Submission Instructions

You will be uploading your submission to Gradescope. The Gradescope server may be updated (to address oversights or errors in the marking) up until Sunday July 7th . After July 7th, the server will only be updated if absolutely necessary, and you will be notified if this happens. You must adhere to the following rules (as your submission will be subjected to automatic marking system):

1. Download the compressed file "comp2402a4.zip" from cuLearn.
2. Retain the directory structure (i.e., if you find a file in subfolder "comp2402a4", you must not remove it).
3. Retain the package directives (i.e., if you see "package comp2402a4;" in a file, you must not remove it).
4. Do not rename any of the methods already present in the files provided.
5. Do not change the visibility of any of the methods provided (e.g., do not change private to public).
6. You may add tests to the main method of the A4Tree or A4TreeDemo class. Some initial tests are there for you.
7. Upload a compressed file "comp2402a4.zip" to the assignment server to submit your assignment as receive your mark immediately. By default, your last submission is your mark. However, if you click on `Submission History` you may select an earlier submission as your mark.

Please also note that your code may be marked for efficiency as well as correctness - this is accomplished by placing a hard limit on the amount of time your code wil be permitted for execution. If you select/apply your data structures correctly, your code will easily execute within the time limit, but if your choice or usage of a data structure is incorrect, your code may be judged to be too slow and it may receive a grade of zero.

You are expected to demonstrate good programming practices at all times (e.g., choosing appropriate variable names, provide comments in your code, etc.) and your code may be penalized if it is poorly written. The server won't judge this, but in case of discrepancies, this will be evaluated.

Instructions

Start by downloading the comp2402a4.zip file from Brightspace, which contains a comp2402a4 folder. This folder has multiple files. You will be modifying two of them. You may add helper functions or classes as you require.

Introduction

This assignment revolves around the `A4Tree.java`. The `A4Tree` class is a subclass of the `BinarySearchTree` class. However, you will start by modifying the `BinaryTree.java` class (which is a super super class of `A4Tree`) to facilitate tree traversals. When that is complete, you will finish a method in the `A4Tree` class.

Part 1: Binary Tree Traverals

In the `BinaryTree.java` file you will see three (3) methods that you will finish.

Observe the methods, `Node firstInNode()` and `Node nextInNode(Node w)`. These methods are complete - you do not need to modify them. The `firstInNode()` method returns the first `Node` for an in-order traversal of the `A4Tree`. Given a current `Node` as argument, `nextInNode(Node w)` returns the next `Node` in an in-order traversal of the `A4Tree`. Using these methods it is possible to do an efficient, in-order traversal of an `A4Tree`.

You will complete similar methods for pre-order and post-order traversals. The methods `nextPreNode(Node w)` and `nextPostNode(Node w)` are currently incomplete. `nextPostNode(Node w)` has a helper function, `Node`

`firstPostNode()` that returns the first `Node` in a post-order traversal in this tree, which is also incomplete. There is NO method `firstPreNode()`, since the first `Node` in a pre-order traversal is just the root.

To help you, the `A4Tree` class has methods to print the elements of the tree in pre- and post-order. Two methods (one for pre-order and one for post-order) will use the methods you wrote for the `BinaryTree` class. The remaining two methods are recursive, and will give you the correct order for pre- and post-order traversals. You can use these recursive methods to make sure your traversals are correct.

The methods you make *should not be recursive*. If you use recursion your submission will result in a `stackoverflow` error on the Gradescope server.

To help visualize and familarize yourself with the problem, there is a class, `A4TreeDemo`. You should compile and run this class. It will give you a visualization of an `A4Tree` built from random input, and print out the different traversals to the console (from the constructor). Feel free to adjust this code and adapt the tests - `A4TreeDemo` will not be tested.

Part 2: getLE(T x)

In the `A4Tree.java` file you will see one (1) method that you will finish. You are to complete the `List<T> getLE(T x)` method. This method should make a new `List<T>`, add every element from the `A4Tree` that is less than or equal to `x` to the `List`, and return the `List`. This method should run in $O(h + m)$ time, where h is the height of the `A4Tree` and m is the number of elements less than or equal to x. That means this method should do an efficient traversal that only visits the nodes that it has to in order to get the necessary information. Note that this is also a traversal, but it is a bit more complex than simple in-order, pre-order, or post-order. Though you can of course make use of `nextPreNode`, `nextPostNode`, etc, in your solution if you wish.