- **Arithmetic Coding** :- Generating variable length codes through arithmetic coding method.

≐ **Coding Rate** :- Average nos of bits used to represent a symbol from a source.

≐ **Entropy** :- lowest rate at which the source can be coded.

≐ **Pmax** :- Probability of most frequently occuring symbol

eg consider a source that puts out iid letters from the alphabet $A = \{a_1, a_2 a_3\}$ with the probability model $P(a_1) = 0.95$, $P(a_2) = 0.02$ and $P(a_3) = 0.03$. Entropy for this source is 0.335 bits/symbol. Average length is = 1.05 bits/symbol.

symbol	codeword
a_1	0
a_2	11
a_3	10

- the difference b/w the average code length and entropy is 0.715 bits/symbol is redundecy

which is 213% of the entropy.

- Another approach is extended alphabet approach.
- We can see that it is more efficient to generate codeward ~~for q~~ for group or sequences of symbols rather than generating a seperate codeward for each symbol in a sequence.
- However, this approach becomes impractical when we try to obtain Huffman codes for long sequence.

~~## Arithmetic~~

Arithmetic coding :- In arithmetic coding, a unique identifier or tag is generated for the sequence to be encoded.

- The tag corresponds to binary fractions which becomes the binary code for the sequence.
- A unique arithmetic code can be generated for a sequence of length 'm' without the need of generating codewords for all sequence of length 'm'.

II Coding a Sequence :- • In order to distinguish a sequence of symbol from another sequence of symbols, we need a unique tag with unique identifier.

• One possible set of tags for representing sequences of symbols are the no's in the unit interval [0, 1). For this we need a function that will map sequences of symbols into unit interval.

• A function that maps random variables and sequences of random variables, into the unit interval is the cumulative distribution function (cdf) of the random variable associated with the source.

• cdf will be used in developing in arithmetic codes.

IIa Generating a Tag :-

• Tag works by reducing the size of the interval in which tag resides as more and more elements of the sequence are received.

- We start by first dividing the unit interval into subintervals of the form $[F_x(i-1), F_x(i)]$, $i = 1, 2 \ldots m$, as the minimum value of cdf is '0' and minimum value is '1'. this exactly partitions the unit interval.
- We associate the subinterval $[F_x(i-1), F_x(i)]$ with symbol a_i. the 'j'th interval corresponding to the symbol 'a_j' is given by

$$\left[F_x(k-1) + F_x(j-1)\Big/\big(F_x(k) - F_x(k-1)\big),\ F_x(k-1) + F_x(j)\Big/\big(F_x(k) - F_x(k-1)\big)\right]$$

eg:- consider a 3-letter alphabet $A = \{a_1, a_2 a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, $P(a_3) = 0.2$, Also $F_x(1) = 0.7$, $F_x(2) = 0.8$ and $F_x(3) = 1$

Sol:- the partition in which the tag resides depends on the first symbol of the sequence being encoded. for eg- if the first symbol is a_1, the tag lies in the interval $[0.0, 0.7)$; if the symbol is a_2, than tag lies in the interval $[0.8, 1.0)$.

- Once the interval containing the tag has been determined, the rest of the unit interval

is discarded, and this restricted interval is again divided in some proportions as the original interval.

- Suppose the first symbol was a_1, the tag would be contained is the subinterval $[0.0, 0.7)$.
- This subinterval is than subdivided in exactly the same proportions as the original intervals, yielding the subintervals $[0.0, 0.49), [0.49, 0.56)$ and $[0.56, 0.7)$.
- The first partions as before corresponds to a_1, the second partition to a_2, The tag value is then restricted to lie in the interval $[0.49, 0.56)$ .. - and so on.

⇒ Generating a Tag :- Define a the random variable $X(a_i) = i$, Suppose we wish to encode the sequence 1321. from the probability model, we know that

$F_x(k) = 0 \quad , k \leq 0, \quad F_x(1) = 0.8, \quad F_x(2) = 0.82,$

$F_x(3) = 1.$

$F_x(k) = 1 \quad , k > 3$

Initializing $u^{(0)}$ to 1, and $l^{(0)}$ to 0, the first element of sequence 1 results in the

following update

$$l^{(1)} = 0 + (1-0)0 = 0$$

$$u^{(1)} = 0 + (1-0)(0.8) = 0.8$$

ie, the tag is contained in the interval $[0, 0.8)$.

⇒ the second element of the sequence is 3, using the update equation we get

$$l^{(2)} = 0 + (0.8-0) F_x(2) = 0.8 \times 0.82 = 0.656$$

$$u^{(2)} = 0 + (0.8-0) F_x(3) = 0.8 \times 1.0 = 0.8$$

Therefore, the interval containing the tag for the sequence 13 is $[0.656, 0.8)$.

⇒ the third element 2, then

$$l^{(3)} = 0.656 + (0.8 - 0.656) F_x(1)$$

$$= 0.656 + 0.144 \times 0.8 = 0.7712$$

$$u^{(3)} = 0.656 + (0.8 - 0.656) F_x(2)$$

$$= 0.656 + 0.144 \times 0.82 = 0.774080.$$

and the interval for the tag is $[0.7712, 0.774080)$.

⇒ Continuing with the last element, the upper and lower interval for sequence 4th element is $l^{(4)} = 0.7712$ & $u^{(4)} = 0.773504$.

and the Tag for the sequence 1321 can be generated as -

$$\bar{T}_x(1321) = \frac{0.7712 + 0.773504}{2}$$

$$\boxed{\bar{T}_x(1321) = 0.772352}$$

Deciphering the Tag :-

[Reverse procedure of Generating a tag]

⇒ The tag is useless unless we can also decipher it with minimal computational cost.

eg :- The tag value obtained in previous eg. is 0.772352. The interval containing this tag value is a subset of every interval obtained in the encoding process.

⇒ For decoding procedure, we decode the element in the sequence in such a way that the upper and lower limits $u^{(k)}$ & $l^{(k)}$ will always contain the tag value for each k.

∴ we start with $l^{(0)} = 0$ & $u^{(0)} = 1$. After decoding the first element of the sequence x_1 & the upper & lower limits become -

$$l^{(1)} = 0 + (1-0) F_x(x_1 - 1) = F_x(x_1 - 1)$$
$$u^{(1)} = 0 + (1-0) f_x(x_1) = f_x(x_1)$$

ie the interval containing the tag is ~~xcs~~)

$[F_x(x_1-1), F_x(x_1)]$

✓ * Now choose or find the value of x_1 for which 0.772352 lies in the interval $[F_x(x_1-1), F_x(x_1)]$, if $x_1 = 1$ the interval is $[0, 0.8)$, if $x_1 = 2$ the interval is $[0.8, 0.82)$ if $x = 3$ the value of interval is $[0.82, 1.0)$,

→ As 0.772352 lies in the interval $[0.0, 0.8)$ we take $x = 1$.

⇒ Now repeat this procedure for 2nd, 3rd & 4th elements.

$$l^{(2)} = 0 + (0.8-0) F_x(x_2 - 1) = 0.8 F_x(x_2 - 1)$$
$$u^{(2)} = 0 + (0.8-0) f_x(x_2) = 0.8 f_x(x_2)$$

&

$$l^{(3)} = 0.656 + (0.8 - 0.656) F_x(x_3 - 1)$$
$$u^{(3)} = 0.656 + (0.8 - 0.656) f_x(x_3)$$

&

$$l^{(4)} = 0.7712 + (0.77408 - 0.7712) F_x(x_4 - 1)$$
$$u^{(4)} = 0.7712 + (0.77408 - 0.7712) f_x(x_4)$$

Hence the Algorithm for decipher the tag.

1. Initialize $l^{(0)} = 0$ & $u^{(0)} = 1$
2. For each k find $t^* = (\text{tag} - l^{(k-1)}) / (u^{(k-1)} - l^{(k-1)})$
3. Find the value of x_k for which $F_x(x_k - 1) \leq t^* < F_x(x_k)$
4. Update $u^{(k)}$ & $l^{(k)}$
5. Continue until the entire sequence has been decoded.

⇒ There are two ways to know when the entire sequence has been decoded.

① The decoder may know the length of the sequence

② a particular symbol is denoted as an end-of-transmission symbol, the decoding of this symbol would bring the decoding process to a close.

Example:- Encode "BILL GATES" using Arithmatic coding Technique

(1)

chr.	freq.
A	0.1
B	0.1
E	0.1
G	0.1
I	0.1
L	0.2
S	0.1
T	0.1

Conti.

(2)

Character	Probability	Range
Space	0.1	[0.00, 0.10)
A	0.1	[0.10, 0.20)
B	0.1	[0.20, 0.30)
E	0.1	[0.30, 0.40)
G	0.1	[0.40, 0.50)
I	0.1	[0.50, 0.60)
L	0.2	[0.60 – 0.80)
S	0.1	[0.80 – 0.90)
T	0.1	[0.90 – 1.00)

char	low	high
B	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
Space	0.25720	0.25724
G	0.257216	0.2572220
A	0.2572164	0.25721680
T	0.25721676	0.25721680
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

hence the final low value - 0.2572167752 will unique encode the Name "BILL GATES."

Arithmetic Coding Vs Huffman Coding -

- Arithmetic coding requires less m/o as symbol representation is calculated on the fly.
- It is more suitable for high performance models where there are confident predictions
- In large collections of text and images, Huffman coding is likely to be used for text, and its redundancy is quite small.
- Arithmetic coding is used for the images and much better than Huffman coding unless a blocking technique is used.
- Huffman decoding is generally faster than arithmetic decoding.
- In arithmetic coding it is not easy to start decoding in the middle of the stream, while in Huffman coding we can use
- A major advantage of arithmetic coding over Huffman coding is the ability to separate the modeling and coding aspects of the compression approach.

Generating a Binary Code :-

- We want a binary code that will represent 'x' in a unique and different way in a efficient manner.
- 'Tags' forms a unique representation for the sequence. This means that the binary representation of the tag forms a unique binary code for all sequence.

Uniqueness and Efficiency of the Arithmetic Code

- $\bar{T}_x(x)$ is the number in the interval [0, 1). A binary code for $\bar{T}_x(x)$ can be obtained by taking the binary representation of this no. and truncating it to $\boxed{l(x) = \lceil \log \frac{1}{P(x)} \rceil + 1 \text{ bits.}}$

eg:- Consider a source A that generates letters from an alphabet of size four.

$A = \{a_1, a_2, a_3, a_4\}$ with Probability

$P(a_1) = \frac{1}{2}$, $P(a_2) = \frac{1}{4}$, $P(a_3) = 1/8$, $P(a_4) = 1/8$

⇒ A Binary code for four letter alphabets

Symbol	P_x	$\bar{T}_x$	$\bar{T}_x$ in Binary	$\lceil \log \frac{1}{P_x} \rceil + 1$	Code
1.	0.5	0.25	0.010	2	01
2.	0.75	0.625	0.101	3	101
3.	0.875	0.8125	0.1101	4	1101
4.	1.0	0.9375	0.1111	4	1111

Here we can see that, a code obtained in this fashion is a unique decodable code.

Dictionary Techniques :-

- Here we will look at techniques that incorporate the structure in the data in order to increase the amount of compression.
- These techniques - both static and dynamic or Adaptive build a list of commonly occuring patterns and encode these patterns by transmitting their index in the list.

⇒ Introduction :- some application, the O/P of the source consists of recurring patterns. A classic eg is a text source in which certain patterns or words recur constantly.
- Some patterns do not occur or if they do, occur with rarity. So techniques for handling these 2 data is different.

1) Static Approach
2) Adaptive Approach.

I) Static Approach :- (Dictionary)

- This approach is best if we have prior knowledge about the source.

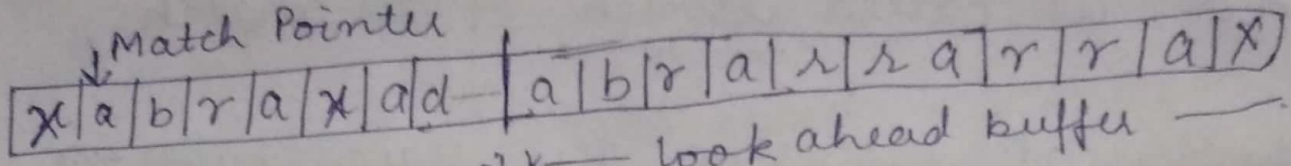
Eg: If the task were to compress, the student records at university, a static dictionary approach may be the best.

- The reason is- we know ahead of time that certain words such as "Name" and "student_id" are going to appear in almost all the records.
- These schemes would work well only for the application and data they were designed for.

II.) Adaptive Dictionary :-

- Most of the adaptive dictionary-based techniques have their roots in two landmark papers by Jacob Ziv and Abraham Lempel, in 1977 and 1978.
- These papers provide two different approaches to adaptively building dictionaries and each approach has given rise to number of variations
- The approach based on 1977 papers are said to belong to LZ77 family, while approaches based on the 1978 papers are said to belong to LZ78 or LZ2 family

IIa The LZ77 Approach :-



- In LZ77 approach, dictionary is simply a portion of the previously encoded sequence.
- The sliding window has the input sequence, encoder examines the sequence.
- Window has 2 parts :-
  1. Search Buffer :- contains portion of recently encoded sequence
  2. Look ahead Buffer :- contains next sequence of the buffer to be encoded
- To encode the sequence in the look ahead buffer, the encoder moves a search pointer back through the search buffer until it encounter the first match to the first symbol in the look-ahead buffer.
- offset - It is the distance of the pointer from the lookahead buffer.
- Length of the match :- The number of consecutive symbols in the search buffer that match consecutive symbols in the look ahead buffer, starting with the first

symbol, is called length of the match.

- The encoder searches the search buffer for the longest match.
- Once the longest match has been found, the encoder encodes it with a triple <o, l, c>, where 'o' is the offset, 'l' is the length of the match and 'c' is the codeword corresponding to the symbol in the look-ahead buffer that follows the match.
- If the size of the search buffer is 'S', the size of the window (search and look ahead buffer) is 'W' and the size of the source alphabet is 'A', then the number of bits needed to code the triple using fixed length code is ~~$\lceil \log_2 S \rceil$~~

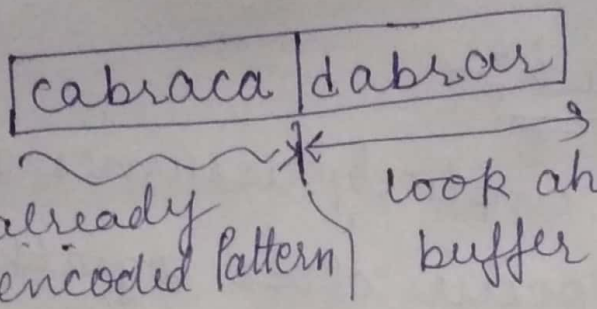
$$\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 A \rceil$$

eg:- using the LZ 77 approach-

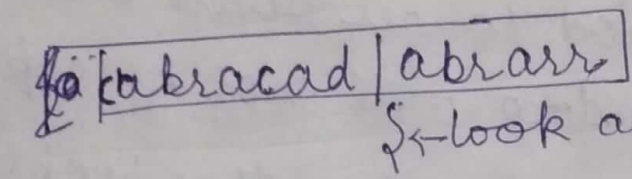
Suppose the sequence to be encoded is

.....cabracadabrarrarrad.

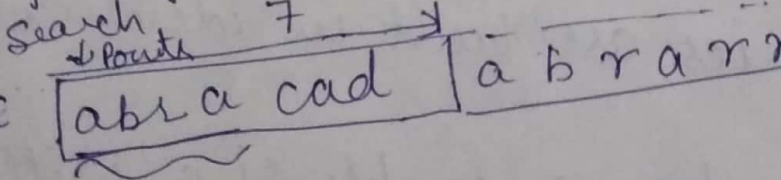
and suppose the length of window is 13, the size of the look ahead buffer is 6.



Sol. we look back in the already encoded portion of the window to find a match for d. There is no match, so we transmit the triple <0, 0, c(d)>. c(d) is the code for character d.



- Now we find a match for a at an offset of 2. The length of this match is 1.
- So now we encode the string 'abra' with the triple <7, 4, c(r)> and move the window forward by five characters.



- As we see that, LZ77 scheme is very simple adaptive scheme that requires no prior knowledge of the source and seems to require no assumptions about the characteristics of the source.

IIb The LZ 78 approach:-

- The LZ-77 approach implicitly assumes that like patterns will occur close together.
- It makes use of this structure by using the recent past of the sequence as the dictionary for encoding.
- However this means that any pattern that occurs over a period longer than that covered by the coder window will not be captured.
- The LZ-78 algorithm solves this problem by dropping the reliance on the search buffer and keeping an explicit dictionary.
- This dictionary has to be built at both the encoder and decoder and care must be taken that the dictionaries are built in identical manner.
- The input are coded as a double <i, c> with 'i' being an index corresponding to the dictionary entry that was longest match to the input, and 'c' being the code for the character in the input following the match portion of the input. '0' is used in case of no match.

eg:- (LZ-78 approach)

The sequence is -

'wabbaβwabbaβwabbaβwabbaβwoowβwoowβwoo'

Initial dictionary -

Index	Entry
1	w
2	a
3	b

* → while LZ-78 algo has the ability to capture patterns and hold them indefinitely. It also has a serious drawback, dictionary keeps growing without bound.

∴ Development of Dictionary

Encoder	Index	Entry
<0,C(w)>	1	w
<0, c(a)>	2	a
<0, c(b)>	3	b
<3, c(a)>	4	ba
<0, c(β)>	5	β
<1, c(a)>	6	wa
<3, c(b)>	7	bb
<2, c(β)>	8	aβ
<6, c(b)>	9	wab
<4, c(b)>	10	baβ
<9, c(b)>	11	wabb
<8, c(w)>	12	aβw
<0, C(o)>	13	o
<13, c(b)>	14	oβ
<1, C(o)>	15	wo
<14, C(w)>	16	oβw
<13, C(o)>	17	oo

II c) LZW Encoding :-

- Let us take the previous sequence -
"wabbabwabbabwabbabwabbabwoobwoobwoo"
- The alphbet for the source is = { b, a, b, o, w }
- The LZW dictionary initialy looks like.

Index	Entry
1	b̸
2	a
3	b
4	o
5	w

⇒ The LZW dictionary for encoding for the given sequence - "wabba - - - . . . oo"

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

Index	Entry
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

hence the encoder o/p sequence is —

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

IId). LZW- Decoding :-

- the o/p sequence of LZW- encoding procedure is —

⇒ first ~~abbo~~ take the encoded sequence —

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

⇒ Now constructing 11th index of LZW dictionary while decoding — .

index	entry
1	b
2	a
3	b
4	o
5	w
6	wa
7	ab
8	bb
9	ba
10	ab
11	b....

and so on —

- The index value '5' corresponds to the letter 'w'; so we decode 'w' as the first element of our sequence.
- The next decoder input is 2, which is corresponds to letter 'a'.
- We decode an 'a' and concatenate it with our current pattern and become the pattern 'wa' As this does not exist with dictionary, so we add it at the 6th element of the dictionary and start new pattern begining with letter 'a'
- The next four inputs 3 3 2 1 correspond to the letters b b a ϕ and generate the dictionary entries ab, bb, ba, aϕ
- The next input is '6', which is the index of the pattern 'wa'. Therefore we decode 'aw' and an 'a'; we first concatenate 'w' to the existing pattern, which is ϕ and form the pattern ϕw.
- Similarly do for further sequence. ——

III Applications of Dictionary Techniques

- Terry Welche's article publication has been steadily increasing number of applications that use some variant of the LZ78 algorithm
- Among the LZ78 variant by far, the most popular is the LZW algorithm.
- There are two best known applications of LZW is —
  - GIF and V.42 bits.

File Compression – Unix Compress :-

- The unix compress command is one of the earlier applications of LZW.
- The size of the dictionary is adaptive, we start with a dictionary of size 512 ie code-word are 9 bits long.
- Once the dictionary is doubled to 1024 entries the codeword transmitted is now 10 bits.
- The size of the dictionary is progessively doubled as it fills up.
- ~~The size of the dictionary.~~
- The maximum size of the codeword bmax, can be set up by the user to b/w 9 & 16 with

16 bits being the default. After 3^{rd} max entries, compress becomes a static dictionary coding technique.

- At this point the algorithm monitors the compression ratio. If the compression ratio falls below a threshold, the dictionary is flushed and the dictionary building process is restarted.

Image Compression – The Graphics Interchange Format (GIF) :–

- The GIF was developed by Compuserve information service to encode graphical images
- It is another implementation of LZW algorithm and is very similar to the compress command.
- The compressed image is stored with the first byte being the min. number of bits b/pixel in the original image
- The binary number 2^b is defined to be the clear code. This code is used to reset all compression & decompression parameter to a start-up date.

- The initial size of the dictionary is 2^{b+1} when this fills up, the dictionary size is doubled as was done in compress algorithm.

- The codeword from the LZW algorithm are stored in blocks of character.
- The characters are 8 bits long and maximum block size is 255.
- Each block is preceded by a header that contains the block size.
- GIF has become quite popular for encoding all kinds of images both computer generated and natural ~~generated~~ images.
- Even if we account for the extra overhead in the GIF files, for these images GIF barely holds its own even with simple arithmatic coding of the original pixels.
- Therefore, for images like these, the straight forward dictionary coding is done.
- ~~End~~

Image compression — Portable N/w Graphics (PNG) :-

- The PNG standard is one of the first standards to be collaboratively developed over the internet.
- The community decided that a patent free replacement for GIF should be developed, and within 3 months, PNG took place
- Unlike GIF, the compression algorithm used in PNG is based on LZ-77.
- In particular it is based on the deflate implementation of LZ-77.
- This implementation allow for match length of b/w 3 & 258. At each step the encoder examines 3 bytes. If it cannot fined a match of 3 bytes, it puts the first byte and examine next 3 bytes.
- An initial estimate of the pixel is first made by adding the pixel to the left and the pixel that is above and subtracting the pixel to the upper left. Then the pixel that is closed to the initial estimate is taken as the estimate.

⇒ Comparison of PNG with GIF and arithmetic coding :-

Image name (sample)	PNG	GIF	Arithmetic Code of Pixel Value	Arithmetic code of Pixel difference
Sena	31,577	51,085	53,431	31,847
Sensin	34,488	60,649	58,306	37,126
Earth	26,995	34,276	38,248	32,137
Omaha	50,185	61,580	56,061	51,393

Comparison Over Modems - V.42 bits :-

- The ITU-T Recommendation V.42 bits is a compression standard devised for use over a telephone n/w along with error-correcting procedures described in OCITT recommendation V.42.
- This algorithm is used in modems connecting computers to remote users.
- The reason for the existence of two modes is that at times the data being transmitted do not have repetitive structure and therefore cannot be compressed using the

LZW algorithm.

- The V.42 bits recommendation suggest periodic testing of the o/p of the compression algorithm to see if data expansion is taking place.
- In the compressed mode, the system uses LZW compression with a variable size dictionary.
- The intial dictionary size is negotiated at the time a link is established b/w the transmitter and receiver.
- When the numbers of entries in the dictionary exceed a pre arranged threshold C_3, the encoders size is incremented by 1 bit. At the same time, the threshold C_3 is also doubled.

Sub Note

- Dictionary-based algo. are being used to compress all kinds of data; ~~however~~
- this approach is most useful when structural constraints restrict the frequently occurring patterns to a small subset of all possible patterns.