Module 4 Assignment: Tree Traversals

Marcar esta página

Coding Assignment fecha límite Sep 2, 2023 22:05 -05

Tree Traversals

For this assignment, you will implement 3 different ways of traversing a tree: **pre-order** traversal, **in-order** traversal, and **post-order** traversal. Since trees are naturally recursive structures, each of these traversals should be **implemented recursively**. Intuitively, pre-order, in-order, and post-order are all depth traversals that go as deep as they can before "taking a step back" and repeating. All three traversals are very similar; each follows the pattern of writing the data at the current node **(C)**, recursing left **(L)**, and recursing right **(R)**, just in different orders. Recall that **pre-order is CLR**, **in-order is LCR**, and **post-order is LRC**. You may use Java's ArrayList or LinkedList for the traversal methods.

IMPORTANT:

- **You will be given 5 attempts on this assignment, with a 30 minute cooldown between submissions.**
- **Please run your code before each submission to ensure that there are no formatting errors! If there are formatting errors in your code, your code will not be graded and a submission attempt will be logged. For more information, please review the Vocareum overview below.**

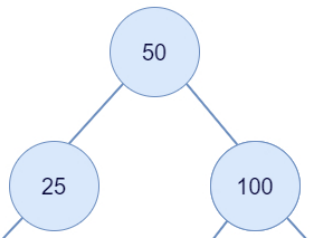
TreeNode

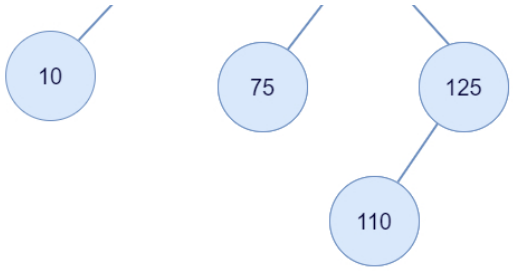
A **TreeNode** class is provided to you and will be used to represent the nodes in the passed in tree. This file should be treated as **read-only** and should not be modified in any way. This TreeNode class contains getter and setter methods to access and mutate the structure of the nodes. Please make sure that you understand how this class works, as interaction with this class is crucial for this assignment.

Helper Methods

You'll also notice that the public method stubs we've provided do not contain the parameters necessary for recursion to work efficiently, so these public methods should act as "wrapper methods" for you to use. You will have to write private recursive helper methods and call them in these wrapper methods. All of these helper methods **must be private**. To reiterate, do **not** change the method headers for the provided methods.

Traversal Example





Pre-order: 50, 25, 10, 100, 75, 125, 110
In-order: 10, 25, 50, 75, 100, 110, 125
Post-order: 10, 25, 75, 110, 125, 100, 50

General Tips

- If you remember the recursion orders for each traversal, then the implementation of each traversal will be very similar!
- Do not forget about the base case; this should be evaluated first in your helper method before continuing to recurse further down the tree.
- We highly recommend copying the starter code and working in your preferred IDE in order to have access to features such as code completion, auto-formatting, and much more!

Here are general assignment guidelines that should be followed.

- Do not include any package declarations in your classes.
- Do not change any existing class headers, constructors, instance/global variables, or method signatures. For example, do not add throws to the method headers since they are not necessary. Instead, exceptions should be thrown as follows: `throw new InsertExceptionHere("Error: some exception was thrown");`
- All helper methods you choose to write should be made private. Recall the idea of Encapsulation in Object-Oriented Programming!
- Do not use anything that would trivialize the assignment. (e.g. Don't import/use java.util.ArrayList for an ArrayList assignment.)
- Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered inefficient unless that is absolutely required (and that case is extremely rare).
- If applicable, use the generic type of the class; do not use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`.

Use of the following statements should be avoided at all times.

package	System.arraycopy()	clone()
assert()	Arrays class	Array class
Thread class	Collections class	Collection.toArray()
Reflection APIs	Inner or nested classes	Lambda Expressions

The Vocareum (code editor) interface has six main components:

- The **Drop-Down** in the top left. This lets you choose from multiple available files. Note that this drop-down will only be visible in assignments that require multiple files.
- The **Run** button. This will compile your code and run a file scan. Running your code will not count towards your total allowed submission attempts, therefore you are free to run as many times as needed.
- The **Submit** button. This will compile your code, run a file scan, grade your assignment, and output results to console. Note that for most assignments in this class, you will only be allowed a limited number of submissions. A submission is counted when the submit button is clicked, regardless of whether or not your code can compile or if there are any file issues. Therefore, we **highly recommend** that you run your code before submitting to ensure that there are no issues that will prevent your code from being graded and that every submission attempt will generate meaningful results.
- The **Reset** button. This will revert all your changes and reset your code to the default code template.
- The **Code Window**. This is where you will write your code. For large coding assignments, we highly recommend

copying the starter code and working in your preferred IDE to have access to features such as code completion, auto-formatting, and much more!

- The **Output Window**. This window will appear whenever you run or submit your code and will display the output for you to view.

For additional help, please visit the Vocareum information page located in the course information module!

< Anterior | Siguiente >