Ques:-

$P(a_1)$	0.1
$P(a_2)$	0.3
$P(a_3)$	0.25
$P(a_4)$	0.35

(I) Fun
a.

(II) S

①	a_4	0.35
	a_2	0.30
	a_3	0.25
	a_1	0.10

(III)

②	a_4	0.35	
	a_3'	0.35	$a_1 \rightarrow 1$
	a_2	0.30	$\rightarrow 1$

$a_3 \rightarrow a_1 0$
$a_1 \rightarrow a_1 1$

③	a_3''	0.65	a_2 0
	a_1	0.35	1

(IV)

$a_3' \rightarrow a_2 0 \rightarrow 00$
$a_2 \rightarrow a_2 1 \rightarrow 01$

But $a_3' \rightarrow a_1$
$\Rightarrow a_1 \rightarrow 00$

$a_1 \rightarrow 001$ $a_3 \rightarrow 000$
$a_2 \rightarrow 01$ $a_4 \rightarrow 1$

$l = 3 \times 0.1 + 2 \times 0.3 + 3 \times 0.25 + 1 \times 0.35$

$= 0.3 + 0.6 + 0.75 + 0.35$

$= 2$ bits/symbol

$H = -[0.1 \log_2(0.1) + 0.3 \log_2(0.3) + 0.25 \log_2(0.25) + 0.35 \log_2(0.35)]$

$= -[0.1 \times (-3.321928) + 0.3 \times (-1.736966) + 0.25 \times (-2) + 0.35 (-1.5145731)]$

$= -[-0.3321928 - 0.5210088 - 0.50 - 0.53010061]$

$= 1.883388$ bits/sym

Redundancy $= 2 - 1.883388$

≈ 0.1166 bits/symbol

$$H = \sum_{i=1}^{n} P_i \log P_i$$

$$= -[0.4 \log_2(0.4) + 0.2 \log_2(0.2) + 0.2 \log_2(0.2) + 0.1 \log_2(0.1) + 0.1 \log_2(0.1)]$$

$$= -[0.4(\log_2 4 - \log_2 10) + 0.4(\log_2 2 - \log_2 10) + 0.2(\log_2 1 - \log_2 10)]$$

$$= -[0.4(2 - 3.32192) + 0.4(1 - 3.32192) + 0.2(0 - 3.32192)]$$

$$= -[0.4 \times (-1.32192809) + 0.4 \times (-2.32192) + 0.2 \times (-3.32192)]$$

$$= -[0.528771038 - 0.928768 - 0.664384]$$

$$= 2.129223238$$

Redundancy = $H - L$

$= 2.129223238 - 2.2$

$\simeq 0.07707$ bits/symbol.

of size J. Each block is then coded using one of the following options —

(I) Fundamental Sequence → Number n is represented by a sequence of n 0s followed by a 1.

(II) Split Sample Options → code for a k-bit number n using m^{th}-split option consits of the m least significant bits of k-bit number n followed by a unary code representing the k-m most significant bits.

(III) Second Extension Option → In the second extension Option the sequence is divided into consecutive pairs of samples. Each paire is used to obtain an index γ using the following transformation —

$$\gamma = \frac{1}{2}(x_i + x_{i+1})(x_i + x_{i+1} + 1) + x_{i+1}$$

Value of γ is encoded using a unary code. The value of γ is an index to a look-up table with each value of γ corresponding to a pair of x_i, x_{i+1}.

(IV) Zero-block Option → The zero block option is used when one or more of the blocks x_i are zero.

Golumb Code → L-16

→ Golumb code is based on the assumption that the larger an integer, the lower is its probability of occurence. The simplest code for this situation is unary code. One The unary code for a positive integer n is simply n is followed
10. Other way of coding is to split the integer into two parts; representing one part with unary code and other part with a different code.

Golumb code is parameterized by an integer $m>0$. In the golumb code with parameter m, we represent an integer $n>0$ using two numbers q and r, where

$$q = \left\lfloor \frac{n}{m} \right\rfloor \quad \text{and} \quad r = n - qm$$

The quotient q can take on values $0, 1, 2, \ldots$ and is represented by unary code of q. The remainder r can take on the values $0, 1, 2, \ldots m-1$. If m is a power of two, we use $\log_2 m$ bit binary representation of r. If m is not a power of two, ~~we use $\lceil \log_2 m \rceil$ bits. To reduce the no of bits required.~~ we use $\lfloor \log_2 m \rfloor$ bit binary representation of r for first $2^{\lceil \log_2 m \rceil} - m$ values, and $\lceil \log_2 m \rceil$ bit binary representation of $r + 2^{\lceil \log_2 m \rceil} - m$ for rest of the values

Ex→ Design a golumb code for $m=5$.

Here, $m=5$

$\Rightarrow \lceil \log_2 5 \rceil = 3, \quad \lfloor \log_2 5 \rfloor = 2$

First $2^{\lceil \log_2 m \rceil} - m$ values i.e $2^3 - 5 = 3$ values of r will represented by $\lfloor \log_2 m \rfloor$ (i.e 2) bits.

Next, $r + 2^{\lceil \log_2 m \rceil} - m = r + 2^3 - 5 = \underline{r+3}$

So, rest of the values will be represented by 3-bits representation of $r+3$.

Golumb code for $m=5$

$$q = \left\lfloor \frac{n}{m} \right\rfloor$$

n	q	r	Codeword
0	0	0	0 00
1	0	1	0 01
2	0	2	0 10
3	0	3	0 110
4	0	4	0 111
5	1	0	10 00
6	1	1	10 01
7	1	2	10 10
8	1	3	10 110
9	1	4	10 111
10	2	0	110 00
11	2	1	110 01
12	2	2	110 10
13	2	3	110 110
14	2	4	110 111
15	3	0	1110 00

Rice Codes

→ It can be viewed as an adaptive Golomb code.

→ In this code, a sequence of non-negative integers is divided into blocks of J integers a piece. Each block is then coded using one of several options, and the option resulting in the least no. of coded bits is selected.

→ The easiest way to understand the rice code is to study the implementation of CCSDS.

CSDS – Consultative Committee on Space Data Standards.

CCSDS Recommendation for Lossless Compression –

→ This algorithm consists of a preprocessor and a binary coder.

→ The preprocessor removes correlation from the input and generates a sequence of non-negative integers. This sequence has the property that smaller values are more probable than larger values.

→ The binary coder generates a bitstream to represent integer sequence.

→ The preprocessor functions as follows – Given a sequence $\{y_i\}$ for each y_i, we generate a prediction $\hat{y}_i$. as –

$$\hat{y}_i = y_{i-1}$$

→ Then, generate a sequence whose elements are difference between y_i and its predicted value $\hat{y}_i$.

$$d_i = y_i - \hat{y}_i$$

→ Let y_{max} and y_{min} be the largest and smallest values that the sequence $\{y_i\}$ takes on.

Define $T_i = \min\{ y_{max} - \hat{y}_i,\ \hat{y}_i - y_{min} \}$

→ The sequence $\{d_i\}$ can be converted into a sequence of non-negative integers $\{x_i\}$ using –

$$x_i = \begin{cases} 2d_i & 0 \le d_i \le T_i \\ 2|d_i| - 1 & -T_i \le d_i < 0 \\ T_i + |d_i| & \text{otherwise} \end{cases}$$

→ Further, the sequence $\{x_i\}$ is divided into segments. Segment is further divided into blocks

Assignment/Tutorial

(2) For an alphabet $A = \{a_1, a_2, a_3\}$ with probabilities $P(a_1) = 0.7$, $P(a_2) = 0.2$, $P(a_3) = 0.1$. Design a 3-bit Tunstall code.

(i)

Letter	Probability
a_1	0.7
a_2	0.2
a_3	0.1

(ii)

Letter	Probability
a_2	0.2
a_3	0.1
$a_1 a_1$	0.49
$a_1 a_2$	0.14
$a_1 a_3$	0.07

(III)

Letter	Codeword
a_2	000
a_3	001
$a_1 a_2$	010
$a_1 a_3$	011
$a_1 a_1 a_1$	100
$a_1 a_1 a_2$	101
$a_1 a_1 a_3$	110

(1)

Letter	Probability
A	0.6
B	0.3
C	0.1

(2) Remove highest probability item i.e A from list. Add all other two-letter strings beginning with A

Letter	Probability
B	0.3
C	0.1
AA	0.36
AB	0.18
AC	0.06

(3) Remove highest probability entry i.e AA from the list and add all other 3-letter strings starting with AA.

Letter	Probability
B	000
C	001
AB	010
AC	011
AAA	100
AAB	101
AAC	110

Now, Size of codebook = 7 $\leq 2^n (8)$

Adding one more iteration will increase the size of codebook to 10, which is exceeding the limit. Hence, this will be the final code.

Tunstall Codes L-17

In this code, all codewords are of equal length. However, each codeword represents a different no. of letter The main advantage of a Tunstall code is that errors in codewords do not propagate; unlike other variable length codewords.

The algorithm is as follows—

Suppose, we want n-bit Tunstall code for a source that generates iid (independent, identically distributed) letters from an alphabet of size N. The no. of codewords is 2^n. Start with N letters of source alphabet in our codebook. Remove the entry from codebook that has highest probability and add N strings obtained by concatenating this letter with every letter in the alphabet This increases the size of codebook from N to N+(N-1). The probabilities of new entries will be the product of the probabilities of the letters concatenated to form the new entry. Now, look through the N+(N-1) entries in the codebook. and find the entry that has highest probability. If this operation is performed K times, it must satisfy following expression—

$$N + K(N-1) \leq 2^n$$

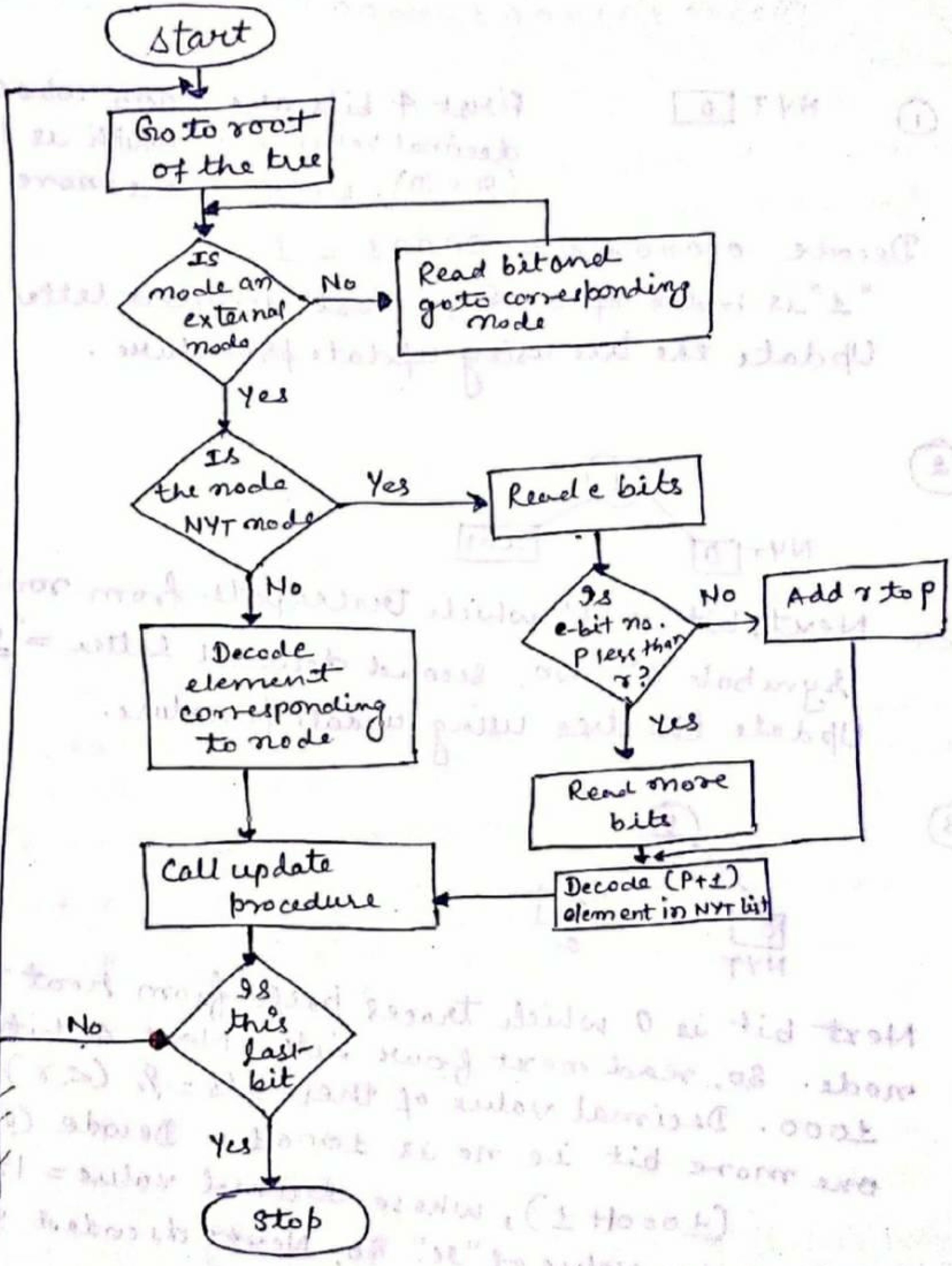
Ex:- Design a 3-bit Tunstall code for a memoryless source with following alphabet—

$$A = \{A, B, C\}$$

$$P(A) = 0.6, \quad P(B) = 0.3, \quad P(C) = 0.1$$

Decoding procedure:-

L - 15



0 0 0 0 0	1	0 1 0 0 0 1	0 0	0 0 0	1 1 0 0 1 1 0 1
					NYT

L-16

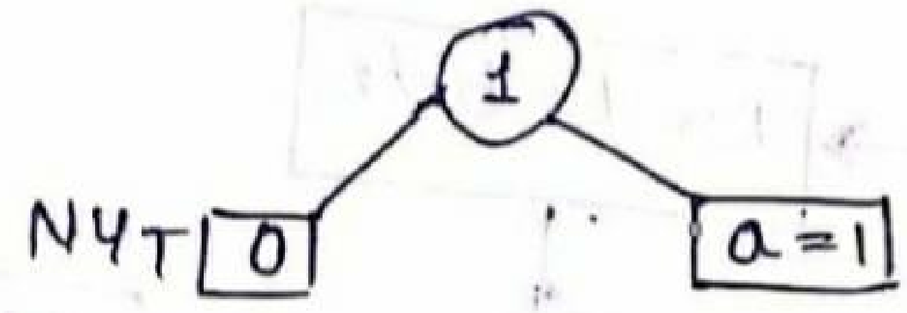
① NYT [0] First 4 bits are 0000 whose decimal value = 0 which is less than ($\gamma = 10$). So, read one more bit.

Decode 00000 + 1 = 00001 = 1.

"1" is index of a. So, First decoded letter = a

Update the tree using update procedure.

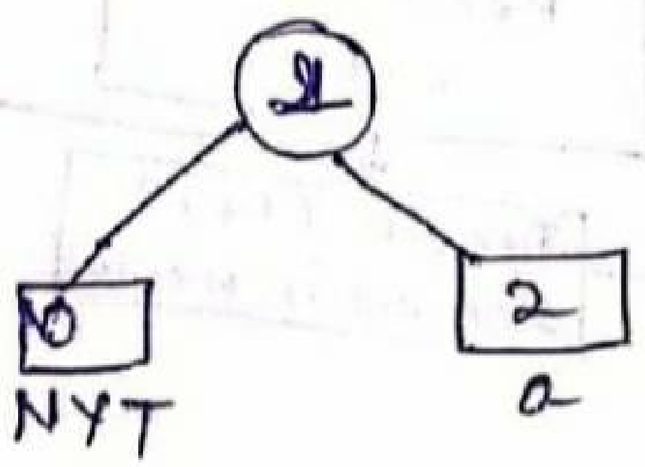
②



Next, bit is '1' which traces path from root to Symbol "a". So, second decoded letter = a

Update the tree using update procedure.

③



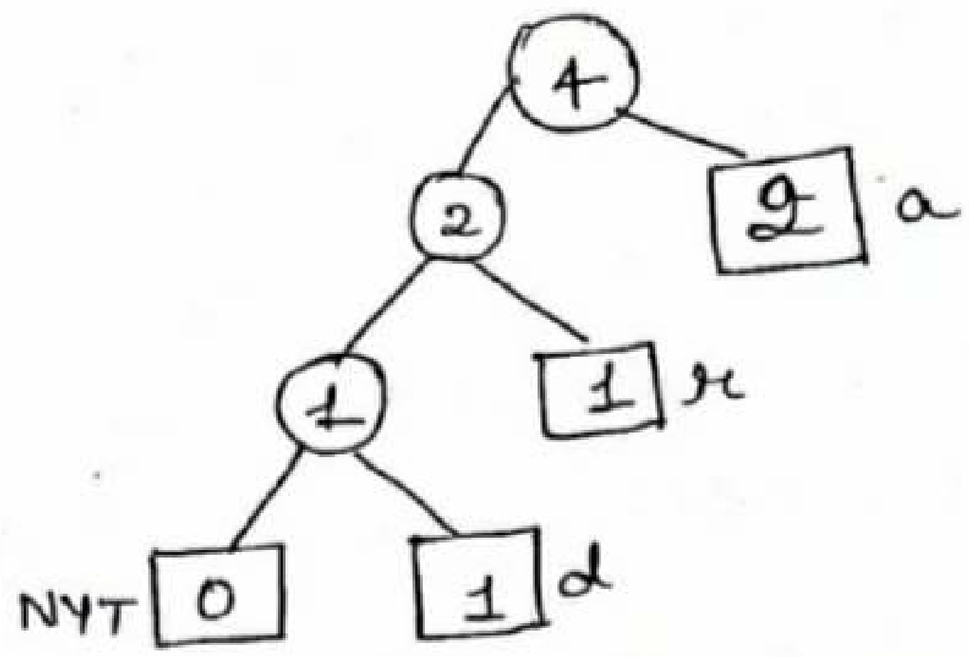
Next bit is 0 which traces path from root node. So, read next four bits. Next 4 bits 1000. Decimal value of these bits = 8 ($< \gamma$) one more bit i.e no is 10001. Decode (p (10001+1), whose decimal value = 1

"18" is index value of "r". So, Next decoded update tree using update procedure.

* Next two bits 00. traces path from root to NYT node. So, decoding next four bits 0001 gives decimal no. 1 which is less than r (10).

So, ~~decode~~ read one more bit (i.e 00011).
After decoding (P+1) element, we get 4, which is index value of "d". Therefore, decoded element is "d". Update tree using update procedure.

(5)

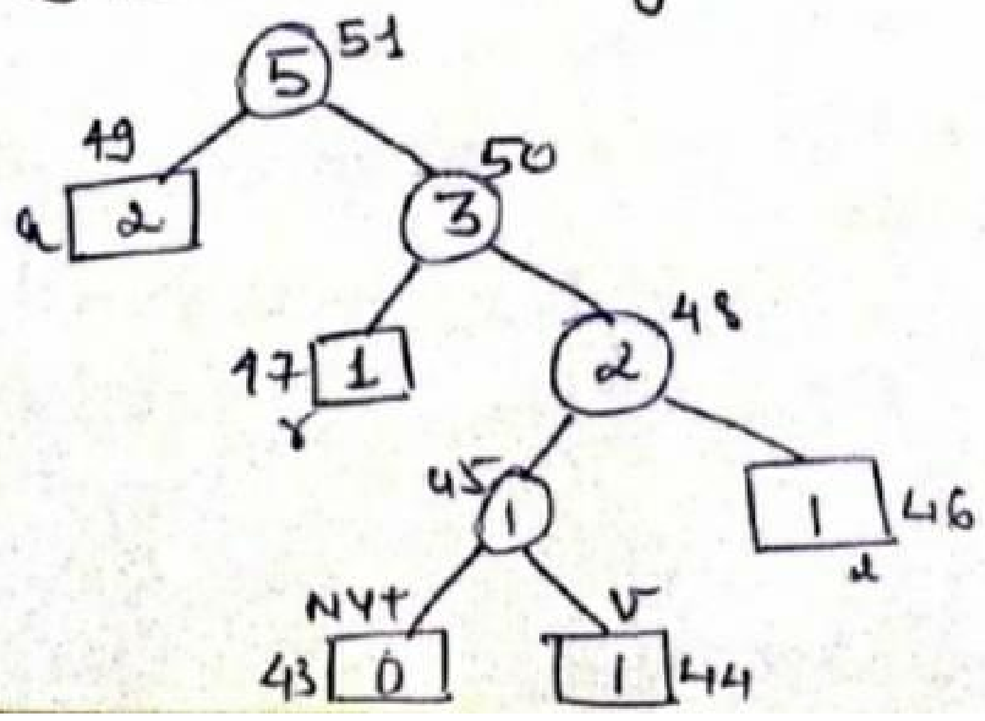


Next three bits 000 corresponds to NYT node. Read next four bits, which are 1011. Decimal value of this = 11 which is > 10 (r).

So, Add r to p. ⇒ 10 + 11 = 21 ⇒ (P=21)
Decode (P+1) element. ⇒ 22nd element of alphabet = v.

So, Next decoded element = v

Decoded String = aadrv



(6) '0' traces path from root to node "a". So, next decoded letter is "0".

Ex:- Encode the message [a a r dv ar k], where no. of coding in alphabet = 26

Pick e and r such that $m = 2^e + r$

$\Rightarrow 26 = 2^e + r$

$\Rightarrow e = 4, r = 10$, where $0 \leq r < 2^e$

Step 1 – First symbol is a. For a, K = 1. So, a will be encoded as (e+1) bit representation of K-1. So, a will be encoded as 00000. Huffman tree will be updated. (and $1 \leq K \leq 1 \leq 1 \leq 2$

Step-2 Again, next symbol is a. Simply, we traverse the tree from root to external node corresponding to a. So, 'a' will be encoded as 1.

Step-3 "r" is being transmitted for the first time. So, we send the code for the NYT node followed by the code for r. r is 18th letter of the alphabet. So, r will be encoded as 5-bit representation of 17. Therefore, r will be encoded as – 0 1 0 0 0 1.

Step-4 "d" is being transmitted for the first time.

Now, NYT code = 00

code for "d" = 00011

So, d is encoded as – 00 00011

Step-5 → Next, v is 22nd symbol in alphabet. But its index value is greater than 20. So, it will be encoded as 5-bit binary representation of K-r-1, which is 22-10-1 = 11

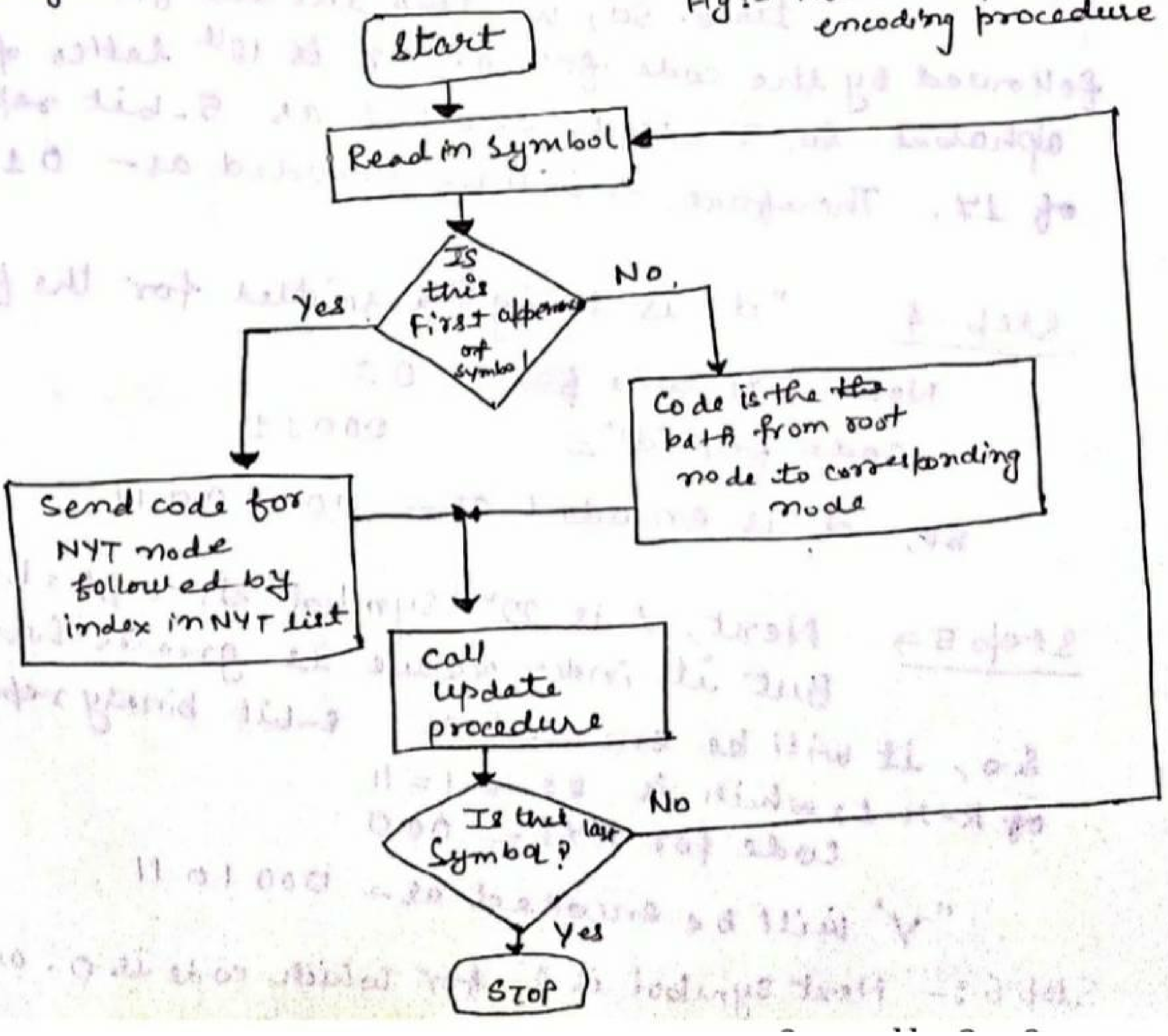
code for NYT = 000

"v" will be encoded as – 000 10 11

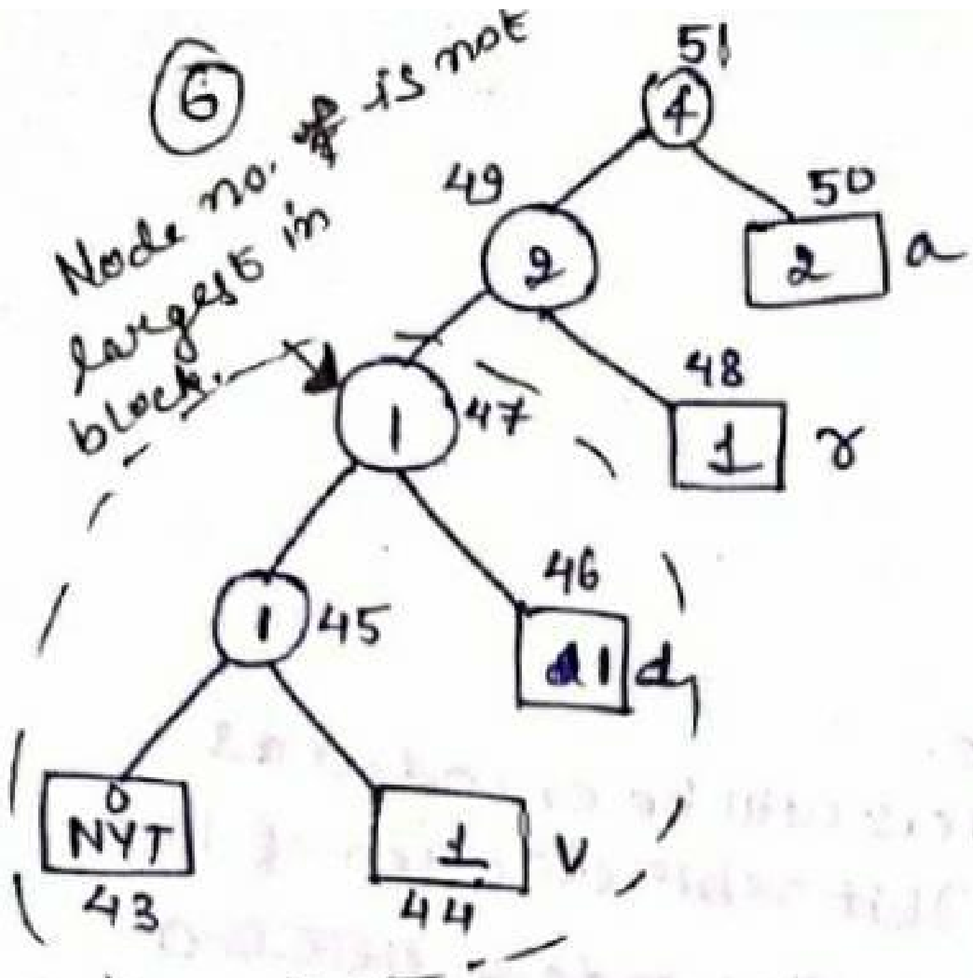
Encoding Procedure — L-14

Initially, the tree at both the encoder and decoder consists of a single node, the NYT node. Therefore, the codeword for the very first symbol is a fixed code. After very first symbol, whenever we want to encode a symbol that is being encountered for the first time, we send the code for the NYT node, followed by fixed code for the symbol. The code for the NYT node is obtained by traversing the Huffman tree from the root to the NYT node. If the node to be encoded has a corresponding node in the tree, then code for the symbol is generated by traversing the tree from the root to the external node to the symbol.

Fig:- Flow chart of encoding procedure

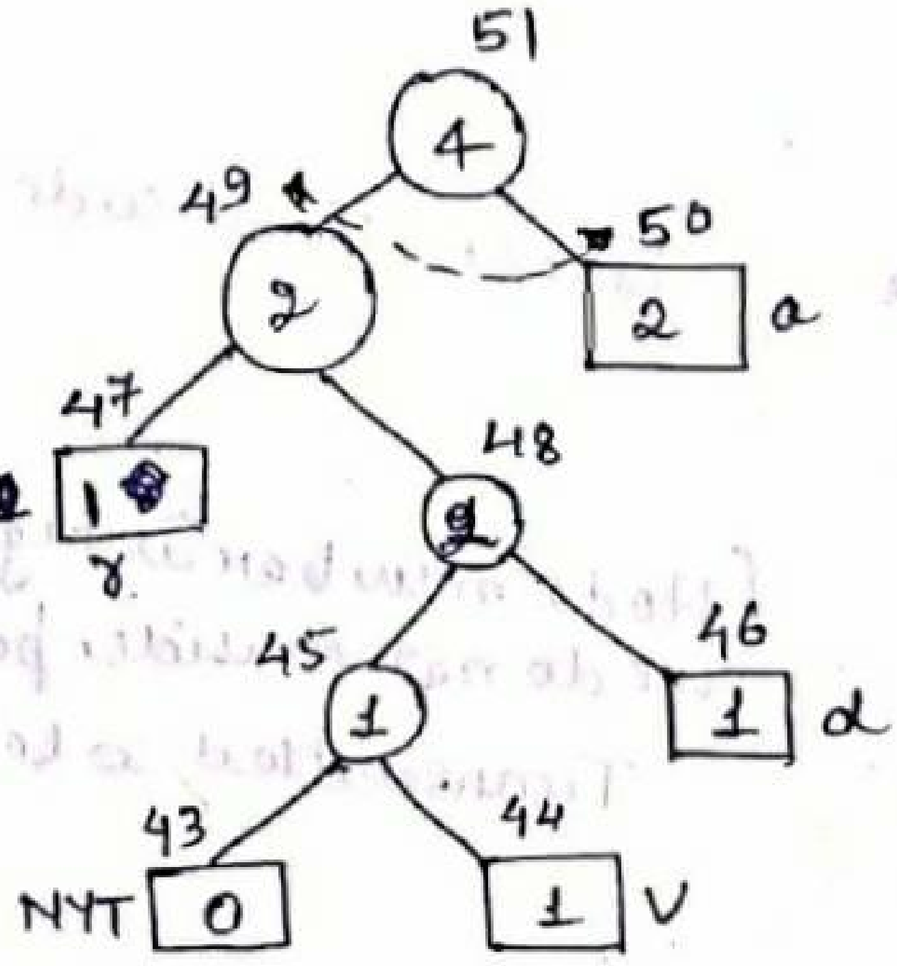


(6)



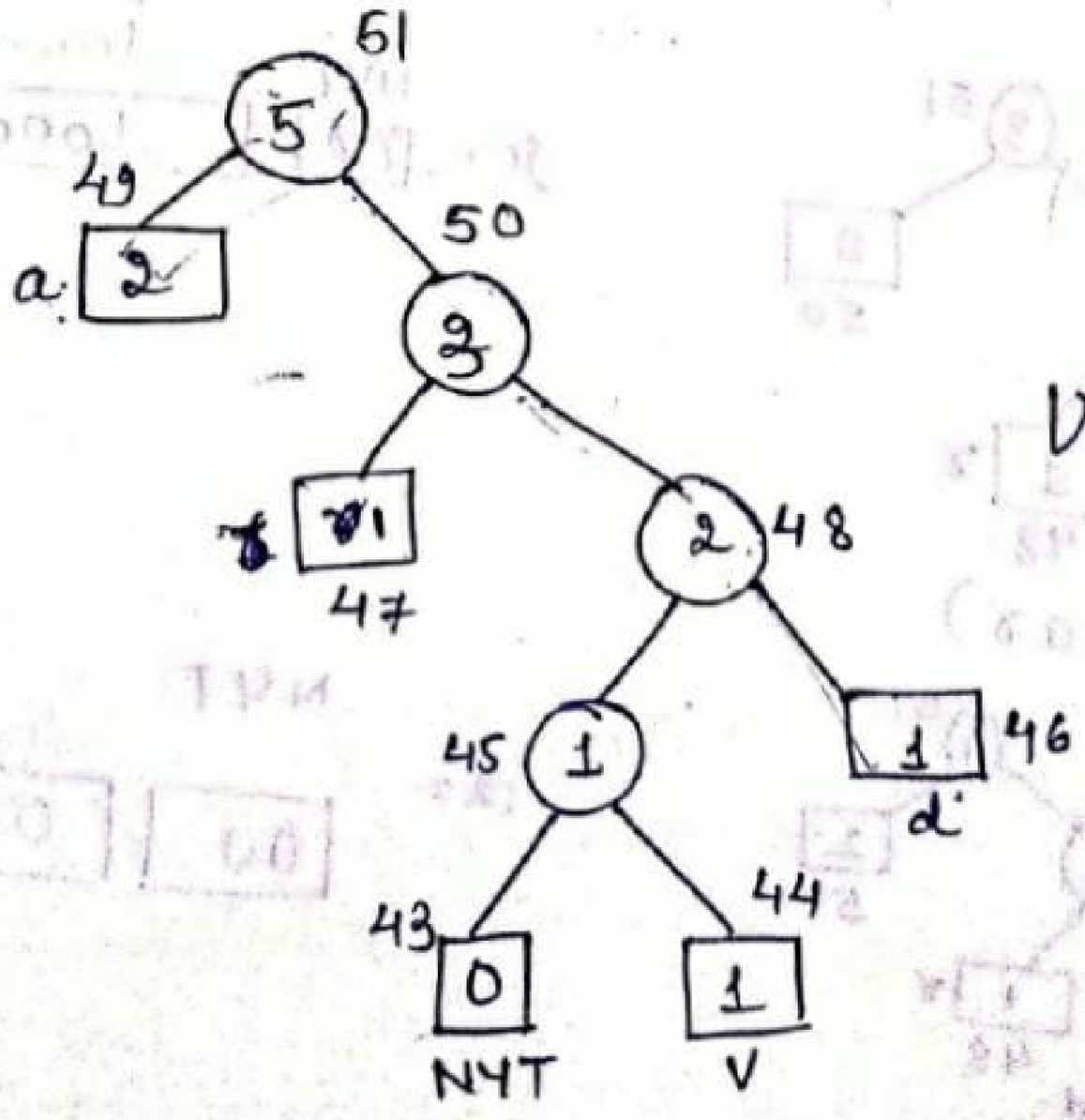
Node no. 47 and 48 will be exchanged.

(7)



Again, node number 49 and 50 are exchanged.

(8)



NYT

V = 000 1011

e and r will be,

$$n = 2^e + r$$

$$\Rightarrow 26 = 2^e + r$$

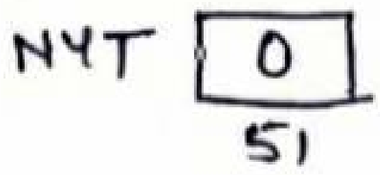
$$\Rightarrow e = 4$$

$$r = \pm 0$$

L - 14

Procedure -

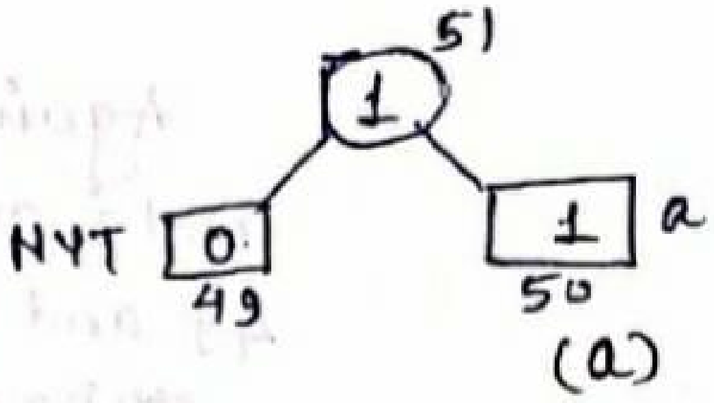
(1) Begin with only one NYT node.



Letters will be encoded as (e+1) bit representation of K-1.

~~Transmitted code~~ = ~~0000~~ 0

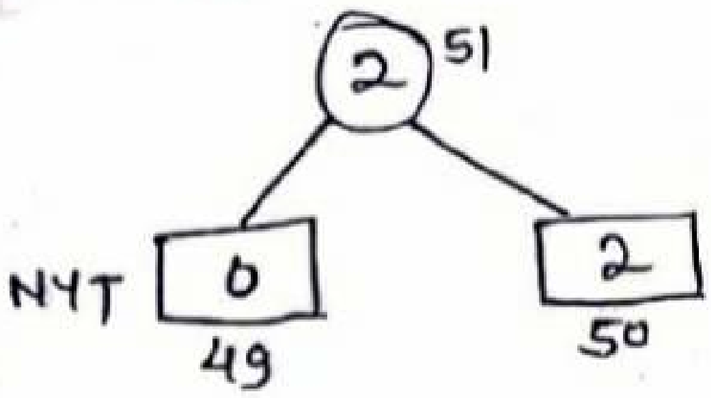
(2)



(a)

Transmitted code = 00000

(3)

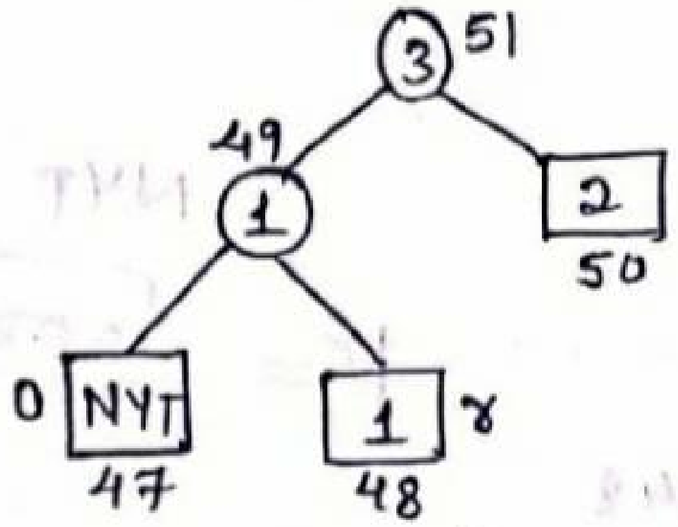


(aa)

(Node number is highest, if we do not conside parent)

Transmitted code = 1

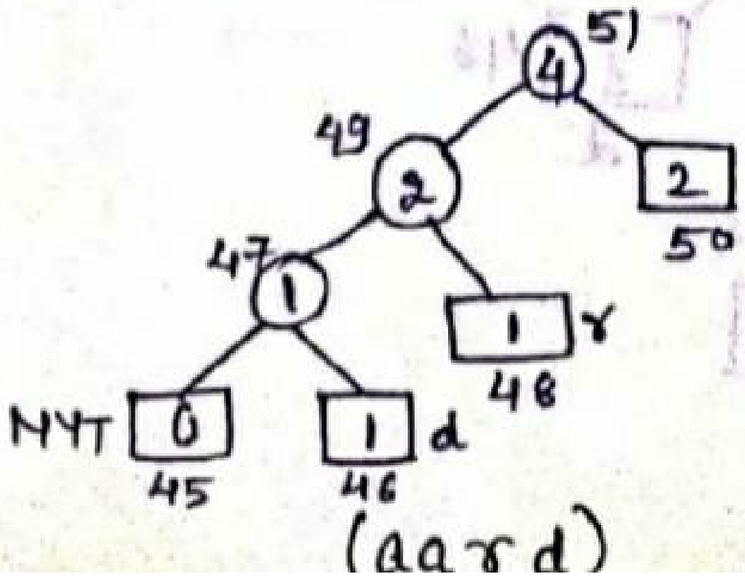
(4)



(aar)

r = NYT (00) | 10001 | decimal → 17

(5)



(aard)

d2 = NYT (00) | 00011 | decimal (3)

Ex⇒ Encode the message [a a r d v a rk], where our alphabet consists of the 26 lowercase letters of the English alphabet.

Here, $n = 26$

Total no. of nodes = $2n-1 = 2\times26-1 = 51$

Update Procedure L-13

The function of update procedure is to preserve (Reelb) = the sibling property. The largest node number is given to the root of the tree and smallest no. is assigned to NYT node. The set of nodes with same weight makes up a block.

After a symbol has been encoded or decoded the external node corresponding to the symbol is examined to see if it has the largest node number in its block. If external node does not have the largest node number, it is exchanged with the node ~~number~~ that has largest node number in the block. Then weight of external node is incremented. Then parent node of that external node is examined to see if it has the largest node number in block. If not, again it is exchanged until it has the largest no. in the block. This process is repeated until root of the tree is reached.

If the symbol to be encoded or decoded has occured for the first time, a new external node is assigned to the symbol and a new NYT is appended to the tree. And weight of these two nodes is ~~is~~ incremented by 1. Updation is repeated ~~until~~ until root of the tree is reached.

as compared with Huffman encoding. This makes it easy to implement "Buffer" which has to be of finite size. Output of source coder is given to buffer. The purpose of buffer is to smooth out the variations in the bit generation rate.

Redundancy of the code— It is a measure of efficiency of the code. It is the difference between the entropy & the average length.

i.e Redundancy = Entropy of code − Avg. length of code

In above example,

Entropy can be calculated as -

$$H = -\sum_{i=1}^{m} p_i \log p_i$$

$$= -[\,0.4 \log_2(0.4) + 0.2 \log_2(0.2) + 0.2 \log_2(0.2) + 0.1 \log_2(0.1) + 0.1 \log_2(0.1)\,]$$

(6) Design Huffman code and Calculate Entropy for given alphabet —

Sol:-

$p(a_1) =$ 0.95
$p(a_2) =$ 0.02
$p(a_3) =$ 0.03

Letter	probability
a_1	0.95
a_2	0.02
a_3	0.03

		code	
a_1	0.95	.	→ 0
a_2'	0.05	α_1	→ 1

$a_2 \rightarrow \alpha_1 0 \rightarrow 10$
$a_3 \rightarrow \alpha_1 1 \rightarrow 11$

$a_1 \rightarrow 0$
$a_2 \rightarrow 10$
$a_3 \rightarrow 11$

Average length = $0.95 \times 1 + 0.02 \times 2 + 0.03 \times 2$
= $0.95 + 0.04 + 0.06$
= 1.05 bits/symbol

Entropy = $-0.95 \times \log_2(0.95) - 0.02 \log_2(0.02) - 0.03 \log_2(0.03)$
= 0.335 bits/symbol.

Redundancy = $0.335 - 1.05 = -0.71$ bits/sym

$-0.95 \times (-0.074) - 0.02 \times (-5.64) - 0.03 \times (-5.058)$
$0.0703 + 0.1128 + 0.1517$
≈ 0.34 bits/symbol.

L-12 Adaptive Huffman Coding →

→ In this algorithm, two parameters are added to the binary tree - weight of each leaf, and node number.

→ Weight of each external node is no. of times the symbol corresponding to the leaf has been encountered. The weight of each internal node is the sum of its offspring.

→ (i) The node number Y_i is a unique number assign to each internal and external node. If we have an alphabet of size n, then $2n-1$ internal and external nodes can be numbered as $Y_1, Y_2, \ldots, Y_{2n-1}$ such that if x_j is the weight of node Y_j, we have

$$x_1 \leq x_2 \leq \cdots \leq x_{2n-1}$$

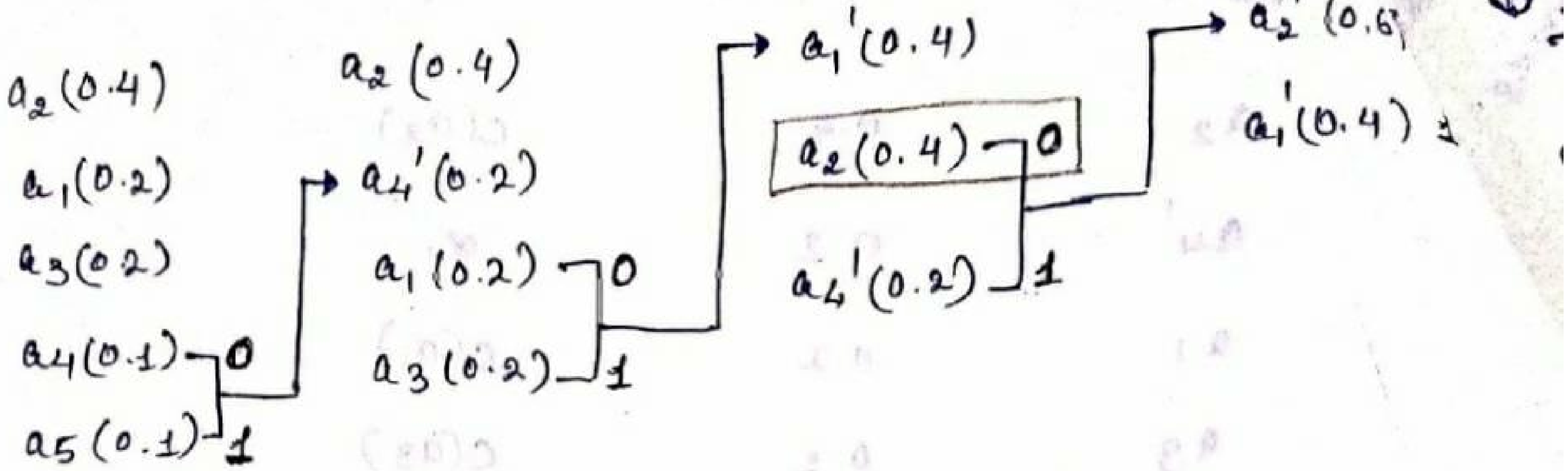
(II) Furthermore, the nodes Y_{2j-1} and Y_{2j} are offspring of same parent node, for $1 \leq j < n$, and node number for the parent node is greater than Y_{2j-1} and Y_{2j}.

These two properties are called the sibling property

If source has an alphabet $(a_1, a_2, \ldots, a_m)$ of size m, then pick e and r such that $m = 2^e + r$ and $0 \leq r < 2^e$.

The letter a_k is encoded as $(e+1)$-bit binary representation of $k-1$, if $1 \leq k \leq 2r$; else a_k is encoded as e-bit binary representation of ~~k-1~~, ~~if~~ ~~$1 \leq k \leq 2r$~~; ~~else~~ $k-r-1$

Miniman Variance Huffman Encoding procedure



$a_1 \rightarrow 10$

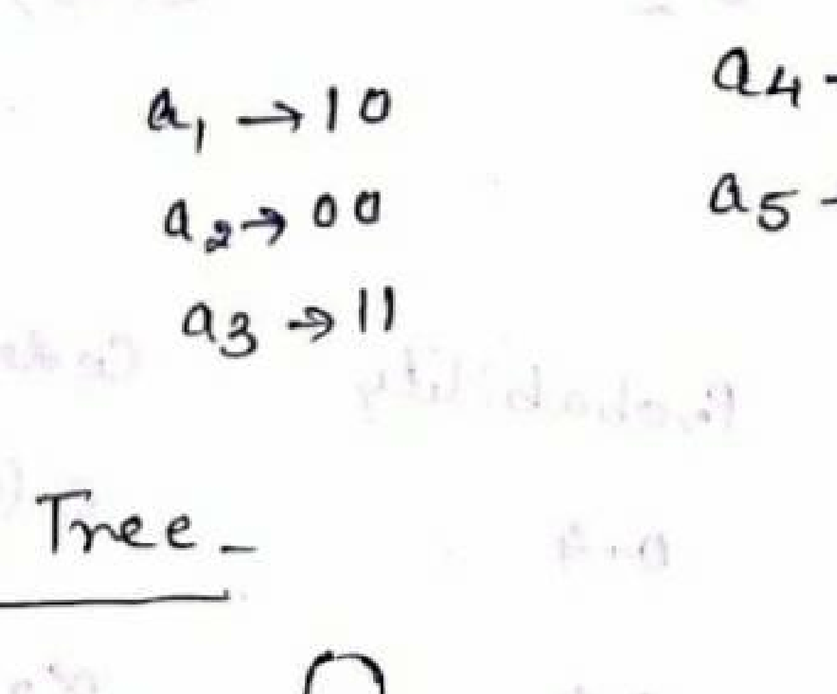
$a_2 \rightarrow 00$

$a_3 \rightarrow 11$

$a_4 \rightarrow 010$

$a_5 \rightarrow 011$

Huffman Tree -



Length of the code -

$$l = 0.2 \times 2 + 0.4 \times 2 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3$$

$$= 0.4 + 0.8 + 0.4 + 0.3 + 0.3$$

$$= 2.2 \text{ bits/symbol.}$$

→ Although the redundancy is same in both the procedures, but it is preferred to use Miniman Variance Huffman encoding. This is because Variation in codeword length is less in miniman variance Huffman encoding

Step-2

letter	Probability	Codeword
a_2	0.4	$c(a_2)$
a_4'	0.2	α_1
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$

$c(a_4) = \alpha_1 * 0$
$c(a_5) = \alpha_1 * 1$

$c(a_1) = \alpha_2 * 0$
$c(a_3) = \alpha_2 * 1$

Step-3

Letter	Probability	Codeword
~~a_2~~	~~0.4~~	~~$c(a_2)$~~
a_1'	0.4	α_2
a_2	0.4	$c(a_2)$
a_4'	0.2	α_1

$c(a_2) = \alpha_2 * 0$
$c(a_4') = \alpha_3 * 1$
$\alpha_1 = \alpha_3 * 1$

Huffman code Ans.

$a_1 = 001$
$a_2 = 01$
$a_3 = 000$
$a_4 = 1$

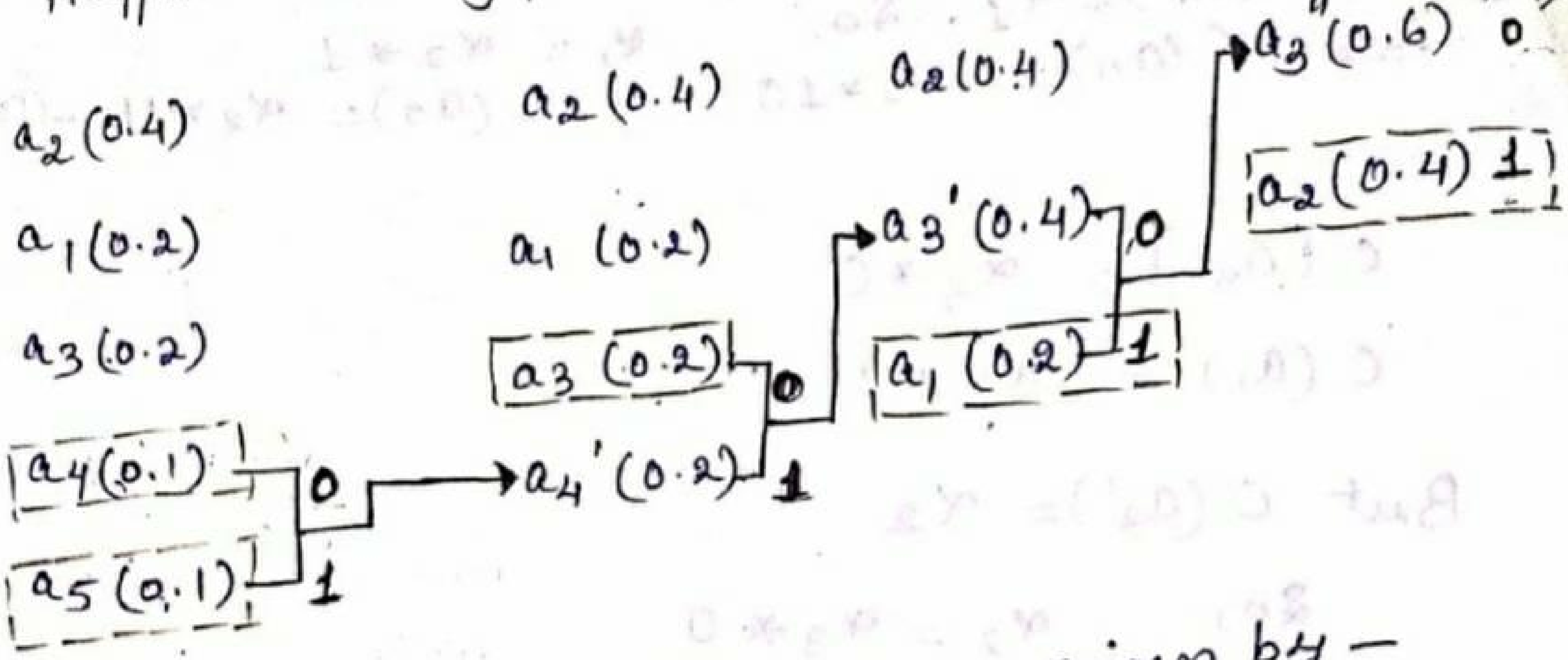
Step-4

Letter	Probability	Codeword
a_2'	0.6	α_3 — 0
a_1'	0.4	α_2 — 1

Minimam Variance Huffman Code →

Letter	Probability	Codeword
a_2	0.4	00
a_1	0.2	10
a_3	0.2	11
a_4	0.1	010
a_5	0.1	011

Huffman Encoding procedure –



Average length for the code is given by –

$$l = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4$$

$$= 0.4 + 0.4 + 0.6 + 0.4 + 0.4$$

$$= 2.2 \text{ bits/Symbol}$$

2. Calculate Huffman code for alphabet $A = \{a_1, a_2, a_3, a_4\}$ where $P(a_1) = 0.1$, $P(a_2) = 0.3$, $P(a_3) = 0.25$ & $P(a_4) = 0.35$

Minimam Variance Huffman Code :–

In this variation of Huffman code, sorting is done in a slightly different manner.

Step-1

Letter	Probability	Codeword
a_2	0.4	$c(a_2)$
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$
a_4	0.1	$c(a_4)$
a_5	0.1	$c(a_5)$

$$c(a_3') = \alpha_3 * 0$$

$$c(a_1) = \alpha_3 * 1$$

But $c(a_3') = \alpha_2$

So, $\alpha_2 = \alpha_3 * 0$

Put this value in eqn (1).

$$c(a_3) = \alpha_2 * 0 = \alpha_3 * 00$$
$$c(a_4) = \alpha_2 * 10 = \alpha_3 * 010$$
$$c(a_5) = \alpha_2 * 11 = \alpha_3 * 011$$

— (2) —

Step-4 Again construct new alphabet with two symbols, where $P(a_3'') = P(a_3') + P(a_1)$

Letter	Probability	Codeword
a_3''	0.6	α_3
a_2	0.4	$c(a_2)$

Codewords can be assigned as -

$$c(a_3'') = \alpha_3 = 0$$

$$c(a_2) = 1$$

Put $\alpha_3 = 0$ in eqn (2).

$$c(a_4) = 010$$
$$c(a_5) = 011$$
$$c(a_3) = 000$$
$$c(a_1) = 01$$

$$c(a_4) = \alpha_1 * 0$$
$$c(a_5) = \alpha_1 * 1$$

where α_1 is a string and * denotes concatenation. But

Step-2 Define new alphabet A' with four letters a_1, a_2, a_3, a_4'. a_4' is composed of a_4 & a_5.

And $P(a_4') = P(a_4) + P(a_5)$. Again sort new alphabet A' is descending order.

Letter	Probability	Codeword
a_2	0.4	$c(a_2)$
a_1	0.2	$c(a_1)$
a_3	0.2	$c(a_3)$
a_4'	0.2	α_1

Step-3 Same process defined in step ② is repeated with a_3 and a_4'.

And, $P(a_3') = P(a_3) + a(a_4')$

Letter	Probability	Codeword
a_2	0.4	$c(a_2)$
a_3'	0.4	α_2
a_1	0.2	$c(a_1)$

In this case, two least probable letters are a_3' & a_1. Therefore, we can assign codewords as —

L-11 Huffman Coding [UNIT-2]

Huffman Coding Algorithm-

It is based on following two observations-

(1) In an optimum code, symbols that occur more frequently will have shorter codewords than symbols that occur less frequently.

(2) In an optimum code, the two symbols that occur least frequently will have the same length. This is done by having codewords corresponding to two lowest probabilities differ only in last bit.

ex→ Design Huffman code for a source that puts out letters from alphabet $A = \{a_1, a_2, a_3, a_4, a_5\}$ with $P(a_1) = P(a_3) = 0.2$

$P(a_2) = 0.4$ and $P(a_4) = P(a_5) = 0.1$

Soln- Step-1 → Arrange letters in descending order of probabilities.

Letter	Probability	Codeword