AVL Trees

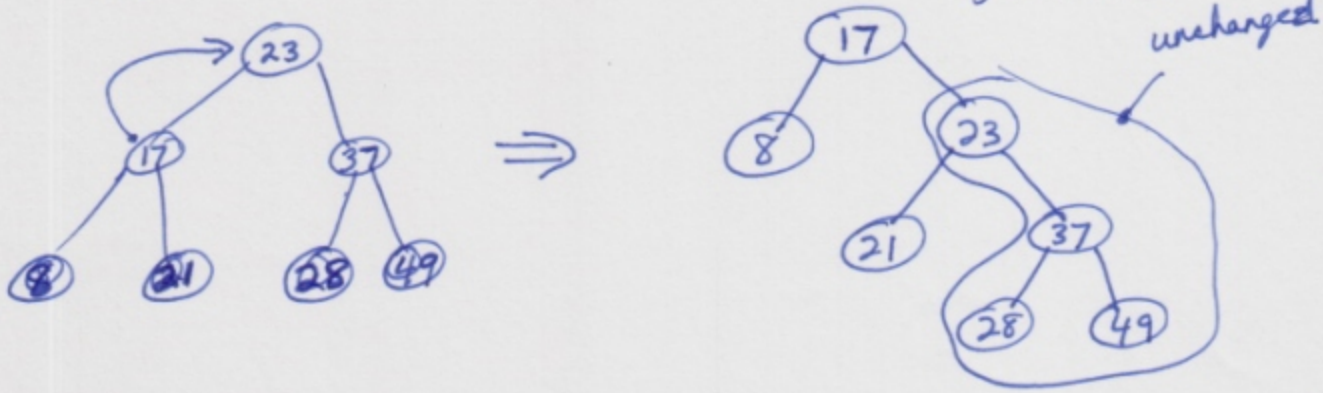
AVL trees are balanced binary trees (also mention 2-3 Trees)

they allow a tree to be rebalanced when inserts and/or deletes cause it to become unbalanced

the rebalancing operations make use of "rotate right" and "rotate left" operations

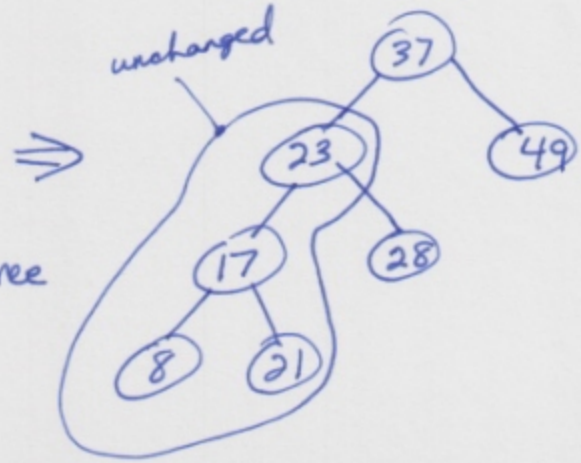
Rotate right:

21 > 17, so its entire subtree must move to right subtree



Rotate left:

1) right child becomes root
2) 28 (and its subtree) must move to left subtree



Defining balance of a tree (or subtree):

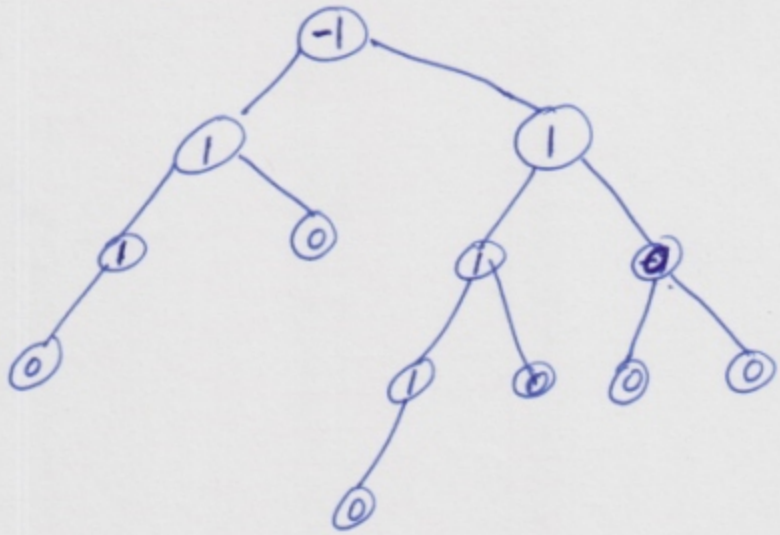
balance of a node is: height of left subtree − height of right subtree

balance of tree = balance of root

AVL trees will maintain balance = −1, 0, or 1 at every node, i.e. left/right heights never differ by more than one, at every node in tree

(this is sufficient to guarantee $O(\log n)$ search time in an AVL-based BST)

Balance example:



There are 2 ways a tree could become unbalanced after an insert:

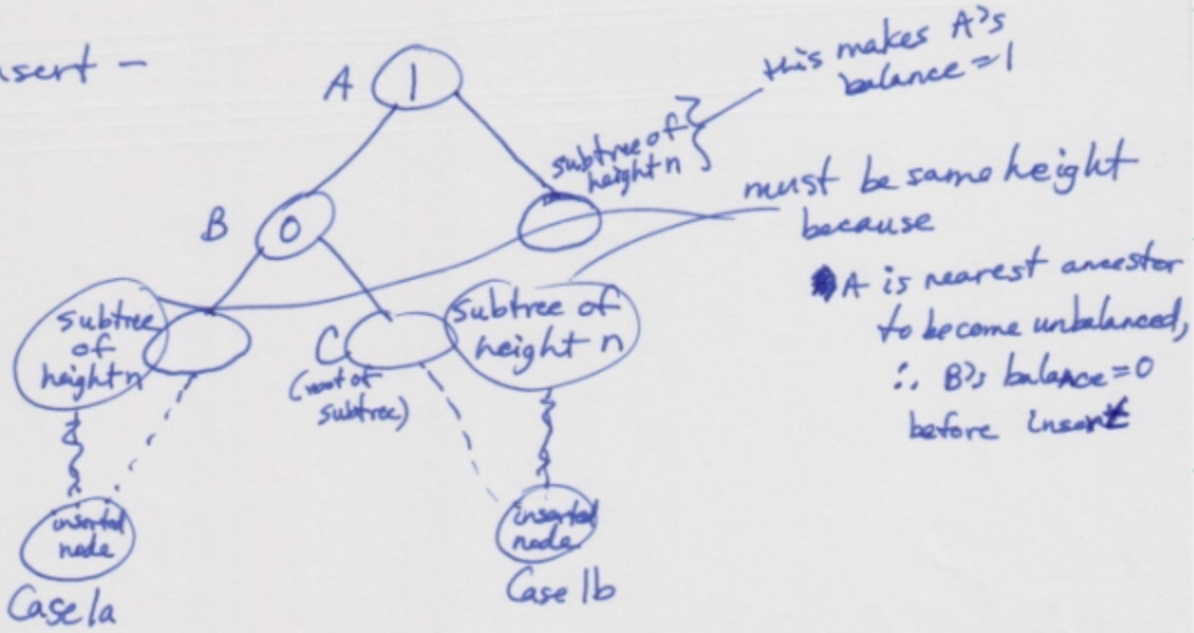
1) inserted node increases height of left subtree of a node w/ balance = 1
2) inserted node increases height of right subtree of a node w/ balance = −1

there are two possible sub-cases for each of these main cases

Consider Case 1:

focus on the nearest ancestor to become unbalanced after insert (A in below figure)

before insert –



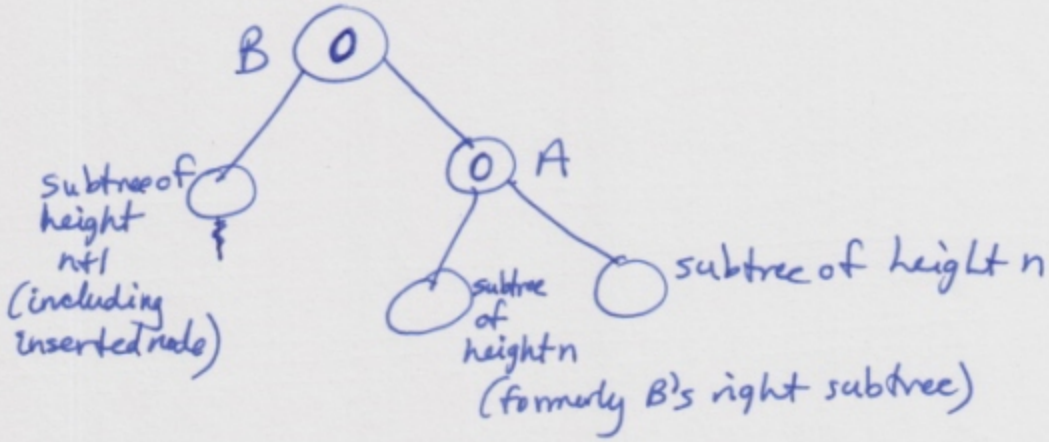
in Case 1a, B's balance becomes 1, A's balance becomes 2 after insert

in Case 1b, B's balance becomes −1, A's balance becomes 2 after insert

<u>Case 1a</u>:

to rebalance, just do a right rotation of tree rooted at A

after right rotation:



So, A and B are now balanced, other subtrees are still balanced

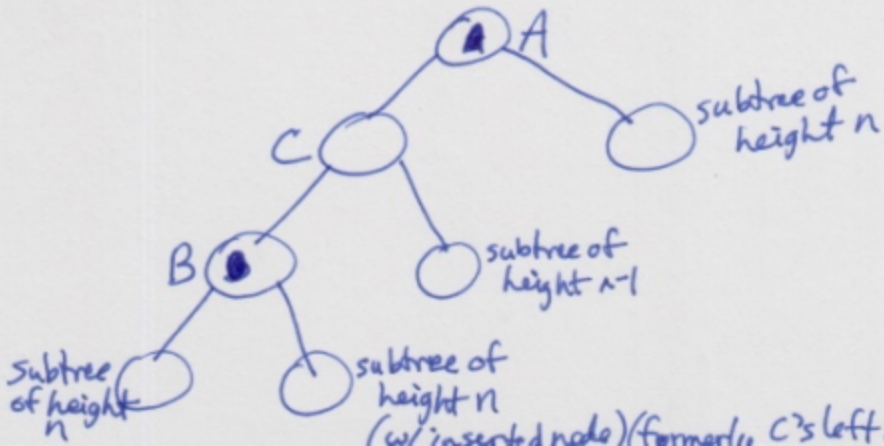
also, B's height = n+2 and A's height before insertion = n+2

since B replaced A, all ancestors of B are unaffected

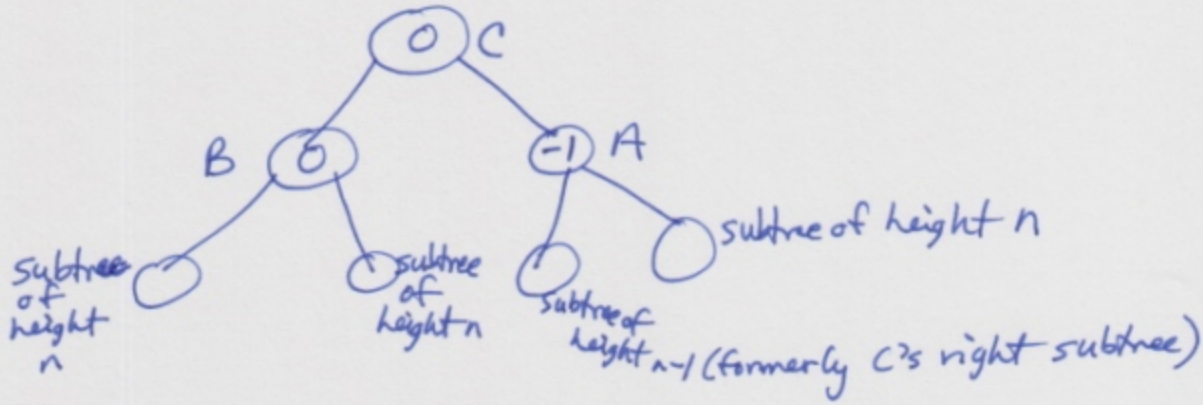
<u>Case 1b (w/ inserted node in left subtree of C):</u>

to rebalance, do a left rotation at B followed by a right rotation at A

after left rotation of B:



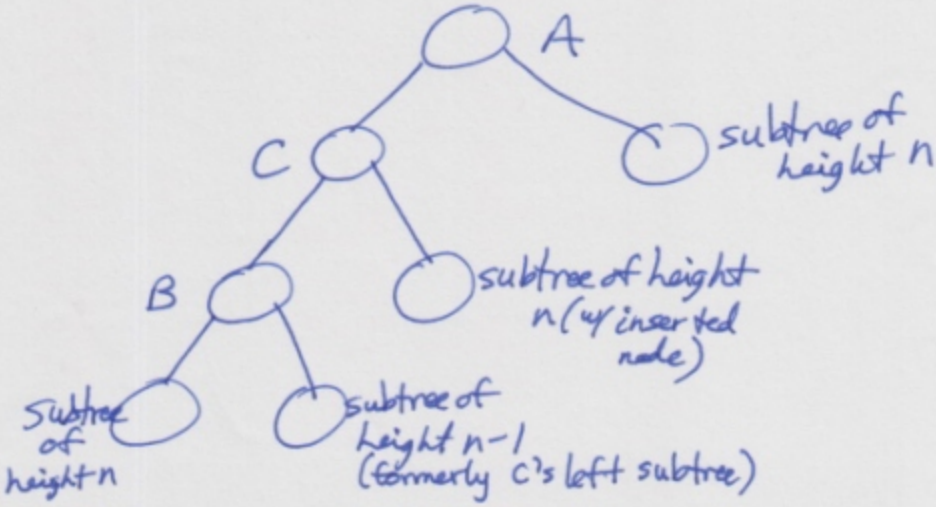
after right rotation at A:



again, height of C = n+2 so ancestors of C unaffected

<u>Case 1b (w/ inserted node in right subtree of C):</u>

after left rotation at B:



after right rotation at A: