Programming Assignment 4

Trevor Goodwin

C202-Fall 2016

11-11-2016

In programming assignment four we are tasked with creating a spell checking program that compares the words from a text document called “oliver.txt” to another un-alphabetized dictionary file (excludes the definitions for the words) called “random\_dictionary.txt”. In this programming assignment we are supposed to use a data organizer/container commonly known as a “Linked List” to hold data. The “Linked List” is a data container that is populated with values called nodes that start at the root and go out to the right like an array with reference values for each preceding and proceeding value (root has no preceding reference (null)). Think of it like an array without a defined size.

When starting this program we will notice that the “random\_dictionary.txt” file data is not sorted in alphabetical order. There are many ways to accomplish this task, but in this particular assignment we are instructed to build a “Linked List” for each letter of the English alphabet (so 26 trees). After we create (instantiate) all the lists we then must determine the first letter of each word and place it into the appropriate “Linked List”. Once all these lists have been created we can then read our “oliver.txt” document and compare the words contained therein with the “random\_dictionary.txt” alphabetized lists we created previously. Upon looking at the “oliver.txt” file we will notice that there are “special characters (ex. !@#$)” and “White Space”, white space is a spot in our file where no data resides. After removing these before mentioned issues, we can then compare our data line by line with the alphabetic lists we have already created. This can be easily done by using code from the “Lab 6” project as a sort of “search engine”. In order to fully complete this assignment we must track the number of correct words, number of incorrect words (or words that are not contained in our random\_dictionary file), the average number of comparisons that it took to find all the words that ARE contained in the dictionary file, and the average number of comparisons that it took to find all the words that are NOT in the dictionary file.

Upon running this program, I found that it took quite a while (around 28-30 seconds) each runtime to populate the lists then search line by line through our “oliver.txt” file. This is due to the fact that the “contains” method has to search the entire size of the list every time if a word is not found (Worst Case Time Complexity). Without any true alphabetic ordering (only the first character is used as a reference for the indices for each linked list), like “az” could be before “ab” in this list so the placement of that word in the list is completely random. I found that many of the words were correct, but about 65,000 came out as being not correct. There was a staggering 7,000+ average comparison count for the incorrect words which is an unacceptable margin for efficiency standards at least for tackling this particular problem. The correct words average comparison count was around 3500 which is still a lot of comparisons (in upwards of 3 billion) for only about 1,000,000 words.

**Output:**

run:

Number of words found: 914054.0

Number of words NOT found: 64537.0

Average computations for correct words: 3553.083493973004

Average computations for incorrect words: 7428.011621240528

BUILD SUCCESSFUL (total time: 31 seconds)