Căutare.

SD 2015/2016

Conținut

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	$O(n)$	$O(n)$	$O(1)$

Conținut

Căutare binară: aspect static

- Mulțimea univers este total ordonată: $(U, \leq)$

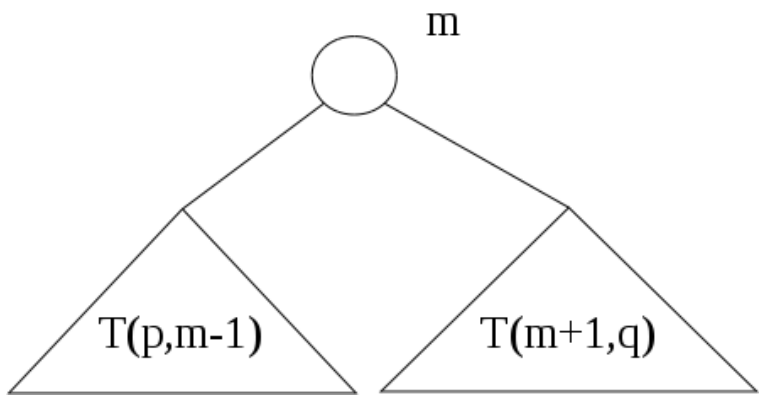
- Structura de date utilizată:
  - tabloul $s[0..n-1]$
  - $s[0] < ... < s[n-1]$

Căutare binară: aspect static

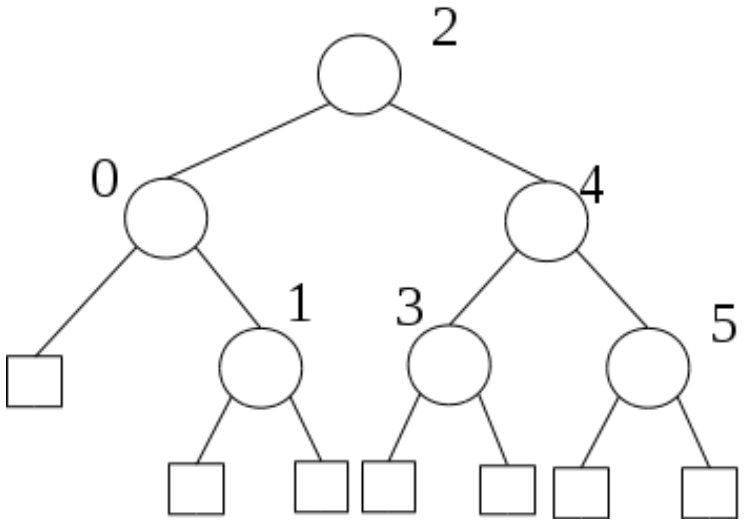
Function $poz(s[0..n-1], n, a)$
begin
    $p \leftarrow 0; q \leftarrow n-1$
    $m \leftarrow (p+q)/2$
    while $(s[m] != a \text{ and } p < q)$ **do**
        if $(a < s[m])$ **then**
            $q \leftarrow m-1$
        else
            $p \leftarrow m+1$
        $m \leftarrow (p+q)/2$
    if $(s[m]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

Arborele binar asociat căutarii binare

$T(p, q)$



$T = T(0, n-1)$
$n = 6$



Conținut

Căutare binară: aspect dinamic

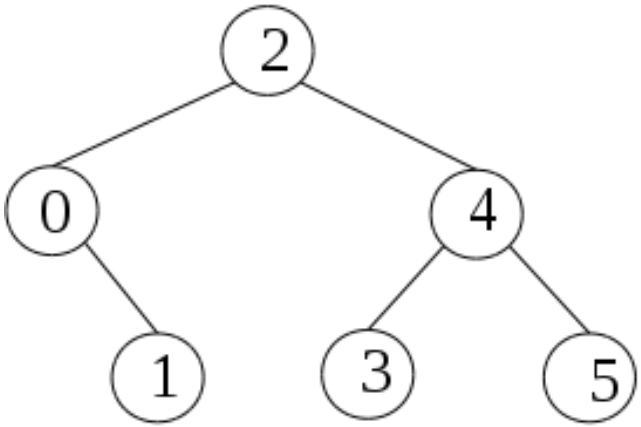
Mulțimea S suferă operații de actualizare în timp (inserare / ștergere).

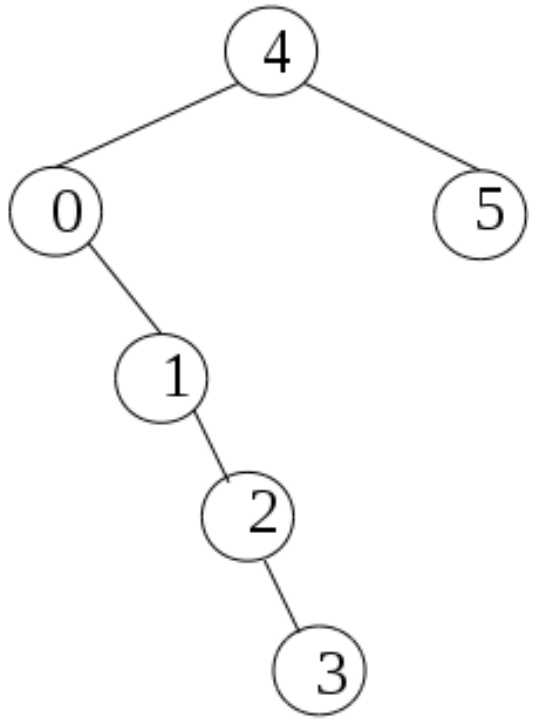
Arbore binar de căutare:

- 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.

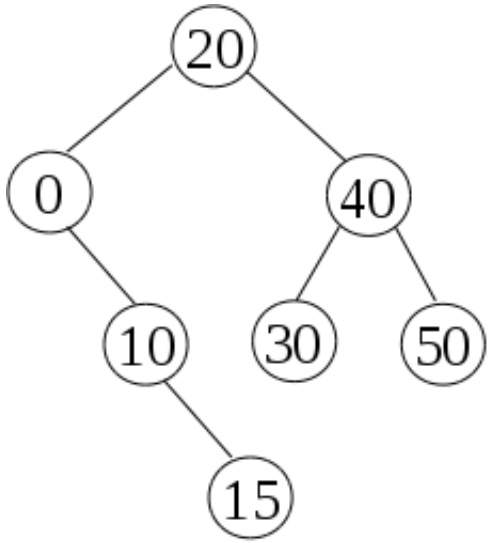




Arbori binari de căutare: sortare

- Parcurgere în inordine

Function *inordine(v, viziteaza)*
begin
  if *(x == NULL)* **then**
    return
  else
    inordine($v \rightarrow stg$, viziteaza)
    viziteaza(v)
    inordine($v \rightarrow drp$, viziteaza)
end



- Complexitatea timp: $O(n)$

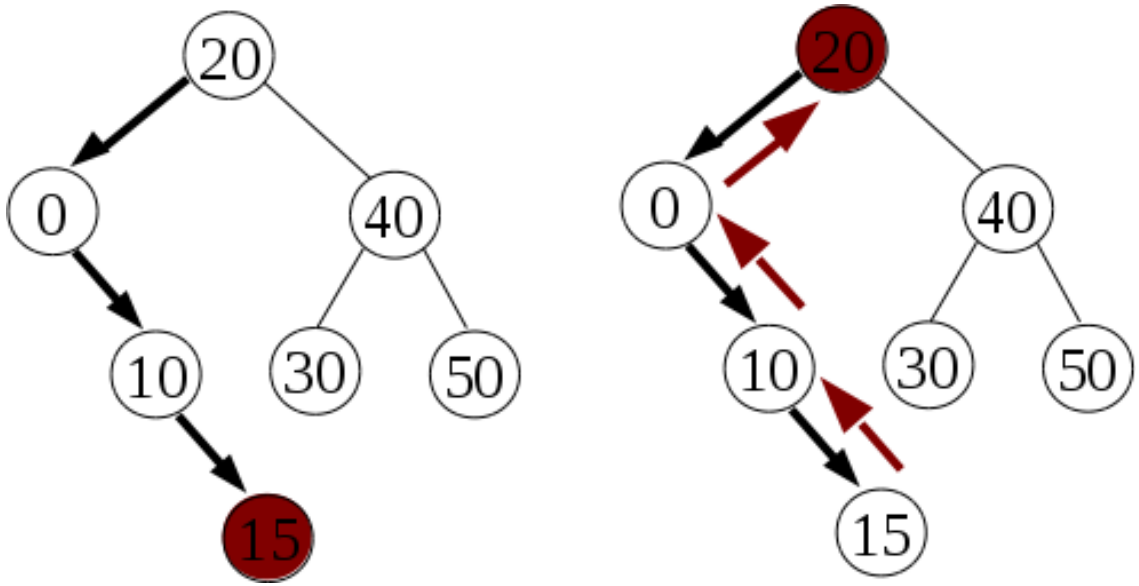
Arbori binari de căutare: căutare

Function $poz(t, x)$
begin
    $p \leftarrow t$
    while $(p! = NULL \text{ and } p \rightarrow val! = x)$ **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).



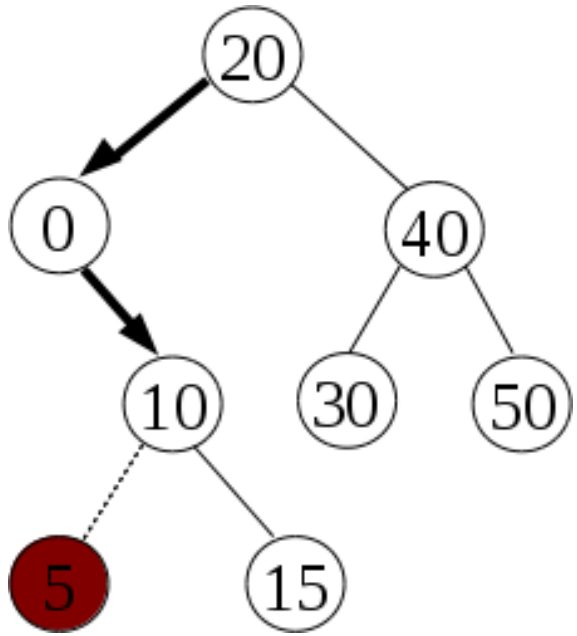
predecesorul lui 18
succesorul lui 18

Succesor

```
Function succesor(t)
begin
    if (t → drp! = NULL) then
        /*min(t → drp)*/
        p ← t → drp
        while (p → stg! = NULL) do
            p ← p → stg
        return p
    else
        p ← pred[t]
        while (p! = NULL and t == p → drp) do
            t ← p
            p ← pred[p]
        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 subarborele 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

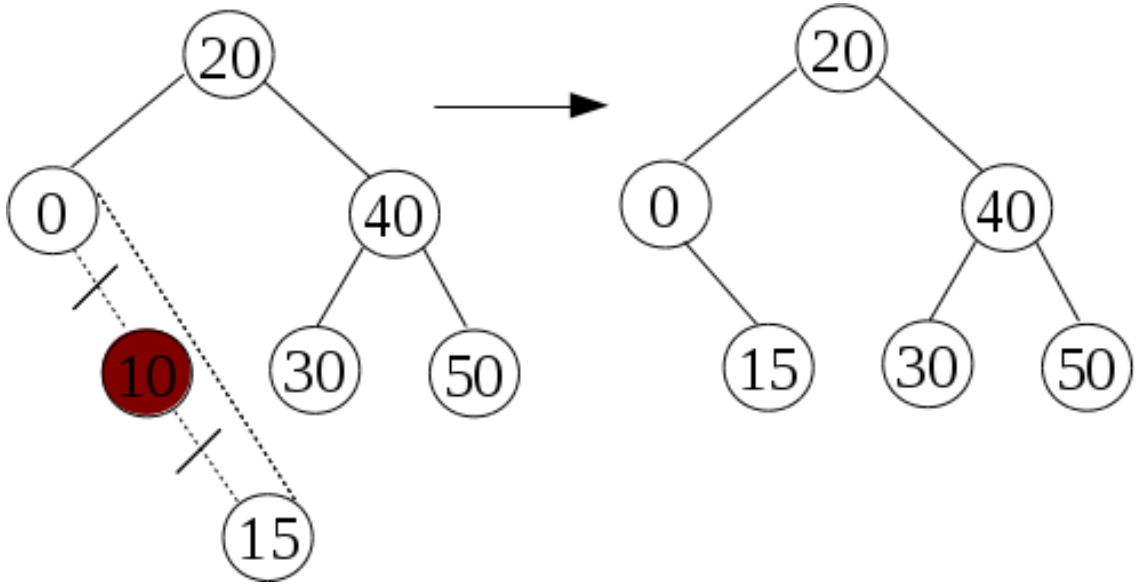
Se caută x în arborele t; dacă îl găsește atunci se disting cazurile:

- 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.



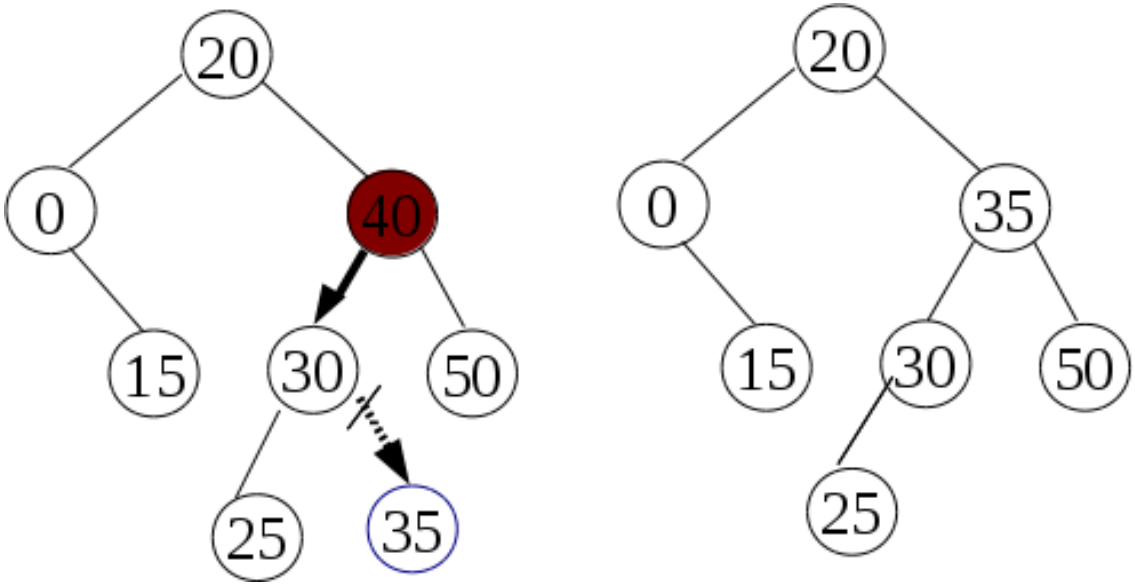
Arbori binari de căutare: eliminare

- 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

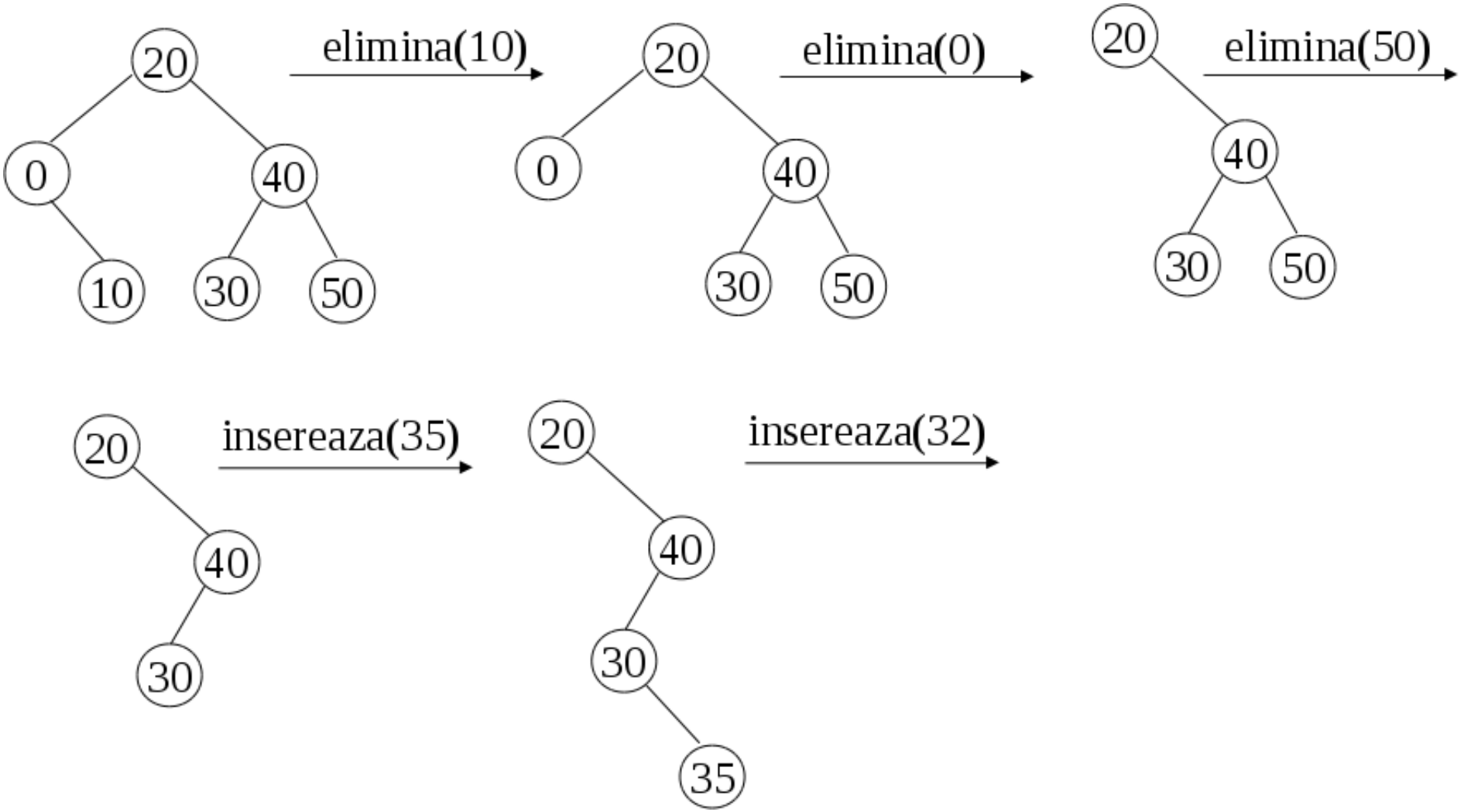
```
Procedure elimArbBinCautare(t, x)
begin
    if (t! = NULL) then
        p ← t; predp ← NULL
        while (p! = NULL and p → val! = x) do
            predp ← p
            if (x < p → val) then p ← p → stg;
            else p ← p → drp;
        if (p! = NULL) then
            if (p → stg == NULL or p → drp == NULL) then
                elimCaz1sau2(t, predp, p)
            else
                q ← p → stg; predq ← p
                while (q → drp! = NULL) do
                    predq ← q; q ← q → drp
                p → val ← q → val
                elimCaz1sau2(t, predq, q)
end
```

Arbori binari de căutare: analiză

Complexitatea timp

- Cazul cel mai nefavorabil: $O(n)$, n elemente
- Cazul mediu: $O(logn)$

Degenerarea căutării binare în căutare liniară



Conținut

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.

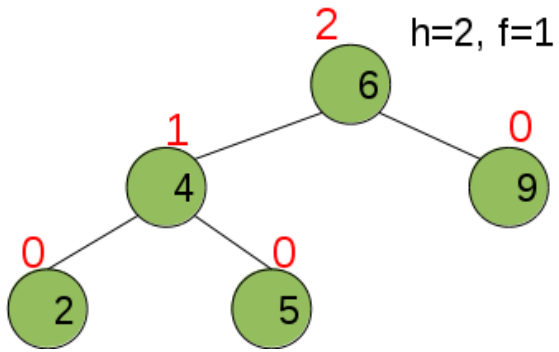
- $\mathcal{C}$ este clasă de arbori echilibrați $O(\log n)$-stabilă dacă
  există algoritmi pentru operațiile de căutare, inserare, ștergere în $O(\log n)$, iar arborii rezultați fac parte din clasa $\mathcal{C}$.

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,

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

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



- **Lema**

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

Arbori AVL

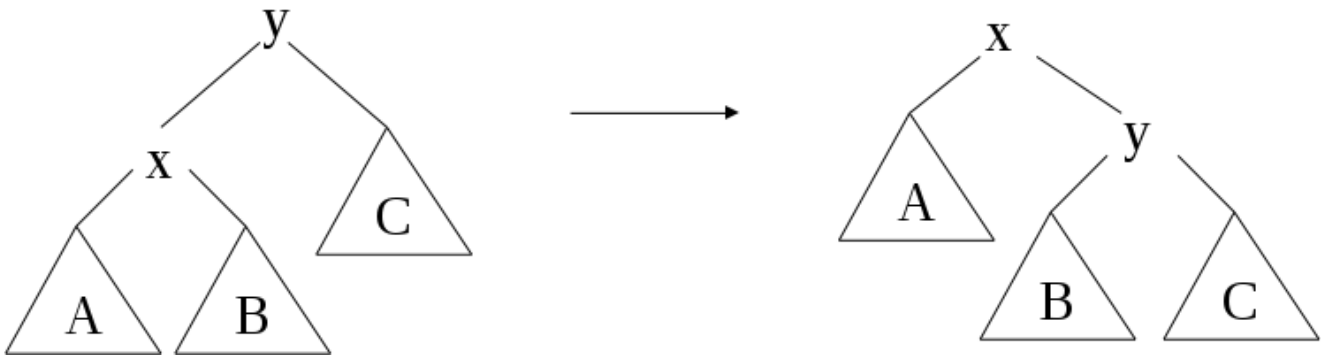
- **Teoremă**

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

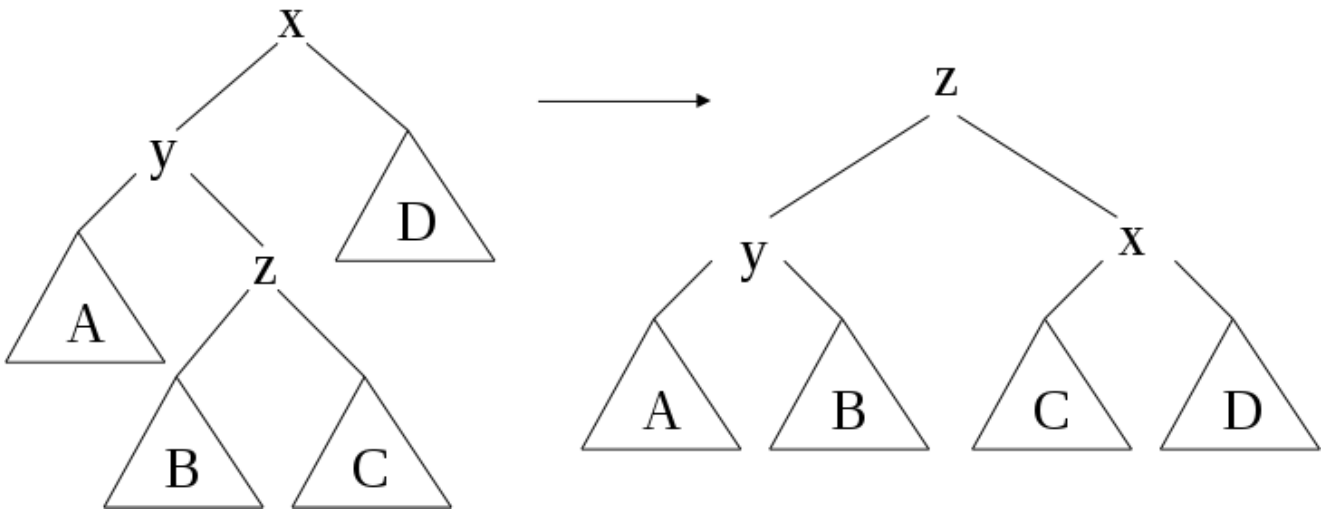
- Algoritmul de inserare/ștergere
  - Nodurile au memorate și factorii de echilibrare $(-1, 0, 1)$.
  - Se memorează drumul de la rădăcină la nodul adăugat/șters într-o stivă ($O(\log n)$).
  - Se parcurge drumul memorat în stivă în sens invers și se reechilibează nodurile dezechilibrate cu una dintre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

Rotații

Rotație dreapta simplă

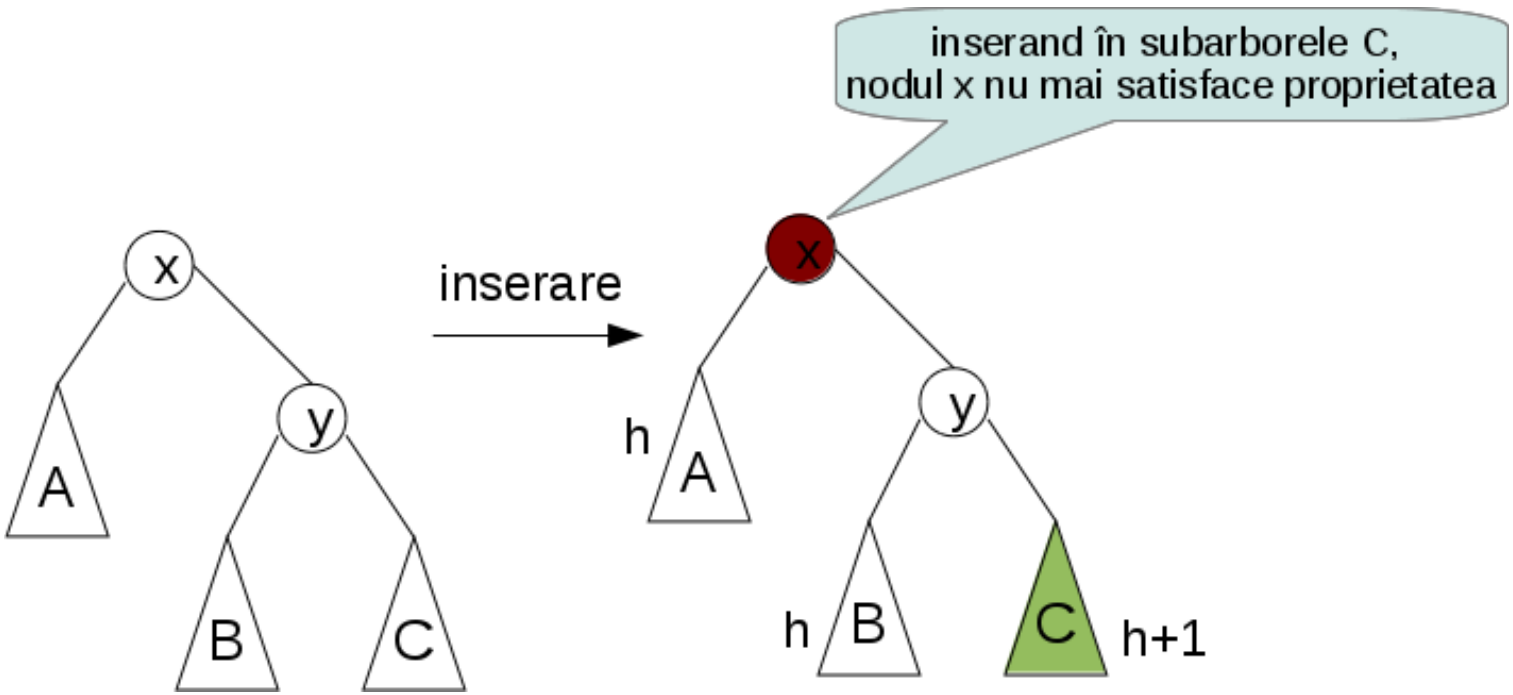


Rotație dreapta dublă

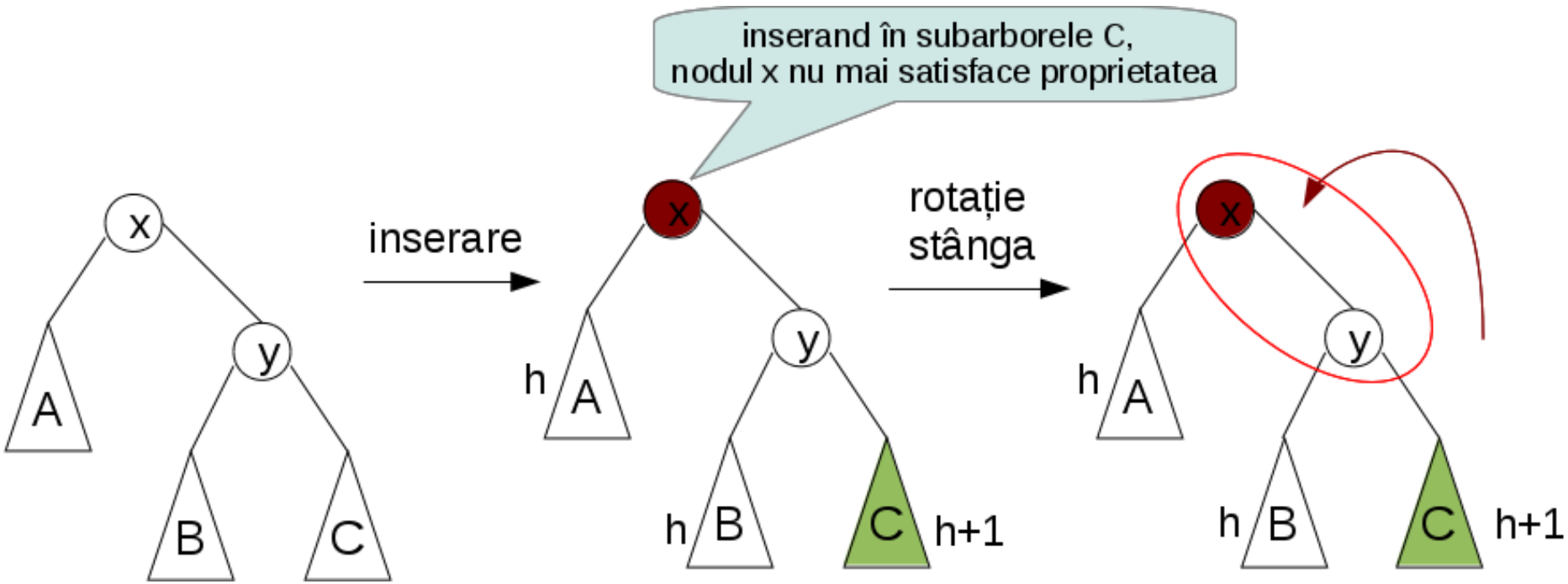


Similar pentru rotație stânga simplă, respectiv rotație stânga dublă.

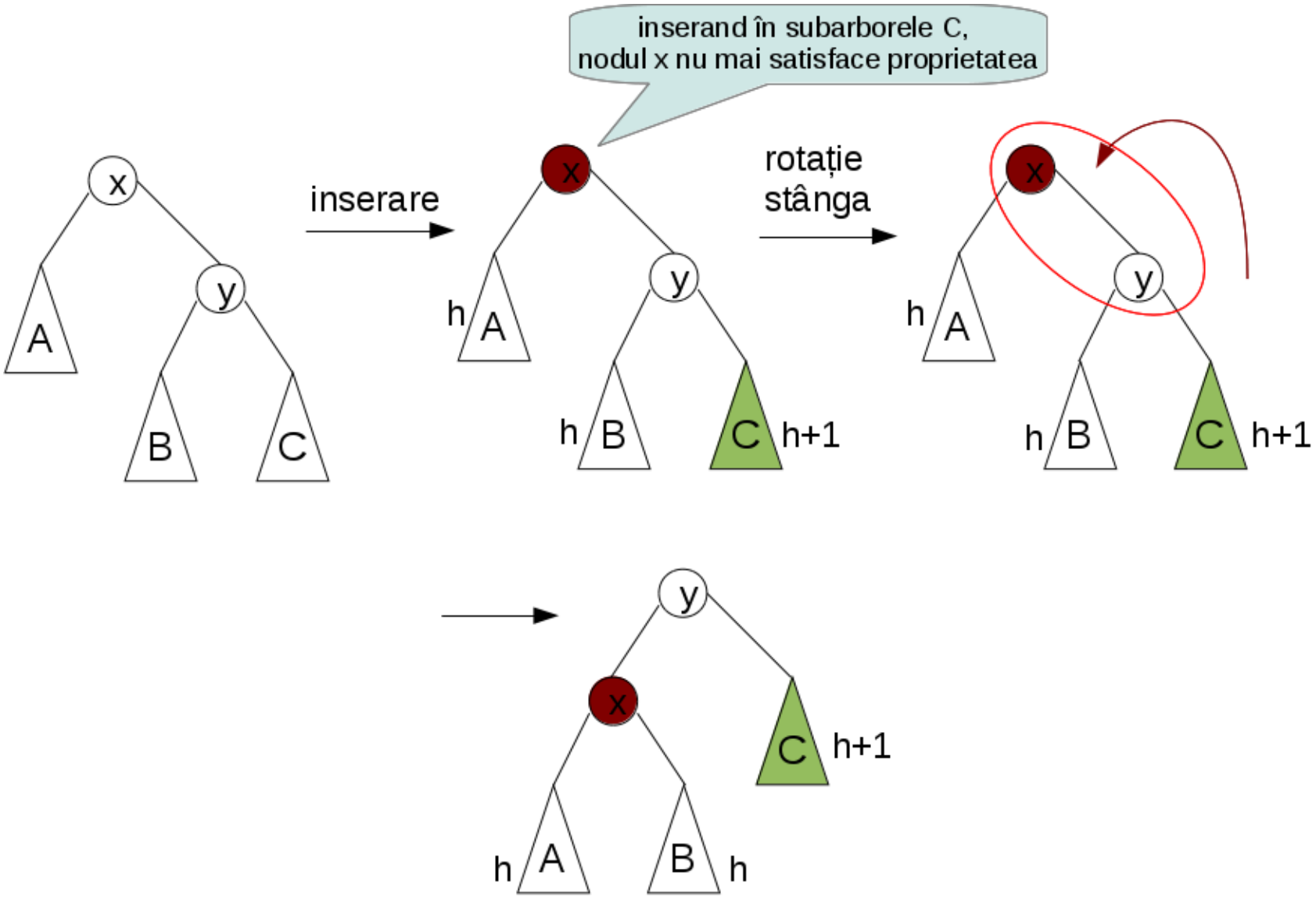
Rotație stânga simplă



Rotație stânga simplă (cont.)



Rotație stânga simplă (cont.)



Rotație stânga simplă

Procedure *rotatieStanga(x)*
begin

$y \leftarrow x \rightarrow drp$

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

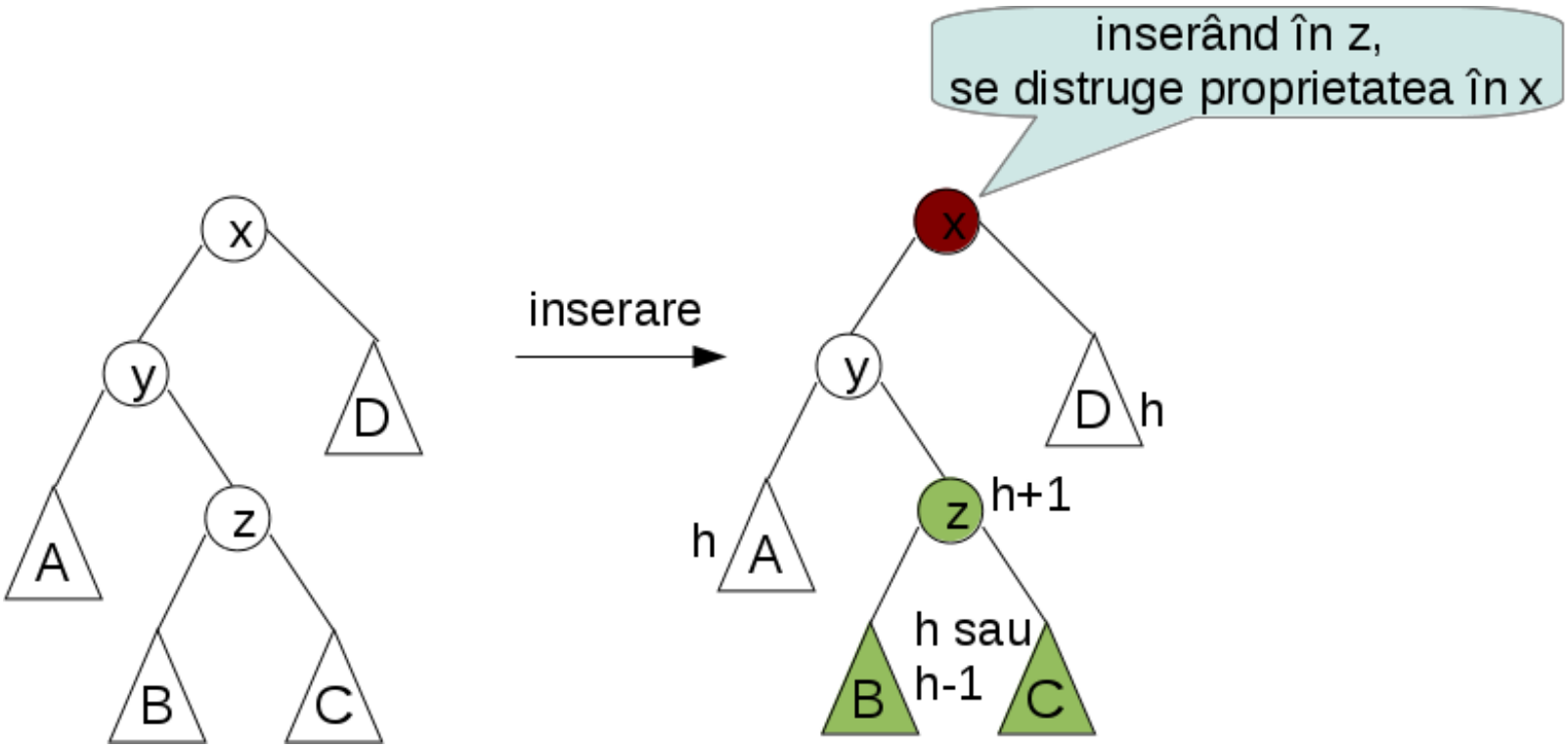
$y \rightarrow stg \leftarrow x$

return y

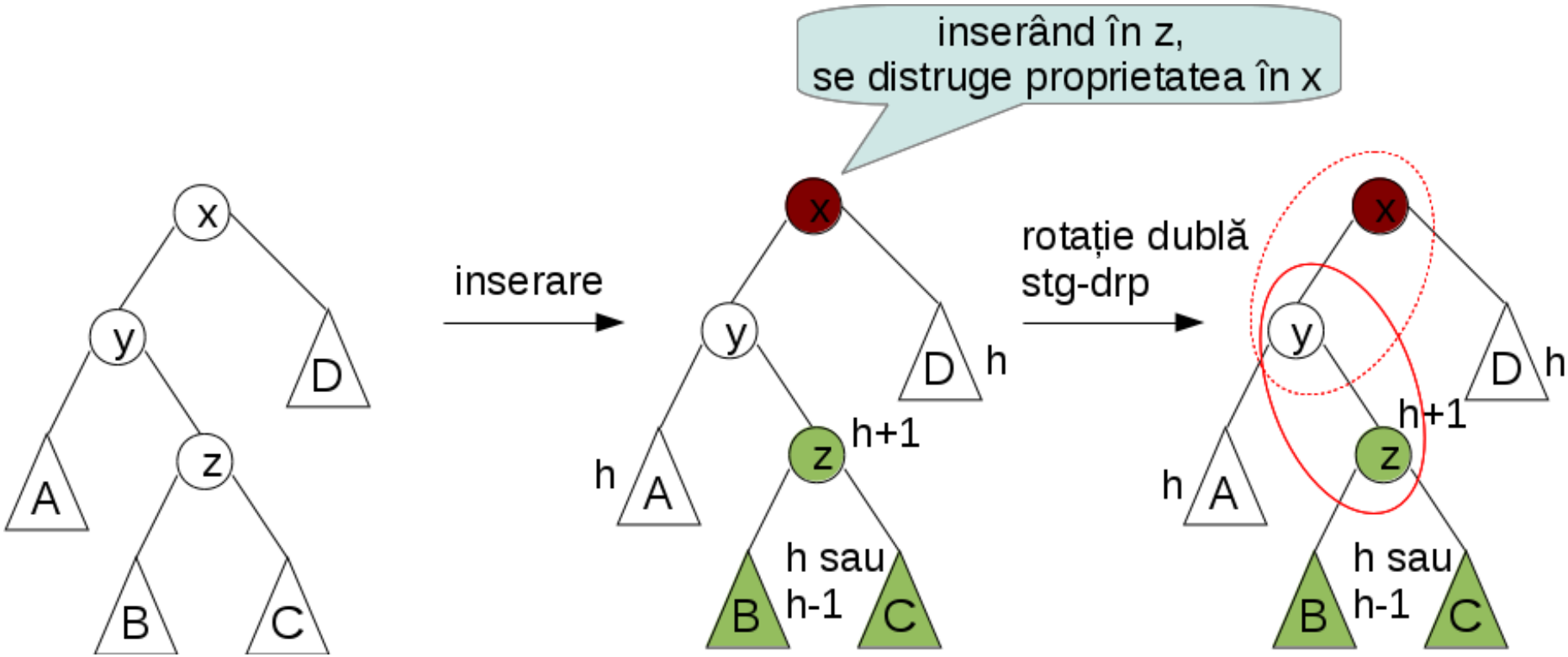
end

- Complexitatea timp: $O(1)$

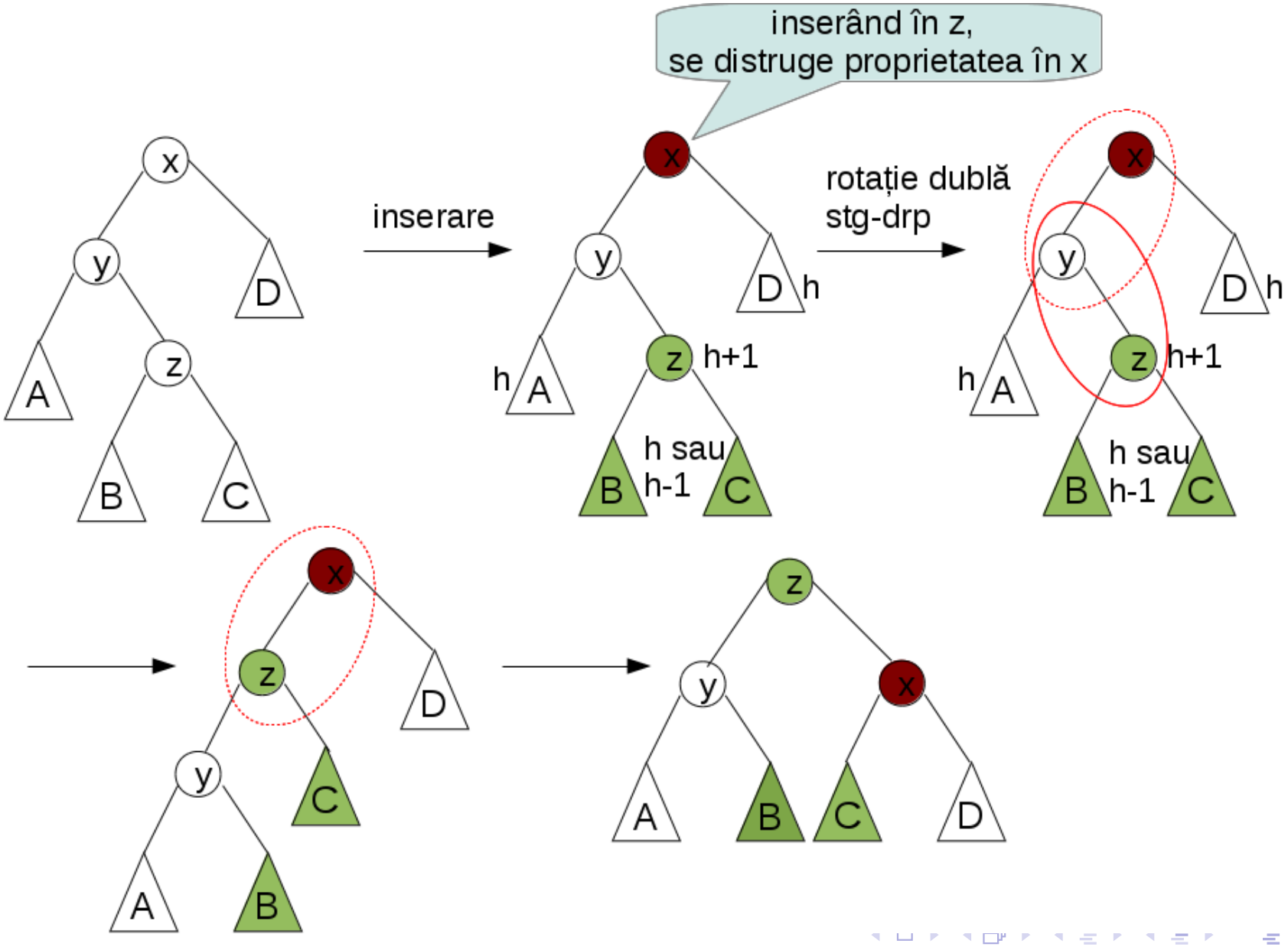
Rotație dublă



Rotație dublă (cont.)



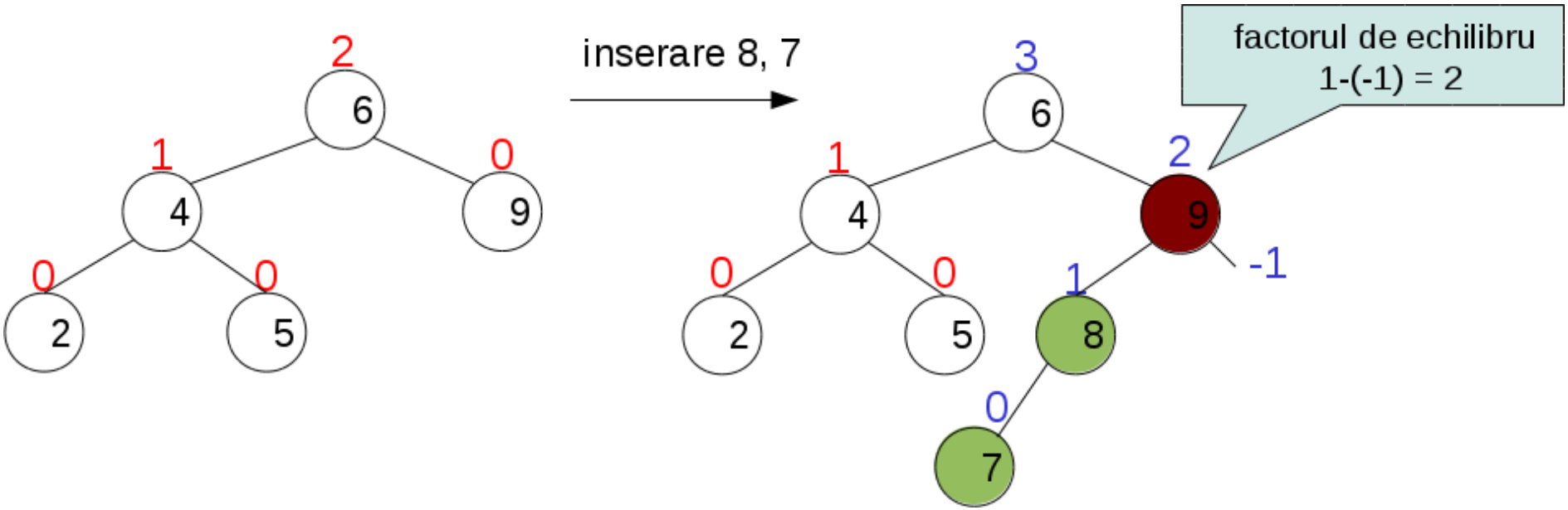
Rotație dublă (cont.)



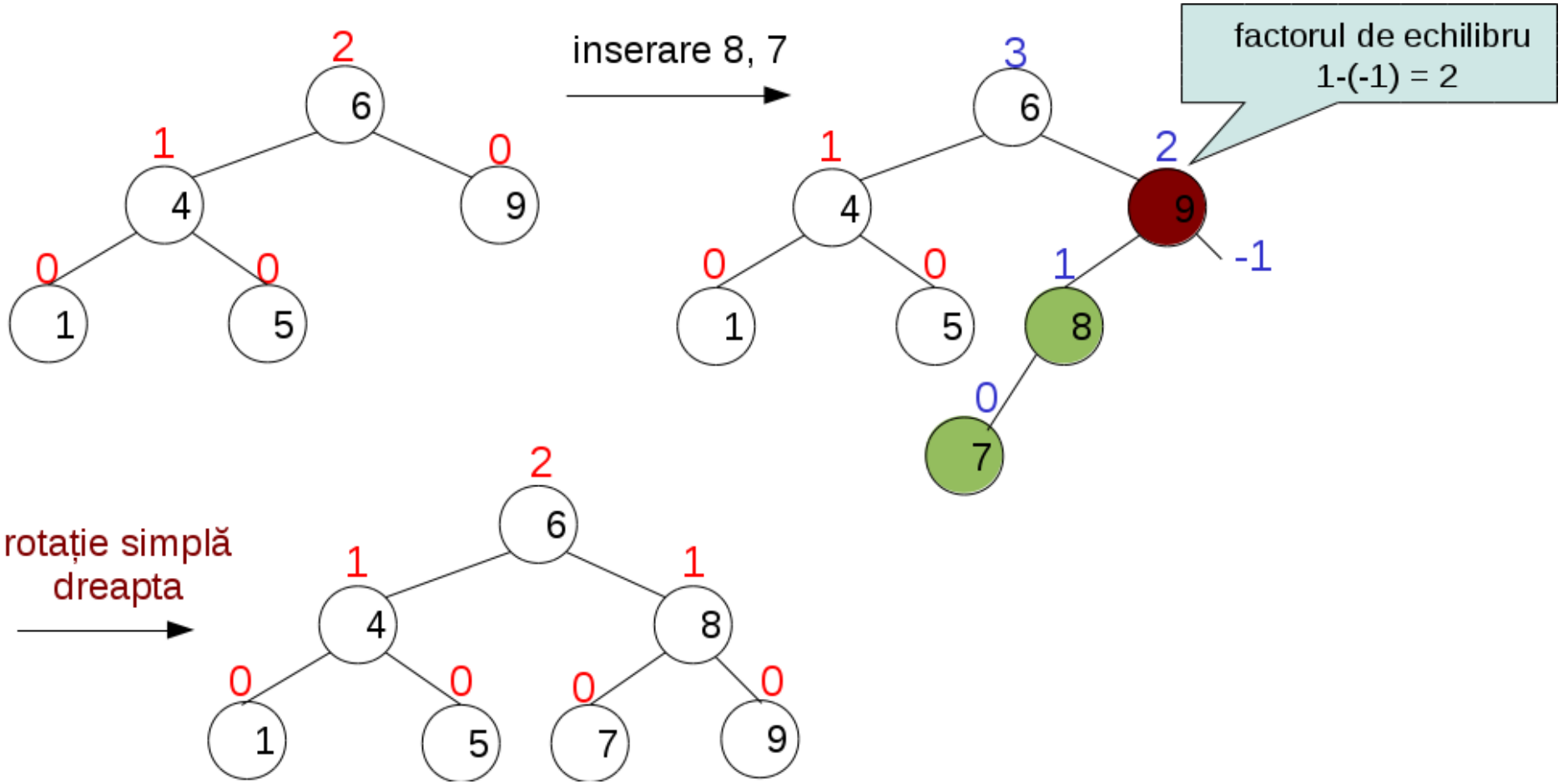
Inserare: algoritm

Procedure *echilibrare*(t, x)
begin
    while $(x! = NULL)$ **do**
        /* actualizeaza inaltimea $h(x)$ */
        if $(h(x \to stg)) \geq 2 + h(x \to drp))$ **then**
            if $(h(x \to stg \to stg)) \geq h(x \to stg \to drp))$ **then**
                $rotatieDreapta(t, x)$
            else
                $rotatieStanga(t, x \to stg)$; $rotatieDreapta(t, x)$
        else
            if $(h(x \to drp)) \geq 2 + h(x \to stg))$ **then**
                if $(h(x \to drp \to drp)) \geq h(x \to drp \to stg))$ **then**
                    $rotatieStanga(t, x)$
                else
                    $rotatieDreapta(t, x \to drp)$; $rotatieStanga(t, x)$
        $x \leftarrow pred[x]$
end

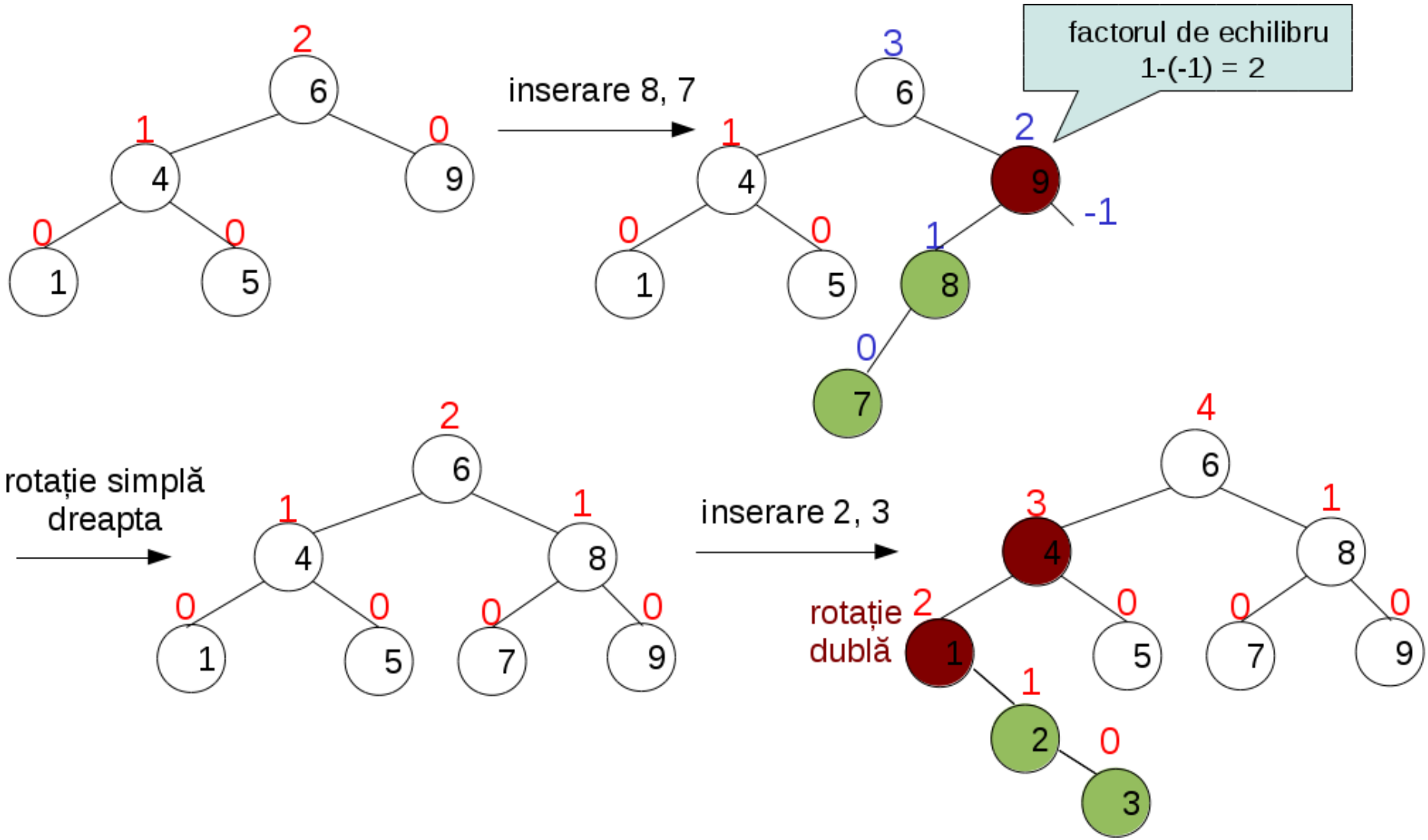
Exemplu: inserare



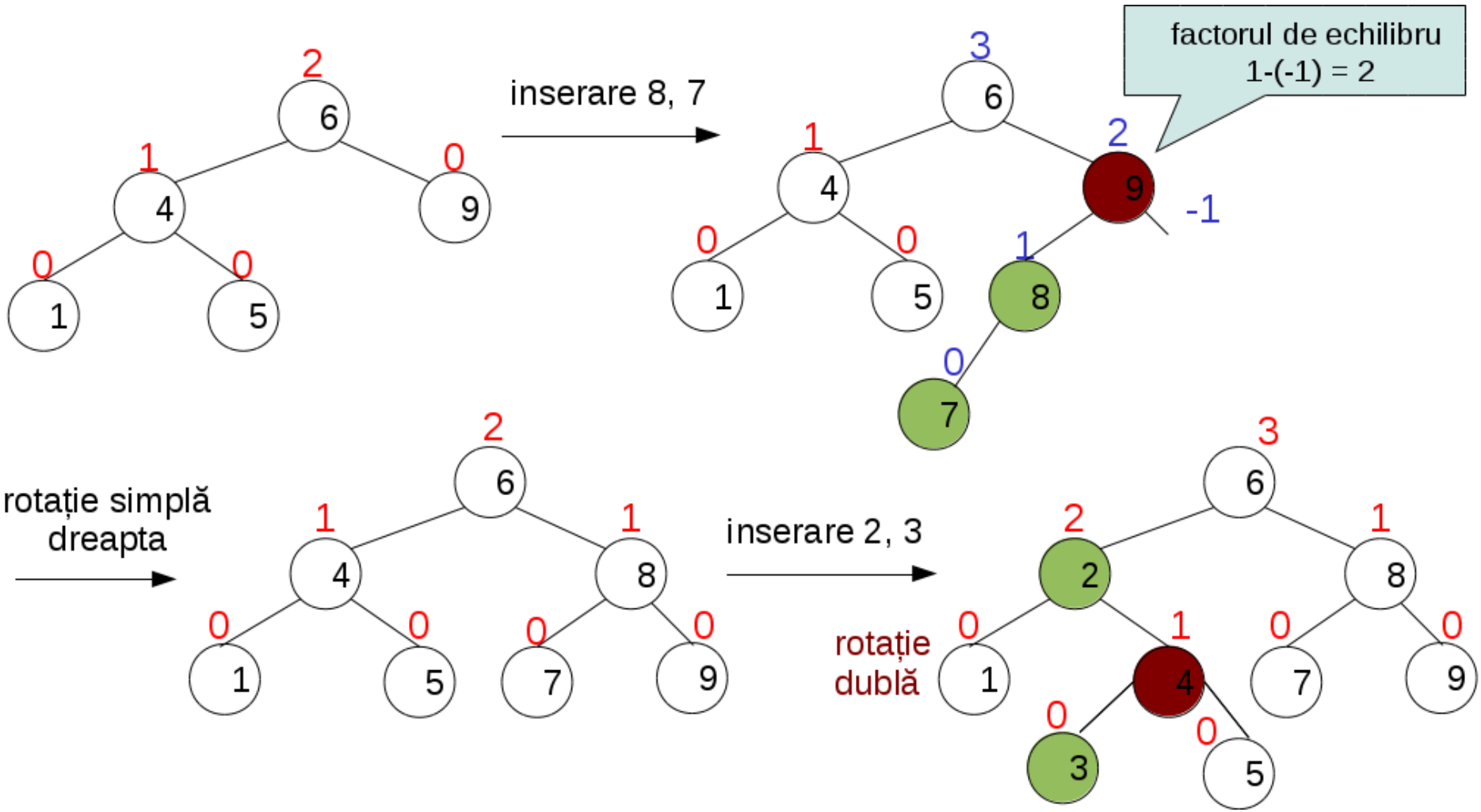
Exemplu: inserare (cont.)



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 cand facem mai multe căutări și mai puține inserări și ștergeri
- *Data Analysis*, *Data Mining*