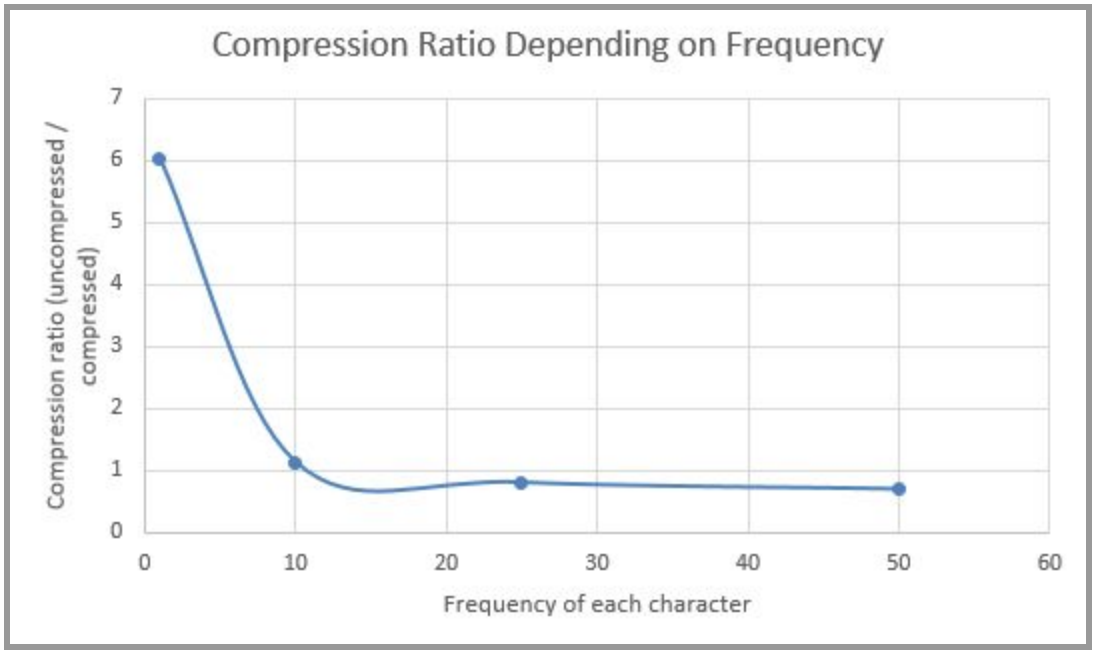
Brayden Carlson
u0959889

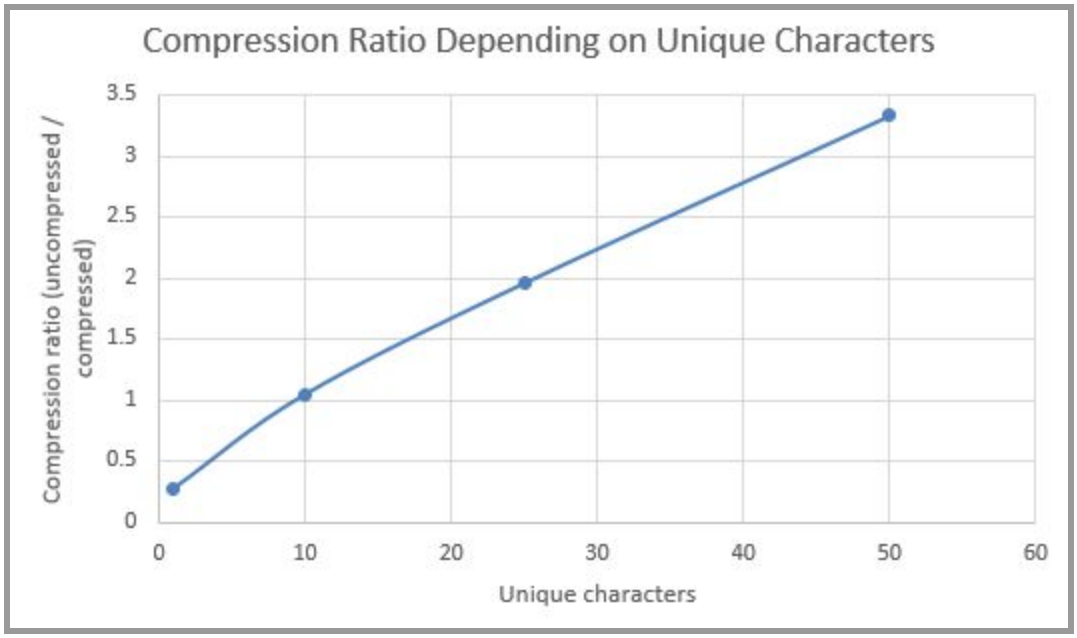
1. Design and conduct an experiment to evaluate the effectiveness of Huffman's algorithm. How is the compression ratio (compressed size / uncompressed size) affected by the number of unique characters in the original file and the frequency of the characters?

Experiment Steps: Create 4 text files to measure the effects of frequency. In one file, paste the alphabet one time (each has a frequency of one). In the next file, paste the alphabet 10 times (frequency of 10), and so forth for 25 and 50. Compress the four files and measure their compression ratio. Plot the results.

Then, create 4 new text files to measure the effects of unique characters. Keep all the files the same size (100 bytes), but modify the number of unique characters. Do this for 1, 10, 25, and 50 unique characters. Compress the four files and measure their compression ratio. Plot the results.



As seen in this graph, the higher the frequency of each character, the lower the compression ratio. This is as expected.



As expected in the graph as well, the more unique characters there are, the higher the compression ratio. This is because the more unique characters we have, the more we have to store in the header, and less we can assign low bit values.

Overall, this experiment went exactly how I imagined. Both frequency and unique characters had a huge impact on the compression ratio. The higher the frequency and the lower the unique characters, the lower the compression ratio.

2. For what input files will using Huffman's algorithm result in a significantly reduced number of bits in the compressed file? For what input files can you expect little or no savings?

You can expect significantly reduction in the number of bits with files that have a low number of unique characters and files that have a large frequency. An example of this would be

You can expect to have little or no savings in files that have a high number of unique characters and a small frequency of each.

3. Why does Huffman's algorithm repeatedly merge the two smallest-weight trees, rather than the two largest-weight trees?

It does this so that the smallest weight/frequency nodes are at the bottom of tree. The characters with the highest frequency need to be further up the tree so that they get assigned smaller bit values. If it repeatedly merged the two largest-weighted trees, the opposite would happen. The highest frequency characters would get assigned larger bit values and the file would do the opposite of compress.

4. Does Huffman's algorithm perform lossless or lossy data compression? Explain your answer. (A quick google search can define the difference between lossless and lossy compression).

Huffman’s algorithm performs lossless data compression. This is because no data is lost when the file is uncompressed. The original file and the decompressed file will always be identical.

5. How many hours did you spend on this assignment?

5 hours.