Structuri de Date și Algoritmi
Arbori de căutare echilibrați

Mihai Nan

Departamentul de Calculatoare
Facultatea de Automatică și Calculatoare
Universitatea POLITEHNICA din București

Anul Universitar 2022–2023

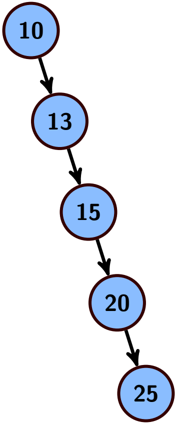
Conținutul cursului

Motivație

Cum o să arate arborele binar de căutare rezultat în urma inserării valorilor: $10, 13, 15, 20, 25$?

Motivație

Cum o să arate arborele binar de căutare rezultat în urma inserării valorilor: 10, 13, 15, 20, 25?



Motivație

Ce se întâmplă cu performanțele obținute de un astfel de arbore pentru operațiile de căutare / ștergere?

Motivație

Ce se întâmplă cu performanțele obținute de un astfel de arbore pentru operațiile de căutare / ștergere?

Am văzut că performanțele pentru cele două operații, în cazul unui arbore cu înălțimea H, sunt:

- **căutare** – $O(H)$
- **inserarea** – $O(H)$

Motivație

Cât o să fie înălțimea arborelui în acest caz?

Motivație

Cât o să fie înălțimea arborelui în acest caz?

În acest caz, înălțimea arborelui este egală cu numărul de noduri din arbore minus 1 ($H = N - 1 \Rightarrow O(H) = O(N)$).

Motivație

Cât o să fie înălțimea arborelui în acest caz?

În acest caz, înălțimea arborelui este egală cu numărul de noduri din arbore minus 1 ($H = N - 1 \Rightarrow O(H) = O(N)$).

Cum putem elimina acest inconvenient?

Motivație

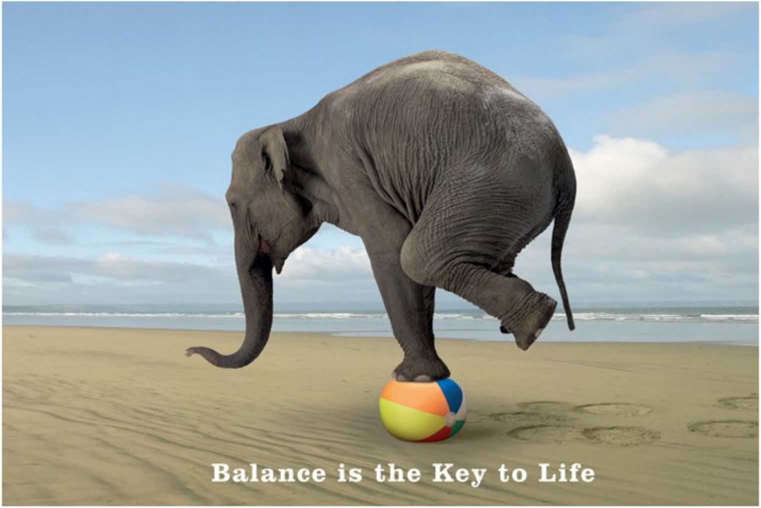
Cât o să fie înălțimea arborelui în acest caz?

În acest caz, înălțimea arborelui este egală cu numărul de noduri din arbore minus 1 ($H = N - 1 \Rightarrow O(H) = O(N)$).

Cum putem elimina acest inconvenient?

Putem să ne asigurăm că în permanență avem un arbore binar de căutare echilibrat.

Motivație



Arbori binari de căutare echilibrați în înălțime

- **Înălțimea unui arbore cu rădăcina N**
  $h(N) =$ numărul de legături de-a lungul celei mai lungi căi a unui arbore de la rădăcina N la o frunză
- **Factorul de echilibru** (balanced factor BF)

$$BF(N) = h(Rt(N)) - h(Lt(N))$$

Spunem că un arbore binar cu rădăcina N este echilibrat în înălțime dacă:

- $Rt(N)$ și $Lt(N)$ sunt arbori binar echilibrați
- $BF(N) \in \{-1, 0, +1\}$

Definiția se poate extinde și la arbori multicăi

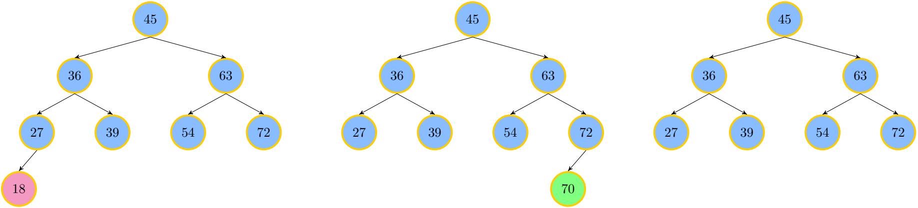
Arborii AVL sunt arbori echilibrați în înălțime!

Arbori binari de căutare echilibrați în înălțime

Important

Într-un arbore AVL, BF este 1, 0 sau -1 pentru orice nod!

- $BF(N) < 0 \Rightarrow$ **left-heavy**
- $BF(N) > 0 \Rightarrow$ **right-heavy**
- $BF(N) = 0 \Rightarrow$ **balanced**

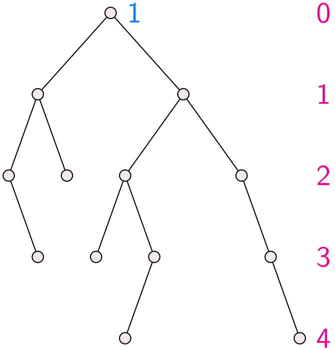


- Height balaced trees (AVL, Red-Black)
- Weight balanced trees

Arbori AVL

Arbori AVL

- În cazul nodurilor frunză, factorul de echilibru este 0.



Important

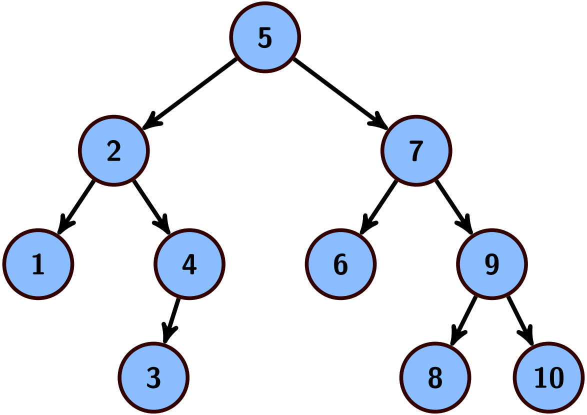
Într-un arbore AVL, factorul de echilibru al fiecărui nod din arbore poate lua una dintre următoarele valori: -1, 0, 1.

Arbori AVL – Proprietăți

- Spunem că $\mathscr{C}$ este clasa arborilor echilibrați dacă pentru orice arbore din $\mathscr{C}$, având rădăcina `root` și n vârfuri, este îndeplinită următoarea condiție $height(root) \leq c \cdot \log n$, unde c este o constantă.
- $\mathscr{C}$ reprezintă clasa arborilor echilibrați și este $O(\log n)$-stabilă dacă există algoritmi pentru operațiile de căutare, inserare, ștergere care au complexitatea $O(\log n)$, iar arborii rezultați în urma aplicării acestor operații fac parte tot din clasa $\mathscr{C}$.
- **Lemă:** Dacă `root` este rădăcina unui arbore AVL-echilibrat cu n noduri interne, atunci $height(root) = \Theta(\log n)$.
- **Teoremă:** Clasa arborilor echilibrați este $O(\log n)$-stabilă.

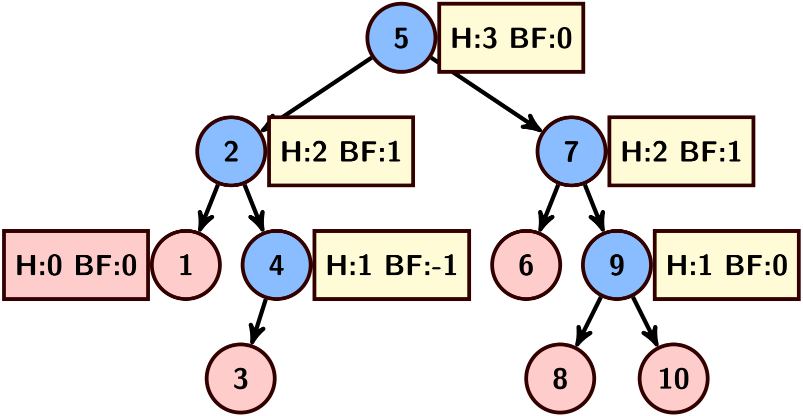
Arbori AVL – Exemple

Este următorul arbore un arbore AVL?



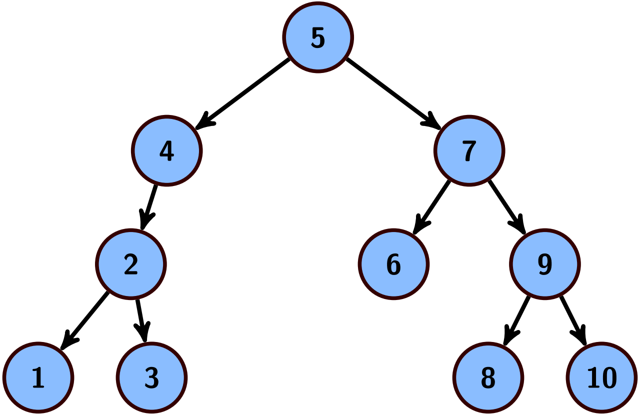
Arbori AVL – Exemple

Este un arbore AVL pentru că $BF(N) \in \{-1, 0, 1\}, \forall N$.



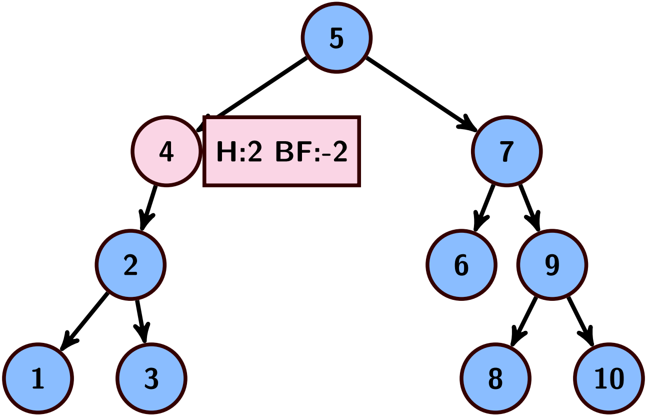
Arbori AVL – Exemple

Este următorul arbore un arbore AVL?



Arbori AVL – Exemple

Nu este pentru că $BF(4) = -2$.



Cum putem reprezenta o structură de tip arbore AVL?

Arbori AVL – Reprezentarea structurii

Cum putem reprezenta o structură de tip arbore AVL?

Putem porni de la reprezentarea folosită anterior pentru arbori binari.

```c
typedef int T;

typedef struct node {
    T value;
    struct node *left, *right;
} TreeNode, *AVLTree;
```

Arbori AVL – Reprezentarea structurii

```c
typedef int T;
typedef struct node {
    T value;
    struct node *left, *right;
} TreeNode, *AVLTree;
```

Mai avem nevoie de un câmp pentru `height`.

```c
typedef int T;
typedef struct node {
    T value;
    int height;
    struct node *left, *right;
} TreeNode, *AVLTree;
```

Arbori AVL – Reprezentarea structurii

Oare de ce avem nevoie de acest câmp suplimentar?

Arbori AVL – Reprezentarea structurii

Oare de ce avem nevoie de acest câmp suplimentar?

Pentru a nu calcula de fiecare dată `height`.

```c
int height(AVLTree root) {
    if (root == NULL)
        return -1;
    return root->height;
}
int max(int x, int y) {
    if (x > y)
        return x;
    return y;
}
```

Arbori AVL – Inițializarea unui nod

```c
AVLTree create(T value) {
    AVLTree root = malloc(sizeof(TreeNode));
    root->value = value;
    root->left = NULL;
    root->right = NULL;
    root->height = 0;
    return root;
}
```

Arbori AVL – Căutarea unei valori

Cum putem realiza căutarea unei valori într-un arbore AVL?

Arbori AVL – Căutarea unei valori

Cum putem realiza căutarea unei valori într-un arbore AVL?

Căutarea se face la fel ca în orice arbore binar de căutare.

```
int contains(AVLTree root, T value) {
    if (root == NULL)
        return 0;
    if (root->value == value)
        return 1;
    if (value < root->value)
        return contains(root->left, value);
    return contains(root->right, value);
}
```

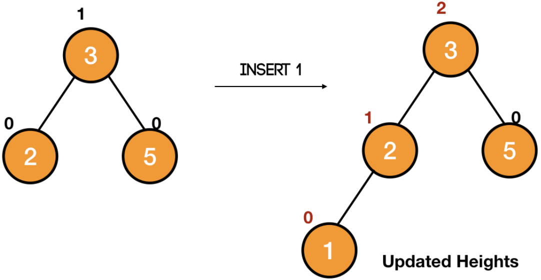
Arbori AVL – Căutarea unei valori

Putem implementa și iterativ funcția de inserare, utilizând un `while`.

```c
int contains_iter(AVLTree root, T value) {
    while (root != NULL && root->value != value) {
        if (value < root->value)
            root = root->left;
        else
            root = root->right;
    }
    if (root == NULL)
        return 0;
    return 1;
}
```

Arbori AVL – Inserarea unei valori

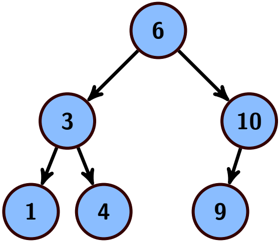
- Introducerea unui nou nod într-un arbore AVL poate determina modificarea factorului de echilibru al unui nod (acesta va deveni 2 sau -2).



- Numai înălțimile nodurilor aflate pe calea de la nodul inserat până la rădăcină pot fi modificate. Trebuie ca după ce am realizat inserare să ne întoarcem și să actualizăm înălțimile pentru aceste noduri.
- Această problemă poate fi remediată prin utilizarea rotațiilor.

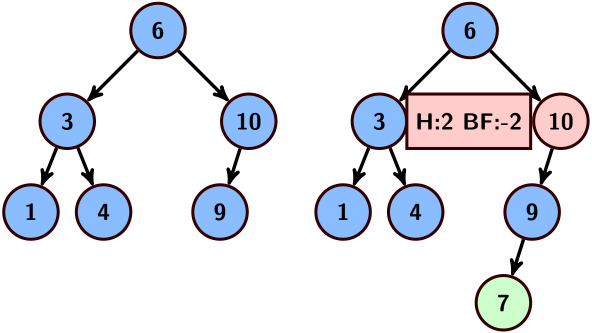
Arbori AVL – Inserarea unei valori

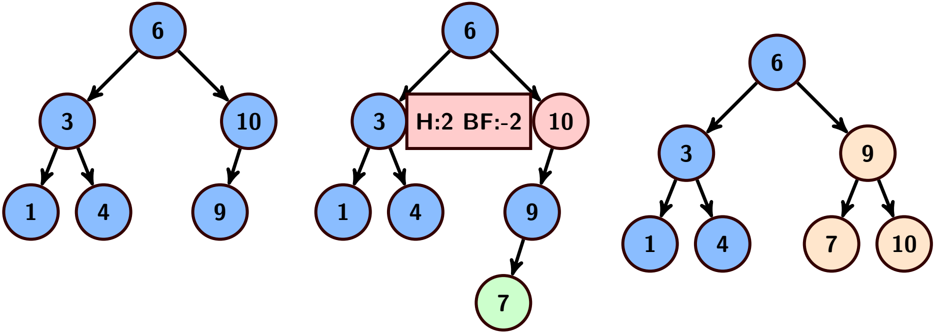
Exemplu



Arbori AVL – Inserarea unei valori

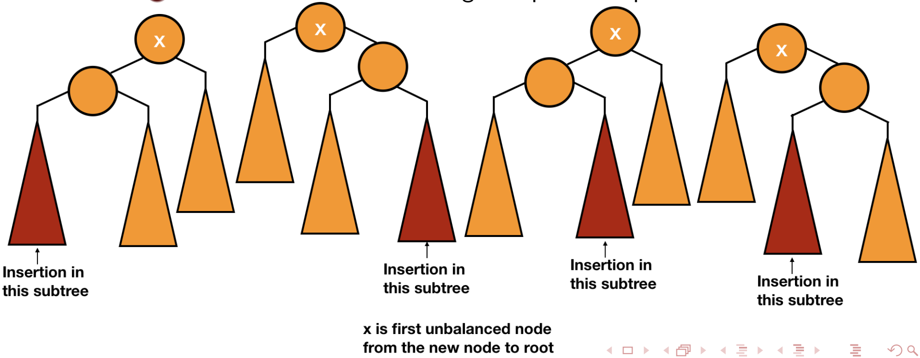
Exemplu




6
3
10
1
4
9
6
3
H:2 BF:-2
10
1
4
9
7
6
3
9
1
4
7
10

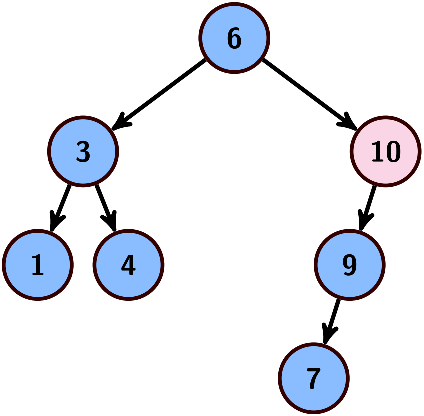
Arbori AVL – Inserarea unei valori

- Atunci când inserăm un nod într-un arbore cu rădăcina x, poate să apară unul dintre următoarele 4 cazuri posibile:
  - Cazuri care necesită o singură rotație
    1. inserăm în subarborele stâng al copilului stâng al lui x;
    2. inserăm în subarborele drept al copilului drept al lui x;
  - Cazuri care necesită două rotații
    3. inserăm în subarborele drept al copilului stâng al lui x;
    4. inserăm în subarborele stâng al copilului drept al lui x.



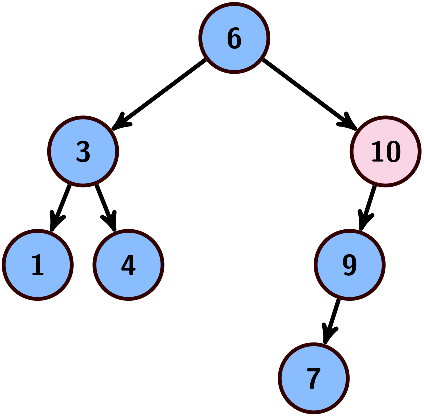
Arbori AVL – Inserarea unei valori

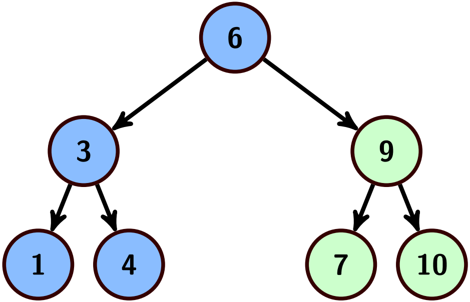
Cum putem echilibra un arbore?



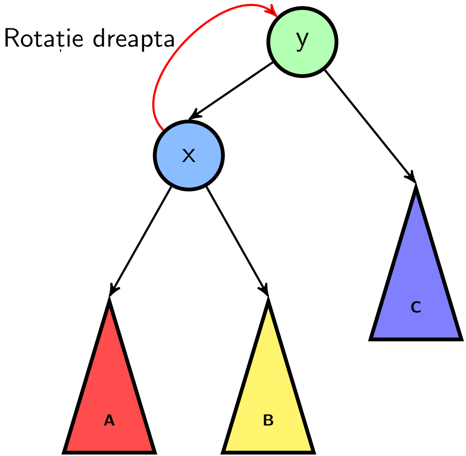
Arbori AVL – Inserarea unei valori

Cum putem echilibra un arbore?

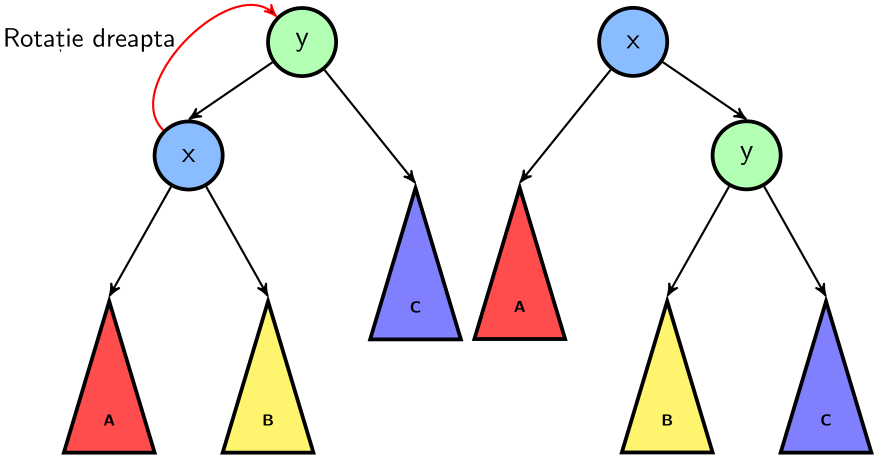




Arbori AVL – Rotație dreapta



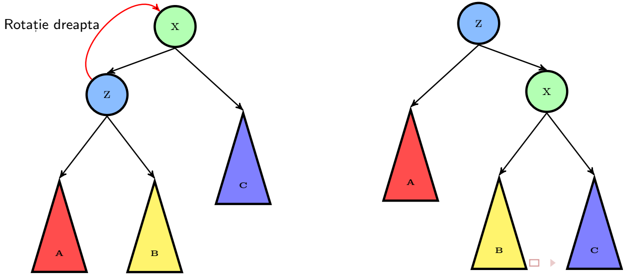
Arbori AVL – Rotație dreapta



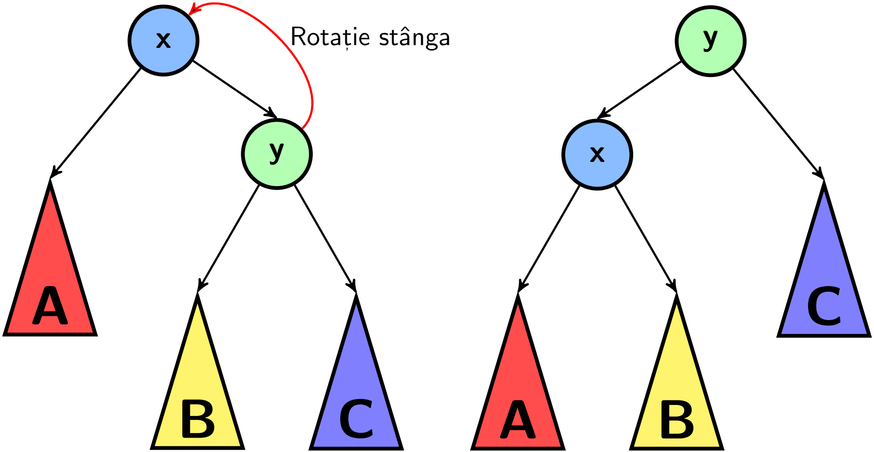
Rotire simplă dreapta

- Funcția realizează rotirea între X și copilul stâng (poate fi apelată numai dacă X are un copil stâng). De asemenea, actualizează înălțimea și întoarce rădăcina.

```
AVLTree rotate_right(AVLTree x) {
    AVLTree z;
    z = x->left;
    x->left = z->right;
    z->right = x;
    x->height = max(height(x->left), height(x->right)) + 1;
    z->height = max(height(z->left), x->height) + 1;
    return z;
}
```



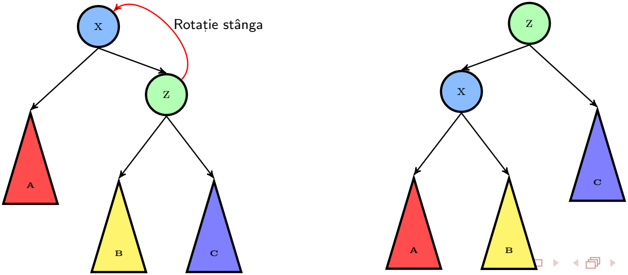
Arbori AVL – Rotație stânga



Rotire simplă stânga

- Funcția realizează rotirea între X și copilul drept (poate fi apelată numai dacă X are un copil drept). De asemenea, actualizează înălțimea și întoarce rădăcina.

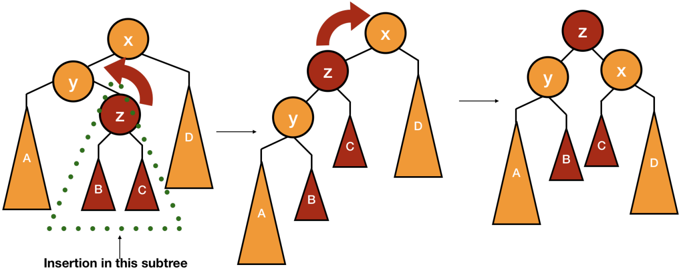
```
AVLTree rotate_left(AVLTree x) {
    AVLTree z;
    z = x->right;
    x->right = z->left;
    z->left = x;
    x->height = max(height(x->left), height(x->right)) + 1;
    z->height = max(x->height, height(z->right)) + 1;
    return z;
}
```



Rotire dublă stânga - dreapta

- Funcția se poate apela dacă X are un copil stâng și copilul stâng a lui X are un copil drept

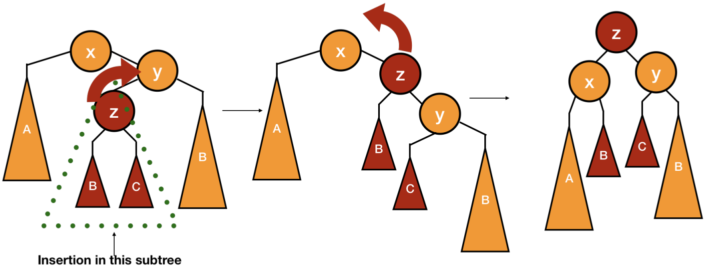
```
AVLTree rotate_left_right(AVLTree x) {
    // rotește între Y și Z
    x->left = rotate_left(x->left);
    // rotește între X și Z
    return rotate_right(x);
}
```



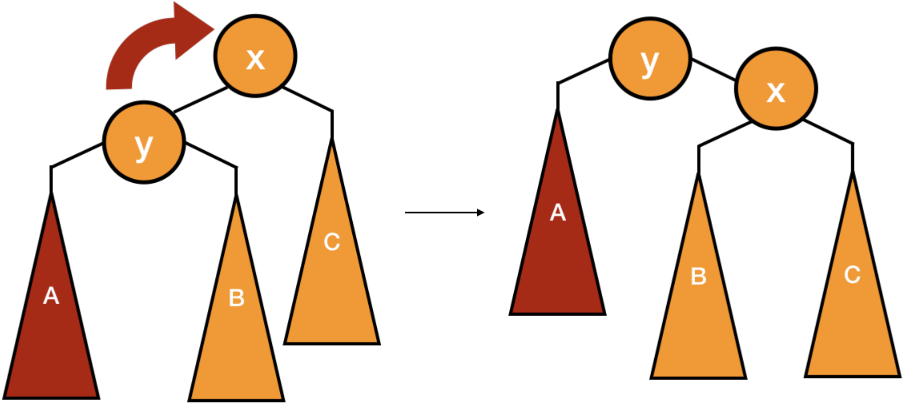
Rotire dublă dreapta - stânga

- Funcția se poate apela dacă X are un copil drept și copilul drept a lui X are un copil stâng

```
AVLTree rotate_right_left(AVLTree x) {
    // rotește între Y și Z
    x->right = rotate_right(x->right);
    // rotește între X și Z
    return rotate_left(x);
}
```



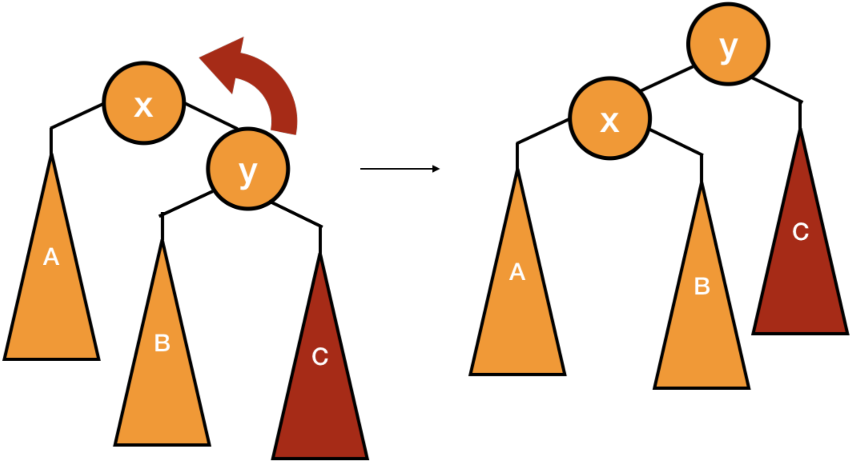
- Putem realiza o rotație dreapta pentru nodul x.



Arbori AVL – Inserarea unei valori

Cazul 2 – inserăm în subarborele drept al copilului drept al lui x

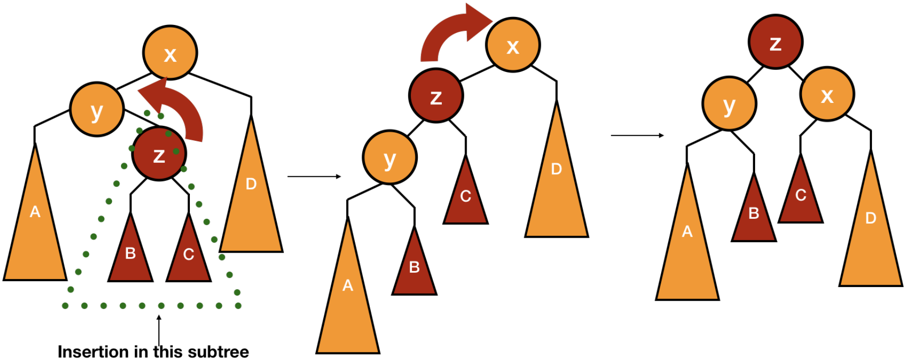
- Putem realiza o rotație stânga pentru nodul x.



Arbori AVL – Inserarea unei valori

Cazul 3 – inserăm în subarborele drept al copilului stâng al lui x

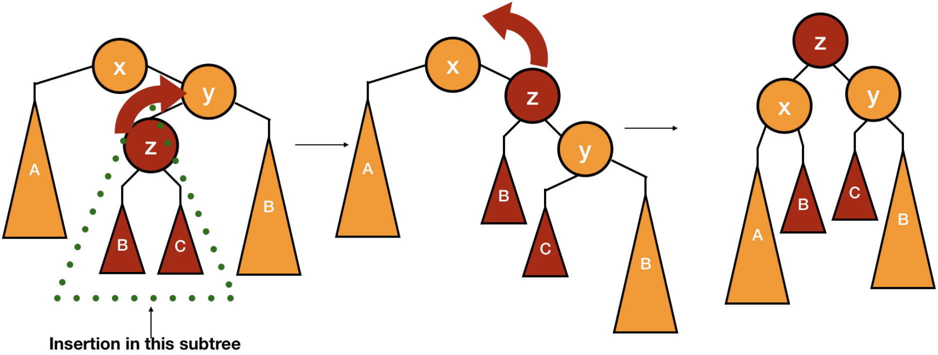
- Prima dată vom realiza o rotație stânga pentru nodul y, iar apoi o rotație dreapta pentru nodul x.



Arbori AVL – Inserarea unei valori

Cazul 4 – inserăm în subarborele stâng al copilului drept al lui x

- Prima dată vom realiza o rotație dreapta pentru nodul y, iar apoi o rotație stânga pentru nodul x.



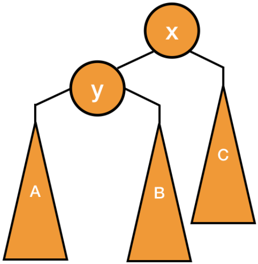
Arbori AVL – Inserarea unei valori

```
AVLTree insert(AVLTree root, T value) {
    if (root == NULL) {
        root = create(value);
        return root;
    }
    // Căutăm poziția pentru valoare în arbore
    if (value < root->value) {
        root->left = insert(root->left, value);
    } else if (value > root->value) {
        root->right = insert(root->right, value);
    } else
        return root;
    // Actualizez înălțimea
    root->height = max(height(root->left),
↪  height(root->right)) + 1;
```

Arbori AVL – Inserarea unei valori

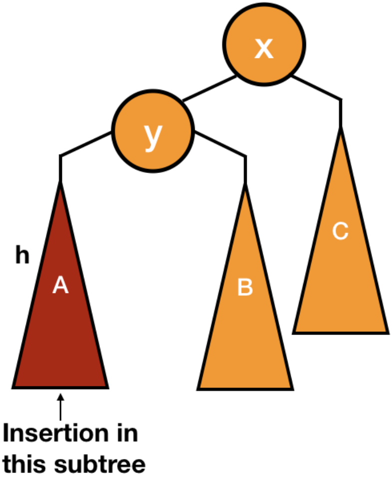
```
    int bf = height(root->left) - height(root->right);
    // Cazul Left Left
    if (bf > 1 && value < root->left->value)
        return rotate_right(root);
    // Cazul Right Right
    if (bf < -1 && value > root->right->value)
        return rotate_left(root);
    // Cazul Left Right
    if (bf > 1 && value > root->left->value)
        return rotate_left_right(root);
    // Cazul Right Left
    if (bf < -1 && value < root->right->value)
        return rotate_right_left(root);
    return root;
}
```

- Am văzut ce cazuri apar atunci când realizăm inserarea și cum le-am putea trata pe fiecare în parte.
- Mai rămâne să descoperim dacă acestea garantează păstrarea proprietăților arborilor AVL după inserare și modificare prin rotații.
- Pentru acest lucru, vom studia cele două categorii mari de cazuri, pornind de la următorul arbore AVL în care urmează să inserăm o valoare:

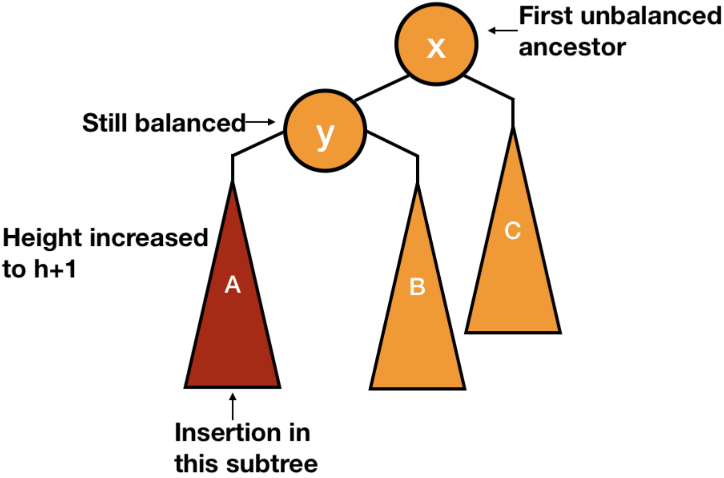


Subtree Before Insertion

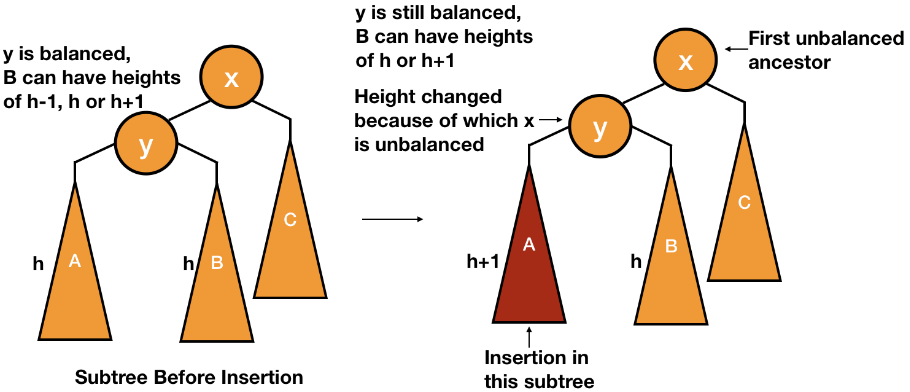
1. Cazuri care necesită o singură rotație



1. Cazuri care necesită o singură rotație

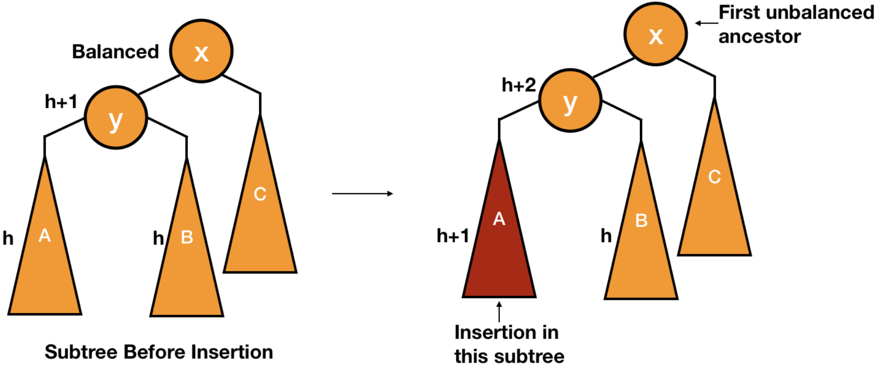


1. Cazuri care necesită o singură rotație

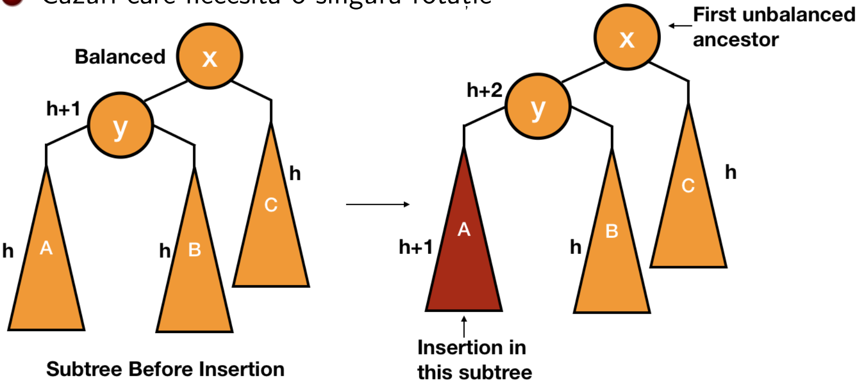


If height of B is h+1, y must have initial height of h+2 and can't change due to insertion. So, only possible height of B is h

1. Cazuri care necesită o singură rotație

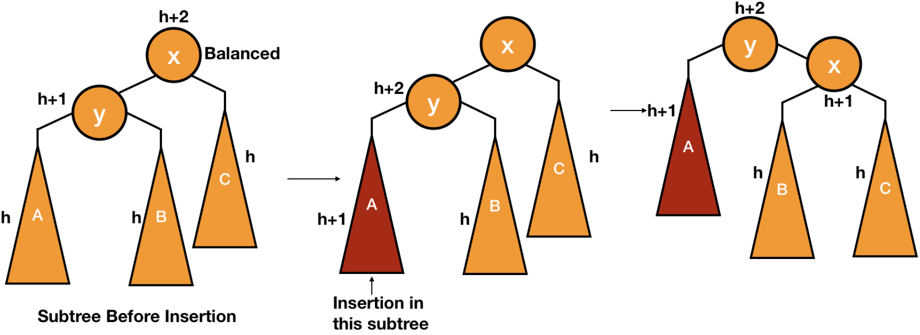


1. Cazuri care necesită o singură rotație

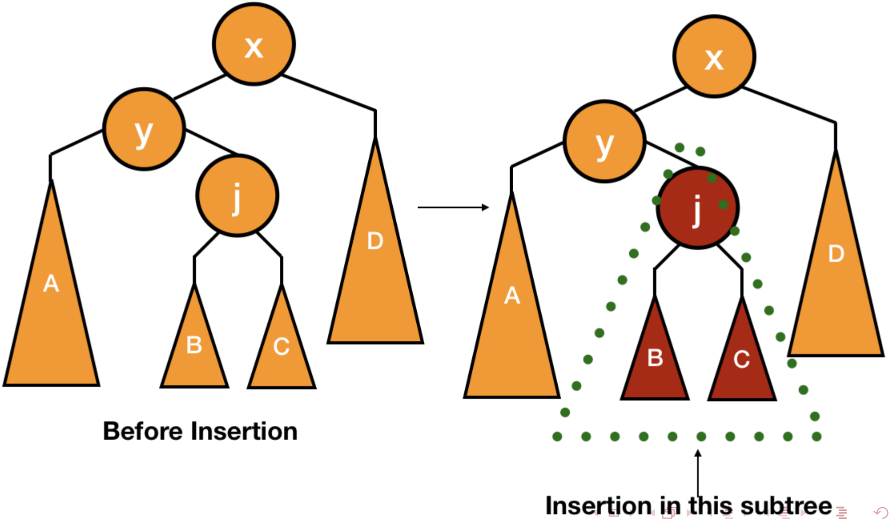


**If height of C is decreased from h,
It will not be balanced before insertion.
If it is increased from h, It will become balanced after insertion.**

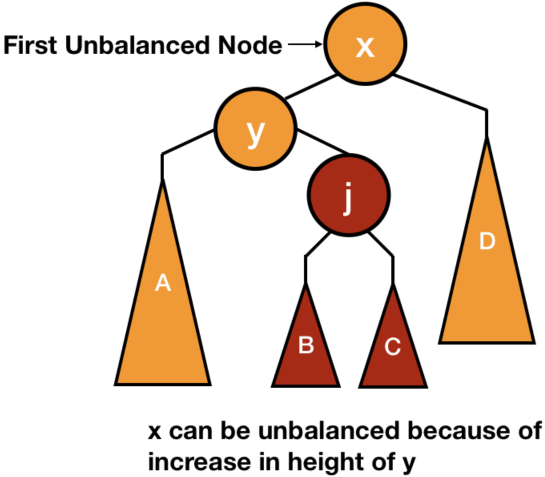
1. Cazuri care necesită o singură rotație



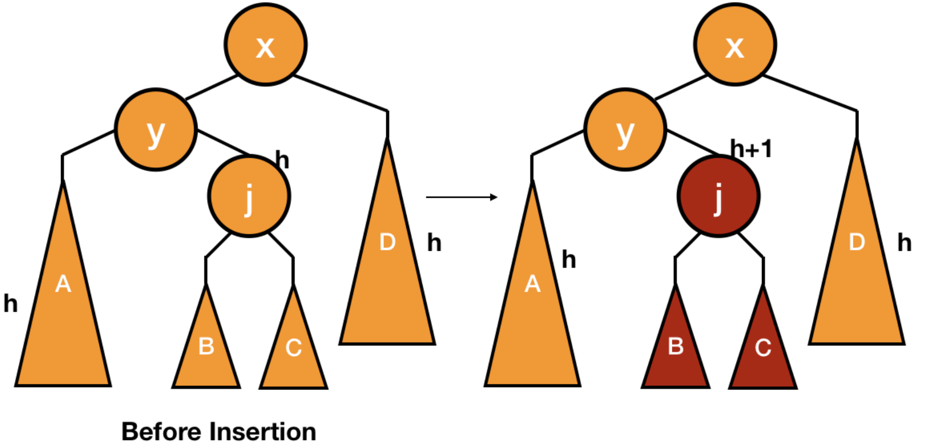
2. Cazuri care necesită două rotații



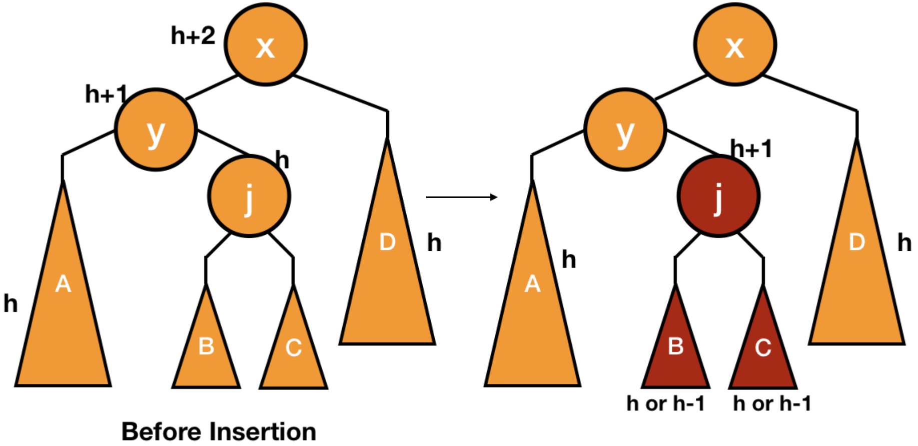
2. Cazuri care necesită două rotații



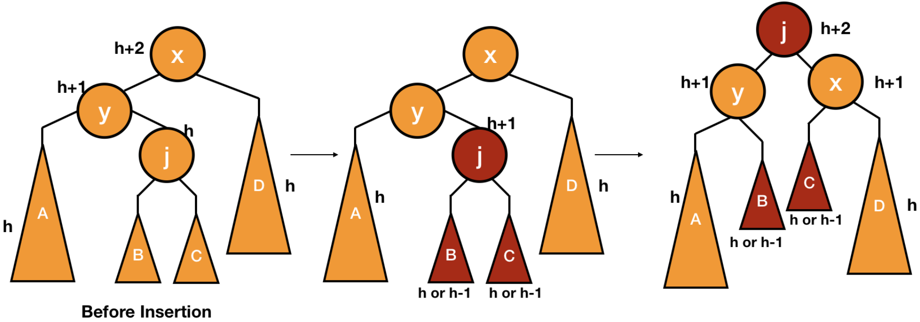
2. Cazuri care necesită două rotații



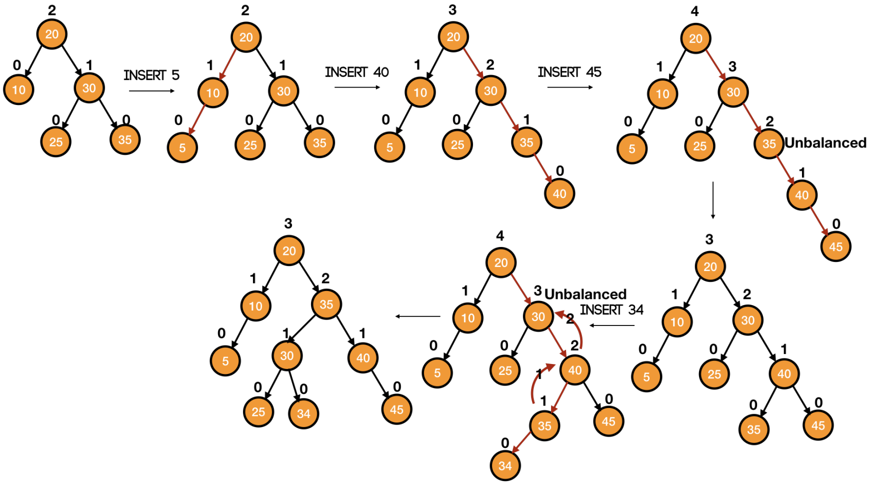
2. Cazuri care necesită două rotații



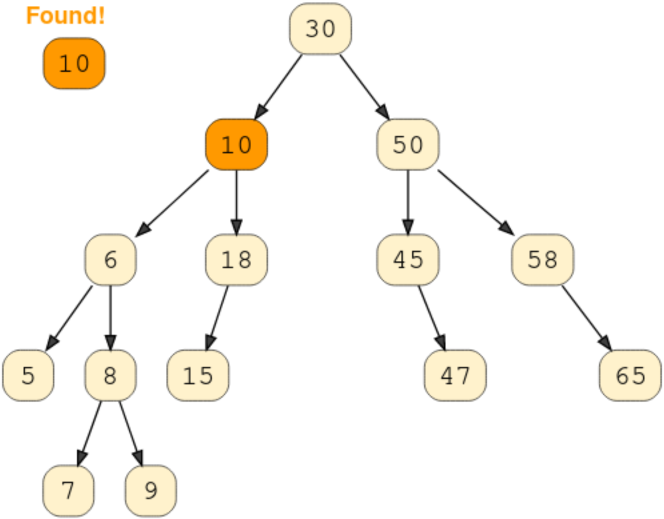
2. Cazuri care necesită două rotații



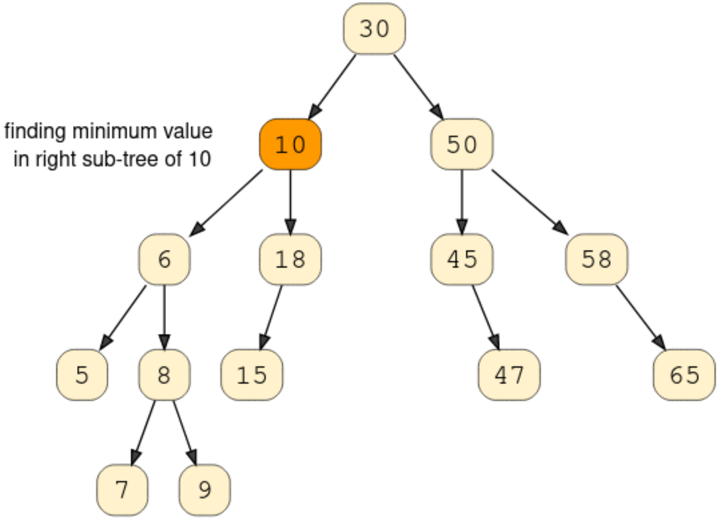
2. Cazuri care necesită două rotații



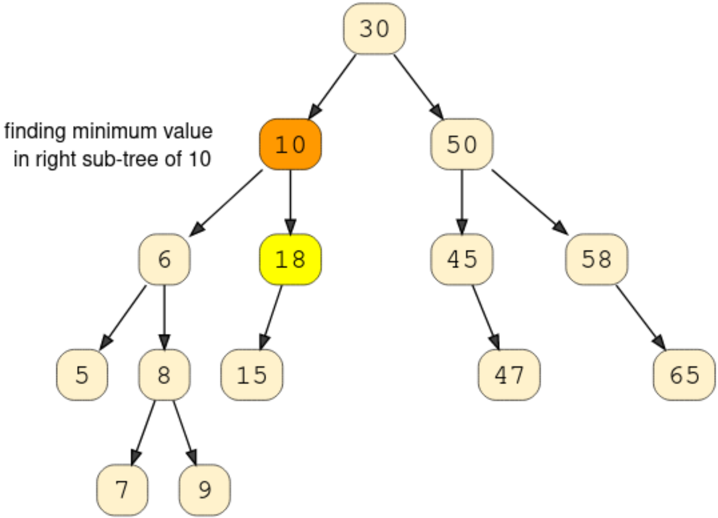
Arbori AVL – Ștergerea unui element



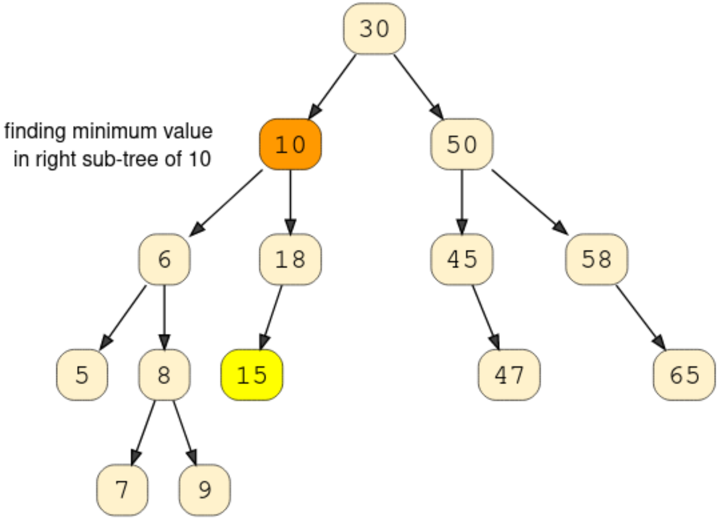
Arbori AVL – Ștergerea unui element



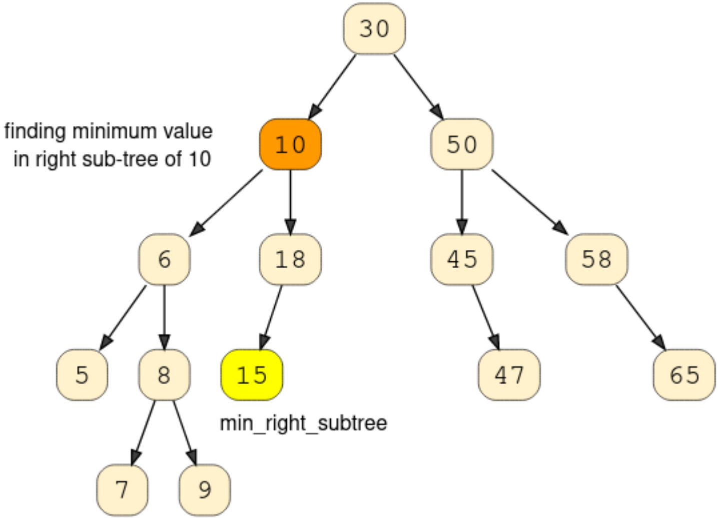
Arbori AVL – Ștergerea unui element



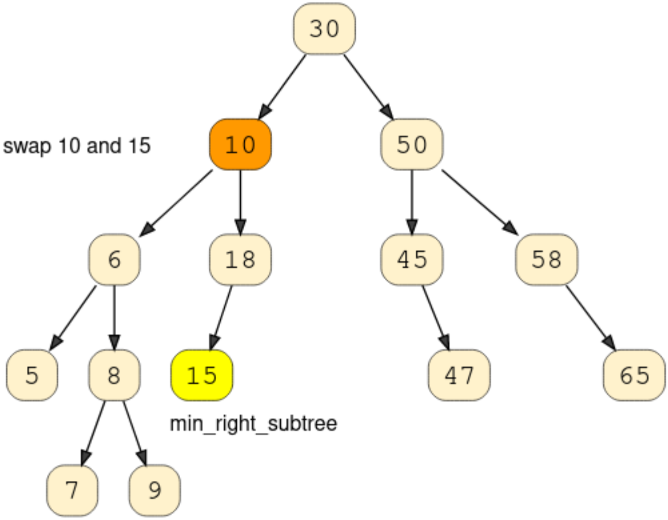
Arbori AVL – Ștergerea unui element



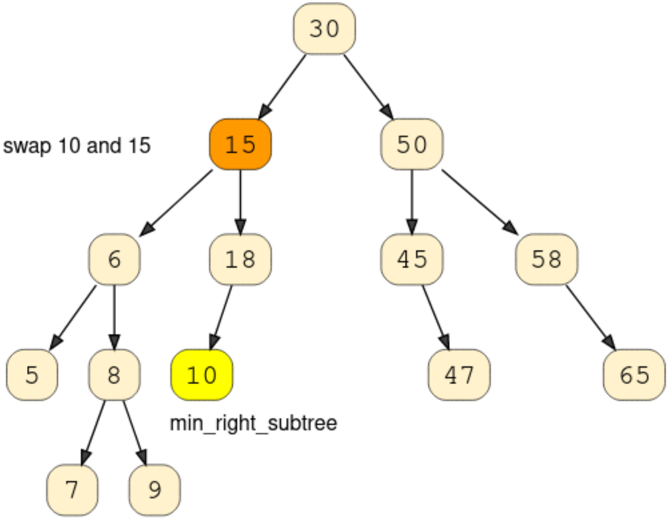
Arbori AVL – Ștergerea unui element



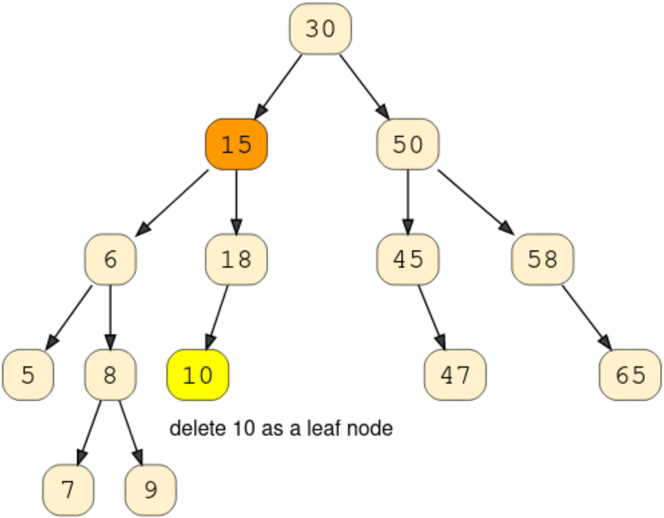
Arbori AVL – Ștergerea unui element



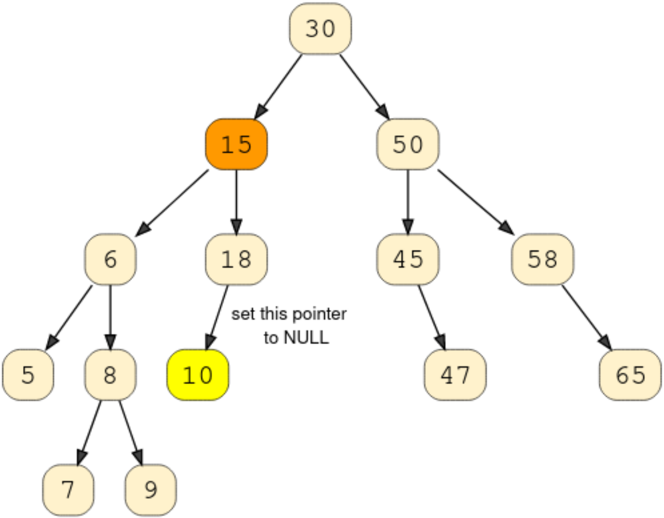
Arbori AVL – Ștergerea unui element



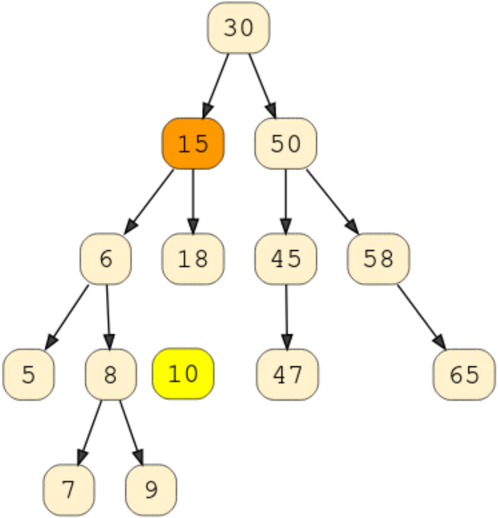
Arbori AVL – Ștergerea unui element



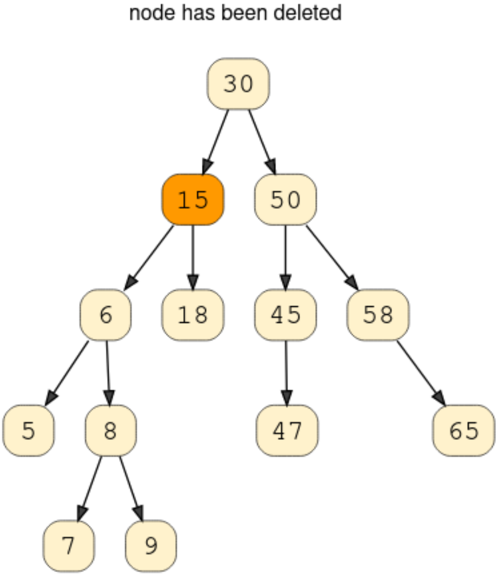
Arbori AVL – Ștergerea unui element



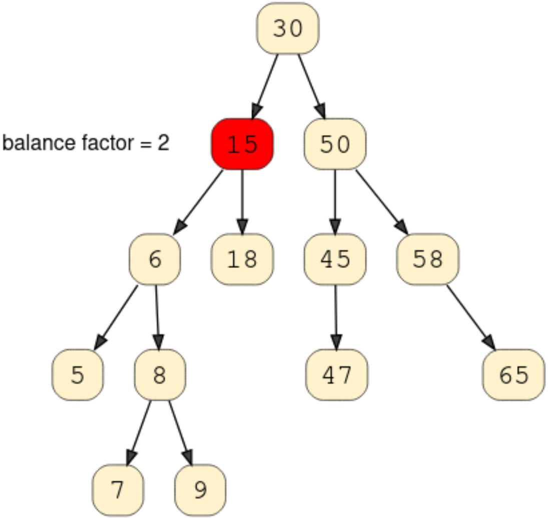
Arbori AVL – Ștergerea unui element



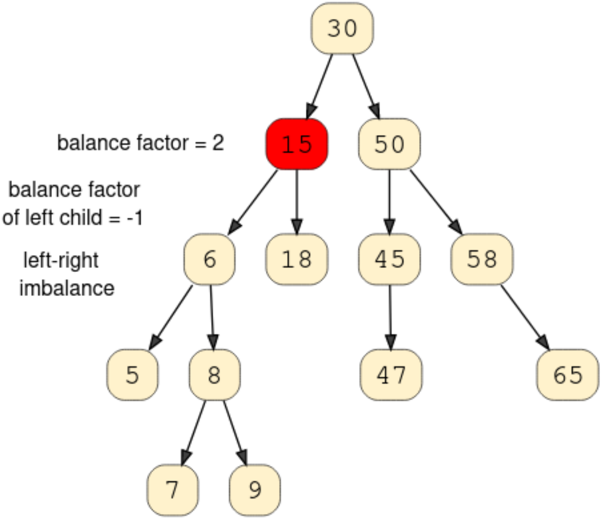
Arbori AVL – Ștergerea unui element



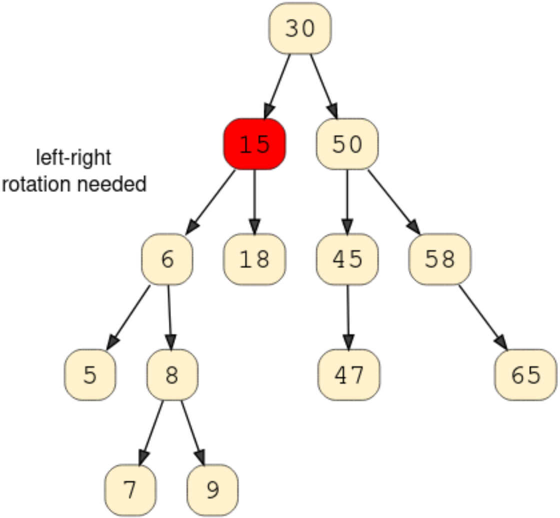
Arbori AVL – Ștergerea unui element



Arbori AVL – Ștergerea unui element

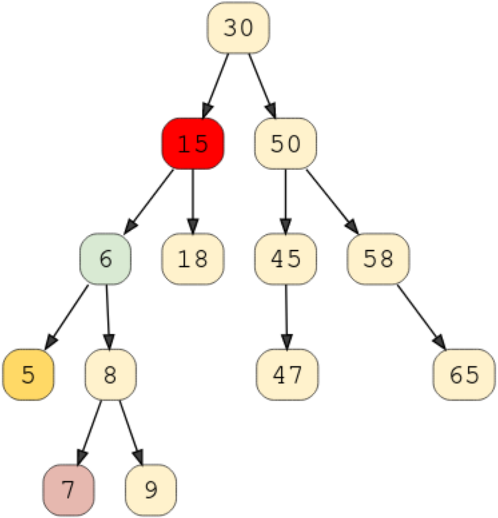


Arbori AVL – Ștergerea unui element

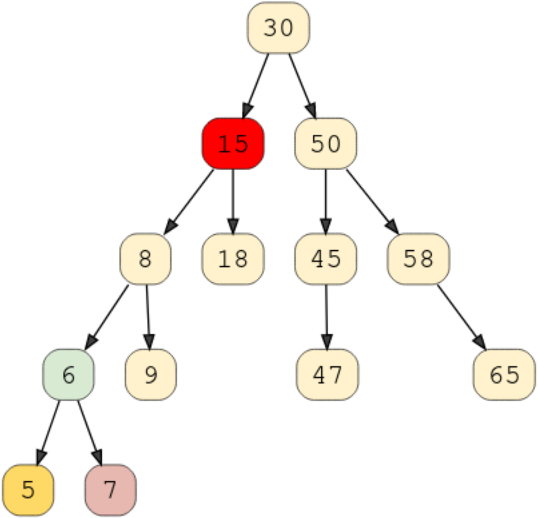


Arbori AVL – Ștergerea unui element

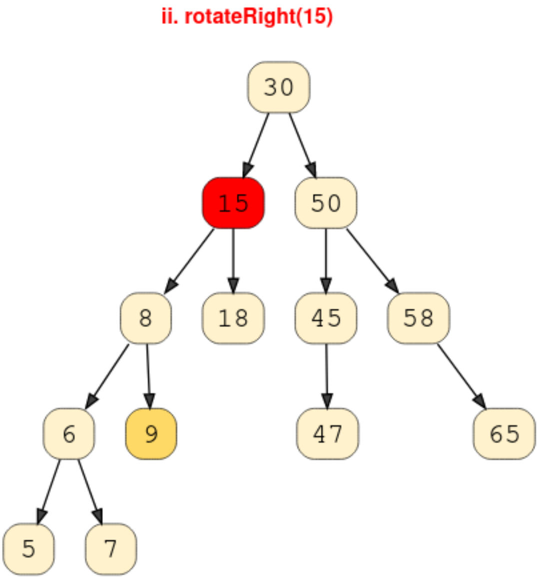
i. rotateLeft(6)



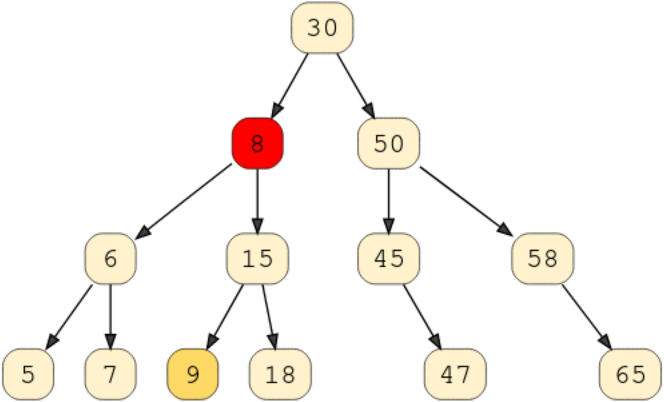
Arbori AVL – Ștergerea unui element



Arbori AVL – Ștergerea unui element



Arbori AVL – Ștergerea unui element



Vă mulțumesc pentru atenție!