Assignment 4

Learning Outcomes

- Trees
- Recursion

Introduction

In this assignment, you will design and implement a class that manages a collection of binary tree objects and provides various functionalities to perform operations between them.

Binary trees are fundamental data structures in computer science, used to represent hierarchical relationships and support efficient searching, insertion, and deletion operations. However, real-world applications often require managing multiple binary trees and performing complex operations that involve more than one tree. This assignment should challenge you to think beyond single-tree operations and develop algorithms that handle collections of trees.

In this assignment, you will implement three classes. **TreeNode**, **BinarySearchTree**, and **BinarySearchTreeCollection**.

Note that you may add as many private helper methods as you'd like. You will likely want to use private recursive methods that are called within the public methods described in this document (similar to what we do in the first week 5 lecture video).

TreeNode

The **TreeNode** class should have the following public instance variables:

int value

TreeNode leftChild

TreeNode rightChild

Additionally, the TreeNode class should have a constructor (which takes an int) that creates an object of type TreeNode and sets left and right to null. It should also set the instance variable 'value' to the argument passed to the constructor.

BinarySearchTree

The **BinarySearchTree** class should have the following public instance variables:

TreeNode root

The **BinarySearchTree** class should have the following methods:

public void addNode(int value)

- The addNode method is used to insert a new value into the binary search tree. This method serves as an entry point for adding nodes to the tree and ensures that the insertion operation adheres to the properties of a binary search tree.
- Takes an integer value as its parameter.
- You can ignore duplicates (i.e., if a TreeNode with a value of value already exists in the binary search tree, it should be ignored.
- **Hint**: You may want to write a private method **addRecursive** (which is called within the addNode method), that takes the root of the tree and the value to be inserted. The addRecursive method would perform the actual insertion of a new TreeNode into the binary search tree, using recursion to traverse the tree and find the correct location for the new value, ensuring that binary search tree properties are maintained. The addRecursive method would return a TreeNode object.

public int getNodeCount()

- The getNodeCount method does not take any parameters.
- It returns an integer representing the total number of nodes in the tree.

public ArrayList<Integer> bfsTraversal()

- Traverses each node in the BinarySearchTree using the breadth-first search algorithm, adding the value of each node to an ArrayList in the order of visitation.

public boolean containsNode(int value)

- This method takes an integer value and returns a boolean indicating whether a node with the specified value exists in the binary search tree.

public int getHeight()

- This method returns the height of the binary search tree.

BinarySearchTreeCollection

As mentioned, the BinarySearchTreeCollection class is designed to manage a collection of BinarySearchTree objects and provide various functionalities to perform operations between them. This class allows for higher-level management and manipulation of multiple binary search trees, facilitating complex operations that involve more than one tree.

The **BinarySearchTreeCollection** class should have the following private instance variables:

ArrayList<BinarySearchTree> trees

The **BinarySearchTreeCollection** class should have the following methods:

public BinarySearchTreeCollection()

- A constructor for the BinarySearchTreeCollection class
- Initializes the ‘trees’ attribute (the ArrayList designated to hold instances of BinarySearchTree).

public void addTree(BinarySearchTree tree)

- Adds a new BinaryTree object to the collection

public BinarySearchTree getTree(int index)

- Given an index, get a specific tree from the collection
- If the index is invalid, throw an IllegalArgumentException with the message “Invalid argument!”

public void deleteTree(int index)

- Given an index, remove a specific tree from the collection
- If the index is invalid, throw an IllegalArgumentException with the message “Invalid argument!”

public int getNumberOfTrees()

- Return the number of trees in the BinarySearchTreeCollection

Assignment 4

public boolean areStructurallyEquivalent(int[] indices)

- Given an array of indices within the instance variable 'trees', check to see if each BinarySearchTree at those indices are structurally equivalent (meaning the trees have the same structure and corresponding TreeNode objects in all trees contain the same data.
- There must be at least two indices in the int array passed as an argument. Otherwise, throw and IllegalArgumentException with the message "Invalid argument!".
- If an int in indices represents a non-existent index (less than zero or greater than the length of trees-1), throw an IllegalArgumentException with the message "Invalid argument!".

public void merge(int[] indices)

- Given an array of indices within the instance variable 'trees', combine the BinarySearchTree objects into one larger tree, removing the initial trees from the trees ArrayList, and adding the newly constructed BinarySearchTree to the end of trees.
- Collect all values from the specified trees, remove any duplicate values, and sort them (you can use the sort method from the Collections class) before inserting them into the new tree. This ensures that the values are added in a consistent order, resulting in the same structure for the merged tree.
- There must be at least two indices in the int array passed as an argument. Otherwise, throw and IllegalArgumentException with the message "Invalid argument!".
- If an int in indices represents a non-existent index (less than zero or greater than the length of trees-1), throw an IllegalArgumentException with the message "Invalid argument!".
- **Note**: The resulting binary search tree will be incredibly imbalanced. That's fine, we're going to leave that as-is to avoid making this method more complicated for you.

public int getTotalNodes()

- Sum and return the number of TreeNodes in all BinarySearchTree objects within the trees ArrayList.
- Note that the values of each TreeNode does not need to be unique while counting nodes.

Provided Files

The following is a list of files provided to you for this assignment.

- **BinarySearchTreeCollectionTests.java** (I suggest you edit this class to make more tests for your own purposes, as the ones I have provided are not necessarily exhaustive and do not contain everything that may be tested through the Gradescope autograder)

Marking Notes

Functional Specifications

- Does the program behave according to specification?
- Does the program produce the correct output and pass all tests?
- Does the code run on another machine (i.e., is anything hardcoded? It shouldn't be)
- Does the code produce compilation or runtime errors?
- Does the program follow all stated rules and match specifications?

Non-Functional Specifications

- Are there comments throughout the code?
- Are variables within the methods given appropriate and meaningful names?
- Is the code clean and readable with proper indenting and white space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e., missing files, too many files, etc.) will receive a penalty of 5%.
- Including a 'package' line at the top of a file will receive a penalty of 5%.

** Remember you must do all the work on your own. Do not copy the work of another student or copy code from any AI tools. All submitted code will be run through similarity detection software.

Assignment 4

Rules

- Please only submit the files specified below.
- Do not upload the .class files! A 5% penalty will be applied for this.
- Submit your assignment on time. Late submissions will receive a penalty of 10% per day up to three days.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope.
- Assignment files are not to be emailed to the instructor or TAs. They will not be marked if sent by email.

Files to Submit

- TreeNode.java
- BinarySearchTree.java
- BinarySearchTreeCollection.java

Grading Criteria

Total Marks [13]

Passing custom tests [11.05]

- Your code will be tested on custom tests.

Following document specifications [1.95]

- Does your code comply with all 'non-functional specifications' discussed above?

Good luck, and have fun!