

Algoritmusok és adatszerkeztek II.

1. gyakorlat

AVL fa

- ❖ Az előző félévben tanult bináris keresőfa a hatékony keresést szolgálja, úgy, hogy a módosító műveletek (beszúrás, törlés) is hatékonyan elvégezhetők legyenek.
- ❖ Viszont épp a módosító műveletek során a fa alakja nagyon eltorzulhat (listává torzult bináris keresőfa keletkezhet), ami lerontja a keresés hatékonyságát.
- ❖ Ezért fontosak az úgynevezett „önkiegyensúlyozó” bináris keresőfák, melyek forgatásokkal biztosítják, hogy az alakjukra vonatkozó invariánsok fennálljanak a módosító műveletek után is. Ezzel biztosítják, hogy a keresés $O(\log n)$ hatékonyságú maradjon.
- ❖ A módosító műveletek utáni esetleges forgatások pedig konstans lépésszámú pointer állítással működnek.
- ❖ Két nevezetes ilyen fa:
 - ❖ Piros-fekete fa
 - ❖ AVL fa (ezzel fogunk most foglalkozni)

AVL fa definíció

- ❖ Az AVL mozaik szó: Adelszon-Velszkij és Landisz szovjet matematikusok publikálták 1962-ben. (*Georgij Maximovics Adelszon-Velszkij, Evgenij Mikhailovics Landisz*)
- ❖ **Definíció:**
Az AVL fák magasság szerint kiegyensúlyozott bináris keresőfák. minden (*p) pontjukra teljesül a következő (AVL) tulajdonság:
$$| h(p \rightarrow \text{left}) - h(p \rightarrow \text{right}) | \leq 1$$
- ❖ Azaz (*p) Node bal-, és jobb részfájának magasságának különbsége legfeljebb egy lehet.
- ❖ **Tétel:**
tetszőleges n csúcsú AVL fa h magasságára teljesül, hogy:
$$\lfloor \log n \rfloor \leq h \leq 1,45 * \log n \quad \text{azaz} \quad h \in \Theta(\log n)$$

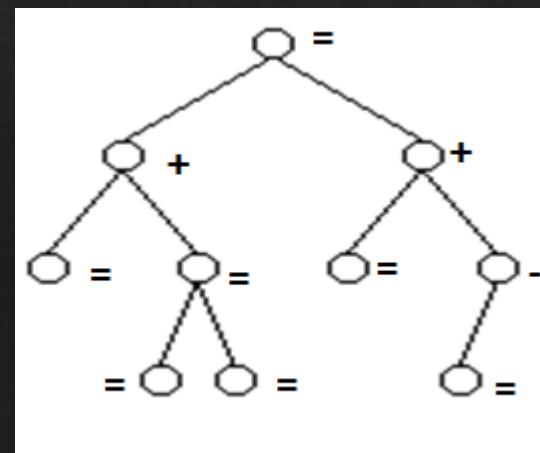
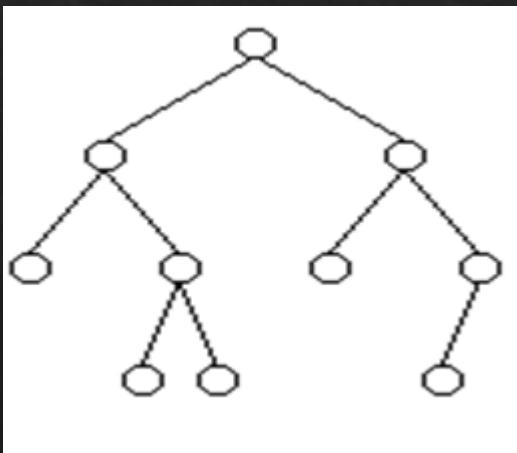
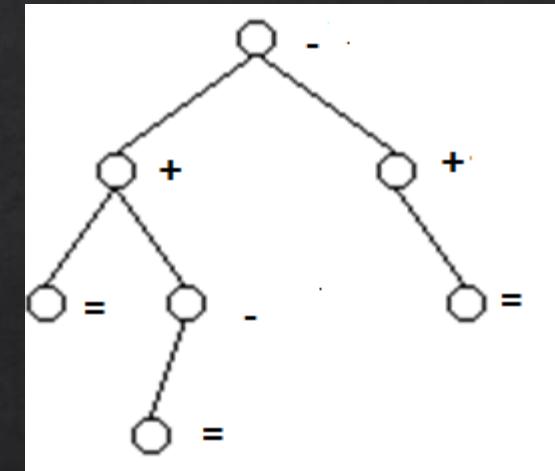
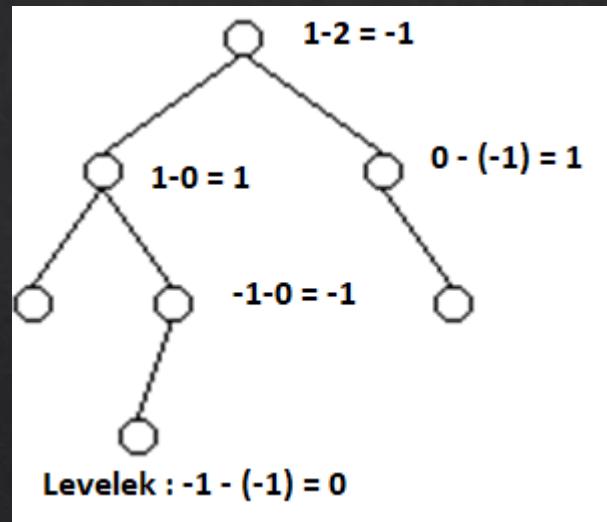
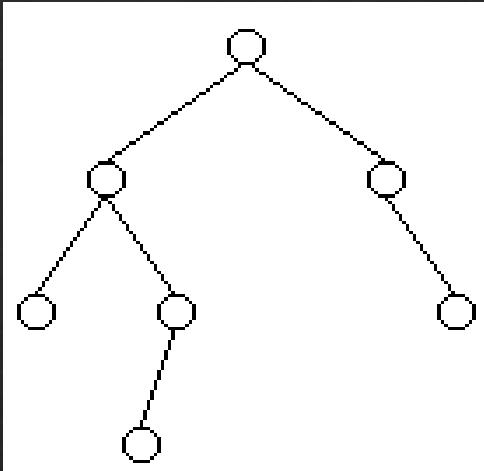
Megjegyzések:

- ❖ Az AVL-fára, mint speciális alakú keresőfára, változatlanul érvényesek a keresőfákra bevezetett műveletek.
- ❖ minden művelet (beszúrás és törlés) után ellenőrizzük, és ha kell, helyreállítjuk az AVL-tulajdonságot.
- ❖ Az AVL fát láncoltan reprezentáljuk és a csúcsban tároljuk az egyensúlyát (balance), ahol $p \rightarrow b := h(p \rightarrow right) - h(p \rightarrow left)$ és $p \rightarrow b \in \{-1, 0, +1\}$

Node
+ <i>key</i> : \mathcal{T} // \mathcal{T} is some known type
+ <i>b</i> : $-1..1$ // the balance of the node
+ <i>left, right</i> : Node*
+ Node() { <i>left</i> := <i>right</i> := \emptyset ; <i>b</i> := 0 } // create a tree of a single node
+ Node(<i>x</i> : \mathcal{T}) { <i>left</i> := <i>right</i> := \emptyset ; <i>b</i> := 0 ; <i>key</i> := <i>x</i> }

- ❖ Megjegyzés: tárolhatnánk egyensúly helyett az aktuális részfa magasságát a Node-ban.

Döntsük el, hogy az alábbi bináris fák AVL tulajdonságúak-e:

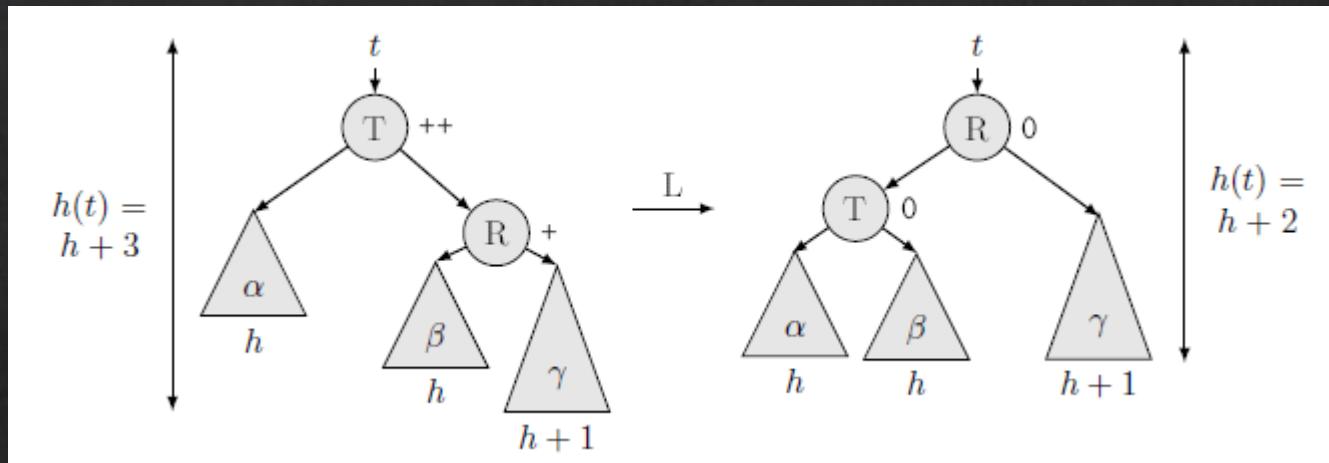


- ❖ Beveztjük az alábbi jelöléseket a rajzokon:
 - ❖ A csúcs jelzője (indikátora) az '=' , ha a csúcs két részfájának magassága egyenlő.
(Jegyzet ,0'-val jelöli.)
 - ❖ A csúcs jelzője a '-' , ha a csúcs baloldali részfájának magassága eggyel nagyobb, mint a jobboldali részfáé.
 - ❖ A csúcs jelzője a '+' , ha a csúcs jobboldali részfájának magassága eggyel nagyobb, mint a baloldali részfáé.
- ❖ Megjegyzések:
 - ❖ A levelek jelzője mindenkor az '=' .
 - ❖ Ha egy csúcs jelzője beszúrás vagy törlés miatt '++' , vagy '- -' lesz (ez jelzi, hogy elromlott az AVL-tulajdonság), javítanunk kell!

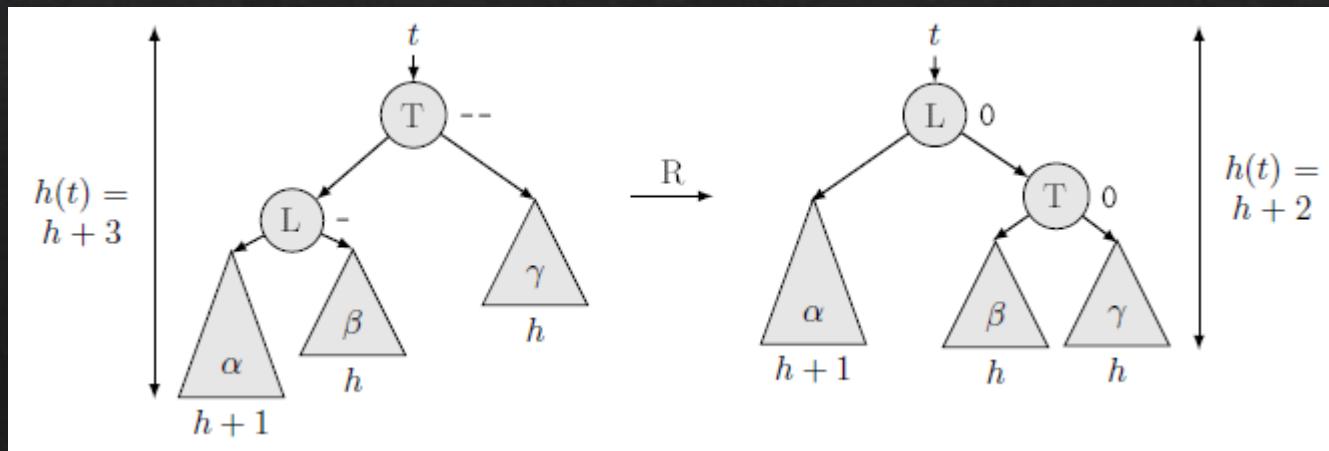
Forgatási sémák (beszúrás esetei)

- ❖ Négy forgatási sémát használunk a beszúrás utáni helyreállításokhoz:
- ❖ $(++,+)$ $(--, -)$ $(++, -)$ $(--, +)$
- ❖ $(++, +)$ séma tükörképe a $(--, -)$
- ❖ Hasonlóan $(++, -)$ séma tükörképe a $(--, +)$
- ❖ Így tulajdonképpen két sémát kell csak megtanulnunk.

(++,+) és (--,--) séma



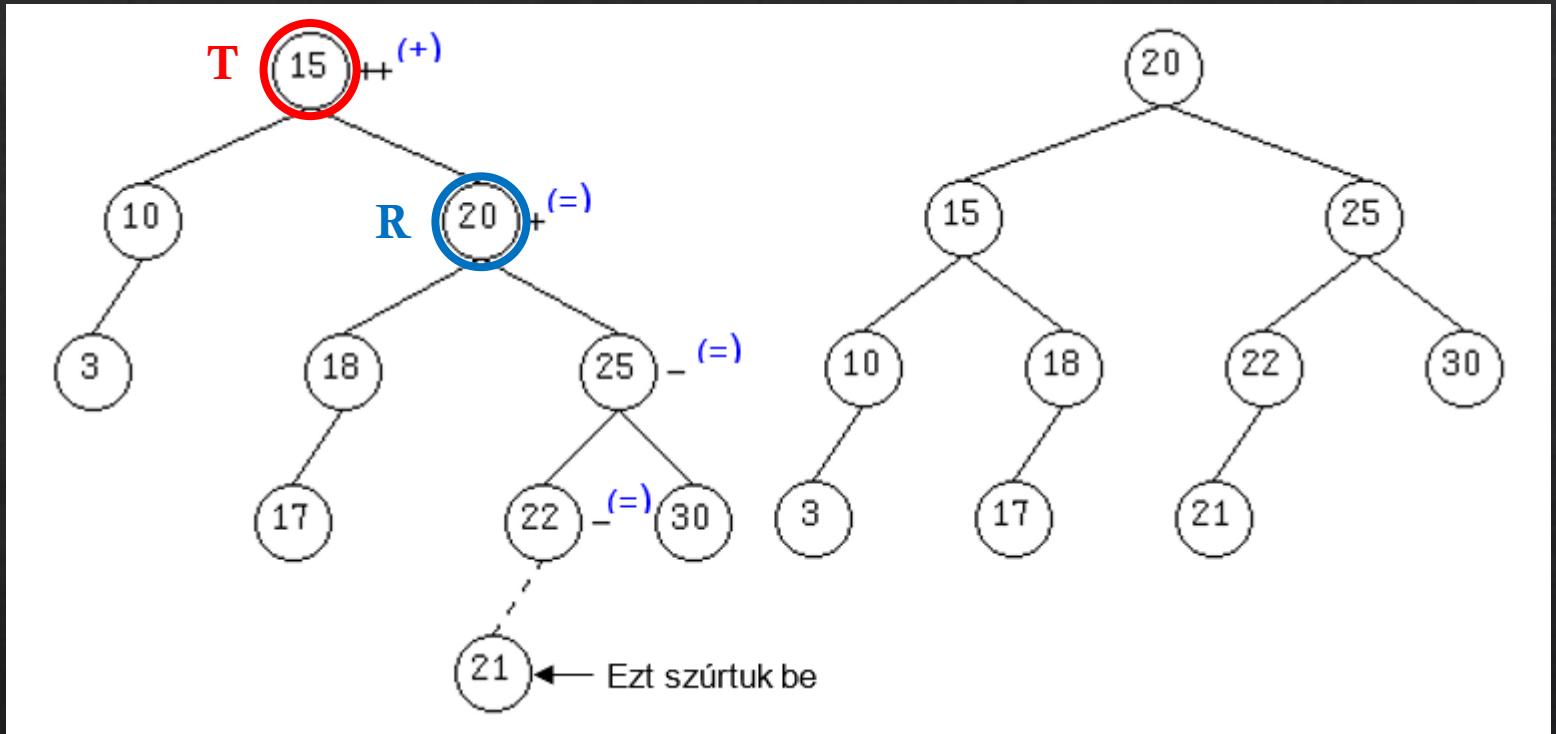
- γ részfába történt a beszúrás
- A forgatás helyreállította az eredeti $h+2$ magasságot és,
- megtartotta a keresőfa tulajdonságát:
 $\alpha < T < \beta < R < \gamma$



- α részfába történt a beszúrás
- A forgatás helyreállította az eredeti $h+2$ magasságot és,
- megtartotta a keresőfa tulajdonságát:
 $\alpha < L < \beta < T < \gamma$

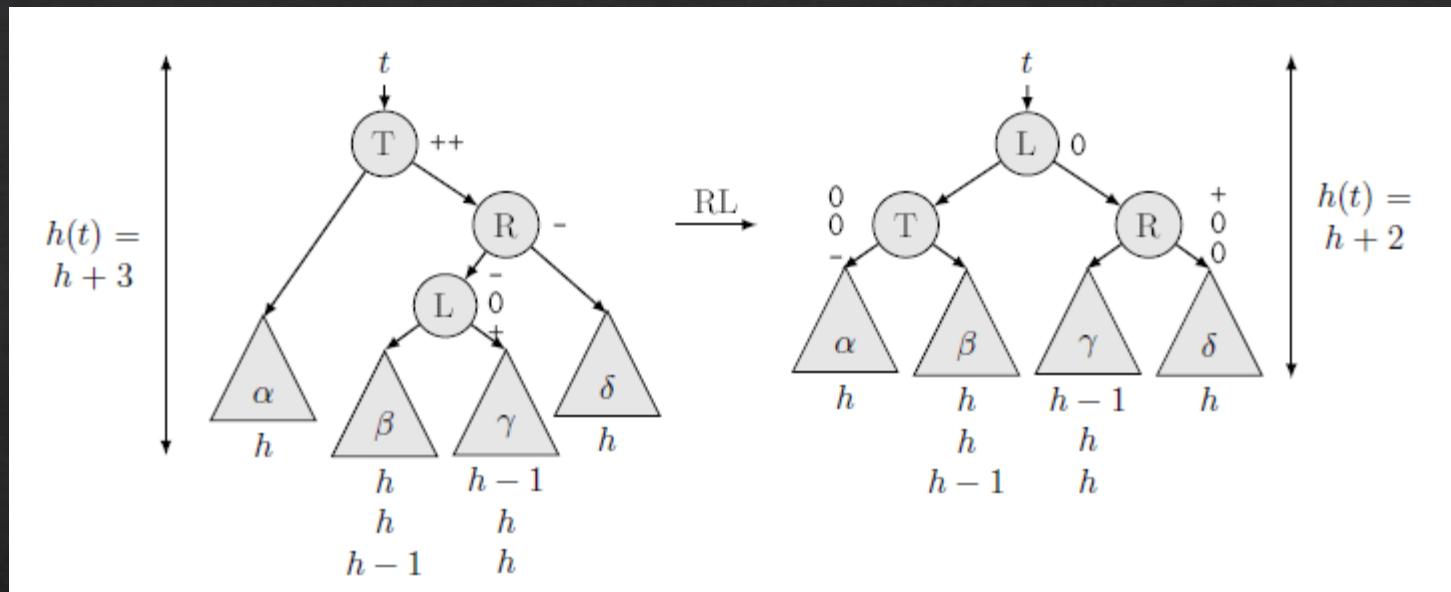
(++,+) példa

- ❖ Induljunk el a beszúrt csúcs szülőjétől a gyökér felé,
- ❖ Addig menjünk, amíg (a jelzők korrekcióját elvégezve) '=' vagy '++' (' - - ') nem alakul ki, illetve a gyökérig nem érünk,
- ❖ A '++' (' - - ') esetében javítunk (forgatunk), és tovább már nem kell nézni



(++,-) forgatás

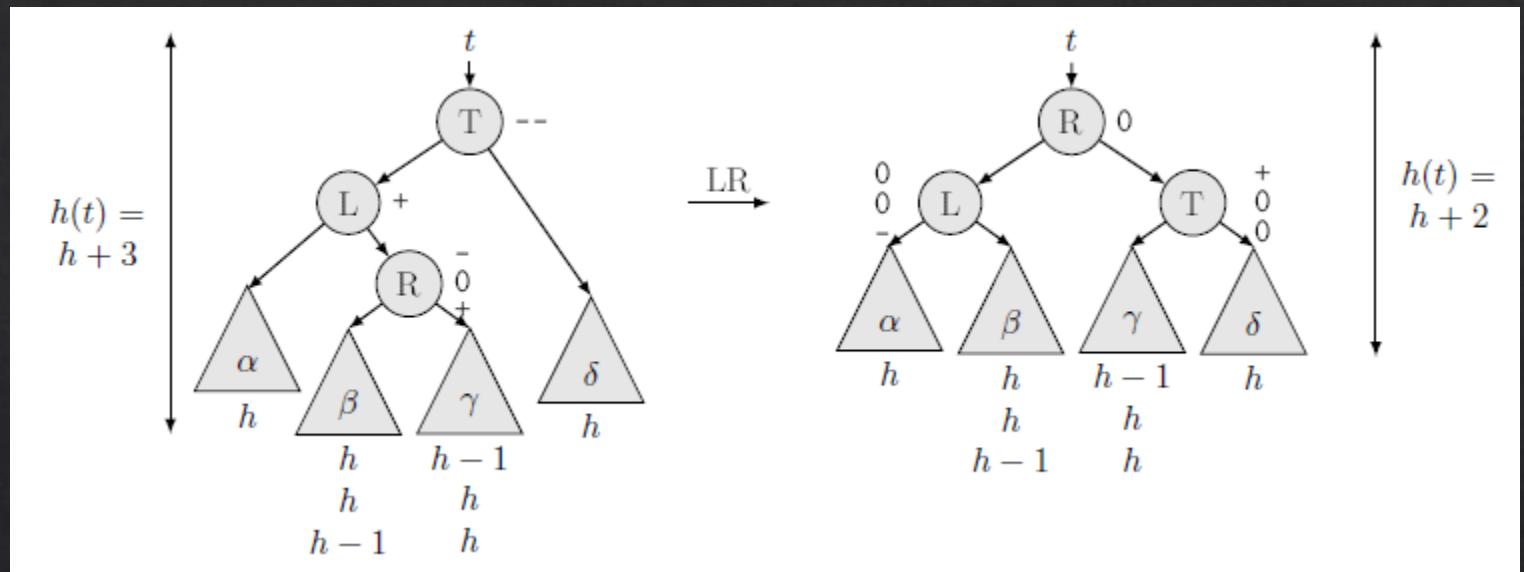
- ❖ Ha megjegyezzük, hogy L csúcs lesz a részfa új gyökere, akkor a keresőfa tulajdonságából adódik a többi csúcs és részfa „helye”.
- ❖ Itt is igaz, hogy helyreállt az eredeti magasság, és a fa megtartotta a keresőfa tulajdonságát.



- ❖ Megjegyzés: A (+ + , -) eset háromféleképpen állhat elő:
 - ❖ Az új elem a β részfába került (h , $h-1$)
 - ❖ Az 'L' az új elem, részfák nincsenek (h , h)
 - ❖ Az új elem a γ részfába került ($h-1$, h)

(--, +) forgatás

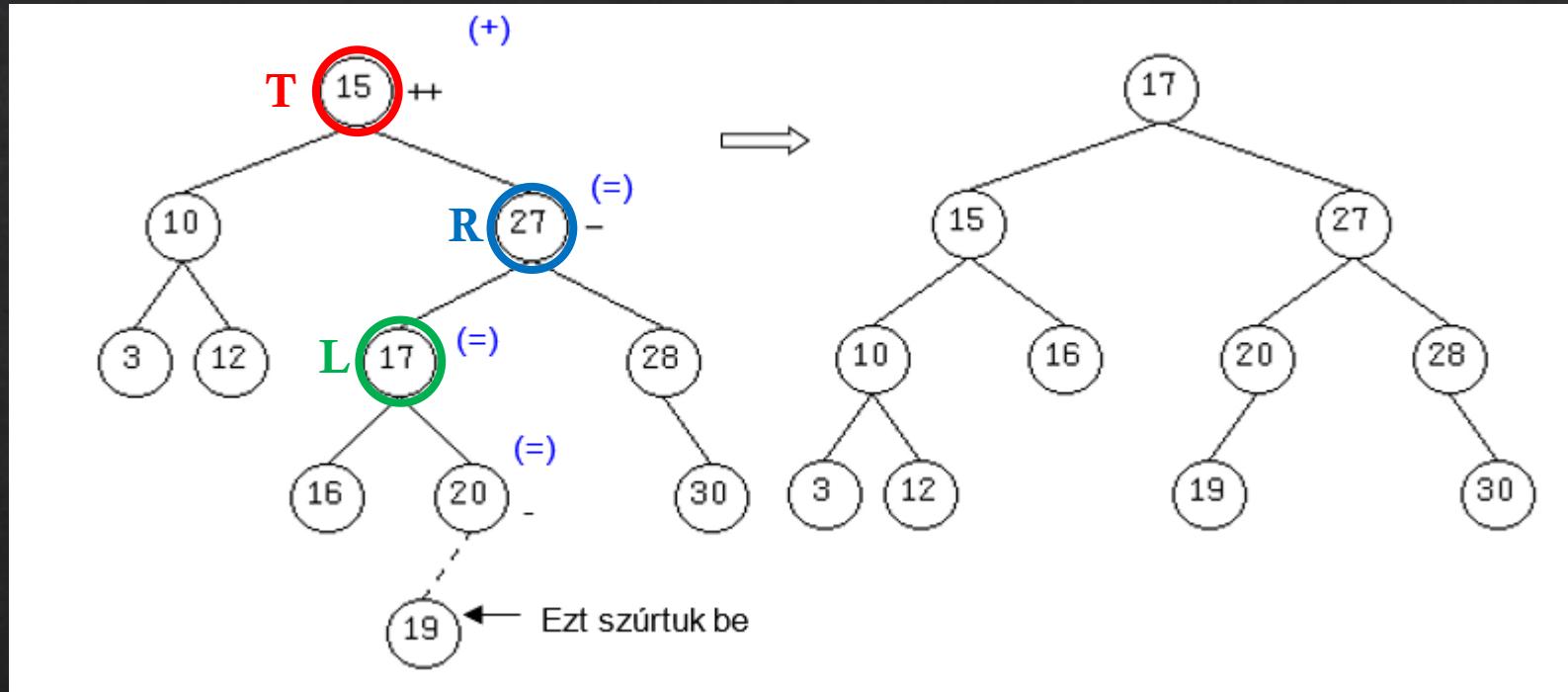
- ❖ Ha megjegyezzük, hogy R csúcs lesz a részfa új gyökere, akkor a keresőfa tulajdonságából adódik a többi csúcs és részfa „helye”.
- ❖ Itt is igaz, hogy helyreállt az eredeti magasság, és a fa megtartotta a keresőfa tulajdonságát.



- ❖ Megjegyzés: A (--, +) eset háromféleképpen állhat elő:
 - ❖ Az új elem a β részfába került (h , $h-1$)
 - ❖ Az 'R' az új elem, részfák nincsenek (h , h)
 - ❖ Az új elem a γ részfába került ($h-1$, h)

(++,-) példa

- ❖ Az algoritmus itt is hasonló:
- ❖ A beszúrás helyétől elindulunk a gyökér felé, és addig folytatjuk az átsúlyozást, amíg „=”, nem lesz, vagy a gyökérhez nem érünk.
- ❖ Ha az egyensúly „++”, vagy „--” lesz, elvégezzük a forgatást.



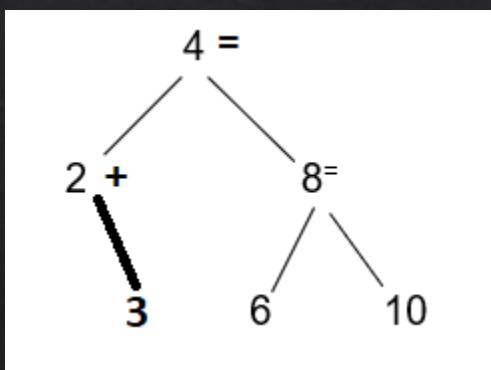
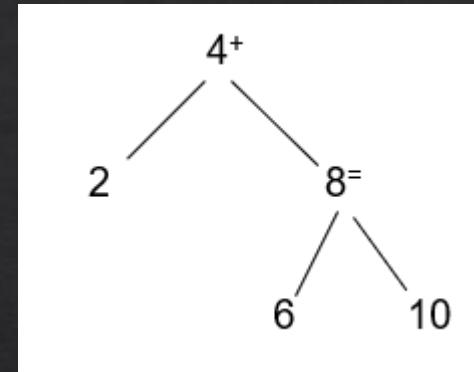
- ❖ Megjegyzés: interneten található videókban a (++,-) és (--,+) forgatásokat két lépésre bontják, például a (++,-) eset:
- ❖ Elsőként a 27 gyökerű részfán egy „R” forgatás (mint a (–, -) séma)
- ❖ Majd a T gyökerű részfán egy „L” forgatás (mint a (++,+) séma)
- ❖ De ne ezt tanuljuk meg! (ELTE IK: **egy** RL forgatás! Hatékonyság!!!)

Gyakorló feladat

- ❖ Egymás utáni beszúrásokkal építsünk AVL fát a következő kulcsokból:
- ❖ 12, 34, 13, 10, 11, 80, 20, 17, 29, 25, 28, 22, 33, 50, 99, 85, 5
- ❖ Ha az AVL tulajdonság elromlik, ábrázoljuk a beszúrás utáni új balance értékeket,
- ❖ Jelöljük a forgatás helyét (forgatott részfa gyökerét),
- ❖ Adjuk meg a forgatás típusát,
- ❖ Végezzük el a forgatást,
- ❖ Ábrázoljuk a forgatás utáni balance értékeket.
- ❖ Segítség: a példában minden forgatási séma elő fog fordulni.

AVL fa szöveges megadása, feladat

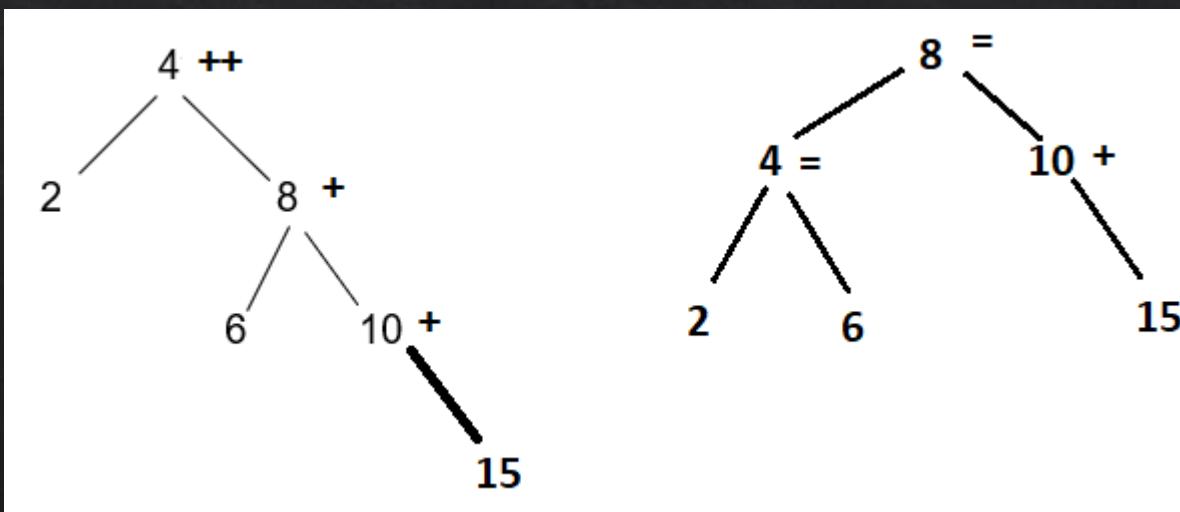
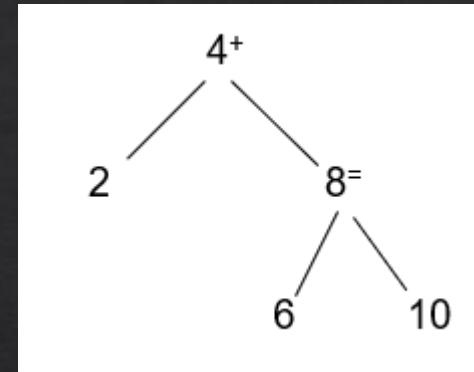
- ❖ A mellékelt fán végezzük el a következő műveleteket, az eredményt adjuk meg szövegesen és grafikusan is.
- ❖ Induló fa szöveges megadása:
 $\{ [2] 4+ [(6) 8= (10)] \}$
- ❖ Szúrjuk be a fenti fába a 3-as értéket! Írjuk le a kapott fát szöveges jelöléssel is!



$\{ [2+ (3)] 4= [(6) 8= (10)] \}$

AVL fa szöveges megadása, feladat

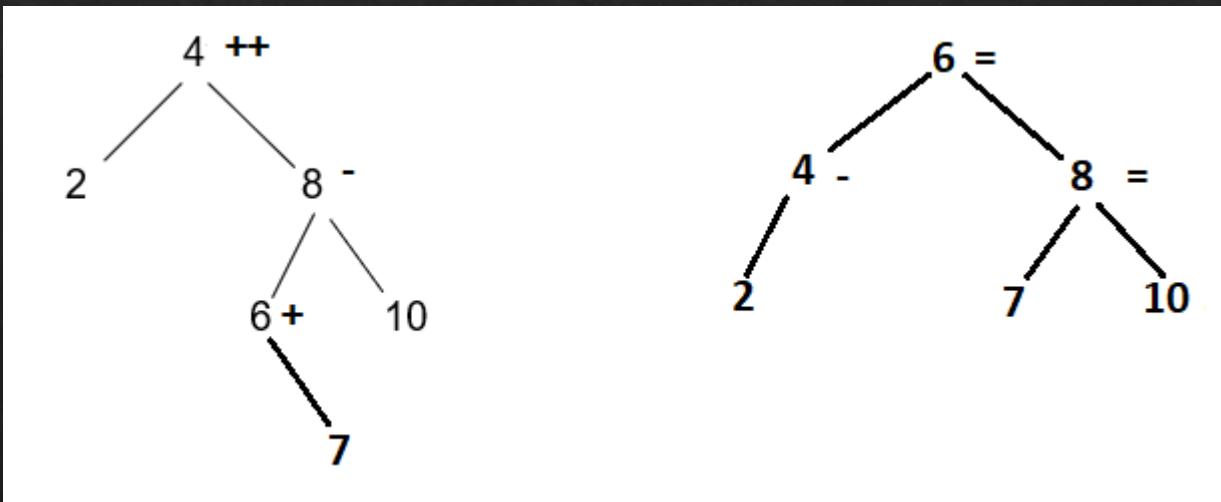
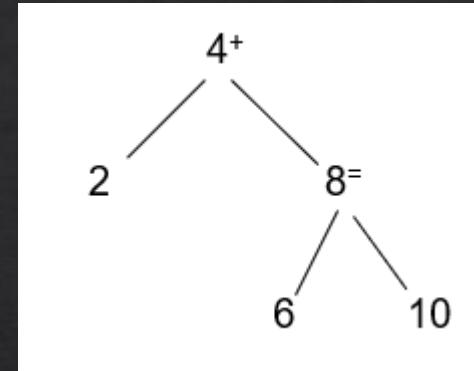
- ❖ A mellékelt fán végezzük el a következő műveleteket, az eredményt adjuk meg szövegesen és grafikusan is.
- ❖ Induló fa szöveges megadása:
 $\{ [2] 4+ [(6) 8= (10)] \}$
- ❖ Szúrjuk be a fenti fába a 15-ös értéket! Írjuk le a kapott fát szöveges jelöléssel is!



$\{ [(2) 4= (6)] 8= [10+ (15)] \}$

AVL fa szöveges megadása, feladat

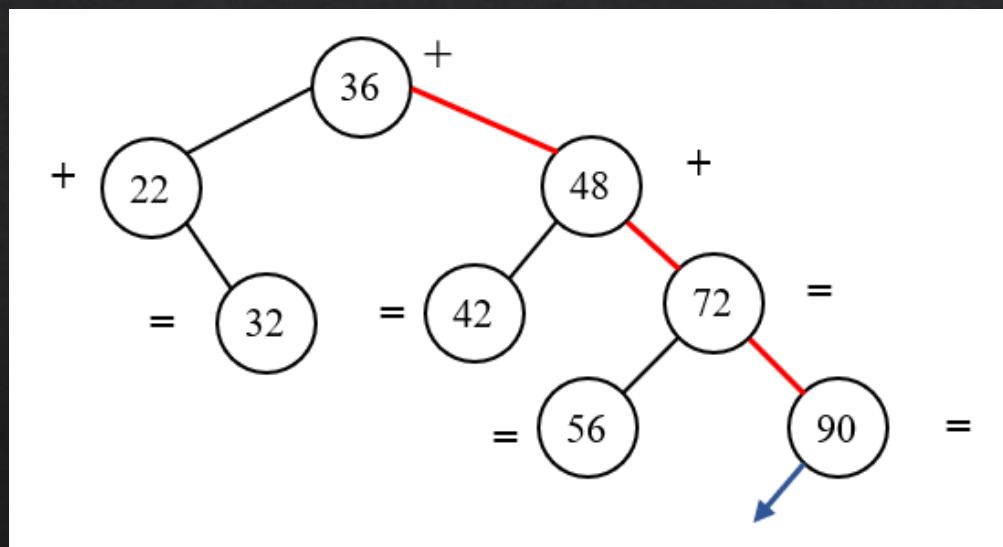
- ❖ A mellékelt fán végezzük el a következő műveleteket, az eredményt adjuk meg szövegesen és grafikusan is.
- ❖ Induló fa szöveges megadása:
 $\{ [2] 4+ [(6) 8= (10)] \}$
- ❖ Szúrjuk be a fenti fába a 7-es értéket! Írjuk le a kapott fát szöveges jelöléssel is!



$\{ [(2) 4-] 6= [(7) 8= (10)] \}$

AVL beszúró algoritmus működése

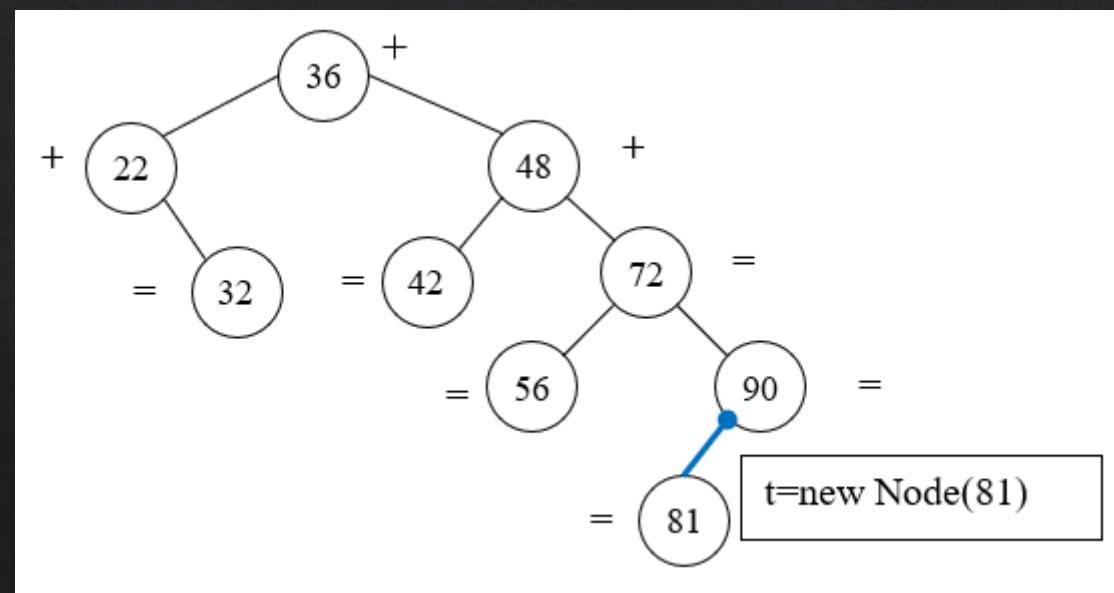
- ❖ Adott az ábrán látható AVL fa, szúrjuk be a 81 kulcsot!



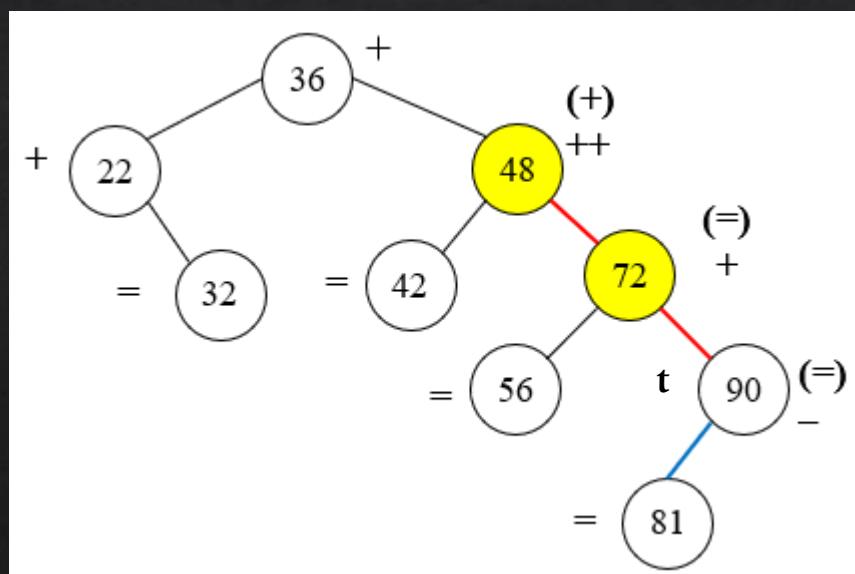
AVLinsert(<i>&t:Node*</i> ; <i>k:T</i> ; <i>&d:B</i>)		
<i>t = ⊖</i>		
<i>t := new Node(k)</i>	<i>k < t → key</i> AVLinsert(<i>t → left, k, d</i>)	<i>k > t → key</i> AVLinsert(<i>t → right, k, d</i>)
<i>d := true</i>	<i>d</i> <i>leftSubTreeGrown (t, d)</i>	<i>d</i> <i>rightSubTreeGrown (t, d)</i>
	SKIP	SKIP
		<i>d := hamis</i>

- ❖ t pointer tartalma (ami a 90-es Node left pointere) nulla,
- ❖ Létrejön az új Node,
- ❖ Címe t pointerbe kerül, ami a 90-es Node left pointere!
- ❖ Így befűződik az új kulcs a fába.
- ❖ d=true, és megindul a rekurzió visszafelé.

AVLInsert(&t:Node* ; k:T ; &d:B)			
	$t = \emptyset$	$k < t \rightarrow key$	$k > t \rightarrow key$
$t := \text{new Node}(k)$	AVLInsert($t \rightarrow \text{left}, k, d$)	AVLInsert($t \rightarrow \text{right}, k, d$)	ELSE
$d := \text{true}$	d	d	$d := \text{hamis}$
	leftSubTreeGrown (t, d)	SKIP	rightSubTreeGrown (t, d)
			SKIP



- ❖ Mivel a 90-es csúcsnál a bal részfára hívtuk meg rekurzívan a beszúró algoritmust, továbbá d igaz, a „leftSubTreeGrown” függvény fut le.

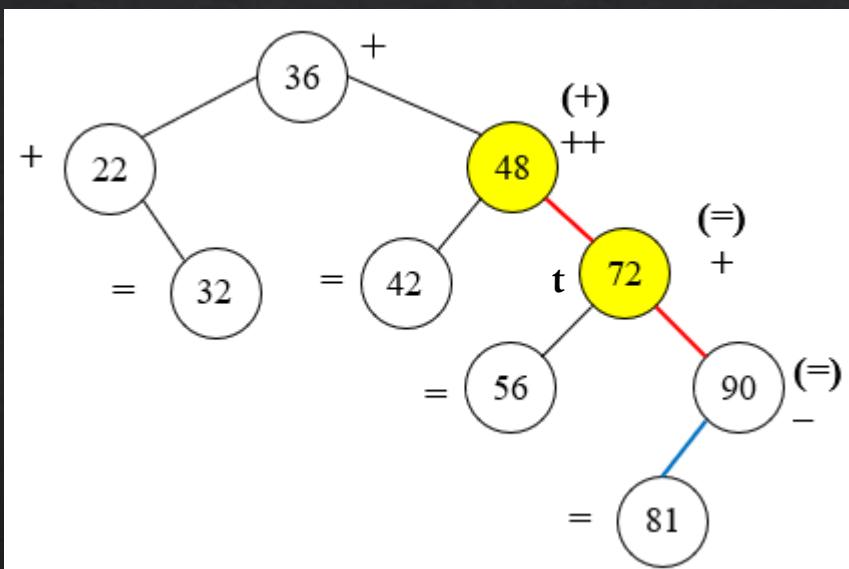


AVLInsert($\&t:\text{Node}^*$; $k:\mathcal{T}$; $\&d:\mathbb{B}$)			
$t = \emptyset$			
$k < t \rightarrow \text{key}$		$k > t \rightarrow \text{key}$	
$t := \text{new Node}(k)$	AVLInsert($t \rightarrow \text{left}, k, d$)	AVLInsert($t \rightarrow \text{right}, k, d$)	ELSE
$d := \text{true}$	d	d	$d := \text{hamis}$
	leftSubTreeGrown (t, d)	SKIP	rightSubTreeGrown (t, d)
		SKIP	

leftSubTreeGrown($\&t:\text{Node}^*$; $\&d:\mathbb{B}$)	
$t \rightarrow b = -1$	
$l := t \rightarrow \text{left}$	$t \rightarrow b := t \rightarrow b - 1$
$l \rightarrow b = -1$	
balanceMMm(t, l)	balanceMMP(t, l)
	$d := \text{false}$

- ❖ A 90-es csúcs balansza 0 volt, a jobb ágra kerülünk, az új balansz -1 lesz, d továbbra is igaz marad. (A 90-es részfája magassága nőtt.)

- ◆ A 72 csúcsnál „jobb irányból érkezünk”, és d igaz, így a „rightSubTreeGrown” algoritmus fut le.

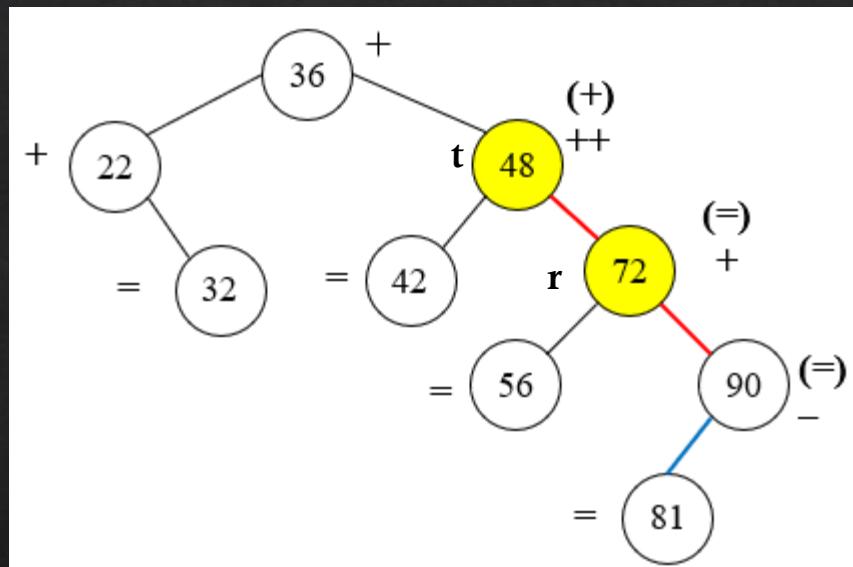


AVLinsert($\&t:\text{Node}^*$; $k:\mathcal{T}$; $\&d:\mathbb{B}$)				
$t = \emptyset$				
$k < t \rightarrow \text{key}$		$k > t \rightarrow \text{key}$		ELSE
$t := \text{new Node}(k)$	d	$\text{AVLinsert}(t \rightarrow \text{left}, k, d)$	$\text{AVLinsert}(t \rightarrow \text{right}, k, d)$	
$d := \text{true}$		$\text{leftSubTreeGrown}(t, d)$	SKIP	$d := \text{hamis}$
			$\text{rightSubTreeGrown}(t, d)$	SKIP

rightSubTreeGrown($\&t:\text{Node}^*$; $\&d:\mathbb{B}$)			
$t \rightarrow b = 1$			
$r := t \rightarrow \text{right}$		$t \rightarrow b := t \rightarrow b + 1$	
$r \rightarrow b = 1$			
$\text{balancePPP}(t, r)$	$\text{balancePPm}(t, r)$	$d := (t \rightarrow b > 0)$	
		$d := \text{false}$	

- ◆ A 72-es csúcs balansza 0 volt, így a jobb ágra kerülünk, a balansz +1 lesz, d továbbra is igaz marad (a 72-es részfa magassága is nőtt).

- ◆ A 48 csúcsnál is „jobb irányból érkezünk”, és d igaz, így a „rightSubTreeGrown” algoritmus fut le.

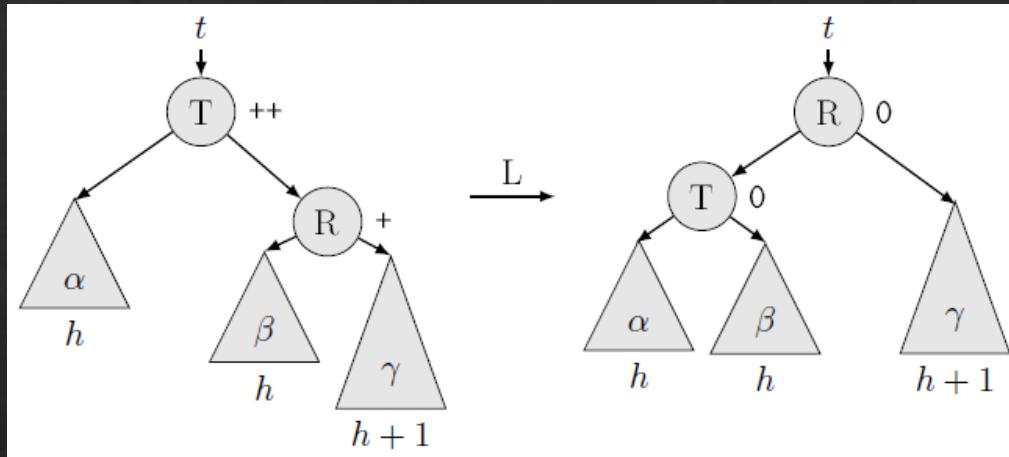


AVLinsert(&t:Node* ; k:T ; &d:B)		
$t = \emptyset$		
$t := \text{new Node}(k)$	$k < t \rightarrow \text{key}$	$k > t \rightarrow \text{key}$
$d := \text{true}$	AVLinsert($t \rightarrow \text{left}, k, d$)	AVLinsert($t \rightarrow \text{right}, k, d$)
	d	d
	leftSubTreeGrown (t, d)	SKIP
		rightSubTreeGrown (t, d)
		SKIP
		$d := \text{hamis}$

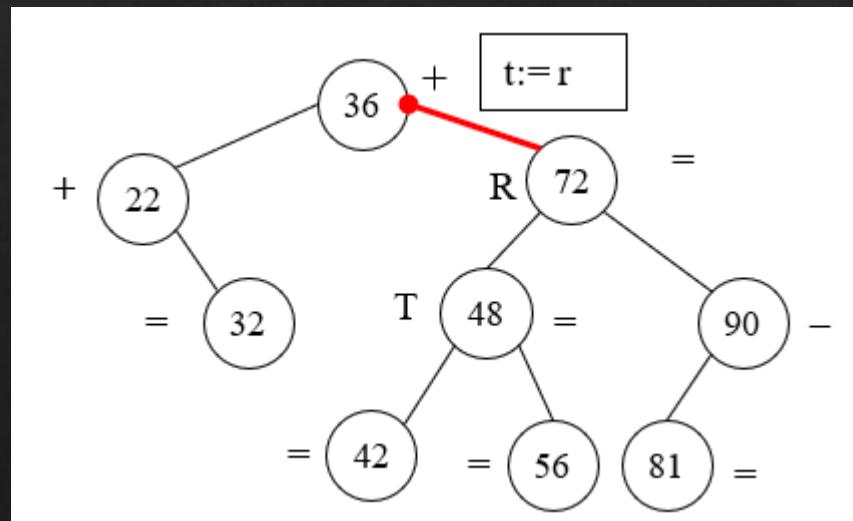
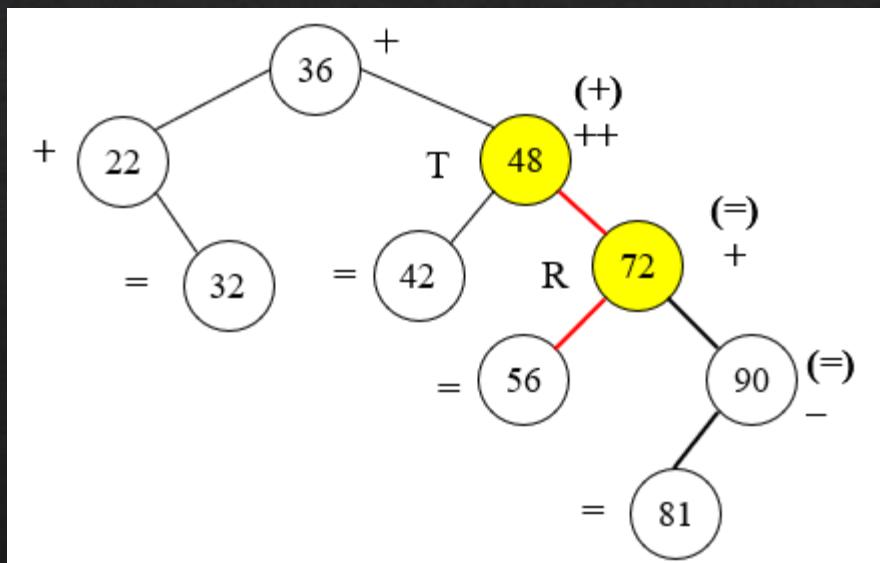
rightSubTreeGrown(&t:Node* ; &d:B)		
$t \rightarrow b = 1$		
$r := t \rightarrow \text{right}$		$t \rightarrow b := t \rightarrow b + 1$
$r \rightarrow b = 1$		
$\text{balancePPp}(t, r)$	$\text{balancePPm}(t, r)$	$d := (t \rightarrow b > 0)$
	$d := \text{false}$	

- ◆ A 48-as csúcs balansza 1 volt, így a csúcs ++ jelzőjű lenne, forgatás jön, bal ágra lépünk.
- ◆ Jobb gyerek a 72, a 72 balansza +1, így a balancePPp algoritmus következik.
- ◆ $d = \text{false}$ értékkadás miatt innenől a „SKIP” ágon fut vissza a rekurzió a fa gyökeréig.

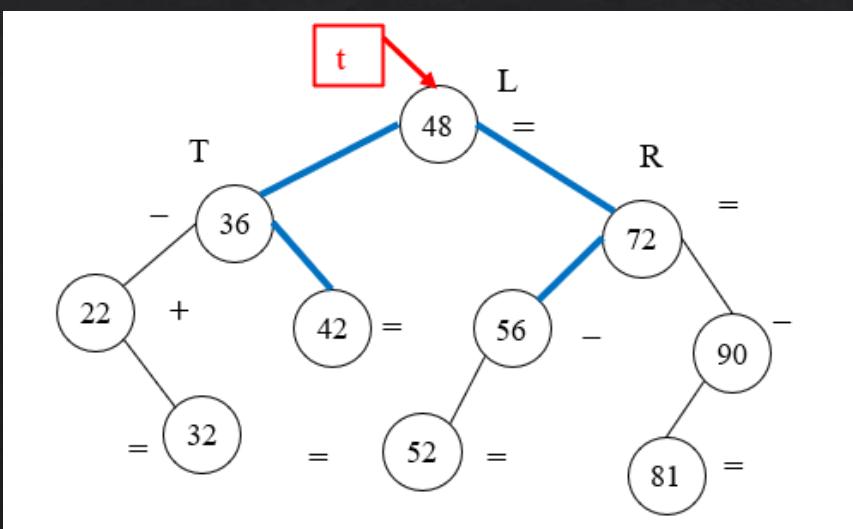
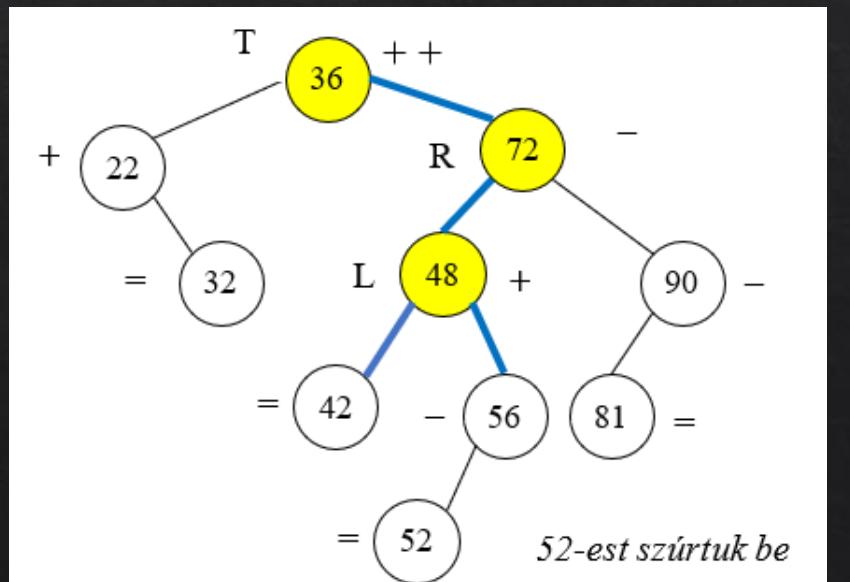
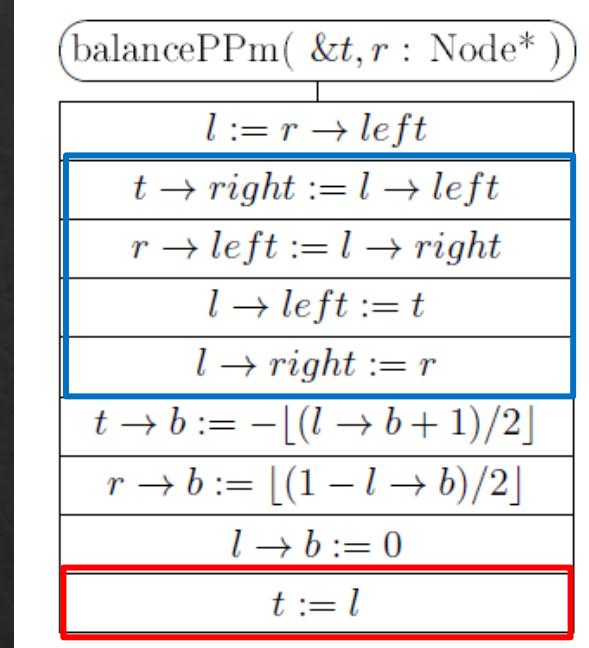
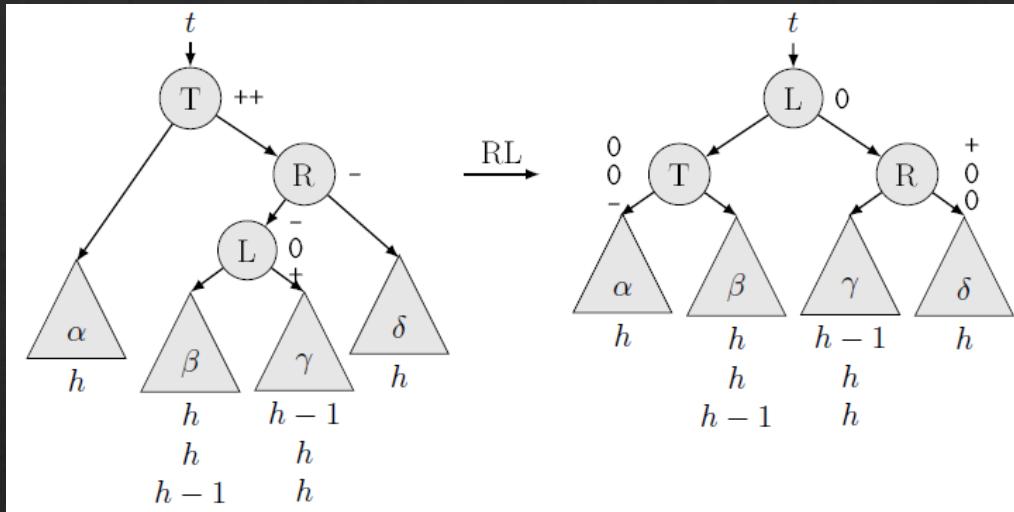
❖ $(++, +)$ forgatás lépései



```
balancePPP( &t, r : Node* )
    t → right := r → left
    r → left := t
    r → b := t → b := 0
    t := r
```

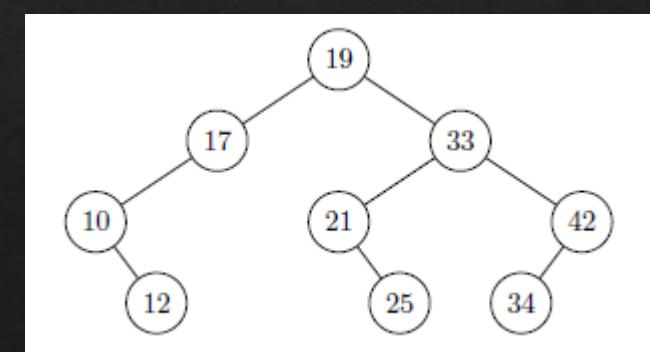
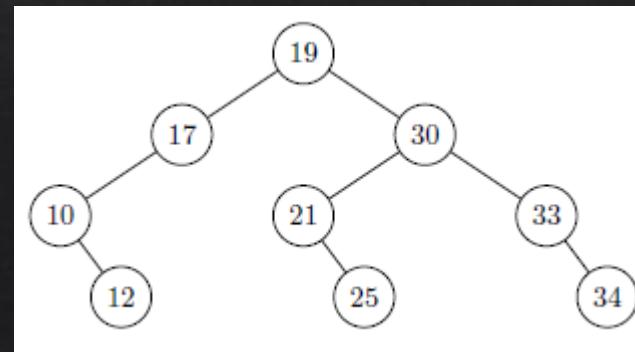
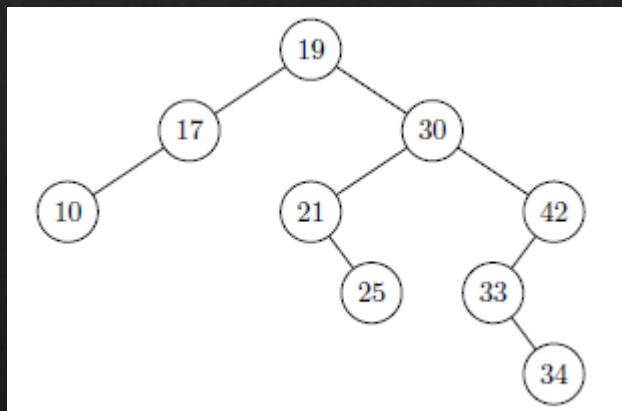
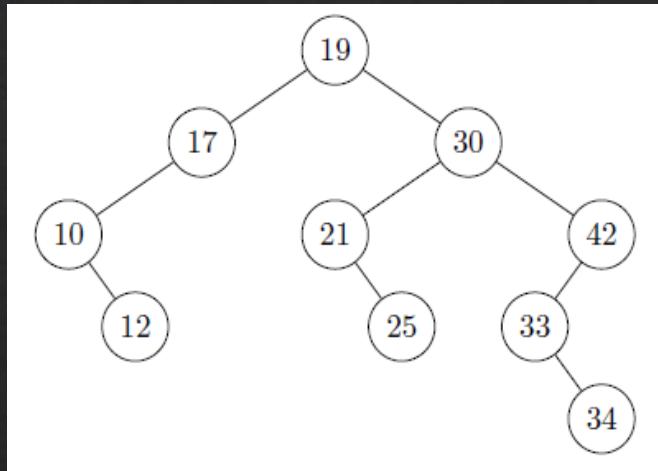


❖ Nézzük végig egy (++,-) forgatás lépései is



Törlés AVL fából

- ❖ Ismételjük át a bináris keresőfából való törlés algoritmusát! A törlés esetei:
 - ❖ Levél törlése: 12
 - ❖ Egy gyerek nélküli belső pont törlése: 42
 - ❖ Két gyerek nélküli belső pont törlése: 30 (jobb részfából a legkisebb elem kerül a helyére)

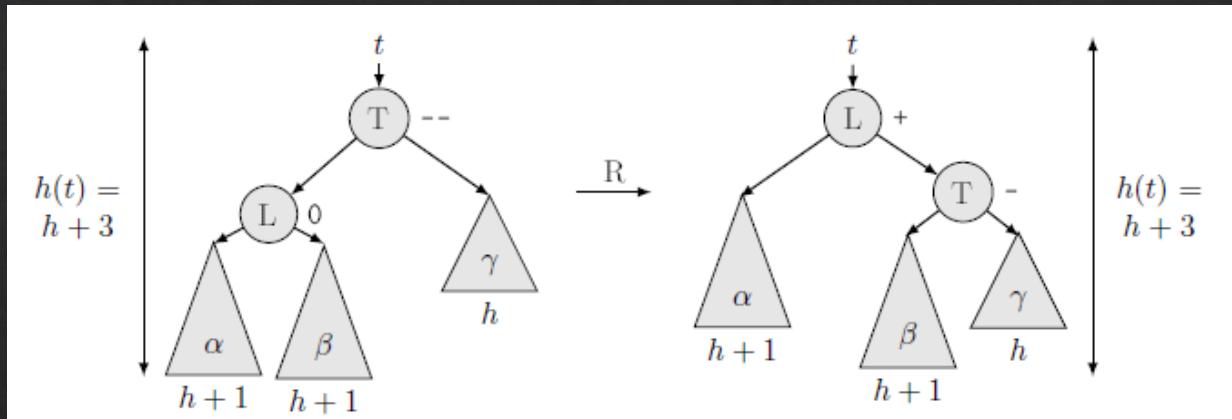


AVL törlés

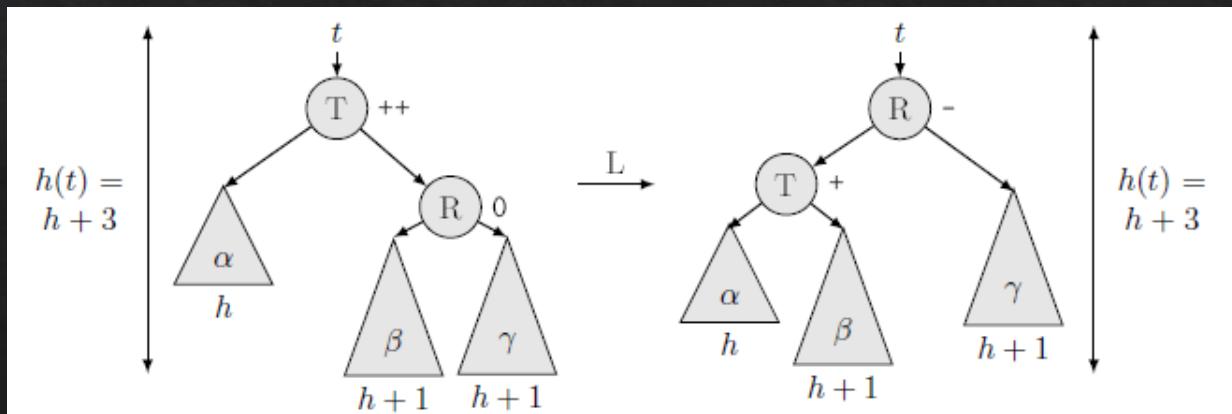
- ❖ Elvégezzük a törlést a bináris keresőfából való törlés algoritmusa szerint.
- ❖ A törléstől a gyökér felé indulva átcímkézzük a csúcsokat. Ahol az AVL tulajdonság elromlik, a tanult forgatási séma szerint elvégezzük a forgatást.
- ❖ A beszúrás esetei itt is előfordulhatnak $(++,+)$ $(--, -)$ $(++, -)$ $(--, +)$
- ❖ Két új forgatási sémára is szükség lehet: $(++,=)$ $(--, =)$
- ❖ Egy forgatás nem biztos, hogy elegendő!
- ❖ A $(++,+)$ $(--, -)$ $(++, -)$ $(--, +)$ sémák csökkentik a részfa magasságát, így a szülő csúcsban is szükség lehet forgatásra.
- ❖ Akár a gyökérig felfuthat a forgatási sorozat!

Új forgatási sémák

$(--,=)$ séma, törlés a γ részfából történt:



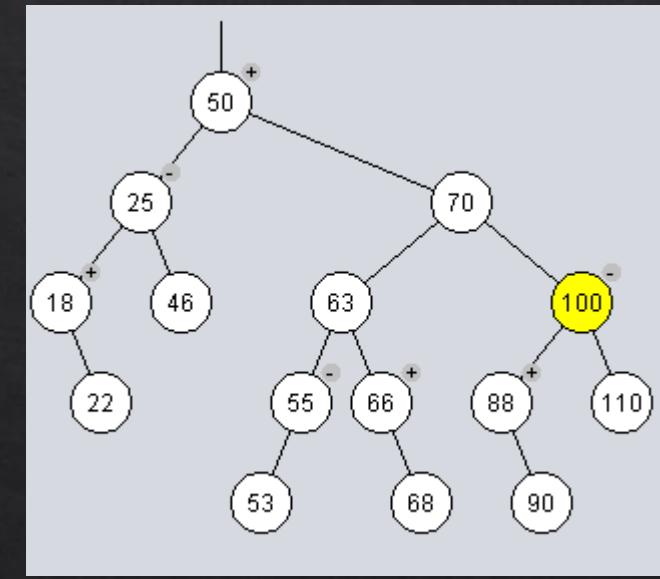
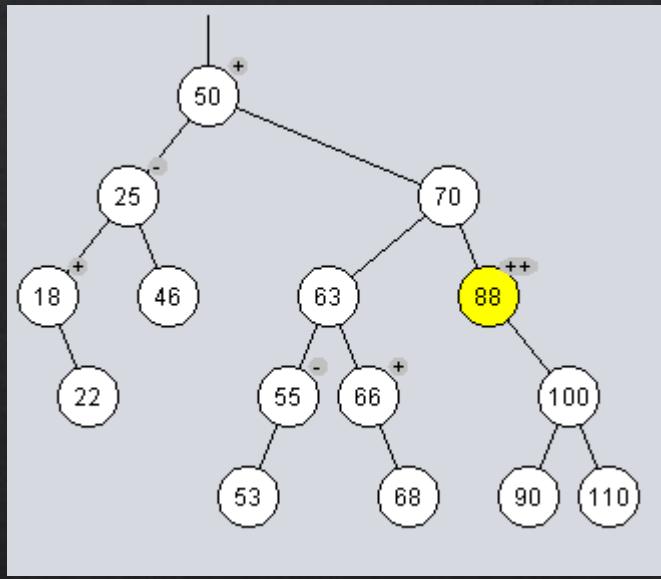
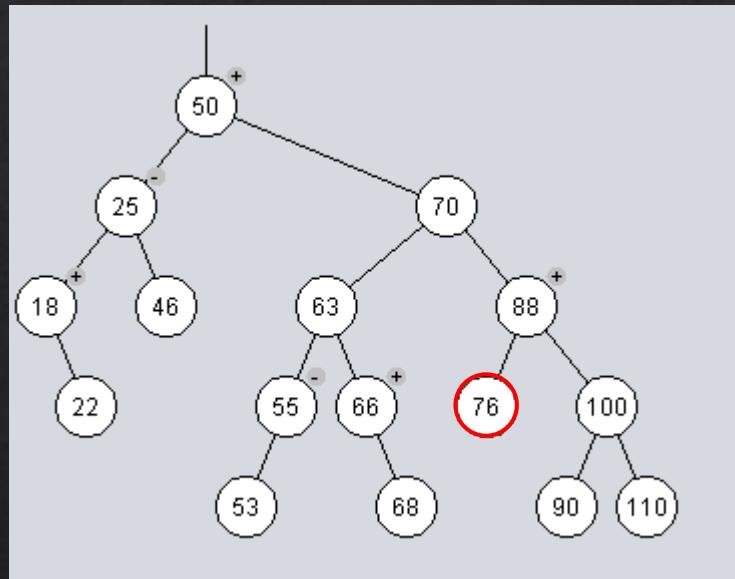
$(++,=)$ séma, törlés az α részfából történt:



Megjegyzés:
Mivel a forgatásban érintett fa magassága nem változik, így ezen forgatások után nem szükséges a szülő vizsgálata, biztosan nem lesz szükség újabb forgatásra.

Törlés, példák

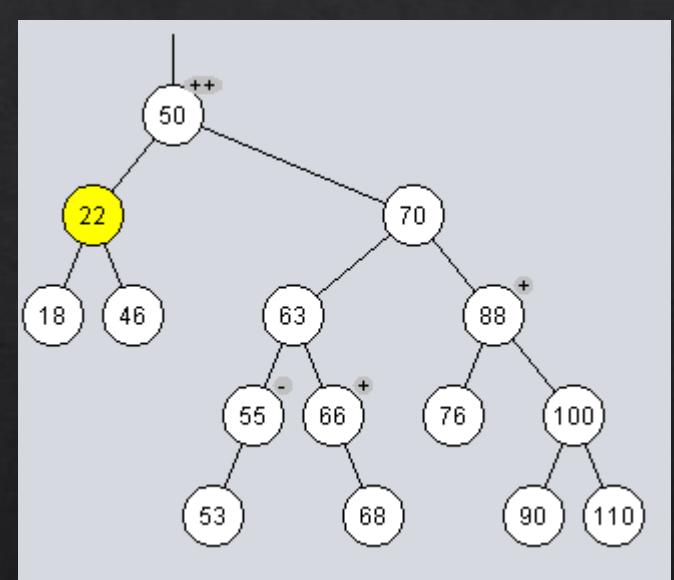
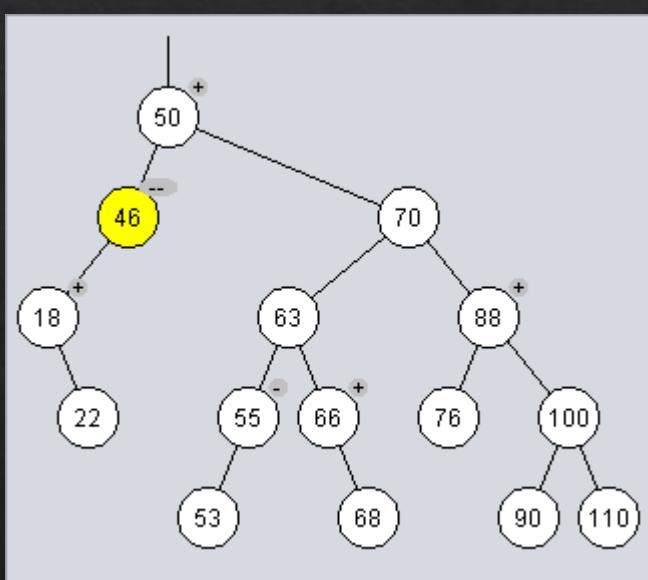
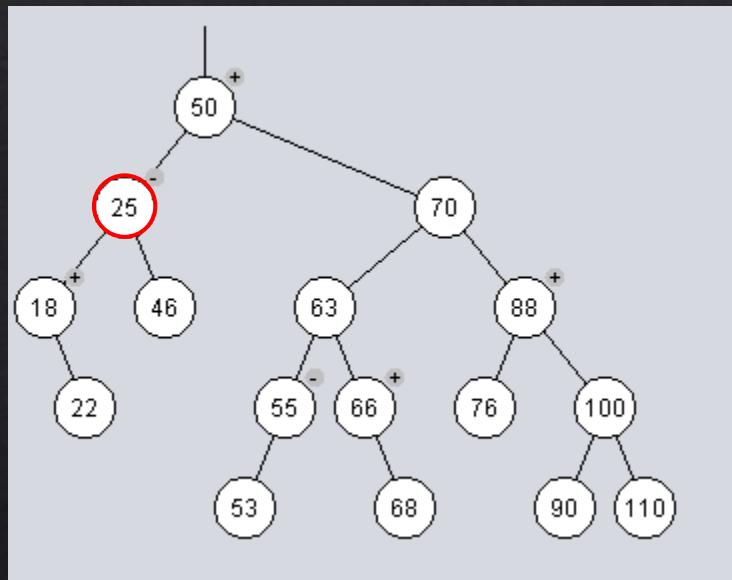
- ◆ Töröljük a 76-os kulcsot (++,=) séma szerint fordul a 88-as kulcsnál.



- ◆ Mivel a forgatott részfa magassága nem változott, a 70-es szülő csúcsnál nem keletkezhet újabb forgatás, és átsúlyozásra sem lesz szükség.

Törlés, több forgatás

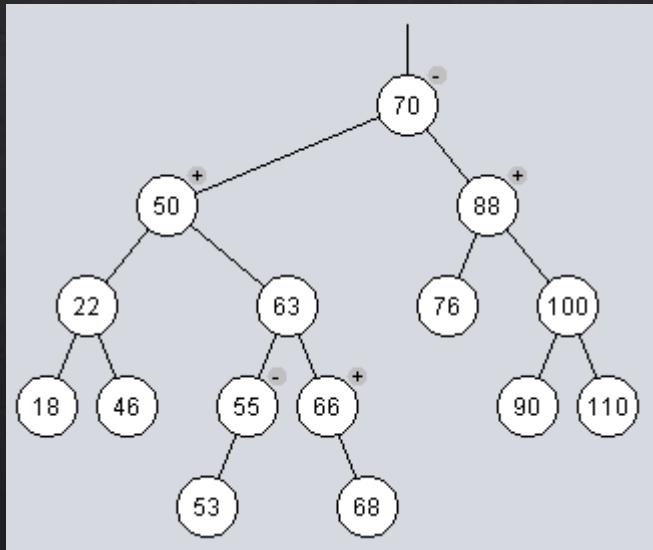
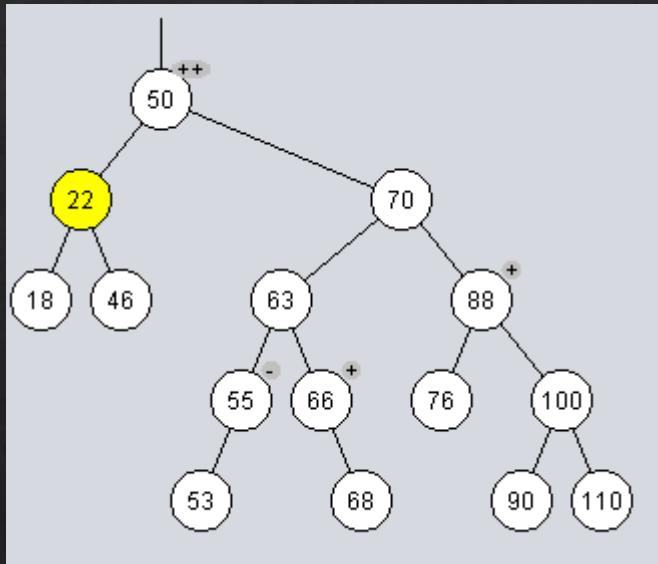
- ❖ Töröljük a 25-ös kulcsot: jobb részfa minimuma kerül a helyére.
- ❖ Első forgatás: $(-, +)$



- ❖ A $(-, +)$ forgatás csökkentette a forgatott részfa magasságát, így tovább gyűrűzik a forgatás a szülő felé

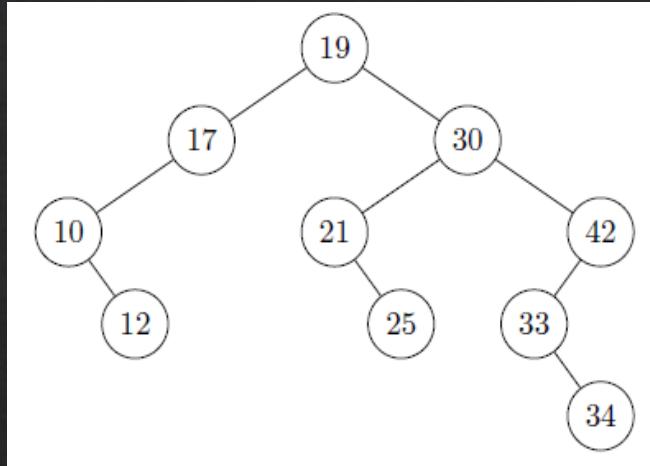
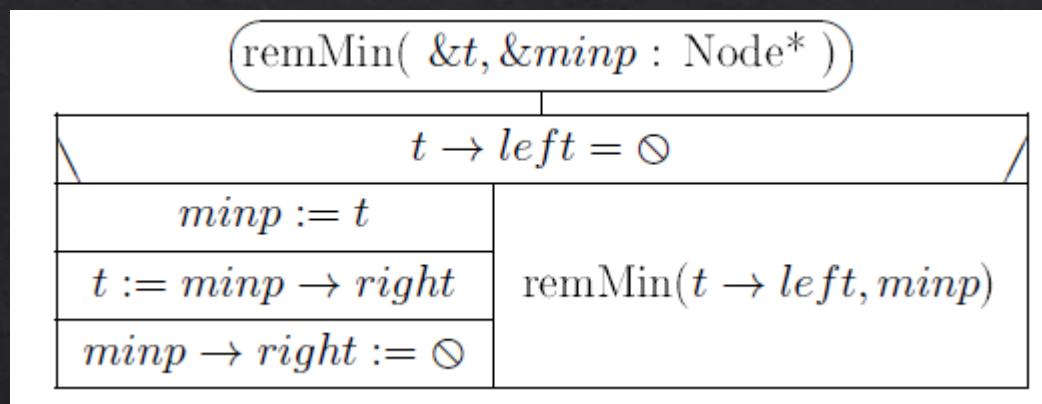
Törlés, több forgatás

- ❖ Gyökér bal részfájának magassága csökkent, így ++ lett a címkéje,
- ❖ Második forgatás: gyökérnél, (++,:) séma szerint, helyreállítja a fa AVL tulajdonságát.

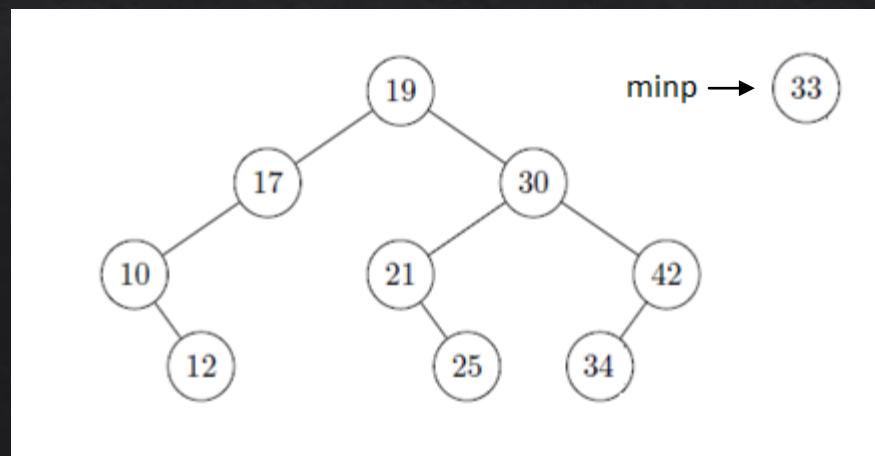


remMin bináris keresőfára (ismétlés)

- ◆ Bináris keresőfa remMin algoritmusa

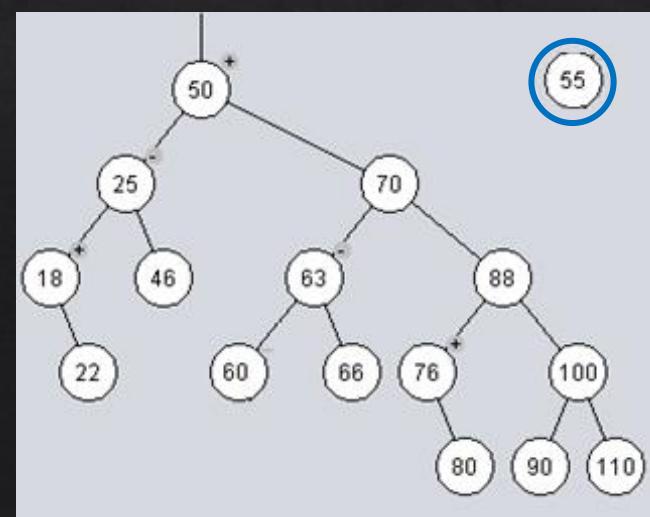
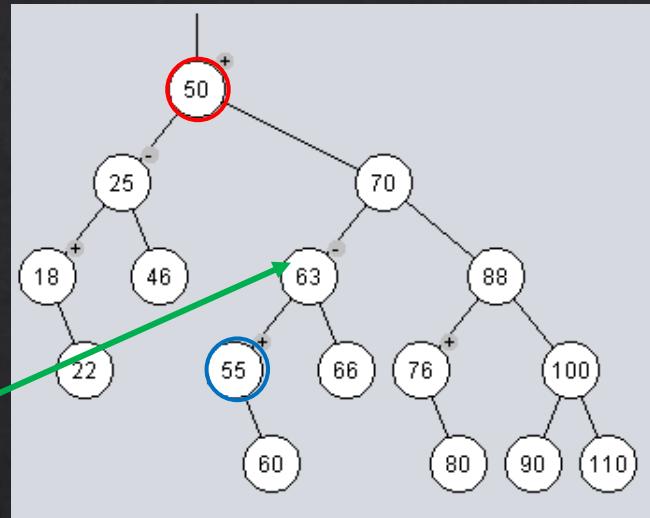
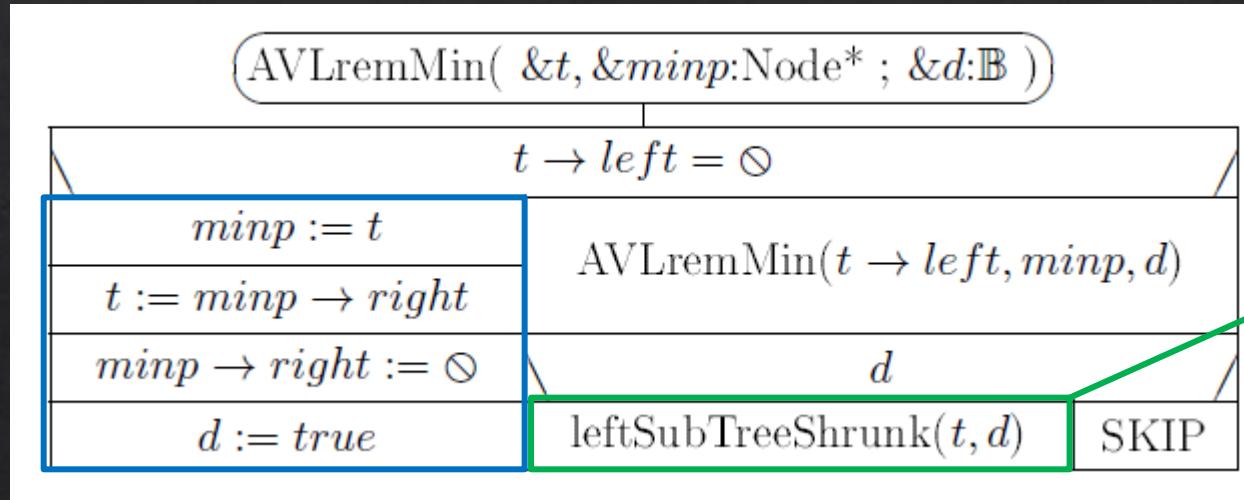


- ◆ Két gyerek nélküli belső pont törlésének segéd eljárása
- ◆ Például: 30-as kulcs törlésénél használjuk:
a 42 gyökerű részfára hívódik meg.



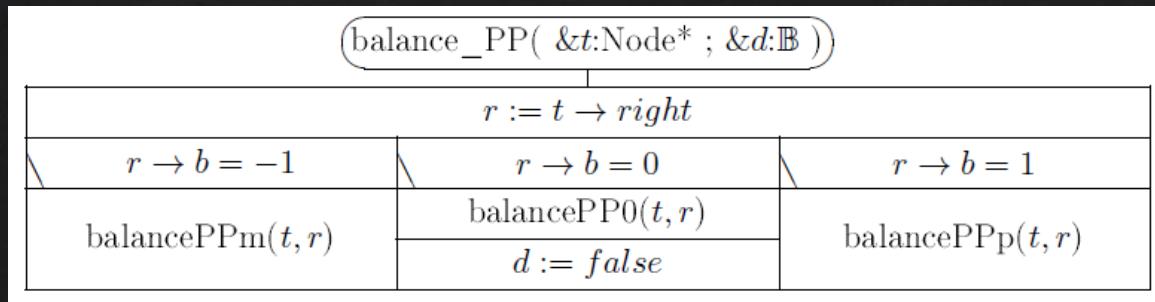
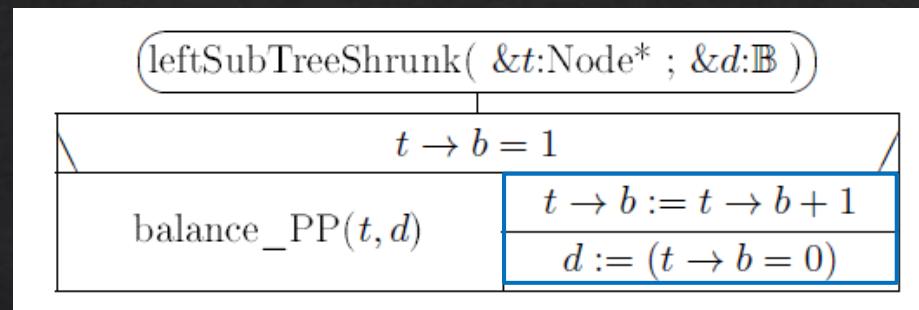
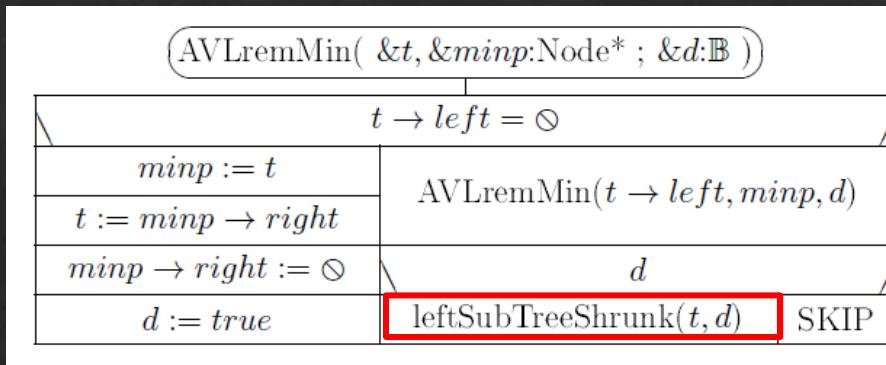
remMin AVL fára

- ◆ AVL fa remMin algoritmusa:

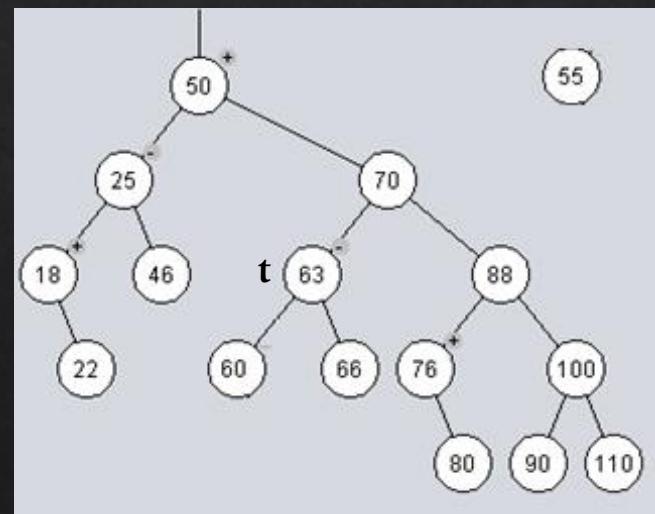


- ◆ Például: 50-es kulcs törlésénél használjuk, jobb részfájának mimimumát kifűzzük:
55-öt kifűzi a fából, d-t igazra állítja.
- ◆ A rekurzióból visszatérve, ha a bal részfa magassága csökken, meghívódik a leftSubTreeShrunk algoritmus

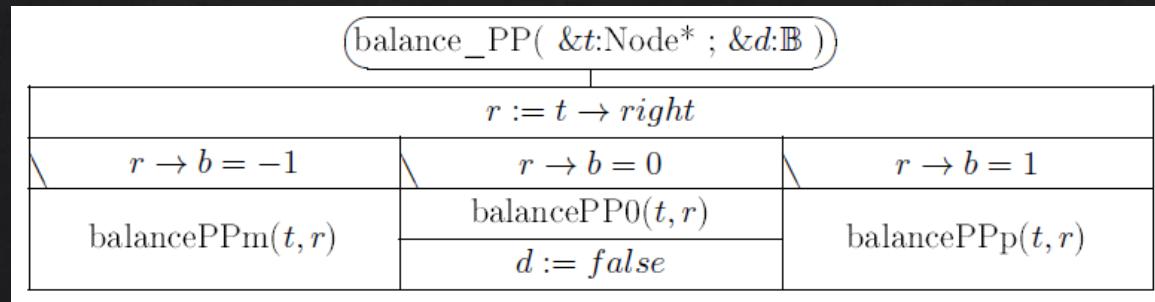
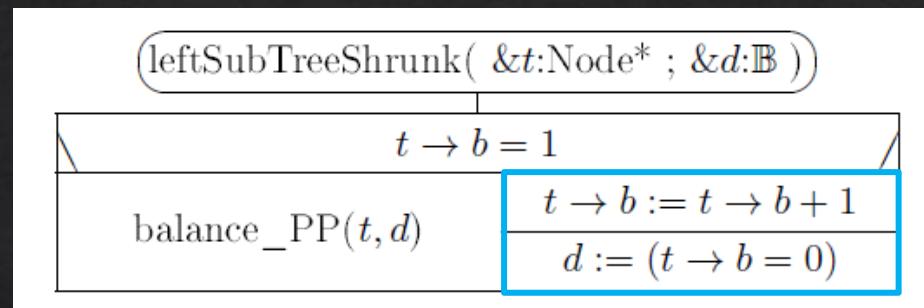
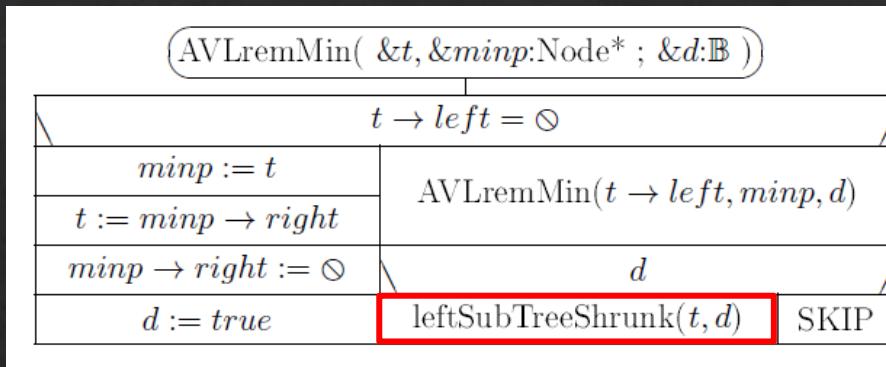
leftSubTreeShrunk



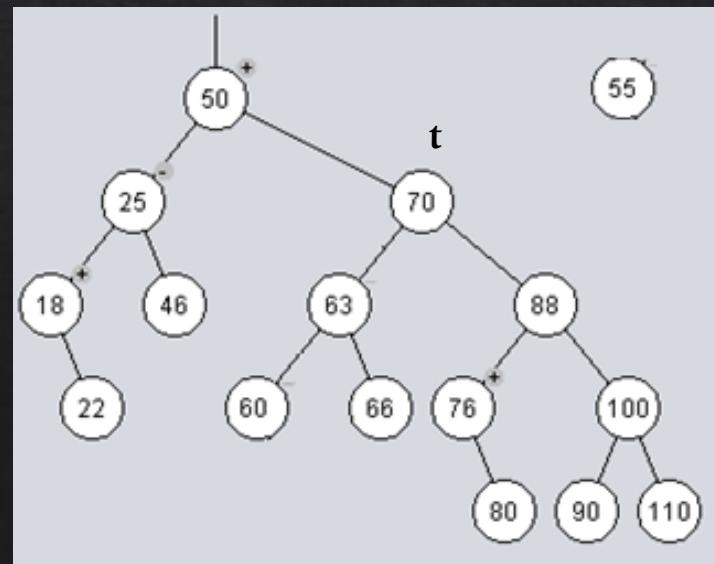
- ❖ Visszatérve a rekurzív hívásból t a 63-as csúcsra mutat, d igaz
- ❖ Meghívja „leftSubTreeShrunk” algoritmust,
- ❖ t balance=-1, tehát a jobb ágon fut,
- ❖ t balance=0 lesz, d tehát újból igazra állítódik, folytatódik az átcímkézés.



leftSubTreeShrunk

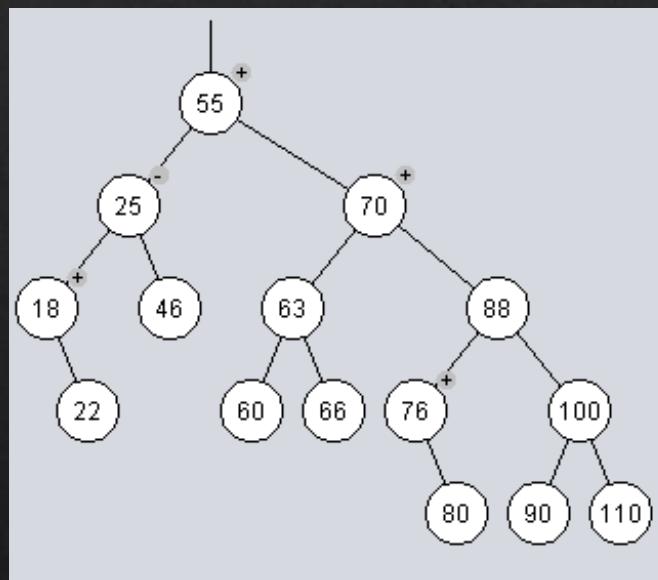
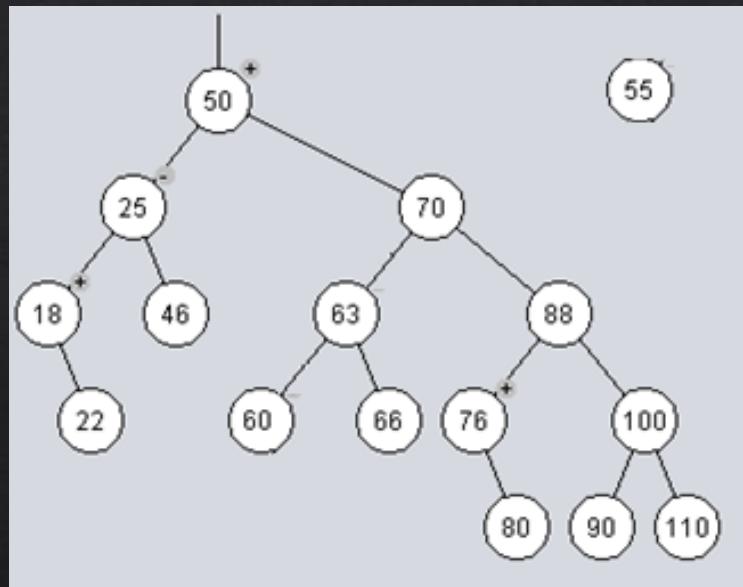


- ❖ Most t a 70-es csúcsra mutat, d igaz,
- ❖ Meghívja „leftSubTreeShrunk” algoritmust,
- ❖ t balance=0, tehát a jobb ágon fut, t balance=1 lesz, d tehát hamis: nem változott a 70-es részfa magassága, ilyenkor vége az átcímkézésnek.
- ❖ AVLremMin visszaért a hívás helyére, ezért itt vége az algoritmusnak!
- ❖ Még hátra van az 55 befűzése a 50-es helyére.



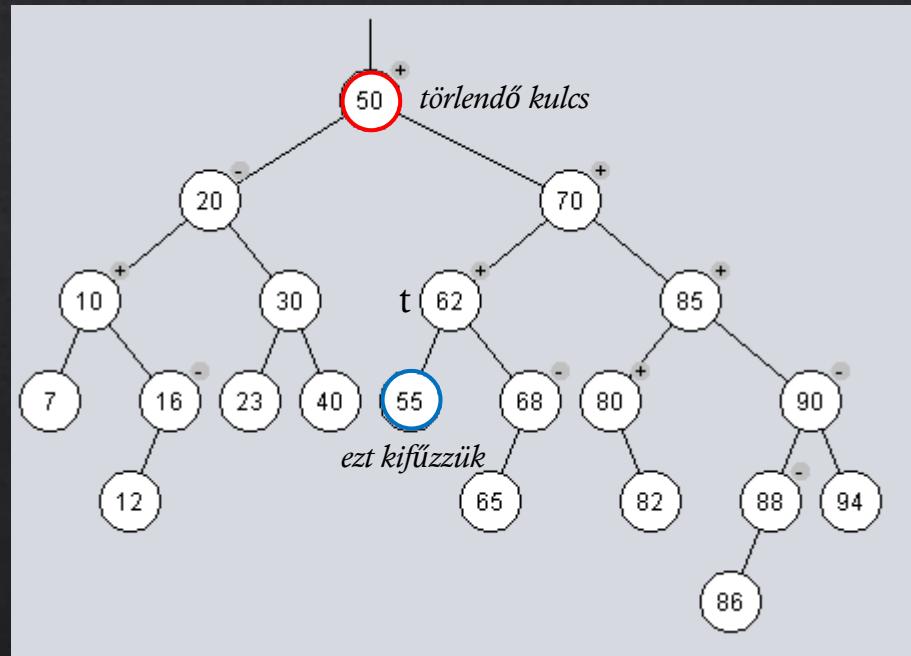
Végül az 50 törlése után kapott fa így fog majd kinézni

- ❖ 55 lett az új gyökér, balansz nem változik!
- ❖ A törlő algoritmus felsőbb szintjeinek működésére még visszatérünk, az AVLremMin egy kis részfeladat megoldását végzi.

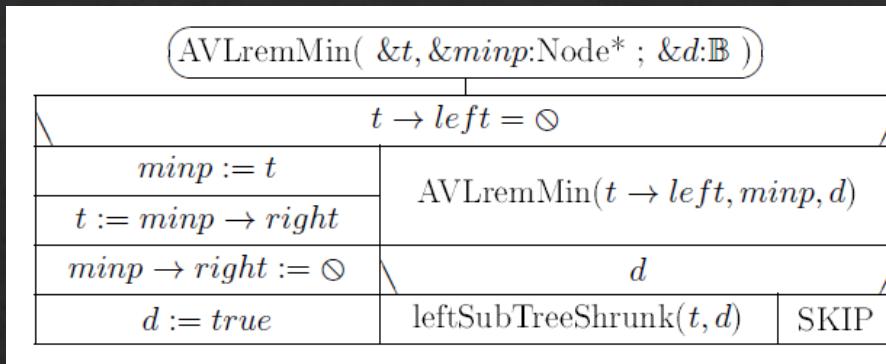


remMin egy olyan esetre, amikor forgatás is szükséges

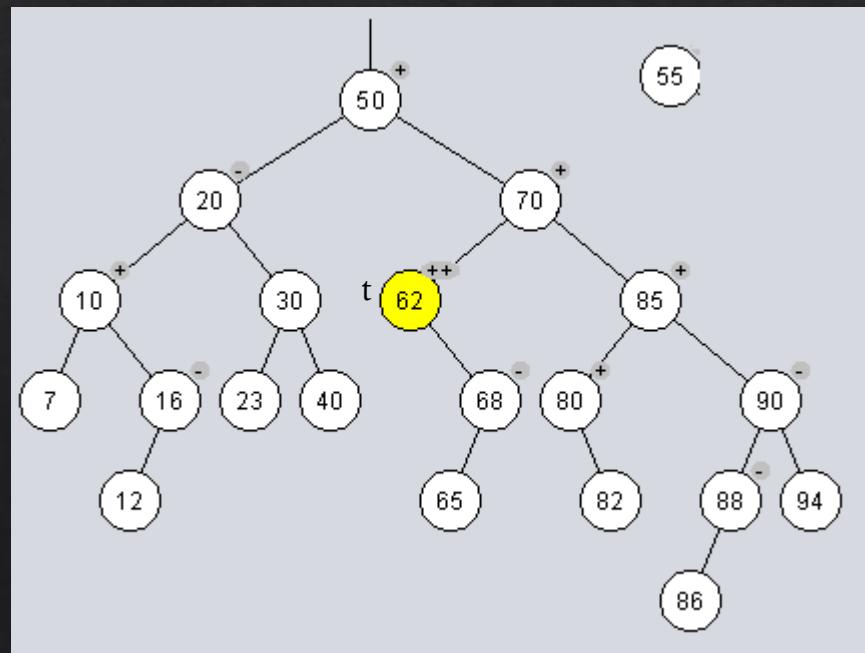
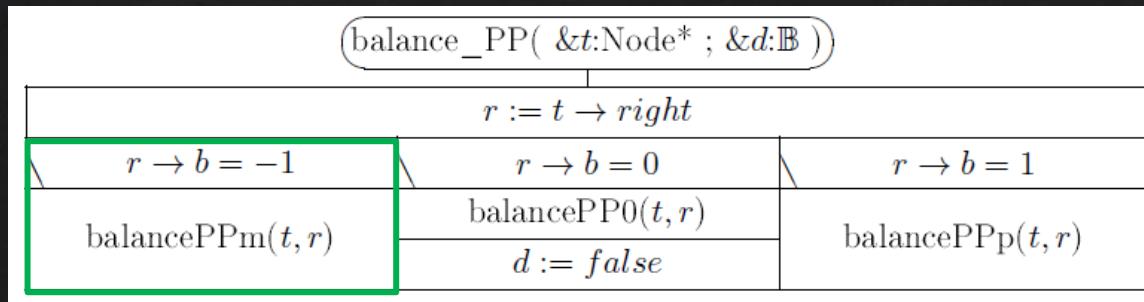
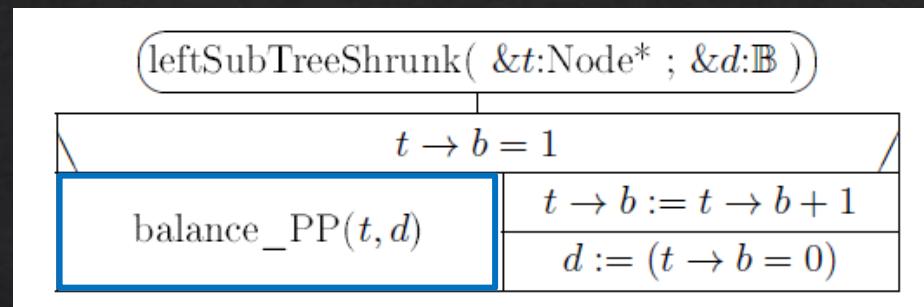
- ◆ A képen látható AVL fából a gyökér elemet, az 50-es kulcsot fogjuk törölni.
- ◆ Az AVLremMin algoritmus a `t->right` részfára hívódik meg, azaz a 70-es gyökerű részfára.
- ◆ Leszaladunk balra, és kifűzzük az 55-ös elemet, ami egy levél.
- ◆ Majd `t` a 62-es csúcsra mutat, és belépünk a `leftSubTreeShrunk` algoritmusba.
- ◆ Innen kövessük tovább a lépéseket.



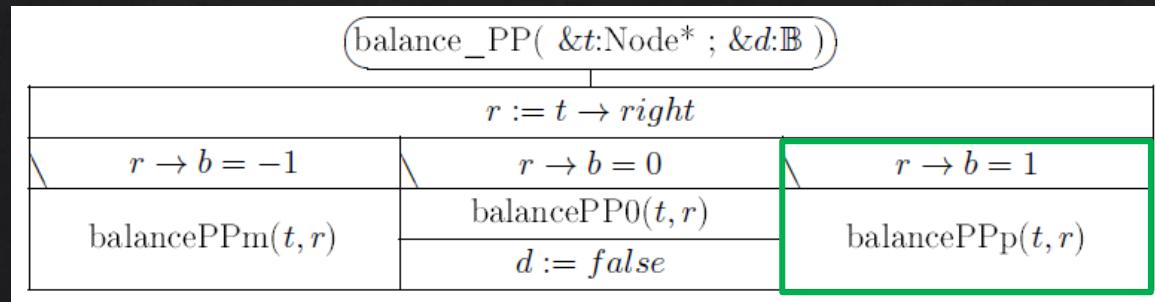
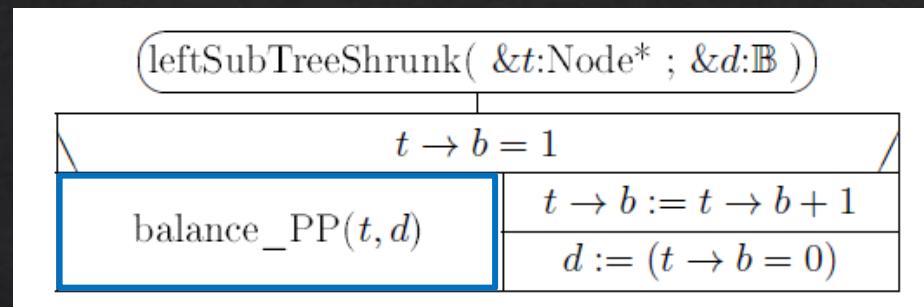
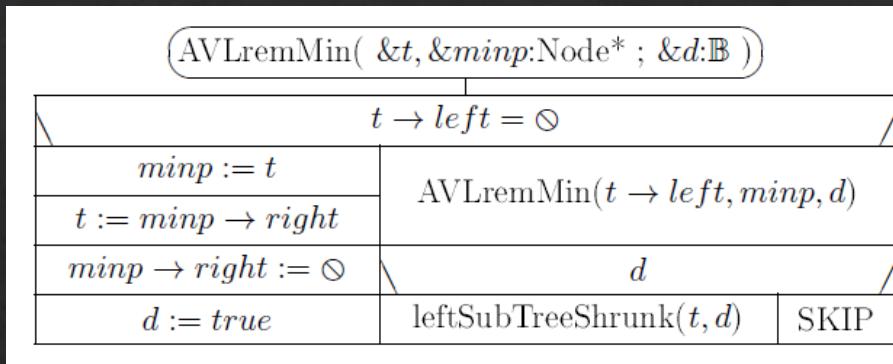
leftSubTreeShrunk



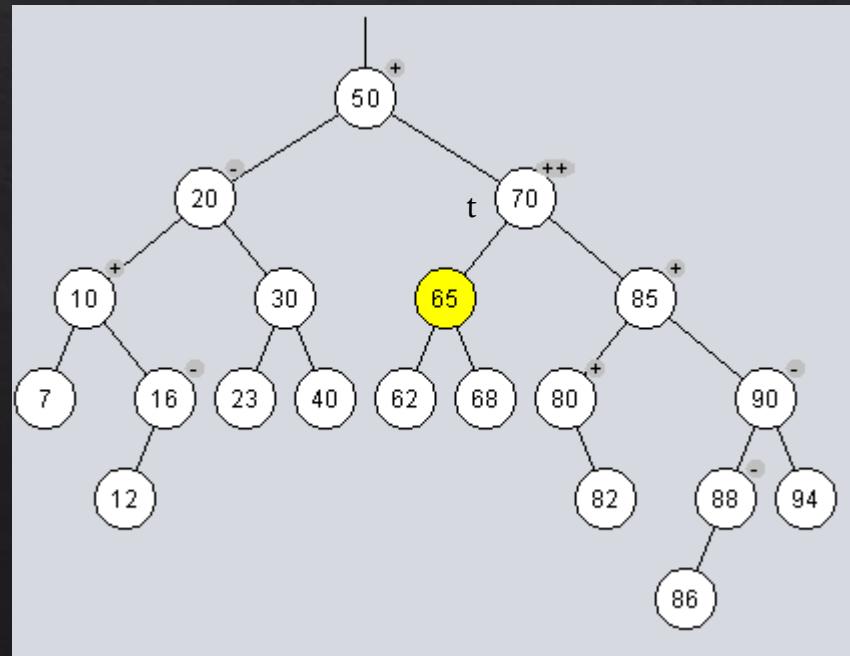
- ❖ Most t a 62-es csúcsra mutat, d igaz,
- ❖ Meghívja „leftSubTreeShrunk” algoritmust,
- ❖ t balance eredeti értéke 1, tehát a bal ágon fut, meghívódik balance_PP
- ❖ t jobb gyerekének balansz értéke -1,
- ❖ tehát végrehajtódik egy (++,-) forgatás



leftSubTreeShrunk

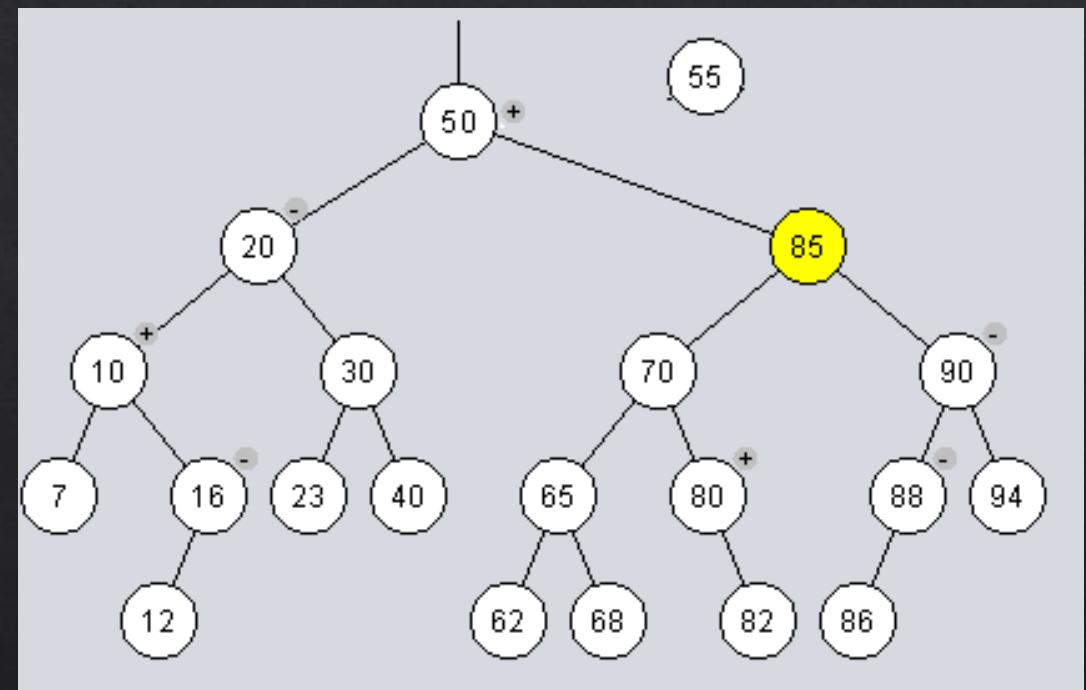


- ❖ Most t a 70-es csúcsra mutat, d igaz,
- ❖ Meghívja „leftSubTreeShrunk” algoritmust,
- ❖ t balance eredeti értéke 1, tehát a bal ágon fut, meghívódik `balance_PP`
- ❖ t jobb gyerekének balansz értéke 1, tehát végrehajtódik egy $(++, +)$ forgatás



Folytatás?

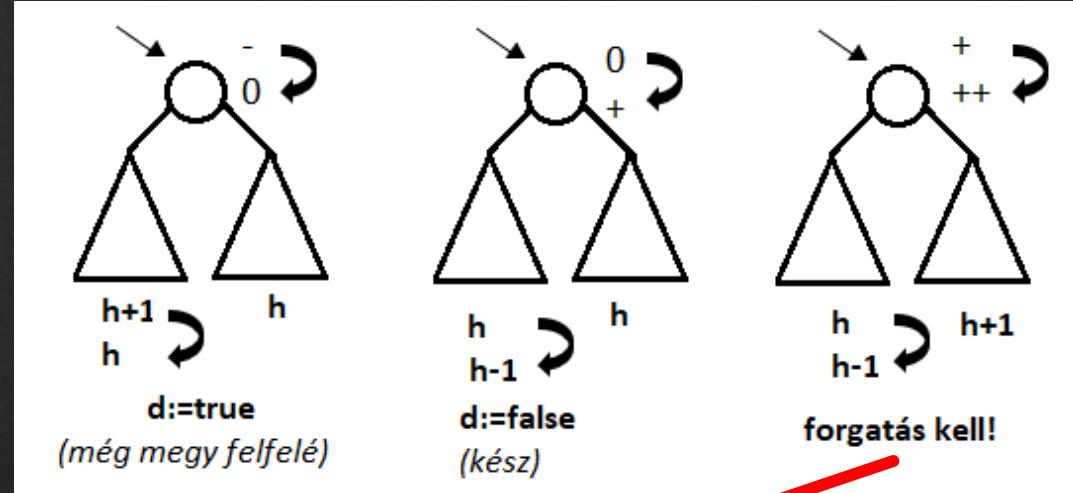
- ❖ Ezzel az AVLremMin algoritmus véget ért.
- ❖ De még nincs a helyén az 55-ös Node, és a fa gyökerének egyensúlya sincs rendben.
- ❖ Hogy ezeket az utolsó lépéseket is megértsük, meg kell nézni, hogyan működik a kulcs törlő algoritmus AVL fa esetén.



Előtte nézzünk meg egy összefoglaló ábrát a leftSubTreeShrunk és a balancePP algoritmusok működéséről

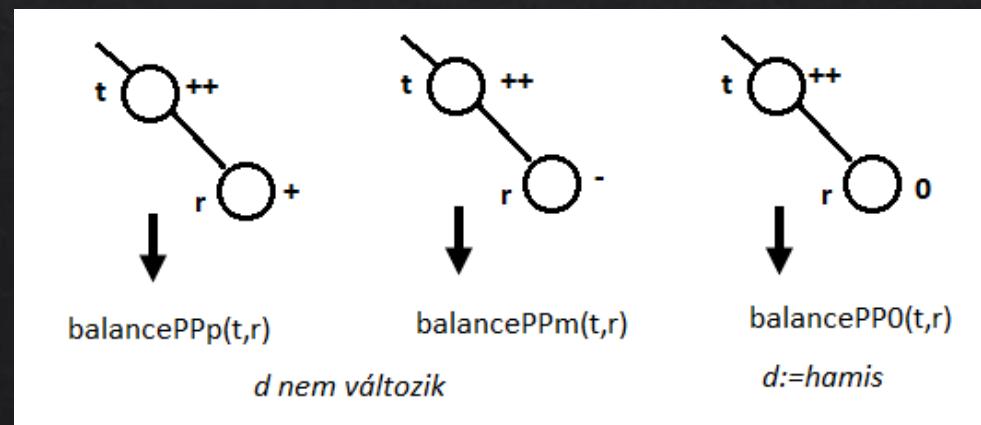
- ◊ leftSubTreeShrunk(&t:Node*; &d:B) esetei:

leftSubTreeShrunk(<i>&t:Node*</i> ; <i>&d:B</i>)		
<i>t</i> → <i>b</i> = 1		
balance_PP(<i>t, d</i>)	<i>t</i> → <i>b</i> := <i>t</i> → <i>b</i> + 1	
	<i>d</i> := (<i>t</i> → <i>b</i> = 0)	



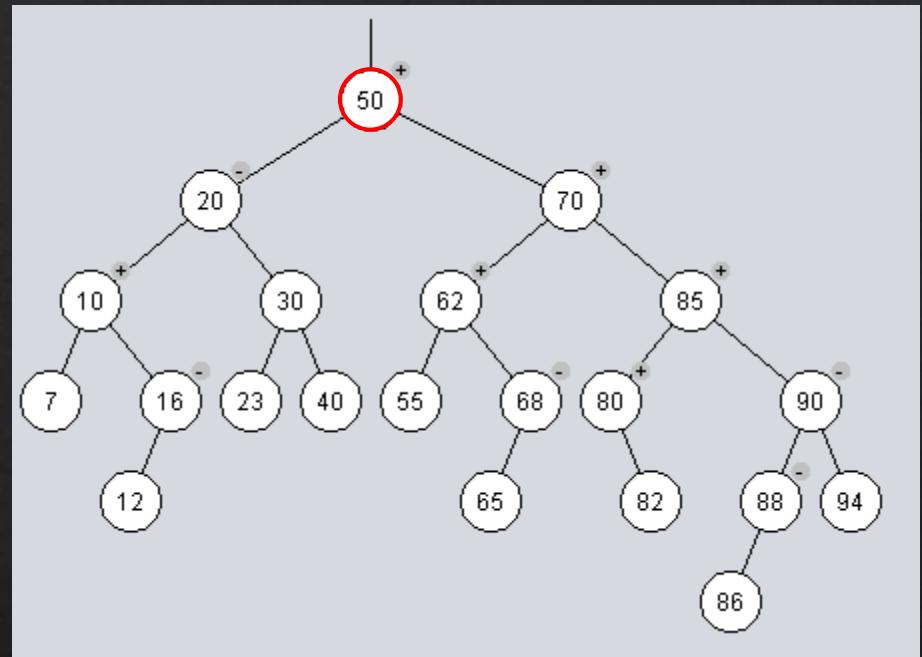
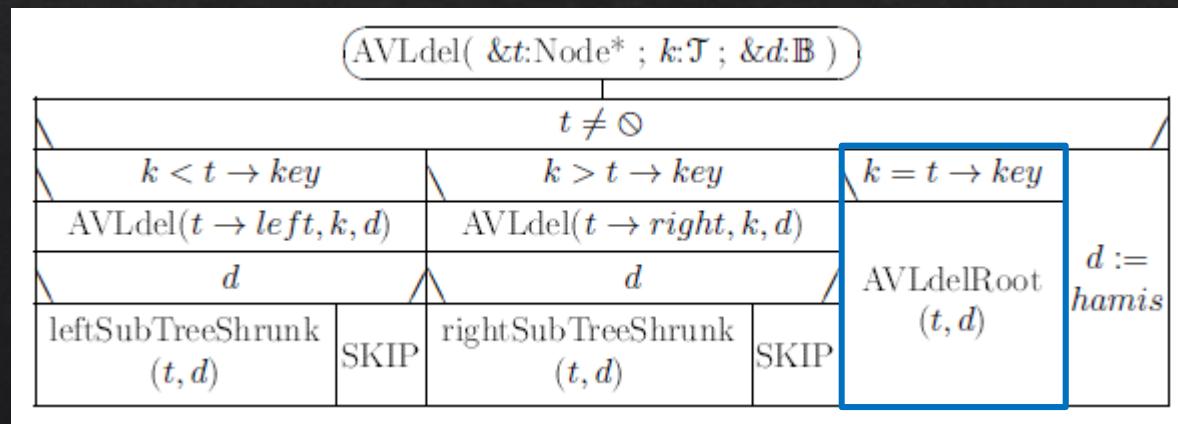
- ◊ balance_PP(&t:Node*; &d:B) esetei:

balance_PP(<i>&t:Node*</i> ; <i>&d:B</i>)		
<i>r</i> := <i>t</i> → <i>right</i>		
<i>r</i> → <i>b</i> = -1	<i>r</i> → <i>b</i> = 0	<i>r</i> → <i>b</i> = 1
balancePPm(<i>t, r</i>)	balancePP0(<i>t, r</i>)	balancePPp(<i>t, r</i>)



AVLdel algoritmus

- ❖ Kulcs törlés esetén megkeressük a kulcsot a fában,
 - ❖ Ha sikerült a keresés, az AVLdelRoot következik,
 - ❖ A példában rögtön a gyökérben megtaláltuk a törlendő kulcsot.
 - ❖ Kövessük nyomon, mi fog történni.

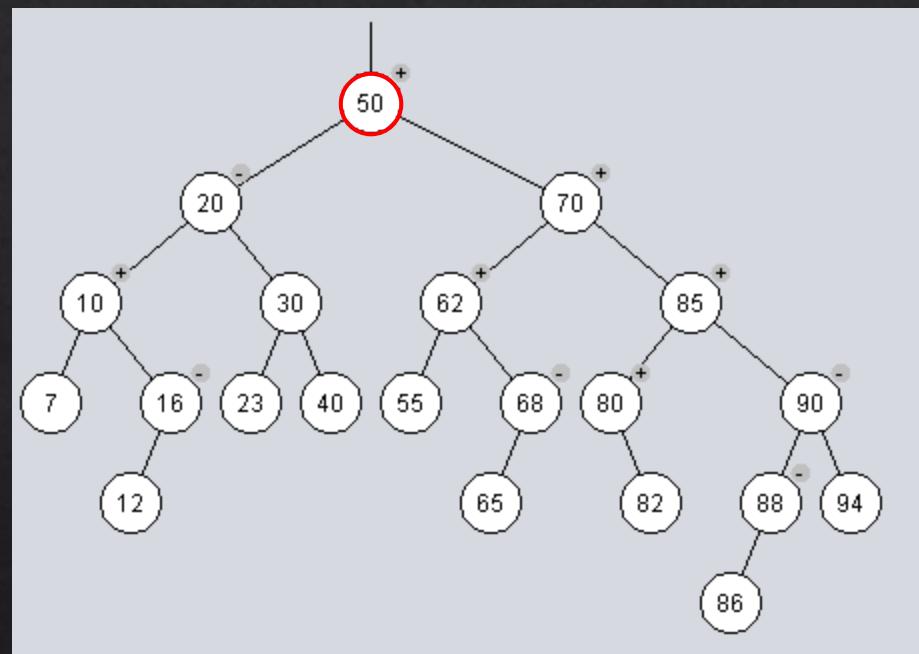


AVLdelRoot algoritmus

- ❖ A harmadik ágat kell követnünk, mert az 50-es Node két gyerkes.
- ❖ Következik a rightSubTreeMinToRoot algoritmus!

AVLdelRoot(&t:Node* ; &d:B)			
t → left = Ø	t → right = Ø	t → left ≠ Ø ∧ t → right ≠ Ø	
p := t	p := t	rightSubTreeMinToRoot(t, d)	
t := p → right	t := p → left		d
delete p	delete p		
d := true	d := true	rightSubTreeShrunk(t, d)	SKIP

rightSubTreeMinToRoot(&t:Node* ; &d:B)	
AVLremMin(t → right, p, d)	
<i>p → left := t → left ; p → right := t → right ; p → b := t → b</i>	
delete t ; t := p	

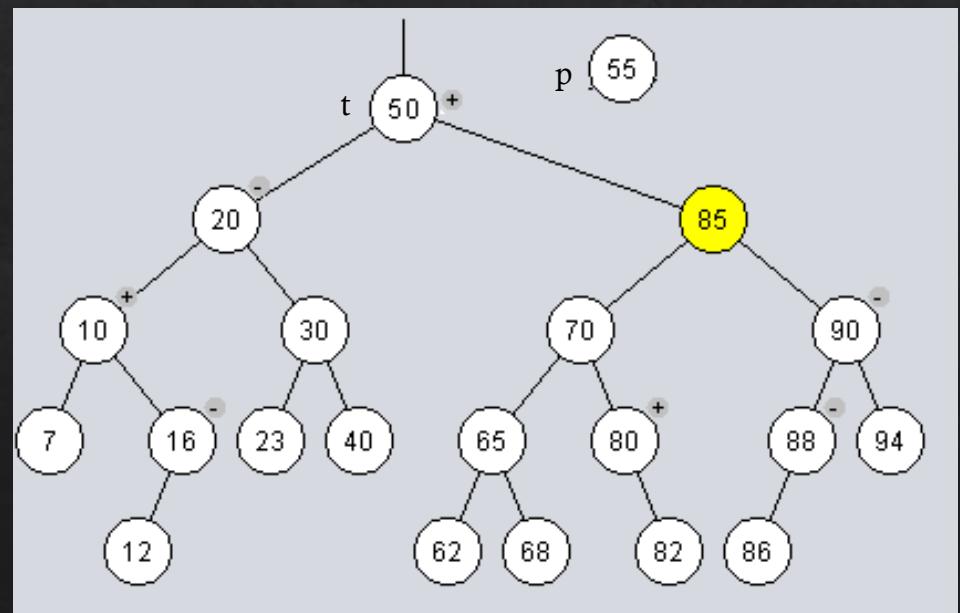


rightSubTreeMinToRoot algoritmus

- ◆ Itt történt tehát az előbb végig követett AVLremMin hívás (piros keret), az 50-es jobb részfájára, melynek eredményét látjuk az ábrán. A d: bool változó igaz értékű.
- ◆ A kék keretben lévő utasítások befűzik az 55-ös Node-ot az 50 helyére, a balansz +1 lesz! Majd törlődik az 50-es Node, és a fa gyökerére mutató t pointerbe bekerül az 55-ös Node címe.
- ◆ Hátra van még a gyökér balanszának a kiszámítása. Mivel d igaz, a rightSubTreeShrunk az új gyökérre is lefut.

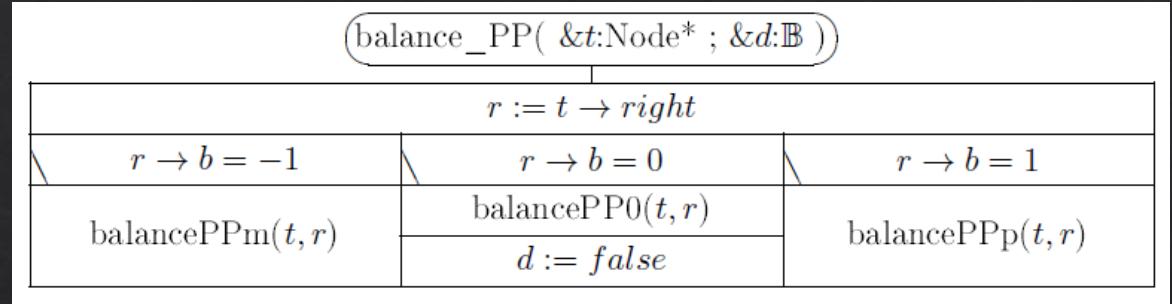
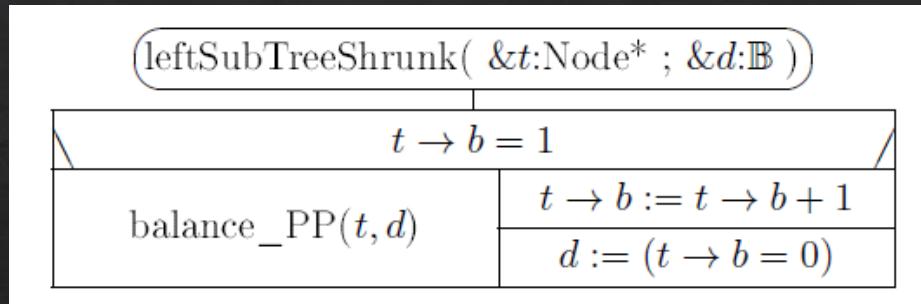
AVLdelRoot(&t:Node* ; &d:B)		
$t \rightarrow left = \emptyset$	$t \rightarrow right = \emptyset$	$t \rightarrow left \neq \emptyset \wedge t \rightarrow right \neq \emptyset$
$p := t$	$p := t$	rightSubTreeMinToRoot(t, d)
$t := p \rightarrow right$	$t := p \rightarrow left$	
delete p	delete p	
$d := true$	$d := true$	d
		rightSubTreeShrunk(t, d)
		SKIP

rightSubTreeMinToRoot(&t:Node* ; &d:B)		
AVLremMin($t \rightarrow right, p, d$)		
$p \rightarrow left := t \rightarrow left$	$p \rightarrow right := t \rightarrow right$	$p \rightarrow b := t \rightarrow b$
delete t ; $t := p$		



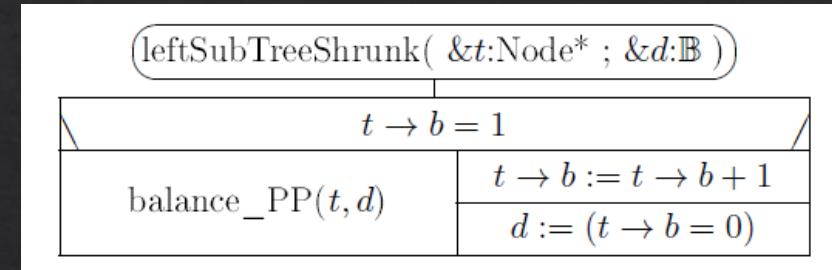
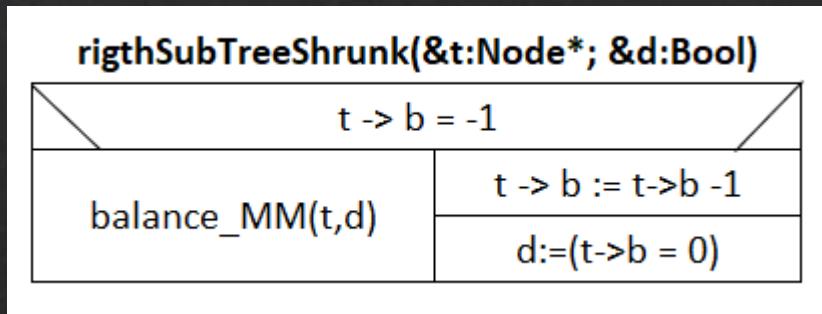
rightSubTreeShrunk

- ❖ A jegyzet feladatként tűzi ki, hogy a leftSubTreeShrunk mintájára dolgozzuk ki az algoritmust!



- ❖ Ha a *t* gyökerű részfa jobb részfájának magassága csökkent, akkor forgatásra akkor lesz szükség, ha *t* csúcs -1 balanszú volt (ilyenkor -- címkéjű lett a *t* Node).
- ❖ Ha *t* csúcs 0 balanszú volt, és a jobb részfa magassága csökkent, akkor -1 balanszú lett, de a magasságát megtartotta, így *d* hamis kell legyen.
- ❖ Ha *t* csúcs +1 balanszú volt, akkor 0 balanszú lett, de a magasság csökkent, így *d* igaz kell legyen.

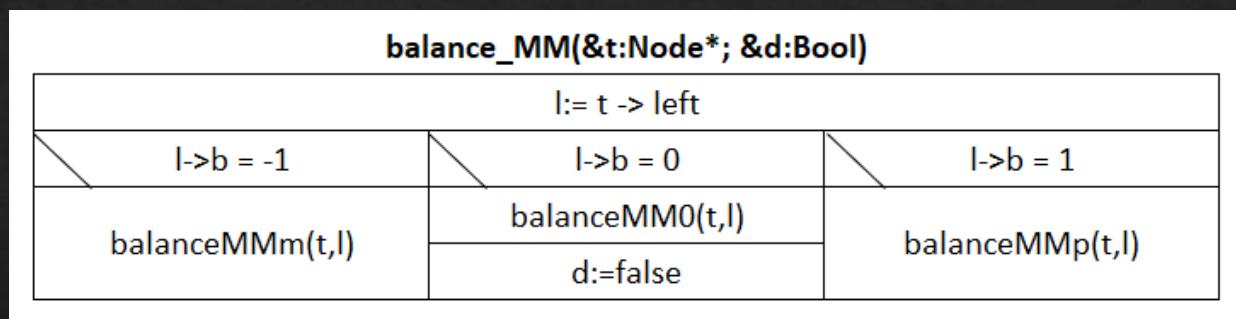
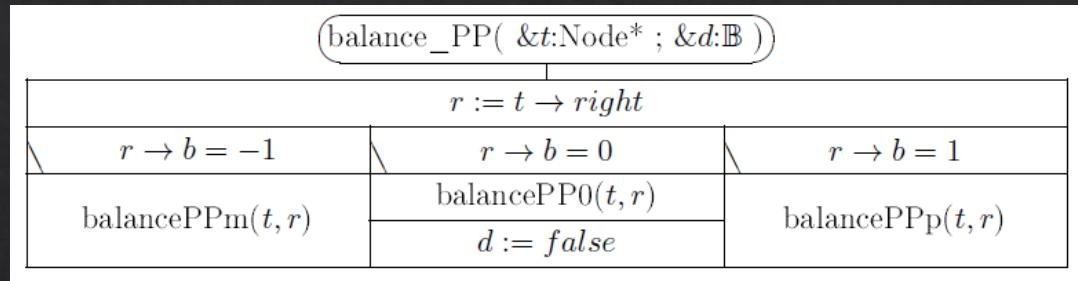
rightSubTreeShrunk



- ❖ Bal ág: a t gyökerű részfa jobb részfájának magassága csökkent, ha t csúcs -1 balanszú volt, akkor - - címkéjű lett a t Node, tehát a megfelelő forgatást meg kell majd hívni.
- ❖ Jobb ág: csökkentjük t csúcs balansz értékét eggyel. Ha 0 balanszú volt, akkor -1 lesz, ha 1 balanszú, akkor 0 balanszú lesz.
- ❖ Majd d-t igazra/hamisra állítjuk. Ha a csúcs balansza +1-ről 0-ra csökkent, akkor a részfa magassága is csökkent, tehát feljebb is vizsgálódni kell, d legyen ilyenkor igaz.
- ❖ Ha nulláról -1 –re változott, akkor a magassága nem változott, tehát nem kell tovább vizsgálódni.

balance_MM

- ❖ Balance_PP mintájára pedig könnyen elkészíthető a balance_MM algoritmus

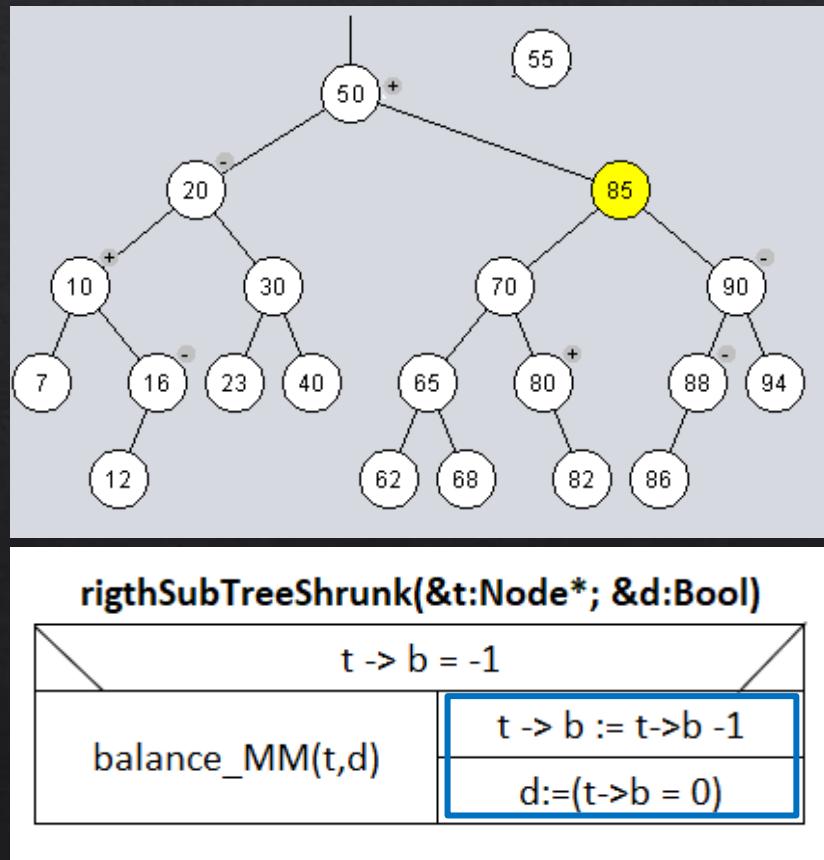


- ❖ Az 1 pointerbe a bal gyerek címét tesszük,
- ❖ Ha 1 balansa -1, akkor (-,-,-) forgatást kell végrehajtani, ez a részfa magasságát csökkenti, tehát d igaz marad,
- ❖ Ha 1 balansza 0, akkor (-,-,0) forgatást hajtunk végre, a magasság nem csökken, tehát d legyen hamis,
- ❖ Ha 1 balansza 1, akkor (-,-,+) forgatást kell végrehajtani, d igaz marad, mert ilyenkor is csökken a magasság

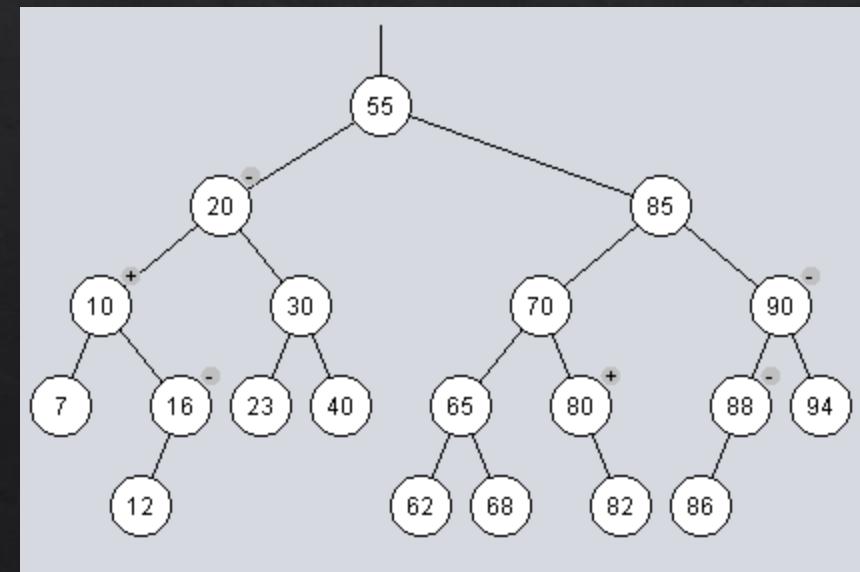
Az 50-es kulcs
törlése
befejeződött!

Itt hagytuk abba...

- ◆ A gyökér helyére befűztük az 55-öt, balansa 1, de lefut a rightSubTreeShrunk a gyökérre!



- ◆ Így a kapott fa gyökerének balansa 0-ra módosul.



Szorgalmi házi feladat

- ❖ A szorgalmi házi feladatokat minden a következő heti gyakorlat előtt lehet beküldeni a CANVAS felületen keresztül. Lehetőleg elektronikusan készítsük, de beküldhető kézzel írt, lefényképezett megoldás is, ez esetben ügyeljünk az olvashatóságra és a kép minőségére.
- 1. A gyakorló feladathoz hasonlóan készítsünk olyan beszúrási sorozatot, mely bemutatja minden a négy forgatási séma alkalmazását legalább egyszer. Ügyeljünk a következőkre: ne minden a gyökérnél forduljon a fa, és a forgatásban részt vevő csúcsoknak legyenek leszármazott részfáik!

Házi feladat:

Tekintsük át a jegyzetben a beszúrás strogramjait!

Gyakorló feladatok

- ❖ Rajzoljunk le konkrét kulcsokkal egy F6 Fibonacci fát. Melyek azok az elemek, melynek törlése a gyökérig felfutó forgatási sorozatot eredményez? Hajtsunk is végre egy ilyen forgatási sorozatot!
- ❖ Az alábbi szöveges leírás alapján rajzoljuk le az AVL fát, írjuk a csúcsok mellé a balansz értékeket. Hajtsuk végre az AVLremMin algoritmust a fa gyökerére, majd a kapott AVL fába szúrjuk be az 5-ös kulcsot. Adjuk meg ábrával és szöveges leírással is az AVLremMin, illetve beszúrás után kapott AVL fát. A szöveges megadásban ábrázoljuk a belső csúcsok balanszát is {+, -, =} karakterekkel.
{ [(2) 4 ({6} 8 {10})] 12 [14 (16)] }
- ❖ Szúrjuk be egy üres AVL fába az alább megadott kulcsokat, ebben a sorrendben. A beszúrások után, ha szükséges forgassuk meg a tanult módon a fát.
12 14 13 10 5 70 80 18 50 60 3 65
Majd a fából töröljük ebben a sorrendben a 10, 80, 50, 18 és 13 kulcsokat.
A törlések után is szemléltessük a szükséges forgatásokat, adjuk meg típusaikat!

Szorgalmi házi feladat

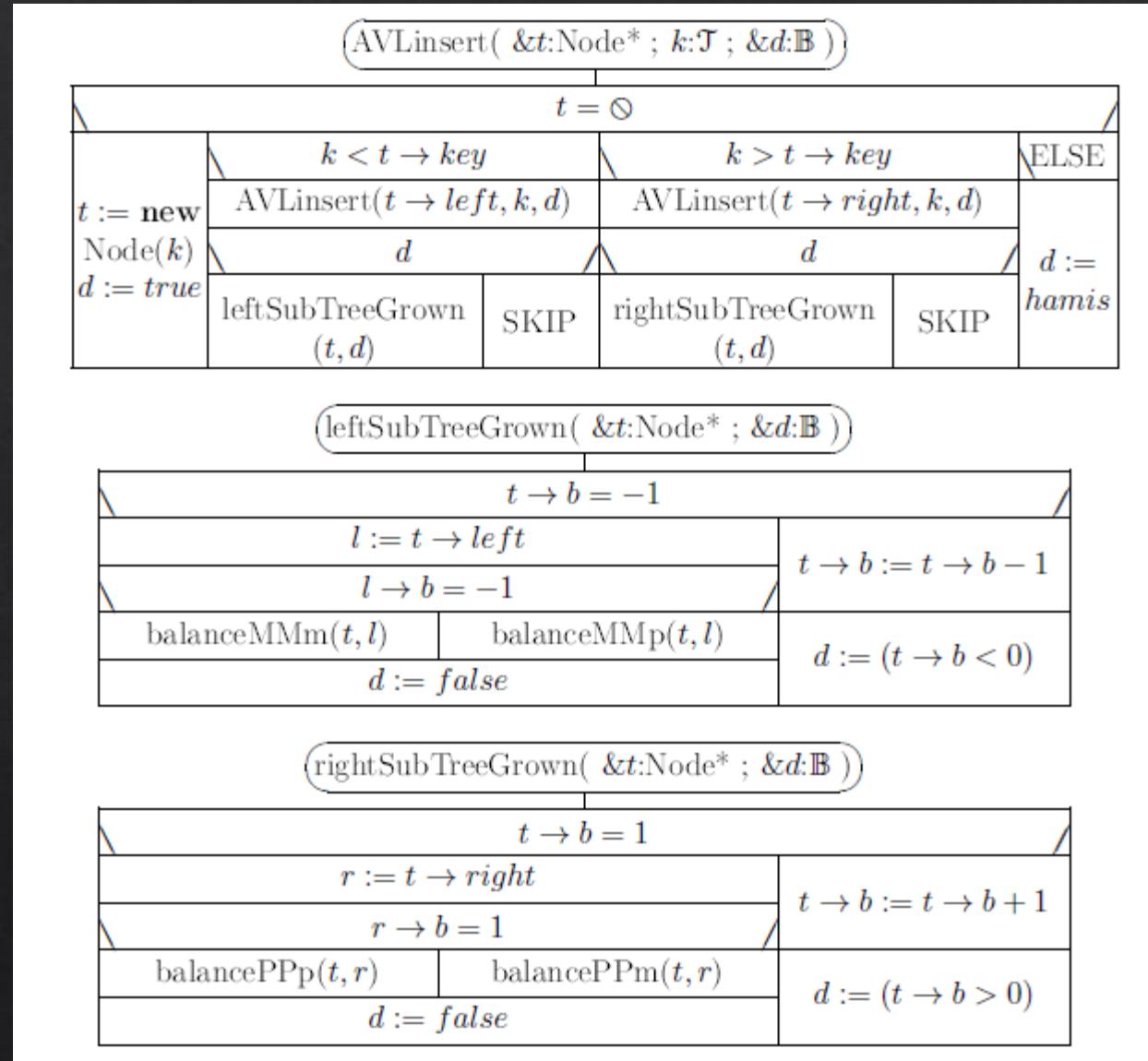
- ❖ A szorgalmi házi feladatokat minden a következő heti gyakorlat előtt lehet beküldeni a CANVAS felületen keresztül. Lehetőleg elektronikusan készítsük, de beküldhető kézzel írt, lefényképezett megoldás is, ez esetben ügyeljünk az olvashatóságra és a kép minőségére.
1. Tudjuk, hogy két gyerekes Node törlése esetén a bal részfából kiemelt legnagyobb kulccsal is helyettesíthetjük a törlendő elemet. Készítsük el az AVLremMax algoritmust, mely a részfa legnagyobb elemét kifűzi, és paraméterben visszaadja a címét.
 2. Készítsük el a balanceMM0 forgató algoritmust.

Mintául használjuk a megfelelő tanult algoritmusokat! Ha szükséges, a jegyzettben találhatő illetve az órán elkészített segéd eljárásokat felhasználhatjuk.

3+2 pont

Beszúrás felső szintje

Beszúrás helyétől visszafelé haladva, az egyensúlyozás jelzésének módosítása, ha ++, vagy – egyensúly keletkezne, a megfelelő forgatás elindítása



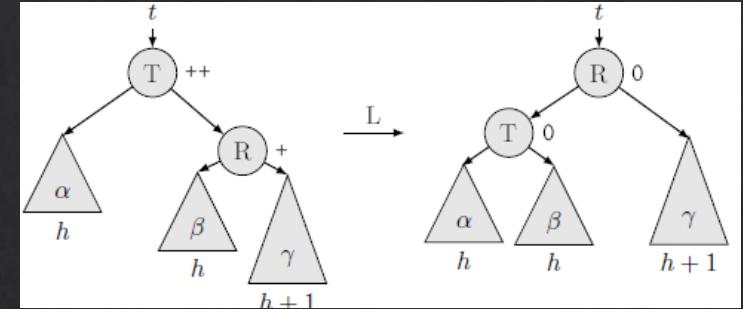
$(++, +)$ és $(--, -)$ forgatások

balancePPP($\&t, r : \text{Node}^*$)

$t \rightarrow \text{right} := r \rightarrow \text{left}$
$r \rightarrow \text{left} := t$
$r \rightarrow b := t \rightarrow b := 0$
$t := r$

balanceMMm($\&t, l : \text{Node}^*$)

$t \rightarrow \text{left} := l \rightarrow \text{right}$
$l \rightarrow \text{right} := t$
$l \rightarrow b := t \rightarrow b := 0$
$t := l$



$(++, -)$ és $(--, +)$ forgatások

balancePPm($\&t, r : \text{Node}^*$)

$l := r \rightarrow \text{left}$
$t \rightarrow \text{right} := l \rightarrow \text{left}$
$r \rightarrow \text{left} := l \rightarrow \text{right}$
$l \rightarrow \text{left} := t$
$l \rightarrow \text{right} := r$
$t \rightarrow b := -\lfloor (l \rightarrow b + 1)/2 \rfloor$
$r \rightarrow b := \lfloor (1 - l \rightarrow b)/2 \rfloor$
$l \rightarrow b := 0$
$t := l$

balanceMMP($\&t, l : \text{Node}^*$)

$r := l \rightarrow \text{right}$
$l \rightarrow \text{right} := r \rightarrow \text{left}$
$t \rightarrow \text{left} := r \rightarrow \text{right}$
$r \rightarrow \text{left} := l$
$r \rightarrow \text{right} := t$
$l \rightarrow b := -\lfloor (r \rightarrow b + 1)/2 \rfloor$
$t \rightarrow b := \lfloor (1 - r \rightarrow b)/2 \rfloor$
$r \rightarrow b := 0$
$t := r$

