矩阵与高斯消元

李淳风

长郡中学

2024 年 4 月 13 日

向量

既有大小又有方向的量称为向量。数学上研究的向量为自由向量，即只要不改变它的大小和方向，起点和终点可以任意平行移动的向量。记作 $\vec{a}$ 或 $\mathbf{a}$。

可以把一个长度为 n 的向量 $\vec{a} = (a_1, a_2, \cdots, a_n)$ 看作 n 维空间中，可以平移的有向线段。

例如在平面直角坐标系中，A 的坐标为 (1,1)，B 的坐标为 (3,2)，C 的坐标为 (4,4)，则向量 $\overrightarrow{AB} = (2,1), \overrightarrow{BC} = (1,2), \overrightarrow{AC} = (3,3)$。

向量加法为 $\vec{c} = \vec{a} + \vec{b}$，则 $\forall i \in [1,n], c_i = a_i + b_i$。向量的加法可以理解为有向线段的拼接，也就是 $\overrightarrow{AC} = \overrightarrow{AB} + \overrightarrow{BC}$。

向量的数乘为 $\vec{c} = k\vec{a}$，则 $\forall i \in [1,n], c_i = k * a_i$。向量的数乘可以理解为有向线段的延长。

介绍

一个 n 行 m 列的矩阵可以看作一个 $n \times m$ 的二维数组。
两个矩阵相加/减就是把对应位置上的数相加/减。例如：

$$\begin{bmatrix} 1 & 9 & 8 \\ 3 & 2 & 0 \\ 1 & 8 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 8 & 4 \\ 0 & 5 & 15 \\ 1 & 9 & 6 \end{bmatrix} = \begin{bmatrix} 10 & 17 & 12 \\ 3 & 7 & 15 \\ 2 & 17 & 9 \end{bmatrix}$$

也就是 $C = A \pm B \Leftrightarrow \forall i \in [1, n], \forall j \in [1, m], C_{i,j} = A_{i,j} \pm B_{i,j}$。

矩阵乘法

矩阵乘法就要稍微复杂一些。设 A 是 $n \times m$ 的矩阵，B 是 $m \times p$ 的矩阵，则 $C = A * B$ 是 $n \times p$ 的矩阵，并且有：

$$\forall i \in [1, n], \forall j \in [1, p], C_{i,j} = \sum_{k=1}^{m} A_{i,k} * B_{k,j}$$

矩阵乘法

矩阵乘法就要稍微复杂一些。设 A 是 $n \times m$ 的矩阵，B 是 $m \times p$ 的矩阵，则 $C = A * B$ 是 $n \times p$ 的矩阵，并且有：

$$\forall i \in [1, n], \forall j \in [1, p], C_{i,j} = \sum_{k=1}^{m} A_{i,k} * B_{k,j}$$

也就是说，参与矩阵乘法的两个矩阵 A 和 B，必须满足 A 的列数等于 B 的行数。对于 $C_{i,j}$，实际上就是 A 的第 i 行和 B 的第 j 列，依次相乘后的和。

小练习

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

小练习

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}$$

小练习

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}=\begin{bmatrix}22 & 28\\49 & 64\end{bmatrix}$$

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

小练习

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}=\begin{bmatrix}22 & 28\\ 49 & 64\end{bmatrix}$$

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}=\begin{bmatrix}9 & 12 & 15\\ 19 & 26 & 33\\ 29 & 40 & 51\end{bmatrix}$$

矩阵乘法

需要注意的是，矩阵乘法满足结合律和分配律，但不满足交换律。也就是 $(A*B)*C=(A*B)*C$，$(A+B)*C=A*C+B*C$，但是 $A*B\neq B*A$。

矩阵乘法

需要注意的是，矩阵乘法满足结合律和分配律，但不满足交换律。也就是 $(A*B)*C=(A*B)*C$，$(A+B)*C=A*C+B*C$，但是 $A*B\neq B*A$。
考虑一种特殊的情形，F 是 $1\times n$ 的矩阵，A 是 $n\times n$ 的矩阵，则 $F'=F*A$ 也是 $1\times n$ 的矩阵，并且 $\forall j\in[1,n], F'_j=\sum_{k=1}^{n}F_k*A_{k,j}$。
如果我们把 A 矩阵看作常量，那么可以视为通过对 $F_1,F_2,\cdots,F_n$ 乘上一些系数并相加，来得到 $F'_1,F'_2,\cdots,F'_n$。
所以我们可以把一些递推式写成矩阵乘法的形式，再来优化。

例题

Fibonacci

在斐波那契数列中，
$Fib_0 = 1, Fib_1 = 1, Fib_n = Fib_{n-1} + Fib_{n-2} (n > 1)$。给定整数 n， m $(0 \leq n \leq 2 \times 10^9, m = 10000)$，求 $Fib_n \bmod m$。

例题

Fibonacci

在斐波那契数列中，$Fib_0 = 1, Fib_1 = 1, Fib_n = Fib_{n-1} + Fib_{n-2} (n > 1)$。给定整数 n， m $(0 \leq n \leq 2 \times 10^9, m = 10000)$，求 $Fib_n \bmod m$。

直接递推计算的话，复杂度是 $O(n)$ 的。不过，Fib_n 只与 Fib_{n-1} 和 Fib_{n-2} 有关，我们可以试着把递推式表示成矩阵乘法的形式。
用 F(n) 表示一个 1×2 的矩阵 $[Fib_n \quad Fib_{n+1}]$，我们希望通过 F(n-1) 来计算出 F(n)。因此令 $F(n) = F(n-1) * A$，可以解出 A= $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$，
也就是 $[Fib_n \quad Fib_{n+1}] = [Fib_{n-1} \quad Fib_n] \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$。

例题

Fibonacci

在斐波那契数列中，

$Fib_0 = 1, Fib_1 = 1, Fib_n = Fib_{n-1} + Fib_{n-2}(n > 1)$。给定整数 n， m $(0 \le n \le 2 \times 10^9, m = 10000)$，求 $Fib_n \bmod m$。

既然我们知道了 $F(n) = F(n-1) * A$，而且 $F(0) = \begin{bmatrix} 0 & 1 \end{bmatrix}$，所以我们就可以得到 $F(n) = F(0) * A^n$。

我们做快速幂的时候实际上是运用了结合律，因此可以对矩阵进行快速幂求出 A^n，进而求出 F(n)。

时间复杂度 $O(2^3 \log n)$。

例题

```
int main(){
    int n;
    int f[2]={0,1};
    int a[2][2]={{0,1},{1,1}};
    scanf("%d",&n);
    while(n){
        if(n&1) mul(f,a);
        mulself(a);
        n>>=1;
    }
    printf("%d",f[0]);
    return 0;
}
```

例题

```
int mod=10000;
void mul(int f[2],int a[2][2]){//f*a
    int c[2]={0,0};
    for(int j=0;j<2;j++)
        for(int k=0;k<2;k++){
            c[j]=(c[j]+1ll*f[k]*a[k][j])%mod;
        }
    memcpy(f,c,sizeof(c));
}
void mulself(int a[2][2]){//a*a
    int c[2][2]={{0,0},{0,0}};
    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]+1ll*a[i][k]*a[k][j])%mod;
            }
    memcpy(a,c,sizeof(c));
}
```

矩阵乘法加速递推

通过这道题，大家应该对“矩阵乘法加速递推”有了一些了解。一般来说，如果一类问题具有如下特点：

- 可以抽象为一个长度为 n 的一维向量，该向量在每个单位时间内发生一次变化；
- 变化的形式是一个线性递推（只有若干加法，或者乘上一个常数）；
- 该递推式可能作用于不同的数据，但本身不变；
- 向量变化时间很长（即递推次数很多），但向量本身不大；

那么可以考虑使用矩阵乘法来优化。我们把这个长度为 n 的向量称为“状态矩阵”，用于与状态矩阵相乘的固定不变的矩阵 A 称为“转移矩阵”。若当前状态的第 x 个数对下一时间状态的第 y 个数有影响，则把 $A_{x,y}$ 赋值为对应的系数。

时间复杂度为 $O(n^3 \log T)$。

小练习

$$f_0 = 0, f_1 = 1, f_2 = 1, f_n = 3f_{n-1} + 7f_{n-2} + 5f_{n-3} (n > 2)$$

小练习

$$f_0 = 0, f_1 = 1, f_2 = 1, f_n = 3f_{n-1} + 7f_{n-2} + 5f_{n-3} (n > 2)$$

$$\begin{bmatrix} f_n & f_{n+1} & f_{n+2} \end{bmatrix} \begin{bmatrix} 0 & 0 & 5 \\ 1 & 0 & 7 \\ 0 & 1 & 3 \end{bmatrix} = \begin{bmatrix} f_{n+1} & f_{n+2} & f_{n+3} \end{bmatrix}$$

小练习

$$f_0 = 0, f_1 = 1, f_2 = 1, f_n = 3f_{n-1} + 7f_{n-2} + 5f_{n-3} (n > 2)$$

$$\begin{bmatrix} f_n & f_{n+1} & f_{n+2} \end{bmatrix} \begin{bmatrix} 0 & 0 & 5 \\ 1 & 0 & 7 \\ 0 & 1 & 3 \end{bmatrix} = \begin{bmatrix} f_{n+1} & f_{n+2} & f_{n+3} \end{bmatrix}$$

$$f_0 = 0, f_1 = 1, f_n = 9f_{n-1} + 6f_{n-2} + 5 (n > 1)$$

小练习

$$f_0 = 0, f_1 = 1, f_2 = 1, f_n = 3f_{n-1} + 7f_{n-2} + 5f_{n-3} (n > 2)$$

$$\begin{bmatrix} f_n & f_{n+1} & f_{n+2} \end{bmatrix} \begin{bmatrix} 0 & 0 & 5 \\ 1 & 0 & 7 \\ 0 & 1 & 3 \end{bmatrix} = \begin{bmatrix} f_{n+1} & f_{n+2} & f_{n+3} \end{bmatrix}$$

$$f_0 = 0, f_1 = 1, f_n = 9f_{n-1} + 6f_{n-2} + 5 (n > 1)$$

$$\begin{bmatrix} f_n & f_{n+1} & 1 \end{bmatrix} \begin{bmatrix} 0 & 6 & 0 \\ 1 & 9 & 0 \\ 0 & 5 & 1 \end{bmatrix} = \begin{bmatrix} f_{n+1} & f_{n+2} & 1 \end{bmatrix}$$

高斯消元

高斯消元是一种求解由 M 个 N 元一次方程组成的方程组的方法。我们可以把一个线性方程组写成一个 M 行 N+1 列的增广矩阵：

$$\begin{cases} x_1 + 2x_2 - x_3 = -6 \\ 2x_1 + x_2 - 3x_3 = -9 \\ -x_1 - x_2 + 2x_3 = 7 \end{cases} \Rightarrow \begin{bmatrix} 1 & 2 & -1 & -6 \\ 2 & 1 & -3 & -9 \\ -1 & -1 & 2 & 7 \end{bmatrix}$$

同时，我们定义矩阵的初等行变换：

- 用一个非零的数乘以某一行
- 把其中一行的若干倍加到零一行上
- 交换两行的位置

高斯消元

我们可以对增广矩阵进行若干次初等行变换来求解方程组：

$$\begin{bmatrix} 1 & 2 & -1 & -6 \\ 2 & 1 & -3 & -9 \\ -1 & -1 & 2 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & -1 & -6 \\ 0 & -3 & -1 & 3 \\ -1 & -1 & 2 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & -1 & -6 \\ 0 & -3 & -1 & 3 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 1 & 2 & -1 & -6 \\ 0 & 1 & 1 & 1 \\ 0 & -3 & -1 & 3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & -1 & -6 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & -1 & -6 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

最后一个矩阵的含义实际上是 $\begin{cases} x_1 + 2x_2 - x_3 = -6 \\ x_2 + x_3 = 1 \\ x_3 = 3 \end{cases}$，依次带入回去就完成了求解。

高斯消元

实际上矩阵还可以继续化简:

$$\begin{bmatrix}1 & 2 & -1 & -6\\ 0 & 1 & 1 & 1\\ 0 & 0 & 1 & 3\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 2 & 0 & -3\\ 0 & 1 & 0 & -2\\ 0 & 0 & 1 & 3\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 1 & 0 & -2\\ 0 & 0 & 1 & 3\end{bmatrix}$$

像这样通过初等行变换，把增广矩阵变为简化阶梯型矩阵就是高斯消元算法。该算法的思想是，对于每一个未知量 x_i，找到一个 x_i 系数不为零但是 x_1 x_{i-1} 系数都为零的方程，利用该方程通过初等行变换把其它方程的 x_i 系数全部消成 0。

高斯消元

在高斯消元的过程中，有可能会出现 $0 = d$ 这样的方程，其中 d 是一个非零常量。出现这种情况则意味着方程无解。
其次，有可能找不到 x_i 系数不为零的方程。例如：

$$\begin{bmatrix} 1 & 2 & -1 & 3 \\ 2 & 4 & -8 & 0 \\ -1 & -2 & 6 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & 0 & -6 & -6 \\ 0 & 0 & 5 & 5 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 0 & 4 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

在这个例子中找不到 x_2 系数为 0 的方程，解可以写为 $\begin{cases} x_1 = 4 - 2x_2 \\ x_3 = 1 \end{cases}$

不论 x_2 取任何值，都可以求出对应的 x_1，也就是方程组有无穷多的解。这种情况下，我们把 x_1, x_3 称为主元，把 x_2 称为自由元。
可以发现，在最后的阶梯型系数矩阵中，每个主元仅仅在一个位置上的系数 (i,j) 非零，且第 i 行、第 j 列其他位置都为 0。

高斯消元

```
int n;
double a[21][21];
void Gauss(){//保证有唯一解的情况下进行求解
    for(int i=1;i<=n;i++){
        for(int j=i;j<=n;j++)
            if(fabs(a[j][i])>1e-8){
                for(int k=i;k<=n+1;k++)
                    swap(a[i][k],a[j][k]);
                break;
            }
        for(int j=1;j<=n;j++){
            if(i==j) continue;
            double rate=a[j][i]/a[i][i];
            for(int k=i;k<=n+1;k++) a[j][k]-=a[i][k]*rate;
        }
    }
}
```

逆矩阵

定义 $n \times n$ 的单位矩阵：

$$I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

对于一个 $m \times n$ 的矩阵 A，如果存在一个 $n \times m$ 的矩阵 B 满足 $AB = I_m, BA = I_n$，则称 A 可逆，B 为 A 的逆矩阵，记为 A^{-1}。
实际上，如果 A 可逆，那么 A^{-1} 唯一，且 $n = m$。

逆矩阵

现在给出一个 $n \times n$ 的矩阵 A，请你求出它的逆矩阵 A^{-1}，或者判断 A^{-1} 不存在。

逆矩阵

现在给出一个 $n \times n$ 的矩阵 A，请你求出它的逆矩阵 A^{-1}，或者判断 A^{-1} 不存在。

对于矩阵 $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$，我们把它和 I_3 拼在一起，得到：

$$[A|I] = \left[\begin{array}{ccc|ccc} 2 & -1 & 0 & 1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 1 & 0 \\ 0 & -1 & 2 & 0 & 0 & 1 \end{array}\right]$$

然后通过初等行变换把 A 变为单位矩阵：

$$[I|B] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0.75 & 0.5 & 0.25 \\ 0 & 1 & 0 & 0.5 & 1 & 0.5 \\ 0 & 0 & 1 & 0.25 & 0.5 & 0.75 \end{array}\right]$$

右边的 B 矩阵即为 A^{-1}。

异或方程组

异或方程组指形如 $\begin{cases} a_{1,1}x_1 \oplus a_{1,2}x_2 \cdots a_{1,n}x_n = b_1 \\ a_{2,1}x_1 \oplus a_{2,2}x_2 \cdots a_{2,n}x_n = b_2 \\ \cdots \\ a_{m,1}x_1 \oplus a_{m,2}x_2 \cdots a_{m,n}x_n = b_m \end{cases}$ 的方程组。

异或其实就是不进位的加法，我们仍然可以写出增广矩阵，然后使用高斯消元法。不同的是，每个变量的取值只有 0 和 1，因此若最终有 q 个自由元，方程解的数量为 2^q。

在编写程序时可以使用二进制来进行状态压缩，或是直接使用 bitset 加速。

概念

线性空间是关于向量加法和向量数乘封闭的所有向量的集合。也就是说，如果 $\vec{a}$ 和 $\vec{b}$ 在同一个线性空间中，那么 $\vec{a}+\vec{b}$，$k\vec{a}$ 这些向量都在这一个线性空间中。

概念

线性空间是关于向量加法和向量数乘封闭的所有向量的集合。也就是说，如果 $\vec{a}$ 和 $\vec{b}$ 在同一个线性空间中，那么 $\vec{a}+\vec{b}$，$k\vec{a}$ 这些向量都在这一个线性空间中。
给定若干个向量 $\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$，若 $\vec{b}$ 能由这 k 个向量通过加法或者数乘运算得到，则称向量 $\vec{b}$ 能被向量 $\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$ 表出。
显然向量 $\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$ 能标出的所有向量构成了一个线性空间，$\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$ 被称为这个线性空间的生成子集。

概念

线性空间是关于向量加法和向量数乘封闭的所有向量的集合。也就是说，如果 $\vec{a}$ 和 $\vec{b}$ 在同一个线性空间中，那么 $\vec{a}+\vec{b}$，$k\vec{a}$ 这些向量都在这一个线性空间中。
给定若干个向量 $\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$，若 $\vec{b}$ 能由这 k 个向量通过加法或者数乘运算得到，则称向量 $\vec{b}$ 能被向量 $\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$ 表出。
显然向量 $\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$ 能标出的所有向量构成了一个线性空间，$\vec{a_1}, \vec{a_2}, \cdots, \vec{a_k}$ 被称为这个线性空间的生成子集。
任意选出一个线性空间中的若干个向量，若其中存在一个向量能够被其它向量表出，则称这些向量线性相关，否则称这些向量线性无关。

概念

线性无关的生成子集被称为线性空间的基底，简称基。基还可以被定义为一个线性空间的极大线性无关子集。一个线性空间所有的基的向量个数相等，这个个数也被称为线性空间的维数。
例如，平面直角坐标系中的所有向量构成一个二维线性空间，$(0,1),(1,0)$ 就是它的一个基。$(1,2),(2,1)$，$(0,5),(1,4)$ 都是这个线性空间的基，但是 $(0,5),(0,9)$ 不是。

概念

线性无关的生成子集被称为线性空间的基底，简称基。基还可以被定义为一个线性空间的极大线性无关子集。一个线性空间所有的基的向量个数相等，这个个数也被称为线性空间的维数。
例如，平面直角坐标系中的所有向量构成一个二维线性空间，$(0,1),(1,0)$ 就是它的一个基。$(1,2),(2,1)$，$(0,5),(1,4)$ 都是这个线性空间的基，但是 $(0,5),(0,9)$ 不是。
对于一个 $n \times m$ 的矩阵，我们可以把它的每一行看作一个长度为 m 的向量，称为行向量。矩阵的所有行向量能表出的所有向量构成一个线性空间，这个线性空间的维数被称为矩阵的“行秩”。类似的，我们可以定义列向量和列秩。实际上，矩阵的行秩一定等于列秩，都被称为矩阵的秩。
把这个矩阵进行高斯消元之后，所有非零行向量线性无关，因为高斯消元不改变行向量表出的线性空间。于是，高斯消元之后得到的向量就是该线性空间的一个基，非零向量的数目就是矩阵的秩。

例题

装备购买

小明玩的一款游戏里有 n 件装备，每件装备有 m 个属性，可以用向量 $Z_i = (a_{i,1}, a_{i,2}, \cdots, a_{i,m})$ 来表示，购买每件装备需要花费 c_i。
对于小明来说，如果一件装备能够被已经购买的装备组合出，即假设小明已经买了第 $i_1, i_2, \cdots, i_p$ 这 p 件装备，那么如果存在实数 $b_1, b_2, \cdots, b_p$ 满足 $Z_k = b_1 Z_{i_1} + b_2 Z_{i_2} + \cdots + b_p Z_{i_p}$，那么小明就不会买第 k 件装备。
现在小明想知道他最多能买多少件装备，并且在这个前提下最少花多少钱。

例题

把 n 件装备看作 n 个长度为 m 的向量，最多能买下的装备的数量，就是求这 n 个向量表出的线性空间的基，可以使用高斯消元来求。

例题

把 n 件装备看作 n 个长度为 m 的向量，最多能买下的装备的数量，就是求这 n 个向量表出的线性空间的基，可以使用高斯消元来求。
还要求花费尽量小，实际上就是在高斯消元的过程中，使用贪心策略，对于每个主元 x_i，在所有第 i 列不为零的行向量中选择价格最低的一个。
证明也很简单，使用反证法，假设最优解是 $Z_{i_1}, Z_{i_2}, \cdots, Z_{i_p}$ 这 p 个向量，并且某一个主元价格最低的向量 Z_k 不在其中。那么由于 $Z_{i_1}, Z_{i_2}, \cdots, Z_{i_p}$ 是基，所以 Z_k 能被 $Z_{i_1}, Z_{i_2}, \cdots, Z_{i_p}$ 表出，设 $Z_k = b_1 Z_{i_1} + b_2 Z_{i_2} + \cdots + b_p Z_{i_p}$.
由于 $b_1, b_2, \cdots, b_p$ 不全为 0，不妨设 $b_p \neq 0$，则有 $Z_{i_p} = (Z_k - b_1 Z_{i_1} - \cdots - Z_{i_{p-1}})/b_p$，即 Z_{i_p} 能被 $Z_{i_1}, \cdots, Z_{i_{p-1}}, Z_k$ 表出，这也是一组基，并且代价更小。

异或空间

可以把线性空间的概念推广开来，异或空间就是一个比较常见的形式。我们可以仿照线性空间中的定义，定义线性空间中的“表出”“线性无关”“基”等概念。

异或空间

可以把线性空间的概念推广开来，异或空间就是一个比较常见的形式。
我们可以仿照线性空间中的定义，定义线性空间中的“表出”“线性无关”“基”等概念。
给定 n 个 $0\ 2^m - 1$ 的整数 $a_1, a_2, \cdots, a_n$，我们同样可以使用高斯消元的方法来求这 n 个整数表出的异或空间的基。
还有另一种方法来求异或空间基。我们开一个数组 c 来存储基，对于每个数 a_i，我们从高往低枚举位数 j，如果 a_i 的第 j 位是 1，如果 c_j 不存在，则令 $c_j = a_i$ 并停止扫描，否则令 $a_i = a_i \oplus c_j$ 并继续。
这样实际上也是高斯消元的思想。我们只需要再从后往前做一遍消元，就能得到和高斯消元一样的结果。

高斯消元代码

```
unsigned long long a[100010];
int main(){
    int n;
    int now=1;
    for(int k=63;k>=0;k--){
        for(int i=now;i<=n;i++)
            if(a[i] & (1ull << k)){
                swap(a[now],a[i]);
                break;
            }
            if(!(a[now] & (1ull << k))) continue;
            for(int i=1;i<=n;i++){
                if(i==now) continue;
                if(a[i] & (1ull << k))
                    a[i]^=a[now];
            }
        now++;
    }
}
```

例题

XOR

有 n 个整数 $a_1, a_2, \cdots, a_n$，和 m 个询问，每次询问给出一个正整数 k，询问在 从这 n 个数中若干个数进行异或运算能得到的所有整数构成的集合 中，第 k 小的数是多少。

例题

XOR

有 n 个整数 $a_1, a_2, \cdots, a_n$，和 m 个询问，每次询问给出一个正整数 k，询问在 从这 n 个数中若干个数进行异或运算能得到的所有整数构成的集合 中，第 k 小的数是多少。

使用高斯消元法求出异或空间的基，假设为 $b_1, b_2, \cdots, b_t$，且满足 $b_1 > b_2 > cdots > b_t$。显然从这 t 个数中任意选数进行异或，能得到 2^t 个不同的整数，第 k 小的数就在其中。
之前提到过，高斯消元法在最终的阶梯型系数矩阵中，每个主元仅仅在一个位置上的系数 (i,j) 非零，且第 i 行、第 j 列其他位置都为 0。

例题

因此，在 2^t 个数中，选 b_1 的结果一定比不选 b_1 大，以此类推，除 0 之外，b_t 就是这个异或空间中最小的数。
先假设 0 在异或空间中，我们从 b_1 往 b_t 以此考虑每个数选不选，选不选 b_1 都有 2^{t-1} 种结果，而不选 b_1 的 2^{t-1} 种结果肯定更小。因此我们可以枚举到 b_i 时判断 k 是否大于 2^{t-i}，是则选上 b_i 并把 k 减去 2^{t-i}。这一过程相当于找到最大的不超过 k 的数，所以我们可以直接把 (k-1) 进行二进制分解，如果 (k-1) 的第 i 位为 1，把 b_{t-i} 选上即可。
注意判断 0 是否在线性空间中。

代码

```
unsigned long long getans(unsigned long long k){
    unsigned long long ans=0;
    if(!zero) k++;
    //0 不在异或空间中，把 0 加入异或空间中，再求第 k+1 小
    if(k > 1llu << t) return -1;
    else{
        for(int i=t-1;i>=0;i--)
            if(k > (1ull << i))
                ans^=a[t-i],k-=1ull<<i;
    }
    return ans;
}
```

代码

```
unsigned long long getans(unsigned long long k){
    unsigned long long ans=0;
    if(zero) k--;//如果 0 在异或空间中，把 k-1 进行分解
    //否则相当于把 0 加入异或空间中，再求第 k+1 小，把 k 进行分解
    if(k >= 1llu << t) return -1;
    else{
        for(int i=t-1;i>=0;i--)
            if(k & (1ull << i))
                ans^=a[t-i];
    }
    return ans;
}
```