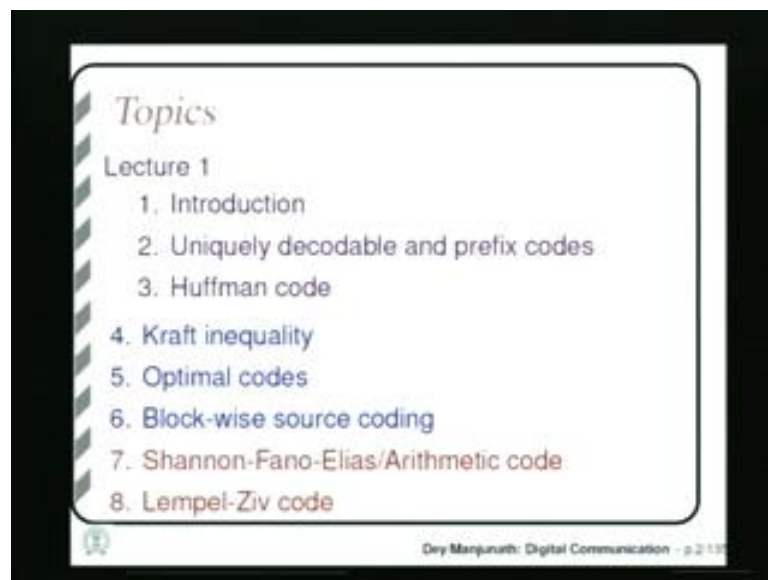


Digital Communication
Prof. Bikash Kumar Dey
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Lecture - 26
Source Coding (Part – 1)

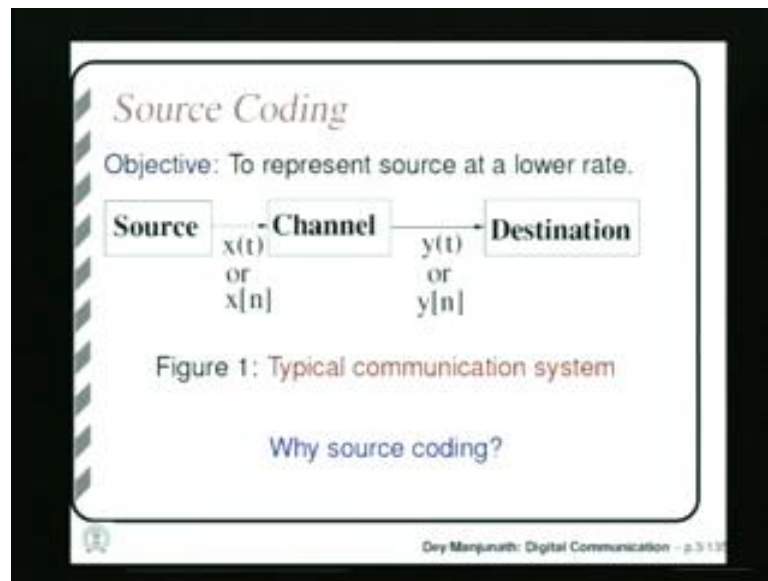
Hello everyone, we will start a new module today and that will be on source coding. So, we will cover these topics under source coding today out of these topics, we will cover the following 3 topics in this class.

(Refer Slide Time: 01:06)



First we will have introduction. So, there we will see why we need source coding? And why it is possible to do source coding for most of the sources. And then we will describes of the desirable properties of source codes and in particular we will discuss what is called uniquely decodable codes and prefix codes. And then we will discuss 1 particular very popular source coding technique called Huffman code. This source coding technique is very important source coding technique and we will see why later. And rest of the topics will be covered in later classes.

(Refer Slide Time: 01:50)



So, let us see why we need source coding. So, this is a typical communication system it involves 1 source which generates the message or information that is to be transmitted through a channel to the destination. It generates a continuous time signal or a discrete time signal and the destination receives a distorted version of the signal y_t or y_n . And the channel distorts the signal in some way or the other and. So, why do you need source coding.

(Refer Slide Time: 02:34)



Because the signal source generates may have redundancy or undesired or unimportant extra information that, the destination does not require, For example, telephone quality

speech can be compressed without compromising much on quality to as little as 1 to 2 Kbps better rate. So, even if, we do not consider all the nonsense we talk there is lot of redundancy in the representation of the signal itself.

(Refer Slide Time: 03:07)

$x(t)/x[n]$ may have **redundancy** or undesired/unimportant extra information

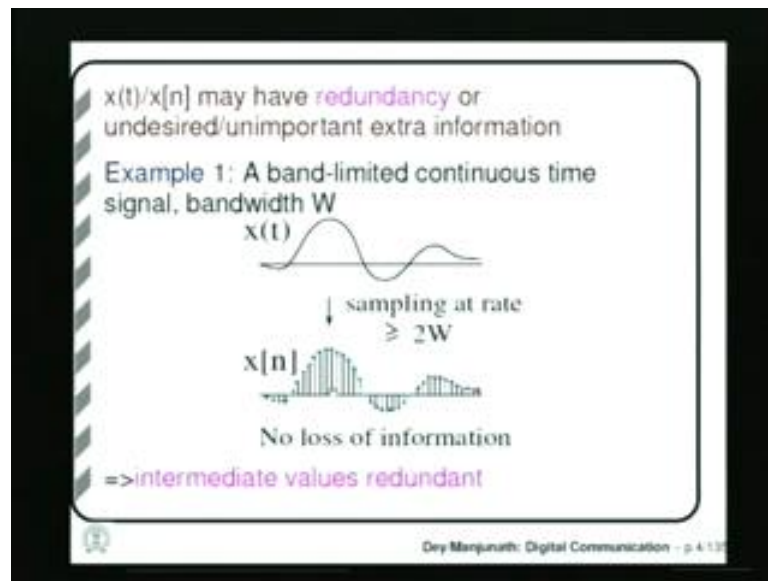
Audio Demonstration:
(Courtesy Prof. Preeti Rao,
Dept. of EE, IIT-Bombay)

1. Original audio recording:
2. Compressed audio: 1.5 Kbps

Dey Manjunath: Digital Communication - p.4.13

So, here is particular example, where you will hear a particular original audio recording and then a compressed signal which will be compressed to 1.5 Kbps. So, here is the original signal (refer time: 03:23) So, we hear the same signal compressed to 1.5 Kbps now (refer time: 03:30) as you can see there is not much difference at least there is no perceptible differ difference between the 2 signals. That means all the information that the original recording contained is more or less contained in the compressed version as well.

(Refer Slide Time: 03:52)



So, here is a particular example, of how 1 can do source coding. This is a very typical example, the example is of sampling. So, if you have a band limited continuous time signal of bandwidth W and if we sample that signal at sampling that greater than equal 2 times W that is the Nyquist rate then we know that we can recover thus continuous time signal back from the sample signal.

So; that means; that there is no loss of information when we sampled this continuous time signal at a rate greater than equal to 1 times W . And that in turn means that all the intermediate values between the samples which we have discarded did not really contain any extra information over what these samples contained. So, inter values intermediate values are ideally redundant.

(Refer Slide Time: 04:57)

• The channel may not have the "capacity" to communicate at the source information rate
=> need to represent source at a lower rate with some loss of information

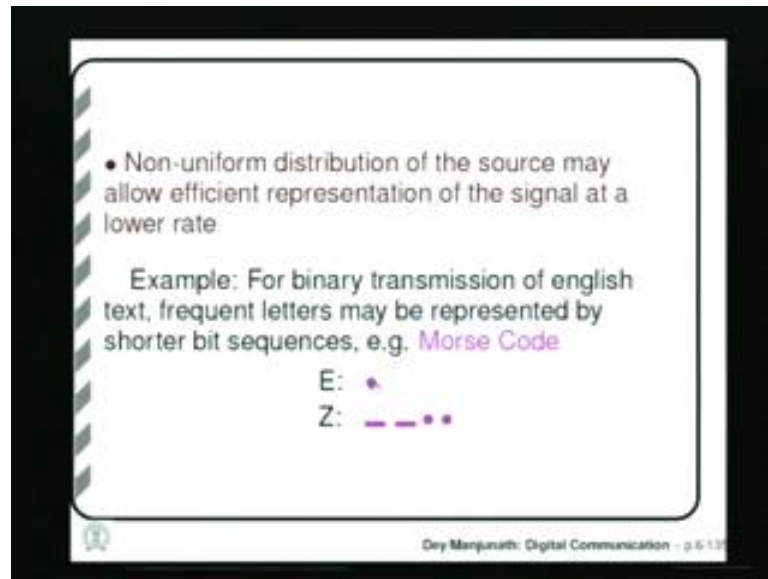
Example: A real valued sample has infinite information. It needs to be **quantized** before transmission through a discrete memoryless channel (DMC)

Dey Manjunath: Digital Communication - p.5.13

Another the reason why we may need to do source coding is that, the channel through which we are going to transmit the data may not have the capacity to communicate at the same rate at which the source is generating the information. So, in such a situation we may need to compress the source with some loss of information. So, we have already seen 2 types of source coding. 1 is 1 in which there was no loss of information and here is another where there is some loss of information and here is an example, of source coding with some loss of information.

If, you have a real value sample signal every sample contains infinite information because the precision of the sample is infinite. So, to transmit this, these samples. So, any finite capacity channel we need to quantize the samples before transmission. So, that quantization will involve some loss of information.

(Refer Slide Time: 06:12)



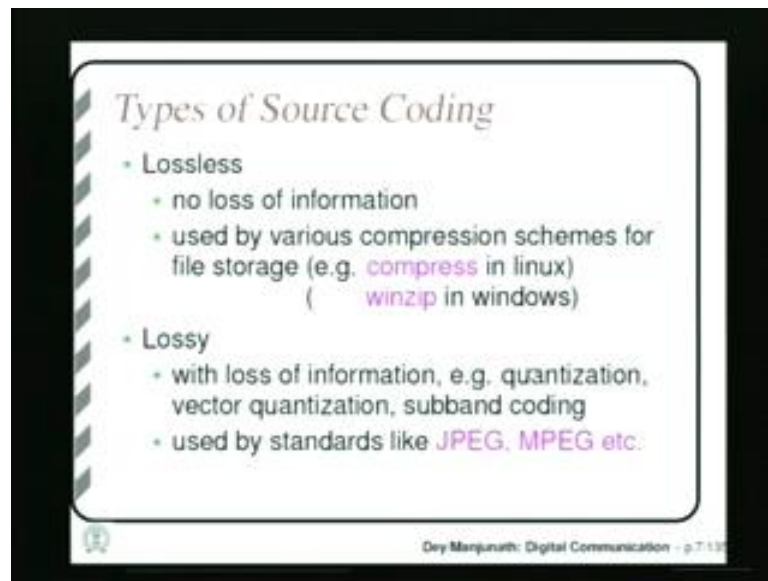
Another reason why the source another way the source signal may contain redundancy is through non uniform distribution of the source. The symbols source generates may have non uniform distribution and as a result the representation of the source may be improved. For example in a typical English text, we know that some letters come very frequently and some letters do not come so, frequently.

So, when you try to represent the English text every, English letter using binary sequences it will be a good idea to represent the frequent letters using smaller number of bits and rare letters using longer bit sequences. And this is exactly what is done in Morse Code, which is a very traditional code used for telegraphic communication you might have seen in movies or even in labs a small instrument through which telegraphic information is transmitted by pressing a key in different ways.

So, a very short small sound of dot and combination of dots and longer sounds of dashes actually communicate the letters. So, every letter in Morse code is represented by a sequence of dots and dashes. So, in Morse code the most frequent English letter E is represented wisely by a single dot and not so, frequent. In fact, very rare letter Z is represented by longer sequence of dots and dashes that is 2 dashes and 2 dots.

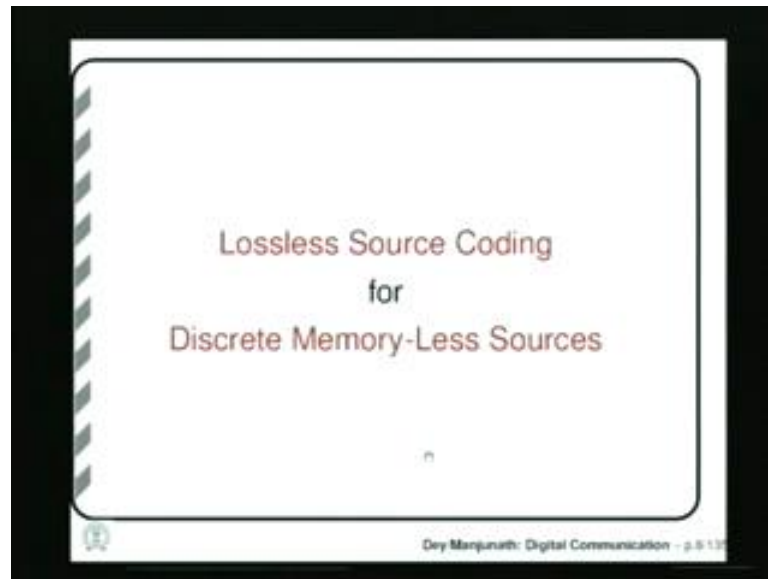
So, this is very traditional example of this type of source coding where the source coding is possible because of the non uniform distribution of the symbols generated by the source.

(Refer Slide Time: 08:31)



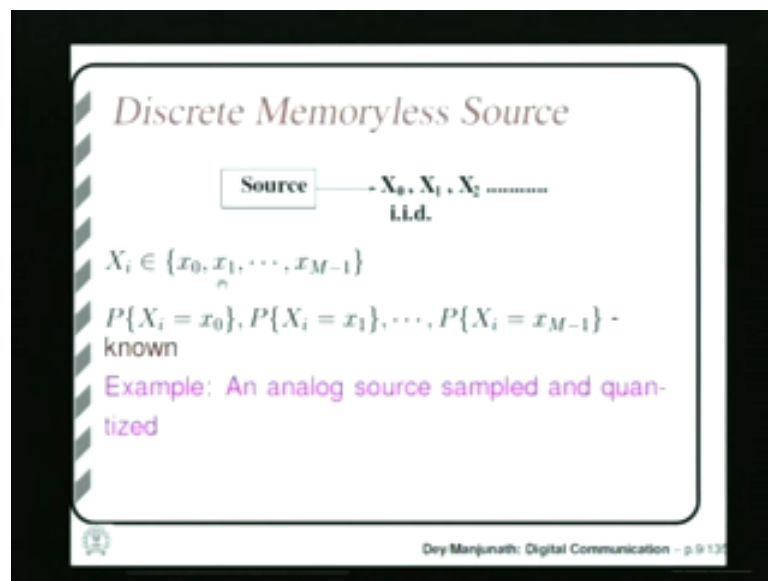
So, we have seen two types of source coding 1 is lossless where there is no loss of information due to coding and this is used by a various file compression techniques in computers for example, compress in Linux or WinZip in windows and then another type of source coding called Lossy source coding which involves loss of information. And we have seen. In fact, that quantization is a is an example of such source coding and vector quantization is a generalization of source coding will not discuss this technique in detail here and subband coding is another very popular source coding technique. And all these techniques are used these Lossy source coding technique are used in standards like JPEG, MPEG etcetera.

(Refer Slide Time: 09:28)



From now on we will consider, Lossless source coding for discrete memory less sources. So, we will see what discrete memory less source means:

(Refer Slide Time: 09:39)



A discrete memory less source generates these symbols in discrete times X_0, X_1, X_2, \dots each of them is a random variable and each of them take values from this set. So, M possible values and each of them has some probability. And we assume from now on the these probabilities are known to us while designing source code for this source this probabilities will be assumed to be known.

A typical example, of discrete memory less source is an analog source sampled and quantized sample to make a discrete time. So, it is discrete and quantized to make this set finite because each random variable should take finite number of values And it is memory less means that each symbol here are uncorrelated from each other. So, to make a sample signal memory less we actually need to sample the analog signal at a lower rate.

So, if you sample analog signal at exactly the Nyquist rate we will get a memory less source. If we sample at the higher rate we will not get a memory less source memory less sampled output.

(Refer Slide Time: 11:08)

If $M \leq 2^b$, then we can represent the symbols by binary fixed length codes as:

symbol x_i	codes $C(x_i)$	length l_i	probability P_i
x_0	0 ... 000	$l_0 = b$	P_0
x_1	0 ... 001	$l_1 = b$	P_1
x_2	0 ... 010	$l_2 = b$	P_2
\vdots	\vdots	\vdots	\vdots

This will require b -bits per symbol

Dey Manjunath: Digital Communication - p.10/13

So, let us see without doing source coding what is an obvious way of coding the discrete memory less source? So, every symbol takes M values and let us say that, M is less than or equal to 2 to the power b. In fact, we will assume that M is greater than 2 to the power b minus 1, but less equal to 2 to the power b that is to represents this symbol using binary streams with equal length binary streams we will need b at least b bits.

So, we have all this symbols as possible outcomes of the source and each symbol needs to be represented by a fixed length code. So, if we code using fixed length code we will need b number of bits for each of the symbols even though they have different probabilities we are coding them with equal number of bits. This will require b number of bits per symbol whereas, we can do better.

(Refer Slide Time: 12:13)

Something better possible!

Example:

symbol:	x_0	x_1	x_2	x_3
probability:	0.5	0.25	0.125	0.125
fixed length:	00	01	10	11
variable length:	0	10	110	111

Average length for fixed length code = 2,
for variable length code = 1.75

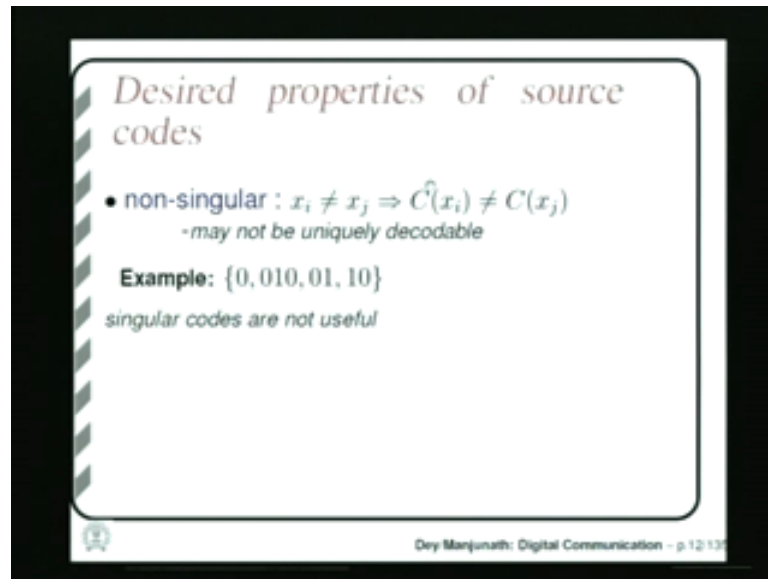
Entropy, $H(x) = 1.75$

Dey Manjunath: Digital Communication - p.11:13

If we have for example, 4 symbols with these probabilities these unequal probabilities. If we want to do fixed length coding we will need 2 bits per symbol because there are 4 symbols here. So, average number of bits required per symbol is 2 bits using fixed length code. Whereas, if you use variable length code like this then what is the expected length of the code. It will be 1 times 0.5 plus 2 times 0.25 plus 3 times 0.125 plus 3 times 0.125 and that will give us 1.75 bits per symbol.

So, here we have been able to represent the source with less number of bits on average than using a fixed length code. And, in fact, this is the best for this random variable we can do because entropy of the source itself is 1.75. And by a source coding theorem of Shannon this is the minimum number of bits required to code the random variable.

(Refer Slide Time: 13:33)



Desired properties of source codes

- non-singular : $x_i \neq x_j \Rightarrow \hat{C}(x_i) \neq C(x_j)$
-may not be uniquely decodable

Example: {0, 010, 01, 10}

singular codes are not useful

Dey Manjunath: Digital Communication - p.12/13

So, here some desired properties of source codes. Obviously 2 different symbols should not be coded into a same code because, if done that way if we receive a code word we will not be able to decipher the we will not be able to decode the code properly at the destination because if we receive that, code word we will not know which 1 of these 2 were transmitted because code for both the symbols are same. So, this code cannot be decoded uniquely at the destination.

For example, so this is that kind of code will be called singular and the code where every distinct symbol is encoded into different code sequences is a called non-singular code. So, for example, this code is a non-singular code because all these four codes code words are different, but even if we have these even, if we have a non-singular code will not we may not be able to decode a code uniquely. We will see later an example of that. So, for now we will note that singular codes are not useful; that means, we should have distinct code word for each different symbol.

(Refer Slide Time: 15:11)

Desired properties of source codes

- uniquely decodable: For any two finite sequences
 $x_1x_2\dots x_m \neq x'_1x'_2\dots x'_n$
 $\Rightarrow C(x_1)C(x_2)\dots C(x_m) \neq C(x'_1)C(x'_2)\dots C(x'_n)$
-decoder may need to observe future bits

Non-example: {0, 010, 01, 10}
01010 = 01 010 = 010 10 = 01 0 10

Example: {10, 00, 11, 110}

Dey Manjunath: Digital Communication - p.12.137

So, just now we say that even, if we have a non-singular code it may not be uniquely decodable. So, what can go wrong what can wrong is this that, if we now we will be transmitting these symbols 1 after another. So, if we transmit M consecutive symbols. So, and the total codes sequence for that will be this. If we receive this code sequence it should not be code sequence for any other symbol sequence that means: if we have 2 distinct symbol sequences there are code sequence such should not be there. Code sequences should not be same.

So, that the decoder can decode this whole string uniquely. So, are there non-singular codes which not uniquely decodable yes in fact, there are we have, in fact seen an example in the last slide will see again this code 001 001 and 10 this code is; obviously, non-singular because all the code words are different, but if we receive this string this string can be decoded in many different ways.

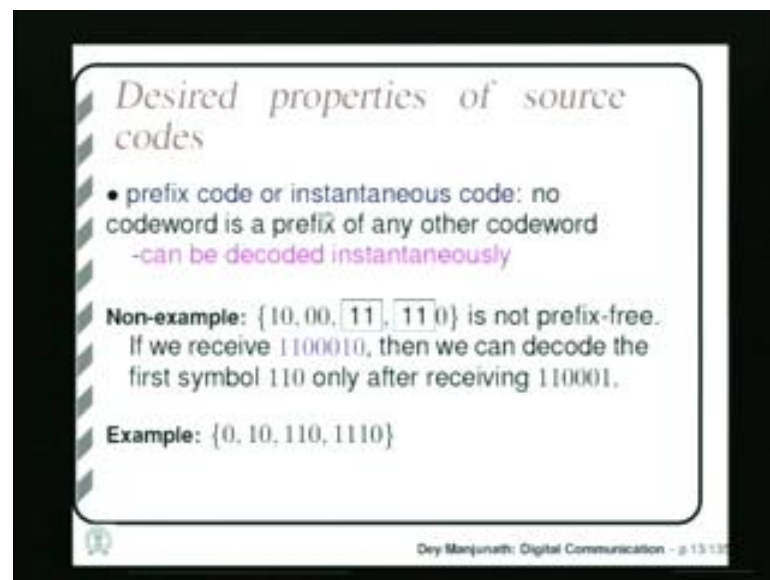
If we break this string into 01 and then 010 then we will think that, we have received this symbol and then this symbol, but if we break this as 010 and then 10 then we will think that we have received this 1 first and then we have received this 1. On the other have we can also break this string into 0 1 0 and 10. So, we will think that we are received this, this and this symbol. So, a non-singular code need not be uniquely decodable.

So, this is a non-example not an example of uniquely decodable code whereas, this here is an uniquely decodable code. We will not check the this is uniquely decodable there are

ways of checking that, but we will not go into that now. So, even for uniquely decodable codes we will not be able to decode the received sequence instantaneously meaning by whenever, we have received a code word we may not be able to decode that, code word instantaneously we may need to wait for some for longer time to see what are the future bits we receive and then we will be able to decode the whole sequence.

So, we need to wait for future symbols before we can decode a present symbol that is the problem with uniquely decodable codes in general. So, we need to be able to decode instantaneously we need some more properties and that is here.

(Refer Slide Time: 18:14)



We need prefix codes meaning by any code should not be any code word should not be a prefix of another code word, if that condition is satisfied the code can be decoded instantaneously. So, here is a non-example we have seen this example before this is a uniquely decodable, but this is not a prefix free code prefix codes are also called prefix free codes. This is a not prefix free code because this code word 11 is epi adding as a prefix of this code word.

So, if we receive this stream for example, then this stream is 1101 code word then 00 another code word and then 10 another code word. So, the 3 code words, but after we have received 110 we will not be able to decode this immediately because after receiving 110 we may think that this may be 11 and then 0 is starting of another code word this 1. So, we do not know whether it is 11 or 11 and 00 or 110 and future bits.

So, we have to wait longer to decide on that. So, in fact we will have to wait till all the 6 bits before we can decode the first symbol for this code. So, we see that this code cannot be decoded instantaneously. So, here it is prefix code and for prefix codes we can decode instantaneously because whenever, we have received a code word we can decode that every time we check we receive first 1 then 1 then we received 0 then we know that, that is code word. So, we can decode that because this cannot be part of this cannot be prefix of another code word.

So, this is surely the code word we have received and all the future bits can be decoded fresh later. So, the prefix codes are the best among these 3 classes all these are desired properties the prefix code is uniquely decodable. So, it is also non-singular.

(Refer Slide Time: 20:40)

Example : Classes of codes

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	10	111

01010 = 01 010 = 010 10 = 01 0 10

Dey Manjunath: Digital Communication - p.14.137

So, here is a summary of all the classes of codes we have considered. Singular; obviously, this is the most trivial example of a singular code all the code words are same non-singular, but not uniquely decodable. We have seen this particular example before these non-singular, but this is not uniquely decodable because we can combine a few code words and then we will get a string which can be decoded as string of 2 different strings of symbols. And this is uniquely decodable, but not a prefix code because this 11 is appearing here.

So, this is the example, this example where this string could not be decoded uniquely because it could be broken into streams of code words in many different ways. So, it

could not be decoded uniquely and this code is uniquely decodable, but it is not a prefix code because this code word is appearing as the prefix of this code word. and this code is prefix code.

(Refer Slide Time: 21:51)

The slide contains the following text:

Exercise

3. [MacKay 5.19] Is the code {00, 11, 0101, 111, 1010, 100100, 0110} uniquely decodable?

Answer: No. Consider the sequence: 111010111.
It can be decoded as either

11 | 1010 | 111
or
111 | 0101 | 11.

At the bottom right of the slide, it says: Dey Manjunath: Digital Communication - p.15.117

So, here is exercise is this code uniquely decodable? So, when we try to answer this question we have to see whether we can combine several code words which can be also broken into code words in a different way. So, if we try this way then we will see that, in fact, there are such streams which can be broken into code words in different ways. For example this 1, if we take this stream, this stream can be this bit stream can be broken into code words like this 11 is a code word then 1010 is code word and then 111 is also a code word.

Whereas, this can also we broken into code words in this way 111 and then 0101 is here and then 11 here. So, this stream can be broken into code words in 2 different ways; that means; 2 different symbols streams can be encoded into the same codes stream. So, this code is not uniquely decodable.

(Refer Slide Time: 23:06)

Huffman code:

X	Code-word	Probability
3	01	0.25
4	10	0.25
1	11	0.20
2	000	0.15
5	001	0.15

Diagram illustrating the Huffman coding process. The probabilities are arranged in decreasing order: 0.55, 0.45, 0.30, 0.25, 0.20, 0.15. The process shows the combination of 0.25 and 0.25 to form 0.50, then 0.50 and 0.30 to form 0.80, then 0.80 and 0.20 to form 1.00. The final Huffman code words are: 3 (01), 4 (10), 1 (11), 2 (000), 5 (001).

Dey Manjunath: Digital Communication - p.16/137

Now, we come to a very important source coding technique called Huffman code. So, we will see with an example first how to construct a Huffman code. So, if we have a random variable X taking 5 values without loss of generality we will assume that, these 5 values are 1 2 3 4 and 5. If they are named differently you can put those names here.

So, there are these 5 values this random variable can take and we will write the design code words here and these probabilities are known to us beforehand. The 1 comes with probability 0.22 comes with probability 0.15 and so, on. So, the first step in designing Huffman code is to arrange these probabilities in probabilities and. So, the symbols in decreasing order. So, the symbols will be arranged in a in decreasing probability first. So, we have this these probabilities we will arrange these symbols in such a way that, these probabilities are arranged in decreasing order. Here so, this symbols are arranged in such a way that, this probabilities are in decreasing order then we will combine these 2 probabilities add these to probabilities to get 0. and then consider these 2 symbols as a single symbol.

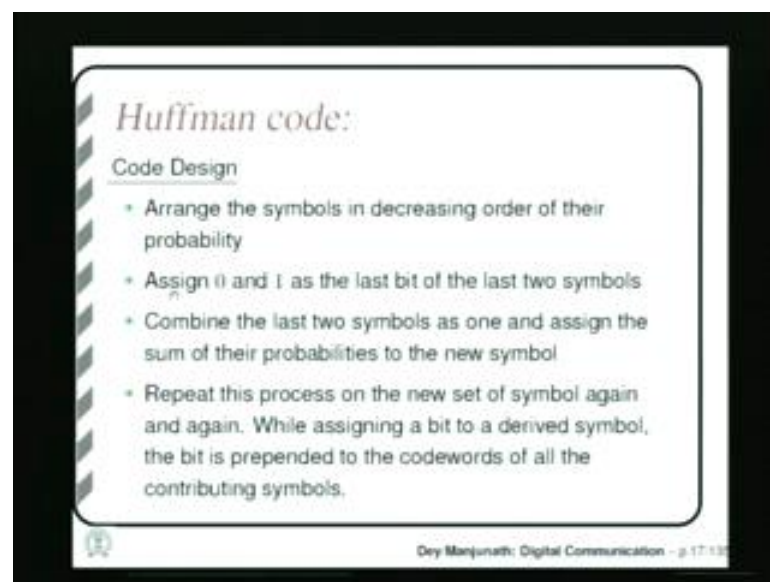
So, next step in the next step we will combine these 2 symbols as 1 and write the probabilities again in decrease order. So, probabilities of the new symbol which is the combination of these 2 will be 0.3 and we already have these 3 probabilities. So, 0.3 and these 3 probabilities will be arranged in decreasing order here. So, point the combination of these 2 symbols goes in the first location where it has the highest probability and then the other probabilities.

And while we combined 2 symbols we will assign 0 to the first symbol and 1 to the second symbol. So, 0 and 1 will be the last 2 bits last bits of these 2 symbols later we will add more and more bits on the left of these 2 bits. So, again we repeat this process now we will combine these 2 symbols into 1 and add their probabilities as the probability of the new symbol. So, what is the probability of the new symbol in this step it is 0.45

So, 0.45 and we add 0 to this symbol and 1 to this symbol. So, 0 to this symbol and this came from 4. So, we have 0 here and 1 to this symbol so, 0 and 1. Now, we repeat this process again we combine these 2 symbols. So, we get the probability 0.55 here and add 0 to this symbol and 1 to symbols of 0 goes to both these symbols. So, we have added 0 to both these symbols and 1 to this symbol which came from here 1 and the last step we combine the last 2 symbols with probability one. So, we add 0 to 0.55 and 0.55 came from these 2 that is these 2 and these 2 came from this 1 and the last 2.

So, we add 0 to this symbol and these 2 symbols and we add 1 to this symbol this probability came from these 2 that is: these 2. So, we add 1 to these 2 symbols. So, this is the final code book we have and this the Huffman code.

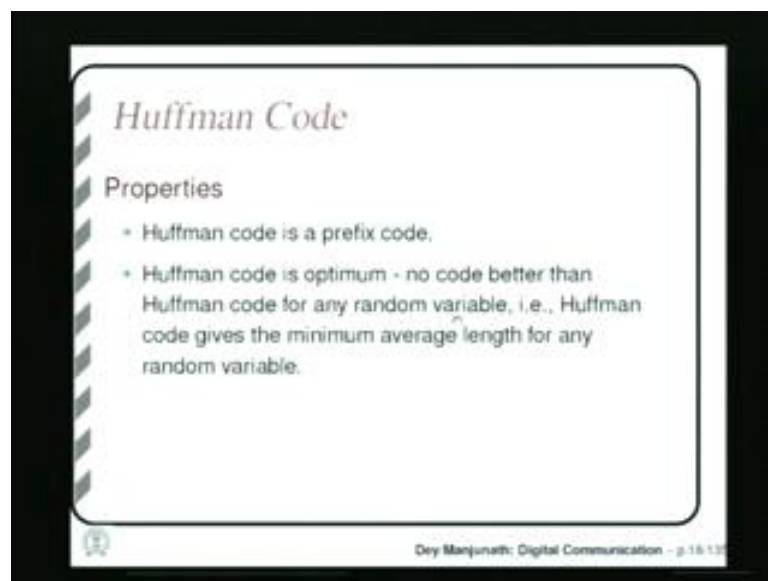
(Refer Slide Time: 27:26)



So, here we summarize the design process of Huffman code first we arrange the symbols in decreasing order of their probability. And there we assign 0 and 1 as the last bit of the last 2 symbols then combine the last 2 symbols as 1 and assign the sum of their probabilities to the new symbol.

So, and then we repeat this process again and again we again arrange these new probabilities in decreasing order and then keep doing this again and again. And while we when we want to add assign a bit to a new symbol that is the derived symbol we actually assign we actually add that bit to the left of all the symbols from which that, new symbol came. We have seen that before, in fact when we tried to when we wanted to assign when we wanted add 0 to this symbol we actually added 0 on the left the all the symbols from which that, new symbol came that is, this 1 and then the last 2. We assigned we added 0 to all these symbols.

(Refer Slide Time: 28:43)



Here are some nice properties of Huffman code. Huffman code is prefix code well we are not going to prove it, but we can see at least from this example, that this code is really a prefix code there is no code word which is a prefix of another code word for example, take 01. 01 is not a code is a not a prefix of any other code word. 10 is not is prefix of prefix of any other code word 11 is also not a prefix of any other code word.

So, this is a prefix code and then Huffman code is optimum meaning by there is no code better then Huffman code for any random variable. So, if we have any discrete random variable we cannot design a better code than Huffman code. So, that means; Huffman code gives the minimum average length for any random variable. So, we will now conclude this class with few we will summarize what we did in this class.

(Refer Slide Time: 29:57)

Highlights: Types of Source Coding

- Lossless
 - no loss of information
 - used by various compression schemes for file storage (e.g. *compress* in linux) (*winzip* in windows)
- Lossy
 - with loss of information, e.g. quantization, vector quantization, subband coding
 - used by standards like *JPEG, MPEG etc.*

Dey Manjunath: Digital Communication - p.20.11

So, we have seen there are 2 types of source coding 1 is Lossless the other is Lossy in the Lossless source coding there is no loss of information. We have seen example for example, we have seen the sampling at greater than Nyquist rate for low pass signals is actually lossless compression. And these are some file compression commands which use Lossless source coding techniques. And then Lossy source coding actually loses some information. For example, quantization vectors quantization and subband coding and this loss Lossy source coding techniques are used by this, popular standards like JPEG, MPEG.

(Refer Slide Time: 30:51)

Highlights: Variable Length Code

Example:

symbol:	x_0	x_1	x_2	x_3
probability:	0.5	0.25	0.125	0.125
fixed length:	00	01	10	11
variable length:	0	10	110	111

Average length for fixed length code = 2,
for variable length code = 1.75
Entropy, $H(x) = 1.75$

Dey Manjunath: Digital Communication - p.21.11

We have seen that instead of doing fixed length coding that is: instead of representing each symbol by a fixed length binary string, if we represent them with variable length code words after considering their corresponding probabilities then we can represent a random variable with less average length than using fixed length code. So, while doing that we have keep in mind that all the frequent symbols like this with 0.5 probability should be represented by shorter sequences and less probable symbols should be represented by longer sequences.

So that, the average length code length will be minimum. So, we have seen that for this particular example of variable length coding we have the average length 1.75 which is better than this.

(Refer Slide Time: 32:03)

Highlights : Classes of codes

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

01010 = 01|010 = 010|10 = 01|0|10

Dey Manjunath: Digital Communication - p.22/137

Then we have seen 3 types of 3 properties which are desirable in source codes Lossless source codes. This is not desirable the singular code where 2 code words will be same in this particular example all the code words are same. So, this code is not desirable at all because we cannot decode the code at the destination. If we receive 0 here we do not know which of these 4 was transmitted. So, a code should be non singular that is all the code words should be different.

That is 1 desirable property another desirable property is it should be uniquely decodable that it need not it should not only been non singular, but it should be also be uniquely decodable. That is when we combine several symbols together we get a long bit stream

that cannot be decoded in 2 different ways. It can be decoded only in 1 way. So, that is uniquely decodable code and then even uniquely decodable codes will not be decodable instantaneously. For example, if we receive 1 1 this symbol we do not know whether we have we have received this symbol or we are received part of this symbol. So, will not be able to decode this code instantaneously.

So, for that we need another condition that is: prefix condition that is no code word should be a prefix of another code word like this here this code word is a prefix of this code word. So, we cannot decode this code instantaneously. So, for that we need this kind of code where, no code word is a prefix of another code word. Then we have seen a popular source coding technique that is Huffman code where we first arrange the symbols in decreasing order of their probability like this and then we combine last 2 probabilities into 1 that is here point 3 and order the new probability with the rest in decreasing order like this.

So, and while doing that we assign 0 to this symbol and 1 to this symbol and then go on doing this again and again. And when we want to add 0 to this we add 0 to this original symbol from which this probability came and 1 to this. Then again we want to add 0 here. So, add 0 to these 2 and 1 to this 1 again do that. So, we get this Huffman code and we have seen that, Huffman code is a prefix code.

For example, this code is prefix and Huffman code is also optimum meaning by we cannot find any better code than Huffman code for any given random variable. So, this is the end of the summary, but we will do some excise now on what we have done.

(Refer Slide Time: 35:22)

Exercise

1. [Cover and Thomas, 5.4] Consider the random variable

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ 0.49 & 0.26 & 0.12 & 0.04 & 0.04 & 0.03 & 0.02 \end{pmatrix}$$

(a) Find a binary Huffman code for X .

(b) Find the expected codelength for this encoding.

(c) What is the minimum length of any fixed length code for this random variable?

Dey Manjunath: Digital Communication - p.26.115

So, this exercise says consider that, consider this random variable this random variable takes these 7 values x_1 to x_7 with these probabilities. Find a binary Huffman code for this random variable x . So, we will do as we did for the other example.

(Refer Slide Time: 35:56)

x_i	codewords	probabilities
x_1	1	0.49
x_2	00	0.26
x_3	011	0.12
x_4	01000	0.04
x_5	01001	0.04
x_6	01010	0.03
x_7	01011	0.02

Expected code length = $0.49 \times 1 + 0.26 \times 2 + 0.12 \times 3 + 0.04 \times 5 + 0.04 \times 5 + 0.03 \times 5 + 0.02 \times 5 = 2.02$

So, we will first write these symbols here x_1 x_2 x_3 we will write their code words here. write these probabilities they are already given in decreasing order 0.49 0.26 0.12 0.04 0.04 0.03 0.02. So, the first step in designing the Huffman code is to combine the last 2 symbols as 1. So, what probability do we get. We get 0.05 that will come before 0.04, but after 0.12. So, we again write the new probabilities in decreasing order 0.49 0.26

0.12 then we get 0.05 then 0.04 0.04. We have got this new probability from these 2 we assign 0 here and 1 here. That means; we assign 0 to this symbol and 1 to this symbol and these probabilities are written as they are next step is to combine these 2 symbols as 1 and we get the probability 0.08.

So, you have got 0.08 from these 2 probabilities we assign 0 and 1 here. So, 0 goes to this symbol and 1 goes to this symbol and all the other probabilities remain as they are, next step is to combine these 2 probabilities 0.08 and 0.05. So, we get point 0.13. We assign 0 here 1 here. So, 0 goes to 0.08 which came from these 2 probabilities that is: these 2 probabilities. So, we add 0 to both these symbols and 1 to this probability which came from these 2.

So, we add 1 to this and this then these probabilities remain as they are. So, we then in the next step combine these 2 probabilities. To get 0.25 0.25 get 0.49 0.26 0.25 0.1. So, 0 goes to these 2 probabilities, these 2 probabilities came from these 3 which came from all these 4 So, 0 goes to this and 1 goes to this symbol then we combine the last 2 probabilities to get 0.51 So, we get 0.51 and 0.49 this probability we have got these 2.

So, we add 0 here and 1 here 0 goes to 0.26 here and 1 goes 0.5 to this 1 and this probability came from all these probabilities. So, 1 will be added 2 all these symbols. Now, as the final step we combine these 2 probabilities we get 1 at 0 here and add 1 here.

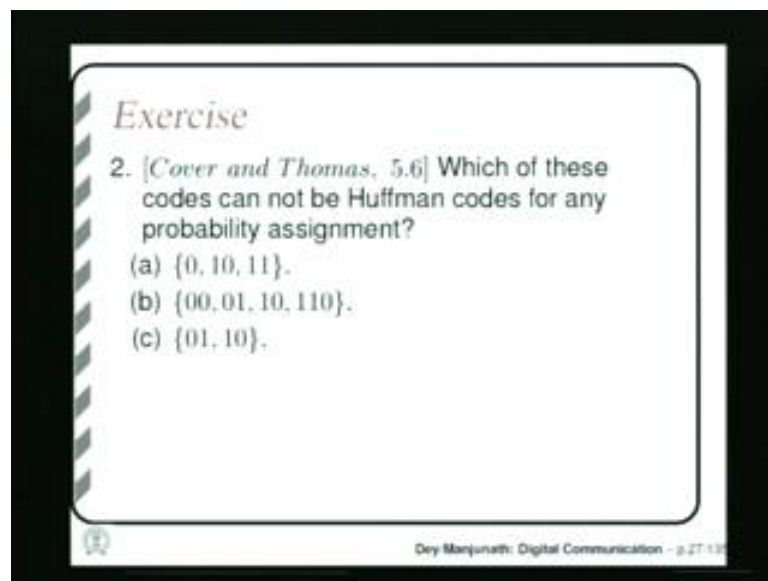
So, 1 goes to 0.5 from 0.51 came from all these probabilities all, but the first 1. So, we add 0 to all these symbols and 120.49 which is the probability of x 1. So, we add 1. So, this a Huffman code for this random variable find the next question is find the expected code length for this encoding. What is the expected code length for this encoding. We have 1 code word of length 1 which probability 0.49. So, expected code length is. So, expected code length is 0.49 times 1 plus 0.26 times code length 2 Plus 0.12 times code length 3 plus 0.04 times code length of x4 is 5 plus again 0.04 times code length 5 plus 0.03 times code length 5 plus 0.0 2 times code length 5 and when we multiply this and add all that terms we get 2.02.

So, this is the expected code length of this Huffman code for this, random variable and as this as Huffman code gives the best code for any random variable we can also rest assured that, 2.02 is the minimum expected code length we can ever get for this random variable. We cannot get any better than 2.02 expected code length. What is the minimum

length of any fixed length code for this random variable. So, if we use a fixed length code for what would be the minimum length for this random variable. We have 7 possible values.

So, to do fixed length encoding for this random variable we will need at least 3 bits. It will 2 bits we can encode only four symbols with a 3 bits we can encode 8 symbols. So, here the number of symbols is 7. So, we have to have 3 bits to encode this.

(Refer Slide Time: 45:05)



Next exercise, which of these codes cannot be Huffman codes for any probability assignment. So, to see why to go to solve this problem we need to observe certain facts in the design process of Huffman code. So, what did you do when we try to design Huffman code. We first ordered the probabilities and decreasing order and then combine the least 2 probabilities into 1.

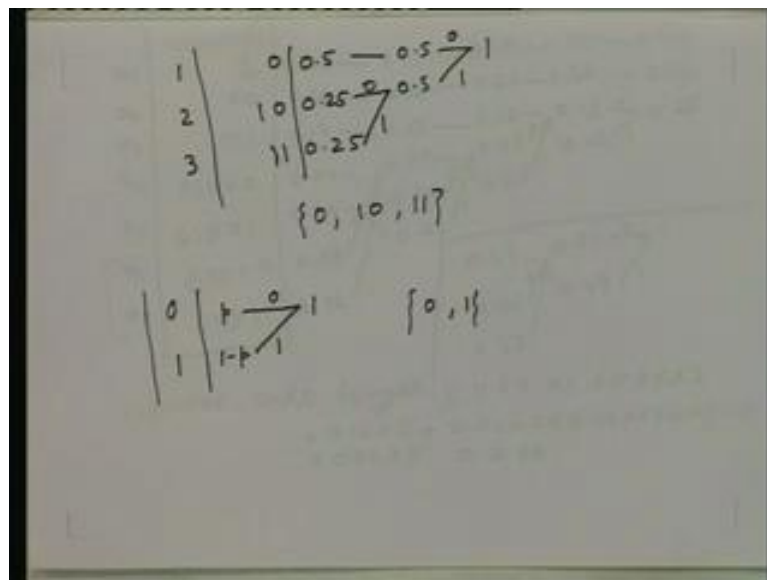
So, and combined these 2 symbols as 1 new symbol from this step onwards. So, from this step onwards these 2 symbols were considered a single symbol. So, the code the symbols having the least probabilities 2 symbols last 2 symbols will have the same bits except the last 2 bits that is for sure because in this step we have added 0 and 1 to different bits to these 2 symbols from then on these 2 symbols are considered as 1 symbol. So, whatever bit is added from this step onwards to this probability was added to both this symbols.

So, as we can see except for the first 2 bits except for the first 2 bits first bits the other bits of these two symbols are same. So, in other words and also these 2 symbols will have the maximum code length because they have the smallest probabilities. So, we can say that the maximum code length will be therefore, at least 2 symbols and 2 such symbols will have these 2 code words differing only in the last bits. All the other bits of the longest 2 code words will be same except for the first last 2 bits last bits.

So, after we have observed this fact we can say that, this particular code cannot be Huffman code because it has only 1 longest codeword. So, in the design process we have seen that the longest codeword should be there for at least 2 symbols and those 2 symbols will have the same code length differing only in the last 2 last bits. So, this cannot be a Huffman code can this be Huffman code yes this can be Huffman code we at least see that, this satisfies the observation we have just made in the design process.

That is it has maximum length to and there are 2 code words of the length 2 and differ only in the last bits. So, this can be Huffman code and in fact, we can see that this will be a Huffman code.

(Refer Slide Time: 48:39)

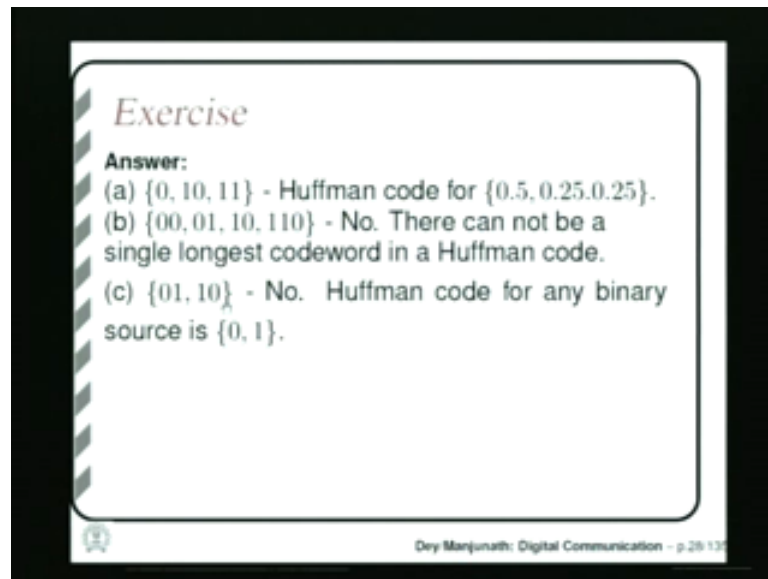


For this, will be Huffman code for the exactly this random variable. If we have random variable taking values 1 2 and 3 with probabilities 0.5 0.25 and 0.25 then what will be the Huffman code we will combine these 2 probabilities. So, first step is this and then we

will combine these 2 probabilities to get probability 1. So, add 0 here and 1 here 0 to this symbol and then 1 to both these symbols.

So, in fact we have got exactly the code that is given in the exercise. So, we have got this code and the this is Huffman code for the random variable taking 3 values if probability 0.5, 0.25 and 0.25.

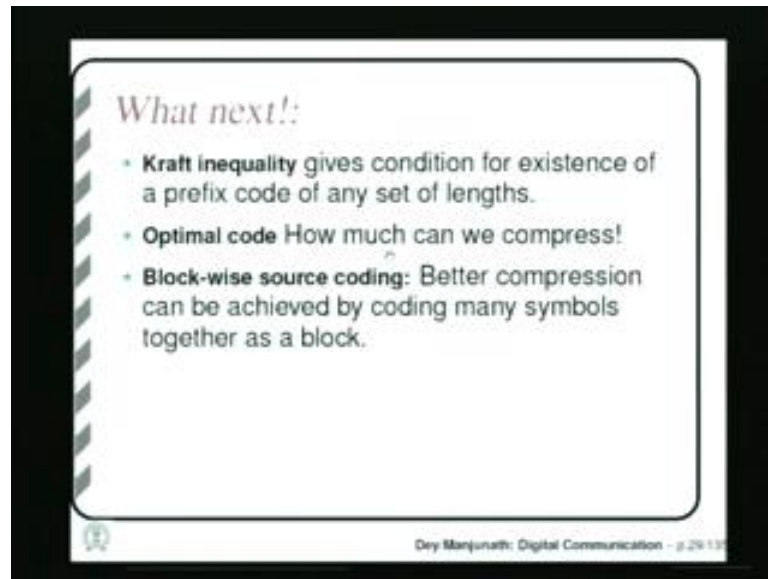
(Refer Slide Time: 49:50)



So, this is a Huffman code for the random variable with these probabilities and this is not a Huffman code because they are cannot be a single longest code word in a Huffman code. The longest other the maximum length should be therefore, 2 for 2 2 code words for any Huffman code. This can also not be a Huffman code because it has 2 symbols and for any binary source the Huffman code will be just 0 and 1 because for a binary source suppose, the first probability is p and the other is $1 - p$ the first probability is p than the other must be $1 - p$.

Suppose, p is longest the p is larger p is greater than $1 - p$ then in the construction of Huffman code just involves 1 step. The total is 1 combine these 2 probabilities we assign 0 here 1 here. So, the codes code words will be 0 and 1. So, for any binary source the Huffman code is just 0 and 1 it cannot be 0110. So, Huffman code for any binary source is 0 1 it cannot be 0110.

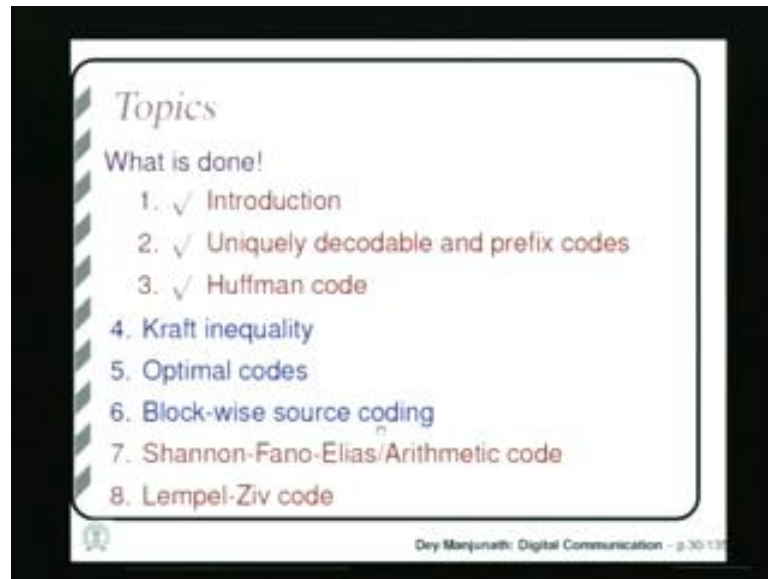
(Refer Slide Time: 51:23)



So, in the next will do Kraft inequality, Kraft inequality gives a condition for existence of a prefix code for any set of lengths. So, if we add given a set of lengths 2 3 4 4 is there a prefix code of this lengths. So, the Kraft inequality will answer that question it will give a necessary and sufficient condition for existence of a prefix code for any given lengths. We will discuss about optimal code we will see what is optimal code and what is the expected length of optimal code.

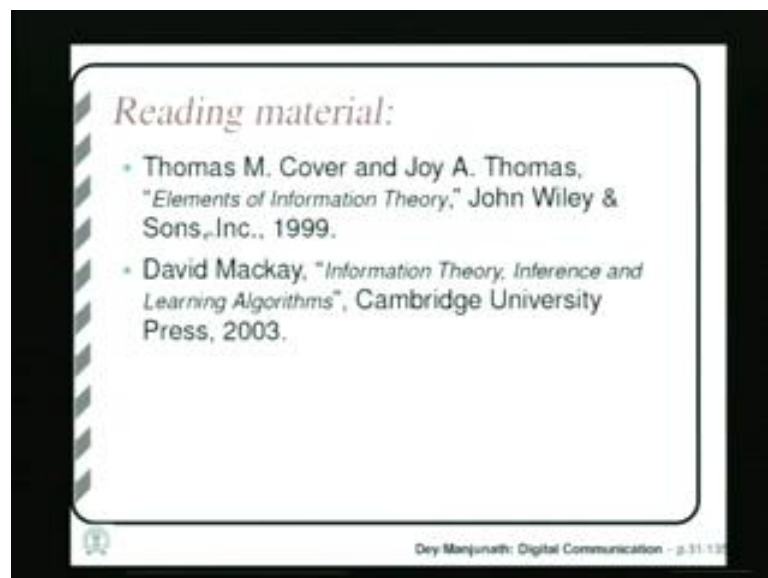
So, this will also answer some questions about how much compression can be done for the random variable and then we will discuss block wise source coding here, we will see that better compression can be achieved by coding many symbols together as block instead of coding every symbol independently we can code we can combine many symbols together and do coding. So, that will give us better result.

(Refer Slide Time: 52:30)



So, in this class, we have covered introduction uniquely decodable and prefix codes then we have seen what is Huffman code and how to construct that and some properties of Huffman codes. And in the next class, we look a Kraft inequality optimal codes and block wise source coding.

(Refer Slide Time: 52:50)



Here are some reading materials from which these topics can be studied. Thomas M Cover, elements of information theory is a very classic book on information theory. And this is a more recent book by, David Mackay information theory inference and learning algorithms. And this presents these techniques in a very lucid manner. So, I will

encourage the audience to read this particular text book. So, we conclude this lecture here see again later.