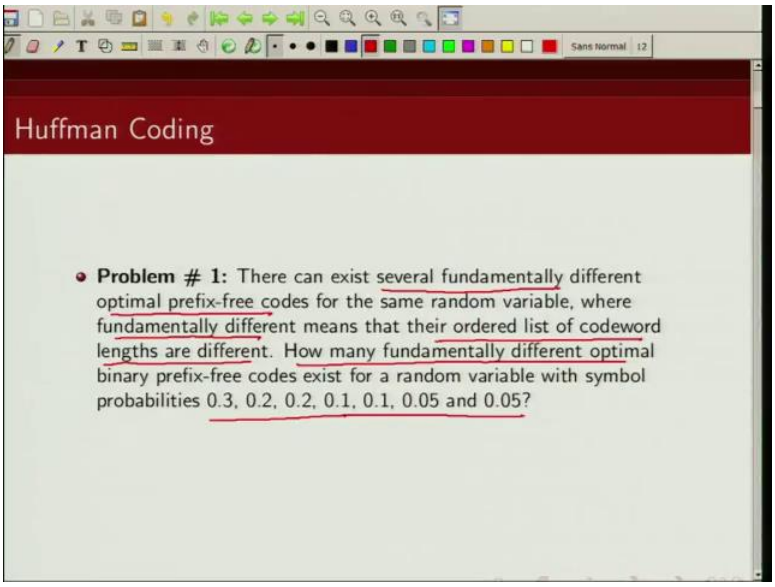
An Introduction to Information Theory
Prof. Adrish Banerjee
Department of Electronics and Communication Engineering
Indian Institute of Technology, Kanpur

Lecture – 6C
Problem Solving Session – II

Welcome to the course on An Introduction to Information Theory. I am Adrish Banerjee. So far we have studied block to variable length coding and variable to block length coding. Today let us try to solve some problems in source compression.

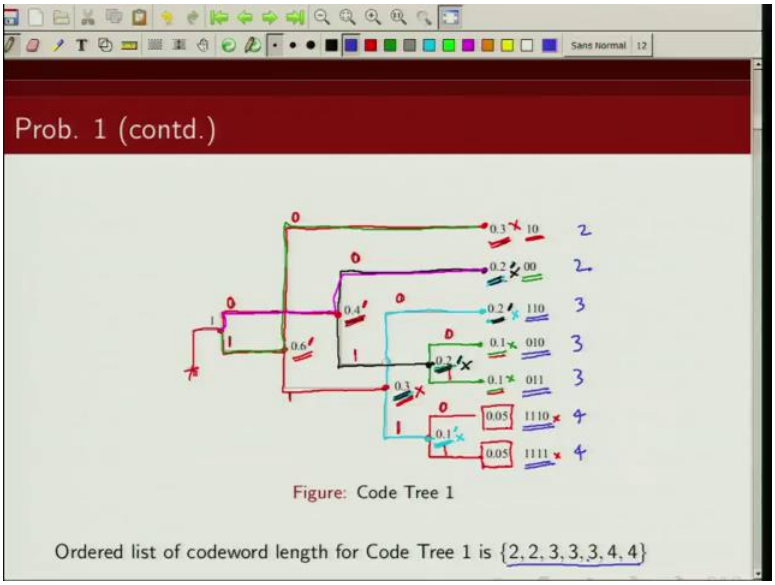
(Refer Slide Time: 00:32)



So, first problem relates to Huffman coding. So, there exist several fundamentally different optimal prefix free codes. Now we know Huffman coding is the optimal prefix free code. So, there exist several fundamental different Huffman code and what do I mean by fundamentally different - if I write down the ordered list of codeword length that is different. So, when I say fundamentally different optimal prefix free code I mean these are Huffman codes and if I arrange the Huffman code by the length of the codeword's there ordered list of codeword length is different, if they are fundamentally different optimal prefix free code.

The question is how many fundamentally different optimal binary prefix free code exist for a random variable which has following symbol probabilities given by 0.3 0.2 0.2 0.1 0.1 0.05 and 0.05. So, this random variable takes seven different values and these are the probabilities associated with it. The question is we have to construct optimal binary prefix free code and you have to tell how many fundamentally different optimal prefix free codes exist.

(Refer Slide Time: 02:11)



So, we know how to construct optimal prefix free code we know the properties of Huffman code, recall that if it is a binary optimal prefix free code the 2 least likely codeword's they differ in only one bit location and there is no leaf which is unused. And if you recall the way we proceed is we start with 2 symbols which have the least probability, we join them create a new node deactivate old nodes and this new node is now it has probability equal to some other probabilities of this old nodes.

So, this way we at each step we combine the 2 least likely nodes until we are left with only one node which is a roof node. So, we will do that and we will see how we can get fundamentally different optimal prefix free code. So, in this example these are the probabilities of u's i's 0.05 0.05 0.1 0.1 0.2 0.2 and 0.3. So, first we are going to combine these 2. So, if we combine these 2 we create a new node which has probability of 0.1 and

we deactivate these 2 nodes at this point we have now (Refer Time: 03:50) the active nodes this one, this one, this one, this one, this one and this one.

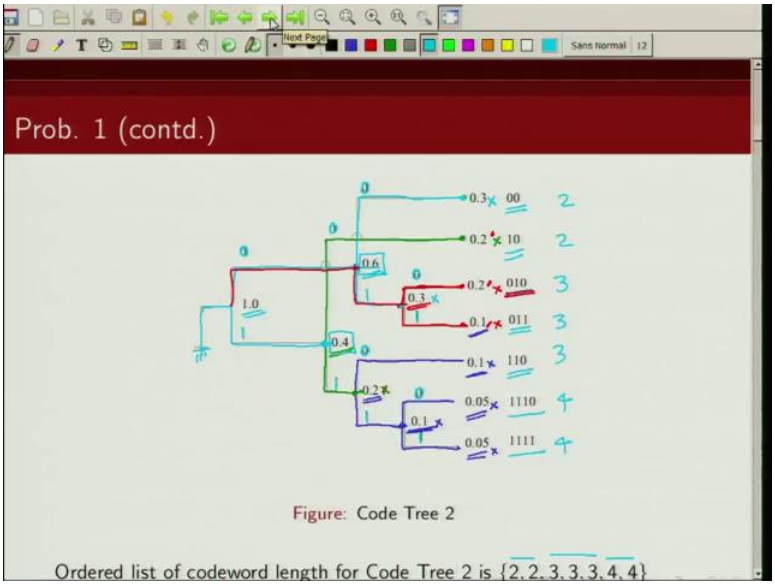
Now, note we have 3 nodes and if which have probability 0.1, this one, this one and this one. So, we could either combine these 2, we could combine these 2 or we could combined these 2. So, in this case I am just combining these 2 if I do that I create a new node which is probability given by 0.2 and I deactivate these 2 nodes. So, at this point my active nodes are this one, this one, this one, this one and this one. Now I have this as probability 0.1 this is 0.2 and this is 0.2 and this is 0.2. So, I could combine this with this I could combine I could combine this 0.1 with this 0.2 or I can combine this 0.1 with this 0.2 or I can combine this 0.1 with this 0.2.

So, there are 3 point 2s this one, this one and this one, I could combine this node which has probability 0.1 with any of these 3 nodes. So, in this example I am just combining with this one. So, I get a new node whose probability is given by 0.3 and I deactivate this node I deactivate this node. So, at this point my active nodes are this one, this one, this one and this one, and what are the probabilities I have I have 0.3 here, 0.2 here, 0.2 here, 0.2 here and 0.3 here. So, again note that I have 3 nodes which have probability 0.2 - this one, this one and this one. So, I can combine any 2 of them right. So, in this case I am combining this you have this. So, then I create a new node which has probability 0.4 and I deactivate this node and I deactivate this node.

Now, at this point my active nodes are this one, this one and this one. So, these 3 are my active node. So, now, I have this node probability 0.3, this node probability 0.3, and this node probability 0.4. So, I am going to combine these 2 nodes and this has my probability 0.6 and finally, I deactivate these 2 nodes I am left with on the 2 active nodes one is this one and other is this one. So, combine both of them and the final step is I assign 0s and 1s to these one set. So, if I do that what I come out with is the codeword for this particular symbol is 10 this you can see 10 codeword for this comes out to be 00 you can see 0 0 and similarly you can find out codeword's (Refer Time: 07:58) this is 110 110 011 1110 and 1111.

So, this is codeword length 2 2 3 3 3 4 4. So, if I write the ordered list of codeword's I have 2 of them with length 2 3 of them having length 3 and 2 of them having length 4. Now let us look at another way in which we can construct this optimal binary prefix free code.

(Refer Slide Time: 08:40)



So, same example, the 2 smallest probabilities are 0.05 and 0.05. So, we can combine these 2 and create a new node, which is probability given by 0.05 plus 0.05 that is 0.1 we deactivate these node and activate this new node. So, now we have this node which is probability 0.1, this node probability 0.1, this node probability 0.1, so we can combine any 2 of them. So, let us combine these 2.

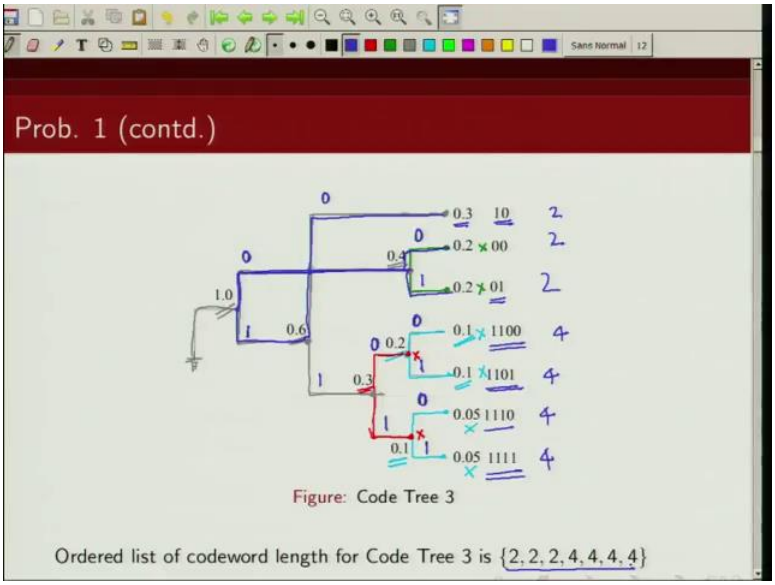
If you combine these 2 we get a new node with probability 0.2 we deactivate this node we deactivate this node. So, now the active nodes are this one, this one, this one, this one and this one. Now we have one node with probability point which is this one and there are 3 nodes with probability 0.2 this one, this one and this one. So, we have to combine this leaf which has probability 0.1 with any of these leaves or nodes which has probability 0.2. So, let us say we combine these 2 we get a new node which is probability 0.3 we deactivate this p node in these. So, then what are the activate nodes at this point

this one, probability 0.3 - this one probability 0.2, this one probability 0.3 and this probability 0.2.

So, we combine these 2 nodes which have probability 0.2 get a new node with probability 0.4 and we deactivate these 2 nodes. So, now, we have this active node with probability 0.4 this active node with probability 0.3 and this activate node with probability 0.3. So, we combine these 2 nodes create a new node with probability 0.6 and deactivate these 2 nodes. So, now, we have 2 nodes left one this one with probability 0.6 and this one with probability 0.4, you combine them we make this as a root node with probability 1 and finally, we assigned 0s and ones to these branches right assign 0s and ones to these branches. So, if you do that we get the codeword's of this particular symbol as 00, of this one 10, this 1010 and then you can check for example, let us go (Refer Time: 11:56) this is 0 and this is 1 and this is 0. So, it will be 010. Codeword for this is similarly 011, this 1110, this is 1110 and this is 1111.

So, we have this codeword of length 2 then 2 3 3 3 4 4. So, the ordered list of code which we have 2 codeword's of length 2, 3 codeword's of length 3, and 4 at 2 codeword's of length 4. Now if you compare with previous Huffman code is also had p codeword's of length 2, 3 codeword's of length 3 and 2 codeword's of length 4. So, it is not fundamentally, for the code 3 2 is not fundamentally different from code one because if I write down the ordered list of a codeword's though this nothing is different, but it is not fundamentally they are the same of set of ordered codeword.

(Refer Slide Time: 13:10)



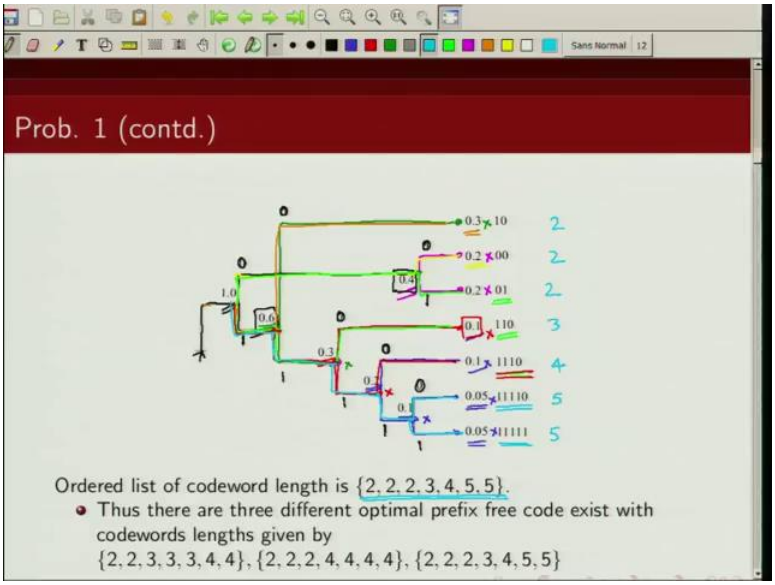
Now, let us look at another way of combining these nodes. So, and then you start up with these 2 nodes which have the least probability. So, we combine them create a new node which is probability of 0.1 we deactivate these 2 nodes. Next we have this node with probability 0.1, this particular leave it probability 0.1 and this node it probability 0.1. So, you can combine any two of them if you combine these 2 get a new node which has probability 0.2 we deactivate these nodes. So, at this point active nodes at this one with probability 0.1, this one with probability 0.2, this one with probability 0.2, this one with probability 0.2 and this one with probability 0.3. So, if we combine them 0.1 with any of these are nodes which have probability 0.2 in this case I am going to this one. So, what I get is a new node with probability 0.3 and I deactivate these nodes. So, at this point my activate node is this one with probability 0.3, this one with probability 0.2, this one with probability 0.2 and this one with probability 0.3

So, if we combine these 2 which has the least probability 0.2 0.2 and get a new node which is probability 0.4 I deactivate these nodes. So, my activate nodes here is this one with probability 0.3, this one with probability 0.4 and this one with probability 0.3. So, they will combine these 2 nodes, we will get a new node with probability 0.6 and I have another node with probability 0.4 I join them and create my root node which has probability one. Now if is look at the codeword's here again we labeled each of these

branches by 0 and 1 - 0 1 0 1 0 1 0 1 0 1. So, you can see codeword corresponding to this is 1 0, codeword corresponding to this is 0 0, this one codeword is 0 1. So, similarly you can find out this one has code word 1100 1101 1110 and 1111. So, you can see I have 3 codeword's of length 2 and I have 4 codeword's of length 4.

So, if I write the ordered list of codeword's in length of the codeword you can see that this particular optimal Huffman code is ordered list is different from what we got in the earlier two cases. So, this prefix free code optimal prefix free code is fundamentally different from what we have got in the previous examples. Another way in which I can combine these somewhere with this probability and look at this, again start up with the in these probable these.

(Refer Slide Time: 17:00)



See this has probability 0.5, this has probability 0.5, I combine them get new node with probability 0.1 I deactivate these 2 nodes. So, this one, this one and this one, do you have probability 0.1 I can combine any 2 of them let us say combine these 2.

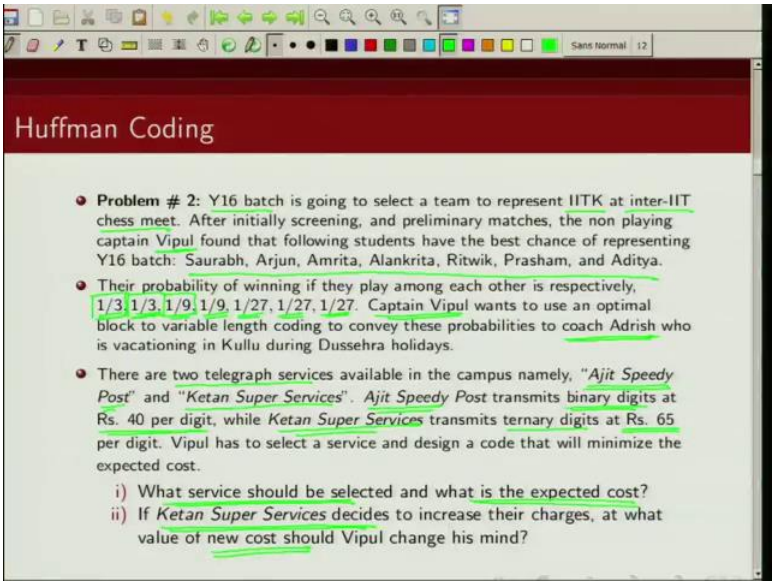
So, I deactivate these 2 nodes and create a new node with probability 0.2. So, what are my active nodes at this point? I have this one with probability 0.2, this one with probability 0.1, and this one with probability 0.2, this one probability 0.2, and this one

with probability 0.3. So, I need to combine this particular node with any node which has probability, any node on this which has probability point 2. So, I combine if we combine this with these I create a new node with probability 0.3 I deactivate these 2 nodes. So, at this point then my active nodes is this one with probability 0.3, this lead with probability 0.2 this one with probability 0.2 and this one with probability 0.3. So, I combine these 2 together because these are the 2 least likely active nodes and create a new node which has probability 0.4 I deactivate these 2 nodes. So, now, I have in active leaf with probability 0.3 and activate node with probability 0.4 and activate node with probability 0.3.

So, I will combine these 2 which of probability 0.3, create a new node which is probability 0.6 and finally, a combine my activate nodes. So, I deactivate this and this. Finally, my active nodes are this one and this one. So, I combine these 2 and what I get is a route node with probability one. And finally, as I said I am going to assign 0s and 1s to these branches 0 1 0 1 0 1, I can see now codeword's for this is 10, codeword's for this particular symbol is 00, codeword for this is 01, codeword for this is 110 codeword for this is 1110, codeword for this is 11110 and finally, this one is 11111.

So, now I have 3 codeword's of length 2 a one codeword of length 3 I have one codeword of length 4 and I have 2 codeword's of length 5. So, this is also fundamentally different optimal prefix free code. Please note these all the different ways in which we have combining these nodes we are essentially following the rules of how to create this optimal binary prefix free code. So, you can see from this example - there a fundamentally 3 different ways in which you can have the ordered list of codeword's then 3 for code 3, 1 and 2 we have this. This particular ordered list for code 3 3 we had this and for the code 3 4 we had this one. So, you can see for a source which emits this (Refer Time: 21:38) random variable with these probabilities there are fundamentally 3 different ways in which we could create or you can construct optimal binary prefix free code.

(Refer Slide Time: 21:58)



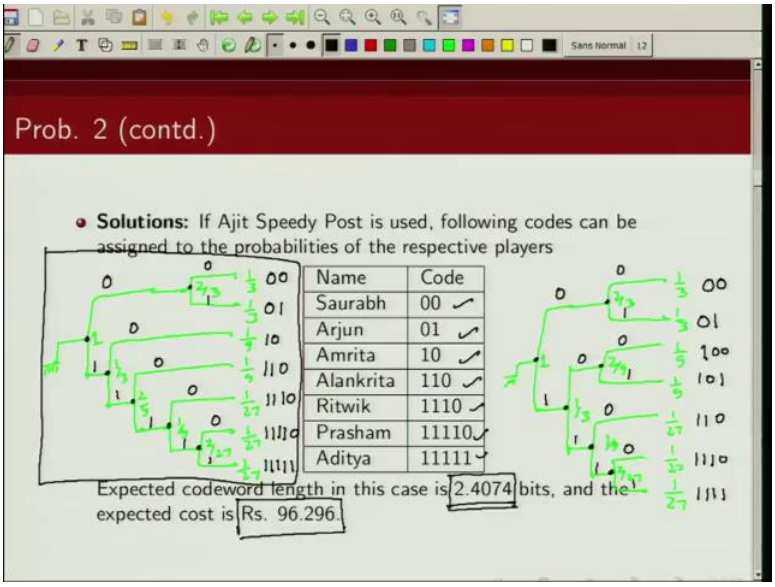
Now, let us move to a next example. So, these are also problem related to Huffman coding. So, you have Y16 batch which is going to represent IIT, Kanpur in Inter-IIT chess meet and for the captain Vipul found out that these other best student in terms of representing the IIT team. He organizes the chess competition where they played against each other and he found out that these are the corresponding probabilities of their winning against to each there. So, around one-third of his match Arjun won, one-third of his match Amritha won, one-ninth of our matches. So, these are the probabilities of they are winning the match when they played against each other.

Now, Vipul wants to convey this information to his coach Adrish who is vacationing in Kollur. So, he has to transmit this information to his coach. So, he has to communicate essentially that these are the probabilities of winning of each of these team members, he wants to convey these probabilities. So, essentially we have a random variable which has 7 possible values. So, I mean it 2 or them can take or. So, he wants to communicate. So, u 1 happens with probability 1 by 3, u 2 have with probability 1 by 3, u 3 u 4 happens with probability 1 by 9 and u 5, u 6, u 7, happens with probability 1 by 27. Now he wants to encode this information within and optimal block to variable length coding. Now there are 2 telegraph services in the campus one is known as Ajit Speedy Post and other is Ketan Super Services.

Now, Ajit speedy post this is transmit's binary bits and it charges 40 rupees for binary bit whereas Ketan super services charges 65 rupees per digit. Now what Vipul has to do is he has to select one of these telegraph services, he has to either select Ajit's speedy post which you this it is sends binary bits and charges 40 bits, 40 rupees per bit or he has to choose Ketan super services which transmit's ternary bits and charges 65 rupees per ternary bit. So, what service should Vipul use to communicate this information to it is coach Adrish and what is the expected course. So, that is the first question and second question is if Ketan super services decide to increase his charges at what new rate people will change his mind.

So, this is the problem occurs. So, clearly one service uses binary bits other uses ternary bits. So, Vipul first has to create a Huffman because he is using block to length variable coding to encode these probabilities. So, in first case he has to construct a binary Huffman code and the second case he needs to construct a ternary Huffman code, and then he needs to find out what is the expected codeword length in these 2 cases and he has to multiply them by their average cost. So, that would give him the total expected cost.

(Refer Slide Time: 26:02)



So, let us look at first binary Huffman tree. So, we have nodes which have probability 1 by 3, 1 by 3, 1 by 9, 1 by 9, 1 by 27, 1 by 27 and 1 by 27.

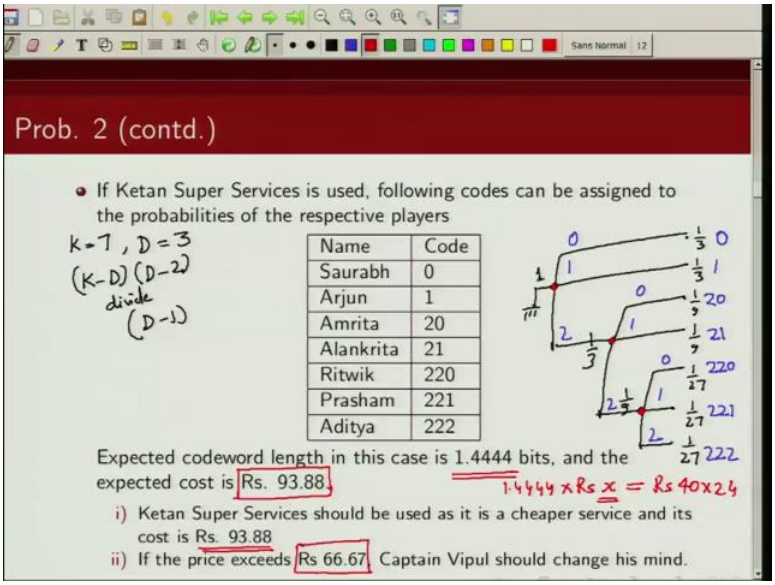
So, first we are going to combine these 2 nodes right and get probability 2 by 27, then next we need to combine these 2 nodes we will get 1 by 9, you can combine let us say these 2 nodes it should be 2 by 9 right and we can combine these 2 nodes which will be (Refer Time: 26:27) 1 by 3 and we can combine these 2 nodes it should be 2 by 3 and finally, we can get this. So, if probability now and if you assign 0s and 1 say 01, if 0 1 0 1 0 1 0 1. So, this codeword corresponding to this will be 00, this will be 0 1, this will be 100, 101, this will be 110, this will be 1110 and this will be 1111. Now we can do the same thing in another way. So, both will essentially get the same expected value let us try doing the same thing in another way. So, let us say we have this 1 3 code at probability one-third, you know these with probability 1 by 9 and then we have 3 more with probability 1 by 27.

So, I start off with these 2 nodes related new node which is probability 2 by 27, when I combine these 2 I get a new node which is probability 1 by 9 you can combine these 2 nodes, but new node if probability 2 by 9 and then you can combine these 2 I get a new node which is probability 1 by 3. You can combine these 2 nodes, but new nodes with probability 2 by 3 and then I can combine these 2 get node root node with probability 1. Now if I assign bits 0s and 1s 0 1 0 1 0 1 0 1 0 1. So, this will have code word 00, this will have 01, this will have 10, this will have 110, this will have 1110, this will be 11110 and this will be 11111.

So, there are multiple ways in which I can create this Huffman free each one of them will get the same expected value you can verify that, what I have written here is actually this one. Now you can find out what the expected codeword length is from path length lemma in the expected codeword length is nothing, but it is the sum of probabilities of all the nodes including the node, using 1 plus 1 by 3 plus 2 by 9 plus 1 by 9 plus 2 by 27 plus 2 by 3 which comes out to be 2.4. We can calculate the same thing from here you will get the same answer you can verify 1 plus 1 by 3 plus 1 by 9 plus 2 by 27 plus 2 by 9 plus 2 by 3 you get the same.

So, the expected codeword length if he use a binary Huffman code it is going to be 2 0.4 bits. Now what was the cost for Ajiths speedy post it was 40 rupees for binary bit. So, if we multiply by 40 you are going to go the expected cost if we use Ajiths speedy post. So, the cost involved if Vipul uses Ajith speedy post is 96.3 rupees.

(Refer Slide Time: 31:20)



Now, let us look at Ketan super services. Now Ketan super services uses ternary bits right and we have (Refer Time: 31:28) which takes value 1 by 3, 1 by 9 and 1 by 27. Now for k in this case is 7, what is d? It is a ternary course, so d is 3. So, first thing we need to find out at the initial stage how many unused (Refer Time: 31:55) are there and how do we find out that out k minus d into d minus 2, if we divide this by d minus 1 whatever remained that is remaining. So, k minus d in this case is 4 and d minus 2 is 1. So, 4 divided by 2 - remainder is 0.

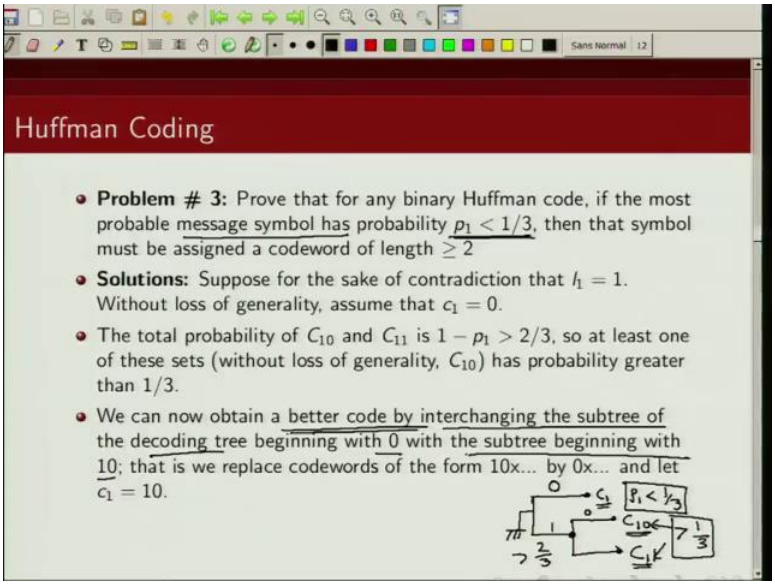
So, there are low unused leaves if we use this particular source and encoded within ternary Huffman code. So, then in the initial stage we are going to combine 3 of these leaves. So, these are the 3 leaf likely node leaves. So, we combine them and this will have probability equal to 1 by 27 plus 1 by 27 plus 1 by 27. So, this is probability is 1 by 9. Next we combine this node, this node and this node and their probability 1 by 9, 1 by 9 plus 1 by 9 that is 1 by 3 and finally, we combine these 3 nodes this is this is root this is

probability one and if you assign 0 1 2 to these branches, I get the code corresponding to this is 0, code corresponding to this is 1, code corresponding to this is 20, code corresponding to this is 20, code corresponding to this 220, 221 and 222.

So, again I can find out the expected code where length using path length lemma. So, this is sum of their probabilities of all the nodes including the root node - this will be 1 plus 1 by 3 plus 1 by 9 and that comes out to be 1.44 bits. Now, Ketan super services charges 65 rupees per ternary bit. So, if I multiply 65 into this expected code was length, the cost that comes out is 93.88, and what was the cost for Ajith speedy post? It was 96, so that is more. So, Ketan super services is Vipul super services is cheaper. So, Vipul is going to use Ketan super services. And next part of the question was by what price if he increases Ketan his charges at what point Vipul will change his decision.

So, that we can calculate, we know the expected codeword length for ternary which is 1.4444 into cost let us say cost is rupees x that and the cost of binary transmission is 40 and number of bits where 2.4. So, if you from here you can find out what is the break even cost if Ketan increases the cost beyond this point then (Refer Time: 35:28) is going to change a decision and that price comes out to be 66.67. So, if Ketan increases cost of his ternary transmission of ternary bits more than 66.67 per ternary bit then Vipul is going to change his decision and will go for Ajith's speed speedy post. Now, next we are going to prove some result of a binary Huffman code.

(Refer Slide Time: 35:56)



So, prove that if you have a binary Huffman code and it is most probable symbol has probability which is given by less than 1 by 3 then this symbol must be sign a codeword length of at least 2. See if the most likely symbol has probability less than 1 by 3 then you must assign if this most probable symbol at least a codeword of length 2. Now how do we prove this result? So we will use the method of contradiction. So, let us say that we are assigning this most probable symbol which has probability less than one-third, we will assume let us say we are assigning it a codeword of length 1 and then we will show that this is not possible. This particular message symbol which has probability most of the symbol which has probability less than 1 by 3, we will show that it is not possible for it to be assign a codeword length of one.

Hence, we will prove that it requires at least a codeword of length 2. So, we start up with. So, let us assume that c 1 is this message symbol which has probability given by p 1 which is less than one-third and without loss of generality we are assuming. So, we assuming that that this has a codeword of length 1 and without loss of generality we are assuming that codeword is 0. So, what we have seen is a situation like this. So, we have a binary tree and this message symbol is assigned a codeword c which is basically 0 that is what we are saying and this has probability p 1 which is less than 1 by 3. Now if this is

so, then let us draw this tree again. So, this is corresponding to 0 which is c 1, which is probability p 1 which is less than 1 by 3.

So, then if you look at what is a probability of this node, if this has probability less than one-third then this particular node has probability greater than 2 by 3 and how was this node formed, this node must have been formed using some other they are combining some other nodes. So, if this has probability greater than 2 by 3 and there are 2 nodes or least which are merging here. So, at least one of them then must have probability greater than 1 by 3, if this particular node has probability greater than 2 by 3 then at least I am calling this has c 10, I am calling it c 11. So, at least one of them either c 10 or c 11 has probability greater than one-third, correct. Now if that is true then we can obtain a better code by interchanging the subtree of the decoding tree beginning with 0 with the subtree beginning with 10 and what are we seeing, we have a situation like this we are saying it is the most likely symbol which is been assigned a codeword of length one and this has probability p 1 less than 1 by 3. But we just now found out if this thing has to hold this particular node should have probability greater than 2 by 3.

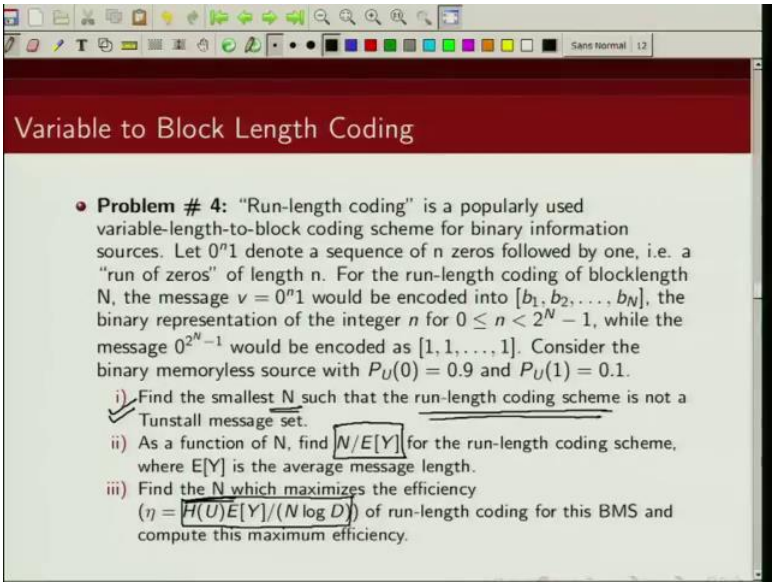
In other words this particular code c 10 or c 11 either of them should have probability greater than 1 by 3. If that is a situation when we can reduce prospected codeword length by replacing by inter changing c 1 with c 10 or c 11 whichever has probability greater than 1 by 3. So, so that is what I am saying that we can obtain a better code by interchanging the subtree beginning with 0 with the subtree beginning with 10 why because, one of these nodes have probability which is greater than one third, whereas this one has probability less than one third

So, if you do this interchanging we are reducing the expected codeword length, because either this or this has probability 1 by 3 if you assign them codeword length of 1 instead of 2 and we assign this codeword length of 2 instead of 1 then expected codeword length is going to decrease, because this c 1 has less probability than one of these. So, what we have shown then is this assumption that this message symbol which is the most problematic message symbol is it has probability greater than one third, then we cannot assign it codeword of length one. Because we just now saw that if that is a situation we can actually get a better code by interchanging c 1 with either c 10 or c 11. Hence, we

have proved that this symbol which is most probable symbol which has probability less than one third must be assigned a codeword length of at least 2 for a binary Huffman code.

So, as I said this improvement contradicts the assumption that the most probable message symbol should have probability should have codeword length to one. And hence we have shown that this most probable message symbol which has probability less than 1 by 3 must have a codeword of length at least 2, it cannot have codeword length of 1.

(Refer Slide Time: 43:28)

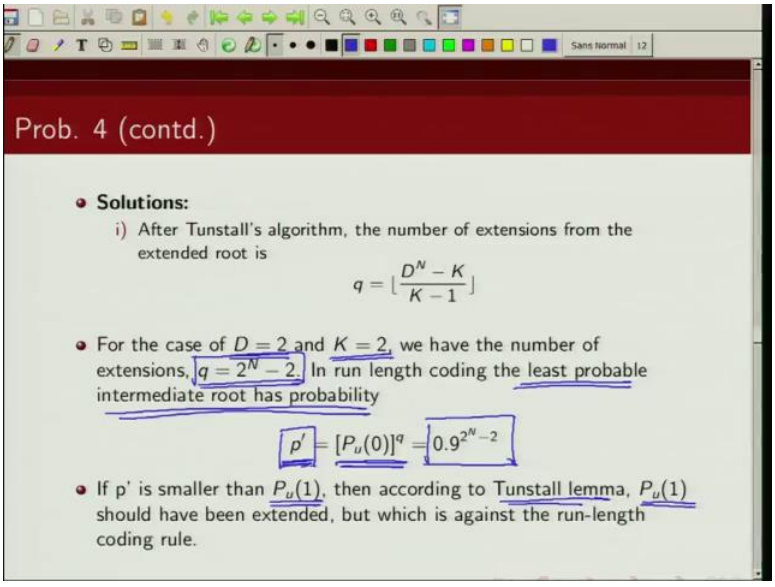


Now next example is on variable to block length coding. Now there is a variable to block length coding called Run-length coding. Now what does run-length coding does? So, let 0 raise power n 1 denotes a sequence of n zero followed by a single one. So, this is a run of a 0s of length n followed by 1 that is it is sequence and that is how we encode, so this run-length coding works has follows. So, for run-length coding of a block length of n the message which is a run of n zeros is encoded into a codeword of length n and which is essentially of binary representation of the integer n for when n lies between 0 2 to raise power n minus 1, and all 0 sequence encoded has all ones.

So, that is how a run-length coding works. So, any sequence of the form this is encoded into a sequence of length n which is basically binary representation where n lies between this and if you have a run of all 0s this is encoded as sequence of 1 of length n. Let us consider we have a source which emits 0s with probability 0.9, and which emit 1 with probability 1. So, we are considering a discrete memoryless source on a binary source. So, it is emits 0s and ones my question is as follows. Find the smallest n such that run-length coding scheme is not a Tunstall message set. We need find the smallest n such that this run-length coding which we described here is not a Tunstall message set.

The next question is at the function of this n find this ratio of output codeword length which is n divided by input expected code word length of input with expected value of y. Find out this ratio at the function of n and the third part is find out the value of n which will maximize this efficiency. In other words find out the value of n which will minimize this. So, the first part is found out the smallest n such that the run-length coding scheme is not a Tunstall message set.

(Refer Slide Time: 46:53)



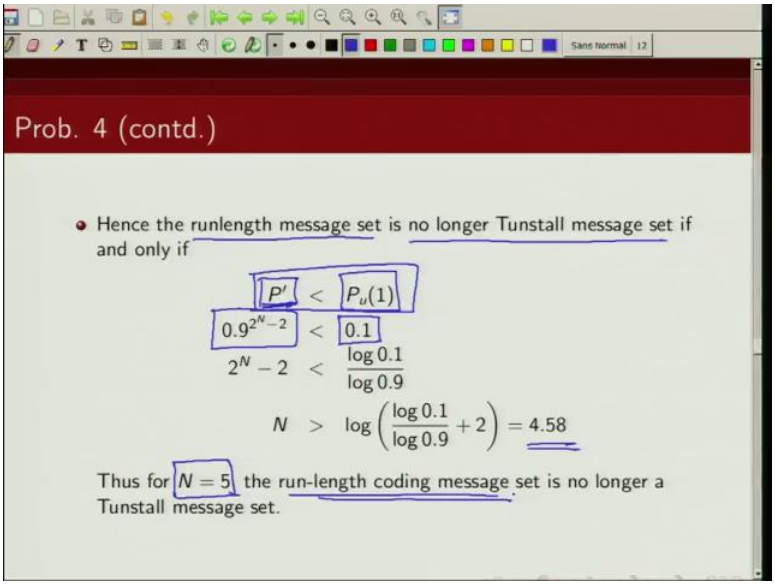
In a Tunstall message set what do we do? We do q extensions of the extended root and each time we extend the most lightly leaf that is your Tunstall message set. And how many set extensions do we do? If you recall the number of extensions that we do is given

by this expression. So, (Refer Time: 47:22) of d raise power n minus k divided by k minus 1, because when we are extending a leaf we are creating from the extended root we are creating k new leaves but one leaf becomes a node now, so at each time we are actually adding k minus 1 new leaves.

So, number of extensions that we would require to create a Tunstall message set is q which is given by this expression. Now in this example we are considering binary codes, so k is 2 and d is 2 and our input x 0s and ones, so k is also 2. In this case the number of extensions comes out to be 2 raise power n minus 2. So, in run-length coding what we are doing, we are doing basically 10 runs of 0 followed by 1. So, in the run-length coding the least probable intermediate note will have probability p dash given by probability of 0 raise power q, because in a run-length coding we are having n n runs of 0s.

So, if we want to find out whether this run-length coding belongs to a Tunstall message set then every time we are having a run of 0 that particular node should be the most lightly leaf. So, if we are doing q extensions the intermediate node of probability p dash which is given by this probability and probability of 0 is given by 0.9 q we just calculated is given by this expression, so this probability is given by this. Now for it to be a run-length coding every time basically we are extending run-length coding every time we are having runs of n zeros. So, this probability p dash if this probability is smaller than p u then according to Tunstall lemma we should have extended p of u I u 1, if this intermediate probability p dash becomes less than probability of u B 1 then according to Tunstall message set construction we should have extended p of u 1. But then that would have been against the rules of run-length coding, because in run-length coding we extend 0s we have runs of 0s and n runs of 0s.

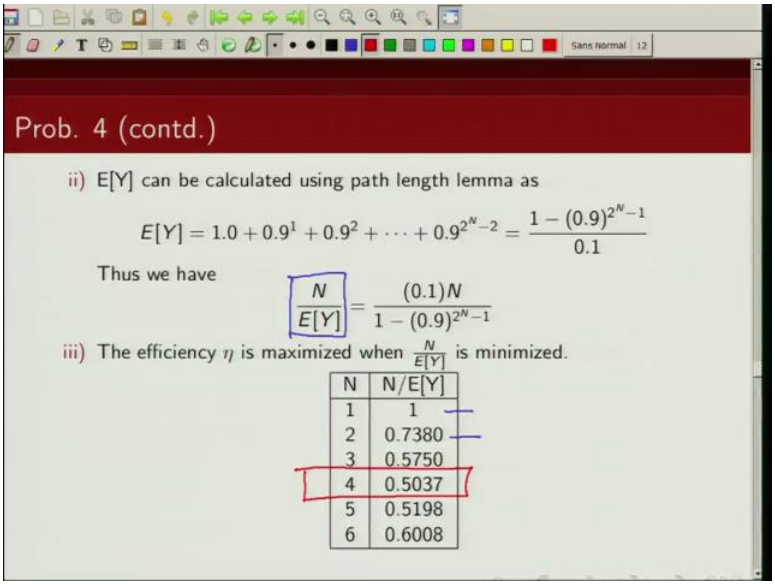
(Refer Slide Time: 50:37)



So, then run-length message set is no longer going to get Tunstall message set if this intermediate probability becomes less than probability of u equal to 1, because if this intermediate probability becomes less than probability of u equal to 1 then according to Tunstall lemma we should have extended the node or leaf corresponding to u equal to 1. So, if the intermediate probability becomes less than probability of u equal to 1 in that case our run-length message set is no longer going to be a Tunstall message set. And what is this intermediate probability, that intermediate probability is given by this we have just calculated and probability of 1 is given by this.

So, after simplification what we get is n should be greater than this. That means, if n is greater than 5 then run-length coding is no longer going to be a Tunstall message set. So, for n less than 5 n less than n 4 3 2 those for those values of n run-length coding for this particular source is going to be Tunstall message set, but for n greater than equal to 5 this for this particular source the run-length message set is not going to be a Tunstall message set.

(Refer Slide Time: 52:2)



Now the next part of the question was to calculate the ratio of the output codeword length to the input expected codeword length. Now how do we find out the input expected codeword length, we can invoke the path length lemma to find out the expected value of y; we know that this is nothing but some of probability of all nodes including the root node. So, root node has probability. This now next node because you are in run-length coding you are extending the node corresponding to branch corresponding to u equal to 0. If I draw the run-length coding it would be like this.

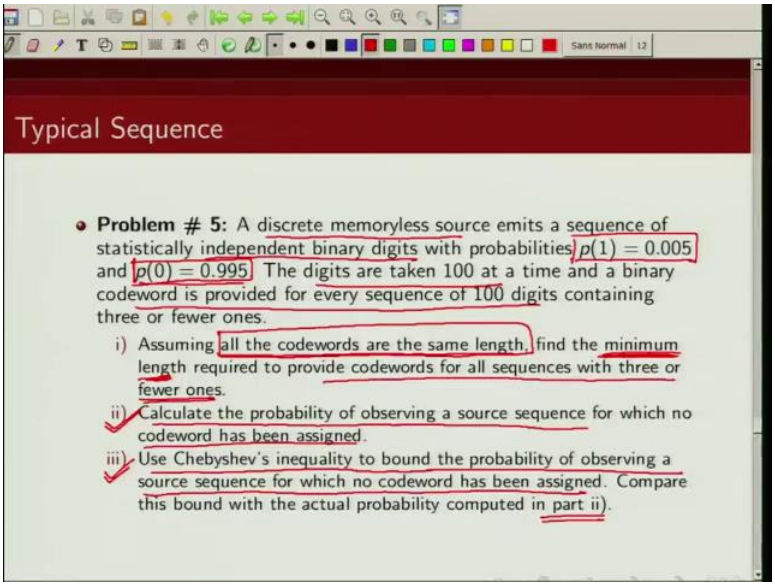
So, initially you have this is 0 and 1 this is probability 1 this is probability 0.9 this is probability 0.1. Then again you are extending this, so this is probability 0.9 into 0.9 square, then again you extend this will be 0.9 cubes. This you will do until the q extensions of this leaf containing the 0. So, the expected codeword length in case of this run-length coding can be written like this and this is equal to this. Now the expected this ratio of n divided by expected of value y comes out to be this.

And the third part of the question says what is the value of n for which we get maximum efficiency, and of course you will get maximum efficiency when this this ratio smaller because you want to compress large block of data into, remember we are doing variable

to block length coding so the output block length is fixed. So, we will get more compression if we are able to combine larger blocks of data into this block of length n.

So, we evaluate this ratio for various values of n I have written it here for n equal to 1, this is one for n equal to 2 this comes out to be this. So, you can see this where ratio is smallest for this quantity. In other words we can say that for n equal to 4 this expected ratio is of n divide by expected value y is minimized. In other words efficiency is maximum for n equal to 4.

(Refer Slide Time: 54:48)



So, let us look at the next problem. We have a discrete memory less source that emits statistically independent binary bits with probability of 1 given by 0.005 and probability of 0 given by 0.995, so most of the time it is submitting 0s. The digits are taken 100 at a time and a binary codeword is provided for every sequence of 100 bits. We have a binary discrete memory less source which is emitting 0s and ones; 0s would this probability and 1 with this probability. We are taking 100 bits at a time and we are assigning a codeword.

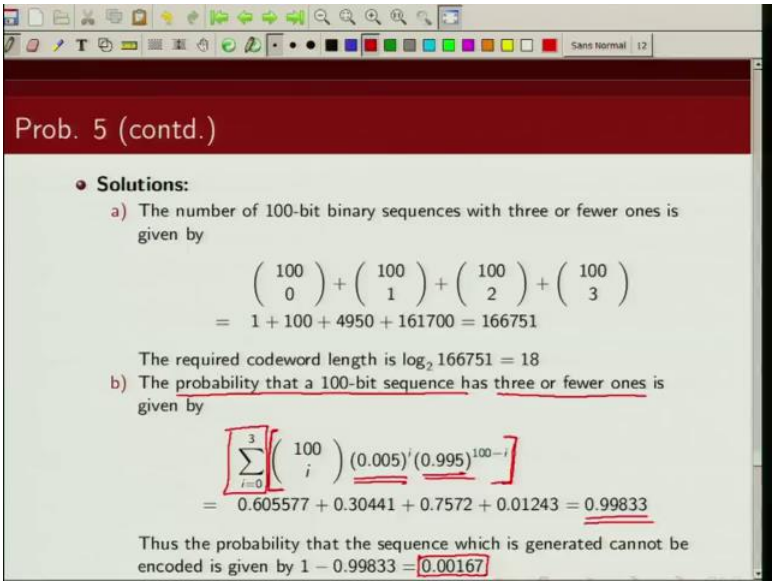
Now, assuming all codeword's are of same length find the minimum length required to provide codeword's for all sequences containing 3 or fewer ones. So, clearly you can see that most of the most typical sequences which come out of this source are going to be the

ones which contains all 0 sequences. So, we are providing codeword's to all sequences which have length number of ones 3 or less. So, the question asked what is the minimum length required to provide codeword's to all sequences which has 3 or less 1. We are assuming that all codeword's are of same length. So, that is a first part of question.

The second says calculate the probability of observing a source sequence for which no codeword has been assigned. Now, when will no codeword be assigned? When we get among these 100 bits number of one's which is 4 or more, in that case we are not assigning any codeword's. So, the second part of the question asks us to calculate this probability. And the third of the questions is use Chebyshev's inequality to bound the probability of observing a source sequence for which no codeword has been assigned.

So, in part B we are going to compute that exact probability, in part 3 we are going to use Chebyshev's inequality to get and upper bound and that probability and we will compare that probability with the actual probability that you will computed in part 2.

(Refer Slide Time: 58:15)



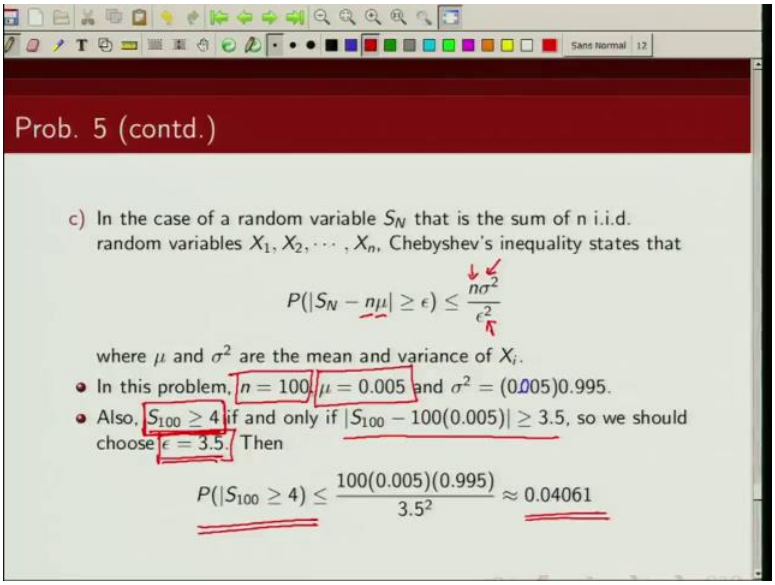
So, remember we are considering these binary sequence 100 bits at a time. So, what is the number such sequences which has 3 or fewer ones. This is number of sequences which have all 0s, this number of sequences which have 1 1 this is number of sequences

of length 100 which has 2 ones. These are number of sequence of length 100 which has 3 ones, so this total comes out to be this. Then the required codeword length and this case are going to be 18 log 2 base to of number of such codeword's. We can represent all sequences of length 100 which is coming out of these this discrete memoryless source using a codeword of length 18, assuming we are assigning codeword's to all such sequences which has 3 or less ones and all codeword's are of same length.

Now, when are we going to assign a codeword to a sequence of 100 bits? Whenever we get a sequence which has either 0 ones or 1 1, 2 1 or 3 1, so let us compute the probability that the 100 bits sequence has 3 or fewer ones. This is a probability of occurrence of 1 this is the probability of occurrence of 0. So, this will give us to a probability that I errors have or like if this is a probability that I number of ones are there in the sequence of 100 bits. So, this will give us the probability that there are I ones in the sequence of 100 bits. Since, we are interested in finding the probability that there are 3 or fewer ones we sum it over all is going from 0 to 3 and this probability comes out be 0.99833.

In other words probability that we were not going to assign any codeword to a sequence of these 100 bits is given by 1 minus this and this probability comes out to be 0.00167. So, this is roughly has a order of 10 to minus 3. That is a probability that we are not going to assign a codeword to a 100 bits sequence, we call that we are assigning codeword to all 100 bits sequence which has all 0s or at least 1 1, 2 1 or 3 ones.

(Refer Slide Time: 61:31)



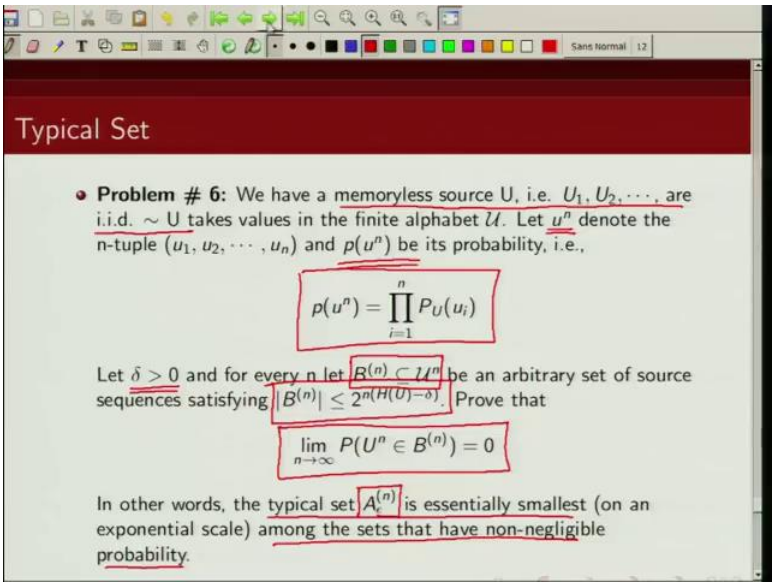
Next we are going to use Chebyshev's inequality which basically says that. If S N is the sum of n i i d random variables in probability that we have 2 difference between sum and n times mu if greater than equal to epsilon is given by n times sigma square by epsilon square 2 n mu is the mean of x I and sigma square is the variance of x i.

So, in this problem our n is 100 mu is 0.005 and sigma square is 0.005 into 0.995. We are interested in knowing that this sum is basic sum of ones is basically a greater than equal to 4, so we choose our epsilon in such a way such that our sum of these 100 bits comes out to be greater than equal to 4. So, for the given value of n and mu our epsilon should be chosen as 3.5 then only we will get this conditioned has sum of this 100 bits is has more than 3 ones.

So, in this particular case or epsilon should be chosen as 3.5, for the given n which is this and number of mean of this 0 is basically is 0.00, this mu is this is mean of this number of 1s is 0.005. So, epsilon if we choose with 3.5 we get a condition that sum of these 100 bits has more than 3 ones. So, if we plug in the value of n sigma square and epsilon which is 3.5 in here we get the condition that some contains 4 or more ones that probability comes out to be upper bounded by point 0.04061. Please note that this is actually quite a loose bound because exact probability comes out to be 1.67 10 to minus

3. Whereas, upper bound is 4 into 10 to power minus 2. So, this upper bound is quite loose.

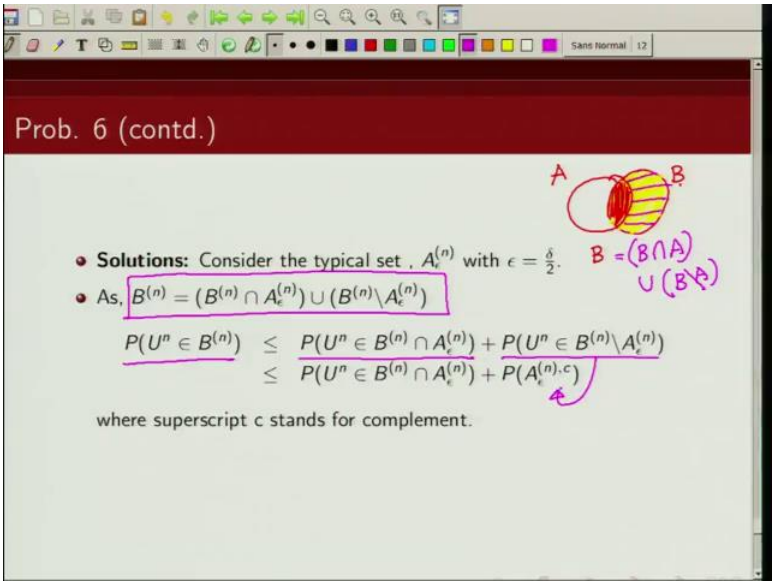
(Refer Slide Time: 64:18)



Finally, we are going to show you that a typical set is essentially the smallest among all the sets that have non-negligible probability. So, let us consider a memory less source u which is uniting is random variable u is and these are i i d distributed. Let U n denotes an n tuple; u 1, u 2, u 3, U n that is an n tuple that is denoted by U n and it is probability is given by p of u n.

And since it is a discrete memory less source you can write this probability of U n as probability of u i and product of over i going from 1 to n. Now let delta B greater than 0 and B is subset of this U n where this sequence satisfy this condition that a cardinality of B n is less than equal to 2 raise power n into h of u minus delta. Then we one to show that as n tends to infinity probability that U n is an B n is basically goes to 0, which essentially will show that typical set is essentially the smallest set which will have non-negligible probability of occurrence when n asymptotically when n goes to infinity.

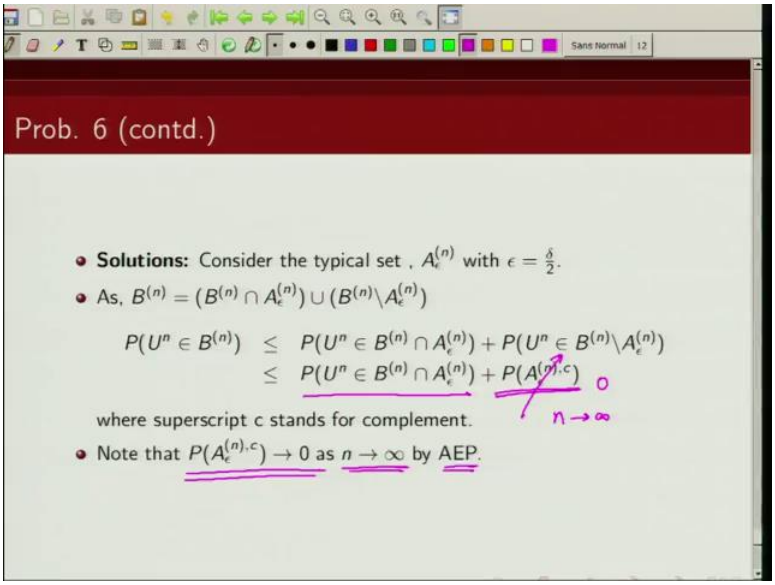
(Refer Slide Time: 66:03)



So, let us prove this. We consider a typical set with epsilon we choose as delta by 2.Now let us take let us say you of 2 sets A and B; so 2 sets A and B. Now if I ask you to write B you can write B as; what is B? B is just this set. Now what is B? B is this portions this portion plus this portion.

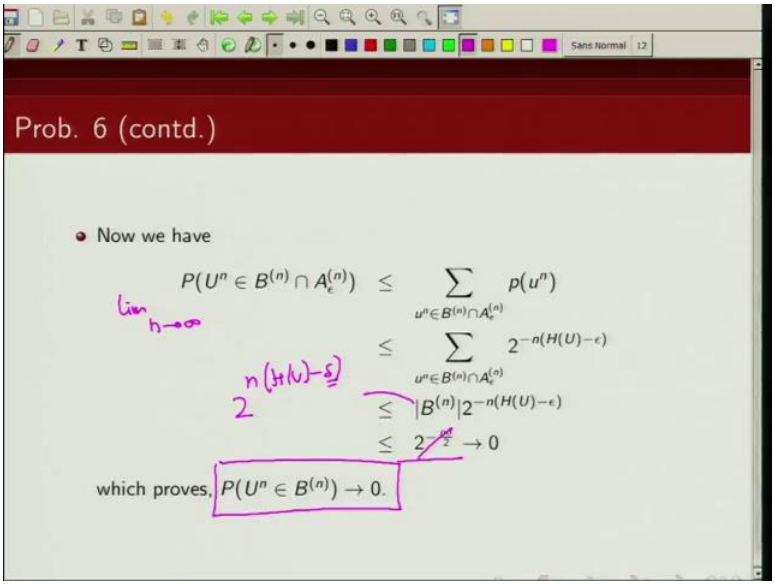
So, I can write B as B intersection A union B minus this set A, so that is what I am writing here. I am writing this B n as B n intersection this typical set union B minus this typical set. So, probability that U n belongs to B n can be then written as probability that U n belongs to intersection of B n and it is typical set and probability that U n belongs to B n minus this typical set. So, this is nothing but compliment of typical set.

(Refer Slide Time: 67:52)



Now we know that sum the property of asymptotic equipartition property we know that as n goes to infinity this probability is negligible. This was very very small some epsilon it is goes to 0. As n goes to infinity this probability basically will go to 0, as n goes to infinity. So, limit when n goes to infinity would be then upper bounded by limit when n goes to infinity this probability.

(Refer Slide Time: 68:24)



Now what is this probability? This probability can be upper bounded by summation of this p of U n where U n is n intersection of B n and this typical set. Now, this can be written as now cardinality of B n is just a second, so this one if you go back this p n is is less than 2 raise power n h of u minus delta. So, the probability can be basically upper bounded by 2 raise power minus n h of u minus epsilon and summing over this particular set, now when I summing over this particular set I can write this as more than equal to cardinality of this set multiplied by this probability.

Now this probability if you recall this from the properties of typical sequences we have we know what is the probability of a typical sequence that is upper bounded by this quantity that is what from here to here you get this relation from the property of typical sequence that probability of U n when u basically belongs to a typical sequence is upper bounded by 2 raise power minus n h of u minus epsilon. And this we are summing over this particular and this particular set, so this would be cardinality of B n multiplied by this.

And what is B n? B n we just now saw it is given by 2 raise power n h of u minus delta. So, if I plug in this value of B n which is 2 raise power n h of u minus delta and plug in the value of delta which we chose has delta we chose has 2 times epsilon, so epsilon we chose has delta by 2. So, if I do that then what I get here is 2 raise power minus n into delta by 2. Now if I take limit now n goes to infinity this term will essentially go to 0, so which essentially proves that typical set is the smallest set which will have a non-zero probability.

With this we will conclude this lecture.

Thank you.