Project 1 - Design Report

Zihan Zhao (zzhao383@wisc.edu) 4 February 2022

1 Data Structure

Initial Thoughts of Trie Having read the instructions, I knew I would need to build some kind of tree-based dictionary to hold all the words and make them easy to search. Immediately, I thought of one of the data structures I had learned in CS 400 (perhaps), i.e., trie or prefix tree. The predominant advantage of a basic trie over other trees, like BST, AVL, etc., is that it takes up way smaller space in big datasets **comprised of strings** than others due to the reusability of trienodes. Therefore, I decided to use the basic trie as the primary data structure for this assignment.

Optimization of Trie - Radix Tree With the memory profiler provided by Visual Studio 2019, I found that my program's memory usage was fairly large when processing wrnpc.txt. In this file, there were over 560k words, which means that there would be over 560k positive integers in memory to keep track of the word count of each word. Since the largest value that can be stored in a 16-bit unsigned integer is 65,535, I had to use 32-bit (or 4-byte) unsigned integers for files as large as wrnpc.txt. Therefore, my program would take up at least $4N$ bytes of space in memory, where N is the number of words in the file, and there would be no way to reduce this part of memory consumption. However, with a little more digging into my data structure, I found there were unnecessary nodes across the trie. For example, in a basic trie, "apple" and "app" would look like: 'a' $\rightarrow$ 'p' $\rightarrow$ 'p' $\rightarrow$ 'l' $\rightarrow$ 'e'. This was unnecessary. It would be totally fine to reduce this branch to "app" $\rightarrow$ "le", where fewer nodes were used, thus fewer space occupied. Therefore, I optimized my trie in that way, which I later learned that it was called radix tree. This optimization ended up successfully reducing the number of nodes from 50,000+ to 24,000+.

2 Complexity Analysis

Time Complexity of Search The **best case** time complexity for radix tree is $O(L)$ where L is the length of the word. The **average case** time complexity for radix tree is $O(L)$ where L is the length of the word. The **worst case** time complexity for radix tree is $O(WL)$ where W is the number of nodes in each level of the tree and L is the length of the word. However, it is notable that the maximum number of nodes in one level is 37 because valid characters in this assignment only include alphanums and apostrophes. Therefore, W is essentially a small constant.

Space Complexity of Radix Tree The **average case** space complexity for radix tree is $O(N)$ where N is the number of words in a file. However, it is notable that the linear trend starts flatting down when more words are inserted. This is because of the reusability of nodes in the radix tree.