Tuesday, February 7, 2023

CS 2210B, Winter 2023
Assignment 2 (Programming)
Due February 16, 11:55 pm

Overview

In this assignment, you will write a simple spell-checker. Your program should be named ***Spell*** and take two file names as command-line arguments. Your program will be invoked from the command line as

```
Spell dictionary.txt fileToCheck.txt
```

The first file name is the dictionary with correctly spelled words, which I provide. The second file is the text to be spell-checked. Your program should check all words in fileToCheck.txt. Nothing needs to be done for words that are correctly spelled, i.e. those found in the dictionary file. Your program should suggest possible correct spellings for words not in the dictionary file by printing to the standard output. You should perform the following modifications of a misspelled word to handle commonly made mistakes:

- Letter substitution: iterate over characters in the misspelled word, trying to replace the current character with a different character. For example, in a misspelled word' lat', substituting' c' instead of' l' will produce the word' cat' in a dictionary. There are 25 different characters to try for the current character. Thus, if the length of a word is *k* characters, the number of modifications to try is 25*k*.
- Letter omission: try to omit (in turn, one by one) a single character in the misspelled word and check if the resulting new word is in the dictionary. For example, if the misspelled word is' catt', omitting the last character' t' will produce the word' cat' in the dictionary. In this case, there are *k* modifications to try where *k* is the number of characters in the word.
- Letter insertion: try to insert a letter in the misspelled word. For example, for' plce', inserting' a' after' l' produces a correctly spelled word' place'. If a word is k characters long, there are 26*(*k*+1) modifications to try since there are 26 characters to insert and *k+1* places (including in the beginning and at the end of the word) to insert a character.
- Letter reversal: Try swapping every pair of adjacent characters. For example, in' paernt', swapping' e' and' r' produces a correctly spelled word' parent'. For a word of length *k*, there are *k - 1* pairs to swap.

This assignment has 10% extra credit from the total assignment grade for comparing the Hash Table-based dictionary to a linked-list-based dictionary. The extra credit portion will transfer to the total assignments' grade but not toward exams.

Example Input and Output

File `'toyDictionary.txt'` contains the words:

cats, like, on, of, to, play

And file `'fileToCheck.txt'` contains the words:

Catts lik o play

The output of `Spell toyDictionary.txt fileToCheck.txt` is as follows:

```
catts: Incorrect Spelling
catts => cats
lik: Incorrect Spelling
lik => like
o: Incorrect Spelling
o => to, of, on
play: Correct Spelling
```

Each misspelled word should be placed on a new line. The list of suggested correct spelling may not contain repeated words. In the example above, for the misspelled word' catts', removing the first' t' or the second' t' leads to the same word' cats', but' cats' should appear in the output only once. If no modification of the word produces a correctly spelled word, your program should print out "**no suggestions**" for that word.

Implementation

You are to implement the program based on a dictionary data structure. First, your program should read all words from the input dictionary file (specified by the first command line argument) and insert them into a dictionary data structure, let us call it *D*. Then your program should open the second file (containing the text to be spell-checked). As you read words from the second file, if any word is not in *D*, that means it is a misspelled word that needs to be handled. You must try all possible modifications of the word outlined in the 'overview' section.

It's expected that your implementation will convert all words into lowercase so that the words 'Cat' and' cat' are treated the same. Thus, all the words you read from a file using your program will be lowercase.

Provided Files

- Spell.java: A template that you can use as a starting point to implement your program. You might find the comments in the file useful. However, feel free to use such a template or come up with your own Spell.java file.
- TestHashDictionary.java: This is a program the TA will use to test your implementation. Compile and run it after you have implemented your program. It will run some tests and let you know which tests are passed/failed. Each test is worth 4 marks.
- toyDictionary.txt: Use it as demonstrated above
- dictionary.txt: This is a larger dictionary file that contains 10,000 words.
- fileToCheck.txt: Use it as demonstrated above

Note: All files are hosted under OWL-> Resources -> Assignments -> Assignments 2.

Extra credit: Hash Table vs. Linked List Dictionary Implementation

- Implement a dictionary based on a linked list. You can use Java's built-in class LinkedList for this purpose. Your linked-list-based dictionary must follow the Dictionary interface.
- You will create six dictionary files of different sizes (d1.txt, d2.txt, ..., d6.txt). Run your program with these six different dictionaries and the same file to check the spelling of words, the file' fileToCheck.txt'. Namely, the second command line argument stays the same, while the first goes through d1.txt, ..., d6.txt.
- Get the running time using System.currentTimeMillis(). Note that this method will return the current time, not the running time from the program's start. Therefore, to get the total time (in milliseconds) your program took to complete, you should measure the time at the very start of the program at the very end and subtract the two. Since what changes between the different runs is the size of the dictionary file, you should plot the running time vs. the size of the dictionary file, that is, the number of words in the dictionary file. Count the number of words in each dictionary file and plot, on the same chart, the number of words versus running time for the list and hash-based dictionary implementation.
- The running time is essentially the time it takes to insert all the dictionary words since the second file for spell-checking has only a few words to check. When we insert a word into a dictionary, we must check if that word is already in the dictionary. For a hash table, checking and inserting are expected to take a constant amount of time, and therefore inserting all elements in the dictionary should take linear time. For a linked list, inserting is a constant amount of time, but checking if the element is already in the list is a linear amount of time. Therefore, inserting all elements in the dictionary should take non-linear time. Thus, hash-based implementation running time plots should resemble a linear function, and list-based implementation should resemble a non-linear function.

Coding Style

Your mark will be based partly on your coding style.

- Variable and method names should be chosen to reflect their purpose in the program.
- Comments, indenting, and whitespace should be used to improve readability.
- No variable declarations should appear outside methods ("instance variables") unless they contain data to be maintained in the object from call to call. In other words, variables needed only inside methods, whose value does not have to be remembered until the next method call, should be declared inside those methods.
- All variables declared outside methods ("instance variables") should be declared private (not protected) to maximize information hiding. Any access to the variables should be done with accessor methods (like key() and value()).

Grading

Your grade will be computed as follows.

- Spell class implementation with necessary methods: 25 marks.
- Program compiles produce a meaningful output: 15 marks.

- HashDictionaryTest pass: 40 marks, 10 tests; each test is worth 4 marks.
- Coding Style: 20 marks
- (Extra credit) Comparison chart of Linked List vs. Hash table running times: 10 marks.

Handing In Your Program

Submit your assignment via the online OWL system as scheduled above.

Good Luck!