Assignment

Homework 13 - Spell checker

In this assignment, you will build a trie that functions as a dictionary for a spell checker. The spell checker is a program with a Graphical User Interface (GUI), which has already been implemented. You can download the main spell check program and the template file for your data structure using the starter files link.

The SpellCheck program

The class `SpellCheck` implements a GUI for doing dictionary lookup based on the words found in a text file. Its `main` method is invoked with an array of `String`s, where each `String` is the name of a file from which to read words into the dictionary. (There can be more than one.) The dictionary is contained in the variable `dict` of type `WordDictionary`.

Words are read into the dictionary by parsing the file(s) and calling the `dict.add()` method of the dictionary for each word found. (You can use any of the sample dictionary files to test, although you probably want to start with even smaller files that you create yourself in a text editor. Note that on arbitrary text files, the parser will assume words are separated by whitespace, and the parser ignores any non-alphabetic characters.) Once words are successfully read into the dictionary, the `SpellCheck` program opens a GUI window. The top pane contains a toolbar with a text field, a spell button, a distance value, and a number of results to return.

As a person types into the text field, calls are made to `dict.getCompletions()` to return words in the dictionary that start with the prefix typed into the text field. The number of words returned should equal the number specified in the window. When the user presses Enter or clicks the Spell button, a call is made to `dict.findClosest()` to find the words in the dictionary most close in spelling to the one typed into the text field. The maximum number of letters that can be different in any of the suggested words is the number in the hamming-distance text box in the toolbar.

Spelling suggestions and completion suggestions are displayed in the main text pane of the `SpellCheck` window.

Levenshtein Distance
The Levenshtein distance between two strings is the minimum number of substitutions, insertions, and deletions used to transform one string into another.

For example:

Distance 1: boat - moat, guard - guards, farm - form, stone - stove

Distance 2: give - lived, grow - flow, placed - lace, green - greener, flip - fair, undo - redo

Distance 3: pot - potter, a - able, chance - clack

The Levenshtein distance will determine the extent to which you spawn branches in your traversal of the trie. **Thus, it is recommended that you keep the distance limit small.**

Your Assignment

You must fill in the code to implement the data structure for the `WordDictionary`. It is recommended that you use a Trie. The following methods need to be implemented:

1. `boolean WordDictionary.add(String str)`
   > This method will add the word `str` to the dictionary. Note that it should be case-insensitive. Return `true` if the word was successfully added; `false` if the word was already there or some other problem occurs. You can assume the input strings contain only alphabetic characters.

2. `Collection<String> WordDictionary.getCompletions(String str, int max)`
   > This method will return up to `max` words that begin with the prefix `str`. If you are using a Trie, this amounts to returning up to`max` words in the subtrie rooted at `str`. Return the words in some sort of Java library data structure, for example, a `LinkedList`.
   >
   > If there are no completions, you can either return null or return a collection of size 0; either will make the program display "no matches."
   >
   > In your implementation, you should prefer shorter words to longer; *e.g.*, if the prefix is "ca", it's better for the user to see "cab" and "cat" than "cat" and "cattle." If the prefix is itself a word, it can be returned too.

3. `Collection<String> WordDictionary.findClosest(String str, int dist, int max)`
   > This method will return up to `max` words that are closest to the spelling of `str`. The words you return must differ in at most`dist` characters from `str` (in the Levenshtein-distance sense).
   >
   > If there are no close words, then return either null or an empty collection; both will make the program display "no suggestions."
   >
   > Ideally your implementation should prefer to include "closer" words in the collection that is returned, *e.g.*, don't return words at distance 3 if there are some at distance 1.

Submission

Upload only your WordDictionary.java file using the link below.