Namra Joshi

Professor Mortezaie

CS 146-04

October 7, 2019

Homework 3

1- Write a program to implement a generic binary search tree. This class has a private member variable called root.

Answer:

The complete source code for the Assignment:

```
public class BinarySearchTree<E extends Comparable<? super E>>{


private static class BinaryNode<E>

{

E key;

BinaryNode<E> left, right;


public BinaryNode(E item)

{
```

```
key = item;

left = null;

right = null;

}

}


private BinaryNode<E> root;


public BinarySearchTree()

{

root = null;

}


public void insert(E value)

{

root = insert(root, value);

}


private BinaryNode<E> insert(BinaryNode root, E value)

{

if(root == null)

{
```

```
root = new BinaryNode(value);

return root;

}


if(value.compareTo((E)root.key) < 0)

root.left = insert(root.left, value);

else if(value.compareTo((E)root.key) > 0)

root.right = insert(root.right, value);


return root;

}

// return the number of nodes in the binary tree.

public int getNumberOfNodes()

{

return (countNumberOfNodes(root));

}


private int countNumberOfNodes(BinaryNode<E> node)

{

if (node == null)

return 0;

if (node.left == null && node.right == null)
```

```
return 1;

else

return countNumberOfNodes(node.left) + countNumberOfNodes(node.right);

}

//printInOrder function

public String printInOrder()

{

return (inOrderTraversal(root));

}


private static String inOrder = "";

private String inOrderTraversal(BinaryNode<E> root)

{

if(root != null)

{

inOrderTraversal(root.left);

inOrder += root.key + " ";

inOrderTraversal(root.right);

}

return inOrder;

}

//  returns a String that contains the PostOrder traversal of the binary search tree.
```

```
public String postOrder()

{

return (postOrderTraversal(root));

}


private static String postOrder = "";

private String postOrderTraversal(BinaryNode<E> root)

{

if(root != null)

{

postOrderTraversal(root.left);

postOrderTraversal(root.right);

postOrder += root.key + " ";

}

return postOrder;

}

  // returns the height of the binary tree

public int getHeightOfBST()

{

return (heightOfBST(root));

}
```

```
private int heightOfBST(BinaryNode<E> root)
{
  if(root == null)
return 0;

return (1+ Math.max(heightOfBST(root.left), heightOfBST(root.right)));
}
```

/* This method accepts a BinaryNode and determines if the difference of height of the right subtree and left subtree is greater than 1. If it is greater than 1, it should return false and otherwise it should return true.

*/

```
private boolean passedTest(BinaryNode<E> node)
{
if(node == null)
return true;

int heightOfRightSubtree = heightOfBST(node.right);
int heightOfLeftSubtree = heightOfBST(node.left);

if(heightOfRightSubtree - heightOfLeftSubtree > 1)
return false;
else
```

```
return true;

}

// returns true if all the nodes are satisfying the passed condition

public boolean allPassed()

{

boolean res = false;

while(root != null)

{

if(!passedTest(root))

{

boolean resLeft = passedTest(root.left);

boolean resRight = passedTest(root.right);

if(resLeft && resRight)

res = true;

else

{

res = false;

break;

}

}

else

{
```

```
res = false;

break;

}

}

return res;

}




public static void main(String[]args)

{

BinarySearchTree bst = new BinarySearchTree();


bst.insert(50);

bst.insert(30);

bst.insert(20);

bst.insert(40);

bst.insert(70);

bst.insert(60);

bst.insert(80);

bst.insert(100);

bst.insert(90);
```

bst.insert(10);

System.out.println("Height of the binary search tree = " + bst.getHeightOfBST());

System.out.println("Number of nodes in the binary search tree = " + bst.getNumberOfNodes());

System.out.println("In-order traversal: " + bst.printInOrder());

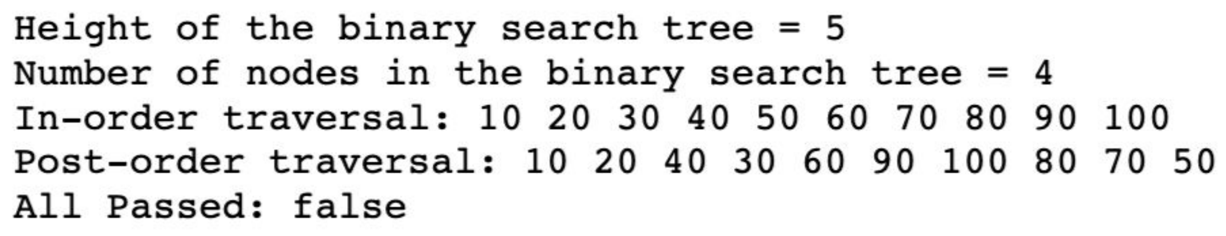
System.out.println("Post-order traversal: " + bst.postOrder());

System.out.println("All Passed: " + bst.allPassed());

}

}

Sample Test Screenshot for the all the problem statements:



```
Height of the binary search tree = 5
Number of nodes in the binary search tree = 4
In-order traversal: 10 20 30 40 50 60 70 80 90 100
Post-order traversal: 10 20 40 30 60 90 100 80 70 50
All Passed: false
```

2 Implement a public method for the above class that returns the height of the binary tree.

Answer:

Source code for the problem statement:

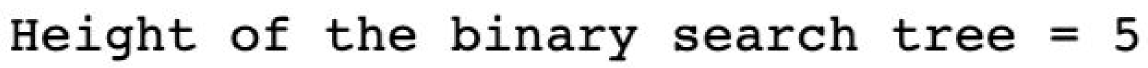
// returns the height of the binary tree

public int getHeightOfBST()

{

```
return (heightOfBST(root));
}
private int heightOfBST(BinaryNode<E> root)
{
  if(root == null)
return 0;

return (1+ Math.max(heightOfBST(root.left), heightOfBST(root.right)));
}
```

Sample Test Screenshot for the problem statement:



```
Height of the binary search tree = 5
```

3 Write a private method in BinarySearch called passedTest. This method accepts a BinaryNode and determines if the difference of height of the right subtree and left subtree is greater than 1. If it is greater than 1, it should return false and otherwise it should return true.

Answer:

Source code for the problem statement:

```
/* This method accepts a BinaryNode and determines if the difference of height of the right subtree and left subtree is greater than 1. If it is greater than 1, it should return false and otherwise it should return true.
*/
private boolean passedTest(BinaryNode<E> node)
```

```
{

if(node == null)

return true;


int heightOfRightSubtree = heightOfBST(node.right);

int heightOfLeftSubtree = heightOfBST(node.left);


if(heightOfRightSubtree - heightOfLeftSubtree > 1)

return false;

else

return true;

}
```

4 Write a private method called allPassed in BinarySearch that returns true if all the nodes are satisfying the passed condition as defined in problem-2. Note that you may traverse the tree using any technique that you like.

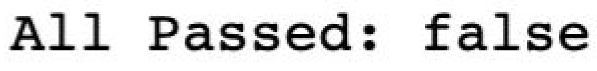
Answer:

Source code for the problem statement:

```
/ returns true if all the nodes are satisfying the passed condition

public boolean allPassed()

{

boolean res = false;

while(root != null)

{
```

```
if(!passedTest(root))
{
boolean resLeft = passedTest(root.left);
boolean resRight = passedTest(root.right);
if(resLeft && resRight)
res = true;
else
{
res = false;
break;
}
}
else
{
res = false;
break;
}
}
return res;
}
```

Sample Test Screenshot for the problem statement:



```
All Passed: false
```

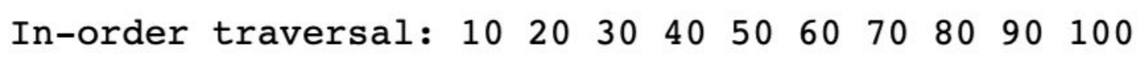
5 Implement the function printInOrder.

Answer:

Source code for the problem statement:

```
//printInOrder function

public String printInOrder()

{

return (inOrderTraversal(root));

}



private static String inOrder = "";

private String inOrderTraversal(BinaryNode<E> root)

{

if(root != null)

{

inOrderTraversal(root.left);

inOrder += root.key + " ";

inOrderTraversal(root.right);

}

return inOrder;

}
```

Sample Test Screenshot for the problem statement:



```
In-order traversal: 10 20 30 40 50 60 70 80 90 100
```

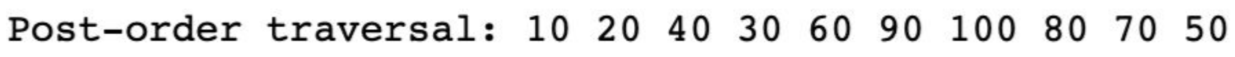
6 Implement a method that returns a String that contains the PostOrder traversal of the binary search tree.

Answer:

Source code for the problem statement:

```
//  returns a String that contains the PostOrder traversal of the binary search tree.

public String postOrder()

{

return (postOrderTraversal(root));

}


private static String postOrder = "";

private String postOrderTraversal(BinaryNode<E> root)

{

if(root != null)

{

postOrderTraversal(root.left);

postOrderTraversal(root.right);

postOrder += root.key + " ";

}

return postOrder;

}
```

Sample Test Screenshot for the problem statement:



```
Post-order traversal: 10 20 40 30 60 90 100 80 70 50
```

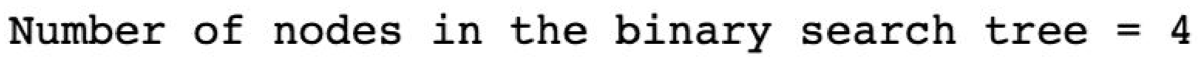
7 Implement a public method that return the number of nodes in the binary tree.

Answer:

Source code for the problem statement:

```
// return the number of nodes in the binary tree.

public int getNumberOfNodes()

{

return (countNumberOfNodes(root));

}


private int countNumberOfNodes(BinaryNode<E> node)

{

if (node == null)

return 0;

if (node.left == null && node.right == null)

return 1;

else

return countNumberOfNodes(node.left) + countNumberOfNodes(node.right);

}
```

Sample Test Screenshot for the problem statement:



```
Number of nodes in the binary search tree = 4
```