LAB_06

ICS 202

Jalal Ali Zainaddin | 202154790 | 10/11/2023

Task 1.1: BT Recursive subtreesHaveEqualNumberOfNodes():

The Recursive code:

We used helper method to check for special cases, such as tree is empty, and to compare between left and right subtrees by calling a method that returns an integer of counter of how many nodes, and check whether both are equal using equate operator '=='. In that method, each time the node is not empty, we would return one plus calling two methods again, one which contains the left, and other a right node, if the node is null, return zero and stop the method.

This will count exactly the total number of nodes the provided parent has.

Code:

```
public boolean subtreesHaveEqualNumberOfNodes(){
    // throw exception if tree is empty
    if (root == null)
        throw new UnsupportedOperationException();
    // test whether the number of nodes of the right part matches the left part of the root
    return subtreesHaveEqualNumberOfNodes(root.left) == subtreesHaveEqualNumberOfNodes(root.right);
}
4 usages
private int subtreesHaveEqualNumberOfNodes(BTNode<T> node){
    // stop the recursion when it reaches the end
    if (node == null)
        return 0;
    // add one and divide them to left and right node
    return 1 + subtreesHaveEqualNumberOfNodes(node.left) + subtreesHaveEqualNumberOfNodes(node.right);
}
```

Task 1.2: BT Recursive numOneChildNodes():

The Recursive code:

We used helper method to check for special cases, such as tree is empty. We have four cases:

1- stop the recursion when it reaches the end, and add one
2- If the node has left and right subtree, then call the method again but without adding anything
3- if the node has only left or right subtree, add one and continue from the available part
4- if the node left and right is null, return zero

This will sure that we only count the odd-sub nodes

Code:

```java
public int numOneChildNodes(){
    // throw exception if tree is empty
    if (root == null)
        throw new UnsupportedOperationException();
    return numOneChildNodes(root);
}
5 usages
private int numOneChildNodes(BTNode<T> node){
    // stop the recursion when it reaches the end, and add one
    if (node == null)
        return 1;
    // if the node have two subtrees, continue the recursion without adding anything
    if (node.right != null && root.left != null)
        return numOneChildNodes(node.left) + numOneChildNodes(node.right);
    // if the node has only left or right subtree, add one and continue from the available part
    else if (node.right != null)
        return 1 + numOneChildNodes(node.right);
    else if (node.left != null) {
        return 1+ numOneChildNodes(node.left);
    }
    // if both left and right subtree are null (not available), stop the recursion and add nothing
    return 0;
}
```

Output:

```
The number of one-child nodes in the tree is 2
The root subtrees have equal number of nodes: false
The key 5 is in tree
Preorder traversal: 1 2 4 5 12 3 8
Inorder traversal: 4 2 12 5 1 3 8
Before deleting key 3 level order traversal of binary tree is:
1 2 3 4 5 8 12 The tree is:
tR----1
   L----2
   |  L----4
   |  R----5
   |     L----12
   R----3
      R----8
After deleting key 3 level order traversal of binary tree is:
1 2 3 4 5 8 12 The tree is:
tR----1
   L----2
   |  L----4
   |  R----5
   |     L----12
   R----3
      R----8
```

Task 2.1: BST Recursive getPathToLeafNode(T e):

The Recursive code:

We used helper method to check for special cases, such as tree is empty, we have four cases:

1- when node is null, throw exception because the value does not exist

2- if current data of the node is bigger than the argument, print data and go to the left

3- if current data of the node is smaller than the argument, print data and go to the right
4- else if it is equal, print the data and stop the method

The output will be the path from tree root to the given value.

Code:

```
public String getPathToLeafNode(T e){
    // throw exception if tree is empty
    if (root == null)
        throw new UnsupportedOperationException();

    return getPathToLeafNode(e,root);
}
3 usages
private String getPathToLeafNode(T e, BTNode<T> node){
    // when node is null, throw exception because the value does not exist
    if (node == null)
        throw new NoSuchElementException();
    // if current data of the node is bigger than the argument, go to the left
    if (node.data.compareTo(e) > 0)
        return node.data + " " + getPathToLeafNode(e, node.left);
        // if current data of the node is smaller than the argument,print data and go to the right
    else if (node.data.compareTo(e) < 0)
        return node.data + " " + getPathToLeafNode(e,node.right);
    //else if it is equal, print the value and stop the method
    else
        return node.data + "";
}
```

Task 2.2: BST Recursive getNodeLevel(T e):

The Recursive code:

We used helper method to check for special cases, such as tree is empty, we have four cases:

1- when node is null, throw exception because the value does not exist

2- if current data of the node is bigger than the argument, add one and go to the left

3- if current data of the node is smaller than the argument, add one and go to the right

5- else if it is equal, add zero and stop the method

The output will be the level of the node that has given value.

Output:

```
The BST is:
tR----D
   L----B
   |  L----A
   |  R----C
   R----F
      R----H
         L----G
         R----J
Preorder traversal: D B A C F H G J
Inorder traversal: A B C D F G H J
Level order traversal: D B F A C H G J
Level order traversal by levels:
 D
 B F
 A C H
 G J
The level of node G is: 3
The path to leaf node 'G': D F H G
D F H G Element G Found
java.lang.NullPointerException: Cannot read field "data" because "root" is null
Tree after deleting G using delete by copying
tR----D
   L----B
   |  L----A
   |  R----C
   R----F
      R----H
         R----J
```