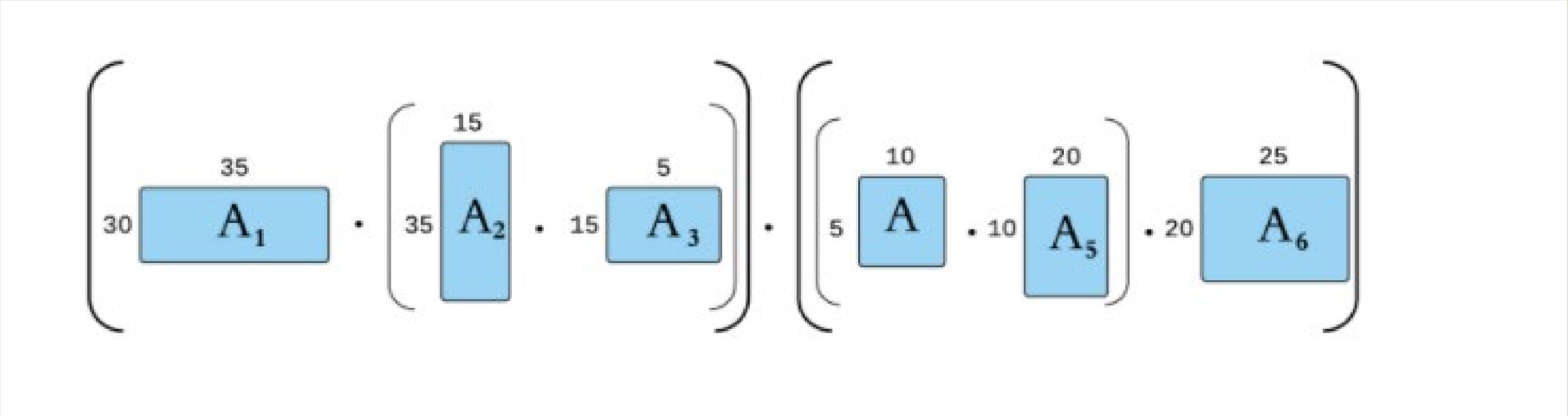
Matrix Chain Multiplications (15.2)

Given n matrices $A_1, A_2, \ldots, A_n$, to be multiplied, determine how to parenthesize the matrices to minimize the total number of multiplications.

For 4 matrices there are five ways to place parenthesis

$(A_1 (A_2 (A_3 A_4)))$
$(A_1 ((A_2 A_3) A_4))$
$((A_1 A_2) (A_3 A_4))$
$((A_1 (A_2 A_3)) A_4)$
$(((A_1 A_2) A_3) A_4)$

$$((A_1 (A_2 A_3)) ((A_4 A_5) A_6))$$



Assume matrices have the correct dimensions

- Matrix A_i has dimensions p_{i-1} by p_i
- Use standard cubic time matrix multiplication (15.2)
- Does the order matter?

Dimensions: A_1: 10 x 100, A_2: 100 x 5, A_3: 5 x 50

Assume matrices have the correct dimensions

- Matrix A_i has dimensions p_{i-1} by p_i
- Use standard cubic time matrix multiplication (15.2)
- Does the order matter?

Dimensions: A_1: 10 x 100, A_2: 100 x 5, A_3: 5 x 50

$((A_1 A_2) A_3)$

versus

$(A_1 (A_2 A_3))$

Assume matrices have the correct dimensions

- Matrix A_i has dimensions p_{i-1} by p_i
- Use standard cubic time matrix multiplication (15.2)
- Does the order matter?

Dimensions: A_1: 10 x 100, A_2: 100 x 5, A_3: 5 x 50

$((A_1 A_2) A_3)$

$(A_1 A_2)$ 10 x 100 x 5 = 5000

$((A_1 A_2) A_3)$ 10 x 5 x 50 = 2500 **7,500 multiplications**

$(A_1 (A_2 A_3))$

$(A_2 A_3)$ 100 x 5 x 50 = 25000

$(A_1 (A_2 A_3))$ 10 x 100 x 50 = 50000 **75,000 multiplications**

A product of matrices is **fully parenthesized** if it is either

- a single matrix, or
- a product of two fully parenthesized matrices, surrounded by parentheses

Each parenthesization makes **n-1** matrix multiplications.

We need to choose the parenthesis ordering that corresponds to the best ordering!

$$\mathbf{((A_1 A_2) A_3) (A_4 A_5)}$$

$$\mathbf{(A_1 (A_2 ((A_3 A_4) A_5)))}$$

Each parenthesization results in **n-1** matrix multiplications. We need to choose the best parenthesis ordering. ***How many orderings are there?***

Let P(k) be the number of ways to parenthesize k matrices.

- $\mathbf{P(n)} = \sum_{k=1}^{n-1} P(k)P(n-k) \quad \text{if } n \geq 2$

Example: $((A_1 A_2) A_3) \; (A_4 A_5)$

Recurrence is related to the Catalan numbers $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!}$

P(n) can be estimated as $\mathbf{\Theta(4^n/n^{3/2})}$

DP solution

1. ***Characterize the structure of an optimal solution***

   Look at the last multiplication done, say it is

$$(A_1 \ldots A_k)*(A_{k+1} \ldots A_n).$$

We know that in an optimal solution both subsolutions are solved in an optimal way.

We do not know the value of k.

What should the parameters of the DP recurrence be?

2. Recursively define the value of an optimal solution

Matrix A_i has dimensions p_{i-1} by p_i

m[i,j] = minimum number of multiplications computing $A_i * A_{i+1} \ldots *A_j$

2. Recursively define the value of an optimal solution

Matrix A_i has dimensions p_{i-1} by p_i

m[i,j] = minimum number of multiplications computing $A_i * A_{i+1} \dots *A_j$

m[i,j] = 0 for i=j

2. Recursively define the value of an optimal solution

Matrix A_i has dimensions p_{i-1} by p_i

m[i,j] = minimum number of multiplications computing $A_i * A_{i+1} \ldots * A_j$

m[i,j] = 0 for i=j

$(A_i \ldots A_k)*(A_{k+1} \ldots A_j)$

m[i,j]= ?

2. Recursively define the value of an optimal solution

Matrix A_i has dimensions p_{i-1} by p_i

m[i,j] = minimum number of multiplications computing $A_i * A_{i+1} \ldots * A_j$

m[i,j] = 0 for i=j

$(A_i \ldots A_k)*(A_{k+1} \ldots A_j)$

$$\boldsymbol{m}[\boldsymbol{i},\boldsymbol{j}] = \min\{\boldsymbol{m}[\boldsymbol{i},\boldsymbol{k}] + \boldsymbol{m}[\boldsymbol{k}+\mathbf{1},\boldsymbol{j}] + \boldsymbol{p}_{\boldsymbol{i}-\mathbf{1}} \cdot \boldsymbol{pk} \cdot \boldsymbol{p}_{\boldsymbol{j}}\}$$

for i<j and i≤k<j

3. Compute the entries in array m

Construct loops

- how many are nested?

Need to make sure all values needed have been computed

- Compute values diagonal by diagonal
  - first diagonal has n zeroes
  - second diagonal has n-1 elements, etc
- Last entry computed is m(1,n)
  - It represents the minimum number of multiplications.

MATRIX-CHAIN-ORDER(p)

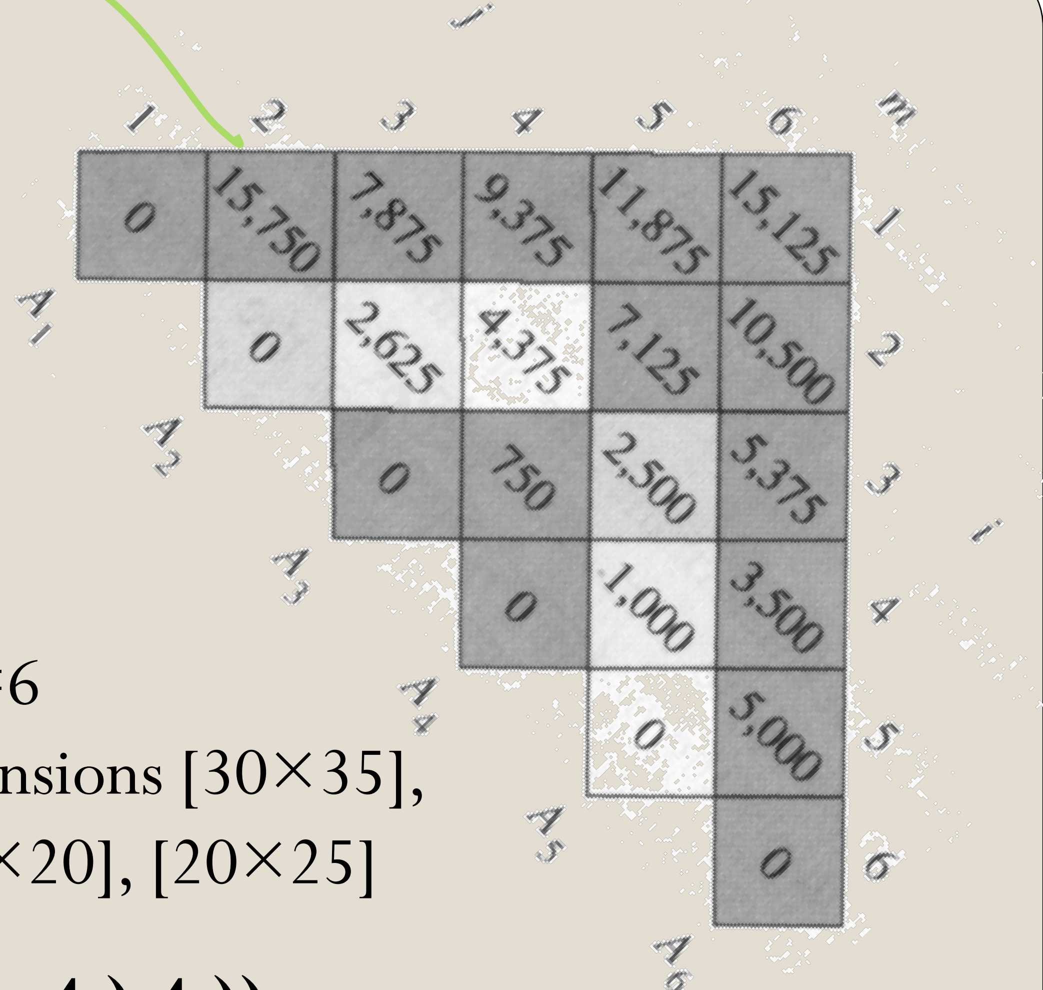
```
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            q = m[i, k] + m[k + 1, j] + p_{i-1}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

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    Index of diagonal
    for i = 1 to n - l + 1          Visit every position on the diagonal
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1          determine final value of m[i.j]
            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
```

$$A_i \times A_{i+1} \times \ldots A_k \times A_{k+1} \times \ldots \times A_{j-1} \times A_j$$

$m[1,2] = 30 \times 35 \times 15$



The *m*-table computed for n=6

$A_1, A_2, A_3, A_4, A_5, A_6$ with dimensions $[30 \times 35]$, $[35 \times 15]$, $[15 \times 5]$, $[5 \times 10]$, $[10 \times 20]$, $[20 \times 25]$

$$((A_1(A_2\,A_3))\,((A_4\,A_5)\,A_6))$$

$$m[1,6] = m[1,3] + m[4,6] + 30 \times 5 \times 25 =$$

$$7875 + 3500 + 3750 = 15{,}125$$

4. Construct an optimal solution

- Keep track on what indices contributed to the subsolutions
  - Do without increasing time and space (array s in code).
- The solution is created by tracing back
  - s[i,j]=k records the split point giving the optimum 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 ")"
```

4. *Construct an optimal solution*

- Keep track on what indices contributed to the subsolutions
  - Do without increasing time and space (array s in code).
- The solution is created by tracing back
  - s[i,j]=k records the split point giving the optimum solution.

Total: $O(n^3)$ time and $O(n^2)$ space

Note: there exists a more efficient (not DP) algorithm

A second solution in the book (15.3)

- "memoized" version
- Entries in matrix are not computed in a systematic iterative way, but on a need-basis so the recursion avoids recomputations
- Top-down approach
- Also $O(n^3)$ time and $O(n^2)$ space
- In general, bottom-up DP solutions are more efficient
  - Have more regular access to the matrices/tables (can be large)
  - In some problems, space requirements can be reduced

2. *Recursively define the value of an optimal solution*

matrix A_i has dimensions p_{i-1} by p_i

m[i,j] = minimum number of multiplications computing $A_i \ldots A_j$

$(A_i \ldots A_k) * (A_{k+1} \ldots A_j)$.

m[i,j] = 0 for $i=j$

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

for $i<j$ and $i \le k < j$

Memoized Recursive Algorithm

```
MEMOIZED-MATRIX-CHAIN(p)
n ← length[p] − 1
for i ← 1 to n
      do for j ← i to n
            do m[i, j] ← ∞
return LOOKUP-CHAIN(p, 1, n)
```

```
LOOKUP-CHAIN(p, i, j)
if m[i, j] < ∞
   then return m[i, j]
if i = j
   then m[i, j] ← 0
   else for k ← i to j − 1
              do q ← LOOKUP-CHAIN(p, i, k)
                       + LOOKUP-CHAIN(p, k + 1, j) + p_{i−1} p_k p_j
                 if q < m[i, j]
                    then m[i, j] ← q
return m[i, j]
```