Red-Black tree

Revanth.NM
1BM18CS081

```
insert ( root, Node * pt) {
    If (root == NULL) return pt;
    If (pt->data < root->data) {
        root->left = Insert (root->left, pt);
        root->left->parent = root;
    }
    else if ( pt->data > root->data){
        root -> right = insert (root, pt);
        root -> right -> parent = root;
    }
    return root;
}

Balance ( Nood *& root, Node *& pt) {
    Node *parent_pt = NULL;
    Node * grand_parent_pt = NULL;

    while ((pt != root) && (pt -> color != Black) &&
            (pt -> parent -> color == RED))
    {
        parent_pt = pt -> parent;
        grand_parent_pt = pt -> parent -> parent;

        If (parent_pt == grand_parent_pt -> left)
        {
            Node *uncle_pt = grand_parent_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;
```

Revanth.NM
1BM18CS081

```
else {
    if(pt== parent_pt->right){
        rotateleft (root, parent_pt);
        pt = parent_pt;
        parent_pt = pt -> parent;
    }
    rotateRight (root, grand_parent_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;
  }
}
}
```

Revanth.NM