Ryan Goode

Programming Assignment 5

11-27-2017

Problem

The objective of this assignment was to create a linked list of random words to create a dictionary. Then, text file 'Oliver' which contains several thousand lines of text will then be compared word for word to the dictionary. Words being compared and the number of comparisons will be counted. There should be a counter for both the number of words found and not found. The number of comparisons for words found and the number of comparisons for words not found will be counted as well. After the files have been read calculate and report the average number of comparisons for words found and the average number of comparisons for words not found.

Algorithm and Program Design

The design is to instead of a linked list dictionary, a binary search tree will be used using the algorithm from lab7. The binary search tree will have 26 string nodes, one for each letter of the alphabet.

The dictionary() method will take the empty BST as an argument and fills it by reading the random_dictionary.txt file, removes all regex that's not an alphabetical character, puts it to all lower case and finds the ascii number for the first character of that word. The ascii number of 'a' (97) will then be subtracted from the ascii number of that character. This is the index position the word will be stored at in the linked list.

The oliver() method takes the filled BST as an argument and uses a modified search() method from lab7. The modified search() method takes the current word in oliver.txt and an array with just one index to hold the amount of comparisons. The oliver() method creates a count[] array that will be filled by the search() method. oliver() then reads in each line in the file, puts everything in lowercase, removes all non alphabet characters, trims the line and splits the line into individual words and stores them into an array. The method will then iterate over that array, first checking that the original line was not just a white space. The first character's ascii code will then have 97 subtracted from it to find the index position that the word will be compared to in the dictionary linked list, using the search() method. If there is a match the wordsFound variable is increased by one and the amount of comparisons tracked in the search() method is then added to compsFound variable. If there is no match wordsNotFound is increased by one and the count tracker is added to compsNotFound.

The search() method takes an object, which will be the word being compared from oliver() and the single index int array that will hold the count of the comparisons as an argument. The head Node at the current index (first character's ascii number of word being compared – 97) is assigned to 'current' variable and a count variable is declared at 1. The head node is then compared to the current word and if they match the count variable is that added to the count[] array and its only index and the method returns true. If they do not match then the method iterates over all the nodes in the current BST index using the compareTo() method. While the current node is not empty increase the count by one to count the first comparison that will happen. If the result of the first comparison is less than one, the search moves down the left side of the Node. If the result is greater than one increase the comparison count by one, then the search moves down the right side of the Node. If the result is equal to 0 that means there is a

match. The comparison count increases by one to record the second compareTo(), sets the count variable and returns true. If the current node is equal to null, then return false.

Once all methods have been run the average comparisons per words found and comparisons not found can be calculated and reported.

Prog4 and Prog5 comparison

This assignment was essentially a comparison between using link lists and binary search trees. The code was very similar however using a binary search tree compiled the code much faster and the average comparisons were significantly lower.