Unit-3

Coding a sequence, generating binary code, comparision binary & huffman, Applications: Bilevel image compression - JBIG, JBIG2, Image compression. Dictionary techniques: Intro, Static Dictionary: Diagram coding, Adaptive dictionary, LZ77, LZ78, Applications: file compressions - UNIX compress, Image compression: GIF, compression over modems: V.42 bits, Predictive coding: ppm: basic algo, ESCAPE SYMBOL, length of context, Exclusion principle, Burrows-Wheeler Transform; Move to front coding, CALIC, JPEG-LS, Multi-resolution Approach, Facsimile encoding, dynamic markov compression

Arithmetic coding is method of generating variable-length codes. It is useful when dealing with sources with small alphabets, such as binary sources, and alphabets with highly "skewed probabilities".

① **coding a sequence**:- A unique identifier is used to distinguish a sequence of symbols from another sequence of symbols called 'tag'. One possible set can be in unit interval $[0,1)$. ~~soit is~~

- In order to assign a unique tag to each sequence, we need a mapping function which will map sequence of symbols to the interval. A function that maps random variables into a unit interval is cumulative distribution function (cdf) of random variables associated with the source.

② **generating a tag**: Suppose there is a sequence $X = \{x_1, x_2, \ldots x_n\}$ then for generation of tag we have

$$l^n = l^{n-1} + (u^{n-1} - l^{n-1}) \cdot F_x(x_{n-1})$$
$$u^n = l^{n-1} + (u^{n-1} - l^{n-1}) \cdot F_x(x_n)$$

where n is length of sequence

and tag value will be

$$\bar{T}_x(n) = \frac{l^n + u^n}{2}$$

Therefore, tag for any sequence can be computed in sequential fashion. The only information required is cumulative distribution function (cdf) of source.

Q: Generate the tag for sequence {1 3 2 1} from cdf of source as follows:

$F_x(K) = 0, K \le 0$ & $F_x(1) = 0.8$, $F_x(2) = 0.82$, $F_x(3) = 1$ and $F_x(K) = 1, K > 3$.

Sol: Let $l^0 = 0$ & $u^0 = 1$

- [1] : $l^1 = l^0 + (u^0 - l^0) \cdot F_x(1-1) = 0 + 1 * 0 = 0$
  $u^1 = l^0 + (u^0 - l^0) \cdot F_x(1) = 0 + 1 * 0.8 = 0.8$
- [3] : $l^2 = l^1 + (u^1 - l^1) \cdot F_x(3-1) = 0 + (0.8) \cdot (0.82) = 0.656$
  $u^2 = l^1 + (u^1 - l^1) \cdot F_x(3) = 0 + (0.8) \cdot (1) = 0.8$
- [2] : $l^3 = l^2 + (u^2 - l^2) \cdot F_x(2-1) = 0.656 + (0.8 - 0.656) * 0.8 = 0.7712$
  $u^3 = l^2 + (u^2 - l^2) \cdot F_x(2) = 0.656 + (0.8 - 0.656) * 0.82 = 0.77408$
- [1] : $l^4 = l^3 + (u^3 - l^3) \cdot F_x(1-1) = 0.7712$
  $u^4 = l^3 + (u^3 - l^3) \cdot F_x(1) = 0.7712 + (0.77408 - 0.7712) * 0.8$
  $= 0.773504$

Tag value: $\bar{T} = \frac{l^4 + u^4}{2} = 0.772352$

- Note, each succeeding interval is contained in preceding interval.

⇒ • Main disadvantage of this method is that the intervals gets smaller and smaller and require higher precision as the sequence gets longer. To combat this problem, a rescaling strategy needs to be adopted.

(5) **Deciphering the tag** : Algorithm to decipher the tag is:

(1) Initialize $l^{(0)}=0$ and $u^{(0)}=1$.

(2) for each k find $l^k = (tag - l^{k-1}) / (u^{k-1} - l^{k-1})$

(3) Find value of x_k for which

$$F_x(x_k - 1) \le l^k \le F_x(x_k)$$

(4) Update $u^{(k)}$ and $l^{(k)}$.

(5) continue until entire sequence is decoded.

- To know the entire sequence is decoded, we have two ways
  1. knowing the length of sequence
  2. decoding to get end-of-transmission symbol.

Q: for tag value 0.772352 and previously given cdf generate 4 bit code.

Sol: $l^0=0$, $u^0=1$

$l' = 0 + (1-0) * F(x_k - 1) = 0$
$u' = 0 + (1-0) * F(x_k) = 0.8$

put $x_k = 1$

$l^2 = 0 + (0.8-0) * F(x_k - 1) = 0.656$
$u^2 = 0 + (0.8-0) * F(x_k) = 0.8$

put $x_k = 3$

$l^3 = 0.656 + (0.8 - 0.656) * F(x_k - 1) = 0.7712$
$u^3 = 0.656 + (0.8 - 0.656) * F(x_k) = 0.77408$

put $x_k = 2$

$l^4 = 0.7712 + (0.77408 - 0.7712) * F(x_k - 1) = 0.7712$
$u^4 = 0.7712 + (0.77408 - 0.7712) * F(x_k) = 0.773504$

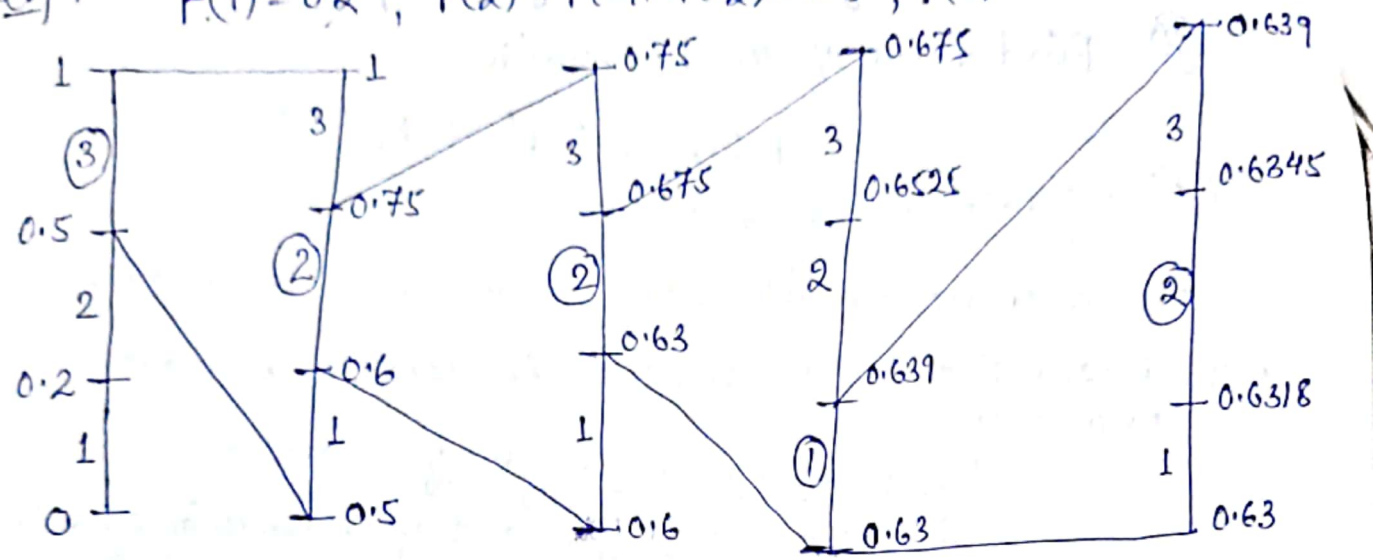
put $x_k = 1$

So, decoded 4 bit sequence

$$\{1\ 3\ 2\ 1\}$$

Q: Decode a sequence of length 5 having tag value 0.63215699 and probability model $P(a_1)=0.2$, $P(a_2)=0.3$ & $P(a_3)=0.5$

Sol: $F(1)=0.2$, $F(2)=P(a_1)+P(a_2)=0.5$, $F(3)=0.2+0.3+0.5=1$



$l^0=0$, $u^0=1$

$l^1 = l^0+(u^0-l^0)\cdot F(x-1) = 1*F(x-1)$
$u^1 = l^0+(u^0-l^0)\cdot F(x) = 1*F(x)$
} Find x such that tag $\in [l^1, u^1]$

put $\boxed{x=3}$: we get $l^1=0.5$ & $u^1=1$

$l^2 = l^1+(u^1-l^1)\cdot F(x-1) = 0.5+0.5*F(x-1)$
$u^2 = u^1+(u^1-l^1)\cdot F(x) = 0.5+0.5*F(x)$
} x such that tag $\in [l^2, u^2]$

put $\boxed{x=2}$ we get $l^2=0.6$ & $u^2=0.75$

$l^3 = l^2+(u^2-l^2)\cdot F(x-1) = 0.6+0.15*F(x-1)$
$u^3 = l^2+(u^2-l^2)\cdot F(x) = 0.6+0.15*F(x)$
} x such that tag $\in [l^3, u^3]$

put $\boxed{x=2}$, we get $l^3=0.63$ & $u^3=0.675$

$l^4 = l^3+(u^3-l^3)\cdot F(x-1) = 0.63+0.045*F(x-1)$
$u^4 = l^3+(u^3-l^3)\cdot F(x) = 0.63+0.045*F(x)$
} x such that tag $\in [l^4, u^4]$

put $\boxed{x=1}$, we get $l^4=0.63$ & $u^4=0.639$

$l^5 = 0.63+0.009*F(x-1)$
$u^5 = 0.63+0.009*F(x)$
} x such that tag $\in [l^5, u^5]$
put $\boxed{x=2}$ $l^5=0.6318$ $u^5=0.6345$

Final decoded sequence is {3 2 2 1 2}

2) **generating binary codes**: The range of values for tag at n^{th} iteration is given by $[l^n, u^n]$ such that

$$l^n = l^{n-1} + (u^{n-1} - l^{n-1}) \cdot F_x(x_{n-1})$$
$$u^n = l^{n-1} + (u^{n-1} - l^{n-1}) \cdot F_x(x_n).$$

As n gets larger, these values comes closer & closer together, i.e. in order to represent all subintervals uniquely we need increasing precision as length of sequence increases. so we perform scaling of intervals:

→ Find out the interval belongs to upper half $[0.5, 1)$ or lower half $[0, 0.5)$. Once the encoder & decoder know which half contains tag, then other half can be ignored. Then mapping or rescaling of intervals can be done by following mapping function to scale to full interval $[0, 1]$

$E_1 : [0, 0.5) \to [0, 1)$ $\quad E_1(x) = 2x$ send 0

$E_2 : [0.5, 1) \to [0, 1)$ $\quad E_2(x) = 2(x - 0.5)$ send 1

Q: Encode the sequence {1 3 2 1} from following probability model $P(a_1) = 0.8$, $P(a_2) = 0.02$, $P(a_3) = 0.18$ in form of binary sequence.

Sol: Start by $l^0 = 0$ & $u^0 = 1$ $\quad F_x(1) = 0.8$, $F_x(2) = 0.82$, $F_x(3) = 1$

for 1: $l^1 = 0 + (1-0) * 0 = 0$
$u^1 = 0 + (1-0) * (0.8) = 0.8$

$[0, 0.8)$ is not in any half. move to next.

for 3 $\quad l^2 = 0 + (0.8-0) * 0.82 = 0.656$
$u^2 = 0 + (0.8-0) * 1 = 0.8$

$[0.656, 0.8) \in$ upper half

send code 1

rescale the interval

$l^2 = 2(0.656 - 0.5) = 0.312$
$u^2 = 2(0.8 - 0.5) = 0.6$

$[0.312, 0.6)$ does not belong to any half, move to next.

for [2] : $l^3 = 0.312 + (0.6-312) * f_x(1) = 0.5424$

$u^3 = 0.312 + (0.6-312) * f_x(2) = 0.54816$

$[0.5424, 0.54816)$ belongs to upperhalf

send 1

rescaling performed

$l^3 = 2(0.5424-0.5) = 0.0848$ } lower half

$u^3 = 2(0.54816-0.5) = 0.09632$ }

send 0

rescaling

$l^3 = 2*(0.0848) = 0.1696$ } lower half

$u^3 = 2*(0.09632) = 0.19264$ }

send 0

rescaling

$l^3 = 2.*(0.1696) = 0.3392$ } lower half

$u^3 = 2*(0.19264) = 0.38528$ }

send 0

rescaling

$l^3 = 0.6784$ } upper half

$u^3 = 0.77056$ }

send 1

rescaling

$l^3 = 0.3568$ } does not belong to any half, move to next

$u^3 = 0.54112$ }

for [1] : $l^4 = 0.3568 + (0.54112-0.3568) * f_x(1-1) = 0.3568$

$u^4 = 0.3568 + (0.54112-0.3568) * f_x(1) = 0.504256$

does not belong to any half

$\Rightarrow$ stop by sending any value b/w $[0.3568, 0.504256)$ as final status of tag value.

binary encoded sequence = 110001 Ans

decoding a binary code : To decide the number of bits to decode the sequence, we take the min of difference of interval probabilities.

$2^{-K} < \text{Difference}$ K = no. of bits to start with.

- If interval belongs to any half, rescaling is applied & shift by 1 bit to generate new tag & repeat

Q: Decode the sequence

"1100011 000 .... 00"

$P(a_1) = 0.8$, $P(a_2) = 0.02$ & $P(a_3) = 0.18$.

Sol: min difference interval : $[0.8, 0.82] = 0.02$

$2^{-K} < 0.02 \Rightarrow \boxed{K=6}$

$f(1) = 0.8$
$f(2) = 0.82$
$f(3) = 1$

$l^0 = 0$, $u^0 = 1$: first 6 bits:

$\Rightarrow$ '110001' $\Rightarrow$ tag $= \frac{1}{2} + \frac{1}{4} + \frac{1}{64} = 0.765625$

first

$l' = 0 + (1-0) \cdot f_x(x-1)$
$u' = 0 + (1-0) \cdot f_x(x)$

$\boxed{x=1}$ we get $l' = 0$, $u' = 0.8$ tag $\in [0, 0.8]$ & does not lies in any half

second

$l^2 = 0 + (0.8)(0.82) = 0.656$
$u^2 = 0 + (0.8)(1) = 0.8$

$\boxed{x=3}$, tag $\in [0.656, 0.8)$ upper half

rescaling: $l^2 = 0.312$, $u^2 = 0.6$ | shift by 1 bit

Next 6 bit $\Rightarrow$ 100011 $\Rightarrow$ new tag value = 0.546875

third

$l^3 = 0.312 + (0.6 - 0.312) * f(1) = 0.5424$
$u^3 = 0.312 + (0.6 - 0.312) * f(2) = 0.54816$

$\boxed{x=2}$ tag $\in [l^3, u^3]$

$[l^3, u^3] \in$ upper half ; shift by 1 bit & scaling

$[l^3, u^3] = (0.0848, 0.09632)$

new tag: .000110 $= \frac{1}{16} + \frac{1}{32} = 0.09375$ repeat

$\boxed{\text{decoded seq: } 1321}$

Comparison of Huffman and Arithmetic coding

Method	Arithmetic	Huffman
Compression Ratio	Very good	Poor
Compression Speed	Slow	Fast
Decompression speed	Slow	Fast
Memory space	Very low	Low
Compressed pattern matching	No	Yes
Permits random Access	No	Yes
	$H(X) \le l_A \le H(X) + \frac{2}{m}$	$H(X) \le l_H \le H(X) + \frac{1}{m}$

Dictionary techniques :→ Static dictionary :- digram coding
↳ Adaptive/dynamic dictionary :- LZ family

Static dictionary : technique is most appropriate when considerable prior knowledge about the source is available. Static scheme would work well only for applications and data they are designed for.

- One of static dictionary technique is digram coding, that is less specific to single application.

digram Coding : In this form of static dictionary technique, the dictionary consists of all letters of source alphabet followed by as many pairs of letters, called digrams; as can be accommodated by dictionary.

- The diagram encoder reads a ~~two~~ two character input and searches the dictionary consists of all letters of source alphabet ~~followed by as many~~ to see if input exists in dictionary. If it does, the corresponding index is encoded & ~~transmitted~~. If doesnot, first character is encoded then second character becomes first character of next digram

Q: 5-letter alphabet = {a, b, c, d, r} and knowledge of course as

code	Entry	code	Entry
000	a	100	r
001	b	101	ab
010	c	110	ac
011	d	111	ad

Encode the seq: a b r a c a d a b r a

Code ⇒ 101 100 110 111 101 100 000

Adaptive Dictionary:

Static dictionary	VS	Adaptive Dictionary
• Most appropriate when we have the prior information of the source data.		• appropriate when we have no prior knowledge about the source data.
• Static in nature i.e. once created then new elements cannot be added later.		• New elements can be added after creation of dictionary.
• less efficient and more useful for specific applications.		• More efficient and broader application usability.
example: digram coding		example: LZ77, LZ78, LZW etc.

LZ family of adaptive dictionary technique:

Abraham Lempel — Jacob Ziv

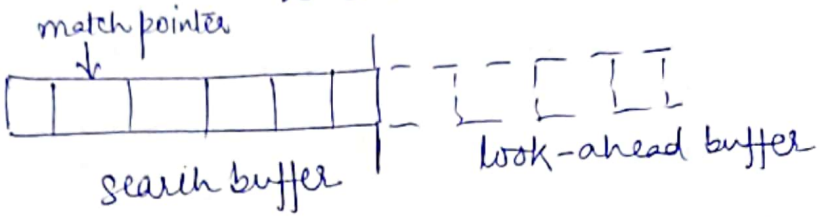
1977 → LZ77 / LZ1

1978 → LZ78 / LZ2.

LZ77 Approach: The window consists of two parts,

(1) a search buffer: contains a portion of recently encoded sequence.

(2) a look-ahead buffer: contains next portion of sequence to be encoded.

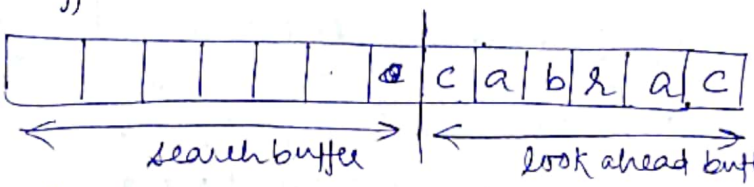


For Encoding a sequence in look ahead buffer, the encoder moves the search pointer back through the search buffer until it encounters a first match. The encoder examines the next symbol for match and proceeds in similar way until a mismatch found.

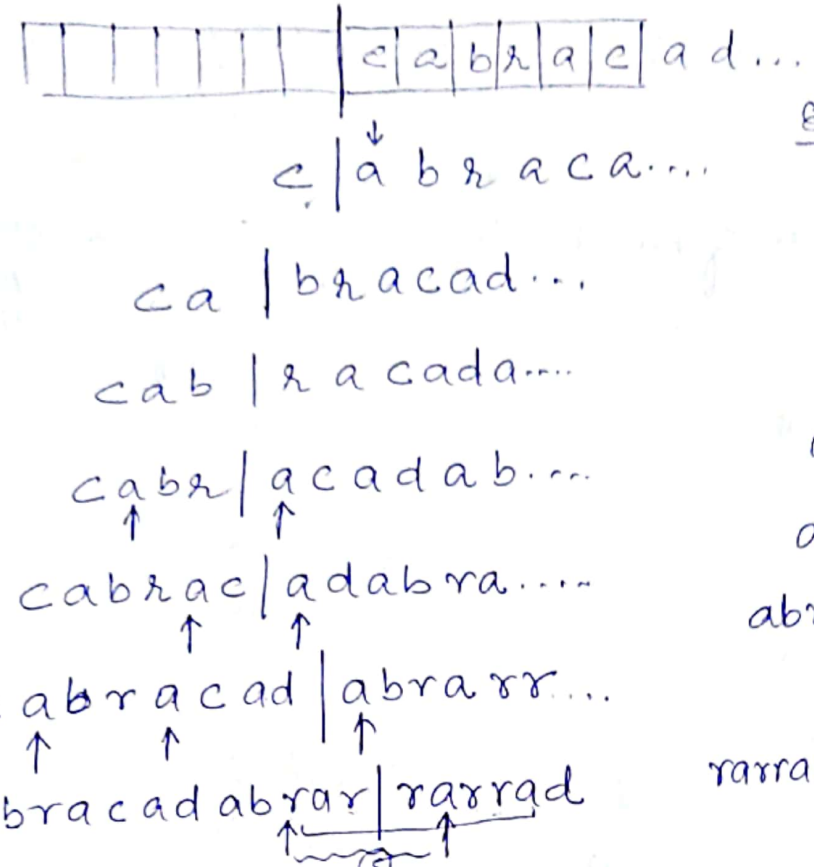
- The distance of pointer (for first match) from look-ahead buffer is called offset. and starting from first match to last match, the sequence length is called length of match.
- encoder encodes it as $<O, l, C>$ where
  O is offset, l: length of match
  C: codeword corresponding to symbol that follows the match in look-ahead buffer.

Q: Encode sequence 'cabracadabrarrarrad' length of window is 13 and size of look-ahead buffer is 6.

Sol:



Assuming search buffer being empty initially

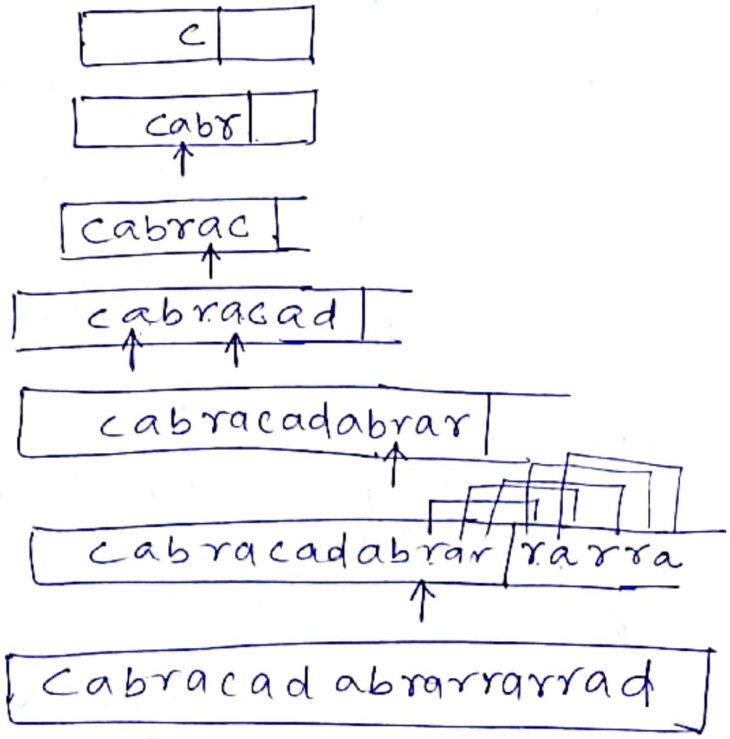


Entry	code
c	<0, 0, c(c)>
a	<0, 0, c(a)>
b	<0, 0, c(b)>
r	<0, 0, c(r)>
ac	<3, 1, c(c)>
ad	<2, 1, c(d)>
abrar	<7, 4, c(r)>
rarrad	<3, 5, c(d)>

Decoding in LZ77 | LZ1

window size : 13
look Ahead buffer : 6

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



LZ78 Approach | LZ2 : doublet used <i, c>
index

- No need of search buffers.
- Index value of newest entry is '0'. → start from next symbol till the symbol it is encoded.

Q: Encode the following sequence:

wabbbawbawbbbawaabwwbabb

Index	Entry	code
1	w	<0, C(w)>
2	a	<0, C(a)>
3	b	<0, C(b)>
4	b	<0, C(b)>
5	ba	<3, C(a)>
6	wb	<1, C(b)>
7	aw	<2, C(w)>
8	bb	<4, C(b)>
9	awa	<7, C(a)>
10	ab	<2, C(b)>
11	ww	<1, C(w)>
12	bbab	<5, C(b)>

Q: Encode using LZ78/LZ2.

ABCDABCABCDAABCBABCE

Index	Entry	code
1	A	<0, C(A)>
2	B	<0, C(B)>
3	C	<0, C(C)>
4	D	<0, C(D)>
5	AB	<1, C(B)>
6	CA	<3, C(A)>
7	BC	<2, C(C)>
8	DA	<4, C(A)>
9	ABC	<5, C(C)>
10	ABCE	<9, C(E)>

decoding in LZ78 decode the following

<0, c(w)>, <0, c(a)>, <0, c(b)>, <3, c(a)>, <0, c(␣)>,
<1, c(a)>, <3, c(b)>, <2, c(␣)>, <6, c(b)>, <4, c(␣)>,
<9, c(b)>, <8, c(w)>, <0, c(0)>, <13, c(␣)>, <1, c(0)>,
<14, c(w)>, <13, c(0)>

Sol:

index	code	entry
1	0, c(w)	w
2	0, c(a)	a
3	0, c(b)	b
4	3, c(a)	ba
5	0, c(␣)	␣
6.	1, c(a)	wa
7	3, c(b)	bb
8.	2, c(␣)	a␣
9.	6, c(b)	wab
10.	4, c(␣)	ba␣
11.	9, c(b)	wabb
12.	8, c(w)	a␣w
13.	0, c(0)	0
14.	13, c(␣)	0␣
15.	1, c(a)	w0
16.	14, c(w)	0␣w
17.	13, c(0)	00

Finally decoded sequence be

w a b ba ␣ w a bb a␣ wabbba␣ wabba␣w
0 0␣ w0 0␣w 00.

✓

LZW approach

Q: Encode the given sequence, assuming initial dictionary given as

Index	Entry
1	ß
2	a
3	b
4	o
5	w

wabbaßwabbaßwabbaßwabbaßwooßwooßwoo

Index	Entry
1	ß
2	a
3	b
4	o
5	w
6	wa
7	ab
8	bb
9	ba
10	aß
11	ßw
12	wab
13	bba
14	aßw
15	wabb
16	baß
17	ßwa
18	abb
19	baßw
20	wo
21	oo
22	~~ßwoo~~ oß
23	~~oo~~ ßwo
24	~~oß~~ ooß
25	ßwoo

- Start from the same letter, where previous pattern ends.

~~Codes to be send~~

Codes to send

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

→ Start from same symbol, at which last code ended.

p*a: ⇒ again start from 'a'.

Q: **LZW decoding :**

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

Initial dictn:

1 — a
2 — ␣
3 — r
4 — t

Index	Entry
1	a
2	␣
3	r
4	t
5	ra
6	at
7	ta
8	ata
9	atat
10	t␣
11	␣a
12	␣a␣
13	␣r ← only need till 13

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

r a t at

ata t ␣ a ␣ ra

so, final answer

. rat at at at ␣ a ␣ rat ␣ at ␣ a ␣ rat.

	LZ77	LZ78	LZW
Output	<0, l, c> (offset, length, code)	<i, c> (index, code)	<i> (index)
Space	more space	less than LZ77	Least space
Efficient	less efficient	more than LZ77	Most efficient
Matching limit	can't find pattern size of greater window size	find all possible pattern	find pattern of variable length
Decoding Speed	easy slow	easy moderate	complex fast
complexity	more	moderate	less

Applications

- Best known applications of LZW: File compression, Graphics Interchange Format (GIF) & V.42.

GIF : It is implementation of LZW algorithm.

PNG : Portable Network Graphics ≈ LZ77

next pg

Q: Encode the following sequence using LZ77

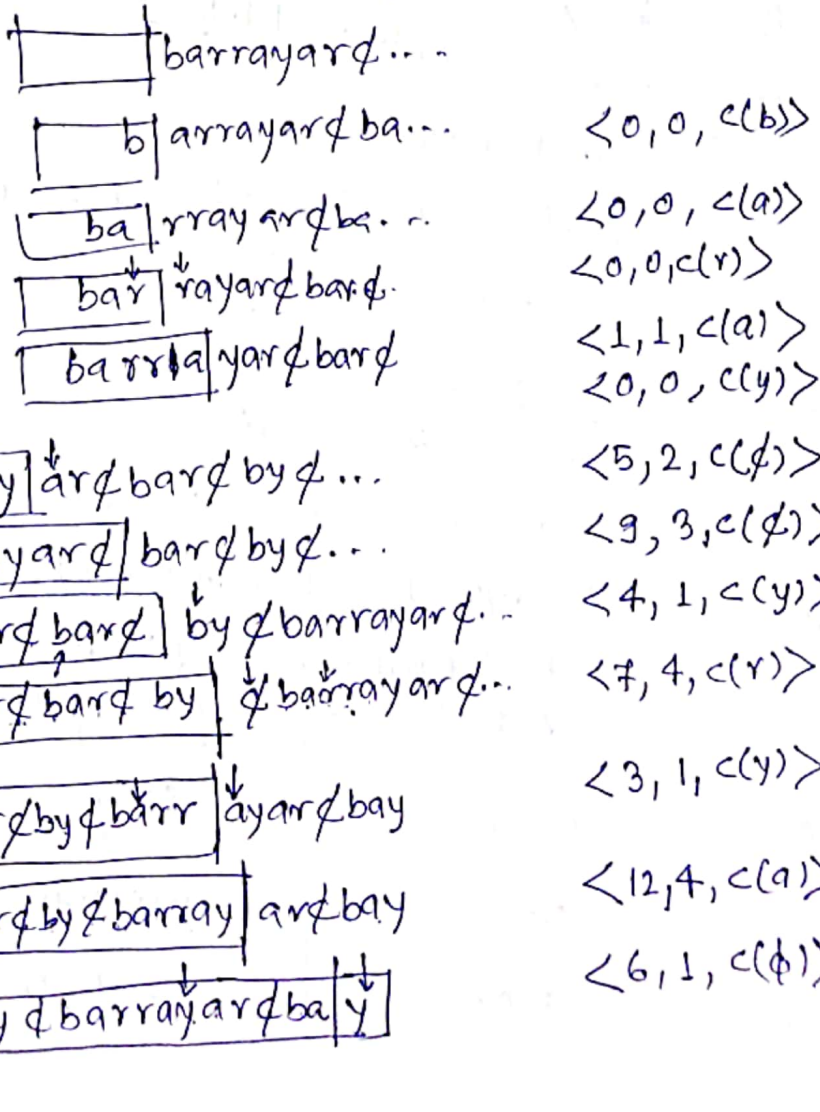
barrayar¢bar¢by¢barrayar¢bay.

window size = 30 & look-ahead buffer of size 15. Furthermore assume c(a)=1, c(b)=2, c(¢)=3, c(r)=4 & c(y)=5.

Sol:

Search window = 15
Look Ahead = 15

barrayar¢bar¢by¢barrayar¢bay



Note: full search window →

Applications –

(1) File compression – UNIX compress : One of earlier applications of LZW : The size of dictionary is adaptive . Once dictionary is filled up, the size of dictionary is doubled. ~~to 1024~~ If max size of codeword, b_{max}, is set by user then once dictionary contains $2^{b_{max}}$ entries, compress becomes static dictionary coding technique. At this ~~tg~~ time algorithm monitors the compression ratio. If compression ratio falls below a threshold, the dictionary is flushed, and dictionary building process is restarted.

(2) Image compression – **Graphics Interchange Format (GIF)**
GIF was developed by compuserve Information Service to encode graphical images. It is another implementation of LZW algorithm and is very similar to compress command. The compressed image is stored with first byte being the minimum number of bits b per pixel in original image. The binary number 2^b is defined to be the clear code. This code is used to reset all compression and decompression parameters to a start-up state. The initial size of dictionary is 2^{b+1}, when this fills up, the size is doubled, until maximum dictionary size is reached.

- The codewords from LZW algorithm are stored in blocks of characters. The block is terminated by a block terminator. The end of compressed image is denoted by an end-of-information code with a value of 2^b+1. This codeword should appear before the block terminator.

(3) **Compression over Modems :- V.42 bis**

The ITU-T (International Telecommunication Union) recommendation V.42 bis is a compression standard devised for use over a telephone network along with error-correcting procedures described in CCITT (consultative committee for International Telephony and Telegraphy). This algorithm is used in modems connecting computers to remote users. The algorithm is described in two modes:

(a) transparent modes
(b) compressed mode.

- In transparent mode data is transmitted in uncompressed form while in the compressed mode an LZW algorithm is used to provide compression.
- In compressed mode, initial dictionary size is negotiated at the time a link is established b/w transmitter and receiver. V.42 bis recommendation suggests a value of 2048 for dictionary size. It specifies minimum size of dictionary to be 512 i.e. each codeword 9 bits long, whereas three entries in the dictionary are reserved for control codewords.

Codeword	Name	Description
0	ETM	Enter transparent mode
1	FLUSH	Flush data
2	STEPUP	Increment codeword size.

- By using these codewords and certain procedure, dictionary is always matched to the current source statistics.

PNG : Portable Network Graphics : is based on LZ77, It allows for match lengths of b/w 3 and 258. At each step the encoder examines three bytes. If it cannot find a match of at least three bytes it puts out the first byte and examines the next three bytes. ~~If it cannot find a match of at least three bytes it put~~ So at each step it either puts out the value of a single byte, or literal, or pair <match length, offset>. The alphabets of Literal and match length are combined to form an alphabet of size 286 (indexed by 0-285)

0-255 : Literal bytes
256 : end of block symbol
257-285 : codes for ranges of lengths b/w 3 and 258.

Advantage of PNG over GIF :-

PNG format offers several advantages over GIF files:

- Better compression, resulting in reduced file size (typically 5-25% better).
- PNG supports variable transparency (alpha channels).
- PNG offer control of image brightness (via gamma correction) and color correction.
- PNG supports two-dimensional interlacing (a method of progressive display) while GIF doesnot

Bilevel image compression :

JBIG is an early lossless image compression standard from the Joint Bi-level Image experts Group, standardized as ISO/IEC standard 11544. It is widely implemented in fax machines. JBIG is also known as JBIG1 and was designed for compression of binary images, particularly for faxes, but can also be used on other images. JBIG is based on arithmetic coding that also uses a relatively minor refinement. It bases the probability estimates for each encoded bit on the values of previous bits and the values in previous lines of pictures. It uses 1024 to 4096 codes depending on resolution.

JBIG2 : is an image compression standard for bi-level images, developed by Bi-level Image Experts Group. It is suitable for both lossless and lossy compression. It typically generates files 2-4 times smaller than JBIG. Ideally, a JBIG2 encoder will segment the input page into regions of texts, regions of halftone images and regions of other data. Various ~~different~~ algorithms are applied on each regions independently and compression is done according to demand.

1) Generic decoding procedure } → generic region decoding procedure
   → generic refinement region decoding procedure
2) Symbol region decoding } dictionary based decoding procedure
3) Halftone Region decoding }

Context-based Algo. / Predictive Coding

- If a sequence of symbols being encoded doesnot consists of independent occurences of symbols, then the knowledge of which symbols have occurred in neighbourhood of symbol being encoded, will give idea of value of the symbol being encoded. i.e. distribution is based on history of sequence. Such scheme is known as predictive or context based algorithm.

PPM (Prediction with partial match) :→

In ppm algorithm, we store those contexts that have occurred in the sequence being encoded.

- At the begining of encoding, we will need to code letters that have not occurred previously in the context.
- If letter encountered have not occured previously in this context, then escape symbol <ESC> is used to signal that the letter to be encoded has not been seen in this context.

Basic Algorithm – attempts to use the largest context. The size of largest content is predetermined. If the symbol to be encoded has not previously been encountered in this context, an escape symbol is encoded and the algorithm attempts to use the next smaller context.

- Each time a symbol is encountered, the count corresponding to that symbol is updated.
- By following l^n and u^n we can find the symbol to transmit:

$$l^n = l^{n-1} + \frac{(u^{n-1} - l^{n-1} + 1)(cum\_count(x_n) - 1)}{total\_count}$$

$$u^n = l^{n-1} + \left\lfloor \frac{(u^{n-1} - l^{n-1} + 1)(cum\_count(x_n))}{total\_count} \right\rfloor - 1$$

⇒ **Escape symbol** :→ Different Approaches have been used for number of counts to be assigned to escape symbol.

Method A : Assign the count 1 for escape symbol.

context	symbol	count
prob	a	10
	l	9
	o	3
	<ESC>	1
	Total count	23

Method B : In this method, we reduce the count of each of symbols by 1 and that value is added to escape symbol

context	symbol	count
prob	a	9
	l	8
	o	2
	<ESC>	3
	Total count	22

Method C : It is a variant of method B, the count assigned to the escape symbol is the number of symbols that have occured in that context.

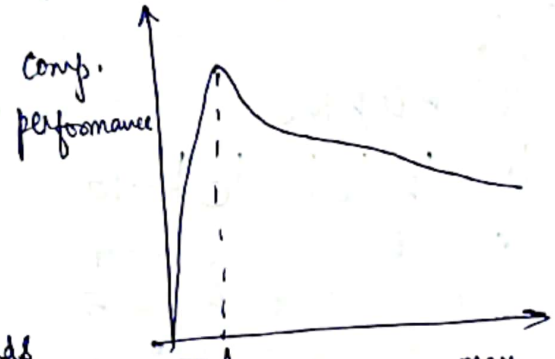
context	symbol	count
prob	a	10
	l	9
	o	3
	<ESC>	3
	Total count	25

. On the average of three methods depending on variation in data, Method C seems to provide the best performance.

Length of context :

A longer maximum length results in a higher probability, if the symbol to be encoded has a non-zero count with respect to that context. In other ways, a long maximum length also means long sequences of escapes which can increase the number of bits used to encode the sequence.

Compression performance vs maximum context length.



→ initial sharp increase in performance, after some value A, steady drop in performance found

'A' value depends on the characteristics of source sequence.

Exclusion Principle : The basic idea behind arithmetic coding is the division of unit interval into subintervals, The smaller the subinterval, the more bits are required to distinguish it from other subintervals. This in turn means that if size of subinterval increases, the number of bits required to represent the letter will decreases, the reduction rate in bits is given by exclusion principle in ppm (prediction with partial match).

eg.

context	Symbol	count
ob	l	10
	0	3
	<ESC>	2
	TC	15
b	l	5
	o	3
	a	4
	r	2
	<ESC>	4
	TC	18

prob ↑ to be encoded

modified table for exclusion example

context	Symbol	count
b	a	4
	r	2
	<ESC>	2
	Total count :	8

Burrows-Wheeler Transform (BWT) ⇒

Given a sequence of length N, create N-1 other sequences where each of these N-1 sequences is a cyclic shift of original sequence. These N sequences are arranged in lexicographic order. The encoder then transmits the sequence of length N created by taking last letter of each sorted, cyclically shifted sequence. This sequence of last letters L and the position of original sequence in sorted list are coded and sent to decoder.

eg: Encode the sequence 'this␣is␣the' 11 character ⇒ 11 permutations

cyclic shift ↳ Chronological order

•	t	h	i	s	␣	i	s	␣	t	h	e
	h	i	s	␣	i	s	␣	t	h	e	t
	i	s	␣	i	s	␣	t	h	e	t	h
	s	␣	i	s	␣	t	h	e	t	h	i
	␣	i	s	␣	t	h	e	t	h	i	s
	i	s	␣	t	h	e	t	h	i	s	␣
	s	␣	t	h	e	t	h	i	s	␣	i
	␣	t	h	e	t	h	i	s	␣	i	s
	t	h	e	t	h	i	s	␣	i	s	␣
	h	e	t	h	i	s	␣	i	s	␣	t
	e	t	h	i	s	␣	i	s	␣	t	h

→ 10, 4, 5, 7, 0, 6, 8, 1, 9, 3, 2

$$L = \begin{bmatrix} s \\ s \\ h \\ t \\ t \\ h \\ ␣ \\ i \\ i \\ ␣ \\ e \end{bmatrix}_{0\,1\,2\,3\,4\,5\,6\,7\,8\,9\,10} \quad F = \begin{bmatrix} ␣ \\ ␣ \\ e \\ h \\ h \\ i \\ i \\ s \\ s \\ t \\ t \end{bmatrix}_{0\,1\,2\,3\,4\,5\,6\,7\,8\,9\,10}$$

Real sequence at position = 10

for decoding: Real sequence at 10

f[10] = t

t = L[4] = f[4] = h
h = L[5] = f[5] = i
i = L[7] = f[7] = s
s = L[0] = f[0] = b
b = L[6] = f[6] = i
i = L[8] = f[8] = s ... soon.

decoded sequence
this b is ....

Move to front coding: Start with initial listing of source alphabet. The symbol at top of list is assigned the number 0, the next one is assigned the number 1, and so on. The first time a particular symbol occurs, the numbers corresponding to its place in the list is transmitted. and then it is moved to top of list.

example: encode: sshttthbiike assume source alphabet $A = \{b, e, h, i, s, t\}$

sf:

b e h i s t	
0 1 2 3 4 5	s = 4
s b e h i t	s = 0
h s b e i t	h = 3
t h s b e i	t = 5
h t s b e i	t = 0
b h t s e i	h = 1
i b h t s e	b = 3
b i h t s e	i = 5
e b i h t s	i = 0
	b = 1
	e = 5

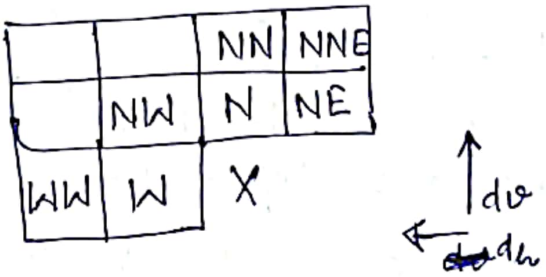
final sequence of codes: 4035013 5015

CALIC : Context Adaptive lossless image compression :

- CALIC scheme uses context and prediction of pixel values. It functions in two modes : for
  1. gray scale images.
  2. bi-level images.

In an image, a given pixel generally has a value close to one of its neighbours, which neighbour has closest value depends on local structure of image.

- Let us consider pixel to be encoded is marked with X, the pixel above it is called north pixel, left is west pixel and so on.



For boundaries

$$d_h = |W - WW| + |N - NW| + |NE - N|$$
$$d_v = |W - NW| + |N - NN| + |NE - NNE|$$

The relative value of d_h and d_v are used to obtain the initial prediction of pixel X.

1. If d_h is much higher than d_v :⇒ large horizontal variation ⇒ pick North (N) to be initial prediction.
2. If d_v is much higher than d_h :⇒ large vertical variation ⇒ pick west (W) to be initial prediction
3. If differences are moderate or smaller ⇒ predicted value is weighted average of neighbourhood pixels.

pseudo code for initial prediction ($\hat{x}$) ⇒

```
if d_h - d_v > 80
    x̂ ⇐ N
else if d_v - d_h > 80
    x̂ ⇐ W
else
{
    x̂ ← (N+W)/2 + (NE - NW)/4
    if d_h - d_v > 32
        x̂ ← (x̂ + N)/2
    else if d_v - d_h > 32
        x̂ ← (x̂ + W)/2
    else if d_h - d_v > 8
        x̂ ← (3x̂ + N)/4
    else if d_v - d_h > 8
        x̂ ← (3x̂ + W)/4
}
```

- Now we perform refinement of our initial prediction by comparing with vector

$$[N, W, NW, NE, NN, WW, 2N-NN, 2W-WW]$$

then vertical and horizontal variations and previous error is predicted by $\boxed{\delta = d_h + d_v + 2|N - \hat{N}|}$

- Once prediction is obtained, the difference b/w pixel value and prediction has to be encoded. Then coding context is found.

$$\left.\begin{aligned} 0 \le \delta \le q_1 &\Rightarrow \text{context } 1 \\ q_1 \le \delta < q_2 &\Rightarrow \text{context } 2 \\ q_2 \le \delta < q_3 &\Rightarrow \text{context } 3 \\ &\vdots \\ q_7 \le \delta \le q_8 &\Rightarrow \text{context } 8. \end{aligned}\right\}$$

- Finally the residual is coded using coding context.

CALIC algorithm: Summary ⇒

1. Find initial prediction $\hat{x}$.
2. Compute prediction context.
3. Refine prediction by removing the estimate of bias in that context.
4. Update bias estimate
5. Obtain residual and remap it so the residual value lie b/w 0 and M-1, M = size of initial alphabet.
6. Find the coding context K.
7. Code the residual using coding context.

JPEG-LS: In JPEG-LS, the prediction approach is a variation of Median Adaptive Prediction in which predicted value is the median of N, W, NW pixels

The initial prediction is obtained by

```
if NW ≥ max(W,N)
    x̂ = max(W,N)
else
    { if NW ≤ min(W,N)
        x̂ = min(W,N)
      else
        x̂ = W+N−NW
    }
```

} Algo represents finding median

The initial prediction is then refined using average value of prediction error in that particular context. measures of differences are computed as:

$D_1 = NE - N$

$D_2 = N - NW$

$D_3 = NW - W$

NW N NE
W X

The values of these differences defines a three-component context vector Q. The components of Q (Q_1, Q_2 and Q_3) are defined by following mapping:

$$D_i \le -T_3 \Rightarrow Q_i = -4$$
$$-T_3 < D_i \le -T_2 \Rightarrow Q_i = -3$$
$$-T_2 < D_i \le -T_1 \Rightarrow Q_i = -2$$
$$-T_1 < D_i \le 0 \Rightarrow Q_i = -1$$
$$D_i = 0 \Rightarrow Q_i = 0$$
$$0 < D_i \le T_1 \Rightarrow Q_i = 1$$
$$T_1 < D_i \le T_2 \Rightarrow Q_i = 2$$
$$T_2 < D_i \le T_3 \Rightarrow Q_i = 3$$
$$T_3 < D_i \Rightarrow Q_i = 4$$

where T_1, T_2, T_3 are +ve coeff. defined by user.

Given 9 possible values for each component of context vector, $9 \times 9 \times 9 = 729$ possible context. to reduce context, for coding process, replace context vector Q whose first non-zero elts is negative by $-Q$. This reduces the context number to 365.

- SIGN is used as prediction refinement step and finally prediction error r_n is mapped into a interval of size of original pixel values. Mapping used are:

$$r_n < \frac{-M}{2} \Rightarrow r_n \leftarrow r_n + M$$
$$r_n > M/2 \Rightarrow r_n \leftarrow r_n - M.$$

finaly prediction errors are encoded using golomb codes.

Multiresolution Approaches :-

Multiresolution models generates representations of an image with varying spatial resolution. This usually results in a pyrahid like representation of image, with each layer of pyramid serving as prediction model for layer immediately below.

One of the applications of multiresolution approach is progressive image transmission :

Suppose a user wants to browse through a number of images in a remote database, suppose images are of size 1024×1024 and user have to look through 30 images, then it may take much time but a better approach would be send an approximation of each image first, which does not require too many bits

but still is sufficiently accurate to give users an idea of what image looks like. If users find the image to be of interest, they can request a further refinement of approximation, or the complete image. This approach is called progressive image transmission.

Facsimile Encoding: In facsimile (or fax) transmission, a page is scanned and converted into a sequence of black or white pixels. Considering the time to transmit an A4-size document over phone lines, four groups are described as: CCITT (now ITU-T) recommendations

analog scheme

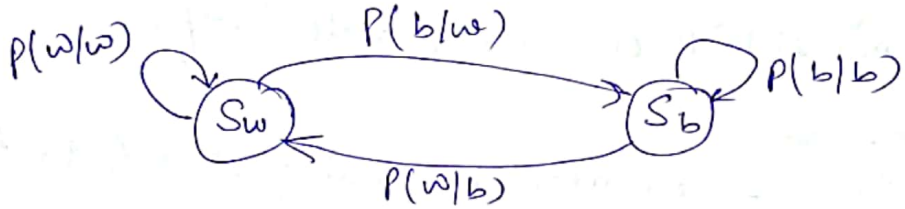
- **Group 1**: capable of transmitting an A4-size document in about six minutes over phone lines using an analog scheme. Apparatus is standardised in recommendation T.2.
- **Group 2**: " " three minutes ..... recommendation T.3.

digital scheme

- **Group 3**: apparatus uses a digitized binary representation of facsimile and because it is a digital scheme, it can use data compression and is capable of transmitting an A4-size document in about a minute. Apparatus is standardised in recommendation T.4.
- **Group 4** This apparatus has same speed requirement as group 3. The apparatus is standardized in recommendations T.6, T.503, T.521 and T.563.

ITU-T Recommendations: T.82 ← JBIG
T.88 ← JBIG2

Run-length coding: Capon model is used to describe run-length coding; a two-state Markov model with states S_w and S_b and corresponding probabilities is shown as



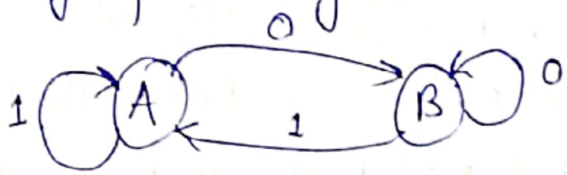
In case of run-length coding, rather than coding the color of each pixel separately, simply code of length of the runs of each color is found out.

For eg: suppose, we have 190 white pixel, followed by 30 black pixels, followed by 210 white pixels. then instead of coding 430 (190+30+210) pixels individually, we would code the sequence 190, 30, 210 along with an indication of the color of first string of pixels. Coding of runs instead of coding individual value is called run-length coding.

Dynamic Markov compression ⇒ DMC uses more general framework to take advantage of relationships and correlations, or contexts, that extends beyond a single symbol.

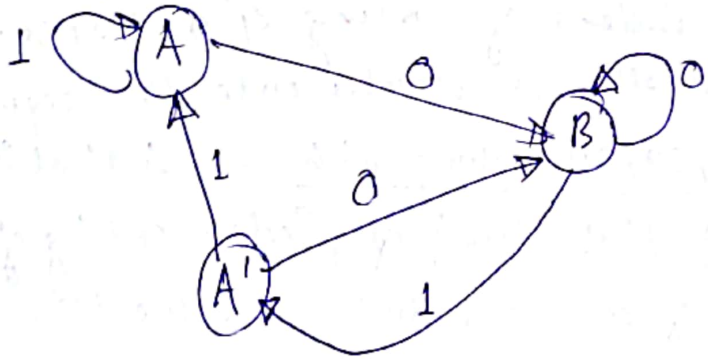
- Consider a sequence of black & white pixels, represented as, black by 0 and white by 1. If current value is 0, the probability that the next value is 0 is higher than if current value was 1. It can be.

shown by following two state model:



Consider state A, the probability of next value being 1 changes depending on whether we reached state A from state B or state A ~~state~~ itself.

- ~~We can have~~
  - A three state model, by cloning state A to A', Now if we see a white pixel value after a run of black pixels, we go to state A'. The probability that next value will be 1 is being very high.



By this way, estimate of probabilities of next pixel value, take into account the current pixel & previous pixel as well.