README FOR ASSIGNMENT 2

Philippe Clesca/ Ronnie Jebara

Implementation:

Implementing the codebook builder was probably the longest part. That is all covered in asst2.h and asst2.c. The First thing of note in there is how we built the tree. The leaves in our tree are built in two structs. The first struct, called treeNode, contains a pointer to the leaf on the left and leaf on the right and a pointer to another struct, called fileList. FileList contains a frequency tracker called counter, the file the word was found in, and a pointer to the file that comes next. At first when we built this we tried using only one struct. This wasn't working after a while and was slow when it did work. Somehow splitting this up into two structs and linking them together was faster and worked better. Eventually after going through the directory recursively or just the file it populates the tree with all the words. Then It passes this tree of "double structs" (the term I coined to describe the weird struct structure we built to make this work) to a minheap making function called "createMinHeap". We did this because making the bit representations of the words in the tree was easier this way over using the tree alone. Then from there the program would assign values to the words using 0's and 1's and add these to the codebook. Something to note is that our codebook doesn't print out in a specific order. For example, the word "hi" could have the highest frequency and have a bit representation of '0', but if it's the seventh word encountered it will print out seventh in the codebook. This was something we couldn't fix.

Implementation of compression is also done in asst2.c. How it works is by taking the file name/directory name and find the file/directory. After finding it, it iterates through the file/files and tokenizes them sequentially not based of frequency and generates the new file based off what words appear and using the codebook.

The program segfaults because of an IO error on write even though we are not trying to writ to a file yet. "0x00007ffff7a880ae in _IO_new_file_write" is what gdb states, although we couldn't figure out why.

Retrospective:

I do think that this project was the most difficult of the three. Building the directory parser was not the problem. Huffman coding was where all our troubles really were. But that is neither here nor there. For efficiency we believe our code is very efficient. During the start of the project we restarted it several times because it was not performing fast enough. It was very slow and inefficient. Now with the way we currently have it built, we are getting our fastest times and we are coming up with no ideas on how to make it faster. Looking back the only change I would have made was trying out the double struct idea sooner. That was an idea I had earlier but didn't try until late on in the process and it appeared to be the best and fastest way we had to actually parse through a file/directory.

README FOR ASSIGNMENT 2

Philippe Clesca/ Ronnie Jebara