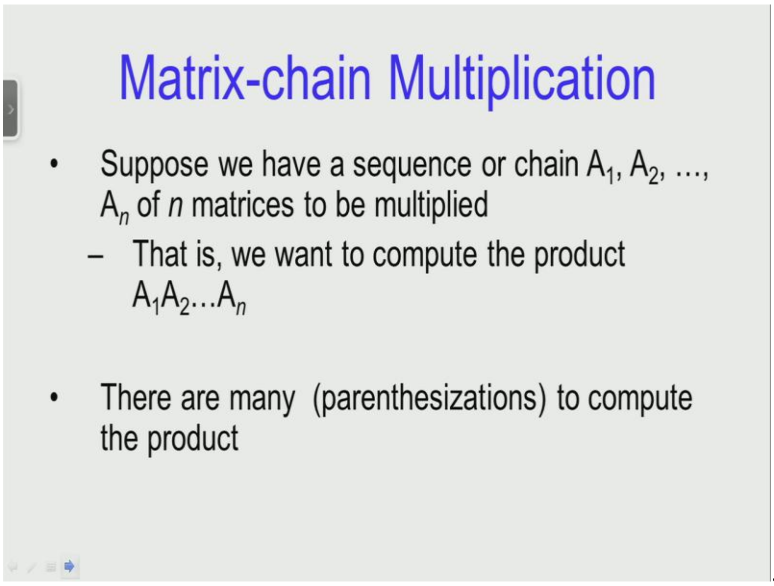
Programming, Data Structures and Algorithms
Prof. N.S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institution Technology, Madras

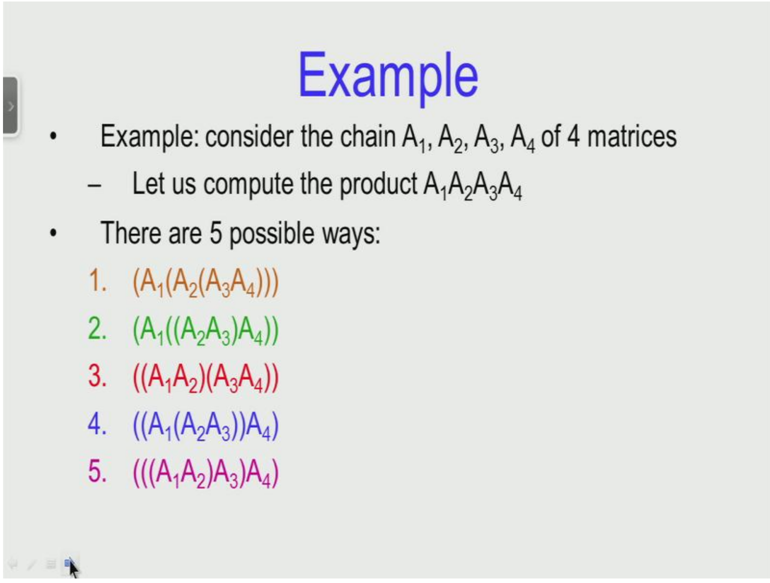
Lecture – 57
Dynamic Programming

(Refer Slide Time: 00:31)



Today we are going to study a second problem which is solved using the method of dynamic programming; this problem is called the matrix chain multiplication. The problem has that we are given a sequence of matrices that can be multiplied. So, A 1, A 2 up to A n are n matrices, which are given. And our aim is to compute the product of these n matrices. Let us assume that the product can be perform, in other words the orders of the matrices are appropriately given. There are many parenthesizations to compute the product. What is the pareanthesization? The paranthasization is an allocation of parenthesizes around the expression that has been given. So, that the expression can be evaluated has specified by the paranthasization.

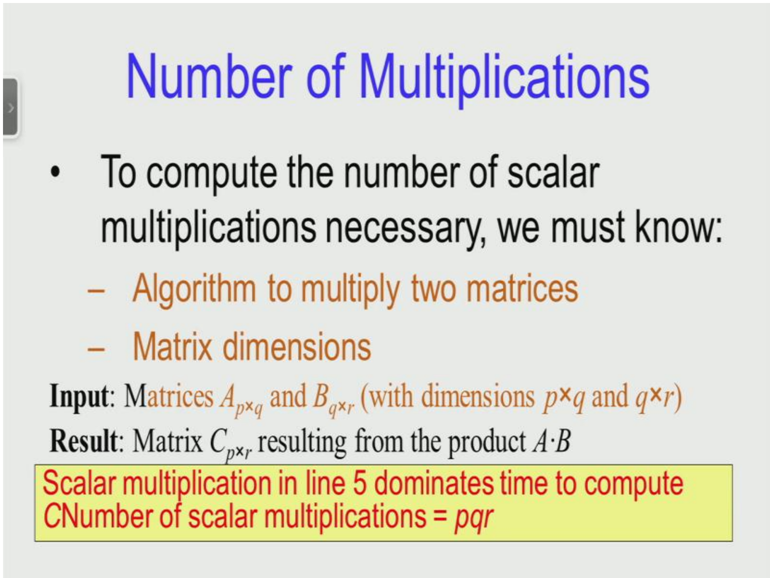
(Refer Slide Time: 01:11)



Let us now consider this example, where there are 4 matrices - A 1, A 2, A 3, and A 4; and aim is to compute the product of these 4 matrices. There are 5 possible paranthesization, let us inspect these each of paranthesization, this is the first one. We know that in any expression which is parenthesize, the innermost parenthesize is evaluated first. Therefore, in this paranthasization the first one. A 3 and A 4 are multiplied first, the result is then multiplied with A 2 and the result is then multiplied with A 1. In the next parenthesization the inner most parenthesis is the one that encloses the product A 2, A 3 which is computed first. The result is then computed is the result is then multiplied with A 4, the result of this matrix multiplication is then multiplied with A 1.

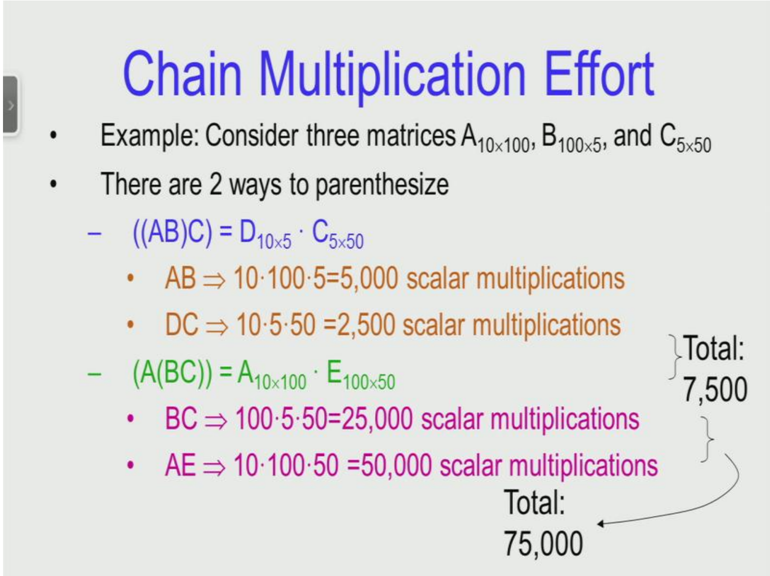
The other parenthesis this is more interesting then the first two, because it is different observe that there are two inner most parenthesis; one parenthesis contains the matrix product A 1 A 2, and the second inner most parenthesis contains the matrix product A 3 A 4, and again using expression evaluation rules, the leftmost innermost parenthesis is evaluated first. Therefore, the expression involving the product A 1 A 2 is evaluated first, then the expression involving the product A 3 A 4 is evaluated, then the two results are multiplied and this is given us a result of the matrix product of the four matrices. The following two are symmetric to the second and first respectively, and the explanation is similar. What distinguishes these 5 different parenthesisation? Let us just see that.

(Refer Slide Time: 03:13)



What distinguishes them is the total number of multiplications. Our aim is to now count the total number of scalar multiplications which are necessary. To do this, let us understand the number of multiplications required to multiply two matrices, in this case let us assume that the matrix dimensions are given, the two matrices are A which is a p by q matrix, and B which is a q by r matrix. We knows that the result is a p by r matrix, and let us call this result matrix, the value the matrix C. It is clear that the total number of matrix multiplication that need to be performed is p q r. It is clear the total number of matrix multiplications that need to be perform is p multiplied by q multiplied by r. How to this affect? The behavior of chain matrix multiplication.

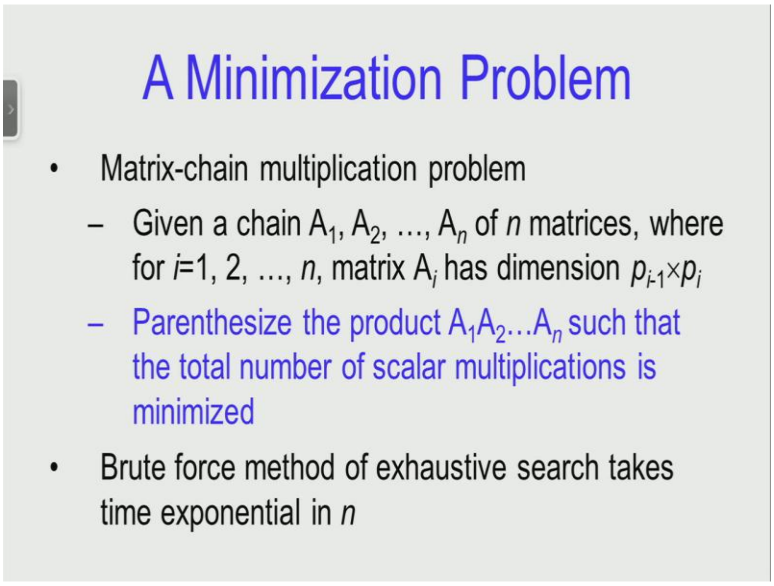
(Refer Slide Time: 04:22)



For this let us consider the following example, where we consider three matrices; A B and C. A is a order 10 by 100, B is a order 100 by 5, and C is a order 5 by 50. Clearly the three matrices can be multiplied that is the product A B C can be computed. There are two ways of parenthesizing this, this is the first way where the product AB is computed first the result is multiplied with C. Let us assume that the product AB is called D, we know that it is a 10 by 5 matrix, and C is a 5 by 50 matrix the multiplication AB takes 5000 scalar multiplications.

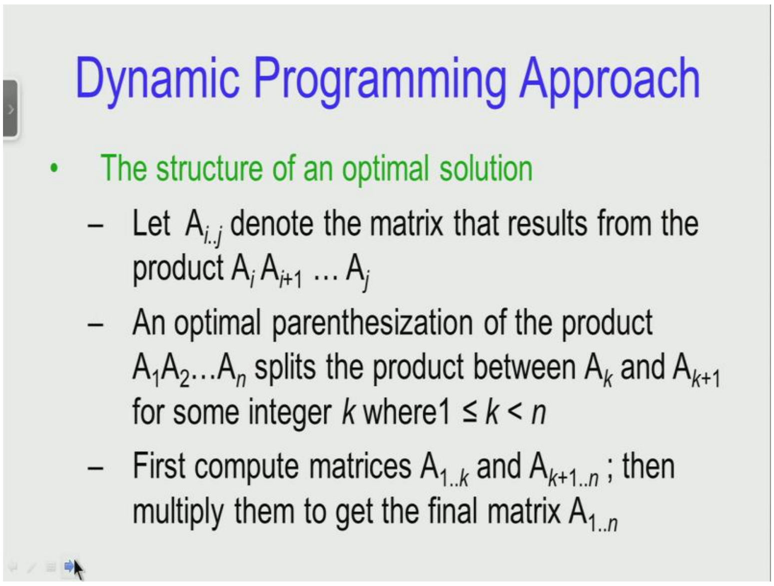
The product DC takes 2500 scalar multiplications, therefore total number of scalar multiplications is 7500. Let us consider the second parenthesizations, where B and C are multiplied first followed by A. So, let the outcome of multiplying B and C be the matrix E which is the 100 by 50 matrix. So, the product B multiplied by C which performs first takes 25000 scalar multiplications. The sub sequence product of A and E takes 50000 scalar multiplications, and therefore we can already see that the first paranthasization uses only 7500 scalar multiplications, but the second parenthesization uses 10 times more number of scalar multiplications, that is it uses 75000 scalar multiplications. Clearly from a efficient C point of you, the first parethesization is a more preferred parenthesization, then the second parenthesization.

(Refer Slide Time: 06:27)



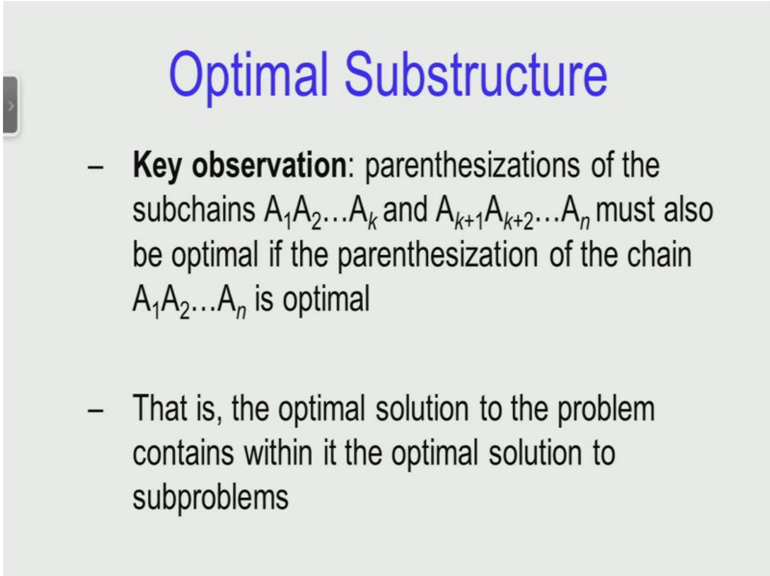
This gives rise to a very interesting minimization problem. This minimization problem is called the matrix chain multiplications problem. The input to the matrix chain multiplications problem is a chain of n matrices; A 1, A 2 to A n. For every i the i'th matrix has dimension p i minus 1 cross p i; that is the matrix A i has p i minus 1 rows and p i columns. The first matrix A 1 has p 0 rows and p 1 columns. The goal is the parenthesize this matrix, the goal is the parenthesize this chain A 1, A 2 to An. So, that the total number of scalar multiplications is minimize. Recall from the previous example that the recall from the previous example that different parenthesization give raise to different number of scalar multiplications, and our aim is to choose the optimal scalar multiple optimal parenthesization to minimize the total number of scalar multiplications. One natural approach is the brute force method, where we try all possible parenthesizations, I leave this an exercise to the student to calculate how many parenthesization are there for a chain of M matrices. It is indeed an exponential and n the exact function is left as an exercise to the student.

(Refer Slide Time: 08:02)



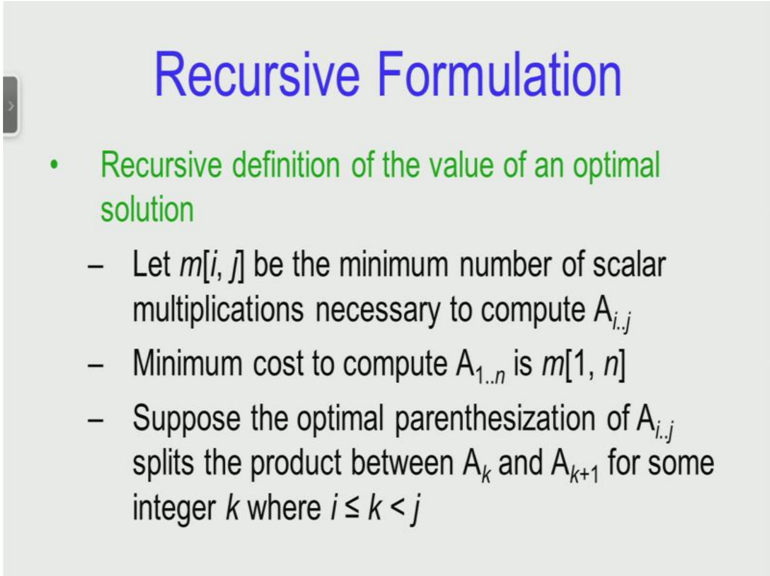
So, now let us using a dynamic programming approach to come up with an algorithm to find the minimum parenthesization. Let us use the dynamic programming approach to come out with an algorithm which will come out with the parenthesization, that uses from the minimum number of scalar multiplications. To do this let us understand the structure of an optimal solution which in this case is a parenthesization. For this we need some simple notation, we use the notation A sub scripted by i upto j to denote the matrix which is a result of the product A i A i plus 1 and so on upto A j. Let us now observe that in an optimal parenthesization which we do not know which is whatever the algorithm is trying to compute. In an optimal parenthesization, let k be the index where the product A 1, A 2 to A n is split, therefore the approach for the computing the product would first be to compute the matrices A 1 k and A k plus 1 n, and them compute the product of these two matrices to get the final matrices A 1 n.

(Refer Slide Time: 09:32)



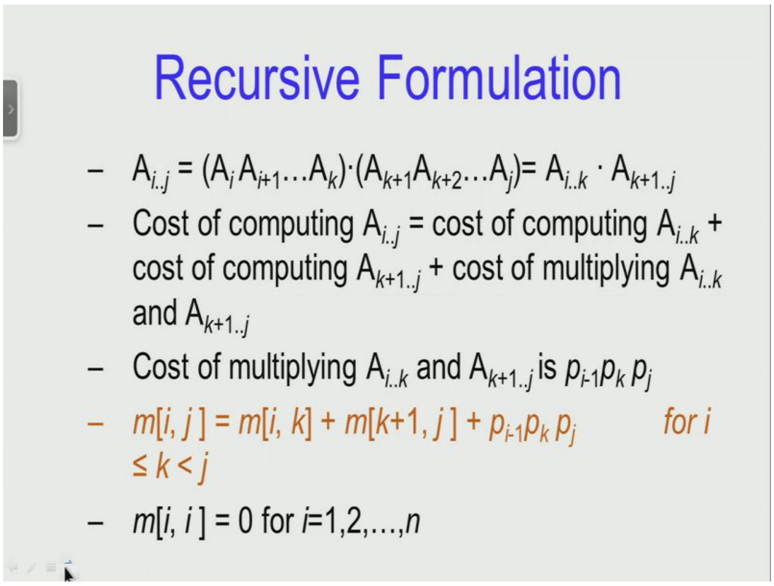
The key observation that we make about these whole exercise is that if we consider an optimal parenthesization of the change A 1, A 2 to A n, then the parenthesiztion of the sub change A 1, A 2 to A k and A k plus 1 A k plus 2 to A n will also we optimal. This is the optimal sub structure, recall that from the previous lecture this is one of the properties of recall from the previous lecture for dynamic programing to be use the problem must have the optimal sub structure. In other words in this case the optimal solution to the parenthesization contains within it the optimal solution to sub problems.

(Refer Slide Time: 10:33)



So, we will verify the client that this problem has optimal sub structure while coming that with a recursive formulation of the optimum values. In this case we again introduce a few variables which are necessary for us to compute the minimum number of scalar multiplications. So, we use the two-dimensional array m i comma j to denote the minimum number of scalar multiplications necessary to compute A i j. We let m i comma j denote, let m i comma j by b, let m i comma j be the minimum number of scalar multiplications necessary to compute A i j. Now we can see with the minimum cost of compute the chain product A 1 to A n, recall this is A sub scripted by the range 1 to n is the value m of 1 comma n. Suppose the optimal parenthesization of A i j splits the product between A k and A k plus 1 where k is a number in the range i to j. Then we write down a recursive formulation of m of i comma j.

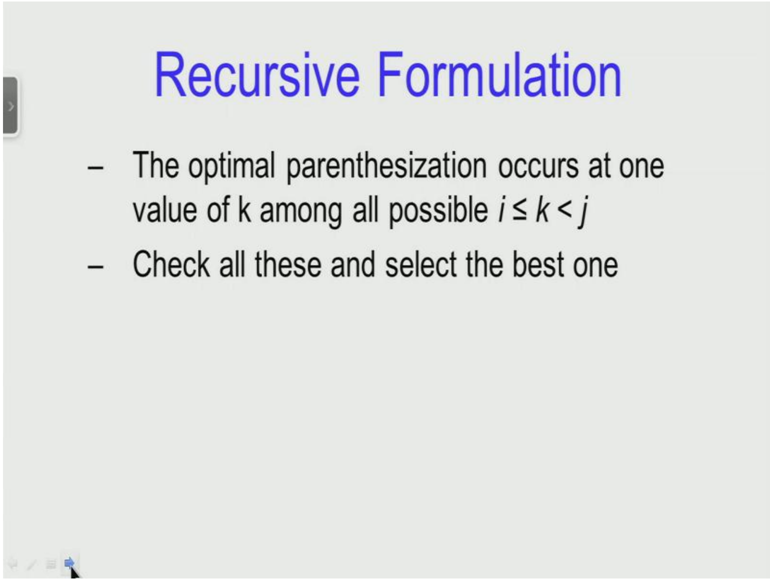
(Refer Slide Time: 12:08)



So, recursive formulation uses this parenthesization. The matrix A i j is obtain by the parethesization, the matrix A i j is obtain by multiplying the matrix change A i to A k with the result of the matrix chain A k plus1 to A j. In other word, this is the product of the two matrices; A i k multiplied by A k plus 1 j. Therefore, the total cost of computing A i j is the cost of computing A i k plus the cost of computing A k plus 1 j plus the cost of multiplying the two matrices A i k and A k plus 1 j. Note here that the cost is the total number of scalar multiplications. So, we know that the third term, the cost of multiply A i k and A k plus 1 j is p i minus 1 multiplied by p k multiplied by p j, this is the because the order of the two matrices are p i minus 1 cross p k and p k cross p j.

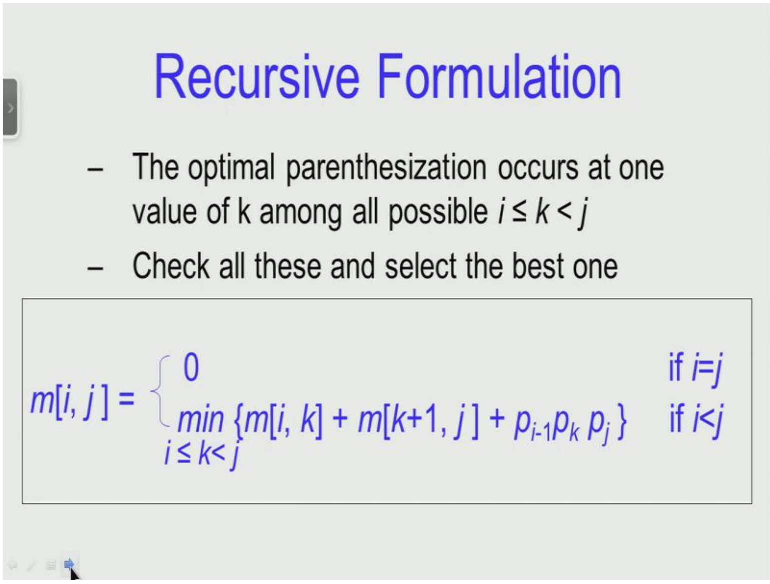
So, we specify the recursive now completely, which says that the minimum number of scalar multiplications for the chain, the minimum number of scalar multiplications for multiplying the change a i to a j is equal to m of i comma k plus m of k plus 1 comma j plus p i minus 1multiplied by p k multiplied by p j for k between i and j. And indeed the number of multiplications to compute an empty product is 0; that is m of i comma i is the cost of multiplying A i where they are no multiplications operations involve, therefore this take it be the value 0.

(Refer Slide Time: 14:36)



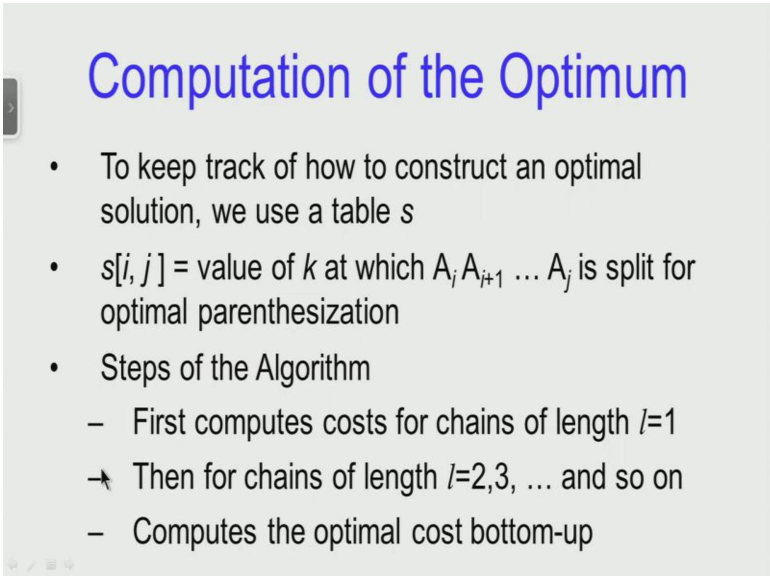
To complete the recursive of formulation, let us observe that we optimal parenthesization occurs at one of the values of k between i and j. We do not know which one it is, but the algorithmic idea is very simple, we check all the possible values of k between the range i and j and select the one that gives the least value.

(Refer Slide Time: 15:05)



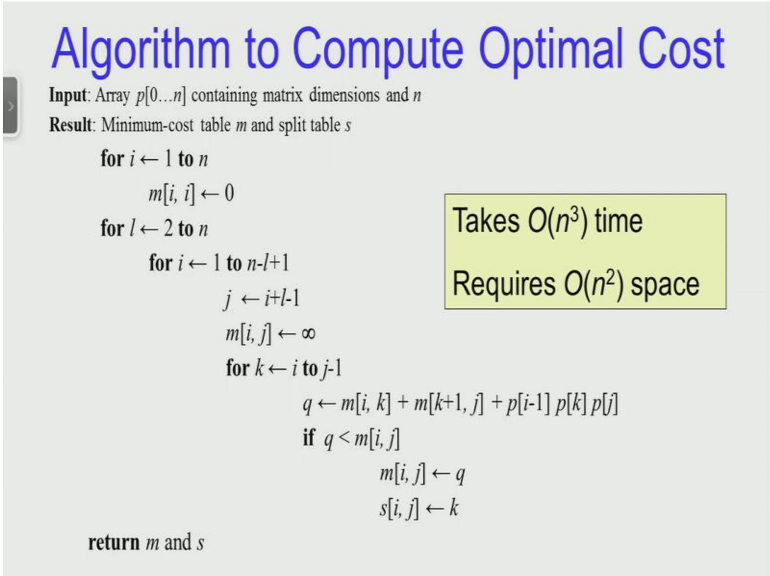
And this specifies completely the recursive formulation of m of i comma j. If i and j are the same, it is 0 because we do not have perform any multiplication. If i not equal to j and i is strictly smaller than j then m of i comma j we know stores the minimum number of scalar multiplications to multiply the chain product A i to A j. So, this is obtain by finding the best value of k by computing m of i comma k plus m of k plus 1 comma j plus p i minus 1 multiplied by p k multiplied by p j and choosing the best possible k that gives the minimum value of m of i comma j.

(Refer Slide Time: 15:59)



This completes the recursive formulation of the minimum that we are interested in. Now we need to convert this recursive formulation into an algorithm, and we have to specify the algorithm and efficient algorithm to compute the minimum. To do this we introduce a second two dimensional array which we call S; S stands for split and we refer to this two-dimensional array as the split table. The split table tells us where to split a chain. In other word S of i comma j is that value of k at which the chain A i to A j is split for an optimal parenthesization. The steps in this algorithm are to compute the minimum number of scalar multiplications for chains of length 1 from there we compute the minimum number of parenthesization for chains of length 2, and 3, and so on. This is the bottom of calculation method of the optimum value of m of 1comma n.

(Refer Slide Time: 17:17)



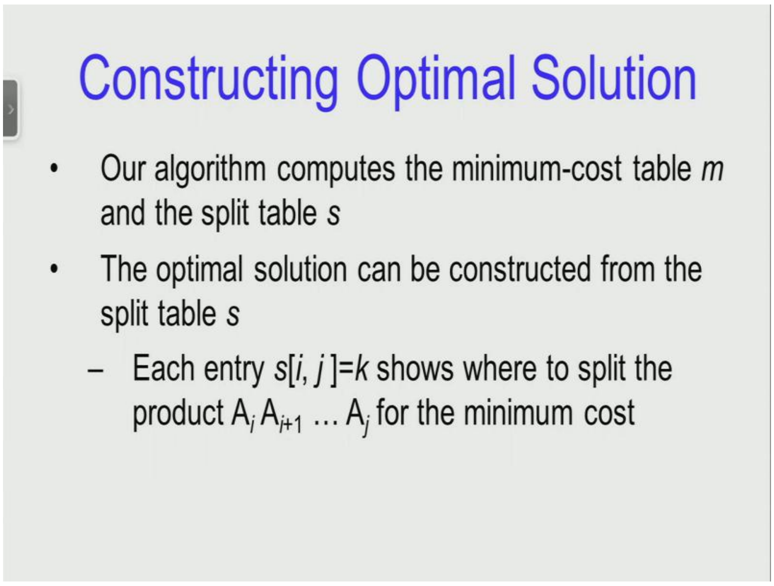
This is the algorithm description. There is an initialization face where the min cos table m is initialize with 0 for all the diagonal entries, because they do not involve any multiplication. This is followed by three nested iteration to essentially implement the recurrence, and the outer loop iterates over… So, let us consider this algorithmic description to compute the optimal cost. Using this data we will then compute the optimal parenthesization also. The input to these algorithm is an array which an n plus one element array which contains the dimensions of the matrices. For example, the feels p of 0 and p of 1 give us the information about the dimensions of matrix A 1, that is p 0 cross p 1, the array entries p 1 and p 2 tell us the dimension of the matrix A 2 and so on. The result of this algorithm is we get a min cos table and is split table; these are two arrays that we get. The min cos table stores the value of the minimum cost parenthesization of the chain multiplication involving the chains involving the chain of matrices A i A i plus 1 upto A j. Similarly the split table the entry S of i j stores the value of the index k at which the chain A i to A j is to be split.

The algorithmic as follows it has 4 for loops. The first for loop is an initialization face where the diagonal entries are all initialize to 0, this is natural because the diagonal entries store the value 0 to denote the fact that there is no matrix multiplication involving the single matrix. The remaining 3 for loops are nested and the intent of these for loop is to fill the remaining entries in the upper half of the matrix is to fill the entries in the upper half with the matrix, and this is done by filling each diagonal. Observe that there are m minus 1diagonals apart from the principle diagonal of the matrices. The outer loop iterates over the diagonals of the matrix, the outer for loop which is index by the variable l, iterates over the diagonals.

The next for loop is setup to instantiate each element in the appropriate diagonal and the innermost for loop is the one that evaluates the value of the min cos parenthesization. So, in the second for loop, the initialization of the variable j to i plus l minus 1 is the choice of the appropriate element in the l'th diagonal. So, m of i comma j is initialize to the value empty which is the standard think for the minimization problem which takes a positive values m of i comma j is initialized to the value infinity which is standard practice for minimization problems which takes positive values. The inner most for loop is the loop that implements the recurrence that we have return to formulate the value of m of i comma j. The way this is done is to iterate over all the possible values of k starting from i to j minus 1, and the value q is computed has m of i comma k plus m of k plus 1 comma j plus p of i minus 1 multiplied by p of k multiplied by p of j.

The if statement updates the value of m of i comma j, if the value of q is smaller and it also updates the value of the split entry, if the value of q is smaller than the current value of m of i comma j. At the end of this whole iteration the matrices m and s are computed, and this store the optimum parenthesization for every i comma j, this store the optimum number of scalar multiplications for the chain multiplication involving A i to A j and the parenthesization information is stored by keeping track of a split value in the matrix S.

(Refer Slide Time: 22:55)



The split table is use to compute an optimum solution, the algorithm computes first the min cos table and the split table S as we saw in the previous slide. And the optimum solution can be calculated from the split table using the value k, which is stored in S of i comma j. This enables has to compute S of 1 comma n recursively.

Thank you.