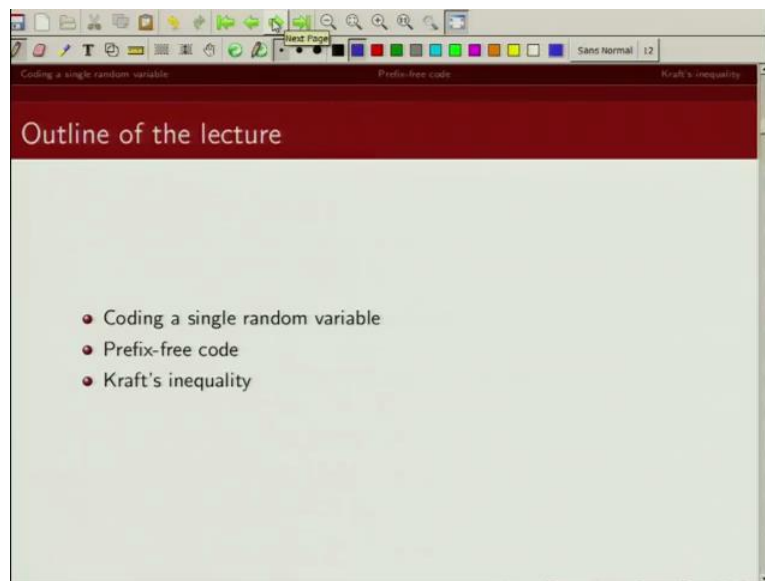


**An Introduction to Information Theory**  
**Prof. Adrish Banerjee**  
**Department of Electronics and Communication Engineering**  
**Indian Institute of Technology, Kanpur**

**Lecture – 3A**  
**Block to Variable Length Coding-I: Prefix-Free Code**

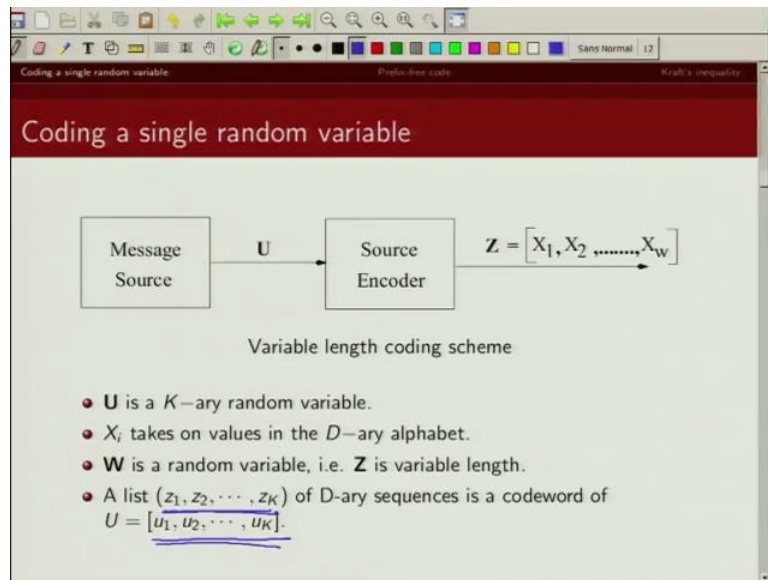
Welcome to the course on an introduction to information theory. I am Adrish Banerjee. And, today we are going to start a new topic that is on source compression, block to variable length coding. So, our input block size is fixed and we get compression by varying the output block size depending on the probability of occurrence of this block. So, in this lecture we are first going to talk about what do we mean by a prefix-free code and how we can and what is the condition for the existence of a prefix-free code.

(Refer Slide Time: 00:54)



So, we will first motivate this problem of block to variable length coding for a single random variable. And then, we will define what we mean by unique, uniquely decodable code and a prefix-free code. And then, we will talk about Kraft's inequality which talks about the existence of prefix-free code; condition for existence for prefix-free code.

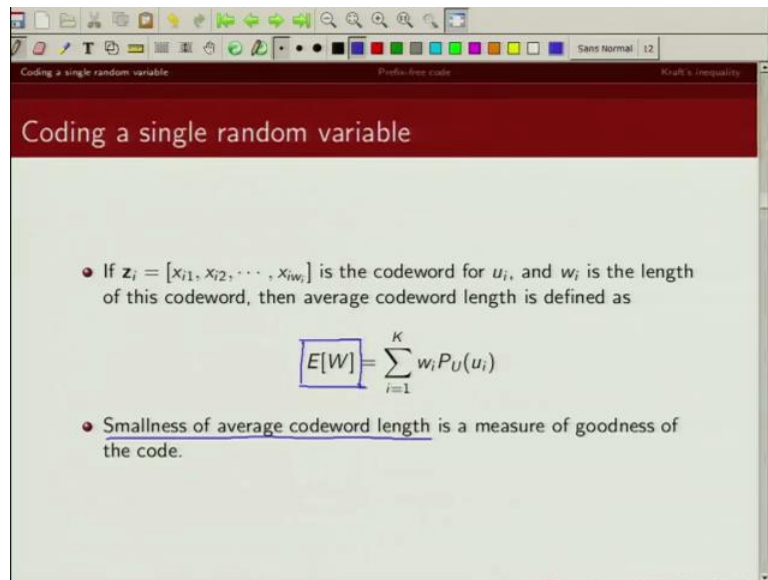
(Refer Slide Time: 01:20)



So, this is the system model that we are looking at. We have a message source that you want to compress into set of code words  $Z$ . So, we have our information. So, this is a source that we want to compress. So, this is input to our source encoder. And, output of the source encoder is the set of code words, which we denote by  $Z$ . Now,  $U$  is the  $K$ -ary random variable; that means,  $U$  takes  $K$  different possible values. Let us just call them  $U_1, U_2, U_3, U_K$ . So, these are the  $K$  possible values  $U$  takes with some probability;  $P$  of  $U_1$  and  $P$  of  $U_2$  and  $P$  of  $U_K$ . Now, our output codeword consist of  $D$ -ary alphabet, where  $D$  can be different from  $K$ . And, the length of the codeword which we are denoting by  $W$  is a random variable. So, that is variable length. So,  $W$  is a random variable. As a result, our output set of code words have variable length.

So, a list of code words is then denoted by this  $Z$  is; so,  $Z$  is are  $D$ -ary sequence. And, these are essentially codeword corresponding to our input, which is a  $K$ -ary input denoted by  $U_1, U_2, U_3, U_K$ .

(Refer Slide Time: 03:04)



The image shows a presentation slide with a red header containing the title "Coding a single random variable". Below the header, there is a bulleted list. The first bullet point states: "If  $z_i = [x_{i1}, x_{i2}, \dots, x_{iw_i}]$  is the codeword for  $u_i$ , and  $w_i$  is the length of this codeword, then average codeword length is defined as". Below this text is a mathematical equation: 
$$E[W] = \sum_{i=1}^K w_i P_U(u_i)$$
 The second bullet point states: "Smallness of average codeword length is a measure of goodness of the code." The slide also shows a standard software toolbar at the top and a status bar at the bottom with the text "Saas Normal 12".

Now, let us assume that  $Z_i$  is the codeword corresponding to the source  $U_i$ . And, let us denote by  $W_i$ , the length of this codeword. So, this  $Z_i$  is  $X_{i1}, X_{i2}$  up to  $X_{iw_i}$ . So,  $W_i$  is the length of this codeword  $Z_i$ , which is the codeword corresponding to  $U_i$ . So, then what is the average codeword length? Average codeword length is basically given by length of the codeword for  $U_i$ , probability of occurrence of  $U_i$ , and this we sum over all  $K$ -ary alphabets of  $U$ . So, this is my average codeword length.

Now, when I am doing block to variable length coding, I want this average codeword length to be low. We are subsequently, see what is our minimum number of digits are required to reproduce the source losslessly. So, then the smallness of this average codeword length is the measure of how good my source encoder is. So, if I can represent my source  $U$  in smaller number of digits without losing information, then that is a better source encoder. So, our objective is to find what is a minimum codeword length required to encoder source  $U$ .

(Refer Slide Time: 04:51)

**Uniquely decodable codes**

- A code is uniquely decodable if every finite sequence of the code corresponds to at most one message.
- One way to ensure unique decodability is to require no codeword to be prefix of another codeword.
- A D-ary sequence  $z$  is a prefix of another D-ary sequence  $z'$  if  $z$  has length  $n$  and the first  $n$  digits of  $z'$  form exactly the sequence  $z$ .
- No codeword should be the prefix of another codeword for instantaneous decodability.
- A code for  $U$  that satisfies the above condition is known as prefix-free code.
- Can a non-prefix free code be uniquely decodable?

$U$	$Z$
$u_1$	0
$u_2$	10
$u_3$	11

Prefix-free code

$U$	$Z$
$u_1$	1
$u_2$	00
$u_3$	11

Not a prefix-free code

Now, let us move into defining what do we mean by a uniquely decodable code and what is a prefix-free code. So, a code is uniquely decodable, if for every finite sequence of code corresponds to at most one message. So, we can uniquely decode if a finite sequence of code corresponds to only at one message. Now, one way to ensure unique decodability is to ensure that no codeword is a prefix of another codeword. So, if we can ensure that no codeword is a prefix of any other codeword, then we can ensure unique decodability.

And, what do we mean by prefix of another codeword? So, let us say you have a D-ary sequence  $Z$ . Now, it is a prefix of another D-ary sequence  $Z'$ . If  $Z$  is of length  $n$  and the first  $n$  digits of  $Z'$  are exactly same as that of  $Z$ . So, let us just take an example. Let us say  $Z$  is 010 and this  $Z'$  is 10110. Now, you can see  $Z$  is not a prefix of  $Z'$ ; it is not a prefix of  $Z'$  because 010, this  $Z'$  does not start with 010. But, if I consider any other codeword, let us say  $Z$  is 11101, and then you can see that the first three digits of this  $Z'$  are same as this  $Z$ . So, in this case  $Z$  is a prefix of  $Z'$ .

So, we say a D-ary sequence is a prefix of another D-ary sequence  $Z'$ . If  $Z$  is of length  $n$ , then obviously length  $n$  is greater than or equal to 1. And, the first  $n$  digits of  $Z'$  are exactly same as the sequence  $Z$ . Now, if we want instantaneous decodability, please note that in this we are talking about variable length code. So, when we are talking about variable length codes, we need to know where the code boundaries are. For

example, if your codeword is 010110111. Now, if I have a string of code words just write something. Let us just see 010110111. Now, note here if this is the set of code words that I have codeword corresponding to U 1, codeword corresponding to U 2, codeword corresponding to U 3, codeword corresponding to U 4. And, this is the coded sequence that I am sending. Then, I can easily find out what these code words are. So, look 0, 0 corresponds to codeword of U 1; then 1, there is no codeword with 1; 10, yes, and this is this codeword. Now 110 that is another codeword; 111 that is another codeword; zero that is another codeword; 111 that is another codeword.

So, you can see, I know precisely where these code boundaries are. So, I can do instantaneous decodability. As supposed to if I let us say change the codeword. So, let us say 0 to 11 and 110. And, let us say 01. If I change it something like this, let us say this is my codeword. Let us say this is my set of code words. And, in this case you can see if I get 11, I will have to wait and see what the next bit is. If the next bit is a single 1,0, then it is probably this codeword; if there are even number of zeros, then probably this codeword has been transmitted.

So, in this particular example; this is some example of a code, which is not instantaneous decodable because I will have to look beyond what I have received to actually decode back the sequence. Whereas, in this particular example when I get a zero, I exactly know that my, this is the codeword for U 1; when I get one, I know I need to look further; when I get 10, I know exactly that this is the codeword corresponding to U 2. Whereas, in this particular example if I get 11, I am not able to make a decision whether this codeword is transmitted or this codeword, I need to go further to make a decision.

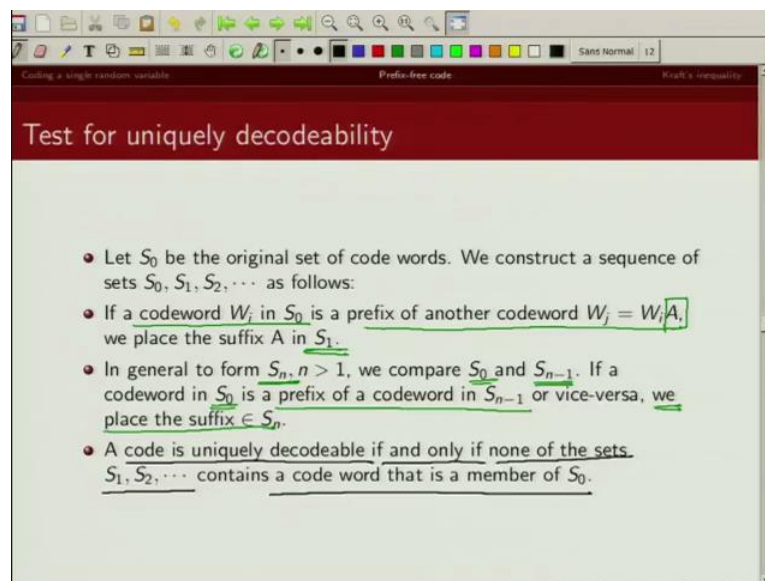
So, in case of instantaneous decodability, these code boundaries are exactly known. So, I can just decode the codeword. I will write there. So, no codeword should be a prefix of another codeword, if you want your code to be instantaneous decodable. And, you can see in this particular example, none of these code words are prefix of any other code. This codeword is 0. You can see none of these three code words have zero as prefix. This has 10. Again, these two code words do not have 10 as its prefix. And, this is 110 and 111; they are not prefix of any other codeword. So, for instantaneous decodability we want this condition that no codeword should be prefix of another codeword.

Now, code that satisfies this property that no codeword is a prefix of any other codeword,

this particular code is known as prefix-free code. And, as we can see from the example that we did, a prefix-free code can be instantaneously decodable. So, this is an example of a prefix-free code. You can see a codeword corresponding to U 1 is 0, codeword corresponding to U 2 is 10 and codeword corresponding to U 3 is 11. Now, if I get any set of sequence, let us just call it this is 01011010. You can see I can exactly find out what the code words are. So, this is the codeword corresponding to U 1, this is the codeword corresponding to U 2, this is the codeword corresponding to U 3, this is the codeword corresponding to U 1, this is the codeword corresponding to U 2.

Now, look at this example. Is this a prefix-free code? No. Why because U 1 code is 1. And, you can see U 1 is a prefix of this codeword U 3 because U 1 is 1 and U 3 also starts with 1. So, then this is, this particular code is not prefix free. Now, the question is can we decode a non-prefix free code? The answer is yes. But, they cannot be instantaneously decodable. So, in the next slide we will talk about the condition for uniquely decodable code.

(Refer Slide Time: 13:05)



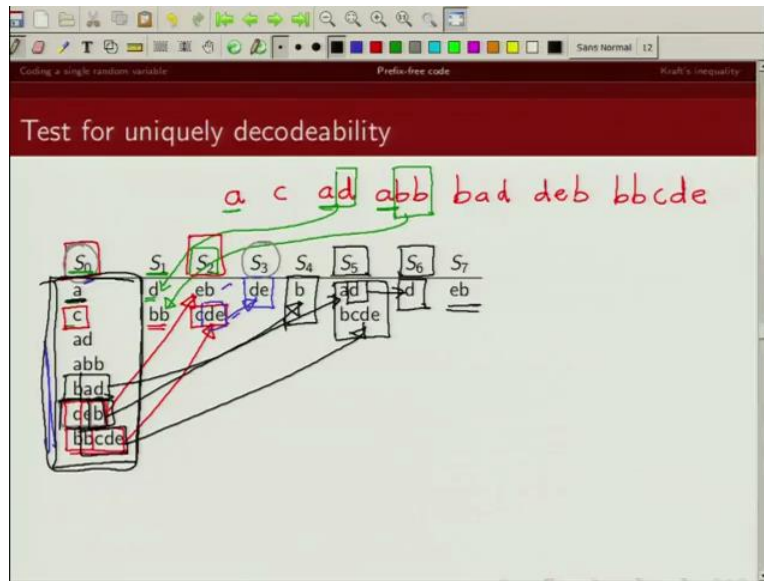
Now, let us just take an example. So, let us say we have a set of code words. And, if say they corresponds to; this is the set of partitioning of the code words, let us say A 1, A 2, A 3, A 4. And, let us say I can have another set of code words corresponding to the same bits. Let us call it B 1, B2, B3, B 4. This is an example where set of set of strings, basically it can be; set of code words that can be partitioned in two different ways. One is

A 1, A 2, A 3, A 4; another is B1, B 2, B 3, B 4. So, this is the case of code not being uniquely decodable.

Now, look at what is the condition that we are getting when the code is not uniquely decodable. One thing that we can see here is there is one codeword, which is a prefix of another codeword. Here, we can see A1 is the prefix of this codeword B 1. And then, whatever the dangling suffix, the suffix which is; if from B 1, I remove A 1, so whatever suffix is remaining, I can see this suffix is a prefix of another codeword A2. This is the prefix of another codeword A 2. So, what I can see here is in the case when the code is not uniquely decodable, it is not a prefix-free code. And, whatever suffix is remaining from a particular, after you take out the prefix that suffix is then again part of another codeword, that is, a prefix of another codeword. So, the test for uniquely decodable code words can then be written as follows.

Let us assume that we denote by  $S_0$ , the set of original code words. And, we want to find out whether these set of code words are basically or uniquely decodable. So, we construct a set of sequence of sets  $S_0, S_1, S_2$  as follows. If a codeword  $W_i$  in  $S_0$  is a prefix of another codeword, so if it is a prefix of another codeword, then we would have another codeword  $W_j$ , which would be of the form  $W_i A$ , where  $W_i$  is basically a codeword in  $S_0$  and  $A$  is some suffix in  $S_1$ . So, if we have a codeword  $W_i$  in  $S_0$ , which is prefix of another codeword, then what we do is we place this suffix. So, after you remove this prefix  $W_i$ , whatever is remaining is suffix. So, we put this suffix  $A$  in  $S_1$ . Now, to explain this let us just fast forward and look at this example.

(Refer Slide Time: 16:24)



So, I have this set of code words  $a, c, a, d, a, b, b, b, a, d, d, e, b$  and  $b, b, c, d, e$ . Now, let us look at these code words and see if any of these code words are prefix to any other codeword. So, let us look at this  $a$ . Is  $a$  a prefix to any other codeword; Yes, it is a prefix to  $a, d$ ; it is a prefix to  $a, b, b$ . So, what is the suffix left if I remove this prefix? That is  $d$ . So, I put this  $d$  in this set  $S_1$ . Next, here if I remove this prefix  $a$ , what is left is this suffix  $b, b$ . I put this  $b, b$  in  $S_1$ . Any other codeword; I do not see any codeword which has  $c$  as prefix; any of these code words? Similarly,  $b, d, b, a, d$ ; there is no codeword, which has  $b, a, d$  as its prefix. So,  $S_1$  will have  $b$  and  $d$ , sorry,  $d$  and  $b, b$ . So, that is what I meant. If you have a codeword  $W_i$  in  $S_0$ , which is a prefix of another codeword, then you put this suffix in this set  $S_1$ .

Now, in general we will form this set  $S_n$ , in this fashion. We will compare  $S_0$  and  $S_{n-1}$ . If there is a codeword in  $S_0$ , which is a prefix of another codeword in  $S_{n-1}$  or vice versa, we place the suffix in  $S_n$ . So, how do we form this set  $S_n$ ? We look at  $S_0$ , code words in  $S_0$  and code words in  $S_{n-1}$ . And, if any code in  $S_0$  is a prefix of any codeword in  $S_{n-1}$  or any codeword in  $S_{n-1}$  is a prefix of any codeword in  $S_0$ , then we take the suffix out and place it in this set  $S_n$ .

So, let us go back to our example. So, how do we get this set  $S_2$ ? So, we need to compare this set  $S_0$  and  $S_1$ . We need to compare this set  $S_0$  and  $S_1$ . So, what do we have here? So, let us look at  $a$ . Do we have any code words in  $S_1$ , which has  $a$  as prefix?



No, we do not see the same thing with c, a d, a b b, b a d, d e b, b b c d e. We do not see any code, any of these code words been prefix of any code words in S 1.

Now, look at code words in S 1. d. Do we have any codeword here which has d as its prefix? The answer is yes. And that is this codeword. So, then what we will do? We will remove this prefix and we will put this suffix in set S 2. Any other code words in S 0 which are starting with d? No. What about d b? Do we have any codeword in S 0 which start with b b? The answer is yes; that is this one. Now, what do we do? We remove this prefix and we put this suffix, which is c d e in this set S 2. So, this is how we get our set S 2.

Now, how will we get our set S 3? Again, we are going to compare S 0 and S 1. We will first try to see is there any codeword in S 0, which is a prefix of codeword in S 2 or is there any codeword in S 2 which is a prefix of codeword in S 0. So, let us look here. So, we can see here S 0 has codeword c, and this has a codeword c d e. So, clearly c is a prefix of c d e. Then, what is the suffix? The suffix is d e. And, we put this suffix here in the set S 3. Is there any other codeword? You can check, a is not a, code, prefix of e b or c d e. a d is not a prefix of any of these two code words. Similarly, none of these code words are prefix of these code words. And, we can check e b. There is no codeword in S 0 which starts with e b. Similarly, there is no codeword in S 0 which start with c d e. So, S 3 will only consist of d e.

Now, how do we find this set S 4? We are now going to compare S 0 with S 3. So, compare S 0 with S 3. Now, let us, d e. Do we have any codeword in S 0 which starts with d which has prefix d e? The answer is yes; that is this one. So, if you remove this prefix d e, the suffix that is left is b, and then S 4 will be b. Do we have any codeword in S 0 which is a prefix of codeword in S 3? We can check a c, a d, a b b. None of these are prefix to d e. So, S 4 consists of only b.

Now, in the similar fashion we can find the set S 5. So, we look at code words in S 0 and we look at his codeword in S 4. Now, any of these code words have been prefix to this codeword? No. Let us look at code words in S 0 which starts with b. So, that is, one of them is this; b a d. So, the suffix here is a d. So, a d is here and the next codeword is b b c d e. So, here the suffix is c d, b c d e. So, this b c d e comes in set S 5.

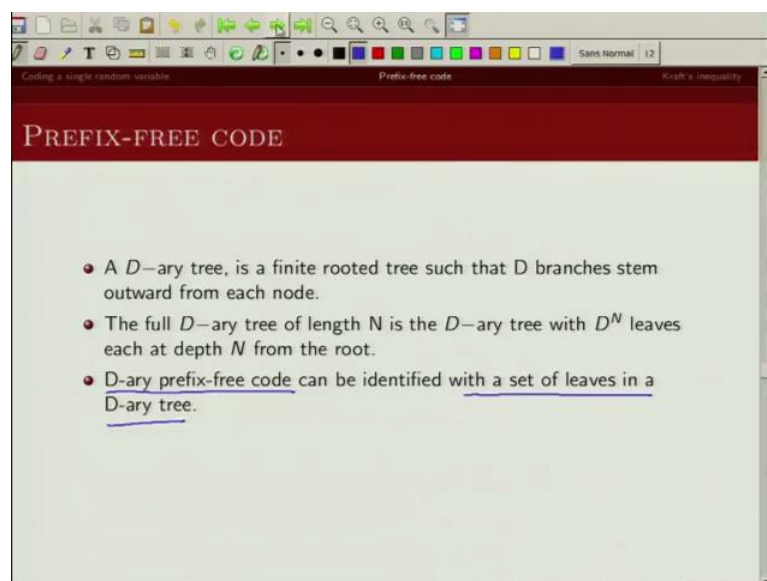
How do we find this set S 6? Again, we are going to look at these code words and

compare it with these code words. And, we will try to find out if any codeword in  $S_0$  is a prefix of codeword in  $S_5$  or is there any codeword in  $S_5$  which is a prefix of a codeword in  $S_0$ . So, we can see  $a$  is a prefix. And that is the only prefix that we can find.  $a$  is a prefix of  $ad$ . So, the suffix here in this case is  $d$ . So, this  $S_6$  is  $d$ .

And, similarly we can find out  $S_7$ ; because if we compare  $S_0$  and  $S_6$ , we see there is a codeword here;  $deb$ , which has  $d$  as prefix. So, the suffix is  $eb$ . Now, do we have any codeword in  $S_7$  or  $S_0$  which is a prefix of each other? So, is there any codeword in  $S_0$  which is a prefix of codeword in  $S_7$ ? The answer is no because we do not have any codeword here starting with  $e$ . Similarly, there is no codeword in  $S_0$  starting with  $eb$ . So, we stop here.

So, this is how we find these set  $S_1, S_2, S_3, S_n$  of  $n$ . Now, once we have found the set, a code is uniquely decodable, if and only if none of these sets  $S_1, S_2, S_3, S_n$  contains the codeword that is a member of  $S_0$ . So, the condition that we want is none of these codeword like  $d, bb, eb, cde, deb$ , none of these should be part of  $S_0$ . So, let us look at these code words in  $S_1, S_2, S_3, S_4, S_7$ . And, they should not have anything common with  $S_0$ . Now, that is not the case here. You can see  $S_5$  has  $a$  and  $S_0$  also has  $a$ . So, this is not a uniquely decodable code.

(Refer Slide Time: 26:32)



Now, we have described what is a prefix-free code; that no codeword should be prefix of any other codeword. Now this prefix-free code, we can also represent it using a  $D$ -ary

tree. So, let us define what is a D-ary tree. So, a D-ary tree is a finite rooted tree, such that there are D branches stemming outwards from each node.

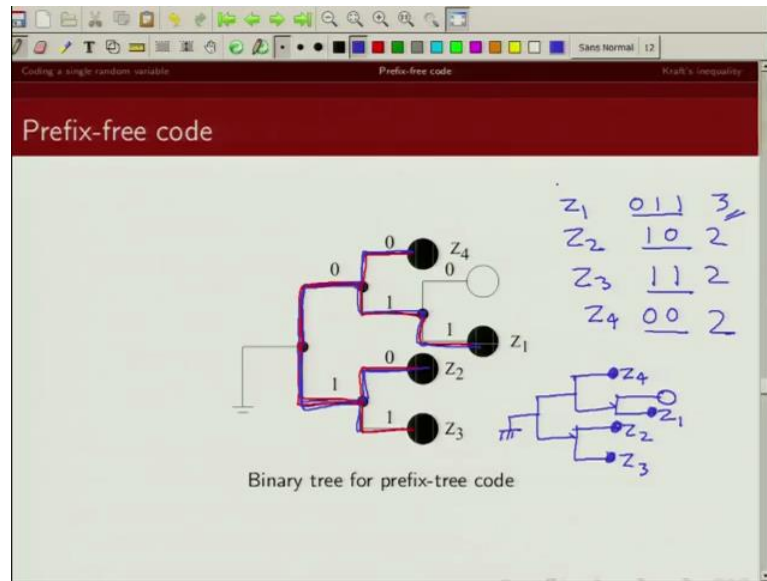
So, let us take an example. Let us say D is equal to 2. So, that is a binary tree. So, binary tree will be something like this. So, this is my root and there are two branches stemming from each node. So, from the root node there are two branches; from this, there are two branches. So, a D-ary tree is a finite rooted tree, such that there are D branches stemming from each node. These are nodes, leaves.

We say a D-ary tree is full of length N. If it is a D-ary tree with  $D^N$  leaves at depth of N from the root. So, let us define first what we mean by depth and what do we mean by leaves. So, you have a tree. This is a root and it is a D-ary tree. So, there are D branches stemming from each node. Let us consider the example of D equal to 2; binary tree. So, there are two branches stemming from each node. This is a root node. So, there are two branches coming from this node. So, this is, depth is defined as how far from the root, these leaves are. So, this is at depth one.

Now, leaf is basically a node from where there are no further branches stemming. So, if I draw a tree like this, this has basically two branches and these are my leaves. Now, if I draw a tree like this, you can see, this is, these nodes are at depth one. And then, you have leaves at depth two. The leaves are basically node from which there is no branching, no branch getting out. And, this is depth one; this is depth two. Similarly, I can have. This is a binary tree up to depth three. And, there are eight leaves possible at depth three. So, a D-ary tree of length N is full, if it has  $D^N$  leaves at the depth.

So, for example, if I draw a tree like this, this is not a full tree of length two. Why because I do not have four leaves. Here, I only have; this is only extended up to depth. This particular branch is only extended up to depth one from the root. So, this is not a full tree. However, if I extend this, if I extend this node and I have something like this, then it is a full tree because I have four leaves at depth two. So, a full tree will have all  $D^N$  leaves. So, it is, all the nodes are extended up to depth N; that is the full D-ary tree of length N.

(Refer Slide Time: 31:34)



Now, a D-ary prefix-free code can be identified by a set of leaves in a D-ary tree. So, the claim I am making is we can identify a D-ary prefix-free code from the set of leaves of a D-ary tree.

Let us take an example. So, I have code words. Let me write  $Z_1$  is 011,  $Z_2$  is 10,  $Z_3$  is 11 and  $Z_4$  is 00. Now, you can see that none of these code words  $Z_1, Z_2, Z_3, Z_4$  are prefix of each other. And, we can see that since none of them is prefix of each other, each one of them can be represented using this D-ary tree. So, here I have binary bits 0 and 1. So, I can consider a binary tree. So, and the maximum length of the codeword is 3. So, I will extend up to maximum three. So, let us see. I want  $Z_1$ .  $Z_1$  is 011. So, I extend this root node to 0 and 1. Then, to get 011 I extend this node 0 to 00 and 01. So, I get 01. And then, I extend it to 1. So, that is my  $Z_1$ .

Similarly, how do I get  $Z_2$ ? I have this 1 and then I take this leaf 10, that is, my  $Z_2$ .  $Z_3$  is 11. That is this. And,  $Z_4$  is given by this. So, you can say from the root there is a unique path. Path from root to  $Z_1$  is this. And, there is no codeword along this path. Similarly, path from root to  $Z_2$  is unique. There is no codeword in this path.  $Z_3$ , there is a unique path from root;  $Z_4$ , there is a unique path from root. So, we can represent these prefix-free codes as leaves of a D-ary tree. Now, having represented a prefix-free code using a D-ary tree, now let us prove what is the condition for existence of a prefix-free code.

(Refer Slide Time: 33:59)

The image shows a presentation slide with a red header containing the title "Kraft's inequality". Below the header, there is a bulleted list. The first bullet point states: "There exists a D-ary prefix-free code whose codeword lengths are the positive integers  $w_1, w_2, \dots, w_K$  if and only if". Below this text is a mathematical equation enclosed in a hand-drawn blue box: 
$$\sum_{i=1}^K D^{-w_i} \leq 1$$
. Underneath the equation, the text "Sketch of Proof:" is followed by two more bullet points. The second bullet point says: "In the full D-ary tree of length N,  $D^{N-w}$  leaves stem from each node at depth w where  $w < N$ ". The third bullet point says: "Suppose there exist a D-ary prefix-free code, construct a tree for the code by pruning the full D-ary tree of length  $N = \max_j w_j$  at all vertices corresponding to codewords.". The slide also shows a standard software toolbar at the top.

So, the Kraft's inequality talks about when does a prefix-free code exist. So, it says there exists a D-ary prefix-free code, whose codeword lengths are given by  $w_1, w_2, w_3, \dots, w_K$ , if and only if this condition holds. So, what does it mean? It means that if there exists a prefix-free code which has codeword length given by  $w_1, w_2, \dots, w_K$ , then this relationship must hold. And, if this relationship holds and there are code words of length  $w_1, w_2, \dots, w_1$ , then that codeword must be; that code must be a prefix-free code. So, let us prove this.

Now, what we have shown so far is we can represent a prefix-free code using a rooted D-ary tree. So, if you have a full D-ary tree, now again what is the full D-ary tree of length N? It is a tree which has  $D^N$  leaves. So, all the nodes are extended up to length N. So, in a full D-ary tree the number of leaves stemming from each node at depth w is given by  $D^{N-w}$ . We can see this.

Let us just draw a binary tree. We are just drawing a binary tree of depth 3. This is a full D-ary tree of length 3. And there are, these are my eight leaves. Now, you can see that if you look at nodes at depth two which is this node, this node and this node like N in this case is 3; D in this case is 2.

Now, let us look at nodes at depth two which is this node, this node and this node. Now, take any node. Let us say we take this node. Now, how many leaves are stemming from this particular node? That is, this leaf and this leaf; that is two. And, this is what we get

from here;  $D$  is 2,  $N$  is 3,  $W$  here is 2. So, there are two leaves which are stemming from this node.

Similarly, you can take from any. This is a again another node at depth 2. There are two leaves which are stemming from this node. And, this is what I am claiming here. Now, take a node at depth 1. So, let us take this, this corresponding to  $W$  equal to 1. Now, how many nodes, how many leaves are stemming from this node? That is 1, 2, 3 and 4. You see this is one leaf, this is one leaf, this is one leaf and this is one leaf. So, there are four leaves, which are stemming from this node at depth 1. And, you can again check  $D$  is 2,  $N$  is 3,  $W$  is 1. So, that should be this is 2 that should be 4. And, that is what we are getting.

So, if you have a full  $D$ -ary tree of length  $N$ , you have  $D$  raise to power  $N$  minus  $W$  leaves, which are stemming from each node at depth  $W$ . Where, mostly  $W$  is less than  $N$ . So, suppose there exist a  $D$ -ary prefix-free code, so we are first trying to prove that if there exist a  $D$  free,  $D$ -ary prefix-free code, whose codeword lengths are  $W_1, W_2, \dots, W_K$ , then this condition must be satisfied. So, we are saying there exist; let us say there exist a prefix-free code;  $D$ -ary prefix-free code. So, if there exists a  $D$ -ary prefix-free code, we can construct a  $D$ -ary tree corresponding to this prefix-free code. And, how do we construct this tree? We construct this tree by pruning its full  $D$ -ary tree of length  $N$ , when  $N$  is a maximum of  $W_1, W_2, \dots, W_K$ . And, we are going to prune it at length  $W_1, W_2, \dots, W_K$ . We are going to prune that. So, that is how we are going to get a prefix free, a  $D$ -ary tree representation of a prefix-free code.

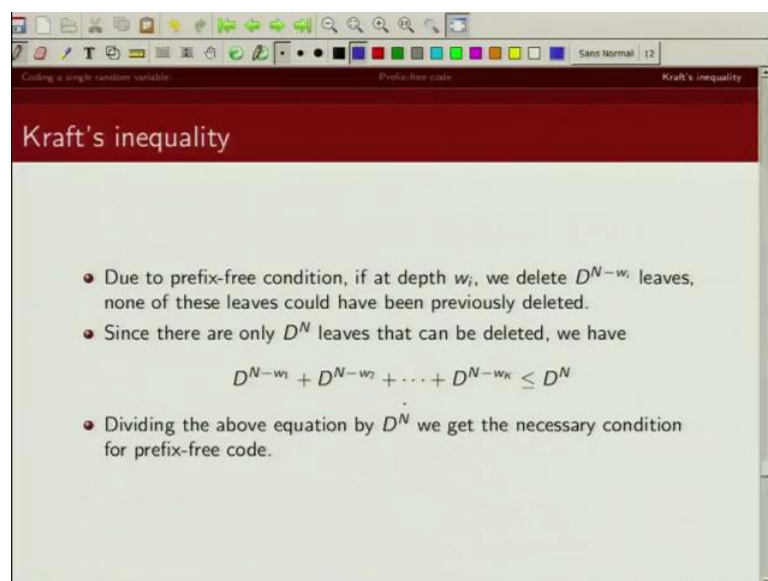
For example, if you look at this particular case; now, here we require a prefix-free code of length 3, 2, 2 and 2. So, how do we get it? We can construct a  $D$ -ary tree up to depth 3 and then we start pruning them at, to get  $Z_1; I, 1, 2, 3$ . So, this is I just pruned it. So, I just pruned a tree at three. Then, for  $Z_2$  I just pruned the tree. So, I do not take into account the other two leaves, which are stemming from this node  $Z_2$ . So, to construct a prefix, to construct a  $D$ -ary tree corresponding to this prefix-free code, as I said I will first construct a full  $D$ -ary tree, right, of depth 3 because maximum codeword length is 3. So, this is my  $D$ -ary tree of depth of length 3.

Now, I can assign these code words by pruning the tree at appropriate level. So, for example, for a  $Z_3$  I need 1 and 2. So, I just prune this tree here. For  $Z_2$ , again I need 1

and 2. So, I just prune this tree here. For Z 1, I need 3. So, I have 1, 2, 3. So, just, let me just write it as well. This can be; let us say this can be Z 3, this can be Z 2, and this can be Z 1. And then, for Z 4 again I need 2. So, I can just take this and this. I can just prune this tree. So, this can be Z 4. So, these are my leaves corresponding to my prefix-free code and this is a unused leaf. I am not using this leaf.

So, I hope it is clear how we can map a prefix-free code to a D-ary tree of length N. And, this we do it by pruning the D-ary tree at appropriate length W 1, W 2, W 3, W k.

(Refer Slide Time: 42:09)



Now, due to prefix-free nature of this code, if at W i, I delete these many leaves, none of these leaves would have been previously deleted. Why because my codeword is prefix free. So, there is only one path from the root to this particular codeword. So, this D N minus W i leaves that I am pruning to construct a or to map a prefix-free code of length W i, those leaves would not have been removed earlier. So, then to map a prefix-free code of length W 1, I am deleting so many leaves; to map a prefix-free code of length W 2, I am deleting so many leaves; to map a prefix-free code of length W K, I am deleting so many leaves.

Now, what is the maximum number of leaves I can have? That is basically corresponding to number of leaves in a full D-ary tree, and that number is D raise power N. So, total number of leaf that I can delete while mapping each of these prefix-free codes; this must be less than equal to D raise power N. So, then I divide the left hand side and right hand

side by  $D$  raise power  $N$ . What I get is this condition that summation  $D$  raise power minus  $W_i$  sum over  $i$  to  $K$  must be less than equal to 1. And, thus this is known as Kraft's inequality. So, what we have shown is if there exists a  $D$ -ary prefix-free code, then we can map that prefix-free code to a corresponding  $D$ -ary tree. And, we have shown that if a prefix-free codeword, a code has codeword length  $W_1, W_2, W_3, \dots, W_K$ , and then this Kraft's inequality has to be satisfied.

(Refer Slide Time: 44:51)

**Kraft's inequality**

$K=3$   
 $w_1=2, w_2=2, w_3=3$      $D=2$

- Sketch of converse proof: Suppose  $w_1, \dots, w_K$  are positive integers such that Kraft inequality is satisfied. Consider an ordered list of length  $w_1 \leq w_2 \leq \dots \leq w_K$ . Consider the following algorithm
- (1)  $i \leftarrow 1$
- (2) Choose  $z_i$  as any surviving node or leaf at depth  $w_i$ , and prune the tree. Stop if there is no such surviving node or leaf.
- (3)  $i = K$  stop, otherwise  $i \leftarrow i + 1$

$w_1=2$   
 $w_2=2$   
 $w_3=3$

Now, we are going to show that if this Kraft's inequality is satisfied. And, if you have a codeword of length  $W_1, W_2, W_3, \dots, W_K$ , then this must be a prefix-free code. So, we are proving the converse. So, suppose  $W_1, W_2, W_3, \dots, W_K$  are positive integers, such that Kraft's inequality is satisfied. Now, without loss of generality we are just considering an ordered list of this codeword length. So, we are assuming  $W_1$  is less than equal to  $W_2$  is less than equal to  $W_3$ . And, which is subsequently  $W$  less than equal to  $W_K$ .

Now, let us consider the following algorithm. So, start with  $i$  equal to 1. So, we start with  $W_1$ . Now, choose  $Z_i$  as the surviving node or leaf at depth  $W_i$  and prune the tree. So, what we are trying to do is we are given a set of code words which have length  $W_1, W_2, W_3, \dots, W_K$ . And, what we are going to do is we are going to map these code words to this  $D$ -ary tree. So, how are we mapping it? We are mapping this codeword  $Z_i$  as a surviving node at depth  $W_i$  and then we prune the tree. Stop, if there is no surviving node or leaf. And, this process will continue from  $i$  equal to 1 to  $K$ .



So, let us take an example. Let us say  $K$  is 3 and let us take  $W_1$  to be 2,  $W_2$  to be 2 and  $W_3$  to be 3. So, what we are saying is this algorithm is as follows. So, first we consider an ordered list of code words of length  $W_1, W_2, W_3$  and we choose  $Z_i$  as a surviving node at depth  $W_i$  and prune the tree. So, here you can see  $W_K$  is 3; that is the maximum depth.

So, we will consider a, and again let us consider  $D$  to be 2. So, let us consider a binary tree. So, if we draw a binary tree of length three, it will look something like this. So, binary tree of length three will have eight leaves. Full tree; 1,2,3,4,5,6,7,8.

Now, how are we going to map these code words of length  $W_1$  equal to 2? So, what we are going to do is so from the root, you go at depth one and depth two. I go here and then I prune all these leaves which are starting from this. So, I prune this. So, this is my codeword corresponding to  $W_1$  equal to 2.

Next, I will look at this codeword which is of length two. So, again I start from here. Let us say I go here; 1, 2. So, I delete this. So, what I have here is this is; so, I deleted those two leaves which were starting from this node. And, if I consider  $W$  equal to 3, then I have this 1, 2, and one second, 3. I can consider this, this  $W_3$  corresponding to 3. And, rest is all unused. Rest is all unused leaves. These are all unused leaves. So, to repeat what I said, to map each of these code words of length  $W_1, W_2, W_3$ , what I do is I first create a full  $D$ -ary tree of length  $N$ , which is maximum value of  $W_i$  which is  $W_K$ . And then, I go at depth  $W_i$ , which is  $W_1, W_2$ , and  $W_3$ . And, I delete all the nodes and the leaves or the branches which are starting from that particular node at depth  $W_i$ .

(Refer Slide Time: 49:55)

The image shows a presentation slide with a red header containing the title "Kraft's inequality". The slide content includes a bullet point explaining the construction of a prefix-free code, a mathematical equation, and another bullet point discussing surviving leaves at depth N. The equation is 
$$D^N - (D^{N-w_1} + D^{N-w_2} + \dots + D^{N-w_{i-1}}) = D^N \left( 1 - \sum_{j=1}^{i-1} D^{-w_j} \right)$$

Now, if I am able to map all these code words  $W$  of length  $W_1, W_2, \dots, W_K$ , then I will ensure that all the prefix-free codes have been mapped. So, if we choose our  $Z_K$  and we are able to construct a prefix-free code, then we are done. Then, we are able to show that the codeword is prefix-free because there is a unique mapping of each of these code words to our  $D$ -ary tree.

So, let us assume that  $Z_1, Z_2, Z_3, \dots, Z_i$  have already been chosen. So, if they have already been chosen, then when I choose  $Z_1$ , at that time I deleted these many number of leaves which is  $D$  raise power  $N$  minus  $W_1$ ; when I choose  $Z_2$ , I deleted these many leaves; when I choose  $Z_i$ , I deleted this many number of codes. So, then when I am at point and try to map  $Z_i$ , how many leaves are left? The number of leaves which are left is; this is the number of leaves for a full tree; this is the number of leaves I have deleted to map  $i$  minus one codeword. So, the remaining leaves are then given by this. So, this can be further simplified into this expression.

Now, please note if the Kraft's inequality is satisfied, then we know  $\sum_{1 \text{ to } K}$ . This sum is less than equal to 1. In other words, if Kraft's inequality is satisfied, then this summation would be greater than zero. What does it mean? That means, there still exists leaves which can be deleted and I can map my  $Z_i$ . So, you can see that if Kraft's inequality is satisfied, this quantity is greater than zero. So, I still add. So, there are still leaves left. So, there are surviving leaves left at depth  $N$ , which is greater than 0.

And, if I have leaves left in my prefix-free in my D-ary tree, I can always map my prefix-free code to such leaves. So, if there is a surviving leaf at depth N, there must be unused nodes at depth  $W_i$ , which is less than N. Why because there is one-to-one mapping between this D-ary tree and prefix-free code. So, if there are surviving leaves at depth N, it means there are unused nodes at depth  $W_i$  less than N, which have not been chosen as codeword by any of these previous codeword  $Z_1, Z_2, \dots, Z_{i-1}$ . And, thus I can assign  $Z_i$  to this particular node. And, note this condition will hold until  $i$  is K because from Kraft's inequality we know that from if  $j$  is 1 to K, summation  $D^{-j}$  is less than equal to 1.

So, in other words what we have shown is then each of these codeword of length  $W_1, W_2, W_3$  up to  $W_K$  can be mapped to a D-ary tree. And, we know that if we can map these code words to a D-ary tree, then they correspond to code words of a prefix-free code. So, hence we have shown that if the Kraft's inequality is satisfied and we have code words of length  $W_1, W_2, \dots, W_k$ , then this code must be a prefix-free code.

(Refer Slide Time: 54:07)

**Prefix-free code**

- Construct a ternary prefix-free code with lengths  $w_1 = 1, w_2 = 2, w_3 = 2, w_4 = 3, w_5 = 4$ .
- Since  $\sum_{i=1}^5 3^{-w_i} = 1/3 + 1/9 + 1/9 + 1/27 + 1/81 = 49/81 < 1$ , a prefix-free code exists.

Let us just take an example to illustrate how we can map a prefix-free code to a D-ary tree. So, consider a ternary prefix-free code. So, ternary means D is equal to 3 and these are my codeword lengths;  $W_1, W_2, W_3, W_4$ , and  $W_5$ . They are already ordered. You can see  $W_1$  is equal to  $W_2$  is equal to  $W_3$  is less than  $W_4$  is less than  $W_5$ . So, I already have these codeword lengths ordered. So, what is the condition for a prefix-free

code to exist? The Kraft's inequality must be satisfied. So, summation of  $D$  raised to power minus  $W_i$  sum over  $i$  1 to 5 must be less than equal to 1. So, you can see here. So, this summation in this example comes out to be  $49/81$ , which is less than 1. Hence, a prefix-free code exists.

Now, how do we construct a prefix-free code? So, maximum codeword length is 4. So, we can construct a  $D$ -ary full tree of length four and then we start pruning it. So, we start pruning it for  $W$  equal to 1. So, all the branches which were starting from this node have been pruned. And, this is a codeword corresponding to the  $W$  1.

Similarly, we have two code words of length 2. So, again we, this is length 1, this is length 2, this is length 1, this is length 2. So, the branches starting from these nodes have all been pruned. So, these are my two code words. And, similarly you can see this is 1, 2, 3. And then beyond this point, the leaves have been pruned and I have one codeword of length four, which is basically 1, 2, 3 and 4. And, to, the once which I have shown by empty circles, these are unused leaves. Now, since this is a ternary code; so that, you can see there are three branches leaving from each node. And, I am just labeling them by 0, 1, and 2. So, this is basically my prefix-free mapping of my prefix-free code, a ternary prefix-free code, corresponding to these  $W$  is.

So, with this I will conclude this lecture.

Thank you.