University College Dublin

Huffman Project

Allyssa Ark

19206924

COMP20290

Dr. Mark Matthews

05/03/2020

Phrase: "There is no place like home"

Letter Frequency code

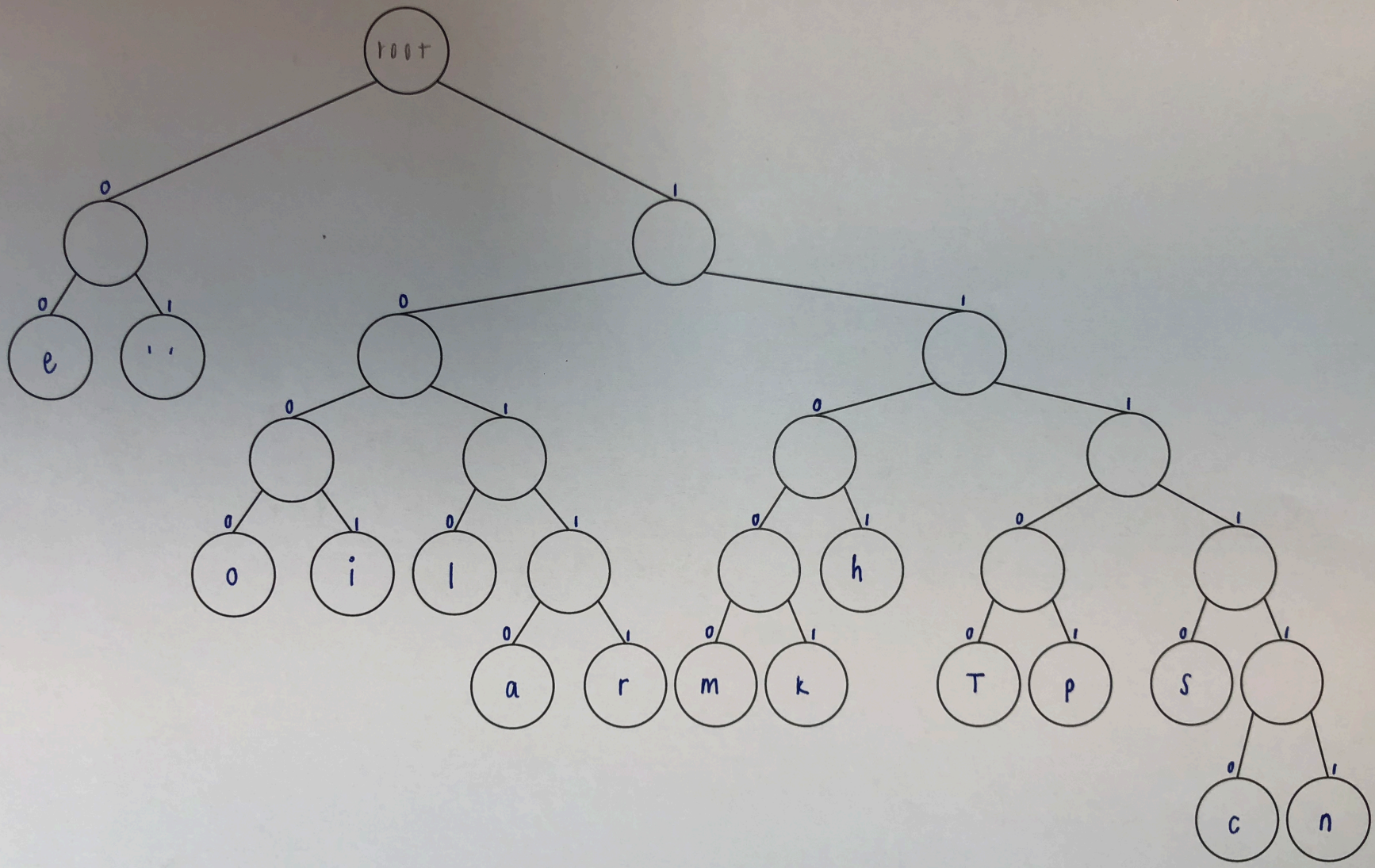
Letter	T	h	e	r	i	s	n	o	p	l	a	c	k	m	' '
Frequency	1	2	5	1	2	1	1	2	1	2	1	1	1	1	5

Huffman code

Letter	Freq	Code	Bits
" "	5	01	2
e	5	00	2
h	2	1101	4
i	2	1001	4
o	2	1000	4
l	2	1010	4
T	1	11100	5
r	1	10111	5
s	1	11110	5
n	1	111111	6
p	1	11001	5
a	1	10110	5
c	1	111110	6
k	1	11001	5
m	1	11000	5

Total Bits = 99 bits

Huffman Tree



Task 3 : Compression Analysis

Step 1. Compressing Text Files

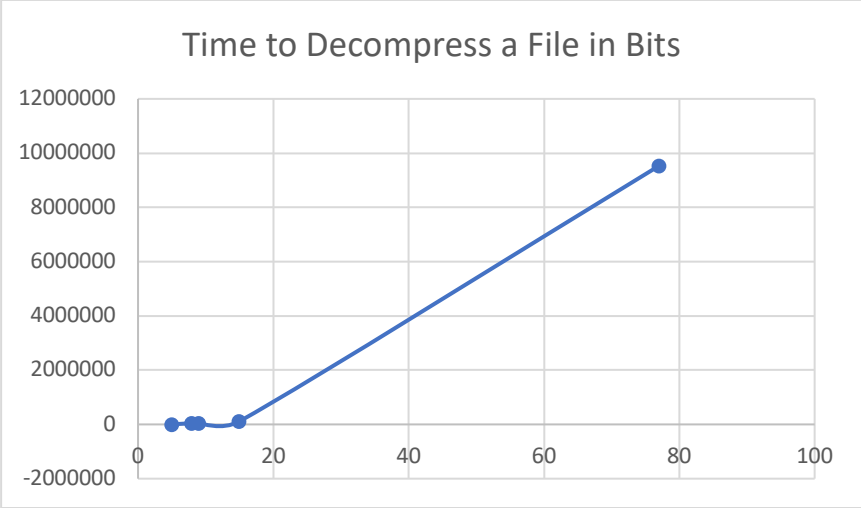
Input File Name	Time to Compress	File Size After Compression	File Size Before Compression	Compression Ratio
q32x48.bin	4 ms	816 bits	1536 bits	0.5313
medTale.txt	14 ms	23888 bits	45024 bits	0.5306
genomeVirus.txt	13 ms	12576 bits	50008 bits	0.2515
cognitive.txt	22 ms	65112 bits	114816 bits	0.5671
mobydick.txt	141 ms	5341208 bits	9531696 bits	0.5604

*compression ratio = compressed bits / original bits

Step 2. Decompressing the Compressed Text Files

Input File Name	Time to Decompress	File Size After Decompression	Final Bits in Decompressed File (8)
binCO.txt	5 milliseconds	1536 bits	00000000
medTaleCO.txt	8 milliseconds	45024 bits	01101101
genomeCO.txt	9 milliseconds	50008 bits	01000001
cognitiveCO.txt	15 milliseconds	114816 bits	00001010
mobydickCO.txt	77 milliseconds	9531696 bits	00101110

* please note, for the additional file I chose to use “cognitive.txt”



For the most part, the compression ratios were relatively the same. However, the compression ratio for genomeVirus.txt was significantly lower. I believe that this happened because there are only four letters used to represent genomes – as such, the text was comprised of these four characters only. Additionally, I noticed that the time to compress / decompress the file appears to be linear to the size of the file. The graph above depicts the time (x-axis) to decompress the file (in bits on the y-axis) and appears to be proportional.

Step 3. Analysis of Results

Question 3 : What happens if you try to compress one of the already compressed files? Why do you think this occurs?
When I tried to compress the file cognitiveCO.txt, it took nearly twice the amount of time to compress the file, at 31 milliseconds. Additionally, the compressed version of cognitiveCO.txt ended up being larger. This text file is labeled trial.txt and contains 8339 bytes, while cognitiveCO.txt was only 8139 bytes. I think that the compressed version ended up being larger because it may have taken more bytes to build the Huffman Tree when creating it in bytes. It may have taken longer because it would have to convert the bitstream back into characters. Additionally, it would have to look for a more optimized way to compress the already compressed file. The compressed file cannot be further compressed using the Huffman algorithm, because the same characters are being used with the same frequencies.

Question 4: Use the provided RunLength function to compress the bitmap file q32x48.bin. Do the same with your Huffman algorithm. Compare your results. What reason can you give for the difference in compression rates?
When I used the RunLength function, it compressed the original file from 192 to 15 bytes for a compression ratio of 7.81%. When I ran the bitmap file with my Huffman algorithm, it compressed the original file from 192 to 102 bytes for a compression ratio of 53.13%. I think the difference in compression rates is caused by RunLength taking into account how long the runs of characters are. For example, if the characters are repeated multiple times and appear in runs, then it can compress with a better ratio since there will be less bytes needed to represent the whole string. However, if there were many characters used in short or no runs, it would take more bytes to represent the provided string. As such, it can be reasoned that the file q32x48.bin must have had characters that appeared in runs, allowing for a better compression rate.