Harshit Hiremath
IBM18CS036

14 October

AVL Tree

~~Insertion~~

Utility functions :-

```
int max(int a, int b) {
    return (a > b)? a : b;
}

int height (Node *n) {
    if (n == NULL)
        return 0
    else return N->height;
}

Node* rightRotate (Node* y) {
    Node* x = y->left;
    Node* temp = x->right;

    x->right = y
    y->left = temp;

    y->height = max(height(y->left), height(y->right))+1;
    x->height = max(height(x->left), height(x->right))+1;

    return x;
}
```

```
int getBalance(Node* N){
    if (N==NULL) return 0;
    return height(N->left) - height(N->right);
}

Node* leftRotate(Node* x) {
    Node* y = x->right;
    Node* temp = y->left;
    y->left = x;
    x->right = temp;
    x->height = max(height(x->left), height(x->right))+1;
    y->height = max(height(y->left), height(y->right))+1;

    return y;
}
```

Insert Function :-

```
Node* insert(Node* node, int key){
    if (node ==NULL)
        return(newNode(key));
    if (key < node->key)
        node->left = insert(node->left, key)
    else if (key > node->key)
        node->right = insert(node->right, key)
    else
        return node
    int balance = getBalance(node);
```

```
node->height = 1 + max(height(node->left), height(node
                                              ->right));

if (balance > 1 && key < node->left->key)
    return rightRotate(node);
if (balance < -1 && key > node->right->key)
    return leftRotate(node);
if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
}
if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right)
    return leftRotate(node)
}
return node;
}
```

Delete Function.

```
Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != NULL)
        current = current->left;
    return current
}

Node* deleteNode(Node* root, int key) {
    if (root == NULL)
        return NULL
```

```
if (key < root->key)
    root->left = deleteNode (root->left, key)
else if (key> root->key)
    root->right = deleteNode (root->right, key)
else {
    if ((root->left == NULL) || (root->right == NULL)) {
        Node* temp = root->left ? root->left : root->right;
        if (temp == NULL) {
            temp = root
            root = NULL
        }
        else *root = *temp
        free(temp);
    }
    else {
        Node* temp = minValueNode (root->right)
        root->key = temp->key
        root->right = deleteNode (root->right, temp->key)
    }
}
if (root == NULL)
    return NULL
root->height = max(height(root->left), height(root->right)) + 1;

if (balance > 1 && getBalance(root->left) >= 0)
    return RightRotate(root)
if (balance > 1 && getBalance (root->left) < 0) {
    root->left = leftRotate (root->left)
    return rightRotate (root);
}
if (balance < -1 && getBalance (root->right) <= 0)
    return leftRotate (root);
```

```
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
```