Yannis Papadopoulos

Layachi Bentabet

Challenge 4 Report

When working on this project the first problem I faced when trying to deal with correcting text was how to search through my dictionary in a time efficient manner. At first I was using a TreeMap because it keeps the data sorted by key but I then realized that I could just use a HashMap to do the same thing in faster time since I do not need the data sorted alphabetically (since a TreeMap has search time of O(log n) and HashMap has a search time of O(1)).

After making sure my dictionary was properly built and the words stored using the HashMap, I started working on the correction part of the project. I decided the best way to identify errors was to first determine if the word existed in the previously built dictionary. The given text for this project had some spelling mistakes which means the dictionary is built using these mistakes. This can/will lead to errors when correcting. When verifying if a word is properly spelt, if the word does not exist within the dictionary, 4 different cases will be tested on the given word. This will allow us to see all possible corrections for the word. I chose that the best way to correct would be to figure out all results and choose the most frequent word out of all possibilities. However, my method can lead to some small errors. For example, let’s say you type a word that is one letter away from two different words. The program will choose the word that occurs most frequently, even if you were referring to the word that appears less frequently.

I started working on missingChar. This function is used to check if the word is possibly missing a character. For example: “aple” -> “apple”. This function will add a character before and after each letter in the given word and will search if the word exists within the dictionary. To do this I have an array which consists of every letter in the alphabet in order. I loop over the word adding each letter one at a time in each position. For example: “helo” first run will check “ahelo” and on the final run will check “heloz”. This will let the program check every possible missing character mistake that could occur. If any of the words with the additional letters are found within the dictionary then the word is added to a HashMap of possible correct words.

Next I worked on wrongChar. This function is used to check if maybe the word has a character that was wrongly typed. For example: “aople” -> “apple”. When working on this function I decided to use the same method I used in missingChar but instead it will replace each character one by one from the list of alphabetical characters. For example: “helo” first run will check “aelo” and on the final run will check “helz”. Then if any of the words with a different character occur in the dictionary they are added to the HashMap of possible words.

I then started working on swapChar, this one was pretty simple and straight forward. This function is used to check if maybe the word has two adjacent characters in the wrong place. For example: “aplpe” -> “apple”. I simply swap every adjacent character one by one until every version has been tested. I then check if the word is valid in the dictionary. For axample: “helo” first run will check “aelo” and on the final run will check “helz”. If the word exists within the dictionary it is then added to the HashMap of possible words.

The final function used to check for spelling errors is extraChar. This function is used to check if the word contains one too many characters. For example: “appple” -> “apple”. This function is also very straight forwards and simple. It removes every character one by one and checks the dictionary every time to see if the word exists. For example: “helo” first run will check “elo” and on the final run will check “hel”. If the word exists within the dictionary it is then added to the HashMap of possible words.

After a word is ran through each of the four functions and has collected a HashMap full of possible words as corrections it checks which words occur most frequently within the dictionary. If two words have the same number of occurrence then it will choose the one that comes first in alphabetical order. If there is no recommended word for correction it will just return the word that was given.