Module - II

Syllabus

Basic Compression Technique - Run length encoding, RLE Text compression, statistical Methods - Prefix codes, Binary Huffman coding, Non-binary Huffman Algorithms, Arithmetic coding.

Run Length Encoding (RLE)

Run-length Encoding (RLE) is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the orginal run.

If a data item 'd' occurs 'n' consecutive times in the input stream,

then replace the n occurences with with the single pair nd. The 'n' Consecutive occurances of a data item are called a runlength of 'n'.

RLE Text Compression

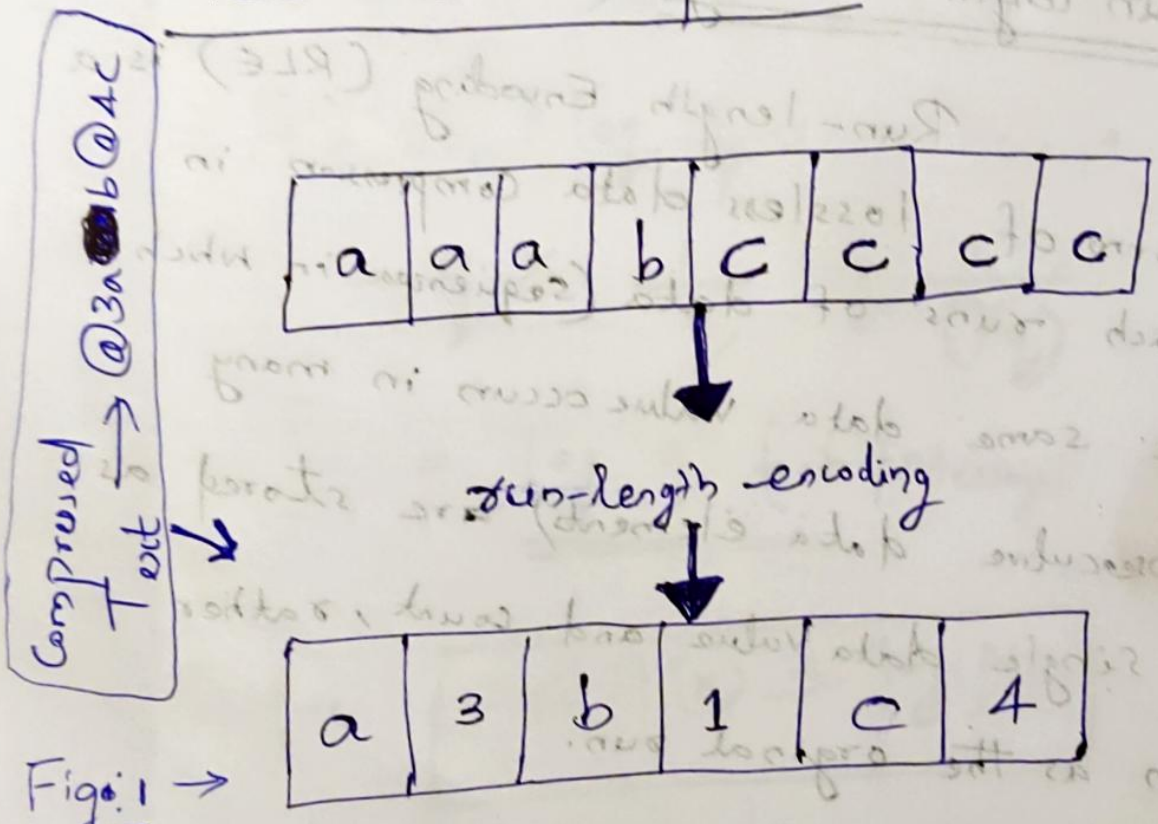


Fig: 1 →

This type of compression works best with simple images and

animations that have a lot of redundant pixels.

The following figure shows the flowchart for a simple run-length text compressor.

After reading the first character, the repeat-count is 1 and the character is saved. Subsequent characters are compared with the one already saved, and if they are identical to it, the repeat-count is incremented. When a different character is read, the operation depends on the value of the repeat-count. If it is small, the saved character is ~~saved~~ written on the

Compressed file, and the newly-read character is saved. Otherwise, an '@' is written followed by the repeat-count and the saved character.

Decompression is also shown in the following flow chart. When an '@' is read, the repetition count 'n' and the actual character are immediately read, and the character is written 'n' times on the output stream.

The ~~decoding~~ compressed data corresponding to Fig:1 is @3ab@4c.

Flowchart of RLE Compression

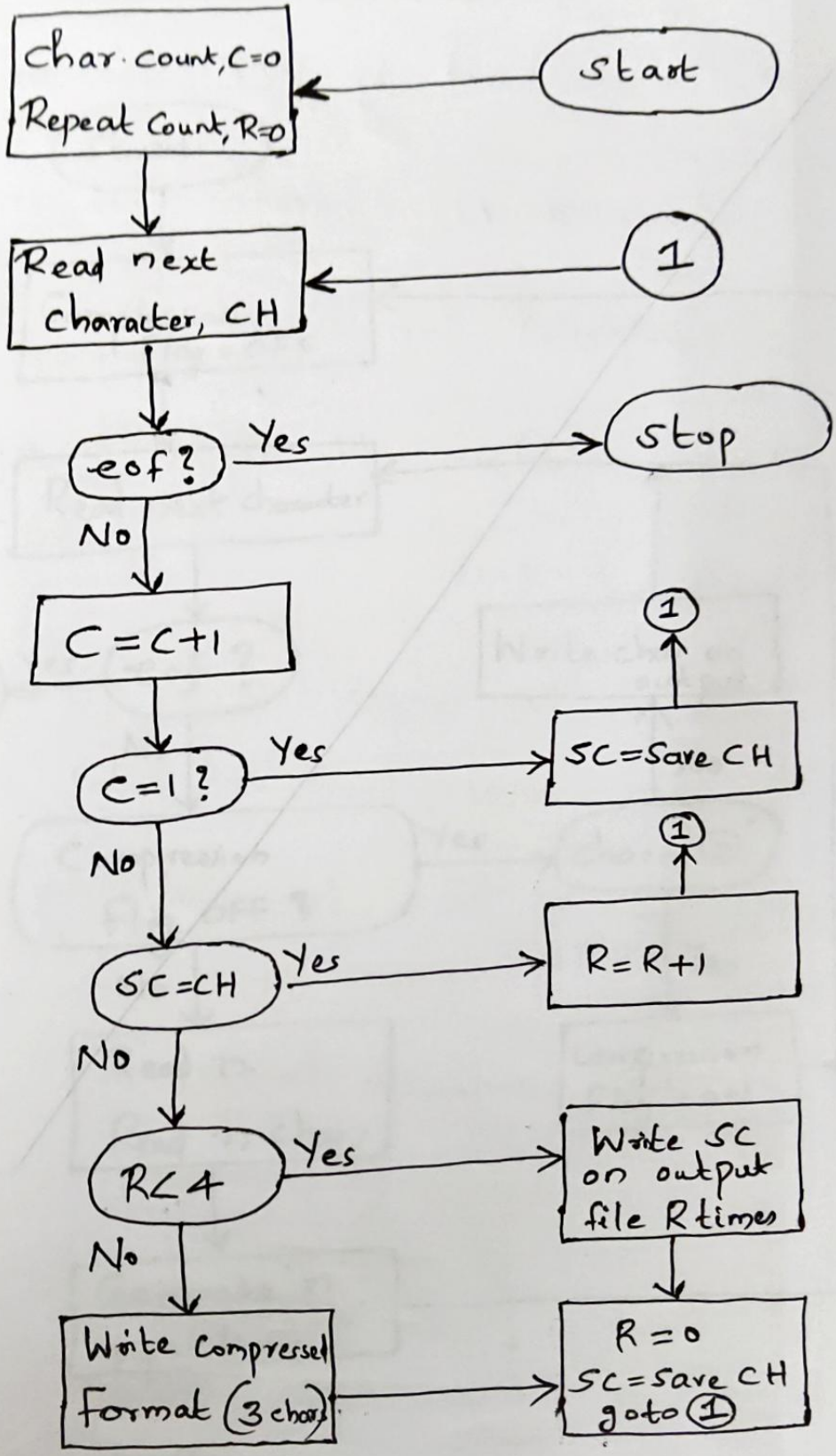


Fig:- RLE – Compression

Flowchart of RLE Decompression

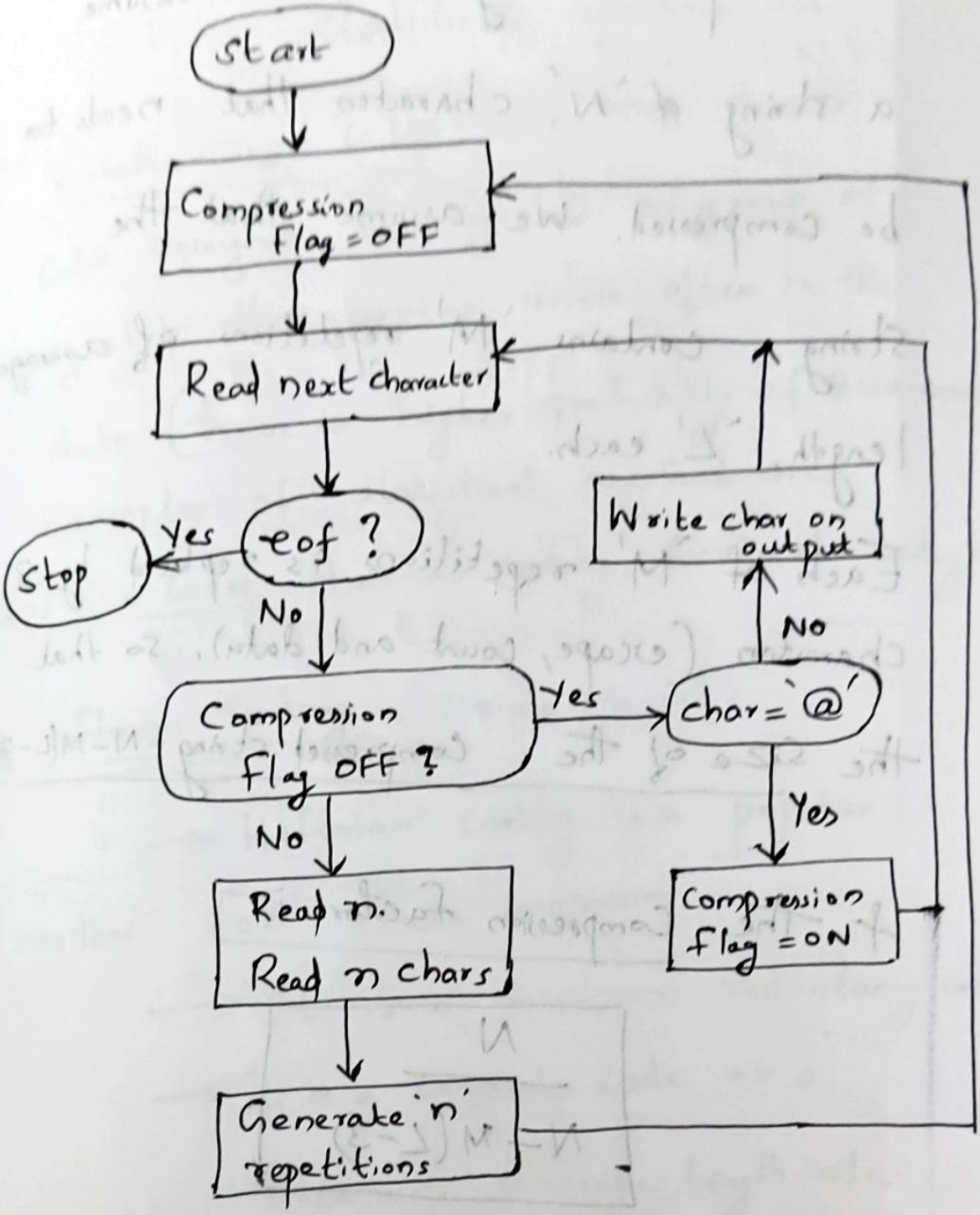


Fig- RLE Decompression.

To get an idea of compression ratios produced by RLE, We assume a string of 'N' characters that needs to be compressed. We assume that the string contains 'M' repetitions of average length, 'L' each.

Each of 'M' repetitions is replaced by 3 characters (escape, count and data). So that the size of the compressed string = $N - M(L-3)$

∴ the Compression factor is

$$\frac{N}{N - M(L-3)}$$

Statistical Methods for Compression

Statistical methods use Variable-size codes, with the shorter codes assigned to symbols or group of symbols that appear more often in the data (have a higher probability of occurence). Examples of statistical methods are Prefix Codes, Huffman coding, Arithmatic coding etc.

(already studied in Module I)

Huffman Coding (Simple Method)

→ Huffman coding is a popular method for data compression.

→ It is a minimum redundancy code.

→ It is a compact code or a minimum average length code.

Binary.

Algorithm for Huffman coding

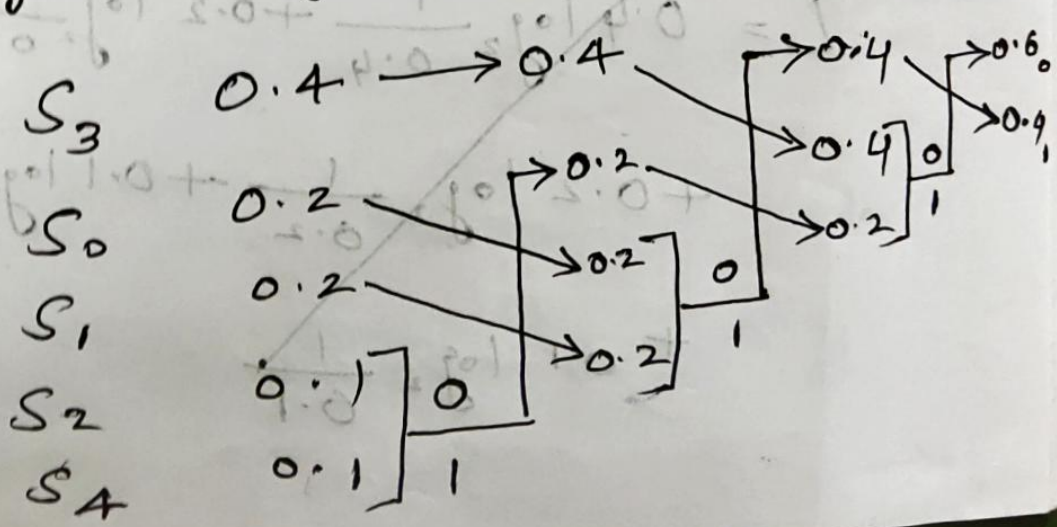
Step 1:- List the source symbols in the decreasing order of probabilities. The two source symbols having the lowest probabilities are assigned 0 and 1.

Step 2:- These two source symbols are regarded as being combined to a new source symbol with probability equal to the sum of two orginal probabilities. Probability of this new symbol is placed in the list in accordance with this value.

step 3:- This procedure is repeated until two sources will be remained for which 0 & 1 are assigned. The code for each orginal symbol is found by working Backwards and tracing sequences of 0's and 1's.

Q The source symbol, $S = \{s_0, s_1, s_2, s_3, s_4\}$ has probabilities $\{0.2, 0.2, 0.1, 0.4, 0.1\}$. Obtain Huffman coding. Calculate the average size of this code.

Ans:- Average according to decreasing probabilities.



Symbol	Code	Original code by bit reversal
S_3	00	00
S_0	01	10
S_1	11	11
S_2	010	010
S_4	110	011

Average size of the code,

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

$$L = 0.4 \log_2 \frac{1}{0.4} + 0.2 \log_2 \frac{1}{0.2} + 0.2 \log_2 \frac{1}{0.2} + 0.1 \log_2 \frac{1}{0.1} + 0.1 \log_2 \frac{1}{0.1}$$

$$\bar{L} = 0.4 \times 2 + 0.2 \times 2 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3$$

$$= 2.2 \text{ bit/symbol}$$

HW The message 's' consists of 5 individual message $\{s_1, s_2, s_3, s_4, s_5\}$ & their probabilities are $\{0.55, 0.15, 0.15, 0.1, 0.05\}$ obtain Huffman code and its efficiency.

Hint:- Efficiency $= \eta = \frac{H(s)}{\bar{L}}$ → $H(s)$ → Entropy, $\bar{L}$ → Average Length of code

Redundancy $= 1 - \eta$

Note:- Length of Huffman codes:- $H(s) \leq \bar{L} \leq H(s)+1$

$H(s)$ → Entropy
$\bar{L}$ → Average code word length of Huffman code.

★ Huffman Tree

→ The Huffman code can be also ~~created~~ made by creating Huffman trees. →

Q. Given the 8 symbols A, B, C, D, E, F, G, H with probabilities, $\frac{1}{30}, \frac{1}{30}, \frac{1}{30}, \frac{2}{30}, \frac{3}{30}, \frac{5}{30}, \frac{5}{30}, \frac{12}{30}$. respectively. Draw Huffman tree and calculate the average size of code tree in bits per symbol.

Ans:- Let, $\frac{1}{30} \rightarrow A$

$\frac{1}{30} \rightarrow B$

$\frac{1}{30} \rightarrow C$

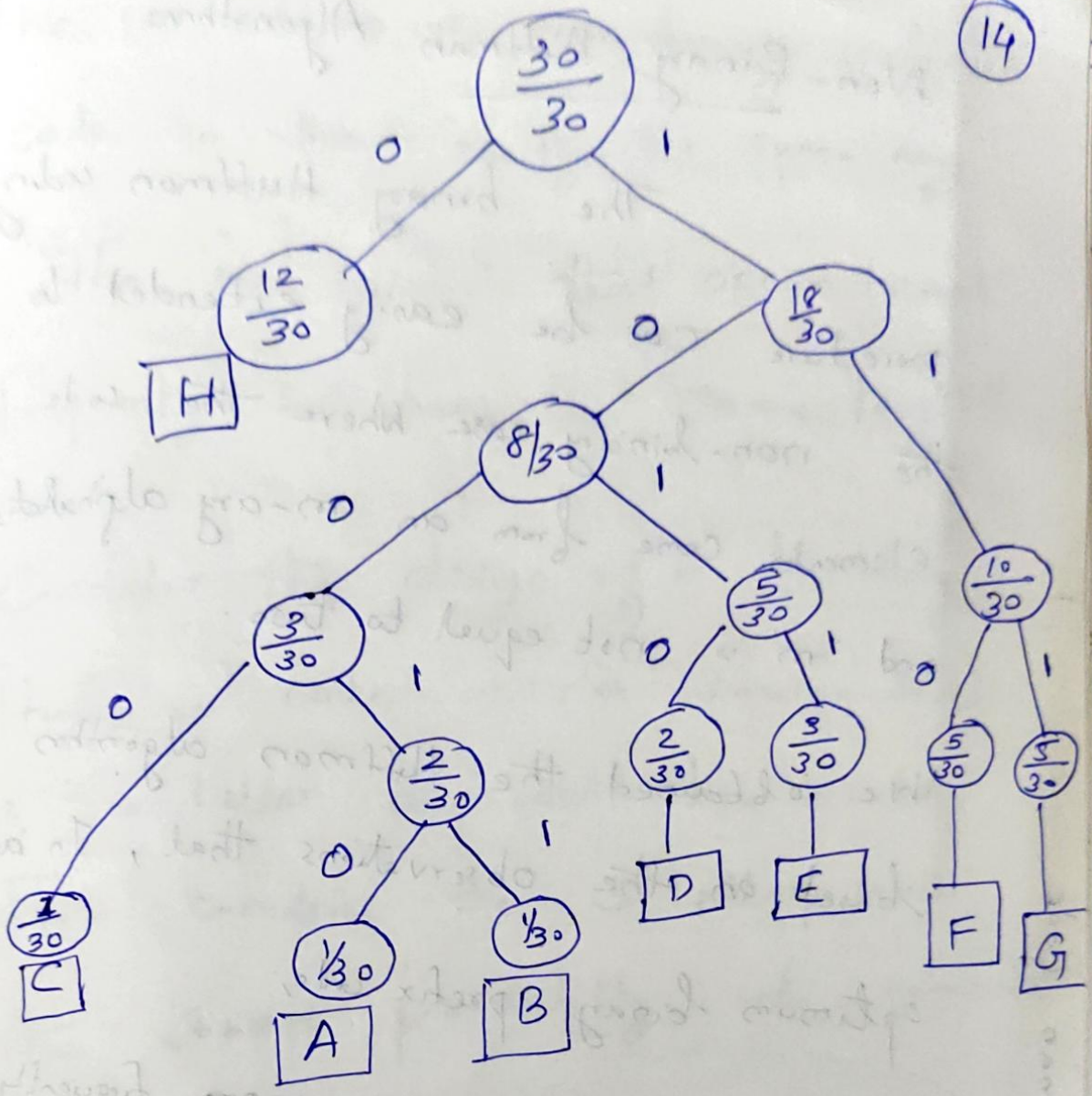
$\frac{2}{30} \rightarrow D$

$\frac{3}{30} \rightarrow E$

$\frac{5}{30} \rightarrow F$

$\frac{5}{30} \rightarrow G$.

$\frac{12}{30} \rightarrow H$



$\frac{1}{30} \rightarrow$

Code size $= \frac{1}{30} \times 5 + \frac{1}{30} \times 5 + \frac{1}{30} \times 4$

$+ \frac{2}{30} \times 4 + \frac{3}{30} \times 4 + \frac{5}{30} \times 3 + \frac{5}{30} \times 3$

$+ \frac{12}{30} \times 1 = \frac{76}{30}$ bits/symbol

Non-Binary Huffman Algorithms

The binary Huffman coding procedure can be easily extended to the non-binary case where the code elements come from an m-ary alphabet, and 'm' is not equal to two.

We obtained the Huffman algorithm based on the observations that, In an optimum binary prefix code,

observations for Huffman code

(i) symbols that occur more frequently will have shorter codewords than symbols that occur less frequently.

(ii) the two symbols that occur least frequently will have the same length.

We can obtain a Non binary Huffman code in almost exactly the same way.

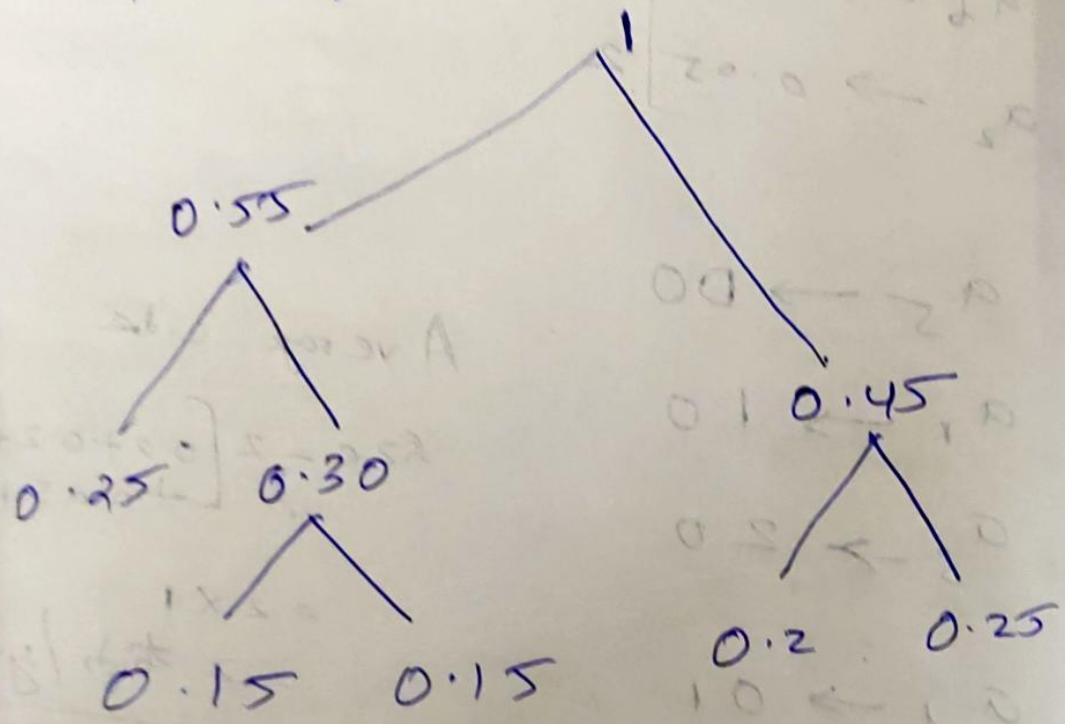
"The 'm' symbols that occur least frequently will have the same length".

Consider the design of a ternary Huffman code for a source with a 6-letter alphabet. We would first combine the 3 letters with the lowest probability into a composite letter. This would give us a reduced alphabet with four letters.

Huffman Binary ~~Ternary~~ Code Tree

Q. Generate a Huffman code tree for five symbols with probabilities 0.15, 0.15, 0.2, 0.25, and 0.25. Find Average code size.

Ans:-

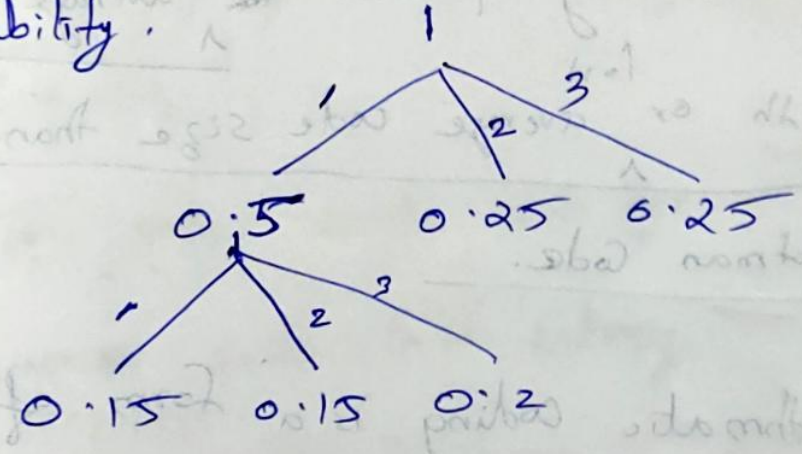


Average code size,

$2 \times 0.25 + 3 \times 0.15 + 3 \times 0.15 + 2 \times 0.2 + 2 \times 0.25 = 2.3$ bits/symbol

Huffman Ternary Tree (for the same problem)

If the tree is constructed by selecting three symbols with smallest probabilities and merging them into one parent symbol with the combined probability.



Here, Average code size

$= 2 \times 0.15 + 2 \times 0.15 + 2 \times 0.2$

$+ 1 \times 0.25 + 1 \times 0.25$

$= 1.5$ trits/symbols

Arithmetic Coding

The main difference between Arithmatic coding and Huffman Coding is that Arithmatic coding provides a low average code length or low average code size than Huffman Code.

Arithmatic coding is a form of entropy encoding used in lossless data compression.

Arithmatic coding differs from other forms of entropy encoding such as Huffman coding in that, rather

than seperating the inputs into component symbols and replacing each with a code, arithmatic coding encodes the entire message into a single number, an arbitrary precision factor q, where $0 < q \leq 1$.

→ This code works well for sequences with low entropy.

Eg: Main characteristics

→ It is a stream based encoding

→ Lossless Data compression

→ Non Block code

→ Efficient code

Arithmetic Encoding Procedure

Q. Consider the transmission of a message "went." comprising a string of characters with ~~prob~~ probability

e → 0.3, n → 0.3, t → 0.2, w → 0.1,

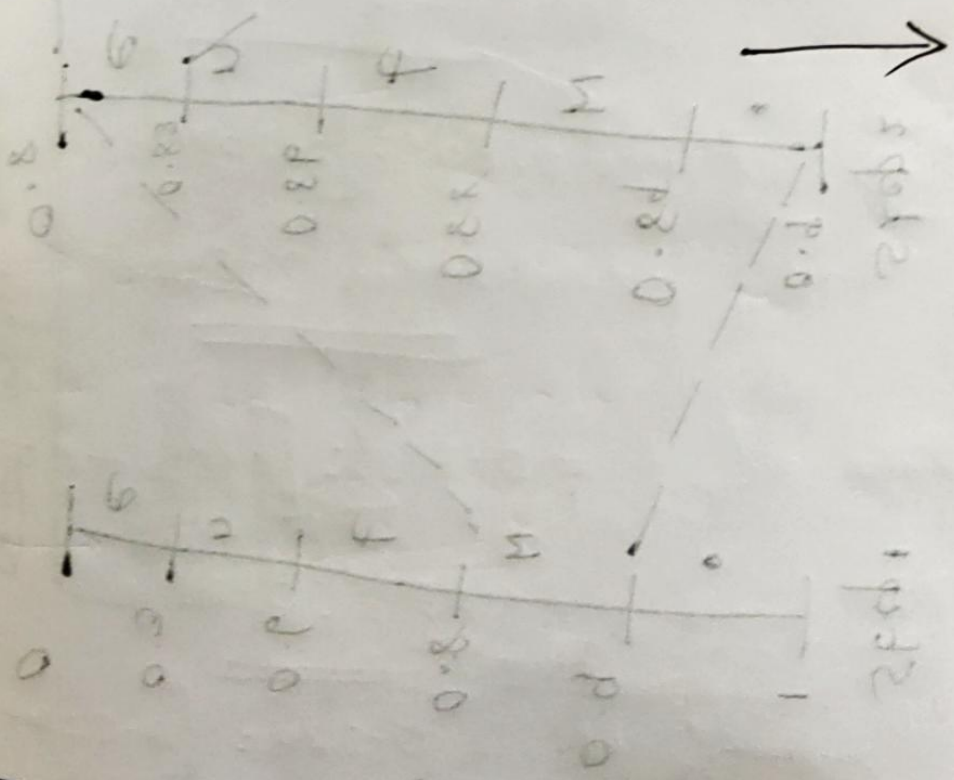
. → 0.1. Perform Arithmetic Encoding for lossless compression

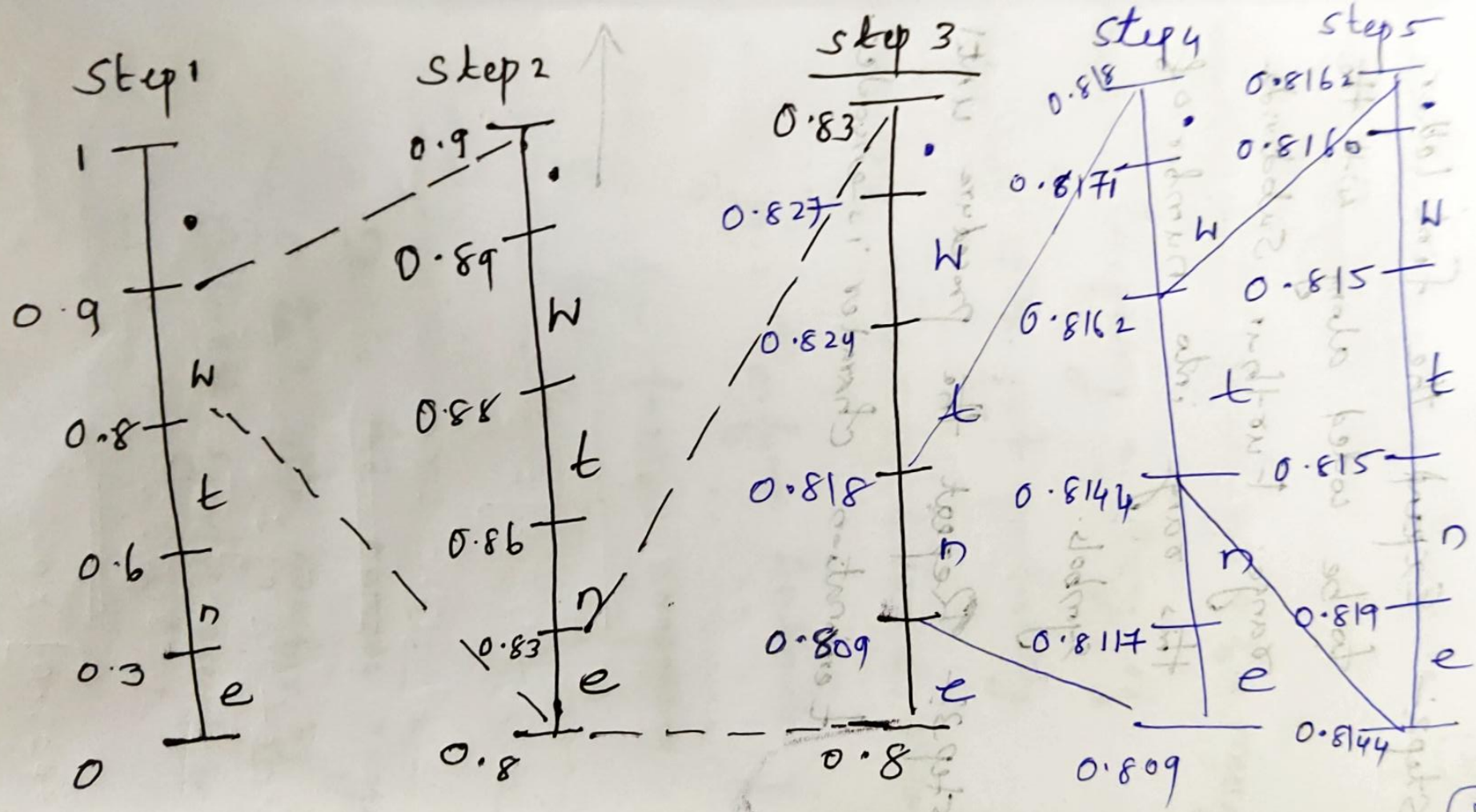
Ans: Procedure

step 1:- Divide the numeric range 0 to 1 into number of different symbol present in the message.

step2:- Expand the first letter to be coded along with the range. Further subdivide this range into number of symbols.

step3:- Repeat the procedure until termination character is encoded.




Step 1
1
0.9
0.8
0.6
0.3
0
W
t
n
e
Step 2
0.9
0.89
0.88
0.86
0.83
0.8
W
t
n
e
Step 3
0.83
0.827
0.824
0.818
0.809
0.8
W
t
n
e
Step 4
0.818
0.8171
0.8162
0.8144
0.8117
0.809
W
t
n
e
Step 5
0.8162
0.8160
0.815
0.815
0.819
0.8144
W
t
n
e

Step 2 Calculations

d = upper bound − lower bound $= 0.9 - 0.8 = 0.1$

Range of symbol = lower limit : lower limit + d (probability of symbol)

∴ Range of 'n' = $0.83 : [0.83 + 0.1\,(0.3)]$

$= 0.83 : 0.86$

Range of 't' $= 0.86 : [0.86 + 0.1\,(0.2)]$

$= 0.86 : 0.88$

Range of 'N' $= 0.88 : [0.88 + 0.1\,(0.1)]$

$= 0.88 : 0.89$

Range of 'I' $= 0.89 : 0.9$

Step 3 Calculation

d = Upper bound − lower bound

= 0.83 − 0.8 = 0.03

We know that, → Range of symbol = lower limit :

lower limit + d (Probi of symbol)

∴ Range of 'e' = 0.8 : [0.8 + 0.03 (0.3)]

= 0.8 : 0.809

Range of 'n' = 0.809 : [0.809 + 0.03 (0.3)]

= 0.809 : 0.818

Range of 't' = 0.818 : [0.818 + 0.03 (0.2)]

= 0.818 : 0.824

Range of 'W' = 0.824 : [0.824 + 0.03(0.1)]

$= 0.824 : 0.827$

Range of ~~0.82~~ 'i'

$= 0.87 : [0.827 + 0.03(0.1)]$

$= 0.827 : 0.83.$

Step 4 Calculation

Similarly,

We calculate,

Range of 'e' = 0.809 : 0.8117

Range of 'n' = 0.8117 : 0.8144

Range of 't' = 0.8144 : 0.8162

Range of 'N' = 0.8162 : 0.8171

Range of '.' = 0.8171 : 0.818

Step 5

Range of 'e' = 0.8144 : 0.8194

" 'n' = 0.81946 : 0.81548

" 't' = 0.81548 : 0.81584

" 'w' = 0.81584 : 0.81602

" '.' = 0.81602 : 0.8162

Finally,

the Arithmatic code Word is the codeword between the limits of the termination character.

ie, $0.81602 < \text{codeword} < 0.8162$

Generation of Tag:

$$\text{Tag} = (\text{Upper limit of codeword} + \text{Lower limit of codeword})/2$$

$$\text{Tag} = \frac{(0.8162 + 0.81602)}{2}$$

$$= 0.81611$$

$$Tag = \left(\text{Upper limit of codeword} + \text{Lower limit of codeword}\right)/2$$

$$Tag = \frac{(0.8162 + 0.8160)}{2}$$

$$= 0.81611$$

Comparison of Huffman code and Arithmatic code

Arithmatic code	Huffman code
• It is not a statistical Method	• It is a statistical Method
• It yields an optimum result.	• It doesn't yields an optimum result.
• There is no one to one correspondence b/w source symbol and code word	• There is one to one correspondence b/w source symbol and code word.

Arithmatic Code	Huffman code
• There is only one unique code to entire message	• seperate symbols are assigned for each symbols.

Advantages of Arithmatic code over Huffman code

→ Arithmetic coding typically has a better compression ratio than Huffman coding, as it produces a single symbol rather than several seperate codewords.

→ Arithmatic coding surpasses the Huffman techniques in its compression ability.

→ In terms of complexity, arithmetic coding is better than Huffman coding.