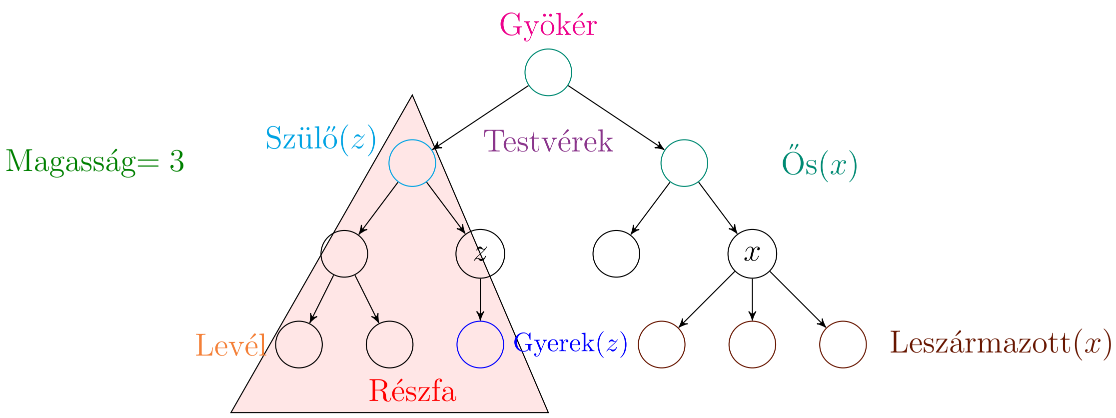
Algoritmusok és adatszerkezetek gyakorlat – 07

Keresőfák

Fák

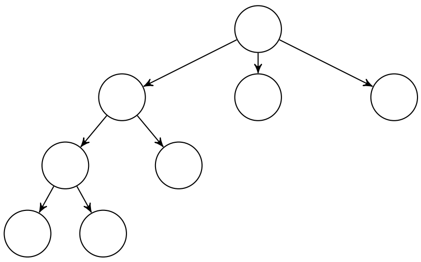
- **Fa:** összefüggő, körmentes gráf, melyre igaz, hogy:
  - (Általában) egy gyökér csúcsa van, melynek 0 vagy több részfája van
  - Pontosan egy út vezet bármely két csúcsa között
  - A gyökéren kívül minden csúcsnak pontosan egy szülője van
- **Szülő(i):** az a csúcs, amely közvetlenül az i felett van
- **Gyerek(i):** az i csúcs alatt közvetlen lévő csúcsok
- **Testvérek:** ugyanannak a csúcsnak a gyerekei
- **Gyökér:** az egyetlen csúcs, aminek nincsen szülője
- **Levél:** olyan csúcs, amelynek nincs gyereke
- **Magasság:** a gyökértől bármely levélbe vezető leghosszabb út
- **Részfa(n):** az a fa, amelynek a gyökere az n
- **Ős(n):** minden csúcs az n-től a gyökérig vezető úton
- **Leszármazott(n):** minden csúcs, amely az n gyökerű részfában van



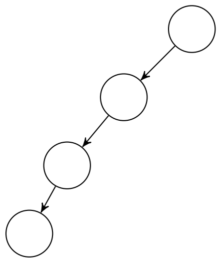
Bináris fák

- **Bináris fa:** minden csúcsnak legfeljebb 2 gyereke van
- **Teljes (full) bináris fa:** olyan bináris fa, ahol minden szint teljesen ki van töltve
- **Majdnem teljes (complete) bináris fa:** olyan bináris fa, ahol maximum a legalsó szint nincs teljesen kitöltve, csak balról jobbra haladva kitöltött néhány elemig
- **Kiegyensúlyozott bináris fa:** olyan fa, ahol minden csúcs gyerekeinek részfáinak magassága maximum eggyel tér el.

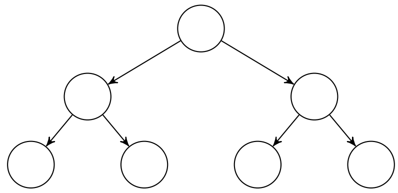
1. Feladat Döntsük el a következő fákról, hogy mely tulajdonság igaz rájuk a következők közül: fa, bináris fa, teljes bináris fa, majdnem teljes bináris fa, kiegyensúlyozott bináris fa!



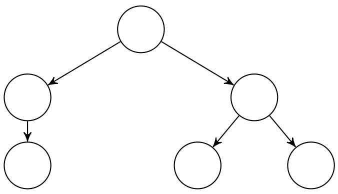
(a)



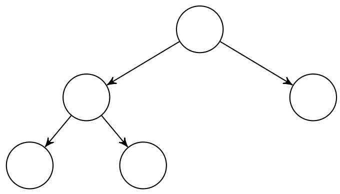
(b)



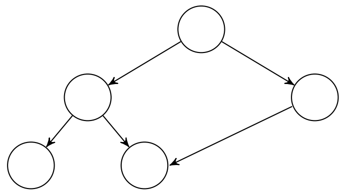
(c)



(d)



(e)



(f)

Megoldás

(a) fa
(b) fa, bináris fa
(c) fa, bináris fa, majdnem teljes bináris fa, teljes bináris fa, kiegyensúlyozott bináris fa
(d) fa, bináris fa, kiegyensúlyozott bináris fa
(e) fa, bináris fa, majdnem teljes bináris fa, kiegyensúlyozott bináris fa
(f) egyik sem, mert sérti a ,,minden csúcsnak csak egy szülője van” feltételt

Bináris keresőfák (BST)

A bináris keresőfa egy olyan adatszerkezet, amely olyan elemeket tárol, melyeknek kulcsa egy tejesen rendezett univerzumból való (pl. egészek). Feltesszük, hogy minden elem kulcsa egyedi. Egy bináris keresőfa a következő műveleteket támogatja:

- search(i): visszaadja azt az elemet, aminek a kulcsa i
- insert(i): beszúrja az i kulcsú elemet a fába (ha még nem volt benne)
- delete(i): törli az i kulcsú elemet a fából, ha az létezik

A bináris keresőfa **legfontosabb tulajdonsága**, hogy minden x csúcsra a bal részfájában lévő összes elem kisebb, mint az x kulcsa, míg a jobb részfájában lévő összes elem nagyobb, mint az x kulcsa.

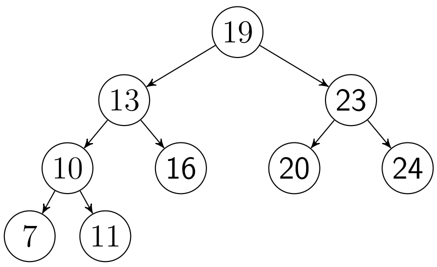
Keresés bináris keresőfában

```
search(x, i){
if(key(x) == i)  return x
else if (i < key(x))
  if(left(x) == NIL) return x
  else return search(left(x), i)
else if (i > key(x))
  if(right(x) == NIL) return x
  else return search(right(x), i)}
```

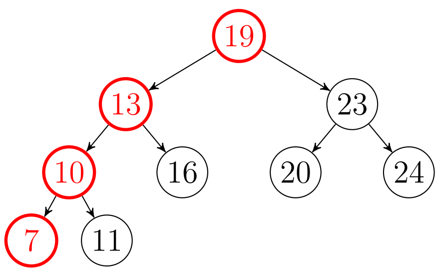
Keressük az x gyökerű részfában az i kulcsú elemet.

1. Ha a gyökér az, adjuk vissza
2. Ha a i kisebb, mint a gyökér kulcsa, keressük a bal részfájában, ha van neki
3. Ha a i nagyobb, mint a gyökér kulcsa, keressük a jobb részfájában, ha van neki
4. Ha a részfa, amire lépnénk x-ről, nem létezik, adjuk vissza az x-et

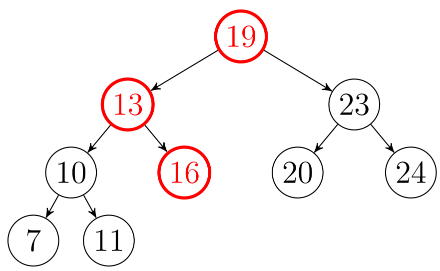
2. Feladat Keressük meg a következő keresőfában az alábbi elemeket: $5, 16, 24$.



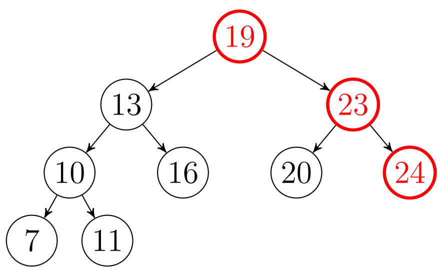
Megoldás



(a) search(5)



(b) search(16)



(c) search(24)

Beszúrás bináris keresőfába

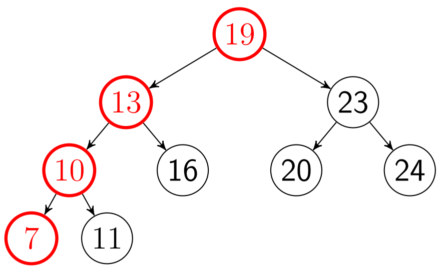
```
insert(i){
  x = search(i)
  if(key(x)==i) return
  y = new node()
  key(y) = i
  left(y)= NIL
  right(y)= NIL
  p(y)= x
  if (i < key(x))
    left(x) = y;
  else
    right(x) = y}
```

Az i csúcs beszúrásának lépései (továbbra is feltesszük, hogy a fában a kulcsok egyediek, így nem szúrhatjuk be kétszer)

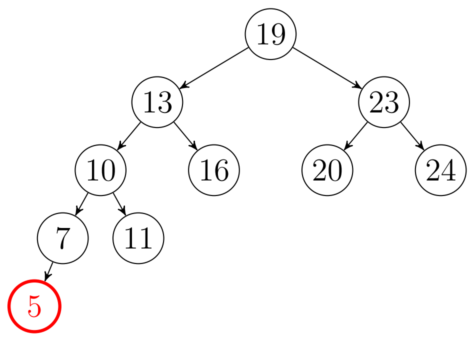
- Keressünk rá az i-re, ez ha még nincs benne a fában, vissza fogja adni azt a csúcsot, ami alá be kell szúrni az i-t (ő lesz a szülője).
- Hozzunk létre egy új csúcsot (y) és szúrjuk be:
  - Ha i kisebb, mint az x kulcsa, akkor bal gyereknek
  - Ha i nagyobb, mint az x kulcsa, akkor jobb gyereknek

3. Feladat Szúrjuk be az előző feladatban látott keresőfába az 5 és 22 értékeket.

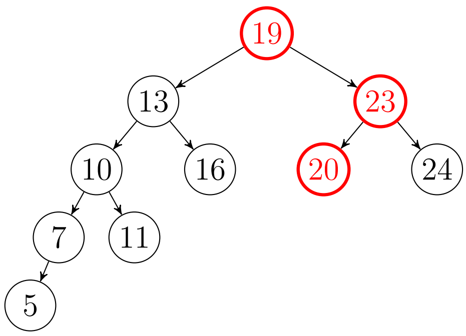
Megoldás



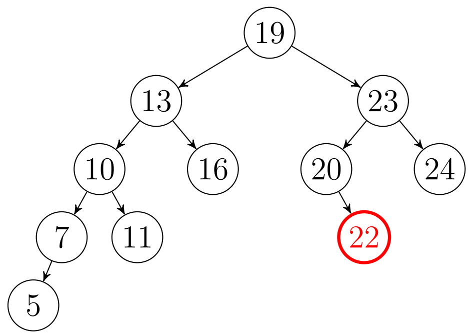
(a) search(5)



(b) new node()



(c) search(22)



(d) new node()

Törlés bináris keresőfából

Három esetet különböztetünk meg az x csúcs törlésekor:

1. Ha x-nek nincs gyereke, töröljük, a szülő rá mutató pointerét NIL-re cseréljük.

2. Ha x-nek pontosan egy gyereke van c, mindegy, hogy bal vagy jobb gyerek volt, felemeljük az x helyére (x szülőjének gyereke c, c szülője az x szülője lesz).

3. Ha az x-nek két gyereke van (c_1 bal és c_2 jobb gyerek), megkeressük az x közvetlen rákövetkezőjét z-t, és őt tesszük x helyére a fában.

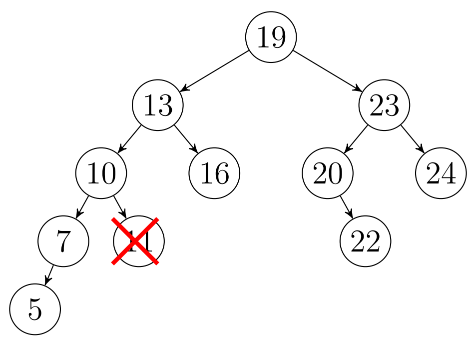
   - Ebben az esetben jegyezzük meg, hogy mivel z a c_2 gyökerű részfában van így egyszerűen rákereshetünk ($search(c2, key(x))$).
   - Viszont mivel z az x rákövetkezője, így biztosan nincs bal gyereke, viszont jobb gyereke még lehet.
   - Ha van jobb gyereke, állítsuk rá z szülőjének pointerét (és z gyerekének szülője legyen z szülője).
   - Cseréljük ki x-et z-vel és állítsuk be a megfelelő pointereket, majd töröljük x-et, mint az 1. esetben.

   (Egy másik megoldás, hogy x közvetlen megelőzőjét keressük meg és a fentieket tükrözve hajtjuk végre a cseréket.)

4. Feladat Töröljük az előző feladatban beszúrások után kapott fából a 11, 7 és 19 elmeket.

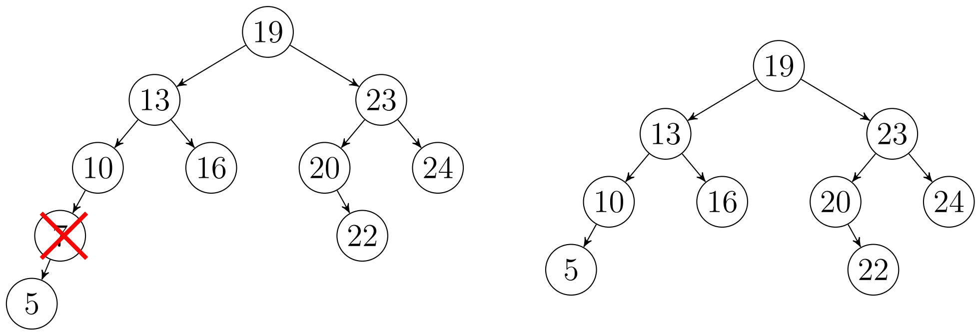
Megoldás

A 11-nek nincsen gyereke: 1. eset. Simán kitörölhetjük.



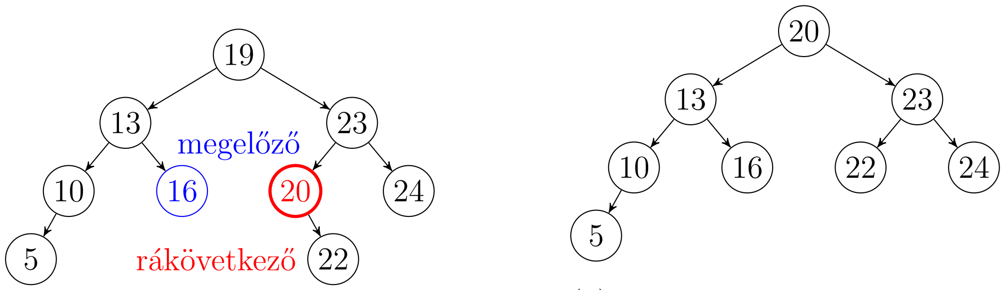
4. Ábra.: delete(11)

A 7-nek egy gyereke van: 2. eset. Rakjuk a helyére az 5-öt.



5. Ábra.: delete(7)

A 19-nek két gyereke van: 3. eset. Keressük meg a rákövetkezőjét.



(a) $search(23, 19)$

(b) 20 helyére tegyük a 22-t, 19 helyére a 20-at

6. Ábra.: delete(19)

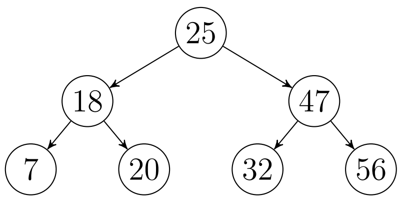
Megjegyzés: Egy helyes megoldás lett volna az is, ha a 19-et a 16-tal cseréljük ki.

Futásidő

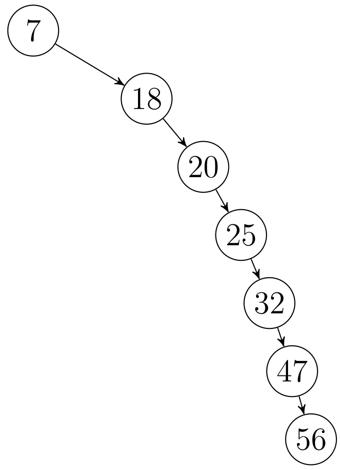
- Keresés: maximum annyit megyünk lefelé, amilyen magas a fa, tehát $O(magassag)$.
- Beszúrás és törlés: mindkettőben használjuk a keresést, a többi művelet (pointerek átállítása) konstans időben végezhető, így ez is $O(magassag)$

5. Feladat Határozzuk meg mi a minimum és maximum lehetséges magassága egy n elemű fának. Ehhez szúrjuk be egy fába a következő sorrendben a $25, 18, 47, 7, 20, 32, 56$ elemeket és egy másikba $7, 18, 20, 25, 32, 47, 56$ sorrendben.

Megoldás



(a) A $25, 18, 47, 7, 20, 32, 56$ sorrend fája



(b) A $7, 18, 20, 25, 32, 47, 56$ sorrend fája

Tehát egy bináris fa mérete akkor a legalacsonyabb, ha (majdnem) teljes a fa. Egy n-csúcsú teljes fa magassága pedig $\log n$, így egy keresés időigénye $O(\log n)$.

Legrosszabb esetben, ha például egy n elemű növekvő, vagy csökkenő sorozatot kell eltárolnunk egy bináris fában, akkor egy n hosszú láncot kapunk. Ekkor a keresés $O(n)$ ideig tart.

Megjegyzés: tehát érdekünk a fa magasságát minél alacsonyabban tartani, minél közelebb egy teljes bináris fához. Ezt a gyakorlatban lokális forgatásokkal oldják meg különböző technikák és feltételek alapján. Például kiegyensúlyozott bináris keresőfák az AVL fák, Piros-fekete fák vagy Splay fák.

6. Feladat Írjunk algoritmust, ami egy elsőfiú-testvér ábrázolású fának megállapítja a magasságát! (6.1)

Megoldás

Az algoritmus azon rekurzív összefüggés alapján számítja ki a fa magasságát, hogy minden csúcs magassága a gyerekei magasságának a maximuma +1. (A levelek magassága 0.)

```
magassag(f)
  max = -1
  g = f.elsofiu
  while(g!=NIL)
    m = magassag(g)
    if(m > max)
      max = m
    g = g.testver
  return max + 1
```

7. Feladat Írjunk olyan rekurzív algoritmust, ami egy elsőfiú-testvér ábrázolású fának megadja a levelei számát! (6.8)

Megoldás

Az algoritmus azon rekurzív összefüggés alapján számítja ki a fa leveleinek számát, hogy minden csúcs alatt található levelek száma a csúcs gyerekei alatt található gyerekek számának összege. (A leveleken ez a szám 1.)

```
levelek(f)
  if(f.elsofiu = NIL) return 1
  sum = 0
  g = f.elsofiu
  while(g != NIL)
    sum = sum + levelek(g)
    g = g.testver
  return sum
```

8. Feladat Írjunk olyan rekurzív algoritmust, ami egy elsőfiú-testvér ábrázolású fában megadja adott k paraméterre, hogy a fának hány csúcsa van a k szinten! (6.5)

Megoldás

Az algoritmus azon rekurzív összefüggés alapján számítja ki a k. szinten lévő csúcsok számát, hogy az első szinten a gyökér van, így az alapeset 1. A k. szinten annyi elem van, ahány részfa kezdődik azon a szinten.

```
KSzint(f, k)
  if(k = 1) return 1
  if(f.elsofiu = NIL) return 0
  sum = 0
  g = f.elsofiu
  while(g != NIL)
    sum = sum + KSzint(g, k-1)
    g = g.testver
  return sum
```