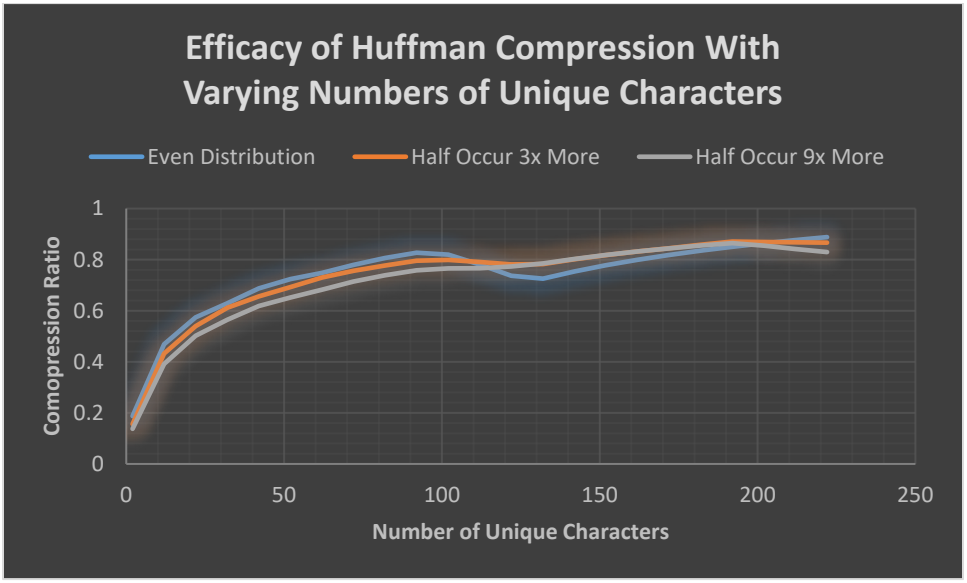
Dylan Johnson
u1024233

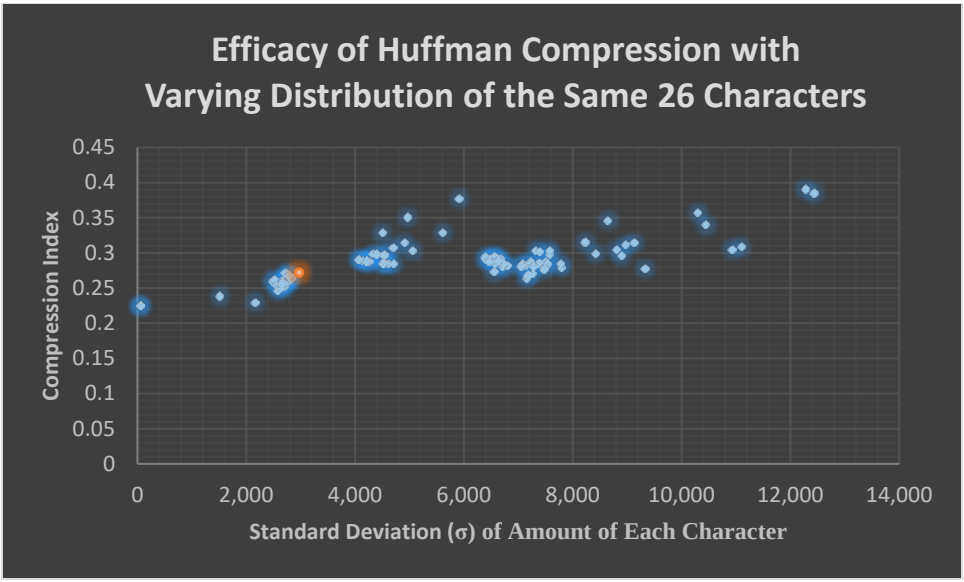
1. Design and conduct an experiment to evaluate the effectiveness of Huffman's algorithm. How is the compression ratio (compressed size / uncompressed size) affected by the number of unique characters in the original file and the frequency of the characters?

In order to evaluate how Huffman Compression depends on the number of unique characters, I created a simple experiment which relates the two. For various amounts of unique characters – ranging from 2 to 222 – I generated a file of millions of such characters, with three distinct frequencies. Although these frequencies are generic and poorly measured, they serve to demonstrate the consistency of the data's trends:



As the data shows, the more unique characters a document contains, the less effective Huffman Compression becomes. This is because more characters results in a taller Huffman tree, and thus longer compression codes.

In order to evaluate how Huffman Compression depends on the frequency of characters, I performed an almost identical experiment to the above, except I varied frequency rather than the number of unique characters. Thus, I used a set of 26 characters – arbitrarily selected as the lower-case English alphabet – and varied the frequencies of these characters, recording the compression index for various frequencies. As part of the experiment, I needed a method to measure the relative variation in frequency, and so I calculated the standard deviation of the count of each character. This value acts as the independent variable for the experiment, and reflects the variations in the weight of the nodes (i.e. the leftmost point is when every letter occurs almost the exact same number of times, whereas the rightmost point is when there exist massive variations in which characters appear at what counts):



The data follows no distinct pattern, excepting a vague upward trend. Indeed, neither frequency variations in either graph provide noticeable differences in compression indices. This lack of structure seems most pronounced at the extremes of the number of unique characters – few unique characters or a lot of unique characters – but more testing similar to that in the second experiment would be necessary to draw more complete conclusions. The orange point above represents the frequency of letters found in the English language, as a decent representation of Huffman Compression's efficacy with a document of just English words.

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?

Files with a lot of repeat characters, especially those which consist of almost entirely the same character, are significantly reduced with Huffman Compression. This results from Huffman Compression's use of a binary tree. In documents with few – for example, less than 50 – evenly distributed, unique characters, the height of the tree is rarely greater than 4 or 5 and so eight bit characters all reduce to at most 5 bits. Conversely, more characters necessitate a taller tree, yielding longer codes and therefore less compression.

Similarly, even if there are a lot of unique characters, large variations in frequency will also increase the efficacy of Huffman Compression. This results from Huffman Compression's use of frequency as the determining factor in the length of compressed characters. Specifically, wide variation in frequency allows the tree to be incomplete, resulting in 1 or 2 bit codes for common characters, and longer codes for less common characters. In this way, Huffman compression takes advantage of differing frequencies, by saving the most on the most common characters, thereby maximizing compression of the majority of the document. As a point of clarification, this only holds true in extremely different cases. That is, it occurs in widely different scenarios as in the first graph, but by no means holds true in the much more holistic, particulate data found in the second graph.

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

If Huffman’s algorithm were to always merge the largest-weight trees, then the most frequent characters would percolate to the bottom of the tree as more and more merges occur. In this situation, the most recurrent characters would have the longest codes, and would therefore contribute to a larger compression ratio than could otherwise be achieved. By merging the smallest-weight trees, the Huffman algorithm performs the opposite: It forces the most common characters to be merged last, thereby placing them at the top of the tree with the smallest codes. Smaller codes, of course, yield better compression.

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

Huffman’s algorithm performs lossless data compression, because it encapsulates the entirety of the original document, but simply reduces – in most cases – the bits required to represent the document. Because the compressed document’s header stores the compressed code for each of the original character, undoing the compression yields a document identical to that of the original. In other words, because no precision is sacrificed in determining the compressed codes for each character, no data is lost.

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

Approximately 1 hour on the coding itself, 2 hours on JUnit testing, 2 hours on timing experiments, and an additional 1 hour on this document.