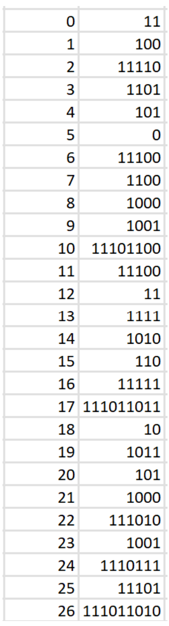
Space for project label

# CATAM: 19.2 Information content of natural language

## 1

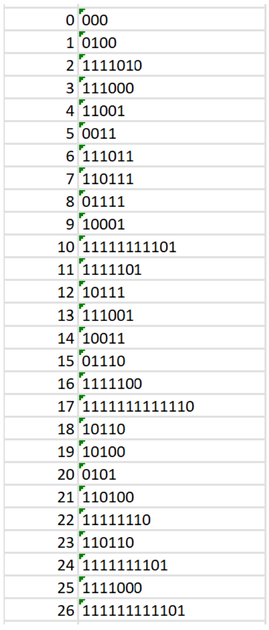
Text A is chosen to answer the questions in this project. Refer to q1.m in Appendix (A) for the program for this question.
The source entropy found from q1.m is returned in h = 4.0688 (4 d.p.)
Huffman code returned from the program can be seen in figure (1)



| | |
|---|---|
| 0 | 11 |
| 1 | 100 |
| 2 | 11110 |
| 3 | 1101 |
| 4 | 101 |
| 5 | 0 |
| 6 | 11100 |
| 7 | 1100 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 11101100 |
| 11 | 11100 |
| 12 | 11 |
| 13 | 1111 |
| 14 | 1010 |
| 15 | 110 |
| 16 | 11111 |
| 17 | 111011011 |
| 18 | 10 |
| 19 | 1011 |
| 20 | 101 |
| 21 | 1000 |
| 22 | 111010 |
| 23 | 1001 |
| 24 | 1110111 |
| 25 | 11101 |
| 26 | 111011010 |

Figure 1: left column is the alphabet and right column is the corresponding huffman code

The expected codeword length is returned as E = 4.1142 (4 d.p.)
Shannon-fano code returned from the program can be seen in figure (2) Where in the program I have used the method of working out the lengths with $l_i = \lceil -log_2(p_i) \rceil$, where $l_i$ is the length and $p_i$ is the probability of the letter represented by i. Then ordering the probabilities such that $p_1 \geq p_2 \geq ... \geq p_{27}$, define cumulative probability $P_i = \sum_{j=1}^{i-1} p_i$. Then define the codeword for letter i as the first $l_i$ digits of the binary representation of $P_i$ after the decimal point.
This contructs a prefix-free code of the desired lengths:
From the definition of $l_i$, we know $log_2(\frac{1}{p_i}) \leq l_i < log_2(\frac{1}{p_i}) + 1 \Rightarrow \frac{1}{2^{l_i}} \leq p_i < \frac{1}{2^{l_i - 1}}$. Therefore the code for $P_i$ will differ from all succeeding codes in one or more of its $l_i$ places, all the remaining $P_i$ are at least $\frac{1}{2^{l_i}}$ larger, so their binary expansions will differ in the first $l_i$ places. This means the code generated from this method is prefix free.
The expected codeword length is returned as ESF = 4.6136 (4 d.p.)
The expected codeword length for Huffman's code is less than for the Shannon-fano code, and both are greater than the source entropy, as expected from Shannon's noiseless encoding theorem.



| | |
|---|---|
| 0 | 000 |
| 1 | 0100 |
| 2 | 1111010 |
| 3 | 111000 |
| 4 | 11001 |
| 5 | 0011 |
| 6 | 111011 |
| 7 | 110111 |
| 8 | 01111 |
| 9 | 10001 |
| 10 | 11111111101 |
| 11 | 1111101 |
| 12 | 10111 |
| 13 | 111001 |
| 14 | 10011 |
| 15 | 01110 |
| 16 | 1111100 |
| 17 | 1111111111110 |
| 18 | 10110 |
| 19 | 10100 |
| 20 | 0101 |
| 21 | 110100 |
| 22 | 11111110 |
| 23 | 110110 |
| 24 | 1111111101 |
| 25 | 1111000 |
| 26 | 111111111101 |

Figure 2: left column is the alphabet and the right column is the corresponding Shannon-Fano code

How would segmentation improve expected length if source was assumed Bernoulli?
Previously we know by Shannon's noiseless encoding theorem that $h \leq \mathbb{E}(S) < h + 1$, where S is the length of optimal codewords and h is the source entropy.
With segmentation, let $S^{(n)}$ be the length of an optimal codeword for a segment of length n, and let $h_n$ be the source entropy of the new code after segmentation. Then from the same way as before, with Shannon's noiseless encoding theorem, $\frac{h_n}{n} \leq \mathbb{E}(\frac{S^{(n)}}{n}) < \frac{h_n}{n} + \frac{1}{n}$. Where we look at $\frac{S^{(n)}}{n}$ instead as this represents the length of codeword per source letter, which is more comparable to the results before segmentation.[1]
By claim (1.1), we then know that $h \leq \mathbb{E}(S^{(n)}) < h + \frac{1}{n}$. Hence $\mathbb{E}(S^{(n)}) \to h$ as $n \to \infty$.
Thus there is advantage in segmented codes in terms of length of code. But this is ignoring the increased storage requirements need to store the new increased alphabet $I_m^n$ and their respective codewords.

**Claim 1.1.** *For a Bernoulli source, $h_n = n \times h$.*

*Proof.* Since source is Bernoulli, the probability of a message of length n occuring is $p_{i_1}p_{i_2}...p_{i_n}$, where $p_{i_j}$ is the probability of the letter corresponding with $i_j$ occuring, where $i_1, i_2, ..., i_n \in \{0, ..., 26\}$.

$$\Rightarrow h_n = -\sum_{i_1,i_2,...,i_n} p_{i_1}p_{i_2}...p_{i_n} log(p_{i_1}p_{i_2}...p_{i_n})$$

$$\Rightarrow h_n = -\sum_{i_1,i_2,...,i_n} p_{i_1}p_{i_2}...p_{i_n}(\sum_{j=1}^{n} log(p_{i_j}))$$

Rearranging the sum:

$$\Rightarrow h_n = n \times \sum_{j=1}^{n} log(p_j)p_j(\sum_{i_1,i_2,...,i_{n-1} \neq j} p_{i_1}p_{i_2}...p_{i_{n-1}}) = nh$$

□

## 2

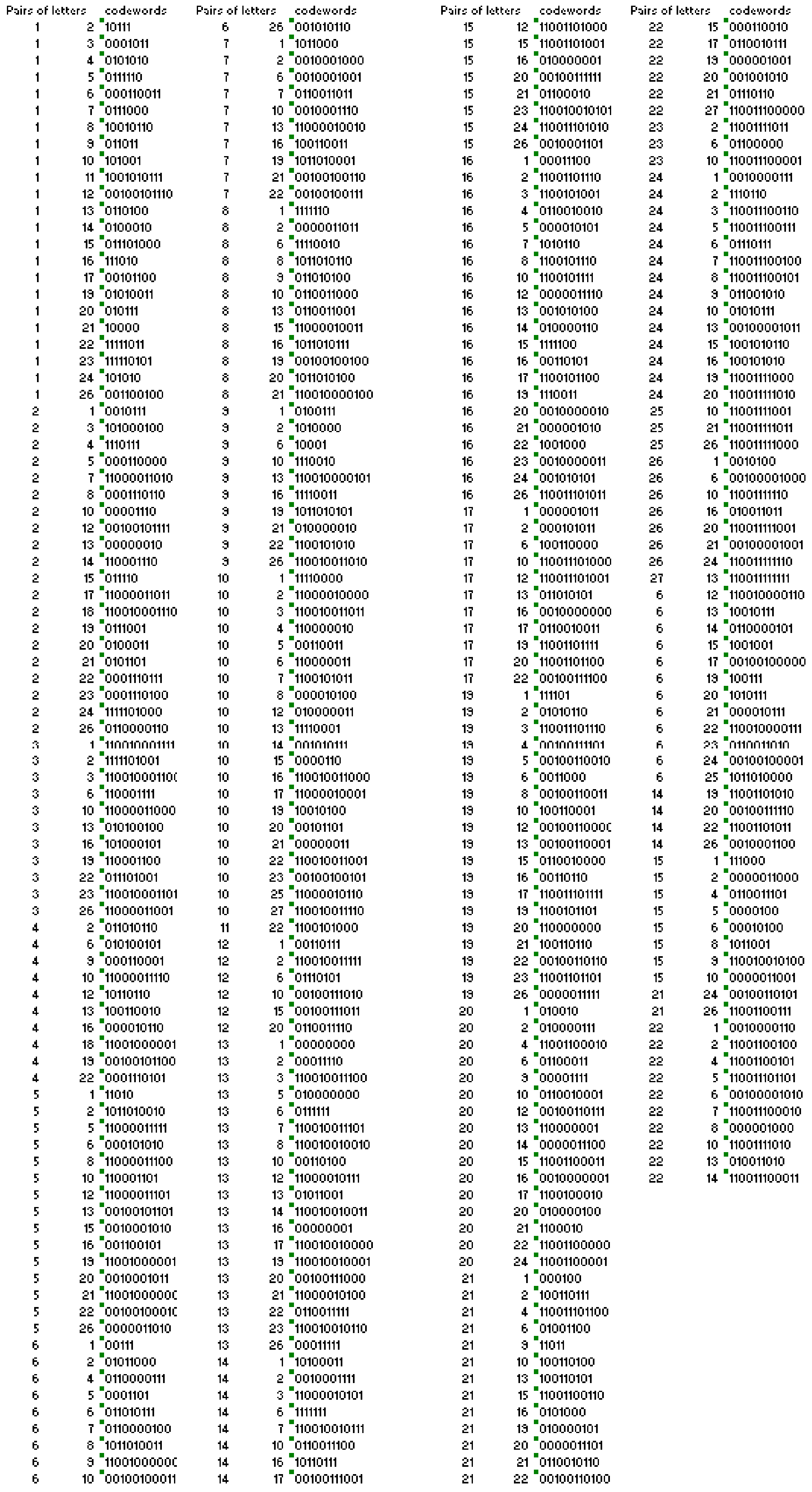
Refer to q2.m in Appendix (B) for the program for this question.
The English text is definitely not Bernoulli, for example if the previous letter is space, the probability of the following letter being space is 0.
Looking at the probabilities of pairs occuring deduced from the source text in the program, from the 729 possible combinations, only 334 are non-zero. If source was Bernoulli, then all the probabilities would be non-zero because the probabilities for the original alphabet is all non-zero.
Certain pairs of letters that do not appear in the text are assumed to have probability 0, so will be ignored and not assigned any codeword. See figure (3) for Huffman code assigned to pairs of letters for the pairs that appear in the text. In this table the numbers associated with the alphabet has been shifted by 1, ie. 1 - space, 2 - A, 3 - B, etc.

The expected length is then also calculated from this with the result of $\mathbb{E}(S^{(n)}/n) = 3.6572$ (4 d.p.).
This is an improvement on expected word length per source letter compared to the Huffman code in question 1 with E = 4.1142 before segmentation. Hence segmentation reduces the expected length of codewords.

If the source was Bernoulli, the expected codeword length per source letter would be at least the source entropy from question 1 (h = 4.0688). So English text has shorter expected length. This confirms that English text is not Bernoulli, as the result does not match what we proved in question 1 for Bernoulli sources. In fact segmentation improves the expected word length more than expected of a bernoulli source, this is because segmentation helps include some of the relations of the letters to each other.

## References

[1] Goldie CM, Pinch RGE. Communication Theory; 1991.



| Pairs of letters | | codewords |
|---|---|---|
| 1 | 2 | 10111 |
| 1 | 3 | 0001011 |
| 1 | 4 | 0101010 |
| 1 | 5 | 0111110 |
| 1 | 6 | 000110011 |
| 1 | 7 | 0111000 |
| 1 | 8 | 10010110 |
| 1 | 9 | 011011 |
| 1 | 10 | 101001 |
| 1 | 11 | 1001010111 |
| 1 | 12 | 00100101110 |
| 1 | 13 | 0110100 |
| 1 | 14 | 0100010 |
| 1 | 15 | 011101000 |
| 1 | 16 | 111010 |
| 1 | 17 | 00101100 |
| 1 | 19 | 01010011 |
| 1 | 20 | 010111 |
| 1 | 21 | 10000 |
| 1 | 22 | 11111011 |
| 1 | 23 | 111110101 |
| 1 | 24 | 101010 |
| 1 | 26 | 001100100 |
| 2 | 1 | 0010111 |
| 2 | 3 | 101000100 |
| 2 | 4 | 1110111 |
| 2 | 5 | 000110000 |
| 2 | 7 | 11000011010 |
| 2 | 8 | 0001110110 |
| 2 | 10 | 00001110 |
| 2 | 12 | 00100101111 |
| 2 | 13 | 00000010 |
| 2 | 14 | 110001110 |
| 2 | 15 | 011110 |
| 2 | 17 | 11000011011 |
| 2 | 18 | 110010001110 |
| 2 | 19 | 0111001 |
| 2 | 20 | 0100011 |
| 2 | 21 | 0101101 |
| 2 | 22 | 0001110111 |
| 2 | 23 | 0001110100 |
| 2 | 24 | 1111101000 |
| 2 | 26 | 0110000110 |
| 3 | 1 | 110010001111 |
| 3 | 2 | 1111101001 |
| 3 | 3 | 11001000110 |
| 3 | 6 | 110001111 |
| 3 | 10 | 11000011000 |
| 3 | 13 | 010100100 |
| 3 | 16 | 101000101 |
| 3 | 19 | 110001100 |
| 3 | 22 | 011101001 |
| 3 | 23 | 110010001101 |
| 3 | 26 | 11000011001 |
| 4 | 2 | 011010110 |
| 4 | 6 | 010100101 |
| 4 | 9 | 000110001 |
| 4 | 10 | 11000011110 |
| 4 | 12 | 10110110 |
| 4 | 13 | 100110010 |
| 4 | 16 | 000010110 |
| 4 | 18 | 11001000001 |
| 4 | 19 | 00100101100 |
| 4 | 22 | 0001110101 |
| 5 | 1 | 11010 |
| 5 | 2 | 1011010010 |
| 5 | 5 | 11000011111 |
| 5 | 6 | 000101010 |
| 5 | 8 | 11000011100 |
| 5 | 10 | 110001101 |
| 5 | 12 | 11000011101 |
| 5 | 13 | 00100101101 |
| 5 | 15 | 0010001010 |
| 5 | 16 | 001100101 |
| 5 | 19 | 11001000001 |
| 5 | 20 | 0010001011 |
| 5 | 21 | 11001000000 |
| 5 | 22 | 00100100010 |
| 5 | 26 | 0000011010 |
| 6 | 1 | 00111 |
| 6 | 2 | 01011000 |
| 6 | 4 | 0110000111 |
| 6 | 5 | 0001101 |
| 6 | 6 | 011010111 |
| 6 | 7 | 0110000100 |
| 6 | 8 | 1011010011 |
| 6 | 9 | 11001000000 |
| 6 | 10 | 00100100011 |
| 6 | 26 | 001010110 |
| 7 | 1 | 1011000 |
| 7 | 2 | 0010001000 |
| 7 | 6 | 0010001001 |
| 7 | 7 | 0110011011 |
| 7 | 10 | 0010001110 |
| 7 | 13 | 11000010010 |
| 7 | 16 | 100110011 |
| 7 | 19 | 1011010001 |
| 7 | 21 | 00100100110 |
| 7 | 22 | 00100100111 |
| 8 | 1 | 1111110 |
| 8 | 2 | 0000011011 |
| 8 | 6 | 11110010 |
| 8 | 8 | 1011010110 |
| 8 | 9 | 011010100 |
| 8 | 10 | 0110011000 |
| 8 | 13 | 0110011001 |
| 8 | 15 | 11000010011 |
| 8 | 16 | 1011010111 |
| 8 | 19 | 00100100100 |
| 8 | 20 | 1011010100 |
| 8 | 21 | 110010000100 |
| 9 | 1 | 0100111 |
| 9 | 2 | 1010000 |
| 9 | 6 | 10001 |
| 9 | 10 | 1110010 |
| 9 | 13 | 110010000101 |
| 9 | 16 | 11110011 |
| 9 | 19 | 1011010101 |
| 9 | 21 | 010000010 |
| 9 | 22 | 1100101010 |
| 9 | 26 | 110010011010 |
| 10 | 1 | 11110000 |
| 10 | 2 | 11000010000 |
| 10 | 3 | 110010011011 |
| 10 | 4 | 110000010 |
| 10 | 5 | 00110011 |
| 10 | 6 | 110000011 |
| 10 | 7 | 1100101011 |
| 10 | 8 | 000010100 |
| 10 | 12 | 010000011 |
| 10 | 13 | 11110001 |
| 10 | 14 | 001010111 |
| 10 | 15 | 0000110 |
| 10 | 16 | 110010011000 |
| 10 | 17 | 11000010001 |
| 10 | 19 | 10010100 |
| 10 | 20 | 00101101 |
| 10 | 21 | 00000011 |
| 10 | 22 | 110010011001 |
| 10 | 23 | 00100100101 |
| 10 | 25 | 11000010110 |
| 10 | 27 | 110010011110 |
| 11 | 22 | 1100101000 |
| 12 | 1 | 00110111 |
| 12 | 2 | 110010011111 |
| 12 | 6 | 01110101 |
| 12 | 10 | 00100111010 |
| 12 | 15 | 00100111011 |
| 12 | 20 | 0110011110 |
| 13 | 1 | 00000000 |
| 13 | 2 | 00011110 |
| 13 | 3 | 110010011100 |
| 13 | 5 | 010000000 |
| 13 | 6 | 0111111 |
| 13 | 7 | 110010011101 |
| 13 | 8 | 110010010010 |
| 13 | 10 | 00110100 |
| 13 | 12 | 11000010111 |
| 13 | 13 | 01011001 |
| 13 | 14 | 110010010011 |
| 13 | 16 | 00000001 |
| 13 | 17 | 110010010000 |
| 13 | 19 | 110010010001 |
| 13 | 20 | 00100111000 |
| 13 | 21 | 11000010100 |
| 13 | 22 | 0110011111 |
| 13 | 23 | 110010010110 |
| 13 | 26 | 00011111 |
| 14 | 1 | 10100011 |
| 14 | 2 | 0010001111 |
| 14 | 3 | 11000010101 |
| 14 | 6 | 1111111 |
| 14 | 7 | 110010010111 |
| 14 | 10 | 0110011100 |
| 14 | 16 | 10110111 |
| 14 | 17 | 00100111001 |
| 15 | 12 | 11001101000 |
| 15 | 15 | 11001101001 |
| 15 | 16 | 010000001 |
| 15 | 20 | 00100111111 |
| 15 | 21 | 01100010 |
| 15 | 23 | 110010010101 |
| 15 | 24 | 110011101010 |
| 15 | 26 | 0010001101 |
| 16 | 1 | 00011100 |
| 16 | 2 | 11001101110 |
| 16 | 3 | 1100101001 |
| 16 | 4 | 0110010010 |
| 16 | 5 | 000010101 |
| 16 | 7 | 1010110 |
| 16 | 8 | 1100101110 |
| 16 | 10 | 1100101111 |
| 16 | 12 | 0000011110 |
| 16 | 13 | 001010100 |
| 16 | 14 | 010000110 |
| 16 | 15 | 1111100 |
| 16 | 16 | 00110101 |
| 16 | 17 | 1100101100 |
| 16 | 19 | 1110011 |
| 16 | 20 | 0010000010 |
| 16 | 21 | 000001010 |
| 16 | 22 | 1001000 |
| 16 | 23 | 0010000011 |
| 16 | 24 | 001010101 |
| 16 | 26 | 110011101011 |
| 17 | 1 | 000001011 |
| 17 | 2 | 000101011 |
| 17 | 6 | 100110000 |
| 17 | 10 | 110011101000 |
| 17 | 12 | 110011101001 |
| 17 | 13 | 011010101 |
| 17 | 16 | 0010000000 |
| 17 | 17 | 0110010011 |
| 17 | 19 | 11001101111 |
| 17 | 20 | 11001101100 |
| 17 | 22 | 00100111100 |
| 19 | 1 | 111101 |
| 19 | 2 | 01010110 |
| 19 | 3 | 110011101110 |
| 19 | 4 | 00100111101 |
| 19 | 5 | 00100110010 |
| 19 | 6 | 0011000 |
| 19 | 8 | 00100110011 |
| 19 | 10 | 100110001 |
| 19 | 12 | 00100110000 |
| 19 | 13 | 00100110001 |
| 19 | 15 | 0110010000 |
| 19 | 16 | 00110110 |
| 19 | 17 | 110011101111 |
| 19 | 19 | 1100101101 |
| 19 | 20 | 110000000 |
| 19 | 21 | 100110110 |
| 19 | 22 | 00100110110 |
| 19 | 23 | 11001101101 |
| 19 | 26 | 0000011111 |
| 20 | 1 | 010010 |
| 20 | 2 | 010000111 |
| 20 | 4 | 11001100010 |
| 20 | 6 | 01100011 |
| 20 | 9 | 00001111 |
| 20 | 10 | 0110010001 |
| 20 | 12 | 00100110111 |
| 20 | 13 | 110000001 |
| 20 | 14 | 0000011100 |
| 20 | 15 | 11001100011 |
| 20 | 16 | 0010000001 |
| 20 | 17 | 1100100010 |
| 20 | 20 | 010000100 |
| 20 | 21 | 1100010 |
| 20 | 22 | 11001100000 |
| 20 | 24 | 11001100001 |
| 21 | 1 | 000100 |
| 21 | 2 | 100110111 |
| 21 | 4 | 110011101100 |
| 21 | 6 | 01001100 |
| 21 | 9 | 11011 |
| 21 | 10 | 100110100 |
| 21 | 13 | 100110101 |
| 21 | 15 | 11001100110 |
| 21 | 16 | 0101000 |
| 21 | 19 | 010000101 |
| 21 | 20 | 0000011101 |
| 21 | 21 | 0110010110 |
| 21 | 22 | 00100110100 |
| 22 | 15 | 000110010 |
| 22 | 17 | 0110010111 |
| 22 | 19 | 000001001 |
| 22 | 20 | 001001010 |
| 22 | 21 | 01110110 |
| 22 | 27 | 110011100000 |
| 23 | 2 | 11001111011 |
| 23 | 6 | 01100000 |
| 23 | 10 | 110011100001 |
| 24 | 1 | 0010000111 |
| 24 | 2 | 1110110 |
| 24 | 3 | 110011100110 |
| 24 | 5 | 110011100111 |
| 24 | 6 | 01110111 |
| 24 | 7 | 110011100100 |
| 24 | 8 | 110011100101 |
| 24 | 9 | 011001010 |
| 24 | 10 | 01010111 |
| 24 | 13 | 00100001011 |
| 24 | 15 | 1001010110 |
| 24 | 16 | 100101010 |
| 24 | 19 | 11001111000 |
| 24 | 20 | 110011111010 |
| 25 | 10 | 11001111001 |
| 25 | 21 | 110011111011 |
| 25 | 26 | 110011111000 |
| 26 | 1 | 0010100 |
| 26 | 6 | 00100001000 |
| 26 | 10 | 1100111110 |
| 26 | 16 | 010011011 |
| 26 | 20 | 110011111001 |
| 26 | 21 | 00100001001 |
| 26 | 24 | 110011111110 |
| 27 | 13 | 110011111111 |
| 6 | 12 | 110010000110 |
| 6 | 13 | 10010111 |
| 6 | 14 | 0110000101 |
| 6 | 15 | 1001001 |
| 6 | 17 | 00100100000 |
| 6 | 19 | 100111 |
| 6 | 20 | 1010111 |
| 6 | 21 | 000010111 |
| 6 | 22 | 110010000111 |
| 6 | 23 | 0110011010 |
| 6 | 24 | 00100100001 |
| 6 | 25 | 1011010000 |
| 14 | 19 | 11001101010 |
| 14 | 20 | 00100111110 |
| 14 | 22 | 11001101011 |
| 14 | 26 | 0010001100 |
| 15 | 1 | 111000 |
| 15 | 2 | 0000011000 |
| 15 | 4 | 0110011101 |
| 15 | 5 | 0000100 |
| 15 | 6 | 00010100 |
| 15 | 8 | 1011001 |
| 15 | 9 | 110010010100 |
| 15 | 10 | 0000011001 |
| 21 | 24 | 00100110101 |
| 21 | 26 | 11001100111 |
| 22 | 1 | 0010000110 |
| 22 | 2 | 11001100100 |
| 22 | 4 | 11001100101 |
| 22 | 5 | 110011101101 |
| 22 | 6 | 00100001010 |
| 22 | 7 | 110011100010 |
| 22 | 8 | 000001000 |
| 22 | 10 | 11001111010 |
| 22 | 13 | 010011010 |
| 22 | 14 | 110011100011 |

Figure 3: pairs of letters and their corresponding huffman codewords

# Appendix

## A q1.m

```
function [h, codewords, codewordsSF] = q1()
%h stores source entropy
%codewords contains codewords after hamming's code
%codewordsSF contains codewords after shannon-fano code
clear
format long
A = readmatrix("https://www.maths.cam.ac.uk/undergrad/catam/data/II-19-2-dataA.txt");
A(end, :) = [];
A = A + 1;%add one to every letter, so we work in 1= space, 2 = A, 3 = B, ... to fit matlab
    ↪ better
%disp(A)
%first need to create list of probabilities for alphabet in this text.
n = 400*25; %is the number of characters in the text
p = zeros(27, 1);
for i = 1:n
    for j = 1:27
        if A(i) == (j)
            p(j) = p(j) + 1;
        end
    end
end
%now divide all entries of p my n so that it is a probability
p = p./n;

%with this we can work out source entropy
h = 0;
for i = 1:27
    h = h - p(i) * log2(p(i));
end

letters = [1:1:27]';

p = [letters p];

psorted = sortrows(p, 2, {'descend'});%sorted by probabilities

codewords = strings(27, 1);%This will store the code words corresponding to the alphabet
%where position 1 stores cw for space, position 2 stores cw for A, etc.

I = num2cell(psorted); %first create copy of p as cell

for i = 1:26
    last1 = cell2mat(I(end, 1));
    last2 = cell2mat(I(end-1, 1));

    for k = 1:size(last1, 2)
        codewords(last1(k)) = '1' + codewords(last1(k));
    end
    for l = 1 : size(last2, 2)
        codewords(last2(l)) = '0' + codewords(last2(l));
    end

    I(end-1, 1) = {[last2, last1]};
    I{end-1, 2} = I{end-1, 2} + I{end, 2};
    I(end, :) = [];
    I = sortrows(I, 2, 'descend');
end

%now with codewords we can work out expected word length, E
E = 0;
for i = 1:27
    E = E + p(i, 2) * strlength(codewords(i));
end

%Now that's done do the same for Shannon-fano

lengths = ceil(-log2(p(:, 2)));
%lengths = [letters lengths];
%lengths = sortrows(lengths, 2);

cumulativep = zeros(27, 1);%will store the cumulative probabilities of the sorted p

for i = 2:27
    cumulativep(i) = cumulativep(i-1) + psorted(i-1, 2);
end

cumulativep = [psorted(:, 1) cumulativep];
cumulativep = sortrows(cumulativep, 1, 'ascend');

%Now workout codewords. again pos1 stores space, pos2 stores A etc
codewordsSF = strings(27, 1);

for i = 1:27
    for j = 1:lengths(i)
        temp = cumulativep(i, 2)*2;
        if temp >= 1
            codewordsSF(i) = codewordsSF(i) + '1';
            cumulativep(i, 2) = cumulativep(i, 2)*2 - 1;
        else
            codewordsSF(i) = codewordsSF(i) + '0';
            cumulativep(i, 2) = cumulativep(i, 2)*2;
        end
    end
end

%now work out corresponding expected length
ESF = 0;

for i = 1:27
    ESF = ESF + p(i, 2)*strlength(codewordsSF(i));
end

end
```

## B q2.m

```
function [] = q2()

clear
format long
A = readmatrix("https://www.maths.cam.ac.uk/undergrad/catam/data/II-19-2-dataA.txt");
A(end, :) = [];
A = A + 1;%add one to every letter, so we work in 1= space, 2 = A, 3 = B, ... to fit matlab
    ↪ better
```

```
%disp(A)
%first need to create list of probabilities for alphabet in this text.
n = 400*25; %is the number of characters in the text

%now create a new alphabet consisting of any combination of pairs of
%letters
alphabet = zeros(27*27, 2);
m = 27*27; %size of alphabet

for i = 1:27
    for j = 1:27
        alphabet((i-1)*27 + j, :) = [i, j];
    end
end

A = A';%take A transposed so we go through entries row by row of original A
p = zeros(m, 1);
for i = 1:2:n-1
    for j = 1:m
        if A(i) == alphabet(j, 1) && A(i+1) == alphabet(j, 2)
            p(j) = p(j) + 1;
        end
    end
end

%A contains n/2 pairs
p = p./(n/2);



%rename alphabet for easier process
L = [1:1:27*27]';
p = [L p];

psorted = sortrows(p, 2, 'descend');

%we can delete rows where probability is 0, as we assume those cannot
%occur.
for i = m:-1:1
    if psorted(i, 2) == 0
        psorted(i, :) = [];
    end
end

% %with this we can work out source entropy(Actually not needed)
% h = 0;
% for i = 1:size(psorted, 1)
% h = h - psorted(i, 2) * log2(psorted(i, 2));
% end

%Now do Huffman again
codewords = strings(m, 1);

I = num2cell(psorted); %first create copy of p as cell

for i = 1:size(psorted, 1)-1
    last1 = cell2mat(I(end, 1));
    last2 = cell2mat(I(end-1, 1));

    for k = 1:size(last1, 2)
        codewords(last1(k)) = '1' + codewords(last1(k));
    end
    for l = 1 : size(last2, 2)
        codewords(last2(l)) = '0' + codewords(last2(l));
    end

    I(end-1, 1) = {[last2, last1]};
    I{end-1, 2} = I{end-1, 2} + I{end, 2};
    I(end, :) = [];
    I = sortrows(I, 2, 'descend');
end

%Now find expected length
E = 0;
for i = 1:m
    E = E + p(i, 2) * strlength(codewords(i))/2;
end

end
```