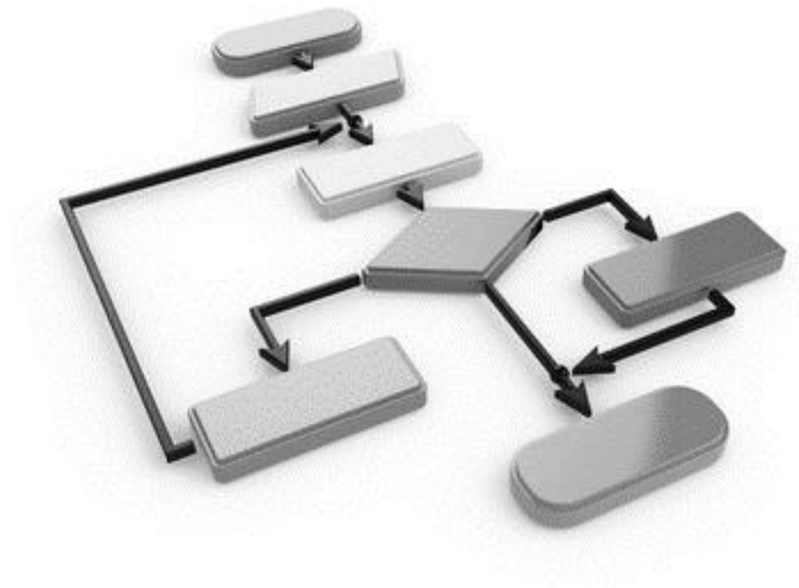


Lecture 11

Dynamic Programming

CSE373: Design and Analysis of Algorithms

Matrix-chain Multiplication

- Input: a sequence (chain) $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices
- Aim: compute the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$
- A product of matrices is fully parenthesized if
  - It is either a single matrix
  - Or, the product of two fully parenthesized matrix products surrounded by a pair of parentheses.

    ▷ $\left(A_i(A_{i+1}A_{i+2} \ldots A_j)\right)$

    ▷ $\left((A_iA_{i+1}A_{i+2} \ldots A_{j-1})A_j\right)$

    ▷ $\left((A_iA_{i+1}A_{i+2} \ldots A_k)(A_{k+1}A_{k+2} \ldots A_j)\right)$ for $i \le k < j$
  - All parenthesizations yield the same product; matrix product is associative

Matrix-chain Multiplication

- Input: $\langle A_1, A_2, A_3, A_4 \rangle$
- 5 distinct ways of full parenthesization

$$(A_1(A_2(A_3A_4)))$$

$$(A_1((A_2A_3)A_4))$$

$$((A_1A_2)(A_3A_4))$$

$$((A_1(A_2A_3))A_4)$$

$$(((A_1A_2)A_3)A_4)$$

- The way we parenthesize a chain of matrices can have a dramatic effect on the cost of computing the product

Matrix-chain Multiplication

Cost of Multiplying two Matrices

Matrix has two attributes

- rows[A]: # of rows
- cols[A]: # of columns

# of scalar mult-adds in $C \leftarrow AB$ is

rows[A]×cols[B]×cols[A]

$$\left.\begin{array}{l} A: (p \times q) \\ B: (q \times r) \end{array}\right\} C = A \cdot B \text{ is } p \times r.$$

# of mult-adds is $p \times r \times q$

MATRIX-MULTIPLY(A, B)

```
if cols[A]≠rows[B] then
    error("incompatible dimensions")
for i ←1 to rows[A] do
    for j←1 to cols[B] do
        C[i,j] ← 0
        for k←1 to cols[A] do
            C[i,j]←
C[i,j]+A[i,k]·B[k,j]
return C
```

Matrix-chain Multiplication

Input: a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, A_i is a $p_{i-1} \times p_i$ matrix

Aim: fully parenthesize the product $A_1 \cdot A_2 \cdot \ldots \cdot A_n$ such that the number of scalar mult-adds are minimized.

- Ex.: $\langle A_1, A_2, A_3 \rangle$ where A_1: 10×100; A_2: 100×5; A_3: 5×50

$$((\underbrace{A_1 A_2}_{10\times5})\ \underbrace{A_3}_{5\times50}): \underbrace{10 \times 100 \times 5}_{A_1 A_2} + \underbrace{10 \times 5 \times 50}_{(A_1 A_2)A_3} = 7500$$

$$(\underbrace{A_1}_{10\times100}\ (\underbrace{A_2 A_3}_{100\times50})): \underbrace{100\times5\times50}_{A_2 A_3} + \underbrace{10\times100\times50}_{A_1 (A_2 A_3)} = 75000$$

$\Rightarrow$ First parenthesization yields 10 times faster computation.

Number of Parenthesizations

- Brute force approach: exhaustively check all parenthesizations
- $P(n)$: # of parenthesizations of a sequence of n matrices
- We can split sequence between kth and $(k+1)$st matrices for any $k=1, 2, \ldots, n-1$, then parenthesize the two resulting sequences independently, i.e.,

$$(A_1 A_2 A_3 \ldots A_k)(A_{k+1} A_{k+2} \ldots A_n)$$

- We obtain the recurrence

$$P(1) = 1 \text{ and } P(n) = \sum_{k=1}^{n-1} P(k) P(n-k)$$

Number of Parenthesizations

- The recurrence generates the sequence of Catalan Numbers
- Solution is $P(n) = C(n-1)$ where

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega(4^n/n^{3/2})$$

- The number of solutions is exponential in n
- Therefore, brute force approach is a poor strategy

Establishing the Recurrence

Consider the subproblem of parenthesizing

$A_{i\ldots j} = A_i \, A_{i+1} \cdots A_j$ for $1 \le i \le j \le n$

$p_{i-1}p_kp_j$

$= A_{i\ldots k} \, A_{k+1\ldots j}$ for $i \le k < j$

m[i, k] m[k+1,j]

Assume that the optimal parenthesization splits the

product $A_i \, A_{i+1} \cdots A_j$ at k ($i \le k < j$)

$$m[i, j] = \underbrace{m[i, k]}_{\substack{\text{min \# of multiplications}\\ \text{to compute } A_{i\ldots k}}} + \underbrace{m[k+1, j]}_{\substack{\text{min \# of multiplications}\\ \text{to compute } A_{k+1\ldots j}}} + \underbrace{p_{i-1}p_kp_j}_{\substack{\text{\# of multiplications}\\ \text{to compute } A_{i\ldots k}A_{k\ldots j}}}$$

Recursive Matrix-chain

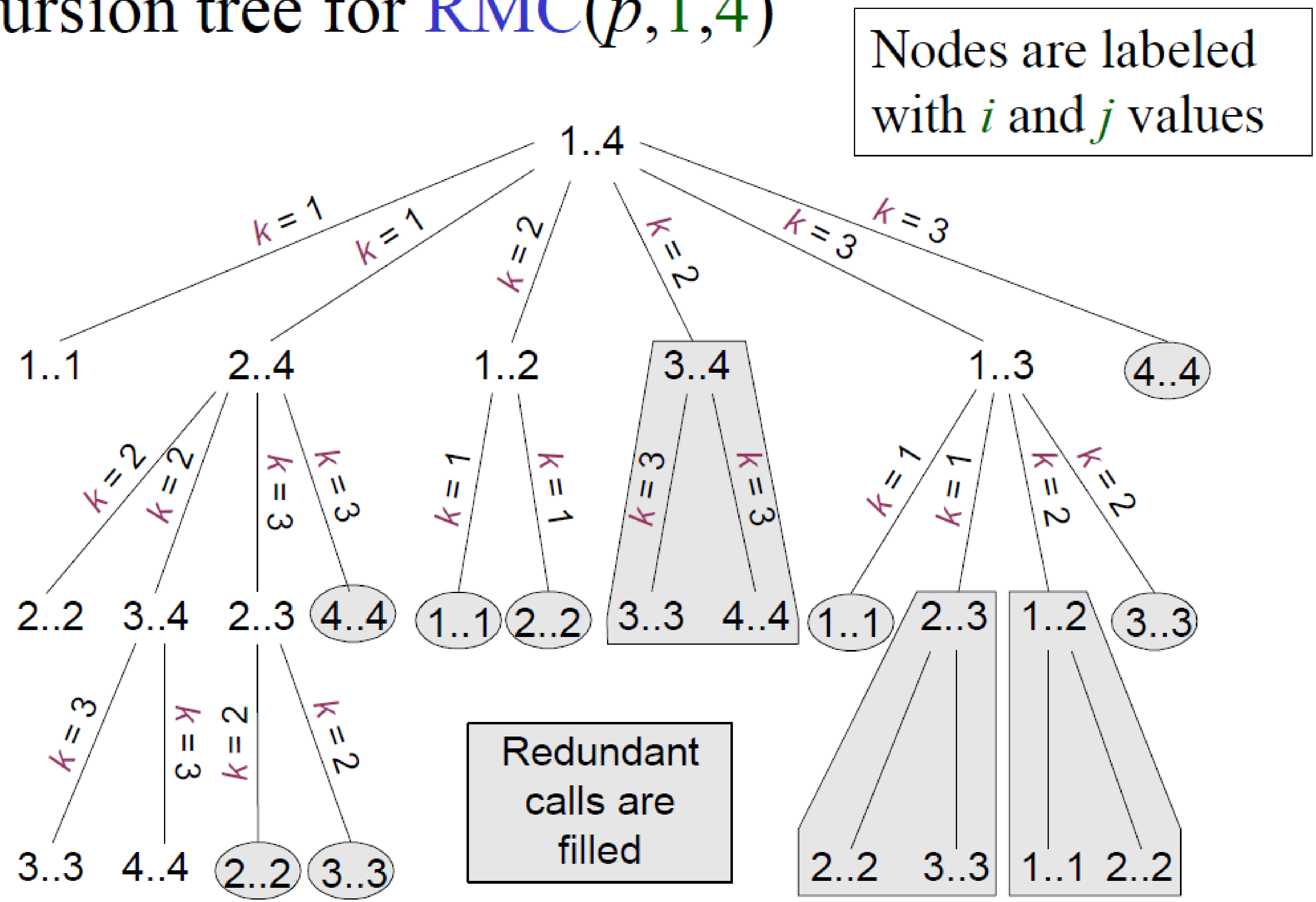
Recursive matrix-chain order

RMC(p, i, j)

if $i = j$ **then**
    return 0
$m[i, j] \leftarrow \infty$
for $k \leftarrow i$ **to** $j - 1$ **do**
    $q \leftarrow \text{RMC}(p, i, k) + \text{RMC}(p, k+1, j) + p_{i-1} p_k p_j$
    if $q < m[i, j]$ **then**
        $m[i, j] \leftarrow q$
return $m[i, j]$

Recursive Matrix-chain

Recursion tree for RMC(p,1,4)



Running Time of Recursive Matrix-chain

$$T(1) \geq 1$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} \left(T(k) + T(n-k) + 1 \right) \text{ for } n > 1$$

- For $i = 1, 2, \ldots, n$ each term $T(i)$ appears twice
  - Once as $T(k)$, and once as $T(n-k)$
- Collect $n-1$ 1's in the summation together with the front 1

$$\boxed{T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n}$$

Running Time of Recursive Matrix-chain

- Try to show that $T(n) \geq 2^{n-1}$ (by substitution)

<u>Base case</u>: $T(1) \geq 1 = 2^0 = 2^{1-1}$ for $n = 1$

<u>IH</u>: $T(i) \geq 2^{i-1}$ for all $i = 1, 2, \ldots, n-1$ and $n \geq 2$

$$T(n) \geq 2\sum_{i=1}^{n-1} 2^{i-1} + n$$

$$= 2\sum_{i=0}^{n-2} 2^{i} + n = 2(2^{n-1} - 1) + n$$

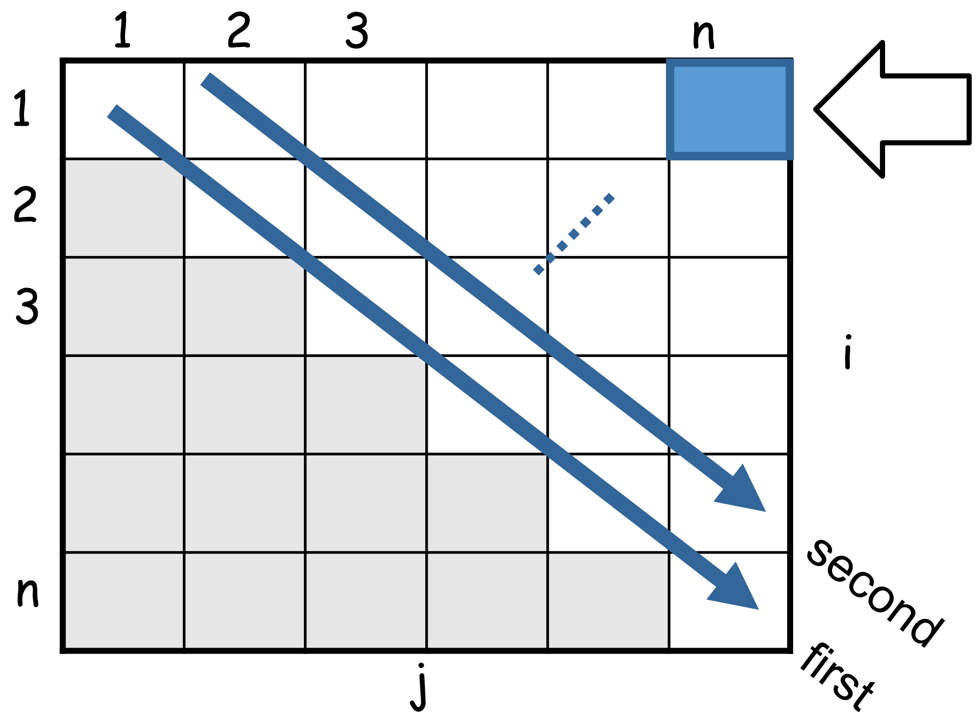
$$= 2^{n-1} + (2^{n-1} - 2 + n)$$

$\Rightarrow T(n) \geq 2^{n-1}$ Q.E.D.

Computing the Optimal Costs

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

- Length = 1: $i = j$, $i = 1, 2, \ldots, n$
- Length = 2: $j = i + 1$, $i = 1, 2, \ldots, n-1$



$m[1, n]$ gives the optimal solution to the problem

Compute rows from bottom to top and from left to right

In a similar matrix s we keep the optimal values of k

Matrix-Chain-Order

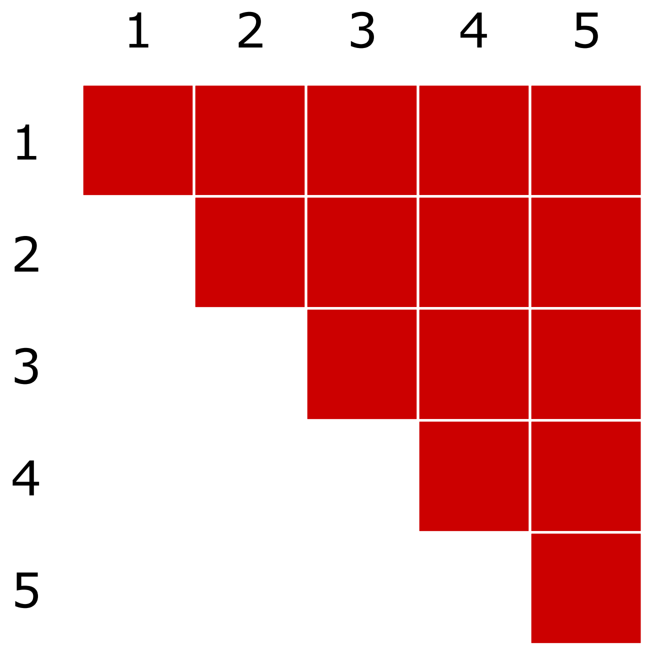
Alg.: MATRIX-CHAIN-ORDER(p)

1. $n = p.length - 1$
2. let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tabels
3. **for** $i = 1$ **to** n
4. $\quad m[i, i] = 0$
5. **for** $l = 2$ **to** n
6. $\quad$ **for** $i = 1$ **to** $n - l + 1$
7. $\quad j = i + l - 1$
8. $\qquad m[i, j] = \infty$
9. $\qquad$ **for** $k = i$ **to** $j - 1$
10. $\qquad\qquad q = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$
11. $\qquad\qquad$ **if** $q < m[i, j]$
12. $\qquad\qquad\quad m[i, j] = q$
13. $\qquad\qquad\quad s[i, j] = k$
14. **return** m and s

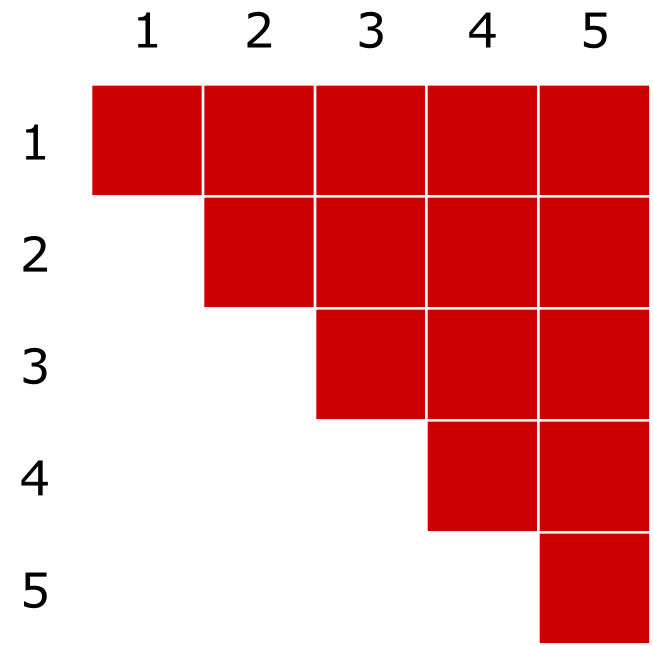
Example:

p = (10, 5, 1, 10, 2, 10)

[10×5]×[5×1]×[1×10]×[10×2]×[2×10]



c(i,j), i ≤ j



s(i,j), i ≤ j

Example:

p = (10, 5, 1, 10, 2, 10)

[10×5]×[5×1]×[1×10]×[10×2]×[2×10]

c(i,i) = 0

	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

c(i,j), i ≤ j

	1	2	3	4	5
1					
2					
3					
4					
5					

s(i,j), i ≤ j

Example:

$p = (10, 5, 1, 10, 2, 10)$

$[10\times5]\times[5\times1]\times[1\times10]\times[10\times2]\times[2\times10]$

$c(i,i+1) = p_i p_{i+1} p_{i+2}$

$s(i,i+1) = i$

	1	2	3	4	5
1	0	50			
2		0	50		
3			0	20	
4				0	200
5					0

$c(i,j),\ i \le j$

	1	2	3	4	5
1		1			
2			2		
3				3	
4					4
5					

$s(i,j),\ i \le j$

Example:

$p = (10, 5, 1, 10, 2, 10)$

$[10{\times}5]\times[5{\times}1]\times[1{\times}10]\times[10{\times}2]\times[2{\times}10]$

$$c(i,i+2) = \min\{c(i,i) + c(i+1,i+2) + p_i p_{i+1} p_{i+3},$$
$$c(i,i+1) + c(i+2,i+2) + p_i p_{i+2} p_{i+3}\}$$

	1	2	3	4	5
1	0	50			
2		0	50		
3			0	20	
4				0	200
5					0

$c(i,j),\ i \le j$

	1	2	3	4	5
1		1			
2			2		
3				3	
4					4
5					

$s(i,j),\ i \le j$

Example:

$p = (10, 5, 1, 10, 2, 10)$

$[10\times5]\times[5\times1]\times[1\times10]\times[10\times2]\times[2\times10]$

$c(2,4) = \min\{c(2,2) + c(3,4) + p_2 p_3 p_5,\ \ c(2,3) + c(4,4) + p_2 p_4 p_5\}$

$c(3,5) = \ldots$

	1	2	3	4	5
1	0	50	150		
2		0	50	30	
3			0	20	40
4				0	200
5					0

$c(i,j),\ i \leq j$

	1	2	3	4	5
1		1	2		
2			2	2	
3				3	3
4					4
5					

$s(i,j),\ i \leq j$

Example:

$[10\times5]\times[5\times1]\times[1\times10]\times[10\times2]\times[2\times10]$

$$
\begin{aligned}
c(i,i+3) = \min\{ & c(i,i) + c(i+1,i+3) + p_i p_{i+1} p_{i+4}, \\
& c(i,i+1) + c(i+2,i+3) + p_i p_{i+2} p_{i+4}, \\
& c(i,i+2) + c(i+3,i+3) + p_i p_{i+3} p_{i+4}\}
\end{aligned}
$$

	1	2	3	4	5
1	0	50	150	90	
2		0	50	30	90
3			0	20	40
4				0	200
5					0

$c(i,j),\ i \le j$

	1	2	3	4	5
1		1	2	2	
2			2	2	2
3				3	3
4					4
5					

$s(i,j),\ i \le j$

Example:

$[10\times5]\times[5\times1]\times[1\times10]\times[10\times2]\times[2\times10]$

$c(i,i+4) = \min\{c(i,i) + c(i+1,i+4) + p_i p_{i+1} p_{i+5},$

$c(i,i+1) + c(i+2,i+4) + p_i p_{i+2} p_{i+5},\ c(i,i+2) + c(i+3,i+4) + p_i p_{i+3} p_{i+5},$

$c(i,i+3) + c(i+4,i+4) + p_i p_{i+4} p_{i+5}\}$

	1	2	3	4	5
1	0	50	150	90	190
2		0	50	30	90
3			0	20	40
4				0	200
5					0

$c(i,j),\ i \le j$

	1	2	3	4	5
1		1	2	2	2
2			2	2	2
3				3	3
4					4
5					

$s(i,j),\ i \le j$

Print optimal parenthesis

Alg.: PRINT-OPTIMAL-PARENS(s, i, j)

if $i == j$

    print "A"$_i$

else print "("

    PRINT-OPTIMAL-PARENS(s, i, $s[i, j]$)

    PRINT-OPTIMAL-PARENS(s, $s[i, j] + 1$, j)

    print ")"

Example:

Optimal multiplication sequence

s(1,5) = 2

▶ $M_{15} = M_{12} \times M_{35}$

	1	2	3	4	5
1	0	50	150	90	190
2		0	50	30	90
3			0	20	40
4				0	200
5					0

c(i,j), i ≤ j

	1	2	3	4	5
1		1	2	2	2
2			2	2	2
3				3	3
4					4
5					

s(i,j), i ≤ j

Example:

$M_{15} = M_{12} \times M_{35}$

$s(1,2) = 1$ ▶ $M_{12} = M_{11} \times M_{22}$

$\rightarrow M_{15} = (M_{11} \times M_{22}) \times M_{35}$

	1	2	3	4	5
1	0	50	150	90	190
2		0	50	30	90
3			0	20	40
4				0	200
5					0

c(i,j), i ≤ j

	1	2	3	4	5
1		1	2	2	2
2			2	2	2
3				3	4
4					4
5					

s(i,j), i ≤ j

Example:

$M_{15} = (M_{11} \times M_{22}) \times M_{35}$

$s(3,5) = 4 \blacktriangleright M_{35} = M_{34} \times M_{55}$

$\rightarrow M_{15} = (M_{11} \times M_{22}) \times (M_{34} \times M_{55})$

	1	2	3	4	5
1	0	50	150	90	190
2		0	50	30	90
3			0	20	40
4				0	200
5					0

c(i,j), i ≤ j

	1	2	3	4	5
1		1	2	2	2
2			2	2	2
3				3	4
4					4
5					

s(i,j), i ≤ j

Example:

$M_{15} = (M_{11} \times M_{22}) \times (M_{34} \times M_{55})$

$s(3,4) = 3$ ▶ $M_{34} = M_{33} \times M_{44}$

$\rightarrow M_{15} = (M_{11} \times M_{22}) \times ((M_{33} \times M_{44}) \times M_{55})$

	1	2	3	4	5
1	0	50	150	90	190
2		0	50	30	90
3			0	20	40
4				0	200
5					0

$c(i,j),\ i \leq j$

	1	2	3	4	5
1		1	2	2	2
2			2	2	2
3				3	4
4					4
5					

$s(i,j),\ i \leq j$

Analysis

- Our algorithm computes the minimum-cost table m and the split table s
- The optimal solution can be constructed from the split table s
- Each entry s[i, j]=k shows where to split the product Ai Ai+1 … Aj for the minimum cost
- There are 3 nested loops and each can iterate at most n times, so the total running time is Θ(n3).

Memoization

- Offers the efficiency of the usual DP approach while maintaining top-down strategy
- Idea is to memoize the natural, but inefficient, recursive algorithm

Memoization

- Maintains an entry in a table for the soln to each subproblem
- Each table entry contains a special value to indicate that the entry has yet to be filled in
- When the subproblem is first encountered its solution is computed and then stored in the table
- Each subsequent time that the subproblem encountered the value stored in the table is simply looked up and returned

Memoized Matrix-Chain

Alg.: MEMOIZED-MATRIX-CHAIN(p)

1. $n \leftarrow$ length[p] – 1
2. **for** $i \leftarrow 1$ **to** n
3. **do for** $j \leftarrow i$ **to** n
4. **do** $m[i, j] \leftarrow \infty$
5. **return** LOOKUP-CHAIN(p, 1, n)

Initialize the m table with large values that indicate whether the values of $m[i, j]$ have been computed

Top-down approach

Memoized Matrix-Chain

Alg.: LOOKUP-CHAIN(p, i, j)

if m[i, j] $< \infty$

  then return m[i, j]

if i = j

  then m[i, j] $\leftarrow$ 0

  else for k $\leftarrow$ i **to** j – 1

    do q $\leftarrow$ LOOKUP-CHAIN(p, i, k) +

     LOOKUP-CHAIN(p, k+1, j) + $p_{i-1}p_kp_j$

     if q < m[i, j]

      then m[i, j] $\leftarrow$ q

return m[i, j]