Compression

# Introduction

# Homework Exercise

## Task 1

A screenshot of a computer code

Description automatically generated

001001011011110111 = 00 100 101 101 111 01 11 = ACDDFB 11

The last 2 digits can not be converted because there is no code for 11. If we reverse the characters, we can fully decode it getting:

111011110110100100 = 111 01 111 01 101 00 100 = FBFBDAC

However, in both cases the message is un-readable so without more context it is impossible to tell if this was a mistake or reversed to further obfuscate the message.

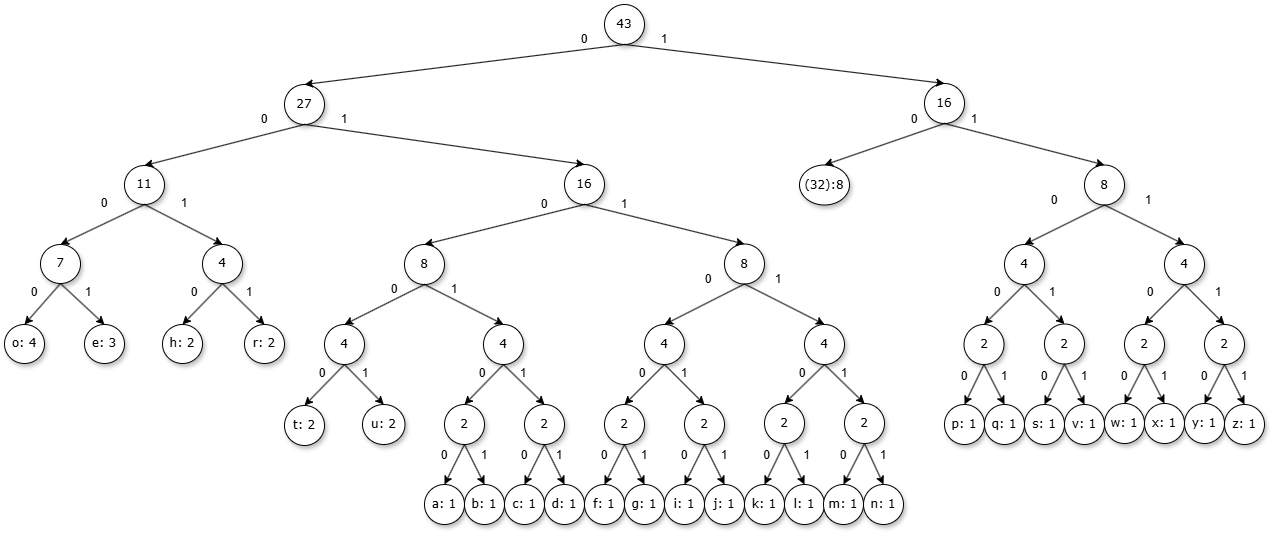
## Task 2

The first step is to count the occurrence of each character in the sentence, (32) is used to represent a space character. The output from the program written to do this is shown in Figure 2. We then put this data into some diagram software and build the Huffman tree as shown in Figure 3. I have also provided Table 1, which is easier to use for practical purposes. Finally, we use the table to produce the following encoded binary message (hyphen delimited for easier reading):

01000-0010-0001-10-11001-01001-011010-010110-011100-10-010101-0011-0000-11100-011111-10-011000-0000-11101-10-011011-01001-011110-11000-11010-10-0000-11011-0001-0011-10-01000-0010-0001-10-011101-010100-11111-11110-10-010111-0000-011001

A screenshot of a computer

Description automatically generated



A white sheet with black lines and letters

Description automatically generated with medium confidence

## Task 3

### Compression Ratio

Using an 8-bit ASCII encoding for this message the number of bits used would be 43 characters x 8 bits = 344 bits.

As there are 192 characters in the encoded message and as each character represents a single bit the Huffman encoded message uses 192 bits.

The compression ratio achieved using Huffman encoding would be 192 / 344 = 0.558. This means that the encoded message contains the same information in 55.8% of the storage space provided the recipient knows or can derive the dictionary.

### Fixed-Length Encodings

The string only uses the 26 lowercase alphabetic characters and the space character and thus could be represented by any number of bits n where 2n >= 27. The shortest of these is 5 bits which if used would produce an encoded message with a compression ratio of 5/8 or 0.625, significantly less efficient that Huffman encoding.

## Task 4

The way a Huffman tree is constructed indirectly compares the sum of bits saved by encoding a character as a reduced length encoding to the sum of bits cost by doing so and this means by its very structure it produces the most efficient binary encoding possible and cannot be beaten by a fixed-length encoding. Importantly this ignores the cost of communicating the encoding and so in some cases where the message is small, or the Huffman encoding is identical to the fixed-length encoding an agreed upon fixed-length encoding may be more efficient.

Shannon Entropy in this context is a way to measure the amount of entropy in a string using the formula in Figure 5. If the string contained only one type of character, then entropy would be 0 and we would theoretically require 0 bits per character. In the worst case all characters have an equal probability, and we get an entropy of log2(n) where n is the number of unique characters in the string. I have created a program to calculate the Shannon Entropy and the average code length of a given Huffman encoding with the output for my encoding shown in figure 6. Shannon Entropy for the string is 4.385 bits/character and our encoding has achieved 4.465 bits/character making it a reasonable approximation given that bits are discrete.

A mathematical equation with numbers and symbols

Description automatically generated

A screenshot of a computer program

Description automatically generated

# Extension Exercise