Lecture 06

Adelson, Velskii & Landis Tree

AVL Trees

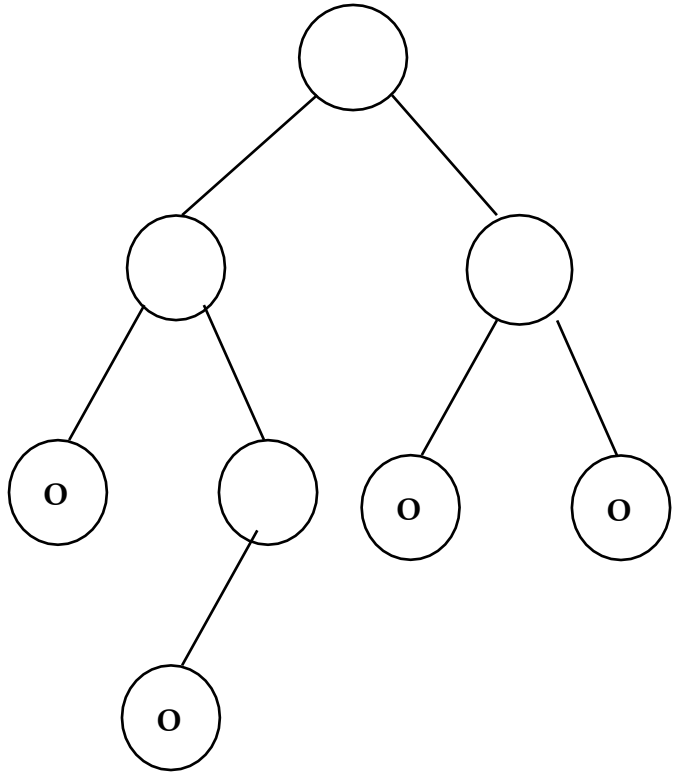
AVL Tree

- AVL trees are height-balanced binary search trees
- Balance factor of a node=
  - height(left sub tree) - height(right sub tree)
- An AVL tree has balance factor calculated at every node
  - For every node, heights of left and right sub tree can
    - differ by no more than 1
      - Store current heights in each node

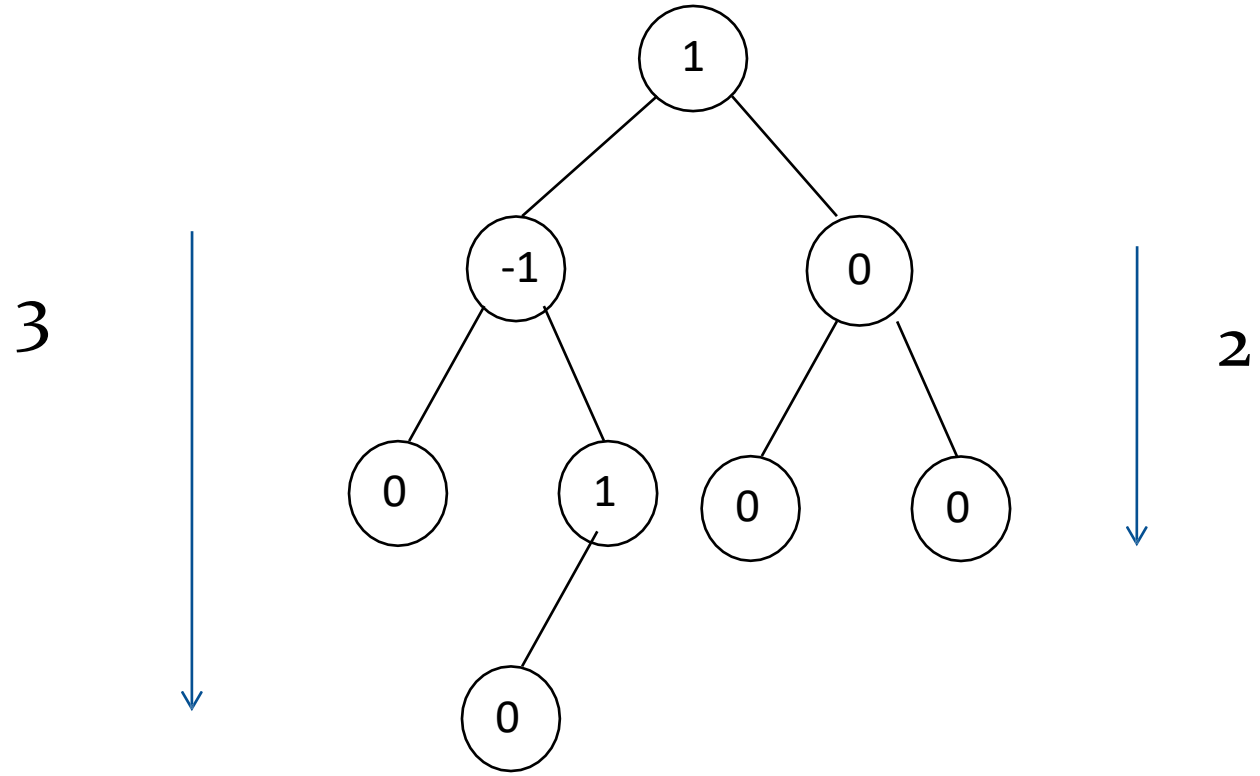
AVL Tree

- A binary tree in which the difference of height of the right and left subtree of any node is less than or equal to 1 is known as AVL Tree.
- Height of left subtree – height of right subtree can be either -1,0,1

AVL Tree

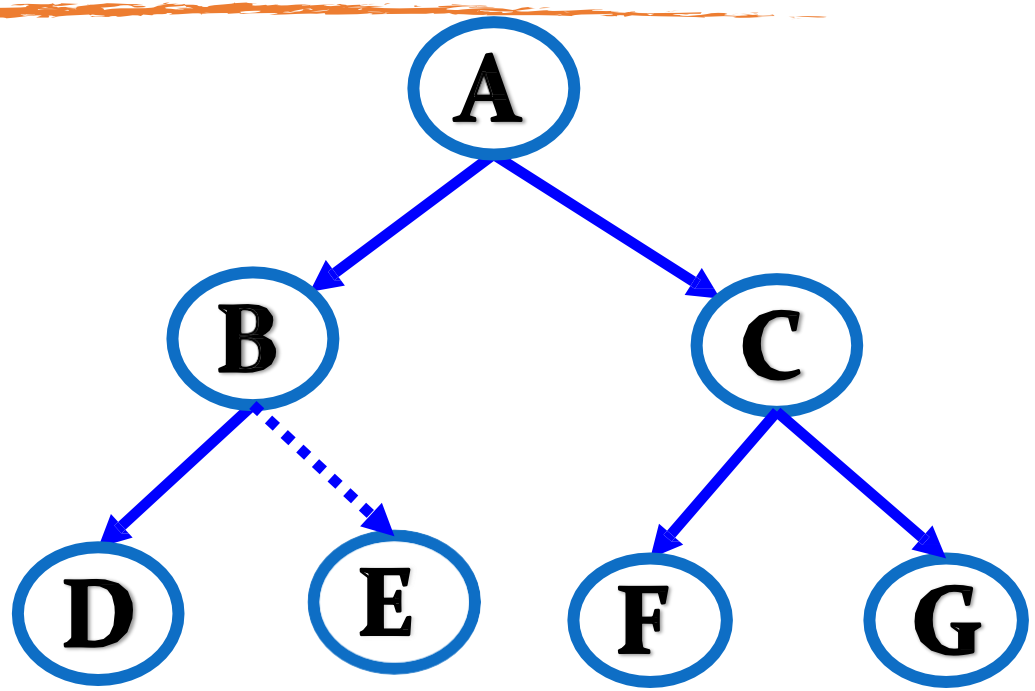


AVL Tree



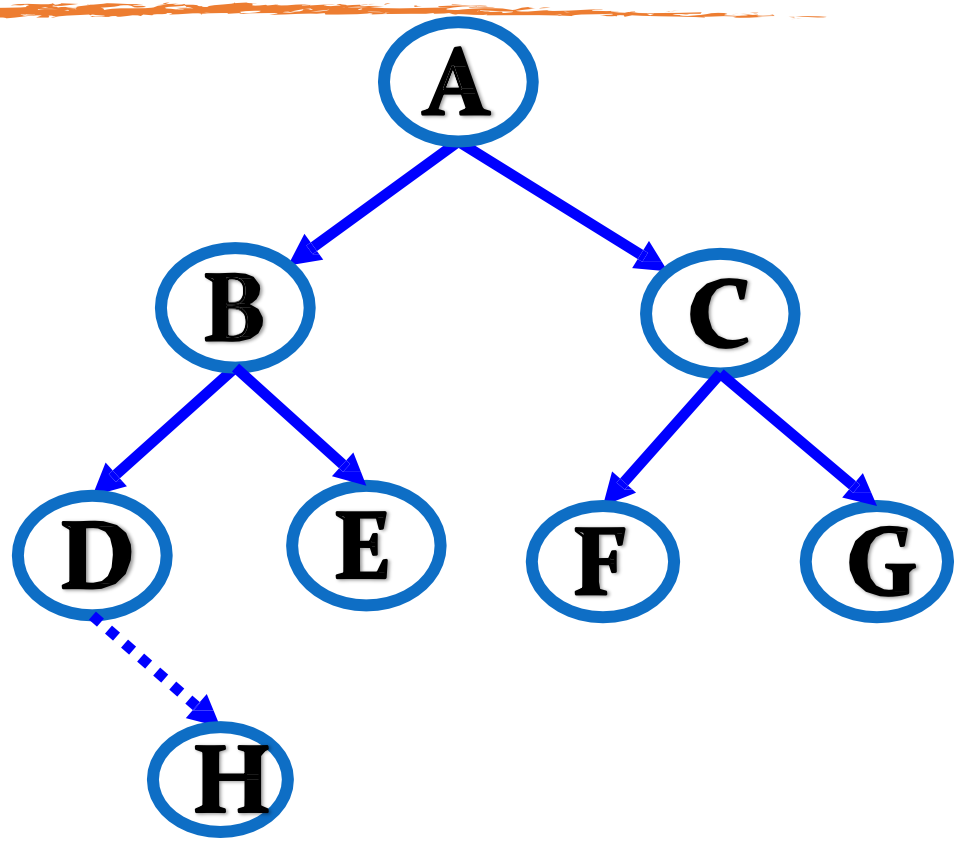
Balanced as LST-RST=1

Insertion in AVL Tree



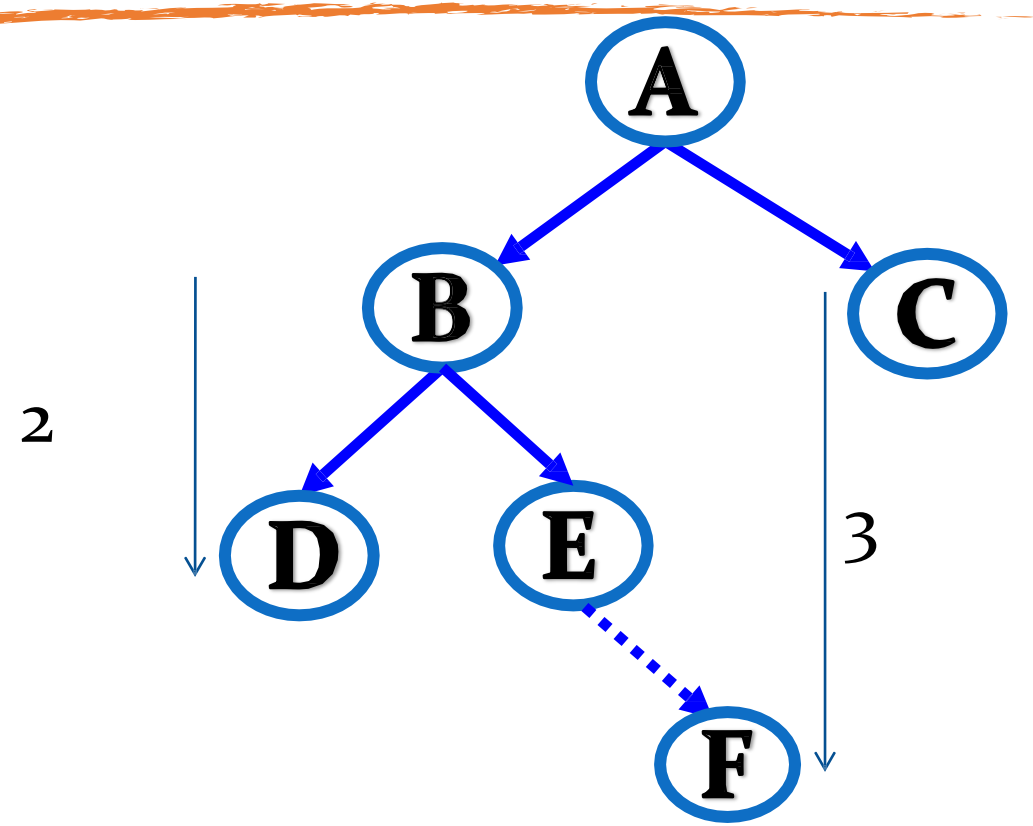
Case 1: the node was either left heavy or right heavy and has become balanced

Insertion in AVL Tree



Case 2: the node was balanced and has now become left or right heavy

Insertion in AVL Tree



Case 3: the node was heavy and the new node has been inserted in the heavy sub tree thus creating an unbalanced sub tree

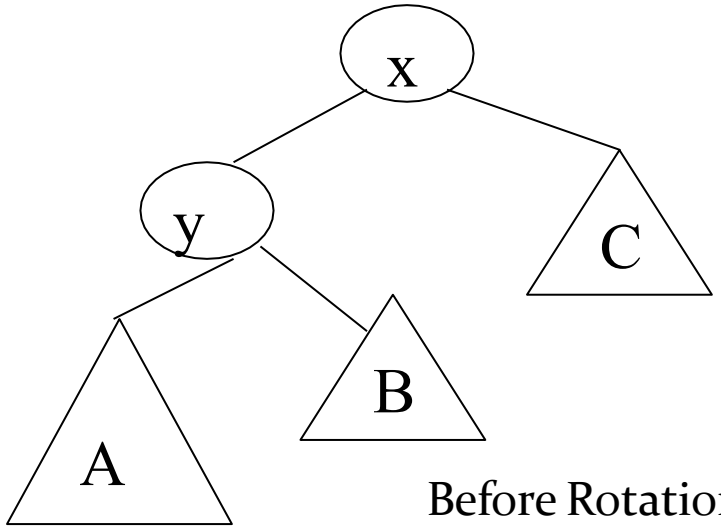
Rebalancing

- When the tree structure changes (e.g., insertion or deletion), we need to transform the tree to restore the AVL tree property.
- This is done using single rotations or double rotations.

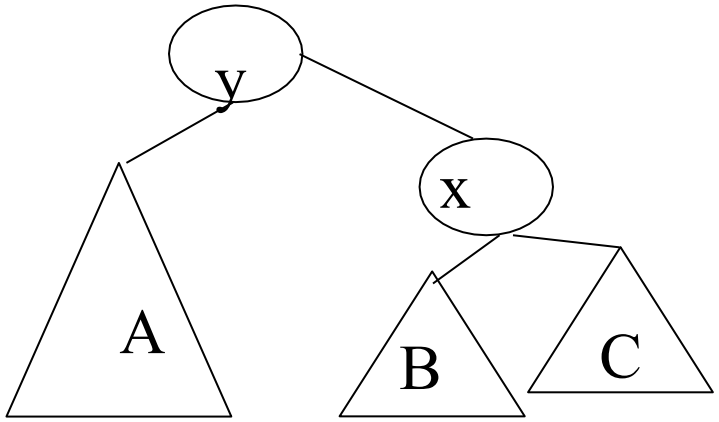
Rotations

- single rotations

e.g. Single Rotation



Before Rotation



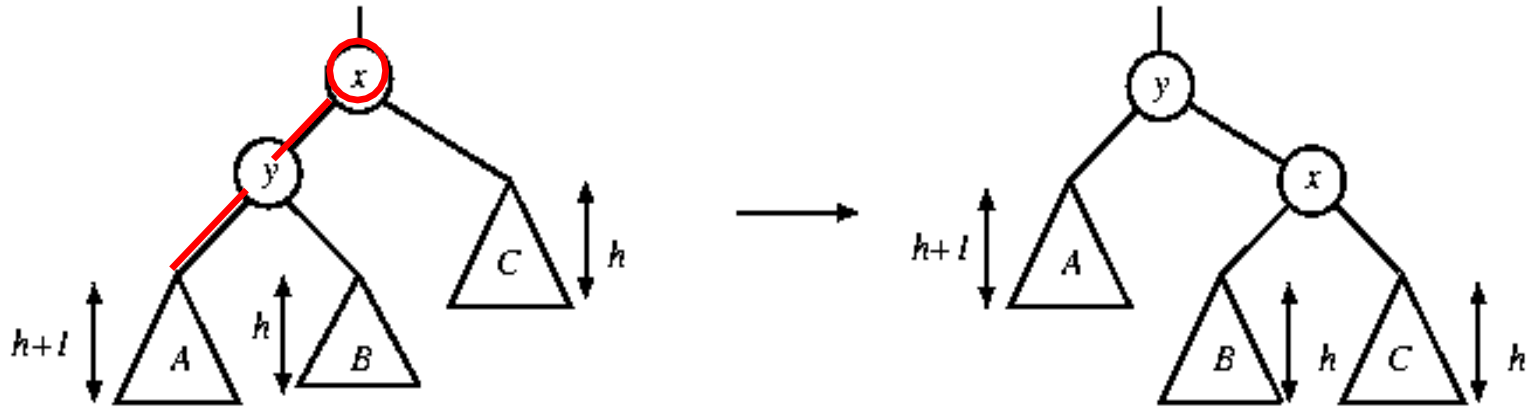
After Rotation

Rotations

- Since an insertion/deletion involves adding/deleting a single node, this can only increase/decrease the height of some subtree by 1
- Thus, if the AVL tree property is violated at a node x, it means that the heights of left(x) and right(x) differ by exactly 2.
- Rotations will be applied to x to restore the AVL tree property.

Single Rotation

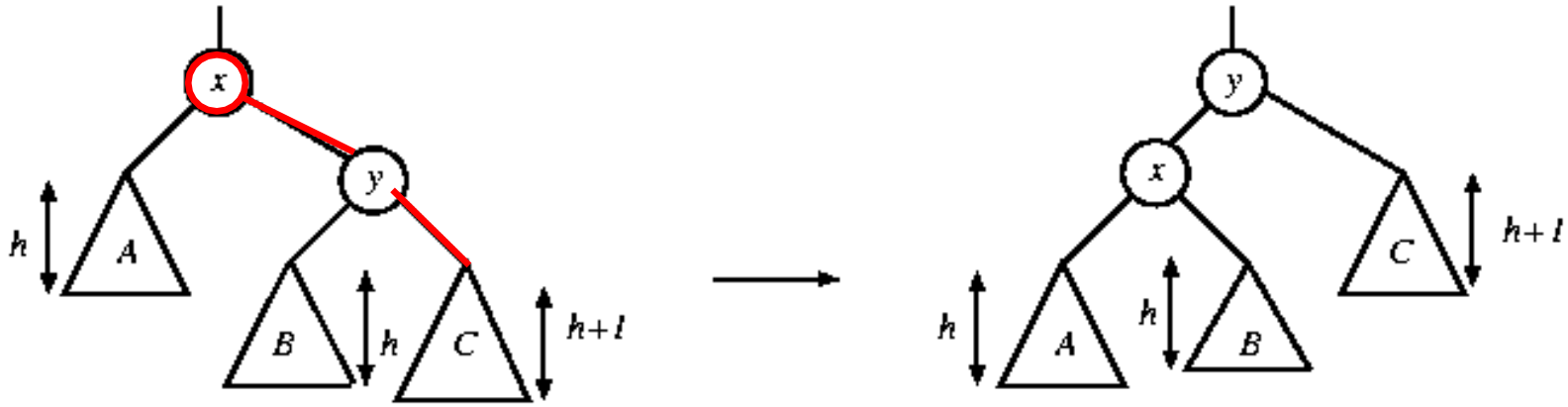
The new item is inserted in the subtree A.
The AVL-property is violated atx
height of left(x) is h+2
height of right(x) is h.



Rotate with left child

Single Rotation

The new item is inserted in the subtreeC.
The AVL-property is violated atx.

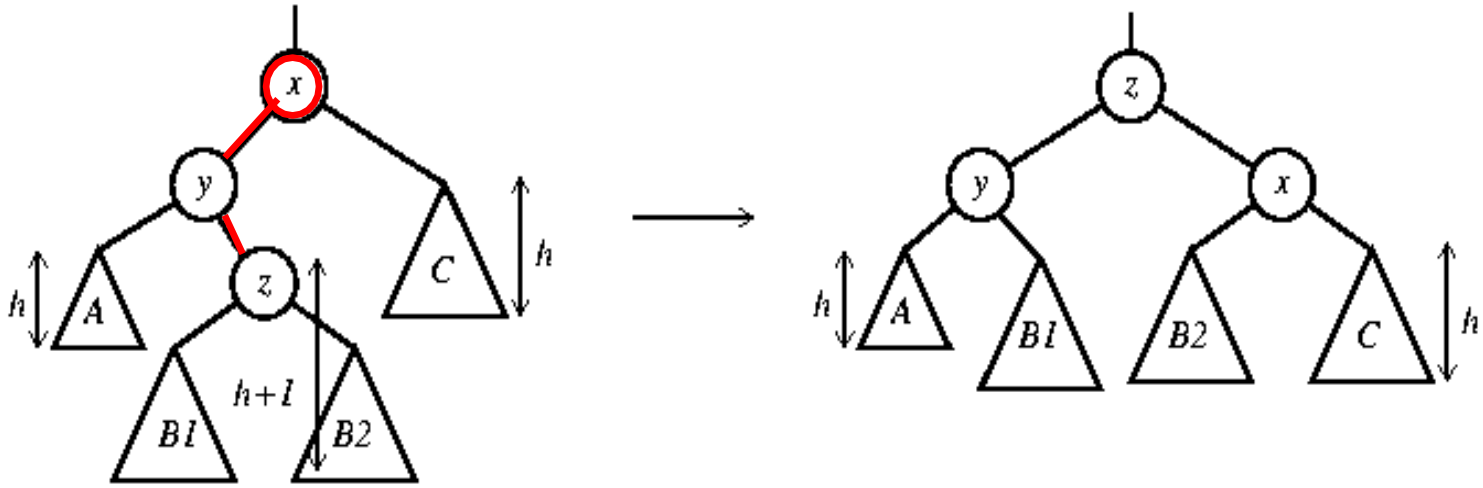


Rotate with right child

Single rotation takes O(1) time.
Insertion takes O(log N) time.

Double Rotation

The new key is inserted in the subtree B1 or B2.
The AVL-property is violated atx.
x-y-z forms a zig-zagshape

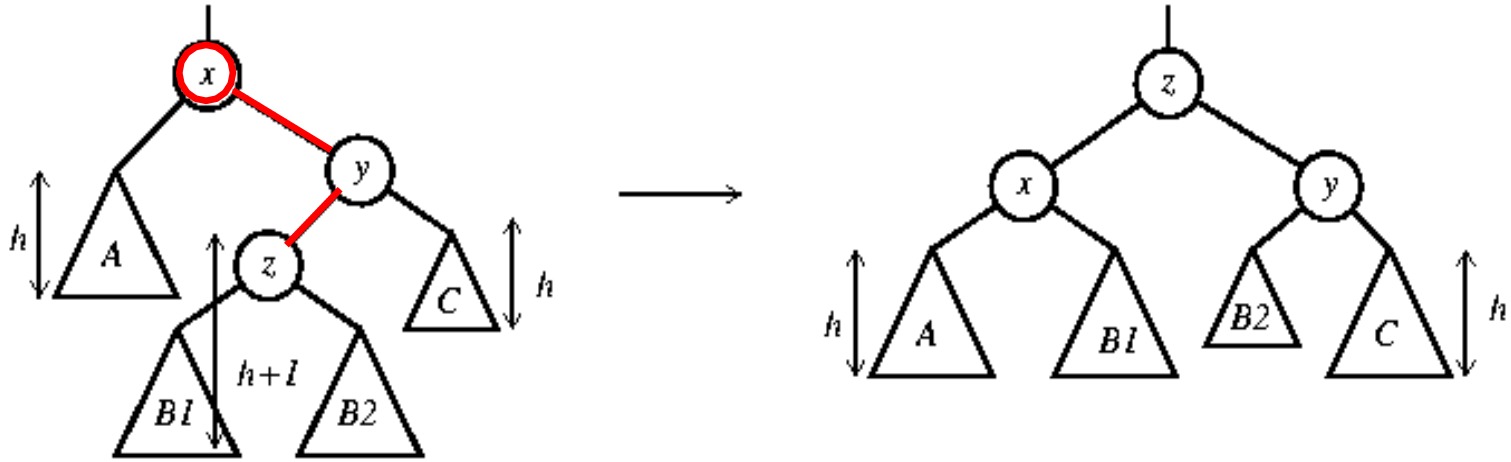


Double rotate with left child

also called left-rightrotate

Double Rotation

The new key is inserted in the subtree B1 or B2. The AVL-property is violated atx.

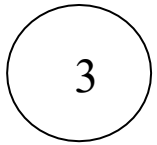


Double rotate with right child

also called right-leftrotate

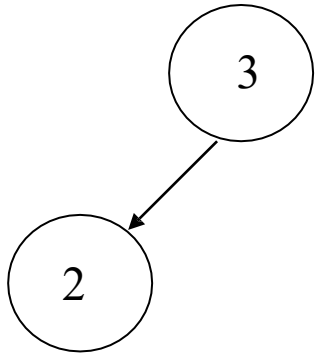
Example

- Insert 3,2,1,4,5,6,7



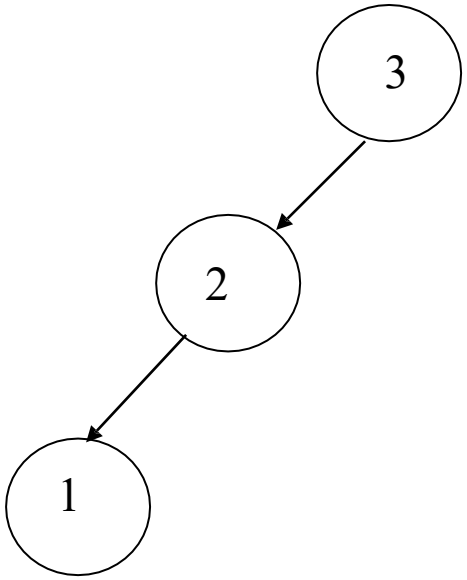
Example

- Insert 3,2,1,4,5,6,7



Example

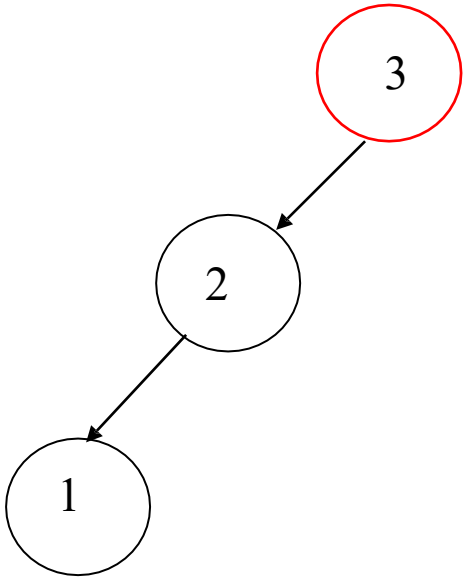
- Insert 3,2,1,4,5,6,7



Example

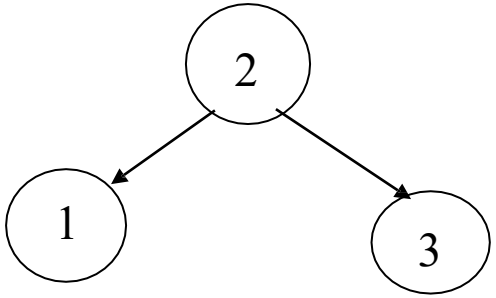
- Insert 3,2,1,4,5,6,7

Single rotation



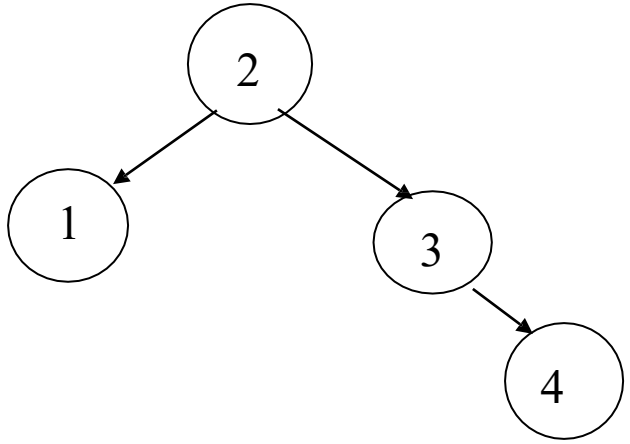
Example

- Insert 3,2,1,4,5,6,7



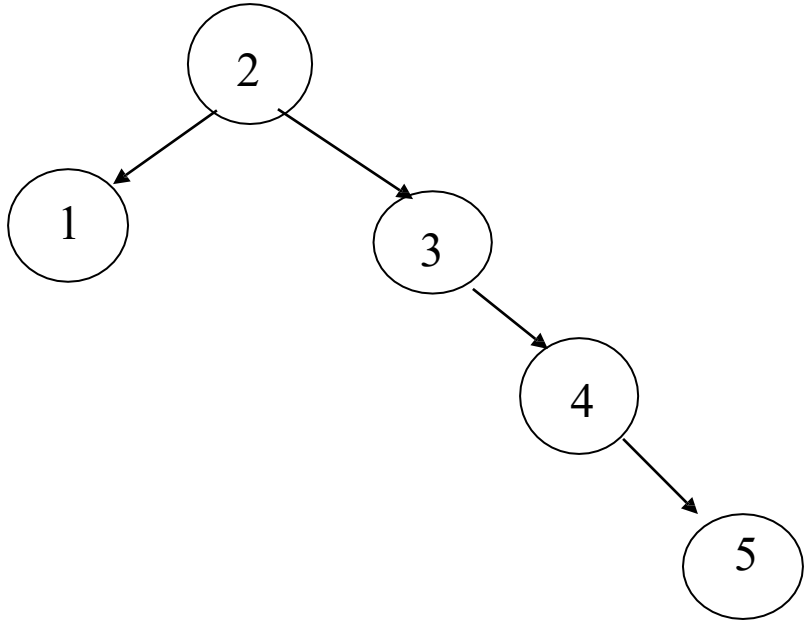
Example

- Insert 3,2,1,4,5,6,7



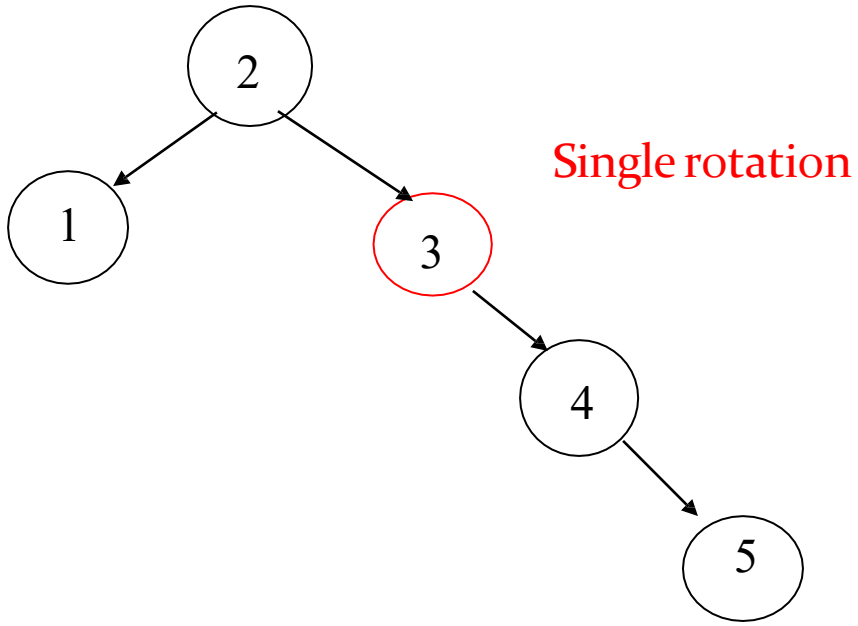
Example

- Insert 3,2,1,4,5,6,7



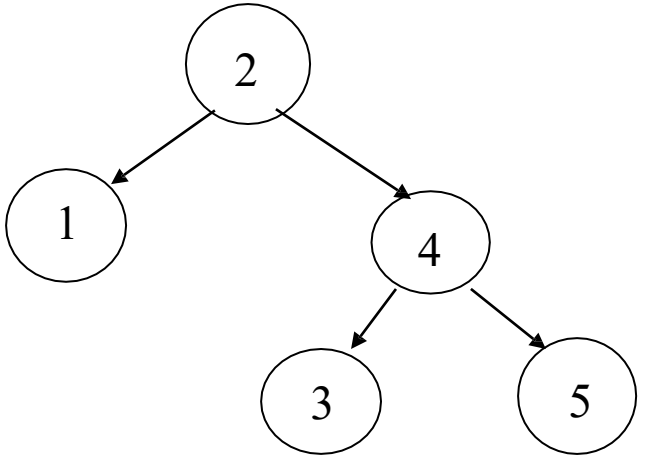
Example

- Insert 3,2,1,4,5,6,7



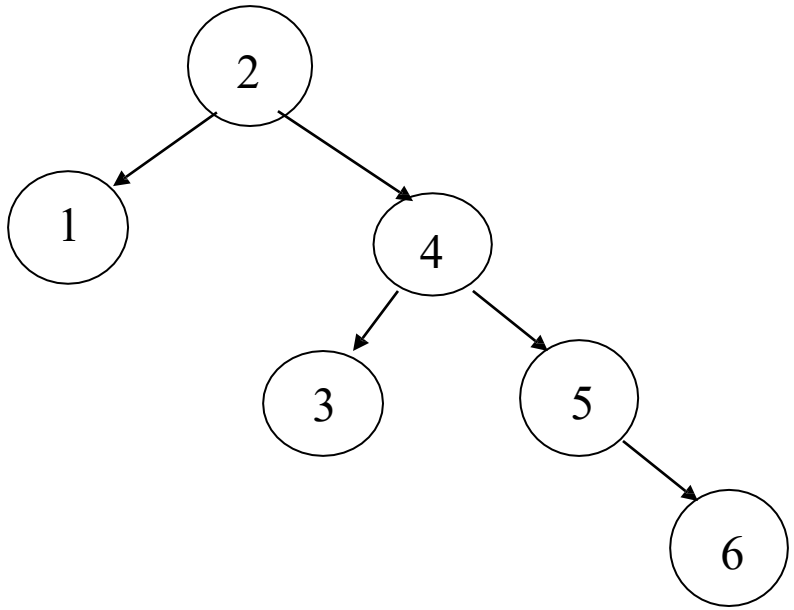
Example

- Insert 3,2,1,4,5,6,7



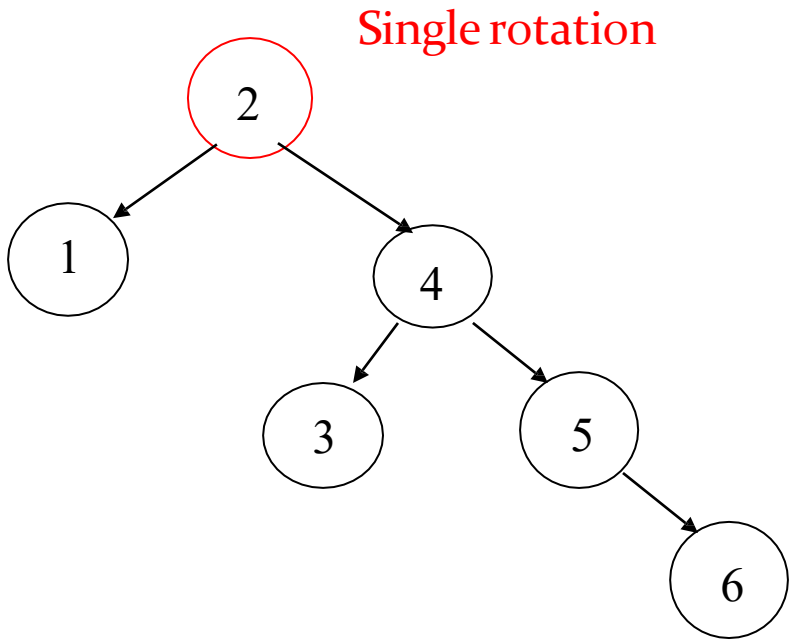
Example

- Insert 3,2,1,4,5,6,7



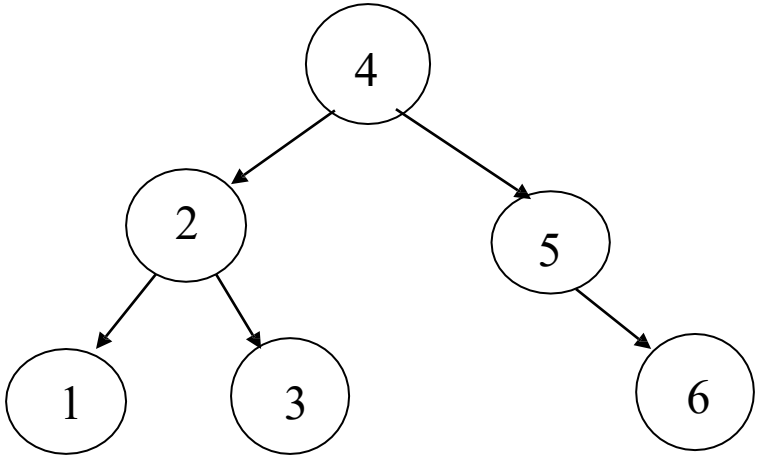
Example

- Insert 3,2,1,4,5,6,7



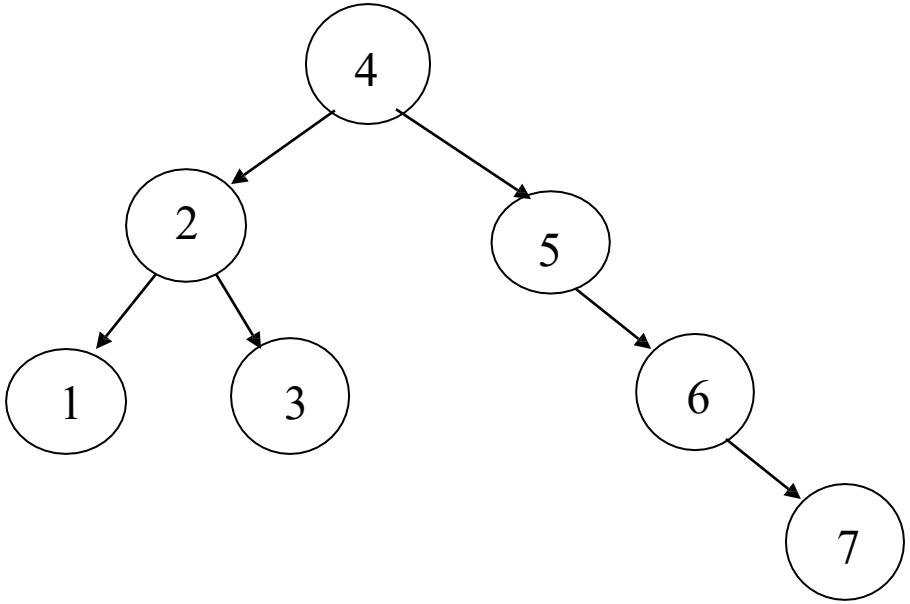
Example

- Insert 3,2,1,4,5,6,7



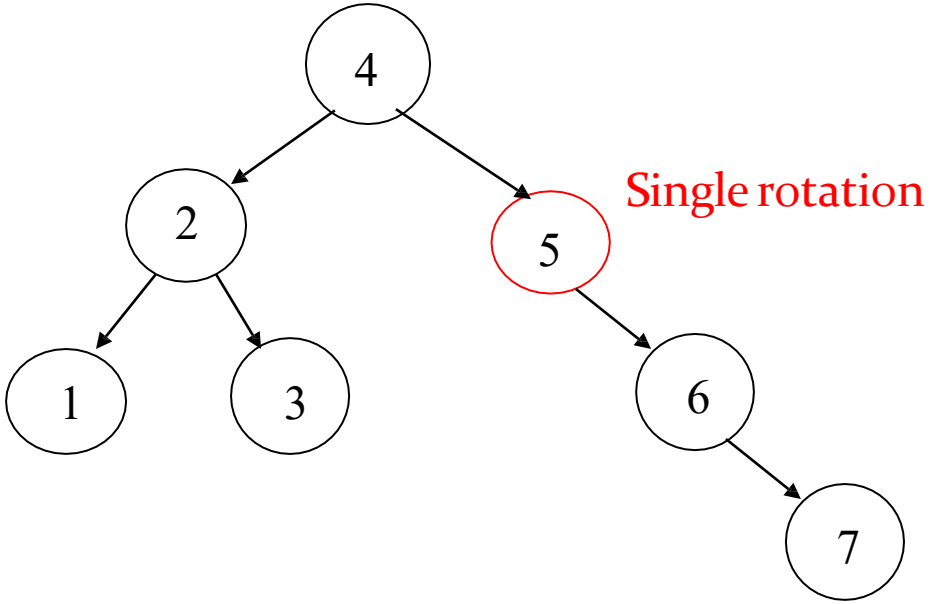
Example

- Insert 3,2,1,4,5,6,7



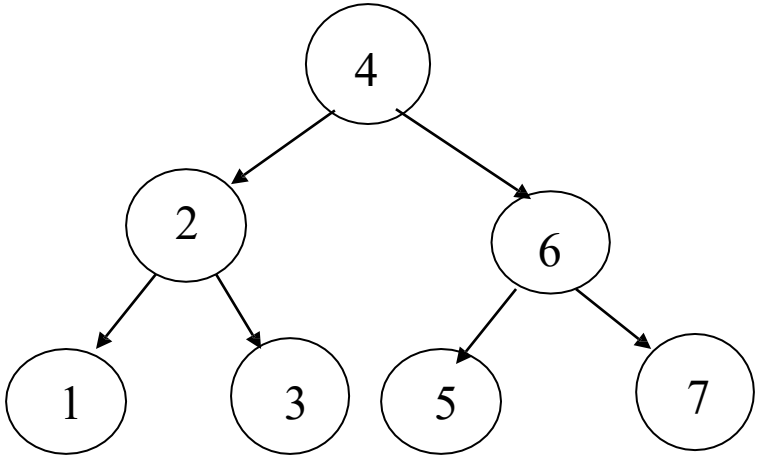
Example

- Insert 3,2,1,4,5,6,7



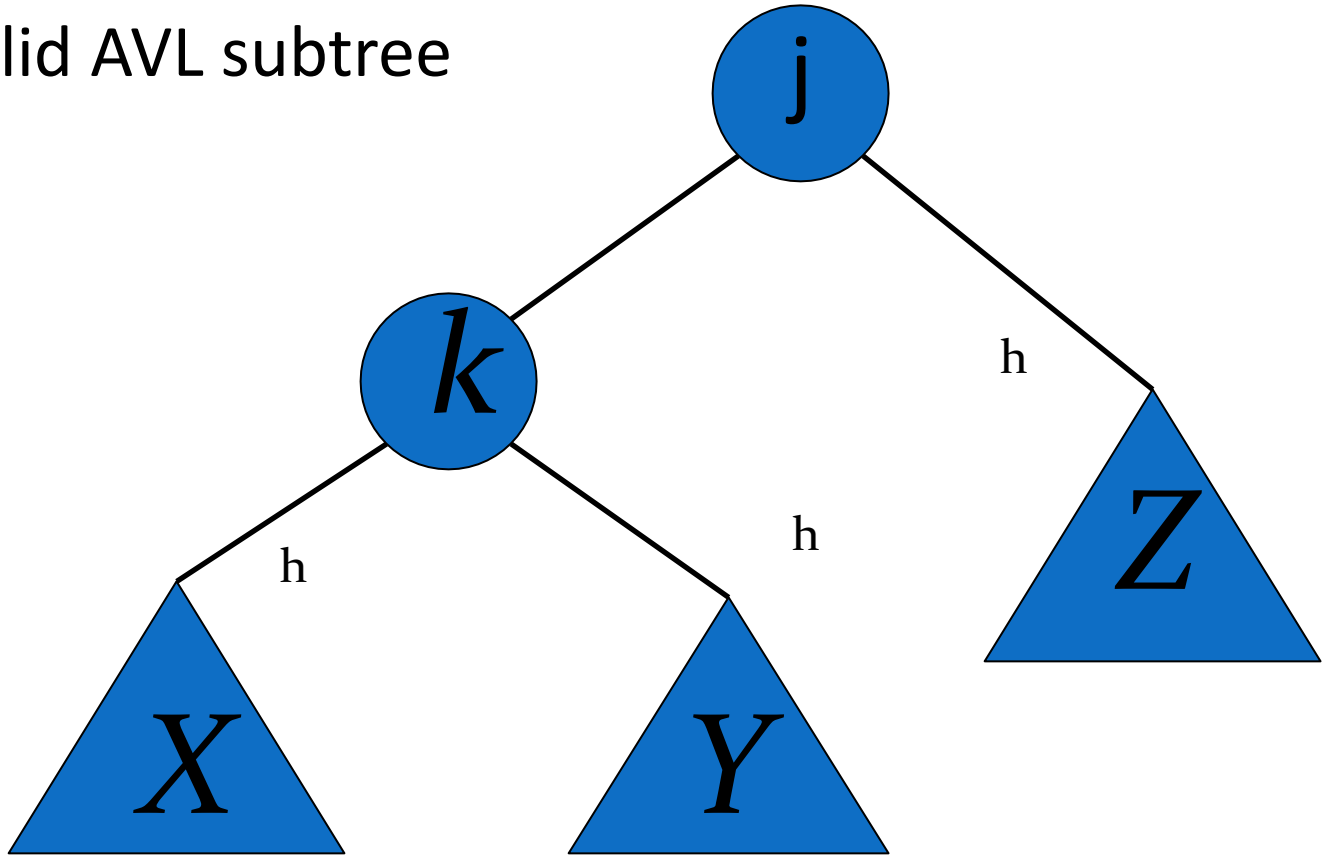
Example

- Insert 3,2,1,4,5,6,7



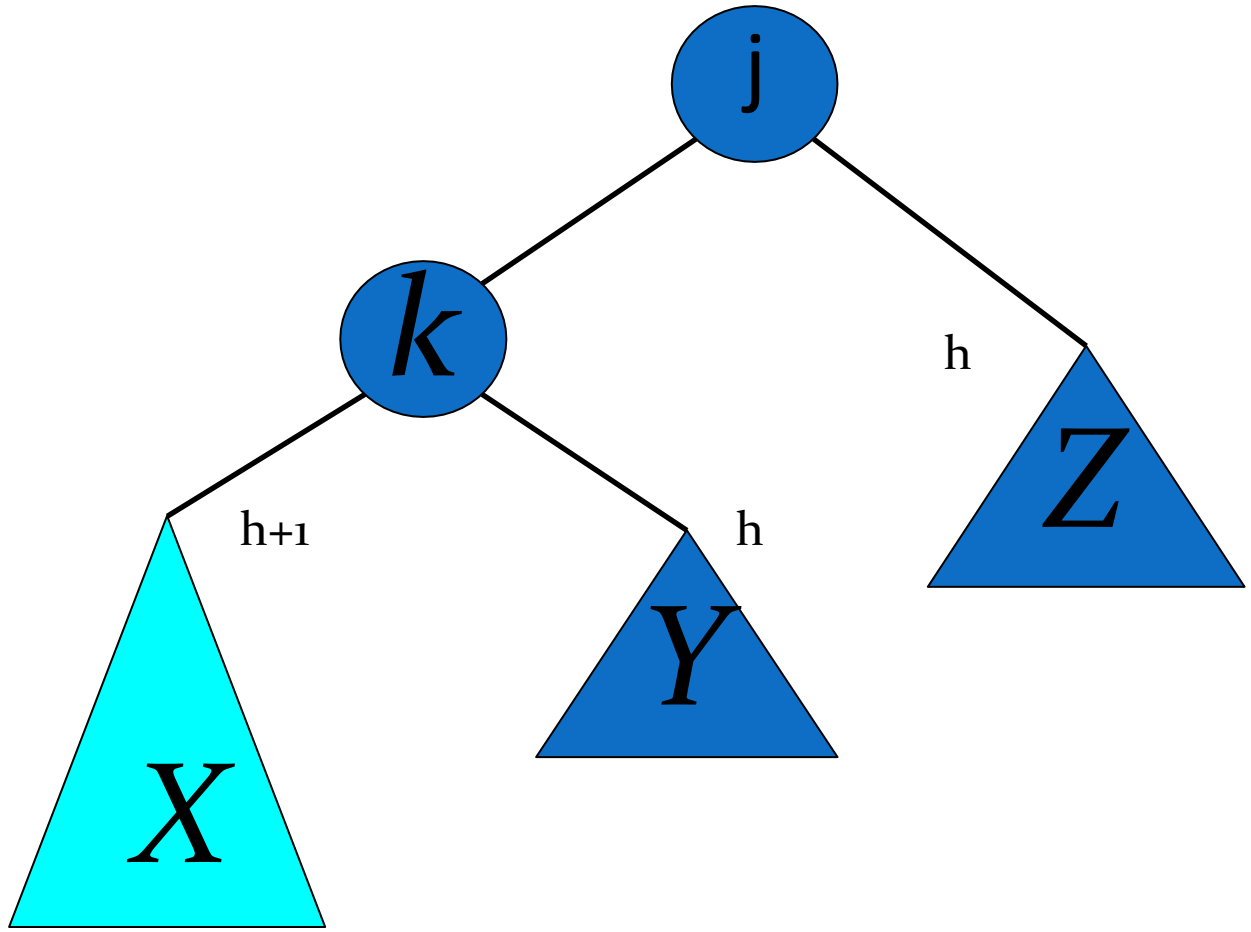
AVL Insertion: Outside Case

- Consider a valid AVL subtree

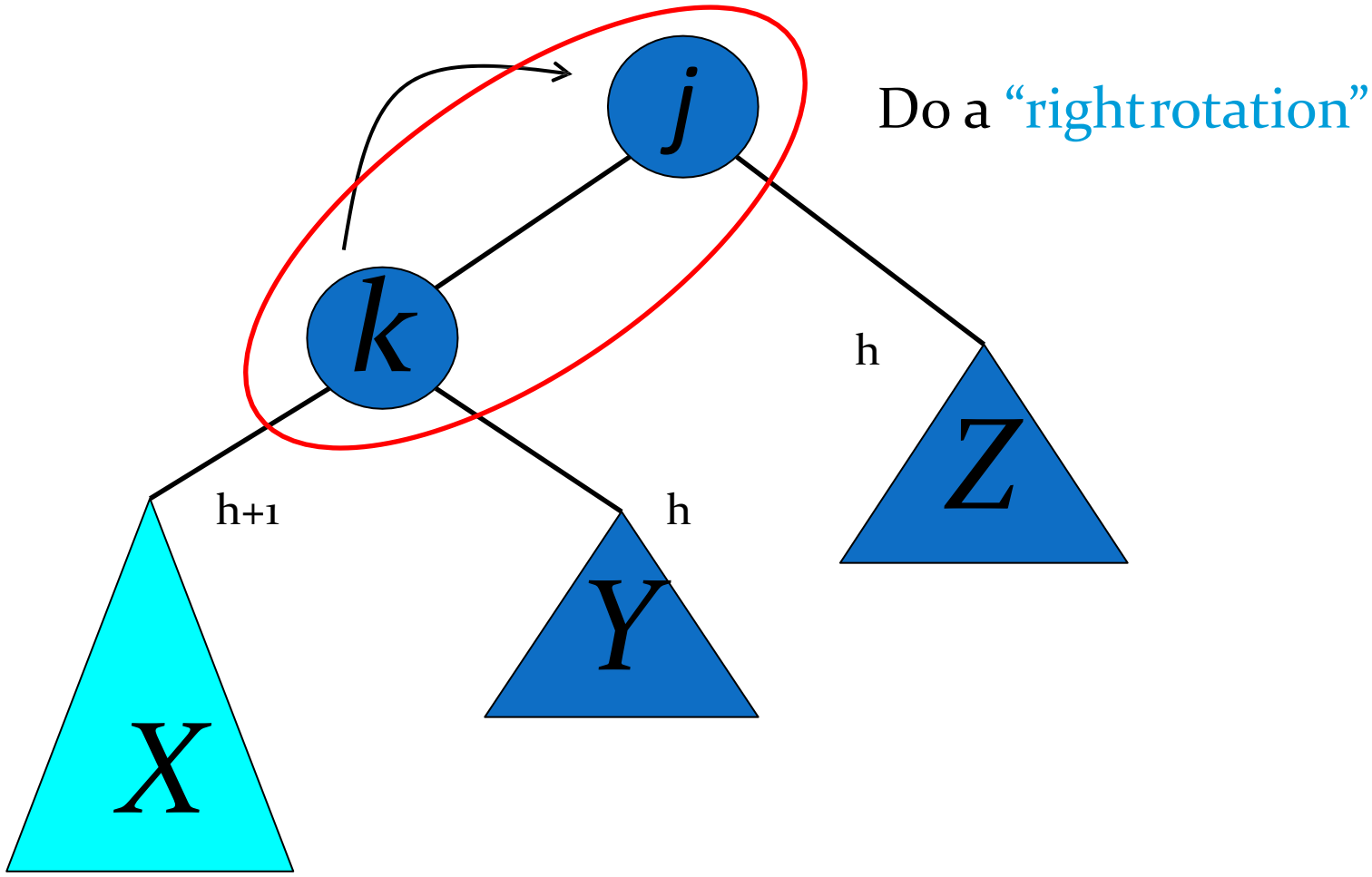


AVL Insertion: Outside Case

- Inserting into X destroys the AVL property at node j

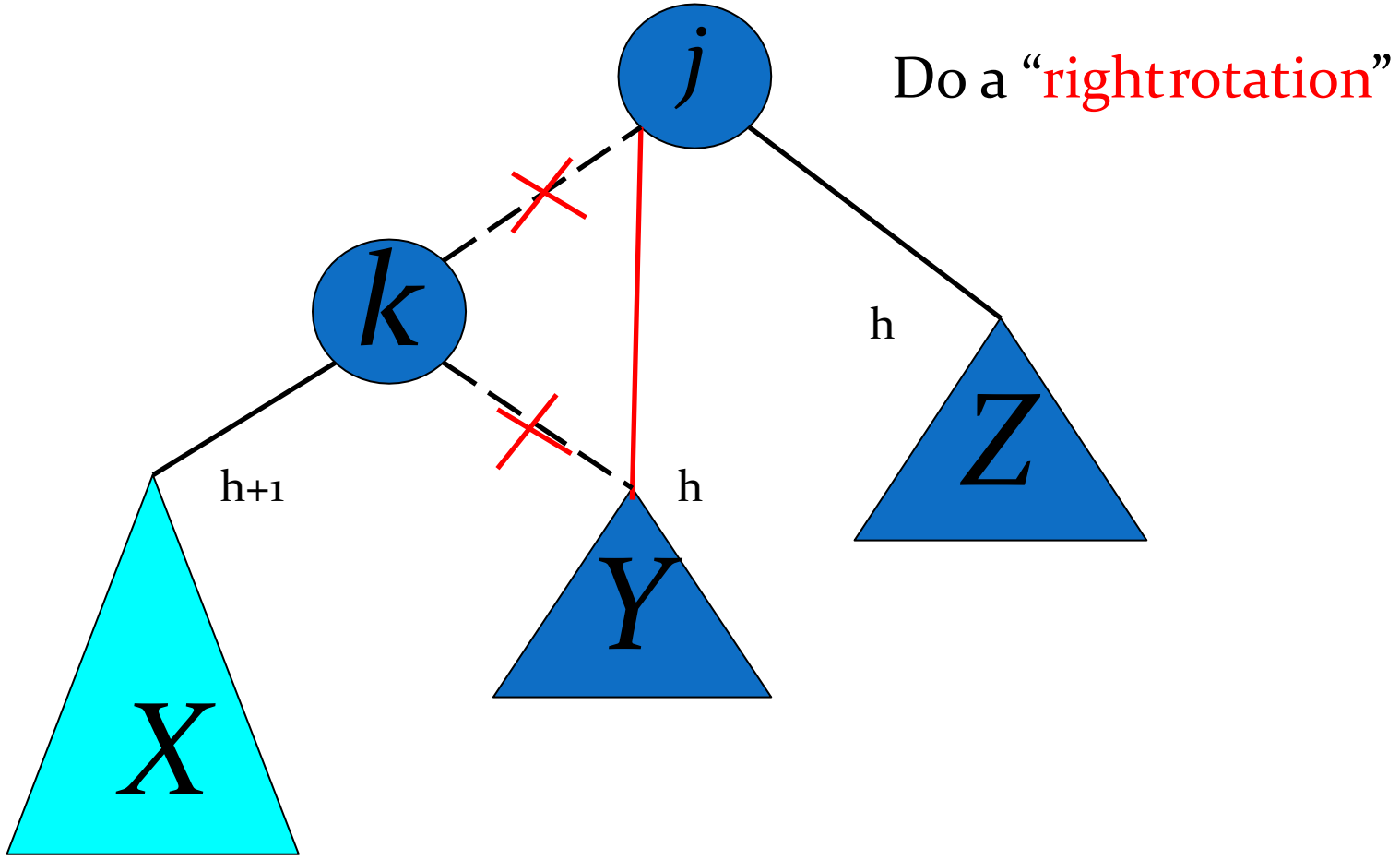


AVL Insertion: Outside Case

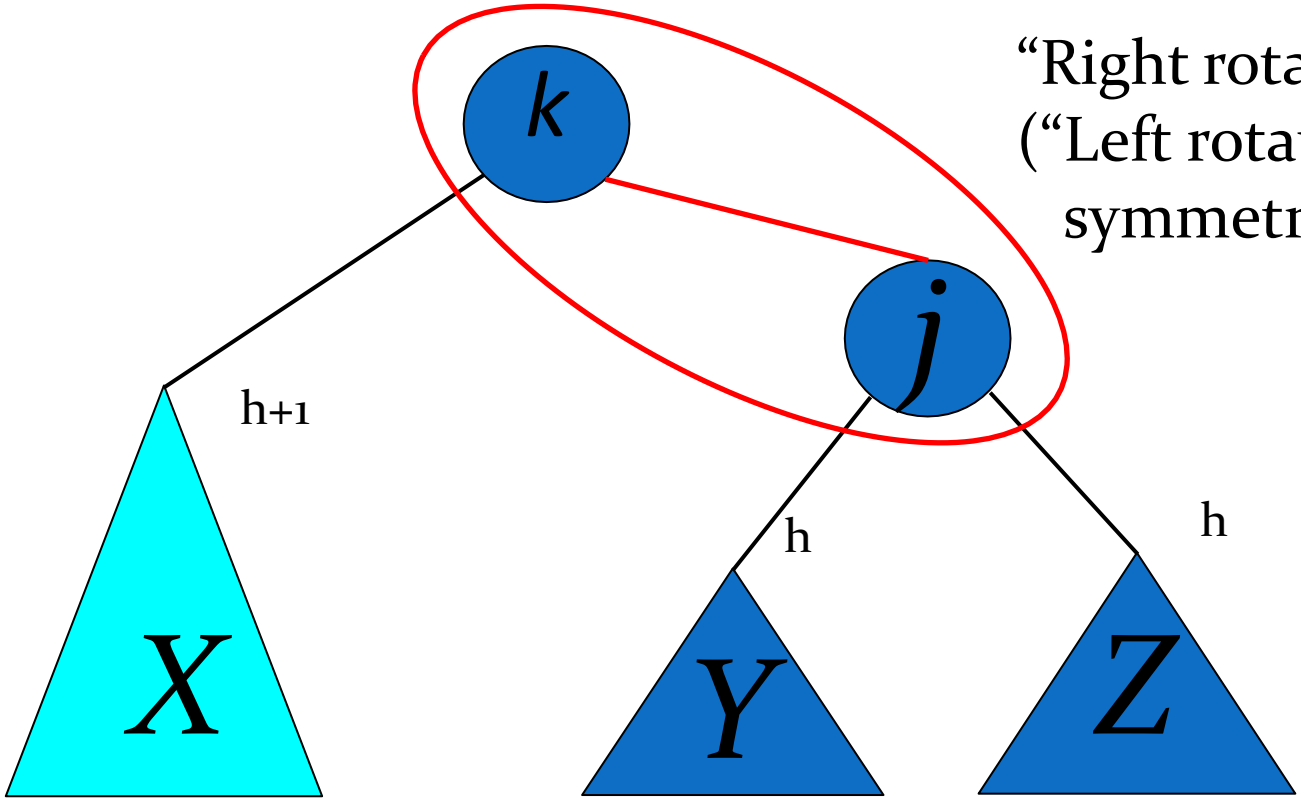


Do a "rightrotation"

Single right rotation



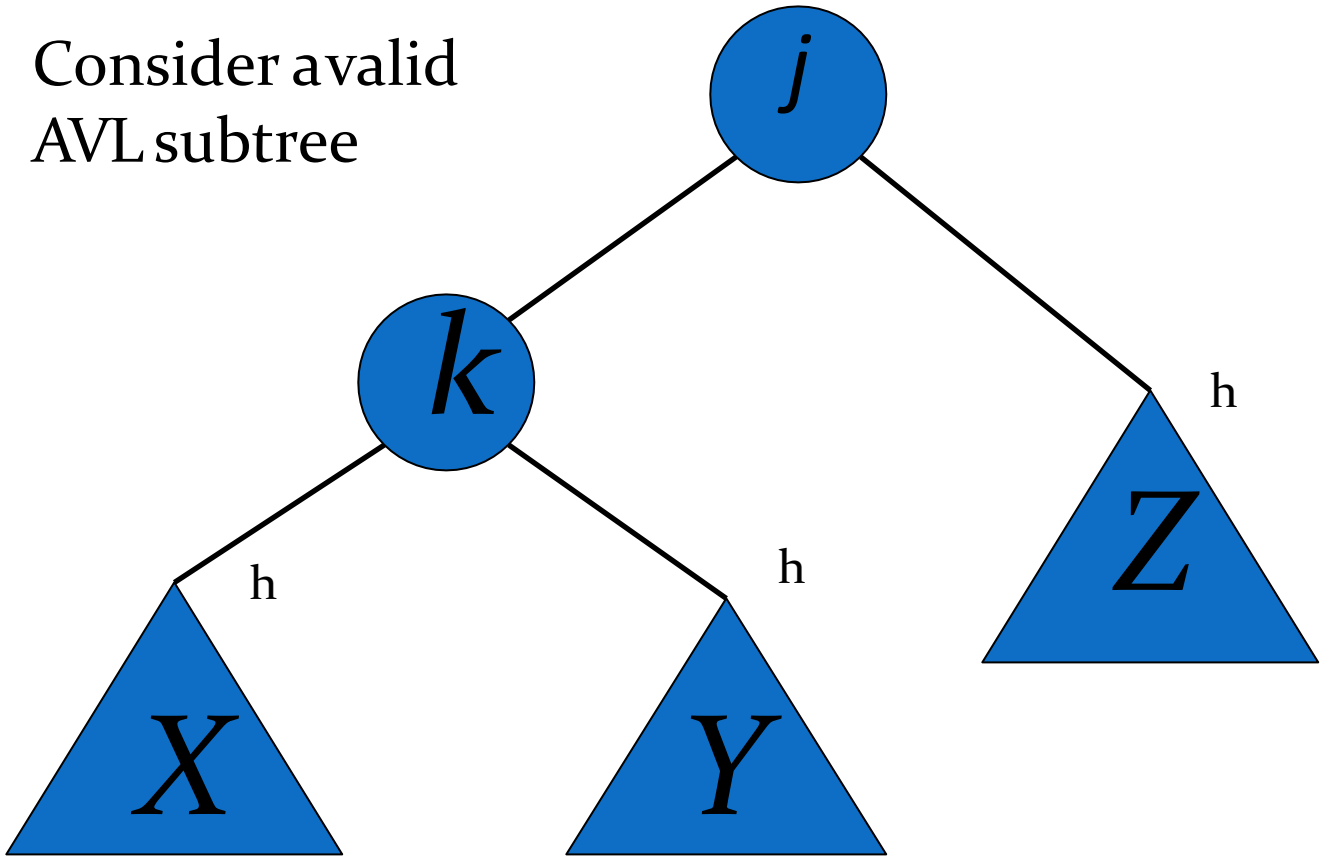
Outside Case Completed



"Right rotation" done!
("Left rotation" is mirror symmetric)

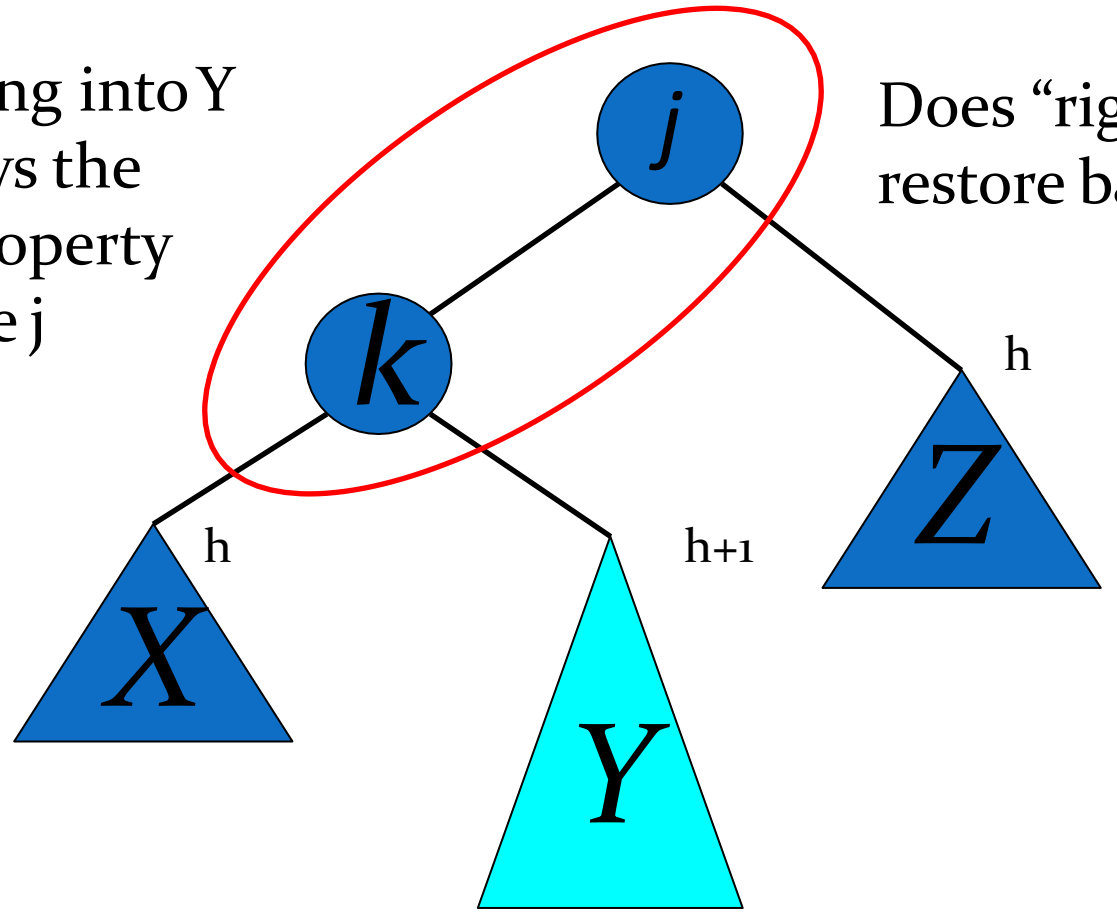
AVL property has been restored!

AVL Insertion: Inside Case



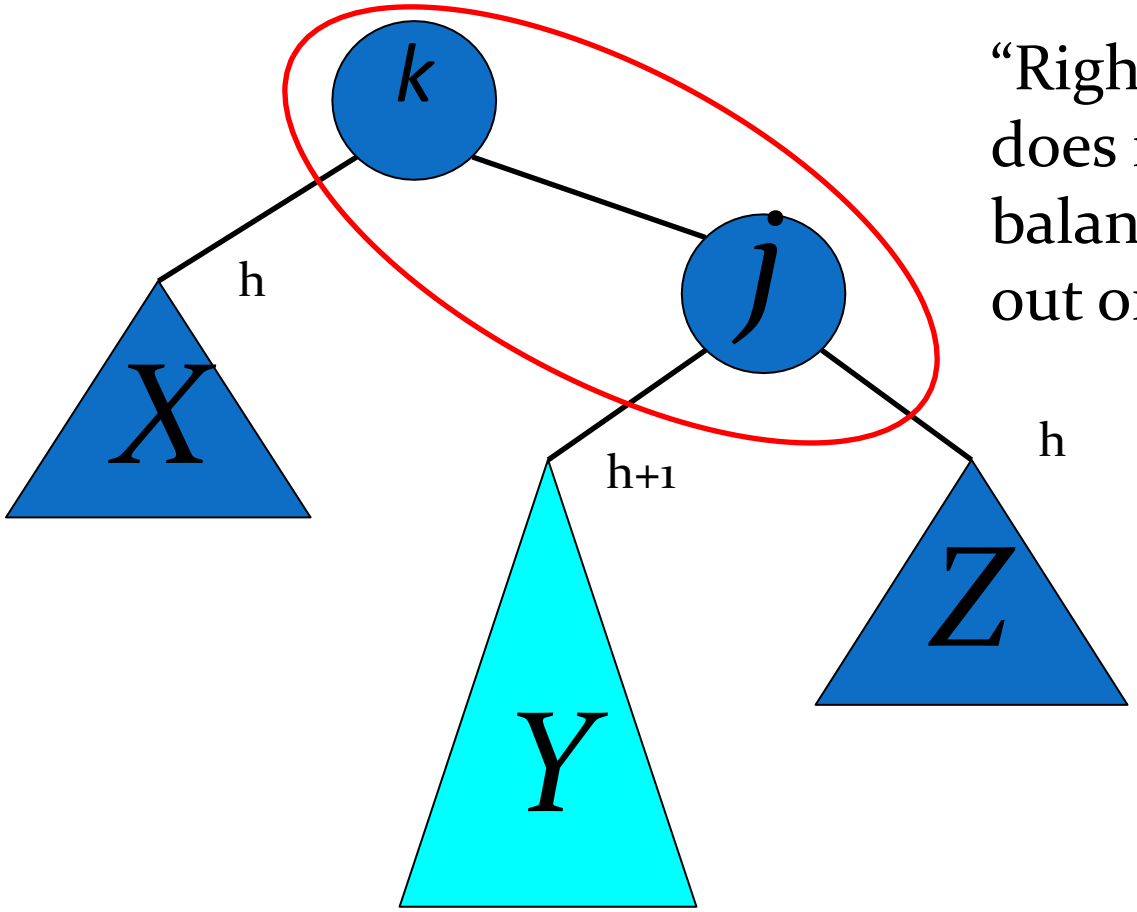
AVL Insertion: Inside Case

Inserting into Y destroys the AVL property at node j



Does "right rotation" restore balance?

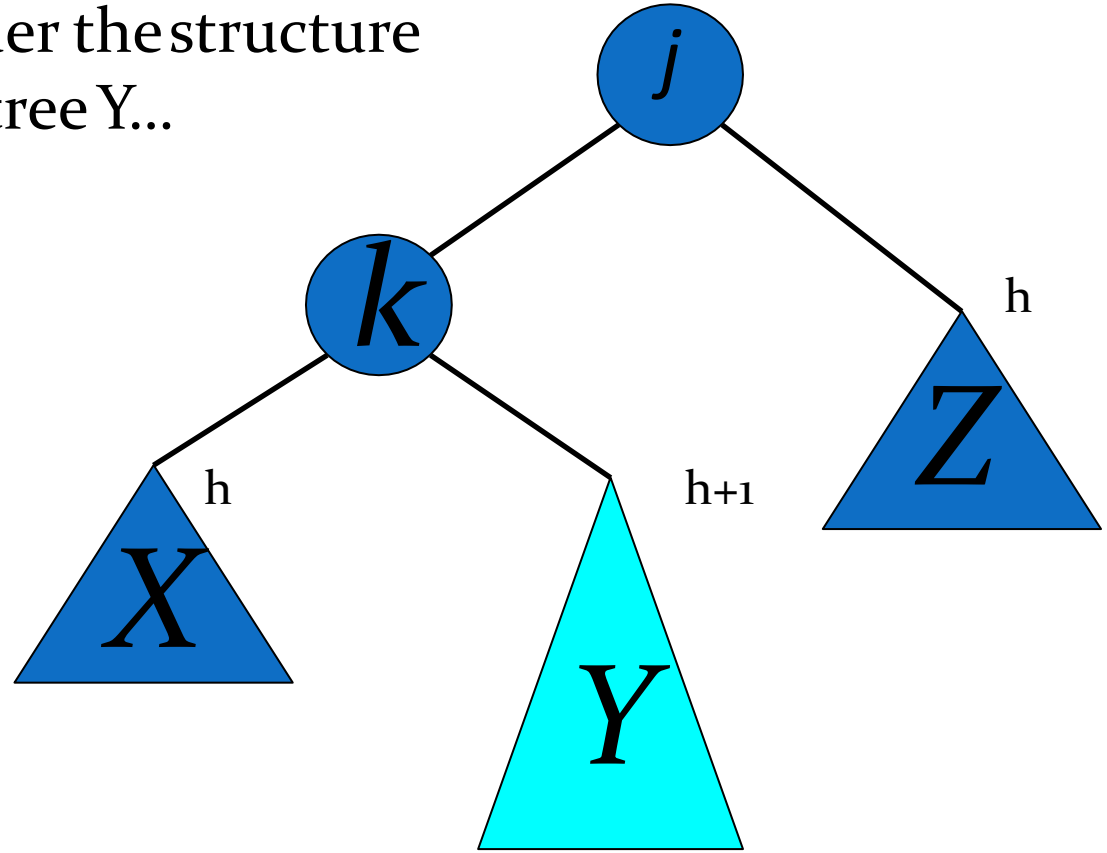
AVL Insertion: Inside Case



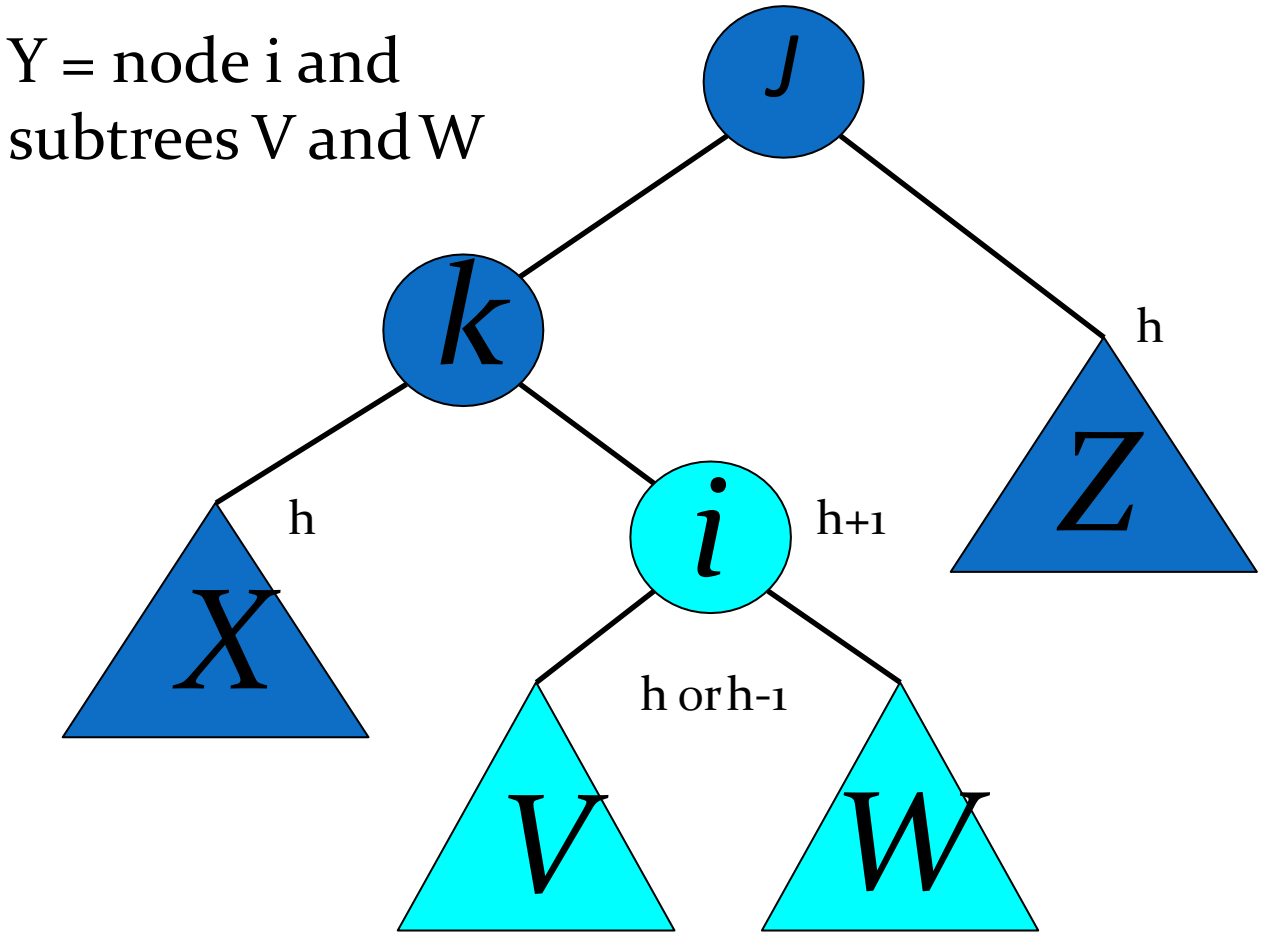
"Right rotation" does not restore balance… now k is out of balance

AVL Insertion: Inside Case

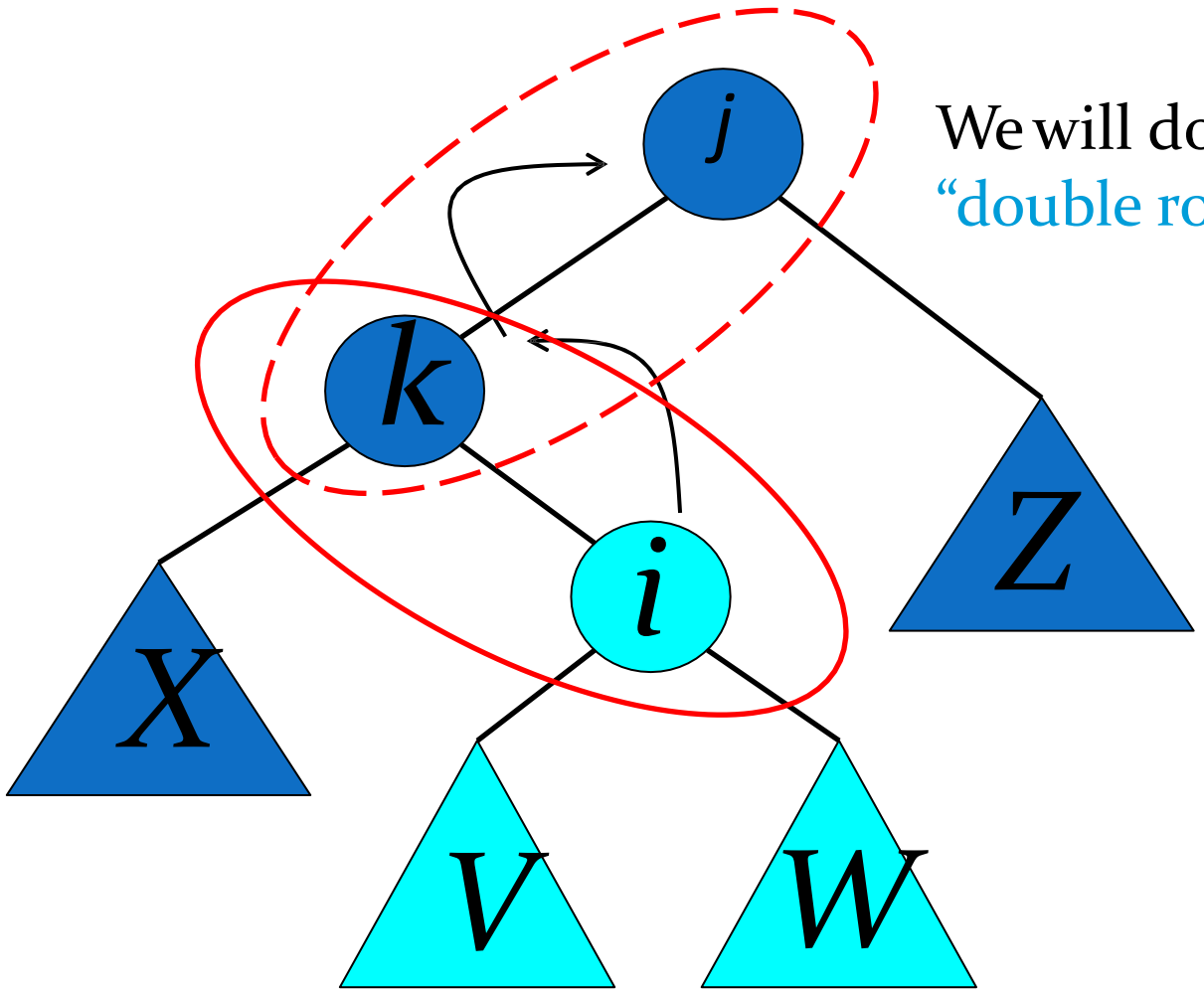
Consider the structure of subtree Y...



AVL Insertion: Inside Case

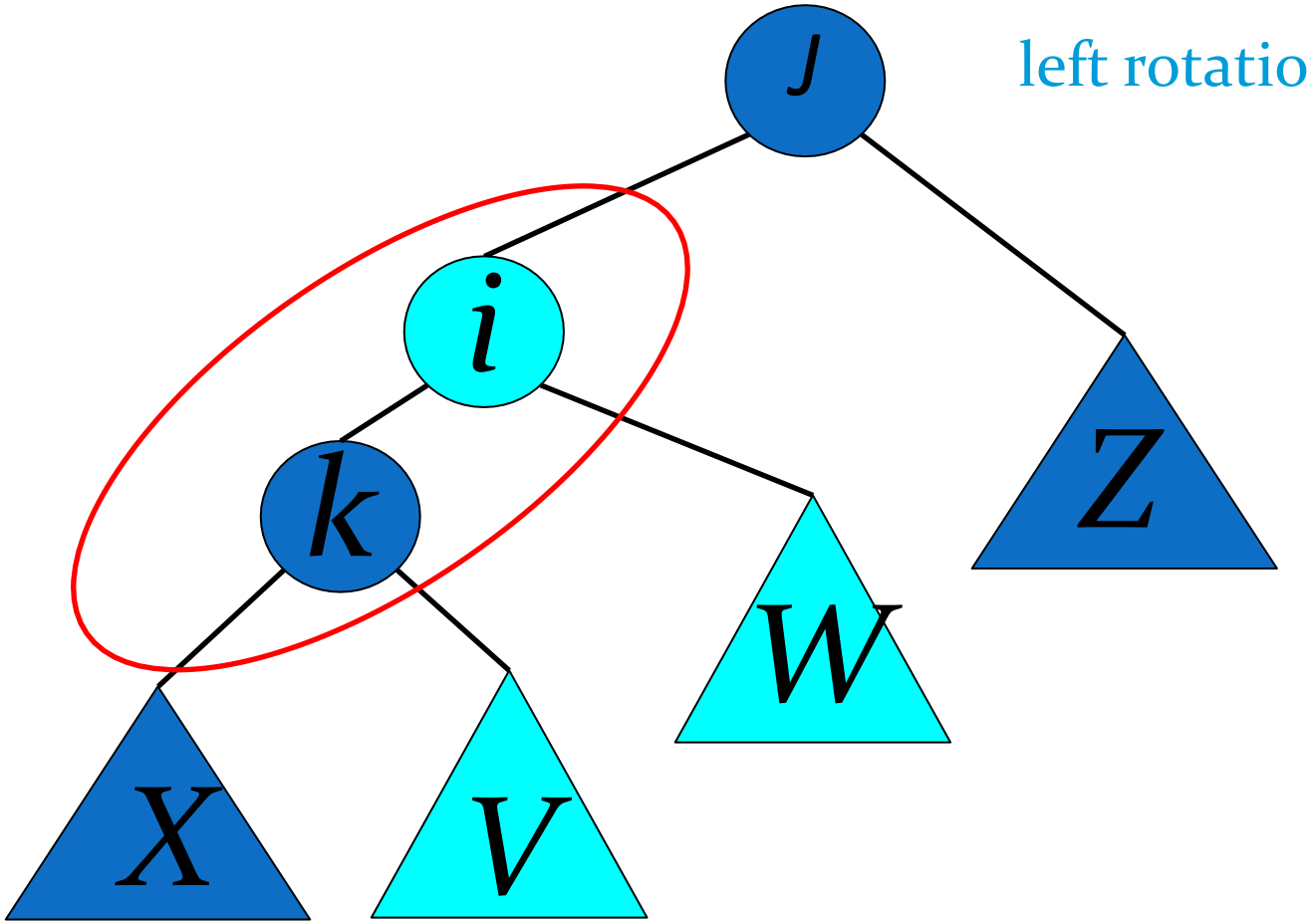


AVL Insertion: Inside Case



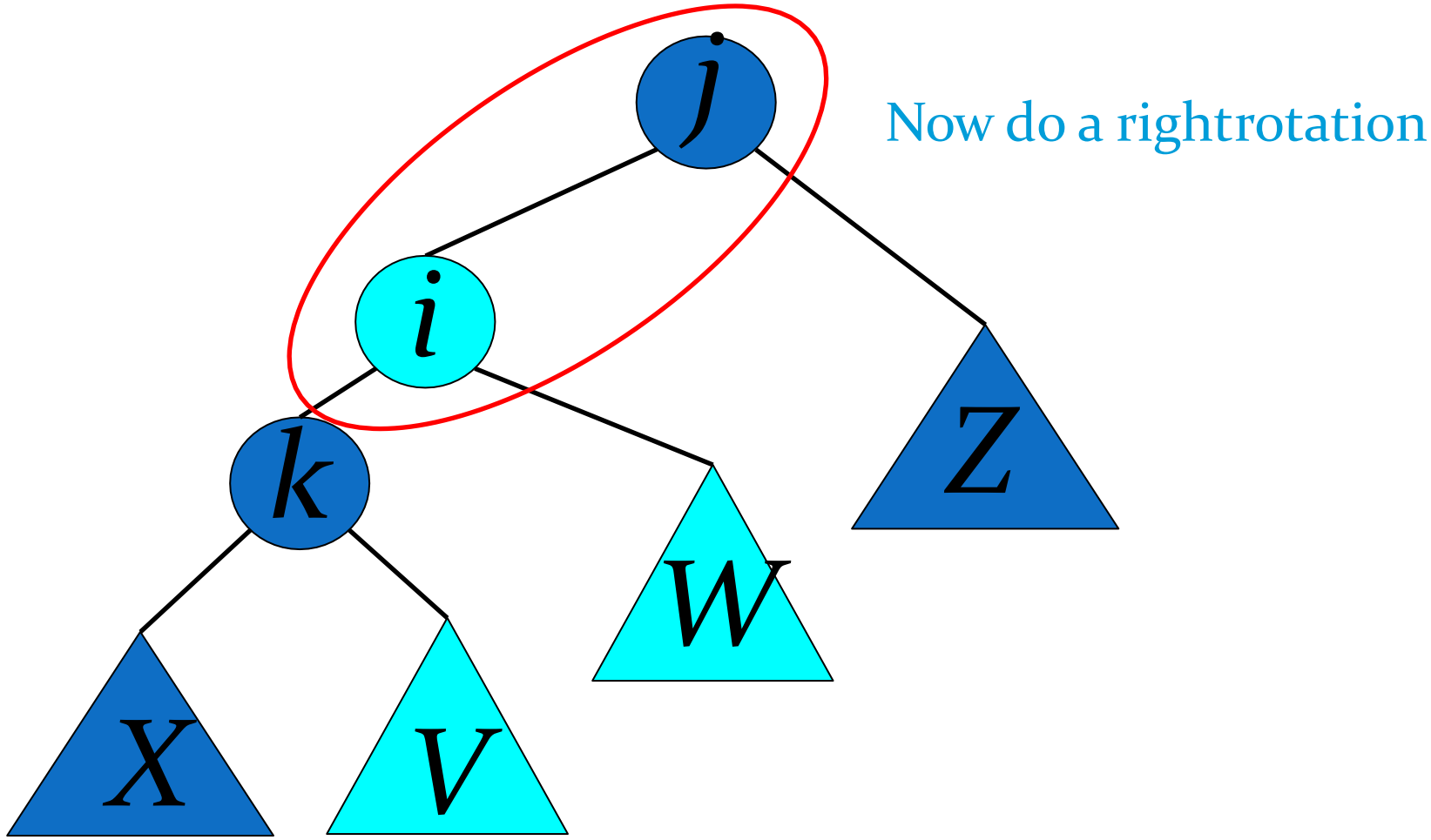
We will do a left-right "double rotation" . . .

Double rotation : first rotation



left rotation complete

Double rotation : second rotation



Now do a rightrotation

Double rotation : second rotation