AVL

Input = { 20 , 30 , 8 , 47 , 39 , 18 , 40 , 24 } ①

(24 = last four digit of my student number.)

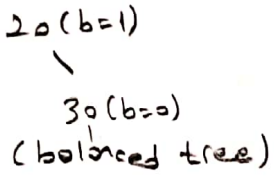
root = null

b = balance amount of node

① add(20):

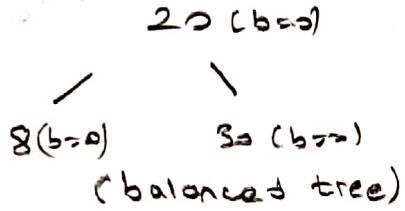
20 b=0
(balanced tree)

② add(30):



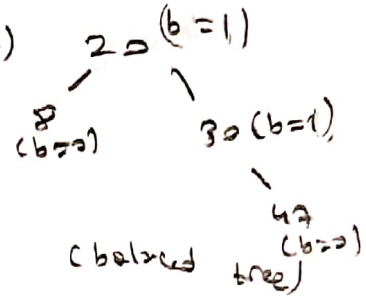
(balanced tree)

③ add(8)



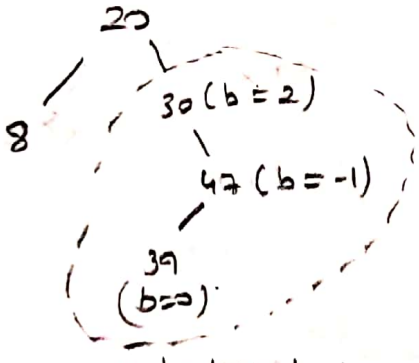
(balanced tree)

④ add(47)

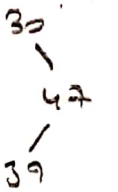


(balanced tree)

⑤ add(39)

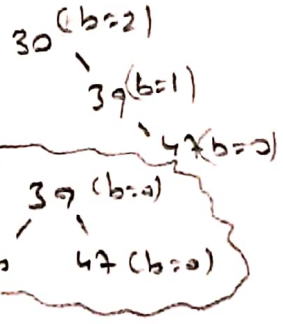


unbalanced

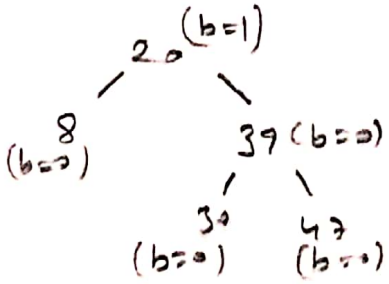


Right - left situation
1) Rotate child to right
2) Rotate parent to left

after 1)

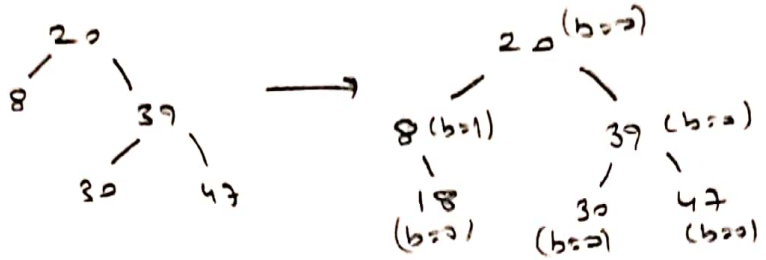


after 2)



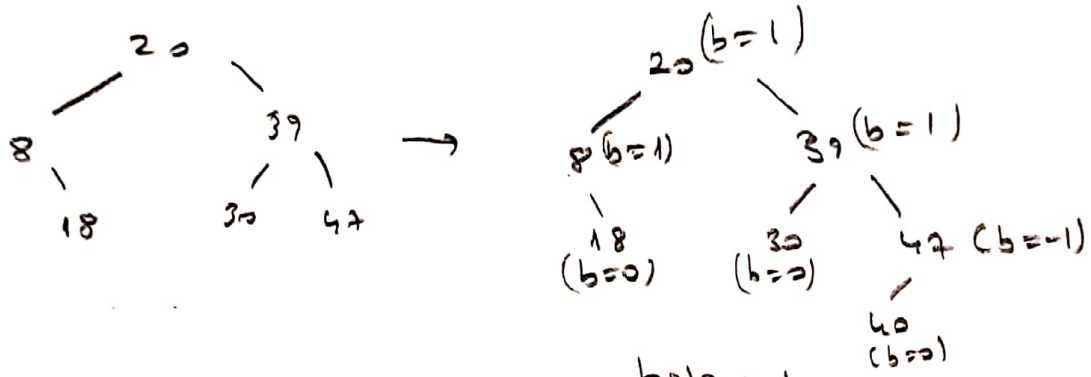
balanced

add(18):



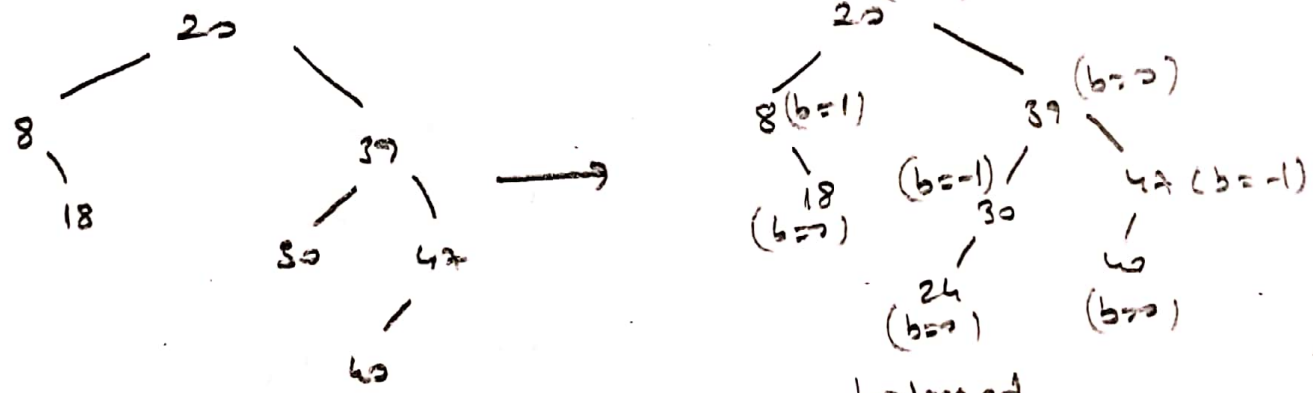
balanced

add(40):



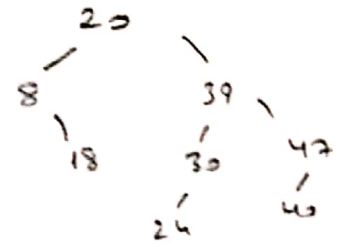
balanced

add(24):



balanced

current tree:



* In removing process, we first remove the item according to removing node from binary-search-tree that is defined in our book.

* Then, we need to check balance of each node and make some rotation in regarding to result of balance.

Removing (Same as in binary-Search-tree)

1) if left node of removed node is empty and right node of removed node is not empty, pull out the node from its parent and link the removed node right subtree to parent of removed node (if it does exist)

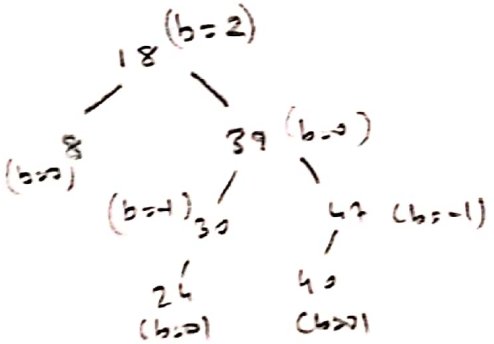
2) if rule number (1) true for right child. Apply it for right child.

3) if left child of right child is empty, make removed node element same as its left child, then make removed element reference its left child reference.

4) find the largest element of its left sub-tree then make removed node item same as largest element which is found. Assign found left sub-tree to right sub-tree of left-sub-tree which belongs to removed node

---

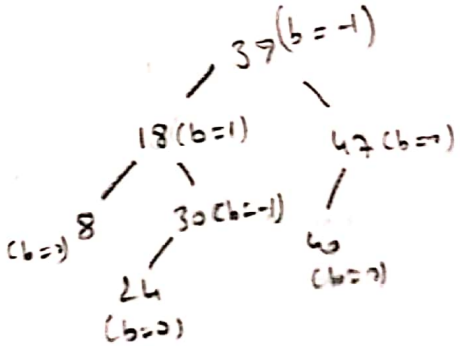
remove (20)



unbalanced

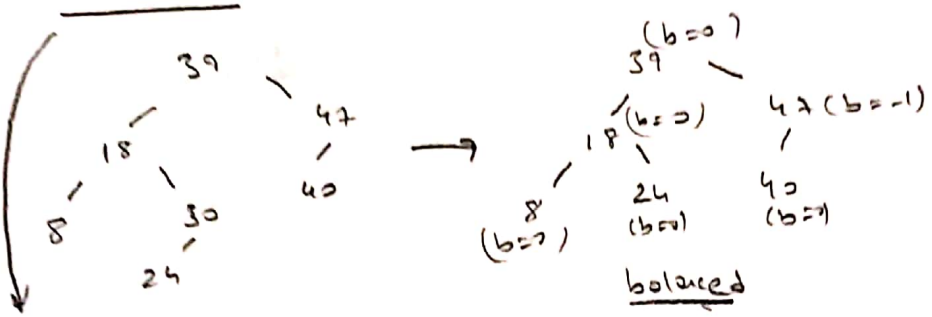
case: Root right sub-tree unbalanced

* we need to rotate to the left. While doing that, we need to choose right-left reference to rotate. 39 will be root, left sub-tree of 39 will be linked to as right sub-tree of 18.

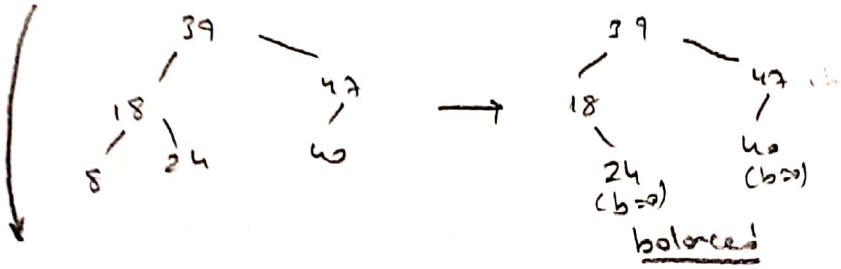


balanced

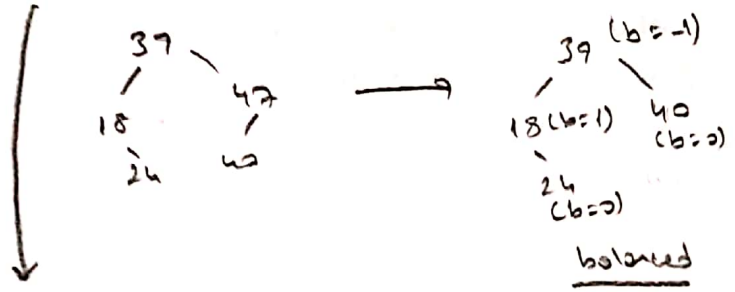
remove (30)



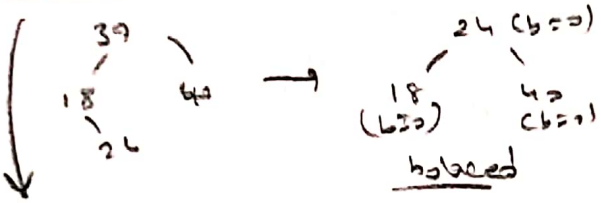
remove (8)



remove (47)



remove (39)



remove (18)

remove (24) 24 → -

(empty tree)