Due: 11:59pm Wednesday, November 29

Lab 2

Fall 2017, Bhakta
CS 315, Algorithms

Overview In this lab, you will implement a rudimentary spell checker. Your lab will read a input text file and correct any spelling errors it has by replacing the misspelled words with its best guess of what the user meant by those words.

Spell Checking These are the assumptions that our rudimentary spell checker will make.

- If an input word is not valid in your dictionary, it would make sense that the user meant to input a word that was "close" to what they actually typed. So we need a notion of distance between words.
- In class, we studied the Edit Distance, which is a measurement of how many changes (additions, deletions, and replacements) are needed to turn one word into another. This is a good place to start.
- In practice, not all modifications are equal. Although "hig" and "hug" have edit distance 1, and "hig" and "pig" also have edit distance 1, it is more likely that a user who typed "hig" meant "hug" because "i" and "u" are very close on the keyboard while "p" and "h" are farther apart. You are more likely to mistype two close characters than insert a new character, but more likely to insert a character than mistype two far apart characters. You are more likely to miss a keystroke than add an extra one.

Directions: We will use a modified version of the edit distance algorithm to better capture these realities of spell checking.

- Pre-compute distances between letters in a keyboard. Our definition of distance between letters will be work as follows: Give each letter a coordinate on the x,y grid. "Q" will be at (0,2), "W" will be at (1,2), etc. "A" will be at (1/3,1), "S" will be at (4/3, 1), etc. and "Z" will be at (2/3,0), "X" will be at (5/3,0), etc. The distance between letters will be the normal Euclidean distance between these points - the sum distance between the coordinates. For example, . You should be able to compute these distances with a couple of for-loops and a little math starting with the String array ["qwertyuiop","asdfghjkl","zxcvbnm"].
- Implement a modified version of the edit distance algorithm (which finds the fewest number of changes needed to change one word into another) where you instead find the smallest cost needed to change one word into the other.

  The cost of changing a letter to another letter is:

  - Modify character "x" to character "y": $d(x, y)$ where $d(x, y)$ is the distance between x and y as above.
  - Insert a character : 7
  - Delete a character : 6

  Your function should take two strings as input and output a number.
- Finish the rest of your implementation. You should read a string from standard input, replace each word with the closest word in the dictionary, and output these words to standard output. Ignore capitalization. You may write in any programming language that you like.

Bonus Disclaimer: Our rudimentary spell checker isn't nearly as good as modern ones. All the numbers used in the computation above were basically made up in a semi-reasonable way, and not at all optimized for reality.

Not only is the above approach likely slower than modern spell checkers, even with our smaller dictionary, but modern spell checkers also use much more sophisticated models of distance between words. They can take into consideration that mistypes between letters that would be typed by the same finger are more common, and some words are just commonly misspelled. You are also more likely to have a mistake near the end of the word than near the beginning - people will usually get the first characters right.

More advanced spell checkers can factor in the likelyhood of usage of a word (you are more likely to use the word "discuss" than "discus"), the context of other words in a sentence ("pretty ood" is more likely to be "pretty good" than "pretty food") , and some can even infer subject matter from other parts of your writing (you are more likely to use the word "dog" when writing about animals than "log").

You may earn (3-15) bonus points for *every* additional improvement you make to the algorithm above, depending on the improvement. Submit the unimproved algorithm in a separate file from the improved one. Looking up more accurate distances for the above is worth (3-5) points depending on how many changes you make. Getting a faster running time by restricting the search set of words intelligently will be worth 10 points, and anything more advanced would probably be worth (15+). Talk to me if you have other ideas for improvements and how many points they would be worth.