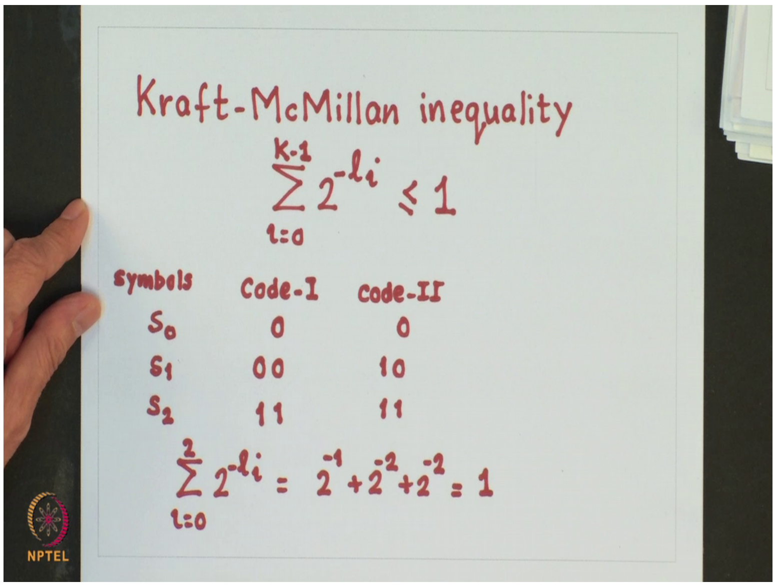
Principles of Digital Communications

Prof. Shabbir N. Merchant
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Lecture – 05
Huffman Coding

Hello, welcome back. In the last class we studied prefix code which is a subset of a uniquely decodable code. We also saw that prefix code and uniquely decodable codes they satisfy what is known as McKraft-McMillan inequality. So, the inequality is as follows, correct.

(Refer Slide Time: 00:46)



Kraft-McMillan inequality

$$\sum_{i=0}^{K-1} 2^{-l_i} \leq 1$$

Symbols	Code-I	Code-II
S_0	0	0
S_1	00	10
S_2	11	11

$$\sum_{i=0}^{2} 2^{-l_i} = 2^{-1} + 2^{-2} + 2^{-2} = 1$$

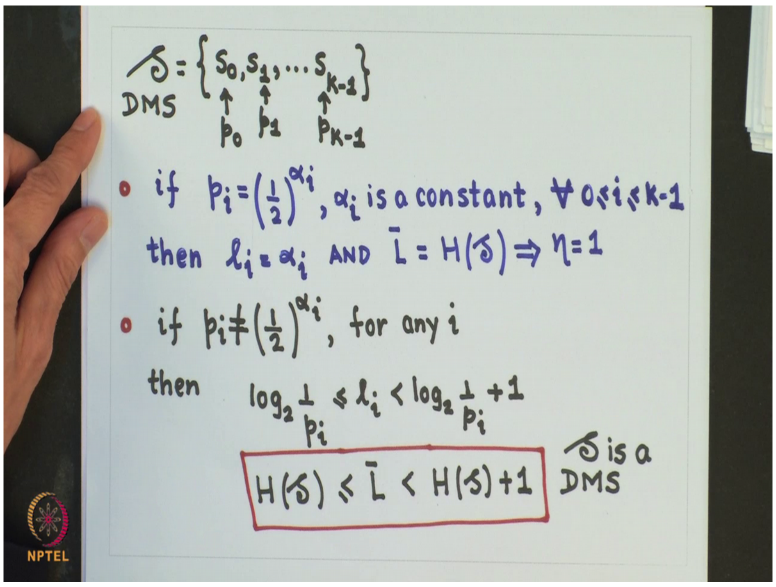
And what it says basically that all uniquely decodable codes should satisfy this inequality and this is a necessary and sufficient condition. So, what it means that if I have a prefix or uniquely decodable code then this condition will be satisfied and if this condition is satisfied then it is possible for me to generate a uniquely decodable code of prefix code.

Now, it is important to stress that this inequality is on the length of the code. So, let us take a I have a source with 3 symbols and I have two codes which I have shown here. Now, both these codes satisfy the length condition, if I evaluate this I get this equal to 1.

So, this condition is satisfied, but we know that code 1 is unacceptable whereas, code 2 is acceptable for the same lengths of the code rules. So, this condition only tells you that their code exists, but does not specify the procedure to design such codes.

We also studied a strategy for choosing the lengths of the code words for a prefix code and that is summarized in the slide here.

(Refer Slide Time: 02:36)



$\mathcal{S} = \{s_0, s_1, \ldots s_{K-1}\}$ DMS, with p_0, p_1, p_{K-1}

- if $p_i = \left(\frac{1}{2}\right)^{\alpha_i}$, α_i is a constant, $\forall\, 0 \leq i \leq K-1$
  then $l_i = \alpha_i$ AND $\bar{L} = H(\mathcal{S}) \Rightarrow \eta = 1$
- if $p_i \neq \left(\frac{1}{2}\right)^{\alpha_i}$, for any i
  then $\log_2 \frac{1}{p_i} \leq l_i < \log_2 \frac{1}{p_i} + 1$

$$\boxed{H(\mathcal{S}) \leq \bar{L} < H(\mathcal{S}) + 1}$$ $\mathcal{S}$ is a DMS

So, what we said that given a source with the probabilities associate with the symbols of the source we said that if pi the probability of a symbol is of this form half raised to a constant where alpha i is an integer, then I can choose my l i's equal to alpha is which will be equal to log to the base 2 of 1 by p i.

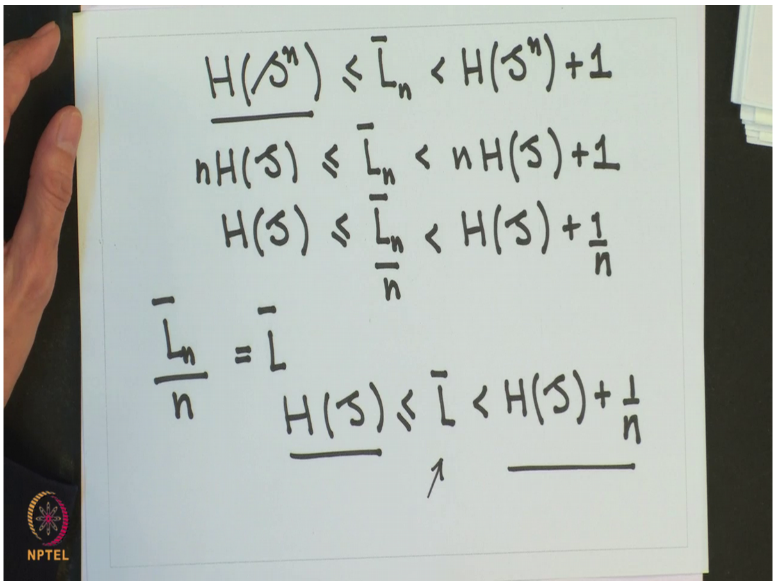
And we showed that for such a case the average length of the code turns out to be the entropy of the source and the efficiency turns out to be 1, but for the case where the condition is not satisfied where you do not have of this form where alpha is a integers then we said that we could choose the lengths of the code words given by this inequality.

And we showed that if we use this inequality using this lower bound Kraft-McMillan inequality gets satisfied. So, what it is means that I should be able to design a prefix code and for that prefix code I will get this result. So, what it says that that the average length of the code cannot be smaller than the entropy of the source and using this strategy we can get the average of length of the code in binits within 1 binit of the entropy of a discrete memoryless source.

Now, this strategy may not be very good specifically when the entropy of our source is a low value we will see this little later on. But if you want to improve this coding efficiency another way would be to extend it to coding of nth extension of the source.

So, let us extend this strategy of choosing the lengths of the codes given by this inequality to the nth extension of the source and if we do that then we should get this result ok, where L n bar represents the average length of the code words corresponding to the symbols from the nth extension of the source s.

(Refer Slide Time: 05:20)



$$\underline{H(S^n)} \leq \bar{L}_n < H(S^n) + 1$$

$$nH(S) \leq \bar{L}_n < nH(S) + 1$$

$$H(S) \leq \frac{\bar{L}_n}{n} < H(S) + \frac{1}{n}$$

$$\frac{\bar{L}_n}{n} = \bar{L}$$

$$\underline{H(S)} \leq \underset{\uparrow}{\bar{L}} < \underline{H(S) + \frac{1}{n}}$$

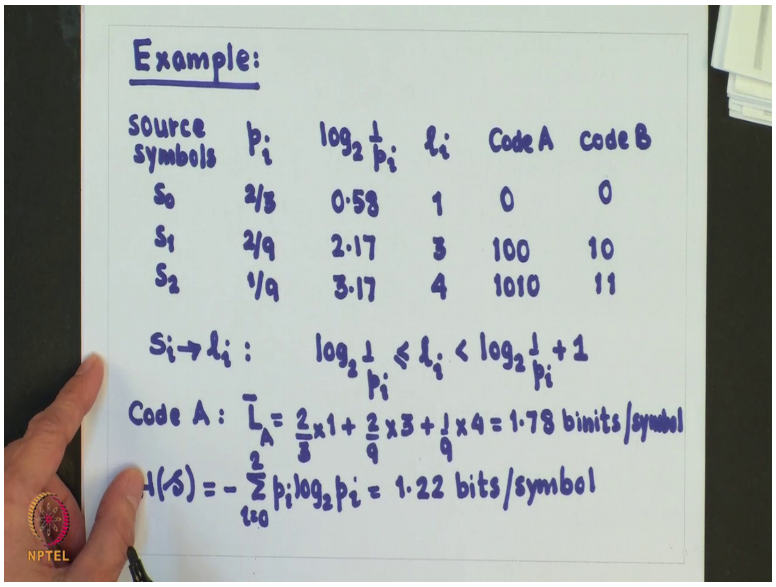
So, if we assume our source is discrete memoryless source then I can write this quantity has n times H s and then by dividing by n on all the sides I get this inequality.

Now, I can this quantity is nothing, but the average length in binits required to code the original source symbol correct therefore, I can write this as as follows. So, what this result tells us that for source encoding we can make the average number of binits per source symbol as small as, but not smaller than the entropy of the source and for given n the result also tells us that the average length will be not greater than the right side of this equation. So, you will have the average length within one by n binit of the entropy of the source.

So, by increasing n we can get the value of L average to be as close as possible to H s that is the entropy of the source. But the price we pay for this is that the coding complexity will increase because of large number of symbols which you will have in the nth extension of the source. It is also important to know that this strategy of choosing the lengths for the code word, does not tell us what is the value of this L average correct and another thing is that it does not guarantee that I if I choose my code word lengths as per this strategy then the code which I will get will have the average length which is the minimum, correct.

So, I will let us take a simple example which will serve to show that this choice may indeed provide a poor way to choose the codeword lengths that is an example out here.

(Refer Slide Time: 09:00)



Example:

Source symbols	p_i	$\log_2 \frac{1}{p_i}$	l_i	Code A	code B
s_0	2/3	0.58	1	0	0
s_1	2/9	2.17	3	100	10
s_2	1/9	3.17	4	1010	11

$s_i \to l_i : \quad \log_2 \frac{1}{p_i} \leq l_i < \log_2 \frac{1}{p_i} + 1$

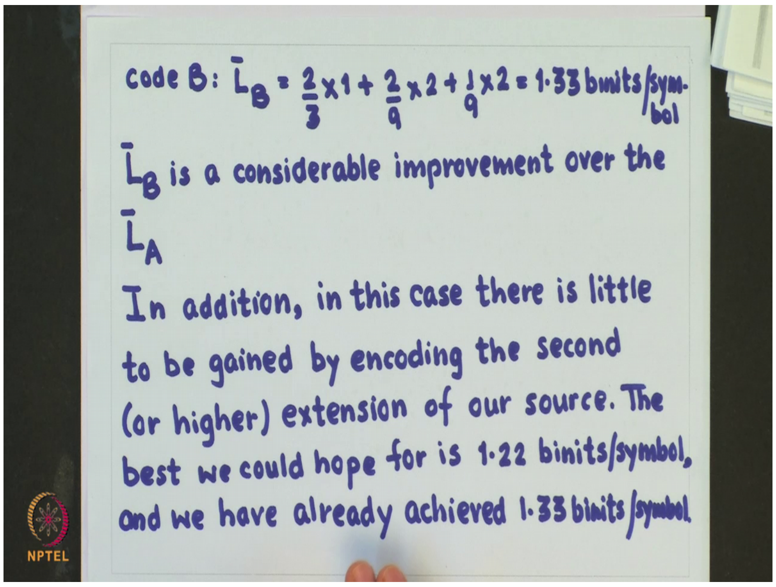
Code A: $\bar{L}_A = \frac{2}{3} \times 1 + \frac{2}{9} \times 3 + \frac{1}{9} \times 4 = 1.78$ binits/symbol

$H(S) = -\sum_{i=0}^{2} p_i \log_2 p_i = 1.22$ bits/symbol

I have a 3 source symbols and for 3 4 symbols the probabilities have been specified. So, I can calculate log to 1 by pi and I get this quantity and then using this strategy I choose my lengths to be 1 3 4, and then you can show that I can generate a code this satisfies this length condition ok. So, I get this code. So, I have used this strategy.

Now, if you calculate the length for this code A you will come out it will turn out to be 1.78 binits per symbol, and the entropy of the source turns out to be 1.22 bits. So, this basically strategy thus satisfied this relationship between the average length and the entropy of the source. So, that is fine correct. But we know that we can design another code which is more efficient than this and that code basically is given by code B. Now, if you take code B and if you calculate the length for this it turns out to be 1.33 binits per symbol.

(Refer Slide Time: 10:12)



code B: $\bar{L}_B = \frac{2}{3}\times 1 + \frac{2}{9}\times 2 + \frac{1}{9}\times 2 = 1.33$ binits/symbol

$\bar{L}_B$ is a considerable improvement over the $\bar{L}_A$

In addition, in this case there is little to be gained by encoding the second (or higher) extension of our source. The best we could hope for is 1.22 binits/symbol, and we have already achieved 1.33 binits/symbol

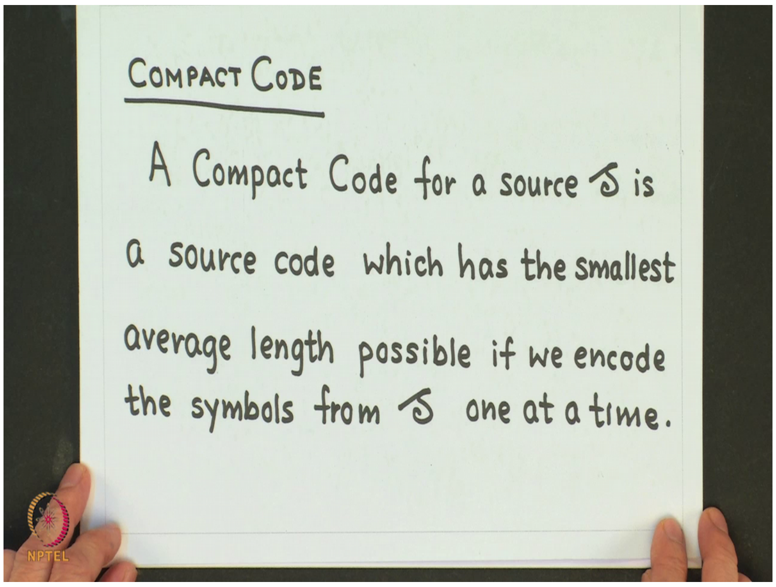
So, what it means basically the average length which I got using the code B is a considerable improvement over the average length which I could get using the code A and the code A was constructed using this inequality or this strategy which we had discussed earlier, correct.

So, I just want to say that that this strategy need not provide the best codeword lengths or the code average code length which is minimum, ok. And what this strategy also this code B also shows that there is little to be gained by encoding the second or higher extension of our source because best we can hope is 1.22 binits per symbol and using the code B we have already achieved 1.33 binits per symbol.

So, from the results of our study we can conclude that if you want to improve the efficiency of source encoding then it is better to carry out the coding in terms of n symbols of a source. And by increasing the value of n the efficiency can be increased, but at the expense of coding complexity. Now, the question is that given a fixed end is it possible for me to find a source code or design a source code or construct a source code such that the average length of that source code will be minimum.

So, we will define what is a compact code. A compact code for a source s is a source code which has the smallest average length possible if we encode the symbols from s one at a time, and one algorithm to get this compact code is Huffman coding.

(Refer Slide Time: 12:12)

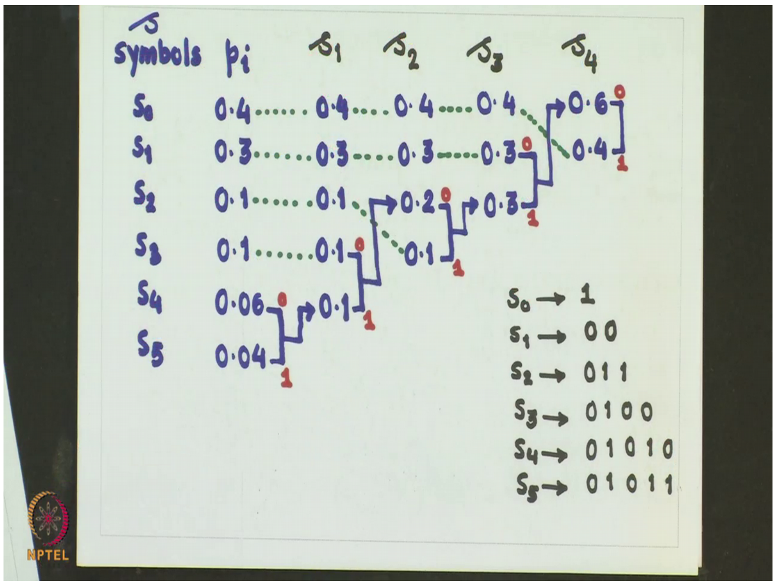


COMPACT CODE

A Compact Code for a source $\mathcal{S}$ is a source code which has the smallest average length possible if we encode the symbols from $\mathcal{S}$ one at a time.

So, without going into the formal proof of proving that Huffman code is a compact code we will show the construction of a Huffman code for a given source with the help of an example.

(Refer Slide Time: 12:47)



So, I have a source consisting of 6 symbols and the probabilities have also been shown here in the second column. For the first step of Huffman coding is to order the symbols of the source s in decreasing order of probabilities.

The next step is basically, combine the 2 symbols with the lowest probability into 1 symbol with probability equal to the sum of the probabilities of the symbols being combined, and obtain a new source from s containing only k minus 1 symbols in this case 5 symbols. We refer to this new source as a reduction of s and we call it as s 1 now the symbols of this reduction of s may be reordered and again we may combine the two list probable symbols to form a reduction of this reduction of s. So, we get s 2 from s 1 which is reduction of reduction of s.

So, by proceeding in this manner we construct a sequence of sources each containing one symbol fewer than the previous one until we arrive at a source with only two symbols. So, at this stage we find the code word for the s 4 and the codewords for this s 4 symbols are just 0 and one once we have this codewords we proceed to the preceding source and construct the codewords for the symbols in that source and that is done as follows 0.4 this symbol is also same as the symbol here in s 3. So, the codeword for this will be the identical to the codeword which we have in s 4. So, this will be equal to 1.

Now, to determine the codewords for this two symbols in s 3 we take the codeword from s 4 that is 0 and to that we add 0 and 1 arbitrarily. So, this becomes 0 0 and this will become 0 1, then come to s 2 this was 1, so this will remain 1, this is same as this. So, this was basically 0 0. So, this will be 0 0 and as far as the codewords for this and this is concerned the codeword for this was 0 1, so we take that codeword 0 1 and to that we add 0, so 0 1 0 0 1 1. So, this is the way we proceed till we come at the first column here and we get the codewords for the source symbols and this is the codeword which are given here.

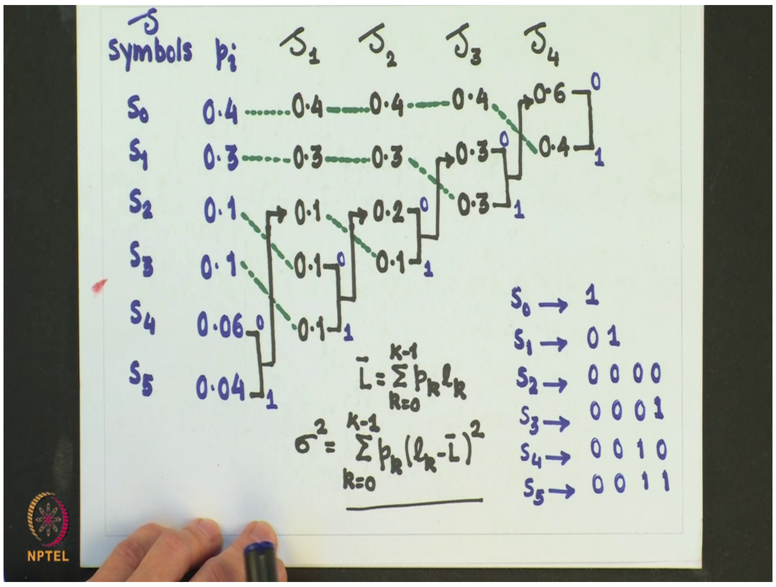
Now, notice that when we assigning this 0 and 1, here also 0 1 0 1 this is a bit arbitrary I could have assigned this 1 and 0 here also I could have done 1 and 0 correct. So, and if I do that and if I generate the codewords for that kind of assignment; obviously, the codewords would be different, but the codeword lengths will remain the same and average length of that code will be the same as the average length of this code, correct.

And what I said this arbitrary assignment of 0 and 1 can be also achieved by just flipping a particular position in all the codewords of this code.

So, if I take this say let us take a second position and if I flip this position in all the codewords. So, this becomes 1, this becomes 0 0 0 0 I will get a new set of codewords; that means, I will get a new source code, but again the length will be the same. So, these are the trivial differences you get, but there is a an important difference which occurs in this example is as follows if you take this combination we get 0 1 probability, now this probability is same as the probability of other two symbols here correct and I have left this symbol right at the below. What I could have done I could have taken it at the top.

So, if you follow that strategy and generate the code using the Huffman coding principle for the same source I would get the code as follows.

(Refer Slide Time: 18:18)



$\mathcal{S}$ Symbols p_i $\mathcal{S}_1$ $\mathcal{S}_2$ $\mathcal{S}_3$ $\mathcal{S}_4$
S_0 0.4 0.4 0.4 0.4 0.6 0
S_1 0.3 0.3 0.3 0.3 0 0.4 1
S_2 0.1 0.1 0.2 0 0.3 1
S_3 0.1 0.1 0 0.1 1
S_4 0.06 0 0.1 1
S_5 0.04 1
$\bar{L} = \sum_{k=0}^{K-1} p_k l_k$
$\sigma^2 = \sum_{k=0}^{K-1} p_k (l_k - \bar{L})^2$
$S_0 \rightarrow$ 1
$S_1 \rightarrow$ 0 1
$S_2 \rightarrow$ 0 0 0 0
$S_3 \rightarrow$ 0 0 0 1
$S_4 \rightarrow$ 0 0 1 0
$S_5 \rightarrow$ 0 0 1 1
NPTEL

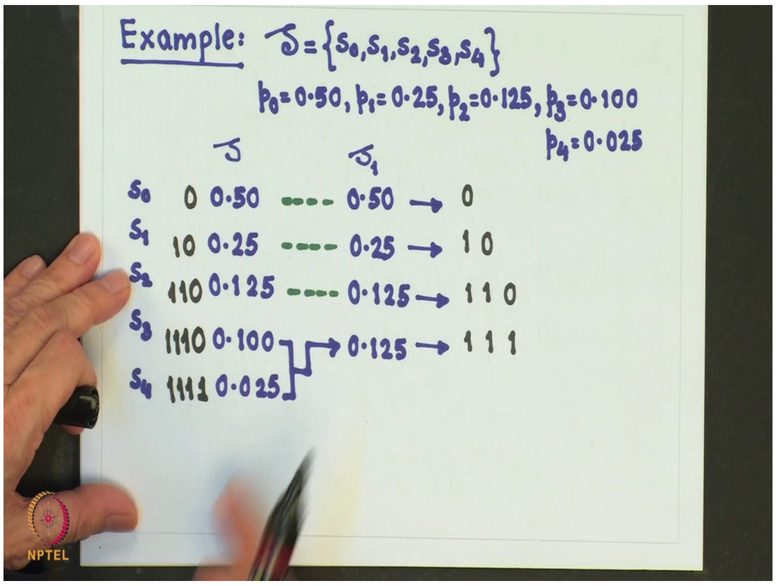
So, I combined this and the combined one I take it as high as possible. So, I take it here and other two basically I put it below then again I combine this I get 0.2, when I combine this I get 0.3 which is the same as this 0.3, but I take it as high as possible and I put in the top correct.

And then I reach the last reduction where there are only two symbols and then do the assignments of the 0s and 1s at these branches of bifurcations correct and then you get the codeword by traversing from backward. So, from here you move you can get the codeword for s 5.

And now if you see basically the codeword lengths are different, correct. So, if you adopt this principle basically and calculate the average length you will find the average length turns out to be exactly the same as the earlier one. But there is a difference is that that if you calculate the variance of the codewords in this code by this formula you will find that variance will be lower than the variance which we will obtain in the earlier case. And in practical scenarios having a code of a lower variance is always better because it avoids the chances of buffer overflow.

I would like to a note one more observation in the construction of Huffman code and that is that once you have a reduction where you can construct a compact code then it is not necessary for me to have further reductions because once I can generate the codeword or compact code for that reduction then I can generate the compact code for the preceding reduction. Take a simple example as shown here.

(Refer Slide Time: 20:40)



Example: $S = \{s_0, s_1, s_2, s_3, s_4\}$

$p_0 = 0.50, p_1 = 0.25, p_2 = 0.125, p_3 = 0.100$
$p_4 = 0.025$

		S		S_1	
s_0	0	0.50	----	0.50 →	0
s_1	10	0.25	----	0.25 →	10
s_2	110	0.125	----	0.125 →	110
s_3	1110	0.100	→	0.125 →	111
s_4	1111	0.025			

If I have a source consisting of 5 symbols and the probabilities are given as shown here. Now, if you arrange them in the decreasing order of probability this is what I get correct, and now if you take this order and if you combine this using offline code if I combine this two what I will get is 0.125. And now if you look at the source symbol probabilities in this reduced source s 1 this are of the form 1 by 2 raised to an integer. So, we know that for this kind of probabilities we can easily generate the compact code and for that the compact code is as shown here, correct.

So, now once I have a compact code for this reduction I need not proceed ahead for further reduction I can go backwards. So, what I can do is basically I can generate the codewords for my source s now this is 0 this is 0, so this will remain 0. This is 1 0, this will still remain 1 0, this is same, so this is 1 1 0 and for this basically will use the Huffman coding algorithm I have the codeword 1 1 1 and when I come here I split it 1 1 1 and add 0 to this and add 1 to this. So, I get the code for this given here in just one reduction.

So, in this class we have studied Huffman coding and we will continue our discussion on this in the next class.

Thank you.