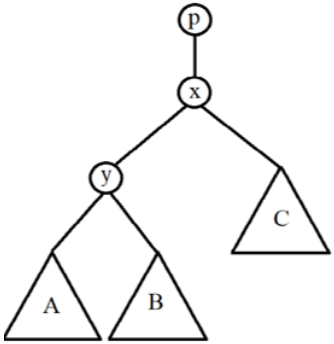


# AVL fák: Forgatás



## Jobb forgatás

- Szülő: **x**
- Bal gyerek: **y**



# AVL fák: Forgatás

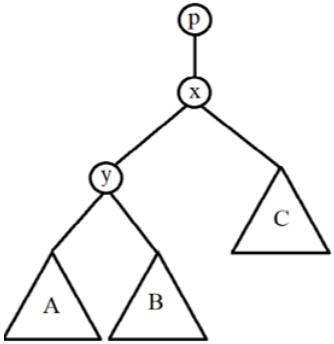


## Jobb forgatás

- Szülő: **x**
- Bal gyerek: **y**

**Tudjuk, hogy**:

- $x > y$



# AVL fák: Forgatás

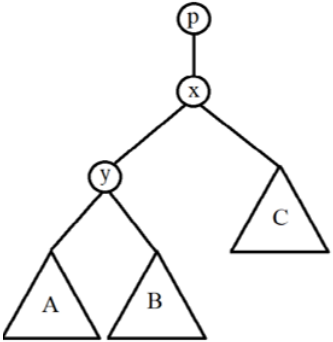


## Jobb forgatás

- Szülő: **x**
- Bal gyerek: **y**

**Tudjuk, hogy**:

- $x > y$
- $\forall e \in A, e < x < y$
- $\forall e \in B, y < e < x$
- $\forall e \in C, x < y < e$



# AVL fák: Forgatás

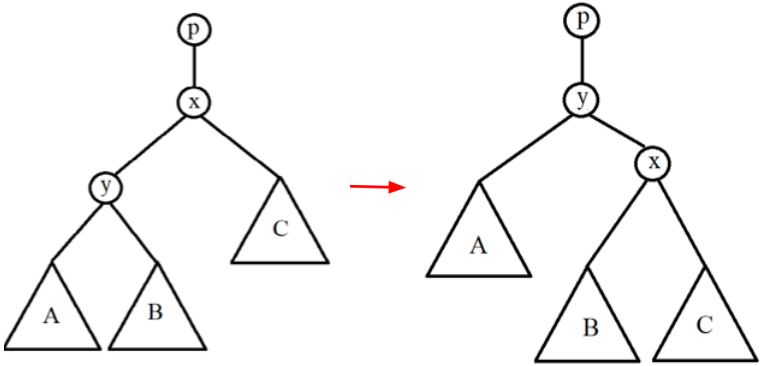


## Jobb forgatás

- Szülő: **x**
- Bal gyerek: **y**

**Tudjuk, hogy**:

- $x > y$
- $\forall e \in A, e < x < y$
- $\forall e \in B, y < e < x$
- $\forall e \in C, x < y < e$



# AVL fák: Műveletek



## Beszúrás

Feltételezzük, hogy az új elem beszúrása növelné a fa *bal részfájának a mélységét*.

Legyen $m(bal)$ és $m(jobb)$ a két részfa beszúrás előtti mélysége. Ekkor a következő három eset fordulhat elő:

# AVL fák: Műveletek



## Beszúrás

Feltételezzük, hogy az új elem beszúrása növelné a fa *bal részfájának a mélységét*.

Legyen $m(bal)$ és $m(jobb)$ a két részfa beszúrás előtti mélysége. Ekkor a következő három eset fordulhat elő:

1. $m(bal) = m(jobb)$, beszúrás után $\rightarrow$ $m(bal) = m(jobb) + 1$
   - az AVL tulajdonság nem sérül

# AVL fák: Műveletek



## Beszúrás

Feltételezzük, hogy az új elem beszúrása növelné a fa *bal részfájának a mélységét*.

Legyen $m(bal)$ és $m(jobb)$ a két részfa beszúrás előtti mélysége. Ekkor a következő három eset fordulhat elő:

1. $m(bal) = m(jobb)$, beszúrás után $\rightarrow$ $m(bal) = m(jobb) + 1$
   - az AVL tulajdonság nem sérül
2. $m(bal) < m(jobb)$, beszúrás után $\rightarrow$ $m(bal) = m(jobb)$
   - az AVL tulajdonság nem sérül

# AVL fák: Műveletek



## Beszúrás

Feltételezzük, hogy az új elem beszúrása növelné a fa *bal részfájának a mélységét*.

Legyen $m(bal)$ és $m(jobb)$ a két részfa beszúrás előtti mélysége. Ekkor a következő három eset fordulhat elő:

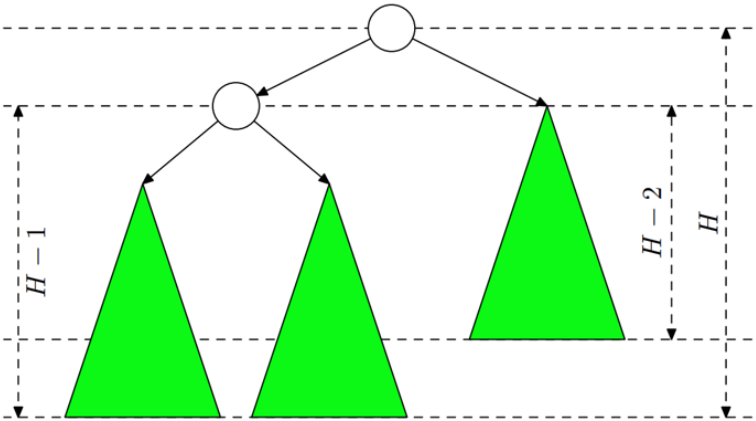
1. $m(bal) = m(jobb)$, beszúrás után $\rightarrow$ $m(bal) = m(jobb) + 1$
   - az AVL tulajdonság nem sérül
2. $m(bal) < m(jobb)$, beszúrás után $\rightarrow$ $m(bal) = m(jobb)$
   - az AVL tulajdonság nem sérül
3. $m(bal) > m(jobb)$, beszúrás után $\rightarrow$ $m(bal) = m(jobb) + 2$
   - az AVL tulajdonság sérül, az egyensúlyt helyre kell állítani

# AVL fák: Műveletek



## **LL** beszúrás

**Alaphelyzet**
Az AVL tulajdonság fennáll.



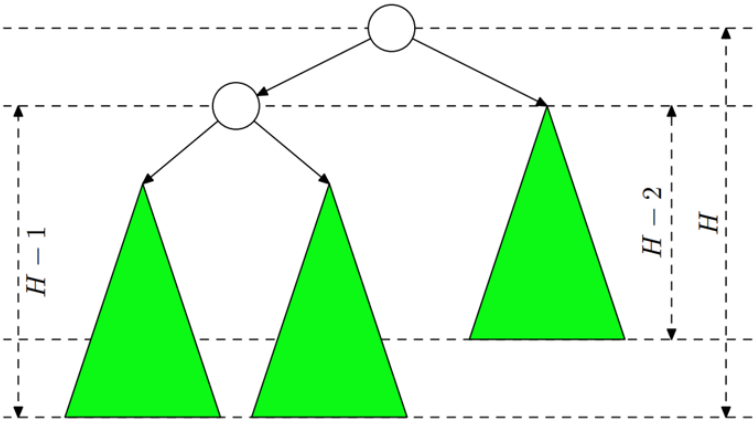
# AVL fák: Műveletek



## LL beszúrás

**Alaphelyzet**
Az AVL tulajdonság fennáll.

**LL beszúrás**:
A *bal* oldali részfa *bal* oldali részfájába szúrunk be.



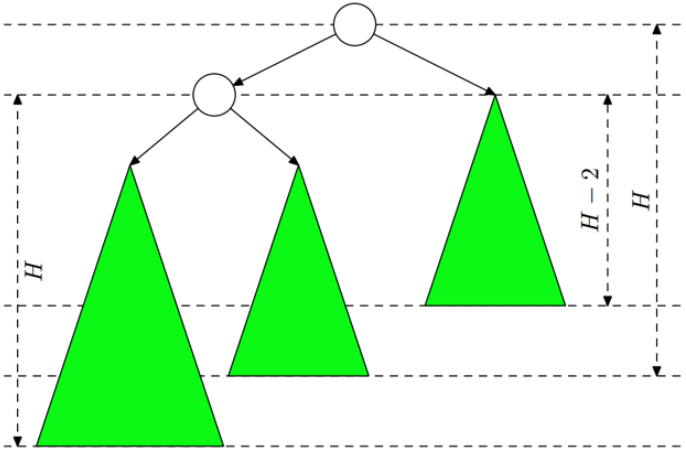
# AVL fák: Műveletek



## LL beszúrás

**LL beszúrás**:

A *bal* oldali részfa *bal* oldali részfájába szúrunk be.



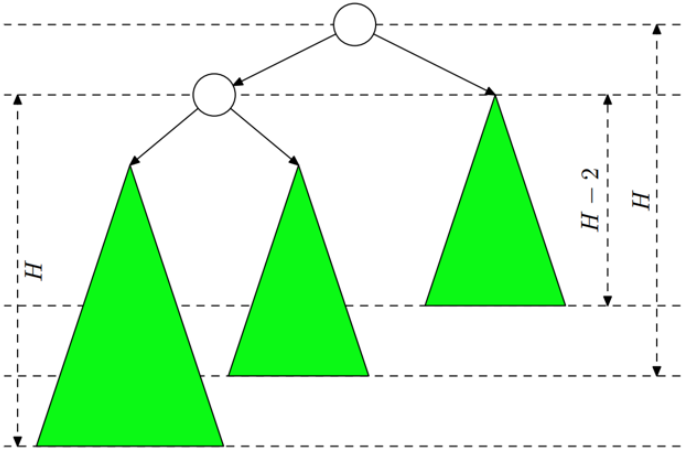
# AVL fák: Műveletek



## **LL** beszúrás

**LL beszúrás**:
A *bal* oldali részfa *bal* oldali részfájába szúrunk be.

Az AVL tulajdonság **sérül**.



# AVL fák: Műveletek



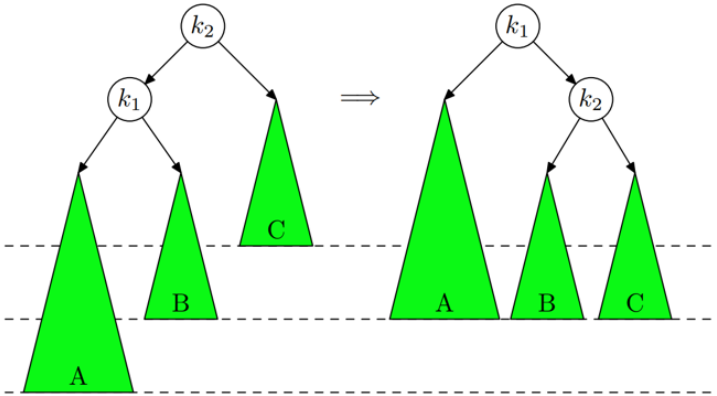
## **LL** beszúrás

**LL beszúrás**:

A *bal* oldali részfa *bal* oldali részfájába szúrunk be.

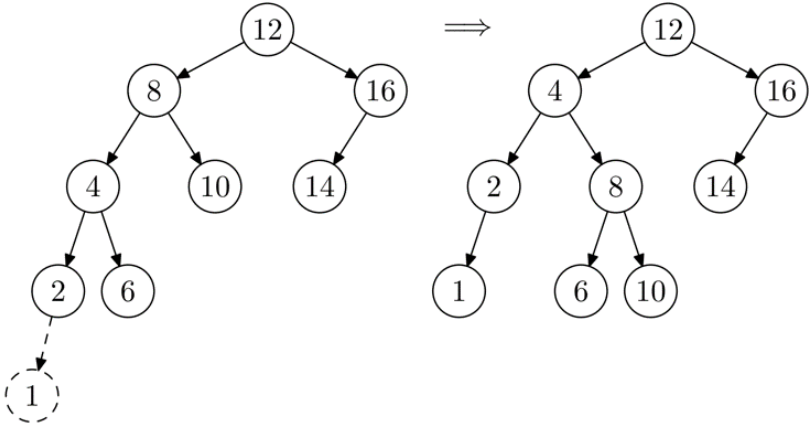
Az AVL tulajdonság **sérül**.

Az AVL tulajdonságot egy **jobb** forgatással vissza tudjuk állítani



# AVL fák: Műveletek

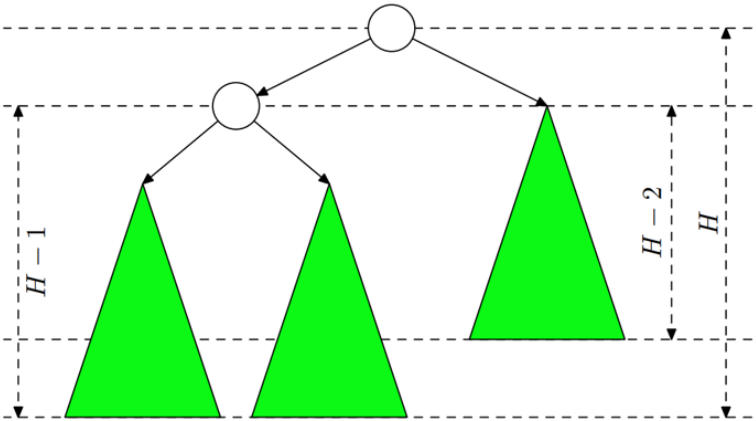
## Példa: **LL** beszúrás



# AVL fák: Műveletek

## LR beszúrás

**Alaphelyzet**
Az AVL tulajdonság fennáll.



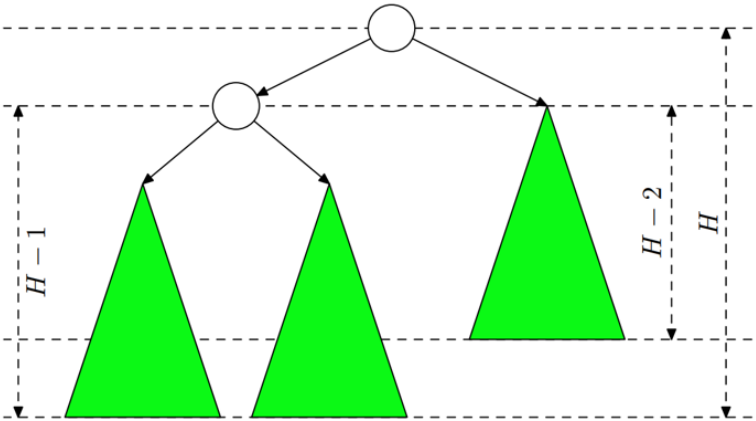
# AVL fák: Műveletek



## LR beszúrás

**Alaphelyzet**
Az AVL tulajdonság fennáll.

**LR beszúrás**:
A *bal* oldali részfa *jobb* oldali részfájába szúrunk be.



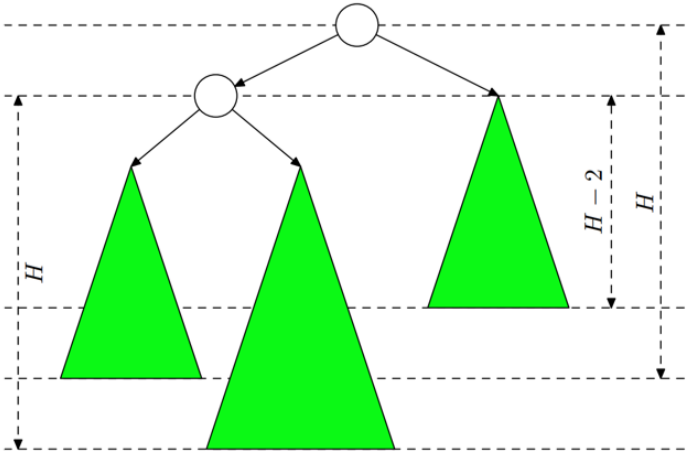
# AVL fák: Műveletek



## **LR** beszúrás

**LR beszúrás**:

A *bal* oldali részfa *jobb* oldali részfájába szúrunk be.



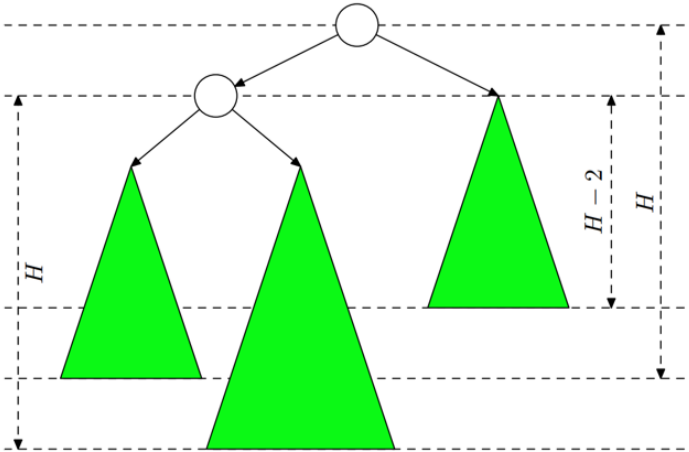
# AVL fák: Műveletek



## **LR** beszúrás

**LR beszúrás**:
A *bal* oldali részfa *jobb* oldali részfájába szúrunk be.

Az AVL tulajdonság **sérül**.



# AVL fák: Műveletek

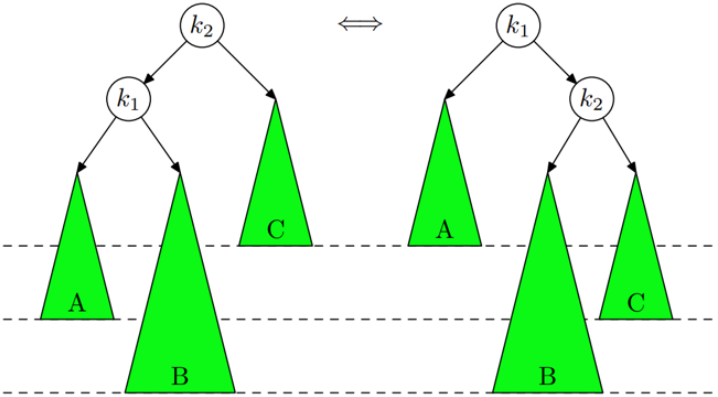


## LR beszúrás

**LR beszúrás**:
A *bal* oldali részfa *jobb* oldali részfájába szúrunk be.

Az AVL tulajdonság **sérül**.

Nem oldható meg egyetlen forgatással.

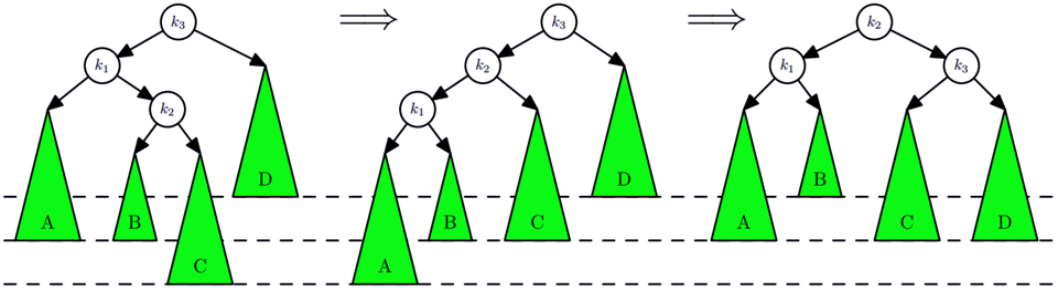


# AVL fák: Műveletek

## **LR** beszúrás

AVL tulajdonság visszaállítása

- **Megoldás**: két forgatás
  - I. eset:



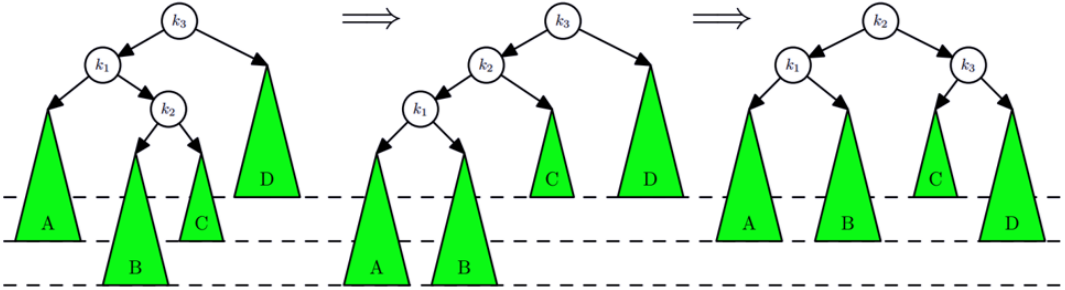
# AVL fák: Műveletek



## **LR** beszúrás

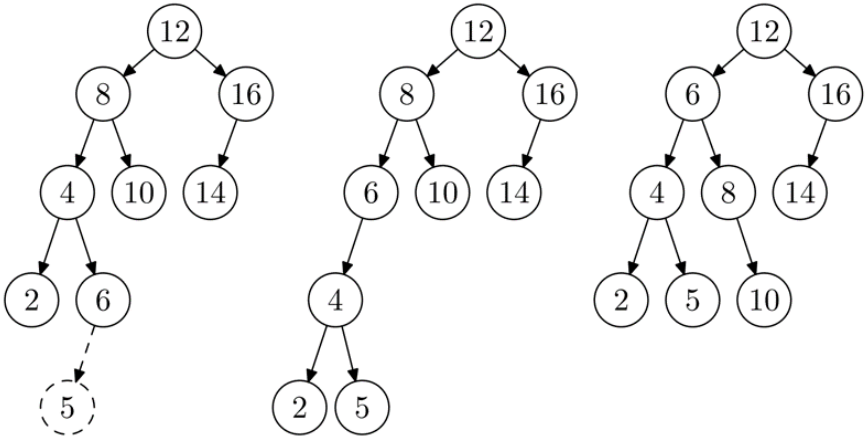
AVL tulajdonság visszaállítása

- **Megoldás**: két forgatás
  - II. eset:



# AVL fák: Műveletek

## Példa: **LR** beszúrás



# AVL fák: Műveletek



## Törlés

- Az **LL** és **LR** beszúráshoz hasonlóan létezik **RL** és **RR** beszúrás is.
- **RL** beszúrás: a *jobb* részfa *bal* részfájába szúrunk be.
- **RR** beszúrás: a *jobb* részfa *jobb* részfájába szúrunk be.
- Hasonlóan járunk el mint az **LL** és **LR** beszúrás esetén

# AVL fák: Műveletek



## Törlés

Egy AVL fából ugyanúgy törlünk mint egy keresőfából.

A törlést követően a következő *esetek* fordulhatnak elő:

- A fa továbbra is **kiegyensúlyozott**: ebben az esetben nincs további teendő
- A fa **elveszíti kiegyensúlyozottságát**:
  - A törlés helyétől a gyökér felé haladva megkeressük az első olyan elemet, amelyhez mint gyökérelemhez tartozó részfa nem kiegyensúlyozott.
  - A beszúrásnál ismertetett esetek közül a megfelelőt alkalmazva kiegyensúlyozzuk.
  - Ezt a lépést a gyökér felé haladva addig ismételjük, amíg a teljes fa kiegyensúlyozott nem lesz.

# AVL fák

Adatszerkezetek
Vekov Géza
Bináris keresőfa
Adatok
Műveletek
Tökéletesen egyensúlyozott
Kiegyensúlyozott bináris fák
AVL fák
Műveletek
Forgatás

## Mikor használjuk?

- Gyors műveletek rendezett adathalmaz esetén
- Gyors keresés

## Hátrányok

- Gyakori módosítás esetén gyakran kell kiegyensúlyoznunk a fát
- A kiegyensúlyozás költséges és bonyolult

## Hasonló adatszerkezet

- például a piros-fekete fa