Standard Code Library

F0RE1GNERS

Jiangxi Normal University HeartFireY

September 1202

Contents

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 $a_n = ka_{n-1} - a_{n-2}$（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，$h(a,b,c,n)=(\frac{a}{c})^2n(n+1)(2n+1)/6+(\frac{b}{c})^2(n+1)+(\frac{a}{c})(\frac{b}{c})n(n+1)+h(a \bmod c, b \bmod c, c, n)+2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n)+2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$；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
- $f_n=\frac{\binom{2n}{n}}{n+1}$

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

```
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)

```
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 $k\left\{\begin{matrix}n-1\\k\end{matrix}\right\}$ 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

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

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 $\left\{\begin{matrix}n\\i\end{matrix}\right\}$ 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 $\left\{\begin{matrix}n\\i\end{matrix}\right\}$ 的 $i!$ 倍。于是

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

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 $\left\{\begin{matrix}n\\i\end{matrix}\right\}$。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 $F(x)=\sum_{i=1}^{+\infty}\frac{x^i}{i!}=e^x-1$。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，$\exp F(x)=\sum_{i=0}^{+\infty}\frac{F^i(x)}{i!}$ 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。

```cpp
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

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

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

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

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

```
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();
  }
};
```

数据结构

1. 离散化

数组版

```
for(int i = 1; i <= n; i++) std::cin >> a[i], b[i] = a[i];
std::sort(a + 1, a + 1 + n);
len = std::unique(a + 1, a + 1 + n) - a - 1;
auto query_pos = [&](int x) { return std::lower_bound(a + 1, a + 1 + len, x) - a; };
```

向量版

```
vector<int> a;
std::sort(a.begin(),a.end());
a.erase(unique(a.begin(),a.end()),a.end());
auto query_pos = [&](int x) { return lower_bound(a.begin(), a.end(), c) - a.begin() + 1;
```

```
//查找下标从 1 开始
}
```

2. 并查集

路径压缩 + 按秩合并

```
struct UnionFind {
    std::vector<int> par, rank, size;
    int c;
    UnionFind(int n) : par(n), rank(n, 0), size(n, 1), c(n) {
        for(int i = 0; i < n; ++i) par[i] = i;
    }
    int find(int i) { return (par[i] == i ? i : (par[i] = find(par[i]))); }
    bool same(int i, int j) { return find(i) == find(j); }
    int get_size(int i) { return size[find(i)]; }
    int count() { return c; }
    int merge(int i, int j) {
        if((i = find(i)) == (j = find(j))) return -1;
        else --c;
        if(rank[i] > rank[j]) std::swap(i, j);
        par[i] = j, size[j] += size[i];
        if(rank[i] == rank[j]) rank[j]++;
        return j;
    }
};
```

可回滚并查集

- 注意这个不是可持久化并查集
- 查找时不进行路径压缩
- 复杂度靠按秩合并解决

```
namespace uf {
    int fa[maxn], sz[maxn];
    int undo[maxn], top;
    void init() { memset(fa, -1, sizeof fa); memset(sz, 0, sizeof sz); top = 0; }
    int findset(int x) { while (fa[x] != -1) x = fa[x]; return x; }
    bool join(int x, int y) {
        x = findset(x); y = findset(y);
        if (x == y) return false;
        if (sz[x] > sz[y]) swap(x, y);
        undo[top++] = x;
        fa[x] = y;
        sz[y] += sz[x] + 1;
        return true;
    }
    inline int checkpoint() { return top; }
    void rewind(int t) {
        while (top > t) {
            int x = undo[--top];
            sz[fa[x]] -= sz[x] + 1;
            fa[x] = -1;
        }
    }
}
```

3.ST 表

- 预处理：$O(n \log n)$
- 查询：$O(1)$

```
#include <bits/stdc++.h>
using namespace std;

int n, q;

namespace ST{
    const int N = 2000010;
```

```
    int stmax[N][22], stmin[N][22], mn[N], a[N];

    void init(int n){
        mn[0] = -1;
        for (int i = 1; i <= n; i++){
            mn[i] = ((i & (i - 1)) == 0) ? mn[i - 1] + 1 : mn[i - 1];
            stmax[i][0] = stmin[i][0] = a[i];
        }
        for (int j = 1; j <= mn[n]; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++){
                stmax[i][j] = max(stmax[i][j - 1], stmax[i + (1 << (j - 1))][j - 1]);
                stmin[i][j] = min(stmin[i][j - 1], stmin[i + (1 << (j - 1))][j - 1]);
            }
    }

    inline int rmq_max(int L, int R){
        int k = mn[R - L + 1];
        return max(stmax[L][k], stmax[R - (1 << k) + 1][k]);
    }

    inline int rmq_min(int L, int R){
        int k = mn[R - L + 1];
        return min(stmin[L][k], stmin[R - (1 << k) + 1][k]);
    }
}

signed main(){
    cin >> n >> q;
    for(int i = 1; i <= n; i++) cin >> ST::a[i];
    ST::init(n);
    while(q--){
        int l, r; cin >> l >> r;
        cout << ST::rmq_max(l, r) << ' ' << ST::rmq_min(l, r) << endl;
    }
    return 0;
}
```

4. 树状数组 (Fenwick)

朴素树状数组

查询区间第 K 大需要权值树状数组!

```
namespace Fenwick{
    int tree[N], len;
    #define lowbit(x) ((x) & (-x))
    inline void init(){
        memset(tree, 0, sizeof(tree));
        for(int i = 1, tmp = 0; i <= len; i++){
            tree[i] += a[i];
            tmp = i + lowbit(i);
            if(tmp <= len) tree[tmp] += tree[i];
        }
    }

    inline void update(int i, int x){
        for(int pos = i; pos <= len; pos += lowbit(pos)) tree[pos] += x;
    }

    inline int getsum(int i, int ans = 0){
        for(int pos = i; pos; pos -= lowbit(pos)) ans += tree[pos];
        return ans;
    }

    inline int query(int l, int r){ return getsum(r) - getsum(l - 1); }

    //* 查询区间第 K 大需要权值树状数组!
    int kth(int i){
        int cnt = 0, ret = 0;
        for(int i = log2(len); ~i; --i){
            ret += 1 << i;
```

```
            (ret >= len || cnt + tree[ret] >= i) ? (ret -= 1 << i) : (cnt += tree[ret]);
        }
        return ret + 1;
    }
}
```

区间加/区间求和

若维护序列 a 的差分数组 b，此时我们对 a 的一个前缀 r 求和，即 $\sum_{i=1}^{r} a_i$，由差分数组定义得 $a_i = \sum_{j=1}^{i} b_j$

进行推导

$$\sum_{i=1}^{r} a_i$$

$$= \sum_{i=1}^{r} \sum_{j=1}^{i} b_j$$

$$= \sum_{i=1}^{r} b_i \times (r - i + 1)$$

$$= \sum_{i=1}^{r} b_i \times (r + 1) - \sum_{i=1}^{r} b_i \times i$$

区间和可以用两个前缀和相减得到，因此只需要用两个树状数组分别维护 $\sum b_i$ 和 $\sum i \times b_i$，就能实现区间求和。

```
namespace Fenwick_Plus{
    #define lowbit(x) ((x) & (-x))
    #define MAXN $Array_Max_Sizes$
    int tree1[MAXN], tree2[MAXN], a[MAXN], len;
    //对两个树状数组进行更新
    inline void add(int i, int x){
        int x1 = i * x;
        for(int pos = i; pos <= len; pos += lowbit(pos)) tree1[pos] += x, tree2[pos] += x1;
    }
    //将区间加差分为两个前缀和
    inline void update(int l, int r, int x){
        add(l, x), add(r + 1, -x);
    }
    //对指定的树状数组求前 n 项和
    inline int getsum(int *tree, int i){
        int sum = 0;
        for(int pos = i; pos; pos -= lowbit(pos)) sum += tree[i];
        return sum;
    }
    //区间和查询
    inline int query(int l, int r){
        return (r + 1) * getsum(tree1, r) - l * getsum(tree1, l - 1) - (getsum(tree2, r) - getsum(tree2, l - 1));
    }
}
```

4. 朴素线段树

区间和：区间修改/单点修改/单点查询/区间查询

```
#define SEGRG 1, 1, n //! 此处定义范围，注意 n 取值

const int N = 1e5 + 10;

namespace SegmentTree{
    #define ls rt << 1
    #define rs rt << 1 | 1
    #define lson rt << 1, l, mid
    #define rson rt << 1 | 1, mid + 1, r
    int len, q, tree[N << 2], lazy[N << 2];
    inline void push_up(int rt) { tree[rt] = tree[ls] + tree[rs]; }

    inline void push_down(int rt, int m){
        if(!lazy[rt]) return;
```

```
        lazy[ls] += lazy[rt], lazy[rs] += lazy[rt];
        tree[ls] += lazy[rt] * (m - (m >> 1));
        tree[rs] += lazy[rt] * (m >> 1);
        lazy[rt] = 0;
    }

    static void build(int rt, int l, int r){
        tree[rt] = lazy[rt] = 0;
        if(l == r){
            tree[rt] = 0; //!build leaf_node here
            return;
        }
        int mid = l + r >> 1;
        build(lson), build(rson);
        push_up(rt);
    }

    static void update_part(int rt, int l, int r, int L, int R, int val){
        if(l >= L && r <= R){
            lazy[rt] += val, tree[rt] += (r - l + 1) * val;
            return;
        }
        int mid = l + r >> 1;
        push_down(rt, r - l + 1);
        if(mid >= L) update_part(lson, L, R, val);
        if(mid < R) update_part(rson, L, R, val);
        push_up(rt);
    }

    static void update_point(int rt, int l, int r, int pos, int val){
        if(l == r){
            tree[rt] += val;
            return;
        }
        push_down(rt, r - l + 1);
        int mid = l + r >> 1;
        if(mid >= pos) update_point(lson, pos, val);
        else update_point(rson, pos, val);
        push_up(rt);
    }

    static int query(int rt, int l, int r, int L, int R){
        if(l >= L && r <= R) return tree[rt];
        int mid = l + r >> 1, ans = 0;
        push_down(rt, r - l + 1);
        if(mid >= L) ans += query(lson, L, R);
        if(mid < R) ans += query(rson, L, R);
        return ans;
    }
}
```

5. 动态开点线段树

```
namespace SegmentTree{
    const int N = 3e5 + 10;
    int tree[N << 2], lson[N << 2], rson[N << 2], lazy[N << 2], tot = 0, root = 0;

    inline void push_up(int rt){ tree[rt] = tree[lson[rt]] + tree[rson[rt]]; }
    inline void push_down(int rt, int m){
        if(!lazy[rt]) return;
        if(!lson[rt]) lson[rt] = ++tot;
        if(!rson[rt]) rson[rt] = ++tot;
        lazy[lson[rt]] += lazy[rt], lazy[rson[rt]] += lazy[rt];
        tree[lson[rt]] += lazy[rt] * (m - (m >> 1));
        tree[rson[rt]] += lazy[rt] * (m >> 1);
        lazy[rt] = 0;
    }

    static void update_part(int &rt ,int l, int r, int L, int R, int val){
        if(!rt) rt = ++tot;
        if(l >= L && r <= R){
```

```
            lazy[rt] += val;
            tree[rt] += val * (r - l + 1);
            return;
        }
        push_down(rt, r - l + 1);
        int mid = l + r >> 1;
        if(mid >= L) update_part(lson[rt], l, mid, L, R, val);
        if(mid < R) update_part(rson[rt], mid + 1, r, L, R, val);
        push_up(rt);
    }

    static void update_point(int &rt, int l, int r, int pos, int val){
        if(!rt) rt = ++tot;
        if(l == r){
            tree[rt] += val;
            return;
        }
        int mid = l + r >> 1;
        if(mid >= pos) update_point(lson[rt], l, mid, pos, val);
        else update_point(rson[rt], mid + 1, r, pos, val);
        push_up(rt);
    }

    static int query(int rt, int l, int r, int L, int R){
        if(!rt) return 0;
        if(l >= L && r <= R) return tree[rt];
        push_down(rt, r - l + 1);
        int mid = l + r >> 1, ans = 0;
        if(mid >= L) ans += query(lson[rt], l, mid, L, R);
        if(mid < R) ans += query(rson[rt], mid + 1, r, L, R);
        return ans;
    }
}//DynamicSegmentTree
```

6. 可持久化权值线段树 (主席树)

0. 模板题/模板封装

给定排列 $p_1, p_2, p_3, \dots, p_n$, 定义 A_i 表示在 p_i 左侧并比 p_i 小的数字个数, B_i 表示在 p_i 右侧并比 p_i 小的数字个数, $C_i = \min(A_i, B_i)$。现在给定多个操作 (l, r)，求每个操作，交换 (p_i, p_j) 后的 $\sum C_i$。

首先考虑如何处理初始时的 C_i 值，观察到以下性质:

- 对于 A_i 值的求解过程类似求逆序对的思想，可以直接上树状数组维护，$O(n \log n)$ 求得全部的 A_i
- 由于是排列，$B_i = p_i - 1 - A_i$ 可以 $O(1)$ 求得
- 那么 $C_i = \min(A_i, B_i)$ 也是 $O(1)$ 得到的

由于每个询问相互独立，那么考虑交换 (p_l, p_r) 操作对 C_i 的影响:

- 对于 $[1, l), (r, n]$ 范围的数字，C_i 值一定不影响。因为交换操作均在单侧进行
- 对于 p_l，交换到 r 位置后，$A_l \to A_l + \quad [l, r] \quad p_l \quad$ ，B_l' 仍然可以直接求

  对于 p_r，交换到 l 位置后，$A_r \to A_r - \quad [l, r] \quad p_r \quad$ ，B_r' 仍然可以直接求

  如果我们在线询问 (主席树维护)，那么对于 p_l, p_r，实际上可以直接两个 $O(logn)$ 重新求。
- 那么重点是对于 $[l + 1, r - 1]$ 区间内的数字的 C_i 值变化，如何维护?
- 对于 $p_l \le p_i \le p_r$，如果 $A_i \le B_i$，则交换后 $A_i - 1, B_i + 1$，从而 $C_i - 1$

  对于 $p_l \ge p_i \ge p_r$，如果 $A_i \ge B_i$，则交换后 $A_i + 1, B_i - 1$，从而 $C_i - 1$
- 对于 $p_l \le p_i \le p_r$，如果 $A_i - 1 \ge B_i + 1, A_i \ge B_i$，则交换后 $C_i + 1$

  对于 $p_l \ge p_i \ge p_r$，如果 $A_i - 1 \le B_i + 1, A_i \le B_i$，则交换后 $C_i + 1$

那么对于以上四种情况，我们可以分别用四棵主席树进行维护。同时，对于 p_l, p_r 的贡献计算还需要支持区间 $< K$ 的数字个数查询，因此共需五棵主席树进行维护，复杂度 $O(m \times 4 \log n)$。

```cpp
namespace PresidentTree{
    int root[N], sum[N << 5][5], lc[N << 5], rc[N << 5], cnt;
    #define ls l, mid
    #define rs mid + 1, r

    void update(int &rt, int pre, int l, int r, int x, bset5 inc){
        rt = ++cnt, lc[rt] = lc[pre], rc[rt] = rc[pre];
        for(int i = 0; i <= 5; i++) sum[rt][i] = sum[pre][i] + (inc[i] ? 1 : 0);
        if(l == r) return;
        int mid = l + r >> 1;
        if(x <= mid) update(lc[rt], lc[rt], l, mid, x, inc);
        else update(rc[rt], rc[rt], mid + 1, r, x, inc);
    }

    int query(int st, int ed, int l, int r, int L, int R, int id){
        if(l == L && r == R) return sum[ed][id] - sum[st][id];
        int mid = l + r >> 1;
        if(mid >= R) return query(lc[st], lc[ed], l, mid, L, R, id);
        else if(mid >= L) return query(lc[st], lc[ed], l, mid, L, mid, id) + query(rc[st], rc[ed], mid + 1, r, mid + 1, R, id);
        else return query(rc[st], rc[ed], mid + 1, r, L, R, id);
    }

}
```

1. 主席树维护静态区间第 K 大

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 10;
int root[N], tot;
int lc[N << 5], rc[N << 5], sum[N << 5];
int a[N], b[N], n, m;

namespace cmt{
    void update(int &rt, int pre, int l, int r, int pos, int v){
        rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
        if(l == r) return;
        int mid = l + r >> 1;
        if(pos <= mid) update(lc[rt], lc[pre], l, mid, pos, v);
        else update(rc[rt], rc[pre], mid + 1, r, pos, v);
    }

    int query(int ql, int qr, int l, int r, int k){
        if(l == r) return l;
        int mid = l + r >> 1, summ = sum[lc[qr]] - sum[lc[ql]];
        if(summ >= k) return query(lc[ql], rc[qr], l, mid, k);
        else return query(rc[ql], rc[qr], mid + 1, r, k - summ);
    }
}

signed main(){
    cin >> n >> m;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        b[i] = a[i];
    }
    sort(b + 1, b + 1 + n);
    int n_1 = unique(b + 1, b + 1 + n) - (b + 1);
    for(int i = 1; i <= n; i++) cmt::update(root[i], root[i - 1], 1, n_1, lower_bound(b + 1, b + 1 + n_1, a[i]) - b, 1);
    for(int i = 1; i <= m; i++){
        int l, r, k; cin >> l >> r >> k;
        cout << b[cmt::query(root[l - 1], root[r], 1, n_1, k)] << endl;
    }
    return 0;
}
```

2. 主席树维护线段树区间修改 (标记永久化)

主席树可以维护线段树的区间修改，但是要求线段树动态开点。

在区间更新的时候，对每个点打永久化标记，对于跨越区间的点需要直接修改。查询时找到对应的完全覆盖区间加上这个区间的标记 *(区间长度)

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;

const int N = 1e5 + 10;
int root[N], lc[N << 6], rc[N << 6], tot = 0, cur = 0;
ll sum[N << 6], lazy[N << 6];
int n, m;

void build(int &rt, int l, int r){
    rt = ++tot, lazy[rt] = 0;
    if(l == r){
        scanf("%lld", &sum[rt]);
        return;
    }
    int mid = l + r >> 1;
    build(lc[rt], l, mid);
    build(rc[rt], mid + 1, r);
    sum[rt] = sum[lc[rt]] + sum[rc[rt]];
}

void update(int &rt, int pre, int l, int r, int L, int R, int c){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], lazy[rt] = lazy[pre], sum[rt] = sum[pre] + 1ll * (min(r, R) - max(l, L) + 1) * c;
    if(L >= l && R <= r){
        lazy[rt] += c;
        return;
    }
    int mid = L + R >> 1;
    if(l <= mid) update(lc[rt], lc[pre], l, r, L, mid, c);
    if(r > mid) update(rc[rt], rc[pre], l, r, mid + 1, R, c);
}

ll query(int rt, int L, int R, int l, int r){
    if(L >= l && R <= r) return sum[rt];
    int mid = L + R >> 1;
    ll ans = lazy[rt] * (min(r, R) - max(l, L) + 1);
    if(l <= mid) ans += query(lc[rt], L, mid, l, r);
    if(r > mid) ans += query(rc[rt], mid + 1, R, l, r);
    return ans;
}

signed main(){
    while(scanf("%d%d", &n, &m) != EOF){
        char op[10];
        cur = tot = 0;
        build(root[0], 1, n);
        while(m--){
            scanf("%s", op);
            if(op[0] == 'C'){
                int l, r, c;
                scanf("%d%d%d", &l, &r, &c);
                cur++;
                update(root[cur], root[cur - 1], l, r, 1, n, c);
            }
            else if(op[0] == 'Q'){
                int l, r;
                scanf("%d%d", &l, &r);
                printf("%lld\n", query(root[cur], 1, n, l, r));
            }
            else if(op[0] == 'H'){
                int l, r, h;
                scanf("%d%d%d", &l, &r, &h);
                printf("%lld\n", query(root[h], 1, n, l, r));
```

```
            }
            else if(op[0] == 'B'){
                scanf("%d", &cur);
            }
        }
    }
    return 0;
}
```

3. 求区间内不同的数字个数/求区间大于 K 的数字有多少

求区间内不同数字个数的问题可以转化为求区间内小于等于 K 的数字个数:

对于每个数字记录下一个最近的相同数字下标 $nxt[i]$，那么查询区间 $[L,R]$ 内不同数字的个数实际上就是在查询区间内 $nxt[i] > R$ 的个数 (下一个相同的数字位于区间之外)。那么现在不难发现对于给定区间 $[l,r]$，如果 $nxt[i] > r,\ i \in [l,r]$，那么表示与 i 相同数字点位于区间之外。那么求不同数字个数问题便转化为给定求所有满足 $l <= i <= r,\ nxt[i] > r$ 的个数。

那么我们只需要处理出 nxt 数组，然后用主席树对每个节点维护权值数组，然后区间查询数目即可。

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1e6 + 10;
int a[N], head[N], nxt[N];
int root[N], sum[N << 5], lc[N << 5], rc[N << 5], cnt;

inline int read(){
    int f = 1, x = 0; char s = getchar();
    while(s < '0'||s > '9'){ if(s =='-') f = -1; s = getchar(); }
    while(s >= '0' && s <= '9'){ x = x * 10 + s - '0'; s = getchar();}
    return x *= f;
}

void build(int &rt, int l, int r){
    rt = ++cnt;
    if(l == r) return;
    int mid = l + r >> 1;
    build(lc[rt], l, mid);
    build(rc[rt], mid + 1, r);
}

void update(int &rt, int pre, int l, int r, int x){
    rt = ++cnt, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
    if(l == r) return;
    int mid = l + r >> 1;
    if(x <= mid) update(lc[rt], lc[pre], l, mid, x);
    else update(rc[rt], rc[pre], mid + 1, r, x);
}

int query(int L, int R, int l, int r, int k){
    if(l == r) return sum[R] - sum[L];
    int mid = l + r >> 1, ans = 0;
    if(k <= mid) ans += query(lc[L], lc[R], l, mid, k) + sum[rc[R]] - sum[rc[L]];
    else ans += query(rc[L], rc[R], mid + 1, r, k);
    return ans;
}

signed main(){
    int n = 0;
    n = read();
    for(int i = 1; i <= n; i++){
        a[i] = read();  //cin >> a[i];
        if(head[a[i]]) nxt[head[a[i]]] = i;
        head[a[i]] = i;
    }
    for(int i = 1; i <= n; i++)
        if(!nxt[i]) nxt[i] = n + 1;

    build(root[0], 1, n + 1);
    for(int i = 1; i <= n; i++) update(root[i], root[i - 1], 1, n + 1, nxt[i]);

```

```
    int m = read();
    while(m--){
        int l = read(), r = read();
        printf("%d\n", query(root[l - 1], root[r], 1, n + 1, r + 1));
    }
    return 0;
}
```

4. 求区间小于等于 K 的数字个数 (二分查询)

建立权值数组，对每个节点维护一颗主席树。查询为单点查询，查询数字对应的权值数组的个数。然后对于每个询问，我们在区间内二分枚举所有可能的数字 k，然后查询区间第 k 小。反复查询求得一个满足 $k \leq h$ 的最大 k。那么这个 k 就是我们想要得到的答案。

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;

const int N = 1e5 + 10;

ll root[N], sum[N << 5], lc[N << 5], rc[N << 5], tot = 0;
ll a[N], b[N];

inline void update(ll &rt, ll pre, ll l, ll r, ll k){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
    ll mid = l + r >> 1;
    if(l == r) return;
    if(k <= mid) update(lc[rt], lc[pre], l, mid, k);
    else update(rc[rt], rc[pre], mid + 1, r, k);
}

ll query(ll u, ll v, ll L, ll R, ll k){
    if(L == R) return L;
    ll mid = L + R >> 1;
    ll res = sum[lc[v]] - sum[lc[u]];
    if(res >= k) return query(lc[u], lc[v], L, mid, k);
    else return query(rc[u], rc[v], mid + 1, R, k - res);
}

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int t = 0, T = 0; cin >> t;
    while(t--){
        ll n, m; cin >> n >> m;
        for(int i = 1; i <= n; i++){
            cin >> a[i]; b[i] = a[i];
        }
        sort(b + 1, b + 1 + n);
        ll sz = unique(b + 1, b + 1 + n) - b - 1;
        for(int i = 1; i <= n; i++){
            ll x = lower_bound(b + 1, b + 1 + sz, a[i]) - b;
            update(root[i], root[i - 1], 1, sz, x);
        }
        cout << "Case " << ++T << ":" << endl;
        while(m--){
            ll u, v, k; cin >> u >> v >> k;
            u++, v++;
            ll ans = 0, L = 0, R = v - u + 1;
            while(L < R){
                ll mid = (L + R + 1) >> 1;
                ll t = query(root[u - 1], root[v], 1, sz, mid);
                if(b[t] <= k) L = mid;
                else R = mid - 1;
            }
            cout << L << endl;
        }
    }
    return 0;
}
```

5. 求区间 Mex

给定 n 长度的数组，$\{a_1, a_2, ..., a_n\}$，以及 m 次询问，每次给出一个数对 (l, r) 表示区间起点终点，要求对于给定的询问，回答在该区间内最小未出现的数字。

建立权值数组，对于每个点建立一棵主席树，维护权值最后一次出现的位置，那么对于查询 $[l, r]$ 就是查找第 r 棵树上出现位置小于 l 的权值，那么只需要维护最后一次出现位置的最小值即可。

主席树解决该问题属于在线算法。这种题目可以用莫队强制离线处理。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 10;

int tot, root[N], tree[N << 5], lc[N << 5], rc[N << 5];

void update(int &rt, int pre, int l, int r, int x, int val){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre];
    if(l == r){
        tree[rt] = val;
        return;
    }
    int mid = l + r >> 1;
    if(x <= mid) update(lc[rt], lc[pre], l, mid, x, val);
    else update(rc[rt], rc[pre], mid + 1, r, x, val);
    tree[rt] = min(tree[lc[rt]], tree[rc[rt]]);
}

int query(int rt, int ql, int l, int r){
    if(l == r) return l;
    int mid = l + r >> 1;
    if(tree[lc[rt]] < ql) return query(lc[rt], ql, l, mid);
    else return query(rc[rt], ql, mid + 1, r);
}


signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int n, m; cin >> n >> m;
    for(int i = 1, x; i <= n; i++){
        cin >> x; x++;
        if(x > n) root[i] = root[i - 1];
        else update(root[i], root[i - 1], 1, n + 1, x, i);
    }
    while(m--){
        int l, r; cin >> l >> r;
        cout << query(root[r], l, 1, n + 1) - 1 << endl;
    }
    return 0;
}
```

6. 求区间内出现次数大于 >=k 次的最前数

对于给定的序列，输出待查询区间内出现次数严格大于区间长度一半的数字。

思路：考虑对查询过程进行剪枝，排除非法子树，向合法子树搜索。

首先考虑非法状态：因为对于主席树上任意一个节点，其代表的意义是管辖区间内数字的个数。因此对于主席树上某个节点，如果其代表区间数字的数目比区间长度的一半 (也就是 $\frac{r-l+1}{2}$) 要小，那么子区间不回再出现满足该条件的数，在这种情况下可以直接返回 0。

剩下的部分就是查询的板子。非法状态实际上就是在对查询过程进行剪枝。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 5e5 + 10;

int a[N], b[N];
int tot, root[N << 5], sum[N << 5], lc[N << 5], rc[N << 5];

```

```cpp
void update(int &rt, int pre, int l, int r, int v){
    rt = ++tot, lc[rt] = lc[pre], rc[rt] = rc[pre], sum[rt] = sum[pre] + 1;
    if(l == r) return;
    int mid = l + r >> 1;
    if(v <= mid) update(lc[rt], lc[pre], l, mid, v);
    else update(rc[rt], rc[pre], mid + 1, r, v);
}

int query(int L, int R, int l, int r, int k){
    if(sum[R] - sum[L] <= k) return 0;
    if(l == r) return l;
    int now = sum[lc[R]] - sum[lc[L]], mid = l + r >> 1;
    if(now > k) return query(lc[L], lc[R], l, mid, k);
    else return query(rc[L], rc[R], mid + 1, r, k);
}

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int n, q; cin >> n >> q;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        b[i] = a[i];
    }
    sort(b + 1, b + n + 1);
    int m = unique(b + 1, b + n + 1) - b - 1;
    for(int i = 1; i <= n; i++){
        int x = lower_bound(b + 1, b + m + 1, a[i]) - b;
        update(root[i], root[i - 1], 1, m, x);
    }
    while(q--){
        int l, r; cin >> l >> r;
        int k = (r - l + 1) >> 1;
        cout << b[query(root[l - 1], root[r], 1, m, k)] << endl;
    }
    return 0;
}
```

7. 主席树 + 树上路径

给定一棵 n 个节点的树，每个点有一个权值。有 m 个询问，每次给你 u, v, k 你需要回答 u xor last 和 v 这两个节点间第 k 小的点权。

动态查询树上区间点权第 k 小，且查询之间具有关系，因此考虑建立主席树维护区间信息。

首先回顾主席树维护线性区间第 k 大/小时我们的处理思路：

对于全局区间第 k 小时，我们建立全局权值线段树，维护的区间和表示某点子树中点的个数。那么我们在寻找区间第 k 小时，只需要左右二分寻找即可。而对于某个区间的第 k 小，一个朴素的方式便是我们每次都建立一颗线段树，但显然这样是不明智的算法。那么我们是如何查询这个区间第 k 小的呢?

对于这个维护的过程我们很容易联想到前缀和的概念，我们可以先离线建树，对于每个点建立一棵主席树，维护 $[1, i]$ 区间，那么对于区间查询 $[l, r]$ 时，我们只需要查询到区间 $[1, l-1]$ 和 $[1, r]$ 即可取得区间的信息，实现区间查询。

然后分析样例，作出样例所示的树 (假设以 1 为根节点)：

那么我们可以发现，对于树上区间查询，我们也可以利用类似于线性区间查询的思路进行解决，但是由于树的结构限制，我们把线性区间的前缀和改为树上前缀和的形式：$query(u, v) = sum[u] + sum[v] - sum[lca(u, v)] - sum[fa[lca(u, v)]]$ 下面我们来说明这个式子：

如上，从根节点到 5 号节点的路径 + 从根节点到 7 号节点的路径重复了两次，那么我们要减去重叠的信息：对于根节点到交点父节点的信息均重复两次，到交点的信息重复一次 (因为交点在链上，需要保留一次信息)，因此前缀和形式便是 $sum[u] + sum[v] - sum[lca(u, v)] - sum[fa[lca(u, v)]]$。

```cpp
#include <bits/stdc++.h>
#define id(x) (lower_bound(b + 1, b + le + 1, a[x]) - b)
#define rid(x) (b[x])
using namespace std;

const int N = 1e5 + 10;

int n, m, le, ans = 0, lastans = 0;
```

```cpp
int a[N], b[N], f[N][19], dep[N];
vector<int> g[N];

struct node{ int sum, lc, rc; }tree[N << 5];
int root[N], cnt = 0;


void build(node &rt, int l, int r){
    rt.sum = 0;
    if(l == r) return;
    int mid = l + r >> 1;
    build(tree[rt.lc = ++cnt], l, mid);
    build(tree[rt.rc = ++cnt], mid + 1, r);
}

inline void update(node pre, node &rt, int l, int r, int p){
    rt.sum = pre.sum + 1;
    if(l == r) return;
    int mid = l + r >> 1;
    if(p <= mid) update(tree[pre.lc], tree[rt.lc = ++cnt], l, mid, p), rt.rc= pre.rc;
    else update(tree[pre.rc], tree[rt.rc = ++cnt], mid + 1, r, p), rt.lc = pre.lc;
}

inline int query(node u, node v, node lca, node lca_fa, int l, int r, int k){
    if(l == r) return l;
    int sum = tree[u.lc].sum + tree[v.lc].sum - tree[lca.lc].sum - tree[lca_fa.lc].sum;
    int mid = l + r >> 1;
    if(sum >= k) return query(tree[u.lc], tree[v.lc], tree[lca.lc], tree[lca_fa.lc], l, mid, k);
    return query(tree[u.rc], tree[v.rc], tree[lca.rc], tree[lca_fa.rc], mid + 1, r, k - sum);
}

inline void dfs(int u, int fa){
    update(tree[root[fa]], tree[root[u] = ++cnt], 1, le, id(u));
    f[u][0] = fa;
    dep[u] = dep[fa] + 1;
    for(register int i = 1; i <= 18; i++) f[u][i] = f[f[u][i - 1]][i - 1];
    for(auto v : g[u]){
        if(v == fa) continue;
        dfs(v, u);
    }
}

inline int lca(int u, int v){
    if(dep[u] < dep[v]) swap(u, v);
    for(register int i = 18; i >= 0; i--)
        if(dep[f[u][i]] >= dep[v]) u = f[u][i];
    if(u == v) return u;
    for(register int i = 18; i >= 0; i--)
        if(f[u][i] != f[v][i]) u = f[u][i], v = f[v][i];
    return f[u][0];
}

inline int querypath(int u, int v, int k){
    int lcaa = lca(u, v);
    return rid(query(tree[root[u]], tree[root[v]], tree[root[lcaa]], tree[root[f[lcaa][0]]], 1, le, k));
}

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    //freopen("stdin.in", "r", stdin);
    //freopen("stdout.out", "w", stdout);
    cin >> n >> m;
    for(register int i = 1; i <= n; i++){ cin >> a[i]; b[i] = a[i]; }
    for(register int i = 1, u, v; i < n; i++){
        cin >> u >> v;
        g[u].push_back(v), g[v].push_back(u);
    }
    sort(b + 1, b + 1 + n);
    le = unique(b + 1, b + n + 1) - (b + 1);
    build(tree[root[0] = ++cnt], 1, le);
    dfs(1, 0);
```

```
    while(m--){
        int u, v, k; cin >> u >> v >> k;
        ans = querypath(u ^ lastans, v, k);
        cout << ans << endl;
        lastans = ans;
    }
    return 0;
}
```

7. 树套树

树状数组套主席树

动态区间第 k 大

```
typedef vector<int> VI;
struct TREE {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    struct P {
        int w, ls, rs;
    } tr[maxn * 20 * 20];
    int sz = 1;
    TREE() { tr[0] = {0, 0, 0}; }
    int N(int w, int ls, int rs) {
        tr[sz] = {w, ls, rs};
        return sz++;
    }
    int add(int tt, int l, int r, int x, int d) {
        if (x < l || r < x) return tt;
        const P& t = tr[tt];
        if (l == r) return N(t.w + d, 0, 0);
        return N(t.w + d, add(t.ls, lson, x, d), add(t.rs, rson, x, d));
    }
    int ls_sum(const VI& rt) {
        int ret = 0;
        FOR (i, 0, rt.size())
            ret += tr[tr[rt[i]].ls].w;
        return ret;
    }
    inline void ls(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].ls; }); }
    inline void rs(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].rs; }); }
    int query(VI& p, VI& q, int l, int r, int k) {
        if (l == r) return l;
        int w = ls_sum(q) - ls_sum(p);
        if (k <= w) {
            ls(p); ls(q);
            return query(p, q, lson, k);
        }
        else {
            rs(p); rs(q);
            return query(p, q, rson, k - w);
        }
    }
} tree;
struct BIT {
    int root[maxn];
    void init() { memset(root, 0, sizeof root); }
    inline int lowbit(int x) { return x & -x; }
    void update(int p, int x, int d) {
        for (int i = p; i <= m; i += lowbit(i))
            root[i] = tree.add(root[i], 1, m, x, d);
    }
    int query(int l, int r, int k) {
        VI p, q;
        for (int i = l - 1; i > 0; i -= lowbit(i)) p.push_back(root[i]);
        for (int i = r; i > 0; i -= lowbit(i)) q.push_back(root[i]);
        return tree.query(p, q, 1, m, k);
    }
} bit;
```

```
void init() {
    m = 10000;
    tree.sz = 1;
    bit.init();
    FOR (i, 1, m + 1)
        bit.update(i, a[i], 1);
}
```

8.K-D Tree

在一个初始值全为 0 的 $n \times n$ 的二维矩阵上，进行 q 次操作，每次操作为以下两种之一：

1. `1 x y A`：将坐标 (x, y) 上的数加上 A。
2. `2 x1 y1 x2 y2`：输出以 (x_1, y_1) 为左下角，(x_2, y_2) 为右上角的矩形内（包括矩形边界）的数字和。

强制在线。内存限制 `20M`。保证答案及所有过程量在 `int` 范围内。

构建 2-D Tree，支持两种操作：添加一个 2 维点；查询矩形区域内的所有点的权值和。可以使用 **带重构**的 k-D Tree 实现。

在查询矩形区域内的所有点的权值和时，仍然需要记录子树内每一维度上的坐标的最大值和最小值。如果当前子树对应的矩形与所求矩形没有交点，则不继续搜索其子树；如果当前子树对应的矩形完全包含在所求矩形内，返回当前子树内所有点的权值和；否则，判断当前点是否在所求矩形内，更新答案并递归在左右子树中查找答案。

已经证明，如果在 $2-D$ 树上进行矩阵查询操作，已经被完全覆盖的子树不会继续查询，则单次查询时间复杂度是最优 $O(\log n)$，最坏 $O(\sqrt{n})$ 的。将结论扩展到 k 维的情况，则最坏时间复杂度是 $O(n^{1-\frac{1}{k}})$ 的。

```
const int maxn = 200010;
int n, op, xl, xr, yl, yr, lstans;

struct node{ int x, y, v; } s[maxn];

bool cmp1(int a, int b) { return s[a].x < s[b].x; }

bool cmp2(int a, int b) { return s[a].y < s[b].y; }

double a = 0.725;
int rt, cur, d[maxn], lc[maxn], rc[maxn], L[maxn], R[maxn], D[maxn], U[maxn],
    siz[maxn], sum[maxn];
int g[maxn], t;

void print(int x){
    if (!x) return;
    print(lc[x]);
    g[++t] = x;
    print(rc[x]);
}

void maintain(int x){
    siz[x] = siz[lc[x]] + siz[rc[x]] + 1;
    sum[x] = sum[lc[x]] + sum[rc[x]] + s[x].v;
    L[x] = R[x] = s[x].x;
    D[x] = U[x] = s[x].y;
    if (lc[x])
        L[x] = min(L[x], L[lc[x]]), R[x] = max(R[x], R[lc[x]]),
        D[x] = min(D[x], D[lc[x]]), U[x] = max(U[x], U[lc[x]]);
    if (rc[x])
        L[x] = min(L[x], L[rc[x]]), R[x] = max(R[x], R[rc[x]]),
        D[x] = min(D[x], D[rc[x]]), U[x] = max(U[x], U[rc[x]]);
}

int build(int l, int r){
    if (l > r) return 0;
    int mid = (l + r) >> 1;
    double av1 = 0, av2 = 0, va1 = 0, va2 = 0;
    for (int i = l; i <= r; i++)
        av1 += s[g[i]].x, av2 += s[g[i]].y;
    av1 /= (r - l + 1);
    av2 /= (r - l + 1);
    for (int i = l; i <= r; i++)
```

```
        va1 += (av1 - s[g[i]].x) * (av1 - s[g[i]].x), va2 += (av2 - s[g[i]].y) * (av2 - s[g[i]].y);
    if (va1 > va2) nth_element(g + l, g + mid, g + r + 1, cmp1), d[g[mid]] = 1;
    else nth_element(g + l, g + mid, g + r + 1, cmp2), d[g[mid]] = 2;
    lc[g[mid]] = build(l, mid - 1);
    rc[g[mid]] = build(mid + 1, r);
    maintain(g[mid]);
    return g[mid];
}

void rebuild(int &x){
    t = 0;
    print(x);
    x = build(1, t);
}

bool bad(int x) { return a * siz[x] <= (double)max(siz[lc[x]], siz[rc[x]]); }

void insert(int &x, int v){
    if (!x){
        x = v;
        maintain(x);
        return;
    }
    if (d[x] == 1){
        if (s[v].x <= s[x].x) insert(lc[x], v);
        else insert(rc[x], v);
    }
    else{
        if (s[v].y <= s[x].y) insert(lc[x], v);
        else insert(rc[x], v);
    }
    maintain(x);
    if (bad(x)) rebuild(x);
}

int query(int x){
    if (!x || xr < L[x] || xl > R[x] || yr < D[x] || yl > U[x]) return 0;
    if (xl <= L[x] && R[x] <= xr && yl <= D[x] && U[x] <= yr) return sum[x];
    int ret = 0;
    if (xl <= s[x].x && s[x].x <= xr && yl <= s[x].y && s[x].y <= yr) ret += s[x].v;
    return query(lc[x]) + query(rc[x]) + ret;
}

int main(){
    scanf("%d", &n);
    while (~scanf("%d", &op)){
        if (op == 1){
            cur++, scanf("%d%d%d", &s[cur].x, &s[cur].y, &s[cur].v);
            s[cur].x ^= lstans;
            s[cur].y ^= lstans;
            s[cur].v ^= lstans;
            insert(rt, cur);
        }
        if (op == 2){
            scanf("%d%d%d%d", &xl, &yl, &xr, &yr);
            xl ^= lstans;
            yl ^= lstans;
            xr ^= lstans;
            yr ^= lstans;
            printf("%d\n", lstans = query(rt));
        }
        if (op == 3) return 0;
    }
}
/* Test Case
//INPUT
4
1 2 3 3
2 1 1 3 3
1 1 1 1
2 1 1 0 7
```

```
3
//OUTPUT
3
5
*/
```

9.Trie(字典树)

A.0-1 Trie

01 字典树主要用于解决求异或最值的问题

```
namespace Trie01{
    int tot, nxt[20*N][2];
    inline void clear() { tot = nxt[0][0] = nxt[0][1] = 0; }
    void insert(int x) {
        int rt = 0;
        for (int i = 20; i >= 0; --i) {
            int cur = (x >> i & 1);
            if (!nxt[rt][cur]) {
                nxt[rt][cur] = ++tot;
                nxt[tot][0] = nxt[tot][1] = 0;
            }
            rt = nxt[rt][cur];
        }
    }

    int query_min(int x) {
        int rt = 0, ans = 0;
        for (int i = 20; i >= 0; -- i) {
            int cur = (x >> i & 1), need = cur;
            if (!nxt[rt][need]) ans = (ans << 1) | 1, need = !need;
            else ans = (ans << 1);
            rt = nxt[rt][need];
        }
        return ans;
    }

    int query_max(int x) {
        int rt = 0, ans = 0;
        for (int i = 20; i >= 0; -- i) {
            int cur = (x >> i & 1), need = (cur ^ 1);
            if (!nxt[rt][need]) ans = (ans << 1), need = !need;
            else ans = (ans << 1) | 1;
            rt = nxt[rt][need];
        }
        return ans;
    }
}
```

B.Normal Trie

给定 n 个模式串 $s_1, s_2, \ldots, s_n$ 和 q 次询问，每次询问给定一个文本串 t_i，请回答 $s_1 \sim s_n$ 中有多少个字符串 s_j 满足 t_i 是 s_j 的**前缀**。保证 $1 \leq T, n, q \leq 10^5$，且输入字符串的总长度不超过 3×10^6。

```
const int N = 3e6 + 10, MOD = 1e9 + 7;
const int DICT_SIZE = 65;
namespace Trie{
    int tot, nxt[N][DICT_SIZE], cnt[N];

    void clear(){
        for(int i = 0; i <= tot; i++){
            cnt[i] = 0;
            for(int j = 0; j <= 122; j++) nxt[i][j] = 0;
        }
        tot = 0;
    }

    int getnum(char x){
        if(x >= 'A' && x <= 'Z') return x - 'A';
```

```
        else if(x >= 'a' && x <= 'z') return x - 'a' + 26;
        else return x - '0' + 52;
    }

    void insert(char str[]){
        int p = 0, len = strlen(str);
        for(int i = 0; i < len; i++){
            int c = getnum(str[i]);
            if(!nxt[p][c]) nxt[p][c] = ++tot;
            p = nxt[p][c];
            cnt[p]++;
        }
    }

    int find(char str[]){
        int p = 0, len = strlen(str);
        for(int i = 0; i < len; i++){
            int c = getnum(str[i]);
            if(!nxt[p][c]) return 0;
            p = nxt[p][c];
        }
        return cnt[p];
    }
}

char s[N];

inline void solve(){
    Trie::clear();
    int n, q; cin >> n >> q;
    for(int i = 0; i < n; i++){
        cin >> s; Trie::insert(s);
    }
    for(int i = 0; i < q; i++){
        cin >> s; cout << Trie::find(s) << endl;
    }
}
/* TEST CASE
//INPUT
3
3 3
fusufusu
fusu
anguei
fusu
anguei
kkksc
5 2
fusu
Fusu
AFakeFusu
afakefusu
fusuisnotfake
Fusu
fusu
1 1
998244353
9
//OUTPUT
2
1
0
1
2
1
```

10. 笛卡尔树

给定一个 $1 \sim n$ 的排列 p，构建其笛卡尔树。

即构建一棵二叉树，满足：

1. 每个节点的编号满足二叉搜索树的性质。
2. 节点 i 的权值为 p_i，每个节点的权值满足小根堆的性质。

这棵树的每个结点有两个子树，分为左右子树，子树可以为空；

一个结点的左子树中的所有结点的第一个权值都小于其第一个权值（空子树也满足）；

一个结点的右子树中的所有结点的第一个权值都大于其第一个权值（空子树也满足）；

一个结点的两棵子树中的所有结点的第二个权值都大于其第二个权值（空子树也满足）。

```
const int N = 1e7 + 7;
int n, a[N], stk[N], ls[N], rs[N];

void build(int n){
    for (int i = 1, pos = 0, top = 0; i <= n; ++i){ //这是按下标顺序插入元素的代码
        pos = top;
        while (pos && a[stk[pos]] > a[i]) pos--;
        if (pos) rs[stk[pos]] = i;
        if (pos < top) ls[i] = stk[pos + 1];
        stk[top = ++pos] = i;
    }
}

inline void solve(){
    int n = read();
    for(int i = 1; i <= n; i++) a[i] = read();
    build(n);
    long long L = 0, R = 0;
    for (int i = 1; i <= n; ++i)
        L ^= 1LL * i * (ls[i] + 1), R ^= 1LL * i * (rs[i] + 1);
    printf("%lld %lld", L, R);
}
/*TEST CASE
//INPUT
5
4 1 3 2 5
//OUTPUT
19 21
*/
```

11.Treap

FROM ECNU 板子库

原始 Treap

- 非旋 Treap
- v 小根堆
- lower 第一个大于等于的是第几个 (0-based)
- upper 第一个大于的是第几个 (0-based)
- split 左侧分割出 rk 个元素

```
namespace treap {
    const int M = maxn * 17;
    extern struct P* const null;
    struct P {
        P *ls, *rs;
        int v, sz;
        unsigned rd;
        P(int v): ls(null), rs(null), v(v), sz(1), rd(rnd()) {}
        P(): sz(0) {}

        P* up() { sz = ls->sz + rs->sz + 1; return this; }
        int lower(int v) {
            if (this == null) return 0;
            return this->v >= v ? ls->lower(v) : rs->lower(v) + ls->sz + 1;
        }
```

```
        int upper(int v) {
            if (this == null) return 0;
            return this->v > v ? ls->upper(v) : rs->upper(v) + ls->sz + 1;
        }
    } *const null = new P, pool[M], *pit = pool;

    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (l->rd < r->rd) { l->rs = merge(l->rs, r); return l->up(); }
        else { r->ls = merge(l, r->ls); return r->up(); }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, o->ls); r = o->up(); }
        else { split(o->rs, rk - o->ls->sz - 1, o->rs, r); l = o->up(); }
    }
}
```

- 持久化 Treap

```
namespace treap {
    const int M = maxn * 17 * 12;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int v, sz;
        LL sum;
        P(P* ls, P* rs, int v): ls(ls), rs(rs), v(v), sz(ls->sz + rs->sz + 1),
                                                    sum(ls->sum + rs->sum + v) {}
        P() {}

        void* operator new(size_t _) { return pit++; }
        template<typename T>
        int rk(int v, T&& cmp) {
            if (this == null) return 0;
            return cmp(this->v, v) ? ls->rk(v, cmp) : rs->rk(v, cmp) + ls->sz + 1;
        }
        int lower(int v) { return rk(v, greater_equal<int>()); }
        int upper(int v) { return rk(v, greater<int>()); }
    } pool[M], *pit = pool, *const null = new P;
    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (rnd() % (l->sz + r->sz) < l->sz) return new P{l->ls, merge(l->rs, r), l->v};
        else return new P{merge(l, r->ls), r->rs, r->v};
    }
    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, r); r = new P{r, o->rs, o->v}; }
        else { split(o->rs, rk - o->ls->sz - 1, l, r); l = new P{o->ls, l, o->v}; }
    }
}
```

- 带 pushdown 的持久化 Treap
- 注意任何修改操作前一定要 FIX

```
int now;
namespace Treap {
    const int M = 10000000;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int sz, time;
        LL cnt, sc, pos, add;
        bool rev;

        P* up() { sz = ls->sz + rs->sz + 1; sc = ls->sc + rs->sc + cnt; return this; } // MOD
        P* check() {
            if (time == now) return this;
            P* t = new(pit++) P; *t = *this; t->time = now; return t;
        };
        P* _do_rev() { rev ^= 1; add *= -1; pos *= -1; swap(ls, rs); return this; } // MOD
```

```
        P* _do_add(LL v) { add += v; pos += v; return this; } // MOD
        P* do_rev() { if (this == null) return this; return check()->_do_rev(); } // FIX & MOD
        P* do_add(LL v) { if (this == null) return this; return check()->_do_add(v); } // FIX & MOD
        P* _down() { // MOD
            if (rev) { ls = ls->do_rev(); rs = rs->do_rev(); rev = 0; }
            if (add) { ls = ls->do_add(add); rs = rs->do_add(add); add = 0; }
            return this;
        }
        P* down() { return check()->_down(); } // FIX & MOD
        void _split(LL p, P*& l, P*& r) { // MOD
            if (pos >= p) { ls->split(p, l, r); ls = r; r = up(); }
            else          { rs->split(p, l, r); rs = l; l = up(); }
        }
        void split(LL p, P*& l, P*& r) { // FIX & MOD
            if (this == null) l = r = null;
            else down()->_split(p, l, r);
        }
    } pool[M], *pit = pool, *const null = new P;
    P* merge(P* a, P* b) {
        if (a == null) return b; if (b == null) return a;
        if (rand() % (a->sz + b->sz) < a->sz) { a = a->down(); a->rs = merge(a->rs, b); return a->up(); }
        else                                  { b = b->down(); b->ls = merge(a, b->ls); return b->up(); }
    }
}
```

Treap-序列

- 区间 ADD，SUM

```
namespace treap {
    const int M = 8E5 + 100;
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, val, add, sum;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), val(v), add(0), sum(v) {}
        P(): sz(0), val(0), add(0), sum(0) {}

        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            sum = ls->sum + rs->sum + val + add * sz;
            return this;
        }
        void upd(int v) {
            if (this == null) return;
            add += v;
            sum += sz * v;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                val += add;
                add = 0;
            }
            return this;
        }

        P* select(int rk) {
            if (rk == ls->sz + 1) return this;
            return ls->sz >= rk ? ls->select(rk) : rs->select(rk - ls->sz - 1);
        }
    } pool[M], *pit = pool, *const null = new P, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b->up();
        if (b == null) return a->up();
        if (rand() % (a->sz + b->sz) < a->sz) {
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
```

```
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= rk) {
            split(o->ls, rk, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, rk - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

    inline void insert(int k, int v) {
        P *l, *r;
        split(rt, k - 1, l, r);
        rt = merge(merge(l, new (pit++) P(v)), r);
    }

    inline void erase(int k) {
        P *l, *r, *_, *t;
        split(rt, k - 1, l, t);
        split(t, 1, _, r);
        rt = merge(l, r);
    }

    P* build(int l, int r, int* a) {
        if (l > r) return null;
        if (l == r) return new(pit++) P(a[l]);
        int m = (l + r) / 2;
        return (new(pit++) P(a[m], build(l, m - 1, a), build(m + 1, r, a)))->up();
    }
};
```

- 区间 REVERSE，ADD，MIN

```
namespace treap {
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, v, add, m;
        bool flip;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), v(v), add(0), m(v), flip(0) {}
        P(): sz(0), v(INF), m(INF) {}

        void upd(int v) {
            if (this == null) return;
            add += v; m += v;
        }
        void rev() {
            if (this == null) return;
            swap(ls, rs);
            flip ^= 1;
        }
        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            m = min(min(ls->m, rs->m), v) + add;
            return this;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                v += add;
                add = 0;
            }
            if (flip) {
                ls->rev(); rs->rev();
```

```
                flip = 0;
            }
            return this;
        }

        P* select(int k) {
            if (ls->sz + 1 == k) return this;
            if (ls->sz >= k) return ls->select(k);
            return rs->select(k - ls->sz - 1);
        }

    } pool[M], *const null = new P, *pit = pool, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rnd() % (a->sz + b->sz) < a->sz) {
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int k, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= k) {
            split(o->ls, k, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, k - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

    P* build(int l, int r, int* v) {
        if (l > r) return null;
        int m = (l + r) >> 1;
        return (new (pit++) P(v[m], build(l, m - 1, v), build(m + 1, r, v)))->up();
    }

    void go(int x, int y, void f(P*&)) {
        P *l, *m, *r;
        split(rt, y, l, r);
        split(l, x - 1, l, m);
        f(m);
        rt = merge(merge(l, m), r);
    }
}
using namespace treap;
int a[maxn], n, x, y, Q, v, k, d;
char s[100];

int main() {
    cin >> n;
    FOR (i, 1, n + 1) scanf("%d", &a[i]);
    rt = build(1, n, a);
    cin >> Q;
    while (Q--) {
        scanf("%s", s);
        if (s[0] == 'A') {
            scanf("%d%d%d", &x, &y, &v);
            go(x, y, [](P*& o){ o->upd(v); });
        } else if (s[0] == 'R' && s[3] == 'E') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o){ o->rev(); });
        } else if (s[0] == 'R' && s[3] == 'O') {
            scanf("%d%d%d", &x, &y, &d);
            d %= y - x + 1;
```

```
            go(x, y, [](P*& o){
                P *l, *r;
                split(o, o->sz - d, l, r);
                o = merge(r, l);
            });
        } else if (s[0] == 'I') {
            scanf("%d%d", &k, &v);
            go(k + 1, k, [](P*& o){ o = new (pit++) P(v); });
        } else if (s[0] == 'D') {
            scanf("%d", &k);
            go(k, k, [](P*& o){ o = null; });
        } else if (s[0] == 'M') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o) {
                printf("%d\n", o->m);
            });
        }
    }
}
```

- 持久化

```
namespace treap {
    struct P;
    extern P*const null;
    P* N(P* ls, P* rs, LL v, bool fill);
    struct P {
        P *const ls, *const rs;
        const int sz, v;
        const LL sum;
        bool fill;
        int cnt;

        void split(int k, P*& l, P*& r) {
            if (this == null) { l = r = null; return; }
            if (ls->sz >= k) {
                ls->split(k, l, r);
                r = N(r, rs, v, fill);
            } else {
                rs->split(k - ls->sz - fill, l, r);
                l = N(ls, l, v, fill);
            }
        }


    } *const null = new P{0, 0, 0, 0, 0, 0, 1};

    P* N(P* ls, P* rs, LL v, bool fill) {
        ls->cnt++; rs->cnt++;
        return new P{ls, rs, ls->sz + rs->sz + fill, v, ls->sum + rs->sum + v, fill, 1};
    }

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rand() % (a->sz + b->sz) < a->sz)
            return N(a->ls, merge(a->rs, b), a->v, a->fill);
        else
            return N(merge(a, b->ls), b->rs, b->v, b->fill);
    }

    void go(P* o, int x, int y, P*& l, P*& m, P*& r) {
        o->split(y, l, r);
        l->split(x - 1, l, m);
    }
}
```

12. 莫队

扩展顺序

```
while (l > q.l) mv(--l, 1);
while (r < q.r) mv(r++, 1);
while (l < q.l) mv(l++, -1);
while (r > q.r) mv(--r, -1);
```

13.CDQ 分治

```
const int maxn = 2E5 + 100;
struct P {
    int x, y;
    int* f;
    bool d1, d2;
} a[maxn], b[maxn], c[maxn];
int f[maxn];

void go2(int l, int r) {
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go2(l, m); go2(m, r);
    for(int i = l; i < m; i++) b[i].d2 = 0;
    for(int i = m; i < r; i++) b[i].d2 = 1;
    merge(b + l, b + m, b + m, b + r, c + l, [](const P& a, const P& b)->bool {
            if (a.y != b.y) return a.y < b.y;
            return a.d2 > b.d2;
        });
    int mx = -1;
    for(int i = l; i < r; i++) {
        if (c[i].d1 && c[i].d2) *c[i].f = max(*c[i].f, mx + 1);
        if (!c[i].d1 && !c[i].d2) mx = max(mx, *c[i].f);
    }
    for(int i = l; i < r; i++)  b[i] = c[i];
}

void go1(int l, int r) { // [l, r)
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go1(l, m);
    for(int i = l; i < m; i++) a[i].d1 = 0;
    for(int i = m; i < r; i++) a[i].d1 = 1;
    copy(a + l, a + r, b + l);
    sort(b + l, b + r, [](const P& a, const P& b)->bool {
            if (a.x != b.x) return a.x < b.x;
            return a.d1 > b.d1;
        });
    go2(l, r);
    go1(m, r);
}
```

14. 珂朵莉树/老司机树

```
struct node{
    int l, r;
    mutable int val;
    node (int lpos): l(lpos) {}
    node (int lpos, int rpos, int vall): l(lpos), r(rpos), val(vall) {}
    bool operator< (const node &a) const { return l < a.l; }
};

set<node> s;
using sit = set<node>::iterator;

sit split(int pos){
    sit it = s.lower_bound(node(pos));
    if(it != s.end() && it -> l == pos) return it;
    --it;
    int l = it -> l, r = it -> r, val = it -> val;
    s.erase(it);
    s.insert(node(l, pos - 1, val));
```

```
    return s.insert(node(pos, r, val)).first;
}

void assign(int l, int r, int val){
    sit itr = split(r + 1), itl = split(l);
    s.erase(itl, itr);
    s.insert(node(l, r, val));
}

void add(int l, int r, int val){
    sit itr = split(r + 1), itl = split(l);
    //for(auto it = itl; it != itr; it++) it -> val += val;
    while(itl != itr) itl -> val += val, itl++;
}

int kth(int l, int r, int k){
    sit itr = split(r + 1), itl = split(l);
    vector<pair<int, int>> v;
    v.clear();
    for(sit it = itl; it != itr; it++) v.emplace_back(make_pair(it -> val, it -> r - it -> l + 1));
    sort(v.begin(), v.end());
    for(int i = 0; i < v.size(); i++){
        k -= v[i].second;
        if(k <= 0) return v[i].first;
    }
    return -1;
}

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

int query(int l, int r, int x, int y){
    sit itr = split(r + 1), itl = split(l);
    int res(0);
    for(sit it = itl; it != itr; it++)
        res = (res + (it -> r - it -> l + 1) * binpow(it -> val, x, y)) % y;
    return res;
}

int n, m, vmax, seed;
int rnd() {
    int ret = (int)seed;
    seed = (seed * 7 + 13) % 1000000007;
    return ret;
}

inline void solve(){
    cin >> n >> m >> seed >> vmax;
    for(int i = 1; i <= n; i++){
        int a = rnd() % vmax + 1;
        s.insert(node{i, i, (int) a});
    }
    s.insert(node{n + 1, n + 1, 0});
    for(int i = 1; i <= m; i++){
        int l, r, x, y;
        int op = rnd() % 4 + 1;
        l = rnd() % n + 1, r = rnd() % n + 1;
        if(l > r) swap(l, r);
        if(op == 3) x = rnd() % (r - l + 1) + 1;
        else x = rnd() % vmax + 1;
        if(op == 4) y = rnd() % vmax + 1;
        if(op == 1) add(l, r, x);
        else if(op == 2) assign(l, r, x);
        else if(op == 3) cout << kth(l, r, x) << endl;
        else if(op == 4) cout << query(l, r, x, y) << endl;
    }
}
```

exSTL

优先队列

- binary_heap_tag
- pairing_heap_tag 支持修改
- thin_heap_tag 如果修改只有 increase 则较快，不支持 join

```cpp
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

typedef __gnu_pbds::priority_queue<LL, less<LL>, pairing_heap_tag> PQ;
__gnu_pbds::priority_queue<int, cmp, pairing_heap_tag>::point_iterator it;
PQ pq, pq2;

int main() {
    auto it = pq.push(2);
    pq.push(3);
    assert(pq.top() == 3);
    pq.modify(it, 4);
    assert(pq.top() == 4);
    pq2.push(5);
    pq.join(pq2);
    assert(pq.top() == 5);
}
```

平衡树

- ov_tree_tag
- rb_tree_tag
- splay_tree_tag
- mapped: null_type 或 null_mapped_type（旧版本）为空
- Node_Update 为 tree_order_statistics_node_update 时才可以 find_by_order & order_of_key
- find_by_order 找 order + 1 小的元素（其实都是从 0 开始计数），或者有 order 个元素比它小的 key
- order_of_key 有多少个比 r_key 小的元素
- join & split

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using Tree = tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>;
Tree t;
```

持久化平衡树

```cpp
#include <ext/rope>
using namespace __gnu_cxx;
rope<int> s;

int main() {
    FOR (i, 0, 5) s.push_back(i); // 0 1 2 3 4
    s.replace(1, 2, s); // 0 (0 1 2 3 4) 3 4
    auto ss = s.substr(2, 2); // 1 2、
    s.erase(2, 2); // 0 1 4
    s.insert(2, s); // equal to s.replace(2, 0, s)
    assert(s[2] == s.at(2)); // 2
}
```

哈希表

```cpp
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace __gnu_pbds;

```

```
gp_hash_table<int, int> mp;
cc_hash_table<int, int> mp;
```