Assignment 3: Automated Test Input Generation

Due Wednesday, October 31

Goals for This Assignment

By the time you have completed this work, you should be able to:

- Write an automated test case generator for a binary tree

This assignment may be completed in groups.

Step-by-Step Instructions

Step 1: Prepare Existing Solution, or Download and Unzip Needed Code

This assignment is based directly off of Assignment 2. You can either start from your assignment 2 solution, or download the original template code here. In this assignment, you will write a test case generator for one method of your choosing.

Step 2: Select Implementation to Write

Select **one** of the following **four** methods to implement. You will later write an automated test case generator for your chosen method. As a hint, some of these are easier than others.

1. A breadth-first search over a tree, returning the items in the tree. Different method signatures are possible for this. If you're stuck, you can use the following signature:

```
public static <A> ArrayList<A> bfs(final Node<A> root)
```

2. A pre-order traversal over a tree, returning the items in the tree. Different method signatures are again possible; you can use the following if you're stuck:

```
public static <A> ArrayList<A> preorder(final Node<A> root)
```

3. A calculation of the maximum depth of a given tree. Different method signatures are again possible; you can use the following if you're stuck:

   ```
   public static <A> int maxDepth(final Node<A> root)
   ```

4. A calculation of the number of nodes in a given tree. Leaf nodes (`null`) count towards this count. Different method signatures are again possible; you can use the following if you're stuck:

   ```
   public static <A> int nodeCount(final Node<A> root)
   ```

You may write helper methods if you wish. You can re-use your implementation code from Assignment 2, but only pick **one**. As with the in-class exercise, you should write your method implementation in `src/main/java/trees/TreeOperations.java`.

Step 3: Write Automated Test Case Generator

Write an **automated** test case generator in `src/test/java/trees/TreeOperationsTest.java`. This can use any generation and search strategy you prefer. The only stipulation is that it must use JUnit for the actual testing. See Assignment 2 for details of how to run these.

Step 4: Ensure 100% Coverage of Implemented Method

Gather code coverage information as you did for Assignment 2. While you don't need 100% coverage overall, for this assignment I'm requiring that your method you implemented was fully covered. You can determine this information by looking at the detailed report JaCoCo gives (the code coverage tool we're using), which will include a line-by-line analysis of your implementation. For a 100% covered implementation, each line should be highlighted in green, and each branch (conditions around `if`, `while`, `for`, etc.) will have a green node next to it.

If your implementation is not 100% covered, revisit your generator.

In theory, you can do the same thing with mutation analysis, but due to oddities with PIT (the mutation analysis tool we're using), it will always appear to have no coverage. There is a workaround to this problem (namely, rewriting your test case generator to work with JUnit's parameterized tests), but it's more trouble that it's worth.

Step 5: Submit Everything

Zip up everything in your `coverage_mutation_template` directory, *including* the `target` directory. **Be sure your `target` directory holds up-to-date coverage results.** Name your zipfile `automated_coverage_mutation_solution`, and submit it on Canvas. In the comments for the submission, list everyone you worked with, if applicable.