e

0.83
0.827 —
0.824 — w
0.818 — t
0.809 — n
e
0.8

0.824 + 0.01 × 0.03
0.818 + 0.03 × 0.02
0.8 + 0.03 × 0.3
d = 0.03

t

0.818 + 0
w = 0.8162 + 0.1 × 0.009 = 0.8171
t = 0.8144 + 0.2 × 0.009 = 0.8162
n = 0.8117 + 0.3 × 0.009 = 0.8144
e = 0.809 + 0.3 × 0.009 = 0.8117
0.809

t

0.8162 —
•
w
b
n
e
0.8144

→ Any code in this range represents went.

0.81548 + 0.0018 × 0.1
0.81602

0.81548 + 0.0018 × 0.2
= 0.81584

0.81494 + 0.0018 × 0.3
= 0.81548

0.8144 + 0.0018 × 0.3
= 0.81494

d = 0.0018

avg = (0.8162 + 0.81602) / 2

we send this average

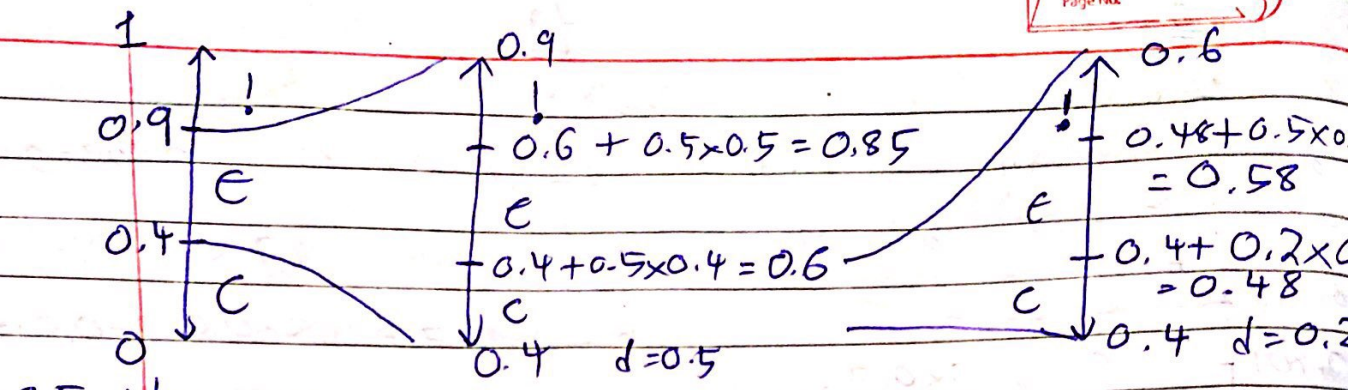
DECODING

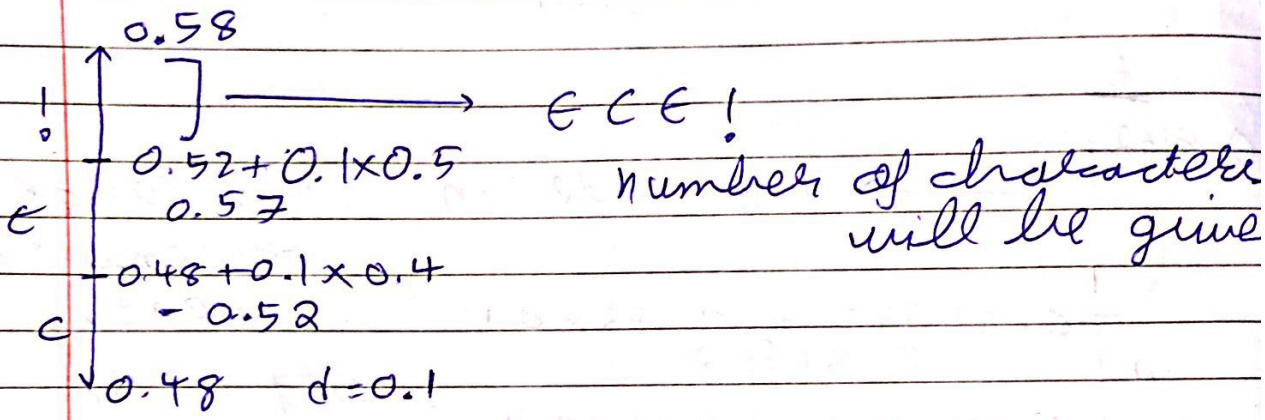
C = 0.4 E = 0.5 ! = 0.1

Code = 0.572

Order = CE!

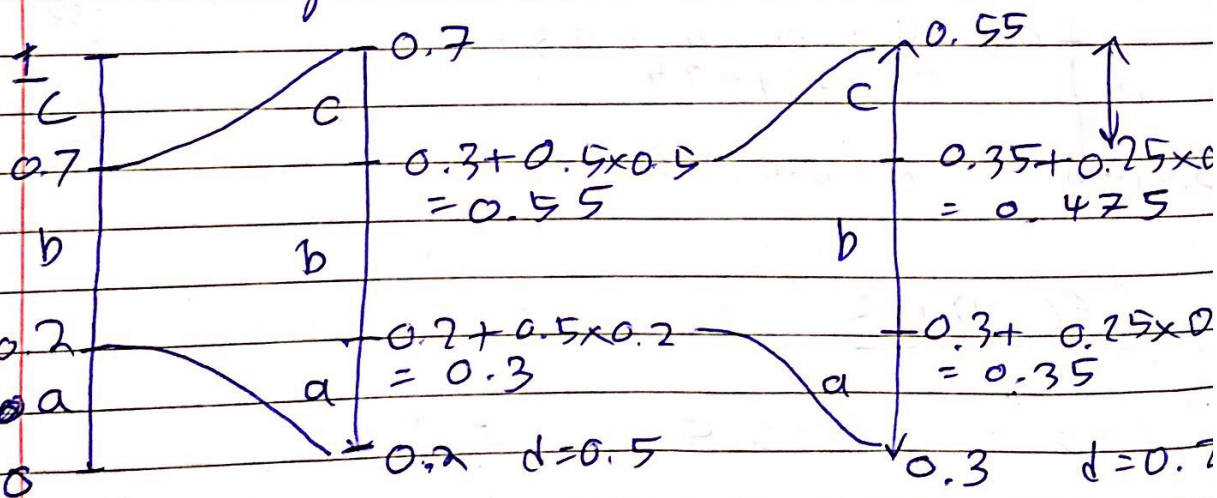
vijeta





(2) a = 0.2 b = 0.5 c = 0.3

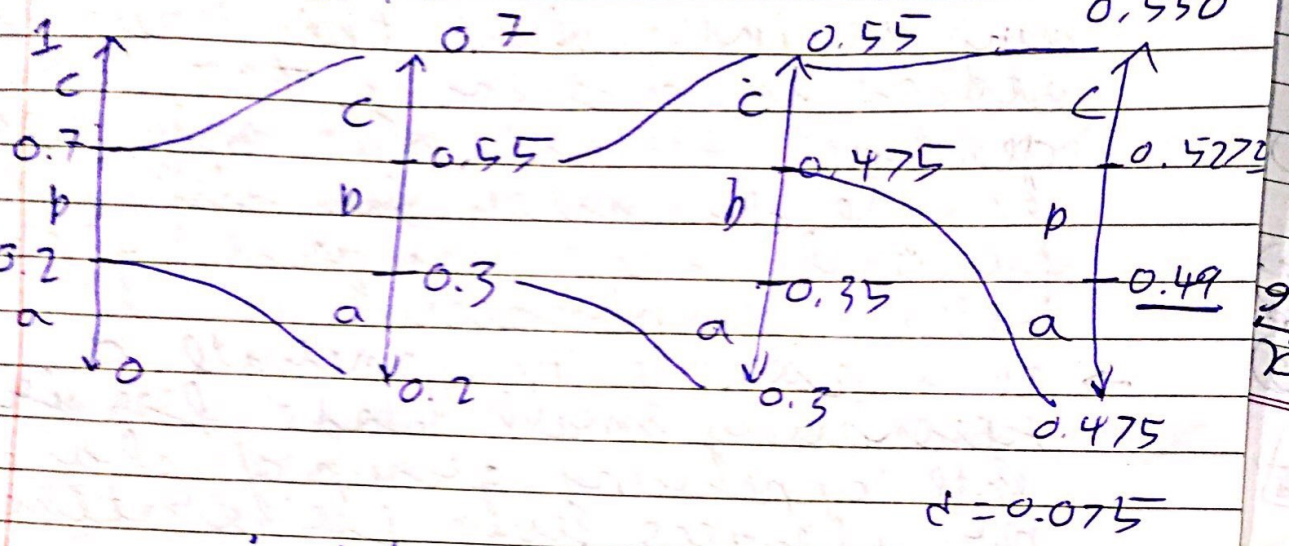
string = bbc



$$\frac{0.475 + 0.55}{2} = 0.5125$$

b

DECODE 0.49



$d = 0.075$

∴ b b c a

DICTIONARY BASED COMPRESSION TECHNIQUE

- Static
  - DIAGRAM CODER
- Dynamic (ADAPTIVE)
  - LZ77
  - LZ78
  - LZW

→ for long sentences and long phrases.
→ Compression method like Shannon Fehicon, Huffman and arithmetic coding take symbol of input string and produce equivalent code after exploiting the probability of symbols
→ Dictionary based coding depends on concept of maintaining the indices.
→ In dictionary based compression technique, words sentences and phrases are represented as token

vijeta

Since dictionary is of fixed size, therefore only those frequent group of characters are placed in dictionary whose probability of occurrence is relatively high. This technique shows significant improvement on Shannon & Huffman

Q. $\Sigma = \{a, b, c, d, r\}$. Based on knowledge about source, we build the dictionary shown in the table

CODE	STR ENTRY	TEXT
000	a	abracadabra
001	b	
010	c	
011	d	
100	r	
101	ab	
110	ac	
111	ad	

→ Dictionary is prepared based on most frequent pair of each and every letter in frequency

a b r a c a d a b r a

ab r ac ad ab r a

101 100 110 111 101 100 000

→ 101100110 111 101 100 000
will always produce least smallest string because dictionary is created on basis of frequency of characters.

Q. Sentence comprises of {A, B, C, D, E, F}

Code	Entry	TEXT
0000	a	a f a b d a a b d e a a b
0001	b	
0010	c	
0011	d	
0100	e	
0101	f	
0110	ab	
0111	ac	
1000	bc	
1001	ef	
1010	de	
1011	aab	
1100	abc	
1101	bac	
1110	bcde	
1111	aabb	

0000 0101 0110 0011 1011 1010 1111

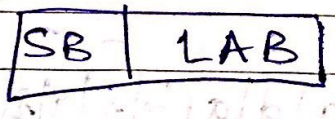
ADAPTIV DIC. TECH

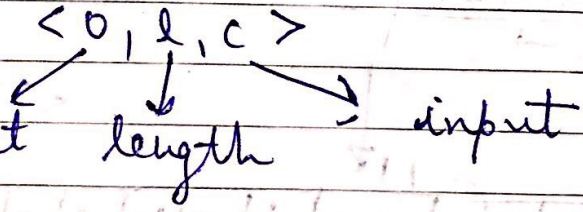
(1) LZZ 77 / LZ1 / Sliding Window

When i/p text is taken for compression, it starts building dictionary. Thus, the symbols, words & phrases which have been scanned previously appears again in the same text. When the token corresponding to the entry of that phrase in dictionary is generated. LZ77 Scheme use the text window which is divided into two parts:-

1. Scanned Buffer / Search Buffer (SB)
2. Look Ahead Buffer (LAB)

⚡ Search Buffer contains previously scanned text and Look Ahead Buffer contains upcoming input text.



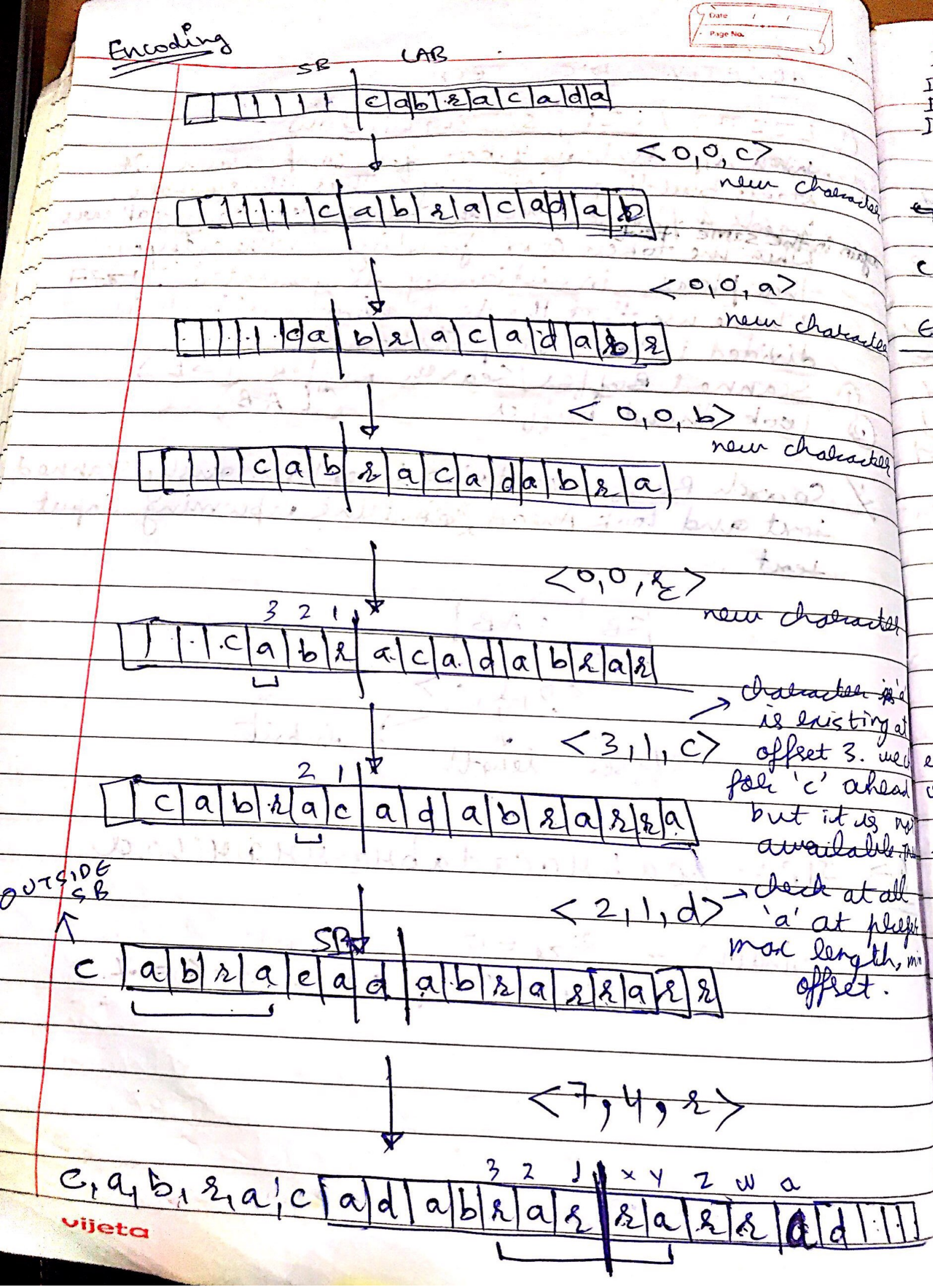


Q⇒ I/P:- cabracadabrarrarrad

Window:- Size = 13

∴ LAB = 6, SB = 13 - 6 = 7

Encoding



Index 3 = Index x
Index 2 = Index y
Index 1 = Index z
Index x = Index w
Index y = Index a

(Continue in LAB) <3,5,d>

c,a,b,r,a,c,a,d,a,b,r,a

ENCODED = Set of triplets <offset, length, char>

Decoding

<0,0,c> → c

<0,0,a> → ca

<0,0,b> → cab

<0,0,r> → cabr

Copy char 3 for length 1 and then c

<3,1,c> → cabrac

<2,1,d> → cabracad

<7,4,r> → cabracadabra

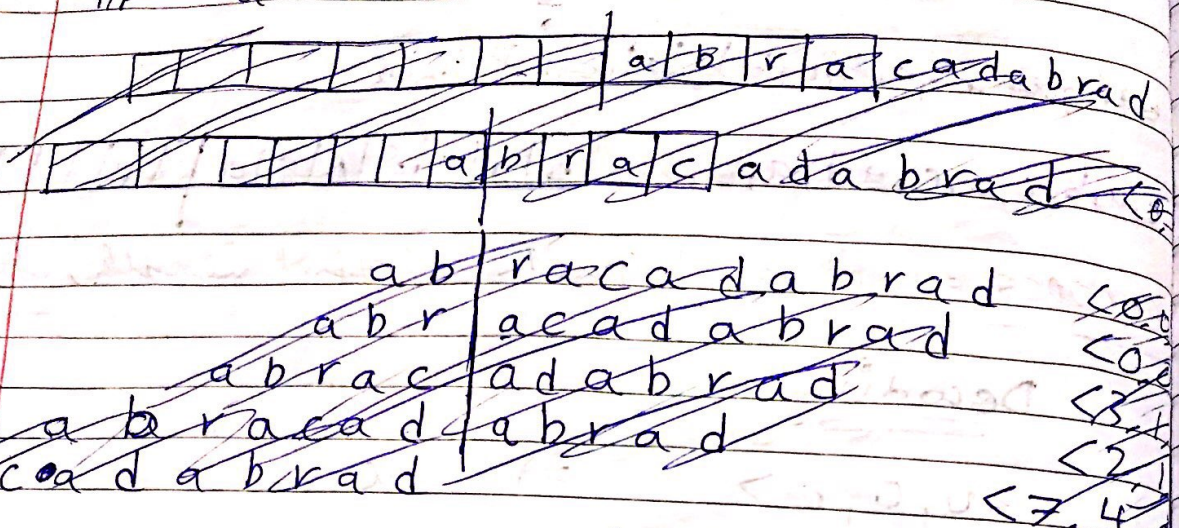
<3,5,d> → cabracadabrarrarrad

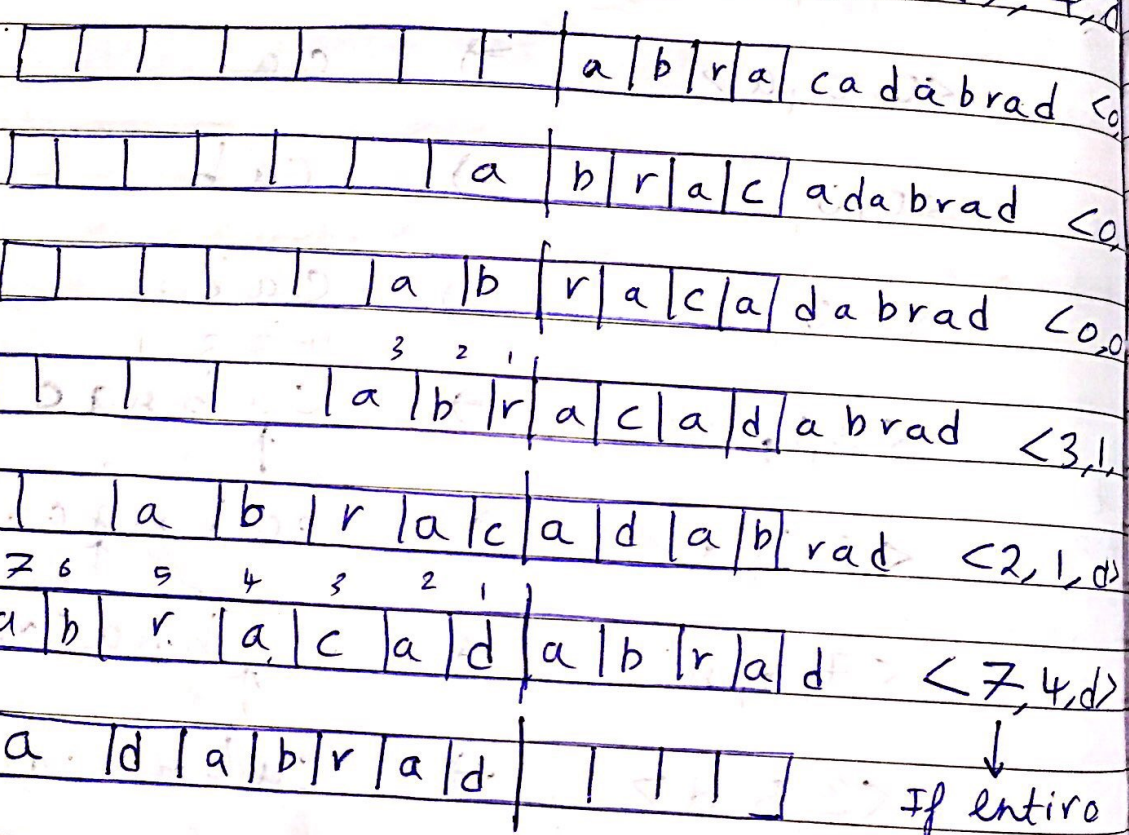
Take in continuation

vijeta

eg→ SB = 7 LAB = 4

I/P abracadabrad





abrac a d a b r a d

↓

If entire string is found at end, add '-' character

ENCODED STRING

<0,0,a> <0,0,b> <0,0,r> <3,1,c> <2,1,d> <7,4,d>

vijeta

DECODING

<0,0,a>	a
<0,0,b>	a b
<0,0,r>	a b r
<3,1,c>	a b r a c 3 2 1
<2,1,d>	a b r a c a d 5 4 3 2 1
<7,4,d>	a b r a c a d a b r a d 7 6 5 4 3 2 1

FINAL STRING → abracadabrad

Drawback:

→ The search buffer contain part of input msg being ~~coded~~ coded.

→ The look ahead buffer contain upcoming stream of char

→ Size of LAB and SB is fixed

→ LAB is moved over input stream in ~~four~~ forward direction. First char in LAB is checked.

→ If it is there in SB, a pointer in SB search it

→ The distance of match from LAB is called offset.

→ After match occured, the pointer is moved in forward direction to check if next consecutive character has a match in the next symbol in LAB

→ Total no. of consecutive char matched is known as length of match.

→ The compression attempt to match longest match in LAB.

→ After match has been found <offset, length, symbol>. symbol corresponds to first symbol from where match has been found

→ Drawback: Searching for phrases is complex task and it is tedious affair if size of dictionary ↑

LZ78

A B C D A B C A B C D A A B C A B C E

Encoder O/P	Index	Entry
<0, A>	1	A (New char)
<0, B>	2	B
<0, C>	3	C
<0, D>	4	D
<1, B>	5	AB // Character pattern then get
<3, A>	6	CA
<2, C>	7	BC
<4, A>	8	DA
<5, C>	9	ABC

<9, E> 10 ABCE

if string doesn't have ∈

<9_>

DECODE

Encoder o/p	Index	Entry
<0, A>	1	A
<0, B>	2	B
<0, C>	3	C
<0, D>	4	D
<1, B>	5	AB
<3, A>	6	CA
<2, C>	7	BC
<4, A>	8	DA
<5, C>	9	ABC
<9, E>	10	ABCE

String ABC DABCABC DAABC ABCE

write all entries together

LZW

wabbabwabbabwabbabwabbabwabbabwoowoowoo

Index	Entry
1	b
2	a
3	b
4	o
5	w

→ GIVEN

is dictionary it, offset index

b

1 → 1

3

vijeta

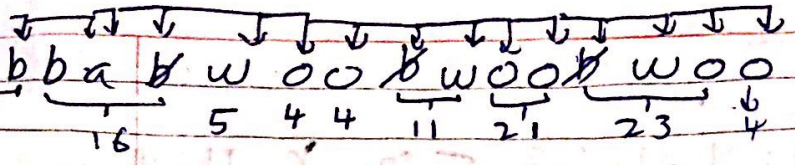
eg 1 w a b b a b̸ wa b b a b̸ wa b b a b̸ w a

5 2 3 3 2 1 6 8 10 12 9 11

Index	Entry
1	b̸
2	a
3	b
4	o
5	w
6	wa
7	ab
8	bb
9	ba
10	ab̸
11	b̸w
12	wab
13	bba
14	ab̸w
15	wabb
16	bab̸
17	b̸wa
18	abb
19	bab̸w
20	wo
21	oo
22	ob̸
23	b̸wo
24	oob̸
25	b̸woo

Encoded string:
5, 2, 3, 3, 2, 1, 6, 8, 10, 12, 9, 11, 7, 16, 5, 4, 4, 11, 21, 23, 4

→ Start with the first char. If not present in dictionary add it and move to next char. If present then check s_i s_{i+1} is in dictionary or not. This is continued till you get a word not present in dictionary
→ Add it to dictionary and encoded string is added with index of last value found in dictionary
→ start the process from s_{i+n} where $s_i, s_{i+1} \ldots s_{i+n}$ wasn't found in dictionary



When dictionary not given

1	w
2	a
3	b
4	bb
5	ab
6	b
7	wa
8	ab
9	bb
10	bab
11	bw
12	wab
13	bba
14	abw
15	wabb
16	babw
17	wo
18	oo
19	ob
20	bwo
21	oo
22	obw
23	woo
~~24~~	
~~25~~	

1, 2, 3, 3, 2, 6, 1, 2, 3, 4, 6, 7, 9, 5, 12, 10, 1, 18, 18, 11, 18, 19, 17, 18

ENCODED STRING ↑

vijeta

eg 2 LZW the/rain/in/spain/falls/mainly/on/the/plain

the/rain/in/spain/falls/mainly/on/the/plain

1,2,3,4,5,6,7,8,4,7,8,4,13,14,6,10,4,18,6,20,20,20,13,4,25,15,8,20,29,4,31,11,1,2,3,4,14,20,26

Index	Entry
1	t
2	h
3	e
4	/
5	r
6	a
7	i
8	n
9	/i
10	in
11	n/
12	/s
13	sp
14	p
15	ai
16	in/
17	/f
18	~~fa~~f
19	al
20	l
21	ll
22	ls
23	s/
24	/m
25	m
26	~~aa~~
	ain

Index	Entry
~~11~~ 27	~~27~~ nl
28	ly
29	y
30	/o
31	on
32	n/t
33	th
34	ne
35	e/
36	/p
37	pll
38	la
~~39~~	~~ai~~
~~40~~	

ENCODED:

1,2,3,4,5,6,7,8,4,7,8,4,13,14,6,10,4,18,6,20,20,20,13,4,25,15,8,20,29,4,31,11,1,2,3,4,14,20,26

vijeta

→ **DECOMPRESS:**

r a t at ata t ✗ a ✗ ra t✗ at ✗a ✗r at

3, 1, 4, 6, 8, 4, 2, 1, 2, 5, 10, 6, 11, 13, 6

Index	Entry	
1	a	→ Initial dictionary is always given.
2	✗	
3	r	
4	t	
5	ra	
6	at	
7	ta	(tat, but ta is not available so take ta only)
8	ata	
9	atat	
10	t✗	
11	✗a	
12	a✗	
13	✗r	
14	rat	
15	t✗a	
16	at✗	
17	✗a✗	
18	✗ra	
~~19~~	~~a~~	
~~20~~		

we take 8 = at_ as No 3 character is available till yet, it will be unique in dictionary. We replace 8 by "at_" in the string. Then we start checking from 6

a ✓

at ✓

ata ✗ NO PRESENT. So we must add ata to dictionary.

In other words. _ = a

ratatatat✗a✗rat✗at✗a✗rat

DECOMPRESS

1, 1, 2, 6, 1, 3, 7, 9, 11, 4

2	b
3	c
4	d
5	ad
6	ab
7	ba
8	aba
9	ac
10	cb
11	baa
12	acb
13	baad

a_i s_i s_{i+1} ... s_n

a_{i+1} s_n ... always

same

1, 1, 2, 6, 1, 3, 7, 9, 11, 4

a a b ab a c ba ac baa d

ratatatatatbarabatbabrat

Encoded String:

Index	Entry
1	a
2	b
3	r
4	t
5	ra
6	at
7	ta
8	ata
9	atat
10	tb
11	ba
12	ab
13	br
14	rat
15	tba
16	atb
17	bab
18	bra

3, 1, 4, 6, 8, 4, 2, 1, 2
5, 10, 6, 11, 13, 6

≈ same as before

COMPRESS "aabababcbaacbaad"

Index	Entry
1	a
2	b
3	c
4	d
5	aa
6	ab
7	ba
8	aba
9	ac
10	cb
11	baa
12	acb
13	baad

Encoded String:
1, 1, 2, 6, 1, 3, 7, 9, 11, 4

3	b
4	o
5	w
6	wa
7	ab
8	bb
9	ba
10	a#
11	#w
12	wab
13	bba
14	a#w
15	wabb
18	abb
19	ba#w
20	wo
21	oo
22	o#
23	#wo
24	oo#
25	woo

DECOMPRESSED STRING

wabbba#wabba#wabb

of many pixel ... pixel
The no. of pixel determine quality of image known as resolution. Higher resolution always yield better quality.

A bit map representation stores the graphics image data in the same manner that the computer monitor stores in video memory.

BIT-MAP IMAGES:
→ Each pixel is stored as single bit 0/1
→ 640 x 480 bitmap requires 37.5 KB of storage
→ Dithering is ~~offe~~ often used for displaying monochrome images.

GRAY-SCALE IMAGES:
→ Each pixel is stored as a byte b/w 0-255.
→ A dark pixel has value of 10 and bright with 240.
→ A 640 x 480 grayscale image requires 300 KB of storage.

DITHERING:
→ Is often used when converting gray scale images to bit map I. For eg in case of printing. The main strategy

vijeta

such that no. of printed dots approximate the gray scale level of original image. If a pixel is replaced by 4x4 array of dots, the intensity can be approximated b/w 0 to 16

↓ No dot ↓ full dot.

The size of dithering image must be much larger since each pixel is replaced by 4x4 array of dots. So image may be 16 times larger.

24-bit colour images

→ Each pixel is represented by 3 byte
→ 256x256x256 possible colors which is 16 million.
→ 640x480 image required 921.6KB.
→ Some color images are 32 bit image. Extra byte is used to store an alpha value, representing a special effect.

8-bit colour image

→ 1 byte for each pixel
→ Supports 256 colours
→ Acceptable colour quality
→ Require colour ~~to~~ lookup table

→ 640×480 image require 307KB
→ eg Grayscale.

COLOUR LOOKUP TABLE:

→ Store only the index of colour LUT for each pixel.

→ Lookup table to find colour (RGB) for the index
→ LUT needs to be build when converting 24 bit image to 8 bit.
→ It requires grouping similar colour
→ possible for pattern animation by changing the color map.

Tones
- Continuous → composed of varying shades of colours.
- Half tone → Subset of colors ∴ all colors are not possible
- Bitone → 2 colors, white & black.

RGB color models/space

ways to reproduce a broad array of colours

→ RGB is device dependent color model i.e different device detect & reproduce a given RGB value differently. Since the color elements (phosphorous) and there response to RGB levels varies from manufacturer to manufacturer.

→ It is used in T.V, image scanner, video camera dependent on the technology like CRT LCD plasma etc.

→ Inside a CRT electron beams falling on red, blue and green phosphorous dots & produce corresponding color lights.

* RED AND Green → Yellow
Red and Blue → Magenta
Green and Blue → Cyan
Red + Green + Blue → White
None → Black

rgb(255,

eg newspaper magazine

* for eg Orange = 96% red, 40% green, 14% blue.

CMYK COLOUR MODEL

→ C = Cyan, M = Magenta, Y = Yellow, K = black.

→ Blue spot on paper appears blue because it absorbs other components of light and reflects only colour in our eyes

→ for RGB model, if red spot is mixed blue spot, magenta is produced. but red ink will try to absorb blue light and blue ink absorbs red light. So if we combine the two the spot looks black.

→ The CMYK is based on the fact that surface appears to be certain colour because wavelength of light absorbs and reflects light. The CMYK subtractive model take advantage of phenomenon by depositing ink on surface in order to absorb selectively absorb certain colours of light.

→ The light that isn't absorbed by the ink is reflected to observer's eyes.

DEVICE DEPENDENCY AND GAMET

RGB or CMYK don't have universal or absolute color value.

So there is no absolute color.

→ RGB and CMYK are device dependent model

→ Total range of color supported by each model is called Gamet

→ RGB has larger Gamet than CMYK model. All colours of RGB can't be expressed in CMYK model. The image displayed on screen has vibrant colour when presented on paper

DEVICE INDEPENDENT COLOUR MODEL

→ Colour information is a way the human eye receive them. It interpret colour info in terms luminance and ~~pl~~ chrominance.

→ Luminence depict brightness info of image. Chrominance component depict colour information in diff. part of image.

International Committee of Illumination

~~Retinal colour~~ Retinal colour are transmitted into distinction b/w light and dark, ~~blue~~ red and green. blue and yellow. Central axis are lightness. Values range from 0-100 (Black and white). 0 is neutral gray for both a^* and b^*

HSB MODEL
HUE SATURATION BRIGHTNESS

→ Here define

→ Value of Hue axis vary from 0-360

Begning with red and ending with red, b/w blue & green.

~~Also~~

→ saturation Indicates degree to which the hue differs from natural gray Values runs from 0% with in no color saturation and 100% to fullest saturation hue.

Brightness indicates level of illumination value runs from zero % to 100%.
(black) (white)

Image compression

Both images graphical & computer generated and digital images are displayed in form of 2-d matrix, of individual picture elements.

~~Both~~

Graphical image is represented differently in form of programme return in graphical programming lang.

Lossless compression algo must be used for transferring graphical image

To transfer digital image, compression algo is applied:

Combination of run-length encoding and statistical encoding

eg. transfer of lossless and transfer docs by fax machine

combination of transform and differential encoding.

GIF (GRAPHICAL INTERCHANGE FORMAT)

choosing 256 colours
colours. Rather than sending each
pixel in 24 bit, 8 bit index to
colour entry in table is send.
→ Table of colour can relate either
to global colour table or Positi
of image
→ contents of table are send along
with other info like screen size
and aspect ratio

→ L-ZW Coding

can be used to obtain further compression

→ Basic colour table extended dynamica
→ Occurence of common string of pixel
such as long strings of same colour
are detected and entered in colour
table after 256 colours
→ To represent each string of pixel
values, the corresponding string of
8 bit indices to basic color table

build up in a progressive way as the data arise

(iii) compress data is divided in 4 groups. First contain 1/8 of total compressed image. Second further 1/4. Third further 1/4. last 1/2

TIFF (TEXT-IMAGE FILE FORMAT)

It support resolution upto 48 bit. 16 bit each for RGB. It is capable of transfering digital doc and images. Uses a no. of different formats represented by a code no. from un-compressed format (code no. 1) to LZW (code no. 5). Code no. 2, 3, 4 are digital document

LZW has basic colour table with 256 colours and table can be extended to 4096 entries containing common strings of pixels in image.