Dynamic Programming

Dynamic Programming (DP)

- Like divide-and-conquer, solve problem by combining the solutions to sub-problems.
- Differences between divide-and-conquer and DP:
  - Independent sub-problems, solve sub-problems independently and recursively, (so same sub(sub)problems solved repeatedly)
  - Sub-problems are dependent, i.e., sub-problems share sub-sub-problems, every sub(sub)problem solved just once, solutions to sub(sub)problems are stored in a table and used for solving higher level sub-problems.

Application domain of DP

- Optimization problem: find a solution with optimal (maximum or minimum) value.
- *An* optimal solution, not *the* optimal solution, since may more than one optimal solution, any one is OK.

Typical steps of DP

- Characterize the structure of an optimal solution.
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution in a bottom-up fashion.
- Compute an optimal solution from computed/stored information.

Matrix Multiplication

- Note that any matrix multiplication between a matrix with dimensions **ixj** and another with dimensions **jxk** will perform **ixjxk** element multiplications creating an answer that is a matrix with dimensions **ixk**.

- Also note the condition that the second dimension in the first matrix and the first dimension in the second matrix must be equal in order to allow matrix multiplication (the number of columns of the first matrix should be equal to the number of rows of the second matrix) .

Matrix Multiplication

Algorithm to Multiply 2 Matrices

Input: Matrices $A_{p\times q}$ and $B_{q\times r}$ (with dimensions $p\times q$ and $q\times r$)

Result: Matrix $C_{p\times r}$ resulting from the product $A\cdot B$

```
MATRIX-MULTIPLY($A_{p\times q}$, $B_{q\times r}$)
1.  for $i \leftarrow 1$ to $p$
2.      for $j \leftarrow 1$ to $r$
3.          $C[i, j] \leftarrow 0$
4.          for $k \leftarrow 1$ to $q$
5.              $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
6.  return $C$
```

Scalar multiplication in line 5 dominates time to compute C
Number of scalar multiplications = pqr

Matrix Chain Multiplication

- Problem: given $<A_1, A_2, \ldots, A_n>$, compute the product: $A_1 \times A_2 \times \ldots \times A_n$, find the fastest way (i.e., minimum number of multiplications) to compute it.
- Given some matrices to multiply, determine the best order to multiply them so you minimize the number of single element multiplications.
  - i.e. Determine the way the matrices are parenthesized.

- First off, it should be noted that matrix multiplication is associative, but not commutative. But since it is associative, we always have:

- ((AB)(CD)) = (A(B(CD))), or any other grouping as long as the matrices are in the same consecutive order.

- BUT NOT: ((AB)(CD)) = ((BA)(DC))

Matrix-chain Multiplication

- Example: consider the chain A_1, A_2, A_3, A_4 of 4 matrices
  - Let us compute the product $A_1A_2A_3A_4$
- There are 5 possible ways:

1. $(A_1(A_2(A_3A_4)))$
2. $(A_1((A_2A_3)A_4))$
3. $((A_1A_2)(A_3A_4))$
4. $((A_1(A_2A_3))A_4)$
5. $(((A_1A_2)A_3)A_4)$

Matrix-chain Multiplication

- Matrix-chain multiplication problem
  - Given a chain $A_1, A_2, \ldots, A_n$ of n matrices, where for i=1, 2, …, n, matrix A_i has dimension $p_{i-1} \times p_i$
  - Parenthesize the product $A_1A_2 \ldots A_n$ such that the total number of scalar multiplications is minimized
- Brute force method of exhaustive search takes time exponential in n

Matrix-chain multiplication

- It may appear that the amount of work done won't change if you change the parenthesization of the expression, but we can prove that is not the case!
- Let us use the following example:
  - Let A be a 2x10 matrix
  - Let B be a 10x50 matrix
  - Let C be a 50x20 matrix

Matrix-chain multiplication

  - Let A be a 2x10 matrix
  - Let B be a 10x50 matrix
  - Let C be a 50x20 matrix
- Consider computing **A(BC):**
  - # multiplications for (BC) = 10x50x20 = 10000, creating a 10x20 answer matrix
  - # multiplications for A(BC) = 2x10x20 = 400
  - Total multiplications = 10000 + 400 = 10400.
- Consider computing **(AB)C**:
  - # multiplications for (AB) = 2x10x50 = 1000, creating a 2x50 answer matrix
  - # multiplications for (AB)C = 2x50x20 = 2000,
  - Total multiplications = 1000 + 2000 = 3000

Matrix-chain multiplication

- The second way is faster than the first!!!
- Different parenthesizations will have different number of multiplications for product of multiple matrices.
- Thus, our **goal** is:

"Given a chain of matrices to multiply, determine the fewest number of multiplications necessary to compute the product."

Matrix-chain multiplication –MCM DP

- Denote $<A_1, A_2, \ldots, A_n>$ by $< p_0, p_1, p_2, \ldots, p_n>$
  - i.e, $A_1(p_0,p_1)$, $A_2(p_1,p_2)$, …, $A_i(p_{i-1},p_i)$,… $A_n(p_{n-1},p_n)$
- Intuitive brute-force solution: Counting the number of parenthesizations by exhaustively checking all possible parenthesizations.
- Let P(n) denote the number of alternative parenthesizations of a sequence of n matrices:
  - $P(n) = \begin{cases} 1 \text{ if } n=1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) \text{ if } n\geq 2 \end{cases}$
- The solution to the recursion is $\Omega(2^n)$.
- So brute-force will not work.

MCP DP Steps

- Step 1: structure of an optimal parenthesization
  - Let $A_{i..j}$ ($i \leq j$) denote the matrix resulting from $A_i \times A_{i+1} \times \ldots \times A_j$
  - Any parenthesization of $A_i \times A_{i+1} \times \ldots \times A_j$ must split the product between A_k and A_{k+1} for some k, ($i \leq k < j$). The cost = # of computing $A_{i..k}$ + # of computing $A_{k+1..j}$ + # $A_{i..k} \times A_{k+1..j}$.
  - If k is the position for an optimal parenthesization, the parenthesization of "prefix" subchain $A_i \times A_{i+1} \times \ldots \times A_k$ within this optimal parenthesization of $A_i \times A_{i+1} \times \ldots \times A_j$ must be an optimal parenthesization of $A_i \times A_{i+1} \times \ldots \times A_k$.
  - $\underleftrightarrow{\underleftrightarrow{A_i \times A_{i+1} \times \ldots \times A_k} \times \underleftrightarrow{A_{k+1} \times \ldots \times A_j}}$

MCP DP Steps

- Step 2: a recursive relation
  - Let m[i,j] be the minimum number of multiplications for $A_i \times A_{i+1} \times \ldots \times A_j$
  - m[1,n] will be the answer
  - $$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i<j \end{cases}$$

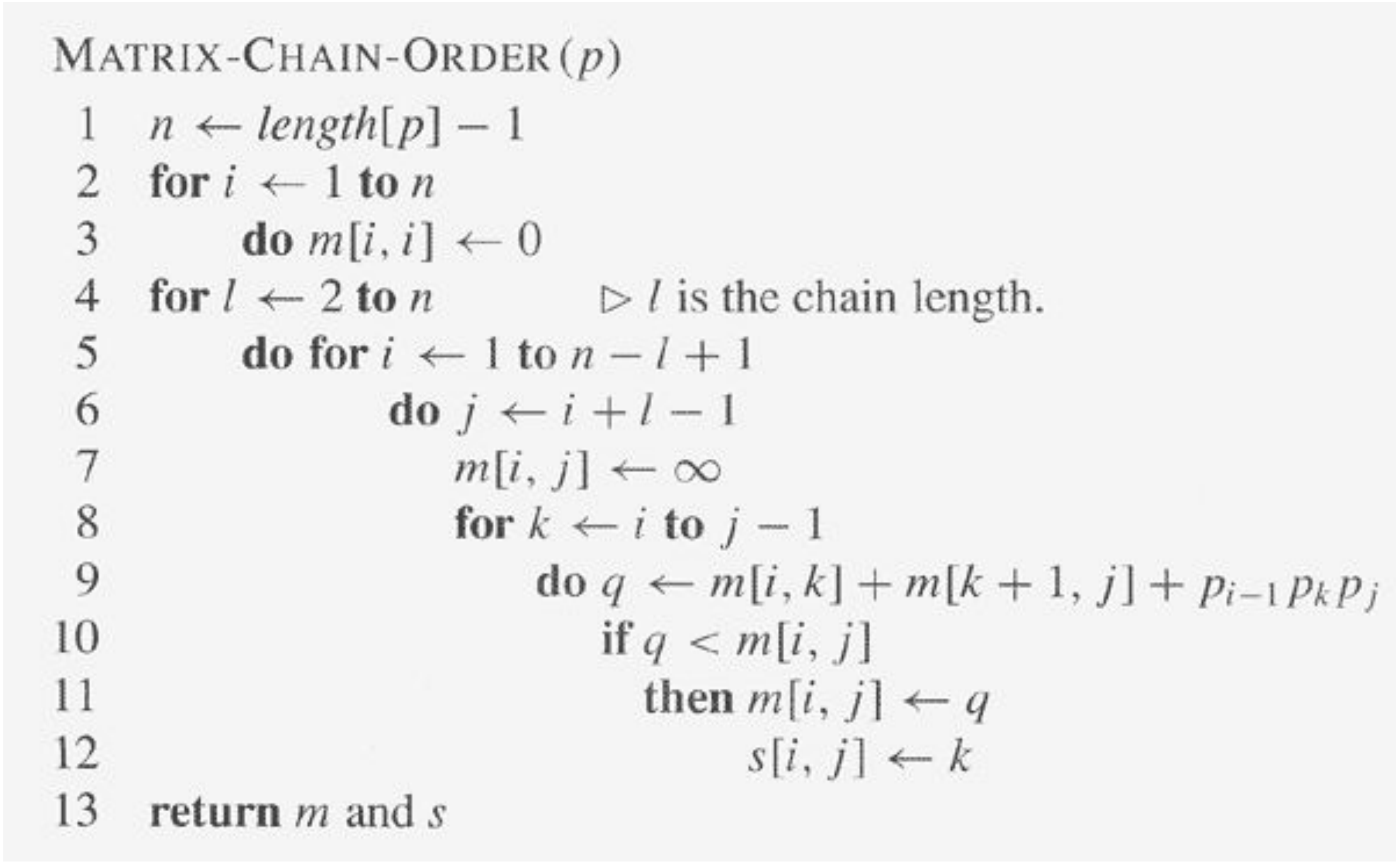
MCM DP Steps

- Step 3, Computing the optimal cost
  - If by recursive algorithm, exponential time $\Omega(2^n)$, no better than brute-force.
  - Total number of subproblems: $\binom{n}{2}+n = \Theta(n^2)$
  - Recursive algorithm will encounter the same subproblem many times.
  - If tabling the answers for subproblems, each subproblem is only solved once.
  - The second hallmark of DP: overlapping subproblems and solve every subproblem just once.

MCM DP Steps

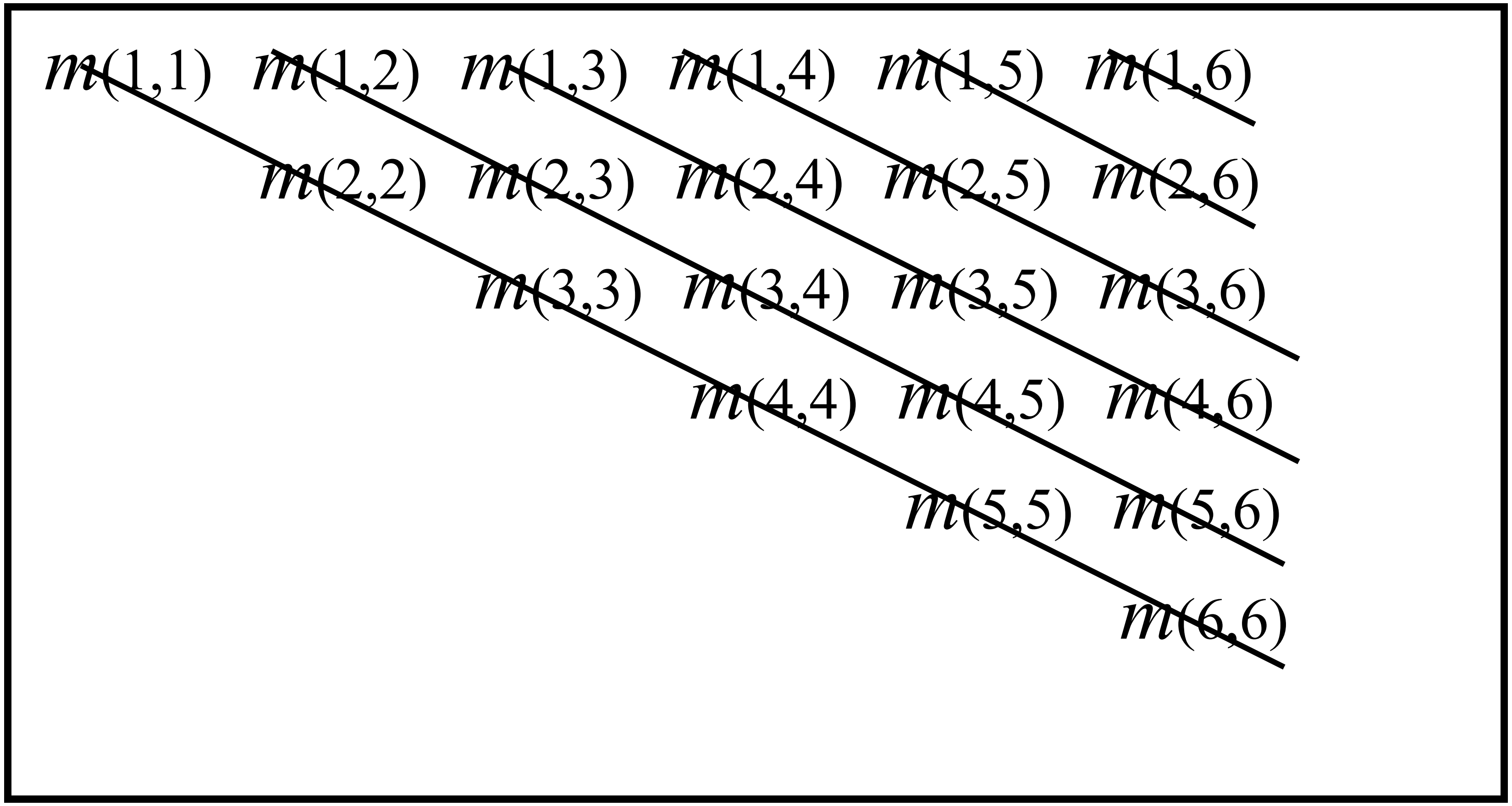
- Step 3, Algorithm,
  - array $m[1..n,1..n]$, with $m[i,j]$ records the optimal cost for $A_i \times A_{i+1} \times \ldots \times A_j$.
  - array $s[1..n,1..n]$, $s[i,j]$ records index k which achieved the optimal cost when computing $m[i,j]$.
  - Suppose the input to the algorithm is $p=< p_0 , p_1 ,\ldots, p_n >$.

MCM DP Steps



```
MATRIX-CHAIN-ORDER($p$)
$n \leftarrow length[p] - 1$
for $i \leftarrow 1$ to $n$
     do $m[i, i] \leftarrow 0$
for $l \leftarrow 2$ to $n$              ▷ $l$ is the chain length.
     do for $i \leftarrow 1$ to $n - l + 1$
            do $j \leftarrow i + l - 1$
               $m[i, j] \leftarrow \infty$
               for $k \leftarrow i$ to $j - 1$
                   do $q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$
                      if $q < m[i, j]$
                        then $m[i, j] \leftarrow q$
                             $s[i, j] \leftarrow k$
return $m$ and $s$
```

MCM DP—order of matrix computations



$m(1,1)$ $m(1,2)$ $m(1,3)$ $m(1,4)$ $m(1,5)$ $m(1,6)$
$m(2,2)$ $m(2,3)$ $m(2,4)$ $m(2,5)$ $m(2,6)$
$m(3,3)$ $m(3,4)$ $m(3,5)$ $m(3,6)$
$m(4,4)$ $m(4,5)$ $m(4,6)$
$m(5,5)$ $m(5,6)$
$m(6,6)$

Example

- Show how to multiply this matrix chain optimally

Matrix	Dimension
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

MCM DP Example

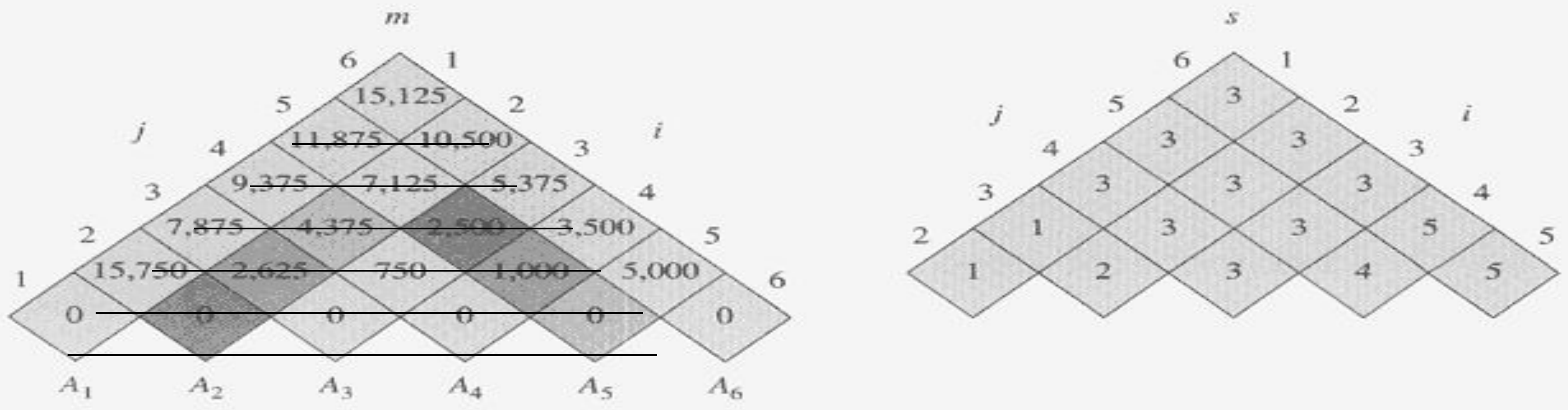


Figure 15.3 The m and s tables computed by MATRIX-CHAIN-ORDER for $n = 6$ and the following matrix dimensions:

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

The tables are rotated so that the main diagonal runs horizontally. Only the main diagonal and upper triangle are used in the m table, and only the upper triangle is used in the s table. The minimum number of scalar multiplications to multiply the 6 matrices is $m[1,6] = 15{,}125$. Of the darker entries, the pairs that have the same shading are taken together in line 9 when computing

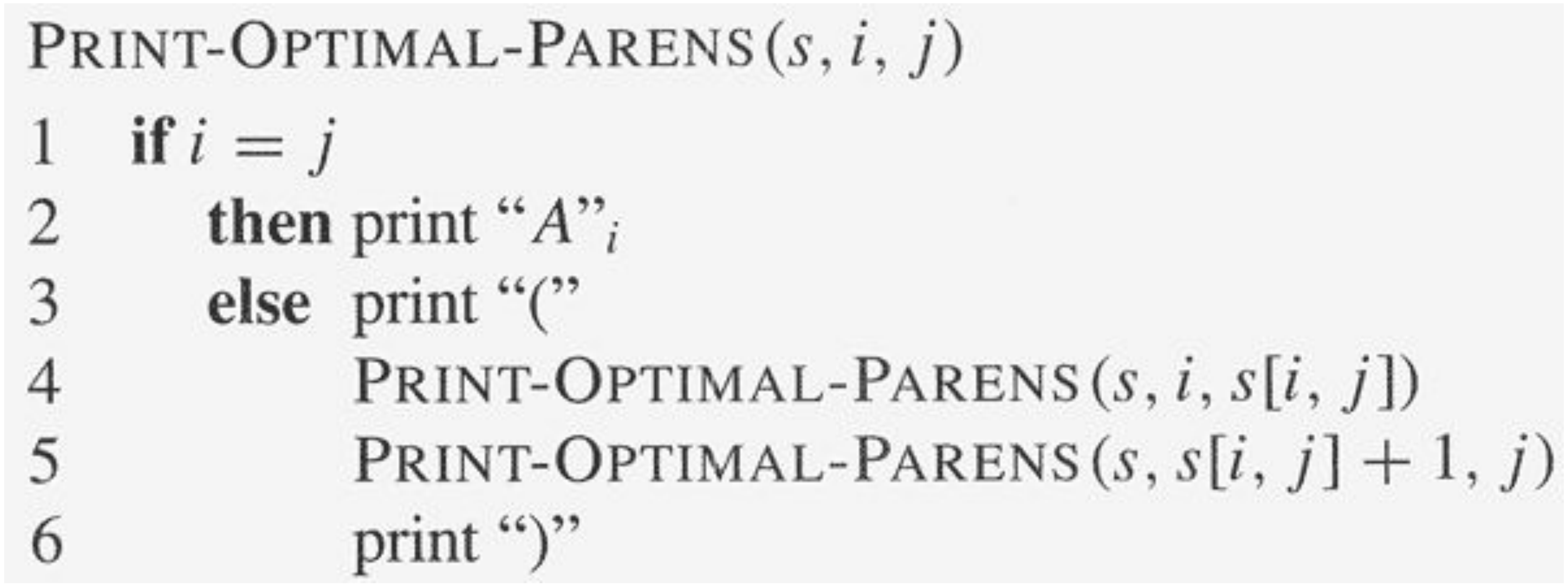
$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 & = 13000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 & = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 & = 11375 \end{cases}$$
$$= 7125\,.$$

MCM DP Steps

- Step 4, constructing a parenthesization order for the optimal solution.
  - Since $s[1..n,1..n]$ is computed, and $s[i,j]$ is the split position for $A_iA_{i+1}\ldots A_j$, i.e, $A_i\ldots A_{s[i,j]}$ and $A_{s[i,j]+1}\ldots A_j$, thus, the parenthesization order can be obtained from $s[1..n,1..n]$ recursively, beginning from $s[1,n]$.

MCM DP Steps

- Step 4, algorithm



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

Acknowledgments

- Slides adapted from: http://www.cs.ucf.edu/~dmarino/ucf/cop3503/lectures/
- https://home.cse.ust.hk/~dekai/271/notes/L12/L12.pdf