Chapter 23

Matrices

A **matrix** is a mathematical concept that corresponds to a two-dimensional array in programming. For example,

$$A = \begin{bmatrix} 6 & 13 & 7 & 4 \\ 7 & 0 & 8 & 2 \\ 9 & 5 & 4 & 18 \end{bmatrix}$$

is a matrix of size 3×4, i.e., it has 3 rows and 4 columns. The notation $[i, j]$ refers to the element in row i and column j in a matrix. For example, in the above matrix, $A[2,3] = 8$ and $A[3,1] = 9$.

A special case of a matrix is a **vector** that is a one-dimensional matrix of size $n \times 1$. For example,

$$V = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

is a vector that contains three elements.

The **transpose** A^T of a matrix A is obtained when the rows and columns of A are swapped, i.e., $A^T[i, j] = A[j, i]$:

$$A^T = \begin{bmatrix} 6 & 7 & 9 \\ 13 & 0 & 5 \\ 7 & 8 & 4 \\ 4 & 2 & 18 \end{bmatrix}$$

A matrix is a **square matrix** if it has the same number of rows and columns. For example, the following matrix is a square matrix:

$$S = \begin{bmatrix} 3 & 12 & 4 \\ 5 & 9 & 15 \\ 0 & 2 & 4 \end{bmatrix}$$

Operations

The sum $A + B$ of matrices A and B is defined if the matrices are of the same size. The result is a matrix where each element is the sum of the corresponding elements in A and B.

For example,

$$\begin{bmatrix} 6 & 1 & 4 \\ 3 & 9 & 2 \end{bmatrix} + \begin{bmatrix} 4 & 9 & 3 \\ 8 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 6+4 & 1+9 & 4+3 \\ 3+8 & 9+1 & 2+3 \end{bmatrix} = \begin{bmatrix} 10 & 10 & 7 \\ 11 & 10 & 5 \end{bmatrix}.$$

Multiplying a matrix A by a value x means that each element of A is multiplied by x. For example,

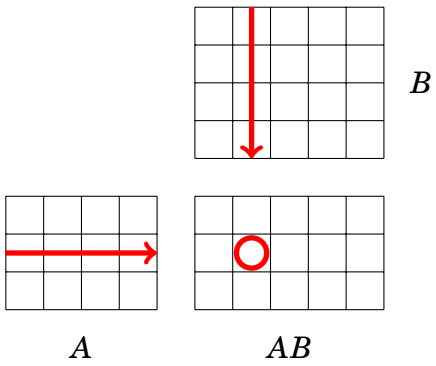
$$2 \cdot \begin{bmatrix} 6 & 1 & 4 \\ 3 & 9 & 2 \end{bmatrix} = \begin{bmatrix} 2 \cdot 6 & 2 \cdot 1 & 2 \cdot 4 \\ 2 \cdot 3 & 2 \cdot 9 & 2 \cdot 2 \end{bmatrix} = \begin{bmatrix} 12 & 2 & 8 \\ 6 & 18 & 4 \end{bmatrix}.$$

Matrix multiplication

The product AB of matrices A and B is defined if A is of size $a \times n$ and B is of size $n \times b$, i.e., the width of A equals the height of B. The result is a matrix of size $a \times b$ whose elements are calculated using the formula

$$AB[i,j] = \sum_{k=1}^{n} A[i,k] \cdot B[k,j].$$

The idea is that each element of AB is a sum of products of elements of A and B according to the following picture:



For example,

$$\begin{bmatrix} 1 & 4 \\ 3 & 9 \\ 8 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 6 \\ 2 & 9 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 4 \cdot 2 & 1 \cdot 6 + 4 \cdot 9 \\ 3 \cdot 1 + 9 \cdot 2 & 3 \cdot 6 + 9 \cdot 9 \\ 8 \cdot 1 + 6 \cdot 2 & 8 \cdot 6 + 6 \cdot 9 \end{bmatrix} = \begin{bmatrix} 9 & 42 \\ 21 & 99 \\ 20 & 102 \end{bmatrix}.$$

Matrix multiplication is associative, so $A(BC) = (AB)C$ holds, but it is not commutative, so $AB = BA$ does not usually hold.

An **identity matrix** is a square matrix where each element on the diagonal is 1 and all other elements are 0. For example, the following matrix is the 3×3 identity matrix:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying a matrix by an identity matrix does not change it. For example,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 4 \\ 3 & 9 \\ 8 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 3 & 9 \\ 8 & 6 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 4 \\ 3 & 9 \\ 8 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 3 & 9 \\ 8 & 6 \end{bmatrix}.$$

Using a straightforward algorithm, we can calculate the product of two $n \times n$ matrices in $O(n^3)$ time. There are also more efficient algorithms for matrix multiplication[1], but they are mostly of theoretical interest and such algorithms are not necessary in competitive programming.

Matrix power

The power A^k of a matrix A is defined if A is a square matrix. The definition is based on matrix multiplication:

$$A^k = \underbrace{A \cdot A \cdot A \cdots A}_{k \text{ times}}$$

For example,

$$\begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix}^3 = \begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 48 & 165 \\ 33 & 114 \end{bmatrix}.$$

In addition, A^0 is an identity matrix. For example,

$$\begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix}^0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The matrix A^k can be efficiently calculated in $O(n^3 \log k)$ time using the algorithm in Chapter 21.2. For example,

$$\begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix}^8 = \begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix}^4 \cdot \begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix}^4.$$

Determinant

The **determinant** $\det(A)$ of a matrix A is defined if A is a square matrix. If A is of size 1×1, then $\det(A) = A[1,1]$. The determinant of a larger matrix is calculated recursively using the formula

$$\det(A) = \sum_{j=1}^{n} A[1,j]C[1,j],$$

where $C[i,j]$ is the **cofactor** of A at $[i,j]$. The cofactor is calculated using the formula

$$C[i,j] = (-1)^{i+j} \det(M[i,j]),$$

[1]The first such algorithm was Strassen's algorithm, published in 1969 [63], whose time complexity is $O(n^{2.80735})$; the best current algorithm [27] works in $O(n^{2.37286})$ time.

where $M[i,j]$ is obtained by removing row i and column j from A. Due to the coefficient $(-1)^{i+j}$ in the cofactor, every other determinant is positive and negative. For example,

$$\det(\begin{bmatrix} 3 & 4 \\ 1 & 6 \end{bmatrix}) = 3 \cdot 6 - 4 \cdot 1 = 14$$

and

$$\det(\begin{bmatrix} 2 & 4 & 3 \\ 5 & 1 & 6 \\ 7 & 2 & 4 \end{bmatrix}) = 2 \cdot \det(\begin{bmatrix} 1 & 6 \\ 2 & 4 \end{bmatrix}) - 4 \cdot \det(\begin{bmatrix} 5 & 6 \\ 7 & 4 \end{bmatrix}) + 3 \cdot \det(\begin{bmatrix} 5 & 1 \\ 7 & 2 \end{bmatrix}) = 81.$$

The determinant of A tells us whether there is an **inverse matrix** A^{-1} such that $A \cdot A^{-1} = I$, where I is an identity matrix. It turns out that A^{-1} exists exactly when $\det(A) \neq 0$, and it can be calculated using the formula

$$A^{-1}[i,j] = \frac{C[j,i]}{det(A)}.$$

For example,

$$\underbrace{\begin{bmatrix} 2 & 4 & 3 \\ 5 & 1 & 6 \\ 7 & 2 & 4 \end{bmatrix}}_{A} \cdot \underbrace{\frac{1}{81}\begin{bmatrix} -8 & -10 & 21 \\ 22 & -13 & 3 \\ 3 & 24 & -18 \end{bmatrix}}_{A^{-1}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{I}.$$

Linear recurrences

A **linear recurrence** is a function $f(n)$ whose initial values are $f(0), f(1), \ldots, f(k-1)$ and larger values are calculated recursively using the formula

$$f(n) = c_1 f(n-1) + c_2 f(n-2) + \ldots + c_k f(n-k),$$

where $c_1, c_2, \ldots, c_k$ are constant coefficients.

Dynamic programming can be used to calculate any value of $f(n)$ in $O(kn)$ time by calculating all values of $f(0), f(1), \ldots, f(n)$ one after another. However, if k is small, it is possible to calculate $f(n)$ much more efficiently in $O(k^3 \log n)$ time using matrix operations.

Fibonacci numbers

A simple example of a linear recurrence is the following function that defines the Fibonacci numbers:

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2) \end{aligned}$$

In this case, $k = 2$ and $c_1 = c_2 = 1$.

To efficiently calculate Fibonacci numbers, we represent the Fibonacci formula as a square matrix X of size 2×2, for which the following holds:

$$X \cdot \begin{bmatrix} f(i) \\ f(i+1) \end{bmatrix} = \begin{bmatrix} f(i+1) \\ f(i+2) \end{bmatrix}$$

Thus, values $f(i)$ and $f(i+1)$ are given as "input" for X, and X calculates values $f(i+1)$ and $f(i+2)$ from them. It turns out that such a matrix is

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

For example,

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} f(5) \\ f(6) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 8 \end{bmatrix} = \begin{bmatrix} 8 \\ 13 \end{bmatrix} = \begin{bmatrix} f(6) \\ f(7) \end{bmatrix}.$$

Thus, we can calculate $f(n)$ using the formula

$$\begin{bmatrix} f(n) \\ f(n+1) \end{bmatrix} = X^n \cdot \begin{bmatrix} f(0) \\ f(1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The value of X^n can be calculated in $O(\log n)$ time, so the value of $f(n)$ can also be calculated in $O(\log n)$ time.

General case

Let us now consider the general case where $f(n)$ is any linear recurrence. Again, our goal is to construct a matrix X for which

$$X \cdot \begin{bmatrix} f(i) \\ f(i+1) \\ \vdots \\ f(i+k-1) \end{bmatrix} = \begin{bmatrix} f(i+1) \\ f(i+2) \\ \vdots \\ f(i+k) \end{bmatrix}.$$

Such a matrix is

$$X = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ c_k & c_{k-1} & c_{k-2} & c_{k-3} & \cdots & c_1 \end{bmatrix}.$$

In the first $k-1$ rows, each element is 0 except that one element is 1. These rows replace $f(i)$ with $f(i+1)$, $f(i+1)$ with $f(i+2)$, and so on. The last row contains the coefficients of the recurrence to calculate the new value $f(i+k)$.

Now, $f(n)$ can be calculated in $O(k^3 \log n)$ time using the formula

$$\begin{bmatrix} f(n) \\ f(n+1) \\ \vdots \\ f(n+k-1) \end{bmatrix} = X^n \cdot \begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(k-1) \end{bmatrix}.$$