

XCPC Templates

Khoray

October 8, 2022

Contents

1 Data Structure

1.1

1.2 线段树

- 维护的信息 Data 满足么半群性质（有么元，结合，封闭）。
- Mapping 是区间修改操作，必须满足可以进行 Composition（用于合并 Lazy 标记），Mapping 需要存在么元。

```

1  template<class S, S (*merge)(S, S), S (*s_id)(), class F, S (*mapping)(F, S), F (*
    composition)(F, F), F (*f_id)()>
2  struct lazy_segment_tree {
3      int sz;
4      vector<S> a;
5      struct node {
6          int l, r;
7          S val;
8          F tag;
9          node *lc, *rc;
10     };
11
12     node *root;
13     lazy_segment_tree(int n) : sz(n), a(n + 1), root(new node()) {}
14     lazy_segment_tree(vector<S> &x) : sz(x.size() - 1), a(x), root(new node()) {
        build(root, 1, sz); }
15
16     void build(node *now, int L, int R) {
17         now->l = L, now->r = R, now->val = s_id(), now->tag = f_id();
18         if(L == R) {
19             now->val = a[L];
20             return;
21         }
22         int mid = L + R >> 1;
23         build(now->lc = new node(), L, mid);
24         build(now->rc = new node(), mid + 1, R);
25         now->val = merge(now->lc->val, now->rc->val);
26     }
27
28     void build(int L, int R) {
29         build(root, L, R);
30     }
31
32     S query(node *now, int L, int R) {
33         if(now->l > R || now->r < L) return s_id();
34         if(now->l >= L && now->r <= R) return now->val;
35         push_down(now);
36         return merge(query(now->lc, L, R), query(now->rc, L, R));
37     }
38
39     S query(int L, int R) {
40         return query(root, L, R);
41     }

```

```

42
43 void update(node *now, int pos, S val) {
44     if(now->l == now->r && now->l == pos) {
45         now->val = val;
46         return;
47     }
48     int mid = now->l + now->r >> 1;
49     push_down(now);
50     update(pos <= mid ? now->lc : now->rc, pos, val);
51     now->val = merge(now->lc->val, now->rc->val);
52 }
53
54 void update(int pos, S val) {
55     update(root, pos, val);
56 }
57
58 void push_down(node *now) {
59     now->lc->val = mapping(now->tag, now->lc->val);
60     now->rc->val = mapping(now->tag, now->rc->val);
61     now->lc->tag = composition(now->tag, now->lc->tag);
62     now->rc->tag = composition(now->tag, now->rc->tag);
63     now->tag = f_id();
64 }
65
66 void update_range(node *now, int L, int R, F f) {
67     if(now->l > R || now->r < L) return;
68     if(now->l >= L && now->r <= R) {
69         now->val = mapping(f, now->val);
70         now->tag = composition(f, now->tag);
71         return;
72     }
73     push_down(now);
74     update_range(now->lc, L, R, f);
75     update_range(now->rc, L, R, f);
76     now->val = merge(now->lc->val, now->rc->val);
77 }
78
79 void update_range(int L, int R, F f) {
80     update_range(root, L, R, f);
81 }
82
83 template<class T>
84 pair<int, S> find_r(node *now, int pos, S now_val, T check_val) {
85     // 如果线段树的区间完全小于要查询的点
86     if(now->r < pos) return {sz + 1, s_id()};
87     // 如果线段树的区间完全大于要查询的点
88     if(now->l >= pos) {
89         S all_val = merge(now_val, now->val);
90         if(check_val(all_val)) return {sz + 1, all_val};
91         if(now->l == now->r) return {now->l, all_val};
92     }
93     // 如果不满足条件, 在这个区间内二分
94     auto [lp, lval] = find_r(now->lc, pos, now_val, check_val);

```

```
95     if(lp != sz + 1) {
96         return {lp, lval};
97     } else {
98         return find_r(now->rc, pos, lval, check_val);
99     }
100 }
101
102 template<class T>
103 pair<int, S> find_r(int pos, S now_val, T check_val) {
104     return find_r(root, pos, now_val, check_val);
105 }
106 };
```

1.3 珂朵莉树

2 Math

2.1 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 }

```

2.2 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 = (ll) ret * a % MOD_KSM;
6         }
7         a = (ll) a * a % MOD_KSM;
8         b >>= 1;
9     }
10    return ret;
11 }

```

2.3 组合数

2.3.1 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 }

```

2.3.2 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 }

```

2.3.3 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] = (1ll) 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] = (1ll) 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 (1ll) 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 }

```

2.4 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 }

```

2.5 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] = (1ll) lg[i - 1] * (n - i) % mod;
9   }
10  for(int i = r; i >= 1; i--) {
11      rg[i - 1 + 1] = (1ll) rg[i - 1 + 2] * (n - i) % mod;
12  }
13  for(int i = 1; i <= r; i++) {
14      if(r - i & 1) {
15          ret = (ret - (1ll) y[i] * lg[i - 1] % mod * rg[i - 1 + 2] % mod * facinv[i
16              - 1] % mod * facinv[r - i] % mod + mod) % mod;
17      } else {
18          ret = (ret + (1ll) y[i] * lg[i - 1] % mod * rg[i - 1 + 2] % mod * facinv[i
19              - 1] % mod * facinv[r - i] % mod) % mod;
20      }
21  }
22  return ret;
23 }

```

2.6 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 $\forall p | \phi(m) : g^{\frac{\phi(m)}{p}} \neq 1$
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 }

```

2.7 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 }

```

2.8 逆元

2.8.1 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 }

```

2.8.2 快速幂求逆元

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

2.8.3 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 $\frac{a}{b} \bmod p = \frac{a \bmod (p \times b)}{b}$

2.9 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 }

```

2.10 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     /// @brief 构造线性基，向量长度是n
5     /// @param n
6     linear_basis(int n) : base(n), size(0), max_size(n), builded(0) {}
7     /// @brief 插入一个数
8     /// @param x
9     void insert(int x) {
10         builded = 0;
11         for(int i = max_size - 1; i >= 0; i--) {
12             if((x >> i) & 1) {
13                 if(!base[i]) {
14                     base[i] = x, size++;
15                     break;
16                 }
17                 else x ^= base[i];
18             }
19         }
20     }
21     /// @brief 获取最大值
22     /// @return 最大值
23     int get_max() {
24         int ret = 0;
25         for(int i = max_size - 1; i >= 0; i--) {
26             if(!((ret >> i) & 1) && base[i]) ret ^= base[i];
27         }
28         return ret;
29     }
30     /// @brief 查询是否能等于x
31     /// @param x
32     /// @return 查询是否能等于x
33     bool can_eq(int x) {
34         int now = 0;
35         for(int i = max_size - 1; i >= 0; i--) {
36             if(((now >> i) & 1) != ((x >> i) & 1)) {
37                 if(!base[i]) return false;
38                 else now ^= base[i];
39             }
40         }
41         return true;
42     }
43
44     /// @brief 询问第k大的值，insert后需要先buildk

```

```

45  /// @param k
46  /// @return 第k大的值
47  int get_kth(int k) {
48      if(k >= 111 << size) return -1;
49      if(!builded) buildk();
50      int ret = 0;
51      for(int i = size - 1; ~i; i--) {
52          if(k >> i & 1) {
53              ret ^= kth[i];
54          }
55      }
56      return ret;
57  }
58  /// @brief 找到小于等于x的数的个数
59  /// @param x
60  /// @return 小于等于x的数的个数
61  int get_rank(int x) {
62      int tmpsz = size, ret = 0, now = 0;
63      for(int i = max_size - 1; i >= 0; i--) {
64          if(base[i]) tmpsz--;
65          if((x >> i) & 1) {
66              if(!((now >> i) & 1)) {
67                  ret += 111 << tmpsz;
68                  if(!base[i]) return -1;
69                  else now ^= base[i];
70              } else {
71                  if(base[i]) ret += 111 << tmpsz;
72              }
73          } else {
74              if((now >> i) & 1) {
75                  if(!base[i]) return -1;
76                  else now ^= base[i];
77              }
78          }
79      }
80      return ret;
81  }
82  private:
83  /// @brief 消成上阶梯矩阵
84  void buildk() {
85      builded = 1;
86      kth.resize(size);
87      int cnt = size;
88      for(int i = max_size - 1; ~i; i--) {
89          if(base[i]) {
90              for(int j = i - 1; ~j; j--) {
91                  if(base[i] >> j & 1) {
92                      base[i] ^= base[j];
93                  }
94              }
95          }
96      }
97      for(int i = max_size - 1; ~i; i--) {

```

```

98         if(base[i]) {
99             kth[--cnt] = base[i];
100         }
101     }
102 }
103 };

```

2.11 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。

2.11.1 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 }
```

2.11.2 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 guass(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 }

```

2.11.3 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 }

```

2.12 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 }

```

2.13 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 }

```

2.14 拆系数 FFT/MTT

- FFT, 内含 MTT, 比较灵活。

```

1  #define fp(i, a, b) for (int i = a, i##_ = (b) + 1; i < i##_; ++i)
2  #define fd(i, a, b) for (int i = a, i##_ = (b) - 1; i > i##_; --i)
3  using namespace std;
4  #define int long long
5  using ll = int64_t;
6  using db = double;
7  using Poly = vector<ll>;
8  struct cp {
9      db x, y;
10     cp(db real = 0, db imag = 0) : x(real), y(imag){};
11     cp operator+(cp b) const { return {x + b.x, y + b.y}; }
12     cp operator-(cp b) const { return {x - b.x, y - b.y}; }
13     cp operator*(cp b) const { return {x * b.x - y * b.y, x * b.y + y * b.x}; }
14 };
15 using vcp = vector<cp>;
16 const int mod = 1e9 + 7;
17 namespace FFT {
18     const db pi = acos(-1);
19     vcp Omega(int L) { // In order to reduce the accuracy error
20         vcp w(L); w[1] = 1;
21         for (int i = 2; i < L; i <= 1) {
22             auto w0 = w.begin() + i / 2, w1 = w.begin() + i;
23             cp wn(cos(pi / i), sin(pi / i));
24             for (int j = 0; j < i; j += 2)
25                 w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
26         }
27         return w;
28     }
29     auto W = Omega(1 << 23); // NOLINT
30     void DIF(cp *a, int n) {
31         cp x, y;
32         for (int k = n >> 1; k; k >>= 1)

```

```

33     for (int i = 0; i < n; i += k << 1)
34         for (int j = 0; j < k; ++j)
35             x = a[i + j], y = a[i + j + k],
36             a[i + j + k] = (x - y) * W[k + j], a[i + j] = x + y;
37     }
38     void IDIT(cp *a, int n) {
39         cp x, y;
40         for (int k = 1; k < n; k <= 1)
41             for (int i = 0; i < n; i += k << 1)
42                 for (int j = 0; j < k; ++j)
43                     x = a[i + j], y = a[i + j + k] * W[k + j],
44                     a[i + j + k] = x - y, a[i + j] = x + y;
45         const db Inv = 1. / n;
46         fp(i, 0, n - 1) a[i].x *= Inv, a[i].y *= Inv;
47         reverse(a + 1, a + n);
48     }
49 }
50 namespace Polynomial {
51     void DFT(vcp &a) { FFT::DIF(a.data(), a.size()); }
52     void IDFT(vcp &a) { FFT::IDIT(a.data(), a.size()); }
53     int norm(int n) { return 1 << (__lg(n - 1) + 1); }
54
55     // Poly mul
56     vcp &dot(vcp &a, vcp &b) { fp(i, 0, a.size() - 1) a[i] = a[i] * b[i]; return a;
57     }
58     Poly &operator+=(Poly &a, Poly b) {
59         a.resize(max(a.size(), b.size()));
60         fp(i, 0, a.size() - 1) a[i] += b[i];
61         return a;
62     };
63     Poly &operator-=(Poly &a, Poly b) {
64         a.resize(max(a.size(), b.size()));
65         fp(i, 0, a.size() - 1) DEC(a[i], b[i]);
66         return a;
67     };
68     Poly operator * (Poly A, Poly B) {
69         int n = A.size() + B.size() - 1;
70         int L = norm(n);
71         Poly res(n); vcp a(L), b(L), c(L), d(L);
72         fp(i, 0, A.size() - 1) a[i] = cp(A[i] & 0x7fff, A[i] >> 15);
73         fp(i, 0, B.size() - 1) b[i] = cp(B[i] & 0x7fff, B[i] >> 15);
74         FFT::DIF(a.data(), L), FFT::DIF(b.data(), L);
75         for (int k = 1, i = 0, j = 0; k < L; j ^= k, k <= 1)
76             for (; i < k * 2; i++, j = i ^ (k - 1)) {
77                 c[i] = cp(a[i].x + a[j].x, a[i].y - a[j].y) * b[i] * 0.5,
78                 d[i] = cp(a[i].y + a[j].y, -a[i].x + a[j].x) * b[i] * 0.5;
79             }
80         FFT::IDIT(c.data(), L), FFT::IDIT(d.data(), L);
81         for (int i = 0; i < n; i++) {
82             ll x = c[i].x + 0.5, y = c[i].y + 0.5, z = d[i].x + 0.5, w = d[i].y + 0.5;
83             x %= mod;
84             y %= mod;
85             z %= mod;

```

```

85         w %= mod;
86         res[i] = (x + ((y + z) << 15) % mod + (w << 30) % mod) % mod;
87     }
88     return res;
89 }
90 vcp operator * (vcp a, vcp b) {
91     int n = a.size() + b.size() - 1;
92     int L = norm(n);
93     a.resize(L), b.resize(L);
94     DFT(a), DFT(b);
95     dot(a, b);
96     IDFT(a);
97     return a;
98 }
99 }
100 using namespace Polynomial;

```

2.15 多项式全家桶 (Number-Theoretic-Transform)

- 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 {
15         return a * b % mod;
16     }
17     pll MUL(pll a, pll b) const {
18         return {(a.X * b.X + I2 * a.Y % mod * b.Y) % mod, (a.X * b.Y + a.Y * b.X) %
19                 mod};
20     }
21     template<class T> T ksm(T a, int b, T x) {
22         for (; b; b >>= 1, a = MUL(a, a)) if(b & 1) x = MUL(x, a);
23         return x;
24     }
25 public:

```

```

25  Cipolla(int p = 0) : mod(p) {}
26  pair<int, int> sqrt(int n) {
27      int a = rand(), x;
28      if(!(n % mod)) return {0, 0};
29      if(ksm(n, (mod - 1) >> 1, 111) == mod - 1) return {-1, -1};
30      while(ksm(I2 = ((11) a * a - n + mod) % mod, (mod - 1) >> 1, 111) == 1) a =
          rand();
31      x = (int) ksm(pll{a, 1}, (mod + 1) >> 1, {1, 0}).X;
32      if(2 * x > mod) x = mod - x;
33      return {x, mod - x};
34  }
35  #undef X
36  #undef Y
37  };
38  /*-----Modular-----*/
39  #define MUL(a, b) ((a) * (b) % mod)
40  #define ADD(a, b) (((a) += (b)) >= mod ? (a) -= mod : 0) // (a += b) %= P
41  #define DEC(a, b) (((a) -= (b)) < 0 ? (a) += mod : 0) // ((a -= b) += P) %= P
42  Poly getInv(int L) {
43      Poly inv(L);
44      inv[1] = 1;
45      fp(i, 2, L - 1) inv[i] = MUL((mod - mod / i), inv[mod % i]);
46      return inv;
47  }
48  int ksm(ll a, int b = mod - 2, ll x = 1) {
49      for(; b >= 1, a = a * a % mod) if(b & 1) x = x * a % mod;
50      return x;
51  }
52  auto inv = getInv(N); // NOLINT
53  /*-----NTT-----*/
54  namespace NTT {
55      const int g = 3;
56      Poly Omega(int L) {
57          int wn = ksm(g, mod / L);
58          Poly w(L);
59          w[L >> 1] = 1;
60          fp(i, L / 2 + 1, L - 1) w[i] = MUL(w[i - 1], wn);
61          fd(i, L / 2 - 1, 1) w[i] = w[i << 1];
62          return w;
63      }
64      auto W = Omega(1 << 20); // NOLINT
65      void DIF(int *a, int n) {
66          for(int k = n >> 1; k; k >>= 1)
67              for(int i = 0, y; i < n; i += k << 1)
68                  for(int j = 0; j < k; ++j)
69                      y = a[i + j + k], a[i + j + k] = MUL(a[i + j] - y + mod, W[k + j]), ADD
                          (a[i + j], y);
70      }
71      void IDIT(int *a, int n) {
72          for(int k = 1; k < n; k <<= 1)
73              for(int i = 0, x, y; i < n; i += k << 1)
74                  for(int j = 0; j < k; ++j)
75                      x = a[i + j], y = MUL(a[i + j + k], W[k + j]),

```

```

76         a[i + j + k] = x - y < 0 ? x - y + mod : x - y, ADD(a[i + j], y);
77     int Inv = mod - (mod - 1) / n;
78     fp(i, 0, n - 1) a[i] = MUL(a[i], Inv);
79     reverse(a + 1, a + n);
80 }
81 }
82 /*-----Polynomial 全家桶
      -----*/
83 namespace Polynomial {
84     // basic operator
85     int norm(int n) {
86         return 1 << (__lg(n - 1) + 1);
87     }
88     void norm(Poly &a) {
89         if(!a.empty()) a.resize(norm(a.size()), 0);
90         else a = {0};
91     }
92     void DFT(Poly &a) {
93         NTT::DIF(a.data(), a.size());
94     }
95     void IDFT(Poly &a) {
96         NTT::IDIT(a.data(), a.size());
97     }
98     Poly &dot(Poly &a, Poly &b) {
99         fp(i, 0, a.size() - 1) a[i] = MUL(a[i], b[i]);
100        return a;
101    }
102
103    // MUL / div int
104    Poly &operator*=(Poly &a, int b) {
105        for(auto &x : a) x = MUL(x, b);
106        return a;
107    }
108    Poly operator*(Poly a, int b) {
109        return a *= b;
110    }
111    Poly operator*(int a, Poly b) {
112        return b * a;
113    }
114    Poly &operator/=(Poly &a, int b) {
115        return a *= ksm(b);
116    }
117    Poly operator/(Poly a, int b) {
118        return a /= b;
119    }
120
121    // Poly ADD / sub
122    Poly &operator+=(Poly &a, Poly b) {
123        a.resize(max(a.size(), b.size()));
124        fp(i, 0, b.size() - 1) ADD(a[i], b[i]);
125        return a;
126    }
127    Poly operator+(Poly a, Poly b) {

```



```

128     return a += b;
129 }
130 Poly &operator+=(Poly &a, Poly b) {
131     a.resize(max(a.size(), b.size()));
132     fp(i, 0, b.size() - 1) DEC(a[i], b[i]);
133     return a;
134 }
135 Poly operator-(Poly a, Poly b) {
136     return a -= b;
137 }
138
139 // Poly MUL
140 Poly operator*(Poly a, Poly b) {
141     int n = a.size() + b.size() - 1, L = norm(n);
142     if(a.size() <= 8 || b.size() <= 8) {
143         Poly c(n);
144         fp(i, 0, a.size() - 1) fp(j, 0, b.size() - 1)
145             c[i + j] = (c[i + j] + (ll) a[i] * b[j]) % mod;
146         return c;
147     }
148     a.resize(L), b.resize(L);
149     DFT(a), DFT(b), dot(a, b), IDFT(a);
150     return a.resize(n), a;
151 }
152
153 // Poly inv
154 Poly Inv2k(Poly a) { // |a| = 2 ^ k
155     int n = a.size(), m = n >> 1;
156     if(n == 1) return {ksm(a[0])};
157     Poly b = Inv2k(Poly(a.begin(), a.begin() + m)), c = b;
158     b.resize(n), DFT(a), DFT(b), dot(a, b), IDFT(a);
159     fp(i, 0, n - 1) a[i] = i < m ? 0 : mod - a[i];
160     DFT(a), dot(a, b), IDFT(a);
161     return move(c.begin(), c.end(), a.begin()), a;
162 }
163 Poly Inv(Poly a) {
164     int n = a.size();
165     norm(a), a = Inv2k(a);
166     return a.resize(n), a;
167 }
168
169 // Poly div / mod
170 Poly operator/(Poly a, Poly b) {
171     int k = a.size() - b.size() + 1;
172     if(k < 0) return {0};
173     reverse(a.begin(), a.end());
174     reverse(b.begin(), b.end());
175     b.resize(k), a = a * Inv(b);
176     a.resize(k), reverse(a.begin(), a.end());
177     return a;
178 }
179 pair<Poly, Poly> operator%(Poly a, const Poly& b) {
180     Poly c = a / b;

```

```

181     a -= b * c, a.resize(b.size() - 1);
182     return {c, a};
183 }
184
185 // Poly calculus
186 Poly deriv(Poly a) {
187     fp(i, 1, a.size() - 1) a[i - 1] = MUL(i, a[i]);
188     return a.pop_back(), a;
189 }
190 Poly integ(Poly a) {
191     a.push_back(0);
192     fd(i, a.size() - 1, 1) a[i] = MUL(inv[i], a[i - 1]);
193     return a[0] = 0, a;
194 }
195
196 // Poly ln
197 Poly Ln(Poly a) {
198     int n = a.size();
199     a = deriv(a) * Inv(a);
200     return a.resize(n - 1), integ(a);
201 }
202
203 // Poly exp
204 Poly Exp(Poly a) {
205     int n = a.size(), k = norm(n);
206     Poly b = {1}, c, d;
207     a.resize(k);
208     for(int L = 2; L <= k; L <= 1) {
209         d = b, b.resize(L), c = Ln(b), c.resize(L);
210         fp(i, 0, L - 1) c[i] = a[i] - c[i] + (a[i] < c[i] ? mod : 0);
211         ADD(c[0], 1), DFT(b), DFT(c), dot(b, c), IDFT(b);
212         move(d.begin(), d.end(), b.begin());
213     }
214     return b.resize(n), b;
215 }
216
217 // Poly sqrt
218 Poly Sqrt(Poly a) {
219     int n = a.size(), k = norm(n);
220     a.resize(k);
221     Poly b = {(new Cipolla(mod))->sqrt(a[0]).first, 0}, c;
222     for(int L = 2; L <= k; L <= 1) {
223         b.resize(L), c = Poly(a.begin(), a.begin() + L) * Inv2k(b);
224         fp(i, L / 2, L - 1) b[i] = MUL(c[i], (mod + 1) / 2);
225     }
226     return b.resize(n), b;
227 }
228
229 // Poly pow
230 Poly Pow(Poly &a, int b) {
231     return Exp(Ln(a) * b); // a[0] = 1
232 }
233 Poly Pow(Poly a, int b1, int b2) { // b1 = b % mod, b2 = b % phi(mod) and b >= n

```

```

234     iff a[0] > 0
235     int n = a.size(), d = 0, k;
236     while(d < n && !a[d]) ++d;
237     if((ll) d * b1 >= n) return Poly(n);
238     a.erase(a.begin(), a.begin() + d);
239     k = ksm(a[0]), norm(a *= k);
240     a = Pow(a, b1) * ksm(k, mod - 1 - b2);
241     a.resize(n), d *= b1;
242     fd(i, n - 1, 0) a[i] = i >= d ? a[i - d] : 0;
243     return a;
244 }
245 // a0, a1, ..., a_{k-1}, f1, f2, ..., fk
246 // a 是值, f 是递推系数
247 // 常数齐次线性递推
248 int LinearRecursion(vector<int> &a, vector<int> &f, int n) {
249     int k = a.size();
250     Poly t(k + 1);
251     for(int i = 0; i < k; i++) {
252         t[i] = (mod - f[k - i]) % mod;
253     }
254     t[k] = 1;
255     Poly g{0, 1}, ret{1};
256     while(n) {
257         if(n & 1) ret = (ret * g % t).second;
258         g = (g * g % t).second;
259         n >>= 1;
260     }
261     int ans = 0;
262     for(int i = 0; i < k; i++) {
263         ADD(ans, ret[i] * a[i] % mod);
264     }
265     return ans;
266 }
267 Poly mult(Poly a, Poly b) {
268     int n = a.size(), m = b.size();
269     reverse(b.begin(), b.end());
270     a = a * b;
271     for(int i = 0; i < n; i++) {
272         a[i] = a[i + m - 1];
273     }
274     a.resize(n);
275     return a;
276 }
277 // x是点, a是多项式
278 vector<int> eval(Poly a, vector<int> x) {
279     int m = max(x.size(), a.size());
280     vector<int> ans(x.size());
281     x.resize(m + 1);
282     vector<Poly> divd(4 * m);
283     function<void(int, int, int)> divide_mul = [&](int l, int r, int id) -> void
284     {
285         if(l == r) {

```

```

285         divd[id] = Poly{1, (mod - x[1]) % mod};
286         return;
287     }
288     int mid = l + r >> 1;
289     divide_mul(l, mid, id << 1), divide_mul(mid + 1, r, id << 1 | 1);
290     divd[id] = divd[id << 1 | 1] * divd[id << 1];
291 };
292 function<void(int, int, int, Poly)> getans = [&](int l, int r, int id, Poly
    now) -> void {
293     if(l == r) {
294         if(l < ans.size()) {
295             ans[l] = now[0];
296         }
297         return;
298     }
299     int mid = l + r >> 1;
300     now.resize(r - l + 2);
301     getans(l, mid, id << 1, mulT(now, divd[id << 1 | 1]));
302     getans(mid + 1, r, id << 1 | 1, mulT(now, divd[id << 1]));
303 };
304 divide_mul(0, m - 1, 1);
305 a.resize(m);
306 getans(0, m - 1, 1, mulT(a, Inv(divd[1])));
307 return ans;
308 }
309 }
310 using namespace Polynomial;

```

3 Math Formula

3.1 多项式牛顿迭代

3.2 牛顿恒等式

设 $E(x) = 1 + e_1x + e_2x^2 + \dots$

$$\begin{aligned}\prod_{i=1}^n (1 + a_i x) &= E(x) \\ (\ln \prod_{i=1}^n (1 + a_i x))' &= E'(x)/E(x) \\ \sum_{i=1}^n \frac{a_i}{1 + a_i x} &= E'(x)/E(x) \\ E'(x)/E(x) &= \sum_{i=1}^n \sum_{j \geq 0} (-1)^j a_i^{j+1} x^j \\ &= \sum_{j \geq 0} (-1)^j \left(\sum_{i=1}^n a_i^{j+1} \right) x^j \\ &= \sum_{j \geq 0} (-1)^j p_{j+1} x^j\end{aligned}$$

结论:

$$\begin{aligned}k e_k &= \sum_{i=1}^k (-1)^{i-1} e_{k-i} p_i \\ 0 &= \sum_{i=k-n}^k (-1)^{i-1} e_{k-i} p_i\end{aligned}$$

3.3 生成函数/形式幂级数

3.4 数论公式

3.4.1 莫比乌斯反演

一般反演:

$$\begin{aligned}f(n) &= \sum_{d|n} g(d) \iff g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) \\ f(n) &= \sum_{\substack{n|d \\ d \leq N}} g(d) \iff g(n) = \sum_{\substack{n|d \\ d \leq N}} \mu\left(\frac{d}{n}\right) f(d)\end{aligned}$$

gcd 反演结论:

$$[\gcd(i, j) = 1] = \sum_{d|\gcd(i, j)} \mu(d)$$

3.4.2 杜教筛

$$g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i)S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

3.4.3 Min_25 筛/Ex-Eratosthenes-Sieve

第一步：筛出所有质数部分（此处的 $g(n)$ 是 $f(n), n \in \mathbb{P}$ 的通项公式）

$$G_k(n) := \sum_{i=1}^n [p_k < \text{lpf}(i) \vee \text{isprime}(i)] g(i)$$

$$G_k(n) = G_{k-1}(n) - [p_k^2 \leq n] g(p_k) (G_{k-1}(n/p_k) - G_{k-1}(p_{k-1}))$$

第二部：筛出质数和合数部分，我们令 $F_{\text{prime}}(n) = \sum_{\substack{2 \leq p \leq n \\ p \in \text{prime}}} f(p)$

$$\begin{aligned} F_k(n) &= \sum_{i=2}^n [p_k \leq \text{lpf}(i)] f(i) \\ &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + \sum_{\substack{k \leq i \\ p_i \leq n}} f(p_i) \\ &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \text{常用!!} \\ &= \sum_{\substack{k \leq i \\ p_i^2 \leq n}} \sum_{\substack{c \geq 1 \\ p_i^{c+1} \leq n}} (f(p_i^c) F_{i+1}(n/p_i^c) + f(p_i^{c+1})) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \end{aligned}$$

```

1 // sieve first!!
2
3 namespace min25 {
4
5 ll n, sqr, w[N];
6 int c;
7 ll g[N];
8
9 inline int id(ll x) {
10     return x ? x <= sqr ? c - x + 1 : n / x : 0;
11 }
12
13 void cal_g(ll n_) { // 这里的 g 只能是一个没有系数的单项式!! 不是极值函数f
14     n = n_; sqr = sqrt(n_); c = 0;
15     for (ll l = 1, r; l <= n; l = r + 1) {
16         ll v = w[++c] = n / l; r = n / v;
17         g[c] = (v - 1) % mod;
18     }
19     for (int i = 1; i <= prinum; i++) {
20         int p = pri[i];
21         if (1ll * p * p > n) break;
22         for (int j = 1; 1ll * p * p <= w[j]; ++j)
23             g[j] -= g[id(w[j] / p)] - g[id(p - 1)];
24     }
25 }
26 int cal_s(int n, ll x, int y) { // 用 g 的单项式乘以系数加起来。binom那个地方是填f(p^e)
27     if(x <= pri[y]) return 0;
28     int ans = (g[id(x)] - g[id(pri[y])]) + mod * n % mod;
29
30     for(int i = y + 1; i <= prinum && pri[i] * pri[i] <= x; i++) {
31         ll P = pri[i];
32         for(int e = 1; P <= x; e++, P *= pri[i]) {

```

```

33         ans = (ans + (1ll) binom(n + e - 1, n - 1) * (cal_s(n, x / P, i) + (e != 1)
34             ) % mod) % mod;
35     }
36     return ans;
37 }
38
39 }

```

3.5 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	$o.g.f = \prod_{i \geq 1} \left(\frac{1}{1-x^i} \right)^{f_i} = \exp \left(\sum_{j \geq 1} \frac{1}{j} F(x^j) \right)$

3.6 第二类斯特林数

性质：

- $\{n\}_k = \{n-1\}_k + k\{n-1\}_{k-1}$ 边界条件 $\{x \neq 1\}_0 = 0, \{0\}_0 = 1$
- $\{n\}_k = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^n$ (用来求第二类斯特林数 · 行，考虑容斥)
- $k^n = \sum_{i=0}^k \{n\}_i k^i$ (性质 2 的反演)
- 重要公式： $x^n = \sum_{k=0}^n \{n\}_k x^{\underline{k}}$ (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)

3.7 第一类斯特林数

性质:

1. $\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}$ 边界条件 $\begin{bmatrix} x \neq 1 \\ 0 \end{bmatrix} = 0, \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$
2. 重要公式: $x^{\overline{n}} = \sum_{k=0}^n \begin{bmatrix} n \\ k \end{bmatrix} x^k$ (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: $S_1(n, k) = (-1)^{n+k} \begin{bmatrix} n \\ k \end{bmatrix}$

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

3.8 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)

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

3.10 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 $R_G = \{(x, y) \mid x, y \in X \wedge (\exists g \in G. y = g(x))\}$, $xR_G y$ 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:

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

边置换群:

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

面置换群:

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

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

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

4 Misc

4.1 德州扑克

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  struct card{
4      char suit;
5      int rank;
6      card(){
7      }
8      bool operator <(card C){
9          return rank < C.rank || rank == C.rank && suit < C.suit;
10     }
11 };
12 istream& operator >>(istream& is, card& C){
13     string S;
14     is >> S;
15     if (S[0] == 'A'){
16         C.rank = 14;
17     } else if (S[0] == 'K'){
18         C.rank = 13;
19     } else if (S[0] == 'Q'){
20         C.rank = 12;
21     } else if (S[0] == 'J'){
22         C.rank = 11;
23     } else if (S[0] == 'T'){
24         C.rank = 10;
25     } else {
26         C.rank = S[0] - '0';
27     }
28     C.suit = S[1];
29     return is;
30 }
31 vector<int> hand(vector<card> C){
32     sort(C.begin(), C.end());
33     set<char> suits;
34     for (int i = 0; i < 5; i++){
35         suits.insert(C[i].suit);
36     }
37     if (suits.size() == 1 && C[4].rank - C[0].rank == 4){
38         if (C[4].rank == 14){
39             return vector<int>{9};
40         } else {
41             return vector<int>{8, C[4].rank};
42         }
43     }
44     if (suits.size() == 1 && C[3].rank == 5 && C[4].rank == 14){
45         return vector<int>{8, 5};
46     }
47     if (C[0].rank == C[3].rank){
48         return vector<int>{7, C[0].rank, C[4].rank};
49     }
```

```

50  if (C[1].rank == C[4].rank){
51      return vector<int>{7, C[1].rank, C[0].rank};
52  }
53  if (C[0].rank == C[2].rank && C[3].rank == C[4].rank){
54      return vector<int>{6, C[0].rank, C[3].rank};
55  }
56  if (C[2].rank == C[4].rank && C[0].rank == C[1].rank){
57      return vector<int>{6, C[2].rank, C[0].rank};
58  }
59  if (suits.size() == 1){
60      return vector<int>{5, C[4].rank, C[3].rank, C[2].rank, C[1].rank, C[0].rank};
61  }
62  if (C[1].rank - C[0].rank == 1 && C[2].rank - C[1].rank == 1 && C[3].rank - C[2].
    rank == 1 && C[4].rank - C[3].rank == 1){
63      return vector<int>{4, C[4].rank};
64  }
65  if (C[0].rank == 2 && C[1].rank == 3 && C[2].rank == 4 && C[3].rank == 5 && C[4].
    rank == 14){
66      return vector<int>{4, 5};
67  }
68  if (C[0].rank == C[2].rank){
69      return vector<int>{3, C[0].rank, C[4].rank, C[3].rank};
70  }
71  if (C[1].rank == C[3].rank){
72      return vector<int>{3, C[1].rank, C[4].rank, C[0].rank};
73  }
74  if (C[2].rank == C[4].rank){
75      return vector<int>{3, C[2].rank, C[1].rank, C[0].rank};
76  }
77  if (C[0].rank == C[1].rank && C[2].rank == C[3].rank){
78      return vector<int>{2, C[2].rank, C[0].rank, C[4].rank};
79  }
80  if (C[0].rank == C[1].rank && C[3].rank == C[4].rank){
81      return vector<int>{2, C[3].rank, C[0].rank, C[2].rank};
82  }
83  if (C[1].rank == C[2].rank && C[3].rank == C[4].rank){
84      return vector<int>{2, C[3].rank, C[1].rank, C[0].rank};
85  }
86  if (C[0].rank == C[1].rank){
87      return vector<int>{1, C[0].rank, C[4].rank, C[3].rank, C[2].rank};
88  }
89  if (C[1].rank == C[2].rank){
90      return vector<int>{1, C[1].rank, C[4].rank, C[3].rank, C[0].rank};
91  }
92  if (C[2].rank == C[3].rank){
93      return vector<int>{1, C[2].rank, C[4].rank, C[1].rank, C[0].rank};
94  }
95  if (C[3].rank == C[4].rank){
96      return vector<int>{1, C[3].rank, C[2].rank, C[1].rank, C[0].rank};
97  }
98  return vector<int>{0, C[4].rank, C[3].rank, C[2].rank, C[1].rank, C[0].rank};
99  }
100 int dfs(vector<vector<int>> &alice, vector<vector<int>> &bob, int a, int b, int p){

```

```

101  if (p == 6){
102      if (alice[a] > bob[b]){
103          return 1;
104      } else if (alice[a] < bob[b]){
105          return -1;
106      } else {
107          return 0;
108      }
109  } else {
110      int mx = -1;
111      for (int i = 0; i < 6; i++){
112          if ((a >> i & 1) == 0 && (b >> i & 1) == 0){
113              if (p % 2 == 0){
114                  mx = max(mx, -dfs(alice, bob, a | (1 << i), b, p + 1));
115              } else {
116                  mx = max(mx, -dfs(alice, bob, a, b | (1 << i), p + 1));
117              }
118          }
119      }
120      return mx;
121  }
122 }
123 int main(){
124     int T;
125     cin >> T;
126     for (int i = 0; i < T; i++){
127         vector<card> a(2);
128         cin >> a[0] >> a[1];
129         vector<card> b(2);
130         cin >> b[0] >> b[1];
131         vector<card> c(6);
132         cin >> c[0] >> c[1] >> c[2] >> c[3] >> c[4] >> c[5];
133         vector<vector<int>> alice(1 << 6), bob(1 << 6);
134         for (int j = 0; j < (1 << 6); j++){
135             if (__builtin_popcount(j) == 3){
136                 vector<card> ha = {a[0], a[1]};
137                 vector<card> hb = {b[0], b[1]};
138                 for (int k = 0; k < 6; k++){
139                     if ((j >> k & 1) == 1){
140                         ha.push_back(c[k]);
141                         hb.push_back(c[k]);
142                     }
143                 }
144                 alice[j] = hand(ha);
145                 bob[j] = hand(hb);
146             }
147         }
148         int ans = dfs(alice, bob, 0, 0, 0);
149         if (ans == 1){
150             cout << "Alice" << endl;
151         } else if (ans == -1){
152             cout << "Bob" << endl;
153         } else {

```

```

154     cout << "Draw" << endl;
155 }
156 }
157 }

```

4.2 Expression Parser

```

1  // 运算符
2  function<int(int, int, char)> ca = [&] (int a, int b, char sg) {
3      if(sg == '+') return (a + b) % 10000;
4      if(sg == '-') return a - b;
5      if(sg == '*') return (a * b) % 10000;
6      return 0;
7  };
8  // 设定优先级
9  map<char, int> prio;
10 prio['+'] = prio['-'] = 1;
11 prio['*'] = 2;
12 // calculate 传入前两个是操作数, 第三个是操作符 priority['('] = 0;
13 int parse_expression(string s, map<char, int> priority, function<int(int, int, char)
    > calculate) {
14     int pt = 0;
15     string tmp;
16     for(int i = 0; i < s.length(); i++) if(s[i] != ' ') tmp += s[i];
17     s = tmp;
18     vector<int> stk;
19     vector<char> sgn;
20     // 读入取模在这里改
21     auto readint = [&]() {
22         int ret = 0, fl = 1;
23         if(s[pt] == '-') fl = -1, pt++;
24         while(pt < s.length() && isdigit(s[pt])) {
25             ret = ret * 10 + s[pt] - '0';
26             pt++;
27         }
28         return ret * fl;
29     };
30     auto calc_single = [&] () {
31         char sg = sgn.back();
32         sgn.pop_back();
33         int v = stk.back();
34         stk.pop_back();
35         int u = stk.back();
36         stk.pop_back();
37         stk.push_back(calculate(u, v, sg));
38     };
39     int mode = 0;
40     // 0 is number, 1 is sign
41     while(pt < s.length()) {
42         if(mode == 0) {
43             if(isdigit(s[pt]) || s[pt] == '-') {
44                 int num = readint();

```

```

45         stk.push_back(num);
46         mode = 1;
47     } else if(s[pt] == '(') {
48         sgn.push_back(s[pt]);
49         pt++;
50     }
51 } else {
52     if(s[pt] != ')') {
53         while(sgn.size() && priority[sgn.back()] >= priority[s[pt]]) {
54             calc_single();
55         }
56         sgn.push_back(s[pt]);
57         mode = 0;
58     } else if(s[pt] == ')') {
59         while(sgn.back() != '(') {
60             calc_single();
61         }
62         sgn.pop_back();
63     }
64     pt++;
65 }
66 }
67 while(sgn.size()) {
68     calc_single();
69 }
70 assert(stk.size() == 1);
71 return stk[0];
72 };

```

4.3 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  template<typename T> void debug(T a) { cout << a; }
8
9  template<typename T, typename...args> void debug(T a, args...b) {
10     cout << a << ", ";
11     debug(b...);
12 }
13
14 template<typename T>
15 ostream& operator << (ostream &os, const vector<T> &a) {
16     os << "[";
17     int f = 0;
18     for(auto &x : a) os << (f++ ? ", " : "") << x;
19     os << "];";
20     return os;
21 }

```

```
21
22 template<typename T>
23 ostream& operator << (ostream &os, const set<T> &a) {
24     os << "{";
25     int f = 0;
26     for(auto &x : a) os << (f++ ? ", " : "") << x;
27     os << "}";
28     return os;
29 }
30
31 template<typename T>
32 ostream& operator << (ostream &os, const multiset<T> &a) {
33     os << "{";
34     int f = 0;
35     for(auto &x : a) os << (f++ ? ", " : "") << x;
36     os << "}";
37     return os;
38 }
39
40 template<typename A, typename B>
41 ostream& operator << (ostream &os, const map<A, B> &a) {
42     os << "{";
43     int f = 0;
44     for(auto &x : a) os << (f++ ? ", " : "") << x;
45     os << "}";
46     return os;
47 }
48
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 }
```