University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

Coding Techniques
تقنيات الترميز

Assist. Prof. Dr. Nuha Jameel
أ.م.د. نهى جميل

cs.uotechnology.edu.iq

Principles of Information theory

• Introduction

Data compression addresses the problem of reducing the amount of data required to represent a digital file, so that it can be stored or transmitted so efficiently. The principle of data compression is that, it compress data by removing redundancy from the original data in the source file.

On the other hand, information theory tells us that the amount of information conveyed by an event relates to its probability of occurrence. An event that is less likely to occur is said to contain more information than an event that is more likely to occur. The problem of representing the source alphabet symbols (usually the binary system consisting of the two symbols 0 & 1) is the main topic of **coding theory**.

An optimum coding scheme will use more bits for the symbols that less likely to occur, and a fewer bits for the symbols that frequently occur. Coding theory leads to information theory and ***information theory*** provides the performance limits on what can be done by suitable encoding of the information. Thus the two theories are intimately related. Both ***coding and information*** theory give a central role to errors ***(noise)*** and are therefore of special interest since in real-life noise is everywhere.

The conventional system is modeled by:

1- An information source.
2- An encoding of this source.
3- A channel over or through which the information is sent.
4- A noise (error) source that is added to the signal in the channel.
5- A decoding and recovery of the original information from the received signal with noise.
6- A destination for the information.

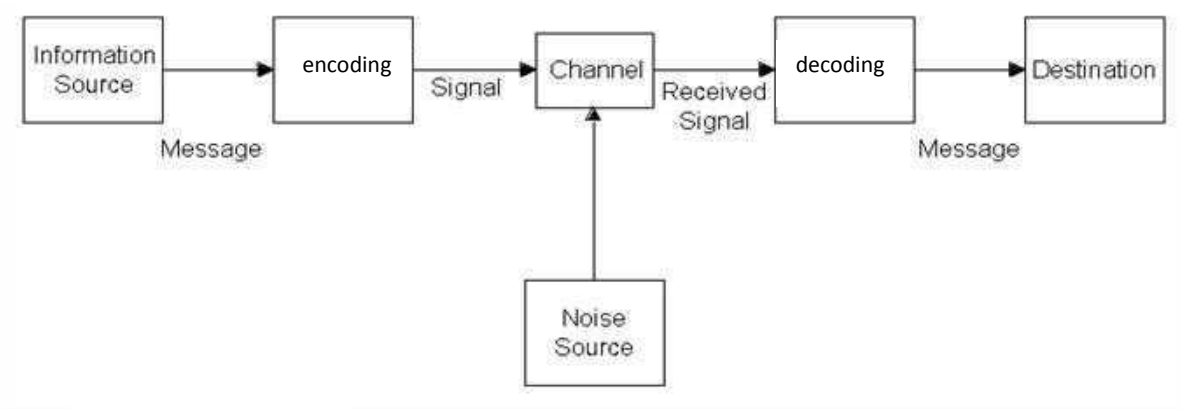


FIGURE 1 communication system

- **Information Theory**

Information theory tells us that the amount of information conveyed by an event relates to its probability of occurrence. An event that is less likely to occur is said to contain more information than an event that is more likely to occur. The amount of information of an event and its probability are thus opposite.

Information Theory is a mathematical subject dealing with ***three basic concepts:***

1- The measure of information.
2- The capacity of communication channel to transfer information.
3- Coding as a means of utilizing channels at full capacity.

These three concepts are tied together in what can be called the fundamental theorem of information theory as follows:

Given an information source and a communication channel, there exists a coding technique such that the information can be transmitted over the channel at any rate less than the channel capacity and with arbitrarily frequency of error despite the presence of noise.

Coding Techniques

Information is rarely transmitted directly to the receiver, without any processing due to the followings:

• source alphabet is different from channel one, therefore some adaptation of the source to the channel is needed (*information representation codes*).

• channel must be very efficiently used (close as possible to its full capacity), meaning that the source must be converted into an optimal one. For an efficient use of the channel - in order to minimize transmission time and/or storage space – source compression is needed (*compression codes*).

• it is also necessary to adapt the source to the channel to ensure synchronization between transmitter and receiver.

• information transmitted over a noisy channel is distorted by channel noise; this is why error detecting and correcting codes are used in error control procedures.

• information confidentiality from unauthorized persons must be provided in some applications.

The need for source processing prior to transmission (or storage) is leading to the processing known as coding.

The problem of representing the source alphabet symbols S_i in term of another system of symbols (usually the binary system consisting of the two symbols 0 and 1) is the main topic of coding.

The two main problems of representation are the following:

1. How to represent the source symbols so that their representations are far apart in same suitable sense. As a result, in spite of small changes (noise) in their representation, the altered symbols can be discovered to be wrong and even possibly corrected (i.e Error – Detecting codes & Error- correcting codes).

2. How to represent the source symbols in a minimal form for purposes of efficiency. The average code length is minimized, where $_i$ is the length of the

representation of the ith symbol S_i the entropy function provides a lower bound on L ($L \geq H(x)$).

$$L = \sum_{i=1}^{N} P_i l_i$$

Average Length of Code L

Let a code transform the source symbols $x_1,x_2,...,x_N$ into the code symbols $C_1,C_2,...,C_N$. let the probabilities of the source symbols be $P_1,P_2,\ldots, P_N$ and let the length of the code words be $l_1, l_2,\ldots, l_N$, then

$$L = \sum_{i=1}^{N} P_i l_i$$

Average Information (Entropy)

In practice we are often more interested in the ***average*** information conveyed in some process than in the ***specific*** information in each event. For example, a source may produce a set of events of probability $P_1, P_2, P_3, \dots, P_i, \dots, P_n$. In a long sequence of n, event i will occur n*Pi times, contributing n * - log Pi bits. The average information over all events is called the entropy H, and is given by:

$$\bar{I} = \sum_{i=1}^{N} P_i l$$

$$\bar{I} = \mathbf{H} = -\sum_{i=1}^{N} \mathbf{pi \log pi} \qquad \mathbf{bit\backslash symbol}$$

In the case of letters of alphabet, the probabilities are not really all the same. The entropy is given by: H = - (P(A) log P(A) + P(B) log P(B) + … + P(Z) log P(Z)) ≈ 4 .1 b its (using standard values for P(A), P(B), etc.) The units are often given as bits/letter or bits/symbol to stress the average nature of the measure. The function H of the probability distribution P_i measures the amount of ***uncertainty, surprise, or information*** the distribution contains. The term "entropy" was deliberately chosen for the name of measure of average information, because of its similarity to entropy in thermodynamics. In ***thermodynamics***, entropy is a measure of the ***degree of disorder*** of a system, and disorder is clearly related to information.

Example1: A binary source produces a stream of 0s and 1s with probabilities P(0) = 1/8 and P(1) = 7/8 respectively, find the entropy of this source.

Sol :

$$\mathbf{H} = -\sum_{i=1}^{N} \mathbf{pi \log pi} = \mathbf{-\ (1/8 \log 1/8 + 7/8 \log 7/8)}$$

$$= -(\ 0.125 \times -3 + 0.875 \times -\ 0.192)$$

$$= -\ (\ -\ 0.375 -\ \ 0.168)\ = 0.543 \text{ bits } \backslash \text{ Symbol}$$

Example2: For the above source assume that the source send the below stream of letters **ADBAAEBACBAAACABDAAB**

Find

i. the amount of information each letter convey?
ii. the amount of information that the total stream of letters (message) convey?

Sol:

We can see that the total number of letters in the message is equal to 20.

P(A) = 10/20 = 0.5
P(B) = 5/20 = 0.25
P(C) = P(D) = 2/20 = 0.1
P(E) = 1/20 = 0.05

i. I_A = - log 0.5 =1
I_B = - log 0.25 =2
$I_C = I_D$ = - log 0.1 = 3.322
I_E = - log 0.05 = 4.322

ii. the amount of information that the message convey is :

$I_{message} = 10 \times 1 + 5 \times 2 + 2 \times 3.322 + 2 \times 3.322 + 1 \times 4.322$
$= 10 + 10 + 6.644 + 6.644 + 4.322$
$= 37.61$

Example3: The above source produces a stream of letters (A, B, C, D, and E) with the probabilities bellow

Letter	**A**	**B**	**C**	**D**	**E**
Probability	**0.5**	**0.25**	**0.1**	**0.1**	**0.05**

Find the entropy of this source.

Sol

$H = - \sum Pi \text{ Log } Pi$

= - (0.5 log 0.5 +0.25 log 0.25 +0.1 log 0.1 +0.1 log 0.1 +0.05 log 0.05)

= - (0.5 *-1 + 0.25 * -2 + 2 * 0.1 * -3.322 + 0.05 * -4.322)

= 1.88 bits/symbol

Or

H = I (message) / total number of letters

= 37.61 / 20

= 1.88 bits / symbols

Example4: Find and then compare between the entropies of the following three binary

sources source i **source ii** **source iii**

A1	A2
1/256	255/256

B1	B2
1/2	1/2

C1	C2
7/16	9/16

Sol

H_i= -(1/256 log 1/256 +255/256 log 255/256) = 0.0369 bit/symbol

H_{ii}= -(1/2 log 1/2 + 1/2 log 1/2) =1 bit/ symbol

H_{iii}= - (7/16 log 7/16 + 9/16 log 9/16)= 0.989 bit/ symbol

Source ii give more information than source iii and source iii give more information than source i.

Example5: a source producing three letters A,B,C with the probabilities :

i . 1/3, 1/3, 1/3 respectively

ii. 1/2, 1/4, 1/4 respectively

Find and compare between the entropies of the two cases.

Sol

i. H= -3(1/3 log 1/3)

= log 3

= 1.58 bit/ symbol

ii. H= - (1/2 log 1/2 + 1/4 log 1/4 +1/4 log 1/4)

= - (-0.5 – 0.5- 0.5)

= 1.5 bit/ symbol

source in case i give more information than the source in case ii.

Example6: a source produce a stream of **twenty** letters (A,B,C,D,E) with probabilities

P(A) =P(E),

P(B) = P(D),

P(A) = 0.5 P(B) = 0.25 P(C).

Find

a. The entropy for this source

b. The amount of information each letter convey

c. The amount of information that the total message convey.

Sol

P(A) = P(E) = 0.1

P(B) = P(D) = 0.2

P(C) = 0.4

a. **H = - ∑ pi log pi**

= - (0.1 log 0.1 + 0.2 log0.2 +0.4 log0.4 +0.2 log0.2 + 0.1 log0.1) =

- (2 × 0.1 log 0.1 + 2 × 0.2 log 0.2 + 0.4 log 0.4)

= - (2 × 0.1 × -3.322 + 2 × 0.2× -2.322 + 0.4 × -1.322)

= 0.66444 + 0.92888 + 0.52888

= 2.1222

b. I_A = - log 0.1 = 3.3222 =I_E

I_B= - log 0.2 = 2.3222 =I_D

I_C = - log 0.4 = 1.3222

c . $I_{message}$ =2× 0.3222 +4× 2.3222 +8 ×1.3222 + 4 ×2.3222 +2 × 3.3222

= 42.444

Or

$I_{message}$ = no. of letters × Ī

= 20 * 2.1222

= 42.444

Entropy and Length of the Code

One of the key concepts in coding theory : we want to assign a fewer number of bits to code the more likely events.

0 ≤ Entropy ≤ log2 (M) also 0 ≤ H ≤ L;

This illustrate that the more randomness that exist in the source symbols, the more bits per symbol are required to represent those symbols.

{ For 2 symbols H = 1 , L=1 ; for 4 symbols H=2, L= 2 ,

for 8 symbols H=3, L=3, … }

on the other hands, entropy provides us with the theoretical minimum for the average number of bits per symbol (average length of the code) that could be used to encode the same symbol. The closer L is to the entropy, the better the coder.

Code Efficiency and Redundancy

$$\xi_{code} = \frac{H(x)}{L} * 100\% \quad where\ \xi_{code} = \text{code Efficiency}$$

$$R_{code} = \frac{L - H(x)}{L} * 100\% = \left(1 - \frac{H(x)}{L}\right) * 100\%$$

$$= \left(1 - \xi_{code}\right) * 100\% \qquad where\ R_{code} = Code\ Redundancy$$

Source Coding Techniques

1. Fixed Length Coding

In fixed length coding technique all symbols assigned with equal length because the coding don't take the probability in account.

The benefit of the fixed length code is ease of applied (easy in coding and decoding)

Example1: Let x = { $x_1,x_2,\ldots,x_{16}$} where pi= 1/16 for all i ,find $\zeta_{\text{source code}}$

Sol

$H(x) = \log_2 M$

$= \log_2 16$

$= 4$ bit/symbol (because $p_1=p_2=\ldots=p_{16}= 1/M$)

For fixed length code

$L = {}_i = [\, log_2 M \,] = [\, log_2 16] = [4] = 4$

$\therefore \ \xi source\ code = H(x) / L * 100\% = 4/4 *100\% = 100\%$

source symbols	probability	codeword	code length
X_1	P_1	0000	4
X_2	P_2	0001	4
.	.	.	.
.	.	.	.
X_{16}	P_{16}	1111	4

Example2: Let x = { x_1,x_2, x_3,x_4,x_5} where pi= 1/5 for all i ,find $\zeta_{\text{source code}}$

Sol

$H(x) = \log_2 M$

$= \log_2 5$

$= 2.322$ bit/symbol

For fixed length code

$L = {}_i = [\, log_2 M \,] = [\, log_2 5] = [2.322] = 3$ bit

$\therefore \ \xi source\ code = H(x) / L * 100\% = 2.322/3 *100\% = 77\%$

Symbol	Probability	Code	l_i
x_1	0.2	000	3
x_2	0.2	001	3
x_3	0.2	010	3
x_4	0.2	011	3
x_5	0.2	100	3

Example3: Let x = { $x_1,x_2,\ldots,x_{12}$} where pi= 1/12 for all i ,find $\zeta_{\text{source code}}$

Sol

$H(x) = \log_2 M$

$= \log_2 12$

= 3.585 bit/symbol

For fixed length code

$L = {}_i = [\, log_2\ M\ ] = [\, log_2\ 12] = [3.585] = 4$ bit

$\therefore\ \xi\ source\ code$ = H(x) / L * 100% = 3.585/4 *100% =89%

Symbol	Probability	Code	l_i
X_1	1/12	0000	4
X_2	1/12	0001	4
.	.	.	.
.	.	.	.
X_{12}	1/12	1100	4

2- Variable length code

The codes we have looked above have all fixed lengths and they are called ***block code*** from the fact that the message, are of fixed block lengths in the stream of symbols being sent.

We now examine variable length code in more detail. The ***advantage*** of a code where the message symbols are of variable length is that some time the code is more efficient in the sense that to represent the same information we can use ***fewer digits on the average***. To accomplish this we need to know something about the statistics of the message being sent. If every symbol is as likely as every other one, then the fixed length code are about as efficient as any code can be. But if some symbols are more probable than others, then we can take advantage of this feature to make the ***most frequent symbols correspond to the shorter encodings*** and the ***rare symbols correspond to the longer encodings***. This is exactly the idea behind variable length coding.

However, variable length code bring with them a fundamental problem, at the receiving end, how do you recognize each symbol of the code? In, for example, a binary system how do you recognize the end of one code word and the beginning of the next ?

If the probabilities of the frequencies of occurrence of the individual symbols are sufficiently different, then variable – length encoding can be significantly more efficient than fixed – length encoding

$$L = \sum_{i=1}^{N} P_i l_i \quad \textbf{bit/symbol}$$

Note: for each symbol increase the probability decrease the code length

Symbol	probability	code length
X 1	p 1	I_1
X 2	p 2	I_2
.	.	.
.	.	.
X m	p m	Im

Source coding for special source

Source coding can achieve 100 % efficiency when r* -level code is used with source having a probability in the form $P(x_i)= r^{-Ii}$ (where Ii is an integer) for all xi .

Ex1: design a binary code for the following source :

X_i	$P(x_i)$	$P(x_i)$	l_i	Code
x_1	1/4	2^{-2}	2	00
x_2	1/4	2^{-2}	2	11
x_3	1/8	2^{-3}	3	010
x_4	1/8	2^{-3}	3	111
x_5	1/16	2^{-4}	4	0000
x_6	1/16	2^{-4}	4	0101
x_7	1/16	2^{-4}	4	1001
x_8	1/16	2^{-4}	4	1111

check

$L = \sum I_i\, p(x_i)$

$=2/4 +2/4+3/8+3/8+4/16+4/16+4/16+4/16$

$=1/2+1/2+3/4+1=2.75$ bit/symbol

$H(x)= -\sum p(x_i) \log p(x_i)$

$= 2.75$ bit/symbol

ξ code $= H(x)/L * 100\% = 275/275 *100\% = 100\%$

Ex2: A source produce 7 symbols $x_1, x_2, \ldots, x_7$, with probabilities 0.0625, 0.25, 0.125, 0.25, 0.125, 0.0625, 0.125. design a binary code for the above source, then determine the code efficiency.

X i	P(x i)	P(xi)	l i	Code
x 1	1/16	2^{-4}	4	1000
x 2	1/4	2^{-2}	2	00
x 3	1/8	2^{-3}	3	010
x 4	1/4	2^{-2}	2	11
x 5	1/8	2^{-3}	3	101
x 6	1/16	2^{-4}	4	1001
x 7	1/8	2^{-3}	3	110

check

$L = \sum I_i \, p(x_i)$

$=4/16+2/4+3/8+2/4+3/8+4/16+3/8$

$= 0.25 + 0.5 + 0.375 + 0.5 + 0.375 + 0.25 + 0.375 = 2.625$ Bit/symbol

$H(x) = -\sum P(xi) \log P(xi)$

$= - (1/16*-4 +1/4*-2 +1/8*-3 + 1/4*-2 +1/8*-3 +1/16 *-4 +1/8 * -3)$

$= 2.625$ Bit/symbol

ξ code$= H(x) / L * 100\ \% = 2.625 / 2.625 * 100\% = 100\ \%$

Shannon – Fano method

To encode a message using Shannon-Fano method, you can follow the below steps :

1. Sort the symbols in descending order according to their probabilities. 2. Divide the list of symbols into two parts : upper and lower, so that the summation of the probabilities of the upper part is equal as possible to the summation of the lower part symbols.

3. Assign "0" code to each of the upper part symbols, and "1" code to each of the lower part symbols.

4. Divide each of the upper and lower part into upper and lower subdivision as in step (2) above, and assign the code "0" and "1" as in step (3) above.

5. Continue in step(4) until each subdivision contains only one symbols.

Ex1: A source produce 5 independent symbols (x_1, x_2, x_3, x_4, x_5) with its corresponding probabilities 0.1, 0.3, 0.15, 0.25, 0.2 . design a binary code for the above source symbol using Shannon – fanon method.

Sol

symbols	P i	code	Ii
x 2	0.3	0 0	2
x 4	0.25	0 1	2
x 5	0.2	1 0	2
x 3	0.15	1 1 0	3
x 1	0.1	1 1 1	3

$L = \sum I_i\, p(x_i)$

$= 2*0.3 + 2*0.25 + 2*0.2 + 3*0.15 + 3*0.1 = 2.25$ Bit\symbol

$H = -\sum P(xi) \log P(xi)$

$= -(0.3 \log 0.3 + 0.25 \log 0.25 + 0.2 \log 0.2 + 0.15 \log 0.15 + 0.1 \log 0.1) = 2.228$ Bit/symbol

ξ code$= H(x) / L * 100\% = 2.228 / 2.25 * 100\% = 99\%$

Note: If we use fixed length coding $L = [\log 2] = [2.3219] = 3$

ξ code$= H(x) / L * 100\% = 2.228 / 3 * 100\% = 74\%$

.: coding in Shannon – fanon is more efficient than coding in fixed length coding

.

Ex2 : A source produce 5 independent symbols (x_1, x_2, x_3, x_4, x_5) with its corresponding probabilities 0.1, 0.05, 0.25, 0.5, 0.1. design a binary code for the above source symbol using Shannon – fanon method .

Sol

symbols	P i	code	Ii
x_4	0.5	0	1
x_3	0.25	1 0	2
x_1	0.1	1 1 0	3
x_5	0.1	1 1 1 0	4
x_2	0.05	1 1 1 1	4

$L = \sum I_i\, p(x_i)$

$= 1*0.5 + 2*0.25 + 3*0.1 + 4*0.1 + 4*0.05 = 1.9$ Bit / symbol

$H = -\sum P(xi) \log P(xi)$

$= -(0.5 \log 0.5 + 0.25 \log 0.25 + 0.1 \log 0.1 + 0.1 \log 0.1 + 0.05 \log 0.05) = 1.88$ Bit/symbol

ξ code$= H(x) / L * 100\% = 1.88 / 1.9 * 100\% = 99\%$

Ex3 : A source produce 5 independent symbols (x_1, x_2, x_3, x_4, x_5) with its corresponding probabilities 0.1, 0.35, 0.3, 0.05, 0.2. design a binary code for the above source symbol using Shannon – fanon method .

Sol

symbols	Pi	code	Ii
x 2	0.35	0	1
x 3	0.3	1 0	2
x 5	0.2	1 1 0	3
x 1	0.1	1 1 1 0	4
x 4	0.05	1 1 1 1	4

$L = \sum I_i \, p(x_i)$

$= 1*0.35 + 2*0.3 + 3*0.2 + 4*0.1 + 4*0.05 = 2.15$ Bit / symbol

$H = -\sum P(xi) \log P(xi)$

$= - (0.35 \log 0.35 + 0.3 \log 0.3 + 0.2 \log 0.2 + 0.1 \log 0.1 + 0.05 \log 0.05) = 2.062$ Bit/symbol

ξ code $= H(x) \, L * 100\% = 2.062 / 2.15 * 100\% = 96\%$

Sol 2

symbols	Pi	code	Ii
x 2	0.35	0 0	2
x 3	0.3	0 1	2
x 5	0.2	1 0	2
x 1	0.1	1 1 0	3
x 4	0.05	1 1 1	3

$L = \sum P_i l_i$

$= 1*0.35 + 2*0.3 + 2*0.2 + 3*0.1 + 3*0.05 = 2.15$ Bit / symbol

Ex4 : A source produce 5 independent symbols (A, B, C, D, Z) with the below probabilities:

1. For the letters (A, B, C, D) , the probability of each letter is twice as its successor

2. The probability of the letter Z is equal to the summation of the probabilities of B and D.

design a binary code for the above source symbol using Shannon – fanon method

.

Sol:

symbols	**Pi**	**code**	**Ii**
A	0.4	0	1
Z	0.25	1 0	2
B	0.2	1 1 0	3
C	0.1	1 1 1 0	4
D	0.05	1 1 1 1	4

$L = \sum I_i \, p(x_i)$

$= 1*0.4 + 2*0.25 + 3*0.2 + 4*0.1 + 4*0.05 = 2.1$ Bit / symbol

$H = -\sum P(xi) \log P(xi)$

$= -(0.4 \log 0.4 + 0.25 \log 0.25 + 0.2 \log 0.2 + 0.1 \log 0.1 + 0.05 \log 0.05)$

$= 2.025$ Bit/symbol

ξ code $= H(x) / L * 100\% . = 2.025 / 2.1 * 100\% = 96\%$

Huffman Coding method

Huffman coding is a popular method for compressing data with variable-length codes. Given a set of data symbols and their frequencies of occurrence, the method constructs a set of variable-length codewords with the shortest average length for the symbols. The Huffman method is similar to the Shannon–Fano method, It generally produces better codes, it produces the best code when the probabilities of the symbols are negative powers of 2. The main difference between the two methods is that Shannon–Fano constructs its codes top to bottom (from the leftmost to the rightmost bits), while Huffman constructs a code tree from the bottom up (builds the codes from right to left).

1. Binary Huffman Codes

To encode a message using Huffman method you can follow the below steps:

1) Sort the symbols in descending order according to their probabilities.

2) Assign "0" and "1" code for the two symbols with the lower probabilities.

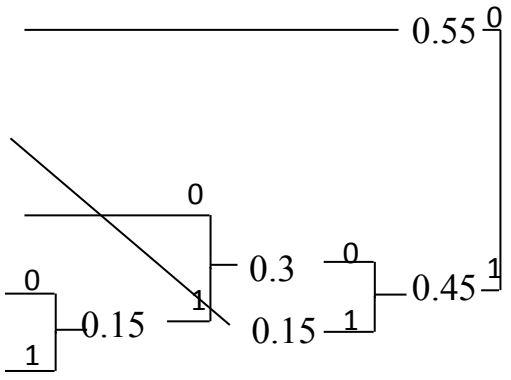
3) Combine those two symbols in step (2) to construct a new symbol with probability equal to the summation of the two probabilities, then enter the new symbol in the list at a new position appropriate to its new probability.

4) Repeat step (2 and 3) until the list has only one symbol.

5) The code word of any symbol may be obtained by following the series of binary codes (0s, 1s) which has been assigned to that symbol.

Example1: Design a binary code for the below source symbol using Huffman method.

Xi	Pi	code	Ii
x 1	0.55	0	1
x2	0.15	11	2
x3	0.15	100	3
x4	0.1	1010	4
x5	0.05	1011	4

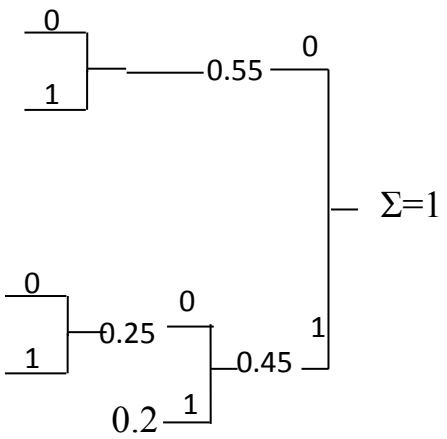


$L=\Sigma\ Ii\ p(X_i)$

$= (0.55 * 1) + (0.15 * 2) + (0.15 * 3) + (0.1 * 4) + (0.05 * 4) = 1.9$

Example2: Design a binary code for the below source symbol using Huffman method.

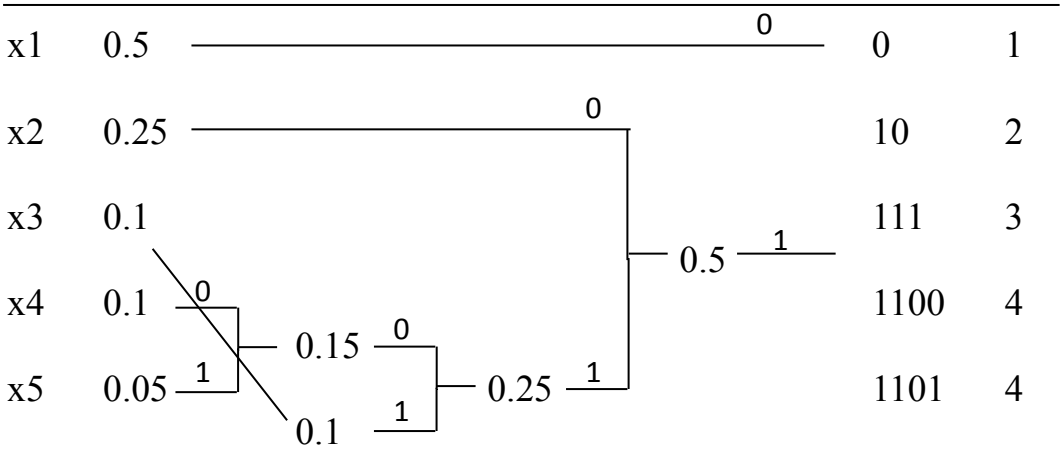
Xi	Pi	code	Ii
x 1	0.3	00	2
x2	0.25	01	2
x3	0.2	11	2
x4	0.15	100	3
x5	0.1	101	3



$L=\Sigma\ Ii\ p(X_i) = 2.25$

$= (0.3 * 2) + (0.25 * 2) + (0.2 * 2) + (0.15 * 3) + (0.1 * 3) = 2.25$

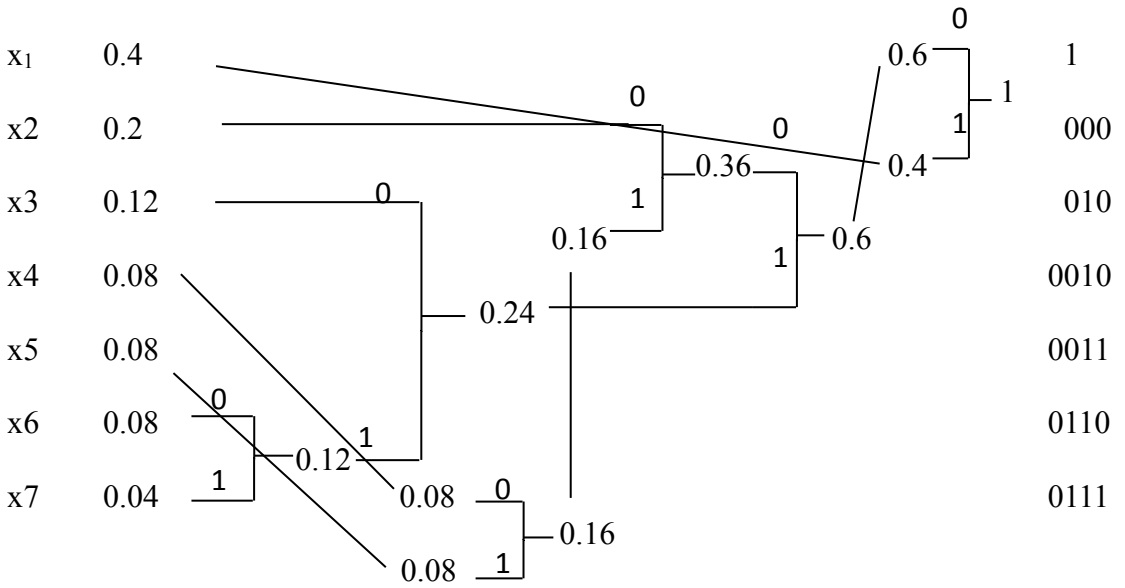
Example3: Design a binary code for the below source using Huffman method.



$L=\Sigma\ Ii\ p(X_i)$

$= (0.5 * 1) + (0.25 * 2) + (0.1 * 3) + (0.1 * 4) + (0.05 * 4) = 1.9$

Example4: Design a binary code for the below source symbol using Huffman method



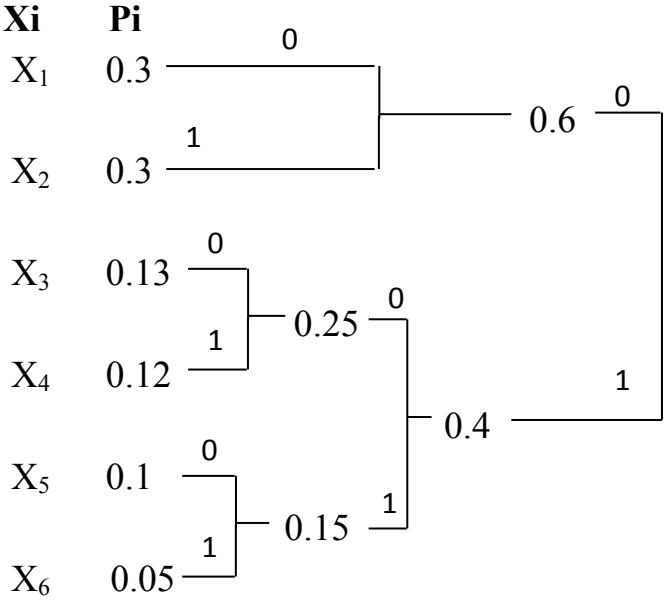
$L=\Sigma\ P(X_i)\ Ii = 2.48$ bite/symbol

$H(x)=2.419$

ξ code $= H(x)/L * 100\% = 2.419/2.48 * 100\ \% = 97.54\ \%$

for Shannon –fano
L= 2.52 bit/symbol
ξ code =2.419/2.52 * 100 % = 95.99 %

Example5: Design a binary code for the below source using Huffman method.



L=Σ Ii p(X_i)

= (0.3 * 2) + (0.3 * 2) + (0.13 * 3) + (0.12 * 3) + (0.1 * 3) + (0.05 * 3)

= 2.4

2. Ternary Huffman Codes

Huffman also presented the general case of its algorithm for a channel with an alphabet containing more than two symbols (m>2). Unlike for binary coding, when coding in a larger m base, each reduction will be formed from m symbols. For the design of general r-ary Huffman codes the Huffman coding algorithm is as follows:

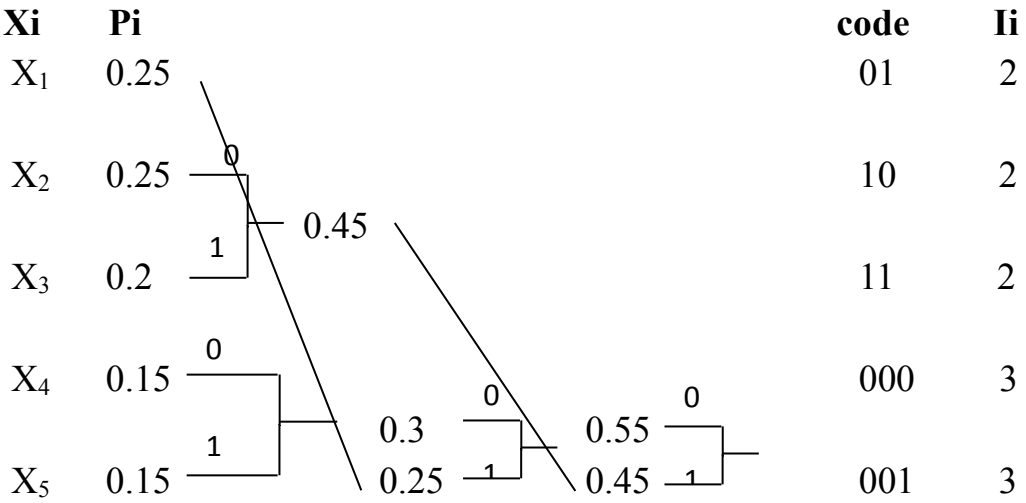
1. Calculate α= q-r/ (r-1). If α is a non-integer value then append "dummy" symbols to the source with zero probability until there are q= r+ [α] (r-1) symbols.
2. Re-order the source symbols in decreasing order of symbol probability.

3. Successively reduce the source S to X_1, then X_2 and so on, by combining the last r symbols of X_j into a combined symbol and re-ordering the new set of symbol probabilities for X_{j+1} in decreasing order. For each source keep track of the position of the combined symbol, X_{q-r+1} Terminate the source reduction when a source with exactly r symbols is produced. For a source with q symbols the reduced source with r symbols will be X[α].

4. Assign a compact r-ary code for the final reduced source. For a source with r symbols the trivial code is {0,1,…,r}.

5. Backtrack to the original source r assigning a compact code for the jth reduced source by the method described in Result 3.4. The compact code assigned to S, minus the code words assigned to any "dummy" symbols, is the r-ary Huffman code.

Example1: Given five symbols with probabilities 0.15, 0.15, 0.2, 0.25, and 0.25, construct binary and ternary Huffman code using a ternary alphabet (m=3) and determine the average code size.



$L=\Sigma\ Ii\ p(X_i)$

$2\times0.25 + 2\times0.25 + 2\times0.2 + 3\times0.15 + 3\times0.15 = 2.3$ bits/symbol.

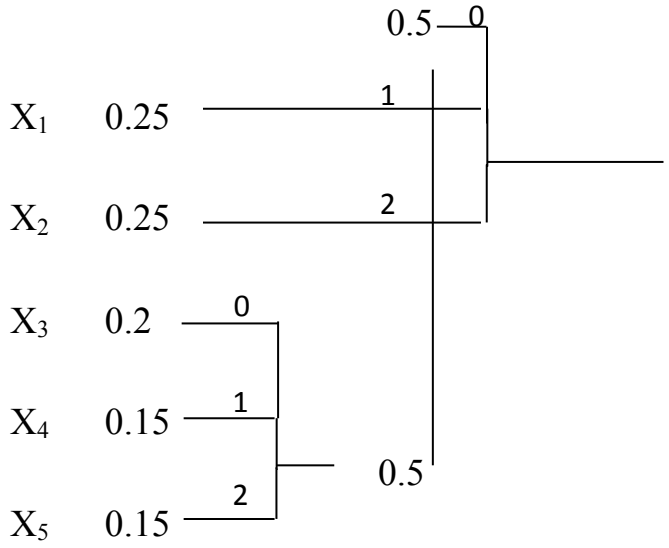
$H(Xi)=2.28$

ξ code = H(x)/L * 100% =2.28/2.3 * 100 % = 99 %

r=3 , q=5

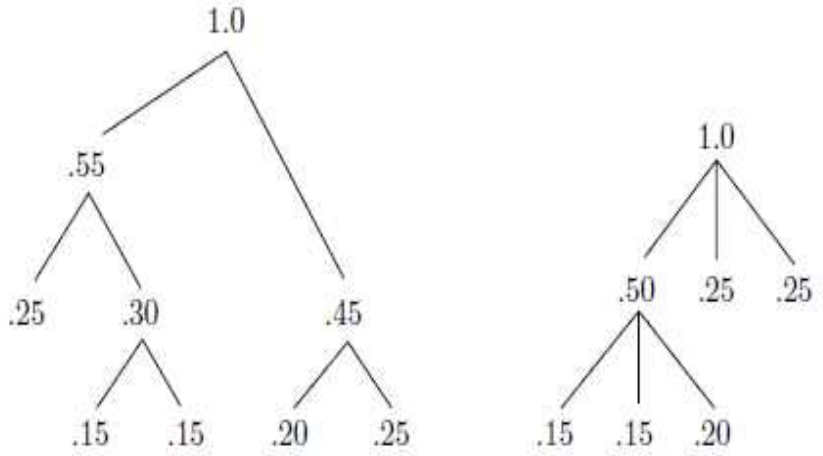
α= q-r/ (r-1) = 5-3/ 2 = 2/2= 1
q= r+ [α] (r-1) = 3+ 1 * (2) =5 symbols where [α] =1

Xi	Pi	code	Ii
X_1	0.25	1	1
X_2	0.25	2	1
X_3	0.2	00	2
X_4	0.15	01	2
X_5	0.15	02	2



$L=\Sigma\ Ii\ p(X_i)$
$1\times0.25 + 1\times0.25 + 2\times0.2 + 2\times0.15 + 2\times0.15 = 1.5$ bits/symbol.

Notice that the ternary codes use the digits 0, 1, and 2.



Example2: Encode the source X:

x_1	x_2	x_3	x_4	x_5	x_6
0.3	0.25	0.15	0.15	0.10	0.05

using a ternary alphabet (r=3) and determine the efficiency

Solution

r=3 , q=6
α= q-r/ (r-1) = 6-3/ 2 = 4/2= 2
q= r+ [α] (r-1) = 3+ 2 * (2) =7 symbols where [α] =2
Note that a supplementary symbol s_7 of zero probability must be added.

x_i	p_i	c_i	R_1		R_2	
x_1	0.3	1	0.3		0.45	0
x_2	0.25	2	0.25		0.3	1
x_3	0.15	00	0.15	00	0.25	2
x_4	0.15	01	0.15	01		
x_5	0.10	020	0.15	02		
x_6	0.05	021				
x_7	0.0	022				

$L=\Sigma\ Ii\ p(X_i)$
= 0.3 X 1 + 0.25 X 1 + 0.15 X 2 + 0.15 X 2 + 0.10 X 3 + 0.05 X 3
= 1.6 bit/ symbol

Extension of code

when we extend a source with order of n , we have the following equation for L and H

$$L = \frac{Ln}{n}$$

$$H_{(x)} = \frac{Hn(x)}{n}$$ where n is the order of extension

We also have the important relation for coding a source,
$\mathbf{H_{(x)} \le L < H_{(x)+1}}$

Now , if we take the n^{th} extension of the code the above relation is also applied, so
$\mathbf{H_{n(x)} \le L_n < H_{(x)+1}}$ Where L_n is the average code length ,for the extended source
Since $L_n = n\ L$
And $H_{n(x)} = n_{H(x)}$
Then
$nH_{(x)} \le n\ L < n\ H_{(x)+1}$ divided by n

$$H_{(x)} \le L < H_{(x)} + \frac{1}{n}$$

Example : A binary source produce two symbols x_1 ,x_2 with probabilities $p(x_1)$ = 0.8 , $p(x_2)$ =0.2 . find ξ for the 1st 2nd 3rd extension of binary code for the above source .

Solution:

1) n=1 , coding with extension

Xi	Pi	code	Ii
x_1	0.8	0	1
x_2	0.2	1	1

L=1 Bit /symbol

$H(x) = - \Sigma Pi \log Pi$

$= - (0.8 \log 0.8 + 0.2 \log 0.2)$

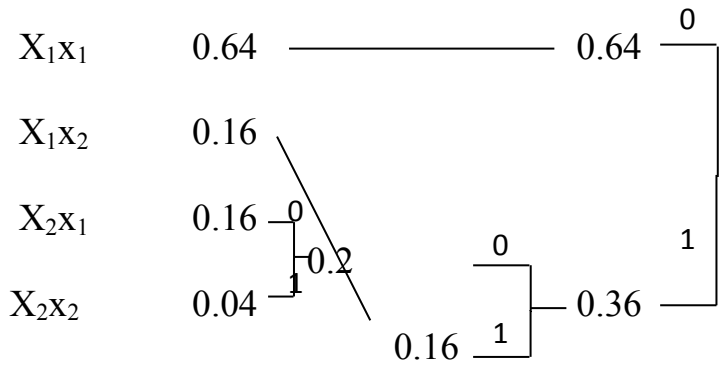
$= 0.72$ Bit/symbol

ξ code = $H(x)/L * 100\%$

$= 0.72 * 100\% = 72\ \%$

2) n=2 , coding with extension of order 2

Symbol	Pi	code	Ii
X_1x_1	0.64	0	1
X_1x_2	0.16	11	2
X_2x_1	0.16	100	3
X_2x_2	0.04	101	3



$L_2 = \Sigma I_i P(X_i)$

$= 1*0.64 + 2*0.16 + 3* 0.16 + 3* 0.04 = 1.56$

$L = L_2/2 = 1.56/2 = 0.78$ bit / symbol

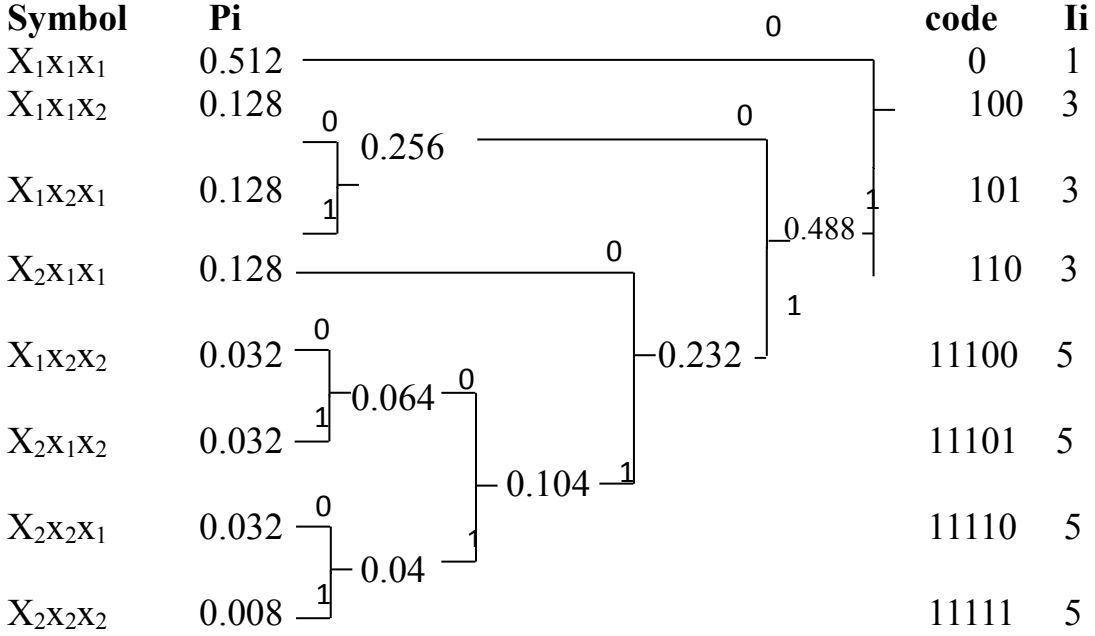
$H_2(x) = - (0.64 \log 0.64 + 2*0.16 \log 0.16 + 0.04 \log 0.04)$

$= 1.44$

$H(x) = H_2(x) / 2 = 1.44/2 = 0.72$ bit/symbol

ξ code = $H(x)L * 100\% = 0.72 / 0.78 * 100\ \% = 94\ \%$

3) n= 3 ,coding with extension of order 3



$L_3 = \Sigma I_i P(X_i)$
=2.184
L = L3 / 3 = 2.184 / 3 = 0.728 bit / symbol

H3 (x) = -Σ pi log pi
=2.16

H(x) = H3(x) / 3 = 2.16 / 3 = 0.72 bit/symbol

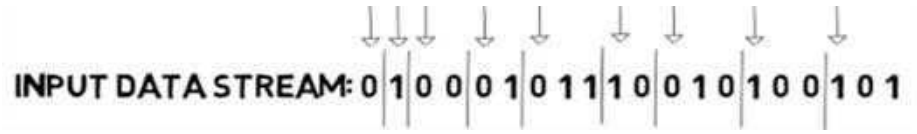
ξ code = H(x) / L * 100% = 0.72 / 0.728 * 100 % = 98.9 %

Lempel-Ziv Algorithm (LZ)

The dictionary techniques build the dictionary of common sub-streams either at the same time with their detection or as a distinct step. Each dictionary sub-stream is put into correspondence with a codeword and the message is then transmitted through encoded sub-streams from the dictionary. Dictionary techniques must allow the transmission of the dictionary from the compressor (transmitter), to the de-compressor (receiver). When using semi adaptive dictionary algorithms, the first step is to transmit the dictionary and then to compress the message. Adaptive dictionary techniques do not require explicit transmission of the dictionary. Both transmitter and the receiver simultaneously build their dictionary as the message is transmitted. At each moment of the encoding process the current dictionary is used for transmitting the next part of the message.

From adaptive dictionary techniques category, the most used ones are the algorithms presented by J. Ziv and A. Lempel, these algorithms are known as "LZ algorithms". The basic idea of LZ algorithms is to replace some sub-streams of the message with codewords so that for their each new occurrence only the associated codeword will be transmitted. Lempel-Ziv code is an adaptive coding technique that does not require prior knowledge of symbol probabilities.

Example: Input data stream 0 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1



Numerical position	1	2	3	4	5	6	7	8	9
Subsequences	0	1	00	01	011	10	010	100	101
Representation	-	-	11	12	42	21	41	61	62

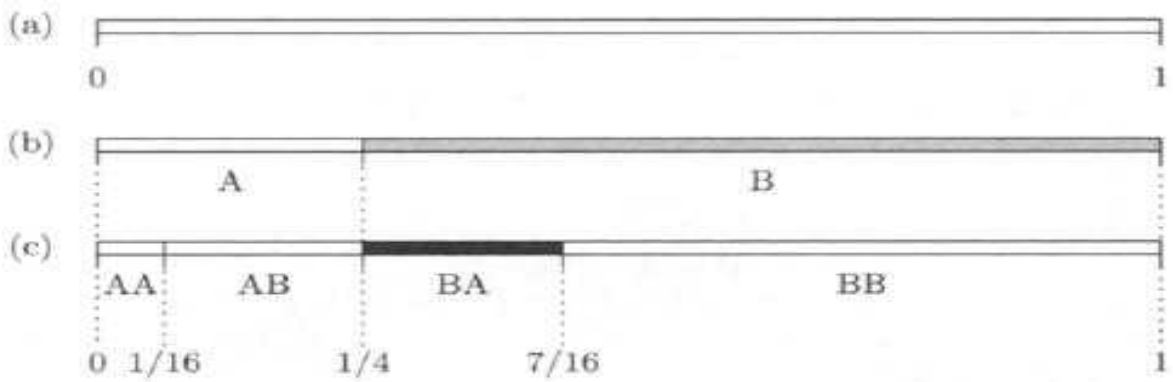
Example: input data stream A A B A B B B A B A A B A B B B A B B A B B

Numerical position	1	2	3	4	5	6	7	8	9
Subsequences	A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
Representation	0A	1B	2B	0B	2A	5B	4B	3A	7

Arithmetic coding algorithm

Arithmetic coding is a technique for coding that allows the information from the messages in a message sequence to be combined to share the same bits. The technique allows the total number of bits sent to asymptotically approach the sum of the self information of the individual messages. Arithmetic coding overcomes the problem of assigning integer codes to the individual symbols by assigning one (normally long) code to the entire input file. The steps of arithmetic coding algorithm are:

1. Put the symbols in decreasing order of probabilities: $p1 \leq p2 \leq \ldots \leq pn$.
2. Divide the interval [0; 1) into n intervals of dimensions p1, p2 ,…, pn .
3. Read the first symbol of the message and memorize its associated interval.
4. Divide this interval into n intervals, each of them proportional to p1, p2 ,…, pn .
5. Read the next symbol and memorize the interval associated to it.
6. Continue the process following the same algorithm.
7. Transmit one number from the last memorized interval.



Example: Encode the message "abac" using an arithmetical code designed for the source S:

$$S:\begin{pmatrix} a & b & c \\ 1/2 & 1/3 & 1/6 \end{pmatrix}$$

Processed Interval			Read symbol	Memorized Interval
1/2 (1-0)	1/3 (1-0)	1/6 (1-0)	a	[0,1/2)
0 a	1/2 b	5/6 1 c		
1/2 (1/2 -0)=1/4	1/3 (1/2-0)=1/6	1/12	b	[1/4,5/12)
0 a	1/4 b	5/12 1/2 c		
1/2(5/12-1/4)=1/12	1/18	1/36	a	[1/4,5/3)
1/4 a	1/3 b	7/18 5/12 c		
1/2(1/3-1/4)=1/24	1/36	1/72	c	[23/72,1/3)
1/4 a	7/24 b	23/72 1/3 c		

- **Error Control**

Environmental interference and physical defects in the communication medium can cause random bit errors during data transmission. Error coding is a method of detecting and correcting these errors to ensure information is transferred intact from its source to its destination. The "code word" can then be decoded at the destination to retrieve the information. The extra bits in the code word provide *redundancy* that, according to the coding scheme used, will allow the destination to use the decoding process to determine if the communication medium introduced errors and in some cases correct them so that the data need not be retransmitted. Different error coding schemes are chosen depending on the types of errors expected, the communication medium's expected error rate, and

whether or not data retransmission is possible. The former strategy uses **Error-detecting Codes** and **Error-Correcting Codes**.

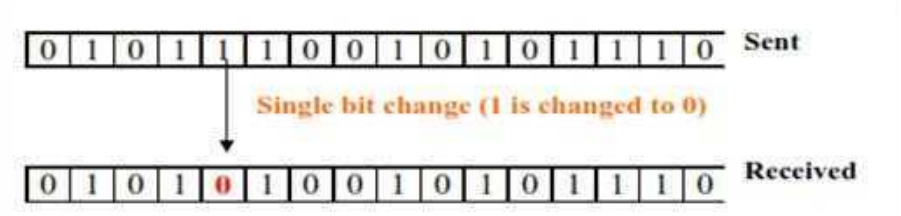
Error control is the process of detecting and correcting both the bit level and packet level errors.

- **Types of errors**

These interferences can change the timing and shape of the signal. If the signal is carrying binary encoded data, such changes can alter the meaning of the data. These errors can be divided into two types: Single-bit error and Burst error.

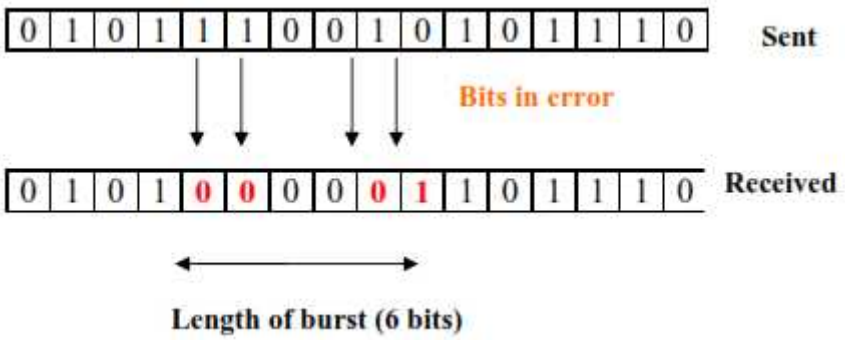
- **Single-bit Error**

The term single-bit error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1.



- **Burst Error**

The term burst error means that two or more bits in the data unit have changed from 0 to 1 or vice-versa. Note that burst error doesn't necessary means that error occurs in consecutive bits. The length of the burst error is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not be corrupted. Burst errors are mostly likely to happen in serial transmission.



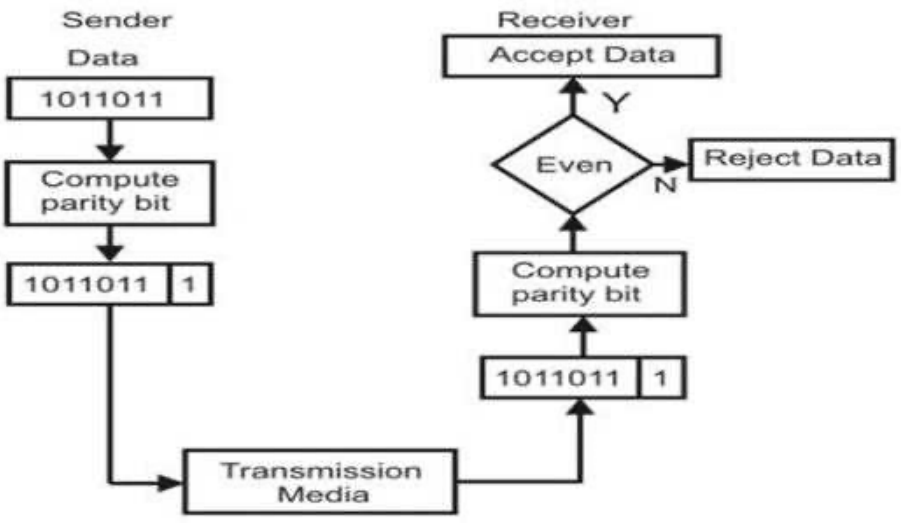
Error Detecting Codes

Error detection is the process of detecting the error during the transmission between the sender and the receiver. Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors. Popular techniques are:

- Simple Parity check or One-dimension Parity Check
- Two-dimensional Parity check
- Checksum
- Cyclic redundancy check

- **Simple Parity Checking or One-dimension Parity Check**

The most common and least expensive mechanism for error- detection is the simple parity check. In this technique, a redundant bit called **parity bit**, is appended to every data unit so that the number of 1s in the unit. Blocks of data from the source are subjected to a check bit or Parity bit generator form, where a parity of 1 is added to the block if it contains an odd number of 1's (ON bits) and 0 is added if it contains an even number of 1's. At the receiving end the parity bit is computed from the received data bits and compared with the received parity bit, as shown in Fig This scheme makes the total number of 1's even, that is why it is called even parity checking.

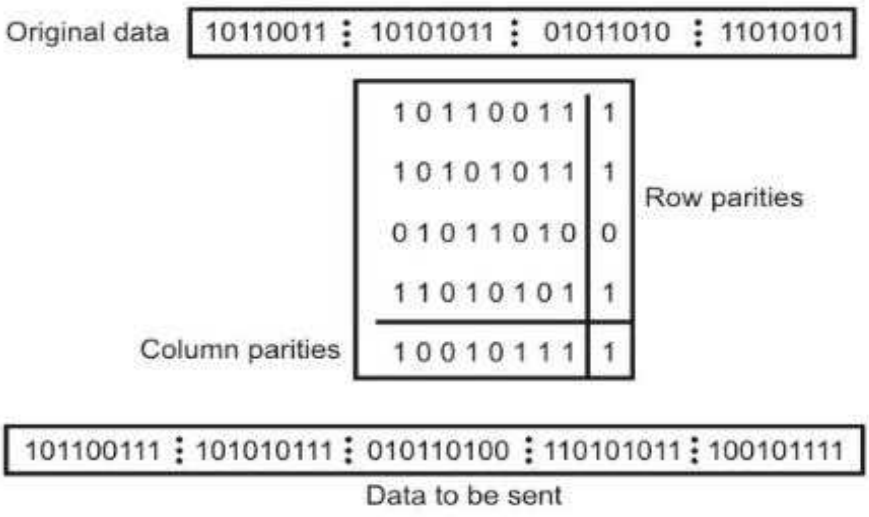


Considering a 4-bit word, different combinations of the data words and the corresponding code words are given in Table

Decimal value	Data Block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101
3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

- **Two-dimension Parity Check**

Performance can be improved by using two-dimensional parity check, which organizes the block of bits in the form of a table. Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check bits are also calculated for all columns then both are sent along with the data. At the receiving end these are compared with the parity bits calculated on the received data. This is illustrated in Fig.



- **Checksum**

In checksum error detection scheme, the data is divided into k segments each of m bits. In the sender's end the segments are added using 1's complement arithmetic to get the sum. The sum is complemented to get the checksum. The checksum segment is sent along with the data segments. At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented. If the result is zero, the received data is accepted; otherwise discarded. The checksum detects all errors involving an odd number of bits. It also detects most errors involving even number of bits.

Example:

k=4, m=8

```
  10110011
  10101011
 ----------
  01011110
        1
 ----------
  01011111
  01011010
 ----------
  10111001
  11010101
 ----------
  10001110
        1
 ----------
Sum :  10001111
Checksum 01110000
```

Example: Received data

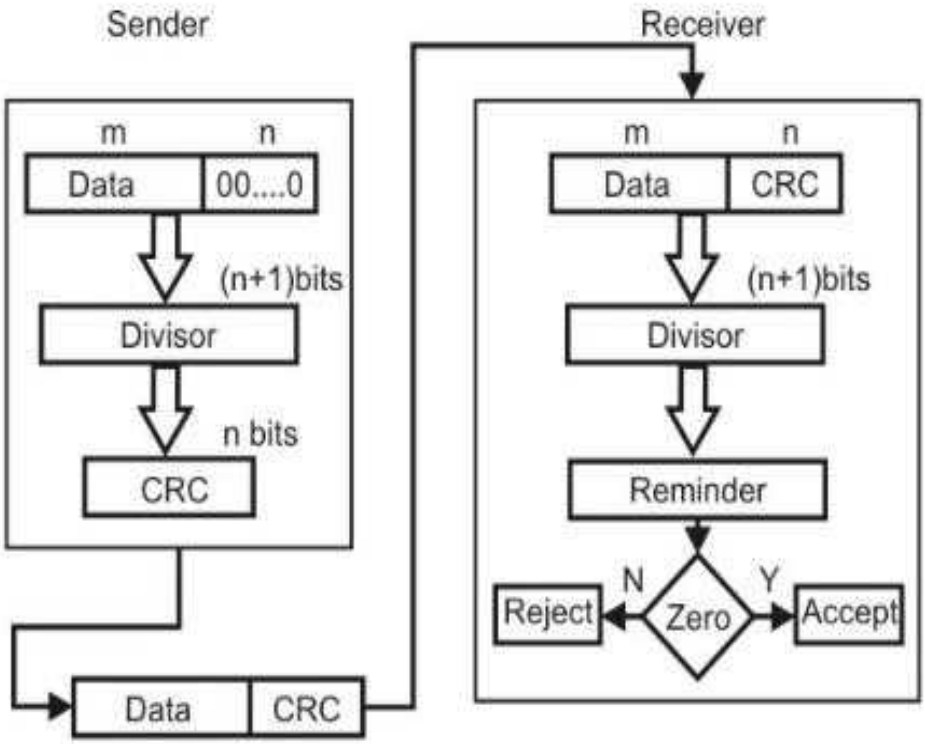
```
  10110011
  10101011
 ----------
  01011110
        1
 ----------
  01011111
  01011010
 ----------
  10111001
  11010101
 ----------
  10001110
        1
 ----------
  10001111
  01110000
 ----------
Sum: 11111111
Complement = 00000000
Conclusion = Accept data
```

- **Cyclic Redundancy Checks (CRC)**

This Cyclic Redundancy Check is the most powerful and easy to implement technique. Unlike checksum scheme, which is based on addition, CRC is based on binary division. In CRC, a sequence of redundant bits, called **cyclic redundancy check bits**, are appended to the end of data unit so that the resulting

data unit becomes exactly divisible by a second, predetermined binary number. At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted. A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

If a k bit message is to be transmitted, the transmitter generates an r-bit sequence, known as Frame Check Sequence (FCS) so that the (k+r) bits are actually being transmitted. Now this r-bit FCS is generated by dividing the original number, appended by r zeros, by a predetermined number. This number, which is (r+1) bit in length, can also be considered as the coefficients of a polynomial, called Generator Polynomial. The remainder of this division process generates the r-bit FCS. On receiving the packet, the receiver divides the (k+r) bit frame by the same predetermined number and if it produces no remainder, it can be assumed that no error has occurred during the transmission.



Consider the case where k=1101 represented by the polynomial

$k = x^3 + x^2 + 1$

Consider a generating polynomial $G(x) = x^3 + x + 1$ (1011)

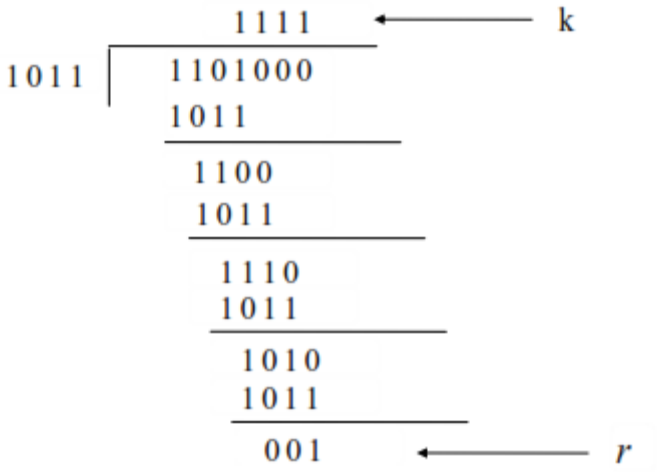
This is used to generate a 3 bit CRC = r to be appended to k.

Steps:

1. Multiply k by x^3 (highest power in G(x)). Add 3 zeros. 1101000
2. Divide the result by G(x). The remainder = r.

   1011 long division into 1101000 (with subtraction mod 2)

   = 1111 remainder 001



transmit: (k + r) = 1101000 + 001 = 1101001

Receiver end: Receive (k + r). Divide by G(x), should have remainder 0.

Example:

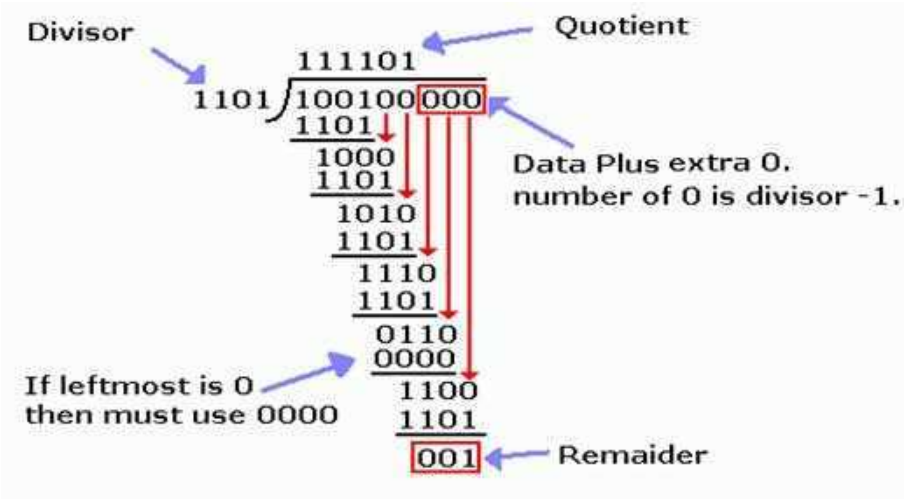
k= 100100 by the polynomial $k = x^5 + x^2$

$G(x) = x^3 + x^2 + 1$ (1101)

Add 3 zeros (the highest power in G(x)).. 100100000

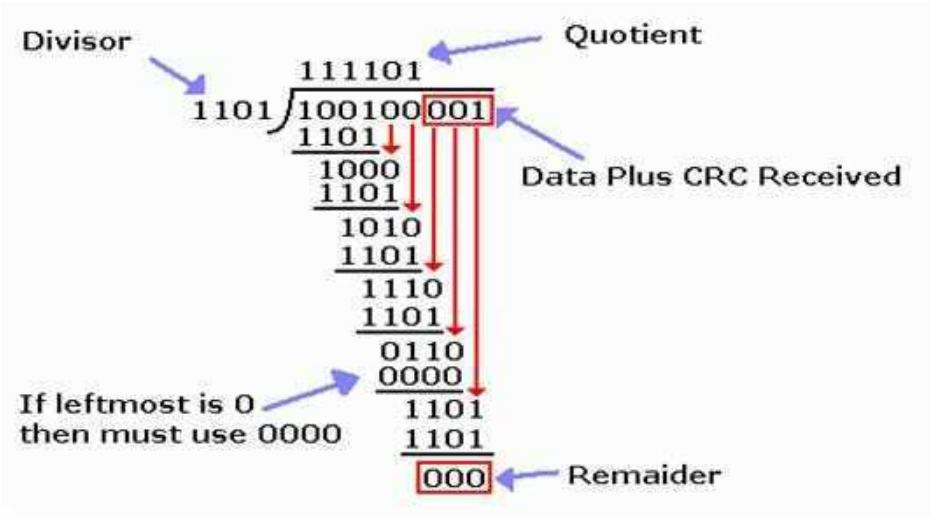
Divide the result by G(x). The remainder = r

1101 long division into 100100000 = 111101 remainder 001



transmit: k+r = 100100000 + 001 = 100100001

Receiver end: Receive (k+r). Divide by G(x), should have remainder 0.



Performance

CRC is a very effective error detection technique. Its performance can be summarized as follows:

CRC can detect all single-bit errors

- CRC can detect all double-bit errors
- CRC can detect any odd number of errors (X+1)
- CRC can detect all burst errors of less than the degree of the polynomial
- CRC detects most of the larger burst errors with a high probability.

Error Correcting Codes

This type of error control allows a receiver to reconstruct the original information when it has been corrupted during transmission. **Error Correction** can be handled in two ways.

- One is when an error is discovered; the receiver can have the sender retransmit the entire data unit. This is known as **backward error correction**.
- In the other, receiver can use an error-correcting code, which automatically corrects certain errors. This is known as **forward error correction**.

In theory it is possible to correct any number of errors atomically. Error-correcting codes are more sophisticated than error detecting codes and require more redundant bits. The number of bits required to correct multiple-bit or burst error is so high that in most of the cases it is inefficient to do so. For this reason, most error correction is limited to one, two or at the most three-bit errors.

Hamming Codes

In particular the ability to detect and correct errors called Hamming Codes.

Hamming code An error-detecting and error-correcting binary code, used in data transmission, that can

(a) detect all single- and double-bit errors and

(b) correct all single-bit errors.

In this method redundant bits are included with the original data. Now, the bits are arranged such that different incorrect bits produce different error results and the corrupt bit can be identified. Once the bit is identified, the receiver can reverse its value and correct the error. Hamming code can be applied to any length of data unit and uses the relationships between the data and the redundancy bits. Hamming Codes are still widely used in computing, telecommunication, and other applications including data compression.

A Hamming code satisfies the relation $2^m \geq n+1$. For any positive integer $m \geq 3$, there exists a Hamming code with the following characteristics:

The total number of bits in the block (length) $n = 2^m - 1$

Number of information bits in the block $k = 2^m - m - 1$

Number of parity check bits $n - k = m$

Error-correction capability $t = 1$, ($d$min = 3)

The parity check matrix $\boldsymbol{H}$ of these codes is formed of the non-zero columns of m bits, and can be implemented in systematic form:

$$\boldsymbol{H} = [\boldsymbol{Im}\ \boldsymbol{Q}]$$

where the identity submatrix $\boldsymbol{Im}$ is a square matrix of size $m \times m$ and the submatrix $\boldsymbol{Q}$ consists of the $2^m - m - 1$ columns formed with vectors of weight 2 or more.

Example : Determine the parity check matrix $\boldsymbol{H}$ for the linear block code $C_b(7, 4)$ generated by the generator matrix

$$G = \begin{pmatrix} 1\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,1\,\,0\,0 \\ 1\,1\,1\,0\,0\,1\,0 \\ 1\,0\,1\,0\,0\,0\,1 \end{pmatrix}$$

For the simplest case, for which $m = 3$,

$n = 2^3 - 1 = 7$

$k = 2^3 - 3 - 1 = 4$

$n - k = m = 3$ $\quad t = 1(d \text{ min} = 3)$

$$\boldsymbol{G} = \underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}}_{\text{submatrix } \boldsymbol{P}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{submatrix } \boldsymbol{I}}$$

the parity check matrix $\boldsymbol{H}$ is constructed using these submatrices:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

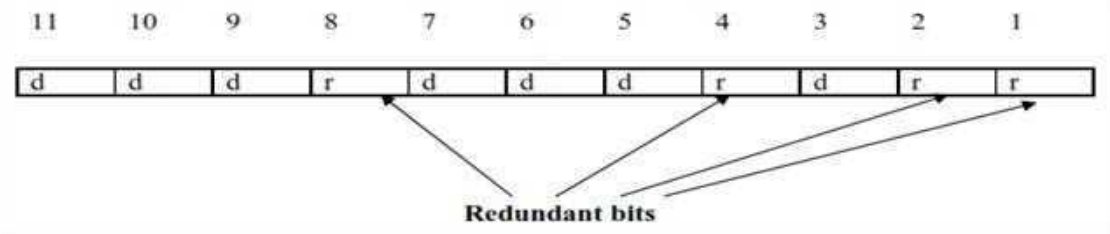
Single-bit error correction

Concept of error-correction can be easily understood by examining the simplest case of single-bit errors. A single-bit error can be detected by addition of a parity bit (VRC) with the data, which needed to be send. A single additional bit can detect error, but it's not sufficient enough to correct that error too. For correcting an error one has to know the exact position of error, i.e. exactly which bit is in error (to locate the invalid bits). For example, to correct a single-bit error in an ASCII character, the error correction must determine which one of the seven bits is in error. To this, we have to add some additional redundant bits.

To calculate the numbers of redundant bits (r) required to correct d data bits, let us find out the relationship between the two. So we have (d+r) as the total number of bits, which are to be transmitted; then r must be able to indicate at least d+r+1 different values. Of these, one value means no error, and remaining d+r values indicate error location of error in each of d+r locations. So, d+r+1 states must be distinguishable by r bits, and r bits can indicates 2^r states. Hence, 2^r must be greater than d+r+1.

2^r >= d+r+1

A technique developed by R.W.Hamming provides a practical solution. The solution or coding scheme he developed is commonly known as Hamming Code. Hamming code can be applied to data units of any length and uses the relationship between the data bits and redundant bits.



Basic approach for error detection by using Hamming code is as follows:

• To each group of m information bits k parity bits are added to form (m+k) bit code.

• Location of each of the (m+k) digits is assigned a decimal value.

• The k parity bits are placed in positions 1, 2, …, 2^{k-1} positions.–K parity checks are performed on selected digits of each codeword.

• At the receiving end the parity bits are recalculated. The decimal value of the k parity bits provides the bit-position in error, if any.

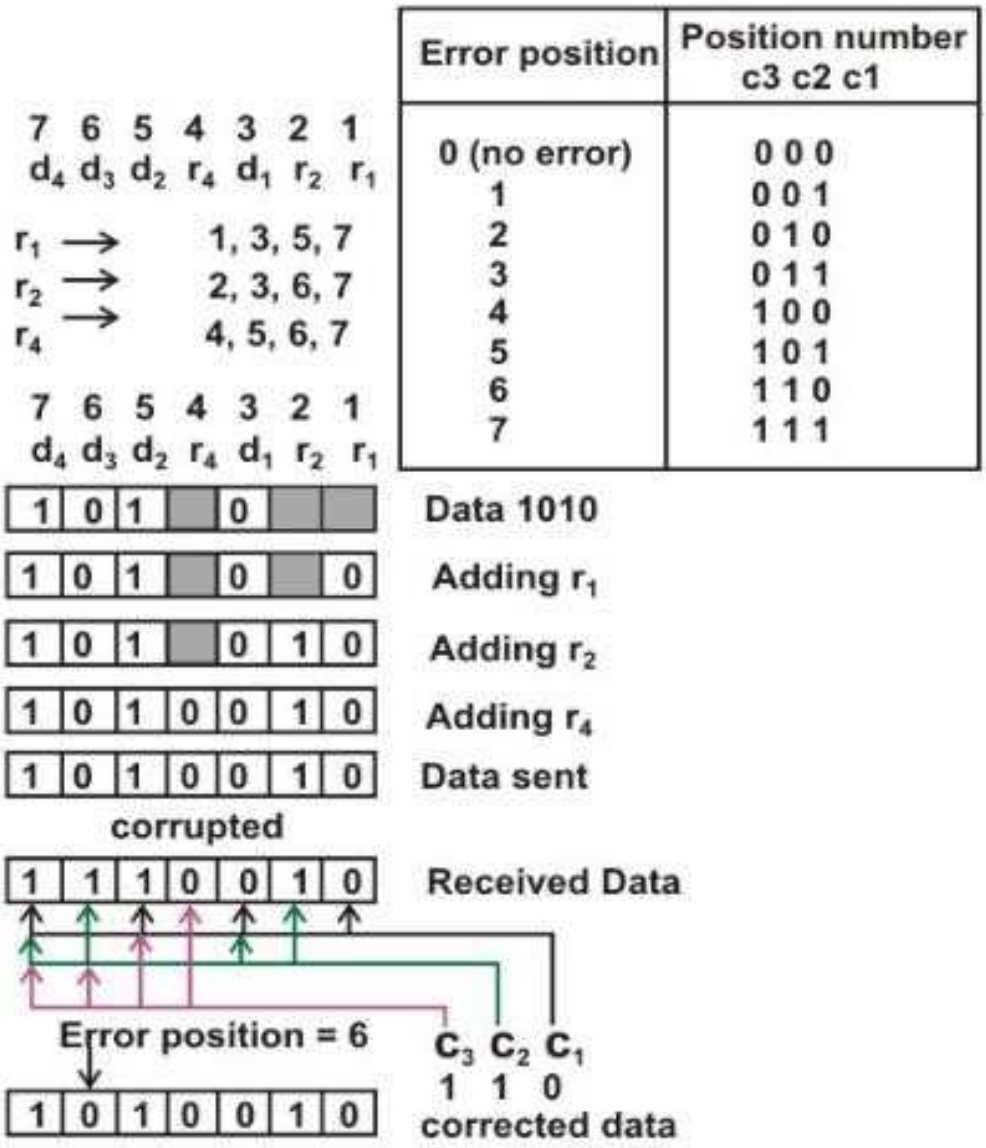
General algorithm

The following general algorithm generates a single-error correcting (SEC) code for any number of bits.

1. Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.
2. Write the bit numbers in binary. 1, 10, 11, 100, 101, etc.
3. All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits.
4. All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.
5. Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

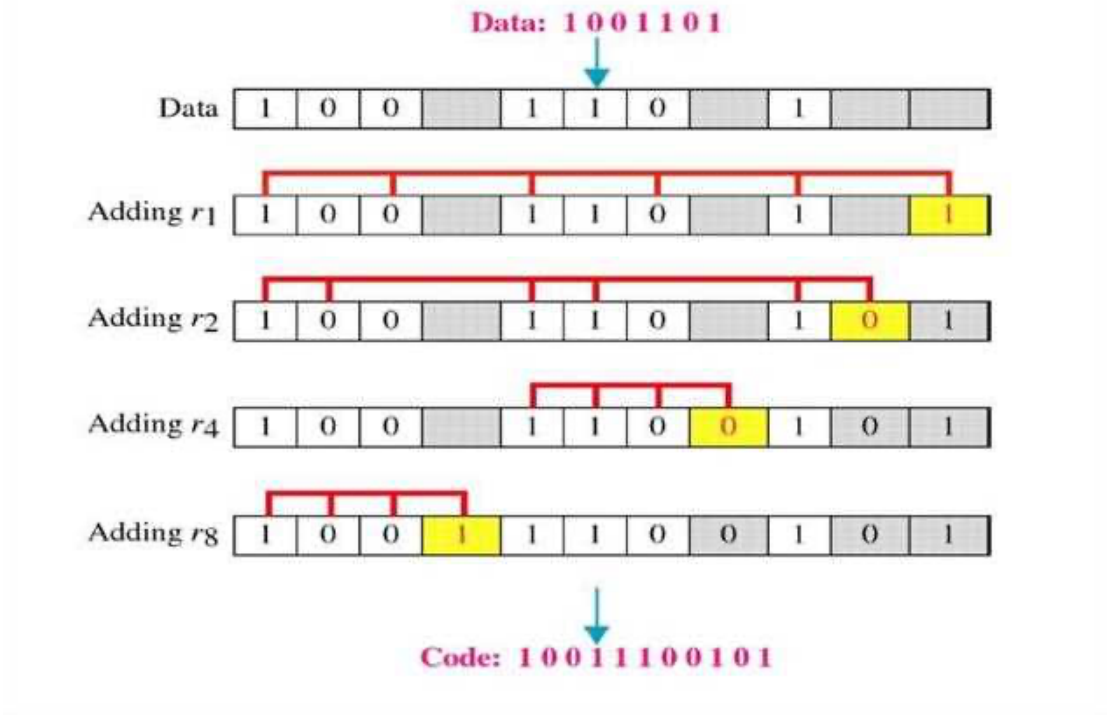
    1. Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
    2. Parity bit 2 covers all bit positions which have the second least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
    3. Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4–7, 12– 15, 20–23, etc.
    4. Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8–15, 24–31, 40–47, etc.

5. In general each parity bit covers all bits where the binary AND of the parity position and the bit position is non-zero.

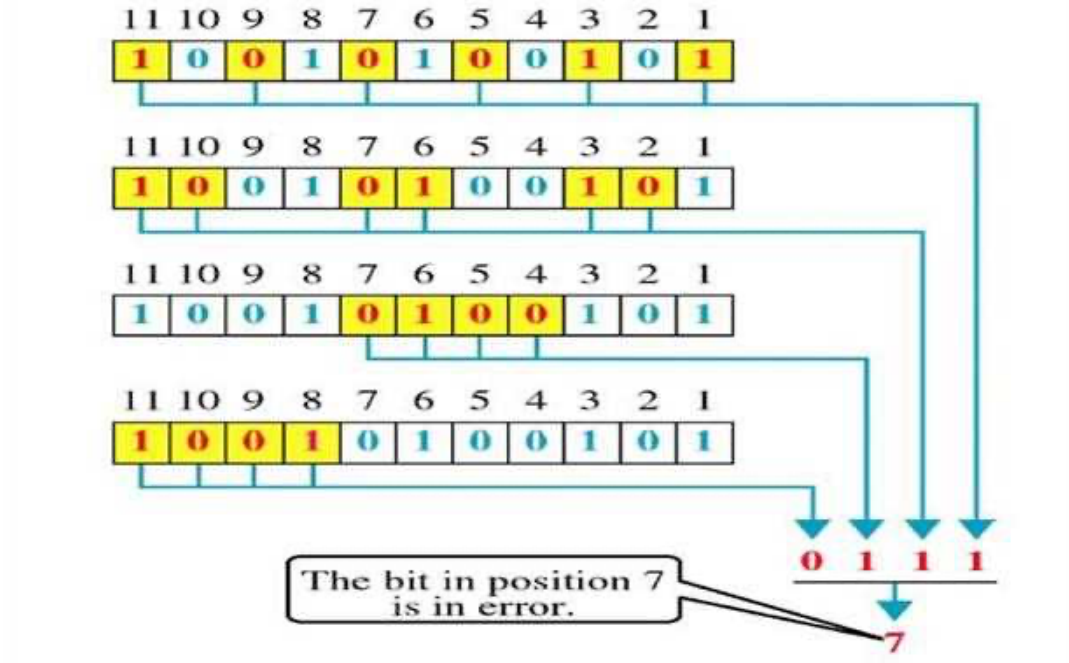


Error position	Position number c3 c2 c1
0 (no error)	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

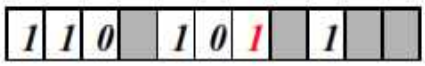
Example: data has been transmitted is 1001101



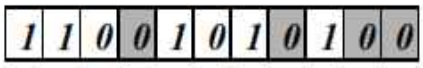
The code received as the form 10010100101



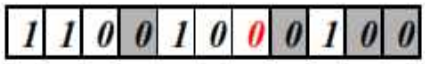
Example: Let us consider an example for 5-bit data. Here 4 parity bits are required. Assume that during transmission bit 5 has been changed from 1 to 0. The receiver receives the code word and recalculates the four new parity bits using the same set of bits used by the sender plus the relevant parity (r) bit for each set. Then it assembles the new parity values into a binary number in order of r positions (r8, r4, r2, r1).



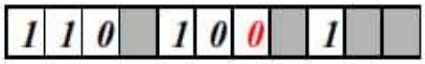
Data to be send



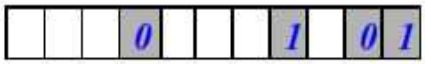
Data to be send along with redundant bits



Data Received



Data Received Minus Parity Bits



Parity bits recalculated

Example: A hamming code 1000110 is being received. Find the correct code which is being transmitted.

7	6	5	4	3	2	1
d4	d3	d2	r4	d1	r2	r1
1	0	0	0	1	1	0

P1= 1,3,5,7
P2= 2,3,6,7
P3= 4,5,6,7

C1=0
C2=1
C3=1
in reverse order 110, then error has occurred at position 6. The exact code transmitted is 1100110