14/10/20

Chandan A.M.
1BM18CS025

④ AVL Tree Insertion and Deletion :

~~//Insertion:~~
~~struct node * insert(struct node * node~~

```
//structure
struct node{
        int data;
        struct node* left ;
        struct node * right;
}

// Insertion

        if (root = NULL){
                // create a new node
                // assign values
                return root;
        }
        else if ( value  < root -> data ) {
                // insert in left subtree
                //recur  for root -> left
                root   = rotation (root);
        }
        else if ( value  >= root -> data ) {
                // insert in right subtree
                // recur  for root -> right
                root  = rotation ( root );
        }
        return root;
}
```

```
// Deletion :

    if (p = NULL){
      return p;
    }
    if (value < p -> data) {
       // recur for p -> left
       p = rotation (p);
    }
    else if (value > p -> data) {
       // recur for p -> right
       p = rotation (p);
    }
    else {
       // i.e. node with only 1 child .
       NODE temp = root -> left ? root -> left
                       : root -> right ;
       if (temp == NULL){
          temp = p;
          p = NULL;
       } else {
          p = temp;
          p = rotation (p);
       }
       free temp;
    }
  // NODE temp = minValueNode (p -> right);
  // make temp -> data as p -> data
   p -> right = deleteElement (p -> right , temp -> data);
   p = rotation (p);
   return p;
}
```

Chandan A.M

```
// minValueNode
NODE minValueNode (NODE){
        // finds the leftmost leaf.
}

// Left Rotate :
NODE leftRotate (NODE p){
        // store right pointer of p in y
        // if (y has stored left pointer of y,)
        // update left node of y as p
        // else
        // update right of p as left of y
        // return new root i.e. y.
}

// Right Rotate :
NODE rightRotate (NODE p){
        y ← left node of p
        t2 ← right node of y
        // update right node of y as p
        // update left node of p as t2
        // return new root y.
}

// Balance Factor
int getBalance Factor( NODE temp) {
      // recursive call
      // store height of left recursive call in
                                            lheigh
      // store height of right recursive call in
                                            rheigh
```

Chandan A.M

```
        //return diff of lheight & rheight ;
    }

// Rotation
NODE rotation ( NODE p) {
        //get balanceFactor of p
        if ( bal > 1) {
            // if (bal factor of p->left > 0).
                right Rotate(p) ;
            //else {
                p -> left = leftRotate (p->left);
                p = right Rotate(p),
            }
        }
        else if ( bal < -1)
            if ( bal of p -> right >0)
                p -> right = right Rotate (p->right);
                p = leftRotate(p);
            }
            else {
                // left rotate
            }
        }
        return p;
    }
```