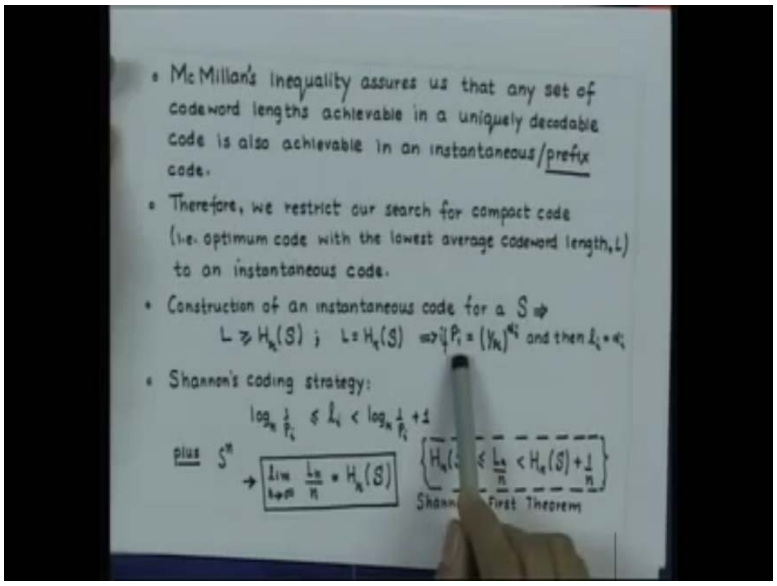
Information Theory and Coding

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

Lecture - 11
Coding Strategies and Introduction to Huffman Coding

The Fundamental problem of coding information sources is that of finding compact codes for that source. We have seen that Macmillan’s inequality assures us that any set of code words (Refer Time: 01:00) lens achievable in a uniquely decodable code, is also achievable in an instantaneous or prefix code. In the literature, instantaneous code is also known sometimes as prefix code. Therefore we restrict our search for compact code, by compact we mean, optimum code which has the average length which is the smallest. So, we restrict the search for our compact code to an instantaneous code. We have seen that, if you wish to construct an instantaneous code for a source s, then the average length of a code is always greater than or equal to the entropy of the source.

(Refer Slide Time: 02:11)



- McMillan's Inequality assures us that any set of codeword lengths achievable in a uniquely decodable code is also achievable in an instantaneous/prefix code.
- Therefore, we restrict our search for compact code (i.e. optimum code with the lowest average codeword length, L) to an instantaneous code.
- Construction of an instantaneous code for a $S \Rightarrow$
  $L \geq H_r(S)$; $L = H_r(S) \Rightarrow$ if $P_i = (1/r)^{\alpha_i}$ and then $l_i = \alpha_i$
- Shannon's coding strategy:
  $\log_r \frac{1}{P_i} \leq l_i < \log_r \frac{1}{P_i} + 1$

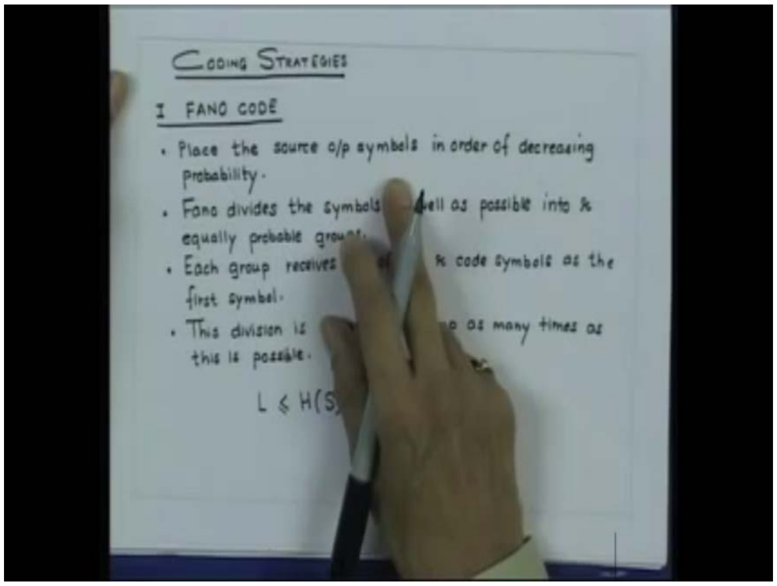
  plus S^n $\rightarrow \boxed{\lim_{n \to \infty} \frac{L_n}{n} = H_r(S)}$ $\left\{ H_r([illegible] \leq \frac{L_n}{n} < H_r(S) + \frac{1}{n} \right\}$
  Shann[illegible] First Theorem

We have also seen that in order to achieve this lower bound of entropy that is the average length equal to entropy of the source, in that case, this condition should be satisfied, where the probability of source symbols of the source that is P i should be of the form, equal to 1 by r raise to alpha i, where r is the size of the code alphabet. By choosing your l i is equal to alpha i, I can get my average length of the code equal to the entropy of the source. We have also seen that in a special case like this, how to construct an instantaneous code.

Now, for a very general case, where this condition is not satisfied for all i-s, we could use Shannon's coding strategy. Shannon's coding strategy says that I can choose my l i-s to be greater than or equal to log of 1 by P i, less than log of 1 by P i plus1. It means that we choose l i which is the first integer greater than log to the base r of 1 by P i. If I use this strategy and if I do the coding for the nth extension of the source, then we have also seen that we get this relationship. This relationship says that the average length of the code symbols per source symbol will always be bounded by this inequality. What this inequality says is that n tends to infinity; I can get this value of l n by n, as close as possible to the entropy of the source. This inequality is in fact Shannon's first theorem or noiseless coding theorem.

The question of the limiting value of l n by n is really of no relevance in the definition of compactness because we have defined a compact code for a source s, to be a code which has the smallest average length l possible. If we encode the symbols from the source s one at a time, the next question that arises is, how one can construct a compact or optimum code for a fixed source. So far we have not looked into any strategy for coding information sources, except for the special case, where the probabilities of the symbols are of the form 1 by r raise to alpha i. Before we look into the design methodology of a compact code, which is based on Huffman coding strategy, let us have a look at some simpler techniques in order to appreciate the coding strategies being followed in a practical case. With this motivation let us take up the first coding strategy that is popularly known as Fano coding.

(Refer Slide Time: 06:00)



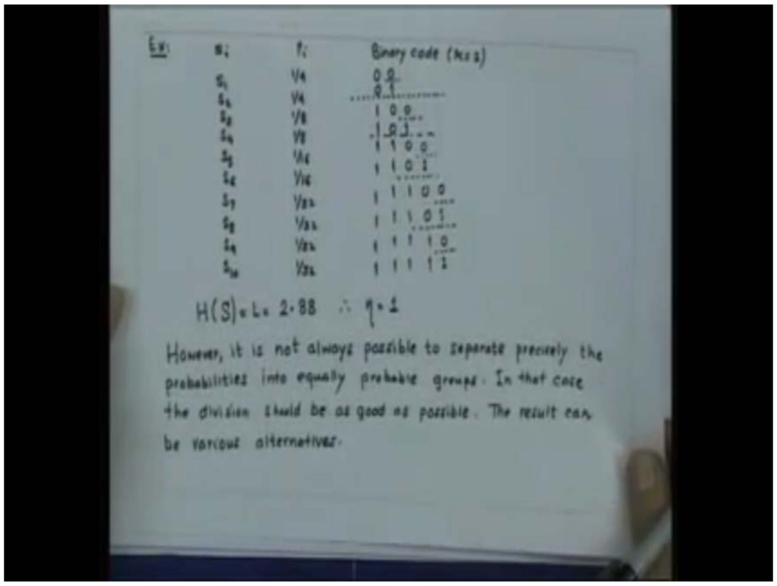
Coding Strategies

I Fano Code

- Place the source o/p symbols in order of decreasing probability.
- Fano divides the symbols [illegible]ell as possible into r equally probable grou[illegible]
- Each group receives [illegible] r code symbols as the first symbol.
- This division is [illegible] as many times as this is possible.

$L \leq H(S$[illegible]

(Refer Slide Time: 07:33)



Ex:

s_i	P_i	Binary code (r = 2)
s_1	1/4	0 0
s_2	1/4	0 1
s_3	1/8	1 0 0
s_4	1/8	1 0 1
s_5	1/16	1 1 0 0
s_6	1/16	1 1 0 1
s_7	1/32	1 1 1 0 0
s_8	1/32	1 1 1 0 1
s_9	1/32	1 1 1 1 0
s_{10}	1/32	1 1 1 1 1

$H(S) = L = 2.88 \quad \therefore \eta = 1$

However, it is not always possible to separate precisely the probabilities into equally probable groups. In that case the division should be as good as possible. The result can be various alternatives.

The coding strategy of the Fano code goes as follows. Given the source output symbols, you arrange this source output symbols in order of decreasing probability. Then the next step is basically to divide the symbols as well as possible into r equally probable groups. So r is the size of the code alphabet, which we want to use for coding. Each group receives one of the r code symbols as the first symbol, and the final step is this division is repeated per group as many times as this is possible. Let us try to use this strategy and understand it with the help of an example.

I have a source here with ten source symbols. The probabilities of the source symbols have been given to me in this column. First thing what you do is try to arrange this source symbols in the decreasing order of probability. This has already been done in this first step. Next is to divide this group into r possible group. As far as the discussion is concerned for the next few classes, we will try to restrict our discussion to binary alphabet. There are two reasons for this; the first reason is, it is easy for us to understand the concepts with the binary alphabet, and the next is in most of the time in the practical scenarios, we always use the binary alphabets. But it is not difficult to extend the ideas to any other code alphabet of size r. So for our discussion at the moment we will restrict ourselves to binary code alphabet.

So, in our case we have to divide this list of probabilities into two groups because we are considering binary alphabet. The first thing is you divide this into two groups, this becomes half and this obviously is half. So, I divide this into two groups and I allot to each source symbol in this group, binary symbols. This is the first group and this is the second group. So, for the first group I allot 0, 0 and to the next group I allot all 1-s. So this is the first step I have got. I have got two groups.

Let us consider the first group. The first group which I can again divide this into half. So, when I divide them into half, I get two groups. Each group consisting of only one source symbols and so again you allot 0 and 1.So, we are through with this group. Now, we move over to the next group. When we move over to the next group, this is a group again we divide this group into two parts equally as far as possible. It is very easy to do this in this example. So, I divide this into two groups, this plus this turns out to be one fourth and this probability of symbol, if I sum them up is again one fourth. So, I divide them into two and I again allot 0, 0 to the first group and 1, 1 to the next group. Once I have done this, I repeat this process for this group. So, when I repeat the process for this group again I have got one eight and one eight. So, when I divide into half, I get this 0 and 1. So, we are through with this group.

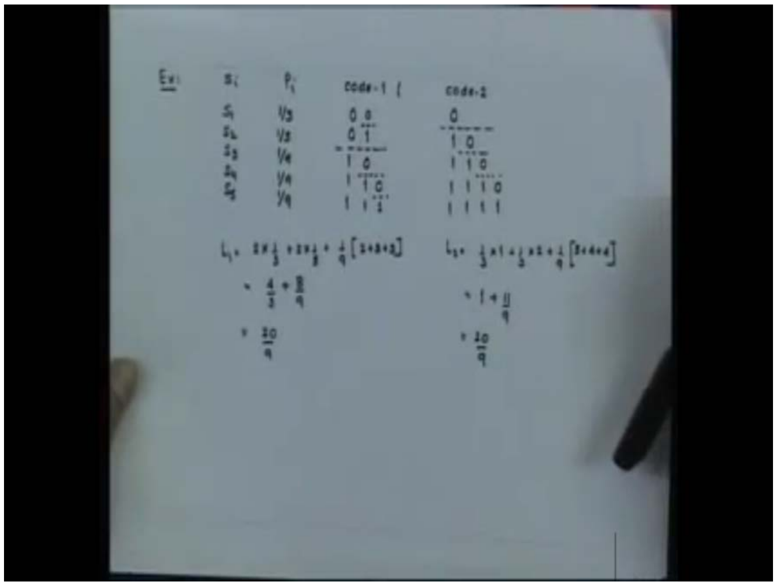
Now, we move over to this group again. Again this group has to be divided into two parts, so when I do this, I get this, I allot 0, 0, 1, 1, 1,1. Then this group I again divide into two parts, I get

0, 1. I move over to this group, I divide into two parts I get 0, 0, 1, 1. Then I move to this group, I divide into two parts 0,1 and finally this group, I divide it into0,1, and this is the code words which I have got for the source symbols based on the Fano coding strategy. If you calculate the entropy for the source, assuming that the source is IIDHS that is identically independent distribute symbols, then H S turns out to be 2.88 and for this case the average length also turns out to be 2.88. Therefore the efficiency which is defined as the entropy of the source divided by the average length that is H S by l turns out to be equal to 1.

There are some difficulties when we are carrying out coding using this strategy. First difficulty is that, it is not always possible to separate precisely the probabilities into equally probable groups. It was possible in this case because the probabilities of the symbols were very nicely behaving, but this may not be always possible. In that case the division should be as good as possible; the result can be various alternatives.

Now, if you look at this P i-s are each of the form 1 by 2 raise to alpha i. So, we could have use the strategy of our instantaneous code design, which we have discussed earlier and the result would be same. Let us take a case where this is not satisfied, where it is not possible to equally divide the group. In that case what do you do? So, let us take up one specific example for that.

(Refer Slide Time: 13:49)



Ex:

s_i	P_i	code-1	code-2
s_1	1/3	0 0	0
s_2	1/3	0 1	1 0
s_3	1/9	1 0	1 1 0
s_4	1/9	1 1 0	1 1 1 0
s_5	1/9	1 1 1	1 1 1 1

$$L_1 = 2\times\frac{1}{3} + 2\times\frac{1}{3} + \frac{1}{9}[2+3+3] = \frac{4}{3} + \frac{8}{9} = \frac{20}{9}$$

$$L_2 = \frac{1}{3}\times 1 + \frac{1}{3}\times 2 + \frac{1}{9}[3+4+4] = 1 + \frac{11}{9} = \frac{20}{9}$$

I have another example out here. I have a source consisting of five source symbols and the probabilities of the source symbols have been given here. I again assume that this source is IID that is identically independent distributed. If you go by the Fano's coding strategy, you have to divide this group equally into two parts because we are assuming binary alphabet. So, in this case one way of doing is you divide this into two third and one third. There is no other way to do it. I could have done one third and two third that is also fine, we will see what happens that way, but for the time being let us assume that, I divide into two third and one third.

I divide this into two portions, two third and one third. I assign to the above group 0, 0 the below group 1, 1.This I again divide it into two equal parts, it is possible for me to do because one third and one third, so that I get 0 and 1. Now I come back to this group, I have to divide this into two parts. Again the division is not possible. So, one strategy could be divide it one ninth and two by ninth. If you do that I get 0,1, 1. This group is done because there is only one source symbol out here, but here in this group, there are two source symbols. So, I have to again further divide it into two parts and when I do this I get 0, 1.

This I will call it as code one. Instead of this division process, if I had followed another one, for example if I had divided initially as one third and two third, then I would get 0, 1, 1, 1, 1. This group I do not have to do anything because there is only one source symbol. This I can divide it again, this is unequal, this is possible, I can divide it into two half exactly. So, one third and this is also one third. So, I get 0, 1, 1, 1 and this group is done here. Again I can divide it as 0, 1, 1. I could have used a different strategy; I could have divide it here like we did previously. But if I follow this strategy I get this and then again I have to divide this group and finally I get this value of 0, 1.

So, these are two codes which we have got based on the Fano's coding strategy. Now, how do you decide between these two codes is to calculate the average length for both of these. If you calculate the average length for this code one, the average length turns out to be 20 by 9 and average length for this code also turns out to be 20 by 9. It means that efficiency for both this code one and two are the same. So, it makes no difference which codes are used.

Next thing which I wanted to bring going out with the help of this Fano's coding strategy was to look at what happens to the average length of a code, when my source has been given and when I change the size of my code alphabet. I have the source; it is a fixed source the source symbols have been given to me. I have been given the probabilities of source symbols and now I am asked to design a coding for this source. For coding I have to decide my code alphabet. Let us look at what happens when I choose different sizes of code alphabet.

Again, I will try to explain this with the help of an illustration. Let us look at an example. I have a source consisting of eight symbols with probabilities of the symbols given as follows. This already has been arranged in the descending order that is the first step which you should carry out while carrying out the Fano's coding. Let me assume that I want to study the effect of increasing the size of the code alphabet x. Let us take the first case of the binary. If I take my first case of binary and following the same strategy which I just discussed, you can show that the code using the binary strategy is this. So, when you are using the design for a binary alphabet you have to divide them into two groups. That is why you see everywhere the division is into two groups.

Let us move to the case where instead of two bits, two binutes let us use code of size equal to three. When I take size of code size that is code alphabet size of three, then I arrange them again in the decreasing order of probabilities and divide this into three equal sections. Whatever the best way I can do, one strategy which I have followed here is I have shown that I have divided it into three groups like this. Once you divide into three groups to each group you assign the code symbol. So, we will have three code symbols because the size of the code alphabet is three. So, the three symbols which we have chosen for the code alphabets are 0, 1, 2. So, three groups are being assigned as 0, 1, 1 and 2, 2.

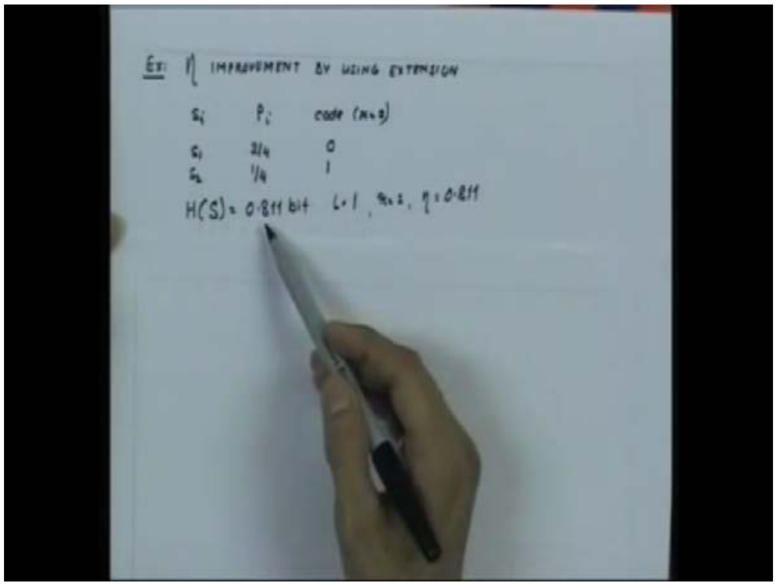
You go move to this group, I do not have anything to do. So I do not have to assign anything. When I move to this, there are only two source symbols. So, I assign 0 and 1 to this. When I move to the third group there are five source symbols which are left out. So, I have to again divide it as into three equal groups and when I divide this, this is one strategy which I can follow. I divide them into three groups like this and again for this I do not have to assign anything for this group. There are only three source symbols, so I can assign them 0, 1, 2.

Similarly, if I extend this argument to the code alphabet size four, where my code letters will be 0, 1, 2, 3, initially to start with I have to divide my source symbols into four groups. So, I have to divide it into four almost equal groups 0, 1, 2, 2, 3, 3. So for this I have already assigned this, there is no need to assign 2, 2, I can assign 0 and 1. For this last group I have got four symbols remaining with me. So, I can assign 0, 1, 2, 3.

If you look at the calculations of the average lengths, which we get from these three codes, the entropy of the source is 2.64 bits per symbol, whereas the length for this code turns out to be 2.66. The length for this code is 1.83 and the length for this code is it turns out to be 1.45. So, on the average you will find that the l keeps on decreasing, but this (Refer Time: 22:41) code symbols per source symbols. If you look at the efficiency, these are the efficiency values which we get for these three cases.

I would like to take and illustrate the concept of how the coding efficiency improves, when I go for the higher extension of a source. We have proved the Shannon's theorem, but let us have a look at one particular illustration in order to understand the concepts, which we have studied so far in a better way.

(Refer Slide Time: 23:53)



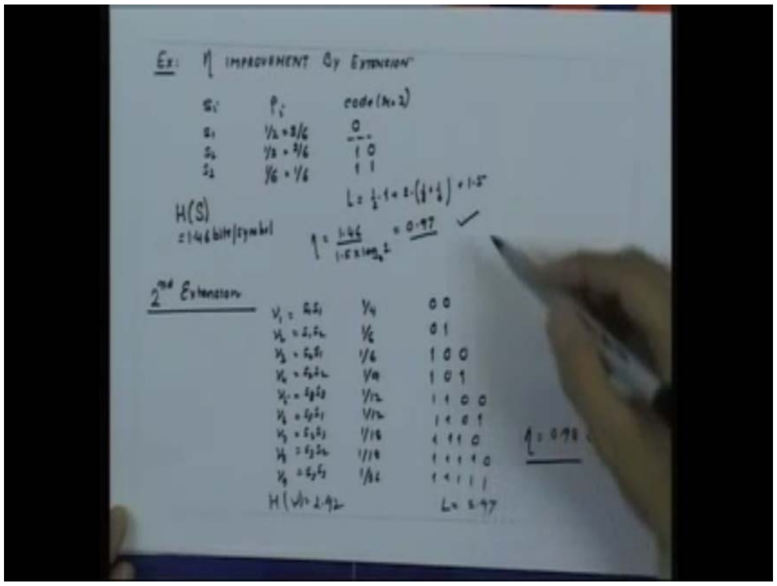
Ex: η IMPROVEMENT BY USING EXTENSION

s_i	P_i	code $(r=2)$
s_1	$3/4$	0
s_2	$1/4$	1

$H(S) = 0.811$ bit $L = 1$, $r = 2$, $\eta = 0.811$

I take another example out here where I want to show the efficiency improvement by using the extension of a source. So, let us assume that I have two source symbols given as s 1,s 2 with probabilities of occurrence are three four and one four. Simple way of a binary code I will have 0 and 1, the entropy of the source is 0.81 bit, the average length is obviously 1. This is a binary alphabet, so r is equal to 2, so efficiency turns out to be 0.811. Now let us see basically is it possible for us to improve this efficiency by going for the higher extension of the source and we will do that based on the Fano's coding strategy. If we use the Fano's coding strategy, the first thing is that I have to get the source symbols of my extension of the source.

(Refer Slide Time: 24:49)



Ex: η IMPROVEMENT BY EXTENSION

s_i	P_i	code (r=2)
s_1	$1/2 = 3/6$	0
s_2	$1/3 = 2/6$	1 0
s_3	$1/6 = 1/6$	1 1

$H(S) = 1.46$ bits/symbol

$L = \frac{1}{2}\cdot 1 + 2\cdot\left(\frac{1}{3}+\frac{1}{6}\right) = 1.5$

$\eta = \frac{1.46}{1.5 \times \log_2 2} = 0.97$ ✓

2nd Extension

$V_1 = s_1s_1$	$1/4$	0 0
$V_2 = s_1s_2$	$1/6$	0 1
$V_3 = s_2s_1$	$1/6$	1 0 0
$V_4 = s_1s_3$	$1/12$	1 0 1
$V_5 = s_3s_1$	$1/12$	1 1 0 0
$V_6 = s_2s_2$	$1/12$	1 1 0 1
$V_7 = s_2s_3$	$1/18$	1 1 1 0
$V_8 = s_3s_2$	$1/18$	1 1 1 1 0
$V_9 = s_3s_3$	$1/36$	1 1 1 1 1

$H(V) = 2.92$

$L = 2.97$

$\eta = 0.98$

In this case we will assume the second extension in the source. If I have the second extension of the source, these are the four new source symbols which I will generate from the second extension of the source, and if I assume that the source is IID, then I can calculate the probabilities of this new source symbols. This we have seen earlier how to do that. So, I get my probability of the new source symbols and based on the probabilities of the new source symbols I can design my instantaneous code which is based on the Fano's coding strategy.

I divide them again into two groups 0, 1, 1 then again two divisions 0 and finally I get this code for the second extension based on the Fano's coding strategy. Now if you look at this, the entropy of this source was 0.8,1 the entropy for this source because it is an IID the second extension of the source will be just twice of this entropy that turns out to be 1.622 and if you calculate the average length for this code will be 27 by 16.

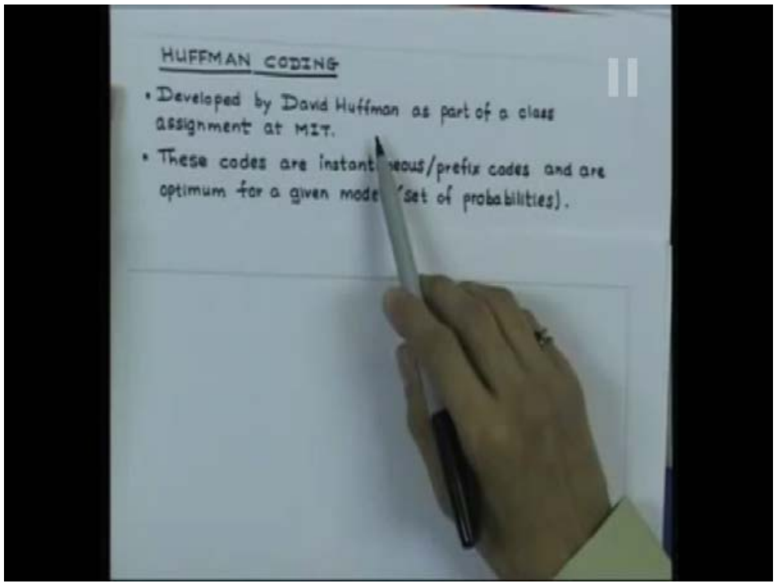
This27 by 16 and if you take 1.622 divided by 16 you will get efficiency of the code to be equal to 0.961. So what I wanted to bring forward was that by using second extension of a source I can improve the coding scheme for the source based on a particular coding strategy. In this case, it was possible for me to get this kind of improvement, but it is not always necessary that you could get this kind of large improvement by going just for the second extension. It may possible that you may have to go for higher order before you achieve the efficiency which is ideally equal to 1.

Let us take one more final illustration before we go ahead with another discussion. I have another example out here. I have source consisting of three symbols, the probabilities have been given, if I based on my Fano's coding strategy, I can get the code here, I get my entropy of the source and I get my efficiency equal to 0.97. This efficiency of 0.97 is already quite close to the expected or to the ideal efficiency which would decide that is equal to 1.

In this case if you go for the higher order of extension for the source, let us take an simple example of second extension. There will be eight nine source symbols. From the second extension of the original source, you can calculate probabilities of the new symbols out here based on the assumptions of IID and then based on the Fano's coding strategy we can get the code for this source.

If you calculate the efficiency it turns out to be 0.98. In this case again we see the entropy of the source is twice of this, but the average length comes out to be very close to the entropy of the source that is 2.97 is almost equal to 2.92. So, this example shows that if the efficiency is already very close to the ideal expected value of 1, then going for higher order may not improve very much. So, when I go for the second order extension I get an improvement of approximately 0.01. Let us have a look at the design of the compact instantaneous code, for a given probability model, for a source s based on a simple algorithm, which was developed by Huffman. In literature this is popularly known as Huffman coding.

(Refer Slide time: 29:48)

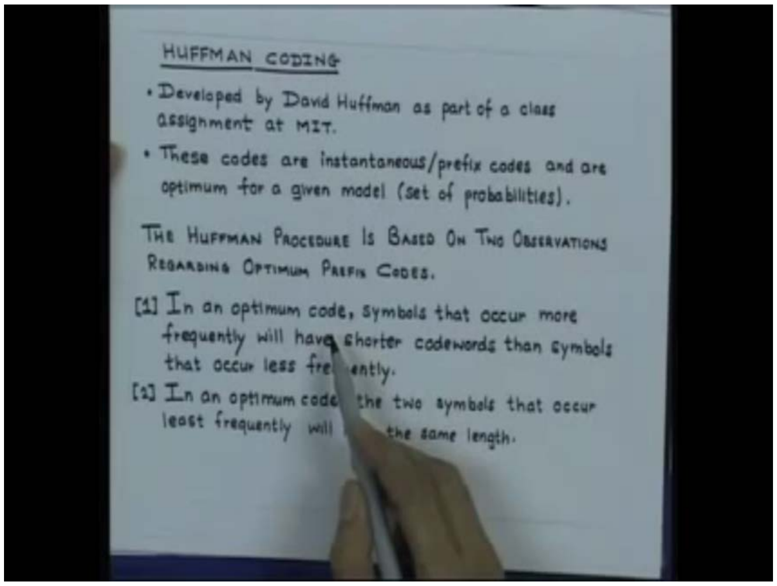


HUFFMAN CODING

- Developed by David Huffman as part of a class assignment at MIT.
- These codes are instant[illegible]eous/prefix codes and are optimum for a given mode[illegible] (set of probabilities).

Huffman coding was developed by David Huffman who was a student of Fano, as a part of a class assignment at MIT. These codes are instantaneous or prefix code and are optimum for a given model that is the set of probabilities.

(Refer Slide Time: 30:16)



HUFFMAN CODING

- Developed by David Huffman as part of a class assignment at MIT.
- These codes are instantaneous/prefix codes and are optimum for a given model (set of probabilities).

THE HUFFMAN PROCEDURE IS BASED ON TWO OBSERVATIONS REGARDING OPTIMUM PREFIX CODES.

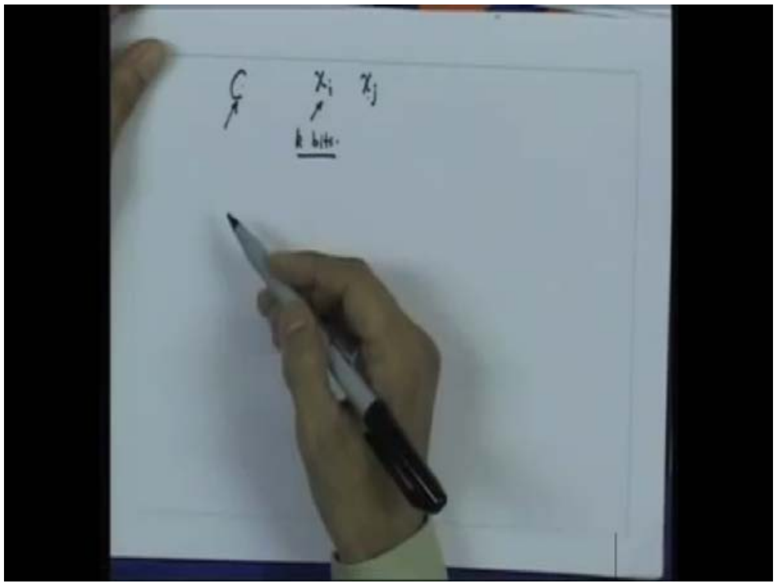
[1] In an optimum code, symbols that occur more frequently will have shorter codewords than symbols that occur less fre[illegible]ently.

[2] In an optimum code[illegible] the two symbols that occur least frequently will [illegible] the same length.

The Huffman procedure is based on two observations regarding optimum prefix codes. In an optimum code symbols that occur more frequently will have shorter code words than symbols that occur less frequently. That is the first condition which any optimum prefix code should satisfy. Another condition which an optimum prefix code will satisfy is that in an optimum code the two symbols that occur least frequently will have the same length.

It is not very difficult to see that both these observations are true. Let us look at the first observation. If these were not true and let us assume that, there was an optimum code where the symbols that occur more frequently were assigned longer code words than symbols that occurred less frequently. If we had optimum code of that kind then what that would mean is that, on the average the code word length of such an optimum code would be always larger than if the conditions were reverse. Therefore code that assigns longer code words to symbols occurring more frequently cannot be optimum. So, this is true. Now it is not very difficult to see that the second observation is also true.

(Refer Slide Time: 32:17)



To understand this let us assume that I have a code c and there are two code words x i and x j which correspond to the two list probable source symbols. If they do not have the same length let us assume that one of this that is x i is larger than x j and let us assume that this x i, the codeword is larger than this codeword by k bits. Now, if this code c is a prefix code then x j which is smaller codeword cannot be prefix of x i. It means that if I remove this k bits from x i, then I will get a shorten codeword for the source symbol corresponding to x i.

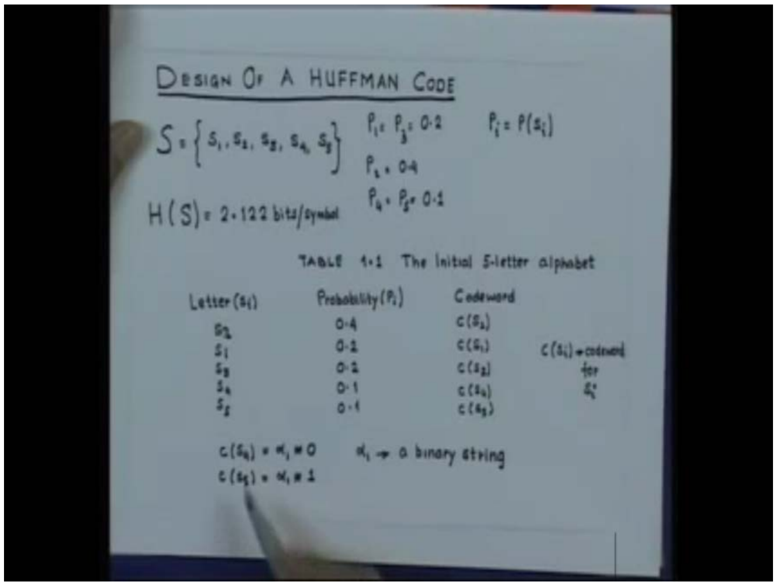
Since, x j cannot be a prefix of this code word, even on shortening it, this cannot be prefix of the new codeword which I get after removing this k bits. Since both this code words correspond to the source symbols which are the least probable ones, this code word even after shortening cannot be a prefix to another code word existing in the code c. But what will happen that after I remove this k bits from this code word x i, the average length of the new code c which I will get after removing this k bits will be smaller than the code, with which I started initially, which consistent of x i, x j with no k bits in x i compared to x j.

So the average length of the new code which I get will be smaller than this. So what it means that, the new code which I get is now an optimum code, but this actually violates our original assumption that this code c was optimum. Based on this argument, it is clear that in optimum code the two symbols that occur less frequently will have the same length. These two conditions will prove them more mathematically little later on, when we try to provide the justification of the optimality of Huffman codes.

Besides these two observations the Huffman code is basically based on one more observation or a simple requirement is added to this two observation. This requirement is that the code words corresponding to the two least probability symbols differ only in the last bit. So, this is the requirement which we add. What we mean by the requirement is that if we have gamma and delta as the two least probable symbols in an alphabet and if the code word for gamma was m with the last bit 0, again we are assuming a binary alphabet, so m denotes the code word binary string and the last code symbol in this codeword is 0.

The codeword for delta would be m star m. Here m is a string of ones and zeroes because we are considering binary code and the star denotes concatenation. This requirement does not really violate our two observations and leads to a very simple encoding procedure. Before we try to provide a formal proof of showing that Huffman code is really a compact code, let us try to understand a coding strategy based on Huffman. So, to do that I take a simple example.

(Refer Slide Time: 37:45)



DESIGN OF A HUFFMAN CODE

$S = \{s_1, s_2, s_3, s_4, s_5\}$ $P_1 = P_3 = 0.2$ $P_i = P(s_i)$

$P_2 = 0.4$

$H(S) = 2.122$ bits/symbol $P_4 = P_5 = 0.1$

TABLE 1.1 The Initial 5-letter alphabet

Letter (s_i)	Probability (P_i)	Codeword	
s_2	0.4	$c(s_2)$	
s_1	0.2	$c(s_1)$	$c(s_i) \rightarrow$ codeword for s_i
s_3	0.2	$c(s_3)$	
s_4	0.1	$c(s_4)$	
s_5	0.1	$c(s_5)$	

$c(s_4) = \alpha_1 * 0$ $\alpha_1 \rightarrow$ a binary string

$c(s_5) = \alpha_1 * 1$

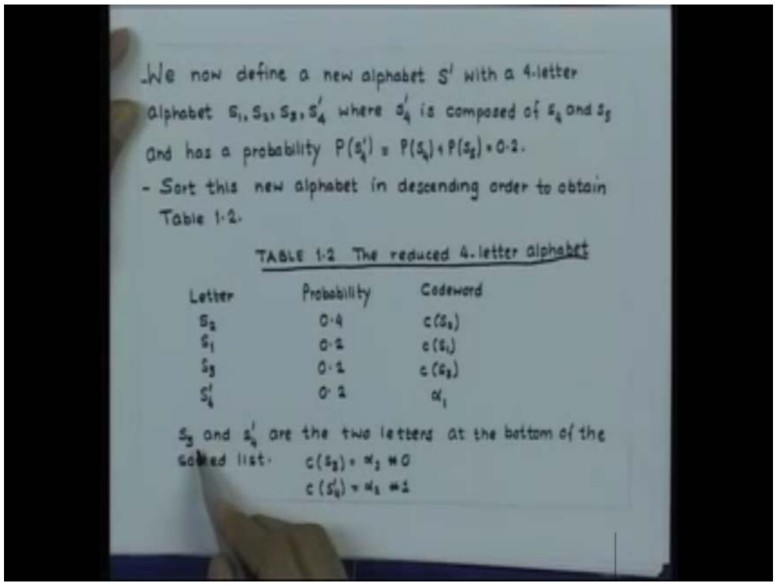
We are looking for the design of a Huffman code. I have a source consisting of five alphabets. I have been given that this five source symbols occur with this probabilities p 1 p 3 is equal to this, p i is nothing but probability of source symbol s i and entropy for this source, assuming its zero memory source is 2.122 bits per symbol. If my Huffman code is really a very efficient code I expect that my average length of this Huffman code should be very close to 2.122 bits per symbol.

Let us look into that. The first thing for coding using Huffman technique is to take the source symbols and order them in a decreasing order of probability. In this case s 2 is maximum. So this will be s 2 followed by s 1. So, we have initial five letters of the source arrange them into decreasing order of probabilities, these are the probabilities which I get and let us assume that finally I want to get the code words for this denoted by c s i. So, c s i denotes the codeword for s i. I want to get the code words for this letters.

We follow the simple strategy which we just discussed. In this case for the Huffman coding, since we are considering binary alphabet, you consider first two lowest source symbols which have the lowest probabilities and combine them. So, this is 0.1, 0.1. So, this two source symbols will differ, they will have the maximum codeword length and then differ only by one bit that is in the last position. If I assume that the c s 4 will be equal to alpha 1 concat with 0 and c s 5 will be equal to alpha 1 concat with 1, where alpha 1 is a binary string which I want to find out, what is that alpha 1.

So, this is the first step. Let us call this table 1.1 that is the initial five letter alphabet which I have got. Now, the second step is to define a new alphabet s dash. This is your s and now we define a new source alphabet s dash with the four letter alphabet obtained by combining this two source symbol s 4and s 5. This can be explained as follows.

(Refer Slide Time: 41:20)



- We now define a new alphabet S' with a 4-letter alphabet s_1, s_2, s_3, s'_4 where s'_4 is composed of s_4 and s_5 and has a probability $P(s'_4) = P(s_4) + P(s_5) = 0.2$.
- Sort this new alphabet in descending order to obtain Table 1.2.

TABLE 1.2 The reduced 4-letter alphabet

Letter	Probability	Codeword
s_2	0.4	$c(s_2)$
s_1	0.2	$c(s_1)$
s_3	0.2	$c(s_3)$
s'_4	0.2	α_1

s_3 and s'_4 are the two letters at the bottom of the sorted list.

$c(s_3) = \alpha_2 * 0$

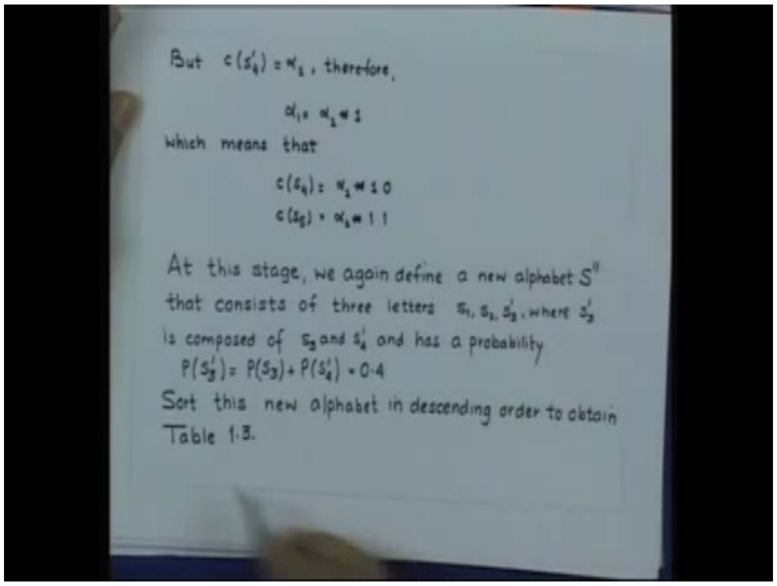
$c(s'_4) = \alpha_2 * 1$

We define a new alphabet s dash with a four letter alphabet s 1, s 2, s 3, s dash 4 where s dash 4 is composed of s 4 and s 5 corresponding to the two least probable source symbols occurring in the source. So, s 4 and s 5 are at the bottom of the order. I combine them and get a new source symbol that is called s dash 4. This new source symbol which I generate s dash 4 has a probability which is equal to probability of s 4 and s 5 that is equal to 0.2.

Once I get this new alphabet which is of reduced order, again you order this new alphabet in a descending order to obtain table 1.2. So I get a table1.2. This is a reduced four letter alphabet which I have got now. It consists of s 2, s 1, s 3, s 4 dash, s 4 dash consist of my two original source symbols s 4 and s 5. Again I order them in decreasing values of probabilities and this will remain as it is. The code word for s 4 dash would be alpha 1 because s 4 dash is a combination of s 4 and s 5, s 4 had the codeword which was alpha 1 concat with 0 and codeword for s 5 was alpha 1 concat 1. Therefore the codeword for s dash 4 would be alpha 1.

Now, s 3 and s dash 4 are the two letters at the bottom of the sorted list. This is sorted list s 3 and s 4 dash are at the bottom of the sorted list. Again going by the same strategy of Huffman coding this two source symbols, in this new reduced four letter alphabet will be the code words for this, will be the same except for the last bit. So c s 3 would be equal to alpha 2 substring concat with 0 and c s 4 dash would be equal to alpha 2 concat with 1. We already know the codeword for c s 4 dash that is equal to alpha 1. So we can equate this to alpha 1.

(Refer Slide Time: 44:42)



But $c(s_4') = \alpha_1$, therefore,

$$\alpha_1 = \alpha_2 * 1$$

which means that

$$c(s_4) = \alpha_2 * 1\,0$$
$$c(s_5) = \alpha_2 * 1\,1$$

At this stage, we again define a new alphabet S'' that consists of three letters s_1, s_2, s_3', where s_3' is composed of s_3 and s_4' and has a probability
$P(s_3') = P(s_3) + P(s_4') = 0{\cdot}4$
Sort this new alphabet in descending order to obtain Table 1.3.

We know this c s 4 dash is equal to 1. Therefore, I equate the quantity, I get alpha 1 is equal to alpha 2 concat with ,1 which means that going back substitution c s 4 is equal to alpha 2 concat 1, 0 and c s 5 is alpha 2 concat 1, 1 because based on the previous observation. At this stage we generate again a new reduced source and the new alphabet for that new source we call it as s double prime that consists of three letters s 1, s 2and s 3 prime where s 3 prime is composed of s 3 and s 4 prime because s 3 and s 4 prime were lying at the bottom of the sorted list.

So, I take this together and the probability of this new source symbol which I generate s 3 prime will be summation of p s 3 plus that turns out to be 0.4.Once I reduce this to the new alphabet s double prime, this new alphabet s double prime will have three letters and again you sort out these three letters in a descending order to obtain the table 1.3.

(Refer Slide Time: 46.24)

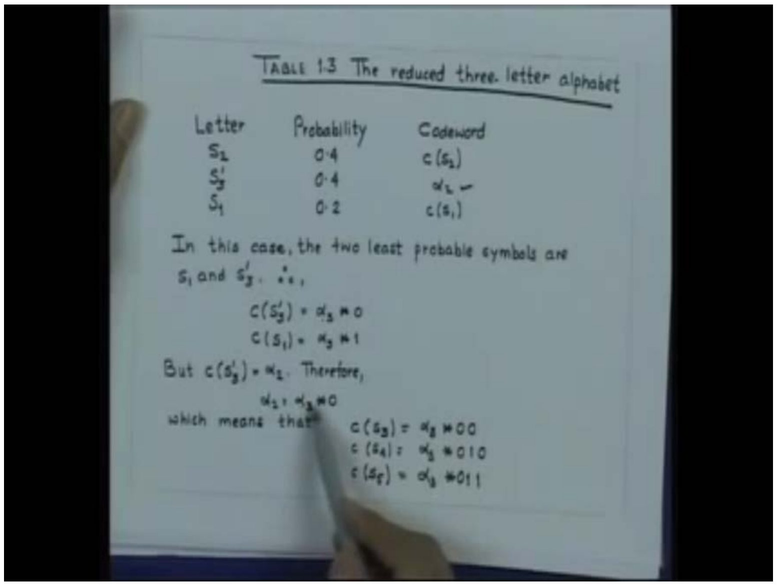


TABLE 1.3 The reduced three letter alphabet

Letter	Probability	Codeword
s_2	0.4	$c(s_2)$
s_3'	0.4	α_2
s_1	0.2	$c(s_1)$

In this case, the two least probable symbols are s_1 and s_3'. $\therefore$,

$$c(s_3') = \alpha_3 * 0$$
$$c(s_1) = \alpha_3 * 1$$

But $c(s_3') = \alpha_2$. Therefore,

$$\alpha_2 = \alpha_3 * 0$$

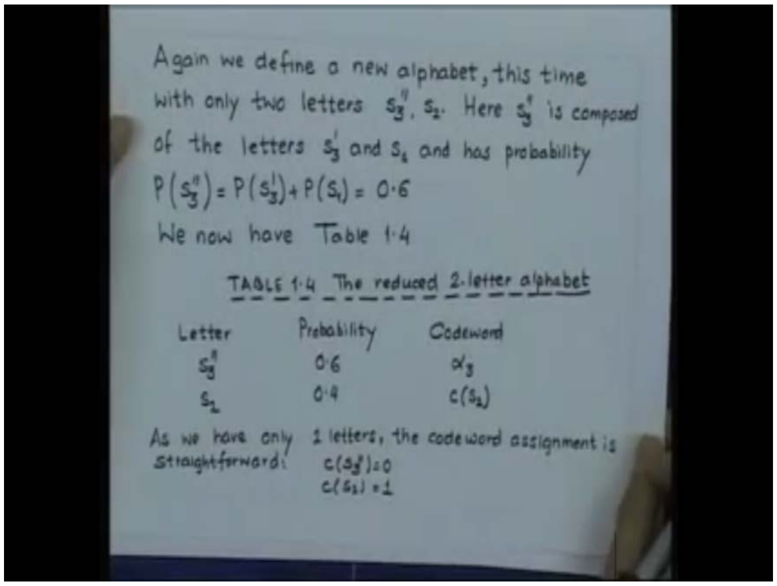
which means that

$$c(s_3) = \alpha_3 * 00$$
$$c(s_4) = \alpha_3 * 010$$
$$c(s_5) = \alpha_3 * 011$$

So, I get a table, this is the reduced three letter alphabet s 2, s 3 prime s 1 this order has changed because s 1 probability is lower than s 3 prime. So we have changed the order and this we continue with this s 3 prime the codeword based on the previous discussion would be alpha 2. Again in this case the two least probable symbols are s 1 and s 3 prime. So, both of this will differ only in the last bit. So, let us assume that c s 3 prime is alpha 3 concat with 0 and code word for s 1 would be alpha 3 concat with 1. But we already know that the codeword for s 3 prime is equal to alpha 2. So, if you substitute alpha 2 is equal to alpha 3 concat 0 and go back and substitute you will get this result codeword for s 3 codeword for s 4 and codeword for s 5.

Now, we have reached to a stage where we have the alphabet of size three. We have to go to one more step of reduction. Finally we combine these two symbols in this reduced three letter alphabet to get the reduced two letter alphabet and we combine s 3 prime and s 1 prime to get s 3 double prime.

(Refer Slide Time: 48:18)



Again we define a new alphabet, this time with only two letters s_3'', s_2. Here s_3'' is composed of the letters s_3' and s_1 and has probability $P(s_3'') = P(s_3') + P(s_1) = 0.6$

We now have Table 1.4

TABLE 1.4 The reduced 2-letter alphabet

Letter	Probability	Codeword
s_3''	0.6	α_3
s_2	0.4	$c(s_2)$

As we have only 2 letters, the codeword assignment is straightforward:

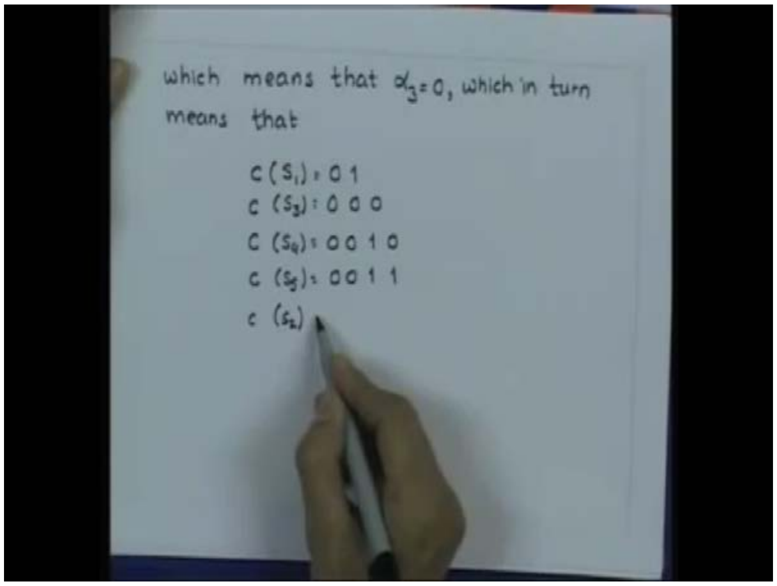
$c(s_3'') = 0$

$c(s_2) = 1$

What we do is that again we define a new alphabet this time with only two letters s 3 prime and s 2, s 2 remains intact from the previous reduced three letter alphabet, but s 3 double prime is composed of the letters s 3 prime and s 1 and the probability for s 3 double prime will be equal to probability of s 3 prime plus probability of s 1, that turns out to be 0.6. Now we have finally a source alphabet consisting of only two letters that is s 3 prime s 2 arrange that in the decreasing order. If I do that I get 0.6 and 0.4 and we know that for s3 prime the codeword is alpha 3 and for s 2 we have c s 2.

Since there are only two letters in this finally reduced two letter alphabet, the codeword assignment is straightforward. We assign c s 3 double prime as 0 and c s 2 equal to 1. So c s 2 is equal to 1 c s 3 double prime is equal to 0 which means that alpha 3 because c s 3 prime is alpha 3 which means alpha 3 is equal to 0 and since we have solved for alpha 3, we know alpha 2, we know alpha 1.

(Refer Slide Time: 49:58)



which means that $\alpha_3 = 0$, which in turn means that

$c(s_1) = 0\,1$

$c(s_3) = 0\,0\,0$

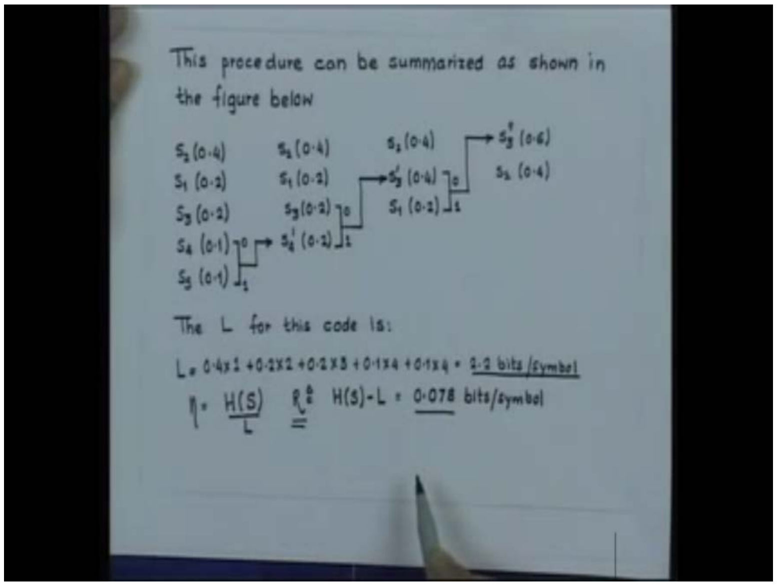
$c(s_4) = 0\,0\,1\,0$

$c(s_5) = 0\,0\,1\,1$

$c(s_2)$

We can go back and re-substitute for the code words and finally what we get is this. So, to complete c of s 2 is equal to 1. So, this is how we generate the code words for the source s based on the Huffman coding procedure and this procedure can be summarised as shown in the figure below.

(Refer Slide Time: 50:35)



This procedure can be summarized as shown in the figure below

The L for this code is:

$L = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = \underline{2.2 \text{ bits/symbol}}$

$\eta = \frac{H(S)}{L}$ $\quad \underline{R} \triangleq H(S) - L = \underline{0.078}$ bits/symbol

I order the source symbols in a decreasing order 0.4, 0.2, I combine this two I get a new source symbol, arrange them in a decreasing order again I do not change, I have kept it here. Combine this two because they are in the bottom most of the sorted list. I combine them I get probability equal to 0.4, 0.4 I move it above s 1. So, I get a new alphabet of a reduced size that is three letters, I move it up here I get this and finally I combine this two and I get s 3 double prime 0.6, 0.4. The whole procedure of what we have done for the Huffman coding gets summarised in this figure and now you can calculate the length that is the average length for this code as 0.4 multiply by 1 because s 2 is of length 1 and what I get is 2.2 bits per symbol.

If you calculate efficiency for this, it turns out to be H S by L, you can find out it will be very close to 1. There is another parameter for codes which is defined as a redundancy of a code and that by definition is equal to H S minus L and if we use the parameter of redundancy to define the efficiency or to define the compactness of a code, then this redundancy for this code which we have designed comes out to be 0.078 bits per symbol. Ideally this value is expected to be as close as possible to 0. So with this we have seen how to design a compact code based on Huffman coding procedures and this Huffman coding procedure was based on some simple observations for design of compact codes. The same Huffman coding procedure can also be looked from the point of view of tree structure and we will talk about this in our next class.