**// Charles Buyas, cjb8qf, 420(lol)-17, postlab10.pdf**

**My Implementation**

Throughout the writing process, this code gave me quite a bit of trouble. The most difficult part, I believe, was trying to decide what data structures to use and when. When I began my prelab, I had to wrack my brain to figure out the best way to be able to store letters and prefix codes in a way such that they could be accessed by one another (Similar to a dictionary in python with key/value pairs). So, my initial thought was to use an array or vector in order to store codes at the locations specified by the character. How? I decided to type-cast the char as an int, essentially turning it into it's location code. Using that location specified by the ascii value, I stored the prefix code. In that way I had an easy way to access codes, and each time a code was accessed, so was the associated character value given by reversing the type-cast of the index of the prefix code. This worked best for my brain in trying to find out how to keep the knowledge of the letter and prefix. In a similar way I did the same thing but with a different vector (of ints this time instead of prefix strings) to store the frequencies of characters.
Following I had to figure out how to make and traverse a tree. I figured that for making a tree, the easiest structure to use would be a pointer to a node, which I called huffTreeNode. Using this I was able to easily create a tree structure using left and right pointers, as well as storing variables such as c (character) and pref (prefix code) into each node as I went. I did not, however, create a .cpp file for the huffTreeNode class. This I found to be unnecessary, as I didn't do anything in the node class. As the TA's pointed out, though, this ended up being slightly more work for me later on since I didn't have a constructor, forcing me to manually type out the starting values whenever I created a new node. But, I digress.

**Efficiency Analysis**

When I encoded in the prelab, I had a few steps to follow. First, I had to initialize and fill the vectors I made that I would store the character, frequency, and prefix values in. Since both of those were required to iterate through the entirety of the input text, that slowed down my code a bit, but not too bad. For the most part, I believe this just isn't space efficient, and I feel like there's probably a better way to store the variables than in a vector of size 128. Then after that, I had to physically build my tree. The time complexity on this one was more complex (lol) than any other function I was running. Building a tree ran through for all nodes in the heap up to a size of 1. This already uses a method call size(), however the size() method simply returns a variable that we have named heap_size, so this can be done in a constant time. But it runs for size() > 1, so it has a linear run time. Inside the loop, it calls deleteMin(), which is a method in heap.cpp that also calls push_back() (constant) and percolateDown() (logn). So that means we are running at a logn runtime for n number of times. Hence, this has a run time of (worst case) n*log(n). The space being utilized is quite fluid. As we build the tree, the size of it goes up, but we also have the size of the heap that it's coming from is going down.
For each data structure, I had a slightly variable size. My node object was (assuming that prefix strings never reach lengths greater than 8 characters long) 21 to 29 bytes in memory depending on the prefix length. My heap object was surprisingly small, only needed a prefix and a pointer to a node, so that was only a variable 9-16 bytes. My trees were totally depending on the huffman Nodes that built it, but as a minimum were 16 bytes. Finally, I used several vectors, one

of ints and one of prefix strings. The vector of ints was 128 spaces of ints, resulting in 128*4 = 512 bytes; while my vector of strings was 128 spaces of strings from size 1-8, so that vector could have been anywhere from 128 bytes to 1024 bytes in size.

**Citations:**


- http://stackoverflow.com/questions/2477850/how-to-use-string-substr-function
- https://www.tutorialspoint.com/makefile/makefile_dependencies.htm
- https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html