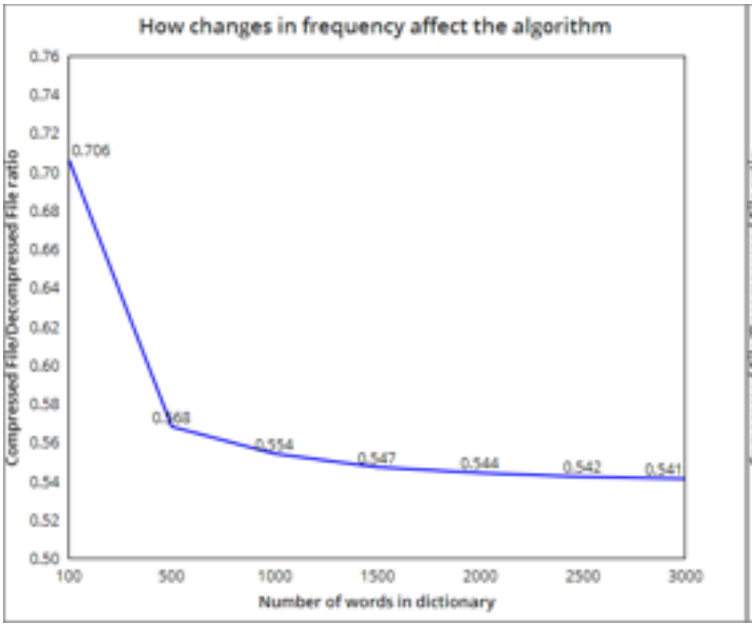

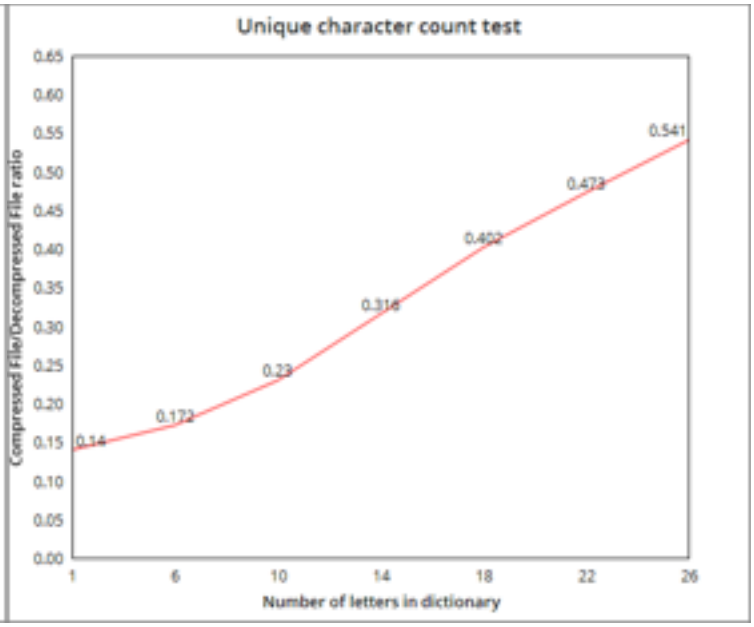

1 Design and conduct an experiment to evaluate the effectiveness of Huffman's algorithm. How is the compression ratio (compressed size / uncompressed size) affected by the number of unique characters in the original file and the frequency of the characters?

These tests were rather complicated, especially the unique character count, so I will explain them separately

For the frequency test, I used a dictionary file which we were given in a previous assignment, and ran the compression on that file at a various number of words in the file. I then looked at the size of the original dictionary file and compared it to the size of the compressed file in bytes. The graph shows those comparisons, and trends in the way you would expect, though the trend is fairly small at larger numbers. The larger the dictionary, the smaller the compressed file was when compared to the dictionary, as would be expected with Huffman's algorithm.

For unique characters, I used the same dictionary file at its full length, approximately 3000 words. I measured the compression ratio of the file while slowly taking out letters and replacing them all with the letter "e", chosen because it is the most common language in the dictionary. I took the letters out and tested the ratio in the following order:

1: a, q, h, c
2: b, x, i, l
3: r, t, y, z
4: o, n, d, f
5: s, j, k, u
6: p, m, g, w, v

The letters were taken out in that order in an attempt to remove approximately the same amount of letters every time according to their frequency in the dictionary. The results of the test were exactly as expected and quite linear, as the fewer number of letters there were in the dictionary the better the compression ratio was. This is precisely as you would expect with Huffmans algorithm.

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?

For large files or files with few unique characters, Huffman's algorithm will significantly reduce the number of bits, as the previous test showed. For extremely small files or files with many unique characters, Huffman's algorithm could save very little or even make the file size larger.

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

Because you want to keep the items with a large weight near the top of the tree so that they are easier to access, since you will have to access them more often when

4. Does Huffman's algorithm perform lossless or lossy data compression? Explain your answer.

Lossless, since absolutely no data is lost when the compressed file is decompressed. If, for example, when decompressed the new file was missing every 10th letter, that would be an example of lossy compression. But, our algorithm doesn't do that, instead accounting for all of the data in a file.

5. How many hours

I spent around 10 hours on this assignment