Lab 08

AVL Tree

Objective:

By completing the AVL Tree Lab, you will be able to:
• Implement functions to rotate nodes and balance a Binary Search Tree.
• Insert nodes from a Binary Search Tree while maintaining tree balance.
• Remove nodes from a Binary Search Tree while maintaining tree balance.

Activity Outcomes:
This lab teaches you the following topics:

- Height of the Tree
- Rotate Left
- Rotate Right
- Insertion in AVL Tree
- Deletion in AVL Tree

1) Useful Concepts

An AVL tree is a self-balancing binary search tree. In an AVL tree, the heights of the two subtrees of any node differ by at most one. Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

Activity 1:

Height of the Tree

```
public int height(Node T) {
        int lh, rh;
        if (T == null)
            return 0;
        if (T.left == null)
            lh = 0;
```

```
        else
            lh = 1 + T.left.ht;
        if (T.right == null)
            rh = 0;
        else
            rh = 1 + T.right.ht;

        if (lh > rh)
            return lh;

        return rh;
    }
```

Activity 2:

Rotate Right

```
  public Node rotateRight(Node x) {
        Node y;
        y = x.left;

        x.left = y.right;
        y.right = x;
        x.ht = height(x);
        y.ht = height(y);
        return y;
    }
```

Activity 3:

Apply the left right rotation

```
public Node LR(Node T) {
        T.left = rotateLeft(T.left);
        T = rotateRight(T);
        return T;
    }
```

Activity 4:

Find the balance factor

```
public int BF(Node T) {
        int lh, rh;
        if (T == null)
            return 0;
        if (T.left == null)
            lh = 0;
        else
            lh = 1 + T.left.ht;
        if (T.right == null)
            rh = 0;
        else
            rh = 1 + T.right.ht;
        return lh - rh;
    }
```

Activity 5:

Insert the new node in AVL

```
public Node insert(Node T, int x) { // T is head node
        if (T == null) {
            T = new Node();
            T.data = x;
            T.left = null;
            T.right = null;
        } else if (x > T.data) { // insert in right subtree
            T.right = insert(T.right, x);
            if (BF(T) == -2)
                if (x > T.right.data)
                    T = RR(T);
                else
                    T = RL(T);
        } else if (x < T.data) {
            T.left = insert(T.left, x);
```

```
            if (BF(T) == 2)
                if (x < T.left.data)
                    T = LL(T);
                else
                    T = LR(T);
        }
        T.ht = height(T);
        return T;
    }
```

Activity 6:

Delete from AVL Tree

```
public Node delete(Node T, int x) {
        Node p;
        if (T == null) {
            return null;
        } else if (x > T.data) { // insert in right subtree
            T.right = delete(T.right, x);
            if (BF(T) == 2)
                if (BF(T.left) >= 0)
                    T = LL(T);
                else
                    T = LR(T);
        } else if (x < T.data) {
            T.left = delete(T.left, x);
            if (BF(T) == -2) // Rebalance during windup
                if (BF(T.right) <= 0)
                    T = RR(T);
                else
                    T = RL(T);
        } else {
            // data to be deleted is found
            if (T.right != null) {
                // delete its inorder successor
                p = T.right;
```

```
                while (p.left != null)
                    p = p.left;
                T.data = p.data;
                T.right = delete(T.right, p.data);
                if (BF(T) == 2) // Rebalance during windup
                    if (BF(T.left) >= 0)
                        T = LL(T);
                    else
                        T = LR(T);
            } else
                return T.left;
        }
        T.ht = height(T);
        return T;
    }//end delete
```

Task

1. Apply level order traversal of AVL Tree.