Dr. VUK
SC-1.

Source Coding (SC) Module 2

→ The process by which the output of an information source is converted into a r-ary Seq. The task is acheived by 'source encoder'.

r-ary sequence: $r=2$ (binary); $r=3$ (ternary); $r=4$ (quarternary) etc

Let 'S' be the source alphabet consist of 'q' message symbols

$$\Rightarrow \quad S = \{s_1, s_2, \dots s_q\}$$

Let another alphabet 'X' called 'code alphabet' consist of r number of coding symbols given by

$$X = \{x_1, x_2, \dots, x_r\}$$

The term "coding" can now be defined as transformation of the source symbols into some sequences of symbols from code alphabet-X

(A) **properties of Codes:** A block code is a code which maps each of the symbols of the source alphabet S into some "finite sequences" of code symbols from the code alphabet X and each of these finite sequences is called a "code-word".

Eg., $S = \{s_1, s_2, s_3, s_4\}$ & $X = \{0, 1\}$

A block code can now be constructed as:

example: 1 ⇒ $s_1 \to 00$, $s_2 \to 01$, $s_3 \to 10$, $s_4 \to 11$, s_1, s_2, s_3 & $s_4 \to$ source symbols
00, 01, 10, 11 → code words.

(CODE-A)

(B) **Non Singular Code:** A block code is said to be "non singular" if & only if all the code words are distinct & easily distinguishable from one another.

e.g. $s_1 \to 00$, $s_2 \to 01$, $s_3 \to 10$, and $s_4 \to 11$ are non singular codes.

Example:

S	codes
s_1	0
s_2	00
s_3	01
s_4	11

×

000 → $s_1 s_1 s_1$ or $s_1 s_2$ or $s_2 s_1$

code B → Singular code. not uniquely decodable

(C) **Uniquely Decodable codes:** A non-singular code is said to be uniquely decodable or uniquely decipherable if every code word present in a long received sequence can be uniquely identified.

Defn: A block code is said to be "uniquely decodable" if and only if the n^{th} extension codewords of the code is non singular for every finite value of 'n'.

Instantaneous codes: A uniquely decodable code is said to be "instantaneous" if it is possible to recognize the end of any code in any received sequence, without reference to the succeeding symbols i.e., there is no time delay in the process of decoding is decoding is done instantaneously as and when the symbol arrives at the receiver.

Ex:

		FLC	VLC	VLC
	source symbols	code C	code D	CODE E
	S_1	00	0	0
	S_2	01	10	01
	S_3	10	110	011
	S_4	11	1110	0111

Let the received Sequence be 001100

If Code-C is used then it is decoded as $S_1 S_4 S_1$

If code-D is used then it is decoded as $S_1 S_{[illegible]} S_3 S_1$

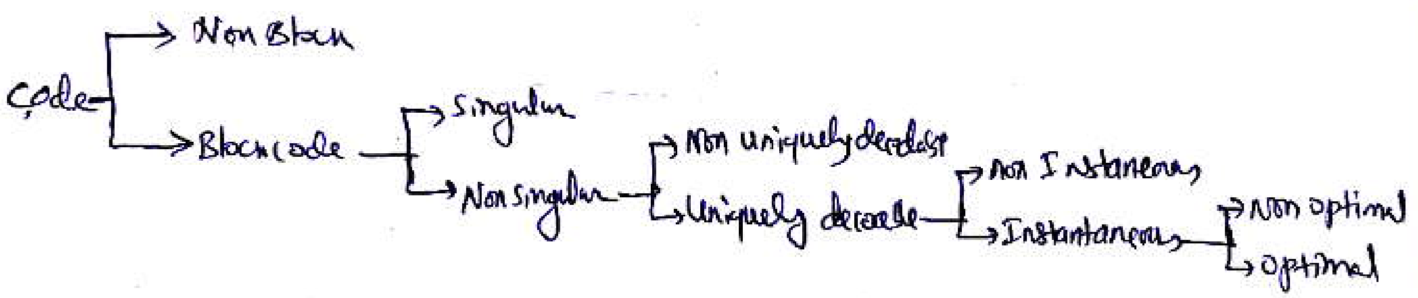
If Code-E is used " " " " " $S_1 S_3 S_1 S_1$

For C & D decoding can be done instanteneously whereas for 'E' should wait ~~for~~ from starting 0 & next 0.

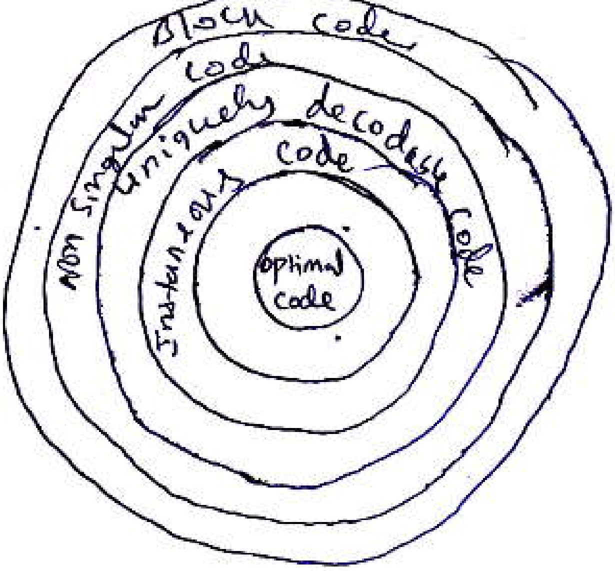
∴ code E in not an Instantaneous code.

Optimal Codes: An instantaneous code is said to be "optimal code" if it has "minimum average length L" for a source with given probability assignment for the source symbols.

Code property Tree Diagram



Circle Diagram:



Prefix of a Codes: Let $X = x_1, x_2, x_3 \cdots x_m$ be a codeword of some code. Then the sequences $x_1, x_2 \cdots x_j$ for all $j \leq m$ are called "prefixes" of code X.

Example: Let $X = \underset{x_1}{0} \; \underset{x_2}{1} \; \underset{x_3}{1} \; \underset{x_4}{1}$ $\Rightarrow m = 4$

When $j=1$ → code is 0
$j=2$ → " " 01
$j=3$ → " " 011
$j=4$ → " " 0111

These are the all possible prefixes of code X.

0 is the prefix of 01, ill'y 01 is prefix of 011, & 011 is the prefix of 0111. ⇛ complete code word must be prefix of other

Example 2:

0
10
110
1110

⇒ no word is prefix of other

1,11, 111 are not complete codewords.

Test for instantaneous property (Prefix property)

A necessary and sufficient condition for a uniquely decodable code to be instantaneous is that "No complete word of a code be a prefix of any other codeword.

Kraft-Inequality or Kraft Mcmillan Inequality

A necessary and sufficient condition for the existance of an instantaneous code with word lengths $l_1, l_2, \ldots l_q$ is that-

$$\sum_{i=1}^{q} r^{-l_i} \leq 1.$$

$X = \{0, 1\} \Rightarrow r = 2$
$S = 00, 01, 10, 11$
$\Rightarrow q = 4, \; l = 2$ (const)
$l_1 = l_2 = l_3 = l_4$

where $r \rightarrow$ no. of different symbols used in the code alphabet- X.

l_i = word length in binary digits of the code-word corresponding to its source symbols

q = no. of source symbols

Proof: The word lengths $l_1, l_2, \ldots l_q$ are arranged in the ascending order so that-

$$l_1 \leq l_2 \leq l_3 \leq \ldots \leq l_q$$

If we have to choose "One length" codewords for all the 'q' number of source symbols satisfying prefix property, we can do only when $q \leq r$

e.g. $X = \{0, 1, 2, 3\} \Rightarrow r = 4$

then $q \leq 4$ $\quad s_1 = 0, \; s_2 = 1, \; s_3 = 2 \;\&\; s_4 = 4$

4-symbol of one length

when $q > r$ then we have to go for combinations of r symbols

e.g. $X = \{0, 1\} \Rightarrow r = 2$

$q = 4 \Rightarrow s_1, s_2, s_3$, and s_4

$\Rightarrow$ 00, 01, 10, & 11 for FLC

or 0, 10, 110, 111 etc.

Let n_i represent the number of messages encoded into code words of length "i". Then, we have

for $i = 1$ $\quad n_1 \leq r$

for $i = 2$, for getting an instantaneous code, we must start encoding using $(r - n_1)$ symbols only as

.c.-3 as the first digit and the second digit can be any of r symbols

for i=2 $n_2 \le (r-n_1)\times r$

$S_1 \to 0$ $X=\{0,1\}$

zero is already used then $r-n_1$

$(2-1)\times 2 = 1\times 2$

2 symbols with '1' and length '2'

i.e., 10 & 11

or $n_2 \le r^2 - n_1 r$

i=3 |||ly $n_3 \le (r^2 - n_1 r - n_2)\times r$

or $n_3 \le r^3 - n_1 r^2 - n_2 r$

⑥ In general

$n_i \le r^i - n_1 r^{(i-1)} - n_2 r^{i-2} \cdots - n_{(i-1)} r$

$\therefore n_i + n_1 r^{i-1} + n_2 r^{i-2} + \dots + n_{i-1} r \le r^i$

Multiplying throughout by r^{-i} we get

$n_i r^{-i} + n_1 r^{-1} + n_2 r^{-2} + \dots + n_{i-1} r^{-(i-1)} \le 1$

Writing the above in the reverse way, we get

$n_i r^{-i} + n_{i-1} r^{-(i-1)} + n_{(i-2)} r^{-(i-2)} + \dots + n_1 r^{-1} \le 1$

or $\sum_{m=1}^{i} n_m r^{-m} \le 1$ — (1)

Since the actual no. of messages n_i has to be interger, we can re-write equation (1) as

$$\sum_{m=1}^{i} n_m r^{-m} = \underbrace{r^{-1} + r^{-1} + r^{-1} + \dots r^{-1}}_{n_1 \text{ terms}} + \underbrace{r^{-2} + r^{-2} + \dots r^{-2}}_{n_2 \text{ terms}} + \dots$$

$$\underbrace{r^{-i} + r^{-i} + r^{-i} + \dots r^{-i}}_{n_i \text{ terms}} \le 1$$

$$= \underbrace{\sum_{j=1}^{n_1} r^{-1}}_{\text{length } l_1} + \underbrace{\sum_{j=1}^{n_2} r^{-2}}_{\text{length } l_2} + \dots + \underbrace{\sum_{j=1}^{n_i} r^{-1}}_{\text{length } l_q} \le 1$$

Combining all groups {since $n_1 + n_2 + n_3 + \dots n_i = q$}, we can write

$\sum_{i=1}^{q} r^{-l_i} \le 1$ proved

For $r=2$ (binary)

$$\sum_{i=1}^{q} 2^{-l_i} \le 1.$$

Example: Considering 5 different codes F, G, H, I & J .

source symbols	codeF	code G	codeH	codeI	codeJ
s_1	00	0	0	0	0
s_2	01	100	10	100	10
s_3	10	110	110	110	110
s_4	11	111	111	11	11

For code F: The word lengths in bits of the code words for s_1, s_2, s_3 and s_4 are given by $l_1 = 2$ bits $= l_2 = l_3 = l_4$

substituting these values in KMI.

$$\sum_{i=1}^{q} 2^{-l_i} = 2^{-l_1} + 2^{-l_2} + 2^{-l_3} + 2^{-l_4} = 2^{-2} + 2^{-2} + 2^{-2} + 2^{-2} = 1.$$

Kraft inequality is satisfied with the equality sign.

Code G: $l_1 = 1$, $l_2 = l_3 = l_4 = 3$ bits

$$\therefore \sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-3} + 2^{-3} + 2^{-3} = 7/8 < 1$$

$\Rightarrow$ code G satisfies Kraft inequality. Therefore, with the words-lengths $1 \to l_1$, $3 \to l_2, l_3$, and l_4, it is possible to construct an instaneous code

code H: $l_1 = 1$, $l_2 = 2$, $l_3 = l_4 = 3$ bits

$$\therefore \sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 1$$ satisfies KMI

$\Rightarrow$ instanteneous code is possible

codeI: $l_1 = 1$, $l_2 = l_3 = 3$ bits & $l_4 = 2$ bits

$$\Rightarrow \sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-3} + 2^{-3} + 2^{-2} = 1$$

satisfies KMI but it is not instantaneous as $s_4 \Rightarrow 11$ is prefix of s_3. Therefore codewords must be readjusted.

c-4 code-J: $l_1 = 1, l_2 = 2, l_3 = 3$ and $l_4 = 2$ bits.

$$\therefore \sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-2} = 1\tfrac{1}{8} > 1.$$

code J does not satisfy KMI. Hence instantaneous code with these lengths is not possible.

Example: Consider the four codes listed below in table 3.7, Identify the instantaneous codes and construct their individual decision trees.

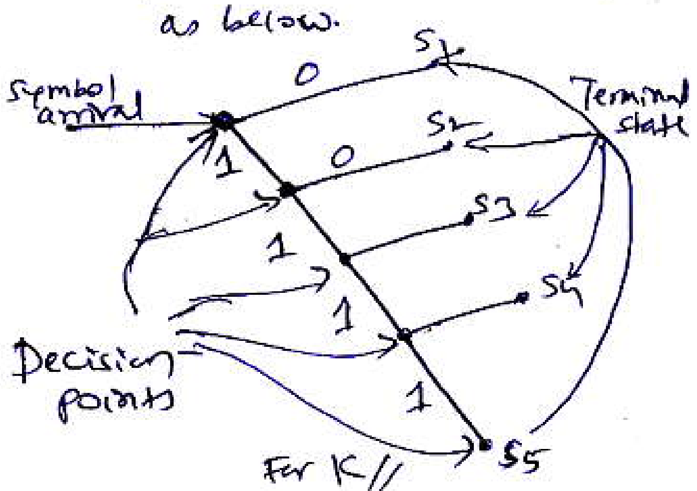
Source Symbols	Code-K	Code L	Code M	Code N
S1	0	0	0	00
S2	10	01	01	01
S3	110	001	011	10
S4	1110	0010	110	110
S5	1111	0011	111	111

K & N are instantaneous as the prefixes of any code word are not present as the code-words of the remaining symbols.

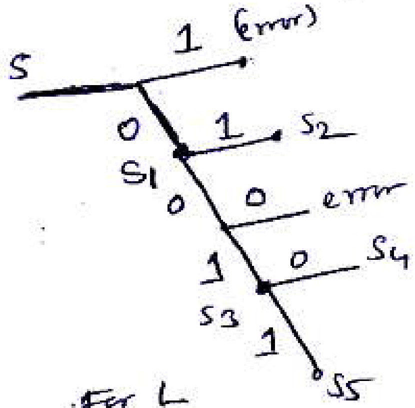
But code L - is not not instantaneous, since the code for s_1 is a prefix of codes for S_2, S_3, S_4 and S_5

|||ly code-M is also not instantaneous since the code for s_1 is a prefix of codes for S_2 & S_3 and code for S_2 is a prefix of code for S_3.

The individual decision tree (code-tree) can be constructed as below.



For K//



For L

Example: Which of the following sets of word-lengths specified following table are acceptable for the existence of an instantaneous code given $X=\{0,1,2\}$

Number of words of word length l_i			word length l_i
Code - P	code - Q	code - R	
2	2	1	1
1	2	4	2
2	2	6	3
4	3	0	4
1	1	0	5

trinity

Code P: It is given that $X=\{0,1,2\}$ $\therefore r=3$

from KMI: $\sum_{i=1}^{9} r^{-l_i} \le 1$

$= 2\times 3^{-1} + 1\times 3^{-2} + 2\times 3^{-3} + 4\times 3^{-4} + 1\times 3^{-5}$

$= 0.90535 < 1$ ✓

Code Q: $\sum_{i=1}^{9} r^{-l_i} = 2\times 3^{-1} + 2\times 3^{-2} + 2\times 3^{-3} + 3\times 3^{-4} + 1\times 3^{-5}$

$= 1.0041 > 1$ ✗

code R: $\sum_{i=1}^{9} r^{-l_i} = 1\times 3^{-1} + 4\times 3^{-2} + 6\times 3^{-3} + 0\times 3^{-4} + 0\times 3^{-5}$

$= 1$. It is possible.

Example: Find the minimum number of symbols, 'r' in the coding alphabet for devising an instantaneous code such that $W=\{0,5,0,5,5\}$. device such a code

W→ represents the set of code words of lengths 1, 2, 3...

Soln: For the given problem table is shown below

Number of words of length l_i (W)	word length l_i
0	1
5	2
0	3
5	4
5	5

From the table we observe that there are five codewords of length 2, 5 codewords of length 4, 5 code words of length 5

$$\Rightarrow \sum_{i=1}^{15} r^{-l_i} = (r^{-2}+r^{-2}+r^{-2}+\cdots r^{-2}) + (r^{-4}+r^{-4}+\cdots+r^{-4}) + (r^{-5}+r^{-5}+\cdots+r^{-5})$$

5 source symbols with code length 2 — for 5 source symbols with codeword length 4 — For 5 source symbols with code word length 5.

$$= 5r^{-2}+5r^{-4}+5r^{-5} \le 1$$

Let $r=2$, then $5\times2^{-2}+5\times2^{-4}+5\times2^{-5} = 1.71875 > 1$

KMI is <u>not</u> satisfied

Let $r=3 \Rightarrow 5\times3^{-2}+5\times3^{-4}+5\times3^{-5} = 0.63706 < 1$

KMI is satisfied

$\therefore r=3. \quad \therefore X=\{0,1,2\}$

<u>One possible code:</u>

s_1	00	5	~~s_8~~	1202	5
s_2	01		s_9	~~1003~~ 2000	
s_3	02		s_{10}	2001	
s_4	10		s_{11}	21000	5
s_5	11		s_{12}	21001	
s_6	1200		s_{13}	21002	
s_7	1201		s_{14}	21010	
			s_{15}	21011	

<u>Example:</u> Consider a binary block code with 2^n code-words of same length n. Show that the Kraft inequality is satisfied for such a code

<u>Soln:</u> 2^n codewords — $l_i = n$

$\therefore$ no. of source symbols $= q = 2^n$ with $r=2$ for binary code

Then LHS of KMI

$$\sum_{i=1}^{q} r^{-l_i} = \sum_{i=1}^{2^n} 2^{-n} = \underbrace{2^{-n}+2^{-n}+\cdots+2^{-n}}_{2^n \text{ terms}}$$

$\therefore \quad 2^{-n}\times 2^{n}=1$ $\quad\therefore$ KMS is satisfied with equality sign

Considering 2 specific cases for n

i) Let $n=2$; $\quad q=2^2=4$

$l_i = n = 2$ for $i=1,2,3,4$

The instantaneous code for this case is shown in table.

source symbols	code for $n=2$
s_1	00
s_2	01
s_3	10
s_4	11

ii) Let $n=3 \Rightarrow q = 2^3 = 8$; $l_i = n = 3$, for $i = 1, 2, \dots 8$

$\Rightarrow s_1 \cdots s_8 = 000 - 111.$

Code Efficiency & Redundancy :

$$L = \sum_{i=1}^{q} p_i l_i \quad \text{bits/symbol}$$

$$\& \; H(S) = \sum_{i=1}^{q} p_i \log \frac{1}{p_i} \quad \text{bits/symbol}$$

Always $L \geqslant H(S)$ for binary
& $L \geqslant H_r(S)$ for r-ary $\qquad H_r(S) \to$ r-ary entropy

$$\boxed{\text{or} \quad H_r(S) = \frac{H(S)}{\log_2 r}}$$

$$\therefore \eta_c = \text{code efficiency} = \frac{H(S)}{L}$$

$$\eta_c(\%) = \eta_c \times 100 \qquad \eta_c \leq 100\%.$$

$$\text{Code Redundancy} = R_{\eta c} = 1 - \eta_c \qquad \times 100 \text{ for } \%$$

Source coding theorem (shannon's first theorem or Shanon's fundamental theorem, or noiseless coding theorem)

Let X be the ensemble of letters ~~to~~ from a DMS with finite entropy $H(X)$ and the output symbols $x_k, k = 1, 2, \dots L$ occurring with probabilities $P(X_k)$ &, $k = 1, 2, \dots L$. It is possible to construct a code that satisfies the prefix condition and has an average length $\bar{R}$ that satisfies the inequality

$$H(X) \leq \bar{R} \leq H(X) + 1$$

Proof: First consider the lower bound of the inequality. For codewords that have length n_k for $1 \leq k \leq L$,

$n = l \Rightarrow n_k = l_k$

sc-6 The difference $H(X) - \bar{R}$ can be expressed as

$$H(X) - \bar{R} = \sum_{k=1}^{L} P(x_k) \log_2 \frac{1}{P(x_k)} - \sum_{k=1}^{L} P(x_k) \cdot n_k$$

$$= \sum_{k=1}^{L} P(x_k) \cdot \log_2 \left(\frac{2^{-n_k}}{P(x_k)} \right)$$

We now make use of log inequality $\ln x \leq x - 1$

$$H(X) - \bar{R} \leq \log_2 e \sum_{k=1}^{L} P(x_k) \left(\frac{2^{-n_k}}{P(x_k)} - 1 \right)$$

$$\leq \log_2 e \left(\underbrace{\sum 2^{-n_k}}_{\leq 1 \text{ (from KMI)}} - 1 \right) \leq 0$$

<u>upper bound</u>: choosing codeword lengths n_k such that

$$2^{-n_k} \leq P(x_k) < 2^{-n_k+1}$$

First consider $2^{-n_k} \leq P(x_k)$

summing both sides over $1 \leq k \leq L$

$$\sum_{k=1}^{L} 2^{-n_k} \leq \sum_{k=1}^{L} P(x_k) = 1$$ as it satisfies KMI

for which there exists a code satisfying the prefix condition.

Next consider $P(x_k) < 2^{-n_k+1}$

take 'log' on both sides

$$\log_2 P(x_k) < -n_k + 1$$

or $n_k < 1 - \log_2 P(x_k)$

on multiplying both sides by $P(x_k)$ and summing over $1 \leq k \leq L$ we obtain

$$\sum_{k=1}^{L} n_k \cdot P(x_k) < \underbrace{\sum_{k=1}^{L} P(x_k)}_{1} - \underbrace{\sum_{k=1}^{L} P(x_k) \cdot \log_2 (P(x_k))}_{H(S)}$$

$$\Rightarrow \quad \bar{R} < H(S) + 1$$

//

Shanon's First Encoding Theorem

Shanon has proposed ~~of~~ one of the first algorithms on source encoding. The procedure is

step 1: Given the source alphabet 's' and the corresponding probabilities

2. Arrange the probabilities in the non-increasing order

3. Compute the length l_i for the codeword corresponding to each symbol S_i given by

$$l_i \geq \log_2 \frac{1}{P_i} \quad \text{where } P_i \text{ is the prob. of symbol } s_i$$

4. Define the following parameters from the probability set

$$q_1 = 0$$
$$q_2 = P_1 = q_1 + P_1$$
$$q_3 = P_1 + P_2 = q_2 + P_2 \qquad n = N$$
$$q_4 = P_1 + P_2 + P_3 = q_3 + P_3$$
$$\vdots$$
$$q_{n+1} = P_1 + P_2 + P_3 + \cdots P_n = q_n + P_n = 1$$

where n is the total number of symbols in the source alphabet

5. Expand q_i in binary till 'l_i' number of places after decimal pt.

6. The numbers after decimal places in the binary repres^n of q_i are the codewords for the corresponding symbol S_i

Example 1: Let $S = \{A, B, C, D\}$ with $P = \{0.1, 0.2, 0.3, 0.4\}$ use Shanon's first Encoding theorem for coding

Sol^n

Step 1: $P = \{0.4, 0.2, 0.2, 0.1\} \Rightarrow S = \{D, C, B, A\}$

2: Find the minimum value of l_i such that

$$l_i \geq \log_2 \frac{1}{P_i}$$

$$\Rightarrow \quad l_1 \geq \log_2 \frac{1}{P_1} = \log_2 1/0.4 \Rightarrow l_1 = 2$$
$$l_2 \geq \log_2 \frac{1}{P_2} = \log_2 \frac{1}{0.3} \Rightarrow l_2 = 2$$
$$l_3 \geq \log_2 \frac{1}{P_3} = \log_2 \frac{1}{0.2} \Rightarrow l_3 = 3$$
$$l_4 \geq \log_2 \frac{1}{P_4} = \log_2 \frac{1}{0.1} \Rightarrow l_4 = 4$$

calculating the parameters q_i

$$q_1 = 0$$
$$q_2 = P_1 = q_1 + P_1 = 0.4$$
$$q_3 = P_2 + P_1 = 0.4 + 0.3 = 0.7$$
$$q_4 = P_3 + P_2 + P_1 = 0.4 + 0.3 + 0.2 = 0.9$$
$$q_5 = P_1 + P_2 + P_3 + P_4 = 0.4 + 0.3 + 0.2 + 0.1 = 1$$

Step4: Represent $q_1, q_2, q_3,$ and q_4 in binary upto l_i places after decimal points:

$q_1 = (0.0)_{10} = (0.00)_2$ $\quad l_i = l_1 = 2 \quad i=1$

$q_2 = (0.4)_{10} = (0.01)_2$ $\quad l_i = l_2 = 2 \quad i=2$

$q_3 = (0.7)_{10} = (0.101)_2$ $\quad l_i = l_3 = 3 \quad i=3$

$q_4 = (0.9)_{10} = (0.1110)_2$ $\quad l_i = l_4 = 4 \quad i=4$

∴ The codewords for the symbols using Shanon's algorithm are

symbol	probability	codeword	length(l_i)
D	0.4	00	2
C	0.3	01	2
B	0.2	101	3
A	0.1	1110	4

Step5: To find the efficiency, we have

$$\eta_s = \frac{H(S)}{L} \times 100$$

$$\therefore H(S) = \sum_{i=1}^{n} P_i \log \frac{1}{P_i} = 1.8464 \text{ bit/sym}$$

and $L = \sum_{i=1}^{n} P_i l_i = (0.4\times2) + (0.3\times2) + (0.2\times3) + (0.1\times4) = 2.4$ bit/sym

$$\therefore \eta_s = \frac{1.8464}{2.4} \times 100 = 76.93\%.$$

$$R_{\eta_s} = 1 - \eta_s = 23.07\%.$$

Example 2: Consider a discrete memoryless source with $S = (X, Y, Z)$ with corresponding probabilities $P = (0.5, 0.3, 0.2)$

a) Find the codewords for the symbols using shanon's algorithm. Also, find the source efficiency and redundancy.

b) Consider the second order extension of the source. Recompute the codewords and the efficiency.

step1 $\quad S = (X, Y, Z) \quad P = (0.5, 0.3, 0.2)$ Already in non increasing order

step2. $\quad l_i \geq \log_2 \frac{1}{P_i} \Rightarrow l_1 = 1,\ l_2 = 2,$ and $l_3 = 3$.

step3. calculate parameter $q_i \Rightarrow q_1 = 0,\ q_2 = 0.5,\ q_3 = 0.8,\ q_4 = 1$

step4: $\quad q_1 = (0.0)_{10} \Leftrightarrow (0.0)_{(2)}$; $\quad q_2 = (0.5)_{10} \Rightarrow (0.10)_{(2)}$

$q_3 = (0.8)_{10} \Leftrightarrow (0.110)_{(2)}$

∴ Codewords are

			prob
X	— 0	$l_i = 1 (l_1)$	0.5
Y	— 10	$l_i = 2 (l_2)$	0.3
Z	— 110	$l_i = 3 (l_3)$	0.2

$$\therefore \eta = \frac{H(S)}{L} \times 100 = \frac{1.4854}{1.7} = 87.37\% \Rightarrow R_s = 1 - \eta_s = 12.63\%.$$

(b) Second order Extension:

Symbols	Prob.
XX	0.25 (0.5×0.5)
XY	0.15
XZ	0.10
YX	0.15
YY	0.09
YZ	0.06
ZX →	0.10
ZY →	0.06
ZZ →	0.04

$\Sigma P_i = 1.$

⇒ ∴ P(0.25, 0.15, 0.15, ~~0.10~~, 0.10, 0.09, 0.01, 0.06, 0.04)
non increasing order

⇒ S (XX, XY, YX, XZ, ZX, YY, YZ, ZY, ZZ)

Decimal	Binary (up to l_i places)
$q_1 = 0.0$ →	$0.00_{(2)}$
$q_2 = 0.25$ →	$0.010_{(2)}$
$q_3 = 0.40$ →	$0.011_{(2)}$
$q_4 = 0.55$ →	$0.1000_{(2)}$
$q_5 = 0.65$ →	$0.1010_{(2)}$
$q_6 = 0.75$ →	$0.1100_{(2)}$
$q_7 = 0.84$ →	$0.1101_{(2)}$
$q_8 = 0.90$ →	$0.11100_{(2)}$
$q_9 = 0.96$ →	$0.11110_{(2)}$

Efficiency $\eta_s = \frac{H(S)}{L} \times 100$

$$\Rightarrow H(S^2) = \sum_{i=1}^{n} P_i \log \frac{1}{P_i} = 2.9709 \text{ bits/sym} = 2 \times H(S)$$

$$\& \ \bar{L} = \sum_{i=1}^{n} P_i l_i = 3.36 \text{ bits/sym}$$

$$\therefore \eta_s = \frac{2.9709}{3.36} = 88.41\%$$

$$\& \ R_{\eta s} = 1 - \eta_s = 11.59\%$$

Shannon-Fano Encoding Algorithm:

Step 1: Arrange the probabilities in the non increasing order

Step 2: Group the probabilities in to exactly two sets such that the sum of probabilities in both groups is almost equal. Assign bit '0' to all elements of the first group and bit 1 to all elements of group 2.

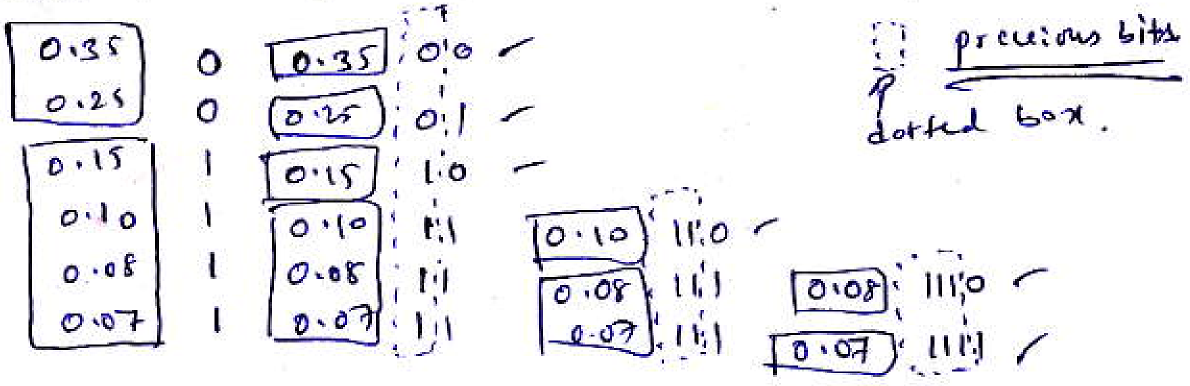
Step 3: Repeat step 2 by dividing each group ~~into~~ in two subgroups till no further division is possible.

Shannon-Fano:

c-8 Example: $S = \{A, B, C, D, E, F\}$; $P = (0.10, 0.15, 0.25, 0.35, 0.08, 0.07)$

- Find the codewords for the source using Shann-Fano algorithm.
- Also find the source efficiency and redundancy.

Soln, Arrange the probabilities in "non increasing order".



$$\therefore \eta_s = \frac{H(S)}{L} \times 100 = \frac{2.33}{2.4} = 97.08\%$$

and $R_s = 1 - \eta_s = 2.92\%$.

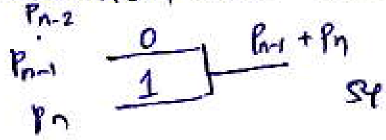
Huffman Codes (compact-codes):

=> Achieve minimum codword length amongst all other coding algorithms. => Hence these are compact codes.

R-base.

Algorithm: i) Arrange the source symbols in decreasing order of their probabilities.

ii) Take the bottom two symbols and tie them together as shown



=> Add the probabilities of the two symbols and write it on the combined node

=> Label the two branches with a '1' and a '0' as depicted in

iii) Treat this sum of probabilities as a new probability associated with a new symbol. Again pick the two smallest probabilities, tie them together to form a new probability. Each time we perform the combination of two symbols we reduce the total no. of symbols by one. Whenever we tie together two probabilities (nodes) we label the two branches with a '1' and a '0'.

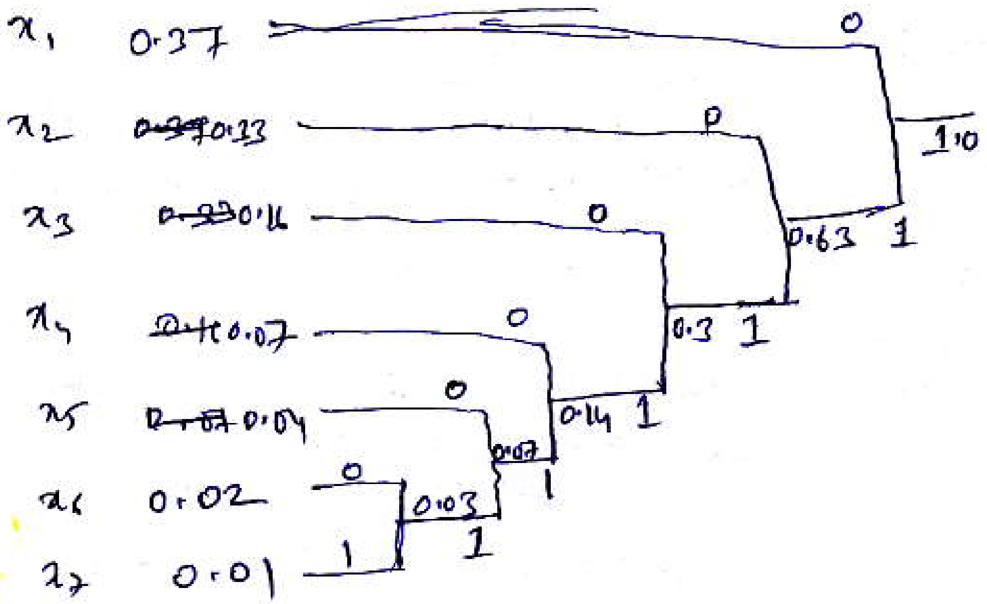
(iv) Continue the procedure until only one probability is left (and it should be 1 at the end). This complete the construction of Huffman code

(v) To find out the prefix codeword for any symbol, follow the br
from the final node back to the symbol. While tracing back the route, read out the labels on the branches. This is the codeword for the symbol.

Example:

Symbol	probability	self info bits/sym	codeword
x_1 →	0.37 →	1.4344 →	0
x_2 →	0.33 →	1.5995 →	10
x_3 →	0.16 →	2.6439 →	110
x_4 →	0.07 →	3.8365 →	1110
x_5 →	0.04 →	4.6439 →	11110
x_6 →	0.02 →	5.6439 →	111110
x_7 →	0.01 →	6.6439 →	111111

$$H(X) = \sum_{k=1}^{7} p(x_k).\log\frac{1}{p(x_k)} = 2.1152 \text{ bits.}$$



$$\bar{R} = \sum_{k=1}^{7} n_k p(x_k) = 1\times0.37+2\times0.33+3\times0.16+4\times0.07+$$
$$5\times0.02+6\times0.01 = 2.17 \text{ bits.}$$

$$\therefore \eta = 2.1152/2.17 = 97.47\%.$$

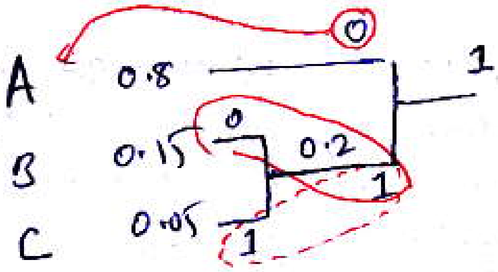
$$\text{Red} = 1-\eta = 2.53\%.$$

02, 07, 08, 09, 11, 13, 14, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29,
33, 34, 36 39, 45, 46, 48, 50, 55, 59, 60, 61, 65

Abmo
13/9

Extended Huffman coding:

Let $S = (A, B, C)$ with probabilities $P = (0.8, 0.15, 0.05)$



code words are:

		l_i
A →	0	1
B →	10	2
C →	11	2

$$\bar{L} = 1\times0.8 + 2\times0.15 + 0.05\times2$$

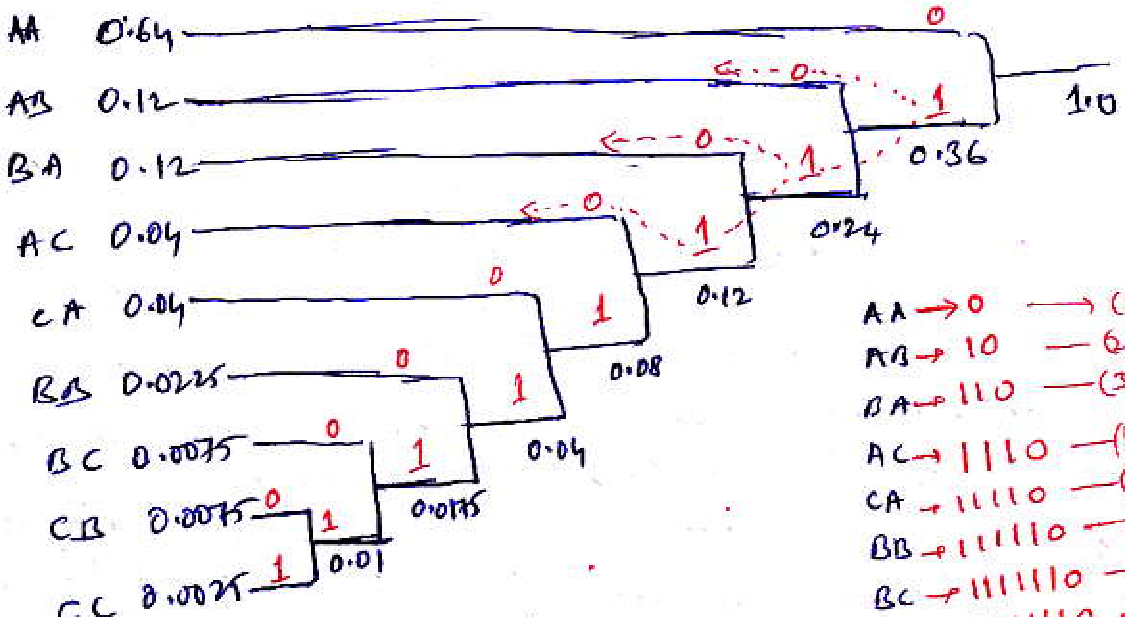
$$= 0.8 + 0.3 + 0.1 = 1.2$$

$H(S) = 0.88$ $\therefore$ $\eta = \frac{H(S)}{L} = \frac{0.88}{1.2} = 73.33\%$

Efficiency can be increased further after extending the source. ⇒ 2nd extension.

AA → P(A). P(A)
→ 0.8 × 0.8 = 0.64

AA → 0.64 BA → 0.12 CA → 0.04
AB → 0.12 BB → 0.0225 CB → 0.0075
AC → 0.04 BC → 0.0075 CC → 0.0025



AA → 0 —→ (1)
AB → 10 — (2)
BA → 110 — (3)
AC → 1110 — (4)
CA → 11110 — (5)
BB → 111110 — (6)
BC → 1111110 — (7)
CB → 11111110 — (8)
CC → 11111111 — (8)

$H(S^2) = 2H(S)$
$= 2\times0.88 = 1.76$

$\therefore \eta = \frac{1.76}{1.8675}$

$= 94.24\%$

improved!

$$\bar{L} = 1\times0.64 + 2\times0.12 + 3\times0.12 + 4\times0.04 + 5\times0.04 + 6\times0.0225 + 7\times0.0075 + 8\times0.0075 + 8\times0.0025$$

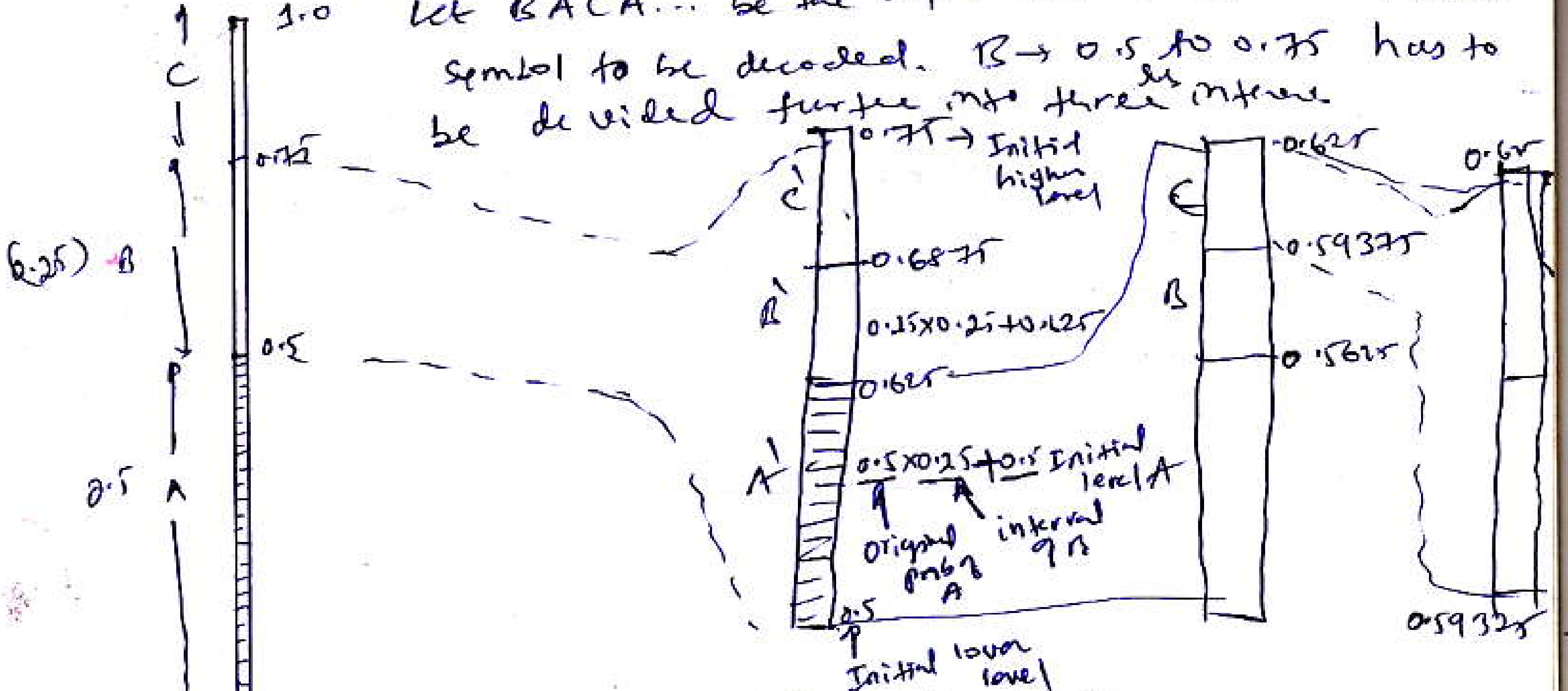
$$= 0.64 + 0.24 + 0.36 + 0.16 + 0.2 + 0.135 + 0.0525 + 0.06 + 0.02 = 1.8675$$

Airthmatic Coding: Huffman codes are only optimal if the Probabilities of the symbols are negative powers of 2. This is due to the fact that all prefix codes worn at bit level.

Airthmatic coding does not have this restriction. It works by representing the file to be enclosed by an interval of real numbers between 0 & 1. Successive symbols in the message reduce this interval in accordance with the probability of that symbol. The more likely symbols reduce the range by less, and thus add fewer bits to the message.

Example: Let A, B, and C be the three symbols with probabilities P(A) = 0.5, P(B) = 0.25, and P(C) = 0.25. We first devide the interval (0, 1) in to three intervals proportional to their probabilities. ∴ A → 0 to 0.5, B → 0.5 to 0.75 and C → 0.75 to 1.0

Let BACA... be the input symbols. 'B' is the first symbol to be decoded. B → 0.5 to 0.75 has to be devided further into three intervals



⇒ 0.59375 is the code.

At the receiver when 0.59375 is received it will look at the interval where it lies i.e., 0.59375 is the segment B ∴ first coded will be taken as B. Then this segment is further divided as A, B, and C. Then again 0.59375 is ~~compared~~ looked where it lies. ~~Obv~~ Obviously is in the A segment. Hence next symbol is A. The process is contd. But to stop some algorithm is required.

Lempel-Ziv Algorithm:

→ Earlier source coding techniques required the statistics of the symbols a priori.

→ However, in real-time situations, the symbol probabilities would be unknown.

→ Also, most of the information sources used in the real time applications do posses memory. i.e, symbol emitted in the present instant depends on previous symbol emitted.

→ Therefore, efficiency of such codes degrade.

→ This method uses correlations amongst symbols in source coding

→ LZ algorithm is a dictionary based approach in which a common dictionary is built at both sender and receiver end depending upon the string to be sent.

→ The code word will be of the form <index, code> where index is the index of the dictionary where the longest match for the input sequence is found, code is the codeword for the symbol succeding the longest match.

→ Index '0' is used if the symbol is encountered for the first time and not found in dictionary. The dictionary is updated in accordance with the input string. Longer the string to be encoded, better will be the compression is achieved.

Example: Encode the following information using LZ algorithm.
THIS_IS_HIS_HIT

Sol^n: Step1: Dictionary will be initially empty. The first letter in the input string is 'T' which is appearing for the first time. Thus an index '0' followed by the code of 'T' is transmitted. Dictionary is updated with the first entry 'T' at index 1.

Step2: Similarly, since the succeding symbols 'H', 'I', 'S', and _ appear for the first time, index '0' is used for encoding and the corresponding entries are made in the dictionary.

Step3: The next symbol in the input string is 'I' is already present in the dictionary.

Hence, the algorithm checks for the next symbol in the string, i.e., 'S' for a better match. But the string 'IS' is not present in the updated dictionary. Thus, it is encoded as <index I, code S> and the string 'IS' is updated in the dictionary.

Step4: The next symbol in the input string is '_' that is present in location 5. However; the next symbol succeding in the String is 'H' resulting in '_H' that is not yet updated in the dictionary. Thus, the string '_H' is codeded as <index(_), code(H)> and '_H' is updated in the next location of the dictionary.

Step5: The next symbol in the string is 'I' that is present in the dictionary. ~~Thus~~ the succeding symbol is 'S' resulting in 'IS' that is also present in the dictionary. Thus, the algorithm takes the next symbol '_' for a better match resulting in 'IS_' that is not present. Thus, the coded sequence would be <index(IS), code(_)> and the string IS_ is updated in the dictionary.

steps will be repeated for all further symbols

Dictionary in LZ algorithm | Encode string

Index	Symbol	Symbol	Encoded output
1	T	~~<0, code(T)>~~ T	<0, code(T)>
2	H	H	<0, code(H)>
3	I	I	<0, code(I)>
4	S	S	<0, code(S)>
5	_	_	<0, code(_)>
6	IS	IS	<3, code(S)>
7	_H	_H	<5, code(H)>
8	IS_	IS_	<6, code(_)>
9	HI	HI	<2, code(I)>
10	T_	T_	<1, code(_)>