

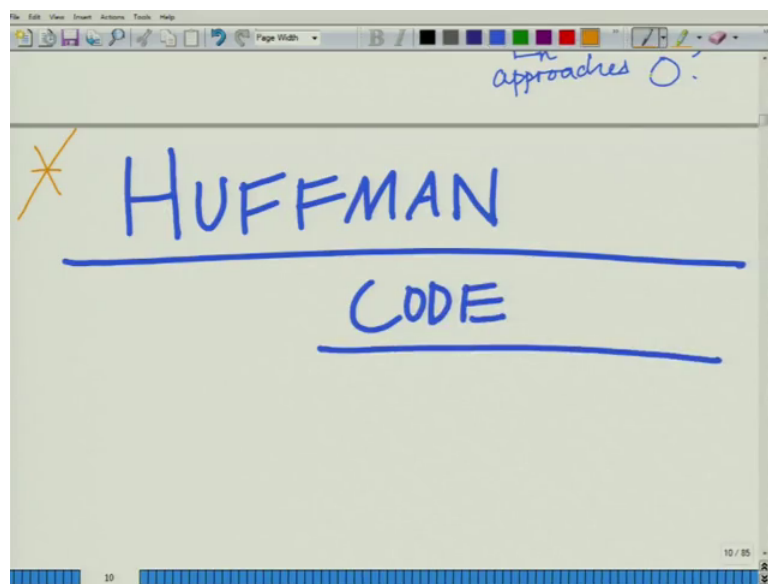
Principles of Communication Systems - Part II
Prof. Aditya K. Jagannatham
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

Lecture – 47
Huffman Code, Algorithm, Example, Average Code Length

Hello, welcome to another module in this massive open online course. So, we were looking at various aspects of source coding the fundamental bound. On the lowest possible average length of the source code for a given source and also we have seen the mechanism by which we can approach this lower bound, that is by coding symbols across larger and larger block lengths all right.

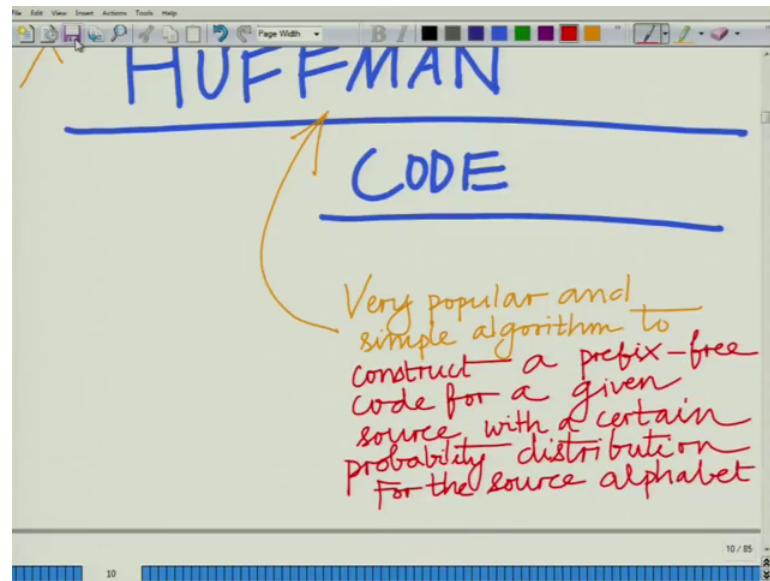
So, in this module we look at a practical scheme when algorithm to construct an efficient source code for a given source which is a very simple and elegant scheme termed as the Huffman code. So, that is the algorithm we look at in this module.

(Refer Slide Time: 00:54)



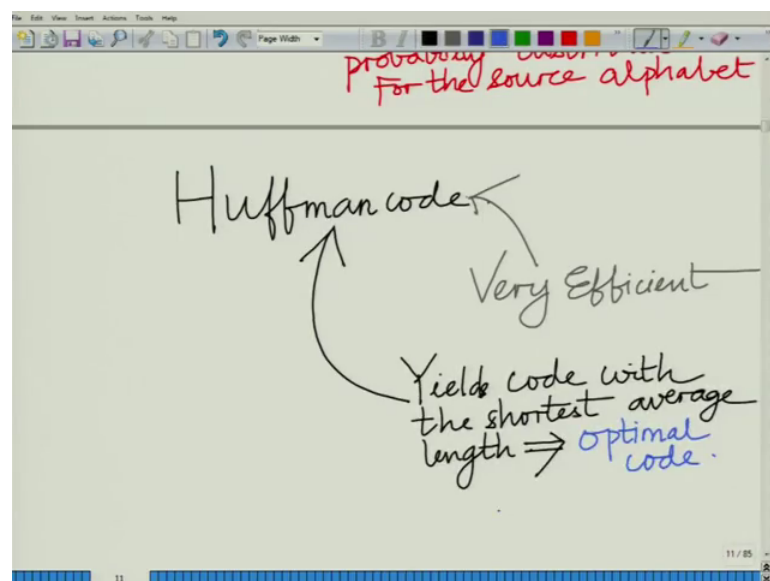
So, we would like to understand more about the Huffman code which is a very popular and efficient code. Or an algorithm to construct it is not a code, it is an very popular algorithm to construct an efficient.

(Refer Slide Time: 01:16).



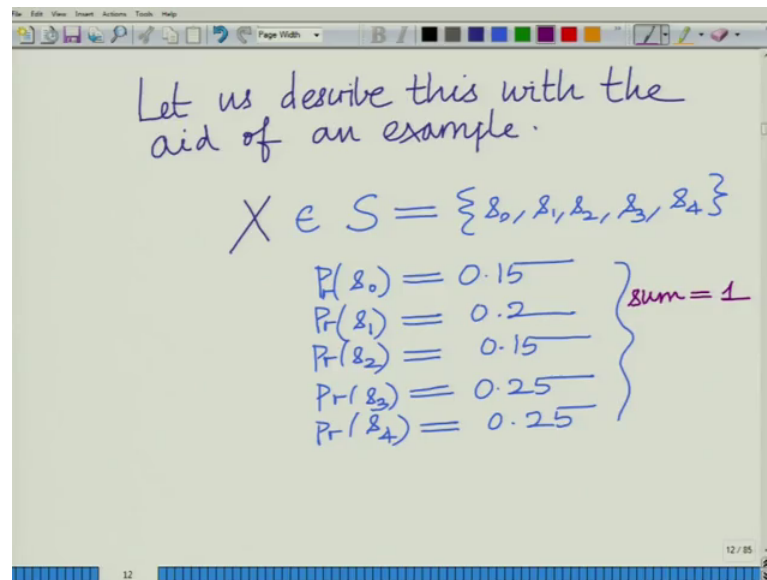
So, it is a very popular and simple algorithm to construct a prefix free code for a given source with a source alphabet and a certain probability distribution, with a certain probability distribution for the source, with a certain probability distribution for the source alphabet. In the Huffman code is the optimal code in the sense that it yields the code with the shortest average length. Or the smallest average length for a source with corresponding to that particular probability distribution. So, the Huffman code is very efficient.

(Refer Slide Time: 02:58)



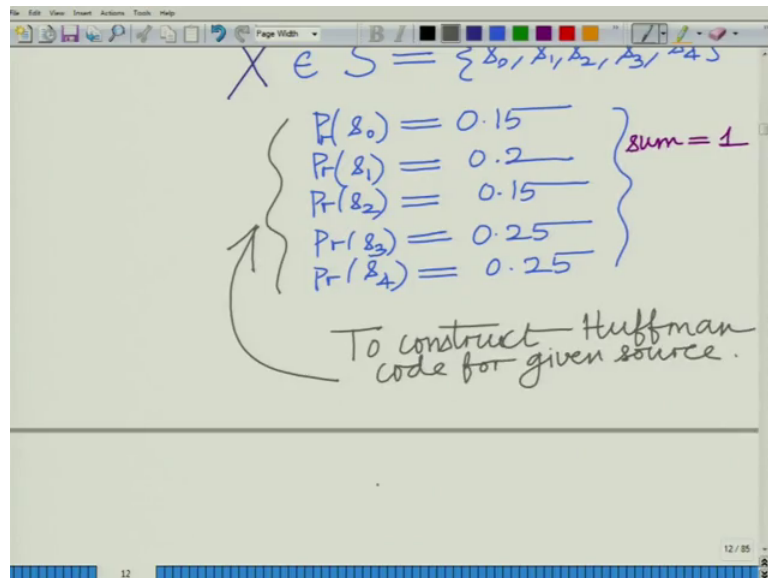
It is a very efficient code or rather a mechanism to construct a very efficient code and yields the code with the shortest, this can be shown that it yields the code, yield is a code with a shortest average length. In that sense it is the optimal code. So, it yields the code with the shortest average length in that sense it is the optimal code.

(Refer Slide Time: 04:12)



And we will illustrate this with the aid of an example. So, let us illustrate this with the aid of an example. Let us describe this with an aid of an example. Consider X symbols X drawn from a source with alphabet s_0, s_1, s_2, s_3, s_4 . And the probabilities are the probability of s_0 equals 0.15 in the probability of s_1 equals 0.2, probability of s_2 equals 0.15, probability of s_3 equals 0.25 and probability s_4 equals 0.25. And you can see that even before we start this is a valid probability distribution because sum of all probabilities, the sum of all probabilities is equal to 1. Now to start with So, what we want to do is basically we want to construct the optimal code that is a Huffman code is we want to illustrate the procedure, to construct the Huffman code for this source with this source alphabet and the probability distribution that is given ok.

(Refer Slide Time: 06:14)



$X \text{ e } S = \{s_0, s_1, s_2, s_3, s_4\}$

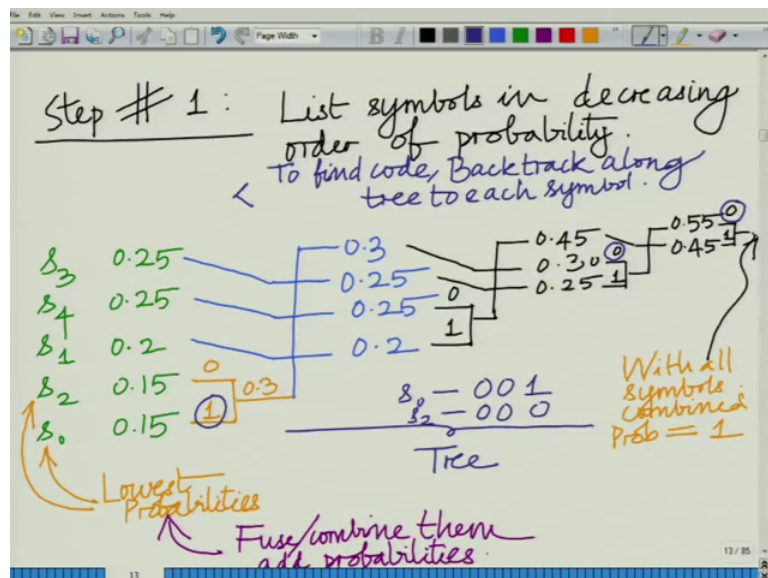
$P(s_0) = 0.15$
 $P(s_1) = 0.2$
 $P(s_2) = 0.15$
 $P(s_3) = 0.25$
 $P(s_4) = 0.25$

sum = 1

To construct Huffman code for given source.

So, that is what you want to illustrate all right. And, so we are interested in to construct the Huffman code for this source.

(Refer Slide Time: 06:42)



Step # 1: List symbols in decreasing order of probability.
To find code, Backtrack along tree to each symbol.

s_3 0.25
 s_4 0.25
 s_1 0.2
 s_2 0.15
 s_0 0.15

Tree

$s_3 = 001$
 $s_4 = 000$

With all symbols combined Prob = 1

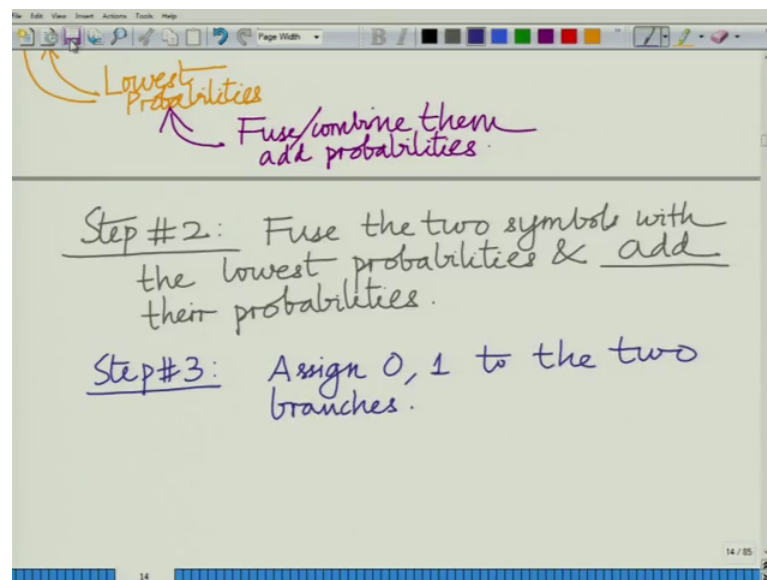
Lowest Probabilities
Fuse/combine them add probabilities.

To construct the Huffman code for the given source. Now the first step that is if we were to write this in steps for clarity, list symbols in decreasing order of, list symbols in decreasing order of probability. List the symbols in decreasing order of probability. So, you can see that the probabilities are 0.5, 0.15, and 0.2. So, if I were to list the symbols in decreasing order of probability, I can arrange them as well, s_3 and s_4 which have

probabilities let me write that as much to the left as possible as we are going to see shortly in that we are going to keep progressing towards a right So, $s_1, s_1 s_2 s_0$. So, this is 0.25, s_4 probability 0.25, s_1 probability 0.2, s_2 which has the next lower probability of course, s_2 and 0 have the lower probabilities which are 0.15 each ok.

Now, we are what we do is we fuse or combine 2 symbols with a lowest probabilities. So now, if you look at this s_2 and s_0 have the lowest probabilities. These have the lowest probabilities. So, in the first in the next step, these have the lowest probabilities. So, therefore, what we do is we fuse or basically we combine this. So, we combine this. We fuse this 2 assign a 0 to one branch let us to the top branch one to one branch, then net probability becomes 0.3.

(Refer Slide Time: 09:20)



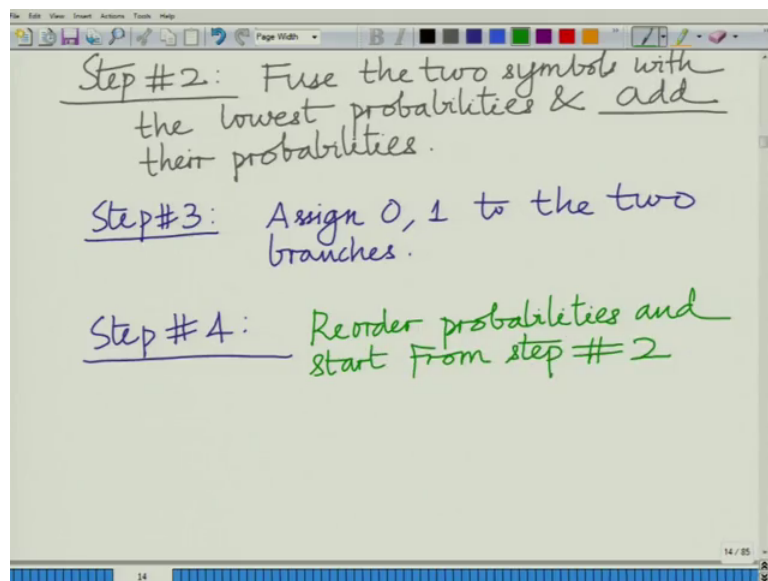
And. So, we fuse or combine this 2 symbols and add their probabilities. So, we are taking the lowest probability symbols fuse or basically combine them and add their probabilities. So, in step number 2 what we are doing in step number 2 is we take the symbols fuse combine or fuse the 2 symbols with the lowest probabilities. And we add their probabilities, that is the most important thing we add their corresponding probabilities.

So, we are fusing these 2 symbols which have the lowest probabilities and then we are corresponding adding, we are adding their corresponding probabilities. And so, to one of the branches we are assigning 0, to the other branch we are assigning a 1. And if the

significance of that we are going to seeing slightly later. Now what we do, so we had assigning a 0 to one branch assign a 1 to another branch. So, we assign 0 or 1 to the 2 branches. And now what we do, now we have these two. So, we have the original symbols s_3 s_4 s_1 and this fused symbols which is s_2 and s_1 .

Now we repeat the same step, now we go to step one, they is resulting symbols. So now, you have to look at the effective symbols set that we have we have the infuse symbols s_3 s_4 s_1 , that is infuse symbols at stage one and the fused symbols that is s_2 and s_0 . Now again reorder their probabilities fuse the symbols with a lowest probabilities, to branches assign 0 to one branch, 1 to another branch. Proceed until all the symbols are fused in to a one effective symbol. And naturally when you fuse all the symbols since you adding their probabilities the total probability of the next few symbol remaining the end will be 1 that is where we stop this ok.

(Refer Slide Time: 12:36)



So, now step number 4 we reorder order the probabilities this is very simple reorder the probabilities and start from step 2. So, reorder the probabilities start from step number 2 that is by fusing the probabilities corresponding fusing the symbols with a lowest probabilities and adding their corresponding probabilities. Now if you look at this stage, in this stage the probabilities that we have our 0.25, 0.25, 0.2 and the fuse symbol with 0.3. Now naturally this fused symbol with probability 0.3 will go to the top because, it now has a highest probability by virtue of the addition of probability. So, that will go to

the top. So, that will go here, that will be 0.3 and the rest of the symbols for instance will come down when you order them. So, s after that you have s 3 after that you have s 4 finally, you have s 1, which is 0.2.

Now, the lowest probability symbols are one their s 1 with probability 0.2 and s 4 with probability 0.25 well, you can either choose s 4 or s 3 it does not matter both have probability 0.2 five. So now, you fuse well, now you fuse a s 1 and s 4 assign 0 to 1 branch, 1 to another branch, it becomes 0.55, 0.45 the net probability. So, that again when you reorder it goes to the top. So, this is 0.45 and then you have 0.3 which will be the next one and then you will have 0.25 which is s 3. Now the lowest probabilities are s 3 and s the lowest probabilities are 0.3 which corresponds to remember the fuse symbol s 2 and s 0 and of course, s 3 with 0.25. So now, you fuse these 2.3 and 0.25 assign 1 to one branch 0 to one branch, 1 to another branch this becomes 0.55 you have 0.45 which is another fused symbol and now when you fuse it becomes, and again you assign 0 to one branch, 1 to another branch and then you get the effective fuse symbol this completes the algorithm.

So, this is where all symbols fused with all symbols combined. So, the probability of that, since we are adding all the probabilities in the final stage we will get a fusion of all the symbols correct and the probability of that will be 1 all right. Since we are adding all the probably, since we are fuse we adding all the probabilities now. So now, that completes the process of constructing this Huffman tree.

So, this has completed if you can see the tree, and now to find the code you back track along the tree. So, to find the code back track along the tree, back track along tree to each alphabet or each symbol, correct? You back track along the tree to each symbol. For instance, to find the code corresponding to let say symbol s naught well s naught will be s naught remember s naught is going through 1 and then it is going through the top to 0.3 finally, 0.3 the branch 0.3 is assign 0 here and finally, it is going in to 0.55 which is assigned another 0, so the code for s naught. So, the code for s naught will be, the code for s naught will be 0, 0, 1 that is if you back track this find the path to. So, you start with 0, 0, 1. So, s naught corresponds to 0, 0, 1.

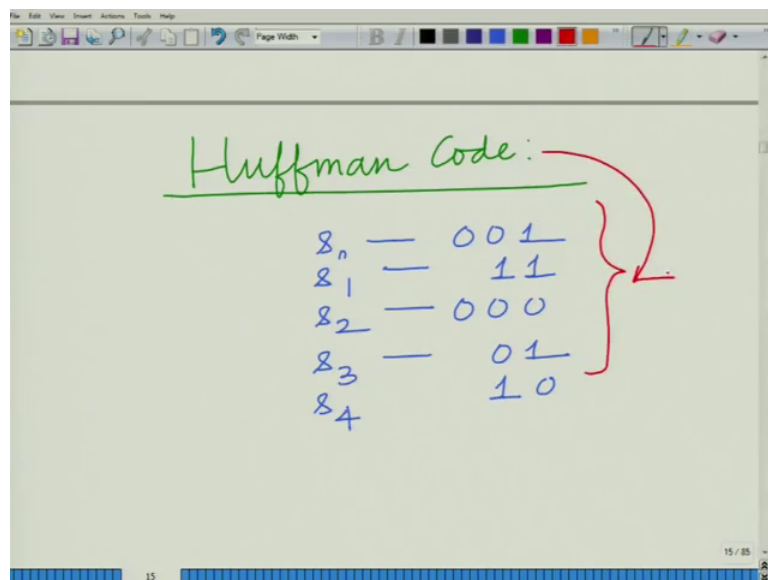
Let me repeat this procedure again if you trace the path from s naught to this root node, you can see you start with s naught the corresponding branch has 1 then the fusion

results in probability 0.3, 0.3 goes over here. Now in this stage where you are fusing 0.3 and 0.25, 0.3 is assign 0. Again it goes in to 0.55 and in this stage finally, when you are fusing 0.55 and 0.45, 0.55 is assign 0.

So, now if you back track along this path to the path of 0, what you will have is 0, 0, 1. Similarly if you back track along the path to path s 2, s 2 will be 0, 0, 0. So, basically while back tracking right, looking at the path and looking at the bits on the path that eventually leads to So, we have what we have doing is we are back tracking starting from this effective symbol because probability 1. We are back tracking the symbol to back tracking the route or back tracking the path along this tree this Huffman tree to each symbol. And along the way we are collecting all the bits that we are observing on the various branches. And that basically forms the code word for that particular symbol I have to disc describe it in very simple topic.

Basically the sequence of bits along the paths starting from this effective towards fused symbol towards each of these, towards each of this symbols in the alphabet all right. We have seen s 0 is 0, 0, 1 s 2 is 0, 0, 0 and you can similarly formulate the list.

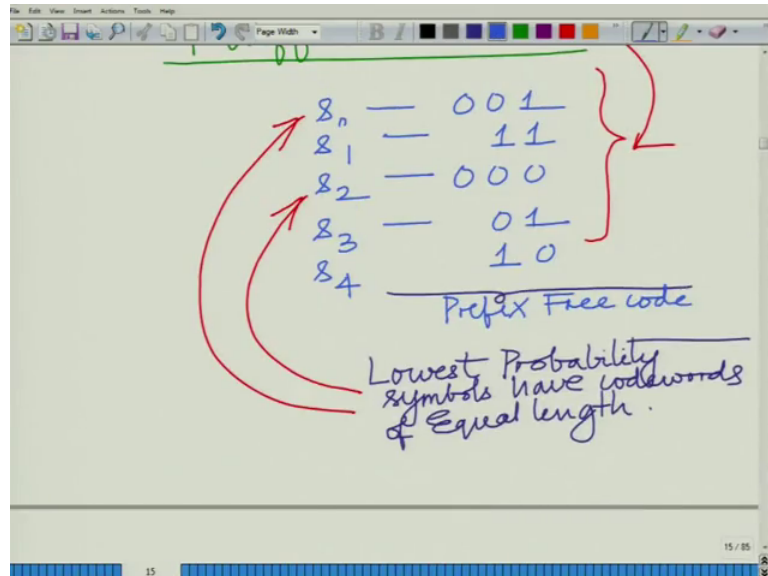
(Refer Slide Time: 19:58)



So, the final code Huffman code will be the Huffman code that has been finally, designed is basically for s 0 we have 0, 0, 1 for s 1 we have 2 bits 1, 1. You can find this from the tree for s 2 we have already seen we have 0, 0, 0. S 3 we have 0 1 and s 4 is represented

using the 2 bits 1 0 and you can observe. So, this is basically your Huffman code. And you can observe that the 2 lowest probabilities.

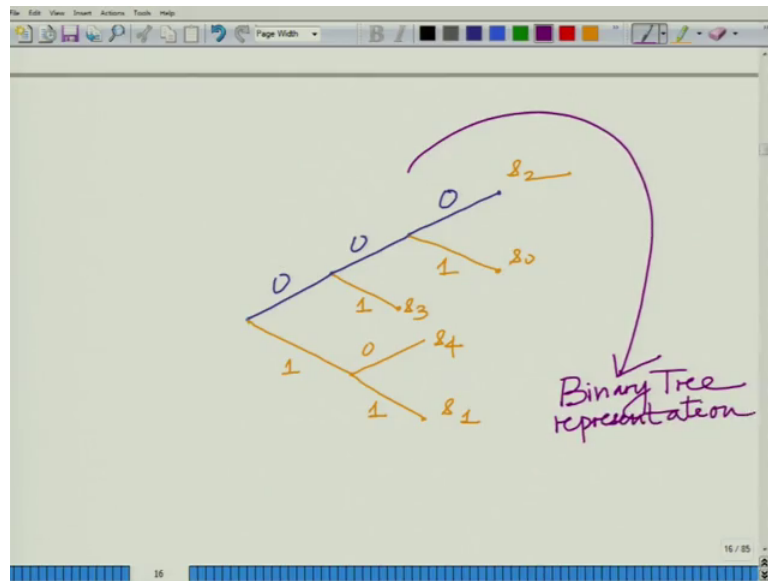
(Refer Slide Time: 20:57)



What is interesting thing this is the property of the Huffman code, which is always true is that the 2 symbols lowest probability symbols have code word lowest, probability symbols have code word of code words of equal length. So, the lowest probability symbols have code words of equal length.

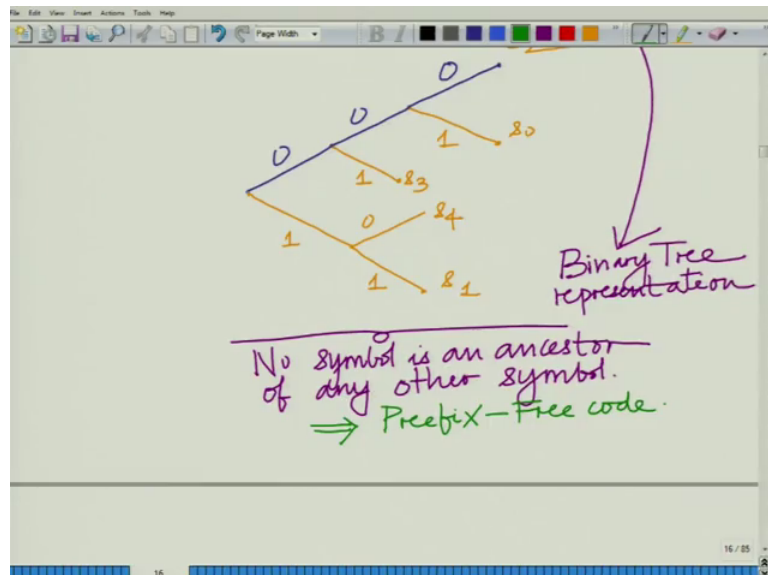
So that is what is an interesting property of the Huffman code. The lowest probability 2 symbols with the lowest probabilities have equal lengths. And now what we are going to do is, let us compute the average code length for this code word for the Huffman now. Before that let us first verify that this is a prefix free code. And we know how to verify that this is a prefix free code so, want to of course. So, let us check that this is a prefix free code so, we have s naught.

(Refer Slide Time: 22:21).



So, let say 0 represents a top branch. Remember we can always verify prefix free code using the binary tree representation, so 0, 0, 0 that gives s naught. 0, 0 I am sorry, 0, 0, 1 gives s naught, so 0, 0, 0 gives s 2. 0, 0, 1 gives s naught. 0 1 this gives s 3 and 1 0 this gives s 4 and 1, 1 gives s 1.

(Refer Slide Time: 23:30)



So, this is a binary tree representation for the code that we have constructed. And observe from this binary tree that, no code word or no symbol, no symbol are ancestor. No symbol is an ancestor of any other symbol that is no symbol lies along thus, that is no

symbol lies along the path to another symbol in this binary code tree which implies this is a prefix free code. Or an instantaneous code which implies this is a prefix free code. And we would like to calculate.

(Refer Slide Time: 24:35)

A handwritten table titled "Average Code Length" is shown on a digital whiteboard. The table has four columns: a column for symbols (s_0 to s_4), a column for binary codes, a column for code lengths, and a column for probabilities. The data is as follows:

Symbol	Code	Length	Prob
s_0	001	3	0.15
s_1	11	2	0.2
s_2	000	3	0.15
s_3	01	2	0.25
s_4	10	2	0.25

Now, what is average code length to calculate the average code length let us write this table. So, we would like to calculate what is the average code length. So, to calculate the average code length let us start with a table. So, this is your symbol s_0 s_1 s_2 s_3 s_4 and the code. And the code corresponding code is s_0 is 0, 0, 1 s_1 is 1, 1 s_2 is well 0, 0, 0 s_3 is 0 1 and s_4 is 1 0. And which means the length that is also obvious s_0 is length 3 bits, s_1 is length 2 bits, s_2 has length 3 bits, s_3 has length 2 bits, s_4 has length 2 bits. And finally, let us light the probability of each we know what are the probabilities of these different code words, what are the probabilities. So, the probability of s_0 is 0.15, probability of s_1 is 0.2, probability of s_2 is 0.15 s_3 and s_4 have probabilities 0.25 each.

(Refer Slide Time: 26:24).

The whiteboard shows a table with two rows and four columns. The first row is labeled s_3 and contains the values 01, 2, and 0.25. The second row is labeled s_4 and contains the values 10, 2, and 0.25. Below the table, the average code length is calculated as follows:

$$\text{Average code length} = \sum_{i=0}^{M-1} p_i l_i$$

$$= 3 \times 0.15 + 0.2 \times 2 + 0.15 \times 3$$

$$+ 0.25 \times 2 + 0.25 \times 2$$

$$= 0.45 + 0.4 + 0.45$$

$$+ 0.5 + 0.5$$

$$\bar{L} = 2.3 \text{ bits/symbol.}$$

So, therefore, average length equals summation i equal to 0 to m minus 1 $p_i l_i$ which in this case would be 3 into 0.15 plus 0.2 into 2 plus 0.15 into 3 plus 0.25 into 2 plus 0.25 into 2. Which if you calculate is basically 0.45 plus of course, point 4 plus 0.45 plus 0.5 plus 0.5 and you can clearly see this is basically 1, 1.4 plus 9 this equal to 2.3 bits per symbol. That is your \bar{L} for the Huffman code, this is the what we have just calculated is basically your average length. So, this is the average length for the Huffman code.

(Refer Slide Time: 27:53)

The whiteboard shows the calculation of entropy $H(X)$ for the Huffman code. It starts with the average length $\bar{L} = 2.3$ bits/symbol, which is circled in green. Below it, the entropy is calculated as follows:

$$H(X) = 2 \times 0.15 \times \log_2\left(\frac{1}{0.15}\right)$$

$$+ 0.2 \times \log_2\left(\frac{1}{0.2}\right)$$

$$+ 2 \times 0.25 \times \log_2\left(\frac{1}{0.25}\right)$$

$$H(X) = 2.2855$$

So, this is the average length for the Huffman code. To see remember, to see how close this is the lower bound of any such code, not just the Huffman code, but for any prefix free code is given by the entropy. So, to see how close it is to the lower bound let us calculate the entropy. And the entropy is given as this is equal to 2 symbols with probability point 0.5. So, twice into 0.15 log to the base 2 1 by 0.15 plus 0.2 one symbol with probability 0.2 log to the base log to the base 2 1 over 0.2 plus twice into 0.25 into log to the base 2 one over 0.25 which is equal to 2.2855, this is the lower bound from the entropy. And you can see the Huffman code is tantalizingly close to this.

(Refer Slide Time: 29:19).

The image shows a whiteboard with the following handwritten content:

$$L - H(X) = 2.3 - 2.2855$$

$$= 0.0145 \text{ bits/sym}$$

Below the calculation, there is a red arrow pointing to the result and the text: "0.0145 bits/symbol from lower bound."

Below that, in blue ink, it says: "Huffman code very efficient. By coding over larger and larger block lengths achieve average code length arbitrarily close to Entropy."

If you can look at $L - H(X)$ that is equal to 2.3 minus 2.2855 which is equal to you know 0.0145, 0.0145 bits per symbol. So, the Huffman code is only 0.0145 bits per a symbol from the lower bound which is given by the entropy. And therefore, the Huffman code so, for this shows that even for this simple code. So, this shows that the Huffman code is very efficient it is only 0.014 or approximately 0.015 bits from per symbol from the lower from the lower bound. Further you can reduce now, remember to approach this become arbitrary close to the lower bound one has to code over larger and larger block lengths all right.

So, by employing the Huffman code or this scheme for constructing the Huffman code over larger and larger block lengths one can be, one can achieve an average code length which is arbitrarily close, arbitrarily in the sense that is achieve an average code length

which is as close to this lower bound that is entropy as desired. So, by coding over larger and larger block lengths achieve average code length that is arbitrarily close.

Achieve average code length that is arbitrary. So, by coding over larger and larger block lengths one can achieve an average code length that is arbitrarily close to the lower bound which is the entropy of source. So, basically in this module we have seen the Huffman code which is an elegant and a very simple scheme to construct an optimal code, all right and which is very close to the lower bound that is. So, we have illustrated this using an example constructed an Huffman code verified that it indeed gives a prefix free code, all right we have seen the procedure in detail and also we have seen that the average code length is fairly close to that given by the lower bound that is the entropy.

And one can further decrease the gap between the average code length and entropy by using this procedure the Huffman code over larger and larger blocks lengths, as we have seen on the previous modules that is once you code over larger and larger block lengths one can achieve an average code length that is as close to the lower bound as desired. So, we will stop here and look at other aspects in subsequent modules.

Thank you very much.