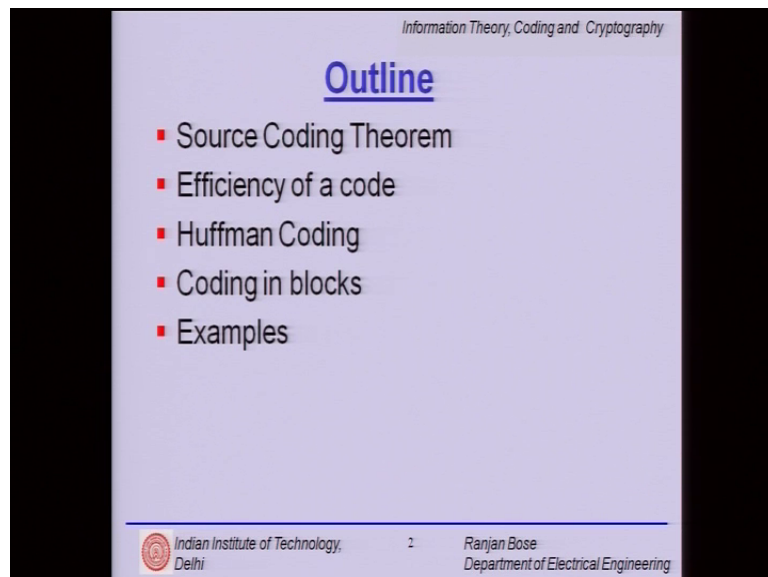


Information Theory, Coding and Cryptography
Dr. Ranjan Bose
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Module - 05
Source Coding
Lecture - 05

Hello and welcome to module 5 of Source Coding. Let us look at the brief outline for today's lecture.

(Refer Slide Time: 00:22)



We will revisit source coding theorem, we will again look at what do we mean by efficiency of a code. Then we will look at one practical prefix coding scheme called the Huffman coding scheme. And then we will explore coding in blocks not one symbol at a time, but coding several blocks at a time. And finally, we will look at some examples.

So, this is the broad outline for today's talk. We start with a quick recap of what we did in the previous lectures.

(Refer Slide Time: 01:00)

Information Theory, Coding and Cryptography

Recap

- Variable Length Codes
- Kraft Inequality
- Source Coding Theorem
- Efficiency of a code

Indian Institute of Technology, Delhi 3 Ranjan Bose
Department of Electrical Engineering

We have established that whenever symbols occur with different probabilities there is scope for compression that is there is a possibility to represent them more efficiently. That brought us to the notion of variable length codes which means that symbols may have different lengths of codewords attached to them.

We looked at Kraft inequality and we started to scratch the surface of the source coding theorem.

(Refer Slide Time: 01:40)

Information Theory, Coding and Cryptography

Example of a Prefix Code

Letter	Codeword	Letter	Codeword
A	00	E	101
B	010	F	110
C	011	G	1110
D	100	H	1111

▪ It can be seen that no codeword forms the prefix of any other codeword.

Indian Institute of Technology, Delhi 4 Ranjan Bose
Department of Electrical Engineering

So, let us recap a quick example of a prefix code we did this example in the previous class. We have the first 8 alphabet of the English language ABCD up to H and we have a corresponding codeword.

So, this whole set is a code a code is a collection of cod words and what we observed in the previous lecture is that no codeword forms the prefix of any other codeword. We can do a quick sanity check for example, 0 0 which is a codeword of A; nothing begins with a 0 0 or 0 1 1 which is a corresponding code word for C none of the codewords start with 0 1 1 this means not no codeword is a prefix of any other codeword; this is the basic essence of this code.

(Refer Slide Time: 02:37)

Information Theory, Coding and Cryptography

Prefix Codes

- “No codeword forms the prefix of any other codeword”.
- This is called the **prefix condition**.
- So, as soon as a sequence of bits corresponding to any one of the possible codewords is detected, we can declare that symbol decoded.

A **Prefix Code** is one in which no codeword forms the prefix of any other codeword.

- Such codes are also called **Instantaneous Codes**.

Indian Institute of Technology, Delhi 5 Ranjan Bose Department of Electrical Engineering

This condition that no codeword forms the prefix of any other codeword is called the prefix condition right. And since, these codes are such that we do not have to wait to declare the results. If we have an input bit stream coming the moment we find a sequence of bits corresponding to any valid codeword from a table, we declare the result then in there we do not have to wait for it. Therefore, these codes are also called instantaneous codes please note that suppose we make this condition suffix codes as opposed to prefix code.

Then we probably cannot instantaneously declare the results ok. So, we have to go through several code words coming back and then we have to walk backwards to understand what was really sent. So, prefix codes work whereas, suffix code may not

work though both are uniquely decodable. We also saw examples of codes which are not uniquely decodable that is you can have multiple possible answers to the bit stream that you receive.

(Refer Slide Time: 03:56)

Information Theory, Coding and Cryptography


Kraft Inequality

- A necessary and sufficient condition for the existence of a binary code with codewords having lengths $n_1 \leq n_2 \leq \dots \leq n_L$ that satisfy the **prefix condition** is

$$\sum_{k=1}^L 2^{-n_k} \leq 1$$

- For prefix codes over an alphabet of size M we have

$$\sum_{k=1}^L M^{-n_k} \leq 1$$

 Indian Institute of Technology, Delhi
 6
Ranjan Bose
Department of Electrical Engineering

We observed in the previous class the Kraft inequality which is shown as follows summation k is equal to 1, through L 2 is for minus k n k less than or equal to 1 right this is a condition for prefix condition.


(Refer Slide Time: 04:17)

Information Theory, Coding and Cryptography

Source Coding Theorem

- Let X be the ensemble of letters from a DMS with finite entropy $H(X)$ and the output symbols $x_k, k = 1, 2, \dots, L$, occurring with probabilities $P(x_k), k = 1, 2, \dots, L$.
- It is possible to construct a code that satisfies the prefix condition and has an **average length** that satisfies the inequality

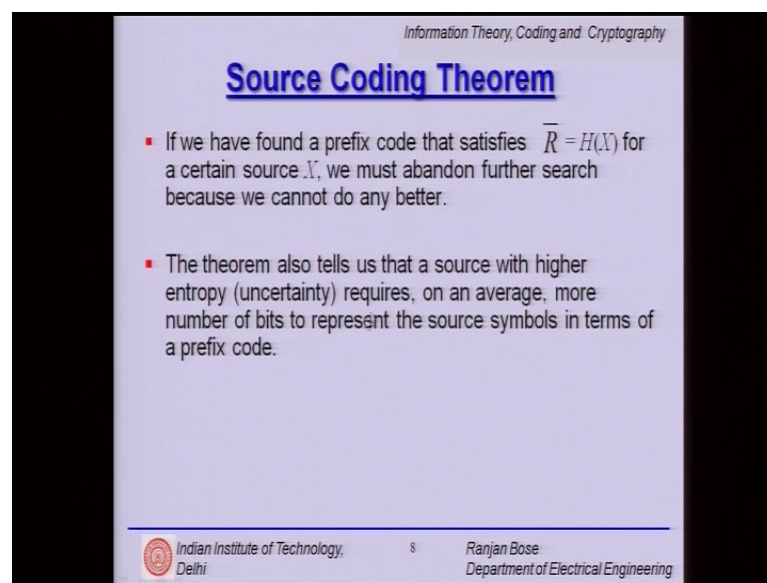
$$H(X) \leq \bar{R} < H(X) + 1.$$

 Indian Institute of Technology, Delhi
 7
Ranjan Bose
Department of Electrical Engineering

Let us look at the source coding theorem that we talked about here we have L output symbols from a source; the source has a finite entropy H of X . Now each of the symbols x_k have an associated probability P of x_k then the source coding theorem says that it is possible to construct a code that satisfies the prefix condition and also has an average length \bar{R} which is limited by H of X as the lower bound and H of X plus 1 as the upper bound the units are in bits.

So, please note that \bar{R} which is the average length of the code provided it follows the prefix condition is pretty efficient. The best you can do is to take it down up to H of X , but even if you cannot and sometimes it will not be in your control to make the average length of the code down up to H of X ; if it is a prefix code it will at worst be upper limited by HX plus 1.

(Refer Slide Time: 05:45)



Information Theory, Coding and Cryptography

Source Coding Theorem

- If we have found a prefix code that satisfies $\bar{R} = H(X)$ for a certain source X , we must abandon further search because we cannot do any better.
- The theorem also tells us that a source with higher entropy (uncertainty) requires, on an average, more number of bits to represent the source symbols in terms of a prefix code.

Indian Institute of Technology, Delhi

Ranjan Bose
Department of Electrical Engineering

So, it is off by 1-bit not too bad. So, it is worth exploring what \bar{R} is when we have a prefix code. It also tells us that once we have a prefix code that satisfies \bar{R} is equal to H of X , we should abandon further search because we have reached the ultimate. And intuitively the source coding theorem tells us that if a source is higher entropy H of X then on an average we require more number of bits to represent the symbols; please note the key word is on an average.

(Refer Slide Time: 06:21)

Information Theory, Coding and Cryptography

Efficiency of a Code

- The **Efficiency** of a prefix code is defined as

$$\eta = \frac{H(x)}{R}$$

- It is clear from the source coding theorem that the efficiency of a prefix code $\eta \leq 1$.
- Efficient representation of symbols leads to **compression** of data.
- Source coding is primarily used for compression of data (speech, text, image, video etc.).

Indian Institute of Technology, Delhi

9

Ranjan Bose
Department of Electrical Engineering

We also defined the efficiency of a code as a ratio of H of X divided by R bar. So, clearly it is less than or equal to 1 efficiency of the code really tells us how much we can compress the source symbols. And we will use this source coding to compress data, speech, text, audio, image, X-ray data any samples that you get from the vibrations of the bridge, any signal that you get wherever there is redundancy we should be able to compress it and we must compress it. Because today the storage of bits and the transmission of bits both require resources and is expensive.


So, it is important that whatever can be compressed should be compressed.

(Refer Slide Time: 07:18)

Information Theory, Coding and Cryptography

Huffman Coding

- A **variable length** encoding algorithm was suggested by Huffman in 1952, based on the source symbol probabilities $P(x_i), i = 1, 2, \dots, L$.
- The algorithm is **optimal** in the sense that the average number of bits required to represent the source symbols is a minimum and also meets the prefix condition.

 Indian Institute of Technology, Delhi10Ranjan Bose
Department of Electrical Engineering

Since this prefix code is such a good case to argue for we should look at an example how to come up with a prefix code. And this was introduced in 1952 by Huffman as a variable length encoding algorithm. What it requires is the source symbol probabilities $P x i$; so, suppose there are L possible symbols each one has an associated probability $P x i$.

Now, this Huffman coding is optimal in the sense that the average number of bits required to represent the source symbol is a minimum we cannot do any better than this on an average and it also meets the prefix condition.

(Refer Slide Time: 08:13)

Information Theory, Coding and Cryptography

Huffman Coding

- Arrange the source symbols in *decreasing* order of their probabilities.
- Take the bottom two symbols and tie them together as shown below.
- Add the probabilities of the two symbols and write it on the combined node.


High

↑

↓

Low

$$\begin{array}{c} p_{n-2} \\ p_{n-1} \quad \left(\begin{array}{c} 0 \\ 1 \end{array} \right) \quad \left| \begin{array}{c} p_{n-1} + p_n \\ p_n \end{array} \right. \\ p_n \end{array}$$

 Indian Institute of Technology, Delhi11Ranjan Bose
Department of Electrical Engineering

So, let us look at it how do we do Huffman coding the first thing is that you collect all the source symbol probabilities and arrange them in a decreasing order. So, in this example let us have 1 2 3 4 up to n probabilities of course, we are not showing p_1 p_2 p_3 , but right at the bottom we have p_n as the lowest probability p_{n-1} , little higher p_{n-2} higher than that and so and so, forth.

So, we always start with the bottom of the ladder. So, on this ladder they have arranged all the probabilities of occurrences of the symbols, but bottom is the smallest and top is the highest. So, we focus our attention on the bottom on the later. So, what we do is take the bottom 2. And we will explain the rational behind it, but first idea is to take the bottom 2 and combine them how do how do you combine them you tie them up together and add their probabilities.

So, in a sense we are making one complex signal from the bottom 2 symbols which are least probable. And then once we add them we write the net probabilities because probabilities will add up.

(Refer Slide Time: 09:47)

Information Theory, Coding and Cryptography

Huffman Coding

- Each time we perform the combination of two symbols we reduce the total number of symbols by one.
- Whenever we tie together two probabilities (nodes) we label the two branches with a '1' and a '0'.
- Continue the procedure until only one probability is left (and it should be 1 if your addition is right!).
- This completes the construction of the Huffman tree.

Indian Institute of Technology, Delhi
12
Ranjan Bose
Department of Electrical Engineering

Now what we do is we will repeat this step for 2 symbols at a time as we go along for example, now p_n and $n-1$ together have formed one complex signal. That becomes the lowermost provided $p_{n-1} + p_n$ the sum of the probabilities is still lower than p_{n-2} . Otherwise, you have to rearrange this complex symbol along the ladder and place it as a level where it is again in the decreasing order.

So, each time we perform the combination of 2 symbols. Obviously, we reduce the total number of the symbol by 1. So, at every step we lose one symbol and as we go along we will end up finally, with one symbol and that symbol has probabilities of all the symbols added up and it should add up to 1 because the probabilities must add up to 1. So, here we have taken the bottom 2 symbol labeled them as a complex symbol it is still lower than p_{n-2} .

So, it occupies a lower level and then we combine p_{n-2} , the probability with this probability to give another combined symbol. So, after 2 steps out of these 3 possible symbols we are only left with one at every step we will lose one symbol and now we write the combined probability here.

(Refer Slide Time: 11:37)

Information Theory, Coding and Cryptography

Example

- Consider a DMS with seven possible symbols $x_i, i = 1, 2, \dots, 7$ and the corresponding probabilities
 $p_1 = 0.37, p_2 = 0.33, p_3 = 0.16, p_4 = 0.07, p_5 = 0.04, p_6 = 0.02, p_7 = 0.01$.
- We first arrange the probabilities in the decreasing order and then construct the Huffman tree.

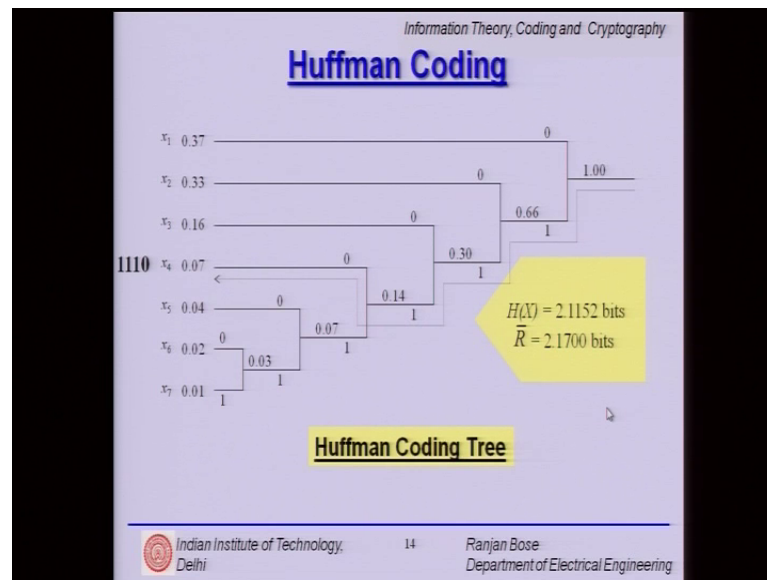
Symbol	Probability
x_1	0.37
x_2	0.33
x_3	0.16
x_4	0.07
x_5	0.04
x_6	0.02
x_7	0.01

Indian Institute of Technology, Delhi
 13
Ranjan Bose
Department of Electrical Engineering

Let us look at an example suppose there is a discrete memory less source with only 7 possible symbols $x_i, i = 1$ through 7 and we have the corresponding probabilities. So, in this table we write the probabilities. Just for reference suppose these probabilities were all equal suppose the symbols were equiprobable then in order to represent 7 or even 8 symbols we would clearly require 3 bits per symbol; $\log_2 8$ will give me 3. So, 3 symbols 3 bits are sufficient to represent the symbols, but can we do better? That is the question and here we have the probabilities.

So, first step is we have arranged them in a decreasing order x_1 most probable x_7 least probable.

(Refer Slide Time: 12:35)



Now here is the arrangement in the decreasing order; we start with the bottom 2 0.01 and 0.02 I combine them up and I am glad to see that 0.03 still less than 0.04. So, my decreasing order is 0.37, 0.33 going up to 0.04 and then point 0.3. So, I still have now instead of 7 we have effectively 6 symbols with their probabilities right here ok. Now we continue this process we combine now this 0.03 with 0.04 it will add up to 0.07 and then 0.07 combine with 0.07 and so and so, forth. So, I keep adding them up and I form this structure ok.

So, please note 0.01 plus 0.02 is 0.03; 0.03 plus 0.04 bottom 2; 0.07 0.07 0.07 big question which is bigger? Well we can toss a coin to choose which one is bigger right now let us say point this 0.07 is lower than this, but nothing stops me from putting this symbol higher up than this. We will look at a subsequent example to prove this. So, 0.07 plus 0.07 0.14 still lower than 0.16; so, the probabilities at this stage is 0.37, 0.33, 0.16 and 0.14 great we combined bottom 2; 0.30 probability of 0.37, 0.33, 0.30 still in the decreasing order I have got deeper into the tree and so and so forth I keep combining 0.66 and so and so forth. We can take it up to 1, we can flip this also, but whether the last stage. So, 1 means I have added all of them correctly.

Now, suppose having constructed this Huffman coding tree; I am curious to find out what should be all the codewords associated with x_1 to x_7 let us choose x_4 for example, all we have to do is follow this tree backwards. So, this is the mother node and

we go up to x_4 by following it up. So, I go here I have a choice I go up or go down I need to go to x_4 . So, I go down the moment I go down I add 1. So, whenever I have to take a choice go up I will add a 0 go down I add up as one therefore, these are labeled 0 1 0 1 0 1 and so and so, forth.

Since my target symbol is x_4 and I go down one and I need to go down 1, but now to go to x where I go up and I get 0. So, I add 1 1 1 and 0 and that should be my symbol the codeword for x_4 and so and so forth. So, what is unique what is interesting is as follows; please note the highest the higher the probability, the fewer bifurcations it has to go through. So, this particular guy x_1 most probable had to just go through 1 bifurcation. So, it just gets the number 0, but this guy probably had to go to 1 2 3 bifurcations.

So, as we go down the ladder; we will have to carry on through more steps and consequently I will have a larger representation for less probable symbols right. Now I have got these probabilities; so, it is very easy for me to calculate the H of X again I could have corresponding codewords. So, it is very easy for me to multiply the length of the codeword with the probability of occurrence and average it out to find out the average codeword length.

So, invariably I will calculate H of X, plug in the values I get some numbers. So, answer is 2.152 bits and I get the \bar{R} average codeword length as 2.17 bits. It is not surprising to see that \bar{R} is greater than the H of X H of X is the Holy Grail the best we can do and I can take the ratio and I can give you the efficiency of the code.

(Refer Slide Time: 17:25)

Information Theory, Coding and Cryptography

Example

$$H(X) = -\sum_{k=1}^7 P(x_k) \log_2 P(x_k) = 2.1152 \text{ bits}$$

$$\bar{R} = \sum_{k=1}^7 n_k P(x_k) = 1(0.37) + 2(0.33) + 3(0.16) + 4(0.07) + 5(0.04) + 6(0.02) + 6(0.01) = 2.1700 \text{ bits.}$$

The efficiency of this code is $\eta = (2.1152/2.1700) = 0.9747$

Symbol	Probability	Self Information	Codeword
x_1	0.37	1.4344	0
x_2	0.33	1.5995	10
x_3	0.16	2.6439	110
x_4	0.07	3.8365	1110
x_5	0.04	4.6439	11110
x_6	0.02	5.6439	111110
x_7	0.01	6.6439	111111

Indian Institute of Technology, Delhi
15
Ranjan Bose
Department of Electrical Engineering

So, here is a detailed calculation how I received H of X and R bar H of X is nothing but summation over all symbols $7 P \times k \log$ to the base 2 $P \times k$ if you solve this you get this value and similarly for the average codeword length; length of the codeword into probability of utterance. So, this one is 1 bit long and 0.37; so, I multiplied it 1 the second one is 2 bit long multiplied by 0.33 and so and so forth. So, I get 2.17 bits and efficiency is 0.97.

Now let us look at it a little bit more carefully what have we done? So, far we had this 9 symbols x_1 through x_7 and then I reached in the decreasing order of the probability and we have been able to find the codewords associated with them. Categorically for x_4 we had found 1 1 1 0. Similarly we find for all others, but what we have added is this table for self information. Clearly x_1 which is more probable has a less self information as compared to x_7 which is very very rare remember self information inversely proportional to the probability of occurrence.

So, clearly the self information which necessarily does not have to be an integer are all over the place right from 1.43 right up to 6.64. Codewords on the other hands must be integers fine; so, we cannot really fine tune too much in terms of the codeword lengths as opposed to self information. Where does the compression come from? Well if the information is less I should use fewer bits because units of information is also bits.

So, in principle I should actually use 1.43 bits to represent this first symbol, but I am using 1-bit I should use 1.599 bits to represent the second symbol, but I am using 2 bits wasteful, but I have no choice; either I use 1-bit represent or 2 bits to represent and so on. So, for this is my dilemma 2.64 this symbol $\times 3$ only commands information worth 2.64 bits, but unfortunately Huffman has allocated 3 bits wasteful and I will be paying in terms of my poor R bar and hence the poorer efficiency of the code.

Where are we losing the efficiency? It is all here I just do not have a choice look at x_6 it requires 5.64 bits to represent it and I have got 1 2 3 4 5 6 or estimated, but all is not bad x_7 requires 6.64 bits, but again I have got only 6 bits. So, win some lose some, but in on an average I am a little off than my H of X H of X is completely determined by the self information average self information is H of X ok; so, this is the first weakness of Huffman code.

(Refer Slide Time: 21:16)

Information Theory, Coding and Cryptography

Another Example

- Consider a DMS with seven possible symbols $x_i, i = 1, 2, \dots, 7$ and the corresponding probabilities

Symbol	Probability
x_1	0.46
x_2	0.30
x_3	0.12
x_4	0.06
x_5	0.03
x_6	0.02
x_7	0.01

Indian Institute of Technology, Delhi
 16
Ranjan Bose
Department of Electrical Engineering

Let us look at another example again we have are 7 symbols hm; please note if we had equiprobable then we would have no choice, but to represent 3 bits per symbol please note here.

(Refer Slide Time: 21:31)

Information Theory, Coding and Cryptography

Example

$$H(X) = -\sum_{k=1}^7 P(x_k) \log_2 P(x_k) = 2.1152 \text{ bits}$$

$$\bar{R} = \sum_{k=1}^7 n_k P(x_k) = 1(0.37) + 2(0.33) + 3(0.16) + 4(0.07) + 5(0.04) + 6(0.02) + 6(0.01) = 2.1700 \text{ bits.}$$

The efficiency of this code is $\eta = (2.1152/2.1700) = 0.9747$

Symbol	Probability	Self Information	Codeword
x_1	0.37	1.4344	0
x_2	0.33	1.5995	10
x_3	0.16	2.6439	110
x_4	0.07	3.8365	1110
x_5	0.04	4.6439	11110
x_6	0.02	5.6439	111110
x_7	0.01	6.6439	111111

Indian Institute of Technology, Delhi
 15
Ranjan Bose
Department of Electrical Engineering

In the previous example, we were able to reach up to 2.17 bits instead of the worst case analysis of 3 bits per symbol. But in this example let us see what we have again we have probabilities and I have taken the liberty to arrange them in decreasing order of probability.

(Refer Slide Time: 21:53)

Information Theory, Coding and Cryptography

Another Example

- We construct the Huffman Tree as follows

$H(X) = 1.9781 \text{ bits}$
 $\bar{R} = 1.9900 \text{ bits}$

Indian Institute of Technology, Delhi
 17
Ranjan Bose
Department of Electrical Engineering

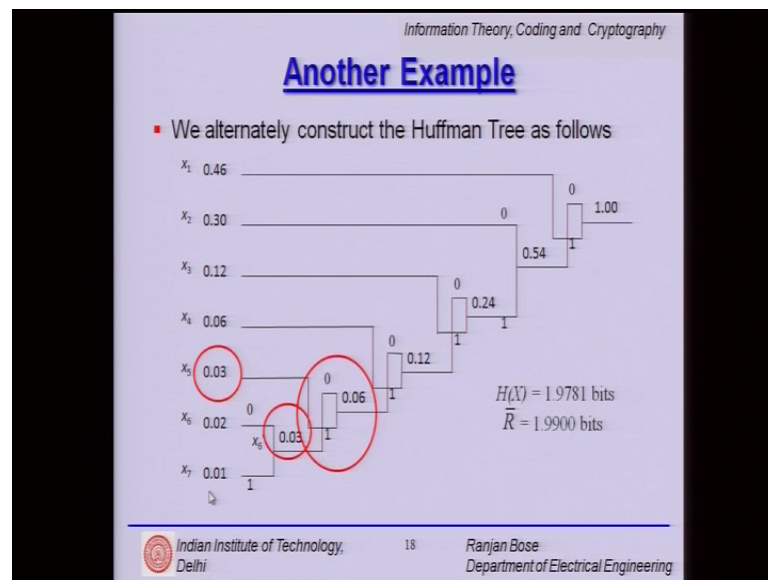
The construction of the Huffman tree is pretty mechanical decreasing order of probabilities, take the bottom to combine. So, 0.01 plus 0.02 combine to get 0.03; 0.03 plus 0.03 0.06, but what is going on? 0.03 and 0.03 that the same nobody is holier than

the other one there is no reason why this symbol should be lower than this or for the time being less keep them like this. But we should be able to switch them also, but what is this 0.03 0.03 add up to 0.06 I have to compare again because they have to be in decreasing order of probability.

So, there is no reason why this 0.06 should be lower than 0.06 here, but we have no choice. So, let us keep it here, but we could have tossed a coin and put this symbol higher up than this, but today is not my day 0.12 and 0.12 again they are equal. Maybe I should exchange them in terms of ordering. So, the decreasing order probability ladders say is 0.46, 0.30 0.1 to 0.12, but this 0.12 can be higher up than this 0.1 and then we have 0.24 and so on so forth.

So, if you do this and again calculate the H of X it comes out to be 1.9781 bits and R bar as expected is larger than H of X is cos is equal to 1.99 bits for this different example. So, please note 7 bits in this example. I change the probabilities, but my R bar and H of X both have changed its obvious if I change probabilities H of X will change and R bar will also change.

(Refer Slide Time: 24:00)



Now we change our thinking and we say look this 0.03 should be rated higher than this 0.03. This 0.06 should be above this 0.06. So, if what if we change the probabilities? What if we change the ordering? So, same probabilities 0.46, 0.30, 0.12 you can check with the previous example, but this time when I get 0.01 plus 0.02 as 0.03; I put this

above; above what? This peer. So, this one goes down please not it will change the codeword because whether you take the upper branch or lower branch will decide whether you add a 0 and a 1 0.06 again 0.06 I said this guy is higher up than this guy. So, again I interchange 0.12; again I play the same creek this 0.12 goes up this 0.12 comes down.

So, see how convoluted this second cases as opposed to the previous case where each one time when I had an option; I have really twisted the arms each time of the Huffman tree and so on. So, when I add up to one. So, I am sure that my addition is correct; so, well taking this 0.03 and switched it over.

Now what do we find? Well we can again calculate H of X it goes without saying that H of X is the same as previous one because I have not touched the input probabilities. H of X only depends on the input probabilities; so it is the same, but what about \bar{R} ? Well, \bar{R} part is also same; so, the average codeword length has not changed.

The codewords would have changed for sure have they? Why do not we look at it.

(Refer Slide Time: 26:16)

Information Theory, Coding and Cryptography

The two possible Codes

Symbol	Probability	Self Information	Codeword	Codeword Length
x_1	0.46	1.1203	1	1
x_2	0.30	1.7370	00	2
x_3	0.12	3.0589	010	3
x_4	0.06	4.0589	0110	4
x_5	0.03	5.0589	01110	5
x_6	0.02	5.6439	011110	6
x_7	0.01	6.6439	011111	6

Symbol	Probability	Self Information	Codeword	Codeword Length
x_1	0.46	1.1203	1	1
x_2	0.30	1.7370	00	2
x_3	0.12	3.0589	011	3
x_4	0.06	4.0589	0101	4
x_5	0.03	5.0589	01001	5
x_6	0.02	5.6439	010000	6
x_7	0.01	6.6439	010001	6

$H(X) = 1.9781$ bits
 $\bar{R} = 1.9900$ bits

Indian Institute of Technology, Delhi
 19
Ranjan Bose
Department of Electrical Engineering

So, this table tells us the 2 different realizations of the Huffman tree for the same set of input probabilities ok. So, what are those?; so, we have x_1 through x_7 and you can verify that the probabilities are the same. And needless to say the self information must

also be the same right, but in the first case we did not reorder the equal probabilities whereas, in the second case we reordered them.

So, we ended up with different sets of codewords ok. So, here the third one x 3 was 0 1 0 here x 3 0 1 1 x 4 is 0 1 1 0 here 0 1 0 1 and so and so forth. The bottom one the least probable is 0 1 1 1 1 1 1 and here it is 0 1 0 0 1. Now what is to be noted is since it is based on a binary tree; the prefix condition will always be satisfied. So, you can verify that no codeword either in case 1 or in case 2 is a prefix of any other codeword. And finally, we write the codeword length the codeword lengths are also the same.

So, clearly when I find R bar I multiply 1 into 0.46 plus 2 into 0.30 into plus 3 into 0.12 and so and so forth 6 into 0.01 add them up I get the same answer as this, but mind you this always does not have to be. I could have done additional jugglery in between by choosing not to have all of them switched, I could have flipped the first one and flipped the third one, but not flipped this one and I will get another set of codewords. And they may not be of equal lengths correspondingly still the R bar magically will come out to be the same.

So, in this both the cases H of X and R bar are the same. So, if you look at the efficiency of the code.

(Refer Slide Time: 28:50)

Information Theory, Coding and Cryptography

Non uniqueness of Huffman Code

- The efficiency of this code is
$$\eta = (1.9781/1.9900) = 0.9940.$$
- Thus both the codes are equally efficient.
- This also tells us that Huffman codes are not unique for a given set of probabilities.

Indian Institute of Technology, Delhi 20 Ranjan Bose, Department of Electrical Engineering

You can divide the H of X by R bar to get 0.9940. So, both the codes are equally efficient, but it tells us that Huffman codes are not unique for a given set of probabilities ok. You can have more than one realization, but efficiency is the same ok. So, that is the take home message that we can have multiple realizations of the Huffman code for the same input probabilities.

(Refer Slide Time: 29:30)

Information Theory, Coding and Cryptography

Yet Another Example

- Consider a DMS with seven possible symbols $x_i, i = 1, 2, \dots, 7$

Symbol	Probability	Self Information	Codeword	Codeword Length
x_1	2^{-1}	1	1	1
x_2	2^{-2}	2	00	2
x_3	2^{-3}	3	010	3
x_4	2^{-4}	4	0110	4
x_5	2^{-5}	5	01110	5
x_6	2^{-6}	6	011110	6
x_7	2^{-6}	6	011111	6

$H(X) = 1.9688$ bits
 $R = 1.9688$ bits

$\eta = 1$

Indian Institute of Technology,
Delhi

21

Ranjan Bose
Department of Electrical Engineering

So, let us explore a little bit deeper and look at another example; again our discrete memory less source is generating possibly 7 different kinds of outputs x_1 up to 7. And then these probabilities are associated with these symbols. So, x_1 has probability $1/2$, x_2 $1/4$, x_3 $1/8$ appears to be a pattern to respond minus 1; 2 respond minus 2 to respond minus 3 upto to respond minus 6, but this one last one I need all the probabilities to add up to 1 such again 2 respond minus 6.

So, how does it work? Add these 2 up you get 2^{-5} , add these 2 up 2^{-4} raised to minus 3 keep adding and it will add up to one. So, the probabilities add up to one it is a valid probability measure, but having put this probability in this funny manner we note another very interesting thing. We find that the self information are all integers; $\log_2(1/P(x_i))$; so, here it is 1, 2 right up to 6.

Suppose we will do Huffman coding on it we get this following prefix code. Now we write the corresponding codeword length side by side what we observe is very interesting; finally, the codeword length which has to be integer is the same as the self

information which is by design and integer. We have been able to finally, map the length to the self information length 1 to 1. So, x_4 has the self information 4 and we have done justice to it we have only allocated 4 bits the real event x_7 has self information 6 bits and I have given it 6 bits.

So, we have really done allocation optimally bits are resources we have not over allocated or under allocated, we have matched it. So, what it means is this very very unique distribution only allows us to do this. And this should not be a surprise if you calculate the H of X it is 1.9688 bits and if we calculate the average codeword length it is also 1.9688 bits.

So, we clearly have the efficiency equal to 1; we have reached we have reached the rock bottom the best we can do. This special probability distribution and only this special probability distribution allows me to reach \bar{R} equal to the lower bound H of X . And this also tells you why in all other cases my \bar{R} will be greater than H of X because I just cannot match the self information with the integer codeword length fine.

(Refer Slide Time: 33:16)

Information Theory, Coding and Cryptography

Yet Another Example (cont'd)

- A probability distribution is called D -adic with respect to D if each of the probabilities is equal to D^{-n} for some integer n . By definition, the distribution in the above table is D -adic.
- The Huffman coding scheme has been able to match the codeword lengths *exactly* to the probabilities (and thus the self information) of the symbols.
- For example, if the self information of x_5 is 5 bits, the Huffman code has actually allocated a codeword of length 5 to this symbol.
- It is clear that to achieve optimality ($\eta = 1$) the self information of the symbols must be integers, which in turn, implies that the probabilities must be negative powers of 2.
- This is not always true in the real-world.

Indian Institute of Technology, Delhi

22

Ranjan Bose
Department of Electrical Engineering

So, this probability distribution is called D addict. So, the distribution probabilities are D respond minus n for some integer n the Huffman coding scheme has been able to match the codeword lengths exactly to the probabilities and hence the self information of the symbol. And therefore, we have been able to achieve η equal to 1 the efficiency equal to 1, but this kind of a nice beautiful distribution is not always found in the real world.

So, this is where we are; so, let us look at the Huffman coding from a slightly different perspective. Let us look at a person whose job is to allocate bits alright. So, every time he reached reaches a bifurcation this person has to traverse a path from this base up to one of the valid symbols and allocate the codewords right. Let us look at even a simpler example; so, what we are trying to do is try to help this person allocate a codeword to a particular symbol and the way he does it is he either takes the upper path or the lower path, but to take a path to take a decision he uses a bit ok.

So, the value of the decision is 1-bit which path to choose? The upper path or the lower path, whenever you have to take a decision you have to invest in bits, but the problem with this guy is whether the decision is easy or the decision is difficult he has to allocate 1-bit each time ok. What do you mean by decision is easy? Well if the probabilities are 0.9 and 0.1 right it is an easy decision.

But if it is 0.5 and 0.5 it is a difficult decision. So, that difficult decision warrants 1 bit, but an easy decision does not require it to waste 1-bit for each 1 and that is the reason why the resource allocation while going up to the end is not optimal. Only when the bifurcation is equiprobable say 0.6 and 0.6 then it is worth allocating 1-bit for this decision.

But if I am mapping 0.46 and 0.50 they are not equiprobable each one does not deserve 1-bit or if I am doing some other 0.24 and 0.3 adding up to 0.54; again they do not each deserved resolved 1-bit. Why am I saying this? Tomorrow if you have to improve upon this method we should be able to allocate fractional or proportionate amount of bits when we go up and down; then we can do better than this. And we will come to a method which allows us to allocate bits on the real line, but right now we are stuck with allocating either a 1 or a 0 when we go up and down the stream ok.

(Refer Slide Time: 37:14)

Information Theory, Coding and Cryptography

Working blockwise

- In the above examples, encoding is done symbol by symbol. A more efficient procedure is to encode blocks of B symbols at a time. In this case the bounds of the source coding theorem becomes

$$BH(X) \leq \bar{R}_B < BH(X) + 1.$$
- since the entropy of a B -symbol block is simply $BH(X)$, and \bar{R}_B is the average number of bits per B -symbol block.
- We can rewrite the bound as

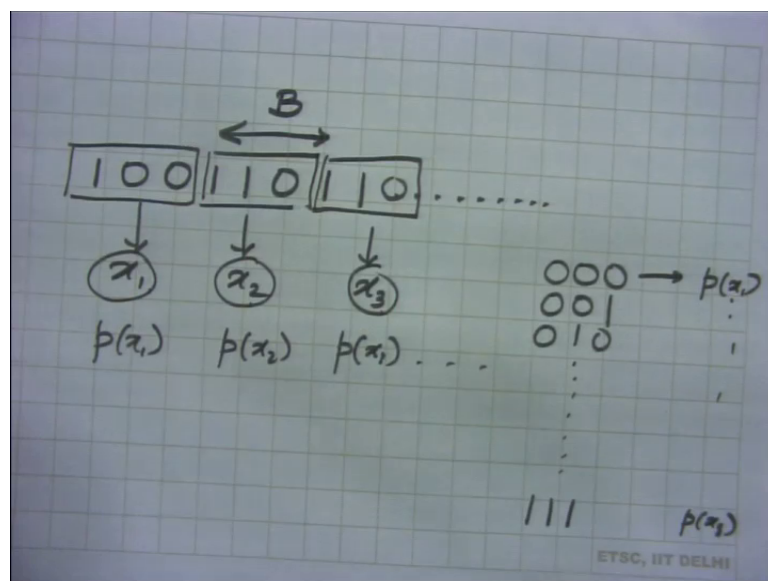
$$H(X) \leq \frac{\bar{R}_B}{B} < H(X) + \frac{1}{B}$$

 where $\frac{\bar{R}_B}{B} = \bar{R}$

Indian Institute of Technology, Delhi
23 Ranjan Bose
Department of Electrical Engineering

Now let us look at a slightly different approach; we ask ourselves the question why do we work only symbol wise? What if we combine 2 or more symbols together ok; so well suppose we have a bit string and we do not work symbol by symbol, but by blocks of simple alright. But, since they are independent if once the source has an entropy of H of X ; the blocks of b symbol would have an entropy BH of X . And if we apply source coding theorem, because we treat this combined blocks as the super symbols. Then we will have a corresponding R_B and it is upper bounded by BH of X plus 1.

(Refer Slide Time: 38:25)



So, let me explain suppose we have a bit stream and we choose to take 3 bits at a time and declare them symbols x_1, x_2, x_3 . Well, these themselves have their own probabilities p_1, p_2, p_3 and so and so, forth. In fact, if you look at this example you have right from 0 0 0 0 0 1, 0 1 0 up to 1 1 1 you have got corresponding probabilities p_1, p_2 and so on and so, forth up to p_8 ; what it means is that I can now do Huffman coding at this complex symbol levels.

Let us look at the equivalent source coding realization for that. So, we now have instead of $H(X)$ we now have $BH(X)$ and R_B is the number of average number of bits per block ok. This happens to be might be; so, R_B is the average number of bits per B symbol block. So, we can divide all the sides by B and we can rewrite the bound as $H(X) \leq R_B/B \leq H(X) + 1/B$. But R_B is the average number of bits required to represent B symbol block, but how many symbols does it have? B symbols. So, R_B divided by B is average number of bits per symbol right and that is my R .

So, again we can compare apples with apples and I am left with $H(X) \leq R + 1/B$. So, working block wise has been able to pinch, squeeze the upper bound earlier it was $H(X) + 1$. Now it is $1/B$ and as I make the block size be larger I try to squeeze the upper bound and the upper bound squeezes and it tends to $H(X)$. So, Huffman coding in blocks would be able to give me a much higher efficiency.

(Refer Slide Time: 41:47)

Information Theory, Coding and Cryptography

Example


Symbol	Probability	Self Information	Codeword
x_1	0.40	1.3219	1
x_2	0.35	1.5146	00
x_3	0.25	2.0000	01

↓

$H(X) = 1.5589$ bits
 $\bar{R} = 1.6000$ bits

↓

$\eta = 0.9743$

 Indian Institute of Technology, Delhi24Ranjan Bose
Department of Electrical Engineering

Let us look at an example this time let us have 3 symbols only with probabilities 0.4, 0.35, 0.25 and mechanically I find myself information for these 3 and I do a Huffman coding and I get 1 0 0 0 1 as my 3 codewords, a very simple H of X is 1.55 and average codeword length is 1.6 bits. So, clearly efficiency is less than 1 or how much is it is 0.9743 fair enough 3 bits given probabilities all the steps are mechanical this is the best I can do.

But I now wonder can I work block by block? Theory says that I can do better why do not we give it a try. So, what are my pairs I make B equal to 2?

(Refer Slide Time: 42:45)

Information Theory, Coding and Cryptography

Example

Symbol Pairs	Probability	Self Information	Codeword
x_1x_1	0.1600	2.6439	10
x_1x_2	0.1400	2.8365	001
x_2x_1	0.1400	2.8365	010
x_2x_2	0.1225	3.0291	011
x_1x_3	0.1000	3.3219	111
x_2x_1	0.1000	3.3219	0000
x_2x_3	0.0875	3.5146	0001
x_3x_2	0.0875	3.5146	1100
x_3x_3	0.0625	4.0000	1101

$$2H(X) = 3.1177 \text{ bits} \Rightarrow H(X) = 1.5589 \text{ bits}$$

$$2\bar{R} = 3.1775 \text{ bits} \Rightarrow \bar{R} = 1.5888 \text{ bits}$$

$\eta = 0.9812$

Better than

$\eta = 0.9743$

Indian Institute of Technology, Delhi
25
Ranjan Bose, Department of Electrical Engineering

So, I pair up each one and I first find out the probabilities alright. So, what does it mean? I have $x_1 x_1$, $x_1 x_2$, $x_2 x_1$, $x_2 x_2$ and so, forth up to $x_3 x_3$; so, 9 independent probabilities multiply so, x_1 was 0.4; so, $x_1 x_2$ or $x_1 x_1$ is 0.16 right. x_3 is 0.25; so $x_3 x_3$ is 0.0625 and now I found the self information and I do a Huffman coding on it this is pretty mechanical. But this time I find that if I do your H of X . So, we find for the symbol which is $2H$ of X is 3.1177 bits consequently H of X is 1.558 bits and for a block of 2 symbols; $2\bar{R}$ is 3.1775 bits and \bar{R} that is per symbol is 1.5888; how does it compare with the previous one? Where this time η efficiency is 0.9812, but what was it last time? Well last time it was 0.9743. So, I have been able to squeeze a little bit more efficiency its better than my previous case.

So, coding in blocks helps I will not stop here I can go beyond I can go to $x_1 x_1 x_1 x_1$ $x_1 x_1 x_2$ right up to $x_3 x_3 x_3$ and I get the probability corresponding probability distribution and I can write the self information and codeword and find it is better why is it working better?.

(Refer Slide Time: 44:51)

Information Theory, Coding and Cryptography

Example

Symbol	Probability	Self Information	Codeword
x_1	0.40	1.3219	1
x_2	0.35	1.5146	00
x_3	0.25	2.0000	01

$H(X) = 1.5589$ bits
 $\bar{R} = 1.6000$ bits

$\eta = 0.9743$

Indian Institute of Technology, Delhi
24
Ranjan Bose
Department of Electrical Engineering

The answer is pretty simple if you look at here, the self information is very coarse where the probabilities of very coarse. There are 3 symbols 3 probabilities they must add up to 1; whatever I have I have written the self information. I am trying to match the codeword length to the best; so, 1.3 is mapped to 1 length 1.5 is mapped 2 and 2 is mapped 2.

(Refer Slide Time: 45:26)

Information Theory, Coding and Cryptography

Example

Symbol Pairs	Probability	Self Information	Codeword
x_1x_1	0.1600	2.6439	10
x_1x_2	0.1400	2.8365	001
x_2x_1	0.1400	2.8365	010
x_2x_2	0.1225	3.0291	011
x_1x_3	0.1000	3.3219	111
x_2x_1	0.1000	3.3219	0000
x_2x_3	0.0875	3.5146	0001
x_3x_2	0.0875	3.5146	1100
x_3x_3	0.0625	4.0000	1101

$2H(X) = 3.1177$ bits $\Rightarrow H(X) = 1.5589$ bits
 $2\bar{R} = 3.1775$ bits $\Rightarrow \bar{R} = 1.5888$ bits

Indian Institute of Technology, Delhi
25
Ranjan Bose
Department of Electrical Engineering

So, I have only done just its to x 3 in terms of allocation of the number of bits to the self information value just all I have not done a good job. But if you look at this to symbol

pairs right; the number of elements now has gone up in my combined symbol list it is now 9, but probabilities must add up to 1.

So, the granularity it is become much more fine; the distribution has become much more distributed. So, the self information values as you can see and not such discrete it is slightly easier to map them up; so, 2.8 is now mapped to 3 as opposed to 1.5 being mapped 2.

(Refer Slide Time: 46:08)

Information Theory, Coding and Cryptography

Example

Symbol	Probability	Self Information	Codeword
x_1	0.40	1.3219	1
x_2	0.35	1.5146	00
x_3	0.25	2.0000	01


↓

$H(X) = 1.5589$ bits
 $\bar{R} = 1.6000$ bits

↓

$\eta = 0.9743$

↓

Indian Institute of Technology,
Delhi24Ranjan Bose
Department of Electrical Engineering

What a poor map; so, we are trying to understand why this grouping works; mathematically it works, but we are trying to understand why it works.

(Refer Slide Time: 44:16)

Information Theory, Coding and Cryptography

Example

Symbol Pairs	Probability	Self Information	Codeword
x_1x_1	0.1600	2.6439	10
x_1x_2	0.1400	2.8365	001
x_2x_1	0.1400	2.8365	010
x_2x_2	0.1225	3.0291	011
x_1x_3	0.1000	3.3219	111
x_2x_1	0.1000	3.3219	0000
x_2x_3	0.0875	3.5146	0001
x_3x_2	0.0875	3.5146	1100
x_3x_3	0.0625	4.0000	1101

$$2H(\tilde{X}) = 3.1177 \text{ bits} \Rightarrow H(\tilde{X}) = 1.5589 \text{ bits}$$

$$2\bar{R} = 3.1775 \text{ bits} \Rightarrow \bar{R} = 1.5888 \text{ bits}$$

$$\eta = 0.9812$$
➔
Better than
➔

$$\eta = 0.9743$$

So, if you can see again 2.83 its trying to map to 3 not a correct map, but a closer web 3.02 map to 3 I am doing a better job right and so and so, forth and somewhere of course, it is not so, good. So, 3.5 is mapped to 4 not so, good. And therefore, a pay and a my efficiency is farther away from 1, this 4 is mapped to 4. So, we have done a good job in the end.

So, the bottom line is if you combine if you work in groups you have a chance of improving your efficiency. Price is the computational complexity goes up you can see the size of the Huffman tree will increase exponentially and that itself will cause additional problems at the cost of giving you a better efficiency ok

(Refer Slide Time: 47:22)

Information Theory, Coding and Cryptography

Summary

- Source Coding Theorem
- Efficiency of a code
- Huffman Coding
- Coding in blocks
- Examples

Indian Institute of Technology, Delhi 26 Ranjan Bose
Department of Electrical Engineering

So, we now come to the end of this talk let me summarize what we have done; we revisited source coding theorem, we talked about the efficiency of a code very important closely linked to the compression then we looked at Huffman coding.

We finally, looked at an example why coding in blocks helps and we of course, looked at several examples throughout today's lecture what if conditions, what if you interchange probabilities, equal probabilities what if you distort the 2 Huffman tree twist the branches and things like that and we got some insight into why Huffman tree works. And why it does not work beyond a certain limit and probably, how you can make it work even more efficiently.

So, we come to the end of this module.