Programming Assignment 5

Trevor Goodwin

C202-Fall

November 11, 2016

In programming assignment five 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 the previous programming assignment (assignment 4) we used a data organizer/container commonly known as a “Linked List” to hold data. In this assignment however we will have to change it to only work with a container known as a “Binary Search Tree”. The “Binary Search Tree” is a data container that resembles an upside down “V” that is populated with values that are less than the “parent” on the left, and values greater than the “parent” on the right.

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 “Binary Search Tree” for each letter of the English alphabet (so 26 trees). After we create (instantiate) all the trees we then must determine the first letter of each word and place it into the appropriate “Binary Search Tree”. Once all these trees have been created we can then read our “oliver.txt” document and compare the words contained therein with the “random\_dictionary.txt” alphabetized trees 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 trees we have already created. This can be easily done by using code from the “Lab 7” 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 ran much faster than programming assignment 4 while still using the same methods to read and compare the files. It only took around two seconds to fully complete and display the tracking data using the “Binary Search Tree” container. Running the same program with a “Linked List” takes 15 times longer on average (around 28 – 30 seconds every runtime). I also found that there were far less comparisons for correct words using trees, than by using lists (almost 15 million using binary search trees, around 3 billion using linked lists). My results seem plausible due to the fact that the “Binary Search Trees” are organized with values on left being smaller and values on the right being larger. This essentially cut the amount of comparisons in half with each consecutive traversal either to the left or right. The amount of incorrect comparisons was much smaller as well due to the fact mentioned above. The result previously mentioned makes sense, due to the fact that the search cuts off many of the unnecessary nodes each time even when the word is not found. There are also far less words not found than there are found. However, Linked Lists have to be traversed one node at a time till they reach the end.

**Outputs:**

run:

Number of words found: 914054.0

Number of words NOT found: 64537.0

Average computations for correct words: 15.356375006290657

Average computations for incorrect words: 11.490199420487473

BUILD SUCCESSFUL (total time: 2 seconds)