Algoritmusok és adatszerkezetek II.
2. Előadás

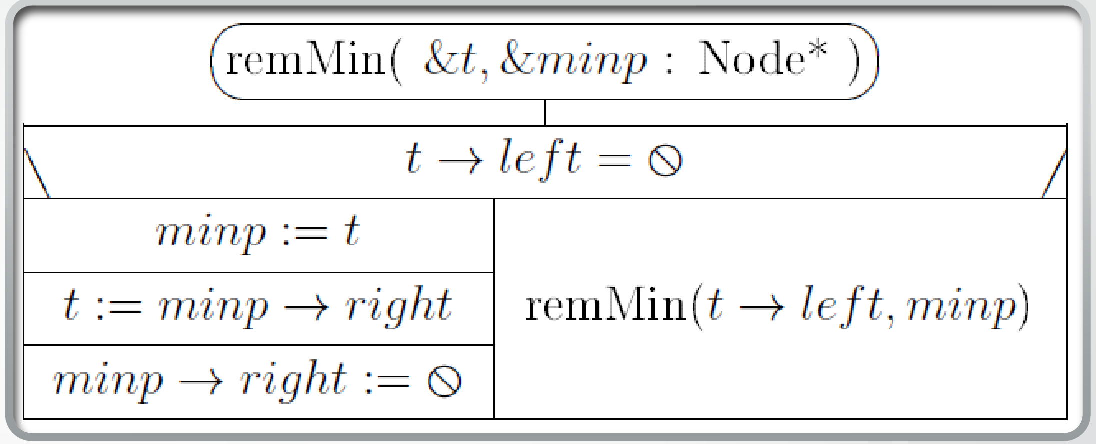
AVL fa II.

AVL fa II.: adott kulcsú csúcs törlése, Fibonacci fák, AVL fa magassága. Általános fák.

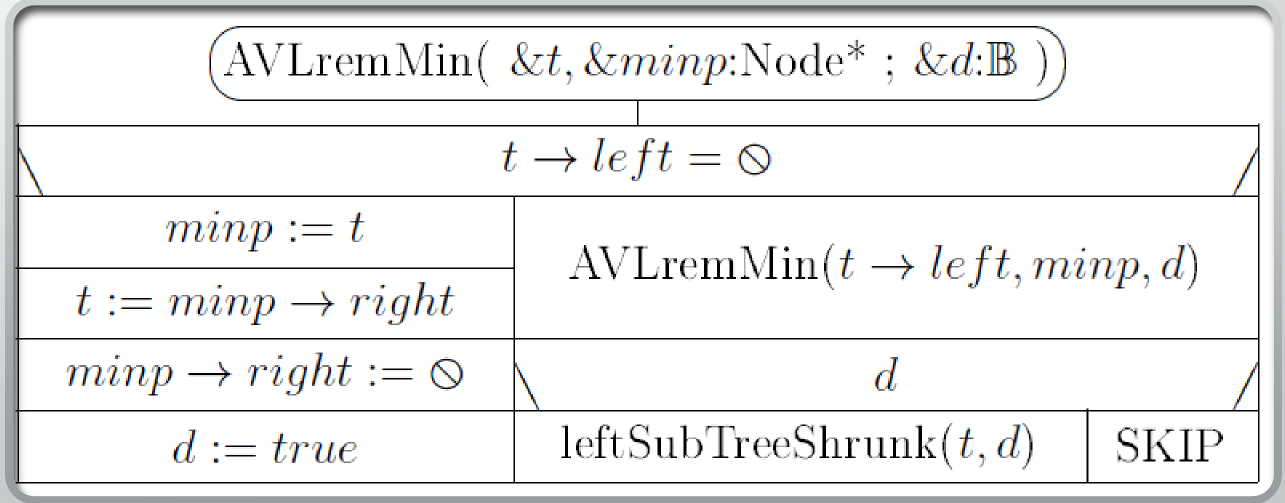
Tartalom

AVL fák: a remMin(*t*; *minp*) eljárás

- Bináris keresőfákra a remMin eljárás:
  - Pl. { [2]4+[(6)8◦ (10)] } AVL fára alkalmazva
  - Eredmény: *minp*-> *key* = 2 és
    { 4++[(6)8◦ (10)] } kiegyensúlyozatlan binkerfa
  - Balra forgatás-> kiegyensúlyozza a fát
  - Nem csökkenti az aktuális részfa magasságát



- AVL fákra alkalmazott, a megfelelő kiegyensúlyozásokkal kiegészített AVLremMin eljárás:
  - *d*: csökkent-e az aktuális részfa magassága



remMin(*t*; *minp*) eljárás segédeljárásai

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

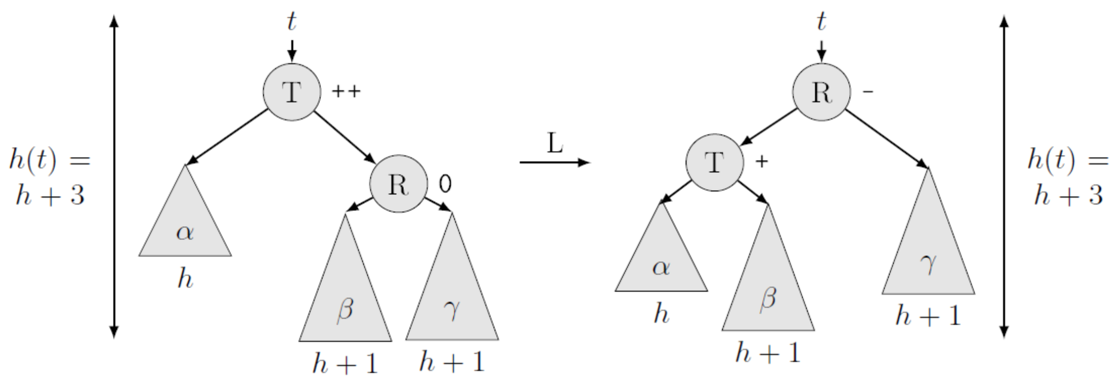
$t \to b = 1$	
balance_PP(t, d)	$t \to b := t \to b + 1$
	$d := (t \to b = 0)$

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

$r := t \to right$

$r \to b = -1$	$r \to b = 0$	$r \to b = 1$
balancePPm(t, r)	balancePP0(t, r)	balancePPp(t, r)
	$d := false$	

remMin(*t*; *minp*) folytatás



balancePP0($\&t, r$: Node*)

$t \to right := r \to left$
$r \to left := t$
$t \to b := 1$
$r \to b := -1$
$t := r$

- Algoritmus helyességének igazolása:
  - ~ beszúrás
- Lényeges különbség:
  - Beszúrásnál: minden beszúrást csak egyetlen kiegyensúlyozás követ
  - Minimum törlésnél: minden felette lévő szinten ki kell egyensúlyozni

AVL fák: törlés

- Bináris keresőfákra a del(*t*; *k*) és a delRoot(*t*) eljárások:
- Változtatások:
  - ~ AVLremMin(*t*; *minp*; *d*)
  - + *d* paraméter

del(&t:Node* ; k:$\mathcal{T}$)

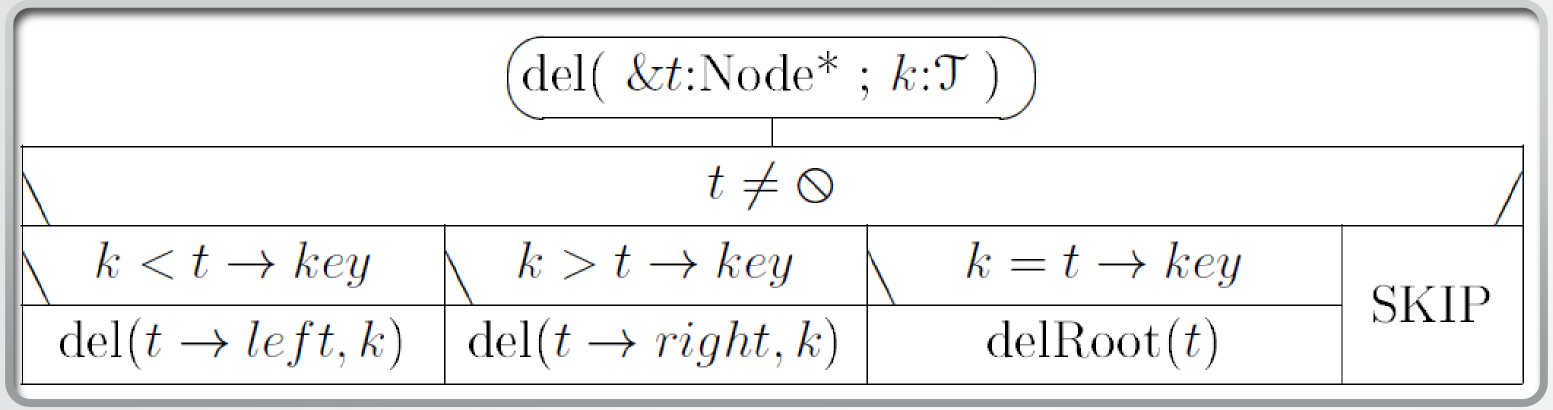
$t \neq \oslash$			
$k < t \to key$	$k > t \to key$	$k = t \to key$	SKIP
$\mathrm{del}(t \to left, k)$	$\mathrm{del}(t \to right, k)$	$\mathrm{delRoot}(t)$	

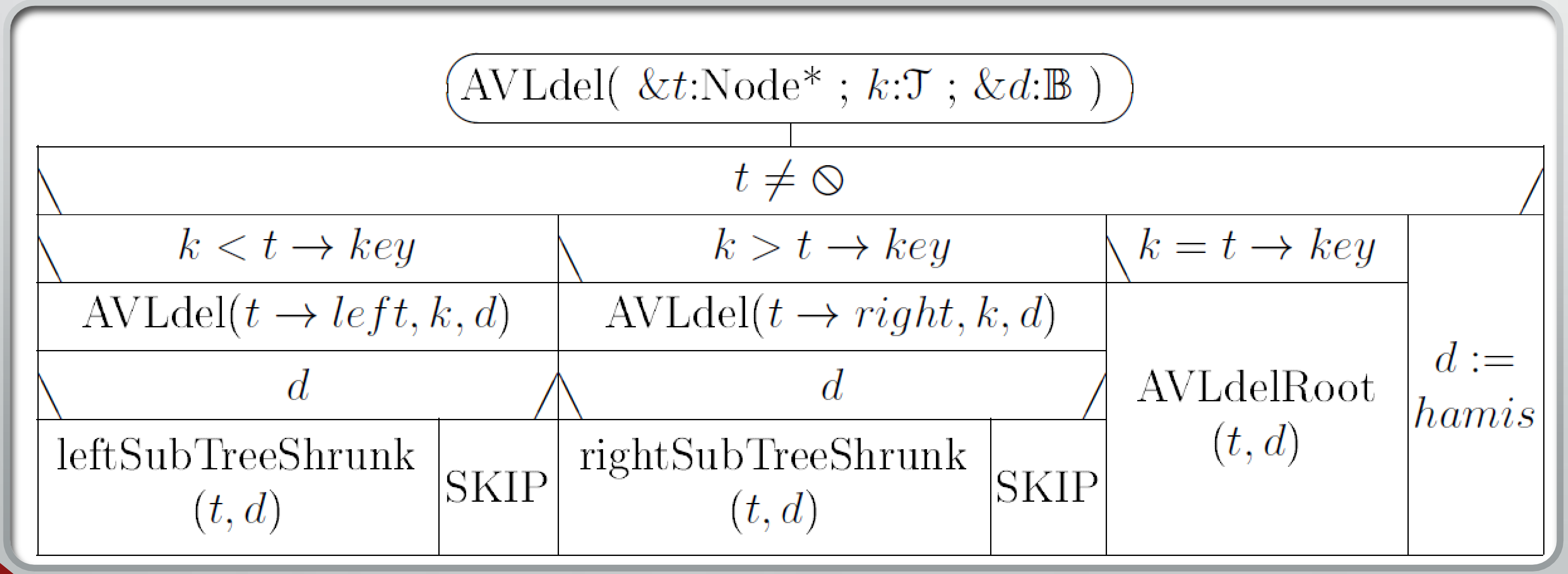
  - ha csökkent az aktuális részfa magassága:
    - módosítjuk a *t* -> *b* értéket
    - ha kell: kiegyensúlyozunk
    - ha kell: *d*-t beállítjuk.

delRoot(&t:Node*)

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

del(t, k) eljárás





delRoot(t) eljárás

delRoot($\&t$:Node*)

$t \to left = \emptyset$	$t \to right = \emptyset$	$t \to left \neq \emptyset \wedge t \to right \neq \emptyset$
$p := t$	$p := t$	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 ; $t := p$

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

$t \to left = \emptyset$	$t \to right = \emptyset$	$t \to left \neq \emptyset \wedge t \to right \neq \emptyset$	
$p := t$	$p := t$	rightSubTreeMinToRoot(t, d)	
$t := p \to right$	$t := p \to left$		
delete p	**delete** p	d	
$d := true$	$d := true$	rightSubTreeShrunk(t, d)	SKIP

delRoot(t) eljárás folytatás

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 ; $t := p$

- Előforltt is lehetséges, hogy több szinten ke, a legrosszabb esetben akár minden szinten is ki kell egyensúlyozni
- Futási idő
  - egyetlen kiegyensúlyozás sem tartalmaz se rekurziót, se ciklust, és ezért konstans számú eljáráshívásból áll
  - ➢ sem itt, sem az AVLremMin eljárásnál nem befolyásolja a futási időnek az AVLinsert eljárásra is érvényes nagyságrendjét

➢ $\boldsymbol{MT \in \Theta(\log n)}$, ahol $n = |t|$

AVL fa magassága*

- **Tétel:** Tetszőleges *n* csúcsú nemüres AVL fa *h* magasságára:

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

- **Bizonyítás:**
  - $\lfloor \log n \rfloor \leq h$
    - elég azt belátni, hogy ez tetszőleges nemüres bináris fára igaz
      - Egy tetszőleges bináris fa 0.szintjén legfeljebb $2^0 = 1$ csúcs van, az 1. szintjén maximum $2^1 = 2$ csúcs, a 2. 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.

      ➢ egy *h* mélységű bináris fa csúcsainak *n* számára:

      $$n \leq 2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1 < 2^{h+1}$$

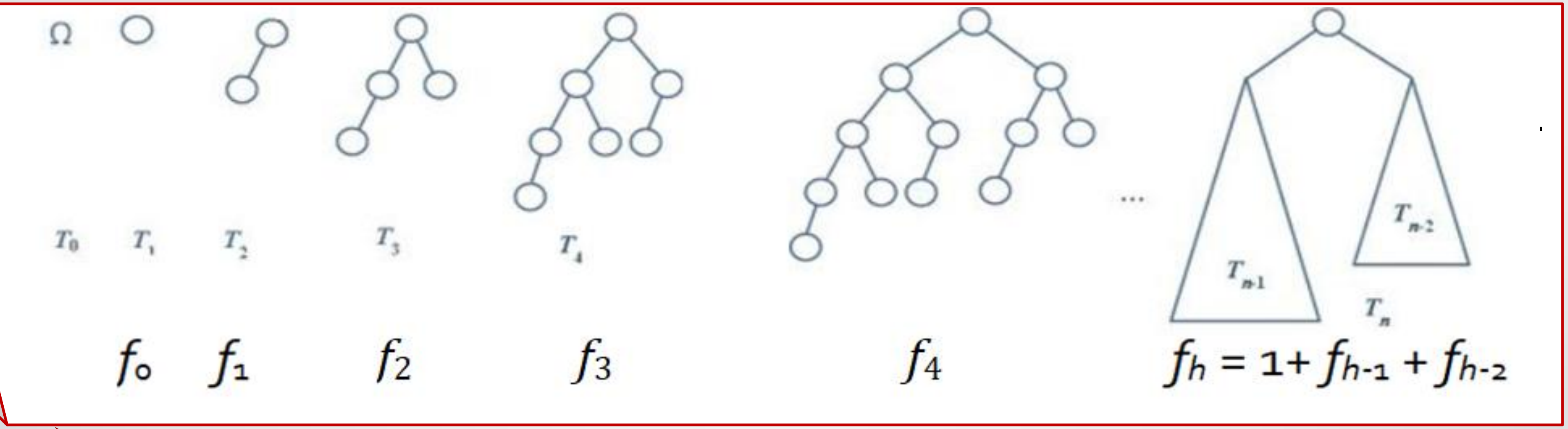
      ➢ $\lfloor \log n \rfloor \leq h < \log 2^{h+1} = h + 1$

      ➢ $\lfloor \log n \rfloor \leq h$

Bizonyítás folytatása

- $h \leq 1{,}45 \log n$ bizonyítása:
  - elég azt belátni, hogy ez tetszőleges nemüres KBF-ra igaz
    - KBF: kiegyensúlyozott, bináris fák
  - a $h \geq 0$ magasságú, KBF-ek csúcsainak minimális f_h számának meghatározása
    - $f_0 = 1$ (egy nulla magasságú KBF csak a gyökércsúcsból áll)
    - $f_1 = 2$ (egy magasságú KBF-ek pedig ((B)G(J)), ((B)G), vagy (G(J)) alakúak)
    - az $< f_0; f_1; f_2; \ldots >$ sorozat szigorúan monoton növekvő
      - 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!
      - ➢ $f_{i+1} = s(t) > s(f) \geq f_i$
    - *$h \geq 2$ esetén:* $f_h = 1 + f_{h-1} + f_{h-2}$

Bizonyítás folytatása

- *h ≥ 2 esetén:* $f_h = 1 + f_{h-1} + f_{h-2}$
  - 

    - $f_h = 1 + f_{h-1} + f_{h-2}$
  - A képlet emlékeztet a Fibonacci sorozatra: $F_0 = 0$, $F_1 = 1$, $F_h = F_{h-1} + F_{h-2}$, ha $h \geq 2$
- A h magasságú, és f_h méretű KBF-eket Fibonacci fáknak hívjuk.
- Az előbbiek szerint egy h-2 magasságú Fibonacci fa
  - (φ- 1 G φ- 2) vagy (φ- 2 G φ- 1) alakú
  - ahol φ- 1 és φ- 2 h - 1, illetve h -2 magasságú Fibonacci fák

Összefüggés az $< f_h : h \in \mathbb{N} >$ és az $< F_h : h \in \mathbb{N} >$ 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			

- $f_h = F_{h+3} - 1$
  $h \geq 0$

- Bizonyítás: Teljes indukcióval
  - Első néhány értékre: táblázat ($0 \leq h \leq k \geq 1$)
  - $h = k + 1$-re: $f_h = f_{k+1} = 1 + f_k + f_{k-1} = 1 + F_{k+3} - 1 + F_{k+2} - 1 = F_{k+4} - 1 = F_{h+3} - 1.$

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

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

Összefüggés az $< f_h : h \in \mathbb{N} >$ és az $< F_h : h \in \mathbb{N} >$ sorozatok között

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

- $2 < \sqrt{5} < 3 \quad \rightarrow \quad \frac{|1-\sqrt{5}|}{2} < 1$ és $\left|\left(\frac{1-\sqrt{5}}{2}\right)^{h+3}\right| < 1$

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

- Mivel $$\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{3} = \frac{1+3\sqrt{5}+3(\sqrt{5})^2+(\sqrt{5})^3}{8\sqrt{5}} = \frac{16+8\sqrt{5}}{8\sqrt{5}} = \frac{2+\sqrt{5}}{\sqrt{5}}$$
- Behelyettesítve az előbbi *n*-re vonatkozó összefüggésbe: $$n > \frac{2+\sqrt{5}}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h} - \frac{1+\sqrt{5}}{\sqrt{5}}$$

Összefüggés az $< f_h : h \in \mathbb{N} >$ és az $< F_h : h \in \mathbb{N} >$ sorozatok között

- Az első együtthatót tagokra bontva, a disztributív szabállyal:

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

- Eszerint $h \geq 1$ esetén: $n > \left(\frac{1+\sqrt{5}}{2}\right)^h$
- $h = 0 \rightarrow n = 1$

➢ Azaz tetszőleges, nemüres KBF n méretére és h magasságára: $n \geq \left(\frac{1+\sqrt{5}}{2}\right)^h$

- Vegyük mindkét oldal kettes alapú logaritmusát, majd osszunk $\log\frac{1+\sqrt{5}}{2}$ -tel: $h \leq \frac{1}{\log\frac{1+\sqrt{5}}{2}} \log n$

- $1{,}44 < 1{,}44042 < \frac{1}{\log\frac{1+\sqrt{5}}{2}} < 1{,}4404201 < 1{,}45$

➢ $\boldsymbol{h \leq 1{,}45 \log n}$

Általános fák

- Egy csúcsnak tetszőlegesen sok gyereke lehet
- Tetszőleges csúcshoz tartozó részfák száma pontosan egyenlő a gyerekek számával azaz nem tartoznak hozzá üres részfák
- Ha a gyerekek sorrendje lényeges: rendezett fákról beszélünk
- Általános fák használata:
  - modellezhetjük vele a számítógépünkben a mappák hierarchiáját, a programok blokkstruktúráját, a függvénykifejezéseket, a családfákat és bármelyik hierarchikus struktúrát.
- Általános fa nem általánosítása az *r*-áris fa fogalmának
  - ugyan minden konkrét általános fában van az egy csúcshoz tartozó gyerekek számára valamilyen *r* felső korlát
  - de ez a korlát nem abszolút: a fa tetszőleges csúcsa gyerekeinek száma tetszőlegesen növelhető
  - Másrészt: itt nem értelmezzük az üres részfa fogalmát
- Speciális csúcsok:
  - gyökércsúcs: nincs szülője
  - Levélcsúcs: nincs gyereke

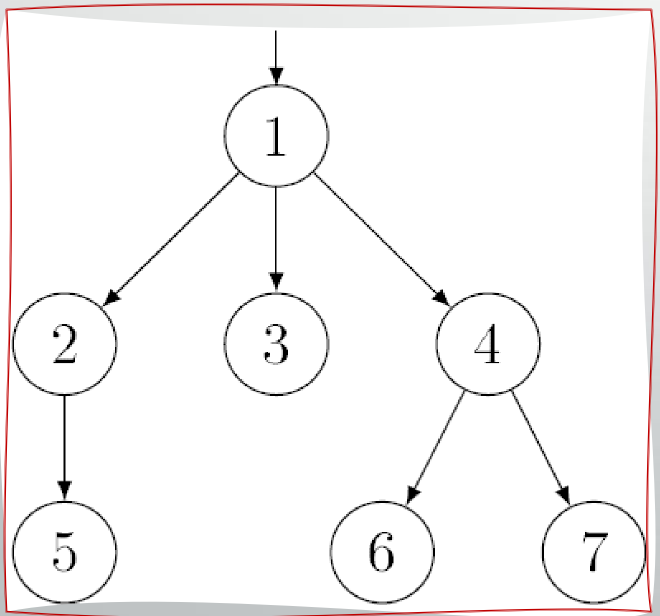
Általános fák ábrázolása

- Természetes ábrázolási módja: a bináris láncolt reprezentáció
  - egy csúcs szerkezete:
    - *child1*: az első gyerekre mutat
    - *sibling*: a következő testvérre mutat
    - *szülő* (opcionális): a gyerekek közös szülőjére mutat vissza
  - A **p* csúcs levél -> *p->child1* = $\emptyset$
  - A **p* csúcs utolsó testvér -> *p->sibling* = $\emptyset$

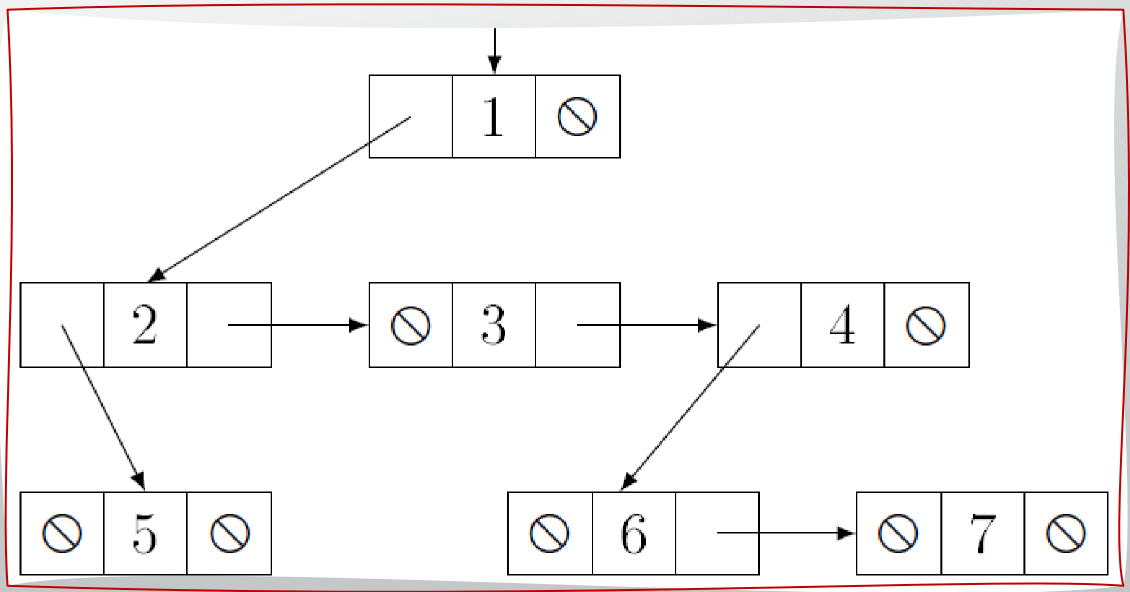
Node
+ $child1, sibling$: Node* // $child1$: első gyerek; $sibling$: következő testvér + key : T // T ismert típus
+ Node() { $child1 := sibling := \oslash$ } // egycsúcsú fát képez belőle + Node(x:T) { $child1 := sibling := \oslash$; $key := x$ }

Általános fák ábrázolása

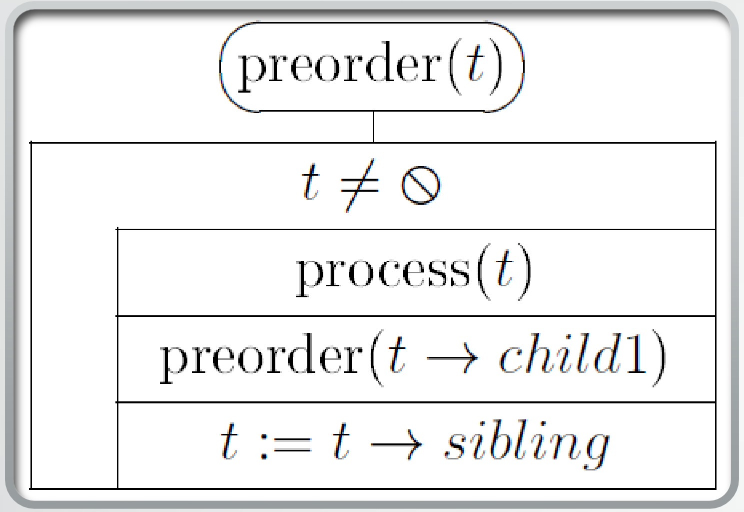
- Szöveges (zárójeles) reprezentáció:
  - A gyökér előre kerül
  - Egy nemüres fa általános alakja ($G\ t_1 \ldots t_n$)
    ahol G a gyökércsúcs tartalma, $t_1 \ldots t_n$ pedig a részfák
- { 1 [2 (5)] (3) [4 (6) (7)] } általános fa:
- Az általános fa absztrakt szerkezete:



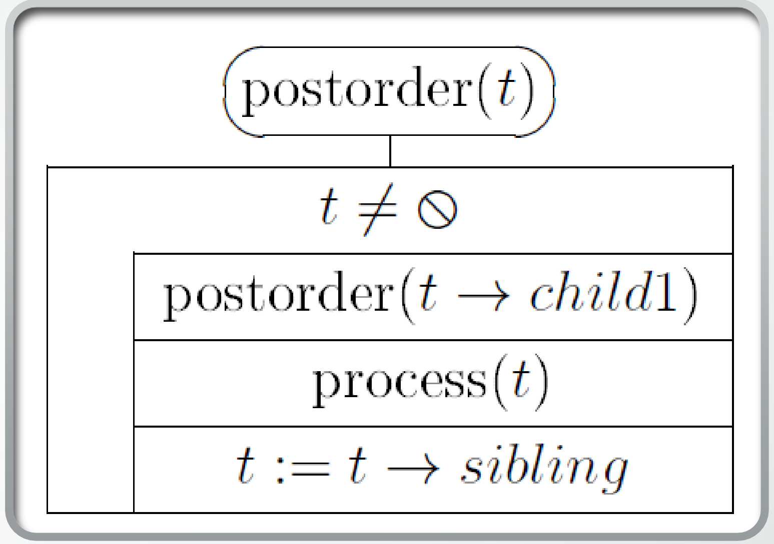
Az általános fa bináris reprezentációja:



Általános fa bejárásai



- Megfeleltetések:
  - *child1 ~ left*
  - *sibling ~ right*
- Felhasználása:
  - Fájlrendszereknél keresés



- Megfeleltetések:
  - *Postorder ~ inorder*
- Felhasználása:
  - függvénykifeJezések kiértékelése (kivéve a lusta kiértékelést)

- Inorder bejárás:
  - nem szimulálható a bináris reprezentáció egyik nevezetes bejárásával sem
  - A (G $t_1 \ldots t_n$) fa bejárása $n>0$ esetén: t_1, G $t_2 \ldots t_n$
    - $n=0$ esetén : G

Általános fa C# kód*

```csharp
using System;

public class Node<T>
{
    public T Key { get; set; }
    public Node<T>? Child { get; set; }   // első gyerek
    public Node<T>? Sibling { get; set; } // következő testvér

    // Üres konstruktor
    public Node()
    {
        Child = null;
        Sibling = null;
    }

    // Kulcsot kapó konstruktor
    public Node(T key)
    {
        Key = key;
        Child = null;
        Sibling = null;
    }
}
```

```csharp
public class GeneralTree<T>
{
    private readonly Action<T> _process;

    public GeneralTree(Action<T> process)
    {
        _process = process;
    }
```

```csharp
// Preorder bejárás (gyökér -> gyerekek -> testvérek)
public void Preorder(Node<T>? t)
{
    while (t != null)
    {
        _process(t.Key);          // process(t)
        Preorder(t.Child);        // preorder(t->child1)
        t = t.Sibling;            // t := t->sibling
    }
}
```

```csharp
// Postorder bejárás (gyerekek -> gyökér -> testvérek)
public void Postorder(Node<T>? t)
{
    while (t != null)
    {
        Postorder(t.Child);       // postorder(t->child1)
        _process(t.Key);          // process(t)
        t = t.Sibling;            // t := t->sibling
    }
}
}
```

```csharp
// Példa használatra
class Program
{
    static void Main()
    {
        // Példa fa:
        //        A
        //      / | \
        //     B  C  D
        //       / \
        //      E   F

        var root = new Node<string>("A");
        root.Child = new Node<string>("B");
        root.Child.Sibling = new Node<string>("C");
        root.Child.Sibling.Sibling = new Node<string>("D");
        root.Child.Sibling.Child = new Node<string>("E");
        root.Child.Sibling.Child.Sibling = new Node<string>("F");

        var tree = new GeneralTree<string>(Console.WriteLine);
        Console.WriteLine("Preorder:");
        tree.Preorder(root);
        Console.WriteLine("\nPostorder:");
        tree.Postorder(root);
    }
}
```

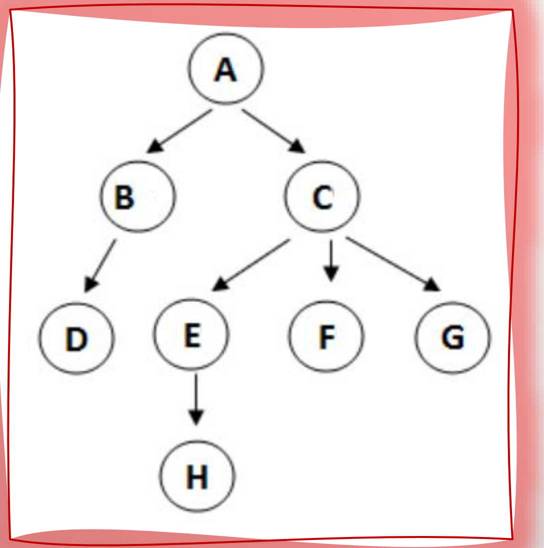
Kimenet:

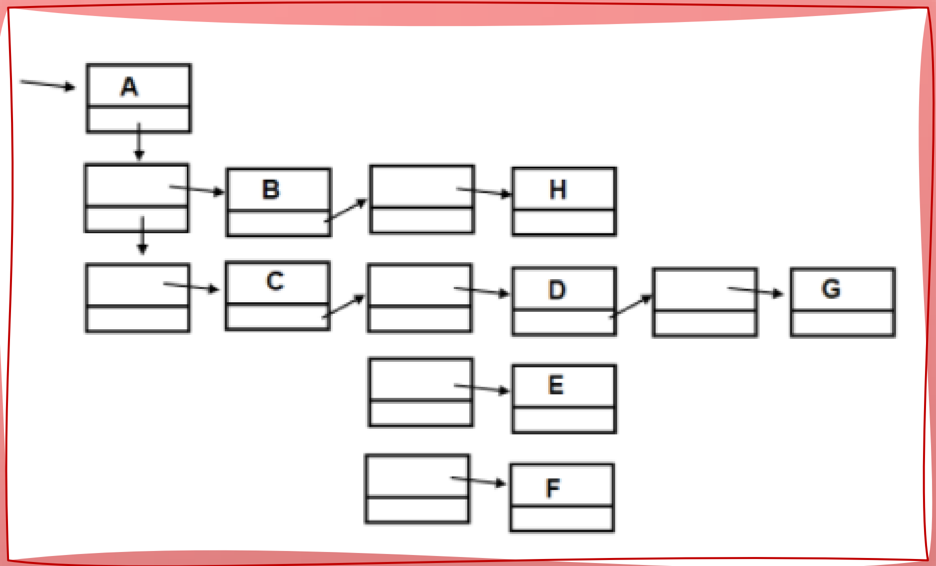
```
Preorder:
A
B
C
E
F
D
```

```
Postorder:
B
E
F
C
D
A
```

Általános fák ábrázolása másképp*

Multilistás ábrázolás





Minden csomópont egy láncolt lista. A lista első eleme tartalmazza az adatot, a többi csomópont már csak hivatkozásokat a leszármazottakra

Ellenőrző kérdések

1. Mit jelöl a **d** változó az AVLremMin eljárásban?
2. Hogyan biztosítja az **AVLremMin** algoritmus, hogy a fa magasság-különbsége mindig legfeljebb 1 maradjon?
3. Adj példát arra, mikor hívódik meg a **leftSubTreeShrunk(*t, d*)** eljárás!
4. Szemléltessük az 1 2 7 3 5 6 8 9 4 számok egymás utáni beszúrását egy, kezdetben üres AVL fába! Az így adódó fából indulva, ezután szemléltessük az 1 3 4 8 9 2 számok egymás utáni törlését! Minden egyes beszúrás illetve törlés után rajzoljuk újra fát! Jelöljük, ha ki kell egyensúlyozni, a kiegyensúlyozás helyét, és a kiegyensúlyozás után is rajzoljuk újra fát! A rajzokon jelöljük a belső csúcsok egyensúlyait is, a szokásos módon!
5. Rajzoljunk 0, 1, 2, 3, 4 és 5 magasságú Fibonacci fákat, amelyek egyben AVL fák is!
6. Mutassunk olyan, öt magasságú AVL fát, amelynek adott levelét törölve, minden, a fában a törölt csúcs feletti szinten ki kell egyensúlyozni!
7. Az előadásról ismert **leftSubTreeShrunk(*t, d*)** eljárás mintájára dolgozzuk ki az ott csak felhasznált **rightSubTreeShrunk(*t, d*)** eljárást, ennek segédeljárásait, és az ehhez szükséges kiegyensúlyozási sémát!

Köszönöm a figyelmet!

Pusztai Kinga

A bemutató Ásványi Tibor: Algoritmusok és adatszerkezetek II. Előadásjegyzete (Fák) és Fekete István: Algoritmusok és adatszerkezetek / AVL-Fák előadásjegyzete alapján készült.