Chandan C. Bagan
1BM18CS026

ADS

Red-black Trees

```
enum color {RED, BLACK};
struct node{
    int data;
    bool color;
    node * left, * right, * parent;
    node(int data) {
        this -> data = data;
        left = NULL;
        right = NULL;
        parent = NULL;
        this -> color = RED;
    }
};
class redblack{
    private: node * root;
    protected: void rotateleft ( node * &, node * & );
               void rotateright (node*&, node *& );
               void fixviolation(node *&, node *& );
    Public : redblack & { root = NULL;}
             Void insertion ( const int & n);
             Void inorder ();
             void levelorder ();
};
void redblack :: rotate left( node * & root, node *& pt)
{
    node * ptr = pt -> right;
    Pt -> right = pt -> right -> left;
    if (Pt -> right != NULL)
        pt -> right -> parent = pt;
    Pt -> parent = pt -> parent;
```

```
    if (pt -> parent == NULL)
        root = ptr;
    else if (rt == pt -> parent -> left)
        pt -> parent -> left = ptr;
    ptr -> left = pt;
    pt -> parent = ptr;
}
void redblack :: rotate right (node *& root, node *& pt)
{
    node *ptl = pt -> left;
    pt -> left = ptl -> right;
    if (pt -> left != NULL)
        pt -> left -> parent = pt;
    ptl -> parent = pt -> parent;
    if (pt -> parent == NULL)
        root = ptl;
    else if (pt == pt -> parent -> left)
        pt -> parent -> right = ptl;
    else
        pt -> parent -> right = ptl;
    ptl -> right = pt;
    pt -> parent = ptl;
}
void redblack :: fixviolation (node *& root, node *& pt)
{
    node * parentpt = NULL;
    node * gparentpt = NULL;
    while ((pt != root) && (pt -> color != BLACK) &&
           (pt -> parent -> color == RED))
    {
        parentpt = pt -> parent;
        gparentpt = pt -> parent -> parent;
```

```
if (parentpt == gparentpt -> left){   // case A
    node * unclept = gparentpt -> right;
  if (unclept != NULL && unclept -> color == RED) {
    gparentpt -> color = RED;
    parentpt -> color = BLACK;
    unclept -> color = BLACK;
    pt = gparentpt;
  }
  else {
    if (pt == parentpt -> right) {   // case 2
      rotateleft (root, parentpt);
      pt = parentpt;
      parentpt = pt -> parent;
    }
    rotateright (root, gparentpt);
    swap (parentpt -> color, gparentpt -> color);
    pt = parentpt;
  }
} // case B
else { node * unclept = gparentpt -> left;
  if (unclept != NULL) && (unclept -> color == RED)) {
    gparentpt -> color = RED;
    parentpt -> color = BLACK;
    unclept -> color = BLACK;
    pt = gparentpt;
  }
  else { if (pt == parentpt -> left) {
      rotateright (root, parentpt);
      pt = parentpt;
      parentpt = pt -> parent;
    } // case 3.
    rotateleft (root, gparentpt);
    swap (parentpt -> color, gparentpt -> color);
    pt = parentpt;
} } }
```

```
    red → color = BLACK;
}
Void RedBlack :: insertion ( const int &n){
    node * pt = new node (n);
     root = insertionBST (root, pt);
     fixViolation (root, pt);
 }
Node * insertionBST (node * root, node * pt){
    if (root == NULL)
        return pt;
    if (pt → data < root → data) {
        root → left = insertionBST (root → left, pt);
        root → left → parent = root;
    }
    else if (pt → data > root → data) {
        root → right = insertionBST (root → right, pt);
        root → right → parent = root;
    }
    return root;
}
```