Brian Burton
u1038667
04/25/17
CS 2420
Assignment 12

Huffman Analysis Document

1. Title Page

For assignment twelve, Huffman encoding, I was blessed with the opportunity to work with Lisa Richardson again. We learned quite a bit together about Huffman encoding. Many of the things that we learned dealt with the important properties of building a Huffman tree. Some of those properties include the following.

We learned the hard way that you have to build your Huffman tree from the bottom up. This means that you create the root node last. We also learned that in order to do this, you must use a min priority queue. The way that you choose which element is the smallest is, you see how often each symbol appears in a text file. The symbols that appear the least amount of time are the same symbols that come out of the priority queue first. Another essential point that needs to be made is you always take out the first two elements out of the priority queue.

The final essential detail about building Huffman trees has to do with the nodes of the tree. In a Huffman tree, the leaf nodes never have children. They are always at the bottom of the tree. Also, in our Huffman tree, we decided that when reading the tree, we start at one of the leaf nodes and go up the tree. If the binary number was zero then we knew that the leaf node was to the left of its parent. If the binary number was one then we knew that the leaf node was to the right of its parent. We also created dummy nodes. We did this because we needed to represent a binary number. It would be impossible to represent a full tree without other nodes connecting our nodes. It also allows us to have a more balanced tree.

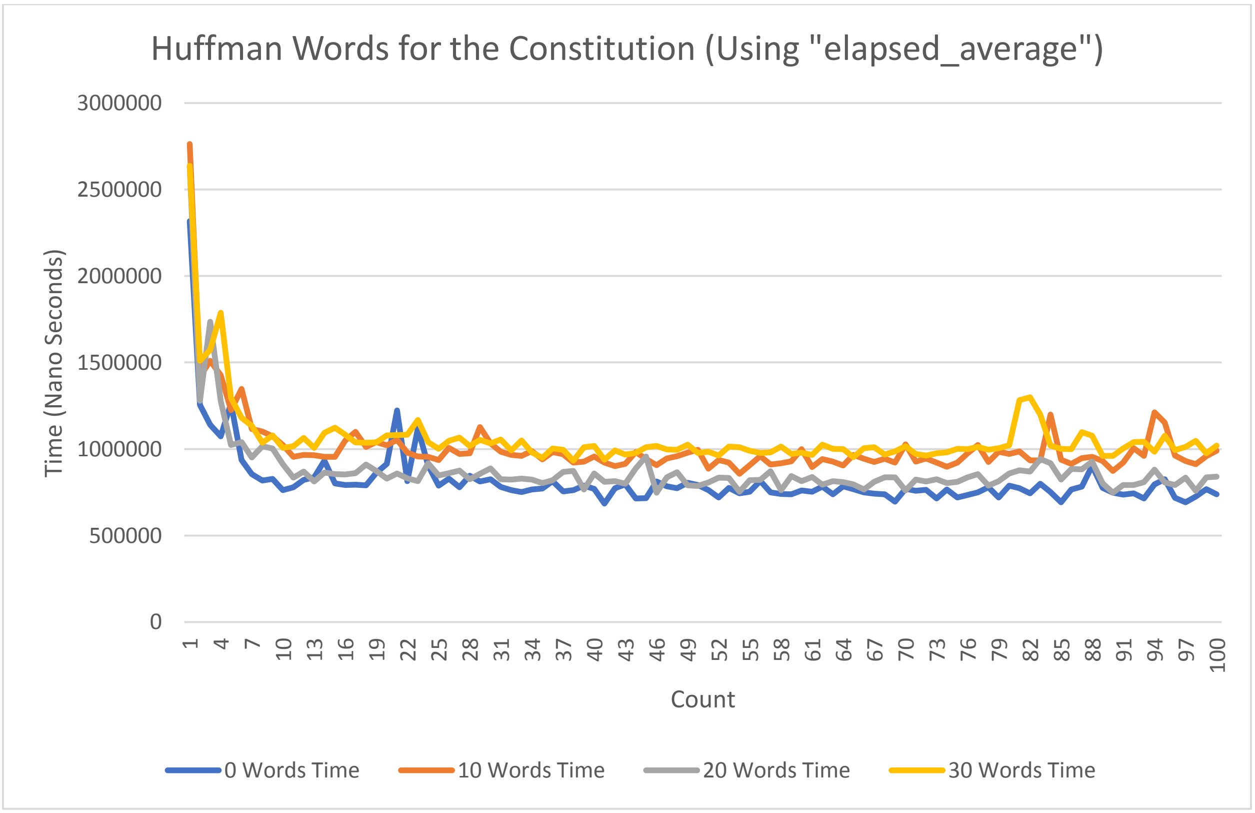
Finally, we never have to delete any elements from the Huffman tree. So that is pretty nice in my opinion. Although if you needed to be able to delete a node for some reason, you certainly could. In short, a Huffman tree must be constant. This means that you must decide whether every left reference represents a one or a zero, and then stick to that. You must also build your tree from the bottom up, connecting the two leaf nodes that come out of the priority queue with a dummy node.

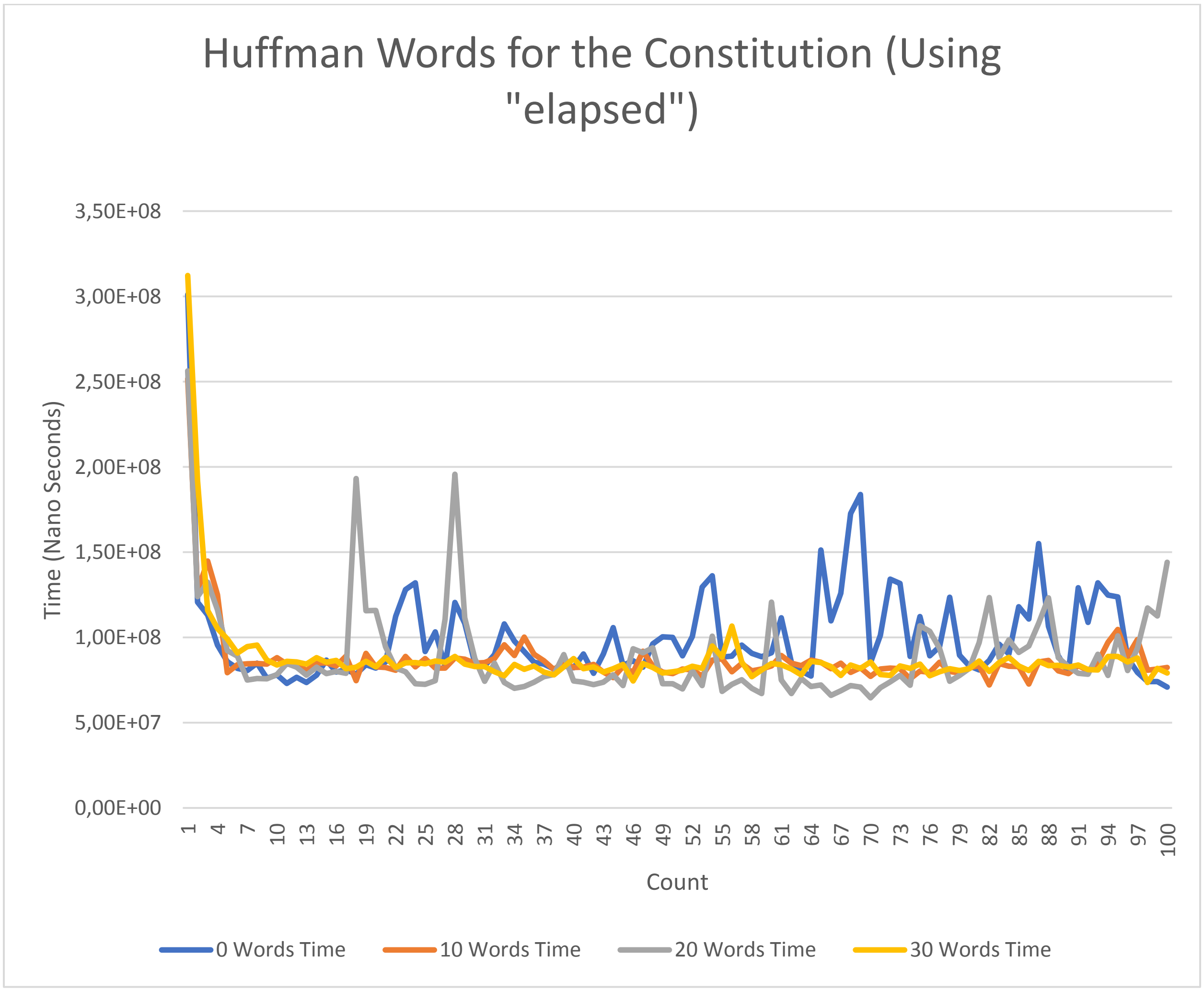
In the real world, at last similar ideas to the Huffman encoding algorithm are used everyday I bet. This is because compressing files is very important to nearly everyone's daily lives. It allows a user to send large files to another user across the world.

2. Timing Experiment

What Lisa and I were attempting to determine empirically was what the optimal number of 'word' symbols were to get the best compression. After running some timing code, we found an odd outcome to our experiment. The more words that we allowed our algorithm to have, the longer that it took to compute. After some discussion, we decided that either our algorithm was incorrect, or our compute the number of words method was taking longer since it was having to compute more words.

The way that we conducted our experiment was we just let our compression algorithm run on the same text file, only changing the number of words allowed in our Huffman encoding class. We timed each one of them individually then plotted the points. Below is the graph that was the outcome.





3. Software Engineering

It took us approximately eighteen hours to complete this assignment. We ran into every issue that you could possibly imagine. Every method that we had to write in the HuffmanTreeUsingWords class caused us grief. They each created their own unique bugs. The best part was that each method affected each other. This meant that once we believed that we had fixed a method, we would move onto another method only to find out while debugging the new method that the old method still had a bug or two.

The thing that I can say that I learned about hash tables is how to use them in other aspects. What I mean by this is I never would have thought to use a hash table to help the computer know which leaf node to start at in order to obtain the bits that represent that symbol. Now that I have learned about Huffman encoding, it make total sense.

I also learned a lot about binary numbers. I never new how to represent different numbers in binary. I learned how to manipulate them using BitSets. I am definitely not a pro by any means but I can now say that I have used them. BitSets help you to represent bytes, or eight bits. In our case, we switched every byte to have the opposite bits.

Finally, I learned more about priority queues. Up unto this point, I thought priority queues could only be used to make decisions, not build a tree. I learned that priority queues can be quite nice when you want things to be done in an exact order every single time.