Two famous prefix codes
① Shannon - Fano code
② Huffman code

ECS452 2014/1 Part I.2 Dr.Prapun

2.29. For our present purposes, a better code is one that is uniquely decodable and has a shorter expected length than another uniquely decodable code. We do not consider other issues of encoding/decoding complexity or of the relative advantages of block codes or variable length codes. [3, p 57]

2.2 Optimal Source Coding: Huffman Coding

In this section we describe a very popular source coding algorithm called the Huffman coding.

Definition 2.30. Given a source with known probabilities of occurrence for symbols in its alphabet, to construct a binary Huffman code, create a binary tree by repeatedly combining[7] the probabilities of the two least likely symbols.

steps ① combine two least likely symbols
② Repeat ① until you can not do it anymore.

- Developed by David Huffman as part of a class assignment[8].

[7] The Huffman algorithm performs *repeated source reduction* [3, p 63]:

- At each step, two source symbols are combined into a new symbol, having a probability that is the sum of the probabilities of the two symbols being replaced, and the new reduced source now has one fewer symbol.
- At each step, the two symbols to combine into a new symbol have the two lowest probabilities.
  - If there are more than two such symbols, select any two.

[8] The class was the first ever in the area of information theory and was taught by Robert Fano at MIT in 1951.

- Huffman wrote a term paper in lieu of taking a final examination.
- It should be noted that in the late 1940s, Fano himself (and independently, also Claude Shannon) had developed a similar, but suboptimal, algorithm known today as the ShannonFano method. The difference between the two algorithms is that the ShannonFano code tree is built from the top down, while the Huffman code tree is constructed from the bottom up.

- By construction, Huffman code is a prefix code.

Example 2.31.

x	$p_X(x)$	Codeword $c(x)$	$\ell(x)$
1	0.5	0	1
2	0.25	10	2
3	0.125	110	3
4	0.125	111	3

$\mathbb{E}[\ell(X)] = \frac{1}{2}\times 1 + \frac{1}{4}\times 2 + \frac{1}{8}\times 3 + \frac{1}{8}\times 3 = 1\frac{3}{4} = \frac{7}{4} = 1.75$ bits/symbol

Note that for this particular example, the values of $2^{\ell(x)}$ from the Huffman encoding is inversely proportional to $p_X(x)$:

$$p_X(x) = \frac{1}{2^{\ell(x)}}.$$

Recall,

$$\mathbb{E}[g(X)] = \sum_x p_X(x) g(x)$$

$$g(x) = -\log_2 p_X(x)$$

In other words,

$$\ell(x) = \log_2 \frac{1}{p_X(x)} = -\log_2(p_X(x)).$$

Therefore,

$$\mathbb{E}[\ell(X)] = \sum_x p_X(x)\ell(x) = \underbrace{\sum_x p_X(x)\left(-\log_2 p_X(x)\right) = \mathbb{E}\left[-\log_2 p_X(X)\right]}_{\equiv H(X) \leftarrow \text{Entropy of RV } X}$$

Example 2.32.

x	$p_X(x)$	Codeword $c(x)$	$\ell(x)$
'a'	0.4	0	1
'b'	0.3	10	2
'c'	0.1	110	3
'd'	0.1	1110	4
'e'	0.06	11110	5
'f'	0.04	11111	5

$$\mathbb{E}[\ell(X)] = 0.4\times 1 + 0.3\times 2 + 0.1\times 3 + 0.1\times 4 + 0.06\times 5 + 0.04\times 5$$
$$= 0.4 + 0.6 + 0.3 + 0.4 + 0.5 = 2.2 \text{ bits/symbol}$$

$H(X) = 2.1435 < \mathbb{E}[\ell(X)]$

Example 2.33.

x	$p_X(x)$	Codeword $c(x)$	$\ell(x)$
1	0.25	10	2
2	0.25	00	2
3	0.2	01	2
4	0.15	110	3
5	0.15	111	3

$\mathbb{E}[\ell(X)] = 2.3$ bits/symbol

Example 2.34.

x	$p_X(x)$	Codeword $c(x)$	$\ell(x)$
1	1/3	00	2
2	1/3	01	2
3	1/4	10	2
4	1/12	11	2

$\mathbb{E}[\ell(X)] = 2$ bits/symbol

x	$p_X(x)$	Codeword $c(x)$	$\ell(x)$
1	1/3	0	1
2	1/3	10	2
3	1/4	110	3
4	1/12	111	3

$\mathbb{E}[\ell(X)] = \frac{1}{3}\times 1 + \frac{1}{3}\times 2 + \frac{1}{4}\times 3 + \frac{1}{12}\times 3 = 2$ bits/symbol

2.35. The set of codeword lengths for Huffman encoding is not unique. There may be more than one set of lengths but all of them will give the same value of expected length.

Definition 2.36. A code is **optimal** for a given source (with known pmf) if it is uniquely decodable and its corresponding expected length is the shortest among all possible uniquely decodable codes for that source.

2.37. The Huffman code is optimal.

2.3 Source Extension (Extension Coding)

2.38. One can usually (not always) do better in terms of expected length (per source symbol) by encoding blocks of several source symbols.

Definition 2.39. In, an ***n*-th extension** coding, n successive source symbols are grouped into blocks and the encoder operates on the blocks rather than on individual symbols. [1, p. 777]

Example 2.40.

x	$p_X(x)$	Codeword $c(x)$	$\ell(x)$
Y(es)	0.9	0	1
N(o)	0.1	1	1

(a) First-order extension:

$\mathbb{E}[\ell(X)] = 1$ bit/symbol $= L_1$

0 1 1 0 0 0 1 0 0 1 1 1

YNNYYYNYYNNN...

10 110 0 110 10 111

(b) Second-order Extension:

x_1x_2	$p_{X_1,X_2}(x_1,x_2)$	$c(x_1,x_2)$	$\ell(x_1,x_2)$
YY	$0.9 \times 0.9 = 0.81$	0	1
YN	$0.9 \times 0.1 = 0.09$	10	2
NY	$0.1 \times 0.9 = 0.09$	110	3
NN	$0.1 \times 0.1 = 0.01$	111	3

$\mathbb{E}[\ell(X_1,X_2)] = 0.81 \times 1 + 0.09 \times 2 + 0.09 \times 3 + 0.01 \times 3 = 1.29$ bits/2 symbols

$L_2 = 0.645$ bits/symbol

(c) Third-order Extension:

$x_1x_2x_3$	$p_{X_1,X_2,X_3}(x_1,x_2,x_3)$	$c(x_1,x_2,x_3)$	$\ell(x_1,x_2,x_3)$
YYY	$0.9 \times 0.9 \times 0.9 = 0.729$		
YYN	$0.9 \times 0.9 \times 0.1 = 0.081$		
YNY	$0.9 \times 0.1 \times 0.9 = 0.081$		
$\vdots$			

$\mathbb{E}[\ell(X_1,X_2,X_3)] = 1.5980$ bits/3 symbols

$L_3 = 0.5327$ bits/symbol