David Olin

Assignment# 4

Linked List Spell Checker

Due: 11/01/2015 11:59 p.m.

Abstract

The Assignment 4 is an edited version of a previous Assignment to spell check text files. In effect, a more organized, accurate, and faster alternative is being used. The previous Assignment used a binary search method to search the dictionary from the inside out for a specific word in the text file, which is read into the program one letter at a time until an empty value or a punctuation symbol is found. The new linked list method stores all the dictionary words in a collection of linked lists based on the first letter of every word, and uses the first letter of every word read from the text file to determine which dictionary linked list to search. Another features is also included; the user is allowed to determine if they want the letters ‘a’ and ‘i’ added to the beginning or end of the appropriate linked lists. Because linked lists can only be read one piece at a time from beginning to end, this feature simply demonstrates the search time based on position of words in each dictionary list. If the algorithm searches for the words ‘a’ or ‘i’ after they are inserted into the beginning of the list, it has that list’s minimal time complexity and its maximum time complexity if ‘a’ and ‘i’ are added to the end. Time complexity is calculated by dividing the total number of checks for all text file words against the dictionary by the total number of words found or missing. This determines how many times on average the algorithm has to test the text file words in the dictionary.

The program implementation of this algorithm first requests the name of the text file; then if the letters ‘a’ and ‘i’ are to be added to the beginning or end of the appropriate lists or not at all. The dictionary is then read into 26 lists. At this time, the text file is ready to be read and each word tested. The program reads the text file one letter at a time composing words as it goes; then, the dictionary list that corresponds to read word’s first letter is searched for that word.

The program returns different results based on time complexity algorithms which demonstrates that not one method is correct but some are more accurate than others in specific situations. In the case of smaller text files, any of these algorithms would suffice; however, in the presence of larger files, the more to the point, simple, yet accurate algorithms are to best interest.

OUTPUTS:

Unaltered Dictionary:

run:

Enter the name of the File you'd like to check: oliver

add a and i?

to front: FY end:LY :N

number words found = 937527

average number of comparisons for every Correct word = 3252.7655907509866

number of words not found = 54671

average number of comparisons on words not found = 7381.38118929597

BUILD SUCCESSFUL (total time: 1 minute 22 seconds)

‘i’ and ‘a’ at Beginning:

run:

Enter the name of the File you'd like to check: oliver

add a and i?

to front: FY end:LY :FY

number words found = 974370

average number of comparisons for every Correct word = 3129.959637509365

number of words not found = 17828

average number of comparisons on words not found = 7245.597823648194

BUILD SUCCESSFUL (total time: 1 minute 18 seconds)

‘i’ and ‘a’ at End:

run:

Enter the name of the File you'd like to check: oliver

add a and i?

to front: FY end:LY :LY

number words found = 974370

average number of comparisons for every Correct word = 3411.400932910496

number of words not found = 17828

average number of comparisons on words not found = 7245.597823648194

BUILD SUCCESSFUL (total time: 1 minute 34 seconds)