

XCPC Templates

Khoray

April 4, 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 Ex-Baby-Step-Giant-Step-Algorithm	7
1.8 逆元	9
1.8.1 exgcd 求逆元	9
1.8.2 快速幂求逆元	9
1.8.3 整数除法取模	9
1.9 上下取整	9
1.10 线性基	10
1.11 高斯消元	11
1.11.1 解异或线性方程组	12
1.11.2 解 double 线性方程组	13
1.11.3 解模意义线性方程组	14
1.12 Miller-Rabin	15
1.13 Pollard-Rho	16
1.14 多项式全家桶	17
1.15 数学公式	21
1.15.1 多项式牛顿迭代	21
1.15.2 生成函数/形式幂级数	21
1.15.3 莫比乌斯反演	21
1.15.4 分配问题	21
1.15.5 第二类斯特林数	22
1.15.6 第一类斯特林数	22
1.15.7 分拆数	22
1.15.8 五边形数	22
1.15.9 Polya	23
2 杂项	25
2.1 debugger	25

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 $f(val^{power})$

```

1 // all in which f[pq] = f[p] * f[q] (gcd(p, q) = 1)
2 const int N = 1e7 + 5;
3 int pri[N / 5], notpri[N], prinum, minpri_cnt[N], f[N];
4
5 int calc_f(int val, int power) {
6
7 }
8
9 void init_pri() {
10     for(int i = 2; i < N; i++) {
11         if(!notpri[i]) pri[++prinum] = i, minpri_cnt[i] = 1, f[i] = calc_f(i, 1);
12         for(int j = 1; j <= prinum && pri[j] * i < N; j++) {
13             notpri[pri[j] * i] = pri[j];
14             if(i % pri[j] == 0) {
15                 minpri_cnt[pri[j] * i] = minpri_cnt[i] + 1;
16                 f[pri[j] * i] = f[i] / calc_f(pri[j], minpri_cnt[i]) * calc_f(pri[j],
17                                     minpri_cnt[i] + 1);
18                 break;
19             }
20             minpri_cnt[pri[j] * i] = 1;
21             f[pri[j] * i] = f[i] * calc_f(pri[j], 1);
22         }
23     }
24 }

```

1.2 快速幂

```

1 int ksm(int a, int b = mod - 2, int MOD_KSM = mod) {
2     int ret = 1;
3     while(b) {
4         if(b & 1) {
5             ret = ret * a % MOD_KSM;
6         }
7         a = a * a % MOD_KSM;
8         b >>= 1;
9     }
10    return ret;
11 }

```

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！

```

1 int binom(int n, int k) {
2     if(n < 0 || k < 0 || k > n) { return 0; }
3     k = min(n - k, k);
4     int u = 1, v = 1;
5     for(int i = 0; i < k; i++) {
6         v = v * (i + 1) % mod;
7         u = u * (n - i) % mod;
8     }
9     return u * ksm(v, mod - 2) % mod;
10 }

```

1.3.2 递推

- $O(n^2)$ 递推求，模数随意。

```

1 const int N = 31;
2 int C[N][N];
3
4 void init_C() {
5     C[0][0] = C[1][0] = C[1][1] = 1;
6     for(int i = 2; i < N; i++) {
7         C[i][0] = 1;
8         for(int j = 1; j <= i; j++)
9             C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
10    }
11 }
12
13 int binom(int n, int k) {
14     if(n < 0 || k < 0 || n < k) return 0;
15     return C[n][k];
16 }

```

1.3.3 逆元

- 模数必须是质数！
- 前置：快速幂

```

1 const int N = 1e7 + 5;
2 const int mod = 1e9 + 7;
3 int facinv[N], fac[N];
4 void init_fac() {
5     fac[0] = fac[1] = 1;
6     for(int i = 2; i < N; i++) {
7         fac[i] = fac[i - 1] * i % mod;
8     }
9     facinv[N - 1] = ksm(fac[N - 1], mod - 2);
10    for(int i = N - 2; i >= 0; i--) {

```

```

11     facinv[i] = facinv[i + 1] * (i + 1) % mod;
12 }
13 }
14 int binom(int n, int k) {
15     if(n < 0 || k < 0 || k > n) { return 0; }
16     return fac[n] * facinv[n - k] % mod * facinv[k] % mod;
17 }
18
19 int inv[N];
20 void init_inv() {
21     inv[0] = inv[1] = 1;
22     for(int i = 2; i < N; i++) {
23         inv[i] = (mod - mod / i) * inv[mod % i] % mod;
24     }
25 }

```

1.4 ExGCD

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

```

1 // ax + by = gcd(a, b)
2 // return gcd(a, b)
3 // all sol: x = x_0 + k[a, b] / a, y = y_0 - k[a, b] / b;
4 int exgcd(int &x, int &y, int a, int b) {
5     if(!b) {
6         x = 1;
7         y = 0;
8         return a;
9     }
10    int ret = exgcd(x, y, b, a % b);
11    int t = x;
12    x = y;
13    y = t - a / b * y;
14    return ret;
15 }

```

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 $f(l), \dots, f(r)$ ，求解 $f(n)$ 。 $O(r - l)$
- 前置：逆元组合数
- 模数为质数

```

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

```

```

7   for(int i = 1; i <= r; i++) {
8       lg[i - 1 + 1] = (long long) lg[i - 1] * (n - i) % mod;
9   }
10  for(int i = r; i >= 1; i--) {
11      rg[i - 1 + 1] = (long long) rg[i - 1 + 2] * (n - i) % mod;
12  }
13  for(int i = 1; i <= r; i++) {
14      if(r - i & 1) {
15          ret = (ret - (long long) y[i] * lg[i - 1] % mod * rg[i - 1 + 2] % mod *
16                  facinv[i - 1] % mod * facinv[r - i] % mod + mod) % mod;
17      } else {
18          ret = (ret + (long long) y[i] * lg[i - 1] % mod * rg[i - 1 + 2] % mod *
19                  facinv[i - 1] % mod * facinv[r - i] % mod) % mod;
20      }
21  }
22  return ret;
23 }

```

1.6 原根

若 g 满足:

$$(g, m) = 1$$

$$\delta_m(g) = \phi(m)$$

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 $\forall p | \phi(m) : g^p \neq 1$
2. 找小于 m 与 $\phi(m)$ 互质的数 k , 则 g^k 也是原根 (能覆盖所有原根) 个数为 $\phi(\phi(m))$ 个。

题目: 找出 n 的所有原根, 间隔 d 输出。

```

1  void solve() {
2      // d 是间隔输出 (对题目无影响
3      int n, d; cin >> n >> d;
4      vector<int> pf; // 质因数分解
5      int pn = phi[n];
6      while(notpri[pn]) {
7          int now = notpri[pn];
8          pf.push_back(now);
9          while(pn % now == 0) pn /= now;
10     }
11     if(pn != 1) {
12         pf.push_back(pn);
13     }
14     int cnt = 0;
15     int ming = -1;
16     vector<int> ans, vis(n); // 记录答案
17     // 找到最小的原根 min_g
18     for(int i = 1; i < n; i++) {
19         if(__gcd(i, n) != 1) continue;
20         int judge = 1;
21         for(auto &p : pf) {

```

```

22         if(ksm(i, phi[n] / p, n) == 1) {
23             judge = 0;
24             break;
25         }
26     }
27     if(judge) {
28         ming = i;
29         break;
30     }
31 }
32 // 还原出所有原根 g
33 if(ming > 0) {
34     for(int i = 1; i < n; i++) {
35         if(__gcd(i, phi[n]) == 1) {
36             int cur = ksm(ming, i, n);
37             if(!vis[cur]) {
38                 vis[cur] = 1;
39                 ans.push_back(cur);
40             }
41         }
42     }
43 }
44 // 排序输出所有原根
45 cout << ans.size() << '\n';
46 sort(ans.begin(), ans.end());
47 for(auto as : ans) {
48     cnt++;
49     if(cnt % d == 0) {
50         cout << as << ' ';
51     }
52 }
53 cout << '\n';
54 }

```

1.7 Ex-Baby-Step-Giant-Step-Algorithm

BSGS

求解 $a^x = b \pmod{p}, (0 \leq x < p)$

令 $x = A \lceil \sqrt{p} \rceil - B$, 其中 $0 \leq A, B \leq \lceil \sqrt{p} \rceil$, 则有 $a^{A \lceil \sqrt{p} \rceil - B} \equiv b \pmod{p}$, 稍加变换, 则有 $a^{A \lceil \sqrt{p} \rceil} \equiv ba^B \pmod{p}$ 。

我们已知的是 a, b , 所以我们可以先算出等式右边的 ba^B 的所有取值, 枚举 B , 用 ‘hash’/‘map’ 存下来, 然后逐一计算 $a^{A \lceil \sqrt{p} \rceil}$, 枚举 A , 寻找是否有与之相等的 ba^B , 从而我们可以得到所有的 x , $x = A \lceil \sqrt{p} \rceil - B$ 。

注意到 A, B 均小于 $\lceil \sqrt{p} \rceil$, 所以时间复杂度为 $\Theta(\sqrt{p})$, 用 ‘map’ 则多一个 \log 。

exBSGS

其中 a, p 不一定互质。

当 $a \perp p$ 时, 在模 p 意义下 a 存在逆元, 因此可以使用 BSGS 算法求解。于是我们想办法让他们变得互质。

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

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

同理, 这样不停的判断下去。直到 $a \perp \frac{p}{d_1 d_2 \cdots d_k}$ 。

记 $D = \prod_{i=1}^k d_i$, 于是方程就变成了这样:

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

由于 $a \perp \frac{p}{D}$, 于是推出 $\frac{a^k}{D} \perp \frac{p}{D}$ 。这样 $\frac{a^k}{D}$ 就有逆元了, 于是把它丢到方程右边, 这就是一个普通的 BSGS 问题了, 于是求解 $x - k$ 后再加上 k 就是原方程的解啦。

注意, 不排除解小于等于 k 的情况, 所以在消因子之前做一下 $\Theta(k)$ 枚举, 直接验证 $a^i \equiv b \pmod{p}$, 这样就能避免这种情况。

- 注意, inv 必须由扩欧求!
- 注意开 long long
- 前置: ksm, exgcd 求逆元

```

1 int bsgs(int a, int b, int p) { //BSGS算法
2     unordered_map<int, int> f;
3     int m = ceil(sqrt(p));
4     b %= p;
5     for(int i = 1; i <= m; i++) {
6         b = b * a % p;
7         f[b] = i;
8     }
9     int tmp = ksm(a, m, p);
10    b = 1;
11    for(int i = 1; i <= m; i++) {
12        b = b * tmp % p;
13        if(f[b]) {
14            return (i * m - f[b] + p) % p;
15        }
16    }
17    return -1;
18 }
19 int exbsgs(int a, int b, int p) {
20     b %= p;
21     a %= p;
22     if(b == 1 || p == 1) {
23         return 0; //特殊情况, x=0时最小解
24     }
25     int g = __gcd(a, p), k = 0, na = 1;
26     while(g > 1) {
27         if(b % g != 0) {
28             return -1; //无法整除则无解
29         }
30         k++;
31         b /= g;
32         p /= g;

```



```

33     na = na * (a / g) % p;
34     if(na == b) {
35         return k; //na=b说明前面的a的次数为0, 只需要返回k
36     }
37     g = __gcd(a, p);
38 }
39 int f = bsgs(a, b * inv(na, p) % p, p);
40 if(f == -1) {
41     return -1;
42 }
43 return f + k;
44 }

```

1.8 逆元

1.8.1 exgcd 求逆元

- 前置: exgcd
- $(x, p) = 1$

```

1 int inv(int x, int p) {
2     int y, k;
3     int gcd = exgcd(y, k, x, p);
4     int moder = p / gcd;
5     return (y % moder + moder) % moder;
6 }

```

1.8.2 快速幂求逆元

根据费马小定理: $p \in \text{primes} \rightarrow a^{-1} \equiv a^{p-2} \pmod{p}$

1.8.3 整数除法取模

如果 $\frac{a}{b} \in \mathbb{N}$, $b \times p$ 可以在计算机中表示, 那么 $\frac{a}{b} \bmod p = \frac{a \bmod (p \times b)}{b}$

1.9 上下取整

- b 必须为正整数。

```

1 // b must be positive integer
2
3 int updiv(int a, int b) {
4     return a > 0 ? (a + b - 1) / b : a / b;
5 }
6
7 int downdiv(int a, int b) {
8     return a > 0 ? a / b : (a - b + 1) / b;
9 }

```

1.10 线性基

- $O(\log x)$ insert
- $O(\log^2 x)$ get-kth
- $O(\log x)$ get-max
- 如果问能否通过选一些数（不能不选）异或得到 0，必须特判。

```

1 struct linear_basis {
2     vector<int> base, kth;
3     int size, max_size, builded;
4     linear_basis(int n) : base(n), size(0), max_size(n), builded(0) {}
5
6     void insert(int x) {
7         builded = 0;
8         for(int i = max_size - 1; i >= 0; i--) {
9             if((x >> i) & 1) {
10                 if(!base[i]) {
11                     base[i] = x, size++;
12                     break;
13                 }
14                 else x ^= base[i];
15             }
16         }
17     }
18
19     int get_max() {
20         int ret = 0;
21         for(int i = max_size - 1; i >= 0; i--) {
22             if(!((ret >> i) & 1) && base[i]) ret ^= base[i];
23         }
24         return ret;
25     }
26
27     bool can_eq(int x) {
28         int now = 0;
29         for(int i = max_size - 1; i >= 0; i--) {
30             if(((now >> i) & 1) != ((x >> i) & 1)) {
31                 if(!base[i]) return false;
32                 else now ^= base[i];
33             }
34         }
35         return true;
36     }
37
38
39     int get_kth(int k) {
40         if(k >= 1ll << size) return -1;
41         if(!builded) buildk();
42         int ret = 0;
43         for(int i = size - 1; ~i; i--) {
44             if(k >> i & 1) {

```

```

45         ret ^= kth[i];
46     }
47 }
48 return ret;
49 }
50
51 int get_rank(int x) { // return the number of values less than x TODO
52     int tmpsz = size, ret = 0, now = 0;
53     for(int i = max_size - 1; i >= 0; i--) {
54         if(base[i]) tmpsz--;
55         if((x >> i) & 1) {
56             if(!((now >> i) & 1)) {
57                 ret += tmpsz * tmpsz;
58                 if(!base[i]) break;
59                 else now ^= base[i];
60             } else {
61                 if(base[i]) ret += tmpsz * tmpsz;
62             }
63         } else {
64             if((now >> i) & 1) {
65                 if(!base[i]) break;
66                 else now ^= base[i];
67             }
68         }
69     }
70     return ret;
71 }
72 private:
73 void buildk() {
74     builded = 1;
75     kth.resize(size);
76     int cnt = size;
77     for(int i = max_size - 1; ~i; i--) {
78         if(base[i]) {
79             for(int j = i - 1; ~j; j--) {
80                 if(base[i] >> j & 1) {
81                     base[i] ^= base[j];
82                 }
83             }
84         }
85     }
86     for(int i = max_size - 1; ~i; i--) {
87         if(base[i]) {
88             kth[--cnt] = base[i];
89         }
90     }
91 }
92 };

```

1.11 高斯消元

- equ 是方程个数，n 是变元个数，答案存在 ans。

- return : 无解 (-1), 自由变元个数。

1.11.1 解异或线性方程组

```
1  const int N = 1005;
2  template<int N>
3  struct Matrix {
4      bitset<N> mat[N];
5      Matrix() { }
6      bitset<N> &operator [] (int idx) { return mat[idx]; }
7  };
8
9  template<int N>
10 int guass(int n, int equ, Matrix<N> a, vector<int> b, vector<int> &ans) {
11     fill(ans.begin(), ans.end(), 0);
12     vector<int> fre(n + 1);
13     int row, col;
14     for(row = 1, col = 1; col <= n; col++) {
15         if(!a[row][col]) {
16             int sw = 0;
17             for(int i = row + 1; i <= equ; i++) {
18                 if(a[i][col]) {
19                     swap(a[row], a[i]);
20                     swap(b[row], b[i]);
21                     sw = 1;
22                     break;
23                 }
24             }
25             if(!sw) {
26                 fre[col] = 1;
27                 continue;
28             }
29         }
30         for(int i = row + 1; i <= equ; i++) {
31             if(a[i][col]) {
32                 a[i] ^= a[row];
33                 b[i] ^= b[row];
34             }
35         }
36         row++;
37     }
38     if(row <= equ) {
39         for(int i = row; i <= equ; i++) {
40             if(b[i]) return -1;
41         }
42     }
43     int all = 0;
44     for(col = n; col >= 1; col--) {
45         if(fre[col]) {
46             ans[col] = 1;
47             all++;
48         } else {
49             row--;
```

```

50     ans[col] = b[row];
51     for(int i = col + 1; i <= n; i++) {
52         if(a[row][i]) ans[col] ^= ans[i];
53     }
54 }
55 }
56 return all;
57 }

```

1.11.2 解 double 线性方程组

```

1  const int N = 1005;
2  template<int N>
3  struct Matrix {
4      bitset<N> mat[N];
5      Matrix() { }
6      bitset<N> &operator [] (int idx) { return mat[idx]; }
7  };
8  template<int N>
9  int gauss(int n, int equ, Matrix<N> a, vector<double> b, vector<double> &ans) {
10     fill(ans.begin(), ans.end(), 0);
11     vector<int> fre(n + 1);
12     int row, col;
13     for(row = 1, col = 1; col <= n; col++) {
14         double mx = fabs(a[row][col]);
15         int mxp = row;
16         for(int i = row + 1; i <= equ; i++) {
17             if(fabs(a[i][col]) > mx) {
18                 mx = fabs(a[i][col]);
19                 mxp = i;
20             }
21         }
22         if(mxp != row) {
23             for(int i = col; i <= n; i++) {
24                 swap(a[row][i], a[mxp][i]);
25             }
26             swap(b[row], b[mxp]);
27         }
28         if(fabs(a[row][col]) < eps) {
29             fre[col] = 1;
30         }
31         for(int i = row + 1; i <= equ; i++) {
32             if(fabs(a[i][col]) > eps) {
33                 double k = a[i][col] / a[row][col];
34                 for(int j = col; j <= n; j++) {
35                     a[i][j] -= a[row][j] * k;
36                 }
37                 b[i] -= b[row] * k;
38             }
39         }
40         row++;
41     }

```

```

42 // 判断解是否存在
43 if(row <= equ) {
44     for(int i = row; i <= equ; i++) {
45         if(fabs(b[i]) > eps) return -1;
46     }
47 }
48
49 // 回代求解
50 int all = 0;
51 for(col = n; col >= 1; col--) {
52     if(fre[col]) {
53         ans[col] = 0;
54         all++;
55     } else {
56         row--;
57         ans[col] = b[row];
58         for(int i = col + 1; i <= n; i++) {
59             ans[col] -= ans[i] * a[row][i];
60         }
61         ans[col] /= a[row][col];
62     }
63 }
64 return all;
65 }

```

1.11.3 解模意义线性方程组

- 时间复杂度 $O(n^3 \log mod)$

```

1 #define mul(a, b) (1l(a) * (b) % mod)
2 #define add(a, b) (((a) += (b)) >= mod ? (a) -= mod : 0) // (a += b) %= P
3 #define dec(a, b) (((a) -= (b)) < 0 ? (a) += mod : 0) // ((a -= b) += P) %= P
4
5 const int N = 1005;
6 const int mod = 1e9 + 7;
7 template<int N>
8 struct Matrix {
9     int mat[N][N];
10     Matrix() { }
11     int* operator [] (int idx) { return mat[idx]; }
12 };
13 template<int N>
14 int guass(int n, int equ, Matrix<N> a, vector<int> b, vector<int> &ans) {
15     fill(ans.begin(), ans.end(), 0);
16     vector<int> fre(n + 1);
17     int row, col;
18     for(row = 1, col = 1; col <= n; col++) {
19         if(!a[row][col]) {
20             int sw = 0;
21             for(int i = row + 1; i <= equ; i++) {
22                 if(a[i][col]) {
23                     for(int j = col; j <= n; j++) {

```

```

24         swap(a[row][j], a[i][j]);
25     }
26     swap(b[row], b[i]);
27     sw = 1;
28     break;
29 }
30 }
31 if(!sw) {
32     fre[col] = 1;
33     continue;
34 }
35 }
36 for(int i = row + 1; i <= equ; i++) {
37     if(a[i][col]) {
38         int k = a[i][col] * ksm(a[row][col]) % mod;
39         for(int j = col; j <= n; j++) {
40             dec(a[i][j], a[row][j] * k % mod);
41         }
42         dec(b[i], b[row] * k % mod);
43     }
44 }
45 row++;
46 }
47 if(row <= equ) {
48     for(int i = row; i <= equ; i++) {
49         if(b[i]) return -1;
50     }
51 }
52 int all = 0;
53 for(col = n; col >= 1; col--) {
54     if(fre[col]) {
55         ans[col] = 0;
56         all++;
57     } else {
58         row--;
59         ans[col] = b[row];
60         for(int i = col + 1; i <= n; i++) {
61             dec(ans[col], ans[i] * a[row][i] % mod);
62         }
63         mul(ans[col], ksm(a[row][col]));
64     }
65 }
66 return all;
67 }

```

1.12 Miller-Rabin

- 前置：快速幂 (___int128!!!)
- int 范围: 2, 7, 61
- long long 范围: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- 4E13: 2, 2570940, 211991001, 3749873356

- 3E15: 2, 2570940, 880937, 610386380, 4130785767
- 注意看判断范围是 int 还是 long long

```

1 bool is_prime(int n) {
2     if(n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
3     int A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
4         s = __builtin_ctzll(n - 1), d = n >> s;
5     for(int a : A) { // ^ count t r a i l i n g z e r o e s
6         int p = ksm(a % n, d, n), i = s;
7         while(p != 1 && p != n - 1 && a % n && i--)
8             p = (__int128) p * p % n;
9         if(p != n - 1 && i != s) return 0;
10    }
11    return 1;
12 }

```

1.13 Pollard-Rho

- 前置: Miller-Rabin

```

1 inline int pollard_rho(int x) {
2     auto f = [&](int x, int c, int n) {
3         return ((__int128) x * x + c) % n;
4     };
5     int s = 0, t = 0, c = 111 * rand() % (x - 1) + 1;
6     int stp = 0, goal = 1;
7     int val = 1;
8     for(goal = 1;; goal <= 1, s = t, val = 1) {
9         for(stp = 1; stp <= goal; ++stp) {
10            t = f(t, c, x);
11            val = (__int128)val * abs(t - s) % x;
12            if((stp % 127) == 0) {
13                int d = __gcd(val, x);
14                if(d > 1)
15                    return d;
16            }
17        }
18        int d = __gcd(val, x);
19        if(d > 1)
20            return d;
21    }
22 }
23
24 inline void get_factor_a(int x, vector<int> &fac) {
25     if(x < 2) return;
26     if(is_prime(x)) {
27         fac.push_back(x);
28         return;
29     }
30     int p = x;
31     while(p >= x) p = pollard_rho(x);

```



```

32 while((x % p) == 0) x /= p;
33 get_factor_a(x, fac), get_factor_a(p, fac);
34 }

```

1.14 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
- 注意这里的 `ksm()` 第三个参数是初值而不是模数

```

1 #define fp(i, a, b) for (int i = (a); i <= (b); i++)
2 #define fd(i, a, b) for (int i = (a); i >= (b); i--)
3 const int N = 3e5 + 5, mod = 998244353; // (N = 4 * n)
4
5 using ll = int64_t;
6 using Poly = vector<int>;
7 /*-----*/
8 // 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,
16         (a.X * b.Y + a.Y * b.X) % mod}; }
17     template<class T> T ksm(T a, int b, T x) { for (; b >= 1; a = MUL(a, a)) if
18         (b & 1) x = MUL(x, a); return x; }
19 public:
20     Cipolla(int p = 0) : mod(p) {}
21     pair<int, int> sqrt(int n) {
22         int a = rand(), x;
23         if (!(n % mod)) return {0, 0};
24         if (ksm(n, (mod - 1) >> 1, 1ll) == mod - 1) return {-1, -1};
25         while (ksm(I2 = ((1ll) a * a - n + mod) % mod, (mod - 1) >> 1, 1ll) == 1) a =
26             rand();
27         x = (int) ksm(pll{a, 1}, (mod + 1) >> 1, {1, 0}).X;
28         if (2 * x > mod) x = mod - x;
29         return {x, mod - x};
30     }
31 }
32 #undef X
33 #undef Y
34 };
35 /*-----Modular-----*/
36 #define MUL(a, b) (ll(a) * (b) % mod)
37 #define ADD(a, b) (((a) += (b)) >= mod ? (a) -= mod : 0) // (a += b) %= P
38 #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 >= 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]),
54                         ADD(a[i + j], y);
55     }
56     void IDIT(int *a, int n) {
57         for (int k = 1; k < n; k <= 1)
58             for (int i = 0, x, y; i < n; i += k << 1)
59                 for (int j = 0; j < k; ++j)
60                     x = a[i + j], y = MUL(a[i + j + k], W[k + j]),
61                         a[i + j + k] = x - y < 0 ? x - y + mod : x - y, ADD(a[i + j], y);
62         int Inv = mod - (mod - 1) / n;
63         fp(i, 0, n - 1) a[i] = MUL(a[i], Inv);
64         reverse(a + 1, a + n);
65     }
66 }
67 /*-----Polynomial 全家桶-----*/
68 namespace Polynomial {
69     // basic operator
70     int norm(int n) { return 1 << (___lg(n - 1) + 1); }
71     void norm(Poly &a) { if (!a.empty()) a.resize(norm(a.size()), 0); else a = {0}; }
72     void DFT(Poly &a) { NTT::DIF(a.data(), a.size()); }
73     void IDFT(Poly &a) { NTT::IDIT(a.data(), a.size()); }
74     Poly &dot(Poly &a, Poly &b) { fp(i, 0, a.size() - 1) a[i] = MUL(a[i], b[i]);
75         return a; }
76     // MUL / div int
77     Poly &operator*=(Poly &a, int b) { for (auto &x : a) x = MUL(x, b); return a; }
78     Poly operator*(Poly a, int b) { return a *= b; }
79     Poly operator*(int a, Poly b) { return b * a; }
80     Poly &operator/=(Poly &a, int b) { return a /= ksm(b); }
81     Poly operator/(Poly a, int b) { return a /= b; }

```

```

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 ((1ll) 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;

```

1.15 数学公式

1.15.1 多项式牛顿迭代

1.15.2 生成函数/形式幂级数

1.15.3 莫比乌斯反演

1.15.4 分配问题

1. n 个球放到 k 个盒子，每个盒子只有一种形态。

n 个球	k 个盒子	盒子可以为空	每个盒子内至少有一个球
有标号	有标号	k^n	$k!S_2(n, k)$
有标号	无标号	$\sum_{i=1}^k S_2(n, i)$	$S_2(n, k)$
无标号	有标号	$C(n+k-1, k-1)$	$C(n-1, k-1)$
无标号	无标号	$p(n+k, k)$	$p(n, k)$

其中 $S_2(n, k)$ 为第二类 ‘Stirling 数’， $p(n, k)$ 为 ‘分拆数’

2. n 个球放到 k 个盒子，每个盒子至少一个球，装有 i 个球的盒子有 f_i ($i \geq 1$) 种形态。

$F(x)$ 是 f_i 的 o.g.f.， $E(x)$ 是 f_i 的 e.g.f.

n 个球	k 个盒子	关于 n 方案的生成函数
有标号	有标号	e.g.f = $E(x)^k$
有标号	无标号	e.g.f = $\frac{1}{k!} E(x)^k$
无标号	有标号	o.g.f = $F(x)^k$
无标号	无标号	不会

3. n 个球放到若干盒子，每个盒子至少一个球，装有 i 个球的盒子有 f_i ($i \geq 1$) 种形态。

n 个球	盒子	方案的生成函数
有标号	有标号	e.g.f = $\frac{1}{1-E(x)}$
有标号	无标号	e.g.f = $\exp(E(x))$
无标号	有标号	o.g.f = $\frac{1}{1-F(x)}$
无标号	无标号	o.g.f = $\prod_{i \geq 1} \left(\frac{1}{1-x^i} \right)^{f_i} = \exp \left(\sum_{j \geq 1} \frac{1}{j} F(x^j) \right)$

1.15.5 第二类斯特林数

性质:

1. $\{n\}_k = \{n-1\}_k + k\{n-1\}_{k-1}$ 边界条件 $\{x \neq 1\}_0 = 0, \{0\}_0 = 1$
2. $\{n\}_k = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n$ (用来求第二类斯特林数 · 行, 考虑容斥)
3. $k^n = \sum_{i=0}^k \{n\}_i k^i$ (性质 2 的反演)
4. 重要公式: $x^n = \sum_{k=0}^n \{n\}_k x^k$ (基于 3, x 个球 n 个盒子)
5. 关于 n 的 $e.g.f. = \frac{(e^x - 1)^k}{k!}$ (考虑将 n 个物品染成 k 种颜色, 每一种物品的指数生成函数为 $e^x - 1$) (用于求第二类斯特林数 · 列)

1.15.6 第一类斯特林数

性质:

1. $[n]_k = [n-1]_k + (n-1)[n-1]_{k-1}$ 边界条件 $[x \neq 1]_0 = 0, [0]_0 = 1$
2. 重要公式: $x^n = \sum_{k=0}^n [n]_k x^k$ (x 个球 n 个盒子) (用于计算第一类斯特林数 · 行)
3. 有符号的第一类斯特林数: $S_1(n, k) = (-1)^{n+k} [n]_k$

$$x^n = \sum_{k=0}^n S_1(n, k) x^k$$
4. 关于 n 的 $e.g.f. = \frac{(-\ln(1-x))^k}{k!}$. (列)

1.15.7 分拆数

性质:

1. $p(n, k) = p(n-1, k-1) + p(n-k, k)$
2. $o.g.f. = x^k \prod_{i=1}^k \frac{1}{1-x^i}$ (考虑 Ferrers 图中长度为 i 的一列有多少个)

1.15.8 五边形数

性质:

1. $p(n) = \sum_{k=1}^n p(n, k)$
2. $o.g.f. = \prod_{i \geq 1} \frac{1}{1-x^i}$
 考虑求 \ln , $o.g.f. = \exp(\sum_{n \geq 1} \sum_{d|n} \frac{x^n}{d}) O(n \log n)$
3. 递推公式:
 - (a) 考虑 $Q(x) = \prod_{i \geq 1} (1-x^i)$
 - (b) $Q(x) = \sum_{n \geq 0} (q_{\text{even}}(n) - q_{\text{odd}}(n)) x^n$ $q_{\text{even/odd}}$ 表示有偶数/奇数行, 每一行的个数都不相同, 大部分 $q_{\text{even}}(n)$ 和 $q_{\text{odd}}(n)$ 抵消, 小部分也就是有 b 行, $n = \frac{b(3b-1)}{2}$ 和 $n = \frac{b(3b+1)}{2}$ 无法抵消. 其系数都是 $(-1)^b$

$$(c) \text{ 则 } Q(x) = 1 + \sum_{i \geq 1} (-1)^i x^{\frac{(3i+1)i}{2}}$$

$$(d) P(x) = Q^{-1}(x) \text{ 则 } p(n) = \sum_{k \geq 1} (-1)^{k-1} \left(p\left(n - \frac{(3k-1)k}{2}\right) + p\left(n - \frac{(3k+1)k}{2}\right) \right)$$

1.15.9 Polya

置换群

G 是置换的集合, \circ 是置换的复合, 且 (G, \circ) 为一个群时, 称 (G, \circ) 为一个置换群。

旋转群:

设 n 元环的 n 个结点分别为 a_1, a_2, \dots, a_n , 旋转操作可以看成 $A = a_1, \dots, a_n$ 上的 n 个置换, 其中第 i 个置换为 $g_i = [i+1, i+2, \dots, n, 1, \dots, i]$ 。

设集合 $G = \{g_0, g_1, \dots, g_{n-1}\}$, 则 (G, \circ) 是一个置换群, 称为正 n 边形的旋转群。

群对集合的作用

一个操作会将一个对象改变为另一个对象, 形式化地:

设 (G, \circ) 是一个群, 其单位元为 e , X 是一个集合, 群 G 对集合 X 的一个作用是一个 $G \times X$ 到 X 的映射 f , 满足:

- $\forall x \in X. f(e, x) = x$
- $\forall g, h \in G. f(h \circ g, x) = f(h, f(g, x))$ 我们把 $f(g, x)$ 简记成 $g_f(x)$ 或 (在没有歧义的情况下记成) $g(x)$ 。
- 设 n 元环的 n 个结点分别为 a_1, a_2, \dots, a_n , 令 $A = \{a_1, \dots, a_n\}$ 。设颜色集合为 $B = \{b_1, b_2, \dots, b_m\}$ 。给 n 元环染色可以看成 A 到 B 的一个映射 $x: A \rightarrow B$, 令 X 是所有这些映射的集合, 即 $X = \{x \mid x: A \rightarrow B\}$ 。
- 令 $G = \{g_0, g_1, \dots, g_{n-1}\}$ 是正 n 边形的旋转群, 定义 $G \times X$ 到集合 X 的映射 f , 其中 $\forall i = 0..n-1, x \in X. y = f(g_i, x)$ 满足 $\forall j = 1..n. y(a_j) = x(g_i(a_j))$, 可以证明 f 是群 G 到集合 X 的一个作用, 因此我们简记 $y = f(g_i, x)$ 为 $y = g_i(x)$ 。
- X 上的 G 关系为 $R_G = \{(x, y) \mid x, y \in X \wedge (\exists g \in G. y = g(x))\}$, $x R_G y$ 当且仅当染色方案 x 能通过旋转得到 y 。

要求所有不同的染色方案, 即是求 X 上的 G -轨道的数量。

Burnside 引理

设有限群 (G, \circ) 作用在有限集 X 上, 则 X 上的 G -轨道数量为

$$N = \frac{1}{|G|} \sum_{g \in G} \Psi(g)$$

其中 $\Psi(g)$ 表示 $g(x) = x$ 的 x 的数量。

轮换指标

设 (G, \circ) 是一个 n 元置换的置换群, 它的轮换指标为

$$P_G(x_1, x_2, \dots, x_n) = \frac{1}{|G|} \sum_{g \in G} x_1^{b_1} x_2^{b_2} \dots x_n^{b_n}$$

$x_i^{b_i}$ 表示 g 这个置换的长度为 i 的环有 b_i 个。

正 n 边形旋转群轮换指标:

$$P_G = \frac{1}{n} \sum_{d|n} \varphi(d) x_d^{n/d}$$

正 n 边形二面体群轮换指标 (即可对称):

$$P_G = \frac{1}{2n} \sum_{d|n} \varphi(d) x_d^{n/d} + \begin{cases} \frac{1}{2} x_1 x_2^{\frac{n-1}{2}}, & n \text{ 为奇数} \\ \frac{1}{4} \left(x_2^{\frac{n}{2}} + x_1^2 x_2^{\frac{n-2}{2}} \right), & n \text{ 为偶数} \end{cases}$$

正方体置换群:

顶点置换群:

$$P_G = \frac{1}{24} (x_1^8 + 8x_1^2 x_3^2 + 9x_2^4 + 6x_4^2)$$

边置换群:

$$P_G = \frac{1}{24} (x_1^{12} + 8x_3^4 + 6x_1^2 x_2^5 + 3x_2^6 + 6x_4^3)$$

面置换群:

$$P_G = \frac{1}{24} (x_1^6 + 8x_3^2 + 6x_2^3 + 3x_1^2 x_2^2 + 6x_1^2 x_4)$$

Polya 定理

集合 X 可以看成是给集合 $A = \{a_1, a_2, \dots, a_n\}$ 的每个元素赋予式样 (颜色, 种类等) 的映射的集合

引入表示式样的集合 B , 令 $X = \{x \mid x: A \rightarrow B\}$, 记为 B^A

式样清单: G 作用在 B^A 上的 G -轨道的集合称为 B^A 关于 G 的**式样清单**, 记为 F

种类的权值: 假设 B 上的每个元素 b 都赋予了权值 $w(b)$

$f \in B^A$ 的权值: 定义 $w(f) := \prod_{a \in A} w(f(a))$

G -轨道的权值: $w(F) := w(f)$, 任选一个 $f \in F$

定理:

B^A 关于 G 的**式样清单**记为 \mathcal{F} , 则

$$\sum_{F \in \mathcal{F}} w(F) = P_G \left(\sum_{b \in B} w(b), \sum_{b \in B} w(b)^2, \dots, \sum_{b \in B} w(b)^n \right)$$

2 杂项

2.1 debugger

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 #define out(args...) { cout << "Line " << __LINE__ << ": [" << #args << "] = [";
5     debug(args); cout << "]\n"; }
6
7
8 template<typename T> void debug(T a) { cout << a; }
9
10 template<typename T, typename...args> void debug(T a, args...b) {
11     cout << a << ", ";
12     debug(b...);
13 }
14
15 template<typename T>
16 ostream& operator << (ostream &os, const vector<T> &a) {
17     os << "[";
18     int f = 0;
19     for(auto &x : a) os << (f++ ? ", " : "") << x;
20     os << "]";
21     return os;
22 }
23
24 template<typename T>
25 ostream& operator << (ostream &os, const set<T> &a) {
26     os << "{";
27     int f = 0;
28     for(auto &x : a) os << (f++ ? ", " : "") << x;
29     os << "}";
30     return os;
31 }
32
33 template<typename T>
34 ostream& operator << (ostream &os, const multiset<T> &a) {
35     os << "{";
36     int f = 0;
37     for(auto &x : a) os << (f++ ? ", " : "") << x;
38     os << "}";
39     return os;
40 }
41
42 template<typename A, typename B>
43 ostream& operator << (ostream &os, const map<A, B> &a) {
44     os << "{";
45     int f = 0;
46     for(auto &x : a) os << (f++ ? ", " : "") << x;
47     os << "}";
48     return os;
49 }
```

```
49 template<typename A, typename B>
50 ostream& operator << (ostream &os, const pair<A, B> &a) {
51     os << "(" << a.first << ", " << a.second << ")";
52     return os;
53 }
54
55 template<typename A, size_t N>
56 ostream& operator << (ostream &os, const array<A, N> &a) {
57     os << "{";
58     int f = 0;
59     for(int i = 0; i < N; i++) {
60         os << (f++ ? ", " : "") << a[i];
61     }
62     os << "}";
63     return os;
64 }
```