Batch 3

ADS LAB

Red Black Tree

Chirag Swamy
1BM19CS020

7]

```
class RBTree
{
  private:
    Node *root;
  Protected:
    void rotateLeft (Node *&, Node *&);
    void rotateRight (Node *&, Node *&);
    void fixViolation (Node *&, Node *&);

  public:
    RBTree () { root = NULL; }
    void insert (const int &n);
    void inorder ();
    void levelOrder ();
};
void inordertHelper(Node *root)
{
  if (root == NULL)
    return;
  inordertHelper (root -> left);
  cout << root -> data << " ";
  inordertHelper (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;
```

Chinaey Swamy
IBM, ACSAC

```
    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;
```

Chirag Swamy
IBM19CS402

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