Max Johnson

02/05/2022

Word Locator Design Report

For this project, I decided to implement the data structure for the word locator as a Trie. Trie's are a tree based data structure that is optimal for storing a library of words. Each Trie Node had a list of at most 37 child nodes (for each possible next character). I considered using a BST, but that didn't really seem like it would make sense to me because its hard to distinguish one word from the next without some sort of hash function, and in the end I thought it would be less efficient and effective. Trie's are also used as the main data structure for file system, so I thought I would try my hand at implementing one for this project.

Let's say the character to search for in the data structure has M characters. The best case scenario is that the word you are searching for does not exist, so that would be O(1) and exit immediately (no traversing). Worst case, the word exists, and the time complexity of searching for the word would be the word length, M, so O(M). The worst case is also the best case, O(N), since there are no other possibilities.

Lets say the number of words in the data structure is N. The number of letters in each character, on average, is going to be called M. The size of the alphabet (possible characters to store in each word) is K = 37. So, the memory space of the data structure on average is going to be O(N*M*K);