格雷码:

给定整数 `i`，其 **n 位格雷码** 为：

$$G(i) = i \oplus (i >> 1)$$

即当前数和其右移一位异或。

```
int grayToBinary(int g) {
    for (int mask = g >> 1; mask; mask >>= 1)
        g ^= mask;
    return g;
}

```

Miller-RaBin:

快速判断质数

```
#include <bits/stdc++.h>
using namespace std;
using u64 = unsigned long long;
using u128 = __uint128_t;

// 快速乘 (防止溢出)
u64 mul(u64 a, u64 b, u64 mod) {
    return (u128)a * b % mod;
}

// 快速幂
u64 power(u64 a, u64 d, u64 mod) {
    u64 res = 1;
    while (d) {
        if (d & 1) res = mul(res, a, mod);
        a = mul(a, a, mod);
        d >>= 1;
    }
    return res;
}

// Miller-Rabin 判断是否为质数
bool isPrime(u64 n) {
    if (n < 2) return false;
    static u64 testPrimes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    for (u64 p : testPrimes)
        if (n % p == 0) return n == p;

    // 分解 n-1 = d * 2^s
    u64 d = n - 1, s = 0;
    while ((d & 1) == 0) d >>= 1, ++s;

    // 保证对 64-bit 整数是确定性的底数集合
    for (u64 a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
```

```
        if (a % n == 0) continue;
        u64 x = power(a, d, n);
        if (x == 1 || x == n - 1) continue;
        bool composite = true;
        for (u64 r = 1; r < s; ++r) {
            x = mul(x, x, n);
            if (x == n - 1) { composite = false; break; }
        }
        if (composite) return false;
    }
    return true;
}

int main() {
    u64 n;
    cin >> n;
    cout << (isPrime(n) ? "Prime" : "Composite") << endl;
}

```

快速gcd:

```
#include <bits/stdc++.h>
using namespace std;
using u64 = unsigned long long;

// ⚡ 极致优化版 gcd：Stein算法（Binary GCD）
inline u64 fast_gcd(u64 a, u64 b) {
    if (!a) return b;
    if (!b) return a;
    int shift = __builtin_ctzll(a | b); // 公共2因子个数
    a >>= __builtin_ctzll(a);
    do {
        b >>= __builtin_ctzll(b);
        if (a > b) swap(a, b);
        b -= a;
    } while (b);
    return a << shift;
}

```

裴蜀定理:

`exgcd`（扩展欧几里得，返回 gcd 并求出 x,y 满足 ax+by=g）

模逆 `mod_inv`

线性同余方程求解 `solve_linear_congruence`（求解 `a*x ≡ b (mod m)`）

两个同余的 CRT（支持非互质模）与多模 CRT 接口

```

```

```cpp
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
using u64 = unsigned long long;

/*
 * exgcd: 扩展欧几里得
 * 输入 a, b（可为负）
 * 输出 g = gcd(|a|,|b|)，并通过引用返回 x,y，使得 a*x + b*y = g
 * 复杂度 O(log min(|a|,|b|))
 */
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
    if (b == 0) {
        x = (a >= 0) ? 1 : -1; // 确保符号一致（可选）
        y = 0;
        return std::llabs(a);
    }
    i64 x1, y1;
    i64 g = exgcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return g;
}

/*
 * mod_inv: 模逆（若不存在返回 -1）
 * 返回 x 使得 (a * x) % mod == 1
 * 要求 mod > 0
 */
i64 mod_inv(i64 a, i64 mod) {
    i64 x, y;
    i64 g = exgcd(a, mod, x, y);
    if (g != 1) return -1;            // 不可逆
    x %= mod;
    if (x < 0) x += mod;
    return x;
}

/*
 * solve_linear_congruence:
 * 求解 a * x ≡ b (mod m)
 * 返回 pair(has_solution, pair(x0, mod')):
 *  - has_solution: 是否有解
 *  - x0: 一个解，使得所有解为 x = x0 + k * (m / g)
 *  - mod': 模长 m / g （解的步长）
 */
pair<bool, pair<i64,i64>> solve_linear_congruence(i64 a, i64 b, i64 m) {
    if (m < 0) m = -m;
    if (m == 0) { // 特殊：模为0 -> a*x = b over integers
        if (a == 0) return {b == 0, {0, 0}}; // 任意 x 或 无解
        if (b % a != 0) return {false, {0,0}};
        return {true, {b / a, 0}}; // 唯一解 x = b/a
    }
```

```cpp
    i64 x, y;
    i64 g = exgcd(a, m, x, y);
    if (b % g != 0) return {false, {0, 0}};
    i64 m2 = m / g;
    i64 x0 = ( (__int128)x * (b / g) ) % m2; // 避免溢出
    if (x0 < 0) x0 += m2;
    return {true, {x0, m2}};
}

/*
 * crt_pair: 两个同余合并（支持非互质模）
 * 解 x ≡ a1 (mod m1)
 *    x ≡ a2 (mod m2)
 * 返回 pair(has_solution, pair(x_mod, lcm_mod))
 * 若无解 has_solution=false
 * 若模为0的特殊处理同上（0 表示 "无模"，即等式为整数等式）
 */
pair<bool, pair<i64,i64>> crt_pair(i64 a1, i64 m1, i64 a2, i64 m2) {
    // 规范化：处理模为0的情况
    if (m1 < 0) m1 = -m1;
    if (m2 < 0) m2 = -m2;
    if (m1 == 0 && m2 == 0) {
        if (a1 == a2) return {true, {a1, 0}};
        return {false, {0, -1}};
    }
    if (m1 == 0) {
        // a1 must equal a2 (mod m2)
        if ((a1 - a2) % m2 == 0) return {true, {a1, 0}};
        return {false, {0, -1}};
    }
    if (m2 == 0) {
        if ((a2 - a1) % m1 == 0) return {true, {a2, 0}};
        return {false, {0, -1}};
    }

    // 一般情况
    i64 x, y;
    i64 g = exgcd(m1, m2, x, y);
    if ((a2 - a1) % g != 0) return {false, {0, -1}};
    i64 l = m1 / g * m2; // lcm
    // 计算乘子：(a2 - a1)/g * x mod (m2/g)
    i64 t = ( (__int128)(a2 - a1) / g * x ) % (m2 / g);
    if (t < 0) t += m2 / g;
    __int128 res = (__int128)a1 + (__int128)m1 * t;
    i64 r = (i64)(res % l);
    if (r < 0) r += l;
    return {true, {r, l}};
}

/*
 * crt: 多模 CRT（传入 a[], m[]）
 * 返回 (has_solution, pair(x_mod, mod))，与 crt_pair 约定一致
 */
```

```cpp
pair<bool, pair<i64,i64>> crt(const vector<i64>& a, const vector<i64>& m) {
    if (a.size() != m.size()) return {false, {0, -1}};
    pair<i64,i64> cur = {a[0], m[0]};
    for (size_t i = 1; i < a.size(); ++i) {
        auto res = crt_pair(cur.first, cur.second, a[i], m[i]);
        if (!res.first) return {false, {0, -1}};
        cur = res.second;
    }
    return {true, cur};
}

/* -------- 简短使用示例 --------
int main() {
    // exgcd
    i64 x, y;
    i64 g = exgcd(30, 18, x, y); // g=6, 30*x + 18*y = 6
    // mod_inv
    cout << mod_inv(3, 11) << "\n"; // 4
    // 线性同余
    auto sol = solve_linear_congruence(4, 6, 14); // 4x ≡ 6 (mod 14)
    if (sol.first) cout << "x0=" << sol.second.first << " step=" << sol.second.second
<< "\n";
    // CRT
    vector<i64> a = {2, 3, 2}, m = {3, 5, 7};
    auto crtres = crt(a, m);
    if (crtres.first) cout << "x = " << crtres.second.first << " (mod " <<
crtres.second.second << ")\n";
}
*/
```

拓展中国剩余定理：

```cpp
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
using i128 = __int128_t;

/*
 * exgcd: 扩展欧几里得
 * 返回 gcd(a,b)，并求出 x, y 满足 a*x + b*y = gcd(a,b)
 */
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    i64 x1, y1;
    i64 g = exgcd(b, a % b, x1, y1);
    x = y1;
```

```
    y = x1 - (a / b) * y1;
    return g;
}

/*
 * 拓展中国剩余定理（Extended CRT）
 * 输入：
 *   a[i] ≡ x (mod m[i])
 * 允许 m[i] 不互质
 *
 * 返回：
 *   pair {x, lcm}，表示最终解为：
 *     x ≡ ans (mod lcm)
 *   若无解，返回 {0, -1}
 *
 * 复杂度：O(k log M)，k=方程数
 */
pair<i64, i64> exCRT(const vector<i64> &a, const vector<i64> &m) {
    i64 x = a[0];
    i64 mod = m[0];
    for (size_t i = 1; i < a.size(); ++i) {
        i64 a2 = a[i], m2 = m[i];
        i64 k1, k2;
        i64 g = exgcd(mod, m2, k1, k2);
        if ((a2 - x) % g != 0)
            return {0, -1};  // 无解

        // 注意防溢出：使用 __int128
        i128 t = (i128)(a2 - x) / g * k1;
        t %= (m2 / g);
        if (t < 0) t += m2 / g;

        x = (i128)x + (i128)mod * t;
        mod = mod / g * m2;
        x = (x % mod + mod) % mod;
    }
    return {x, mod};
}
int main() {
    vector<i64> a = {2, 3, 2};
    vector<i64> m = {3, 5, 7};
    auto res = exCRT(a, m);
    if (res.second == -1)
        cout << "No solution\n";
    else
        cout << "x = " << res.first << " (mod " << res.second << ")\n";
}
```

lucas:

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

```cpp

ll mod_pow(ll a, ll b, ll p) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}

// ---------- 预处理阶乘和逆元 ----------
const int MAXP = 1000005;  // 根据需要设置 <= p
ll fac[MAXP], invfac[MAXP];
ll p;

// 计算 n! mod p
void init_fac(ll prime) {
    p = prime;
    fac[0] = 1;
    for (ll i = 1; i < p; ++i)
        fac[i] = fac[i - 1] * i % p;
    invfac[p - 1] = mod_pow(fac[p - 1], p - 2, p);
    for (ll i = p - 2; i >= 0; --i)
        invfac[i] = invfac[i + 1] * (i + 1) % p;
}

// 计算组合数 C(n, m) mod p （要求 n,m < p）
ll C_mod_p(ll n, ll m) {
    if (m > n) return 0;
    return fac[n] * invfac[m] % p * invfac[n - m] % p;
}

// Lucas 定理递归部分
ll Lucas(ll n, ll m) {
    if (m == 0) return 1;
    return C_mod_p(n % p, m % p) * Lucas(n / p, m / p) % p;
}

int main() {
    ll n, m;
    cin >> n >> m >> p;  // 输入 n, m, p（p 为质数）
    init_fac(p);
    cout << Lucas(n, m) % p << "\n";
    return 0;
}

```

欧拉筛：

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000005;
```

```cpp

int phi[N];         // 欧拉函数
bool vis[N];        // 是否被标记
vector<int> prime;  // 素数表

void euler_sieve(int n) {
    phi[1] = 1;
    for (int i = 2; i <= n; ++i) {
        if (!vis[i]) {
            prime.push_back(i);
            phi[i] = i - 1;   // 素数 phi(p) = p-1
        }
        for (int p : prime) {
            if (i * p > n) break;
            vis[i * p] = true;
            if (i % p == 0) {
                phi[i * p] = phi[i] * p; // p | i
                break;
            } else {
                phi[i * p] = phi[i] * (p - 1);
            }
        }
    }
}

```

欧拉函数求逆元：

```cpp
// 快速幂
long long mod_pow(long long a, long long b, long long mod) {
    long long res = 1;
    a %= mod;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

// a 的模逆元 mod m
long long mod_inv(long long a, long long mod) {
    // 如果 mod 是质数
    return mod_pow(a, mod - 2, mod);
}

```

快速求组合数：(**大整数组合数取模** 或 **组合数学计数类问题**)

阶乘预处理 + 模逆（质数模）

当 p 是质数，并且 n 较小（比如 $n \le 10^6$）：

1️⃣ 公式

$$C(n,k) \equiv \frac{n!}{k!(n-k)!} \pmod{p}$$

使用 **模逆元** 代替除法：

$$C(n,k) \equiv n! \cdot (k!)^{-1} \cdot ((n-k)!)^{-1} \pmod{p}$$

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

const int N = 1000005;
ll fac[N], invfac[N];
ll p;

ll mod_pow(ll a, ll b, ll mod) {
    ll res = 1;
    a %= mod;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

void init_fac(ll prime) {
    p = prime;
    fac[0] = 1;
    for (ll i = 1; i < p; ++i) fac[i] = fac[i-1] * i % p;
    invfac[p-1] = mod_pow(fac[p-1], p-2, p); // mod p 逆元
    for (ll i = p-2; i >= 0; --i) invfac[i] = invfac[i+1] * (i+1) % p;
}

ll C_mod_p(ll n, ll k) {
    if (k > n) return 0;
    return fac[n] * invfac[k] % p * invfac[n-k] % p;
}

int main() {
    ll n, k, prime;
    cin >> n >> k >> prime;
    init_fac(prime);
    cout << C_mod_p(n, k) << "\n";
}

```

exgcd:

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

// 扩展欧几里得
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return d;
}

// a 的模逆元 mod m
ll mod_inv(ll a, ll m) {
    ll x, y;
    ll g = exgcd(a, m, x, y);
    if (g != 1) {
        // 无逆元
        return -1;
    }
    x %= m;
    if (x < 0) x += m;
    return x;
}

int main() {
    ll a = 3, m = 7;
    ll inv = mod_inv(a, m);
    if (inv == -1) cout << "No inverse\n";
    else cout << "Inverse of " << a << " mod " << m << " = " << inv << "\n";
}

```

快速幂:

```cpp
#include <bits/stdc++.h>
using namespace std;
using u64 = unsigned long long;
using u128 = __uint128_t;

// ⚡ 快速乘，防止溢出
u64 mul(u64 a, u64 b, u64 mod) {
    return (u128)a * b % mod;
}

// 快速幂（大数取模）
u64 mod_pow(u64 a, u64 b, u64 mod) {
```

```cpp
    u64 res = 1;
    a %= mod;
    while (b) {
        if (b & 1) res = mul(res, a, mod);
        a = mul(a, a, mod);
        b >>= 1;
    }
    return res;
}

// 示例：阶乘分块求 n! % mod (大数)
u64 factorial_mod(u64 n, u64 mod) {
    u64 res = 1;
    for (u64 i = 1; i <= n; ++i) {
        res = mul(res, i, mod);
    }
    return res;
}

int main() {
    u64 a = 123456789012345, b = 98765432109876, mod = 1000000007;

    cout << "mod_pow(a,b,mod) = " << mod_pow(a, b, mod) << endl;

    u64 n = 100000;
    cout << n << "! mod " << mod << " = " << factorial_mod(n, mod) << endl;
}

```