Matrix-chain Multiplication

Kuan-Yu Chen (陳冠宇)

2019/04/17 @ TR-310-1, NTUST

Review

- Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems

- We typically apply dynamic programming to ***optimization problems***
  - Such problems can have many possible solutions
  - Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value

Matrix-chain Multiplication.

- The next example of dynamic programming is an algorithm that solves the problem of matrix-chain multiplication
  - We are given a sequence (chain) $\{A_1, A_2, \cdots, A_n\}$ of n matrices to be multiplied
  - Fully Parenthesized
    - For example, if the chain of matrices is $\{A_1, A_2, A_3, A_4\}$, then we can fully parenthesize the product $A_1 A_2 A_3 A_4$ in five distinct ways
    - $\left(A_1\left(A_2(A_3 A_4)\right)\right)$
    - $\left(A_1\left((A_2 A_3) A_4\right)\right)$
    - $\left((A_1 A_2)(A_3 A_4)\right)$
    - $\left(\left(A_1(A_2 A_3)\right) A_4\right)$
    - $\left(\left((A_1 A_2) A_3\right) A_4\right)$
  - Matrix multiplication is associative, and so all parenthesizations yield the same result

Matrix-chain Multiplication..

- Actually, how we parenthesize a chain of matrices can have a dramatic impact on the cost of evaluating the product
  - Given a chain $\{A_1, A_2, A_3\}$ of three matrices
  - Suppose that the dimensions of the matrices are 10×100, 100×5, and 5×50, respectively
  - If we multiply according to the parenthesization $\left((A_1 A_2) A_3\right)$
    - $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$ multiplications
  - If we multiply according to the parenthesization $(A_1(A_2 A_3))$
    - $100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$ multiplications

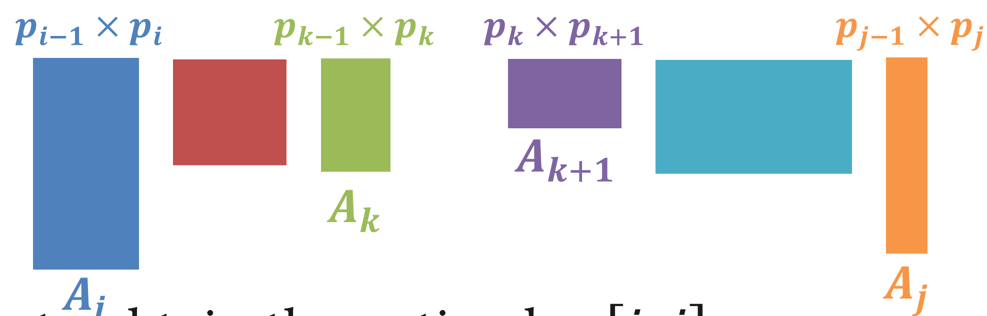
```
MATRIX-MULTIPLY(A, B)
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

Matrix-chain Multiplication…

- We state the ***matrix-chain multiplication problem*** as follows: given a chain $\{A_1, A_2, \cdots, A_n\}$ of n matrices, where for $i = 1, \cdots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $\{A_1, A_2, \cdots, A_n\}$ in a way that minimizes the number of scalar multiplications
  - Note that 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**

DP for Matrix-chain Multiplication.

- Let us adopt the notation $A_{i\dots j}$, where $i \leq j$, for the matrix that results from evaluating the product $A_{i\dots j} = A_i A_{i+1} \cdots A_j$
  - Let $m[i,j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i\dots j}$
  - It is easy to understand that $m[i,i] = 0$
    - Only a matrix A_i
    - No scalar multiplications are necessary to compute the product



  - On the other hand, in order to obtain the optimal $m[i,j]$, we should split $A_{i\dots j}$ between A_k and A_{k+1}

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$$

  - A general form is

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

DP for Matrix-chain Multiplication..

- For the original problem, we can split the whole product between A_k and A_{k+1} for some integer k

$$A_{1\ldots n} = A_1 A_2 \cdots A_n = A_1 \cdots A_k A_{k+1} \cdots A_n = A_{1\ldots k} A_{k+1\ldots n}$$

  – The lowest cost way to compute $A_{1\ldots n}$ would thus be $m[1, n]$

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

DP for Matrix-chain Multiplication…

- Let's consider a bottom-up approach Matrix-Chain-Order
  - Since the matrix A_i has dimensions $p_{i-1} \times p_i$
  - The input is thus a sequence $p = \{p_0, p_1, \cdots, p_n\}$
    - $p.length = n + 1$
  - An auxiliary table $m[1 \ldots n, 1 \ldots n]$ is used to store the $m[i, j]$ costs
  - An auxiliary table $s[1 \ldots n-1, 2 \ldots n]$ is used to store which index of k achieved the optimal cost in computing $m[i, j]$

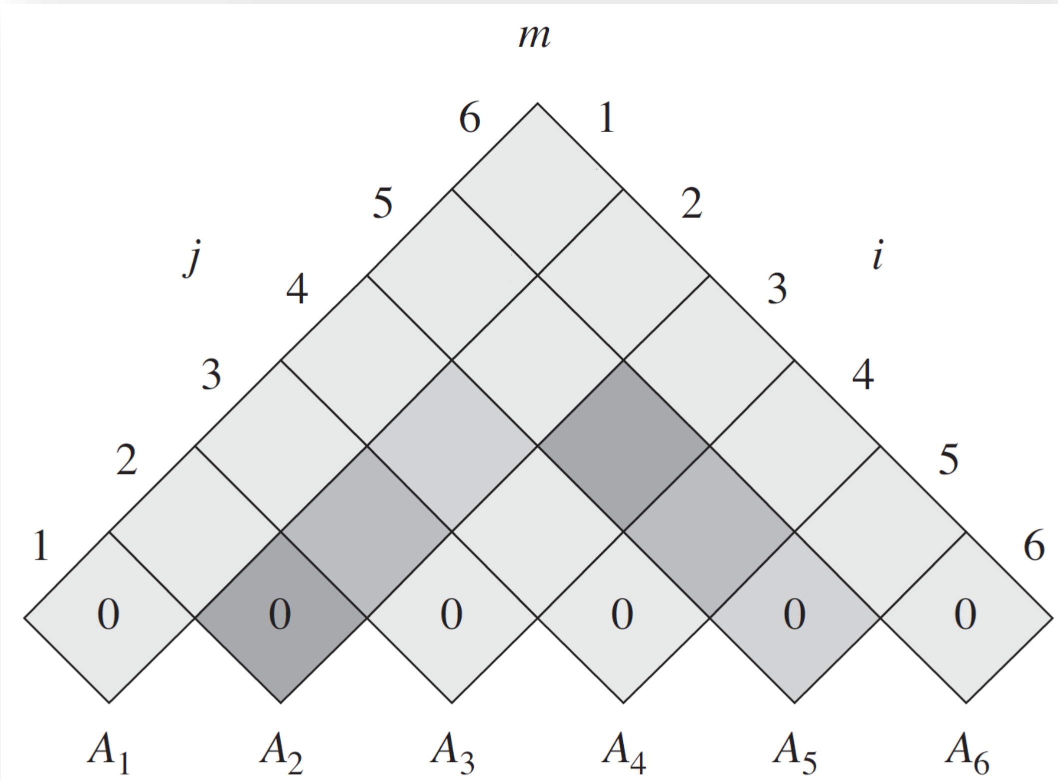
```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

DP for Matrix-chain Multiplication….

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                  // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example.

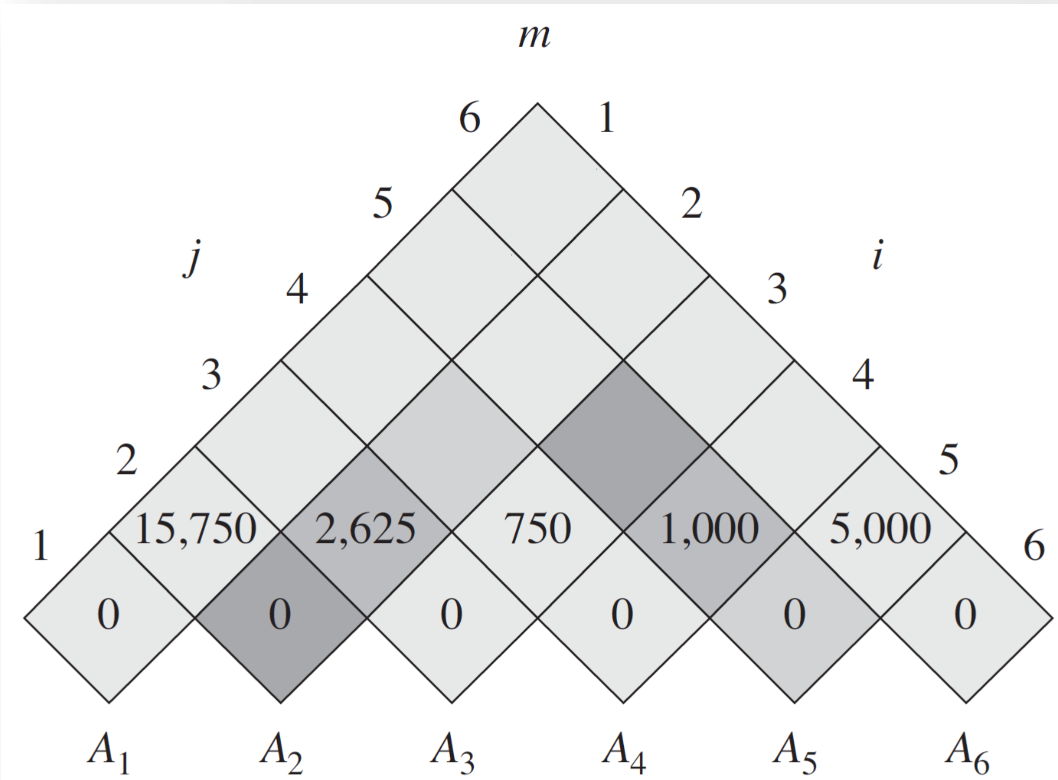
matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25




```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example..

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

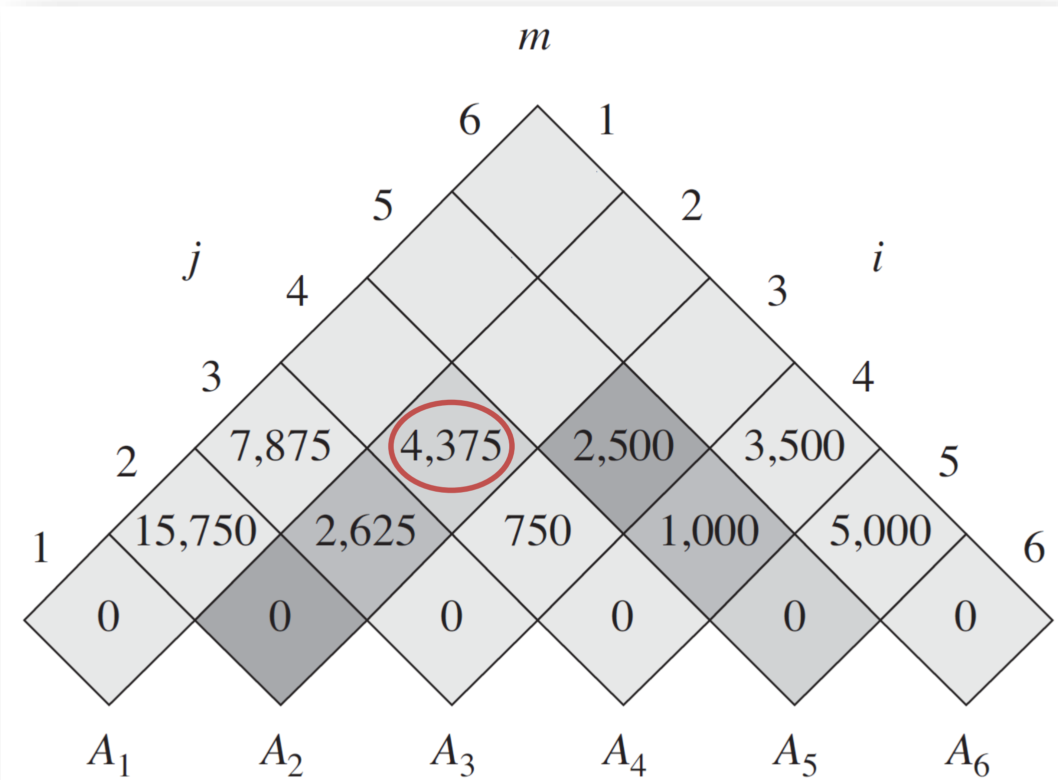



```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n          // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example...

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

$$m[2,4] = \begin{cases} m[2,3] + m[4,4] + 35 \times 5 \times 10 = 2625 + 0 + 1750 = 4375 \\ m[2,2] + m[3,4] + 35 \times 15 \times 10 = 0 + 750 + 5250 = 6000 \end{cases}$$



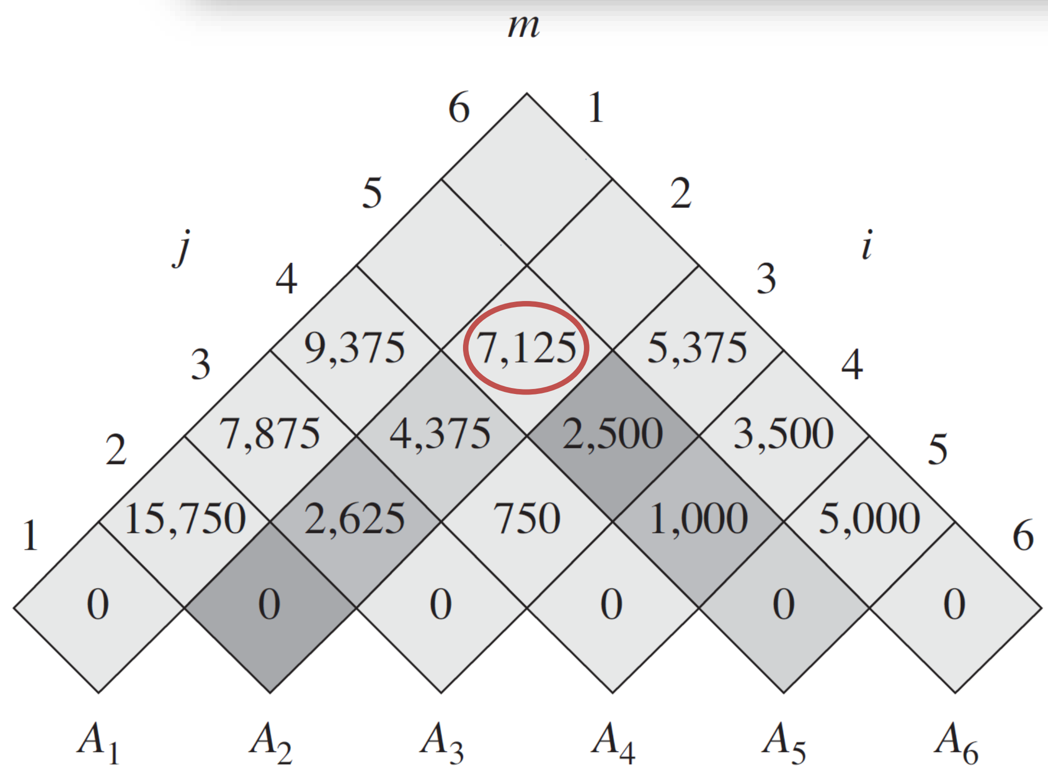
MATRIX-CHAIN-ORDER(*p*)

```
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example....

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

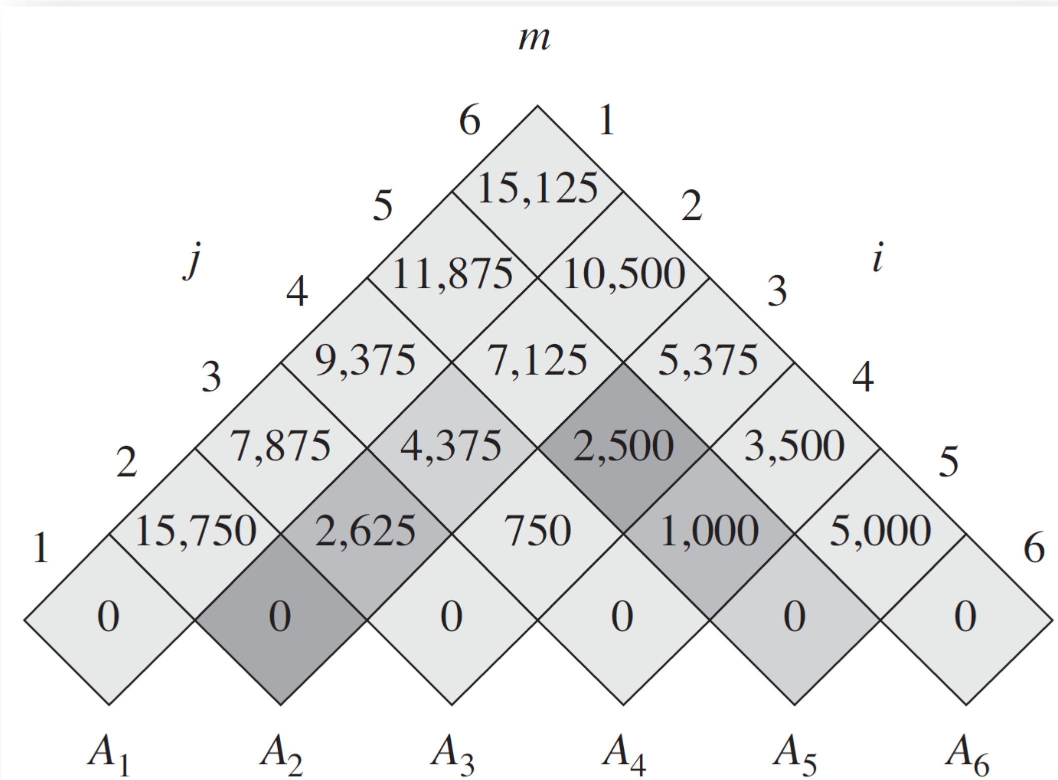
$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000 \,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125 \,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$




```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n              // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example.....

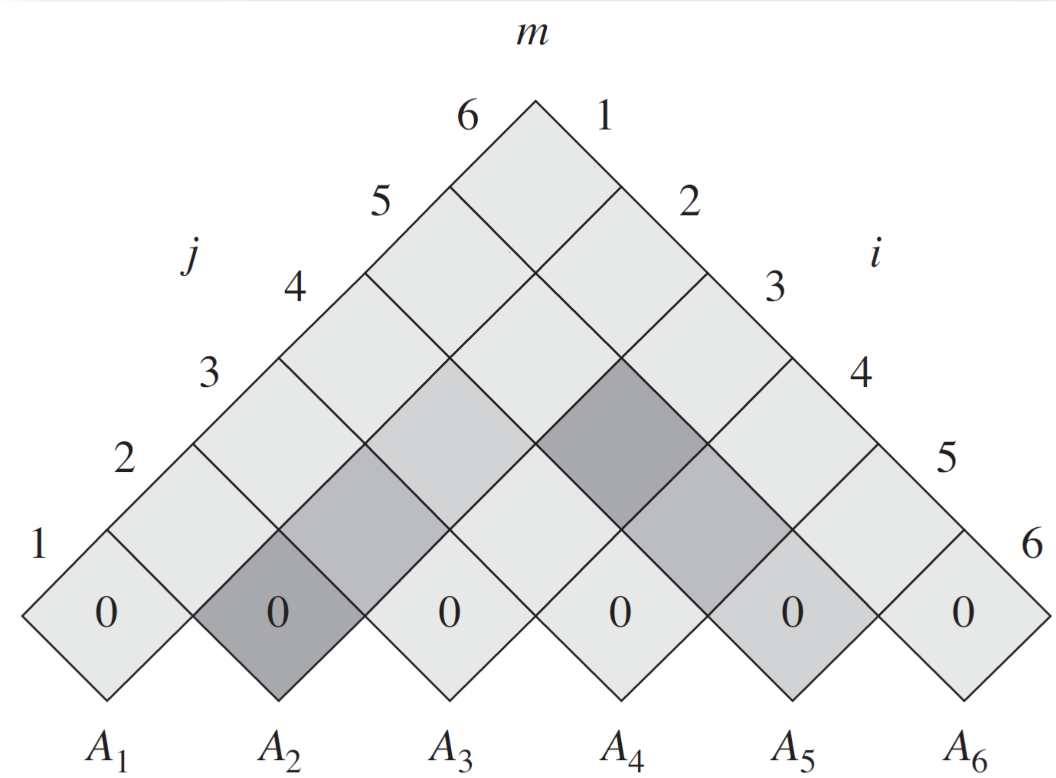
matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

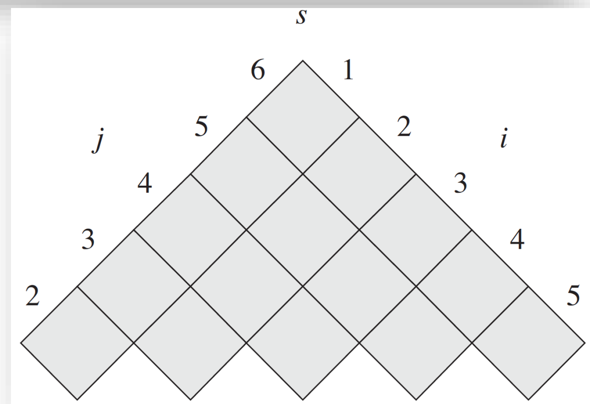



```
MATRIX-CHAIN-ORDER(p)
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i,i] = 0
for l = 2 to n            // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i,j] = ∞
        for k = i to j - 1
            q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
            if q < m[i,j]
                m[i,j] = q
                s[i,j] = k
return m and s
```

Example……

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

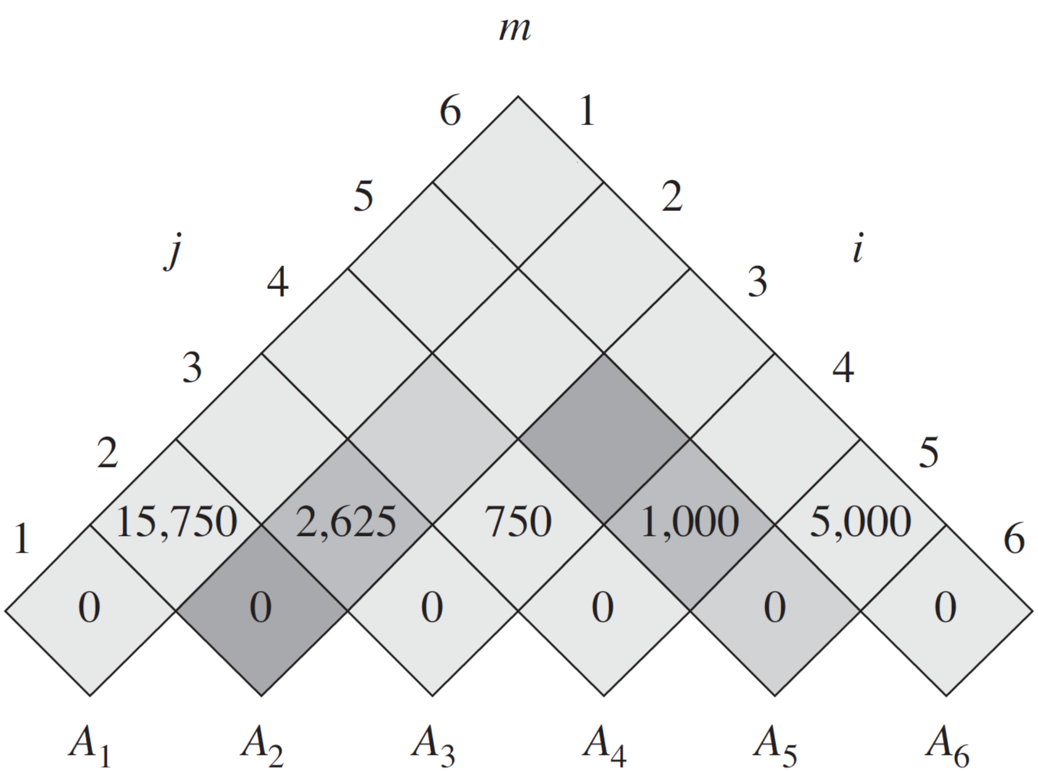


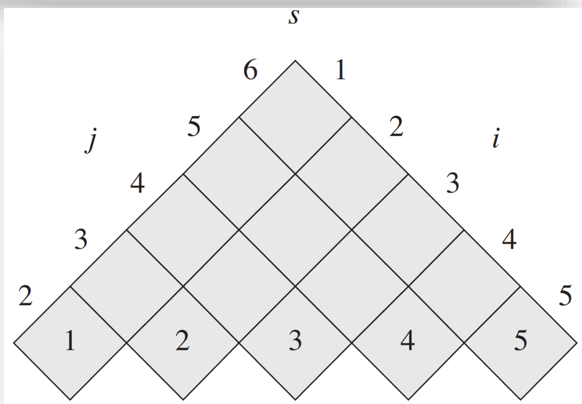



```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                  // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example…….

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25





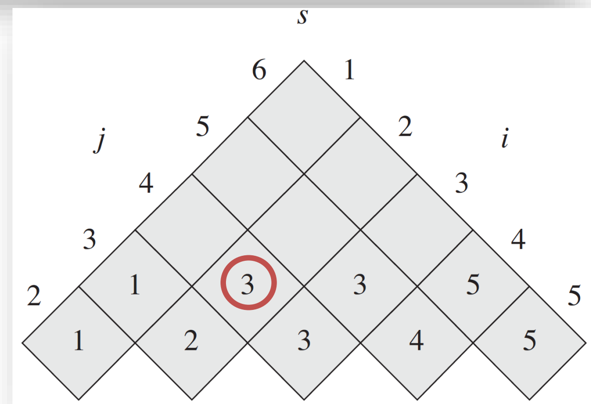
MATRIX-CHAIN-ORDER(p)

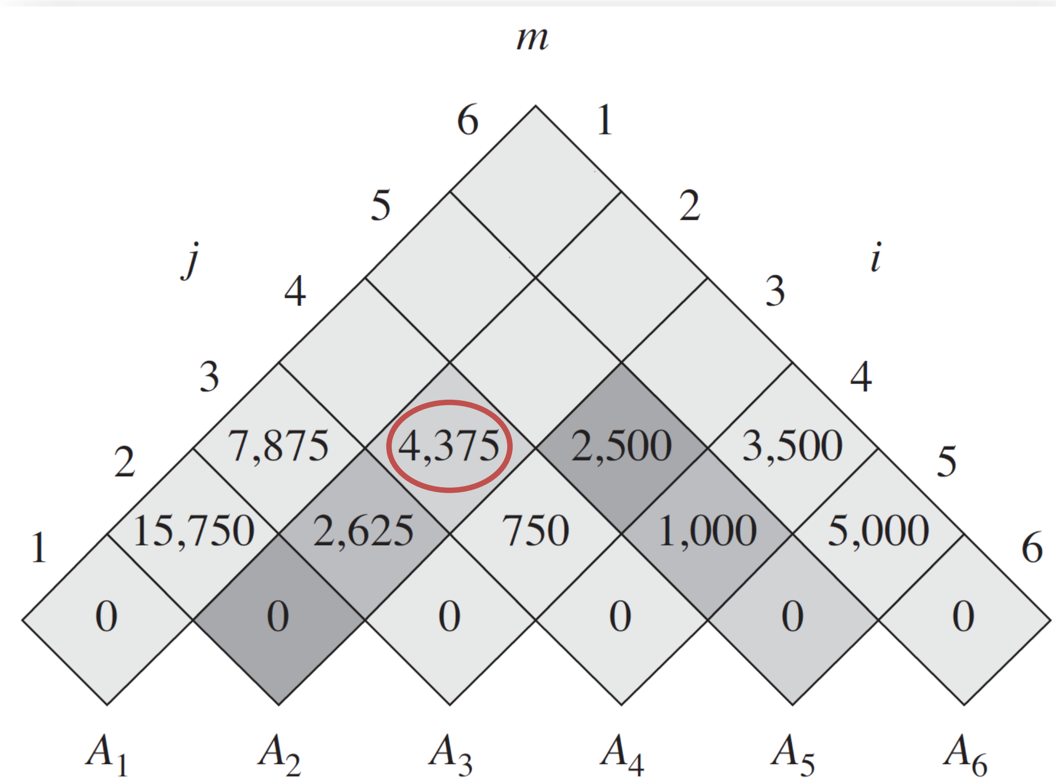
```
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                  // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example........

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

$$m[2,4] = \begin{cases} m[2,3] + m[4,4] + 35 \times 5 \times 10 = 4375 \\ m[2,2] + m[3,4] + 35 \times 15 \times 10 = 6000 \end{cases}$$





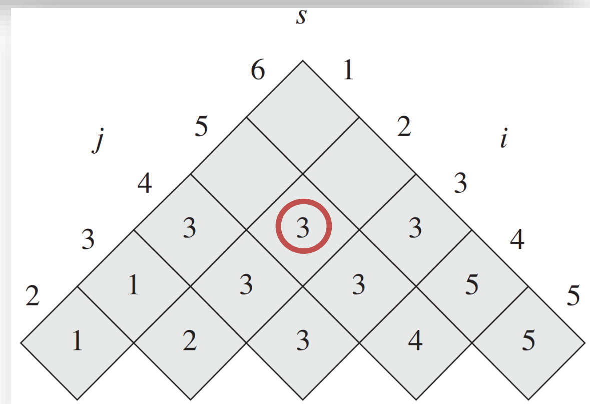
MATRIX-CHAIN-ORDER(p)

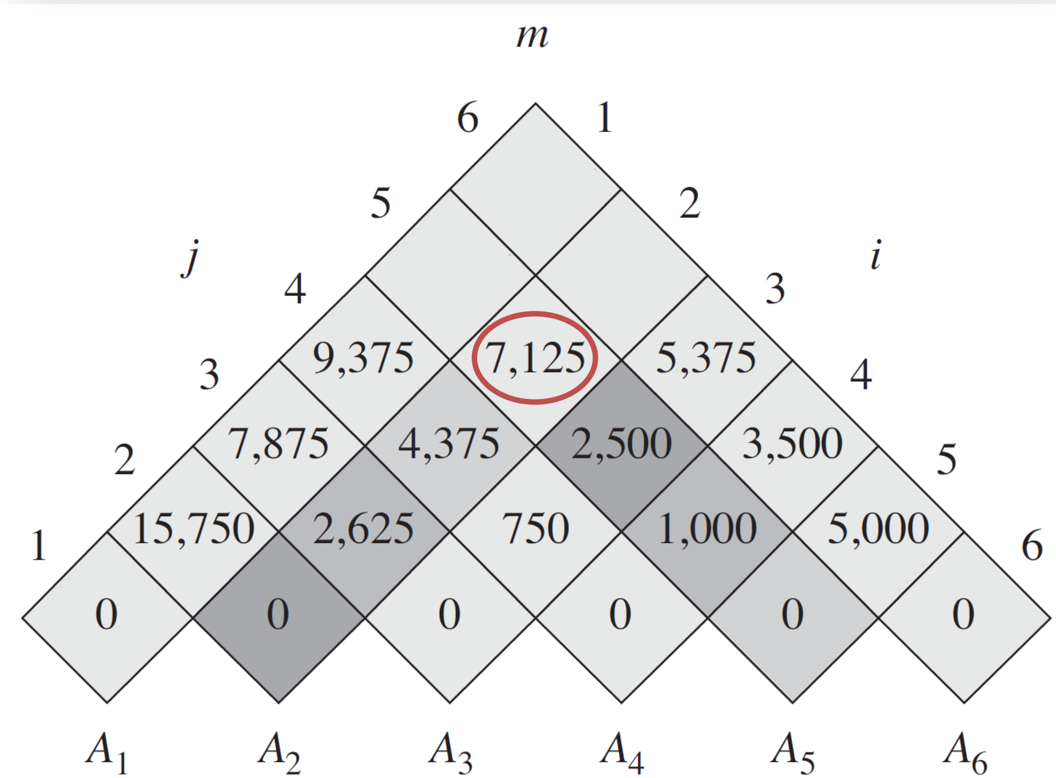
```
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n              // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example………

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$



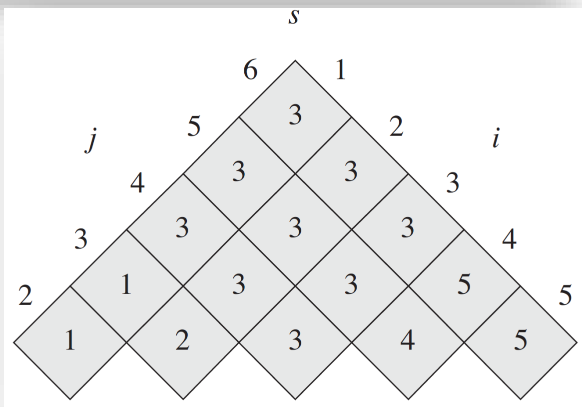


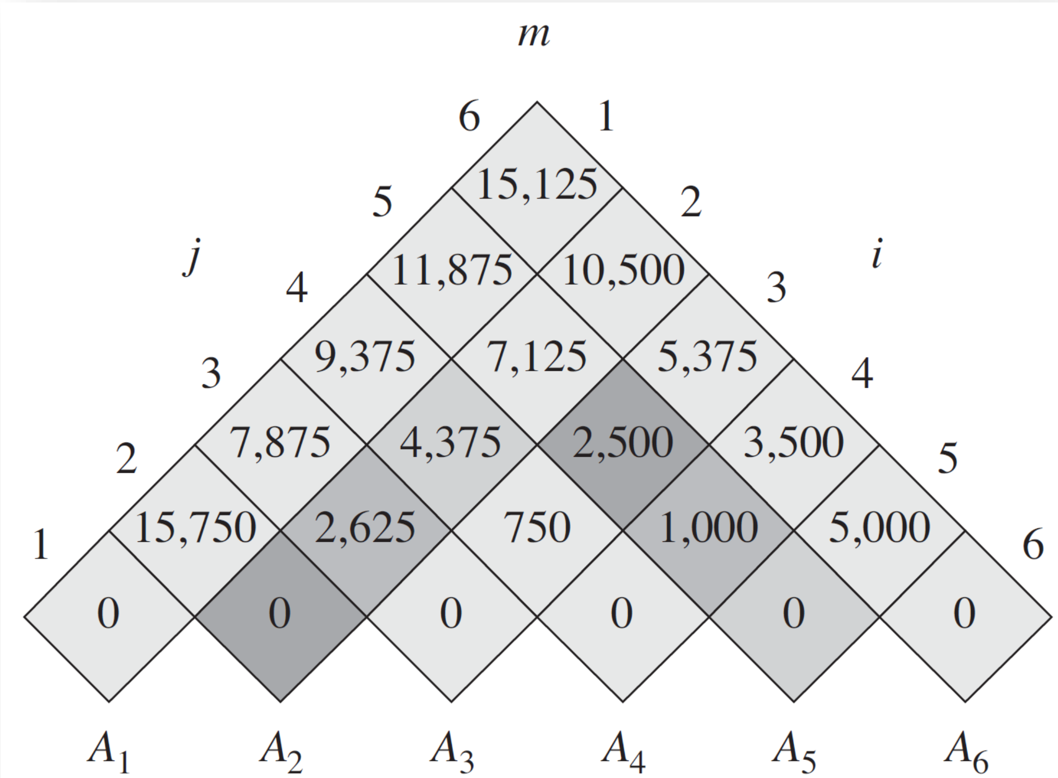
MATRIX-CHAIN-ORDER(p)

```
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n            // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Example.........

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25





MATRIX-CHAIN-ORDER(p)

```
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i,i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i,j] = ∞
        for k = i to j − 1
            q = m[i,k] + m[k+1,j] + p_{i−1} p_k p_j
            if q < m[i,j]
                m[i,j] = q
                s[i,j] = k
return m and s
```

DP for Matrix-chain Multiplication…..

- Although Matrix-Chain-Order determines the optimal number of scalar multiplications needed to compute a matrix-chain product, it does not directly show how to multiply the matrices

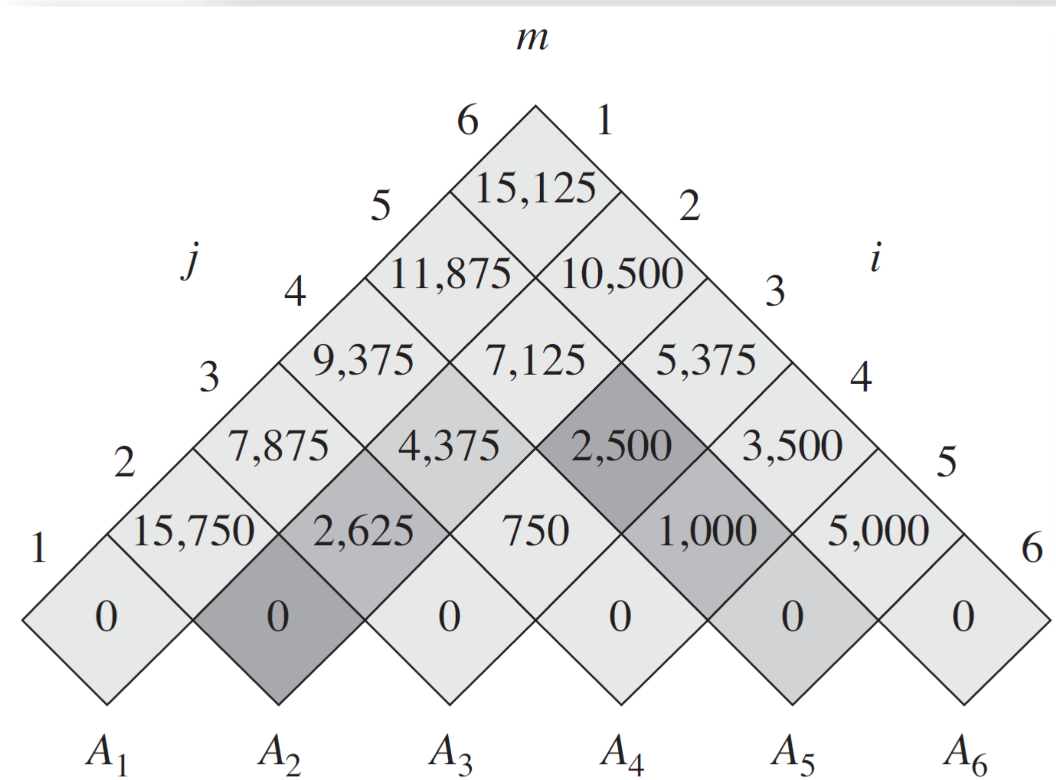
- The table $s[1 \ldots n-1, 2 \ldots n]$ gives us the information
  - Each entry $s[i, j]$ records a value of k such that an optimal parenthesization of $A_i A_{i+1} \cdots A_j$ splits the product between A_k and A_{k+1}
    - $s[i, j] = k$

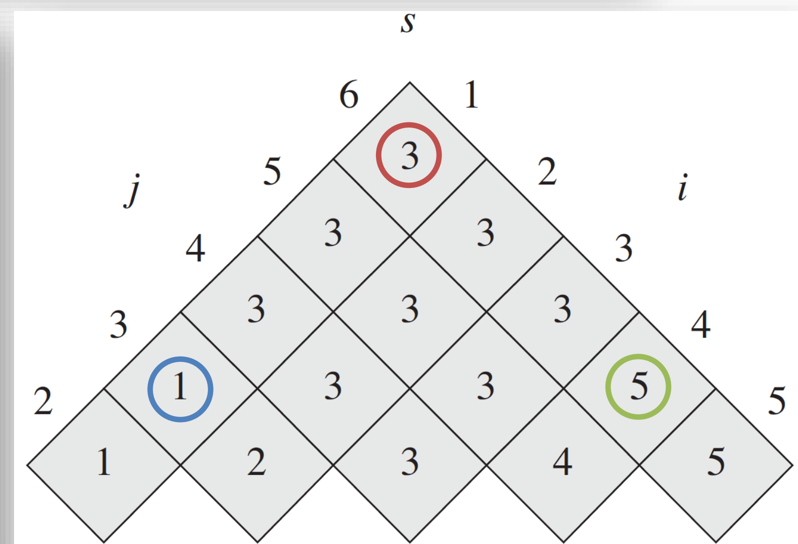
```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

Example

- The optimal parenthesization is $\big((A_1(A_2A_3))((A_4A_5)A_6)\big)$

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25





Questions?

kychen@mail.ntust.edu.tw