矩阵乘法

河南省实验中学信息技术组

2026 年 05 月 04 日

矩阵概念

- 一个由 $m \times n$ 个数排列成的一个 m 行 n 列的数表

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

被称为一个 m 行 n 列矩阵，也可称为 $m \times n$ 的矩阵。

- 例如：一个 2×3 的矩阵如下：

$$\begin{bmatrix} 4 & 2 & 3 \\ 3 & 1 & 4 \end{bmatrix}$$

矩阵概念

- 只有一行的矩阵被称为行矩阵，又称为行向量。

$$\begin{bmatrix} 4 & 2 & 3 \end{bmatrix}$$

- 只有一列的矩阵被称为列矩阵，又称为列向量。

$$\begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix}$$

- 矩阵行数和列数相同的矩阵被称为方阵。
- 矩阵中所有的数都为 0 的矩阵被称为零矩阵，记为 O。

$$\begin{bmatrix} 4 & 3 & 1 \\ 2 & 0 & 4 \\ 3 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

矩阵概念

- 方阵中，从左上角到右下角的直线被称为该方阵的主对角线，从右上角到左下角的直线被称为副对角线。
- 在方阵中，若主对角线以下的元素全为 0，则称之为上三角形矩阵。反之，若主对角线以上的元素全为 0，则称之为下三角矩阵。
- 方阵中主对角线上的元素均为 1，除此以外全都为 0 的矩阵被称为单位矩阵，记为 E。

$$\begin{bmatrix} 4 & 3 & 1 \\ 0 & 2 & 4 \\ 0 & 0 & 3 \end{bmatrix} \quad \begin{bmatrix} 4 & 0 & 0 \\ 2 & 2 & 0 \\ 3 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

矩阵乘法

河南省实验中学信息技术组

矩阵加法

- 设有两个 $m \times n$ 的矩阵 A, B，那么矩阵 A 与 B 的和记为 $A + B$，规定为

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

- 例如，如下两个矩阵的和

$$\begin{bmatrix} 4 & 2 \\ 3 & 9 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 6 & 5 \\ 4 & 14 \end{bmatrix}$$

- 矩阵加法满足以下运算规律：
  - 交换律：$A + B = B + A$
  - 结合律：$(A + B) + C = A + (B + C)$

矩阵减法

- 设有两个 $m \times n$ 的矩阵 A, B，那么矩阵 A 与 B 的差记为 $A - B$，规定为

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}-b_{11} & a_{12}-b_{12} & \cdots & a_{1n}-b_{1n} \\ a_{21}-b_{21} & a_{22}-b_{22} & \cdots & a_{2n}-b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}-b_{m1} & a_{m2}-b_{m2} & \cdots & a_{mn}-b_{mn} \end{bmatrix}$$

- 例如，如下两个矩阵的差

$$\begin{bmatrix} 4 & 2 \\ 3 & 9 \end{bmatrix} - \begin{bmatrix} 2 & 3 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 2 & 4 \end{bmatrix}$$

矩阵数乘

- 一个数 λ 和矩阵 A 的乘积记为 λA 或 $A\lambda$，规定为

$$\lambda A=\begin{bmatrix}\lambda a_{11} & \lambda a_{12} & \cdots & \lambda a_{1n}\\ \lambda a_{21} & \lambda a_{22} & \cdots & \lambda a_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ \lambda a_{m1} & \lambda a_{m2} & \cdots & \lambda a_{mn}\end{bmatrix}$$

- 例如，如下矩阵乘以 2：

$$2\times\begin{bmatrix}4 & 2\\ 3 & 9\end{bmatrix}=\begin{bmatrix}8 & 4\\ 6 & 18\end{bmatrix}$$

- 矩阵数乘满足以下运算规律：
  - 结合律：$(\lambda\mu A)=\lambda(\mu A)$
  - 分配律：$\lambda(A+B)=\lambda A+\lambda B$

矩阵乘法

- 矩阵乘法略复杂，设 A 是一个 $m \times s$ 的矩阵，B 是一个 $s \times n$ 的矩阵，即

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{ms} \end{bmatrix} B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{s1} & b_{s2} & \cdots & b_{sn} \end{bmatrix}$$

那么，规定 A 与 B 的乘积是一个 $m \times n$ 的矩阵 C，即

$$C = AB = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}$$

其中，$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{is}b_{sj} = \sum_{k=1}^{s} a_{ik}a_{kj}$。

矩阵乘法

- 参与运算的第一个矩阵的列数必须等于第二个矩阵的行数，所得结果矩阵的行数、列数分别为第一个矩阵的行数、第二个矩阵的列数。
- 结果矩阵的第 i 行第 j 列的数是由第一个矩阵的第 i 行 s 个数和第二个矩阵的第 j 列的 s 个数分别相乘再相加得到的。
- 例如，如下两个矩阵相乘

$$\begin{bmatrix} 4 & 2 \\ 3 & 1 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 4 & 2 & 3 & 3 \\ 3 & 1 & 4 & 0 \end{bmatrix} = \begin{bmatrix} 22 & 10 & 20 & 12 \\ 15 & 7 & 13 & 9 \\ 10 & 4 & 11 & 3 \end{bmatrix}$$

- 矩阵乘法满足如下运算律：
  - 结合律：$(AB)C = A(BC)$
  - 分配律：$A(B+C) = AB + AC, (B+C)A = BA + CA$
  - $\lambda AB = (\lambda A)B = A(\lambda B)$
  - 特别地：$AE = EA = A$
  - 矩阵乘法一般不满足交换律，即 $AB \neq BA$。

矩阵乘法的理解

- 对于如下矩阵的乘法

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

它的计算过程如下：

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_n \end{aligned}$$

矩阵乘法的理解

- 上述矩阵乘法可以理解为矩阵 A 将列向量 x 转化为另外一个向量 b。例如，平面直角坐标系中的顶点 $x=(p,q)$ 关于 y 轴对称的点

$$x' = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} -p \\ q \end{bmatrix}$$

- 在算法竞赛中，我们用一个列向量 x 来表示当前状态，那么可以通过将其乘上矩阵 A 来转化为状态 b。
- 因此，对于一些需要进行大量重复状态转移的问题，可以将其转化为矩阵乘法。
- 由于矩阵乘法满足结合律，因此可以使用快速幂来加快矩阵乘法进而加快状态转移。

矩阵快速幂

- 因为矩阵乘法满足结合律，所以求解矩阵的幂可以使用二分快速幂。

$$A^n=\begin{cases} A^{\lfloor \frac{n}{2} \rfloor}\times A^{\lfloor \frac{n}{2} \rfloor}, & n\%2=0 \\ A\times A^{\lfloor \frac{n}{2} \rfloor}\times A^{\lfloor \frac{n}{2} \rfloor}, & n\%2=1 \end{cases}$$

- 如果矩阵 A 是一个 $m\times m$ 的方阵，那么矩阵快速幂的时间复杂度为 $\mathrm{O}(M^3\log N)$。
- 存储矩阵需要大量空间，所以一般矩阵快速幂都使用非递归的形式实现。

矩阵快速幂实现


```
void mul(int a[M][M], int b[M][M])
{
    int c[M][M]; memset(c, 0, sizeof(c));
    for(int i = 1; i <= m; ++i)
        for(int j = 1; j <= m; ++j)
            for(int k = 1; k <= m; ++k)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
    for(int i = 1; i <= m; ++i)
        for(int j = 1; j <= m; ++j)
            a[i][j] = c[i][j];
}

int a[M][M], ans[M][M];
memset(ans, 0, sizeof(ans));
// A^0 = E
for(int i = 1; i <= m; ++i) ans[i][i] = 1;
while(n != 0)
{
    if(n & 1) mul(ans, a);
    mul(a, a);
    n >>= 1;
}
```

【例】斐波那契数列

【题目描述】

在斐波那契数列中，已知 $fib_1 = fib_2 = 1, fib_n = fib_{n-1} + fib_{n-2}(n \geq 3)$。现给定整数 $n(1 \leq n \leq 10^{18})$，求 fib_n 对 $10^9 + 7$ 取余的结果。

【输入格式】

一行一个整数 n。

【输出格式】

一行一个整数，表示数列第 n 项对 $10^9 + 7$ 取余的结果。

【样例 1 输入】

```
4
```

【样例 1 输出】

```
3
```

【样例 2 输入】

```
12345
```

【样例 2 输出】

```
858955650
```

【例】斐波那契数列

- 定义 $F(n)$ 表示一个 2×1 的列向量 $\begin{bmatrix} fib_n \\ fib_{n-1} \end{bmatrix}$，那么 $F(n-1)=\begin{bmatrix} fib_{n-1} \\ fib_{n-2} \end{bmatrix}$，显然有：

$$F(n)=A\times F(n-1)=\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} fib_{n-1} \\ fib_{n-2} \end{bmatrix}=\begin{bmatrix} fib_{n-1}+fib_{n-2} \\ fib_{n-1} \end{bmatrix}=\begin{bmatrix} fib_n \\ fib_{n-1} \end{bmatrix}$$

- 上述是一个矩阵乘法递推式，初值为 $F(1)=\begin{bmatrix} 1 \\ 0 \end{bmatrix}$，则 $F(n)=A^{n-1}\times F(1)$。可以用快速幂计算 $A^{n-1}\times F(1)$(对 10^9+7 取余)，最后得到列向量第 1 行的数字就是答案。
- 时间复杂度为 $O(2^3\log N)$。
- 技巧 *1*：只需要写出目标状态和目标状态的上一个状态，然后用递推公式即可推导出转移矩阵。

【例】斐波那契数列

矩阵乘法

河南省实验中学信息技术组


```cpp
// a*b=b
void mul(long long a[2][2], long long b[2][2])
{
    long long c[2][2]; memset(c, 0, sizeof(c));
    for(int i = 0; i < 2; ++i)
        for(int j = 0; j < 2; ++j)
            for(int k = 0; k < 2; ++k)
                c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % M;
    for(int i = 0; i < 2; ++i)
        for(int j = 0; j < 2; ++j)
            b[i][j] = c[i][j];
}

long long f[2][2] = {{1, 0}, {0, 0}};
long long a[2][2] = {{1, 1}, {1, 0}};
n -= 1;
while(n)
{
    if(n & 1) mul(a, f);
    mul(a, a);
    n >>= 1;
}
printf("%lld", f[1][0]);
```

【例】魔法部落

【题目描述】

小 Biu 所在的部落是一个魔法部落，部落中一共有 $n+1$ 个人，小 Biu 是魔法部落中最菜的，所以他的魔力值为 1，魔法部落中 n 个人的魔法值都不相同，第一个人的魔法值是小 Biu 的 3 倍，第二个人的魔法值是第一个人的 3 倍，以此类推。现在小 Biu 想知道整个部落的魔法值和是多少？由于答案比较大，请把答案对 10^9+7 取模之后输出。

【输入格式】

输入一个数 $n(n \leq 10^9)$，部落里一共有 $n+1$ 个人。

【输出格式】

整个部落的魔法值和，结果对 10^9+7 取余。

【样例输入】

```
40
```

【样例输出】

```
3
```

【例】魔法部落

- 定义 $f_n = 3^0 + 3^1 + \cdots + 3^n$，那么显然有 $f_n = 3 \times f_{n-1} + 1$。
- 如何是定义矩阵呢？矩阵中包含常数 1，如何处理？
- 定义 $F(n)$ 表示一个 2×1 的列向量 $\left[\begin{smallmatrix} f_n \\ 1 \end{smallmatrix}\right]$，则我们用 $F(n-1)$ 可以递推出 $F(n)$ 为：

$$F(n) = A \times F(n-1) = \begin{bmatrix} 3 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} f_{n-1} \\ 1 \end{bmatrix} = \begin{bmatrix} f_n \\ 1 \end{bmatrix}$$

- 上述是一个矩阵乘法递推式，初值为 $F(0) = \left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$，则 $F(n) = A^n \times F(0)$。可以用快速幂计算 $A^n \times F(0)$(对 10^9+7 取余)，最后得到列向量第 1 行的数字就是答案。
- 时间复杂度为 $\mathrm{O}(2^3 \log N)$。
- 技巧 2：使用矩阵乘法加速递推时，遇到常数项时，可以在状态矩阵中添加一个额外位置并令其始终为 1，然后乘上转移矩阵的系数，就可以实现更新状态矩阵的目标。

【例】魔法部落

- 上述转移矩阵 $A = \begin{bmatrix} 3 & 1 \\ 0 & 1 \end{bmatrix}$ 可以理解为 $\begin{bmatrix} 3^1 & 3^0 \\ 0 & 1 \end{bmatrix}$，那么有

$$A^2 = A \times A = \begin{bmatrix} 3^1 & 3^0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3^1 & 3^0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3^2 & 3^0 + 3^1 \\ 0 & 1 \end{bmatrix}$$

那么 $A^n = \begin{bmatrix} 3^n & 1 + 3^1 + \cdots + 3^{n-1} \\ 0 & 1 \end{bmatrix}$，于是 $A^{n+1} = A^n \times A =$

$$\begin{bmatrix} 3^n & 1 + 3^1 + \cdots + 3^{n-1} \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3^1 & 3^0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3^{n+1} & 1 + 3^1 + \cdots + 3^n \\ 0 & 1 \end{bmatrix}$$

- 所以答案为 $A^{n+1}[0][1]$。
- 利用矩阵快速幂，时间复杂度为 $\mathrm{O}(2^3 \log N)$。

【例】魔法部落


```
// 2 × 2 矩阵乘 2 × 2 矩阵
void mul(long long a[2][2], long long b[2][2])
{
    long long c[2][2]; memset(c, 0, sizeof(c));
    for(int i = 0; i < 2; ++i)
        for(int j = 0; j < 2; ++j)
            for(int k = 0; k < 2; ++k)
                c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % M;
    for(int i = 0; i < 2; ++i)
        for(int j = 0; j < 2; ++j)
            a[i][j] = c[i][j];
}

long long a[2][2] = { {3, 1}, {0, 1}};
long long ans[2][2] = { {1, 0}, {0, 1}};
n += 1;
while(n)
{
    if(n & 1) mul(ans, a);
    mul(a, a);
    n >>= 1;
}
printf("%lld", ans[0][1] % M);
```

【例】送给圣诞夜的礼品

圣诞老人让小精灵们准备圣诞礼品。可是由于精神消耗的缘故，他们所做的礼品的质量越来越小，于是圣诞老人把小精灵们聚集起来，说明了自己的看法：“现在你们有 n 个礼品，其质量也就是降序排列的。那么为了使得这个礼品序列保持平均，不像现在这样很有规律的降序，我这里有一个列表。列表共有 m 行，这 m 行都称作操作（不是序列），每一行有 n 个数字，这些数字互不相同而且每个数字都在 1 到 n 之间。一开始，礼品的序列就是现在礼品所处的位置，也就是说，一开始礼品的序列就是 $1, 2, 3 \cdots, n$；那么然后，我们看列表的第一行操作，设这一行操作的第 i 个数字为 $a[i]$，那么就把原来序列中的第 $a[i]$ 个礼物放到现在这个序列的第 i 的位置上，然后组成新的礼物序列。然后，看列表的第二行操作……、第三行操作……一直到最后一行操作，重复上面的操作。当最后一行的操作结束，组成了的序列又按照第一行来操作，然后第二行操作……第三行操作……一直循环下去，直到一共操作了 k 行为止。最后生成的这个序列就是我们最终礼品送给孩子们的序列了。因为 m 值可能很大很大，而小精灵们的操作速度有限。所以可能在圣诞老人去送礼物之前完成不了这个任务。你能帮帮他们吗?

【例】送给圣诞夜的礼品

【输入格式】

第一行三个数，n，m 和 k。
接下来 m 行，每行 n 个数。

【样例输入】

```
7 5 8
6 1 3 7 5 2 4
3 2 4 5 6 7 1
7 1 3 4 5 2 6
5 6 7 3 1 2 4
2 7 3 4 6 1 5
```

【数据范围】

$1 \le n \le 100, 1 \le m \le 10, 1 \le k \le 2^{31} - 1$
对于 50% 的数据，保证 $k \le 500$。

【输出格式】

一行，一共 n 个数，表示最终的礼品序列。

【样例输出】

```
2 4 6 3 5 1 7
```

【例】送给圣诞夜的礼品

- 如果直接模拟，那么时间复杂度为 $O(NK)$，得分 50 分。
- 我们将礼品序列看作一个 n 行的列向量，对于样例而言 $x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}^{\mathrm{T}}$。
- 那么我们将礼物的变换转化为列向量与矩阵相乘。例如，第一行的转换 $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}^{\mathrm{T}} \longrightarrow \begin{bmatrix} 6 & 1 & 3 & 7 & 5 & 2 & 4 \end{bmatrix}^{\mathrm{T}}$，可以理解为：

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \\ 3 \\ 7 \\ 5 \\ 2 \\ 4 \end{bmatrix}$$

【例】送给圣诞夜的礼品

- 因此，可以将这 m 个转换分别描述成矩阵 $A_1, A_2, \cdots, A_m$，然后将它们合并 (相乘) 得到一个矩阵 $A = A_m \cdots A_2 A_1$，然后利用矩阵 A 进行 $\lfloor k/m \rfloor$ 次乘法，此时可以用矩阵快速幂加快速度。
- 剩余 $k\%m$ 次变换，依次乘以前 $k\%m$ 个变换矩阵即可。
- 最后得到的列向量即为要求的礼物顺序。
- 时间复杂度 $\mathrm{O}(N^3 \log \frac{K}{M})$。

【例】送给圣诞夜的礼品

```
// a[0] 存储 m 个转换相乘之后的矩阵
for(int i = 1; i <= n; ++i) a[0][i][i]= 1;
for(int i = 1; i <= m; ++i)
{
    for(int j = 1; j <= n; ++j)
    {
        int x; scanf("%d", &x);
        a[i][j][x] = 1;
    }
    mul(a[i], a[0]);
}
for(int i = 1; i <= n; ++i) f[i][i] = 1, ans[i][1] = i;
int T = k / m;
while(T)
{
    if(T & 1) mul(a[0], f);
    mul(a[0], a[0]);
    T >>= 1;
}
mul(f, ans);
for(int i = 1; i <= k % m; ++i) mul(a[i], ans);
for(int i = 1; i <= n; ++i) printf("%d ", ans[i][1]);
```

【例】石头游戏

石头游戏在一个 n 行 m 列 $(1 \leq n, m \leq 8)$ 的网格上进行，每个格子对应一种操作序列，操作序列至多有 10 种，分别用 $0 \sim 9$ 这 10 个数字指明。

操作序列是一个长度不超过 6 且循环执行、每秒执行一个字符的字符串。每秒钟，所有格子同时执行各自操作序列里的下一个字符。

序列中的每个字符是以下格式之一：

1. 数字 $0 \sim 9$：表示拿 $0 \sim 9$ 个石头到该格子。
2. NWSE：表示把这个格子内所有的石头推到相邻的格子，N 表示上方，W 表示左方，S 表示下方，E 表示右方。
3. D：表示拿走这个格子的所有石头。

给定每种操作序列对应的字符串，以及网格中每个格子对应的操作序列，求石头游戏进行了 t 秒之后，石头最多的格子里有多少个石头。

在游戏开始时，网格是空的。

【例】石头游戏

【输入格式】

第一行 4 个整数 $n, m, t, s (1 \le n, m \le 8, 1 \le t \le 10^8, 1 \le C \le 10)$，分别表示网格大小，游戏进行时间和操作序列种类。

接下来 n 行，每行 m 个字符，分别表示每个格子对应的操作序列 (序列中的字符有 `0~9NWSED`)。

接下来 C 行，每行一个字符串，表示每个操作序列 (编号从 0 开始)。

【输出格式】

一个整数，表示游戏进行 t 秒之后，所有格子中石头最多的格子有多少个石头。

【样例输入】

```
1 6 10 3
011112
1E
E
0
```

【样例输出】

```
3
```

矩阵乘法

河南省实验中学信息技术组

【例】石头游戏

时间	① 1E	② E	③ E	④ E	⑤ E	⑥ 0
第1秒	1	0	0	0	0	0
第2秒	0	1	0	0	0	0
第3秒	1	0	1	0	0	0
第4秒	0	1	0	1	0	0
第5秒	1	0	1	0	1	0
第6秒	0	1	0	1	0	1
第7秒	1	0	1	0	1	1
第8秒	0	1	0	1	0	2
第9秒	1	0	1	0	1	2
第10秒	0	1	0	1	0	3

【例】石头游戏

- 将网格看作长度为 $n \times m$ 的向量，定义 $F_k[p(i,j)]$ 表示 k 秒后格子 (i,j) 中石头的个数，其中 $p(i,j) = (i-1) \times m + j$，表示 (i,j) 在向量中的坐标。
- 特别地，定义 $F_k[0]$ 始终等于 1，显然 $F_0 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}}$(为什么？)。
- 注意到操作序列不超过 6，而 $1 \sim 6$ 的最小公倍数为 60，所以每经过 60 秒，所有操作序列都会重新处于最开始的字符处。
- 对于 $1 \sim 60$ 之间的每个数 k，每个格子在第 k 秒执行的操作可以转化为一个矩阵 A_k，矩阵的构造方法如下：
  - 若 (i,j) 第 k 秒操作的字符为 `'N'` 且 $i > 1$，则令 $A_k[p(i-1,j)][p(i,j)] = 1$，`'W'`,`'S'`,`'E'` 依此类推。
  - 若 (i,j) 第 k 秒操作的字符为 `'D'`，令 $A_k[p(i,j)][p(i,j)] = 0$(无需实现)。
  - 若 (i,j) 第 k 秒操作是一个数字 x，则令 $A_k[p(i,j)][0] = x$，$A_k[p(i,j)][p(i,j)] = 1$，表示格子内原本的石头不动，再拿 x 块石头。
  - 令 $A_k[0][0] = 1$，[1]其余部分均为 0。

[1]此处是为了保证 $F_k[0]$ 始终为 1。

【例】石头游戏

- 令 $T = \lfloor \frac{t}{60} \rfloor$，需要循环 T 轮，剩余的 $t\%60$ 次，单独乘上转移矩阵即可。
- 时间复杂度：$\mathrm{O}(N^3 \log T)$。

```cpp
// a[0] 存储 60 个矩阵之积
for(int i = 0; i <= n * m; ++i) a[0][i][i] = 1;
for(int k = 1; k <= 60; ++k)
{
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= m; ++j)
        {
            int x = op[i][j];  // 执行的操作编号
            int d = (k - 1) % sl[x];   // 第 k 秒进行的操作
            if(s[x][d] >= '0' && s[x][d] <= '9')
                a[k][p[i][j]][0] = s[x][d] - '0', a[k][p[i][j]][p[i][j]] = 1;
            else if(s[x][d] == 'N' && i > 1) a[k][p[i - 1][j]][p[i][j]] = 1;
            else if(s[x][d] == 'W' && j > 1) a[k][p[i][j - 1]][p[i][j]] = 1;
            else if(s[x][d] == 'S' && i < n) a[k][p[i + 1][j]][p[i][j]] = 1;
            else if(s[x][d] == 'E' && j < m) a[k][p[i][j + 1]][p[i][j]] = 1;
        }
    a[k][0][0] = 1;
    mul(a[k], a[0]);
}
```

【例】石头游戏


```
// f 矩阵的第 0 列存储答案
f[0][0] = 1;
int T = t / 60;    // 轮数
while(T)
{
    if(T & 1) mul(a[0], f);
    mul(a[0], a[0]);
    T >>= 1;
}
for(int i = 1; i <= t % 60; ++i) mul(a[i], f);  // 剩余的时间
long long ans = 0;
for(int i = 1; i <= n * m; ++i) ans = max(ans, f[i][0]);
```

小结

- 一般来说，如果一类问题具有以下特点：
  - 可以抽象为一个 n 行的列向量，该向量在每个单位时间发生一次变化。
  - 变化的形式是一个线性递推 (只有若干加法或乘一个系数的运算)。
  - 向量变化时间 (递推轮数) 很长，但向量长度 n 不大。

  那么可以考虑采用矩阵乘法进行优化。我们把这个 n 行的列向量称为*状态矩阵*，与状态矩阵相乘的描述递推的矩阵称为*转移矩阵*。
- 线性递推式中如果有常数项，需要在状态矩阵中增加一个额外位置并令其始终为 1，然后乘上转移矩阵的系数，就可以实现更新状态矩阵的目标。
- 借助二分快速幂加快矩阵乘法后，算法的时间复杂度一般为 $O(N^3 \log T)$，其中 T 为递推次数。

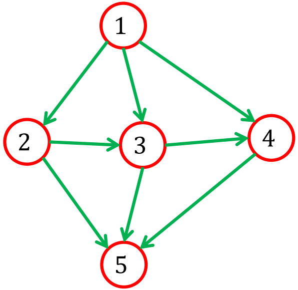
练习

- 矩阵快速幂 (COGS 2469)
- 魔法部落 (COGS 3264)
- 矩阵幂之和 (COGS 2481)
- 斐波那契数列 (COGS 2925)
- 送给圣诞夜的礼物 (COGS 3545)
- 数列求值 (COGS 1163)
- 递推关系 (COGS 1493)
- 233 矩阵 (COGS 2647)
- 石头游戏 (COGS 3433)
- 魔法值 [NOI Online 2020 3rd](COGS 3407)
- 细胞自动机 (COGS 1494)

k 边路径数

给定一个有向图，求任意两点之间经过 k 条边的不同方案数。

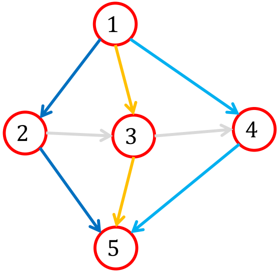
k 边路径数



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

- 右侧矩阵是左侧图的邻接矩阵 G。此时，我们换一种方式来理解它：将 $G[i][j]$ 理解为从顶点 i 到顶点 j经过 1 条边的路径的方案数。

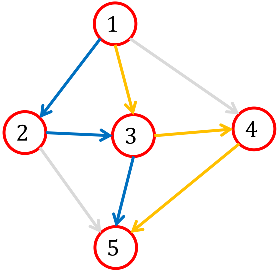
k 边路径数



- 将邻接矩阵 G 相乘得到 $G2 = G \times G$，此时 $G2[1][5]$ 的含义是从顶点 1 出发到顶点 5经过 2 条边的方案数。

$$G \times G = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

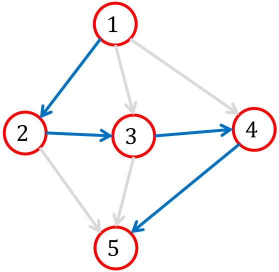
k 边路径数



- 将邻接矩阵 G 相乘得到 $G3 = G^2 \times G$，此时 $G3[1][5]$ 的含义是从顶点 1 出发到顶点 5经过 3 条边的方案数。

$$G^2 \times G = \begin{bmatrix} 0 & 0 & 1 & 1 & 3 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

k 边路径数



- 将邻接矩阵 G 相乘得到 $G4 = G^3 \times G$，此时 $G4[1][5]$ 的含义是从顶点 1 出发到顶点 5经过 4 条边的方案数。

$$G^3 \times G = \begin{bmatrix} 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

k 边路径数

- 设 G 为无权图的邻接矩阵，那么 G^k 得到的矩阵 G' 中的 $G'[i][j]$ 表示从顶点 i 到顶点 j 经过 k 条边的路径数。
- 也可以从动态规划的角度理解，定义 $f(i,j,k)$ 表示从顶点 i 到顶点 j 经过 k 条边的路径数。
- 显然有 $f(i,i,0)=1, f(i,j,1)=G[i][j]$，其余均为 0。
- 状态转移方程：

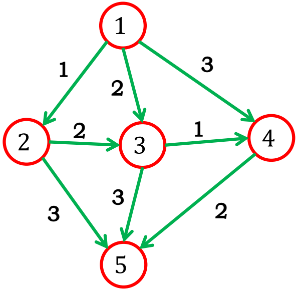
$$f(i,j,k)=\sum_{1\le x\le n} f(i,x,k-1)\times f(x,j,1)=\sum_{1\le x\le n} f(i,x,k-1)\times G[x][j]$$

- 因为第 k 维的状态只与 $k-1$ 维度有关，因此可以使用滚动数组实现。

k 边最短路

给定一个带权有向图，求任意两点之间经过 k 条边的最短路。

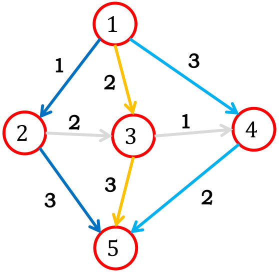
k 边最短路



$$G=\begin{bmatrix}\infty & 1 & 2 & 3 & \infty\\ \infty & \infty & 2 & \infty & 3\\ \infty & \infty & \infty & 1 & 3\\ \infty & \infty & \infty & \infty & 1\\ \infty & \infty & \infty & \infty & \infty\end{bmatrix}$$

- 将邻接矩阵 G 中的 $G[i][j]$ 理解为从顶点 i 到顶点 j经过 1 条边的最短路。
- 对于带权图的邻接矩阵，我们也可以将其邻接矩阵相乘，此时我们将原矩阵乘法进行如下改变
  - 乘法 $\longrightarrow$ 加法
  - 加法 $\longrightarrow$ 取最小值

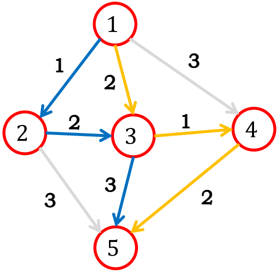
k 边最短路



- 对于带权图的邻接矩阵，我们也可以将其邻接矩阵相乘，此时我们将原矩阵乘法进行如下改变：
  - 乘法 $\longrightarrow$ 加法
  - 加法 $\longrightarrow$ 取最小值
- 此时得到的新矩阵 $G2 = G \otimes G$ 中的 $G2[1][5]$ 表示从顶点 1 到顶点 5经过 2 条边的最短路。

$$G \otimes G = \begin{bmatrix} \infty & 1 & 2 & 3 & \infty \\ \infty & \infty & 2 & \infty & 3 \\ \infty & \infty & \infty & 1 & 3 \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \otimes \begin{bmatrix} \infty & 1 & 2 & 3 & \infty \\ \infty & \infty & 2 & \infty & 3 \\ \infty & \infty & \infty & 1 & 3 \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty & 3 & 3 & 4 \\ \infty & \infty & \infty & 3 & 5 \\ \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

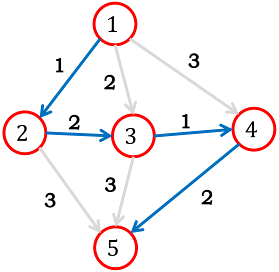
k 边最短路



- 对于带权图的邻接矩阵，我们也可以将其邻接矩阵相乘，此时我们将原矩阵乘法进行如下改变：
  - 乘法 $\longrightarrow$ 加法
  - 加法 $\longrightarrow$ 取最小值
- 此时得到的新矩阵 $G3 = G2 \otimes G$ 中的 $G3[1][5]$ 表示从顶点 1 到顶点 5经过 3 条边的最短路。

$$G2 \otimes G = \begin{bmatrix} \infty & \infty & 3 & 3 & 4 \\ \infty & \infty & \infty & 3 & 5 \\ \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \otimes \begin{bmatrix} \infty & 1 & 2 & 3 & \infty \\ \infty & \infty & 2 & \infty & 3 \\ \infty & \infty & \infty & 1 & 3 \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty & 4 & 5 \\ \infty & \infty & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

k 边最短路



- 对于带权图的邻接矩阵，我们也可以将其邻接矩阵相乘，此时我们将原矩阵乘法进行如下改变：
  - 乘法 $\longrightarrow$ 加法
  - 加法 $\longrightarrow$ 取最小值
- 此时得到的新矩阵 $G4 = G3 \otimes G$ 中的 $G4[1][5]$ 表示从顶点 1 到顶点 5经过 4 条边的最短路。

$$G3 \otimes G = \begin{bmatrix} \infty & \infty & \infty & 4 & 5 \\ \infty & \infty & \infty & \infty & 5 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \otimes \begin{bmatrix} \infty & 1 & 2 & 3 & \infty \\ \infty & \infty & 2 & \infty & 3 \\ \infty & \infty & \infty & 1 & 3 \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

k 边最短路

- 设 G 为带权图的邻接矩阵，将矩阵乘法中的乘法变为加法，加法变为取最小值，那么 G^k 得到的矩阵 G' 中的 $G'[i][j]$ 表示从顶点 i 到顶点 j 经过 k 条边的最短路。
- 也可以从动态规划的角度理解，定义 $f(i,j,k)$ 表示从顶点 i 到顶点 j 经过 k 条边的最短路。
- 显然有 $f(i,i,0)=1, f(i,j,1)=G[i][j]$，其余均为无穷大。
- 状态转移方程：

$$f(i,j,k)=\min_{1\le x\le n} f(i,x,k-1)+f(x,j,1)=\min_{1\le x\le n}\{f(i,x,k-1)+G[x][j]\}$$

- 因为第 k 维的状态只与 $k-1$ 维度有关，因此可以使用滚动数组实现。

练习

- K 边路径数 (COGS 2517)
- K 边最短路 (COGS 2568)
- How many ways?(HDU 2157)
- 冻结 [BJWC 2012](COGS 3414)
- 魔法 [NOI Online 2020 1st](COGS 3377)
- 信使 (COGS 3946)