Red - Black Tree

r → red
b → black

(1)

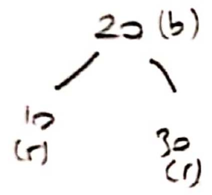
* In insertion process of red-black tree, there are some cases to apply, I will list these cases below and while inserting, these cases will be really helpful to apply I took that cases from the book.

* There are 4 main rule for building red-black tree:

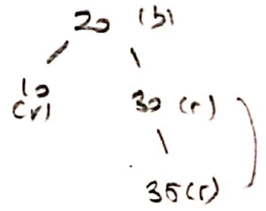
⇒
1) A node is either red or black
2) The root is always black
3) A red node always has black children.
4) The number of black node in any path from the root to a leaf is the same.

Helpful Cases :

Case 1



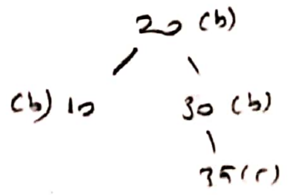
→ inserting (35)



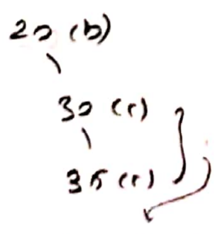
invariant 3 is violated

* So we need to change parent color as black. Grand-parent has red tree at the beginning. It is important rule for applying that solution.

* After solution our tree will be:

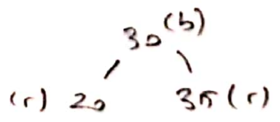


Case 2

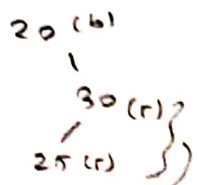


invariant 3 is violated

* we need to change the color of parent (30) as black. and notice that all nodes are right aligned. Then we need to rotate left.

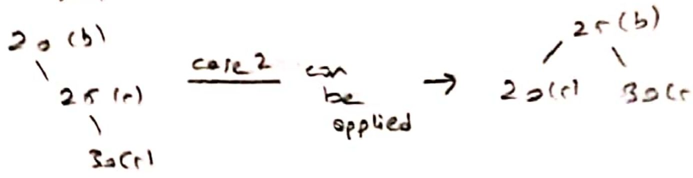


case 3



invariant 3 is violated

* Notice that nodes alignments are not the same so different than case 2. In that case we need to take this structure and create a form which case 2 can be applied. So we need to rotate 30 (r) toward right
After rotation to right.



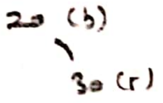
* Applying first insertion is made according to binary-search-tree.

Input = { 20, 30, 8, 47, 39, 18, 40, 24 }
$\underbrace{40, 24}$ → last four digits of my number

① **add (20)**

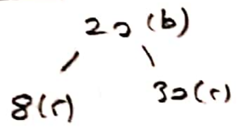
20 (b) * inserted as black initially.

② **add (30)**



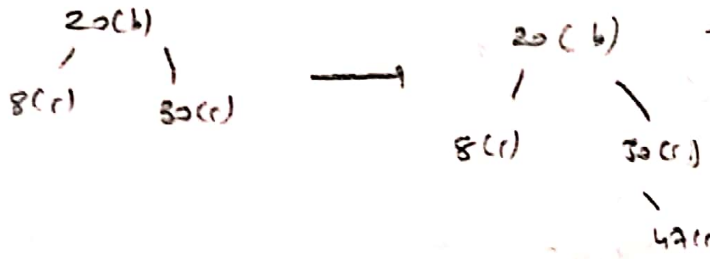
→ * There is no violation, we can insert as red.

③ **add (8)**



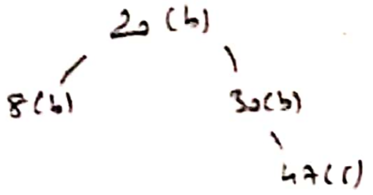
* There is no violation, we can insert as left child

④ **add (47)**



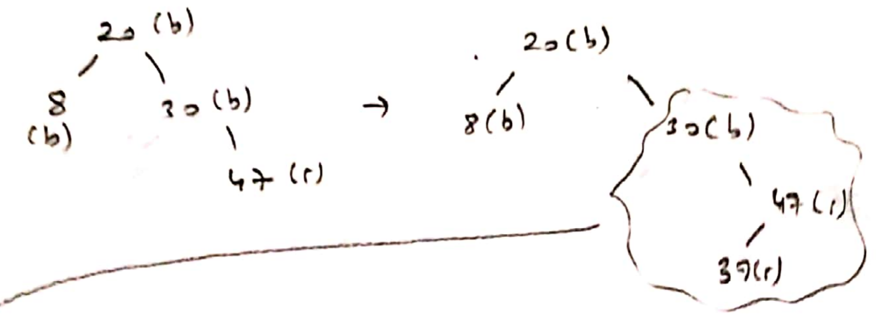
→ There is a violation of rule 3. And case 1 can be applied as solution

* we need to change the colors of the parents



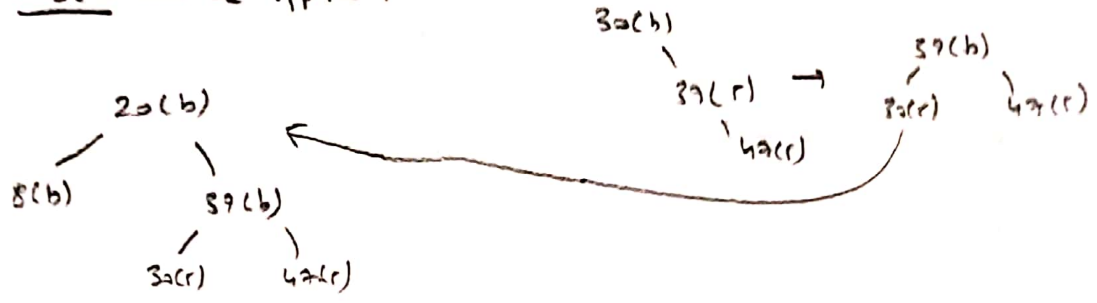
balanced.

add(39)



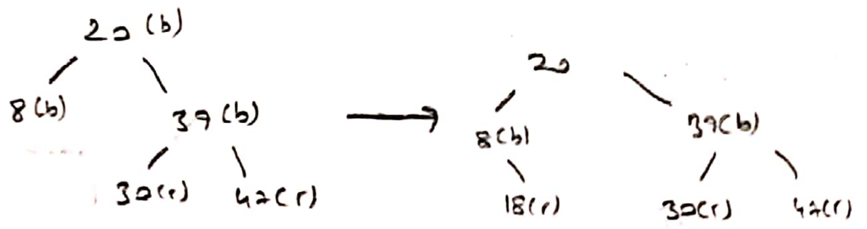
*There is violation of rule 3.

→ case 3 can be applied.



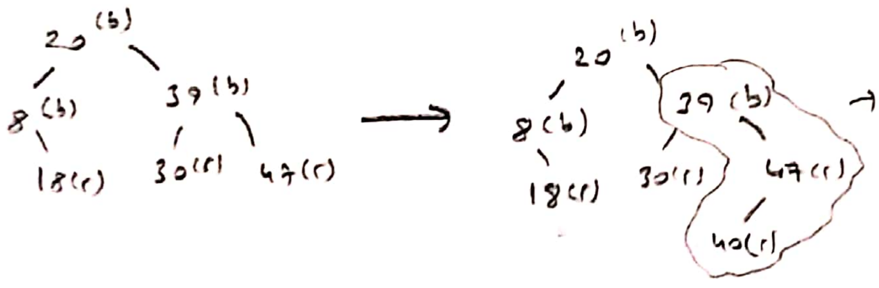
balanced

add(18)

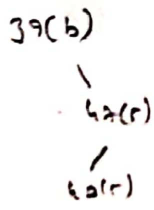


(balanced)

add(40)

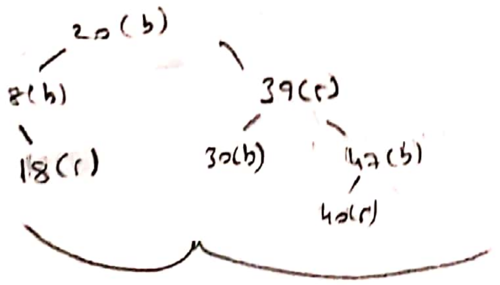


→ There is violation of rule number 3.



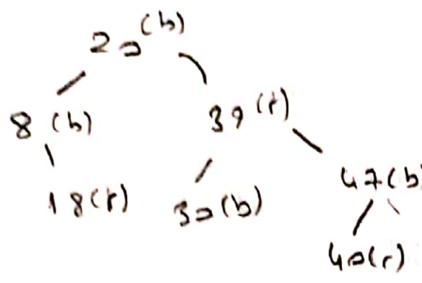
*It looks like case1 coz the grand-parent or 40(r) has black sub-tree. But not same as case1

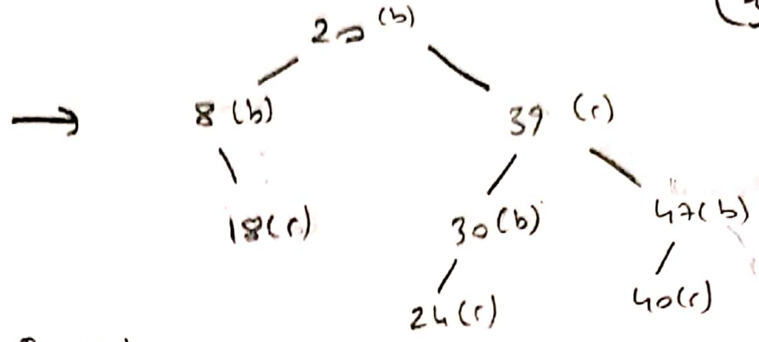
$\boxed{\text{* colouring has to be applied.}}$



balanced

add (24)





* It is obvious that tree that I get is equivalent to tree that construct in AVL tree. It is exciting.

balanced

(Final Tree)

* In removing process of RedBlackTree, there are several cases. Remove a node only if it is leaf or it has only one child. otherwise the node that contains inorder predecessor of the value being removed is the one that is removed.

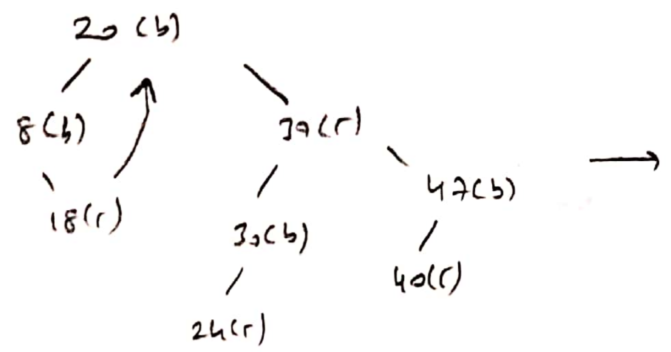
* If the node that is removed is Red, nothing further must be done. Because Red nodes do not affect the ReBlackTree balance. If the node to be removed is black and has a red child then the red child takes it place, and we color it black

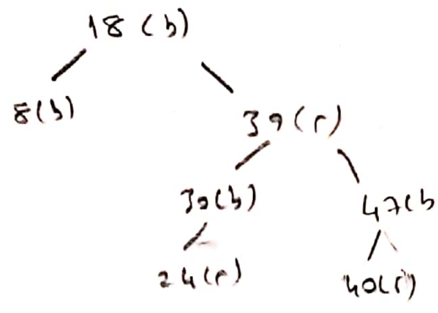
* If we remove a black leaf, then the black height is now out of balance there are several cases must be done, these cases are explained in our book on page 485.

* Steps that our below made according to these steps.

remove (20):

* We need to examine the color of inorder predecessor which is 18(r) the color of the predecessor is red, according to algorithm we do not have to do anything other than move the value of predecessor to root.

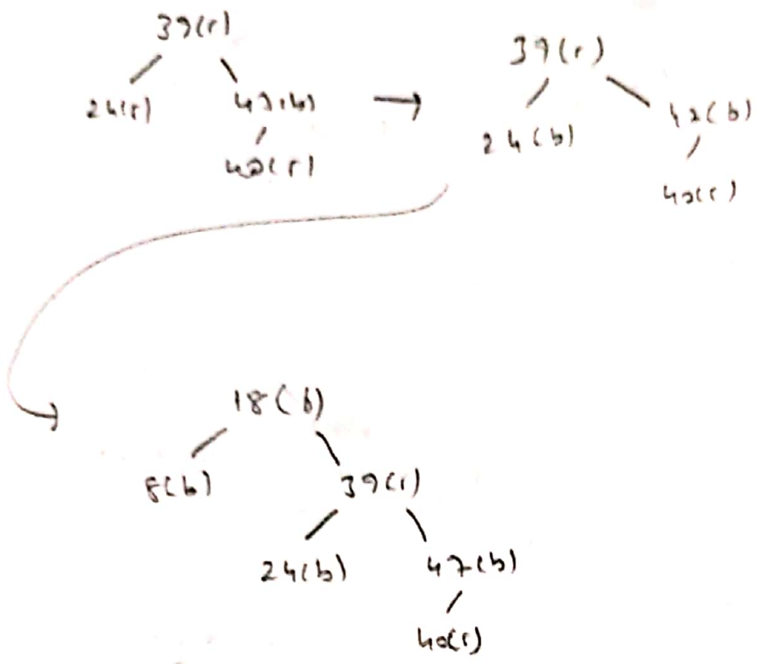




balanced

remove(30)

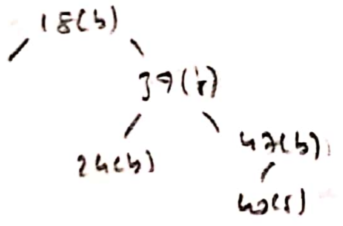
* 30(r) is a red node. Therefore, we need to just delete 30(r) and link left and right child to its parent node. Finding inorder predecessor will be helpful



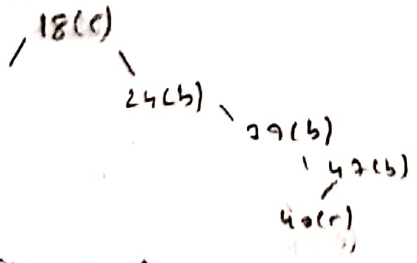
remove(8):

* This deletion looks like case(4) in the book.

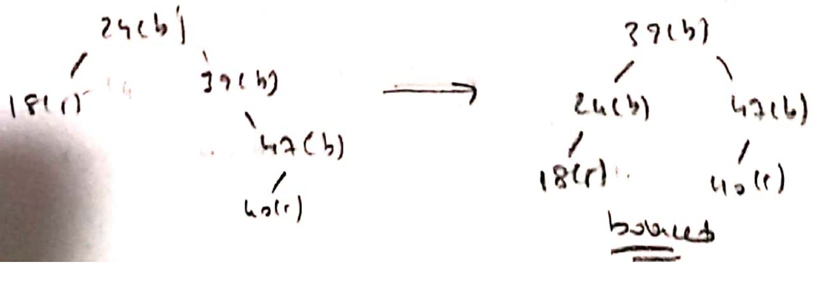
*Firstly we need to recolor the sibling of parent which is 18(b) to black



* After that we need to rotate the sibling right.! and change the color of 18 to red.



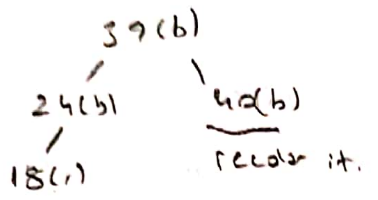
* After that we need to rotate the parent two times left.



remove (47)

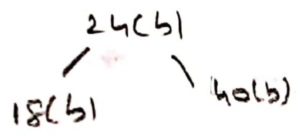
* parent and the other child is black so this case looks like case 5 in the book.

* We change the color of the parent to red after deletion.

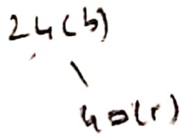


remove (39)

* Inorder predecessor of the 39 is 24 so we can safely move 24 to root, also we need to change the colour of the 18 for balancing tree.



remove (18)



* we delete the 18(b) and need to recolor the left subtree.

remove (40)

24(b) * we just remove the 40 because it is red node.

remove (20)

— * Deletion of root node.