Project 7: Spellcheker

CMS 170, Fall 2014

Due Thursday, December 4, 11:59:59 PM

Description

In this project, you'll use a binary search tree to implement a spelling checker tool. On creation, the checker will open a large list of common English words and construct a search tree. The checker can then search the tree to determine if it contains a given input word; if not, the word is either uncommon or misspelled. For additional fun, your checker will support a method to suggest possible corrections to misspelled words.

Class Description

Implement a class called `Spellchecker` that provides the following methods:

- `public Spellchecker(String word file)`: constructor that opens the given word file and builds a binary search tree of the words it contains
- `public boolean search(String searchWord)`: search the tree for the given input word; return `true` if it is present and `false` otherwise
- `public void traverse()`: helper function to print the contents of the tree; if your insertion method is correct, the traversal should be in alphabetical order
- `public void doTraverse(TreeNode node)`: method that actually performs the recursive in-order traversal of the tree
- `public ArrayList<String> suggestions(String word)`: return a list of words that could be correct alternatives for the given input word
- `public void checkFile(String filename)`: read the words in a file and spell check each one; print all of the misspelled words along with their suggestions

In addition, your `Spellchecker` class should contain a private `TreeNode` class that represents a single node in the binary tree. The data value of the `TreeNode` is a `String`: the word the node contains.

The project zip file contains a file called `shuffled_word_list.txt` that you can use to create your tree. I have randomized the order of the list so that your binary search tree will be roughly balanced without any extra work on your part. All of the words in the list are lowercase; convert each word you check to lowercase before searching the tree.

The starting point of your implementation should be the `BinarySearchTree` example we discussed in class. Note, however, that a few things need to change. In particular, make sure that you are handling the `String` data correctly when performing comparisons.

- Use the `equals` method, not `==`
- Use `compareTo`, not `<` or `>`

Check the `String` class documentation if you have questions about these methods.

Suggestion Lists

The `suggestions` method returns an `ArrayList<String>` containing possible corrections to its input `String`. In your implementation, the corrections that are returned will be the words in the search tree that have an *edit distance* of 1 from the input word. An edit is one of the following operations:

- Inserting a new character into the word, including a new character at the beginning or end
- Changing one character of the word into another character
- Deleting one character from the word

Therefore, your `suggestions` method can work by taking the input word, generating all possible `String`s that have an edit distance of 1, and returning the subset of those that exist in the word list.

To get you started, here's an code fragment that generates all possible character substitutions:

```
String letters = "abcdefghijklmnopqrstuvwxyz";
for (int i = 0; i < word.length(); i++) {
  for (int j = 0; j < letters.length(); j++) {
    String candidate = word.substring(0, i)
        + letters.charAt(j) + word.substring(i+1, word.length());

    if (this.search(candidate) == true) {
      // Add candidate to output ArrayList
    }
```

```
  }
}
```

Take a look at `String`'s `substring` method; it will be useful for the insertion and deletion edits.

Testing

The zip file that contains this document also contains `the_raven.txt`, a poem with some misspelled words. Process that input file and identify all of the misspellings along with their suggested corrections. Note that the poem might contain some words (like "Lenore") that aren't in the word list; it's fine to flag these as misspelled, since there's no way to identify them as correct outside of the context of the poem.