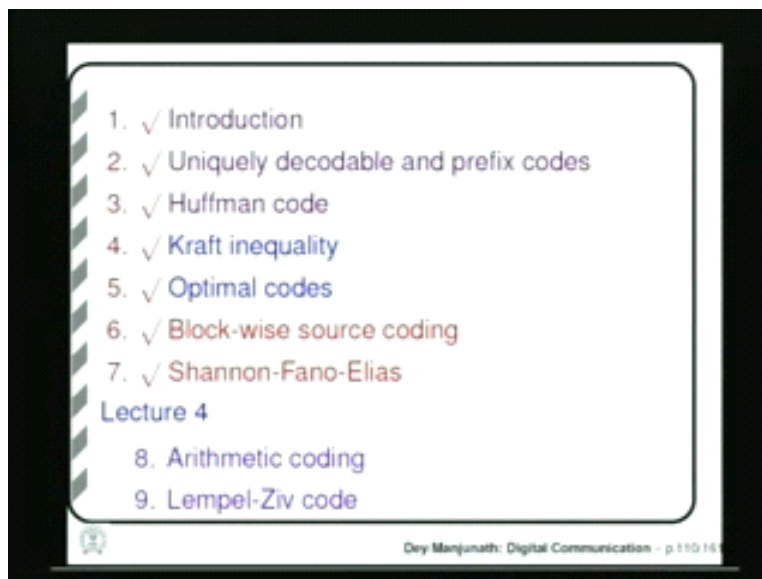


Digital Communication
Prof. Bikash Kumar Dey
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Lecture - 29
Source Coding (Part-4)

We have already had 3 classes on source coding hopefully, this will be the 4th class and the last class on source coding. So, far we have done the following topics: we have done introduction, we have seen why we should do source coding? Why it is possible to do source coding for most of the practical sources? And then we have classified different source coding techniques.

(Refer Slide Time: 01:16)

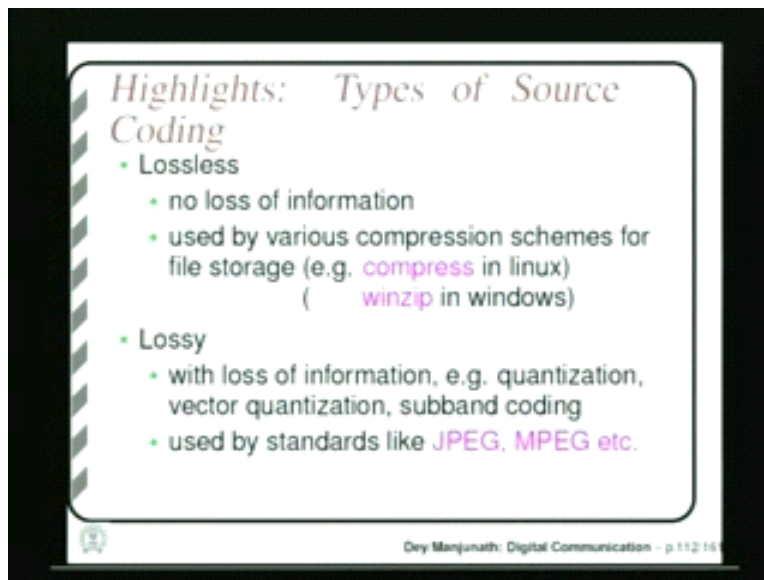


Specially, desired source coding properties we have seen that we should have uniquely decodable codes so that, we can decode any encoded string uniquely in the destination. And then we have seen that, even all uniquely decodable codes are not always instantaneously decodable. So, to have instantaneously decodable code we should have prefix code. So that, no codeword is a prefix of another codeword.

And we have so, far two source coding techniques: 1 of them Huffman code; we have seen that it is an optimal code and it is a prefix code. However, we need to have we need to have all the probabilities of the source to do Huffman coding. And then we have seen Kraft inequality which gives us a condition necessary and sufficient condition for having a code prefix code of a given set of lengths. So, and then we have seen some properties of optimal codes what how much compression is possible to do for a given source. And we have seen that, for getting better compression; it is not always possible to do source coding with individual symbols.

So, most of the times we have to combine many symbols together to do block-wise source coding which will give us better compression. And then we have discussed Shannon-Fano-Elias code which is which we also saw to be asymptotically optimal though not optimal symbol wise. So, we will just go through what we have done so, far. We have seen that there are 2 types of source coding techniques: 1 is lossless and another it is lossy.

(Refer Slide Time: 02:56)



Lossless source coding technique involved some involves no loss of information due to source coding. And in lossy source coding technique there is always some loss of information. So, like source coding technique compression techniques like: JPEG, MPEG

they involve some loss of information. Whereas, our file compression techniques in computer like: WinZip in windows or compress in Linux involved no loss of information. Because, we know that we can recover the file back by using uncompress on such commands.

And we have seen that variable length in coding gives advantage gives compression compared fixed length coding.

(Refer Slide Time: 03:46)

X	Singular	Non-singular but not UD	UD, but not prefix	prefix
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

Dey Manjunath: Digital Communication - p.114-115

Then we have seen these 3 kinds of codes. Singular code has a singular code has 2 same code words in this case, there are all the four code words are same. Non-singular code has all the code words different, but still it may not be uniquely decodable, once you encode a string. Uniquely decodable code is uniquely decodable string-wise, but even then it is not decodable instantaneously. Whereas, prefix code where no code word is a prefix of another code word can we decoded instantaneously.

(Refer Slide Time: 04:22)

Highlights: Huffman code

X	Code-word	Probability			
3	0 1	0.25	0.30	0.30	0.55 → 1
4	1 0	0.25	0.25	0.30	0.45 ↗
1	1 1	0.20	0.25	0.25	
2	0 0 0	0.15	0.20		
5	0 0 1	0.15			

Dry Manjunath: Digital Communication - p.115-116

We have seen the Huffman coding technique. You first arrange the probabilities in decreasing order and then combine the last 2 probabilities into 1 and then assign 0 and 1 to the last 2 probabilities last 2 symbols and then you go on doing it and adding bits to the symbols. Then Huffman code we have seen is a prefix code specially the example we have seen is a prefix code we have we have not proved that, any Huffman code is a prefix code, but anyway we accept that.

(Refer Slide Time: 04:39)

Highlights: Huffman Code

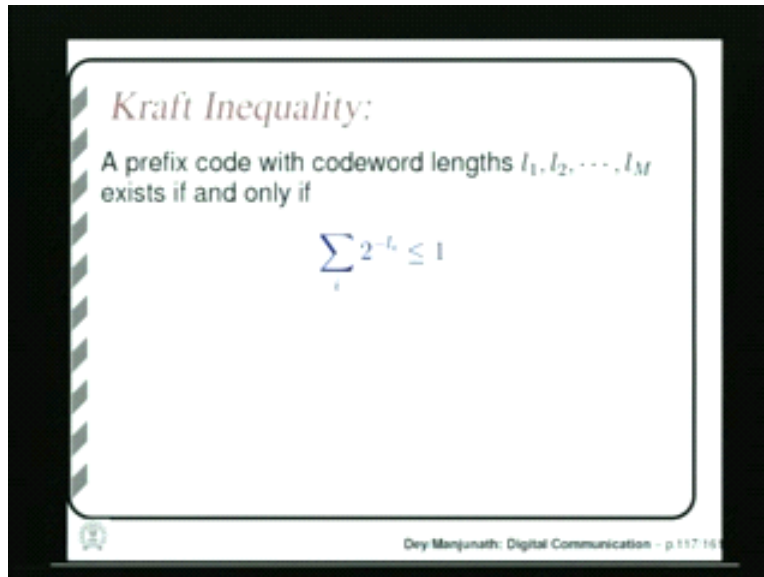
Properties

- Huffman code is a prefix code.
- Huffman code is optimum - no code better than Huffman code for any random variable, i.e., Huffman code gives the minimum average length for any random variable.

Dry Manjunath: Digital Communication - p.116-117

And Huffman code is optimum there is for any random variable there is no code better than Huffman code. So, that is something very interesting and very nice to have.

(Refer Slide Time: 05:02)



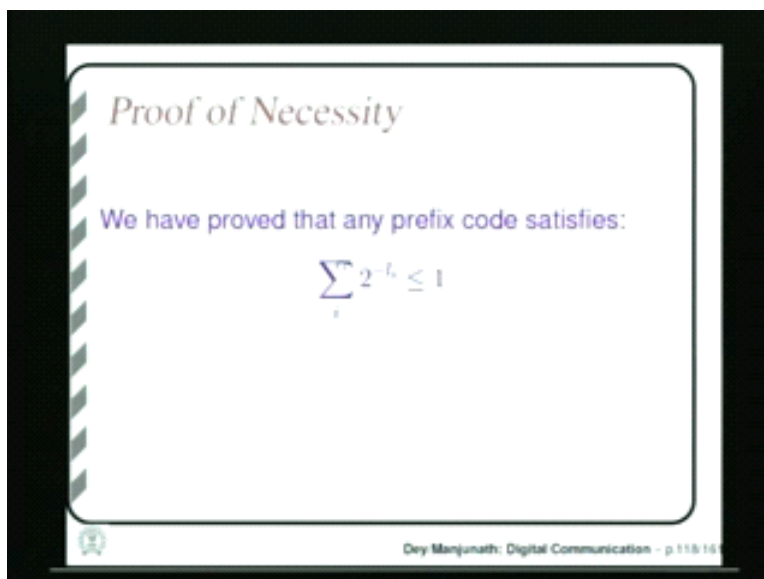
Kraft Inequality:
A prefix code with codeword lengths l_1, l_2, \dots, l_M exists if and only if

$$\sum_i 2^{-l_i} \leq 1$$

Dry Manjunath: Digital Communication - p.117,14

And Kraft inequality it gives a necessary and sufficient condition for a prefix code to exist for a given set of lengths. So, if these lengths are given we know that a prefix code exists with these lengths if and only if this condition is satisfied.

(Refer Slide Time: 05:18)



Proof of Necessity

We have proved that any prefix code satisfies:

$$\sum_i 2^{-l_i} \leq 1$$

Dry Manjunath: Digital Communication - p.118,14

So, we have proved necessity and we have proved sufficiency of the Kraft inequality condition.

Proof of Sufficiency

For a given set of lengths $\{l_1, l_2, \dots, l_M\}$ which satisfies the condition

$$\sum_i 2^{-l_i} \leq 1,$$

we have constructed a prefix code.

So, we have proved that the condition $\sum_i 2^{-l_i} \leq 1$ is sufficient for existence of a prefix code.

Dey Manjunath: Digital Communication - p.119/18

And in Kraft inequality we have seen that we have we have commented that, Kraft inequality also holds for uniquely decodable codes.

(Refer Slide Time: 05:23)

Kraft Inequality: Comments

- Kraft inequality also holds for uniquely decodable codes
- UDC with l_1, l_2, \dots, l_m exists \Rightarrow prefix code with l_1, l_2, \dots, l_m exists

Dey Manjunath: Digital Communication - p.120/18

So, it is not only for prefix codes, but Kraft inequality also gives necessary and sufficient condition for existence of a uniquely decodable code for any set of lengths. And. So, we

from there we are good that, if for a given set of lengths there is a uniquely decodable code there is also a prefix code we got. Because, if there is uniquely decodable code of certain lengths then Kraft inequality is satisfied; then if Kraft inequality is satisfied then there is a prefix code also with those lengths.

So, we should always go for a prefix code of for whatever lengths are given. And also that equality in Kraft inequality means that all the nodes of the maximum level are exhausted are exhausted by the code words. And we have seen that, an optimal code length must be in between these 2 length these 2 limits.

(Refer Slide Time: 06:31)

Optimal code:
A code for a random variable X is optimal if there is no code for the same random variable with smaller average length

Denote the average length of an optimal code as L .

Theorem 0

$$H(x) \leq L < H(x) + 1$$

Dey Manjunath: Digital Communication - p.121-14

We know that it is greater than equal to $H(x)$, but this is something interesting and we have proved both the parts.

(Refer Slide Time: 06:37)

Need to combine many source symbols for source coding

symbols	probability	best symbol-wise code
1	0.9	1
0	0.1	0

Average length = 1 bit
Entropy = $-(0.9 \log_2(0.9) + 0.1 \log_2(0.1))$
= 0.469 bits

Symbol-wise coding can assure only $L \leq H(X) + 1$.
Source coding theorem says: $L \rightarrow H(X)$

Dey Manjunath: Digital Communication - p.122-14

Now, we have seen that block-wise source coding gives better compression.

(Refer Slide Time: 06:43)

Block-wise Encoding

Source Stream:
 $X_1, X_2, \dots, X_n, X_{n+1}, X_{n+2}, \dots, X_{2n}, X_{2n+1}, \dots$

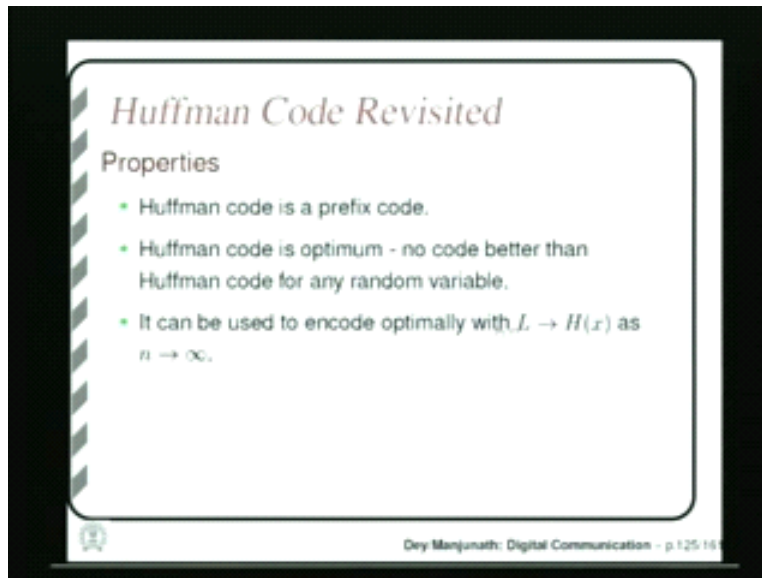
$[X_1, X_2, \dots, X_n]$, $[X_{n+1}, X_{n+2}, \dots, X_{2n}]$, $[X_{2n+1}, \dots]$

$[10 \dots 1]$, $[11 \dots 0]$, $[01 \dots]$

Dey Manjunath: Digital Communication - p.123-14

And where we do we combine N symbols together and encode them as a single code word. And then, we have seen that by doing block-wise source coding and as we increase block length N to infinity we actually, asymptotically reach the source coding bound. So, we get an average code length tends to H_x .

(Refer Slide Time: 07:14)



Huffman Code Revisited

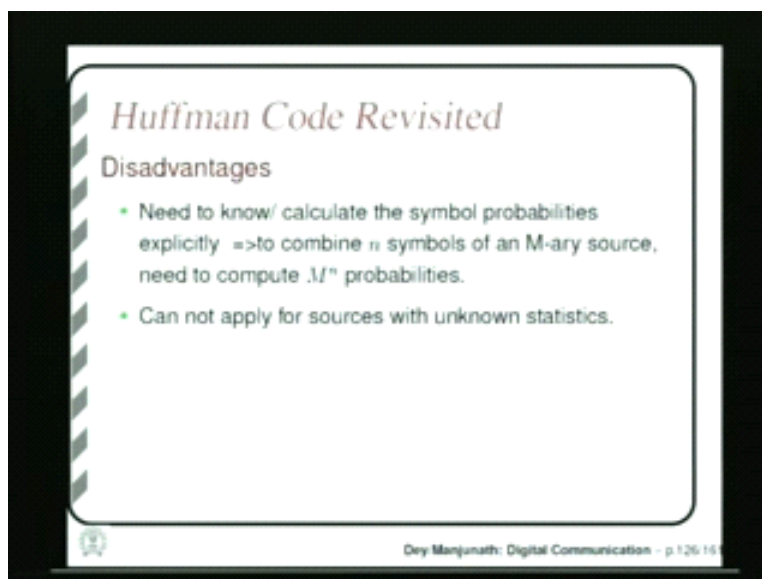
Properties

- Huffman code is a prefix code.
- Huffman code is optimum - no code better than Huffman code for any random variable.
- It can be used to encode optimally with $L \rightarrow H(x)$ as $n \rightarrow \infty$.

Dey Manjunath: Digital Communication - p.125/14

Then we have seen that, Huffman code using Huffman code block-wise we can actually achieve source coding theorem the limit given by source coding theorem on the average code length because, Huffman code is optimal. So, L must be in that range within range H_x and H_x plus 1.

(Refer Slide Time: 07:33)



Huffman Code Revisited

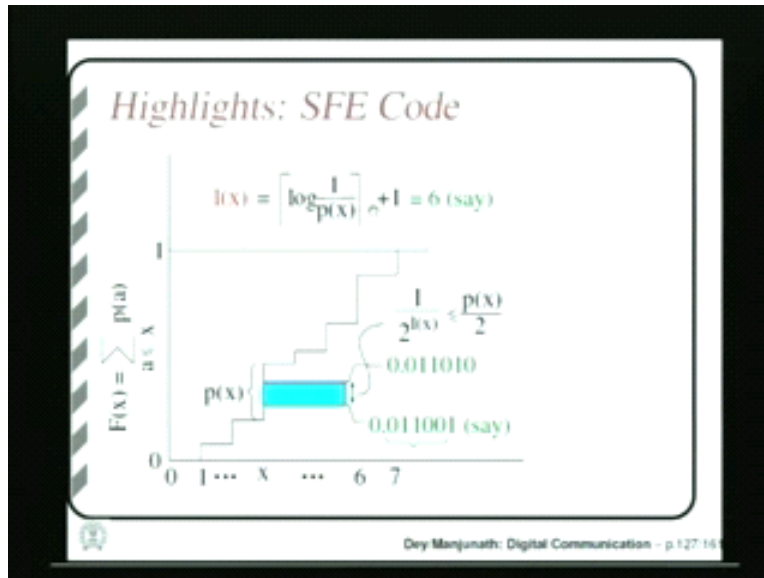
Disadvantages

- Need to know/ calculate the symbol probabilities explicitly => to combine n symbols of an M -ary source, need to compute M^n probabilities.
- Can not apply for sources with unknown statistics.

Dey Manjunath: Digital Communication - p.126/14

Then we have seen that Huffman code has some disadvantages that you have to compute all the probabilities and it cannot to be applied to sources with unknown statistics. Then we have seen that another source coding technique called Shannon-Fano-Elias code.

(Refer Slide Time: 07:48)



Where we have first drawn the, you have taken the cumulative distribution function of the random variable and then if you want to get the code word for x we see this jump. And we take the half way between them take this midpoint and then take compute this length of the code word to be this way. And then, quantize that binary number to so many bits. So, in this case if it is 6 we take the midpoint express it in binary representation. So, 0 point something. So, there are bits after point.

So, you take only 6 bits out of it that this $l(x)$ whatever is $l(x)$. So, then you take this as the code word after the point there are 6 bits. And if you add 1 to the next to the last bit we get an interval which gives, which has all the numbers all the code words having this as prefix because, any code word which has this as prefix will represent a binary number which will lie in between these 2 limits because; it will be less than this.

This is obtained from this by adding 1 to this, but all the other code words will have this number as the first 6 bits. So, we this way we have seen that all the code words certainly code word for next symbol will be somewhere here represent a number here. So, this

cannot be somewhere inside this. So, any code word cannot be a prefix of another code word if we construct the code in this using this technique. So, this gives us a prefix code. And we have seen that this code also gives us an average length which is slightly higher than, what is possible to get for an optimal code.

(Refer Slide Time: 09:40)

Performance of SFA code

- $$L = \sum_i P(x_i)l_i = \sum_i P(x_i) \left(\lceil \log \left(\frac{1}{P(x_i)} \right) \rceil + 1 \right)$$

$$= \sum_i P(x_i) \lceil \log \left(\frac{1}{P(x_i)} \right) \rceil + 1$$

$$\leq \sum_i P(x_i) \log \left(\frac{1}{P(x_i)} \right) + 2$$

$$= H(X) + 2 \Rightarrow \text{suboptimal}$$
- SFA is **asymptotically optimum**
 For combined coding of n symbols,
 $L = L_n/n \leq H(X) + 2/n \rightarrow H(X)$ as $n \rightarrow \infty$.

Dey Manjunath: Digital Communication - p.129/140

For optimal code we know that L is less than equal to H_x plus 1 whereas, for this we are guaranteed that average length is less than equal to H_x plus two. But even this minute increase in the average length does no harm asymptotically; as N tends to infinity we still achieve H_x . Now, we will we are going to arithmetic coding which will it is basically Shannon-Fano-Elias code used for for block length large block length and iterative process and we will do Ziv-Lempel code a Lempel-Ziv code.

(Refer Slide Time: 10:22)

What we are going to do

- **Arithmetic Code:** No need to compute all the source probabilities. - Universal and asymptotically optimal.
- **Lempel-Ziv Code:** Tabular coding - universal.

Dey Manjunath: Digital Communication - p.130/14

Which is a tabular coding where, probability is not at all explicitly coming in to the construction of the code. So, that is something different from arithmetic code as well as, as well as Huffman code.

(Refer Slide Time: 10:41)

Arithmetic code: What is it!

1. A way of doing SFA encoding of large blocks of symbols sequentially.
2. Sequential encoding reduces delay.
3. Block length n can be increased without increasing encoding/decoding delay.

Dey Manjunath: Digital Communication - p.132/14

So, we start with the arithmetic code. What is arithmetic code? Arithmetic code is a way to do Shannon-Fano-Elias code coding for large block length iteratively. So, as symbols

come we keep constructing the code. So, we do not receive the whole block and then do coding. So, this has several advantages: 1 is that sequential encoding reduces delay if we receive the whole block and then do coding the whole coding starts after we receive that block of symbols. So, the encoding delayed the total delay is much more then if you keep doing partially the sequentially the encoding.

So, also block length N can be increased if you do sequentially without increasing the delay. If you receive the whole block and then encode then if we increase the block length we have to receive. So much, much larger than if you take smaller block length and then, so the delay will increase. But if you keep encoding sequentially the delay will not increase as we increase the block length because, as you get partial first if you say we received four symbols you probably have encoded few bits 3 bits you know first 3 bits for example. Then you can transmit those 3 bits and as you go on receiving more symbols you can encode more and more bits.

So, receive the symbols sequentially and you fairly keep encoding the bits. So, this reduces the delay and it is it does not increase as the block length is increasing. So, in fact, when we use arithmetic code for file compression for example, we take the whole file as a single block. So, the compression is really maximum, you are taking the maximum possible block length so because; we do not pay in terms of delay there.

So, it is possible to take whole block length as a single symbol in such a case. Okay. So, we are going to basically do Shannon-Fano-Elias coding for a block of symbols. So, we are we are trying to see if we can do encoding sequentially. So, first of all let us see what is the Shannon-Fano-Elias Code for the whole block then, we will see whether we can do it sequentially. So, if we have N symbols and we want to do Shannon-Fano-Elias Code first thing we have do is, we have to draw the cumulative distribution function of that N symbols together.

So, for the N symbols together we have to draw cumulative distribution function. So, before doing that, we have to first arrange all possible N symbol blocks in sequential order because; to draw cumulative distribution function you have arrange them in sequential order. So, what is the ordering we should use? For example, if the random

variable is binary 0 1 if you combine 2 bits then you have 0110 you know 0011 you have to arrange them in sequential order.

So, you have to use some kind of order in the blocks. So, the most commonly used order is the lexicographic ordering. Its called so, because a dictionary that's how it is that is how the words are arranged. So, if you want to arrange 0011 1001 in a sequential order, the most natural is the lexicographic ordering that is 0001 1011. So, that is how we do. If we have 3 symbols and we have to combine 2 symbols together 3 possible values and 2 symbols together then this is the natural ordering lexicographic ordering.

(Refer Slide Time: 14:19)

Block-wise SFA

Lexicography ordering of (X_1, X_2, \dots, X_n)

Example: The pairs (x, y) ($x, y \in \{1, 2, 3\}$) are ordered as $(1, 1) < (1, 2) < (1, 3) < (2, 1) < (2, 2) < (2, 3) < (3, 1) < (3, 2) < (3, 3)$.

- Similar to alphabetic ordering in a dictionary.

Dey Manjunath: Digital Communication - p.133-14

This is similar to alphabetic ordering in a dictionary.

(Refer Slide Time: 14:30)

Computing $F(X_1, X_2, \dots, X_i)$ iteratively
Denote (x_1, x_2, \dots, x_i) by $\mathbf{x}^{(i)}$. Then

$$\begin{aligned} F(\mathbf{x}) &= \sum_{\mathbf{y}^{(i)} \leq \mathbf{x}^{(i)}} P(\mathbf{y}^{(i)}) \\ &= \sum_{\mathbf{y}^{(i-1)} < \mathbf{x}^{(i-1)}} P(\mathbf{y}^{(i-1)}) + \sum_{y_i \leq x_i} P(\mathbf{x}^{(i-1)}, x_i) \\ &= F(\mathbf{x}^{(i-1)} - 1) + \sum_{y_i \leq x_i} P(\mathbf{x}^{(i-1)})P(x_i | \mathbf{x}^{(i-1)}) \\ &= F(\mathbf{x}^{(i-1)} - 1) + P(\mathbf{x}^{(i-1)})F(x_i | \mathbf{x}^{(i-1)}) \end{aligned}$$

the $(i-1)$ -tuple immediately before $\mathbf{x}^{(i-1)}$

Dey Manjunath: Digital Communication - p.134-15

So, suppose we have combined that way we have combined the N N symbol values N length block values in sequential order. Then let us see, what will be the cumulative distribution function value at a certain I 2? Say yeah, here we are taking I number of symbols together. So, we will increase I and see what is happening later, but here let us see that for I number of symbols what will be the value.

So, this is basically by the definition of cumulative distribution function is a summation of all the probability of all the I triples which are before which come before xi before or equal to xi. This is the; this is the definition. Now, this less than equal to is basically according the lexicographic ordering. So, if you arrange these I triples for example, in terms like a dictionary then all the I triples which come before that, in that ordering those probabilities you have to add.

Now, this can be broken into parts this is take the I minus 1 number of first I minus 1 block symbols leave the last 1. So, probability of that so, these things have to be less than this. So, this is this probability plus we can say all the yi less than xi probability this will be yi this this and this. Now this will not be these two will be yi. So, this is probability that xi yi is less than equal to xi. So, if you add these two things we get Fx. Now, this can be written in as this in terms of conditional probability this is yi given xi minus 1.

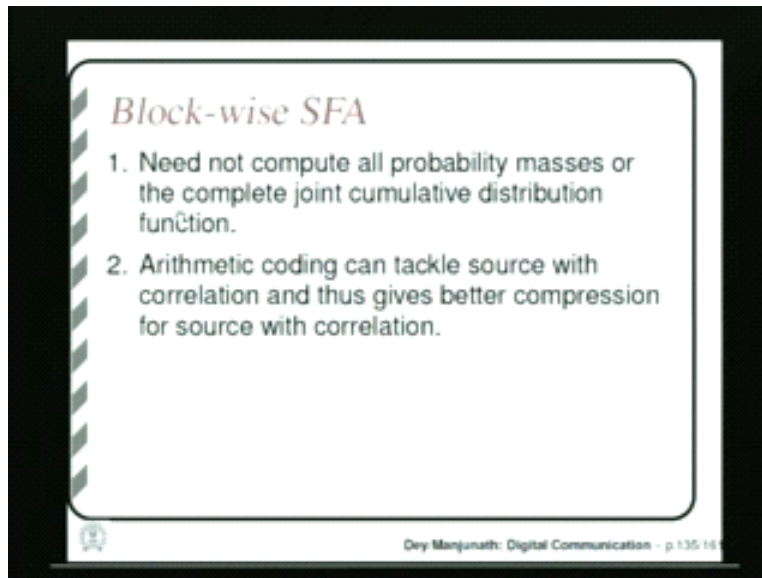
So, probability that the I th symbol is less than x_i less than equal to x_i given that you have received $I - 1$ symbols this is this probability. Now, this can be taken out of this because this is independent of y_i . So, this is this comes outside and this probability is basically the conditional cumulative distribution function evaluated at x_i , this is y_i . So, we get this. Now, what does this mean?

This means that, suppose you compute this function iteratively at $I - 1$ after you receive $I - 1$ you compute the value of the cumulative distribution function of at that $I - 1$ triple the value you have received. So, you will keep that computed and then after you receive I -th symbol you will keep compute the cumulative distribution function of the sequence till I th symbol; using the previously computed value.

So, this thing this minus 1 actually denotes the $I - 1$ triple just before x_{i-1} . So, you have computed this at the after you have received $I - 1$ symbols now you have got x_i also. So, what you need to compute this is also computed before this is does not depend on x_i this basically, denotes the $I - 1$ symbols a block of $I - 1$ symbols. So, that probability also can be computed before hand and then newly you have to compute this probability, this cumulative conditional cumulative distribution function you have to evaluate at x_i .

So, then if you multiply with this and then add with this you get F_x . So, you have to newly compute only this much.

(Refer Slide Time: 18:20)

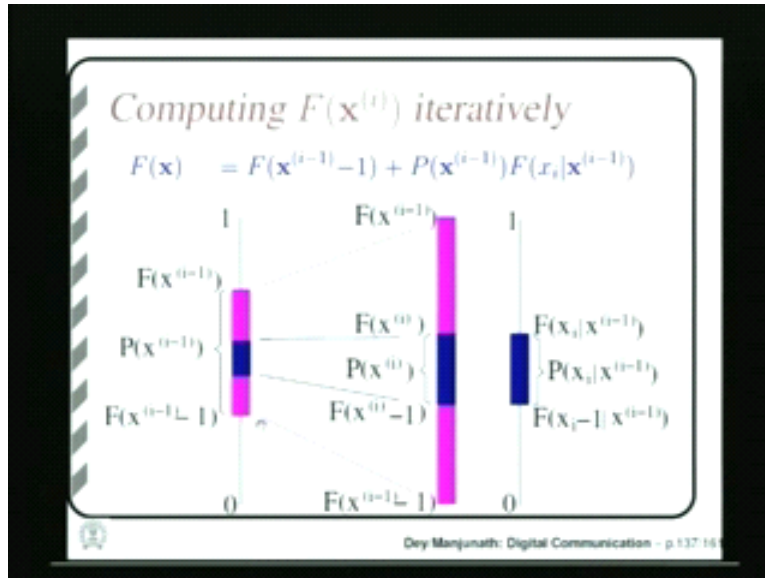


So, what it means is that, you need not compute all the probability masses or the complete joint cumulative distribution function. Because, after you have received x_{i-1} and you have received newly x_i you need not compute F at all the other values. You need to compute at only suppose x_i is 1 then, you need to compute the cumulative distribution function value only at that point. So, conditional cumulative distributions function at that point.

So, that is sufficient; so, you need not compute the whole probability distribution which is required for Huffman coding for example. So, we need not compute the whole probability distribution and arithmetic coding can tackle sources with correlation first of all and thus give better compression for source with correlation. This we are we will not show why, but actually that can be done for any coding with block wise any block wise coding.

So, that will take advantage of correlation also to some to extent. So, geometrically what does it mean to compute the probability distribution iteratively?

(Refer Slide Time: 19:32)



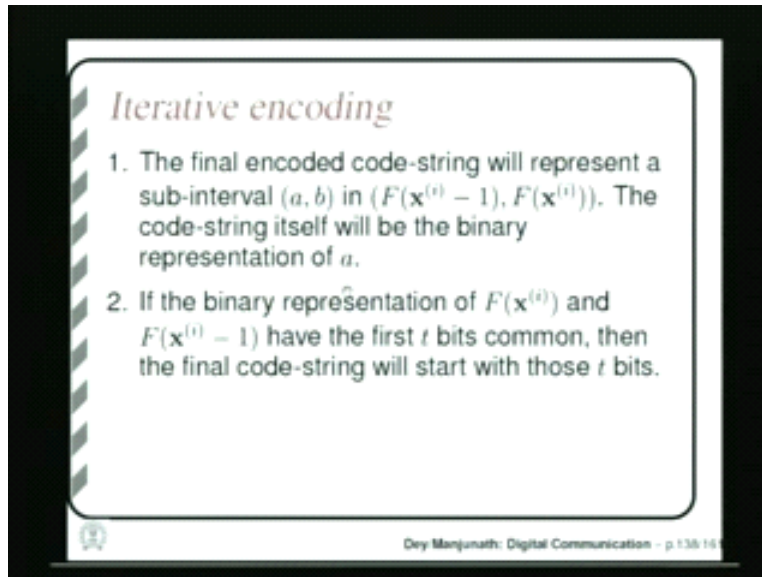
It means following. You have computed say these 3 values in the I minus one'th iteration. You have received I minus 1 symbols and you have computed these three values. Now, as you have received x_i what you are going to do is you are going to compute this F there are conditional probabilities these 3 conditional this is cumulative distribution function at x_i . This is the probability of x_i given this conditional and this is the before x_i what is the cumulative distribution function value.

So, we will newly compute these three we need not compute all the other probabilities cumulative distribution function values here we need to compute only these three values. So, after you have done that we need to compress this whole thing inside this. So, this point will come here this point will come here this point will come here. So, this blue part will come somewhere here. So, this is basically, corresponding to correspond to multiplying this thing by scaling this thing by $P(x_i|x^{(i-1)})$ which we had actually in the computation. We have seen that this needs to be multiplied by this, this corresponds to scaling this part to this much.

So, this will be scaled to this value that is this. This point will come here similarly these 2 points will come here scaled down. So, after you have done that we get these

probabilities at the I -th state. So, we have we had these 3 values now we have these 3 values we keep doing that.

(Refer Slide Time: 21:13)



Now, how do you do the coding? How we do the coding is, we know that in the Shannon-Fano-Elias coding we know that, in the encoded code word the code word will actually represent a number binary number somewhere in this interval. Because, as we keep increasing I actually this interval will be smaller and smaller; first you had this interval then this is scaled down. And you have this much coming inside. So, you have reduced the interval.

So, this actually the final code word will lie somewhere inside this interval it will represent a binary number in this interval somewhere. At I plus 1 level you will have a smaller length inside this smaller interval inside this. So, a binary number inside this will be a code word will be the code word for x that whole block. Now, if these 2 points have the same few say these 2 points we express as binary number both the points. Now, the code word will be somewhere in between.

So, if these 2 points have binary expression common for few bits say first 6 bits, first 7 bits are common in both these numbers. Then we know that, in between any number will also have the same those 7 bits. Afterwards, it may vary from these 2, but these 2 have

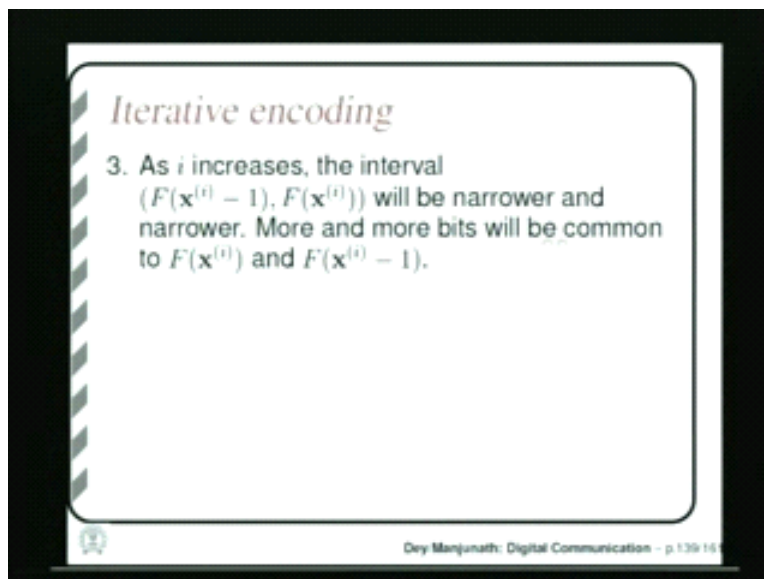
the same 7 bits, but any number in between that also will have the same seven bits. So, we know the first 7 bits of the final encoded code word.

So, we need not wait for the whole file to come before we encode the first 7 bits. We can encode the first 7 bits at the I th iteration itself. So, that is the advantage that is iterative encoding we are talking about. So, to summarize the final encoded code stream will present to subinterval like this a b inside this interval. So, its sub-interval is basically it will be a code a binary number and you add 1 to the last bit you get an interval.

So, these binary numbers will be the code word, but this interval denotes all the numbers of which this is a prefix those will not be code words because, you are constructing a prefix code. So, a will be the; binary representation of a will be the code word. So, we know that now if this 2 have first 3 bits common as you said its first 7 bits common. Then, a will also have first 7 bits same those seven bits then we can take those 7 bits as a partial encoding of the final code.

As i increases the interval this will be narrower and narrower more and more bits will be common to these 2 because, this will be narrower this interval will be narrower. So, more and more bits will come common.

(Refer Slide Time: 24:14)



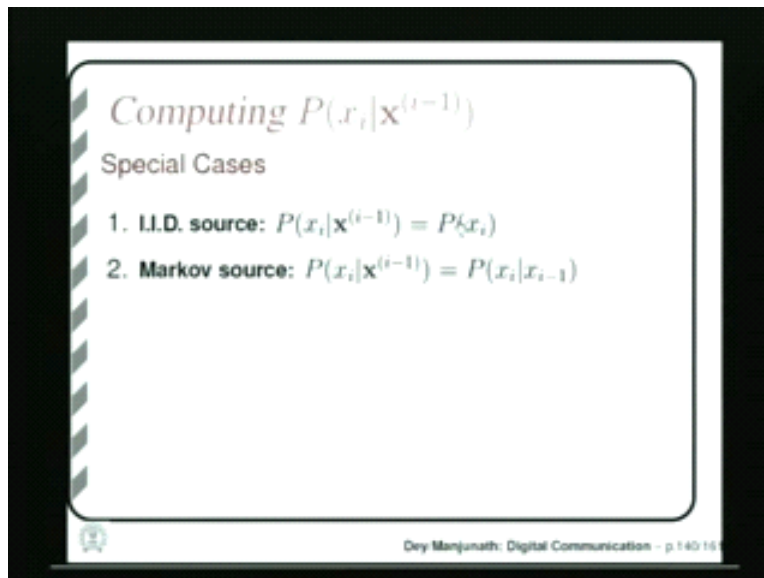
Iterative encoding

3. As i increases, the interval $(F(x^{(i)} - 1), F(x^{(i)}))$ will be narrower and narrower. More and more bits will be common to $F(x^{(i)})$ and $F(x^{(i)} - 1)$.

Dey Manjunath: Digital Communication - p.139/140

First for at I-th level you have 7 bits common at the I plus one'th level you have you may have 9 bits common. So, you can then encode more 2 more bits. So, more and more bits can be encoded as I increases. So, that is the iterative encoding. Now, how do you compute this conditional probability?

(Refer Slide Time: 24:42)



We have seen that this is the key to encoding. So, there are some special cases where this computation is easy. For example, for IID source if you know this probability distribution then conditioning does not change the distribution. So, for IID source it is easy to compute this from Markov source it is simply it is also easy because this conditioning is simply conditioned on X_{i-1} . X_i depends on only X_{i-1} not on the previous symbols.

So, it is easy to compute for these 2 special cases. Now, if you do not know the statistics of the source before hand. How do you tackle the situation? So, the idea is that idea is that as you keep receiving bits or symbols more and more you keep estimating the probability distribution of the source also and use that distribution to do your encoding. So, you have received 1 bit you do not know anything what the distribution is. So, you encode safely you take that uniform distribution or something.

Or some applied distribution you assume initial distribution and then as you receive more and more bits you can estimate the distribution according to some model. And then, use that distribution to encode the further symbols that you receive. So, here are some models we can use to estimate the probability distribution.

(Refer Slide Time: 26:08)

Computing $P(x_i | \mathbf{x}^{(i-1)})$

Models for unknown source statistics

1. Laplace model:

$$P(X_i = a | \mathbf{x}^{(i-1)}) = \frac{F_a + 1}{\sum_b (F_b + 1)}$$

where F_a is the number of a in $\mathbf{x}^{(i-1)}$.
2. Dirichlet model:

$$P(X_i = a | \mathbf{x}^{(i-1)}) = \frac{F_a + \alpha}{\sum_b (F_b + \alpha)}$$

A smaller α tries to learn the statistics faster.

Dey Manjunath: Digital Communication - p.141-142

If suppose, we have received I minus 1 symbols and we are going to receive the I th symbol then and you can estimate the probability distribution; the conditional distribution of the I th symbol based on what you have received in the following way. This is the Laplace model here we basically, suppose this is a binary no suppose this a source any this discrete source then you count the number of a's. So, you are evaluating the conditional distribution conditional probability of receiving a at the I-th symbol if you have received I minus 1 symbols before these values you know.

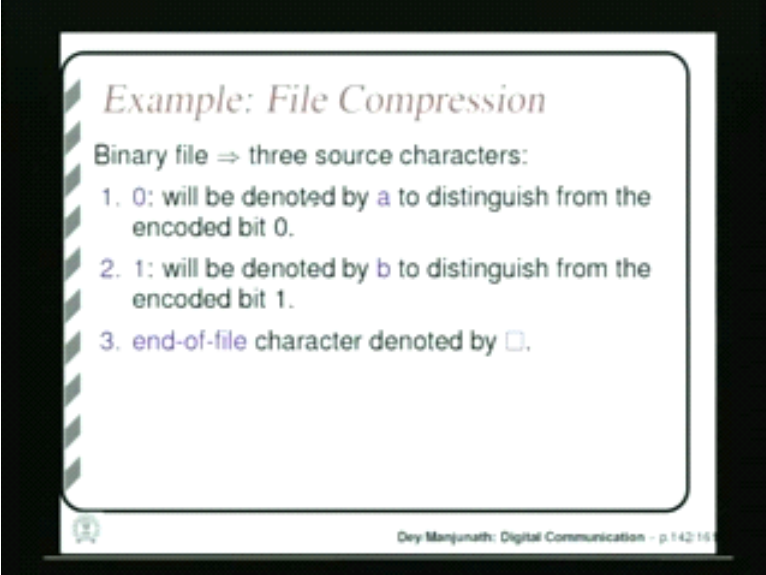
So, you count number of a's you have received before and you count the total number of. So, this is F_a the number a you have received before this plus 1 by this is number of b you have received before. Now, b you take all other values except a all including a a and other values. So, this summation F_b is basically I minus 1 and then plus the number of symbols here. So, this is a model this is basically taking a kind of the frequency of a before hand before previously.

So, for example, if the source is binary and you have received $I - 1$ bits you count the number of zeros and number of ones. Number of zeros is suppose 6 number of ones is seven and you take $\frac{6}{6+1}$ by $\frac{7}{6+7+1}$. So, this will give you the frequency estimate of frequency. Why are these ones kept here? This is because initially you should not have 0 by 0 because, you have you have if you have received 0 symbols. When you are taking $I - 1$ equal to 1 you have received 0 symbols.

So, both F_a F_b all these are zeros. So, you are taking basically 1 by uniform distribution in such a case initially, you assuming uniform distribution. For binary you will assume 0 with probability half 1 with probability half. So, that is the initial distribution we are taking and then, as we take if you have received 0 in the first symbol automatically we will wait 0 more we assume that 0 has a higher probability. So, that way we just keep estimating the probability.

There is another model Dirichlet model where, you replace 1 by alpha. So, if you take alpha less than 1 you try to estimate the estimate the probability faster, but that will have more variation though; this will have more error in the estimate. So, you try to reach to the actual probability faster, but you sacrifice compromise also something. So, in this is the special case of this as you can see if alpha is equal is to alpha is 1 than you get this.

(Refer Slide Time: 29:02)



Example: File Compression

Binary file \Rightarrow three source characters:

1. 0: will be denoted by **a** to distinguish from the encoded bit 0.
2. 1: will be denoted by **b** to distinguish from the encoded bit 1.
3. end-of-file character denoted by \square .

Dey Manjunath: Digital Communication - p.142-143

So, now, we take an example of file compression using that using Laplace model. So, we assume that the file is binary. We have 2 symbols 0 1 and another symbol which can come is end of file we do not know when the file will end. So, we assume that at every time the file may end with probability something so, we assume that probability fixed. We take some probability for that and then except that probability except for end of file probability at a at any instance. There is some or rest of the probability is divided in to 0 and 1.

So, at any time the next bit can be end of file 0 or one. So, its not bit next symbol can be 0 1 or end of file.

(Refer Slide Time: 29:50)

Probability model

1. $P(\square|\mathbf{x}^{(i-1)}) = 0.15$
2. $P(x_i = a|\mathbf{x}^{(i-1)}) = 0.85 \times \frac{F_a + 1}{F_a + F_b + 2}$
3. $P(x_i = b|\mathbf{x}^{(i-1)}) = 0.85 \times \frac{F_b + 1}{F_a + F_b + 2}$

We take a short file `bbba□` for illustration.

Dey Manjunath: Digital Communication - p.143/144

So, we take this model, this example is from Mackay's book. You we take this probability distribution this model. We assume that at anytime the file may end with probability 0.15. So, this is the conditional probability. If you have received anything whatever you have received the file may end with probability this at the I-th symbol. And then the rest of probability is 0.85 that is divided between these 2 depending on how many a;s you have received and how many b's you have received.

So, we will actually these are actually 0 and 1 a and b are actually 0 and 1, but we are denoting by a and b because,we will denote the output bits as 0 1. So, you we want to separate we want to distinguish the input symbols and output symbols both are binary. So, input we are denoting by a b output we are denoting by 0 1. So, 0 may come with some probability and we are basically counting a in the previous symbols and counting b and then taking this estimate of the probability.

So, this is divided between a and b this probability. Now suppose, we have we have received a file which is a very short file of this content bbba then file ends.

(Refer Slide Time: 31:05)

Arithmetic encoding: Example

We need to compute the following probabilities:

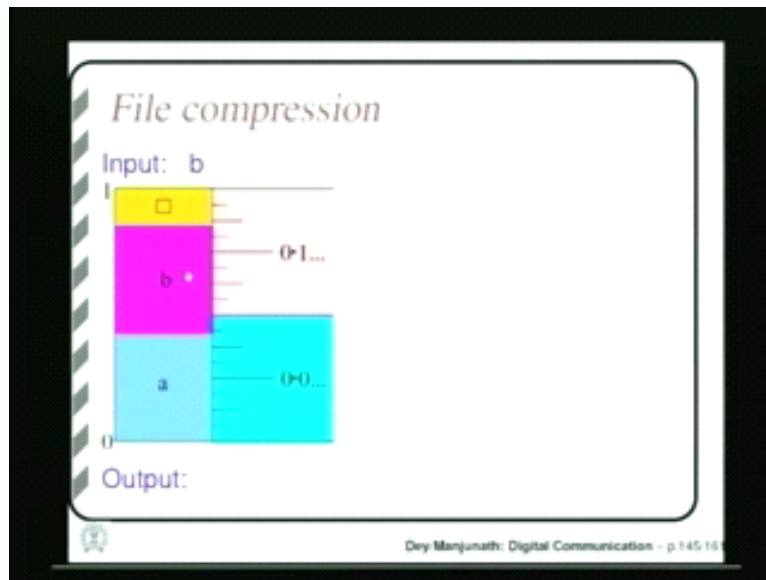
Context $\mathbf{x}^{(i-1)}$	Conditional Probabilities		
	$P(a \mathbf{x}^{(i-1)})$	$P(b \mathbf{x}^{(i-1)})$	$P(\square \mathbf{x}^{(i-1)})$
	0.425	0.425	0.15
b	0.28	0.57	0.15
b \bar{b}	0.21	0.64	0.15
bbb	0.17	0.64	0.15
bbba	0.28	0.57	0.15

Dey Manjunath: Digital Communication - p.144-15

So, now we will see how to do the encoding. First what should I do first you have received nothing. So, you have compute probability of a probability of b and probability of end of file. So, we assume the probability of end of file is 0.15. We have received no bits so you take equal probability for a and b point this summation is 0.85. Then you have received b because the file is bbba, so, you have received b. Then you compute given b what is the next symbol; probability of next symbol.

So, if you compute according to the previous slide you will get this probability. Similarly, after you have received b you have received bb you can compute that probabilities of the third symbol. What is the probability that the third symbol is a. What is the probability that it is b similarly, you compute the probability of end of file. So, similarly you compute all these probabilities conditional probabilities. So, this column is basically probability of a given we have received bb is this. Given bbb what is the probability of a in the next symbol is this. So, similarly compute all this.

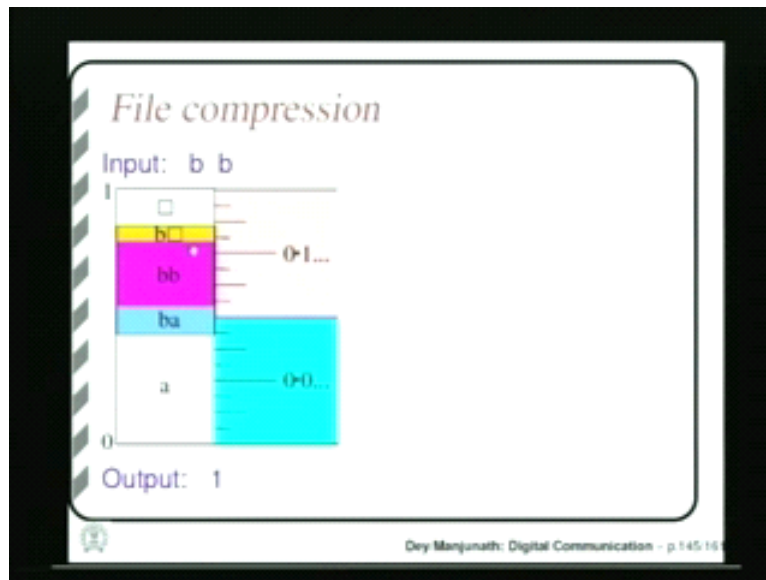
(Refer Slide Time: 32:10)



Now, so we are drawing here the cumulative distribution function yes, we will see now how to do the encoding. So, here is the 0 1 range. Is now we have cumulative distribution function a is less than b and b is than this we have assumed is the ordering we have taken. Now, the distribution function is if you take those probability we can see that it will be basically approximately this much length a is probably 0.425.B is probably 0.4251 is end of file is probability 1.15.

Now, we have received b we know; so we are interested in this jump. So, are these 2 intervals having any bit common any initial bit common that we have to see. Then we can encode that bit that is the idea, but we are seeing that the this number has the first bit 1. Because all these numbers have the first bit 1, all these numbers have first 0 this is basically point five number. So, this bit has first bit 1 this bit has this number has first bit 0. So, there is no common bit between these 2 numbers. So, we cannot decide on the first bit also at this moment after receiving 1 symbol.

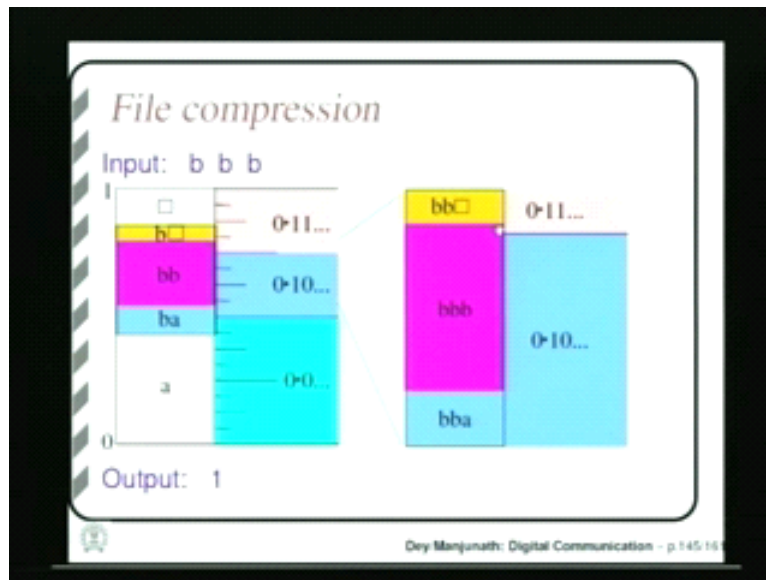
(Refer Slide Time: 33:34)



So, we receive the next symbol then, we are interested in this much this interval the interval has narrowed down to this bb this part. Now, we have to see whether these 2 numbers have any bit common. Now, we see that they have 1 bit common point 1 both start with point 1. So, we can output 1 as output because 1 is certainly the first bit of the encoded bit stream because, the final number code will be somewhere in between this it will have first bit 1.

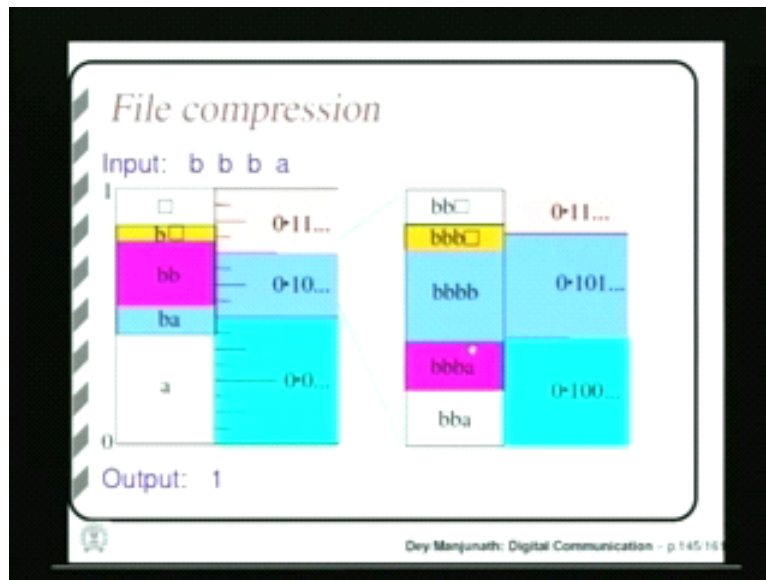
Similarly, we go on doing it. Now, are there any if further bits common to these 2 no because second bit of this is 1 third bit of second bit of this is zero. So, we cannot encode the second bit.

(Refer Slide Time: 34:24)



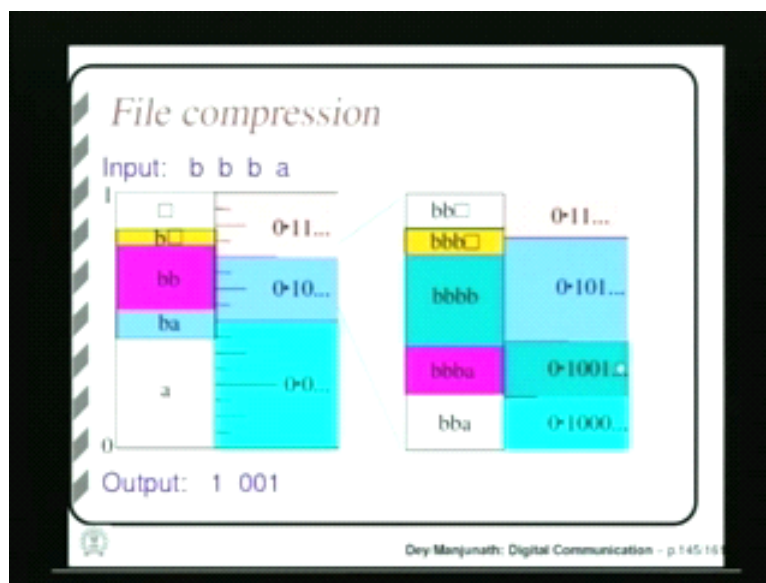
So, we receive another symbol now so we are magnifying this for convenience of viewing now, we have basically this much magnified to this. So, it is 0.11 here 0.10 here. So, now we receive another symbol this probability we are interested in this jump we have received bbb. So, we are interested in this much because they still have no second bit, not common. Second bit of this interval is 1 second bit of this interval this number is 0. So, we cannot encode the second bit still. We have received three symbols. So, far we have encoded 1 bit.

(Refer Slide Time: 35:02)



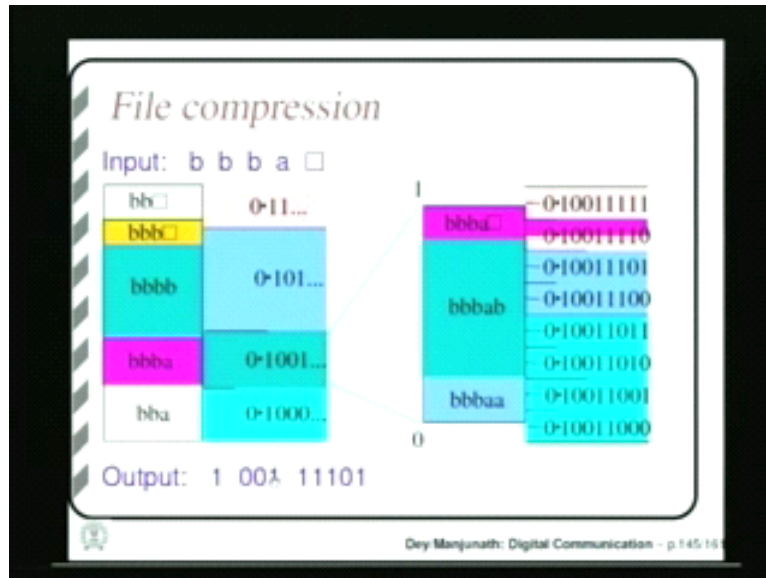
We received the fourth symbol now. Now, we are interested in this interval the interval has narrowed down to this much because we have received a bbba So, now do they have these 2 interval numbers have any bit further bits common? Yes, both of them have 0 at the second bit and also 0 at the third bit.

(Refer Slide Time: 35:30)



So, and also 1 as the fourth bit both these numbers have all 4 of these 4 of these bits common because 0.1001 is the first part of all the numbers in between this range. So, these 2 numbers have the same four bits. So, we can encode 2 further bits.

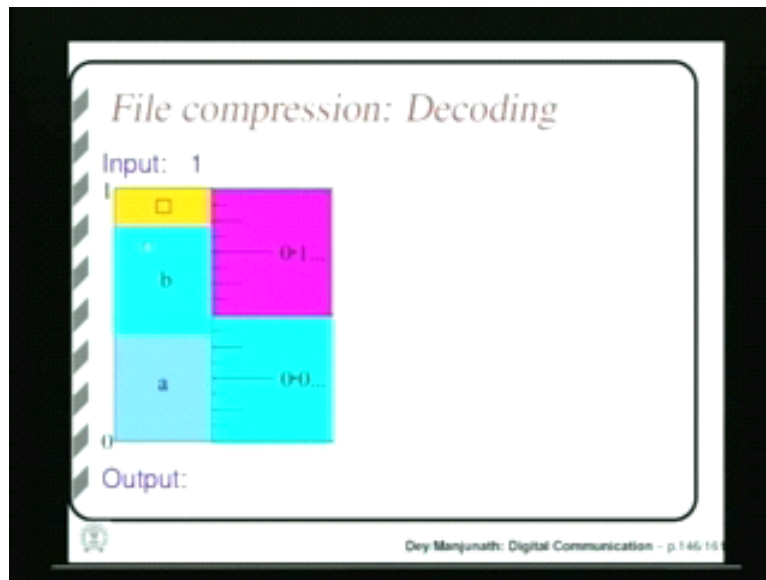
(Refer Slide Time: 35:54)



We go on doing it we have this is the previous pages figure now, we keep doing it we receive again we divide this in to 3 parts bbbaa b and end of file. Now, we receive end of file here. End of file now these 2 intervals do they have any these 2 numbers do they have any common further bits? Yes, they have these further bits 111 and so 11101 End of file 11101 there is something missing here; it has to be extended to some more.

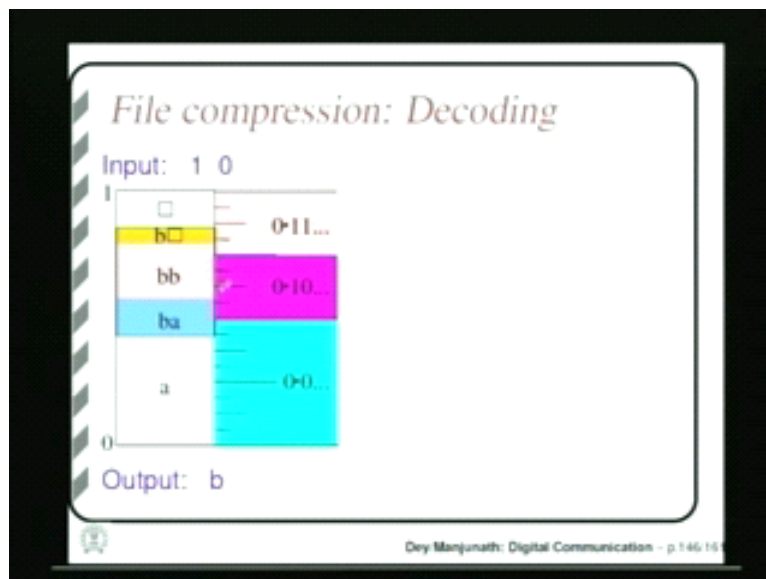
So, we can encode if you see carefully we will see that we can encode further 5 bits. So, this is the encoded this is the code, code word, code string. So, this is the final code string the file ends here.

(Refer Slide Time: 37:11)



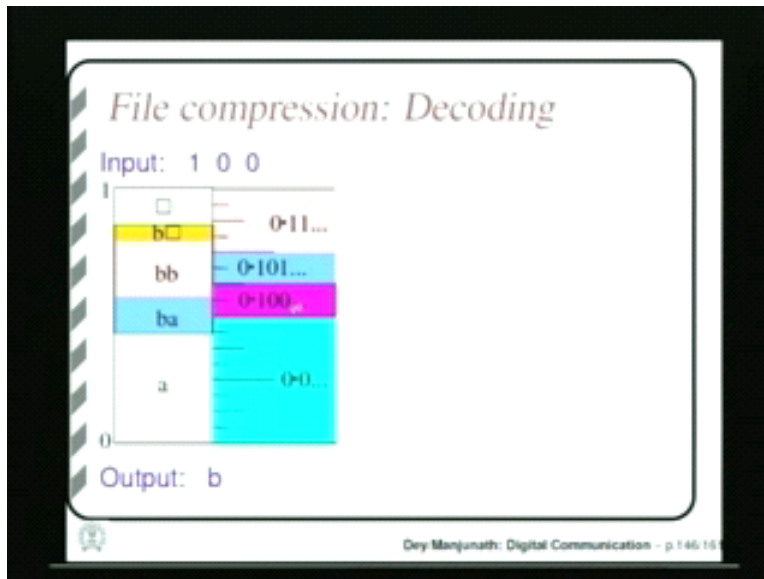
Now, how do you do the decoding? We do exactly similar to encoding we take this and first bit we receive is 1. So, we take this interval. Now, can we see whether the first symbol was end of file or b we cannot see because the number final number code word is going to be some number here, but it may be in this part or in this part. So, we do not now; so we cannot still give any output in the decoding

(Refer Slide Time: 37:37)



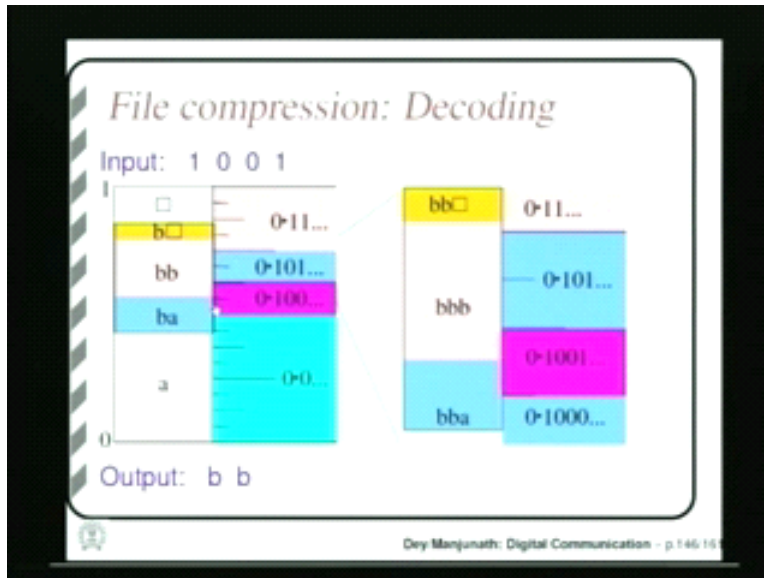
Then we take the second bit 0 so we see this interval. Do this interval lie in between any symbol here? Yes, it is lying in this b first of all this b interval. So, we know that the first symbol must be b, but can we decide on the second symbol no because it may be bb or it may be ba it may be here ba or it may be bb here. So, we do not know the second bit, so we have out we have 1 bit output b.

(Refer Slide Time: 38:10)



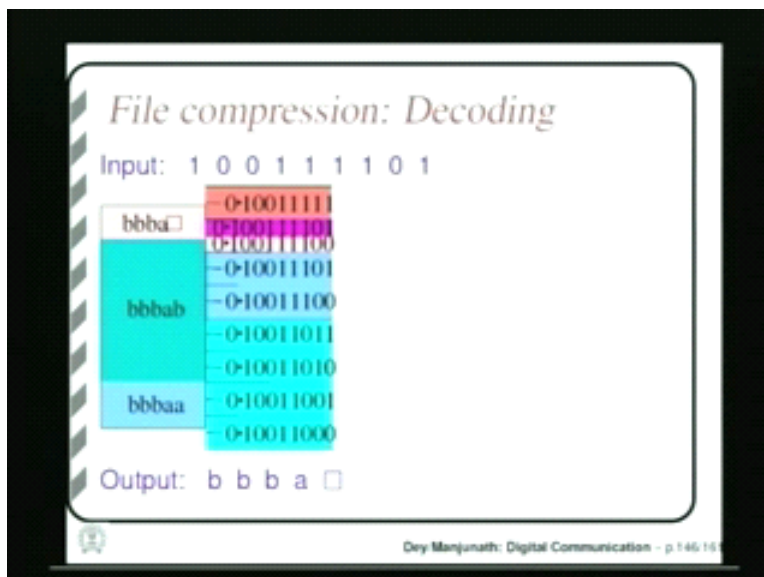
Then we receive 1 0 another bit third bit. Now, this again we cannot decode the second bit the second symbol because it may be here bb or it may be here ba.

(Refer Slide Time: 38:23)



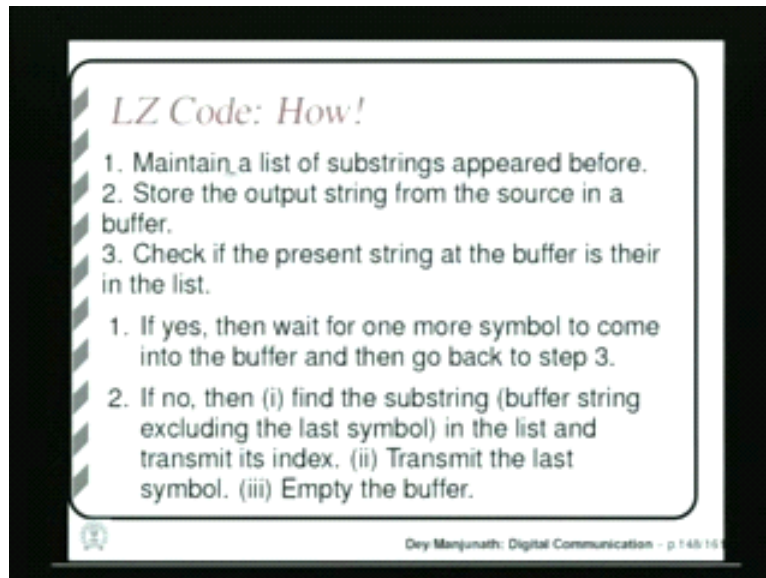
So, this is another bit we are expanding this here. Now, we are receiving another bit this 0. Now, still yes we can say that it is now in this bb bb is divided now here. So, bb starts from here and here. So, it is inside that so we can decode the second symbol also. So, we keep doing that and we see that as we receive bits we can still we can also decode. So, final decoded output will be this bbba end of file.

(Refer Slide Time: 38:54)



So, we can do encoding and decoding in this sequential manner that is the most interesting part in arithmetic coding. So, we have basically used the same principle as Shannon-Fano-Elias code it is basically Shannon-Fano-Elias coding, but in a sequential manner. We have seen how to compute the probability in a sequential manner and we have used that technique to do arithmetic coding. So, arithmetic coding is basically sequential encoding of a encoding in Shannon-Fano-Elias coding.

(Refer Slide Time: 39:37)



So, we now go to another coding technique called Lempel-Ziv code. So, this is a tabular coding technique. What is done is at any time a list of substrings previously you have received is maintained. So, we will first we receive 1 we have not received 1 before. So, this is new substring we have received. So, we keep that in a list. So, we enter 1 as 1 element of the list than we receive suppose, 10 then 11001 something like that. So, you have received second bit 1 then we see that 1 is already there. So, we have not received any new substring so far. So, 1 0 we have received next 1 then 0; so we combine 1 0.

So, 1 0 is there in the list it is not there. So, we enter that new substring in the list we keep entering new substrings which is not there in the list. Now, whenever we enter something that input is discarded we receive again fresh. So, that is the, that is how we

construct the list. So, we maintain a list of substrings appeared before store the output string from the source in a buffer.

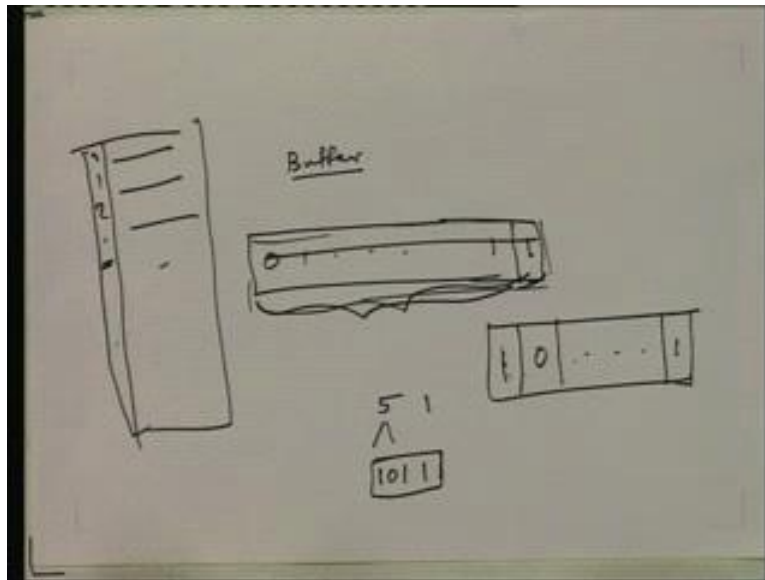
So, when we encode when we enter a substring in a list we also we also discard that we actually we will encode that and discard that from the buffer. Then we will receive fresh bits and keep it in the buffer and we will see whether, we will keep entering receiving more and more bits and to check at the same time whether it is there in the list. If it is not there whenever it is not there we will encode that and enter that substring in the list. We will see with an example how exactly we do it.

So, check if the present string at the buffer is there in the list spelling is wrong check if the present string at the buffer is there in the list. If yes, then wait for 1 more symbol to come into the buffer and then go back to step 3. So, whenever we have received a string which is there in the list. So, if it is there then we will receive another bit if it is still there in the list then an extra the longer substring then we will take another bit.

So, we will continue this still we receive a substring we get a substring in the buffer; which is not there in the list. So, then what will you do? So, if we have received a substring in the buffer which is not there in the list. Then first we will find the substring in the list that is we excluding 1 bit we know that the rest of the substring is their in the list. Because we have not we have got that substring in the list then we have received another bit and then that is not there in the substring.

So, if we remove the last bit we have received we know that that substring is there in the list. So, we see which element in the list is that substring. So, if you remove 1 bit from the buffer now you get a substring in the list you see the index of that. So, you transmit that index that is the part of the encoding that is part of the code, code word.

(Refer Slide Time: 42:59)



So, what is done so, in the buffer we have fixed many bits. The last bit say it is 1 then 0101 something. So, we know that this much is there in the list that is why we received another bit we take took another bit and considered this extra longer substring. Now, this bit we consider later first this sequence is there in the list. There is a list here, where all these codes words are listed and there is a serial number here 0 1 2 this way. So, we see the index of that. So, that index suppose it is 5 the 5 is part of the code word encoded code word and then we add 1 to that so 1.

So, now five of course, will not be transmitted this way five will be, will have binary of that. So, we will have 101 then 1 so we will transmit this as the code word for this. So, we take the index of this in the list and then add 1 to that that is the code word for us. Okay. So, this is the way we keep doing the encoding then we will once we have encoded we have got this codeword we will empty the buffer this buffer will be emptied.

So, we have a empty buffer now. Now, we keep receiving fresh bits next bit is suppose 1 we give we get this next bit is 0 we check when we receive 1 see whether it is there in the list. It is there, then receive another bit it is their 1 0 is there then receive another bit keep doing it till you receive a sequence which is not there in the list. Then you do the encoding as you did before for the first previous symbol.

(Refer Slide Time: 44:45)

Encoding: Example

Input: 1 0 11 01 010 00 10

000	001	010	011	100	101	110	111
□	1	0	11	01	010	00	10

Output: 1 0,0 01,1 10,1 100,0 010,0 001,0

1. There is no apparent compression!!
2. Reason: The input string is too short and 0, 1 are almost equiprobable
3. There will be compression for long strings with unequal frequency of 0 and 1.

Dey Manjunath: Digital Communication - p. 149-151

So, we will take an example now, we suppose we are receiving input, but before we receive any input we have this list. This is basically the zero'th the first element in that is a null because; we have to have something in the list before starting. So, that is the initial state. We have 1 element here that is the null element. So, we check we take the first bit we have received 1, we see whether 1 is there in this We see that it is not there the only element in the list presently is null.

So, we have got a string substring which is this is in the buffer which is not there in the list. So, what do we do? We first encode the string which is preceding the last bit last bit. So, that is nothing so null. So, we have do not have anything so we do not have anything then, we just take 1 this last bit. So, 1 is output if you have received 1, but we have also entered this last bit whatever is there in the buffer we forgot to mention in the previous part when we are explaining. That when we have this after encoding this new string which is not there in the list this total is entered in the list as a new extra element.

So, that is entered as a element in the table then we receive another bit 0 we check whether 0 is there in the list. We see that it is not there we have only null and one. So, 0 is not there. So, here the output should have been 0 1 because we there was null before 1

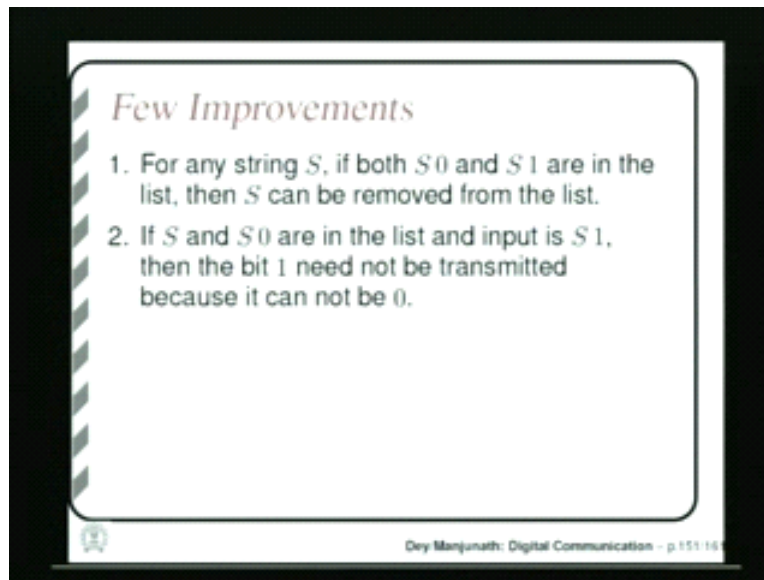
there was nothing actually, that is null. So, 0 1 and then we have received 0. So, 0 is not there in the list so we see that the previous symbol is basically null previous bit is null.

So, null is zero'th so 0 and then the then the bit is 0. So, this is the index 0 and this is the bit 0 this is the extra code word this is the output so far. So, then we have to enter 0 here, so when we enter 0 you have to index it. So, 0 1 2, but now you cannot represent the index using 1 bit. So, you have to go to 2 bits so 00 01 10. So, then we receive 1 we see that 1 is there; so we receive another bit 1 again 1; so, 11 11 is not there is the list. So, we now start encoding it.

So, first bit excluding the last bit is 1. 1 is there in the list that is 0 1 so, 001 is the index of the substring 1 then 1 is added. So, we keep doing that then 01 when you go to the next entry of the list we have to increase the index to 3 bits because, we have 5 symbols 5 elements in the list now, we keep doing that. But, after we do all this we see that there is no apparent compression there are more bits here than here what is the reason.

The reason is that the file is too short we would not possibly get any compression for too short a file like this using Ziv-Lempel coding. So, and another thing is that 0 and 1 are almost equally likely here they are almost with same number. So, we would not get possibly any compression for that reason also. So, if you have a sufficiently long file with unequal distribution of 0 and 1 then we will get compression using this technique. How do you do the decode decoding? So, decoding we are getting the encoded code word.

(Refer Slide Time: 50:18)

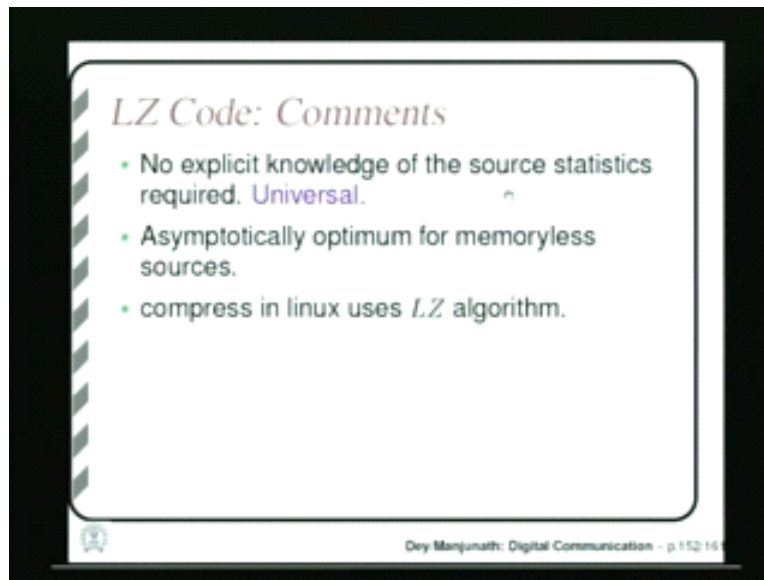


Now, we can do improvement in certain way here for example, if we have 0 and 1 both in the list for example, we had it here that 0 and 1 are both in the list. Then, we know that 0 and 1 are in the list like if S_0 and S_1 are in the list; then S can be removed from the list. So, let us see what can be removed here. So, 11 and 10 are both there so 1 will never be will 1 need not be kept here, so we can put a new element here.

So, we can minimize the size of the list and that way we can have less number of bits for index. So, that will give us a better compression we need not keep say 56 bits. So, if you can remove some elements from the list then we can keep the list smaller and as a result. we need less number of bits to represent the index. So, that will give us better compression. So, we can if S is substring and 0 is the next string 0 is another this 0 is 1 string S_1 is 1 string both are there then, S substring need not be kept in the list; that can be removed.

So, if S_0 and S are both there and we receive S_1 now we know the last bit need not be transmitted because, S_0 is there. So, last bit cannot be 0 otherwise it would be found in the list. So, last bit cannot, need not be transmitted if S with 0 and S alone are both there in the list. And we can limit the size of the list by removing any element which is not used for a long time; so, which is the least used element in the list.

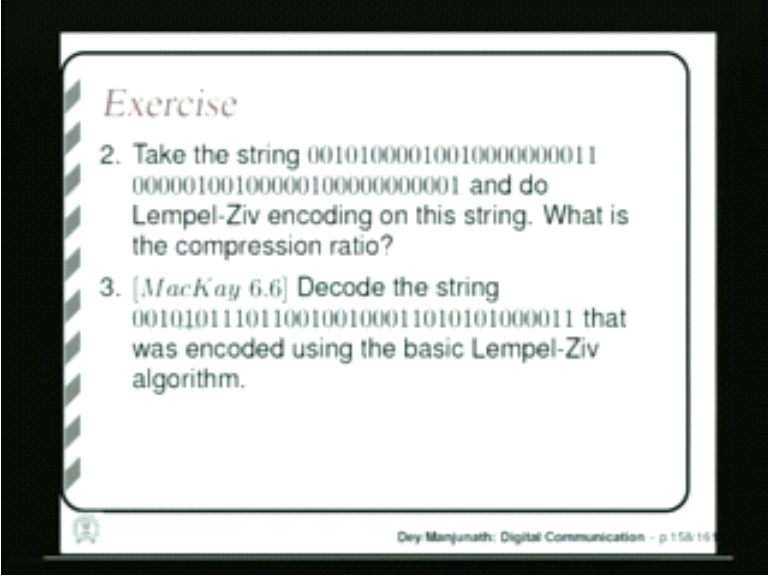
(Refer Slide Time: 52:12)



So, there are some comments is a universal coding we can use it for any source statistics it will adapt itself automatically. Asymptotically optimum for memory less source and also Linux compressed in Linux which we give an example, as example of a lossless coding uses LZ coding. So, we have done we have seen how to do iterative encoding of Shannon-Fano-Elias coding technique that is arithmetic coding. We have taken an example of arithmetic encoding of a file using Laplace model of adapting the probability statistics.

So, and we have also take done LZ coding which is not mentioned here, but we have a done a tabular coding technique which is LZ coding where we do not need the probability explicitly. It will automatically adapt the probability.

(Refer Slide Time: 53:06)



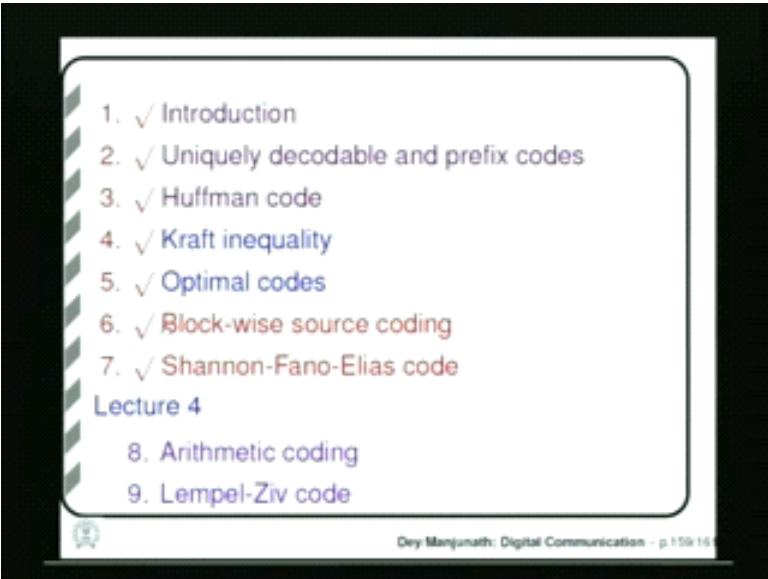
Exercise

2. Take the string 0010100001001000000001100000100100000100000000001 and do Lempel-Ziv encoding on this string. What is the compression ratio?
3. [Mackay 6.6] Decode the string 00101011101100100100011010101000011 that was encoded using the basic Lempel-Ziv algorithm.

Dey Manjunath: Digital Communication - p. 158/160

These are some exercises; this is a string the exercise is to do Ziv-Lempel coding on this string. Then this is a string this is Mackay's book 6.6 exercise this is encoded string which basic Ziv-Lempel coded coding which we have just described. You do decoding for this.

(Refer Slide Time: 53:24)



1. ✓ Introduction
2. ✓ Uniquely decodable and prefix codes
3. ✓ Huffman code
4. ✓ Kraft inequality
5. ✓ Optimal codes
6. ✓ Block-wise source coding
7. ✓ Shannon-Fano-Elias code

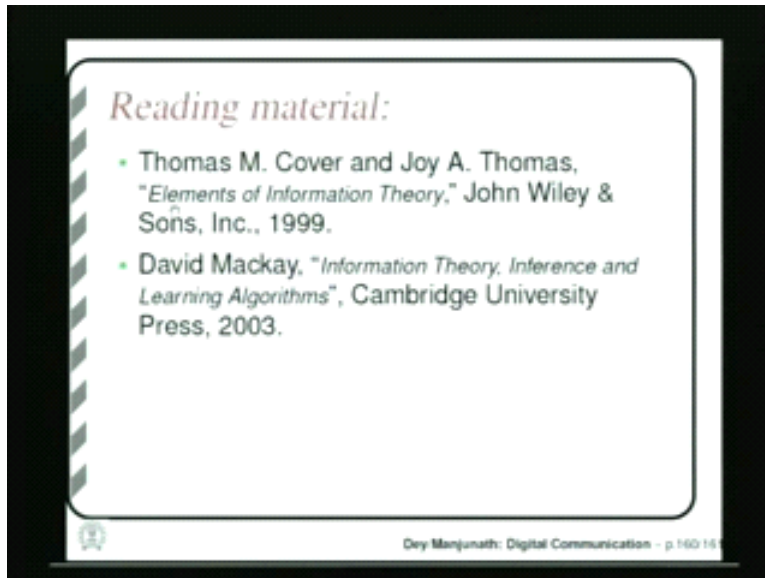
Lecture 4

8. Arithmetic coding
9. Lempel-Ziv code

Dey Manjunath: Digital Communication - p. 159/160

So, we have done all this, so we have completed arithmetic coding in this class and we have done another source coding technique Liv-Ziv a Lempel-Ziv coding. This is the end of the source coding. We will have new topics in the next class.

(Refer Slide Time: 53:42)



So, again this is these 2 books are very good books. There are other books on information theory very popular books, but this book is a very lucidly written book and good for undergraduate. Information Theory Inference and Learning Algorithms, it is available in Indian edition with low cost. So, I will encourage you to read this book and solve the exercise there. Thank you. This is the end of the source coding technique class.