Dynamic Programming
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:

$C = A \cdot B$

$col_A = row_B$

$row_C = row_A$

$col_C = col_B$

$C = A_1 \cdot A_2 \cdots A_i \cdot A_{i+1} \cdots A_n$

$col_i = row_{i+1}$

$row_C = row_{A1}$

$col_C = col_{An}$

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

Example

$$A_1 \cdot A_2 \cdot A_3$$

- A_1: 10 x 100
- A_2: 100 x 5
- A_3: 5 x 50

1. $((A_1 \cdot A_2) \cdot A_3)$: $A_1 \cdot A_2 = 10 \times 100 \times 5 = 5{,}000$ (10 x 5)

   $((A_1 \cdot A_2) \cdot A_3) = 10 \times 5 \times 50 = 2{,}500$

   Total: 7,500 scalar multiplications

2. $(A_1 \cdot (A_2 \cdot A_3))$: $A_2 \cdot A_3 = 100 \times 5 \times 50 = 25{,}000$ (100 x 50)

   $(A_1 \cdot (A_2 \cdot A_3)) = 10 \times 100 \times 50 = 50{,}000$

   Total: 75,000 scalar multiplications

one order of magnitude difference!!

Matrix-Chain Multiplication: Problem Statement

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

$$A_1 \cdot A_2 \cdots A_i \cdot A_{i+1} \cdots A_n$$

$$p_0 \times p_1 \quad p_1 \times p_2 \quad p_{i-1} \times p_i \quad p_i \times p_{i+1} \quad p_{n-1} \times p_n$$

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 $\Omega(4^n/n^{3/2})$

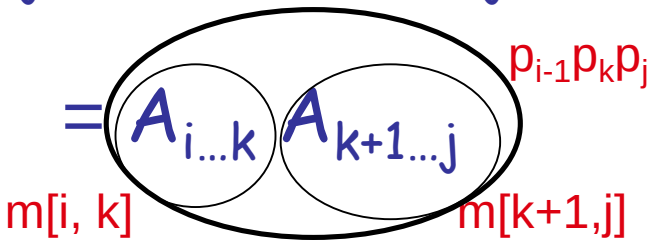
  (see page 333 in your textbook)

2. A Recursive Solution

- Consider the subproblem of parenthesizing

$$A_{i \ldots j} = A_i A_{i+1} \cdots A_j \qquad \text{for } 1 \le i \le j \le n$$

$$= A_{i \ldots k} \; A_{k+1 \ldots j} \qquad \text{for } i \le k < j$$



- Assume that the optimal parenthesization splits the product $A_i A_{i+1} \cdots A_j$ at k ($i \le k < j$)

$$m[i, j] = \underbrace{m[i, k]}_{\text{min \# of multiplications to compute } A_{i \ldots k}} + \underbrace{m[k+1, j]}_{\text{min \# of multiplications to compute } A_{k+1 \ldots j}} + \underbrace{p_{i-1}p_kp_j}_{\text{\# of multiplications to compute } A_{i \ldots k}A_{k \ldots j}}$$

2. A Recursive Solution (cont.)

$$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$$

- We do not know the value of k
  - There are j – i possible values for k: k = i, i+1, …, j-1
- Minimizing the cost of parenthesizing the product $A_i A_{i+1} \cdots A_j$ becomes:

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

3. Computing the Optimal Costs

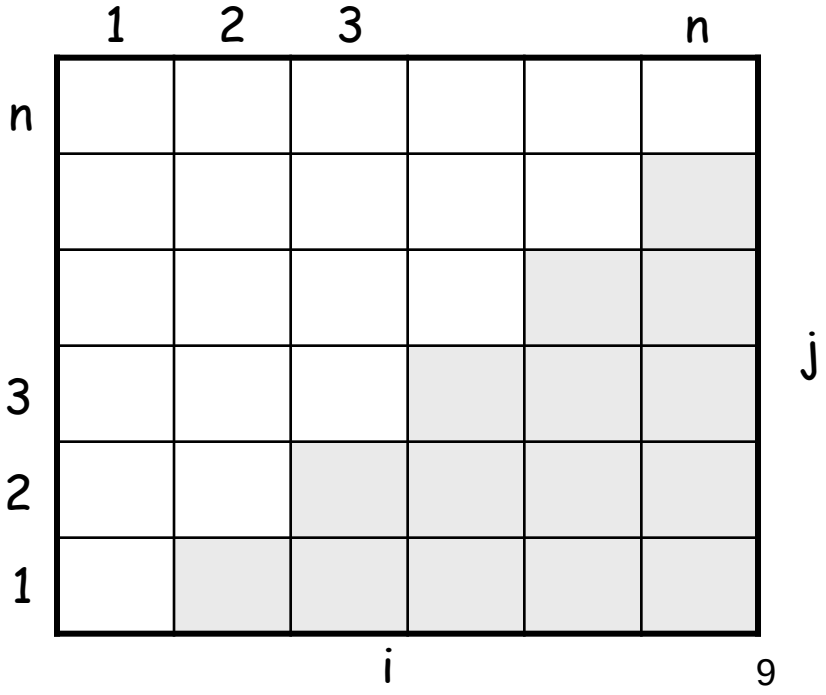
$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\} & \text{if } i < j \end{cases}$$

- 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.)

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

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

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

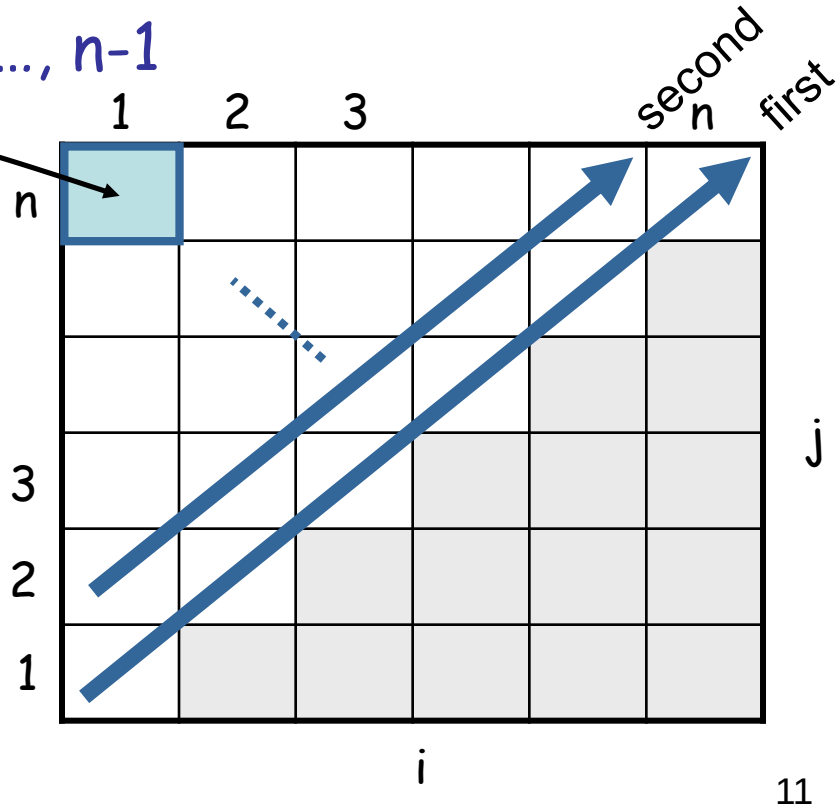
  - Subproblems' size is one less than the original size
  - **Idea:** fill in m such that it corresponds to solving problems of increasing length

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

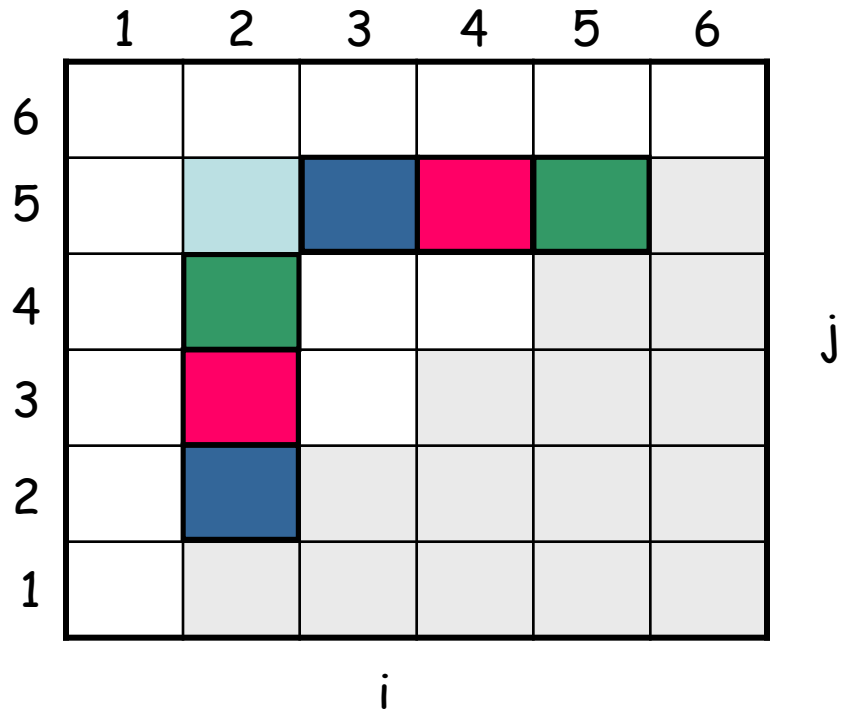
$m[1, n]$ gives the optimal solution to the problem



Compute rows from bottom to top and from left to right

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

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



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

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

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)

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

$m[i, i] = 0$ 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)

$= 0 + 0 + 100 * 5 * 50 = 25{,}000$

$m[1, 3] = \min \begin{cases} m[1, 1] + m[2, 3] + p_0p_1p_3 = 75{,}000 & (A_1(A_2A_3)) \\ m[1, 2] + m[3, 3] + p_0p_2p_3 = 7{,}500 & ((A_1A_2)A_3) \end{cases}$

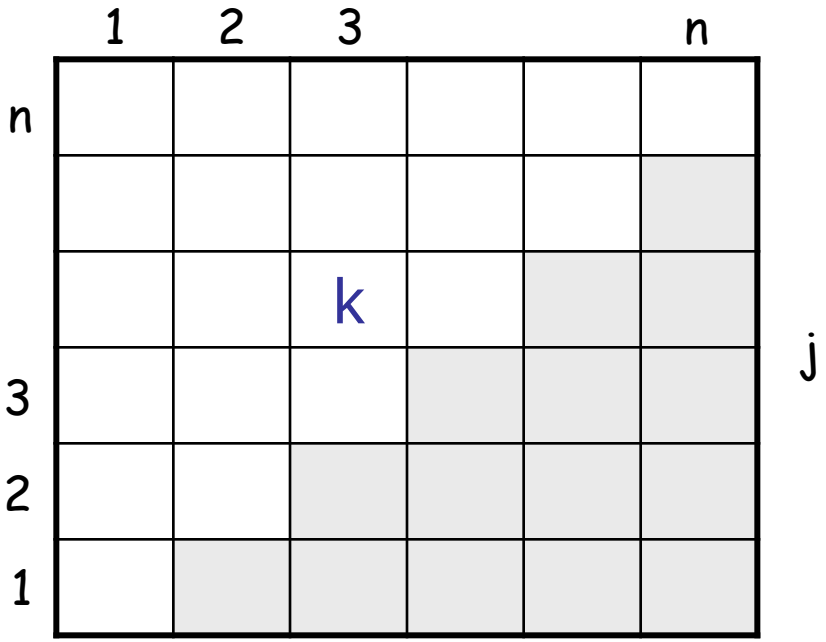
Matrix-Chain-Order(p)

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

$O(N^3)$

4. Construct the Optimal Solution

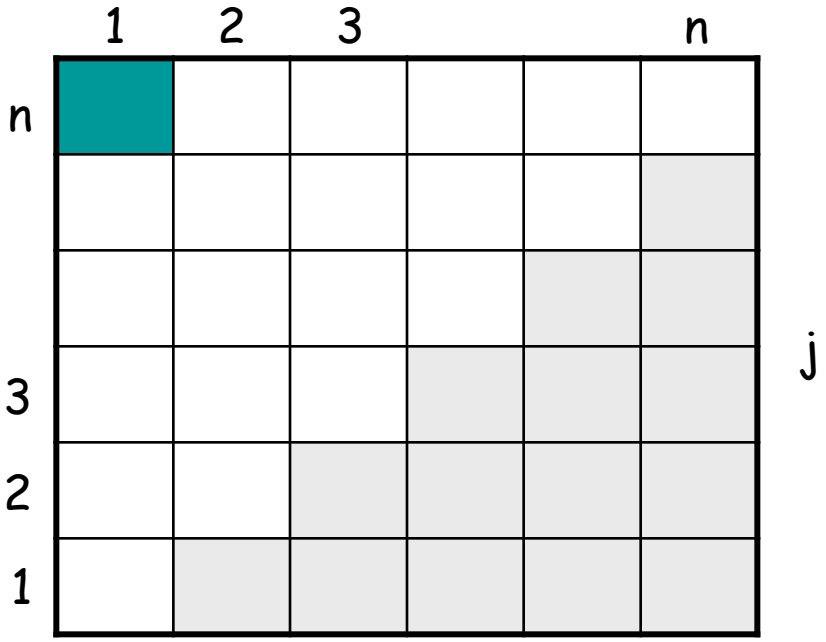
- In a similar matrix s we keep the optimal values of k
- $s[i, j]$ = a value of k such that an optimal parenthesization of $A_{i..j}$ splits the product between A_k and A_{k+1}



4. Construct the Optimal Solution

- $s[1, n]$ is associated with the entire product $A_{1..n}$
  - The final matrix multiplication will be split at k = $s[1, n]$

    $A_{1..n} = A_{1..s[1, n]} \cdot A_{s[1, n]+1..n}$
  - For each subproduct recursively find the corresponding value of k that results in an optimal parenthesization



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}

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	-					

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

4. Construct the Optimal Solution (cont.)

```
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 ")"
```

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	-					

Example: $A_1 \cdot \cdot \cdot A_6$ ((A_1 (A_2 A_3)) ((A_4 A_5) A_6))

```
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 ")"
```

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

	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

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) ⇒ "A_1"

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

i = 2, j = 3 "(" P-O-P (s, 2, 2) ⇒ "A_2"

P-O-P (s, 3, 3) ⇒ "A_3"

")"

")" ...

Memoization

- Top-down approach with the efficiency of typical dynamic programming approach
- Maintaining an entry in a table for the solution to each subproblem
  - **memoize** the inefficient recursive algorithm
- When a subproblem is first encountered its solution is computed and stored in that table
- Subsequent “calls” to the subproblem simply look up that value

Memoized Matrix-Chain

Alg.: MEMOIZED-MATRIX-CHAIN(p)

```
1.  n ← length[p] – 1
2.  for i ← 1 to n
3.       do for j ← i to n
4.            do m[i, j] ← ∞
5.  return LOOKUP-CHAIN(p, 1, n)
```

Initialize the m table with large values that indicate whether the values of m[i, j] have been computed

Top-down approach

Memoized Matrix-Chain

Alg.: LOOKUP-CHAIN(p, i, j)

Running time is $O(n^3)$

```
1.  if m[i, j] < ∞
2.      then return m[i, j]
3.  if i = j
4.     then m[i, j] ← 0
5.     else for k ← i to j – 1
6.              do q ← LOOKUP-CHAIN(p, i, k) +
                        LOOKUP-CHAIN(p, k+1, j) + p_{i-1} p_k p_j
7.                 if q < m[i, j]
8.                    then m[i, j] ← q
9.  return m[i, j]
```

- A: 10×20
- B: 20×30
- C: 30×40
- D: 40×50

The dimension array p is:

$$p = [10, 20, 30, 40, 50]$$

The task is to find the minimum number of scalar multiplications to multiply all matrices together by filling in the cost table m and the split point table s.

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

Step-by-Step Application:

1. **Initialize**:
   - $n = p.length - 1 = 4$
   - $m[1..4][1..4]$ and $s[1..3][2..4]$ are initialized.
2. **Base Case**:
   - For each matrix, the cost of multiplying a single matrix is 0:

$$m[i, i] = 0 \text{ for } i = 1, 2, 3, 4$$

Thus:

$$m[1, 1] = 0, m[2, 2] = 0, m[3, 3] = 0, m[4, 4] = 0$$

3. **Chain Length $l = 2$**:

- This involves multiplying pairs of matrices (A-B), (B-C), and (C-D).
- **For $i = 1, j = 2$** (Multiply A and B):

$$m[1,2] = \min_{k=1}^{1} (m[1,1] + m[2,2] + p[0] \times p[1] \times p[2])$$

$$m[1,2] = 0 + 0 + 10 \times 20 \times 30 = 6000$$

The split occurs at $k = 1$, so $s[1,2] = 1$.

- **For $i = 2, j = 3$** (Multiply B and C):

$$m[2,3] = \min_{k=2}^{2} (m[2,2] + m[3,3] + p[1] \times p[2] \times p[3])$$

$$m[2,3] = 0 + 0 + 20 \times 30 \times 40 = 24000$$

The split occurs at $k = 2$, so $s[2,3] = 2$.

- **For $i = 3, j = 4$** (Multiply C and D):

$$m[3,4] = \min_{k=3}^{3} (m[3,3] + m[4,4] + p[2] \times p[3] \times p[4])$$

$$m[3,4] = 0 + 0 + 30 \times 40 \times 50 = 60000$$

The split occurs at $k = 3$, so $s[3,4] = 3$.

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

4. **Chain Length $l = 3$** (Multiplying triplets of matrices):

- **For $i = 1, j = 3$** (Multiply A, B, and C):

$$m[1,3] = \min_{k=1}^{2} (m[1,1] + m[2,3] + p[0] \times p[1] \times p[3],\ m[1,2] + m[3,3] + p[0] \times p[2] \times p[3])$$

Calculate for both values of k:

$$m[1,3] = \min (0 + 24000 + 10 \times 20 \times 40,\ 6000 + 0 + 10 \times 30 \times 40)$$

$$m[1,3] = \min (24000 + 8000 = 32000,\ 6000 + 12000 = 18000)$$

So, $m[1,3] = 18000$ and $s[1,3] = 2$.

- **For $i = 2, j = 4$** (Multiply B, C, and D):

$$m[2,4] = \min_{k=2}^{3} (m[2,2] + m[3,4] + p[1] \times p[2] \times p[4],\ m[2,3] + m[4,4] + p[1] \times p[3] \times p[4])$$

Calculate for both values of k:

$$m[2,4] = \min (0 + 60000 + 20 \times 30 \times 50,\ 24000 + 0 + 20 \times 40 \times 50)$$

$$m[2,4] = \min (60000 + 30000 = 90000,\ 24000 + 40000 = 64000)$$

So, $m[2,4] = 64000$ and $s[2,4] = 3$.

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

5. **Chain Length $l = 4$** (Multiplying all four matrices):

- **For $i = 1, j = 4$** (Multiply A, B, C, and D):

$$m[1,4] = \min_{k=1}^{3} (m[1,1] + m[2,4] + p[0] \times p[1] \times p[4],\ m[1,2] + m[3,4] + p[0] \times p[2] \times p[4],\ m[1,3] + m[4,4] + p[0] \times p[3] \times p[4])$$

Calculate for all values of k:

$$m[1,4] = \min (0 + 64000 + 10 \times 20 \times 50,\ 6000 + 60000 + 10 \times 30 \times 50,\ 18000$$

$$m[1,4] = \min (64000 + 10000 = 74000,\ 6000 + 60000 + 15000 = 81000,\ 18000$$

So, $m[1,4] = 38000$ and $s[1,4] = 3$.

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

Final Tables:

- Cost Table m:

$$\begin{matrix} 0 & 6000 & 18000 & 38000 \\ - & 0 & 24000 & 64000 \\ - & - & 0 & 60000 \\ - & - & - & 0 \end{matrix}$$

- Split Table s:

$$\begin{matrix} - & 1 & 2 & 3 \\ - & - & 2 & 3 \\ - & - & - & 3 \end{matrix}$$

Split refers to paranthesis ()

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

Final Answer:

The minimum number of scalar multiplications required to multiply matrices A, B, C, and D is **38000**, with the optimal split occurring at $k = 3$. **(ABC), D**

Final Tables:

- **Cost Table** m:

$$\begin{matrix} 0 & 6000 & 18000 & 38000 \\ - & 0 & 24000 & 64000 \\ - & - & 0 & 60000 \\ - & - & - & 0 \end{matrix}$$

- **Split Table** s:

$$\begin{matrix} - & 1 & 2 & 3 \\ - & - & 2 & 3 \\ - & - & - & 3 \end{matrix}$$

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                  // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

In matrix chain multiplication, the **cost table** m (the table that stores the minimum scalar multiplication cost for subproblems) is filled only in the **upper triangle** because we are only interested in computing the costs for multiplying matrices from index i to j, where $i \leq j$.

Here's why:

- The **lower triangle** (where $i > j$) corresponds to invalid subproblems in the context of matrix chain multiplication because it doesn't make sense to multiply a sequence of matrices in reverse order (from j to i). We can only multiply matrices A_i to A_j for $i \leq j$. So, there's no need to compute or store any values for $m[i, j]$ where $i > j$.