Image Compression

* Stored in form of pixels
* Pixels are arranged in form of 2d-array

Amplitude (value stored in each picture)

* No. of pixels ∝ Resolution

Eg. 512 x 512 — Higher resolution compared to 8x8 image

* Better resolution → Occupies more space

Storage & Transmission is diff

∴ We need to compress.

Types of Images

① Bilevel / Monochrome Image

All Pixel values : 1 bit only 2^1 different values either 0 or 1

② Grayscale Image

(Based on no. of bits for each pixel. no. of different values is decided, improves quality of image)

③ Continuous Tone

No abrupt frequency (natural image)

④ Discrete Tone (sharp edges)

⑤ Cartoon like (purely man-made)

Grayscale :

Bit plane : No. of bits for each pixel

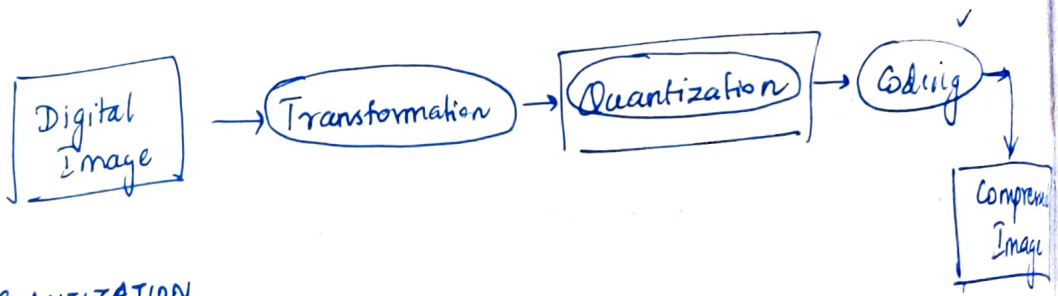
Eg 8 bits, Each bit plane is of size 8

If we remove few LSB, we can perform compression.

Same compression technique cannot be used for all types of images.

Sampling : Technique to convert continuous to dicrete data.

Loss of data is there when continuous is converted to digital image.



QUANTIZATION

Scalar Quantization

simply remove certain bitplanes.

Not optimized

8 bit image

203 - 214

214

203 :

2	203	
2	101	1
2	50	1
2	25	0
2	12	1
2	6	0
2	3	0
	1	1

11001011

(Remove LSB 4 bits)

One row 9 pixels

203	11001011	11000000
204	11001100	11000000
205	11001101	11000000
206	11001110	11000000
207	11001111	11000000
208	11010000	11010000
209	11010001	11010000
210	11010010	11010000
211	11010011	11010000
212	11010100	11010000
213	11010101	11010000
214	11010110	11010000

approximated to same value

approximated to same value

We obtain sharp boundary. ~~Not~~ Decompressed image will not like a natural image

∴ This is not an optimized method.

Grayscale Quantization

* SNo	* Value	* rsm (random no. based on neighbour~~ing~~ pixels)
1	11010101	0000 0000

* Compressed Stream

SNo	Value	rsm	Compressed Stream	
1	1101 0101	0000 0000	1101	(1st step alone left as it is)
2	1001 0111	1001 0111	1001	(Otherwise: First 4 bits of Rsm)
3	1010 1010	1011 0001	1011	

In case value is of the form

1111 xxxx :- (Rsm updated with value of pixel itself (no addition with prev Rsm)

Compressed stream:

1111

No Banding Effect

Because neighbourhood pixel values incorporated for next pixel.

Better than scalar quantization

Principles of Image Compression

* Neighborhood pixels carry similar values (almost equal)

Difference Matrix

∴ Instead of coding each pixel value as it is we can code diff b/w consecutive pixels

Eg:

	212	215	216	216	217	219	220	218
Code:	212	3	1	0	1	2	1	-2

→ Lossless Compression

Difference coding Δ = Avg - Pixel

More correlation ⇓ redundant ⇓ less # bits

Difference Matrix

* Range has become less
* Correlation has been removed
* If there are more # zeroes or ones, we can

code it efficiently

Bilevel Image — Run length Encoding

Scan through entire image ( or )

Count # consecutive zeroes & ones

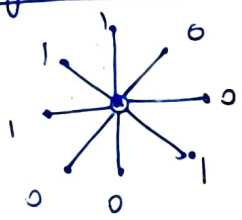
store it with freq

Eg: 0 0 0 0 0 0 . . . 0 1 1 1 . . . 1

Store it as

0 10 1 9 (0 - 10 times
1 - 9 times)

Bilevel Image Context



(Based on neighbourhood context is created for a pixel)

Represented as a n-bit number

This bit-sequence might appear no. of times.

Calc freq. prob. Do arithmetic coding

~~* context based encoding can be appl... each bit plane...~~

Transform based Compression

Spatial domain data of image (2-D space) ⟶ Frequency domain data

(Can identify uniform (low freq) and non-uniform (high freq) areas of an image

~~* This removes inter-pixel redundancy~~

Transform is not compression

It just represents image in a different format

Only when we apply quantization, we achieve compression.

Continuous Tone Image

Natural Images: No sharp edges

Color Image – 3 channels → R, G, B

Divide each pixel into three components

Apply the compression technique for each component separately

Eg:	R	G	B
P_1:	1000	0011	0000
P_2:	1001	0011	0000

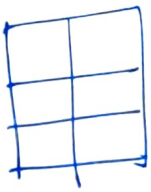
Both pixels almost identical

Overall value significantly diff

But R G B separately are very similar

Fractal Image Compression

Divide Image into blocks (regions)



Code a block.

If other blocks are similar, no need to code them separately. A pointer to the coded block is stored.

JPEG

- Compression Technique for images & videos
- Cannot be used for bilevel image
- Can be used for grayscale images & colour images.

Merits:

* It has parameters to decide quality of decompressed image (Block Size, Quantization Bit)
* It can be used as either a lossy or lossless technique.
* Better Quality
* JPEG works efficiently in all image features like size, colorspace, aspect ratio.

Evaluated from

* Good to Excellent
* Progressive Transmission

Prominence : Hue - Dominant wavelength of spectrum

Saturation - Amt of white light

Luminance : Brightness

Downsampling :

Subtract 128 from all pixels

Good Dates

Char	Frequency	Prob	Interval
G			
o			
d			
a			
t			
e			
s			

color info

Y Cr Cb

Luminance: Gives the picture (grayscale image)

For each pixel:

$$Y = \left(\frac{77}{256}\right) R + \left(\frac{150}{256}\right) G + \left(\frac{29}{256}\right) B$$

$$C_b = \left(\frac{44}{256}\right) R - \left(\frac{87}{256}\right) G + \left(\frac{131}{256}\right) B + 128$$

$$C_r = \left(\frac{131}{256}\right) R - \left(\frac{110}{256}\right) G - \left(\frac{21}{256}\right) B + 128$$

To convert Y Cr Cb to RGB:

$$R : Y + 1.371 (C_r - 128)$$

$$G : Y - 0.698 (C_r - 128) - 0.366 (C_b - 128)$$

$$B : Y + 1.732 (C_b - 128)$$

Chromaticity Diagram
(3D Diagram)

DCT
(Discrete Cosine Transform)

2-dimensional DCT

Blockwise only DCT is applied because

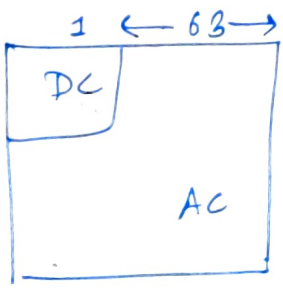
* Only neighbouring pixels are correlated
* High no.of operations in DCT & IDCT if applied for entire image.

$$DCT(x,y) = \frac{1}{4}(C_i C_j)\sum_{x=0}^{7}\sum_{y=0}^{7} P(x,y)\left(\cos\left(\frac{2x+1}{16}\right)i\pi\right)\cos\left(\frac{2y+1}{16}j\pi\right)$$

$C_i, C_j \Rightarrow [0,7]$

$$C_f = \begin{cases} 1/\sqrt{2}, & f=0 \\ 1, & f>0 \end{cases}$$

Most values accumulated in upper-left corner. Less value in bottom right corner



Quantization

Divide image after DCT into 8×8 blocks and apply quantization using basic table (8×8)

Upper left - more info ⇒ less quantization

Lower right - less info ⇒ More quantization

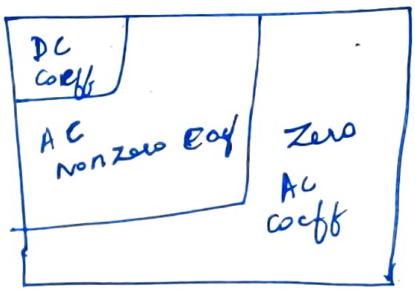
JPEG Parameters : 8×8 quantization table

To create own quantization table:

(i) $1 + (i+j) * R$

(ii) Simply divide by 2, 4, 6

Entropy coding



Different entropy coding for 3 components

Zig AC Coefficients

→ Highly correlated

→ Mostly 0s

→ We can use RLE < 000 1 2 2 2 2 3 3 3 5
3, 0, 1, 1, 5, 2, 3, 3, 1, 5
(skip, value) pairs

Skip #x values and 1st occurrence of a non-zero value

Non-zero Ac coefficients

Huffman Coding Technique

DC Coefficients

DPCM : Differential Pulse code Modulation (Lossless Technique)

$$DP(i) = DC(i) - DC(i-1)$$

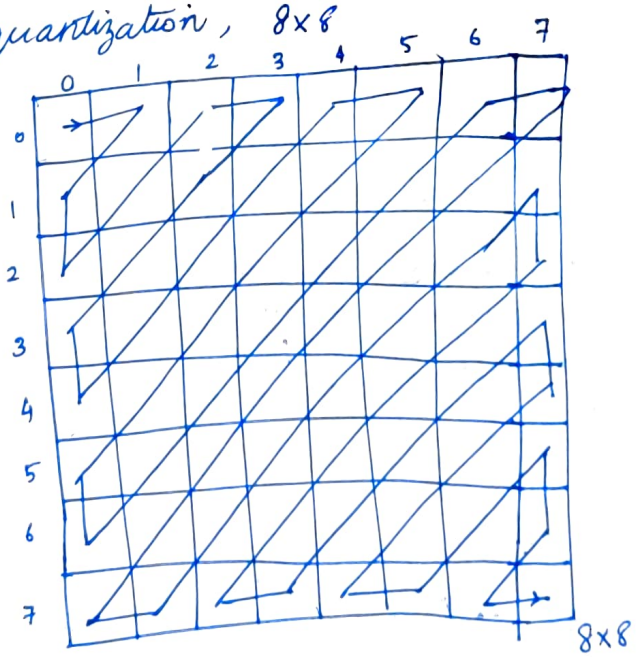
1118 , 1114 , 1112 , ...

↳ 1118 , -4 , -2 , ...

After DPCM, perform Huffman Coding

Specialised table only of DC coefficients is created.

After quantization, 8×8



If we read ⇉ mixage of zero & non-zero values

∴ We use zig-zag scan ↯

Hence, we convert 8x8 block into 1-D data with 64 comp.

Co-ordinates

(0,0) ~~(0,1)~~

(0,1) (1,0)

(2,0), (1,1), (0,2)

~~(0,3)~~, (0,3), (1,2), (2,1), (3,0)

After Huffman, we obtain output stream

To decode, we need to send few tables

① Huffman table and

② DCBM Huffman Table :

Size	Value	Code
0	0	0
1	-1, 1	~~0~~ ~~1~~ 10
2	-3, -2, 2, 3	~~00~~ ~~01~~ ~~10~~ ~~11~~ 110
3	-7, -6, -5, -4, 4, 5, 6, 7	1110

$\{-(2^i-1)$ to $+(2^i-1)\}$ — $\{-(2^{i-1}-1)$ to $(2^{i-1}-1)\}$

16 -65536 ..... 65536 ..... 1111111....10

15 15

Row	value	code
i	$-(2^i-1)$ to (2^i-1)	(i 1's) 0

Represented as (R, C)

Eg: 1 1118

R = 11 Code

C = [redacted] 1111111111 0

Eg 2 −2

R = +2

C = 2

Code : (110, 10)

From table

(R, C)

Eg: 3 −1

R : 1 (10, 1)

C : 1

Code value from table

Code C in binary R bit

Eg 4 −7

(1110, 001)

R = 3

C = 1

AC Components : Low value, close to zero

JPEG has separate separate table for luminance & chrominance components.

(R, Z) — No. of Zero coefficients appearing before it

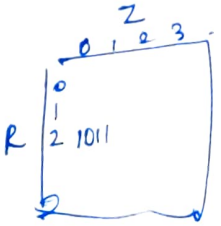
1118, −2, 0, 2, 0, 0, [redacted]1

R: 2 C: 2

Z = 0

Table

(1011, 10)



Code :

(Table R, Z , C in binary R bits)

2

↓ R = 2 C = 3 Z = 1

From table

(R=2 Z=1 ↓ , 11) C in 2-bits

2 → R = 2
C = 3
Z = 3

End of block

EOB ~~Block~~ Code Coded with 1010

DC ⟶ AC ⟶ Full zero components

Before compression

Each 8x8 block

64 × 24 = 1536

8 8 8
Y Cr Cb

Ratio = 1536 / 138 = 11

With JPEG

DC Component : 23 bits 46 × 3 = 138

AC Components 23

Video Compression

→ Spatial Redundancy
- Coding
- Interpixel
- Psychovisual (lossy removal, no difference to eye)

→ Temporal Redundancy

Remove Coding Redundancy : Any encoding technique (Huffman, Arithmetic, RLE)

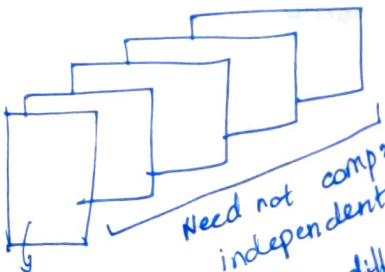
Interpixel : DCT, Wavelet Transformation etc.

Psychovisual : Quantization

Video : Comprises of Frames

Neighbourhood frames are very similar (almost same)

* First frame will be compressed independently



Interframe : Which depends on prev. frame for encoding & decoding. Also called as Predictive frame

Intraframe : Coded / Decoded independently

In case of error in k^{th} frame it will be carried in $k+1, k+2, \ldots$ frames

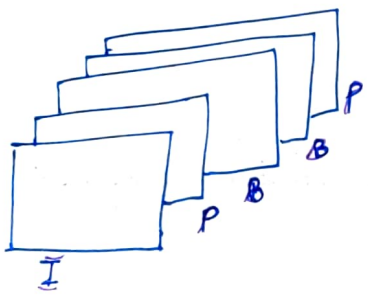
We need B - frame (Bidirectional frame)

depends on prev & next frame .

I frame → Coded independently

P frame → Coded by privious I / P frame

B frame → Coded with previous and successive P frame or I frame .



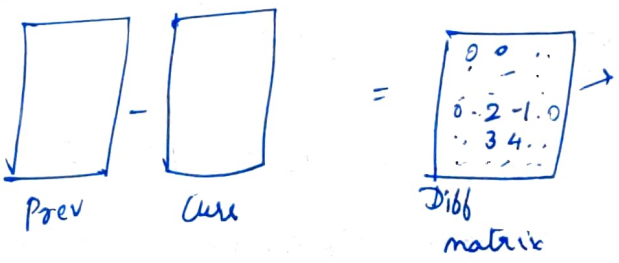
Techniques :

① Subsampling

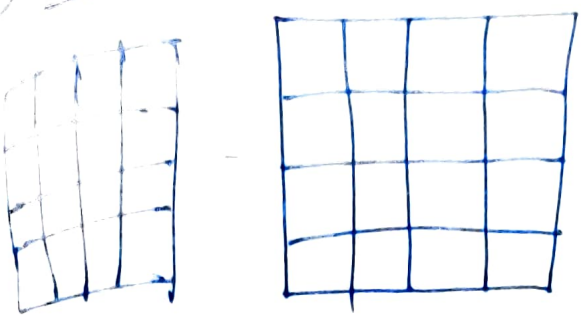
* Encode alternate frames only

* Decoding : do duplication : 1, 1, 3, 3, 5, 5, ...

② Differencing



3) Block differencing — Divide frame into blocks
Find diff b/o blocks



4) Motion Compression

$$(C_x - b_x, C_y - b_y) = \Delta x, \Delta y$$

C_x: x-coord in prev frame; b_x: x-coord in current frame; C_y: y in prev; b_y: y in curr frame

$\Delta x, \Delta y$: Motion vector

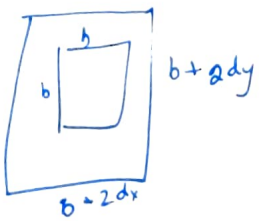
* Segregate image into blocks
* Check if block is present in prev image

Block size should be selected optimally
If block size is very large, entire block should be searched. To find absolute match for large block is hard

Threshold value

Limit to which diff can be accepted

Block search Procedure



$(2dx+1)\cdot(2dy+1)$ # blocks

Each block : $b \times b$ block size

Distortion measure

$$\text{Mean absolute pixel difference} = \frac{1}{b^2} \sum_{i=1}^{b} \sum_{j=1}^{b} [c_{ij} - b_{ij}]$$

c_{ij} → pixel in block in prev frame

b_{ij} → pixel in block in current frame

First frame + Motion vector ⇒ Second frame

↳ Diff b/w x & y coord of upper left corner of matching frame in current frame & previous frame

Motion compensation

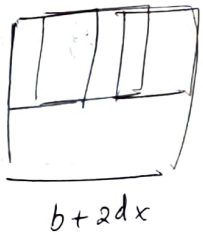
H261 Standard

- Video Compression Technique
- Motion compensation to create predicted image
- Diff b/w predicted & original frame
- DCT
- Quantization
- Entropy Coding

Search for

block in in prev. frame

search area



Overlapping search

Each block in search area, matching criterion is seen.

Create Motion vector.

comparisons : $(2dx+1)(2dy+1)$

If $dx = dy = 20$,

No. of comparisons = 1681

for one 8×8 block

41
41
41
164
1681

Solutions

- Increase block size

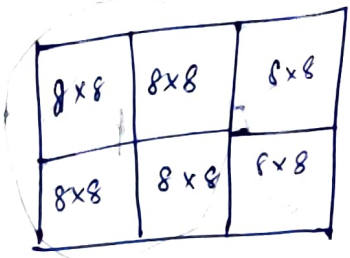
But block may comprise of >1 object

Macro-block

512 x 512 Image
- 512 x 512 Luminance (Y) block — Outline
- 512 x 512 Cr block — colour
- 512 x 512 Cb block — colour

More important : Luminance component

Macro Block (4 - Luminance
2 - Cr, Cb blocks)



* Image is divided into Macro-blocks
* Motion-Compension done w.r.t luminance component only

Block to be encoded — (x_c, y_c)

Block used for Prediction — (x_p, y_p)

Diff should be < 15

* Filtering Operation (Preprocessing) — Before motion compensation

To remove noise (sharp edges)

Do filtering

for each pixel

| 1 | 2 | 3 |

New val (2) = $\frac{1}{4}(1) + \frac{1}{2}(2) + \frac{1}{4}(3)$

* Row-wise filtering separate
* Column-wise filtering separate

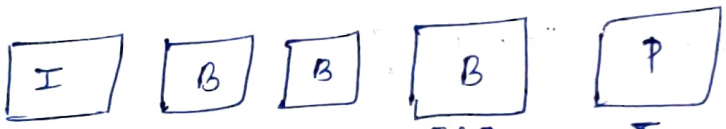
If no of I frames is more, time will be saved
Delay reduced.

Store and retrieve - H261 - Caused delay.

Moving Picture Expert Group (MPEG-I)

Compression

I - Independent frame / Intraframe — Less
P - Predictive Coded frame — More
B - Bidirectional Predictive Coded frame — High



Coded based on : - I & P I & P I & P I

B frames tolerate error. B frames not used for prediction of other frames. ∴ Error doesn't propagate.

I frame cannot have error

P frame can have error. This gets propagated

Display Order

I	B	B	P	B	B	P	B	B	P	B	B	I
1	2	3	4	5	6	7	8	9	10	11	12	13

Processing order / Bitstream Order

1, 4, 2, 3, 7, 5, 6, 10, 8, 9, 13, 11, 12

Frames are sent to decoder in the bitstream order

Optimal size of frame : 768 X 576

Macro blocks : 396 — 25 frames or less / sec
330 — 30 frames or less / sec

MPEG :
Good for Slow / Medium moving videos
Fast moving images. doesn't work well

STATISTICAL MODELLING

* **Zero - Order Model:**

  (1) The accurate probability table

  (2) Deviation of the data from uniform distribution

These 2 factor are imp. for performing compression.

* **Finite Context Modelling**

Instead of considering individual char & their prob, find the prob of each char based on its context.

↳ Order 0 : That symbol alone (1 Table)

↳ Order 1 : Symbol & its prev char

Suppose, Total = 256 char (For each char, 256 possibilities for prev char)

∴ 256 Tables

↳ Order 2 : ___ Each char based on prev 2 char

Order 0 :

— 8 bit → 2 bit

Order 1

— — 8+8 → 2 bit

Order 2

— — — 8+8+8 → 2 bits

↓ Compression increases

Problem:

All tables should be sent to decoder

Solution:

Adaptive Modelling

No need to send static prob table to decoder

Disadv: Need of scanning data before encoding / decoding

Less info abt data

Dictionary Based Compression Techniques

Dictionary
- Static (* No deletions, * Some additions)
- Dynamic

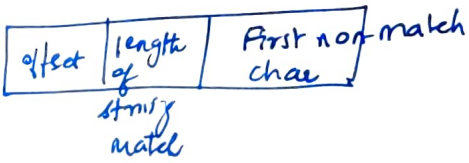
Static Dictionary

* 2^{19} records => 19 bits required to represent index
* Restricted to certain words
  Not customizable for demand

General purpose static dictionary: Not used much
Special purpose static dictionaries created to fulfill demands of specific applications.
Eg. Medical field, programming, etc. → Consumes more space

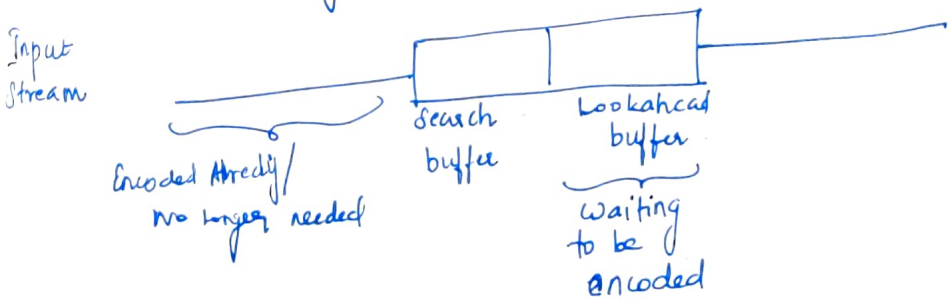
∴ We use Dynamic Dictionary

Adaptive Dictionary

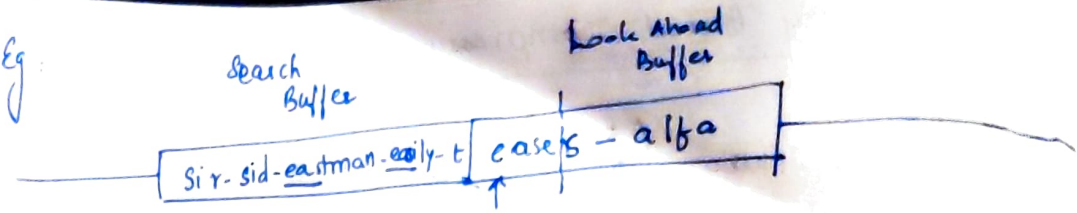


~~LZ~~ LZ77 - Algorithm / Sliding Window Algorithm

* Uses sliding window:



Eg



Check if e is present in search buffer

offset	Length of match	First non match
16	3	e

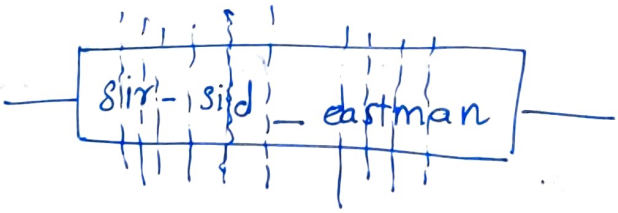
Move sliding window

28	1	_

Start of i/p stream

Token will be

		1st letter
0	0	(s)



0	0	s
0	0	i
0	0	r
0	0	i
4	2	d
4	1	e
0	0	a
10	1	t
0	0	m
4	1	n

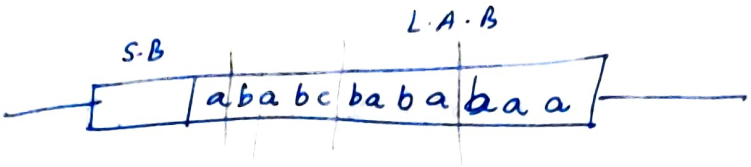
offset → $\log_2 S$

String match length → $\log_2 (L-1)$

Next char to encode → 8 bit

Total each token ≈ 24 bits

Ex: I/p string : ababcbababaa



Compressed data :

[0, 0, 'a']
[0, 0, 'b']
[2, 2, 'c']
[4, 3, 'a']
[8, 2, 'a']

→ Decoder

Decoder

ab ab c bababaa

Eg:

[3, 4, '_']

Eg:

Coding

[0, 0, 'M']
[0, 0, 'I']
[0, 1, 'S']
[3, 3, 'P']
[1, 1, 'I']

Decoding

MISSISSIPPI

LZSS Algorithm

* Variant of LZ77 algorithm

* LZ77 :- DS Used was Circular Queue

* To optimize search procedure, LZSS uses BST

* Search Buffer : 16 bytes

* Token only for matching sequence : Token (offset, length)

* Flag Bit also stored in token

Uncompressed : 2 bytes + 2 bits

Compressed 2 bytes + 1 bit (Flag . offset . Length)
1 8 8

Eg : MISS_MISSISSIPPI

MISS_MISSISSIPPI

[0, 'M']
[0, 'I']
~~[0, 0, 1]~~
[0, '_']
[1, 5, 4]
[1, 8, 3]
[1, 11, 1]
[0, 0, 1]

[0, 'M']
[0, 'I']
[0, 'S']
[1, 1, 1]
[0, '_']
[1, 5, 4]
[1, 8, 3]
[1, 11, 1]
[0, 'P']
[1, 1, 1]

[1, 4, 1]

LZ78 Algorithm

Adaptive ~~Algorithm~~, Dictionary based alg.

3 types of tokens

Unmatched char : (0, unmatched char)

"Good Morning"

Dictionary

index	string
0	NULL
1	G
2	O
3	od
4	_
5	M
6	or
7	n
8	i
9	ng

O/P string

(0, 'G')

(0, 'O') (matches → read next char also)

(2, 'd')

O's index

(0, '_')

(0, 'M')

(2, 'r')

(0, 'n')

(0, 'i')

(7, 'g')

bits for index depends on max value required for index

bits = $\log_2 M$ M - max index value

Stream of bits sent to decoder

Decoder should construct dictionary

Decoding

Index	Char	o/p string
0	NULL	
1	G	G
2	O	GO
3	OD	GOOD
4	_	GOOD_
5	M	GOOD_M
6	Or	GOOD_MOR
7	N	GOOD_MORN
8	I	GOOD_MORNI
9	NG	GOOD_MORNING

No comp,

each char : 8 bits

After

$$\left[(\log_2 9) \times \#\text{entries} \right] + \left[8 \times \#\text{char} \right]$$

Eg aababacbacacabac

Compression

Index	Char	o/p stream
0	NULL	
1	a	(0, 'a')
2	ab	(1, 'b')
3	aba	(2, 'a')
4	c	(0, 'c')
5	b	(0, 'b')
6	ac	(1, 'c')
7	aca	(6, 'a')
~~8~~	~~ca~~	~~(4, a)~~
8	ba	(5, 'a')
		(4)

Decoding

Index	Char	o/p string
0	NULL	
1	a	a
2	ab	aab
3	aba	aababa
4	c	aababac
5	b	aababacb
6	ac	aababacbac
7	aca	aababacbacaca
8	ba	aababacbacacaba
		aababacbacacabac

Before compression $16 \times 8 = 128$ bits

After compression

$(\log_2 8 \times 9) + (9 \times 8) = 27 + 72 = 99$ bits

9×12

4 6;c +8

12

Each token

$(8 + \lceil \log_2 9 \rceil) = 12$

$(12 \times 8) + (4) = 96 + 4 = 100$ bits

LZW - Algorithm

Initially, dictionary is not empty

It has $2^8 = 256$ entries (for each char of alphabet)

ASCII Table

a b a b a c

a b ab a c

0 1 256 0 2

256 ab

257 ba

258 aba

259 ac

Decoding

0 1 256 0 2

a b ab a c

b 0
a 1

Dict

0 a
1 b
256 ba
257 ab
258 baa
259 aba
260 aa
a

b a b a b a a a

1 256 257 0 260

Decoding

b a ba ab a a a

ab abb abbb abbbb

1 0 256 257 0 260

b a ba ab a a

b a ba aba aa