Final Project

RELEASE DATE: 05/14/2013

DUE DATE: 06/27/2013, **noon** ON CEIBA

Unless granted by the instructor in advance, no late submissions will be allowed. Also, the gold medals cannot be used on the final project.

Introduction

The main theme of the final project is a spell checker program. The spell checker program helps check whether each word in the article appears in a prescribed dictionary. If not, the word is considered mis-spelled. Of course, we expect the checker to effectively use the computational and storage resources of your computer. So the data structure (and the associated algorithm) used for representing the dictionary can be crucial.

Problem Description

The spell checker needs the following basic functionality. You can use your own function prototype for implementing the functionality.

- *build(text_file, dictionary_file)* converts a text file to a binary dictionary file that matches your data structure
- *dictionary = load(dictionary_file)* loads the dictionary file to the memory
- *check(dictionary, word)* checks whether a given word is within the dictionary in the memory
- *words = basic_suggest(dictionary, word)* returns a list of suggested words from the dictionary
- *add(word, replacement)* records that the user intends to replace "word" with "replacement"

When *add(word1, word2)* is executed, you should increase a counter `count[word1, word2]` that records the number of times that the user replaces `word1` with `word2`. The counter is initially 0, of course.
For the function *basic_suggest*, consider the following common typos:

- missing one character from a word, like dictionary → dictonary
- adding one character to a word, like dictionary → dicktionary
- typing more than needed for a word, like dictionary → dictionaryblah
- typing less than needed for a word, like dictionary → dictio
- replacing one character in a word, like dictionary → dicsionary
- switching the order of two or three consecutive characters in the word, like dictionary → ditcionary or dictionary → diictonary

The function should try to see if the mis-spelled word comes from a typo of any word in the dictionary. If so, it include the word in the suggestion list. The list should also include any "replacement" that the user has provided to the suggestion list. Then, duplicated items in the suggestion list shall be removed and the unique suggestions need to be sorted by

- firstly, `count[mis-spelled word, suggestion]`, from high to low;
- secondly (if there is a tie), the lexicographic order.

For instance, for a mis-spelled word `aet`, if there are three other words in the original dictionary `eat`, `at`, `pet`, and the user replaces `aet` with `at` twice, and replaces `aet` with `the` once (note that `the` will not be suggested by the common typos and is user-entered). Then the suggested list should be ordered like

```
at the eat pet
```

Survey Report

You are asked to study at least **THREE** data structures for dealing with the dictionary. Then, you should make a comparison of those data structures according to some different perspectives, such as average speed, worst speed, space, implementation, popularity, etc.. Based on the results of your comparison, you are asked to **recommend** the best one for the spell checker program, and provide the "cons and pros" of the choice.

The survey report should be less than or equal to **ten** A4-pages with readable font sizes and formats. Criteria for evaluating your survey report would include, but are not limited to, clarity, strength of your reasoning, "correctness" in using the data structures, and the work loads of team members.

Competition

We will hold a mini-competition for the project. Each team is asked to submit the source code of your spell checker to be automatically compiled on the CSIE Linux machines for the mini-competition. The competition ends at 6:00AM, 06/20/2013.
The mini-competition tests the basic functionality. In particular, you need to provide a program named `project` such that

```
./project -b text_file dictionary_file
```

reads the `text_file` and outputs a `dictionary_file`.

Furthermore, the spell checker is invoked with the following command:

```
./project -d dictionary_file input_file
```

The input file will contain two kinds of lines:

```
c word
```

means checking a word;

```
r word replacement
```

means replacing a word with replacement. When checking each word and finding that it is not in the (original) dictionary, please output the basic suggestions in a line like:

```
aet: at eat pet
```

The exact formats and the sample files will be announced online.

Three things will be tested in the competition:

- the accuracy of the spell checker, i.e., the percentage of output lines that represent correct suggestions
- if the accuracy is 100%, the size of the *dictionary_file* produced
- if the accuracy is 100%, the speed of your spell checker

Every team is asked to submit to the mini-competition at least three times (with the three data structures used) and list the results in the survey report. Of course, more submissions are encouraged and welcomed.

Submission File

Please upload a single ZIP compressed file (.zip) to CEIBA. The ZIP file should contain ONLY the following items:

- the source files of your final spell checker, including any package you use (see below); those source files can be different from what you submitted to the mini-competition
- the report with at most ten A4 pages in PDF format. The report should contain the following items:

  (1) the team members' names and school IDs

(2) how you divide the responsibilities of the team members

(3) the data structures you compared, including the results submitted to the mini-competition site

(4) the data structure you recommend

(5) the advantages of the recommendation

(6) the disadvantages of the recommendation

(7) how to compile your code and use the spell checker

(8) the bonus features you implement and why you think they deserve the bonus

You do not need to submit a printed version of your report.

Misc Rules

Report: No, you do not need to submit a hard-copy.

Teams: By default, you are asked to work as a team of size two. A one-person team is allowed only if you are willing to be as good as a two-people team. It is expected that all team members share balanced work loads. Any form of unfairness in a two-people team, such as the intention to cover the other member's work, is considered a violation of the honesty policy and will cause both members to receive zero or negative score.

Data Structures and Algorithms: You can use any data structures and algorithms, regardless of whether they were taught in class.

Packages: You can couple your spell checker with any software package (as long as you are not violating any copyright) but you need to *clearly cite where you get the code* and *clearly describing what the source code does* in your report.

I/O restrictions: Your program should only open the *text_file*, *dictionary_file*, *input_file*. Your program cannot open any other files nor access any other things through the Internet.

Platform and Language: As usual, you can only use C/C++ to design your main program. If you use packages from other languages, you still need to call them from C/C++. You can either use Linux or Windows as the running platform of your spell checker. But the submission to the mini-competition needs to be Linux-compatible with a Makefile (details to be announced).

Grade: The grading TAs would grade qualitatively with letters: A++[210], A+[196], A[186], B+[176], B[166], C+[156], C[146], D+[136], D[126], F+[116], F[76], F-[36], Z[0]. The score of the team would be the average of all the grading TAs. We reserve the possibility to adjust individual scores in the team based on performance/workload if necessary. The final project is equivalent to four usual homework sets; the midterm exam is equivalent to another four. Your raw score in the class would be calculated by

$$\frac{\text{best homework} * 1.5 + \text{worst homework} * 0.5 + \text{other homework} + \text{midterm} * 2 + \text{final} * 4}{12}.$$

If your spell checker meets the basic functionality and you write down every item reasonably in the survey report, you will get at least B.

Bonus: We encourage everyone to think about making your spell checker better. The room between B[176] and A++[210] is basically left for bonus. To get bonus points, you need to justify that the additional features/functionality of the spell checker is worth being the bonus in your report. For instance, you can try to compare more data structures/algorithms or add your creativity in designing some good data structures/algorithms for the spell checker. A fancy GUI may be another possibility, but not the only way and very likely not an important way. After all, *we are seeking for better data structures and algorithms in this class, not just better GUI.*

Collaboration: The general collaboration policy applies. In addition to the competitions, we still encourage collaborations and discussions between different teams.