

Căutare.

SD 2019/2020

Problema căutării

Căutare binară

Arbori binari de căutare

Arbori de căutare echilibrați

Problema căutării

- ▶ Aspectul static:

- ▶ U mulțime univers, $S \subseteq U$
- ▶ operația de căutare:
 - ▶ Instanță: $a \in U$
 - ▶ Întrebare: $a \in S?$

- ▶ Aspectul dinamic:

- ▶ operația de inserare
 - ▶ Intrare: $S, \quad x \in U$
 - ▶ Ieșire: $S \cup \{x\}$
- ▶ operația de ștergere
 - ▶ Intrare: $S, \quad x \in U$
 - ▶ Ieșire: $S - \{x\}$

Căutare în liste liniare - complexitate

Tip de date	Implementare	Căutare	Inserare	Ștergere
Lista liniară	Tablouri	$O(n)$	$O(1)$	$O(n)$
	Liste înlănțuite	$O(n)$	$O(1)$	$O(1)$
Lista liniară ordonată	Tablouri	$O(\log n)$	$O(n)$	$O(n)$
	Liste înlănțuite	$O(n)$	$O(n)$	$O(1)$

Problema căutării

Căutare binară

Arbori binari de căutare

Arbori de căutare echilibrați

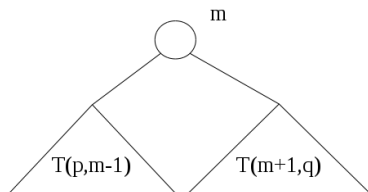
- ▶ Mulțimea univers este total ordonată: (U, \leq)
- ▶ Structura de date utilizată:
 - ▶ tabloul $s[0..n-1]$
 - ▶ $s[0] < \dots < s[n-1]$

Căutare binară: aspect static

```
Function poz( $s[0..n-1]$ ,  $n$ ,  $a$ )  
begin  
     $p \leftarrow 0$ ;  $q \leftarrow n - 1$   
     $m \leftarrow (p + q)/2$   
    while ( $s[m] \neq a$  and  $p < q$ ) do  
        if ( $a < s[m]$ ) then  
             $q \leftarrow m - 1$   
        else  
             $p \leftarrow m + 1$   
             $m \leftarrow (p + q)/2$   
        if ( $s[m] = a$ ) then  
            return  $m$   
        else  
            return  $-1$   
end
```

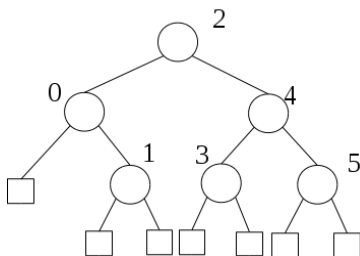
Arborele binar asociat căutării binare

$$T(p, q)$$



$$T = T(0, n - 1)$$

$$n = 6$$



Problema căutării

Căutare binară

Arbori binari de căutare

Arbori de căutare echilibrați

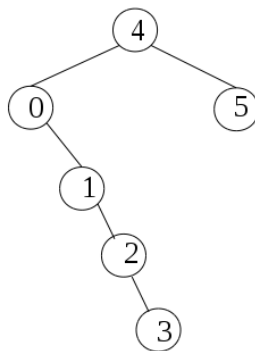
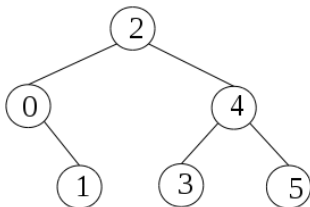
Mulțimea S suferă operații de actualizare în timp (inserare / ștergere).

Arbore binar de căutare:

- ▶ În orice nod v este memorată o valoare dintr-o mulțime total ordonată.
- ▶ Valorile memorate în subarborele din stânga lui v sunt mai mici decât valoarea din v .
- ▶ Valoarea din v este mai mică decât valorile memorate în subarborele din dreapta lui v .

Arbori binari de căutare

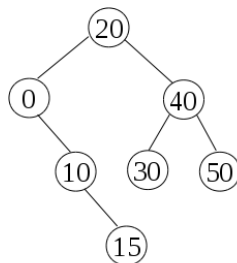
- ▶ Arborele binar de căutare asociat unei mulțimi de chei nu este unic.



Arbori binari de căutare: sortare

- ▶ Parcurgere în inordine

```
Function inordine(v, viziteaza)  
begin  
  if (x == NULL) then  
    return  
  else  
    inordine(v → stg, viziteaza)  
    viziteaza(v)  
    inordine(v → drp, viziteaza)  
end
```



- ▶ Complexitatea timp: $O(n)$

Arbori binari de căutare: căutare

Function $poz(t, x)$

begin

$p \leftarrow t$

while $(p \neq NULL \text{ and } p \rightarrow val \neq x)$ **do**

if $(x < p \rightarrow val)$ **then**

$p \leftarrow p \rightarrow stg$

else

$p \leftarrow p \rightarrow drp$

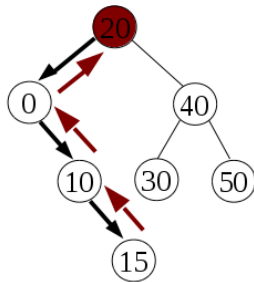
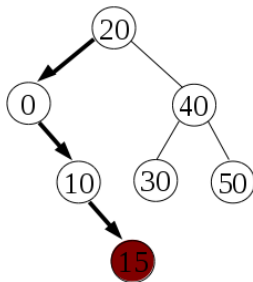
return p

end

- Complexitatea timp: $O(h)$, h înălțime

Predecesor/Succesor

- Modifică operația de căutare: dacă valoarea căutată x nu se găsește în arbore, atunci returnează:
 - fie cea mai mare valoare $< x$ (predecesor),
 - fie cea mai mică valoare $> x$ (succesor).

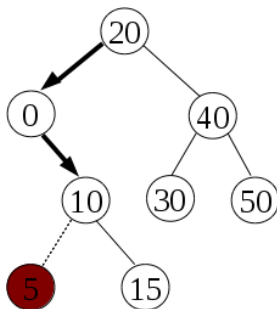


predecesorul lui 18
succesorul lui 18

```
Function successor(t)  
begin  
  if ( $t \rightarrow drp! = NULL$ ) then  
    /*min( $t \rightarrow drp$ )*/  
     $p \leftarrow t \rightarrow drp$   
    while ( $p \rightarrow stg! = NULL$ ) do  
       $p \leftarrow p \rightarrow stg$   
    return  $p$   
  else  
     $p \leftarrow t \rightarrow pred$   
    while ( $p! = NULL$  and  $t == p \rightarrow drp$ ) do  
       $t \leftarrow p$   
       $p \leftarrow p \rightarrow pred$   
    return  $p$   
end
```

Arbori binari de căutare: inserare

- ▶ Se caută în arbore locul în care va fi inserat noul element (similar operației de căutare).
- ▶ Se adaugă nodul cu noua informație, iar subarborii stâng, respectiv drept fiind NULL.



Complexitate timp: $O(h)$, h înălțimea arborelui.

Arbori binari de căutare: inserare

Procedure *insArbBinCautare*(*t*, *x*)

begin

if (*t* == *NULL*) **then**

new(*t*); *t* → *val* ← *x*; *t* → *stg* ← *NULL*; *t* → *drp* ← *NULL*

else

p ← *t*

while (*p*! = *NULL*) **do**

predp ← *p*

if (*x* < *p* → *val*) **then** *p* ← *p* → *stg*;

else

if (*x* > *p* → *val*) **then** *p* ← *p* → *drp*;

else *p* ← *NULL*;

if (*predp* → *val*! = *x*) **then**

if (*x* < *predp* → *val*) **then**

/* adauga x ca fiu stinga al lui *predp* */

else /* adauga x ca fiu dreapta al lui *predp* */ ;

end

Arbori binari de căutare: eliminare

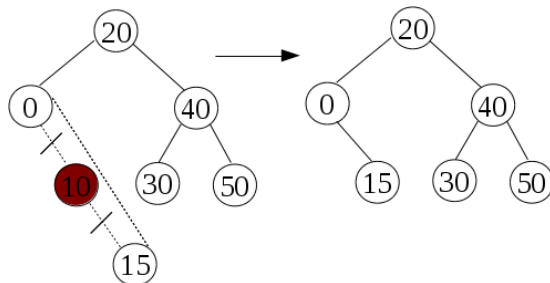
Se caută x în arborele t ; dacă îl găsește atunci se disting cazurile:

- ▶ Cazul 1: nodul p care memorează x nu are fii;
- ▶ Cazul 2: nodul p care memorează x are un singur fiu;
- ▶ cazul 3: nodul p care memorează x are ambii fii.
 - ▶ Determină nodul q care memorează cea mai mare valoare y mai mică decât x (coboară din p la stânga și apoi coboară la dreapta cât se poate).
 - ▶ Interschimbă valorile din p și q .
 - ▶ Șterge q ca în cazul 1 sau 2.

Complexitatea timp: $O(h)$, h înălțime.

Arbori binari de căutare: eliminare

► Cazul 2. Exemplu.



Arbori binari de căutare: eliminare

- cazul 1 sau 2

Procedure *elimCaz1sau2*(*t*, *predp*, *p*)

begin

if (*p* == *t*) **then**

 /* *t* devine vid sau */

 /* unicul fiu al lui *t* devine radacina */

else

if (*p* → *stg* == *NULL*) **then**

 /* inlocuieste in *predp* pe *p* cu *p* → *drp* */

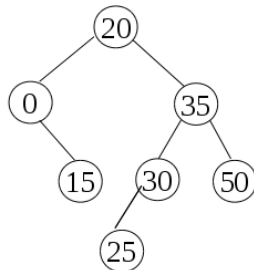
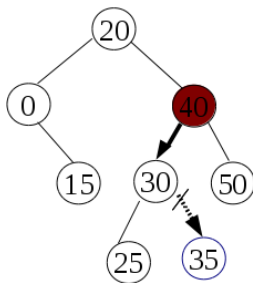
else

 /* inlocuieste in *predp* pe *p* cu *p* → *stg* */

end

Arbori binari de căutare: eliminare

► Cazul 3. Exemplu.



Arbori binari de căutare: eliminare

Procedure *elimArbBinCautare*(*t*, *x*)

begin

if (*t* \neq *NULL*) **then**

p \leftarrow *t*; *predp* \leftarrow *NULL*

while (*p* \neq *NULL* and *p* \rightarrow *val* \neq *x*) **do**

predp \leftarrow *p*

if (*x* $<$ *p* \rightarrow *val*) **then** *p* \leftarrow *p* \rightarrow *stg*;

else *p* \leftarrow *p* \rightarrow *drp*;

if (*p* \neq *NULL*) **then**

if (*p* \rightarrow *stg* $==$ *NULL* or *p* \rightarrow *drp* $==$ *NULL*) **then**
 elimCaz1sau2(*t*, *predp*, *p*)

else

q \leftarrow *p* \rightarrow *stg*; *predq* \leftarrow *p*

while (*q* \rightarrow *drp* \neq *NULL*) **do**

predq \leftarrow *q*; *q* \leftarrow *q* \rightarrow *drp*

p \rightarrow *val* \leftarrow *q* \rightarrow *val*

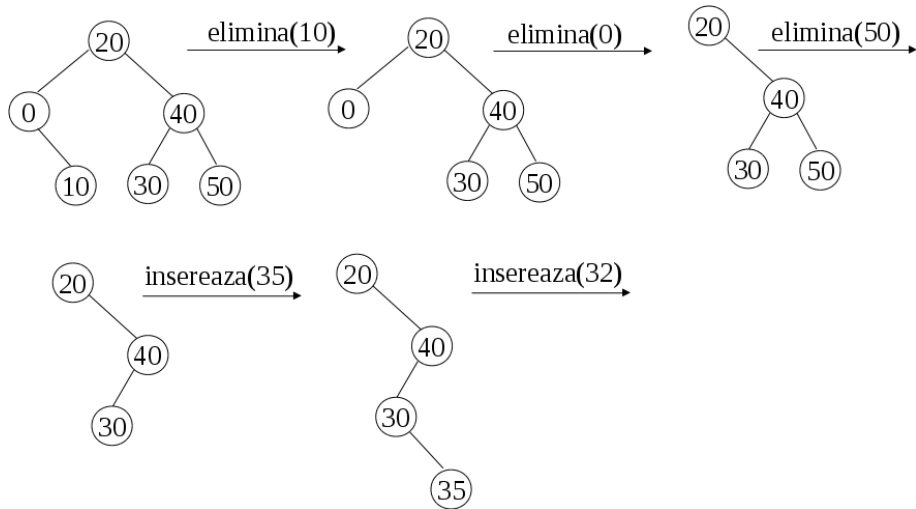
 elimCaz1sau2(*t*, *predq*, *q*)

end

Complexitatea timp

- ▶ Cazul cel mai nefavorabil: $O(n)$, n elemente
- ▶ Cazul mediu: $O(\log n)$

Degenerarea căutării binare în căutare liniară



Problema căutării

Căutare binară

Arbori binari de căutare

Arbori de căutare echilibrați

Arbori de căutare echilibrați

- ▶ Arbori AVL (Adelson-Velsii and Landis, 1962)
- ▶ Arbori B/2-3-4 arbori (Bayer and McCreight, 1972)
- ▶ arbori roșu-negru (Bayer, 1972)
- ▶ Arbori Splay (Sleator and Tarjan, 1985)
- ▶ Treaps (Seidel and Aragon, 1996)

Arbori de căutare echilibrați

- ▶ \mathcal{C} este clasă de arbori echilibrați dacă pentru orice arbore t cu n vârfuri din \mathcal{C} : $h(t) \leq c \log n$, c constantă.
- ▶ \mathcal{C} este clasă de arbori echilibrați $O(\log n)$ -stabilă dacă există algoritmi pentru operațiile de căutare, inserare, ștergere în $O(\log n)$, iar arborii rezultați fac parte din clasa \mathcal{C} .

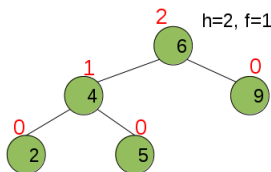
Arbori AVL

(G. **A**delson-**V**elskii, E.M. **L**andis 1962)

- ▶ Un arbore binar de căutare t este un arbore **AVL-echilibrat** dacă pentru orice vârf v ,

$$|h(v \rightarrow stg) - h(v \rightarrow drp)| \leq 1$$

- ▶ $h(v \rightarrow stg) - h(v \rightarrow drp)$ se numește **factor de echilibrare**.
- ▶ Exemplu:



▶ Lema

Dacă t este AVL-echilibrat cu n noduri interne atunci $h(t) = \Theta(\log n)$.

Demonstratie. La curs.

► Teoremă

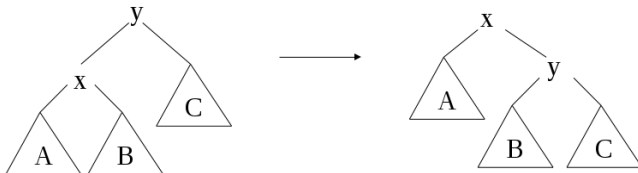
Clasa arborilor AVL-echilibrați este $O(\log n)$ stabilă.

► Algoritmul de inserare/ștergere

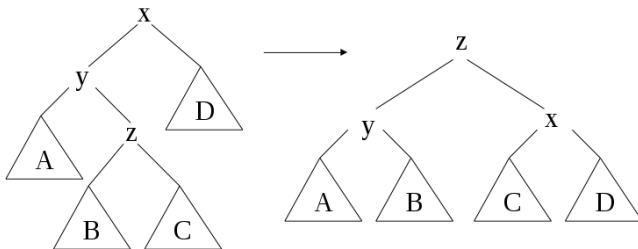
- Nodurile au memorate și factorii de echilibrare $(-1, 0, 1)$.
- Se memorează drumul de la rădăcină la nodul adăugat/șters într-o stivă ($O(\log n)$).
- Se parcurge drumul memorat în stivă în sens invers și se reechilibrează nodurile dezechilibrate cu una dintre operațiile: rotație stânga/dreapta simplă/dublă ($O(\log n)$).

Rotații

Rotație dreapta simplă

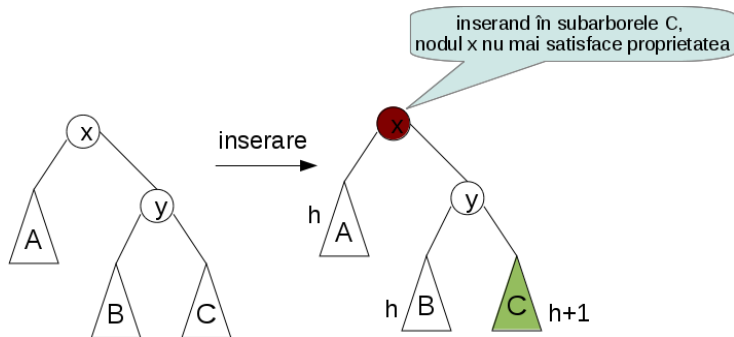


Rotație dreapta dublă

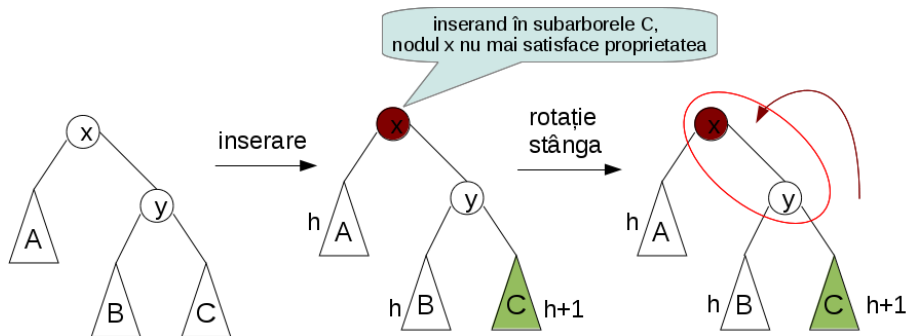


Similar pentru rotație stânga simplă, respectiv rotație stânga dublă.

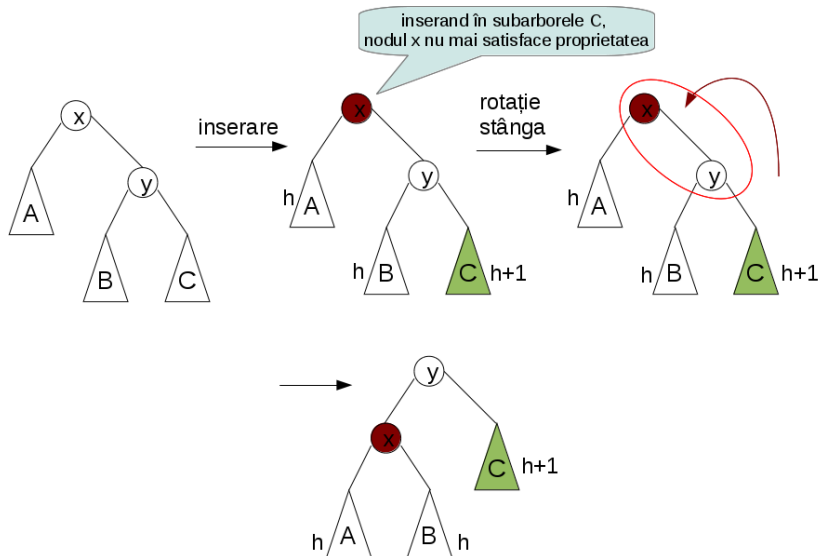
Rotație stânga simplă



Rotație stânga simplă (cont.)



Rotație stânga simplă (cont.)



Rotație stânga simplă

Procedure *rotatieStanga*(x)
begin

$y \leftarrow x \rightarrow drp$

$x \rightarrow drp \leftarrow y \rightarrow stg$

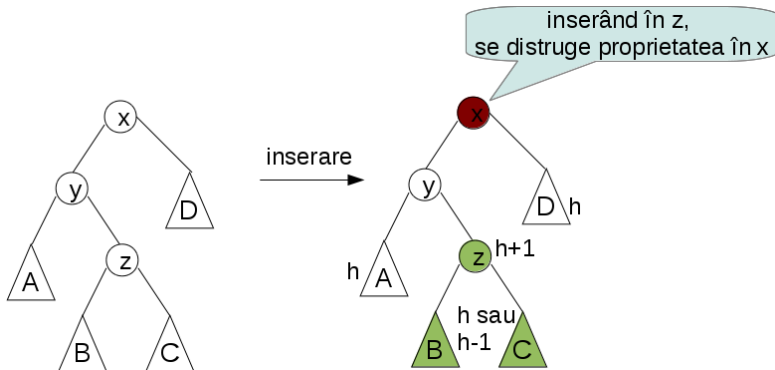
$y \rightarrow stg \leftarrow x$

return y

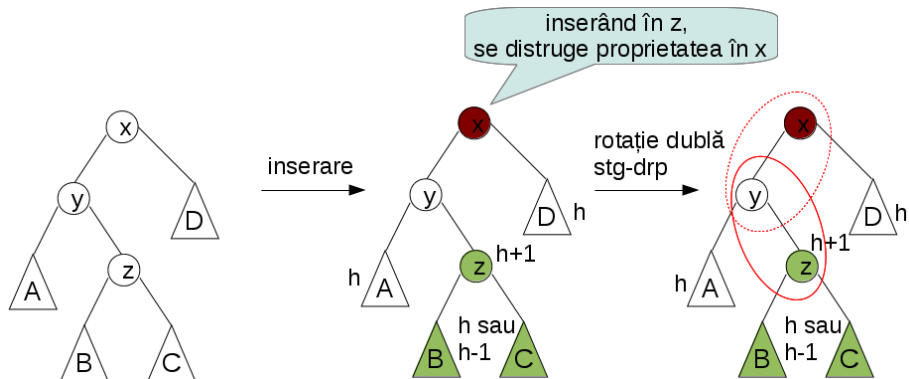
end

- Complexitatea timp: $O(1)$

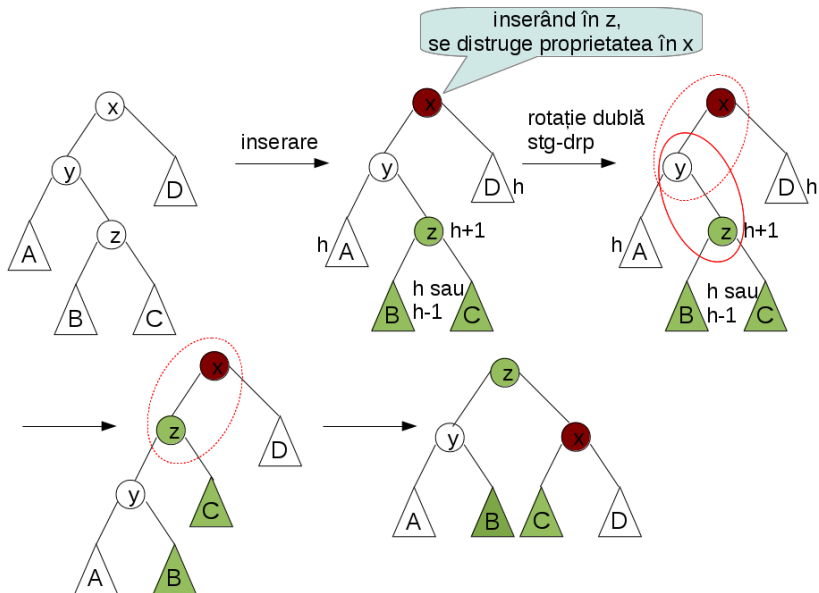
Rotație dublă



Rotație dublă (cont.)



Rotație dublă (cont.)



Inserare: algoritm

Procedure *echilibrare*(t, x)

begin

while ($x \neq \text{NULL}$) **do**

 /* actualizeaza inaltimea $h(x)$ */

if ($h(x \rightarrow \text{stg}) \geq 2 + h(x \rightarrow \text{drp})$) **then**

if ($h(x \rightarrow \text{stg} \rightarrow \text{stg}) \geq h(x \rightarrow \text{stg} \rightarrow \text{drp})$) **then**
 rotatieDreapta(t, x)

else

rotatieStanga($t, x \rightarrow \text{stg}$); *rotatieDreapta*(t, x)

else

if ($h(x \rightarrow \text{drp}) \geq 2 + h(x \rightarrow \text{stg})$) **then**

if ($h(x \rightarrow \text{drp} \rightarrow \text{drp}) \geq h(x \rightarrow \text{drp} \rightarrow \text{stg})$) **then**
 rotatieStanga(t, x)

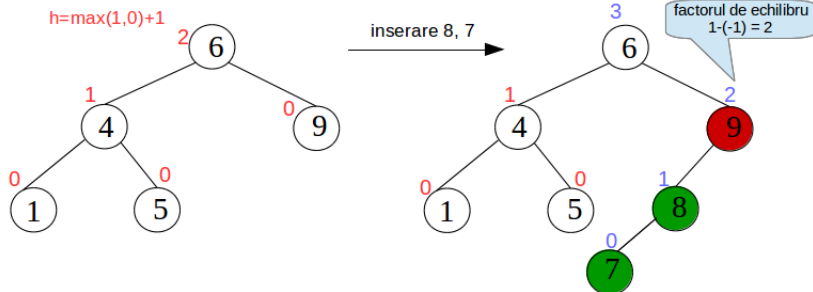
else

rotatieDreapta($t, x \rightarrow \text{drp}$); *rotatieStanga*(t, x)

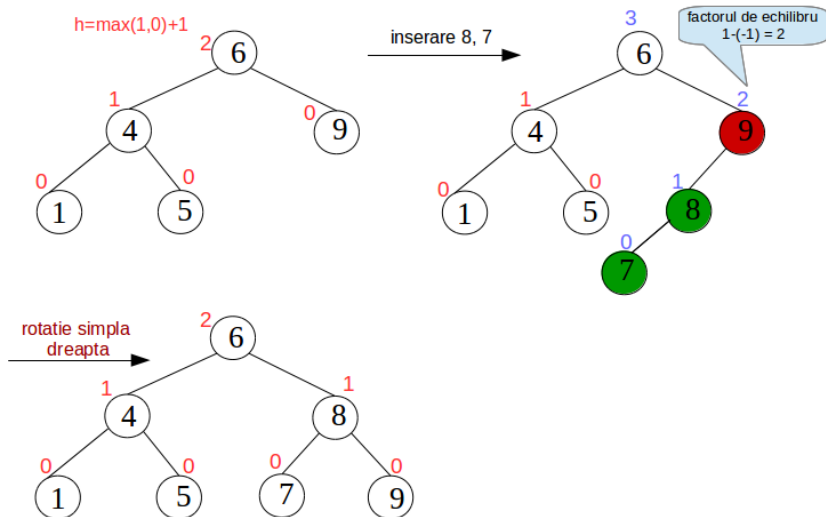
$x \leftarrow \text{pred}[x]$

end

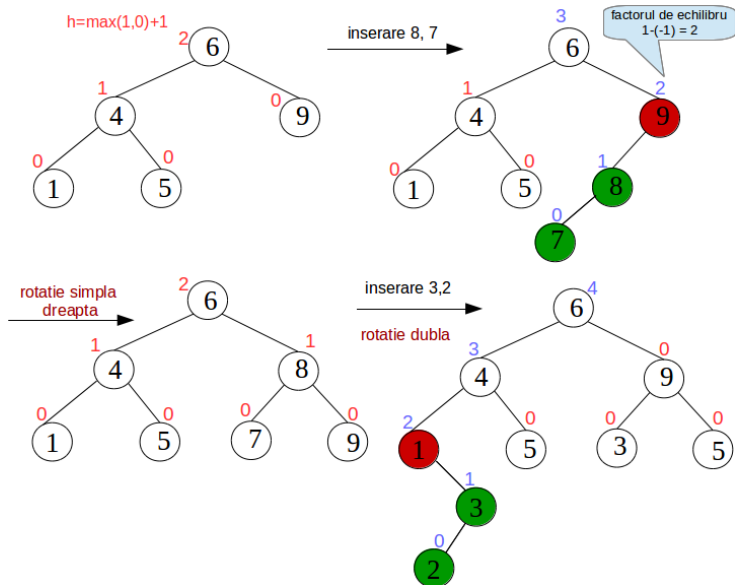
Exemplu: inserare



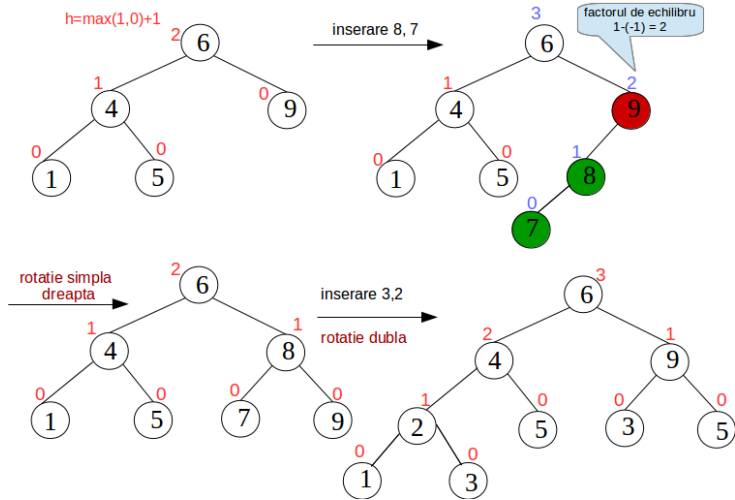
Exemplu: inserare (cont.)



Exemplu: inserare (cont.)



Exemplu: inserare(cont.)



Avantaje/dezavantaje ale arborilor AVL

- ▶ Avantaje:
 - ▶ Căutarea, inserarea și ștergerea se realizează cu complexitatea $O(\log n)$.
- ▶ Dezavantaje:
 - ▶ Spațiu suplimentar pentru memorarea înălțimii / factorului de echilibrare.
 - ▶ Operațiile de re-echilibrare sunt costisitoare.
- ▶ Sunt preferați când avem de realizat mai multe căutări și mai puține inserări și ștergeri
- ▶ Aplicații în *Data Analysis*, *Data Mining*