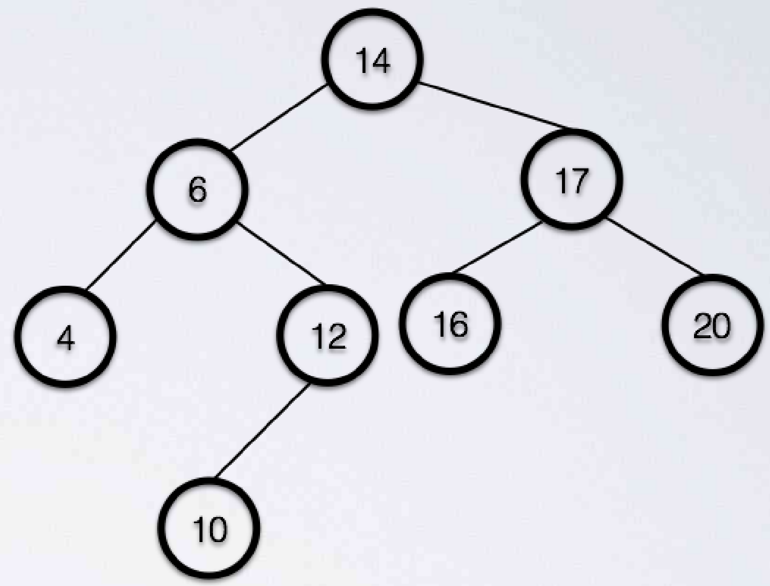
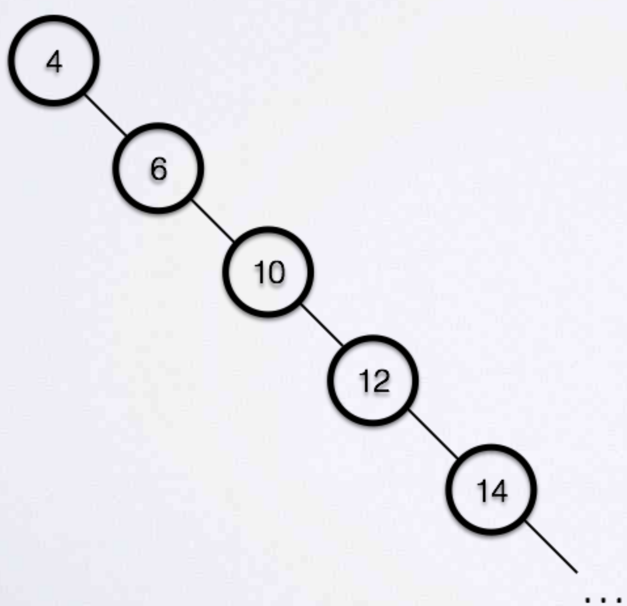
SDA CURS 4: Arbori binari de cautare echilibrati. Arbori AVL

ARBORI BINARI DE CAUTARE

- Cautare: *O(h)*
- Inserare: *O(h)*
- Stergere: *O(h)*
- *h = log n ?*
- Cazul defavorabil?





PERFORMANTA ABC

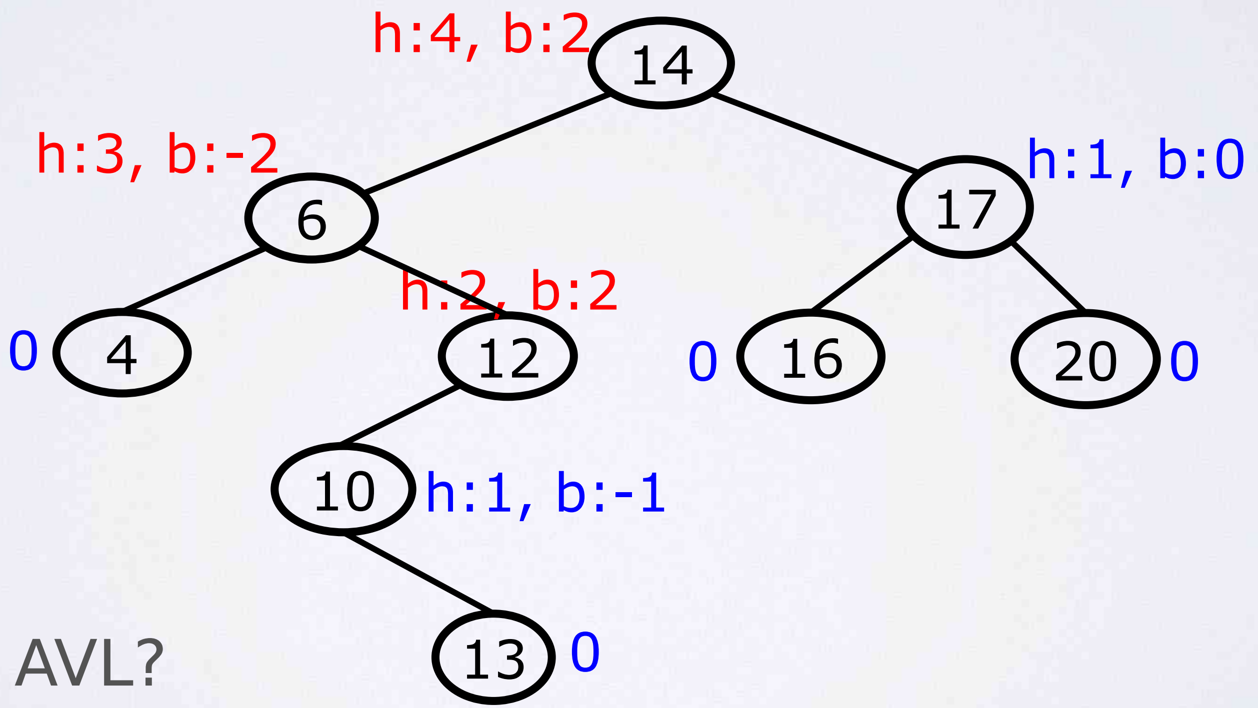
Putem garanta *h~log n*?

- constructia initiala
  - chei ordonate -> mediana
  - operatii ulterioare de inserare/stergere nu garanteaza mentinerea conditiei
  - <u>Perfectly Balanced Trees</u>: |*balance(n)*| *≤1,* where balance(n) = |no_nodes(n.left) – no_nodes(n.right)|
- noduri inserate aleator:
  - necesitatea unei conditii de echilibru, care:
    1. sa garanteze ca <u>*inaltimea este log n*</u> in orice situatie
    2. este <u>*usor de mentinut*</u> la inserare/stergere

ARBORI AVL
ADELSON-VELSKI-LANDIS

pentru orice nod *n*:

- **|*balance(n)*|** ≤ 1, unde
  - height(n) = nr max muchii de la nod la o frunza
  - balance(n)=height(n.left) - height(n.right)



Este AVL?

Care este cel mai adanc nod dezechilibrat?

AVL - CONDITIA DE ECHILIBRU

1. Sa garanteze ca inaltimea este *O(log n)*
   - *n(h)* - numarul minim de noduri pt. AVL de inaltime h
   - *n(0) = 1, n(1)=2*
   - *n>=2,* AVL contine <u>cel putin</u>:
     - *radacina*
     - *sub-arbore AVL de inaltime h-1*
     - *sub-arbore AVL de inaltime h-2*
   - Adica: $n(h) = 1 + n(h-1) + n(h-2)$
   - Intrucat $n(h-1) > n(h-2) \Rightarrow n(h) > 2n(h-2) > 4n(h-4) > 8n(h-6) > \ldots > 2^i n(h-2i)$
   - Rezolvand: $n(h) > 2^{h/2}$
   - Aplicam log: $h < 2\log n(h)$

AVL - CONDITIA DE ECHILIBRU

2. Usor de intretinut
   - traditional, se mentine factorul de echilibru (balance factor) la fiecare nod: +1, 0 sau -1
   - Proprietatile algoritmului de echilibrare
     - dupa *Insert*:
       - modificarea informatiei de echilibru se produce la mai multe noduri inspre radacina, DAR
       - de indata ce s-a executat o rotatie simpla/dubla arborele se re-echilibreaza
     - dupa *Delete:*
       - este posibil sa fie nevoie de rotatii pe <u>toate</u> nodurile de pe calea de cautare

AVL - OPERATII

Cautare: la fel ca si in ABC

Insert:

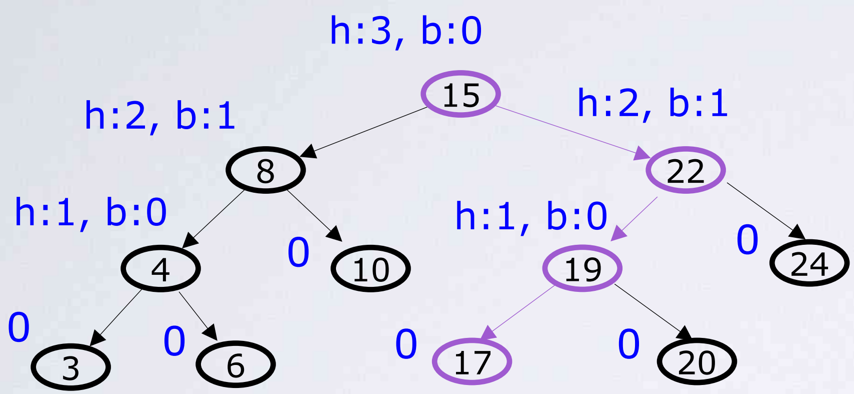
- inserare ca si frunza (ca si in ABC)
- verificare echilibru
- echilibrare (4 cazuri diferite)
  - se rezolva cu rotatii simple/duble
  - un cel mai adanc nod care este dezechilibrat
    - daca acesta se reechilibreaza, totul deasupra lui este echilibrat! (de ce?)

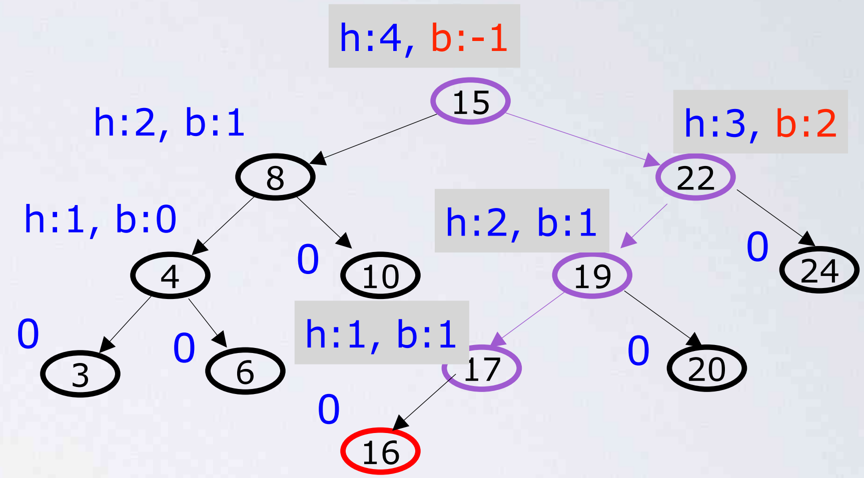
Stergere:

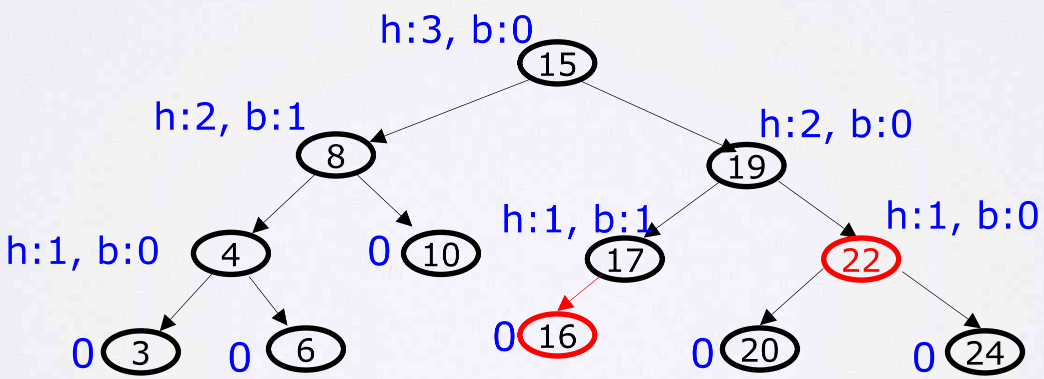
- se elimina nodul ca si in ABC
- se reechilibreaza arborele

AVL - EXEMPLU INSERARE - ROTATIE SIMPLA

Insert(16)







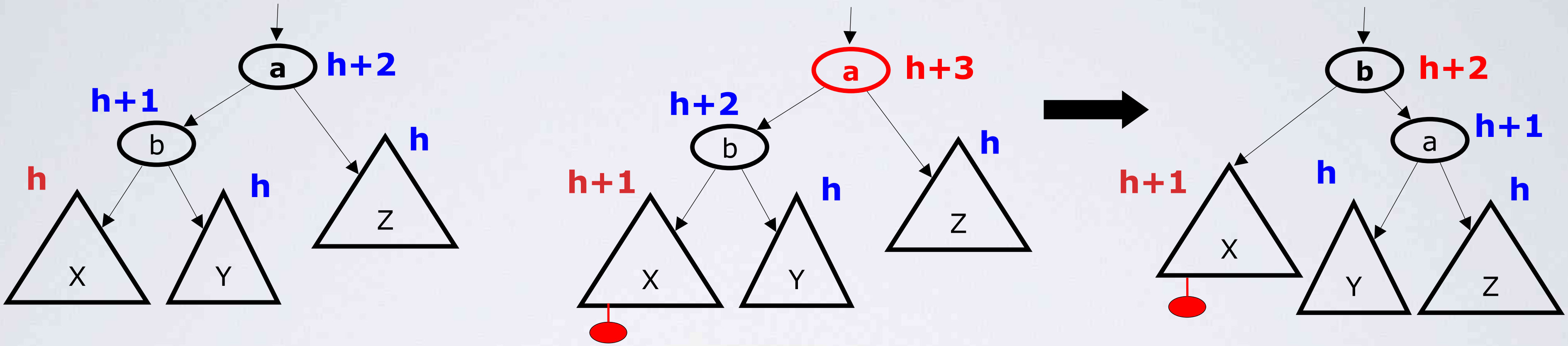
Ce puteti spune despre inaltimea celui mai adanc nod unde s-a produs ezechilibrul? (inaltimea de dinainte de insert, si cea de dupa echilibrare)

AVL – TIPURI DE ROTATII

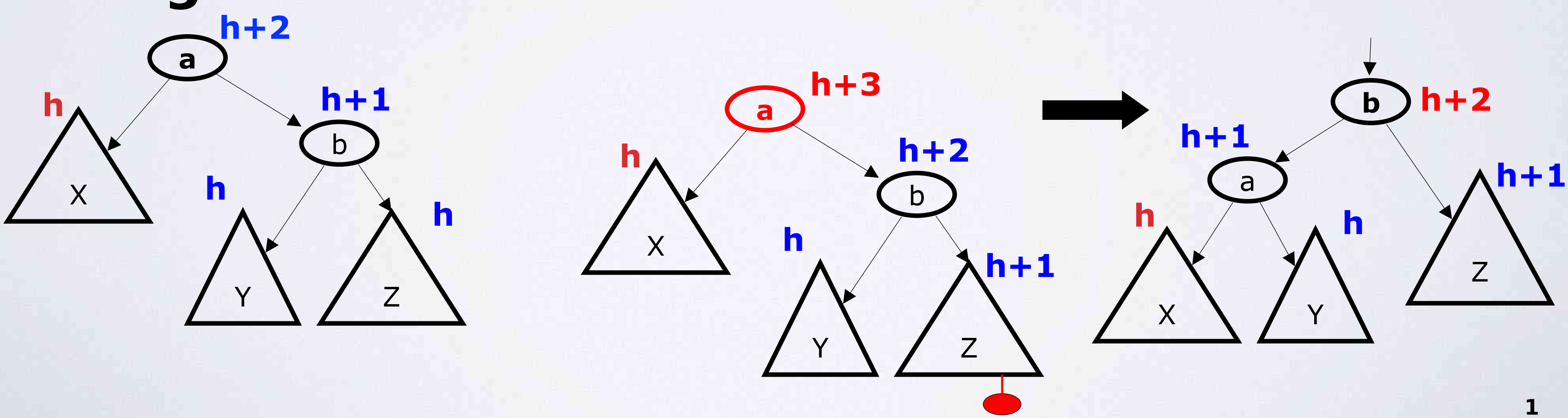
- Rotatii
  - Simple
    - **Stanga – L Rotation:** nod inserat in subarborele drept al unui copil cu dezechilibru pe dreapta
    - **Dreapta – R Rotation:** nod inserat in subarborele stang al unui copil cu dezechilibru pe stanga
  - Duble
    - **Stanga Dreapta – LR Rotation** nod inserat in sub-arborele **drept** al copilului cu dezechilibru pe **stanga**
    - **Dreapta Stanga – RL Rotation** nod inserat in sub-arborele **stang** al copilului cu dezechilibru pe **drepta**

AVL - ROTATII SIMPLE

Dreapta *Right rotation*

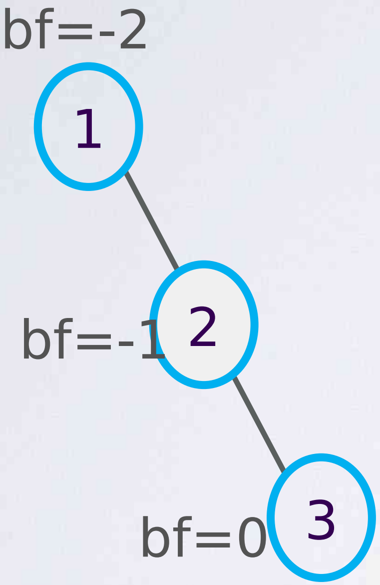


Stanga *Left rotation*

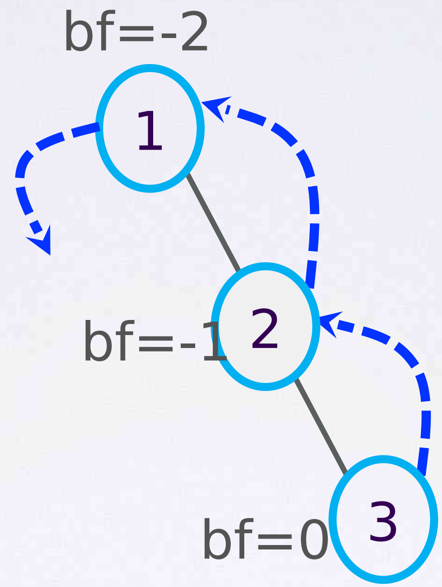


ROTATIE STANGA - EXEMPLU

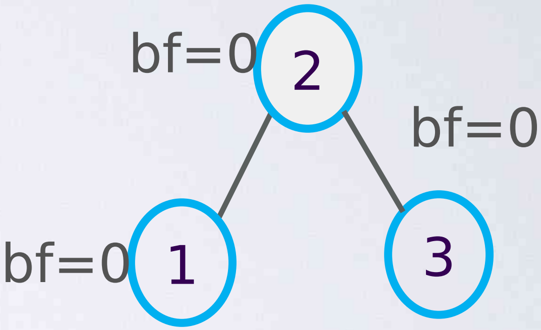
Insert 1, 2, 3



Arbore ne-echilibrat



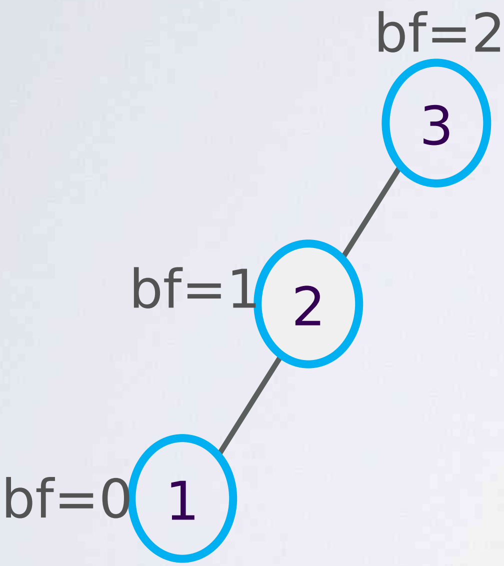
Ca sa il echilibram folosim rotatia left (stanga) care “muta” nodurile cu o pozitie la stanga.



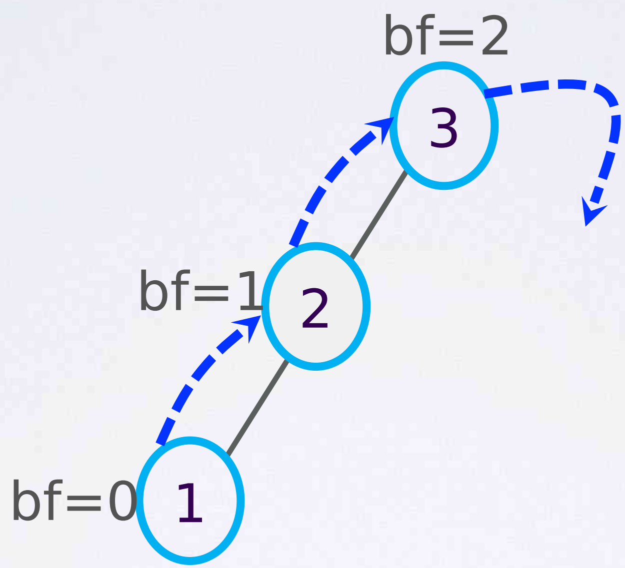
Dupa rotatia stanga arborele devine echilibrat.

ROTATIE DREAPTA – EXEMPLU

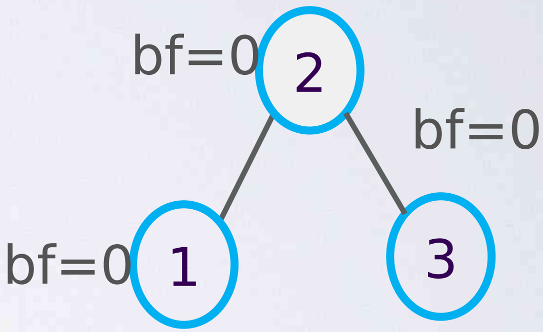
Insert 3, 2, 1



Arbore ne-echilibrat deoarece nodul 3 are bf = 2

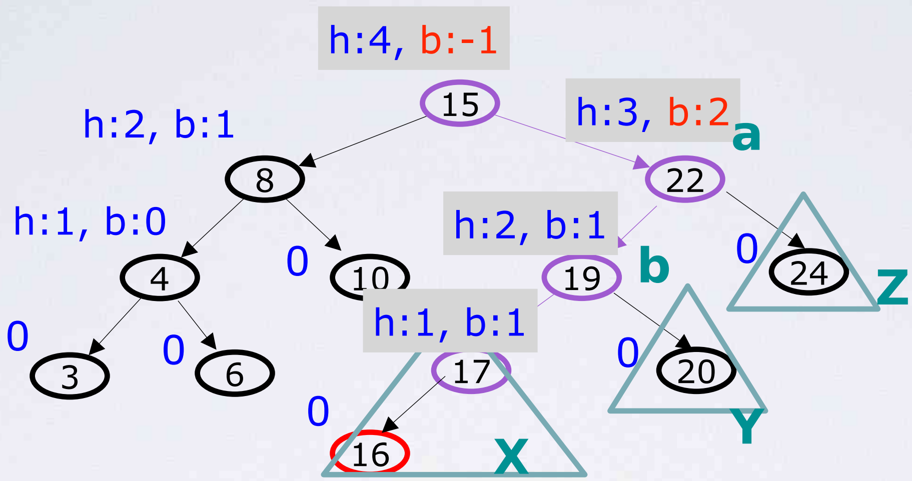


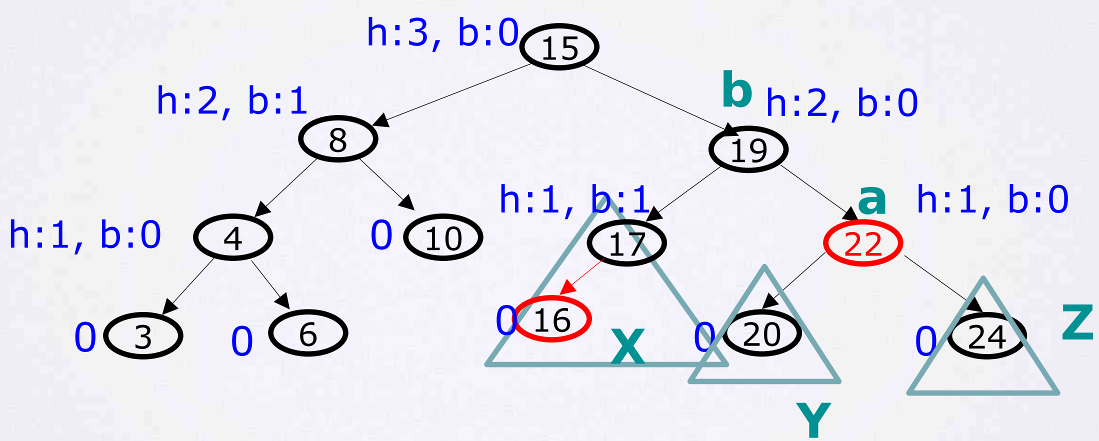
Ca sa il echilibram folosim rotatia dreapt care “muta” nodurile cu o pozitie la dreapta.



Dupa rotatia dreapta (right) arborele devine echilibrat.

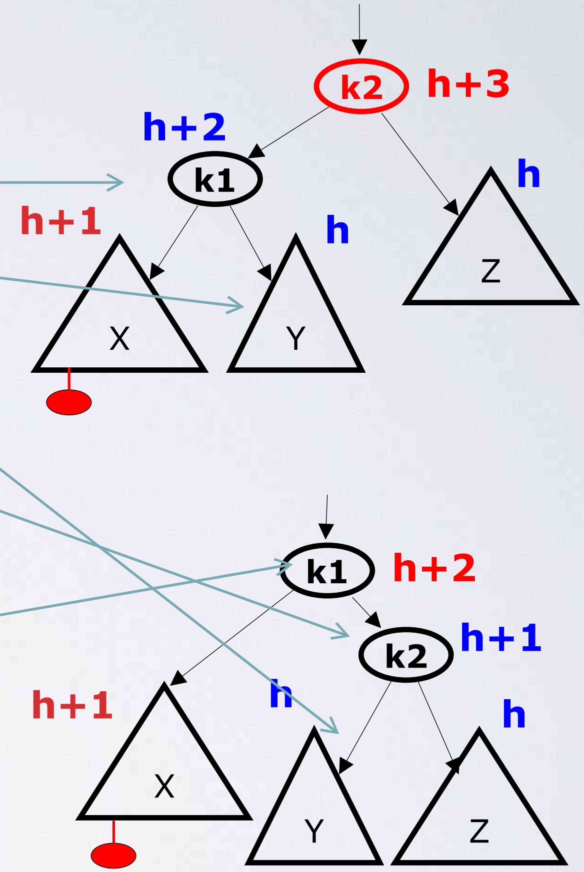
AVL - ROTATIE DREAPTA EXEMPLU





AVL - ROTATIE DREAPTA (EXEMPLU COD)

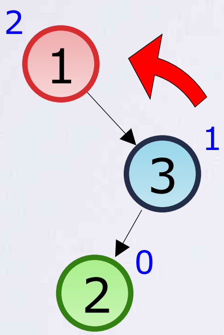
```
void snglRotRight(AVLNodeT **k2)
{
   AVLNodeT *k1 ;
   k1 = (*k2)->left ;
   (*k2)->left  = k1->right ;
   k1->right = *k2 ;
   (*k2)->height = max(
        (*k2)->left -> height,
        (*k2)->right -> height) + 1;
   k1->height = max(
      k1-> left -> height,
     (*k2)-> height ) + 1 ;
   *k2 = k1; // assign new  root
}
/* snglRotLeft is symmetric */
```



AVL - PROPRIETATI ROTATII

- Nodurile din sub-arborele nodului rotat nu sunt afectate!
- O rotatie ia O(1)
- Inainte si dupa arborele isi pastreaza ordonarea de ABC
- Nota: codul pentru rotatie stanga este simetric

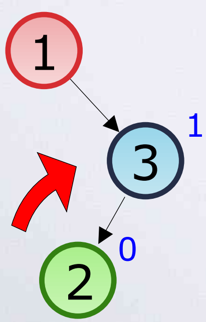
Rotatiile simple nu sunt suficiente!!!



(1)

What if….(2), apoi (1)?

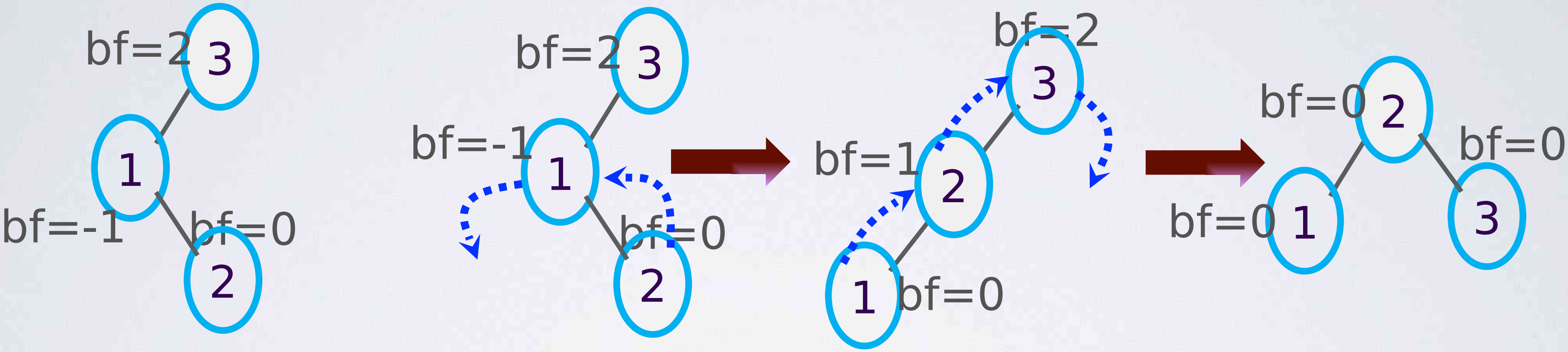
- rotatie intre copil si nepot problematici
- …apoi intre nod si noul copil



(2)

ROTATIE STANGA DREAPTA (LR ROTATION)

Insert 3, 1, 2



Arbore ne-echilibrat deoarece nodul 3 are bf = 2

Rotatie stanga

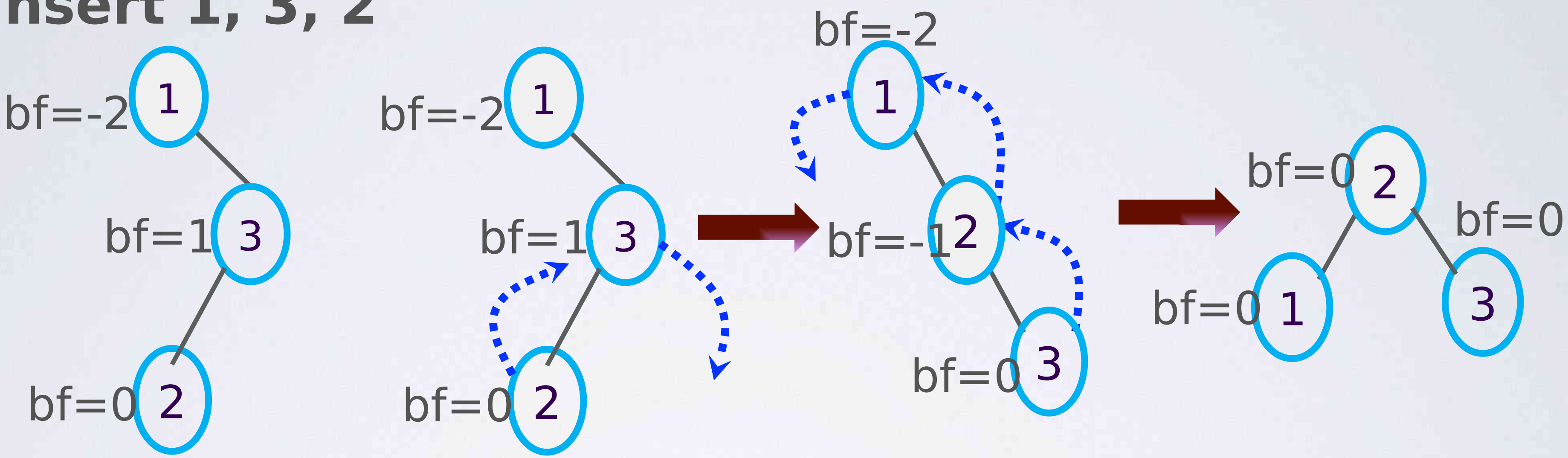
Rotatie dreapta

Arbore echilibrat

ROTATIE DREAPTA STANGA (RL ROTATION)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Insert 1, 3, 2



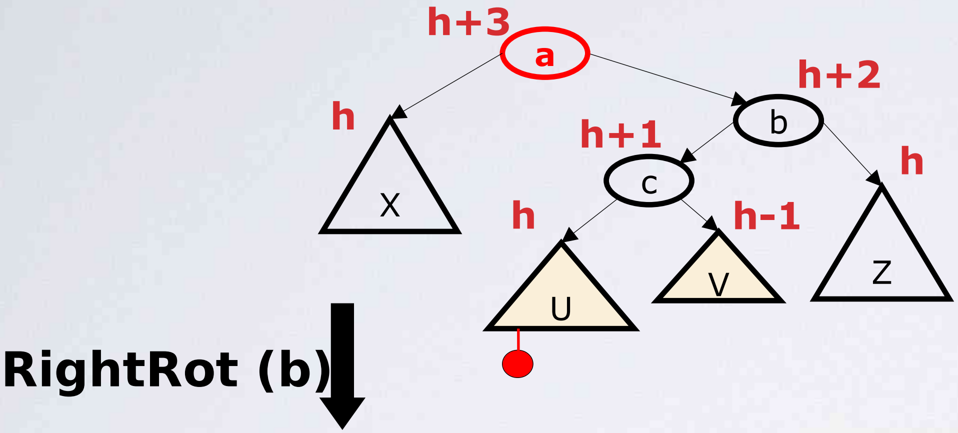
Arbore ne-echilibrat deoarece nodul 1 are bf = -2

Rotatie dreapta

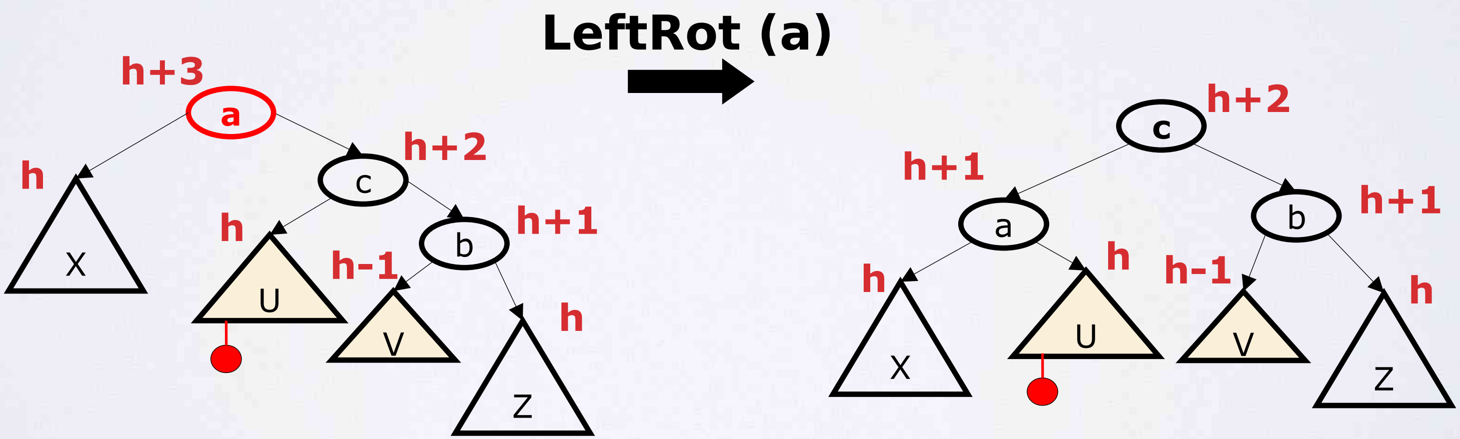
Rotatie stang

Arbore echilibrat

AVL - ROTATIE DUBLA: DREAPTA-STANGA

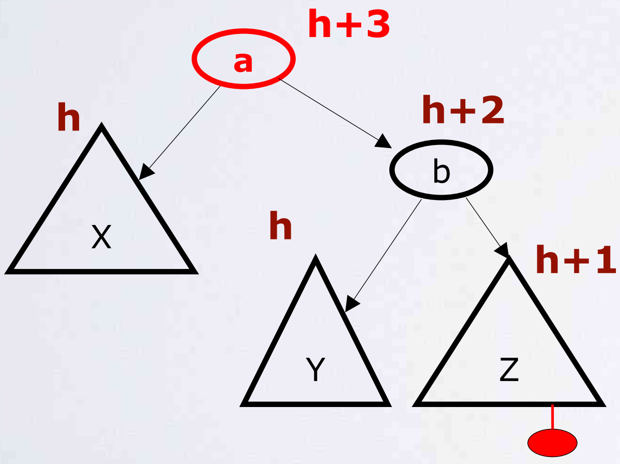



```
void  dblRotLeft( AVLPtr k3 )
{
   // rotate between k1 and k2
   snglRotRight((*k3)->left);
   // rotate between k3 and k2
   snglRotLeft(k3);
}
```

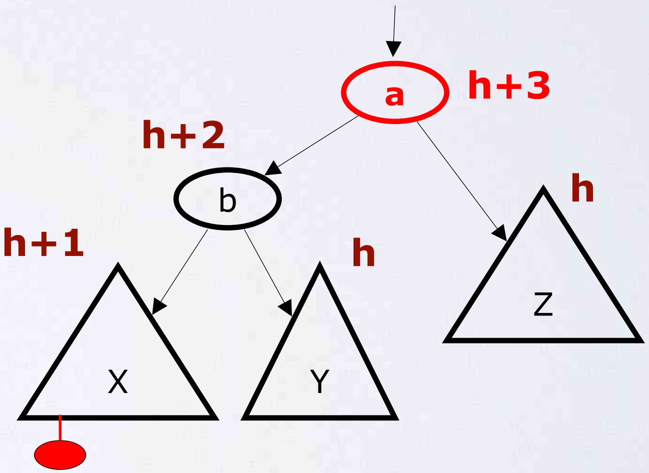


AVL - CUM FOLOSIM ROTATIILE SIMPLE?

Rotatie stanga: cand un nod este inserat la ***dreapta*** copilului **dreapta (b)** a celui mai apropiat stramos cu *bf* = *-2 (dupa inserare)* **(a)**



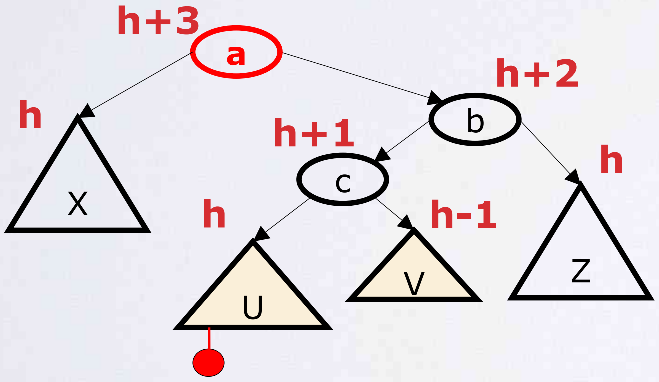
Rotatie dreapta: cand un nod este inserat in sub-arborele **stang** al copilului **stanga (b)** al celui mai apropiat stramos *cu bf = +2 (dupa inserare)* **(a)**



AVL - CUM FOLOSIM ROTATIILE DUBLE?

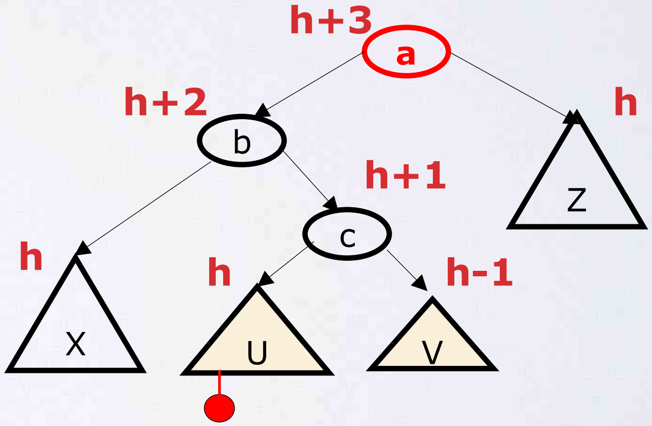
Dreapta-stanga: cand un nod este inserat in sub-arborele **stang** al copilului **drept (b)** al celui mai apropiat stramos cu

bf = -2 (dupa inserare) **(a)**



Stanga-dreapta: cand un nod este inserat in sub-arborele **drept** al copilului **stang (b)** al celui mai apropiat stramos cu

bf = +2 (dupa inserare) **(a)**



AVL - INSERT - COMPLEXITATE

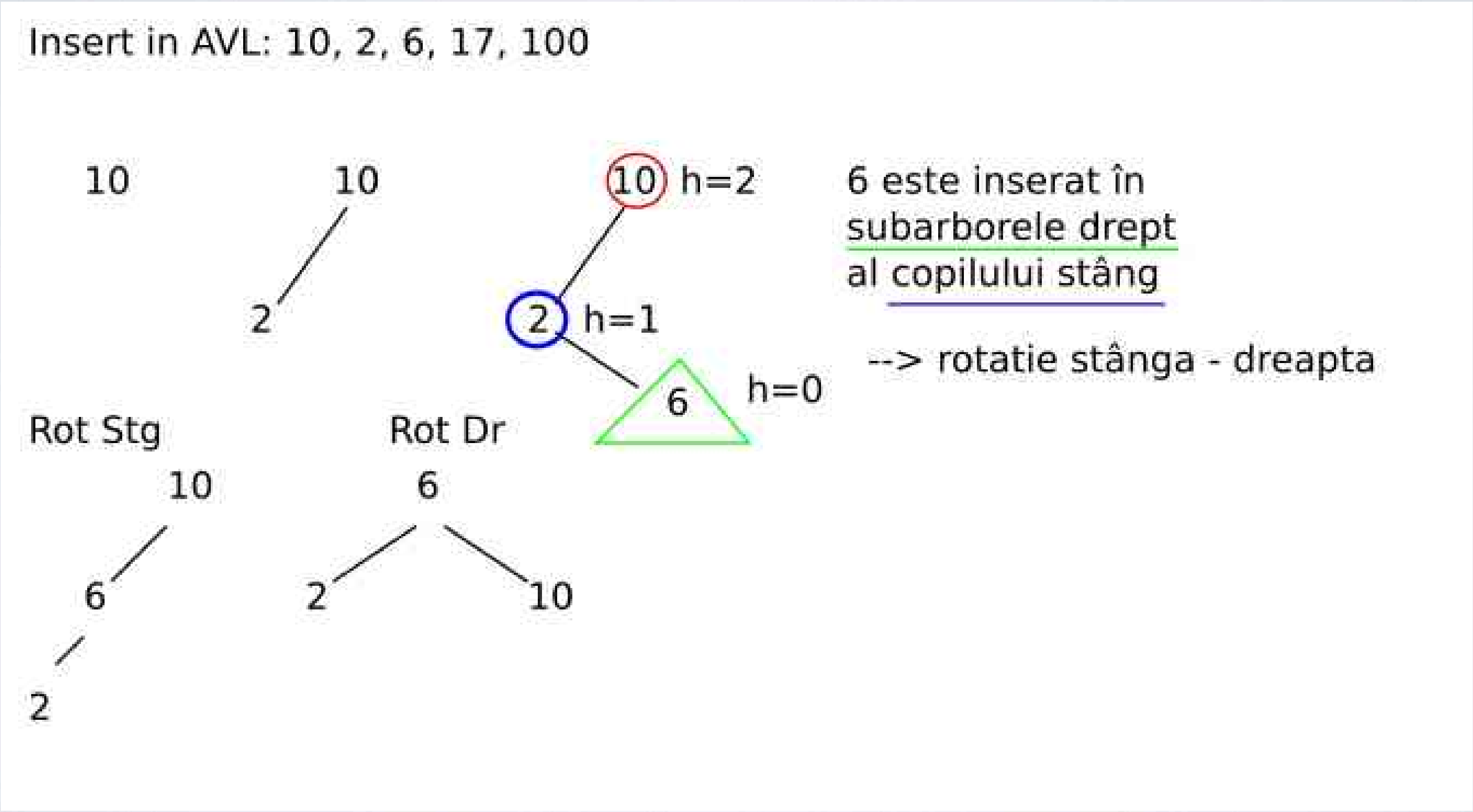
Cazul defavorabil: *O(log n)*

- Rotatie: *O(1)*
- Lungimea caii catre radacina: *O(log n)* (de ce?)
- Cel mult 2 rotatii la o inserare (de ce?)

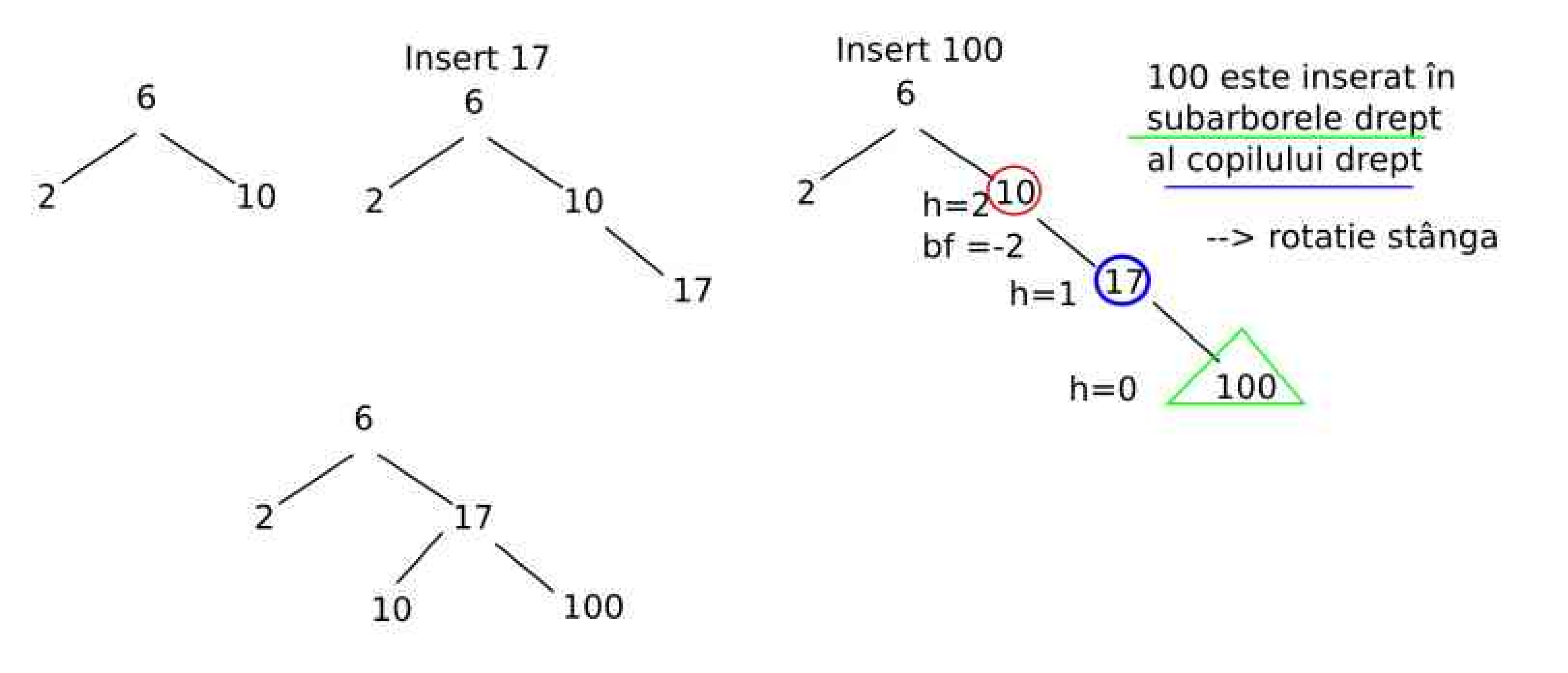
Complexitate *Search*?

Complexitate *constructie* arbore?

AVL - INSERT - EXEMPLU



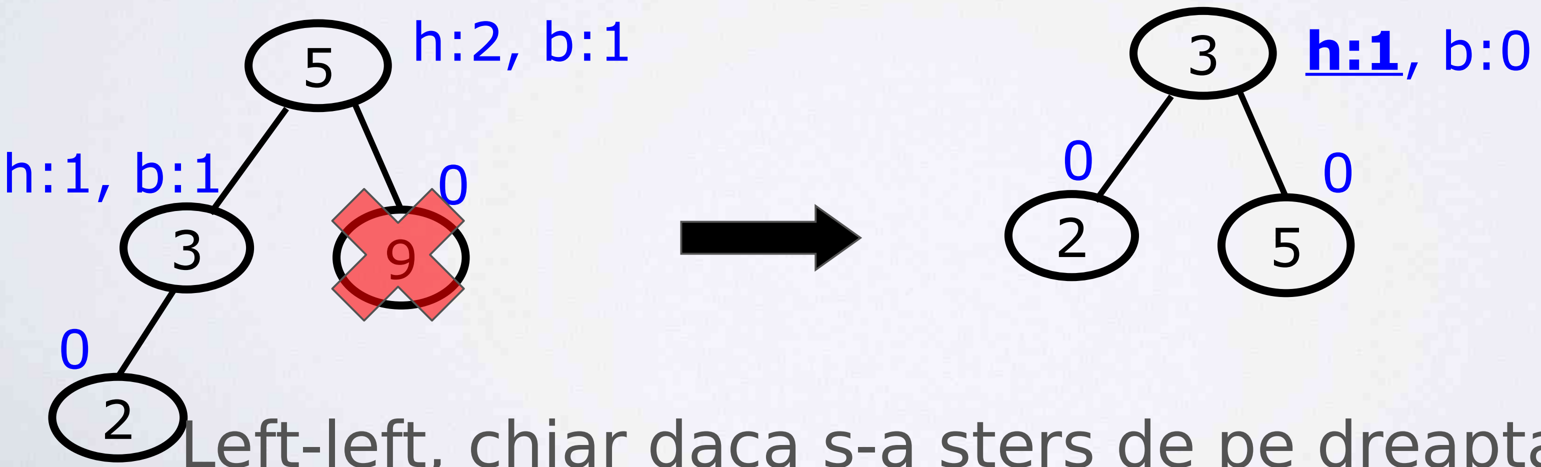
AVL - INSERT - EXEMPLU



AVL - STERGERE

- Eliminarea nodului se face folosind strategia ABC de inlocuire cu succesor/predecesor
- Dezechilibrul se repara prin rotatii (simple/duble)
- Spre deosebire de inserare, ***1 sau 2 rotatii s-ar putea sa nu fie suficiente pentru a restabili echilibrul in arbore!*** *De ce?*

Exemplu: insert(5), insert(3), insert(9), insert(2), delete(9)

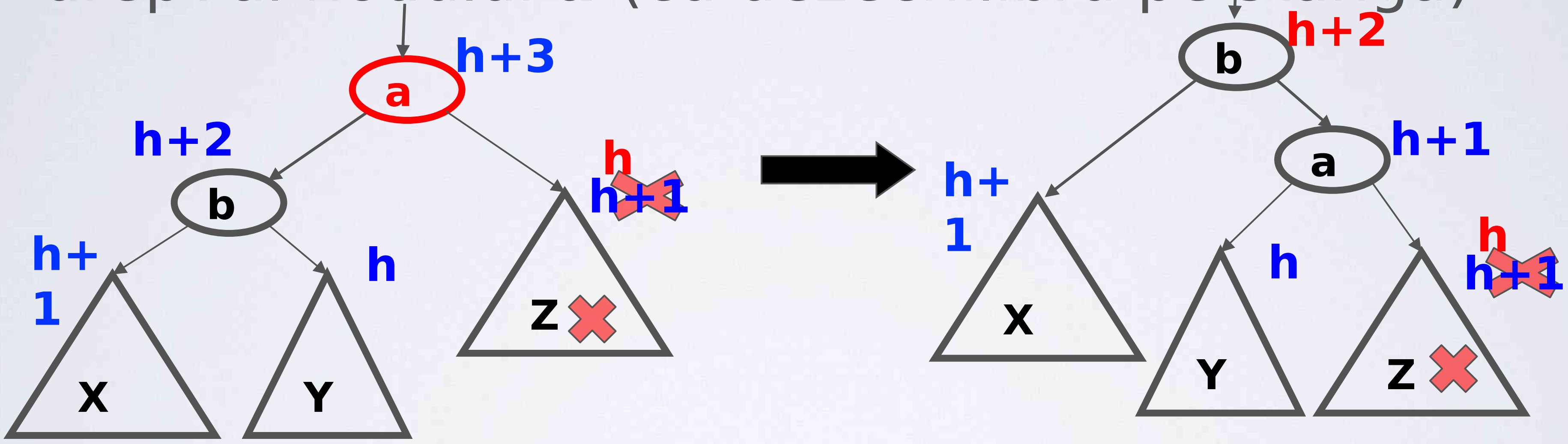


Left-left, chiar daca s-a sters de pe dreapta; inaltimea subarborelui final se modifica

AVL - STERGERE - RE-ECHILIBRARE

CAZ #1: LEFT-LEFT

- Datorata stergerii unui nod din subarborele drept al nodului **a** (cu dezechilibru pe stanga)



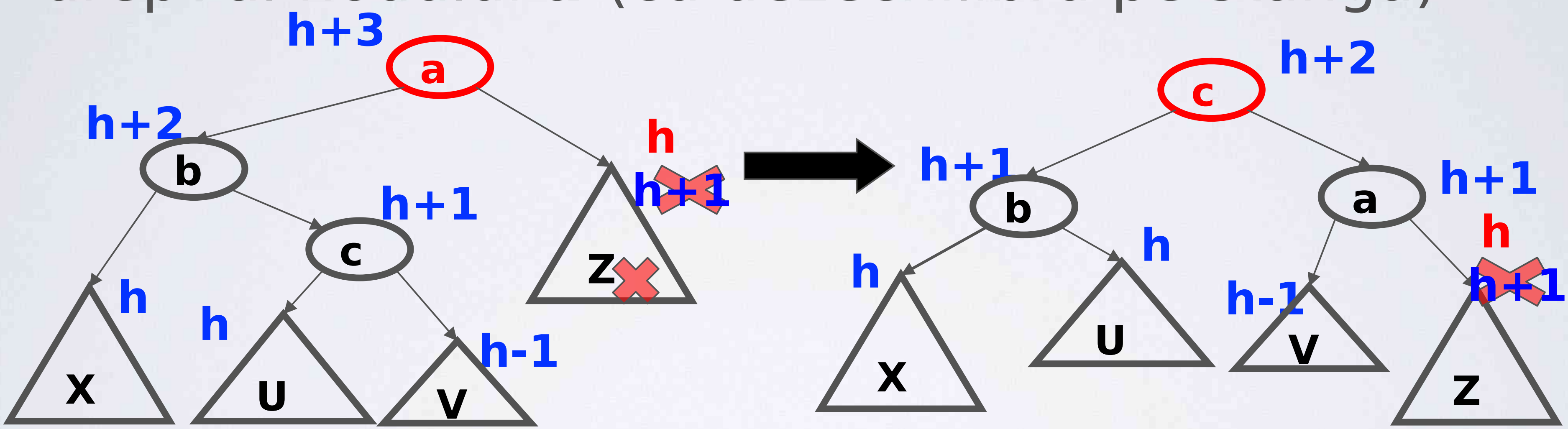
Rotatie simpla la dreapta asupra nodului **a**
(ca si la inserare in X) DAR!
Inaltimea subarborelui rezultat scade cu 1 =>
s-ar putea sa fie nevoie sa reechilibram mai sus !!

AVL - STERGERE - RE-ECHILIBRARE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

CAZ #2: LEFT-RIGHT

- Datorata stergerii unui nod din subarborele drept al nodului **a** (cu dezechilibru pe stanga)



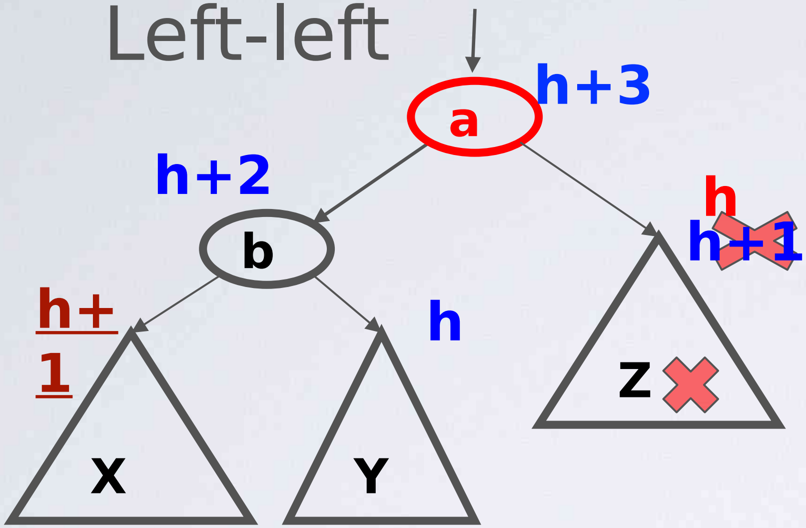
Aceeasi rotatie dubla ca si la inserarea left-right, cand **c** devine mai inalt

DAR!

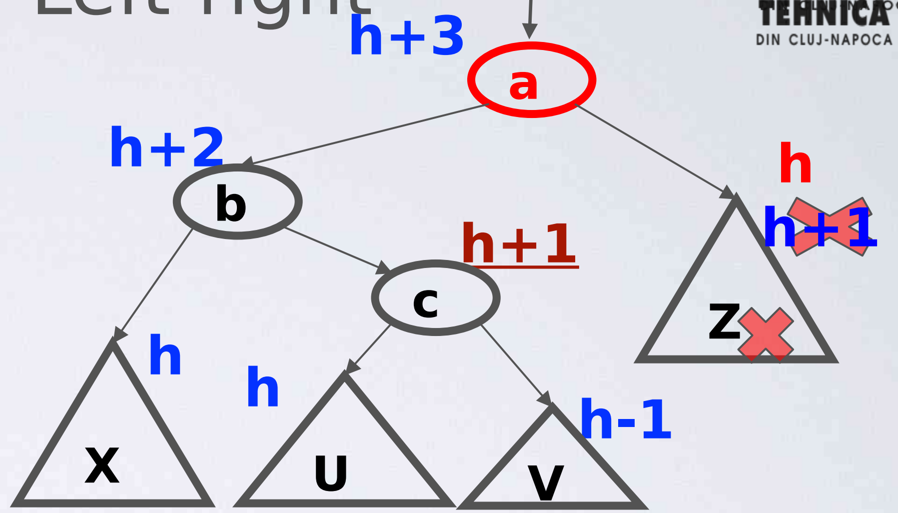
Inaltimea subarborelui rezultat scade cu 1 =>

s-ar putea sa fie nevoie sa reechilibram mai sus !!

Left-left



Left-right



Cazurile de pana acum: unul din nepotii de pe stanga au inaltime h+1

Ce se intampla daca amandoi ajung sa fie “prea inalti”?

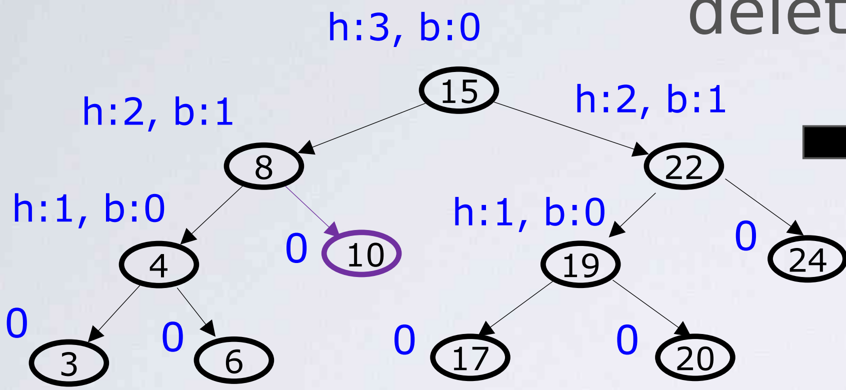
- #1 (Left-left) functioneaza si in cazul asta
- Copiii noului nod din varf vor avea inaltimi ce vor diferi cu 1, dar subarborele este echilibrat

AVL - STERGERE - RE-ECHILIBRARE

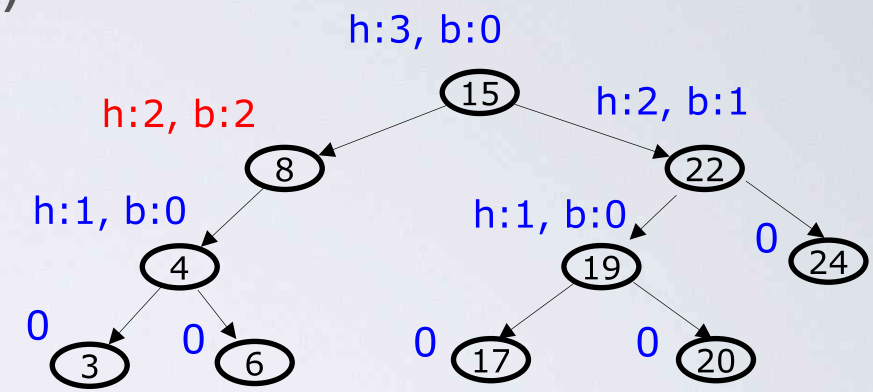
Cazurile #3 RIGHT-RIGHT, #4 RIGHT-LEFT

- Right-right: stergerea de pe stanga face ca nepotul dreapta-dreapta sa devina prea inalt
- Right-left: stergerea de pe stanga face ca nepotul dreapta-stanga sa devina prea inalt
- (daca ambii nepoti dreapta devin prea inalti, cazul #3 functioneaza)

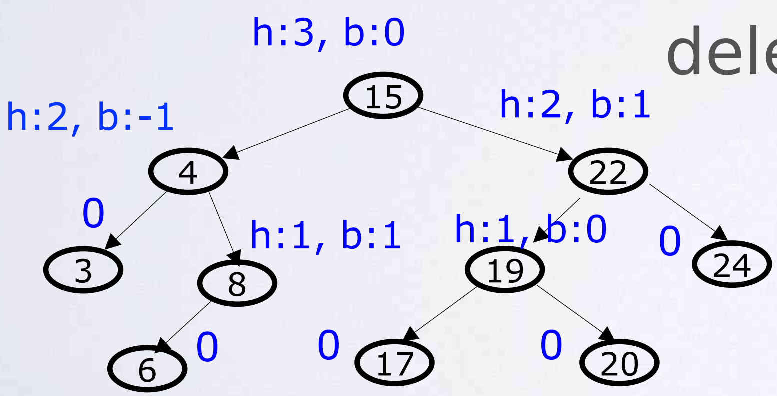
delete(10)



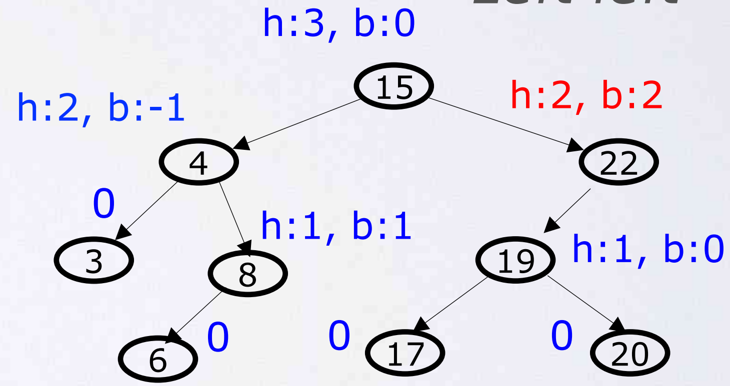
Left-left



delete(24)



Left-left



...

AVL - STERGERE

Delete - Complexitate

- search: *O(log n)*
- reechilibrare prin rotatii *de la nodul sters* ***fizic*** *pana la radacina: O(log n)*

AVL – PROS & CONS

- Pro:
  - Operatii in timp O(log n) in cazul defavorabil
  - Echilibrarea pe inaltime nu creste complexitatea cu mai mult de un factor constant
- Con:
  - Dificil de implementat si depanat
  - Memorie suplimentara pt informatia de inaltime
  - Asimptotic rapizi, dar in practica echilibrarea se simte
  - Multe cautari pe volume mari de date (e.g. baze de date) se fac pe disc, si atunci volumul arborelui il face sa nu mai incapa in memorie => avem nevoie de arbori mai “shallow” (e.g. B-trees)

ARBORI ROSU SI NEGRU

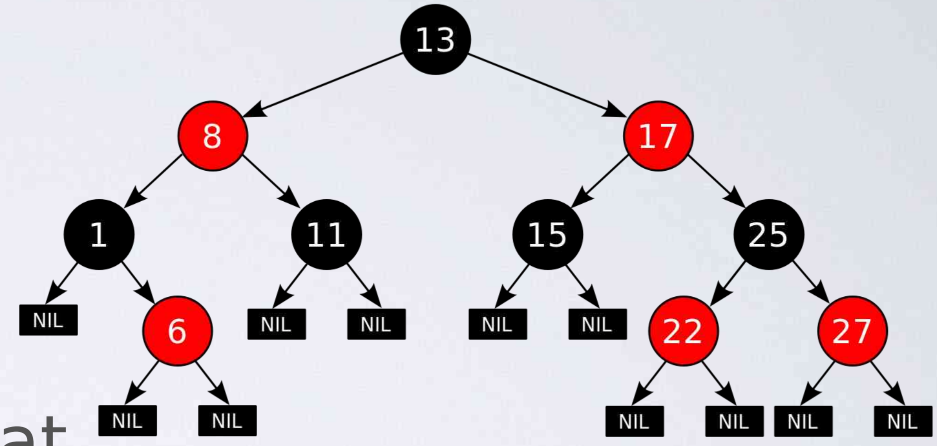
Orice nod: **rosu** sau **negru**

Radacina este **neagra**

Nodurile **NIL** sunt **negre**

Daca un nod este **rosu**, ambii copii sunt **negri**

Orice cale de la un nod dat la un NIL contine acelasi numar de noduri **negre** (*black depth)*



https://en.wikipedia.org/wiki/Red%E2%80%93black_tree#/media/File:Red-black_tree_example.svg

Calea de la radacina la cea mai indepartata frunza nu este mai lunga decat dublul lungimii drumului de la radacina la cea mai apropiata frunza.
Aproximativ echilibrat pe inaltime!
(mai multe detalii anul urmator la AF...)

PROBLEME PROPUSE

1. Inserare in AVL (sursa: aici)
2. Stergere din AVL (sursa: aici)
3. (*) Se da un ABC. Descrieti un algoritm, care colecteaza, intr-o lista, toate cheile care provin din noduri care sunt radacini de arbori AVL.

EXERCITII

EXERCITII

UNIVERSITATEA TEHNICA DIN CLUJ-NAPOCA

1. Se da urmatoarea secventa de chei: 14, 17, 11, 7, 53, 4, 13, 12, 8. Inserati cheile succesiv intr-un arbore AVL. Desenati arborele dupa fiecare inserare. Stergeti cheile 53, 11, 8. Desenati arborele dupa fiecare stergere.

2. Se da urmatoarea secventa de chei: 17, 4, 23, 6, 15, 11, 14, 18. Inserati cheile, succesiv, intr-un arbore AVL. Desenati arborele dupa fiecare inserare. Stergeti cheile: 23, 14. Desenati arborele dupa fiecare stergere.

3. Dati un exemplu de cea mai dezavantajoasa configuratie pentru un arbore AVL de inaltime 4.

4. Se da urmatoarea secventa de chei: A, V, L, T, R, E, E, I, S, O, K. Care va fi parintele nodului O in arbore?

5. Dar daca se da secventa: A, V, L, T, R, E, E, I, S, F, U, N. Care este parintele nodului F?

EXERCITII – PSEUDOCOD + COD

1. Descrieti un algoritm care verifica daca un arbore binar de cautare este sau nu AVL.
2. Se da un sir ordonat de chei intregi. Descrieti un algoritm eficient care construieste un ABC perfect echilibrat din cheile date in sir.
3. Se da un arbore binar de cautare. Se citeste de la tastarura un numar s. Sa se determine doua noduri din arbore care au suma egala cu s.
4. Se da un arbore binar de cautare. Sa se construiasca o lista dublu inlantuita din arborele dat (seria A - reluam problema de la Cursul 3)
5. Se da un arbore binar de cautare care stocheaza informatia de inaltime a fiecarui nod. Inserati nodurile 49, 29, 18, 32, 10, 2, 54, 50, 34, 67, 38. Proiectati si implementati urmatoarele functii:
   - Inserare in arborele binar de cautare.
   - Parcurgerea in inordine, postordine, preordine – cu afisarea cheilor nodurilor si a inaltimii pentru fiecare nod.
   - Crearea unei cozi in care puneti toate nodurile cu inaltime impara. Afisati continutul cozii.

BIBLIOGRAFIE

U. Washington, CSE332: Data Abstractions Lecture 7: AVL Trees - https://courses.cs.washington.edu/courses/cse332/10sp/lectures/lecture7.pdf

http://btechsmartclass.com/DS/U5_T2.html