COMPX201/241-19A — Assignment Three

Binary Search Trees

Due: Wednesday, May 22nd, 2019 — 11pm

Description: This assignment is intended to give you experience maintaining a binary search tree (BST) of words. You must write your own BST class and all its required operations. Your program must compile and run on the Linux machines in R Block.

Input/Output: Your program should run as a console program from the Linux command-line (i.e. no GUI). Your program class should be called OddWords.java and it should accept as a command-line argument the name of a text file expected to be in the same directory. Your program is to process the contents of that file as a stream of words according to the following specification:

- a word is any contiguous sequence of alpha-numeric characters — specifically, the letters A to Z in either upper or lower case, and digits.
- any other characters are ignored (i.e. treated as blank space between words)
- any word found on input must be converted entirely to lower case letters (n.b. the java String class has a toLowerCase method that can be used for this)
- given the above, it follows that the contents of the input file can be treated as a stream of words composed entirely from lower case letters and digits
- write code to maintain a BST of words that can insert, delete and find words (i.e. words are keys); your BST must be a class called BSTlex defined in a file called BSTlex.java; it must make use of self-referential structures, and you must define all its methods
- write four different JUnit tests for each of the insert, delete and find operations, making sure you think about testing for "edge" cases, like an empty BST.
- your program OddWords must use an instance of your BSTlex class to achieve the following:
  - each word found on input must be added to the BST if it is not already there, but deleted if it is found
  - each time your program searches for a word (either for insertion or deletion) all the words along the search path down the BST must be printed out on a single line, separated by spaces, with the target word appearing last.
  - if the word is not found, it is added to the tree and the label "INSERTED" is to appear after the target word at the end of the output line.
  - if the word is found, then it is removed from the tree and the label "DELETED" is to appear after the target word at the end of the output line
  - write four different JUnit tests for your program class: two for the case where a word to be added is not in the BST, and two for the case when the word is already in the tree
- reiterating: each search for a word produces one line of output that includes a space-separated list of all the words found along the search path, in the order they are examined, followed by the word being looked for and either the label "INSERTED" or "DELETED" at the very end.
- once the entire file has been processed, your BST will obviously contain all words that appeared an odd number of times in the file, because any that were found an even number of times will have been deleted, perhaps many times over.
- after the file has been processed, have your program output the label "LEXICON:" on a line by itself.
- following the "LEXICON:" label, print out all the words still in the BST, one word per line and in dictionary order — that is, you must do an in-order traversal of the BST and print out all the words still in it.
- all output is to standard out

A sample file and its correct output will be posted on the course webpage (i.e. Moodle) so that you can check your program conforms to this specification.

Submission: Make sure your code is properly formatted and well-documented, including your name and student identification number. Submit your source code only (no .class files or data) using Moodle. Ideally, your submission will be made as one archive file (see Linux "tar" or "zip").

Tony C. Smith, 28/4/2019