ADS LAB
Red-Black tree

Mrinmal Patil
1BM17ES062

```
class RBTree
{
Private:
Node *root;
protected:
 Void rotateleft(Node *X, Node *X);
 Void rotateRight(Node *X, Node *X);
Void fixViolation(Node *X, Node *X);

Public:
 RBTree() {root = NULL;}
 Void insert(const int &n);
 Void inorder();
 Void levelorder();
};

Void inorderHelper(Node *root)
{
 if(root == NULL)
   return;
inorderHelper(root->left);
cout << root->data << " ";
 inorderHelper(root->right);
}
Node *BSTInsert(Node *root, Node *pt)
{
 if(root == NULL)
   return pt;
 if(pt->data < root->data)
```

```
{
root->left = BSTInsert(root->left, pt);
root->left->parent = root;
}
else if (pt->data > root->data)
{
root->right = BSTInsert(root->right, pt);
root->right->parent = root;
}
return root;
}
void levelOrderHelper(Node *root)
{
  if(root == NULL)
    return;
std::queue<Node *> q;
  q.push(root);
while(!q.empty())
{
Node *temp = q.front();
cout << temp->data << " ";
  q.pop();
if(temp->left != NULL)
    q.push(temp->left);
if(temp->right != NULL)
    q.push(temp->right);
}
}
void RBTree::fixViolation(Node *&root,
  Node *&pt)
{
Node *parent_pt = NULL;
```

```
Node * grand_Parent_Pt = NULL;
while ((Pt != root) && (Pt -> color != Black) &&
  (Pt -> Parent -> colour == RED))
{
 Parent_Pt = Pt -> Parent;
 grandParent_Pt = Pt -> Parent -> Parent;
 if (Parent_Pt == grand_Parent_Pt -> left)
{
 Node * uncle_Pt = grandParent_Pt -> right;
 if (uncle_Pt != NULL && uncle_Pt -> color == Red)
{
 grand_Parent_Pt -> color = Red;
  Parent_Pt -> color = Black;
 uncle_Pt -> color = Black;
 Pt = grand Parent_Pt;
}
else
{
 if (Pt == Parent_Pt -> right)
{
 rotate left (root, parent_Pt);
    Pt = Parent Pt;
  Parent_Pt = Pt -> Parent;
}
 rotate right (root grandparent_Pt);
Swap (Parent_Pt -> color, grand_Parent_Pt -> color);
    Pt = Parent_Pt;
 }
}
```

```
else
{
Node *uncle_pt = grand_parent_pt->left;
if ((uncle_pt != NULL) && (uncle_pt->color == red))
{
grand_parent_pt->color = red;
parent_pt->color = black;
uncle_pt->color = black;
pt = grand_parent_pt;
}
else
{
 if (pt == parent_pt->left)
{
 rotateRight(root, parent_pt);
 pt = parent_pt;
 parent_pt = pt->parent;
}
 rotateLeft(root, grand_parent_pt);
 swap(parent_pt->color, grand_parent_pt->color);
 pt = parent_pt;
 }
}
}
 root->color = black;
}
void RBTree::insert(const int &data)
{ Node *pt = new Node(data);
   root = BSTInsert(root, pt);
   fixViolation(root, pt);
}
```