Programming Assignment Five

Nathaniel Sigafoos

4/11/17

In Programming Assignment Five, we were tasked with creating a program that would read through a file and check the spelling of the words within, by searching for them in a dictionary file. We had already developed a solution to this problem, using Linked Lists as the data structure for our dictionary, but this time we were tasked with making it more efficient by using Binary Search Trees as our data structure.

Designing this algorithm was very simple due to the fact that we already had almost all of the solution already implemented. All we had to do was make a few syntax changes to deal with the new data structure.

The reason for implementing this change is due to the efficiency of Binary Search Trees compared to Linked Lists. In order to find an element that is in a Linked List, we must look through approximately *n/2* elements (where *n* is the number of elements in the list). And if the element is not in the list, we must look through all *n* elements. Compare this to Binary Search Trees where, on average, we have to look through *log(n)* elements to determine if an element is in the tree. What this means is that we have to make significantly less comparisons when using Binary Search Trees over Linked Lists, thus drastically decreasing the time the algorithm takes to run. For example, with the included file and dictionary, the Linked Lists version of the algorithm made about 3.7 billion comparisons to check for all the words were in the dictionary, with an average of 3,558.93 to find a word, and 7,393.75 to determine a word was not in the dictionary. Contrast this to the algorithm using Binary Search Trees, which made a total of about 15 million comparisons, with an average of 16.35 comparisons to find a word, and 9.94 to to determine a word was not in the dictionary, using the same file and dictionary. This huge difference is made even more obvious by the difference in the time each algorithm takes to run. The first algorithm takes (on average) 30-50 seconds, whereas the second takes 2-10 seconds. The reason for this huge gap comes down to the fact the Binary Search Trees allow the algorithm to massively cut down the amount of information it must look through to produce the wanted result.