Algorithms COMP20290
Huffman Compression Algorithm

Group Members

- Adam Leacy – 18313471
- Daniel Gallagher – 18401492
- Jeff Ryan – 18381323

Task 1

[redacted]

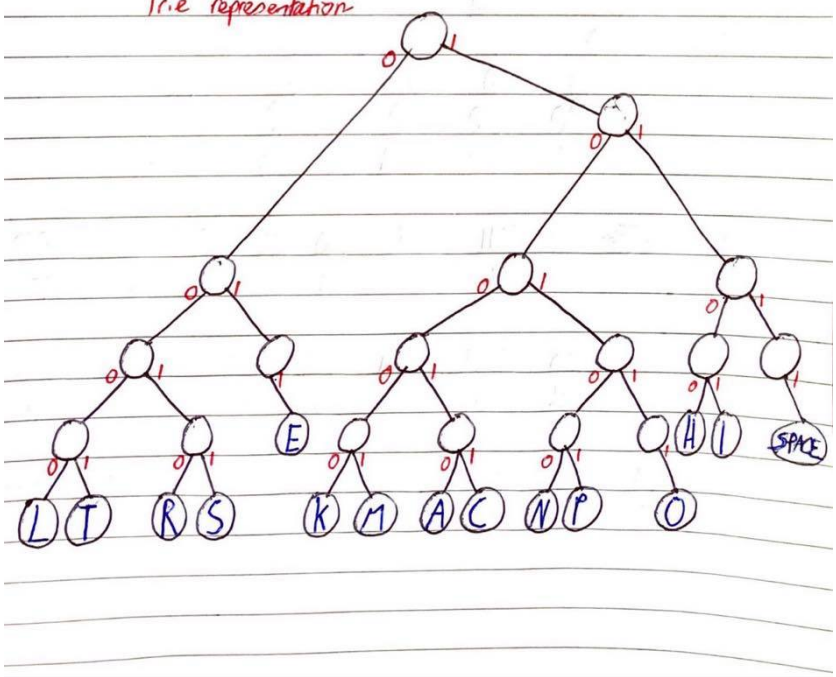
Frequency Table

T - 1	I - 2	P - 1	k - 1
H - 2	S - 1	L - 2	M - 1
E - 5	N - 1	A - 1	Space - 5
R - 1	O - 2	C - 1	

Codeword Table

T - 0001	I - 1101	P - 01010	k - 10000
H - 1100	S - 0011	L - 0000	M - 10001
E - 011	N - 10100	A - 10010	Space - 1111
R - 0010	O - 10111	C - 10011	

Trie representation



[redacted]

Task 3

Files used

- medTale.txt
- genomeVirus.txt
- mobydick.txt
- custom.txt

Compression Analysis:

Time it took to compress each file

File	Huffman Compression Algorithm (in nanoseconds)
medTale.txt	96714245.00
genomeVirus.txt	75322473.00
mobydick.txt	513077223152.00
custom.txt	15275278.00

Compression ratios of each file

Bits before compression	Bits after compression	Ratio
45024	22864	50.78%
50008	12502	25%
9531696	5179339	54.33%
192	92	47.91%

Decompression Analysis:

Time it took to decompress each file

File	Huffman Compression Algorithm (in nanoseconds)
medTaleCompressed.txt	152668512.00
genomeVirusCompressed.txt	157800193.00
mobydickCompressed.txt	707959387528.00
customCompressed.txt	30817269.00

Number of bits in each file after decompressing

Bits before decompression	Bits after decompression
22864	44240
12502	50008
5179339	9354440
92	192

We can see that the Huffman compression algorithm, for a majority of the text files passed into it, halved the number of bits in each file.

To test what happens when you compress an already compressed file, I used 'genomeVirusCompressed.txt'. Before re-compression, the number of bits in the file was 22864 and then after I tried to compress it again, the number of bits remained unchanged. This happened because a good compression algorithm is supposed to be able to remove redundancy's from a file in order to compress it. Thus, when we try to compress an already compressed file, it shouldn't change the number of bits or size as all redundancy's should already be removed by the first compression.

Comparison between RunLength and Huffman using 'q32x48.bin'

Before compression the binary file contained 1536 bits.

After compressing 'q32x48.bin' using RunLength, I found the number of compressed bits to be 1144..
Compression ratio: 74.47%

After compressing 'q32x48.bin' using our Huffman algorithm, the number of bits within the file was found to be 581.
Compression ratio: 37.82%

We find that the Huffman algorithm compressed the binary file much better than the RunLengthEncoding algorithm. This is possibly because Huffman is designed to remove all redundant letters from a file while RunLength does this to a lesser extent. RunLength will only remove characters that succeed each other consecutively. This means RunLength can only replace repeated chars beside each other with a single number stating how many times that character was repeated in that word. Whereas Huffman will analyse how frequent a letter is throughout the entire text and then using a frequency-sorted binary tree, encodes the text.