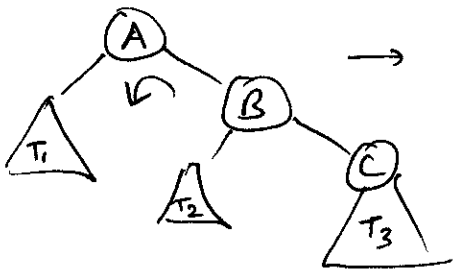
AVL Trees : A binary search tree with the following additional properties. Every node has an additional field "height" that stores the height of that node. Every node of the tree must satisfy the following balance condition:

$$|\text{node.left.height} - \text{node.right.height}| \leq 1.$$

A tree in which every node is balanced satisfies $h = O(\log n)$.

When a node goes out of balance after add/remove operation, the tree is rotated to bring all nodes back into balance:

Zig Zig Case

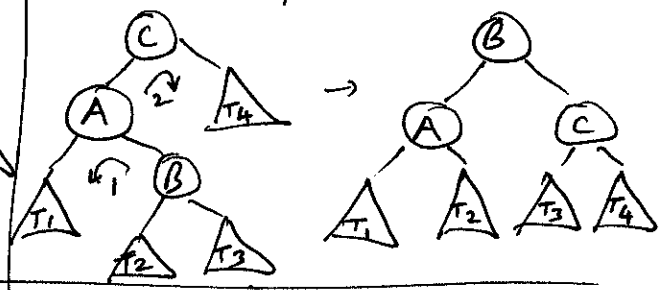


Node A is out of balance because of a new node inserted into C's subtree (T_3)

C = A.right.right

(Case left.left is symmetric)

Zig Zag Case



Node C is out of balance because of a new node inserted into B's subtree.

B = C.left.right

(right.left is symmetric)

Find/contains — same as BST

add(x) — same as BST + After node is inserted into tree, height field has to be updated on the way back to root:

node.height = max(node.left.height, node.right.height) + 1

The first node where an imbalance is detected, performs Zig Zig / Zig Zag rotation to bring tree back to balance.

remove(x) - Follows BST's remove - on the way back up to root, if an imbalance is found, tree is rotated to restore balance.

In all cases, it can be shown that balance can be restored by just one Zig-Zig or Zig-Zag rotation.

AVL tree example

Insert: 1, 3, 5, 7, 9, 6 in that order.