Matrix-chain multiplication:

We are given a sequence (chain) $< A_1, A_2, \ldots, A_n >$ of n matrices to be multiplied, and we wish to compute the product $A_1 A_2 \cdots A_n$.

We can evaluate this product using the standard algorithm for multiplying pairs of matrices as a subroutine once we have parenthesized it to resolve all ambiguities in how the matrices are multiplied together.

A product of matrices is **fully parenthesized** if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses. (recursive definition)

Matrix multiplication is associative, and so all parenthesizations yield the same product.

Eg: For example, if the chain of matrices is $< A_1, A_2, A_3, A_4 >$, the product $A_1 A_2 A_3 A_4$ can be fully parenthesized in five distinct ways:

$$\begin{array}{l}(A_1(A_2(A_3A_4)))\ ,\\ (A_1((A_2A_3)A_4))\ ,\\ ((A_1A_2)(A_3A_4))\ ,\\ ((A_1(A_2A_3))A_4)\ ,\\ (((A_1A_2)A_3)A_4)\ .\end{array}$$

The way we parenthesize a chain of matrices can have a dramatic impact on the cost of evaluating the product.

Note:

A full parenthesization of an n-element expression has exactly $n-1$ pairs of parentheses. \\Exercise eg $\langle A_1 \rangle$ (single matrix is fully parenthesized) (only one way)

Eg $\langle A_1 A_2 \rangle$

$(A_1 A_2)$ (only one possible way to parenthesize a sequence of two matrices.)

Eg $\langle A_1 A_2 A_3 \rangle$

$$\left.\begin{array}{l} ((A_1 A_2) A_3) \\ (A_1 (A_2 A_3)) \end{array}\right\}$$ Total 2 ways.

Eg $\langle A_1 A_2 A_3 A_4 \rangle$

$$\left.\begin{array}{l} (A_1 (A_2 (A_3 A_4))) \\ ((A_1 (A_2 A_3)) A_4) \\ (A_1 ((A_2 A_3) A_4)) \\ (((A_1 A_2) A_3) A_4) \\ ((A_1 A_2) (A_3 A_4)) \end{array}\right\}$$ total 5 ways.

2020/11/11 08:31

Multiplying two matrices

We can multiply two matrices A and B only if they are compatible: the number of columns of A must equal the number of rows of B. If A is a p ×q matrix and B is a q ×r matrix, the resulting matrix C is a p ×r matrix.

The standard algorithm is given by the following pseudocode. The attributes rows and columns are the numbers of rows and columns in a matrix.

```
MATRIX-MULTIPLY(A, B)
1  if columns[A] ≠ rows[B]
2     then error "incompatible dimensions"
3     else for i ← 1 to rows[A]
4              do for j ← 1 to columns[B]
5                     do C[i, j] ← 0
6                        for k ← 1 to columns[A]
7                            do C[i, j] ← C[i, j] + A[i, k] · B[k, j]
8           return C
```

The time to compute C is dominated by the number of scalar multiplications in line 7, which is **pqr**. In what follows, we shall express costs in terms of the number of scalar multiplications.

Illustrative Example:

To understand the different costs incurred by different parenthesizations of a matrix product, consider the problem of 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.

Case1: If we multiply according to the parenthesization *((A_1 A_2)A_3)*

we perform 10*100*5 =5000 scalar multiplications to compute the 10 × 5 matrix product (A1 A2),

and another 10*5*50 = 2500 scalar multiplications to multiply this matrix by A3, for a total of **7500 s**calar multiplications.

Case2: If we multiply according to the parenthesization *(A_1(A_2 A_3))*,

we perform 100*5 *50 = 25,000 scalar multiplications to compute the 100 × 50 matrix product (A2 A3), and another 10*100* 50 = 50,000 scalar multiplications to multiply A1 by this matrix, for a total of **75,000** scalar multiplications.

Thus, computing the product according to the first parenthesization is 10 times faster.

Problem Statement:

The MCM problem can be stated as follows:

Given a chain $< A_1, A_2, \ldots, A_n >$ of n matrices,

where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} * p_i$,

fully parenthesize the product $A_1 \; A_2 \cdot\cdot\cdot A_n$ 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 f**or multiplying matrices that has the **lowest cost**.

Counting the number of parenthesizations

Before solving the matrix-chain multiplication problem by dynamic programming, let us convince ourselves that exhaustively checking all possible parenthesizations does not yield an efficient algorithm.

Denote the number of alternative parenthesizations of a sequence of n matrices by P(n).

When n = 1, there is just one matrix and therefore only one way to fully parenthesize the matrix product.

When $n \geq 2$, a fully parenthesized matrix product is the product of two fully parenthesized matrix

subproducts, and the split between the two subproducts may occur between the kth and (k + 1)st matrices for any $k = 1, 2, \ldots, n - 1$.

Therefore, we obtain the following recurrence relation:

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

The solution to the above recurrence (15.11) is $\Omega(2^n)$ \\ exercise ques

The number of solutions is thus exponential in n, and the brute-force method of exhaustive search is therefore a poor strategy for determining the optimal parenthesization of a matrix chain.

Eg

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

Find the value $P(5)$? $\langle A_1, A_2, A_3, A_4, A_5 \rangle$

Sol^n

$$P(5) = \sum_{k=1}^{(5-1)} P(k)\, P(5-k)$$

$$\Rightarrow P(5) = P(1)P(5-1) + P(2)P(5-2) + P(3)P(5-3) + P(4)P(5-4)$$

* $A_1 | A_2 A_3 A_4 A_5$; $A_1 A_2 | A_3 A_4 A_5$; $A_1 A_2 A_3 | A_4 A_5$; $A_1 A_2 A_3 A_4 | A_5$

(for understanding purpose)

$= 1 \times 5 + 1 \times 2 + 2 \times 1 + 5 \times 1$

$= 5 + 2 + 2 + 5$

$= 14$ ways

we have
$P(1) = 1$
$P(2) = 1$
$P(3) = 2$
$P(4) = 5$

Step 1: The structure of an optimal parenthesization

Our first step in the dynamic-programming paradigm is to **find the optimal substructure** and then use it to construct an optimal solution to the problem from optimal solutions to subproblems.

For the matrix-chain multiplication problem, we can perform this step as follows:

Let $A_{i..j}$, denotes the matrix that results from evaluating the product $A_i\ A_{i+1} \dots A_j$, where $i \leq j$.

Observe that if the problem is nontrivial, i.e., $i < j$,

then any parenthesization of the product $A_i\ A_{i+1} \dots A_j$ must split the product between A_k and A_{k+1} for some integer k in the range $i \leq k < j$.

That is, for some value of k, we first compute the matrices $A_{i..k}$ and $A_{k+1..j}$ and then multiply them together to produce the final product $A_{i..j}$.

Therefore, the cost of this parenthesization is thus the cost of computing the matrix $A_{i..k,}$ plus the cost of computing $A_{k+1..j,}$ plus the cost of multiplying them together.

The **optimal substructure** of this problem is as follows:

Suppose that an optimal parenthesization of $A_i\ A_{i+1} \dots A_j$ splits the product between A_k and A_{k+1}.

Then the parenthesization of the "prefix" subchain $A_iA_{i+1} \dots A_k$ within this optimal parenthesization of $A_i\ A_{i+1} \dots A_j$ must be an optimal parenthesization of $A_i\ A_{i+1} \dots A_k$. **Why?**

If there were a less costly way to parenthesize $A_iA_{i+1} \dots A_k$, substituting that parenthesization in the optimal parenthesization of $A_i\ A_{i+1} \dots A_j$ would produce another parenthesization of $A_i\ A_{i+1} \dots A_j$ whose cost was lower than the optimum: a contradiction.

A similar observation holds for the parenthesization of the subchain $A_{k+1}A_{k+2} \cdots A_j$ in the optimal parenthesization of $A_i\ A_{i+1} \dots A_j$: it must be an optimal parenthesization of $A_{k+1}A_{k+2} \cdots A_j$.

Thus, any solution to a nontrivial instance of the matrix-chain multiplication problem requires us to split the product, and that any optimal solution contains within it optimal solutions to subproblem instances.

We must ensure that when we search for the correct place to split the product, we have considered all possible places so that we are sure of having examined the optimal one.

Step 2: A recursive solution

Now, we define the cost of an optimal solution recursively in terms of the optimal solutions to subproblems.

For MCM problem, we pick as our subproblems the problems of determining the minimum cost of a parenthesization of $A_i\, A_{i+1} \dots A_j$ for $1 \le i \le j \le n$.

Let m[i, j] be the minimum number of scalar multiplications needed to compute the matrix Ai.. j ; for the full problem, the cost of a cheapest way to compute $A_{1..n}$ would thus be m[1, n].

We can define m[i, j] recursively as follows:

If i = j , the problem is trivial;

the chain consists of just one matrix $A_{i..i} = A_i$, so that no scalar multiplications are necessary to compute the product. Thus, m[i, i] = 0 for i = 1, 2, . . . , n.

To compute m[i, j] when i < j , we take advantage of the structure of an optimal solution **from step 1.** Let us assume that the optimal parenthesization splits the product $A_i\, A_{i+1} \dots A_j$ between A_k and A_{k+1}, where $i \le k < j$. Then, m[i, j] is equal to the minimum cost for computing the subproducts $A_{i..k}$ and $A_{k+1..j}$, plus the cost of multiplying these two matrices together.

Recalling that each matrix A_i is $p_{i-1} \times p_i$, we see that computing the matrix product $A_{i..k}\, A_{k+1..j}$ takes $p_{i-1}p_kp_j$ scalar multiplications. Thus, we can write:

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

This recursive equation assumes that we know the value of k, which we do not.

There are only j −i possible values for k, however, namely k = i, i +1, . . . , j −1. Since the optimal parenthesization must use one of these values for k, we need only check them all to find the best.

Thus, our recursive definition for the minimum cost of parenthesizing the product $A_i\, A_{i+1} \dots A_j$ becomes

$$m[i,j] = \begin{cases} 0 & \text{if } i = j\,, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\} & \text{if } i < j\,. \end{cases}$$

To help us keep track of how to construct an optimal solution, let us define s[i, j] to be a value of k at which we can split the product $A_i\, A_{i+1} \dots A_j$ to obtain an optimal parenthesization.

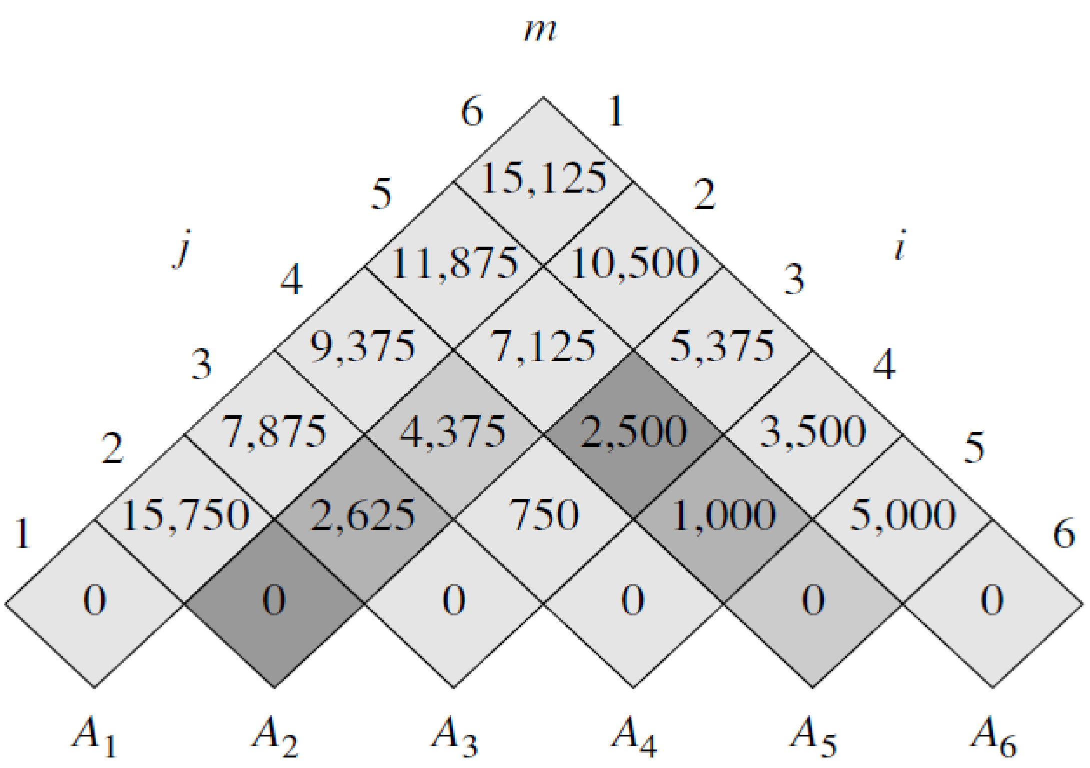
That is, s[i, j] equals a value k such that $m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$.

Step 3: Computing the optimal costs

Example: Consider the chain of six matrices <A1, A2, A3, A4, A5, A6>; where matrices have following 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

Fully parenthesize the product A1A2…A6 in a way that minimize the number of scalar multiplications.



m
j
i
1
2
3
4
5
6
1
2
3
4
5
6
15,125
11,875
10,500
9,375
7,125
5,375
7,875
4,375
2,500
3,500
15,750
2,625
750
1,000
5,000
0
0
0
0
0
0
A_1
A_2
A_3
A_4
A_5
A_6