Algoritmusok és adatszerkezetek II.
előadásjegyzet: Fák

Ásványi Tibor – asvanyi@inf.elte.hu

2025. szeptember 11.

Tartalomjegyzék

Hivatkozások

[1] Ásványi Tibor, Algoritmusok és adatszerkezetek II. előadásjegyzet: Bevezetés, Jelölések, Tematika

[2] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., **magyarul:** Új Algoritmusok, *Scolar Kiadó*, Budapest, 2003. ISBN 963 9193 90 9
angolul: Introduction to Algorithms (Third Edititon), *The MIT Press*, 2009.

[3] Fekete István, Algoritmusok jegyzet http://ifekete.web.elte.hu/

[4] Rónyai Lajos – Ivanyos Gábor – Szabó Réka, Algoritmusok, *TypoTEX Kiadó*, 1999. ISBN 963 9132 16 0 https://www.tankonyvtar.hu/hu/tartalom/tamop425/2011-0001-526_ronyai_algoritmusok/adatok.html

[5] Weiss, Mark Allen, Data Structures and Algorithm Analysis, *Addison-Wesley*, 1995, 1997, 2007, 2012, 2013.

[6] Carl Burch, Ásványi Tibor, B+ fák

[7] Ásványi Tibor, Algoritmusok és adatszerkezetek I. előadásjegyzet

[8] Wirth, N., Algorithms and Data Structures, *Prentice-Hall Inc.*, 1976, 1985, 2004.
magyarul: Algoritmusok + Adatstruktúrák = Programok, *Műszaki Könyvkiadó*, Budapest, 1982. ISBN 963 10 3858 0

1. AVL fák (AVL trees)

Az előző félévben tanultunk a bináris keresőfákról, mint (a hasító táblák mellett) adathalmazok, szótárak reprezentációjára szolgáló adatszerkezetekről. Láttuk, hogy a tipikus szótárműveletek (mint egy kulcs – és a hozzá tartozó adatok – keresése, beszúrása és eltávolítása) átlagos esetben megfelelő hatékonyságúak, mert a futási idő legfeljebb a fa magasságával arányos, ennek várható értéke pedig $\Theta(\log n)$, ahol n a fa mérete, azaz csúcsainak száma, és a $\log n$ függvény nagyon lassan nő.

A legrosszabb esetben azonban a bináris keresőfa (BST) magassága $n-1$, így a szótárműveletek sem elég hatékonyak. Ezen segít a kiegyensúlyozott keresőfák (balanced search trees) alkalmazása, mert ezeknél a fa magassága minden esetben $\Theta(\log n)$, ami – megfelelő implementáció mellett – garantálja a szótárműveletek megfelelő hatékonyságát.

A különböző fajta kiegyensúlyozott bináris keresőfák (balanced BSTs) klasszikus típusai a piros-fekete [2] és az AVL fák. Ez utóbbiakat tárgyaljuk itt. Amennyiben az alkalmazásunkban lényegesen több a szótárban való keresés, mint az azt módosító művelet, az AVL fák alkalmazása előnyösebb, mert a piros-fekete fáknál az AVL fák erősebben kiegyensúlyozottak. Az ideális a majdnem teljes bináris keresőfák alkalmazása lehetne (ui. ezek magassága minimális), de ezek módosítására nem ismerünk hatékony megoldásokat. Az AVL rövidítés mögött *Georgy Adelson-Velsky* és *Evgenii Landis* szovjet matematikusok nevét fedezhetjük fel, akik 1963-ban dolgozták ki koncepciójukat.

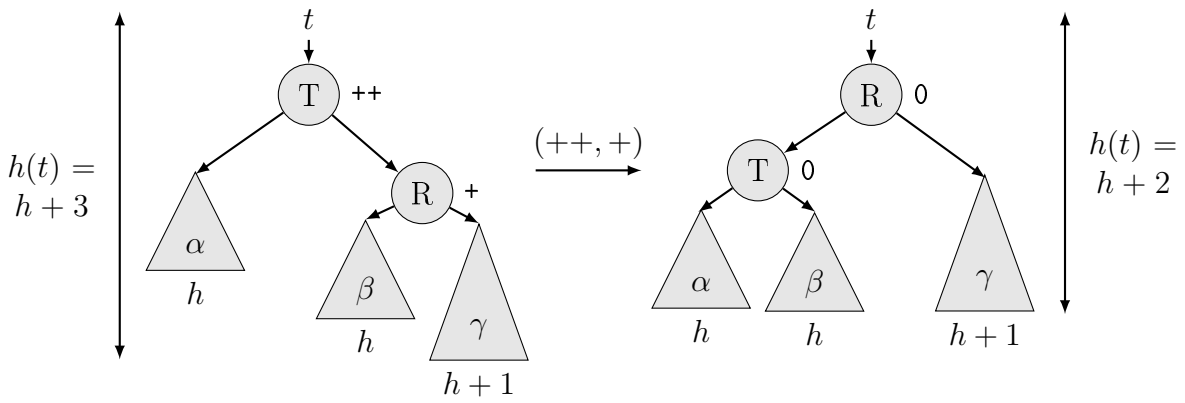
Az AVL fa módosításával járó műveletek, mint pl. a beszúrás és a törlés, elronthatják a kiegyensúlyozottságot, amit – kiegyensúlyozó forgatások (*balancing rotations*) segítségével – még a művelet befejezése előtt helyre kell állítani.

A témával ismerősek kedvéért elöljáróban megjegyezzük, hogy az AVL fák kiegyensúlyozásának szabályai megtalálhatók az 1-6. ábrákon.

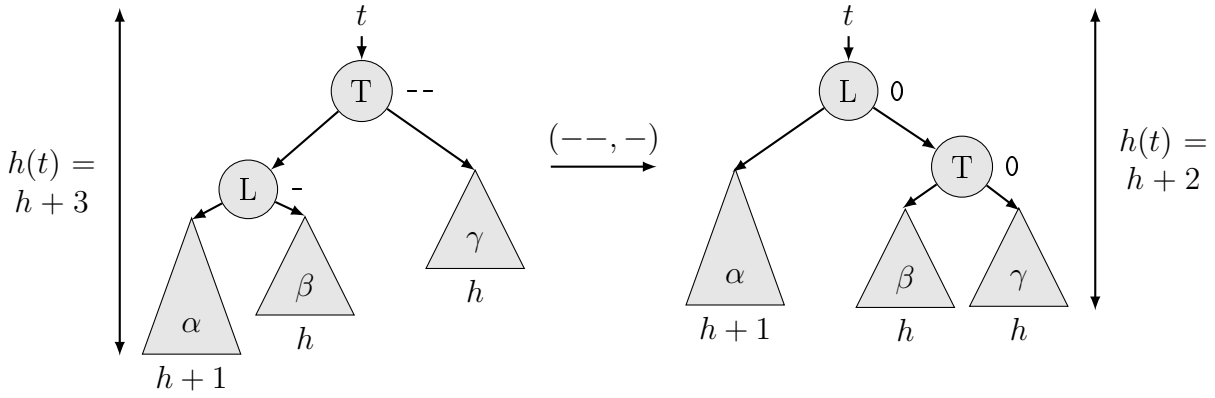
Ebben a fejezetben a továbbiakban részletesen tárgyaljuk az AVL fa fogalmát, tulajdonságait, egy különösen hatékony megvalósítását lehetővé tevő reprezentációját, és ennek megfelelően a műveleteit.

Az *AVL fák* magasság szerint kiegyensúlyozott bináris keresőfák (height-balanced BSTs). Egy bináris fa *magasság szerint kiegyensúlyozott*, ha minden csúcsa kiegyensúlyozott. (Mostantól *kiegyensúlyozott* alatt *magasság szerint kiegyensúlyozottat* értünk.) Egy bináris fa egy $(*p)$ csúcsa *kiegyensúlyozott* (balanced node), ha a csúcs $p \to b$ egyensúlyára $|p \to b| \leq 1$. A $(*p)$ csúcs egyensúlya [the $(*p)$ node's balance] definíció szerint:

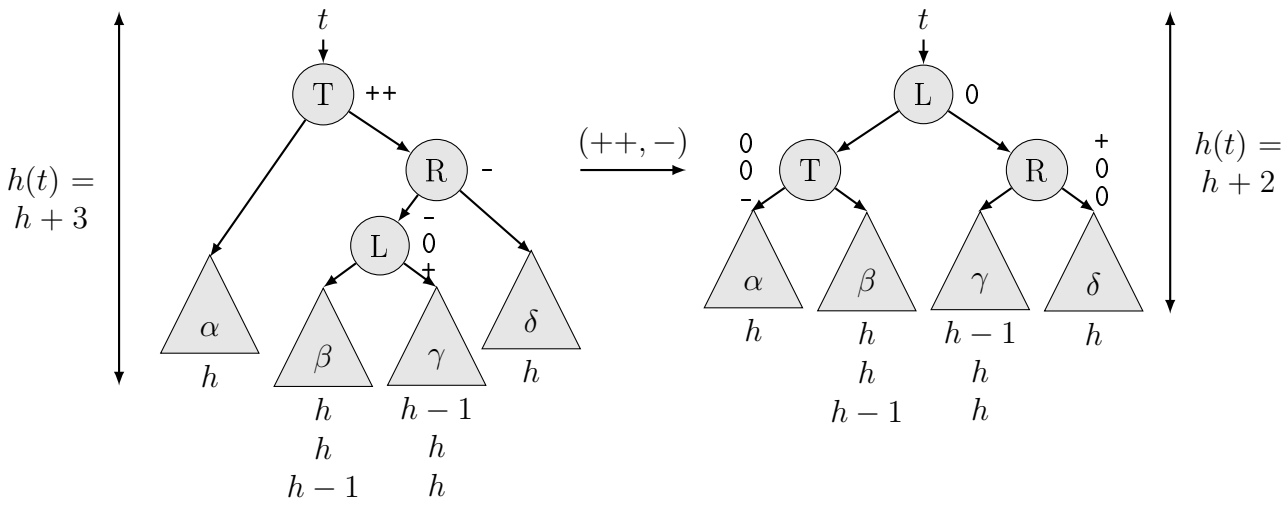
$$p \to b = h(p \to right) - h(p \to left)$$



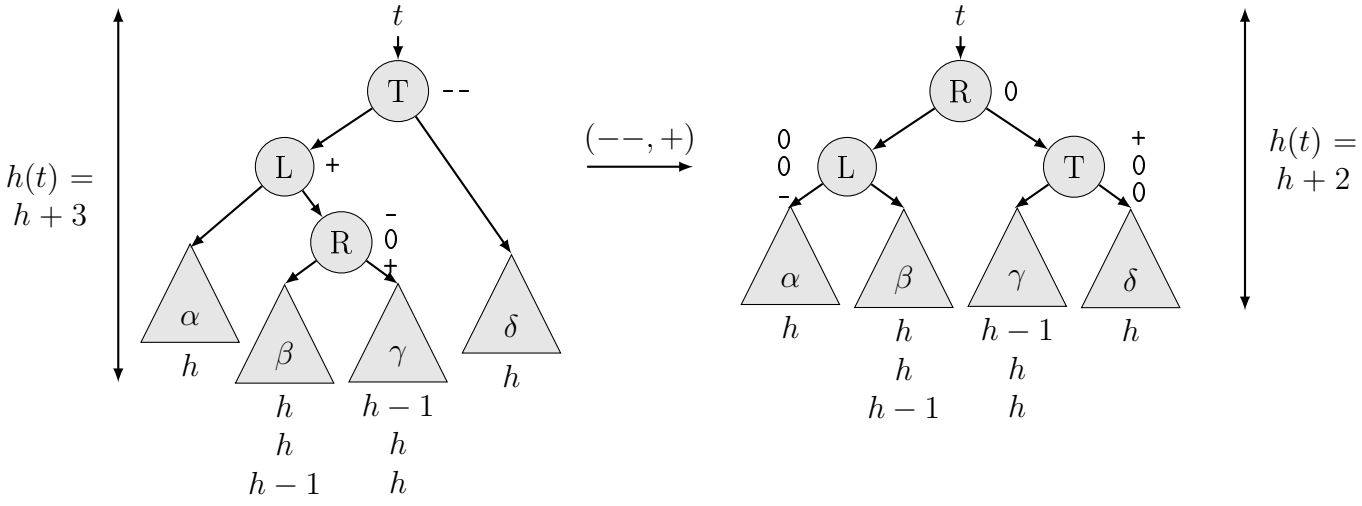
1. ábra. $(++,+)$ forgatás. A kiegyensúlyozatlan (T) csúcsnak és magasabb részfája (R) gyökércsúcsának egyensúlya nem ellentétes előjelű. Így a magasabb részfájához tartozó gyermeke, (R), a kiegyensúlyozó forgatás (*balancing rotation*) után az új gyökércsúcs. A többiek helyét a BST tulajdonság már meghatározza.



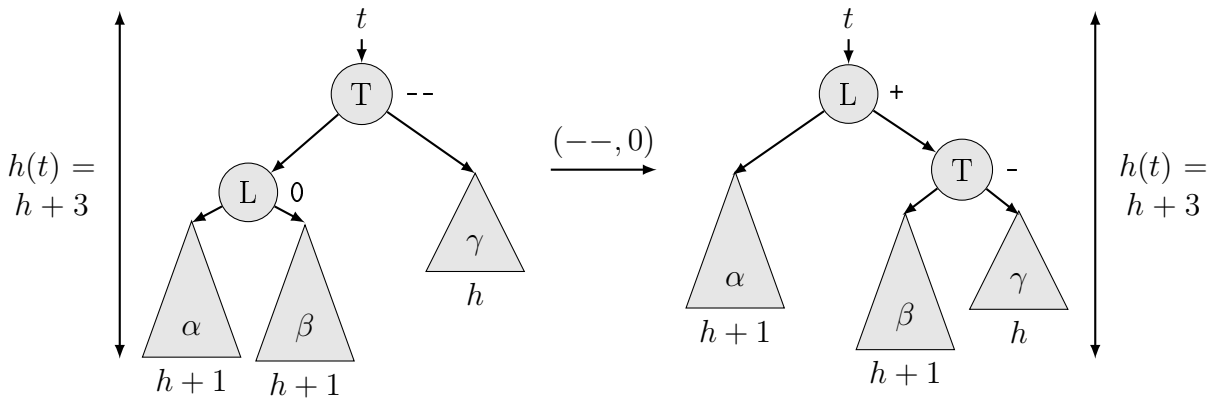
2. ábra. $(--,-)$ forgatás. A kiegyensúlyozatlan (T) csúcsnak és magasabb részfája (L) gyökércsúcsának egyensúlya nem ellentétes előjelű. Így a magasabb részfájához tartozó gyermeke, (L), a kiegyensúlyozó forgatás (*balancing rotation*) után az új gyökércsúcs. A többiek helyét a BST tulajdonság már meghatározza.



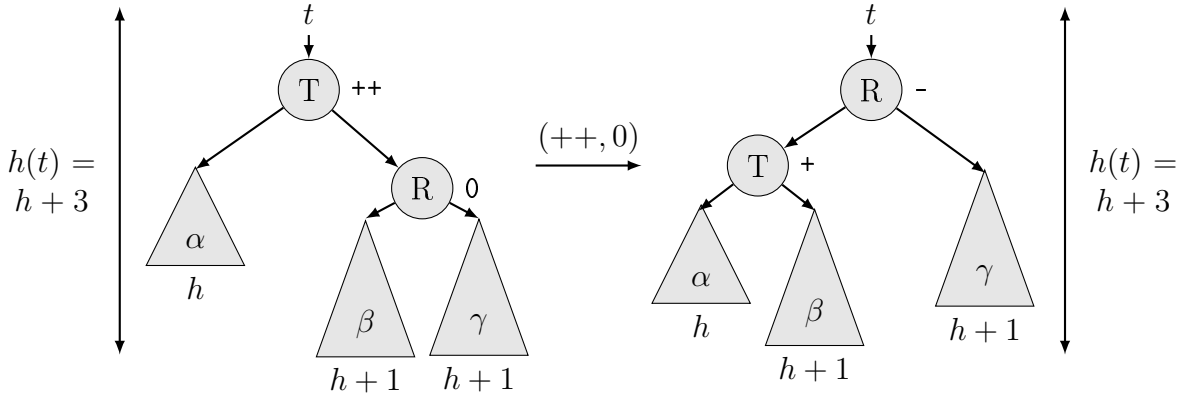
3. ábra. $(++,-)$ forgatás. A kiegyensúlyozatlan (T) csúcsnak és magasabb részfája (R) gyökércsúcsának egyensúlya ellentétes előjelű. Így a magasabb részfájához tartozó unokája, (L), a kiegyensúlyozó forgatás után az új gyökércsúcs. A többiek helyét a BST tulajdonság már meghatározza.



4. ábra. $(--,+)$ forgatás. A kiegyensúlyozatlan (T) csúcsnak és magasabb részfája (L) gyökércsúcsának egyensúlya ellentétes előjelű. Így a magasabb részfájához tartozó unokája, (R), a kiegyensúlyozó forgatás után az új gyökércsúcs. A többiek helyét a BST tulajdonság már meghatározza.



5. ábra. $(--, 0)$ forgatás. A kiegyensúlyozatlan (T) csúcsnak és magasabb részfája (L) gyökércsúcsának egyensúlya nem ellentétes előjelű. Így a magasabb részfájához tartozó gyermeke, (L), a kiegyensúlyozó forgatás (*balancing rotation*) után az új gyökércsúcs. A többiek helyét a BST tulajdonság már meghatározza.



6. ábra. $(++, 0)$ forgatás. A kiegyensúlyozatlan (T) csúcsnak és magasabb részfája (R) gyökércsúcsának egyensúlya nem ellentétes előjelű. Így a magasabb részfájához tartozó gyermeke, (R), a kiegyensúlyozó forgatás (*balancing rotation*) után az új gyökércsúcs. A többiek helyét a BST tulajdonság már meghatározza.

Az AVL fákat láncoltan reprezentáljuk. A csúcsokban a b egyensúly attribútumot expliciten tároljuk. (Egy másik lehetőség a két részfa magasságainak tárolása lenne; egy harmadik, hogy csak az aktuális részfa magasságát tároljuk.) A Node osztályt tehát a következőképpen módosítjuk:

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

Egyelőre részletes bizonyítás nélkül közöljük az alábbi eredményt:

Tétel: Tetszőleges n csúcsú nemüres AVL fa h magasságára:

$$\lfloor \log n \rfloor \leq h \leq 1,45 \log n, \quad \text{azaz} \quad h \in \Theta(\log n)$$

A bizonyítás vázlata: Először a h magasságú, nemüres KBF-ek (kiegyensúlyozott, bináris fák) n méretére adunk alsó és felső becslést. Az $n < 2^{h+1}$ becslésből azonnal adódik $\lfloor \log n \rfloor \leq h$. Másrészt meghatározzuk a h mélységű, legkisebb méretű KBF-ek csúcsainak f_h számát. Erre kapjuk, hogy $f_0 = 1$, $f_1 = 2$, $f_h = 1 + f_{h-1} + f_{h-2} \quad (h \geq 2)$. Ezért az ilyen fákat *Fibonacci fáknak hívjuk.* Mivel tetszőleges h magasságú KBF n méretére $n \geq f_h$, némi matematikai ügyességgel kaphatjuk a $h \leq 1,45 \log n$ egyenlőtlenséget.

Mivel az AVL fák magassága $\Theta(\log n)$, ezért a bináris keresőfák search(t, k), min(t) és max(t) függvényeire t AVL fa esetén automatikusan
$MT(n) \in \Theta(\log n)$, ahol $n = |t|$.

Az insert(t, k), a del(t, k), a remMin$(t, minp)$ és a remMax$(t, maxp)$ eljárások azonban változtatják a fa alakját. Így elromolhat a kiegyensúlyozottság, és már nem garantált a fenti műveletigény. Ennek elkerülésére ezeket az eljárásokat úgy módosítjuk, hogy minden egyes rekurzív eljáráshívás után ellenőrizni fogjuk, hogyan változott a megfelelő részfa magassága, és ez hogyan befolyásolta a felette levő csúcs kiegyensúlyozottságát, szükség esetén helyreállítva azt. Ez minden szinten legfeljebb konstans mennyiségű extra műveletet fog jelenteni, és így a kiegészített eljárások futási ideje megtartja az $MT(n) \in \Theta(\log n)$ (ahol $n = |t|$) nagyságrendet.

A példákhoz a *(bal_részfa gyökér jobb_részfa)* jelölést használjuk, ahol az üres (rész)fákat üres zárójelpár jelöli. A könnyebb olvashatóság kedvéért [],

{} és esetleg ⟨⟩ zárójeleket is alkalmazunk, a levécsúcsoknál pedig a két üres részfát nem jelöljük.

Például a {[2]4[(6)8(10)]} bináris keresőfa gyökere a 4; bal részfája a [2], ami egyetlen levélcsúcsból (és annak üres bal és jobb részfáiból) áll; jobb részfája a [(6)8(10)], aminek gyökere a 8, bal részfája a (6), jobb részfája a (10). *Az így ábrázolt fák inorder bejárása a zárójelek elhagyásával adódik.* A belső csúcsok egyensúlyait *kb* formában jelöljük, ahol *k* a csúcsot azonosító kulcs, *b* pedig a csúcs egyensúlya, a 0:○, 1:+, 2:++, -1:−, -2:−− megfeleltetéssel. Mivel a levélcsúcsok egyensúlya mindig nulla, a leveleknél nem jelöljük az egyensúlyt. A fenti AVL fa például a megfelelő egyensúlyokkal a következő: {[2]4+[(6)8○(10)]}

Az AVL fák műveletei során a kiegyensúlyozatlan részfák kiegyensúlyozásához *forgatásokat* fogunk használni. Az alábbi forgatási sémákban görög kisbetűk jelölik a részfákat (amelyek minden esetben AVL fák lesznek), latin nagybetűk pedig a csúcsok kulcsait (1. és 2. ábrák, az egyensúlyok nélkül):

Balra forgatás (Left rotation): $[\alpha\ \mathrm{T}\ (\beta\ \mathrm{R}\ \gamma)] \to [(\alpha\ \mathrm{T}\ \beta)\ \mathrm{R}\ \gamma]$
Jobbra forgatás (Right rotation): $[(\alpha\ \mathrm{L}\ \beta)\ \mathrm{T}\ \gamma] \to [\alpha\ \mathrm{L}\ (\beta\ \mathrm{T}\ \gamma)]$

Vegyük észre, hogy a fa inorder bejárását egyik forgatás sem változtatja, így a bináris keresőfa tulajdonságot is megtartják. Mint látni fogjuk, a kiegyensúlyozatlan részfák kiegyensúlyozása minden esetben egy vagy két forgatásból áll. *Ezért a bináris keresőfa tulajdonságot a kiegyensúlyozások is megtartják.*

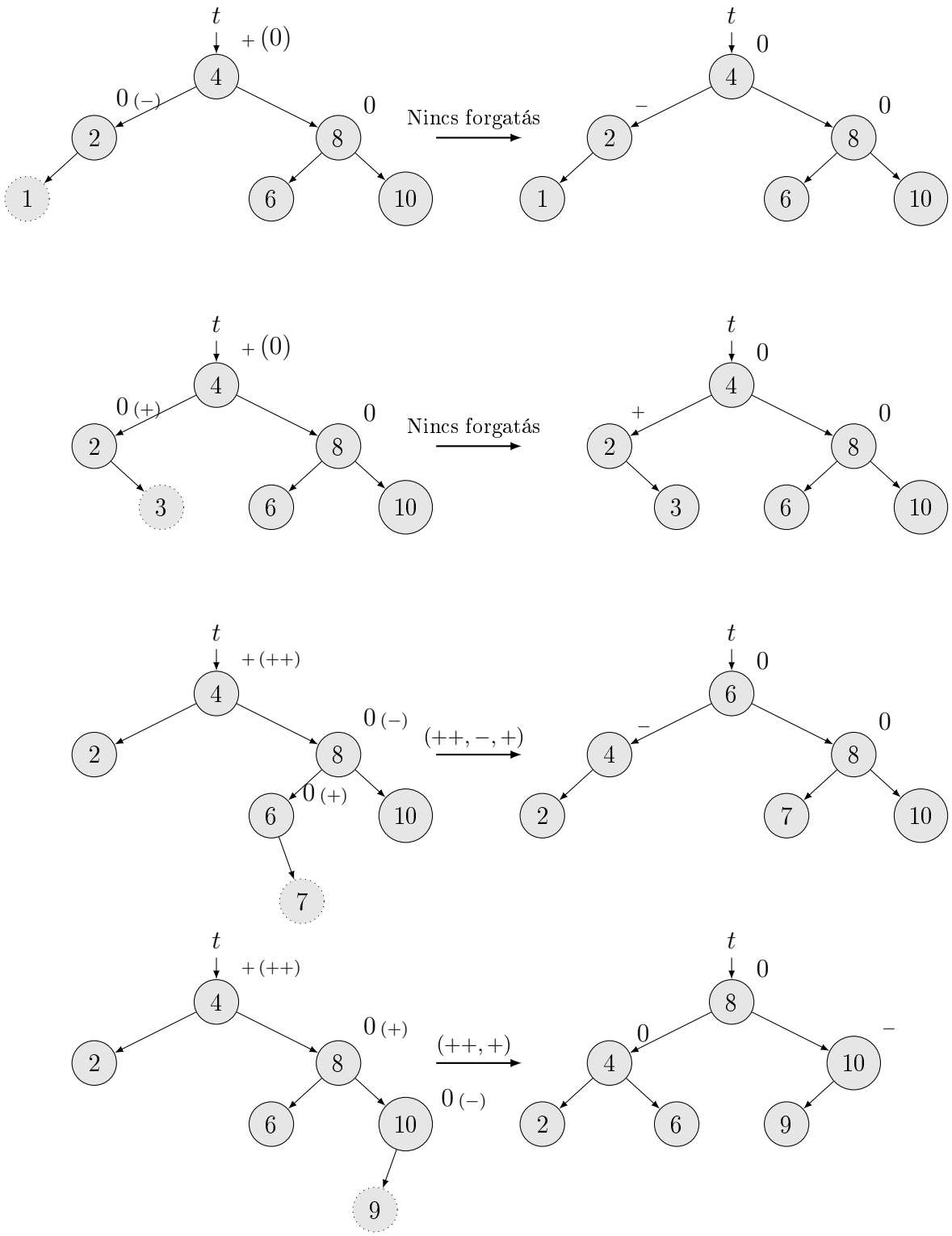
1.1. AVL fák: beszúrás

Nézzük most először a bináris keresőfák insert(t, k) eljárását!

A {[2]4+[(6)8○(10)]} AVL fából
az 1 beszúrásával az {[(1)2−()]4○[(6)8○(10)]} AVL fa adódik,
a 3 beszúrásával pedig az {[()2+(3)]4○[(6)8○(10)]} AVL fa. (Ld. a 8. ábra első két sorát!)

A fenti {[2]4+[(6)8○(10)]} AVL fából
a 7 beszúrásával azonban a {[2]4++[(⟨⟩6+⟨7⟩)8−(10)]} fát kapjuk,
a 9 beszúrásával pedig a {[2]4++[(6)8+(⟨9⟩10−⟨⟩)]} fát.
Mindkettő bináris keresőfa, de már nem AVL fa, mivel a 4++ csúcs nem kiegyensúlyozott. (Ld. a 8. ábra utolsó két sorát!)

A legutolsó esetben a bináris keresőfa könnyen kiegyensúlyozható, ha meggondoljuk, hogy az a következő sémára illeszkedik, amiben görög kisbetűk



7. ábra. Az AVL fa beszúrás néhány alapesete. A bal oldali ábrákon az új csúcsot halványabban jelöltük. A felette levő csúcsok átsúlyozásának eredményét zárójelbe tettük. A jobb oldalon látható a megfelelő kiegyensúlyozó forgatások (3. sor → 3. ábra 3. alesete; 4. sor → 1. ábra) végeredménye.

jelölik a kiegyensúlyozott részfákat, latin nagybetűk pedig a csúcsok kulcsait:
[α T++ (β R+ γ)], ahol T=4, α=[2], R=8, β=(6) és γ=($\langle$9$\rangle$10−$\langle\rangle$).

A kiegyensúlyozáshoz szükséges transzformáció a fa *balra forgatása* (1. ábra):

$$[\ \alpha\ \mathrm{T}{+}{+}\ (\beta\ \mathrm{R}{+}\ \gamma)\ ] \rightarrow [\ (\alpha\ \mathrm{T}{\circ}\ \beta)\ \mathrm{R}{\circ}\ \gamma\ ]$$

A fenti példán ez a következőt jelenti (8. ábra utolsó sor):
{[2]4++[(6)8+($\langle$9$\rangle$10−$\langle\rangle$)]} → { [(2)4∘(6)] 8∘ [(9)10−()] }

A transzformáció helyességének belátásához bevezetjük a $h = h(\alpha)$ jelölést. Ebből a kiinduló fára a T++ és R+ egyensúlyok miatt
$h((\beta\ \mathrm{R}\ \gamma)) = h{+}2$, $h(\gamma) = h{+}1$ és $h(\beta) = h$ adódik, innét pedig az eredmény fára $h(\alpha) = h = h(\beta)$ miatt T∘; továbbá $h((\alpha\ \mathrm{T}{\circ}\ \beta)) = h + 1 = h(\gamma)$ miatt R∘ adódik.

Az eredeti {[2]4+[(6)8∘(10)]} AVL fába a 7 beszúrásával kapott
{ [2]4++[($\langle\rangle$6+$\langle$7$\rangle$)8−(10)] } kiegyensúlyozatlan bináris keresőfa kiegyensúlyozásával pedig {[(2)4−()] 6∘ [(7)8∘(10)]} adódik (8. ábra, utolsó előtti sor), amire az

$$\{\ \alpha\ \mathrm{T}{+}{+}\ [(\beta\ \mathrm{L}{-}{\circ}{+}\ \gamma)\ \mathrm{R}{-}\ \delta]\ \} \rightarrow \{\ [\alpha\ \mathrm{T}{\circ}{\circ}{-}\ \beta]\ \mathrm{L}{\circ}\ [\gamma\ \mathrm{R}{+}{\circ}{\circ}\ \delta]\ \}$$

séma (3. ábra) szolgál, ahol α, β, γ és δ AVL fák, és a három jelből álló egyensúlyok sorban három esetet jelentenek úgy, hogy a bal oldalon az i-edik alternatívának a jobb oldalon is az i-edik eset felel meg ($i \in \{1; 2; 3\}$).

A fenti séma a példában T=4, α = [2], R=8, δ = (10), L=6, + egyensúllyal (harmadik eset), $\beta = \oslash$ és $\gamma = \langle 7\rangle$ helyettesítéssel alkalmazható.

A transzformációt *kettős forgatásnak* is tekinthetjük: Az
{ α T [(β L γ) R δ] } fára először az R csúcsnál alkalmaztunk egy jobbra forgatást, aminek eredményeképpen az { α T [β L (γ R δ)] } bináris keresőfát kaptuk, majd az eredmény fát balra forgattuk, ami az { [α T β] L [γ R δ] } AVL fát eredményezte.

A kettős forgatás helyességének ellenőrzéséhez kiszámítjuk az eredmény fa egyensúlyait. Most is bevezetjük a $h = h(\alpha)$ jelölést. Mivel a kettős forgatás előtti fában T++, azért $h([(\beta\ \mathrm{L}\ \gamma)\ \mathrm{R}\ \delta]) = h + 2$. Innét R− miatt
$h((\beta\ \mathrm{L}\ \gamma)) = h{+}1$ és $h(\delta) = h$. Most L lehetséges egyensúlyai szerint három eset van.
(1) L− esetén $h(\beta) = h$ és $h(\gamma) = h - 1$ ⇒ az eredmény fában T∘ és R+.
(2) L∘ esetén $h(\beta) = h$ és $h(\gamma) = h$ ⇒ az eredmény fában T∘ és R∘.
(3) L+ esetén $h(\beta) = h - 1$ és $h(\gamma) = h$ ⇒ az eredmény fában T− és R∘.

Mindhárom esetben $h([\alpha\ \mathrm{T}\ \beta]) = h + 1 = h([\gamma\ \mathrm{R}\ \delta])$, így L$\circ$.
Ezzel beláttuk a kettős forgatási séma helyességét.

Vegyük észre, hogy a fentiek alapján, az L csúcsnak a kettős forgatás előtti egyensúlyát s-sel, a T és a R csúcsoknak pedig a kettős forgatás utáni egyensúlyát rendre s_t-vel, illetve s_r-vel jelölve a következő összefüggéseket írhatjuk fel:

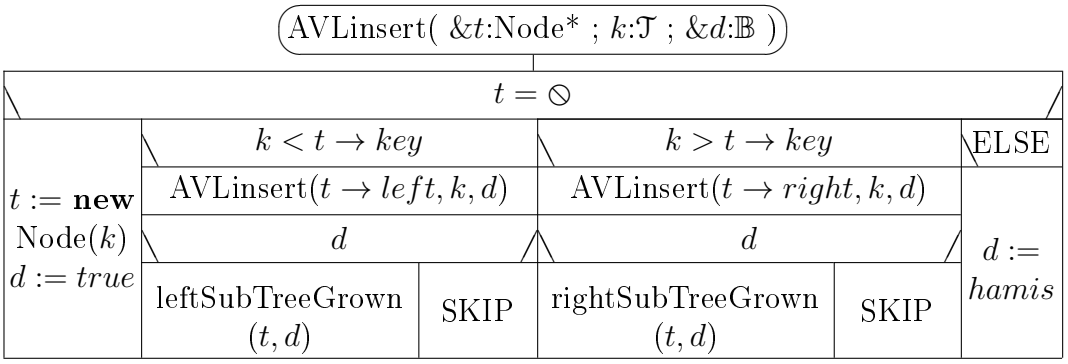
$$s_t = -\lfloor (s+1)/2 \rfloor \quad \text{és} \quad s_r = \lfloor (1-s)/2 \rfloor$$

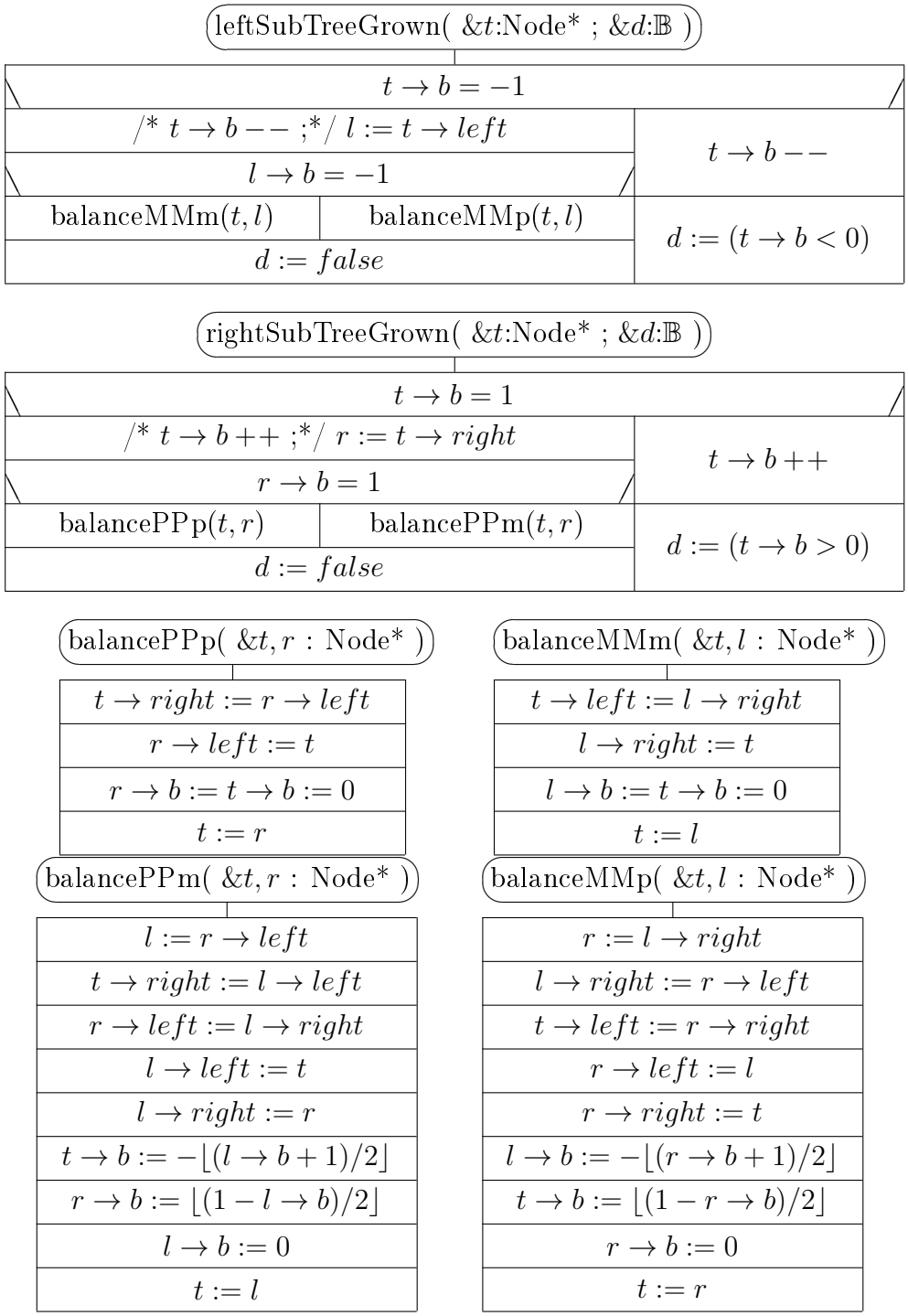
Az eddigi két kiegyensúlyozási séma mellett természetes módon adódnak ezek tükörképei, a forgatások előtt a T$--$ csúccsal a gyökérben. Ezek tehát a következők (2. és 4. ábra):

$$[\ (\alpha\ \mathrm{L}{-}\ \beta)\ \mathrm{T}{--}\ \gamma\ ] \to [\ \alpha\ \mathrm{L}{\circ}\ (\beta\ \mathrm{T}{\circ}\ \gamma)\ ]$$
$$\{\ [\alpha\ \mathrm{L}{+}\ (\beta\ \mathrm{R}{-\circ+}\ \gamma)]\ \mathrm{T}{--}\ \delta\ \} \to [\ (\alpha\ \mathrm{L}{\circ\circ-}\ \beta)\ \mathrm{R}{\circ}\ (\gamma\ \mathrm{T}{+\circ\circ}\ \delta)\ ]$$
$$s_l = -\lfloor (s+1)/2 \rfloor \quad \text{és} \quad s_t = \lfloor (1-s)/2 \rfloor$$

ahol s az R csúcs kettős forgatás előtti egyensúlya; s_l, illetve s_t pedig rendre az L és T csúcsoknak a kettős forgatás utáni egyensúlya

Eddig nem beszéltünk arról, hogy az AVL fában az új csúcs beszúrása után mely csúcsoknak és milyen sorrendben számoljuk újra az egyensúlyát, sem hogy mikor ütemezzük be a kiegyensúlyozást. Csak a beszúrás nyomvonalán visszafelé haladva kell újraszámolnunk az egyensúlyokat, és csak itt kell kiegyensúlyoznunk, ha kiegyensúlyozatlan csúcsot találunk. Így a futási idő a fa magasságával arányos marad, ami AVL fákra $O(\log n)$. Most tehát részletezzük a beszúró és kiegyensúlyozó algoritmus működését. Ennek során a fa magassága vagy eggyel növekszik ($d = true$), vagy ugyanannyi marad ($d = false$).





Az AVL fába való beszúrást röviden összefoglalva:

1. Megkeressük a kulcs helyét a fában.
2. Ha a kulcs benne van a fában, STOP.

3. Ha a kulcs helyén egy üres részfa található, beszúrunk az üres fa helyére egy új, a kulcsot tartalmazó levélcsúcsot. Ez a részfa eggyel magasabb lett.
4. Ha a gyökércsúcsnál vagyunk, STOP. Különben egyet fölfelé lépünk a keresőfában. Mivel az a részfa, amiből fölfele léptünk, eggyel magasabb lett, az aktuális csúcs egyensúlyát megfelelőképp módosítjuk. (Ha a jobb részfa lett magasabb, hozzáadunk az egyensúlyhoz egyet, ha a bal, levonunk belőle egyet.)
5. Ha az aktuális csúcs egyensúlya 0 lett, akkor az aktuális csúcshoz tartozó részfa alacsonyabb ága hozzánőtt a magasabbikhoz, tehát az aktuális részfa most ugyanolyan magas, mint a beszúrás előtt volt, és így egyetlen más csúcs egyensúlyát sem kell módosítani: STOP.
6. Ha az aktuális csúcs új egyensúlya 1 vagy -1, akkor előtte 0 volt, ezért az aktuális részfa magasabb lett eggyel. Ekkor a 4. ponttól folytatjuk.
7. Ha az aktuális csúcs új egyensúlya 2 vagy -2[1], akkor a hozzá tartozó részfát ki kell egyensúlyozni. A *kiegyensúlyozás után az aktuális részfa visszanyeri a beszúrás előtti magasságát*, ezért már egyetlen más csúcs egyensúlyát sem kell módosítani: STOP.

Az az állítás, hogy ebben az algoritmusban a *kiegyensúlyozás után az aktuális részfa visszanyeri a beszúrás előtti magasságát*, még igazolásra vár. Azt az esetet nézzük meg, amikor a kiegyensúlyozandó részfa gyökere T++. A T−− eset hasonlóan fontolható meg. A T++ esethez tartozó kiegyensúlyozási sémák (1. és 3. ábra):

$$[\ \alpha\ \mathrm{T}{+}{+}\ (\beta\ \mathrm{R}{+}\ \gamma)\ ] \rightarrow [(\alpha\ \mathrm{T}{\circ}\ \beta)\ \mathrm{R}{\circ}\ \gamma]$$
$$\{\ \alpha\ \mathrm{T}{+}{+}\ [(\beta\ \mathrm{L}{-}{\circ}{+}\ \gamma)\ \mathrm{R}{-}\ \delta]\ \} \rightarrow \{\ [\alpha\ \mathrm{T}{\circ}{\circ}{-}\ \beta]\ \mathrm{L}{\circ}\ [\gamma\ \mathrm{R}{+}{\circ}{\circ}\ \delta]\ \}$$

Először belátjuk, hogy ha a T csúcs jobboldali R gyereke + vagy − súlyú, akkor a fenti sémák közül a megfelelő alkalmazható: Mivel a beszúró algoritmus a fában a beszúrás helyétől egyesével fölfele lépked, és az első kiegyensúlyozatlan csúcsnál azonnal kiegyensúlyoz, ez alatt nincs kiegyensúlyozatlan csúcs, azaz az α, β és γ, illetve a δ részfák is kiegyensúlyozottak, ez pedig éppen a fenti forgatások feltétele, amellett, hogy bináris keresőfát akarunk kiegyensúlyozni, amit viszont a kiegyensúlyozás nélküli beszúró algoritmus garantál.

Most még be kell látni, hogy a fenti sémák minden esetet lefednek, azaz R+ vagy R−: egyrészt, R nem lehet a beszúrás által létrehozott új csúcs, mert különben T-nek a beszúrás előtti jobboldali részfája üres lett volna, tehát most nem lehetne T++. Másrészt, ha a fölfele lépkedés során nulla

[1] A 2 és -2 eseteket a struktogramban nem számoltuk ki expliciten, hogy az egyensúly tárolására elég legyen két bit.

egyensúlyú csúcs áll elő, akkor a fölötte levő csúcsok egyensúlya már nem módosul, így kiegyensúlyozatlan csúcs sem állhat elő. Márpedig most T++. Így tehát az új csúcstól T-ig fölfelé vezető úton minden csúcs, azaz R egyensúlya is + vagy −.

Most belátjuk, hogy a kiegyensúlyozások visszaállítják a részfa beszúrás előtti magasságát. A T++ kiegyensúlyozatlan csúcs a beszúrás előtt kiegyensúlyozott volt. Mivel a beszúrás, a beszúrás helyétől kezdve T-ig fölfelé vezető úton mindegyik részfa magasságát pontosan eggyel növelte, a beszúrás előtt T+ volt. Ezért a beszúrás előtt, $h = h(\alpha)$ jelöléssel, a T gyökerű részfa $h+2$ magas volt. A beszúrás után tehát T++ lett, és így a T gyökerű részfa $h+3$ magas lett. A beszúrás utáni, de még a kiegyensúlyozás előtti állapotot tekintve a továbbiakban megkülönböztetjük a T jobboldali R gyerekére a R+ és a R− eseteket.

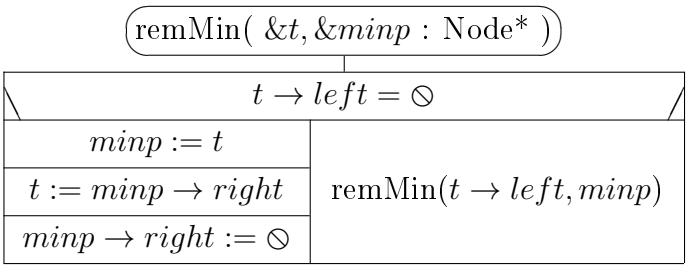
R+ esetén már láttuk, hogy $h(\alpha) = h = h(\beta)$ és $h(\gamma) = h + 1$. Ezért a kiegyensúlyozás után $h([(\alpha\ \mathrm{T}\ \beta)\ \mathrm{R}\ \gamma]) = h + 2$.

R− esetén pedig már láttuk, hogy $h(\alpha) = h = h(\delta)$ és $h([\beta\ \mathrm{L}\ \gamma]) = h+1$. Ezért $h(\beta), h(\gamma) \leq h$. Így a kiegyensúlyozás után $h(\{[\alpha\ \mathrm{T}\ \beta]\ \mathrm{L}\ [\gamma\ \mathrm{R}\ \delta]\}) = h + 2$.

Ezzel beláttuk, hogy a kiegyensúlyozások mindkét esetben visszaállítják a részfa beszúrás előtti magasságát, mégpedig úgy, hogy a beszúrás által eggyel megnövelt magasságot eggyel csökkentik. Szimmetria okokból ez hasonlóan látható be T−− esetén is, figyelembe véve a L− és a L+ eseteket.

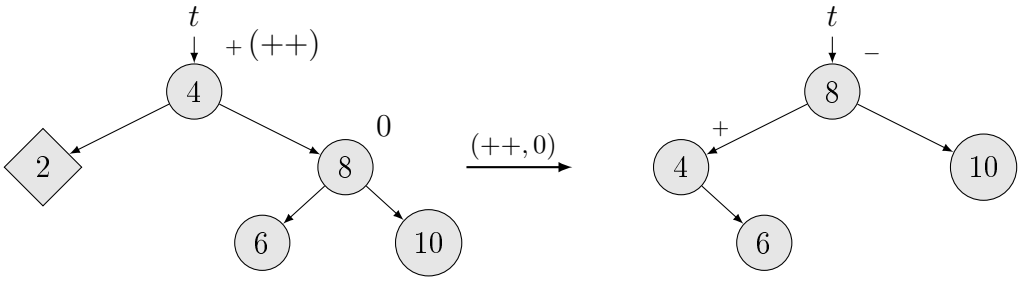
1.2. AVL fák: a remMin($t, minp$) eljárás

Bináris keresőfákra a remMin eljárás:



Ezt például a { [2]4+[(6)8∘(10)] } AVL fára alkalmazva, a $minp \to key = 2$ eredmény mellett, a { [ ] 4++[(6)8∘(10)] } kiegyensúlyozatlan bináris keresőfa adódna.

Látjuk, hogy a kiegyensúlyozatlan 4++ csúcs jobboldali gyereke a 8∘. Ennek az esetnek a kezelésére új kiegyensúlyozási sémát kell alkalmaznunk.



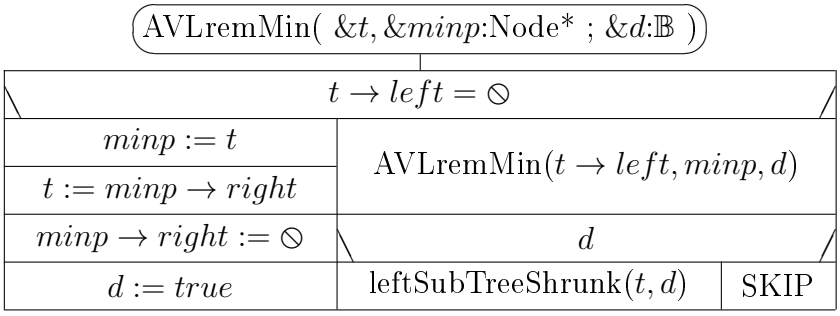
8. ábra. Az AVL fák remMin() műveletének új alapesete. A bal oldali ábrán az eltávolított csúcsot rombusszal jelöltük. A felette levő csúcs átsúlyozásának eredményét zárójelbe tettük. A jobb oldalon látható a megfelelő kiegyensúlyozó forgatás (6. ábra) végeredménye.

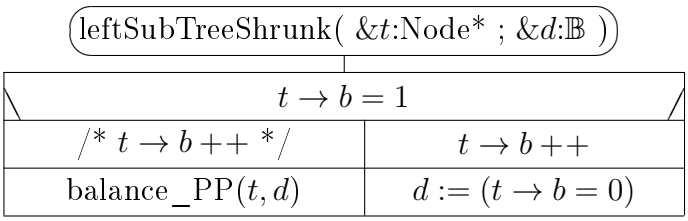
Szerencsére egy balra forgatás, a megfelelő egyensúly beállítások mellett most is megoldja a problémát (6. ábra):

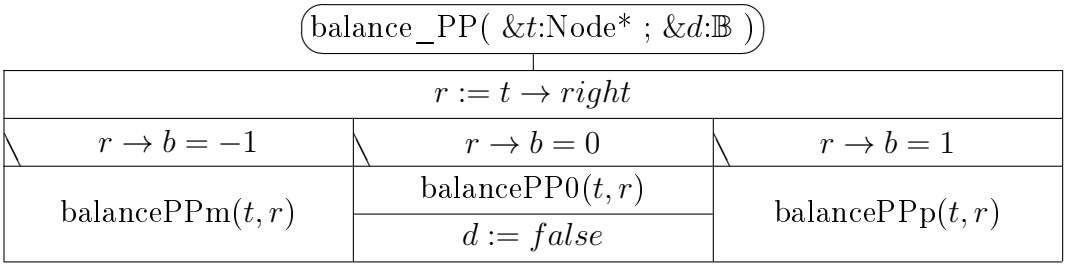
$$\{\ \alpha\ \mathrm{T{+}{+}}[\beta\ \mathrm{R}\circ\ \gamma]\ \} \to \{\ [\alpha\ \mathrm{T{+}}\ \beta]\ \mathrm{R}{-}\ \gamma\ \}.$$

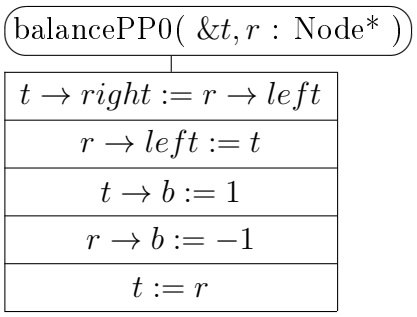
T++ miatt $h = h(\alpha)$ jelöléssel nyilván $h(\beta) = h + 1 = h(\gamma)$, így a forgatás után az egyensúlyokat a fenti séma szerint kell beállítani. A fa magassága a forgatás előtt és után is $h + 3$ lesz, ez a fajta kiegyensúlyozás tehát az eddigiekkel szemben *nem* csökkenti az aktuális részfa magasságát.

Ezután az AVL fákra alkalmazott, a megfelelő kiegyensúlyozásokkal kiegészített AVLremMin eljárás pl. a következő lehet (d most azt jelöli, hogy csökkent-e az aktuális részfa magassága):









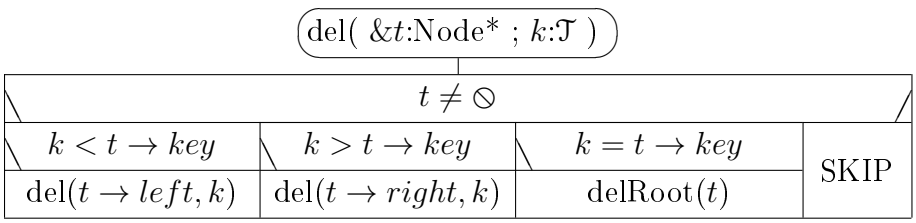
Az algoritmus helyessége hasonlóan gondolható meg, mint a beszúrás esetén. Lényeges különbség azonban, hogy ott minden beszúrást csak egyetlen kiegyensúlyozás követ, míg itt előfordulhat, hogy a minimális csúcs eltávolítása után, minden felette lévő szinten ki kell egyensúlyozni.

1.1. Feladat. *Mutassunk ilyen, legalább négy magasságú fát!*

1.2. Feladat. *Írjuk meg az AVLremMin(t, minp, d) eljárás mintájára az AVLremMax(t, maxp, d) eljárást és segédeljárásait!*

1.3. AVL fák: törlés

Bináris keresőfákra a del(t, k) és a delRoot(t) eljárások:



delRoot(&t:Node*)

$t \to left = \oslash$	$t \to right = \oslash$	$t \to left \neq \oslash \wedge t \to right \neq \oslash$
$p := t$	$p := t$	remMin$(t \to right, p)$
$t := p \to right$	$t := p \to left$	$p \to left := t \to left$
		$p \to right := t \to right$
delete p	**delete** p	**delete** t ; $t := p$

Ezeket fogjuk most az AVLremMin$(t, minp, d)$ eljárás mintájára kiegészíteni a d paraméterrel; majd a rekurzív töröl hívásokból, valamint az AVLremMin eljárásból való visszatérés után megkérdezzük, csökkent-e az aktuális részfa magassága. Ha igen, az AVLremMin$(t, minp, d)$ eljárás mintájára módosítjuk a $t \to b$ értéket, ha kell, kiegyensúlyozunk, és ha kell, d-t beállítjuk.

AVLdel(&t:Node* ; k:$\mathcal{T}$; &d:$\mathbb{B}$)

<table>
<tr><td colspan="5">$t \neq \oslash$</td></tr>
<tr><td colspan="2">$k < t \to key$</td><td colspan="2">$k > t \to key$</td><td>$k = t \to key$</td></tr>
<tr><td colspan="2">AVLdel$(t \to left, k, d)$</td><td colspan="2">AVLdel$(t \to right, k, d)$</td><td rowspan="3">AVLdelRoot (t, d)</td></tr>
<tr><td colspan="2">d</td><td colspan="2">d</td></tr>
<tr><td>leftSubTreeShrunk (t, d)</td><td>SKIP</td><td>rightSubTreeShrunk (t, d)</td><td>SKIP</td></tr>
</table>

$d := hamis$

AVLdelRoot(&t:Node* ; &d:$\mathbb{B}$)

<table>
<tr><td>$t \to left = \oslash$</td><td>$t \to right = \oslash$</td><td colspan="2">$t \to left \neq \oslash \wedge t \to right \neq \oslash$</td></tr>
<tr><td>$p := t$</td><td>$p := t$</td><td colspan="2" rowspan="2">rightSubTreeMinToRoot(t, d)</td></tr>
<tr><td>$t := p \to right$</td><td>$t := p \to left$</td></tr>
<tr><td>**delete** p</td><td>**delete** p</td><td colspan="2">d</td></tr>
<tr><td>$d := true$</td><td>$d := true$</td><td>rightSubTreeShrunk(t, d)</td><td>SKIP</td></tr>
</table>

rightSubTreeMinToRoot(&t:Node* ; &d:$\mathbb{B}$)

AVLremMin$(t \to right, p, d)$
$p \to left := t \to left$; $p \to right := t \to right$; $p \to b := t \to b$
delete t ; $t := p$

Itt is lehetséges, hogy több szinten, a legrosszabb esetben akár minden szinten is ki kell egyensúlyozni. Mivel azonban egyetlen kiegyensúlyozás sem tartalmaz se rekurziót, se ciklust, és ezért konstans számú eljáráshívásból áll, ez sem itt, sem az AVLremMin eljárásnál nem befolyásolja a futási időnek az AVLinsert eljárásra is érvényes $MT(n) \in \Theta(\log n)$ (ahol $n = |t|$) nagyságrendjét.

1.3. Feladat. *A leftSubTreeShrunk(t, d) eljárás mintájára dolgozzuk ki a rightSubTreeShrunk(t, d) eljárást, ennek segédeljárásait, és az ehhez szükséges kiegyensúlyozási sémát (megoldás: 5. ábra)! Mutassuk be az AVLdel(t, k) eljárás működését néhány példán! Mutassunk olyan, legalább négy magasságú fát és kulcsot, amelyre az AVLdel(t, k) eljárás minden, a fizikailag törölt csúcs feletti szinten kiegyensúlyozást végez!*

1.4. Feladat. *Írjuk meg az AVLdel(t, k) eljárás egy olyan változatát, amely abban az esetben, ha a k kulcsot olyan belső csúcs tartalmazza, aminek két gyereke van, ezt a törlendő csúcsot a bal részfája legnagyobb kulcsú csúcsával helyettesíti!*

1.4. Az AVL fák magassága*

Tétel: Tetszőleges n csúcsú nemüres AVL fa h magasságára:

$$\lfloor \log n \rfloor \leq h \leq 1,45 \log n$$

Bizonyítás:

Az $\lfloor \log n \rfloor \leq h$ egyenlőtlenség bizonyításához elég azt belátni, hogy ez tetszőleges nemüres bináris fára igaz. Egy tetszőleges bináris fa nulladik (gyökér) szintjén legfeljebb $2^0 = 1$ csúcs van, az első szintjén maximum $2^1 = 2$ csúcs, a második szintjén nem több, mint $2^2 = 4$ csúcs. Általában, ha az i-edik szinten 2^i csúcs van, akkor az $i + 1$-edik szinten legfeljebb 2^{i+1} csúcs, hiszen minden csúcsnak maximum két gyereke van. Innét egy h mélységű bináris fa csúcsainak n számára $n \leq 2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1 < 2^{h+1}$. Innét
$\lfloor \log n \rfloor \leq \log n < \log 2^{h+1} = h + 1$, amiből $\lfloor \log n \rfloor \leq h$.

A $h \leq 1,45 \log n$ egyenlőtlenség bizonyításához elég azt belátni, hogy ez tetszőleges nemüres, kiegyensúlyozott bináris fára (KBF-re) igaz. Ehhez először meghatározzuk egy $h \geq 0$ magasságú (azaz nemüres) KBF csúcsainak minimális f_h számát. Nyilván $f_0 = 1$ és $f_1 = 2$, hiszen egy nulla magasságú

KBF csak a gyökércsúcsból áll, az egy magasságú KBF-ek pedig ((B)G(J)), ((B)G), vagy (G(J)) alakúak.

Az is világos, hogy az $< f_0, f_1, f_2, \cdots >$ sorozat szigorúan monoton növekvő. (Ennek igazolásához vegyünk egy $i+1$ magas, f_{i+1} csúcsú t KBF-et! Ekkor a t bal és jobb részfái közül az egyik magassága i. Legyen ez az f részfa! Jelölje most $s(b)$ egy tetszőleges b bináris fa csúcsainak számát! Akkor $f_{i+1} = s(t) > s(f) \geq f_i$.)

Ezután $h \geq 2$ esetén $f_h = 1 + f_{h-1} + f_{h-2}$. (Ha ugyanis t egy h magasságú, minimális, azaz f_h méretű KBF, akkor ennek bal és jobb részfáiban is, a részfák magasságához mérten a lehető legkevesebb csúcs van. Az egyik részfája kötelezően $h-1$ magas, ebben tehát f_{h-1} csúcs van. Mivel t KBF, a másik részfája $h-1$ vagy $h-2$ magas. A másik részfában tehát f_{h-1} vagy f_{h-2} csúcs van, és $f_{h-2} < f_{h-1}$, tehát a másik részfában f_{h-2} csúcs van, és így $f_h = 1 + f_{h-1} + f_{h-2}$.)

A képlet emlékeztet a Fibonacci sorozatra:
$F_0 = 0$, $F_1 = 1$, $F_h = F_{h-1} + F_{h-2}$, ha $h \geq 2$.

Megjegyzés: A h magasságú, és f_h méretű KBF-eket ezért *Fibonacci fáknak* hívjuk. Az előbbiek szerint egy $h \geq 2$ magasságú Fibonacci fa mindig $(\varphi_{h-1}\ \mathrm{G}\ \varphi_{h-2})$ vagy $(\varphi_{h-2}\ \mathrm{G}\ \varphi_{h-1})$ alakú, ahol φ_{h-1} és φ_{h-2} $h-1$, illetve $h-2$ magasságú Fibonacci fák.

1.5. Feladat. *Rajzoljunk $h \in 0..4$ magasságú Fibonacci fákat!*

Megvizsgáljuk, van-e valami összefüggés az
$\langle f_h : h \in \mathbb{N} \rangle$ és az $\langle F_h : h \in \mathbb{N} \rangle$ sorozatok között.

h	0	1	2	3	4	5	6	7	8	9
F_h	0	1	1	2	3	5	8	13	21	34
f_h	1	2	4	7	12	20	33			

A fenti táblázat alapján az első néhány értékre $f_h = F_{h+3} - 1$. Teljes indukcióval könnyen látható, hogy ez tetszőleges $h \geq 0$ egész számra igaz: Feltéve, hogy $0 \leq h \leq k \geq 1$ esetén igaz, $h = k+1$-re:
$f_h = f_{k+1} = 1 + f_k + f_{k-1} = 1 + F_{k+3} - 1 + F_{k+2} - 1 = F_{k+4} - 1 = F_{h+3} - 1$.

Tudjuk, hogy

$$F_h = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^h - \left(\frac{1-\sqrt{5}}{2} \right)^h \right]$$

Az F_h-ra vonatkozó explicit képlet segítségével összefüggést adunk tetszőleges KBF n mérete és h magassága között.

$$n \geq f_h = F_{h+3} - 1 = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{h+3} - \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right] - 1 \geq$$

$$\geq \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{h+3} - \left| \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right| \right] - 1$$

Mivel $2 < \sqrt{5} < 3$, azért $|1-\sqrt{5}|/2 < 1$ és $\left| \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right| < 1$. Eszerint

$$n > \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{h+3} - 1 \right] - 1 = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{3} \left(\frac{1+\sqrt{5}}{2} \right)^{h} - \frac{1+\sqrt{5}}{\sqrt{5}}$$

Mivel

$$\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{3} = \frac{1+3\sqrt{5}+3(\sqrt{5})^2+(\sqrt{5})^3}{8\sqrt{5}} = \frac{16+8\sqrt{5}}{8\sqrt{5}} = \frac{2+\sqrt{5}}{\sqrt{5}}$$

Ezt behelyettesítve az előbbi, n -re vonatkozó egyenlőtlenségbe:

$$n > \frac{2+\sqrt{5}}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{h} - \frac{1+\sqrt{5}}{\sqrt{5}}$$

Az első együtthatót tagokra bontva, a disztributív szabállyal:

$$n > \left(\frac{1+\sqrt{5}}{2} \right)^{h} + \frac{2}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{h} - \frac{1+\sqrt{5}}{\sqrt{5}}$$

Most

$$\frac{2}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{h} - \frac{1+\sqrt{5}}{\sqrt{5}} \geq 0 \iff \left(\frac{1+\sqrt{5}}{2} \right)^{h} \geq \frac{1+\sqrt{5}}{2} \iff h \geq 1$$

Eszerint $h \geq 1$ esetén

$$n > \left(\frac{1+\sqrt{5}}{2} \right)^{h}$$

$h = 0$ -ra pedig $n = 1$, és így $n = \left(\frac{1+\sqrt{5}}{2}\right)^h$

A fentiekből tetszőleges, nemüres KBF n méretére és h magasságára

$$n \geq \left(\frac{1+\sqrt{5}}{2}\right)^h$$

Innét, ha vesszük mindkét oldal kettes alapú logaritmusát, majd $\log \frac{1+\sqrt{5}}{2}$ -tel osztunk:

$$h \leq \frac{1}{\log \frac{1+\sqrt{5}}{2}} \log n$$

Mivel $1,44 < 1,44042 < \frac{1}{\log \frac{1+\sqrt{5}}{2}} < 1,4404201 < 1,45$, azért tetszőleges, nemüres KBF n méretére és h magasságára

$$h \leq 1,45 \log n$$

2. Általános fák (General trees)

Az általános fák esetében, a bináris fákkal összehasonlítva, egy csúcsnak tetszőlegesen sok gyereke lehet. A fák most is irányítottak, és az utak a gyökércsúcstól a levelek felé vezetnek. Itt azonban, tetszőleges csúcshoz tartozó részfák száma pontosan egyenlő a gyerekek számával, azaz nem tartoznak hozzá üres részfák. Ha a gyerekek sorrendje lényeges, akkor *rendezett fáról* (*ordered tree*) beszélünk.

Általános fákkal modellezhetjük például a számítógépünkben a mappák hierarchiáját, a programok blokkstruktúráját, a függvénykifejezéseket, a családfákat és bármelyik hierarchikus struktúrát.

Vegyük észre, hogy ugyan minden konkrét általános fában van az egy csúcshoz tartozó gyerekek számára valamilyen r felső korlát, de ettől a fa még nem tekinthető r-árisnak, mert ez a korlát nem abszolút: a fa tetszőleges csúcsa gyerekeinek száma tetszőlegesen növelhető. Másrészt, mivel itt nem értelmezzük az *üres részfa* fogalmát, ez mégsem általánosítása az r-áris fa fogalmának. Értelmezzük azonban a gyökércsúcs (nincs szülője) és a levélcsúcs (nincs gyereke) fogalmát, azaz továbbra is gyökeres fákról (*rooted trees*) beszélünk.[2]

[2]Ellentétben az ún. *szabad fákkal*, amik irányítatlan, körmentes gráfok.

Az általános fák természetes ábrázolási módja a *bináris láncolt* reprezentáció. Itt egy csúcs szerkezete a következő (ahol most *child*1 az első gyerekre, *sibling* pedig a következő testvérre mutat). A $*p$ csúcs akkor levél, ha $p \to child1 = \oslash$; és a $*p$ csúcs akkor utolsó testvér, ha $p \to sibling = \oslash$.

Node
+ *child*1, *sibling* : Node* // *child*1: első gyerek; *sibling*: következő testvér + *key* : T // T ismert típus
+ Node() { *child*1 := *sibling* := $\oslash$ } // egycsúcsú fát képez belőle + Node(x:T) { *child*1 := *sibling* := $\oslash$; *key* := x }

Természetesen itt is lehet *szülő* pointer, ami a gyerekek közös szülőjére mutat vissza.

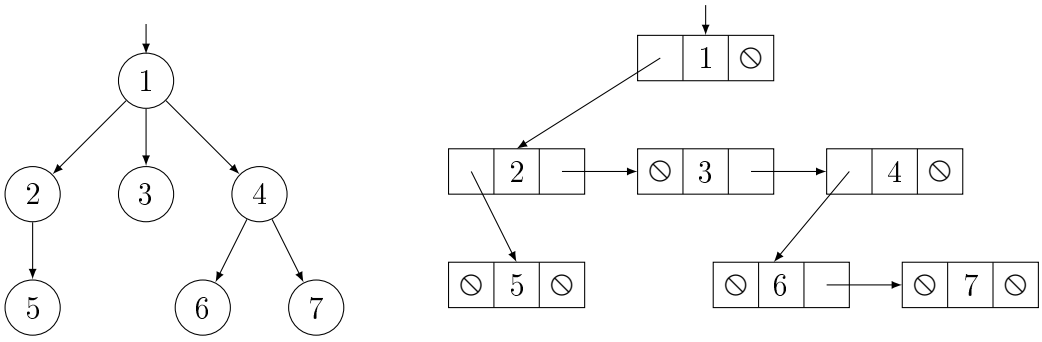
2.1. Feladat. *Próbáljunk megadni az általános fákra másfajta láncolt reprezentációkat is! Hasonlítsuk össze az általunk adott ábrázolásokat és a bináris láncolt reprezentációt, memóriaigény és rugalmasság szempontjából!*

A szöveges (zárójeles) reprezentációban az általános fáknál a gyökeret előre szokás venni. Így egy nemüres fa általános alakja (G $t_1 \dots t_n$), ahol G a gyökércsúcs tartalma, $t_1 \dots t_n$ pedig a részfák.

Így pl. az { 1 [2 (5)] (3) [4 (6) (7)] } általános fában az 1 van a gyökérben, a gyerekei a 2, a 3 és a 4 kulcsú csúcsok, a hozzájuk tartozó részfák pedig sorban a [2 (5)], a (3) és a [4 (6) (7)]. A fa levelei az 5, a 3, a 6 és a 7 kulcsú csúcsok (9. ábra).

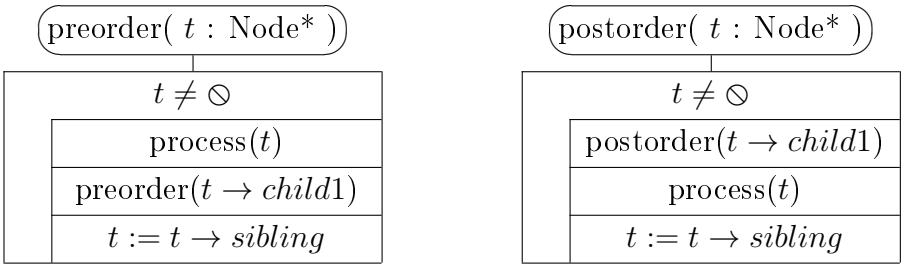
2.2. Feladat. *Írjunk programot, ami kiír egy binárisan láncolt fát a fenti zárójeles alakban! Írjunk olyat is, ami egy szövegfájlból visszaolvassa! Készítsük el a kiíró és a visszaolvasó programokat az általunk kidolgozott alternatív láncolt ábrázolásokra is! (A visszaolvasó programokat általában nehezebb megírni.)*

Ha a fa csúcsai 1-től n-ig (vagy általában, az egész számok egy folytonos intervallumával) indexelhetők, mint a 9. ábrán látható példában, ahol $n = 7$; a fa *fordított reprezentációja* a lehető legtömörebb ábrázolás. Pl. egy $Parent/1 : \mathbb{N}[n]$ tömböt használhatunk, ahol $Parent[i]$ az i indexű csúcs szülőjének indexe. Ha i a gyökércsúcsra hivatkozik, akkor $Parent[i] = 0$. Egy másik fajta fordított reprezentáció szerint a gyökérnél $Parent[i] = i$. Az előbbi tradíció szerint a 9. ábrán látható fa fordított ábrázolása a $\langle 0, 1, 1, 1, 2, 4, 4 \rangle$ tömb, az utóbbi szokás szerint pedig az $\langle 1, 1, 1, 1, 2, 4, 4 \rangle$ tömb. A fordított faábrázolásokkal a gráfalgoritmusoknál fogunk találkozni.



9. ábra. Az { 1 [2 (5)] (3) [4 (6) (7)] } általános fa absztrakt szerkezete (balra) és bináris reprezentációja (jobbra).

Az általános fa preorder bejárása[3] a $child1 \sim left$ és $sibling \sim right$ megfeleltetéssel a bináris reprezentáció preorder bejárását igényli. Az általános fa postorder bejárásához[4] azonban (az előbbi megfeleltetéssel) a bináris reprezentáció inorder bejárása szükséges.[5]



2.3. Feladat. *Lássuk be a fenti bejárások helyességét! (Ötlet: A testvérek listájának mérete szerinti teljes indukció pl. célravezető.) Lássuk be egy ellenpélda segítségével, hogy az általános fa inorder bejárása*[6] *nem szimulálható a bináris reprezentáció egyik nevezetes bejárásával*[7] *sem! Írjuk meg a bináris láncolt reprezentációra az általános fa inorder bejárását és szintfolytonos bejárását is!*

[3]először a gyökér, majd sorban a gyerekeihez tartozó részfák

[4]előbb sorban a gyökér gyerekeihez tartozó részfák, végül a gyökér

[5]A fájlrendszerekben általában preorder bejárás szerint keresünk, míg a függvénykifejezéseket (eltekintve most a lusta kiértékeléstől, aminek a tárgyalása túl messzire vezetne) postorder bejárással értékeljük ki.

[6]A (G $t_1 \dots t_n$) fára $n > 0$ esetén előbb t_1-et járja be, majd feldolgozza G-t, aztán bejárja sorban a $t_2 \dots t_n$ részfákat; $n = 0$ esetén csak feldolgozza G-t.

[7]preorder, inorder, postorder, szintfolytonos

2.4. Feladat. *Írjuk meg a fenti bejárásokat az általunk adott alternatív láncolt reprezentációkra is! Írjunk olyan programokat, amelyek képesek tetszőleges általános fa egy adott reprezentációjából egy adott másik ábrázolású másolatot készíteni!*

3. B+ fák és műveleteik

Ld. B+ fa.pdf a Canvas-ben!