Dynamic Programming

Matrix-Chain Multiplication

Chapter 15

Matrix Multiplication - Review

- We can multiply two matrices A and B only if they are compatible
  - Number of columns in A is the same as number of rows in B

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 2 & -1 \\ 1 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ -1 & 2 \end{bmatrix}$$

Matrix-Multiply(A, B)

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
```

- What is the running time if A is a p X q matrix and B is a q X r matrix?

Matrix-Chain Multiplication

- Suppose we have a sequence or chain A_1, A_2, …, A_n of n matrices to be multiplied
  - That is, we want to compute the product $A_1A_2\ldots A_n$

- There are many possible ways (parenthesizations) to compute the product

Example

- 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. $(((A_1A_2)A_3)A_4)$

Example

- A_1 is 10 x 100
- A_2 is 100 x 5
- A_3 is 5 x 50
- A_4 is 50 x 1
- $A_1A_2A_3A_4$ is a 10 by 1 matrix

- Let $A_{ij} = A_i \ldots A_j$

Example

- $(A_1(A_2(A_3A_4)))$
  - $A_{34} = A_3A_4$, 250 mults, result is 5 by 1
  - $A_{24} = A_2A_{34}$, 500 mults, result is 100 by 1
  - $A_{14} = A_1A_{24}$, 1000 mults, result is 10 by 1
  - Total is 1750

Example

- $((A_1A_2)(A_3A_4))$
  - $A_{12} = A_1A_2$, 5000 mults, result is 10 by 5
  - $A_{34} = A_3A_4$, 250 mults, result is 5 by 1
  - $A_{14} = A_{12}A_{34})$, 50 mults, result is 10 by 1
  - Total is 5300

Example

- $(((A_1A_2)A_3)A_4)$
  - $A_{12} = A_1A_2$, 5000 mults, result is 10 by 5
  - $A_{13} = A_{12}A_3$, 2500 mults, result is 10 by 50
  - $A_{14} = A_{13}A_4$, 500 mults, results is 10 by 1
  - Total is 8000

Example

- $((A_1(A_2A_3))A_4)$
  - $A_{23} = A_2A_3$, 25000 mults, result is 100 by 50
  - $A_{13} = A_1A_{23}$, 50000 mults, result is 10 by 50
  - $A_{14} = A_{13}A_4$, 500 mults, results is 10 by
  - Total is 75500

Example

- $(A_1((A_2A_3)A_4))$
  - $A_{23} = A_2A_3$, 25000 mults, result is 100 by 50
  - $A_{24} = A_{23}A_4$, 5000 mults, result is 100 by 1
  - $A_{14} = A_1A_{24}$, 1000 mults, result is 10 by 1
  - Total is 31000

Conclusion

- Order of the multiplications makes a difference
- How do we determine the order of multiplications that has the lowest cost?
- Note: We are not actually multiplying!

Parenthesization

- 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 defines a set of n-1 matrix multiplications. We just need to pick the parenthesization that corresponds to the best ordering.
- How many parenthesizations are there?

Parenthesization

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

$$P(n) = \begin{cases} 1 & if\ n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & if\ n \geq 2 \end{cases}$$

- Solution to this recurrence is $\Omega(2^n)$
- Checking all possible parenthesizations is not efficient!

Dynamic Programming

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution bottom-up
4. Construct an optimal solution from the computed information

1. Structure

- If the outermost parenthesizations is

$$((A_1 A_2 \cdots A_i)(A_{i+1} \cdots A_n))$$

- Then the optimal solution consists of solving

  A_{1i} and $A_{i+1,n}$

  optimally and then combining the solutions

2. Recursive Solution

- Let A_i have the dimension: p_{i-1} X p_i
- Let m[i,j] be the cost of computing A_{ij}
- If the final multiplication for A_{ij} is

$$A_{ij} = A_{ik} A_{k+1,j}$$

then

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

2. Recursive Solution

- We don't know k a priori, so we take the minimum

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

- Direct recursion on this does not work
  - It takes exponential time!
  - No better than brute force method

Step 3: Compute Optimal Cost

Matrix-Chain-Order(p)

```
n ← length[p] − 1
for i ← 1 to n
        do m[i, i] ← 0
for l ← 2 to n                        ▷ l is the chain length.
        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
return m and s
```

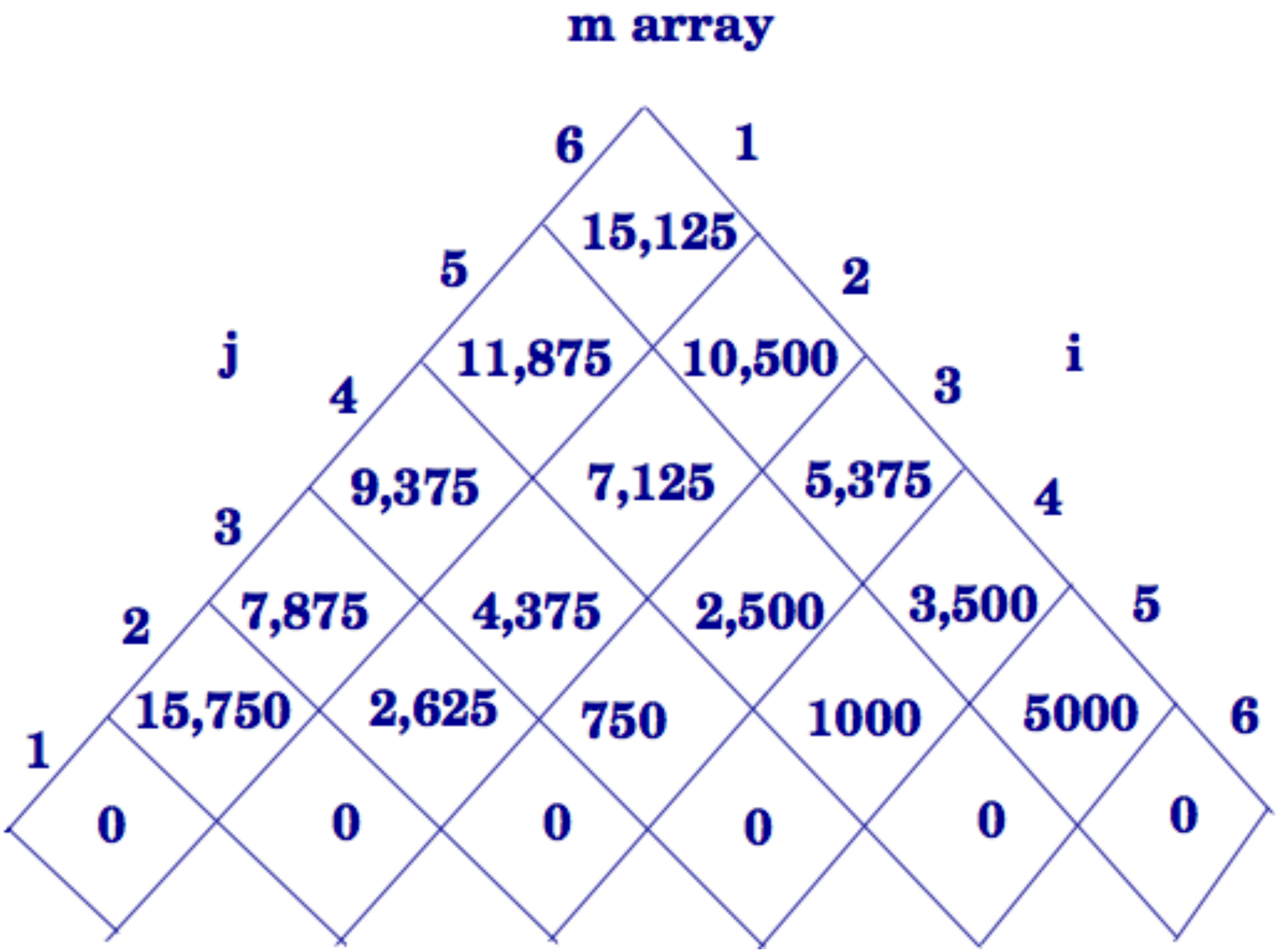
Example

- Let n=6
- Let p be:

matrix		**A_1**	**A_2**	**A_3**	**A_4**	**A_5**	**A_6**
Dimension		30x35	35x15	15x5	5x10	10x20	20x25

- What are the contents of m and s?

Arrays m and s



Cont.

- What is the optimal cost for multiplying the six matrices?

- Use the table m to calculate m[2,5]

Step 4: Constructing Solution

- So, we know the lowest cost, but what is the optimal parenthesization?

```
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 ")"
```