Assignment 2 - Second

Structure

I used a trie structure, where each node has a character, a prefix count, a prefix buffer int, an occurrence count, 0 if never appeared and -1 if not a word, and two node pointers pointing to the next node or its children node. All nodes linked with next are in alphabetical order.

When reading dict, I read a letter at a time, inserting to the trie, and make occurrence count 0 if word ended. When reading data, if I found the word, I increment its occurrence count by 1. If the pointer has a child, I will increment the child's prefix buffer by 1. After reading everything in data, I recursively traverse the trie and increment their prefix count according to buffer. I used recursion to print trie out in order.

Big O analysis

Max # of words = m

Max word length = k

# of unique words = n

- **Time complexity:**

Inserting: for all words in dict insert m times, each word takes k ∗ 1 times to create the node: **O(mk)**

Matching: for all words in data (m ∗ k nodes) traverse through trie: **O(mk)**

Prefix increment: for all words in data (m ∗ k nodes) traverse through trie: **O(mk)**

Printing: traverse all nodes (m ∗ k nodes), print if is an end of word: **O(mk)**

Total running time: O(mk) ∗ 4 = **O(mk)**

- **Space complexity:**

n words each has at most k nodes; each node is a constant amount of space: **O(nk)**

Comments

I went through second pretty quick, but encountered a problem with efficiency. According to the Professor, the time constraint is 45 seconds, and he tested my code to be about 46.5 seconds. So I created a buffer for every node. Instead of traversing the trie every time, I just increment the buffer and traverse when finished reading data file. With the buffer my code is able to finish outputting around 3.5 seconds.