Chain Matrix Multiplication

Advanced Programming and Algorithmic Design

Alberto Casagrande
Email: acasagrande@units.it

a.a. 2018/2019

Problem Definition

Intuition for the Matrix-chain Multiplication Problem

Consider the matrices A_1, A_2, A_3

- A_1 having dimention 50×5
- A_2 having dimention 5×100
- A_3 having dimention 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 $(A_1 \times A_2) \times A_3$

$$\begin{aligned} 50 * 100 * 5 &= 25000 & \text{(to compute } A_1 \times A_2) \\ 50 * 10 * 100 &= 50000 & \text{(to compute } (A_1 \times A_2) \times A_3) \end{aligned}$$

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

$$\begin{aligned} 5 * 10 * 100 &= 5000 & \text{(to compute } A_2 \times A_3) \\ 50 * 10 * 5 &= 2500 & \text{(to compute } A_1 \times (A_2 \times A_3)) \end{aligned}$$

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 $(A_1 \times A_2) \times A_3$

$$\begin{aligned} 50 * 100 * 5 &= 25000 && \text{(to compute } A_1 \times A_2) \\ 50 * 10 * 100 &= 50000 && \text{(to compute } (A_1 \times A_2) \times A_3) \end{aligned}$$

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

$$\begin{aligned} 5 * 10 * 100 &= 5000 && \text{(to compute } A_2 \times A_3) \\ 50 * 10 * 5 &= 2500 && \text{(to compute } A_1 \times (A_2 \times A_3)) \end{aligned}$$

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 $\langle A_1, \ldots, A_n \rangle$ where

Problem Definition

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

Problem Definition

Consider the chain of matrices $\langle A_1, \ldots, A_n \rangle$ 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 parenthetized 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 parenthetized as

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

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

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 parenthetized as

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

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

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

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

Counting Parenthesizations

$\langle A_1, \ldots, A_n \rangle$ 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

$\langle A_1, \ldots, A_n \rangle$ 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

$\langle A_1, \ldots, A_n \rangle$ 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 $(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n)$ is optimal for the chain,
  - the 1st part is optimal for $\langle A_1, \ldots A_k \rangle$
  - the 2nd part is optimal for $\langle A_{k+1}, \ldots A_n \rangle$

- many branches of the 'naive' recursive approach perform the very same computations

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

Some Breakthrough Observations

- if $(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n)$ is optimal for the chain,
  - the 1st part is optimal for $\langle A_1, \ldots A_k \rangle$
  - the 2nd part is optimal for $\langle A_{k+1}, \ldots A_n \rangle$

- many branches of the "naïve" recursive approach perform the very same computations

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

Some Breakthrough Observations

- if $(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n)$ is optimal for the chain,
  - the 1st part is optimal for $\langle A_1, \ldots A_k \rangle$
  - the 2nd part is optimal for $\langle A_{k+1}, \ldots A_n \rangle$

- many branches of the "naïve" recursive approach perform the very same computations

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

Some Breakthrough Observations

- if $(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n)$ is optimal for the chain,
  - the 1st part is optimal for $\langle A_1, \ldots A_k \rangle$
  - the 2nd part is optimal for $\langle A_{k+1}, \ldots A_n \rangle$

- many branches of the "naïve" recursive approach perform the very same computations

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

Some Breakthrough Observations

- if $(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n)$ is optimal for the chain,
  - the 1st part is optimal for $\langle A_1, \ldots A_k \rangle$
  - the 2nd part is optimal for $\langle A_{k+1}, \ldots A_n \rangle$

- many branches of the "naïve" recursive approach perform the very same computations

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

Idea: Solution optimal <-> every slice is suboptimaldynamic programming avoid searc

Recursively compute optimal parenthesizations and use dynamic programming

A Dynamic Programming Solution

Dynamic Programming Solution

!!!Ai has dimension p_i-1 * pi

Suppose u have A1 x...x Ai x...x Aj x ....x AnDefine m[i.j]= number of scalar products require

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

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

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

split problem into subproblems: Ai...Aj -> Ai...Ak...Aj

number of scala

Dynamic Programming Solution

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

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

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

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

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

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).

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

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).

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

I want to put the p

1	2	3	4	
0	?	?	$\boxed{?}$	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 A1 as a try for a k. A1 isolated -> no multiplication: i=j=1 -> m[i,j]=0. The whole diagor

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

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).

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

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

10*2*3

m

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

s

between A3,A4 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).

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

m:

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

5*10*3+60

s:

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

Seeing among A2, A4

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]= parenth

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	
	$\boxed{0}$	?	?	$\boxed{?}$	1
		$\boxed{0}$	$\boxed{?}$	$\boxed{210}$	2
			$\boxed{0}$	**60**	3
				$\boxed{0}$	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
1	0	?	?	?
2		0	100	210
3			0	60
4				0

m

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

s

between A2 and A3 we c

Dynamic Programming Solution: Example

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

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

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

m

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

s

(A2xA3)100 op + 5x2x3

This is best way to parenthesize (A2 A3 A4). 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).

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

m:

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

s:

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

isolating A1, the best parentesisation 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 hov

	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	
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

between A2 and A1

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

m

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

m

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

m

	2	3	4
1	1	1	1
2		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).

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

Total: 130+ 3x2x3

	1	2	3	4	
	0	150	$\boxed{130}$	$\boxed{175}$	1
		0	100	130	2
			0	60	3
				$\boxed{0}$	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).

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

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

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

How to use s: I want to know A1..A4, look at 4,1 -> insert parentesisnow I have (A1..A3

Dynamic Programming Solution: Example

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

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

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

m

	2	3	4	
	1	**1**	$\boxed{\mathbf{3}}$	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**
2		**0**	**100**	**130**
3			**0**	**60**
4				**0**

m

	2	3	4
1	**1**	$\boxed{\mathbf{1}}$	$\boxed{\mathbf{3}}$
2		**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
1	0	150	130	148
2		0	100	130
3			0	60
4				0

m

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

m

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

s

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**	?	?
2		**0**	**100**	?
3			**0**	**60**
4				**0**

m

	2	3	4
1	**1**	?	?
2		**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.

1	2	3	4	
0	**150**	**130**	?	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: An Iterative Version

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

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: 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 $j \in [i,n]$,

$$T_C(n) = \sum_{i=n}^{n} \sum_{j=i}^{n} \Theta(j-i) = \Theta\left(\sum_{i=1}^{n}\left(\sum_{j=i}^{n} j\right) - n*i\right)$$

$$= \Theta\left(\sum_{i=1}^{n} \frac{n*(n+1)}{2} - \frac{i*(i+1)}{2} - n*i\right) = \Theta\left(n^3\right)$$

much better than 2^n