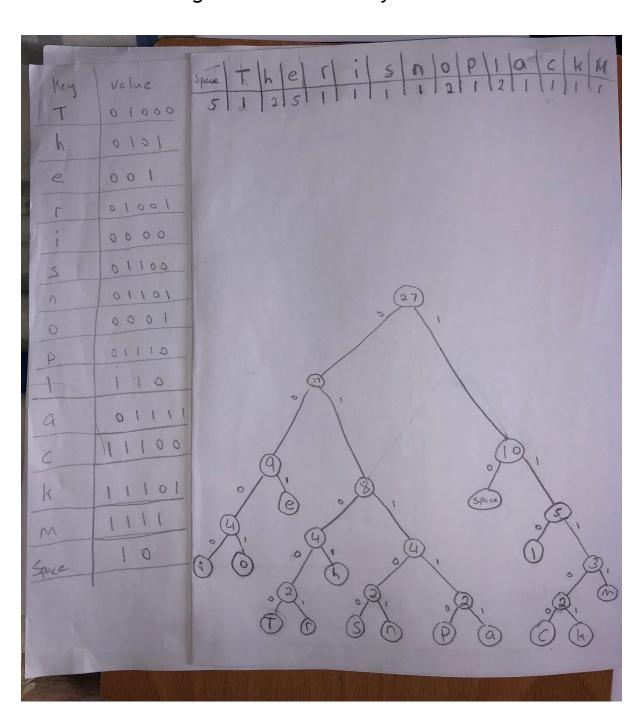
COMP20290 Algorithms

Assignment 1

Morgan Collins McCarthy 18761429



First, I tested to see if my program was running correctly.

Here I get the original bits of the test file.

I then compress the file and get 88 bits.

Next, I compress the file and the decompress it and we can see I get the original 128 bits.

C:\Users\Morgan\Workspace\Huffman\src>java HuffmanAlgorithm c < test.txt | java HuffmanAlgorithm TESTTESTTEST

Finally, I compress the file and decompress it to see if I get the original text which is correct.

File ▼	Compression Time (nano) 🔻	Compressed Size 🔻	Original Size 🔻	Ratio ▼	Decompress Time ▼
genomeVirus.txt	17684200	12576	50008	0.2514798	8795400
medTale.txt	23151500	23912	45056	0.5307173	13215600
mobydick.txt	3080514500	5341208	9531704	0.5603623	2264261100
q32x48.bin	5347500	816	1536	0.53125	5211900

I then when on to compress each of the files provided to a new compressed txt file and recorded the time in nano seconds it took to compress. I recorded the original file sizes and the compressed sizes. I then decompressed each file and recorded the time it took. I also made sure the decompressed bits matched the original bits.

C:\Users\Morgan\Workspace\Huffman\src>java HuffmanAlgorithm c < mobydickCompressed.txt > d.txt

I then tried to compress a compressed file, this resulted in no change in bits, this is because the Huffman algorithm is being applied to a file that has already been compressed using Huffman compression resulting in no change. Using run length encoding on q32x48.bin we get a result of:

1144/1536 = 0.74479166666

Huffman we get a result of:

816/1536 = 0.53125

We see Huffman compression give us a better compression ratio in this case. This is because RLE works better when there is a lot of repeated data. RLE would compress better than Huffman if there were more repeated bits in the bit map.