Corderrius Muse

COMP 4030 (Algorithms)

Final Project Writeup

Introduction

I initially chose this project because I had an interest in file compression. I always wondered what went on under the hood when I used programs such as Winzip, and I figured this project would give me a good chance to learn a thing or two about basic file compression. As a bonus, the focus of the project was to compress media files such as mp4s, jpegs, mp3s, etc so that was an additional source of motivation. Everyone has tons of media files sitting around on their hard drives taking up space, when they'd much rather have that space available for other things. Not to mention, too much clutter could also slow your PC down; therefore, I felt like this project would open up a new window for me as a coder and allow me to make a homebrew compression program that could decrease the size of my media library. However, even though it didn't turn out quite as planned, I still managed to get some functionality out of the app. It may not meet its initial expectations, but it does "something" right.

Initially, I wanted to follow the assignment posted here and use LZW compression on the file to get a bit more bang for my buck when it came to the overall compression. After the LZW compression was finished, the next step would be to send the file through my Huffman Compression algorithm implementation and see if I could reap the benefits of the compression - i.e have a smaller media file that was still accessible (viewable) on the PC. From there, the only thing left would be to send the file through the decompression algorithm I wrote and see if I could indeed get back to the original compressed LZW file I started with. Certain types of

images were supposed to have certain unique characteristics when passed through Huffman Compression. For example, GIF files were supposed to appear virtually unaffected by the compression algorithm and remain the same size because there is already a form of compression done on GIF images. However, when I tried to tackle this implementation, I ran into two major roadblocks: 1) There were little to no resources to follow about using LZW compression in conjunction with Huffman and 2) There were even fewer, practically zero resources online showing you how to implement Huffman Coding on media files.

Algorithm and/or Model

I used C++ and used header files for my Tree, Tree Nodes, and Bitstream. Basically, my algorithm uses a Frequency Table to keep track of all non-zero frequencies and if a frequency isn't already tabbed in the table, it is added. After the loop goes through the entire file, a compressed version is created based on whatever name the user choses. Note: you do not need to name the file in the following manner: **somefile.doc.compressed**. That would be too cumbersome to type, so you can just rename the compressed version of **somefile.doc** to **somefile1.doc** and the program will still work fine - meaning that it will detect the file is still a compressed file for the decompression step. I actually spent most of my time fiddling with running test cases rather than actually spending it all on code. The decompression algorithm seemed to be pretty straight forward after the compression algorithm was completed. The program also will not decompress a non-compressed file. If the file wasn't compressed with the program, it will detect it and throw an error to the terminal window.

Datasets Used

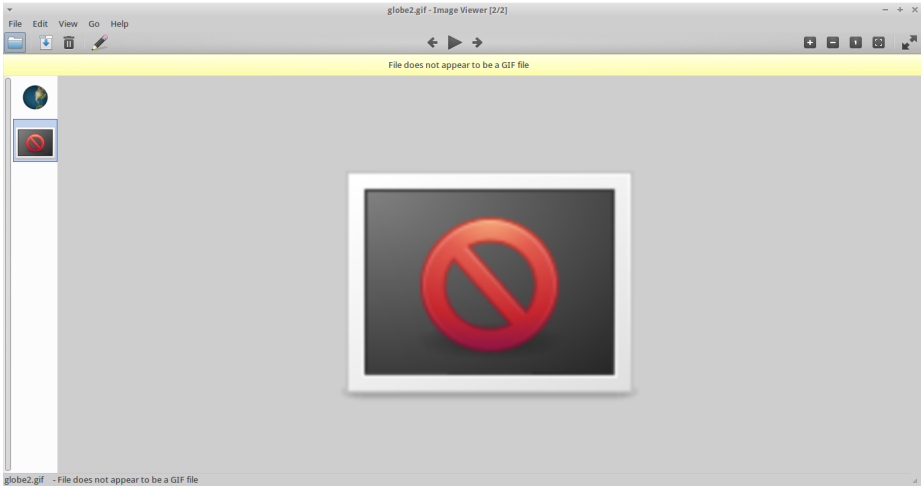
I tested with a various media formats and document formats. I used:

- 1 animated gifs
- 1 jpg
- 1 png
- 1 scalable vector graphic
- 1 bitmap image
- 1 small mp3 from a Korean podcast
- 1 text file
- 1 doc file
- Mp4
- m4v
- mov
- mpeg
- and a few more misc media formats

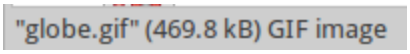
Experiment Setup

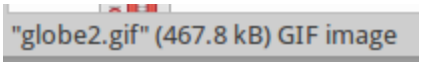
I tested each file one at a time to see if there was any memory saved. I would have liked to see if I could still open the images and miscellaneous media files after the compression took place, but sadly, I wasn't able to. Besides doc files (and a few txt files), no other files could be viewed after compression. However, they could during decompression.

original GIF file alongside its compressed version.



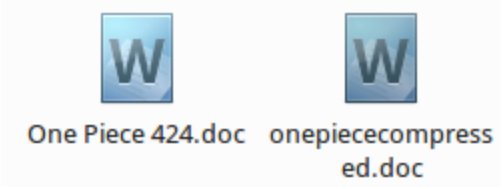
As you can see, I'm not able to actually view the compressed file.

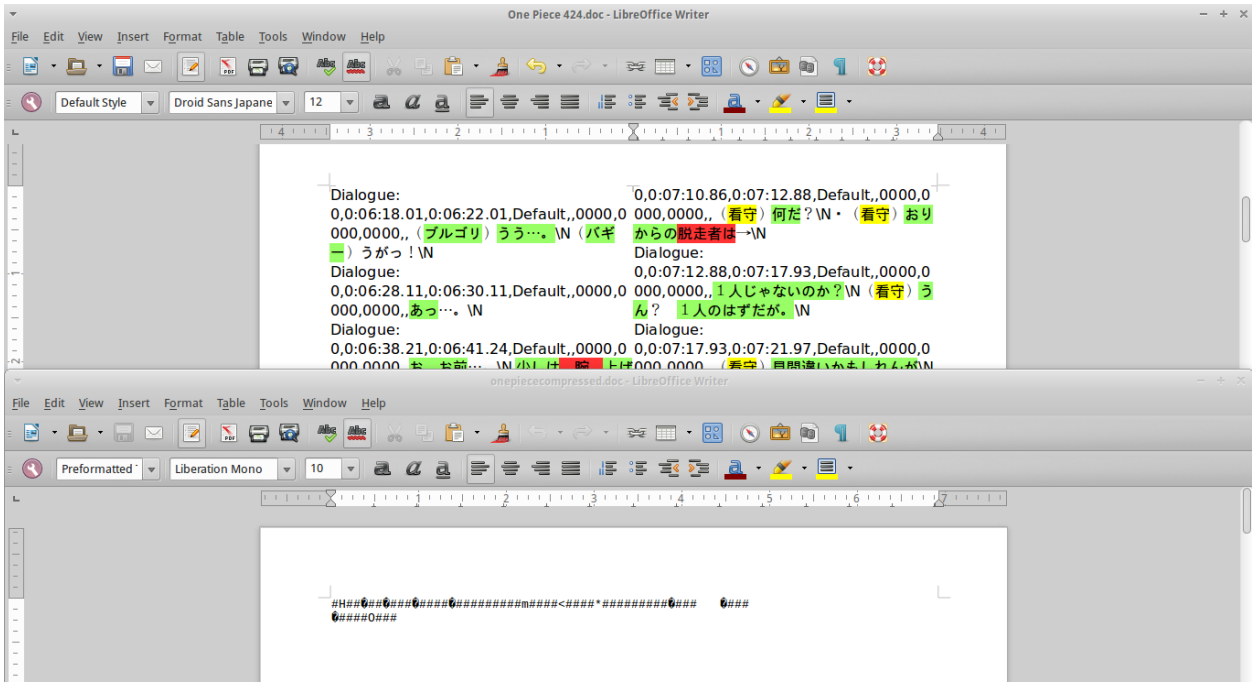


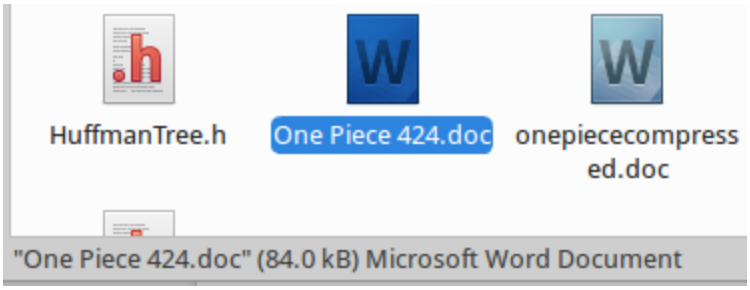


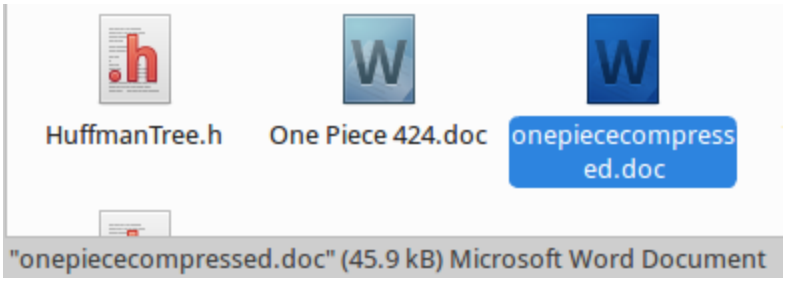
However, in this case, I did manage to save a very minute amount of space.

However, I saw significant compression when I ran this with a word document and I was actually able to open both files.









Results

Overall, the algorithm runs pretty slow depending on the file size. For small files such as these, the algorithm will take only a few moments to run. If I tried this with a 10mb mp3 file, it

would take about a full minute or 2 for the compression to finish, and the file would either be the same size or come out bigger than the original. Here are my results for different types of files (I will also include these in the zip file):

Filetype	Size (Original)	Size (Compressed)	Savings
doc	**84.0kB**	**45.9kB**	**38.1kB**
GIF	469.8kB	467.8kB	2kB
txt	327 bytes	414 bytes	-87 bytes
png	357.7kB	358.7kB	-1 kB
jpg	34.4kB	35.7kB	-1.3kB
bmp	488.2kB	218.6kB	269.6kB
svg	10.0kB	6.0kB	4kB
mp3	4.1MB	4.0MB	.1MB
mp4	245.8kB	245.5kB	.3kB
m4v	2.2MB	2.2MB	0
mov	82.4kB	79.6kB	2.8kB
mpeg	880.0kB	865.9kB	14.1kB
3gp	28.6kB	29.0kB	-.4kB

Overall, besides Microsoft Word documents, it seems like reaping the benefits of Huffman Compression isn't worth the trouble. However, this could solely be due to my implementation. I asked around and one of my friends in the Military let me talk to one of his friends who works in the Computer Science field. He'd gotten contracted a few times to do Huffman Compression on CCTV streams and told me that compressing media files is far more complex, and would require

far more technical calculations. He said that something of that nature is more of an enterprise/corporate level skill and would be pretty technical to implement if you are just starting out. However, this program wasn't a total bust. Even though it didn't compress most files to the level I'd hoped for when I started the project, it did essentially compress a majority of the files I threw at it.