# Chain Matrix Multiplication Advanced Programming and Algorithmic Design

Alberto Casagrande Email: acasagrande@units.it

a.a. 2018/2019

•00



Consider the matrices  $A_1, A_2, A_3$ 

- $A_1$  having dimention  $50 \times 5$
- $A_2$  having dimention  $5 \times 100$
- $A_3$  having dimention  $100 \times 10$

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

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



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  $(A_1 \times A_2) \times A_3$ 

$$50 * 100 * 5 = 25000$$
 (to compute  $A_1 \times A_2$ )

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

$$5*10*100 = 5000$$
 (to compute  $A_2 \times A_3$ )

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

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  $(A_1 \times A_2) \times A_3$ 

$$50 * 100 * 5 = 25000$$
 (to compute  $A_1 \times A_2$ )  
 $50 * 10 * 100 = 50000$  (to compute  $(A_1 \times A_2) \times A_3$ )

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

$$5*10*100 = 5000$$
 (to compute  $A_2 \times A_3$ )  
 $50*10*5 = 2500$  (to compute  $A_1 \times (A_2 \times A_3)$ )

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

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

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



#### Recursive Solution

We may try to search among all the possible parenthesizations

- if n = 1, the parenthesization is obvious
  - o 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  $(A_1, A_2)$  and  $(A_1, A_2)$
  - 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  $\langle A_1, \dots, A_k \rangle$  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  $\langle A_1, \dots, A_k \rangle$  and  $\langle A_{k+1}, \dots, A_n \rangle$ 

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

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

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

$$\langle A_1,\ldots,A_n 
angle$$
 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)

- if  $(A_1 \times ... \times A_k) \times (A_{k+1} \times ... \times A_n)$  is optimal for the chain, the 1st part is optimal for  $(A_1, ... A_k)$ 
  - many order es of the recursive approach perform the
  - e.g or e w parenthesization of  $A_1 \times ... \times A_k$ , the

- if  $(A_1 \times ... \times A_k) \times (A_{k+1} \times ... \times A_n)$  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 "naive" recursive approach perform the very same computations
  - e.g., for every parenthesization of  $A_1 \times ... \times A_k$ , the parenthesizations are recomputed  $A_{k+1} \times ... \times A_n$

- if  $(A_1 \times ... \times A_k) \times (A_{k+1} \times ... \times A_n)$  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 "naive" recursive approach perform the very same computations
  - e.g., for every parenthesization of  $A_1 \times ... \times A_k$ , the parenthesizations are recomputed  $A_{k+1} \times ... \times A_n$

- if  $(A_1 \times ... \times A_k) \times (A_{k+1} \times ... \times A_n)$  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  $A_1 \times ... \times A_k$ , the parenthesizations are recomputed  $A_{k+1} \times ... \times A_n$

- if  $(A_1 \times ... \times A_k) \times (A_{k+1} \times ... \times A_n)$  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  $A_1 \times ... \times A_k$ , the parenthesizations are recomputed  $A_{k+1} \times ... \times A_n$

#### Idea:

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

Recursively compute optimal parenthesizations and use dynamic programming



## 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:

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{k \in [i,j-1]} \{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$
split problem into subproblems: Ai...Aj -> Ai...Ak...Aj

number of scale

#### **Dynamic Programming Solution**

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

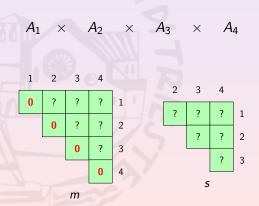
Recursively, compute m[i,j] as:

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

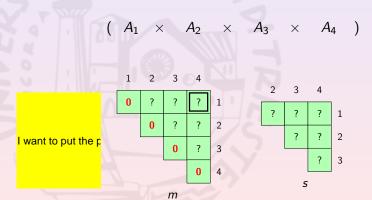
For each i, j also store in s[i, j] the k that minimizes

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

i.e., the parenthesization for the current level



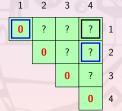
Problem Definition

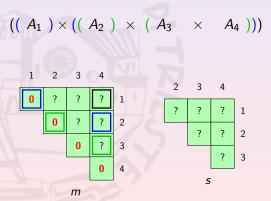


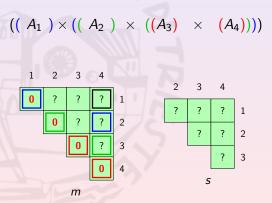
Consider  $A_1$  (3 × 5),  $A_2$  (5 × 10),  $A_3$  (10 × 2) and  $A_4$  (2 × 3).

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

Isolate A1 as a try for a k. A1 isolated -> no multiplication: i=j=1 -> m[i,j]=0. The whole diagor

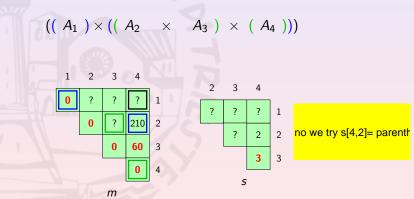




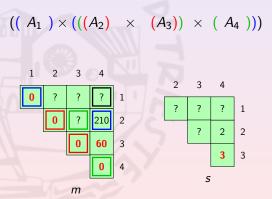


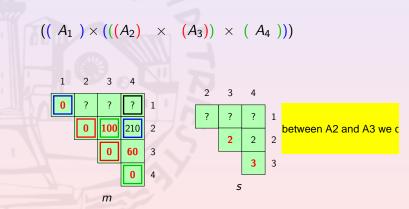
Consider  $A_1$  (3 × 5),  $A_2$  (5 × 10),  $A_3$  (10 × 2) and  $A_4$  (2 × 3).

will store the k which minimizes m[i,i



Problem Definition





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

This is best way to parenthesize (A2 A3 A4). Will not be computed again

Problem Definition

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

isolating A1, the best parentesisation gives 175 op at least

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

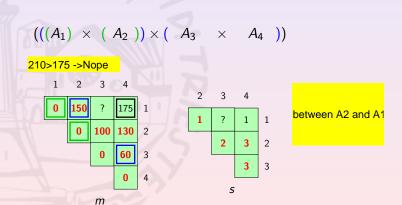
$$((A_1 \times A_2) \times (A_3 \times A_4))$$

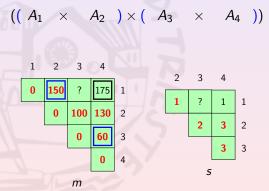
change case: parenthesis closed after 2 and 4. We already know how

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

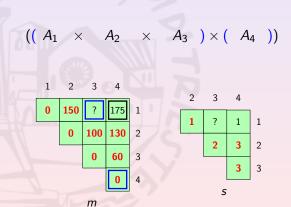
| 2 | 3 | 4 |     |  |  |
|---|---|---|-----|--|--|
| ? | ? | 1 | 1   |  |  |
|   | 2 | 3 | 2   |  |  |
|   |   | 3 | (1) |  |  |
| 5 |   |   |     |  |  |

m

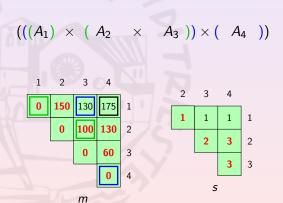




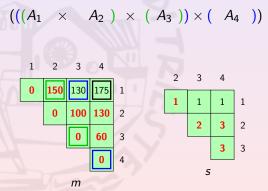
Problem Definition

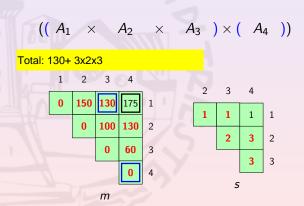


Problem Definition



Problem Definition



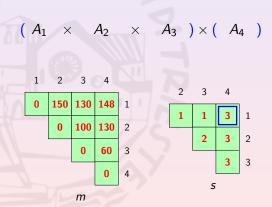


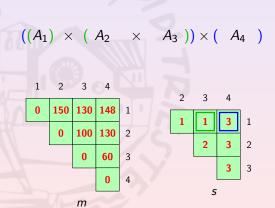
Consider  $A_1$  (3 × 5),  $A_2$  (5 × 10),  $A_3$  (10 × 2) and  $A_4$  (2 × 3).

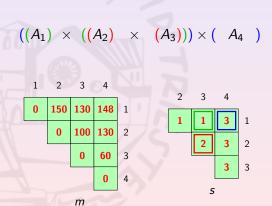
How to use s: I want to know A1..A4, look at 4,1 -> insert parentesisnow I have (A1..A3

Problem Definition

## Dynamic Programming Solution: Example







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

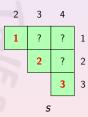
U can work on the upperdiag of m so that we can avoid recursion.

| 1 | 2 | 3 | 4   |   |
|---|---|---|-----|---|
| 0 | ? | ? | ?   | 1 |
|   | 0 | ? | ?   | 2 |
|   | Æ | 0 | ?   | 3 |
|   |   |   | 0   | 4 |
|   |   | m | IL, |   |

| 2 | 3 | 4 |     |  |  |
|---|---|---|-----|--|--|
| ? | ? | ? | 1   |  |  |
|   | ? | ? | 2   |  |  |
|   |   | ? | (1) |  |  |
| 5 |   |   |     |  |  |

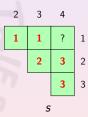
Both m and s can be computed iteratively from the shortest sub-chains to the longest one.

|  | 1 | 2   | 3   | 4  |   |  |
|--|---|-----|-----|----|---|--|
|  | 0 | 150 | ?   | ?  | 1 |  |
|  |   | 0   | 100 | ?  | 2 |  |
|  |   | 崖   | 0   | 60 | 3 |  |
|  |   |     | 0   | 4  |   |  |
|  |   |     | m   |    |   |  |



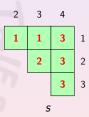
Both m and s can be computed iteratively from the shortest sub-chains to the longest one.

|  | 1 | 2   | 3   | 4   |   |  |
|--|---|-----|-----|-----|---|--|
|  | 0 | 150 | 130 | ?   | 1 |  |
|  |   | 0   | 100 | 130 | 2 |  |
|  |   | JE. | 0   | 60  | 3 |  |
|  |   |     | 0   | 4   |   |  |
|  |   |     | m   | K,  |   |  |



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 |  |
|   | JE. | 0   | 60  | 3 |  |
|   |     |     | 0   | 4 |  |
|   |     | m   | K   |   |  |



enddef

# Dynamic Programming Solution: Code

```
def MatrixChain(P):
   m \leftarrow allocate(1..n, 1..n)
    s \leftarrow allocate(1..n-1, 2..n)
    for i \leftarrow 1...n:
       m[i, i] \leftarrow 0
    for 1 \leftarrow 2 ... n: I is the diagonal
        for i \leftarrow 1..(n-l+1):
             i \leftarrow i + l - 1
             MatrixChainAux(P,m,s,i,j)<sub>subproblem</sub>
        endfor
      endfor
     return (m, s)
```

# Dynamic Programming Solution: Code

```
def MatrixChainAux(P,m,s,i,j):
   m[i,j] \leftarrow INFINITY
    for k \leftarrow i ... (j-1):
       q \leftarrow m[i,k] + m[k+1,j] +
                 P[i-1]*P[k]+P[i]
        if q < m[i,j]:
           m[i,j] \leftarrow q
           s[i,i] \leftarrow 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<sup>n</sup>