Red-Black Tree | Set 2 (Insert)

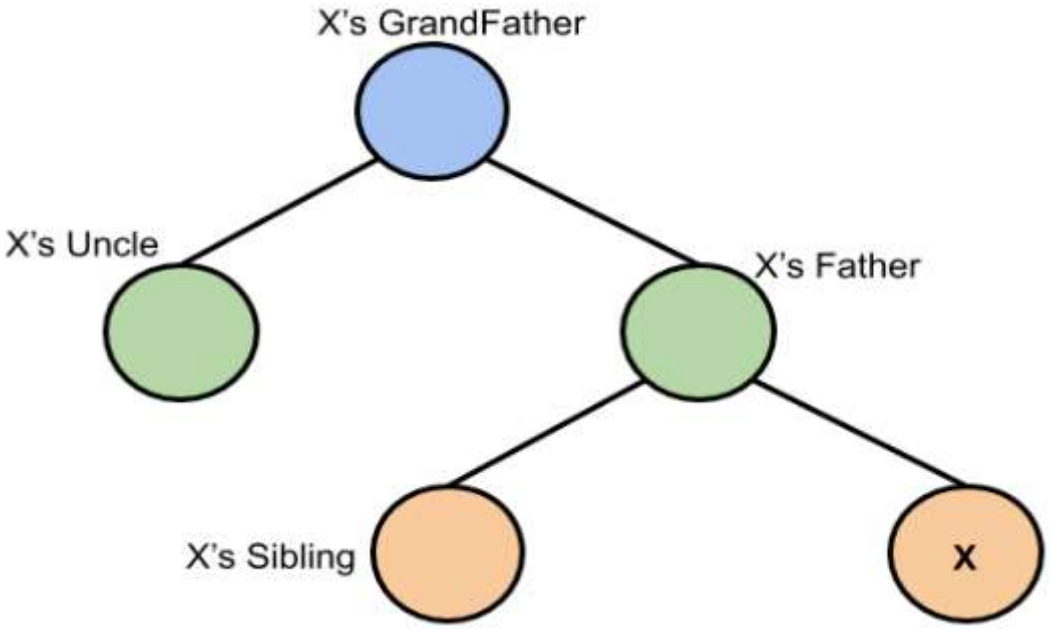
Difficulty Level : Medium • Last Updated : 22 Dec, 2021

In the previous post, we discussed the introduction to Red-Black Trees. In this post, insertion is discussed. In AVL tree insertion, we used rotation as a tool to do balancing after insertion. In the Red-Black tree, we use two tools to do the balancing.

1. Recoloring
2. Rotation

Recolouring is the change in colour of the node i.e. if it is red then change it to black and vice versa. It must be noted that the colour of the NULL node is always black. Moreover, we always try recolouring first, if recolouring doesn't work, then we go for rotation. Following is a detailed algorithm. The algorithms have mainly two cases depending upon the colour of the uncle. If the uncle is red, we do recolour. If the uncle is black, we do rotations and/or recolouring.

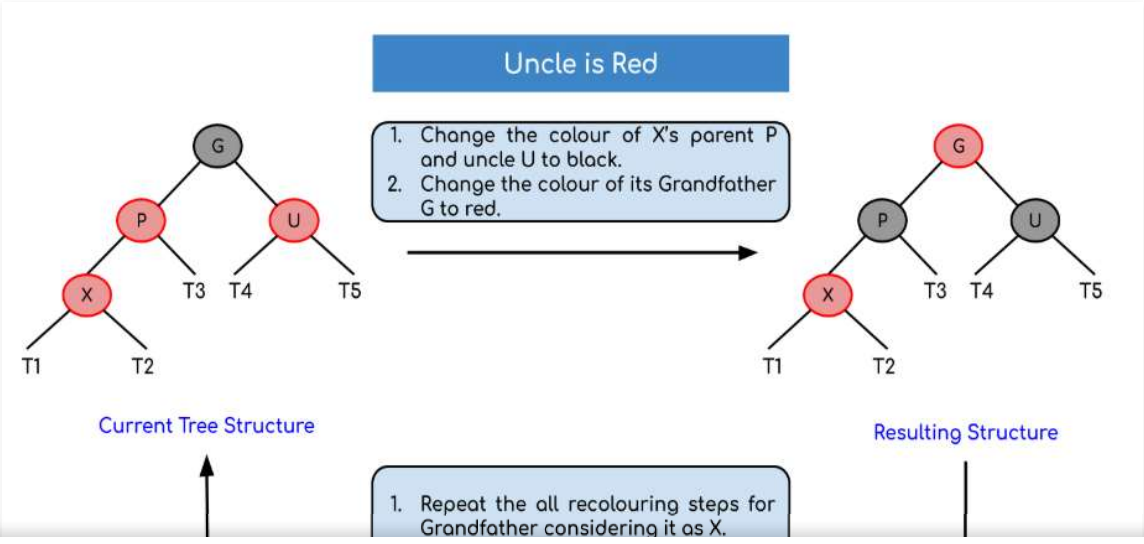
The representation we will be working with is:



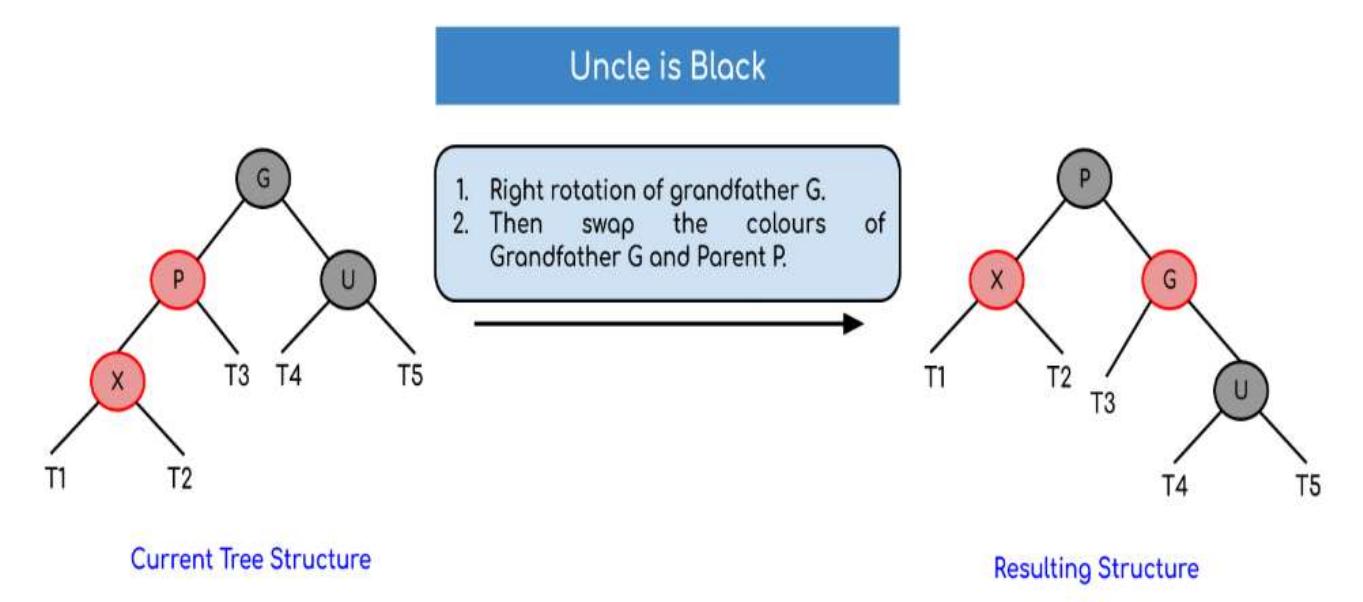
This representation is based on X

Logic:

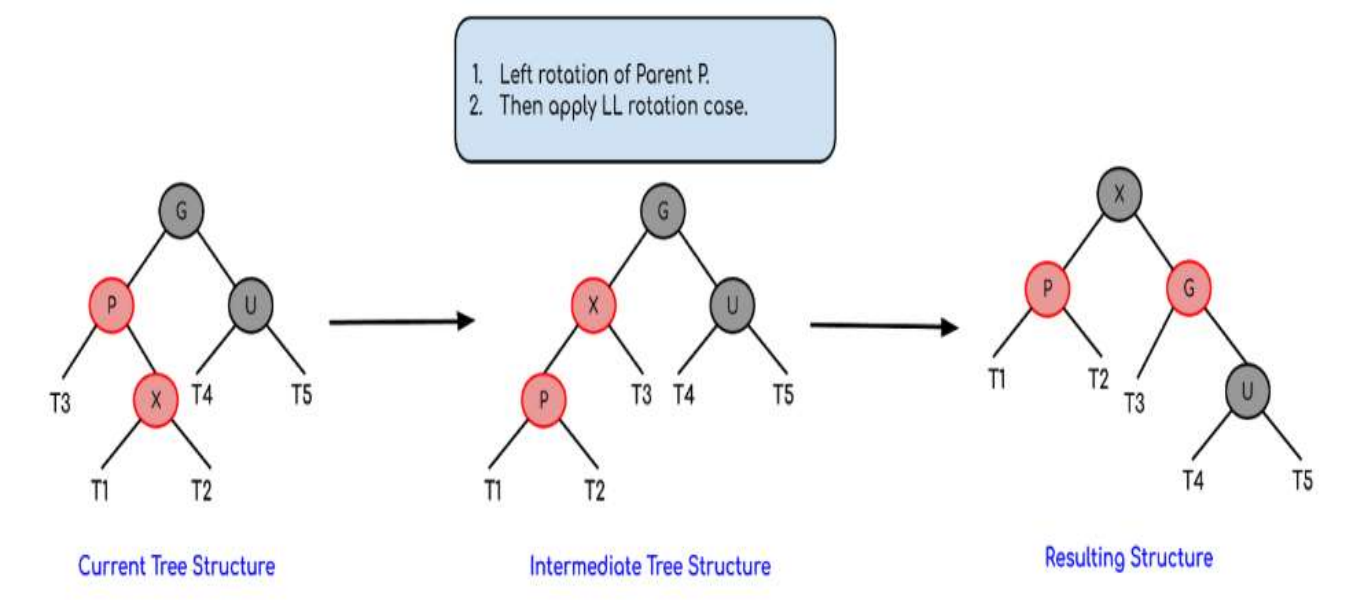
First, you have to insert the node similarly to that in a binary tree and assign a red colour to it. Now, if the node is a root node then change its colour to black, but if it is not then check the colour of the parent node. If its colour is black then don't change the colour but if it is not i.e. it is red then check the colour of the node's uncle. If the node's uncle has a red colour then change the colour of the node's parent and uncle to black and that of grandfather to red colour and repeat the same process for him (i.e. grandfather).



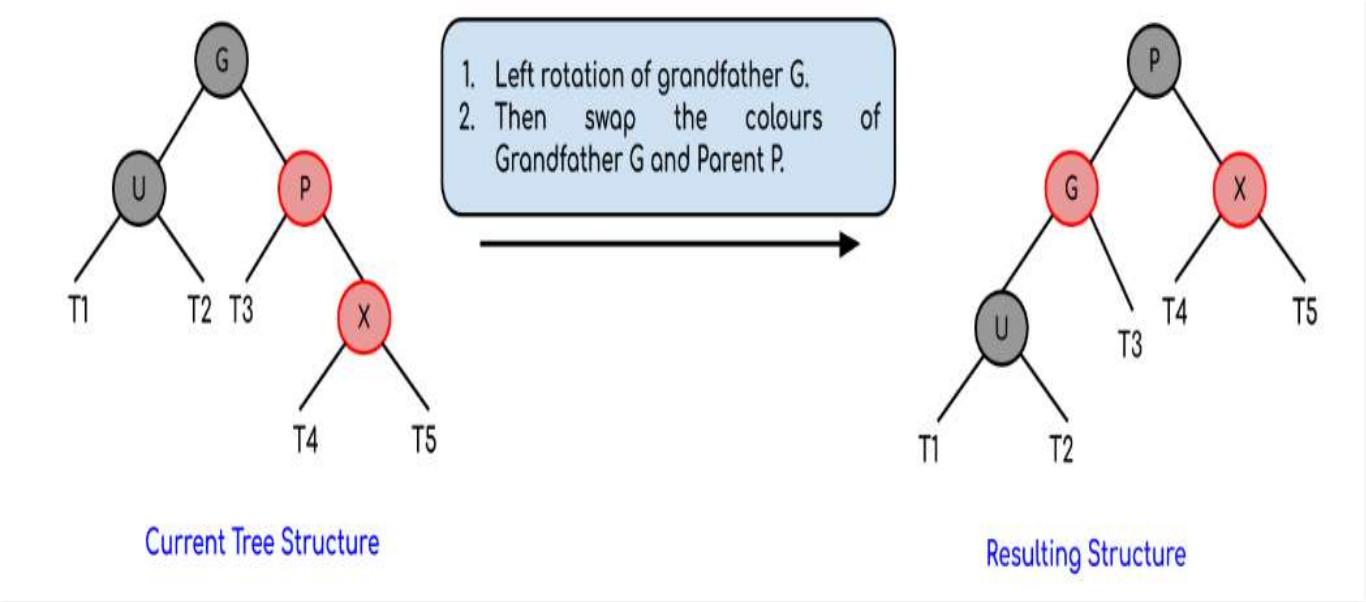
- Left Left Case (LL rotation):



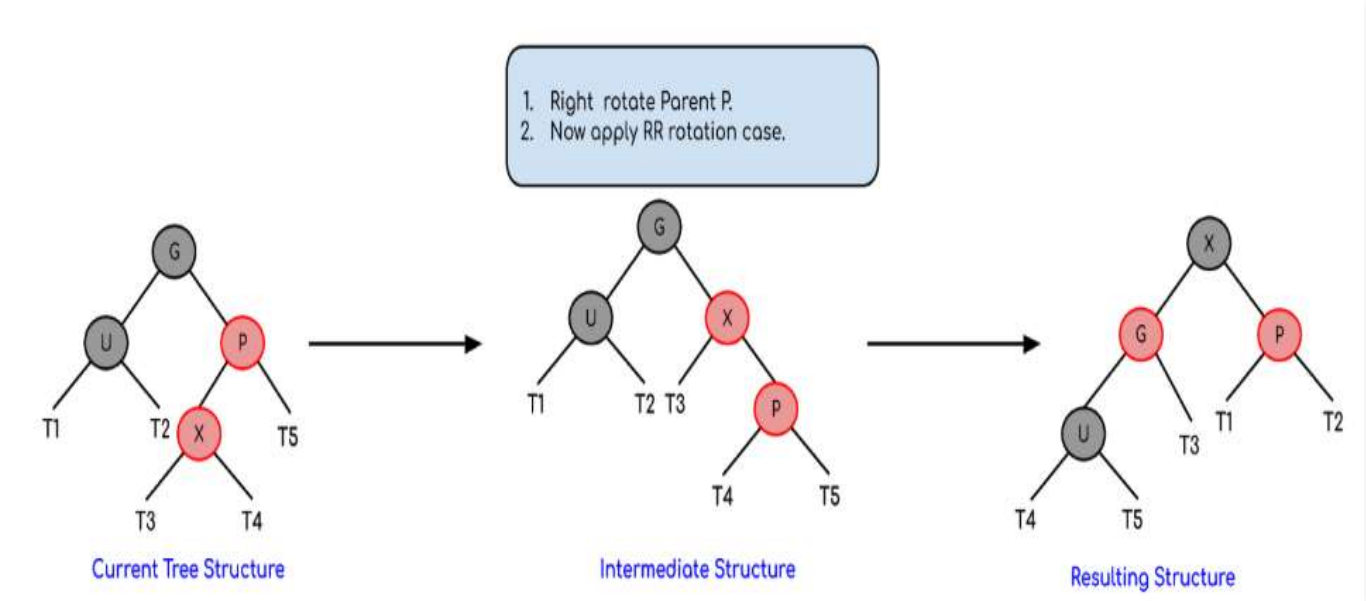
- Left Right Case (LR rotation):



- Right Right Case (RR rotation):



- Right Left Case (RL rotation):



Now, after these rotations, if the colours of the nodes are miss matching then recolour them.

Algorithm:

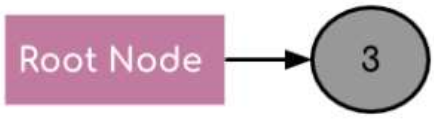
Let x be the newly inserted node.

3. Do the following if the color of x's parent is not BLACK **and** x is not the root.
   a) If x's uncle is RED (Grandparent must have been black from property 4)
   (i) Change the colour of parent and uncle as BLACK.
   (ii) Colour of a grandparent as RED.
   (iii) Change x = x's grandparent, repeat steps 2 and 3 for new x.
   b) If x's uncle is BLACK, then there can be four configurations for x, x's parent (**p**) and x's grandparent (**g**) (This is similar to AVL Tree)
   (i) Left Left Case (p is left child of g and x is left child of p)
   (ii) Left Right Case (p is left child of g and x is the right child of p)
   (iii) Right Right Case (Mirror of case i)
   (iv) Right Left Case (Mirror of case ii)

Example: Creating a red-black tree with elements 3, 21, 32 and 15 in an empty tree.

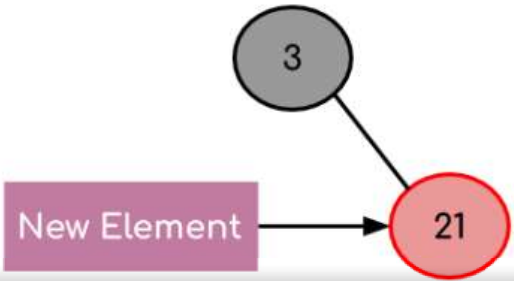
Solution:

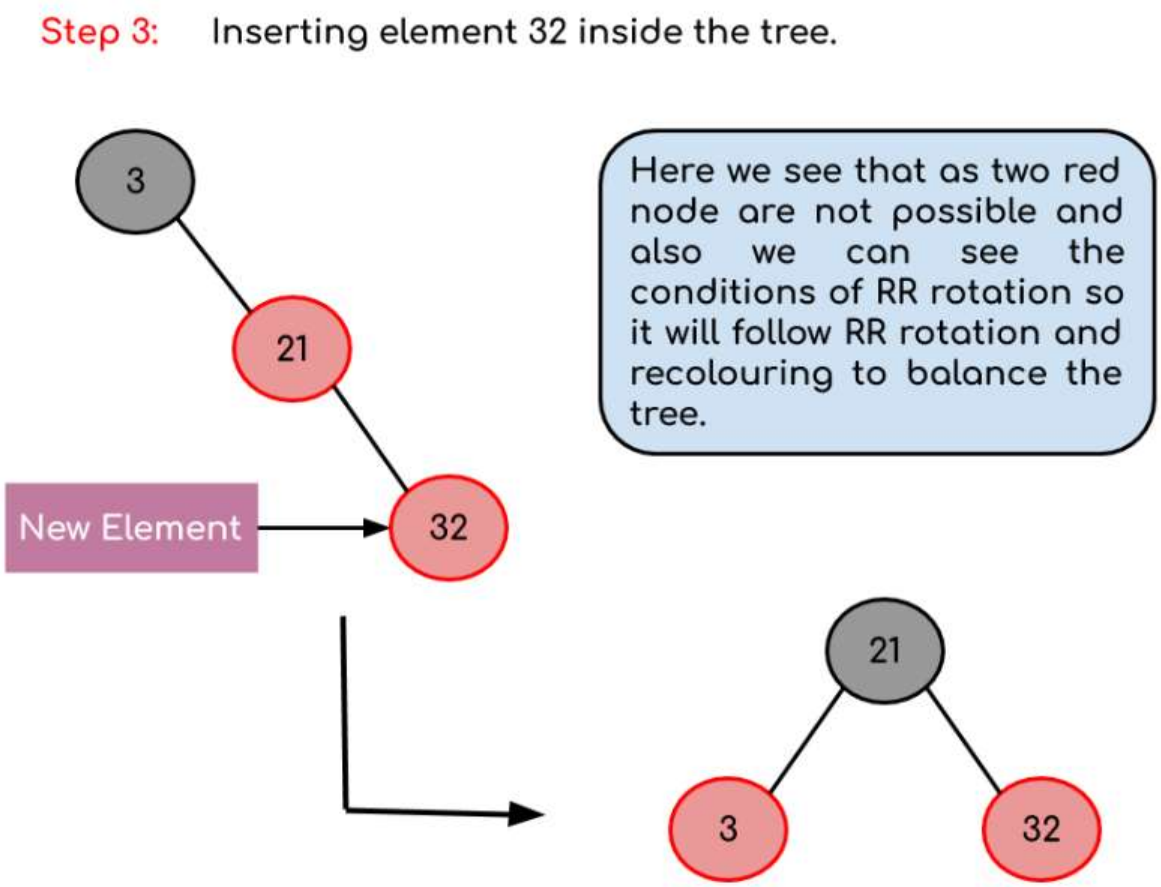
Step 1: Inserting element 3 inside the tree.



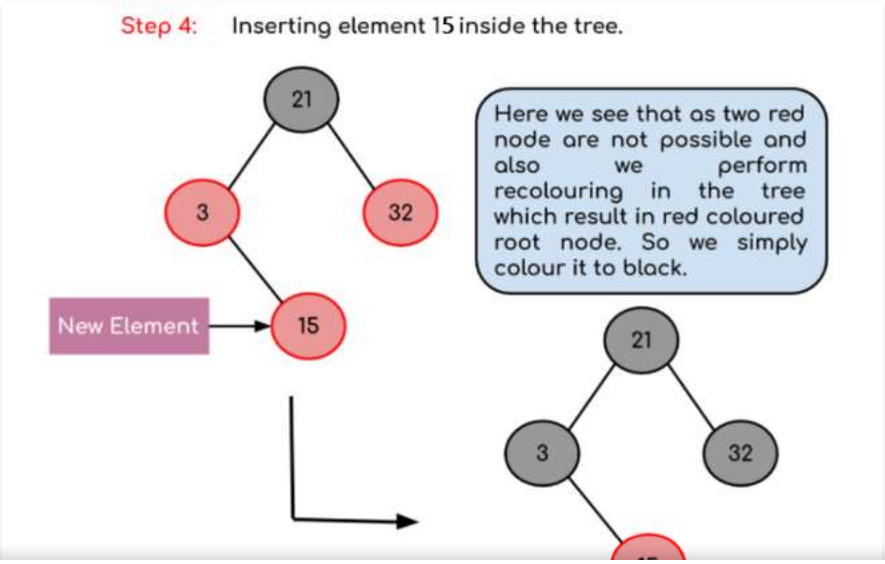
When the first element is inserted it is inserted as a root node and as root node has black colour so it acquires the colour black.

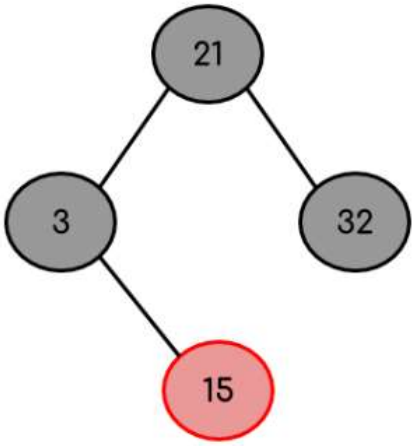
Step 2: Inserting element 21 inside the tree.





Now, as we insert 32 we see there is a red father-child pair which violates the Red-Black tree rule so we have to rotate it. Moreover, we see the conditions of RR rotation (considering the null node of the root node as black) so after rotation as the root node can't be Red so we have to perform recolouring in the tree resulting in the tree shown above.





The final tree will look like this

Please refer **C Program for Red Black Tree Insertion** for complete implementation of the above algorithm.

Red-Black Tree | Set 3 (Delete)

Code for Insertion in Java.

Here is the code written in **java** for **implementation of RED-BLACK Trees**

The following code also implements tree insertion as well as tree traversal.

at the end you can visualize the constructed tree too!!!.

```
/*package whatever //do not write package name here */

import java.io.*;

// considering that you know what are red-black trees here is the implementation in
// RedBlackTree class. This class contains subclass for node
// as well as all the functionalities of RedBlackTree such as - rotations, insertic
// inoredr traversal
public class RedBlackTree
{
    public Node root;//root node
    public RedBlackTree()
```

```
{
    int data;
    Node left;
    Node right;
    char colour;
    Node parent;

    Node(int data)
    {
        super();
        this.data = data;   // only including data. not key
        this.left = null; // left subtree
        this.right = null; // right subtree
        this.colour = 'R'; // colour . either 'R' or 'B'
        this.parent = null; // required at time of rechecking.
    }
}
// this function performs left rotation
Node rotateLeft(Node node)
{
    Node x = node.right;
    Node y = x.left;
    x.left = node;
    node.right = y;
    node.parent = x; // parent resetting is also important.
    if(y!=null)
        y.parent = node;
    return(x);
}
//this function performs right rotation
Node rotateRight(Node node)
{
    Node x = node.left;
    Node y = x.right;
    x.right = node;
    node.left = y;
    node.parent = x;
    if(y!=null)
        y.parent = node;
    return(x);
}


// these are some flags.
```

```
// helper function for insertion. Actually this function performs all tasks in
Node insertHelp(Node root, int data)
{
    // f is true when RED RED conflict is there.
    boolean f=false;

    //recursive calls to insert at proper position according to BST properties.
    if(root==null)
        return(new Node(data));
    else if(data<root.data)
    {
        root.left = insertHelp(root.left, data);
        root.left.parent = root;
        if(root!=this.root)
        {
            if(root.colour=='R' && root.left.colour=='R')
                f = true;
        }
    }
    else
    {
        root.right = insertHelp(root.right,data);
        root.right.parent = root;
        if(root!=this.root)
        {
            if(root.colour=='R' && root.right.colour=='R')
                f = true;
        }
    // at the same time of insertion, we are also assigning parent nodes
    // also we are checking for RED RED conflicts
    }

    // now lets rotate.
    if(this.ll) // for left rotate.
    {
        root = rotateLeft(root);
        root.colour = 'B';
        root.left.colour = 'R';
        this.ll = false;
    }
    else if(this.rr) // for right rotate
    {
        root = rotateRight(root);
        root.colour = 'B';
```

```
        root.right.parent = root;
        root = rotateLeft(root);
        root.colour = 'B';
        root.left.colour = 'R';

        this.rl = false;
    }
    else if(this.lr)  // for left and then right.
    {
        root.left = rotateLeft(root.left);
        root.left.parent = root;
        root = rotateRight(root);
        root.colour = 'B';
        root.right.colour = 'R';
        this.lr = false;
    }
    // when rotation and recolouring is done flags are reset.
    // Now lets take care of RED RED conflict
    if(f)
    {
        if(root.parent.right == root)  // to check which child is the current r
        {
            if(root.parent.left==null || root.parent.left.colour=='B')  // case
            {// perform certaing rotation and recolouring. This will be done wh
                if(root.left!=null && root.left.colour=='R')
                    this.rl = true;
                else if(root.right!=null && root.right.colour=='R')
                    this.ll = true;
            }
            else // case when parent's sibling is red
            {
                root.parent.left.colour = 'B';
                root.colour = 'B';
                if(root.parent!=this.root)
                    root.parent.colour = 'R';
            }
        }
        else
        {
            if(root.parent.right==null || root.parent.right.colour=='B')
            {
                if(root.left!=null && root.left.colour=='R')
                    this.rr = true;
                else if(root.right!=null && root.right.colour=='R')
```

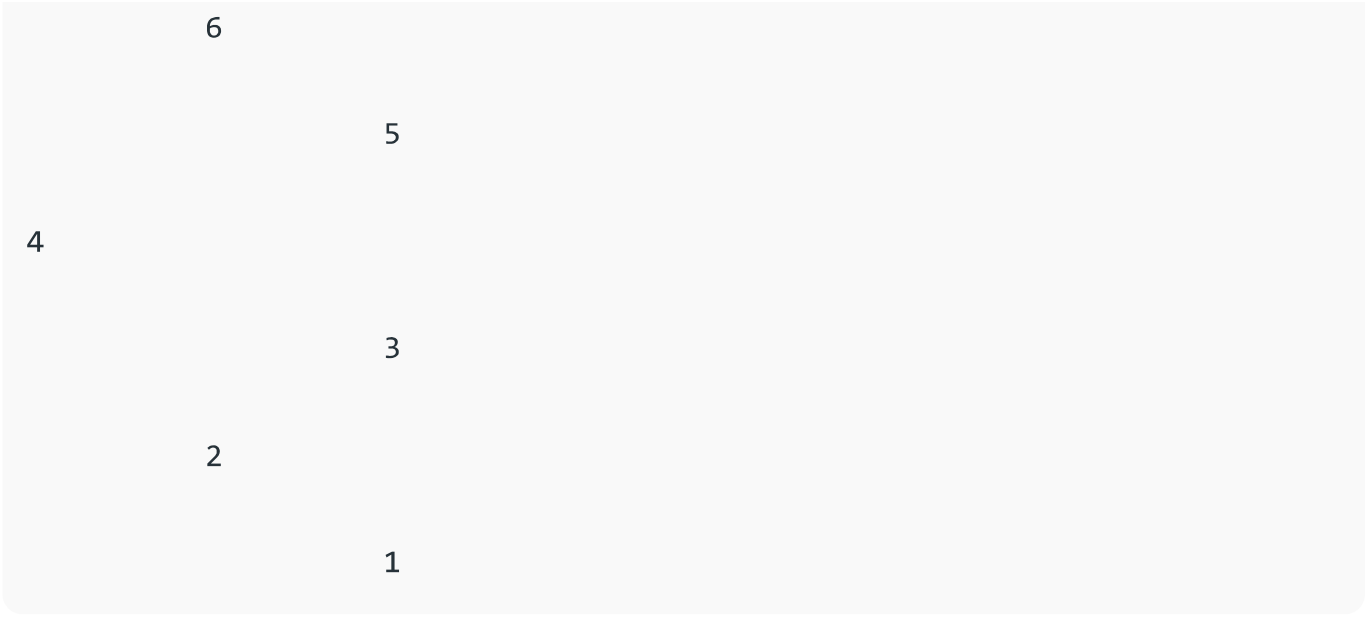
```
                    if(root.parent!=this.root)
                        root.parent.colour = 'R';
                }
            }
            f = false;
        }
        return(root);
    }

    // function to insert data into tree.
    public void insert(int data)
    {
        if(this.root==null)
        {
            this.root = new Node(data);
            this.root.colour = 'B';
        }
        else
            this.root = insertHelp(this.root,data);
    }
    // helper function to print inorder traversal
    void inorderTraversalHelper(Node node)
    {
        if(node!=null)
        {
            inorderTraversalHelper(node.left);
            System.out.printf("%d ", node.data);
            inorderTraversalHelper(node.right);
        }
    }
    //function to print inorder traversal
    public void inorderTraversal()
    {
        inorderTraversalHelper(this.root);
    }
    // helper function to print the tree.
    void printTreeHelper(Node root, int space)
    {
        int i;
        if(root != null)
        {
            space = space + 10;
            printTreeHelper(root.right, space);
            System.out.printf("\n");
```

```
                printTreeHelper(root.left, space);
            }
        }
        // function to print the tree.
        public void printTree()
        {
            printTreeHelper(this.root, 0);
        }
        public static void main(String[] args)
        {
            // let us try to insert some data into tree and try to visualize the tree a
            RedBlackTree t = new RedBlackTree();
            int[] arr = {1,4,6,3,5,7,8,2,9};
            for(int i=0;i<9;i++)
            {
                t.insert(arr[i]);
                System.out.println();
                t.inorderTraversal();
            }
            // you can check colour of any node by with its attribute node.colour
            t.printTree();
        }
}
```

Output

```
1
1 4
1 4 6
1 3 4 6
1 3 4 5 6
1 3 4 5 6 7
1 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
                    9
```



Previous Next

23, Oct 18

02 **K Dimensional Tree | Set 1 (Search and Insert)**
31, Oct 14

03 **Insert Operation in B-Tree**
23, Apr 13

04 **Binary Search Tree insert with Parent Pointer**
22, Apr 17

06 **B-Tree Insert without aggressive splitting**
03, May 19

07 **How to insert Strings into an AVL Tree**
11, Oct 21

08 **Efficiently design Insert, Delete and Median queries on a set**
01, Jul 16

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : ChiragAcharya, Preet Dabre, aditya_taparia, siddharthx_07, madhav_mohan, 20ajinky, ayushrajput1410, saurabharora9106

Article Tags : Red Black Tree, Self-Balancing-BST, Advanced Data Structure,

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks
5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org