Word Statistics Generator (pa2) Design and Time Complexity Analysis

Anil Tilve

Data Structure Choice: The Binary Search Tree

A binary search tree was chosen as the structure of choice when implementing the word statistics generator because of its intuitive implementation (binary search tree algorithms that are written recursively are often easier to read when compared to their iterative counterparts) and quick performance in many real-time applications.

Analysis of Reading the Dictionary File

For a dictionary file containing m words of length k and n unique words, m insertions are attempted. In the worst case, this results in a skewed binary tree taking up nk space. The time complexity is therefore $O(mk)$ while the space complexity is $O(nk)$.

Analysis of Getting the Statistics

For dictionary and data files containing m words of length k and n unique words, m retrievals from the binary search tree are attempted in order to check for occurrences. Given that the files contain the exact same words, up to m^2 comparisons may be made to check for superwords. This results in a time complexity of $O(m^2k)$.

Analysis of Generating the Output File

For a dictionary file containing n unique words of length k, n traversals are performed in order to generate the output file. Because every node is traversed once, this results in an output file consuming nk space in nk time. The time complexity is therefore $O(nk)$ while the space complexity is $O(nk)$.

Overall Time and Space Complexity

The overall time and space complexity of the program is $O(m^2k)$ and $O(nk)$.

Challenges Encountered

The main challenge encountered when undertaking this assignment was manually allocating memory for strings. Unlike languages like Java which allocate the memory for the programmer, C requires a memory for the string be allocated (either statically or dynamically) before it can be used. Doing otherwise results in a segmentation fault. Another challenge encountered was figuring out which C standard library functions to use for the task. Because C's standard library is not as organized as those of object-oriented languages like Java and C++, finding the appropriate function to get the job done requires a bit of trial and error.