CSE250 Fall 2016
Assignment A7 – Spelling Assistant

Due: 12/09/2016, 11:59PM (note that this is Friday!)

Last updated: *2016-11-28 13:59*

Objectives

- Practice the use of dictionary.
- Implement interesting tool.

Introduction

Spelling correction is one important operation frequently showing up in online systems, text editors or UIs in mobile devices. Implementing a full scale corrector is a non-trivial task, however, we can consider slightly simplified version, focused on names and considering mistakes that are only edit distance of 1 from the correct spelling.

Your task in this assignment is to implement a tool that using an input dictionary of names (together with their frequencies) will propose correct spelling for a stream of misspelled names. To simplify the task, the best suggestion of correct spelling is a word that is edit distance of 1 or less from the misspelled word and has highest frequency.

We define edit distance as the minimal number of substitutions, deletions and insertions that we have to apply to one word to obtain the other. For example, suppose that we are using only upper case letters and consider name `JON`. There are 3 words that are edit distance of 1 from it that can be created by deletion: `ON`, `JN` and `JO`. There are 4×26 possible words that are edit distance of 1 that can be created by insertion, starting with `AJON` and ending with `JONZ` (26 because of the size of the Latin alphabet). Finally, there are 3×25 possible words that are edit distance of 1 that can be created by substitution, starting with `AON` and ending with `JOZ`. In general, for a given word s of length n there are $n + 26 \times (n + 1) + 25 \times n$ words that are edit distance of 1 from s.

Instructions

1. Create directory `A7` where you will place your code.
2. Create `Makefile` to automate compilation of your code. Your main source code file should be named `a7.cpp` and it should compile to `a7` executable. You can directly adopt `Makefile` from Assignment 0 (replace `a0` with `a7`).
3. Implement spelling corrector given the following specification:
   (a) Your program should take as an input a dictionary file in which every line stores one name, all upper case, and frequency of that name. For example, if invoked like this: `./a7 dict.txt` your program should read dictionary from a file `dict.txt`.
   (b) Dictionary provides information about correctly spelled names and their frequencies. Name is a single string and frequency is a positive integer. For example, the dictionary below contains three names:

```
BILL 10
JON 100
JIN 50
```

where `JON` is the most frequent.

(c) Your program should take query names from `std::cin` as long as they are provided. For example see how input is taken in `a5.cpp` that was provided to you.

(d) For each query name your program should do the following:

i. If the query name is already in the dictionary, print it together with its frequency separated by space. Frequency of the query name in the dictionary should increase by 1.

ii. If the query name is not in the dictionary, provide a suggestion that is in the dictionary and is edit distance of 1 from the query (this should be the only action taken). If multiple suggestions exist, provide one with the highest frequency, and if several suggestions have the same frequency return the one which is lexicographically smallest. If no suggestion exists print `-` and include the query name in the dictionary with frequency 1.

(e) You can assume that all input data is correct, i.e. dictionary file always exists and is correct, all query and dictionary names are provided in upper case.

To illustrate how your program should work, consider the dictionary below stored in `dummy.dict`:

```
JON 100
JIM 90
BILL 1
WILL 2
```

Your code if invoked like this: `echo "GILL BILL BILL DILL SANTA SANTA" | ./a7 dummy.dict` should produce the following output:

```
WILL 2
BILL 1
BILL 2
BILL 3
-
SANTA 1
```

This is because `GILL` is edit distance 1 from `WILL` and `BILL`, but `WILL` has higher frequency and `SANTA` initially is not in the dictionary.

Extra Challenge

The extra challenge on this assignment is to get 100pt for the main problem. If you get 100pt, you will be automatically upgraded to 120pt.

Submission

1. Remove your binary code and other unrelated files (e.g. your test files).

2. Create a tarball with your `A7` folder.
3. Follow to https://autograder.cse.buffalo.edu and submit `A7.tar` for grading.
4. You have unlimited number of submissions, however, any submission after the deadline will have 50% points deducted.

Grading

- 10pt: `a7.cpp` compiles, runs and has no memory leaks.
- 90pt: If you pass the initial test, there will be nine benchmark tests. You will get 10pt for each correctly completed test.
- If your code is **extremely** inefficient, for instance due to infinite loop, autograder will terminate your code and you will receive 0pt.

Remarks

- Make sure that all files and directory names are exactly as instructed. Otherwise the grading system will miss your submission and you will get 0pt.
- Winter is coming.