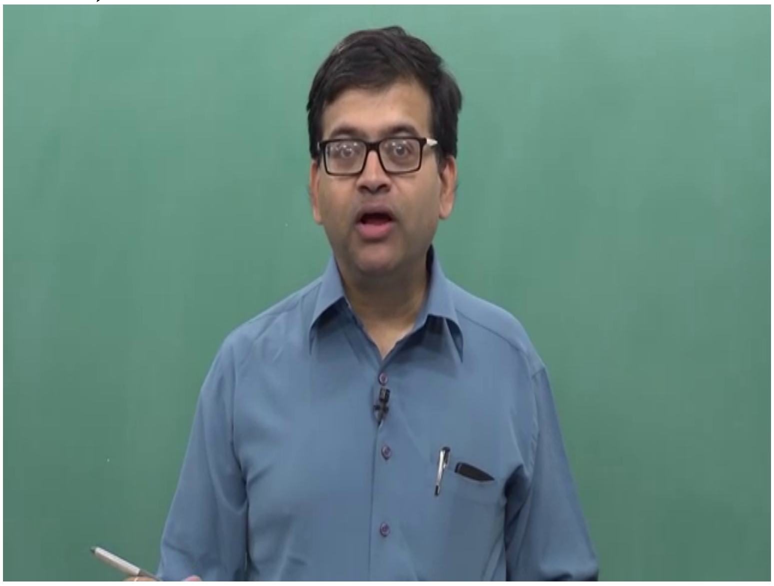
An Introduction to Coding Theory
Professor Adrish Banerji
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Module 04
Lecture Number 18
Convolutional Codes: Distance Properties

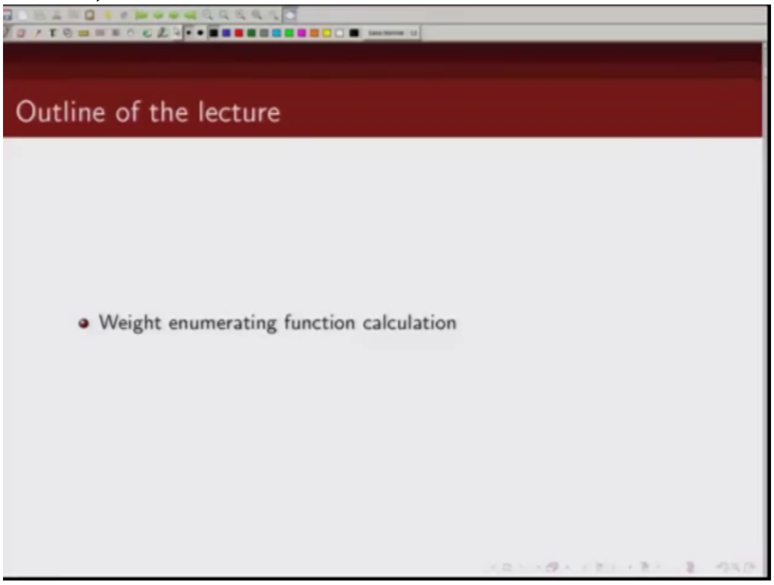
So in this lecture we are going to talk about how we are going to find out the weight

(Refer Slide Time 00:19)



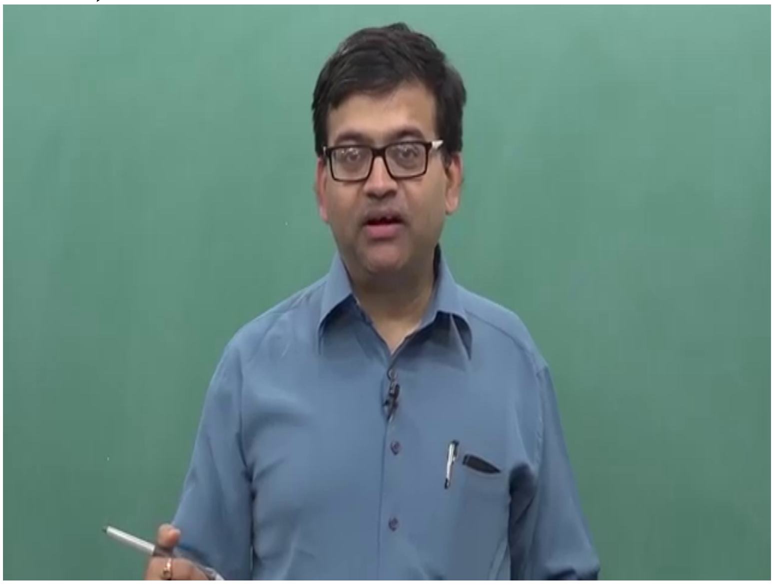
distribution of the convolutional codes and we are going to discuss about the distance properties of convolutional codes. So this

(Refer Slide Time 00:26)



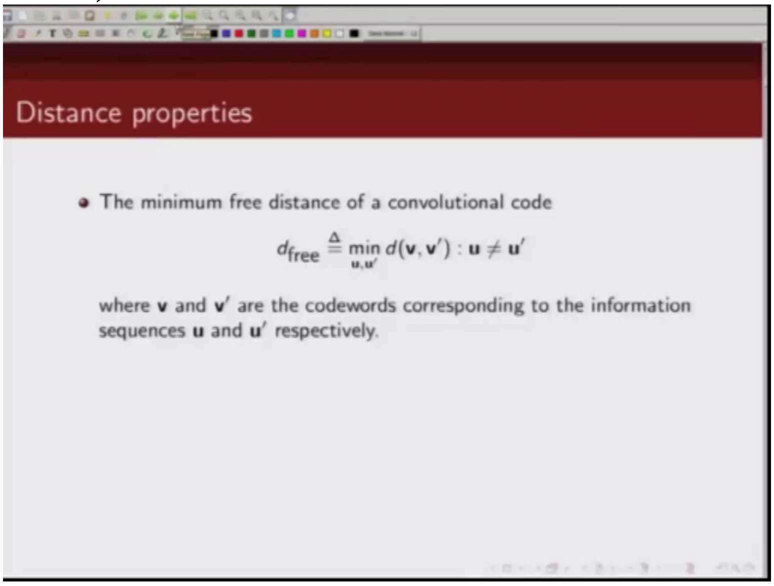
lecture deals with how we can enumerate the distance

(Refer Slide Time 00:30)



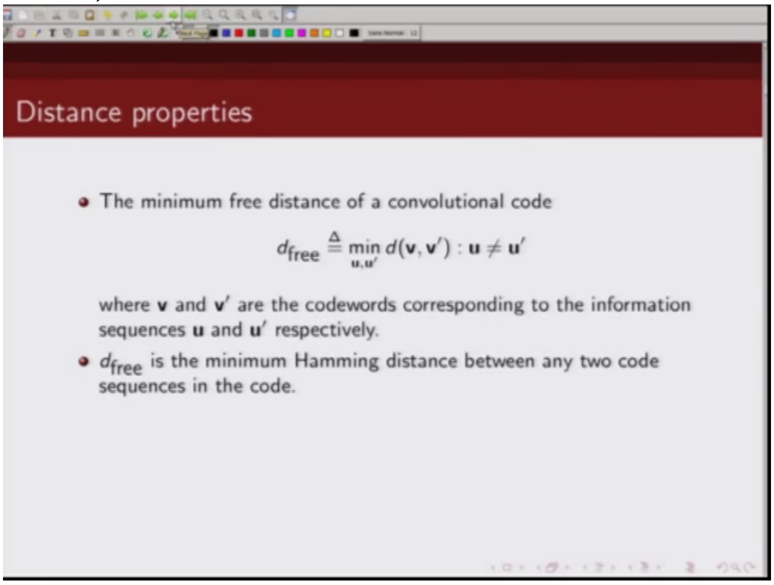
profile or the weight distribution of the convolutional code.

(Refer Slide Time 00:34)



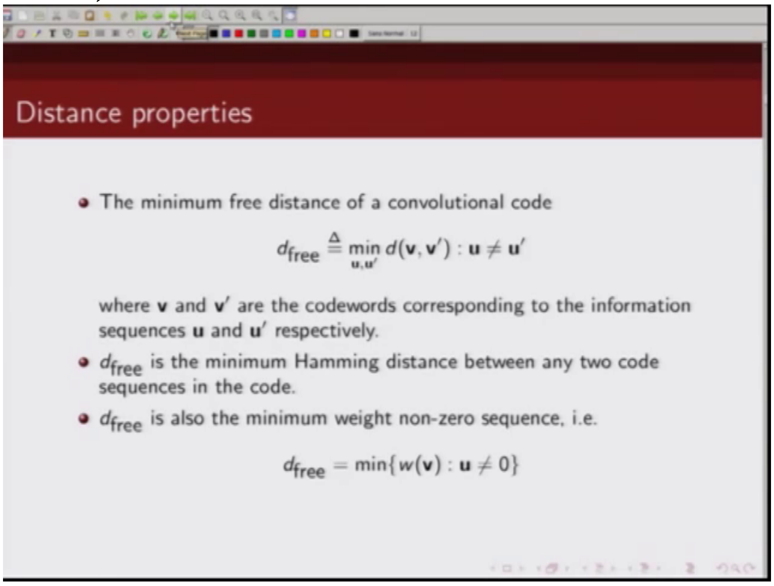
So we can define before we discuss the technique to find out the weight distribution let us define what do we mean by minimum free distance of a convolutional code. So minimum free distance of a convolutional code is defined as minimum Hamming distance between 2 codes v and v hat where v and v hat are 2 codewords corresponding to information sequence u and u hat where u and u hat are 2 different information sequences.

(Refer Slide Time 01:09)



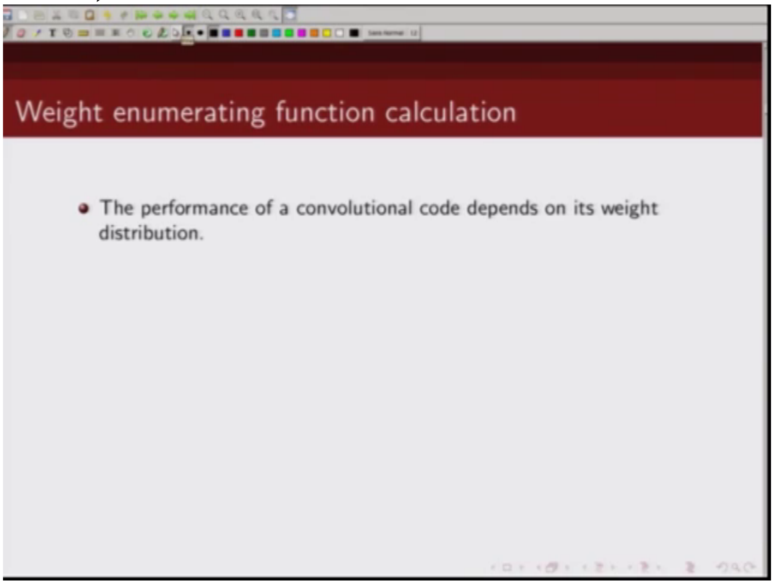
So it is basically, free distance is the minimum Hamming distance between any two codewords in the convolutional code.

(Refer Slide Time 01:19)



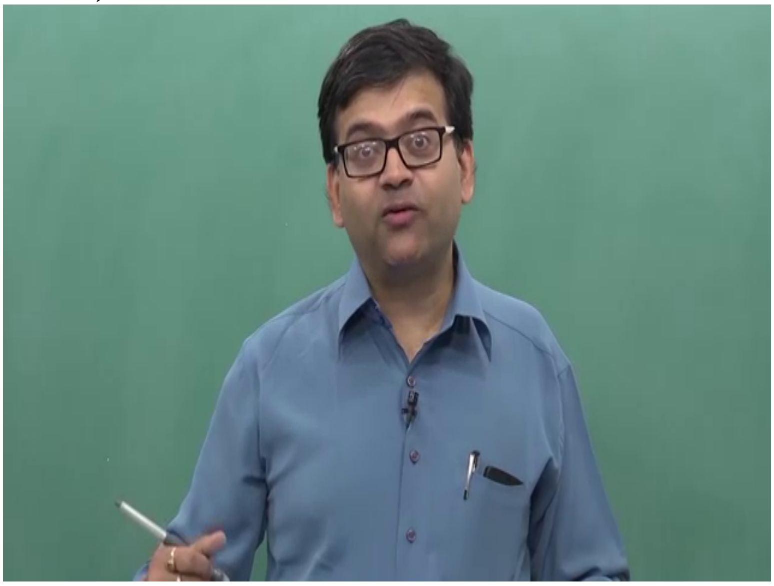
And this is same as minimum weight non-zero so it is the minimum weight of a non-zero codeword. We know

(Refer Slide Time 01:31)



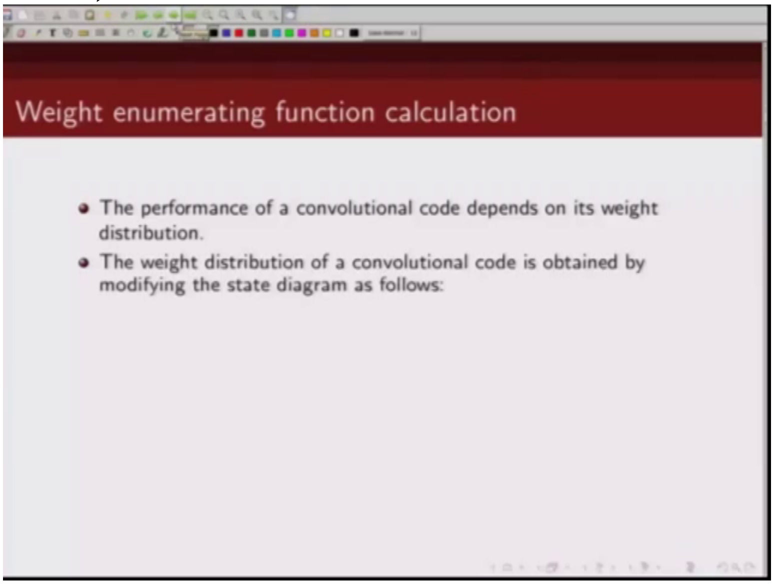
that performance of any convolutional code depends on its weight distribution and that is why we are interested to find out what is the weight distribution

(Refer Slide Time 01:41)



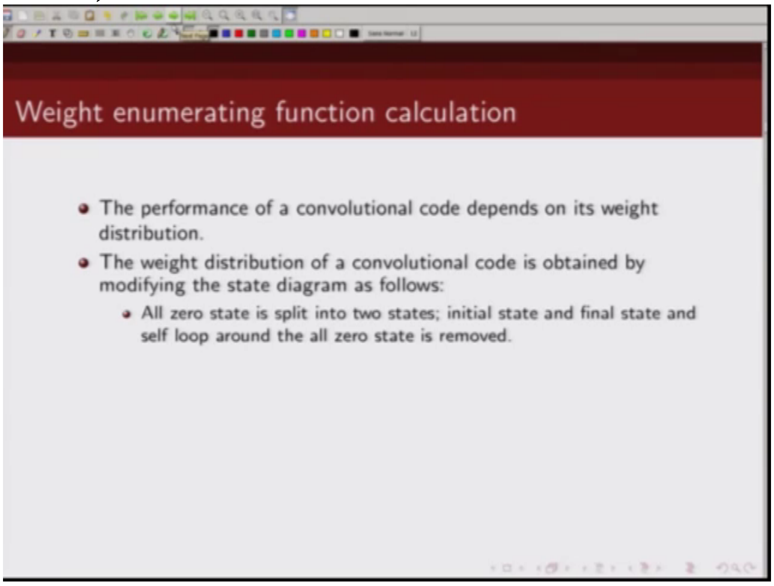
of the convolutional code. In this lecture we are going to talk about a method based on Mason's gain formula to compute the weight distribution for convolutional code.

(Refer Slide Time 01:53)



So in this we are first going to modify the state diagram of a convolutional code. And how are we going to modify the state diagram? We are going

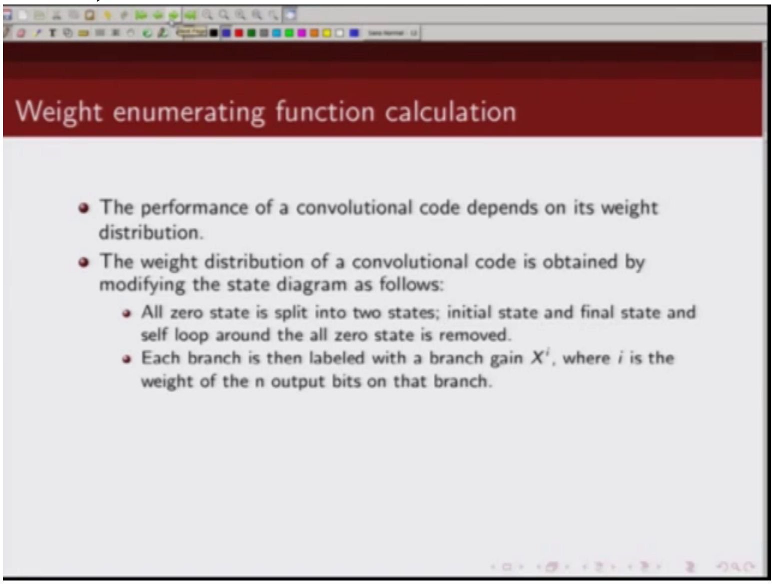
(Refer Slide Time 02:05)



to split the all zero state into 2 states, one initial state and second final state and we are going to remove the self loop around the all zero state.

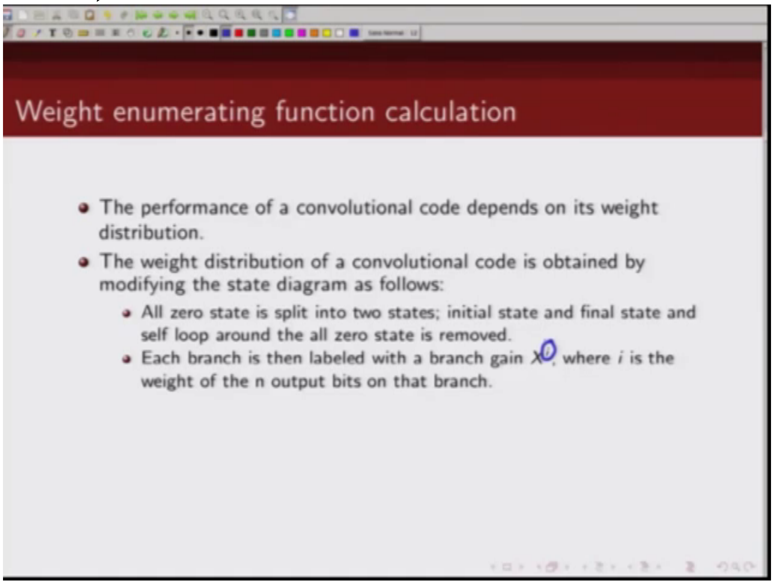
Next,

(Refer Slide Time 02:30)



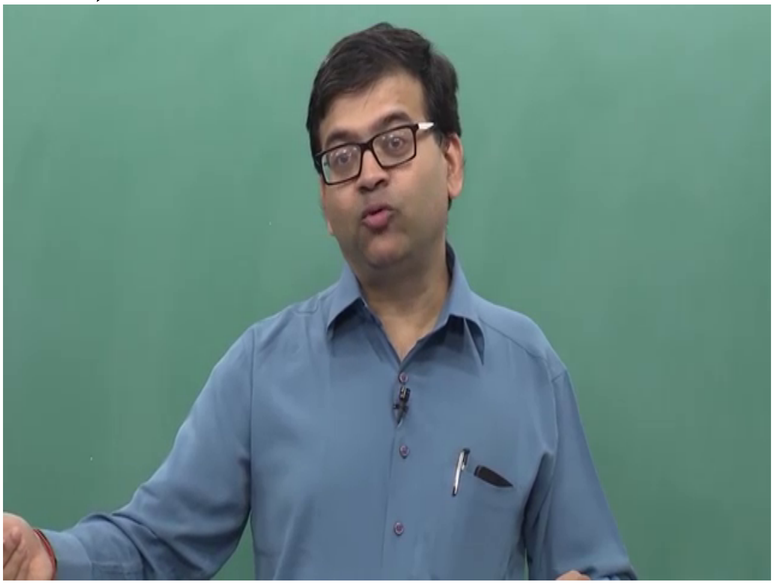
each branch which is linking from one state to another state, this branch will be labeled by the output weight of the codewords. So we are going to denote by x raised to power i where i will be

(Refer Slide Time 02:51)



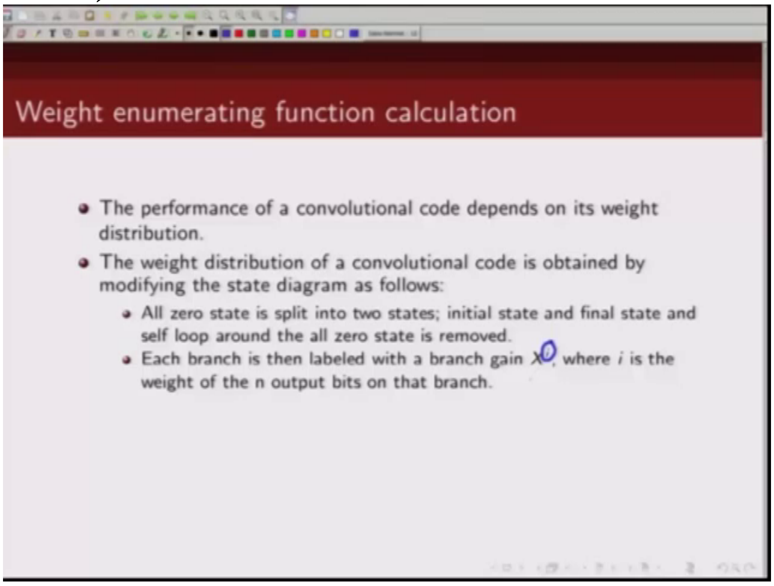
the weight of the coded bits. So for example if we make a transition from state 0 to state 0 1 when input 1 comes and output is 1 1,

(Refer Slide Time 03:06)



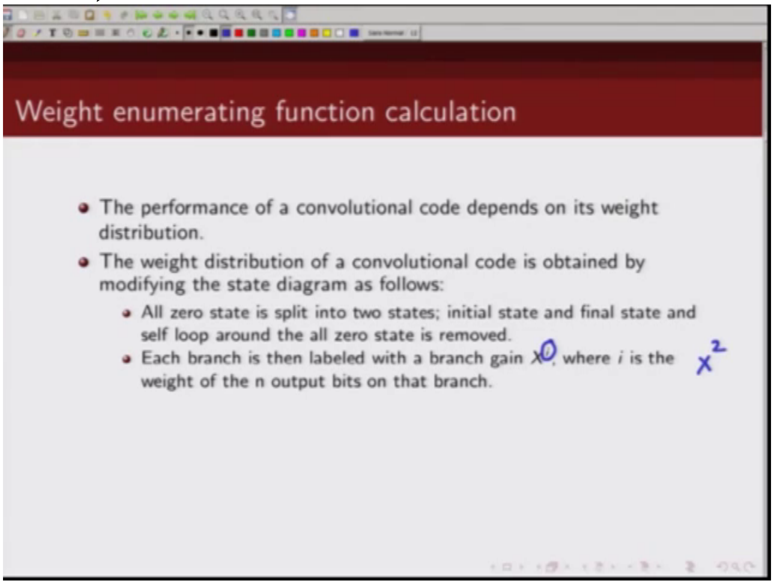
in that case

(Refer Slide Time 03:08)

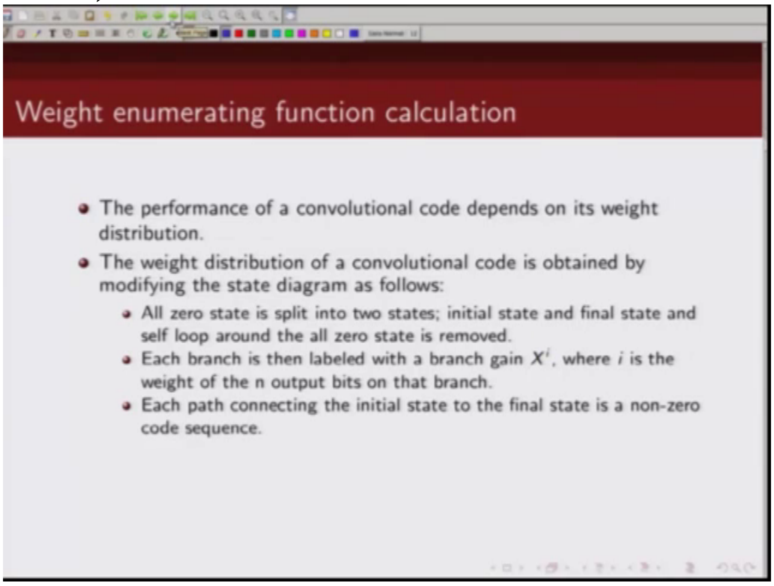


this output is 1 1, we will label that branch by x raised to the power 2.

(Refer Slide Time 03:16)

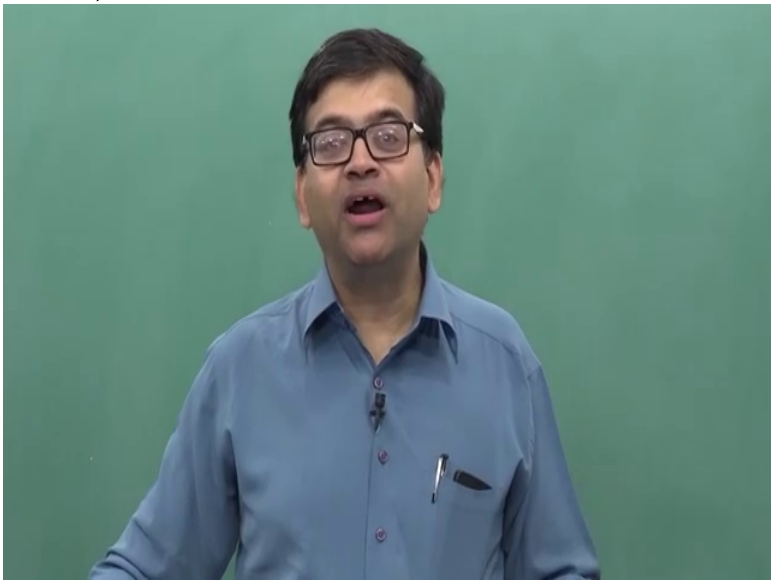


(Refer Slide Time 03:18)



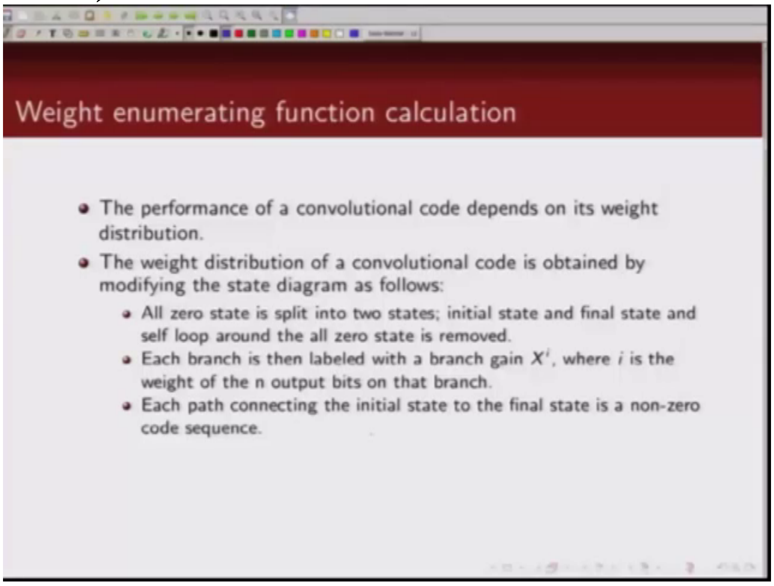
So now after we split this all zero sequence, all zero state into 2 states initial state and final state what we will have is each path starting from this initial state to the final state, that will be

(Refer Slide Time 03:36)



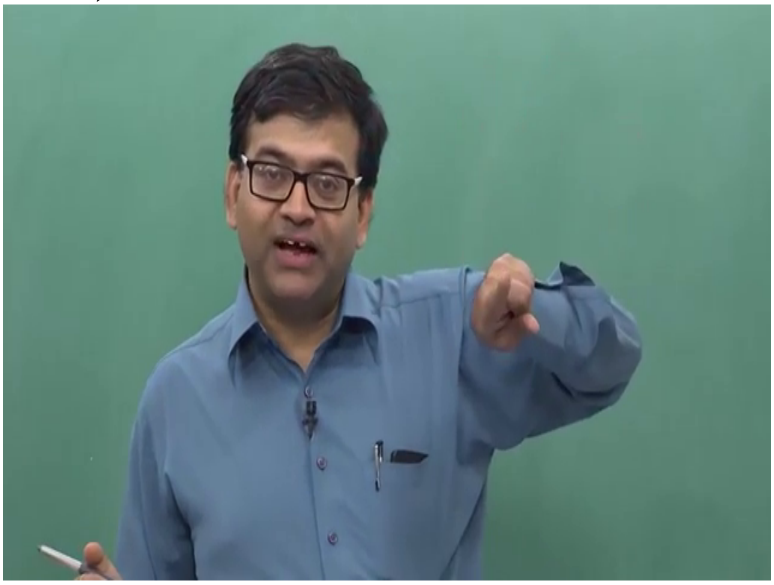
our valid codeword. So each path connecting the initial state

(Refer Slide Time 03:41)



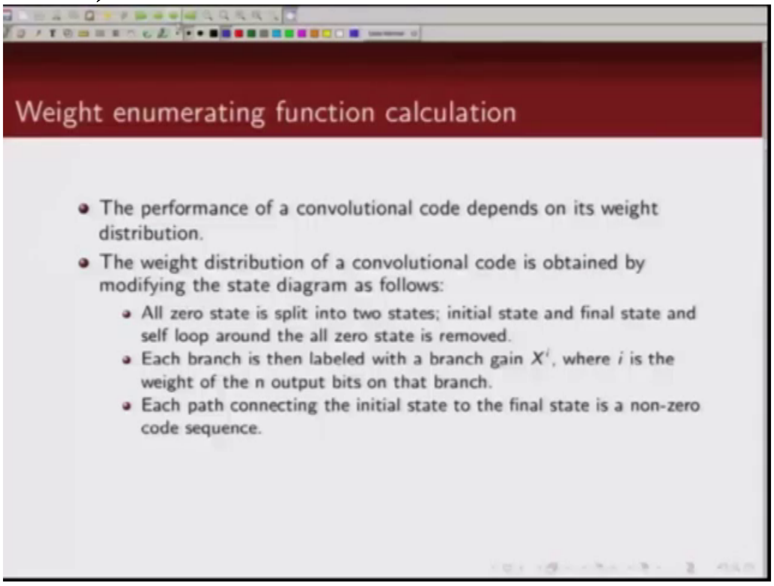
to the final state is a valid non zero codeword because we have removed the self loop

(Refer Slide Time 03:49)



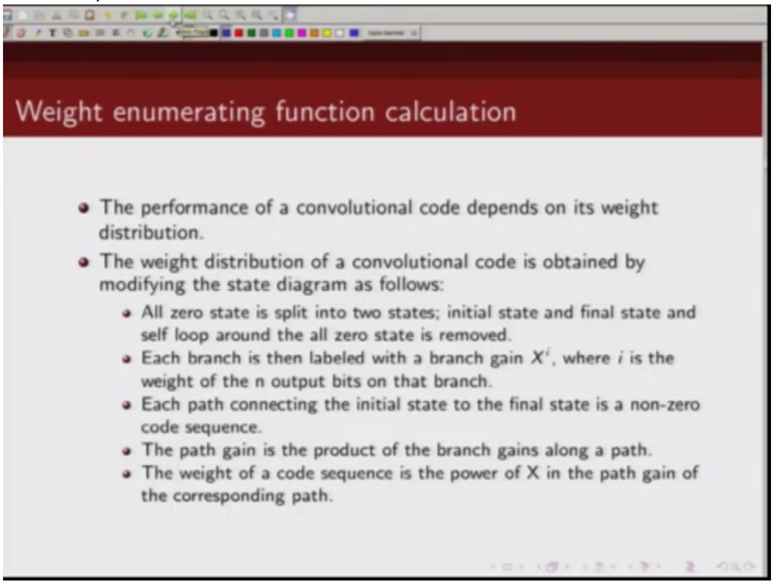
around the all zero state. So

(Refer Slide Time 03:54)



this modified state diagram is now going to show us all possible non zero codewords. We define a path gain as product of branch gains along a path

(Refer Slide Time 04:15)

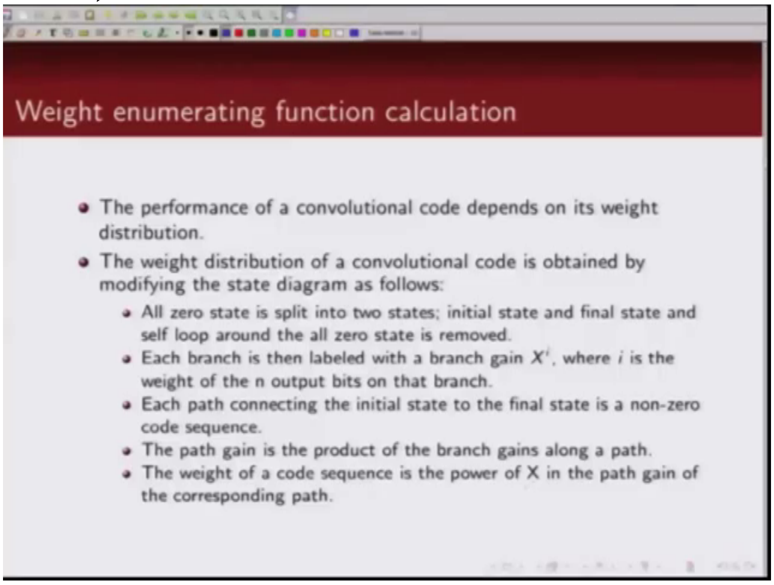


and the weight of the code sequence will be nothing but the power of x in the path gain of the corresponding path because what we are doing is, so each branch is labeled by its corresponding output

(Refer Slide Time 04:34)

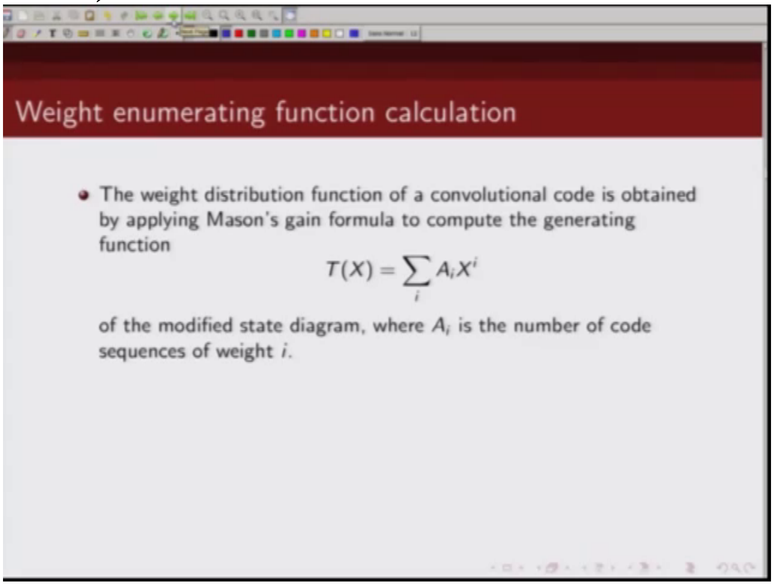
weight. So if we look at each path going from initial state to the final state and we look at the power of x, that will give us

(Refer Slide Time 04:48)



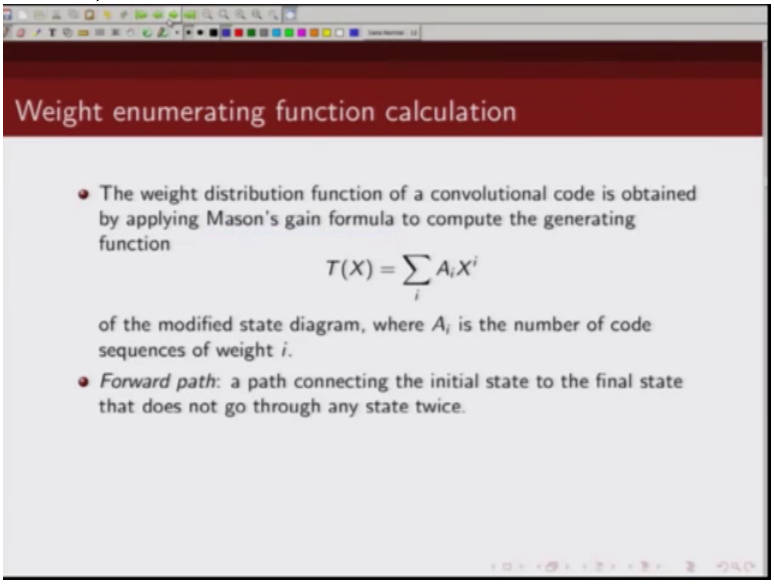
the overall weight of that particular non zero sequence. As we

(Refer Slide Time 04:56)



said we are going to use Mason's gain formula to compute the weight enumerating function for the convolutional code. So we are going to describe how we are going to use the Mason's gain formula. So we are representing by T of X the generating function which will basically enumerate all codewords of weight i.

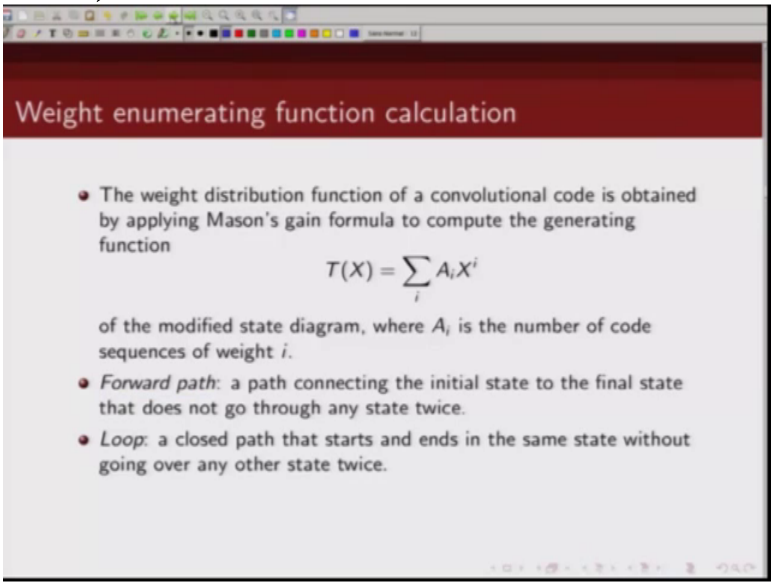
(Refer Slide Time 05:32)



Now let us define few terms that we are going to use in Mason's gain formula.

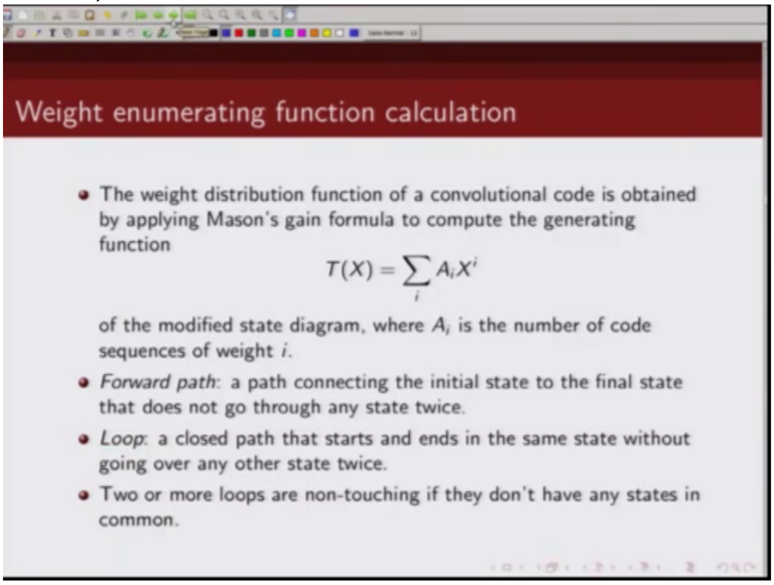
The first term that we are going to define is what is known as forward path. So a forward path is a path from the initial all zero state to the final state, and the condition is this path should not go over any state twice. That is our forward path.

(Refer Slide Time 06:05)



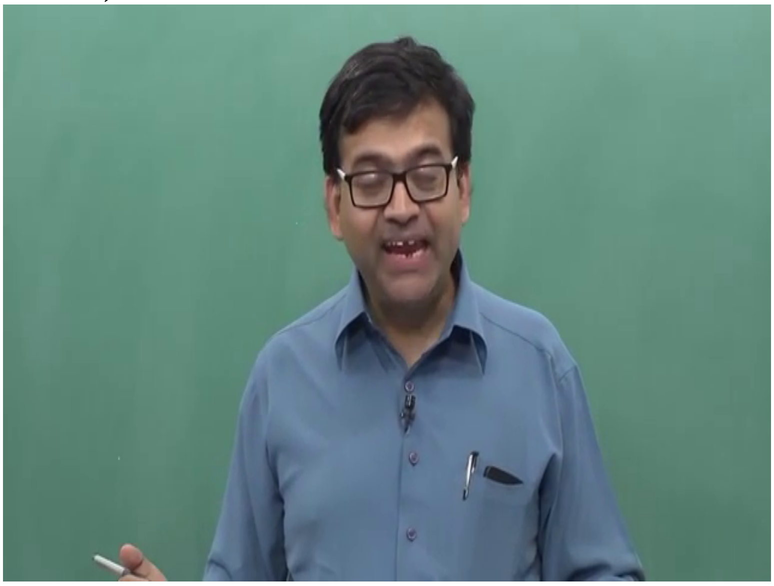
Next term that we define is basically a loop. What is a loop? A loop is a closed path that starts and ends in the same state without going over any state twice. That's a loop.

(Refer Slide Time 06:26)



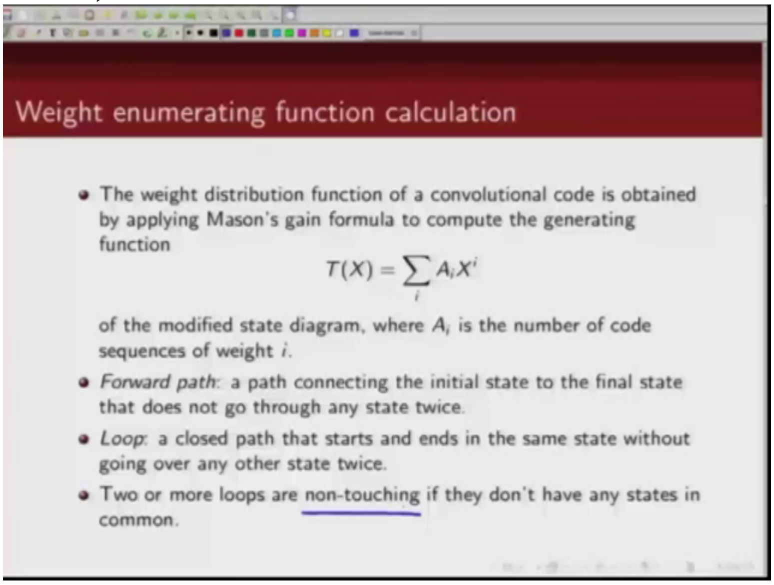
When do we say 2 loops are non-touching? We say two loops are non touching if they do not have any state in common.

(Refer Slide Time 06:37)



So again I repeat these 3 definitions. Forward path is a path from initial

(Refer Slide Time 06:43)

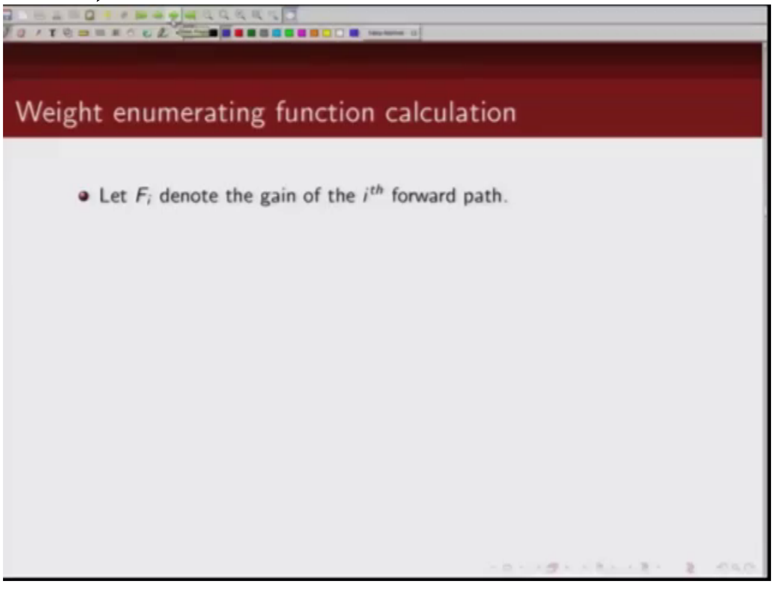


state to the final state without visiting any state twice. A loop is a closed path starting and ending in the same state without going over the same

(Refer Slide Time 06:56)

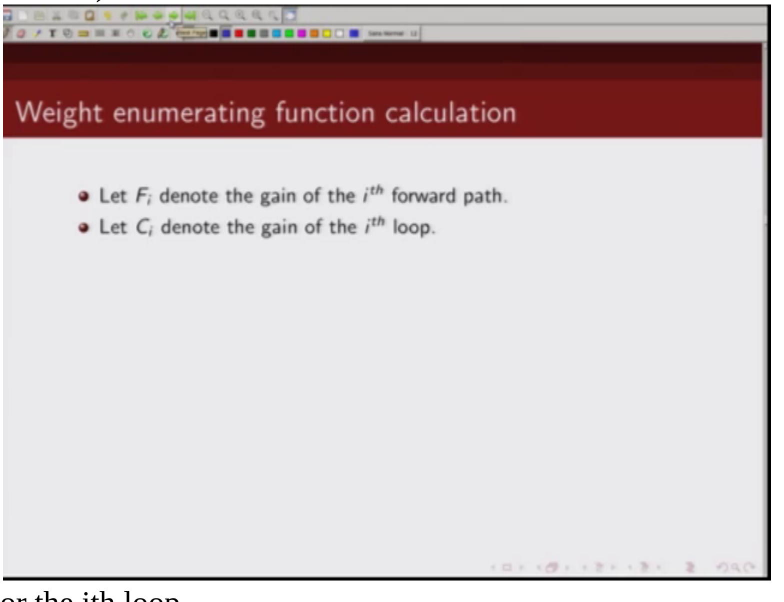
state twice and two or more loops are non touching if they do not have any state in common.

(Refer Slide Time 07:08)



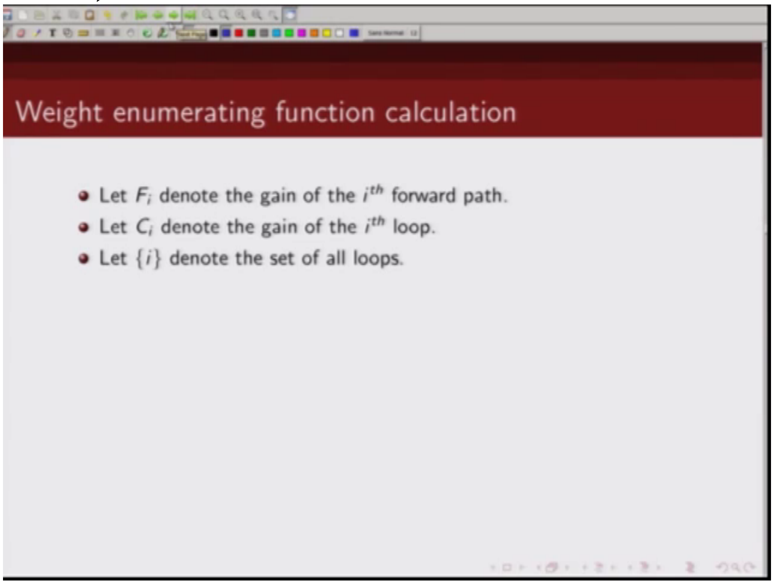
Now let us denote by F sub i, the gain of the ith forward path. And let C i

(Refer Slide Time 07:17)



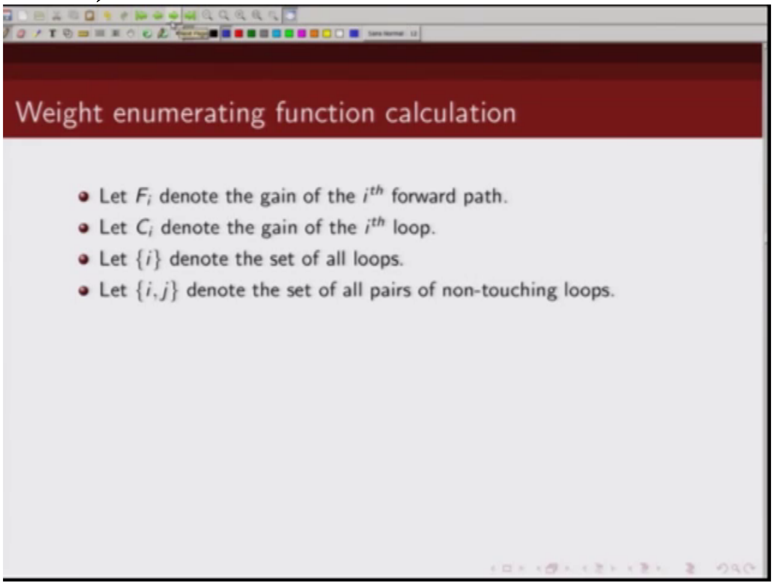
denote the gain for the ith loop.

(Refer Slide Time 07:23)



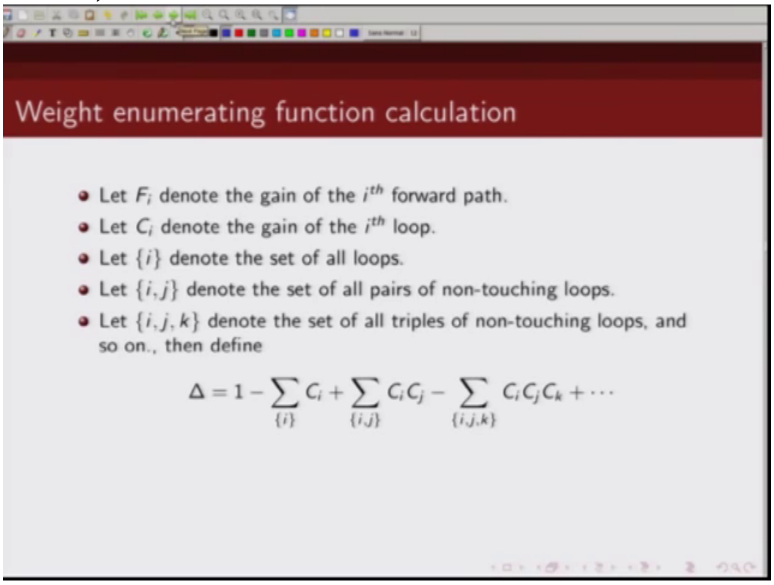
We denote by this the set of all loops. Similarly

(Refer Slide Time 07:30)



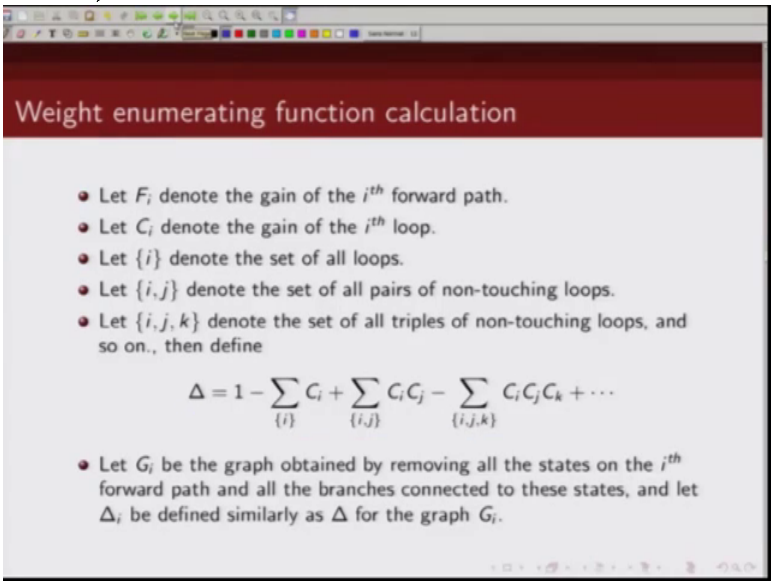
this set of i and j will denote set of all pairs of non touching loops.

(Refer Slide Time 07:40)



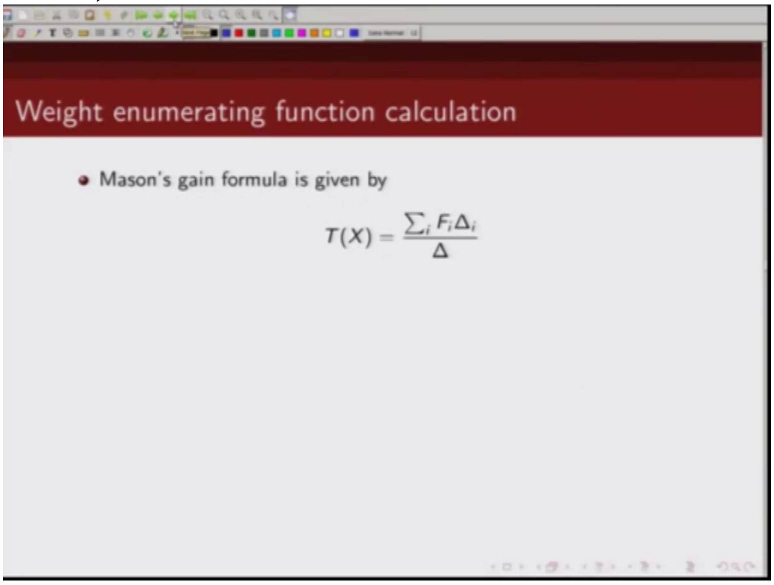
This triplet will define set of all triplets of non touching loops. So if we use this, we define a term delta which is defined as follows. It is 1 minus summation of all the gains of the loops plus product of gains of all those non touching loops minus this is product of, set of all triplets of non touching loops and it goes on like this. So that's our delta.

(Refer Slide Time 08:21)



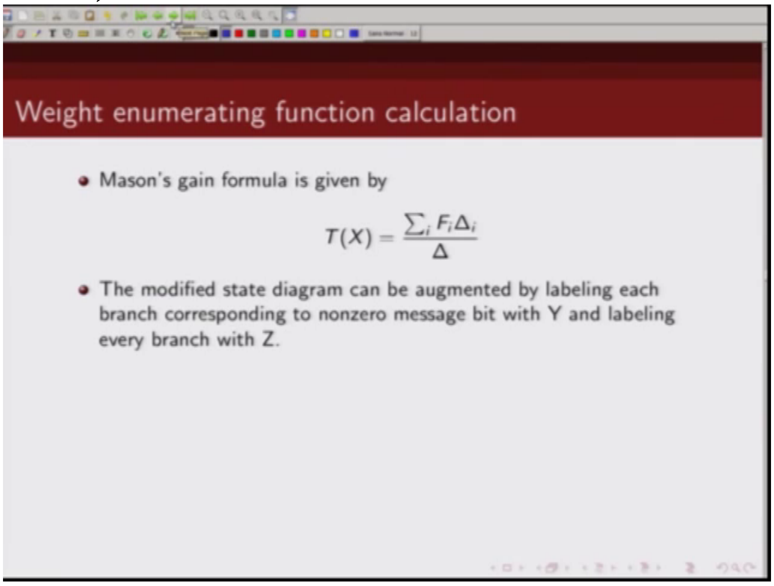
We define our graph that is obtained after we remove all states belonging to ith forward path by G sub i. So G sub i is basically the graph remaining after we remove the ith forward path and the delta corresponding to this modified graph will be denoted by delta i.

(Refer Slide Time 08:53)



So the Mason's gain formula then says that the generator function for convolutional encoder can then be given by this expression. So this

(Refer Slide Time 09:07)

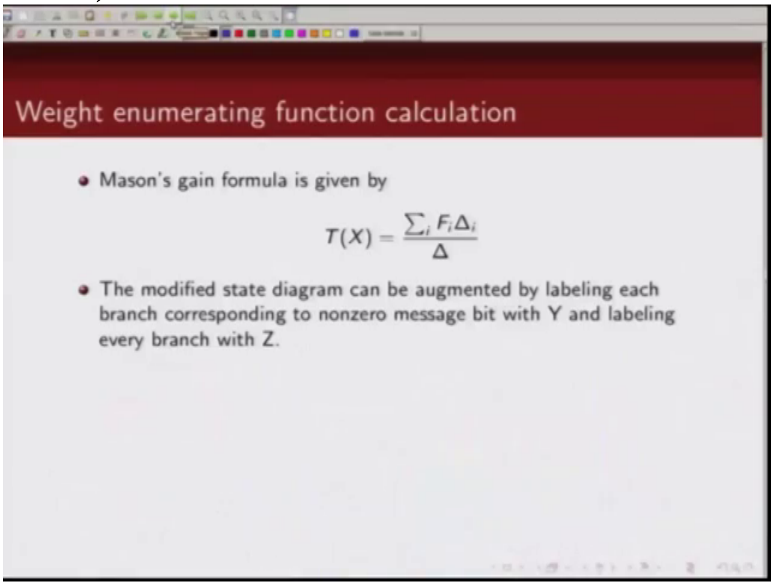


modified state diagram can be augmented to include more information. Now what we had done so far was we labeled the branches

(Refer Slide Time 09:18)

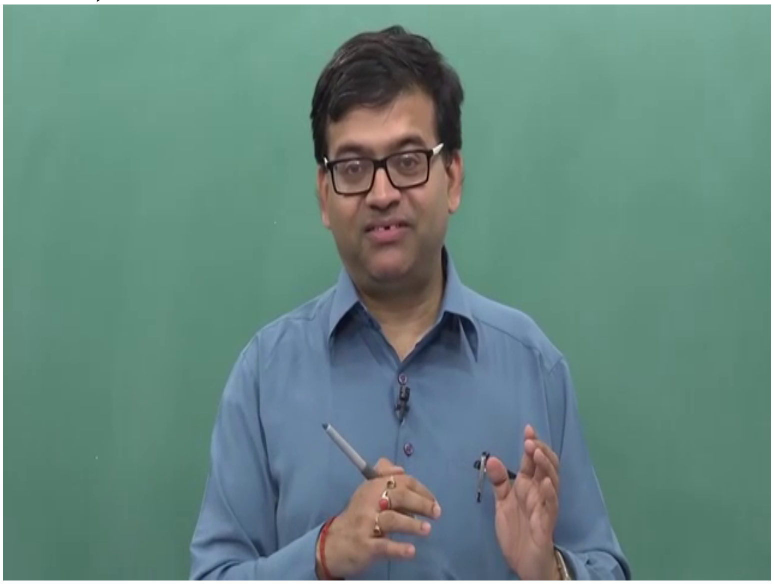
of the state diagram by the overall total weight. Now we can also augment this by mentioning

(Refer Slide Time 09:27)



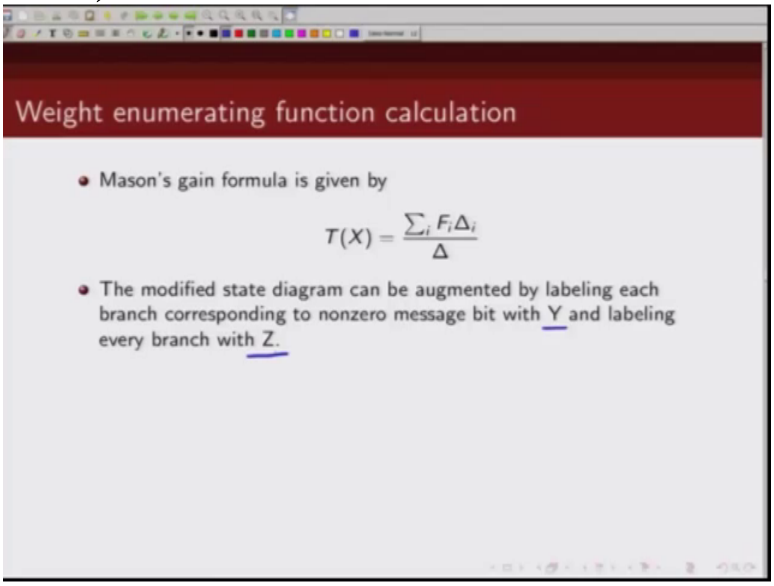
what is the input that results in that output weight. So we can label the weight of the input by y. So the power of y will denote what is the input weight and similarly we can label each branch by z. So in a path gain formula the degree of z will tell us

(Refer Slide Time 09:56)



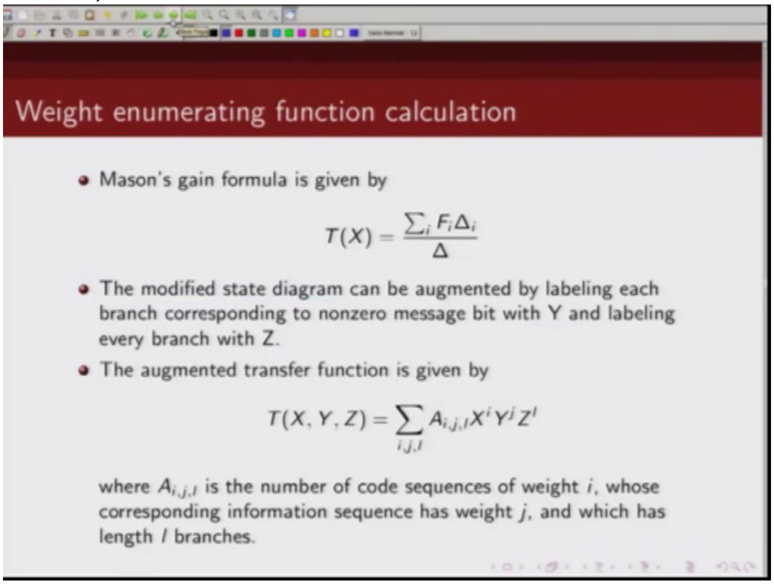
like what's the

(Refer Slide Time 09:59)



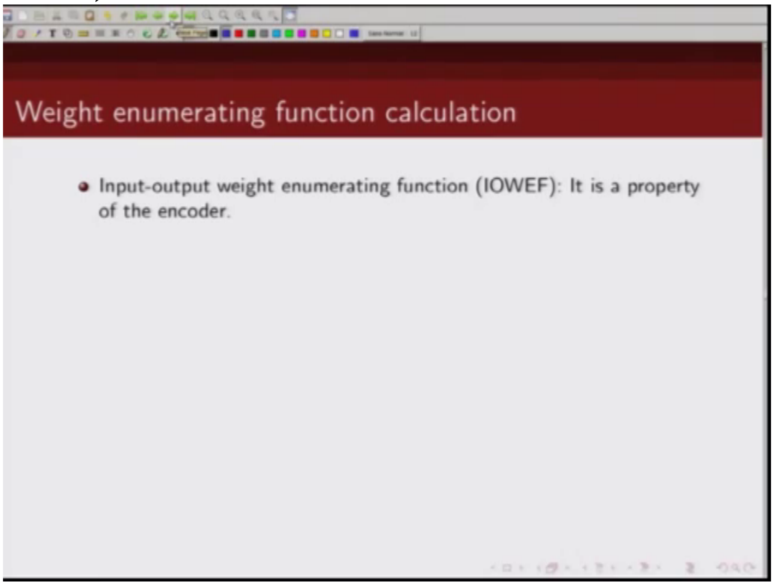
length of non zero path. So the degree of z will tell us like once it diverge from all zero state after how much time it comes back into all zero state. So we can augment our state diagram by what I call a modified state diagram by adding two additional information, one is the weight of the message bit which will be denoted by power of Y i and other is branch which will be denoted by Z.

(Refer Slide Time 10:40)



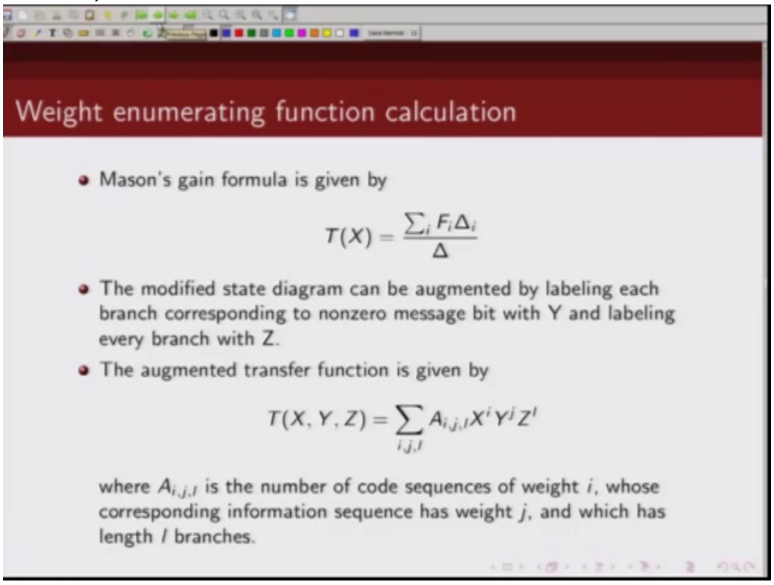
So we can then similarly define an augmented transfer function which will not only tell us the codeword weight but it also tells us what is the input weight that results in that output weight and it will also tell us length of that particular codeword. By length I mean the time it diverges from all zero state until it comes back to all zero state. So this

(Refer Slide Time 11:16)



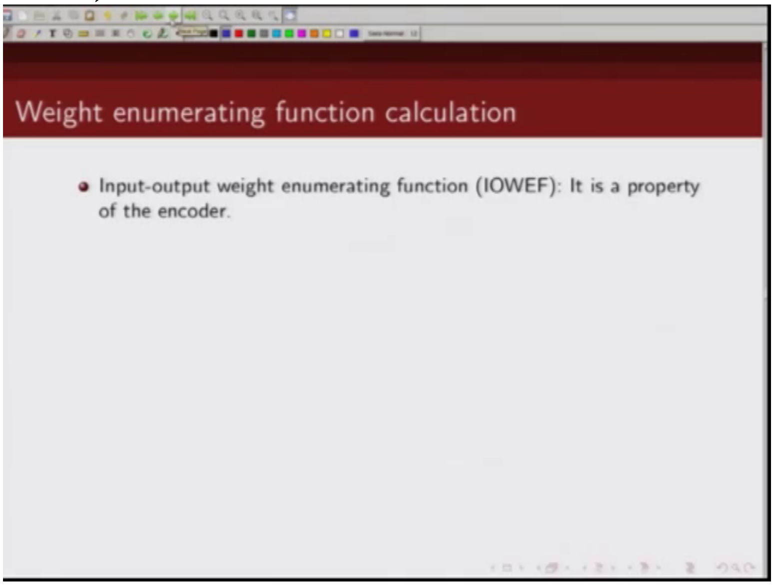
is basically what we call input output weight enumerating function. Because

(Refer Slide Time 11:23)



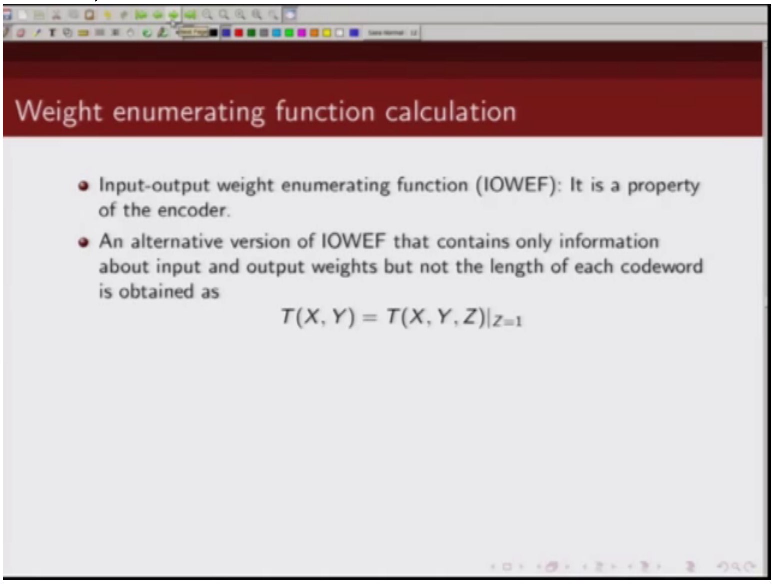
this function is enumerating for what input you get what output, Ok. So this will give us weight input output

(Refer Slide Time 11:34)



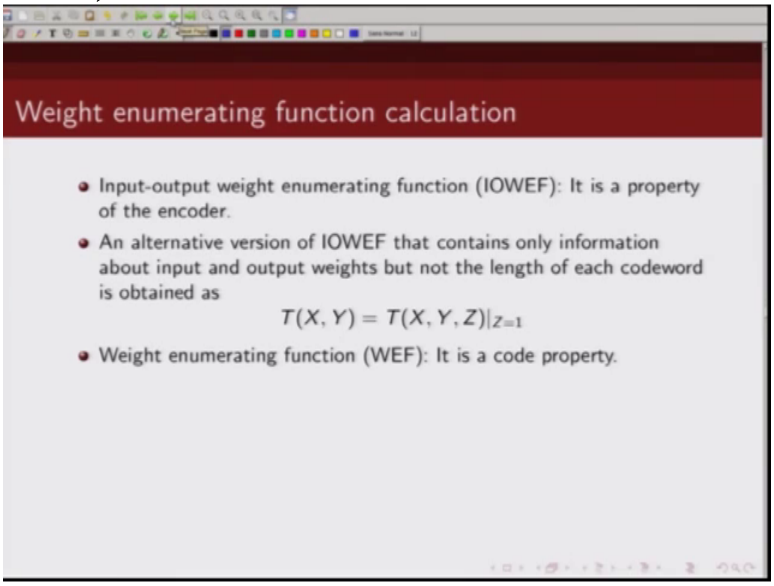
weight enumerating function. And it is a property

(Refer Slide Time 11:38)



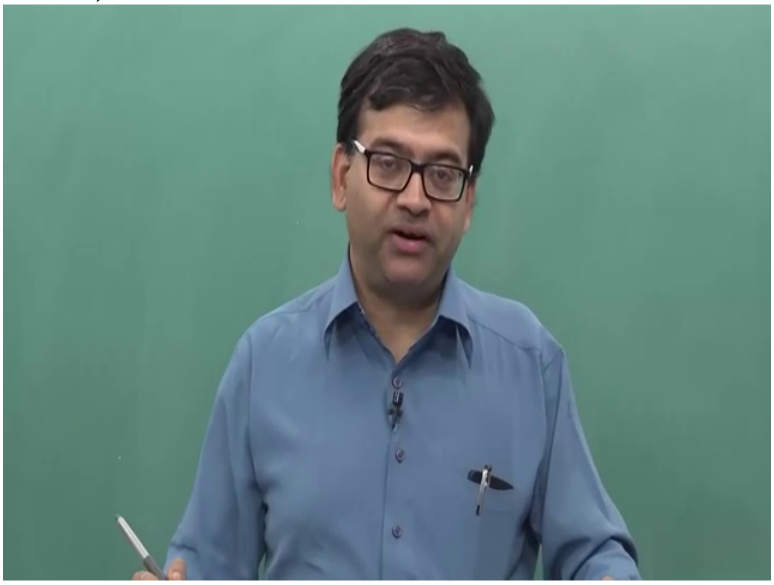
of the encoder. An alternative version of this input output weight enumerator function is one that contains only information about the input and output weight and not the length of each codeword. So if we put Z equal to 1, basically this is going to give us, it is an alternative version of input output weight enumerating function.

(Refer Slide Time 12:03)



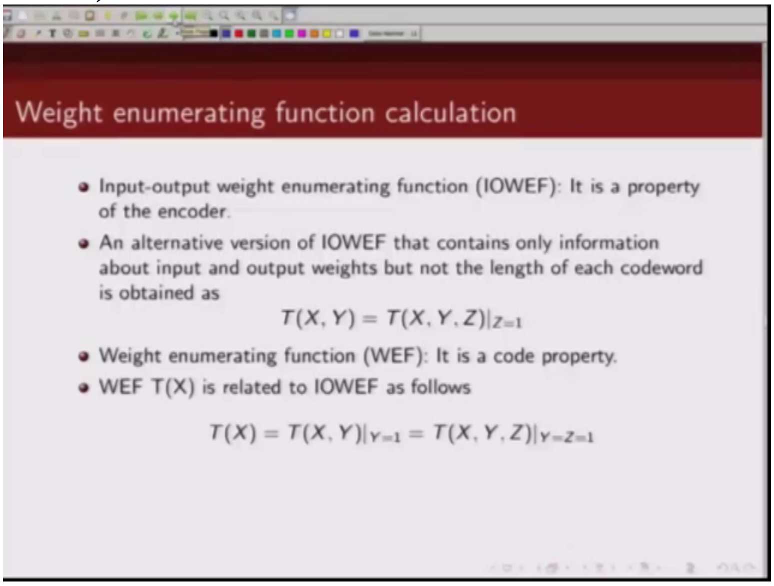
And what is weight enumerating function? Weight enumerating function will only tell us what is the overall codeword weight and this is the property of

(Refer Slide Time 12:13)



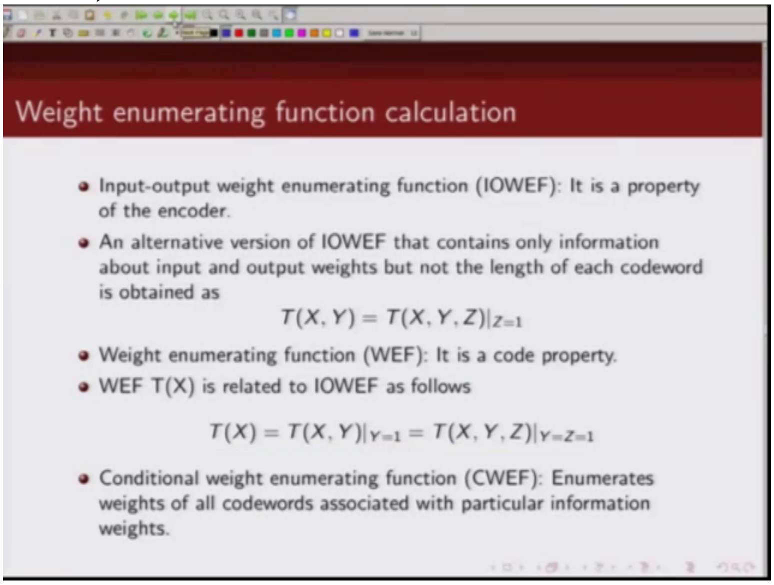
the convolutional code.

(Refer Slide Time 12:16)



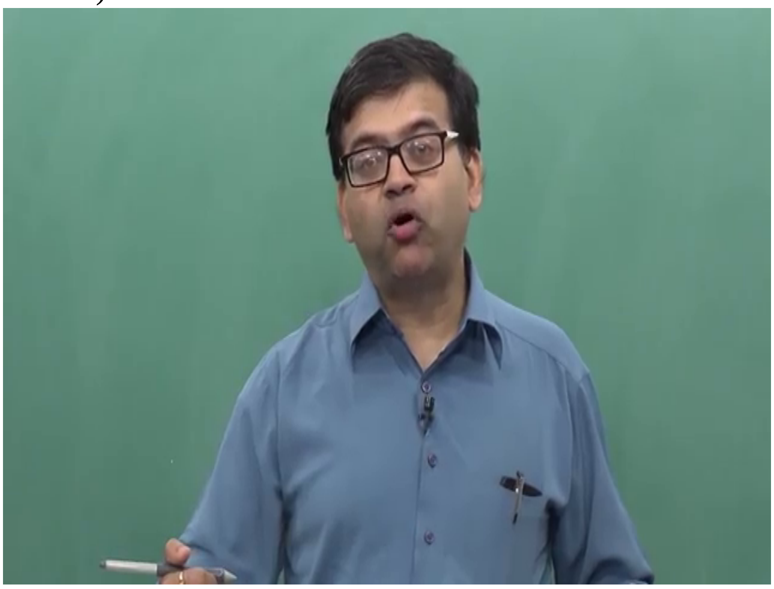
So weight enumerating function is related to input output weight enumerating function in this particular way. So if we put Z and Y as 1 in the input output weight enumerating function, we will get back our weight enumerating function. Similarly

(Refer Slide Time 12:38)



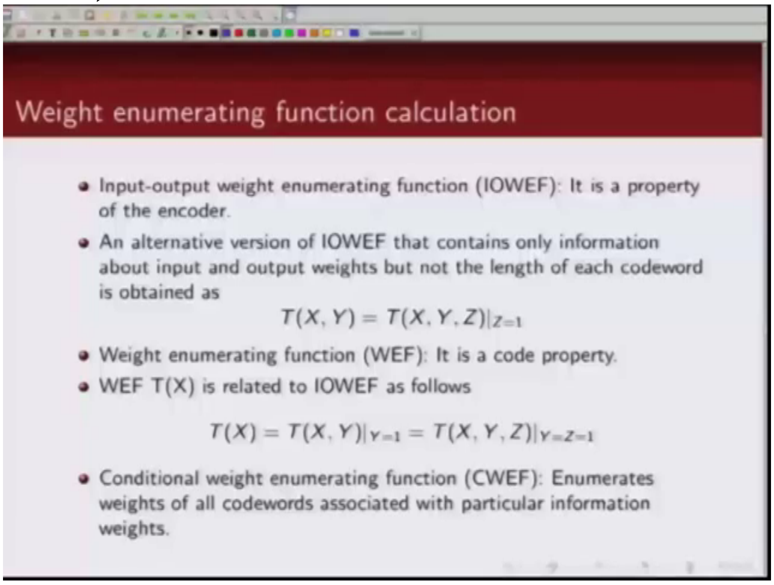
we can define what is known as conditional weight enumerating function. What is conditional weight enumerating function? The conditional weight enumerating function, it enumerates weights of all codewords associated with a particular information weight. So if you are interested in knowing what is the output weight correspond to

(Refer Slide Time 13:03)



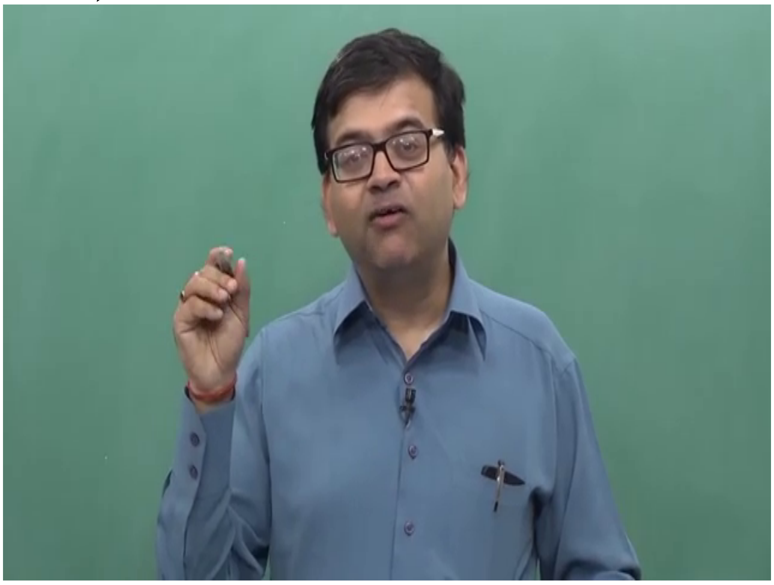
weight 4 input sequence, so from the input output weight enumerating function by collecting all terms which will have

(Refer Slide Time 13:15)



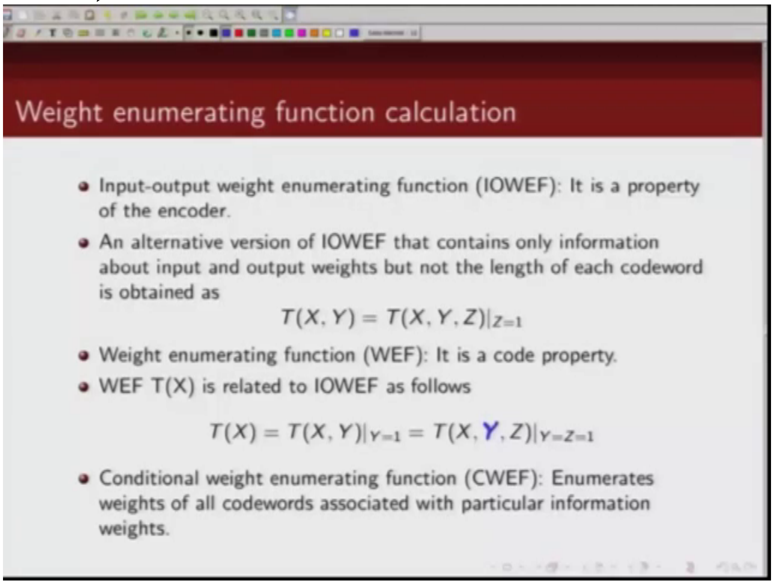
w 4, we can find what is the weight of all codewords corresponding to output weight, input weight of 4. And again we are using Y to denote input weight. So if you are interested in input weight 4, we should look for terms containing Y

(Refer Slide Time 13:41)



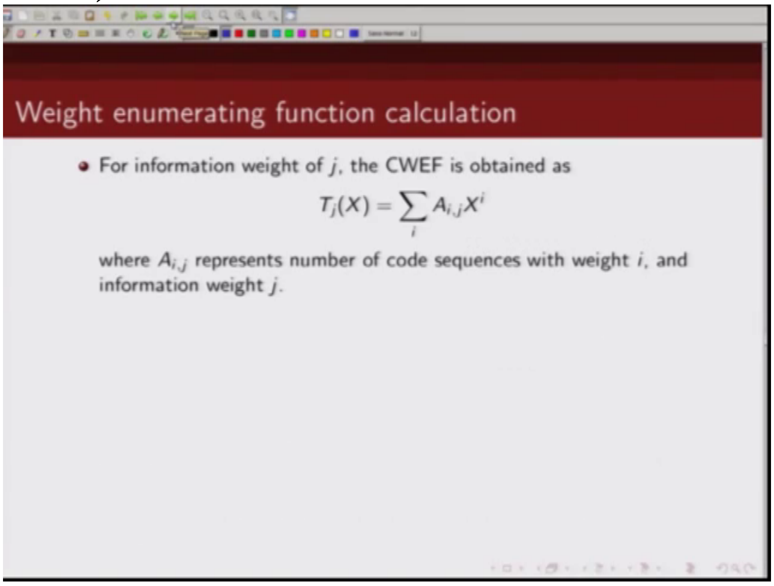
raised to power 4. So this denotes the

(Refer Slide Time 13:47)



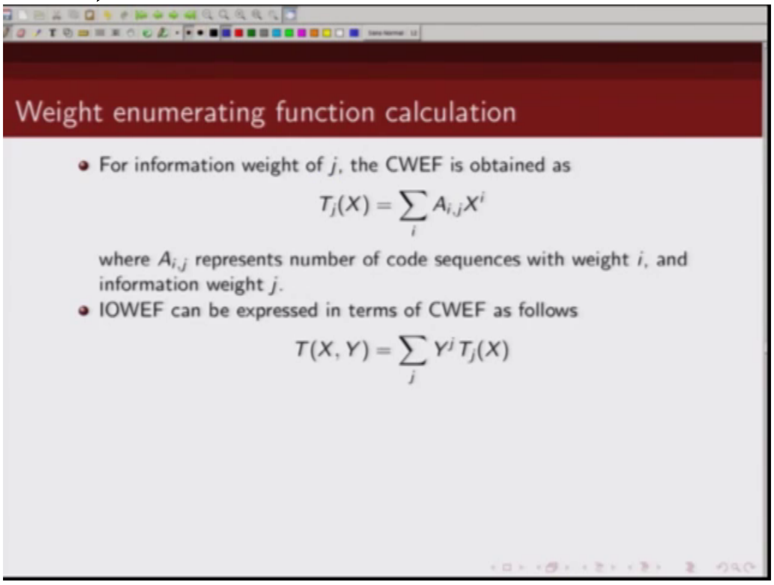
input weight, this denotes the output coded weight and this denotes the length. So as

(Refer Slide Time 13:57)



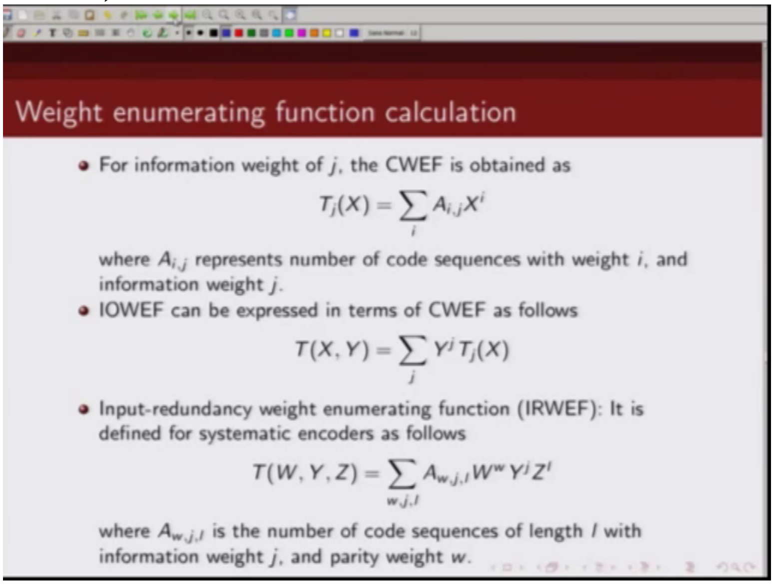
I said, for input weight of j conditional enumerating function will give us what is the output codeweight that you can achieve for a input weight of j. And we can write our

(Refer Slide Time 14:14)



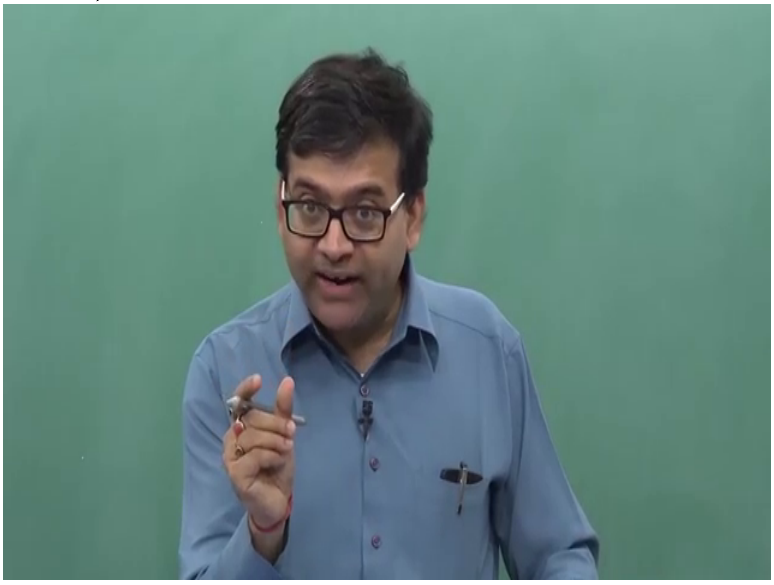
input output weight enumerating function in terms of weight enumerating function so this is basically input of weight j will result in conditional weight enumerating function and we show it for all js that will be our input output weight enumerating function.

(Refer Slide Time 14:41)



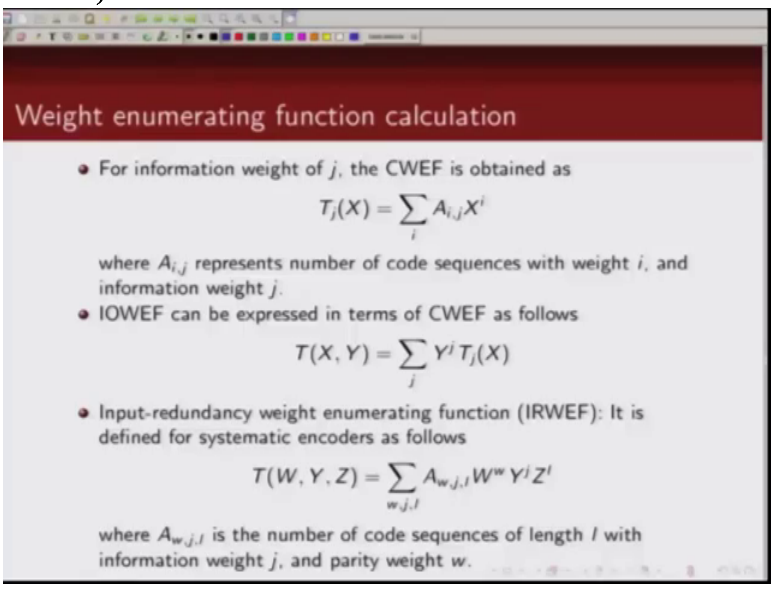
This another property which is defined for systematic encoders which is called input redundancy weight enumerating function. So here because the output weight

(Refer Slide Time 14:56)



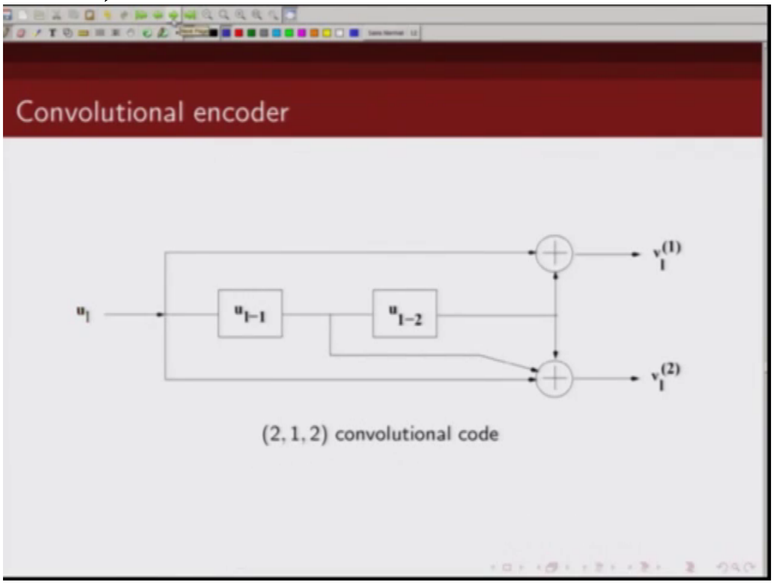
of a systematic encoder consists of weight of the information bits and weight of the parity bits. Now since

(Refer Slide Time 15:05)



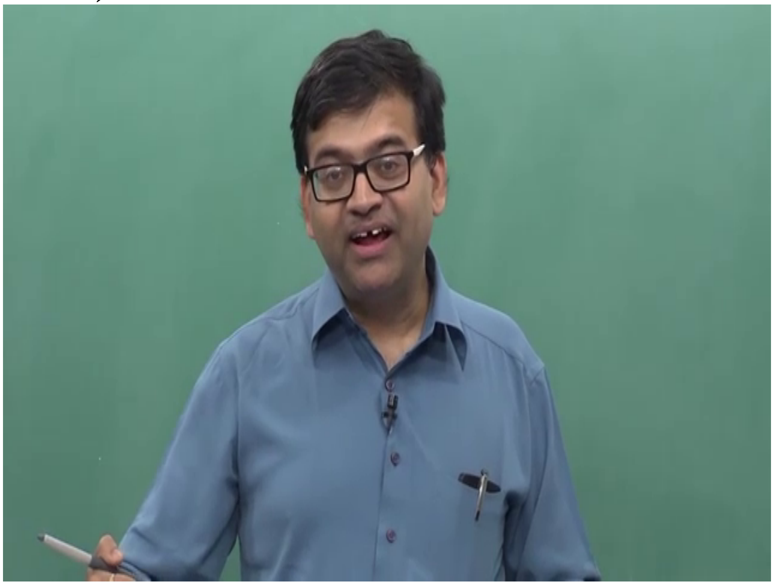
the power of Y already is denoting the weight of the information bits; so when you are asked to show the output weight, instead of saying the output weight you can just specify the weight of the parity bits because it is a systematic encoder, the overall weight will be weight of the parity bits and weight of the information bits. So overall weight would be w plus G. So input redundancy weight enumerating function is defined for systematic encoders. So where instead of specifying the overall coded bit, here you only specify the weight of the parity bits.

(Refer Slide Time 15:53)



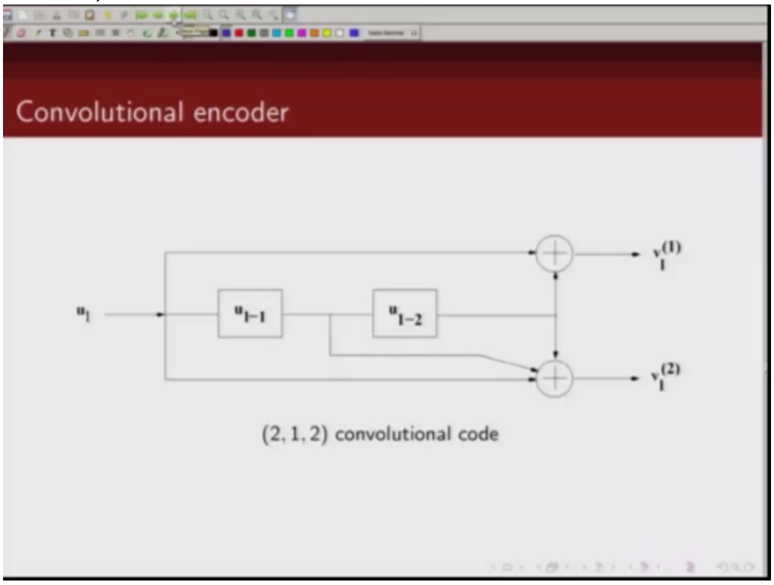
Now let's take an example to illustrate how we can find the

(Refer Slide Time 15:59)



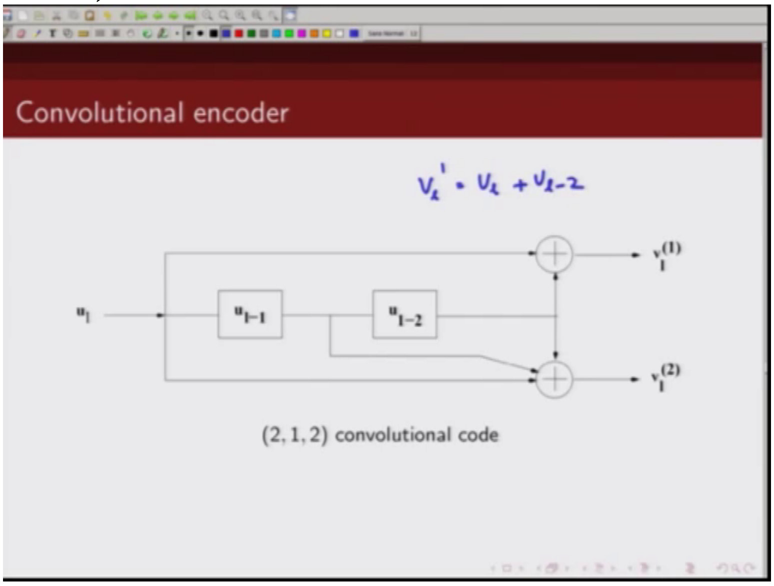
weight enumerating function of a convolutional code. So we are going to consider

(Refer Slide Time 16:05)



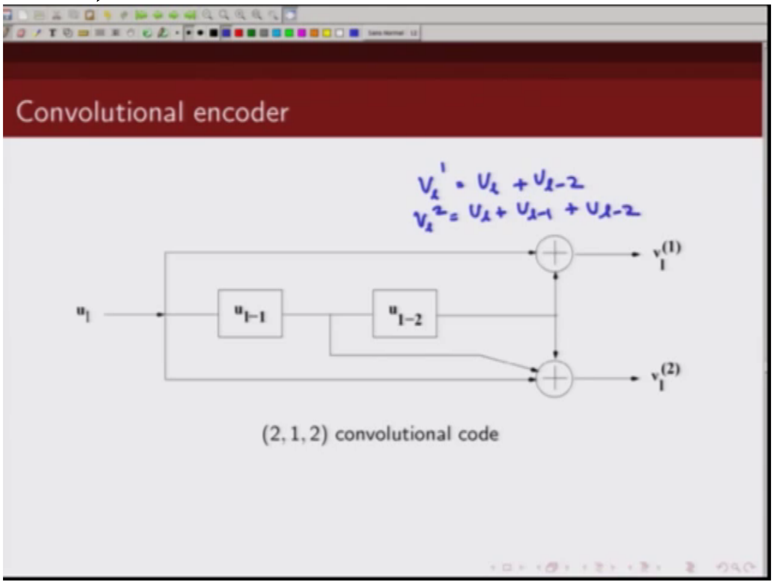
a rate 1 by 2 convolutional code whose memory is 2. So this is the convolutional code. We can see basically v l 1 is u l plus u l minus 2.

(Refer Slide Time 16:24)



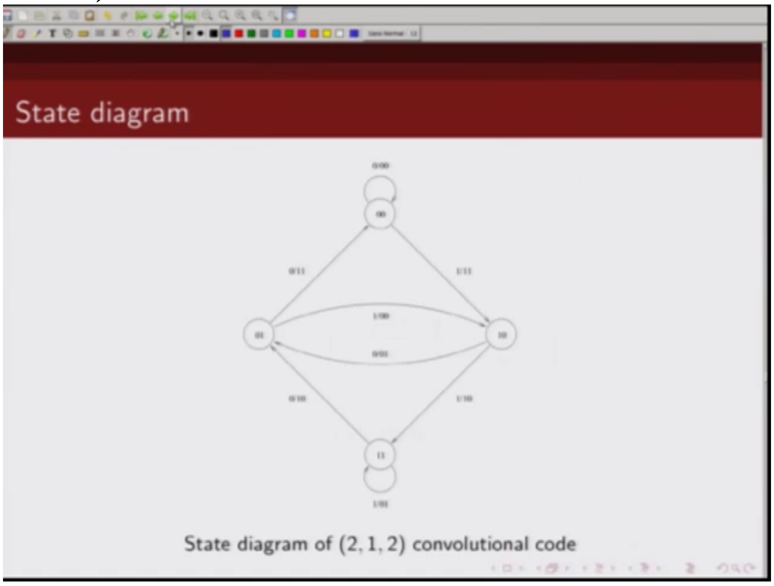
And v l 2 is nothing but u l plus u l minus 1 plus u l minus 2. This is our

(Refer Slide Time 16:36)



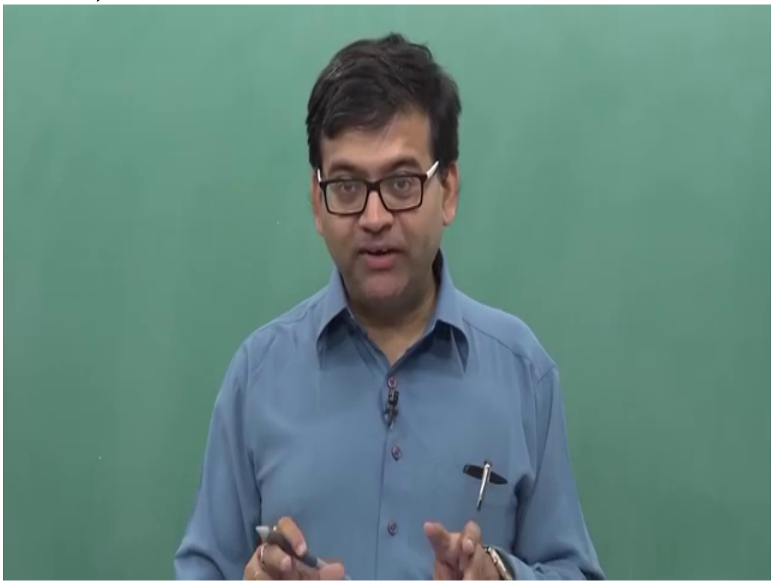
convolutional code. Now the state diagram

(Refer Slide Time 16:40)



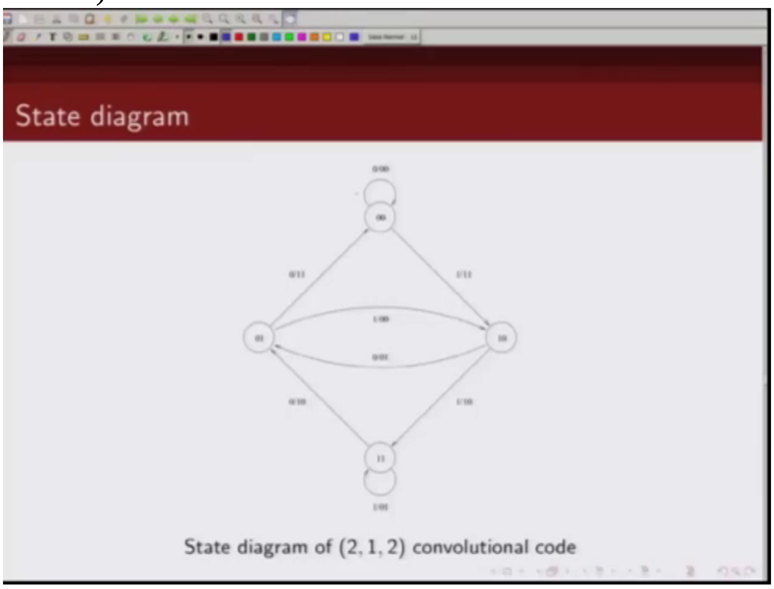
for this convolutional encoder is given by this. Now we are going to modify the state diagram for the purpose of calculating the weight enumerating function.

(Refer Slide Time 16:54)



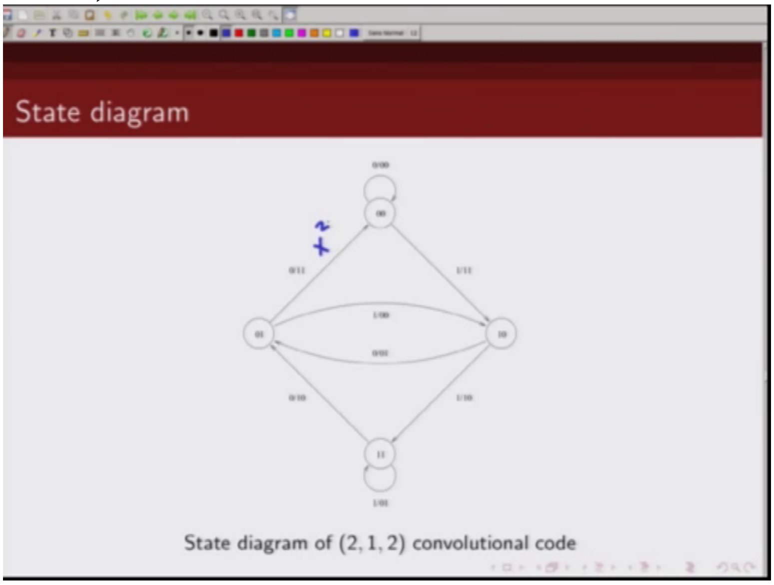
So recall what are the modifications that we have to do? We have to remove

(Refer Slide Time 16:59)



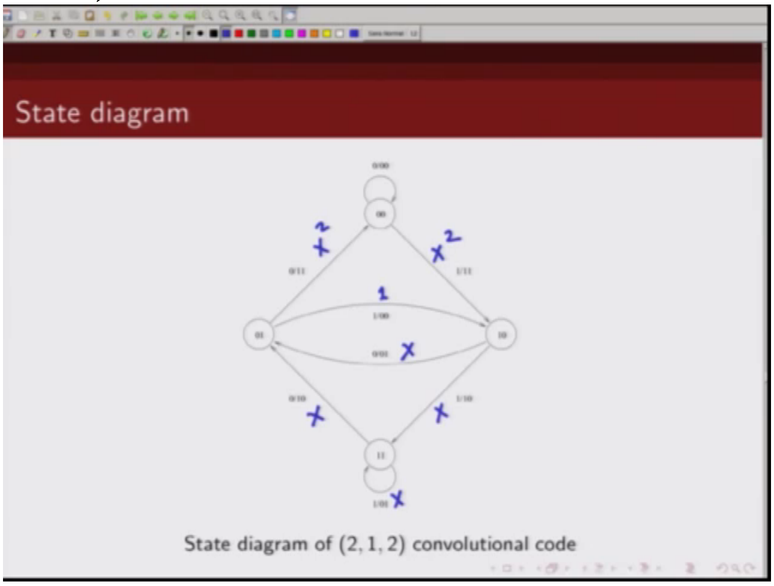
this self loop around all zero state and we have to split this all zero state into two state, initial state and final state. Next we have to label all these branches by weight of the output bit. So this will be x 2, this will be

(Refer Slide Time 17:27)



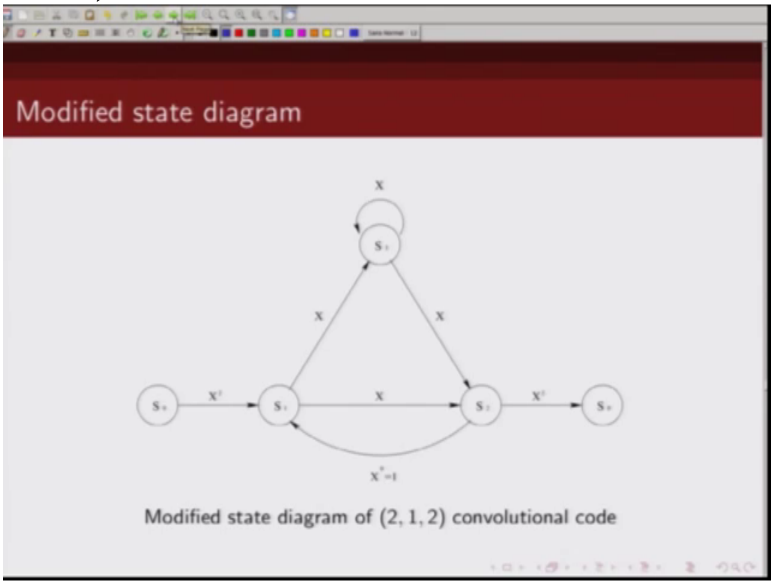
x 0 which is 1, this will be x, this will be x square. This will be x. This will be x. This will be x. If you

(Refer Slide Time 17:40)



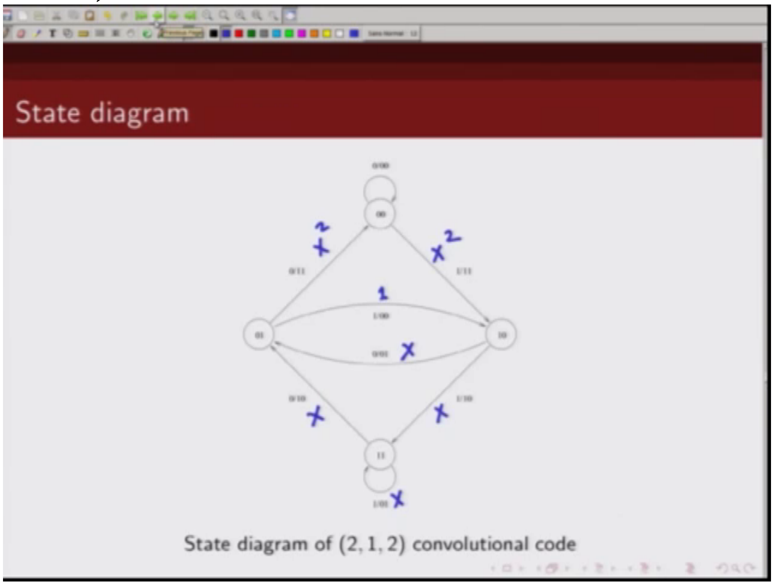
go back, this

(Refer Slide Time 17:42)



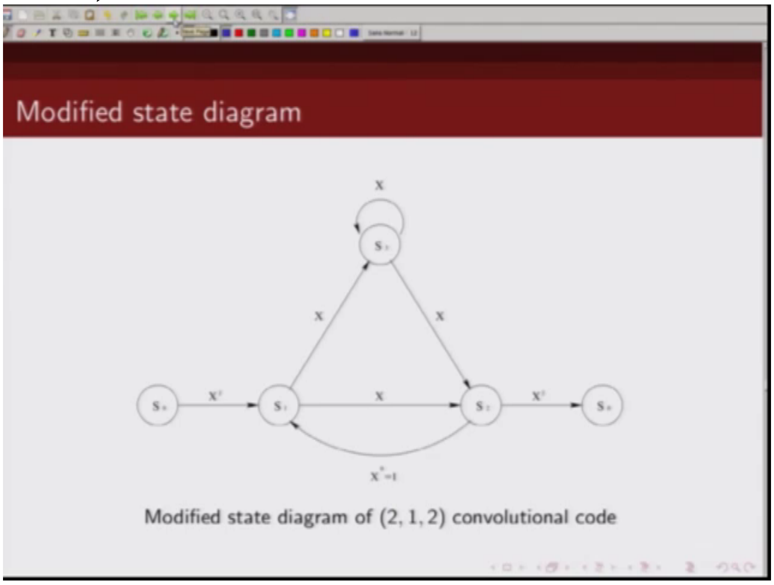
is how my augmented well modified state diagram will look like. So what I did was I have these states

(Refer Slide Time 17:53)



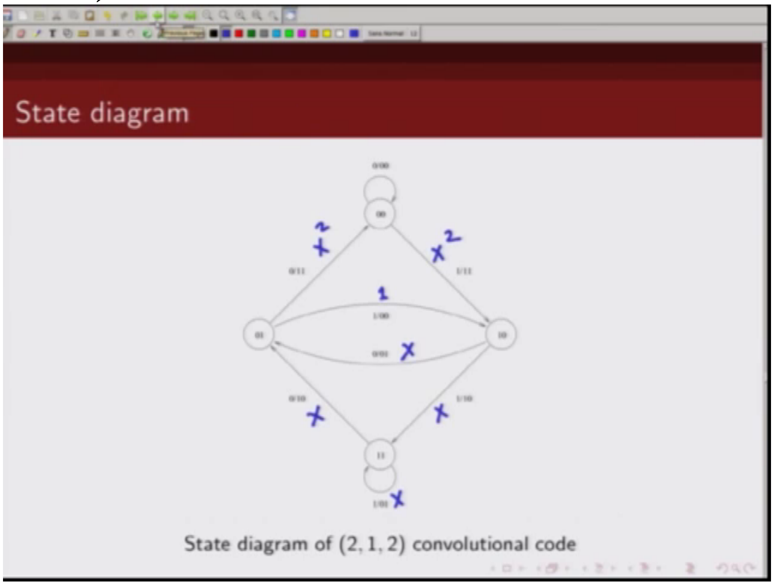
here. I split this all zero state into 2 states. So this state was split into initial

(Refer Slide Time 18:00)



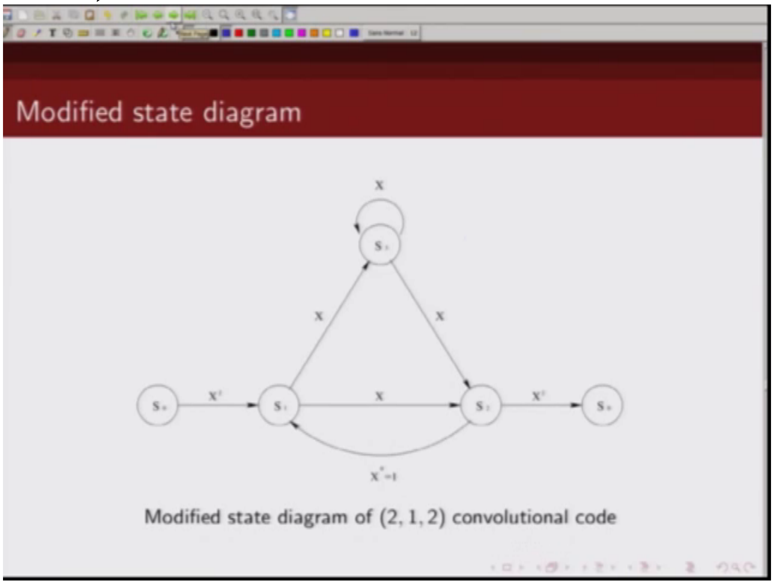
state and final state, Ok. Next what did I do?

(Refer Slide Time 18:08)



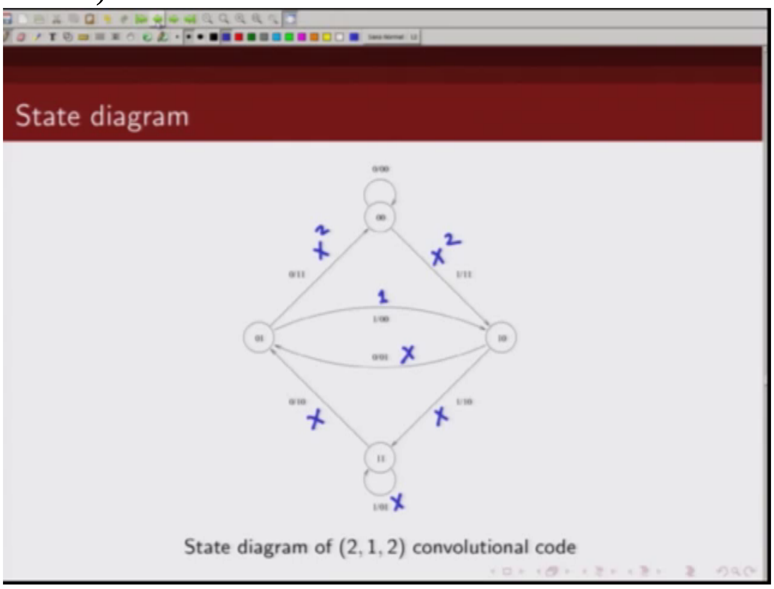
I redrew the same diagram but I labeled each transition by the weight of the output. So you can see from 0 0 I am going

(Refer Slide Time 18:27)



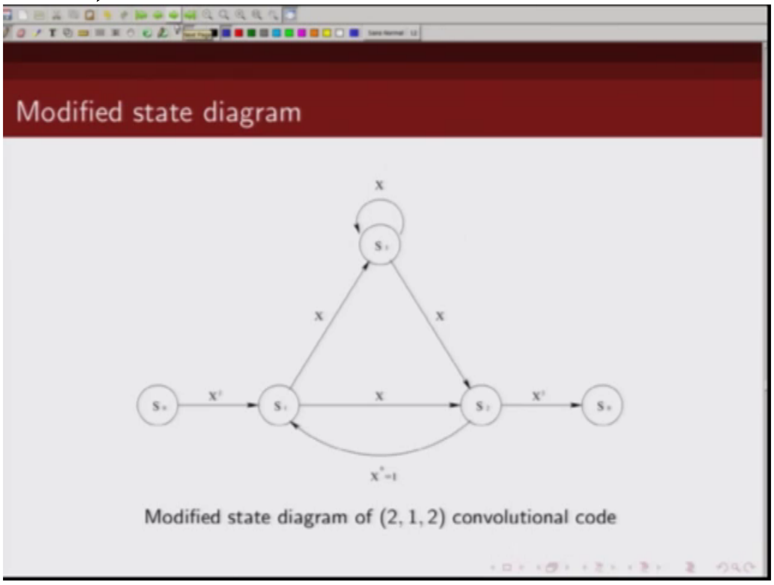
to 1 0 and its output weight is 1 1 which is x square. So let's go back here. From 0 0 I am going to this state and output is x square. This branch is labeled by x square.

(Refer Slide Time 18:37)



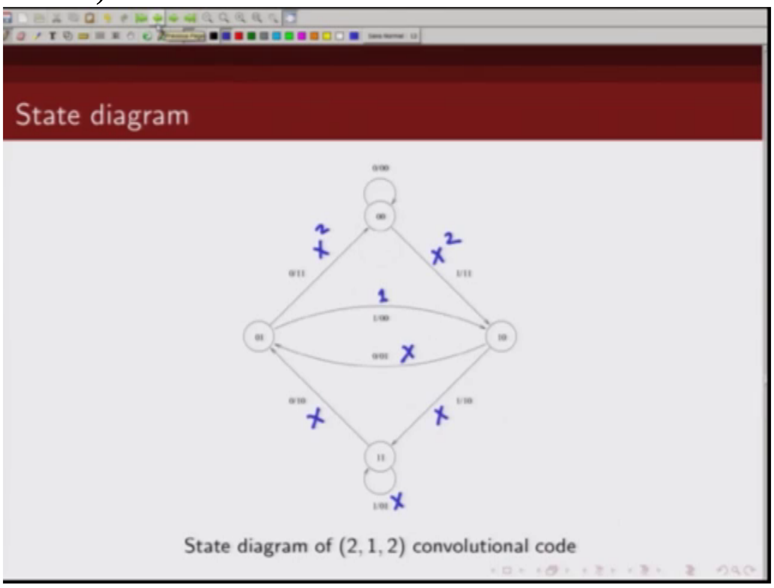
Similarly you can see from, say from this state you are going to this state and the weight is x. You can see from this I am going to

(Refer Slide Time 18:49)



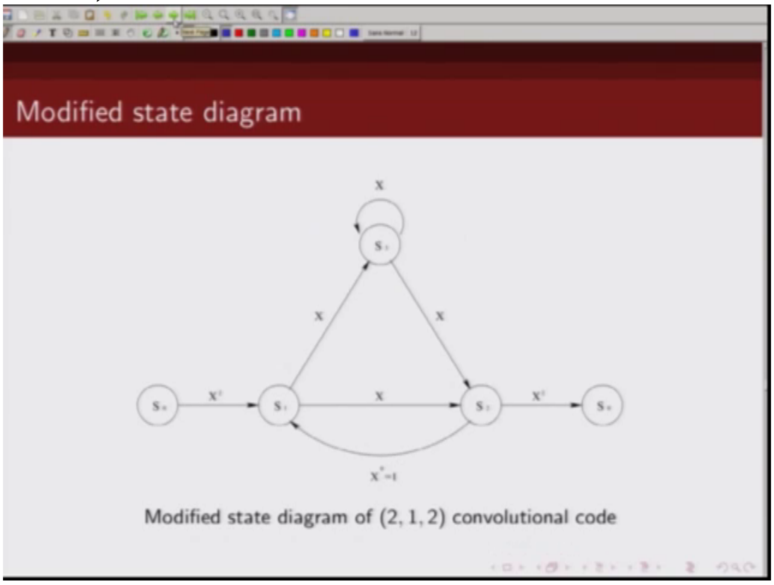
this state and the branch is labeled by x. From this you are

(Refer Slide Time 18:57)



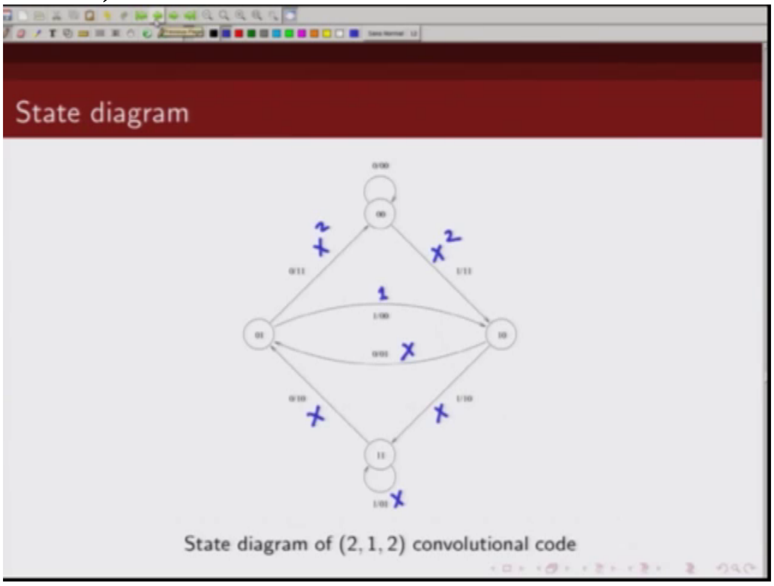
going to this 1 1 state and the branch is labeled by x.

(Refer Slide Time 19:04)



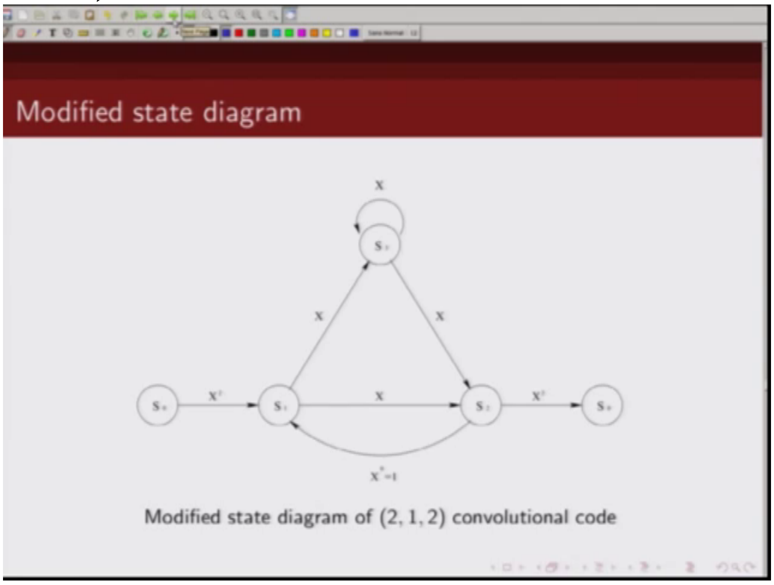
From this state you are going to this 1 1 state and the branch is labeled by x.

(Refer Slide Time 19:11)



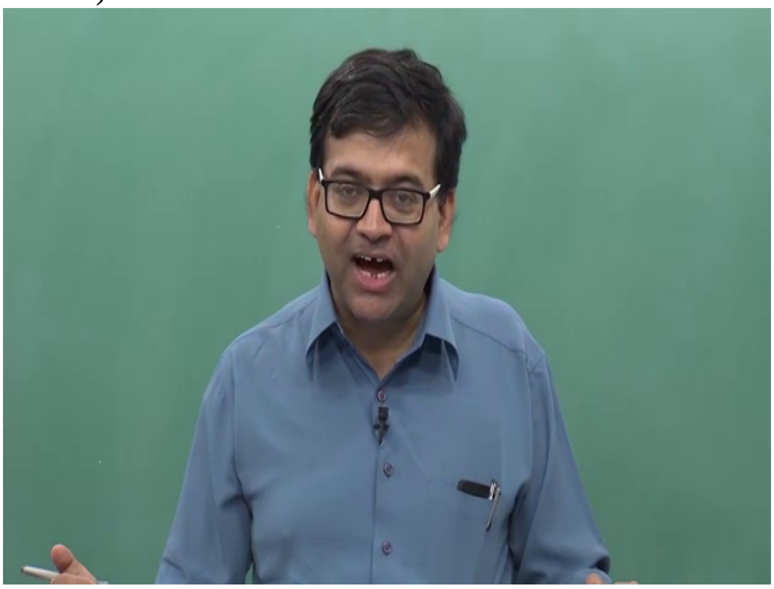
Around this state 1 1 there is a loop which has weight 1 x.

(Refer Slide Time 19:21)



This is my loop around 1 1 which has weight x. So like that basically we modify the state diagram and this is how our modified state diagram

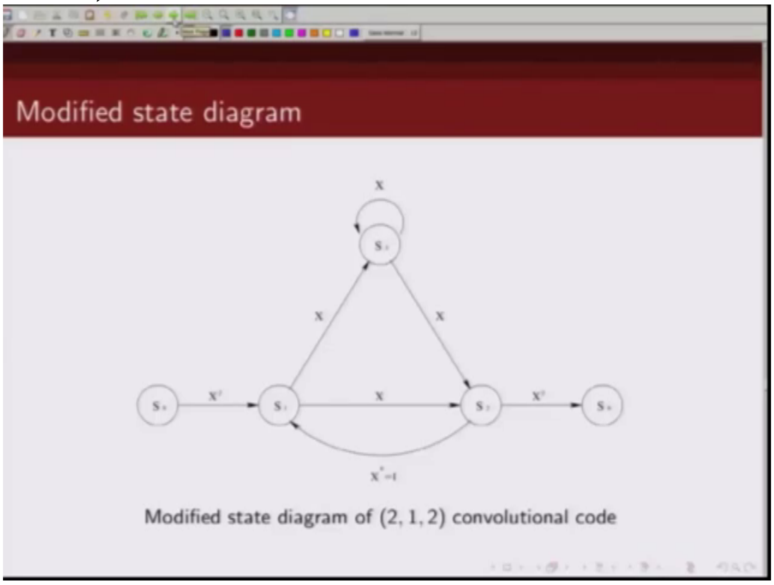
(Refer Slide Time 19:37)



of rate 1 by 2 convolutional code that we just showed looks like, Ok.

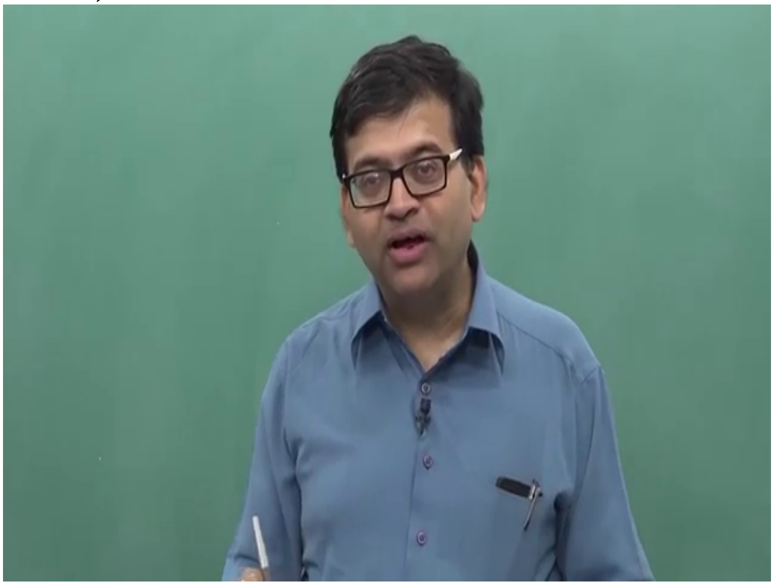
Now the next step is

(Refer Slide Time 19:48)



we need to find out what are the forward paths, what are the loops, what are the non touching loops and then we need to find out the

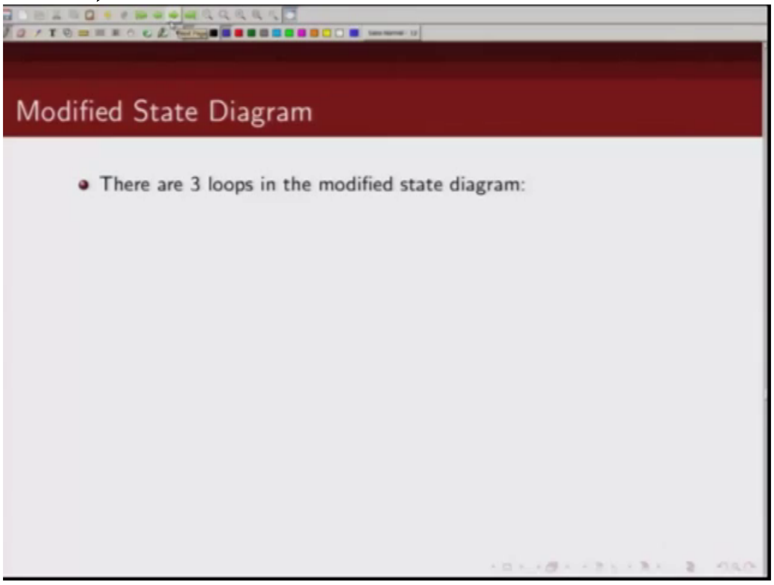
(Refer Slide Time 19:59)



path gains along those forward paths, we need to find those deltas corresponding to these forward paths and then we need to apply Mason's gains formula to get the weight enumerating function.

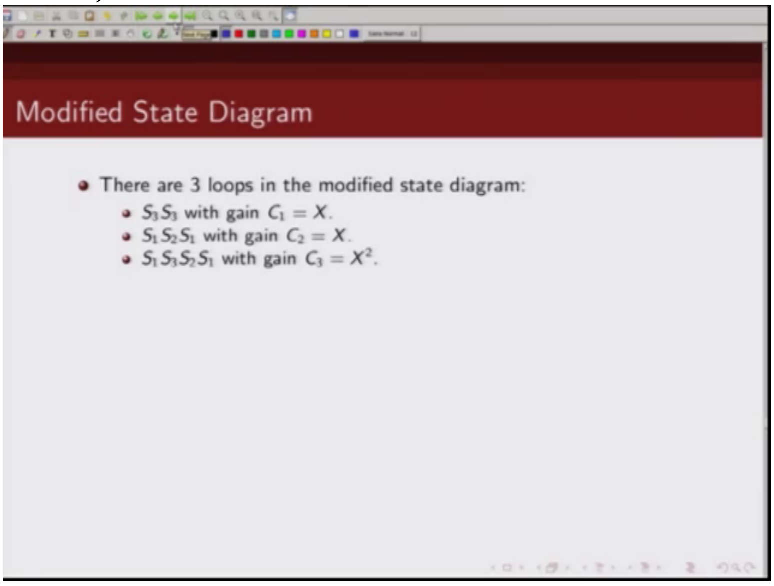
So first

(Refer Slide Time 20:18)



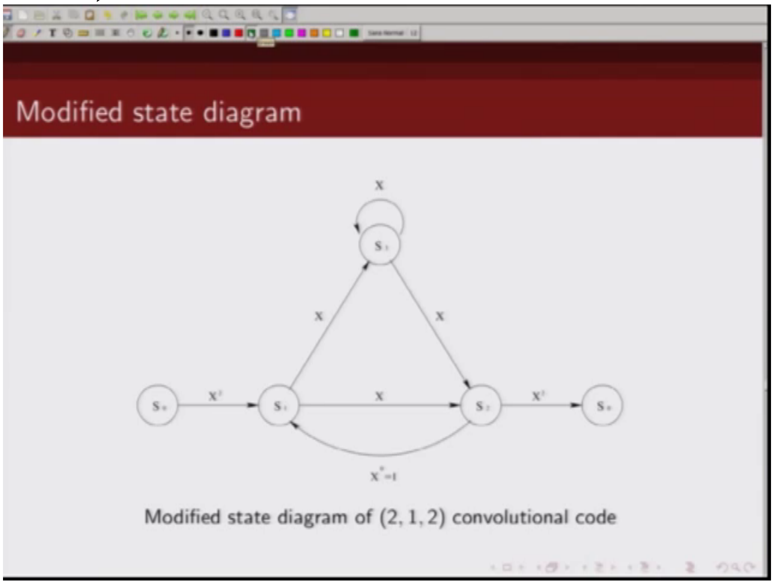
we find out what are the loops. So there are 3 loops in this. One is a

(Refer Slide Time 20:26)



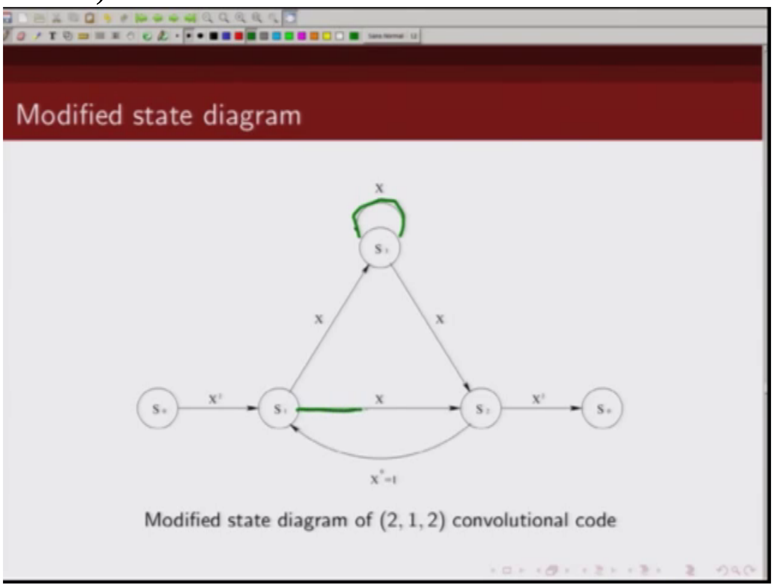
self loop around this state s 3, you can see. There is one

(Refer Slide Time 20:31)



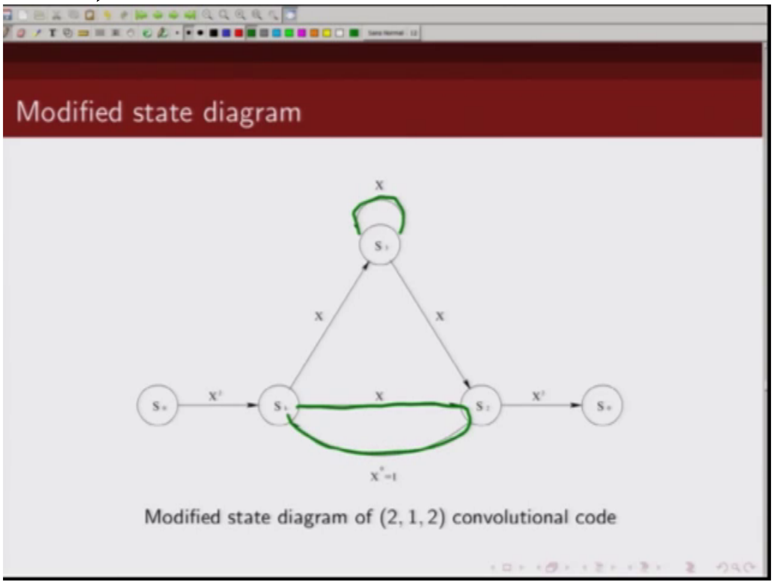
loop, right. There is another loop,

(Refer Slide Time 20:37)



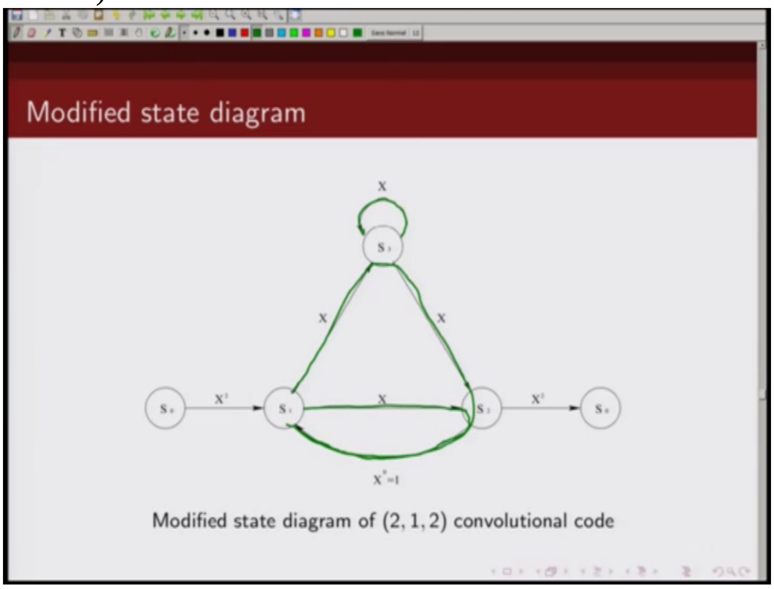
right

(Refer Slide Time 20:40)



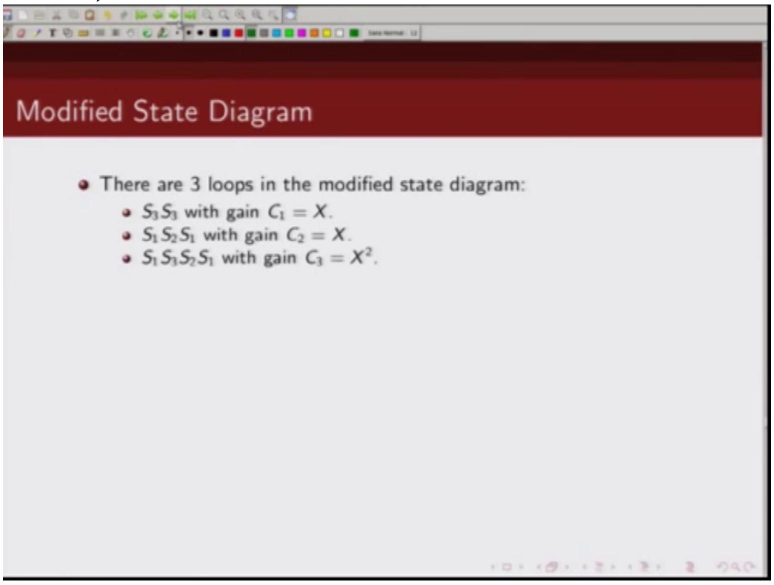
and then there is another loop. So there are 3 loops here. That's what I am denoting

(Refer Slide Time 20:49)



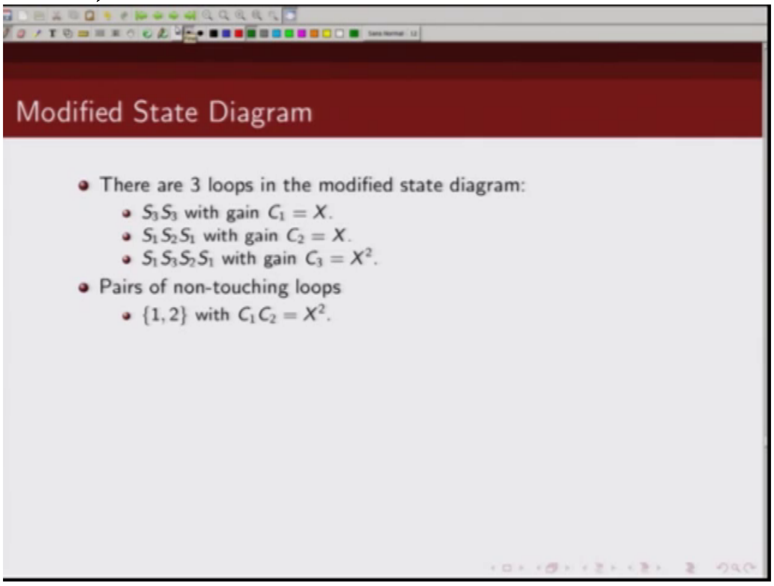
it by S 3, S 3, its gain is x, next one is S 1 S 2 S 1. S 1 S 2 S 1 is this one, Ok and the third loop is given by this. So these are

(Refer Slide Time 21:07)



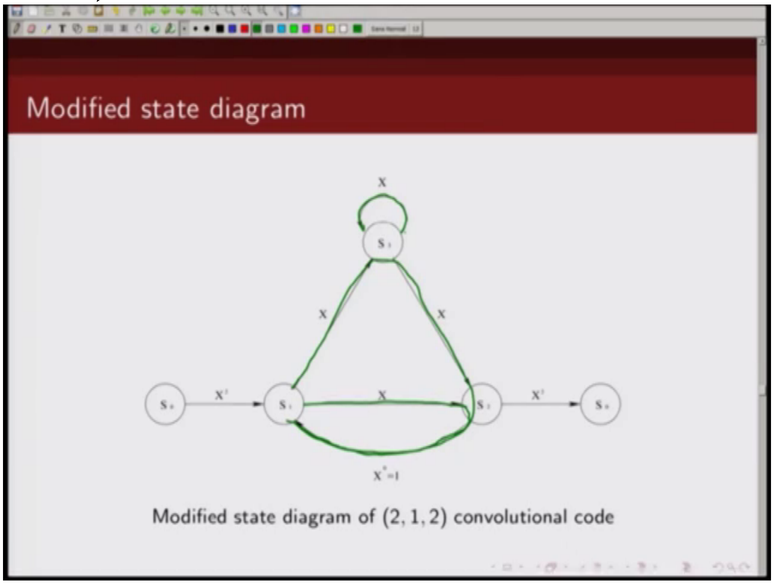
the three loops and corresponding to these 3 loops these are the gains. Next what are the pair of non touching loops? Now only

(Refer Slide Time 21:23)



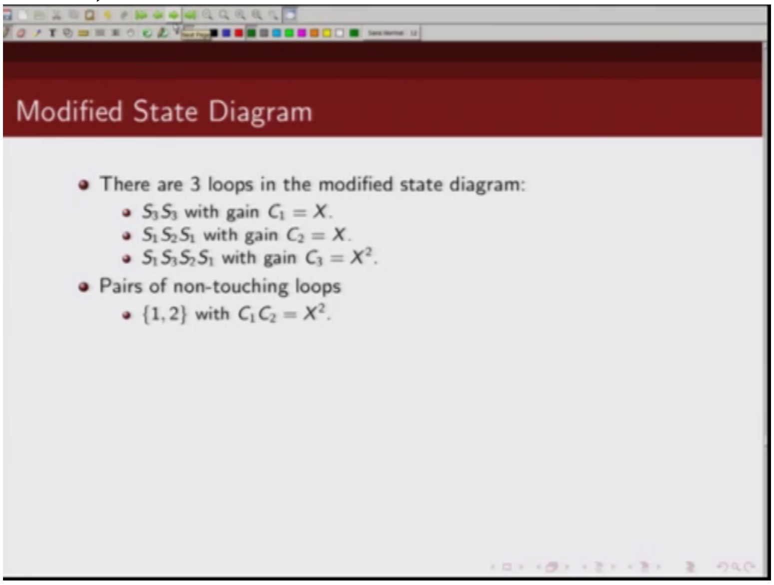
these 2, C 1 and C 2 are non touching loops. You can see, again go back to this example.

(Refer Slide Time 21:32)



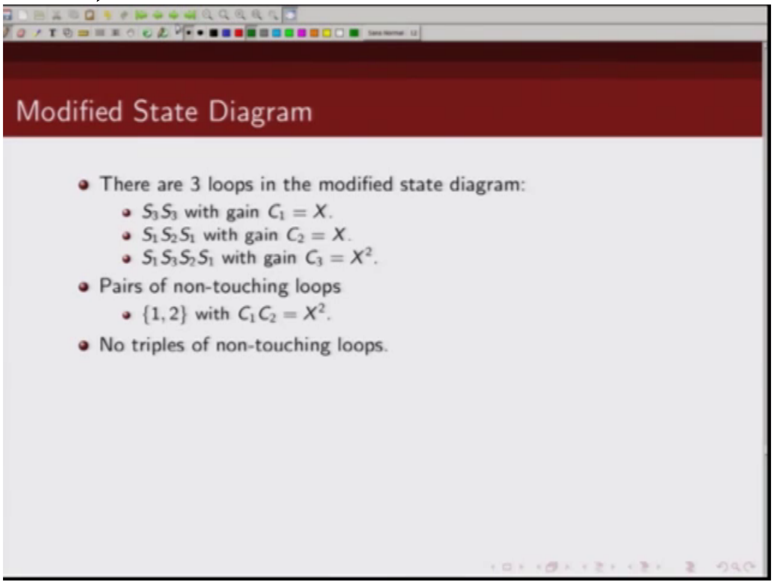
This loop and this loop are non-touching. Why? This loop contains s 3 and this loop contains s 1 and s 2, so they don't have any state common between these 2 loops. So the set of non touching loops is basically this C 1 and C 2 and the gain corresponding to them is basically

(Refer Slide Time 22:00)



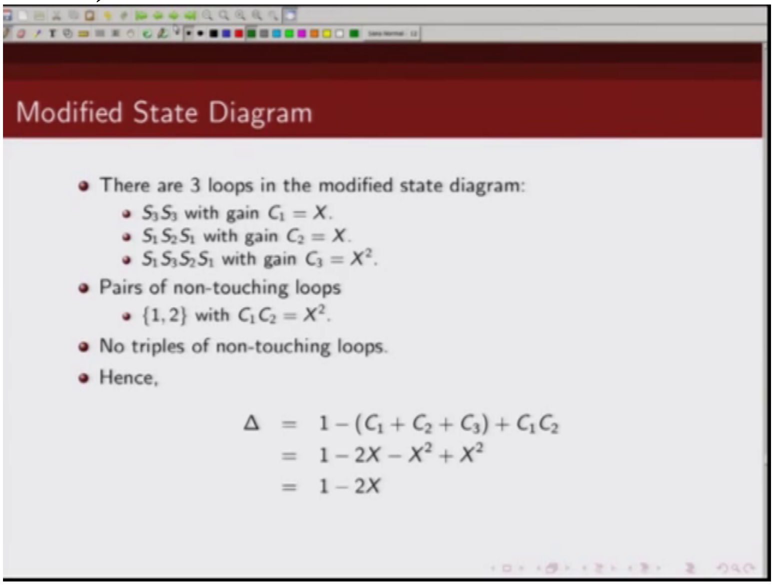
x square. And there is

(Refer Slide Time 22:04)



no set of 3 loops which are non touching. So now

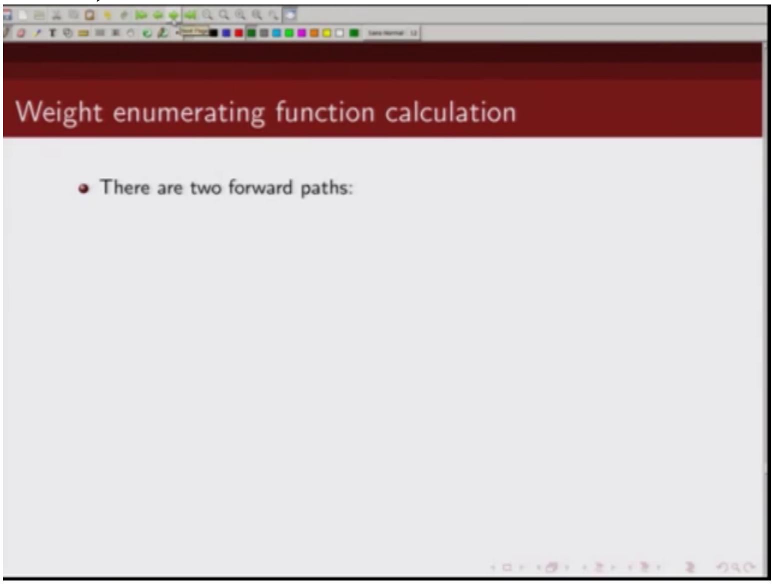
(Refer Slide Time 22:10)



we can then find out the value of delta which is 1 minus summation of these loop gains and plus set of non touching loops so this comes out to be 1 minus 2 x.

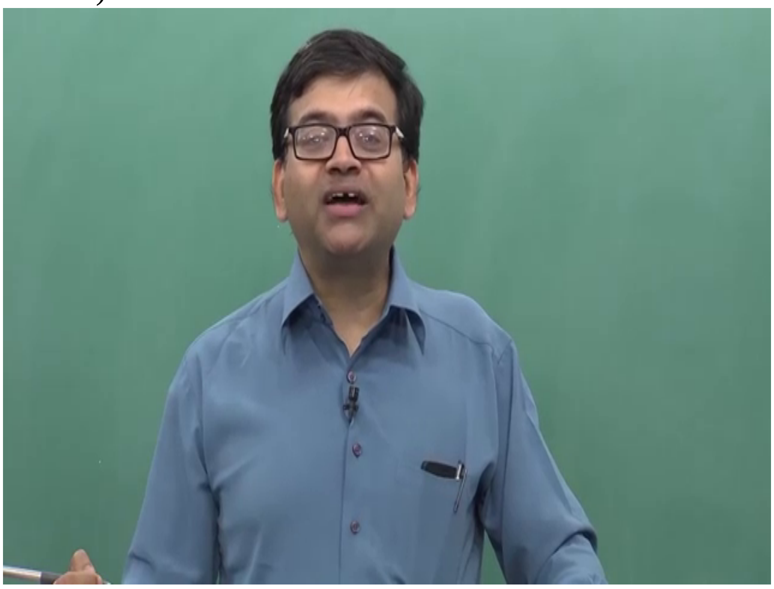
Next

(Refer Slide Time 22:28)



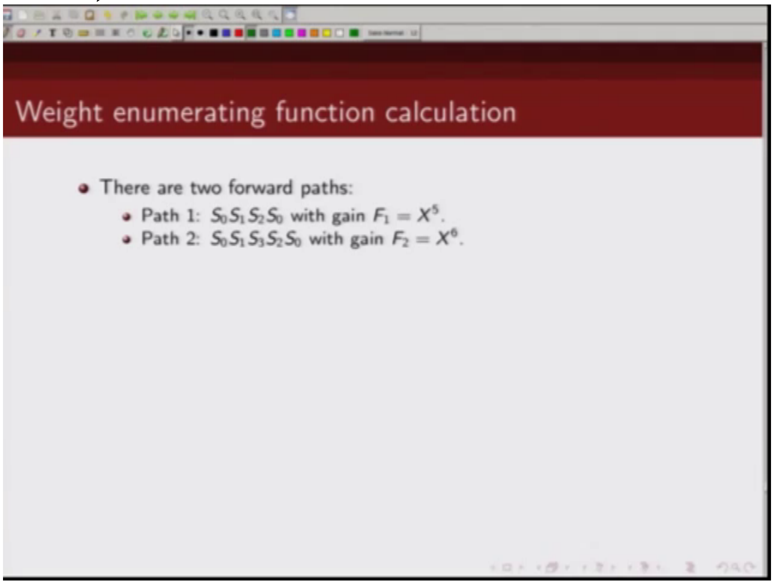
we are going to find out what are the

(Refer Slide Time 22:31)



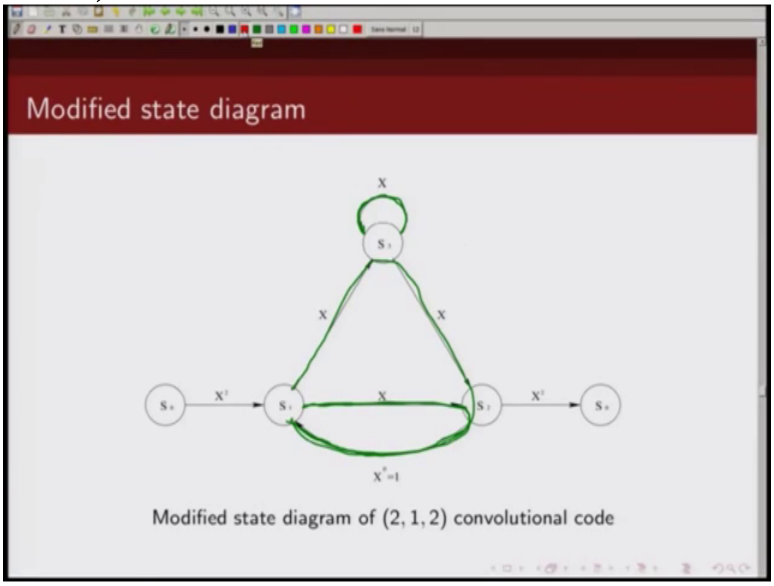
forward paths. So there are 2 forwards paths in

(Refer Slide Time 22:37)



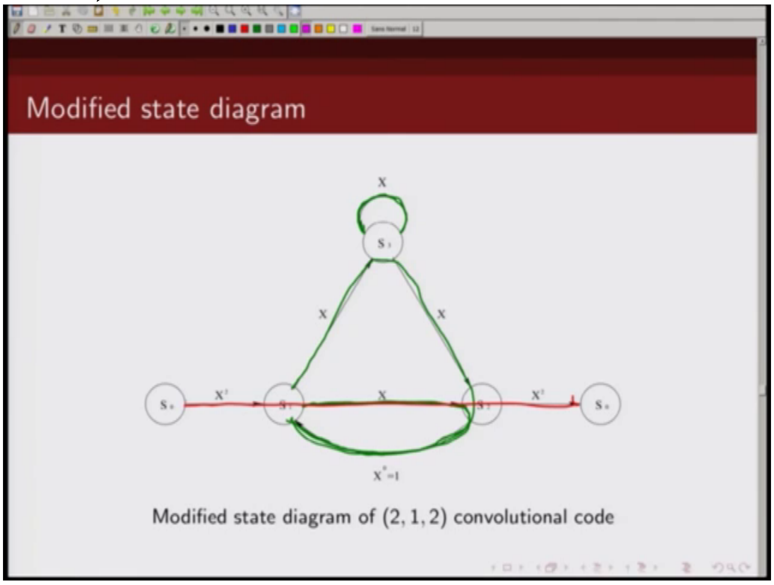
this and we are going to show you. So let's use a different

(Refer Slide Time 22:42)



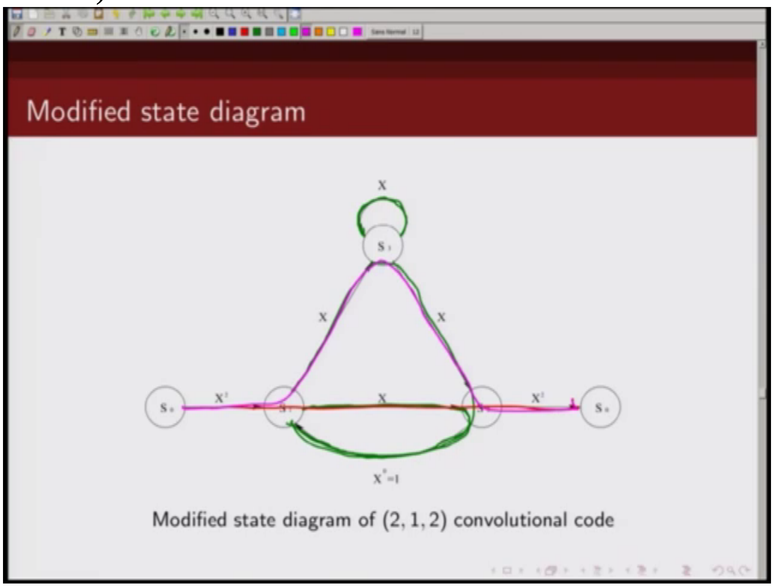
color pen. Let's use a red color pen. Remember what's a forward path? Path from the initial state to the final state without going over any state twice. So one forward path is this, fine. What about another

(Refer Slide Time 23:07)



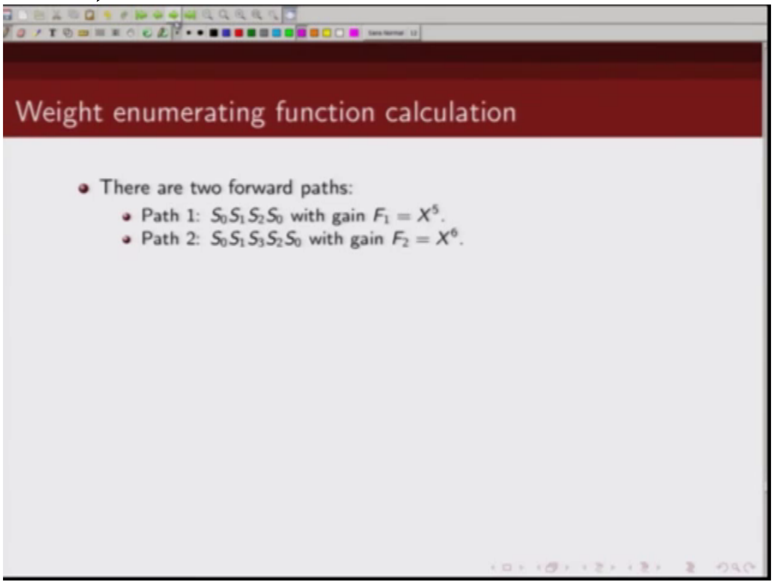
path, another forward path? The another forward path is this.

(Refer Slide Time 23:17)



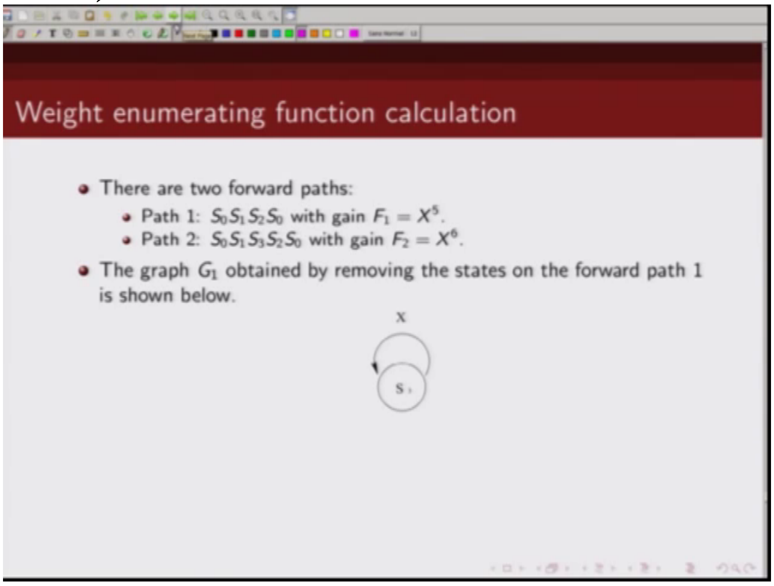
Both the cases you can see I am not going over any state twice. And there are only 2 forward paths in this case. And what are their corresponding path gain? For the one which I marked with red, it is x square x and x square. So this will be x raised to power 5. And this will be x square x x x square. So this will be x raised to power 6. So then

(Refer Slide Time 23:47)



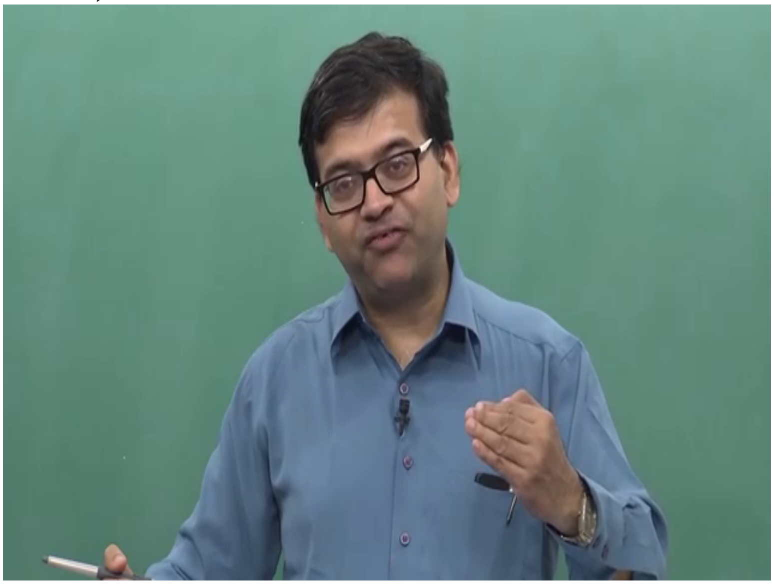
we have 2 forward paths, the one with gain x 5, another one with gain x 6.

(Refer Slide Time 23:57)



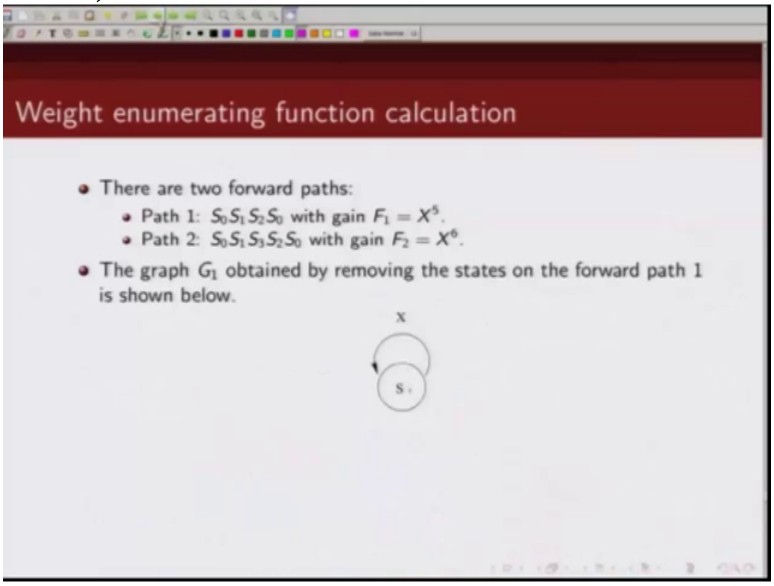
Now what's the next step? We need to remove the forward path and see what is the graph remaining

(Refer Slide Time 24:04)



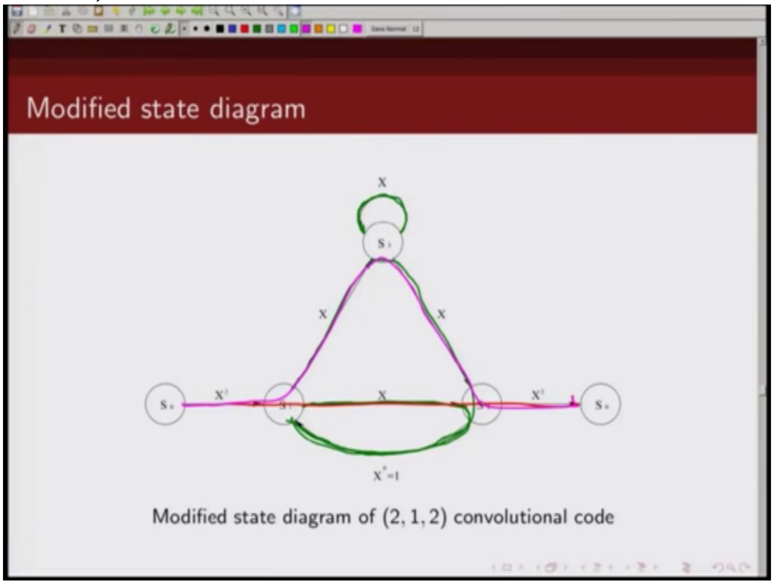
and we need to compute the delta corresponding to that. Now again let's go back

(Refer Slide Time 24:10)



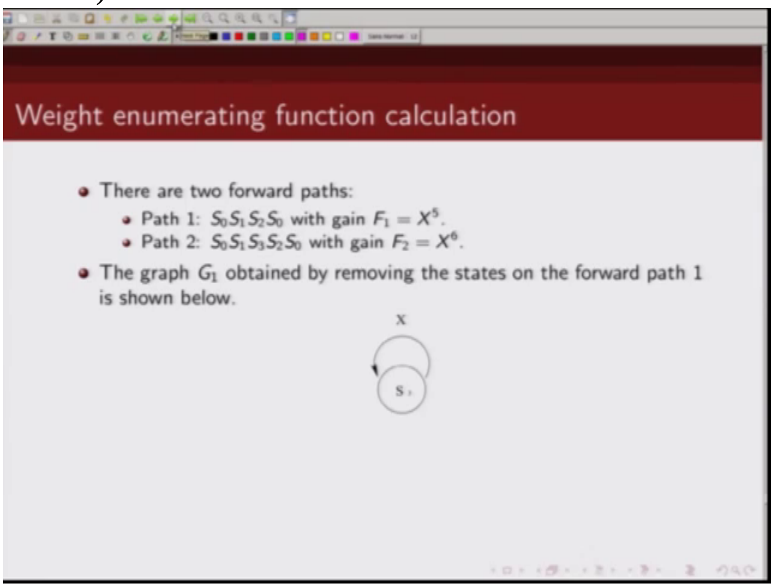
to the same diagram. If I remove this

(Refer Slide Time 24:15)



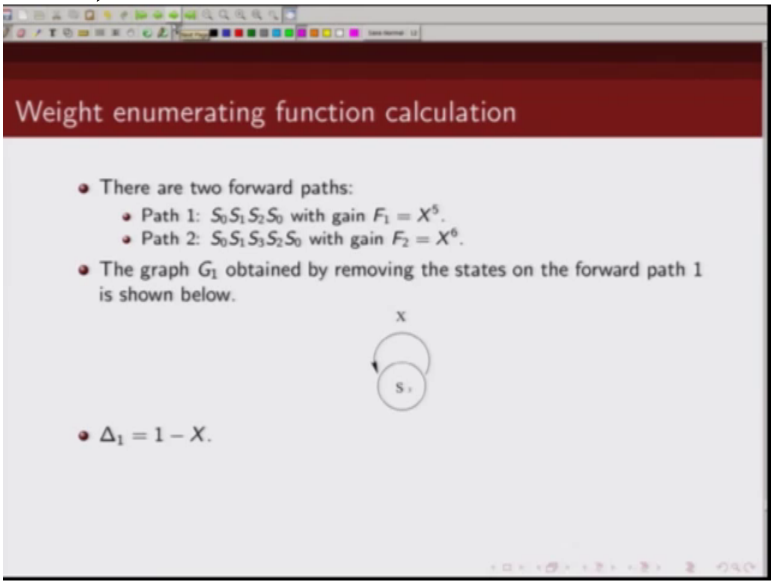
forward path, what is left in the graph? Only this, this node. Only this is remaining. And what if I remove this forward path? If I remove this forward path, everything is gone, nothing left in the graph. So that's what I am saying here. If I remove the

(Refer Slide Time 24:35)



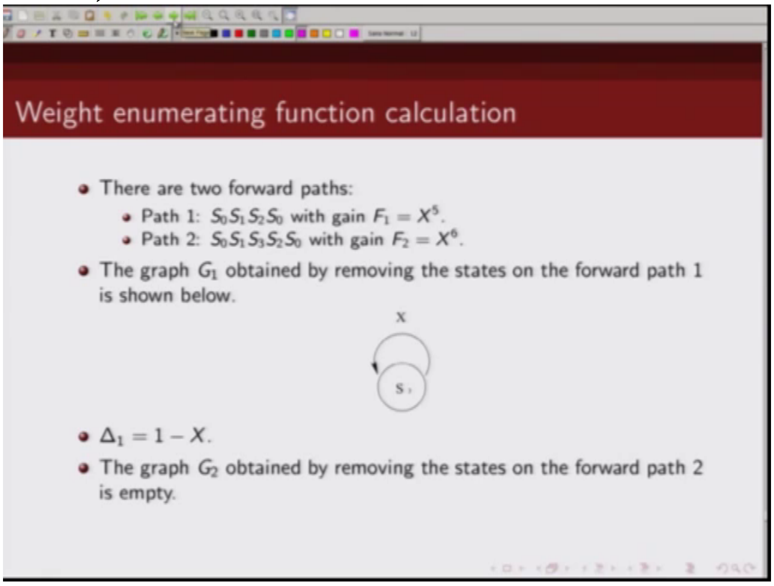
forward path 1, the only graph remaining is this. And the delta

(Refer Slide Time 24:41)



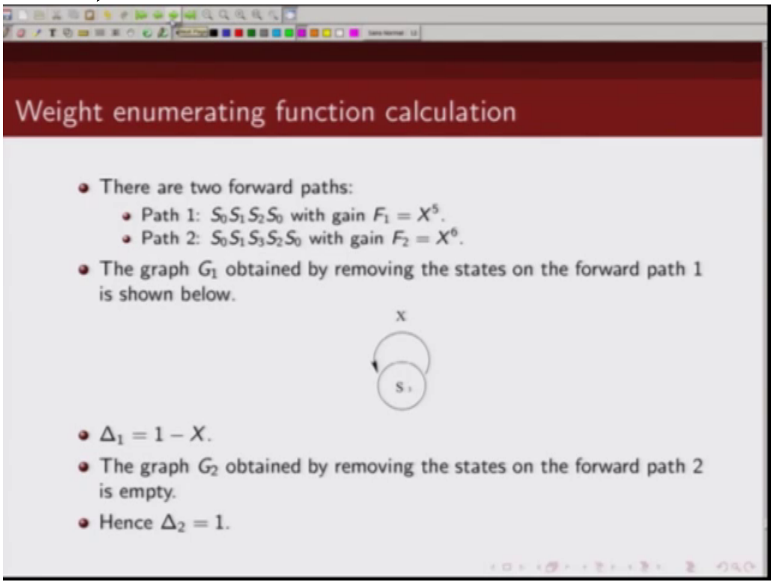
corresponding to this is only one loop with gain x so this is 1 minus x. And for the second case, there is no graph left so delta 2 will be

(Refer Slide Time 24:53)



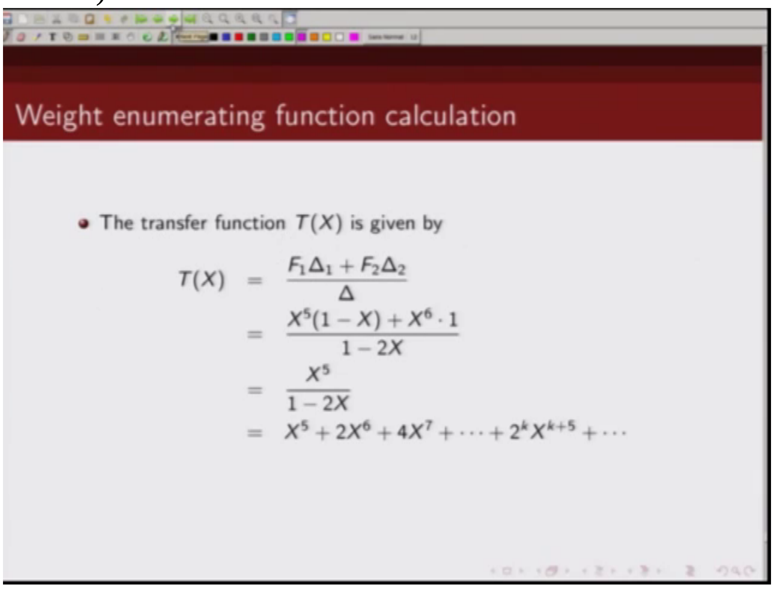
1, Ok. So now I have f 1 delta 1, f 2 delta 2 and I also have the value of delta. So I can then apply Mason's gain formula to

(Refer Slide Time 25:09)



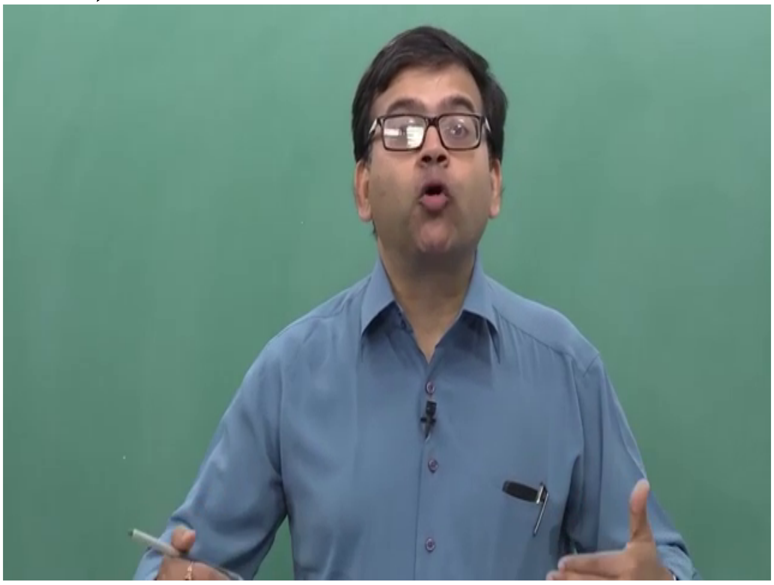
get the weight

(Refer Slide Time 25:12)



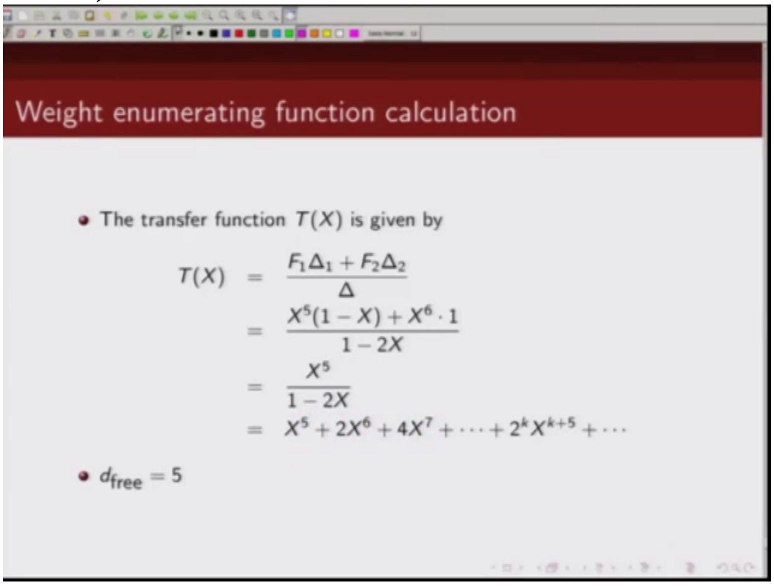
enumerating function. So the weight enumerating function is given by this expression. So I plug in the value of f 1 delta 1, f 2 delta 2 and delta and what I get is this expression which I can write like this. So you can see basically my output consists of 1 codeword of weight 5, 2 codewords of weight 6, 4 codewords of weight 7. So you can see this transfer function is completely enumerating the weight distribution of my convolutional code. The same thing I can do with augmented

(Refer Slide Time 25:56)



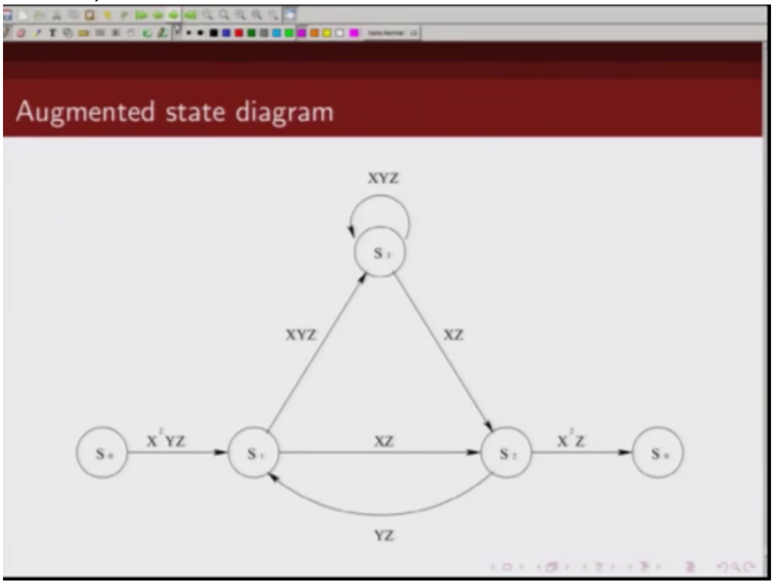
transfer function.

(Refer Slide Time 25:59)



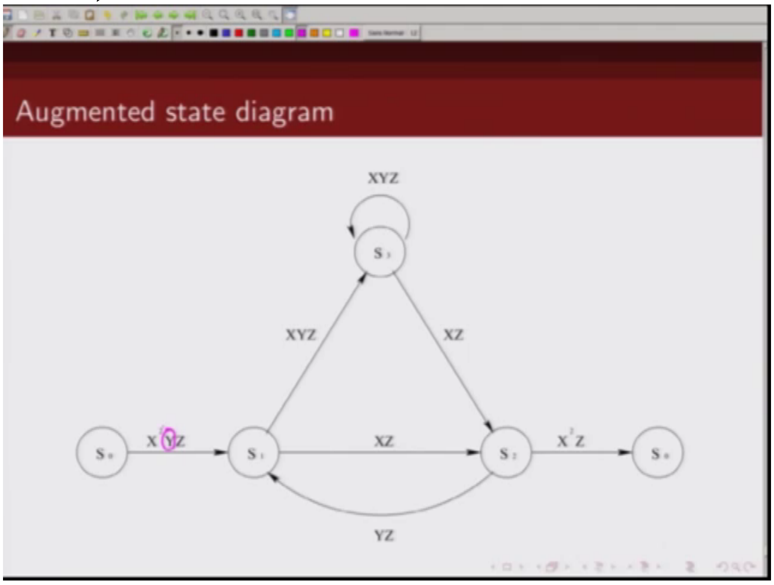
And again because the minimum weight is 5, so free distance of this convolutional code is 5. Now

(Refer Slide Time 26:11)



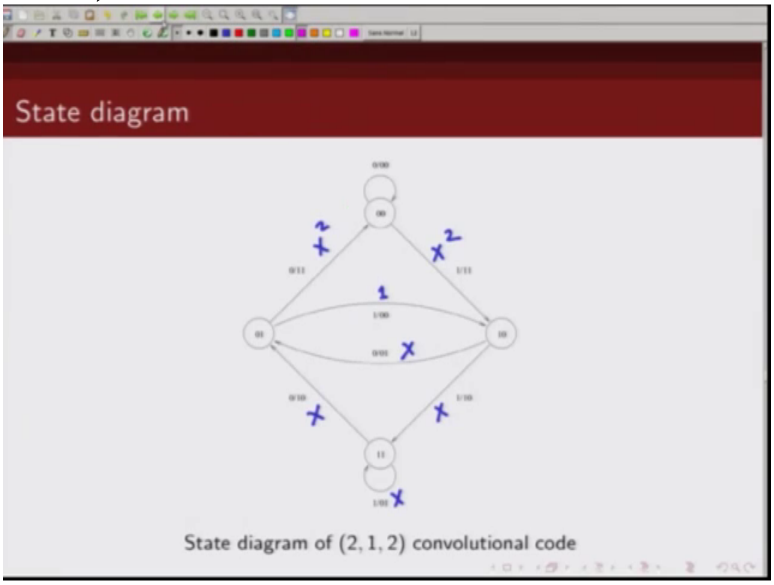
we repeat the same exercise with the augmented state diagram. Now what was augmented state diagram? Each valid branch we added a z to denote this you can reach from one state to another in one step and we also added in each of these branches the weight corresponding

(Refer Slide Time 26:37)



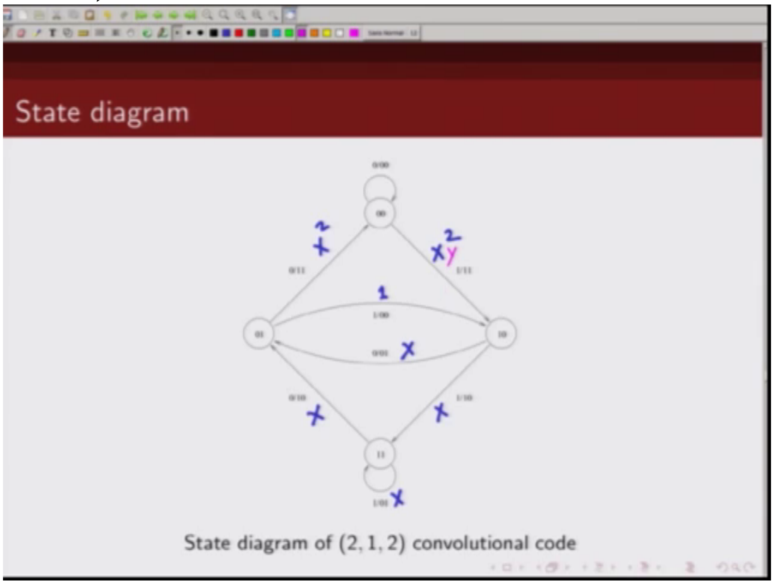
to the information bits. So the information bit weight was 0 so y 0, so that was 1, so you can see in some cases the information sequence weight is 0. So let's just go back to the original state diagram,

(Refer Slide Time 27:00)



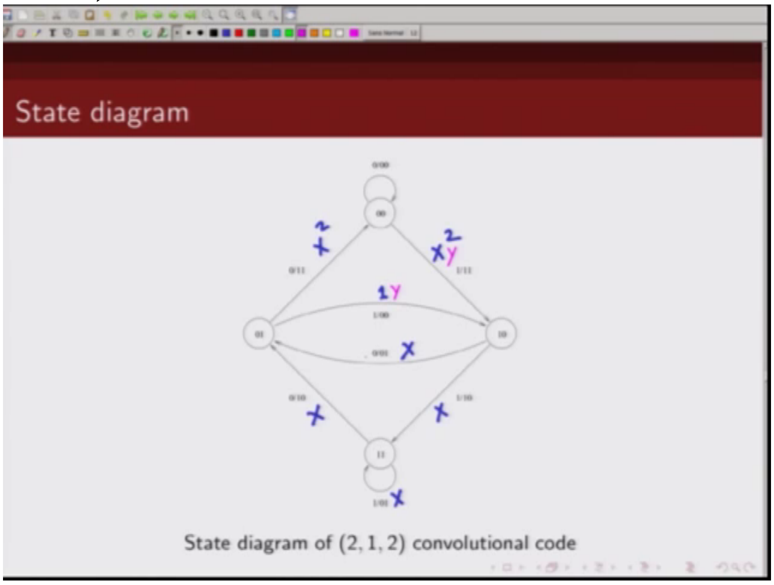
yeah. Let's go back to this. So you can see for this transition from 0 1 to 0 0 what is the weight of the information sequence, that is 0. So y 0 that is basically 1. What about this, the weight of information sequence? Here the input is 1, so this will be y.

(Refer Slide Time 27:24)



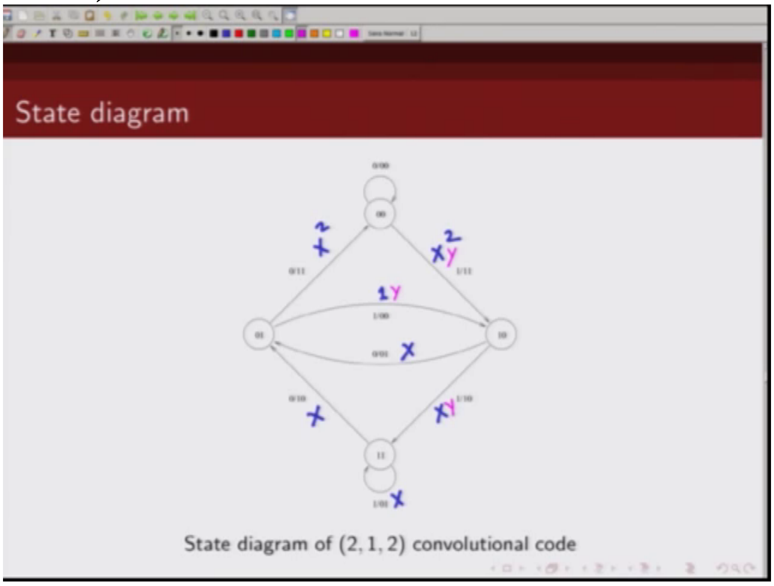
What is the weight of information sequence? That's 1 so it will be y. This weight information

(Refer Slide Time 27:33)



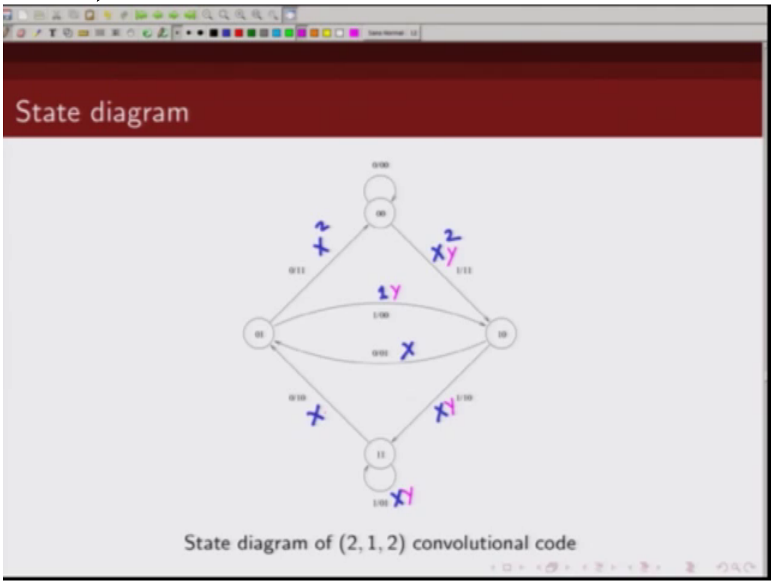
sequence is 0 so y 0 is 1. So wherever you had 1 here you are adding basically y.

(Refer Slide Time 27:40)



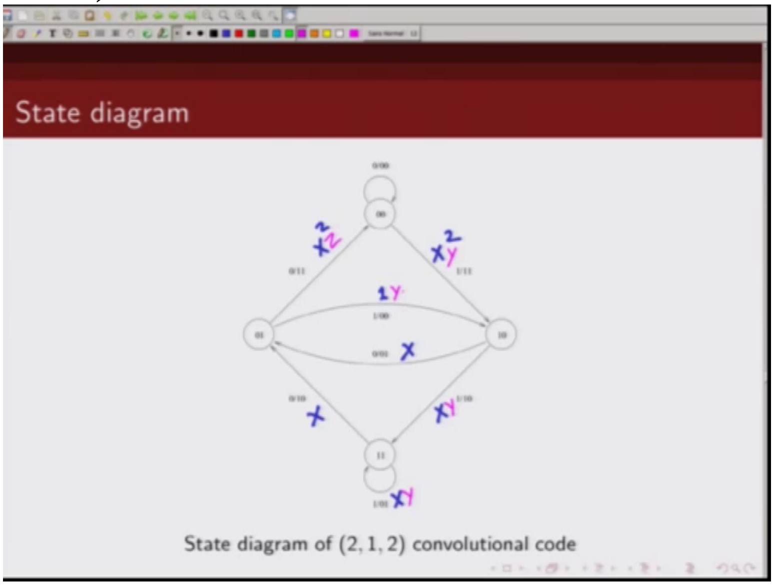
This is 1 and similarly, each of these

(Refer Slide Time 27:45)



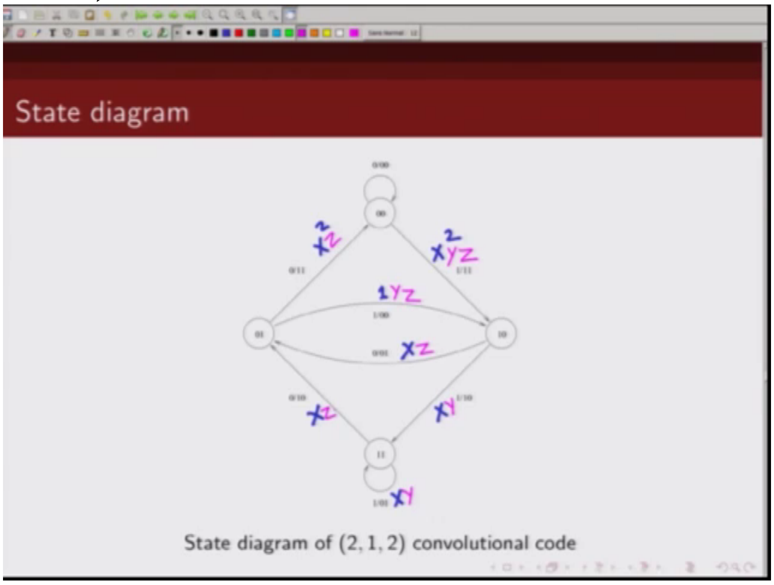
transitions there will be a z added

(Refer Slide Time 27:48)



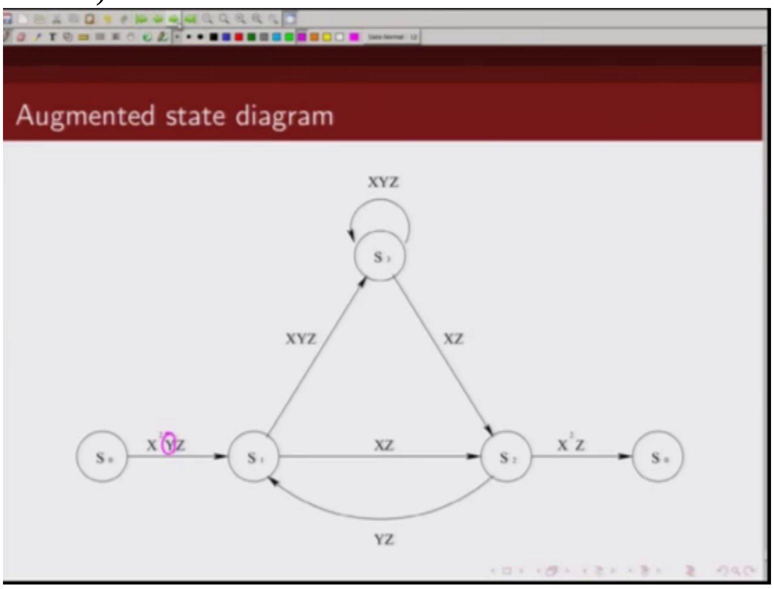
to denote the length

(Refer Slide Time 27:55)



Ok. So that's your augmented state diagram. And that's what; I mean the completed augmented state diagram is what I am showing you here. This

(Refer Slide Time 28:12)

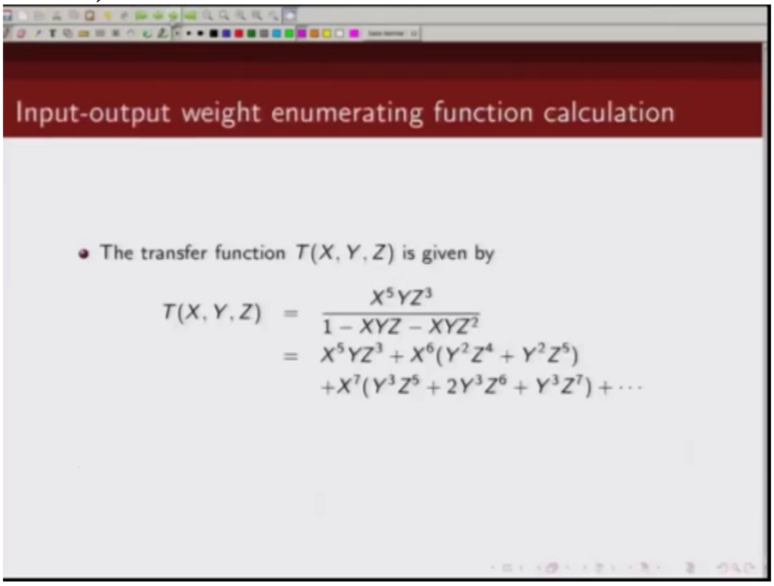


is basically my augmented state diagram where I am not only specifying the coded weight but I am also specifying what input causes that output bit and z to denote the length and I follow the same procedure

(Refer Slide Time 28:32)

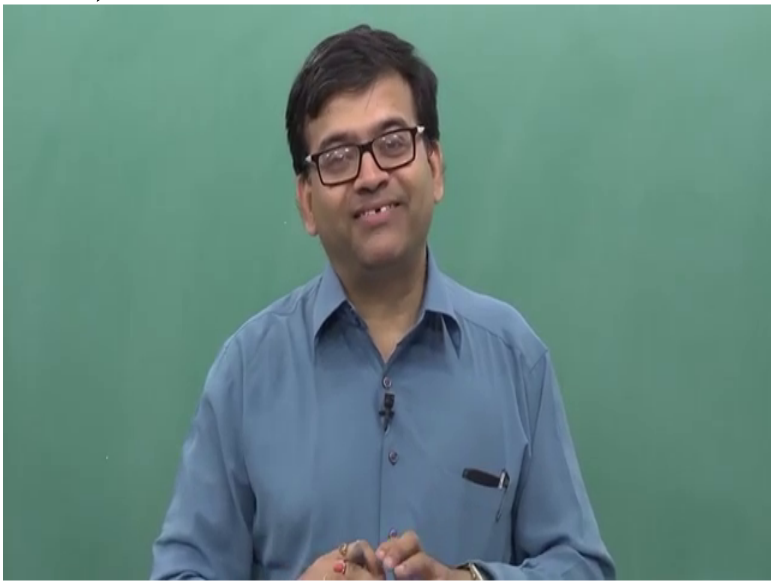
using Mason's gain formula to compute

(Refer Slide Time 28:36)



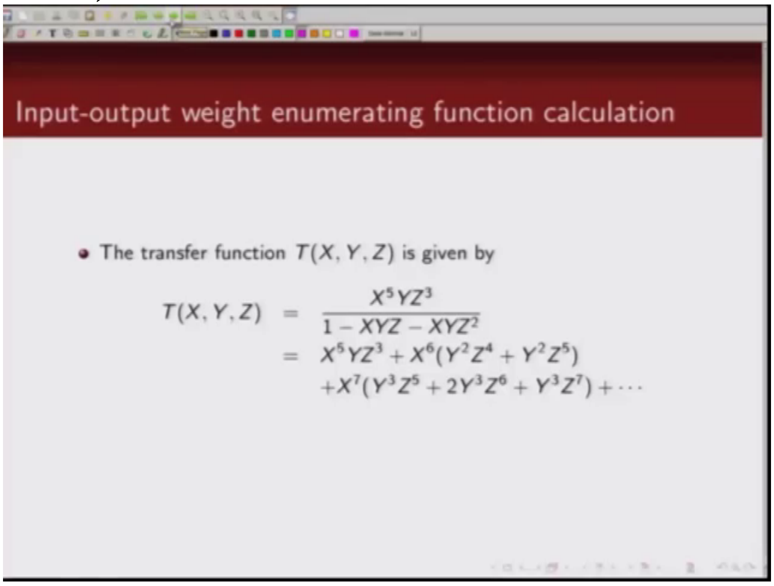
the weight enumerating function. So I get this information. I am skipping the steps. It is exactly the

(Refer Slide Time 28:43)



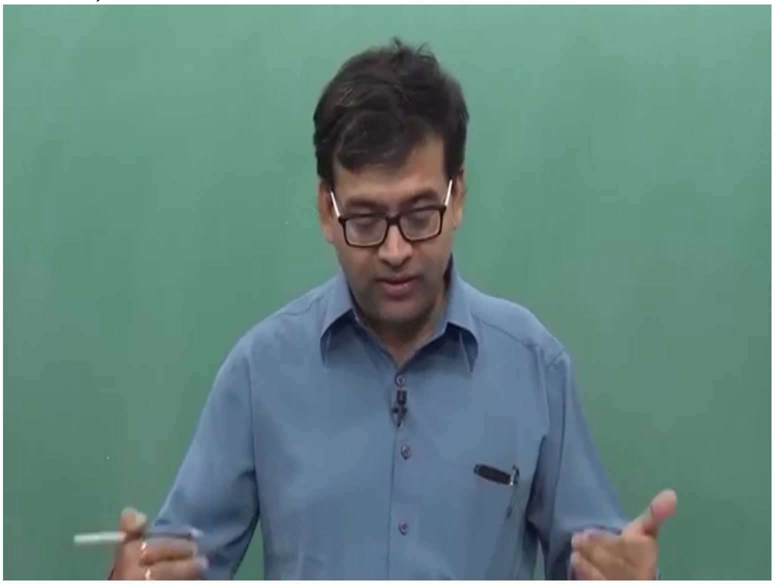
same procedure I just laid out for computing the weight enumerating function and you can see it gives us lot more

(Refer Slide Time 28:51)



information. The weight enumerating function said we had one codeword of weight 5. Now it says that codeword of weight 5 basically was caused by message information bit 1 and the length of the digression from all zero state before it merged with all zero state was 3. Similarly there we had shown there were 2 codewords of weight 6. This completely specifies what those two codewords was. One which was generated by message bit weight 2 of length 4; this was message bit 2 length 5. So you can see the augmented state diagram, if we use it to generate

(Refer Slide Time 29:42)



the transfer function it gives us lot more information. So with this I will conclude this lecture.

Thank you.