CS2110 Homework 7
Binary Search Trees

Due via WebCAT on April 29, 2019 at 11:30pm

Collaboration/Plagiarism Policy: This assignment is **individual work**. Sending, receiving, posting, reading, viewing, comparing, or otherwise copying any part of course assignments or solutions is not allowed except when explicitly permitted in assignment instructions. This includes solutions from other students in the course, past or present. See the course syllabus for penalties for collaboration policy violations.

Learning Objectives

- Practice your Java programming.
- Experiment with recursive data structures.
- Understand trees.

Grading Rubric

- 70% Methods function properly (passing Web-CAT tests) and your JUnit testing
- 30% Coding Style for both sections
  - 10% Correct indentation
  - 10% Naming Conventions
  - 10% Well-commented
    - Comment at the head of each file (see "Style" section below)
    - Comment at the head of every major (and new) method
    - General in-line commenting as needed to highlight interesting logic
- Up to -20% late penalty as defined in the syllabus. The latest submission or resubmission will count as the official submission time.

Complete the `TreeNode` and `BinarySearchTree` classes given, which can hold objects of any data type that implements the `Comparable` interface, such as Strings. You will be required to write several methods for each class and any additional methods that you feel might be useful (extra methods in the `TreeNode`, `toString` methods, etc.). Note: Your `BinarySearchTree` methods should call recursive methods in `TreeNode`. Specifically, you will want a head method in `BinarySearchTree` and a recursive method in `TreeNode` with the same name – there should be a method in both classes with the same signature.

1. `public int size()` - Write a method called `size()` in the `BinarySearchTree` class that returns (an int) the number of nodes in the tree. Note that the `size()` method in `TreeNode` class has been discussed in class.

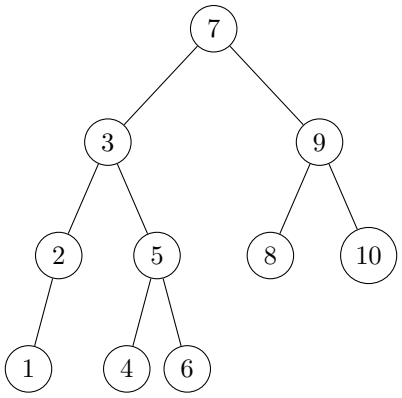

Figure 1: A sample Binary Search Tree.

2. `public int height()` - Write a method called `height()` in the `BinarySearchTree` and `TreeNode` class. The method will return an int that represents the height of the total tree. In the case of the node class, this should return the height of the subtree with this current node as the "root". The root of the tree is at height 1, the children of the root are at height 2, etc. The total height of the tree is the maximum height of all nodes in the tree. Note: this is slightly different than other sources definition, some will take the root to be at height 0.

3. `public boolean find(T val)` - Determines if the value `val` appears within the Binary Search Tree. It should return `true` if it exists, `false` otherwise.

4. `public boolean insert(T val)` - Inserts the value `val` at the appropriate place in the tree. Return `true` if insert succeeded, `false` otherwise.

5. `public boolean delete(T val)` - Removes the first instance of the value `val` in the tree, if it is found. Remember to fix-up the tree after removing the node from the tree. Return `true` if delete successfully removed a node from the tree, `false` otherwise.

6. `public String inOrder()` - Write a method in both classes called `inOrder()` that returns a `String` that represents the data held at each node starting with all the nodes of the left child followed by the root then finally all the nodes of the right child. Figure 1 shows a sample Binary Search Tree. The string for the tree in the figure should look like this:

   ```
   (1)(2)(3)(4)(5)(6)(7)(8)(9)(10)
   ```

   That is, every data entry enclosed in parentheses printed in the order described above.

7. `public String postOrder()` - Write a method in both classes called `postOrder()` that returns a `String` that represents the data held at each node starting with all the nodes of the left child, followed by all the nodes of the right child, followed by the root value. Figure 1 shows a sample Binary Search Tree. The string for the tree in the figure should look like this:

   ```
   (1)(2)(4)(6)(5)(3)(8)(10)(9)(7)
   ```

   That is, every data entry enclosed in parentheses printed in the order described above.

8. `public String toString()` - Write a `toString()` method in `BinarySearchTree` that uses `inOrder()` to list the items in the binary search tree.

In addition, add one method to the `BinarySearchTree` class that builds the tree from a list:

1. UPDATED! `public boolean buildFromList(ArrayList<T> list)` - A method that takes in a list of values and builds the binary search tree. This method may repeatedly call the `insert()` method above. If the tree already has data, this should empty the tree (remove the root) and start over. It returns `true` if the binary search tree could be built, `false` otherwise.

Testing: You will need to write at least **2 JUnit methods** for each method above except for getters and setters (no tests required for getters and setters although you can write some if you would like!). You will need to submit these to Web-Cat along with the rest of your code. Use standard naming conventions for these JUnit tests (i.e. the JUnit test names must have the word "test" in it somewhere), however, we do not mind what you call them. Remember, it would be best if you do test more than one thing per JUnit test case (method) however you can have more than one assert statement per method. There is no upper limit on how many JUnit test cases you write. Place all your JUnit test cases in one single file (`HW7Tests.java`).

Submission Information

Style: Follow the CS2110 Coding Style Guide, posted under Resources on Collab. This includes:

- Correct naming conventions, including appropriate camelCasing and TitleCasing.
- Comment each file, with a block at the top of the file denoting assignment information and comments for each field and method of your classes. You should also comment portions of your code that may be difficult to follow. We would like you to get into the habit of commenting your code. This adds to the readability of your code which contributes to "good quality" code.
- Use correct indentation. Eclipse makes this easy: select all code and choose "Correct Indentation" or Control-I (Windows/Linux) or Command-I (Mac).
- Do not put your classes into a package. (If you don't know what this means, don't worry about it.)
- If two methods share identical logic, you should factor that out into a separate method (a "helper" method).

Submission: You must submit on Web-CAT. Submit all files (`BinarySearchTree.java`, `TreeNode.java`, and `HW7Tests.java`) by "Exporting" that project from Eclipse. They must have those names exactly (including capitalization). Make sure you do not submit the `.class` files. **When you submit, Web-CAT will NOT give feedback. You should therefore perform main method testing to ensure your results match our examples.** If you fail our tests your grade will be based on how many tests you pass. If you fail a test, Web-CAT will give you a small hint about what was being tested. If your code does not compile, you will receive a 0 and Web-CAT will try to point out why it was unable to compile your code. (Usually, this is because something was spelled differently than specified above. Be sure to follow all naming conventions for getters and setters.)

To submit on Collab you need to zip up your files. If you are not sure how to do this you can follow the instructions below:

1. Right-click your `src` folder in Eclipse
2. Select `Export...`, `General`, `Archive File`
3. Check the `.java` files you want to submit
4. Browse to a save location you can find again
5. Finish
6. In a browser, go to `web-cat.cs.vt.edu`
7. Log in
8. Click the Submit button next to an assignment
9. Browse and select the `.zip` file you just created
10. Upload
11. Confirm