Lam Dang and Anh Dang
CS330
Group Written Assignment 9.2

I/Testing *(Using spellCheckTwoDistance)*

1/ Long words

-Created a file containing 4 long words: uncopyrightable(correct) subdermatoglyphic(correct) Unimaginatively(correct) honorificabilitudinittibus(incorrect)

-Run on Haskell

-Result:

- Time: 36s
- Words need fixing: uncopyrightable, subdermatoglyphic, honorificabilitudinittibus
- “uncopyrightable” return one suggestion , others doesn’t

- Observation:
  - Long words take a long time due to the large list of editted words it needs to generate and compare with dictionary
  - Long words tend do not exist in dictionary files provided
  - Long words have very few alternatives/suggestion
  - Have high chance of suggesting the correct word that we look for

2/ Short words

-Created a file containing 5 short words: can(correct) may(correct) fi(correct) dg(incorrect) lve(incorrect)

-Run of Haskell

-Result:

-Time: 2s
-Words need fixing: dg, lve
-Both words have 10 suggestions

-Observation:

- Short words take a short time due to the short list of edited word compare with dictionary
- Short words have more chance of occuring in the dictionary than long words
- Short words have a lot of alternatives
- Have low chance of suggesting the correct word we look for

3/ Common words

-Created a file containing 5 words: of(correct), th(acronym existed in dictionary), and h(existed in dictionary), beause(incorrect)

-Run on Haskell

-Result:
    -Time: 2s
    -Words need fixing: beause
    -beause have 10 suggestion
-Observation:
    -Common words take short time because most common words are short
    -Common words have higher chance of occuring in the dictionary
    -Common words have higher chance of suggesting the correct word we look for

4/Rare words
-Created a file of 4 words: biblioklpt(incorrect) acnesti(incorrect) grommet(correct) neldrop(incorrect)

-Run on Haskell

-Result:
    -Time: 5s
    -Words need fixing: biblioklpt, acnesti, neldrop
    -All three produce suggestion

-Observation:
    -Rare words take short time, but longer than common words because they are usually longer
    -Rare words have a lower chance of occuring in the dictionary
    -Rare words have higher chance of suggesting the correct word we look for, because they can be unique if existed in dictionary

II/ Ways to improve

1. First way:
    - Since suggested words is just taking the first ten that appear in the intersected list of words, we can make it better by creating a list of common words and whenever the common words appear in the intersected list of words, it will be print first in the result.
2. Second way:
    - Since misplacement of letters in words is a very common mistake in spelling, we can make it better by creating a transpose function that detect misplacement in words and correct it. Using this way, we can detect the most likely correct word and place it in front of the list
3. Third way:
    - Since EditNextStep of the previous edit will create repeated elements in the list, we can make it better by creating a function that delete repeated elements if it is edited multiple times
4. Fourth way (implemented):
    - Since our implementation only cover up to the second edit distance from the original word, there maybe some suggestion left out. (even though we can

continue to get words with 3 or more edit distance by using the “editNextStep” function but that may take a lot of time)

- Knowing this, we come up with an idea of just find the Levenshtein distances between a word to all words in the dictionary and take a number (in this situation is 10) of words with lowest Levenshtein distance to the original word.
- Our implementation of this method is in the coding file with 4 functions:
    + levDistance: return the Levenshtein distance between two words.
    + sortTup: sort a list of tuples by the first element in each tuple.
    + bestLevDistance: return the list of words from input list of words with lowest Levenshtein distance from the input word.
    + spellCheckBestLev: the main function to call with input/output files for this method.
- This method has both advantages and disadvantage: for the long words test above we can still find 10 suggestions rather than a few or none using the “spellCheckTwoDistance”. We can also guarantee that 10 words are suggested for each misspelled word. However, time efficiency is a problem with this method when the dictionary file is large.
- We can still improve this by just take the first best 10 with restricted Levenshtein distance (can be 1 or 2) therefore it will not have to go through the whole dictionary.