Comp20290 Algorithms

Assignment 2 - Huffman Compression

Ethan Hammond - 19314353

Task 1:

See Task1-HuffmanByHand.png

Task 2:

See Code folder

Task 3:

Q1.

File Name:	Original Bits:	Compressed Bits:	Compression Ratio:	Compression Time:	Decompression Time:
genomeVirus.txt	50008	12576	25.15%	0.03 s	0.007 s
medTale.txt	45056	23912	53.07%	0.015 s	0.005 s
mobyDick.txt	9531704	5341208	56.04%	0.162 s	0.084 s
alice_in_wonderland.txt	1188592	677552	57%	0.05 s	0.038 s
q32x48.bin	1536	816	53.13%	0.01 s	0.001 s

The time complexity of the Huffman algorithm is O(nlogn) thus running time is N (Input size) and R (ASCII alphabet size (256)).

The genomeVirus.txt file showed the best compression ratio of about 25.15%, 30.22% better than the other files average of 55.37%. This is because the text contains genomic code which is only 4 characters. This leads to a balanced Huffman tree and each character is compressed from 8-bit ASCII to 2-bit binary code.

The other three text files are large text files, ranging from less than half a million to over 9 million bits, with many characters. Each took varying amounts of time to compress but all ending up with a compression ratio of just over 50%, 55.37% on average. This compression ratio of just about half is very good.

Q32x48.bin is a bitmap that the Huffman ratio managed to compress by almost half 53.13%

Q2.

File Name:	Original Bits:	Decompressed Bits:	Decompression Time:
outputVirus.txt	12576	50008	0.007 s
outputTale.txt	23912	45056	0.005 s
outputMoby.txt	5341208	9531704	0.084 s
outputAlice.txt	677552	1188592	0.038 s
output.bin	816	1536	0.001 s

Huffman decompression is much faster than compression since the Huffman tree has already been created and the algorithm only needs to traverse the Huffman trie and expand the compressed bits. The decompression process successfully recreates the compressed files, the decompressed bits matching the original bits in my first table. The size of decompression is shown to increase with bit size. E.g. Outputmoby.txt has 5341208 bits and takes 0.084s whereas outputTale.txt has only 23912 bits and only took 0.005 s

Q3.

File Name:	Original Bits:	Compressed Bits:	Ratio:	Time Taken:
outputTale.txt	23912	25960	108.56%	0.029 s
outputVirus.txt	12576	14896	118.45%	0.033 s
output.bin	816	1272	155.88%	0.084 s

The above files were compressed twice, once again the time to compress increased alongside the bit size, however this time, compression ratio also increased to over 100% instead of a ratio of around 55% like previously. This is because the compression algorithm reads the 8-bit ASCII characters in the compressed files. However because the characters are meaningless, the symbols differ significantly and at various frequencies. This increases the the amount of nodes on the Huffman tree, since different symbols occur at smaller frequencies. Thus the file size increases after a second compression.

Q4.

Algorithm:	**File Name:**	**Original Bits:**	**Compressed Bits:**	**Ratio:**	**Time Taken:**
Huffman	q32x48.bin	1536	816	53.13%	0.012 s
RunLength	q32x48.bin	1536	1144	74.48%	N/A

Huffman has a better compression ratio than RunLength, beating it by 21.35%. This is because the runs of 0 bits and 1 bits are all read as 8-bit ASCII characters in the Huffman algorithm. This provides fewer different symbols, thus there are less nodes in the Huffman trie reducing its size and the size of the compressed file. Bitmaps are better compressed by the Huffman algorithm than run length encoding since Huffman uses 8-bit characters that occur frequently which are shorter than the long runs of bits that run length encoding uses and needs to occur consecutively and frequently to be efficient.