Shannon's source coding theorem gives a bound on the lowest data rate. Though, it does not provide us with the means of achieving optimal source encoding.

**Huffman Source-Coding Algorithm:**

Idea: Map frequently occurring source symbols into short codewords and the ones that appear infrequently are mapped into longer ones. By doing this, we increase the information per bit. In the limiting case, the Huffman encoder packs 1 bit of information in every bit.

Huffman code = variable length codes → symbols encoded into different number of bits

**Challenges:** (for variable length encoders)

1) **Synchronization:** There needs to be a way to break the bit stream into codewords.

2) **Instantaneous:** Decoder can decode one word after it is received — no need to wait for next symbol.

3) **Prefix condition:** no symbol is a prefix of another symbol's codeword.

**Example:**

<table>
<thead>
<tr>
<th>Letter</th>
<th>Probability</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
<th>Code 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1/2</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>
Case 2 does not satisfy prefix condition.

Case 2 is not instantaneous:
\[ a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 \]

**Example:**

\[ 1010010000000010 \]

Case 1 is self-synchronizing.

Case 2 is also self-synchronizing.

Average length of case 3:
\[ \frac{1}{2} \times 1 + \frac{1}{4} \times 2 + \frac{1}{8} \times 3 + \ldots + \frac{1}{32} \times 4 \]
\[ = \frac{31}{16} = 1.9375 \]

Entropy of the sequence:
\[ H(x) = -\left[ \frac{1}{32} \log_2 \left( \frac{1}{1/2} \right) \right. \]
\[ + \frac{1}{8} \log_2 \left( \frac{1}{1/4} \right) \]
\[ + \frac{1}{8} \log_2 \left( \frac{1}{1/8} \right) \]
\[ + \frac{1}{16} \log_2 \left( \frac{1}{1/16} \right) \]
\[ = 1.875 \]

Huffman Encoding Algorithm:
* Generate a code that satisfies the prefix condition.

* Given the smallest possible average code word.

* Uniquely decodable.

* Instantaneous.

* Not self-synchronizing.

Step 1: Sort source outputs in decreasing order of their probabilities.

Step 2: Merge the two least-probable outputs into a single output whose probability is the sum of the corresponding probabilities.

Step 3: If the number of outputs is 2, then go to step 5; otherwise, go to step 1.

Step 4: Arbitrarily assign 0 and 1 as one works for the two remaining outputs.

Step 5: If the output is the result of the merger of two outputs in a previous step, append the current code word with a 0 and a 1 to obtain the code word for the previous output and then repeat. If no output is preceded by another output in a previous step, then stop.

Example:
Average length of the code:

\[ L = 0.25 \times 2 \times 2 + 0.2 \times 2 + 0.15 \times 2 \times 3 = 2.3 \]

The entropy of the source:

\[ H(x) = -0.25 \log_2(0.25) - 0.25 \log_2(0.25) - 0.2 \log_2(0.2) \\
\quad - 0.15 \log_2(0.15) - 0.15 \log_2(0.15) = 2.28 \]

Redundancy of the code: \[ R_{HF} = \frac{2.3 - 2.28}{2.3} \approx 0.6\% \]

Compare with natural code:

\[ R_{NT} = \frac{3 - 2.28}{3} = 24\% \]

Example:

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
Symbol & \text{Code word} & Probability & Ewellin & Ewellin^2 & 3 & 4 & 5 & 6 \\
\hline
\hline
x_1 & 0 & 0.28 \quad 0.28 \quad 0.28 \quad 0.28 \\
x_2 & 11 & 0.18 \quad 0.18 \quad 0.18 \quad 0.18 \\
x_3 & 01 & 0.15 \quad 0.15 \quad 0.15 \quad 0.15 \\
\hline
\end{array} \]
Average code length:

\[ L = 0.28 \times x_2 + 0.18 \times x_2 + 0.15 \times x_3 + 0.13 \times x_3 + 0.10 \times x_3 + 0.07 \times x_4 + 0.05 \times x_5 + 0.03 \times x_5 = 2.79 \]

Entropy:

\[ H(x) = -0.28 \log_2(0.28) - 0.18 \log_2(0.18) - 0.15 \log_2(0.15) - 0.13 \log_2(0.13) - 0.10 \log_2(0.10) - 0.07 \log_2(0.07) - 0.05 \log_2(0.05) - 0.03 \log_2(0.03) = 2.75 \]

Redundancy:

\[ R = \frac{2.79 - 2.75}{2.79} = 0.0143 \text{ or } 1.43\% \]

The code is almost optimum. Every bit carries information close to 1 bit.

\[ a_1 \]
\[ a_2 \]
\[ a_3 \]
\[ a_4 \]
\[ a_5 \]

Natural code 3 bits of information

\[ \frac{8}{8} \leq 1 \]