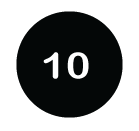
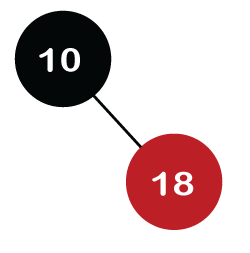
**The red-Black tree** is a binary search tree. The prerequisite of the red-black tree is that we should know about the binary search tree. In a binary search tree, the values of the nodes in the left subtree should be less than the value of the root node, and the values of the nodes in the right subtree should be greater than the value of the root node.

Properties of Red-Black tree

* It is a self-balancing Binary Search tree. Here, self-balancing means that it balances the tree itself by either doing the rotations or recoloring the nodes.
* This tree data structure is named as a Red-Black tree as each node is either Red or Black in color. Every node stores one extra information known as a bit that represents the color of the node. For example, 0 bit denotes the black color while 1 bit denotes the red color of the node. Other information stored by the node is similar to the binary tree, i.e., data part, left pointer and right pointer.
* In the Red-Black tree, the root node is always black in color.
* In a binary tree, we consider those nodes as the leaf which have no child. In contrast, in the Red-Black tree, the nodes that have no child are considered the internal nodes and these nodes are connected to the NIL nodes that are always black in color. The NIL nodes are the leaf nodes in the Red-Black tree.
* If the node is Red, then its children should be in Black color. In other words, we can say that there should be no red-red parent-child relationship.
* Every path from a node to any of its descendant's NIL node should have same number of black nodes.
* **Let's understand the insertion in the Red-Black tree.**
* **10, 18, 7, 15, 16, 30, 25, 40, 60**
* **Step 1:** Initially, the tree is empty, so we create a new node having value 10. This is the first node of the tree, so it would be the root node of the tree. As we already discussed, that root node must be black in color, which is shown below:
* 

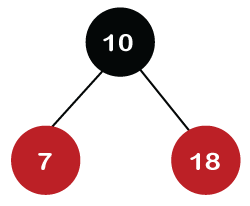
**Step 2:** The next node is 18. As 18 is greater than 10 so it will come at the right of 10 as shown below.



We know the second rule of the Red Black tree that if the tree is not empty then the newly created node will have the **Red** color. Therefore, node 18 has a Red color, as shown in the below figure:

Now we verify the third rule of the Red-Black tree, i.e., the parent of the new node is black or not. In the above figure, the parent of the node is black in color; therefore, it is a Red-Black tree.

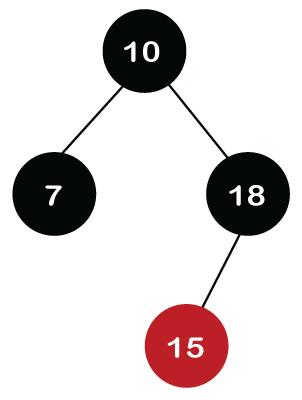
**Step 3:** Now, we create the new node having value 7 with Red color. As 7 is less than 10, so it will come at the left of 10 as shown below.



Now we verify the third rule of the Red-Black tree, i.e., the parent of the new node is black or not. As we can observe, the parent of the node 7 is black in color, and it obeys the Red-Black tree's properties.

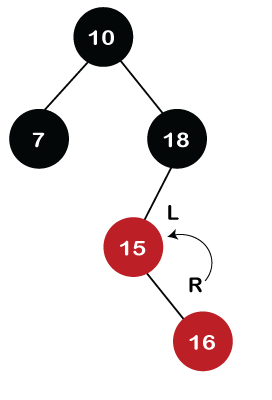
**Step 4:** The next element is 15, and 15 is greater than 10, but less than 18, so the new node will be created at the left of node 18. The node 15 would be Red in color as the tree is not empty.

The above tree violates the property of the Red-Black tree as it has Red-red parent-child relationship. Now we have to apply some rule to make a Red-Black tree. The rule 4 says that ***if the new node's parent is Red, then we have to check the color of the parent's sibling of a new node.*** The new node is node 15; the parent of the new node is node 18 and the sibling of the parent node is node 7. As the color of the parent's sibling is Red in color, so we apply the rule 4a. The rule 4a says that we have to recolor both the parent and parent's sibling node. So, both the nodes, i.e., 7 and 18, would be recolored as shown in the below figure.



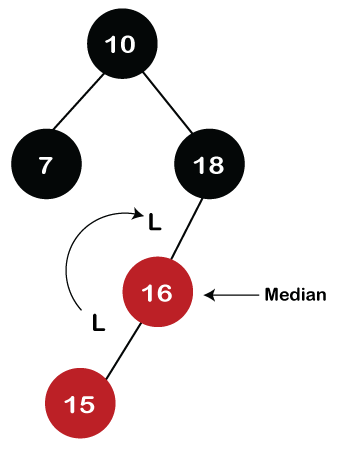
We also have to check whether the parent's parent of the new node is the root node or not. As we can observe in the above figure, the parent's parent of a new node is the root node, so we do not need to recolor it.

**Step 5:** The next element is 16. As 16 is greater than 10 but less than 18 and greater than 15, so node 16 will come at the right of node 15. The tree is not empty; node 16 would be Red in color, as shown in the below figure:

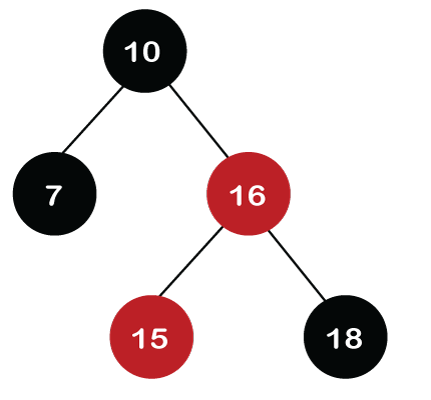


In the above figure, we can observe that it violates the property of the parent-child relationship as it has a red-red parent-child relationship. We have to apply some rules to make a Red-Black tree. Since the new node's parent is Red color, and the parent of the new node has no sibling, so rule **4a** will be applied. The rule **4a** says that some rotations and recoloring would be performed on the tree.

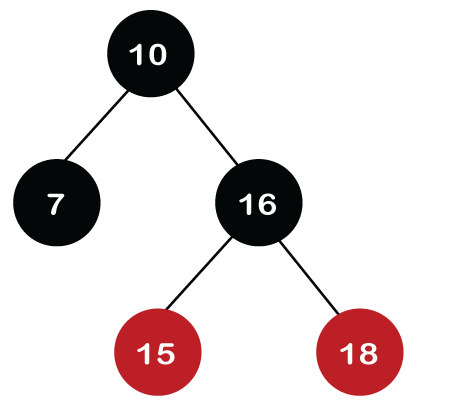
Since node 16 is right of node 15 and the parent of node 15 is node 18. Node 15 is the left of node 18. Here we have **an LR** relationship, so we require to perform two rotations. First, we will perform left, and then we will perform the right rotation. The left rotation would be performed on nodes 15 and 16, where node 16 will move upward, and node 15 will move downward. Once the left rotation is performed, the tree looks like as shown in the below figure:



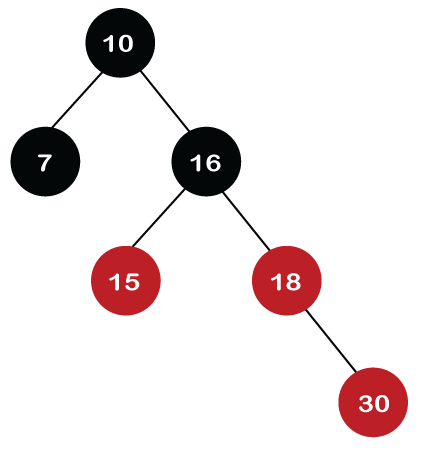
In the above figure, we can observe that there is **an LL** relationship. The above tree has a Red-red conflict, so we perform the right rotation. When we perform the right rotation, the median element would be the root node. Once the right rotation is performed, node 16 would become the root node, and nodes 15 and 18 would be the left child and right child, respectively, as shown in the below figure.



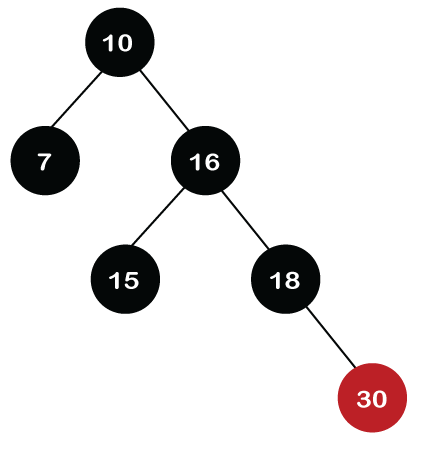
After rotation, node 16 and node 18 would be recolored; the color of node 16 is red, so it will change to black, and the color of node 18 is black, so it will change to a red color as shown in the below figure:



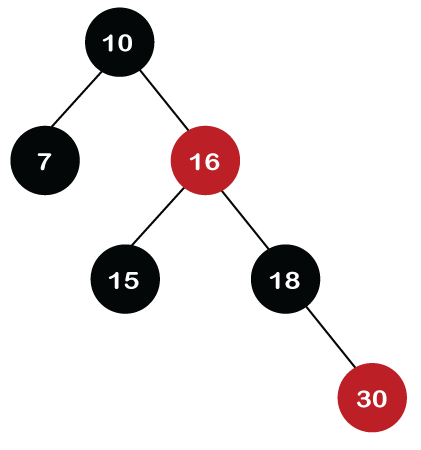
**Step 6:** The next element is 30. Node 30 is inserted at the right of node 18. As the tree is not empty, so the color of node 30 would be red.



The color of the parent and parent's sibling of a new node is Red, so rule 4b is applied. In rule 4b, we have to do only recoloring, i.e., no rotations are required. The color of both the parent (node 18) and parent's sibling (node 15) would become black, as shown in the below image.

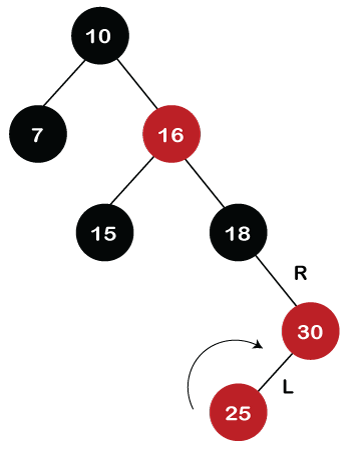


We also have to check the parent's parent of the new node, whether it is a root node or not. The parent's parent of the new node, i.e., node 30 is node 16 and node 16 is not a root node, so we will recolor the node 16 and changes to the Red color. The parent of node 16 is node 10, and it is not in Red color, so there is no Red-red conflict.

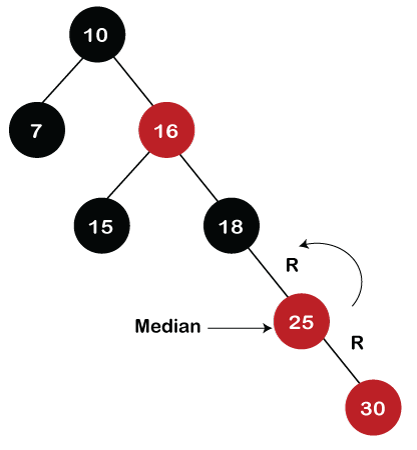


**Step 7:** The next element is 25, which we have to insert in a tree. Since 25 is greater than 10, 16, 18 but less than 30; so, it will come at the left of node 30. As the tree is not empty, node 25 would be in Red color. Here Red-red conflict occurs as the parent of the newly created is Red color.

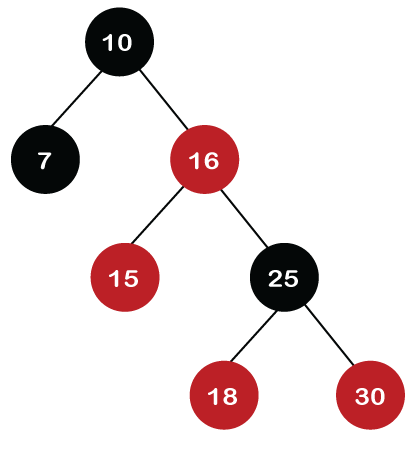
Since there is no parent's sibling, so rule 4a is applied in which rotation, as well as recoloring, are performed. First, we will perform rotations. As the newly created node is at the left of its parent and the parent node is at the right of its parent, so the RL relationship is formed. Firstly, the right rotation is performed in which node 25 goes upwards, whereas node 30 goes downwards, as shown in the below figure.



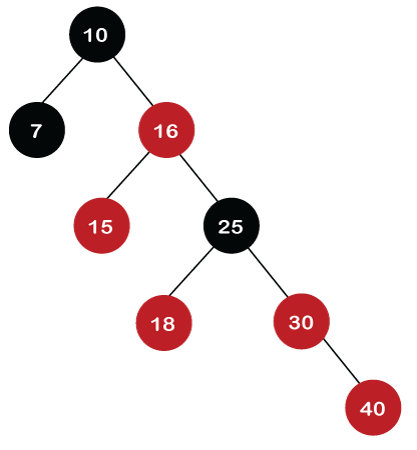
After the first rotation, there is an RR relationship, so left rotation is performed. After right rotation, the median element, i.e., 25 would be the root node; node 30 would be at the right of 25 and node 18 would be at the left of node 25.



Now recoloring would be performed on nodes 25 and 18; node 25 becomes black in color, and node 18 becomes red in color.



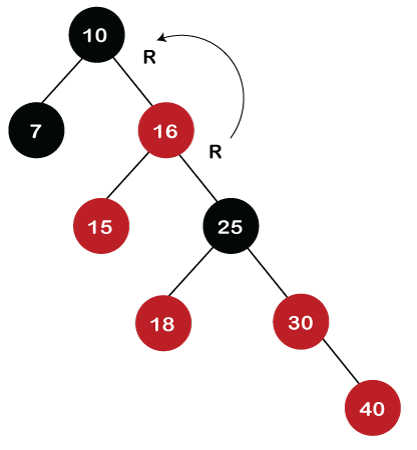
**Step 8:** The next element is 40. Since 40 is greater than 10, 16, 18, 25, and 30, so node 40 will come at the right of node 30. As the tree is not empty, node 40 would be Red in color. There is a Red-red conflict between nodes 40 and 30, so rule 4b will be applied.



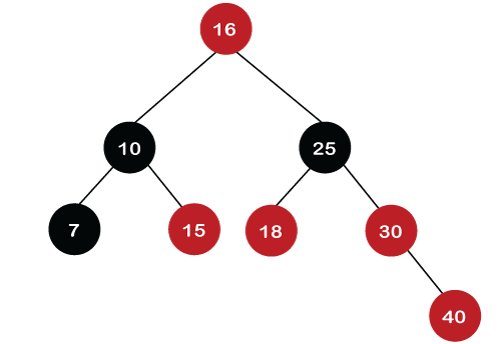
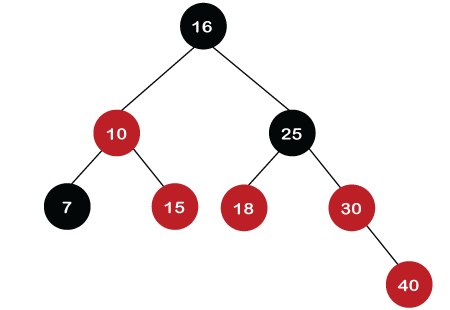
As the color of parent and parent's sibling node of a new node is Red so recoloring would be performed. The color of both the nodes would become black, as shown in the below image.

After recoloring, we also have to check the parent's parent of a new node, i.e., 25, which is not a root node, so recoloring would be performed, and the color of node 25 changes to Red.

After recoloring, red-red conflict occurs between nodes 25 and 16. Now node 25 would be considered as the new node. Since the parent of node 25 is red in color, and the parent's sibling is black in color, rule 4a would be applied. Since 25 is at the right of the node 16 and 16 is at the right of its parent, so there is an RR relationship. In the RR relationship, left rotation is performed. After left rotation, the median element 16 would be the root node, as shown in the below figure.

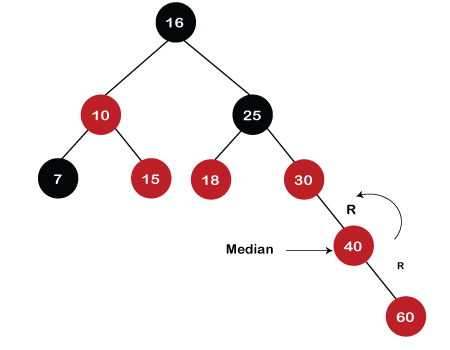


After rotation, recoloring is performed on nodes 16 and 10. The color of node 10 and node 16 changes to Red and Black, respectively as shown in the below figure.

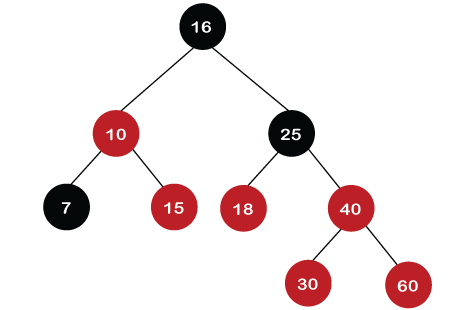
  


**Step 9:** The next element is 60. Since 60 is greater than 16, 25, 30, and 40, so node 60 will come at the right of node 40. As the tree is not empty, the color of node 60 would be Red.

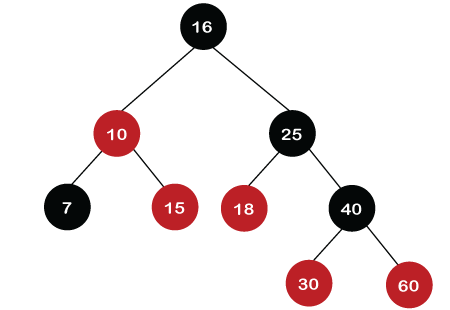
As we can observe in the above tree that there is a Red-red conflict occurs. The parent node is Red in color, and there is no parent's sibling exists in the tree, so rule 4a would be applied. The first rotation would be performed. The RR relationship exists between the nodes, so left rotation would be performed.



When left rotation is performed, node 40 will come upwards, and node 30 will come downwards, as shown in the below figure:



After rotation, the recoloring is performed on nodes 30 and 40. The color of node 30 would become Red, while the color of node 40 would become black.



Deletion in Red Back tree

Let's understand how we can delete the particular node from the Red-Black tree. The following are the rules used to delete the particular node from the tree:

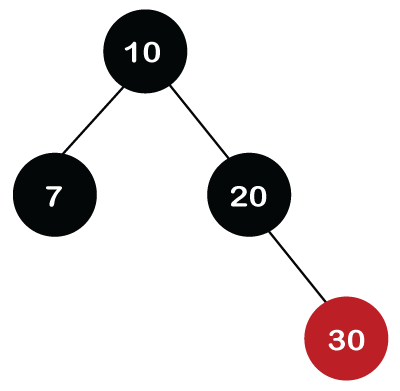
**Step 1:** First, we perform BST rules for the deletion.

**Step 2:**

**Case 1:** if the node is Red, which is to be deleted, we simply delete it.

Let's understand case 1 through an example.

Suppose we want to delete node 30 from the tree, which is given below.



Initially, we are having the address of the root node. First, we will apply BST to search the node. Since 30 is greater than 10 and 20, which means that 30 is the right child of node 20. Node 30 is a leaf node and Red in color, so it is simply deleted from the tree.

If we want to delete the internal node that has one child. First, replace the value of the internal node with the value of the child node and then simply delete the child node.