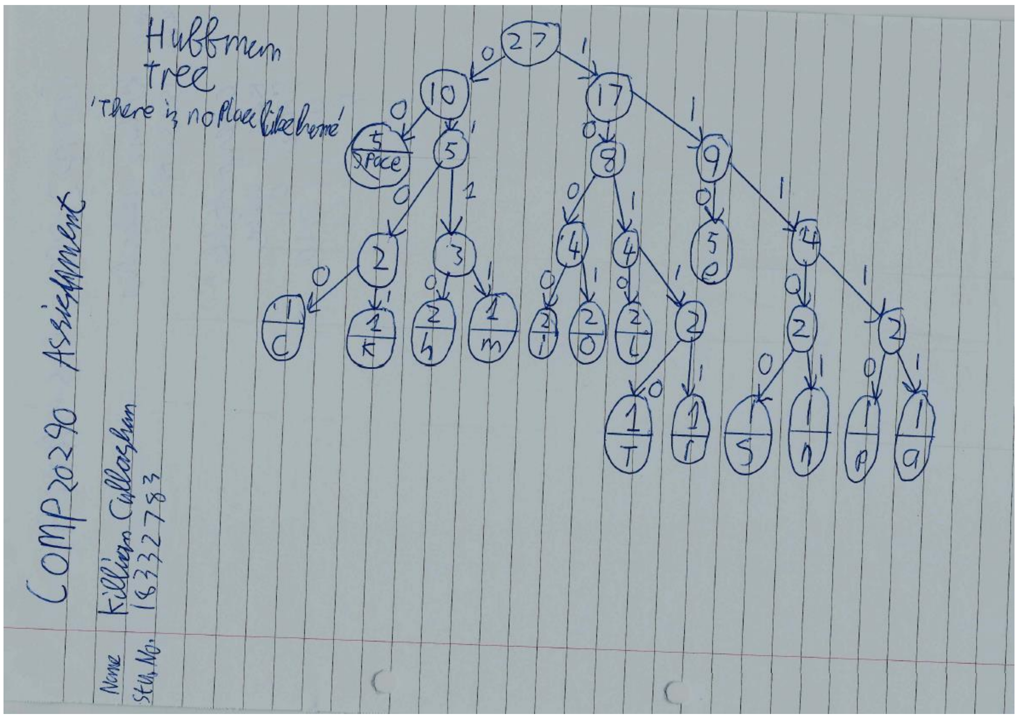
COMP20290 Algorithms Assignment

18332783

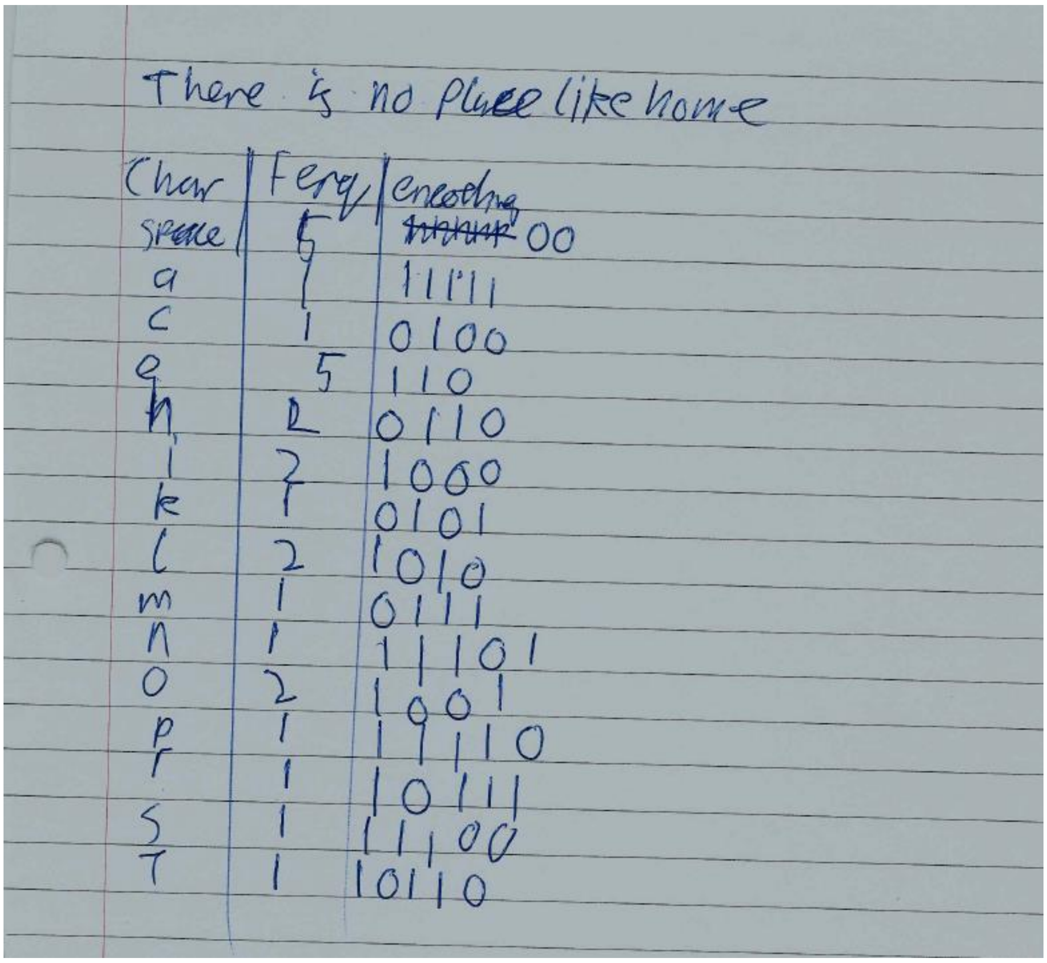
Killian Callaghan

Task 1:

Huffman Tree:



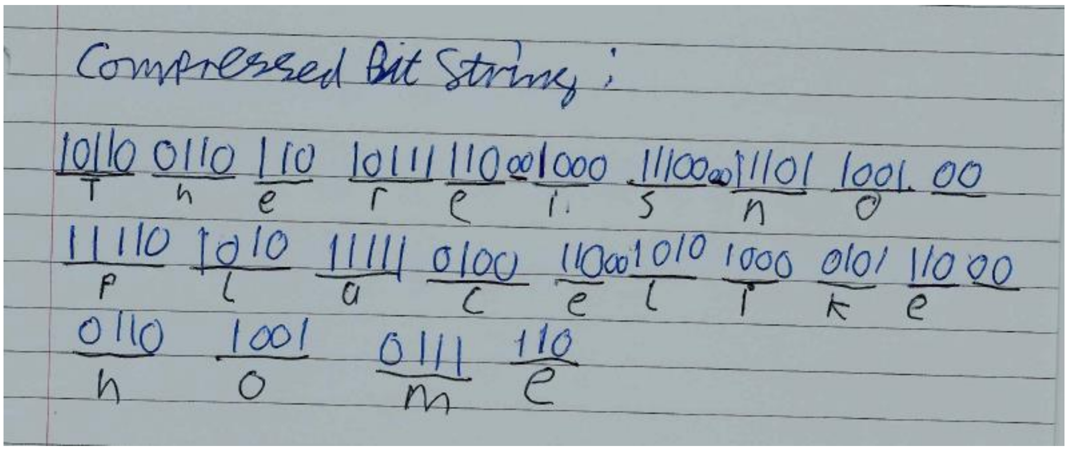
Frequency Table:



There is no place like home

Char	Ferq	encoding
space	5	~~[illegible]~~ 00
a	1	11111
c	1	0100
e	5	110
h	2	0110
i	2	1000
k	1	0101
l	2	1010
m	1	0111
n	1	11101
o	2	1001
p	1	11110
r	1	10111
s	1	11100
T	1	10110

Compressed bit string:



Compressed Bit String :

$\underset{T}{10110}\ \underset{h}{0110}\ \underset{e}{110}\ \underset{r}{10111}\ \underset{e}{110}\ 00\ \underset{i}{1000}\ \underset{s}{11100}\ 00\ \underset{n}{11101}\ \underset{o}{1001}\ 00$

$\underset{p}{11110}\ \underset{l}{1010}\ \underset{a}{11111}\ \underset{c}{0100}\ \underset{e}{110}\ 00\ \underset{l}{1010}\ \underset{i}{1000}\ \underset{k}{0101}\ \underset{e}{110}\ 00$

$\underset{h}{0110}\ \underset{o}{1001}\ \underset{m}{0111}\ \underset{e}{110}$

Task 3:

Step 1 & 2:

Q1 & Q2

File Name	Time taken to encode (nano seconds)	Time taken to decode (nano seconds)	Original size	Encoded size	Decoded size	Compression ratio
mobydick.txt	109617500	79536500	1164KB	5058KB	1142KB	194/843
medTale.txt	76435300	116965500	6KB	23KB	6KB	6/23
genomeVirus.txt	25057400	15235700	7KB	12KB	7KB	7/12
Metal Gear Solid 3 Snake Eater Script.txt	47112400	48105200	249KB	1095KB	234KB	83/365

Step 3:

Assess the results above:

From the results above I can conclude many things. The larger the file size, the longer it takes to encode and decode. This is because the larger files have more characters to process.

We can see with the larger files, when it is encoded and then decoded the file size doesn't return to its original size, but rather slightly smaller. This suggests that the compression isn't lossless. This is because my method does not scan the original file correctly. New lines are not scanned correctly and are just ignored. In large text files these ignored new lines add up and is noticeable. Due to time restrictions and having to work at home on my own, I could not find a solution to this problem.

Another issue is that encoded files are actually larger in size than the original files. I believe this is because there is more characters in the encoded files. For example the letter 'A' could have a prefix '10011'. The prefix has more characters than the single letter. The binary numbers are stored as characters rather than as binary values which are smaller in size compared to characters. My Compression ratio calculations might be inaccurate as a result. Due to time restrictions and having to work at home on my own, I could not find a solution to this problem.

In conclusion I do believe my algorithm works despite the undesirable recordings. My programme doesn’t store new lines correctly is a problem but due to time I cannot solve this problem. I believe I have mostly accomplished what was set for this assignment.

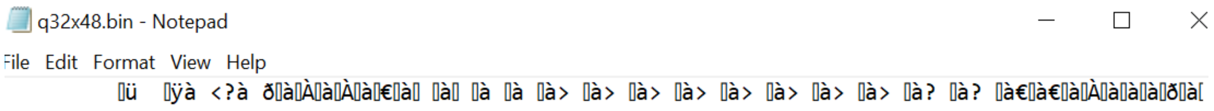
Q3

When I compressed an already compressed file a few things happen. The time taken to encode takes longer. I believe this is because as there is more characters or numbers to process, it takes longer to encode.

The file size remains the same. I believe this occurs because the encoded file contains only ones and zeros, which when compressed are assigned as one or zero. Since there are only two unique characters found in the encoded file, the character or number ‘1’ has a prefix ‘1’ or ‘0’ and the character or number ‘0’ is has a prefix ‘1’ or ‘0’. When I open the file I can see that the file is identical to when it was encoded the first time. This means that ‘1’ has a prefix ‘1’ and ‘0’ has a prefix ‘0’. The size of the file remains the same since it is the same.

Q4

I tried running the provided RunLength function. However, I could not get it to work. Also for running q32x48.bin in my own algorithm it doesn’t work. I cannot figure out why but my guess is it is because the text inside is illegible and my programme is not able to read it.



```
          ü   ÿà <?à ðàÀàÀà€à à à à à> à> à> à> à> à> à> à> à? à? à€à€àÀàààðà
```