Example 1: Delete 30 from the RB tree in fig. 3

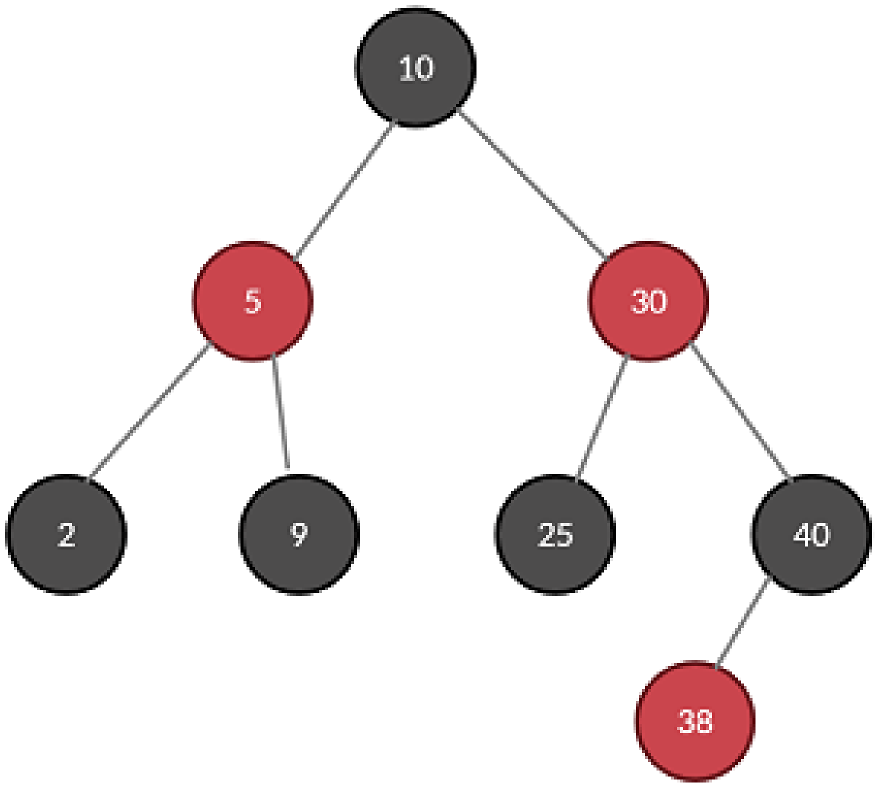

Fig. 3: Initial RB Tree

- First have to search for 30, once found perform BST deletion. For a node with value ‘30’, find either the maximum of the left subtree or a minimum of the right subtree and replace 30 with that value. This is BST deletion

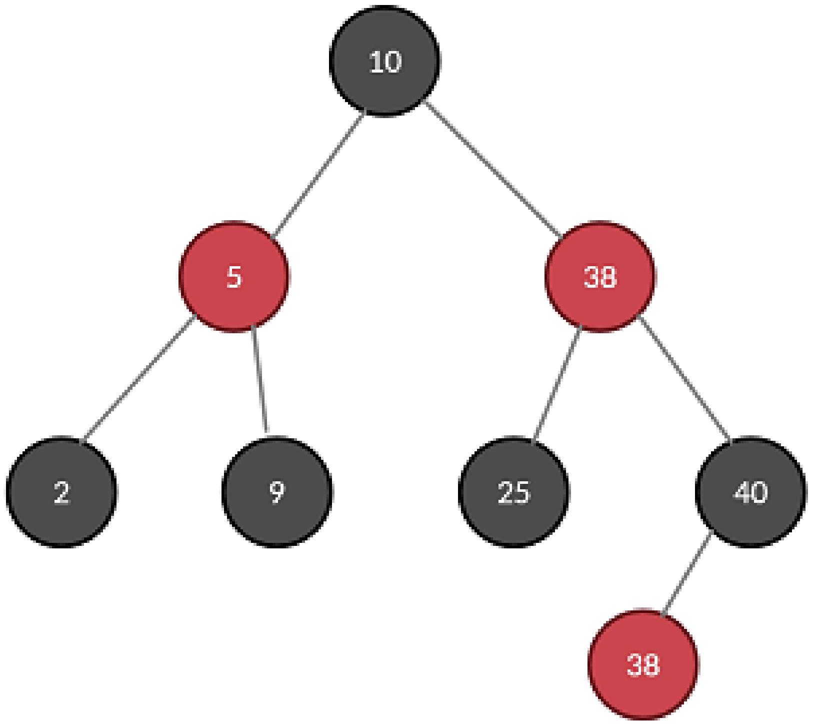

Fig. 4: RB Tree after replacing 30 with min element from right subtree

The resulting RB tree will be like one in fig. 4. Element 30 is deleted and the value is successfully replaced by 38. But now the task is to delete duplicate element 38.

Go to the table above and you'll observe case 1 is satisfied by this tree.

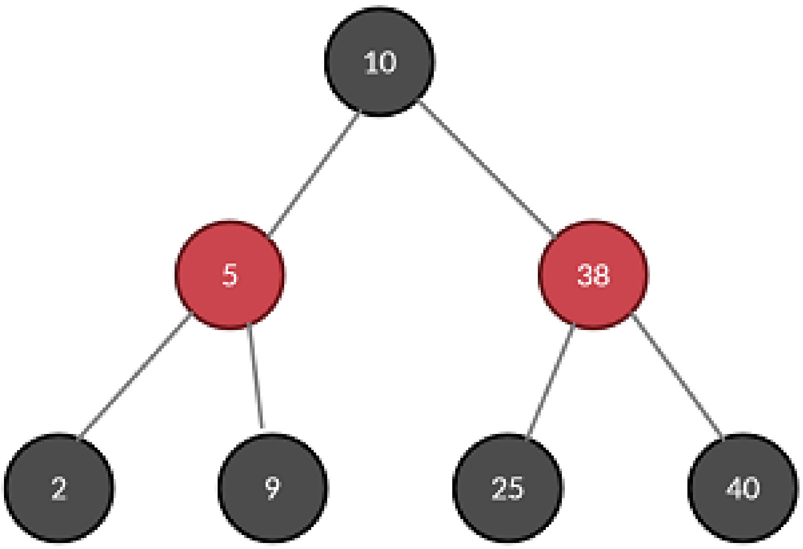

Fig. 5: After removing the red leaf node

Since node with element 38 is a *red* leaf node, remove it and the tree looks like the one in fig. 5.

Example 2: Delete 15 from RB tree in fig. 6

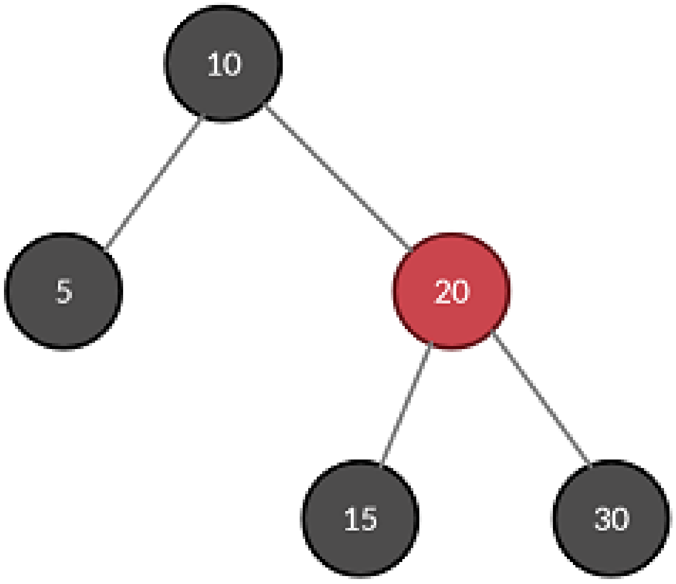

Fig. 6: Initial RB Tree

15 can be removed easily from the tree (BST deletion). In the case of RB trees, if a black leaf node is deleted you replace it with a **double black (DB)** nil node (fig. 7). It is represented by a double circle.

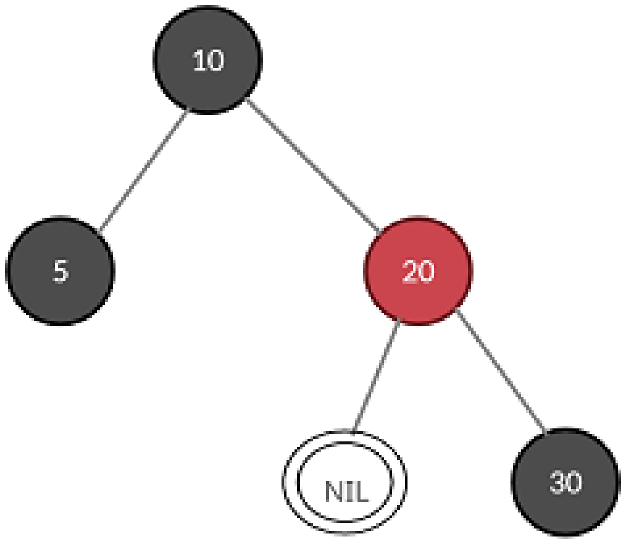

Fig. 7: NIL node added in place of 15

Go back to our rule book (table) and case 3 fits perfectly.

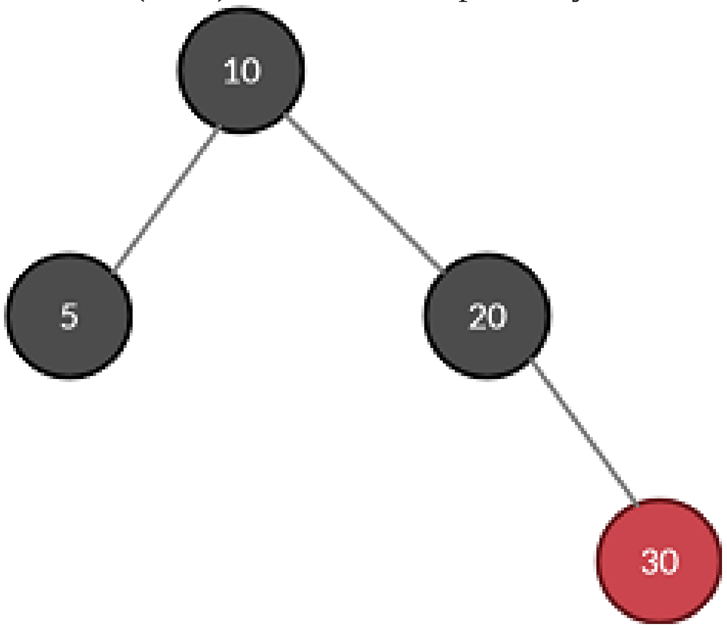

Fig. 8: NIL Node removed after applying actions

In short, remove DB and then swap the color of its sibling with its parent (fig. 8).

Example 3: Delete ‘15’ from fig. 9(A).

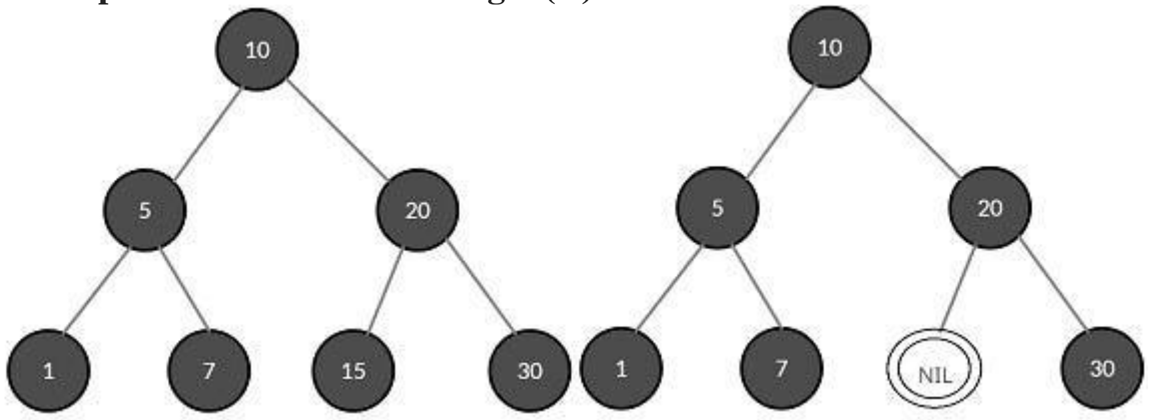

Fig. 9: (A) Initial RB Tree, (B) NIL node added in place of 15

Delete node with value 15 and, as a rule, replace it with DB nil node as shown. Now, DB’s sibling is black and sibling’s both children are also black (don’t forget the hidden NIL nodes), it satisfies all the conditions of case 3. Here,

1. DB’s parent is 20
2. DB’s parent is *black*
3. DB’s sibling is 30 (black)

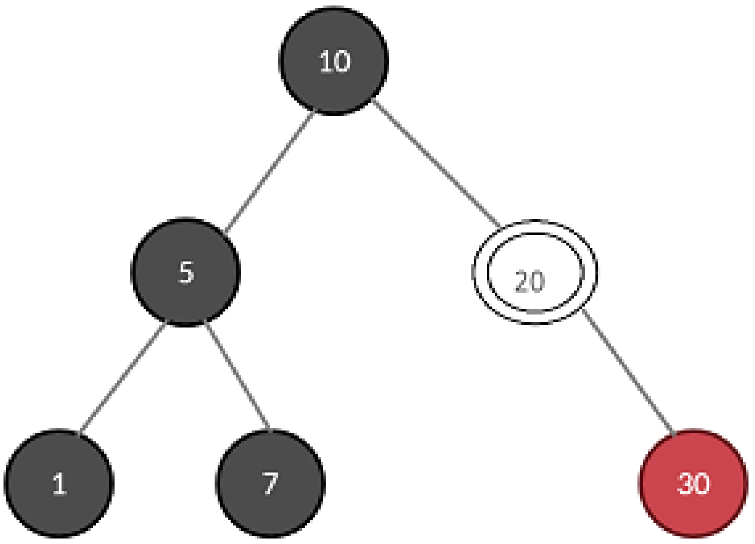

Fig. 10: RB Tree after case 3 is applied

20 becomes DB and hence the problem is not resolved yet. Reapply case 3

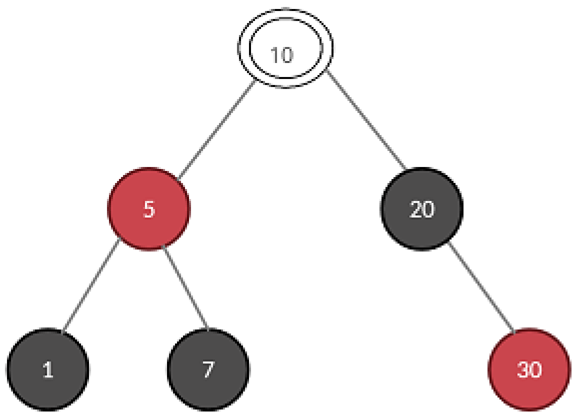

Fig. 11: RB Tree after case 3 is applied

The resulting tree looks like the one in fig. 11.

The DB still exists. Recheck which case will be applicable. (case 2)

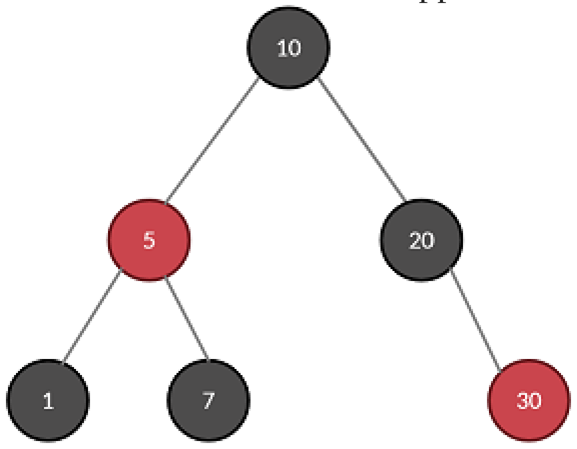

Fig. 12: NIL Node removed after applying actions

The root resolved DB and becomes a *black* node. And you’re done deleting 15 successfully.

Example 4: Delete ‘15’ from fig. 13

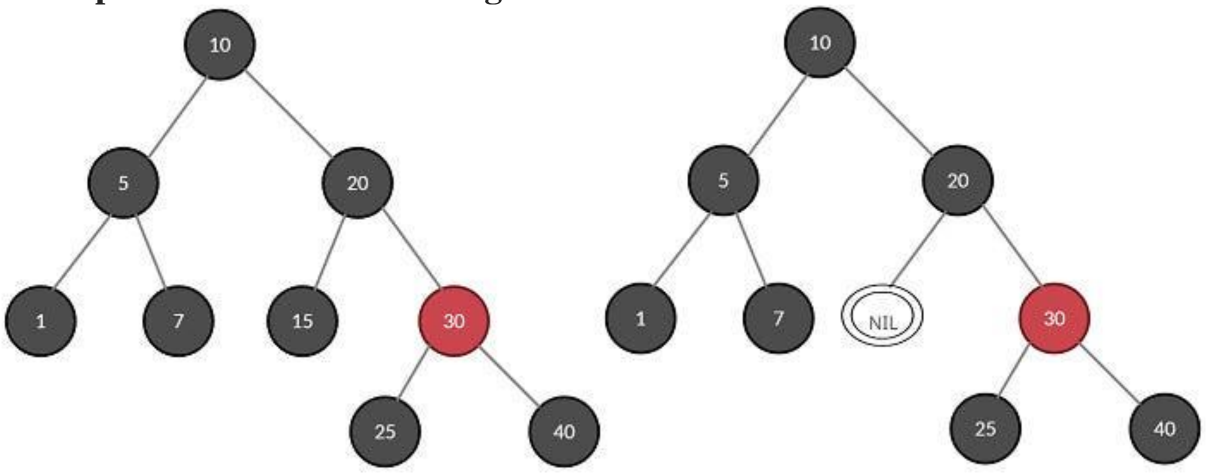

Fig. 13: (A) Initial RB Tree, (B) NIL node added in place of 15

First, Search 15 as per BST rules and then delete it. Second, replace deleted node with DB NIL node as shown in fig. 13 (B).

DB’s sibling is *red*. Clearly, case 4 is applicable.

(a) Swap DB’s parent’s color with DB’s sibling’s color. The tree looks like fig. 14.

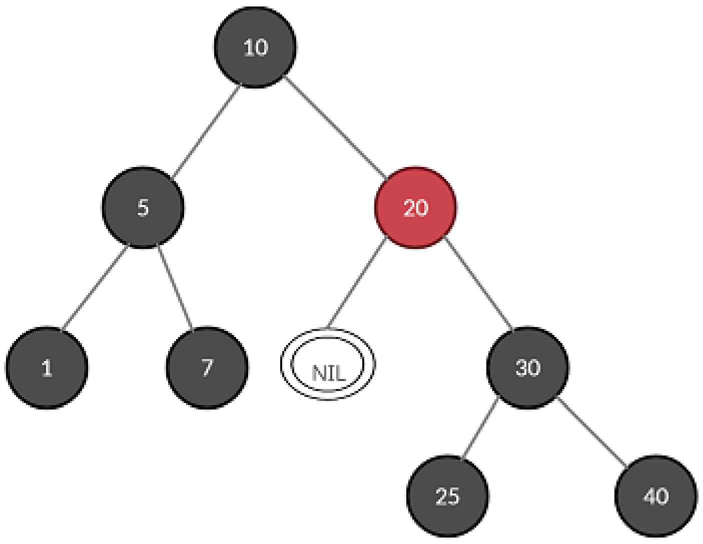

Fig. 14: RB Tree after case 4 is applied

(b) Perform rotation at parent node in direction of DB. The tree becomes like the one in fig. 15. DB is still there.

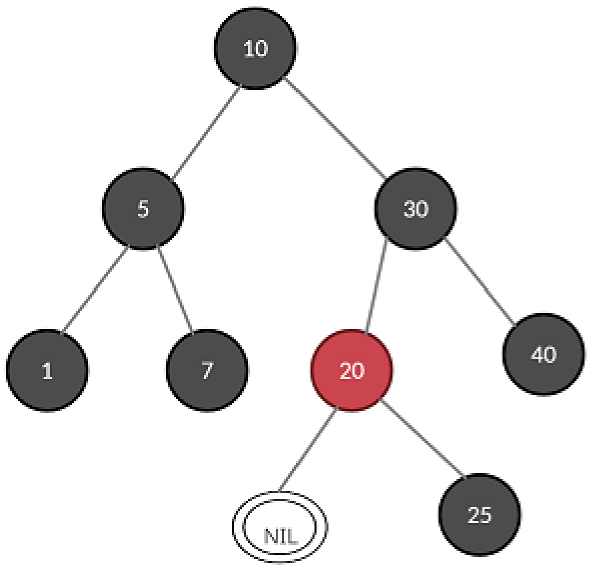

Fig. 15

(c) Check which case can be applied in the current tree. It is case 3

(d) Apply case 3 as explained and the RB tree is free from the DB node as shown in fig. 16.

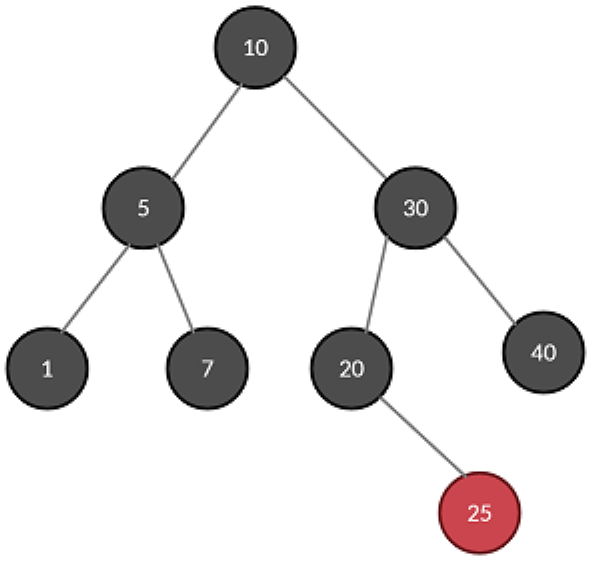

Fig. 16: NIL Node removed after applying actions

Example 5: Delete '1' from fig. 17

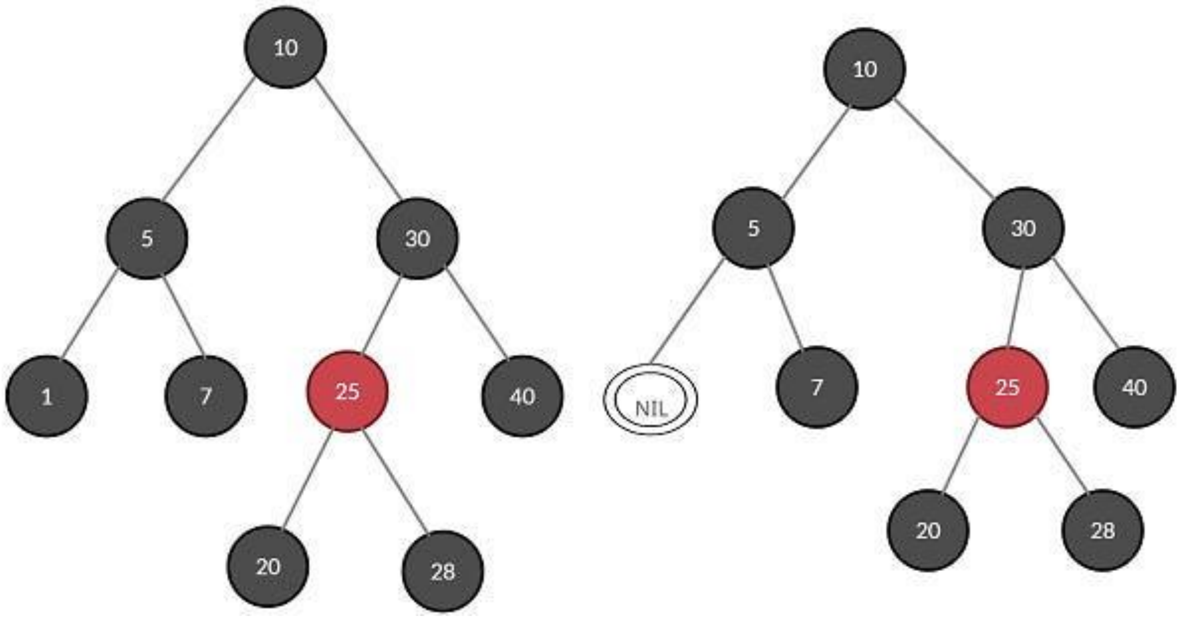

Fig. 17: (A) Initial RB Tree, (B) NIL node added in place of 1

Perform the basic preliminary steps- delete the node with value 1 and replace it with DB NIL node as shown in fig. 17(B). Check for the cases which fit the current tree and it's case 3(DB's sibling is *black*).

Apply case 3 as discussed above and you get the tree as shown in fig. 18.

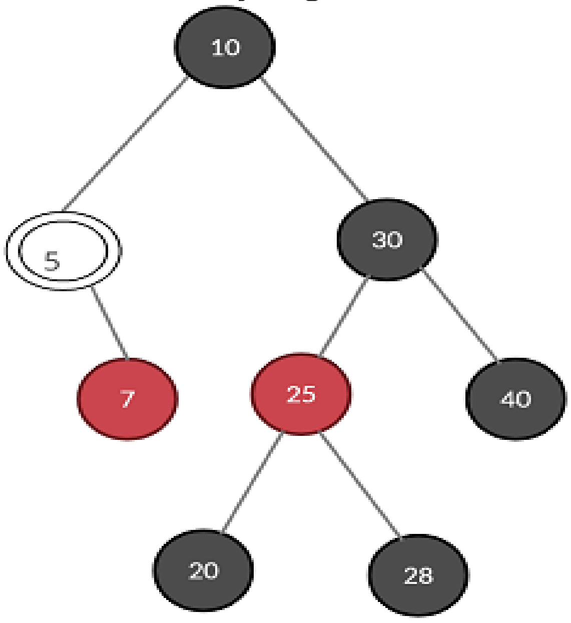

Fig. 18: RB Tree after case 3 is applied

Node 5 has now become a *double black* node. We need to get rid of it.

Search for cases that can be applied and case 5 seems to fit here (not case 3).

Case 5 is applied as follows-

(a) swap colors of nodes 30 and 25 (sibling and red child)(fig. 19(A))

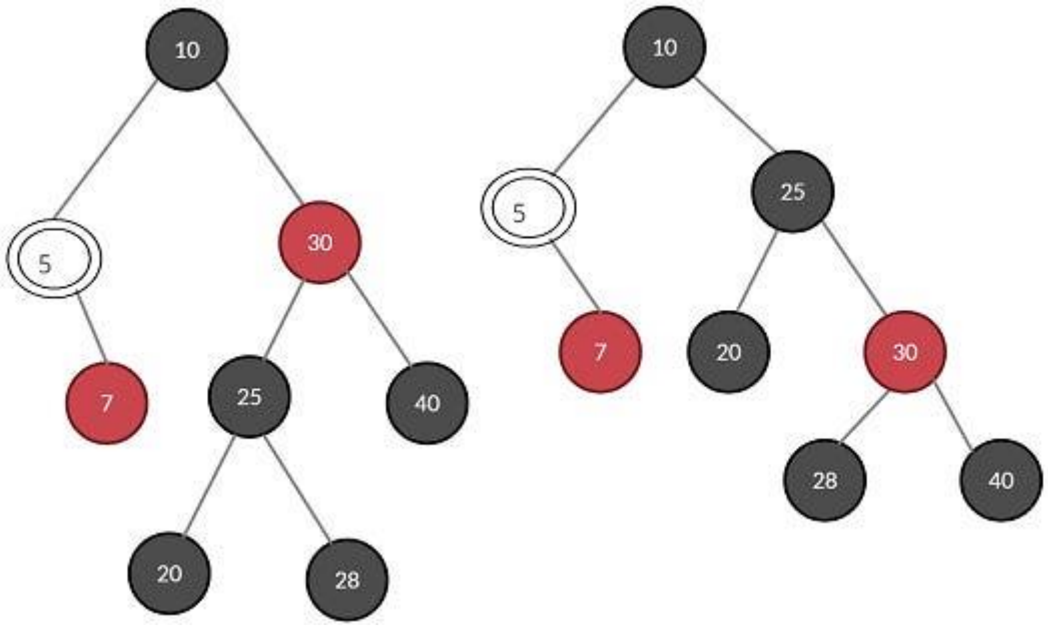

Fig. 19: (A) Tree after swapping colors of 30 & 25 (B) Tree after rotation

(b) Rotate at sibling node in the direction opposite to the DB node. Hence, perform right rotation at node 30 and the tree becomes like fig. 19 (B).

The *double black* node still haunts the tree! Re-check the case that can be applied to this tree and we find that case 6 seems to fit.

Apply case 6 as follow-

(a) Swap colors of DB’s parent with DB’s sibling.

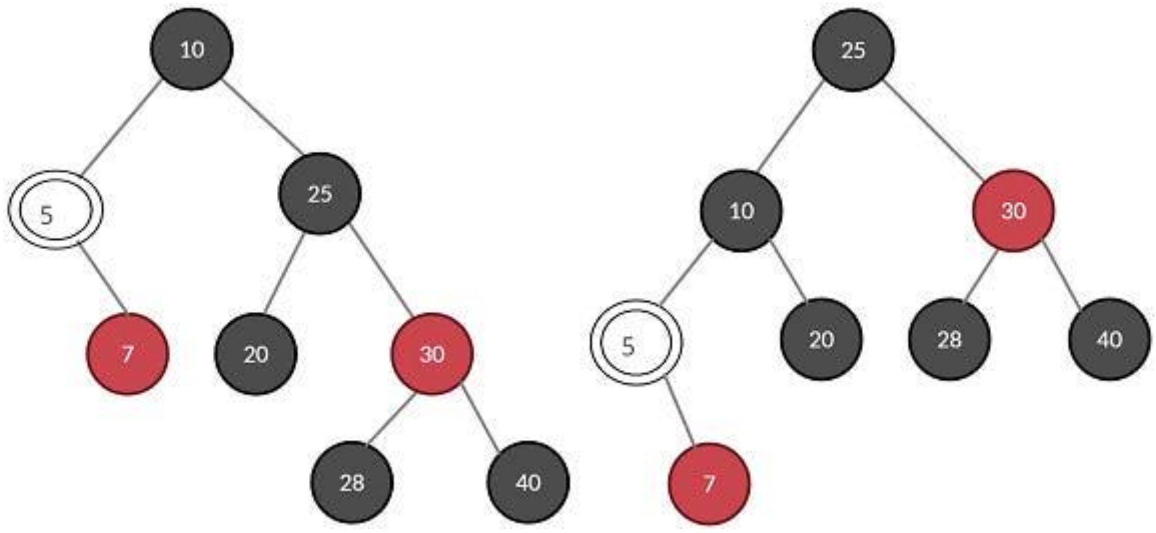

Fig. 20

(b) Perform rotation at DB’s parent node in the direction of DB (fig, 20(B)).

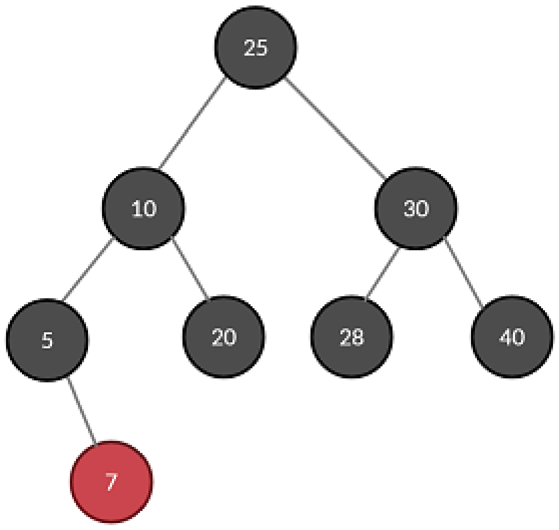

Fig. 21: NIL Node removed after applying actions

(c) Change DB node to *black* node. Also, change the color of DB’s sibling’s far-*red* child to black and the final RB tree will look fig. 21.