Amirkabir University of Technology
(Tehran Polytechnic)

Department of
Computer Engineering

Data Structure & Algorithms

Matrix-Chain Multiplication

Matrix-Chain Multiplication

Problem: given a sequence $\langle A_1, A_2, \ldots, A_n \rangle$, compute the product:

$$A_1 \cdot A_2 \cdots A_n$$

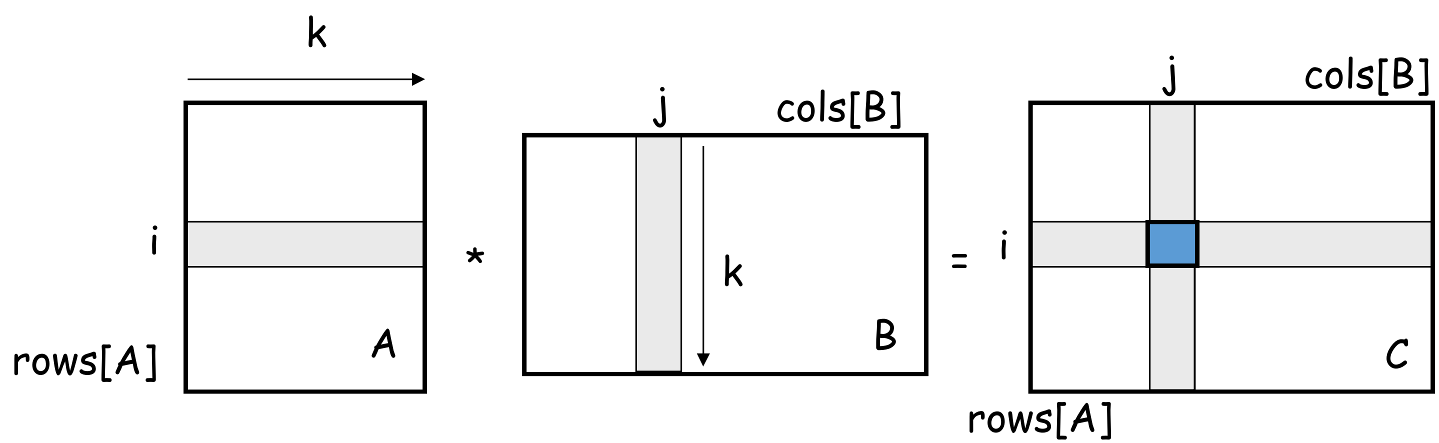
- Matrix compatibility:

$C = A \cdot B$	$C = A_1 \cdot A_2 \cdots A_i \cdot A_{i+1} \cdots A_n$
$col_A = row_B$	$col_i = row_{i+1}$
$row_C = row_A$	$row_C = row_{A1}$
$col_C = col_B$	$col_C = col_{An}$

MATRIX-MULTIPLY(A, B)

```
if columns[A] ≠ rows[B]
   then error "incompatible dimensions"
   else for i ← 1 to rows[A]
             do for j ← 1 to columns[B]
                    do C[i, j] = 0
                         for k ← 1 to columns[A]
                               do C[i, j] ← C[i, j] + A[i, k] B[k, j]
```

rows[A] · cols[A] · cols[B]
multiplications



Matrix-Chain Multiplication

- In what order should we multiply the matrices?

$$A_1 \cdot A_2 \cdots A_n$$

- Parenthesize the product to get the order in which matrices are multiplied
- *E.g.:* $A_1 \cdot A_2 \cdot A_3 = ((A_1 \cdot A_2) \cdot A_3)$

$$= (A_1 \cdot (A_2 \cdot A_3))$$

- Which one of these orderings should we choose?
  - The order in which we multiply the matrices has a significant impact on the cost of evaluating the product

Matrix-Chain Multiplication: Problem Statement

- Given a chain of matrices $\langle A_1, A_2, \ldots, A_n \rangle$, where A_i has dimensions $p_{i-1} x\ p_i$, fully parenthesize the product $A_1 \cdot A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

$$\underset{p_0 \text{ x } p_1}{A_1} \cdot \underset{p_1 \text{ x } p_2}{A_2} \cdots \underset{p_{i-1} \text{ x } p_i}{A_i} \cdot \underset{p_i \text{ x } p_{i+1}}{A_{i+1}} \cdots \underset{p_{n-1} \text{ x } p_n}{A_n}$$

What is the number of possible parenthesizations?

- Exhaustively checking all possible parenthesizations is not efficient!
- It can be shown that the number of parenthesizations grows as $\Omega(4^n/n^{3/2})$ (see page 333 in your textbook)

1. The Structure of an Optimal Parenthesization

- Notation:

$$A_{i\ldots j} = A_i\, A_{i+1} \cdots A_j,\ i \le j$$

- Suppose that an optimal parenthesization of $A_{i\ldots j}$ splits the product between A_k and A_{k+1}, where $i \le k < j$

$$\begin{aligned} A_{i\ldots j} &= A_i\, A_{i+1} \cdots A_j \\ &= A_i\, A_{i+1} \cdots A_k\, A_{k+1} \cdots A_j \\ &= A_{i\ldots k}\, A_{k+1\ldots j} \end{aligned}$$

Optimal Substructure

$$A_{i\ldots j} = A_{i\ldots k} A_{k+1\ldots j}$$

- The parenthesization of the "prefix" $A_{i\ldots k}$ must be an optimal parentesization
- If there were a less costly way to parenthesize $A_{i\ldots k}$, we could substitute that one in the parenthesization of $A_{i\ldots j}$ and produce a parenthesization with a lower cost than the optimum $\Rightarrow$ contradiction!
- *An optimal solution to an instance of the matrix-chain multiplication contains within it optimal solutions to subproblems*

2. A Recursive Solution

- Subproblem:

  determine the minimum cost of parenthesizing $A_{i\ldots j} = A_i A_{i+1} \cdots A_j$ for $1 \leq i \leq j \leq n$

- Let m[i, j] = the minimum number of multiplications needed to compute $A_{i\ldots j}$
  - full problem ($A_{1..n}$): m[1, n]
  - i = j: $A_{i\ldots i} = A_i \Rightarrow m[i, i] = 0$, for i = 1, 2, …, n

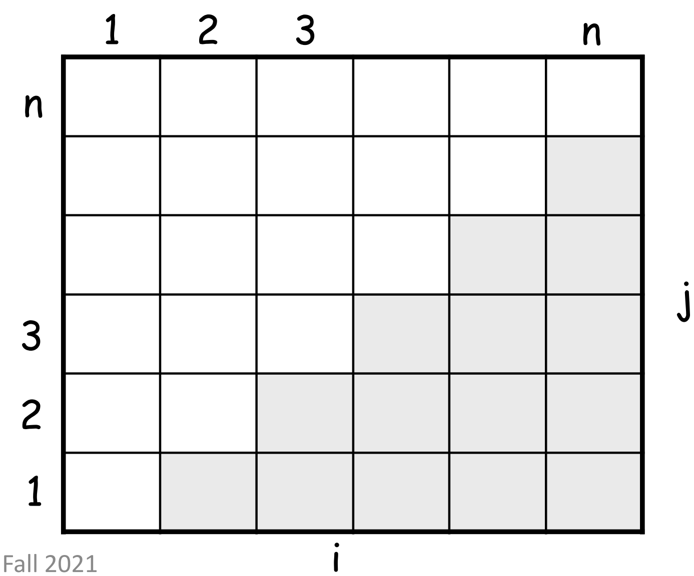
3. Computing the Optimal Costs

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

- Computing the optimal solution recursively takes exponential time!
- How many subproblems?

$\Rightarrow \Theta(n^2)$

  - Parenthesize $A_{i\ldots j}$ for $1 \le i \le j \le n$
  - One problem for each choice of i and j



3. Computing the Optimal Costs (cont.)

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

- How do we fill in the tables m[1..n, 1..n]?
  - Determine which entries of the table are used in computing $m[i, j]$

$$A_{i\ldots j} = A_{i\ldots k}\, A_{k+1\ldots j}$$

  - Subproblems' size is one less than the original size
  - **Idea:** fill in m such that it corresponds to solving problems of increasing length

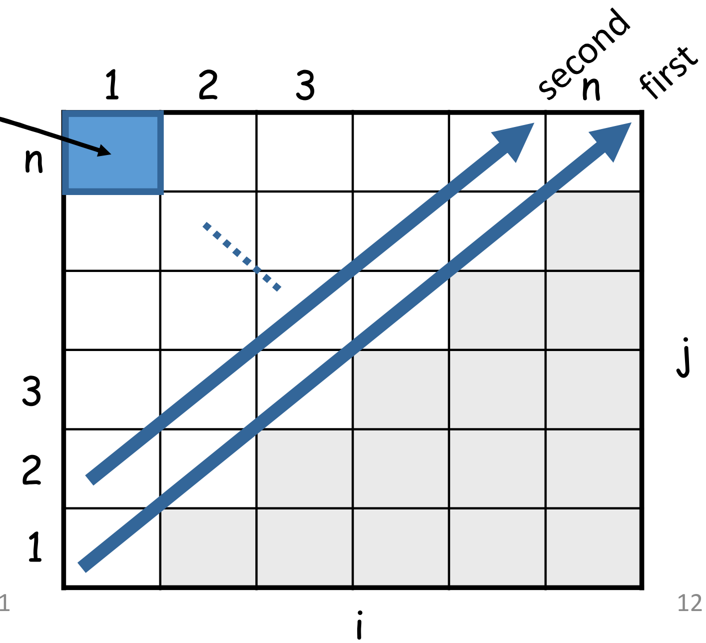
3. Computing the Optimal Costs (cont.)

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

- Length = 1: $i = j$, $i = 1, 2, \ldots, n$
- Length = 2: $j = i + 1$, $i = 1, 2, \ldots, n-1$

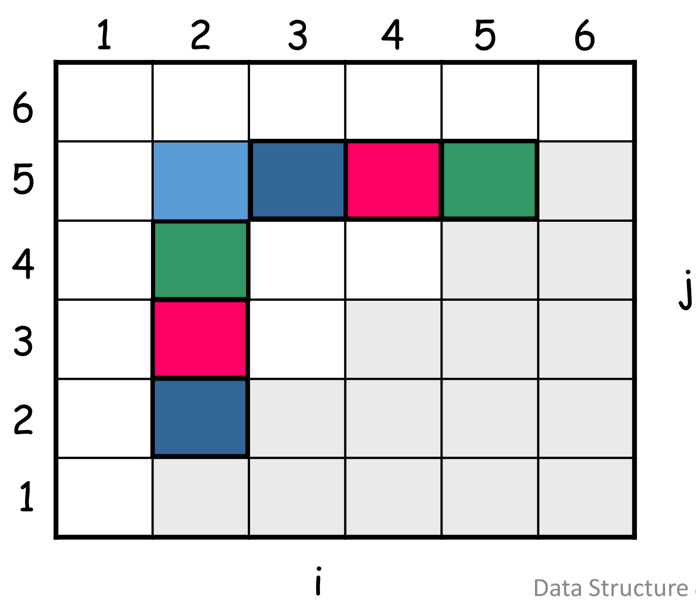
m[1, n] gives the optimal solution to the problem

Compute rows from bottom to top and from left to right



Example: $\min \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\}$

$$m[2, 5] = \min \begin{cases} m[2, 2] + m[3, 5] + p_1p_2p_5 & k = 2 \\ m[2, 3] + m[4, 5] + p_1p_3p_5 & k = 3 \\ m[2, 4] + m[5, 5] + p_1p_4p_5 & k = 4 \end{cases}$$



- Values m[i, j] depend only on values that have been previously computed

Example: min {m[i, k] + m[k+1, j] + $p_{i-1}p_kp_j$}

Compute $A_1 \cdot A_2 \cdot A_3$

- A_1: 10 x 100 (p_0 x p_1)
- A_2: 100 x 5 (p_1 x p_2)
- A_3: 5 x 50 (p_2 x p_3)

m[i, i] = 0 for i = 1, 2, 3

m[1, 2] = m[1, 1] + m[2, 2] + $p_0p_1p_2$ (A_1A_2)

= 0 + 0 + 10 *100* 5 = 5,000

m[2, 3] = m[2, 2] + m[3, 3] + $p_1p_2p_3$ (A_2A_3)

= 0 + 0 + 100 * 5 * 50 = 25,000

m[1, 3] = min m[1, 1] + m[2, 3] + $p_0p_1p_3$ = 75,000 ($A_1(A_2A_3)$)

m[1, 2] + m[3, 3] + $p_0p_2p_3$ = 7,500 ($(A_1A_2)A_3$)

3	2 7500	2 25000	0
2	1 5000	0	
1	0		

Matrix-Chain-Order(p)

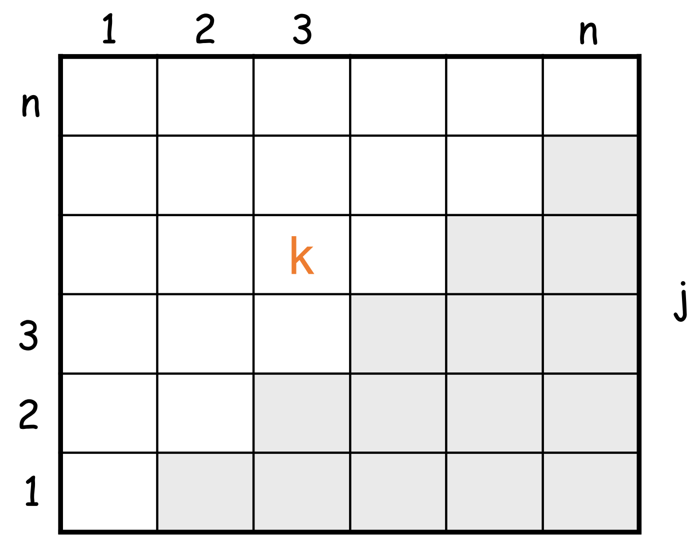
```
MATRIX-CHAIN-ORDER(p)
n ← length[p] − 1
for i ← 1 to n
     do m[i, i] ← 0
for l ← 2 to n                ▷ l is the chain length.
     do for i ← 1 to n − l + 1
              do j ← i + l − 1
                 m[i, j] ← ∞
                 for k ← i to j − 1
                     do q ← m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
                        if q < m[i, j]
                          then m[i, j] ← q
                               s[i, j] ← k
return m and s
```

$O(N^3)$

4. Construct the Optimal Solution

- In a similar matrix s we keep the optimal values of k

- s[i, j] = a value of k such that an optimal parenthesization of $A_{i..j}$ splits the product between A_k and A_{k+1}



4. Construct the Optimal Solution

- $s[i, j]$ = value of k such that the optimal parenthesization of $A_i\ A_{i+1} \cdots A_j$ splits the product between A_k and A_{k+1}

6	3	3	3	5	5	-
5	3	3	3	4	-	
4	3	3	3	-		
3	1	2	-			
2	1	-				
1	-					

j

i

- $s[1, n] = 3 \Rightarrow A_{1..6} = A_{1..3}\ A_{4..6}$
- $s[1, 3] = 1 \Rightarrow A_{1..3} = A_{1..1}\ A_{2..3}$
- $s[4, 6] = 5 \Rightarrow A_{4..6} = A_{4..5}\ A_{6..6}$

4. Construct the Optimal Solution

PRINT-OPT-PARENS(s, i, j)

if i = j

  then print “A”$_i$

  else print “(”

      PRINT-OPT-PARENS(s, i, s[i, j])

      PRINT-OPT-PARENS(s, s[i, j] + 1, j)

      print “)”

	1	2	3	4	5	6
6	3	3	3	5	5	-
5	3	3	3	4	-	
4	3	3	3	-		
3	1	2	-			
2	1	-				
1	-					

j

i

Example: $A_1 \cdot \cdot \cdot A_6$

$((A_1 (A_2 \; A_3)) ((A_4 A_5) A_6))$

s[1..6, 1..6]

j \ i	1	2	3	4	5	6
6	3	3	3	5	5	-
5	3	3	3	4	-	
4	3	3	3	-		
3	1	2	-			
2	1	-				
1	-					

P-O-P(s, 1, 6) **s[1, 6] = 3**

i = 1, j = 6 "(" **P-O-P (s, 1, 3)** **s[1, 3] = 1**

i = 1, j = 3 "(" **P-O-P(s, 1, 1)** $\Rightarrow$ **"A_1"**

P-O-P(s, 2, 3) **s[2, 3] = 2**

i = 2, j = 3 **"("** **P-O-P (s, 2, 2)** $\Rightarrow$ **"A_2"**

P-O-P (s, 3, 3) $\Rightarrow$ **"A_3"**

")"

")"

Matrix-Chain Multiplication - Summary

- The dynamic programming approach can solve the matrix-chain multiplication problem in $O(n^3)$
- This method take advantage of the overlapping subproblems property
- There are only $\Theta(n^2)$ different subproblems
  - Solutions to these problems are computed only once
- Without memoization the natural recursive algorithm runs in exponential time