Design and Analysis of Algorithms
Supplemental

Si Wu

School of CSE, SCUT

cswusi@scut.edu.cn

TA: 1684350406@qq.com

Outline

- **Longest Common Substring**
- **Chain Matrix Multiplication**

Longest Common Substring

A slightly different problem (longest common subsequence) with a similar solution

Given two strings $X = x_1x_2...x_m$ and $Y = y_1y_2...y_n$, find their longest common substring Z , i.e., a largest k for which there are indices i and j with $x_ix_{i+1}...x_{i+k-1} = y_jy_{j+1}...y_{j+k-1}$.

For example:

X : DEADBEEF

Y : EATBEEF

Z : BEEF //pick the longest contiguous substring

Show how to do this by dynamic programming.

LCS Solution

Step 1: Space of Subproblems

For $1 \leq i \leq m$, and $1 \leq j \leq n$,

- Define $d_{i,j}$ to be the length of the longest common substring ending at x_i and y_j . (Does this work?)
- Let D be the $m \times n$ matrix $[d_{i,j}]$.
  - How does D provide answer?

LCS Solution

Step 2: Recursive Formulation

Case 1: If $x_i = y_j$, then $z_k = x_i = y_j$ and

z_{k-1} is a LCS of X and Y ending at x_{i-1} and y_{j-1}

Case 2: If $x_i \neq y_j$, then there cannot be a common substring ending at x_i and y_j!

$$d_{i,j} = \begin{cases} d_{i-1,j-1} + 1 & if\ x_i = y_j \\ 0 & if\ x_i \neq y_j \end{cases}$$

Finally, we can find length of longest common substring by finding maximum $d_{i,j}$ among all possible ending position i and j.

$$LCSSubString(X, Y) = \max\{d_{i,j}\}$$

LCS Solution

Step 3: Bottom-up Computation

Similar to *Longest Common Subsequence* we set the first row and column of the matrix $d[0, j]$ and $d[i, 0]$ to be 0.

Calculate $d[1, j]$ for $j = 1, 2, \dots, \mathrm{n}$
Then, the $d[2, j]$ for $j = 1, 2, \dots, \mathrm{n}$
Then, the $d[3, j]$ for $j = 1, 2, \dots, \mathrm{n}$

etc., filling the matrix row by row and left to right.

For this problem we do not need to create another $m \times n$ matrix for storing arrows. Instead, we use l_{max} and p_{max} to store the largest length of common substring and its i position respectively. This suffices to reconstruct the solution.

LCS Solution

LONGEST-COMMON-SUBSTRING(*X*, *Y*)

$m \leftarrow length(X)$; $n \leftarrow length(Y)$;

$l_{max} \leftarrow 0$; $p_{max} \leftarrow 0$;

$\boldsymbol{for}$ i $\leftarrow 0$ $\boldsymbol{to}$ m // initialization

$\quad d[i, 0] \leftarrow 0$;

$\boldsymbol{for}$ $j \leftarrow 0$ $\boldsymbol{to}$ n

$\quad d[0, j] \leftarrow 0$;

$\boldsymbol{for}$ i $\leftarrow 1$ $\boldsymbol{to}$ m // dynamic programming

$\quad \boldsymbol{for}$ $j \leftarrow 1$ $\boldsymbol{to}$ n

$\quad\quad \boldsymbol{if}(x_i \neq y_j)$

$\quad\quad\quad d[i, j] \leftarrow 0$;

$\quad\quad$ ***else***

$\quad\quad\quad d[i, j] \leftarrow d[i-1, j-1] + 1$;

$\quad\quad\quad \boldsymbol{if}(d[i, j] > l_{max})$

$\quad\quad\quad\quad l_{max} \leftarrow d[i, j]$; $p_{max} \leftarrow i$;

$\quad\quad\quad$ ……

$\boldsymbol{return}$ l_{max}, p_{max};

LCS Example

- Take the two strings: *X* = “EL GATO” and *Y* =“GATER”.
- We’ll fill in the following table *D*:

$$d_{i,j} = \begin{cases} d_{i-1,j-1} + 1 & if\ x_i = y_j \\ 0 & if\ x_i \neq y_j \end{cases}$$

LCS Example

- Take the two strings: *X* = “EL GATO” and *Y* =“GATER”.
- We’ll fill in the following table *D*:

$$d_{i,j} = \begin{cases} d_{i-1,j-1} + 1 & if\ x_i = y_j \\ 0 & if\ x_i \neq y_j \end{cases}$$

When filling *D*, we only look if the two letters in the strings are equal and if they are we add one to the element to the left and up.

	_	E	L	G	A	T	O
_	0	0	0	0	0	0	0
G	0	0	0	1	0	0	0
A	0	0	0	0	2	0	0
T	0	0	0	0	0	3	0
E	0	1	0	0	0	0	0
R	0	0	0	0	0	0	0

Review of Matrix Multiplication

- **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.

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.

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

- 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.

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.,

$$A_1A_2 \neq A_2A_1$$

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:

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

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

The Chain Matrix Multiplication Problem

- Definition (Chain matrix multiplication problem) :

  Given dimensions p_0,p_1, . . . ,p_n, corresponding to matrix sequence $A_1A_2 \dots A_n$ 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 $A_1A_2 \dots A_n$.

- Question: Is there a better approach?

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 $A_{i..j} = A_i A_{i+1} \ldots A_j$.

  Note that $A_{i..j}$ is a $p_{i-1} \times p_j$ matrix.

- How can we solve larger problems using subproblem solutions?

Relationships among Subproblems

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

- 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.

Relationships among Subproblems

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.

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

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

Developing a Dynamic Programming Algorithm

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

- Recurrence:

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 $A_{i..j}$:

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

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

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

$\ldots$

$m[1, n-1], m[2, n]$

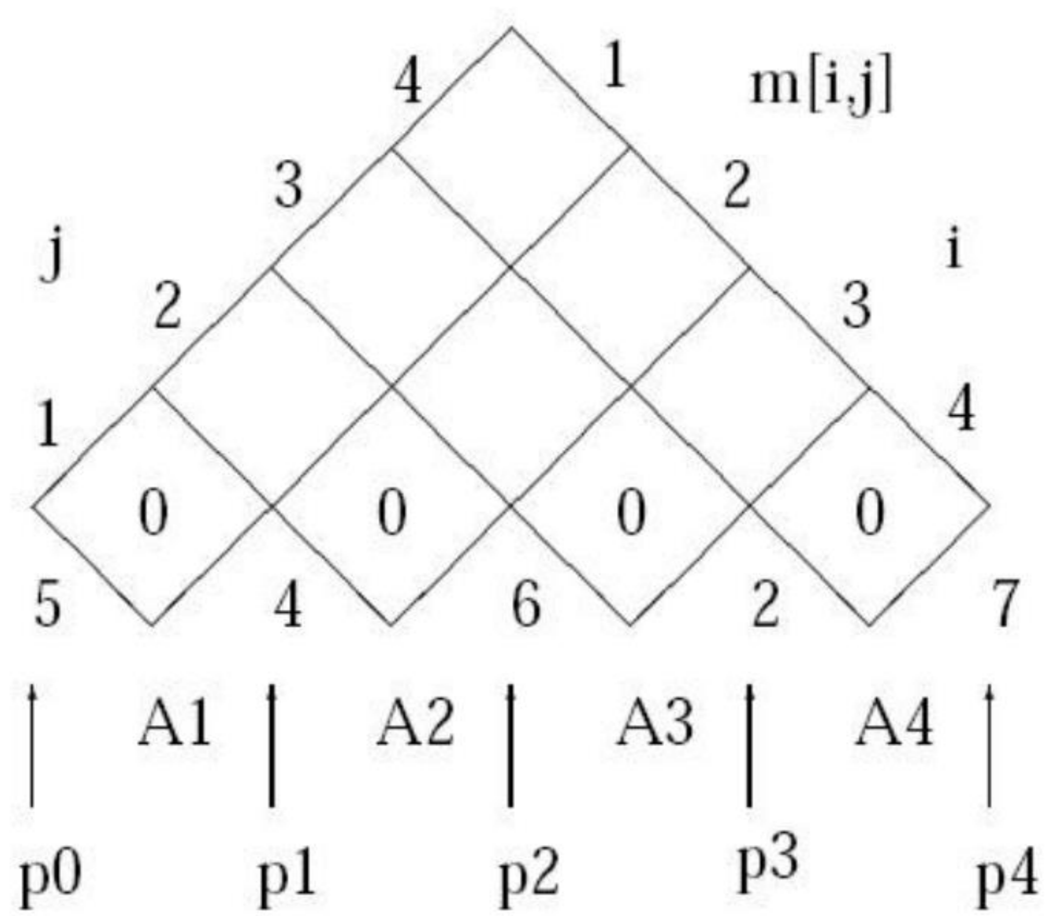
$m[1, n]$

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 $m[1, 4]$.

  S0: Initialization

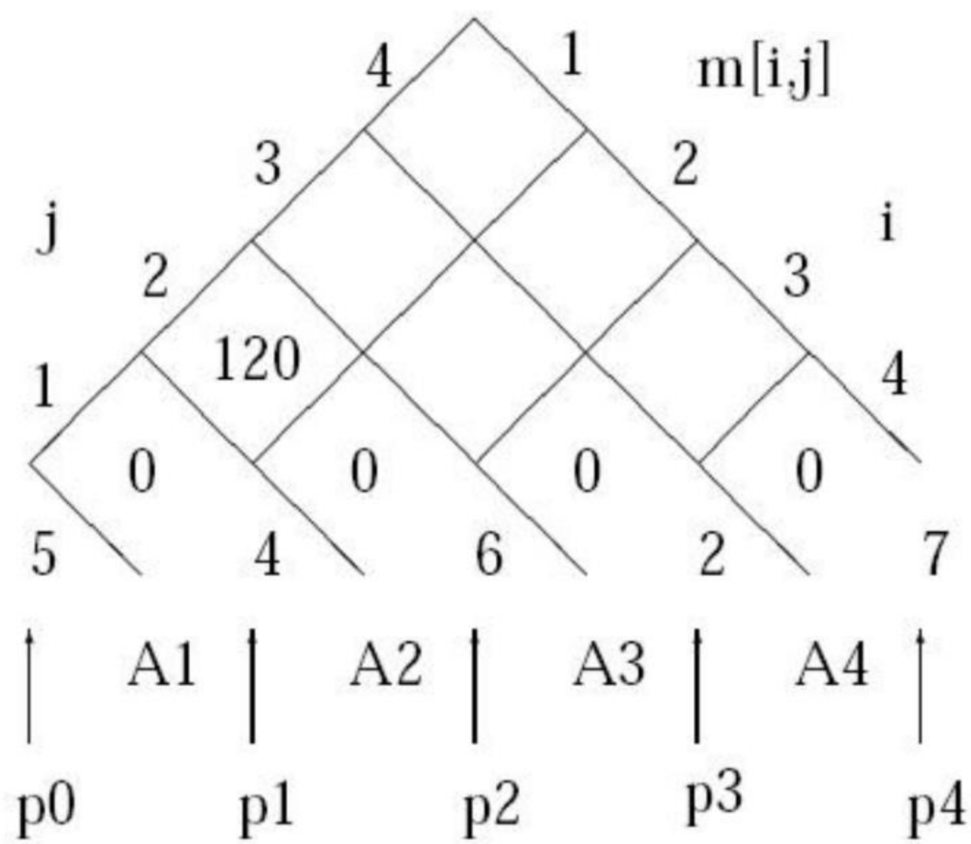


Example – Continued

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

By definition

$$\begin{aligned} m[1,2] &= \min_{1\le k<2}(m[1,k] + m[k+1,2] + p_0 p_k p_2 \\ &= m[1,1] + m[2,2] + p_0 p_1 p_2 = 120 \end{aligned}$$

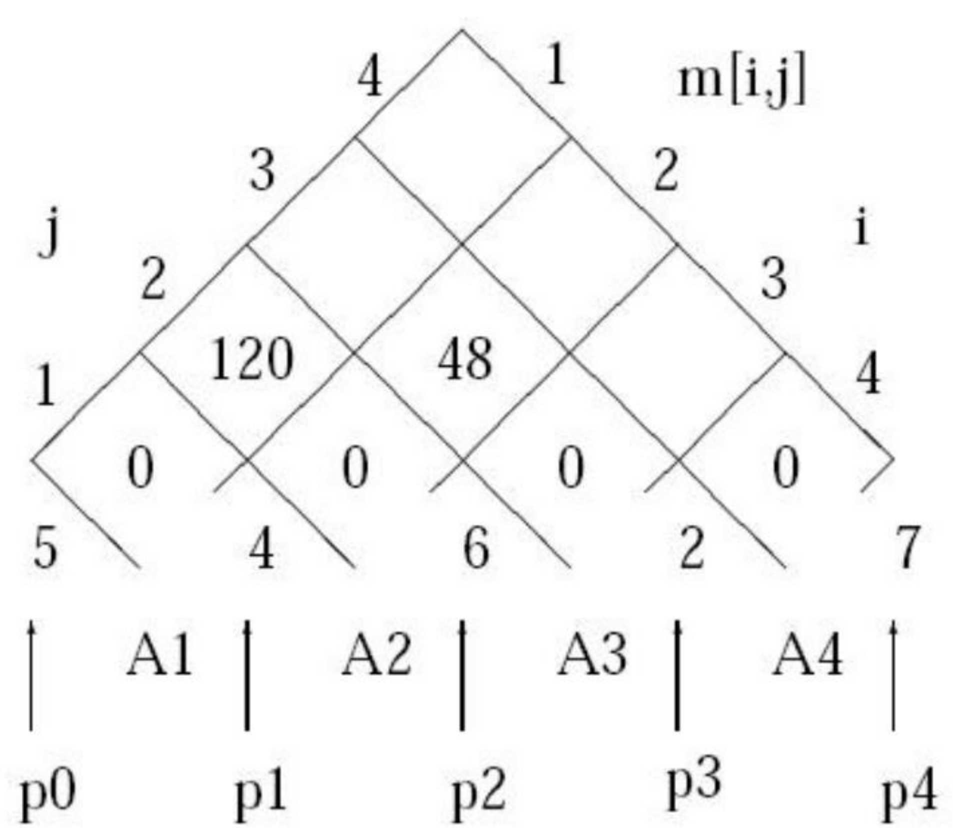


Example – Continued

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

By definition

$$
\begin{aligned}
m[2,3] &= \min_{2 \le k < 3} (m[2,k] + m[k+1,3] + p_1 p_k p_3 \\
&= m[2,2] + m[3,3] + p_1 p_2 p_3 = 48
\end{aligned}
$$

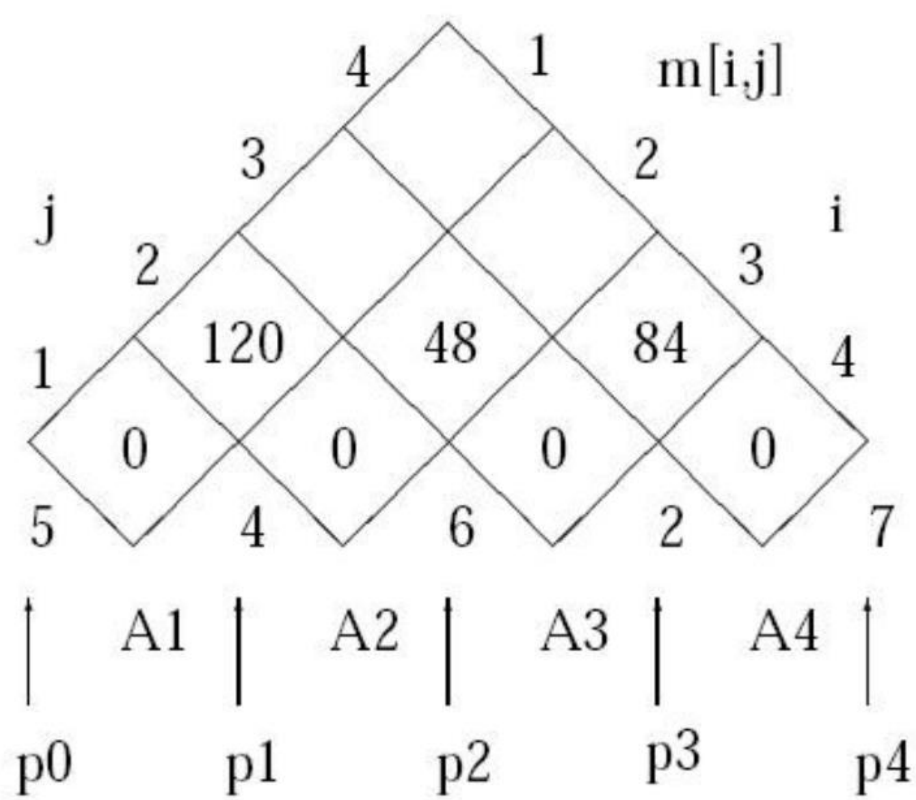


Example – Continued

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

By definition

$$
\begin{aligned}
m[3,4] &= \min_{3 \le k < 4} (m[3,k] + m[k+1,4] + p_2 p_k p_4 \\
&= m[3,3] + m[4,4] + p_2 p_3 p_4 = 84
\end{aligned}
$$



Example – Continued

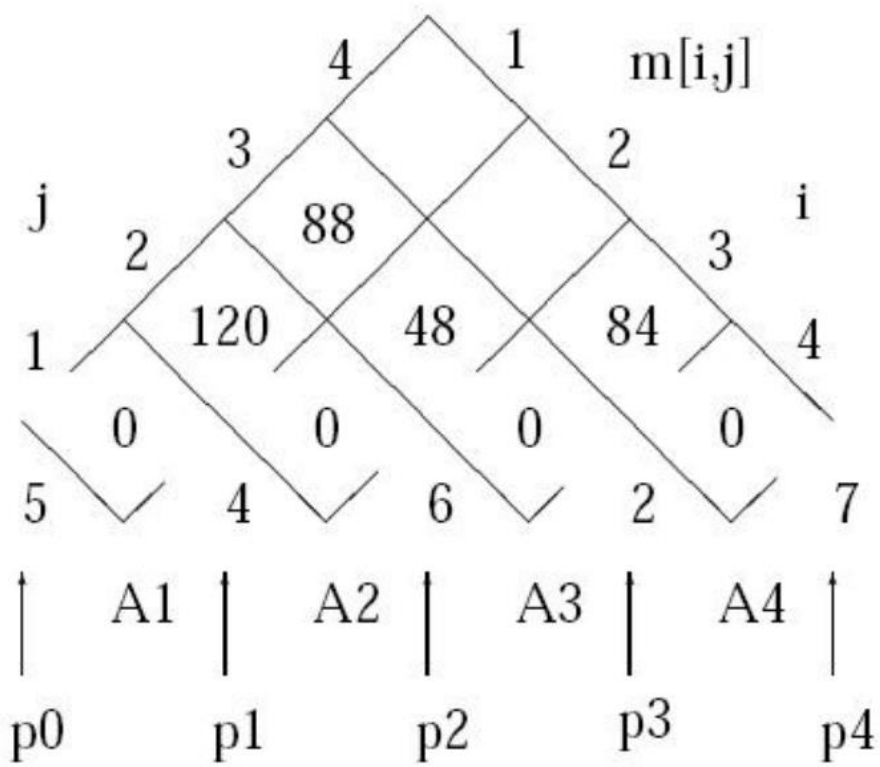
- Step 4: Computing $m[1,3]$

By definition

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

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

$$= 88$$

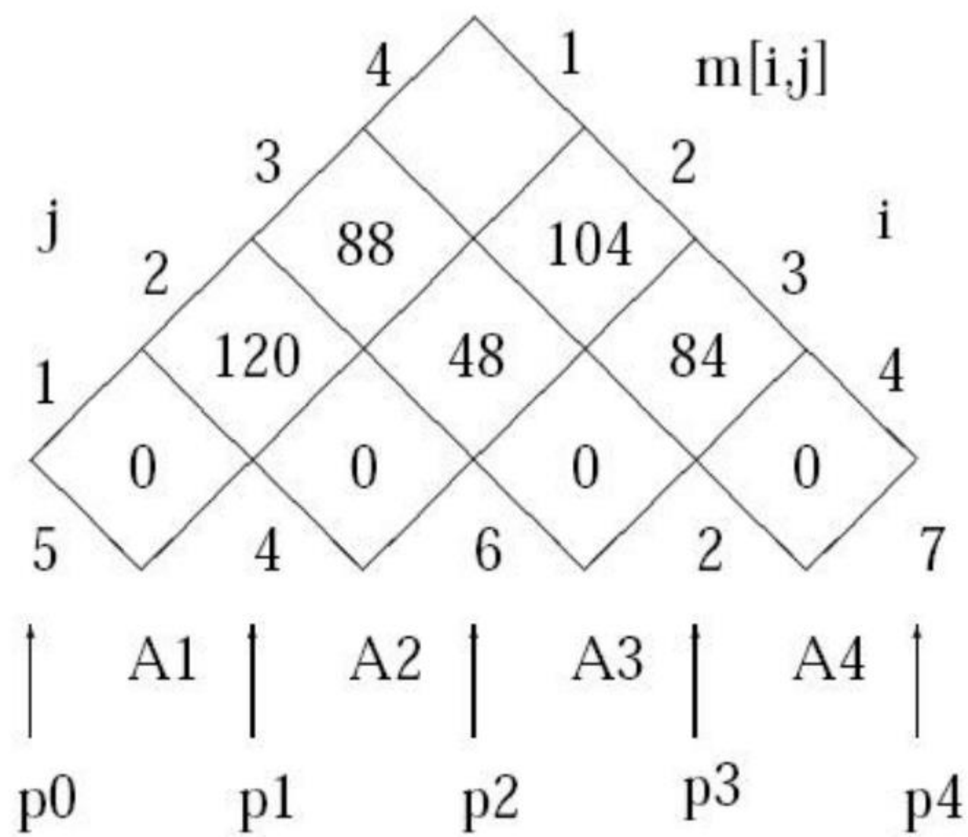


Example – Continued

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

By definition

$$
\begin{aligned}
m[2,4] &= \min_{2\le k<4}(m[2,k]+m[k+1,4]+p_1p_kp_4 \\
&= \min\begin{Bmatrix} m[2,2]+m[3,4]+p_1p_2p_4 \\ m[2,3]+m[4,4]+p_1p_3p_4 \end{Bmatrix} \\
&= 104
\end{aligned}
$$

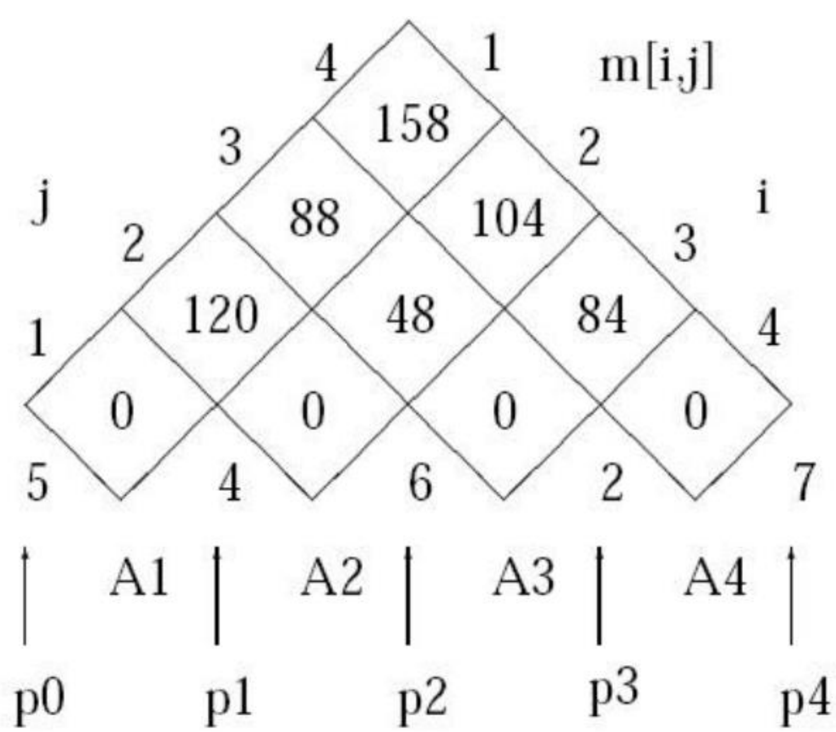


Example – Continued

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

By definition

$$
\begin{aligned}
m[1,4] \quad &= \quad \min_{1 \le k < 4} (m[1,k] + m[k+1,4] + p_0 p_k p_4 \\
&= \quad \min \begin{Bmatrix} m[1,1] + m[2,4] + p_0 p_1 p_4 \\ m[1,2] + m[3,4] + p_0 p_2 p_4 \\ m[1,3] + m[4,4] + p_0 p_3 p_4 \end{Bmatrix} \\
&= \quad 158
\end{aligned}
$$



The Dynamic Programming Algorithm

Matrix-Chain(p, n): // l is length of sub-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**

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

      if $q < m[i, j]$ **then**

        $m[i, j] = q$;

        $s[i, j] = k$;

      end

    end

  end

end

return m and s; (Optimum in $m[1, n]$)