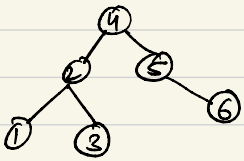
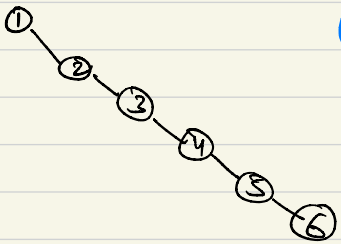
Recitation-09 : AVL Trees :-

BST: left child < parent < right



Not all trees end up like this :-

Sometime: 1, 2, 3, 4, 5, 6



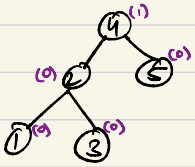
→ Losing Benefit of BST ⇒ why?

One side became too long.

Problem: Worst case Time complexity of Binary Search Trees is O(n)

Solution AVL Trees:- Balanced Trees → Tree Always Remains Balanced.

AVL tree is a self balancing BST in which each node maintains extra info called "Balance Factor", whose value is −1, 0, 1



( ) indicate balance factor.

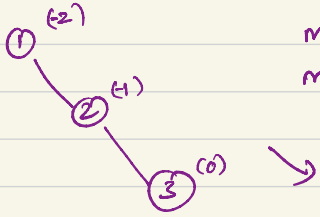
How can we ensure that the tree remains balanced throughout ?

(Move To Demo)

link : www.cs.usfca.edu/~galles/visualization/AVLtree.html).

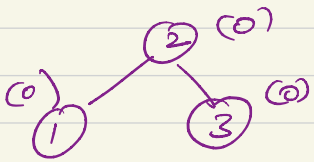
Type of Imbalances :-

(A) LL Rotation

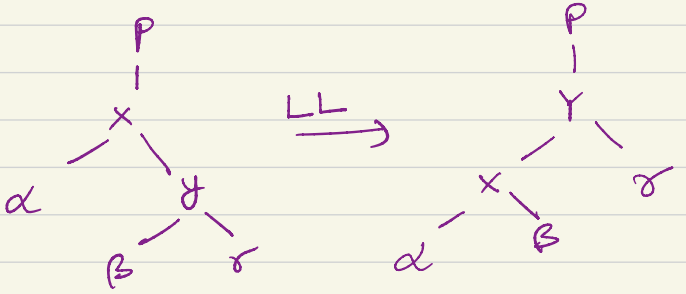


How to fix ?

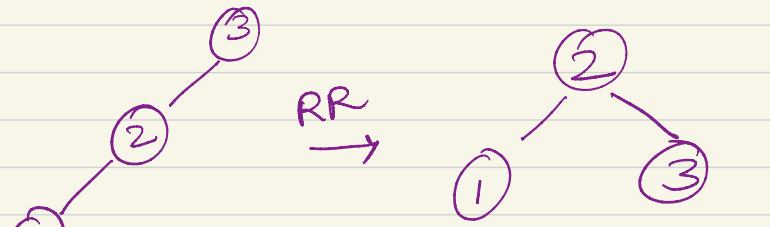
make (2) the main node,
make (2) left = (1)



Move Generally :



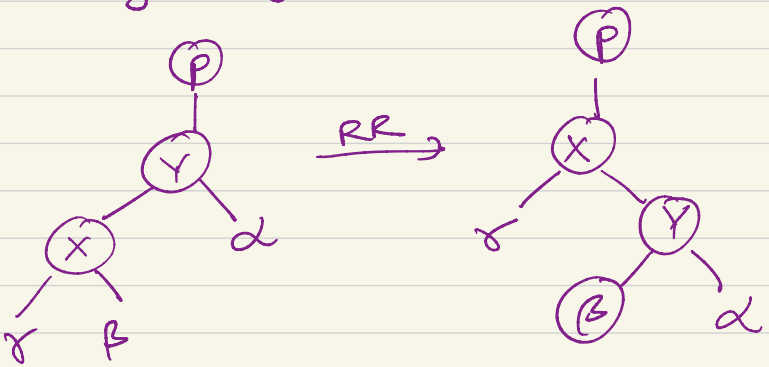
(B) RR Rotation



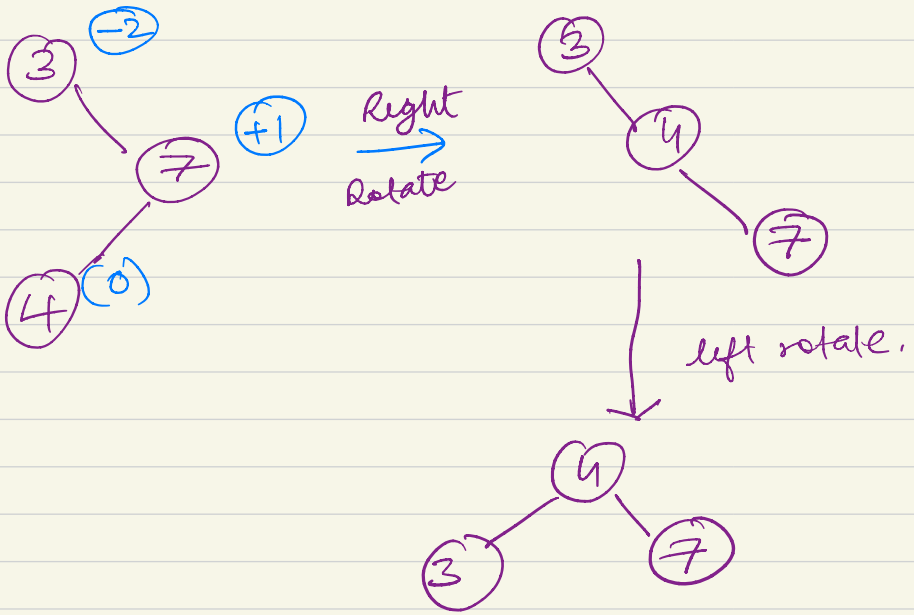
make (2) parent now

2 · Right = (3)

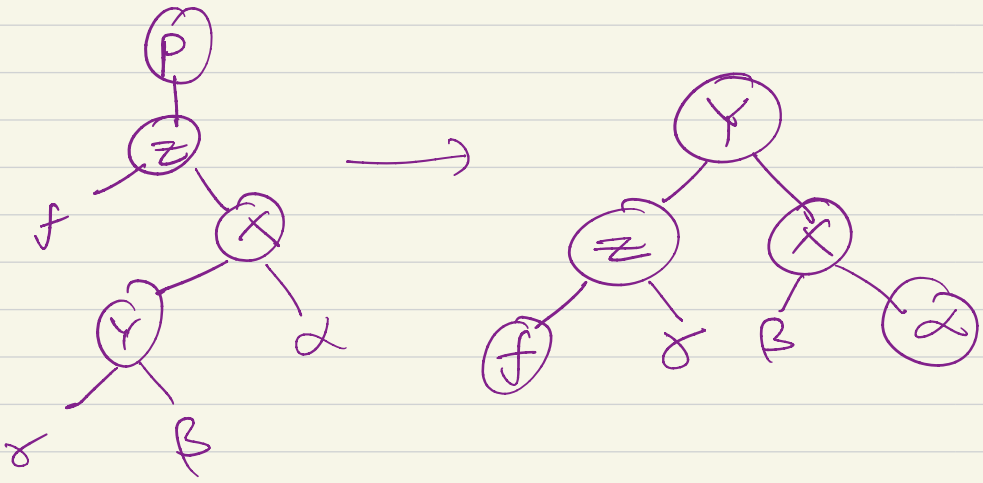
more generally.



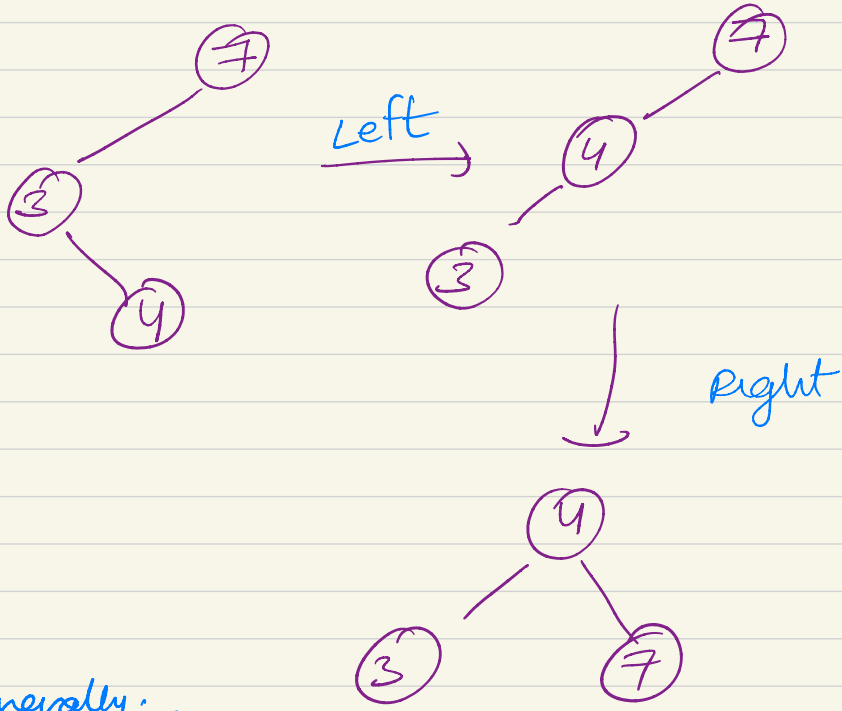
© Right left Rotation :-



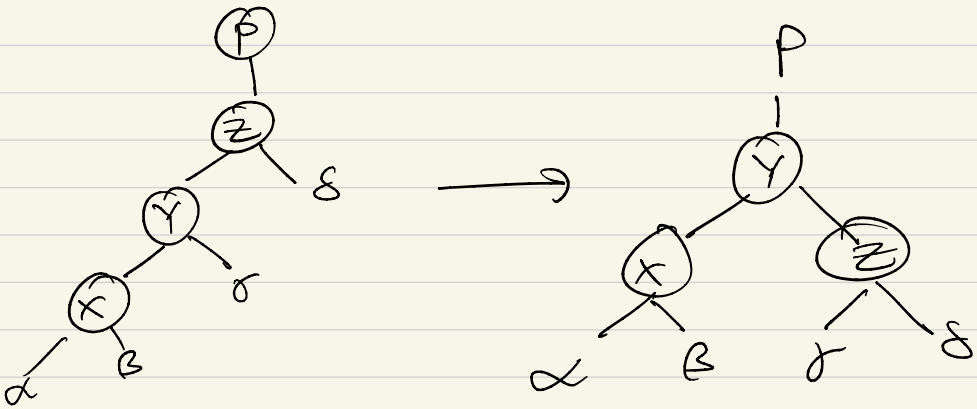
more generally :-



(d) Left Right Rotation :-



More generally :-

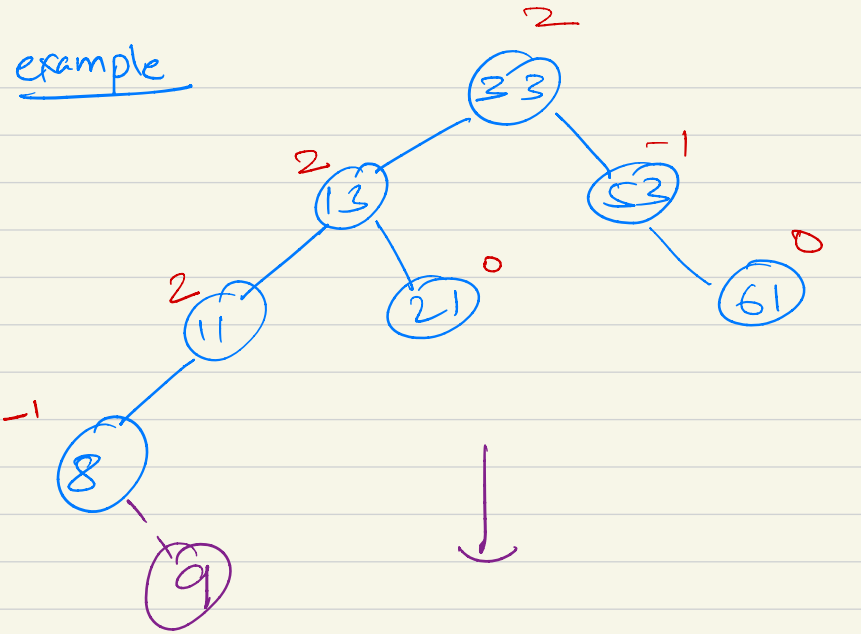


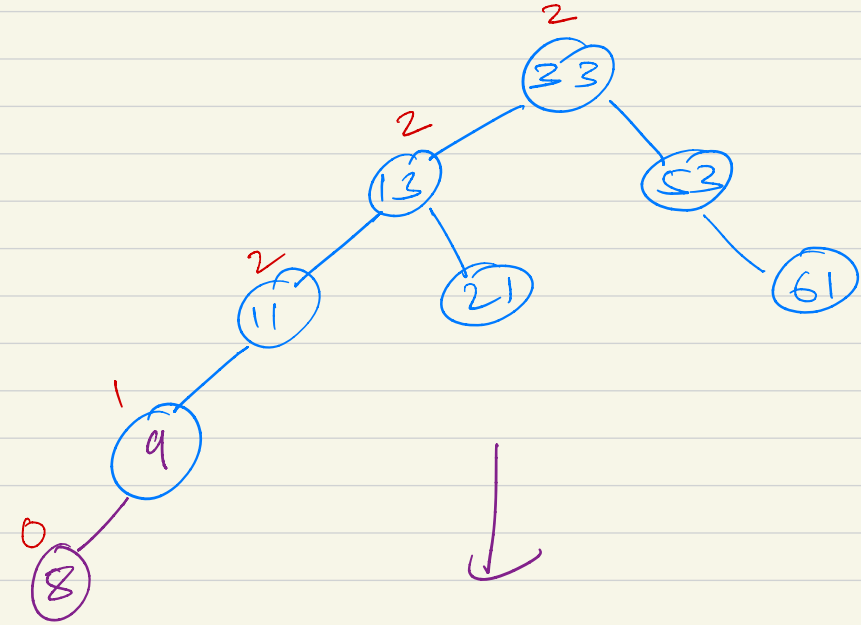
So till now :- Given an unbalanced node with balance -2 or +2 → we know how to balance it.

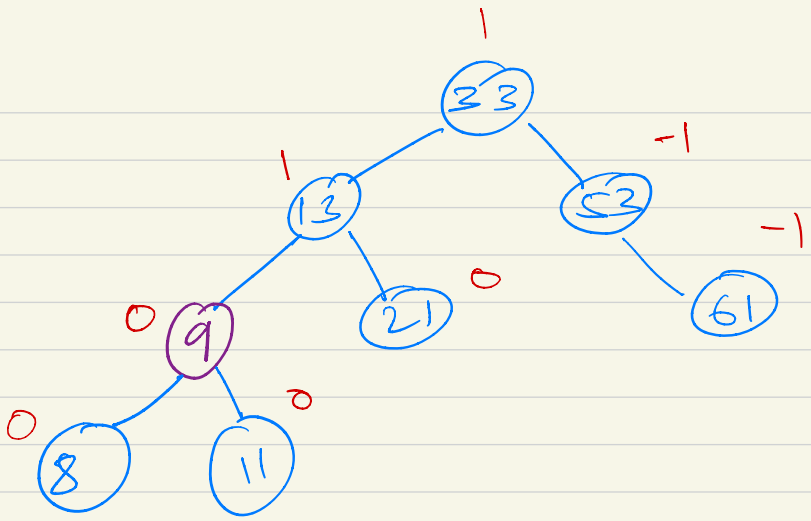
Insert in an AVL tree :-

1. Add node to it like BST
2. Update Balance factor after addto
3. If balance factor ≠ 0, 1, -1 } → balance the node
4. move upward.

example



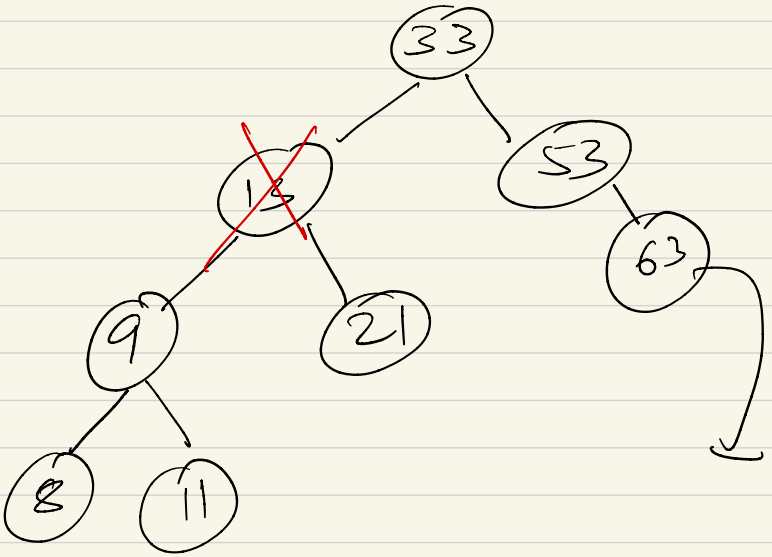




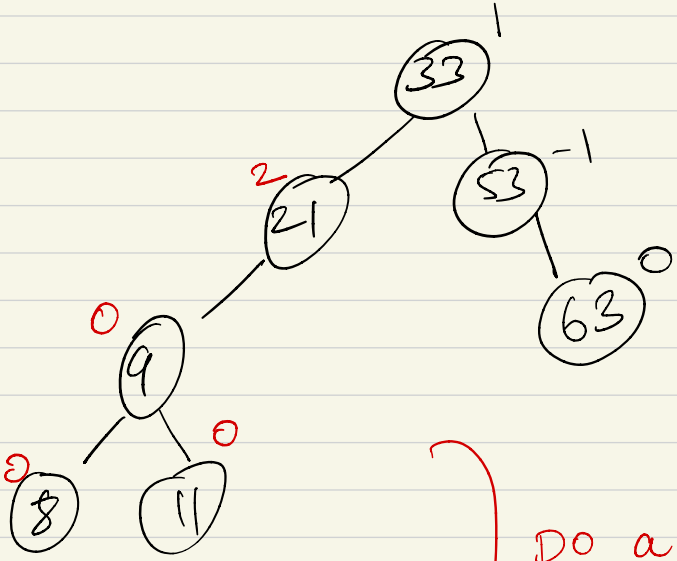
Code :-

- Identify nodes with imbalance
- Do figure out type of imbalance
- Do LL / RR / LR / RR

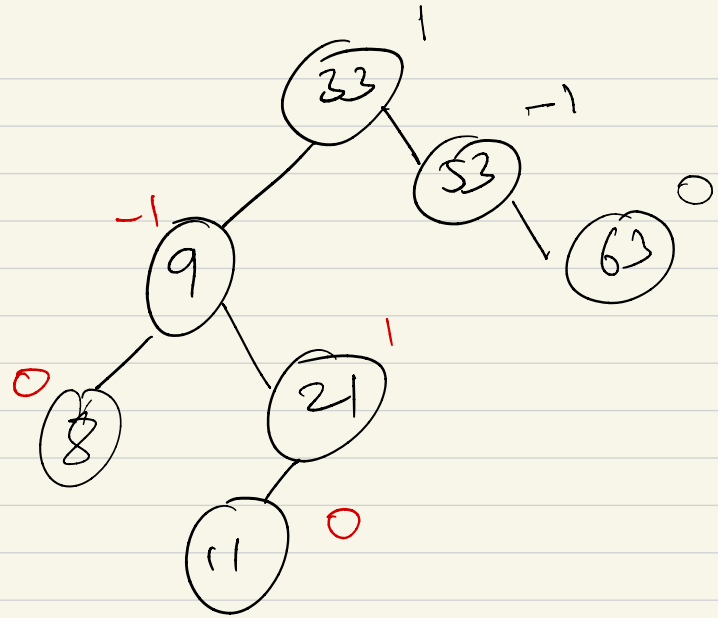
Deleting



replace node to be deleted with pred/ successor



DO a LL



Identify Type of imbalance?

(1) Node ⇉ +2 , let child ⇉ +1 or 0 LL

(2) Node ⇉ −2 , right child ⇉ −1 or 0 RR

(3) Node = +2 , left child = −1 LR

(4) Node = −2 , Right child = +1 RL

Ref uploaded java file !