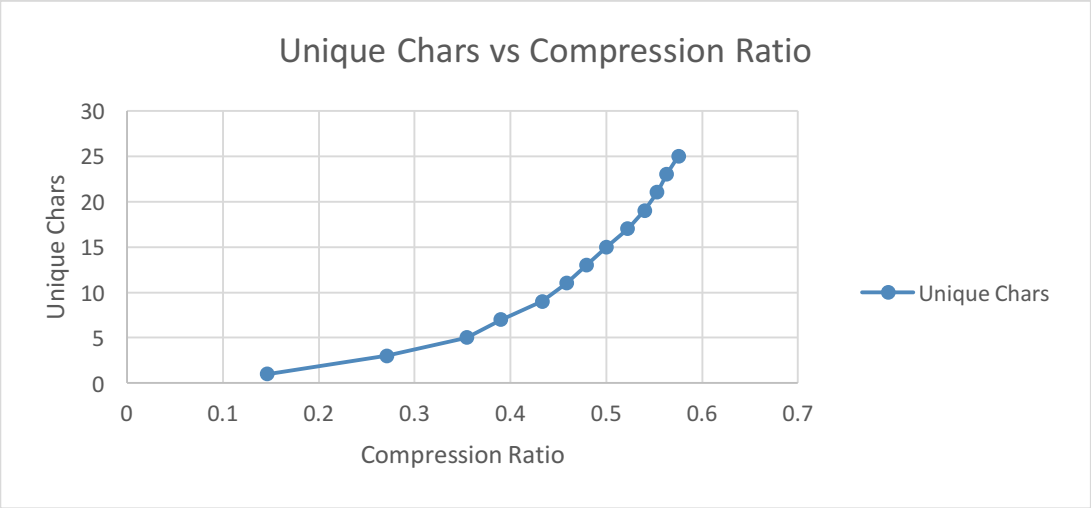
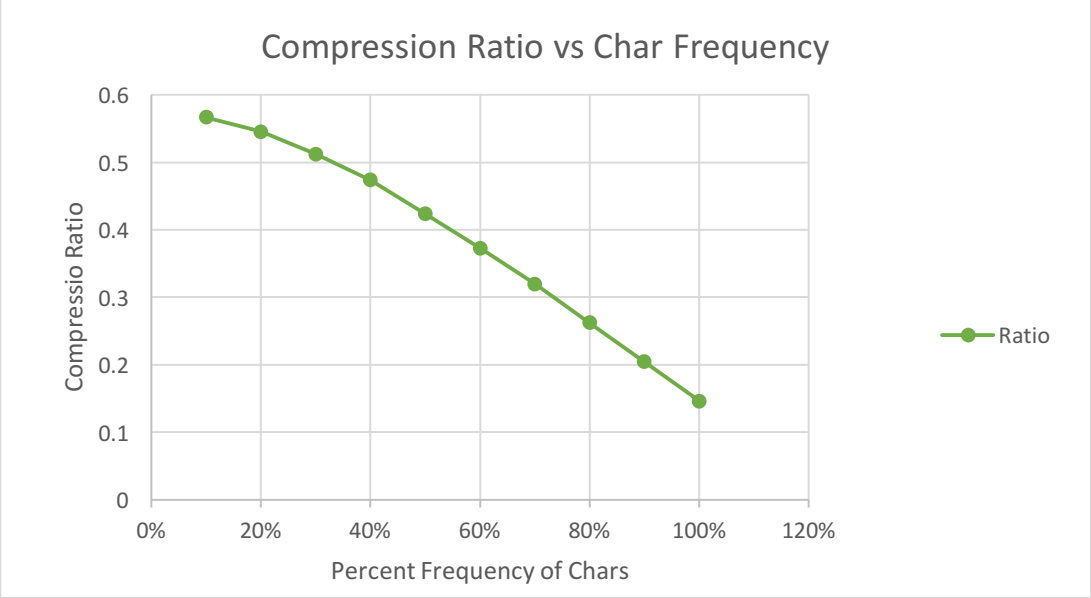
EDUARDO ORTIZ
U0922628

When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 25% of your assignment grade. Ensure that your analysis document addresses the following.

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? Carefully describe your experiment, so that anyone reading this document could replicate your results. Submit any code required to conduct your experiment with the rest of your program and make sure that the code is well-commented.





Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as, your interpretation of the plots is critical.

I began my analysis initially by deciding to measure unique character number in a list whilst comparing that to the final compression ratio attained. I did this by creating a set size file in the compression demo method. This was done by creating a random text file generator. From here I determined the max unique character value for my tree to be the alphabet or a – z (char values 97-122).I then decided to limit all resulting words to be of length 5. I would then record the uncompressed file size before and then after for each permutation of my set which incrementally added the number of unique characters or letters in the alphabet by 2 each time while keeping sure to maintain the same size of file and frequency for all chars throughout. From here I simply calculated the compression ratio by dividing the compressed size by the initial recorded size and then graphing it to note any trends. As noted in the first graph above as the number of unique characters (i.e letters) increased the compression ratio of the files seemed to increase. This upward trend is expected as Huffman's algorithm is based on a binary tree which continuously merges its two smallest trees. As a result of the increasing number of characters the tree will become very unbalanced and as a consequence efficiency will be reduced. In other words, all leaves in the tree will be equally as hard will become increasingly hard to reach because there will be more elements to traverse through and as a result eliminating the benefits given to us by using a binary tree.

The second test I did was fairly similar to the first in that I utilized the compressionDemo class to initialize testing. I began by setting up a test to maintain the same word size in the file throughout as well as word length and to maintain a constant flow equally probable 5 letter sequences to be produced. From here however, I altered the chance of a single character in the test. I removed the letter a from the list and decided to control its frequency within the file. This was accomplished by randomly selecting a value from 0-100. If that value was greater than the percent pre determined it would print from the random char list if it was not, it would print the letter a. As a consequence, A's frequency in the list was controlled to be able to better see the results. From here it was just a matter of simple recording and graphing to show the results. As we ca see above as the chars frequency increased in the the compression ratio decreased as well thus indicating improved performance. This increased performance was expected because Huffman's algorithm works well as all lesser frequency values remain towards the bottom half and all higher frequency are at the top. As the frequency of a became so common the tree simply had to use very little bits to get its compressed value each time as it was always at the top the more frequent it became.

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?

We can expect a minimal number of bits for files which have very high frequently encountered characters. AS seen above as the frequency of the char within he file increases the less amount of work the program must do to find it in the tree and as a result the compressed size was much smaller! Little to no savings can be encountered when the file size contains a very large number of characters. This is simply because as the number of characters increases the overall frequency of each char decreases thus also increasing

the overall efficiency of the tree because less frequent values will be at the top and bottom thus making the tree structure irrelevant.

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

Huffman's algorithm seeks to encode the the most frequent characters with less bits than all others as a result. They are to be located towards the top, while uncommon characters which a much lesser frequency will be encountered towards the bottom. By merging the smallest weighted trees first we ensure that the least common elements remain at the bottom because as larger trees are added on the most frequency of values in them remain towards the top of the tree.

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). Lossless compression refers to the fact that compression done in this way do not lose any data when being compressed, and as such they are limited more in how much they can compress a file when compared to lossy compression. Lossy Compression refers to compression which ignores unnecessary our unneeded information. Huffman's algorithm is LossLess. This is because nothing is lost when compressed everything is simply converted to variably sized bits depending on their frequency in the code.
5. How many hours did you spend on this assignment?

10 Hours testing took more than expected :/

Upload your solution (.pdf only) on the assignment page by 11:59pm on Nov 23.
DATA FOR THE GRAPHS:

Column1	Column 2	Column3	Column4	Column 5	Column 6	Column7
Beginning testing varying frequency on compression ratio						
Frequency of Char is	10%	Compression sizes are:	300000	169979	ratio is:	0.566596667
Frequency of Char is	20%	Compression sizes are:	300000	163538	ratio is:	0.545126667
Frequency of Char is	30%	Compression sizes are:	300000	153696	ratio is:	0.51232
Frequency of Char is	40%	Compression sizes are:	300000	142133	ratio is:	0.473776667
Frequency of Char is	50%	Compression sizes are:	300000	127023	ratio is:	0.42341
Frequency of Char is	60%	Compression sizes are:	300000	111810	ratio is:	0.3727

Frequency of Char is	70%	Compression sizes are:	300000	95880	ratio is:	0.3196
Frequency of Char is	80%	Compression sizes are:	300000	78653	ratio is:	0.262176667
Frequency of Char is	90%	Compression sizes are:	300000	61244	ratio is:	0.204146667
Frequency of Char is	100%	Compression sizes are:	300000	43771	ratio is:	0.145903333
done with testing varying char size on byte Size						
Beginning testing varying unique characters on byte Size for constant char						
Unique Characters =	1	Compresion sizes :	300000	43771	ratio is:	0.145903333
Unique Characters =	3	Compresion sizes :	300000	81281	ratio is:	0.270936667
Unique Characters =	5	Compresion sizes :	300000	106291	ratio is:	0.354303333
Unique Characters =	7	Compresion sizes :	300000	117015	ratio is:	0.39005
Unique Characters =	9	Compresion sizes :	300000	129922	ratio is:	0.433073333
Unique Characters =	11	Compresion sizes :	300000	137571	ratio is:	0.45857
Unique Characters =	13	Compresion sizes :	300000	143831	ratio is:	0.479436667
Unique Characters =	15	Compresion sizes :	300000	150091	ratio is:	0.500303333
Unique Characters =	17	Compresion sizes :	300000	156719	ratio is:	0.522396667
Unique Characters =	19	Compresion sizes :	300000	161953	ratio is:	0.539843333
Unique Characters =	21	Compresion sizes :	300000	165895	ratio is:	0.552983333
Unique Characters =	23	Compresion sizes :	300000	168881	ratio is:	0.562936667
Unique Characters =	25	Compresion sizes :	300000	172641	ratio is:	0.57547