

Chain Matrix Multiplication

Advanced Programming and Algorithmic Design

Alberto Casagrande

Email: `acasagrande@units.it`

a.a. 2018/2019

The background of the slide features a large, faint watermark of the University of Trieste logo. The logo is circular and contains the text "UNIVERSITA' DEGLI STUDI DI TRIESTE" around the perimeter and "E SPLENDI" in the center. In the middle of the logo is a detailed illustration of a building with a dome and a tower, likely a representation of the University's main building or a historical structure.

Problem Definition

Intuition for the Matrix-chain Multiplication Problem

Consider the matrices A_1, A_2, A_3

- A_1 having dimension 50×5
- A_2 having dimension 5×100
- A_3 having dimension 100×10

How many scalar multiplications does $A_1 \times A_2 \times A_3$ require?

Intuition for the Matrix-chain Multiplication Problem

Matrix product is associative i.e., $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$



Intuition for the Matrix-chain Multiplication Problem

Matrix product is associative i.e., $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$

- if we compute $(A_1 \times A_2) \times A_3$

$$50 * 100 * 5 = 25000 \quad (\text{to compute } A_1 \times A_2)$$

$$50 * 10 * 100 = 50000 \quad (\text{to compute } (A_1 \times A_2) \times A_3)$$

- if we compute $A_1 \times (A_2 \times A_3)$

$$5 * 10 * 100 = 5000 \quad (\text{to compute } A_2 \times A_3)$$

$$50 * 10 * 5 = 2500 \quad (\text{to compute } A_1 \times (A_2 \times A_3))$$

75000 $((A_1 \times A_2) \times A_3)$ vs

Intuition for the Matrix-chain Multiplication Problem

Matrix product is associative i.e., $(A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$

- if we compute $(A_1 \times A_2) \times A_3$

$$50 * 100 * 5 = 25000 \quad (\text{to compute } A_1 \times A_2)$$

$$50 * 10 * 100 = 50000 \quad (\text{to compute } (A_1 \times A_2) \times A_3)$$

- if we compute $A_1 \times (A_2 \times A_3)$

$$5 * 10 * 100 = 5000 \quad (\text{to compute } A_2 \times A_3)$$

$$50 * 10 * 5 = 2500 \quad (\text{to compute } A_1 \times (A_2 \times A_3))$$

75000 $((A_1 \times A_2) \times A_3)$ **vs** **7500** $(A_1 \times (A_2 \times A_3))$

Problem Definition

Consider the **chain** of matrices $\langle A_1, \dots, A_n \rangle$ where

Problem Definition

Consider the **chain** of matrices $\langle A_1, \dots, A_n \rangle$ where A_i has dimensions $p_{i-1} \times p_i$ for all $i \in [1, n]$

Problem Definition

Consider the **chain** of matrices $\langle A_1, \dots, A_n \rangle$ where A_i has dimensions $p_{i-1} \times p_i$ for all $i \in [1, n]$

Compute a **parenthesization** that minimizes the # of scalar products for the chain multiplication

A Naïve Approach

Recursive Solution

We may try to search among all the possible parenthesizations

- if $n = 1$, the parenthesization is obvious
- if $n > 1$, the chain can be parenthesized as

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

for any $k \in [1, n - 1]$. Recursively produce the parenthesizations for $\langle A_1, \dots, A_k \rangle$ and $\langle A_{k+1}, \dots, A_n \rangle$

Recursive Solution

We may try to search among all the possible parenthesizations

- if $n = 1$, the parenthesization is obvious
- if $n > 1$, the chain can be parenthesized as

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

for any $k \in [1, n - 1]$. Recursively produce the parenthesizations for $\langle A_1, \dots, A_k \rangle$ and $\langle A_{k+1}, \dots, A_n \rangle$

Recursive Solution

We may try to search among all the possible parenthesizations

- if $n = 1$, the parenthesization is obvious
- if $n > 1$, the chain can be parenthesized as

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

for any $k \in [1, n - 1]$. Recursively produce the parenthesizations for $\langle A_1, \dots, A_k \rangle$ and $\langle A_{k+1}, \dots, A_n \rangle$

How many parenthesizations has $\langle A_1, \dots, A_n \rangle$?

many of all the calls are computing the same things: is trying to compute a suboptimal problem

Counting Parenthesizations

$\langle A_1, \dots, A_n \rangle$ has

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) * P(n-k) & \text{if } n > 1 \end{cases}$$

different parenthesizations

Counting Parenthesizations

$\langle A_1, \dots, A_n \rangle$ has

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) * P(n-k) & \text{if } n > 1 \end{cases}$$

different parenthesizations

It can be proved that $P(n) \in \Omega(2^n)$

Counting Parenthesizations

$\langle A_1, \dots, A_n \rangle$ has

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k) * P(n-k) & \text{if } n > 1 \end{cases}$$

different parenthesizations

It can be proved that $P(n) \in \Omega(2^n)$

Too many parenthesizations to be enumerated!!!
(if you don't believe it, try for $n = 8$)

Some Breakthrough Observations

- if $(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$ is optimal for the chain,
 - the 1st part is optimal for $\langle A_1, \dots, A_k \rangle$
 - the 2nd part is optimal for $\langle A_{k+1}, \dots, A_n \rangle$
- many branches of the 'naïve' recursive approach perform the very same computation.
 - e.g. for every parenthesization of $A_1 \times \dots \times A_k$, the parenthesizations for $A_{k+1} \times \dots \times A_n$

Some Breakthrough Observations

- if $(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$ is optimal for the chain,
 - the 1st part is optimal for $\langle A_1, \dots, A_k \rangle$
 - the 2nd part is optimal for $\langle A_{k+1}, \dots, A_n \rangle$
- many branches of the “naïve” recursive approach perform the very same computations
 e.g., for every parenthesization of $A_1 \times \dots \times A_k$, the parenthesizations are recomputed $A_{k+1} \times \dots \times A_n$

Some Breakthrough Observations

- if $(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$ is optimal for the chain,
 - the 1st part is optimal for $\langle A_1, \dots, A_k \rangle$
 - the 2nd part is optimal for $\langle A_{k+1}, \dots, A_n \rangle$
- many branches of the “naïve” recursive approach perform the very same computations

e.g., for every parenthesization of $A_1 \times \dots \times A_k$, the parenthesizations are recomputed $A_{k+1} \times \dots \times A_n$

Some Breakthrough Observations

- if $(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$ is optimal for the chain,
 - the 1st part is optimal for $\langle A_1, \dots, A_k \rangle$
 - the 2nd part is optimal for $\langle A_{k+1}, \dots, A_n \rangle$
- many branches of the “naïve” recursive approach perform the very same computations

e.g., for every parenthesization of $A_1 \times \dots \times A_k$, the parenthesizations are recomputed $A_{k+1} \times \dots \times A_n$

Some Breakthrough Observations

- if $(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$ is optimal for the chain,
 - the 1st part is optimal for $\langle A_1, \dots, A_k \rangle$
 - the 2nd part is optimal for $\langle A_{k+1}, \dots, A_n \rangle$
- many branches of the “naïve” recursive approach perform the very same computations

e.g., for every parenthesization of $A_1 \times \dots \times A_k$, the parenthesizations are recomputed $A_{k+1} \times \dots \times A_n$

Idea:

Solution optimal \leftrightarrow every slice is suboptimal dynamic programming avoid search

Recursively compute optimal parenthesizations and use dynamic programming

The background of the slide features a large, faint watermark of the University of Trieste logo. The logo is circular and contains the text "UNIVERSITA' DEGLI STUDI DI TRIESTE" around the perimeter. In the center, there is an illustration of a building with a dome and a tower, with the words "E SPLENDI" below it.

A Dynamic Programming Solution

Dynamic Programming Solution

!!! A_i has dimension $p_{i-1} \times p_i$

Suppose we have $A_1 \times \dots \times A_i \times \dots \times A_j \times \dots \times A_n$. Define $m[i,j]$ = number of scalar products required

Store the minimum # of products for all the sub-chains in m

Recursively, compute $m[i, j]$ as:

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

split problem into subproblems: $A_i \dots A_j \rightarrow A_i \dots A_k \dots A_j$

number of scalar

Dynamic Programming Solution

Store the minimum # of products for all the sub-chains in m

Recursively, compute $m[i, j]$ as:

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

For each i, j also store in $s[i, j]$ the k that minimizes

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

i.e., the parenthesization for the current level

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

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

1	2	3	4	
0	?	?	?	1
	0	?	?	2
		0	?	3
			0	4

m

2	3	4	
?	?	?	1
	?	?	2
		?	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$(A_1 \times A_2 \times A_3 \times A_4)$$

I want to put the p

1	2	3	4	
0	?	?	?	1
	0	?	?	2
		0	?	3
			0	4

m

2	3	4	
?	?	?	1
	?	?	2
		?	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times (A_2 \times A_3 \times A_4))$$

Isolate A_1 as a try for a k . A_1 isolated \rightarrow no multiplication: $i=j=1 \rightarrow m[i,j]=0$. The whole diagonal

1	2	3	4	
0	?	?	?	1
	0	?	?	2
		0	?	3
			0	4

m

2	3	4	
?	?	?	1
	?	?	2
		?	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2) \times (A_3 \times A_4)))$$

1	2	3	4	
0	?	?	?	1
	0	?	?	2
		0	?	3
			0	4

m

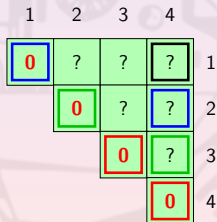
2	3	4	
?	?	?	1
	?	?	2
		?	3

s

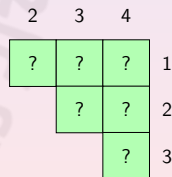
Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2) \times ((A_3) \times (A_4))))$$



m



s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2) \times ((A_3) \times (A_4))))$$

1	2	3	4	
0	?	?	?	1
	0	?	?	2
		0	60	3
			0	4

m

$10 \times 2 \times 3$

2	3	4	
?	?	?	1
	?	?	2
		3	3

s

between A_3, A_4 we can on

will store the k which minimizes $m[i, j]$

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2) \times (A_3 \times A_4)))$$

1	2	3	4
0	?	?	?
	0	?	210
		0	60
			0

m

2	3	4
?	?	?
	?	2
		3

s

$5 \times 10 \times 3 + 60$

Seeing among A_2, A_4

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2 \times A_3) \times (A_4)))$$

1	2	3	4	
0	?	?	?	1
	0	?	210	2
		0	60	3
			0	4

m

2	3	4	
?	?	?	1
	?	2	2
		3	3

s

no we try $s[4,2] = \text{parent}$

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2) \times (A_3)) \times (A_4)))$$

1	2	3	4	
0	?	?	?	1
	0	?	210	2
		0	60	3
			0	4

m

2	3	4	
?	?	?	1
	?	2	2
		3	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2) \times (A_3)) \times (A_4)))$$

1	2	3	4	
0	?	?	?	1
	0	100	210	2
		0	60	3
			0	4

m

2	3	4	
?	?	?	1
	2	2	2
		3	3

s

between A_2 and A_3 we c

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2 \times A_3) \times (A_4)))$$

1	2	3	4	
0	?	?	?	1
	0	100	130	2
		0	60	3
			0	4

m

2	3	4	
?	?	?	1
	2	3	2
		3	3

s

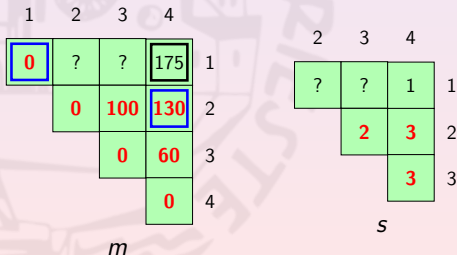
$(A_2 \times A_3) 100 \text{ op} + 5 \times 2 \times 3$

This is best way to parenthesize $(A_2 A_3 A_4)$. Will not be computed again

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times (A_2 \times A_3 \times A_4))$$



isolating A_1 , the best parenthesisation gives 175 op at least

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1 \times A_2) \times (A_3 \times A_4))$$

change case: parenthesis closed after 2 and 4. We already know how

1	2	3	4	
0	?	?	175	1
	0	100	130	2
		0	60	3
			0	4

m

2	3	4	
?	?	1	1
	2	3	2
		3	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$(((A_1) \times (A_2)) \times (A_3 \times A_4))$$

210 > 175 -> Nope

	1	2	3	4	
1	0	150	?	175	1
2		0	100	130	2
3			0	60	3
4				0	4

m

	2	3	4	
1	1	?	1	1
2		2	3	2
3			3	3

s

between A_2 and A_1

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1 \times A_2) \times (A_3 \times A_4))$$

1	2	3	4	
0	150	?	175	1
	0	100	130	2
		0	60	3
			0	4

m

2	3	4	
1	?	1	1
	2	3	2
		3	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1 \times A_2 \times A_3) \times (A_4))$$

1	2	3	4	
0	150	?	175	1
	0	100	130	2
		0	60	3
			0	4

m

2	3	4	
1	?	1	1
	2	3	2
		3	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$(((A_1) \times (A_2 \times A_3)) \times (A_4))$$

1	2	3	4	
0	150	130	175	1
	0	100	130	2
		0	60	3
			0	4

m

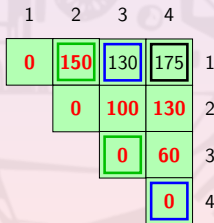
2	3	4	
1	1	1	1
	2	3	2
		3	3

s

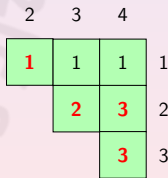
Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$(((A_1 \times A_2) \times (A_3)) \times (A_4))$$



m



s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1 \times A_2 \times A_3) \times (A_4))$$

Total: 130+ 3x2x3

1	2	3	4	
0	150	130	175	1
	0	100	130	2
		0	60	3
			0	4

m

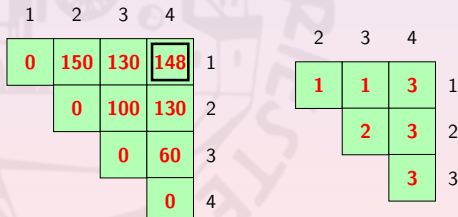
2	3	4	
1	1	1	1
	2	3	2
		3	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$(A_1 \times A_2 \times A_3 \times A_4)$$



How to use s: I want to know $A_1..A_4$, look at 4,1 -> insert parenthesis now I have $(A_1..A_3$

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$(A_1 \times A_2 \times A_3) \times (A_4)$$

1	2	3	4	
0	150	130	148	1
	0	100	130	2
		0	60	3
			0	4

m

2	3	4	
1	1	3	1
	2	3	2
		3	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times (A_2 \times A_3)) \times (A_4)$$

	1	2	3	4	
1	0	150	130	148	1
2		0	100	130	2
3			0	60	3
4				0	4

m

	2	3	4	
1	1	1	3	1
2		2	3	2
3			3	3

s

Dynamic Programming Solution: Example

Consider A_1 (3×5), A_2 (5×10), A_3 (10×2) and A_4 (2×3).

$$((A_1) \times ((A_2) \times (A_3))) \times (A_4)$$

1	2	3	4	
0	150	130	148	1
	0	100	130	2
		0	60	3
			0	4

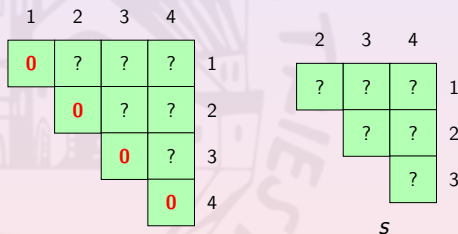
m

2	3	4	
1	1	3	1
	2	3	2
		3	3

s

Dynamic Programming Solution: An Iterative Version

Both m and s can be computed iteratively from the shortest sub-chains to the longest one.



U can work on the upperdiag of m so that we can avoid recursion.

Dynamic Programming Solution: An Iterative Version

Both m and s can be computed iteratively from the shortest sub-chains to the longest one.

	1	2	3	4	
1	0	150	?	?	1
2		0	100	?	2
3			0	60	3
4				0	4

m

	2	3	4	
1	1	?	?	1
2		2	?	2
3			3	3

s

Dynamic Programming Solution: An Iterative Version

Both m and s can be computed iteratively from the shortest sub-chains to the longest one.

	1	2	3	4	
1	0	150	130	?	1
2		0	100	130	2
3			0	60	3
4				0	4

m

	2	3	4	
1	1	1	?	1
2		2	3	2
3			3	3

s

Dynamic Programming Solution: An Iterative Version

Both m and s can be computed iteratively from the shortest sub-chains to the longest one.

	1	2	3	4	
1	0	150	130	148	1
2		0	100	130	2
3			0	60	3
4				0	4

m

	2	3	4	
1	1	1	3	1
2		2	3	2
3			3	3

s

Dynamic Programming Solution: Code

```
def MatrixChain(P):  
    m ← allocate(1..n, 1..n)  
    s ← allocate(1..n-1, 2..n)  
    for i ← 1..n:  
        m[i, i] ← 0  
    for l ← 2..n: l is the diagonal offset  
        for i ← 1..(n-l+1):  
            j ← i + l - 1  
            MatrixChainAux(P, m, s, i, j) subproblem  
        endfor  
    endfor  
  
    return (m, s)  
enddef
```

Dynamic Programming Solution: Code

```
def MatrixChainAux(P,m,s,i,j):  
    m[i,j] ← INFINITY  
    for k ← i..(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  
        endif  
    endfor  
enddef
```

Dynamic Programming Solution: Complexity

The computation of $m[i, j]$ takes time:

$$\sum_{k=i}^{(j-1)} \Theta(1) = \Theta(j - i)$$

Since $i \in [1, n]$ and $j \in [i, n]$,

$$\begin{aligned} T_C(n) &= \sum_{i=1}^n \sum_{j=i}^n \Theta(j - i) = \Theta \left(\sum_{i=1}^n \left(\sum_{j=i}^n j \right) - n * i \right) \\ &= \Theta \left(\sum_{i=1}^n \frac{n * (n + 1)}{2} - \frac{i * (i + 1)}{2} - n * i \right) = \Theta(n^3) \end{aligned}$$

much better than 2^n