INFS-519 - Assignment 3: Binary Search Tree – Build and Operate

Assignment Given: 10/12/2017
Assignment Due: 11/02/2017, 7:30 pm

This assignment has two parts. In the first you will read a transaction stream and build and operate a Binary Search Tree (BST) based on the stream. This will involve building and operating several trees in succession. In the second part you will be required to balance the BST. Part 1 is described on 10/12/2017 so you can get a start on that part. The 10/26/2017 class meeting will cover how to balance a BST and information needed for completing Part 2 will be given at the end of that class meeting.

Part 1. You are to create a Java program that builds and operates a Binary Search Tree (BST). To do that the program reads a stream of transactions that can perform several types of operations. The transactions include a transaction code and a numeric data node value to be stored in the BST. The transaction types and meanings:

Transaction Code	Transaction Argument	Definition/Action to be taken
I	Integer value	Insert a node into the BST (or establish the BST if it's empty)
F	Integer search value	Find a node with a given value in the tree, return some positive indicator, otherwise return "-1"
R	Integer value	Remove the value from the BST if found – return some positive indicator if found and removed, otherwise "-1" if not found
D	<none>	Display (print) the entire tree in a given format and the number of nodes and height at the time of display
E	<none>	Reset the tree to being Empty

In creating and coding the program:

1. The input is a single transaction text file, with the following description:
   - contains a series of transactions, one per line
   - No limit on the number of transaction lines, but plan on no more than 1,000
   - each transaction has the transaction code in the first position, one or more spaces, and an integer value between 0 and 999 (max 3 digits) – ignore all other input on the line
   - the transaction stream drives what the program does – insert, find, remove, etc., and an "E" means reset the BST to being empty and start over with new transactions (if any)
   - continue processing the transaction stream until there are no more
   - the submitted code must read the input file as an argument to the program since that's the way the GTA will evaluate your submission
2. Examples of a transaction stream (your program should ignore comments):

```
I 100      // establish a tree with root node value = 100
I 162      // insert node with value = 162
I 31       //insert 31
I 48       // insert 48
I 92       // insert 92
I 119      // insert 119
F 162      // Find node = 162, return positive indicator
F 131      // Find node = 119, return -1
R 132      // Remove node = 132, return -1
R 48       // Remove node = 48, return positive indicator
D          // Display/print the tree
E          // Reset the tree to being empty
```

(…more transactions could follow, possibly building, displaying, and resetting multiple BSTs.)

3. You must create and use a BST implementation using a linked-type (not array) strategy.

4. Implementations were presented in class and in the book. Provide citations if using that code.
5. I suggest testing and debugging the code using small transaction streams, then build up to larger, more complex ones
6. Output should include an echo of the transaction ("insert node = 100" for the first line above, for example), the result of the transaction ("162 inserted"), trees displayed via the "D" transaction, and an "end of processing line" after the last transaction is read and processed. Before the end of line processing print/display the total number of trees built and total number displayed.
7. While it is always a good idea to validate your input, you may assume the provided transaction file will be error free (and hopefully in a code format readable by all).
8. After the 10/26 class, the additional balancing requirement will be described in enough detail to complete the assignment. Preparing Part 1 will allow you to get the basic structure up and going so that adding the balancing requirement can be done in the allotted time.

Displaying the Trees

All node values of the current BST tree, including the root, must be displayed when a "D" transaction is encountered. You are free to decide how to do that, but the output must clearly display the structure of the tree. Provide a citation for any third party software you use. If you have no other options, display the tree using an indentation strategy along the following lines:

```
100                     // root
    64                  // level 1, left subtree
        32              // level 2,  left-left subtree
        43              // level 2,  left-right subtree
     148                // level 1,  right subtree
         131            // level 2, right-left subtree
         174            // level 2, right-right subtree
              null      // level 2, right-right-left subtree (empty – use "null" or other obvious choice)
              187       // level 3, right-right-right subtree
```

For nodes with fewer than two children, only display "null" if there is a right or left child. If no children, nothing additional needs to be displayed.

Measuring the Timing and Analysis Report

This assignment has no timing requirement and no separate written report.

What and where to submit:

1. All items are to be submitted through Blackboard in a single zipped directory named "PA3-<username>BST.zip". Example: PA3-jdoeBST.zip. Name the code file "ExBST" (without the quotes). Do not make your code part of a package.
2. The Java code used to build and operate the BST, with a comment block that includes your name, date, "INFS-519 Assignment 3", appropriate code attributions (if any), and a short description of the code's input, processing, and outputs. The code will be assessed by visually inspecting the source and executed using the Java command line for a test set of 1,000 transaction lines, two values per line (code and node value). All code needed to execute from the command line must be included.
3. A pdf file that includes a log or screen shot of the output.

How the assignment will be assessed

The Java code will be visually inspected and executed from command line using a set of test transactions from a .txt file. An example file for you to use in debugging will be provided on Blackboard. That file does not necessarily

contain all possible combinations of expressions or errors. Your program must read the input file as an argument and open it for processing. The GTA will not edit your code to hard code a file name. Programs correctly echoing the input, displaying correctly built trees, and correct output counts will be given full credit.

Item	Assessment Description	Max Value
Java Code	All code submitted	45
	Code is readable, commented, attributed when necessary	5
Execution Output	Part 1. Code correctly executes against the transaction file error-free and produces the transaction echo, displayed trees, and tree and node counts (GTA assigns partial credit for less than 100% correct output, using his judgment)	30
	Part 2. All the above plus the code needed to balance the BSTs is included and executes correctly (GTA assigns partial credit for less than 100% correct output, using his judgment)	20
Total		**100**

Extra Credit: As with Assignments #1 and #2, for the class being held the night the assignment is due (11/02), I would like a few students to volunteer to present their code and results to the class. This would be at most a 10 minute presentation that includes (1) a code walk-through, (2) reviewing the results, (3) discussing results, and (4) Q&A from the class. Contact me separately if interested. Priority will be given to those who haven't presented before. Since this assignment is less complicated in terms of analytical results, I will limit the number of presentations to five. Extra credit would be +5% added to your cumulative assignment score (i.e. if the collective grade for all your assignments is 85% for the semester, this increases it to 90%). Presenting your work in front of peers is good experience (professionally and academically).