Lab 11: Binary SearchTree

Objective(s)

- Tree definition and operations.
- implement a Tree as a linked list

Tool(s)/Software

Java programming language with NetBeans IDE.

Description

- Operations
  - Insertion
  - Deletion
  - Searching

  } Binary Search Tree
- Traversal (Tree with no child, one child or two Childs)

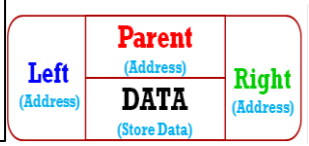
Defining the structure of tree in Java:

A. *BinaryTreeNode* class:

```
class BinaryTreeNode {
   private int value;
   private BinaryTreeNode leftChild;
   private BinaryTreeNode rightChild;
   private BinaryTreeNode parent;

   public BinaryTreeNode(int value, BinaryTreeNode leftChild,
           BinaryTreeNode rightChild, BinaryTreeNode parent) {
       this.value = value;
       this.leftChild = leftChild;
       this.rightChild = rightChild;
       this.parent = parent;
   }
   //Setters and Getters
   public int getValue() {...}
   public BinaryTreeNode getLeftChild() {...}
   public BinaryTreeNode getRightChild() {...}
   public BinaryTreeNode getParent() {...}
   public void setValue(int value) {...}
   public void setLeftChild(BinaryTreeNode leftChild) {...}
   public void setRightChild(BinaryTreeNode rightChild) {...3 lines }
   public void setParent(BinaryTreeNode parent) {...3 lines }

}
```



B. *BinarySearchTree* class:

```
public class BinarySearchTree {

    class BinaryTreeNode {...28 lines }
     private BinaryTreeNode root; //root of BST
     private int size; //number of nodes in BST
     BinarySearchTree() {
         root = null;
         size = 0; }
     BinaryTreeNode getRoot() {return root;}
     int getSize() {return size;}
     boolean isEmpty() {return (size == 0);}
     void insert(int keyVal) {...34 lines }
     public void delete(BinaryTreeNode TreeNode, int keyVal){...35 lines }
     BinaryTreeNode find(BinaryTreeNode node, int keyVal) {...14 lines }
     // Traversal Methods: InOrder, PostOrder, PreOrder
     void inorder(BinaryTreeNode node) {...7 lines }
     void preorder(BinaryTreeNode node) {...7 lines }
     void postorder(BinaryTreeNode node) {...7 lines }
     //Find Min/Max methods
     BinaryTreeNode findMin(BinaryTreeNode node){...8 lines }
     BinaryTreeNode findMax(BinaryTreeNode node){...8 lines }
}
```

Tree Operations: 1. Insert:

```
void insert(int keyVal) {
    BinaryTreeNode newNode = new BinaryTreeNode(keyVal, null, null, null);
    if (isEmpty()) {root = newNode;size++;return;}
    BinaryTreeNode current = root;
    BinaryTreeNode parent;
    while (true) {
        parent = current;
        if (keyVal == current.value) {
            System.out.println("Duplicate Found");
            return;
        } else if (keyVal < current.value) {
            current = current.leftChild;
            if (current == null) {
                parent.leftChild = newNode;
                break;
            }
        } else {
            current = current.rightChild;
            if (current == null) {
                parent.rightChild = newNode;
                break;
            }
        }
    }//end loop
    newNode.parent = parent;size++;
}
```

2. Find:

```
BinaryTreeNode find(BinaryTreeNode node, int keyVal) {
    if (node == null) {
        System.out.println("Searching fail");
        return null;
    }
    if (node.value == keyVal) {
        return node; //found
    }
    if (keyVal < node.value) {
        return find(node.leftChild, keyVal);
    } else {
        return find(node.rightChild, keyVal);
    }
}
```

3. Traversal:

In-order: (Left, Root, Right)

```
void inorder(BinaryTreeNode node) {
    if (node != null) {
        inorder(node.leftChild);
        System.out.println(node.value);
        inorder(node.rightChild);
    }
}
```

Post-Order: (Left, Right, Root)

```
void postorder(BinaryTreeNode node) {
    if (node != null) {
        postorder(node.leftChild);
        postorder(node.rightChild);
        System.out.println(node.value);
    }
}
```

Pre-Order: (Root ,Left, Right)

```
void preorder(BinaryTreeNode node) {
    if (node != null) {
        System.out.println(node.value);
        preorder(node.leftChild);
        preorder(node.rightChild);
    }
}
```

C. *In the main method:*

```
package binarysearchtreeapp;

public class BinarySearchTreeApp {
    public static void main(String[] args) {

        BinarySearchTree bst = new BinarySearchTree();
        bst.insert(50); bst.insert(30);
        bst.insert(100); bst.insert(40);
        bst.insert(70); bst.insert(60); bst.insert(80);

        System.out.println("---InOrder Traversal:------");
        bst.inorder(bst.getRoot());
        System.out.println("---PreOrder Traversal:------");
        bst.preorder(bst.getRoot());

        System.out.println("Minimum Node value: "+ bst.findMin(bst.getRoot()));

        if (bst.find(bst.getRoot(), 10) != null) {
            System.out.println("found");
        } else {
            System.out.println("Not found");
        }

    }

}
```

Tasks/Assignments(s)

1- Write the code for the following methods:

- ☒ Find the **maximum** value in the tree.
- ☒ Find the **minimum** value in the tree.

Deliverables(s)

You are required to implement and deliver a Java program(s) as described in the previous section.