CS 314 Final Review — Binary Subtrees — **Solution**

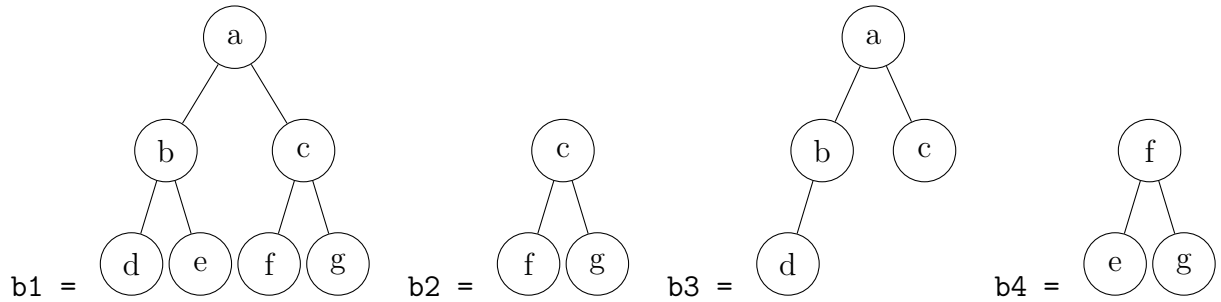
Binary Trees

Implement an instance method for a BinaryTree class which, given another binary tree, determines if the BinaryTree parameter is a subtree of this tree. That is, `this` tree must contain all of the values of the BinaryTree paramter with the same relative structure. These trees are binary trees, but they are not binary search trees.

Complete the following method.

```
// Determines whether "other" is a subtree of "this"
// pre: other != null
// post: Neither tree is altered by this operation
public boolean isSubtree(BinaryTree<E> other) {
```

Here are some sample calls to `isSubtree`:




```
b1.isSubtree(b2) → true      b1.isSubtree(b3) → true
b1.isSubtree(b4) → false     b2.isSubtree(b1) → false
```

You may use the following BinaryTree implementation

```
public class BinaryTree<E>{
  BNode<E> root;
  int size;

  //Nested node class
  private static class BNode<E>{
    BNode<E> left, right;
    E data;
  }
}
```

Do not create any new data structures or use any other Java classes or methods.

```
// Determines whether "other" is a subtree of "this"
// pre: other != null
// post: Neither tree is altered by this operation
public boolean isSubtree(BinaryTree<E> other) {
  //Trivial case
  if(this.size < other.size)
      return false;
  return findRoot(root, other.root);
}

private boolean findRoot(BNode<E> thisN, BNode<E> otherN) {
  if (otherN == null || thisN == null) {
    return false;
  }
  // Check if these two nodes are equal
  boolean equal = thisN.data.equals(otherN.data);
  if (equal) {
    if (containsAllNodes(thisN, otherN))
      return true;
  }
  //Try using a different starting node in this tree
  boolean trySkipping = findRoot(thisN.left, otherN) || findRoot(thisN.right, otherN);
  return trySkipping;
}

private boolean containsAllNodes(BNode<E> thisTreeStart, BNode<E> otherNode) {
  if (otherNode == null)  //Gone through all nodes in the other tree
    return true;
  if (thisTreeStart == null)  //Ran out of nodes in this tree
    return false;
  return thisTreeStart.data.equals(otherNode.data)
      && containsAllNodes(thisTreeStart.left, otherNode.left)
      && containsAllNodes(thisTreeStart.right, otherNode.right);
}
```

A challenging BinaryTree problem. This solution uses two different recursive helper methods. The first, `findRoot`, finds nodes in `this` tree which could serve as the start of the subtree. Then, if a valid node is found in `this` tree, then we call `containsAllNodes` which checks if, given a starting node in `this` (which may not be `this.root`), `this` tree contains all of the nodes in `other`.