WORDSTAT **Zach Scott**

Data Structures

I chose to store the strings in a binary tree, since it was a fairly elegant structure to implement and can easily print the strings lexicographically with any string as the tree's root/head.

The slight modification I made to a traditional binary tree is the addition of a third dimension of "permutation nodes", as I refer to them. Each node in the tree (can) contain a linked list of case sensitive versions of that string, with the first version found acting as the head. When a string is found to be a case insensitive match to an existing string in the tree, it checks that node's permutation linked list, seeing if that specific version of the word has already been seen. If it hasn't, it appends itself on to the end of the chain. If it has, it simply updates that root of that list's permutation *(int)* field. The size of an individual node or struct tree* as they're known is determined by the length of the array (+1 for null sig) inserted at the current call of ProcessStr(), making each node a different size and conserving memory.

Big O

Normally, an insertion in a binary tree takes **log(n**) time – so inserting n strings from a .txt document would take **nlog(n)** operations**.** However, since each node in this tree also a linked list if it has multiple case-sensitive versions, the time is slightly greater due to needing to go through a node's permutation chain of length (k) to discern whether the current string is a unique permutation. If the text file was simply case-varying versions of the same word, the worst case would be **n^2** since the entire tree would be one linked list at the root. Similarly, if the first word inserted was the earliest lexicographically, and every subsequent word seen was greater than the last, this would also result in **n^2** behavior. However, on average, the case will be **n(log(n)) * k** , k being an unknown factor dependent on the number of permutation linked lists the tree winds up containing.

Challenges

Difficulties during the assignment were primarily getting Segfaults, as my TA informed us were to be expected. Managing pointers was a bit tricky to someone used to the object oriented nature of Java, and figuring out the correct sizes for mallocs was also a bit new to me. I'm somewhat used to Eclipse highlighting blatant errors in syntax and logistics, so having to use gdb to figure out where specifically something went wrong was a bit difficult, but definitely helped me get comfortable with using the Unix command line.