Combinatorics + Lucas + Chinese Remainder Theorem

Combinatorics & Counting Template

```
const int N = 1e6 + 9, oo = 1e17, MOD = 1e9+7;
// f[i] = i!
// inv[i] = modular inverse of i
// finv[i] = modular inverse of i!
int f[N], inv[N], finv[N];
// Precompute factorials, inverse of numbers, and inverse factorials
void prec() {
    f[0] = 1;
    for (int i = 1; i < N; i++)
        f[i] = 1LL * i * f[i - 1] % MOD;
    inv[1] = 1;
    for (int i = 2; i < N; i++) {
        // Fermat's trick to compute modular inverse of i
        inv[i] = (-(1LL * MOD / i) * inv[MOD % i]) % MOD;
        inv[i] = (inv[i] + MOD) % MOD;  // make sure it's positive
    }
    finv[0] = 1;
    for (int i = 1; i < N; i++)
        finv[i] = 1LL * inv[i] * finv[i - 1] % MOD;
}
// nCr = number of ways to choose r items from n (order doesn't matter)
int ncr(int n, int r) {
    if (n < r || n < 0 || r < 0) return 0;
    return 1LL * f[n] * finv[n - r] % MOD * finv[r] % MOD;
}

// nPr = number of ways to arrange r items from n (order matters)
int npr(int n, int r) {
    if (n < r || n < 0 || r < 0)
        return 0;
    return 1LL * f[n] * finv[n - r] % MOD;
}
// Fast exponentiation: computes base^pow % MOD in O(log pow)
int fastPower(int base, int pow){
    base %= MOD;
    int res = 1;
    while (pow > 0) {
        if (pow & 1)
            res = res * base % MOD;
        base = base * base % MOD;
        pow >>= 1;
    }
    return res;
}
```

Extra Notes

1. When to use nPr (permutations):
   If order matters
   Keywords: "arrange", "sequence", "line up", "in order"
2. When to use nCr (combinations):
   If order doesn't matter
   Keywords: "choose", "select", "group of"

3. Stars and Bars:
   Used when distributing items into groups
   Example: distribute 10 candies among 3 kids
   Formula: Number of integer solutions for: x1 + x2 + ... + xk = n
   → Answer = C(n + k - 1, k - 1)
4. Pigeonhole Principle:
   If you have more items than containers,
   → at least one container has more than one item.
   Example: 13 people in 12 months → 2 people share a birthday month.

Lucas's Theorem

كود Lucas's Theorem (لما P ≤ N) - P هنا هو الـ mod

```
const int MOD = 1e6 + 3; // لازم يكون P أولي وصغير (<= 1e6 مثلاً)
int fact[MOD];

// Precompute factorials mod P
void init_factorials() {
    fact[0] = 1;
    for (int i = 1; i < MOD; i++)
        fact[i] = 1LL * fact[i - 1] * i % MOD;
}
// Fast power mod P
int power(int base, int exp) {
    int res = 1;
    while (exp > 0) {
        if (exp & 1) res = 1LL * res * base % MOD;
        base = 1LL * base * base % MOD;
        exp >>= 1;
    }
    return res;
}
// Modular inverse using Fermat's Little Theorem
int mod_inverse(int a) {
    return power(a, MOD - 2);
}
// C(n, k) % P when n, k < P
int nCr_mod_p_small(int n, int k) {
    if (k > n) return 0;
    return 1LL * fact[n] * mod_inverse(fact[k]) % MOD * mod_inverse(fact[n - k]) % MOD;
}
// Lucas's Theorem: C(n, k) % P for any n, k
int lucas(int n, int k) {
    if (k == 0) return 1;
    return 1LL * lucas(n / MOD, k / MOD) * nCr_mod_p_small(n % MOD, k % MOD) % MOD;
}
```

💡 Notes

- `lucas(n, k)` بترجع `C(n, k) % P` مهما كان حجم `n` و `k`
- لازم `P` يكون prime
- بتقسم `n` و `k` لـ أرقام صغيرة أقل من `P`
- بتشتغل كويس حتى لما `n > 1e18` لو `P` صغير

Extended Lucas + CRT

```cpp
typedef long long ll;
ll mulmod(ll a, ll b, ll mod) {
    return a * b % mod;
}
ll power(ll a, ll b, ll mod) {
    ll res = 1;
    a %= mod;
    while (b > 0) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
ll modInverse(ll a, ll mod) {
    return power(a, mod - 2, mod); // mod must be prime
}
// compute nCr % p for small n < p
ll C_mod_p(ll n, ll r, ll p, vector<ll>& fact) {
    if (r > n) return 0;
    return fact[n] * modInverse(fact[r], p) % p * modInverse(fact[n - r], p) % p;
}
// Lucas Theorem for a single prime p
ll Lucas(ll n, ll r, ll p) {
    vector<ll> fact(p, 1);
    for (ll i = 1; i < p; ++i)
        fact[i] = fact[i - 1] * i % p;

    ll result = 1;
    while (n || r) {
        ll ni = n % p, ri = r % p;
        result = result * C_mod_p(ni, ri, p, fact) % p;
        n /= p, r /= p;
    }
    return result;
}
// Chinese Remainder Theorem: combine values mod p1, p2, ...
ll CRT(const vector<ll>& rem, const vector<ll>& mod) {
    ll prod = 1, res = 0;
    int k = rem.size();
    for (int i = 0; i < k; ++i)
        prod *= mod[i];
    for (int i = 0; i < k; ++i) {
        ll pp = prod / mod[i];
        res = (res + rem[i] * modInverse(pp, mod[i]) % prod * pp % prod) % prod;
    }
    return res;
}
// Main function: Extended Lucas for mod M = p1 * p2 * ...
ll extendedLucas(ll n, ll r, const vector<ll>& primes) {
    vector<ll> rem;
    for (ll p : primes)
        rem.push_back(Lucas(n, r, p));  // result mod each pi
    return CRT(rem, primes);            // combine using CRT
}
```

```cpp
int main() {
    ll n = 1e18, r = 1e6;
    vector<ll> primes = {3, 11, 61};  // M = 3 * 11 * 61 = 2013
    cout << extendedLucas(n, r, primes) << '\n';
}
```

بجيب بنفسي البرايمز الـ ف المود عشان أحطه مع الـ فانكشن