Assignment 5: Tries
CS 758/858, Fall 2024
Due at **11:30pm on Wed, Sept 25**

Implementation

The skeleton code on the course web page is the start of a program to do spelling correction. Given a dictionary, the program constructs a data structure. Then the program is given a sequence of strings. For each string, it either reports that it is in the dictionary or returns a list of 'possible correct spellings', that is, words that differ by no more than one or two characters from the given string. Currently, the program uses a slow linear scan and generates suggested corrections by performing look-ups on all possible edited versions of an incorrect word. Improve the program by implementing a trie data structure and functions for using it. (You may assume that all characters in the strings will be alphabetic, although you need to support the distinction between upper and lower case.) When looking up incorrect words, design an appropriately limited traversal of the trie that visits only the edits that actually exist, instead of generating all possible edited versions of the incorrect word. Then measure the performance improvement.

For those in 758, 'words that differ by no more than one or two characters' means considering only character substitutions. Those in 858 should also consider character insertions and deletions. Each change (substitution, insertion, or deletion) counts as one 'edit,' and you need to find all words within an 'edit distance' of two. Adds and deletes should be off by default and enabled by the use of the `--adds-deletes` flag which is already parsed by the skeleton code.

Testing

On the course web page, we supply some utility programs and input files. Most of the programs we distribute in this class will tell you their command-line arguments if you run them with the `-help` option.

`spell-check-harness` runs your program, checks its output, and optionally displays a plot of performance. For example:

```
./spell-check-harness -c ./spell-check-reference -o list_output.pdf --adds-deletes
-r 1 -i data/dict2 2 4 8 16
```

will take a while with the reference, due to the poor data structure. Your program will be given sets (5 by default, but this can be changed with the `-r` option) of correct and incorrect words. The performance of your spell checker will be measured on both sets and if the `-d` option is given, a plot will be displayed showing the average time spend to perform the checks (X forwarding required, see programming handout for details). If you are running the harness on a system without an X display then you can use the `-o <filesuffix>` option to write the plots to files instead (suffix should end in ".pdf",".ps" or ".png").

Note that the harness does not pass a flag to your program indicating the data structure to use. You will need to make two executables (call the original one `spell-check-slow` and the new one with the trie `spell-check`) and specify which one the harness should use via the `-c` flag.

Also, note that the harness will crash if you ask for more test words than the size of the dictionary.

`dictionary` is in the `~cs758/data/asn5` directory. It contains 73,895 English words.

`dict2` is in the `~cs758/data/asn5` directory. It contains a subset of the words in `dictionary` (all those beginning with `a`). For example:

```
echo Alex | ./spell-check-reference data/dict2 - -
```

`wlist_match1.txt` is in the `~cs758/data/asn5` directory. It contains many strings, some of which are valid English words occuring in `dictionary`. It might be useful if you want to test your program without the harness (the harness generates invalid words automatically). If you want to test your program separately from the harness, you need to supply the solver with (in order) the data structure, the name of a dictionary file, the name of a file containing a list of query strings, and an outfile name (and optionally the `--adds-deletes` flag). An example would be:

```
./spell-check-reference  cs758/data/asn5/dict2 my-test-words.txt out.log
```

where `my-test-words.txt` is a file you create that contains one string per line, and result output is saved in `out.log`.

Written Problems

1. Briefly list any parts of your program which are not fully working. Include transcripts or plots showing the successes or failures. Is there anything else that we should know when evaluating your implementation work?
2. Problem 12–2 from CLRS.
3. Exercise 14.1–2 from CLRS.
4. Exercise 14.1–3 from CLRS.
5. What suggestions do you have for improving this assignment in the future?

Submission

Electronically submit using the script on agate (eg, `~cs758/scripts/sub758 5-undergrad your-asn5-dir`).

Evaluation

In addition to correctness, your work will be evaluated on clarity and efficiency.
Tentative breakdown:

8 trie

2 written problems