Huffman Code:

To find Huffman code for binary coding, the following procedure can be made:-

1. Arrange messages in decreasing order of probability.
2. The two lowest probability messages are summed (joined) assigning one of them "0" and the other by "1". This sum will replace these two messages when messages are rewritten in a decreasing order.
3. Repeat step 2 many times until all messages are joined (summed) ending up with total sum of "1" unity.
4. Codeword for each message are read from left to right and writing the codeword bits from right to left.

Notes:-

1- For ternary Huffman Code, if the given number of messages is not odd, add a dummy message with probability of zero to make the number of message odd. So as to end up with three messages.

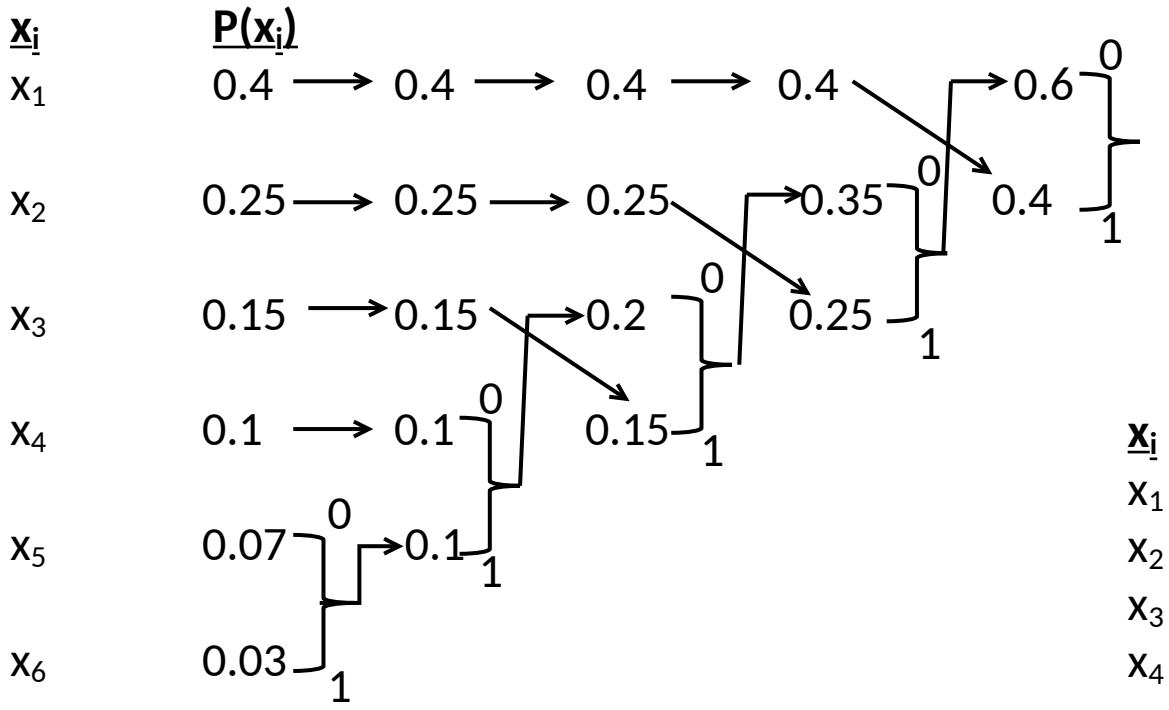
2- In each step, the three lowest probabilities are summed (joined) assigning them as “0”, “1” and “2”.

3- In any coding procedure, then the efficiency η is 100% if $P(x_i)=\left(\frac{1}{D}\right)^r$ for all $i= 1,2,3,\ldots\ldots n$

$D=2$ for binary coding and $D=3$ for ternary coding and r is an integer.

Example:- Develop Huffman Code for the following messages: **$P(x_i)$= [0.4 0.25 0.15 0.1 0.07 0.03]** then find the coding efficiency.

Solution:-



x_i	$P(x_i)$	Codeword c_i	L_i	0_i
x_1	0.4	1	1	0
x_2	0.25	01	2	1
x_3	0.15	001	3	2
x_4	0.1	0000	4	4
x_5	0.07	00010	5	4
x_6	0.03	00011	5	3

$Lc = 2.25$ bits/message and ***H(X)*** = 2.1918 bits/message

$$\rightarrow \eta = \frac{2.1918}{2.25\ Log22} = 97.4\%$$

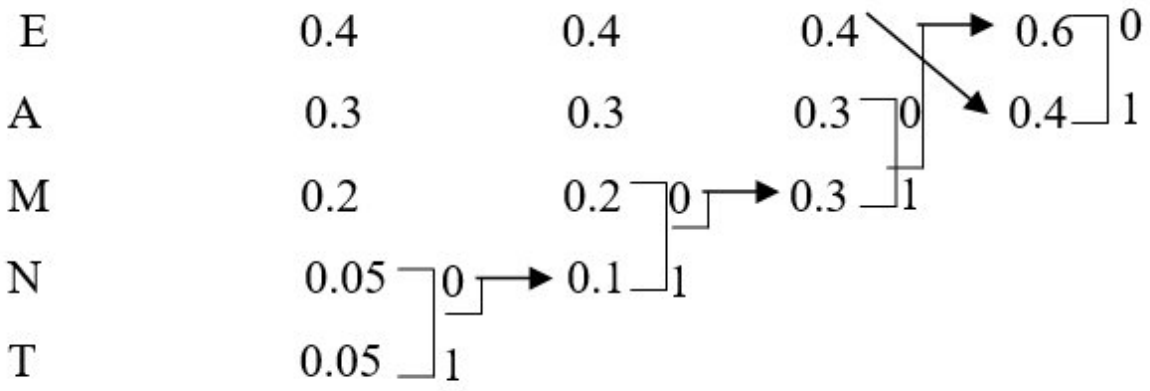
HW:- Develop Ternary Huffman Code for the following messages: **$P(x_i)$= [0.4 0.25 0.15 0.1 0.07 0.03]** then find:
a. Code efficiency ; b. P(0) , P(1) & P(2) at the encoder output

Example: Source symbols A, E, M, N and T with their probabilities 0.3, 0.4, 0.2, 0.05 and 0.05 respectively. Find:-

a) Huffman code for these symbols
b) Coding efficiency and redundancy
c) P(0) & P(1) at the encoder output
d) Decode the stream 01110001010010000110

Solution

a) Arrange symbols in decreasing order of prob



Symbol	P(symbol)	Codeword	Li	0i
E	0.4	1	1	0
A	0.3	00	2	2
M	0.2	010	3	2
N	0.05	0110	4	2
T	0.05	0111	4	1

b*)** ***Lc = L c = (0.4 x 1) + (0.3 x 2) + (0.2 x 3) + (0.05 x 4) + (0.05 x 4) = 2 bits/symbol and ***H(X)*** = 1.946 bits/ symbol

$$\rightarrow Efficency\ \eta = \frac{1.946}{2} = 0.973 = 97.3\% \quad and \text{ Redundancy } = 1-0.973 = 0.027 = 2.7\%$$

c) P(0) = ∑pi * 0i / Lc =(0+ 2*0.3+2*0.2 + 2*0.05+0.05)/2 = 0.575 , P(1) =1-0.575= 0.425

d) 0111 | 00 | 010 | 1 | 010 | 00 | 0110 = **TAMEMAN**
T A M E M A N

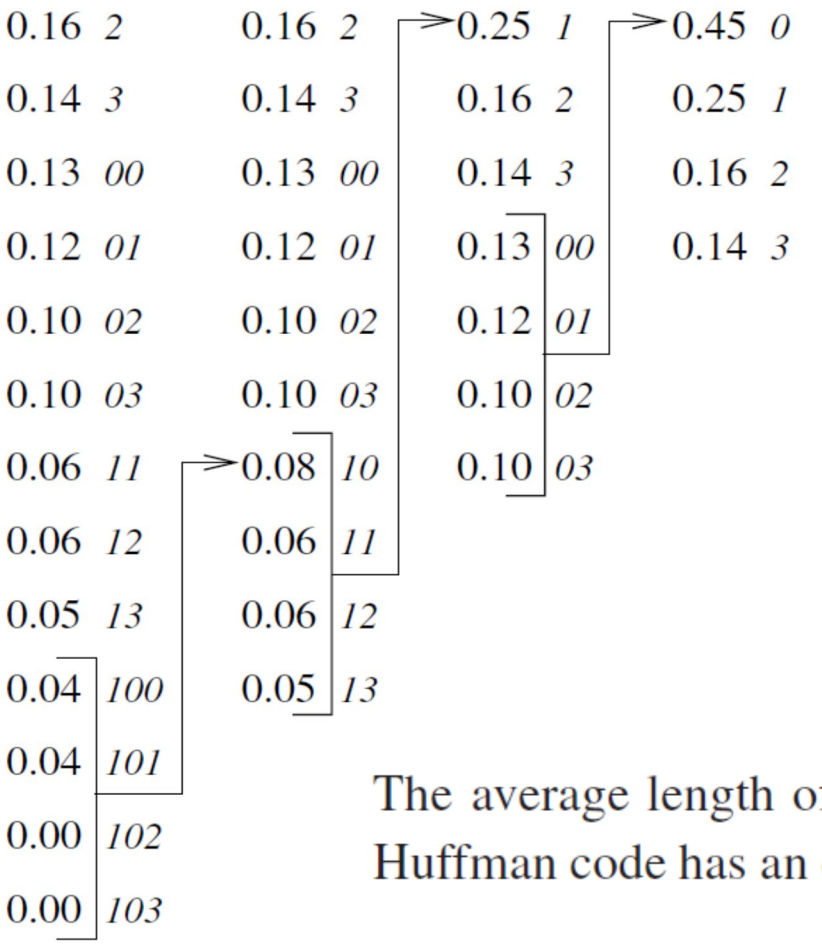
Example; Develop a quaternary (r=4) Huffman Code for the following messages: then find the coding efficiency.

$P(x_i)$= [0.14 0.12 0.10 0.10 0.13 0.16 0.05 0.04 0.04 0.06 0.06].

Solution : Arrange the probabilities in decreasing order and add two symbols with dummy prob.

$P(x_i)$= [0.16 0.14 0.13 0.12 0.10 0.10 0.06 0.06 0.05 0.04 0.04].

H(x)=p(x) log(p(x))= 1.654 4-ary digits per sec. $Lc = \bar{l}i = \sum_{i=1}^{n} li\ P(xi)$

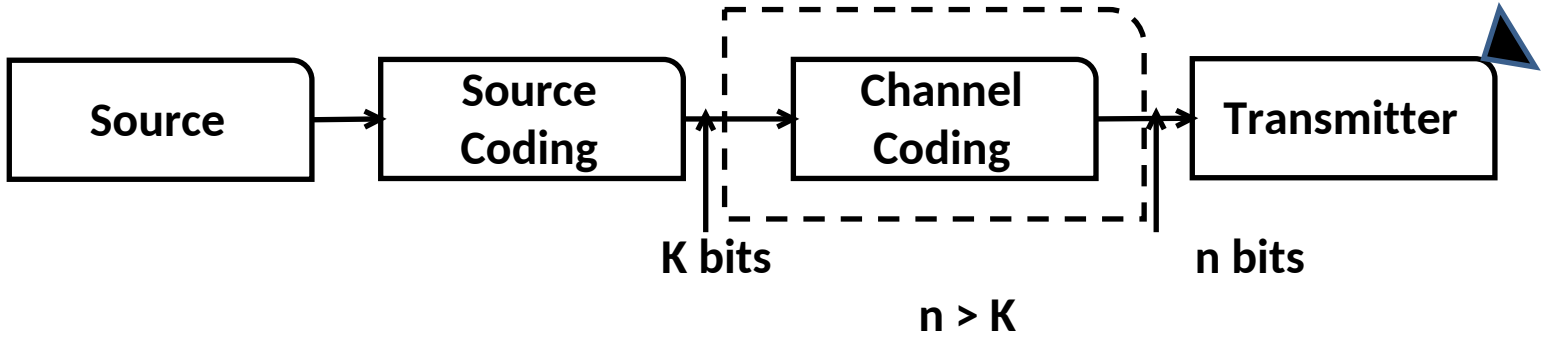


Symbol	$P(s_i)$	Huffman Code
s_1	0.16	2
s_2	0.14	3
s_3	0.13	00
s_4	0.12	01
s_5	0.10	02
s_6	0.10	03
s_7	0.06	11
s_8	0.06	12
s_9	0.05	13
s_{10}	0.04	100
s_{11}	0.04	101

The average length of the code is $L = 1.78$ quaternary units per symbol and Huffman code has an efficiency of $\eta = \frac{H_4(S)}{L} = \frac{1.654}{1.78} = 93\%$.

The purpose of channel coding is:-

- To protect information from channel noise, distortion and jamming which is the subject of error detection and correction codes.
- To protect information from 3rd party "enemy" which is the subject of encryption scrambling.

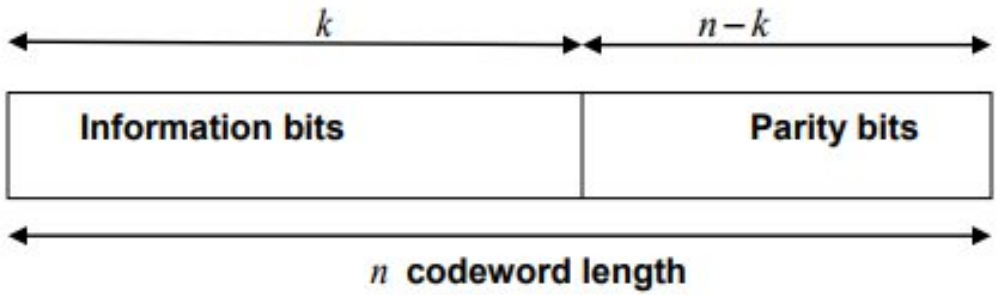


Error Detecting and Correcting Codes:

The basic idea behind these codes is to add **extra bits (digits)** to information such that the receiver can use it to detect and correct errors with limited capabilities, these extra bits are called **parity or check or correction bits**. If for ***k*** digits, ***r*** parity digits are added then the transmitted digits ***n = k + r*** *digits*. We will have ***r*** redundant digits and the code is called ***(n,k)*** code. ***k*** = No. of information bits. ***r*** = No. of check bits. ***n*** = No. of transmitted bits.

With code efficiency or rate of $\left(\frac{K}{n}\right)$.

In general, the ability of detection or correction depends on the technique used and ***n, k*** parameters.



Simple Error Detecting Codes:

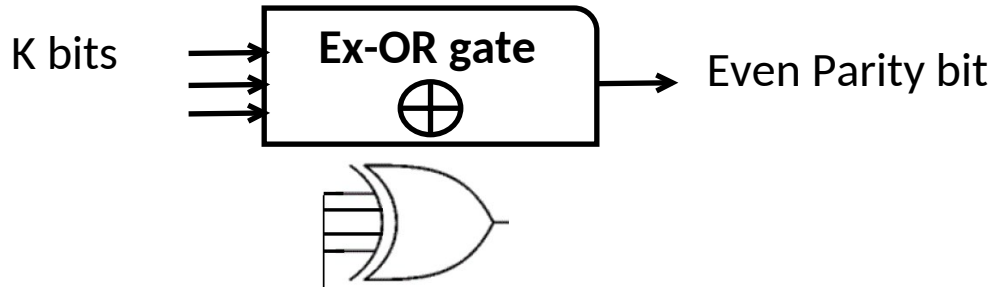
The simplest error detection schemes are the well-known ***even & odd parity generators.*** For even parity generators, an extra bit is added for each ***k*** information bits such that the number of ones is even. At the receiver, an error is detected if the number of ones is odd. However, if the number of 1's is even, either no error occurs or even number of errors occur. Hence,

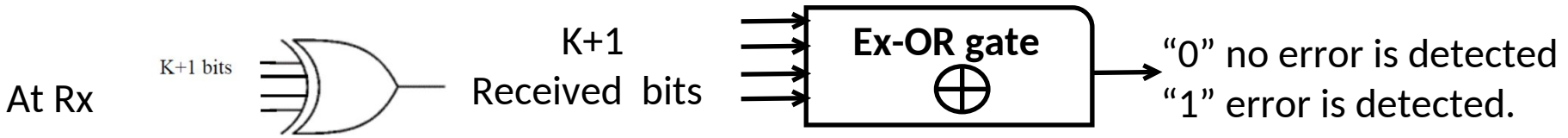
Prob. (detecting errors)=Prob. (odd number of errors).
Prob. (Undetecting errors)=Prob. (even number of errors).

The same idea can be applied when number of ones is adjusted to be odd which is called Odd parity generators. The code rate (efficiency) is $(\frac{K}{K+1})$. Where ***k*** = No. of information bits. ***r*** = No. of parity bits and ***n*** = ***r + K*** No. of transmitted bits.

To implement these parity generators, simple Ex-OR gates are used at the transmitter Tx and receiver Rx as shown :

Information bits	even parity
1010111	1
0110101	0
1011111	0
1101000	1





Error Correcting Codes:

In order to make the receiver have the ability to detect and correct errors, then not only a single parity bit is used, but instead ***r bits*** are used giving what is called **the (n,k) code**. Error Correcting Codes can be classified to **Block Codes** and **Convolutional Codes.**

Basic definitions:

1-systematic and nonsystematic codes: If information bits (a's) are unchanged in their values and positions at the transmitted codeword, then this code is said to be systematic.

Input data [D]=[a_1 a_2 a_3 …….a_k],

Output systematic (n,k) codeword is [C]=[a_1 a_2 a_3 …….a_k c_1 c_2 c_3 …….c_r]

However, if data bits are spread or changed at the output codeword then, the code is said to be nonsystematic:

For example, the output nonsystematic (7,4) codeword **is [C]=[c_2 a_1 c_3 a_2 c_1 a_4 a_3]**

2- Hamming distance: it is the number of differences between corresponding bits and the ability of error detection and correction codes depends on this parameter. The Hamming distance between two codewords C_i and C_j is denoted by d_{ij} which is the number of bits that differ. For a binary (n,k) code with 2^k possible codewords then the minimum Hamming distance (HD) is the min(d_{ij}). Also note that

$$n \geq d_{ij} \geq 0.$$

Example: Find the Hamming distance between the two codewords: [C1]=[1011100] and [C2]=[1011001].

Solution: Here, the no. of bits that differ is 2, hence $d_{12}=2$.

Example: Find the minimum Hamming distance for the 3 codewords:

[C1]=[1011100],

[C2]=[1011001]

[C3]=[1011000].

Solution: Here $d_{12}=2$, $d_{13}=1$ and $d_{23}=1$. Hence min(d_{ij})=1=(HD). Note that the calculation of HD becomes more difficult if number of codewords increases**.**

3-<u>Hamming Weight</u>: This is the number of 1's in the non zero codeward $\mathbf{C_i}$. It is denoted by $\wr_i$. As will be shown later, and for linear codes, $\wr_{min}$=**HD=min(d_{ij})**. This simplifies the calculation of **HD**. As an example, if [C1]=[1011000], then $\wr_1$=3, and for [C2]=[0001010], then $\wr_2$=2, and so on.

<u>4. Hamming Bound:</u> The purpose of Hamming bound is either
1) to choose the number of parity bits (***r***) so that a certain error correction capability is obtained. Or
2) to find the error correction capability (***t***) if the number of parity bits (***r***) is known
For binary codes, this is given by:

$$2^{n-k} = 2^r \geq \sum_{j=0}^{t} C_j n$$

where ***t*** is the number of bits to be corrected and ***C*** is the factorial factor $\frac{n!}{j!(n-j)!}$

<u>Example:</u> for a single correction code with ***k***=4 find the number of parity bits that should be added.

Solution: Using Hamming bound . This gives $2^r \geq C_0^{4+r} + C_1^{4+r} = 1+(4+r)$ and the minimum **r** is **r**=3 (take minimum **r** to have max code efficiency). This is the (7,4) code. the code is said to be **perfect code.**
<u>Perfect code</u>: in Hamming bound ,if the equality is satisfied then this code is said to be a perfect code.

<u>Example</u> if k=5 and up to 3 errors are to be corrected, find the no. of check bits that should be added.

Solution: that gives:
$2^r \geq 1+(5+r)+(5+r)(4+r)/2+(5+r)(4+r)(3+r)/6$, then min ***r*** here is **r**=9, and the code is the (14,5) non perfect code(equal sign is not satisfied).

Note: If the **(n,k)** codewords are transmitted through a channel having error prob=P_e, then prob. of decoding a correct word at the Rx for **t**-error correcting code will be:

P(correct words)=p(no error)+p(1 error)+.....p(t errors)

$$\text{P(correct words)} = \sum_{j=0}^{t} C_j n \quad Pe\, j \quad (1-Pe\,))n-j$$

and prob(erroneous word)=1-P(correct word).

Linear and nonlinear codes: when the **r** parity bits are obtained from a linear function of the **k** information bits then the code is said to be linear, otherwise it is a nonlinear code.

Hamming code:

The first example given above is the Hamming code. It is a **single error correcting perfect code** with the following parameters: $\mathbf{n=2^r-1}$, **HD=3, t=1**. The (7,4), (15,11), (31,26) .....are examples of Hamming codes. Hamming codes are encoded and decoded as a linear block codes.

Notes:

1-A linear code can correct **t**=Int[(HD-1)/2] of random (isolated) errors and detect (HD-1) random(isolated errors).

2- HD is the min Hamming distance= d_{min}

3- To guarantee correction of up to ***t*** errors in all cases, the minimum Hamming distance in a block code must be

$$HD_{min} = 2t + 1$$