Text

- flexibility & ease of use enhances learning
- plaintext: fixed size chars having similar appearance.
- formatted text: appearance changed by font
- hypertext: link different docs.
- ASCII table
- Unicode standard: represent international chars from various languages.
- OCR: generate text automatically from a scanned version of paper document.
- text can be compressed while saving it.
- Types of Text:

(1.) Unformatted

↳ fixed size chars from limited set.

↳ ASCII table.

↳ each char is represented by unique 7 bit binary code ($2^7 = 128$ words).

↳ same height characters.

↳ printable chars

↳ control chars: backspace, linefeed, carriage return, space, delete, escape.

↳ numeric code is retrieved from ROM table in binary form & substituted for actual key for internal processing.

↳ extended ASCII: 256 characters including 128 std & remaining as small graphical symbols.

(2) Formatted Text

↳ apart from actual alphanumeric chars, other control chars. change the appearance of the chars.

↳ bold, underline, italics

↳ structure any document.

(3.) Hypertext.

↳ method of transmitting information

↳ technical doc may contain collection of relatively independent information

↳ cross references.

↳ HTML docs.

↳ underlined text string is anchor & the doc. opened as a result of clicking is target document.

↳ Architecture: 3 layers

Presentation	Hypertext Abstract M/C	Storage
uppermost layer. All fn is executed how data is presented	knows the str of the document. least sys dependency.	database. specific properties of media

Unicode Standard

(2)

↳ universal character coding
↳ encoding multilingual text enabling permanent exchange of data.

↳ types
- characters : smallest component of written language.
- glyphs : represents shapes of displayed characters.

↳ Mapping methods
- UTF (Unicode Transformation format)
- UCS (Universal character set)

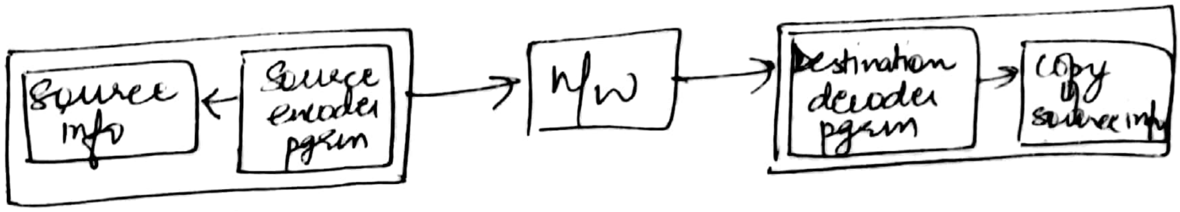
① UCS-4, UTF-32
↳ 32 bit for each character.
↳ fixed length encoding
↳ not efficient
↳ 4 bytes = 32 bits.

② UTF-16
↳ 16 bit encoding format.
↳ Larger than this, nos. are expressed as a combination of two - 16 bit nos.

③ UTF-8
↳ bits divided into series of 8 bit nos.

Compression

- Source encoder
- destination decoder



- lossless compression :- reduce amt. of source info to be transmitted st. during decompression there is no loss of information.
  ↳ reversible
  ↳ transfer of txt file over N/w.

- lossy compression :- aim is normally not to reproduce an exact copy of source info after decompression but rather a version.
  ↳ transfer of digitized images & audio & video.

- entropy encoding
  ↳ lossless & independent of type of info being compressed.
  ↳ run length encoding : when source info comprises long substrings of same character or binary digit.
    ↳ string is transmitted in form of different set of codewords

Scanned by CamScanner

↳ destination knows the set of codewords being used, it can interpret each codeword & o/p the no. of chars or bits.

↳ ex: scanning typed docs: long string of 0's & 1's.

↳ 000011111001 1 . . . : o/p of scanner

0, 4, 1, 5, 0, 2, 1, 2 . . .

↳ individual decimal digits sent in binary form & fixed no. of bits per codeword.

↳ Statistical encoding: use of variable length codewords is not easy. The destination must know the set of codewords being used by the source.

↳ for decoding to be correct, prefix property is req'd: using set of variable length codewords c̄ shortest codewords for most frequenty occuring symbol.

↳ Huffman encoding

↳ minimum avg. no. of bits req'd to transmit a particular source stream is called entropy of the source

Shannon formula: Entropy, $H = -\sum_{i=1}^{n} P_i \log_2 P_i$

$n \rightarrow$ no. of diff symbols in source stream

$P_i \rightarrow$ probability of occurance . . .

- Font
  - Font appearance
    - font name : description of character appearances stored in font files
    - vector format : mathematical description of characters.
    - true type : apperance of fonts remain same
    - bitmap : character is described as collection of pixels.
  - font size & style
    - changing the horizontal gap b/w the characters called kerning & vertical gap b/w 2 lines of text called leading.

- Insertion of Text
  - using keyboard
  - copy & paste
  - OCR

- File formats
  - TXT (text) : data encoded using ASCII & Unicode
  - DOC (document) : not considered a doc exchange format
  - RTF (rich text format) : by microsoft for cross platform document exchanges. human readable codes similar to HTML codes.
  - PDF (portable doc format) by Adobe systems
  - PS (postscript) : page description language for desktop publishing. HLL that can describe contents of a page such that it can accurately display on o/p devices.

Average no. of bits/codeword $= \sum_{i=1}^{n} N_i P_i$

eg: Six diff chars M, F, Y, N, O & 1 each $\bar{c}$ relative frequency of occurance of 0.25, 0.25, 0.125, 0.125 & 0.125 resp.

If the encoding algo uses the following set of codewords:

M = 10, F = 11, Y = 010, N = 011, O = 000, 1 = 001

Compute:

(1.) avg. no of bits per codeword

$$= \sum_{i=1}^{6} N_i P_i = 2\times0.25 + 2\times0.25 + (3\times.125)\times4$$

$$= 2.5$$

(2.) Entropy of source

$$= -\sum_{i=1}^{6} P_i \log_2 P_i = -\left(2\left(0.25 \log_2 0.25\right) + 4\left(.125 \log_2 .125\right)\right)$$

$$= 2.5$$

(3.) min. no. of bits req'd assuming fixed length codewords: 6 diff chars, ∴ 3 bits

(8 combinations).

- Source encoding : exploits particular properties of source info to produce alternative form of representation that is either a compressed version of original form or more amenable.

↳ Differential encoding

↳ where amplitude of value or signal covers large range but diff. in amplitude b/w successive values/signals is small.

↳ set of smaller codewords used.

↳ can be lossy or lossless & depends on no. of bits used to encode difference values.

↳ Transform encoding

↳ transforming source info from one form into another, other form lending itself more readily to the application of compression.

↳ no loss

Scanned by CamScanner

Text Compression

↳ text is represented as strings of chars selected from a defined ~~char~~ set.

↳ compression algo associated c̄ text must be lossless as a loss of any char can modify the meaning of complete string.

↳ entropy encoding, statistical encoding

uses single character as basis of deriving an optimum set of codewords
eg: Huffmann & arithmetic coding algos.

uses variable length strings of characters.
eg: Lempel-Ziv algo.

↳ types of coding for text

Static

↳ text to be compressed has known characteristics in terms of chars used & their relative frequency of occurrence.

↳ optimum set of variable length codewords is derived c̄ shortest codewords for most frequently occurring chars.

dynamic

↳ for more general applis

↳ optimum set of codewords likely to vary from one transfer to another.

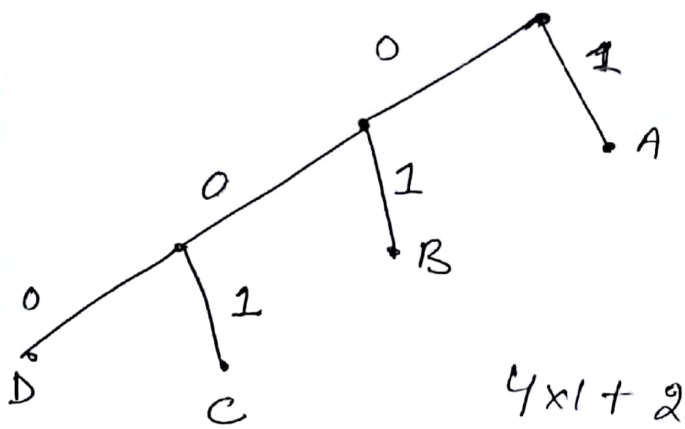
↳ receiver is able to dynamically compute the same set of codewords being used at each transfer.

(1.) Static Huffman Coding

↳ char string to be transmitted is first analyzed & the character types & their relative frequency determined.

↳ create an unbalanced tree

↳ binary tree with branches having value 0 or 1.
left right

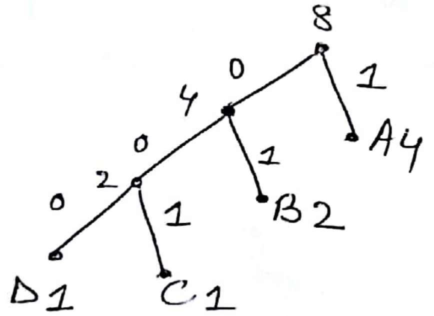


A = 1 (4)
B = 01 (2)
C = 001 (1)
D = 000 (0)

AAAABBCD

4×1 + 2×2 + 1×3 = 14 bits

↳ codewords are determined by tracing the path from the root node out to each leaf & forming a string of binary values associated with each branch traced.



weight order = D1 C1 2 B2 4 A4 8

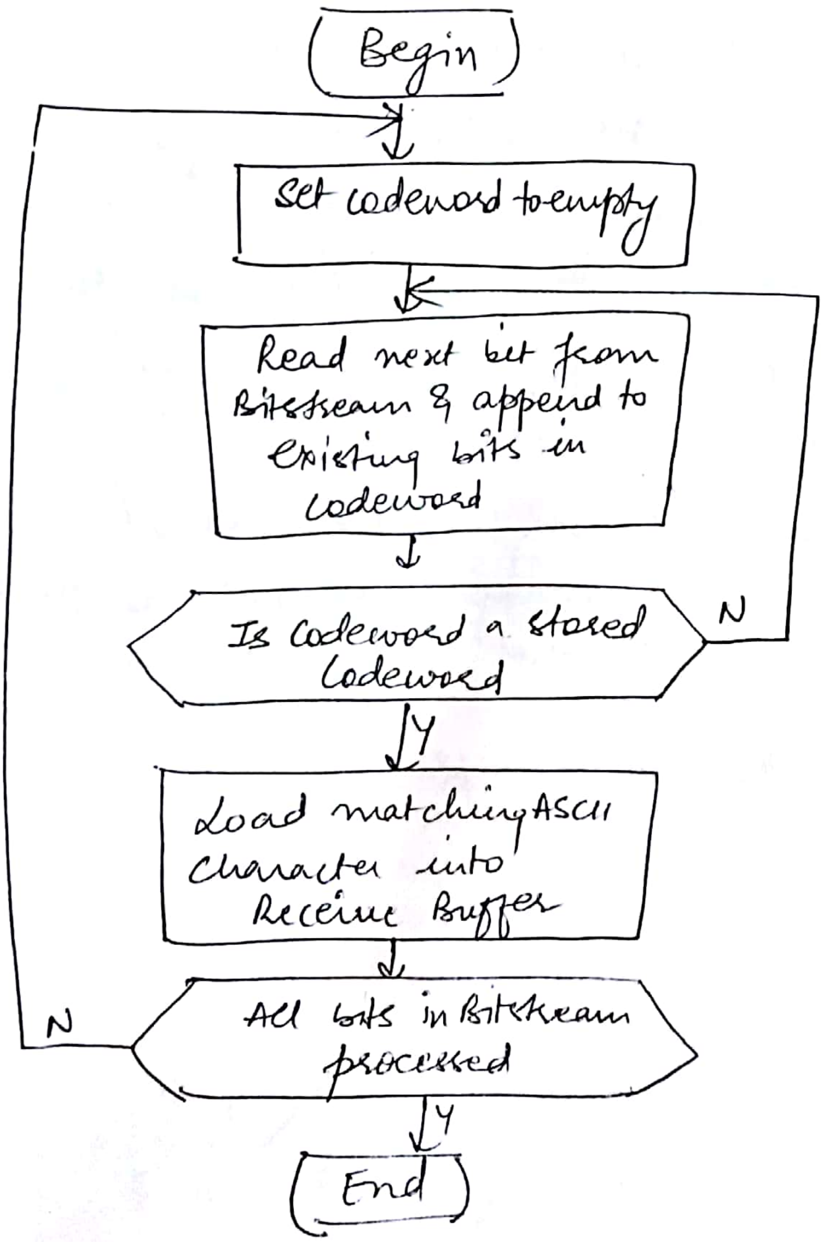
↳ optimum tree if the resulting list increments in weight order.

↳ every character c is encoded has a variable no. of bits, the received bitstream must be interpreted in a bit oriented way.

↳ Shorter codeword will never form the start of a longer codeword: prefix property

received bitstream can be decoded simply by carrying out a recursive search bit by bit until valid codeword is found.

ALGORITHM



↳ assumes a table of codewords at the receiver & it also holds the corresponding ASCII codewords
↳ Bit-stream variable holds the received bitstream & Codeword variable holds the bits in each codeword

② Dynamic Huffman Coding

↳ encoder & decoder build the Huffman tree & hence the codeword table dynamically as the characters are being transmitted/received.

↳ if the char to be transmitted is currently present in the tree, its codeword is determined & sent in the normal way.

↳ if char isn't present or it is 1st occurrence, it is transmitted in its uncompressed form.

↳ encoder updates its Huffman tree by introducing the new char into the tree.

↳ receiver can determine the char from the codeword & also make modifications in its own copy so that it can interpret the next codeword.

↳ example: This is simple...

A thru I

0.05, 0.45, 0.55, 0.10, 0.15

Msg to be transferred comprises chars A thru H. Relative freq. of occurrence is given as:

A & B = 0.25 ; C & D = 0.14 ; E, F, G & H = 0.055.

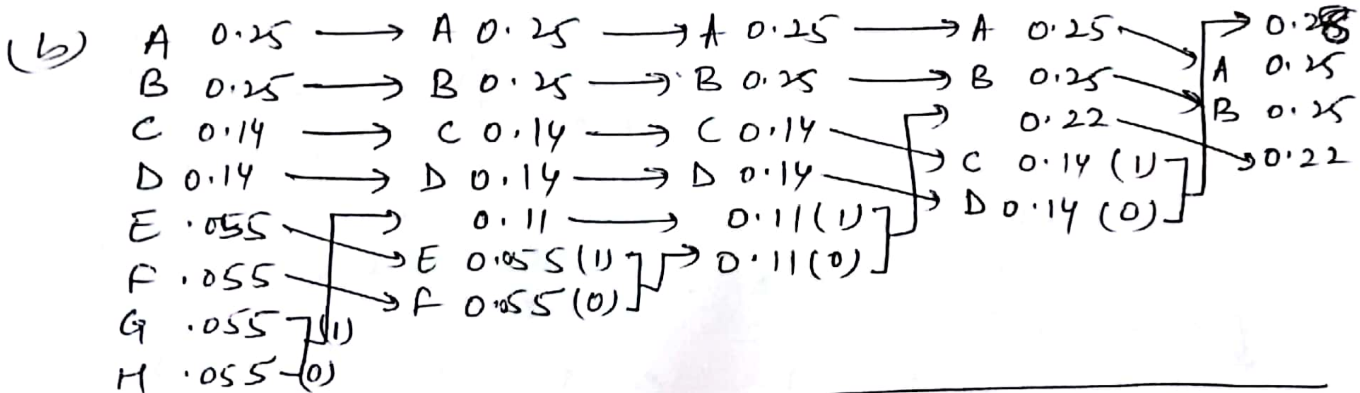
(a) Use Shannon's formula to derive min. avg. no. of bits/char.

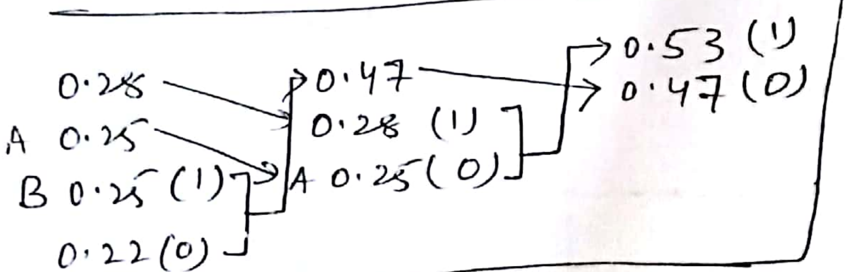
(b) Use Huffman coding to derive a codeword set & prove it is the min. set by constructing the corresponding Huffman code tree.

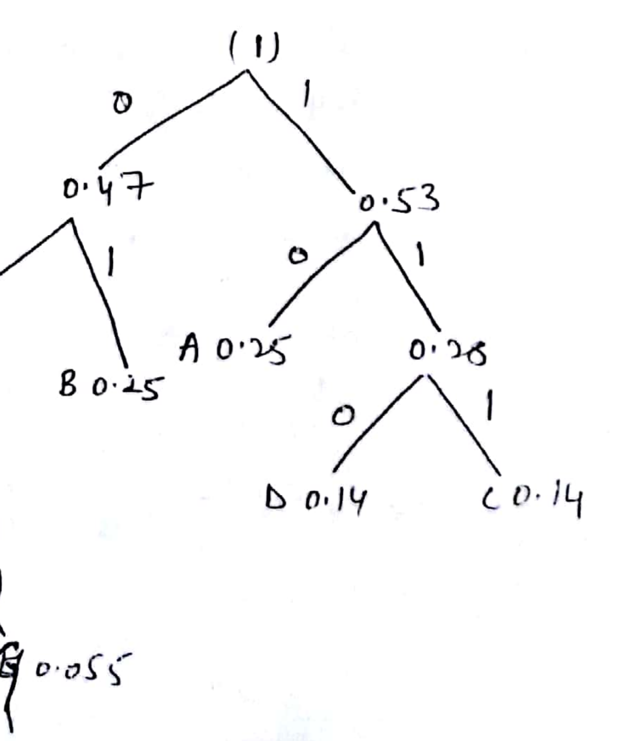
(c) Derive avg. no. of bits/char for ur codeword.

(a) $H = -\sum_{i=1}^{8} P_i \log_2 P_i = 2.175$ bits per codeword.

(b)







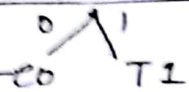
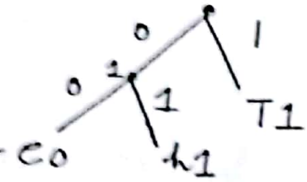
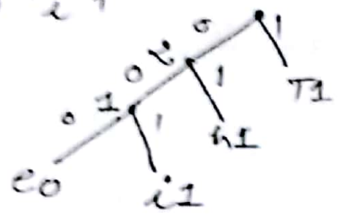
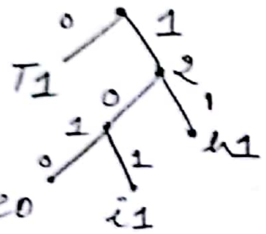
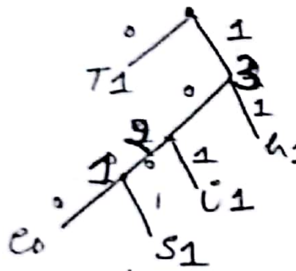
A = 10
B = 01
C = 111
D = 100
E = 0001
F = 0000
G = 0011
H = 0010

Q. Dynamic Huffman coding. This is simple...

␣ = space character   This␣is␣simple

Initial Tree

e0 : empty leaf.

Character	o/p	Updated Tree	list
T	"T"		e0T1
h	0"h"		e0 h1 1 T1
i	00"i"		e0 i1 1 h1 2 T1 ×
			e0 i1 1 h1 T1 2
s	100"s"		e0 s1 1 i1 2 h1 T1 3

Arithmetic Coding

→ Huffman coding achieves Shannon value only if character prob. are integer powers of 1/2.

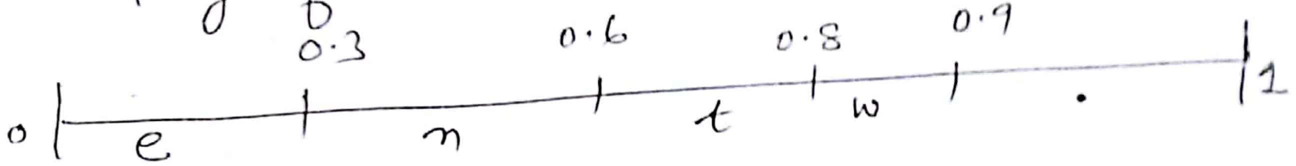
→ ∴ codewords produced are hardly optimum.

→ static coding mode (basic)

↳ e = 0.3, n = 0.3, t = 0.2, w = 0.1, . = 0.1

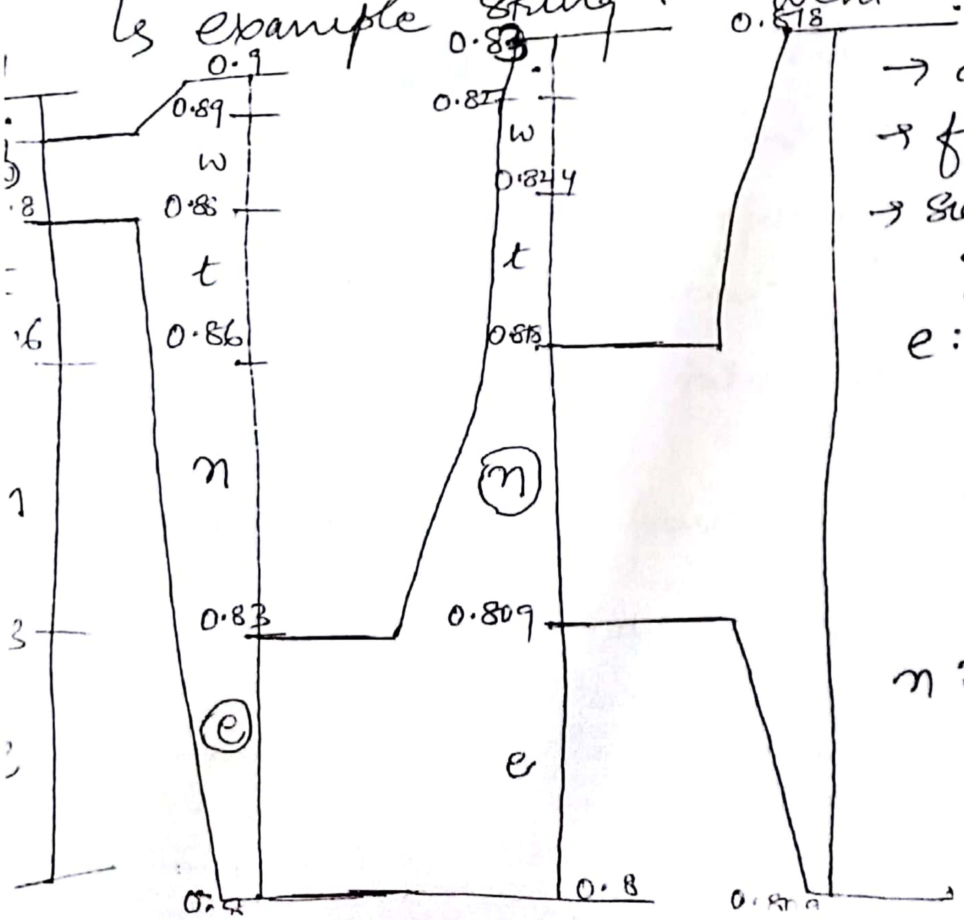
↳ when '.' is encountered, end of string.

↳ single codeword for each ~~character~~ encoded string of chars.



range depending on probability.

↳ example string : "went".



→ divide into segments

→ first char of string

→ subdivide acc. to given probabilities

e : 0.8 to 0.83

(0.8 + .3 × .1)

↓ start ↓ prob ↓ diff in range (0.9 − 0.8)

n : (0.809 + 0.3 × .03)

↓ (.83 − .8)

→ decoder knows set of chars that are present in the encoded msg as well as the segment & its range.

eg: if received codeword is 0.8161, first is 'w' bcoz it is in the range

→ no. of decimal digits in the final codeword ↑ses linearly $\bar{c}$ no. of chars in the string to be encoded.

Lempel-Ziv coding

↳ uses strings of chars for compression.

↳ table shared b/w sender & receiver.

↳ contains all possible char strings that occur in text to be transferred.

↳ rather than sending ASCII codewords for the string, its index of location in the table is sent.

↳ receiver uses this info to recover the text.

↳ used as dictionary.

↳ dictionary based compression.

↳ dictionary for spell check & compression.

$$\begin{array}{r} 3 \\ 4.7 \\ 15 \\ \hline 23\ 5 \\ 4\ 7 \\ \hline 70.5 \end{array}$$

eg: "multimedia" → 10 chars in word.

Using ASCII, no. of bits = 70 bits.

Using this approach, typically 25000 words,

∴ 15 bits sufficient ($2^{15} = 32768$).

∴ Compression ratio :- 70 : 15 = 4.7 : 1

↳ shorter words have lower compression ratio & longer words have higher compression ratio.

Q: LZ algo is to be used to compress text file prior to its transmission. If avg no of chars per word is 6, & dictionary has 4096 words, Derive avg compression ratio achieved relative to using 7 bit ASCII codewords.

→ 4096 words → 2^{12} ∴ 12 bits.

ASCII → $7 \times 6 = 42$ bits

Ratio :- $42 : 12 = 3.5 : 1$.

Lempel-Ziv-Welch coding

↳ build content of dictionary dynamically as the text is transferred.

↳ initially contains only ASCII char set.

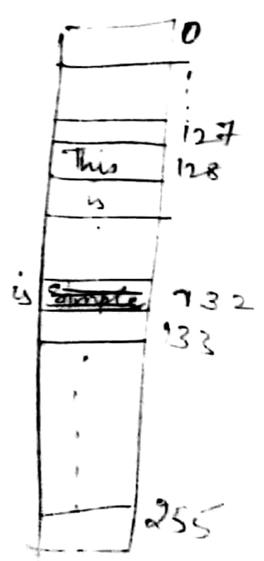
↳ remaining data is added as new text is encountered.

↳ example: char set has 128 chars & dictionary is limited to 4096 entries, then first 128 chars contain single chars & remaining 3968 entries would contain strings of 2 or more chars that make up the words in text being transferred.

eg: "This is simple as it is . . ."

↳

Index = 8 bits



As a char is encountered, 't', 'h', 'i', 's', next char is read as 'space' meaning a word first terminated, ∴ "this" is stored as string in a fresh location & index 128.

↳ can be dynamically ↑sed.