# AVL Trees

D.E ZEGOUR

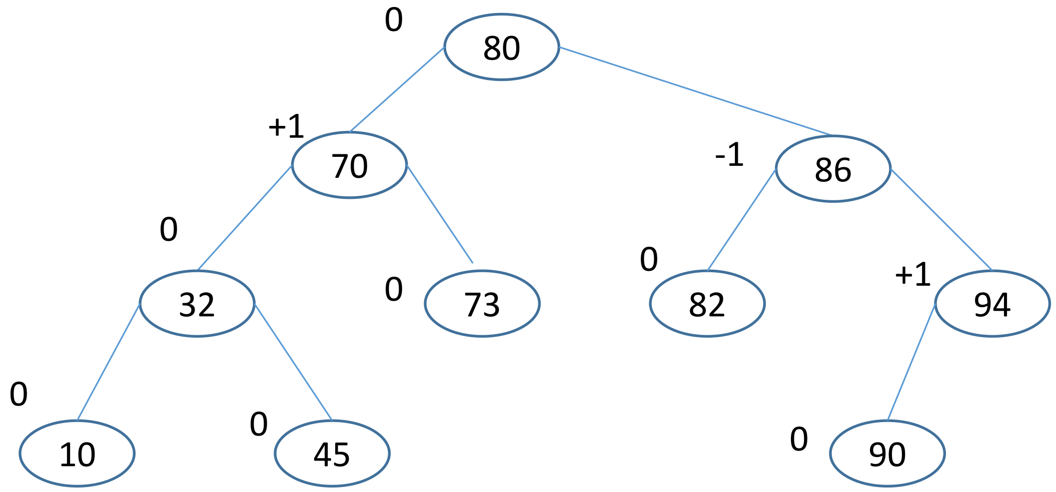
Ecole Supérieure d'Informatique

ESI

# AVL Trees

**Definition**

An AVL tree is a balanced binary search tree



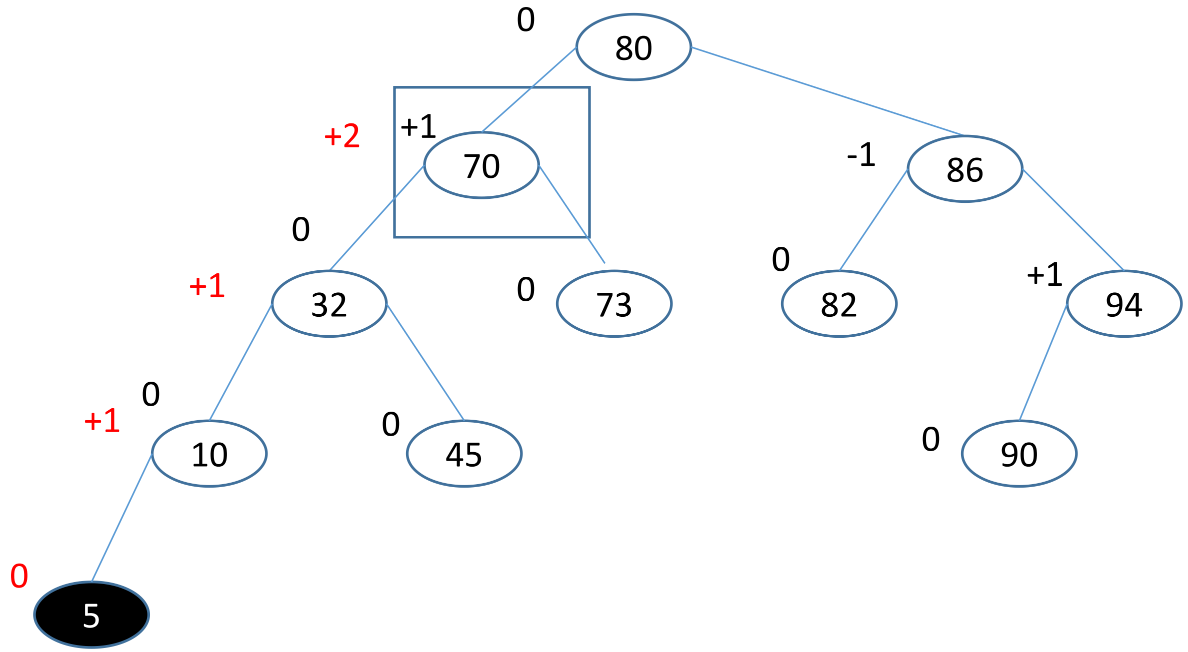

G.M. Abelson-Velskii et E.M. Landis

For any node n:
| Depth(LC(n)) - Depth(RC(n)) | ≤ 1

Requires adding a Balance field (or balance factor) within each node.

# AVL Trees

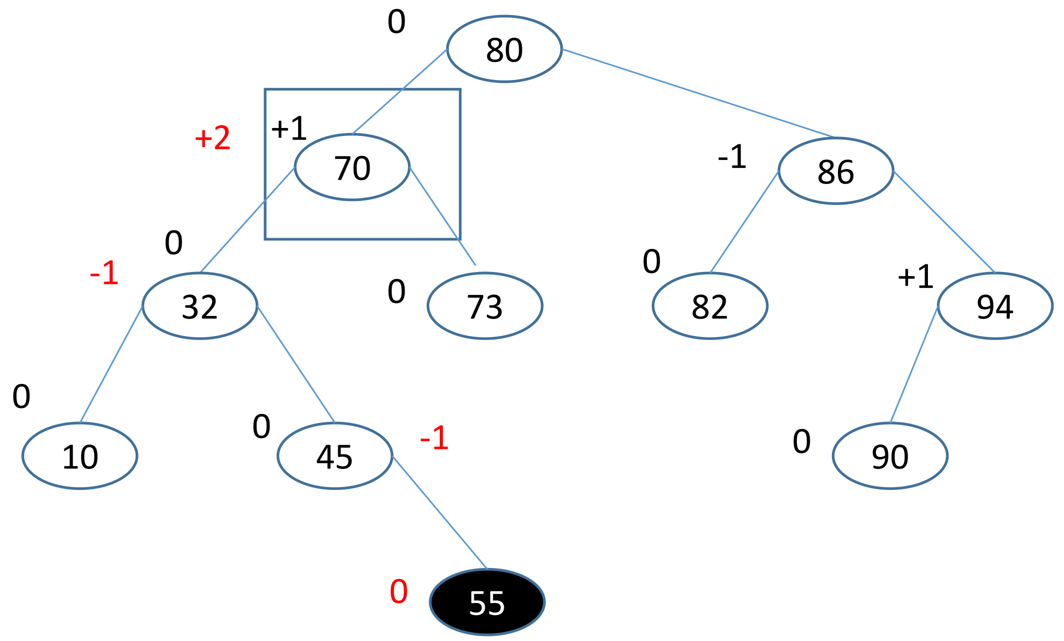
**Insertion : Imbalance Case**




Example 1

# AVL Trees

**Insertion : Imbalance Case**




Example 2

# AVL Trees

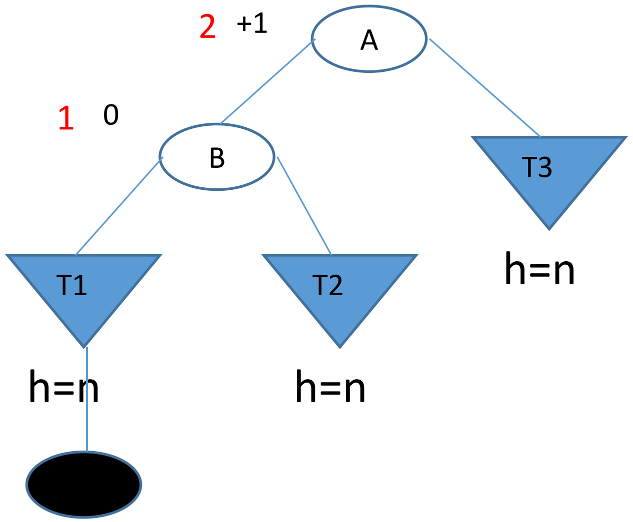
**Insertion : Balancing Technique**

Let's examine a subtree of root A having a balance factor equal to +1

Node A therefore has at least one node to its left, let's call it B.

Case 1 : The new node (in black) is inserted into the left subtree of B.

So, f(B) becomes 1 and f(A) becomes 2



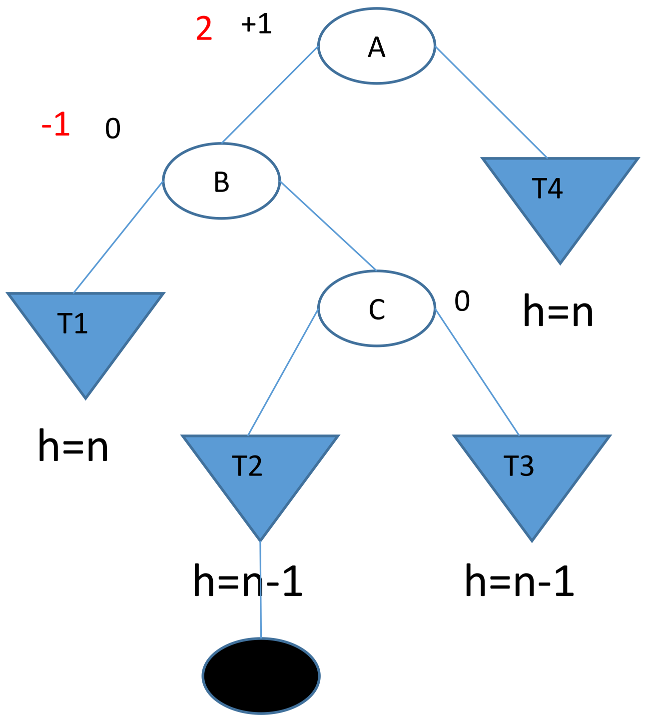
# AVL Trees

**Insertion : Balancing Technique**

Case 2 :The new node is inserted into the right subtree of B.

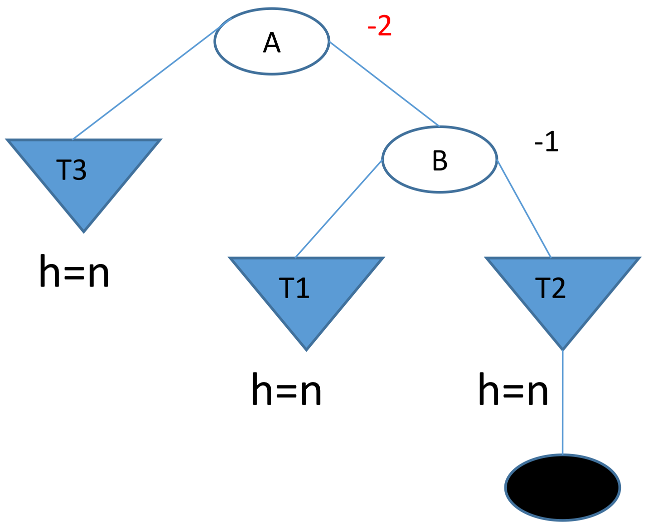
Node B, therefore, has at least one node to its right.

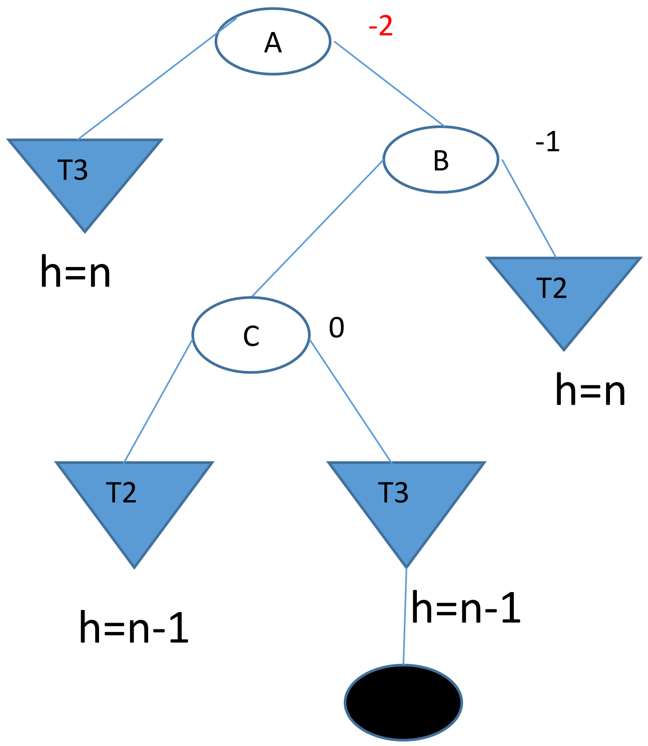
f(B) becomes -1 and f(A) becomes 2.



# AVL Trees

**Insertion : Balancing Technique**




Two symmetrical cases

# AVL Trees

**Insertion : Balancing Technique**

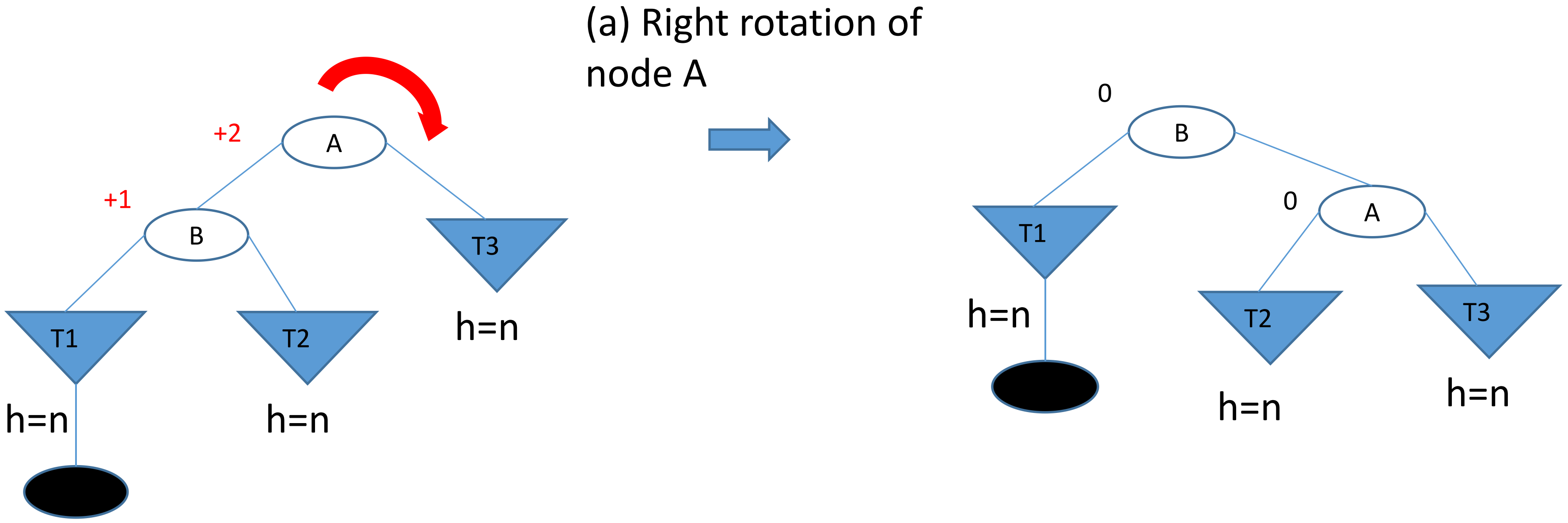
Transform the unbalanced tree in such a way that:

(i) The inorder traversal is preserved.

(ii) The transformed tree is balanced
according to the definition of AVL trees.

# AVL Trees

**Insertion : Balancing Technique**

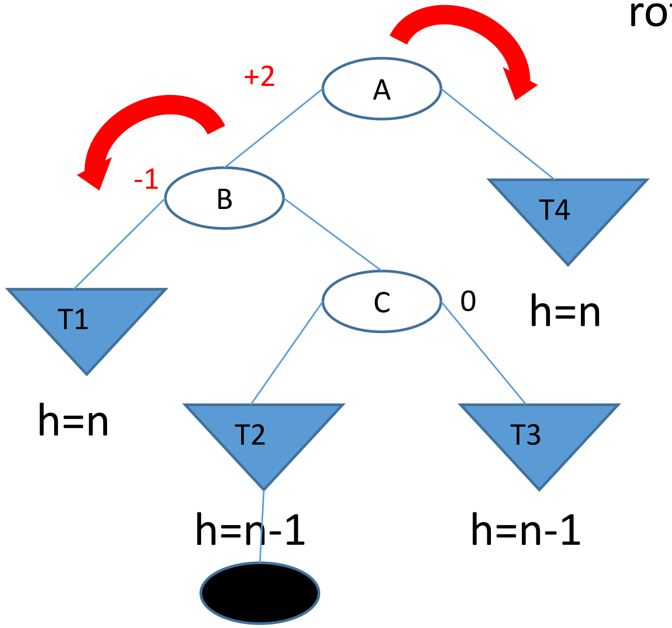


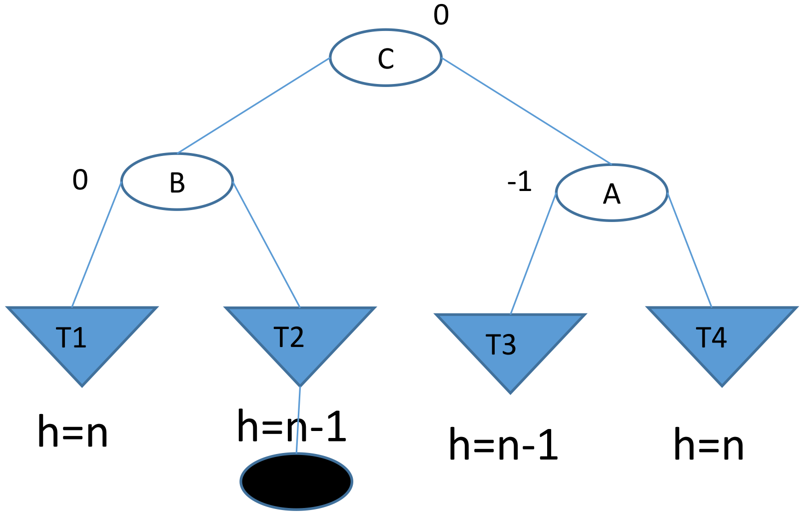

The height remains constant at (n+2), the algorithm comes to a halt.

# AVL Trees

**Insertion : Balancing Technique**

(b) Left rotation of node B followed by a right rotation of node A




The height remains constant at (n+2), the algorithm comes to a halt.

# AVL Trees

**Insertion : Balancing Technique**

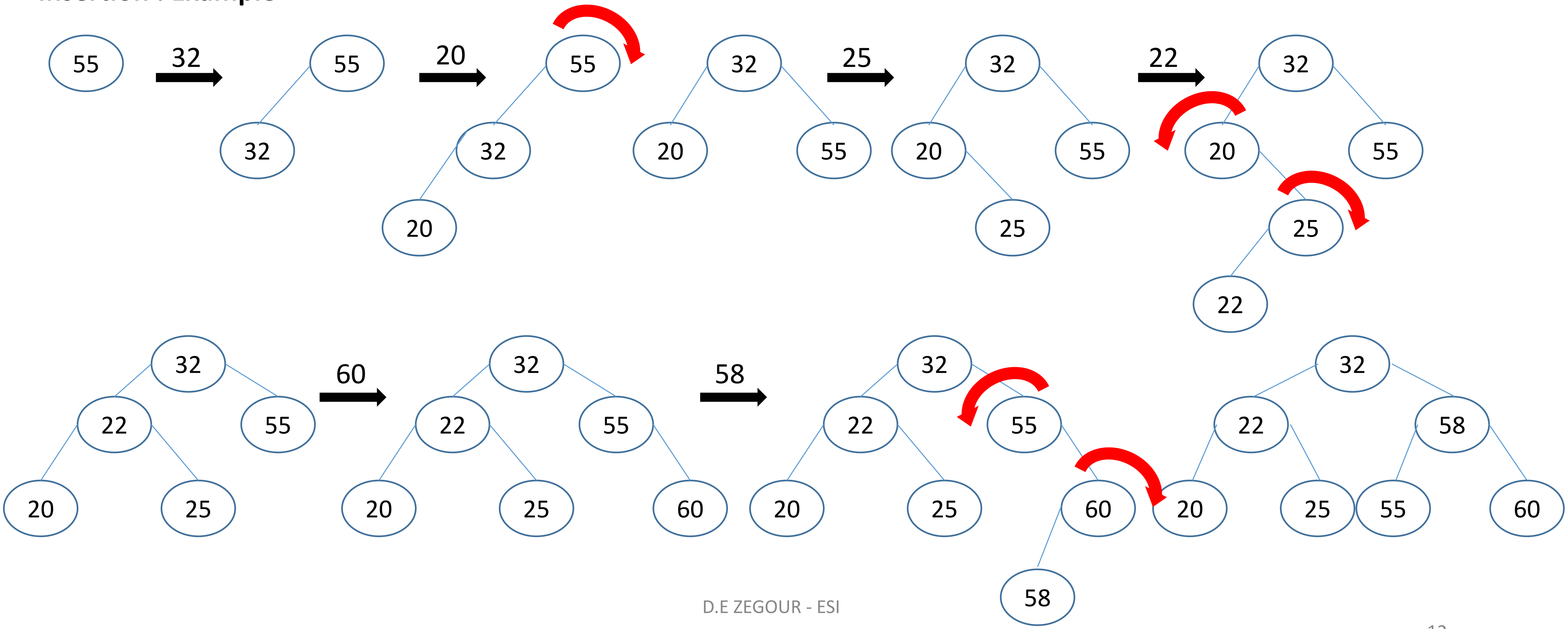
The first part of the algorithm involves inserting the data into the tree without considering the balance factor.

Update the balances and find the youngest ancestor, namely Y, which becomes unbalanced.

The second part carries out the transformation starting from Y.

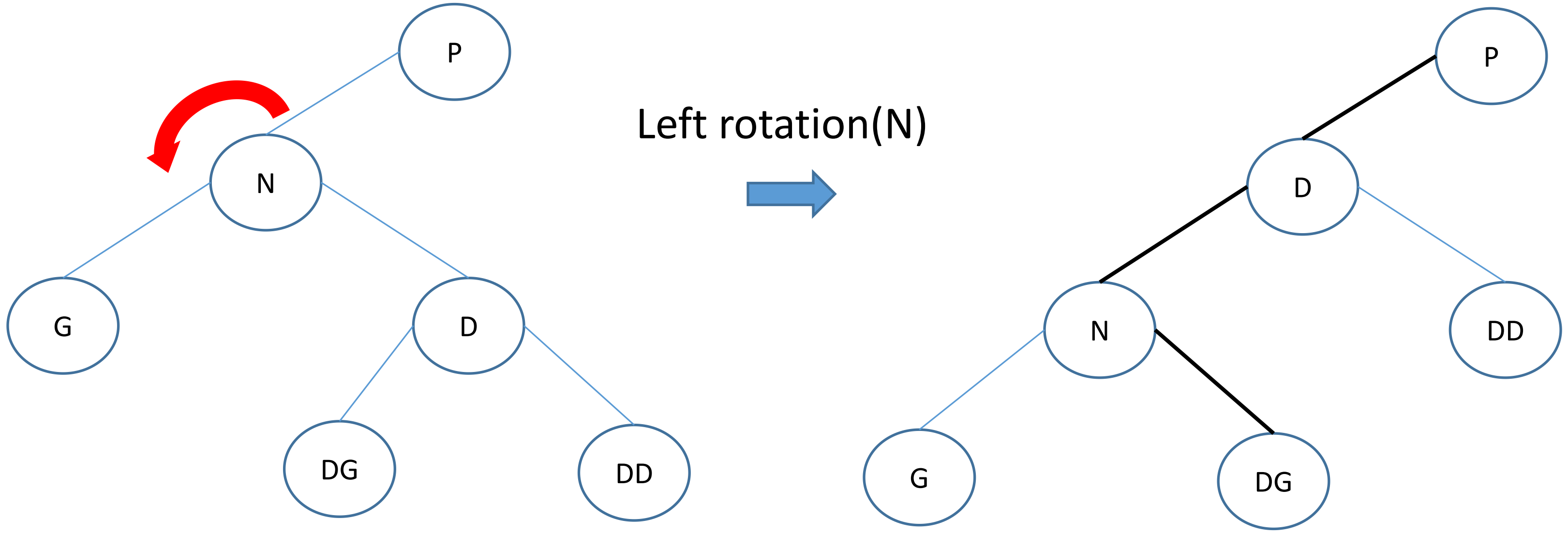
# AVL Trees

**Insertion : Example**



# AVL Trees

**Left Rotation : Algorithm**

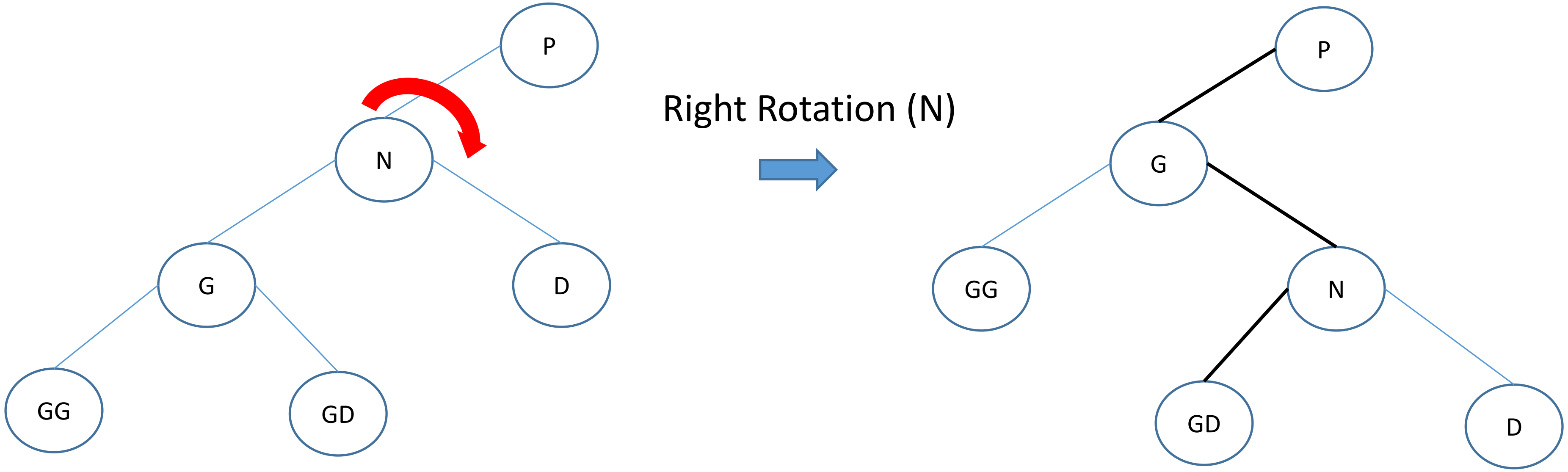



P : Parent of N

D := RC(N)    ASS_RC(N, DG)    ASS_LC(D, N))    ASS_LC(P, D)

# AVL Trees

**Right Rotation : Algorithm**




P : Parent of N

G := LC(N)     ASS_LC(N, GD)     ASS_RC(G, N))     ASS_RC(P G)

# AVL Trees

**Deletion: Principle**

Step 1: as in an ordinary binary search tree.

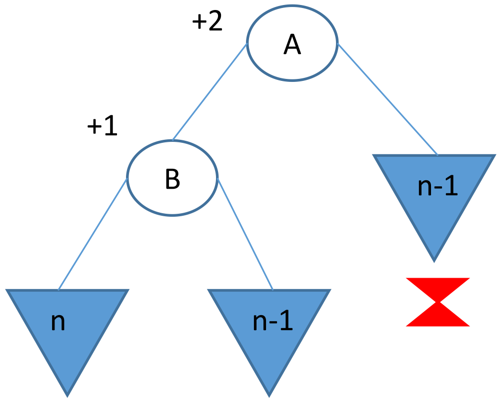
Step 2 : Update balance factors

Step 3 : If a balance factor is violated, maintenance

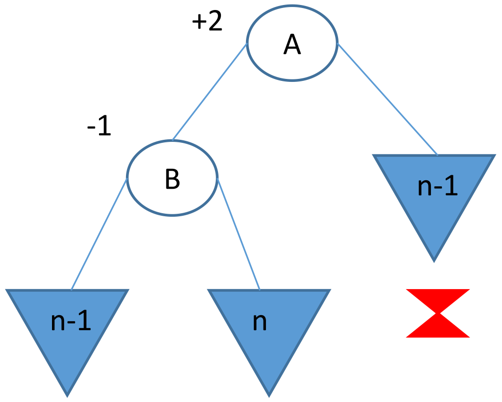
# AVL Trees

**Deletion : Balancing Technique**

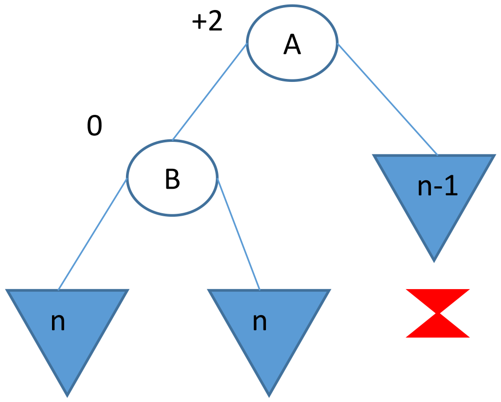
Case where the balance factor of node A becomes +2
→ The left child B of A must exist




B has a balance equal to + 1




B has a balance equal to -1

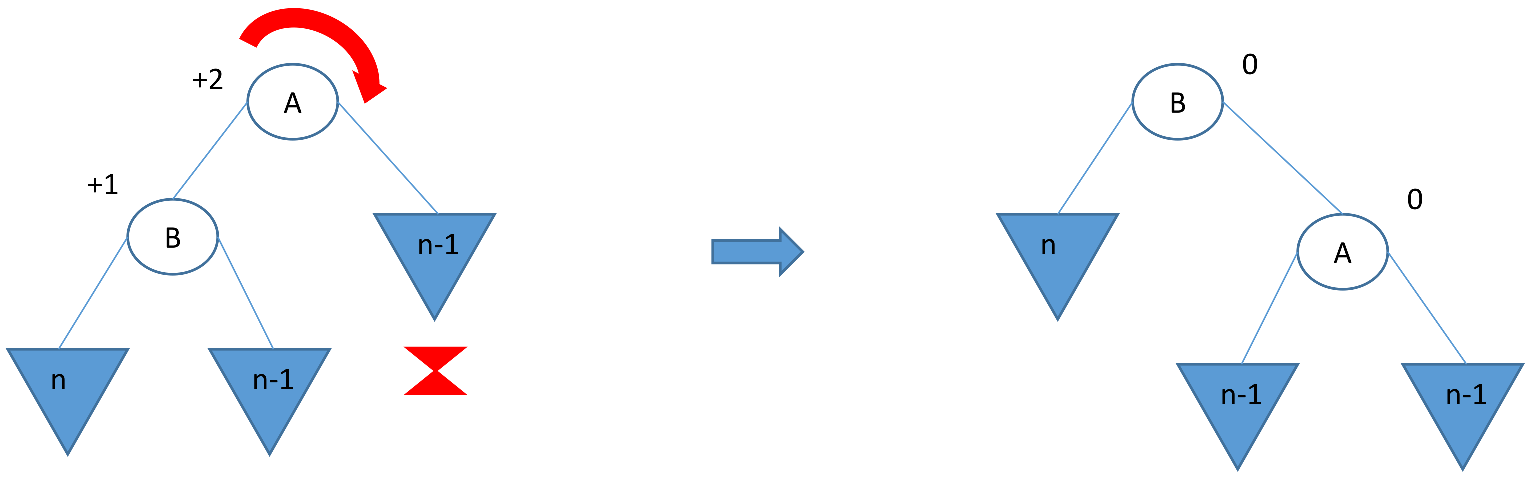



B has a balance equal to 0

# AVL Trees

**Deletion : Balancing Technique**

B has a balance equal to + 1



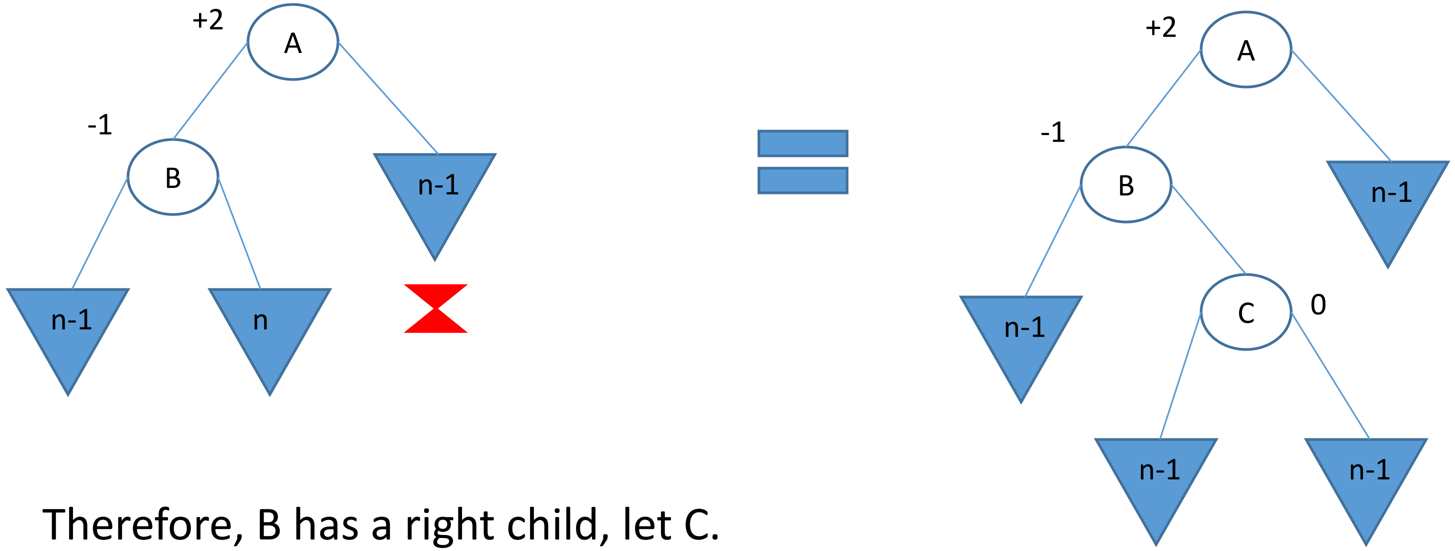

The height changes from (n+2) to (n+1), with a possible cascade effect.

# AVL Trees

**Deletion : Balancing Technique**

B has a balance equal to -1

Case Balance (C)= 0

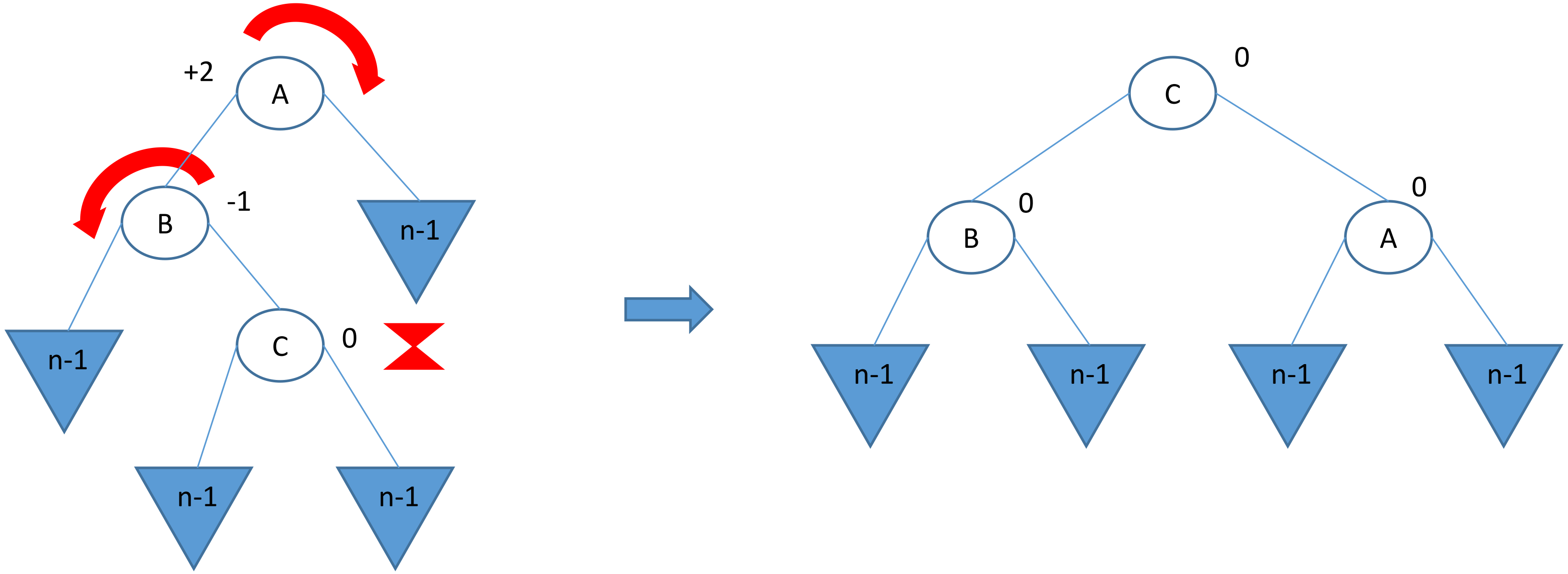



Therefore, B has a right child, let C.

# AVL Trees

**Deletion : Balancing Technique**

B has a balance equal to -1 , C is its right child with a balance equal to 0



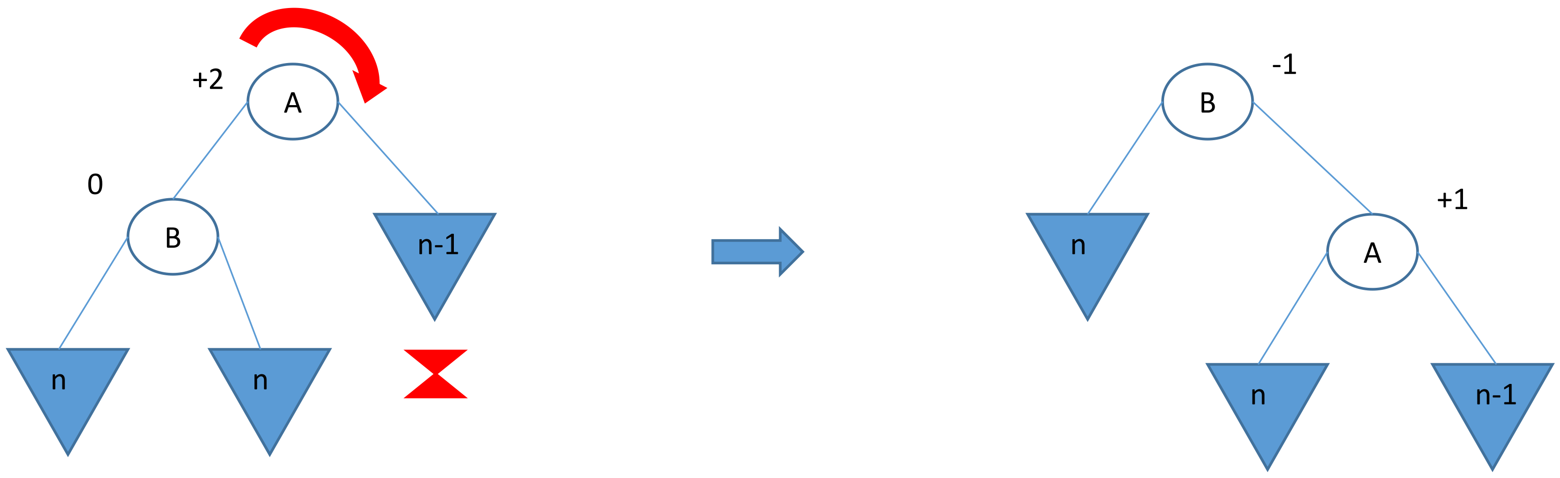

The same goes for balance(C) = +1 or -1.

The height changes from (n+2) to (n+1), with a possible cascade effect.

# AVL Trees

**Deletion : Balancing Technique**

B has a balance equal to 0




The height (n+2) does not change, no cascade.

# AVL Trees

**Deletion: Principle**

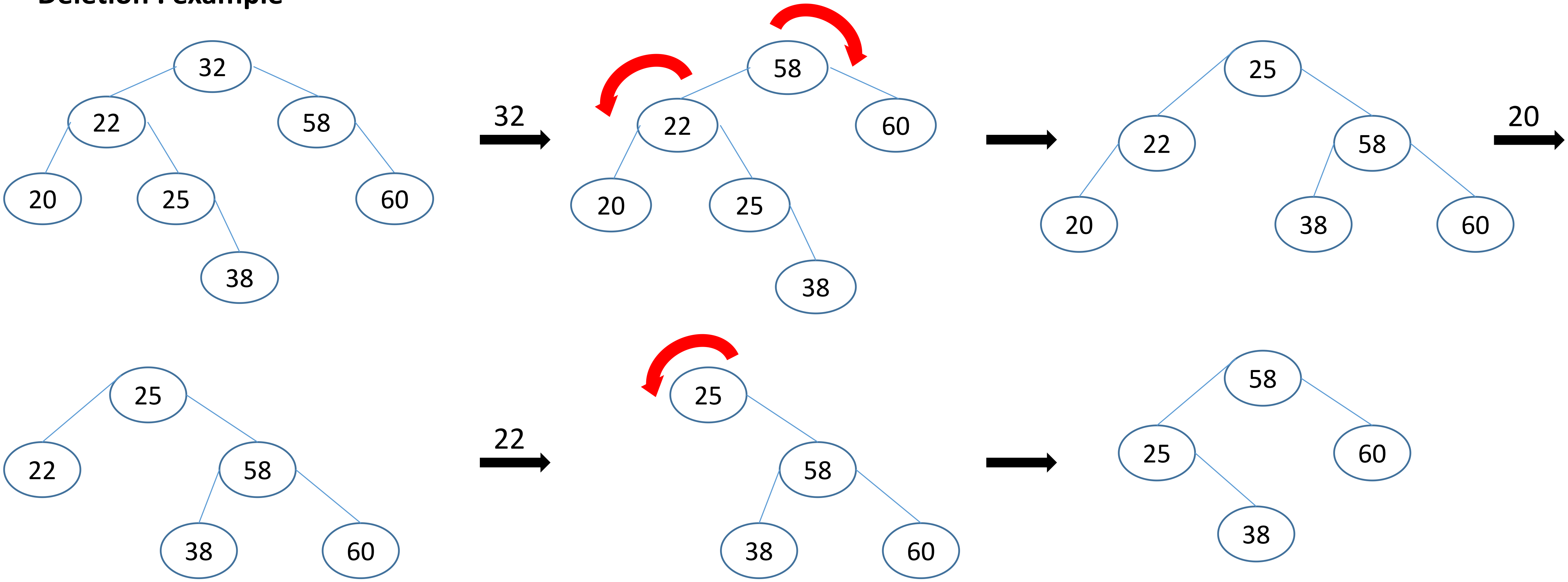
Symmetrical treatment in the case where the balance of a node A becomes -2.

Case where the balance factor of node A becomes - 2
→ The right child B of A must exist

The following cases may occur:
B has a balance equal to +1,
B has a balance equal to –1,
B has a balance equal to 0

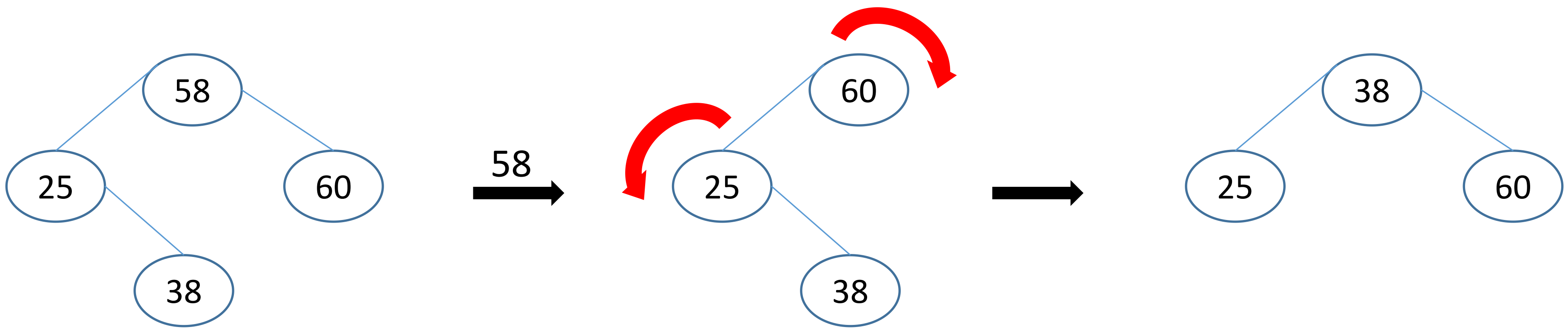
# AVL Trees

**Deletion : example**



# AVL Trees

**Deletion : example**



# AVL Trees

**Synthesis**

Maximum depth of a balanced binary tree: 1.44 * Log2n.

Searching in such a tree never requires more than 44% additional comparisons than for a complete binary tree.

Maintenance operations:
Restructuring = 1 rotation or double rotation

Insertion: at most 1 restructuring
Deletion: at most Log2(N) restructurings"