21AIE203

Assignment 1

Data Structures & Algorithms

M.Kalyana Sundaram

CB.EN.U4AIE21120

1) WRITE A NOTE ON BST . GIVE AN EXAMPLE FOR INORDER TRAVEL . WRITE CODE IN JAVA/PYTHON TO DISPLAY INORDER TRAVEL(TAKE AS LEVEL 4)

```java
public class bst {
    /*
     * Create a class called Node
     * with 3 attributes: key, left, and right.
     */
    class Node{
        int key;
        Node left, right;
        public Node(int item){
            this.key = item; // key: represents the data stored in the node.
            this.left = null; //left: represents the left child of the node.
            this.right = null; // right: represents the right child of the node.
        }
    }
    Node root;//We create the constructor for the class.
    public bst(){
        this.root = null;
    }
    public void insert(int key){// We create the insert method, which takes a value as a parameter.
        this.root = insertRec(this.root, key);//assigns the parameter to this new node
    }
```

A binary search tree arranges the components in a certain order. The left node's value in a binary search tree must be less than the parent node's value, and the right node's value must be greater than the parent node. Recursively, this rule is applied to the root's left and right subtrees.

1) WRITE A NOTE ON BST . GIVE AN EXAMPLE FOR INORDER TRAVEL . WRITE CODE IN JAVA/PYTHON TO DISPLAY INORDER TRAVEL(TAKE AS LEVEL 4)

```java
public Node insertRec(Node root, int key){
        if(root == null){
            root = new Node(key);
            return root;
        }
        if(key < root.key){
            root.left = insertRec(root.left, key);
        }
        else if(key > root.key){
            root.right = insertRec(root.right, key);
        }
        return root;
    }
    public void inorder() {
        inorderRec(this.root);

    }
    public void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            /*
             * We call the function recursively and pass
             *  the left child of the root as the argument.
             */
            System.out.println(root.key);
            /*
             * We call the function recursively and pass
             * the right child of the root as the argument.
             */
            inorderRec(root.right);
        }
    }
```

```java
public static void main(String[] args) {

        // a tree with level 4
        bst tree = new bst();
        tree.insert(50);
        tree.insert(30);
        tree.insert(20);
        tree.insert(40);
        tree.insert(70);
        tree.insert(60);
        tree.insert(80);
        tree.insert(90);
        tree.insert(100);
        tree.insert(110);
        System.out.println("Inorder traversal of binary tree is ");
        tree.inorder();
    }
}
```

Output

```
[kalyan@kalyan-linux Documents]$  cd /home/kalyan/Documents ; /usr/bin/e
Code/User/workspaceStorage/ced3db3f541532c9a62a03faea973e65/redhat.java/
Inorder traversal of binary tree is
5
15
20
25
30
35
40
50
60
70
80
90
100
110
```

2) WRITE A CODE TO CHECK WHETHER THE GIVEN BINARY TREE IS BINARY SEARCH TREE OR NOT

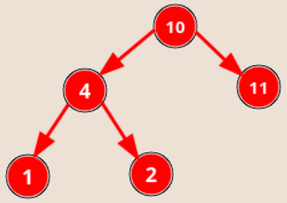
```java
class Node{
    /*
     * key: represents the data stored in the node.
     * left: represents the left child of the node.
     * right : represents the right child of the node.
     */
    int key;
    Node left, right;
    public Node(int item){
        this.key = item;
        this.left = null;
        this.right = null;
    }
}
public class checkbst {

    Node root;
    public checkbst(){
        this.root = null;
    }
    public void insert(int key){
        this.root = insertRec(this.root, key);
    }
    public Node insertRec(Node root, int key){
        if(root == null){
            root = new Node(key);
            return root;
        }
        if(key < root.key){
            root.left = insertRec(root.left, key);
        }
        else if(key > root.key){
            root.right = insertRec(root.right, key);
        }
        return root;
    }
```

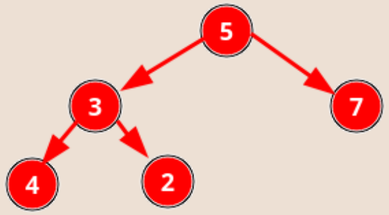
```java
//to check if the tree is a BST
    public int isBST(){
        return isbstchk(this.root, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }
    /*
     * If the root is null, then the tree is a BST and 1 is returned.
     *  If the root is less than the minimum value of the tree or greater
     * than the maximum value of the tree, then the tree is not a BST and 0 is returned.
     * We do this because the left subtree of the root should be less than the root and the
     * right subtree of the root should be greater than the root.
     */
    public int isbstchk(Node root, int min, int max){
        if(root == null){
            return 1;
        }
        /*
         *  If the root is not null and is between the minimum and maximum values of the tree,
         * then the isbstchk() method is called recursively for the left and right subtrees with
         *  the left subtree having the minimum value of the tree and the maximum value of the
         * tree as the root of the tree and the right subtree having the minimum value of the
         * root of the tree and the maximum value of the tree.
         */
        if(root.key < min || root.key > max){
            return 0;
        }
        /*
         * The result is true if both of the recursive calls returned true,
         * otherwise it will return false.
         */
        return isbstchk(root.left, min, root.key-1) & isbstchk(root.right, root.key+1, max);
    }
```

```java
    public static void main(String[] args) {
        // a tree with level 4
        checkbst tree = new checkbst();
        tree.root = new Node(10);
        tree.root.left = new Node(4);
        tree.root.right = new Node(11);
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(2);
        if (tree.isBST() != 0) {
            System.out.println("Given tree is not a BST");
        } else {
            System.out.println("Given tree is a BST");
        }

    }
}
```




```
[kalyan@kalyan-linux Documents]$
deDetailsInExceptionMessages -cp
3e65/redhat.java/jdt_ws/Documents
Given tree is a BST
```




```
[kalyan@kalyan-linux Documents]$
--enable-preview -XX:+ShowCodeDeta
ced3db3f541532c9a62a03faea973e65/r
Given tree is not a BST
[kalyan@kalyan-linux Documents]$
```

Output

3) CREATE A BST WITH LEVEL AS 5 . WRITE A CODE TO

A) DISPLAY THE VALUE OF THE NODE ON THE LEFT SIDE OF TREE..
B) DISPLAY THE VALUE OF THE NODE ON THE RIGHT SIDE OF TREE.

```java
class Node{
    int key;
    Node left, right;
    public Node(int item){
        this.key = item;
        this.left = null;
        this.right = null;
    }
}
public class Tree {

    Node root;
    public Tree(){
        this.root = null;
    }
    public void insert(int key){
        this.root = insertRec(this.root, key);
    }
    public Node insertRec(Node root, int key){
        if(root == null){
            root = new Node(key);
            return root;
        }
        if(key < root.key){
            root.left = insertRec(root.left, key);
        }
        else if(key > root.key){
            root.right = insertRec(root.right, key);
        }
        return root;
    }
```

```java
public void printLeftNodes(Node root){
        if(root == null){
            return;
        }
        if(root.left != null){
            System.out.println(root.left.key);
            printLeftNodes(root.left);
        }
        else if(root.right != null){
            System.out.println(root.right.key);
            printLeftNodes(root.right);
        }
    }
    public void printRightNodes(Node root){
        if(root == null){
            return;
        }
        if(root.right != null){
            System.out.println(root.right.key);
            printRightNodes(root.right);
        }
        else if(root.left != null){
            System.out.println(root.left.key);
            printRightNodes(root.left);
        }
    }
```

```java
public void inorder() {
        inorderRec(this.root);

    }
    public void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.println(root.key);
            inorderRec(root.right);
        }
    }
    public static void main(String[] args) {
        // a tree with level 4
        Tree tree = new Tree();
        tree.root = new Node(10);
        tree.root.left = new Node(4);
        tree.root.right = new Node(11);
        tree.root.left.left = new Node(1);
        tree.root.left.right = new Node(2);
        tree.root.right.left = new Node(12);
        tree.root.right.right = new Node(13);
        System.out.println("The whole tree is: ");
        tree.inorder();
        System.out.println("Left nodes are: ");
        tree.printLeftNodes(tree.root);
        System.out.println("Right nodes are: ");
        tree.printRightNodes(tree.root);
    }
}
```

Output

```
[kalyan@kalyan-linux Documents]$  cd /home/kalyan/Do
532c9a62a03faea973e65/redhat.java/jdt_ws/Documents_f
The whole tree is:
1
4
2
10
12
11
13
Left nodes are:
4
1
Right nodes are:
11
13
```

4) For the previous BST , write a code to display all path from root to leaf node.

```java
public void printPaths(Node root) {
        int path[] = new int[1000];
        printPathsRec(root, path, 0);

    }
    public void printPathsRec(Node root2, int[] path, int i) {
        if(root2 == null){
            return;
        }
        path[i] = root2.key;
        i++;
        /*
         * If the node is a leaf node, it will print the path[] array.
         * If not, it will recursively call the left and right child
         * node and add the node's value to the path[] array.
         */
        if(root2.left == null && root2.right == null){
            printArray(path, i);
        }
        else{
            printPathsRec(root2.left, path, i);
            printPathsRec(root2.right, path, i);
        }

    }
    public void printArray(int[] path, int i) {
        for(int j = 0; j < i; j++){
            System.out.print(path[j] + " ");
        }
        System.out.println();
    }
```

Output

```
1
4
2
10
12
11
13
Left nodes are:
4
1
Right nodes are:
11
13
path to leaf nodes are:
10 4 1
10 4 2
10 11 12
10 11 13
[kalyan@kalyan-linux Documents]$
```

Thank You !