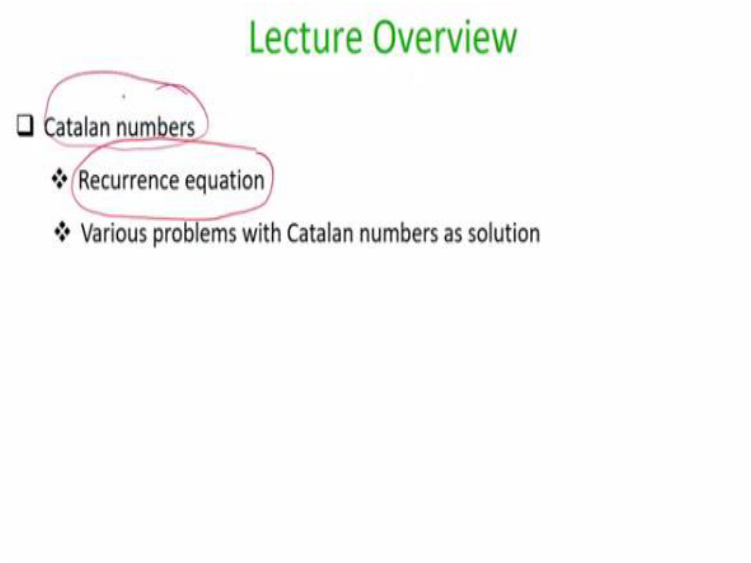
Discrete Mathematics
Prof. Ashish Choudhury
Department of Mathematics and Statistics
International Institute of Information Technology, Bangalore

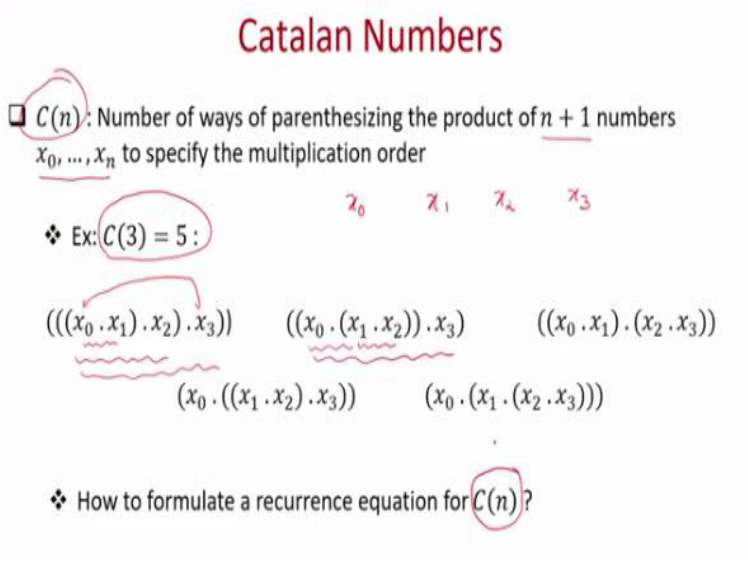
Lecture -40
Catalan Numbers

(Refer Slide Time: 00:22)



Hello everyone. Welcome to this lecture. So, we will continue our discussion regarding how to count number of things for certain problems by formulating recurrence equations. And in today's lecture we will discuss a very interesting class of problems whose solution is a common recurrence equation and the resultant solution for that corresponding recurrence equation gives rise to a sequence of interesting numbers which we call as Catalan numbers. So, we will also discuss various problems in this lecture whose solution is the Catalan numbers.

(Refer Slide Time: 01:06)



So, let me formulate this problem. So, imagine the function C(n) denotes the number of ways of parenthesizing the product of n + 1 numbers to specify the multiplication order. So, you are given n + 1 numbers, I am denoting them as x_0 to x_n. And I can multiply only two numbers at a time. So, I want to specify the order in which I can multiply them. However I do not want to shuffle the positions of the numbers x_0 to x_n; they should be in the same order.

That means x_0 should be there at the first position second position x_1 should be there third position x_2 should be there and so on. I am not allowed to shuffle them, because even though the multiplication is commutative and associative, I am not allowed to invoke those rules here. But I am interested to keep x_0 to x_n at their respective positions and then want to find out the number of ways in which I can parenthesize them to specify the multiplication order.

So, for instance C(3) is equal to 5, because C(3) means I have 4 numbers. So, I stress here if I am considering the problem instance where there are n + 1 number then the resultant number of ways of parenthesizing is C(n), not C(n + 1). So, I have the number x_0, x_1, x_2 and x_3 and these are the five ways of parenthesizing them to specify the multiplication order. So, the first ordering should be interpreted as follows.

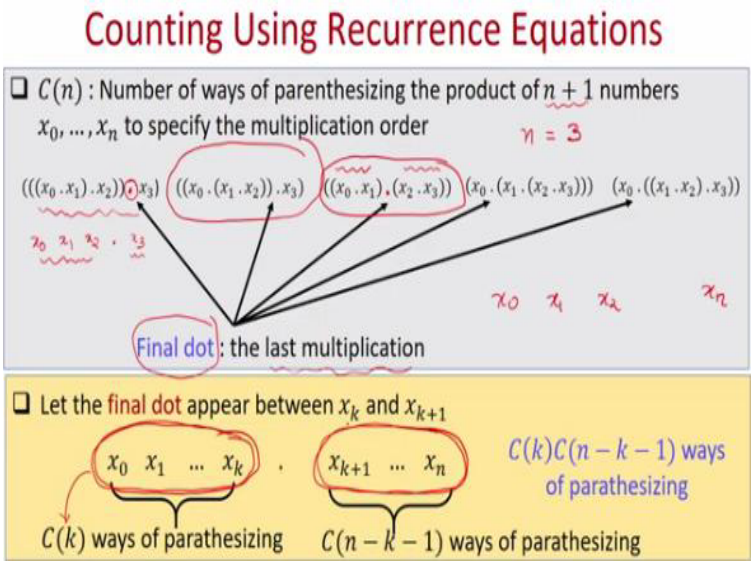
I multiply first x_0 with x_1 then the resultant product is multiplied with x_2 and then the resultant product is multiplied with x_3. As I said I am not allowed to swap positions that means I cannot

say that let me first multiply x_0 and x_3 and then multiply it with x_1 and so on that is not allowed. In this second way of parenthesizing the sequencing is as follows. I am first asking to multiply x_1 and x_2 and then the resultant product is multiplied with x_0.

So, you see I am not changing the position of x_0. And then the resultant product is multiplied with x3 and so on. So, there are five ways if I had four numbers and that is why C(3) is equal to 5. So, now we want to formulate a recurrence equation for C(n) because I want to find out finally a closed form solution for C(n). So, C(n) is an infinite sequence or it is a function which gives you the outcome for different values of n.

So, I want to find out the general formula or the closed form formula for this sequence or this function C(n). How do I do that? For that I first have to formulate a recurrence equation.

(Refer Slide Time: 04:06)



So, let us proceed toward formulating the recurrence equation and I am retaining here the case where I had four numbers and the five ways of parenthesizing them. So, as a general principle whenever we want to formulate a recurrence equation we have to think of how to break a problem instance or the nth problem instance into problem instances of smaller size. And then try to see the relationship.

That is the general philosophy of formulating recurrence equation. So, same thing we will do here and to break a problem instance where we are given a task of sequencing n + 1 number into task of sequencing a sequence of smaller numbers or less number of values, we have to do the following. I focus here on the final dot, namely the last multiplication which needs to be done in any sequencing for this sequence of n + 1 numbers.

So, for instance if my n is equal to 3 here, then in this sequencing it is the final dot is appearing here. So, I will be focusing on this final dot, so I can interpret this sequencing as follows. I can say that my problem instance is divided into two problem instances. Namely I have to figure out how to sequence or how to specify the parenthesizing order for three values namely x_0, x_1, x_2 and another value consisting of x_3.

Once I know the number of ways of parenthesizing x_0, x_1, x_2 and individually I know how to parenthesize or the number of ways of parenthesizing x_3 or the remaining portion of the sequence. If I put a final dot then that gives me a way of parenthesizing the overall sequence. In the same way if I consider this second sequencing here, see I am doing the same thing. In the same way, if I consider the third sequencing I am imagining as if I have two values here and whatever is the number of ways of parenthesizing two values and whatever is the way or whatever is the number of ways of parenthesizing the two values here namely x_2 and x_3, if I put a final dot in between them that will give me the total number of ways of parenthesizing four values and so on. So, the idea here is that in order to formulate the recurrence equation I should focus on the final dot or the last multiplication.

So, in general I can interpret as if the final dot is appearing between x_k and $x_{(k+1)}$. So, remember your n + 1 numbers are x_0, x_1, x_2 and x_n. So, you have n + 1 numbers so you can interpret that in any parenthesizing order if the final dot is appearing between x_k and $x_{(k+1)}$. Then what you can say is that you have the bigger sequence consisting of n + 1 numbers is now divided into two individual sequences.

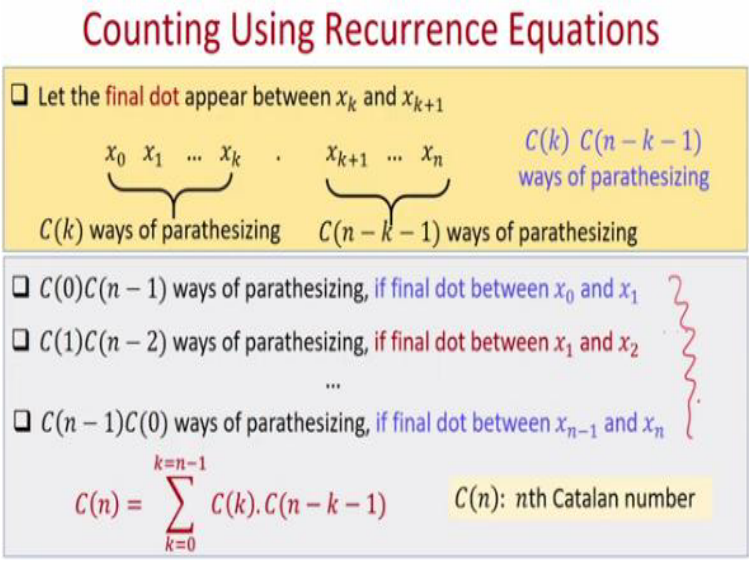
And they are disjoint; the sequence to the left hand side of the final dot and the sequence to the right hand side of the final dot. The sequence to the left hand side of the final dot has k + 1

values. So, what can be the total number of ways of parenthesizing that sequence or specifying the multiplication order for those k + 1 values; well as per our problem definition there are a total C(k) ways of parenthesizing those k + 1 values.

And now we are left with remaining values namely after the final dot; they are the values from $x_{(k+1)}$ to x_n and again as per the problem definition there are C(n-k-1) ways of parenthesizing those remaining values which are going to appear after the final dot. Now if you find out the total number of ways of parenthesizing the LHS part before or the part before the final dot and the number of ways of parenthesizing the expression that is there after the final dot.

Then if you multiply them as per the product rule that will give you the total number of ways of parenthesizing the numbers x_0 to x_n provided the final dot appears between x_k and $x_{(k+1)}$. But now my k could be anything, k ranges from 0 to n.

(Refer Slide Time: 08:57)

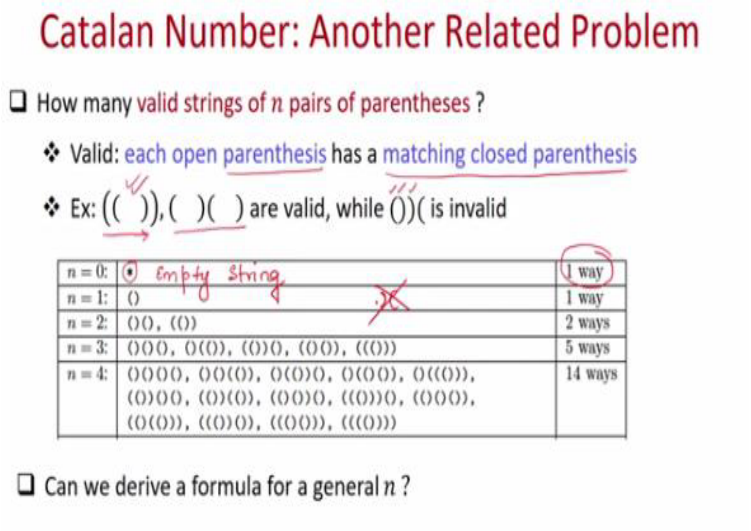


More specifically, if my final dot is between x_0 and x_1, that means I am taking the case where x_1, x_2, x_n. So, whatever way I can parenthesize x_0 and whatever way I can parenthesize the remaining n values and then put the final dot that gives me one category of parenthesizing where the final dot is between x_0 and x_1. Similarly I can have another category of parenthesizing where the final dot is between x_1 and x_2. That means I parenthesize the subsequence x_0, x_1 and I parenthesize the remaining portion namely x_2, x_3 up to x_n.

So, there are C(1) ways of parenthesizing this part, there are C(n – 2) ways of parenthesizing this part. And if I multiply C(1) with C(n – 2) that gives me the total number of ways of parenthesizing where the final dot would have appeared between x_1 and x_2. And now if I continue like that I can argue that if the final dot is between $x_{(n-1)}$ and x_n, then the total ways of parenthesizing will be the product of C(n – 1) and C(0).

So, that is why k ranges from 0 to n - 1 your k could be 0 or 1 or up to n - 1. So, that is why I get the recurrence equation that C(n) is equal to summation of k equal to 0 to n - 1 product of C(k) and C(n - k – 1). And why I am adding all of them, because all these are disjoint categories of parenthesizing. There is no overlap, there is nothing which gets counted twice. I do not have to exclude anything.

(Refer Slide Time: 11:08)



So, this C(n) function is called as the nth Catalan number and it turns out that there are plenty of problems in combinatorics whose general solution is nth Catalan number. So, let us see another related problem. From the problem description it might look similar to the problem of specifying the parenthesizing or multiplication order for n + 1 values. But that is not the case but still the answer is the nth Catalan number.

So, here we want to find out how many valid strings of n pairs of parenthesis we can have. And what is my definition of valid strings of n pairs of parenthesis. Well if I parse that string from left hand side to right hand side. Then it should be the case that each left parenthesis or opening parenthesis should have a corresponding matching closed parenthesis. That is what is my definition of a valid string of n pairs of parenthesis.

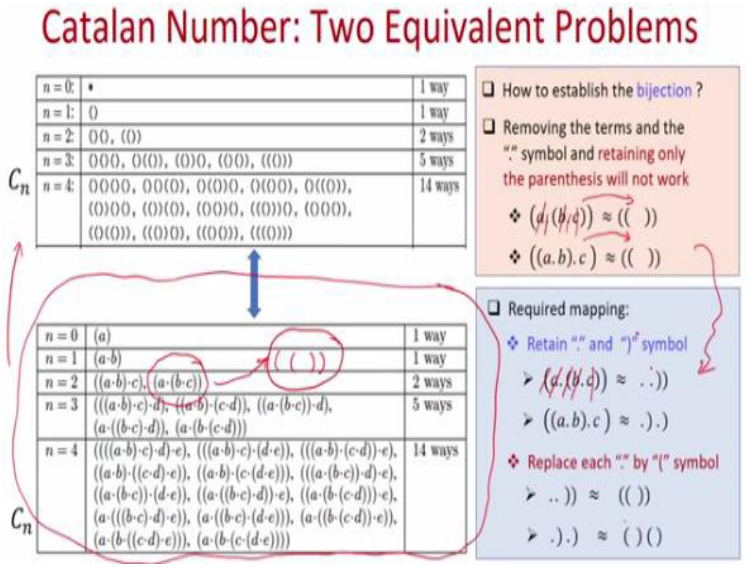
So, for instance this string is valid, because if you scan from left hand side to right hand side then each instance of opening parenthesis has a corresponding matching closing parenthesis. It is not the case that you have an occurrence of closing parenthesis but till that point you do not have an occurrence of a corresponding opening parenthesis. In the same way this is a valid sequence ( ) ( ), but this is invalid ( ))(.

Because you have an opening bracket and then a closing bracket and then followed by a closing bracket. For that you do not have a corresponding opening bracket as of now. So, that is why this is an invalid string. So, we want to find out how many such valid strings of n pairs of parenthesis we can have. So, I have demonstrated here the total number of ways of formulating valid strings of n pairs of parenthesis for different values of n.

If n is equal to 0 then you do not have any string. And so that no string is denoted by an empty string, the star denotes empty string. And since empty string is one of the strings that is why for n equal to 0; I consider that answer is 1. Namely there is only one way of coming up with a valid string consisting of zero pairs of parentheses. That one way is nothing but writing down the null string.

For n equal to 1 you have only one string namely an occurrence of namely a string where you have a left parenthesis followed by a right parenthesis. Whereas a string where you have an occurrence of right parenthesis followed by a left parenthesis this is invalid. Similarly for n equal to 2 you have two strings for n equal to 3 you have five strings and so on. So, now we want to find out or derive a formula for general value of n.

(Refer Slide Time: 14:03)



And it turns out that the general value of n or the number of strings for general n is nothing but the value of the nth Catalan number. To do that, what we can show is the following. We already know that the solution for specifying the multiplication order for n + 1 values is the nth Catalan number. Because that was our starting point, and now we have a new problem. Namely that problem of finding the number of valid strings with n pairs of parentheses.

Instead of trying to find out the solution from scratch for the new problem what we can show is the following. If we show that there is a bijection between the set of each valid parenthesis, each valid way or each valid mechanism of parenthesizing n + 1 numbers and a set of valid strings that you can formulate with n pairs of parenthesis. Then we can say that the solution for both the problems is the Catalan number.

So, how exactly we establish a bijection. So, what we have to show is that you take any sequence of parenthesizing n + 1 values corresponding to that you can formulate or you can find out a valid string with n pairs of parenthesis in an injective fashion and vice versa. If we do that then it establishes a bijection. So, it might look on a very high level that the bijection establishment is very simple.

What you may say is that well you take any sequence or any valid sequence specifying the parenthesizing of n + 1 numbers and if corresponding to this sequence specifying the

parenthesizing order you want to find out the matching valid string with n pairs of parentheses what you can do the following. You can say that you remove all the terms namely forget about $x_0, x_1, \ldots, x_n$ you simply remove them.

Because in our valid string; consisting of n pairs of parentheses we do not have any occurrence of x_0 to x_n. So, you can say that erase x_0 to x_n from your sequencing which specifies the parenthesizing order. And you just retain the parenthesis namely the left parenthesis and the right parenthesis. And the claim is that you obtain a valid string consisting of n pairs of parentheses. If you do that then resultant string will be a valid string of n pairs of parenthesis.

So, for instance what I am saying is you take the sequence, say this sequence (a.(b.c)). And what I am saying is you retain the left parenthesis forget about a, forget about dot then retain the left parenthesis forget about b, forget about dot, forget about c and then take the right parenthesis, then take the right parenthesis. So, you will say that this is the valid string consisting of two pairs of left and right parenthesis corresponding to the multiplication sequence where you multiply b and c first and then multiply the product with a. That is what I am saying here basically.

But it turns out that even though you get a valid string consisting of n pairs of parenthesis. This is not an injective mapping. Because as per this process the sequence where b and c are multiplied first and then the product is multiplied by a will lead to this sequence. Because you will forget about a you will forget about dot return the left parenthesis and then you again have a left parenthesis.

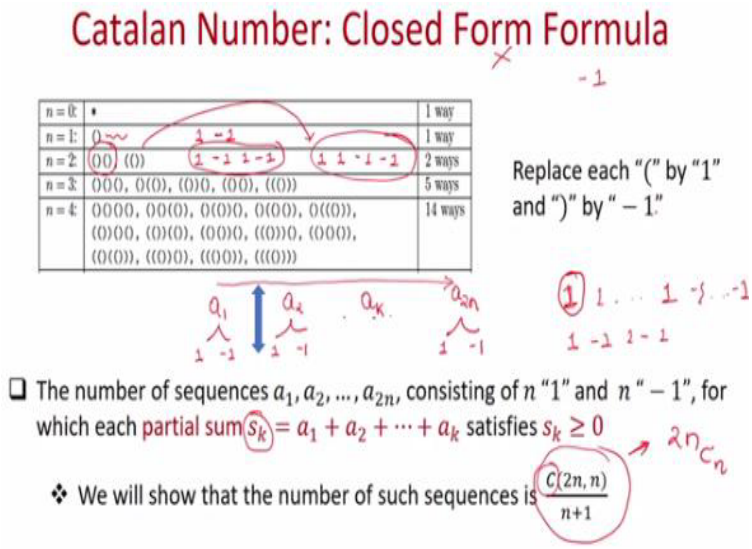
Then you forget about b forget about dot forget about c then you have right parenthesis right parenthesis. You will obtain the same sequence of two pairs of left and right parenthesis for another multiplication order where a and b are multiplied first and then the product is multiplied by c. So, this is not going to lead you to an injective mapping. But in order to claim that the solutions for both the problems the mapping should be a bijective mapping and hence it has to be an injective mapping as well.

So, how exactly we go from this set to this set. We have to do, we have to convert a valid ordering of specifying the multiplication order into a string of n pairs of parentheses in a slightly different way and this is done as follows. So, you take any multiplication order specified by your parenthesizing and you erase all the terms x_0 to xn and you erase all the left parenthesis as well. But you retain the dots and the right parenthesis.

And then so for instance what you do here is if I take the same counter example for my earlier mapping then I forget about this left parenthesis, I forget about this a but I retain the dot and then I forget about this left parenthesis, I will forget about b, I retain the next dot, I remove c and then retain the parenthesis. The same I do for the other string. And now I replace each dot by the left parenthesis if I now do this mapping then I can show that this is an injective mapping.

In fact this is a bijective mapping from this set to this set. And that shows that the solution for the new problem is the Catalan number as well.

(Refer Slide Time: 20:31)



So, as I said there are plenty of problems whose solution is the nth Catalan number; we have seen two of them. And in tutorial we will see some other problems as well. But now the question is how exactly we find a closed form formula for the nth Catalan number. We know the recurrence equation, but that recurrence equation is not what we need. We need a closed form formula namely a value of the nth Catalan number just as a function of n.

Because that will be useful if someone asked me tell me the value of say the 100th Catalan number or the value of the 500th Catalan number and so. How do we do that, so for that what we are going to do is the following. We will introduce a third problem whose solution also will be the Catalan number. And then we will derive the closed form formula for our Catalan number by solving or coming up with the number of solutions for this third problem, so what is this third problem.

So, in this problem we are given 2n values a_1, a_2, a_3, …, a_{2n}. So, we are given 2n values and those 2n values consist of n 1s and n -1s. So, basically it is the string of n number of 1s and n number of -1s. Now this n 1s and n -1s can occur in any order, but I am interested in only those sequences of n 1s and n -1s where in that sequence if I parse from left hand side to right hand side, then at every point the partial sum namely if I am at the kth position in that sequence then the partial sum will be the summation of the k characters which I have encountered till the kth position from the left hand side to the right hand side. So, basically my sequence has the values the first value is a_1 which could be either + 1 or - 1. I have the second value, again which could be + 1, - 1 and like that I have the $2n^{th}$ value which could be + 1 or - 1.

The first restriction is that in this sequence of 2n values +1 should be there at n positions -1 should be there at 1 position that is one of the restrictions. And the second restriction is that if I go from LHS to RHS that means from the starting position to the end position. And if I stop at any position k it could be the first position, I could stop at the second position, I could stop at the third position; if I stop at any position k where k ranges from 1 to 2n, and if I take the summation of the first k values which I have encountered till now while scanning then that partial sum I call it as s_k and that partial sum s_k should be non negative. That means it should be greater than equal to 0, it could be 0 or it should be more than 0. So, in some sense intuitively what it means is that if I parse from left hand side to right hand side then at each and every position the number of occurrences of + 1 should be either more than the number of occurrences of - 1 or it should be same.

It should not happen that the number of occurrences of -1 is more than the number of occurrences of +1 at any point of time when I parse the sequence from LHS to RHS. So, for instance this automatically means that my string cannot start with - 1. Any sequence any valid sequence satisfying these conditions cannot start with - 1. Why so, because if I take k equal to 1 then my partial sum s_1 becomes -1.

And that automatically shows that this is an invalid sequence. So, it automatically means that my sequence has to start with + 1. So, I can have a sequence where I have all the + 1s appearing first and then all the minus 1 appearing later. That is a valid sequence or I can have a sequence where +1 and -1 occurs alternatively. That is also a valid sequence and so on. But I cannot have a sequence of the form where I say start with 1 and then I have a -1 and then I have a -1, that is not allowed.

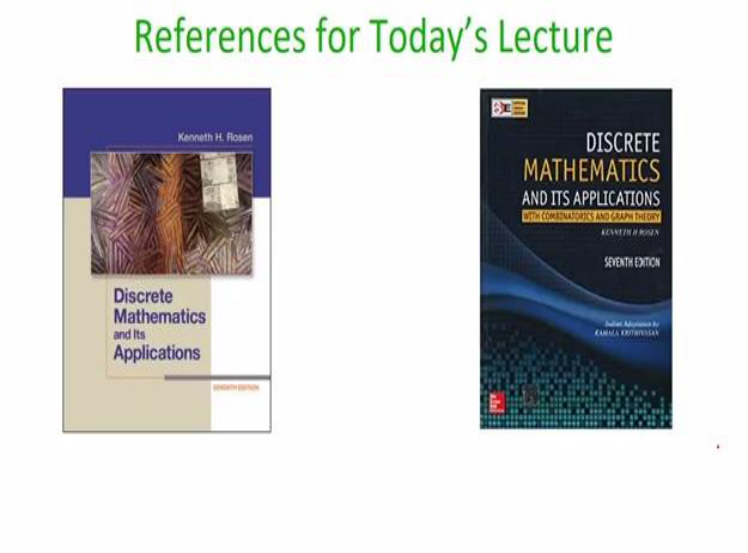
Because if I take this partial sum s_3 in this sequence then s_3 becomes -1, so that is not allowed. So, I am interested to find out the number of sequences consisting of n 1s and n -1s satisfying this condition. And it turns out that the number of sequences satisfying this condition is exactly the same as the number of valid strings consisting of n opening parenthesis and n closing parenthesis.

This is because you can interpret the new problem as the following. You take any valid string consisting of n opening parenthesis and n closing parenthesis. And you replace the opening parenthesis by 1 and you replace the closing parenthesis in those valid strings by - 1. So, you will obtain now a sequence consisting of n numbers of + 1 and n numbers of - 1. And you will see that in that string of n number of 1s and a number of -1s, the number of 1s is always greater than or equal to the number of occurrences of - 1 at any point of time if you scan that string from left hand side to right hand side. So, for instance if you take this string ( ) then this gets converted into +1 followed by -1. If you take this n equal to 2 case then the corresponding string is ( )( ) you have one then you have a closing parenthesis, so you put - 1 then you have an opening parenthesis, then you have a closing parenthesis. Whereas the second string (( )) gets mapped to the following you have left parenthesis left parenthesis. So, you have 1, you have 1 and then you have two closing parenthesis so - 1, - 1 and so on. So, that shows that the solution or the number

of valid sequences satisfying our condition is same as the number of valid strings with n pairs of parenthesis.

And what we will show in our next lecture is that the number of valid sequences satisfying these restrictions is nothing but this formula : C(2n, n) / (n+1). So, this C is not the Catalan function this is now the combinatorics function of 2n choose n. Basically the number of ways of selecting n values out of 2n values. So, the nth Catalan number: its value is 2n choose n over (n + 1). This we will prove in our next lecture.

(Refer Slide Time: 27:46)



So, that brings me to the end of this lecture. These are the references for today's lecture. So, just to summarize, in this lecture we introduced Catalan numbers, we saw three problems and the solution for each of those three problems is the recurrence relation for the Catalan number. And also, I have shown you the value of the Catalan number. In the next lecture I will explicitly solve the recurrence relation for the Catalan number and we will derive that the value of the nth Catalan number is 2n choose n over (n + 1), thank you.