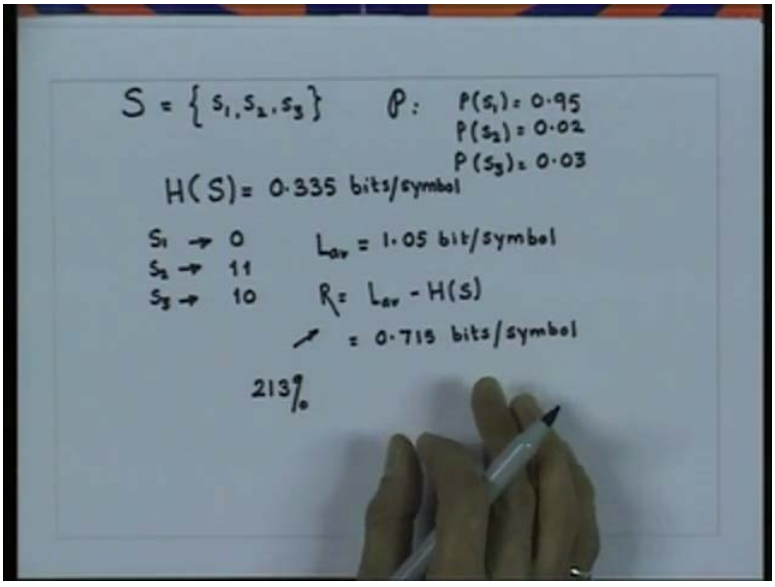
Information Theory and Coding
Prof. S. N. Merchant
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Lecture - 17
Shannon-Fano-Elias Coding and Introduction to Arithmetic Coding

In our study so far we have seen that Huffman code is optimal and it has an expected length within one bit of the entropy of the source. Redundancy which is defined as the difference between the average length of the code, and the entropy of the source for the Huffman codes can be reduced by blocking symbols together. Blocking two symbols together means alphabet size goes from m to m square where m is the size of the alphabet. So, if m is equal to 3 then when we block two symbols together the new alphabet size will become 9.

This size is not an excessive burden for most of applications. However, if the probabilities of the symbols or letters of the source are unbalanced then it would require blocking many more symbols together before the redundancy is reduced to an accepted level. So, as we block more and more symbols together the size of the alphabet grows exponentially and in such a situation the Huffman coding schemes become impractical. Therefore, it is mandatory in such situations to look for other techniques than Huffman coding. Let us try to clear these concepts with an help of an example.

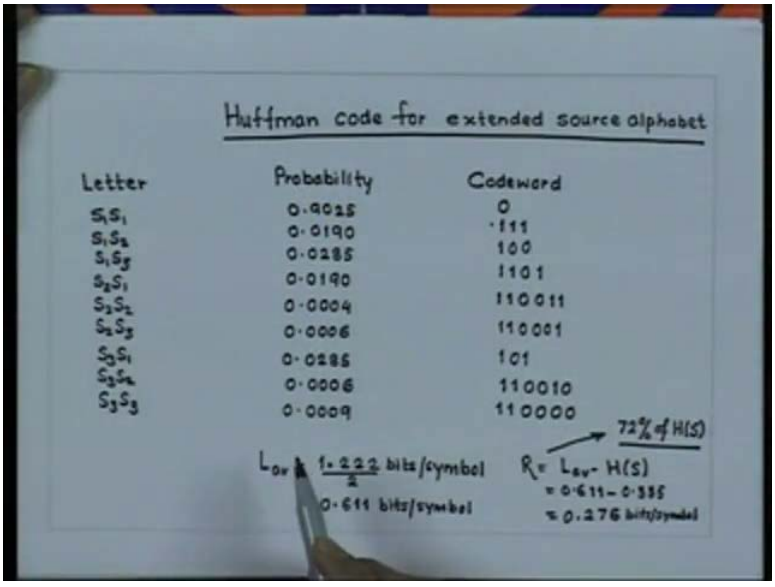
(Refer Slide Time: 02:36)



So, let us consider a source which puts out identically independent distributed letters from the alphabet consisting of three symbols. Let us assume the probability model for this source given as probability of S 2 is equal to 0.02 and probability of S 3 is equal to 0.03. Now, the entropy for this source can be calculated and this is equal to 0.335 bits per symbol. If we considers symbols individually than we can find the Huffman code for this source as S 1, the code word is 0, for S 2 the code word is 11 and for the S 3 the code word is given as 10.

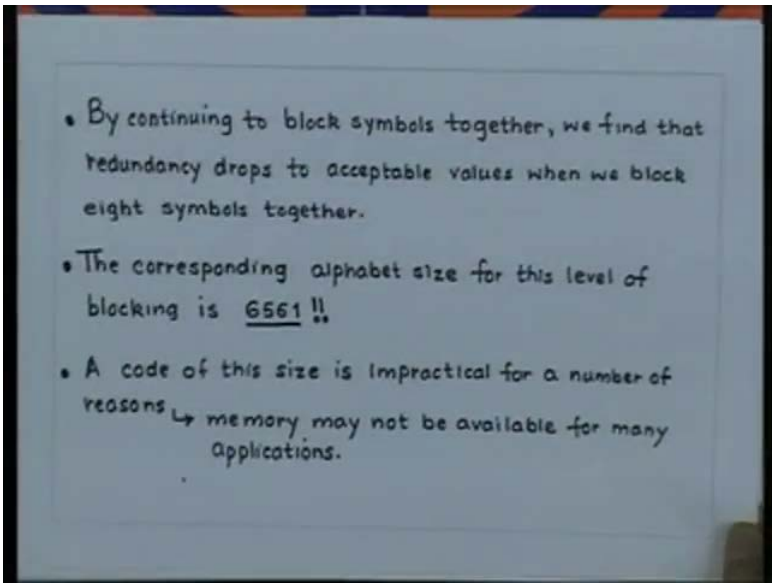
Now, the average length for this code is equal to 1.05 bits per symbol. Redundancy which is defined as the difference between the average length minus the entropy in this case will be equal to 0.715 bits per symbol. So, this is, this entropy, this redundancy is about 213 percent of the entropy. This means that the, that to code this sequence we would need more than twice the number of bits promised by the entropy. Now, same shows if we block the symbols in blocks of two and then code, we can find the Huffman code for the second extension of the source and that is as shown here.

(Refer Slide Time: 05:16)



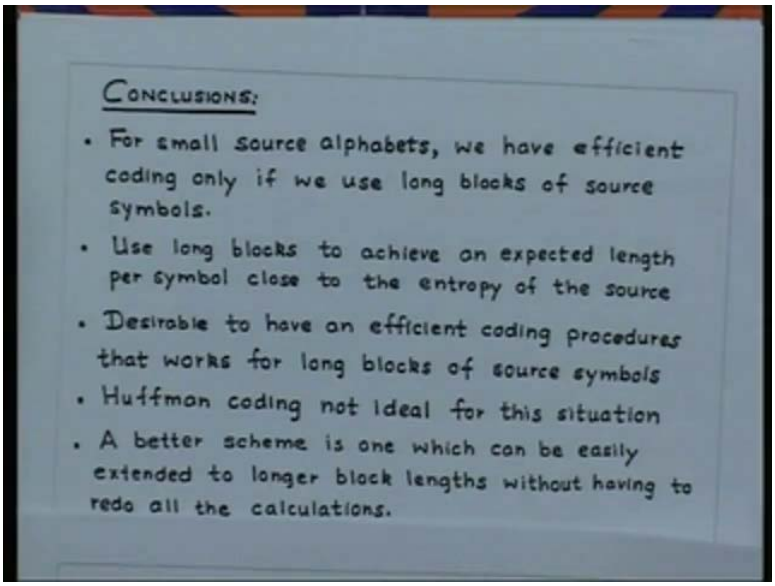
So, we take the second extension of the source, we calculate the probabilities of the new symbols and we can find the code, Huffman code for this new source. And for this new source we find out that the average length comes out to be in terms of the original source as 0.611 bits per symbol. So, redundancy is equal to 0.276 bits per symbol. Still this redundancy is 72 percent of the entropy.

(Refer Slide Time: 06:01)



But what this example shows is that by continuing to block symbols together we find that redundancy drops to acceptable values when we block eight symbols together. The corresponding alphabet size for this level of blocking is 6561 which is quite excessive. So, a code of this size is impractical for a number of reasons. First of all memory may not be available for many applications. Now, even if it were possible to design reasonably efficient encoders, but decoding a Huffman code of this size would be highly inefficient and time consuming. Now, next reason for not using this kind of code is if there were some perturbations in the source model then we would have a major impact on the efficiency of the code.

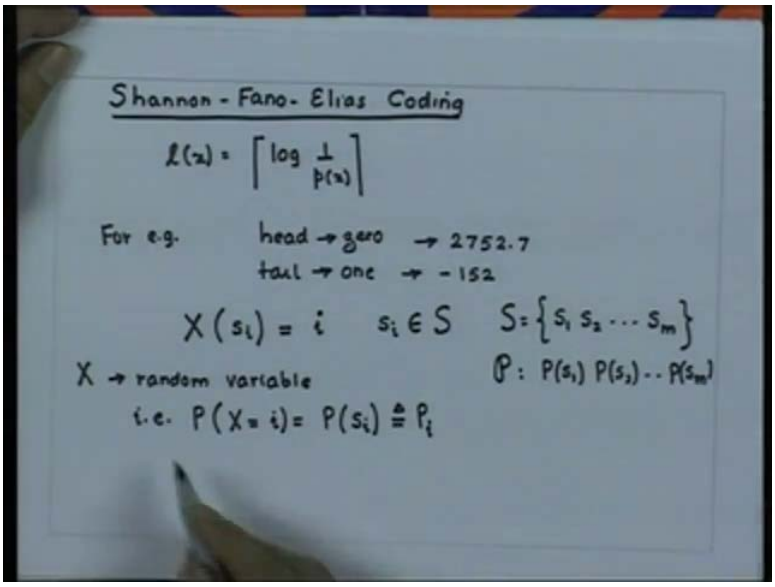
(Refer Slide Time: 07:26)



So, with this example we can reach to these conclusions that for small source alphabets we have efficient coding only if we use long blocks of source symbols. That is only if we go for higher extension of the source. If you want to achieve high efficiency or reduce the redundancy in the code then we have to use long blocks in order to achieve an expected length per symbol close to the entropy of the source. Now, this is, therefore it is desirable to have an efficient coding procedures that works for long blocks of source symbols.

Unfortunately, Huffman coding is not ideal for this situation since Huffman coding is a bottom up procedure that requires the calculation of the probabilities of all source sequences of a particular block length and the construction of the corresponding complete code tree. We are then limited to using that block length. A better scheme is one which can be easily extended to longer block lengths without having to redo all the calculations. And one such way to achieve this is to use arithmetic coding, but before we study arithmetic coding let us have a look at a simplified version of arithmetic coding which is based on what is known as Shannon-Fano-Elias coding. Then we will extend the concepts of Shannon-Fano-Elias coding to arithmetic coding.

(Refer Slide Time: 09:36)



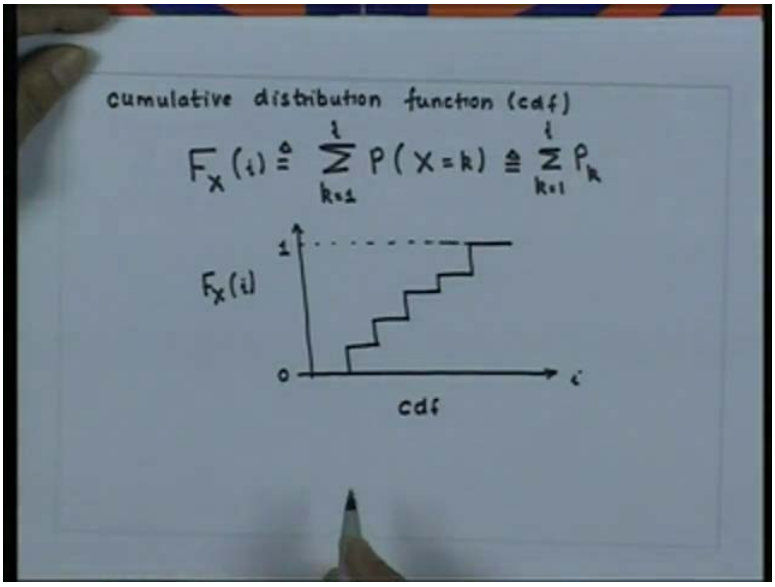
So, we will have a look at Shannon-Fano-Elias coding. We have seen that the set of lengths given as l x is equal to log of 1 by p x can be used to construct a uniquely decodable code for the source. Now, we will consider a simple constructive procedure which uses the cumulative distribution function to allot code words. And this is the idea which is used in Shannon-Fano-

Elias coding. Before we begin development of arithmetic coding or Shannon-Fano-Elias coding let us established some notation. Recall that a random variable maps the outcomes or set of outcomes of an experiment to values on the real number line.

For example, in a coin tossing experiment the random value variable could map a head to 0. So, I could have a mapping as head to zero and a tail to one or I could use a mapping which is head has some number let us say 2752.7 and tail as minus 152. So, to use this technique we need to map the source symbols or letters from the source to numbers. For convenience in the discussion to follow we will use the following mapping. The mapping is given by X and it maps the source symbol or source letter as equal to i where Si belongs to source alphabet.

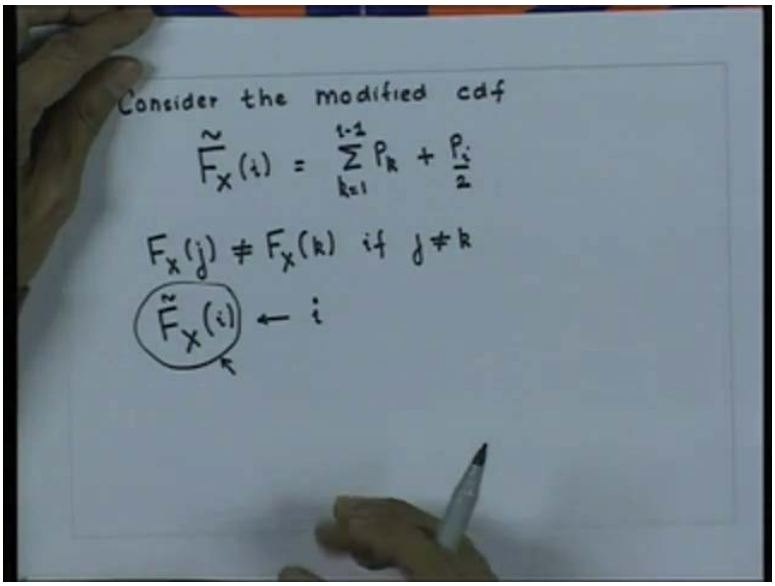
So, your S consists of source letters. This is a discrete source and X is a random variable. Now, when we use this kind of mapping given a probability model P for the source we also have a probability density function for the random variable X that is once we specify the source model. I can get my probability distribution function for the random variable for the discrete random variable as probability of X equal to i is equal to probability of Si, and this for our convenience will indicate it as P suffix i. So, probability model has been given to us as P of S 1 P of S 2 P of S m. Once, we are given the probability distribution function for the discrete random variable X, or you can say also probability mass function for the discrete random variable X, then we can define the cumulative distribution function.

(Refer Slide Time: 14:25)



So, we will define cumulative distribution function that is c d f as probability of X is equal to k. k ranges from1 to i and this by definition is P k k equal to 1 to i. Now, this function is illustrated in the figure here. So, I have my eye on x axis and so on y axis I have my c d f and on x axis the random variable. So, this is your c d f. The minimum value for this is 0 and the maximum value corresponds to 1. Now, let us consider the modified c d f as…

(Refer Slide Time: 16:24)

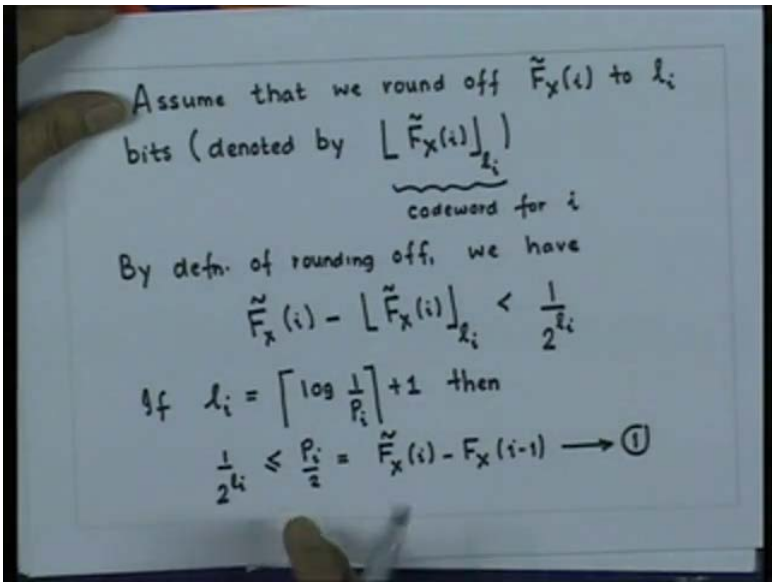


So, consider the modified c d f as F of X i, we will use tilde notation for modified c d f is equal to k is equal to 1i minus 1 plus Pi by 2 where this modified c d f denotes the sum of the probabilities of all symbols less than i plus half the probability of the symbol i. Since, the random variable is discrete the cumulative distribution function consists of steps of size Pi. The value of the function of this modified that is modified F X i tilde is the midpoint of the step corresponding to i. Now, since all the probability are positive what this implies that F X j is not equal to F X k if j is not equal to k.

And hence, we can determine i if we know FX tilde i. So, merely look at that graph of the c d f and find the corresponding i. So, because this is unique by looking at the graph we can once the F Xi is been given we can find out what is the value of i. So, what this implies that the value of modified c d f can be used as a code for i. Now, in general this value is a real number and which is expressible in a practical scenario only when finite number of bits. So, it is not efficient to use the exact value of this modified c d f as the code word for i. We will have to

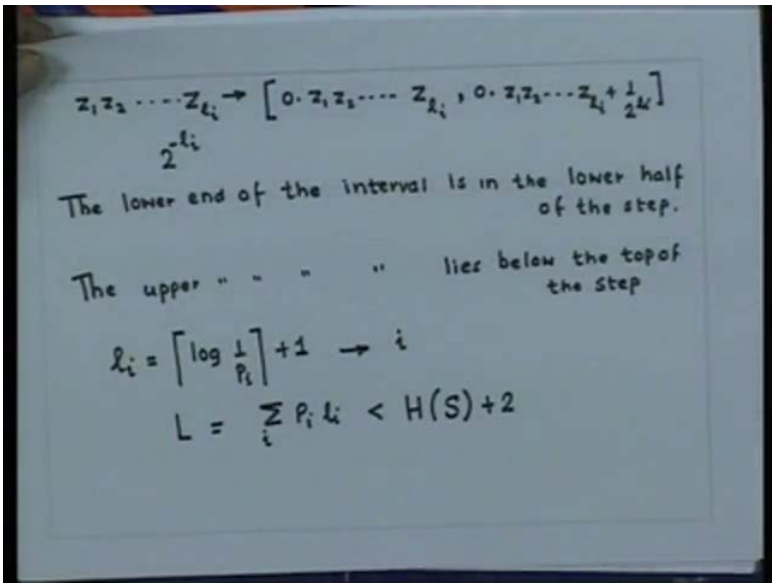
use an approximate value for this modified c d f. So, if we use approximate value the next question is what is the required accuracy? So, let us assume.

(Refer Slide Time: 20:04)



So, let us assume that we round off F tilde to l i bits and this rounding off is denoted by lower floor of F tilde l i. So, what it means that we use the first l i bits of F tilde as a code word for i. So, this is will be acting as a code word for i. Now, by definition of rounding off by we have F tilde minus lower floor F tilde l i less than 1 by 2 raise to l i. This is by the definition of rounding off operation. Now, if we use, if we assume that we are going to choose l i given by the relationship as log of 1 by Pi, upper floor of that plus 1 then we can show that 1 by 2 raise to l i is less than and this is equal to… So, if you use l i given by this expression then it is very easy to show that this inequality is valid. And therefore, what it means that this quantity out here lies within the step corresponding to i. Thus, l i bits suffice to describe i. Now, in addition to requiring that the code word identifies the corresponding letter or symbol we also require the set of code words to be prefix free.

(Refer Slide Time: 24:02)



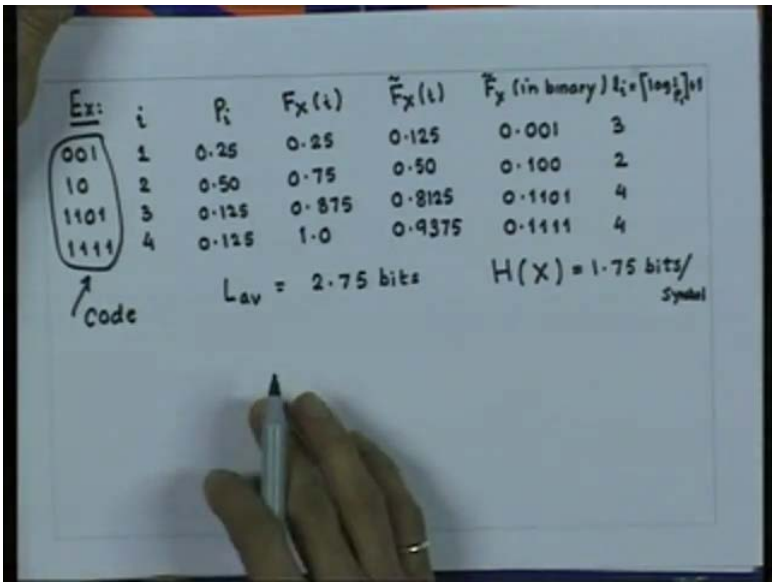
To check whether the code is prefix free we consider each code word which is of the form Z 1 Z 2 up to Z l i to represent not a single point, but the interval given by…So, each code word represents the interval given here. So, the code word is prefix free if and only if you can prove the intervals corresponding to code words are disjoint. Now, let us verify the code is prefix free. Now, the interval corresponding two any code word has a length 2 raised to minus l i which we have shown is less than half the height of the step corresponding to i by equation one. By equation one, we have shown that this interval is less than half the height of the step corresponding to i.

So, what this implies is that the lower end of the interval is in the lower half of the step. Thus, the upper end of the interval lies below the top of the step. And therefore, the interval corresponding to any code words lies entirely within the step corresponding to that symbol in the cumulative distribution function. Therefore, the intervals corresponding to different code words are disjoint and the code is prefix free. Note, that this procedure does not require the symbols to be ordered in terms of probability.

Now, since we use the length which is given by l i is equal to log of 1 by P upper floor of this plus 1 to represent the ith letter. The expected length of the code can be shown to be as equal to Pi l i summation over i is less than or equal to entropy of the source plus 2. So, the average length of the code obtain from Shannon-Fano-Elias coding will be within two bits of the

entropy of the source. Now, let us look at some of the examples to understand this coding scheme.

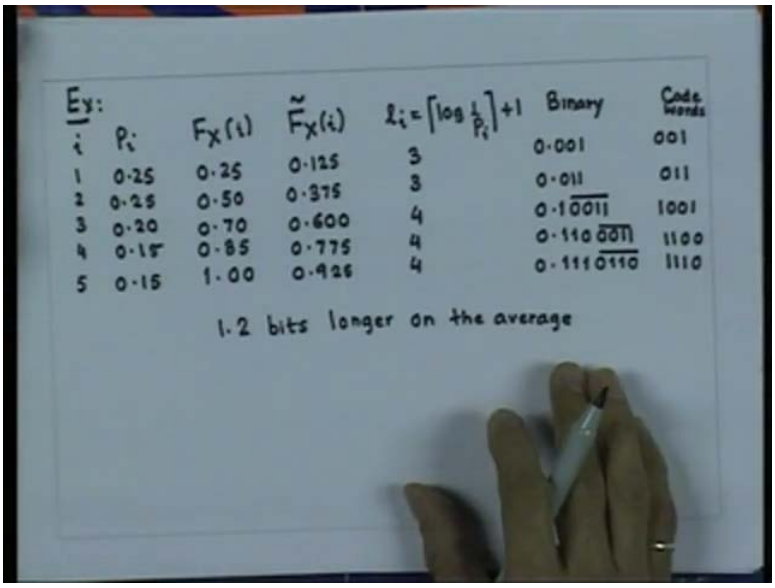
(Refer Slide Time: 28:48)



Let me first consider an example where the probabilities of the letters of the source are dyadic. So, let us assume that we have a source consisting of four source symbols. They are being identified by this mapping as 1 2 3 4 with Pi given as 0.250, 0.500, 0.1250, 0.125. So, for this we can construct c d f as follows and we can construct the modified c d f as… Now, we will represent this in modified c d f in binary. We decided the number of bits for representing this code words by this l i which is given as the upper floor log of 1 by Pi plus1 and which in this case is 3244, and the code words for this can be written as 001,10,1101,1 1 1 1 these are obtained from the binary representation.

So, this forms the code based on Shannon-Fano-Elias coding. And if we calculate average length, average length for this code turns out to be 2.75 bits, while the entropy for this source is equal to 1.75 bits per symbol. The Huffman code for this case achieves the entropy bound. Now, looking at the code word it is obvious that there is some kind of inefficiency. For example, the last bit of the last two code words can be omitted, but if you remove the last bit from all the code words the code is no longer prefix free. Now, this we have seen for the case where the distribution is dyadic.

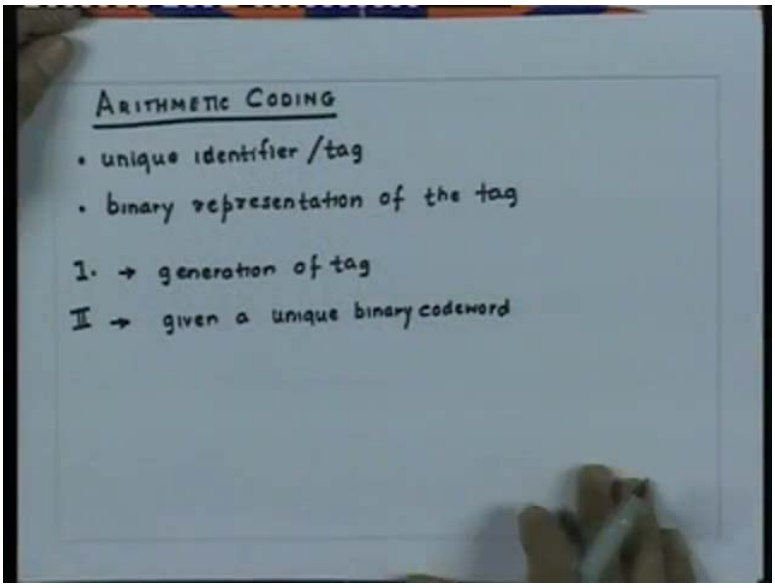
(Refer Slide Time: 32:50)



But if the distribution is not dyadic then let us look at that example. So, the case where the distribution is not dyadic I have a source consisting of five source symbols with the probabilities Pi given as follows. So, for this we can construct my c d f that is 0.250 0.500 0.700 0.85 and 1 and for the same source we can construct the modified c d f as 0.925. Now, if you look at the lengths which is given by log of 1 by Pi upper floor of this plus 1, this comes out to be 3344 4. So, the binary representation of this would be, binary representation will be given 0.001, 0.011, 0.10011, 0011 keeps on recurring. So, we put a bar out here. For this case we find again this is recovering, recurring, so we put a bar and finally for… Now, the code words will be decided by this l i, so this will be 001. So, this will be my code words 001, 011, 1 0 0 1, 1 1 0 0, 11 10.

Now, the above code is 1.2 bits longer on the average than the Huffman code for this source. Now, we will extend the concept of Shannon-Fano-Elias coding and describe a computationally efficient algorithm for encoding and decoding call arithmetic coding. In order to find a Huffman code for a particular sequence of length m, we need code words for all possible sequences of length m. This fact causes exponential growth in the size of the code word. Therefore, we need a way of assigning code words to particular sequences without having to generate code for all sequences of that length. The arithmetic coding technique fulfills this requirement. Let us have a look at the basics of arithmetic coding.

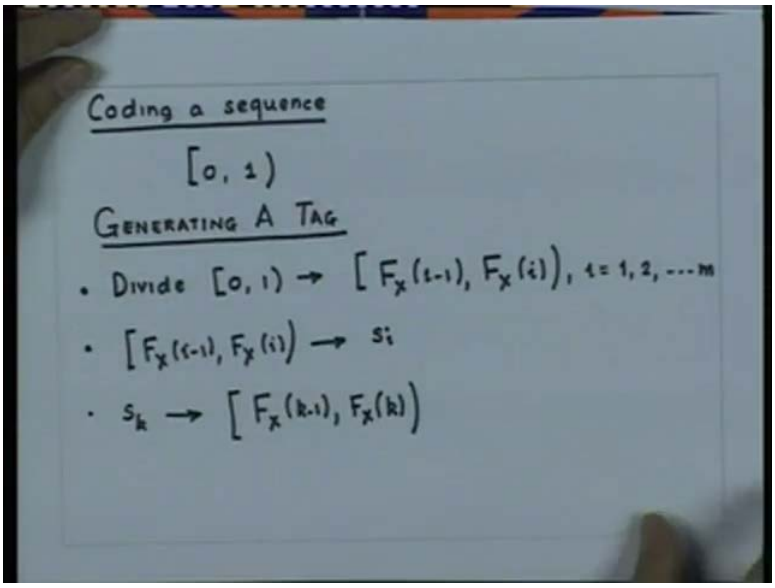
(Refer Slide Time: 37:01)



In arithmetic coding a unique identifier or a tag is generated for the sequence to be encoded. This tag corresponds to a binary fraction which becomes the binary code for the sequence. Impact is the generation of the tag and the binary code are the same process. However, the arithmetic coding approach is easier to understand if we conceptually divide the approach in two phases. So, in the first phase a unique identifier or tag is generated for a given sequence of symbols. And in the second phase this tag is given a unique binary code word.

Now, a unique arithmetic code word can be generated for a sequence of length m. m is the size of the alphabet without the need for generating code words for all sequences of length m. This is unlike the situation for Huffman codes. In order to generate a Huffman code for sequence of length m where the code is not a concatenation of the code words for the individual symbols, we need to obtain the Huffman codes for all sequences of length m. So, let us look at the coding of a sequence in arithmetic coding. So, to code a sequence we follow the following procedure. In order to distinguish a sequence of symbols from another sequence of symbols we need to tag it with a unique identifier.

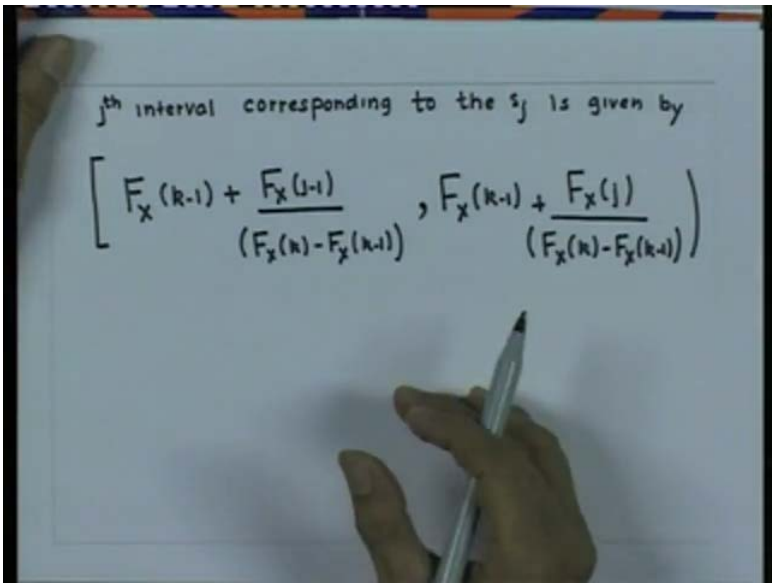
(Refer Slide Time: 40:23)



One possible set of tag for sequences of symbols are the numbers in the unit interval0 to 1.Because the number of numbers in the unit interval is infinite, it should be possible to assign a unique tag to each distinct sequence of symbols. Now, in order to do this we need a function that will map sequences of symbols into the unit interval. A function that maps random variables and sequences of random variables into the unit interval is the cumulative distribution function. So, let us use this function in developing the arithmetic code. So, the next thing is how do we generate a tag?

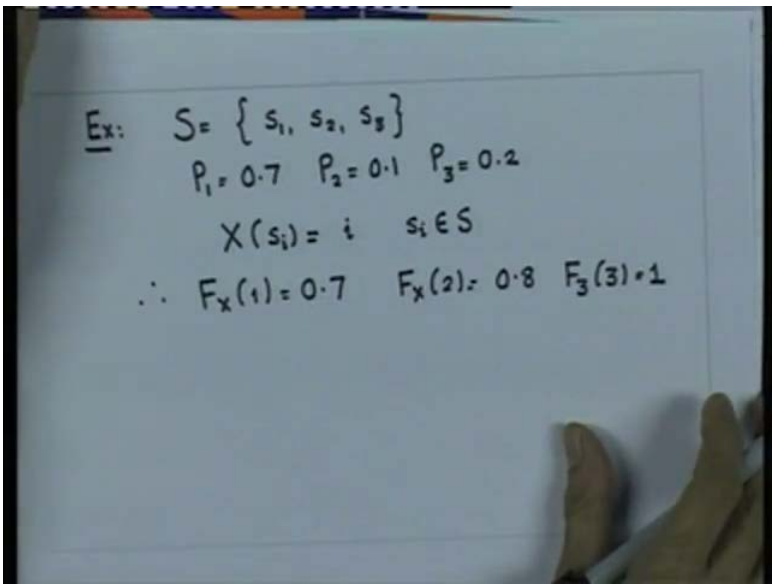
The procedure for generating the tag works by reducing the size of the interval in which the tag resides as more and more elements of the sequence are received. So, the first thing is we start out by dividing the unit interval into subintervals of the forms F Xi minus 1 and F Xi for i equal to 12 m where m is the size of my source alphabet. Now, because the minimum value of the c d f is 0 and the maximum value is 1 this exactly partitions the unit interval. We associate the sub interval F Xi minus 1, F Xi with the symbol Si. The appearance of the first symbol in the sequence restricts the interval containing the tag to one of these sub intervals. Suppose, the first symbol was S k then the interval containing the tag will be the sub interval as given here. Now, this sub interval is now partitioned in exactly the same proportions as the original interval.

(Refer Slide Time: 43:57)



What it means that j th interval corresponding to the symbol S j is given by, on having received S k plus… So, if the second symbol in the sequences S j, then the interval containing the tag value becomes as shown here. Each succeeding symbol causes the tag to be restricted to a sub interval that is further partitioned in the same proportion. This process can be more clearly understood through an example.

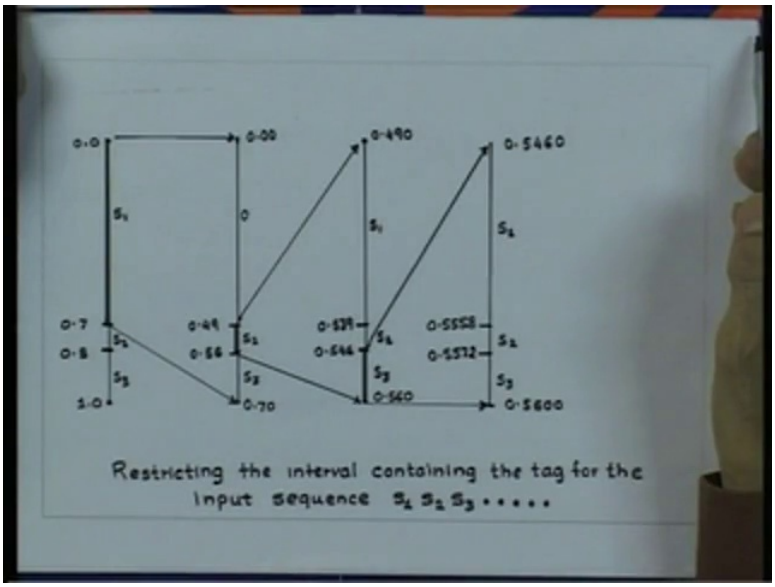
(Refer Slide Time: 45:42)



So, let us take an example of a source consisting of three symbols with the probability model as given here. And we will use the mapping which we discussed for Shannon-Fano-Elias

coding as shown here. Therefore, F X1 is equal to 0.7, F X2 is equal to 0.8 and F X3 is equal to 1. Now, this partitions the unit interval as shown in the figure. So, this gets partition unit interval as shown here. Now, if the, let us assume that the first symbol is S 1 then tag lies in the interval between 0 and 0.7.

(Refer Slide Time: 47:32)



So, the tag will lie in the interval given by 0.0, 0.7. If the first symbol S 2 then the tag will lie in the interval given by 0.7, 0.8 and if the tag lie, if the first symbol is S 3 then the tag will lie in the interval 0.8 to 1.0. Now, once the interval containing the tag has been determined the rest of the unit interval is discarded. And this restricted interval is again divided in the same proportion as the original interval. So, if you assume that the first symbol was S 1 the tag would be containing the interval given by this.

This sub interval is then divided in exactly the same proportion as the original interval yielding, the sub intervals from 0.0 to 0.49, 0.49 to 0.56 and 0.56 to 0.7. So, the first partition as before corresponds to the symbol S 1, the second partition corresponds to symbol S 2 and the third partitions correspond to symbol S 3, suppose the second symbol in the sequence is S 2. The tag value is then restricted to lie in the interval between 0.49 and 0.56. We now partition this interval in the same proportion as the original interval to obtain the sub intervals as shown here.

So, 0.49 to 0.539 will correspond to S 1, 0.539 to 0.546 will correspond S 2 and finally, 0.5546 to 0.560 will correspond to S 3. Now, again if the third symbol is S 3 the tag will be

restricted to this interval which is between 0.546 to 0.560, which can be subdivided further as shown here. Notice, that the appearance of each new symbol restricts the tag to a sub interval that is disjoined from any other sub interval that may have been generated using the process. For the sequence beginning with S 1, S 2 and S 3 by that time the third symbol S 3 is received the tag has been restricted to the interval 0.5460 to 0.5600.

If the third symbol had been S 1 instead of S 3 the tag would have resided in the interval between 0.490 to 0.539, which is disjoint from the sub interval 0.546 to 0.560. Now, even if the two sequences are identical from this point onward one starting with a S 1, S 2 and S 3 and other beginning with S 1, S 2 and S 1 that tag interval for the two sequences will always be disjoint. We will study the tag generation process mathematically starting with the sequences of length 1, and then we will extend this approach to longer sequences by imposing, what is known as lexicographic ordering on the sequences.