Assignment 3, TCSS 342B Spring 2017

OBJECTIVE

The objective of this assignment is to give you practice with the trees and their performance, code reading and writing, generics, and conducting experiments.

ASSIGNMENT SUBMISSION

To get credit for this assignment, you must

- ✓ submit your assignment on time
- ✓ name all your files/classes/methods exactly as instructed (the overall project should be named *yourLastName_pr3*)
- ✓ commit your final running Java project into the course SVN folder
- ✓ submit your *final report* on Canvas
- ✓ your project needs to be compatible with the 342 Java / Eclipse version

DESCRIPTION OF ASSIGNMENT

Word counting is used in all sorts of applications, including text analysis, creating indexes, and even cryptography. In this programming assignment, you will take a text file and compute what words appear in that file and how many times they appear using a dictionary implemented with different tree structures. You will first modify provided code and then run the code on input files of different sizes and different contents to see which data structures seem to perform better in practice.

Overall, the dictionary is to be implemented in four different ways: as a binary search tree, as an AVL tree, as a splay tree, and as a Java TreeMap (red-black tree). Starter code is provided and it already contains code that counts words and prints them using Java built-in red-black tree. Specifications for how to modify starter code follow.

Once you finish modifying the code for this assignment, you will need to perform experiments to compare and graph actual run times of these different implementations using two different kinds of input files: (1) sorted input and (2) standard English text. For each of the categories, you should run the algorithms on three file sizes: small, medium, and large, and you should run it 3 times for each one to get an average. This means you need to run the program 6x per tree type (x 3). The input files are provided in the project in *myfiles* folder.

After you run all your experiments, write a report describing what types of files you performed your experiments on and what conclusions you made. Your report should include all data and 4 graphs all together, with each graph corresponding to the type of a file (sorted vs standard English), with the x-axis showing the number of words being used as input and the y-axis showing the running time:

- 1 graph should depict insertion times only for sorted inputs for all 4 algorithms
- 1 graph should show printing times only for sorted inputs for all 4 algorithms
- 1 graph should show insertion times only for standard English text inputs for all 4 algorithms
- 1 graph should show printing times only for standard English text inputs for all 4 algorithms

Make sure to use linear increments on each axis as to not to skew the graph. Label and color your graphs properly so that they are readable. The sample Excel sheet that can help you with generating the graphs for the report and that shows how to format the data tables and the graphs is provided with the assignment. *When you write your conclusions, tie them back to what you know about the Big O run times of the algorithms selected for this assignment*.

Code Specifications

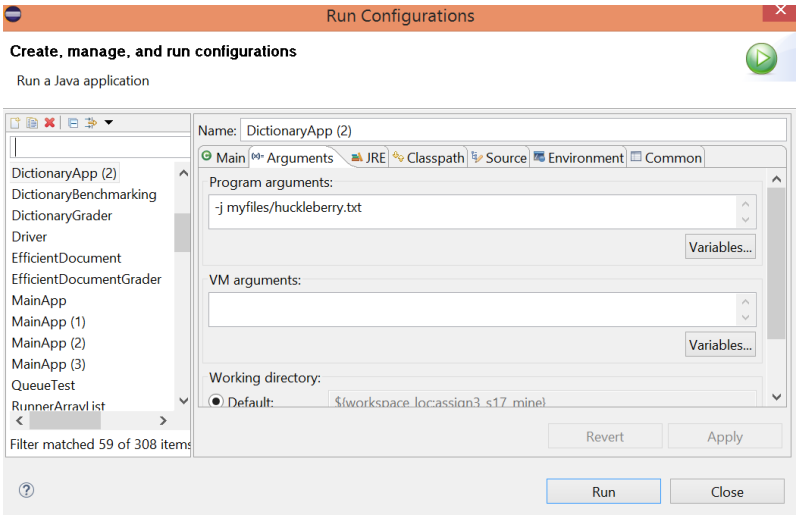
Starter code is provided in *assign3_s17.zip*. It was created using Eclipse export feature, so to get it back, you need to import it into your workspace. File -> Import -> General -> Existing Project into Workspace -> Select Archive File.

MyTreeMap.java

This is an interface that BST, AVL, and Splay tree must implement. When using generic names for the key and the value in those trees, use the same names, i.e. *AnyKey, AnyValue*.

DictionaryApp.java

This is a driver program written by Donald Chinn and modified by me (Monika Sobolewska). The driver parses the command line argument to determine which algorithm is supposed to be used, and what input file is to be used. It also sets up a timer to time how long it takes to read in and insert the words of certain lengths into the dictionary and then to traverse the dictionary and print it out. There is a method called *getWord* in the driver, which gets the next sequence of contiguous alphanumeric characters from the input stream. This is what we will consider to be a word for the purposes of this assignment. Take a look at the *main* method to see how it is used. You may want to run it first using the Java built-in map on one of the provided text files to get an idea as to how the program ought to work. You will need to add the code to make the driver work for the other trees and add the second timer that measures printing only. To run the program in Eclipse with command-line arguments, you need to select Run -> Run Configurations -> Arguments. Then you will type in your argument, e.g. -j myfiles/huckleberry.txt



BST, AVL and Splay Trees

The only operations that each tree has to implement are the operations provided in the interface. You are given the code from the textbook for the BST, AVL, and Splay trees, written by Mark Allen Weiss, that you need to modify to implement the interface. Your methods should use recursive solutions, so you will need private helper methods, and each tree node needs to be modified to contain both the key and the value component. Name these components *key* and *value*. Do NOT change any other class or data component names. It also follows that insert and remove will need to be changed in a fashion consistent with a dictionary – if an item is already in the dictionary, do not add a new node but rather replace the value in the existing node that matches the specified key. If an item is removed, you are to remove the node. The generics template needs to be changed to account for two different types: one for the key and one for the value. *toString* method should be traversing the tree in inorder (alphabetic one). All the functionality should be consistent with Java TreeMap class (e.g. look at toString() format before writing your own toString method). When you submit your code, do not retain any Weiss's methods that are not used in your solution.

Extra credit:

Include additional data structures, e.g. a list, another built-in Java data structure, and run experiments on those data structures as well. Or, provide a tree iterator that the client can use to traverse the trees. If you do that, add another driver, *DictionaryAppEC.java* that uses additional data structures and/or the iterator.

HELP

Whenever in doubt or unsure, your first approach should be to try to solve the issue on your own by consulting the textbook for this course, any other Java textbook you may have from your prior courses, and online Java documentation. Java documentation is available at http://docs.oracle.com/javase/8/docs/api/

When unable to resolve an issue, code the components you can figure out on your own and seek CSS Mentors' help with the issues you need help with, visit instructor's office hours, and stay after class. Ask questions of your classmates but be careful so that you do NOT cross the boundary between helping and plagiarizing.

GRADING RUBRIC

- Report 30%
- BST tree implementation 20%
- AVL tree implementation 20%
- Splay tree implementation 20%
- Overall correctness and coding style 10%
- Extra credit 15%