1.9 数学相关
概率与期望
树上随机游走
min-max容斥
博弈论
Nim Game
Bash Game
Wythoff Game
Fibonacci Game
二分图博弈
删边博弈
SG 函数
线性代数
高斯消元
解实数方程组
解异或方程组
求行列式
树上高斯消元
线性基
基础操作
压位
异或上指定数后的最值
子集异或和第 k 小数
求有多少个子集异或和小于等于 r
两个线性基的并
两个线性基的交
常见应用
可删除的线性基
区间线性基
连通块内任意两点的异或最短路
二阶常系数线性齐次递推式的通项
矩阵的 LU 分解

1.9 数学相关

概率与期望

- **结论1** 设 x 为离散随机变量，且 $x \in \mathbb{N}$，则$E(x)=\sum\limits_{i=1}^{\infty} i \cdot P(x=i)=\sum\limits_{i=1}^{\infty} P(x \geq i)$。
- **Pascal 分布**
  - 成功概率为 p 的实验，成功 r 次，其概率分布为：
    $$P(x=k)=\binom{k-1}{r-1} p^r (1-p)^{k-r}$$
  - 其重复次数的期望为 $\frac{r}{p}$，方差为 $\frac{r(1-p)}{p^2}$。
  - 其失败次数的期望为 $\frac{r(1-p)}{p}$，方差为 $\frac{r(1-p)}{p^2}$。

树上随机游走

- 给定一棵树，从树中的某点 x 出发，设其子树大小为 size_x，结点的度数为 $\deg_x$，每次等概率地选择一条相邻的边游走，$\mathrm{w}(u,v)$ 表示 u 到 v 的边权（距离），son_x 表示其子结点集合，$\mathrm{sibling}_x$ 表示其兄弟结点集合。
- 设其第一次游走到其父结点 fa_x 的期望距离为 f_x，不难得到：

$$f_x=\frac{\mathrm{w}(x,\mathrm{fa}_x)+\sum\limits_{y\in\mathrm{son}_x}(\mathrm{w}(x,y)+f_y+f_x)}{\deg_x}$$

- 移项得：

$$f_x=\mathrm{w}(x,\mathrm{fa}_x)+\sum_{y\in\mathrm{son}_x}\mathrm{w}(x,y)+\sum_{y\in\mathrm{son}_x}f_y$$

- 特别地，若 w 恒为 1，$f_x=2\mathrm{size}_x-1$。
- 类似地，设 g_x 表示从 fa_x 走到 x 的期望距离，不难得到：

$$g_x=\frac{\mathrm{w}(x,\mathrm{fa}_x)+(\mathrm{w}(\mathrm{fa}_x,\mathrm{fa}_{\mathrm{fa}_x})+g_{\mathrm{fa}_x}+g_x)+\sum\limits_{y\in\mathrm{sibling}_x}(\mathrm{w}(\mathrm{fa}_x,y)+f_y+g_x)}{\deg_{\mathrm{fa}_x}}$$

- 化简得 $g_x=g_{\mathrm{fa}_x}+f_{\mathrm{fa}_x}-f_x$，若 w 恒为 1，$g_x=2(n-\mathrm{size}_x)-1$。

min-max容斥

- 常见结论如下，证明过程以顺序编号（1）~（10）指代：

$$\max_{i\in S}x_i=\sum_{T\subseteq S}(-1)^{|T|-1}\min_{i\in T}x_i$$

$$\min_{i\in S}x_i=\sum_{T\subseteq S}(-1)^{|T|-1}\max_{i\in T}x_i$$

$$E(\max_{i\in S}x_i)=\sum_{T\subseteq S}(-1)^{|T|-1}E(\min_{i\in T}x_i)$$

$$E(\min_{i\in S}x_i)=\sum_{T\subseteq S}(-1)^{|T|-1}E(\max_{i\in T}x_i)$$

$$\mathrm{kth}\max_{i\in S}x_i=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}\min_{i\in T}x_i$$

$$\mathrm{kth}\min_{i\in S}x_i=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}\max_{i\in T}x_i$$

$$E(\mathrm{kth}\max_{i\in S}x_i)=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}E(\min_{i\in T}x_i)$$

$$E(\mathrm{kth}\min_{i\in S}x_i)=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}E(\max_{i\in T}x_i)$$

$$\operatorname*{lcm}_{i\in S}x_i=\prod_{T\subseteq S}\left(\gcd_{i\in T}x_i\right)^{(-1)^{|T|-1}}$$

$$\gcd_{i\in S}x_i=\prod_{T\subseteq S}\left(\operatorname*{lcm}_{i\in T}x_i\right)^{(-1)^{|T|-1}}$$

> **证明** 关于（1），可以将取 $\max$ 看作是取值集合取并集，取 $\min$ 看作是取值集合取交集，不难发现就是容斥原理的标准形式。
>
> 关于（2），取取值集合的补集即等同于（1）。
>
> 关于（3）和（4），可以将期望拆成每种选取数的方案乘以其对应取 $\max$ / $\min$ 的结果，因而总可以将和式提到等式外面，即 min-max 容斥在期望的意义下也成立。

> 关于（5）和（6），考虑证明（5），（6）的过程类似。
>
> - 不妨设 $x_i \le x_{i+1}$，
>
> $$\begin{aligned}\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}\min_{j\in T}x_j &= \sum_{i=1}^{|S|}x_i\sum_{j=k-1}^{|S|-i}(-1)^{j+1-k}\binom{|S|-i}{j}\binom{j}{k-1}\\ &= \sum_{i=1}^{|S|}x_i\sum_{j=k-1}^{|S|-i}(-1)^{j-(k-1)}\binom{|S|-i}{k-1}\binom{|S|-i-(k-1)}{j-(k-1)}\\ &= \sum_{i=1}^{|S|}\binom{|S|-i}{k-1}x_i\sum_{j=0}^{|S|-i-(k-1)}(-1)^j\binom{|S|-i-(k-1)}{j}\\ &= \sum_{i=1}^{|S|}\binom{|S|-i}{k-1}x_i[|S|=i+(k-1)]\\ &= x_{|S|-(k-1)} = \operatorname{kth}\max_{i\in S}x_i\end{aligned}$$
>
> 关于（7）和（8），与（3）和（4）同理。
> 关于（9）和（10），将求 gcd 对应于指数上的取 $\min$，求 lcm 对应于指数上的取 $\max$，乘法对应于指数上的加法，不难发现是 min-max 容斥的等价形式。

博弈论

- 待补充。

Nim Game

- 给定 n 堆石子，第 i 堆石子有 a_i 个，两名玩家轮流行动，每次可以任选一堆，取走任意多个石子，但不能不取，取走最后一个石子者获胜。
- **结论1** 两人均采用最优策略，若 $\bigoplus_{i=1}^{n}a_i=0$ 则先手必败，否则先手必胜。

> **证明** 显然末态（全 0 局面）满足先手必败条件，我们只需证明对于任意 $\bigoplus_{i=1}^{n}a_i\neq 0$ 的局面，一定存在一种操作使其能转变为 $\bigoplus_{i=1}^{n}a_i=0$ 的局面。
>
> 设此时 $\bigoplus_{i=1}^{n}a_i=k\neq 0$，$k$ 在二进制下最高位的 1 在第 x 位，则一定有奇数个 a_i 在二进制下第 x 位为 1。找到其中一个 a_i，则 $a_i\oplus k<a_i$，取 $a_i-(a_i\oplus k)$ 个石子，则局面转变为 $\bigoplus_{i=1}^{n}a_i=0$。

- **结论2** 先手必败的玩家总能使接下来的操作每次只取一个石子。

> **证明** 先手必败的玩家在 $\operatorname{lowbit}(a_i)$ 最小的石子堆中取走恰好一个，则先手必胜的玩家也必须在另一堆 $\operatorname{lowbit}(a_i)$ 最小的石子堆中取走恰好一个。

Bash Game

- 给定一堆石子，石子总个数为 n，两名玩家轮流行动，每次至少取 1 个，至多取 m 个，取走最后一个石子者获胜。
- **结论** 两人均采用最优策略，若 $(m+1)|n$ 则先手必败，否则先手必胜。

> **证明** 显然末态（0）满足先手必败条件，对于任意 $(m+1)\not|n$ 的局面，先手取 n 模 $m+1$ 的余数个石子，一定能使其转变为 $(m+1)|n$ 的局面。

Wythoff Game

- 给定两堆石子，石子数分别为 $a, b(a < b)$，两名玩家轮流行动，每次可以在一堆中取石子，或从两堆中取走相同个数的石子，数量不限，取走最后一个石子者获胜。
- **结论** 两人均采用最优策略，若满足 $a = \lfloor \frac{\sqrt{5}+1}{2} k \rfloor, b = a + k, k \in \mathbb{N}$ 则先手必败，否则先手必胜。

> **证明** 待补充。

Fibonacci Game

- 给定一堆石子，石子总个数为 n，两名玩家轮流行动：
  - 先手不能一次取完所有石子，至少取 1 个。
  - 之后每次取的石子数至少为 1，至多为对手取的石子数的 2 倍。
- 取走最后一个石子者获胜。
- **结论** 两人均采用最优策略，若 n 为 Fibonacci 数则先手必败，否则先手必胜。

> **证明** 待补充。

二分图博弈

- 给定一个二分图和起始点，两名玩家轮流行动，每次选择移动到一个与当前点相邻且未经过的点，不能选择点的玩家算输。
- **结论** 两人均采用最优策略，若最大匹配一定包含起始点，则先手必胜，否则先手必败。

> **证明** 若最大匹配一定包含起始点，则先手始终可以沿着匹配边移动，后手不可能选到非匹配点。若后手选到一个非匹配点 P_n，设路径为 $P_1 \to P_2 \to \cdots \to P_n (n\text{为奇数})$，则现有匹配为 $\{P_1 - P_2, \ldots, P_{n-2} - P_{n-1}\}$，则可得到另一个最大匹配 $\{P_2 - P_3, \ldots, P_{n-1} - P_n\}$，与最大匹配一定包含起始点矛盾。
>
> 若最大匹配不一定包含起始点，取其中一个不包含起始点的最大匹配 M，先手每次移动一定会移动到某个匹配点上，否则图中存在增广路，与最大匹配矛盾，后手只需沿匹配边移动即可。

- 先删去该点求最大匹配后，再加上该点及其连边，若能找到增广路则先手必胜，否则先手必败。

删边博弈

- 以下内容摘自《组合游戏略述——浅谈SG游戏的若干拓展及变形》

3.2.1 树的删边游戏

规则如下：

- 给出一个有 N 个点的树，有一个点作为树的根节点。
- 游戏者轮流从树中删去边，删去一条边后，不与根节点相连的部分将被移走。
- 谁无路可走谁输。

我们有如下定理：

[定理]

叶子节点的 SG 值为 0；中间节点的 SG 值为它的所有子节点的 SG 值

加 1 后的异或和。

[证明]（数学归纳法）

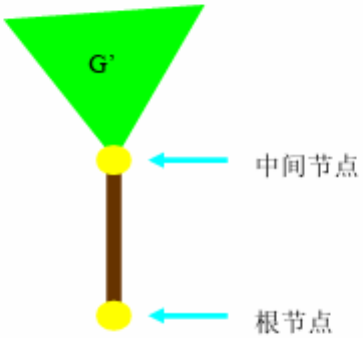
1.一个节点和两个节点的情况显然成立。

2.我们假设小于 K 个节点的任意情况均成立，下面证明（K+1）个节点的情况也成立。我们将证明分成两部分：（1）证明根节点只有一个孩子的情况符合要求；（2）证明根节点有多个孩子的情况符合要求。

先证明第一部分：

我们假设树 G 的根为 A，A 只与 B 相连，图中共有 N+1 各节点，去掉 A 后以 B 为根节点的图为 G’（见图三），下面证明

$$SG(G) = SG(G') + 1$$



G 图

图三

若我们去掉 AB 之间的边，则所有的边都被删除，

∴G 存在 SG 值为 0 的后继局面；　　①

若我们去掉 G 中的除 AB 外的边 E，则删除后总边数小于等于 K。

设以 B 为根的 G’ 去掉 E 以后的后继局面的 SG 值为 P

∵以 A 为根的 G 去掉 E 以后图中的总边数小于等于 K

又∵中间节点的 SG 值为它的所有子节点的 SG 值加 1 后的异或和

∴A 为根的 G 去掉 E 以后的后继局面的 SG 值为 P+1

∵以 B 为根的 G’ 可以通过去边操作使得后继局面的 SG 值变为 0

到 $SG(G')-1$

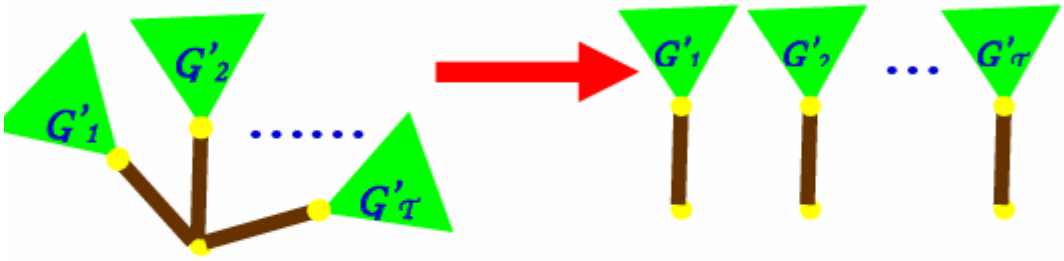
∴以 A 为根的 G 可以通过删除 G’中有的边使得后继局面的 SG 值变为 1 到 $SG(G')$ ②

由①②得， $SG(G)=SG(G')+1$

再证明第二部分：

我们假设树 G 的根为 A，A 与 T 个节点相连，为 B_1、B_2、…、B_T，设去掉(T-1)个与 A 相连的边（AB_X 保留）后，以 A 为根的树为 G_X，第二部分即证明

$$SG(G)=SG(G_1')\oplus SG(G_2')\oplus.....\oplus SG(G_T')$$



图四

聪明的读者可能会发现，我们已经将游戏变成了 NIM 游戏！

G 是总游戏，G_1、G_2、……、G_T 分别是 T 堆石子。

3.2.3 无向图的删边游戏

我们将 Christmas Game 这道题进行一步拓展——去掉对环的限制条件，这个模型应该怎样处理？

无向图的删边游戏：

- 一个无相联通图，有一个点作为图的根。
- 游戏者轮流从图中删去边，删去一条边后，不与根节点相连的部分将被移走。
- 谁无路可走谁输。

对于这个模型，有一个著名的定理——Fusion Principle。

[定理]

我们可以对无向图做如下改动：将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；所有连到原先环上的边全部改为与新点相连。这样的改动不会影响图的 SG 值。

这个原理的证明十分复杂，限于篇幅原因，这里略掉不讲，有兴趣的同学可以参见 Winning Ways，Chapter 7[3]。

这样的话，我们可以将任意一个无向图改成树结构，“无向图的删边游戏”就变成了“树的删边游戏”。

SG 函数

- 在有向图游戏中，规定对于所有出度为 0 的点 x，$SG(x)=0$。
- 对于其余的点 x，设其后继结点分别为 $y_1,y_2,\ldots,y_k$，则 $SG(x)=\operatorname{mex}\{SG(y_i)\},1\le i\le k$。
- $SG(x)=0$ 为必败态，$SG(x)\neq 0$ 为必胜态。
- 定义相互独立的有向图游戏 $G_1,G_2,\ldots,G_m$ 的和 G：
  - 在游戏进行的过程中，游戏者可以任意挑选其中的一个游戏进行决策，最终无法决策者判负。
  - $SG(G)=\bigoplus_{i=1}^{m}SG(G_i)$，证明类似 **Nim Game**。
- 许多博弈题目的 $SG(x)$ 有特殊规律，可以打表得出。
- **注意有向图游戏的定义是无法决策者判负，若题目描述中出现使用某种操作者输，应当认为不能执行这种操作，否则 SG 值的计算会出现错误。**

线性代数

高斯消元

解实数方程组

- 模拟手动消元过程即可。

```
inline bool Gauss()
{
    for (int i = 1; i <= n; ++i)
    {
```

```
        int l = i;
        for (int j = i + 1; j <= n; ++j)
            if (fabs(f[l][i]) < fabs(f[j][i]))
                l = j;
        if (l != i)
            for (int j = 1; j <= n + 1; ++j)
                std::swap(f[l][j], f[i][j]);
        if (fabs(f[i][i]) <= eps)
            return false;
        ld tmp = f[i][i];
        for (int j = 1; j <= n + 1; ++j)
            f[i][j] /= tmp;
        for (int j = 1; j <= n; ++j)
            if (j != i)
            {
                ld tmp = f[j][i];
                for (int k = i; k <= n + 1; ++k)
                    f[j][k] -= f[i][k] * tmp;
            }
    }
    return true;
}
```

解异或方程组

- 大致流程类似，可用 `bitset` 优化。
- 因为系数只有 0/1，题目常常要求计算自由元个数，进而能求出解的组数，注意实现的细节。

```
bitset<N> f[N];

inline int Guass() //返回自由元个数
{
    // n 为未知量个数，m 为方程数
    int ans = 0;
    for (int i = 1; i <= n; ++i)
    {
        int pos = 0;
        for (int j = 1; j <= m; ++j)
            if (!vis[j] && f[j][i])
            {
                pos = j;
                break;
            }
        if (pos)
        {
            if (pos != i)
                swap(f[pos], f[i]);
            vis[i] = true;
            for (int j = 1; j <= m; ++j)
                if (!vis[j] && f[j][i]) f[j] ^= f[i];
        }
        else
            ++ans; //统计自由元个数
    }

    for (int i = 1; i <= m; ++i)
    {
```

```
        bool flag = false;
        for (int j = 1; j <= n; ++j)
            if (f[i][j])
            {
                flag = true;
                break;
            }
        if (!flag && f[i][n + 1]) //无解情况
            return -1;
    }
    return ans;
}
```

求行列式

- 消成上三角矩阵直接计算即可。
- 若模数为质数，可以直接求逆元，否则消元的过程可通过辗转相除法实现。
- $f_{i,i} > 0$ 时每次迭代至少减少一半，实际的时间复杂度为 $\mathcal{O}(n^2 \log w + n^3)$，同样的原理也可以用于分析区间 $\gcd$，实际上合并线段树上 $\mathcal{O}(\log n)$ 个区间的答案总迭代次数也是 $\mathcal{O}(\log w)$。

```
inline int Det()
{
    int res = 1, opt = 0;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = i + 1; j <= n; ++j)
        {
            while (f[i][i])
            {
                int tmp = f[j][i] / f[i][i];
                for (int k = i; k <= n; ++k)
                    dec(f[j][k], 1ll * tmp * f[i][k] % mod);
                std::swap(f[i], f[j]);
                opt ^= 1;
            }
            std::swap(f[i], f[j]);
            opt ^= 1;
        }
        res = 1ll * res * f[i][i] % mod;
        if (!res)
            return 0;
    }
    return opt ? mod - res : res;
}
```

树上高斯消元

- 若方程与树形结构相关，即每个方程中只出现某个结点以及其相邻结点所代表的变量，则我们可以通过**待定系数法**，将消元的时间复杂度降至 $\mathcal{O}(n)$。
- 例如给定一个树上结点的子集 T，要求第一次走到 T 中某一个结点的期望时间，不妨设 f_x 表示从 x 出发，第一次走到 T 中某个结点的期望时间，不难得到转移：
  - 若 $x \in T$，$f_x = 0$。
  - 若 $x \notin T$，$f_x = 1 + \frac{1}{\deg_x} \sum\limits_{(x,y)} f_y$。

- 对于叶子结点，f_x 要么为 0，要么只和其父结点的 f_{fa_x} 有关，因此总可以将 f_x 表示为 $f_x = A_x + B_x f_{\text{fa}_x}$ 的形式，其中 A_x 和 B_x 已知。
- 对于 f 非 0 的非叶子结点，容易得到：

$$\deg_x f_x = \deg_x + \sum_{y \in \text{son}_x} A_y + f_x \sum_{y \in \text{son}_x} B_y + f_{\text{fa}_x}$$

$$f_x = \frac{f_{\text{fa}_x}}{\deg_x - \sum\limits_{y \in \text{son}_x} B_y} + \frac{\deg_x + \sum\limits_{y \in \text{son}_x} A_y}{\deg_x - \sum\limits_{y \in \text{son}_x} B_y}$$

- 所以有：

$$A_x = \frac{\deg_x + \sum\limits_{y \in \text{son}_x} A_y}{\deg_x - \sum\limits_{y \in \text{son}_x} B_y},\ B_x = \frac{1}{\deg_x - \sum\limits_{y \in \text{son}_x} B_y}$$

- 推导到根节点 root 时就可以直接算出 f_{root}，再倒代入每个结点 f_x 的表达式就可以解出所有 f_x。

线性基

- 线性基是一种特殊的基，它具有如下性质：
  - 原集合任意非空子集得到的异或和都能由线性基内某个唯一的非空子集异或得到。
  - 线性基是满足上述性质最小的集合，且没有异或和为 0 的子集。
  - 线性基中每个元素的二进制最高位互不相同。
- 由性质 1 和性质 2，设线性基内元素个数为 cnt，则该线性基子集的异或和种数为 2^{cnt}，**若插入线性基的元素个数为 n，（可为空）子集异或和为 0 的方案数为 2^{n-cnt}，非空子集异或和为 0 的方案数为 $2^{n-cnt}-1$。**
- 由性质 2，若需要判断原集合能否找出一个非空子集异或和为 0，构造其线性基时需要特殊记录（代码中的 `flag`）。

基础操作

- 只作简要叙述，具体实现见代码。

压位

- 若线性基表示的数字较大，可通过重载 `bitset` 进行压位优化。

```
struct bigint
{
    bitset<N> x;

    inline void Clear() {x.reset();}

    inline void scan(char *s, int n)
    {
        x.reset();
        std::reverse(s, s + n);
        for (int i = 0; i < n; ++i)
            if (s[i] == '1')
                x[N - 1 - i] = 1;
    }

    inline bool operator < (const bigint &a) const
    {
```

```
        int p = (x ^ a.x)._Find_first();
        if (p == N)
            return false;
        return !x[p];
    }

    inline bool operator <= (const bigint &a) const
    {
        int p = (x ^ a.x)._Find_first();
        return p == N || !x[p];
    }
}
```

异或上指定数后的最值

- 从高位到低位贪心即可。

子集异或和第 k 小数

- 我们先对构建出的线性基做一次 `reBuild` 操作，具体来说，从低位到高位，尽量用低位的元素消去高位元素上的 1，只保留线性基中非 0 的元素。
- 这样显然不会破坏线性基的性质，同时我们能保证对于线性基内任意一个子集的异或和以及任意一个不在该子集内的元素，异或上该元素都能使该子集的异或和增大。
- 之后从高位到低位贪心即可。

求有多少个子集异或和小于等于 r

- 同样先做一次 `reBuild` 操作，只保留线性基中非 0 的元素。
- 初始答案为 1（$r \geq 0$，而空集一定能够取到 0），从高位到低位贪心，若异或上当前元素仍小于等于 r，则异或上当前元素，将答案加上 $2^{\text{线性基中比该数小的非0元素个数}}$。
- 设线性基内元素个数为 cnt，原集合的元素个数为 n，若要求有多少个原集合的子集异或和小于等于 r，将上述询问的答案乘上 2^{n-cnt} 即可。

两个线性基的并

- 暴力将其中一个线性基的每个元素插入另一个线性基即可。

两个线性基的交

- 给定两个线性基 a, b，求出一个线性基 c，使得其任意子集的异或和在 a, b 中均能由子集异或和表示。
- 记线性基 a, b 中元素分别为 $a_0, a_1, \ldots, a_L$ 和 $b_0, b_1, \ldots, b_L$，对于 c 中某一子集的异或和

$$x = \bigoplus_{i=1}^{n} a_{p_i} = \bigoplus_{i=1}^{m} b_{q_i}$$

- 其中 $0 \leq p_1 < p_2 < \cdots < p_n \leq L, 0 \leq q_1 < q_2 < \cdots < q_m \leq L$。
- 考虑枚举从高位到低位枚举 b_{q_1}，用 res 维护 a 和 $b_{q_1+1}, b_{q_1+2}, \ldots, b_L$ 组成的线性基的并，同时维护 res 中每个元素包含了 a 中的哪些元素。
- 当插入 b_{q_1} 时，若 b_{q_1} 能由 res 的某个子集异或和表示，则我们通过维护的信息可唯一确定 x，将其插入 c 即可。

```
const int L = 60;
ll cur[L + 1];
int cm;

```

```cpp
struct lib
{
    ll g[L + 1];
    bool flag0;

    inline void Clear()
    {
        flag0 = false;
        for (int i = 0; i <= L; ++i)
            g[i] = 0;
    }

    inline bool Insert(ll v)
    {
        for (int i = L; i >= 0; --i)
            if (v >> i & 1)
            {
                if (g[i])
                    v ^= g[i];
                else
                {
                    g[i] = v;
                    return true;
                }
            }
        flag0 = true;
        return false;
    }

    inline int queryCnt()
    {
        int cnt = 0;
        for (int i = 0; i <= L; ++i)
            cnt += g[i] > 0;
        return cnt;
    }

    inline ll queryMax(ll res = 0)
    {
        for (int i = L; i >= 0; --i)
            if ((res ^ g[i]) > res)
                res ^= g[i];
        return res;
    }

    inline ll queryMin(ll res = -1)
    { // res = -1 时表示询问线性基内子集异或和的最小值
        if (res == -1)
        {
            if (flag0)
                return 0;
            for (int i = 0; i <= L; ++i)
                if (g[i])
                    return g[i];
        }
        else
        {
            for (int i = L; i >= 0; --i)
```

```
                if (res >> i & 1)
                    res ^= g[i];
            return res;
        }
    }

    inline void reBuild()
    { // 使用 queryKth() 之前记得调用
        cm = 0;
        for (int i = 0; i <= L; ++i)
            if (g[i])
            {
                for (int j = 0; j < i; ++j)
                    if (g[i] >> j & 1)
                        g[i] ^= g[j];
                cur[cm++] = g[i];
            }
    }

    inline ll queryKth(ll k)
    {
        if (flag0)
            --k;
        if (!k)
            return 0;
        if (k >= (1ll << cm))
            return -1;
        ll res = 0;
        for (int i = 0; i < cm; ++i)
            if (k >> i & 1)
                res ^= cur[i];
        return res;
    }

    inline ll queryRank(ll r)
    {
        ll res = 0, cnt = 1;
        for (int i = cm - 1; i >= 0; --i)
        {
            ll temp = res ^ cur[i];
            if (temp <= r)
            {
                res = temp;
                cnt += 1ll << i;
            }
        }
        return cnt;
    }

    friend inline lib operator | (const lib &a, const lib &b)
    {
        lib c = a;
        for (int i = L; i >= 0; --i)
            if (b.g[i])
                c.Insert(b.g[i]);
        c.flag0 |= b.flag0;
        return c;
    }
```

```
121
122     friend inline lib operator & (const lib &a, const lib &b)
123     {
124         lib c, d, res;
125         c.flag0 = a.flag0 && b.flag0;
126         for (int i = L; i >= 0; --i)
127         {
128             res.g[i] = a.g[i];
129             d.g[i] = 1ll << i;
130         }
131         for (int i = L; i >= 0; --i)
132             if (b.g[i])
133             {
134                 ll v = b.g[i], k = 0;
135                 bool flag = true;
136                 for (int j = L; j >= 0; --j)
137                     if (v >> j & 1)
138                     {
139                         if (res.g[j])
140                         {
141                             v ^= res.g[j];
142                             k ^= d.g[j];
143                         }
144                         else
145                         {
146                             flag = false;
147                             res.g[j] = v;
148                             d.g[j] = k;
149                             break ;
150                         }
151                     }
152                 if (flag)
153                 {
154                     v = 0;
155                     for (int j = L; j >= 0; --j)
156                         if (k >> j & 1)
157                             v ^= a.g[j];
158                     c.Insert(v);
159                 }
160             }
161         return c;
162     }
163 };
```

常见应用

可删除的线性基

- 强行实现可删除的线性基需要记录原集合的信息，较为麻烦且复杂度不科学，这里略去。
- 线性基的撤回操作实现较为容易，与线段树分治结合即可实现离线的可删除。

区间线性基

- 给定序列 a，多次询问 $[l,r]$ 的元素构成的线性基。
  - 显然可以得到线段树或 ST 表的做法。
  - 实际上还有更简便的做法，设 s_i 表示 $[1,i]$ 的元素构成的线性基，同时记录 $p_{i,j}$ 表示 s_i 中第 j 位的元素由 a 中哪个元素构成。构建 s_i 时先令 $s_i=s_{i-1},p_{i,j}=p_{i-1,j}$，若 a_i 能由 s_i 中的

元素表示，则删除表示 a_i 的元素中 $p_{i,j}$ 最小的，并由 a_i 顶替。
    - 询问时只需取出 s_i 中 $p_{i,j} \geq l$ 的元素构建线性基即可。

```cpp
inline bool Insert(int t, ll v)
{
    s[t] = s[t - 1];
    p[t] = p[t - 1];
    ll tmp = t;
    for (int i = L; i >= 0; --i)
        if (v >> i & 1)
        {
            if (p[t].g[i] < tmp)
            {
                std::swap(s[t].g[i], v);
                std::swap(p[t].g[i], tmp);
            }
            v ^= s[t].g[i];
        }
        else
        {
            s[t].g[i] = v;
            p[t].g[i] = tmp;
            return true;
        }
    s[t].flag0 = true;
    return false;
}
```

- 给定序列 a，多次操作，询问 $[l,r]$ 的元素构成的线性基或区间将 $[l,r]$ 异或上一个数。
    - 设 $b_i = a_i \oplus a_{i-1}$，用线段树维护序列 b 的区间线性基，则区间操作可转化为单点修改，同时还需一棵线段树维护序列 a，支持单点查询。
- 给定一棵带点权的树，多次询问两点 (x,y) 路径上点权构成的线性基。
    - 倍增求出 x,y 的 LCA，设为 z，设 $anc_{x,i}$ 表示 x 往上跳 2^i 步得到的结点，预处理 $f_{x,i}$ 表示 $anc_{x,i} \to x$ 这条路径上（不包括 $anc_{x,i}$）的点权构成的线性基。
    - 运用 ST 表 的思想，将 $z \to x$ 和 $z \to y$ 两条路径分别处理，将每条路径的线性基拆成两个可重叠且长度为 2 的整数次幂的路径的线性基的并。
    - 设点数为 n，询问数为 q，可做到时间复杂度 $\mathcal{O}(n \log n \log^2 w + q \log^2 w + q \log n)$。

连通块内任意两点的异或最短路

- 任取连通块中的一棵生成树，对于任意一条非树边，取其与树边构成的环的异或和，则任意两点的异或最短路为它们在生成树上路径的异或和与若干个环的异或和异或的最小值，用线性基维护所有环的异或和即可。
- 若要动态加边和删边可与线段树分治、并查集结合，在任意两点间连边可根据异或的性质转换成在它们并查集祖先间连边。

二阶常系数线性齐次递推式的通项

- 给出递推式 $f_n = af_{n-1} + bf_{n-2}$，已知 f_0, f_1，求 f_n 的通项公式。
- **结论** 设 x_1, x_2 为方程 $x^2 - ax - b = 0$ 的两根（可以为复数）。
    - 若 $x_1 \neq x_2$，$f_n = Ax_1^n + Bx_2^n$。
    - 若 $x_1 = x_2$，$f_n = (A + Bn)x_1^n$。
    - 其中 A, B 可以根据 f_0, f_1 列方程组解出。

证明 （实际可用基础的生成函数证明，以下是另一种证明方式）

尝试把递推式表示成等比数列的形式：

$$\begin{aligned} f_n - x_1 f_{n-1} &= x_2(f_{n-1} - x_1 f_{n-2}) \\ f_n &= (x_1 + x_2) f_{n-1} - x_1 x_2 f_{n-2} \end{aligned}$$

可以列出一个方程组：

$$\begin{cases} x_1 + x_2 = a \\ x_1 x_2 = -b \end{cases}$$

由韦达定理得，x_1, x_2 是方程 $x^2 - ax - b = 0$ 的两根。

设 $c = f_1 - x_1 f_0$，可列出 n 个方程：

$$\begin{cases} f_1 - x_1 f_0 & = c \\ f_2 - x_1 f_1 & = cx_2 \\ f_3 - x_1 f_2 & = cx_2^2 \\ & \cdots \\ f_n - x_1 f_{n-1} & = cx_2^{n-1} \end{cases}$$

将第 i 个方程乘上 x_1^{n-i}，然后将所有方程相加，得到：

$$f_n - x_1^n f_0 = cx_1^{n-1} \sum_{i=0}^{n-1} \left(\frac{x_2}{x_1} \right)^i$$

若 $x_1 = x_2$，此时 $A = f_0, B = \frac{c}{x_1}$：

$$\begin{aligned} f_n - x_1^n f_0 &= cnx_1^{n-1} \\ f_n &= cnx_1^{n-1} + x_1^n f_0 \\ f_n &= \left(f_0 + \frac{c}{x_1} n \right) x_1^n \end{aligned}$$

若 $x_1 \neq x_2$，由等比数列求和，$A = f_0 - \frac{c}{x_2 - x_1}, B = \frac{c}{x_2 - x_1}$：

$$\begin{aligned} f_n - x_1^n f_0 &= cx_1^{n-1} \frac{\left(\frac{x_2}{x_1} \right)^n - 1}{\frac{x_2}{x_1} - 1} \\ f_n &= c \frac{x_2^n - x_1^n}{x_2 - x_1} + x_1^n f_0 \\ f_n &= \left(f_0 - \frac{c}{x_2 - x_1} \right) x_1^n + \frac{c}{x_2 - x_1} x_2^n \end{aligned}$$

矩阵的 LU 分解

- **定义** 对于 n 阶矩阵 A，若存在一个下三角矩阵 L 和一个上三角矩阵 U，使得 $A = LU$，则称 $A = LU$ 为 A 的 LU 分解。
- **定理** 若 n 阶矩阵 A 的前 $n-1$ 个顺序主子式 $D_k \neq 0 (1 \leq k \leq n-1)$，则 A 可以**唯一**地分解成一个**单位**下三角矩阵 L 和一个上三角矩阵 U。

证明 先证明此时一定存在 A 的分解方案。

- 记 A_k 表示 A 前 k 行 k 列构成的子矩阵，则 $D_k = \det(A_k)$。
- 由行列式的定义可知，若我们每次通过第三种初等行变换将 $A_k (1 \leq k \leq n-1)$ 化为上三角矩阵，则 A_k 对角线上的元素一定不为 0。
- 因此，通过第三种初等行变换我们一定能将 A 化为上三角矩阵 U，则有：

$$P_l P_{l-1} \dots P_1 A = U \Leftrightarrow A = (P_l P_{l-1} \dots P_1)^{-1}$$

- 取 $L=(P_lP_{l-1}\dots P_1)^{-1}=P_1^{-1}\dots P_{l-1}^{-1}P_l^{-1}$，由于第三种初等行变换对应的初等矩阵及其逆均为单位下三角矩阵，故 L 也一定也为单位下三角矩阵，则我们构造出了一组 A 的分解方案。
- 再证明分解方案的唯一性，分以下两种情况讨论：
  - 若 A 可逆，
    - 假设分解不唯一，即 $A=L_1U_1=L_2U_2$，则 L_1,U_1,L_2,U_2 均可逆，则有 $U_1U_2^{-1}=L_1^{-1}L_2$，由于 $U_1U_2^{-1}$ 是上三角矩阵，$L_1^{-1}L_2$ 是单位下三角矩阵，有 $U_1U_2^{-1}=L_1^{-1}L_2=E$，所以 $L_1=L_2,U_1=U_2$，与假设矛盾。
  - 若 A 不可逆，
    - 由分块矩阵相关理论，可知：

$$A=\begin{pmatrix}A_{n-1}&\alpha\\\beta^T&a_{n,n}\end{pmatrix}=\begin{pmatrix}L_{n-1}&0\\l^T&1\end{pmatrix}\begin{pmatrix}U_{n-1}&u\\0^T&0\end{pmatrix}$$

    - 因而可得到以下关系式：

$$\begin{aligned}A_{n-1}&=L_{n-1}U_{n-1}\\\alpha=L_{n-1}u&\Leftrightarrow u=L_{n-1}^{-1}\alpha\\\beta^T=l^TU_{n-1}&\Leftrightarrow l^T=\beta^TU_{n-1}^{-1}\end{aligned}$$

    - 由可逆情况的证明，L_{n-1},U_{n-1} 是唯一的，故 u,l^T 也可唯一确定，该情况得证。

- 下面给出 L,U 的计算方法，显然有 $l_{i,j}=u_{j,i}=0(j>i),l_{i,i}=1$。
  - 当 $i\le j$ 时，由 $a_{i,j}=\sum\limits_{k=1}^{i}l_{i,k}u_{k,j}=\sum\limits_{k=1}^{i-1}l_{i,k}u_{k,j}+u_{i,j}$，得 $u_{i,j}=a_{i,j}-\sum\limits_{k=1}^{i-1}l_{i,k}u_{k,j}$，特别地，当 $i=1$ 时，有 $u_{i,j}=a_{i,j}$。
  - 当 $i>j$ 时，由 $a_{i,j}=\sum\limits_{k=1}^{j}l_{i,k}u_{k,j}=\sum\limits_{k=1}^{j-1}l_{i,k}u_{k,j}+l_{i,j}u_{j,j}$，得 $l_{i,j}=\frac{a_{i,j}-\sum\limits_{k=1}^{j-1}l_{i,k}u_{k,j}}{u_{j,j}}$，特别地，当 $j=1$ 时，有 $l_{i,j}=\frac{a_{i,j}}{u_{j,j}}$。
- 套用上式递推计算即可，时间复杂度 $\mathcal{O}(n^3)$。

```cpp
inline void LU_Factorization(double (*a)[N], int n)
{
    double b[N][N]; //由于 L,U 均为三角矩阵，可将两者同时存于一个矩阵中
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            if (i <= j)
            {
                b[i][j] = a[i][j];
                for (int k = 1; k < i; ++k)
                    b[i][j] -= b[i][k] * b[k][j];
            }
            else
            {
                b[i][j] = a[i][j];
                for (int k = 1; k < j; ++k)
                    b[i][j] -= b[i][k] * b[k][j];
                if (fabs(b[j][j]) <= eps)
                {
                    puts("No Solution");
                    return ;
                }
                b[i][j] /= b[j][j];
            }
```

```
24 }
```