Matrix Multiplications

Algorithm to Multiply 2 Matrices

Input: Matrices $A_{p \times q}$ and $B_{q \times r}$ (with dimensions $p \times q$ and $q \times r$)
Result: Matrix $C_{p \times r}$ resulting from the product $A \cdot B$

MATRIX-MULTIPLY($A_{p \times q}$, $B_{q \times r}$)
1. **for** $i \leftarrow 1$ **to** p
2.   **for** $j \leftarrow 1$ **to** r
3. $C[i, j] \leftarrow 0$
4.   **for** $k \leftarrow 1$ **to** q
5. $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
6. **return** C

Scalar multiplication in line 5 dominates time to compute
CNumber of scalar multiplications = *pqr*

Matrix-chain Multiplication Problem

- Suppose we have a sequence or chain A_1, A_2, …, A_n of n matrices to be multiplied
  - That is, we want to compute the product $A_1A_2 \ldots A_n$

- There are many possible ways (parenthesizations) to compute the product

Matrix-chain Multiplication Problem

- Given a chain $< A_1, A_2, \ldots, A_n >$ of n matrices, where for $i=0,1,\ldots,n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product of matrices in a way that minimizes the number of scalar multiplications.

$\langle A_1, A_2, \ldots, A_n \rangle$, where A_i is $p_{i-1} \times p_i$, compute $A_1A_2 \ldots A_n$

Matrix-chain Multiplication ...contd

- To compute the number of scalar multiplications necessary, we must know:
  - Algorithm to multiply two matrices
  - Matrix dimensions

Matrix-chain Multiplication ...contd

- Example: consider the chain A_1, A_2, A_3, A_4 of 4 matrices
  - Let us compute the product $A_1A_2A_3A_4$
- There are 5 possible ways:

1. $(A_1(A_2(A_3A_4)))$
2. $(A_1((A_2A_3)A_4))$
3. $((A_1A_2)(A_3A_4))$
4. $((A_1(A_2A_3))A_4)$
5. $(((A_1A_2)A_3)A_4)$

Example

- Example: Consider three matrices $A_{10\times100}$, $B_{100\times5}$, and $C_{5\times50}$
- There are 2 ways to parenthesize
  - $((AB)C) = D_{10\times5} \cdot C_{5\times50}$
    - $AB \Rightarrow 10\cdot100\cdot5=5{,}000$ scalar multiplications
    - $DC \Rightarrow 10\cdot5\cdot50 =2{,}500$ scalar multiplications

    Total: 7,500
  - $(A(BC)) = A_{10\times100} \cdot E_{100\times50}$
    - $BC \Rightarrow 100\cdot5\cdot50=25{,}000$ scalar multiplications
    - $AE \Rightarrow 10\cdot100\cdot50 =50{,}000$ scalar multiplications

    Total: 75,000

Example

- For example, if the chain of matrices is $\langle A_1, A_2, \ldots, A_n \rangle$, the product of $A_1A_2\ldots A_n$ can be fully parenthesized in five distinct ways:

$$
\begin{aligned}
&(A_1(A_2(A_3A_4))),\\
&(A_1((A_2A_3)A_4)),\\
&((A_1A_2)(A_3A_4)),\\
&((A_1(A_2A_3))A_4),\\
&(((A_1A_2)A_3)A_4).
\end{aligned}
$$

Example

$A_1 \times A_2 \times A_3 \times A_4$

p_i : 13 5 89 3 34

$(A_1(A_2(A_3A_4)))$, $(A_1((A_2A_3)A_4))$, $((A_1A_2)(A_3A_4))$,

$((A_1(A_2A_3))A_4)$, $(((A_1A_2)A_3)A_4)$.

Example – contd.

$(A_1(A_2(A_3A_4))) \rightarrow A_1 \times (A_2A_3A_4) \rightarrow A_2 \times (A_3A_4) \rightarrow A_3 \times A_4$

$$\begin{aligned} \text{cost} &= 13*5*34 + 5*89*34 + 89*3*34 \\ &= 2210 + 15130 + 9078 \\ &= 26418 \end{aligned}$$

$A_1 \times A_2 \times A_3 \times A_4$

13 5 89 3 34

$(A_1(A_2(A_3A_4)))$, costs = 26418

$(A_1((A_2A_3)A_4))$, costs = 4055

$((A_1A_2)(A_3A_4))$, costs = 54201

$((A_1(A_2A_3))A_4)$, costs = 2856

$(((A_1A_2)A_3)A_4)$, costs = 10582

Catalan Number

For any n, # ways to fully parenthesize the product of a chain of n+1 matrices

= # binary trees with n nodes.

= # permutations generated from 1 2 … n through a stack.

= # n pairs of fully matched parentheses.

= n-th Catalan Number = C(2n, n)/(n +1) = $\Omega(4^n/n^{3/2})$

Example – contd.

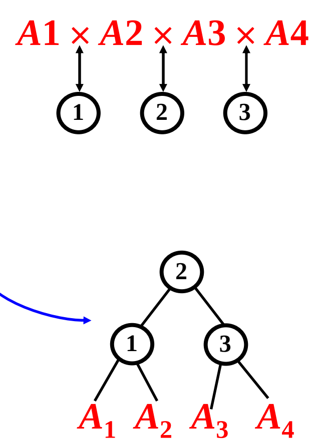
$(A_1(A_2(A_3A_4)))$

$(A_1((A_2A_3)A_4))$

$\boxed{((A_1A_2)(A_3A_4))}$

$((A_1(A_2A_3))A_4)$

$(((A_1A_2)A_3)A_4)$



Matrix-chain Multiplication - Problem

- Matrix-chain multiplication problem
  - Given a chain $A_1, A_2, \ldots, A_n$ of n matrices, where for $i=1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$
  - Parenthesize the product $A_1A_2 \ldots A_n$ such that the total number of scalar multiplications is minimized
- Brute force method of exhaustive search takes time exponential in n

Dynamic Programming Approach

- The structure of an optimal solution
  - Let us use the notation $A_{i..j}$ for the matrix that results from the product $A_i\ A_{i+1} \ldots A_j$
  - An optimal parenthesization of the product $A_1A_2 \ldots A_n$ splits the product between A_k and A_{k+1} for some integer k where $1 \leq k < n$
  - First compute matrices $A_{1..k}$ and $A_{k+1..n}$; then multiply them to get the final matrix $A_{1..n}$

Dynamic Programming Approach ...contd

- **Key observation**: parenthesizations of the subchains $A_1A_2\ldots A_k$ and $A_{k+1}A_{k+2}\ldots A_n$ must also be optimal if the parenthesization of the chain $A_1A_2\ldots A_n$ is optimal (why?)

- That is, the optimal solution to the problem contains within it the optimal solution to subproblems

Dynamic Programming Approach ...contd

- Recursive definition of the value of an optimal solution
  - Let $m[i, j]$ be the minimum number of scalar multiplications necessary to compute $A_{i..j}$
  - Minimum cost to compute $A_{1..n}$ is $m[1, n]$
  - Suppose the optimal parenthesization of $A_{i..j}$ splits the product between A_k and A_{k+1} for some integer k where $i \le k < j$

Dynamic Programming Approach ...contd

- But… optimal parenthesization occurs at one value of k among all possible $i \le k < j$
- Check all these and select the best one

Dynamic Programming Approach ...contd

- $A_{i..j} = (A_i A_{i+1} \ldots A_k) \cdot (A_{k+1} A_{k+2} \ldots A_j) = A_{i..k} \cdot A_{k+1..j}$
- Cost of computing $A_{i..j}$ = cost of computing $A_{i..k}$ + cost of computing $A_{k+1..j}$ + cost of multiplying $A_{i..k}$ and $A_{k+1..j}$
- Cost of multiplying $A_{i..k}$ and $A_{k+1..j}$ is $p_{i-1} p_k p_j$
- $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$ *for* $i \le k < j$
- $m[i, i] = 0$ for $i = 1, 2, \ldots, n$

Dynamic Programming Approach ...contd

- To keep track of how to construct an optimal solution, we use a table s
- $s[i, j]$ = value of k at which $A_i A_{i+1} \dots A_j$ is split for optimal parenthesization
- Algorithm: next slide
  - First computes costs for chains of length l=1
  - Then for chains of length l=2,3, … and so on
  - Computes the optimal cost bottom-up

Step 1

- Step 1: The structure of an optimal parenthesization

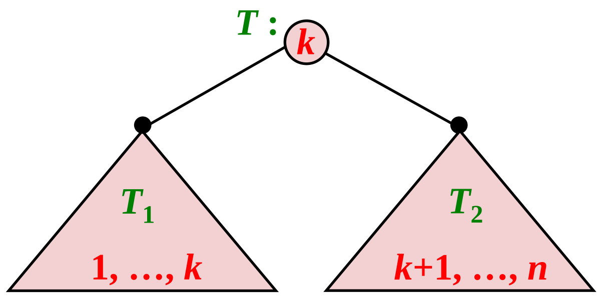
Optimal Optimal

$$((A_1 A_2 \dots A_k)(A_{k+1} A_{k+2} \dots A_n))$$

Combine

Matrix Multiplication – contd.

- **If T is an optimal solution for $A_1, A_2, \ldots, A_n$**



- **then, T_1 (resp. T_2) is an optimal solution for $A_1, A_2, \ldots, A_k$ (resp. $A_{k+1}, A_{k+2}, \ldots, A_n$).**

Matrix Multiplication – contd.

- **Let $m[i, j]$ be the minmum number of scalar multiplications needed to compute the product $A_i \ldots A_j$, for $1 \le i \le j \le n$.**
- **If the optimal solution splits the product $A_i \ldots A_j = (A_i \ldots A_k) \times (A_{k+1} \ldots A_j)$, for some k, $i \le k < j$, then $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}\, p_k\, p_j$. Hence, we have :**

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

Step 2

- Step 2: A recursive solution
  - Compute: $A_i A_{i+1} \ldots A_j$
  - $m[i, j]$= minimum number of scalar multiplications needed to compute the matrix
  - Goal: $m[1, n]$

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

Algorithm to Compute Optimal Cost

Input: Array $p[0 \ldots n]$ containing matrix dimensions and n

Result: Minimum-cost table m and split table s

```
MATRIX-CHAIN-ORDER(p[ ], n)
    for i ← 1 to n
        m[i, i] ← 0
    for l ← 2 to n
        for i ← 1 to n-l+1
              j ← i+l-1
              m[i, j] ← ∞
              for k ← i to j-1
                    q ← m[i, k] + m[k+1, j] + p[i-1] p[k] p[j]
                    if q < m[i, j]
                          m[i, j] ← q
                          s[i, j] ← k
    return m and s
```

Takes $O(n^3)$ time

Requires $O(n^2)$ space

Step 3

```
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
```

Running time: $O(n^3)$

Example

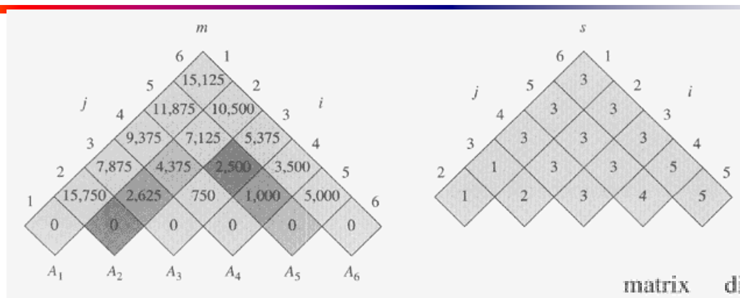
- Show how to multiply this matrix chain optimally
- Solution on the board
  - Minimum cost 15,125
  - Optimal parenthesization $((A_1(A_2A_3))((A_4 A_5)A_6))$

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

MCM DP—order of matrix computations

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

Example



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

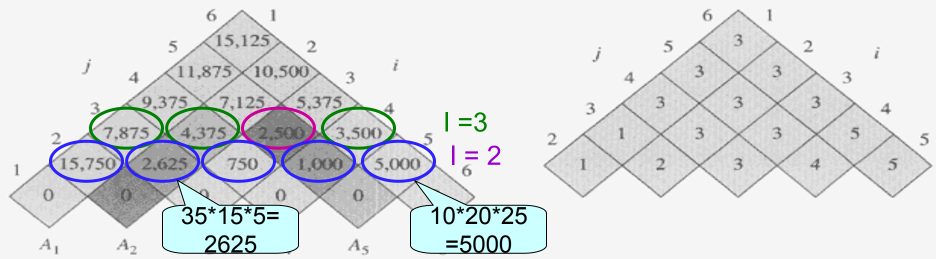


Figure 15.3 The m and s tables computed by MATRIX-CHAIN-ORDER for $n = 6$ and the following matrix dimensions:

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

$$m[3,5] = \min \begin{cases} m[3,4]+m[5,5] + 15*10*20 \\ =750 + 0 + 3000 = 3750 \\ m[3,3]+m[4,5] + 15*5*20 \\ =0 + 1000 + 1500 = 2500 \end{cases}$$

The tables are rotated so that the main diagonal runs horizontally. Only the main diagonal and upper triangle are used in the m table, and only the upper triangle is used in the s table. The minimum number of scalar multiplications to multiply the 6 matrices is $m[1, 6] = 15{,}125$. Of the darker entries, the pairs that have the same shading are taken together in line 9 when computing

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 & = 13000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 & = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 & = 11375 \end{cases}$$
$$= 7125\,.$$

Step 4

- Step 4: Constructing an optimal soultion

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

Example: Print-Optimal-Parens(s, 1, 6): $((A_1(A_2A_3))((A_4A_5)A_6))$

- Constructing an optimal solution
  - Each entry s[i, j]=k records that the optimal parenthesization of $A_iA_{i+1}...A_j$ splits the product between A_k and A_{k+1}
  - $A_{i..j} \rightarrow (A_{i..s[i..j]})(A_{s[i..j]+1..j})$

Matrix Multiplication – contd.

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

- **To fill the entry $m[i, j]$, it needs $\Theta(j-i)$ operations. Hence the execution time of the algorithm is**

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(j-i)=\sum_{j=1}^{n}\sum_{i=1}^{j}(j-i)=\sum_{j=1}^{n}[j^2-\frac{j(j+1)}{2}]$$
$$=\sum_{j=1}^{n}\Theta(j^2)=\Theta(n^3)$$

Time: $\Theta(n^3)$
Space: $\Theta(n^2)$

Matrix Multiplication – contd.

- Consider an example with sequence of dimensions <5,2,3,4,6,7,8>

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

	1	2	3	4	5	6
1	0	30	64	132	226	348
2		0	24	72	156	268
3			0	72	198	366
4				0	168	392
5					0	336
6						0

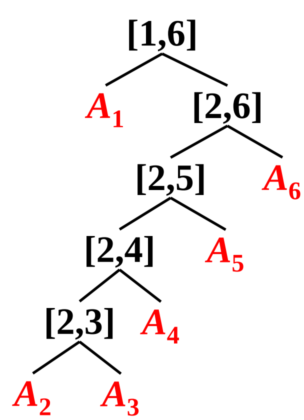
Matrix Multiplication – contd.

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

$s[i, j]$ = a value of k that gives the minimum

s	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5

$A_1((((A_2A_3)A_4)A_5)A_6)$



The End