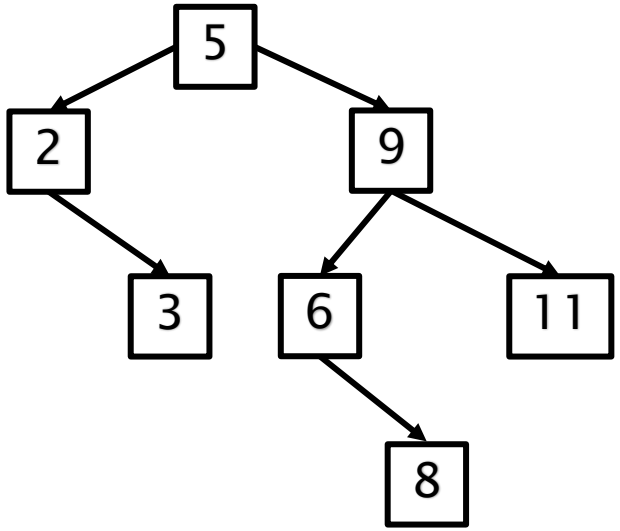
Bináris keresőfa, gyakoriságfa adatszerkezetek

Dr. Szeghalmy Szilvia
Debreceni Egyetem, Informatikai Kar

Bináris keresőfa

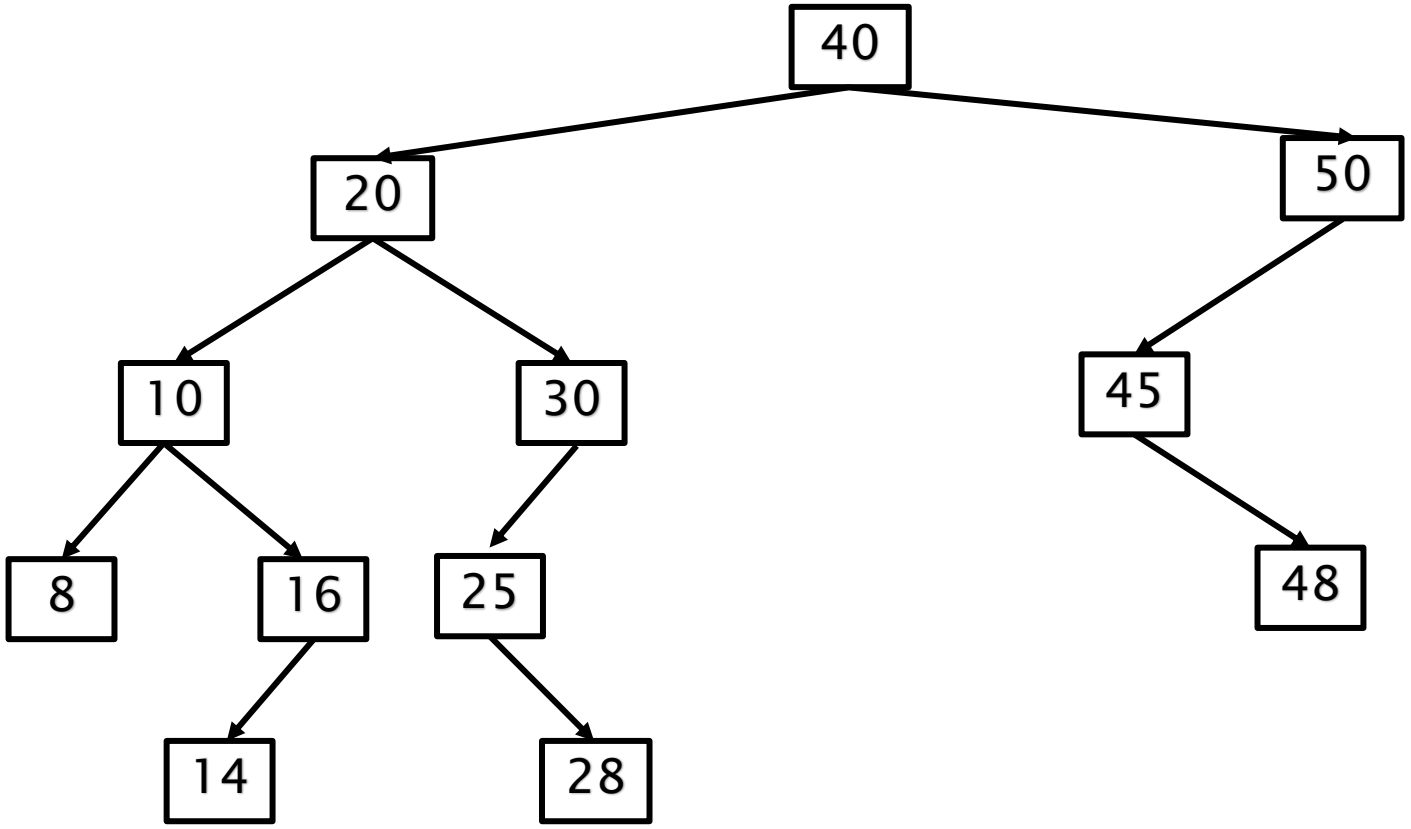
- Az elemei kulcsot (egyedi érték az adatszerkezetben) tartalmaznak
- Olyan binárisfa, melynek bármely részfájára igaz, hogy a gyökér baloldali részfájában csak a gyökérben tárolt kulcstól kisebb, a jobb oldali részfájában csak a gyökérben tárolt kulcstól nagyobb kulccsal rendelkező elemek szerepelnek.
- Pl.:



Keresőfa felépítése

- Az elemeket érkezési sorrendben szúrjuk be a fába, levélelemként.
- http://btv.melezinek.cz/binary-search-tree.html

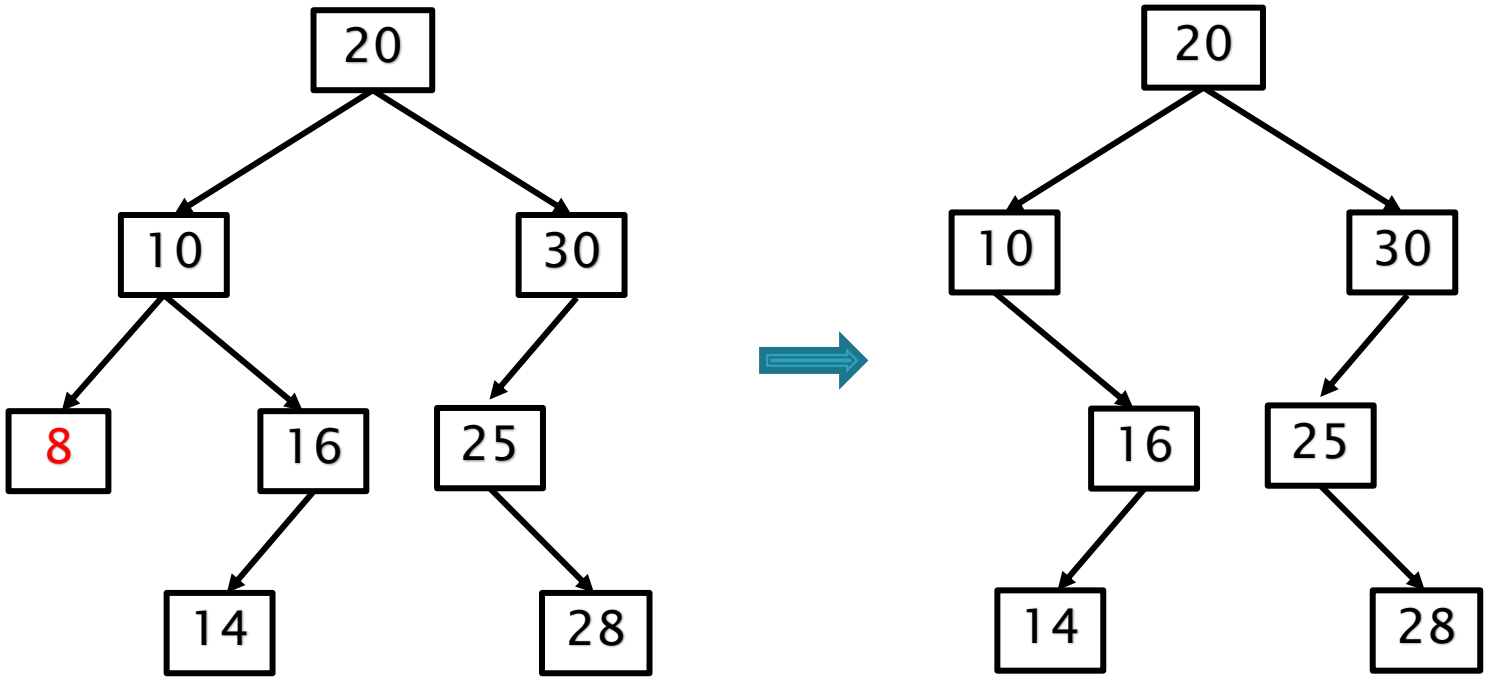
Pl.: 40, 50, 20, 10, 30, 8, 16, 14, 45, 48, 40, 25, 28



Keresőfa: törlés

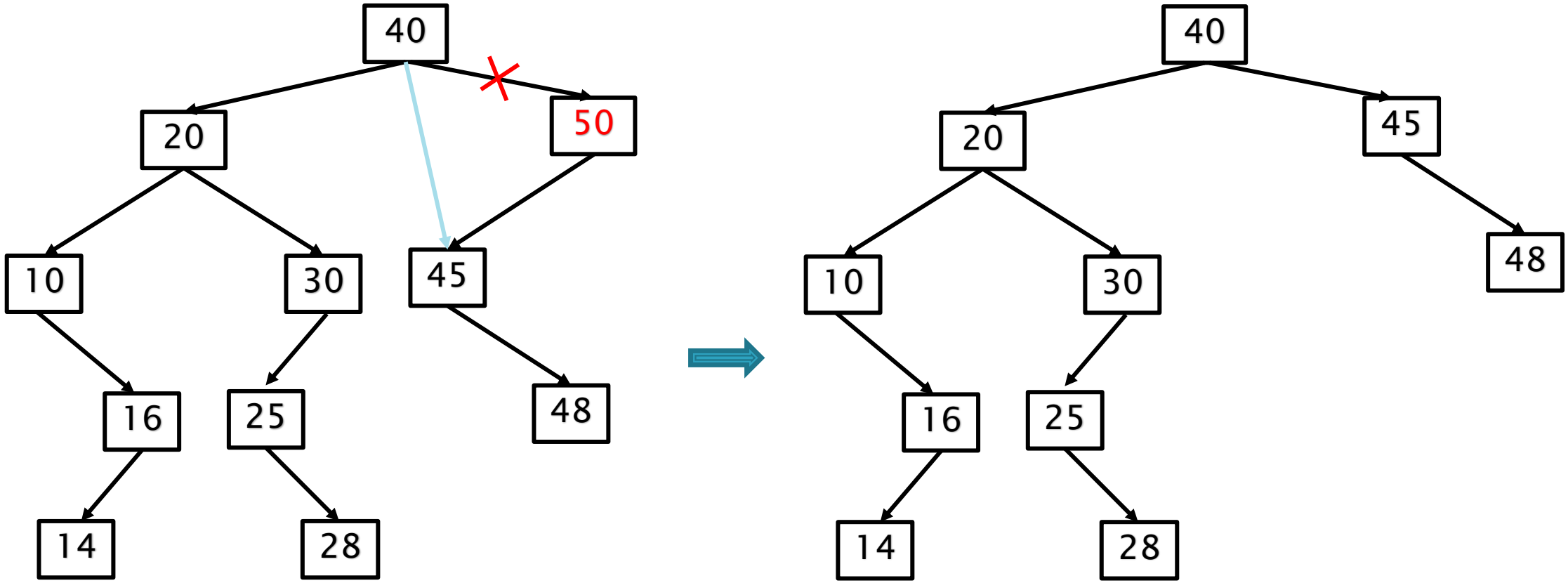
- Levél törlése:
  - felszabadítjuk a memóriát
  - a szülő elem adott oldali mutatóját None-ra állítjuk

- Pl. 8-as törlése



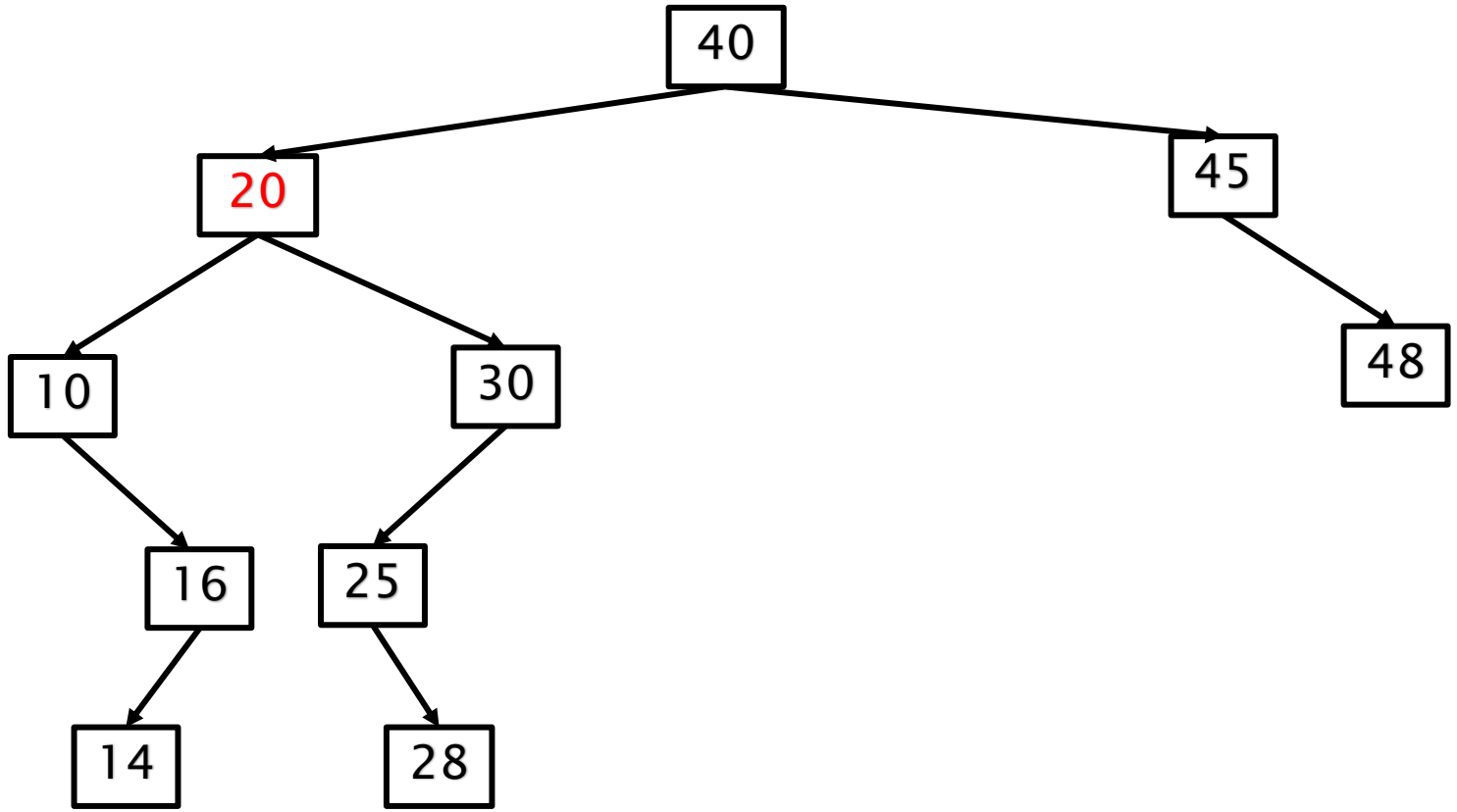
Keresőfa: törlés

- Egy gyerekkel rendelkező elem törlése **átláncolással** történik
  - a szülő elem törlendőre mutató mutatóját a törlendő gyerekelemére állítjuk
  - felszabadítjuk a memóriaterületet (törlendő)
- Pl.: 50-es törlése



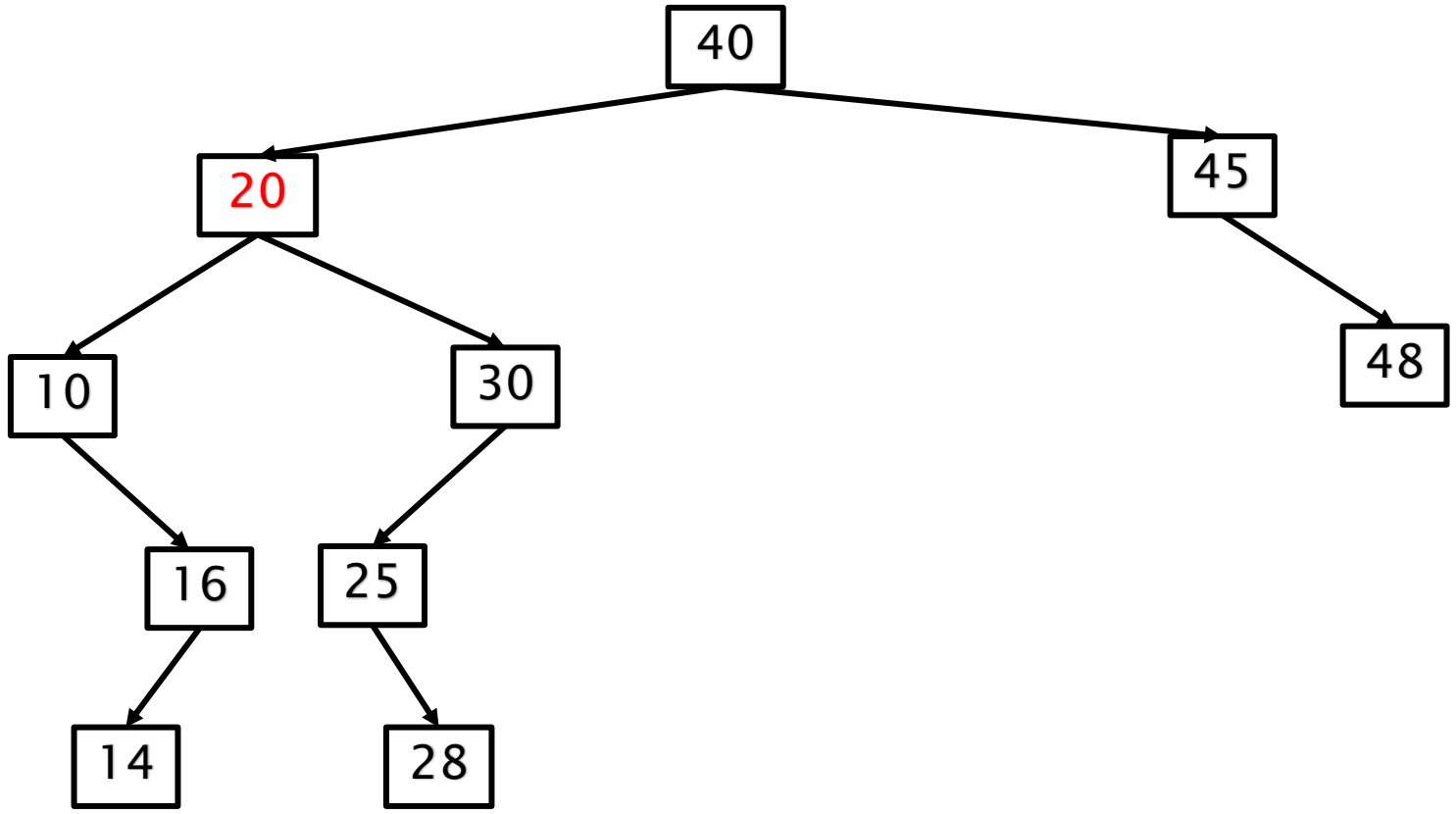
Keresőfa: törlés

- Két gyerekkel rendelkező elem törlése cserével történik
  - írjuk felül a törlendőt a baloldali részfájának legjobboldalibb (legnagyobb kulcsú) elemével
    vagy a jobboldali részfa legbaloldalibb (legkisebb kulcsú) elemével
  - töröljük a felülírásra használt elemet az előző két módszer valamelyikével



Keresőfa: törlés

- 20-as törlése a baloldali részfájának a legjobboldalibb elemével:

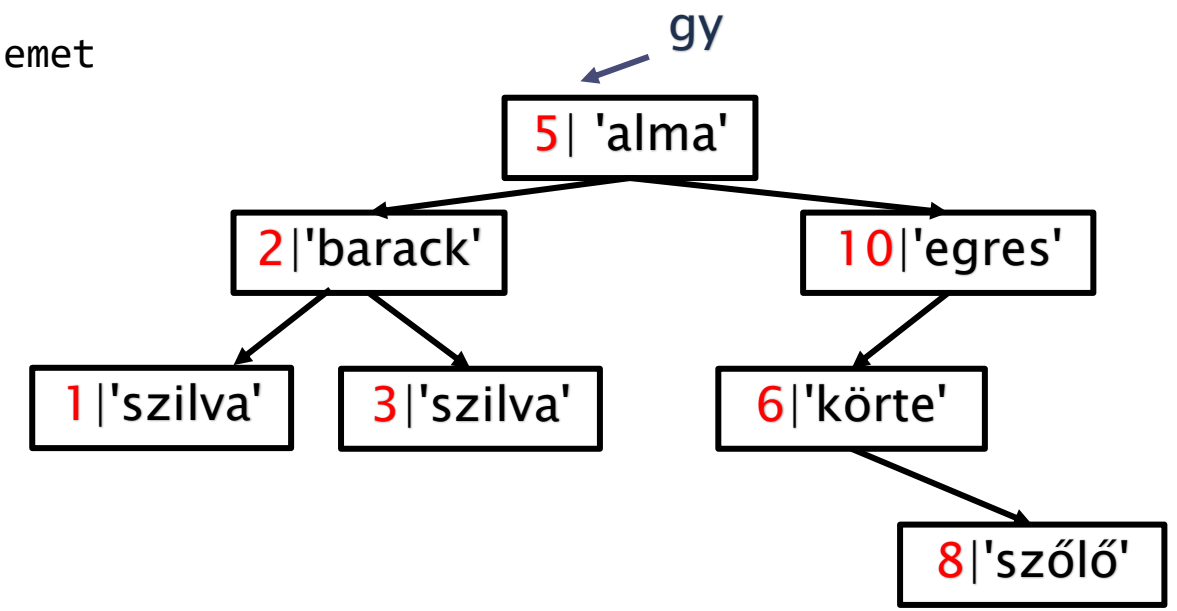


Keresőfa felépítése

- Írj függvényt, mely beszúrja a gyökerével adott keresőfába a paraméterben kapott kulcsot és adatot (amennyiben lehetséges).

```
# LÉTREHOZÁS:
class Faelem:
   def __init__(self,k, a):
       self.kulcs = k
       self.adat = a
       self.bal = None
       self.jobb = None

gy = None
```

```
def beszur(gy, kulcs, adat): #gy: Faelem, kulcs:int, adat:str
    if             :
       return Faelem(kulcs, adat)  # létrehozzuk az elemet
    if                          :  # kulcs alapján
        gy.bal = beszur(gy.bal, kulcs, adat)
    elif                        :
        gy.jobb = beszur(gy.jobb, kulcs, adat)
    # egyenlőség esetén nem csinálunk semmit!
    return gy
```

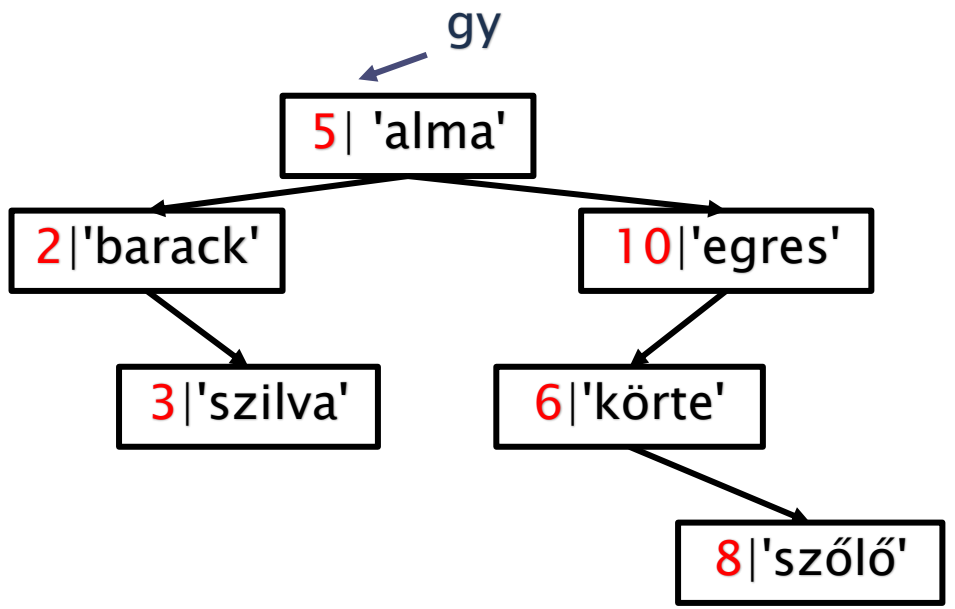


Keresés keresőfában

```
# LÉTREHOZÁS:
class Faelem:
   def __init__(self,k, a):
       self.kulcs = k
       self.adat = a
       self.bal = None
       self.jobb = None

gy = None
```

- Írj függvényt, mely megkeresi a gyökerével adott keresőfában a paraméterben kapott kulcsú elemet és visszadja azt.
- Ha a kulcs nem szerepel, None legyen a visszatérési érték.

```
def keres(gy, kulcs): # gy: Faelem, kulcs: int
```



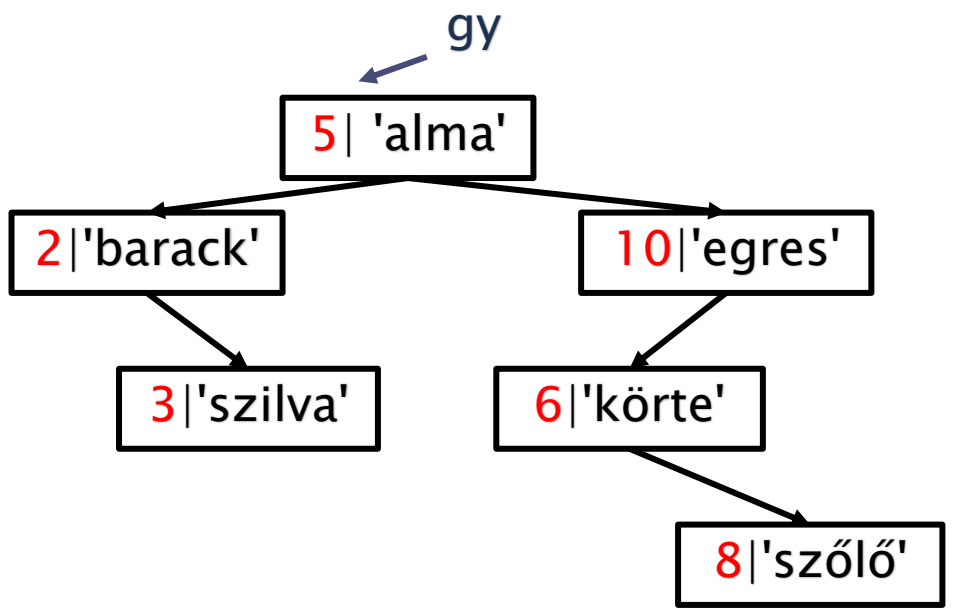
- Mely elemeket érintjük a 6-os elem keresése során?
- Mely elemeket érintenénk egy preorder elven működő keresővel?

Keresőfa == rendezési fa

- Írd függvényt, mely kiírja a gyökerével adott fa kulcsait **növekvő** sorrendben.

```
def rendezve_kiir(gy): # gy: Faelem, kulcs: int
```

```
# LÉTREHOZÁS:
class Faelem:
   def __init__(self,k, a):
       self.kulcs = k
       self.adat = a
       self.bal = None
       self.jobb = None

gy = None
```



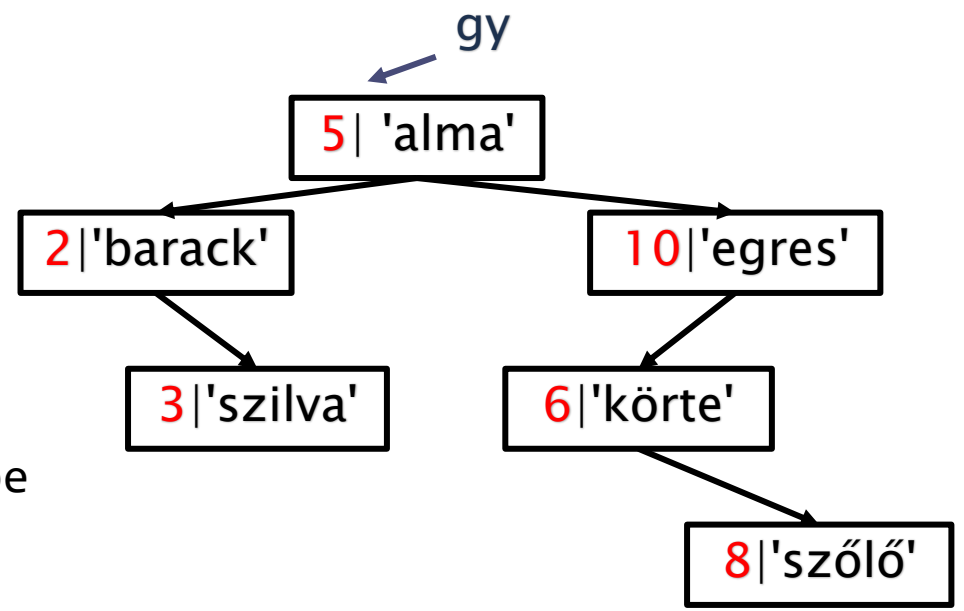
Az ábrán látott fára: 2 3 5 6 8 10

Keresőfa == rendezési fa

- Másoljuk át az elemeket egy tömbbe. Tf. van elég hely a tömbben.

```
def lekepez(gy, tomb, elemszam = 0):
    if not gy:
        return elemszam

    # hány elemet pakoltunk be a bal oldalra
    elemszam = lekepez(gy.bal, tomb, elemszam)

    # a következő helyre betesszük az elemet
    tomb[elemszam] = gy

    # átadjuk, hogy hol tartunk a tömb feltöltésében
    # +1 azért, mert az előbb raktunk egy elemet a tömbbe
    elemszam = lekepez(gy.jobb, tomb, elemszam + 1 )

    return elemszam
```

```
# LÉTREHOZÁS:
class Faelem:
    def __init__(self,k, a):
        self.kulcs = k
        self.adat = a
        self.bal = None
        self.jobb = None

gy = None
```

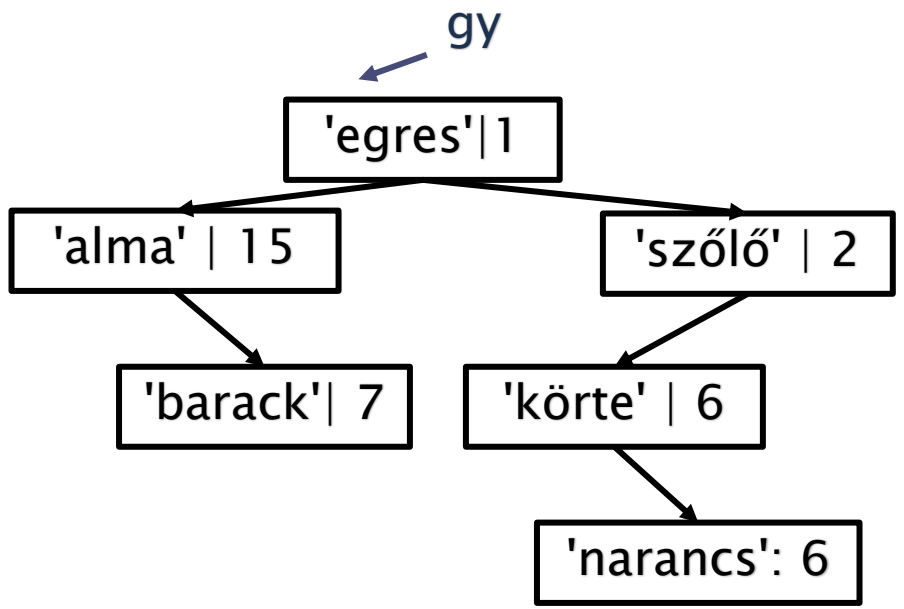


Gyakoriságfa

```
# LÉTREHOZÁS:
class Faelem:
   def __init__(self, gyumi):
       self.gyumolcs = gyumi
       self.gyakorisag = 1
       self.bal = None
       self.jobb = None

gy = None
```

- Minden elemet egyszer tárolunk a fában a keresőfa szabályai szerint, de az adott csúcsban tároljuk az elem multiplicitását is.
- Pl. vásárlók rendeléseiben szereplő gyümölcsök neveit tároljuk el a fában => megtudjuk, hogy mely gyümölcsök mennyire keresettek.
- A gyümölcsök neve egyedi => most a *nev* mező a kulcs a keresőfában.

```
def beszur_gyfa(gy, gyumi):
    if not gy:      # üresfa, létrehozzuk az elemet
         return Faelem(gyumi)

    # létezik a fa, keressük meg az elem helyét
    if                      :
         gy.bal =
    elif                    :
         gy.jobb =
    else:  # az elem szerepel => növeljük a gyakoriságot

    return gy
```

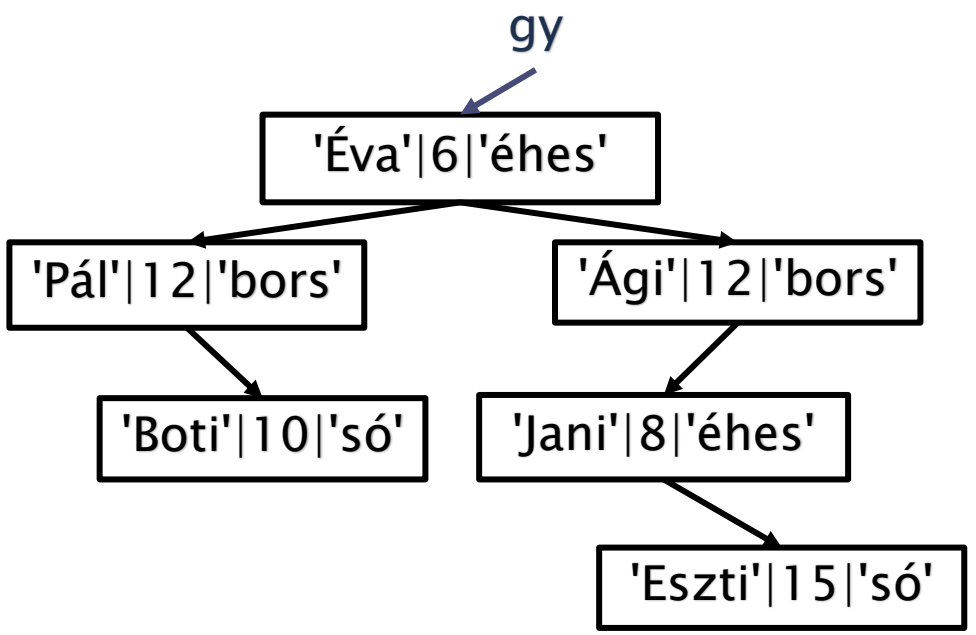


Binárisfa gyakorlás

- Egy gombócevő verseny résztvevőinek nevét (egyedi) és az általuk megevett gombócok számát és a csapatuk nevét egy bináris fában tároljuk.
- Írj függvényt, mely **megjeleníti**, hogy ki hány gombócot evett meg.

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def eredmeny(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return
    print(  ?   ,  ?   )  # a gyökér feldolgozása
    ???          # a bal oldali részfa feldolgozása
    ???          # a jobb oldali részfa feldolgozása
```



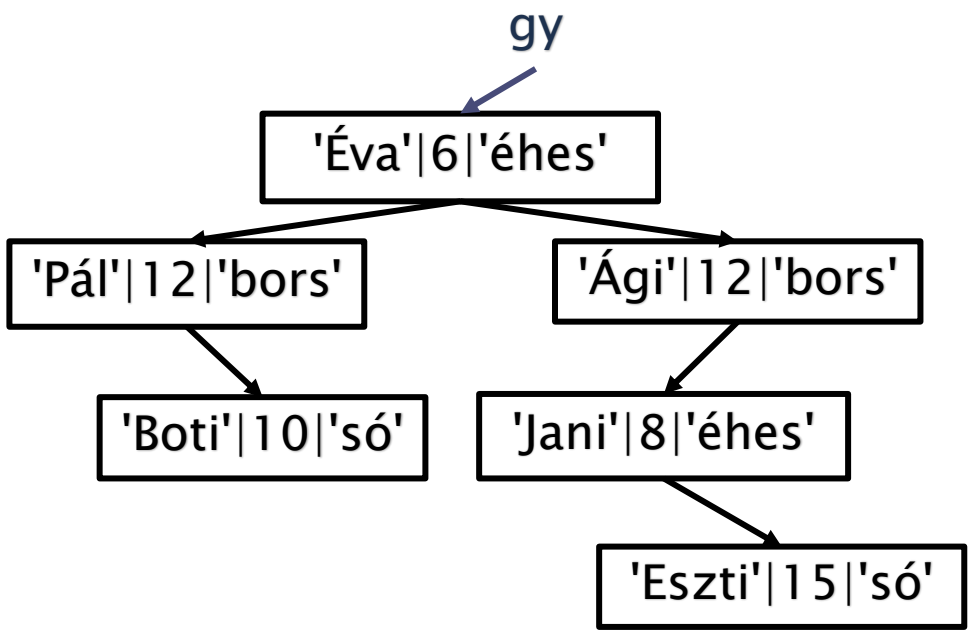
Binárisfa gyakorlás

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

- A verseny közben folyamatosan frissíteni akarjuk az eredményeket:

  írj egy függvényt, mely **megnöveli** eggyel a gombócok számát a paraméterben átadott nevű versenyzőnél.

```
def evett_egyet(gy, nev): # gy:Faelem, nev:str
   if not gy:       # üres fa esete
       return
```

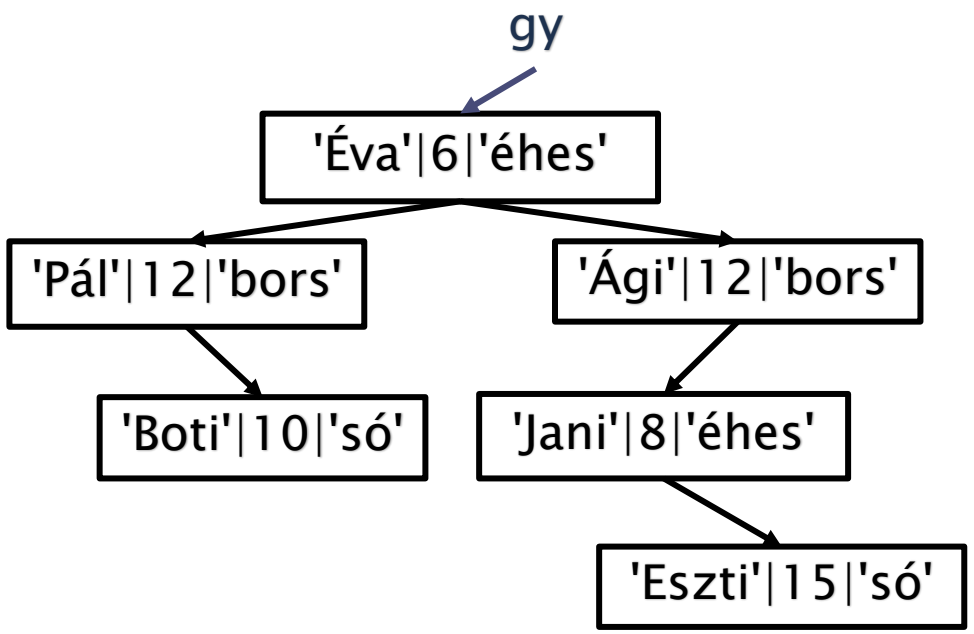


Binárisfa gyakorlás

- Írj függvényt, mely **megjeleníti** egy paraméterben adott csapat tagjainak eredményeit (név, gombócok száma).

```
def csapattag_eredmeny(gy, csapat): # gy:Faelem, nev:str
    # üres (rész)fa esete
    if not gy:
        return
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
      self.nev = n
      self.gomboc = 0
      self.csapat = c
      self.bal = None
      self.jobb = None

gy = None
```

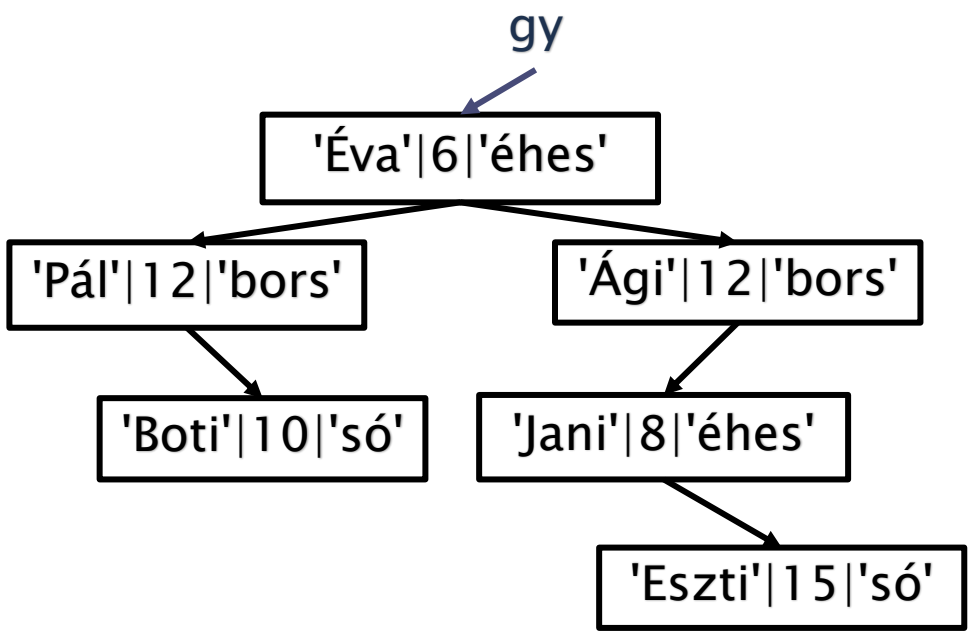


Binárisfa gyakorlás

- Az "éhes" csapat csúnyán lemaradt a versenyben, ezért megvesztegették Eszti-t, hogy igazoljon át hozzájuk.
- A vesztegetés sikerrel járt, írj függvényt, mely Eszti csapatát 'éhes'-re cseréli.

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

```
def atigazol(gy): # gy:Faelem, nev:str
   if not gy:
       return
```



Áttekintés

- Közös pontok az előző feladatokban
  - Az elemek bejárására építettünk
  - Nem volt visszatérési értéke a függvényeknek

- A megoldásaink sablonosak voltak:

```
def fv(gyökér és esetleges egyéb paraméterek):
    - üres (rész)fa kezelése
    - gyökérelem feldolgozása (a gyökeret mindig megkapunk paraméterben)
    - a függvényt meghívjuk a gyökér bal oldali gyerekére
    - a függvényt meghívjuk a gyökér jobb oldali gyerekére
```

Megj.: A különböző részek végrehajtását néha feltételhez kötjük

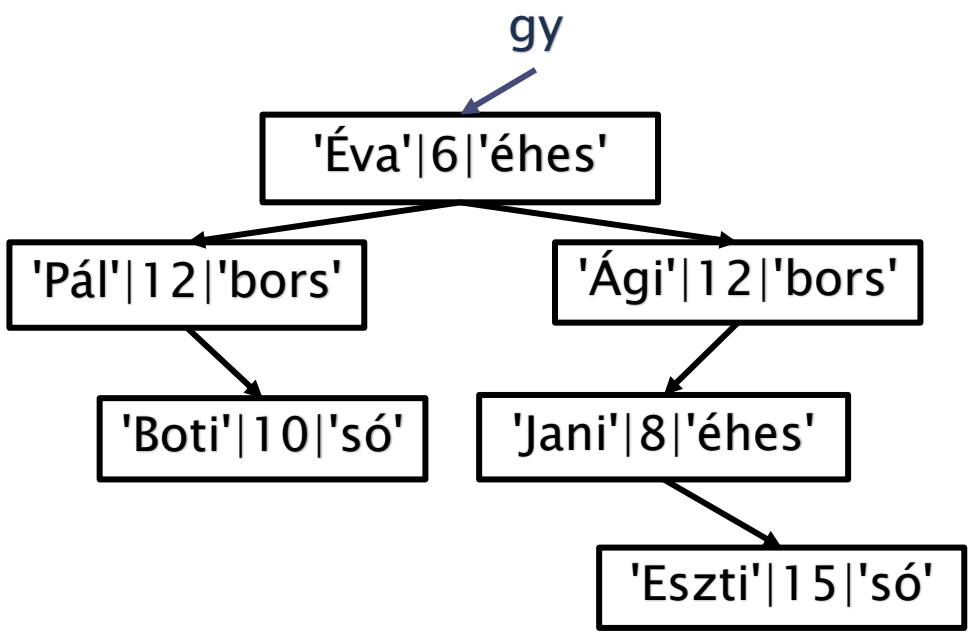
pl. ha egyetlen értéket kell módosítani és azt a részfa gyökerében megtaláltuk, akkor már nem szükséges meghívni a függvényt a bal és jobb oldali részfákra.

Binárisfa gyakorlás

- Írj függvényt, mely **visszaadja**, hogy hány gombócot ettek meg összesen a résztvevők.

```
def ossz_gomboc(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return 0

    # a gyökében lévő versenyző ennyi gombócot evett
    ossz_gy = ???

    # a bal és jobb oldali részfákra kapott érték
    ossz_b = ???
    ossz_j = ???

    return ???
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
      self.nev = n
      self.gomboc = 0
      self.csapat = c
      self.bal = None
      self.jobb = None

gy = None
```

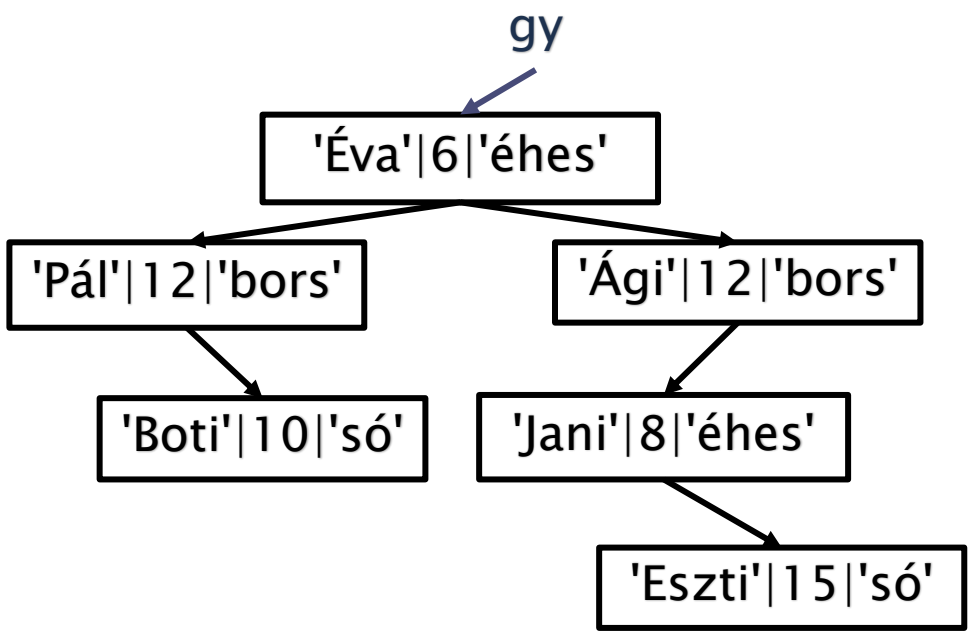


Binárisfa gyakorlás

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

- Ugyanaz rövidebben:

```
def ossz_gomboc(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:
        return 0
    return gy.gomboc + ossz_gomboc(gy.bal) + ossz_gomboc(gy.jobb)
```

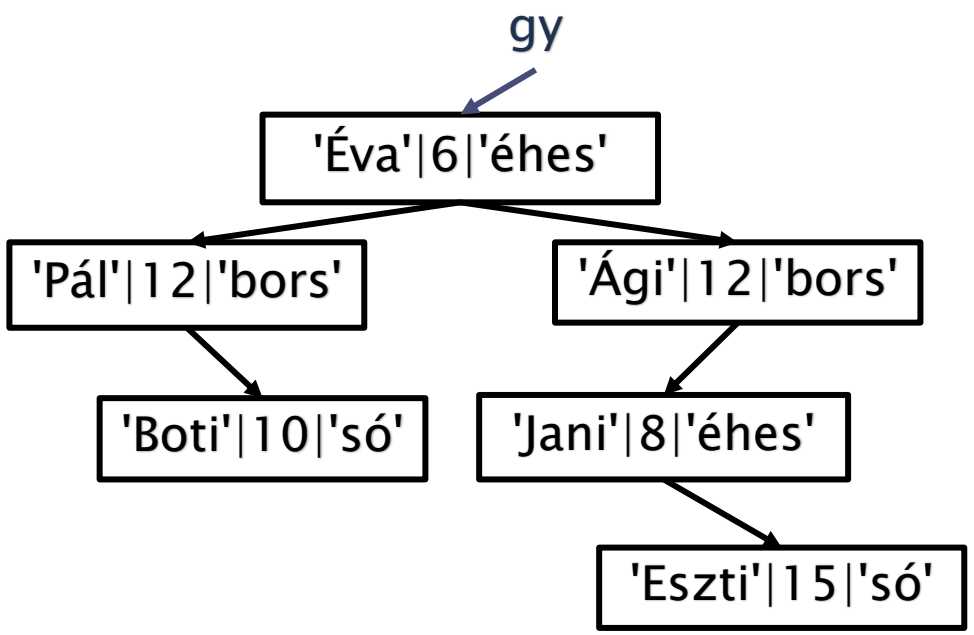


Binárisfa gyakorlás

- Írj függvényt, mely **visszaadja**, hogy hány résztvevő van.
  /a fában lévő elemek száma/

```
def resztvevok(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return 0
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

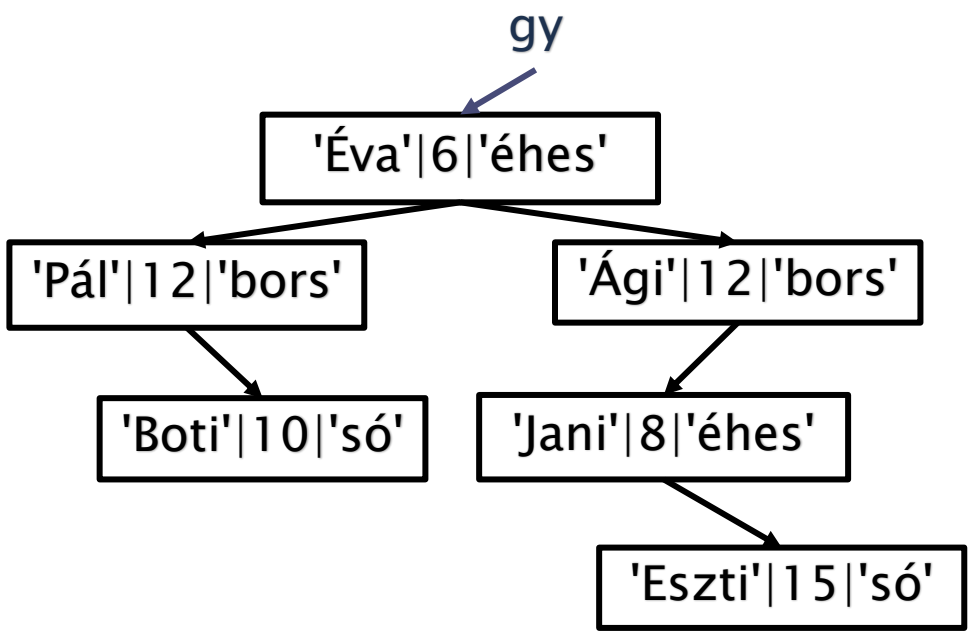


Binárisfa gyakorlás

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

- Ugyanaz rövidebben:

```
def resztvevok(gy):  gy a (rész)fa gyökere, Faelem típusú
    if not gy:       # üres fa esete
        return 0

    return 1 + resztvevok(gy.bal)+resztvevok(gy.jobb)
```

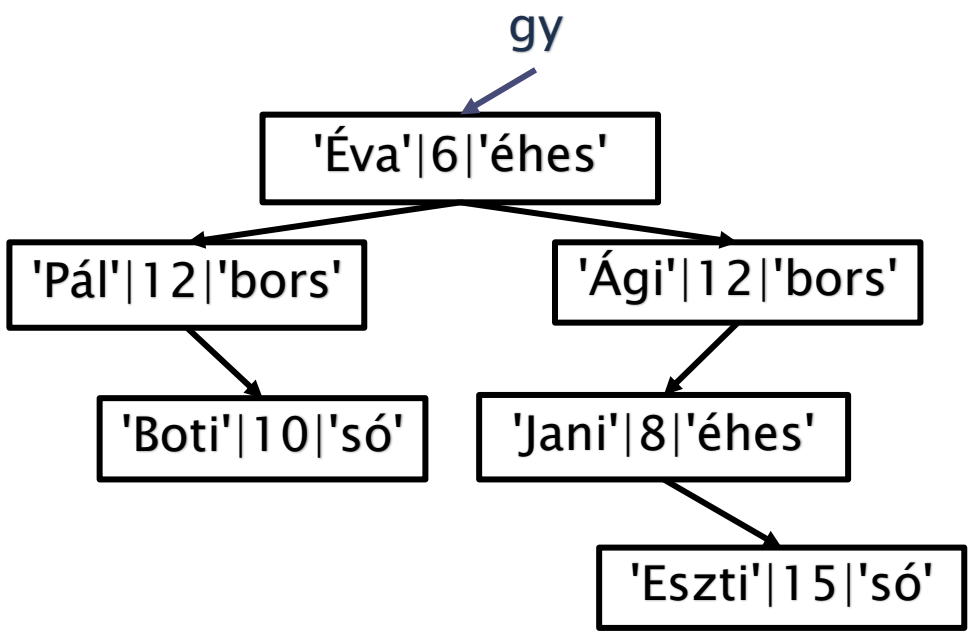


Binárisfa gyakorlás

- Írj függvényt, mely **visszaadja**, hogy hány gombócot evett meg összesen a *csapat* paraméterben átadott nevű csapat.

```
def csapat_eredmeny(gy, csapat):  # gy: Faelem típusú
    if not gy:      # üres fa esete
        return 0
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

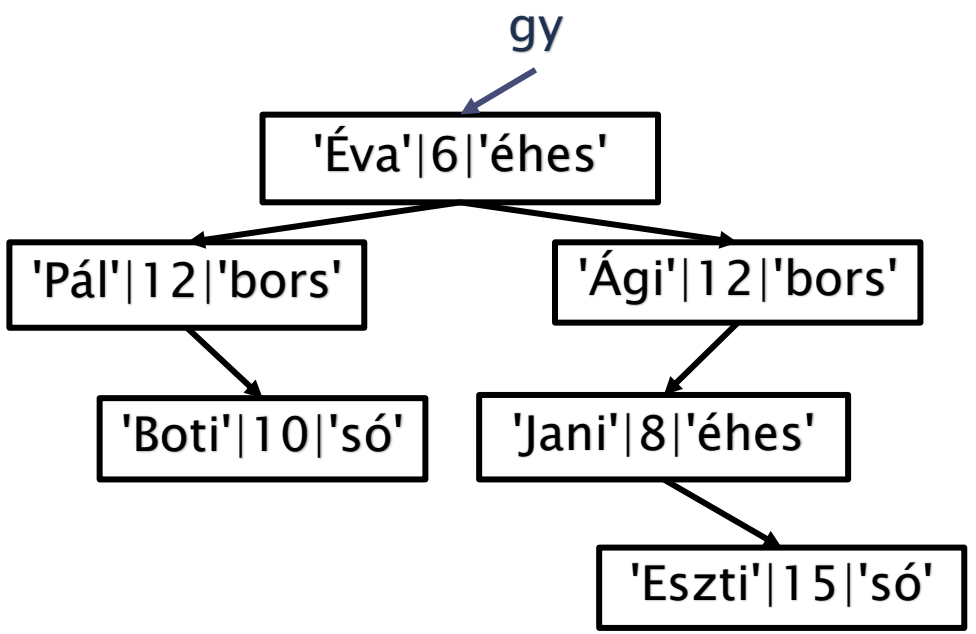


Binárisfa gyakorlás

- Ugyanaz rövidebben

```
def csapat_eredmeny(gy, csapat):  # gy: Faelem típusú
    if not gy:       # üres fa esete
        return 0

    return csapat_eredmeny(gy.bal, csapat) +
           csapat_eredmeny(gy.jobb, csapat) +
           (gy.gomboc if gy.csapat == csapat else 0)
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```

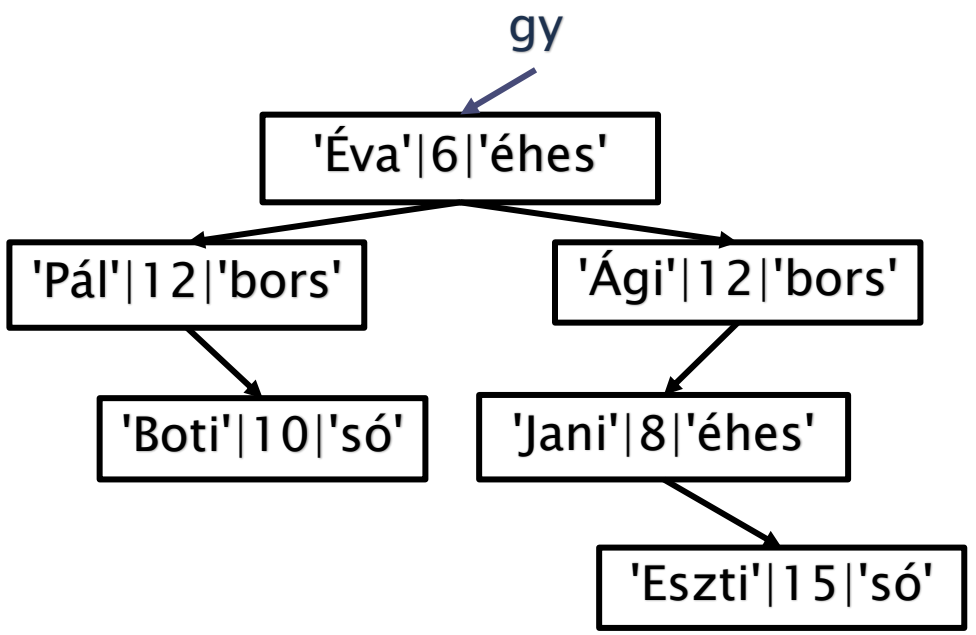


Binárisfa gyakorlás

- Írj függvényt mely visszaadja, hogy Eszti hány gombócot evett meg.
- Ha Eszti nincs a fában, akkor -1 értéket adj vissza.

```
def megevett(gy):  # gy: Faelem típusú
    if not gy:      # üres fa esete
        ???
```

```
# LÉTREHOZÁS:
class Faelem:
  def __init__(self, n, c):
     self.nev = n
     self.gomboc = 0
     self.csapat = c
     self.bal = None
     self.jobb = None

gy = None
```



Áttekintés

- Közös pontok az előző feladatokban
  - Az elemek bejárására építettünk
  - Volt visszatérési értéke a függvényeknek
  - A rekurzív hívásnál MINDIG felhasználtuk a kapott visszatérési értéket

- A megoldásaink "sablonja"

```
def fv(gyökér és esetleges egyéb paraméterek):
    - üres (rész)fa esetén visszaadjuk a megfelelő értéket
    - meghatározzuk a gyökérre vonatkozó értéket (mentjük az értéket)
    - a függvényt meghívjuk a gyökér bal oldali gyerekére (mentjük az értéket)
    - a függvényt meghívjuk a gyökér jobb oldali gyerekére (mentjük az értéket)
    - a három értéket felhasználva meghatározzuk és visszaadjuk az eredményt
```

Megjegyzések

- Az értékek változóba mentése helyett gyakran egy kifejezésben összesíthetjük a gyökérre, bal- és jobb oldali részfára vonatkozó eredményt, így tömörebb a kód.

- Az "összesítés" alatt az érték felhasználását értjük, ami teljesen problémafüggő:
  lehet, hogy összegezzük, szorozzuk, a legnagyobbat, legkisebbet vesszük a három közül, stb.

- A bal és jobb oldali eredmények lekérdezését is feltételhez köthetjük.

  Lásd: keresés

  pl.: ha már megtaláltunk az értéket, a gyökérnél, akkor egyik részfában sem keresünk tovább

  ha megtaláltuk az elemet a bal oldali részfában, akkor nem keresünk a jobb oldalon

Természetesen vannak ennél jóval összetettebb problémák is, melyeket nem lehet ilyen sablonosan megoldani.