Chapter Three
Source Coding

1- Sampling theorem:

Sampling of the signals is the fundamental operation in digital communication. A continuous time signal is first converted to discrete time signal by sampling process. Also it should be possible to recover or reconstruct the signal completely from its samples.

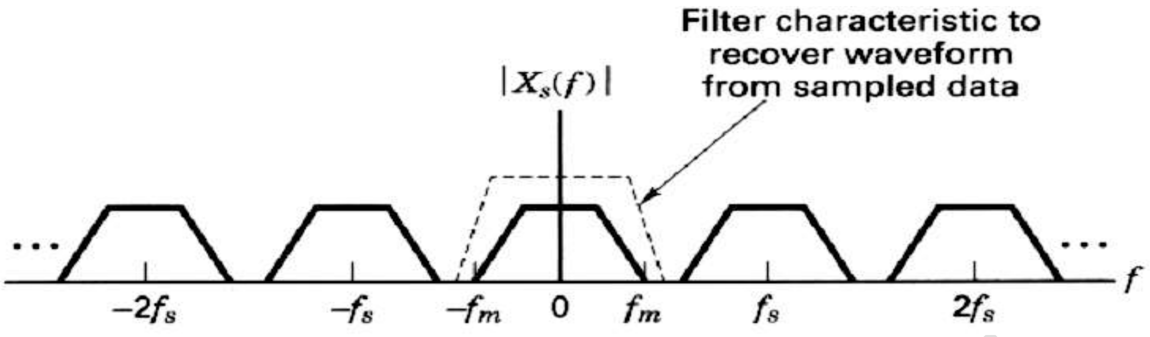
The sampling theorem state that:

i- *A band limited signal of finite energy, which has no frequency components higher than W Hz, is completely described by specifying the values of the signal at instant of time separated by 1/2W second and*

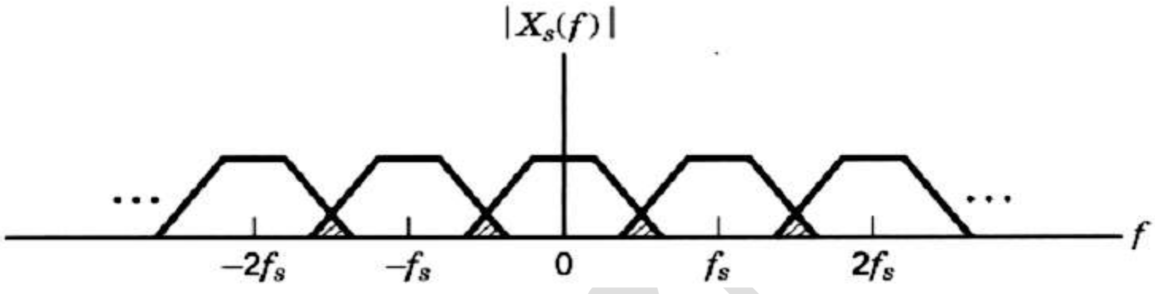
ii- *A band limited signal of finite energy, which has no frequency components higher than W Hz, may be completely recovered from the knowledge of its samples taken at the rate of 2W samples per second.*

When the sampling rate is chosen $f_s = 2f_m$ each spectral replicate is separated from each of its neighbors by a frequency band exactly equal to f_s hertz, and the analog waveform ca theoretically be completely recovered from the samples, by the use of filtering. It should be clear that if $f_s > 2f_m$, the replications will be move farther apart in frequency making it easier to perform the filtering operation.

When the sampling rate is reduced, such that $f_s < 2f_m$, the replications will overlap, as shown in figure below, and some information will be lost. This phenomenon is called aliasing.



Sampled spectrum $f_s > 2f_m$



Sampled spectrum $f_s < 2f_m$

A bandlimited signal having no spectral components above f_m hertz can be determined uniquely by values sampled at uniform intervals of $T_s \leq \frac{1}{2f_m} sec.$

The sampling rate is $f_s = \frac{1}{T_s}$

So that $f_s \geq 2f_m$. The sampling rate $f_s = 2f_m$ is called Nyquist rate.

Example: Find the Nyquist rate and Nyquist interval for the following signals.

i- $m(t) = \frac{\sin(500\pi t)}{\pi t}$

ii- $m(t) = \frac{1}{2\pi}\cos(4000\pi t)\cos(1000\pi t)$

Solution:

i- $wt = 500\pi t \quad \therefore 2\pi f = 500\pi \quad \rightarrow f = 250Hz$

Nyquist interval $= \frac{1}{2f_{max}} = \frac{1}{2\times250} = 2\ msec.$

Nyquist rate $=2f_{max} = 2 \times 250 = 500Hz$

ii- $m(t) = \frac{1}{2\pi}\left[\frac{1}{2}\{\cos(4000\pi t - 1000\pi t) + \cos(4000\pi t + 1000\pi t)\}\right]$

$$= \frac{1}{4\pi}\{\cos(3000\pi t) + \cos(5000\pi t)\}$$

Then the highest frequency is 2500Hz

Nyquist interval $= \frac{1}{2f_{max}} = \frac{1}{2\times 2500} = 0.2\, msec.$

Nyquist rate $=2f_{max} = 2 \times 2500 = 5000Hz$

H. W:

Find the Nyquist interval and Nyquist rate for the following:

i- $\frac{1}{2\pi}\cos(400\pi t).\cos(200\pi t)$

ii- $\frac{1}{\pi}sin\pi t$

Example:

A waveform [20+20sin(500t+30^o)] is to be sampled periodically and reproduced from these sample values. Find maximum allowable time interval between sample values, how many sample values are needed to be stored in order to reproduce 1 sec of this waveform?.

Solution:

$$x(t) = 20 + 20\sin(500t + 30^0)$$

$$w = 500 \rightarrow 2\pi f = 500 \rightarrow f = 79.58\ Hz$$

Minimum sampling rate will be twice of the signal frequency:

$$f_{s(\min)} = 2 \times 79.58 = 159.15\ Hz$$

$$T_{s(max)} = \frac{1}{f_{s(\min)}} = \frac{1}{159.15} = 6.283\ msec.$$

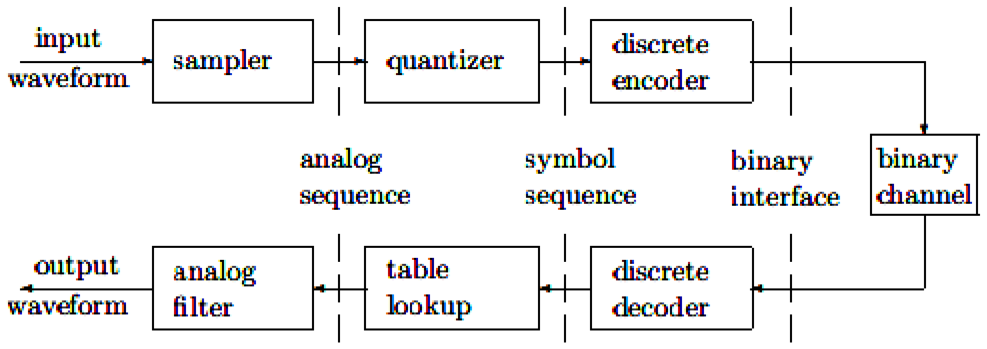
Number of sample in $1sec = \frac{1}{6.283msec} = 159.16 \approx 160\ sample$

2- Source coding:

An important problem in communications is the efficient representation of data generated by a discrete source. The process by which this representation is accomplished is called source encoding. An efficient source encoder must satisfies two functional requirements:

i- The code words produced by the encoder are in binary form.

ii- The source code is uniquely decodable, so that the original source sequence can be reconstructed perfectly from the encoded binary sequence.



The entropy for a source with statistically independent symbols:

$$H(Y) = -\sum_{j=1}^{m} P(y_j)\log_2 P(y_j)$$

We have:

$$\max[H(Y)] = log_2 m$$

A code efficiency can therefore be defined as:

$$\eta = \frac{H(Y)}{\max[H(Y)]} \times 100$$

The overall code length, L, can be defined as the average code word length:

$$L = \sum_{j=1}^{m} P(x_j) l_j \qquad bits/symbol$$

The code efficiency can be found by:

$$\eta = \frac{H(Y)}{\text{L}} \times 100$$

Not that $\max[H(Y)] \quad bits/symbol = L \quad bits/codeword$

i- Fixed- Length Code Words:

If the alphabet X consists of the 7 symbols {a, b, c, d, e, f, g}, then the following fixed-length code of block length L = 3 could be used.

C(a) = 000
C(b) = 001
C(c) = 010
C(d) = 011
C(e) = 100
C(f) = 101
C(g) = 110.

The encoded output contains L bits per source symbol. For the above example the source sequence *bad*... would be encoded into 001000011... . Note that the output bits are simply run together (or, more technically, concatenated). This method is nonprobabilistic; it takes no account of whether some symbols occur more frequently than others, and it works robustly regardless of the symbol frequencies.

This is used when the source produces almost equiprobable messages $p(x_1) \cong p(x_2) \cong p(x_3) \cong \ldots \cong p(x_n)$, then $l_1 = l_2 = l_3 = \ldots = l_n = L_C$ and for binary coding then:

1- $L_C = \log_2 n$ bit/message if $n = 2^r$ ($n = 2,4,8,16,\ldots$ and r is an integer) which gives $\eta = 100\%$

2- $L_C = Int[\log_2 n] + 1$ bits/message if $n \neq 2^r$ which gives less efficiency

Example

For ten equiprobable messages coded in a fixed length code then

$p(x_i)=\frac{1}{10}$ and $L_C=Int[\log_2 10]+1=4$ bits

and $\eta=\frac{H(X)}{L_C}\times 100\%=\frac{\log_2 10}{4}\times 100\%=83.048\%$

Example: For eight equiprobable messages coded in a fixed length code then

$p(x_i)=\frac{1}{8}$ and $L_C=\log_2 8=3$ bits and $\eta=\frac{3}{3}\times 100\%=100\%$

Example: Find the efficiency of a fixed length code used to encode messages obtained from throwing a fair die (a) once, (b) twice, (c) 3 times.

Solution

a- For a fair die, the messages obtained from it are equiprobable with a probability of $p(x_i)=\frac{1}{6}$ with $n=6$.

$L_C=Int[\log_2 6]+1=3$ bits/message

$\eta=\frac{H(X)}{L_C}\times 100\%=\frac{\log_2 6}{3}\times 100\%=86.165\%$

b- For two throws then the possible messages are $n=6\times 6=36$ messages with equal probabilities

$L_C=Int[\log_2 36]+1=6$ bits/message $=6$ bits/2-symbols

while $H(X)=\log_2 6$ bits/symbol $\eta=\frac{2\times H(X)}{L_C}\times 100\%=86.165\%$

c- For three throws then the possible messages are $n=6\times 6\times 6=216$ with equal probabilities

$L_C=Int[\log_2 216]+1=8$ bits/message $=8$ bits/3-symbols

while $H(X)=\log_2 6$ bits/symbol $\eta=\frac{3\times H(X)}{L_C}\times 100\%=96.936\%$

ii- Variable-Length Code Words

When the source symbols are not equally probable, a more efficient encoding method is to use variable-length code words. For example, a variable-length code for the alphabet X = {a, b, c} and its lengths might be given by

C(a)= 0	*l(a)=1*
C(b)= 10	*l(b)=2*
C(c)= 11	*l(c)=2*

The major property that is usually required from any variable-length code is that of unique decodability. For example, the above code C for the alphabet X = {a, b, c} is soon shown to be uniquely decodable. However such code is not uniquely decodable, even though the codewords are all different. If the source decoder observes 01, it cannot determine whether the source emitted (a b) or (c).

Prefix-free codes: A **prefix code** is a type of code system (typically a variable-length code) distinguished by its possession of the "prefix property", which requires that there is no code word in the system that is a prefix (initial segment) of any other code word in the system. For example:

$$\{a = 0, b = 110, c = 10, d = 111\}\ is\ a\ prefix\ code.$$

When message probabilities are not equal, then we use variable length codes. The following properties need to be considered when attempting to use variable length codes:

1) Unique decoding:

<u>Example</u>

Consider a 4 alphabet symbols with symbols represented by binary digits as follows:

$A = 0$

$B = 01$

$C = 11$

$D = 00$

If we receive the code word 0011 it is not known whether the transmission was DC or AAC. This example is not, therefore, uniquely decodable.

2) Instantaneous decoding:

Example

Consider a 4 alphabet symbols with symbols represented by binary digits as follows:

$A = 0$

$B = 10$

$C = 110$

$D = 111$

This code can be instantaneously decoded since no complete codeword is a prefix of a larger codeword. This is in contrast to the previous example where A is a prefix of both B and D. This example is also a 'comma code' as the symbol zero indicates the end of a codeword except for the all ones word whose length is known.

Example

Consider a 4 alphabet symbols with symbols represented by binary digits as follows:

$A = 0$

$B = 01$

$C = 011$

$D = 111$

The code is identical to the previous example but the bits are time reversed. It is still uniquely decodable but no longer instantaneous, since early codewords are now prefixes of later ones.

Shannon Code

For messages $x_1,\ x_2,\ x_3, \dots x_n$ with probabilities $p(x_1),\ p(x_2),\ p(x_3), \dots p(x_n)$ then:

1) $l_i = -\log_2 p(x_i)$ if $p(x_i) = \left(\frac{1}{2}\right)^r$ $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots\}$

2) $l_i = Int[-\log_2 p(x_i)] + 1$ if $p(x_i) \neq \left(\frac{1}{2}\right)^r$

Also define $F_i = \sum_{k=1}^{i-1} p(x_k)$ $1 \geq \omega_i \geq 0$

then the codeword of x_i is the binary equivalent of F_i consisting of l_i bits.

$$C_i = (F_i)_2^{l_i}$$

where C_i is the binary equivalent of F_i up to l_i bits. In encoding, messages must be arranged in a decreasing order of probabilities.

Example

Develop the Shannon code for the following set of messages,

$$p(x) = [0.3 \quad 0.2 \quad 0.15 \quad 0.12 \quad 0.1 \quad 0.08 \quad 0.05]$$

then find:

(a) Code efficiency,
(b) $p(0)$ at the encoder output.

Solution

x_i	$p(x_i)$	l_i	F_i	C_i	0_i
x_1	0.3	2	0	00	2
x_2	0.2	3	0.3	010	2
x_3	0.15	3	0.5	100	2
x_4	0.12	4	0.65	1010	2
x_5	0.10	4	0.77	1100	2
x_6	0.08	4	0.87	1101	1
x_7	0.05	5	0.95	11110	1

To find C_1		To find C_2		To find C_3	
$0\times 2=0$	0 $\downarrow$	$0.3\times 2=0.6$	0	$0.5\times 2=1.0$	1
		$0.6\times 2=1.2$	1 $\downarrow$	$0.0\times 2=0.0$	0 $\downarrow$

To find C_4		To find C_5	
$0.65\times 2=1.3$	1	$0.77\times 2=1.54$	1
$0.30\times 2=0.6$	0	$0.54\times 2=1.08$	1
$0.60\times 2=1.2$	1	$0.08\times 2=0.16$	0
$0.20\times 2=0.4$	0 $\downarrow$	$0.16\times 2=0.32$	0 $\downarrow$

(a) To find the code efficiency, we have

$$L_C=\sum_{i=1}^{7} l_i\, p(x_i)=3.1 \text{ bits/message.}$$

$$H(X)=-\sum_{i=1}^{7} p(x_i)\log_2 p(x_i)=2.6029 \text{ bits/message.}$$

$$\eta=\frac{H(X)}{L_C}\times 100\%=83.965\%$$

(b) $p(0)$ **at the encoder output is**

$$p(0)=\frac{\sum_{i=1}^{7} 0_i\, p(x_i)}{L_C}=\frac{0.6+0.4+0.3+0.24+0.2+0.08+0.05}{3.1}$$

$$p(0)=0.603$$

Example

Repeat the previous example using ternary coding.

Solution

1) $l_i=-\log_3 p(x_i)$ if $p(x_i)=\left(\frac{1}{3}\right)^r$ $\{\frac{1}{3},\frac{1}{9},\frac{1}{27},\ldots\}$

2) $l_i = Int[-\log_3 p(x_i)]+1$ if $p(x_i) \neq \left(\frac{1}{3}\right)^r$ and $C_i = (F_i)_3^{l_i}$

x_i	$p(x_i)$	l_i	F_i	C_i	0_i
x_1	0.3	2	0	00	2
x_2	0.2	2	0.3	02	1
x_3	0.15	2	0.5	11	0
x_4	0.12	2	0.65	12	0
x_5	0.10	3	0.77	202	1
x_6	0.08	3	0.87	212	0
x_7	0.05	3	0.95	221	0

To find C_1

$0\times3=0$ 0 $\downarrow$

To find C_2

$0.3\times3=0.9$ 0 $\downarrow$

To find C_3

$0.5\times3=1.5$ 1 $\downarrow$

To find C_4

$0.65\times3=1.95$ 1 $\downarrow$

$0.95\times3=2.85$ 2

To find C_5

$0.77\times3=2.31$ 2 $\downarrow$

$0.31\times3=0.93$ 0

(a) To find the code efficiency, we have

$$L_C = \sum_{i=1}^{7} l_i p(x_i) = 2.23 \text{ ternary unit/message.}$$

$$H(X) = -\sum_{i=1}^{7} p(x_i)\log_3 p(x_i) = 1.642 \text{ ternary unit/message.}$$

$$\eta = \frac{H(X)}{L_C}\times100\% = 73.632\%$$

(b) $p(0)$ at the encoder output is

$$p(0) = \frac{\sum_{i=1}^{7} 0_i \, p(x_i)}{L_C} = \frac{0.6 + 0.2 + 0.1}{2.23}$$

$$p(0) = 0.404$$

Shannon- Fano Code:

In Shannon–Fano coding, the symbols are arranged in order from most probable to least probable, and then divided into two sets whose total probabilities are as close as possible to being equal. All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1". As long as any sets with more than one member remain, the same process is repeated on those sets, to determine successive digits of their codes.

Example:

The five symbols which have the following frequency and probabilities, design suitable Shannon-Fano binary code. Calculate average code length, source entropy and efficiency.

Symbol	count	Probabilities	Binary codes	Length
A	15	0.385	00	2
B	7	0.1795	01	2
C	6	0.154	10	2
D	6	0.154	110	3
E	5	0.128	111	3

The average code word length:

$$L = \sum_{j=1}^{m} P(x_j) l_j$$

$$L = 2 \times 0.385 + 2 \times 0.1793 + 2 \times 0.154 + 3 \times 0.154 + 3 \times 0.128$$
$$= 2.28\ bits/symbol$$

The source entropy is:

$$H(Y) = -\sum_{j=1}^{m} P(y_j) \log_2 P(y_j)$$

$$H(Y) = -[0.385ln0.385 + 0.1793ln0.1793 + 2 \times 0.154ln0.154 + 0.128l0.128]/ln2$$

$$H(Y) = 2.18567\ bits/symbol$$

The code efficiency:

$$\eta = \frac{H(Y)}{\mathrm{L}} \times 100 = \frac{2.18567}{2.28} \times 100 = 95.86\%$$

Example

Develop the Shannon - Fano code for the following set of messages, $p(x) = [0.35 \quad 0.2 \quad 0.15 \quad 0.12 \quad 0.1 \quad 0.08]$ then find the code efficiency.

Solution

x_i	$p(x_i)$	Code			l_i
x_1	0.35	0	0		2
x_2	0.2	0	1		2
x_3	0.15	1	0	0	3
x_4	0.12	1	0	1	3
x_5	0.10	1	1	0	3
x_6	0.08	1	1	1	3

$$L_C = \sum_{i=1}^{6} l_i\, p(x_i) = 2.45 \text{ bits/symbol}$$

$$H(X) = -\sum_{i=1}^{6} p(x_i) \log_2 p(x_i) = 2.396 \text{ bits/symbol}$$

$$\eta = \frac{H(X)}{L_C} \times 100\% = 97.796\%$$

Example

Repeat the previous example using with $r = 3$

Solution

x_i	$p(x_i)$	Code		l_i
x_1	0.35	0		1
x_2	0.2	1	0	2
x_3	0.15	1	1	2
x_4	0.12	2	0	2
x_5	0.10	2	1	2
x_6	0.08	2	2	2

$$L_C = \sum_{i=1}^{6} l_i\, p(x_i) = 1.65 \quad \text{ternary unit/symbol}$$

$$H(X) = -\sum_{i=1}^{6} p(x_i) \log_3 p(x_i) = 1.512 \text{ ternary unit/symbol}$$

$$\eta = \frac{H(X)}{L_C} \times 100\% = 91.636\%$$

Huffman Code

The Huffman coding algorithm comprises two steps, reduction and splitting. These steps can be summarized as follows:

1) Reduction

a) List the symbols in descending order of probability.

b) Reduce the r least probable symbols to one symbol with a probability equal to their combined probability.

c) Reorder in descending order of probability at each stage.

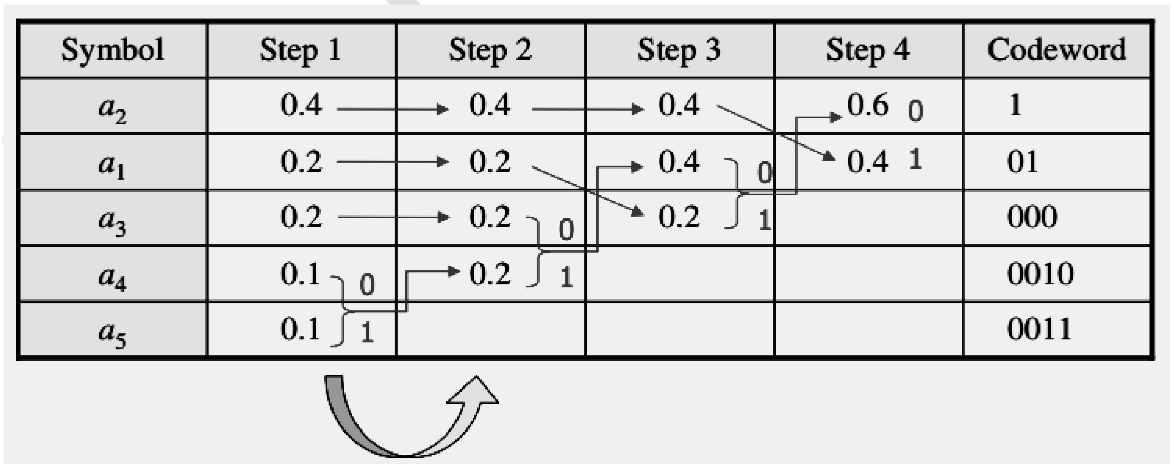
d) Repeat the reduction step until only two symbols remain.

2) Splitting

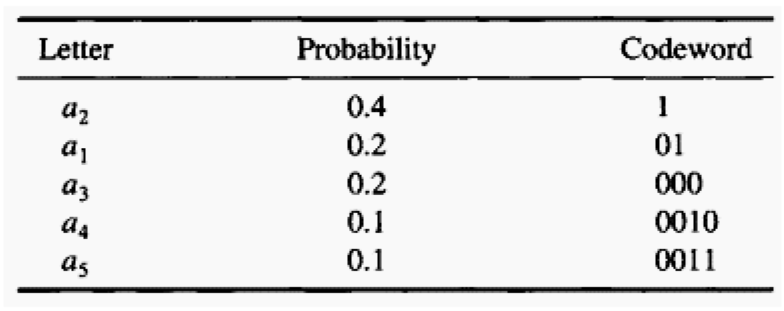
a) Assign $0,1,\ldots r$ to the r final symbols and work backwards.

b) Expand or lengthen the code to cope with each successive split.

Example: Design Huffman codes for $A = \{a_1, a_2, \ldots \ldots . a_5\}$, having the probabilities $\{0.2, 0.4, 0.2, 0.1, 0.1\}$.



Symbol	Step 1	Step 2	Step 3	Step 4	Codeword
a_2	0.4	0.4	0.4	0.6 0	1
a_1	0.2	0.2	0.4 0	0.4 1	01
a_3	0.2	0.2 0	0.2 1		000
a_4	0.1 0	0.2 1			0010
a_5	0.1 1				0011



Letter	Probability	Codeword
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011

The average code word length:

$$L = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = 2.2\ bits/symbol$$

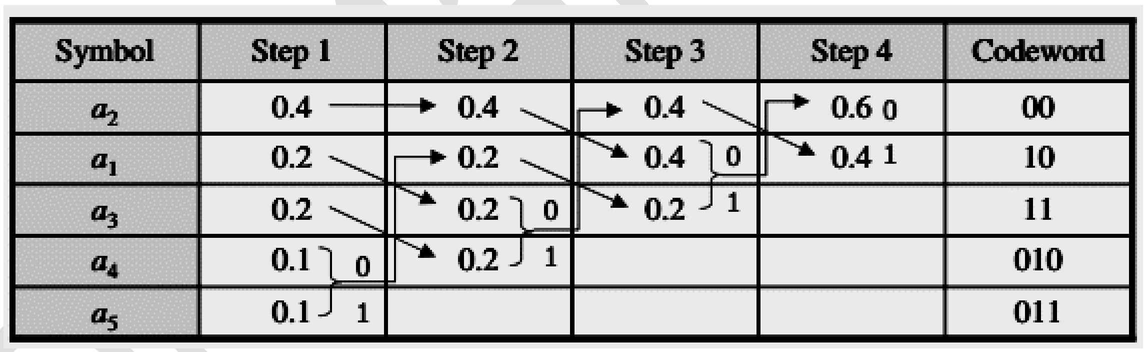
The source entropy:

$$H(Y) = -[0.4ln0.4 + 2 \times 0.2ln0.2 + 2 \times 0.1ln0.1]/ln2 = 2.12193 \text{ bits/symbol}$$

The code efficiency:

$$\eta = \frac{2.12193}{2.2} \times 100 = 96.45\%$$

It can be design Huffman codes with minimum variance:



Symbol	Step 1	Step 2	Step 3	Step 4	Codeword
a_2	0.4	0.4	0.4	0.6 0	00
a_1	0.2	0.2	0.4 0	0.4 1	10
a_3	0.2	0.2 0	0.2 1		11
a_4	0.1 0	0.2 1			010
a_5	0.1 1				011

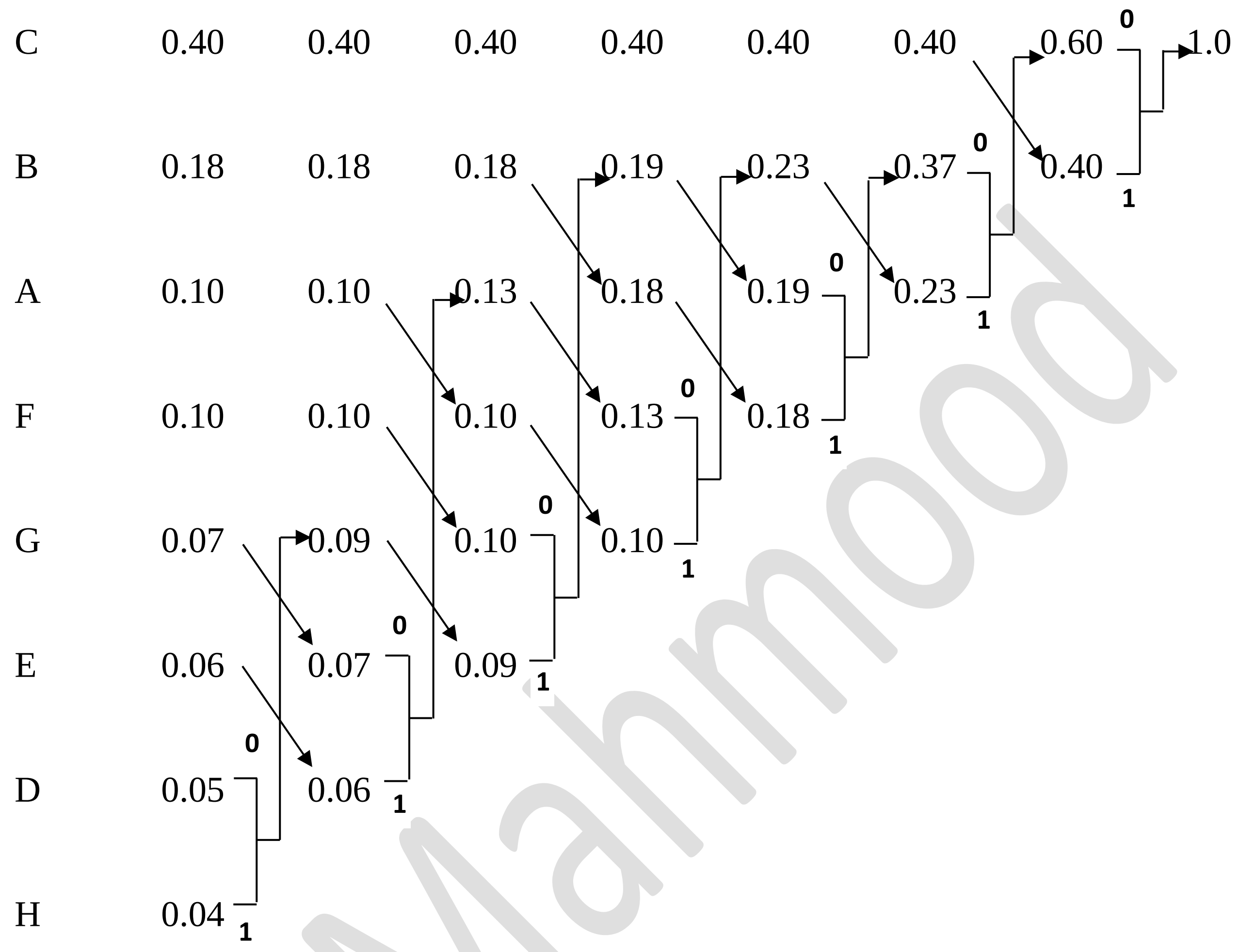
The average code word length is still 2.2 bits/symbol. But variances are different!

Example

Develop the Huffman code for the following set of symbols

Symbol	A	B	C	D	E	F	G	H
Probability	0.1	0.18	0.4	0.05	0.06	0.1	0.07	0.04

Solution


C 0.40 0.40 0.40 0.40 0.40 0.40 0.60 1.0
B 0.18 0.18 0.18 0.19 0.23 0.37 0.40
A 0.10 0.10 0.13 0.18 0.19 0.23
F 0.10 0.10 0.10 0.13 0.18
G 0.07 0.09 0.10 0.10
E 0.06 0.07 0.09
D 0.05 0.06
H 0.04
0
1

So we obtain the following codes

Symbol	A	B	C	D	E	F	G	H
Probability	0.1	0.18	0.4	0.05	0.06	0.1	0.07	0.04
Codeword	011	001	1	00010	0101	0000	0100	00011
l_i	3	3	1	5	4	4	4	5

$$H(X) = -\sum_{i=1}^{8} p(x_i)\log_2 p(x_i) = 2.552 \text{ bits/symbol}$$

$$L_C = \sum_{i=1}^{8} l_i\, p(x_i) = 2.61 \text{ bits/symbol}$$

$$\eta = \frac{H(X)}{L_C} \times 100\% = 97.778\%$$

Data Compression:

In computer science and information theory, data compression, source coding, or bit-rate reduction involves encoding information using fewer bits than the original representation. Compression can be either lossy or lossless.

Lossless data compression algorithms usually exploit statistical redundancy to represent data more concisely without losing information, so that the process is reversible. Lossless compression is possible because most real-world data has statistical redundancy. For example, an image may have areas of color that do not change over several pixels.

Lossy data compression is the converse of lossless data compression. In these schemes, some loss of information is acceptable. Dropping nonessential detail from the data source can save storage space. There is a corresponding trade-off between preserving information and reducing size.

Run-Length Encoding (RLE):

Run-Length Encoding is a very simple lossless data compression technique that replaces runs of two or more of the same character with a number which represents the length of the run, followed by the original character; single characters are coded as runs of 1. RLE is useful for highly-redundant data, indexed images with many pixels of the same color in a row.

Example:

Input: AAABBCCCCDEEEEEEAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA

Output: 3A2B4C1D6E38A

The input message to RLE encoder is a variable while the output code word is fixed, unlike Huffman code where the input is fixed while the output is varied.

<u>Example :</u> Consider these repeated pixels values in an image … 0 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5 0 0 0 0 0 0 0 0 We could represent them more efficiently as (12, 0)(4, 5)(8, 0)

24 bytes reduced to 6 which gives a compression ratio of 24/6 = 4:1.

<u>Example :</u>Original Sequence (1 Row): 111122233333311112222 can be encoded as: (4,1),(3,2),(6,3),(4,1),(4,2). 21 bytes reduced to 10 gives a compression ratio of 21/10 = 21:10.

<u>Example :</u> Original Sequence (1 Row): – HHHHHHHUFFFFFFFFFFFFFF can be encoded as: (7,H),(1,U),(14,F) . 22 bytes reduced to 6 gives a compression ratio of 22/6 = 11:3 .

<u>Savings Ratio :</u> the savings ratio is related to the compression ratio and is a measure of the amount of redundancy between two representations (compressed and uncompressed). Let:

N1 = the total number of bytes required to store an uncompressed (raw) source image.

N2 = the total number of bytes required to store the compressed data.

The compression ratio Cr is then defined as:

$$C_r = \frac{N_1}{N_2}$$

- Larger compression ratios indicate more effective compression
- Smaller compression ratios indicate less effective compression
- Compression ratios less than one indicate that the uncompressed representation has high degree of irregularity.

The saving ratio Sr is then defined as :

$$S_r = \frac{(N_1 - N_2)}{N_1}$$

- Higher saving ratio indicate more effective compression while negative ratios are possible and indicate that the compressed image has larger memory size than the original.

Example: a 5 Megabyte image is compressed into a 1 Megabyte image, the savings ratio is defined as (5-1)/5 or 4/5 or 80%.

This ratio indicates that 80% of the uncompressed data has been eliminated in the compressed encoding.