Algoritmusok és adatszerkezetek II

Fák témakör jegyzete

Készült Ásványi Tibor előadásai és gyakorlatai alapján

Sárközi Gergő, 2021-22-1. félév
Nincsen lektorálva!

Tartalomjegyzék

1. AVL fák **2**
1.1. Beszúrás . . . 3
1.1.1. Kiegyensúlyozás beszúrás miatt . . . 3
1.1.2. Beszúrás algoritmus . . . 5
1.2. Törlés . . . 6
1.2.1. Kiegyensúlyozás törlés miatt . . . 6
1.2.2. remMin algoritmus . . . 7
1.2.3. delRoot algoritmus . . . 8

2. Általános fák **9**

3. B+ fák **10**
3.1. Felépítés . . . 10
3.2. Műveleti igény . . . 10
3.3. Beszúrás . . . 11
3.4. Törlés . . . 12
3.4.1. Megtalált levélcsúcs = gyökér . . . 12
3.4.2. Megtalált levélcsúcs ≠ gyökér . . . 12
3.4.3. Belső (nem gyökér) csúcsból törlés . . . 13
3.4.4. Gyökérből (ami nem levél) törlés . . . 13

1. AVL fák

- (magasság szerint) kiegyensúlyozott bináris keresőfa
  - egy fa kiegyensúlyozott, ha minden csúcsa kiegyensúlyozott
  - $*p$ kiegyensúlyozott, ha $|p \to b| \leq 1$
  - $p \to b = h(p \to right) - h(p \to left)$
- AVL fákat láncoltan reprezentálunk
- legyen $n = |t|$ (csúcsok száma) és $h = h(t)$
- $\lfloor logn \rfloor \leq h \leq 1,45 \log n \implies h \in \Theta(logn)$
  - Felső becslés, alsó határ: $n < 2^{h+1} \implies \lfloor logn \rfloor \leq h$
  - Fibonacci fák: h mélységű, legkisebb méretű KBF-ek (kiegyensúlyozott bináris fák) csúcsainak száma:
    $f_0 = 1, f_1 = 2, f_h = 1 + f_{h-1} + f_{h-2}$ $(h \geq 2)$
    Ebből megkapható a $h \leq 1,45 \log n$
    $f_h = F_{h+3} - 1$ (ahol F a rendes fibonacci sorozat)
- $h(t) \leq n - 1$

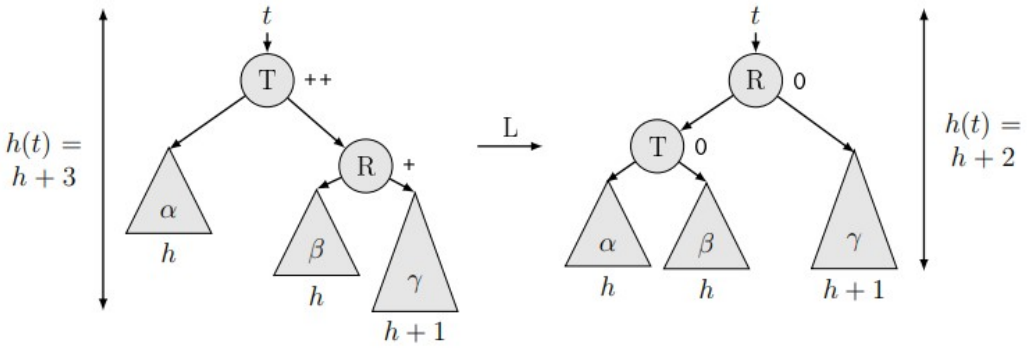
Függvény	Komplexitás	Leírás
search($t : Node^*; k : \mathbb{T}$) : $Node^*$	$\Theta(\log n)$	
min($t : Node^*$) : $Node^*$	$\Theta(\log n)$	ugyan így van max(t)
AVLinsert($\&t : Node^*; k : \mathbb{T}; \&d : \mathbb{B}$)	$\Theta(\log n)$	d igaz, ha nőtt h(t)
AVLremMin($\&t, \&minp : Node^*; \&d : \mathbb{B}$)	$\Theta(\log n)$	d igaz, ha csökkent h(t)
AVLdel($\&t : Node^*; k : \mathbb{T}; \&d : \mathbb{B}$)	$\Theta(\log n)$	d igaz, ha csökkent h(t)

Node
+ $key : \mathbb{T}$ + $b : \{-1, 0, 1\}$ + $left, right : Node^*$
+ $Node()\{left := right := \varnothing; b := 0\}$ + $Node(x : \mathbb{T})\{left := right := \varnothing; b := 0; key := x\}$

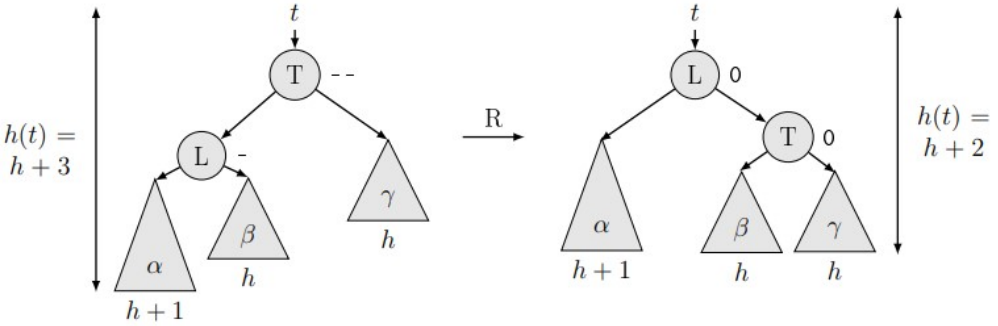
1.1. Beszúrás

1.1.1. Kiegyensúlyozás beszúrás miatt

- Minden esetben 1 vagy 2 lépés (BESZÚRÁS UTÁN CSAK!)
- Nem változtat az inorder bejáráson (logikus, hiszen keresőfa)



1. ábra. $(++, +)$ forgatás.



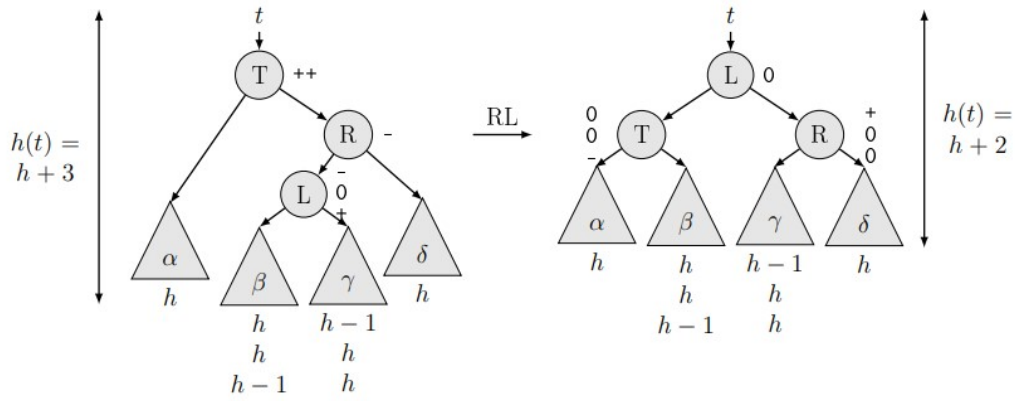
2. ábra. $(--, -)$ forgatás.

balancePPp(&t, r : Node*)

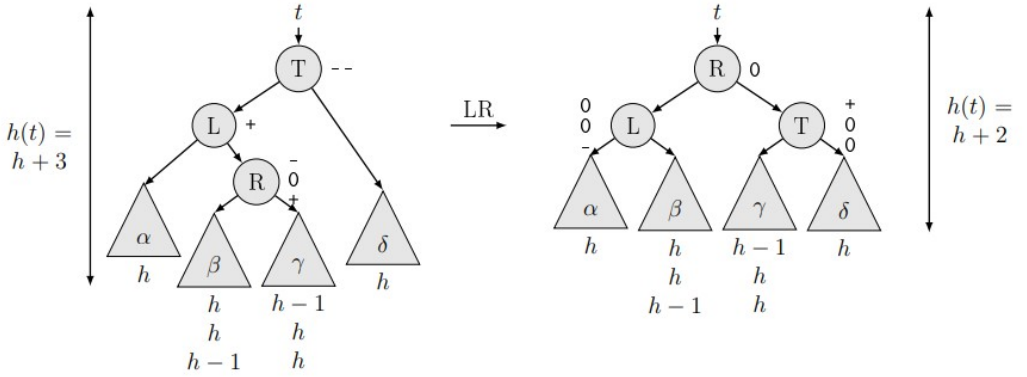
$t \to right := r \to left$
$r \to left := t$
$r \to b := t \to b := 0$
$t := r$

balanceMMm(&t, l : Node*)

$t \to left := l \to right$
$l \to right := t$
$l \to b := t \to b := 0$
$t := l$



3. ábra. $(++, -)$ forgatás.



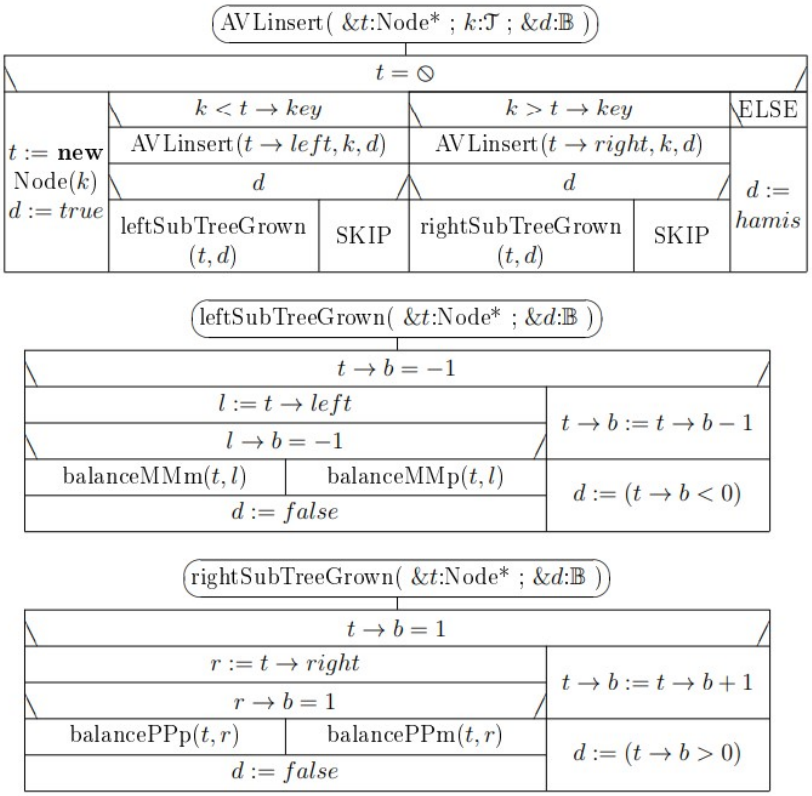
4. ábra. $(--, +)$ forgatás.

balancePPm(&t, r : Node*)
$l := r \to left$
$t \to right := l \to left$
$r \to left := l \to right$
$l \to left := t$
$l \to right := r$
$t \to b := -\lfloor (l \to b + 1)/2 \rfloor$
$r \to b := \lfloor (1 - l \to b)/2 \rfloor$
$l \to b := 0$
$t := l$

balanceMMp(&t, l : Node*)
$r := l \to right$
$l \to right := r \to left$
$t \to left := r \to right$
$r \to left := l$
$r \to right := t$
$l \to b := -\lfloor (r \to b + 1)/2 \rfloor$
$t \to b := \lfloor (1 - r \to b)/2 \rfloor$
$r \to b := 0$
$t := r$

1.1.2. Beszúrás algoritmus

- megkeressük a kulcs helyét
  - ha a kulcs benne van, kész vagyunk
  - ha a kulcs helyén lévő részfa üres: beszúrunk egy levélcsúcsot, így a részfa eggyel magasabb lett
- rálépünk a szülőre és az egyensúlyát módosítjuk (amelyikből ráléptünk, az a gyerek lett eggyel magasabb, szóval balance + vagy - 1)
  - ha balance=0, akkor kész vagyunk
  - ha |balance|=1, akkor ez a részfa magasabb lett eggyel, ismét rálépünk a szülőre, stb.
  - ha |balance|=2, akkor ki kell egyensúlyozni ezt a részfát, miután visszanyeri az eredeti (beszúrás előtti) magasságát, tehát a kiegyensúlyozás után kész vagyunk

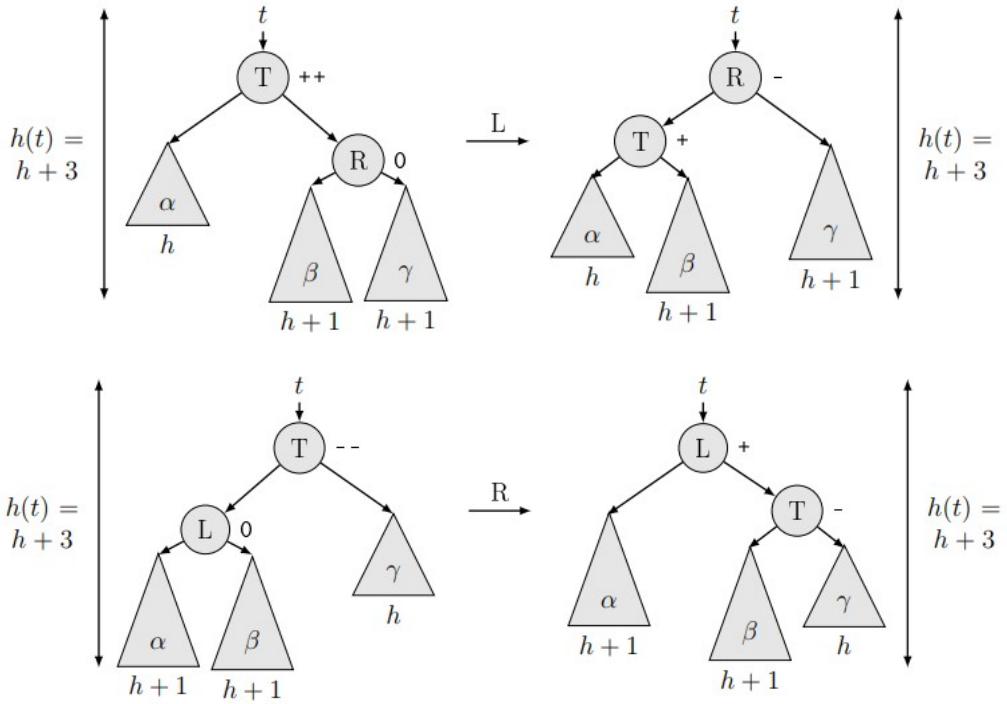


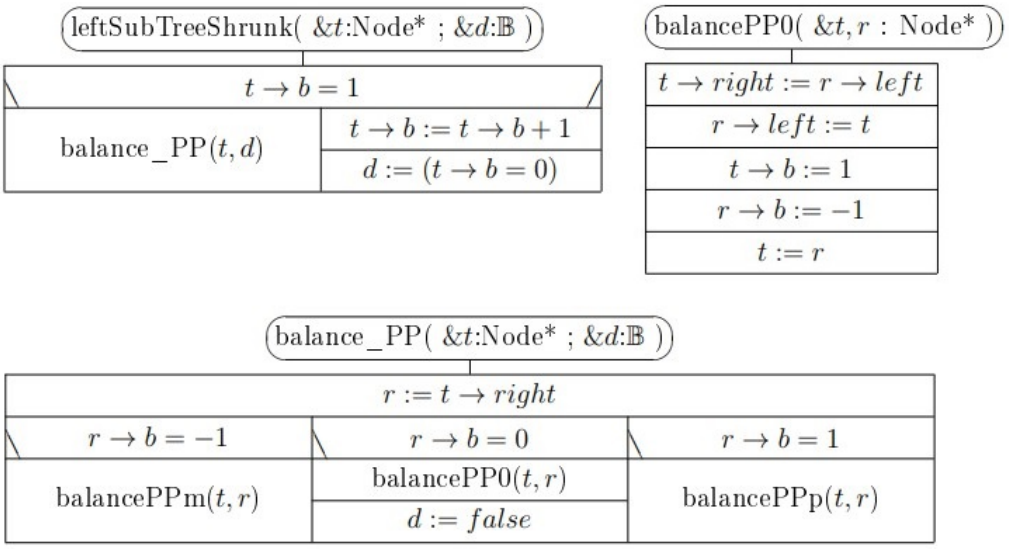
1.2. Törlés

- Törlésnél akár több (akár gyökérig tartó) forgatás (kiegyensúlyozás) kell. Azért, mert a forgatások csökkentik a részfa magasságát, és a törlés is.
- Törlés fajtái:
  - törlendő csúcs egyik részfája üres (levél vagy egy gyerekes csúcs): a másik részfát tesszük a csúcs helyére (ami lehet null) és a szülő balance-ját módosítjuk
  - kétgyerekes csúcs törlése: jobb részfa minimumjának kiemelése, fa kiegyensúlyozása, kicseréljük a törlendő csúcsot a kiemelt csúccsal

1.2.1. Kiegyensúlyozás törlés miatt

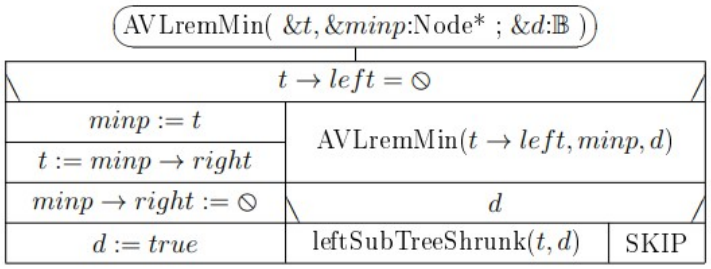
- Akárhány lépés lehet: forgatások és a törlés is csökkentik a részfa magasságát
  - Nem változtat az inorder bejáráson (hiszen keresőfa)



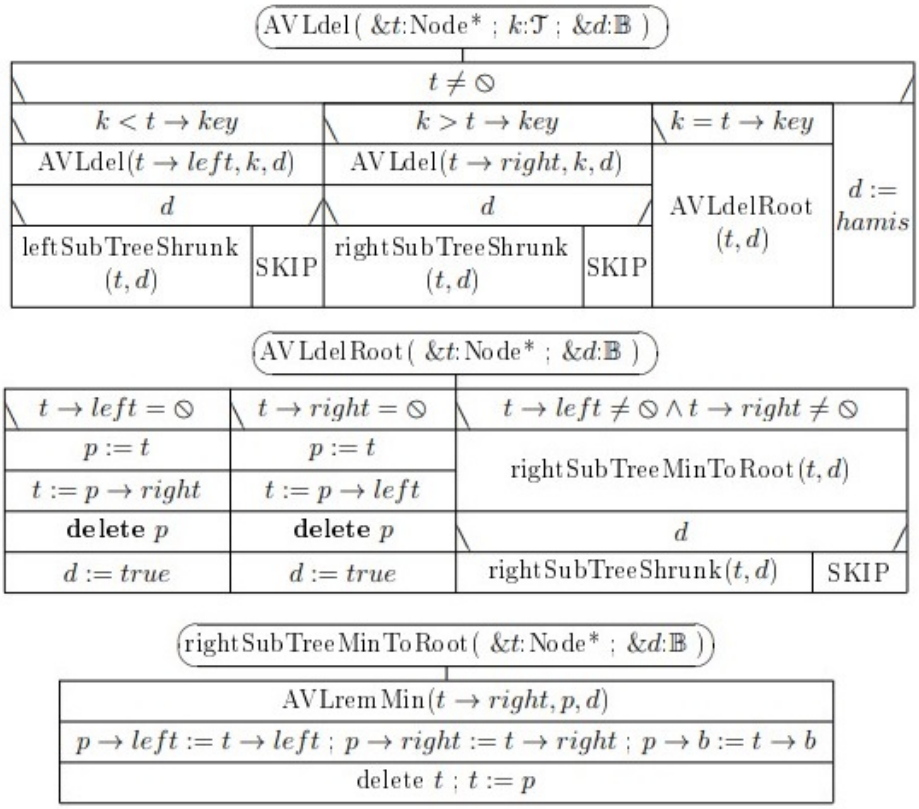


TODO balance MM metódus, stb.

1.2.2. remMin algoritmus

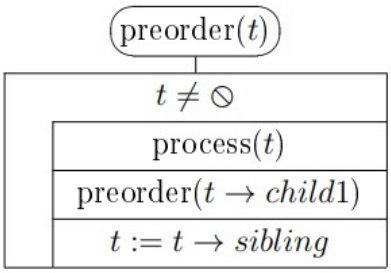


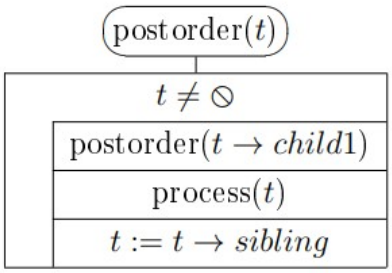
1.2.3. delRoot algoritmus



2. Általános fák

- Csúcsnak tetszőlegesen sok gyereke lehet
- csúcshoz nem tartoznak üres részfák: egy csúcsnak annyi gyereke van, amennyi, nincs üres gyerek
  - ezért nem r-áris fákról beszélni
- rendezett fa: ha a gyerekek sorrendje lényeges
- gyökér, levél fogalma ugyan úgy megvan
- *Node* osztály tagjai: $child1, sibling : Node*$
  - létezhetne szülő pointer is
  - levél: $p \to child1 = \varnothing$
  - utolsó testvér: $p \to sibling = \varnothing$
- szöveges ábrázolás: $(G\ t_1\ t_2\ ...\ t_n)$ (ahol G a gyökér)





3. B+ fák

3.1. Felépítés

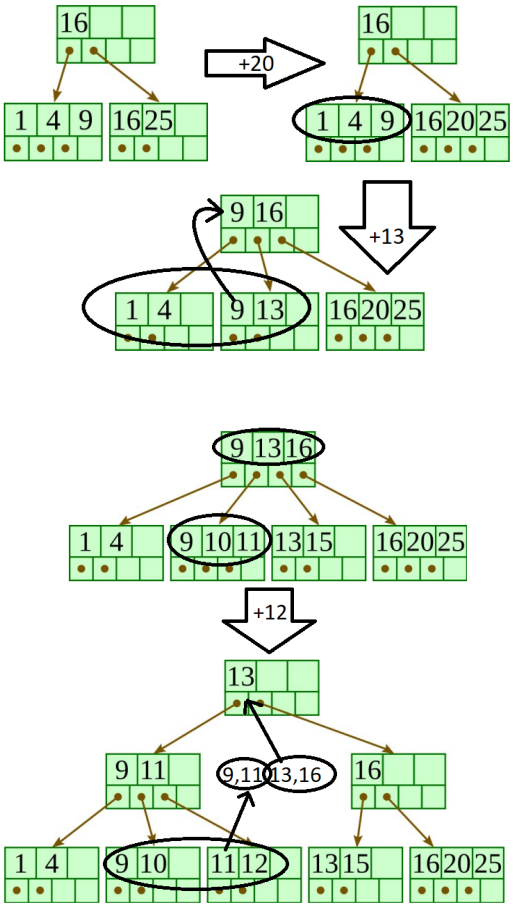
- d a B+ fa fokszáma, $4 \leq d$ (szóval r-áris fa, r=4)
- levelek:
  - azonos szinten (mélységben) vannak
  - adatokat tárolnak: minden kulcshoz tartozik egy adatra mutató
  - szóval a levelekben azonos számú (max $d-1$) kulcs és mutató van
- belső csúcsok:
  - max d mutató és pontosan eggyel kevesebb kulcs
  - belső kulcsok: hasító kulcsok
  - kulcsok viszonya egymáshoz:
    - legyen n a részfák száma, k a részfa bármelyik kulcsa
    - legyen K_i a szülő i. kulcsa $(1 \leq i < n)$
    - legszélső bal részfa: $k < K_1$
    - legszélső jobb részfa: $K_{n-1} \leq k$
    - közbülső, i. részfa: $K_{i-1} \leq k < K_i$
- gyerekek száma: (mindig max d)
  - gyökér: min 2, vagy pontosan 0
  - minden nem gyökér belső csúcs: min $\lfloor d/2 \rfloor$
- B+ fa által reprezentált adathalmaz minden értéke megjelenik egy levél kulcsaként, balról jobbra szigorúan monoton növekvő sorrendben

3.2. Műveleti igény

- Keresés, beszúrás, törlés: $\Theta(\log n)$

3.3. Beszúrás

- Üres a fa: készítsünk gyökeret, tartalmazza az értéket
- Keressük meg a levelet. Ha a levélben már szerepel a csúcs, fail.
- Ha nincs már a fában a kulcs és nem üres a fa:
  - Ha a csúcsban van szabad hely, láncoljuk be oda rendezetten
  - Ha a csúcs tele van, vágjuk szét két csúccsá, felezve az elemeket (elemek közé sorolva az újat is) (ha páratlan: balra menjen több)
    - szúrjuk be a jobb oldali csúcs legkisebb értékét a szülőbe
      - ha nincs szülő (gyökérben vagyunk), hozzuk létre
      - ha nem levél, akkor töröljük a régi helyről
    - beszúrás miatt rekurzívan ismételni, amíg szükséges



3.4. Törlés

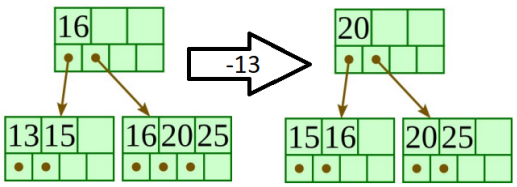
Meg kell keresni a törlendő kulcsot tartalmazó levelet. Ha nincs ilyen: fail.

3.4.1. Megtalált levélcsúcs = gyökér

- Töröljük a kulcsot és a mutatót
- Ha a gyökér tartalmaz még mutatót: kész vagyunk
- Ha a gyökér üres lett: töröljük, üres fát kapunk

3.4.2. Megtalált levélcsúcs ≠ gyökér

- Töröljük a kulcsot és a mutatót
- Ha a levél tartalmaz még elég mutatót ($\lfloor d/2 \rfloor$): kész vagyunk
- Ha a levél már túl kevés mutatót tartalmaz:
  - De van bal/jobb testvére, aki tud adni:
    - Kapjon a testvérétől annyit, hogy egyenlően el legyenek osztva a kulcsok
    - Szülőben a két levélhez tartozó kulcs (1db) átírása a jobb testvér min kulcsára
  - És nincs bal/jobb testvére, aki tudna adni:
    - Egyesítsük egy testvérével: jobból átpakolunk a balba, a jobb oldali levelet töröljük
    - Meghívunk egy törlő eljárást a szülőre: a két testvért eddig elválasztó kulcsot kell törölni (és a törölt csúcsra a mutatót)



3.4.3. Belső (nem gyökér) csúcsból törlés

- Töröljük a egyesített csúcsok közötti hasító kulcsot és az egyesítés során törölt csúcsra a mutatót
- Ha a csúcs tartalmaz még elég mutatót ($\lfloor d/2 \rfloor$): kész vagyunk
- Ha a csúcs már túl kevés mutatót tartalmaz:
  - De van bal/jobb testvére, aki tud adni:
    - Osszuk a két testvér kulcsait és a szülőben az őket elválasztó kulcsot egyenlően
    - A középső kulcs a szülőben lévőt cserélje ki
    - A maradék menjen a két testvérbe (az kapjon többet, akinél eddig is több volt)
  - És nincs bal/jobb testvére, aki tudna adni:
    - Egyesítsük egy testvérével: először a bal oldali kulcsok, utána a két testvér szülőjében lévő elválasztó kulcs, végül a jobb oldali csúcs kulcsai jönnek és a jobb oldali csúcsot töröljük
    - Meghívunk egy törlő eljárást a szülőre: a két testvért eddig elválasztó kulcsot kell törölni (és a törölt csúcsra a mutatót)

3.4.4. Gyökérből (ami nem levél) törlés

- Töröljük a egyesített csúcsok közötti hasító kulcsot és az egyesítés során törölt csúcsra a mutatót
- Ha a gyökérnek van még min 2 gyereke: kész vagyunk
- Ha a gyökérnek 1 gyereke maradt: töröljük a gyökereket, a fa magassága csökken, az egyetlen gyerek lesz az új gyökér