「学习总结」数论

Jiayi Su (ShuYuMo)

2020-12-20 23:41:12

关于 Lucas 定理及其扩展，BSGS 及其扩展，快速质因数分解 Pollard-Rho，素性测试以及原根和阶的一些可爱知识。

数论

Lucas 定理

解决模数为小质数的组合数取模问题。

设模数为 P，设

$$n = \sum_{i \geq 0} a_i P^i, m = \sum_{i \geq 0} b_i P^i$$

$$\binom{n}{m} \equiv \prod_{i \geq 0} \binom{a_i}{b_i} (mod\ P)$$

{#eq:Lucas}

同时 (eq.~@eq:Lucas) 还有一种形式化的写法：

$$\binom{n}{m} \equiv \binom{\lfloor n/P \rfloor}{\lfloor m/P \rfloor} \binom{n \mod P}{m \mod P} (mod\ P)$$

{#eq:Luacs_ex}

若小组合数可以 $\mathcal{O}(1)$ 查询，Luacs 的时间复杂度为 $\mathcal{O}(\log_p(n))$ 。

取模后不能保证 $n \geq m$ ，查询组合数要写的细致一点。

特殊地，如果模数是 2 ，根据 (eq.~@eq:Lucas) 考虑如果把数字进行二进制拆分求组合数，不难发现其在模 2 意义下的值为 $[(n \quad \& \quad m) = m]$ 其中 n, m 的意义同 (eq.~@eq:Lucas)。

```
int _C_(int m, int n){
    if(m < n)  return 0;
    return frac[m] *1ll* ifrac[n] % MOD *1ll* ifrac[m - n] % MOD;
}
int C(int m, int n){
    if(MOD == 2) return ((n & m) == n);
    if(m < n)  return 0; if(n == 0) return 1;
    return C(m / MOD, n / MOD) *1ll* _C_(m % MOD, n % MOD) % MOD;
}
```

扩展 Lucas 定理

解决模数为可能不为质数，且每个质数幂较小的组合数取模问题。扩展卢卡斯定理和卢卡斯定理没有关系 并没有用到 Luacs 的基本思想 (eq.~@eq:Lucas) ，扩展 Lucas 定理 本质上是解决了阶乘逆元不存在的情况下，形如

$$\frac{*}{n!} \mod P$$

{#eq:exl}

式子的求值。考虑组合数通项公式：

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

{#eq:exC}

限制我们不能直接求 (eq.~@eq:exC) 的原因——模数不为质数，无法保证逆元存在。考虑如果让分母和模数互质就可以求出这个式子的值了。

首先可以对模数 p 进行质因数分解，分解成形如 $p = \prod_{i \geq 0} P_i^{e_i}$ ，先分别求模数为 $P_i^{e_i}$ 式子的值，然后 CRT 合并答案。

考虑如何求出模数为 $P_i^{e_i}$ 的值。设函数 $g_p(n)$ 为 $n!$ 中质因子 p 的幂次。设 $a = g_p(n), b = g_p(m), c = g_p(n-m)$ 易知：

$$(eq.\ @eq:exC) \equiv \frac{\frac{n!}{p^a}}{\frac{m!}{p^b}\frac{(n-m)!}{p^c}}\ p^{a-b-c}\ (mod\ P_i^{e_i})$$

{#eq:exLucas}

设 $k = P_i^{e_i}$ ，如果所求为 $\frac{n!}{p_i^a}$ 对于 $n!$ 的每一项 $(1 \times 2 \times 3 \times 4 \times 5 \times \cdots \times n)$，其每一项在模 k 意义下的取值一定是每 k 个一次循环的。

同时每一个 P_i 的倍数都可以提出一个因子 P_i 也可以轻松知道，这个提出来的因子有 $\lfloor \frac{n}{P_i} \rfloor$ 个，即 $P^{\lfloor \frac{n}{P_i} \rfloor}$ 这个式子不计入答案。

这些倍数提出一个 P_i 的因子之后一定会形成另一个阶乘的形式，这是一个子问题，可以递归计算。剩下的数字可以暴力计算一个周期，然后根据周期的出现次数，直接计算式子的值。这样就可以算出 $n!$ 去掉所有质因子 P_i 在模 $P_i^{e_i}$ 意义下的值了。保证了这个值和 P_i 互质，这样就可以求逆元了。根据 (eq.~@eq:exLucas) 计算即可。

关于函数 $g_p(n)$ 的计算：

$$g_p(n) = \prod_{i \geq 1} \lfloor \frac{n}{p^i} \rfloor = \frac{n - f_p(n)}{p-1}$$

其中 $f_p(n)$ 为数字 n 在 p 进制下的数位和。

注意逆元不是质数，别~~傻不拉几的~~冲一个费马小定理。

复杂度是 $\mathcal{O}\left(\sum_{i \geq 0} (\log P + P_i^{e_i}) \log n\right)$ 大概就是 $\mathcal{O}\left(\sum_{i \geq 0} P_i^{e_i} \log n\right)$ 吧。

```
int f(int n, int p){
    int ans = 0;
    while(n) { ans += n % p; n /= p; }
    return ans;
}
int g(int n, int P){ return (n - f(n, P)) / (P - 1); }

int pow(int a, int b, int P){ int ans = 1; while(b) { if(b & 1) ans = ans *1ll* a % P; a = a *1ll* a %
//int inv(int x, int MOD) { return pow(x, MOD - 2, MOD); } // deleted: Fermat's Little Theorem is NOT c
```

```cpp
int exgcd(int a, int b, int &x, int &y){
    if(!b) { x = 1; y = 0; return a; }
    int g = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return g;
}
int inv(int a, int P){ // add: Exgcd
    int x, y; exgcd(a, P, x, y);
    return (x % P +0ll+ P ) % P;
}

int calc(int n, int p, int k){ // calc n!(without p) mod p^k
    if(n == 0) return 1;
    int P = 1; rep(i, 1, k) P *= p;
    int res = n % P, prd0 = 1, prd1 = 1;
    rep(i, 1, P){
        if(i % p == 0) continue;
        if(i <= res) prd0 = prd0 *1ll* i % P;
        prd1 = prd1 *1ll* i % P;
    }
    int ans = calc(n / p, p, k);
    ans = ans *1ll* prd0 % P;
    ans = ans *1ll* pow(prd1, n / P, P) % P; // changed `pow(prd1, n / p, P)` to `pow(prd1, n / P, P) %
    return ans;
}
LL n, m; int MOD;
vector<pair<int, int > > d;
namespace Divide{
    const int _ = 1e6 + 100;
    int np[_], prime[_], tot = 0, Mid[_];
    void init(int n){
        rep(i, 2, n){
            if(!np[i]) prime[++tot] = i, Mid[i] = i;
            for(int j = 1; j <= tot && prime[j] * i <= n; j++){
                int x = prime[j] * i; Mid[x] = prime[j];
                np[x] = 1;
                if(i % prime[j] == 0) break;
            }
        }
    }
    void divide(vector<pair<int, int > > & res, int MOD){
        for(int i = 1; i <= tot; i++){
            if(MOD % prime[i] != 0) continue;
            pair<int, int > ans; ans.fi = prime[i];
            ans.se = 0;
            while(MOD % prime[i] == 0) MOD /= prime[i], ans.se++;
            res.push_back(ans);
```

```
        }
    }// 这里可以直接 $\mathcal{O}(\sqrt{n})$ 试除即可，没必要分解质因数。写的时候太年轻了。
}
int CRT(vector<pair<int, int > > & A){
    int W = MOD;
    int ans = 0;
    rep(i, 0, A.size() - 1){
        pair<int, int > now = A[i];
        ans = (ans +0ll+ ( now.fi *1ll* (W / now.se) % MOD *1ll* inv(W / now.se, now.se) ) % MOD) % MOD
    }
    return ans;
}

vector<pair<int, int > > Ans;
signed main(){ //freopen(".in", "r", stdin);
    Read(n)(m)(MOD); Divide::init(MOD + 2);
    Divide::divide(d, MOD);
    rep(i, 0, d.size() - 1){
        pair<int, int> NowMod = d[i]; int P = pow(NowMod.fi, NowMod.se);
        int rA = calc(n, NowMod.fi, NowMod.se);
        int rB = calc(n - m, NowMod.fi, NowMod.se);
        int rC = calc(m, NowMod.fi, NowMod.se);
        int ans = 0;
        ans = rA *1ll* inv(rB, P) % P *1ll* inv(rC, P) % P;
        ans = ans *1ll* pow(NowMod.fi, g(n, NowMod.fi) - g(n - m, NowMod.fi) - g(m, NowMod.fi), P) % P;
        Ans.push_back(mp(ans, P));
    }
    printf("%lld\n", CRT(Ans));
    return 0;
}
```

BSGS

求解模数为质数的指数方程。$a^x \equiv b \ (mod\ p)\ p \in P$。由欧拉定理可知，本质不同的解 范围为 $[0, p-1]$ 。朴素做法就是枚举一下 $[0, p-1]$ 判断是不是解，但是这样显然太慢了。考虑选一个参数 T，则 $\forall x \in [0, p-1]$ 都可以表示成 $r \times T + c \ \ (b < T)$ 的形式，易知，$r = \lfloor \frac{x}{T} \rfloor, c = x \ mod \ T$

带入原式：

$$a^{rT+c} \equiv b \ (mod\ p)$$

$$a^c \equiv b \times a^{-rT} \ (mod\ p)$$

{#eq:BSGS0}

可以考虑枚举 c 算出每一个 a^c 压入 set 或 hash 表，然后枚举每一个 r 算出每一个 $a^{-rT} \times b$ 在之前算出的答案中查找，如果能找到相同的取值（即：满足 (eq.~@eq:BSGS0)），就说明当前这个 r 就是一个答案。

```
int inv(int a, int P){
    int x, y; exgcd(a, P, x, y);
    return (x % P +0ll+ P) % P;
}
int BSGS(int a, int b, int P) { // a^x = b (mod P) (a, p) = 1;
    static set<int> S; S.clear();
    int T = sqrt(P);
    for(int i = 0, x = 1; i < T; i++, x = x *1ll* a % P) if(x != b) S.insert(x); else return i;
    for(int i = 1; i <= T; i++){
        int now = b *1ll* inv(pow(a, T * i, P), P) % P;
        if(!S.count(now)) continue;
        for(int j = i * T, V = pow(a, j, P); ; j++, V = V *1ll* a % P) if(V == b) return j;
    }
    return -1;
}
```

扩展 BSGS

用于求解模数不一定为质数的指数方程。$a^x \equiv b \ (mod\ p)\ p \in P$。限制不能直接 $BSGS$ 的因素就是不能保证逆元存在，那就考虑如何让 $(a, p) = 1$ 。

具体地，设 $d_1 = \gcd(a, p)$ 。如果 $d_1 \nmid b$ ，则原方程无解。否则我们把方程同时除以 d_1 ，得到

$$\frac{a}{d_1} \cdot a^{x-1} \equiv \frac{b}{d_1} \pmod{\frac{p}{d_1}}$$

如果 a 和 $\frac{p}{d_1}$ 仍不互质就再除，设 $d_2 = \gcd\left(a, \frac{p}{d_1}\right)$ 。如果 $d_2 \nmid \frac{b}{d_1}$ ，则方程无解；否则同时除以 d_2 得到.

$$\frac{a^2}{d_1 d_2} \cdot a^{x-2} \equiv \frac{b}{d_1 d_2} \pmod{\frac{p}{d_1 d_2}}$$

直到模数和 a 互质。记 $D = \prod d_i$，则

$$\frac{a^k}{D} \cdot a^{x-k} \equiv \frac{b}{D} \pmod{\frac{p}{D}}$$

就可以使用 BSGS 求解了。

需要注意的是：

- 解可能小于 k ，可以暴力枚举一下小于 k 的幂次，判断一下
- 如果有 d_i 不是 b 的约数，直接判断无解即可。

```
int exBSGS(int a, int b, int P){
    int D = 1, k = 0, tp = P;
    while(true) { int g = gcd(a, tp); if(g == 1) break; tp /= g; D *= g; k++;  }
    for(int i = 0, V = 1; i <= k; i++, V = V *1ll* a % P) if(V == b) return i; // changed: It must be e
    int S = pow(a, k, tp);
    if(b % D != 0) return -1; // added: It is necessary!
    int B = b *1ll* inv(S, tp) % tp;
    int r = BSGS(a, B, tp);
    return r == -1 ? -1 : r + k;
}
```

已通过 luogu 和 SPOJ 的测试数据，但是 zhx (Orz) 的数据只有 50pts 待填。

Miller_Rabin

快速判断大数素性。

一个结论：如果 n 为质数 $\forall a < n$ 设 $n-1 = d \times 2^r, (d, 2) = 1$，则

$$a^d \equiv 1 \pmod n$$

{#eq:MR0}

$$\exists\, 0 \le i < r, s.t. a^{d\times i^i} \equiv -1 \pmod n$$

{#eq:MR1}

(eq.~@eq:MR0) 和 (eq.~@eq:MR1) 至少一个满足。是否为充要条件带查。

我们需要判断一个数是否为质数，只需要判断是否符合上面的定理即可，但是 $\forall a < n$ 对复杂度不友好。

通常的做法是选择若干质数当作底数 a，进行判断。

关于底数的选择：> 如果选用 2, 3, 7, 61 和 24251 作为底数，那么 10^16 内唯一的强伪素数为 46 856 248 255 981 ——matrix67 博客 (Orz)

选择 `int PrimeList[10] = {2, 3, 7, 61, 24251, 11, 29};` 即可，或者再随意加几个小质数。

需要注意快速乘法的实现。复杂度为 $\mathcal{O}(k \log x)$

```
#define LL long long
#define ULL unsigned long long
inline ULL mul(ULL a, ULL b, ULL MOD){ // unsigned long long
    LL R = (LL)a*b - (LL)((ULL)((long double)a * b / MOD) * MOD);
    // 只关心两个答案的差值，这个差值一定小于 unsigned long long 的最大值，
    // 所以在哪个剩余系下都不重要，不管差值是什么都能还原出原始数值。
    if(R < 0) R += MOD;
    if(R > MOD) R -= MOD;
    return R;
}
inline LL pow(LL a, LL b, LL P) { LL ans = 1; while(b){ if(b & 1) ans = mul(ans, a, P); a = mul(a, a, P
int PrimeList[10] = {2, 3, 7, 61, 24251, 11, 17, 19, 29, 27};
bool Miller_Rabin(int a, LL n){
    LL d = n - 1, r = 0, x;
    while(!(d & 1)) d >>= 1, r++;
    if((x = pow(a, d, n)) == 1) return true;
    for(int t = 1; t <= r; t++, x = mul(x, x, n)) if(x == n - 1) return true;
    return false;
}
bool Prime(LL x){
    if(x <= 2) return x == 2;
    for(int i = 0; i < 7; i++){
        if(x == PrimeList[i]) return true;
        if(!Miller_Rabin(PrimeList[i], x)) return false;
```

```
    }
    return true;
}
```

Pollard-Rho

快速分解质因数的解决方案。复杂度 $\mathcal{O}(Accepted)$。大约为 $\mathcal{O}(n^{1/4})$ ，算法导论给出的是 $\mathcal{O}(\sqrt{n})$，但是随机数据下，$1s$ 分解 10000 个 10^{18} 量级的数字问题不大……

> 任何一个伟大的算法都来自于一个微不足道的猜想。

考虑给出一个数字 n ，如果可以快速找其中一个因子 g ，然后继续递归分解 n/g 和 g 即可完成质因数分解。

考虑如何快速寻找一个数 n 的因子 g；（以下复杂度分析都是基于最坏情况，即素数平方的形式）

1. 枚举 n 的因子 g 试除，复杂度为 $\mathcal{O}(\sqrt{n})$
2. 考虑随机枚举 n 的因子，试除，期望复杂度为 $\mathcal{O}(n)$
3. 没必要随机枚举 n 的因子，只需要随机一个数字 a 那么 $g = gcd(a, n)$ 就是 n 的一个因子，需要找到一个不平凡的因子才可以，这样 a 的合法取值变成了 n 的因子或 n 因子的倍数，合法的 a 的取值有 $\mathcal{O}(\sqrt{n})$ 个。

着重考虑下一种优化：

考虑一次性随机出 k 个数字 分别记作 $a_{1\dots k}$，考虑其中两两的差值，应该有 k^2 种差值。试图让其差值与 n 求最大公约数 c，若 c 不为 1 则 c 就是一个不平凡因子。差值与 n 的最大公约数为 c，等价于 差值在 c 的剩余系下同余 0。考虑这 k^2 个差值都不等于 0 的概率为多少。因为这 k 个数字是随机的，那么他们差值的取值在 c 的剩余系下也是在 $[0, c]$ 等概率分布的。那么都不等于 0 的概率为 $\left(\frac{c-1}{c}\right)^{k^2}$。最坏情况下，合法的 c 只有一个，且等于 $\sqrt{n}$ 根据

$$\left(\frac{n-1}{n}\right)^n \approx \frac{1}{e}$$

当 $k = n^{1/4}$ 时，取不到值得概率为 $\frac{1}{e}$。稍微增大几倍的 k 可以降低找不到概率。注意这样的做法需要枚举所有 k^2 对差值，再加上判断和取不到约数情况的出现，复杂度要劣于 $\mathcal{O}(\sqrt{n})$。枚举 k^2 对差值是上面算法的瓶颈。引入伪随机函数 $f(x) = f^2(x-1) + c \mod n$。这个函数有几点比较优良性质：

1. 仍然可以看成是一种随机函数，保证了上面推导中对随机的依赖性。（个人感觉不能保证完全随机……）
2. 经过一段次数的递推之后会进入一个循环，因为之和前一项的值有关，而且取值只能在 $[0, n-1]$ 内，所以至多 $\mathcal{O}(n)$ 次递推，一定会进入一个循环。
3. $$g \mid f(i) - f(j) \quad \Rightarrow \quad g \mid f(i+1) - f(j-1)$$

   证明：

   $$f(i+1) - f(j+1) = (f^2(i) + c) - (f^2(j) + c) = (f(i) - f(j))(f(i) + f(j))$$

   也就是 g 是否为某两个差值的约数之和两个差值下标的差有关。

考虑两个指针，分别为 L, R ，每次 L 递推一个，R 递推两个，因为函数存在环，这两个差值总会在环上相遇，从开始到相遇之间的时间，每次执行后，判断其差值与 n 的 gcd 是否不等于 1 即可。相遇所需时间，即环的大小约为 $\sqrt{n}$ 。环的大小决定运行最坏时间。

这样还是不够快，其实不需要每次执行都算一下 gcd 可以执行一些步数之后，将其差值乘到一起，一起求 gcd 即可，可以证明，这样的变化保证答案不会变劣。考虑让两个指针倍增的往前跳，倍增若干次后，每次计算两个指针距离之间的所有差值取值，乘在一起计算 gcd。

标准代码中：差值相同的解会计算多次。但是确实大大提升运行效率。这里就不会分析了，但是直观感觉就是，其实那个函数往后递推的次数越多会包含更多的因子。也就是说，如果两对函数值差值下标距离一样，其实下标大的那一对会更

优。

总的来说，快的玄学。

可以写出如下代码：

```
LL Pollard_Rho(LL n){
    LL c = rand(n - 1) + 1;
    LL L = 0;
    for(int s = 0, t = 1; ; s <<= 1, t <<= 1){
        LL R = L, M = 1;
        for(int i = s, step = 1; i < t; i++, step ++){
            R = f(R, c, n);
            M = mul(M, abs(L - R), n);
            if(step % 1000 == 0) { LL g = gcd(M, n); if(g > 1) return g; }
            // 不一定必须等到一轮计算结束之后再计算 gcd 可能中间就已经出现答案了，中间每隔一段时间算几次，可以在出现
        }
        LL g = gcd(M, n); if(g > 1) return g;
        L = R;
    }
}

void factorize(LL n){
    if(Prime(n)) { ans = max(ans, n); return ; }
    LL g = n;
    while(g == n) g = Pollard_Rho(n);
    factorize(g); factorize(n / g);
}
```

阶

若 $a \perp P$，则使 $a^k \equiv 1 \pmod P$ 成立的最小的 k，称为 a 关于模 P 的阶，记作 $\operatorname{ord}_m(a)$。

求法：根据欧拉定理，$\operatorname{ord}_m(a) \mid \varphi(m)$，先求出 $\varphi(m)$，记作 t。

- 枚举 t 的每一个因子，检查是否满足阶的性质。
- 对 t 质因数拆分，以此考虑每个质因子，试图减少质因子的幂次——即如果减少了某个质因子的幂次，$a^t \equiv 1 \pmod P$ 依然成立，那么就直接减少即可。感性理解一下肯定是对的。$\mathcal{O}(\log \varphi(P))$

原根

若 $\operatorname{ord}_m g = \varphi(m)$ 则称 g 为 m 的原根。

通俗的讲：称 g 为 P 的原根，当且仅当 $\{g^0, g^1, g^2, \dots, g^{\varphi(P)-1}\}$ 数两两不同。

即对于任意一个和 P 互质的数字，在 P 的剩余系下，都可以表示成 g^t 的形式。

一个模数存在原根，当且仅当这个模数为 $2, 4, p^s, 2p^s$，p 为奇素数。

判断一个数是否为原根：根据定义可以考虑枚举每一个 $t \in [1, \varphi(P) - 1]$ 判断 g^t 是否都不等于 1。事实上，根据欧拉定理，可能使 $g^t \equiv 1 \pmod P$ 的 t 只可能是是 $\varphi(P)$ 的约数。枚举 $\varphi(P)$ 的每个约数，进行判断即可，复杂度为 $\mathcal{O}(\sqrt{\varphi(P)})$

原根的求法：若一个数 m 存在原根，其最小原根大概在 $m^{1/4}$ 级别。枚举原根再判断一般是可以接受的。

```cpp
bool JudgePrimitiveRoot(int g, int P){
    int phi = P - 1;
    for(int i = 2; i * i <= phi; i++){
        if(phi % i != 0) continue;
        if(pow(g, i, P) == 1) return false;
        if(pow(g, P / i, P) == 1) return false;
    }
    return true;
}
int GetPrimitiveRoot(int P){ for(int i = 2; ; i++) if(JudgePrimitiveRoot(i, P)) return i; }
```

地位相当于自然数域下的唯一分解定理。