线性代数—矩阵、矩阵乘法、广义矩阵乘法

引入

矩阵

一般用圆括号或方括号表示矩阵，形如：

$$A=\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

矩阵表示线性方程组

例如，将线性方程组：

$$\begin{cases} 7x_1+8x_2+9x_3=13 \\ 4x_1+5x_2+6x_3=12 \\ x_1+2x_2+3x_3=11 \end{cases}$$

写成矩阵乘法的形式（将系数抽出来）：

$$\begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 13 \\ 12 \\ 11 \end{pmatrix}$$

简记为：$Ax=b$，其中系数矩阵 $A=\begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$；未知量 $x=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$；常数项 $b=\begin{pmatrix} 13 \\ 12 \\ 11 \end{pmatrix}$.

其本质是：矩阵 A（系数矩阵）左乘一个列向量 x（未知量）等与一个列向量 b（常数项）。

运算

矩阵的线性运算

矩阵的线性运算分为加减法与数乘，它们均为逐个元素进行；

只有同型（规格为 $n \times m$ 与 $n \times m$ 的）矩阵之间可以对应相加减。

例如：$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} + \begin{pmatrix}5 & 6\\7 & 8\end{pmatrix} = \begin{pmatrix}1+5 & 2+6\\3+7 & 4+8\end{pmatrix} = \begin{pmatrix}6 & 8\\10 & 12\end{pmatrix}$.

例如：$3 \times \begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} = \begin{pmatrix}3\times 1 & 3\times 2\\3\times 3 & 3\times 4\end{pmatrix} = \begin{pmatrix}3 & 6\\9 & 12\end{pmatrix}$.

矩阵的转置

矩阵的转置，就是在矩阵的右上角写上转置「T」记号，表示将矩阵的行与列互换。

例如：$\begin{pmatrix}1 & 2 & 3\\3 & 4 & 5\end{pmatrix}^{\mathrm{T}} = \begin{pmatrix}1 & 3\\2 & 4\\3 & 5\end{pmatrix}$

向量内积

对应相乘再相加。

例如：$\begin{pmatrix}1 & 2 & 3\end{pmatrix} \times \begin{pmatrix}4 & 5 & 6\end{pmatrix} = 1\times 4 + 2\times 5 + 3\times 6 = 52$.

矩阵乘法

朴素矩阵乘法

设 A 为 $n \times m$ 的矩阵，B 为 $m \times r$ 的矩阵，即前一矩阵列数等于后一矩阵行数；

设矩阵 $C = A \times B$，则 $C_{i,j} = \sum_{k=1}^{m} A_{i,k}B_{k,j}$。

乘积矩阵中第 i 行第 j 列的数恰好是乘数矩阵 A 第 i 个行向量与乘数矩阵 B 第 j 个列向量的内积，口诀为**左行右列**。

演示网站：`https://rainppr.gitee.io/matrixmultiplication.xyz/`.

矩阵乘法的特殊化——矩阵乘向量

将向量调转，先相乘再相加。

例如：$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}\begin{pmatrix}3\\6\\9\end{pmatrix} = \begin{pmatrix}1\times 3 + 2\times 6 + 3\times 9\\4\times 3 + 5\times 6 + 6\times 9\end{pmatrix} = \begin{pmatrix}42\\96\end{pmatrix}$

矩阵乘法的特殊化——单位矩阵 I

单位矩阵 I：一个方阵（行数 $=$ 列数），只有主对角线（左上、右左下）元素为 1，其他都为 0。

单位矩阵乘任何矩阵都得该矩阵（就像 1 一样），即 $IA = AI = A$。

举例：$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 7 & 8 & 9 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 7 & 8 & 9 & 0 \end{pmatrix}$

广义矩阵乘法

考虑将原公式推广，即广义矩阵乘法：对于矩阵 $A_{n\times m}$ 和 $B_{m\times r}$：

有 $C_{ij} = A \times B = \bigoplus\limits_{k=1}^{m} (A_{ik} \otimes B_{kj})$，我们将其成为 $(\otimes, \oplus)$ 的矩阵乘法。

当满足以下条件时，广义矩阵乘法满足结合律：

- $\otimes$ 具有结合律和交换律；
- $\otimes$ 对 $\oplus$ 存在分配律，即满足 $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$。

常见的矩阵乘法形式有 $(\pm, \max)$、$(\pm, \min)$、$(\wedge, \vee)$。

性质和用途

矩阵乘法满足结合律，不满足一般的交换律，即 $A \times B \not\leq B \times A$。

特殊的，满足以下交换律：
对矩阵加法有结合律，即 $(A+B)C = AC + BC$，$C(A+B) = CA + CB$；
对数乘有结合律，即 $k(AB) = (kA)B = A(kB)$。

利用结合律，矩阵乘法可以利用快速幂的思想来优化；
由于线性递推式可以表示成矩阵乘法的形式，也通常用矩阵快速幂来求线性递推数列的某一项。

详见：https://www.cnblogs.com/RainPPR/p/matrix-dp.html

代码实现

```
const int N = 110;          // 矩阵的最大大小
const int MOD = 1e9 + 7;    // 取模

struct matrix
{
    int n, m, a[N][N];

    // 初始矩阵
    matrix() { memset(a, 0, sizeof a); }
    matrix(int _n, int _m) { n = _n, m = _m, memset(a, 0, sizeof a); }

    // 单位矩阵
    matrix(int _n)
    {
        n = m = _n;
        for (int i = 1; i <= n; ++i)
            a[i][i] = 1;
    }

    // 定义矩阵
    matrix(int _n, int _m, const int t[N][N])
    {
        n = _n, m = _m;
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= m; ++j)
                a[i][j] = t[i][j];
    }

    // 矩阵乘法
    matrix operator*(const matrix &b) const
    {
        matrix res;
        res.n = n, res.m = b.m;
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= b.m; ++j)
                for (int k = 1; k <= m; ++k)
                    res.a[i][j] = (res.a[i][j] + a[i][k] * b.a[k][j] %
MOD) % MOD;
        return res;
    }
};

// 矩阵快速幂
```

```
matrix pow(const int &n, matrix a, int k)
{
    matrix res(n);
    while (k)
    {
        if (k & 1)
            res = res * a;
        k >>= 1, a = a * a;
    }
    return res;
}
```

Reference

[1] http://www.gaosan.com/gaokao/414210.html

[2] https://oi-wiki.org/math/linear-algebra/matrix/

[3] http://matrixmultiplication.xyz/

合集：学习笔记

标签：算法 ，学习笔记