Lab 4 BST(Binary Search Tree)

Xuanhan Tu

16 Oct 2019

Introduction

This lab is designed to help you get familiar with Binary Search Tree , a useful data structure . Your task in this lab is to implement BST individually.

Specification

In this part, you can create a data type TreeNode to be the node of a binary search tree.

following API:

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

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

And then, you can use TreeNode to create BST structure with the following API:

```java
public class BinSearchTreeDemo {
    public TreeNode root;

    //Find the right key
    public TreeNode find(int key)
    //Insert node into BST
    public boolean insert(TreeNode root,int x)
    //Remove a node from BST
    public boolean remove(TreeNode root,int x)
    //preOrder print Tree
    public void preOrder_iterator(TreeNode root)
    //inOrder print Tree
    public void inOrder_iterator(TreeNode root)
    //postOrder print Tree
    public void postOrder_iterator(TreeNode root)
    //return the kth smallest node of the tree
    public TreeNode kthSmallest(TreeNode root, int k)
    //return the depth of the tree
    public int FindTreeDeep(TreeNode root)
    //Find the closest value to target on the tree
    public int closestValue(TreeNode root, double target)
}
```

Test Case

Now, you have implemented the structure of BST.

Your task is to run test case correctly.

```
public class BSTTestDemo {

    public static void main(String[] args) {

        //Create BST
        int[] arr=new int[]{2,1,3,4,14,8,9,5,12};
        BinSerachTreeDemo bst = new BinSerachTreeDemo();
        for(int i=0;i<arr.length;i++)
        {
            bst.insert(new TreeNode(arr[i]));
        }
        TreeNode root=bst.root;
        //Test BST functions
        System.out.println((bst.find(4)==null)? "null":
"find:"+bst.find(4).value);
        bst.inOrder_iterator(root);
        bst.preOrder_iterator(root);
        System.out.println();

        //Insert
        System.out.println((bst.find(7)==null)? "null":
"find:"+bst.find(7).value);
        bst.insert(root,7);
        System.out.println((bst.find(7)==null)? "null":
"find:"+bst.find(7).value);
        bst.inOrder_iterator(root);
        bst.postOrder_iterator(root);
        bst.preOrder_iterator(root);

        //Delete
        bst.remove(root,8);
        bst.remove(root,4);
        System.out.println();
        bst.inOrder_iterator(root);

        //Other functions
        System.out.println("The depth of the tree:"+bst.FindTreeDeep(root));
        System.out.println(bst.kthSmallest(root,3));
        System.out.println(bst.closestValue(root,6.5));
    }
}
```

The output is as follow:

```
find:4
1 2 3 4 5 8 9 12 14
```

```
2 1 3 4 14 8 5 9 12

null
find:7
1 2 3 4 5 7 8 9 12 14
1 7 5 12 9 8 14 4 3 2
2 1 3 4 14 8 5 7 9 12

1 2 3 5 7 9 12 14
The depth of the tree:6
3
7
```

Bonus

Traversal can be of two types: recursive and non-recursive. Try to traverse the tree with both of the two methods.

Deadline

> 2019/10/18 18:00
> Please compress your code and upload into the FTP server with filename in format YourStudentID.zip.