Data Structures & Algorithms

AVL Tree

Height Balanced Trees

- A height balanced tree is one in which the difference in the height of the two subtrees for any node is less than or equal to some specified amount.

- ✓ One type of height balanced tree is the AVL tree named after its originators (Adelson-Velskii & Landis).
- ✓ In this tree *the height difference may be no more than 1.*

AVL Tree

- ✓ In the AVL trees, searches always stay close to the theoretical minimum of O(log n) and never degenerate to O(n).

- ✓ The tradeoff for search efficiency is increased time and complexity for insertion and deletion from these trees since each time the tree is changed it may go out of balance and have to be rebalanced.

Balance Factor

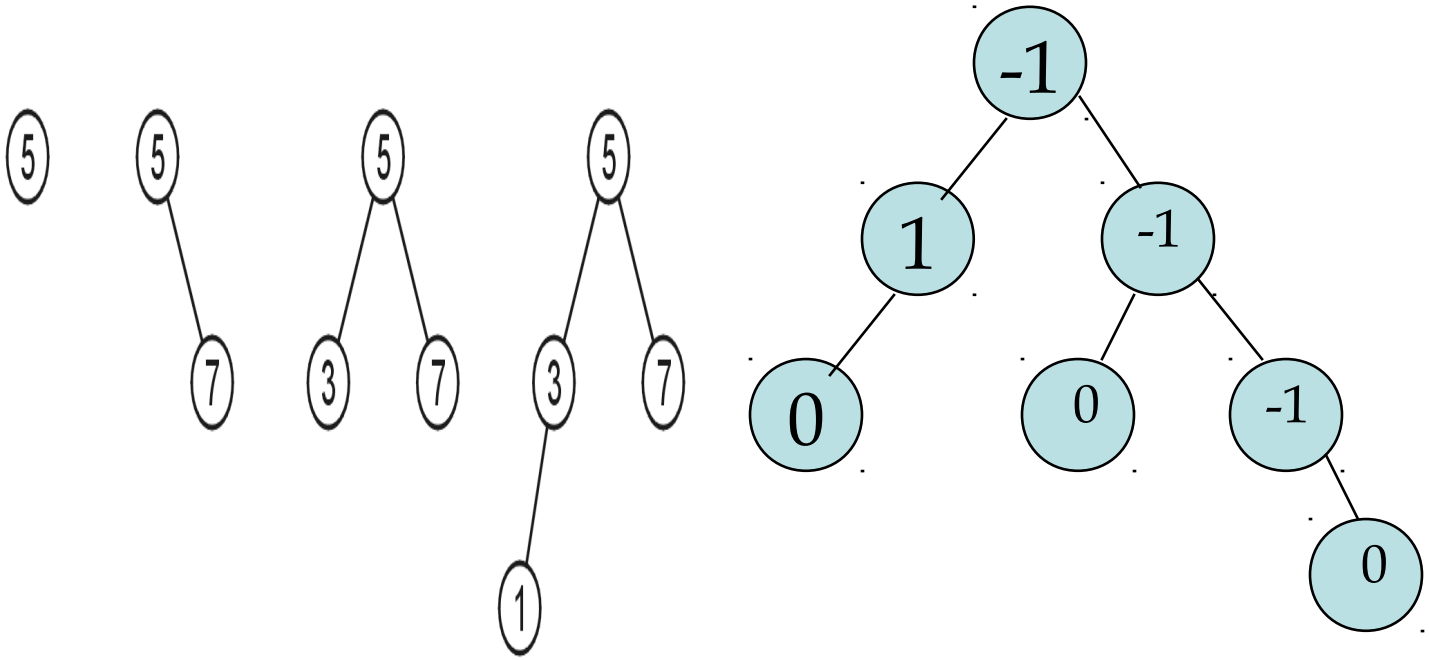
- To implement an AVL tree each node must contain a balance factor which indicates its state of balance relative to its subtrees.

  If balance is determined by

- (height left tree) - (height right tree)
- then the balance factors in a balanced tree can only have values of -1, 0, or 1.

AVL Tree : Examples

AVL tree with 1 to 4 nodes



Insertion of a new Node

- Adding a new node can change the balance factor of any node by at most 1.
- If a node currently has a balance factor of 0 it cannot go out of balance.
- If a node is left high (left subtree 1 level higher than right) and the insertion is in its left subtree it may go out of balance (same applies to right/right).
- If a node is left high and the insertion takes place in its right subtree then

  a) it cannot go out of balance

  b) it cannot affect the balance of nodes above it since the balance factor of higher nodes is determined by the maximum height of this node which will not change.

Pivot Node

- If the balance factor of each node passed through on the path to an insertion is examined then it should be possible to predict which nodes could go out of balance and determine which of these nodes is closest to the insertion point. This node will be called the pivot node.
- Restoring the balance of the pivot node restores the balance of the whole sub-tree and potentially all of the nodes that were affected by the insertion.

- The mechanism of restoring balance to an AVL tree is called rotation.

Tree Rotations

- Correction of the LL or RR imbalance requires only a single rotation about the pivot node but the second case, LR or RL requires a prior rotation about the root of the affected subtree then a rotation about the pivot node.

- A rotation consists of moving the child of the high subtree into the position occupied by the pivot, and moving the pivot down into the low subtree.

LL (RR) Imbalance

- To correct an LL imbalance the high child replaces the pivot and the pivot moves down to become the root of this child's right subtree.
- The original right child of this node becomes the left child of the pivot (for RR exchange left and right in the description).

LR (RL) Imbalance

- To correct an LR imbalance a rotation is first performed about the left child of the pivot as if it were the pivot for a RR imbalance.

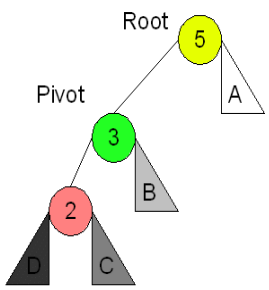
- This rotation effectively converts the imbalance of the original pivot to a LL which can now be corrected as above.

Restoring Balance - Tree Rotation

There are 4 cases in all, choosing which one is made by seeing the direction of the first 2 nodes from the unbalanced node to the newly inserted node and matching them to the top most row.

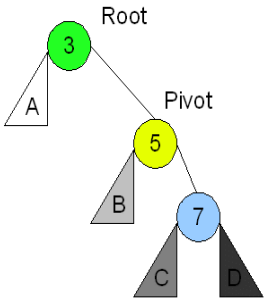
Root is the initial parent before a rotation and **Pivot** is the child to take the root's place.

Left Left Case



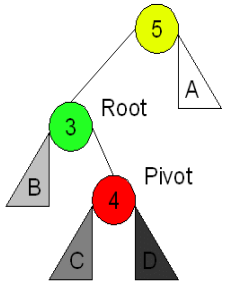
Right Rotation

Right Right Case



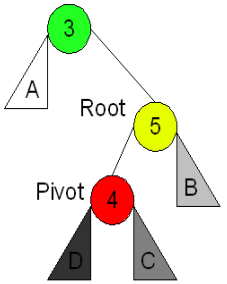
Left Rotation

Left Right Case

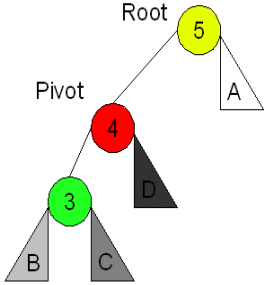


Left Rotation

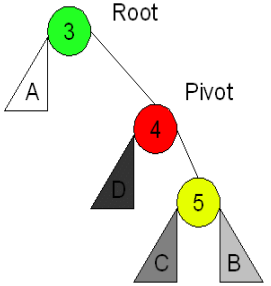
Right Left Case



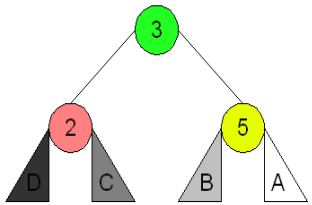
Right Rotation

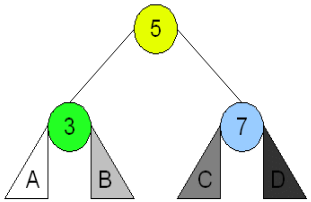


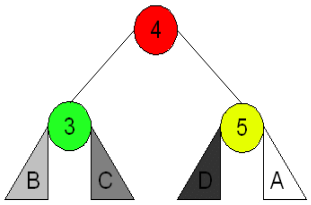
Right Rotation

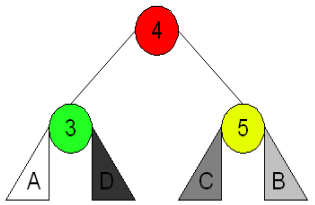


Left Rotation







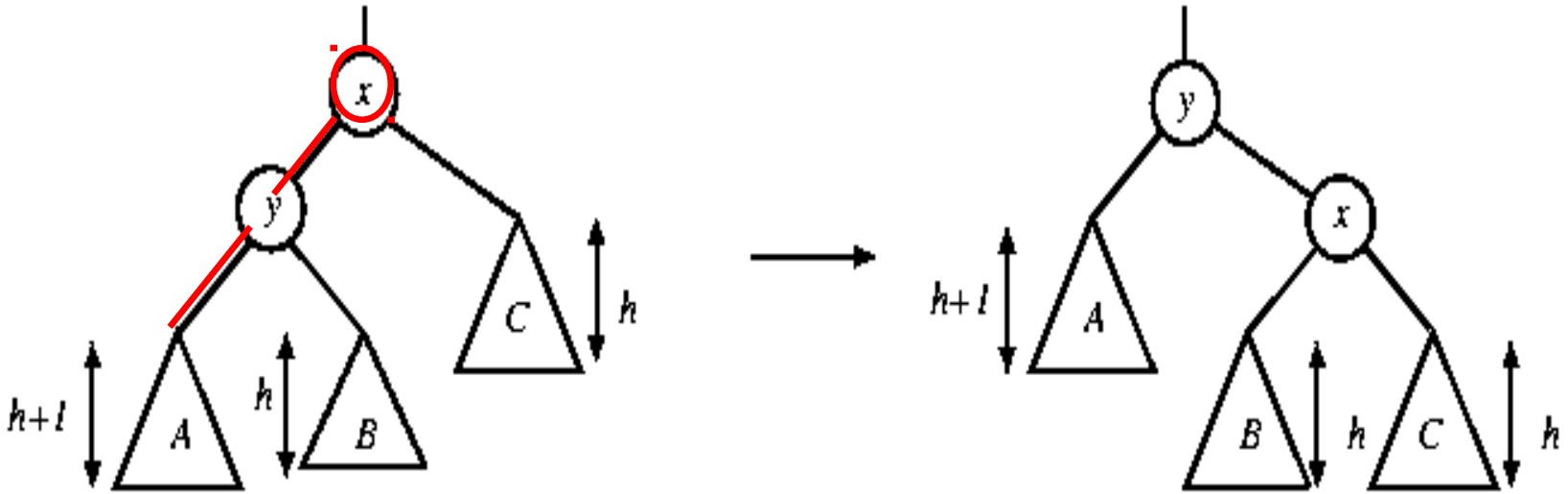


Single rotation

The new key is inserted in the subtree A.

The AVL-property is violated at x

- height of left(x) is h+2
- height of right(x) is h.

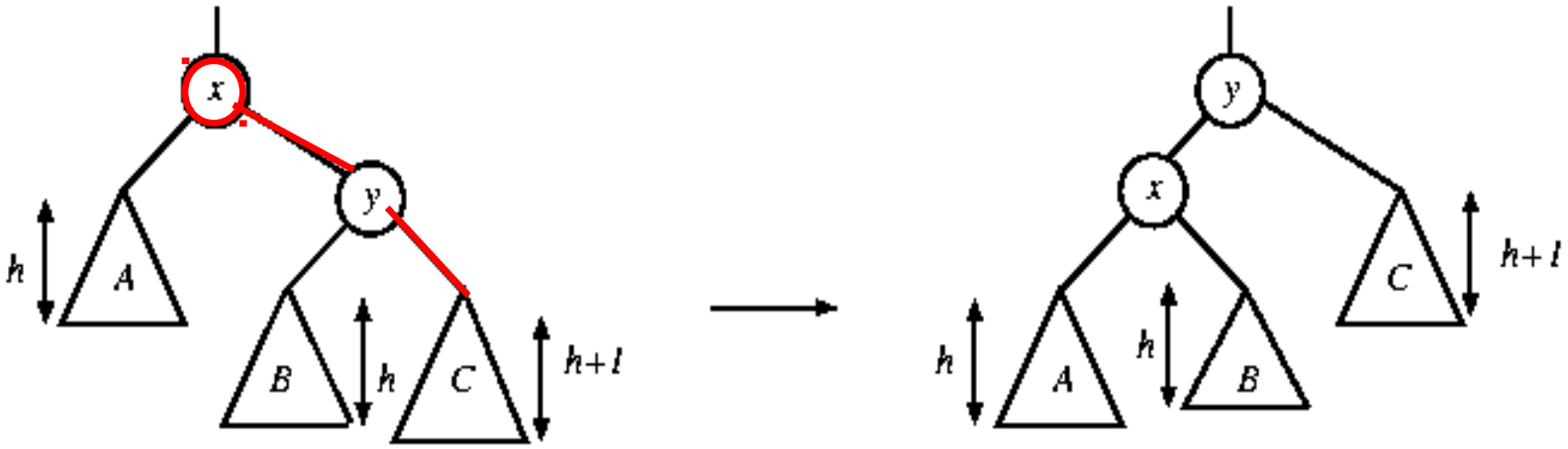


Rotate with left child

Single rotation

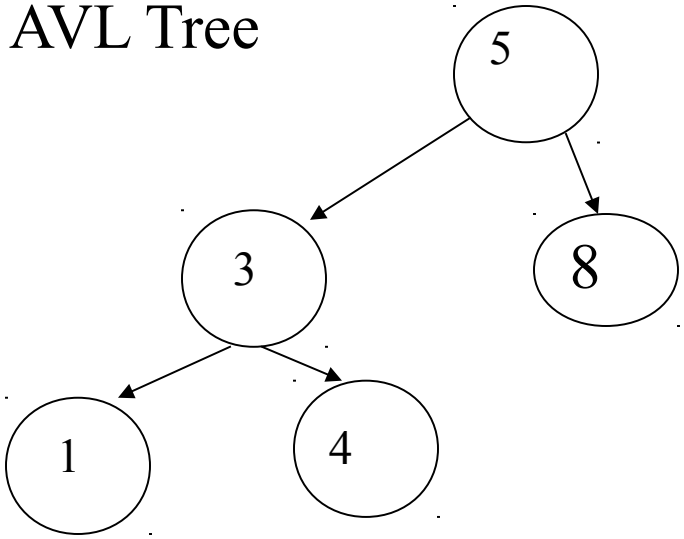
The new key is inserted in the subtree C.

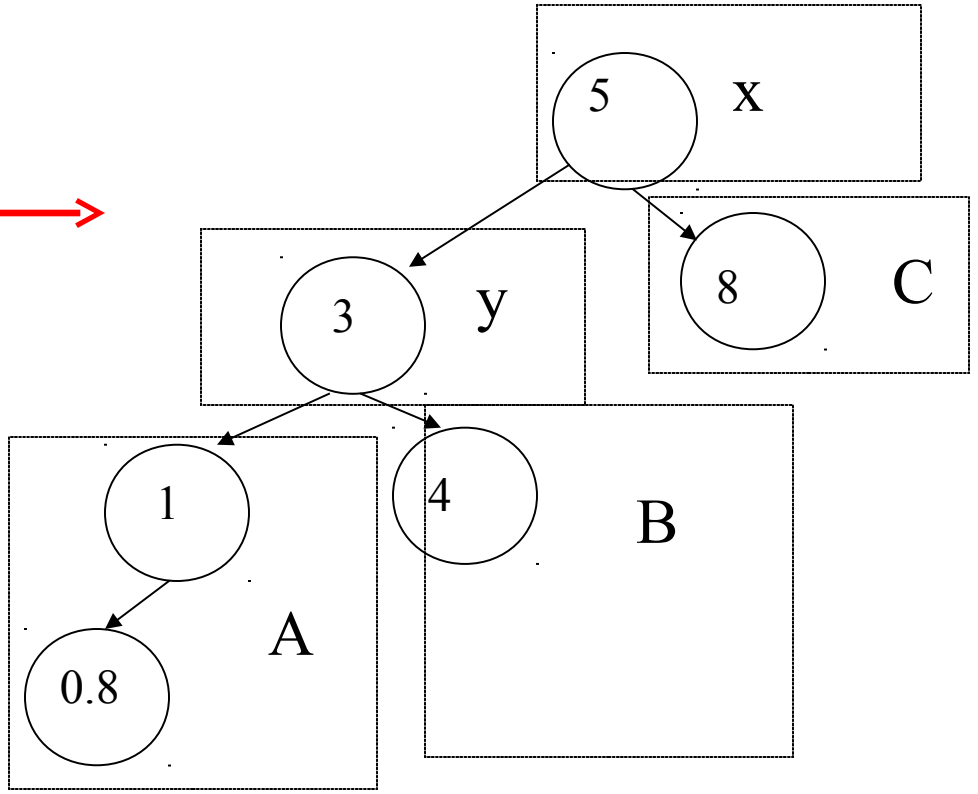
The AVL-property is violated at x.



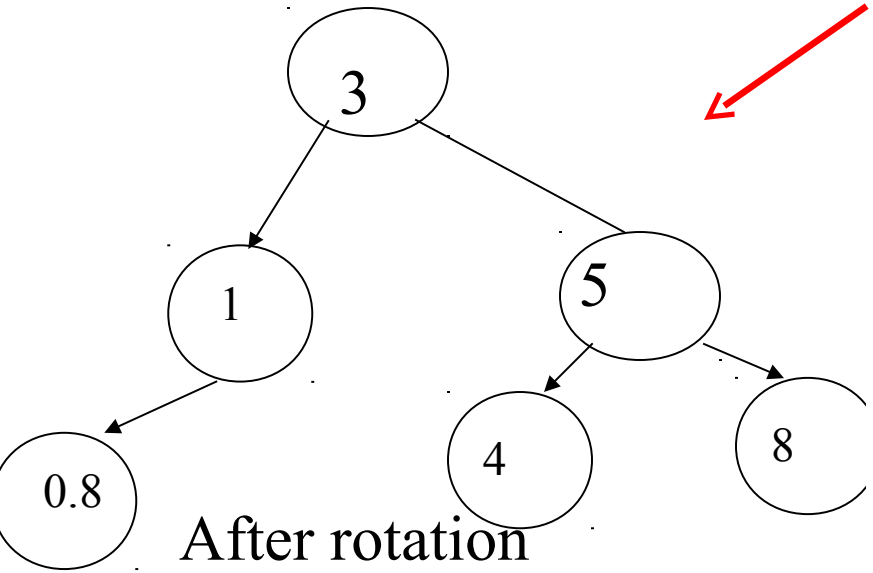
Rotate with right child

AVL Tree

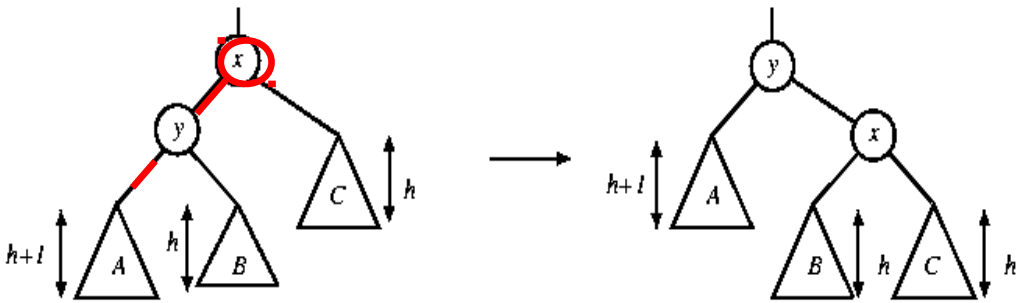




Insert 0.8



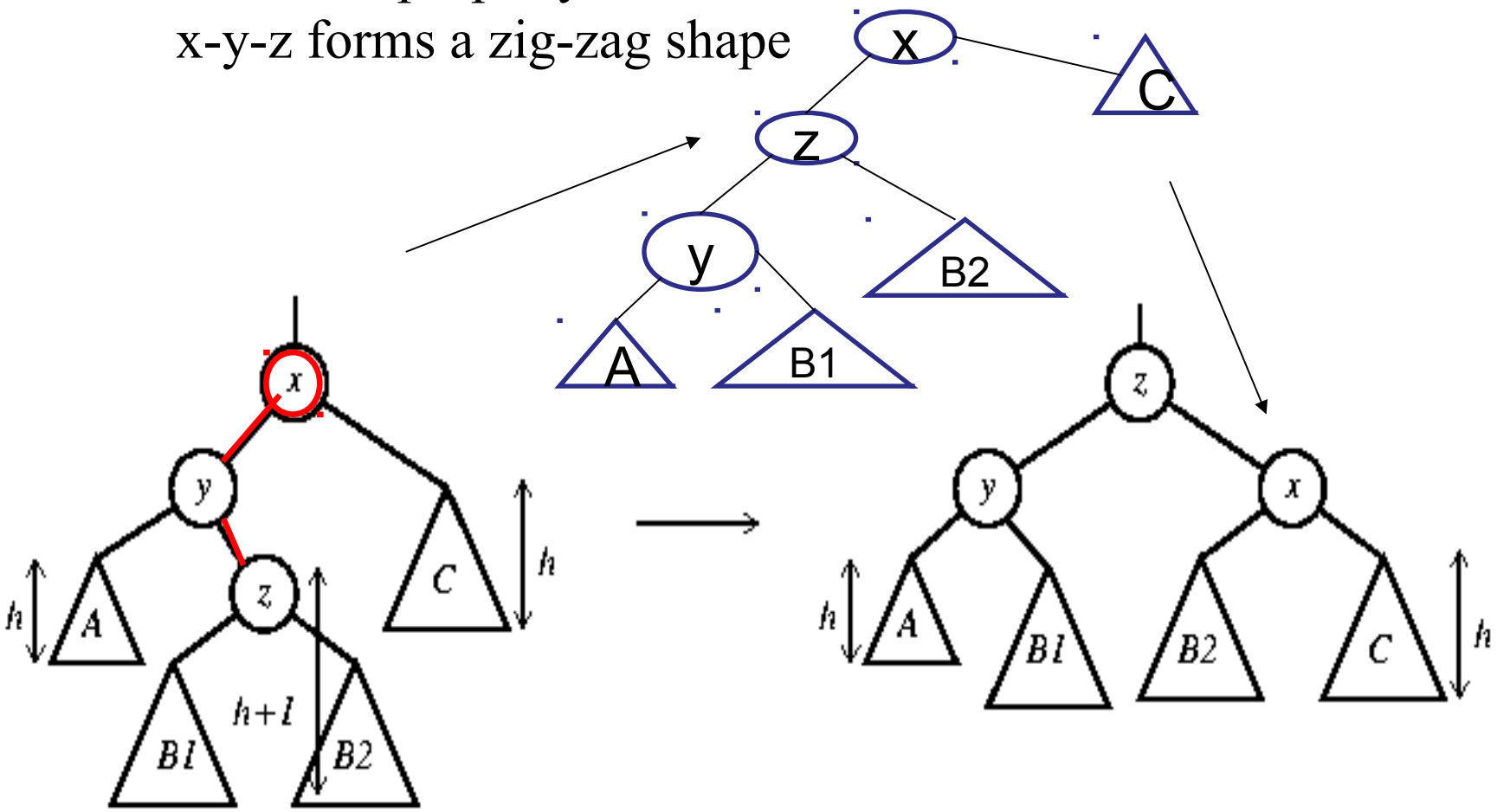
After rotation



Rotate with left child

Double rotation

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

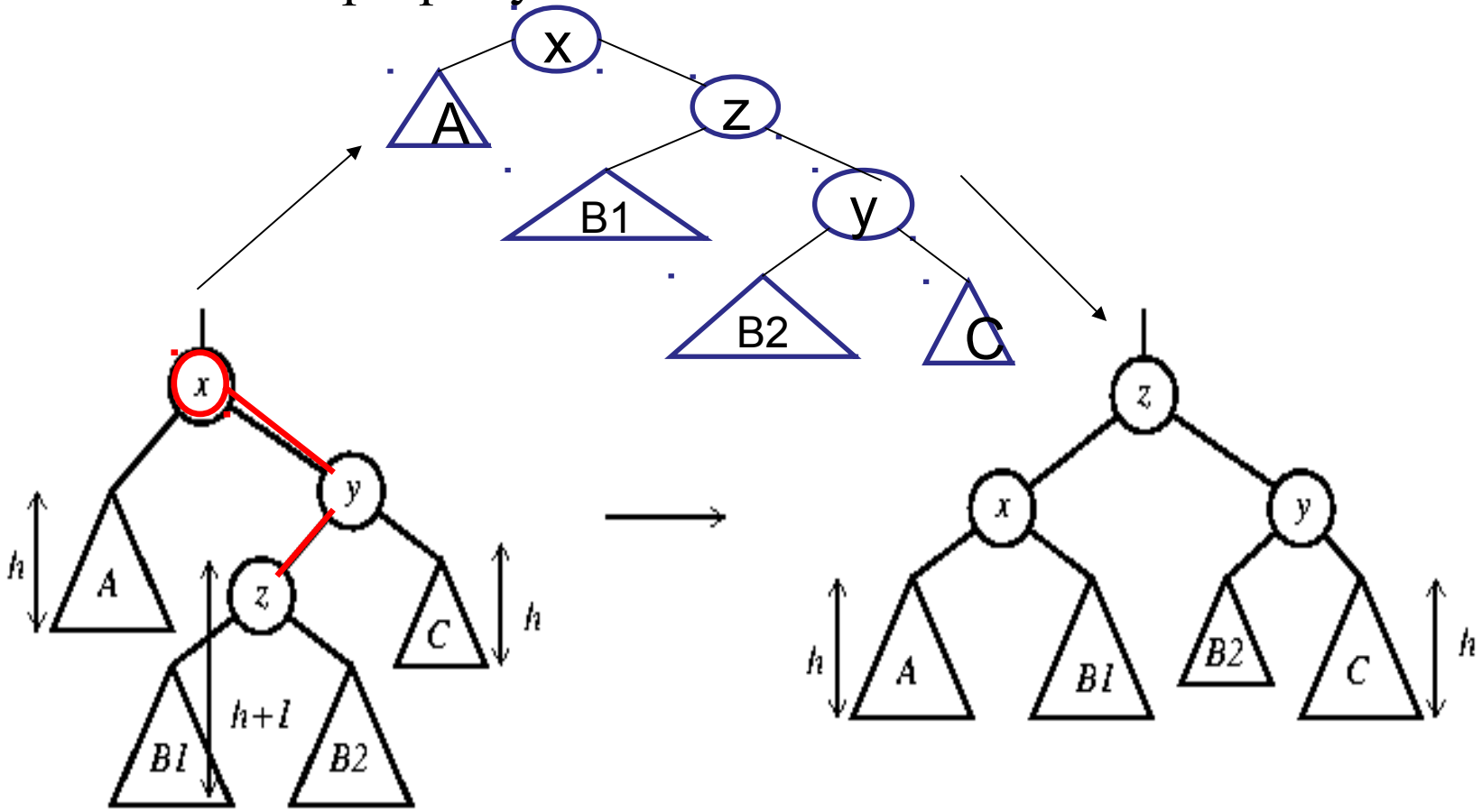


Double rotate with left child

also called left-right rotate

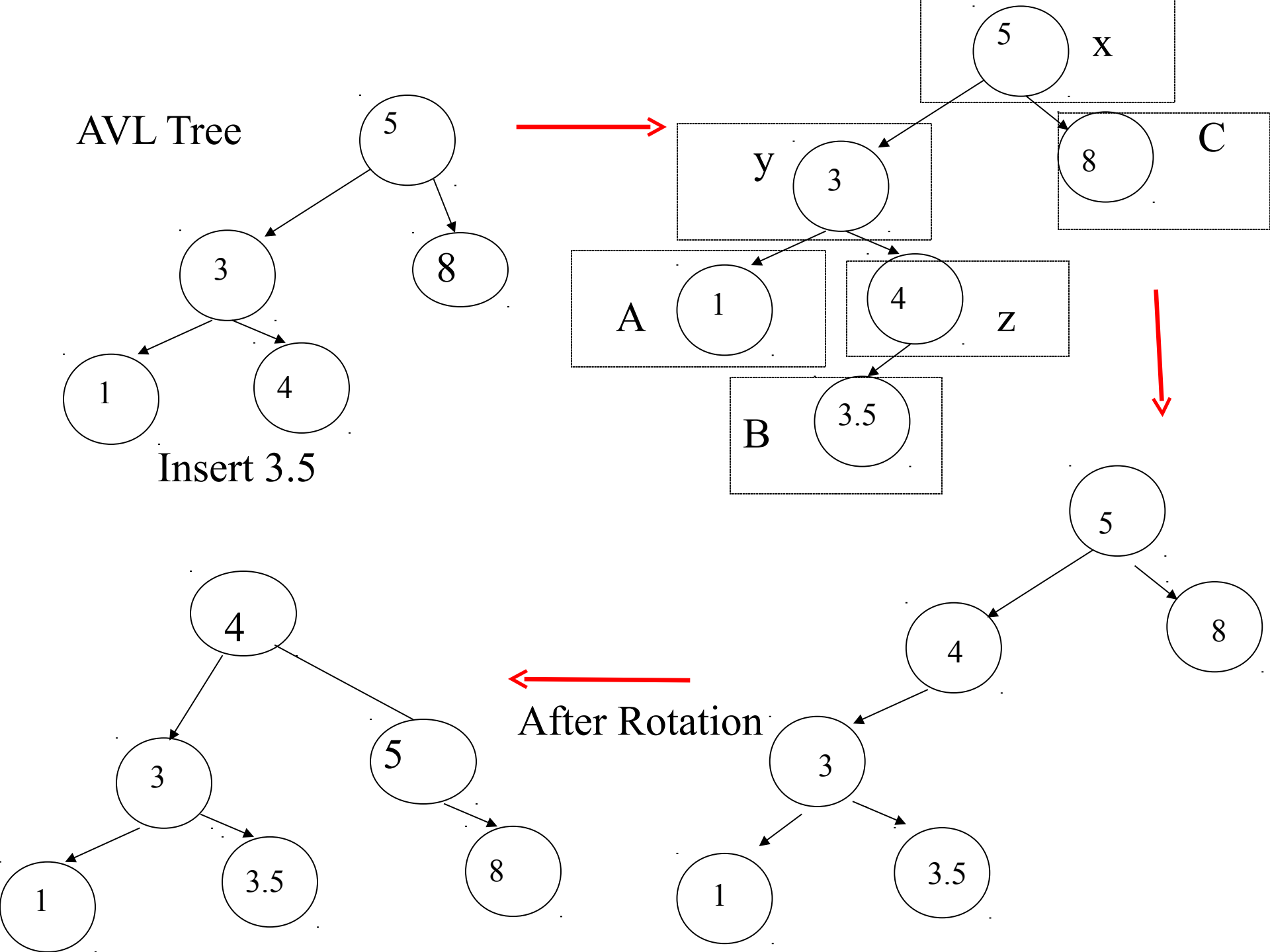
Double rotation

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



Double rotate with right child

also called right-left rotate

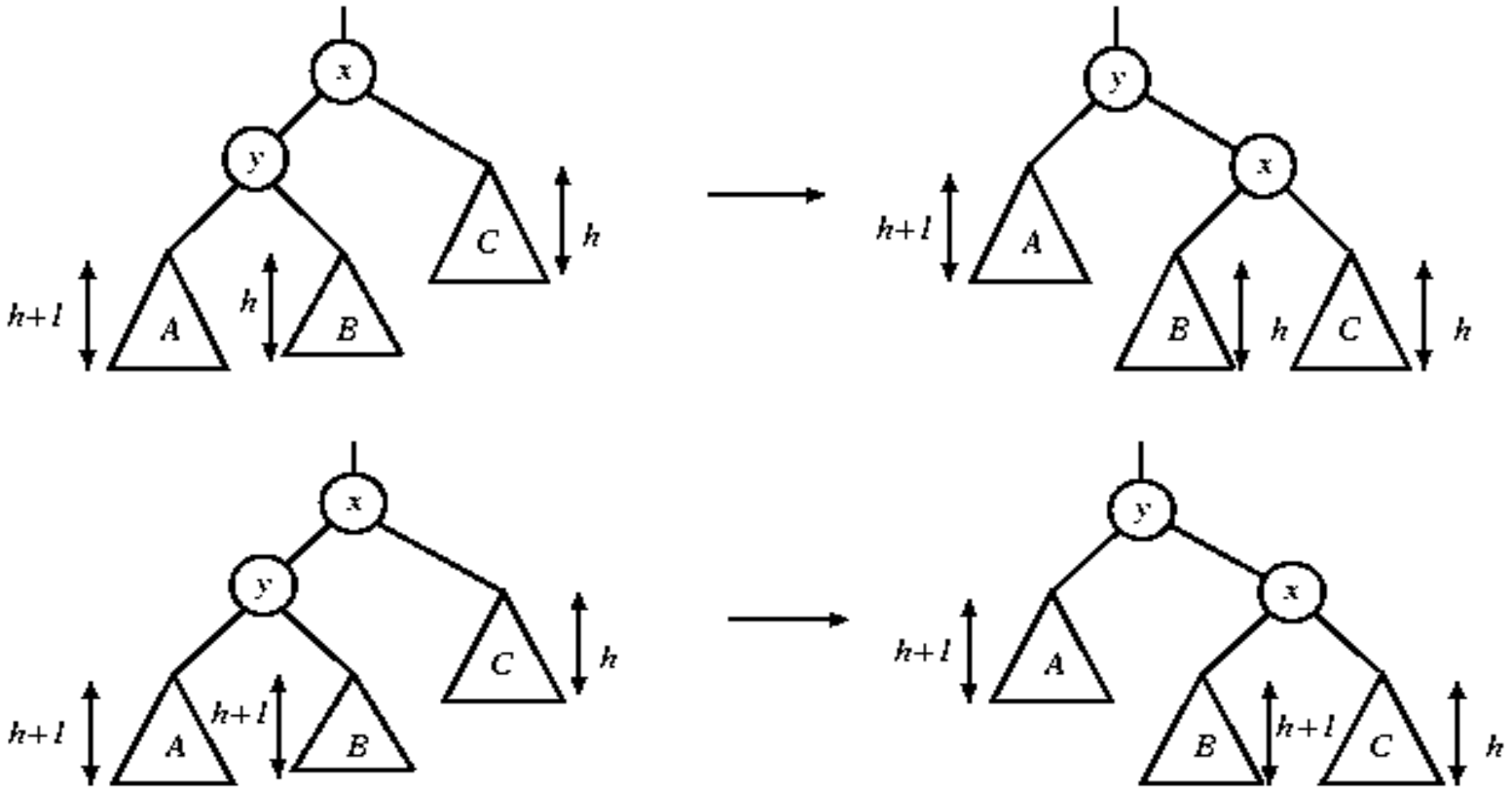

AVL Tree
5
3
8
1
4
Insert 3.5
5
x
y
3
8
C
A
1
4
z
B
3.5
5
4
8
3
1
3.5
After Rotation
4
3
5
1
3.5
8

Deletion

- Delete a node x as in ordinary binary search tree. Note that the last node deleted is a leaf.
- Then trace the path from the new leaf towards the root.
- For each node x encountered, check if heights of left(x) and right(x) differ by at most 1. If yes, proceed to parent(x). If not, perform an appropriate rotation at x.
- For deletion, after we perform a rotation at x, we may have to perform a rotation at some ancestor of x. Thus, we must continue to trace the path until we reach the root.

Single rotations in deletion

In both figures, a node is deleted in subtree C, causing the height to drop to h. The height of y is h+2. When the height of subtree A is h+1, the height of B can be h or h+1. the same single rotation can correct both cases.



Summary : AVL Tree

- Balanced trees demonstrate superior performance to binary search trees due to log (n) tree depth
- Rotations are used as a typical mechanism to restore balance
- Balanced trees are used in lookup-intensive applications