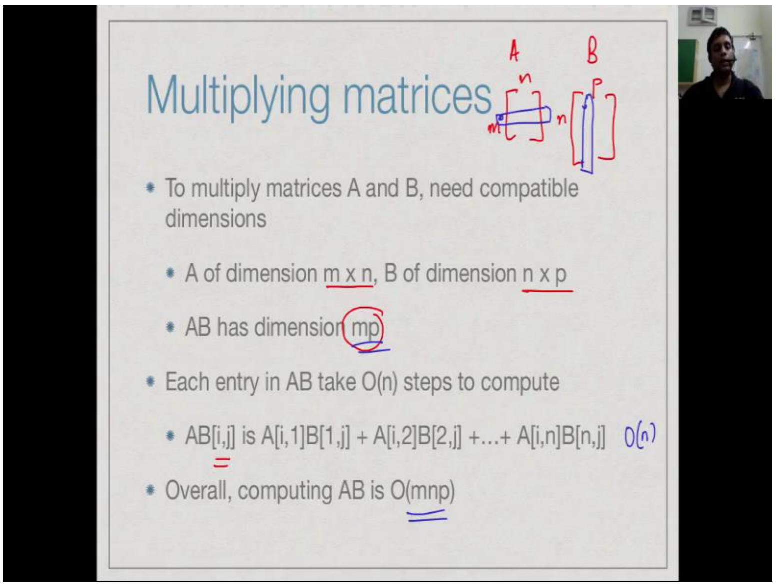
Design and Analysis of Algorithms, Chennai Mathematical Institute
Prof. Madhavan Mukund
Department of Computer Science and Engineering,

Week - 07
Module - 06
Lecture - 49

Matrix Multiplication

For our last example of dynamic programming to this week, now we look at the problem of the efficiently multiplying the sequence of matrices.

(Refer Slide Time: 00:09)



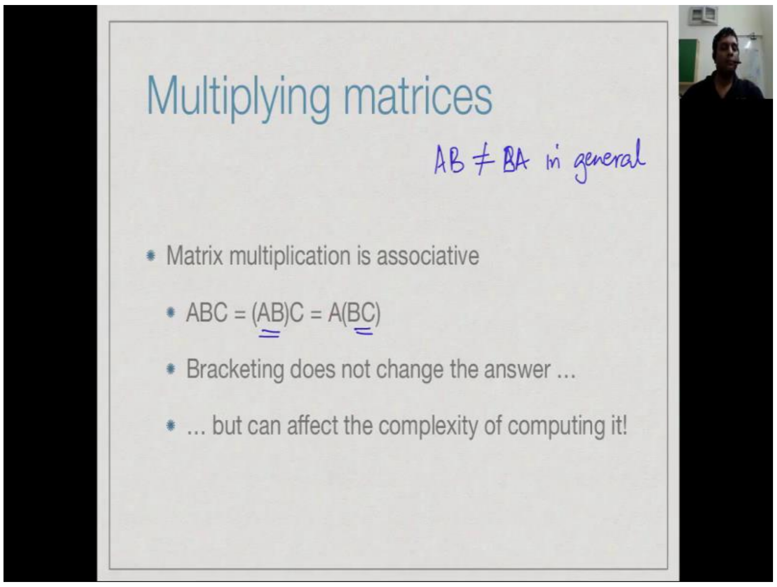
Multiplying matrices

- To multiply matrices A and B, need compatible dimensions
  - A of dimension m x n, B of dimension n x p
  - AB has dimension mp
- Each entry in AB take O(n) steps to compute
  - AB[i,j] is A[i,1]B[1,j] + A[i,2]B[2,j] +...+ A[i,n]B[n,j] $O(n)$
- Overall, computing AB is O(mnp)

So, as you probably know to multiply two matrices A and B, we need compatible dimensions, so if we have A and B, then we need that the number of columns in A must match the number of rows in B. So, we have m times n matrix A and an n times p matrix B and the final matrix is going to be your times m times p, that the same number of rows as A must have same number of columns as B. And the way you compute that product A B, so if I want the i j th entry in A B, then I will take the i th row in A and I some have multiplied with the j th column B. So, I will take the first entry here, the first entry there, so I will take A i 1 and B 1 j, multiply them A i 2 B 2 j multiply them, finally A i n b n j multiply add them all.

So, this takes order n steps, because I am computing order n pair wise products and then adding them all. So, therefore, I have to compute finally m times p entries, each entry requires a linear order n amount to there. So, the total cost of multiplying two matrices in terms of basic arithmetic operations is of the order of m times n times p. So, this is the basic fact that we need for the problem.

(Refer Slide Time: 01:34)



Multiplying matrices

$AB \neq BA$ in general

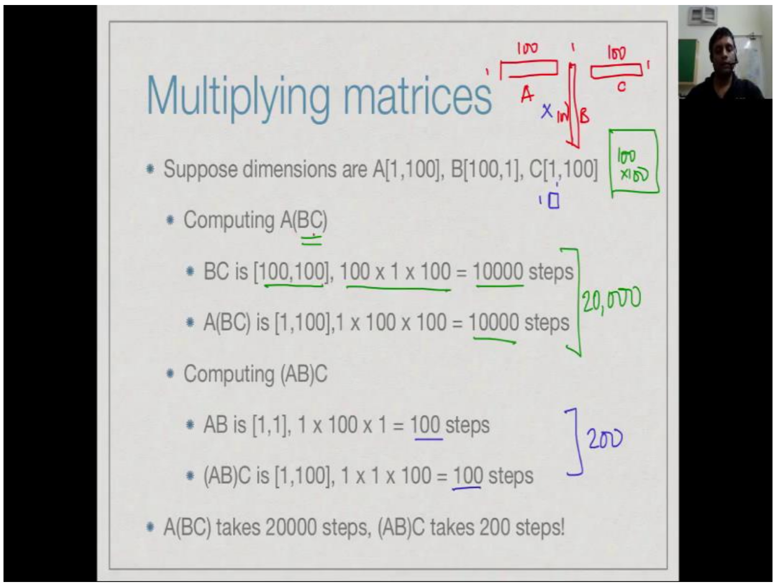
- Matrix multiplication is associative
  - $ABC = (\underline{\underline{AB}})C = A(\underline{\underline{BC}})$
  - Bracketing does not change the answer ...
  - ... but can affect the complexity of computing it!

So, now the concern is not computing the product of two matrices by computing the product of 3 or more. So, supposing I want to multiply three matrices A, B and C, at a time I can even multiply I need 2. So, either I have to multiply it as A times B, let an intermediate matrix A B and then multiply by C or I can do B times C and then multiplied by A from B itself.

So, matrix multiplication is not commutative, we do not have this in general. So, it is important that the order is the same, but within that in which sequence I do this simplification does not matter. So, it is associative, I can bracket it by as A times B followed by C or A times B followed by C, either way I could do two matrix multiplications and what associativity means is that the answer will not change, the final product would remain the same.

But, what is interesting for us, now is that the complexity of computing the answer can change depending on which order I do it, whether I do it as A times B followed by C or first at B times C and then might be multiply by A.

(Refer Slide Time: 02:47)


Multiplying matrices

- Suppose dimensions are A[1,100], B[100,1], C[1,100]
  - Computing A(BC)
    - BC is [100,100], 100 x 1 x 100 = 10000 steps
    - A(BC) is [1,100],1 x 100 x 100 = 10000 steps
  - Computing (AB)C
    - AB is [1,1], 1 x 100 x 1 = 100 steps
    - (AB)C is [1,100], 1 x 1 x 100 = 100 steps
- A(BC) takes 20000 steps, (AB)C takes 200 steps!

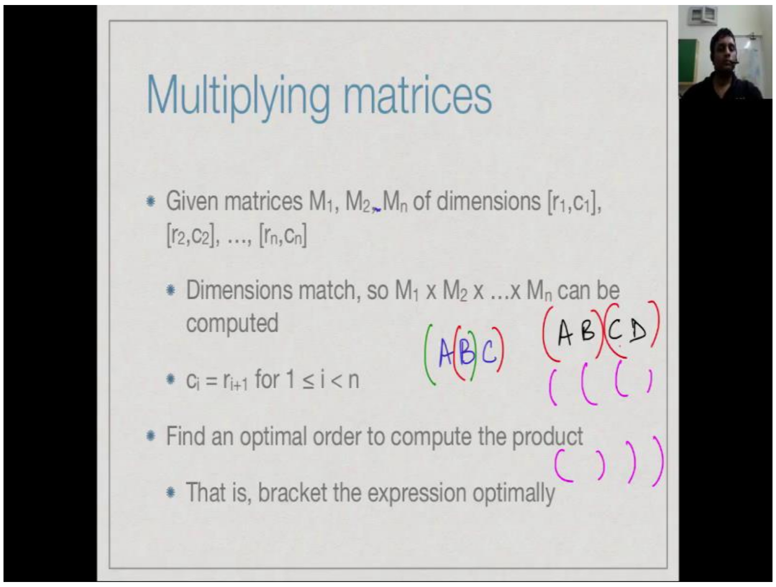
So, let us look at a very trivial example. So, I have three matrices of this kind, so A is 1 and these are matrix which looks like this, it has 1 row and 100 columns, B is a matrix which looks like this, it has 100 rows and 1 column and C is again a matrix, so it looks like this. So, this is my A, this is my B and this is my C. So, now, supposing I do B times C first, if I do B times C what happens is in this 100 will now blow up and I will get a big matrix which is 100 by 100.

Because, I have 100 rows here and 100 columns there and so the final thing is going to be number of rows in B and number of columns in C. So, the B times C matrix is actually 100 by 100 and how many steps it will take, it takes m into n into p which is 100 into 1 into 100 which is 10,000 steps. Now, I have A which is 1 by 100 and B which is 100 by 100, so I am going to produce something which is again 100, 1 by 100 that you I want to take 1 into 100 into 100, we call m into n into p, another term ((Refer Time: 03:56)).

So, together this particular sequence requires 20,000 steps, predict the other way on the other hand, if I computed this product first, then I have collapse it to a simple 1 by 1 matrix. So, the number of rows here is 1 and the number of columns is there 1, so A B is 1 by 1 matrix and it takes only 100 steps to compute. And now, again I have a 1 by 1 times of 100 by 1 matrixes, so again it takes only 100 steps, so this takes only 200 steps.

So, you can see that there can be a dramatic difference in the complexity depending on whether you, the bracketed is A B followed by C or A followed by B C.

(Refer Slide Time: 04:33)



Multiplying matrices

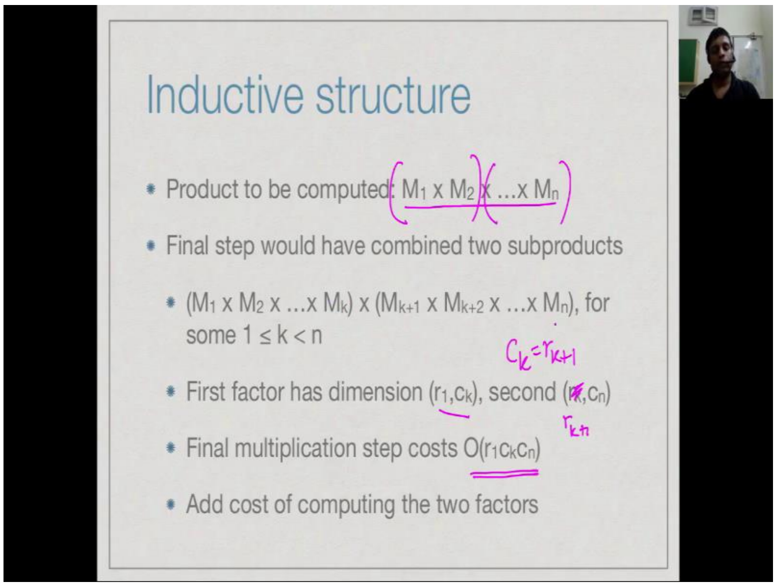
- Given matrices $M_1, M_2, \ldots M_n$ of dimensions $[r_1,c_1]$, $[r_2,c_2]$, …, $[r_n,c_n]$
  - Dimensions match, so $M_1 \times M_2 \times \ldots \times M_n$ can be computed
  - $c_i = r_{i+1}$ for $1 \leq i < n$
- Find an optimal order to compute the product
  - That is, bracket the expression optimally

So, in general we have a sequence of M such matrices, it is right, so M 1, M 2 up to M n and their dimensions will be r 1 C 1, r 2 C 2 up to r n C n number of rows, number of columns. The dimensions will be given to us, so that we can be multiplied. Number that all we need, assuming the entries of course, can be combined by sensible multiplication addition operations, what we need to be able to multiply two matrices, the dimensions should match.

So, be guarantee that the column of each matrix, the number of columns is equal to the number of rows in the next matrix, so that product is well defined. And our goal is to find an optimum way to compute this product, what is the sequence of basic operations multiplying two matrices together there we need to perform to get the minimum overall cost. And this is equivalent to finding an optimum way of bracket. So, remember when I did A times B times C, the choice of whether to put the bracket like this, how to put the bracket like this.

In general, if I give you now A times B times C times D, then you can do many things, you can do A B, C D and then multiply or you can do C D, then B C D, then A B C D or you can do A B, then A B C, then A B C D. So, there are many different ways in which you can partially compute pairs of product, products of pairs and built up the whole thing. So, we want to find the optimum way to do this.

(Refer Slide Time: 05:58)



Inductive structure

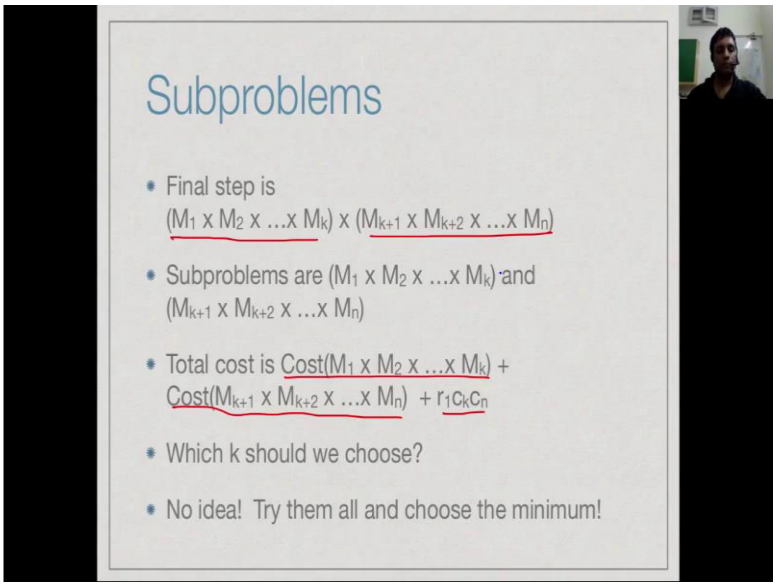
- Product to be computed: $(M_1 \times M_2)(\times \ldots \times M_n)$
- Final step would have combined two subproducts
  - $(M_1 \times M_2 \times \ldots \times M_k) \times (M_{k+1} \times M_{k+2} \times \ldots \times M_n)$, for some $1 \le k < n$
  - First factor has dimension (r_1, c_k), second $(\cancel{r}, c_n)$ $c_k = r_{k+1}$ r_{k+1}
  - Final multiplication step costs $O(r_1 c_k c_n)$
  - Add cost of computing the two factors

So, let us try and identify the inductive structure in this part. So, our goal is to compute this long product M 1 to M n, but remember that at every stage we can only multiply two matrices at a time. So, at the final stage you would multiply two some mat, some two such matrices. Now, we can regroup by bracketing, but we cannot reorder. So, if we did the final stage we would have two groups, so you would have done some product from the left to the midpoint and some product from the midpoint to the n.

So, for some k we would have computed M n to M k and M k plus 1 to M n. In the first part after doing all this, we will have as many rows as M 1 and as many columns as M k, so it will be r 1 times C k. Second one will be r k plus 1 to C n and we know that C k is going to be equal to r k plus 1, so that these two kindly multiplied together. So, the final cost is going to be m into n into p which is r 1 into C k into C n, so we know how much the last step takes.

Assuming that it was broken at M k, the last multiplication causes this much. Now, to get the total cost of doing it with this particular choice of k, we have to get the cost of computing M 1 to M k and M k plus 1 to M n.

(Refer Slide Time: 07:30)



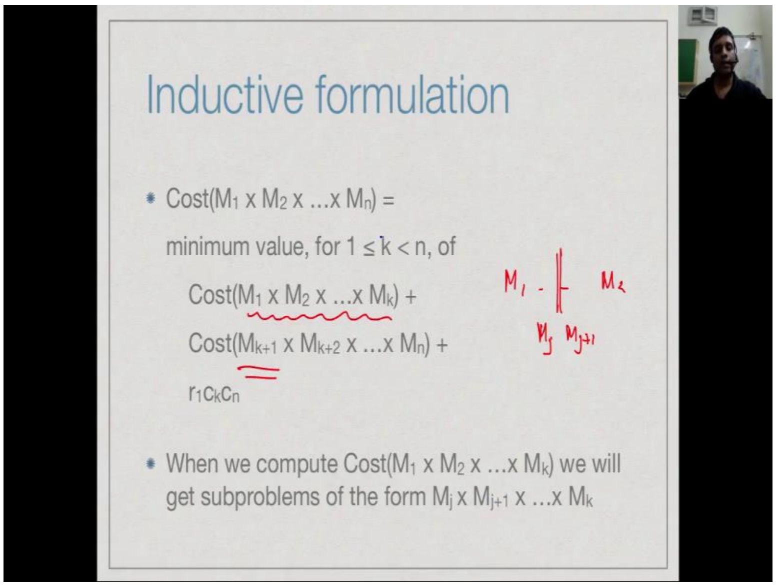
Subproblems

- Final step is $(M_1 \times M_2 \times \ldots \times M_k) \times (M_{k+1} \times M_{k+2} \times \ldots \times M_n)$
- Subproblems are $(M_1 \times M_2 \times \ldots \times M_k)$ and $(M_{k+1} \times M_{k+2} \times \ldots \times M_n)$
- Total cost is $\text{Cost}(M_1 \times M_2 \times \ldots \times M_k) + \text{Cost}(M_{k+1} \times M_{k+2} \times \ldots \times M_n) + r_1c_kc_n$
- Which k should we choose?
- No idea! Try them all and choose the minimum!

So, we have this final situation and now we have these two sub problems, so we have this two sub problems and the total cost is going to be the cost of the first sub problem. However, it much takes to compute M 1 to M k; however, much it takes to compute M k plus 1 to M n plus the cost of the last step which is to multiply these two sub problems after that produce one matrix each to multiply those two matrices together. But, we have said that this k could be anywhere between 1 and n, so which k should we choose.

So, the spirit of these kind of problems that we have been seen this inductive things is that we do not try to make the choice. We say we have no idea in advance which k is good, so we just try out all possible case and take the best one, in this case the minimum value.

(Refer Slide Time: 08:21)

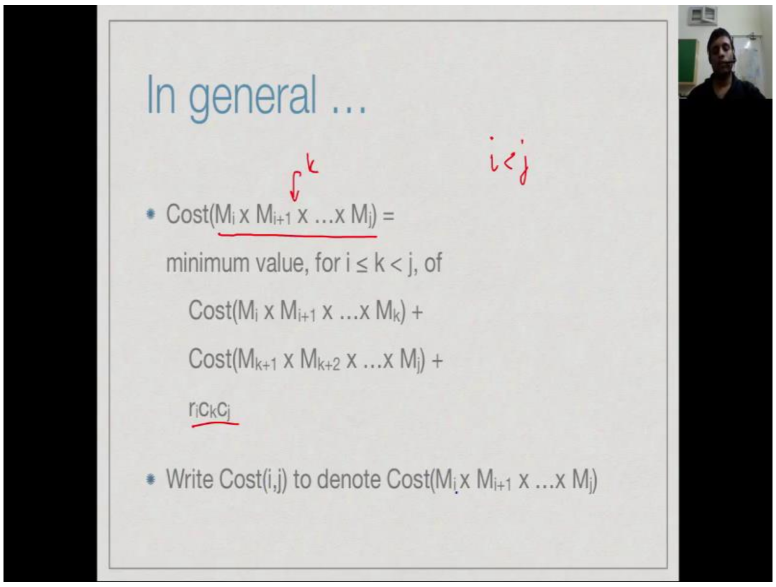


Inductive formulation

- Cost(M_1 x M_2 x …x M_n) = minimum value, for $1 \leq k < n$, of
  - Cost($\underbrace{M_1 \text{ x } M_2 \text{ x } \ldots \text{x } M_k}$) +
  - Cost($\underline{\underline{M_{k+1}}}$ x M_{k+2} x …x M_n) +
  - $r_1 C_k C_n$
- When we compute Cost(M_1 x M_2 x …x M_k) we will get subproblems of the form M_j x M_{j+1} x …x M_k

In other words, the cost for multiplying M 1 to M n in terms of total number of operations should be the minimum value of k between 1 and n, it has to be strictly less than n, because the second part will be n k plus 1. So, between 1 and n or multiplying the matrices 1 to k, multiplying the matrix k plus 1 to n and adding the cost of the last multiplication. So, now in turn if I look at this for example, in a break it out, it could break up at some intermediate point M j. So, I will add some M j plus 1 to M k, so I would get arbitrary segments from 1 to n as my sub problems.

(Refer Slide Time: 09:08)

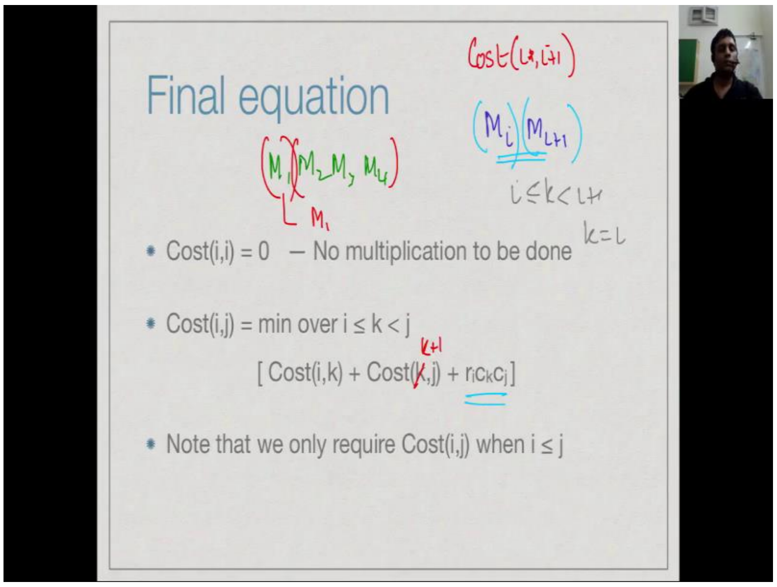


In general …

- Cost($\underline{M_i \text{ x } \overset{k}{M_{i+1}} \text{ x } \ldots \text{x } M_j}$) = minimum value, for $i \leq k < j$, of
  - Cost(M_i x M_{i+1} x …x M_k) +
  - Cost(M_{k+1} x M_{k+2} x …x M_j) +
  - $\underline{r_i C_k C_j}$
- Write Cost(i,j) to denote Cost(M_i x M_{i+1} x …x M_j)

So, a natural thing is to define the inductive structure on an arbitrary segment from M i to M j where of course, i is less than j. So, we want the minimum value for any k in the middle, k between i and j or going for M i to M k and then from M k plus 1 to M j and the cost of the last multiplication which is r 1 into C k into C j. So, as before we will just use the index, so instead of writing cost of M 1, cost of M 2, then whatever we will write i j, cost of i j is the cost of multiplying M i to M j that whole sequence.

(Refer Slide Time: 09:47)



Final equation

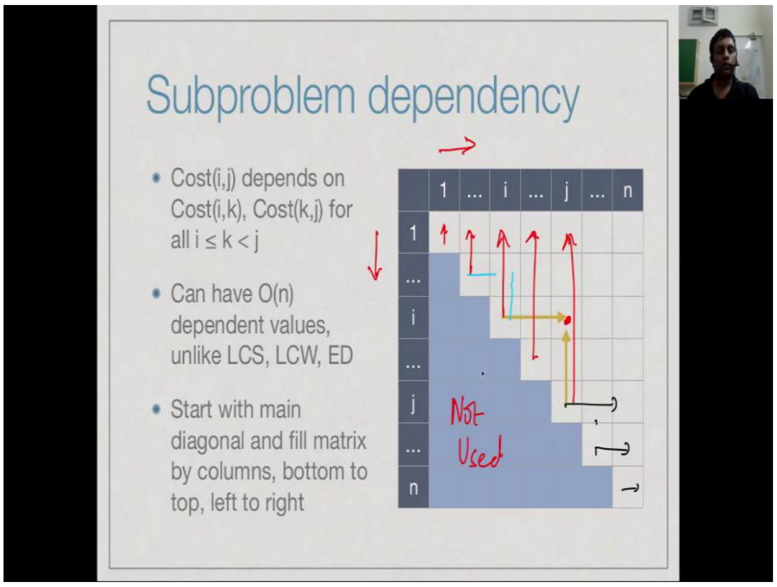
- Cost(i,i) = 0 – No multiplication to be done
- Cost(i,j) = min over i ≤ k < j
  [Cost(i,k) + Cost(k,j) + $r_i c_k c_j$]
- Note that we only require Cost(i,j) when i ≤ j

So, let us look at the final inductive form of the equation that we need to compute. So, the base case is when we are looking at a sequence of length 1, so if you are computing for example, M 1, M 2, M 3, M 4, then one possibility that are break it up as M 1 times M 2, M 3, M 4. Now, if I am doing this break up then the cost of doing M 1 is nothing, because if I am computing from 1 to 1, I am doing no multiplication. So, the cost of i i is 0 and then otherwise I use the recursive formulation we had before.

We take the minimum value for k ranging form i, but not rather strictly less than j of the cost of i to k, cost k plus 1 to j and r i times C k and C j. And of course, we will only compute this when i is less than equal to j, we will never be able to, we will never want to computed when the index j is smaller than i. Now, it is instructed to check what happens when I do something like cost i i plus 1. So, what this will says that I want to take M i M i plus 1, now my only possible value for k, so k should be between i and strictly less than i plus 1.

So, that means the only possible value for k is i, so this will break up this thing as M i to M i and M i plus 1 to M i plus 1 and then inductively those will give me cost 0 and then I will only get the cost of multiplying this pair which will exactly come out of this pair. So, there is no problem with this base case, we only need the base case when the two indices are exactly equal.

(Refer Slide Time: 11:27)



Subproblem dependency

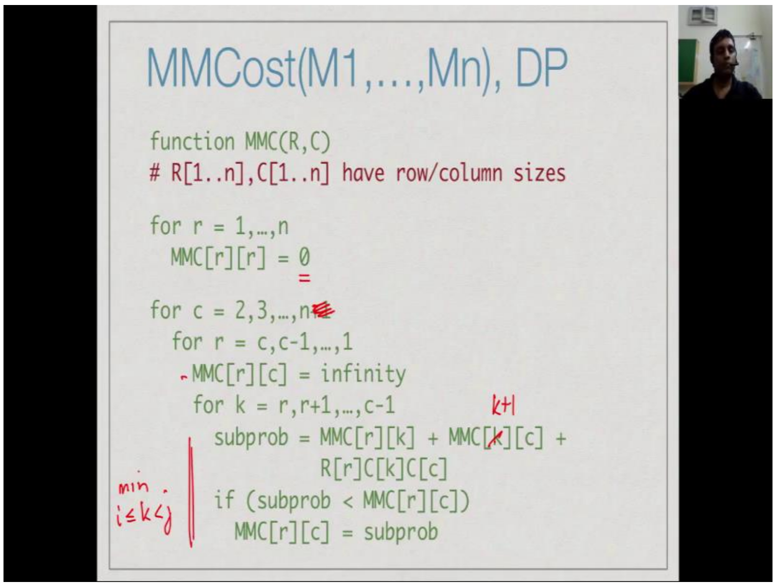
- Cost(i,j) depends on Cost(i,k), Cost(k,j) for all $i \leq k < j$
- Can have O(n) dependent values, unlike LCS, LCW, ED
- Start with main diagonal and fill matrix by columns, bottom to top, left to right

So, now, as before we will have this matrix to fill up, because we have this is not used, because we require, so i is going this way. So, we have the first index going this way and the second index going that way, so we basically never need to look at values where the ending point is before the starting point, so we only need to look top of the time. And now if you are trying to compute the position i j, then we will need cost i k, that is in this row we will need values between for smaller values M j and we will need values of the form j k, k j in other in this column.

So, we will need the values below and to the left which is spectral diagonal, so one way of doing this is to fill up this way, so that wherever we fill up, we have the values below already and we also have the other values here left. So, we will fill up this matrix from bottom top, we can also fill it up by diagonal, but it is very painful to program it. So, it is better to do it either row by row or column, so you can also start like this way, you could also do in this way.

Since that wherever you start, you have the values below and you get that, so you start in the diagonal and both right or up. So, either you start at the top left and work down the diagonal and each diagonal, each column we do bottom to top. Now, you start in the bottom right and work up the diagonal and each row you do left to right, so either these would done.

(Refer Slide Time: 13:01)



MMCost(M1,…,Mn), DP

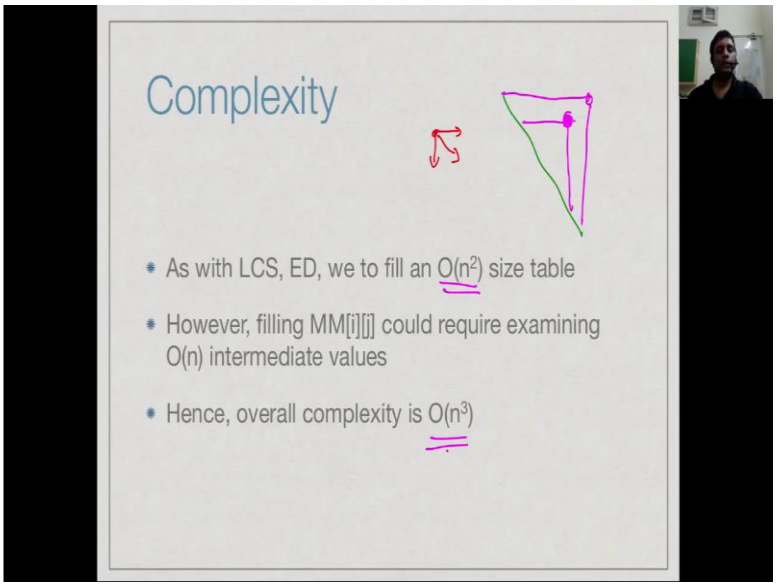
```
function MMC(R,C)
# R[1..n],C[1..n] have row/column sizes

for r = 1,…,n
  MMC[r][r] = 0

for c = 2,3,…,n
  for r = c,c-1,…,1
    MMC[r][c] = infinity
    for k = r,r+1,…,c-1                k+1
      subprob = MMC[r][k] + MMC[~~k~~][c] +
                R[r]C[k]C[c]
      if (subprob < MMC[r][c])
        MMC[r][c] = subprob
```

$\min_{i \le k < j}$

So, here is a pseudo code for that, so we first initialize the diagonal to be 0 and now we do what we said before, we said we start all the columns ((Refer Time: 13:15)). So, this should actually be in a column. So, column 1 we do not have to do because that is the diagonal in which we started column 2 onwards, so 2 to n. Then for every row starting from the diagonal up to the top row, we now need to compute this minimum. So, we initially assume that the value to be filtered is infinity, now infinity could just need the product of all the dimensions plus 1, because you know that the ((Refer Time: 13:42)) the cost is not going to exceed that.

So, you can choose a large value and what we do is now you check for each value of k, you find that inductive thing. You look at r to k, k plus 1 to C and then r times C k times C C. So, you look at this particular value which is the inductive thing and then if it is smaller than the value you have seen so far, so this is basically computing that min over k by setting into infinity and then looking and updating every time. So, this is just a direct implementation of the recursive of the inductive thing and it is just enumerating the sub problems in a way which respects the dependency or gives the input before.

(Refer Slide Time: 14:29)



Complexity

- As with LCS, ED, we to fill an $O(n^2)$ size table
- However, filling MM[i][j] could require examining O(n) intermediate values
- Hence, overall complexity is $O(n^3)$

So, one interesting thing to notice that we are filling up an order n square table, now when we looked at longest common subsequence or edit distance, we said that the complexity of the problem time wise was exactly the complexity of the size of the table. We had an M times M table it takes M times M time to fill it up, because in those two problems we were filling up each entry looking at exactly three neighbors.

So, we had a constant look up, now here what happens is that when I am abort the diagonal at some distance in order to fill this entry, I need to scan this row and this columns. So, the amount of time it takes to fill one entry in my matrix could be linear. In the extreme case when I am very powerful in the diagonal, I could be spending order n time to fill that entry. So, each intermediate value of the matrix could require looking at n intermediate, each value each position could require looking at n intermediate value.

So, though this table is of size n square, the actual complexity of filling the table is of size n cube that is why this is an interesting problem, because it says that you cannot directly conclude the complexity of a dynamic programming algorithm from the size of the table that we are trying to update. Because it also depends on the amount to the effort you have to take to update each entry in the table.