Matrix chain multiplication problem

Let **A** be an $m \times n$ matrix and **B** be an $n \times p$ matrix:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

The matrix product **C** = **AB** is defined to be the $m \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

for $i = 1, ..., m$ and $j = 1, ..., p$.

That is, the entry c_{ij} of the product is obtained by multiplying term-by-term the entries of the i-th row of **A** and the j-th column of **B**, and summing these n products. In other words, c_{ij} is the dot product of the i-th row of **A** and the j-th column of **B**.

Therefore, **AB** can also be written as

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Thus the product **AB** is defined if and only if two matrices **A** and **B** are ***compatible***: the number of columns in **A** equals the number of rows in **B**, in this case n.

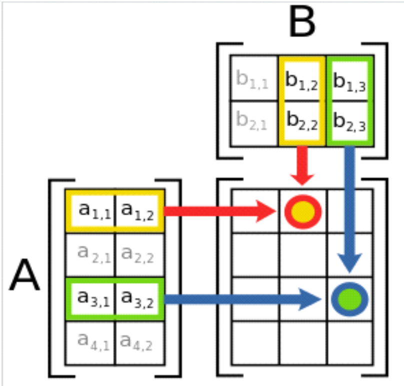
A * B = C

$m \times n$ $n \times p$ $m \times p$

number of operations = m * n * p

To multiply matrix **A** of size $m \times n$ by the matrix **B** of size $n \times p$ we get a matrix **C** of size $m \times p$. Number of operations for matrix multiplication is proportional to

$$m * n * p$$



A * B = C

4 x 2 2 x 3 4 x 3

number of operations = 4 * 2 * 3 = 24

E-OLYMP **1482. Matrix multiplication** Find the product of two matrices.

► Multiply matrices using the formula:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} \text{, where } i = 1, 2, \ldots, m; j = 1, 2, \ldots, q.$$

Store the matrices A, B, C in two dimentional arrays a, b, c. Let A has the size $na \times ma$, B has the size $nb \times mb$. Matrices are ***compatible*** for multiplication if $ma = nb$. Resulting matrix C has the size $na \times mb$.

```
for (i = 0; i < na; i++)
for (j = 0; j < mb; j++)
for (k = 0; k < ma; k++)
  c[i][j] += a[i][k] * b[k][j];
```

Matrix chain multiplication problem

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 $A_1 * A_2 * \ldots * A_n$.

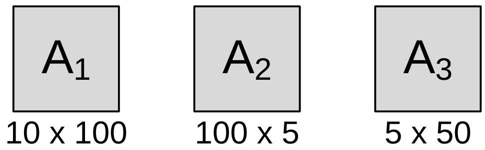
We can evaluate the expression using the standard algorithm for multiplying pairs of matrices as a subroutine once we have parenthesized it to resolve all ambiguities in how the matrices are multiplied together. A product of matrices is *fully parenthesized* if it is either

- a single matrix;
- the product of two fully parenthesized matrix products, surrounded by parentheses;

Matrix multiplication is associative, and so all parenthesizations yield the same product. For example, if the chain of matrices is $<A_1, A_2, A_3, A_4>$, the product $A_1A_2A_3A_4$ can be fully parenthesized in five distinct ways:

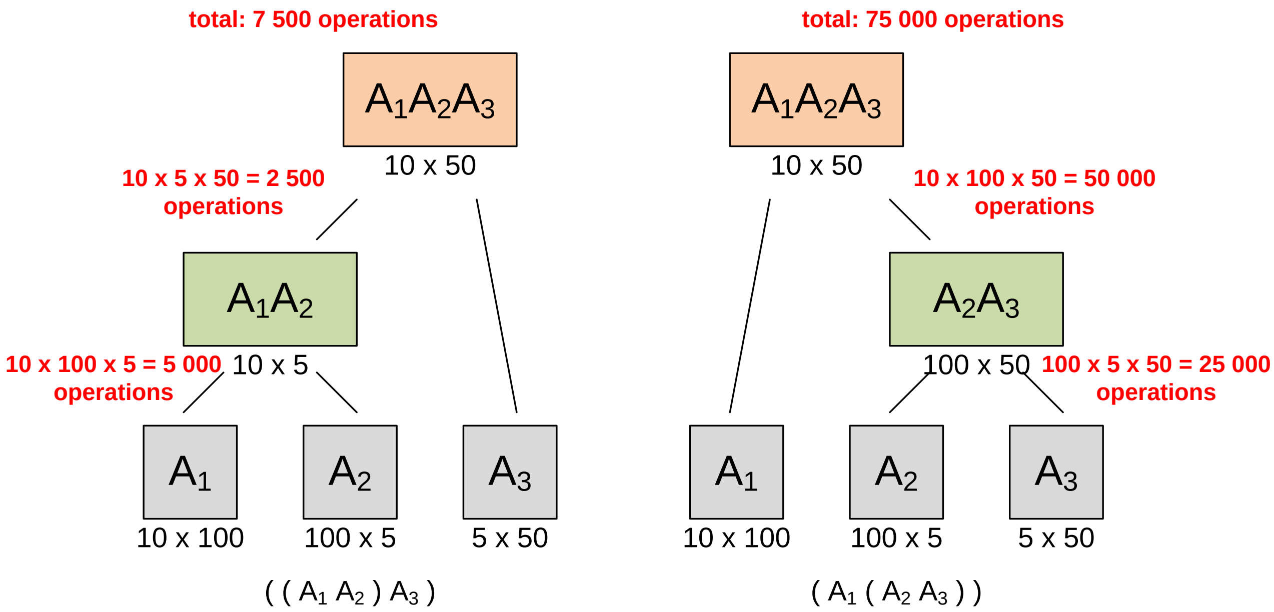
$$(A_1 (A_2 (A_3 A_4))),$$
$$(A_1 ((A_2 A_3) A_4)),$$
$$((A_1 A_2) (A_3 A_4)),$$
$$((A_1 (A_2 A_3)) A_4),$$
$$(((A_1 A_2) A_3) A_4).$$

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



If we multiply according to the parenthesization ((A_1 A_2) A_3), we perform 7500 scalar multiplications.

If we multiply according to the parenthesization (A_1 (A_2 A_3)), we perform 75000 scalar multiplications.



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

The **matrix-chain multiplication problem** can be stated as follows: given a chain <A_1, A_2, . . . , A_n> of n matrices, where for $i = 1, 2, . . . , n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 . . . A_n$ in a way that ***minimizes*** the number of scalar multiplications.

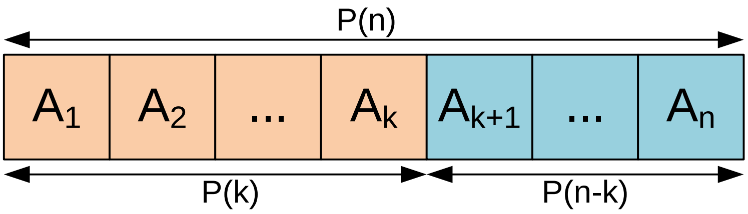
Note that in the matrix-chain multiplication problem, we are not actually multiplying matrices. Our goal is only to determine an **order** for multiplying matrices that has the **lowest cost**.

Counting the number of parenthesizations

First 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).

If $n = 1$, there is just one matrix and therefore only one way to fully parenthesize the matrix product. P(1) = 1

If $n \geq 2$, 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 k-th and $(k + 1)$-st matrices for any $k = 1, 2, \ldots, n - 1$.

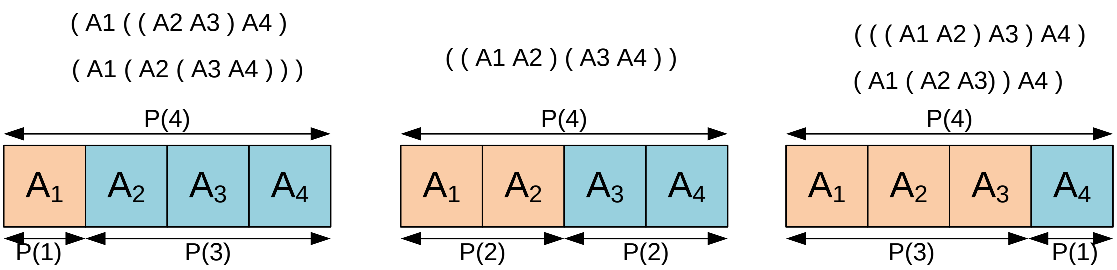


Thus, we obtain the recurrence

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

For example,

P(1) = 1
P(2) = P(1) * P(1) = 1;
P(3) = P(1) * P(2) + P(2) * P(1) = 1 + 1 = 2;
P(4) = P(1) * P(3) + P(2) * P(2) + P(3) * P(1) = 2 + 1 + 2 = 5;
P(5) = P(1) * P(4) + P(2) * P(3) + P(3) * P(2) + P(4) * P(1) = 5 + 2 + 2 + 5 = 14



The solution to a *similar recurrence* is the sequence of **Catalan numbers**, which grows as $\Omega(4^n / n^{3/2})$. 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.

Catalan numbers are given by recurrence relation:

$$c_0 = 1,$$

$$c_n = c_0 c_{n-1} + c_1 c_{n-2} + c_2 c_{n-3} + \ldots + c_{n-1} c_0 = \sum_{k=0}^{n-1} c_k c_{n-k-1}, \text{ if } n > 0$$

We have: $c_0 = 1$, $c_1 = 1$, $c_2 = 2$, $c_3 = 5$, $c_4 = 14$, … . So P(i) = c_{i-1}.

E-OLYMP **9643. Catalan numbers** Compute the n-th Catalan numbers modulo m.

► Let's compute first Catalan numbers:

- $c_0 = 1$
- $c_1 = c_0 c_0 = 1$,

- $c_2 = c_0c_1 + c_1c_0 = 1 + 1 = 2$,
- $c_3 = c_0c_2 + c_1c_1 + c_2c_0 = 2 + 1 + 2 = 5$,
- $c_4 = c_0c_3 + c_1c_2 + c_2c_1 + c_3c_0 = 5 + 2 + 2 + 5 = 14$,
- $c_5 = c_0c_4 + c_1c_3 + c_2c_2 + c_3c_1 + c_4c_0 = 14 + 5 + 4 + 5 + 14 = 42$

Since the value of c_n is recalculated through all the previous values of $c_0, c_1, c_2, ..., c_{n-1}$, then the values of the Catalan numbers we shall store in linear array

```
long long cat[10001]
```

Calculate the Catalan numbers using the recurrent formula.

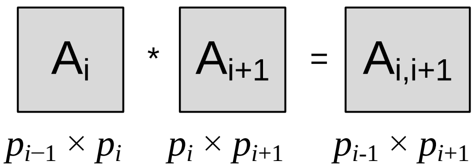
```
cat[0] = 1;
for (i = 1; i <= n; i++)
{
  for (j = 0; j < i; j++)
    cat[i] = cat[i] + cat[j] * cat[i - j - 1];
}
```

Do not forget in this problem to make calculations modulo *m*.

Recurrent formula

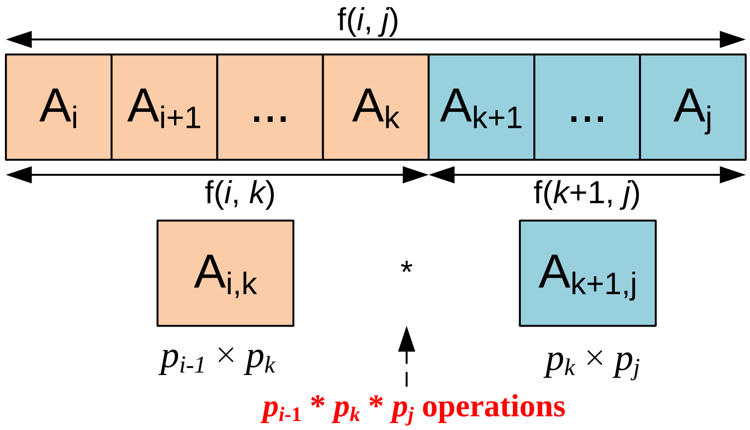
Let A_{ij} be the product of matrices $A_iA_{i+1}\ldots A_j$.
Let f(i, j) be the minimum cost of computing the value of A_{ij}.
It's obvious that:

- f(i, i) = 0 because $A_{ii} = A_i$ (chain consists of just one matrix);
- $f(i, i + 1) = p_{i-1} * p_i * p_{i+1}$ because we multiply matrices of sizes $p_{i-1} \times p_i$ and $p_i \times p_{i+1}$.



Let us assume that the optimal parenthesization splits the product $A_iA_{i+1}\ldots A_j$ between A_k and A_{k+1}, where $i \le k < j$. Note that

- for $k = i$ we have the product $A_i * A_{i+1}\ldots A_j$;
- for $k = j - 1$ we have the product $A_iA_{i+1}\ldots A_{j-1} * A_j$;



The value of f(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 (which is $p_{i-1} * p_k * p_j$). Thus, we obtain

$$f(i, j) = f(i, k) + f(k + 1, j) + p_{i-1} * p_k * p_j$$

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 $k = i, i + 1, \ldots, j - 1$. 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 $A_i A_{i+1} \ldots A_j$ becomes

$$f(i, j) = \begin{cases} 0, \text{ if } i = j \\ \min\limits_{i \le k < j} \left(f(i, k) + f(k+1, j) + p_{i-1} \cdot p_k \cdot p_j \right), \text{ if } i < j \end{cases}$$

Example

Consider the next four matrices that we want to multiply:

A_1	*	A_2	*	A_3	*	A_4
2 x 4		4 x 5		5 x 3		3 x 6

The size of matrix A_i is $p_{i-1} \times p_i$, array p contains the values (2, 4, 5, 3, 6), indexation starts from 0, i.e. $p_0 = 2$.

i	0	1	2	3	4
p_i	2	4	5	3	6

Let the values of f(i, j) will be saved in dp[i][j]. Initially set

$$dp[i][j] = \infty \ (i \ne j), \ dp[i][i] = 0$$

Next compute the values $dp[i][i + 1] = p_{i-1} * p_i * p_{i+1}$:

- $dp[1][2] = p_0 * p_1 * p_2 = 2 * 4 * 5 = 40$;
- $dp[2][3] = p_1 * p_2 * p_3 = 4 * 5 * 3 = 60$;
- $dp[3][4] = p_2 * p_3 * p_4 = 5 * 3 * 6 = 90$;

	1	2	3	4
1	0	∞	∞	∞
2		0	∞	∞
3			0	∞
4				0

dp[i][i] = 0

	1	2	3	4
1	0	40	∞	∞
2		0	60	∞
3			0	90
4				0

$dp[i][i+1] = p_{i-1} * p_i * p_{i+1}$

Continue the computations:

$$dp[1][3] = \textbf{MIN} \begin{cases} dp[1][2] + dp[3][3] + p_0p_2p_3 = 40 + 0 + 2*5*3 = 70 \\ dp[1][1] + dp[2][3] + p_0p_1p_3 = 0 + 60 + 2*4*3 = 84 \end{cases} = \boxed{70}$$

$$dp[2][4] = \textbf{MIN} \begin{cases} dp[2][3] + dp[4][4] + p_1p_3p_4 = 60 + 0 + 4*3*6 = 132 \\ dp[2][2] + dp[3][4] + p_1p_2p_4 = 0 + 90 + 4*5*6 = 210 \end{cases} = \boxed{132}$$

And here is how to find the final value dp[1][4]:

$$dp[1][4] = \textbf{MIN} \begin{cases} dp[1][3] + dp[4][4] + p_0p_3p_4 = 70 + 0 + 2*3*6 = 106 \\ dp[1][2] + dp[3][4] + p_0p_2p_4 = 40 + 90 + 2*5*6 = 190 \\ dp[1][1] + dp[2][4] + p_0p_1p_4 = 0 + 132 + 2*4*6 = 180 \end{cases} = \boxed{106}$$

	1	2	3	4
1	0	40	70	106
2		0	60	132
3			0	90
4				0

E-OLYMP **9647. Optimal Matrix Multiplication** Chain of matrices is given. Print the minimum number of multiplications sufficient to multiply all matrices.

► Declare the constants INF = ∞, MAX = 11 (maximum possible number of matrices in the product). Declare arrays dp and p.

```
#define INF 0x3F3F3F3F3F3F3F3FLL
#define MAX 11
long long dp[MAX][MAX], p[MAX];
```

Function ***Mult*** finds the minimum number of multiplications sufficient to compute $A_{ij} = A_i * A_{i+1} * \ldots * A_{j-1} * A_j$, which is saved in the cell dp[*i*][*j*].

```
long long Mult(int i, int j)
{
  if (dp[i][j] == INF)
    for (int k = i; k < j; k++)
    {
      long long temp = Mult(i, k) + Mult(k + 1, j) +
                       p[i - 1] * p[k] * p[j];
      if (temp < dp[i][j]) dp[i][j] = temp;
  }
  return dp[i][j];
```

```
}
```

In the main part of the program after reading the data, make a call

```
Mult(1,n);
```

to compute the result, the minimum number of multiplications to find the optimal product of matrices $A_1 * A_2 * \ldots * A_{n-1} * A_n$.

E-OLYMP **1521. Optimal Matrix Multiplication - 2** Chain of matrices is given. Find the way to multiply them ***minimizing*** the number of scalar multiplications.

► Use the recurrent formula given above.