Standard Code Library

Part1 - Math

Jiangxi Normal University

HeartFireY, eroengine, yezzz

November 19, 2022

Standard Code Library

HeartFireY, eroengine, yezzz

Jiangxi Normal University

November 19, 2022

Contents

数学 **2**

1. 公式速查 . . . 2
   - 一些数论公式 . . . 2
   - 一些数论函数求和的例子 . . . 2
   - 斐波那契数列性质 . . . 2
   - 常见生成函数 . . . 2
   - 佩尔方程 . . . 3
   - Burnside & Polya . . . 3
   - 皮克定理 . . . 3
   - 莫比乌斯反演 . . . 3
   - 低阶等幂求和 . . . 3
   - 一些组合公式 . . . 4
   - 类欧几里得 . . . 4
   - 欧拉常数 . . . 4
   - $Cayley$ 公式 . . . 4
   - $Catalan$ 数列适用的题型: . . . 4
   - C++ STL 常量 . . . 4
2. 快速幂/龟速乘 . . . 5
3. 筛 (素数筛，积性筛) . . . 5
   - 欧拉筛 . . . 5
   - 欧拉筛 + 莫比乌斯函数 . . . 5
   - 欧拉筛 + 欧拉函数 . . . 5
   - 欧拉函数 . . . 6
   - 筛法求约数和 . . . 6
   - 杜教筛 . . . 6
4. 素数测试 . . . 7
5. 质因数分解 . . . 8
   - Pollard-Rho . . . 8
   - 朴素质因数分解 (带指数)(前置素数筛) . . . 8
   - 朴素质因数分解 (不带指数) . . . 8
5. 矩阵运算类 . . . 9
6. 高斯消元 . . . 9
7. 线性基 . . . 10
8. 扩展欧几里得 . . . 11
   - 扩展欧几里得 . . . 11
   - 扩欧逆元 . . . 11
9. 中国剩余定理 (exGCD) . . . 11
10. 二次剩余 . . . 12
11. 伯努利数和等幂求和 . . . 12
12. 数论分块/整除分块 . . . 13
13. 斯特林数 . . . 13
   - 第一类斯特林数 . . . 13
   - 第二类斯特林数 . . . 13
   - **第二类斯特林数**（斯特林子集数） . . . 13
   - 第一类斯特林数（Stirling Number） . . . 15

应用 . . . 16
   - 上升幂与普通幂的相互转化 . . . 16
   - 下降幂与普通幂的相互转化 . . . 16
   - 多项式下降阶乘幂表示与多项式点值表示的关系 . . . 17

11. 多项式类 . . . 17
12.FFT . . . 21
13.FWT . . . 22

数学

1. 公式速查

一些数论公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \bmod \phi(p)+\phi(p)} \pmod p$
- $\mu^2(n) = \sum_{d^2|n} \mu(d)$
- $\sum_{d|n} \varphi(d) = n$
- $\sum_{d|n} 2^{\omega(d)} = \sigma_0(n^2)$，其中 ω 是不同素因子个数
- $\sum_{d|n} \mu^2(d) = 2^{\omega(d)}$

一些数论函数求和的例子

- $\sum_{i=1}^{n} i[gcd(i,n)=1] = \frac{n\varphi(n)+[n=1]}{2}$
- $\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j)=x] = \sum_d \mu(d)\lfloor\frac{n}{dx}\rfloor\lfloor\frac{m}{dx}\rfloor$
- $\sum_{i=1}^{n}\sum_{j=1}^{m} gcd(i,j) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d|gcd(i,j)}\varphi(d) = \sum_d \varphi(d)\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor$
- $S(n) = \sum_{i=1}^{n}\mu(i) = 1 - \sum_{i=1}^{n}\sum_{d|i,d<i}\mu(d) \overset{t=\frac{i}{d}}{=} 1 - \sum_{t=2}^{n} S(\lfloor\frac{n}{t}\rfloor)$
  - 利用 $[n=1] = \sum_{d|n}\mu(d)$
- $S(n) = \sum_{i=1}^{n}\varphi(i) = \sum_{i=1}^{n} i - \sum_{i=1}^{n}\sum_{d|i,d<i}\varphi(i) \overset{t=\frac{i}{d}}{=} \frac{i(i+1)}{2} - \sum_{t=2}^{n} S(\frac{n}{t})$
  - 利用 $n = \sum_{d|n}\varphi(d)$
- $\sum_{i=1}^{n}\mu^2(i) = \sum_{i=1}^{n}\sum_{d^2|n}\mu(d) = \sum_{d=1}^{\lfloor\sqrt{n}\rfloor}\mu(d)\lfloor\frac{n}{d^2}\rfloor$
- $\sum_{i=1}^{n}\sum_{j=1}^{n} gcd^2(i,j) = \sum_d d^2\sum_t \mu(t)\lfloor\frac{n}{dt}\rfloor^2$
  $\overset{x=dt}{=} \sum_x \lfloor\frac{n}{x}\rfloor^2\sum_{d|x} d^2\mu(\frac{x}{d})$
- $\sum_{i=1}^{n}\varphi(i) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}[i\perp j] - 1 = \frac{1}{2}\sum_{i=1}^{n}\mu(i)\cdot\lfloor\frac{n}{i}\rfloor^2 - 1$

斐波那契数列性质

- $F_{a+b} = F_{a-1}\cdot F_b + F_a\cdot F_{b+1}$
- $F_1 + F_3 + \cdots + F_{2n-1} = F_{2n}, F_2 + F_4 + \cdots + F_{2n} = F_{2n+1} - 1$
- $\sum_{i=1}^{n} F_i = F_{n+2} - 1$
- $\sum_{i=1}^{n} F_i^2 = F_n \cdot F_{n+1}$
- $F_n^2 = (-1)^{n-1} + F_{n-1}\cdot F_{n+1}$
- $gcd(F_a, F_b) = F_{gcd(a,b)}$
- 模 n 周期（皮萨诺周期）
  - $\pi(p^k) = p^{k-1}\pi(p)$
  - $\pi(nm) = lcm(\pi(n),\pi(m)), \forall n \perp m$
  - $\pi(2) = 3, \pi(5) = 20$
  - $\forall p \equiv \pm 1 \pmod{10}, \pi(p) | p-1$
  - $\forall p \equiv \pm 2 \pmod 5, \pi(p) | 2p+2$

常见生成函数

- $(1+ax)^n = \sum_{k=0}^{n}\binom{n}{k}a^k x^k$
- $\dfrac{1-x^{r+1}}{1-x} = \sum_{k=0}^{n} x^k$
- $\dfrac{1}{1-ax} = \sum_{k=0}^{\infty} a^k x^k$
- $\dfrac{1}{(1-x)^2} = \sum_{k=0}^{\infty}(k+1)x^k$
- $\dfrac{1}{(1-x)^n} = \sum_{k=0}^{\infty}\binom{n+k-1}{k}x^k$
- $e^x = \sum_{k=0}^{\infty}\dfrac{x^k}{k!}$

- $\ln(1+x) = \sum_{k=0}^{\infty} \frac{(-1)^{k+1}}{k} x^k$

佩尔方程

若一个丢番图方程具有以下的形式：$x^2 - ny^2 = 1$。且 n 为正整数，则称此二元二次不定方程为**佩尔方程**。

若 n 是完全平方数，则这个方程式只有平凡解 $(\pm 1, 0)$（实际上对任意的 n，$(\pm 1, 0)$ 都是解）。对于其余情况，拉格朗日证明了佩尔方程总有非平凡解。而这些解可由 $\sqrt{n}$ 的连分数求出。

$$x = [a_0; a_1, a_2, a_3] = x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}$$

设 $\frac{p_i}{q_i}$ 是 $\sqrt{n}$ 的连分数表示：$[a_0; a_1, a_2, a_3, \ldots]$ 的渐近分数列，由连分数理论知存在 i 使得 (p_i, q_i) 为佩尔方程的解。取其中最小的 i，将对应的 (p_i, q_i) 称为佩尔方程的基本解，或最小解，记作 (x_1, y_1)，则所有的解 (x_i, y_i) 可表示成如下形式：$x_i + y_i\sqrt{n} = (x_1 + y_1\sqrt{n})^i$。或者由以下的递回关系式得到：

$x_{i+1} = x_1 x_i + n y_1 y_i, y_{i+1} = x_1 y_i + y_1 x_i$。

但是：佩尔方程千万不要去推（虽然推起来很有趣，但结果不一定好看，会是两个式子）。记住佩尔方程结果的形式通常是 $a_n = k a_{n-1} - a_{n-2}$（$a_{n-2}$ 前的系数通常是 -1）。暴力 / 凑出两个基础解之后加上一个 0，容易解出 k 并验证。

Burnside & Polya

- $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$

注：X^g 是 g 下的不动点数量，也就是说有多少种东西用 g 作用之后可以保持不变。

- $|Y^X/G| = \frac{1}{|G|} \sum_{g \in G} m^{c(g)}$

注：用 m 种颜色染色，然后对于某一种置换 g，有 $c(g)$ 个置换环，为了保证置换后颜色仍然相同，每个置换环必须染成同色。

皮克定理

$2S = 2a + b - 2$

- S 多边形面积
- a 多边形内部点数
- b 多边形边上点数

莫比乌斯反演

- $g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(\frac{n}{d})$
- $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

低阶等幂求和

- $\sum_{i=1}^{n} i^1 = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$
- $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$
- $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$
- $\sum_{i=1}^{n} i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$

一些组合公式

- 错排公式：$D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-1} + D_{n-2}) = n!(\frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}) = \lfloor \frac{n!}{e} + 0.5 \rfloor$
- 卡塔兰数（n 对括号合法方案数，n 个结点二叉树个数，$n \times n$ 方格中对角线下方的单调路径数，凸 $n+2$ 边形的三角形划分数，n 个元素的合法出栈序列数）：$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$：当 $a \geq c$ or $b \geq c$ 时，$f(a,b,c,n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$；否则 $f(a,b,c,n) = nm - f(c, c-b-1, a, m-1)$。
- $g(a,b,c,n) = \sum_{i=0}^{n} i\lfloor \frac{ai+b}{c} \rfloor$：当 $a \geq c$ or $b \geq c$ 时，$g(a,b,c,n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$；否则 $g(a,b,c,n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$。
- $h(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$：当 $a \geq c$ or $b \geq c$ 时，$h(a,b,c,n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2(n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$；否则 $h(a,b,c,n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a,b,c,n)$

欧拉常数

$\gamma \approx 0.57721566490153286060651209008240243104215933 5$

$\gamma = \lim_{x \to \infty} [(\sum_{k=1}^{n} \frac{1}{k} - \ln(n)] = \int_1^{\infty} (\frac{1}{\lfloor x \rfloor} - \frac{1}{x})$

用于计算调和级数极限 $(\sum_{k=1}^{n} \frac{1}{k})$

$Cayley$ 公式

- 用来求这样一个问题的：问 n 个有标号点能组成多少棵不同的**无根**树 $ans = n^{n-2}$

$Catalan$ 数列适用的题型：

- 括号匹配，有 n 个左括号与 n 个右括号，合法匹配的方案数
- n 个元素出栈入栈的合法序列的数量
- n 个结点的二叉树的形态数量
- n 边形划分为三角形的方案数
- $n * n$ 的网格中从左下角到右上角的方案数
- $f_n = \frac{\binom{2n}{n}}{n+1}$

C++ STL 常量

- ***e_v***: the mathematical constant e
- ***log2e_v***: $log_2 e$
- ***log10e_v***: $log_{10} e$
- ***pi_v***: π
- ***inv_pi_v***: $\frac{1}{\pi}$
- ***inv_sqrtpi_v***: $\frac{1}{\sqrt{\pi}}$
- ***ln_2***: $\ln 2$
- ***ln10_v***: $\ln 10$
- ***sqrt2_v***: $\sqrt{2}$
- ***sqrt3_v***: $\sqrt{3}$
- ***inv_sqrt3_v***: $\frac{1}{\sqrt{3}}$
- ***egamma_v***: 欧拉常数 (调和级数与自然对数的差值的极限)

2. 快速幂/龟速乘

- 快速幂压行

```
int binpow(int x, int y, int mod, int res = 1){
    for (; y; y >>= 1, (x *= x) %= mod) if (y & 1) (res *= x) %= mod;
    return res;
}
```

- 龟速乘压行

```
int binmul(int x, int y, int mod, int res = 0){
    for (; y; y >>= 1, (x += x) %= mod) if (y & 1) (res += x) %= mod;
    return res;
}
```

- 取模快速乘 $O(1)$

```
int mul(int u, int v, int p) {
    return (u * v - int((long double) u * v / p) * p + p) % p;
}
int mul(int u, int v, int p) { // 卡常
    int t = u * v - int((long double) u * v / p) * p;
    return t < 0 ? t + p : t;
}
```

3. 筛 (素数筛，积性筛)

欧拉筛

```
int vis[1000005],prime[1000005],cnt;
for(int i=2;i<=1000000;i++){
    if(!vis[i]){
        vis[i]=i;
        prime[++cnt]=i;
    }
    for(int j=1;j<=cnt&&prime[j]*i<=1000000;j++){
        vis[prime[j]*i]=prime[j];
        if(i%prime[j]==0)break;
    }
}
```

欧拉筛 + 莫比乌斯函数

```
int mu[N], b[N], pri[N];
void mus(){
    int tot=0;
    mu[1] = 1;
    for(int i=2;i<N;i++){
        if(!b[i]) mu[i] = -1, pri[++tot] = i;
        for(int j=1;j<=tot && i*pri[j]<n;j++){
            b[i*pri[j]] = 1;
            if(i%pri[j]==0) break ;
            mu[i*pri[j]] = -mu[i];
        }
    }
}
```

欧拉筛 + 欧拉函数

```
const int p_max = 1e5 + 100;
int phi[p_max];

void get_phi(){
    phi[1] = 1;
    static bool vis[p_max];
    static int prime[p_max], p_sz, d;
    for(int i = 2; i < p_max; i++){
        if(!vis[i]) prime[p_sz++] = i, phi[i] = i - 1;
        for(int j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j){
            vis[d] = 1;
```

```
            if(i % prime[j] == 0){
                phi[d] = phi[i] * prime[j];
                break;
            } else {
                phi[d] = phi[i] * (prime[j] - 1);
            }
        }
    }
}
```

欧拉函数

```
int euler(int x){ //欧拉函数
    int res=x,sq=sqrt(x*1.0);
    for(int i=2;i<=sq;i++){
        if(x%i==0){
            res=res-res/i;
            while(x%i==0) x/=i;
        }
    }
    if(x>1) res=res-res/x;
    return res;
}

int phi[N];
void euler(int n){
    for(int i=1;i<=n;i++) phi[i]=i;
    for(int i=2;i<=n;i++){
        if(phi[i]==i){
            for(int j=i;j<=n;j+=i) phi[j]=phi[j]/i*(i-1);
        }
    }
}

```

筛法求约数和

f_i 表示 i 的约数和，g_i 表示 i 的最小质因子的 $p^0+p^1+p^2+...p^k$.

```
void pre() {
  g[1] = f[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (!v[i]) v[i] = 1, p[++tot] = i, g[i] = i + 1, f[i] = i + 1;
    for (int j = 1; j <= tot && i <= n / p[j]; ++j) {
      v[p[j] * i] = 1;
      if (i % p[j] == 0) {
        g[i * p[j]] = g[i] * p[j] + 1;
        f[i * p[j]] = f[i] / g[i] * g[i * p[j]];
        break;
      } else {
        f[i * p[j]] = f[i] * f[p[j]];
        g[i * p[j]] = 1 + p[j];
      }
    }
  }
}
```

杜教筛

求 $S(n)=\sum_{i=1}^{n} f(i)$，其中 f 是一个积性函数。

构造一个积性函数 g，那么由 $(f*g)(n)=\sum_{d|n} f(d)g(\frac{n}{d})$，得到 $f(n)=(f*g)(n)-\sum_{d|n,d<n} f(d)g(\frac{n}{d})$。

$$g(1)S(n) = \sum_{i=1}^{n}(f*g)(i)-\sum_{i=1}^{n}\sum_{d|i,d<i} f(d)g(\frac{n}{d}) \tag{1}$$

$$\overset{t=\frac{i}{d}}{=} \sum_{i=1}^{n}(f*g)(i)-\sum_{t=2}^{n} g(t)S(\lfloor\frac{n}{t}\rfloor) \tag{2}$$

当然，要能够由此计算 $S(n)$，会对 f, g 提出一些要求：

- $f * g$ 要能够快速求前缀和。
- g 要能够快速求分段和（前缀和）。
- 对于正常的积性函数 $g(1) = 1$，所以不会有什么问题。

在预处理 $S(n)$ 前 $n^{\frac{2}{3}}$ 项的情况下复杂度是 $O(n^{\frac{2}{3}})$。

```
namespace dujiao {
    const int M = 5E6;
    int f[M] = {0, 1};
    void init() {
        static bool vis[M];
        static int pr[M], p_sz, d;
        for(int i = 2; i < M; i++){
            if (!vis[i]) { pr[p_sz++] = i; f[i] = -1; }
            for(int j = 0; j < p_sz; j++){
                if ((d = pr[j] * i) >= M) break;
                vis[d] = 1;
                if (i % pr[j] == 0) {
                    f[d] = 0;
                    break;
                } else f[d] = -f[i];
            }
        }
        for(int i = 2; i < M ; i++) f[i] += f[i - 1];
    }
    inline int s_fg(int n) { return 1; }
    inline int s_g(int n) { return n; }

    int N, rd[M];
    bool vis[M];
    int go(int n) {
        if (n < M) return f[n];
        int id = N / n;
        if (vis[id]) return rd[id];
        vis[id] = true;
        int& ret = rd[id] = s_fg(n);
        for (int l = 2, v, r; l <= n; l = r + 1) {
            v = n / l; r = n / v;
            ret -= (s_g(r) - s_g(l - 1)) * go(v);
        }
        return ret;
    }
    int solve(int n) {
        N = n;
        memset(vis, 0, sizeof vis);
        return go(n);
    }
}
```

4. 素数测试

- 前置：快速乘、快速幂
- int 范围内只需检查 2, 7, 61
- long long 范围 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- 3E15 内 2, 2570940, 880937, 610386380, 4130785767
- 4E13 内 2, 2570940, 211991001, 3749873356

```
bool checkQ(int a, int n) {
    if (n == 2) return 1;
    if (n == 1 || !(n & 1)) return 0;
    int d = n - 1;
    while (!(d & 1)) d >>= 1;
    int t = binpow(a, d, n);  // 不一定需要快速乘
    while (d != n - 1 && t != 1 && t != n - 1) {
        t = mul(t, t, n);
        d <<= 1;
    }
    return t == n - 1 || d & 1;
```

```
}

bool primeQ(int n) {
    static vector<int> t = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    if (n <= 1) return false;
    for (int k: t) if (!checkQ(k, n)) return false;
    return true;
}
```

5. 质因数分解

Pollard-Rho

```
mt19937 mt(time(0));
int pointard_rho(int n, int c) {
    int x = uniform_int_distribution<int>(1, n - 1)(mt), y = x;
    auto f = [&](int v) { int t = mul(v, v, n) + c; return t < n ? t : t - n; };
    while (1) {
        x = f(x); y = f(f(y));
        if (x == y) return n;
        int d = gcd(abs(x - y), n);
        if (d != 1) return d;
    }
}

int fac[100], fcnt;
void get_fac(int n, int cc = 19260817) {
    if (n == 4) { fac[fcnt++] = 2; fac[fcnt++] = 2; return; }
    if (primeQ(n)) { fac[fcnt++] = n; return; }
    int p = n;
    while (p == n) p = pointard_rho(n, --cc);
    get_fac(p); get_fac(n / p);
}

void go_fac(int n) { fcnt = 0; if (n > 1) get_fac(n); }
```

朴素质因数分解 (带指数)(前置素数筛)

```
int factor[30], f_sz, factor_exp[30];
void get_factor(int x) {
    f_sz = 0;
    int t = sqrt(x + 0.5);
    for (int i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor_exp[f_sz] = 0;
            while (x % pr[i] == 0) {
                x /= pr[i];
                ++factor_exp[f_sz];
            }
            factor[f_sz++] = pr[i];
        }
    if (x > 1) {
        factor_exp[f_sz] = 1;
        factor[f_sz++] = x;
    }
}
```

朴素质因数分解 (不带指数)

```
int factor[30], f_sz;
void get_factor(int x) {
    f_sz = 0;
    int t = sqrt(x + 0.5);
    for (int i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor[f_sz++] = pr[i];
            while (x % pr[i] == 0) x /= pr[i];
        }
    if (x > 1) factor[f_sz++] = x;
}
```

5. 矩阵运算类

```
struct Mat{
    static const int M = 2;
    int v[M][M];
    Mat() { memset(v ,0 ,sizeof(v)); }
    void eye() { for(int i = 0 ;i < M; i++) v[i][i] = 1; }
    int * operator [] (int x) { return v[x]; }
    const int *operator [] (int x) const { return v[x]; }
    Mat operator * (const Mat& B) {
        const Mat &A = *this;
        Mat ret;
        for(int k = 0; k < M; k++){
            for(int i = 0; i < M; i++) if(A[i][k]) {
                for(int j = 0; j < M; j++){
                    ret[i][j] = (ret[i][j] + A[i][k] * B[k][j]) % MOD;
                }
            }
        }
        return ret;
    }
    Mat operator + (const Mat& B) {
        const Mat &A = *this;
        Mat ret;
        for(int i = 0; i < M; i++)
            for(int j = 0; j < M; j++)
                ret[i][j] = (A[i][j] + B[i][j]) % MOD;
        return ret;
    }
    Mat pow(int n) const {
        Mat A = *this, ret; ret.eye();
        for(; n; n >>= 1, A = A * A) if(n & 1) ret = ret * A;
        return ret;
    }
};
```

6. 高斯消元

- n - 方程个数，m - 变量个数，a 是 n * (m + 1) 的增广矩阵，free 是否为自由变量
- 返回自由变量个数，-1 无解
- 浮点数版本

```
typedef double LD;
const LD eps = 1E-10;
const int maxn = 2000 + 10;

int n, m;
LD a[maxn][maxn], x[maxn];
bool free_x[maxn];

inline int sgn(LD x) { return (x > eps) - (x < -eps); }

int gauss(LD a[maxn][maxn], int n, int m) {
    memset(free_x, 1, sizeof free_x); memset(x, 0, sizeof x);
    int r = 0, c = 0;
    while (r < n && c < m) {
        int m_r = r;
        for(int i = r + 1; i < n; ++i)
            if (fabs(a[i][c]) > fabs(a[m_r][c])) m_r = i;
        if (m_r != r)
            for(int j = c; j <= m; ++j)
                swap(a[r][j], a[m_r][j]);
        if (!sgn(a[r][c])) {
            a[r][c] = 0;
            ++c;
            continue;
        }
        for(int i = r + 1; i < n; ++i)
            if (a[i][c]) {
```

```
                LD t = a[i][c] / a[r][c];
                for(int j = c; j <= m; j++) a[i][j] -= a[r][j] * t;
            }
        ++r; ++c;
    }
    for(int i = r; i < n; i++)
        if (sgn(a[i][m])) return -1;
    if (r < m) {
        for(int i = r - 1; i > -1; --i) {
            int f_cnt = 0, k = -1;
            for(int j = 0; j < m; ++j)
                if (sgn(a[i][j]) && free_x[j]) {
                    ++f_cnt;
                    k = j;
                }
            if(f_cnt > 0) continue;
            LD s = a[i][m];
            for(int j = 0; j < m; j++)
                if (j != k) s -= a[i][j] * x[j];
            x[k] = s / a[i][k];
            free_x[k] = 0;
        }
        return m - r;
    }
    for(int i = m - 1; i > -1; --i) {
        LD s = a[i][m];
        for(int j = i + 1; j < m; j++)
            s -= a[i][j] * x[j];
        x[i] = s / a[i][i];
    }
    return 0;
}
/*
3 4
1 1 -2 2
2 -3 5 1
4 -1 1 5
5 0 -1 7
// many

3 4
1 1 -2 2
2 -3 5 1
4 -1 -1 5
5 0 -1 0 2
// no

3 4
1 1 -2 2
2 -3 5 1
4 -1 1 5
5 0 1 0 7
// one
*/
```

7. 线性基

线性基是向量空间的一组基，通常可以解决有关异或的一些题目。

通俗一点的讲法就是由一个集合构造出来的另一个集合，它有以下几个性质：

- 线性基的元素能相互异或得到原集合的元素的所有相互异或得到的值。
- 线性基是满足性质 1 的最小的集合。
- 线性基没有异或和为 0 的子集。
- 线性基中每个元素的异或方案唯一，也就是说，线性基中不同的异或组合异或出的数都是不一样的。
- 线性基中每个元素的二进制最高位互不相同。

构造线性基的方法如下：

对原集合的每个数 p 转为二进制，从高位向低位扫，对于第 x 位是 1 的，如果 a_x 不存在，那么令 $a_x = p$ 并结束扫描，如果存在，令 $p = p \text{ xor } a_x$。

代码：

```
inline void insert(long long x) {
    for (int i = 55; i + 1; i--) {
        if (!(x >> i))  // x 的第 i 位是 0
            continue;
        if (!p[i]) {
            p[i] = x;
            break;
        }
        x ^= p[i];
    }
}
```

查询原集合内任意几个元素 xor 的最大值，就可以用线性基解决。

将线性基从高位向低位扫，若 xor 上当前扫到的 a_x 答案变大，就把答案异或上 a_x。

为什么能行呢？因为从高往低位扫，若当前扫到第 i 位，意味着可以保证答案的第 i 位为 1，且后面没有机会改变第 i 位。

查询原集合内任意几个元素 xor 的最小值，就是线性基集合所有元素中最小的那个。

查询某个数是否能被异或出来，类似于插入，如果最后插入的数 p 被异或成了 0，则能被异或出来。

8. 扩展欧几里得

扩展欧几里得

```
int exgcd(int a, int b, int &x, int &y){
    if (b == 0){
        x = 1, y = 0;
        return a;
    }
    int ans = exgcd(b, a % b, x, y);
    int x1 = x, y1 = y;
    x = y1, y = x1 - a / b * y1;
    return ans;
}
```

扩欧逆元

```
int inv(int a, int b, int x = 0, int y = 0){
    if (exgcd(a, b, x, y) != 1) return -1;
    else return (x % b + b) % b;
}
```

9. 中国剩余定理 (exGCD)

```
int CRT(int *m, int *r, int n) {
    if (!n) return 0;
    int M = m[0], R = r[0], x, y, d;
    for(int i = 1; i < n; i++) {
        d = ex_gcd(M, m[i], x, y);
        if ((r[i] - R) % d) return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        // 防爆 int
        // x = mul((r[i] - R) / d, x, m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
    }
    return R >= 0 ? R : R + M;
}
```

10. 二次剩余

理论知识附页

```cpp
int q1, q2, w;
struct P { // x + y * sqrt(w)
    int x, y;
};

P pmul(const P& a, const P& b, int p) {
    P res;
    res.x = (a.x * b.x + a.y * b.y % p * w) % p;
    res.y = (a.x * b.y + a.y * b.x) % p;
    return res;
}

P bin(P x, int n, int MOD) {
    P ret = {1, 0};
    for (; n; n >>= 1, x = pmul(x, x, MOD))
        if (n & 1) ret = pmul(ret, x, MOD);
    return ret;
}
int Legendre(int a, int p) { return bin(a, (p - 1) >> 1, p); }

int equation_solve(int b, int p) {
    if (p == 2) return 1;
    if ((Legendre(b, p) + 1) % p == 0)
        return -1;
    int a;
    while (true) {
        a = rand() % p;
        w = ((a * a - b) % p + p) % p;
        if ((Legendre(w, p) + 1) % p == 0)
            break;
    }
    return bin({a, 1}, (p + 1) >> 1, p).x;
}

int main() {
    int T; cin >> T;
    while (T--) {
        int a, p; cin >> a >> p;
        a = a % p;
        int x = equation_solve(a, p);
        if (x == -1) {
            puts("No root");
        } else {
            int y = p - x;
            if (x == y) cout << x << endl;
            else cout << min(x, y) << " " << max(x, y) << endl;
        }
    }
}
```

11. 伯努利数和等幂求和

- 预处理逆元
- 预处理组合数
- $\sum_{i=0}^{n} i^k = \frac{1}{k+1} \sum_{i=0}^{k} \binom{k+1}{i} B_{k+1-i} (n+1)^i$.
- 也可以 $\sum_{i=0}^{n} i^k = \frac{1}{k+1} \sum_{i=0}^{k} \binom{k+1}{i} B^+_{k+1-i} n^i$。区别在于 $B^+_1 = 1/2$。(心态崩了)

```cpp
namespace Bernouinti {
    const int M = 100;
    int inv[M] = {-1, 1};
    void inv_init(int n, int p) {
        for(int i = 2; i < n; i++)
            inv[i] = (p - p / i) * inv[p % i] % p;
    }

    int C[M][M];
```

```
    void init_C(int n) {
        for(int i = 0; i < n; i++) {
            C[i][0] = C[i][i] = 1;
            for(int j = 1; j < i; j++)
                C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
        }
    }

    int B[M] = {1};
    void init() {
        inv_init(M, MOD);
        init_C(M);
        for(int i = 1; i < M - 1; i++) {
            int& s = B[i] = 0;
            for(int j = 0; j < i; j++)
                s += C[i + 1][j] * B[j] % MOD;
            s = (s % MOD * -inv[i + 1] % MOD + MOD) % MOD;
        }
    }

    int p[M] = {1};
    int go(int n, int k) {
        n %= MOD;
        if (k == 0) return n;
        for(int i = 1; i < k + 2; i++)
            p[i] = p[i - 1] * (n + 1) % MOD;
        int ret = 0;
        for(int i = 1; i < k + 2; i++)
            ret += C[k + 1][i] * B[k + 1 - i] % MOD * p[i] % MOD;
        ret = ret % MOD * inv[k + 1] % MOD;
        return ret;
    }
}
```

12. 数论分块/整除分块

$f(i) = \lfloor \frac{n}{i} \rfloor = v$ 时 i 的取值范围是 $[l, r]$。

```
for (int l = 1, v, r; l <= N; l = r + 1) {
    v = N / l; r = N / v;
}
```

向上取整:

```
for (int l = 1, r, k; l <= n; l = r + 1){
    k = (N + l - 1) / l;
    r = (N - 1) / (k - 1);
}
```

13. 斯特林数

第一类斯特林数

- 绝对值是 n 个元素划分为 k 个环排列的方案数。
- $s(n,k) = s(n-1,k-1) + (n-1)s(n-1,k)$

第二类斯特林数

- n 个元素划分为 k 个等价类的方案数
- $S(n,k) = S(n-1,k-1) + kS(n-1,k)$

```
S[0][0] = 1;
for(int i = 1; i < N; i++)
    for(int j = 1; j <= i; j++) S[i][j] = (S[i - 1][j - 1] + j * S[i - 1][j]) % MOD;
```

第二类斯特林数（斯特林子集数）

$\begin{Bmatrix} n \\ k \end{Bmatrix}$，也可记做 $S(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数。

递推式

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

边界是 $\begin{Bmatrix} n \\ 0 \end{Bmatrix} = [n = 0]$。

考虑用组合意义来证明。

我们插入一个新元素时，有两种方案：

- 将新元素单独放入一个子集，有 $\begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$ 种方案；
- 将新元素放入一个现有的非空子集，有 $k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$ 种方案。

根据加法原理，将两式相加即可得到递推式。

通项公式

$$\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_{i=0}^{m} \frac{(-1)^{m-i} i^n}{i!(m-i)!}$$

使用容斥原理证明该公式。设将 n 个两两不同的元素，划分到 k 个两两不同的集合（允许空集）的方案数为 G_i，将 n 个两两不同的元素，划分到 k 个两两不同的非空集合（不允许空集）的方案数为 F_i。

显然

$$G_i = k^n$$

$$G_i = \sum_{j=0}^{i} \binom{i}{j} F_j$$

根据二项式反演

$$\begin{aligned} F_i &= \sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} G_j \\ &= \sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} j^n \\ &= \sum_{j=0}^{i} \frac{i!(-1)^{i-j} j^n}{j!(i-j)!} \end{aligned}$$

考虑 F_i 与 $\begin{Bmatrix} n \\ i \end{Bmatrix}$ 的关系。第二类斯特林数要求集合之间互不区分，因此 F_i 正好就是 $\begin{Bmatrix} n \\ i \end{Bmatrix}$ 的 $i!$ 倍。于是

$$\begin{Bmatrix} n \\ m \end{Bmatrix} = \frac{F_m}{m!} = \sum_{i=0}^{m} \frac{(-1)^{m-i} i^n}{i!(m-i)!}$$

同一行第二类斯特林数的计算

“同一行”的第二类斯特林数指的是，有着不同的 i，相同的 n 的一系列 $\begin{Bmatrix} n \\ i \end{Bmatrix}$。求出同一行的所有第二类斯特林数，就是对 $i = 0..n$ 求出了将 n 个不同元素划分为 i 个非空集的方案数。

```
int main() {
      scanf("%d", &n);
      fact[0] = 1;
      for (int i = 1; i <= n; ++i) fact[i] = (ll)fact[i - 1] * i % mod;
      exgcd(fact[n], mod, ifact[n], ifact[0]),
          ifact[n] = (ifact[n] % mod + mod) % mod;
      for (int i = n - 1; i >= 0; --i) ifact[i] = (ll)ifact[i + 1] * (i + 1) % mod;
      poly f(n + 1), g(n + 1);
      for (int i = 0; i <= n; ++i)
        g[i] = (i & 1 ? mod - 1ll : 1ll) * ifact[i] % mod,
```

```
    f[i] = (ll)qpow(i, n) * ifact[i] % mod;
  f *= g, f.resize(n + 1);
  for (int i = 0; i <= n; ++i) printf("%d ", f[i]);
  return 0;
}
```

方法 2. 利用指数型生成函数

一个盒子装 i 个物品且盒子非空的方案数是 $[i > 0]$。我们可以写出它的指数型生成函数为 $F(x) = \sum\limits_{i=1}^{+\infty} \dfrac{x^i}{i!} = e^x - 1$。经过之前的学习，我们明白 $F^k(x)$ 就是 i 个有标号物品放到 k 个有标号盒子里的指数型生成函数，$\exp F(x) = \sum\limits_{i=0}^{+\infty} \dfrac{F^i(x)}{i!}$ 就是 i 个有标号物品放到任意多个无标号盒子里的指数型生成函数（EXP 通过每项除以一个 $i!$ 去掉了盒子的标号）。这里涉及到很多“有标号”“无标号”的内容，注意辨析。

那么 $\begin{Bmatrix} i \\ k \end{Bmatrix} = \dfrac{\left[\dfrac{x^i}{i!}\right] F^k(x)}{k!}$，$O(n \log n)$ 计算多项式幂即可。实际使用时比 $O(n \log n)$ 的方法 1 要慢。

```
int main() {
  scanf("%d%d", &n, &k);
  poly f(n + 1);
  fact[0] = 1;
  for (int i = 1; i <= n; ++i) fact[i] = (ll)fact[i - 1] * i % mod;
  for (int i = 1; i <= n; ++i) f[i] = qpow(fact[i], mod - 2);
  f = exp(log(f >> 1) * k) << k, f.resize(n + 1);
  int inv = qpow(fact[k], mod - 2);
  for (int i = 0; i <= n; ++i)
    printf("%lld ", (ll)f[i] * fact[i] % mod * inv % mod);
  return 0;
}
```

第一类斯特林数（Stirling Number）

第一类斯特林数（斯特林轮换数）$\begin{bmatrix} n \\ k \end{bmatrix}$，也可记做 $s(n, k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空轮换的方案数。

一个轮换就是一个首尾相接的环形排列。我们可以写出一个轮换 $[A, B, C, D]$，并且我们认为 $[A, B, C, D] = [B, C, D, A] = [C, D, A, B] = [D, A, B, C]$，即，两个可以通过旋转而互相得到的轮换是等价的。注意，我们不认为两个可以通过翻转而相互得到的轮换等价，即 $[A, B, C, D] \neq [D, C, B, A]$。

递推式

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix}$$

边界是 $\begin{bmatrix} n \\ 0 \end{bmatrix} = [n = 0]$。

该递推式的证明可以考虑其组合意义。

我们插入一个新元素时，有两种方案：

- 将该新元素置于一个单独的轮换中，共有 $\begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$ 种方案；
- 将该元素插入到任何一个现有的轮换中，共有 $(n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix}$ 种方案。

根据加法原理，将两式相加即可得到递推式。

通项公式

第一类斯特林数没有实用的通项公式。

同一行第一类斯特林数的计算

类似第二类斯特林数，我们构造同行第一类斯特林数的生成函数，即

$$F_n(x)=\sum_{i=0}^{n}\begin{bmatrix}n\\i\end{bmatrix}x^i$$

根据递推公式，不难写出

$$F_n(x)=(n-1)F_{n-1}(x)+xF_{n-1}(x)$$

于是

$$F_n(x)=\prod_{i=0}^{n-1}(x+i)=\frac{(x+n-1)!}{(x-1)!}$$

这其实是 x 的 n 次上升阶乘幂，记做 $x^{\overline{n}}$。这个东西自然是可以暴力分治乘 $O(n\log^2 n)$ 求出的，但用上升幂相关做法可以 $O(n\log n)$ 求出。

同一列第一类斯特林数的计算

仿照第二类斯特林数的计算，我们可以用指数型生成函数解决该问题。注意，由于递推公式和行有关，我们不能利用递推公式计算同列的第一类斯特林数。

显然，单个轮换的指数型生成函数为

$$F(x)=\sum_{i=1}^{n}\frac{(i-1)!x^i}{i!}=\sum_{i=1}^{n}\frac{x^i}{i}$$

它的 k 次幂就是 $\begin{bmatrix}i\\k\end{bmatrix}$ 的指数型生成函数，$O(n\log n)$ 计算即可。

```cpp
int main() {
    scanf("%d%d", &n, &k);
    fact[0] = 1;
    for (int i = 1; i <= n; ++i) fact[i] = (ll)fact[i - 1] * i % mod;
    ifact[n] = qpow(fact[n], mod - 2);
    for (int i = n - 1; i >= 0; --i) ifact[i] = (ll)ifact[i + 1] * (i + 1) % mod;
    poly f(n + 1);
    for (int i = 1; i <= n; ++i) f[i] = (ll)fact[i - 1] * ifact[i] % mod;
    f = exp(log(f >> 1) * k) << k, f.resize(n + 1);
    for (int i = 0; i <= n; ++i)
        printf("%lld ", (ll)f[i] * fact[i] % mod * ifact[k] % mod);
    return 0;
}
```

应用

上升幂与普通幂的相互转化

我们记上升阶乘幂 $x^{\overline{n}}=\prod_{k=0}^{n-1}(x+k)$。

则可以利用下面的恒等式将上升幂转化为普通幂：

$$x^{\overline{n}}=\sum_k\begin{bmatrix}n\\k\end{bmatrix}x^k$$

如果将普通幂转化为上升幂，则有下面的恒等式：

$$x^n=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}(-1)^{n-k}x^{\overline{k}}$$

下降幂与普通幂的相互转化

我们记下降阶乘幂 $x^{\underline{n}}=\dfrac{x!}{(x-n)!}=\prod_{k=0}^{n-1}(x-k)$。

则可以利用下面的恒等式将普通幂转化为下降幂：

$$x^n = \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} x^{\underline{k}}$$

如果将下降幂转化为普通幂，则有下面的恒等式：

$$x^{\underline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k$$

多项式下降阶乘幂表示与多项式点值表示的关系

在这里，多项式的下降阶乘幂表示就是用

$$f(x) = \sum_{i=0}^{n} b_i x^{\underline{i}}$$

的形式表示一个多项式，而点值表示就是用 $n+1$ 个点

$(i, a_i), i = 0..n$

来表示一个多项式。

显然，下降阶乘幂 b 和点值 a 间满足这样的关系：

$$a_k = \sum_{i=0}^{n} b_i k^{\underline{i}}$$

即

$$a_k = \sum_{i=0}^{n} \frac{b_i k!}{(k-i)!}$$

$$\frac{a_k}{k!} = \sum_{i=0}^{k} b_i \frac{1}{(k-i)!}$$

这是一个卷积形式的式子，我们可以在 $O(n \log n)$ 的时间复杂度内完成点值和下降阶乘幂的互相转化。

11. 多项式类

```cpp
constexpr int P(998244353), G(3), L(1 << 18);
inline void inc(int &x, int y) {
  x += y;
  if (x >= P) x -= P;
}
inline void dec(int &x, int y) {
  x -= y;
  if (x < 0) x += P;
}
inline int mod(LL x) { return x % P; }
int fpow(int x, int k = P - 2) {
  int r = 1;
  for (; k; k >>= 1, x = 1LL * x * x % P) {
    if (k & 1) r = 1LL * r * x % P;
  }
  return r;
}
int w[L], fac[L], ifac[L], inv[L], _ = [] {
  w[L / 2] = 1;
  for (int i = L / 2 + 1, x = fpow(G, (P - 1) / L); i < L; i++) w[i] = 1LL * w[i - 1] * x % P;
  for (int i = L / 2 - 1; i >= 0; i--) w[i] = w[i << 1];

  fac[0] = 1;
  for (int i = 1; i < L; i++) fac[i] = 1LL * fac[i - 1] * i % P;
  ifac[L - 1] = fpow(fac[L - 1]);
  for (int i = L - 1; i; i--) {
    ifac[i - 1] = 1LL * ifac[i] * i % P;
    inv[i] = 1LL * ifac[i] * fac[i - 1] % P;
  }
  return 0;
}();
void dft(int *a, int n) {
```

```
  assert((n & n - 1) == 0);
  for (int k = n >> 1; k; k >>= 1) {
    for (int i = 0; i < n; i += k << 1) {
      for (int j = 0; j < k; j++) {
        int &x = a[i + j], y = a[i + j + k];
        a[i + j + k] = 1LL * (x - y + P) * w[k + j] % P;
        inc(x, y);
      }
    }
  }
}
void idft(int *a, int n) {
  assert((n & n - 1) == 0);
  for (int k = 1; k < n; k <<= 1) {
    for (int i = 0; i < n; i += k << 1) {
      for (int j = 0; j < k; j++) {
        int x = a[i + j], y = 1LL * a[i + j + k] * w[k + j] % P;
        a[i + j + k] = x - y < 0 ? x - y + P : x - y;
        inc(a[i + j], y);
      }
    }
  }
  for (int i = 0, inv = P - (P - 1) / n; i < n; i++)
    a[i] = 1LL * a[i] * inv % P;
  std::reverse(a + 1, a + n);
}
inline int norm(int n) { return 1 << std::__lg(n * 2 - 1); }
struct Poly : public std::vector<int> {
#define T (*this)
  using std::vector<int>::vector;
  void append(const Poly &r) {
    insert(end(), r.begin(), r.end());
  }
  int len() const { return size(); }
  Poly operator-() const {
    Poly r(T);
    for (auto &x : r) x = x ? P - x : 0;
    return r;
  }
  Poly &operator+=(const Poly &r) {
    if (r.len() > len()) resize(r.len());
    for (int i = 0; i < r.len(); i++) inc(T[i], r[i]);
    return T;
  }
  Poly &operator-=(const Poly &r) {
    if (r.len() > len()) resize(r.len());
    for (int i = 0; i < r.len(); i++) dec(T[i], r[i]);
    return T;
  }
  Poly &operator^=(const Poly &r) {
    if (r.len() < len()) resize(r.len());
    for (int i = 0; i < len(); i++) T[i] = 1LL * T[i] * r[i] % P;
    return T;
  }
  Poly &operator*=(int r) {
    for (int &x : T) x = 1LL * x * r % P;
    return T;
  }
  Poly operator+(const Poly &r) const { return Poly(T) += r; }
  Poly operator-(const Poly &r) const { return Poly(T) -= r; }
  Poly operator^(const Poly &r) const { return Poly(T) ^= r; }
  Poly operator*(int r) const { return Poly(T) *= r; }
  Poly &operator<<=(int k) { return insert(begin(), k, 0), T; }
  Poly operator<<(int r) const { return Poly(T) <<= r; }
  Poly operator>>(int r) const { return r >= len() ? Poly() : Poly(begin() + r, end()); }
  Poly &operator>>=(int r) { return T = T >> r; }
  Poly pre(int k) const { return k < len() ? Poly(begin(), begin() + k) : T; }
  friend void dft(Poly &a) { dft(a.data(), a.len()); }
  friend void idft(Poly &a) { idft(a.data(), a.len()); }
  friend Poly conv(const Poly &a, const Poly &b, int n) {
    Poly p(a), q;
```

```
    p.resize(n), dft(p);
    p ^= &a == &b ? p : (q = b, q.resize(n), dft(q), q);
    idft(p);
    return p;
  }
  friend Poly operator*(const Poly &a, const Poly &b) {
    int len = a.len() + b.len() - 1;
    if (a.len() <= 16 || b.len() <= 16) {
      Poly c(len);
      for (int i = 0; i < a.len(); i++)
        for (int j = 0; j < b.len(); j++)
          c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
      return c;
    }
    return conv(a, b, norm(len)).pre(len);
  }
  Poly deriv() const {
    if (empty()) return Poly();
    Poly r(len() - 1);
    for (int i = 1; i < len(); i++) r[i - 1] = 1LL * i * T[i] % P;
    return r;
  }
  Poly integ() const {
    if (empty()) return Poly();
    Poly r(len() + 1);
    for (int i = 0; i < len(); i++) r[i + 1] = 1LL * fpow(i + 1) * T[i] % P;
    return r;
  }
  Poly inv(int m) const {
    Poly x = {fpow(T[0])};
    for (int k = 1; k < m; k *= 2) {
      x.append(-((conv(pre(k * 2), x, k * 2) >> k) * x).pre(k));
    }
    return x.pre(m);
  }
  Poly log(int m) const { return (deriv() * inv(m)).integ().pre(m); }
  Poly exp(int m) const {
    Poly x = {1};
    for (int k = 1; k < m; k *= 2) {
      x.append((x * (pre(k * 2) - x.log(k * 2) >> k)).pre(k));
    }
    return x.pre(m);
  }
  Poly sqrt(int m) const {
    Poly x = {1}, y = {1};
    for (int k = 1; k < m; k *= 2) {
      x.append(((pre(k * 2) - x * x >> k) * y).pre(k) * (P + 1 >> 1));
      if (k * 2 < m) {
        y.append(-((conv(x.pre(k * 2), y, k * 2) >> k) * y).pre(k));
      }
    }
    return x.pre(m);
  }
  Poly rev() const { return Poly(rbegin(), rend()); }
  Poly mulT(Poly b) { return T * b.rev() >> b.len() - 1; }
#undef T
};
Poly operator/(Poly a, Poly b) {
  int n = a.len(), m = b.len();
  if (n < m) return {0};
  int k = norm(n - m + 1);
  a = a.rev();
  a.resize(k);
  return (a * b.rev().inv(k)).pre(n - m + 1).rev();
}
std::pair<Poly, Poly> div(Poly a, Poly b) {
  int m = b.len();
  Poly c = a / b;
  return {c, a.pre(m - 1) - (b * c).pre(m - 1)};
}
Poly operator%(Poly a, Poly b) {
```

```
  return div(a, b).second;
}
struct SegTree {
  std::vector<Poly> p;
  int n, raw_n;
  SegTree(Poly a) {
    n = norm(raw_n = a.size());
    p.resize(n * 2);
    for (int i = 0; i < n; i++) {
      p[i + n] = Poly({1, i < raw_n ? P - a[i] : 0});
    }
    for (int i = n - 1; i; i--) {
      int l = i * 2, r = l | 1, k = p[l].size() - 1 << 1;
      p[l].resize(k), dft(p[l]);
      p[r].resize(k), dft(p[r]);
      idft(p[i] = p[l] ^ p[r]);
      p[i].push_back((p[i][0] - 1 + P) % P);
      p[i][0] = 1;
    }
  }
  Poly eval(Poly f) {
    int m = f.size();
    if (m == 1) return Poly(raw_n, f[0]);
    Poly q = f.rev() * p[1].inv(m);
    q.resize(m);
    if (m > n) {
      q >>= m - n;
    } else {
      q <<= n - m;
    }
    for (int k = n, o = 1; k > 1; k >>= 1) {
      for (int i = 0; i < n; i += k, o++) {
        if (i >= raw_n) continue;
        int *a = &q[i], *l = p[o * 2].data(), *r = p[o * 2 + 1].data();
        dft(a, k);
        Poly x(k), y(k);
        for (int j = 0; j < k; j++) x[j] = 1LL * a[j] * r[j] % P;
        for (int j = 0; j < k; j++) y[j] = 1LL * a[j] * l[j] % P;
        idft(x), idft(y);
        for (int j = k / 2; j < k; j++) *a++ = x[j];
        for (int j = k / 2; j < k; j++) *a++ = y[j];
      }
    }
    return q.pre(raw_n);
  }
  Poly interpolate(Poly b) {
    assert(b.len() == raw_n);
    Poly q = eval(p[1].pre(raw_n + 1).rev().deriv());
    for (int i = 0; i < raw_n; i++) q[i] = 1LL * fpow(q[i]) * b[i] % P;
    q.resize(n);
    for (int k = 1, h = n >> 1; k < n; k <<= 1, h >>= 1)
      for (int i = 0, o = h; i < n; i += k << 1, o++) {
        if (i >= raw_n) continue;
        int *a = &q[i], *b = &q[i + k], *l = p[o * 2].data(), *r = p[o * 2 + 1].data();
        Poly x(k * 2), y(k * 2);
        for (int j = 0; j < k; j++) x[j] = a[j];
        for (int j = 0; j < k; j++) y[j] = b[j];
        dft(x), dft(y);
        for (int j = 0; j < k * 2; j++) x[j] = (1LL * x[j] * r[j] + 1LL * y[j] * l[j]) % P;
        idft(x);
        for (int j = 0; j < k * 2; j++) a[j] = x[j];
      }
    q.resize(raw_n);
    return q.rev();
  }
};
```

12.FFT

```cpp
#define fp(i, a, b) for (int i = a, i##_ = (b) + 1; i < i##_; ++i)
using ll = int64_t;
using db = double;
/*-------------------------------------------------------------------*/
struct cp {
    db x, y;
    cp(db real = 0, db imag = 0) : x(real), y(imag){};
    cp operator+(cp b) const { return {x + b.x, y + b.y}; }
    cp operator-(cp b) const { return {x - b.x, y - b.y}; }
    cp operator*(cp b) const { return {x * b.x - y * b.y, x * b.y + y * b.x}; }
};
using vcp = vector<cp>;
using Poly = vector<ll>;
namespace FFT {
    const db pi = acos(-1);
    vcp Omega(int L) {
        vcp w(L); w[1] = 1;
        for (int i = 2; i < L; i <<= 1) {
            auto w0 = w.begin() + i / 2, w1 = w.begin() + i;
            cp wn(cos(pi / i), sin(pi / i));
            for (int j = 0; j < i; j += 2)
                w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
        }
        return w;
    }
    auto W = Omega(1 << 20); // NOLINT
    void DIF(cp *a, int n) {
        cp x, y;
        for (int k = n >> 1; k; k >>= 1)
            for (int i = 0; i < n; i += k << 1)
                for (int j = 0; j < k; ++j)
                    x = a[i + j], y = a[i + j + k],
                    a[i + j + k] = (a[i + j] - y) *  W[k + j], a[i + j] = x + y;
    }
    void IDIT(cp *a, int n) {
        cp x, y;
        for (int k = 1; k < n; k <<= 1)
            for (int i = 0; i < n; i += k << 1)
                for (int j = 0; j < k; ++j)
                    x = a[i + j], y = a[i + j + k] * W[k + j],
                    a[i + j + k] = x - y, a[i + j] = x + y;
        const db Inv = 1. / n;
        fp(i, 0, n - 1) a[i].x *= Inv, a[i].y *= Inv;
        reverse(a + 1, a + n);
    }
}
namespace Polynomial {
    // basic operator
    void DFT(vcp &a) { FFT::DIF(a.data(), a.size()); }
    void IDFT(vcp &a) { FFT::IDIT(a.data(), a.size()); }
    int norm(int n) { return 1 << (__lg(n - 1) + 1); }

    // Poly mul
    vcp &dot(vcp &a, vcp &b) { fp(i, 0, a.size() - 1) a[i] = a[i] * b[i]; return a; }
    Poly operator*(Poly &a, Poly &b) {
        int n = a.size() + b.size() - 1;
        vcp c(norm(n));
        fp(i, 0, a.size() - 1) c[i].x = a[i];
        fp(i, 0, b.size() - 1) c[i].y = b[i];
        DFT(c), dot(c, c), IDFT(c), a.resize(n);
        fp(i, 0, n - 1) a[i] = ll(c[i].y * .5 - .5);
        return a;
    }
}
using namespace Polynomial;
//f = f * g  ==> vector<ll> = vector<ll> * vector<ll>
```

13.FWT

```cpp
#include<iostream>
#define rep(i,a,b) for(int i=a;i<(int)b;i++)
typedef long long ll;
const ll mod = 1000000007;
const int maxn = 2e6 + 9;
#define fwt_loop for(int i=1,j,k;i<n;i*=2)\
    for(j=0;j<n;j+=2*i) for(k=j;k<j+i;k++)
void fwt_or(ll a[], int n, ll x) {
    fwt_loop a[i + k] = (a[i + k] + a[k] * x) % mod;
}
void fwt_and(ll a[], int n, ll x) {
    fwt_loop a[k] = (a[k] + a[i + k] * x) % mod;
}
void fwt_xor(ll a[], int n, ll x) {
    fwt_loop{
        ll y = a[k],z = a[i + k];
        a[k] = (y + z) * x % mod;
        a[i + k] = (y + mod - z) * x % mod;
    }
}
void solve(ll a[], ll b[], ll A[], ll B[], int len, int ty) {
    auto fwt = ty < 2 ? (ty ? fwt_and : fwt_or) : fwt_xor;
    rep(i, 0, len) a[i] = A[i], b[i] = B[i];
    fwt(a, len, 1), fwt(b, len, 1);
    rep(i, 0, len) a[i] = a[i] * b[i] % mod;
    fwt(a, len, ty < 2 ? mod - 1 : (mod + 1) / 2);
    //rep(i, 0, len) printf("%lld ", a[i]);
    //puts("");
}
ll A[maxn], B[maxn], a[maxn], b[maxn];
int main() {
    int n,k,d;
    ll x;
    scanf("%d", &k);
    n = 1 << 19;
    rep(i, 0, k) {scanf("%lld", &x);B[x]=1;d^=x;}
    A[0]=1;
    //rep(i, 0, n) scanf("%lld", &B[i]);
    for(int i=0;i<=20;i++){
        if(A[d]){
            printf("%d",k-i);break;
        }
        solve(a, b, A, B, n, 3);
        for(int i=0;i<n;i++){
            A[i]=a[i];
        }
    }

}
```