18/11/2020
Wednesday

Lab - 7
Writeup

Arjun A.S
1BM18CS019

Red-black tree

// define structure "Node"

// define class "RBtree" with functions

```
RBtree ::
Void rotateleft (Node *&root, Node *&pt)
{
    Node * pt_right = pt -> right;
    pt->right = pt_right -> left;

    if (pt -> right != NULL)
        pt -> right -> parent = pt;

    pt_right -> parent = pt -> parent;

    if (pt -> parent == NULL)
        root = pt_right;

    else if (pt == pt -> parent -> left)
        pt -> parent -> left = pt_right;

    else
        pt -> parent -> right = pt_right;

    pt_right -> left = pt;
    pt -> parent = pt_right;
}

Void RBtree :: rotateright (Node *& root, Node *& pt)
{
    Node *pt_left = pt -> left;

    pt -> left = pt_left -> right;

    if (pt -> left != NULL)
        pt -> left -> parent = pt;

    pt_left -> parent = pt -> parent;
```

Arjun

Arjun A.S
IBM18CS019

```
if (pt->parent == NULL)
    root = pt_left;
else if (pt == pt->parent->left)
    pt->parent->left = pt_left;
else
    pt->parent->right = pt_left;

pt_left->right = pt;
pt->parent = pt_left;

}
}

void RBtree :: correct_violation(Node *&root, Node *&pt)
{
    Node *parent_pt = NULL;
    Node *grandparent_pt = NULL;
    while ((pt != root) && (pt->color != Black) &&
           pt->parent->color == RED))
    {
        parent_pt = pt->parent;
        grandparent_pt = pt->parent->parent;

        // parent of pt is left child of Grandparent of pt
        if (parent_pt == grandparent_pt->left)
        {
            Node *uncle_pt = grandparent_pt->right;

            // Uncle of pt is also red, recoloring.
            if (uncle_pt != NULL && uncle_pt->color ==
                Red)
            {
                grandparent_pt->color = Red;
                parent_pt->color = Black;
                uncle_pt->Color = Black;
                pt = grandparent_pt;
            }
```

Arjun A.S

Arjun A.S
IBM18CS019

```
            // pt is left child of its parent
            // right rotation
            rotateright(root, grandparent_pt);
            swap(parent_pt -> color,
                 grandparent_pt -> color);
            pt = parent_pt;
        }
    }
    else
    {
        Node *unclept = grandparent_pt -> left;
        // Uncle of pt is also red
        if ((uncle_pt != NULL) && (uncle_pt -> color == Red))
        {
            grand_parent_pt -> color = Red;
            parent_pt -> color = Black;
            uncle_pt -> color = Black;
            pt = grandparent_pt;
        }
        else
        {
            // pt is left child of its parent
            if (pt == parent_pt -> left)
            {
                rotateright(root, parent_pt);
                pt = parent_pt;
                parent_pt = pt -> parent;
            }
            // pt is right child & so left rotation
            rotateleft(root, grandparent_pt);
            swap(parent_pt -> color, grandparent_pt -> color);
            pt = parent_pt;
        }
    }
    root -> color = Black;
}
```

Arjun A.S
1BM18CS019

```
void RBTree :: insertion (const int &n)
{
    Node *pt = new Node (data);
    root = BSTinsert (root, pt);
    correct-violation (root, pt);
}

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;
}
```