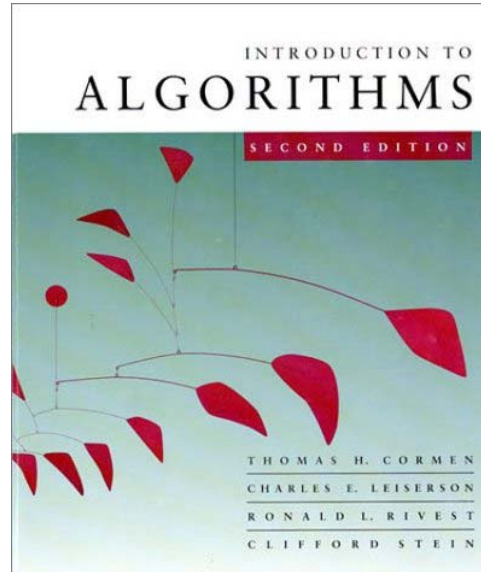
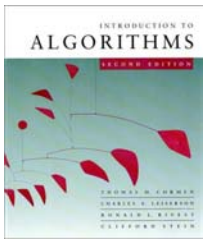


CS 5633 -- Spring 2008



Matrix-chain multiplication

Carola Wenk

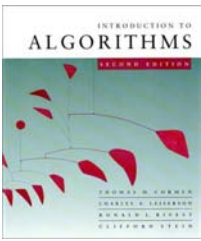


Matrix-chain multiplication

Given: A sequence/chain of n matrices

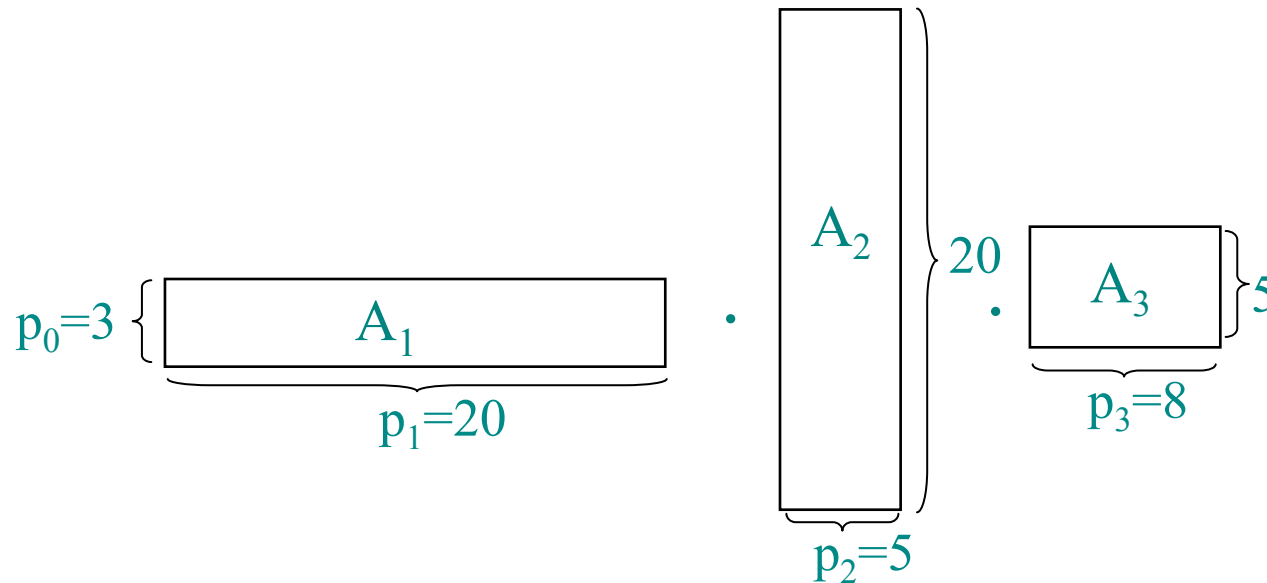
A_1, A_2, \dots, A_n , where A_i is a $p_{i-1} \times p_i$ matrix

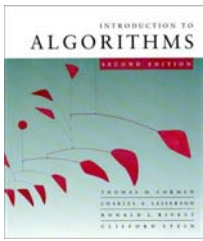
Task: Compute their product $A_1 \cdot A_2 \cdot \dots \cdot A_n$
using the minimum number of scalar
multiplications.



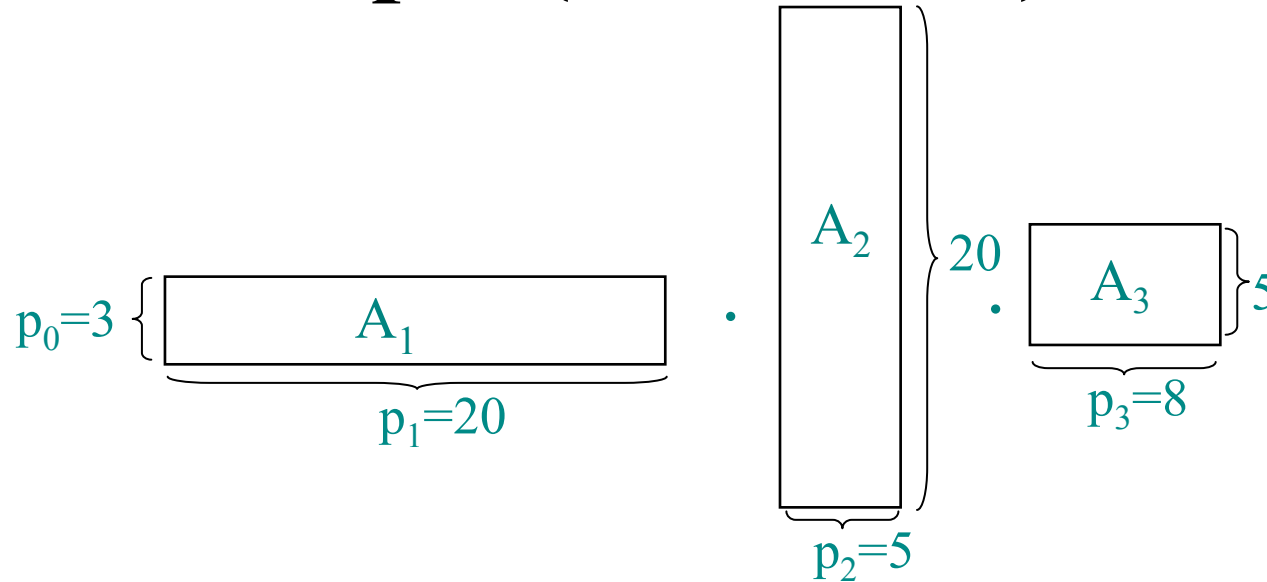
Matrix-chain multiplication example

Example: $n=3$, $p_0=3$, $p_1=20$, $p_2=5$, $p_3=8$. A_1 is a 3×20 matrix, A_2 is a 20×5 matrix, A_3 is a 5×2 matrix. Compute $A_1 \cdot A_2 \cdot A_3$.

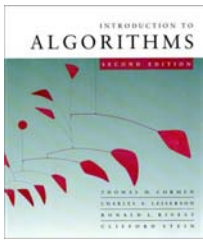




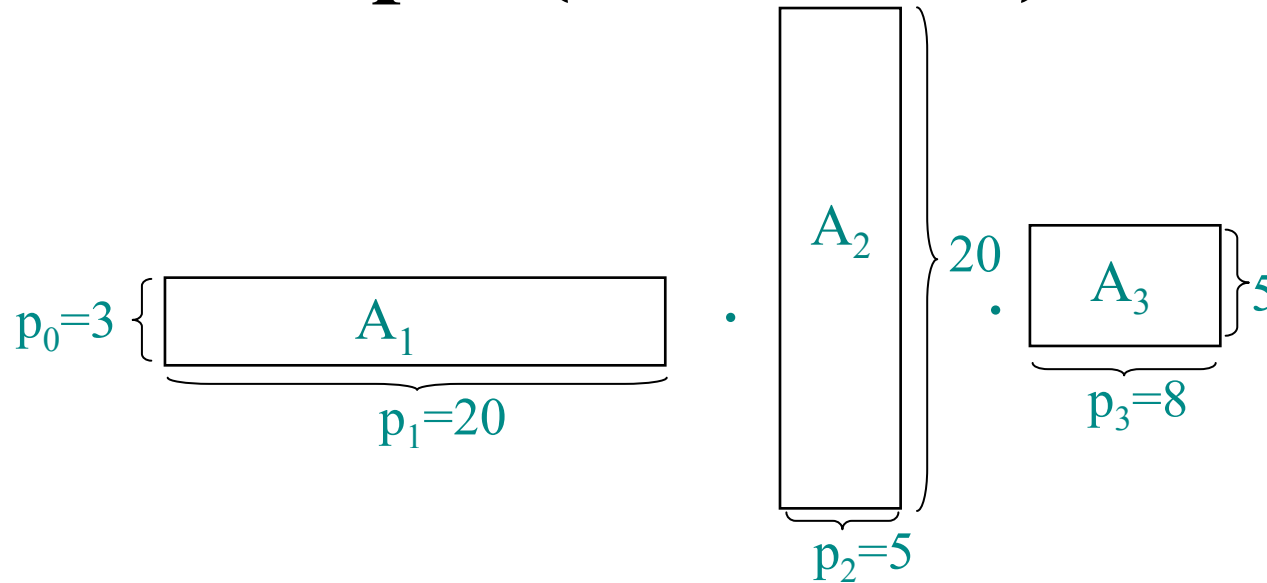
Matrix-chain multiplication example (continued)



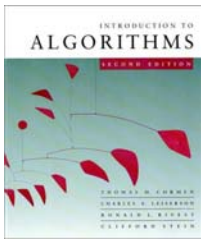
- Computing $A_1 \cdot A_2$ takes $3 \cdot 20 \cdot 5$ multiplications and results in a 3×5 matrix.
- Computing $A_i \cdot A_{i+1}$ takes $p_{i-1} \cdot p_i \cdot p_{i+1}$ multiplications and results in a $p_{i-1} \times p_{i+1}$ matrix.



Matrix-chain multiplication example (continued)



- Computing $(A_1 \cdot A_2) \cdot A_3$ takes $3 \cdot 20 \cdot 5 + 3 \cdot 5 \cdot 8 = 300 + 120 = 420$ multiplications
- Computing $A_1 \cdot (A_2 \cdot A_3)$ takes $20 \cdot 5 \cdot 8 + 3 \cdot 20 \cdot 8 = 800 + 480 = 1280$ multiplications



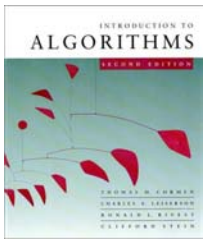
Matrix-chain multiplication

Given: A sequence/chain of n matrices

A_1, A_2, \dots, A_n , where A_i is a $p_{i-1} \times p_i$ matrix

Task: Compute their product $A_1 \cdot A_2 \cdot \dots \cdot A_n$
using the minimum number of scalar
multiplications.

\Rightarrow Find a parenthesization that minimizes
the number of multiplications



Would greedy work?

1. Parenthesizing like this $(\dots((A_1 \cdot A_2) \cdot A_3) \dots \cdot A_n)$ does not work (e.g., reverse our running example).

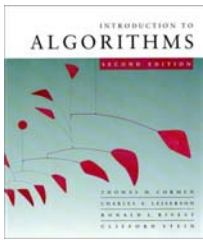
2. Recursively parenthesize like this:

$$\underbrace{(A_1 \cdot \dots \cdot A_k)}_{p_0 \times p_k} \cdot \underbrace{(A_{k+1} \cdot \dots \cdot A_n)}_{p_k \times p_n}$$

Find the k that minimizes $p_0 \cdot p_k \cdot p_n$.

Does not work either (example: $p_0=1, p_1=2, p_2=3, p_3=4$)

⇒ Try dynamic programming



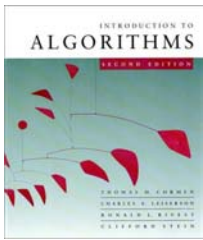
1) Optimal substructure

Let $A_{i,j} = A_i \cdot \dots \cdot A_j$ for $i \leq j$

- Consider an optimal parenthesization for $A_{i,j}$. Assume it splits it at k , so

$$A_{i,j} = (A_i \cdot \dots \cdot A_k) \cdot (A_{k+1} \cdot \dots \cdot A_j)$$

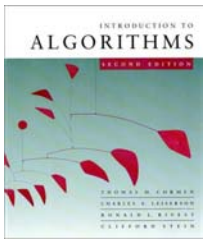
- Then, the par. of the prefix $A_i \cdot \dots \cdot A_k$ within the optimal par. of $A_{i,j}$ must be an optimal par. of $A_{i,k}$. (Assume it is not optimal, then there exists a better par. for $A_{i,k}$. **Copy and paste** this par. into the par. for $A_{i,j}$. This yields a better par. for $A_{i,j}$. Contradiction.)



2) Recursive solution

- a) First compute the minimum number of multiplications
- b) Then compute the actual parenthesization

We will concentrate on solving **a)** now.



2) Recursive solution (cont.)

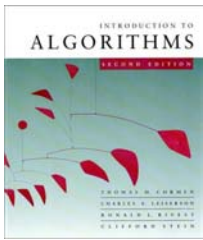
$m[i,j]$ = minimum number of scalar multiplications to compute A_{ij}

Goal: Compute $m[1,n]$

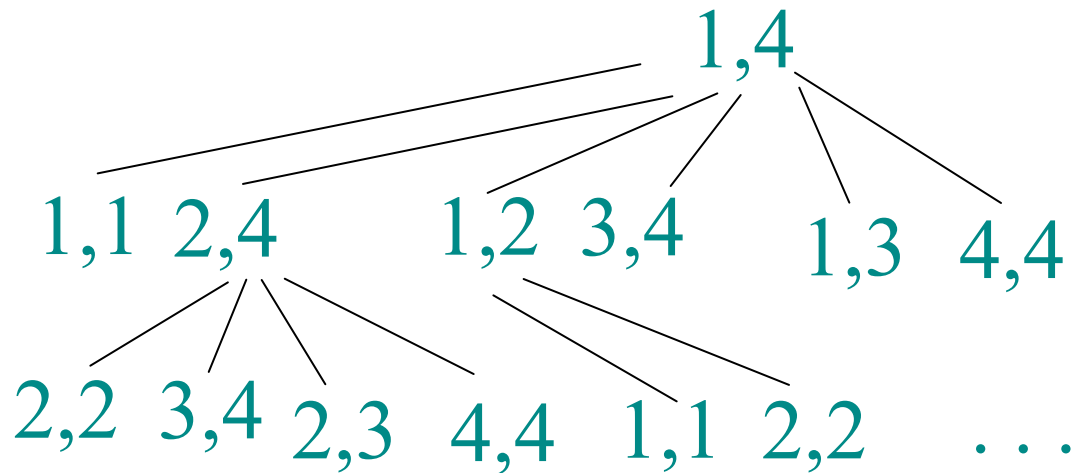
$$A_{i,j} = \underbrace{(A_i \cdots A_k)}_{p_{i-1} \times p_k} \cdot \underbrace{(A_{k+1} \cdots A_j)}_{p_k \times p_j}$$

Recurrence:

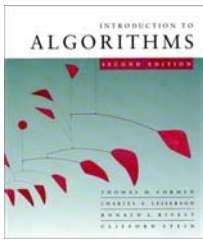
- $m[i,i] = 0$ for $i=1,2,\dots,n$
- $m[i,j] = \min_{i \leq k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j)$



Recursion tree

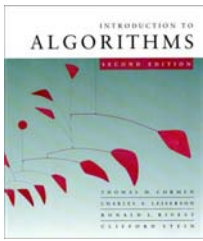


- The runtime of the straight-forward recursive algorithm is $\Omega(2^n)$
- But only $\Theta(n^2)$ different subproblems !



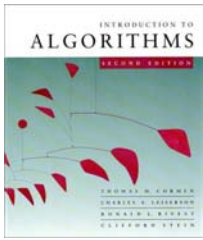
Dynamic programming

```
MATRIX_CHAIN_DP( $p, n$ ):  
  for  $i:=1$  to  $n$  do  $m[i,i]=0$   
  for  $l:=2$  to  $n$  do //  $l$  is length of chain  
    for  $i:=1$  to  $n-l+1$  do  
       $j:=i+l-1$   
       $m[i,j]=\infty$   
      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  
           $m[i,j]=q$   
           $s[i,j]:=k$  //index that optimizes  $m[i,j]$   
  return  $m$  and  $s$ ;
```



Dynamic programming

- Use dynamic programming to fill the 2-dimensional $m[i,j]$ -table
- Bottom-up: Diagonal by diagonal
- For the construction of the optimal parenthesization, use an additional array $s[i,j]$ that records that value of k for which the minimum is attained and stored in $m[i,j]$
- $O(n^3)$ runtime ($n \times n$ table, $O(n)$ min-computation per entry), $O(n^2)$ space
- $m[1,n]$ is the desired value



Construction of an optimal parenthesization

```
PRINT_PARENS( $s, i, j$ ) // initial call: print_parens( $s, 1, n$ )  
  if  $i=j$  then print “A” $i$   
  else print “(  
    PRINT_PARENS( $s, i, s[i, j]$ )  
    PRINT_PARENS( $s, s[i, j]+1, j$ )  
  print “)”
```

Runtime: Recursion tree = binary tree with n leaves. Spend $O(1)$ per node. $O(n)$ total runtime.