Red-Black Tree

Red - Black Tree is another variant of Binary Search Tree in which every node is colored either RED or BLACK. The color of a node is decided based on the properties of Red-Black Tree.

1. **Need for Red Black tree:**

- Most of the Binary Search Tree operations take **O(h)** time (where ‘**h**’ is the height of the tree) for example: Search , Max , Min , Insert , Delete etc.
- If Binary Search Tree becomes skewed then height of the tree will become equal to total number of nodes i.e. **‘n’** and complexity will increase to **O(n)**. So to make the complexity low we should balance the Binary Search Tree after each insertion and deletion operation. This will ensure the height **h** of tree as **log n** and complexity as **O(log n)**.
- the height of a Red Black Tree is always **log n**.
- If frequent insertion and deletion are required then Red Black Tree give better performance than AVL Tree. If insertion and deletion are less frequent then AVL tree give good performance because AVL Trees are more balanced than Red Black Trees but they can cause more rotation and can increase time complexity.

2. **Properties of Red Black Tree**

- Red - Black Tree must be a Binary Search Tree.
- The ROOT node must be colored BLACK.
- The children of Red colored node must be colored BLACK. (There should not be two consecutive RED nodes).

- In all the paths of the tree, there should be same number of BLACK colored nodes.
- Every new node must be inserted with RED color.
- Every leaf (e.i. NULL node) must be colored BLACK.

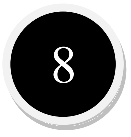
3. Insertion in Red Black Tree :

The insertion operation in Red Black tree is performed using the following steps...

- Step 1 - Check whether tree is Empty.
- Step 2 – if tree is not Empty then insert the new node as leaf node with color Red.
- Step 4 - If the parent of new Node is Black then exit from the operation.
- Step 5 - If the parent of new Node is Red then check the color of parent node's sibling of new Node.
- Step 6 - If it is colored Black or NULL then make suitable Rotation and Recolor it.
- Step 7 - If it is colored Red then perform Recolor.
- Repeat the same until tree becomes Red Black Tree.

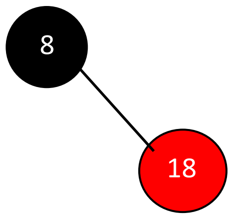
Ex. Create Red-black tree by inserting the following sequence of number:
[8, 18, 5, 15, 17, 25, 40, 80]

Insert 8:
Tree is empty. So insert new node as root node with black color.



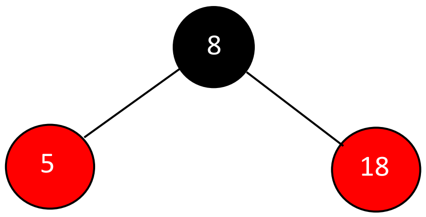
Insert (18)

Tree is not empty. So insert newnode with red color.



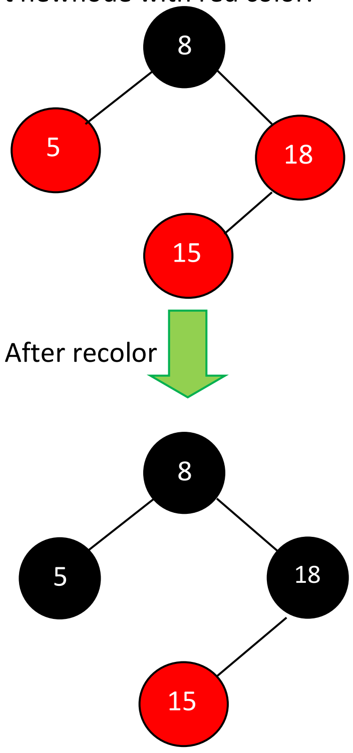
Insert (5)

Tree is not empty. So insert newnode with red color.



Insert (15)

Tree is not empty. So insert newnode with red color.

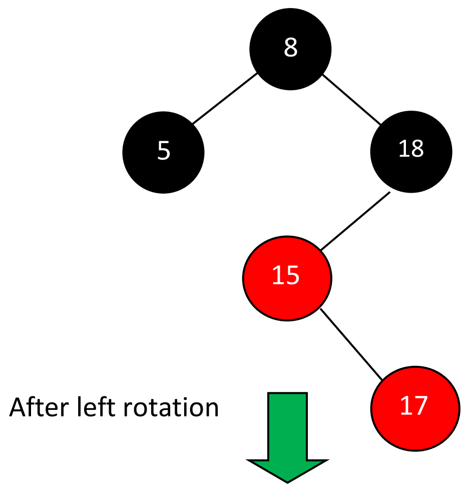


Here there are two consecutive red nodes 18 and 15. The newnode's parent sibling color is red and parent's parent is root node. So we use recolor to make it red black tree.

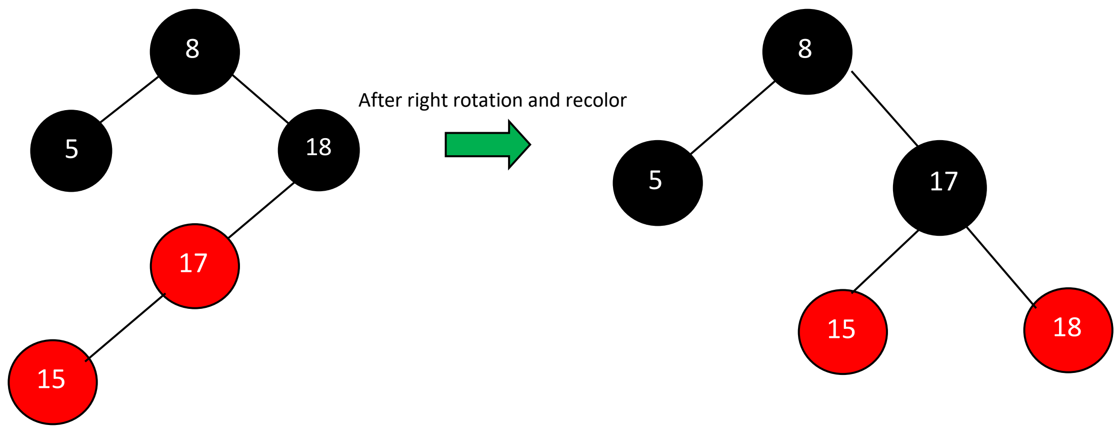
After recolor operation, the trees satisfying all red black tree properties.

Insert (17)

The tree is not empty. So insert newnode with red color.

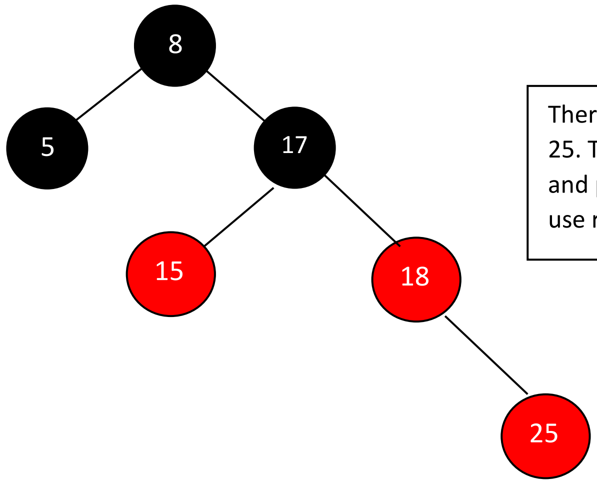


Here there two consecutive red nodes 15 and 17.The newnode's parent sibling is null. So we need rotation. Here we need LR rotation and recolor.



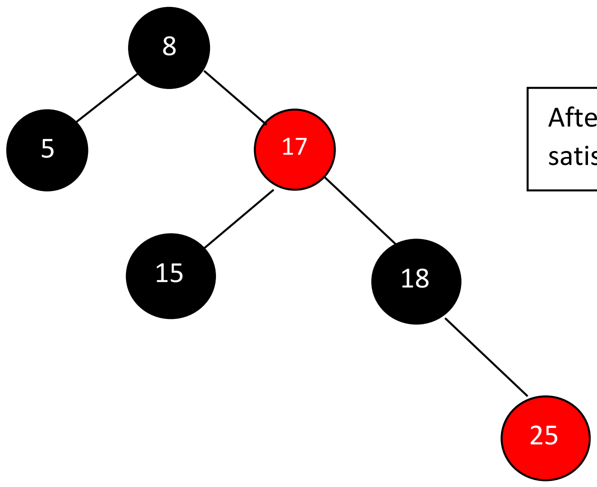
Insert (25)

Tree is not empty. So insert newnode with red color.



There are two consecutive red nodes 18 and 25. The newnode's parent sibling color is red and parent's parent is not root node. So we use recolor and recheck.

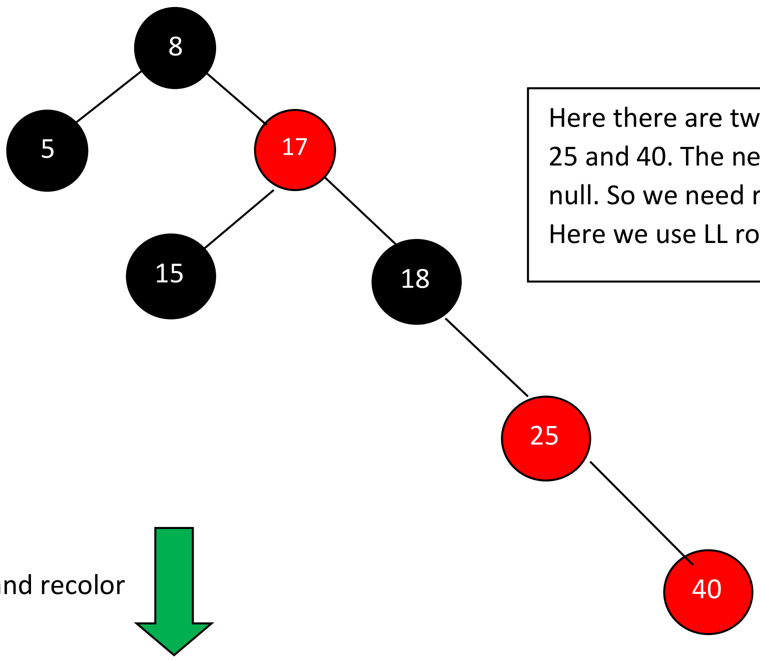
After recolor



After recolor operation, the tree is satisfying all red black tree properties.

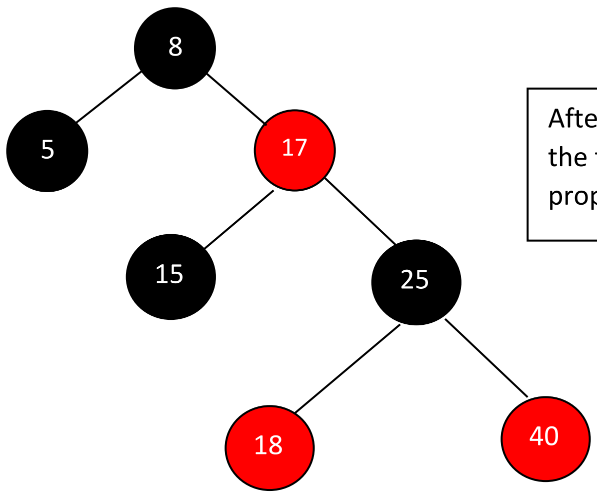
Insert(40)

Tree is not empty. So insert newnode with red color.



Here there are two consecutive red nodes 25 and 40. The newnode's parent sibling is null. So we need rotation and recolor. Here we use LL rotation and recheck.

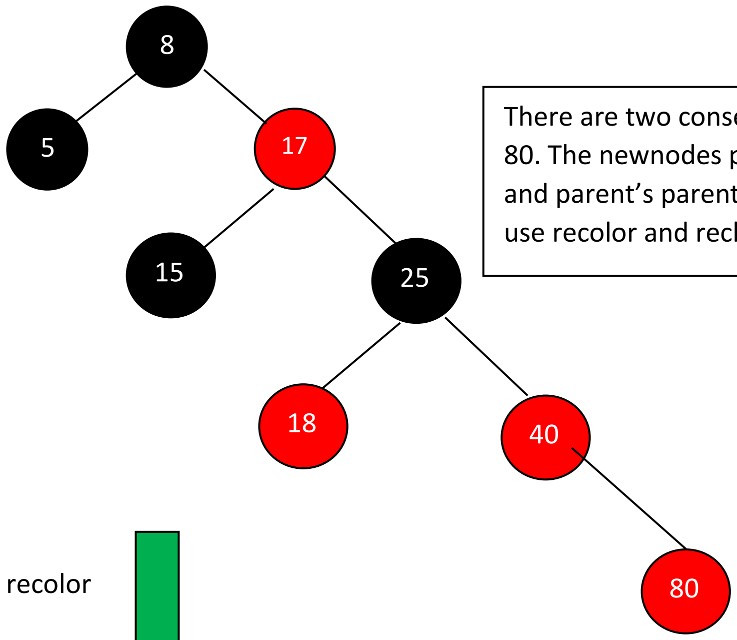
After LL rotation and recolor



After LL rotation and recolor operation, the tree satisfying all red black tree properties.

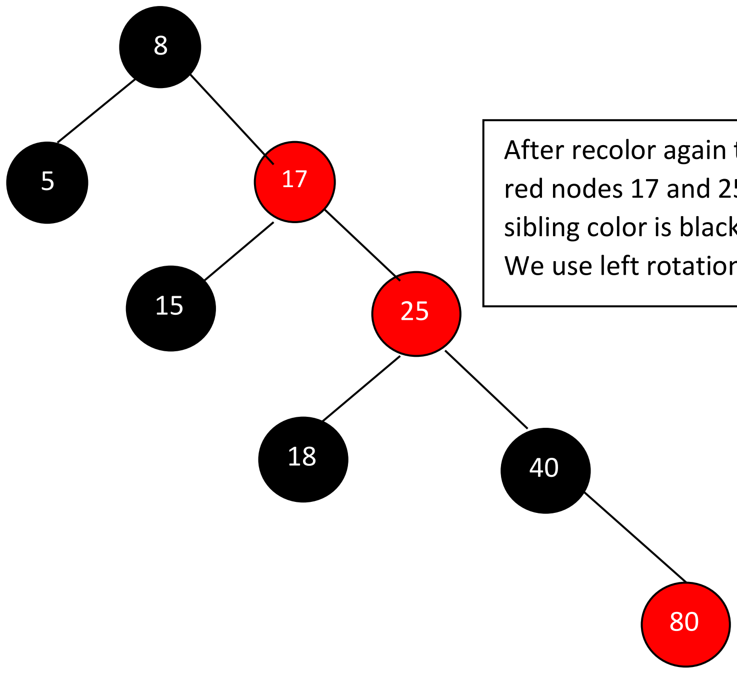
Insert(80)

Tree is not empty. So insert newnode with red color.



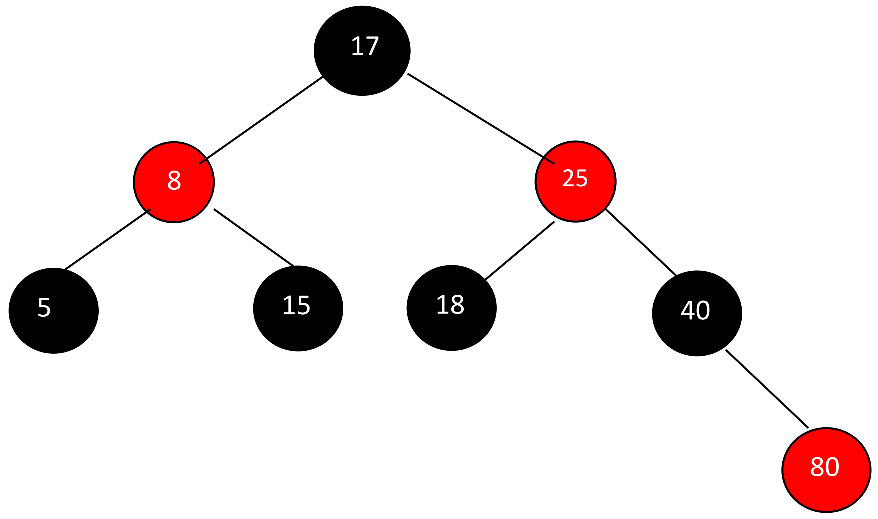
There are two consecutive red nodes 40 and 80. The newnodes parent sibling color is red and parent's parent is not root node. So we use recolor and recheck.

After recolor



After recolor again there are two consecutive red nodes 17 and 25. The newnode's parent sibling color is black. So we need rotation. We use left rotation And recolor.

After left rotation And recolor



Finally above tree is satisfying all the properties of red black tree and it is a perfect red black tree.

Deletion operation in red black tree

The deletion operation in red black tree is similar to deletion operation in BST. But after every deletion operation we need to check with the red black tree properties. If any of the properties violated then make suitable operations like recolor, rotation and rotation followed by recolor to make it red black tree.

Operations on Red Black Tree in details:

Rotations:

A rotation is a local operation in a search tree that preserves *in-order* traversal key ordering.

Note that in both trees, an in-order traversal yields:

```
A x B y C
```

The left_rotate operation may be encoded:

```
left_rotate( Tree T, node x ) {
    node y;
    y = x->right;
    /* Turn y's left sub-tree into x's right sub-tree */
    x->right = y->left;
    if ( y->left != NULL )
        y->left->parent = x;
    /* y's new parent was x's parent */
    y->parent = x->parent;
```