cycleke（菜鸡）的 XCPC 模板

cycleke

May 19, 2021

Contents

1 数学

1.1 素数

素数的数目有近似 $\pi(x) \sim \frac{x}{\ln(x)}$，判定如下:

```cpp
inline ll mmul(const ll &a, const ll &b, const ll &mod) {
  ll k = (ll)((1.0L * a * b) / (1.0L * mod)), t = a * b - k * mod;
  for (t -= mod; t < 0; t += mod) {}
  return t;
}
inline ll mpow(ll a, ll b, const ll &mod) {
  ll res = 1;
  for (; b; b >>= 1, a = mmul(a, a, mod)) (b & 1) && (res = mmul(res, a, mod));
  return res;
}

inline bool check(const ll &x, const ll &p) {
  if (!(x % p) || mpow(p % x, x - 1, x) ^ 1) return false;
  for (ll k = x - 1, t; ~k & 1;) {
    if (((t = mpow(p % x, k >>= 1, x)) ^ 1) && (t ^ (x - 1))) return false;
    if (!(t ^ (x - 1))) return true;
  }
  return true;
}

inline bool Miller_Rabin(const ll &x) {
  if (x < 2) return false;
  static const int p[12] = {2, 3, 5, 7, 11, 13, 17, 19, 61, 2333, 4567, 24251};
  for (int i = 0; i < 12; ++i) {
    if (!(x ^ p[i])) return true;
    if (!check(x, p[i])) return false;
  }
  return true;
}
```

1.2 Pollard Rho

```cpp
mt19937_64 rnd(chrono::high_resolution_clock::now().time_since_epoch().count());
inline ll rand64(ll x) { return rnd() % x + 1; }

inline ll Pollard_rho(const ll &x, const int &y) {
  ll v0 = rand64(x), v = v0, d, s = 1;
  for (int t = 0, k = 1;;) {
    v = (mmul(v, v, x) + y) % x, s = mmul(s, abs(v - v0), x);
    if (!(v ^ v0) || !s) return x;
    if (++t == k) {
      if ((d = __gcd(s, x)) ^ 1) return d;
      v0 = v, k <<= 1;
    }
  }
}

vector<ll> factor;
void findfac(ll n) {
  if (Miller_Rabin(n)) {
    factor.push_back(n);
    return;
  }
  ll p = n;
  while (p >= n) p = Pollard_rho(p, rand64(n));
  findfac(p), findfac(n / p);
}
```

1.3 欧拉函数

欧拉函数的性质:

- 欧拉函数是积性函数;

- $n = \sum_{d|n} \varphi(d)$;

- 若 $n = p^k$, 其中 p 是质数, 那么 $\varphi(n) = p^k - p^{k-1}$;

- 若 $\gcd(a, m) = 1$, 则 $a^{\varphi(m)} \equiv 1 \pmod{m}$;

拓展欧拉定理: $a^b \equiv \begin{cases} a^{b \bmod \varphi(p)} & \gcd(a,,p) = 1 \\ a^b & \gcd(a,,p) \neq 1, b < \varphi(p) \quad (\pmod{p}) \\ a^{b \bmod \varphi(p) + \varphi(p)} & \gcd(a,,p) \neq 1,, b \geq \varphi(p) \end{cases}$

```cpp
int euler_phi(int n) {
  int ans = n;
  for (int i = 2; i * i <= n; i++)
    if (n % i == 0) {
      ans = ans / i * (i - 1);
      while (n % i == 0) n /= i;
    }
  if (n > 1) ans = ans / n * (n - 1);
  return ans;
}
```

1.4 线性筛

lpf 为最小质因子的标号; mu 为莫比乌斯函数; phi 为欧拉函数; e 为质因子最高次幂, d 为因数个数; f 为因数和, g 为最小质因子的幂和, 即 $p + p^1 + p^2 + \cdots + p^k$ 。理论上积性函数都可以线性筛。

```cpp
const int MAXN = 1e7 + 5;
int prime[MAXN / 15], prime_cnt;
int lpf[MAXN], e[MAXN], d[MAXN], mu[MAXN], phi[MAXN];
void sieve() {
  prime[lpf[1] = 0] = 1, e[1] = 0, d[1] = 1, mu[1] = 1, phi[1] = 1;
  for (int i = 2; i < MAXN; ++i) {
    if (!lpf[i]) {
      prime[lpf[i] = ++prime_cnt] = i;
      mu[i] = -1, phi[i] = i - 1;
      e[i] = 1, d[i] = 2;
      g[i] = f[i] = i + 1;
    }
    for (int j = 1, x; j <= lpf[i] && (x = i * prime[j]) < MAXN; ++j) {
      lpf[x] = j;
      if (j < lpf[i]) {
        mu[x] = -mu[i], phi[x] = phi[i] * (prime[j] - 1);
        e[x] = 1, d[x] = d[i] * 2;
        g[x] = 1 + prime[j], f[x] = f[i] * f[prime[j]];
      } else { // i % prime[j] == 0
        mu[x] = 0, phi[x] = phi[i] * prime[j];
        e[x] = e[i] + 1, d[x] = d[i] / e[x] * (e[x] + 1);
        g[x] = g[i] * prime[j] + 1, f[x] = f[i] / g[i] * g[x];
      }
    }
  }
}
```

1.5 拓展欧几里得算法

```
1 int exgcd(int a, int b, int &x, int &y) {
2   if (b == 0) return x = 1, y = 0, a;
3   int g = exgcd(b, a % b, y, x);
4   y -= a / b * x;
5   return g;
6 }
```

1.6 莫比乌斯反演

常见积性函数（$f(ab) = f(a)f(b), (a, b) = 1$）：

- 单位函数：$\epsilon(n) = [n = 1]$（完全积性）

- 恒等函数：$\mathrm{id}_k(n) = n^k$，$\mathrm{id}_1(n)$ 通常简记作 $\mathrm{id}(n)$（完全积性）。

- 常数函数：$1(n) = 1$（完全积性）

- 除数函数：$\sigma_k(n) = \sum_{d|n} d^k$，$\sigma_0(n)$ 通常简记作 $\mathrm{d}(n)$ 或 $\tau(n)$，$\sigma_1(n)$ 通常简记作 $\sigma(n)$。

- 欧拉函数：$\varphi(n) = \sum_{i=1}^{n} [\gcd(i, n) = 1]$

- 莫比乌斯函数：$\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 \mid n \\ (-1)^{\omega(n)} & otherwise \end{cases}$，其中 $\omega(n)$ 表示 n 的本质不同质因子个数，它是一个加性函数（$\omega(ab) = \omega(a) + \omega(b)$）。

Dirichlet 卷积满足交换率、结合率和分配率，常见 Dirichlet 卷积：

- $\varepsilon = \mu * 1$
- $d = 1 * 1$
- $\sigma = \mathrm{id} * 1$

- $\varphi = \mathrm{id} * \mu$
- $\mathrm{id} = \varphi * 1$
- $\mathrm{id}_{k+1} = (\mathrm{id}_k \cdot \varphi) * \mathrm{id}_k$

- $f \cdot d = f * f$（f 为完全积性）

莫比乌斯反演：$f = g * 1 \iff g = \mu * f$。
拓展：对于数论函数 f, g 和完全积性函数 t 且 $t(1) = 1$：

$$f(n) = \sum_{i=1}^{n} t(i) g\left(\left\lfloor \frac{n}{i} \right\rfloor\right) \iff g(n) = \sum_{i=1}^{n} \mu(i) t(i) f\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

常用结论：

- $\sum_{i=x}^{n} \sum_{j=y}^{m} [\gcd(i, j) = k] = \sum_{d=1} \mu(d) \lfloor \frac{n}{kd} \rfloor \lfloor \frac{m}{kd} \rfloor$

- $d(ij) = \sum_{x|i} \sum_{y|j} [\gcd(x, y) = 1] = \sum_{p|i,p|j} \mu(p) d\left(\frac{i}{p}\right) d\left(\frac{j}{p}\right)$

1.7 杜教筛

设 $S(n) = \sum_{i=1}^{n} f(i)$,则 $\sum_{i=1}^{n} (f*g)(i) = \sum_{i=1}^{n} g(i) S(\lfloor \frac{n}{i} \rfloor) \Rightarrow g(1)S(n) = \sum_{i=1}^{n} (f*g)(i) - \sum_{i=2}^{n} g(i) S(\lfloor \frac{n}{i} \rfloor)$。直接分块复杂度 $O(n^{\frac{3}{4}})$，预处理出前 $O(n^{\frac{2}{3}})$ 项复杂度为 $O(n^{\frac{2}{3}})$。常用结论：

- 莫比乌斯函数前缀和：$S(n) = \sum_{i=1}^{n} \epsilon(i) - \sum_{i=2}^{n} S(\lfloor \frac{n}{i} \rfloor)$

- 欧拉函数前缀和：$S(n) = \sum_{i=1}^{n} id(i) - \sum_{i=2}^{n} S(\lfloor \frac{n}{i} \rfloor)$

```
1  map<int, int> mp_mu;
2
3  int S_mu(int n) {
4    if (n < MAXN) return sum_mu[n];
5    if (mp_mu[n]) return mp_mu[n];
6    int ret = 1;
7    for (int i = 2, j; i <= n; i = j + 1) {
8      j = n / (n / i);
9      ret -= S_mu(n / i) * (j - i + 1);
10   }
11   return mp_mu[n] = ret;
12 }
13
14 // 使用莫比乌斯反演
15 ll S_phi(int n) {
16   ll res = 0;
17   for (int i = 1, j; i <= n; i = j + 1) {
18     j = n / (n / i);
19     res += 1LL * (S_mu(j) - S_mu(i - 1)) * (n / i) * (n / i);
20   }
21   return (res - 1) / 2 + 1;
22 }
```

1.8 Min_25 筛

要求：积性函数 $f(p)$ 是一个关于 p 的项数较少的多项式或可以快速求值；$f(p^c)$ 可以快速求值。时间复杂度：$O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$。部分符号和结论：

- $F_{\text{prime}}(n) = \sum_{2 \le p \le n} f(p)$

- $F_k(n) = \sum_{i=2}^{n} [p_k \le \text{lpf}(i)] f(i)$，答案为 $F_1(n) + f(1)$

- $F_k(n) = \sum_{\substack{k \le i \\ p_i^2 \le n}} \sum_{\substack{c \ge 1 \\ p_i^{c+1} \le n}} (f(p_i^c) F_{i+1}(n/p_i^c) + f(p_i^{c+1})) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1})$

- 若 $f(p) = \sum p^{s_i}$，设 $g(p) = p^s, G_k(n) = \sum_{i=1}^{n} [p_k < \text{lpf}(i) \vee \text{isprime}(i)] g(i)$

- $G_k(n) = G_{k-1}(n) - [p_k^2 \le n] g(p_k)(G_{k-1}(n/p_k) - G_{k-1}(p_{k-1})), G_0 = \sum_{i=2}^{n} g(i)$

LOJ 例题：给定 $f(n)$：$f(n) = \begin{cases} 1 & n = 1 \\ p \oplus c & n = p^c \\ f(a)f(b) & n = ab \wedge a \perp b \end{cases}$，求 $f(n)$ 的和。因 $f(p) = p - 1 + 2[p = 2]$，可以按照筛 φ 的方法来处理。

```
1  constexpr int MOD = 1e9 + 7;
2  constexpr int INV2 = MOD / 2 + 1;
3
4  template <typename X, typename Y> void inc(X &x, const Y &y) { x += y, (x >= MOD) && (x -= MOD); }
5  template <typename X, typename Y> void dec(X &x, const Y &y) { x -= y, (x < 0) && (x += MOD); }
6  template <typename X, typename Y> int sum(X x, Y y) { return x + y < MOD ? x + y : x + y - MOD; }
7  template <typename X, typename Y> int sub(X x, Y y) { return x < y ? x + MOD - y : x - y;}
8
9  constexpr int MAX_SIZE = 2e5 + 3;
10 int prime[MAX_SIZE / 10], lpf[MAX_SIZE], spri[MAX_SIZE], prime_cnt;
11
12 void sieve(int n) {
13   for (int i = 2; i <= n; ++i) {
14     if (lpf[i] == 0) {
15       prime[lpf[i] = ++prime_cnt] = i;
16       spri[prime_cnt] = sum(spri[prime_cnt - 1], i);
17     }
18     for (int j = 1, x; j <= lpf[i] && (x = i * prime[j]) <= n; ++j) lpf[x] = j;
```

```
19      }
20  }
21
22  ll g_n, lis[MAX_SIZE];
23  int G[MAX_SIZE][2], Fprime[MAX_SIZE], cnt;
24  int lim, le[MAX_SIZE], ge[MAX_SIZE];
25  #define idx(x) (x <= lim ? le[x] : ge[g_n / x])
26
27  void init(ll n) {
28      for (ll i = 1, j, x; i <= n; i = n / j + 1) {
29          j = n / i, x = j % MOD;
30          lis[++cnt] = j, idx(j) = cnt;
31          G[cnt][0] = sub(x, 1);
32          G[cnt][1] = (x + 2ll) * (x - 1ll) % MOD * INV2 % MOD;
33      }
34  }
35  void calcFPrime() {
36      for (int k = 1; k <= prime_cnt; ++k) {
37          const int p = prime[k];
38          const ll sqrp = 1ll * p * p;
39          for (int i = 1; lis[i] >= sqrp; ++i) {
40              const ll x = lis[i] / p;
41              const int id = idx(x);
42              dec(G[i][0], sub(G[id][0], k - 1));
43              dec(G[i][1], 1ll * p * sub(G[id][1], spri[k - 1]) % MOD);
44          }
45      }
46      // f(p) = g_1(p) - g_0(p)
47      for (int i = 1; i <= cnt; ++i) Fprime[i] = sub(G[i][1], G[i][0]);
48  }
49
50  int f_p(int p, int c) { return p ^ c; }
51  int F(int k, ll n) {
52      if (n < prime[k] || n <= 1) return 0;
53      const int id = idx(n);
54      int res = sub(Fprime[id], sub(spri[k - 1], k - 1));
55      // F_prime(p_{k - 1}) = spri[k - 1] - (k - 1)
56      if (k == 1) res += 2; // 特殊处理 f(2)
57      for (int i = k; i <= prime_cnt && 1ll * prime[i] * prime[i] <= n; ++i) {
58          ll pw = prime[i], pw2 = pw * pw;
59          for (int c = 1; pw2 <= n; ++c, pw = pw2, pw2 *= prime[i])
60              inc(res, (1ll * f_p(prime[i], c) * F(i + 1, n / pw) + f_p(prime[i], c + 1)) % MOD);
61      }
62      return res;
63  }
```

新版 min25 筛，复杂度 $O\left(n^{\frac{2}{3}}\right)$。

```
1   /*****************************
2   f()函数中(31-37行) 填函数在质数幂次处的表达式
3   pow_sum()函数中(38-43行) 填幂和函数（如果需要更高次的话可以在这里添加）
4   202-205行按要求填写
5   f_p[][0/1/2/3/...]分别代表质数个数/质数和/质数平方和/质数三次方和/...根据自己需要添加
6   例：如果该函数在质数处表达式为f(p) =
7   p^2+3*p+1，则表明需要质数个数/质数和/质数平方和，即f_p[][0],f_p[][1],f_p[][2]
8   *****************************/
9
10  inline ll f(ll p, int e) { // return f(p^e)
11      if (p == 1 || e == 0) return 1;
12      ll res = mpow(p, e);
13      return res * res + 3 * res + 1;
14  }
15  ll pow_sum(ll n, int k) { return sum(i^k),i from 1 to n.
16      if (k == 0) return n;
17      if (k == 1) return n * (n + 1) / 2;
18      if (k == 2) return n * (n + 1) * (2 * n + 1) / 6;
19  }
20  ll n, f_p[maxn][3]; // F_prime(id(n/i))
```

```
21  int n_2, n_3, n_6;   // sqrt(n), sqrt3(n), sqrt6(n);
22  ll val_id[maxn];      // give the id, return the id-th number like 'n/i' ,(val_id[1] = 1)
23  int val_id_num;       // how many numbers like 'n/i'
24  int val_id_num_3;     // how many numbers like 'n/i' below n/n_3;
25  int p[200000 + 100];
26  bool isp[maxn];
27  int p_sz_2, p_sz_3, p_sz_6; // pi(n_2), pi(n_3), pi(n_6)
28  void init() {
29    n_2 = (int)sqrt(n);
30    n_3 = (int)pow(n, 1.0 / 3.0);
31    n_6 = (int)pow(n, 1.0 / 6.0);
32    val_id_num = 0;
33    for (ll i = 1; i <= n;) {
34      val_id[++val_id_num] = i;
35      if (i < n) i = n / (n / (i + 1));
36    }
37    memset(isp, 1, sizeof isp);
38    isp[1] = 0;
39    for (int i = 2; i <= n_2; i++) {
40      if (isp[i]) {
41        p[++p_sz_2] = i;
42        if (i <= n_3) p_sz_3++;
43        if (i <= n_6) p_sz_6++;
44      }
45      for (int j = 1; j <= p_sz_2 && p[j] * i <= n_2; j++) {
46        isp[i * p[j]] = 0;
47        if (i % p[j] == 0) break;
48      }
49    }
50  }
51  inline int get_id(ll k) { // give a number like 'n/i', return the id of it
52    return k > n_2 ? val_id_num - n / k + 1 : k;
53  }
54  ll c[maxn];
55  void add(int x, ll d) { for (; x < maxn; x += x & -x) c[x] += d; }
56  ll sum(int x) {
57    ll ans = 0;
58    for (; x; x &= x - 1) ans += c[x];
59    return ans;
60  }
61
62  struct node {
63    int k_max;
64    ll val, f_val;
65  };
66  void update_bfs(int k, int type) {
67    queue<node> q;
68    while (!q.empty()) q.pop();
69    int e = 1;
70    for (ll i = p[k]; i < n / n_3; i *= p[k], ++e)
71      q.emplace(k, i, type == -1 ? f(p[k], e) : mpow(i, type));
72    while (!q.empty()) {
73      node hd = q.front(); q.pop();
74      if ((hd.val != p[hd.k_max] && type >= 0) || type == -1) {
75        ll w = n / hd.val;
76        w = n / w;
77        if (type == -1) {
78          add(get_id(w), hd.f_val);
79          add(val_id_num + 1, -1ll * hd.f_val);
80        } else {
81          add(get_id(w), -1ll * hd.f_val);
82          add(val_id_num + 1, hd.f_val);
83        }
84      }
85      for (int i = hd.k_max + 1; hd.val * p[i] < n / n_3 && i <= p_sz_2; i++) {
86        ll res = p[i];
87        for (int e = 1;; e++) {
88          if (hd.val * res < n / n_3) {
```

```
 89            q.emplace(i, hd.val * res, type == -1 ? hd.f_val * f(p[i], e) : hd.f_val * mpow(res,
       type));
 90          } else break;
 91          res *= p[i];
 92        }
 93      }
 94    }
 95  }
 96  void get_f_p(ll n, int times) {
 97    for (int i = 1; i <= val_id_num; i++)
 98      for (int j = 0; j <= times; j++)
 99        f_p[i][j] = pow_sum(val_id[i], j) - 1;
100    int now;
101    for (now = 1; p[now] <= n_6; now++) {
102      for (int j = val_id_num; j >= 1; j--) {
103        ll w = val_id[j] / p[now];
104        if (w < p[now]) break;
105        ll val = 1;
106        for (int k = 0; k <= times; k++) {
107          f_p[j][k] = f_p[j][k] - val * (f_p[get_id(w)][k] - f_p[p[now - 1]][k]);
108          val *= p[now];
109        }
110      }
111    }
112    int nnow = now, val = 1;
113    for (int tt = 0; tt <= times; tt++) {
114      now = nnow;
115      memset(c, 0, sizeof c);
116      add(1, f_p[1][tt]);
117      for (int i = 2; val_id[i] < n / n_3; i++) add(i, f_p[i][tt] - f_p[i - 1][tt]);
118      for (; p[now] <= n_3; now++) {
119        for (int j = val_id_num; j >= 1; j--) {
120          ll w = val_id[j] / p[now];
121          if (val_id[j] < n / n_3) break;
122          if (w < p[now]) break;
123          f_p[j][tt] = w < n / n_3
124            ? (f_p[j][tt] - (sum(get_id(w)) - sum(p[now - 1])) * mpow(p[now], tt))
125            : f_p[j][tt] = f_p[j][tt] - (f_p[get_id(w)][tt] - sum(p[now - 1])) * mpow(p[now], tt);
126        }
127        update_bfs(now, tt);
128      }
129      for (int i = 1; i <= val_id_num && val_id[i] < n / n_3; i++) f_p[i][tt] = sum(i);
130      for (; now <= p_sz_2; now++) {
131        for (int j = val_id_num; j >= 1; j--) {
132          ll w = val_id[j] / p[now];
133          if (val_id[j] < n / n_3) break;
134          if (w < p[now]) break;
135          f_p[j][tt] -= (f_p[get_id(w)][tt] - f_p[p[now - 1]][tt]) * mpow(p[now], tt);
136        }
137      }
138    }
139
140    for (int i = 1; i <= val_id_num; i++) {
141      // if f(p) = p^2+3p+1,then write: f_p[i][0] = f_p[i][2] + 3*f_p[i][1] + f_p[i][0];
142      f_p[i][0] = f_p[i][2] + 3 * f_p[i][1] + f_p[i][0];
143    }
144  }
145  ll F[2000000 + 100];
146  void get_f_3(ll n) { // V(F_{pi(n^(1/3))+1},n)
147    ll q = p[p_sz_3 + 1];
148    for (int now = 1; now <= val_id_num; now++) {
149      if (val_id[now] < q) {
150        F[now] = 1;
151      } else if (val_id[now] < q * q) {
152        F[now] = 1 + (f_p[now][0] - f_p[q - 1][0]);
153      } else {
154        F[now] = 1 + (f_p[now][0] - f_p[q - 1][0]);
155        for (int pp = p_sz_3 + 1; p[pp] <= (int)(sqrt(val_id[now])) && pp <= p_sz_2; pp++) {
```

```
156          F[now] += f(p[pp], 2) + (f(p[pp], 1)) * (f_p[get_id(val_id[now] / p[pp])][0] - f_p[get_id(
             p[pp])][0]);
157        }
158      }
159    }
160  }
161  void get_f_6(ll n) { // V(F_{pi(n^(1/6))+1},n)
162    memset(c, 0, sizeof c), add(1, F[1]);
163    for (int i = 2; val_id[i] < n / n_3; i++) add(i, F[i] - F[i - 1]);
164    for (int k = p_sz_3; k > p_sz_6; k--) {
165      int now = val_id_num;
166      for (; val_id[now] >= n / n_3; now--) {
167        int e = 1;
168        ll _p = p[k];
169        while (val_id[now] / _p) {
170          if (val_id[now] / _p >= n / n_3) {
171            F[now] += F[get_id(val_id[now] / _p)] * f(p[k], e);
172          } else {
173            F[now] += sum(get_id(val_id[now] / _p)) * f(p[k], e);
174          }
175          _p *= p[k], e++;
176        }
177      }
178      if (k == 1) break;
179      update_bfs(k, -1); // bfs to update [lpf(i)==P{k-1}]f(i)
180    }
181    for (int i = 1; i <= val_id_num && val_id[i] < n / n_3; i++) F[i] = sum(i);
182  }
183  void get_f(ll n) {
184    for (int k = p_sz_6; k >= 1; k--) {
185      for (int now = val_id_num; now >= 1; now--) {
186        int e = 1;
187        ll _p = p[k];
188        while (val_id[now] / _p) {
189          F[now] += F[get_id(val_id[now] / _p)] * f(p[k], e);
190          _p *= p[k], e++;
191        }
192      }
193    }
194  }
195  int main() { // n = 1000000000; 1e10:455052511,0.83s/0.58s; 1e12:37607912018 9.224s/5.105s
196    cin >> n;
197    init();
198    get_f_p(n, 2), get_f_3(n);
199    get_f_6(n), get_f(n);
200    for (int i = 1; i <= val_id_num; i++) cout << val_id[i] << " : " << F[i] << endl;
201  }
```

1.9   类欧几里德算法

求 $f = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor, g = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor, h = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$

```
1   const ll P = 998244353;
2   ll i2 = 499122177, i6 = 166374059;
3   struct data {
4     data() { f = g = h = 0; }
5     ll f, g, h;
6   }; // 三个函数打包
7   data calc(ll n, ll a, ll b, ll c) {
8     ll ac = a / c, bc = b / c, m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
9     data d;
10    if (a == 0) { // 迭代到最底层
11      d.f = bc * n1 % P;
12      d.g = bc * n % P * n1 % P * i2 % P;
13      d.h = bc * bc % P * n1 % P;
14      return d;
15    }
```

```
16    if (a >= c || b >= c) { // 取模
17      d.f = n * n1 % P * i2 % P * ac % P + bc * n1 % P;
18      d.g = ac * n % P * n1 % P * n21 % P * i6 % P + bc * n % P * n1 % P * i2 % P;
19      d.h = ac * ac % P * n % P * n1 % P * n21 % P * i6 % P +
20            bc * bc % P * n1 % P + ac * bc % P * n % P * n1 % P;
21      d.f %= P, d.g %= P, d.h %= P;
22
23      data e = calc(n, a % c, b % c, c); // 迭代
24
25      d.h += e.h + 2 * bc % P * e.f % P + 2 * ac % P * e.g % P;
26      d.g += e.g, d.f += e.f;
27      d.f %= P, d.g %= P, d.h %= P;
28      return d;
29    }
30    data e = calc(m - 1, c, c - b - 1, a);
31    d.f = n * m % P - e.f, d.f = (d.f % P + P) % P;
32    d.g = m * n % P * n1 % P - e.h - e.f, d.g = (d.g * i2 % P + P) % P;
33    d.h = n * m % P * (m + 1) % P - 2 * e.g - 2 * e.f - d.f;
34    d.h = (d.h % P + P) % P;
35    return d;
36  }
```

1.10 中国剩余定理

```
1  ll inv(ll a, ll p) {
2    ll x, y;
3    exgcd(a, p, x, y);
4    return (x + p) % p;
5  }
6  ll CRT(ll n, ll *a, ll *m) {
7    ll lcm = 1, res = 0;
8    for (ll i = 0; i < n; ++i) lcm *= m[i];
9    for (ll i = 0; i < n; ++i) {
10     ll t = lcm / m[i], x = inv(t, m[i]);
11     res = (res + a[i] * t % lcm * x) % lcm;
12   }
13   return res;
14 }
```

模数不互质的情况 $\begin{cases} x \equiv a_1 \quad (\pmod{m})_1 \\ x \equiv a_2 \quad (\pmod{m})_2 \end{cases}$，则转换为 $m_1 p - m_2 q = a_2 - a_1$，最终解（若有解）为 $x \equiv m_1 p + a_1 \pmod{\mathrm{lcm}(m_1, m_2)}$。

拓展中国剩余定理:

```
1  int exctr(int n, int *a, int *m) {
2    int M = m[0], res = a[0];
3    for (int i = 1; i < n; ++i) {
4      int a = M, b = m[i], c = (a[i] - res % b + b) % b, x, y;
5      int g = exgcd(a, b, x, y), bg = b / g;
6      if (c % g != 0) return -1;
7      x = 1LL * x * (c / g) % bg;
8      res += x * M;
9      M *= bg;
10     res = (res % M + M) % M;
11   }
12   return res;
13 }
```

1.11 原根

- 阶: 若 $(a, m) = 1$, 使 $a^l \equiv 1 \pmod{m}$ 成立的最小的 l, 称为 a 关于模 m 的阶, 记为 $\mathrm{ord}_m a$。

- 原根: 若 $(g,m) = 1, \mathrm{ord}_m g = \varphi(m)$, 则称 g 为 m 的一个原根。若 m 有原根, 则 m 一定是下列形式: $\{2, 4, p^a, 2p^a\}$ （这里 p 为奇素数, a 为正整数）。

- 求原根: 设 $p_1, p_2, \cdots, p_k$ 是 $\varphi(m)$ 的所有不同的素因数, 则 g 是 m 的原根 $\iff$ $\forall 1 \leq i \leq k$, 有 $g^{\frac{\varphi(m)}{p_i}} \not\equiv 1 \pmod{m}$, 集合 $S = g^s \mid 1 \leq s \leq \varphi(m), (s, \varphi(m)) = 1$ 给出 m 的全部原根。

1.12 BGSG

大步小步算法用来求离散对数 $x^k \equiv a \pmod{p}$。求解 $a^x \equiv b \pmod{p}$ 则, 令 $x = A\lceil \sqrt{p} \rceil - B, 0 \leq A, B \leq \lceil \sqrt{p} \rceil$, 有 $a^{A\lceil \sqrt{p} \rceil} \equiv ba^B \pmod{p}$, 先枚举 A 之后在哈希表中查找 B 就行。

```cpp
// Finds the primitive root modulo p
int generator(int p) {
  vector<int> fact;
  int phi = p - 1, n = phi;
  for (int i = 2; i * i <= n; ++i) {
    if (n % i == 0) {
      fact.push_back(i);
      while (n % i == 0) n /= i;
    }
  }
  if (n > 1) fact.push_back(n);
  for (int res = 2; res <= p; ++res) {
    bool ok = true;
    for (int factor : fact)
      if (mpow(res, phi / factor, p) == 1) {
        ok = false;
        break;
      }
    if (ok) return res;
  }
  return -1;
}
vector<int> BSGS(int n, int k, int a) {
  if (a == 0) return vector<int>({0});

  int g = generator(n);
  // Baby-step giant-step discrete logarithm algorithm
  int sq = (int)sqrt(n + .0) + 1;
  vector<pair<int, int>> dec(sq);
  for (int i = 1; i <= sq; ++i)
    dec[i - 1] = {mpow(g, i * sq * k % (n - 1), n), i};

  sort(dec.begin(), dec.end());
  int any_ans = -1;
  for (int i = 0; i < sq; ++i) {
    int my = mpow(g, i * k % (n - 1), n) * a % n;
    auto it = lower_bound(dec.begin(), dec.end(), make_pair(my, 0));
    if (it != dec.end() && it->first == my) {
      any_ans = it->second * sq - i;
      break;
    }
  }
  if (any_ans == -1) return vector<int>();
  // Print all possible answers
  int delta = (n - 1) / __gcd(k, n - 1);
  vector<int> ans;
  for (int cur = any_ans % delta; cur < n - 1; cur += delta)
    ans.push_back(mpow(g, cur, n));
  sort(ans.begin(), ans.end());
  return ans;
}
```

1.13 自适应 Simpson

计算 $\int_a^b f(x)dx$。

```cpp
double simpson(double a, double b) {
  double c = a + (b - a) / 2;
  return (f(a) + 4 * f(c) + f(b)) * (b - a) / 6;
}
double integral(double a, double b, double eps, double A) {
  double c = a + (b - a) / 2;
  double L = simpson(a, c), R = simpson(c, b);
  if (fabs(L + R - A) <= 15 * eps) return L + R + (L + R - A) / 15;
  return integral(a, c, eps / 2, L) + integral(c, b, eps / 2, R);
}
double integral(double a, double b, double eps) {
  return integral(a, b, eps, simpson(a, b));
}
```

1.14 卢卡斯定理

卢卡斯定理: $\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$

```cpp
ll lucas(ll n, ll m, int p) {
  ll ret = 1;
  while (n && m) {
    ll nn = n % p, mm = m % p;
    if (nn < mm) return 0;
    ret = ret * fac[nn] % p * inv_fac[mm] % p * inv_fac[nn - mm] % p;
    n /= p, m /= p;
  }
  return ret;
}
```

拓展卢卡斯定理: 用于处理 p 不是质数的情况。

对于 $C_n^m \bmod p$, 我们将其转化为 r 个形如 $a_i \equiv C_n^m \pmod{q_i^{\alpha_i}}$ 的同余方程并分别求解; 对于 $a_i \equiv C_n^m$ $\pmod{q_i^{\alpha_i}}$, 将 C_n^m 转化为 $\frac{\frac{n!}{q^x}}{\frac{m!}{q^y} \frac{(n-m)!}{q^z}} q^{x-y-z}$, 可求逆元; 对于 $\frac{m!}{q^y}$ 和 $\frac{(n-m)!}{q^z}$, 将其变换整理, 可递归求解。

```cpp
ll calc(ll n, ll x, ll p) {
  if (!n) return 1;
  ll s = 1;
  for (ll i = 1; i <= p; ++i) (i % x) && (s = s * i % p);
  s = mpow(s, n / p, p);
  for (ll i = n / p * p; i <= n; ++i) (i % x) && (s = s * (i % p) % p);
  return s * calc(n / x, x, p) % p;
}

ll multi_lucas(ll n, ll m, ll x, ll p) {
  ll cnt = 0;
  for (ll i = n; i; i /= x) cnt += i / x;
  for (ll i = m; i; i /= x) cnt -= i / x;
  for (ll i = n - m; i; i /= x) cnt -= i / x;
  return mpow(x, cnt, p) * calc(n, x, p) % p * inv(calc(m, x, p), p) % p *
         inv(calc(n - m, x, p), p) % p;
}

ll exlucas(ll n, ll m, ll P) {
  ll cnt = 0;
  static ll p[20], a[20];
  for (ll i = 2; i * i <= P; ++i) {
    if (P % i) continue;
    p[cnt] = 1;
    while (P % i == 0) p[cnt] *= i, P /= i;
    a[cnt] = multi_lucas(n, m, i, p[cnt]);
    ++cnt;
  }
  if (P > 1) p[cnt] = P, a[cnt] = multi_lucas(n, m, P, P), ++cnt;
```

```
30      return CRT(cnt, a, p);
31  }
```

1.15　Burnside 引理

设 A 和 B 为有限集合，$X = B^A$ 表示所有从 A 到 B 的映射。G 是 A 上的置换群，X/G 表示 G 作用在 X 上产生的所有等价类的集合（若 X 中的两个映射经过 G 中的置换作用后相等，则它们在同一等价类中），则 $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$。

Gym 101873B: m 边形，每边是 $n \times n$ 的矩形，用 c 种颜色染色，可进行水平旋转，问不同多边形个数。

```
1  for (int i = 1; i <= m; ++i) ans = (ans + mpow(c, n * n * __gcd(i, m))) % MOD;
2  ans = 1LL * ans * mpow(m, MOD - 2) % MOD;
```

1.16　Pólya 定理

前置条件与 Burnside 引理相同，内容修改为 $|X/G| = \frac{1}{|G|} \sum_{g \in G} |B|^{c(g)}$，其中 $c(g)$ 表示置换 g 能拆分成的不相交的循环置换的数量。

1.17　高斯解线性方程

```
1   const double EPS = 1e-9;
2   const int MAXN = MAX_NODE;
3   double a[MAXN][MAXN], x[MAXN];
4   int equ, var;
5
6   int gauss() {
7     int i, j, k, col, max_r;
8     for (k = 0, col = 0; k < equ && col < var; k++, col++) {
9       max_r = k;
10      for (i = k + 1; i < equ; i++)
11        if (fabs(a[i][col]) > fabs(a[max_r][col])) max_r = i;
12      if (fabs(a[max_r][col]) < EPS) return 0;
13
14      if (k != max_r) {
15        for (j = col; j < var; j++) swap(a[k][j], a[max_r][j]);
16        swap(x[k], x[max_r]);
17      }
18
19      x[k] /= a[k][col];
20      for (j = col + 1; j < var; j++) a[k][j] /= a[k][col];
21      a[k][col] = 1;
22
23      for (i = k + 1; i < equ; i++)
24        if (i != k) {
25          x[i] -= x[k] * a[i][col];
26          for (j = col + 1; j < var; j++) a[i][j] -= a[k][j] * a[i][col];
27          a[i][col] = 0;
28        }
29    }
30
31    for (col = equ - 1, k = var - 1; ~col; --col, --k) {
32      if (fabs(a[col][k]) > 0) {
33        for (i = 0; i < k; ++i) {
34          x[i] -= x[k] * a[i][col];
35          for (j = col + 1; j < var; j++) a[i][j] -= a[k][j] * a[i][col];
36          a[i][col] = 0;
37        }
38      }
39    }
40
41    return 1;
42  }
```

1.18 线性回归

```cpp
struct LinearRecurrence {
  using int64 = long long;
  using vec = std::vector<int64>;

  static void extand(vec &a, size_t d, int64 value = 0) {
    if (d <= a.size()) return;
    a.resize(d, value);
  }

  static vec BerlekampMassey(const vec &s, int64 mod) {
    std::function<int64(int64)> inverse = [&](int64 a) {
      return a == 1 ? 1 : (int64)(mod - mod / a) * inverse(mod % a) % mod;
    };
    vec A = {1}, B = {1};
    int64 b = s[0];
    for (size_t i = 1, m = 1; i < s.size(); ++i, m++) {
      int64 d = 0;
      for (size_t j = 0; j < A.size(); ++j) { d += A[j] * s[i - j] % mod; }
      if (!(d %= mod)) continue;
      if (2 * (A.size() - 1) <= i) {
        auto temp = A;
        extand(A, B.size() + m);
        int64 coef = d * inverse(b) % mod;
        for (size_t j = 0; j < B.size(); ++j) {
          A[j + m] -= coef * B[j] % mod;
          if (A[j + m] < 0) A[j + m] += mod;
        }
        B = temp, b = d, m = 0;
      } else {
        extand(A, B.size() + m);
        int64 coef = d * inverse(b) % mod;
        for (size_t j = 0; j < B.size(); ++j) {
          A[j + m] -= coef * B[j] % mod;
          if (A[j + m] < 0) A[j + m] += mod;
        }
      }
    }
    return A;
  }

  static void exgcd(int64 a, int64 b, int64 &g, int64 &x, int64 &y) {
    if (!b)
      x = 1, y = 0, g = a;
    else {
      exgcd(b, a % b, g, y, x);
      y -= x * (a / b);
    }
  }

  static int64 crt(const vec &c, const vec &m) {
    int n = c.size();
    int64 M = 1, ans = 0;
    for (int i = 0; i < n; ++i) M *= m[i];
    for (int i = 0; i < n; ++i) {
      int64 x, y, g, tm = M / m[i];
      exgcd(tm, m[i], g, x, y);
      ans = (ans + tm * x * c[i] % M) % M;
    }
    return (ans + M) % M;
  }

  static vec ReedsSloane(const vec &s, int64 mod) {
    auto inverse = [](int64 a, int64 m) {
      int64 d, x, y;
      exgcd(a, m, d, x, y);
      return d == 1 ? (x % m + m) % m : -1;
```

```
 67        };
 68        auto L = [](const vec &a, const vec &b) {
 69          int da = (a.size() > 1 || (a.size() == 1 && a[0])) ? a.size() - 1 : -1000;
 70          int db = (b.size() > 1 || (b.size() == 1 && b[0])) ? b.size() - 1 : -1000;
 71          return std::max(da, db + 1);
 72        };
 73        auto prime_power = [&](const vec &s, int64 mod, int64 p, int64 e) {
 74          // linear feedback shift register mod p^e, p is prime
 75          std::vector<vec> a(e), b(e), an(e), bn(e), ao(e), bo(e);
 76          vec t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);
 77          ;
 78          pw[0] = 1;
 79          for (int i = pw[0] = 1; i <= e; ++i) pw[i] = pw[i - 1] * p;
 80          for (int64 i = 0; i < e; ++i) {
 81            a[i] = {pw[i]}, an[i] = {pw[i]};
 82            b[i] = {0}, bn[i] = {s[0] * pw[i] % mod};
 83            t[i] = s[0] * pw[i] % mod;
 84            if (t[i] == 0) {
 85              t[i] = 1, u[i] = e;
 86            } else {
 87              for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i])
 88                ;
 89            }
 90          }
 91          for (size_t k = 1; k < s.size(); ++k) {
 92            for (int g = 0; g < e; ++g) {
 93              if (L(an[g], bn[g]) > L(a[g], b[g])) {
 94                ao[g] = a[e - 1 - u[g]];
 95                bo[g] = b[e - 1 - u[g]];
 96                to[g] = t[e - 1 - u[g]];
 97                uo[g] = u[e - 1 - u[g]];
 98                r[g] = k - 1;
 99              }
100            }
101            a = an, b = bn;
102            for (int o = 0; o < e; ++o) {
103              int64 d = 0;
104              for (size_t i = 0; i < a[o].size() && i <= k; ++i) {
105                d = (d + a[o][i] * s[k - i]) % mod;
106              }
107              if (d == 0) {
108                t[o] = 1, u[o] = e;
109              } else {
110                for (u[o] = 0, t[o] = d; t[o] % p == 0; t[o] /= p, ++u[o])
111                  ;
112                int g = e - 1 - u[o];
113                if (L(a[g], b[g]) == 0) {
114                  extand(bn[o], k + 1);
115                  bn[o][k] = (bn[o][k] + d) % mod;
116                } else {
117                  int64 coef =
118                      t[o] * inverse(to[g], mod) % mod * pw[u[o] - uo[g]] % mod;
119                  int m = k - r[g];
120                  extand(an[o], ao[g].size() + m);
121                  extand(bn[o], bo[g].size() + m);
122                  for (size_t i = 0; i < ao[g].size(); ++i) {
123                    an[o][i + m] -= coef * ao[g][i] % mod;
124                    if (an[o][i + m] < 0) an[o][i + m] += mod;
125                  }
126                  while (an[o].size() && an[o].back() == 0) an[o].pop_back();
127                  for (size_t i = 0; i < bo[g].size(); ++i) {
128                    bn[o][i + m] -= coef * bo[g][i] % mod;
129                    if (bn[o][i + m] < 0) bn[o][i + m] -= mod;
130                  }
131                  while (bn[o].size() && bn[o].back() == 0) bn[o].pop_back();
132                }
133              }
134            }
```

```
135        }
136        return std::make_pair(an[0], bn[0]);
137      };
138
139      std::vector<std::tuple<int64, int64, int>> fac;
140      for (int64 i = 2; i * i <= mod; ++i)
141        if (mod % i == 0) {
142          int64 cnt = 0, pw = 1;
143          while (mod % i == 0) mod /= i, ++cnt, pw *= i;
144          fac.emplace_back(pw, i, cnt);
145        }
146      if (mod > 1) fac.emplace_back(mod, mod, 1);
147      std::vector<vec> as;
148      size_t n = 0;
149      for (auto &&x : fac) {
150        int64 mod, p, e;
151        vec a, b;
152        std::tie(mod, p, e) = x;
153        auto ss = s;
154        for (auto &&x : ss) x %= mod;
155        std::tie(a, b) = prime_power(ss, mod, p, e);
156        as.emplace_back(a);
157        n = std::max(n, a.size());
158      }
159      vec a(n), c(as.size()), m(as.size());
160      for (size_t i = 0; i < n; ++i) {
161        for (size_t j = 0; j < as.size(); ++j) {
162          m[j] = std::get<0>(fac[j]);
163          c[j] = i < as[j].size() ? as[j][i] : 0;
164        }
165        a[i] = crt(c, m);
166      }
167      return a;
168    }
169
170    LinearRecurrence(const vec &s, const vec &c, int64 mod)
171        : init(s), trans(c), mod(mod), m(s.size()) {}
172
173    LinearRecurrence(const vec &s, int64 mod, bool is_prime = true) : mod(mod) {
174      vec A = is_prime ? BerlekampMassey(s, mod) : ReedsSloane(s, mod);
175      if (A.empty()) A = {0};
176      m = A.size() - 1;
177      trans.resize(m);
178      for (int i = 0; i < m; ++i) { trans[i] = (mod - A[i + 1]) % mod; }
179      std::reverse(trans.begin(), trans.end());
180      init = {s.begin(), s.begin() + m};
181    }
182
183    int64 calc(int64 n) {
184      if (mod == 1) return 0;
185      if (n < m) return init[n];
186      vec v(m), u(m << 1);
187      int msk = !!n;
188      for (int64 m = n; m > 1; m >>= 1) msk <<= 1;
189      v[0] = 1 % mod;
190      for (int x = 0; msk; msk >>= 1, x <<= 1) {
191        std::fill_n(u.begin(), m * 2, 0);
192        x |= !!(n & msk);
193        if (x < m)
194          u[x] = 1 % mod;
195        else { // can be optimized by fft/ntt
196          for (int i = 0; i < m; ++i) {
197            for (int j = 0, t = i + (x & 1); j < m; ++j, ++t) {
198              u[t] = (u[t] + v[i] * v[j]) % mod;
199            }
200          }
201          for (int i = m * 2 - 1; i >= m; --i) {
202            for (int j = 0, t = i - m; j < m; ++j, ++t) {
```

```
203            u[t] = (u[t] + trans[j] * u[i]) % mod;
204          }
205        }
206      }
207      v = {u.begin(), u.begin() + m};
208    }
209    int64 ret = 0;
210    for (int i = 0; i < m; ++i) { ret = (ret + v[i] * init[i]) % mod; }
211    return ret;
212  }
213
214  vec init, trans;
215  int64 mod;
216  int m;
217 };
```

1.19 线性规划

给定 n 个约束条件, m 个未知数, 求 $\sum(a[0][i] \times x[i])$ 的最大值, 约束条件: $\sum(-a[i][j] \times x[j]) \le a[i][0]$。若要求最小值, 则进行对偶: 即把目标函数的系数和约束条件右边的数交换, 然后把矩阵转置。

```
1  const int MAXN = 3e3 + 3, MAXM = 3e3 + 3, INF = ~0U >> 2;
2  int n, m, a[MAXN][MAXM], nxt[MAXM];
3  void pivot(int l, int e) {
4    a[l][e] = -1;
5    int t = MAXM - 1;
6    for (int i = 0; i <= m; ++i)
7      if (a[l][i]) nxt[t] = i, t = i;
8    nxt[t] = -1;
9    for (int i = 0; i <= n; ++i)
10     if (i != l && (t = a[i][e])) {
11       a[i][e] = 0;
12       for (int j = nxt[MAXM - 1]; ~j; j = nxt[j]) a[i][j] += a[l][j] * t;
13     }
14 }
15 int simplex() {
16   for (;;) {
17     int mi = INF, l = 0, e = 0;
18     for (int i = 1; i <= m; ++i)
19       if (a[0][i] > 0) {
20         e = i;
21         break;
22       }
23     if (!e) return a[0][0];
24     for (int i = 1; i <= n; ++i)
25       if (a[i][e] < 0 && a[i][0] < mi) mi = a[i][0], l = i;
26     pivot(l, e);
27   }
28 }
```

1.20 实数线性规划

求 $\max\{c\vec{x} | A\vec{x} <= b, \vec{x} >= 0\}$。

```
1  typedef vector<double> VD;
2  VD simplex(vector<VD> A, VD b, VD c) {
3    int n = A.size(), m = A[0].size() + 1, r = n, s = m - 1;
4    vector<VD> D(n + 2, VD(m + 1, 0));
5    vector<int> ix(n + m);
6    for (int i = 0; i < n + m; ++i) ix[i] = i;
7    for (int i = 0; i <= n; ++i) {
8      for (int j = 0; j < m - 1; ++j) D[i][j] = -A[i][j];
9      D[i][m - 1] = 1, D[i][m] = b[i];
10     if (D[r][m] > D[i][m]) r = i;
11   }
```

```
12    for (int j = 0; j < m - 1; ++j) D[n][j] = c[j];
13    D[n + 1][m - 1] = -1;
14    for (double d;;) {
15      if (r < n) {
16        swap(ix[s], ix[r + m]);
17        D[r][s] = 1 / D[r][s];
18        vector<int> speed_up;
19        for (int j = 0; j <= m; ++j)
20          if (j != s) {
21            D[r][j] *= -D[r][s];
22            if (D[r][j]) speed_up.push_back(j);
23          }
24        for (int i = 0; i <= n + 1; ++i)
25          if (i != r) {
26            for (int j : speed_up) D[i][j] += D[r][j] * D[i][s];
27            D[i][s] *= D[r][s];
28          }
29      }
30      r = -1, s = -1;
31      for (int j = 0; j < m; ++j)
32        if ((s < 0 || ix[s] > ix[j]) &&
33            (D[n + 1][j] > EPS || (D[n + 1][j] > -EPS && D[n][j] > EPS)))
34          s = j;
35      if (s < 0) break;
36      for (int i = 0; i < n; ++i)
37        if (D[i][s] < -EPS)
38          if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] / D[i][s]) < -EPS ||
39              (d < EPS && ix[r + m] > ix[i + m]))
40            r = i;
41      if (r < 0) return VD(); //无边界
42    }
43    if (D[n + 1][m] < -EPS) return VD(); // 无解
44    VD x(m - 1);
45    for (int i = m; i < n + m; ++i)
46      if (ix[i] < m - 1) x[ix[i]] = D[i - m][m];
47    return x; // 最优值在D[n][m]
48  }
```

UOJ 板题最快榜代码:

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define eps 1e-7
5  int simplex(vector<vector<double>> &a, vector<double> &b, vector<double> &c,
6              vector<int> &basic) {
7    int m = b.size(), n = c.size();
8    while (true) {
9      int k = -1;
10     for (int j = 0; j < n; ++j)
11       if (c[j] < -eps) {
12         k = j;
13         break;
14       }
15     if (k == -1) {
16       double ans = 0;
17       for (int i = 0; i < m; ++i) ans += c[basic[i]] * b[i];
18       return 0;
19     }
20     int l = -1;
21     for (int i = 0; i < m; ++i)
22       if (a[i][k] > eps) {
23         if (l == -1)
24           l = i;
25         else {
26           double ti = b[i] / a[i][k], tl = b[l] / a[l][k];
27           if (ti < tl - eps || (ti < tl + eps && basic[i] < basic[l])) l = i;
28         }
29       }
```

17

```
30        if (l == -1) return -1;
31        basic[l] = k;
32        double tmp = 1 / a[l][k];
33        for (int j = 0; j < n; ++j) a[l][j] *= tmp;
34        b[l] *= tmp;
35        for (int i = 0; i < m; ++i)
36          if (i != l) {
37            tmp = a[i][k];
38            for (int j = 0; j < n; ++j) a[i][j] -= tmp * a[l][j];
39            b[i] -= tmp * b[l];
40          }
41        tmp = c[k];
42        for (int j = 0; j < n; ++j) c[j] -= tmp * a[l][j];
43      }
44  }
45
46  int main() {
47    ios::sync_with_stdio(false);
48    int n, m, T;
49    cin >> n >> m >> T;
50    vector<double> c(n + m, 0);
51    for (int i = 0; i < n; ++i) {
52      cin >> c[i];
53      c[i] *= -1;
54    }
55    auto C = c;
56    vector<vector<double>> a(m, vector<double>(n + m, 0));
57    vector<double> b(m);
58    vector<int> basic(m, -1), tmp;
59    for (int i = 0; i < m; ++i) {
60      for (int j = 0; j < n; ++j) cin >> a[i][j];
61      a[i][i + n] = 1;
62      cin >> b[i];
63      if (b[i] > -eps)
64        basic[i] = i + n;
65      else
66        tmp.push_back(i);
67    }
68    if (!tmp.empty()) {
69      sort(tmp.begin(), tmp.end(), [&](int i, int j) { return b[i] > b[j]; });
70      vector<vector<double>> A;
71      vector<double> B, C(n + m + 1, 0);
72      vector<int> Basic;
73      for (int i : tmp) {
74        vector<double> foo;
75        for (int j = 0; j < n + m; ++j) foo.push_back(-a[i][j]);
76        foo.push_back(1);
77        double bar = -b[i];
78        for (int i = 0; i < A.size(); ++i) {
79          double tmp = foo[Basic[i]];
80          for (int j = 0; j <= n + m; ++j) foo[j] -= tmp * A[i][j];
81          bar -= tmp * B[i];
82        }
83        for (int j = n + m; j >= 0; --j)
84          if (-eps < foo[j] - 1 && foo[j] - 1 < eps) {
85            Basic.push_back(j);
86            break;
87          }
88        for (int i = 0; i < A.size(); ++i) {
89          double tmp = A[i][Basic.back()];
90          for (int j = 0; j <= n + m; ++j) A[i][j] -= tmp * foo[j];
91          B[i] -= tmp * bar;
92        }
93        A.push_back(foo);
94        B.push_back(bar);
95      }
96      for (int i = 0; i < A.size(); ++i)
97        if (Basic[i] == n + m) {
```

```
 98            for (int j = 0; j < n + m; ++j) C[j] = -A[i][j];
 99          }
100        for (int i = 0; i < m; ++i)
101          if (b[i] > -eps) {
102            A.push_back(a[i]);
103            A[A.size() - 1].push_back(0);
104            B.push_back(b[i]);
105            Basic.push_back(basic[i]);
106          }
107        simplex(A, B, C, Basic);
108        bool flag = true;
109        for (int i = 0; i < m; ++i)
110          if (Basic[i] == n + m) {
111            if (B[i] > eps) {
112              cout << "Infeasible\n";
113              return 0;
114            }
115            int k = -1;
116            for (int j = 0; j < n + m; ++j)
117              if (A[i][j] > eps || A[i][j] < -eps) {
118                k = j;
119                break;
120              }
121            if (k != -1) {
122              double tmp = 1 / A[i][k];
123              Basic[i] = k;
124              for (int j = 0; j <= n + m; ++j) A[i][j] *= tmp;
125              B[i] *= tmp;
126              for (int l = 0; l < m; ++l)
127                if (l != i) {
128                  tmp = A[l][k];
129                  for (int j = 0; j <= n + m; ++j) A[l][j] -= tmp * A[i][j];
130                  B[l] -= tmp * B[i];
131                }
132            } else
133              flag = false;
134            break;
135          }
136        if (flag) {
137          A.push_back(vector<double>(n + m, 0));
138          A[A.size() - 1].push_back(1);
139          B.push_back(0);
140          Basic.push_back(n + m);
141          for (int i = 0; i < A.size() - 1; ++i) {
142            double tmp = A[i].back();
143            for (int j = 0; j <= n + m; ++j) A[i][j] -= tmp * A[A.size() - 1][j];
144            B[i] -= tmp * B.back();
145          }
146        }
147        a = A;
148        b = B;
149        basic = Basic;
150        c.push_back(0);
151        for (int i = 0; i < a.size(); ++i) {
152          double tmp = c[basic[i]];
153          for (int j = 0; j <= n + m; ++j) c[j] -= tmp * a[i][j];
154        }
155      }
156      auto foo = simplex(a, b, c, basic);
157      if (foo == -1)
158        cout << "Unbounded" << endl;
159      else {
160        double res = 0;
161        vector<double> ans(n, 0);
162        for (int i = 0; i < basic.size(); ++i)
163          if (basic[i] < n) ans[basic[i]] = b[i];
164        for (int j = 0; j < n; ++j) res -= C[j] * ans[j];
165        cout << setprecision(8) << res << endl;
```

```
166      if (T == 1) {
167        for (int i = 0; i < n; ++i) cout << setprecision(8) << ans[i] << ' ';
168        cout << endl;
169      }
170    }
171    return 0;
172  }
```

CCPC Final 2017 F: 有 N 组人, 每组有 a_i 人, 可进行若干次选择, 每次选择一些至少有 M 人的组, 这些人都中奖。现在要使每个人中奖概率相等, 且中奖概率最大 $N \le 10, M, a_i \le 100$。两种 LP 写法:

```
 1  const int MAXN = int(3e3);
 2  const int MAXM = int(3e3);
 3  const double INF = 1e20, EPS = 1e-9;
 4
 5  int n, m;
 6  double a[MAXM][MAXN], v;
 7
 8  void pivot(int l, int e) {
 9    int i, j;
10    a[l][e] = 1 / a[l][e];
11    for (j = 0; j <= n; ++j)
12      if (j != e) a[l][j] *= a[l][e];
13    for (i = 1; i <= m; ++i)
14      if (i != l && fabs(a[i][e]) > EPS) {
15        for (j = 0; j <= n; ++j)
16          if (j != e) a[i][j] -= a[i][e] * a[l][j];
17        a[i][e] = -a[i][e] * a[l][e];
18      }
19    v += a[0][e] * a[l][0];
20    for (j = 1; j <= n; ++j)
21      if (j != e) a[0][j] -= a[0][e] * a[l][j];
22    a[0][e] = -a[0][e] * a[l][e];
23  }
24
25  double simplex() {
26    int e, l, i;
27    double mn;
28    v = 0;
29    while (true) {
30      for (e = 1; e <= n; ++e)
31        if (a[0][e] > EPS) break;
32      if (e > n) return v;
33      for (i = 1, mn = INF; i <= m; ++i)
34        if (a[i][e] > EPS && mn > a[i][0] / a[i][e])
35          mn = a[i][0] / a[i][e], l = i;
36      if (mn == INF) return INF;
37      pivot(l, e);
38    }
39  }
40
41  void solve() {
42    static int n, m, g[10];
43    static vector<int> con[10], able;
44    scanf("%d %d", &n, &m);
45    for (int i = 0; i < n; ++i) {
46      scanf("%d", g + i);
47      con[i].clear();
48    }
49    if (n == 1) {
50      printf("%.10f\n", m >= g[0] ? 1. : 0.);
51      return;
52    }
53    able.clear();
54    for (int s = 0, S = 1 << n; s < S; ++s) {
55      int sum = 0;
56      for (int i = 0; i < n; ++i)
57        if (s >> i & 1) sum += g[i];
```

```
58        if (sum > m) continue;
59        able.push_back(s);
60        for (int i = 0; i < n; ++i)
61          if (s >> i & 1) con[i].push_back(able.size());
62      }
63      ::n = able.size();
64      ::m = 0;
65      static random_device rd;
66      mt19937 gen(rd());
67      shuffle(able.begin(), able.end(), gen);
68      for (int step = 0; step < n; ++step) {
69        int f = ++::m;
70        for (int i = 0; i <= ::n; ++i) a[f][i] = 0;
71        for (int x : con[step]) ++a[f][x];
72        if (step + 1 < n) {
73          for (int x : con[step + 1]) --a[f][x];
74        } else {
75          for (int x : con[0]) --a[f][x];
76        }
77      }
78
79      ++::m;
80      a[::m][0] = 1;
81      for (int i = 1; i <= ::n; ++i) a[::m][i] = 1;
82
83      ++::m;
84      a[::m][0] = -1;
85      for (int i = 1; i <= ::n; ++i) a[::m][i] = -1;
86
87      for (int i = 0; i <= ::n; ++i) a[0][i] = 0;
88      for (int x : con[0]) ++a[0][x];
89      printf("%.10f\n", simplex());
90    }
```

```
1   const int MAXN = 3000;
2   const int MAXM = 3000;
3   const db EPS = 1e-9;
4   const db INF = 1e200;
5
6   namespace LP {
7   db a[MAXM][MAXN];
8   int idA[MAXN], idB[MAXN];
9   int m, n;
10
11  void put_out(int x) {
12    if (x == 0)
13      printf("Infeasible\n");
14    else
15      printf("Unbounded\n");
16    exit(0);
17  }
18  void pivot(int xA, int xB) {
19    swap(idA[xA], idB[xB]);
20    static int next[MAXN];
21    int i, j, last = MAXN - 1;
22    db tmp = -a[xB][xA];
23    a[xB][xA] = -1.0;
24    for (j = 0; j <= n; j++)
25      if (fabs(a[xB][j]) > EPS) a[xB][last = next[last] = j] /= tmp;
26    next[last] = -1;
27
28    for (i = 0; i <= m; i++)
29      if (i != xB && fabs(tmp = a[i][xA]) > EPS)
30        for (a[i][xA] = 0.0, j = next[MAXN - 1]; ~j; j = next[j])
31          a[i][j] += tmp * a[xB][j];
32  }
33  db calc() {
34    int xA, xB;
```

```
35      db Max, tmp;
36      while (1) {
37        xA = n + 1, idA[xA] = n + m + 1;
38        for (int i = 1; i <= n; i++)
39          if (a[0][i] > EPS && idA[i] < idA[xA]) xA = i;
40
41        if (xA == n + 1) return a[0][0];
42        xB = m + 1, idB[xB] = n + m + 1, Max = -INF;
43        for (int i = 1; i <= m; i++)
44          if (a[i][xA] < -EPS && ((tmp = a[i][0] / a[i][xA]) > Max + EPS ||
45                                  (tmp > Max - EPS && idB[i] < idB[xB])))
46            Max = tmp, xB = i;
47        if (xB == m + 1) put_out(1);
48        pivot(xA, xB);
49      }
50      return a[0][0];
51  }
52  db solve() {
53      for (int i = 1; i <= n; i++) idA[i] = i;
54      for (int i = 1; i <= m; i++) idB[i] = n + i;
55      static db tmp[MAXN];
56      db Min = 0.0;
57      int l;
58      for (int i = 1; i <= m; i++)
59        if (a[i][0] < Min) Min = a[i][0], l = i;
60      if (Min > -EPS) return calc();
61
62      idA[++n] = 0;
63      for (int i = 1; i <= m; i++) a[i][n] = 1.0;
64      for (int i = 0; i <= n; i++) tmp[i] = a[0][i], a[0][i] = 0.0;
65      a[0][n] = -1.0;
66
67      pivot(n, l);
68      if (calc() < -EPS) put_out(0);
69      for (int i = 1; i <= m; i++)
70        if (!idB[i]) {
71          for (int j = 1; j <= n; j++)
72            if (fabs(a[0][j]) > EPS) {
73              pivot(j, i);
74              break;
75            }
76          break;
77        }
78
79      int xA;
80      for (xA = 1; xA <= n; xA++)
81        if (!idA[xA]) break;
82      for (int i = 0; i <= m; i++) a[i][xA] = a[i][n];
83      idA[xA] = idA[n], n--;
84
85      for (int i = 0; i <= n; i++) a[0][i] = 0.0;
86      for (int i = 1; i <= m; i++)
87        if (idB[i] <= n) {
88          for (int j = 0; j <= n; j++) a[0][j] += a[i][j] * tmp[idB[i]];
89        }
90
91      for (int i = 1; i <= n; i++)
92        if (idA[i] <= n) a[0][i] += tmp[idA[i]];
93      return calc();
94  }
95  db ans[MAXN];
96  void findAns() {
97      for (int i = 1; i <= n; i++) ans[i] = 0.0;
98      for (int i = 1; i <= m; i++)
99        if (idB[i] <= n) ans[idB[i]] = a[i][0];
100 }
101 void work() {
102     for (int i = 1; i <= m; ++i)
```

```cpp
103      for (int j = 1; j <= n; ++j) a[i][j] *= -1;
104    printf("%.10f\n", -double(solve()));
105  }
106  } // namespace LP
107
108  void solve() {
109    static int n, m, g[10];
110    static vector<int> con[10], able;
111
112    scanf("%d %d", &n, &m);
113    for (int i = 0; i < n; ++i) {
114      scanf("%d", g + i);
115      con[i].clear();
116    }
117    if (n == 1) {
118      printf("%.10f\n", m >= g[0] ? 1.0 : 0.0);
119      return;
120    }
121    able.clear();
122    for (int s = 0; s < (1 << n); ++s) {
123      int sum = 0;
124      for (int i = 0; i < n; ++i)
125        if (s >> i & 1) sum += g[i];
126      if (sum > m) continue;
127      able.push_back(s);
128      for (int i = 0; i < n; ++i)
129        if (s >> i & 1) con[i].push_back(able.size());
130    }
131
132    LP::n = able.size(), LP::m = 0;
133    for (int step = 0; step < n; ++step) {
134      int &f = ++LP::m;
135      for (int i = 0; i <= LP::n; ++i) LP::a[f][i] = 0;
136      for (int x : con[step]) ++LP::a[f][x];
137      if (step + 1 < n) {
138        for (int x : con[step + 1]) --LP::a[f][x];
139      } else {
140        for (int x : con[0]) --LP::a[f][x];
141      }
142    }
143
144    ++LP::m;
145    LP::a[LP::m][0] = 1;
146    for (int i = 1; i <= LP::n; ++i) LP::a[LP::m][i] = 1;
147
148    ++LP::m;
149    LP::a[LP::m][0] = -1;
150    for (int i = 1; i <= LP::n; ++i) LP::a[LP::m][i] = -1;
151
152    for (int i = 0; i <= LP::n; ++i) LP::a[0][i] = 0;
153    for (int x : con[0]) ++LP::a[0][x];
154
155    static db a2[MAXM][MAXN];
156    for (int i = 1; i <= LP::m; ++i)
157      for (int j = 1; j <= LP::n; ++j) a2[i][j] = LP::a[i][j];
158    for (int i = 1; i <= LP::m; ++i)
159      for (int j = 1; j <= LP::n; ++j) LP::a[j][i] = a2[i][j];
160    swap(LP::n, LP::m);
161    for (int i = 1; i <= max(LP::n, LP::m); ++i) swap(LP::a[0][i], LP::a[i][0]);
162    LP::a[0][0] = 0;
163    for (int i = 1; i <= LP::m; ++i)
164      for (int j = 1; j <= LP::n; ++j) LP::a[i][j] *= -1;
165    for (int i = 1; i <= LP::m; ++i) LP::a[i][0] *= -1;
166    for (int i = 1; i <= LP::n; ++i) LP::a[0][i] *= -1;
167
168    LP::work();
169  }
```

1.21 快速傅里叶变换

```cpp
const int MAXN = 4 * 1e5 + 3;
const double PI = acos(-1);
complex<double> a[MAXN], b[MAXN];

int n, bit;
int rev[MAXN];

void fft(complex<double> *a, int sign) {
  for (int i = 0; i < n; ++i)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int j = 1; j < n; j <<= 1) {
    complex<double> wn(cos(2 * PI / (j << 1)), sign * sin(2 * PI / (j << 1)));
    for (int i = 0; i < n; i += (j << 1)) {
      complex<double> w(1, 0), t0, t1;
      for (int k = 0; k < j; ++k, w *= wn) {
        t0 = a[i + k], t1 = w * a[i + j + k];
        a[i + k] = t0 + t1, a[i + j + k] = t0 - t1;
      }
    }
  }
  if (sign == -1) for (int i = 0; i < n; ++i) a[i] /= n;
}

int main() {
  int n, m, x;
  cin >> n >> m;
  for (int i = 0; i <= n; ++i) cin >> x, a[i].real(x);
  for (int i = 0; i <= m; ++i) cin >> x, b[i].real(x);

  for (::n = 1, bit = 0; ::n <= n + m; ++bit) ::n <<= 1;
  rev[0] = 0;
  for (int i = 1; i < ::n; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
  fft(a, 1), fft(b, 1);
  for (int i = 0; i < ::n; ++i) a[i] *= b[i];
  fft(a, -1);
  for (int i = 0; i < n + m + 1; ++i) cout << int(a[i].real() + .5) << " \n"[i = n + m];
  return 0;
}
```

1.22 快速数论变换

998244353 原根为 3，1004535809 原根为 3，786433 原根为 10，880803841 原根为 26。

```cpp
void ntt(int *x, int n, int sign) {
  for (int i = 0; i < n; ++i)
    if (rev[i] < i) swap(x[i], x[rev[i]]);
  for (int j = 1; j < n; j <<= 1) {
    int gn = mpow(G, (P - 1) / (j << 1));
    for (int i = 0; i < n; i += (j << 1)) {
      for (int k = 0, g = 1; k < j; ++k, g = 1ll * g * gn % P) {
        int &a = x[i + j + k], &b = x[i + k], t = 1ll * g * a % P;
        (a = b - t) < 0 ? a += P : 0;
        (b = b + t) >= P ? b -= P : 0;
      }
    }
  }
  if (sign == -1) {
    reverse(x + 1, x + n);
    for (int i = 0, inv = mpow(n, P - 2); i < n; ++i)
      x[i] = 1ll * x[i] * inv % P;
  }
}
```

1.23　快速沃尔什变换

$C_i = \sum_{i=j \oplus k} A_j B_k$，其中 $\oplus$ 为位运算。

```cpp
void fwt(int *a, int n) {
  for (int d = 1; d < n; d <<= 1)
    for (int m = d << 1, i = 0; i < n; i += m)
      for (int j = 0; j < d; ++j) {
        int x = a[i + j], y = a[i + j + d];
        a[i + j] = x + y;                       // AND
        a[i + j + d] = x + y;                   // OR
        a[i + j] = x + y, a[i + j + d] = x - y; // XOR
      }
}
void ufwt(int *a, int n) {
  for (int d = 1; d < n; d <<= 1)
    for (int m = d << 1, i = 0; i < n; i += m)
      for (int j = 0; j < d; ++j) {
        int x = a[i + j], y = a[i + j + d];
        a[i + j] = x - y;                         // AND
        a[i + j + d] = y - x;                     // OR
        a[i + j] = (x + y) / 2, a[i + j + d] = (x - y) / 2; // XOR
      }
}
```

2　动态规划

2.1　斜率优化

树上斜率优化，定义 dp_i 表示 i 节点传递到根节点的最短耗时，规定 $dp_{root} = -P$，有如下转移方程 $dp_u = dp_v + dist(u,v)^2 + P$，$v$ 为 u 的祖先。

```cpp
vector<pii> adj[MAXN];
ll dp[MAXN], d[MAXN];
int n, p, q[MAXN], head, tail;

inline ll S(int a, int b) { return (d[b] - d[a]) << 1; }
inline ll G(int a, int b) { return dp[b] - dp[a] + d[b] * d[b] - d[a] * d[a]; }

void dfs(int u, int from) {
  vector<int> dhead, dtail;
  if (u ^ 1) {
    while (head + 2 <= tail &&
           S(q[head + 1], q[head]) * d[u] <= G(q[head + 1], q[head]))
      dhead.push_back(q[head++]);
    int v = q[head];
    dp[u] = dp[v] + p + (d[u] - d[v]) * (d[u] - d[v]);
  }
  while (head + 2 <= tail &&
         G(u, q[tail - 1]) * S(q[tail - 1], q[tail - 2]) <=
             G(q[tail - 1], q[tail - 2]) * S(u, q[tail - 1]))
    dtail.push_back(q[--tail]);
  q[tail++] = u;
  for (pii &e : adj[u]) {
    if (e.first == from) continue;
    d[e.first] = d[u] + e.second;
    dfs(e.first, u);
  }
  --tail;
  for (int i = dtail.size() - 1; ~i; --i) q[tail++] = dtail[i];
  for (int i = dhead.size() - 1; ~i; --i) q[--head] = dhead[i];
}

void solve() {
  cin >> n >> p;
  for (int i = 1; i <= n; ++i) adj[i].clear();
```

```
35   for (int i = 1, u, v, w; i < n; ++i) {
36     cin >> u >> v >> w;
37     adj[u].emplace_back(v, w);
38     adj[v].emplace_back(u, w);
39   }
40   dp[1] = -p;
41   head = tail = 0;
42   dfs(1, 1);
43
44   ll ans = 0;
45   for (int i = 1; i <= n; ++i)
46     if (dp[i] > ans) ans = dp[i];
47   cout << ans << '\n';
48 }
```

2.2 整数划分

$f_{i,j}$ 表示选了 i 种不同的数字，和为 j 的方案数。$f_{i,j} = f_{i-1,j-1} + f_{i,j-i}$，意义：要么新选一个 1，要么前面的数都加一。若每个数字最多一个，$f_{i,j} = f_{i-1,j-i} + f_{i,j-i}$。

求将 n 划分为若干整数的方案，则设 g_n 为答案，代码如下，时间复杂度 $O(n\sqrt{n})$。

```
1 int f[732], g[200001];
2 void init() {
3   f[1] = 1, f[2] = 2, f[3] = 5, f[4] = 7;
4   for (int i = 5; i < 732; ++i) f[i] = 3 + 2 * f[i - 2] - f[i - 4];
5   for (int i = g[0] = 1; i <= n; ++i)
6     for (int j = 1; f[j] <= i; ++j)
7       g[i] = ((j + 1) >> 1 & 1) ? (g[i] + g[i - f[j]]) % MOD : (g[i] - g[i - f[j]] + MOD) % MOD;
8 }
```

3 数据结构

3.1 KD Tree

```
1 // 寻找近点
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 const int MAXN = 2e5 + 5;
6 typedef long long ll;
7
8 namespace KD_Tree {
9
10 const int DIM = 2;
11
12 inline ll sqr(int x) { return 1LL * x * x; }
13
14 struct Point {
15   int x[DIM], id, c;
16
17   ll dist2(const Point &b) const {
18     return sqr(x[0] - b.x[0]) + sqr(x[1] - b.x[1]);
19   }
20 };
21 struct QNode {
22   Point p;
23   ll dis2;
24
25   QNode() {}
26   QNode(Point _p, ll _dis2) : p(_p), dis2(_dis2) {}
27
28   bool operator<(const QNode &b) const {
29     return dis2 < b.dis2 || (dis2 == b.dis2 && p.id < b.p.id);
```

```
30    }
31  } ans;
32  struct cmpx {
33    int div;
34    cmpx(int _div) : div(_div) {}
35    bool operator()(const Point &a, const Point &b) {
36      for (int i = 0; i < DIM; ++i)
37        if (a.x[(i + div) % DIM] != b.x[(i + div) % DIM])
38          return a.x[(i + div) % DIM] < b.x[(i + div) % DIM];
39      return true;
40    }
41  };
42
43  bool cmp(const Point &a, const Point &b, int div) {
44    cmpx cp = cmpx(div);
45    return cp(a, b);
46  }
47
48  struct Node {
49    Point e;
50    Node *lc, *rc;
51    int div;
52  } node_pool[MAXN], *tail, *root;
53  void init() { tail = node_pool; }
54  Node *build(Point *a, int l, int r, int div) {
55    if (l >= r) return nullptr;
56    Node *p = tail++;
57    p->div = div;
58    int mid = (l + r) >> 1;
59    nth_element(a + l, a + mid, a + r, cmpx(div));
60    p->e = a[mid];
61    p->lc = build(a, l, mid, div ^ 1);
62    p->rc = build(a, mid + 1, r, div ^ 1);
63    return p;
64  }
65  void search(Point p, Node *x, int div) {
66    if (!x) return;
67    if (cmp(p, x->e, div)) {
68      search(p, x->lc, div ^ 1);
69      if (ans.dis2 == -1) {
70        if (x->e.c <= p.c) ans = QNode(x->e, p.dist2(x->e));
71        search(p, x->rc, div ^ 1);
72      } else {
73        QNode temp(x->e, p.dist2(x->e));
74        if (x->e.c <= p.c && temp < ans) ans = temp;
75        if (sqr(x->e.x[div] - p.x[div]) <= ans.dis2) search(p, x->rc, div ^ 1);
76      }
77    } else {
78      search(p, x->rc, div ^ 1);
79      if (ans.dis2 == -1) {
80        if (x->e.c <= p.c) ans = QNode(x->e, p.dist2(x->e));
81        search(p, x->lc, div ^ 1);
82      } else {
83        QNode temp(x->e, p.dist2(x->e));
84        if (x->e.c <= p.c && temp < ans) ans = temp;
85        if (sqr(x->e.x[div] - p.x[div]) <= ans.dis2) search(p, x->lc, div ^ 1);
86      }
87    }
88  }
89  void search(Point p) {
90    ans.dis2 = -1;
91    search(p, root, 0);
92  }
93  } // namespace KD_Tree
94
95  void solve() {
96    static KD_Tree::Point p[MAXN];
97    int n, m;
```

```
 98 │    cin >> n >> m;
 99 │    for (int i = 0; i < n; ++i) {
100 │      p[i].id = i;
101 │      cin >> p[i].x[0] >> p[i].x[1] >> p[i].c;
102 │    }
103 │    KD_Tree::init();
104 │    KD_Tree::root = KD_Tree::build(p, 0, n, 0);
105 │
106 │    for (KD_Tree::Point q; m; --m) {
107 │      cin >> q.x[0] >> q.x[1] >> q.c;
108 │      KD_Tree::search(q);
109 │      cout << KD_Tree::ans.p.x[0] << ' ' << KD_Tree::ans.p.x[1] << ' '
110 │           << KD_Tree::ans.p.c << '\n';
111 │    }
112 │ }
113 │ int main() {
114 │    ios::sync_with_stdio(false);
115 │    cin.tie(nullptr);
116 │
117 │    int o_o;
118 │    for (cin >> o_o; o_o; --o_o) solve();
119 │
120 │    return 0;
121 │ }
122 │
123 │ // 寻找远点
124 │ inline void cmin(int &a, int b) { b < a ? a = b : 1; }
125 │ inline void cmax(int &a, int b) { a < b ? a = b : 1; }
126 │ inline int ibs(int a) { return a < 0 ? -a : a; }
127 │ struct D {
128 │    int d[2], mx0, mx1, mi0, mi1;
129 │    D *l, *r;
130 │ } t[N], *rt;
131 │ int cpd, ans;
132 │ inline bool cmp(const D &a, const D &b) {
133 │    return (a.d[cpd] ^ b.d[cpd]) ? a.d[cpd] < b.d[cpd]
134 │                                 : a.d[cpd ^ 1] < b.d[cpd ^ 1];
135 │ }
136 │ inline void kd_upd(D *u) {
137 │    if (u->l) {
138 │      cmax(u->mx0, u->l->mx0);
139 │      cmax(u->mx1, u->l->mx1);
140 │      cmin(u->mi0, u->l->mi0);
141 │      cmin(u->mi1, u->l->mi1);
142 │    }
143 │    if (u->r) {
144 │      cmax(u->mx0, u->r->mx0);
145 │      cmax(u->mx1, u->r->mx1);
146 │      cmin(u->mi0, u->r->mi0);
147 │      cmin(u->mi1, u->r->mi1);
148 │    }
149 │ }
150 │ D *kd_bld(int l, int r, int d) {
151 │    int m = l + r >> 1;
152 │    cpd = d;
153 │    std::nth_element(t + l + 1, t + m + 1, t + r + 1, cmp);
154 │    t[m].mx0 = t[m].mi0 = t[m].d[0];
155 │    t[m].mx1 = t[m].mi1 = t[m].d[1];
156 │    if (l ^ m) t[m].l = kd_bld(l, m - 1, d ^ 1);
157 │    if (r ^ m) t[m].r = kd_bld(m + 1, r, d ^ 1);
158 │    kd_upd(t + m);
159 │    return t + m;
160 │ }
161 │ inline void kd_ins(D *ne) {
162 │    int cd = 0;
163 │    D *u = rt;
164 │    while (true) {
165 │      cmax(u->mx0, ne->mx0), cmin(u->mi0, ne->mi0);
```

```
166        cmax(u->mx1, ne->mx1), cmin(u->mi1, ne->mi1);
167        if (ne->d[cd] < u->d[cd]) {
168          if (u->l)
169            u = u->l;
170          else {
171            u->l = ne;
172            return;
173          }
174        } else {
175          if (u->r)
176            u = u->r;
177          else {
178            u->r = ne;
179            return;
180          }
181        }
182        cd ^= 1;
183      }
184    }
185    inline int dist(int x, int y, D *u) {
186      int r = 0;
187      if (x < u->mi0)
188        r = u->mi0 - x;
189      else if (x > u->mx0)
190        r = x - u->mx0;
191      if (y < u->mi1)
192        r += u->mi1 - y;
193      else if (y > u->mx1)
194        r += y - u->mx1;
195      return r;
196    }
197    inline void kd_quy(D *u, const int &x, const int &y) {
198      int dl, dr, d0;
199      d0 = ibs(u->d[0] - x) + ibs(u->d[1] - y);
200      if (d0 < ans) ans = d0;
201      dl = u->l ? dist(x, y, u->l) : inf;
202      dr = u->r ? dist(x, y, u->r) : inf;
203      if (dl < dr) {
204        if (dl < ans) kd_quy(u->l, x, y);
205        if (dr < ans) kd_quy(u->r, x, y);
206      } else {
207        if (dr < ans) kd_quy(u->r, x, y);
208        if (dl < ans) kd_quy(u->l, x, y);
209      }
210    }
```

3.2 zkw 线段树

```
 1   int tree[MAXN * 2], pre;
 2
 3   void init(int n, int *a) {
 4     memset(tree, 0, sizeof(tree));
 5     for (pre = 1; pre <= n; pre <<= 1) {}
 6     for (int i = 1; i <= n; ++i) tree[i + pre] = a[i];
 7     for (int i = pre; i; --i) tree[i] = max(tree[i << 1], tree[i << 1 | 1]);
 8   }
 9
10   void update(int pos, const int &val) {
11     tree[pos += pre] = val;
12     for (pos >>= 1; pos; pos >>= 1)
13       tree[pos] = max(tree[pos << 1], tree[pos << 1 | 1]);
14   }
15
16   int query(int s, int t) {
17     int res = 0;
18     for (s += pre - 1, t += pre + 1; s ^ t ^ 1; s >>= 1, t >>= 1) {
```

```
19       if (~s & 1) res = max(res, tree[s ^ 1]);
20       if (t & 1) res = max(res, tree[t ^ 1]);
21     }
22     return res;
23   }
```

3.3 Splay

```
1   struct Node {
2     long long sum;
3     int id, val, lazy, size;
4     Node *fa, *ch[2];
5   } node_pool[MAXN], *pool_it, *root, *nil;
6
7   Node *newnode(int id, int val) {
8     pool_it->id = id;
9     pool_it->lazy = 0;
10    pool_it->size = 1;
11    pool_it->sum = pool_it->val = val;
12    pool_it->fa = pool_it->ch[0] = pool_it->ch[1] = nil;
13    return pool_it++;
14  }
15
16  void maintain(Node *u) {
17    if (u == nil) { return; }
18    u->size = u->ch[0]->size + u->ch[1]->size + 1;
19    u->sum = u->ch[0]->sum + u->ch[1]->sum + u->val;
20  }
21
22  void push_down(Node *u) {
23    if (u->lazy) {
24      if (u->ch[0] != nil) {
25        u->ch[0]->val += u->lazy;
26        u->ch[0]->sum += 1LL * u->ch[0]->size * u->lazy;
27        u->ch[0]->lazy += u->lazy;
28      }
29      if (u->ch[1] != nil) {
30        u->ch[1]->val += u->lazy;
31        u->ch[1]->sum += 1LL * u->ch[1]->size * u->lazy;
32        u->ch[1]->lazy += u->lazy;
33      }
34      u->lazy = 0;
35    }
36  }
37
38  inline void rot(Node *u) {
39    Node *f = u->fa, *ff = f->fa;
40    int d = u == f->ch[1];
41    push_down(f);
42    push_down(u);
43    if ((f->ch[d] = u->ch[d ^ 1]) != nil) f->ch[d]->fa = f;
44    if ((u->fa = ff) != nil) ff->ch[f == ff->ch[1]] = u;
45    f->fa = u;
46    u->ch[d ^ 1] = f;
47    maintain(f);
48    maintain(u);
49  }
50
51  void splay(Node *u, Node *target) {
52    for (Node *f; u->fa != target; rot(u))
53      if ((f = u->fa)->fa != target) {
54        ((u == f->ch[1]) ^ (f == f->fa->ch[1])) ? rot(u) : rot(f);
55      }
56    if (target == nil) root = u;
57  }
58
```

```
59   inline void insert(int id, int val) {
60     if (root == nil) {
61       root = newnode(id, val);
62       return;
63     }
64     Node *u = root;
65     while (u != nil) {
66       int d = id >= u->id;
67       ++u->size;
68       push_down(u);
69       u->sum += val;
70       if (u->ch[d] != nil) {
71         u = u->ch[d];
72       } else {
73         u->ch[d] = newnode(id, val);
74         u->ch[d]->fa = u;
75         u = u->ch[d];
76         break;
77       }
78     }
79     splay(u, nil);
80   }
81
82   inline Node *find_pred(int id) {
83     Node *u = root, *ret = nil;
84     while (u != nil) {
85       push_down(u);
86       if (u->id < id) {
87         ret = u;
88         u = u->ch[1];
89       } else {
90         u = u->ch[0];
91       }
92     }
93     return ret;
94   }
95
96   inline Node *find_succ(int id) {
97     Node *u = root, *ret = nil;
98     while (u != nil) {
99       push_down(u);
100      if (u->id > id) {
101        ret = u;
102        u = u->ch[0];
103      } else {
104        u = u->ch[1];
105      }
106    }
107    return ret;
108  }
109
110  Node *find_kth(int k) {
111    Node *u = root;
112    while (u != nil) {
113      push_down(u);
114      if (u->ch[0]->size + 1 == k) {
115        splay(u, nil);
116        return u;
117      }
118      if (u->ch[0]->size >= k) {
119        u = u->ch[0];
120      } else {
121        k -= u->ch[0]->size + 1;
122        u = u->ch[1];
123      }
124    }
125    return nil;
126  }
```

```
127
128  Node *range(int l, int r) {
129    Node *pred = find_pred(l);
130    Node *succ = find_succ(r);
131
132    splay(pred, nil);
133    splay(succ, root);
134    push_down(pred);
135    push_down(succ);
136    return root->ch[1]->ch[0];
137  }
138
139  void init() {
140    pool_it = node_pool;
141    nil = pool_it++;
142    nil->ch[0] = nil->ch[1] = nil->fa = nil;
143    nil->id = -1, nil->val = 0;
144    root = nil;
145
146    insert(INT_MIN, 0), insert(INT_MAX, 0);
147  }
```

3.4 LCT

```
 1  struct LCT {
 2    struct node {
 3      int val, add;
 4      node *fa, *ch[2];
 5      void modify(const int &x) {
 6        val += x, add += x;
 7      }
 8    } node_mset[MaxS], *cnode, *null;
 9    LCT() {
10      cnode = node_mset, null = cnode++;
11      *null = (node){0, 0, null, {null, null}};
12    }
13    inline node *newnode() {
14      *cnode = (node){0, 0, null, {null, null}};
15      return cnode++;
16    }
17    inline bool isrt(node *u) const {
18      return (u->fa->ch[0] != u) && (u->fa->ch[1] != u);
19    }
20    inline bool which(node *u) const { return u->fa->ch[1] == u; }
21    void push_down(node *u) {
22      if (!isrt(u)) push_down(u->fa);
23      if (u->add) {
24        u->ch[0]->modify(u->add);
25        u->ch[1]->modify(u->add);
26        u->add = 0;
27      }
28    }
29    inline void rotate(node *u) {
30      node *f = u->fa;
31      int d = which(u);
32      f->ch[d] = u->ch[d ^ 1];
33      f->ch[d]->fa = f;
34      u->ch[d ^ 1] = f;
35      u->fa = f->fa;
36      if (!isrt(f)) f->fa->ch[which(f)] = u;
37      f->fa = u;
38    }
39    inline void splay(node *u) {
40      push_down(u);
41      for (node *f; !isrt(u); rotate(u))
42        if (!isrt(f = u->fa)) rotate(which(u) == which(f) ? f : u);
```

```
43      }
44    inline void access(node *x) {
45      for (node *y = null; x != null; x = x->fa) {
46        splay(x);
47        x->ch[1] = y;
48        y = x;
49      }
50    }
51    inline void cut(node *u) {
52      access(u), splay(u);
53      u->ch[0]->fa = null;
54      u->ch[0] = null;
55    }
56    inline void link(node *u, node *v) {
57      cut(u), u->fa = v;
58    }
59  } tree;
```

4 字符串

4.1 KMP

```
1  void get_next(char *S, int *nxt, int n) {
2    nxt[0] = -1;
3    int j = -1;
4    for (int i = 1; i < n; ++i) {
5      while ((~j) && S[j + 1] != S[i]) j = nxt[j];
6      nxt[i] = (S[j + 1] == S[i]) ? (++j) : j;
7    }
8  }
9
10 int pattern(char *S, char *T, int *nxt, int n, int m) {
11   int j = -1;
12   for (int i = 0; i < m; ++i) {
13     while ((~j) && S[j + 1] != T[i]) j = nxt[j];
14     j += S[j + 1] == T[i];
15     if (j == n - 1) return i - n + 1;
16   }
17   return -1;
18 }
```

4.2 拓展 KMP

next[i]: x[i...m-1] 与 x[0...m-1] 的最长公共前缀，extend[i]: y[i...n-1] 与 x[0...m-1] 的最长公共前缀

```
1  void prework(char x[], int m, int next[]) {
2    next[0] = m;
3    int j = 0;
4    while (j + 1 < m && x[j] == x[j + 1]) ++j;
5    next[1] = j;
6    int k = 1;
7    for (int i = 2; i < m; ++i) {
8      int p = next[k] + k - 1;
9      int L = next[i - k];
10     if (i + L < p + 1)
11       next[i] = L;
12     else {
13       j = max(0, p - i + 1);
14       while (i + j < m && x[i + j] == x[j]) j++;
15       next[i] = j, k = i;
16     }
17   }
18 }
```

```
19  void exkmp(char x[], int m, char y[], int n, int next[], int extend[]) {
20    prework(x, m, next);
21    int j = 0;
22    while (j < n && j < m && x[j] == y[j]) ++j;
23    extend[0] = j;
24    int k = 0;
25    for (int i = 1; i < n; ++i) {
26      int p = extend[k] + k - 1;
27      int L = next[i - k];
28      if (i + L < p + 1)
29        extend[i] = L;
30      else {
31        j = max(0, p - i + 1);
32        while (i + j < n && j < m && y[i + j] == x[j]) j++;
33        extend[i] = j, k = i;
34      }
35    }
36  }
```

4.3 AC 自动机

```
1   int tr[MAX_NODE][26], fail[MAX_NODE], dep[MAX_NODE], node_c;
2
3   int add_char(int u, int id) {
4     if (tr[u][id] < 0) tr[u][id] = node_c++;
5     return tr[u][id];
6   }
7   void build_acam() {
8     queue<int> que;
9     fail[0] = 0;
10    for (int i = 0; i < 26; ++i)
11      if (~tr[0][i]) {
12        que.push(tr[0][i]);
13        fail[tr[0][i]] = 0;
14      } else {
15        tr[0][i] = 0;
16      }
17    while (!que.empty()) {
18      int u = que.front(), f = fail[u];
19      que.pop();
20      for (int i = 0; i < 26; ++i)
21        if (~tr[u][i]) {
22          que.push(tr[u][i]);
23          fail[tr[u][i]] = tr[f][i];
24        } else {
25          tr[u][i] = tr[f][i];
26        }
27    }
28    for (int i = 1; i < node_c; ++i) adj[fail[i]].push_back(i);
29  }
```

4.4 各种哈希

- 树哈希: 将子树当作集合哈希, 加入深度的影响。

- 集合哈希: 可以使用元素的哈希值映射为高进制的某一位, 也可以使用质数的积;

```
1   const unsigned int KEY = 6151;
2   const unsigned int MOD = 1610612741;
3   // 64 位哈希参数 KEY 随意 MOD 461168601842738784ll
4   unsigned int hash[MAXN], p[MAXN];
5
6   unsigned int get_hash(int l, int r) { return (hash[r] + MOD - 1ull * hash[l - 1] * p[r - l + 1] %
        MOD) % MOD; }
```

```
 7
 8   void init(char *s, int n) {
 9     p[0] = 1;
10     for (int i = 1; i <= n; ++i) {
11       p[i] = 1ull * p[i - 1] * KEY % MOD;
12       hash[i] = (1LL * hash[i - 1] * KEY + s[i]) % MOD;
13     }
14   }
```

CCPC 秦皇岛 2020 J: 两次哈希

```
 1   const int MAXN = 3e5 + 3;
 2   const int MAX_PRIME = 8960453 + 3;
 3   const int MOD = 998244353;
 4   const ll BASE = 709;
 5   const ll HASH_MOD = 461168601842738784ll;
 6
 7   char s[MAXN];
 8   int fac[MAXN], inv[MAXN], fac_inv[MAXN], prime[MAXN * 2];
 9   ll ha[MAXN], p[MAXN], pref[MAXN], suff[MAXN], value[MAXN * 2];
10   pair<ll, int> bin[MAXN];
11
12   int cnt[MAXN], pidx[MAXN], sidx[MAXN];
13   bitset<MAX_PRIME> mark;
14
15   ll fmul(ll a, ll b) {
16     ll k = (ll)((1.l * a * b) / (1.l * HASH_MOD)), t = a * b - k * HASH_MOD;
17     for (t -= HASH_MOD; t < 0; t += HASH_MOD) {}
18     return t;
19   }
20   ll getRange(int l, int r) {
21     return (ha[r] - fmul(ha[l - 1], p[r - l + 1]) + HASH_MOD) % HASH_MOD;
22   }
23
24   int gao(int n, int d) {
25     int tot = 0;
26     for (int l = 1, r = d; r <= n; l += d, r += d) value[tot++] = getRange(l, r);
27     int bunch = n / d, rest = n % d;
28     if (rest) {
29       for (int r = n, l = n - d + 1; l >= 1; l -= d, r -= d)
30         value[tot++] = getRange(l, r);
31       sort(value, value + tot);
32       tot = unique(value, value + tot) - value;
33       pref[0] = pref[1] = suff[n] = suff[n + 1] = 1;
34       for (int i = d; i <= n; i += d) {
35         int idx = lower_bound(value, value + tot, getRange(i - d + 1, i)) - value;
36         pidx[i] = idx;
37         pref[i] = fmul(pref[i - d], prime[idx]);
38       }
39       for (int i = n - d + 1; i >= 1; i -= d) {
40         int idx = lower_bound(value, value + tot, getRange(i, i + d - 1)) - value;
41         sidx[i] = idx;
42         suff[i] = fmul(suff[i + d], prime[idx]);
43       }
44       int sc = 0, cur = fac[bunch];
45       memset(cnt, 0, tot * sizeof(int));
46       for (int l = 1, r = rest; r <= n; l += d, r += d) {
47         if (r + d <= n) {
48           ++cnt[sidx[r + 1]];
49           cur = 1ll * cur * inv[cnt[sidx[r + 1]]] % MOD;
50         }
51         bin[sc++] = {fmul(pref[l - 1], suff[r + 1]), l};
52       }
53       sort(bin, bin + sc);
54       mark[bin[0].second] = 1;
55       for (int i = 1; i < sc; ++i)
56         mark[bin[i].second] = bin[i].first != bin[i - 1].first;
57
58       int res = 0;
```

```
59        for (int l = 1, r = rest; r <= n; l += d, r += d) {
60          if (l - 1 >= 1) {
61            ++cnt[pidx[l - 1]];
62            cur = 1ll * cur * inv[cnt[pidx[l - 1]]] % MOD;
63          }
64          if (mark[l]) {
65            res += cur;
66            if (res >= MOD) res -= MOD;
67          }
68          if (r + 1 <= n) {
69            cur = 1ll * cur * cnt[sidx[r + 1]] % MOD;
70            --cnt[sidx[r + 1]];
71          }
72        }
73        return res;
74      } else {
75        sort(value, value + tot);
76        ll pre = value[0];
77        int res = fac[bunch], cnt = 1;
78        for (int i = 1; i < tot; ++i) {
79          if (value[i] != pre) {
80            if (cnt > 1) res = 1ll * res * fac_inv[cnt] % MOD;
81            cnt = 1, pre = value[i];
82          } else {
83            ++cnt;
84          }
85        }
86        if (cnt > 1) res = 1ll * res * fac_inv[cnt] % MOD;
87        return res;
88      }
89    }
90
91    void solve() {
92      cin >> (s + 1);
93      int n = strlen(s + 1);
94      for (int i = 1; i <= n; ++i)
95        ha[i] = (fmul(ha[i - 1], BASE) + s[i]) % HASH_MOD;
96
97      int ans = 0;
98      for (int d = 1; d <= n; ++d) {
99        ans += gao(n, d);
100       if (ans >= MOD) ans -= MOD;
101     }
102     cout << ans << "\n";
103   }
104
105   void prework() {
106     p[0] = 1;
107     for (int i = 1; i < MAXN; ++i) p[i] = fmul(p[i - 1], BASE);
108     fac[0] = fac[1] = 1;
109     fac_inv[0] = fac_inv[1] = inv[1] = 1;
110     for (int i = 2; i < MAXN; ++i) {
111       fac[i] = 1ll * fac[i - 1] * i % MOD;
112       inv[i] = 1ll * (MOD - MOD / i) * inv[MOD % i] % MOD;
113       fac_inv[i] = 1ll * fac_inv[i - 1] * inv[i] % MOD;
114     }
115
116     int pc = 0;
117     for (int i = 2; i < MAX_PRIME; ++i) {
118       if (!mark[i]) prime[pc++] = i;
119       for (int j = 0; j < pc; ++j) {
120         int t = i * prime[j];
121         if (t >= MAX_PRIME) break;
122         mark[t] = 1;
123         if (i % prime[j] == 0) break;
124       }
125     }
126   }
```

```
127
128  int main(int argc, char *argv[]) {
129    ios::sync_with_stdio(false);
130    cin.tie(nullptr), cout.tie(nullptr);
131
132    prework();
133    int T; cin >> T;
134    for (int step = 1; step <= T; ++step) {
135      cout << "Case #" << step << ": ";
136      solve();
137    }
138
139    return 0;
140  }
```

4.5   mancher

```
1   void mancher(char *s, int n) {
2     static char str[2 * MAX_LENGTH];
3     str[0] = '~', str[1] = '!';
4     int len = 2;
5     for (int i = 0; i < n; ++i) {
6       str[len++] = s[i], str[len++] = '!';
7     }
8     str[len] = 0;
9     for (int i = 1, id = 0, mx = 0; i < len; ++i) {
10      p[i] = i < mx ? min(p[2 * id - i], mx - i) : 1;
11      while (str[i + p[i]] == str[i - p[i]]) ++p[i];
12      if (mx < i + p[i]) id = i, mx = i + p[i];
13    }
14  }
```

4.6   回文树

- $ch[chr][x]$: x 两边添加字符 chr 后的回文串结点;

- $fail[x]$: x 代表的回文串的最长回文后缀;

- $l[x]$: x 代表的回文串的长度;

- $cnt[x]$: x 代表的回文串的出现次数。

```
1   struct PT {
2     char s[MAXL];
3     int fail[MAXL], ch[26][MAXL], l[MAXL], dep[MAXL], cnt[MAXL], lst, nc, n;
4     void init() {
5       l[0] = 0, l[1] = -1;
6       fail[0] = fail[1] = 1;
7       for (int i = 0; i < 26; ++i)
8         for (int j = 0; j < nc; ++j) ch[i][j] = 0;
9       for (int i = 2; i < nc; ++i) l[i] = 0, fail[i] = 0;
10
11      lst = 0, nc = 2, n = 0, s[0] = '#';
12    }
13
14    int insert(char c) {
15      int id = c - 'a';
16      s[++n] = c;
17      while (s[n - l[lst] - 1] != s[n]) { lst = fail[lst]; }
18      if (ch[id][lst] == 0) {
19        l[nc] = l[lst] + 2;
20        int f = fail[lst];
21        while (s[n - l[f] - 1] != s[n]) { f = fail[f]; }
```

```
22      fail[nc] = ch[id][f];
23      dep[nc] = dep[fail[nc]] + 1;
24      ch[id][lst] = nc;
25      ++nc;
26    }
27    ++cnt[lst = ch[id][lst]];
28    return lst;
29  }
30
31  void count() { for (int i = nc - 1; ~i; --i) cnt[fail[i]] += cnt[i]; }
32 } pt;
33
34 // 求最长双回文串
35 char S[MAXL];
36 int len[MAXL];
37 int main() {
38   ios::sync_with_stdio(false);
39   cin.tie(0);
40   cout.tie(0);
41
42   cin >> S;
43   int n = strlen(S);
44   pt.init();
45   for (int i = 0; i < n; ++i) { len[i] = pt.l[pt.insert(S[i])]; }
46   pt.init();
47   int ans = 0;
48   for (int i = n - 1; i; --i) {
49     ans = max(ans, len[i - 1] + pt.l[pt.insert(S[i])]);
50   }
51   cout << ans << "\n";
52
53   return 0;
54 }
```

4.7 后缀数组（倍增）

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int MAXN = 1e5 + 3;
5
6  template <int MAX_LENGTH> class SuffixArray {
7  public:
8    int n, sa[MAX_LENGTH], rank[MAX_LENGTH], height[MAX_LENGTH];
9
10   void compute(char *s, int n, int m) {
11     int i, p, w, j, k;
12     this->n = n;
13     if (n == 1) {
14       sa[0] = rank[0] = height[0] = 0;
15       return;
16     }
17     memset(cnt, 0, m * sizeof(int));
18     for (i = 0; i < n; ++i) ++cnt[rank[i] = s[i]];
19     for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
20     for (i = n - 1; ~i; --i) sa[--cnt[rank[i]]] = i;
21     for (w = 1; w < n; w <<= 1, m = p) {
22       for (p = 0, i = n - 1; i >= n - w; --i) id[p++] = i;
23       for (i = 0; i < n; ++i)
24         if (sa[i] >= w) id[p++] = sa[i] - w;
25       memset(cnt, 0, m * sizeof(int));
26       for (i = 0; i < n; ++i) ++cnt[px[i] = rank[id[i]]];
27       for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
28       for (i = n - 1; ~i; --i) sa[--cnt[px[i]]] = id[i];
29       memcpy(old_rank, rank, n * sizeof(int));
30       for (i = p = 1, rank[sa[0]] = 0; i < n; ++i)
```

```
31          rank[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p - 1 : p++;
32      }
33      for (i = 0; i < n; ++i) rank[sa[i]] = i;
34      for (i = k = height[rank[0]] = 0; i < n; height[rank[i++]] = k)
35        if (rank[i])
36          for (k > 0 ? --k : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; ++k) {}
37    }
38
39    void init_st_table(int n) {
40      int lgn = lg[n];
41      for (int i = 0; i < n; ++i) table[0][i] = height[i];
42      for (int i = 1; i <= lgn; ++i)
43        for (int j = 0, l = 1 << (i - 1); j + l < n; ++j)
44          table[i][j] = min(table[i - 1][j], table[i - 1][j + l]);
45    }
46
47    int lcp(int i, int j) {
48      if (i > j) swap(i, j);
49      ++i;
50      int lgl = lg[j - i + 1];
51      return min(table[lgl][i], table[lgl][j - (1 << lgl) + 1]);
52    }
53 private:
54   int table[17][MAX_LENGTH], lg[MAX_LENGTH];
55   int old_rank[MAX_LENGTH], id[MAX_LENGTH], px[MAX_LENGTH], cnt[MAX_LENGTH];
56
57   bool cmp(int x, int y, int w) {
58     return old_rank[x] == old_rank[y] && old_rank[x + w] == old_rank[y + w];
59   }
60 };
61
62 char s[MAXN];
63 SuffixArray<MAXN> sa;
64
65 int main(int argc, char *argv[]) {
66   int n = fread(s, 1, MAXN, stdin);
67   while (s[n - 1] - 97u > 25) --n;
68   for (int i = 0; i < n; ++i) s[i] -= 'a';
69   s[n] = '$';
70   sa.compute(s, n, 26);
71   for (int i = 0; i < n; ++i) printf("%d%c", sa.sa[i] + 1, " \n"[i == n - 1]);
72   for (int i = 1; i < n; ++i) printf("%d%c", sa.height[i], " \n"[i == n - 1]);
73   return 0;
74 }
```

4.8 后缀数组（SAIS）

UOJ 板题最快算法，字符串必须为正数，BUFFER_SIZE 要随 MAX_LENGTH 同步变化，1e6 为 25。

```
1  #include <bits/stdc++.h>
2
3  const int BUFFER_SIZE = 1u << 23 | 1;
4  char buffer[BUFFER_SIZE], *buffer_ptr = buffer;
5  #define alloc(x, type, len)                                            \
6    type *x = (type *)buffer_ptr;                                        \
7    buffer_ptr += (len) * sizeof(type);
8  #define clear_buffer()                                                 \
9    memset(buffer, 0, buffer_ptr - buffer), buffer_ptr = buffer;
10
11 template <int MAX_LENGTH> class SuffixArray {
12 #define L_TYPE true
13 #define S_TYPE false
14 public:
15   int sa[MAX_LENGTH], rank[MAX_LENGTH], height[MAX_LENGTH];
16   void compute(int n, int m, int *s) {
17     sais(n, m, s, sa);
```

```
18        for (int i = 0; i < n; ++i) rank[sa[i]] = i;
19        for (int i = 0, h = 0; i < n; ++i) {
20          if (rank[i]) {
21            int j = sa[rank[i] - 1];
22            while (s[i + h] == s[j + h]) ++h;
23            height[rank[i]] = h;
24          } else {
25            h = 0;
26          }
27          if (h) --h;
28        }
29      }
30
31    private:
32      int l_bucket[MAX_LENGTH], s_bucket[MAX_LENGTH];
33
34      void induce(int n, int m, int *s, bool *type, int *sa, int *bucket,
35                  int *l_bucket, int *s_bucket) {
36        memcpy(l_bucket + 1, bucket, m * sizeof(int));
37        memcpy(s_bucket + 1, bucket + 1, m * sizeof(int));
38        sa[l_bucket[s[n - 1]]++] = n - 1;
39        for (int i = 0; i < n; ++i) {
40          int t = sa[i] - 1;
41          if (t >= 0 && type[t] == L_TYPE) sa[l_bucket[s[t]]++] = t;
42        }
43        for (int i = n - 1; i >= 0; --i) {
44          int t = sa[i] - 1;
45          if (t >= 0 && type[t] == S_TYPE) sa[--s_bucket[s[t]]] = t;
46        }
47      }
48      void sais(int n, int m, int *s, int *sa) {
49        alloc(type, bool, n + 1);
50        alloc(bucket, int, m + 1);
51        type[n] = false;
52        for (int i = n - 1; i >= 0; --i) {
53          ++bucket[s[i]];
54          type[i] = s[i] > s[i + 1] || (s[i] == s[i + 1] && type[i + 1] == L_TYPE);
55        }
56        for (int i = 1; i <= m; ++i) {
57          bucket[i] += bucket[i - 1];
58          s_bucket[i] = bucket[i];
59        }
60        memset(rank, -1, n * sizeof(int));
61
62        alloc(lms, int, n + 1);
63        int n1 = 0;
64        for (int i = 0; i < n; ++i) {
65          if (!type[i] && (i == 0 || type[i - 1])) lms[rank[i] = n1++] = i;
66        }
67        lms[n1] = n;
68        memset(sa, -1, n * sizeof(int));
69        for (int i = 0; i < n1; ++i) sa[--s_bucket[s[lms[i]]]] = lms[i];
70        induce(n, m, s, type, sa, bucket, l_bucket, s_bucket);
71        int m1 = 0;
72        alloc(s1, int, n + 1);
73        for (int i = 0, t = -1; i < n; ++i) {
74          int r = rank[sa[i]];
75          if (r != -1) {
76            int len = lms[r + 1] - sa[i] + 1;
77            m1 += t == -1 || len != lms[rank[t] + 1] - t + 1 ||
78                  memcmp(s + t, s + sa[i], len * sizeof(int)) != 0;
79            s1[r] = m1;
80            t = sa[i];
81          }
82        }
83        alloc(sa1, int, n + 1);
84        if (n1 == m1) {
85          for (int i = 0; i < n1; ++i) sa1[s1[i] - 1] = i;
```

```
86        } else {
87          sais(n1, m1, s1, sa1);
88        }
89        memset(sa, -1, n * sizeof(int));
90        memcpy(s_bucket + 1, bucket + 1, m * sizeof(int));
91        for (int i = n1 - 1; i >= 0; --i) {
92          int t = lms[sa1[i]];
93          sa[--s_bucket[s[t]]] = t;
94        }
95        induce(n, m, s, type, sa, bucket, l_bucket, s_bucket);
96      }
97  #undef S_TYPE
98  #undef L_TYPE
99  };
100
101 const int MAXN = 1e5 + 5;
102 SuffixArray<MAXN> sa;
103 char str[MAXN];
104 int s[MAXN];
105
106 int main() {
107   int n = fread(str, 1, MAXN, stdin);
108   while (str[n - 1] - 97u > 25) --n;
109   for (int i = 0; i < n; ++i) s[i] = str[i] - 'a' + 1;
110   sa.compute(n, 26, s);
111   for (int i = 0; i < n; ++i) printf("%d%c", sa.sa[i] + 1, " \n"[i == n - 1]);
112   for (int i = 1; i < n; ++i) printf("%d%c", sa.height[i], " \n"[i == n - 1]);
113   return 0;
114 }
```

4.9 后缀自动机

SPOJ Lexicographical Substring Search：求字典序第 k 大子串

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int MAXN = 90000 + 3;
5  const int ALPHABET = 26;
6
7  struct Node {
8    int len, cnt;
9    Node *link, *next[ALPHABET];
10   void init(int len = 0) {
11     link = nullptr;
12     this->len = len, cnt = 0;
13     memset(next, 0, sizeof(next));
14   }
15 };
16
17 template <int MAX_LENGTH> class SAM {
18 public:
19   Node *last, *root;
20
21   void init() {
22     pool_ptr = pool;
23     last = root = new_node(0);
24   }
25
26   void extend(int chr) {
27     Node *p = last, *np = new_node(p->len + 1);
28     for (last = np; p && !p->next[chr]; p = p->link) p->next[chr] = np;
29     if (!p) {
30       np->link = root;
31     } else {
32       Node *q = p->next[chr];
33       if (q->len == p->len + 1) {
```

```cpp
34          np->link = q;
35        } else {
36          Node *nq = new_node(p->len + 1);
37          memcpy(nq->next, q->next, sizeof(q->next));
38          nq->link = q->link, q->link = np->link = nq;
39          for (; p && p->next[chr] == q; p = p->link) p->next[chr] = nq;
40        }
41      }
42    }
43
44    void toposort() {
45      int size = pool_ptr - pool;
46      memset(cnt, 0, size * sizeof(int));
47      for (Node *it = pool; it < pool_ptr; ++it) ++cnt[it->len];
48      for (int i = 1; i < size; ++i) cnt[i] += cnt[i - 1];
49      for (Node *it = pool; it < pool_ptr; ++it) order[--cnt[it->len]] = it;
50      for (int i = size - 1; ~i; --i) {
51        Node *u = order[i];
52        for (int j = 0; j < ALPHABET; ++j)
53          u->cnt += u->next[j] ? u->next[j]->cnt + 1 : 0;
54      }
55    }
56
57    void find_kth(int k, char *str) {
58      char *ptr = str;
59      Node *u = root;
60      while (k) {
61        for (int j = 0; j < ALPHABET; ++j) {
62          if (!u->next[j]) continue;
63          if (u->next[j]->cnt + 1 < k) {
64            k -= u->next[j]->cnt + 1;
65            continue;
66          }
67          --k, *ptr++ = j + 'a';
68          u = u->next[j];
69          break;
70        }
71      }
72      *ptr = 0;
73    }
74
75  private:
76    int cnt[MAX_LENGTH * 2];
77    Node pool[MAX_LENGTH * 2], *pool_ptr, *order[MAX_LENGTH * 2];
78
79    Node *new_node(int len) {
80      pool_ptr->init(len);
81      return pool_ptr++;
82    }
83  };
84
85  SAM<MAXN> sam;
86  char str[MAXN];
87
88  int main(int argc, char *argv[]) {
89    ios::sync_with_stdio(false);
90    cin.tie(nullptr), cout.tie(nullptr);
91
92    cin >> str;
93    sam.init();
94    for (char *it = str; *it; ++it) sam.extend(*it - 'a');
95    sam.toposort();
96
97    int q, k;
98    for (cin >> q; q; --q) cin >> k, sam.find_kth(k, str), puts(str);
99
100    return 0;
101  }
```

5 图论

5.1 Tarjan

对于无向图求边双连通，则添加两条有向边；若有重边，则在 $v == from$ 处添加计数器。

```cpp
vector<int> adj[MAXN];
bitset<MAXN> instk, cut;
int bridges, dfs_clk, top, scc, n, m, d;
int dfn[MAXN], stk[MAXN], bel[MAXN], sz[MAXN];

int tarjan(int u, int from) {
  int low = dfn[u] = ++dfs_clk;
  stk[top++] = u, instk[u] = 1;

  int son = 0;
  for (int v : adj[u]) {
    if (v == from) continue;
    if (!dfn[v]) {
      ++son;
      int low_v = tarjan(v, u);
      (low_v < low) && (low = low_v);
      (low_v > dfn[u]) && (++bridges);
      (u != from && low_v >= dfn[u]) && (cut[u] = 1);
    } else if (instk[v] && low > dfn[v]) {
      low = dfn[v];
    }
  }
  (u == from && son > 1) && (cut[u] = 1);

  if (low == dfn[u]) {
    int v, sz = 0;
    sz[++scc] = 0;
    do {
      ++sz;
      v = stk[--top];
      instk[v] = 0, bel[v] = scc;
    } while (u ^ v);
  }
  return low;
}
```

5.2 Hopcroft 算法

pos 表示左边的点匹配右边哪一个，neg 反之，时间复杂度 $O(m\sqrt{n})$。

```cpp
vector<int> adj[MAXN];
int nl, nr, pos[MAXN], neg[MAXN], lx[MAXN], ly[MAXN];

bool dfs(int x) {
  int c = lx[x] + 1;
  lx[x] = -1;
  for (int y : adj[x]) {
    if (ly[y] != c) continue;
    ly[y] = -1;
    if (~neg[y] && !dfs(neg[y])) continue;
    pos[neg[y] = x] = y;
    return true;
  }
  return false;
}

int match() {
  int cnt = 0;
  memset(pos, -1, sizeof(int) * nl);
  memset(neg, -1, sizeof(int) * nr);
```

```
21    for (int x = 0; x < nl; ++x)
22      for (int y : adj[x]) {
23        if (~neg[y]) continue;
24        pos[neg[y] = x] = y, ++cnt;
25        break;
26      }
27    for (;;) {
28      static int q[MAXN];
29      int l = 0, r = 0, ok = 0;
30      memset(lx, -1, sizeof(int) * nl);
31      memset(ly, -1, sizeof(int) * nr);
32      for (int x = 0; x < nl; ++x)
33        if (pos[x] < 0) lx[q[r++] = x] = 0;
34      while (l < r) {
35        int x = q[l++];
36        for (int y : adj[x]) {
37          if (~ly[y]) continue;
38          ly[y] = lx[x] + 1;
39          if (~neg[y] && ~lx[neg[y]]) continue;
40          (~neg[y]) ? lx[q[r++] = neg[y]] = ly[y] + 1 : ok = 1;
41        }
42      }
43      if (!ok) return cnt;
44      for (int x = 0; x < nl; ++x)
45        if (pos[x] < 0 && dfs(x)) ++cnt;
46    }
47  }
```

5.3 KM 算法

点为 1..n（左为 1..nl，右为 1..nr），lk 表示左边的点匹配右边哪一个。
最大费用流时 NOT = 0，最大费用流最大流时 NOT = -1ll * MAXN * ALPHA。

```
1  const int MAXN = 400 + 3;
2  const int ALPHA = 1e9 + 10;
3  const ll NOT = 0;
4  const ll INF = 3ll * MAXN * ALPHA;
5  struct KM {
6    int n, nl, nr, lk[MAXN], pre[MAXN];
7    ll lx[MAXN], ly[MAXN], w[MAXN][MAXN], slack[MAXN];
8    bitset<MAXN> vy;
9
10   void init(int n) {
11     this->n = n;
12     memset(lk, 0, sizeof(int) * (n + 1));
13     memset(pre, 0, sizeof(int) * (n + 1));
14     memset(lx, 0, sizeof(ll) * (n + 1));
15     memset(ly, 0, sizeof(ll) * (n + 1));
16     memset(slack, 0, sizeof(ll) * (n + 1));
17     for (int i = 0; i <= n; ++i) fill(w[i], w[i] + n + 1, NOT);
18   }
19
20   void add_edge(int x, int y, ll z) {
21     if (w[y][x] < z) w[y][x] = z;
22   }
23
24   ll match() {
25     for (int i = 1; i <= n; ++i)
26       for (int j = 1; j <= n; ++j) lx[i] = max(lx[i], w[i][j]);
27     for (int i = 1, py, p, x; i <= n; ++i) {
28       for (int j = 1; j <= n; ++j) slack[j] = INF, vy[j] = 0;
29       for (lk[py = 0] = i; lk[py]; py = p) {
30         ll delta = INF;
31         vy[py] = 1, x = lk[py];
32         for (int y = 1; y <= n; ++y) {
33           if (vy[y]) continue;
34           if (lx[x] + ly[y] - w[x][y] < slack[y])
```

```
35          slack[y] = lx[x] + ly[y] - w[x][y], pre[y] = py;
36          if (slack[y] < delta) delta = slack[y], p = y;
37        }
38        for (int y = 0; y <= n; ++y)
39          if (vy[y]) {
40            lx[lk[y]] -= delta, ly[y] += delta;
41          } else {
42            slack[y] -= delta;
43          }
44      }
45      for (; py; py = pre[py]) lk[py] = lk[pre[py]];
46    }
47
48    ll ans = 0;
49    for (int i = 1; i <= n; ++i) {
50      ans += lx[i] + ly[i];
51      if (w[lk[i]][i] == NOT) ans -= NOT;
52    }
53    return ans;
54  }
55 } km;
56
57 int main() {
58   int nl, nr, m;
59   cin >> nl >> nr >> m;
60   km.init(max(nl, nr));
61   for (int x, y, z; m; --m) {
62     cin >> x >> y >> z;
63     km.add_edge(x, y, z);
64   }
65   cout << km.match() << "\n";
66   for (int i = 1; i <= nl; ++i)
67     cout << (km.w[km.lk[i]][i] == NOT ? 0 : km.lk[i]) << " \n"[i == nl];
68
69   return 0;
70 }
```

5.4　一般图最大匹配

```
1  class GeneralMatch {
2  public:
3    int n;
4    vector<vector<int>> g;
5    vector<int> match, aux, label, orig, parent;
6    queue<int> q;
7    int aux_time;
8
9    GeneralMatch(int n)
10       : match(n, -1), aux(n, -1), label(n), orig(n), parent(n, -1),
11         aux_time(-1) {
12     this->n = n;
13     g.resize(n);
14   }
15
16   void add_edge(int u, int v) {
17     g[u].push_back(v);
18     g[v].push_back(u);
19   }
20
21   int find(int x) { return x == orig[x] ? x : orig[x] = find(orig[x]); }
22
23   int lca(int u, int v) {
24     ++aux_time;
25     u = find(u), v = find(v);
26     for (;; swap(u, v)) {
27       if (~u) {
```

```
28          if (aux[u] == aux_time) return u;
29          aux[u] = aux_time;
30          if (match[u] == -1) {
31            u = -1;
32          } else {
33            u = find(parent[match[u]]);
34          }
35        }
36      }
37    }
38
39    void blossom(int u, int v, int o) {
40      while (find(u) != o) {
41        parent[u] = v;
42        v = match[u];
43        q.push(v);
44        label[v] = 0;
45        orig[u] = orig[v] = o;
46        u = parent[v];
47      }
48    }
49
50    int bfs(int x) {
51      iota(orig.begin(), orig.end(), 0);
52      fill(label.begin(), label.end(), -1);
53      while (!q.empty()) q.pop();
54      q.push(x);
55      label[x] = 0;
56      while (!q.empty()) {
57        int u = q.front();
58        q.pop();
59        for (int v : g[u]) {
60          if (label[v] == -1) {
61            parent[v] = u;
62            label[v] = 1;
63            if (match[v] == -1) {
64              while (v != -1) {
65                int pv = parent[v];
66                int next_v = match[pv];
67                match[v] = pv;
68                match[pv] = v;
69                v = next_v;
70              }
71              return 1;
72            }
73            q.push(match[v]);
74            label[match[v]] = 0;
75          } else if (label[v] == 0 && find(u) != find(v)) {
76            int o = lca(u, v);
77            blossom(u, v, o);
78            blossom(v, u, o);
79          }
80        }
81      }
82      return 0;
83    }
84
85    int find_max_match() {
86      int res = 0;
87      for (int i = 0; i < n; ++i) {
88        if (~match[i]) continue;
89        res += bfs(i);
90      }
91      return res;
92    }
93  };
```

5.5 SAP

```
1  struct MaxFlow {
2    struct Edge {
3      int to, rest;
4    } edges[MAXM * 4];
5
6    vector<int> adj[MAXN];
7    int n, edges_c, dep[MAXN], depc[MAXN], s, t, last[MAXN];
8
9    void init(int _n) {
10     n = _n, edges_c = 0;
11     for (int i = 1; i <= n; ++i) adj[i].clear();
12   }
13
14   void add_edge(int u, int v, int cap) {
15     edges[edges_c] = {v, cap, 0};
16     adj[u].push_back(edges_c++);
17     edges[edges_c] = {u, 0, 0};
18     adj[v].push_back(edges_c++);
19   }
20
21   int dfs(int u, int flow) {
22     if (u == t || !flow) return flow;
23     int v, e, temp, res = 0;
24     for (int &i = last[u]; i < (int)adj[u].size(); ++i) {
25       e = adj[u][i], v = edges[e].to;
26       if (!edges[e].res || dep[v] != dep[u] - 1) continue;
27       temp = dfs(v, min(flow, edges[e].cap - edges[e].flow));
28       res += temp, flow -= temp;
29       edges[e].rest -= temp, edges[e ^ 1].rest += temp;
30       if (!flow || !dep[s]) return res;
31     }
32     last[u] = 0;
33     if (!(--depc[dep[u]])) dep[s] = n + 1;
34     ++depc[++dep[u]];
35     return res;
36   }
37   int max_flow(int s, int t) {
38     this->s = s, this->t = t;
39
40     static queue<int> que;
41     memset(dep + 1, 0, sizeof(int) * n);
42     memset(depc + 1, 0, sizeof(int) * n);
43     memset(last + 1, 0, sizeof(int) * n);
44     while (!que.empty()) que.pop();
45     dep[t] = 1, que.push(t);
46
47     while (!que.empty()) {
48       int u = que.front();
49       que.pop();
50       ++depc[dep[u]];
51       for (int i = 0, v; i < (int)adj[u].size(); ++i) {
52         v = edges[adj[u][i]].to;
53         if (dep[v]) continue;
54         dep[v] = dep[u] + 1;
55         que.push(v);
56       }
57     }
58
59     int res = 0;
60     while (dep[s] <= n) res += dfs(s, INT_MAX);
61     return res;
62   }
63 };
```

5.6   dinic

```cpp
struct MaxFlow {
  struct Edge {
    int to, rest;
  } edges[MAXM * 4];

  vector<int> adj[MAXN];
  int n, edges_c, dep[MAXN], s, t, last[MAXN];

  void init(int _n) {
    n = _n, edges_c = 0;
    for (int i = 1; i <= n; ++i) adj[i].clear();
  }

  void add_edge(int u, int v, int cap) {
    edges[edges_c] = {v, cap, 0};
    adj[u].push_back(edges_c++);
    edges[edges_c] = {u, 0, 0};
    adj[v].push_back(edges_c++);
  }

  bool bfs() {
    memset(dep + 1, -1, sizeof(int) * n);
    static queue<int> q;
    q.push(s);
    dep[s] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int i = 0; i < adj[u].size(); ++i) {
        Edge &e = edges[adj[u][i]];
        if ((~dep[e.to]) || !e.rest) continue;
        dep[e.to] = dep[u] + 1;
        q.push(e.to);
      }
    }
    return ~dep[t];
  }

  int dfs(int u, int flow) {
    if (u == t || flow == 0) return flow;
    int res = 0, temp, e, v;
    for (int &i = last[u]; i < adj[u].size(); ++i) {
      e = adj[u][i], v = edges[e].to;
      if (dep[v] == dep[u] + 1 && edges[e].rest) {
        temp = dfs(v, min(edges[e].rest, flow));
        res += temp, flow -= temp;
        edges[e].rest -= temp, edges[e ^ 1].rest += temp;
        if (!flow) break;
      }
    }
    return flow;
  }

  int max_flow(int s, int t) {
    this->s = s, this->t = t;
    int res = 0;
    while (bfs()) {
      memset(last + 1, 0, sizeof(int) * n);
      res += dfs(s, INF);
    }
    return res;
  }
};
```

5.7　高标预流推进

```cpp
const int N = 1e4 + 4, M = 2e5 + 5, INF = 0x3f3f3f3f;
int n, m, s, t;

struct qxx {
  int nex, t, v;
};
qxx e[M * 2];
int h[N], cnt = 1;
void add_path(int f, int t, int v) { e[++cnt] = (qxx){h[f], t, v}, h[f] = cnt; }
void add_flow(int f, int t, int v) {
  add_path(f, t, v);
  add_path(t, f, 0);
}

int ht[N], ex[N], gap[N]; // 高度；超额流；gap 优化
bool bfs_init() {
  memset(ht, 0x3f, sizeof(ht));
  queue<int> q;
  q.push(t), ht[t] = 0;
  while (q.size()) { // 反向 BFS, 遇到没有访问过的结点就入队
    int u = q.front();
    q.pop();
    for (int i = h[u]; i; i = e[i].nex) {
      const int &v = e[i].t;
      if (e[i ^ 1].v && ht[v] > ht[u] + 1) ht[v] = ht[u] + 1, q.push(v);
    }
  }
  return ht[s] != INF; // 如果图不连通, 返回 0
}
struct cmp {
  bool operator()(int a, int b) const { return ht[a] < ht[b]; }
};                                     // 伪装排序函数
priority_queue<int, vector<int>, cmp> pq; // 将需要推送的结点以高度高的优先
bool vis[N];                           // 是否在优先队列中
int push(int u) { // 尽可能通过能够推送的边推送超额流
  for (int i = h[u]; i; i = e[i].nex) {
    const int &v = e[i].t, &w = e[i].v;
    if (!w || ht[u] != ht[v] + 1) continue;
    int k = min(w, ex[u]); // 取到剩余容量和超额流的最小值
    ex[u] -= k, ex[v] += k, e[i].v -= k, e[i ^ 1].v += k; // push
    if (v != s && v != t && !vis[v])
      pq.push(v), vis[v] = 1; // 推送之后, v 必然溢出, 则入堆, 等待被推送
    if (!ex[u]) return 0; // 如果已经推送完就返回
  }
  return 1;
}
void relabel(int u) { // 重贴标签（高度）
  ht[u] = INF;
  for (int i = h[u]; i; i = e[i].nex)
    if (e[i].v) ht[u] = min(ht[u], ht[e[i].t]);
  ++ht[u];
}
int hlpp() {                     // 返回最大流
  if (!bfs_init()) return 0; // 图不连通
  ht[s] = n;
  memset(gap, 0, sizeof(gap));
  for (int i = 1; i <= n; i++)
    if (ht[i] != INF) gap[ht[i]]++; // 初始化 gap
  for (int i = h[s]; i; i = e[i].nex) {
    const int v = e[i].t, w = e[i].v; // 队列初始化
    if (!w) continue;
    ex[s] -= w, ex[v] += w, e[i].v -= w, e[i ^ 1].v += w; // 注意取消 w 的引用
    if (v != s && v != t && !vis[v]) pq.push(v), vis[v] = 1; // 入队
  }
  while (pq.size()) {
    int u = pq.top();
```

```
67      pq.pop(), vis[u] = 0;
68      while (push(u)) { // 仍然溢出
69        // 如果 u 结点原来所在的高度没有结点了, 相当于出现断层
70        if (!--gap[ht[u]])
71          for (int i = 1; i <= n; i++)
72            if (i != s && i != t && ht[i] > ht[u] && ht[i] < n + 1) ht[i] = n + 1;
73        relabel(u);
74        ++gap[ht[u]]; // 新的高度, 更新 gap
75      }
76    }
77    return ex[t];
78  }
```

5.8 最小费用流

```
 1  class MinCostFlow {
 2  public:
 3    struct Result {
 4      int flow, cost;
 5    };
 6    struct Edge {
 7      int to, next, rest, cost;
 8    };
 9
10    vector<bool> inq;
11    vector<int> head, dist, from, flow;
12    vector<Edge> edges;
13
14    MinCostFlow(int n, int m) : inq(n), head(n, -1), dist(n), from(n), flow(n) {
15      edges.reserve(2 * m);
16    }
17
18    void add_edge(int u, int v, int capacity, int cost) {
19      internal_add_edge(u, v, capacity, cost);
20      internal_add_edge(v, u, 0, -cost);
21    }
22
23    void internal_add_edge(int u, int v, int capacity, int cost) {
24      edges.push_back((Edge){v, head[u], capacity, cost});
25      head[u] = edges.size() - 1;
26    }
27
28    Result augment(int source, int sink) {
29      fill(dist.begin(), dist.end(), INT_MAX);
30      dist[source] = 0;
31      flow[source] = INT_MAX;
32      queue<int> q;
33      q.push(source);
34      while (!q.empty()) {
35        int u = q.front();
36        q.pop();
37        inq[u] = false;
38        for (int it = head[u]; ~it; it = edges[it].next) {
39          auto &e = edges[it];
40          int v = e.to;
41          if (e.rest > 0 && dist[u] + e.cost < dist[v]) {
42            from[v] = it;
43            dist[v] = dist[u] + e.cost;
44            flow[v] = min(e.rest, flow[u]);
45            if (!inq[v]) {
46              q.push(v);
47              inq[v] = true;
48            }
49          }
50        }
51      }
```

```
52
53        if (dist[sink] == INT_MAX) return {0, 0};
54        int min_flow = flow[sink];
55        for (int u = sink; u != source; u = edges[from[u] ^ 1].to) {
56          edges[from[u]].rest -= min_flow;
57          edges[from[u] ^ 1].rest += min_flow;
58        }
59        return {min_flow, dist[sink]};
60      }
61
62      Result min_cost_flow(int source, int sink) {
63        int flow = 0, cost = 0;
64        for (;;) {
65          auto result = augment(source, sink);
66          if (!result.flow) break;
67          flow += result.flow, cost += result.cost;
68        }
69        return {flow, cost};
70      }
71    };
```

5.9　上下界费用流

```
1    const int MAXN = 53;
2    const int MAX_NODE = 113;
3    const int MAX_EDGE = 1e5 + 5;
4    const int INF = 0x3f3f3f3f;
5
6    int n, s, t, ss, tt, tote;
7    int R[MAXN], C[MAXN], board[MAXN][MAXN];
8
9    struct Edge {
10     int to, cap, flow, cost;
11   } edges[MAX_EDGE];
12   vector<int> adj[MAX_NODE];
13
14   int from[MAX_NODE], in[MAX_NODE];
15   void add_edge(int from, int to, int l, int r, int cost) {
16     in[to] += l, in[from] -= l;
17     edges[tote] = (Edge){to, r - l, 0, cost};
18     adj[from].push_back(tote++);
19     edges[tote] = (Edge){from, 0, 0, -cost};
20     adj[to].push_back(tote++);
21   }
22
23   bool spfa(int s, int t) {
24     static queue<int> q;
25     static bool inq[MAX_NODE];
26     static int dist[MAX_NODE];
27     memset(inq + 1, 0, sizeof(bool) * tt);
28     memset(dist + 1, 0x3f, sizeof(int) * tt);
29     memset(from + 1, -1, sizeof(int) * tt);
30     dist[0] = 0, from[0] = -1;
31     q.push(0);
32     while (!q.empty()) {
33       int u = q.front();
34       q.pop();
35       inq[u] = false;
36       for (int e : adj[u]) {
37         if (edges[e].cap == edges[e].flow) continue;
38         int v = edges[e].to, d = dist[u] + edges[e].cost;
39         if (d >= dist[v]) continue;
40         dist[v] = d;
41         from[v] = e;
42         if (!inq[v]) {
43           q.push(v);
```

```
44          inq[v] = true;
45        }
46      }
47    }
48    return dist[t] < INF;
49  }
50
51  pair<int, int> min_cost_max_flow(int s, int t) {
52    int flow = 0, cost = 0;
53    while (spfa(s, t)) {
54      int mi = INF;
55      for (int it = from[t]; ~it; it = from[edges[it ^ 1].to])
56        mi = min(mi, edges[it].cap - edges[it].flow);
57      flow += mi;
58      for (int it = from[t]; ~it; it = from[edges[it ^ 1].to]) {
59        edges[it].flow += mi, edges[it ^ 1].flow -= mi;
60        cost += mi * edges[it].cost;
61      }
62    }
63    return make_pair(flow, cost);
64  }
65
66  void solve() {
67    tote = 0;
68    s = 2 * n + 1, t = 2 * n + 2, ss = 0, tt = 2 * n + 3;
69    for (int i = 0; i <= tt; ++i) adj[i].clear(), in[i] = 0;
70
71    memset(R + 1, 0, sizeof(int) * n);
72    memset(C + 1, 0, sizeof(int) * n);
73
74    for (int i = 1; i <= n; ++i)
75      for (int j = 1; j <= n; ++j) {
76        cin >> board[i][j];
77        R[i] += board[i][j];
78        C[j] += board[i][j];
79      }
80
81    for (int i = 1; i <= n; ++i) {
82      add_edge(s, i, R[i], R[i], 0);
83      add_edge(s, i + n, C[i], C[i], 0);
84    }
85
86    for (int i = 1, l, r; i <= n; ++i) {
87      cin >> l >> r;
88      add_edge(i, t, l, r, 0);
89    }
90    for (int i = 1, l, r; i <= n; ++i) {
91      cin >> l >> r;
92      add_edge(i + n, t, l, r, 0);
93    }
94
95    for (int step = n * n / 2, x1, y1, x2, y2; step; --step) {
96      cin >> x1 >> y1 >> x2 >> y2;
97      if (board[x1][y1] == board[x2][y2]) continue;
98      if (board[x2][y2]) swap(x1, x2), swap(y1, y2);
99      if (x1 == x2)
100         add_edge(y1 + n, y2 + n, 0, 1, 1);
101     else
102         add_edge(x1, x2, 0, 1, 1);
103   }
104   add_edge(t, s, 0, INF, 0);
105   int sum = 0;
106   for (int i = 1; i < tt; ++i) {
107     if (in[i] > 0) {
108       sum += in[i];
109       add_edge(ss, i, 0, in[i], 0);
110     } else if (in[i] < 0) {
111       add_edge(i, tt, 0, -in[i], 0);
```

```
112        }
113      }
114
115      pair<int, int> ans = min_cost_max_flow(ss, tt);
116      if (sum != ans.first) {
117        cout << "-1\n";
118      } else {
119        cout << ans.second << '\n';
120      }
121    }
```

6 计算几何

7 Java

7.1 进制转换

```
1    import java.io.*;
2    import java.util.*;
3    import java.math.*;
4
5    public class Main {
6      public static void main(String[] args) {
7        InputStream inputStream = System.in;
8        OutputStream outputStream = System.out;
9        Scanner in = new Scanner(inputStream);
10       PrintWriter out = new PrintWriter(outputStream);
11       Solver solver = new Solver();
12       int testCount = Integer.parseInt(in.next());
13       for (int i = 1; i <= testCount; i++)
14         solver.solve(i, in, out);
15       out.close();
16     }
17
18     static class Solver {
19       public void solve(int testNumber, Scanner in, PrintWriter out) {
20         int a = in.nextInt();
21         int b = in.nextInt();
22         String num = in.next();
23
24         BigInteger value = BigInteger.ZERO;
25         for (int i = 0; i < num.length(); ++i) {
26           value = value.multiply(BigInteger.valueOf(a));
27           value = BigInteger.valueOf(getValue(num.charAt(i))).add(value);
28         }
29         out.println(a + " " + num);
30
31         if (value.equals(BigInteger.ZERO)) {
32           out.println(b + " 0");
33           out.println();
34           return;
35         }
36
37         out.print(b + " ");
38
39         char[] ans = new char[1000];
40         int length = 0;
41         while (!value.equals(BigInteger.ZERO)) {
42           int digit = value.mod(BigInteger.valueOf(b)).intValue();
43           value = value.divide(BigInteger.valueOf(b));
44           ans[length] = getChar(digit);
45           ++length;
46         }
47
```

```
48        for (int i = length - 1; i >= 0; --i) {
49          out.print(ans[i]);
50        }
51        out.println("\n");
52      }
53
54      private int getValue(char ch) {
55        if (ch >= 'A' && ch <= 'Z') {
56          return ch - 'A' + 10;
57        }
58        if (ch >= 'a' && ch <= 'z') {
59          return ch - 'a' + 36;
60        }
61        return ch - '0';
62      }
63
64      private char getChar(int x) {
65        if (x < 10) {
66          return (char) ('0' + x);
67        } else if (x < 36) {
68          return (char) ('A' + x - 10);
69        } else {
70          return (char) ('a' + x - 36);
71        }
72      }
73    }
74 }
```

8   杂项

8.1   I/O 优化

```
1  namespace FastIO {
2  struct Control {
3    int ct, val;
4    Control(int Ct, int Val = -1) : ct(Ct), val(Val) {}
5    inline Control operator()(int Val) { return Control(ct, Val); }
6  } _endl(0), _prs(1), _setprecision(2);
7
8  const int IO_SIZE = 1 << 16 | 127;
9
10 struct FastIO {
11   char in[IO_SIZE], *p, *pp, out[IO_SIZE], *q, *qq, ch[20], *t, b, K, prs;
12   FastIO() : p(in), pp(in), q(out), qq(out + IO_SIZE), t(ch), b(1), K(6) {}
13   ~FastIO() { fwrite(out, 1, q - out, stdout); }
14   inline char getc() {
15     return p == pp && (pp = (p = in) + fread(in, 1, IO_SIZE, stdin), p == pp)
16                ? (b = 0, EOF)
17                : *p++;
18   }
19   inline void putc(char x) {
20     q == qq && (fwrite(out, 1, q - out, stdout), q = out), *q++ = x;
21   }
22   inline void puts(const char str[]) {
23     fwrite(out, 1, q - out, stdout), fwrite(str, 1, strlen(str), stdout),
24         q = out;
25   }
26   inline void getline(string &s) {
27     s = "";
28     for (char ch; (ch = getc()) != '\n' && b;) s += ch;
29   }
30 #define indef(T)                                                 \
31   inline FastIO &operator>>(T &x) {                              \
32     x = 0;                                                       \
33     char f = 0, ch;                                              \
```

```
34        while (!isdigit(ch = getc()) && b) f |= ch == '-';              \
35        while (isdigit(ch)) x = (x << 1) + (x << 3) + (ch ^ 48), ch = getc();  \
36        return x = f ? -x : x, *this;                                   \
37      }
38      indef(int);
39      indef(long long);
40
41      inline FastIO &operator>>(string &s) {
42        s = "";
43        char ch;
44        while (isspace(ch = getc()) && b) {}
45        while (!isspace(ch) && b) s += ch, ch = getc();
46        return *this;
47      }
48      inline FastIO &operator>>(double &x) {
49        x = 0;
50        char f = 0, ch;
51        double d = 0.1;
52        while (!isdigit(ch = getc()) && b) f |= (ch == '-');
53        while (isdigit(ch)) x = x * 10 + (ch ^ 48), ch = getc();
54        if (ch == '.')
55          while (isdigit(ch = getc())) x += d * (ch ^ 48), d *= 0.1;
56        return x = f ? -x : x, *this;
57      }
58  #define outdef(_T)                                                    \
59      inline FastIO &operator<<(_T x) {                                 \
60        !x && (putc('0'), 0), x < 0 && (putc('-'), x = -x);             \
61        while (x) *t++ = x % 10 + 48, x /= 10;                          \
62        while (t != ch) *q++ = *--t;                                    \
63        return *this;                                                   \
64      }
65      outdef(int);
66      outdef(long long);
67      inline FastIO &operator<<(char ch) { return putc(ch), *this; }
68      inline FastIO &operator<<(const char str[]) { return puts(str), *this; }
69      inline FastIO &operator<<(const string &s) { return puts(s.c_str()), *this; }
70      inline FastIO &operator<<(double x) {
71        int k = 0;
72        this->operator<<(int(x));
73        putc('.');
74        x -= int(x);
75        prs && (x += 5 * pow(10, -K - 1));
76        while (k < K) putc(int(x *= 10) ^ 48), x -= int(x), ++k;
77        return *this;
78      }
79      inline FastIO &operator<<(const Control &cl) {
80        switch (cl.ct) {
81        case 0: putc('\n'); break;
82        case 1: prs = cl.val; break;
83        case 2: K = cl.val; break;
84        }
85        return *this;
86      }
87      inline operator bool() { return b; }
88  };
89  } // namespace FastIO
```

8.2 优化 STL 内存申请

```
1  // useage: vector<int, myalloc<int>> L;
2  static char space[10000000], *sp = space;
3  template <typename T> struct myalloc : allocator<T> {
4    myalloc() {}
5    template <typename T2> myalloc(const myalloc<T2> &a) {}
6    template <typename T2> myalloc<T> &operator=(const myalloc<T2> &a) {
7      return *this;
```

```
 8     }
 9    template <typename T2> struct rebind { typedef myalloc<T2> other; };
10    inline T *allocate(size_t n) {
11      T *result = (T *)sp;
12      sp += n * sizeof(T);
13      return result;
14    }
15    inline void deallocate(T *p, size_t n) {}
16  };
```

8.3 Emacs 配置

```
 1  (defun myc++ ()
 2    (c-set-style "stroustrup")
 3    (setq tab-width 2)
 4    (setq indent-tabs-mode nil)
 5    (setq c-basic-offset 2)
 6    (c-toggle-hungry-state)
 7    (defun compile-and-run()
 8      (interactive)
 9      (setq file-name (file-name-sans-extension (file-name-nondirectory buffer-file-name)))
10      (compile
11       (format "g++ %s.cpp -o %s -Wall -Wextra -Wshadow -O2 && ./%s < in.txt"
12               file-name file-name file-name)))
13    (local-set-key (kbd "C-c C-c") 'compile-and-run)
14    (local-set-key (kbd "C-c C-k") 'kill-compilation))
15  (add-hook 'c++-mode-hook 'myc++)
16
17  (global-set-key [(meta ?o)] 'other-window)
18  (global-set-key [(meta ?/)] 'hippie-expand)
19  (global-set-key [(control tab)] ' senator-completion-menu-popup)
20  (setq hippie-expand-try-functions-list
21        '(try-expand-dabbrev
22          try-expand-dabbrev-visible
23          try-expand-dabbrev-all-buffers
24          try-expand-dabbrev-from-kill
25          try-complete-file-name-partially
26          try-complete-file-name
27          try-expand-all-abbrevs
28          try-expand-list
29          try-expand-line))
30
31  (setq auto-save-mode nil)
32  (setq make-backup-files nil)
33
34  (ido-mode t)
35  (show-paren-mode 1)
36  (delete-selection-mode t)
37  (global-linum-mode t)
38  (global-auto-revert-mode t)
```

8.4 Vim 配置

```
 1  syntax on
 2  set tabstop=2 softtabstop=2 shiftwidth=2 expandtab smarttab autoindent
 3  set incsearch ignorecase smartcase hlsearch
 4  set ruler laststatus=2 showcmd showmode
 5  set nobackup noswapfile noundofile autowrite noerrorbells
 6  set wrap breakindent encoding=utf-8 number title cursorline
 7
 8  autocmd filetype c nnoremap <F5> :w <bar> exec '!gcc '.shellescape('%').' -o '.shellescape('%:r').
        ' -std=c11 -Wall && ./'.shellescape('%:r')<CR>
 9  autocmd filetype cpp nnoremap <F5> :w <bar> exec '!g++ '.shellescape('%').' -o '.shellescape('%:r'
        ).' -std=c++17 -Wall && ./'.shellescape('%:r')<CR>
```

```
10  autocmd filetype java nnoremap <F5> :w <bar> exec '!javac -encoding UTF-8 -sourcepath . -d . '.
        shellescape('%').' && java '.shellescape('%:r')<CR>
11
12  autocmd filetype c nnoremap <F6> :w <bar> exec '!gcc '.shellescape('%').' -o '.shellescape('%:r').
        ' -std=c11 -Wall -g && gdb '.shellescape('%:r')<CR>
13  autocmd filetype cpp nnoremap <F6> :w <bar> exec '!g++ '.shellescape('%').' -o '.shellescape('%:r'
        ).' -std=c++17 -Wall -g && gdb '.shellescape('%:r')<CR>
14
15  function! TrimWhitespace()
16      let l:save = winsaveview()
17      %s/\\\@<!\s\+$//e
18      call winrestview(l:save)
19  endfunction
20
21  imap jk <Esc>
22  nmap t :call TrimWhitespace()<CR>
23
24  nmap <M-1> 1gt
25  nmap <M-2> 2gt
26  nmap <M-3> 3gt
27  nmap <M-4> 4gt
28  nmap <M-5> 5gt
29  nmap <M-6> 6gt
30  nmap <M-7> 7gt
31  nmap <M-8> 8gt
32  nmap <M-9> 9gt
33  nmap <M-t> :tabnew<CR>
34  nmap <M-w> :close<CR>
35  nmap <Tab> :tabnext<CR>
36  nmap <S-Tab> :tabprevious<CR>
```

8.5 对拍

*unix 下对拍:

```
1  while true; do
2    python gen.py > in.txt
3    time ./my < in.txt > out.txt
4    time ./std < in.txt > ans.txt
5    if diff out.txt ans.txt; then
6      echo AC
7    else
8      echo WA
9      exit 0
10   fi
11 done
```

Windows 下对拍:

```
1  @echo off
2  :loop
3  python gen.py > in.txt
4  my.exe < in.txt > out.txt
5  std.exe < in.txt > ans.txt
6  fc out.txt ans.txt
7  if not errorlevel 1 goto loop
8  pause
9  :end
```