CSC 355. Discrete Structures and Basic Algorithms
Project 6. Trees

General Guidelines.

The method signatures indicate the required methods. You may need additional methods or classes that will not be directly tested but may be necessary to complete the assignment, and you are welcome to add those into the classes.

Unless otherwise stated in this handout, you are welcome to add to/alter any provided java files as well as create new java files as needed, but make sure that your code works with the provided test cases. Your solution must be coded in Java.

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! It is also considered academic dishonesty to share your code with another student. Anyone who is found to have violated this policy will be subject to consequences according to the syllabus and university policy.

Search Tree

In a file called `Tree.java`, implement a search tree. You have to implement a binary search tree.

Guidelines

- You need to be able to efficiently get the *height* and *size* of individual nodes in the tree, as well as the *height* and *size* of the tree itself.

Required Methods to be Implemented

Method Signature	Description
`public void put(K key, V val)`	`Insert (key, val) into the tree or update val if key is already in the tree.`
`public V get(K key)`	`Get the value associated with key in the tree or return null if the key does not exist.`

`public boolean isEmpty()`	`Return true if the tree is empty or false otherwise. Should be simple.`
`public int size()`	`Return the number of nodes in the tree. Should be simple.`
`public int height()`	`Return the height of the tree. This should take O(1) time.`
`public int height(K key)`	`Return the height of the subtree whose root node contains the given key. If the key does not exist in the tree, return -1. The time for this should be the same as the time it takes to find the node.`
`public boolean contains(K key)`	`Return true if the key is in the tree and false otherwise.`
`public int size(K key)`	`Return the size of the subtree whose root contains the given key. Return -1 if the key does not exist. The size should include the node itself. Also, the time for this should not be worse than the time it takes to find the node.`

Part 3 Grading

- You need to implement your own `BinaryTreeTest.java` file where you insert all your elements to test your implementation.
- You must submit the screenshots (besides your java files) when you tested your project.

Other notes about grading for the whole project:

- Good Coding Style includes using good indentation, using meaningful variable names, using informative comments, breaking out reusable functions instead of repeating them, etc.

- If you implement something correctly but in an inefficient way, you may not receive full credit.
- In cases where efficiency is determined by counting something like array accesses or grid accesses, it is considered academic dishonesty to *intentionally* try to avoid getting more access counts without improving the efficiency of the solution, so if you are in doubt, it is always best to ask. If you are asking, then we tend to assume that you are trying to do the project correctly.
- If you have questions about your graded project, you may contact the TAs and set up a meeting to discuss your grade with them in person.

Submission

- Submit your java files and screenshots in D2L.

Total Points

- 60 points