cycleke（菜鸡）的 XCPC 模板

哈爾濱工業大學

cycleke

February 14, 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:

```
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

```
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：$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)} & \gcd(a,,p) = 1 \\ a^b & \gcd(a,,p) \neq 1, b < \varphi(p) \\ a^{b \bmod \varphi(p) + \varphi(p)} & \gcd(a,,p) \neq 1,, b \geq \varphi(p) \end{cases} \pmod p$

```
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。

```
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

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

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：$S(n) = \sum_{i=1}^{n} id(i) - \sum_{i=2}^{n} S(\lfloor \frac{n}{i} \rfloor)$

```cpp
map<int, int> mp_mu;

int S_mu(int n) {
  if (n < MAXN) return sum_mu[n];
  if (mp_mu[n]) return mp_mu[n];
  int ret = 1;
  for (int i = 2, j; i <= n; i = j + 1) {
    j = n / (n / i);
    ret -= S_mu(n / i) * (j - i + 1);
  }
  return mp_mu[n] = ret;
}

// 使用莫比乌斯反演
ll S_phi(int n) {
  ll res = 0;
  for (int i = 1, j; i <= n; i = j + 1) {
    j = n / (n / i);
    res += 1LL * (S_mu(j) - S_mu(i - 1)) * (n / i) * (n / i);
  }
  return (res - 1) / 2 + 1;
}
```

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。

```cpp
constexpr int MOD = 1e9 + 7;
constexpr int INV2 = MOD / 2 + 1;

template <typename X, typename Y> void inc(X &x, const Y &y) { x += y, (x >= MOD) && (x -= MOD); }
template <typename X, typename Y> void dec(X &x, const Y &y) { x -= y, (x < 0) && (x += MOD); }
template <typename X, typename Y> int sum(X x, Y y) { return x + y < MOD ? x + y : x + y - MOD; }
template <typename X, typename Y> int sub(X x, Y y) { return x < y ? x + MOD - y : x - y;}

constexpr int MAX_SIZE = 2e5 + 3;
int prime[MAX_SIZE / 10], lpf[MAX_SIZE], spri[MAX_SIZE], prime_cnt;

void sieve(int n) {
  for (int i = 2; i <= n; ++i) {
    if (lpf[i] == 0) {
      prime[lpf[i] = ++prime_cnt] = i;
      spri[prime_cnt] = sum(spri[prime_cnt - 1], i);
    }
    for (int j = 1, x; j <= lpf[i] && (x = i * prime[j]) <= n; ++j) lpf[x] = j;
```

```cpp
  }
}

ll g_n, lis[MAX_SIZE];
int G[MAX_SIZE][2], Fprime[MAX_SIZE], cnt;
int lim, le[MAX_SIZE], ge[MAX_SIZE];
#define idx(x) (x <= lim ? le[x] : ge[g_n / x])

void init(ll n) {
  for (ll i = 1, j, x; i <= n; i = n / j + 1) {
    j = n / i, x = j % MOD;
    lis[++cnt] = j, idx(j) = cnt;
    G[cnt][0] = sub(x, 1);
    G[cnt][1] = (x + 2ll) * (x - 1ll) % MOD * INV2 % MOD;
  }
}
void calcFPrime() {
  for (int k = 1; k <= prime_cnt; ++k) {
    const int p = prime[k];
    const ll sqrp = 1ll * p * p;
    for (int i = 1; lis[i] >= sqrp; ++i) {
      const ll x = lis[i] / p;
      const int id = idx(x);
      dec(G[i][0], sub(G[id][0], k - 1));
      dec(G[i][1], 1ll * p * sub(G[id][1], spri[k - 1]) % MOD);
    }
  }
  // f(p) = g_1(p) - g_0(p)
  for (int i = 1; i <= cnt; ++i) Fprime[i] = sub(G[i][1], G[i][0]);
}

int f_p(int p, int c) { return p ^ c; }
int F(int k, ll n) {
  if (n < prime[k] || n <= 1) return 0;
  const int id = idx(n);
  int res = sub(Fprime[id], sub(spri[k - 1], k - 1));
  // F_prime(p_{k - 1}) = spri[k - 1] - (k - 1)
  if (k == 1) res += 2; // 特殊处理 f(2)
  for (int i = k; i <= prime_cnt && 1ll * prime[i] * prime[i] <= n; ++i) {
    ll pw = prime[i], pw2 = pw * pw;
    for (int c = 1; pw2 <= n; ++c, pw = pw2, pw2 *= prime[i])
      inc(res, (1ll * f_p(prime[i], c) * F(i + 1, n / pw) + f_p(prime[i], c + 1)) % MOD);
  }
  return res;
}
```

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

```cpp
/******************************
f()函数中(31-37行) 填函数在质数幂次处的表达式
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）
202-205行按要求填写
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
例：如果该函数在质数处表达式为f(p) =
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]
******************************/

inline ll f(ll p, int e) { // return f(p^e)
  if (p == 1 || e == 0) return 1;
  ll res = mpow(p, e);
  return res * res + 3 * res + 1;
}
ll pow_sum(ll n, int k) { return sum(i^k),i from 1 to n.
  if (k == 0) return n;
  if (k == 1) return n * (n + 1) / 2;
  if (k == 2) return n * (n + 1) * (2 * n + 1) / 6;
}
ll n, f_p[maxn][3]; // F_prime(id(n/i))
```

```cpp
int n_2, n_3, n_6;  // sqrt(n), sqrt3(n), sqrt6(n);
ll val_id[maxn];    // give the id, return the id-th number like 'n/i' ,(val_id[1] = 1)
int val_id_num;     // how many numbers like 'n/i'
int val_id_num_3;   // how many numbers like 'n/i' below n/n_3;
int p[200000 + 100];
bool isp[maxn];
int p_sz_2, p_sz_3, p_sz_6; // pi(n_2), pi(n_3), pi(n_6)
void init() {
  n_2 = (int)sqrt(n);
  n_3 = (int)pow(n, 1.0 / 3.0);
  n_6 = (int)pow(n, 1.0 / 6.0);
  val_id_num = 0;
  for (ll i = 1; i <= n;) {
    val_id[++val_id_num] = i;
    if (i < n) i = n / (n / (i + 1));
  }
  memset(isp, 1, sizeof isp);
  isp[1] = 0;
  for (int i = 2; i <= n_2; i++) {
    if (isp[i]) {
      p[++p_sz_2] = i;
      if (i <= n_3) p_sz_3++;
      if (i <= n_6) p_sz_6++;
    }
    for (int j = 1; j <= p_sz_2 && p[j] * i <= n_2; j++) {
      isp[i * p[j]] = 0;
      if (i % p[j] == 0) break;
    }
  }
}
inline int get_id(ll k) { // give a number like 'n/i', return the id of it
  return k > n_2 ? val_id_num - n / k + 1 : k;
}
ll c[maxn];
void add(int x, ll d) { for (; x < maxn; x += x & -x) c[x] += d; }
ll sum(int x) {
  ll ans = 0;
  for (; x; x &= x - 1) ans += c[x];
  return ans;
}

struct node {
  int k_max;
  ll val, f_val;
};
void update_bfs(int k, int type) {
  queue<node> q;
  while (!q.empty()) q.pop();
  int e = 1;
  for (ll i = p[k]; i < n / n_3; i *= p[k], ++e)
    q.emplace(k, i, type == -1 ? f(p[k], e) : mpow(i, type));
  while (!q.empty()) {
    node hd = q.front(); q.pop();
    if ((hd.val != p[hd.k_max] && type >= 0) || type == -1) {
      ll w = n / hd.val;
      w = n / w;
      if (type == -1) {
        add(get_id(w), hd.f_val);
        add(val_id_num + 1, -1ll * hd.f_val);
      } else {
        add(get_id(w), -1ll * hd.f_val);
        add(val_id_num + 1, hd.f_val);
      }
    }
    for (int i = hd.k_max + 1; hd.val * p[i] < n / n_3 && i <= p_sz_2; i++) {
      ll res = p[i];
      for (int e = 1;; e++) {
        if (hd.val * res < n / n_3) {
```

```cpp
        q.emplace(i, hd.val * res, type == -1 ? hd.f_val * f(p[i], e) : hd.f_val * mpow(res,
    type));
      } else break;
      res *= p[i];
    }
  }
 }
}
void get_f_p(ll n, int times) {
  for (int i = 1; i <= val_id_num; i++)
    for (int j = 0; j <= times; j++)
      f_p[i][j] = pow_sum(val_id[i], j) - 1;
  int now;
  for (now = 1; p[now] <= n_6; now++) {
    for (int j = val_id_num; j >= 1; j--) {
      ll w = val_id[j] / p[now];
      if (w < p[now]) break;
      ll val = 1;
      for (int k = 0; k <= times; k++) {
        f_p[j][k] = f_p[j][k] - val * (f_p[get_id(w)][k] - f_p[p[now - 1]][k]);
        val *= p[now];
      }
    }
  }
  int nnow = now, val = 1;
  for (int tt = 0; tt <= times; tt++) {
    now = nnow;
    memset(c, 0, sizeof c);
    add(1, f_p[1][tt]);
    for (int i = 2; val_id[i] < n / n_3; i++) add(i, f_p[i][tt] - f_p[i - 1][tt]);
    for (; p[now] <= n_3; now++) {
      for (int j = val_id_num; j >= 1; j--) {
        ll w = val_id[j] / p[now];
        if (val_id[j] < n / n_3) break;
        if (w < p[now]) break;
        f_p[j][tt] = w < n / n_3
          ? (f_p[j][tt] - (sum(get_id(w)) - sum(p[now - 1])) * mpow(p[now], tt))
          : f_p[j][tt] = f_p[j][tt] - (f_p[get_id(w)][tt] - sum(p[now - 1])) * mpow(p[now], tt);
      }
      update_bfs(now, tt);
    }
    for (int i = 1; i <= val_id_num && val_id[i] < n / n_3; i++) f_p[i][tt] = sum(i);
    for (; now <= p_sz_2; now++) {
      for (int j = val_id_num; j >= 1; j--) {
        ll w = val_id[j] / p[now];
        if (val_id[j] < n / n_3) break;
        if (w < p[now]) break;
        f_p[j][tt] -= (f_p[get_id(w)][tt] - f_p[p[now - 1]][tt]) * mpow(p[now], tt);
      }
    }
  }

  for (int i = 1; i <= val_id_num; i++) {
    // 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];
    f_p[i][0] = f_p[i][2] + 3 * f_p[i][1] + f_p[i][0];
  }
}
ll F[2000000 + 100];
void get_f_3(ll n) { // V(F_{pi(n^(1/3))+1},n)
  ll q = p[p_sz_3 + 1];
  for (int now = 1; now <= val_id_num; now++) {
    if (val_id[now] < q) {
      F[now] = 1;
    } else if (val_id[now] < q * q) {
      F[now] = 1 + (f_p[now][0] - f_p[q - 1][0]);
    } else {
      F[now] = 1 + (f_p[now][0] - f_p[q - 1][0]);
      for (int pp = p_sz_3 + 1; p[pp] <= (int)(sqrt(val_id[now])) && pp <= p_sz_2; pp++) {
```

```
      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]);
      }
    }
  }
}
void get_f_6(ll n) { // V(F_{pi(n^(1/6))+1},n)
  memset(c, 0, sizeof c), add(1, F[1]);
  for (int i = 2; val_id[i] < n / n_3; i++) add(i, F[i] - F[i - 1]);
  for (int k = p_sz_3; k > p_sz_6; k--) {
    int now = val_id_num;
    for (; val_id[now] >= n / n_3; now--) {
      int e = 1;
      ll _p = p[k];
      while (val_id[now] / _p) {
        if (val_id[now] / _p >= n / n_3) {
          F[now] += F[get_id(val_id[now] / _p)] * f(p[k], e);
        } else {
          F[now] += sum(get_id(val_id[now] / _p)) * f(p[k], e);
        }
        _p *= p[k], e++;
      }
    }
    if (k == 1) break;
    update_bfs(k, -1); // bfs to update [lpf(i)==P{k-1}]f(i)
  }
  for (int i = 1; i <= val_id_num && val_id[i] < n / n_3; i++) F[i] = sum(i);
}
void get_f(ll n) {
  for (int k = p_sz_6; k >= 1; k--) {
    for (int now = val_id_num; now >= 1; now--) {
      int e = 1;
      ll _p = p[k];
      while (val_id[now] / _p) {
        F[now] += F[get_id(val_id[now] / _p)] * f(p[k], e);
        _p *= p[k], e++;
      }
    }
  }
}
int main() { // n = 1000000000; 1e10:455052511,0.83s/0.58s; 1e12:37607912018 9.224s/5.105s
  cin >> n;
  init();
  get_f_p(n, 2), get_f_3(n);
  get_f_6(n), get_f(n);
  for (int i = 1; i <= val_id_num; i++) cout << val_id[i] << " : " << F[i] << endl;
}
```

1.9 类欧几里德算法

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

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

```
  if (a >= c || b >= c) { // 取模
    d.f = n * n1 % P * i2 % P * ac % P + bc * n1 % P;
    d.g = ac * n % P * n1 % P * n21 % P * i6 % P + bc * n % P * n1 % P * i2 % P;
    d.h = ac * ac % P * n % P * n1 % P * n21 % P * i6 % P +
          bc * bc % P * n1 % P + ac * bc % P * n % P * n1 % P;
    d.f %= P, d.g %= P, d.h %= P;

    data e = calc(n, a % c, b % c, c); // 迭代

    d.h += e.h + 2 * bc % P * e.f % P + 2 * ac % P * e.g % P;
    d.g += e.g, d.f += e.f;
    d.f %= P, d.g %= P, d.h %= P;
    return d;
  }
  data e = calc(m - 1, c, c - b - 1, a);
  d.f = n * m % P - e.f, d.f = (d.f % P + P) % P;
  d.g = m * n % P * n1 % P - e.h - e.f, d.g = (d.g * i2 % P + P) % P;
  d.h = n * m % P * (m + 1) % P - 2 * e.g - 2 * e.f - d.f;
  d.h = (d.h % P + P) % P;
  return d;
}
```

1.10 中国剩余定理

```
ll inv(ll a, ll p) {
  ll x, y;
  exgcd(a, p, x, y);
  return (x + p) % p;
}
ll CRT(ll n, ll *a, ll *m) {
  ll lcm = 1, res = 0;
  for (ll i = 0; i < n; ++i) lcm *= m[i];
  for (ll i = 0; i < n; ++i) {
    ll t = lcm / m[i], x = inv(t, m[i]);
    res = (res + a[i] * t % lcm * x) % lcm;
  }
  return res;
}
```

模数不互质的情况 $\begin{cases} x \equiv a_1 & (\pmod{m})_1) \\ x \equiv a_2 & (\pmod{m})_2) \end{cases}$，则转换为 $m_1p - m_2q = a_2 - a_1$，最终解（若有解）为 $x \equiv m_1p + a_1 \pmod{\operatorname{lcm}(m_1, m_2)}$。

拓展中国剩余定理:

```
int exctr(int n, int *a, int *m) {
  int M = m[0], res = a[0];
  for (int i = 1; i < n; ++i) {
    int a = M, b = m[i], c = (a[i] - res % b + b) % b, x, y;
    int g = exgcd(a, b, x, y), bg = b / g;
    if (c % g != 0) return -1;
    x = 1LL * x * (c / g) % bg;
    res += x * M;
    M *= bg;
    res = (res % M + M) % M;
  }
  return res;
}
```

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 $\text{ord}_m a$。

- 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 $\iff$ $\forall 1\le i\le k$，有 $g^{\frac{\varphi(m)}{p_i}}\not\equiv 1 \pmod m$，集合 $S=g^s\mid 1\le s\le\varphi(m),(s,\varphi(m))=1$ 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 $x=A\lceil\sqrt{p}\rceil-B, 0\le A,B\le\lceil\sqrt{p}\rceil$，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。

```
// 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 $\int_a^b f(x)dx$。

```
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：$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$

```
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。

```
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;
```

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

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。

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

```
const double EPS = 1e-9;
const int MAXN = MAX_NODE;
double a[MAXN][MAXN], x[MAXN];
int equ, var;

int gauss() {
  int i, j, k, col, max_r;
  for (k = 0, col = 0; k < equ && col < var; k++, col++) {
    max_r = k;
    for (i = k + 1; i < equ; i++)
      if (fabs(a[i][col]) > fabs(a[max_r][col])) max_r = i;
    if (fabs(a[max_r][col]) < EPS) return 0;

    if (k != max_r) {
      for (j = col; j < var; j++) swap(a[k][j], a[max_r][j]);
      swap(x[k], x[max_r]);
    }

    x[k] /= a[k][col];
    for (j = col + 1; j < var; j++) a[k][j] /= a[k][col];
    a[k][col] = 1;

    for (i = k + 1; i < equ; i++)
      if (i != k) {
        x[i] -= x[k] * a[i][col];
        for (j = col + 1; j < var; j++) a[i][j] -= a[k][j] * a[i][col];
        a[i][col] = 0;
      }
  }

  for (col = equ - 1, k = var - 1; ~col; --col, --k) {
    if (fabs(a[col][k]) > 0) {
      for (i = 0; i < k; ++i) {
        x[i] -= x[k] * a[i][col];
        for (j = col + 1; j < var; j++) a[i][j] -= a[k][j] * a[i][col];
        a[i][col] = 0;
      }
    }
  }

  return 1;
}
```

1.18 线性回归

```
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;
```

```
  };
  auto L = [](const vec &a, const vec &b) {
    int da = (a.size() > 1 || (a.size() == 1 && a[0])) ? a.size() - 1 : -1000;
    int db = (b.size() > 1 || (b.size() == 1 && b[0])) ? b.size() - 1 : -1000;
    return std::max(da, db + 1);
  };
  auto prime_power = [&](const vec &s, int64 mod, int64 p, int64 e) {
    // linear feedback shift register mod p^e, p is prime
    std::vector<vec> a(e), b(e), an(e), bn(e), ao(e), bo(e);
    vec t(e), u(e), r(e), to(e, 1), uo(e), pw(e + 1);
    ;
    pw[0] = 1;
    for (int i = pw[0] = 1; i <= e; ++i) pw[i] = pw[i - 1] * p;
    for (int64 i = 0; i < e; ++i) {
      a[i] = {pw[i]}, an[i] = {pw[i]};
      b[i] = {0}, bn[i] = {s[0] * pw[i] % mod};
      t[i] = s[0] * pw[i] % mod;
      if (t[i] == 0) {
        t[i] = 1, u[i] = e;
      } else {
        for (u[i] = 0; t[i] % p == 0; t[i] /= p, ++u[i])
          ;
      }
    }
    for (size_t k = 1; k < s.size(); ++k) {
      for (int g = 0; g < e; ++g) {
        if (L(an[g], bn[g]) > L(a[g], b[g])) {
          ao[g] = a[e - 1 - u[g]];
          bo[g] = b[e - 1 - u[g]];
          to[g] = t[e - 1 - u[g]];
          uo[g] = u[e - 1 - u[g]];
          r[g] = k - 1;
        }
      }
      a = an, b = bn;
      for (int o = 0; o < e; ++o) {
        int64 d = 0;
        for (size_t i = 0; i < a[o].size() && i <= k; ++i) {
          d = (d + a[o][i] * s[k - i]) % mod;
        }
        if (d == 0) {
          t[o] = 1, u[o] = e;
        } else {
          for (u[o] = 0, t[o] = d; t[o] % p == 0; t[o] /= p, ++u[o])
            ;
          int g = e - 1 - u[o];
          if (L(a[g], b[g]) == 0) {
            extand(bn[o], k + 1);
            bn[o][k] = (bn[o][k] + d) % mod;
          } else {
            int64 coef =
                t[o] * inverse(to[g], mod) % mod * pw[u[o] - uo[g]] % mod;
            int m = k - r[g];
            extand(an[o], ao[g].size() + m);
            extand(bn[o], bo[g].size() + m);
            for (size_t i = 0; i < ao[g].size(); ++i) {
              an[o][i + m] -= coef * ao[g][i] % mod;
              if (an[o][i + m] < 0) an[o][i + m] += mod;
            }
            while (an[o].size() && an[o].back() == 0) an[o].pop_back();
            for (size_t i = 0; i < bo[g].size(); ++i) {
              bn[o][i + m] -= coef * bo[g][i] % mod;
              if (bn[o][i + m] < 0) bn[o][i + m] -= mod;
            }
            while (bn[o].size() && bn[o].back() == 0) bn[o].pop_back();
          }
        }
      }
```

```
      }
      return std::make_pair(an[0], bn[0]);
    };

    std::vector<std::tuple<int64, int64, int>> fac;
    for (int64 i = 2; i * i <= mod; ++i)
      if (mod % i == 0) {
        int64 cnt = 0, pw = 1;
        while (mod % i == 0) mod /= i, ++cnt, pw *= i;
        fac.emplace_back(pw, i, cnt);
      }
    if (mod > 1) fac.emplace_back(mod, mod, 1);
    std::vector<vec> as;
    size_t n = 0;
    for (auto &&x : fac) {
      int64 mod, p, e;
      vec a, b;
      std::tie(mod, p, e) = x;
      auto ss = s;
      for (auto &&x : ss) x %= mod;
      std::tie(a, b) = prime_power(ss, mod, p, e);
      as.emplace_back(a);
      n = std::max(n, a.size());
    }
    vec a(n), c(as.size()), m(as.size());
    for (size_t i = 0; i < n; ++i) {
      for (size_t j = 0; j < as.size(); ++j) {
        m[j] = std::get<0>(fac[j]);
        c[j] = i < as[j].size() ? as[j][i] : 0;
      }
      a[i] = crt(c, m);
    }
    return a;
  }

  LinearRecurrence(const vec &s, const vec &c, int64 mod)
      : init(s), trans(c), mod(mod), m(s.size()) {}

  LinearRecurrence(const vec &s, int64 mod, bool is_prime = true) : mod(mod) {
    vec A = is_prime ? BerlekampMassey(s, mod) : ReedsSloane(s, mod);
    if (A.empty()) A = {0};
    m = A.size() - 1;
    trans.resize(m);
    for (int i = 0; i < m; ++i) { trans[i] = (mod - A[i + 1]) % mod; }
    std::reverse(trans.begin(), trans.end());
    init = {s.begin(), s.begin() + m};
  }

  int64 calc(int64 n) {
    if (mod == 1) return 0;
    if (n < m) return init[n];
    vec v(m), u(m << 1);
    int msk = !!n;
    for (int64 m = n; m > 1; m >>= 1) msk <<= 1;
    v[0] = 1 % mod;
    for (int x = 0; msk; msk >>= 1, x <<= 1) {
      std::fill_n(u.begin(), m * 2, 0);
      x |= !!(n & msk);
      if (x < m)
        u[x] = 1 % mod;
      else { // can be optimized by fft/ntt
        for (int i = 0; i < m; ++i) {
          for (int j = 0, t = i + (x & 1); j < m; ++j, ++t) {
            u[t] = (u[t] + v[i] * v[j]) % mod;
          }
        }
        for (int i = m * 2 - 1; i >= m; --i) {
          for (int j = 0, t = i - m; j < m; ++j, ++t) {
```

```
            u[t] = (u[t] + trans[j] * u[i]) % mod;
          }
        }
      }
      v = {u.begin(), u.begin() + m};
    }
    int64 ret = 0;
    for (int i = 0; i < m; ++i) { ret = (ret + v[i] * init[i]) % mod; }
    return ret;
  }

  vec init, trans;
  int64 mod;
  int m;
};
```

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：$\sum(-a[i][j] \times x[j]) \leq a[i][0]$。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。

```
const int MAXN = 3e3 + 3, MAXM = 3e3 + 3, INF = ~0U >> 2;
int n, m, a[MAXN][MAXM], nxt[MAXM];
void pivot(int l, int e) {
  a[l][e] = -1;
  int t = MAXM - 1;
  for (int i = 0; i <= m; ++i)
    if (a[l][i]) nxt[t] = i, t = i;
  nxt[t] = -1;
  for (int i = 0; i <= n; ++i)
    if (i != l && (t = a[i][e])) {
      a[i][e] = 0;
      for (int j = nxt[MAXM - 1]; ~j; j = nxt[j]) a[i][j] += a[l][j] * t;
    }
}
int simplex() {
  for (;;) {
    int mi = INF, l = 0, e = 0;
    for (int i = 1; i <= m; ++i)
      if (a[0][i] > 0) {
        e = i;
        break;
      }
    if (!e) return a[0][0];
    for (int i = 1; i <= n; ++i)
      if (a[i][e] < 0 && a[i][0] < mi) mi = a[i][0], l = i;
    pivot(l, e);
  }
}
```

1.20 实数线性规划

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

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

```
  for (int j = 0; j < m - 1; ++j) D[n][j] = c[j];
  D[n + 1][m - 1] = -1;
  for (double d;;) {
    if (r < n) {
      swap(ix[s], ix[r + m]);
      D[r][s] = 1 / D[r][s];
      vector<int> speed_up;
      for (int j = 0; j <= m; ++j)
        if (j != s) {
          D[r][j] *= -D[r][s];
          if (D[r][j]) speed_up.push_back(j);
        }
      for (int i = 0; i <= n + 1; ++i)
        if (i != r) {
          for (int j : speed_up) D[i][j] += D[r][j] * D[i][s];
          D[i][s] *= D[r][s];
        }
    }
    r = -1, s = -1;
    for (int j = 0; j < m; ++j)
      if ((s < 0 || ix[s] > ix[j]) &&
          (D[n + 1][j] > EPS || (D[n + 1][j] > -EPS && D[n][j] > EPS)))
        s = j;
    if (s < 0) break;
    for (int i = 0; i < n; ++i)
      if (D[i][s] < -EPS)
        if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] / D[i][s]) < -EPS ||
            (d < EPS && ix[r + m] > ix[i + m]))
          r = i;
    if (r < 0) return VD(); //无边界
  }
  if (D[n + 1][m] < -EPS) return VD(); // 无解
  VD x(m - 1);
  for (int i = m; i < n + m; ++i)
    if (ix[i] < m - 1) x[ix[i]] = D[i - m][m];
  return x; // 最优值在D[n][m]
}
```

UOJ 板题最快榜代码:

```
#include <bits/stdc++.h>
using namespace std;

#define eps 1e-7
int simplex(vector<vector<double>> &a, vector<double> &b, vector<double> &c,
            vector<int> &basic) {
  int m = b.size(), n = c.size();
  while (true) {
    int k = -1;
    for (int j = 0; j < n; ++j)
      if (c[j] < -eps) {
        k = j;
        break;
      }
    if (k == -1) {
      double ans = 0;
      for (int i = 0; i < m; ++i) ans += c[basic[i]] * b[i];
      return 0;
    }
    int l = -1;
    for (int i = 0; i < m; ++i)
      if (a[i][k] > eps) {
        if (l == -1)
          l = i;
        else {
          double ti = b[i] / a[i][k], tl = b[l] / a[l][k];
          if (ti < tl - eps || (ti < tl + eps && basic[i] < basic[l])) l = i;
        }
      }
```

```
    if (l == -1) return -1;
    basic[l] = k;
    double tmp = 1 / a[l][k];
    for (int j = 0; j < n; ++j) a[l][j] *= tmp;
    b[l] *= tmp;
    for (int i = 0; i < m; ++i)
      if (i != l) {
        tmp = a[i][k];
        for (int j = 0; j < n; ++j) a[i][j] -= tmp * a[l][j];
        b[i] -= tmp * b[l];
      }
    tmp = c[k];
    for (int j = 0; j < n; ++j) c[j] -= tmp * a[l][j];
  }
}

int main() {
  ios::sync_with_stdio(false);
  int n, m, T;
  cin >> n >> m >> T;
  vector<double> c(n + m, 0);
  for (int i = 0; i < n; ++i) {
    cin >> c[i];
    c[i] *= -1;
  }
  auto C = c;
  vector<vector<double>> a(m, vector<double>(n + m, 0));
  vector<double> b(m);
  vector<int> basic(m, -1), tmp;
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) cin >> a[i][j];
    a[i][i + n] = 1;
    cin >> b[i];
    if (b[i] > -eps)
      basic[i] = i + n;
    else
      tmp.push_back(i);
  }
  if (!tmp.empty()) {
    sort(tmp.begin(), tmp.end(), [&](int i, int j) { return b[i] > b[j]; });
    vector<vector<double>> A;
    vector<double> B, C(n + m + 1, 0);
    vector<int> Basic;
    for (int i : tmp) {
      vector<double> foo;
      for (int j = 0; j < n + m; ++j) foo.push_back(-a[i][j]);
      foo.push_back(1);
      double bar = -b[i];
      for (int i = 0; i < A.size(); ++i) {
        double tmp = foo[Basic[i]];
        for (int j = 0; j <= n + m; ++j) foo[j] -= tmp * A[i][j];
        bar -= tmp * B[i];
      }
      for (int j = n + m; j >= 0; --j)
        if (-eps < foo[j] - 1 && foo[j] - 1 < eps) {
          Basic.push_back(j);
          break;
        }
      for (int i = 0; i < A.size(); ++i) {
        double tmp = A[i][Basic.back()];
        for (int j = 0; j <= n + m; ++j) A[i][j] -= tmp * foo[j];
        B[i] -= tmp * bar;
      }
      A.push_back(foo);
      B.push_back(bar);
    }
    for (int i = 0; i < A.size(); ++i)
      if (Basic[i] == n + m) {
```

```
      for (int j = 0; j < n + m; ++j) C[j] = -A[i][j];
    }
  for (int i = 0; i < m; ++i)
    if (b[i] > -eps) {
      A.push_back(a[i]);
      A[A.size() - 1].push_back(0);
      B.push_back(b[i]);
      Basic.push_back(basic[i]);
    }
  simplex(A, B, C, Basic);
  bool flag = true;
  for (int i = 0; i < m; ++i)
    if (Basic[i] == n + m) {
      if (B[i] > eps) {
        cout << "Infeasible\n";
        return 0;
      }
      int k = -1;
      for (int j = 0; j < n + m; ++j)
        if (A[i][j] > eps || A[i][j] < -eps) {
          k = j;
          break;
        }
      if (k != -1) {
        double tmp = 1 / A[i][k];
        Basic[i] = k;
        for (int j = 0; j <= n + m; ++j) A[i][j] *= tmp;
        B[i] *= tmp;
        for (int l = 0; l < m; ++l)
          if (l != i) {
            tmp = A[l][k];
            for (int j = 0; j <= n + m; ++j) A[l][j] -= tmp * A[i][j];
            B[l] -= tmp * B[i];
          }
      } else
        flag = false;
      break;
    }
  if (flag) {
    A.push_back(vector<double>(n + m, 0));
    A[A.size() - 1].push_back(1);
    B.push_back(0);
    Basic.push_back(n + m);
    for (int i = 0; i < A.size() - 1; ++i) {
      double tmp = A[i].back();
      for (int j = 0; j <= n + m; ++j) A[i][j] -= tmp * A[A.size() - 1][j];
      B[i] -= tmp * B.back();
    }
  }
  a = A;
  b = B;
  basic = Basic;
  c.push_back(0);
  for (int i = 0; i < a.size(); ++i) {
    double tmp = c[basic[i]];
    for (int j = 0; j <= n + m; ++j) c[j] -= tmp * a[i][j];
  }
}
auto foo = simplex(a, b, c, basic);
if (foo == -1)
  cout << "Unbounded" << endl;
else {
  double res = 0;
  vector<double> ans(n, 0);
  for (int i = 0; i < basic.size(); ++i)
    if (basic[i] < n) ans[basic[i]] = b[i];
  for (int j = 0; j < n; ++j) res -= C[j] * ans[j];
  cout << setprecision(8) << res << endl;
```

```
    if (T == 1) {
      for (int i = 0; i < n; ++i) cout << setprecision(8) << ans[i] << ' ';
      cout << endl;
    }
  }
  return 0;
}
```

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：

```
const int MAXN = int(3e3);
const int MAXM = int(3e3);
const double INF = 1e20, EPS = 1e-9;

int n, m;
double a[MAXM][MAXN], v;

void pivot(int l, int e) {
  int i, j;
  a[l][e] = 1 / a[l][e];
  for (j = 0; j <= n; ++j)
    if (j != e) a[l][j] *= a[l][e];
  for (i = 1; i <= m; ++i)
    if (i != l && fabs(a[i][e]) > EPS) {
      for (j = 0; j <= n; ++j)
        if (j != e) a[i][j] -= a[i][e] * a[l][j];
      a[i][e] = -a[i][e] * a[l][e];
    }
  v += a[0][e] * a[l][0];
  for (j = 1; j <= n; ++j)
    if (j != e) a[0][j] -= a[0][e] * a[l][j];
  a[0][e] = -a[0][e] * a[l][e];
}

double simplex() {
  int e, l, i;
  double mn;
  v = 0;
  while (true) {
    for (e = 1; e <= n; ++e)
      if (a[0][e] > EPS) break;
    if (e > n) return v;
    for (i = 1, mn = INF; i <= m; ++i)
      if (a[i][e] > EPS && mn > a[i][0] / a[i][e])
        mn = a[i][0] / a[i][e], l = i;
    if (mn == INF) return INF;
    pivot(l, e);
  }
}

void solve() {
  static int n, m, g[10];
  static vector<int> con[10], able;
  scanf("%d %d", &n, &m);
  for (int i = 0; i < n; ++i) {
    scanf("%d", g + i);
    con[i].clear();
  }
  if (n == 1) {
    printf("%.10f\n", m >= g[0] ? 1. : 0.);
    return;
  }
  able.clear();
  for (int s = 0, S = 1 << n; s < S; ++s) {
    int sum = 0;
    for (int i = 0; i < n; ++i)
      if (s >> i & 1) sum += g[i];
```

```
    if (sum > m) continue;
    able.push_back(s);
    for (int i = 0; i < n; ++i)
      if (s >> i & 1) con[i].push_back(able.size());
  }
  ::n = able.size();
  ::m = 0;
  static random_device rd;
  mt19937 gen(rd());
  shuffle(able.begin(), able.end(), gen);
  for (int step = 0; step < n; ++step) {
    int f = ++::m;
    for (int i = 0; i <= ::n; ++i) a[f][i] = 0;
    for (int x : con[step]) ++a[f][x];
    if (step + 1 < n) {
      for (int x : con[step + 1]) --a[f][x];
    } else {
      for (int x : con[0]) --a[f][x];
    }
  }

  ++::m;
  a[::m][0] = 1;
  for (int i = 1; i <= ::n; ++i) a[::m][i] = 1;

  ++::m;
  a[::m][0] = -1;
  for (int i = 1; i <= ::n; ++i) a[::m][i] = -1;

  for (int i = 0; i <= ::n; ++i) a[0][i] = 0;
  for (int x : con[0]) ++a[0][x];
  printf("%.10f\n", simplex());
}
```

```
const int MAXN = 3000;
const int MAXM = 3000;
const db EPS = 1e-9;
const db INF = 1e200;

namespace LP {
db a[MAXM][MAXN];
int idA[MAXN], idB[MAXN];
int m, n;

void put_out(int x) {
  if (x == 0)
    printf("Infeasible\n");
  else
    printf("Unbounded\n");
  exit(0);
}
void pivot(int xA, int xB) {
  swap(idA[xA], idB[xB]);
  static int next[MAXN];
  int i, j, last = MAXN - 1;
  db tmp = -a[xB][xA];
  a[xB][xA] = -1.0;
  for (j = 0; j <= n; j++)
    if (fabs(a[xB][j]) > EPS) a[xB][last = next[last] = j] /= tmp;
  next[last] = -1;

  for (i = 0; i <= m; i++)
    if (i != xB && fabs(tmp = a[i][xA]) > EPS)
      for (a[i][xA] = 0.0, j = next[MAXN - 1]; ~j; j = next[j])
        a[i][j] += tmp * a[xB][j];
}
db calc() {
  int xA, xB;
```

```
  db Max, tmp;
  while (1) {
    xA = n + 1, idA[xA] = n + m + 1;
    for (int i = 1; i <= n; i++)
      if (a[0][i] > EPS && idA[i] < idA[xA]) xA = i;

    if (xA == n + 1) return a[0][0];
    xB = m + 1, idB[xB] = n + m + 1, Max = -INF;
    for (int i = 1; i <= m; i++)
      if (a[i][xA] < -EPS && ((tmp = a[i][0] / a[i][xA]) > Max + EPS ||
                              (tmp > Max - EPS && idB[i] < idB[xB])))
        Max = tmp, xB = i;
    if (xB == m + 1) put_out(1);
    pivot(xA, xB);
  }
  return a[0][0];
}
db solve() {
  for (int i = 1; i <= n; i++) idA[i] = i;
  for (int i = 1; i <= m; i++) idB[i] = n + i;
  static db tmp[MAXN];
  db Min = 0.0;
  int l;
  for (int i = 1; i <= m; i++)
    if (a[i][0] < Min) Min = a[i][0], l = i;
  if (Min > -EPS) return calc();

  idA[++n] = 0;
  for (int i = 1; i <= m; i++) a[i][n] = 1.0;
  for (int i = 0; i <= n; i++) tmp[i] = a[0][i], a[0][i] = 0.0;
  a[0][n] = -1.0;

  pivot(n, l);
  if (calc() < -EPS) put_out(0);
  for (int i = 1; i <= m; i++)
    if (!idB[i]) {
      for (int j = 1; j <= n; j++)
        if (fabs(a[0][j]) > EPS) {
          pivot(j, i);
          break;
        }
      break;
    }

  int xA;
  for (xA = 1; xA <= n; xA++)
    if (!idA[xA]) break;
  for (int i = 0; i <= m; i++) a[i][xA] = a[i][n];
  idA[xA] = idA[n], n--;

  for (int i = 0; i <= n; i++) a[0][i] = 0.0;
  for (int i = 1; i <= m; i++)
    if (idB[i] <= n) {
      for (int j = 0; j <= n; j++) a[0][j] += a[i][j] * tmp[idB[i]];
    }

  for (int i = 1; i <= n; i++)
    if (idA[i] <= n) a[0][i] += tmp[idA[i]];
  return calc();
}
db ans[MAXN];
void findAns() {
  for (int i = 1; i <= n; i++) ans[i] = 0.0;
  for (int i = 1; i <= m; i++)
    if (idB[i] <= n) ans[idB[i]] = a[i][0];
}
void work() {
  for (int i = 1; i <= m; ++i)
```

```
    for (int j = 1; j <= n; ++j) a[i][j] *= -1;
  printf("%.10f\n", -double(solve()));
}
} // namespace LP

void solve() {
  static int n, m, g[10];
  static vector<int> con[10], able;

  scanf("%d %d", &n, &m);
  for (int i = 0; i < n; ++i) {
    scanf("%d", g + i);
    con[i].clear();
  }
  if (n == 1) {
    printf("%.10f\n", m >= g[0] ? 1.0 : 0.0);
    return;
  }
  able.clear();
  for (int s = 0; s < (1 << n); ++s) {
    int sum = 0;
    for (int i = 0; i < n; ++i)
      if (s >> i & 1) sum += g[i];
    if (sum > m) continue;
    able.push_back(s);
    for (int i = 0; i < n; ++i)
      if (s >> i & 1) con[i].push_back(able.size());
  }

  LP::n = able.size(), LP::m = 0;
  for (int step = 0; step < n; ++step) {
    int &f = ++LP::m;
    for (int i = 0; i <= LP::n; ++i) LP::a[f][i] = 0;
    for (int x : con[step]) ++LP::a[f][x];
    if (step + 1 < n) {
      for (int x : con[step + 1]) --LP::a[f][x];
    } else {
      for (int x : con[0]) --LP::a[f][x];
    }
  }

  ++LP::m;
  LP::a[LP::m][0] = 1;
  for (int i = 1; i <= LP::n; ++i) LP::a[LP::m][i] = 1;

  ++LP::m;
  LP::a[LP::m][0] = -1;
  for (int i = 1; i <= LP::n; ++i) LP::a[LP::m][i] = -1;

  for (int i = 0; i <= LP::n; ++i) LP::a[0][i] = 0;
  for (int x : con[0]) ++LP::a[0][x];

  static db a2[MAXM][MAXN];
  for (int i = 1; i <= LP::m; ++i)
    for (int j = 1; j <= LP::n; ++j) a2[i][j] = LP::a[i][j];
  for (int i = 1; i <= LP::m; ++i)
    for (int j = 1; j <= LP::n; ++j) LP::a[j][i] = a2[i][j];
  swap(LP::n, LP::m);
  for (int i = 1; i <= max(LP::n, LP::m); ++i) swap(LP::a[0][i], LP::a[i][0]);
  LP::a[0][0] = 0;
  for (int i = 1; i <= LP::m; ++i)
    for (int j = 1; j <= LP::n; ++j) LP::a[i][j] *= -1;
  for (int i = 1; i <= LP::m; ++i) LP::a[i][0] *= -1;
  for (int i = 1; i <= LP::n; ++i) LP::a[0][i] *= -1;

  LP::work();
}
```

1.21 快速傅里叶变换

```
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;
        w *= wn;
      }
    }
  }
  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。

```
int rev[MAXN];
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 wn = mpow(G, (P - 1) / (j << 1));
    for (int i = 0; i < n; i += (j << 1)) {
      for (int k = 0, w = 1; k < j; ++k, w = 1ll * w * wn % P) {
        int &a = x[i + j + k], &b = x[i + j], t = 1ll * w * a % P;
        (a = b - t) < 0 ? a += P : 0;
        (b = b + t) >= P ? b -= P : 0;
      }
    }
  }
  if (sign == -1)
    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。

```
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。

```
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();
```

```cpp
  for (int i = 1, u, v, w; i < n; ++i) {
    cin >> u >> v >> w;
    adj[u].emplace_back(v, w);
    adj[v].emplace_back(u, w);
  }
  dp[1] = -p;
  head = tail = 0;
  dfs(1, 1);

  ll ans = 0;
  for (int i = 1; i <= n; ++i)
    if (dp[i] > ans) ans = dp[i];
  cout << ans << '\n';
}
```

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 $O(n\sqrt{n})$。

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

3 数据结构

3.1 KD Tree

```cpp
// 寻找近点
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 2e5 + 5;
typedef long long ll;

namespace KD_Tree {

const int DIM = 2;

inline ll sqr(int x) { return 1LL * x * x; }

struct Point {
  int x[DIM], id, c;

  ll dist2(const Point &b) const {
    return sqr(x[0] - b.x[0]) + sqr(x[1] - b.x[1]);
  }
};
struct QNode {
  Point p;
  ll dis2;

  QNode() {}
  QNode(Point _p, ll _dis2) : p(_p), dis2(_dis2) {}

  bool operator<(const QNode &b) const {
    return dis2 < b.dis2 || (dis2 == b.dis2 && p.id < b.p.id);
```

```
  }
} ans;
struct cmpx {
  int div;
  cmpx(int _div) : div(_div) {}
  bool operator()(const Point &a, const Point &b) {
    for (int i = 0; i < DIM; ++i)
      if (a.x[(i + div) % DIM] != b.x[(i + div) % DIM])
        return a.x[(i + div) % DIM] < b.x[(i + div) % DIM];
    return true;
  }
};

bool cmp(const Point &a, const Point &b, int div) {
  cmpx cp = cmpx(div);
  return cp(a, b);
}

struct Node {
  Point e;
  Node *lc, *rc;
  int div;
} node_pool[MAXN], *tail, *root;
void init() { tail = node_pool; }
Node *build(Point *a, int l, int r, int div) {
  if (l >= r) return nullptr;
  Node *p = tail++;
  p->div = div;
  int mid = (l + r) >> 1;
  nth_element(a + l, a + mid, a + r, cmpx(div));
  p->e = a[mid];
  p->lc = build(a, l, mid, div ^ 1);
  p->rc = build(a, mid + 1, r, div ^ 1);
  return p;
}
void search(Point p, Node *x, int div) {
  if (!x) return;
  if (cmp(p, x->e, div)) {
    search(p, x->lc, div ^ 1);
    if (ans.dis2 == -1) {
      if (x->e.c <= p.c) ans = QNode(x->e, p.dist2(x->e));
      search(p, x->rc, div ^ 1);
    } else {
      QNode temp(x->e, p.dist2(x->e));
      if (x->e.c <= p.c && temp < ans) ans = temp;
      if (sqr(x->e.x[div] - p.x[div]) <= ans.dis2) search(p, x->rc, div ^ 1);
    }
  } else {
    search(p, x->rc, div ^ 1);
    if (ans.dis2 == -1) {
      if (x->e.c <= p.c) ans = QNode(x->e, p.dist2(x->e));
      search(p, x->lc, div ^ 1);
    } else {
      QNode temp(x->e, p.dist2(x->e));
      if (x->e.c <= p.c && temp < ans) ans = temp;
      if (sqr(x->e.x[div] - p.x[div]) <= ans.dis2) search(p, x->lc, div ^ 1);
    }
  }
}
void search(Point p) {
  ans.dis2 = -1;
  search(p, root, 0);
}
} // namespace KD_Tree

void solve() {
  static KD_Tree::Point p[MAXN];
  int n, m;
```

```
  cin >> n >> m;
  for (int i = 0; i < n; ++i) {
    p[i].id = i;
    cin >> p[i].x[0] >> p[i].x[1] >> p[i].c;
  }
  KD_Tree::init();
  KD_Tree::root = KD_Tree::build(p, 0, n, 0);

  for (KD_Tree::Point q; m; --m) {
    cin >> q.x[0] >> q.x[1] >> q.c;
    KD_Tree::search(q);
    cout << KD_Tree::ans.p.x[0] << ' ' << KD_Tree::ans.p.x[1] << ' '
         << KD_Tree::ans.p.c << '\n';
  }
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  int o_o;
  for (cin >> o_o; o_o; --o_o) solve();

  return 0;
}

// 寻找远点
inline void cmin(int &a, int b) { b < a ? a = b : 1; }
inline void cmax(int &a, int b) { a < b ? a = b : 1; }
inline int ibs(int a) { return a < 0 ? -a : a; }
struct D {
  int d[2], mx0, mx1, mi0, mi1;
  D *l, *r;
} t[N], *rt;
int cpd, ans;
inline bool cmp(const D &a, const D &b) {
  return (a.d[cpd] ^ b.d[cpd]) ? a.d[cpd] < b.d[cpd]
                               : a.d[cpd ^ 1] < b.d[cpd ^ 1];
}
inline void kd_upd(D *u) {
  if (u->l) {
    cmax(u->mx0, u->l->mx0);
    cmax(u->mx1, u->l->mx1);
    cmin(u->mi0, u->l->mi0);
    cmin(u->mi1, u->l->mi1);
  }
  if (u->r) {
    cmax(u->mx0, u->r->mx0);
    cmax(u->mx1, u->r->mx1);
    cmin(u->mi0, u->r->mi0);
    cmin(u->mi1, u->r->mi1);
  }
}
D *kd_bld(int l, int r, int d) {
  int m = l + r >> 1;
  cpd = d;
  std::nth_element(t + l + 1, t + m + 1, t + r + 1, cmp);
  t[m].mx0 = t[m].mi0 = t[m].d[0];
  t[m].mx1 = t[m].mi1 = t[m].d[1];
  if (l ^ m) t[m].l = kd_bld(l, m - 1, d ^ 1);
  if (r ^ m) t[m].r = kd_bld(m + 1, r, d ^ 1);
  kd_upd(t + m);
  return t + m;
}
inline void kd_ins(D *ne) {
  int cd = 0;
  D *u = rt;
  while (true) {
    cmax(u->mx0, ne->mx0), cmin(u->mi0, ne->mi0);
```

```cpp
    cmax(u->mx1, ne->mx1), cmin(u->mi1, ne->mi1);
    if (ne->d[cd] < u->d[cd]) {
      if (u->l)
        u = u->l;
      else {
        u->l = ne;
        return;
      }
    } else {
      if (u->r)
        u = u->r;
      else {
        u->r = ne;
        return;
      }
    }
    cd ^= 1;
  }
}
inline int dist(int x, int y, D *u) {
  int r = 0;
  if (x < u->mi0)
    r = u->mi0 - x;
  else if (x > u->mx0)
    r = x - u->mx0;
  if (y < u->mi1)
    r += u->mi1 - y;
  else if (y > u->mx1)
    r += y - u->mx1;
  return r;
}
inline void kd_quy(D *u, const int &x, const int &y) {
  int dl, dr, d0;
  d0 = ibs(u->d[0] - x) + ibs(u->d[1] - y);
  if (d0 < ans) ans = d0;
  dl = u->l ? dist(x, y, u->l) : inf;
  dr = u->r ? dist(x, y, u->r) : inf;
  if (dl < dr) {
    if (dl < ans) kd_quy(u->l, x, y);
    if (dr < ans) kd_quy(u->r, x, y);
  } else {
    if (dr < ans) kd_quy(u->r, x, y);
    if (dl < ans) kd_quy(u->l, x, y);
  }
}
```

3.2 zkw 线段树

```cpp
int tree[MAXN * 2], pre;

void init(int n, int *a) {
  memset(tree, 0, sizeof(tree));
  for (pre = 1; pre <= n; pre <<= 1) {}
  for (int i = 1; i <= n; ++i) tree[i + pre] = a[i];
  for (int i = pre; i; --i) tree[i] = max(tree[i << 1], tree[i << 1 | 1]);
}

void update(int pos, const int &val) {
  tree[pos += pre] = val;
  for (pos >>= 1; pos; pos >>= 1)
    tree[pos] = max(tree[pos << 1], tree[pos << 1 | 1]);
}

int query(int s, int t) {
  int res = 0;
  for (s += pre - 1, t += pre + 1; s ^ t ^ 1; s >>= 1, t >>= 1) {
```

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

3.3 Splay

```
struct Node {
  long long sum;
  int id, val, lazy, size;
  Node *fa, *ch[2];
} node_pool[MAXN], *pool_it, *root, *nil;

Node *newnode(int id, int val) {
  pool_it->id = id;
  pool_it->lazy = 0;
  pool_it->size = 1;
  pool_it->sum = pool_it->val = val;
  pool_it->fa = pool_it->ch[0] = pool_it->ch[1] = nil;
  return pool_it++;
}

void maintain(Node *u) {
  if (u == nil) { return; }
  u->size = u->ch[0]->size + u->ch[1]->size + 1;
  u->sum = u->ch[0]->sum + u->ch[1]->sum + u->val;
}

void push_down(Node *u) {
  if (u->lazy) {
    if (u->ch[0] != nil) {
      u->ch[0]->val += u->lazy;
      u->ch[0]->sum += 1LL * u->ch[0]->size * u->lazy;
      u->ch[0]->lazy += u->lazy;
    }
    if (u->ch[1] != nil) {
      u->ch[1]->val += u->lazy;
      u->ch[1]->sum += 1LL * u->ch[1]->size * u->lazy;
      u->ch[1]->lazy += u->lazy;
    }
    u->lazy = 0;
  }
}

inline void rot(Node *u) {
  Node *f = u->fa, *ff = f->fa;
  int d = u == f->ch[1];
  push_down(f);
  push_down(u);
  if ((f->ch[d] = u->ch[d ^ 1]) != nil) f->ch[d]->fa = f;
  if ((u->fa = ff) != nil) ff->ch[f == ff->ch[1]] = u;
  f->fa = u;
  u->ch[d ^ 1] = f;
  maintain(f);
  maintain(u);
}

void splay(Node *u, Node *target) {
  for (Node *f; u->fa != target; rot(u))
    if ((f = u->fa)->fa != target) {
      ((u == f->ch[1]) ^ (f == f->fa->ch[1])) ? rot(u) : rot(f);
    }
  if (target == nil) root = u;
}

```

```cpp
inline void insert(int id, int val) {
  if (root == nil) {
    root = newnode(id, val);
    return;
  }
  Node *u = root;
  while (u != nil) {
    int d = id >= u->id;
    ++u->size;
    push_down(u);
    u->sum += val;
    if (u->ch[d] != nil) {
      u = u->ch[d];
    } else {
      u->ch[d] = newnode(id, val);
      u->ch[d]->fa = u;
      u = u->ch[d];
      break;
    }
  }
  splay(u, nil);
}

inline Node *find_pred(int id) {
  Node *u = root, *ret = nil;
  while (u != nil) {
    push_down(u);
    if (u->id < id) {
      ret = u;
      u = u->ch[1];
    } else {
      u = u->ch[0];
    }
  }
  return ret;
}

inline Node *find_succ(int id) {
  Node *u = root, *ret = nil;
  while (u != nil) {
    push_down(u);
    if (u->id > id) {
      ret = u;
      u = u->ch[0];
    } else {
      u = u->ch[1];
    }
  }
  return ret;
}

Node *find_kth(int k) {
  Node *u = root;
  while (u != nil) {
    push_down(u);
    if (u->ch[0]->size + 1 == k) {
      splay(u, nil);
      return u;
    }
    if (u->ch[0]->size >= k) {
      u = u->ch[0];
    } else {
      k -= u->ch[0]->size + 1;
      u = u->ch[1];
    }
  }
  return nil;
}
```

```

Node *range(int l, int r) {
  Node *pred = find_pred(l);
  Node *succ = find_succ(r);

  splay(pred, nil);
  splay(succ, root);
  push_down(pred);
  push_down(succ);
  return root->ch[1]->ch[0];
}

void init() {
  pool_it = node_pool;
  nil = pool_it++;
  nil->ch[0] = nil->ch[1] = nil->fa = nil;
  nil->id = -1, nil->val = 0;
  root = nil;

  insert(INT_MIN, 0), insert(INT_MAX, 0);
}
```

3.4 LCT

```
struct LCT {
  struct node {
    int val, add;
    node *fa, *ch[2];
    void modify(const int &x) {
      val += x, add += x;
    }
  } node_mset[MaxS], *cnode, *null;
  LCT() {
    cnode = node_mset, null = cnode++;
    *null = (node){0, 0, null, {null, null}};
  }
  inline node *newnode() {
    *cnode = (node){0, 0, null, {null, null}};
    return cnode++;
  }
  inline bool isrt(node *u) const {
    return (u->fa->ch[0] != u) && (u->fa->ch[1] != u);
  }
  inline bool which(node *u) const { return u->fa->ch[1] == u; }
  void push_down(node *u) {
    if (!isrt(u)) push_down(u->fa);
    if (u->add) {
      u->ch[0]->modify(u->add);
      u->ch[1]->modify(u->add);
      u->add = 0;
    }
  }
  inline void rotate(node *u) {
    node *f = u->fa;
    int d = which(u);
    f->ch[d] = u->ch[d ^ 1];
    f->ch[d]->fa = f;
    u->ch[d ^ 1] = f;
    u->fa = f->fa;
    if (!isrt(f)) f->fa->ch[which(f)] = u;
    f->fa = u;
  }
  inline void splay(node *u) {
    push_down(u);
    for (node *f; !isrt(u); rotate(u))
      if (!isrt(f = u->fa)) rotate(which(u) == which(f) ? f : u);
```

```
  }
  inline void access(node *x) {
    for (node *y = null; x != null; x = x->fa) {
      splay(x);
      x->ch[1] = y;
      y = x;
    }
  }
  inline void cut(node *u) {
    access(u), splay(u);
    u->ch[0]->fa = null;
    u->ch[0] = null;
  }
  inline void link(node *u, node *v) {
    cut(u), u->fa = v;
  }
} tree;
```

4 字符串

4.1 KMP

```
void get_next(char *S, int *nxt, int n) {
  nxt[0] = -1;
  int j = -1;
  for (int i = 1; i < n; ++i) {
    while ((~j) && S[j + 1] != S[i]) j = nxt[j];
    nxt[i] = (S[j + 1] == S[i]) ? (++j) : j;
  }
}

int pattern(char *S, char *T, int *nxt, int n, int m) {
  int j = -1;
  for (int i = 0; i < m; ++i) {
    while ((~j) && S[j + 1] != T[i]) j = nxt[j];
    j += S[j + 1] == T[i];
    if (j == n - 1) return i - n + 1;
  }
  return -1;
}
```

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

```
void prework(char x[], int m, int next[]) {
  next[0] = m;
  int j = 0;
  while (j + 1 < m && x[j] == x[j + 1]) ++j;
  next[1] = j;
  int k = 1;
  for (int i = 2; i < m; ++i) {
    int p = next[k] + k - 1;
    int L = next[i - k];
    if (i + L < p + 1)
      next[i] = L;
    else {
      j = max(0, p - i + 1);
      while (i + j < m && x[i + j] == x[j]) j++;
      next[i] = j, k = i;
    }
  }
}
```

```cpp
void exkmp(char x[], int m, char y[], int n, int next[], int extend[]) {
  prework(x, m, next);
  int j = 0;
  while (j < n && j < m && x[j] == y[j]) ++j;
  extend[0] = j;
  int k = 0;
  for (int i = 1; i < n; ++i) {
    int p = extend[k] + k - 1;
    int L = next[i - k];
    if (i + L < p + 1)
      extend[i] = L;
    else {
      j = max(0, p - i + 1);
      while (i + j < n && j < m && y[i + j] == x[j]) j++;
      extend[i] = j, k = i;
    }
  }
}
```

4.3 AC 自动机

```cpp
int tr[MAX_NODE][26], fail[MAX_NODE], dep[MAX_NODE], node_c;

int add_char(int u, int id) {
  if (tr[u][id] < 0) tr[u][id] = node_c++;
  return tr[u][id];
}
void build_acam() {
  queue<int> que;
  fail[0] = 0;
  for (int i = 0; i < 26; ++i)
    if (~tr[0][i]) {
      que.push(tr[0][i]);
      fail[tr[0][i]] = 0;
    } else {
      tr[0][i] = 0;
    }
  while (!que.empty()) {
    int u = que.front(), f = fail[u];
    que.pop();
    for (int i = 0; i < 26; ++i)
      if (~tr[u][i]) {
        que.push(tr[u][i]);
        fail[tr[u][i]][i] = tr[f][i];
      } else {
        tr[u][i] = tr[f][i];
      }
  }
  for (int i = 1; i < node_c; ++i) adj[fail[i]].push_back(i);
}
```

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；

```cpp
const unsigned int KEY = 6151;
const unsigned int MOD = 1610612741;
// 64 位哈希参数 KEY 随意 MOD 4611686018427387847ll
unsigned int hash[MAXN], p[MAXN];

unsigned int get_hash(int l, int r) { return (hash[r] + MOD - 1ULL * hash[l - 1] * p[r - l + 1] % MOD) % MOD; }
```

```

void init(char *s, int n) {
  p[0] = 1;
  for (int i = 1; i <= n; ++i) {
    p[i] = p[i - 1] * KEY % MOD;
    hash[i] = (1LL * hash[i - 1] * KEY + s[i]) % MOD;
  }
}
```

CCPC 秦皇岛 2020 J：两次哈希

```
const int MAXN = 3e5 + 3;
const int MAX_PRIME = 8960453 + 3;
const int MOD = 998244353;
const ll BASE = 709;
const ll HASH_MOD = 4611686018427387847ll;

char s[MAXN];
int fac[MAXN], inv[MAXN], fac_inv[MAXN], prime[MAXN * 2];
ll ha[MAXN], p[MAXN], pref[MAXN], suff[MAXN], value[MAXN * 2];
pair<ll, int> bin[MAXN];

int cnt[MAXN], pidx[MAXN], sidx[MAXN];
bitset<MAX_PRIME> mark;

ll fmul(ll a, ll b) {
  ll k = (ll)((1.l * a * b) / (1.l * HASH_MOD)), t = a * b - k * HASH_MOD;
  for (t -= HASH_MOD; t < 0; t += HASH_MOD) {}
  return t;
}
ll getRange(int l, int r) {
  return (ha[r] - fmul(ha[l - 1], p[r - l + 1]) + HASH_MOD) % HASH_MOD;
}

int gao(int n, int d) {
  int tot = 0;
  for (int l = 1, r = d; r <= n; l += d, r += d) value[tot++] = getRange(l, r);
  int bunch = n / d, rest = n % d;
  if (rest) {
    for (int r = n, l = n - d + 1; l >= 1; l -= d, r -= d)
      value[tot++] = getRange(l, r);
    sort(value, value + tot);
    tot = unique(value, value + tot) - value;
    pref[0] = pref[1] = suff[n] = suff[n + 1] = 1;
    for (int i = d; i <= n; i += d) {
      int idx = lower_bound(value, value + tot, getRange(i - d + 1, i)) - value;
      pidx[i] = idx;
      pref[i] = fmul(pref[i - d], prime[idx]);
    }
    for (int i = n - d + 1; i >= 1; i -= d) {
      int idx = lower_bound(value, value + tot, getRange(i, i + d - 1)) - value;
      sidx[i] = idx;
      suff[i] = fmul(suff[i + d], prime[idx]);
    }
    int sc = 0, cur = fac[bunch];
    memset(cnt, 0, tot * sizeof(int));
    for (int l = 1, r = rest; r <= n; l += d, r += d) {
      if (r + d <= n) {
        ++cnt[sidx[r + 1]];
        cur = 1ll * cur * inv[cnt[sidx[r + 1]]] % MOD;
      }
      bin[sc++] = {fmul(pref[l - 1], suff[r + 1]), l};
    }
    sort(bin, bin + sc);
    mark[bin[0].second] = 1;
    for (int i = 1; i < sc; ++i)
      mark[bin[i].second] = bin[i].first != bin[i - 1].first;

    int res = 0;
```

```
    for (int l = 1, r = rest; r <= n; l += d, r += d) {
      if (l - 1 >= 1) {
        ++cnt[pidx[l - 1]];
        cur = 1ll * cur * inv[cnt[pidx[l - 1]]] % MOD;
      }
      if (mark[l]) {
        res += cur;
        if (res >= MOD) res -= MOD;
      }
      if (r + 1 <= n) {
        cur = 1ll * cur * cnt[sidx[r + 1]] % MOD;
        --cnt[sidx[r + 1]];
      }
    }
    return res;
  } else {
    sort(value, value + tot);
    ll pre = value[0];
    int res = fac[bunch], cnt = 1;
    for (int i = 1; i < tot; ++i) {
      if (value[i] != pre) {
        if (cnt > 1) res = 1ll * res * fac_inv[cnt] % MOD;
        cnt = 1, pre = value[i];
      } else {
        ++cnt;
      }
    }
    if (cnt > 1) res = 1ll * res * fac_inv[cnt] % MOD;
    return res;
  }
}

void solve() {
  cin >> (s + 1);
  int n = strlen(s + 1);
  for (int i = 1; i <= n; ++i)
    ha[i] = (fmul(ha[i - 1], BASE) + s[i]) % HASH_MOD;

  int ans = 0;
  for (int d = 1; d <= n; ++d) {
    ans += gao(n, d);
    if (ans >= MOD) ans -= MOD;
  }
  cout << ans << "\n";
}

void prework() {
  p[0] = 1;
  for (int i = 1; i < MAXN; ++i) p[i] = fmul(p[i - 1], BASE);
  fac[0] = fac[1] = 1;
  fac_inv[0] = fac_inv[1] = inv[1] = 1;
  for (int i = 2; i < MAXN; ++i) {
    fac[i] = 1ll * fac[i - 1] * i % MOD;
    inv[i] = 1ll * (MOD - MOD / i) * inv[MOD % i] % MOD;
    fac_inv[i] = 1ll * fac_inv[i - 1] * inv[i] % MOD;
  }

  int pc = 0;
  for (int i = 2; i < MAX_PRIME; ++i) {
    if (!mark[i]) prime[pc++] = i;
    for (int j = 0; j < pc; ++j) {
      int t = i * prime[j];
      if (t >= MAX_PRIME) break;
      mark[t] = 1;
      if (i % prime[j] == 0) break;
    }
  }
}
```

```cpp

int main(int argc, char *argv[]) {
  ios::sync_with_stdio(false);
  cin.tie(nullptr), cout.tie(nullptr);

  prework();
  int T; cin >> T;
  for (int step = 1; step <= T; ++step) {
    cout << "Case #" << step << ": ";
    solve();
  }

  return 0;
}
```

4.5 mancher

```cpp
void mancher(char *s, int n) {
  static char str[2 * MAX_LENGTH];
  str[0] = '~', str[1] = '!';
  int len = 2;
  for (int i = 0; i < n; ++i) {
    str[len++] = s[i], str[len++] = '!';
  }
  str[len] = 0;
  for (int i = 1, id = 0, mx = 0; i < len; ++i) {
    p[i] = i < mx ? min(p[2 * id - i], mx - i) : 1;
    while (str[i + p[i]] == str[i - p[i]]) ++p[i];
    if (mx < i + p[i]) id = i, mx = i + p[i];
  }
}
```

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。

```cpp
struct PT {
  char s[MAXL];
  int fail[MAXL], ch[26][MAXL], l[MAXL], dep[MAXL], cnt[MAXL], lst, nc, n;
  void init() {
    l[0] = 0, l[1] = -1;
    fail[0] = fail[1] = 1;
    for (int i = 0; i < 26; ++i)
      for (int j = 0; j < nc; ++j) ch[i][j] = 0;
    for (int i = 2; i < nc; ++i) l[i] = 0, fail[i] = 0;

    lst = 0, nc = 2, n = 0, s[0] = '#';
  }

  int insert(char c) {
    int id = c - 'a';
    s[++n] = c;
    while (s[n - l[lst] - 1] != s[n]) { lst = fail[lst]; }
    if (ch[id][lst] == 0) {
      l[nc] = l[lst] + 2;
      int f = fail[lst];
      while (s[n - l[f] - 1] != s[n]) { f = fail[f]; }
```

```
      fail[nc] = ch[id][f];
      dep[nc] = dep[fail[nc]] + 1;
      ch[id][lst] = nc;
      ++nc;
    }
    ++cnt[lst = ch[id][lst]];
    return lst;
  }

  void count() { for (int i = nc - 1; ~i; --i) cnt[fail[i]] += cnt[i]; }
} pt;

// 求最长双回文串
char S[MAXL];
int len[MAXL];
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);

  cin >> S;
  int n = strlen(S);
  pt.init();
  for (int i = 0; i < n; ++i) { len[i] = pt.l[pt.insert(S[i])]; }
  pt.init();
  int ans = 0;
  for (int i = n - 1; i; --i) {
    ans = max(ans, len[i - 1] + pt.l[pt.insert(S[i])]);
  }
  cout << ans << "\n";

  return 0;
}
```

4.7 后缀数组（倍增）

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e5 + 3;

template <int MAX_LENGTH> class SuffixArray {
public:
  int n, sa[MAX_LENGTH], rank[MAX_LENGTH], height[MAX_LENGTH];

  void compute(char *s, int n, int m) {
    int i, p, w, j, k;
    this->n = n;
    if (n == 1) {
      sa[0] = rank[0] = height[0] = 0;
      return;
    }
    memset(cnt, 0, m * sizeof(int));
    for (i = 0; i < n; ++i) ++cnt[rank[i] = s[i]];
    for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
    for (i = n - 1; ~i; --i) sa[--cnt[rank[i]]] = i;
    for (w = 1; w < n; w <<= 1, m = p) {
      for (p = 0, i = n - 1; i >= n - w; --i) id[p++] = i;
      for (i = 0; i < n; ++i)
        if (sa[i] >= w) id[p++] = sa[i] - w;
      memset(cnt, 0, m * sizeof(int));
      for (i = 0; i < n; ++i) ++cnt[px[i] = rank[id[i]]];
      for (i = 1; i < m; ++i) cnt[i] += cnt[i - 1];
      for (i = n - 1; ~i; --i) sa[--cnt[px[i]]] = id[i];
      memcpy(old_rank, rank, n * sizeof(int));
      for (i = p = 1, rank[sa[0]] = 0; i < n; ++i)
```

```
        rank[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p - 1 : p++;
    }
    for (i = 0; i < n; ++i) rank[sa[i]] = i;
    for (i = k = height[rank[0]] = 0; i < n; height[rank[i++]] = k)
      if (rank[i])
        for (k > 0 ? --k : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; ++k) {}
  }

  void init_st_table(int n) {
    int lgn = lg[n];
    for (int i = 0; i < n; ++i) table[0][i] = height[i];
    for (int i = 1; i <= lgn; ++i)
      for (int j = 0, l = 1 << (i - 1); j + l < n; ++j)
        table[i][j] = min(table[i - 1][j], table[i - 1][j + l]);
  }

  int lcp(int i, int j) {
    if (i > j) swap(i, j);
    ++i;
    int lgl = lg[j - i + 1];
    return min(table[lgl][i], table[lgl][j - (1 << lgl) + 1]);
  }

private:
  int table[17][MAX_LENGTH], lg[MAX_LENGTH];
  int old_rank[MAX_LENGTH], id[MAX_LENGTH], px[MAX_LENGTH], cnt[MAX_LENGTH];

  bool cmp(int x, int y, int w) {
    return old_rank[x] == old_rank[y] && old_rank[x + w] == old_rank[y + w];
  }
};

char s[MAXN];
SuffixArray<MAXN> sa;

int main(int argc, char *argv[]) {
  int n = fread(s, 1, MAXN, stdin);
  while (s[n - 1] - 97u > 25) --n;
  for (int i = 0; i < n; ++i) s[i] -= 'a';
  s[n] = '$';
  sa.compute(s, n, 26);
  for (int i = 0; i < n; ++i) printf("%d%c", sa.sa[i] + 1, " \n"[i == n - 1]);
  for (int i = 1; i < n; ++i) printf("%d%c", sa.height[i], " \n"[i == n - 1]);
  return 0;
}
```

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。

```
#include <bits/stdc++.h>

const int BUFFER_SIZE = 1u << 23 | 1;
char buffer[BUFFER_SIZE], *buffer_ptr = buffer;
#define alloc(x, type, len)                                               \
  type *x = (type *)buffer_ptr;                                           \
  buffer_ptr += (len) * sizeof(type);
#define clear_buffer()                                                    \
  memset(buffer, 0, buffer_ptr - buffer), buffer_ptr = buffer;

template <int MAX_LENGTH> class SuffixArray {
#define L_TYPE true
#define S_TYPE false
public:
  int sa[MAX_LENGTH], rank[MAX_LENGTH], height[MAX_LENGTH];
  void compute(int n, int m, int *s) {
    sais(n, m, s, sa);
```

```
    for (int i = 0; i < n; ++i) rank[sa[i]] = i;
    for (int i = 0, h = 0; i < n; ++i) {
      if (rank[i]) {
        int j = sa[rank[i] - 1];
        while (s[i + h] == s[j + h]) ++h;
        height[rank[i]] = h;
      } else {
        h = 0;
      }
      if (h) --h;
    }
  }

private:
  int l_bucket[MAX_LENGTH], s_bucket[MAX_LENGTH];

  void induce(int n, int m, int *s, bool *type, int *sa, int *bucket,
              int *l_bucket, int *s_bucket) {
    memcpy(l_bucket + 1, bucket, m * sizeof(int));
    memcpy(s_bucket + 1, bucket + 1, m * sizeof(int));
    sa[l_bucket[s[n - 1]]++] = n - 1;
    for (int i = 0; i < n; ++i) {
      int t = sa[i] - 1;
      if (t >= 0 && type[t] == L_TYPE) sa[l_bucket[s[t]]++] = t;
    }
    for (int i = n - 1; i >= 0; --i) {
      int t = sa[i] - 1;
      if (t >= 0 && type[t] == S_TYPE) sa[--s_bucket[s[t]]] = t;
    }
  }
  void sais(int n, int m, int *s, int *sa) {
    alloc(type, bool, n + 1);
    alloc(bucket, int, m + 1);
    type[n] = false;
    for (int i = n - 1; i >= 0; --i) {
      ++bucket[s[i]];
      type[i] = s[i] > s[i + 1] || (s[i] == s[i + 1] && type[i + 1] == L_TYPE);
    }
    for (int i = 1; i <= m; ++i) {
      bucket[i] += bucket[i - 1];
      s_bucket[i] = bucket[i];
    }
    memset(rank, -1, n * sizeof(int));

    alloc(lms, int, n + 1);
    int n1 = 0;
    for (int i = 0; i < n; ++i) {
      if (!type[i] && (i == 0 || type[i - 1])) lms[rank[i] = n1++] = i;
    }
    lms[n1] = n;
    memset(sa, -1, n * sizeof(int));
    for (int i = 0; i < n1; ++i) sa[--s_bucket[s[lms[i]]]] = lms[i];
    induce(n, m, s, type, sa, bucket, l_bucket, s_bucket);
    int m1 = 0;
    alloc(s1, int, n + 1);
    for (int i = 0, t = -1; i < n; ++i) {
      int r = rank[sa[i]];
      if (r != -1) {
        int len = lms[r + 1] - sa[i] + 1;
        m1 += t == -1 || len != lms[rank[t] + 1] - t + 1 ||
              memcmp(s + t, s + sa[i], len * sizeof(int)) != 0;
        s1[r] = m1;
        t = sa[i];
      }
    }
    alloc(sa1, int, n + 1);
    if (n1 == m1) {
      for (int i = 0; i < n1; ++i) sa1[s1[i] - 1] = i;
```

```
    } else {
      sais(n1, m1, s1, sa1);
    }
    memset(sa, -1, n * sizeof(int));
    memcpy(s_bucket + 1, bucket + 1, m * sizeof(int));
    for (int i = n1 - 1; i >= 0; --i) {
      int t = lms[sa1[i]];
      sa[--s_bucket[s[t]]] = t;
    }
    induce(n, m, s, type, sa, bucket, l_bucket, s_bucket);
  }
#undef S_TYPE
#undef L_TYPE
};

const int MAXN = 1e5 + 5;
SuffixArray<MAXN> sa;
char str[MAXN];
int s[MAXN];

int main() {
  int n = fread(str, 1, MAXN, stdin);
  while (str[n - 1] - 97u > 25) --n;
  for (int i = 0; i < n; ++i) s[i] = str[i] - 'a' + 1;
  sa.compute(n, 26, s);
  for (int i = 0; i < n; ++i) printf("%d%c", sa.sa[i] + 1, " \n"[i == n - 1]);
  for (int i = 1; i < n; ++i) printf("%d%c", sa.height[i], " \n"[i == n - 1]);
  return 0;
}
```

4.9 后缀自动机

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

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 90000 + 3;
const int ALPHABET = 26;

struct Node {
  int len, cnt;
  Node *link, *next[ALPHABET];
  void init(int len = 0) {
    link = nullptr;
    this->len = len, cnt = 0;
    memset(next, 0, sizeof(next));
  }
};

template <int MAX_LENGTH> class SAM {
public:
  Node *last, *root;

  void init() {
    pool_ptr = pool;
    last = root = new_node(0);
  }

  void extend(int chr) {
    Node *p = last, *np = new_node(p->len + 1);
    for (last = np; p && !p->next[chr]; p = p->link) p->next[chr] = np;
    if (!p) {
      np->link = root;
    } else {
      Node *q = p->next[chr];
      if (q->len == p->len + 1) {
```

```
        np->link = q;
      } else {
        Node *nq = new_node(p->len + 1);
        memcpy(nq->next, q->next, sizeof(q->next));
        nq->link = q->link, q->link = np->link = nq;
        for (; p && p->next[chr] == q; p = p->link) p->next[chr] = nq;
      }
    }
  }

  void toposort() {
    int size = pool_ptr - pool;
    memset(cnt, 0, size * sizeof(int));
    for (Node *it = pool; it < pool_ptr; ++it) ++cnt[it->len];
    for (int i = 1; i < size; ++i) cnt[i] += cnt[i - 1];
    for (Node *it = pool; it < pool_ptr; ++it) order[--cnt[it->len]] = it;
    for (int i = size - 1; ~i; --i) {
      Node *u = order[i];
      for (int j = 0; j < ALPHABET; ++j)
        u->cnt += u->next[j] ? u->next[j]->cnt + 1 : 0;
    }
  }

  void find_kth(int k, char *str) {
    char *ptr = str;
    Node *u = root;
    while (k) {
      for (int j = 0; j < ALPHABET; ++j) {
        if (!u->next[j]) continue;
        if (u->next[j]->cnt + 1 < k) {
          k -= u->next[j]->cnt + 1;
          continue;
        }
        --k, *ptr++ = j + 'a';
        u = u->next[j];
        break;
      }
    }
    *ptr = 0;
  }

private:
  int cnt[MAX_LENGTH * 2];
  Node pool[MAX_LENGTH * 2], *pool_ptr, *order[MAX_LENGTH * 2];

  Node *new_node(int len) {
    pool_ptr->init(len);
    return pool_ptr++;
  }
};

SAM<MAXN> sam;
char str[MAXN];

int main(int argc, char *argv[]) {
  ios::sync_with_stdio(false);
  cin.tie(nullptr), cout.tie(nullptr);

  cin >> str;
  sam.init();
  for (char *it = str; *it; ++it) sam.extend(*it - 'a');
  sam.toposort();

  int q, k;
  for (cin >> q; q; --q) cin >> k, sam.find_kth(k, str), puts(str);

  return 0;
}
```

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。

```
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 $O(m\sqrt{n})$。

```
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);
```

```
  for (int x = 0; x < nl; ++x)
    for (int y : adj[x]) {
      if (~neg[y]) continue;
      pos[neg[y] = x] = y, ++cnt;
      break;
    }
  for (;;) {
    static int q[MAXN];
    int l = 0, r = 0, ok = 0;
    memset(lx, -1, sizeof(int) * nl);
    memset(ly, -1, sizeof(int) * nr);
    for (int x = 0; x < nl; ++x)
      if (pos[x] < 0) lx[q[r++] = x] = 0;
    while (l < r) {
      int x = q[l++];
      for (int y : adj[x]) {
        if (~ly[y]) continue;
        ly[y] = lx[x] + 1;
        if (~neg[y] && ~lx[neg[y]]) continue;
        (~neg[y]) ? lx[q[r++] = neg[y]] = ly[y] + 1 : ok = 1;
      }
    }
    if (!ok) return cnt;
    for (int x = 0; x < nl; ++x)
      if (pos[x] < 0 && dfs(x)) ++cnt;
  }
}
```

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。

```
const int MAXN = 400 + 3;
const int ALPHA = 1e9 + 10;
const ll NOT = 0;
const ll INF = 3ll * MAXN * ALPHA;
struct KM {
  int n, nl, nr, lk[MAXN], pre[MAXN];
  ll lx[MAXN], ly[MAXN], w[MAXN][MAXN], slack[MAXN];
  bitset<MAXN> vy;

  void init(int n) {
    this->n = n;
    memset(lk, 0, sizeof(int) * (n + 1));
    memset(pre, 0, sizeof(int) * (n + 1));
    memset(lx, 0, sizeof(ll) * (n + 1));
    memset(ly, 0, sizeof(ll) * (n + 1));
    memset(slack, 0, sizeof(ll) * (n + 1));
    for (int i = 0; i <= n; ++i) fill(w[i], w[i] + n + 1, NOT);
  }

  void add_edge(int x, int y, ll z) {
    if (w[y][x] < z) w[y][x] = z;
  }

  ll match() {
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j) lx[i] = max(lx[i], w[i][j]);
    for (int i = 1, py, p, x; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) slack[j] = INF, vy[j] = 0;
      for (lk[py = 0] = i; lk[py]; py = p) {
        ll delta = INF;
        vy[py] = 1, x = lk[py];
        for (int y = 1; y <= n; ++y) {
          if (vy[y]) continue;
          if (lx[x] + ly[y] - w[x][y] < slack[y])
```

```
            slack[y] = lx[x] + ly[y] - w[x][y], pre[y] = py;
          if (slack[y] < delta) delta = slack[y], p = y;
        }
        for (int y = 0; y <= n; ++y)
          if (vy[y]) {
            lx[lk[y]] -= delta, ly[y] += delta;
          } else {
            slack[y] -= delta;
          }
      }
      for (; py; py = pre[py]) lk[py] = lk[pre[py]];
    }

    ll ans = 0;
    for (int i = 1; i <= n; ++i) {
      ans += lx[i] + ly[i];
      if (w[lk[i]][i] == NOT) ans -= NOT;
    }
    return ans;
  }
} km;

int main() {
  int nl, nr, m;
  cin >> nl >> nr >> m;
  km.init(max(nl, nr));
  for (int x, y, z; m; --m) {
    cin >> x >> y >> z;
    km.add_edge(x, y, z);
  }
  cout << km.match() << "\n";
  for (int i = 1; i <= nl; ++i)
    cout << (km.w[km.lk[i]][i] == NOT ? 0 : km.lk[i]) << " \n"[i == nl];

  return 0;
}
```

5.4 一般图最大匹配

```
class GeneralMatch {
public:
  int n;
  vector<vector<int>> g;
  vector<int> match, aux, label, orig, parent;
  queue<int> q;
  int aux_time;

  GeneralMatch(int n)
      : match(n, -1), aux(n, -1), label(n), orig(n), parent(n, -1),
        aux_time(-1) {
    this->n = n;
    g.resize(n);
  }

  void add_edge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
  }

  int find(int x) { return x == orig[x] ? x : orig[x] = find(orig[x]); }

  int lca(int u, int v) {
    ++aux_time;
    u = find(u), v = find(v);
    for (;; swap(u, v)) {
      if (~u) {
```

```
        if (aux[u] == aux_time) return u;
        aux[u] = aux_time;
        if (match[u] == -1) {
          u = -1;
        } else {
          u = find(parent[match[u]]);
        }
      }
    }
  }

  void blossom(int u, int v, int o) {
    while (find(u) != o) {
      parent[u] = v;
      v = match[u];
      q.push(v);
      label[v] = 0;
      orig[u] = orig[v] = o;
      u = parent[v];
    }
  }

  int bfs(int x) {
    iota(orig.begin(), orig.end(), 0);
    fill(label.begin(), label.end(), -1);
    while (!q.empty()) q.pop();
    q.push(x);
    label[x] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int v : g[u]) {
        if (label[v] == -1) {
          parent[v] = u;
          label[v] = 1;
          if (match[v] == -1) {
            while (v != -1) {
              int pv = parent[v];
              int next_v = match[pv];
              match[v] = pv;
              match[pv] = v;
              v = next_v;
            }
            return 1;
          }
          q.push(match[v]);
          label[match[v]] = 0;
        } else if (label[v] == 0 && find(u) != find(v)) {
          int o = lca(u, v);
          blossom(u, v, o);
          blossom(v, u, o);
        }
      }
    }
    return 0;
  }

  int find_max_match() {
    int res = 0;
    for (int i = 0; i < n; ++i) {
      if (~match[i]) continue;
      res += bfs(i);
    }
    return res;
  }
};
```

5.5 SAP

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

  vector<int> adj[MAXN];
  int n, edges_c, dep[MAXN], depc[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++);
  }

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

    static queue<int> que;
    memset(dep + 1, 0, sizeof(int) * n);
    memset(depc + 1, 0, sizeof(int) * n);
    memset(last + 1, 0, sizeof(int) * n);
    while (!que.empty()) que.pop();
    dep[t] = 1, que.push(t);

    while (!que.empty()) {
      int u = que.front();
      que.pop();
      ++depc[dep[u]];
      for (int i = 0, v; i < (int)adj[u].size(); ++i) {
        v = edges[adj[u][i]].to;
        if (dep[v]) continue;
        dep[v] = dep[u] + 1;
        que.push(v);
      }
    }

    int res = 0;
    while (dep[s] <= n) res += dfs(s, INT_MAX);
    return res;
  }
};
```

5.6 dinic

```
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();
```

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

5.8 最小费用流

```
class MinCostFlow {
public:
  struct Result {
    int flow, cost;
  };
  struct Edge {
    int to, next, rest, cost;
  };

  vector<bool> inq;
  vector<int> head, dist, from, flow;
  vector<Edge> edges;

  MinCostFlow(int n, int m) : inq(n), head(n, -1), dist(n), from(n), flow(n) {
    edges.reserve(2 * m);
  }

  void add_edge(int u, int v, int capacity, int cost) {
    internal_add_edge(u, v, capacity, cost);
    internal_add_edge(v, u, 0, -cost);
  }

  void internal_add_edge(int u, int v, int capacity, int cost) {
    edges.push_back((Edge){v, head[u], capacity, cost});
    head[u] = edges.size() - 1;
  }

  Result augment(int source, int sink) {
    fill(dist.begin(), dist.end(), INT_MAX);
    dist[source] = 0;
    flow[source] = INT_MAX;
    queue<int> q;
    q.push(source);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      inq[u] = false;
      for (int it = head[u]; ~it; it = edges[it].next) {
        auto &e = edges[it];
        int v = e.to;
        if (e.rest > 0 && dist[u] + e.cost < dist[v]) {
          from[v] = it;
          dist[v] = dist[u] + e.cost;
          flow[v] = min(e.rest, flow[u]);
          if (!inq[v]) {
            q.push(v);
            inq[v] = true;
          }
        }
      }
    }
```

```cpp

    if (dist[sink] == INT_MAX) return {0, 0};
    int min_flow = flow[sink];
    for (int u = sink; u != source; u = edges[from[u] ^ 1].to) {
      edges[from[u]].rest -= min_flow;
      edges[from[u] ^ 1].rest += min_flow;
    }
    return {min_flow, dist[sink]};
  }

  Result min_cost_flow(int source, int sink) {
    int flow = 0, cost = 0;
    for (;;) {
      auto result = augment(source, sink);
      if (!result.flow) break;
      flow += result.flow, cost += result.cost;
    }
    return {flow, cost};
  }
};
```

5.9 上下界费用流

```cpp
const int MAXN = 53;
const int MAX_NODE = 113;
const int MAX_EDGE = 1e5 + 5;
const int INF = 0x3f3f3f3f;

int n, s, t, ss, tt, tote;
int R[MAXN], C[MAXN], board[MAXN][MAXN];

struct Edge {
  int to, cap, flow, cost;
} edges[MAX_EDGE];
vector<int> adj[MAX_NODE];

int from[MAX_NODE], in[MAX_NODE];
void add_edge(int from, int to, int l, int r, int cost) {
  in[to] += l, in[from] -= l;
  edges[tote] = (Edge){to, r - l, 0, cost};
  adj[from].push_back(tote++);
  edges[tote] = (Edge){from, 0, 0, -cost};
  adj[to].push_back(tote++);
}

bool spfa(int s, int t) {
  static queue<int> q;
  static bool inq[MAX_NODE];
  static int dist[MAX_NODE];
  memset(inq + 1, 0, sizeof(bool) * tt);
  memset(dist + 1, 0x3f, sizeof(int) * tt);
  memset(from + 1, -1, sizeof(int) * tt);
  dist[0] = 0, from[0] = -1;
  q.push(0);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    inq[u] = false;
    for (int e : adj[u]) {
      if (edges[e].cap == edges[e].flow) continue;
      int v = edges[e].to, d = dist[u] + edges[e].cost;
      if (d >= dist[v]) continue;
      dist[v] = d;
      from[v] = e;
      if (!inq[v]) {
        q.push(v);
```

```
        inq[v] = true;
      }
    }
  }
  return dist[t] < INF;
}

pair<int, int> min_cost_max_flow(int s, int t) {
  int flow = 0, cost = 0;
  while (spfa(s, t)) {
    int mi = INF;
    for (int it = from[t]; ~it; it = from[edges[it ^ 1].to])
      mi = min(mi, edges[it].cap - edges[it].flow);
    flow += mi;
    for (int it = from[t]; ~it; it = from[edges[it ^ 1].to]) {
      edges[it].flow += mi, edges[it ^ 1].flow -= mi;
      cost += mi * edges[it].cost;
    }
  }
  return make_pair(flow, cost);
}

void solve() {
  tote = 0;
  s = 2 * n + 1, t = 2 * n + 2, ss = 0, tt = 2 * n + 3;
  for (int i = 0; i <= tt; ++i) adj[i].clear(), in[i] = 0;

  memset(R + 1, 0, sizeof(int) * n);
  memset(C + 1, 0, sizeof(int) * n);

  for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= n; ++j) {
      cin >> board[i][j];
      R[i] += board[i][j];
      C[j] += board[i][j];
    }

  for (int i = 1; i <= n; ++i) {
    add_edge(s, i, R[i], R[i], 0);
    add_edge(s, i + n, C[i], C[i], 0);
  }

  for (int i = 1, l, r; i <= n; ++i) {
    cin >> l >> r;
    add_edge(i, t, l, r, 0);
  }
  for (int i = 1, l, r; i <= n; ++i) {
    cin >> l >> r;
    add_edge(i + n, t, l, r, 0);
  }

  for (int step = n * n / 2, x1, y1, x2, y2; step; --step) {
    cin >> x1 >> y1 >> x2 >> y2;
    if (board[x1][y1] == board[x2][y2]) continue;
    if (board[x2][y2]) swap(x1, x2), swap(y1, y2);
    if (x1 == x2)
      add_edge(y1 + n, y2 + n, 0, 1, 1);
    else
      add_edge(x1, x2, 0, 1, 1);
  }
  add_edge(t, s, 0, INF, 0);
  int sum = 0;
  for (int i = 1; i < tt; ++i) {
    if (in[i] > 0) {
      sum += in[i];
      add_edge(ss, i, 0, in[i], 0);
    } else if (in[i] < 0) {
      add_edge(i, tt, 0, -in[i], 0);
```

```
    }
  }

  pair<int, int> ans = min_cost_max_flow(ss, tt);
  if (sum != ans.first) {
    cout << "-1\n";
  } else {
    cout << ans.second << '\n';
  }
}
```

6 计算几何

7 Java

7.1 进制转换

```
import java.io.*;
import java.util.*;
import java.math.*;

public class Main {
  public static void main(String[] args) {
    InputStream inputStream = System.in;
    OutputStream outputStream = System.out;
    Scanner in = new Scanner(inputStream);
    PrintWriter out = new PrintWriter(outputStream);
    Solver solver = new Solver();
    int testCount = Integer.parseInt(in.next());
    for (int i = 1; i <= testCount; i++)
      solver.solve(i, in, out);
    out.close();
  }

  static class Solver {
    public void solve(int testNumber, Scanner in, PrintWriter out) {
      int a = in.nextInt();
      int b = in.nextInt();
      String num = in.next();

      BigInteger value = BigInteger.ZERO;
      for (int i = 0; i < num.length(); ++i) {
        value = value.multiply(BigInteger.valueOf(a));
        value = BigInteger.valueOf(getValue(num.charAt(i))).add(value);
      }
      out.println(a + " " + num);

      if (value.equals(BigInteger.ZERO)) {
        out.println(b + " 0");
        out.println();
        return;
      }

      out.print(b + " ");

      char[] ans = new char[1000];
      int length = 0;
      while (!value.equals(BigInteger.ZERO)) {
        int digit = value.mod(BigInteger.valueOf(b)).intValue();
        value = value.divide(BigInteger.valueOf(b));
        ans[length] = getChar(digit);
        ++length;
      }

```

```
      for (int i = length - 1; i >= 0; --i) {
        out.print(ans[i]);
      }
      out.println("\n");
    }

    private int getValue(char ch) {
      if (ch >= 'A' && ch <= 'Z') {
        return ch - 'A' + 10;
      }
      if (ch >= 'a' && ch <= 'z') {
        return ch - 'a' + 36;
      }
      return ch - '0';
    }

    private char getChar(int x) {
      if (x < 10) {
        return (char) ('0' + x);
      } else if (x < 36) {
        return (char) ('A' + x - 10);
      } else {
        return (char) ('a' + x - 36);
      }
    }
  }
}
```

7.2 杂项

7.3 I/O 优化

```
namespace FastIO {
struct Control {
  int ct, val;
  Control(int Ct, int Val = -1) : ct(Ct), val(Val) {}
  inline Control operator()(int Val) { return Control(ct, Val); }
} _endl(0), _prs(1), _setprecision(2);

const int IO_SIZE = 1 << 16 | 127;

struct FastIO {
  char in[IO_SIZE], *p, *pp, out[IO_SIZE], *q, *qq, ch[20], *t, b, K, prs;
  FastIO() : p(in), pp(in), q(out), qq(out + IO_SIZE), t(ch), b(1), K(6) {}
  ~FastIO() { fwrite(out, 1, q - out, stdout); }
  inline char getc() {
    return p == pp && (pp = (p = in) + fread(in, 1, IO_SIZE, stdin), p == pp)
               ? (b = 0, EOF)
               : *p++;
  }
  inline void putc(char x) {
    q == qq && (fwrite(out, 1, q - out, stdout), q = out), *q++ = x;
  }
  inline void puts(const char str[]) {
    fwrite(out, 1, q - out, stdout), fwrite(str, 1, strlen(str), stdout),
        q = out;
  }
  inline void getline(string &s) {
    s = "";
    for (char ch; (ch = getc()) != '\n' && b;) s += ch;
  }
#define indef(T)                                                    \
  inline FastIO &operator>>(T &x) {                                 \
    x = 0;                                                          \
    char f = 0, ch;                                                 \
    while (!isdigit(ch = getc()) && b) f |= ch == '-';              \
```

```cpp
    while (isdigit(ch)) x = (x << 1) + (x << 3) + (ch ^ 48), ch = getc();      \
    return x = f ? -x : x, *this;                                              \
  }
  indef(int);
  indef(long long);

  inline FastIO &operator>>(string &s) {
    s = "";
    char ch;
    while (isspace(ch = getc()) && b) {}
    while (!isspace(ch) && b) s += ch, ch = getc();
    return *this;
  }
  inline FastIO &operator>>(double &x) {
    x = 0;
    char f = 0, ch;
    double d = 0.1;
    while (!isdigit(ch = getc()) && b) f |= (ch == '-');
    while (isdigit(ch)) x = x * 10 + (ch ^ 48), ch = getc();
    if (ch == '.')
      while (isdigit(ch = getc())) x += d * (ch ^ 48), d *= 0.1;
    return x = f ? -x : x, *this;
  }
#define outdef(_T)                                                             \
  inline FastIO &operator<<(_T x) {                                            \
    !x && (putc('0'), 0), x < 0 && (putc('-'), x = -x);                        \
    while (x) *t++ = x % 10 + 48, x /= 10;                                     \
    while (t != ch) *q++ = *--t;                                               \
    return *this;                                                              \
  }
  outdef(int);
  outdef(long long);
  inline FastIO &operator<<(char ch) { return putc(ch), *this; }
  inline FastIO &operator<<(const char str[]) { return puts(str), *this; }
  inline FastIO &operator<<(const string &s) { return puts(s.c_str()), *this; }
  inline FastIO &operator<<(double x) {
    int k = 0;
    this->operator<<(int(x));
    putc('.');
    x -= int(x);
    prs && (x += 5 * pow(10, -K - 1));
    while (k < K) putc(int(x *= 10) ^ 48), x -= int(x), ++k;
    return *this;
  }
  inline FastIO &operator<<(const Control &cl) {
    switch (cl.ct) {
    case 0: putc('\n'); break;
    case 1: prs = cl.val; break;
    case 2: K = cl.val; break;
    }
    return *this;
  }
  inline operator bool() { return b; }
};
} // namespace FastIO
```

7.4 优化 STL 内存申请

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

```
  template <typename T2> struct rebind { typedef myalloc<T2> other; };
  inline T *allocate(size_t n) {
    T *result = (T *)sp;
    sp += n * sizeof(T);
    return result;
  }
  inline void deallocate(T *p, size_t n) {}
};
```

7.5 Emacs 配置

```
(defun myc++ ()
  (c-set-style "stroustrup")
  (setq tab-width 2)
  (setq indent-tabs-mode nil)
  (setq c-basic-offset 2)
  (c-toggle-hungry-state)
  (defun compile-and-run()
    (interactive)
    (setq file-name (file-name-sans-extension (file-name-nondirectory buffer-file-name)))
    (compile
     (format "g++ %s.cpp -o %s -Wall -Wextra -Wshadow -O2 && ./%s < in.txt"
             file-name file-name file-name)))
  (local-set-key (kbd "C-c C-c") 'compile-and-run)
  (local-set-key (kbd "C-c C-k") 'kill-compilation))
(add-hook 'c++-mode-hook 'myc++)

(global-set-key [(meta ?o)] 'other-window)
(global-set-key [(meta ?/)] 'hippie-expand)
(global-set-key [(control tab)] ' senator-completion-menu-popup)
(setq hippie-expand-try-functions-list
      '(try-expand-dabbrev
        try-expand-dabbrev-visible
        try-expand-dabbrev-all-buffers
        try-expand-dabbrev-from-kill
        try-complete-file-name-partially
        try-complete-file-name
        try-expand-all-abbrevs
        try-expand-list
        try-expand-line))

(setq auto-save-mode nil)
(setq make-backup-files nil)

(ido-mode t)
(show-paren-mode 1)
(delete-selection-mode t)
(global-linum-mode t)
(global-auto-revert-mode t)
```

7.6 Vim 配置

```
syntax enable
set syntax=on
set nobackup
set noswapfile
set noundofile
set nu
set smartindent
set cindent
set noeb
set tabstop=2
set softtabstop=2
set shiftwidth=2
```

```
set expandtab

:imap jk <Esc>

map <F5> : call Complie() <CR>
func Complie()
  exec "w"
  exec "!g++ % -o %< -g -Wall -std=gnu++14 -static"
endfunc

map <F6> : call Run() <CR>
func Run()
  exec "!./%<"
endfunc

map <F9> : call DeBug() <CR>
func DeBug()
  exec "!gdb %<"
endfunc
```

7.7 对拍

*unix 下对拍:

```
while true; do
  python gen.py > in.txt
  time ./my < in.txt > out.txt
  time ./std < in.txt > ans.txt
  if diff out.txt ans.txt; then
    echo AC
  else
    echo WA
    exit 0
  fi
done
```

Windows 下对拍:

```
@echo off
:loop
python gen.py > in.txt
my.exe < in.txt > out.txt
std.exe < in.txt > ans.txt
fc out.txt ans.txt
if not errorlevel 1 goto loop
pause
:end
```