The main data structure used in this program is a prefix tree (in this case a tree with 36 children). Each node only represents a single character and nodes are shared by words that contain the same sequence of leading characters. As an example, CAT and CAR would both consist of the top level C node followed by the C node's A child then it would branch into the A's T and R nodes respectively.

Run time analysis:

Inserting each word requires descending one level down the tree for each character, for an insertion time equal to the word length. Thus, total time to build the tree is O(n * l) where l is the average word length of the file. In most books, magazines, and similar documents l is about 6 or 7.

Reading the tree is a little harder to derive accurately since many words will share characters in most documents. As an upper bound though, it could be assumed that there is no overlap, meaning that traversing the tree causes each character to be read once. This would lead to O(n * l) again.

Total run time: O(n * l) + O(n * l) = O(n * l)

Memory analysis:

Each character node stores two ints (for occurrences and variations), a pointer to the list of variations and a pointer to an array of character node pointers (for the next character(s) in the word(s)). It would likely be more efficient to break the next array into several arrays according to character frequency so that there would be no need to declare all 36 pointers for one follow on character, but I'm not confident in my ability to implement this before the deadline (I used a pointer to the array instead of the array itself so that the bottom level nodes wouldn't need to allocate memory for the whole array). Still, in the worst case no characters overlap and the next array gets declared for all but the last node in a word resulting in worst case O(n*(l-1)*(2int+38point)) total. This means each character takes up ~160 characters worth of memory(on a 32 bit machine), so on an average length word (~7 chars) it takes ~960 chars worth of data (comparable to people who are declaring fixed word lengths of ~1000). This structure is more efficient with large numbers of similar words, because there is more overlap.

The different variations are stored in a linked list where each node only has two pointers, one for the current word and one for the next variation node. The pointer to the current word leads to a node with a 40 character string and a pointer for another string in case the word is longer than 40 characters.

Difficulties:

Interpreting error messages.

Gives segmentation fault when given word greater than about two hundred thousand characters long.