Red black tree

1BM18CS002

```
class RBtree
{
        Node *root;
        void rotateLeft (Node , Node );
        void rotateRight (Node , Node );
        void ~~fix err~~ balancing (Node , Node);
        void insert (int &n);
        void print ();
        RBtree()
        {   root == NULL; }

};
```

1. Same as BST insertion
Node insert ()

```
void RBtree :: rotateLeft (Node &root, Node &pt)
{
      Node p-right = p->right;
       p->right = p-right -> left
       if ( ~~pt~~ -> right != Null)
             p->right -> parent = p;
        p-right -> parent = ~~pt~~ -> parent;
        if (p->parent == NULL )
             root = p-right;
        else if (p==p->parent ->left)
             p -> parent -> left = p-right;
```

```
    p_right -> left = p;
    p -> parent = p_right;
}

void RBtree :: balancing(Node *root, Node & pt)
{
        Node  parent = NULL, grandp = NULL;
        while((pt != root && (pt -> color != Black) &&
                pt -> parent -> color == Red )
        {
            $\boxed{\text{case (I)}}$
            Parent of pt is left child of Grand parent of pt
            then
                Node uncle = grandp -> right;
                case (A)
                If uncle is red then recolour
                if (uncl != NULL && uncle -> color == Red)
                {
                    grandp -> color = Red;
                    parent -> color = Black;
                    uncle -> color = Black
                    pt = grandp;
                }
```

```
else if uncle is black then do rotation accordingly
{   case (B)
      pt is right child of its parent
       if (pt == parent -> right)
       {
                  Left - rotation
       }
    case (C)
       pt is left child of its parent
         {
                  Right - Rotation
         }
 }
[Case : 2] = Parent of pt is right child of grand parent of pt
{
       case (A)
       uncle of pt is red
       if ( uncle -> color == Red )
       {   Recolouring
       }
       else
       {       case (B)
               pt is left child of its parent
               {    Right rotation    }
               case (C)
               pt is right child
               {    Left Rotation    }
       }
}   root -> color = Black;
```