Content Based compression :-

Prediction with Partial Match (PPM) :-

- It is the best known content Based algorithm
- It has been not as popular as the Ziv-lempel based algorithms mainly because of the faster execution ~~spp~~ speed of latter algo.
- The idea of PPM algorithm is very simple.
- We would like to use large contents to determine the probability of the (content) symbol being encoded.
- Instead of estimating these probabilities ahead of time, we can reduce the burden by estimating the probabilities as the coding proceeds.
- This way we need to store only those contents that have occured in the sequence being coded.

⇒ Algorithm :-

- If the symbol to be encoded has not previously been encountered in this content an escape symbol is encoded and the algorithm attempts to use the next smaller content.
- If the symbol has not occurred in this content either, the size is further reduced.

- This process continues until either we obtain a content that has previously been encountered with this symbol or we arrive at the conclusion that the symbol has not been encountered previously in any content.

The Escape Symbol :-

- Count of one for the escape symbol, thus inflating the total count in each content by 1. This is called as PPM.

eg suppose in a given sequence 'a' occurs 10 times in the content of probability 'l'. occurs 9 times and '0' occurs ~~000~~ 3 times in the same content.

- In method A, we assign a count of 'one' to the escape symbol, resulting in a total count of 23, which is one more then the no. of times prob has occurred

Counts using Method A

Content	symbol	Count
Prob	a	10
	l	9
	0	3
	(ESC)	1
Total Count		23

Counts using Method B

Content	symbol	Count
Prob	a	9
	l	8
	0	2
	<esc>	3
Total Count		22

Count using method C

Content	Symbol	Count
Prob	a	10
	l	9
	o	3
	<ESC>	3
Total Count		25

- In a second method known as method 'B', we reduce the count of each of the symbols a, l, o by one and give the escape symbol a count of three resulting in the counts.
- A variant of method 'B' appropriately named method 'C' was proposed by Moffat.
- In Method 'C', the count assigned to the escape symbol is the number of symbols that have occurred in that content.
- In this content, method 'C' is similar to method B. the difference comes in the fact that instead of robbing this from the counts of individual symbols, the total count is inflated by this amount
- while there is some variation in the performance depending on the characteristics of the data being encoded of the three methods for assigning counts to the escape symbol on an avg, method 'C' seems to provide best performance,

Length of Content :-

- A longer maximum length will usually result in a higher probability if the symbol to be encoded has a non-zero count with respect to that content.
- However, a long maximum length also means a higher probability of long sequence of escapes which in turn can increase the no. of bits used to encode the sequence.
- If we plot the compression performance vs. max. content length, we see an initial sharp increase in performance until some value of max length, followed by a steady drop as the max. length is further reduced.
- The value at which we see a downturn in performance changes depending on the characteristic of the source sequence.

The Enclusion Principle :-

- The basic idea behind arithmetic coding is the division of the unit interval into sub-intervals, each of which represents a particular letter.

- the smaller the subinterval, the more bits are required to distinguish it from other subintervals.
- If we can reduce the no. of symbols to be represented, the no. of subintervals goes down as well.
- This in turn means that the size of the subintervals increase, leading to a reduction in the no. of bits required for encoding.
- The exclusion principle used in the PPM provides this kind of reduction in rate.

The Burrows-Wheeler Transform (BWT)

- The BWT algorithm also uses the content of the symbol being encoded in a different way for lossless compression.
- BWT algorithm requires that the entire (compression) sequence to be coded be available to the encoder before the coding takes place.

BWT Algorithm :-

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

- this information is sufficient to recover the original sequence at decoder side.
- We start with the sequence of length 'N' and end with a representation that contains N+1 elements. However, this sequence has a structure that makes it highly amenable to compression.
- In particular we will use a method of coding called move-to-front (mtf) which is particularly effective on the type of structure exhibited by the sequence L.

Move to front coding :-

- A coding scheme that takes advantage of long runs of identical symbols is the move-to front (mtf) coding.
- In this coding scheme, we start with some initial listing of the source alphabet.
- The symbol at the top of the list is assigned the number 0, the next one is assigned 1 and so on.
- The first time a particular symbol occurs, the number corresponding to its place in the list

is transmitted, then it is moved to the top of the list.

- If we have a run of these symbols, we transmit a sequence of 0's. This way long runs of different symbols get transformed to a large number of 0's.

Example (BWT) :-

eg Lets encode the sequence

this␣is␣the

- start with all the cyclic permutations of the given sequence. As here N = 11 so there are 11 permutations.

Permutations of this␣is␣the

0	t	h	i	s	␣	i	s	␣	t	h	e
1	h	i	s	␣	i	s	␣	t	h	e	t
2	i	s	␣	i	s	␣	t	h	e	t	h
3	s	␣	i	s	␣	t	h	e	t	h	i
4	␣	i	s	␣	t	h	e	t	h	i	s
5	i	s	␣	t	h	e	t	h	i	s	␣
6	s	␣	t	h	e	t	h	i	s	␣	i
7	␣	t	h	e	t	h	i	s	␣	i	s
8	t	h	e	t	h	i	s	␣	i	s	␣
9	h	e	t	h	i	s	␣	i	s	␣	t
10	e	t	h	i	s	␣	i	s	␣	t	h

- Now sort these sequences in lexicographic (dictionary) order —

sequences sorted into lexicographic Order

0	␣	i	s	␣	t	h	e	t	h	i	s
1	␣	t	h	e	t	h	i	s	␣	i	s
2	e	t	h	i	s	␣	i	s	␣	i	h
3	h	e	t	h	i	s	␣	i	s	␣	t
4	h	i	s	␣	i	s	␣	t	h	e	t
5	i	s	␣	i	s	␣	t	h	e	t	h
6	i	s	␣	t	h	e	t	h	i	s	␣
7	s	␣	i	s	␣	t	h	e	t	h	i
8	s	␣	t	h	e	t	h	i	s	i	␣
9	t	h	e	t	h	i	s	␣	t	h	e
10	t	h	i	s	␣	i	s	␣	t	h	e

→ L

⇒ the sequence of last letters L in this case is

L = sshtth␣ii␣e

- the original sequence appears as sequence no. 10 in the sorted list.
- so encoding of the sequence consists of the sequence L and the index value 10.

⇒ **Decoding Procedure :-** [continue with previous example.]

- Now lets decode the original sequence by using the sequence 'L' and index to the original sequence in the sorted list.
- All the elements of the initial sequence are contained in L, just need to figure out the permutation that will recover the original seq.

① Obtain the permutation is to generate the sequence F consisting of first element of each row. Therefore the sequence F is simply the sequence L in lenicographic order. so

F = ␣␣ehhiisstt

⁂ Now use f and L to decode the sequence.

⁂ If we know that the original sequence is (lenicographic table)
in the k^{th} row then we can begin unraveling the original sequence starting with k^{th} element of F

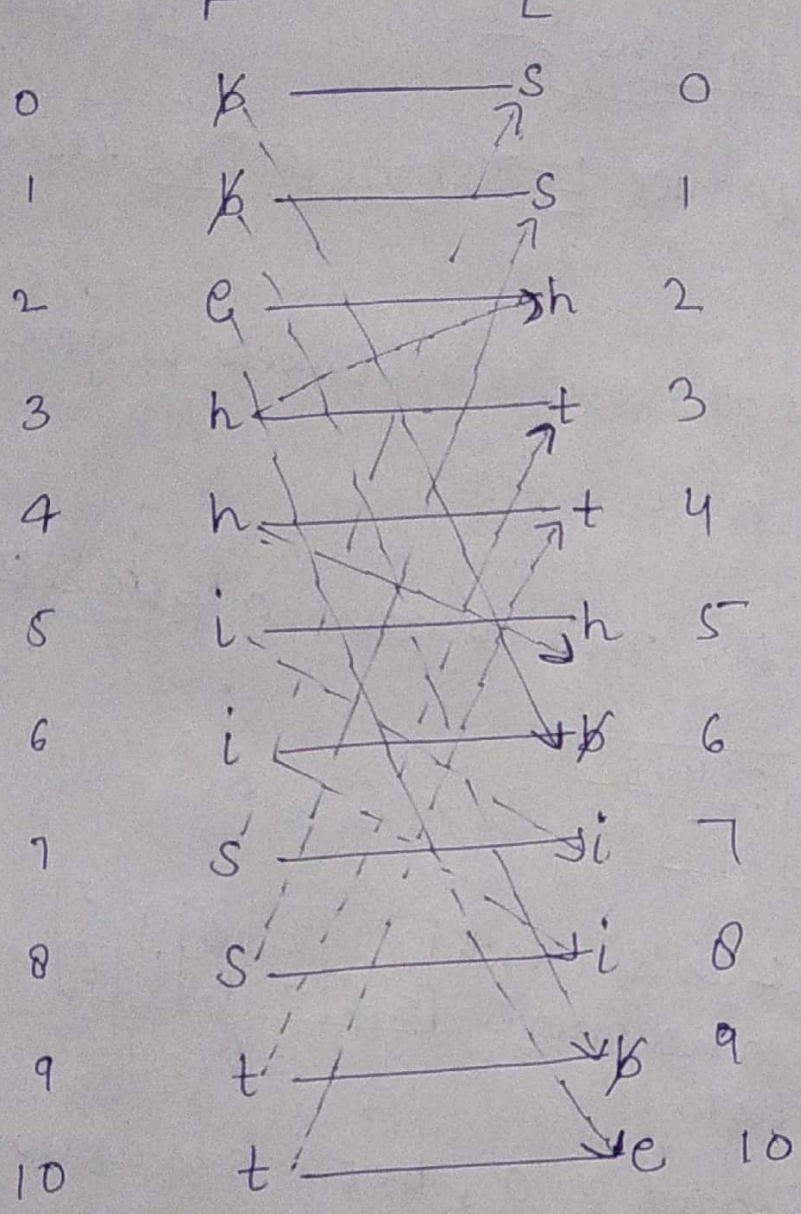
f = [␣ ␣ e h h i i s s t t]

L = [s s h t t h ␣ i i ␣ e]

⁂ the original seq is seq no. 10, so the first letter in of the original seq is F[10] = t.

⁂ To find the letter following t we look for t in the array L.

⁂ There are 2 t's in L so the t in f that was we are working is lower of 2 t's so we conside lower of 2 t's in L.



∴ hence lowest t is at L[4].

∴ Therefore, the next letter in our recommendation sequences is F[4] = h.

∴ the reconstructed sequence is th.

∴ To find the next letter, we look h in L.

∴ Repeat this process.

decoding sequence

index L[4] L[5] L[7] [0] [6] [8] [1] [9] [3] [2] [10]
t h i s ␣ i s ␣ t h e

Findex F[10] [4] F[5] [7] [0] [6] [8] [1] [9] [3] [2].

hence the reconstructed sequence is

this ␣ is ␣ the

Example – MTF coding :-

eg encode the sequence L = ssh t t ␣ i i ␣ e using MTF coding.

Sol Lets assume that source alphabets is given by A = {␣, e, h, i, s, t}

① first we assign the nos to our alphabets

0	1	2	3	4	5
␣	e	h	i	s	t

② first letter of the L is 's' which is encoded as 4. and move s to the top of list.

0	1	2	3	4	5
s	␣	e	h	i	t

③ the next s is encoded as 0, bcoz s is already at the top of the list, hence encode. seq is 40.

④ The next letter is h, which is encoded as 3, & move h at the top of list, seq becomes 403.

0	2	3	4	5
h	s	␣	e	t

⑤ the next letter is t – 5, move t at top of list hence seq - 4035

0	2	3	4	5
t	h	s	␣	e

⑥ continue the procedure for further i/p seq.

hence the encoded seq is **403501350 15**

* '0' means previous that letter is repeat. //