Chain Matrix Multiplication (矩阵链相乘)

Motivation. Suppose we want to multiply several matrices. This will involve iteratively multiplying two matrices at a time.

- Matrix multiplication is not *commutative* (in general $A \times B \neq B \times A$), but it is *associative*:

$$A \times (B \times C) = (A \times B) \times C$$

- We can compute product of matrices in many different ways, depending on how we parenthesize it.

Are some of these better than others?

Complexity of $C_{ik} = A_{ij} \times B_{jk}$

- Each element in C requires j multiplications, totally ik elements $\Rightarrow$ overall complexity $\Theta(ijk)$

Example

Suppose we want to multiply four matrices, $A \times B \times C \times D$, of dimensions 50×20, 20×1, 1×10, and 10×100, respectively.

Parenthesize	Computation	Cost
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	$120,200$
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	$60,200$
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	$7,000$

Example

Suppose we want to multiply four matrices, $A \times B \times C \times D$, of dimensions 50×20, 20×1, 1×10, and 10×100, respectively.

Parenthesize	Computation	Cost
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	$120,200$
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	$60,200$
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	$7,000$

The order of multiplication order makes a big difference in the final complexity.

Example

Suppose we want to multiply four matrices, $A \times B \times C \times D$, of dimensions 50×20, 20×1, 1×10, and 10×100, respectively.

Parenthesize	Computation	Cost
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	$120,200$
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	$60,200$
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	$7,000$

The order of multiplication order makes a big difference in the final complexity.

Natural greedy approach of always perform the cheapest matrix multiplication available may not always yield optimal solution

- see second parenthesization as a counterexample

Brute Force Algorithm

Q. How many different parenthesization methods (add brackets) for $A_1 A_2 \ldots A_n$?

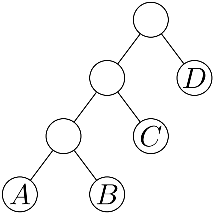
Brute Force Algorithm

Q. How many different parenthesization methods (add brackets) for $A_1 A_2 \ldots A_n$?

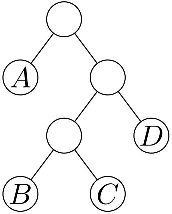
Observation. A particular parenthesiation can be represented naturally by a *full* binary tree

- leaves nodes: individual matrices
- the root node: final product
- interior nodes: intermediate products

$((A \times B) \times C) \times D$



$A \times ((B \times C) \times D)$



Estimate the Number of Possible Orders

The number of possible orders correspond to various full binary trees with n leaves.

Let $C(n)$ be the number of full binary tree with $n+1$ leaves, or, equivalently, with total n internal nodes:

$$C(0) = 1, C(1) = 1, C(2) = C(0)C(1) + C(1)C(0)$$
$$C(3) = C(0)C(2) + C(1)C(1) + C(2)C(0)$$
$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} = \frac{1}{n+1}\binom{2n}{n}$$

The above formula is of convolution form, can be calculated via generating function.

- The result is known as Catalan number, which is exponential in n

Brute Force Algorithm

Catalan number Occur in various counting problems (often involving recursively-defined objects)

- number of parenthesis methods
- number of full binary trees
- number of monotonic lattice paths

Since Catalan number is exponential in $n \rightsquigarrow$ we certainly cannot try each tree, with brute force thus ruled out.

We turn to dynamic programming.

Dynamic Programming

The correspondence to binary tree is suggestive: for a tree to be optimal, its subtrees must be also be optimal $\Rightarrow$ satisfy optimal substructure (has somewhat locality) $\rightsquigarrow$ do not have to try each tree from scratch

- subproblems corresponding to the subtrees: products of the form $A_i \times A_{i+1} \times \cdots A_j$

Optimized function:

$$C(i,j) = \text{minimum cost of multiplying } A_i \times A_{i+1} \times \cdots A_j$$

the corresponding dimension is $m_{i-1}, m_i, \ldots, m_j$

Iteration relation:

$$\underline{C(i,j)} = \begin{cases} 0 & i = j \\ \min_{i \le k < j}\{\underline{C(i,k)} + \underline{C(k+1,j)} + m_{i-1}m_k m_j\} & i < j \end{cases}$$

$$\begin{array}{|ccc|ccc|} \hline A_i & \ldots & A_k & A_{k+1} & \ldots & A_j \\ \hline \end{array}$$

$$m_{i-1} \times m_k \qquad\qquad m_k \times m_j$$

Some Remarks

Key points of DP

- Define subproblems
- Find iterative optimal substructure among subproblems
- Compute the subproblems in the right order

Sometimes the relation among subproblems may misleading. One should interpret and compute it in the right way, i.e., iterative.

Recursive Approach (inefficient)

Algorithm 1: MatrixChain(C, i, j) // subproblem $[i, j]$

$C(i,i) = 0$, $C(i,j) \leftarrow \infty$;
$s(i,j) \leftarrow \perp$ //record split position;
for $k \leftarrow i$ *to* $j-1$ **do**
    $t \leftarrow$ MatrixChain$(C, i, k) +$ MatrixChain$(C, k+1, j) + m_{i-1}m_k m_j$;
    if $t < C(i,j)$ **then** //find better solution
        $C(i,j) \leftarrow t$;
        $s(i,j) \leftarrow k$;
    end
end
return $C(i,j)$;

Iterative Approach (efficient)

size $= 1$: n different subproblems

- $C(i, i) = 0$ for $i \in [n]$ (no computation cost)

size $= 2$: $n - 1$ different subproblems

- $C(1, 2)$, $C(2, 3)$, $C(3, 4)$, $\ldots$, $C(n - 1, n)$

$\ldots$

size $= i$: $n - i + 1$ different subproblems

$\ldots$

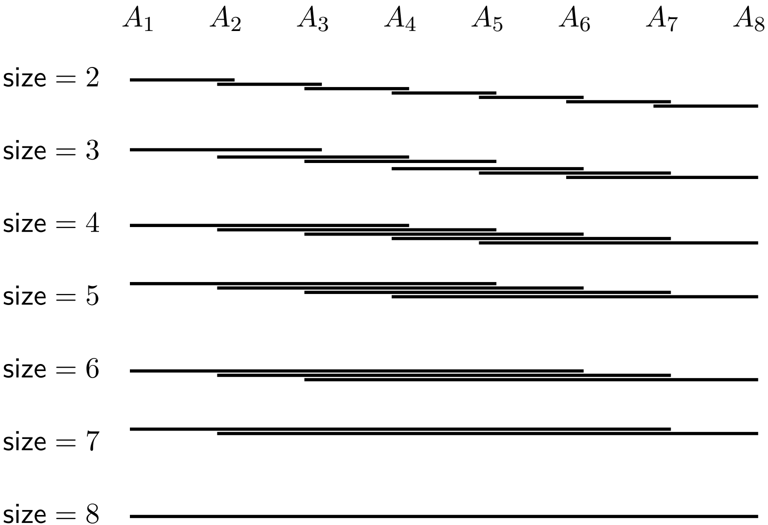
size $= n - 1$: 2 different subproblems

- $C(1, n - 1)$, $C(2, n)$

size $= n$: original problem

- $C(1, n)$

Demo of $n = 8$


A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
size $= 2$
size $= 3$
size $= 4$
size $= 5$
size $= 6$
size $= 7$
size $= 8$

Algorithm 2: MatrixChain(C, n)

$C(i,i) \leftarrow 0$, $C(i,j)_{i \neq j} \leftarrow +\infty$;
for $\ell \leftarrow 2$ *to* n **do** //size of subproblem
    for $i = 1$ *to* $n - \ell + 1$ **do** //left boundary i
        $j \leftarrow i + \ell - 1$ //right boundary j;
        for $k \leftarrow i$ *to* $j - 1$ **do** //try all split position
            $t \leftarrow C(i,k) + C(k+1,j) + m_{i-1}m_k m_j$;
            if $t < C(i,j)$ **then**
                $C(i,j) \leftarrow t$, $s(i,j) = k$ //update
            end
        end
    end
end

Algorithm 3: Trace(s, i, j) //initially $i = 1$, $j = n$

if *i*=*j* **then return**;
output $k \leftarrow s(i,j)$, Trace(s, i, k), Trace$(s, k+1, j)$;

Example

Matrix chain. $A_1A_2A_3A_4A_5$, $A_1 : 30 \times 35$, $A_2 : 35 \times 15$, $A_3 : 15 \times 5$, $A_4 : 5 \times 10$, $A_5 : 10 \times 20$

$\ell = 2$	$C(1,2) = 15750$	$C(2,3) = 2625$	$C(3,4) = 750$	$C(4,5) = 1000$
$\ell = 3$	$C(1,3) = 7875$	$C(2,4) = 4375$	$C(3,5) = 2500$	
$\ell = 4$	$C(1,4) = 9375$	$C(2,5) = 7125$		
$\ell = 5$	$C(1,5) = 11875$			

$\ell = 2$	$s(1,2) = 1$	$s(2,3) = 2$	$s(3,4) = 3$	$s(4,5) = 4$
$\ell = 3$	$s(1,3) = 1$	$s(2,4) = 3$	$s(3,5) = 3$	
$\ell = 4$	$s(1,4) = 3$	$s(2,5) = 3$		
$\ell = 5$	$s(1,5) = 3$			

$$s(1,5) \Rightarrow (A_1A_2A_3)(A_4A_5)$$

$$s(1,3) \Rightarrow A_1(A_2A_3)$$

- optimal computation order: $(A_1(A_2A_3))(A_4A_5)$
- minimum multiplication: $C(1,5) = 11875$