25TH SEPTEMBER BST

Q1: Write an iterative program to search for an element in BST. Also, construct a sample BST and try to search for elements in the same.

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinarySearchTree {
    public TreeNode search(TreeNode root, int target) {
        while (root != null) {
            if (target == root.val) {
                return root;
            } else if (target < root.val) {
                root = root.left;
            } else {
                root = root.right;
            }
        }
        return null;
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
```

```
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(5);
        root.right = new TreeNode(15);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(7);

        int target = 7;
        TreeNode result = bst.search(root, target);

        if (result != null) {
            System.out.println("Element " + target + " found in BST.");
        } else {
            System.out.println("Element " + target + " not found in BST.");
        }
    }
}
```

Q2: Given a BST and a positive number k, find the k'th largest node in the BST.

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinarySearchTree {
    int count = 0;
```

```
    public TreeNode kthLargest(TreeNode root, int k) {
        if (root == null) return null;

        TreeNode right = kthLargest(root.right, k);
        if (right != null) return right;

        count++;
        if (count == k) return root;

        return kthLargest(root.left, k);
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(5);
        root.right = new TreeNode(15);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(7);

        int k = 3;
        TreeNode kthLargestNode = bst.kthLargest(root, k);

        if (kthLargestNode != null) {
            System.out.println("The " + k + "th largest element is: " + kthLargestNode.val);
        } else {
            System.out.println("The BST doesn't have " + k + " nodes.");
        }
    }
}
```

Q3: Given a binary search tree, find a pair with a given sum present in it.

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinarySearchTree {
    public boolean findPairWithSum(TreeNode root, int target) {
        Set<Integer> seen = new HashSet<>();
        return findPair(root, target, seen);
    }

    private boolean findPair(TreeNode root, int target, Set<Integer> seen) {
        if (root == null) return false;

        if (seen.contains(target - root.val)) {
            return true;
        }

        seen.add(root.val);

        return findPair(root.left, target, seen) || findPair(root.right, target, seen);
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(5);
```

```
        root.right = new TreeNode(15);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(7);

        int targetSum = 12;
        boolean pairExists = bst.findPairWithSum(root, targetSum);

        if (pairExists) {
            System.out.println("Pair with sum " + targetSum + " exists in the BST.");
        } else {
            System.out.println("Pair with sum " + targetSum + " does not exist in the BST.");
        }
    }
}
```

Q4: Given a BST, find the inorder predecessor of a given key in it. If the key does not lie in the BST, return the previous greater node (if any) present in the BST.

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinarySearchTree {
    public TreeNode inorderPredecessor(TreeNode root, int key) {
        TreeNode pred = null;
```

```
        while (root != null) {
            if (key <= root.val) {
                root = root.left;
            } else {
                pred = root;
                root = root.right;
            }
        }

        return pred;
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(5);
        root.right = new TreeNode(15);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(7);

        int targetKey = 6;
        TreeNode predecessor = bst.inorderPredecessor(root, targetKey);

        if (predecessor != null) {
            System.out.println("Inorder predecessor of " + targetKey + " is: " +
predecessor.val);
        } else {
            System.out.println("No predecessor found for " + targetKey + " in the BST.");
        }
    }
}
```

Q5: Given a BST and two nodes x and y in it, find the lowest common ancestor (LCA) of x and y. The solution should return null if either x or y is not the actual node in the tree.

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinarySearchTree {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode x, TreeNode y) {
        if (root == null || root == x || root == y) {
            return root;
        }

        TreeNode leftLCA = lowestCommonAncestor(root.left, x, y);
        TreeNode rightLCA = lowestCommonAncestor(root.right, x, y);

        if (leftLCA != null && rightLCA != null) {
            return root;
        } else if (leftLCA != null) {
            return leftLCA;
        } else {
            return rightLCA;
        }
    }

    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        TreeNode root = new TreeNode
```

```
(10);
        root.left = new TreeNode(5);
        root.right = new TreeNode(15);
        root.left.left = new TreeNode(3);
        root.left.right = new TreeNode(7);

        TreeNode x = root.left.left; // Node with value 3
        TreeNode y = root.left.right; // Node with value 7

        TreeNode lca = bst.lowestCommonAncestor(root, x, y);

        if (lca != null) {
            System.out.println("Lowest Common Ancestor of " + x.val + " and " + y.val + " is:
" + lca.val);
        } else {
            System.out.println("Either " + x.val + " or " + y.val + " is not in the BST.");
        }
    }
}
```