XCPC Templates

Khoray

March 24, 2022

Contents

1 数学 **3**
1.1 欧拉筛/积性函数筛/线性筛 . . . 3
1.2 快速幂 . . . 3
1.3 组合数 . . . 3
1.3.1 暴力 . . . 3
1.3.2 递推 . . . 4
1.3.3 逆元 . . . 4
1.4 ExGCD . . . 5
1.5 拉格朗日插值 . . . 5
1.6 原根 . . . 6
1.7 多项式全家桶 . . . 7

1 数学

1.1 欧拉筛/积性函数筛/线性筛

- 积性函数筛，$f(pq) = f(p)f(q), (p, q) = 1$
- calc_f(val, power) 返回 $f(val^{power})$

```
// all in which f[pq] = f[p] * f[q] (gcd(p, q) = 1)
const int N = 1e7 + 5;
int pri[N / 5], notpri[N], prinum, minpri_cnt[N], f[N];

int calc_f(int val, int power) {

}

void init_pri() {
   for(int i = 2; i < N; i++) {
      if(!notpri[i]) pri[++prinum] = i, minpri_cnt[i] = 1, f[i] = calc_f(i, 1);
      for(int j = 1; j <= prinum && pri[j] * i < N; j++) {
         notpri[pri[j] * i] = pri[j];
         if(i % pri[j] == 0) {
            minpri_cnt[pri[j] * i] = minpri_cnt[i] + 1;
            f[pri[j] * i] = f[i] / calc_f(pri[j], minpri_cnt[i]) * calc_f(pri[j],
                minpri_cnt[i] + 1);
            break;
         }
         minpri_cnt[pri[j] * i] = 1;
         f[pri[j] * i] = f[i] * calc_f(pri[j], 1);
      }
   }
}
```

1.2 快速幂

```
int ksm(int a, int b = mod - 2, int MOD_KSM = mod) {
   int ret = 1;
   while(b) {
      if(b & 1) {
         ret = ret * a % MOD_KSM;
      }
      a = a * a % MOD_KSM;
      b >>= 1;
   }
   return ret;
}
```

1.3 组合数

1.3.1 暴力

- 暴力求组合数 $\binom{n}{k}$, 时间复杂度 $O(\min(k, n-k))$.

- 前置：快速幂
- 模数必须是质数!

```
int binom(int n, int k) {
    if(n < 0 || k < 0 || k > n) { return 0; }
    k = min(n - k, k);
    int u = 1, v = 1;
    for(int i = 0; i < k; i++) {
        v = v * (i + 1) % mod;
        u = u * (n - i) % mod;
    }
    return u * ksm(v, mod - 2) % mod;
}
```

1.3.2 递推

- $O(n^2)$ 递推求，模数随意。

```
const int N = 31;
int C[N][N];

void init_C() {
    C[0][0] = C[1][0] = C[1][1] = 1;
    for(int i = 2; i < N; i++) {
        C[i][0] = 1;
        for(int j = 1; j <= i; j++)
            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
    }
}

int binom(int n, int k) {
    if(n < 0 || k < 0 || n < k) return 0;
    return C[n][k];
}
```

1.3.3 逆元

- 模数必须是质数!
- 前置：快速幂

```
const int N = 1e7 + 5;
const int mod = 1e9 + 7;
int facinv[N], fac[N];
void init_fac() {
    fac[0] = fac[1] = 1;
    for(int i = 2; i < N; i++) {
        fac[i] = fac[i - 1] * i % mod;
    }
    facinv[N - 1] = ksm(fac[N - 1], mod - 2);
    for(int i = N - 2; i >= 0; i--) {
```

```
        facinv[i] = facinv[i + 1] * (i + 1) % mod;
    }
}
int binom(int n, int k) {
    if(n < 0 || k < 0 || k > n) { return 0; }
    return fac[n] * facinv[n - k] % mod * facinv[k] % mod;
}

int inv[N];
void init_inv() {
    inv[0] = inv[1] = 1;
    for(int i = 2; i < N; i++) {
        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
    }
}
```

1.4 ExGCD

- 求解 $ax + by = (a, b)$ 的特解 x_0, y_0.
- 通解 $x^* = x_0 + \frac{bk}{(a,b)}, y^* = y_0 - \frac{ak}{(a,b)}$ ($k \in \mathbb{Z}$) .

```
// ax + by = gcd(a, b)
// return gcd(a, b)
// all sol: x = x_0 + k[a, b] / a, y = y_0 - k[a, b] / b;
int exgcd(int &x, int &y, int a, int b) {
    if(!b) {
        x = 1;
        y = 0;
        return a;
    }
    int ret = exgcd(x, y, b, a % b);
    int t = x;
    x = y;
    y = t - a / b * y;
    return ret;
}
```

1.5 拉格朗日插值

- $f(x)$ 是多项式，并且我们知道一系列连续的点值 $f(l), \cdots, f(r)$ ，求解 $f(n)$。$O(r - l)$
- 前置：逆元组合数
- 模数为质数

```
int LagrangeInterpolation(vector<int> &y, int l, int r, int n) {
    if(n <= r && n >= l) return y[n];
    vector<int> lg(r - l + 3), rg(r - l + 3);
    int ret = 0;
    lg[0] = 1;
    rg[r - l + 2] = 1;
```

```
    for(int i = l; i <= r; i++) {
        lg[i - l + 1] = (long long) lg[i - l] * (n - i) % mod;
    }
    for(int i = r; i >= l; i--) {
        rg[i - l + 1] = (long long) rg[i - l + 2] * (n - i) % mod;
    }
    for(int i = l; i <= r; i++) {
        if(r - i & 1) {
            ret = (ret - (long long) y[i] * lg[i - l] % mod * rg[i - l + 2] % mod *
                facinv[i - l] % mod * facinv[r - i] % mod + mod) % mod;
        } else {
            ret = (ret + (long long) y[i] * lg[i - l] % mod * rg[i - l + 2] % mod *
                facinv[i - l] % mod * facinv[r - i] % mod) % mod;
        }
    }
    return ret;
}
```

1.6 原根

若 g 满足：

$$(g, m) = 1$$
$$\delta_m(g) = \varphi(m)$$

则 g 为 m 的原根。找所有原根：

1. 找到最小的原根 g。如果一个数 g 是原根，那么 $\forall p|\varphi(m) : g^{\frac{\varphi(m)}{p}} \neq 1$
2. 找小于 m 与 $\varphi(m)$ 互质的数 k ，则 g^k 也是原根（能覆盖所有原根）个数为 $\varphi(\varphi(m))$ 个。

题目：找出 n 的所有原根，间隔 d 输出。

```
void solve() {
    // d 是间隔输出（对题目无影响
    int n, d; cin >> n >> d;
    vector<int> pf; // 质因数分解
    int pn = phi[n];
    while(notpri[pn]) {
        int now = notpri[pn];
        pf.push_back(now);
        while(pn % now == 0) pn /= now;
    }
    if(pn != 1) {
        pf.push_back(pn);
    }
    int cnt = 0;
    int ming = -1;
    vector<int> ans, vis(n); // 记录答案
    // 找到最小的原根 min_g
    for(int i = 1; i < n; i++) {
        if(__gcd(i, n) != 1) continue;
        int judge = 1;
        for(auto &p : pf) {
```

```
            if(ksm(i, phi[n] / p, n) == 1) {
                judge = 0;
                break;
            }
        }
        if(judge) {
            ming = i;
            break;
        }
    }
    // 还原出所有原根 g
    if(ming > 0) {
        for(int i = 1; i < n; i++) {
            if(__gcd(i, phi[n]) == 1) {
                int cur = ksm(ming, i, n);
                if(!vis[cur]) {
                    vis[cur] = 1;
                    ans.push_back(cur);
                }
            }
        }
    }
    // 排序输出所有原根
    cout << ans.size() << '\n';
    sort(ans.begin(), ans.end());
    for(auto as : ans) {
        cnt++;
        if(cnt % d == 0) {
            cout << as << ' ';
        }
    }
    cout << '\n';
}
```

1.7 多项式全家桶

- 注意调整原根 g，模数 mod，N 开 3 到 4 倍数据范围，附录 A
- 注意 resize()
- 注意 Inv/Ln 的时候常数项不能为 0
- 注意 Exp 的时候常数项必须是 0
- 注意这里面的 ksm() 第三个参数是初值而不是模数

```
#define fp(i, a, b) for (int i = (a); i <= (b); i++)
#define fd(i, a, b) for (int i = (a); i >= (b); i--)
const int N = 3e5 + 5, mod = 998244353; // (N = 4 * n)

using ll = int64_t;
using Poly = vector<int>;
/*------------------------------------------------------------------------*/
// 二次剩余
```

```
9  class Cipolla {
10    int mod, I2{};
11    using pll = pair<ll, ll>;
12 #define X first
13 #define Y second
14    ll mul(ll a, ll b) const { return a * b % mod; }
15    pll mul(pll a, pll b) const { return {(a.X * b.X + I2 * a.Y % mod * b.Y) % mod,
          (a.X * b.Y + a.Y * b.X) % mod}; }
16    template<class T> T ksm(T a, int b, T x) { for (; b; b >>= 1, a = mul(a, a)) if
          (b & 1) x = mul(x, a); return x; }
17 public:
18    Cipolla(int p = 0) : mod(p) {}
19    pair<int, int> sqrt(int n) {
20       int a = rand(), x;
21       if (!(n %= mod)) return {0, 0};
22       if (ksm(n, (mod - 1) >> 1, 1ll) == mod - 1) return {-1, -1};
23       while (ksm(I2 = ((ll) a * a - n + mod) % mod, (mod - 1) >> 1, 1ll) == 1) a =
             rand();
24       x = (int) ksm(pll{a, 1}, (mod + 1) >> 1, {1, 0}).X;
25       if (2 * x > mod) x = mod - x;
26       return {x, mod - x};
27    }
28 #undef X
29 #undef Y
30 };
31 /*-------------------------------------Modular---------------------------------*/
32 #define mul(a, b) (ll(a) * (b) % mod)
33 #define add(a, b) (((a) += (b)) >= mod ? (a) -= mod : 0) // (a += b) %= P
34 #define dec(a, b) (((a) -= (b)) < 0 ? (a) += mod: 0) // ((a -= b) += P) %= P
35 Poly getInv(int L) { Poly inv(L); inv[1] = 1; fp(i, 2, L - 1) inv[i] = mul((mod -
       mod / i), inv[mod % i]); return inv; }
36 int ksm(ll a, int b = mod - 2, ll x = 1) { for (; b; b >>= 1, a = a * a % mod) if (b
        & 1) x = x * a % mod; return x; }
37 auto inv = getInv(N); // NOLINT
38 /*---------------------------------NTT-------------------------------------*/
39 namespace NTT {
40    const int g = 3;
41    Poly Omega(int L) {
42       int wn = ksm(g, mod / L);
43       Poly w(L); w[L >> 1] = 1;
44       fp(i, L / 2 + 1, L - 1) w[i] = mul(w[i - 1], wn);
45       fd(i, L / 2 - 1, 1) w[i] = w[i << 1];
46       return w;
47    }
48    auto W = Omega(1 << 20); // NOLINT
49    void DIF(int *a, int n) {
50       for (int k = n >> 1; k; k >>= 1)
51          for (int i = 0, y; i < n; i += k << 1)
52             for (int j = 0; j < k; ++j)
53                y = a[i + j + k], a[i + j + k] = mul(a[i + j] - y + mod, W[k + j]),
                        add(a[i + j], y);
54    }
55    void IDIT(int *a, int n) {
```

```
56        for (int k = 1; k < n; k <<= 1)
57            for (int i = 0, x, y; i < n; i += k << 1)
58                for (int j = 0; j < k; ++j)
59                    x = a[i + j], y = mul(a[i + j + k], W[k + j]),
60                    a[i + j + k] = x - y < 0 ? x - y + mod : x - y, add(a[i + j], y);
61        int Inv = mod - (mod - 1) / n;
62        fp(i, 0, n - 1) a[i] = mul(a[i], Inv);
63        reverse(a + 1, a + n);
64    }
65 }
66 /*-------------------------------------Polynomial 全家桶
      ----------------------------------*/
67 namespace Polynomial {
68    // basic operator
69    int norm(int n) { return 1 << (__lg(n - 1) + 1); }
70    void norm(Poly &a) { if (!a.empty()) a.resize(norm(a.size()), 0); else a = {0};
          }
71    void DFT(Poly &a) { NTT::DIF(a.data(), a.size()); }
72    void IDFT(Poly &a) { NTT::DIT(a.data(), a.size()); }
73    Poly &dot(Poly &a, Poly &b) { fp(i, 0, a.size() - 1) a[i] = mul(a[i], b[i]);
          return a; }
74
75    // mul / div int
76    Poly &operator*=(Poly &a, int b) { for (auto &x : a) x = mul(x, b); return a; }
77    Poly operator*(Poly a, int b) { return a *= b; }
78    Poly operator*(int a, Poly b) { return b * a; }
79    Poly &operator/=(Poly &a, int b) { return a *= ksm(b); }
80    Poly operator/(Poly a, int b) { return a /= b; }
81
82    // Poly add / sub
83    Poly &operator+=(Poly &a, Poly b) {
84        a.resize(max(a.size(), b.size()));
85        fp(i, 0, b.size() - 1) add(a[i], b[i]);
86        return a;
87    }
88    Poly operator+(Poly a, Poly b) { return a += b; }
89    Poly &operator-=(Poly &a, Poly b) {
90        a.resize(max(a.size(), b.size()));
91        fp(i, 0, b.size() - 1) dec(a[i], b[i]);
92        return a;
93    }
94    Poly operator-(Poly a, Poly b) { return a -= b; }
95
96    // Poly mul
97    Poly operator*(Poly a, Poly b) {
98        int n = a.size() + b.size() - 1, L = norm(n);
99        if (a.size() <= 8 || b.size() <= 8) {
100           Poly c(n);
101           fp(i, 0, a.size() - 1) fp(j, 0, b.size() - 1)
102               c[i + j] = (c[i + j] + (ll) a[i] * b[j]) % mod;
103           return c;
104       }
105       a.resize(L), b.resize(L);
```

```
106         DFT(a), DFT(b), dot(a, b), IDFT(a);
107         return a.resize(n), a;
108     }
109
110     // Poly inv
111     Poly Inv2k(Poly a) { // |a| = 2 ^ k
112         int n = a.size(), m = n >> 1;
113         if (n == 1) return {ksm(a[0])};
114         Poly b = Inv2k(Poly(a.begin(), a.begin() + m)), c = b;
115         b.resize(n), DFT(a), DFT(b), dot(a, b), IDFT(a);
116         fp(i, 0, n - 1) a[i] = i < m ? 0 : mod - a[i];
117         DFT(a), dot(a, b), IDFT(a);
118         return move(c.begin(), c.end(), a.begin()), a;
119     }
120     Poly Inv(Poly a) {
121         int n = a.size();
122         norm(a), a = Inv2k(a);
123         return a.resize(n), a;
124     }
125
126     // Poly div / mod
127     Poly operator/(Poly a,Poly b){
128         int k = a.size() - b.size() + 1;
129         if (k < 0) return {0};
130         reverse(a.begin(), a.end());
131         reverse(b.begin(), b.end());
132         b.resize(k), a = a * Inv(b);
133         a.resize(k), reverse(a.begin(), a.end());
134         return a;
135     }
136     pair<Poly, Poly> operator%(Poly a, const Poly& b) {
137         Poly c = a / b;
138         a -= b * c, a.resize(b.size() - 1);
139         return {c, a};
140     }
141
142     // Poly calculus
143     Poly deriv(Poly a) {
144         fp(i, 1, a.size() - 1) a[i - 1] = mul(i, a[i]);
145         return a.pop_back(), a;
146     }
147     Poly integ(Poly a) {
148         a.push_back(0);
149         fd(i, a.size() - 1, 1) a[i] = mul(inv[i], a[i - 1]);
150         return a[0] = 0, a;
151     }
152
153     // Poly ln
154     Poly Ln(Poly a) {
155         int n = a.size();
156         a = deriv(a) * Inv(a);
157         return a.resize(n - 1), integ(a);
158     }
```

```
159
160     // Poly exp
161     Poly Exp(Poly a) {
162         int n = a.size(), k = norm(n);
163         Poly b = {1}, c, d; a.resize(k);
164         for (int L = 2; L <= k; L <<= 1) {
165             d = b, b.resize(L), c = Ln(b), c.resize(L);
166             fp(i, 0, L - 1) c[i] = a[i] - c[i] + (a[i] < c[i] ? mod : 0);
167             add(c[0], 1), DFT(b), DFT(c), dot(b, c), IDFT(b);
168             move(d.begin(), d.end(), b.begin());
169         }
170         return b.resize(n), b;
171     }
172
173     // Poly sqrt
174     Poly Sqrt(Poly a) {
175         int n = a.size(), k = norm(n); a.resize(k);
176         Poly b = {(new Cipolla(mod))->sqrt(a[0]).first, 0}, c;
177         for (int L = 2; L <= k; L <<= 1) {
178             b.resize(L), c = Poly(a.begin(), a.begin() + L) * Inv2k(b);
179             fp(i, L / 2, L - 1) b[i] = mul(c[i], (mod + 1) / 2);
180         }
181         return b.resize(n), b;
182     }
183
184     // Poly pow
185     Poly Pow(Poly &a, int b) { return Exp(Ln(a) * b); } // a[0] = 1
186     Poly Pow(Poly a, int b1, int b2) { // b1 = b % mod, b2 = b % phi(mod) and b >= n
          iff a[0] > 0
187         int n = a.size(), d = 0, k;
188         while (d < n && !a[d]) ++d;
189         if ((ll) d * b1 >= n) return Poly(n);
190         a.erase(a.begin(), a.begin() + d);
191         k = ksm(a[0]), norm(a *= k);
192         a = Pow(a, b1) * ksm(k, mod - 1 - b2);
193         a.resize(n), d *= b1;
194         fd(i, n - 1, 0) a[i] = i >= d ? a[i - d] : 0;
195         return a;
196     }
197
198 }
199 using namespace Polynomial;
```