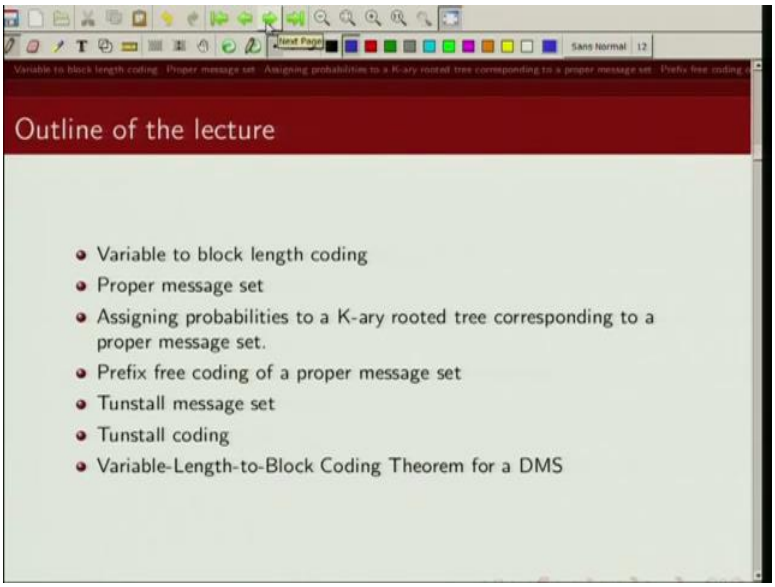
An Introduction to Information Theory
Prof. Adrish Banerjee
Department of Electronics and Communication Engineering
Indian Institute of Technology, Kanpur

Lecture - 05
Variable to Block Length Coding

Welcome to the course on an introduction to information theory. I am Adrish Banarjee, and today we are going to talk about variable to block length coding. So, there are many applications, where the output block sizes fixed, and how do you get compressed store compression in those cases. So, there we like to map large input blocks to fixed length output blocks, and we get compression by varying the input block size depending on the probability of occurrence of those bits. So, in this lecture we are going to talk about how to do variable to block length coding. We will talk about a method to parse input sequence into blocks of variable length, and. In fact, we will talk about one particular case where we know how to do this variable coding optimally.

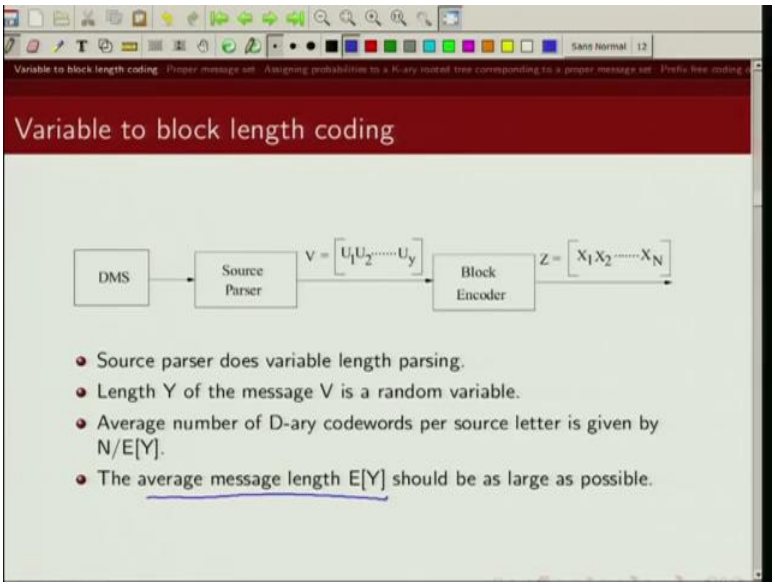
(Refer Slide Time: 01:18)



So, today's topic of discussion is a variable to block length coding. We will describe what we mean by a source parser known as proper message set, and then we will talk about how we assign probabilities to a rooted tree. This we will use to obtain bounds on expected code length. We will also talk about how to do prefix free coding of a proper message set. Subsequently we will define what is an optimal proper message set it is

known as Tunstall message set. And we will talk about optimal variable to block length coding when the source parser is parsed, according to the proper message set, and that is known as Tunstall coding. And finally, we will get basically bounds on expected code word lengths.

(Refer Slide Time: 02:22)

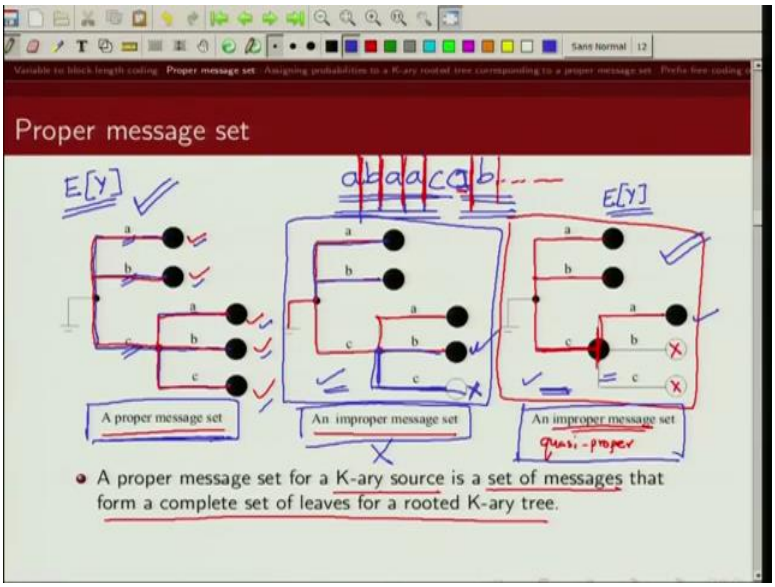


So, let us start this discussion. So, the problem that we are looking at is as follows. So, we have a discrete memory less source, if we want to, which is emitting some bits and we want to derive a source parser of variable length. So, what the source parse does is, it takes these bits and it converts set into blocks of data, where the block length is variable. So, this y is a random variable, and after you parse the source into input into blocks. What this block encoder will do is it will mark these blocks of data to code words of fixed length n. So, n here the output code word length that is fixed input block size y is variable.

So, our objective is to design a source parser, which will do this variable length parsing of the input bits free, and our input block size y is random variable. I really will like y to be large and large, because n is fixed. and average number of bits for the D-ary code word per source letter is given by n by expected value of y, because n is the output code word length and expected value of y is the expected input code word length to this block encoder. So, a measure of goodness of this compression is its ratio n by expected value of y. So, clearly if you want to get good compressor we would like here expected value

of y to be as large as possible for given n. So, our objective here in designing this variable to block length code is, to parse source in such a way, such that expected value of y is maximized. Now how do we do this source parsing?

(Refer Slide Time: 04:49)



So, that brings us to this concept of proper message set of improper message set. So, look at first this example, let say I have a source that emits three types, three bits, it is a ternary source if emits a b and c, and let us look at this particular message set. So, what it does. So, it parses the bit if you get any that is one parsing of the message, if you get b that is the message boundary, if you get a c then we will look at the next bit c a c b or c c. Let us look at some sequence of this thing is a a b a a c c b.

Now note here that if I use this source parser, which I am calling a proper message set. So, how I will parse this input sequence. So, there is a, for a that is one b a a. If I get a c I will look at the next bit c c. So, this is how I am partitioning it into different blocks right. So, if I get an. So, a is one boundary of the parsing b. So, I am parsing this sequence into a b c a c b and c c. So, both here that if you give me any data stream I should able to parse my message stream into this a b c a c b c c. So, what is special here? So, you can see here, if I drawing this message set in form of a tree, you know that all the leaves of this tree have been used for parsing my message sequence.

So, a proper message set of a K-ary source is a set of messages that form complete set of leaves for a rooted K-ary tree. So, if you have a K-ary tree. I am not saying that this tree

as to be full, you need not like extend all the nodes up to the full depth, but you should always have a message corresponds to the root 2 each of the leaves. So, if I get a sequence which has a that is one my that is one my, that is my block b that is my block I get a c I will look at the next bit, and depending on my bit is, that is how I parse my sequence. So, this is my set of sequence that is coming in, and that is how it is getting parsed.

Now, look at this particular example which calling a improper message set. You should look here if I get a and parsing the sequence, if I get b parsing the sequence, if I get c I look at what is my next bit is. So, I parse here it is c a c b, but there is nothing for c c. So, if I get a c c in my input sequence, this particular source parser, which I am calling as improper message set cannot parse my input into blocks of data. Again recall, basically our objective is, to design this. We have a bit of bit stream coming in. We want to divide the bit streams into blocks of data.

Now how do that, how do I parse the source into blocks of data, and of course, remember that my objective is to maximize expected value of y, because my code word length n is fixed. So, I get compression by mapping large blocks of input data to this code word of length n. So, I would like to maximize expected value of y and. So, my source parser I have to design it will parse this input streams into blocks of data, and then I will map these variable length blocks into code word. So, what this proper message set just showed you with an example was, if I choose the proper message set like this, I am able to parse my any primary input set into blocks of data. Here the blocks are a b c a c b and c c.

Now, look at this particular example, the problem in this particular example is, since we are not using this leave. So, whenever had input sequence c c comes, we are not able to parse it in to a block of a data. So, if you look here, if I get my input stream is parsed. So, if I get b my input stream is parsed, if I again get my input stream is parsed a, if I get a c. So, note that if I get a c I need to look at the next bit; the next bit is c c, where c c is accused. So, if I give this particular input stream to my message source parser which is parsing my sequence like this, I cannot parse sequences which have type c c. So, this is an example of improper message set. So, the proper message set we are leaving all the leaves of a K-ary rooted tree; where as in an improper message set we are not using some of the messages. So, clearly we cannot parse a source using improper message set.

Now, I want to look at another type of improper message set, which we can call also as sometime know an quasi proper message set. So, quasi proper message set or improper message set. So, note here also, we are not using these two leaves of the K-ary rooted tree, but this can be used for message parsing, how, just look at it. So, if I get a that is here my one boundary is related to b that is one a message boundary, if I get a c note I will look at what comes after c, if a comes I will use c a that is my message boundary, if b or c comes I will just use c. So, if you look at this particular example, if I get a and I use this message set, this is my message boundary. If I get b this is my message boundary if I get an c this is my message boundary, if I get an that is my message boundary that is how the source getting parse, if I get c I will look at the next bit the next bit is c.

So, in that case then I will parse it at c and then c and then b. like this would have been c a then what I would have been done was, I would have a parsed it, if this would have been c a, then my message boundary would have been c a. So, what is the difference between this and this? What is the difference between this and this? In this particular example note, I am able to parse all possible input sequence. So, if I get c what I am doing is, I am waiting and seeing what is the next bit. If the next bit is a, I marked my message boundary of at c a, but if the next bit is of sequence, which following c is either b or c then I will parse my message I will, and my message boundary is just c.

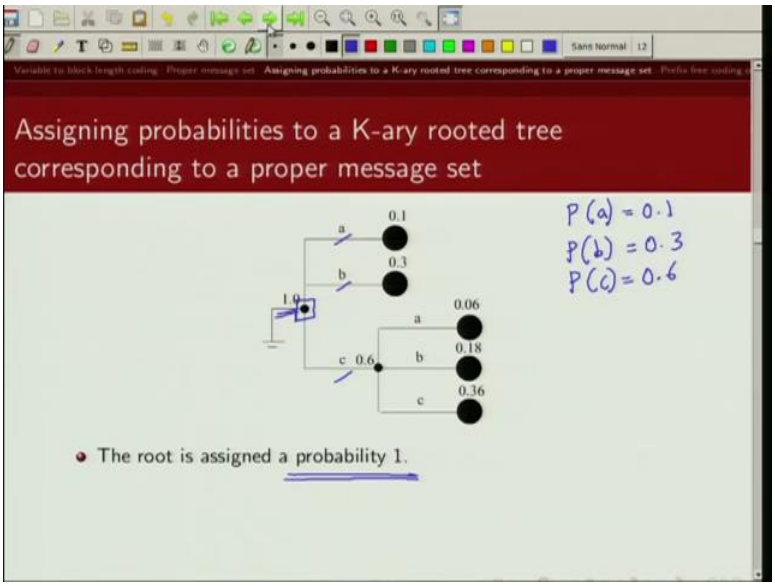
Now, the problem here is, in this particular cases, if I get a c then I of course, if I get a c and a it is similar to I am doing here, but if I get a c and b, then the message boundary it is c b, but if I get a c and c since this leaf is not been used, it cannot parse any sequence which is of the form c c. Now, let us compare this with this. What is the difference between this proper message set and this quasi proper message set? Now both proper message set and quasi proper message set, both can parse all possible input sequence. We have seen that you give a primary input sequence; it can parse primary input sequence. What is the difference? The difference is in case of a proper message set, when bits are coming I know exactly where the boundaries are .For example, then a is coming I know that is my input block length boundary. If b comes I know that is the input block length boundary. If c comes I will look at the next bit, and I know that is my message input message boundary c a c b c c.

Now, happening there if a and b are coming I know that is my message boundary, but if a c comes I do not know instantaneously, whether that is my message boundary or I should look at next bit that depends on. So, once I get a c I need to look at the next bit to decide whether my message boundary is at c a or I should guess my message boundary at c. So, I cannot instantaneously tell how to parse the source message. I need to look at the next bit and then decide where my inputs bit stream should be parsed. So, that is the difference between a proper message set, and a quasi proper set. Both can parse input sequence, in one case I instantaneously when the bits come; I know where the message boundary is. In the second case I may have to look, I will have to look into further bits in the future before I decide where the message boundaries are OK.

So, I have given example of various ways in which you can parse the message, input message sequence. Clearly if I use a proper message set or I can parse all input sequence, and I can instantaneously parse the bits. If I use a quasi proper message set I can parse all input sequence, but I may not be able to do, instantaneously I may have to look at some bits in future and then decide my message boundary whereas if you are using an improper message set, then you cannot parse all possible input sequence. So, we cannot use this improper message set for source parsing. Now in case of a proper message set we know. Remember our objective is to maximize expected value of input code word length. You want to maximize expected value of y. in case of a proper message set; we know a method to do that. We know a method to optimal to find the optimal code, which will maximize expected value of y. Whereas the quasi proper message set, we do not know an algorithm to find a way, or find the code which will maximize expected value of y.

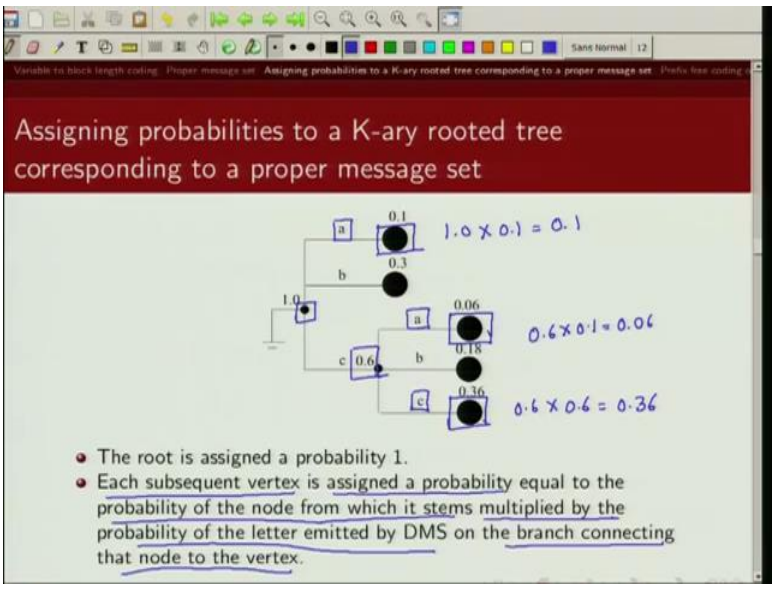
So, in this lecture we are going to concentrate only on proper message set. How do we design a source parser, using proper message set such that our expected value of y is maximized. So, now, first I am going to show you how to assigning probabilities to a K-ary rooted tree, corresponding to a proper message set, it is got clicked message set. So, how do I find probabilities K-ary rooted tree. You have seen in the previous slide, we have represented this proper message set giving a K-ary rooted tree. Now how do we find probabilities to these leaves; that is what we are going to talk about next.

(Refer Slide Time: 18:32)



So, let us take an example again for source I mixed ternary sequences a b and c, and let us say probability of occurrence of a is, 0.1 probability occurrence of b is 0.3, probability of occurrence of c is 0.6. So, first thing the root is assigned a probability of one. So, in this case this is my root. So, this has a probability one.

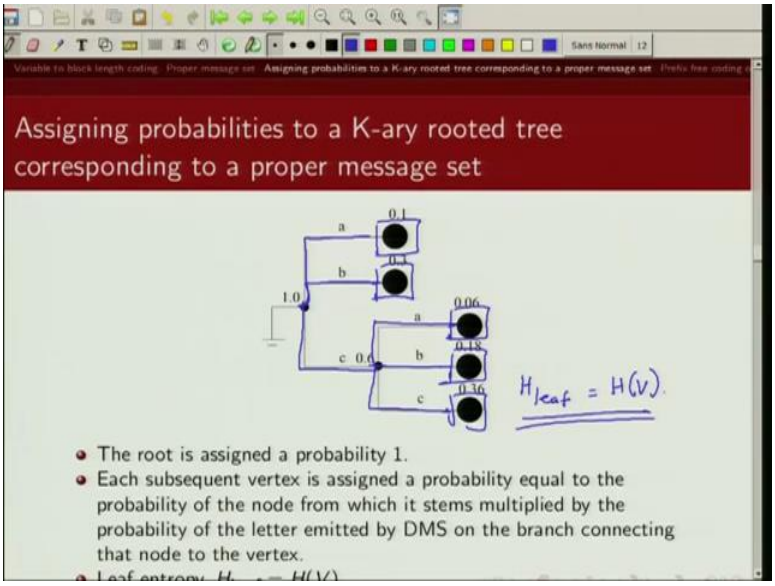
(Refer Slide Time: 19:06)



Each subsequent vertex is assigned a probability, which is equal to probability of the node from which it is tends, multiplied by the probability of the letter, emitted by the discrete memory less source on the branch connecting that node to the vertex. So, then

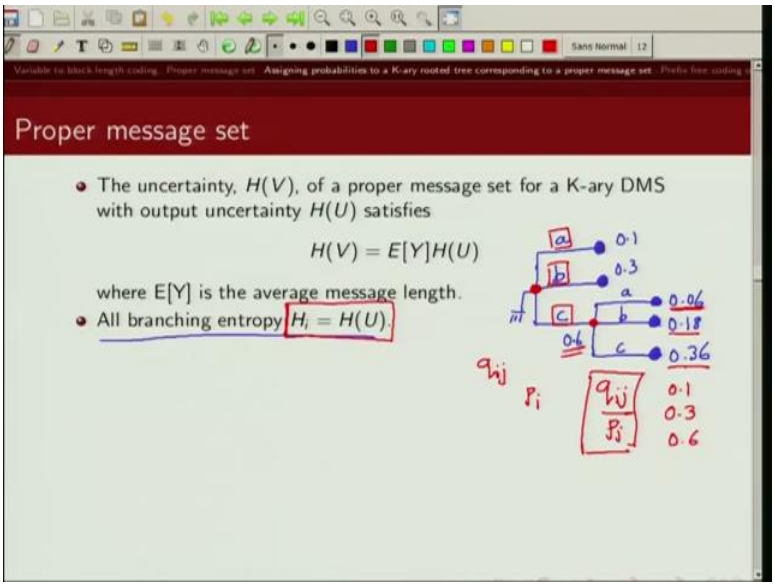
according to this what should be the probability of this, what should be the probability of this vertex? This probability would be probability of this node multiplied by probability of occurrence of this letter a. So, this is one into probability of a was 0.1, so this is 0.1. Similarly what is this probability of this vertex? probability of this node is 0.6, and probability of occurrence of a is 0.1. So, this is 0.6 into 0.1; that is 0.06. So, we can find probability of any vertex, you see what is the probability of this vertex. This is probability of this node which is 0.6 into probability of occurrence of c; that is again 0.6 that is. So, this is how we assign probability to K-ary rooted tree corresponding to a proper message set, and we can write down that leaf entropy is nothing, but h of v. note what is your v. So, v is our output of the source parser; that is why and what is this denoting.

(Refer Slide Time: 20:51)



What is this leaf denoting, what are these leaves denoting. This is my parse sequence a b c a c b c c that is how I parsing my message sequence. So, these are my v i's right and. So, the leaf entropy will give me what is the entropy of v. So, leaf entropy which is the entropy of these leaves, is nothing, but entropy of v, because this is my, these are all parsing my source, this is my v i's. So, just leaf entropy is given by h of v.

(Refer Slide Time: 22:03)

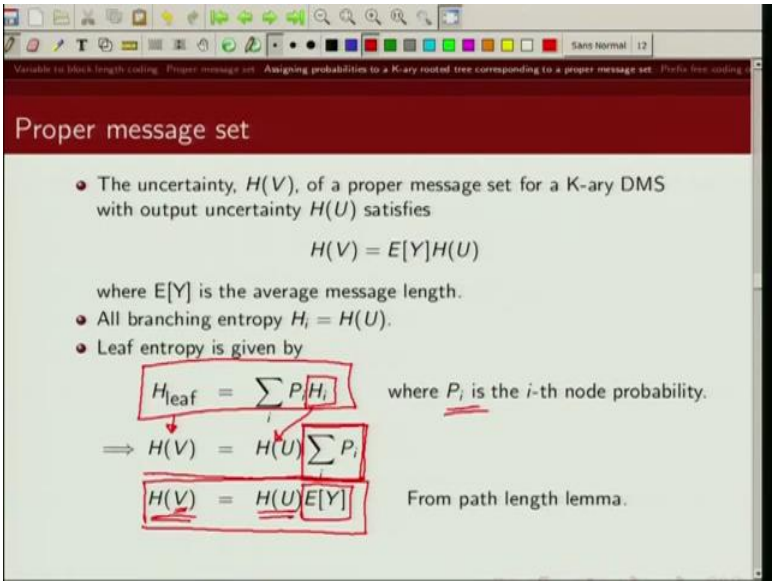


Now we are going to show that the uncertainty of a proper message set, which we are denoting by h y of a K-ary discrete memory less, source with output uncertainty u satisfies this relationship. So, uncertainty in the output of a message parser which is v, is given by expected value of y into uncertainty in the K-ary discrete memory less source output, which is denoting by u. So, again just go back, and just recall what our u's and v's are. So, u is basically my input to the source parser, and v is my output of the source parser. So, if you have a K-ary discrete memory less source, I have u one u 2 u three u k. these are my bits emitted by the discrete memory less source, and v is the output of my source parser. And I am using a proper message set to design my source parser. Now, please note that the branching entropy for this proper message set is given by h of u. now why this is true. So, again please go back and draw a proper message set, just draw the same proper message set that we have been working with.

So, this is my root, this is corresponding to letter a, this corresponding to the letter b, this corresponding to letters. So, this is c a, this is c b, this is c c, and we consider probability of this was 0.1 0.3 0.6, this was 0.06, this is 0.18, this is 0.36. Now, look at Gaussian probability, let us look at this particular node, let say let us, let us look at this particular node right, let us look at this particular node at c before I have to compute branching property. What is the branching probability? That is this probability divided by the probability of the node, uncertainty associated with that.

Now note here what is this and this probability divided by this, this probability divided by this. So, this if we use the notation which we have used earlier, we have used this q I j to denote these probabilities and we have used notation P i to denote the probability of the node, and this ratio q I j by P i for this particular node, this ratio is nothing, but 0.06 by 0.6 which is 0.1, and then 0.18 divide by 0.6 is 0.3, and this is 0.3 6 by 0.6 it is 0.6. So, no matter what node I choose let us choose this node. Again these branch this ratio of q I j by P i j, these are nothing, but probability of these letters a b and c. So, we can see clearly that branching entropy of a proper message set, basically this branching entropy is given by h of u.

(Refer Slide Time: 26:15)



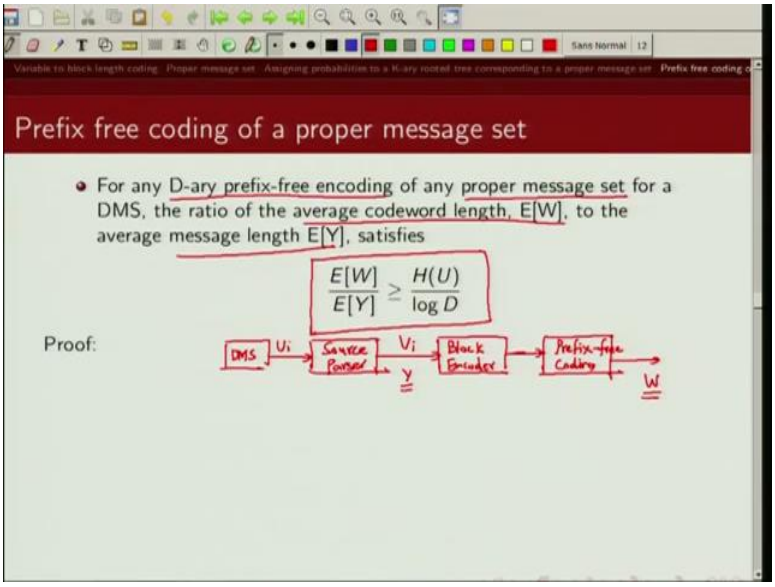
Now, we know how the branching entropy is related to the leaf entropy. Just now we have proved earlier the leaf entropy is given by summation of P i H i where P i is the probability of ith node. Now we know that leaf entropy leaves corresponds to v. So, leaf entropy is h y. we know the branching entropy H i is independent of I it is always equal to h of u. So, I can take that out. So, what I get is then h, of v is equal to h of u summation of these probabilities of the nodes, including the root node. And what is this.

According to the path length lemma, the summation of P i is nothing, but expected value of y right. I have some root to each of these leaf I have my v i's which is the parse message, and the length of that is y one y 2 y three, the expected value of that, from path length lemma is given by this quantity. So, what I have shown you is, the uncertainty

associated with the parse message source for a proper message set, is given by this relation, is a uncertainty in the soft method by divide by expected value of y, which is the length of the parsed message.

Now, let us try to. Now we are talked about the relation between the entropy of the parse message with the uncertainty associated with the source method. Let us just take a small diversion, and let us try to compute and find out how we can do a prefix free coding of a proper message set. Now if we do a prefix free coding of the proper message set, that would done make it a variable to variable encoding; why, because proper message set it takes variable length input and gives a fixed length output, and a prefix length prefix free coding takes a fifth line code and gives the variable length output. So, if you do a prefix free coding of a proper message set. What we will get is the variable to variable length coding. Since we have studied prefix free coding, I am just taking the liberty to deter.

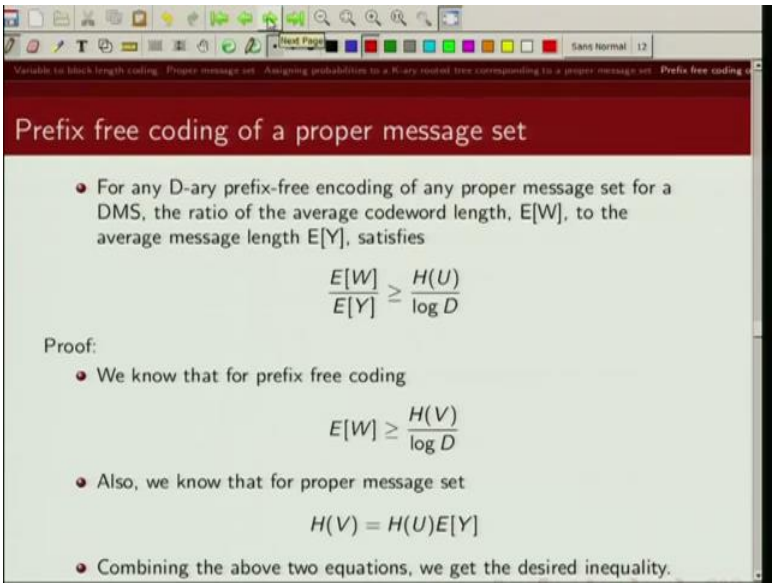
(Refer Slide Time: 29:04)



So, we can show that if you do prefix free coding of a proper message set, then the ratio of average code word length to the average message length satisfies this relationship. So, again let me clarify the scheme that we are looking at. We have a discrete memory less source which was putting up u i's I am source parser as, I am giving a proper message set let see to parse my sequence I get my v i's here. Now, this is parsed if you block encoder. block encoder what it does is it just do one to one mapping of this vi's to a fixed block length code. So, it maps for fixed block length code.

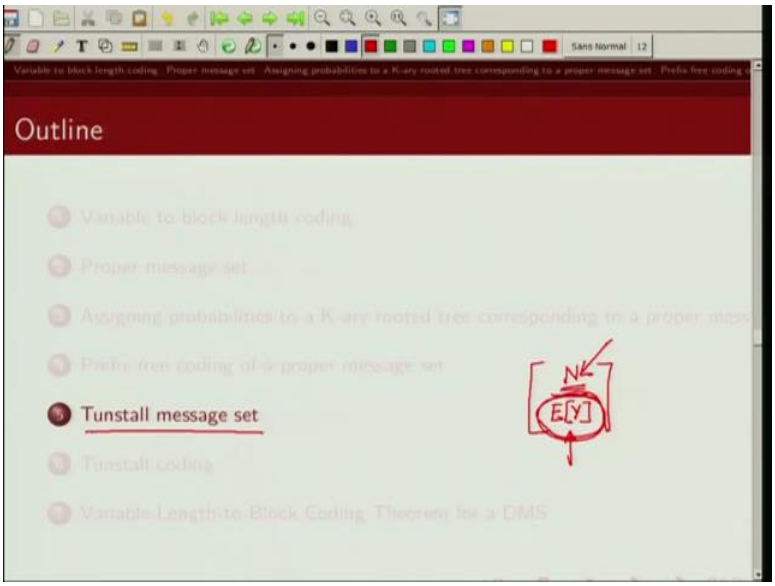
Now that is input to a prefix free coding, prefix free coding so this is these takes as a input we have fixed length, and what it gives out is code words of variable length, and w is the length of this output code word, which is a random variable, and y is the source length of this parsed input sequence and y is also a random variable. So, what I am seeing is if we do a prefix free coding of a proper message set, and then expected value of the output code word length divided by expected value the input code word length which is basically by proper message set, which is input to this encoder that is lower bounded by entropy.

(Refer Slide Time: 31:08)



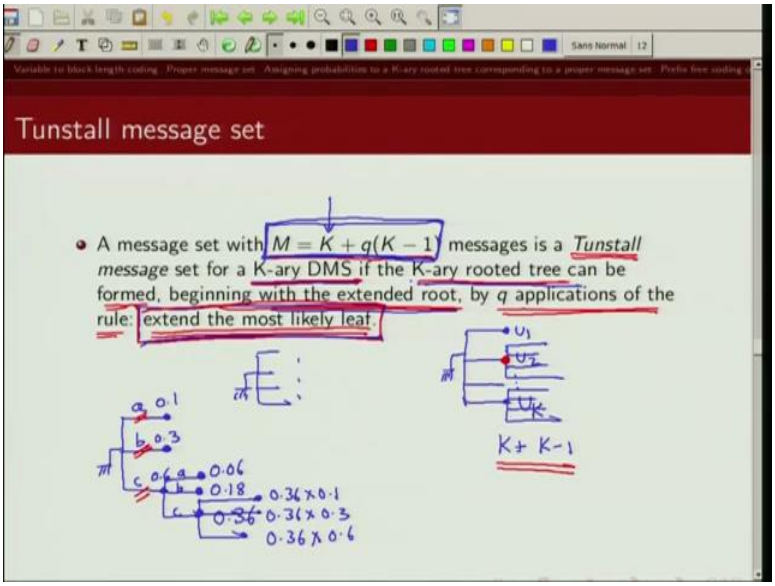
So, we know for a prefix free coding expected value of w which is the output code word length is, lower bounded by entropy of the source. The source in this case is v source h v by log d, and I just now I have proved that h of v is nothing, but h of u expected value of y. If you have plugged this in here and divide both sides by expected value of y, what I get is this relation. So, if I do a prefix free coding of a proper message set, then expected output code word length to expect input code word length is basically lower bounded by entropy.

(Refer Slide Time: 32:06)



Now, let us go back to proper message set, and let us talk about how can we design a proper message set that will maximize my expected value of y. Remember because in case of variable to block length coding, my goodness of the code is determined by how small this quantities. So, this is output code word length, this is the input to the block encoder, because n is fixed I want to maximize my expected value of y. So, I would like to parse my source into sequences, such that my expected value of y is maximize. So, Tunstall message set is a class of proper message set, which will always maximize expected value of y.

(Refer Slide Time: 33:03)



So, a message set which was shown in messages is the tunstall message set for K-ary discrete memory less source, if for the K-ary rooted tree, can be formed beginning from the extended root by q application of the following rule, extend the most likely leaf. Now, let us look at each of the point that we have mentioned here; a message set with so many messages. Now why are message set will have so many messages. If you recall if the if it is a K-ary if we use the K-ary random variable, then starting with the root you will have k leaves, let us call it u one u 2 (Refer Time :34:02) this is u k. So, when you start from the root you have k leaves. Now when I extend any of these leaves let say I am extending this leaves corresponding to u 2, then what I am doing is, I am creating k new leaves, how this would was earlier a leaf it has not become a node. So, the number of leafs which is at this point is k plus k minus one, because I am creating new k leaves, but this particular leaf, this was earlier a leaf it has now become a node. So, now, my total number of leaves is k plus k minus 1.

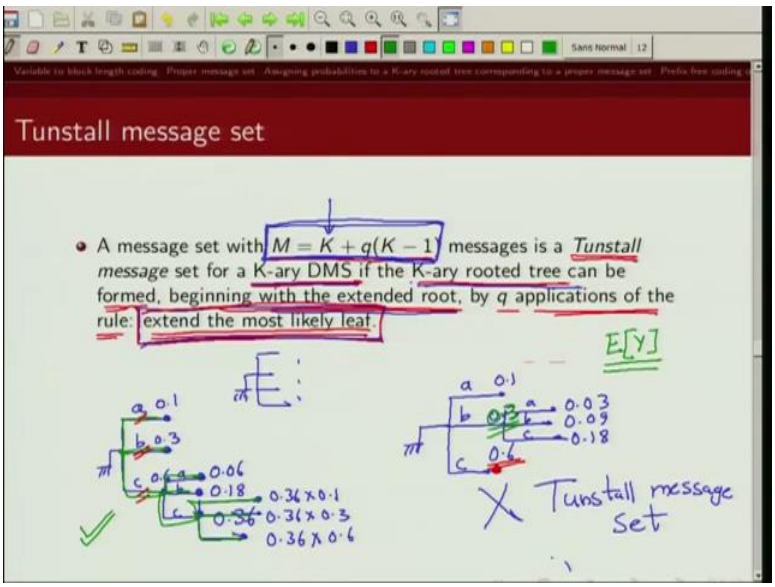
Now again if I extend any node I will be again creating k new leaves, but one of the leaf which was earlier a leaf, is now a node. So, each time I am adding, except the first time when I extend from the root in each time I am adding k minus one leaves. So, if I try to design a proper message set, remember a proper message set will use all it is leaves. So, total number of leaves I can get, if I extend a node q times. So, if I have a, if you are starting with an extending root. So, if I have root which has been extended, and if I now try to extend the leaves of this rooted tree, the total number of messages that I can get is given by this, it is k plus q into k minus one, where q is the number of extensions that I have done from this extended root. I am calling this as extended root. So, I have a root which has been extended leaf that is my, that is what I am calling extended root. So, a extended root I get k leaves, but subsequently when I try to extent any of the leaves of this extended root I am adding only k minus one new leaves.

So, total messages which I can have can have in a proper message set is given by k plus q times k minus 1 now what I am saying is this proper message set which as messages given by k plus q into p minus one messages is going to be a Tunstall message set, only if you follow this rule at each time when you try to extend a leaf, you extend the most likely leaf. So, when you try to extend the leave, you are only going to extend the most likely leave. If you do that then that is the proper message set. So, let us take an example. So, of the primary message set that we had.

So, let say I had here, this was a, this was b this was c, and probability of a it took as 0.1, probability of b, we took at 0.3, probability of c we took as 0.6. These are proper message set, let I want to extent a leaf. Now which one should I extend - should I extend a, should I extend b or should I extend c according to this I should extend the most likely leaf. Now which is the most likely leaf? That is c, because the probability of this vertex is 0.6. So, I am going to extend this particular leaf. Now if I extend this particular leaf what is this probability a b c. this probability is probability of this node, which is 0.6 multiplied by probability of this letter a, which is 0.1. So, this is 0.06, this is 0.6 into point 0.18 and this is 0.36.

So, now I have nodes leaves, this is probability 0.1 this is probability 0.3 this is probability 0.06 this is probability 0.18 and this has probability of 0.36 if. I extend the next leaf which one should be; I should extend this leaf, so next I should extend this leaf. And again similarly probability of this node is 0.36 into 0.1. So, it is 0.36 into 0.1, this is 0.36 into 0.3, and this is 0.36 into 0.6, and this process continues. So, if I do extension of leaves in this particular fashion. If I construct my proper message set in this fashion, then it is known as a tunstall message set. So, my Tunstall message set, corresponding to K-ary discrete memory less source. If the K-ary rooted trees from beginning from the extended root, such that you extend the most likely leaf then that is the tunstall message. Now, let us taken an example of a non tunstall message set, let us just take the same example.

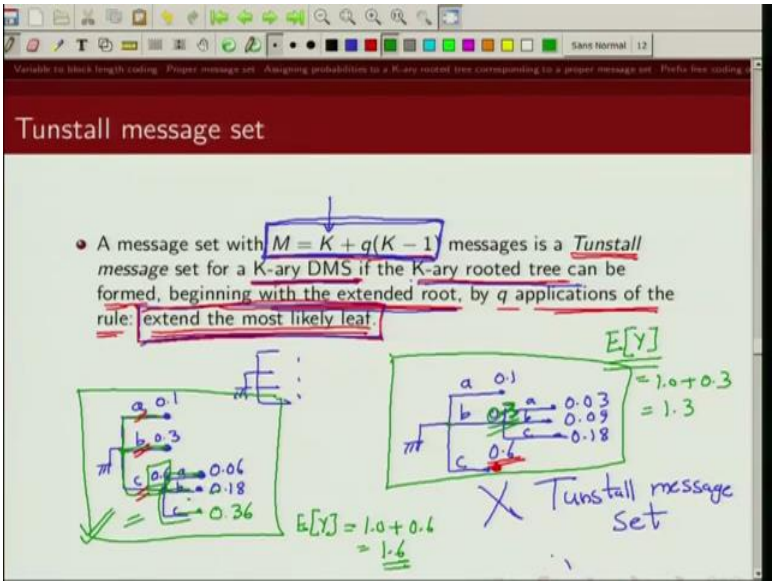
(Refer Slide Time: 40:03)



So, I have a message u which emits primary alphabets a b and c with probability 0.1 0.3 and 0.6. Now according to the tunstall message set with, I should be extended extending this particular node. Now if I do not extend this node, let say I extended this particular node. So, then this node this is a b and c, it is probability is 0.3 into 0.1; that is 0.03 0.3 into 0.3 0.09 and 0.3 into 0.6; that is basically this. Now this is not a Tunstall message set. Why this is not Tunstall message set, because here I should have extended this particular node, because this as probability 0.6. Instead I extended this node which as probability of 0.3. So, this is not a Tunstall message set. Whereas this is a tunstall message set. Now you can see easily why Tunstall message set maximizes expected value of y, because that is our objective. We want to parse a message in a fashion which would maximize expected value of y.

Now, what is expected value of y, what is y. y is essentially the length of this parse sequence average length of this parse sequence. So, in this case we are parsing the sequences like this a b ca c b c c a c c b c c c like that you are parsing right. So, what is expected value of y. Expected value of y is nothing, but from the path length lemma it is the sum of probabilities of all nodes, including the root node right. This is expected value of y according the path length lemma is nothing, but some of probabilities of all the nodes including the root node.

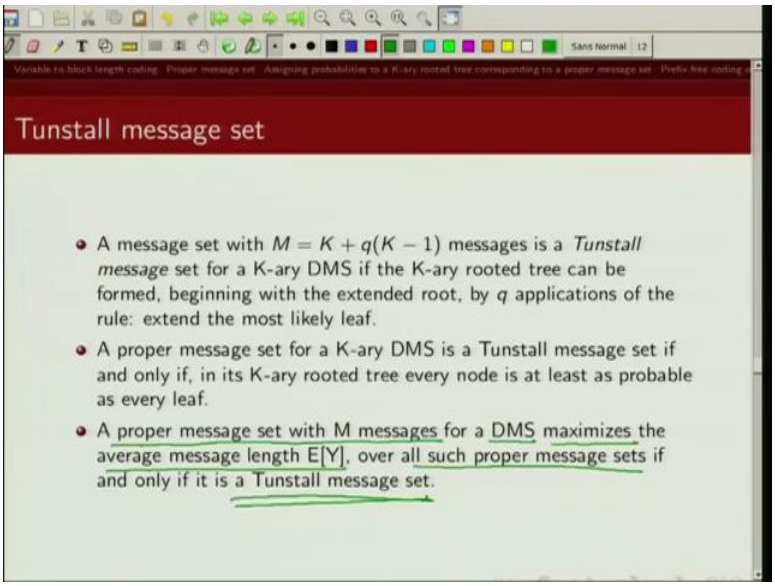
(Refer Slide Time: 42:38)



Now, let us take an example the same example, let us just take this example which was Tunstall message set. Take this example which is a non Tunstall message set. Now what I did was in both the cases I did a single extension after the extended root. In the tunstall message set I extended the node which as leaf which as 0.6 probability. In this case I extended a leaf which as 0.3 probability. Now compute expected value of y here. In this case it will be 1 plus 0.6, so it is 1 0.6. What is the expected value of y here? This is root of probability one, and this node which has been just extended as probability 0.3.

So, this expected value of y is 0.3; whereas here it is 0.1 0.6. So, clearly you can see, if I extend the most likely leaf then that leaf becomes node right, and the probability of node gets added to the expected value of y. If every time I am extending the most likely node, most likely leaf, then that leaf is becoming a node, and that is adding, it is probability getting added to the expected value of y. So, the expected value of y will be maximum when I extend the most likely leaf, and that is why Tunstall message set will maximize my expected value of y.

(Refer Slide Time: 44:38)



So, a proper message set for a K-ary discrete memory less source is a Tunstall message set. If and only if, in it is K-ary rooted tree every node is, at least as probable as it is leaf. Why would every node be at least as probable as the leaf node? Because we are extending the most likely node, if you extend most likely node, the probability of the leaf would be certainly less than probability of the node. So, every node will be at least as

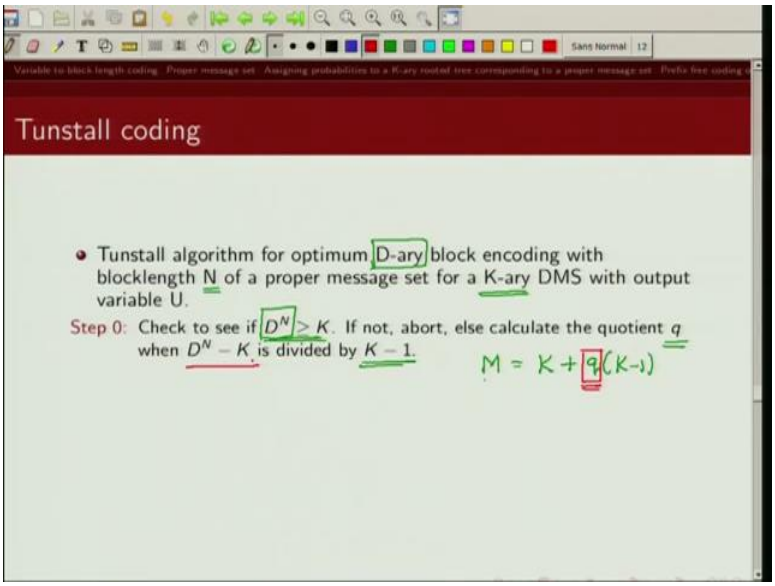
probable as every leaf. You can just take an example here. What is the probability of the leaf here? It is 0.1 0.3 0.06 0.18 0.36, and what is the probability of the nodes.

Probability of nodes is 1 and 0.6 right. You can see the leaf probabilities are less than. So, the node probabilities at least equal to the leaf probabilities at more, so you can see that where as in this particular example, where we have considered a non Tunstall message set. What are the leaf probability is, it is 0.1 0.03 0.09 0.18 and 0.6 where as the nodes of probability 1 and 0.3. So, you can see here for a non Tunstall message set we have a leaf, which has probability more than a node, but this is not going to happen if your message set is a Tunstalll message set, because in a Tunstall message set we are extending the most likely leaf.

So, in case of a Tunstall message set, your node is at least as probable as the leaf, every leaf. This follows from the fact of in Tunstall message set we are extending the most likely, most probable leaf. It is not difficult to see them a proper message set with m messages for a discrete memory less source; we will maximize the average message length expected value of y, over all possible proper message set, if and only if it is a Tunstall message set.

And I have illustrated this with an help of the example, because we are extending the most likely leaf, that leaf now becomes a node, and the computation of expected value of y from path length lemma this is a summation of probabilities of the all the nodes including the root node. So, this particular leaf which was extended now, it is probability gets added to that sum. So, if we extend the most likely leaf, we are ensuring that our expected value of y is maximized. So, over all possible proper message set, Tunstall message set will maximize expected value of y. So, if you want to do optimum variable length coding, variable to block length coding, using proper message set then you should be using Tunstall message set. So, now that we have described how to parse our source into proper message set. Let us explain how we can use this Tunstall message set to do our variable to block length coding.

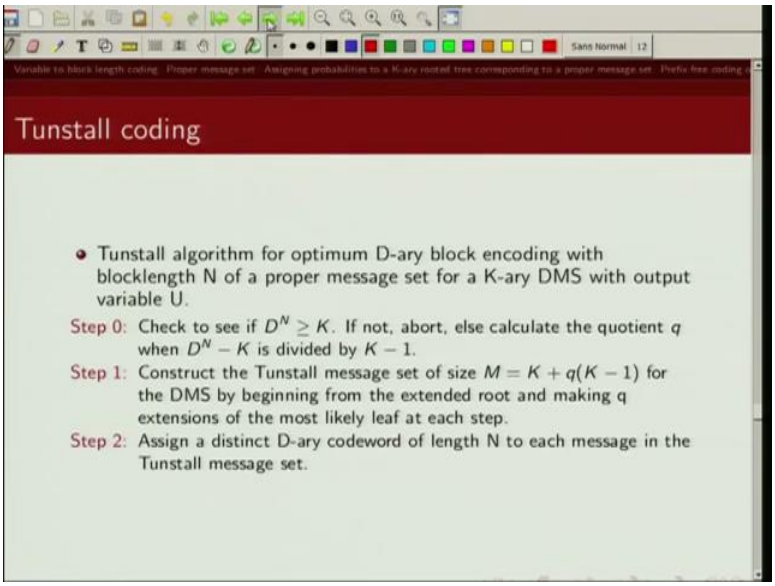
(Refer Slide Time: 48:43)



So, we are interested in optimal block variable to block length coding. Our output code words have block length n. we have a K-ary discrete memory less source, which inputs these random value u. So, u 1 u 2 u k; that is what it is emitting. first thing that we need to do is we need to check if d raise to the power n, is greater than equal to k. If it is not we cannot find a code for that and. So, remember what is n. n is the output code word length, and remember the output code word length is D-ary. So, how many such code words we can have. We can have d raise power n possible code words.

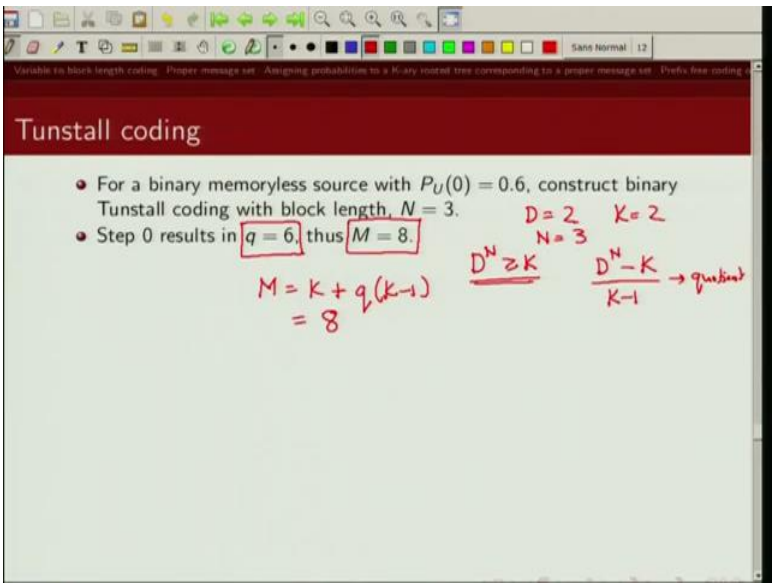
Now, clearly this number as to be more than inputs k; otherwise we cannot map each of these inputs to a code word. So, d raise power n has to be greater than or equal to k. Now if this condition is satisfied, then first thing that we are going to do is, we need to find out how many times we are going to extend the most likely node. Recall that we have set number of messages is given by k plus q times k minus 1 right. And what is the number of messages that we can encode; that is d raise to the power n. So, d raise power n minus k if we divide by k minus 1, that would give us this q. why do we need this q, because we need to know how many message sets to create. Like (Refer Time: 50:43) of messages that we need to create. So, how many extensions of this a message set, proper message set we have to do, we have to first find out right. And how do we find out. we find this out by dividing d just power n minus k by k minus 1 and finding the quotient that would give us this q.

(Refer Slide Time: 51:07)



Next we construct a Tunstall message set of size m, where m is given by k plus q into k minus 1, and we start with the extended root and we will make k extensions of the most likely leaf at each step. Please recall in case of a Tunstall message set, we extend the most likely leaf, and to get a Tunstall message set of size m which is given by this. We need to do k extensions of the extended root. So, at each time we are going to extend the most likely leaf. And once we have formed this messages, then we are going to assign a distinct code word to each of these messages.

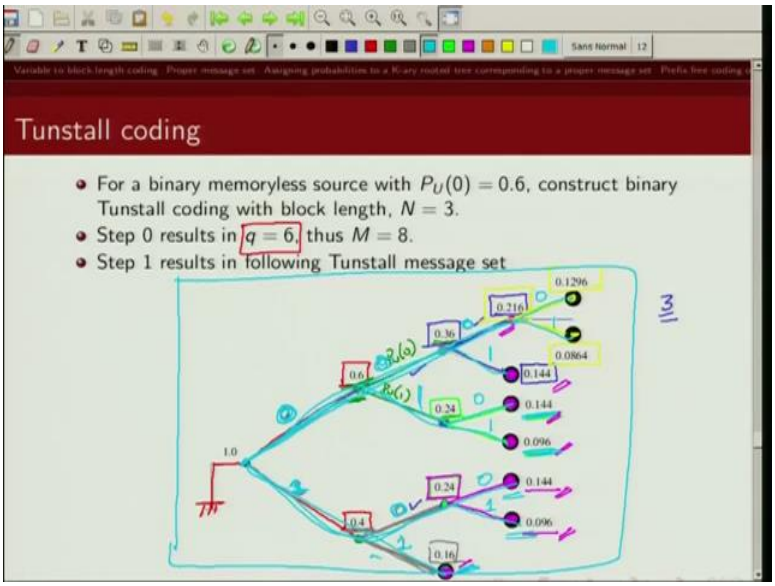
(Refer Slide Time: 52:07)



So, let us take an example to illustrate this. So, consider a binary message. So, k is 2 probability of u being 0 is 0.6, so probability u being 1 is 0.4. Now you have been ask to construct Tunstall coding, and the output block size is given as three, and output also we are considering as binary. So, d is also 2 d is 2 k is 2 n given as three. So, what is the first step you have to first check whether d raise to the power n is less than equal to k, d is 2 n is three. So, 8 is greater than 2.

So, this condition holds. next what we need to do is, we need to find the quotient when d raise to the power n minus k is divided, when it is divided by k minus 1 we need to find the quotient. So, k minus 1 is 1 in this case and d raise to power n is 8 minus 2; that is 6 6 divided by 1 the quotient is 6. So, in this case, in this example q comes out to be 6, and of course, number of the messages is (Refer Time: 53:4) n which is number of messages that we have, is given by k plus q times k minus 1, and this is 2 plus 6 that is 8. So, in this case we can find a Tunstall message set. What we need to do is? We need to do six extensions of the extended root. So, let us do that.

(Refer Slide Time: 54:18)



So, this is a rooted tree and that is the root. So, what I have here at this point is an extended root. The probability of 0 is 0.6 that is given by this, and probability of one is given by this. Now remember I need to do 6 extensions of the most likely leaf correct. So, tell me now. So, this is my extended root, and I need to do 6 extensions of that. So, first what should I do? So, note here 0.6 is more than 0.4. So, first I should extend this

node, because probability 0.6. So, the first extension is something like this. This is my first extension of the node extended root. So, this probability is nothing, but probability of this node is 0.6 into probability of 0 probability of 0 is 0.6, so this is 0.36.

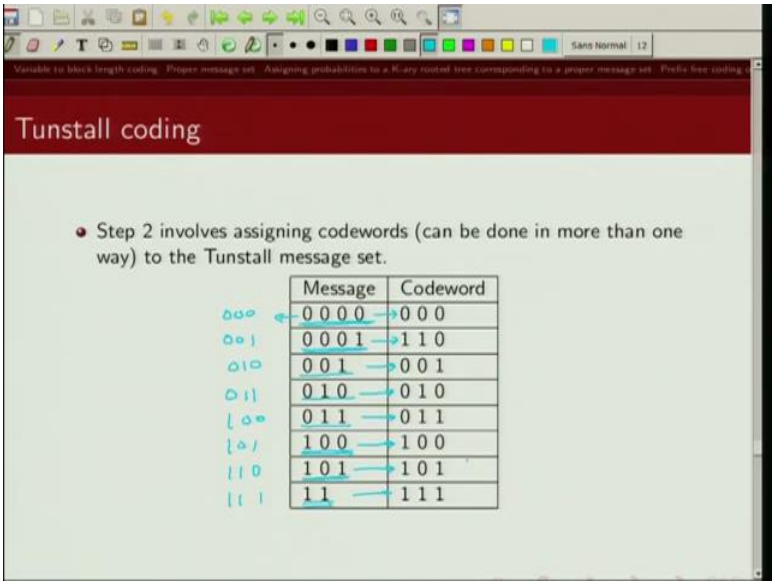
Similarly the probability at this particular leaf is probability of this node it is 0.6 into probability of this letter one, which is 0.4, so 0.6 into 0.4 that is 0.24. So, after first extension this is how my message set looks like. Now let us do the second extension. So, if I do second extension, so what do I have? I have 0.36 0.24 and 0.4. So, I am going to now extend this node, which is as probability 0.4. So, I am not extending this leaf, which has probability 0.4. So, if I extend this leaf now this leaf becomes a node, and the extension is like this. this corresponds to letter 0 this corresponds to letter one, and these probability similarly we can compute, it is 0.4 into 0.6 is 0.24 and 0.4 into 0.4 this 0.16.

So, this is now the second extension of the extended root. Third, now, at this point what are the probabilities of a leaves, this is probability of 0.36 this is probability 0.24, this is probability 0.24, and this is probability 0.16. So, at this point we can see that this particular node as the highest probability. So, our third extension will happen from this particular leaf. So, if I extend this leaf, again this probability is 0.36 into 0.6, and this probability is 0.36 into 0.4. So, at the end of third extension I have these following nodes, this following leaves, with this leaf as probability 0.216, this leaf as probability 0.144, this leaf as probability 0.24, this leaf as probability 0.24, and this leaf as probability 0.16. And I have already done three extensions three extensions have done.

First one was this, which I showed by green, second one was this which was showed by grey color and the third extension was the one which I showed with blue color. Let us use the different color, let us use the light green. Now among the leaves that are left we can see, this has probability 0.24, and this has probability 0.24 and these are two most likely leaves. So, we should be, for the fourth extension we should be extended extending these leaves, any one of them. So, let us extend let say this one, let us extend this, so do that and get this and get this. And similarly I can calculate these probabilities whose 0.24 into 0.6 and this one is 0.24 into 0.4. So, at this point then after four extensions I have this node, this node, this node, this node, this node, and this node. So, at this point the most likely leaf is this, while extend this leaf. So, if I extend this leaf what I get here is, a new leaf with probability 0.144, and a new leaf with probability 0.096. So, I have done so far five extensions of the extended root.

So, at this point the leaf that I have is this, this, this, this, this, this, and this. And look at the probability this is 0.16 0.096 0.144 0.096 0.144 0.144 and this is 0.216. So, at this point the most likely leaf is this one. So, I will extend this, and this has probability 0.216 into 0.6 that is given by 0.1296 and this is 0.216 multiplied by 0.4 which is 0.0864. So, I have done 6 extension of this extended root. So, this is my proper message set. So, what I have constructed is a proper message set that has m equals to 8 messages, and you can see what are those 8 messages. So, one is this one, second one is this one, third one is this one, fourth one is this one, fifth one is this one, sixth, then you have seven, and these are the eighth.

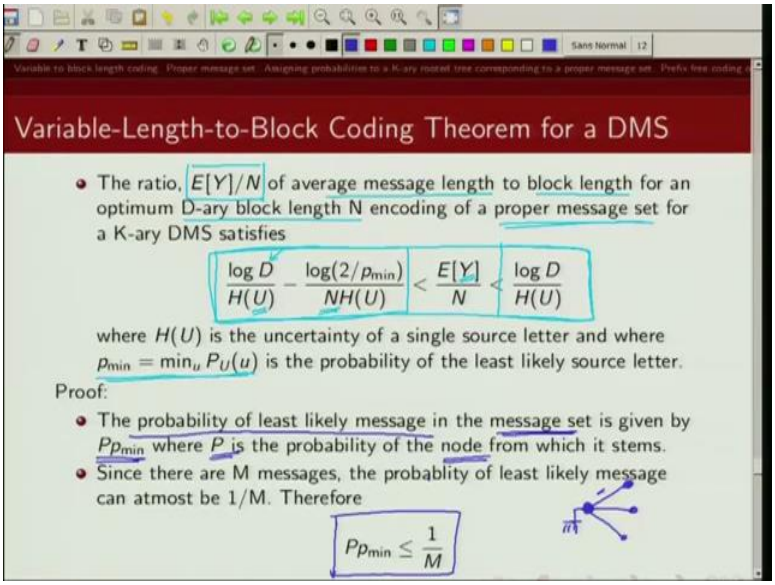
(Refer Slide Time: 61:33)



So, then I can write out those parsed message. So, you can see for example, this one corresponds to 0 0 0 0. This corresponds to 0 0 0 1. This corresponds to 0 0 1. This one corresponds to 0 1 0. This corresponds to 0 1 1. This one corresponds to 1 0 0. This one corresponds to 1 0 1, and this one corresponds to 1 1. So, those have been listed here 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 and 1 1. These are the parsed message that I am getting from the Tunstall message set. Now these are my total m messages. Since I am using binary code word of length three, now, I have to assign fixed length code to each of these messages. So, I can assign any three bit messages. So, I just I just read it 0 0 0 1 1 0 0 0 1.

Essentially from 0 0 0 to 1 1 1, those eight code words you can assign to any of this. Now this can be done in any order, I could have assign this to 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1, and you can do it anywhere you wanted basically. This is one way; this is another way of doing it. So, this is how I am doing variable to block length coding.

(Refer Slide Time: 63:44)



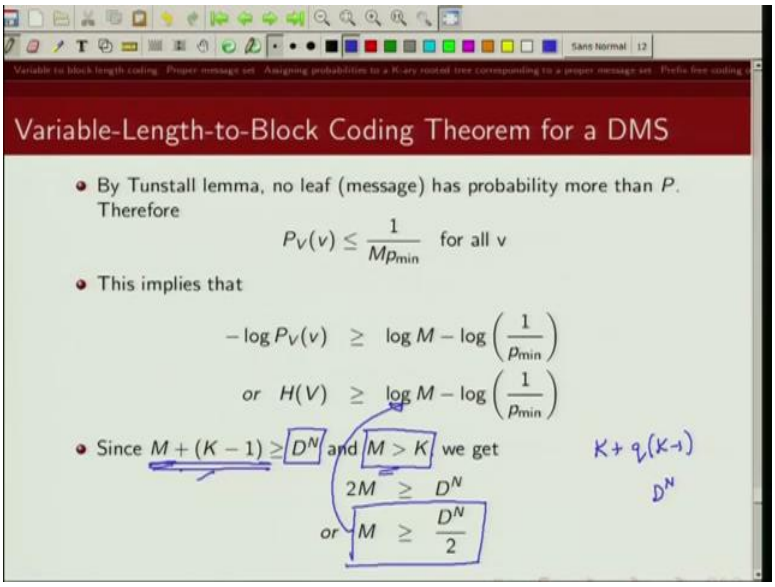
Now, let us define, I am explaining the variable to block length coding theorem for a discrete memory less source. So, this ratio of expected input length by n, which is basically average message length to the block output block length of a D-ary block length n encoding of a proper message set satisfies this relation. Remember you want to make this n by expected value of y. You want this ratio to be as large as possible. So, I am giving bounds on expected value of y given n, for a proper message set. So, this is lower bounded by this term, and upper bounded by this term. Where d is the alphabet size of the block length, output code word length, use my discrete, you use basically my source letter emitted by the discrete memory less source, p min is the minimum probability, this probability of the least likely source letter, n is my output code word length y is the length of the parsed, sequence excepted y is the average input length.

So, let us prove this result. So, the probability of the least likely message in a message set is given by p times p min, where p is the probability of the node from which it stands. Now, we know how, recall how to assign probability to a K-ary rooted tree. So, we have

a tree K-ary rooted tree, if you recall these probabilities are defined as probability of this node multiplied by probability of this letters emitted corresponding to each of these branches.

And what are the messages in the proper message set the messages is nothing, but these are the leaves of the proper message set. So, the claim I am making is the probability of the least likely message, which corresponds to a leaf in the message set, is given by p times p min, where p is the probability of the node from which it stands and p min is the probability of the least likely source letter. Now since we have total m messages right. So, this probability of the least likely message must be less than equal to 1 by m. So, probability of the least likely message in a proper message set is less than equal to 1 by m.

(Refer Slide Time: 67:12)



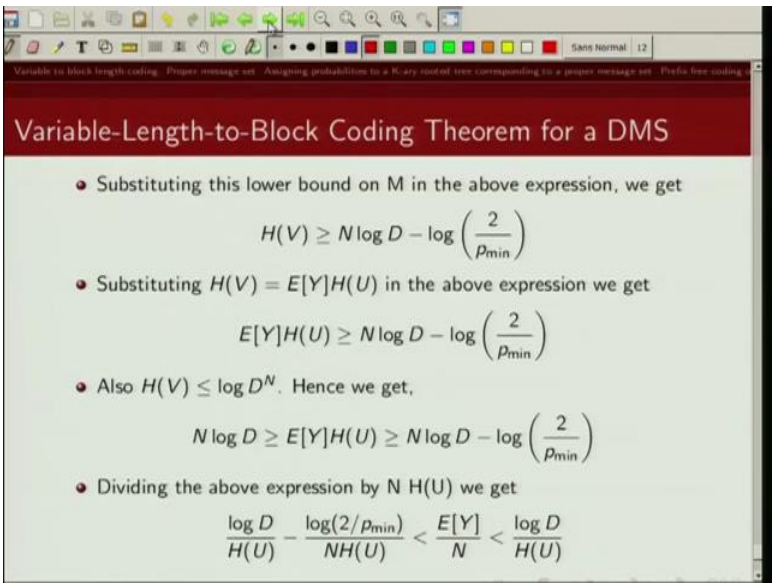
Now, according to Tunstall lemma no leaf has probability more than p; why, because in Tunstall coding we are extending the most likelihood, most likely leaf we are extending. So, there would not be any leaf that as more probability then the nodes right. So, then we can write, if this is the probability. This is the upper bound on probability of the message which correspond the probability of a leaf and then from here we can write p as less than equal to 1 by p minimum. So, we can write this that no leaf has probability more than p. So, the p, the probability of a node is basically less than equal to 1 by 1 by m p min.

Now, if you take log of this quantity, and we multiplied by minus and sum over all p of v.

Now sum over this p of v that is basically one, it would not change anything on the right hand side, you take the log of this will be minus log of m plus log of 1 by p p min, and if I multiplied by minus. So, minus log this less than equal to becomes greater than equal to, and minus log m becomes log m and this become minus log of 1 by p min. then I take summation over all p v. So, this will become entropy term here, and summation of p v this will be log n and this will minus log of p min.

So, uncertainty in the leaf basically leaf entropy is, lower bounded by this quantity. Next, now what is the number of messages we have. Number of message that we have is k plus q times k minus 1. Number of messages that we have, and number of code words we have is d power n. So, number of messages that we can have maximum, is basically up to d power n. So, m plus k minus 1 has to greater than d raise power n, because that is not the case then we could have basically done for the extension of the Tunstall message set. So, m plus k minus 1 is greater than equal to d raise to power n, and of course, our m number of messages more than k. So, from this relation I can write that two times m is greater than equal to d raise power n. So, m is greater than d power n by two. So, m is greater than equal to d raise power n by two, my plug this one in here.

(Refer Slide Time: 70:57)



So, plug this one in here what I get is, this leaf entropy is lower bounded by this quantity. Now we have proved this result which follows from the leaf entropy that uncertainty in v is given by expected value of y times uncertainty in this letter u emitted by this discrete memory less source. If I plug this one here, I get this relationship right. I also know that uncertainty v is less than number of possibilities of v, which is at most can be d raise power n. So, uncertainty in v is upper bounded by log of d n. So, here I get a lower bound, and from here I get a upper bound. So, I put this upper bound here and this lower bound this I will put here. So, combining this upper bound and lower bound I can write, that this is a relation governing expected value of y, uncertainty to u code word length, D-ary alphabet size of the output code word length, and this is the minimum probability of the single letter, emitted by the discrete memory less source.

Next I divide this expression by n times h of u, and what I get is, this relation. So, this will give me a lower bound on expected value of y given by n, as well as then upper bound on expected value of y given s. So, for a proper message set what we have come up which is bounds on average input message length divided by the output message length.

So, with this we will conclude this lecture on variable to block length coding. Now, in this lecture we have shown, if you use the proper message set it has to be a Tunstall message set that we should use for source parsing. Now in general for a non proper message set, we do not know what is the optimal way to do source parsing. So, with this I will conclude this lecture.

Thank you.