# Insertion of Red-Black Tree

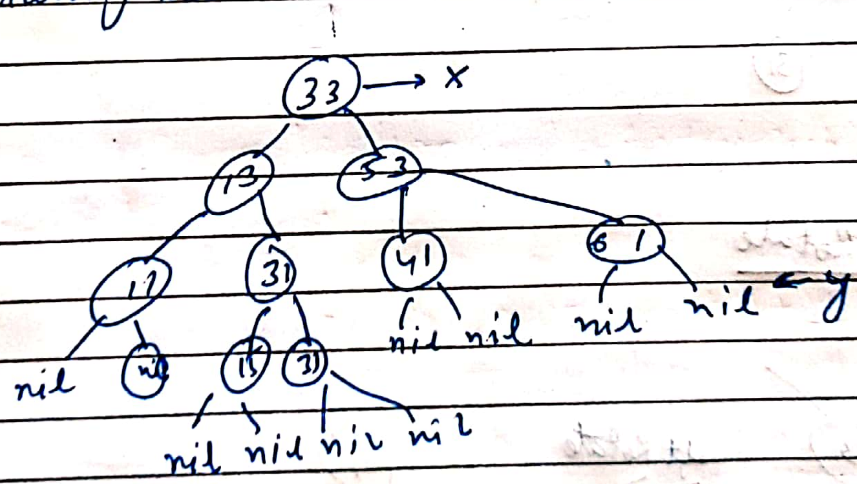
It is a self Balancing search tree in which each node contains an extra bit for denoting the color, either red or black.

new node is always inserted as Red node.

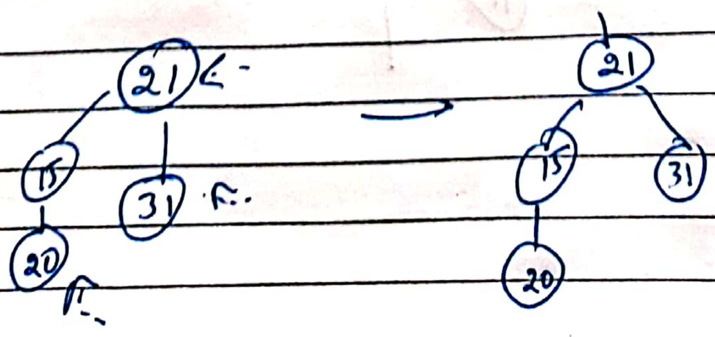
insert a new node

1)

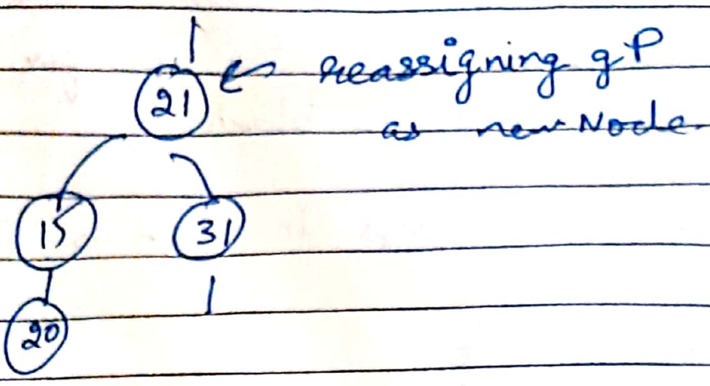
2) Let y be the leaf (i.e NIL) and x be the root of the tree. The new node is inserted in the following tree.



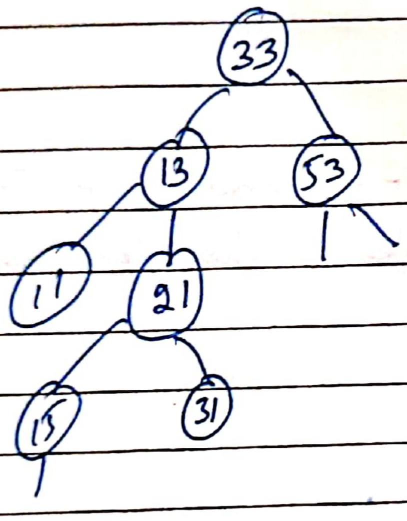
algo to maintain Red-Black property.

(a) 

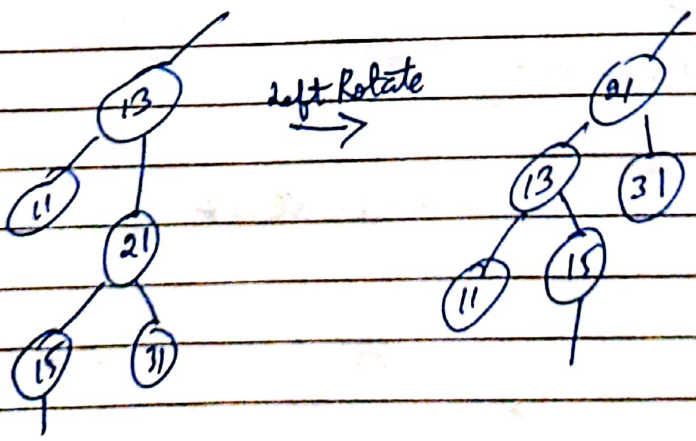
(b) Assign gP to new node



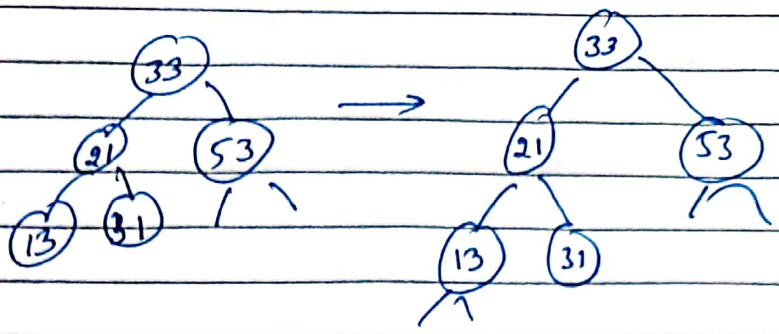
(c)



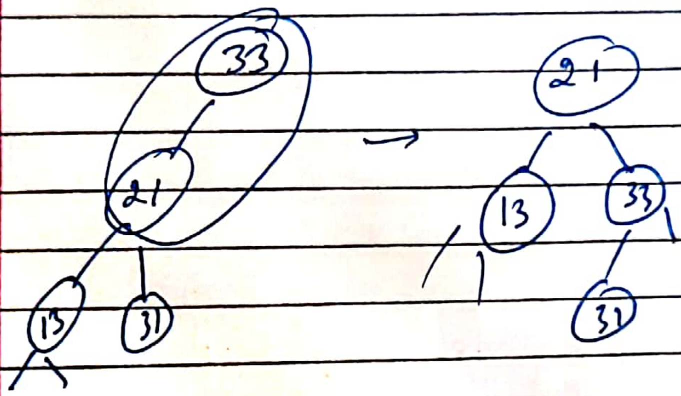
Left Rotate



Case III



Right Rotate



```
private void fixinsert (Node k)
{ Node u;
  while( k. parent .color == 1)
  { if (k.parent == k. parent. parent.right)
    {  u= k.parent.parent.left.
         if (u.color==1)
        { u.color = 0;
            k. perent. color=0;
             k.parent. perent.color=1;
           k = k.perent.perent
          }

    else if ( k == k.parent. left)
     {
        k = k.parent;
        ~~rotate Right (k);~~
       }
      k.parent.color=0;
       k.parent.parent.color = 1;
       left Rotate (k.parent.parent);
       }
```

Scanned with CamScanner

```
else { u = K.parent. right;
        if (u.color = = 1)
          u.color = 0
          K.parent.color = 0;
          K.parent. parent .color = 1;
          K = K.parent.parent;
          }
    else {
          (K = = root)
             break;
       }
    }
    root.color = 0;
    }

private void insert (int Key)

{ node n = new Node();
   node.parent = null;
   node.left = Null;
   node.right -- null;
   node.color = 1;
```

```
Node y = null;
Node x = this.root;
while (x != null)
{
    y = x;
    if(node.data < x.data)
    {   x = x.left;
    }
    else x = x.right;
node.parent = y;
if (y == null) root = node;
elseif (node.data < y.data)
        y.left = node;

else { y.right = node; }

if (node.parent == null) node.color = 0
if (node.parent.parent == null) return;
    fixInsert (node);
}
```