COMP 250: Introduction to Computer Science Fall 2017

Assignment #3

Prepared by Prof. Michael Langer
Posted: Monday Oct 30
Due: Sunday Nov 12, 23:59 PM

General Instructions

- The T.A.s handing this assignment are Ram and Navin. Their office hours and location and email contacts will be posted on mycourses Announcements.
- Same general instructions as Assignments 1 and 2, in particular, the late penalty.
- *We will deduct at least 20 points for any student who has to resubmit after the due date (i.e. late) because the wrong file or file format was submitted.* This policy will hold regardless of whether the student can provide proof that the assignment was indeed "done" on time. This includes submitting a .class file, starter code, a rar or 7z format.

Submission Instructions:

Submit a single zipped file A3.zip which contains the modified ***WordTree.java*** file, to the myCourses assignment A3 folder. Include your name and student ID number in the comment at the top of the ***WordTree.java*** file.

In this assignment, you will work on the problem of storing a set of *n* words in a tree data structure, and a method for efficiently finding all the words that have a given prefix. You should be familiar with the latter problem through the autocomplete feature found on cell phones, web forms, Eclipse. For instance, if you type in "*aar*", then the autocomplete feature may suggest *aardvark, aardvarks, aardwolf, aardwolves, aargh*.

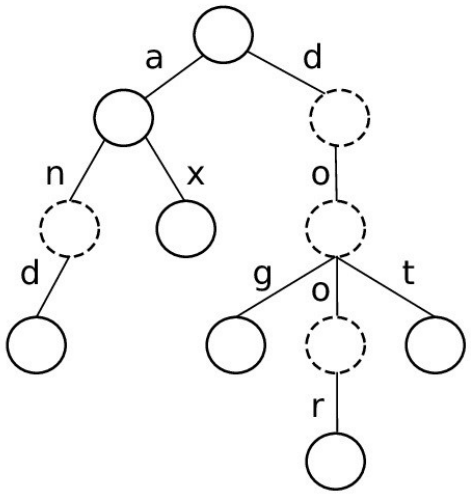
By *word*, we mean a string formed from the set of ASCII characters, specifically alphanumeric characters a-zA-Z. See http://www.asciitable.com/ for a list of all ASCII characters. Note that ASCII encodes text using one byte or 8 bits per character.

WordTree Data Structure

Here we define a special kind of rooted tree that we use to efficiently index words by storing their prefixes. The main property of this tree is that *each edge corresponds to a character*. Thus the path from the root of the tree to any node in the tree defines a string. The string defined by each node is a prefix of all strings defined by the descendents of that node.

Below is an example tree. It contains the following words: *a, and, ax, dog, door, dot.* You will note that the dashed nodes correspond to prefixes in the tree (*an, d, do, doo*) that are not in our list of words. The tree must also keep track of this distinction by storing for each node whether it corresponds to a word or not.

Note that if *C* is the set of all possible characters defined at the edges, then each node can have at most *k* = | *C* | children, where the | C | notation just means the number of elements in set C.



What You Need To Do

Read the provided code (***WordTree.java, A3TesterPosted.java***), including the comments which explain what the various methods do. *This will take you some time, so go slow and read the code carefully.* Note that the ***WordTree*** class has a private inner class ***TreeNode***.

For the ***WordTree*** class, fill in the missing code for the following methods:

- *createChild(), toString()* for the inner class ***TreeNode***
- *getPrefixNode(), insert(), contains(), getListPrefixMatches()*

We suggest you implement them in the order listed above. To get started, you may wish to add a main() method to the ***WordTree*** class. Eventually you should move on to using the Tester file. Don't forget to delete the main method when you submit !

You may only use Java String methods *length()* and *charAt()*. In particular, do not use String searching methods such as *String.startsWith(), String.substring()*, etc. The point of the assignment is for you to organize and compare strings using this tree data structure.

Only add code in the area where it says to add code. The grader may be sensitive to any changes that you make outside that area. Any helper methods that you wish to write should be added at the end of the ***WordTree.java*** file.

Note that all methods and also the inner class have been given the *public* access modifier. This was done to simplify grading. As you will learn later in the course, typically one does not make all methods public.

Submit only the file ***WordTree.java*** as a zip file. Use the ***A3TesterPosted.java*** file to test your code. You do not need to submit this tester file.

Good luck and have fun!