CSE250 Assignment A5 – Spelling Assistant

Due: 08/14/2019, 1:59PM

Last updated: 2019-08-05 20:28

Objectives

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

Introduction

Spell checking is an 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 a slightly simplified version, focusing 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 spellings 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.

Hints

A map (`std::map<>`) or an unordered map (`std::unordered_map`), along with other data structures that we have already discussed in class, may be helpful for this assignment.

Instructions

1. Create directory `A5` where you will place your code.
2. Create `Makefile` to automate compilation of your code. Your main source code file should be named `a5.cpp` and it should compile to `a5` executable. You can directly adopt `Makefile` from Assignment 0 (replace `a0` with `a5`).
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 the frequency of that name. For example, if invoked like this: `./a5 dict.txt` your program should read the dictionary from a file `dict.txt`.

(b) Dictionary provides information about correctly spelled names and their frequencies. Name is a single nonempty string of uppercase letters 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.

(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 current frequency separated by space. Frequency of the query name in the dictionary should then 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.

4. Print a line with five dashes: "`-----`".
5. Output the final state of your dictionary in any ordering.

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 JILL SANTA SANTA" | ./a5 dummy.dict
```

should produce the following output:

```
WILL 2
BILL 1
BILL 2
BILL 3
-
SANTA 1
-----
JON 100
JIM 90
BILL 3
SANTA 2
WILL 2
```

Notes: The first line is because `GILL` is edit distance 1 from `WILL` and `BILL`, but `WILL` has a higher frequency at that time. The first `-` is caused by `SANTA`, which is not initially in the dictionary.

Submission

1. Remove your binary code and other unrelated files (e.g. your test files).
2. Create a tarball with your `A5` folder.
3. Follow to https://autograder.cse.buffalo.edu and submit `A5.tar` for grading.
4. You will have five submissions. Any submission after the deadline will have 50% points deducted.

Grading

- 10pt: `a5.cpp` compiles, runs and has no memory errors.
- 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 an 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.
- Make sure that you start working on the assignment early! If you wait to the very last moment, you may expect delays from autograder (overloaded by other students who like you waited with their submission).
- If you are having trouble, have a public discussion on piazza! Before posting your question, ask yourself: Am I giving enough information where someone who is not looking at my code can still help? Can you describe tests that you have already tested? Do not, of course, share your code.
- Feel free to share dictionary tests as well as input tests on Piazza. Once again, do not share your code.