Design and Analysis of Algorithms

- ***Course Code***: BCSE304L
- ***Course Type***: Theory (ETH)
- ***Slot***: A1+TA1 & & A2+TA2
- ***Class ID***: VL2023240500901
  VL2023240500902

A1+TA1

Day	Start	End
Monday	08:00	08:50
Wednesday	09:00	09:50
Friday	10:00	10:50

A2+TA2

Day	Start	End
Monday	14:00	14:50
Wednesday	15:00	15:50
Friday	16:00	16:50

Syllabus- Module 2

Module:2	Design Paradigms: Dynamic Programming, Backtracking and Branch & Bound Techniques	10 hours

Dynamic programming: Assembly Line Scheduling, Matrix Chain Multiplication, Longest Common Subsequence, 0-1 Knapsack, TSP-
Backtracking: N-Queens problem, Subset Sum, Graph Coloring-
Branch & Bound: LIFO-BB and FIFO BB methods: Job Selection problem, 0-1 Knapsack Problem

Dynamic Programming

- Dynamic programming is an algorithm design strategy for recursively solving problems.
- Dynamic programming technique breaks the problems into sub-problems, and each sub-problem is solved only once, and the result of each sub-problem is stored in a table (array or hash table) for future reference.
- The Technique of storing the sub-problem solution is known as "memoization.“
- These sub-problems solutions are used to obtain the original solution.

Dynamic programming solves optimization problems using Principle of Optimality.

Principle of Optimality

Principle of Optimality:

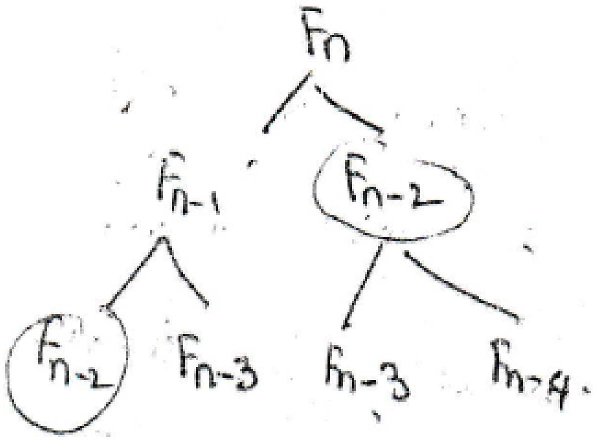
- A problem satisfies the Principle of Optimality if the sub-solutions of an optimal solution to the problem are themselves optimal solutions for their respective subproblems.
- In other words, an optimal solution to a larger problem can be constructed from optimal solutions to its subproblems.

Example - Shortest Path Problem:
The shortest path problem indeed satisfies the Principle of Optimality. The principle is reflected in the following way:
When:
If a sequence of nodes a, x_1, x_2, ..., x_n, b represents the shortest path from node a to node b in a graph, then the portion of the path from x_i to x_j is also a shortest path from x_i to x_j.

In practical terms, this means that if you're looking for the shortest path from one node to another in a graph, the optimal solution for the entire path can be built by combining optimal solutions for each segment of the path.

Normal Recursion

```
fib(n){
 if n≤2 return 1
 Else return fib(n-1)+fib(n-2)
}
```



$$T(n) = T(n-1) + T(n-2) + O(1)$$
$$\geq 2T(n-2) + O(1)$$
$$\geq 2^{n/2} \text{ Exponential}$$

Dynamic Programming approach

```
memo = {}
fib(n) {
 if n in memo: return memo[n]
 else if n≤2 return f=1
 else f=fib(n-1)+fib(n-2)
     memo[n] = f;
     return f;
}
```

$$T(n) = T(n-1) + O(1)$$
$$= O(n)$$

Dynamic Programming approach=Recursion + Reuse

Elements of Dynamic Programming

- **Optimal Substructure**: This property suggests that the optimal solution to a problem can be constructed from optimal solutions of its subproblems. Breaking down a problem into smaller overlapping subproblems facilitates solving the larger problem.

- **Overlapping Subproblems**: This property indicates that the subproblems in a dynamic programming approach are not independent; they share subproblems. The technique exploits this overlap by storing the solutions to subproblems, preventing redundant computations.

- **Memoization**: This is a specific technique in dynamic programming where the results of expensive function calls are cached and reused when the same inputs occur again. It involves creating a table (often implemented as an array or a hash map) to store solutions to subproblems. This helps avoid redundant calculations and improves the efficiency of the algorithm.

Applications of dynamic programming

- Matrix chain multiplication
- Longest common subsequence (LCS)
- Assembly Line Scheduling
- All pair Shortest path problem
- 0/1 knapsack problem
- TSP
- Rod Cutting
- Optimal Binary Search Tree
- Reliability Design

Matrix multiplication

- A, B and C are three matrices then multiplications can be done in two ways $A_{2\times1}$, $B_{1\times2}$, $C_{2\times4}$
  - (A*B)*C

  or
  - A*(B*C)

The final answer is going to be same, because the matrix multiplication have a property called an associativity.

$A_{2\times1}$, $B_{1\times2}$, $C_{2\times4}$

(A*B)*C

$A_{2\times1} * B_{1\times2} = D_{2\times2}$ (Total number of multiplications required is 2*1*2=4)

$D_{2\times2} * C_{2\times4} = E_{2\times4}$ (Total number of multiplications required is 2*2*4=16)

Total =16+4=20

A*(B*C)

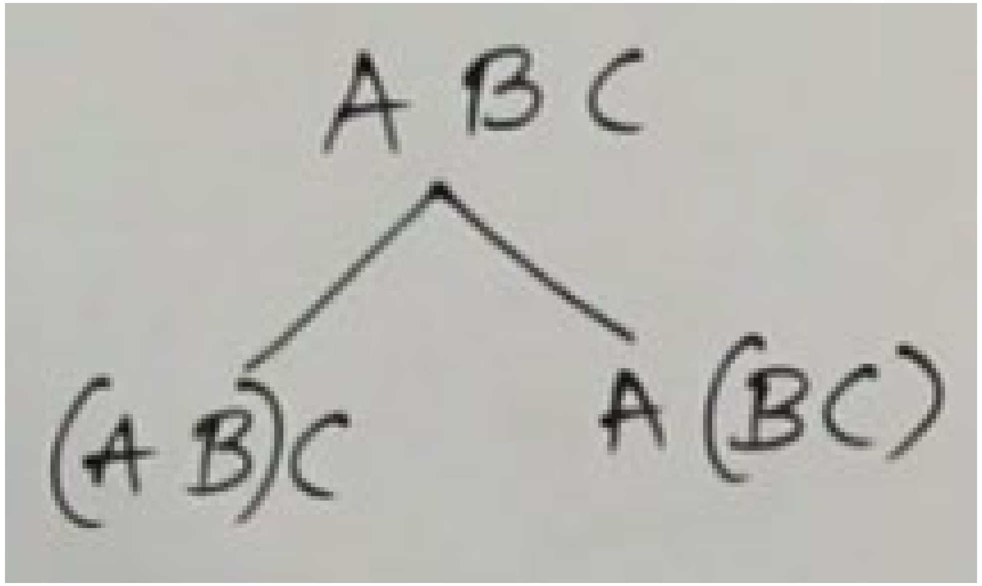
$B_{1\times2} * C_{2\times4} = D_{1\times4}$ (Total number of multiplications required is 1*2*4=8)

$A_{2\times1} * D_{1\times4} = E_{2\times4}$ (Total number of multiplications required is 2*1*4=8)

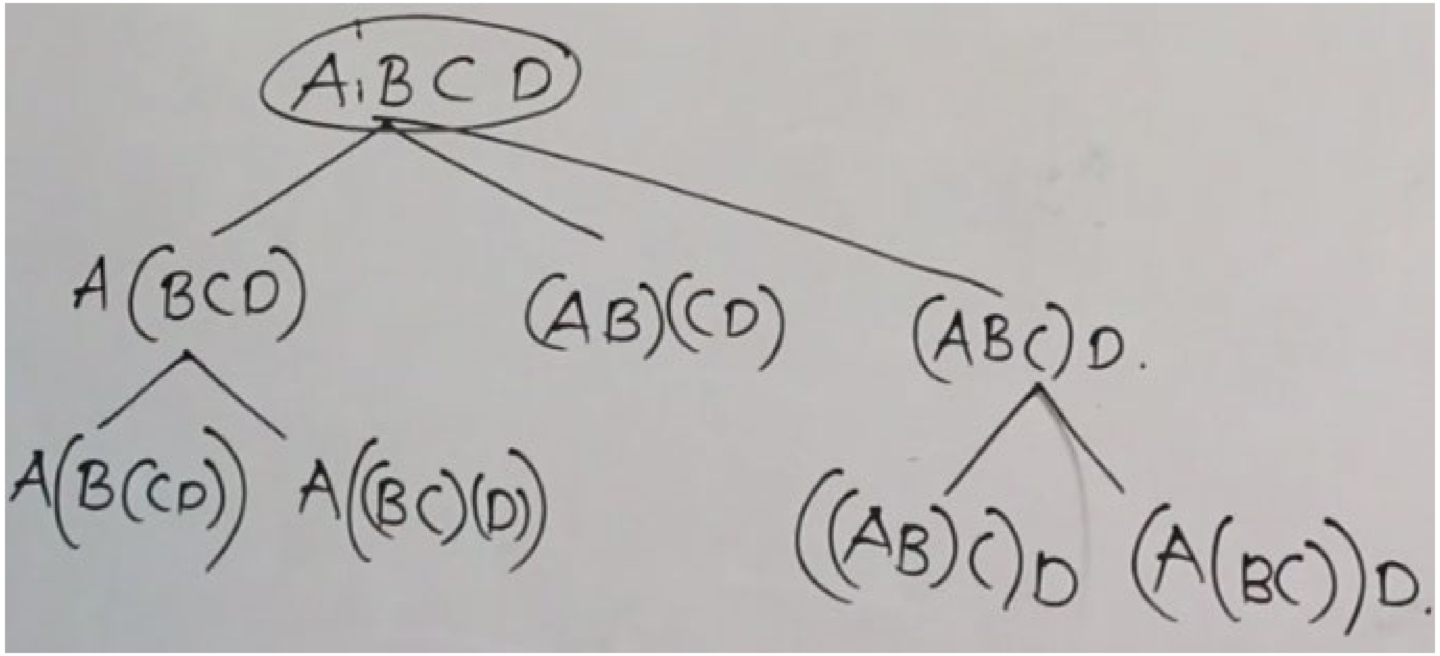
Total=8+8=16

Through these analysis, we can find out that we can save some cost. (That is number of multiplications)

If we have three matrices we have 2 ways multiply:

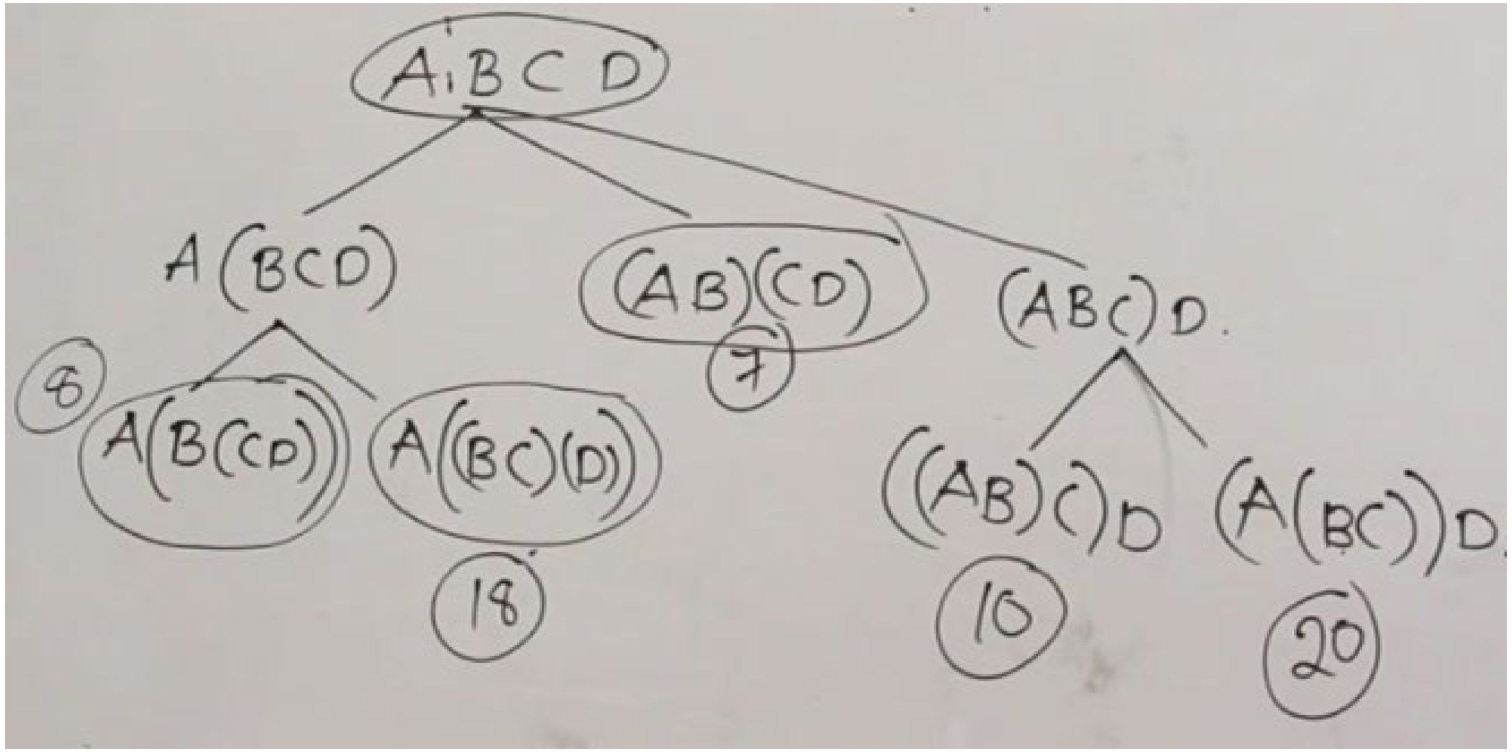


If we have more matrices (i.e., four matrices)



From about information, it is clear there are multiple ways to perform matrix multiplication. Then which one is best? Simple it depends on given instances.

$A_{1\times 2} \quad B_{2\times 1} \quad C_{1\times 4} \quad D_{4\times 1}$



$A(B(CD))$ — 1×2, 2×1, 1×1, 2×1

$\boxed{4+2+2} = 8$.

$A((BC)(D))$ — 1×2, 2×1, 2×4, 4×1

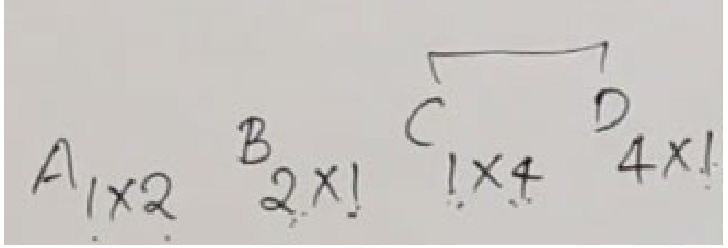
$8+8+2 = 18$.

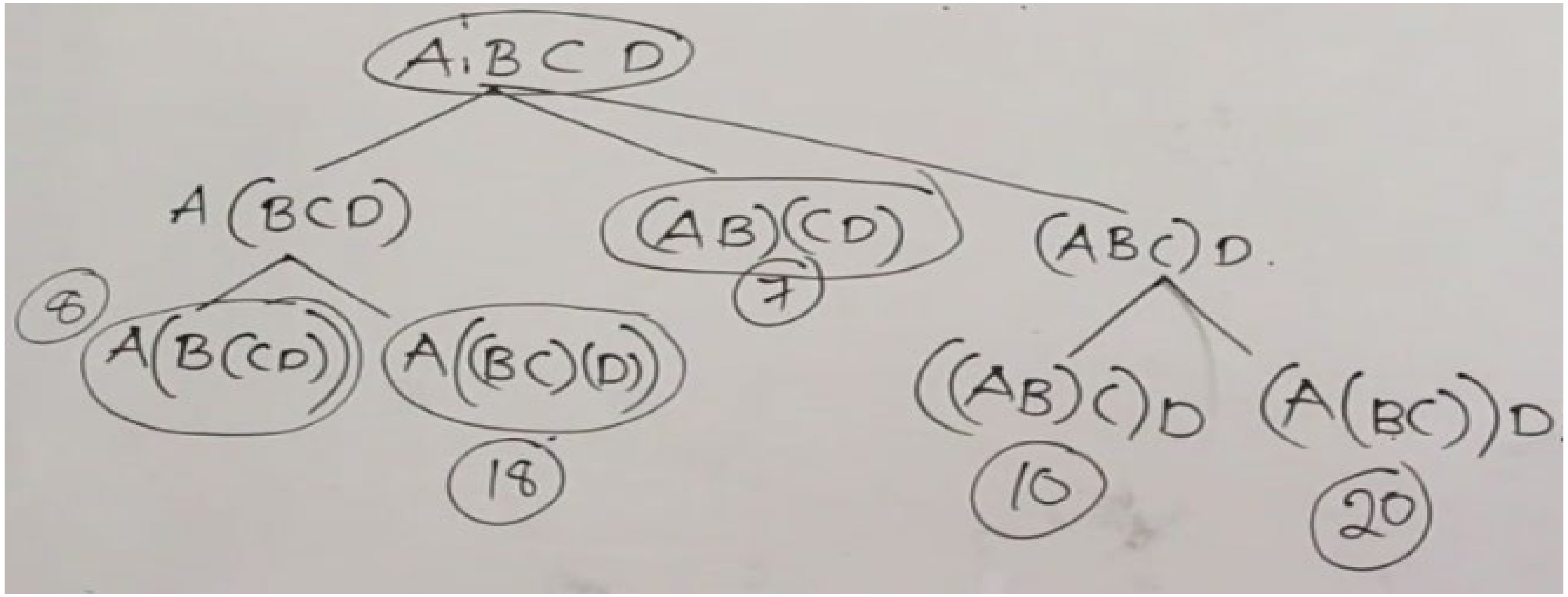
$(AB)(CD)$ — 1×1, 1×1

$2+4+1 = 7$

$((AB)C)D$ — 1×1, 1×4

$2+4+4$.



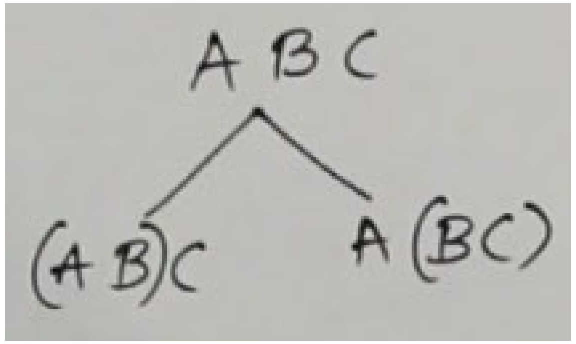


From about information, it is clear there are multiple ways to perform matrix multiplication. Then which one is best? Simple it depends on given instances.

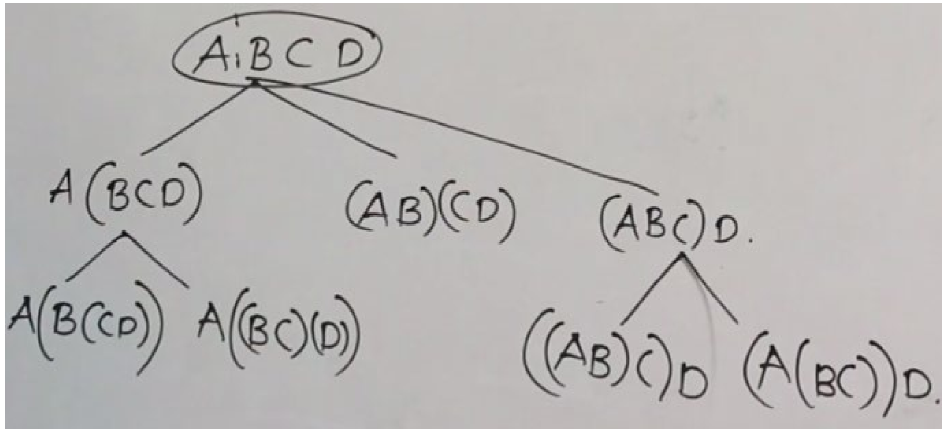
From the given instances: (AB)(CD)=7 operations is the best. In order to find out which one is best, here we evaluated all possible ways.

How many ways are there?

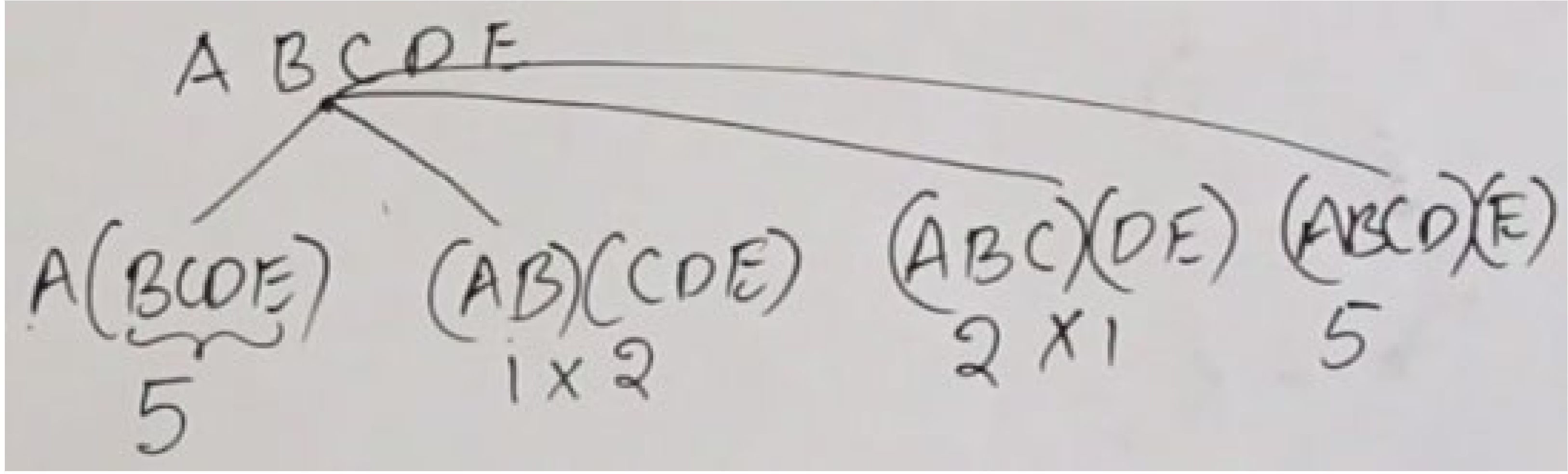
If we have 3 matrices, we have 2ways of matrix multiplication



If we have 4 matrices, we have 5 ways of matrix multiplications



If we have 5 matrices, we have 14 ways.



How many ways are there?

If we have 3 matrices, we have 2ways of matrix multiplication

If we have 4 matrices, we have 5 ways of matrix multiplications

If we have 5 matrices, we have 14 ways.

There is generalized formula to apply this.

The number of ways the matrices parenthesize is $= \dfrac{(2n)!}{(n+1)!\,n!}$

Here n=number martrices-1.

This formula is called Catalan number

How many ways are there? : Catalan number

The number of ways the matrices parenthesize is $= \dfrac{(2n)!}{(n+1)!\,n!}$

Here n=number martrices-1.

- If $n = 2$ (parenthesizing 3 matrices), then the number of ways is $\frac{(2\cdot2)!}{(2+1)!\cdot2!} = \frac{4!}{3!\cdot2!} = 2$ ways.

- If $n = 3$ (parenthesizing 4 matrices), then the number of ways is $\frac{(2\cdot3)!}{(3+1)!\cdot3!} = \frac{6!}{4!\cdot3!} = 5$ ways.

- If $n = 4$ (parenthesizing 5 matrices), then the number of ways is $\frac{(2\cdot4)!}{(4+1)!\cdot4!} = \frac{8!}{5!\cdot4!} = 14$ ways.

Matrix Chain Multiplication Algorithm in Dynamic Programming

Definition: Matrix Chain Multiplication is an optimization problem that involves finding the most efficient way to multiply a given sequence of matrices. The goal is to minimize the total number of scalar multiplications required to compute the product.

Procedure:

1. Input: A sequence of matrices $A_1, A_2, \ldots, A_n$
   - where the dimensions of the i^{th} matrix are $p_{i-1} \times p_i$.
2. Objective: To find the most efficient way to parenthesize the matrices to minimize the total number of scalar multiplications.
3. Dynamic Programming Approach: The problem can be solved using dynamic programming by breaking it into smaller subproblems and building up the solution.

Sample Input and Output:

- **Input**: Dimensions of matrices $p = [p_0, p_1, \ldots, p_n]$ where p_i represents the number of rows in matrix i and p_{i+1} represents the number of columns.
- **Output**: An optimal parenthesization that minimizes the total number of scalar multiplications

Matrix Chain Multiplication Algorithm in Dynamic Programming

Time Complexity in Brute Force Approach:

- In a brute force approach, the time complexity is exponential as it involves trying out all possible parenthesizations and selecting the one with the minimum number of scalar multiplications. It's $O(2^n)$, where n is the number of matrices.

Time Complexity in Dynamic Programming:

- Dynamic programming significantly reduces the time complexity by storing and reusing solutions to subproblems.
- The time complexity is $O(n^3)$, where n is the number of matrices. This is achieved by constructing a table to store intermediate results.

Matrix Chain Multiplication Algorithm in Dynamic Programming

Time Complexity in Dynamic Programming:

- Dynamic programming significantly reduces the time complexity by storing and reusing solutions to subproblems.
- The time complexity is $O(n^3)$, where n is the number of matrices. This is achieved by constructing a table to store intermediate results.

Recurrence Relation:

The recurrence relation for the Matrix Chain Multiplication problem is often defined as follows:

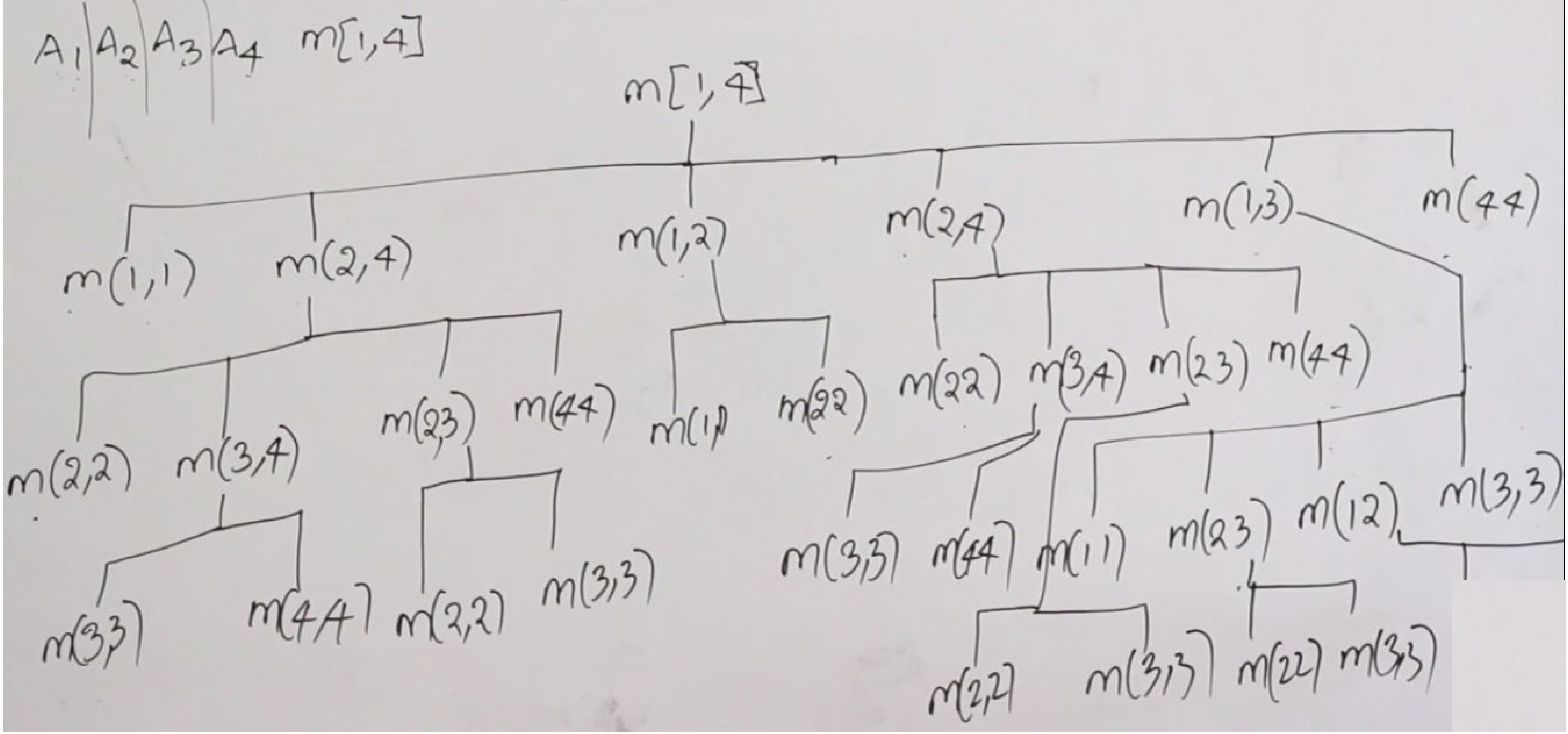
$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

Here, $M[i, j]$ represents the minimum number of scalar multiplications needed to compute the product $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$.

Using dynamic Programming

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

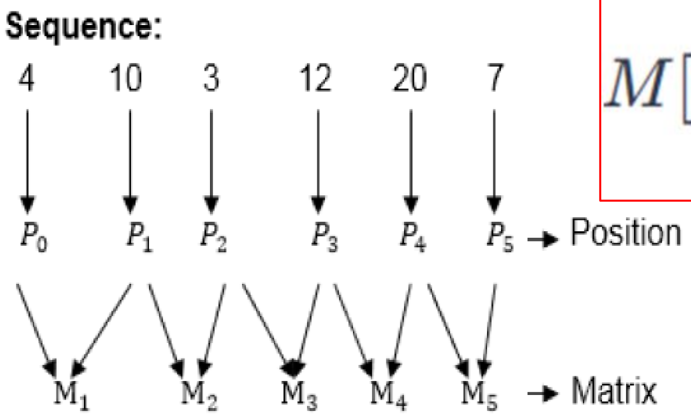
Recursion tree



Matrix Chain Multiplication Algorithm in Dynamic Programming
Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}.

- The matrices have size 4 x10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i]= 0 for all i.



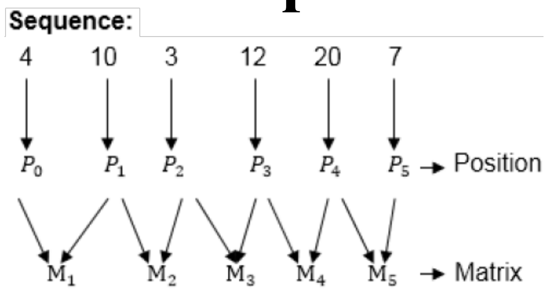
$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

- $M_1 : 4 \times 10$
- $M_2 : 10 \times 3$
- $M_3 : 3 \times 12$
- $M_4 : 12 \times 20$
- $M_5 : 20 \times 7$

1	2	3	4	5	
0					1
	0				2
		0			3
			0		4
				0	5

Matrix Chain Multiplication Algorithm in Dynamic Programming
Example of Matrix Chain Multiplication

1	2	3	4	5	
0					1
	0				2
		0			3
			0		4
				0	5



Here P_0 to P_5 are Position and M_1 to M_5 are matrix of size (p_i to p_{i-1}) On the basis of sequence, we make a formula

- $M_1 : 4 \times 10$
- $M_2 : 10 \times 3$
- $M_3 : 3 \times 12$
- $M_4 : 12 \times 20$
- $M_5 : 20 \times 7$

For $M_i \rightarrow$ p [i] as column
p [i-1] as row

Calculation of Product of 2 matrices:

1. m (1,2) = m_1 x m_2
   = 4 x 10 x 10 x 3
   = 4 x 10 x 3 = 120

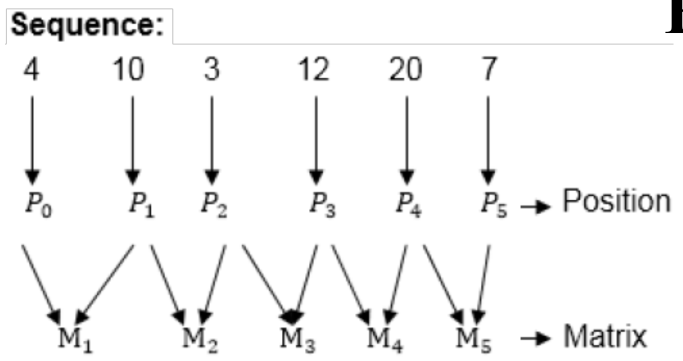
2. m (2, 3) = m_2 x m_3
   = 10 x 3 x 3 x 12
   = 10 x 3 x 12 = 360

3. m (3, 4) = m_3 x m_4
   = 3 x 12 x 12 x 20
   = 3 x 12 x 20 = 720

4. m (4,5) = m_4 x m_5
   = 12 x 20 x 20 x 7
   = 12 x 20 x 7 = 1680

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

Matrix Chain Multiplication Algorithm in Dynamic Programming
Example of Matrix Chain Multiplication



1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264			1
	0	360			2
		0	720		3
			0	1680	4
				0	5

HOW

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

<u>Now product of 3 matrices</u>: M [1, 3] = M1 M2 M3

There are two cases by which we can solve this multiplication:

- (M1 x M2)+ M3,
- M1 + (M2 x M3)

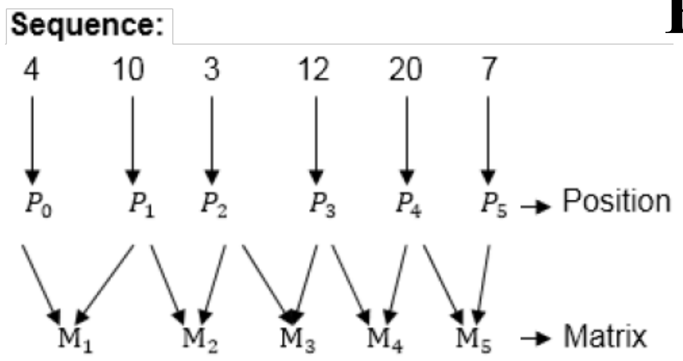
After solving both cases, choose the case in which minimum output is there.

$$M[1,3] = \min \begin{cases} M[1,2] + M[3,3] + p_0 p_2 p_3 = 120 + 0 + 4.3.12 = 264 \\ M[1,1] + M[2,3] + p_0 p_1 p_3 = 0 + 360 + 4.10.12 = 840 \end{cases}$$

- M [1, 3] =264

As Comparing both output **264** is minimum in both cases so we insert 264 in table and (M1 x M2) + M3 this combination is chosen for the output making.

Matrix Chain Multiplication Algorithm in Dynamic Programming
Example of Matrix Chain Multiplication



	1	2	3	4	5
1	0	120	264		
2		0	360		
3			0	720	
4				0	1680
5					0

	1	2	3	4	5
1	0	120	264		
2		0	360	1320	
3			0	720	
4				0	1680
5					0

HOW

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

<u>Now product of 3 matrices</u>: M [2, 4] = M2 M3 M4

There are two cases by which we can solve this multiplication:

- M2x M3)+M4
- M2+(M3 x M4)

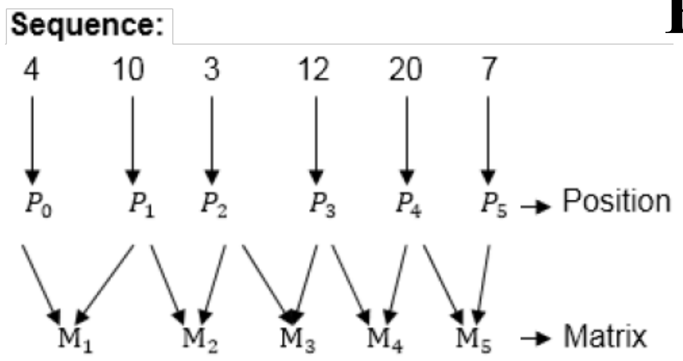
After solving both cases, choose the case in which minimum output is there.

$$M[2,4] = \min \begin{cases} M[2,3] + M[4,4] + p_1p_3p_4 = 360 + 0 + 10.12.20 = 2760 \\ M[2,2] + M[3,4] + p_1p_2p_4 = 0 + 720 + 10.3.20 = 1320 \end{cases}$$

- M [2, 4] = 1320
- As Comparing both output 1320 is minimum in both cases so we insert 1320 in table and M2+(M3 x M4) this combination is chosen for the output making.

Matrix Chain Multiplication Algorithm in Dynamic Programming

Example of Matrix Chain Multiplication



	1	2	3	4	5
1	0	120	264		
2		0	360	1320	
3			0	720	
4				0	1680
5					0

	1	2	3	4	5
1	0	120	264		
2		0	360	1320	
3			0	720	1140
4				0	1680
5					0

HOW

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1}\cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

<u>Now product of 3 matrices</u>: M [3, 5] = M3 M4 M5

There are two cases by which we can solve this multiplication:

- (M3 x M4) x M5
- M3x (M4xM5)

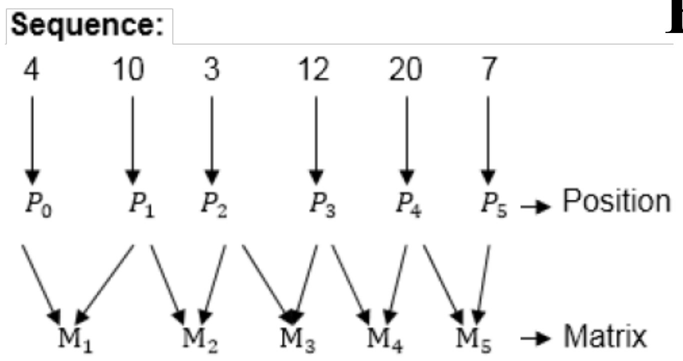
After solving both cases, choose the case in which minimum output is there.

$$M[3,5] = \min \begin{cases} M[3,4] + M[5,5] + p_2p_4p_5 = 720 + 0 + 3.20.7 = 1140 \\ M[3,3] + M[4,5] + p_2p_3p_5 = 0 + 1680 + 3.12.7 = 1932 \end{cases}$$

- M [3, 5] = 1140
- As Comparing both output 1140 is minimum in both cases so we insert 1140 in table and **(M3xM4)+M5** this combination is chosen for the output making.

Matrix Chain Multiplication Algorithm in Dynamic Programming

Example of Matrix Chain Multiplication



→

	1	2	3	4	5
1	0	120	264		
2		0	360	1320	
3			0	720	1140
4				0	1680
5					0

→

	1	2	3	4	5
1	0	120	264	1080	
2		0	360	1320	
3			0	720	1140
4				0	1680
5					0

HOW

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

<u>Now product of 4 matrices:</u> M [1,4] = M1 M2 M3 M4
There are three cases by which we can solve this multiplication:

- (M1 x M2 x M3) M4
- M1 x(M2 x M3 x M4)
- (M1 xM2) x (M3 x M4)

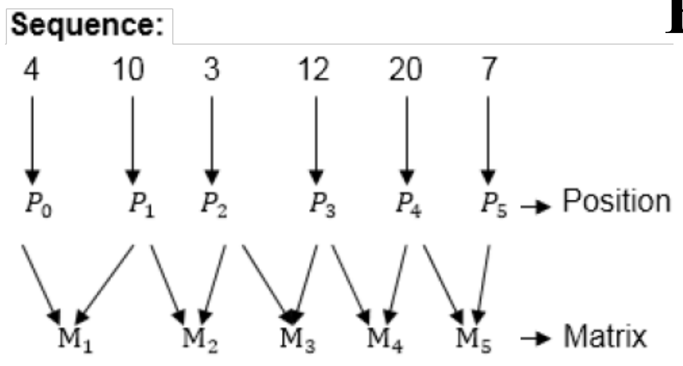
After solving three cases, choose the case in which minimum output is there.

$$M[1,4] = \min \begin{Bmatrix} M[1,3] + M[4,4] + p_0p_3p_4 = 264 + 0 + 4.12.20 = 1224 \\ M[1,2] + M[3,4] + p_0p_2p_4 = 120 + 720 + 4.3.20 = 1080 \\ M[1,1] + M[2,4] + p_0p_1p_4 = 0 + 1320 + 4.10.20 = 2120 \end{Bmatrix}$$

- M [1, 4] =**1080**
- As comparing the output of different cases then '**1080**' is minimum output, so we insert 1080 in the table and (M1 xM2) x (M3 x M4) combination is taken out in output making.

Matrix Chain Multiplication Algorithm in Dynamic Programming

Example of Matrix Chain Multiplication



1	2	3	4	5	
0	120	264	1080		1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

HOW

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

Now product of 4 matrices: M [2, 5] = M2 M3 M4 M5
There are three cases by which we can solve this multiplication:

- (M2 x M3 x M4)x M5
- M2 x(M3 x M4 x M5)
- (M2 x M3)x (M4 x M5)

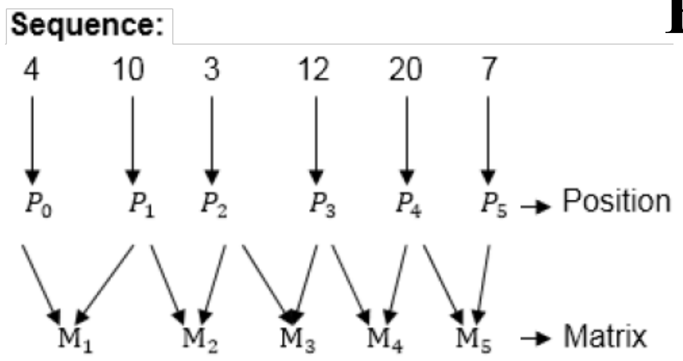
After solving three cases, choose the case in which minimum output is there.

$$M[2,5] = \min \begin{cases} M[2,4] + M[5,5] + p_1p_4p_5 = 1320 + 0 + 10.20.7 = & 2720 \\ M[2,3] + M[4,5] + p_1p_3p_5 = 360 + 1680 + 10.12.7 = & 2880 \\ M[2,2] + M[3,5] + p_1p_2p_5 = 0 + 1140 + 10.3.7 = & 1350 \end{cases}$$

- M [2, 5] = 1350
- As comparing the output of different cases then '**1350**' is minimum output, so we insert 1350 in the table and **M2 x(M3 x M4 xM5)** combination is taken out in output making.

Matrix Chain Multiplication Algorithm in Dynamic Programming

Example of Matrix Chain Multiplication



1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080	1344	1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

HOW

$$M[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j}\{M[i,k] + M[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\} & \text{if } i < j \end{cases}$$

<u>Now product of 5 matrices:</u> M [1, 5] = M [1, 5] = M1 M2 M3 M4 M5
There are Four cases by which we can solve this multiplication:

- (M1 x M2 xM3 x M4)+M5
- M1 +(M2 xM3 x M4 xM5)
- (M1 x M2 xM3)+ (M4 xM5)
- (M1 x M2)+(M3 x M4 xM5)

After solving four cases, choose the case in which minimum output is there.

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + p_0p_4p_5 = 1080 + 0 + 4.20.7 = 1544 \\ M[1,3] + M[4,5] + p_0p_3p_5 = 264 + 1680 + 4.12.7 = 2016 \\ M[1,2] + M[3,5] + p_0p_2p_5 = 120 + 1140 + 4.3.7 = 1344 \\ M[1,1] + M[2,5] + p_0p_1p_5 = 0 + 1350 + 4.10.7 = 1630 \end{cases}$$

- ➢ M [1, 5] = 1344
- ➢ As comparing the output of different cases then '1344' is minimum output, so we insert 1344 in the table and
  - ➢ **(M1 x M2) +(M3 x M4 x M5)** combination is taken out in output making.

Matrix Chain Multiplication Algorithm in Dynamic Programming
Example of Matrix Chain Multiplication

	1	2	3	4	5	
	0	120	264	1080	1344	1
		0	360	1320	1350	2
			0	720	1140	3
				0	1680	4
					0	5

M [1, 5] = M [1, 5] = M1 M2 M3 M4 M5

There are Four cases by which we can solve this multiplication:

- (M1 x M2 xM3 x M4)x M5
- M1 x(M2 xM3 x M4 xM5)
- (M1 x M2 xM3)x M4 xM5
- M1 x M2x(M3 x M4 xM5)

$$M[1,5] = \min \begin{cases} M[1,4] + M[5,5] + p_0p_4p_5 = 1080 + 0 + 4.20.7 = 1544 \\ M[1,3] + M[4,5] + p_0p_3p_5 = 264 + 1680 + 4.12.7 = 2016 \\ M[1,2] + M[3,5] + p_0p_2p_5 = 120 + 1140 + 4.3.7 = 1344 \\ M[1,1] + M[2,5] + p_0p_1p_5 = 0 + 1350 + 4.10.7 = 1630 \end{cases}$$

- As comparing the output of different cases then '**1344**' is minimum output, so we insert 1344 in the table and
  - **M1 x M2 x(M3 x M4 x M5)** combination is taken out in output making.

M [3, 5] = M3 M4 M5

There are two cases by which we can solve this multiplication:

(M3 x M4) + M5

M3+ (M4xM5)

$$M[3,5] = \min \begin{cases} M[3,4] + M[5,5] + p_2p_4p_5 = 720 + 0 + 3.20.7 = 1140 \\ M[3,3] + M[4,5] + p_2p_3p_5 = 0 + 1680 + 3.12.7 = 1932 \end{cases}$$

As Comparing both output 1140 is minimum in both cases so we insert 1140 in table and (M3xM4) xM5 this combination is chosen for the output making.

The final multiplication order is M1 x M2 x((M3xM4)xM5 **)** combination is taken out in output making.

Time Complexity analysis

(1 1)	(22)	(3 3)	(4 4)
(1 2)	(23)	(3 4)	
(1 3)	(2,4)		
(1,4)			

Example-2

$m_1 \quad m_2 \quad m_3 \quad m_4$

$(10\times100)(100\times20)\ (20\times5)\ (5\times80)$

$(((1)(2\ 3))(4))$

19000.

Other Supporting Martials for multiplicatio

Example

- Show how to multiply this matrix chain optimally
- Solution on the board
    - Minimum cost 15,125
    - Optimal parenthesization $((A_1(A_2A_3))((A_4 A_5)A_6))$

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

Example:

$$
\begin{array}{lll}
A_1 & 30 \times 35 & = p_0 \times p_1 \\
A_2 & 35 \times 15 & = p_1 \times p_2 \\
A_3 & 15 \times 5 & = p_2 \times p_3 \\
A_4 & 5 \times 10 & = p_3 \times p_4 \\
A_5 & 10 \times 20 & = p_4 \times p_5 \\
A_6 & 20 \times 25 & = p_5 \times p_6
\end{array}
$$

Order of matrix computations

$m(1,1)$ $m(1,2)$ $m(1,3)$ $m(1,4)$ $m(1,5)$ $m(1,6)$

$m(2,2)$ $m(2,3)$ $m(2,4)$ $m(2,5)$ $m(2,6)$ - length 6

$m(3,3)$ $m(3,4)$ $m(3,5)$ $m(3,6)$ - length 5

$m(4,4)$ $m(4,5)$ $m(4,6)$ - length 4

$m(5,5)$ $m(5,6)$ - length 3

$m(6,6)$ - length 2

- length 1

$m[2,5]=$

min{

$m[2,2]+m[3,5]+p_1p_2p_5=0+2500+35\times15\times20=13000,$

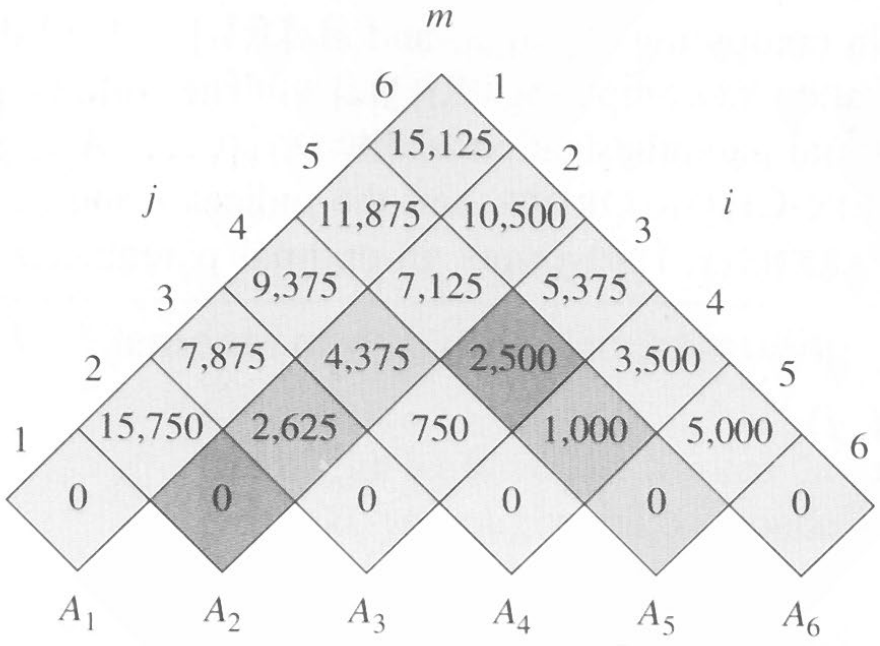
$m[2,3]+m[4,5]+p_1p_3p_5=2625+1000+35\times5\times20=7125,$

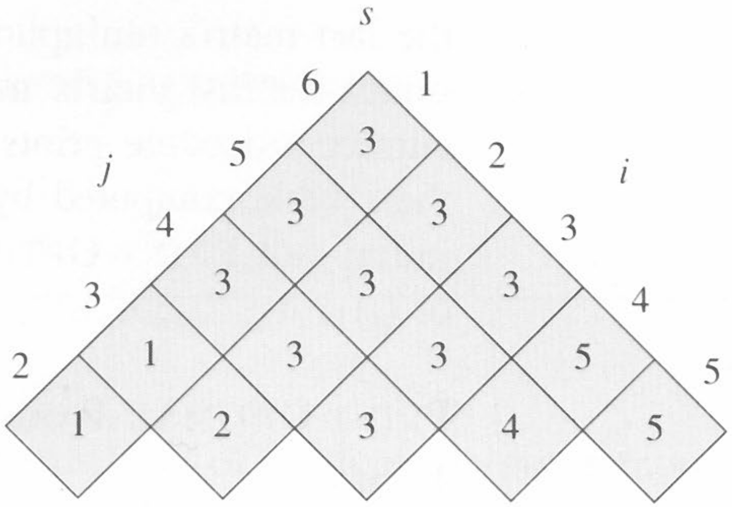
$m[2,4]+m[5,5]+p_1p_4p_5=4375+0+35\times10\times20=11374$

}

$=7125$

the m and s table computed by
MATRIX-CHAIN-ORDER for n=6





Step 4:
Constructing an optimal solution

```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

- example:

$$((A_1(A_2A_3))((A_4A_5)A_6))$$

chains of length 1

i \ j	1	2	3	4
1	0			
2		0		
3			0	
4				0

A_1	30×1
A_2	1×40
A_3	40×10
A_4	10×25

chains of length 2

j / i	1	2	3	4
1	0	1200 1		
2		0	400 2	
3			0	10000 3
4				0

A_1	30×1
A_2	1×40
A_3	40×10
A_4	10×25

$$m[1,2] = m[1,1] + m[2,2] + 30 \cdot 1 \cdot 40 = 1200$$

$$m[2,3] = m[2,2] + m[3,3] + 1 \cdot 40 \cdot 10 = 400$$

$$m[3,4] = m[3,3] + m[4,4] + 40 \cdot 10 \cdot 25 = 10000$$

chains of length 3

i \ j	1	2	3	4
1	0	1200 1	700 1	
2		0	400 2	650 3
3			0	10000 3
4				0

A_1	30×1
A_2	1×40
A_3	40×10
A_4	10×25

$$m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + 30 \cdot 1 \cdot 10 = 700 \\ m[1,2] + m[3,3] + 30 \cdot 40 \cdot 10 = 13200 \end{cases} = 700$$

$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + 1 \cdot 40 \cdot 25 = 11000 \\ m[2,3] + m[4,4] + 1 \cdot 10 \cdot 25 = 650 \end{cases} = 650$$

chains of length 4

i \ j	1	2	3	4
1	0	1200 1	700 1	1400 1
2		0	400 2	650 3
3			0	10000 3
4				0

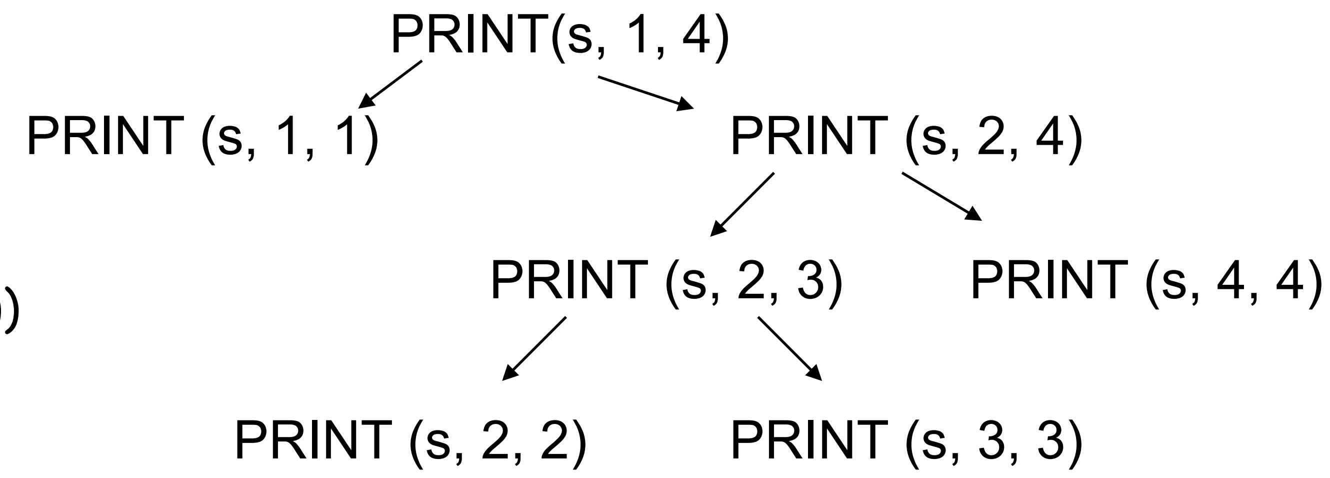
A_1	30×1
A_2	1×40
A_3	40×10
A_4	10×25

$$m[1,4]=\min\begin{cases} m[1,1]+m[2,4]+30\cdot 1\cdot 25=1400 \\ m[1,2]+m[3,4]+30\cdot 40\cdot 25=41200=1400 \\ m[1,3]+m[4,4]+30\cdot 10\cdot 25=8200 \end{cases}$$

Printing the solution

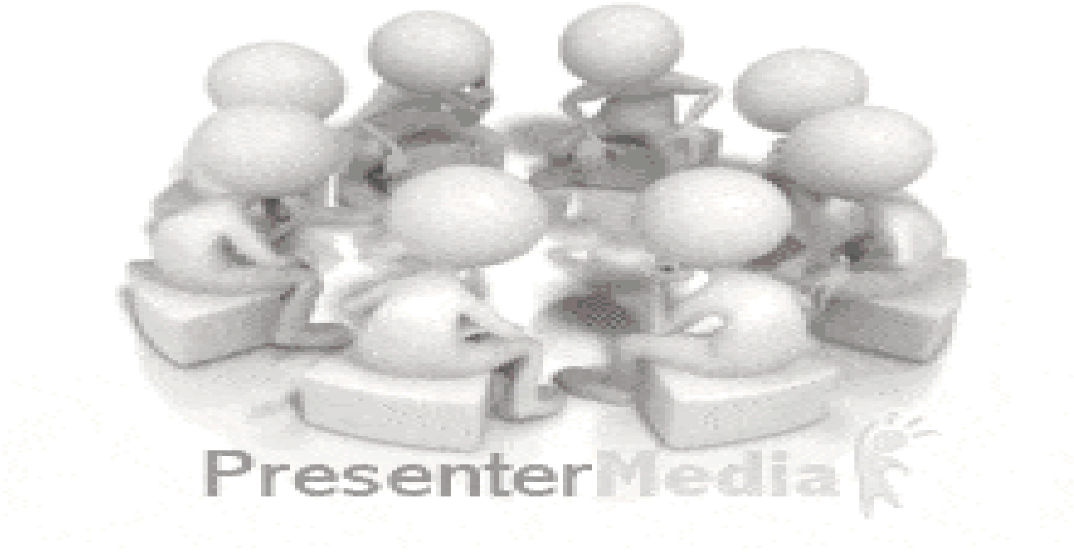
i \ j	2	3	4
1	1	1	1
2		2	3
3			3

A_1	30×1
A_2	1×40
A_3	40×10
A_4	10×25



Output: $(A_1((A_2A_3)A_4))$

Any Question




Thank You!


FOR YOUR
ATTENTION