Day 17 - 7/12/2025

**AVL Tree**

Adelson-Velsky and Evgenii Landis who invented the AVL Tree in 1962.

AVL trees are self-balancing, which means that the tree height is kept to a minimum so that a very fast runtime is guaranteed for searching, inserting and deleting nodes, with time complexity

O(log n).

The Balance Factor (BF) node (X) = difference in height of Right - Left subtrees.

Bf = 3 - 1 = 2

Balance factor values

0: The node is in balance.

more than 0: The node is "right heavy".

less than 0: The node is "left heavy".

Need clarification on this left - right or right - left

**Task 1**: Write Algo for AVL tree

1. create a node

2. check if tree is empty or not

3. if tree is empty the inserted node will be the root node.

4. if tree is not empty , do a binary search tree insertion op and also check the balance factor of the node.

5. if the balance factor exceeds 1, we should do rotations on the heavy weighted tree and repeat the insertion from step 4 onwards.

**Task 2**: Write code for AVL tree

Hint: try to insert nodes

While inserting get the balance of the tree

Create 2 methods for left rotate and right rotate

Try to insert

Finally display

Insert operation task

class Node {

   int key, height;

   Node left, right;

   Node (int d) {

       key = d;

       height = 1;

   }

}

public class Task2\_d17 {

   Node root;

   int height (Node N) {

       if (N == null)

           return 0;

       return N.height;

   }

   int max (int a, int b) {

       return (a > b) ? a : b;

   }

   Node rightRotate (Node y) {

       Node x = y.left;

       Node T2 = x.right;

       x.right = y;

       y.left = T2;

       y.height = max (height (y.left), height (y.right)) + 1;

       x.height = max (height (x.left), height (x.right)) + 1;

       return x;

   }

   Node leftRotate (Node x) {

       Node y = x.right;

       Node T2 = y.left;

       y.left = x;

       x.right = T2;

       x.height = max (height (x.left), height (x.right)) + 1;

       y.height = max (height (y.left), height (y.right)) + 1;

       return y;

   }

   int getBalance (Node N) {

       if (N == null)

           return 0;

       return height (N.left) - height (N.right);

   }

   Node insert (Node node, int key) {

       if (node == null)

           return (new Node (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;

       node.height = 1 + max (height (node.left), height (node.right));

       int balance = getBalance (node);

       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;

   }

   void printTree(Node root){

       if (root == null)

           return;

       if (root != null) {

           printTree(root.left);

           System.*out*.print(root.key + " ");

           printTree(root.left);

       }

   }

   public static void main(String args[]) {

       Task2\_d17 tree = new Task2\_d17();

       tree.root = tree.insert(tree.root, 5);

       tree.root = tree.insert(tree.root, 15);

       tree.root = tree.insert(tree.root, 44);

       tree.root = tree.insert(tree.root, 35);

       tree.root = tree.insert(tree.root, 65);

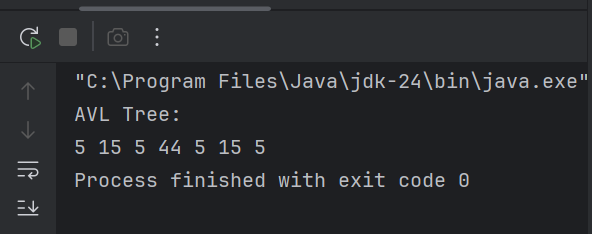
       tree.root = tree.insert(tree.root, 78);

       System.*out*.println("AVL Tree: ");

       tree.printTree(tree.root);

   }

}



**Task 3**: Write algo for Read Black tree insertion

Insert an Element - Red Black Tree −

1. Check tree is empty. If empty, then insert new node - color Black. (Because Root Node - Black in color)

2. else if Tree - not empty then insert new node as leaf node to the end and color - Red.

3. If parent of new node is Red and its neighbours(parent’s) node is also Red,

then Flip the color of the both neighbour and Parent and Grandparents (If it is not Root Node Otherwise Flip the color of the Parent and neighbour only) i.e., Black.

4. If parent of new node is Red and its neighbours(parent’s) node is empty or NULL,

then Rotate (either Left-Left or Left-Right rotation) the new node and parent.

5. we have two types of rotation

- Left Left Rotation and

- Left Right Rotation.

6. we apply Rotation in some conditions only.

The conditions are −

- If parent of new node is Red and neighbour node is empty or NULL, then rotate left or right rotation.

- In Left-Left Rotation flip the color of the parent and grandparent.

Make the parent as Grandparent and grandparent as child

**Task 4**: Wap to insert an element in red black tree

class Node1 {

   int data;

   Node1 left, right, parent;

   boolean isRed;

   Node1(int data) {

       this.data = data;

       this.isRed = true; // new node is red

   }

}

public class Task4\_d17 {

   private Node1 root;

   public void insert(int data) {

       Node1 newNode = new Node1(data);

       root = bstInsert(root, newNode);

       fixTree(newNode);

   }

   private Node1 bstInsert(Node1 root, Node1 node) {

       if (root == null) return node;

       if (node.data < root.data) {

           root.left = bstInsert(root.left, node);

           root.left.parent = root;

       } else if (node.data > root.data) {

           root.right = bstInsert(root.right, node);

           root.right.parent = root;

       }

       return root;

   }

   private void fixTree(Node1 node) {

       while (node != root && node.parent.isRed) {

           Node1 parent = node.parent;

           Node1 grandparent = parent.parent;

           if (parent == grandparent.left) {

               Node1 uncle = grandparent.right;

               if (uncle != null && uncle.isRed) {

                   // Case 3: Parent and Uncle are red

                   parent.isRed = false;

                   uncle.isRed = false;

                   grandparent.isRed = true;

                   node = grandparent;

               } else {

                   if (node == parent.right) {

                       // Left-Right case

                       node = parent;

                       rotateLeft(node);

                   }

                   // Left-Left case

                   parent.isRed = false;

                   grandparent.isRed = true;

                   rotateRight(grandparent);

               }

           } else {

               Node1 uncle = grandparent.left;

               if (uncle != null && uncle.isRed) {

                   parent.isRed = false;

                   uncle.isRed = false;

                   grandparent.isRed = true;

                   node = grandparent;

               } else {

                   if (node == parent.left) {

                       node = parent;

                       rotateRight(node);

                   }

                   parent.isRed = false;

                   grandparent.isRed = true;

                   rotateLeft(grandparent);

               }

           }

       }

       root.isRed = false;

   }

   private void rotateLeft(Node1 x) {

       Node1 y = x.right;

       x.right = y.left;

       if (y.left != null) y.left.parent = x;

       y.parent = x.parent;

       if (x.parent == null)

           root = y;

       else if (x == x.parent.left)

           x.parent.left = y;

       else

           x.parent.right = y;

       y.left = x;

       x.parent = y;

   }

   private void rotateRight(Node1 x) {

       Node1 y = x.left;

       x.left = y.right;

       if (y.right != null) y.right.parent = x;

       y.parent = x.parent;

       if (x.parent == null)

           root = y;

       else if (x == x.parent.left)

           x.parent.left = y;

       else

           x.parent.right = y;

       y.right = x;

       x.parent = y;

   }

   public void inorder() {

       inorderHelper(root);

       System.*out*.println();

   }

   private void inorderHelper(Node1 node) {

       if (node != null) {

           inorderHelper(node.left);

           System.*out*.print(node.data + (node.isRed ? "R " : "B "));

           inorderHelper(node.right);

       }

   }

   public static void main(String[] args) {

       Task4\_d17 tree = new Task4\_d17();

       tree.insert(10);

       tree.insert(20);

       tree.insert(30);

       tree.insert(15);

       tree.insert(25);

       tree.inorder();

   }

}

