Source Coding (Compression)

The information context of any source is given by its entropy, $H(x)$, which is the number of bits per source output.

**Example:** $H(x) = 2$ bits means that on average 2 bits are required to represent each source output symbol.

**Question:** How to design a good encoder?

**Example:**

Source entropy:

$$H(x) = - \sum P_i \log P_i = 1.75\text{bit}$$

The average number of bits of code (per symbol) is

$$R = \sum R_i P_i = 2 > H(x) = 1.75$$

Hence, the code is not efficient (more bits are used than actually needed.)

<table>
<thead>
<tr>
<th>$X_i$</th>
<th>$P_i$</th>
<th>Code word</th>
<th>$R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>1/2</td>
<td>00</td>
<td>$R_1 = 2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>1/4</td>
<td>01</td>
<td>$R_2 = 2$</td>
</tr>
<tr>
<td>$a_3$</td>
<td>1/8</td>
<td>10</td>
<td>$R_3 = 2$</td>
</tr>
<tr>
<td>$a_4$</td>
<td>1/8</td>
<td>11</td>
<td>$R_4 = 2$</td>
</tr>
</tbody>
</table>
The **source-coding theorem**: a source with entropy $H$ can be encoded with arbitrary small error probability (information loss) at any rate $R>H$. If $R<H$, error probability is bounded away from 0, regardless of encoder complexity.

The source-coding theorem gives a **sharp bound on the compression**.

**Basic idea of source coding (compression):** use fewer bits for most frequent symbols. For example, less bits should be allocated for $a_1$ and more for $a_4$. 
Source Codes

Fixed-length source outputs are mapped into variable-length binary sequences (codewords) based on the idea above. Synchronization may be a problem (i.e. the decoder must know when the next code word begins).

**Example:**

<table>
<thead>
<tr>
<th>$X_i$</th>
<th>$P_i$</th>
<th>Code 1</th>
<th>Code 2</th>
<th>Code 3</th>
<th>Code 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>1/2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>$a_2$</td>
<td>1/4</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>$a_3$</td>
<td>1/8</td>
<td>001</td>
<td>100</td>
<td>110</td>
<td>10</td>
</tr>
<tr>
<td>$a_4$</td>
<td>1/16</td>
<td>0001</td>
<td>1000</td>
<td>1110</td>
<td>11</td>
</tr>
<tr>
<td>$a_5$</td>
<td>1/16</td>
<td>00001</td>
<td>10000</td>
<td>1111</td>
<td>110</td>
</tr>
</tbody>
</table>
Source Codes

• Code 1: self-synchronizing (each word ends with 1).
• Code 2: self-synchronizing, but not instantaneous. Both of them are uniquely decodable.

• Code 4 is not uniquely decodable. Example: 110110 can be decoded as \(a_5a_5\) or \(a_4a_2a_3\). It should never be used in practice.

The prefix condition: no any code word is a prefix of another code word. Codes 1 & 3 satisfy it.

Any code is uniquely decodable & instantaneous if and only if it meets the prefix condition.
Good code meets prefix condition (code 1 & 3), and has the smallest average word length. For code 1,

\[ R = \sum_{i} R_i P_i = \frac{31}{16} \]

and R=30/16 for code 3. Hence code 3 is the best one: uniquely decodable & instantaneous (prefix condition), and has the least average word length.
**Huffman Encoding Algorithm**

Basic idea of Huffman algorithm: if we can encode each source output of $P_i$ with $\log(1/P_i)$ bits, then

$$R = \sum_i P_i \log 1/P_i = H$$

i.e. the least possible average length (from the source coding theorem).

The Huffman codes are optimum: among all the codes that satisfy the prefix conditions, they have the minimum average length.
Huffman Encoding Algorithm

1. Sort source outputs in decreasing order of probabilities.
2. Merge the 2 least-probable outputs into one (its probability is the sum of indiv. prob.)
3. If the number of remaining outputs is 2, go to step 4, otherwise go to step 1
4. Assign 0 and 1 as code words for the 2 outputs
5. If an output is a merger of 2 outputs in the proceeding step, append 0 and 1 to the code word
6. Repeat 5 until there are no merged outputs
Huffman Algorithm Flowchart

Start

Sort in decreasing probability order

Merge 2 least prob. outputs

Number of outputs=2?

Yes

Assign 0 or 1 to the two outputs

No

Is any output a merge of two

Yes

Append the code word with 0 or 1

No

Stop
**Design example:**

The tree diagram

The average length of a Huffman code, \( R = \sum_i P(x_i)l(x_i) \) satisfies to the following: \( H(x) \leq R < H(x) + 1 \)

If the code is designed for blocks of \( n \) symbols (rather than single symbols), then

\[
H(x^n) \leq nR < H(x^n) + 1
\]

where \( R^n = nR \), and \( H(x) = H(x^n)/n \). Note that \( R \to H \) as \( n \to \infty \) . This proves the optimality of the Huffman algorithm.
Summary

- Source coding (compression).
- The source coding theorem.
- Huffman code & algorithm.

**Homework**: Reading, Proakis and Salehi (2nd ed.), 6.2, 6.3. Study carefully all the examples, make sure you understand them and can solve with the book closed.