Project D: Trees and Big Data

Submit Assignment

Due Monday by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip
Available until Dec 19 at 11:59pm

First modify a binary search tree so that it allows duplicates. Then, use this new binary search tree to organize records from a big data set.

This project has two parts. In Part A, you will write the binary search tree class. In Part B, you'll use the class to analyze a data set.

Part A Description

The class BinarySearchTreeWithDups represents a binary search tree in which duplicate entries **are** allowed.

Process for Adding Duplicates

A duplicate entry is placed in the entry's **left** subtree. The process for adding to this new binary tree is:

1. Compare the new element to the current element.
    a. If the new element is smaller, go into the left subtree. Return to step 1.
    b. If the new element is larger, go into the right subtree. Return to step 1.
    - Note: This is the same process used in a regular binary search tree.
2. If the new element is equal to the current element:
    a. Go into the **left** subtree. Return to step 1.

Review the provided tree picture that uses data from the driver to make sure you understand how the duplicates are added.

The BinarySearchTreeWithDups Class

We will assume the getEntry method returns the first match it finds and the remove method removes the first match it finds. So the only modification required is the add method.

The class BinarySearchTreeWithDups extends BinarySearchTree,

- The class shell is provided with empty methods that you will modify.
- There are many classes required to get this lab to compile.
    - All are included in the provided zip.
    - Use the versions provided with this project!
    - **Only modify the BinarySearchTreeWithDups class.**
        - Do not modify other classes.
- Begin by closely reviewing BinarySearchTree and BinaryTree classes.

- Make sure you understand how these classes work before you implement BinarySeachTreeWithDups.
- You must have a good understanding of how the regular BST class works before you can make modifications.
- I cannot stress this point enough!

Part A Requirements

As stated in the **Syllabus**, projects that do not compile will receive 0 points. Note that compiler *warnings* are okay (and expected in a class such as this that uses generics). But compiler *errors* that mean you cannot compile your code will result in a score of 0.

You will implement four methods.

Important Note: For full credit, take advantage of the sorted nature of a binary search tree to write efficient code. Consider whether you always need to search both branches of a tree!

1. **(20 points)** Write a **addEntryHelperNonRecursive**(T) method.
    - Override the add method to call a new private addEntryHelperNonRecursive method.
    - The helper method allows duplicate entries to be added, using the algorithm described above.
    - **Important**: This method must be written **without** recursion in order for Part B to run.
    - Hint: review the addEntry method in BinarySearchTree class.
2. **(20 points)** Write a **countEntriesNonRecursive**(T) method.
    - The method counts the number of times an element appears in the tree.
    - **Important**: This method must be written **without** recursion in order for Part B to run.
3. **(20 points)** Write a **countGreaterRecursive**(T) method.
    - The method counts the number of elements in the tree greater than the parameter.
    - The elements in the tree implement Comparable, so you can invoke the compareTo method on the data inside of any node.
    - This method uses recursion. You can add a private helper method if necessary. It's okay if the countGreaterRecursive method isn't actually recursive, but the helper method is recursive.
4. **(20 points)** Write a **countGreaterWithStack**(T) method.
    - The method does the same as the method above, but is not recursive.
    - The method uses a stack in a meaningful way.

Test Your Program

Before you move onto Part B, make sure your class is working correctly.

- Use the Part A driver program to test your methods.
    - The provided files include a picture of the tree created by the driver.
    - You might consider adding more tests to the driver.
- The driver also includes a method you can use to help you evaluate whether you are fully taking advantage of the sorted nature of the BST.
    - I recommend commenting out this method until you have the program fully working with the other tests in the driver.
    - Once you are ready, follow the directions on the tester. You need to add some local variable counts and printlns in the methods in order to use this part of the driver.

Part B Description

Use the working BinarySearchTreeWithDups class to process a big data file.

The Data File

The data file is a list of San Francisco police incident reports for Larceny/Theft from 2003 to 2015 (downloaded from **here (https://data.sfgov.org/Public-Safety/SFPD-Incidents-from-1-January-2003/tmnf-yvry)**). There are over 370,000 records in the file.

In eclipse, place the data file in the same folder as the src/bin folders (one level above the .java files).

Police Report Objects

The PoliceReport class represents a single report. PoliceReport objects are compared for ordering and equivalence by their date. Two reports with the same date are considered logically equivalent- "the same." You can see this in the equals and compareTo methods of the PoliceReport class.

Note: I use a somewhat clunky/hacked method of identifying search criteria (creating a "dummyRecord" with only a single criteria on which to match). The proper way to do this would be to use Comparator objects or, better yet, the Java 8 methods of filtering and matching streams. However, these programming concepts are beyond what you are expected to know for our course, so I used the more simpler (but very inelegant!) approach. Please do not take this as an endorsement of this approach!

The Driver Class

ProjectDPartBDriver reads in the data file and creates a list. It then builds trees from three different orderings of the list: sorted ascending, shuffled, and sorted descending.

The driver then "processes" the lists and trees by counting the number of reports that were submitted on a specified set of days. The code uses your countEntriesNonRecursive method to do this. The program displays the count results and how long the processing takes using the list and the tree.

You will run the driver program to answer the questions below.

Note: I put a limit on the number of records that are read in. On my machine, the driver completed in about 4 minutes using the limit of 100,000 records. You can adjust this limit to be smaller if needed based on your own system. You might need to lower the limit to get data to answer the questions below. (The current code uses 40,000 records.) You can set whether to limit the records and what limit to use in lines 88 and 89 of the PartB driver.

If you want to run the whole data file to play around with it, you can set the "using limit" variable to be false. If you do this, however, I recommend commenting out the Comparison C section, or else you'll be waiting a long time! :)

Part B Requirements

Critical: Make sure Part A is working properly before moving on to Part B!

(20 points) For Part B, write short answers to each of the following questions.

1. Which version of the tree had the fastest processing:
    - the tree from Comparison A (built from sorted list, ascending)
    - the tree from Comparison B (built from shuffled list)
    - or the tree from Comparison C (build from sorted list, descending)?
    - Why do you think that is?
2. How can you explain the difference in building and processing time between the Comparison A tree (built from sorted list, ascending) and the Comparison C tree (built from sorted list, descending)?
    - The tree built from the sorted list ascending took less time to build and process. Why?
    - Hint: Try drawing a small tree to see what is going on. Perhaps a tree built from 1, 1, 2, 2, 3, 4, 4, 5 and from 5, 4, 4, 3, 2, 2, 1.
3. For Comparison B, which processing was faster- the tree (built from the shuffled list) or the actual shuffled list? How would you describe the big-o of what was going on in the processing with these two structures?
4. What is the main characteristic of a binary search tree affects its efficiency?

Provided Files

ProjectDTreesFiles.zip

Extra Credit

(15 points) Write an O(n) countUniqueValues() method for the BinarySearchTreeWithDups class that returns a count of all unique values in the tree.

For example, a tree with the contents (1, 1, 1, 2, 3, 4, 4, 5, 5, 6, 7, 7, 7) would return 7 because there are 7 unique values (1, 2, 3, 4, 5, 6, 7). Your method **must** be linear to receive full credit!

Submission

To submit:

1. Remove any package statements in your classes.
    - To do this, remove "package mypackage;" from the top of all files.
2. Zip together these files. Be sure to zip the .java file, not the .class file. Upload the provided files, even though they have not been modified.
    - BinarySearchTreeWithDups.java (modified)
    - **All unmodified provided files**- including the interfaces, unchanged classes, and driver programs
    - Written answers to Part B in the zip- include these as a .txt, .doc, .rtf, or .pdf file.
3. Upload the .zip file.

You are encouraged to work in groups of up to four students. You can earn up to 20 extra credit project points for working in a group (10 points per project, max of 20).

If submitting as a group:

- Submit one assignment only through one group member's account.
- Put the names of all group members in the comment submission box on Canvas.
- Everyone in the group will receive the same score and feedback.

Reviewing Feedback

You will receive feedback with your score. We always leave comments when points are deducted. Review this **guide for how to review assignment comments (https://community.canvaslms.com/t5/Student-Guide/How-do-I-view-assignment-comments-from-my-instructor/ta-p/283)** .