CMPSCI 187 Discussion #6: Testing Trees for Equality
Individual Handout

David Mix Barrington
17 October 2012

Recursive algorithms are good for computations on recursively defined structures. In this discussion you're going to write a method to determine whether two **binary trees** are equal.

There are many ways to define binary trees – the one we use today is similar to linked lists, except that a node may have either a **left child**, a **right child**, both, or neither, instead of a single successor. The declaration of a node is very similar to the one for `LLNode`:

```
public class TNode<T> {
    private T info;
    private TNode<T> left;
    private TNode<T> right;
    public TNode (T elem) {
       info = elem;
       left = right = null;}
    // three getters, three setters with obvious names, signatures, and code
    }

public class Tree<T> {
    private TNode<T> root;
    public Tree ( ) {
       root = null;}
    // one setter, one getter
    }
```

Your assignment is to add an `equals` method to the class `Tree`. We first have to answer the question of what it means for two trees to be "equal". Here's a definition: **Two trees are equal if both have null roots, or both have root nodes that have the same contents, equal left subtrees, and equal right subtrees.**

This definition looks recursive, and suggests the following method for `equals`:

```
public boolean equals (Tree<T> other) {
   return (this.info.equals (other.info) &&
           this.left.equals (other.left) &&
           this.right.equals (other.right));}
```

Do you see the problem? If any of the components of `this` happen to be `null`, or if `this` itself happens to be `null`, we will get a `NullPointerException`. We can work around this, by either *guarding against* or *catching* the exception, but it gets complicated.

We'd like you to try another approach, with a **helper method**. Write a method `public static boolean equalSubtree (TNode<T> first, TNode<T> second)` that returns `true` if the two parameter nodes have equal subtrees under them. This means that both are null, or both are non-null and have equal left and equal right subtrees. *This* is easier to write as a recursive method, and the `equals` method for `Tree` is easy to write by calling the helper method.

On your response sheet, first write a paper version of the two methods `equalSubtree` and `equals`. Then try adding your methods to the `Tree` class we will provide, and then running our driver on our examples. Record the equal pairs on the response sheet and we will see whether your methods work.