Matrix Chain Multiplication by Dynamic Programming

Spring 2022

Outline

Outline

- Review of matrix multiplication.
- The chain matrix multiplication problem.
- A dynamic programming algorithm for chain matrix multiplication.

Rules to multiply Two Matrices

The number of ***columns of the 1st matrix*** *must equal the number of* ***rows of the 2nd matrix****.*
And the result will have the same number of ***rows as the 1st matrix****, and the same number of* ***columns as the 2nd matrix****.*

$$(m \times n) \cdot (n \times k) = (m \times k)$$

product is defined

$$\begin{array}{c} \vec{a_1}\rightarrow \\ \vec{a_2}\rightarrow \end{array} \begin{bmatrix}1 & 7\\ 2 & 4\end{bmatrix} \cdot \overset{\begin{array}{cc} \vec{b_1} & \vec{b_2} \\ \downarrow & \downarrow \end{array}}{\begin{bmatrix}3 & 3\\ 5 & 2\end{bmatrix}} = \begin{bmatrix}\overrightarrow{a_1}\cdot\overrightarrow{b_1} & \overrightarrow{a_1}\cdot\overrightarrow{b_2}\\ \overrightarrow{a_2}\cdot\overrightarrow{b_1} & \overrightarrow{a_2}\cdot\overrightarrow{b_2}\end{bmatrix}$$

$$A \qquad\qquad B \qquad\qquad\qquad C$$

Matrix Multiplication

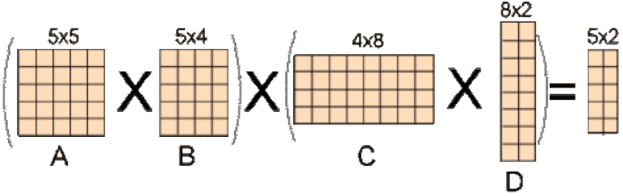
"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \\ & \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix} \checkmark$$

Matrix Multiplication


5x5
5x4
4x8
8x2
5x2
A
B
C
D

Matrix: An $n \times m$ matrix $A = [a[i,j]]$ is a two-dimensional array

$$A = \begin{bmatrix} a[1,1] & a[1,2] & \cdots & a[1,m-1] & a[1,m] \\ a[2,1] & a[2,2] & \cdots & a[2,m-1] & a[2,m] \\ \vdots & \vdots & & \vdots & \vdots \\ a[n,1] & a[n,2] & \cdots & a[n,m-1] & a[n,m] \end{bmatrix},$$

which has n rows and m columns.

Example

A 4×5 matrix:

$$\begin{matrix} 12 & 8 & 9 & 7 & 6 \\ 7 & 6 & 89 & 56 & 2 \\ 5 & 5 & 6 & 9 & 10 \\ 8 & 6 & 0 & -8 & -1 \end{matrix}.$$

Review of Matrix Multiplication

The product $C = AB$ of a $p \times q$ matrix A and a $q \times r$ matrix B is a $p \times r$ matrix C given by

$$c[i,j] = \sum_{k=1}^{q} a[i,k]b[k,j], \qquad \text{for } 1 \le i \le p \text{ and } 1 \le j \le r$$

Review of Matrix Multiplication

The product $C = AB$ of a $p \times q$ matrix A and a $q \times r$ matrix B is a $p \times r$ matrix C given by

$$c[i,j] = \sum_{k=1}^{q} a[i,k]b[k,j], \qquad \text{for } 1 \leq i \leq p \text{ and } 1 \leq j \leq r$$

Complexity of Matrix multiplication: Note that C has pr entries and each entry takes $\Theta(q)$ time to compute so the total procedure takes $\Theta(pqr)$ time.

Review of Matrix

The product $C = AB$ of a $p \times q$ matrix A and a $q \times r$ matrix B is a $p \times r$ matrix C given by

$$c[i,j] = \sum_{k=1}^{q} a[i,k]b[k,j], \quad \text{for } 1 \le i \le p \text{ and } 1 \le j \le r$$

Complexity of Matrix multiplication: Note that C has pr entries and each entry takes $\Theta(q)$ time to compute so the total procedure takes $\Theta(pqr)$ time.

Example

$$A = \begin{bmatrix} 1 & 8 & 9 \\ 7 & 6 & -1 \\ 5 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 8 \\ 7 & 6 \\ 5 & 5 \end{bmatrix}, \quad C = AB = \begin{bmatrix} 102 & 101 \\ 44 & 87 \\ 70 & 100 \end{bmatrix}.$$

- Matrix multiplication is associative, e.g.,

Remarks on Matrix Multiplication

- Matrix multiplication is associative, e.g.,

$$A_1A_2A_3 = (A_1A_2)A_3 = A_1(A_2A_3),$$

so parenthesization does not change result.

Remarks on Matrix Multiplication

- Matrix multiplication is associative, e.g.,

$$A_1A_2A_3 = (A_1A_2)A_3 = A_1(A_2A_3),$$

  so parenthesization does not change result.

- Matrix multiplication is NOT commutative, e.g.,

Remarks on Matrix Multiplication

- Matrix multiplication is associative, e.g.,

  $$A_1A_2A_3 = (A_1A_2)A_3 = A_1(A_2A_3),$$

  so parenthesization does not change result.

- Matrix multiplication is NOT commutative, e.g.,

  $$A_1A_2 \neq A_2A_1$$

- Given $p \times q$ matrix A, $q \times r$ matrix B and $r \times s$ matrix C, ABC can be computed in twoways:

Matrix Multiplication of ABC

- Given $p \times q$ matrix A, $q \times r$ matrix B and $r \times s$ matrix C, ABC can be computed in two ways: $(AB)C$ and $A(BC)$

Matrix Multiplication of ABC

- Given $p \times q$ matrix A, $q \times r$ matrix B and $r \times s$ matrix C, ABC can be computed in two ways: $(AB)C$ and $A(BC)$
- The number of multiplications needed are:

$$
\begin{aligned}
mult[(AB)C] &= pqr + prs, \\
mult[A(BC)] &= qrs + pqs.
\end{aligned}
$$

Matrix Multiplication of ABC

- Given $p \times q$ matrix A, $q \times r$ matrix B and $r \times s$ matrix C, ABC can be computed in two ways: $(AB)C$ and $A(BC)$
- The number of multiplications needed are:

$$\begin{aligned} mult[(AB)C] &= pqr + prs, \\ mult[A(BC)] &= qrs + pqs. \end{aligned}$$

Example

For $p = 5$, $q = 4$, $r = 6$ and $s = 2$,

$$\begin{aligned} mult[(AB)C] &= 180, \\ mult[A(BC)] &= 88. \end{aligned}$$

A big difference!

Matrix Multiplication of *ABC*

- Given $p \times q$ matrix A, $q \times r$ matrix B and $r \times s$ matrix C, ABC can be computed in two ways: $(AB)C$ and $A(BC)$
- The number of multiplications needed are:

$$mult[(AB)C] = pqr + prs,$$
$$mult[A(BC)] = qrs + pqs.$$

Example

For $p = 5$, $q = 4$, $r = 6$ and $s = 2$,

$$mult[(AB)C] = 180,$$
$$mult[A(BC)] = 88.$$

A big difference!

Implication: Multiplication "sequence" (parenthesization) is important!!

Conclusion

Order of operations makes a huge difference. How do we compute the minimum?

Each parenthesization defines a set of **n-1** matrix multiplications. We just need to pick the parenthesization that corresponds to the best ordering.

Trying all possible parenthesizations is a bad idea!!!

- Review of matrix multiplication.
- The chain matrix multiplication problem.
- A dynamic programming algorithm for chain matrix multiplication.

The Chain Matrix Multiplication Problem

Definition (Chain matrix multiplication problem)

Given dimensions $p_0, p_1, \ldots, p_n$, corresponding to matrix sequence A_1, $A_2, \ldots, A_n$ in which A_i has dimension $p_{i-1} \times p_i$, determine the "multiplication sequence" that minimizes the number of scalar multiplications in computing $A_1 A_2 \cdots A_n$.

- i.e.,, determine how to parenthesize the multiplications.

The Chain Matrix Multiplication Problem

Definition (Chain matrix multiplication problem)

Given dimensions $p_0, p_1, \ldots, p_n$, corresponding to matrix sequence A_1, $A_2, \ldots, A_n$ in which A_i has dimension $p_{i-1} \times p_i$, determine the "multiplication sequence" that minimizes the number of scalar multiplications in computing $A_1 A_2 \cdots A_n$.

- i.e.,, determine how to parenthesize the multiplications.

Example

$$A_1A_2A_3A_4 = (A_1A_2)(A_3A_4) = A_1(A_2(A_3A_4)) = A_1((A_2A_3)A_4)$$
$$= ((A_1A_2)A_3)(A_4) = (A_1(A_2A_3))(A_4)$$

Exhaustive search: $\Omega(4^n/n^{3/2})$.

The Chain Matrix Multiplication Problem

Definition (Chain matrix multiplication problem)

Given dimensions $p_0, p_1, \ldots, p_n$, corresponding to matrix sequence A_1, A_2, ..., A_n in which A_i has dimension $p_{i-1} \times p_i$, determine the "multiplication sequence" that minimizes the number of scalar multiplications in computing $A_1 A_2 \cdots A_n$.

- i.e.,, determine how to parenthesize the multiplications.

Example

$$A_1A_2A_3A_4 = (A_1A_2)(A_3A_4) = A_1(A_2(A_3A_4)) = A_1((A_2A_3)A_4)$$
$$= ((A_1A_2)A_3)(A_4) = (A_1(A_2A_3))(A_4)$$

Exhaustive search: $\Omega(4^n/n^{3/2})$.

Question

Is there a better approach?

The Chain Matrix Multiplication Problem

Definition (Chain matrix multiplication problem)

Given dimensions $p_0, p_1, \ldots, p_n$, corresponding to matrix sequence A_1, A_2, ..., A_n in which A_i has dimension $p_{i-1} \times p_i$, determine the "multiplication sequence" that minimizes the number of scalar multiplications in computing $A_1 A_2 \cdots A_n$.

- i.e.,, determine how to parenthesize the multiplications.

Example

$$
\begin{aligned}
A_1A_2A_3A_4 \quad &= (A_1A_2)(A_3A_4) = A_1(A_2(A_3A_4)) = A_1((A_2A_3)A_4) \\
&= ((A_1A_2)A_3)(A_4) = (A_1(A_2A_3))(A_4)
\end{aligned}
$$

Exhaustive search: $\Omega(4^n/n^{3/2})$.

Question

Is there a better approach?

Yes – DP

Developing a Dynamic Programming Algorithm

Step 1: Define Space of Subproblems

Step 1: Define Space of Subproblems

- Original Problem:

  Determine minimal cost multiplication sequence for $A_{1..n}$.

Developing a Dynamic Programming Algorithm

Step 1: Define Space of Subproblems

- Original Problem:
  Determine minimal cost multiplication sequence for $A_{1..n}$.
- Subproblems: For every pair $1 \leq i \leq j \leq n$:
  Determine minimal cost multiplication sequence for
  $A_{i..j} = A_i A_{i+1} \cdots A_j$.
  - Note that $A_{i..j}$ is a $p_{i-1} \times p_j$ matrix.

Example:

dimension $p_{i-1} \times p_i$

i= 1

$$
\begin{array}{lll}
A_1 & 30 \times 35 & = p_0 \times p_1 \\
A_2 & 35 \times 15 & = p_1 \times p_2 \\
A_3 & 15 \times 5 & = p_2 \times p_3 \\
A_4 & 5 \times 10 & = p_3 \times p_4 \\
A_5 & 10 \times 20 & = p_4 \times p_5 \\
A_6 & 20 \times 25 & = p_5 \times p_6
\end{array}
$$

Given a chain $A_1, A_2, \ldots, A_n$ of n matrices, where for i=1, 2, …, n, matrix A_i has dimension $p_{i-1} \times p_i$

Note

In the matrix-chain multiplication problem, we are not actually multiplying matrices.

Our goal is only to ***determine an order for multiplying matrices*** *that has the lowest cost.*

Typically, the time invested in determining this optimal order is more than paid for by the time saved later on when actually performing the matrix multiplications (such as performing only 7500 scalar multiplications instead of 75,000).

Developing a Dynamic Programming Algorithm

Step 1: Define Space of Subproblems

- Original Problem:
  Determine minimal cost multiplication sequence for $A_{1..n}$.
- Subproblems: For every pair $1 \leq i \leq j \leq n$:
  Determine minimal cost multiplication sequence for
  $A_{i..j} = A_i A_{i+1} \cdots A_j$.
  Note that $A_{i..j}$ is a $p_{i-1} \times p_j$ matrix.
- There are $\Theta(n^2)$ such subproblems. (Why?)
- How can we solve larger problems using subproblem solutions?

Relationships among subproblems

At the last step of *any* optimal multiplication sequence (for a subbroblem), there is some k such that the two matrices $A_{i..k}$ and $A_{k+1..j}$ are multipled together.

Relationships among subproblems

At the last step of *any* optimal multiplication sequence (for a subbroblem), there is some k such that the two matrices $A_{i..k}$ and $A_{k+1..j}$ are multipled together. That is,

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

Relationships among subproblems

At the last step of *any* optimal multiplication sequence (for a subbroblem), there is some k such that the two matrices $A_{i..k}$ and $A_{k+1..j}$ are multipled together. That is,

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

Relationships among subproblems

At the last step of *any* optimal multiplication sequence (for a subbroblem), there is some k such that the two matrices $A_{i..k}$ and $A_{k+1..j}$ are multipled together. That is,

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

Question

How do we decide where to split the chain (what is k)?

Relationships among subproblems

At the last step of *any* optimal multiplication sequence (for a subbroblem), there is some k such that the two matrices $A_{i..k}$ and $A_{k+1..j}$ are multipled together. That is,

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

Question

How do we decide where to split the chain (what is k)?

ANS: Can be *any* k. Need to check all possible values.

Relationships among subproblems

At the last step of *any* optimal multiplication sequence (for a subbroblem), there is some k such that the two matrices $A_{i..k}$ and $A_{k+1..j}$ are multipled together. That is,

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

Question

How do we decide where to split the chain (what is k)?

ANS: Can be *any* k. Need to check all possible values.

Question

How do we parenthesize the two subchains $A_{i..k}$ and $A_{k+1..j}$?

Relationships among subproblems

At the last step of *any* optimal multiplication sequence (for a subbroblem), there is some k such that the two matrices $A_{i..k}$ and $A_{k+1..j}$ are multipled together. That is,

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

Question

How do we decide where to split the chain (what is k)?

ANS: Can be *any* k. Need to check all possible values.

Question

How do we parenthesize the two subchains $A_{i..k}$ and $A_{k+1..j}$?

ANS: $A_{i..k}$ and $A_{k+1..j}$ must be computed optimally, so we can apply the same procedure *recursively*.

If the “optimal” solution of $A_{i..j}$ involves splitting into $A_{i..k}$ and $A_{k+1..j}$ at the final step, then parenthesization of $A_{i..k}$ and $A_{k+1..j}$ in the optimal solution must also be optimal

Optimal Structure Property

If the "optimal" solution of $A_{i..j}$ involves splitting into $A_{i..k}$ and $A_{k+1..j}$ at the final step, then parenthesization of $A_{i..k}$ and $A_{k+1..j}$ in the optimal solution must also be optimal

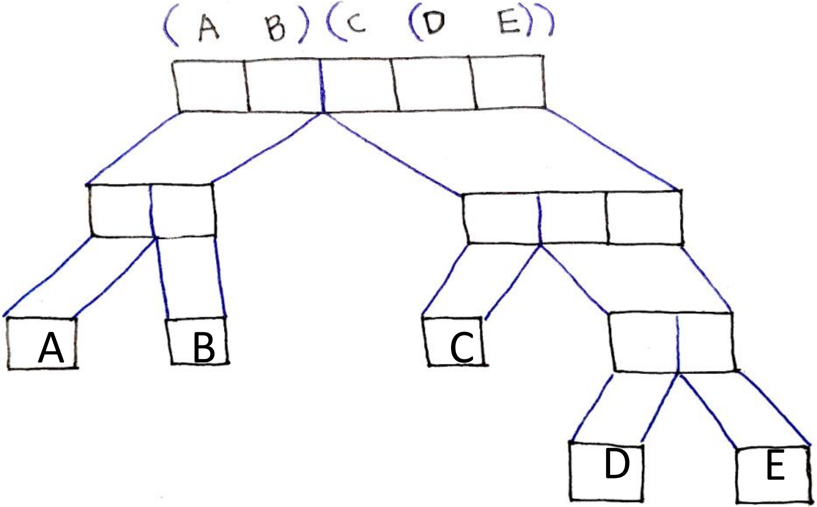
- If parenthesization of $A_{i..k}$ was not optimal, it could be replaced by a cheaper parenthesization, yielding a cheaper final solution, constradicting optimality

Optimal Structure Property

If the "optimal" solution of $A_{i..j}$ involves splitting into $A_{i..k}$ and $A_{k+1..j}$ at the final step, then parenthesization of $A_{i..k}$ and $A_{k+1..j}$ in the optimal solution must also be optimal

- If parenthesization of $A_{i..k}$ was not optimal, it could be replaced by a cheaper parenthesization, yielding a cheaper final solution, constradicting optimality
- Similarly, if parenthesization of $A_{k+1..j}$ was not optimal, it could be replaced by a cheaper parenthesization, again yielding contradiction of cheaper final solution.

Dynamic Programming Approach



Relationships among subproblems

Step 2: Constructing optimal solutions from optimal subproblem solution

Step 2: Constructing optimal solutions from optimal subproblem solution

For $1 \leq i \leq j \leq n$, let $m[i, j]$ denote the minimum number of multiplications needed to compute $A_{i..j}$. This optimum cost must satisfy the following recursive definition.

Step 2: Constructing optimal solutions from optimal subproblem solution

For $1 \leq i \leq j \leq n$, let $m[i,j]$ denote the minimum number of multiplications needed to compute $A_{i..j}$. This optimum cost must satisfy the following recursive definition.

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

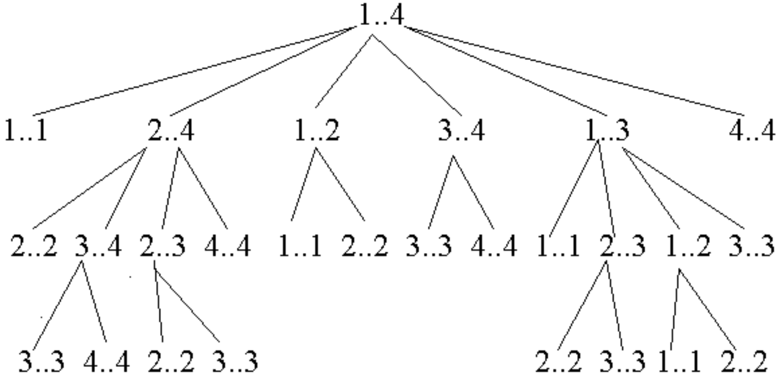
Step 2: Constructing optimal solutions from optimal subproblem solution

For $1 \le i \le j \le n$, let $m[i,j]$ denote the minimum number of multiplications needed to compute $A_{i..j}$. This optimum cost must satisfy the following recursive definition.

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

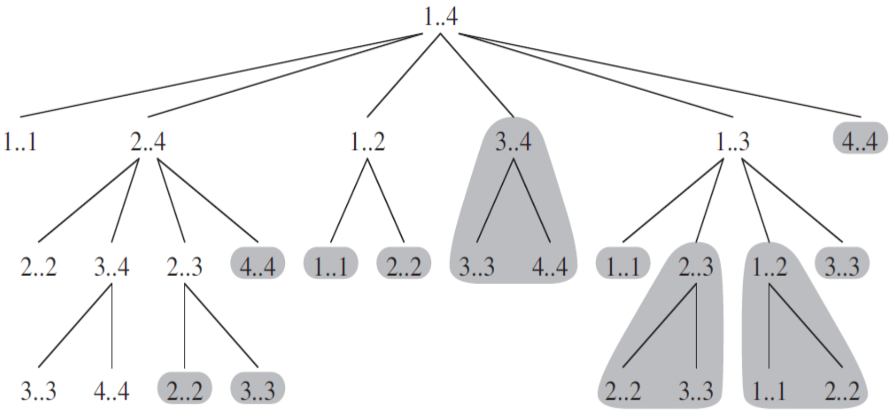
$$A_{i..j} = A_{i..k} A_{k+1..j}$$

Overlapping subproblems (contd.)



recursion tree for computation of RECURSIVE-MATRIX-CHAIN(p1,4)

Overlapping Sub-Problems



Step 3: Bottom-up computation of $m[i,j]$.

Developing a Dynamic Programming Algorithm

Step 3: Bottom-up computation of $m[i,j]$.

Recurrence:

$$m[i,j] = \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_k p_j)$$

Step 3: Bottom-up computation of $m[i,j]$.

Recurrence:

$$m[i,j] = \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j)$$

Fill in the $m[i, j]$ table in an order, such that when it is time to calculate $m[i,j]$, the values of $m[i,k]$ and $m[k+1,j]$ for all k are already available.

Step 3: **Bottom-up computation of $m[i,j]$.**

Recurrence:

$$m[i,j] = \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_kp_j)$$

Fill in the $m[i, j]$ table in an order, such that when it is time to calculate $m[i,j]$, the values of $m[i,k]$ and $m[k+1,j]$ for all k are already available.

An easy way to ensure this is to compute them in increasing order of the size $(j - i)$ of the matrix-chain $A_{i..j}$:

$m[1,2], m[2,3], m[3,4], \ldots, m[n-3,n-2], m[n-2,n-1], m[n-1,n]$

Set of 2 Matrices

Developing a Dynamic Programming Algorithm

Step 3: Bottom-up computation of $m[i,j]$.

Recurrence:

$$m[i,j] = \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j)$$

Fill in the $m[i, j]$ table in an order, such that when it is time to calculate $m[i,j]$, the values of $m[i,k]$ and $m[k+1,j]$ for all k are already available.

An easy way to ensure this is to compute them in increasing order of the size $(j - i)$ of the matrix-chain $A_{i..j}$:

$m[1,2]$, $m[2,3]$, $m[3,4]$, ..., $m[n-3,n-2]$, $m[n-2,n-1]$, $m[n-1,n]$
$m[1,3]$, $m[2,4]$, $m[3,5]$, ..., $m[n-3,n-1]$, $m[n-2,n]$

Set of 3 Matrices

Step 3: Bottom-up computation of $m[i,j]$.

Recurrence:

$$m[i,j] = \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_k p_j)$$

Fill in the $m[i, j]$ table in an order, such that when it is time to calculate $m[i,j]$, the values of $m[i,k]$ and $m[k+1,j]$ for all k are already available.

An easy way to ensure this is to compute them in increasing order of the size $(j - i)$ of the matrix-chain $A_{i..j}$:

$m[1,2], m[2,3], m[3,4], \ldots, m[n-3,n-2], m[n-2,n-1], m[n-1,n]$
$m[1,3], m[2,4], m[3,5], \ldots, m[n-3,n-1], m[n-2,n]$
$m[1,4], m[2,5], m[3,6], \ldots, m[n-3,n]$

Set of 4 Matrices

Step 3: **Bottom-up computation of $m[i,j]$.**

Recurrence:

$$m[i,j] = \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_kp_j)$$

Fill in the $m[i, j]$ table in an order, such that when it is time to calculate $m[i,j]$, the values of $m[i,k]$ and $m[k+1,j]$ for all k are already available.

An easy way to ensure this is to compute them in increasing order of the size $(j - i)$ of the matrix-chain $A_{i..j}$:

$m[1,2], m[2,3], m[3,4], \ldots, m[n-3,n-2], m[n-2,n-1], m[n-1,n]$
$m[1,3], m[2,4], m[3,5], \ldots, m[n-3,n-1], m[n-2,n]$
$m[1,4], m[2,5], m[3,6], \ldots, m[n-3,n]$
$\ldots$
$m[1,n-1],\ m[2,n]$

Developing a Dynamic Programming Algorithm

Step 3: **Bottom-up computation of $m[i,j]$.**

Recurrence:

$$m[i,j] = \min_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right)$$

Fill in the $m[i, j]$ table in an order, such that when it is time to calculate $m[i,j]$, the values of $m[i,k]$ and $m[k+1,j]$ for all k are already available.

An easy way to ensure this is to compute them in increasing order of the size $(j - i)$ of the matrix-chain $A_{i..j}$:

$m[1,2], m[2,3], m[3,4], \ldots, m[n-3,n-2], m[n-2,n-1], m[n-1,n]$
$m[1,3], m[2,4], m[3,5], \ldots, m[n-3,n-1], m[n-2,n]$
$m[1,4], m[2,5], m[3,6], \ldots, m[n-3,n]$
$\ldots$
$m[1,n-1], \ m[2,n]$
$m[1,n]$

Example for the Bottom-Up Computation

Example

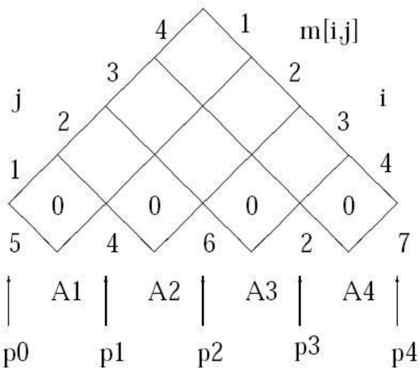
A chain of four matrices A_1, A_2, A_3 and A_4, with $p_0 = 5$, $p_1 = 4$, $p_2 = 6$, $p_3 = 2$ and $p_4 = 7$. Find $m[1, 4]$.

Example for the Bottom-Up Computation

Example

A chain of four matrices A_1, A_2, A_3 and A_4, with $p_0 = 5$, $p_1 = 4$, $p_2 = 6$, $p_3 = 2$ and $p_4 = 7$. Find $m[1, 4]$.

S0: Initialization



Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Step 1: Computing $m[1, 2]$

	1	2	3	4
1	0			
2	NA	0		
3	NA	NA	0	
4	NA	NA	NA	0

$A1_{5x\mathbf{4}}$

$A2_{\mathbf{4}x6}$

COST= 5 x **4** x 6=120

M [1,2]= 120

Example – Continued

$A1_{5x4}$, $A2_{4x6}$, $A3_{6x2}$, $A4_{2x7}$

Step 1: Computing $m[1, 2]$

	1	2	3	4
1	0	**120**		
2	NA	0		
3	NA	NA	0	
4	NA	NA	NA	0

$A1_{5x\mathbf{4}}$

$A2_{\mathbf{4}x6}$

COST= 5 x **4** x 6=120

M [1,2]= 120

$A1_{5x4}$, $A2_{4x6}$, $A3_{6x2}$, $A4_{2x7}$

Step 1: Computing *m*[2, 3]

	1	2	3	4
1	0	120		
2	NA	0	**48**	
3	NA	NA	0	
4	NA	NA	NA	0

$A2_{4x6}$
$A3_{6x2}$

COST= 4 x **6** x 2=48
M [2, 3]= 48

Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Step 1: Computing $m[3,4]$

	1	2	3	4
1	0	120		
2	NA	0	48	
3	NA	NA	0	**84**
4	NA	NA	NA	0

$A3_{6x\mathbf{2}}$

$A4_{\mathbf{2}x7}$

COST= 6 x **2** x 7=84

M [3,4]= 84

Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Step 1: Computing $m[1, 3]$

	1	2	3	4
1	0	120		
2	NA	0	48	
3	NA	NA	0	84
4	NA	NA	NA	0

A1. (A2 . A3) or (A1. A2). A3

Two Possibilities to multiply 3 matrices

Example – Continued

$A1_{5x4}$, $A2_{4x6}$, $A3_{6x2}$, $A4_{2x7}$

Step 1: Computing $m[1, 3]$

A1. (A2 . A3)   or   (A1. A2). A3

$A1_{5x4}$, ($A2_{4x6}$, $A3_{6x2}$)
=m[1,1]+m[2,3]+5*4*2
=0+ 48 + 40
=88

($A1_{5x4}$. $A2_{4x6}$) . $A3_{6x2}$
= m[1,2]+ m[3,3]+ 5x6x2
=120 + 0 + 60
=180

Pick Minimum

Two Possibilities to multiply 3 matrices

Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Step 1: Computing $m[1, 3]$

	1	2	3	4
1	0	120	88	
2	NA	0	48	
3	NA	NA	0	84
4	NA	NA	NA	0

A1. (A2 . A3) or (A1. A2). A3

Two Possibilities to multiply 3 matrices

Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Step 1: Computing $m[2, 4]$

	1	2	3	4
1	0	120	88	
2	NA	0	48	
3	NA	NA	0	84
4	NA	NA	NA	0

A2. (A3 . A4) or (A2. A3). A4

Two Possibilities to multiply 3 matrices

Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Step 1: Computing $m[2, 4]$

A2. (A3 . A4)   or   (A2. A3). A4

$A2_{4x\mathbf{6}}$. ($A3_{\mathbf{6}x2}$, $A4_{2x7}$)

=m[2,2]+m[3,4]+ 4x6x7

=0+ 84 + 168

=252

($A2_{4x6}$. $A3_{6x\mathbf{2}}$) . $A4_{\mathbf{2}x7}$

= m[1,2]+ m[3,3]+ 4x2x7

=48 + 0 + 56

=104

Pick Minimum

Two Possibilities to multiply 3 matrices

Example – Continued

$A1_{5x4}$, $A2_{4x6}$, $A3_{6x2}$, $A4_{2x7}$

Step 1: Computing $m[2,4]$

	1	2	3	4
1	0	120	88	
2	NA	0	48	**104**
3	NA	NA	0	84
4	NA	NA	NA	0

A1. (A2 . A3) or (A1. A2). A3

Two Possibilities to multiply 3 matrices

Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Step 1: Computing $m[2, 4]$

	1	**2**	**3**	**4**
1	0	120	88	
2	NA	0	48	**104**
3	NA	NA	0	84
4	NA	NA	NA	0

A1. (A2 . A3. A4) or (A1. A2). (A3 . A4) or (A1 .A2. A3). A4

Three Possibilities

$A1_{5x4}$, $A2_{4x6}$, $A3_{6x2}$, $A4_{2x7}$

A1. (A2 . A3. A4) or (A1. A2). (A3 . A4) or (A1 .A2. A3). A4

M[1,4]= m[1,1] + m[2,4] + 5 x4 x 7,
m[1,2]+ m[3,4] + 5x6x7,
m[1, 3] + m[4,4] + 5x 2x 7

M[1,4]= 0 + 104+ 140,
120+84+210,
88+0+70

M[1,4]= 244,
414,
158

Three Possibilities

Pick Minimum

Example – Continued

$A1_{5x4}$, $A2_{4x6}$, $A3_{6x2}$, $A4_{2x7}$

Step 1: Computing $m[1, 4]$

	1	2	3	4
1	0	120	88	**158**
2	NA	0	48	104
3	NA	NA	0	84
4	NA	NA	NA	0

A1. (A2 . A3. A4) or (A1. A2). (A3 . A4) or (A1 .A2. A3). A4

Three Possibilities

Example – Continued

$A1_{5x\mathbf{4}}$, $A2_{\mathbf{4}x6}$, $A3_{6x\mathbf{2}}$, $A4_{\mathbf{2}x7}$

Split point Matrix

S	1	2	3	4
1		1	1	**3**
2			2	3
3				3
4				

Example – Continued

Step 1: Computing $m[1, 2]$

By definition

$$m[1,2] = \min_{1 \le k < 2} (m[1,k] + m[k+1,2] + p_0 p_k p_2)$$

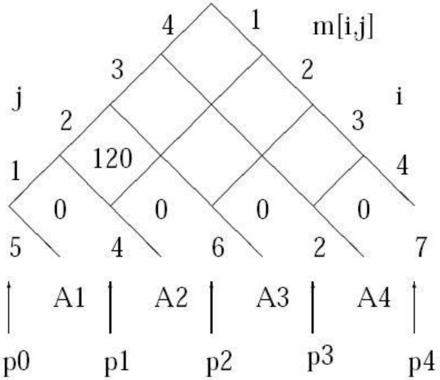
$$= m[1,1] + m[2,2] + p_0 p_1 p_2 = 120 .$$

Step 1: Computing $m[1, 2]$

By definition

$$m[1,2] = \min_{1 \le k < 2} (m[1,k] + m[k+1,2] + p_0 p_k p_2)$$

$$= m[1,1] + m[2,2] + p_0 p_1 p_2 = 120.$$



Example – Continued

Step 2: Computing $m[2,3]$

By definition

$$m[2,3] = \min_{2 \le k < 3} (m[2,k] + pm[k+1,3] + p_1 p_k p_3)$$

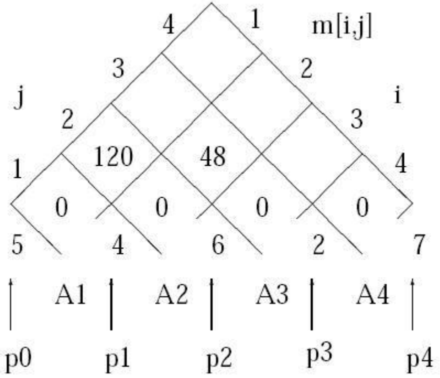
$$= m[2,2] + m[3,3] + p_1 p_2 p_3 = 48 .$$

Step 2: Computing $m[2,3]$

By definition

$$m[2,3] = \min_{2 \le k < 3} (m[2,k] + pm[k+1,3] + p_1 p_k p_3)$$

$$= m[2,2] + m[3,3] + p_1 p_2 p_3 = 48\,.$$



Example – Continued

Step 3: Computing $m[3,4]$

By definition

$$m[3,4] = \min_{3 \le k < 4} (m[3,k] + m[k+1,4] + p_2 p_k p_4)$$

Step 3: Computing $m[3, 4]$

By definition

$$m[3,4] = \min_{3 \le k < 4} (m[3,k] + m[k+1,4] + p_2 p_k p_4)$$

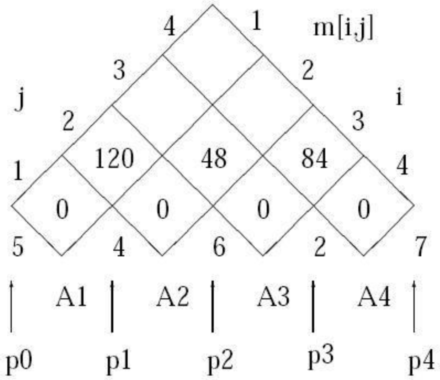
$$= m[3,3] + m[4,4] + p_2 p_3 p_4 = 84.$$

Step 3: Computing $m[3,4]$

By definition

$$m[3,4] = \min_{3 \le k < 4} (m[3,k] + m[k+1,4] + p_2 p_k p_4)$$

$$= m[3,3] + m[4,4] + p_2 p_3 p_4 = 84 .$$



Example – Continued

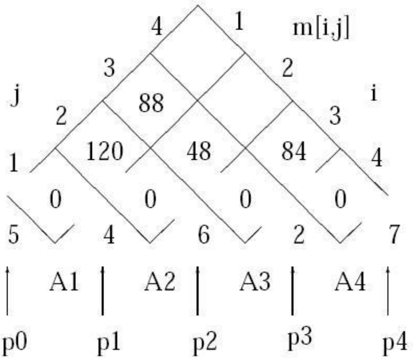
Step 4: Computing $m[1,3]$

By definition

$$m[1,3] = \min_{1 \le k < 3} \left(m[1,k] + m[k+1,3] + p_0 p_k p_3 \right)$$

$$= \min \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 \end{cases}$$

$$= 88 \ .$$



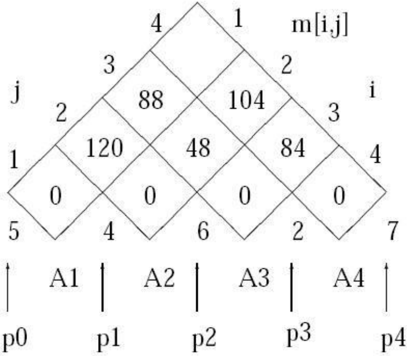
Step 5: Computing $m[2,4]$

By definition

$$m[2,4] = \min_{2 \le k < 4} \left(m[2,k] + m[k+1,4] + p_1 p_k p_4 \right)$$

$$= \min \begin{cases} m[2,2] + m[3,4] + p_1 p_2 p_4 \\ m[2,3] + m[4,4] + p_1 p_3 p_4 \end{cases}$$

$$= 104 \; .$$



Example – Continued

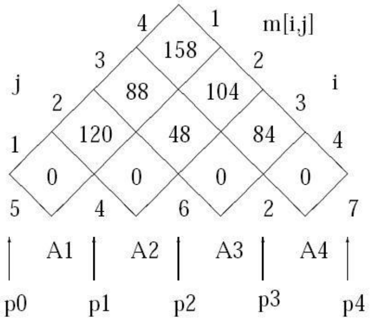
Step 6: Computing $m[1,4]$

By definition

$$m[1,4] = \min_{1 \le k < 4} \left(m[1,k] + m[k+1,4] + p_0 p_k p_4 \right)$$

$$= \min \left\{ \begin{array}{l} m[1,1] + m[2,4] + p_0 p_1 p_4 \\ m[1,2] + m[3,4] + p_0 p_2 p_4 \\ m[1,3] + m[4,4] + p_0 p_3 p_4 \end{array} \right.$$

$$= 158 \,.$$



Constructing a Solution

- $m[i,j]$ only keeps costs but not actual multiplication sequence.
- To solve problem, need to reconstruct multiplication sequence that yields $m[1,n]$.
- Solution: similar to previous DP algorithm(s) keep an auxillary array $s[*,*]$.
- $s[i,j] = k$ where k is the index that achieves minimum in

$$m[i,j] = \min_{i \leq k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_kp_j).$$

Step 4: Constructing optimal solution

Idea: Maintain an array $s[1..n, 1..n]$, where $s[i,j]$ denotes k for the optimal splitting in computing $A_{i..j} = A_{i..k}A_{k+1..j}$.

Step 4: Constructing optimal solution

Idea: Maintain an array $s[1..n, 1..n]$, where $s[i, j]$ denotes k for the optimal splitting in computing $A_{i..j} = A_{i..k}A_{k+1..j}$.

Question

How to Recover the Multiplication Sequence using $s[i, j]$?

Step 4: Constructing optimal solution

Idea: Maintain an array $s[1..n, 1..n]$, where $s[i,j]$ denotes k for the optimal splitting in computing $A_{i..j} = A_{i..k}A_{k+1..j}$.

Question

How to Recover the Multiplication Sequence using $s[i,j]$?

$s[1,n]$ $\quad$ $(A_1 \cdots A_{s[1,n]})(A_{s[1,n]+1} \cdots A_n)$

Step 4: Constructing optimal solution

Idea: Maintain an array $s[1..n, 1..n]$, where $s[i,j]$ denotes k for the optimal splitting in computing $A_{i..j} = A_{i..k}A_{k+1..j}$.

Question

How to Recover the Multiplication Sequence using $s[i,j]$?

$s[1,n]$ $\qquad (A_1 \cdots A_{s[1,n]})(A_{s[1,n]+1} \cdots A_n)$

$s[1,s[1,n]]$ $\qquad (A_1 \cdots A_{s[1,s[1,n]]})(A_{s[1,s[1,n]]+1} \cdots A_{s[1,n]})$

Step 4: Constructing optimal solution

Idea: Maintain an array $s[1..n, 1..n]$, where $s[i,j]$ denotes k for the optimal splitting in computing $A_{i..j} = A_{i..k}A_{k+1..j}$.

Question

How to Recover the Multiplication Sequence using $s[i,j]$?

$s[1,n]$	$(A_1 \cdots A_{s[1,n]})(A_{s[1,n]+1} \cdots A_n)$
$s[1,s[1,n]]$	$(A_1 \cdots A_{s[1,s[1,n]]})(A_{s[1,s[1,n]]+1} \cdots A_{s[1,n]})$
$s[s[1,n]+1,n]$	$(A_{s[1,n]+1} \cdots A_{s[s[1,n]+1,n]})(A_{s[s[1,n]+1,n]+1} \cdots A_n)$

Step 4: Constructing optimal solution

Idea: Maintain an array $s[1..n, 1..n]$, where $s[i,j]$ denotes k for the optimal splitting in computing $A_{i..j} = A_{i..k}A_{k+1..j}$.

Question

How to Recover the Multiplication Sequence using $s[i,j]$?

$s[1,n]$	$(A_1 \cdots A_{s[1,n]})(A_{s[1,n]+1} \cdots A_n)$
$s[1,s[1,n]]$	$(A_1 \cdots A_{s[1,s[1,n]]})(A_{s[1,s[1,n]]+1} \cdots A_{s[1,n]})$
$s[s[1,n]+1,n]$	$(A_{s[1,n]+1} \cdots A_{s[s[1,n]+1,n]})(A_{s[s[1,n]+1,n]+1} \cdots A_n)$
:	:

Step 4: Constructing optimal solution

Idea: Maintain an array $s[1..n, 1..n]$, where $s[i, j]$ denotes k for the optimal splitting in computing $A_{i..j} = A_{i..k}A_{k+1..j}$.

Question

How to Recover the Multiplication Sequence using $s[i, j]$?

$$
\begin{array}{ll}
s[1,n] & (A_1 \cdots A_{s[1,n]})\,(A_{s[1,n]+1} \cdots A_n) \\
s[1,s[1,n]] & (A_1 \cdots A_{s[1,s[1,n]]})\,(A_{s[1,s[1,n]]+1} \cdots A_{s[1,n]}) \\
s[s[1,n]+1,n] & (A_{s[1,n]+1} \cdots A_{s[s[1,n]+1,n]})\,(A_{s[s[1,n]+1,n]+1} \cdots A_n) \\
\vdots & \vdots
\end{array}
$$

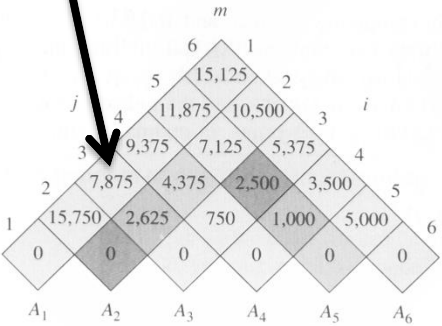
Apply recursively until multiplication sequence is completely determined.

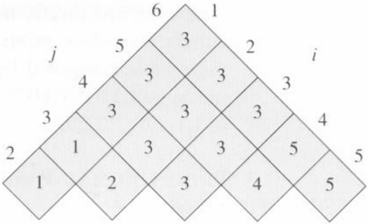
Example 2:

$$
\begin{array}{lll}
A_1 & 30 \times 35 & = p_0 \times p_1 \\
A_2 & 35 \times 15 & = p_1 \times p_2 \\
A_3 & 15 \times 5 & = p_2 \times p_3 \\
A_4 & 5 \times 10 & = p_3 \times p_4 \\
A_5 & 10 \times 20 & = p_4 \times p_5 \\
A_6 & 20 \times 25 & = p_5 \times p_6
\end{array}
$$

Order Products	Matrix Size	
A2 x A3 =	35 x 5	2,625
A1 (A2 x A3) =	30 X 5	5,250 + 2,625

the m and s table computed by MATRIX-CHAIN-ORDER for n=6





$m[2,5]=$
$min\{$
$m[2,2]+m[3,5]+p_1p_2p_5=0+2500+35\times15\times20=13000,$
$m[2,3]+m[4,5]+p_1p_3p_5=2625+1000+35\times5\times20=7125,$
$m[2,4]+m[5,5]+p_1p_4p_5=4375+0+35\times10\times20=11374$
$\}$
$=7125$

Constructing the optimal solution

```
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:** $((A_1(A_2A_3))((A_4A_5)A_6))$

Constructing the optimal solution

```
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:** $((A_1(A_2A_3))((A_4A_5)A_6))$

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6, 1..6]$ has been properly constructed.

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6, 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6, 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$s[1,6] = 3$$

Step 4...

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6, 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$s[1, 6] = 3 \qquad (A_1A_2A_3)(A_4A_5A_6)$$

Step 4...

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6, 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$s[1,6] = 3 \qquad (A_1A_2A_3)(A_4A_5A_6)$$
$$s[1,3] = 1$$

Step 4...

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6, 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$\begin{array}{ll} s[1,6] = 3 & (A_1A_2A_3)(A_4A_5A_6) \\ s[1,3] = 1 & (A_1(A_2A_3)) \end{array}$$

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6,\ 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$
\begin{array}{ll}
s[1,6] = 3 & (A_1A_2A_3)(A_4A_5A_6) \\
s[1,3] = 1 & (A_1(A_2A_3)) \\
s[4,6] = 5 &
\end{array}
$$

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6,\ 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$\begin{array}{ll} s[1,6]=3 & (A_1A_2A_3)(A_4A_5A_6) \\ s[1,3]=1 & (A_1(A_2A_3)) \\ s[4,6]=5 & ((A_4A_5)A_6) \end{array}$$

Step 4...

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6, 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$\begin{array}{ll} s[1,6]=3 & (A_1A_2A_3)(A_4A_5A_6) \\ s[1,3]=1 & (A_1(A_2A_3)) \\ s[4,6]=5 & ((A_4A_5)A_6) \end{array}$$

Hence the final multiplication sequence is

Example (Finding the Multiplication Sequence)

Consider $n = 6$. Assume array $s[1..6,\ 1..6]$ has been properly constructed. The multiplication sequence is recovered as follows.

$$\begin{array}{ll} s[1,6]=3 & (A_1A_2A_3)(A_4A_5A_6) \\ s[1,3]=1 & (A_1(A_2A_3)) \\ s[4,6]=5 & ((A_4A_5)A_6) \end{array}$$

Hence the final multiplication sequence is

$$(A_1(A_2A_3))((A_4A_5)A_6).$$

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for i = 1 **to** n **do** $m[i, i]$ =

The Dynamic Programming Algorithm

Matrix-Chain(*p*, *n*):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;

;

for l =

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for i = 1 **to** n **do** $m[i, i] = 0$;
;
for l = 2 **to**

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for i = 1 **to** n **do** $m[i, i] = 0$;
;
for l = 2 **to** n **do**
    for i = 1 **to**

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;
;
for $l = 2$ **to** n **do**
    for $i = 1$ **to** $n - l + 1$ **do**
        $j =$

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;

;

for $l = 2$ **to** n **do**

  for $i = 1$ **to** $n - l + 1$ **do**

    $j = i + l - 1$;

    $m[i, j] =$

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;
;
for $l = 2$ **to** n **do**
  for $i = 1$ **to** $n - l + 1$ **do**
    $j = i + l - 1$;
    $m[i, j] = \infty$;
    for k =

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
    for i = 1 to n − l + 1 do
        j = i + l − 1;
        m[i,j] = ∞;
        for k = i to
```

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;

;

for $l = 2$ **to** n **do**

for $i = 1$ **to** $n - l + 1$ **do**

$j = i + l - 1$;

$m[i, j] = \infty$;

for $k = i$ **to** $j - 1$ **do**

$q =$

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;

;

for $l = 2$ **to** n **do**

for $i = 1$ **to** $n - l + 1$ **do**

$j = i + l - 1$;

$m[i, j] = \infty$;

for $k = i$ **to** $j - 1$ **do**

$q = m[i, k] +$

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
        for i = 1 to n − l + 1 do
            j = i + l − 1;
            m[i,j] = ∞;
                for k = i to j − 1 do
                    q = m[i,k] + m[k + 1,j] +
```

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
    for i = 1 to n − l + 1 do
        j = i + l − 1;
        m[i,j] = ∞;
        for k = i to j − 1 do
            q = m[i,k] + m[k+1,j] + p[i − 1] *
```

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;

;

for $l = 2$ **to** n **do**

    for $i = 1$ **to** $n - l + 1$ **do**

        $j = i + l - 1$;

        $m[i, j] = \infty$;

        for $k = i$ **to** $j - 1$ **do**

            $q = m[i, k] + m[k + 1, j] + p[i - 1] * p[k] *$

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

for $i = 1$ **to** n **do** $m[i, i] = 0$;
;
for $l = 2$ **to** n **do**
    for $i = 1$ **to** $n - l + 1$ **do**
        $j = i + l - 1$;
        $m[i, j] = \infty$;
        for $k = i$ **to** $j - 1$ **do**
            $q = m[i, k] + m[k + 1, j] + p[i - 1] * p[k] * p[j]$;

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
    for i = 1 to n − l + 1 do
        j = i + l − 1;
        m[i,j] = ∞;
        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] =
```

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
    for i = 1 to n − l + 1 do
        j = i + l − 1;
        m[i,j] = ∞;
        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] =
```

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
        for i = 1 to n − l + 1 do
            j = i + l − 1;
            m[i,j] = ∞;
                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;
                    end
                end
        end
end  return
```

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
    for i = 1 to n − l + 1 do
        j = i + l − 1;
        m[i,j] = ∞;
        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;
            end
        end
    end
end
return m and s;(Optimum in
```

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
        for i = 1 to n − l + 1 do
            j = i + l − 1;
            m[i,j] = ∞;
                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;
                    end
            end
    end
end
return m and s;(Optimum in m[1,n])
```

Complexity: The loops are nested three levels deep.

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do     / l is the chain length
        for i = 1 to n − l + 1 do
            j = i + l − 1;
            m[i,j] = ∞ ;
                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;
                    end
            end
        end
end
return m and s;(Optimum in m[1,n])
```

Complexity: The loops are nested three levels deep. Each loop index takes on $\leq n$ values.

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
        for i = 1 to n − l + 1 do
            j = i + l − 1;
            m[i,j] = ∞;
                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;
                    end
            end
    end
end
return m and s;(Optimum in m[1,n])
```

Complexity: The loops are nested three levels deep. Each loop index takes on

$\leq n$ values. Hence the time complexity is $O(n^3$.

The Dynamic Programming Algorithm

Matrix-Chain(p, n):// l is length ofsub-chain

```
for i = 1 to n do m[i,i] = 0;
;
for l = 2 to n do
        for i = 1 to n − l + 1 do
            j = i + l − 1;
            m[i,j] = ∞;
              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;
                end
            end
        end
end
return m and s;(Optimum in m[1,n])
```

Complexity: The loops are nested three levels deep. Each loop index takes on $\leq n$ values. Hence thetime complexityis $O(n^3)$. Space complexityis $\Theta(n^2)$.

Reference

Introduction to Algorithms

Thomas H. Cormen

Chapter # 15