Templates for ICPC

iNx

Contents

Chapter 1

Basics

1.1 高精度

```
struct Bign {
    int len, sgn, w[MAX_L];
    void clear() {
        len = sgn = 1;
        memset(w, 0, sizeof(w));
    }
    Bign operator=(int x) {
        clear();
        if (x != 0) {
            if (x < 0) {
                sgn = -1, x = -x;
            }
            len = 0;
            while (x) {
                w[len++] = x % 10;
                x /= 10;
            }
        }
        return *this;
    }
    Bign operator=(string s) {
        clear();
        reverse(s.begin(), s.end());
        if (s.back() == '-') {
            sgn = -1, s.pop_back();
        }
        for (int i = 0; i < s.size(); i++) {
            w[i] = s[i] - '0';
        }
        len = s.size();
        return *this;
    }
    Bign() {
        clear();
    }
    Bign(int x) {
```

```
        *this = x;
    }
    friend istream& operator>>(istream& in, Bign& a) {
        string s;
        in >> s;
        a = s;
        return in;
    }
    friend ostream& operator<<(ostream& out, const Bign& a) {
        if (a.sgn < 0) {
            out << "-";
        }
        for (int i = a.len - 1; i >= 0; i--) {
            out << a.w[i];
        }
        return out;
    }
    friend int cmp(const Bign& a, const Bign& b) {
        if (a.sgn != b.sgn) {
            return a.sgn > b.sgn ? 1 : -1;
        }
        int res = a.sgn;
        if (a.len != b.len) {
            return a.len > b.len ? res : -res;
        }
        for (int i = a.len - 1; i >= 0; i--) {
            if (a.w[i] != b.w[i]) {
                return a.w[i] > b.w[i] ? res : -res;
            }
        }
        return 0;
    }
    friend bool eq(const Bign& a, const Bign& b) {
        return cmp(a, b) == 0;
    }
    friend Bign add(const Bign& a, const Bign& b) {
        Bign c;
        c.len = max(a.len, b.len);
        for (int i = 0; i < c.len; i++) {
            if (i < a.len) {
                c.w[i] += a.w[i];
            }
            if (i < b.len) {
                c.w[i] += b.w[i];
            }
            c.w[i + 1] += c.w[i] / 10;
            c.w[i] %= 10;
        }
        if (c.w[c.len] > 0) {
            c.len++;
        }
        return c;
    }
```

```
    friend Bign sub(Bign a, Bign b) {
        Bign c;
        if (a < b) {
            swap(a, b), c.sgn = -1;
        }
        c.len = a.len;
        for (int i = 0; i < c.len; i++) {
            c.w[i] += a.w[i];
            if (i < b.len) {
                c.w[i] -= b.w[i];
            }
            if (c.w[i] < 0) {
                c.w[i] += 10, c.w[i + 1]--;
            }
        }
        while (c.w[c.len - 1] == 0) {
            c.len--;
        }
        return c;
    }
    friend Bign operator+(const Bign& a, const Bign& b) {
        return add(a, b);
    }
    friend Bign operator-(const Bign& a, const Bign& b) {
        return sub(a, b);
    }
    friend bool operator<(const Bign& a, const Bign& b) {
        return cmp(a, b) < 0;
    }
    friend bool operator>(const Bign& a, const Bign& b) {
        return cmp(a, b) > 0;
    }
    friend bool operator==(const Bign& a, const Bign& b) {
        return eq(a, b);
    }
    int remove() {
        int cnt = 0;
        while (cnt < len && w[cnt] == 0) {
            cnt++;
        }
        for (int i = cnt, j = 0; i < len; i++, j++) {
            w[j] = w[i];
        }
        for (int i = 0, j = len - 1; i < cnt; i++, j--) {
            w[j] = 0;
        }
        len -= cnt;
        return cnt;
    }
};
```

Chapter 2

Number Theory

2.1 筛法

2.1.1 线性筛 (欧拉筛)

$O(n)$. 欧拉筛还可以将可乘函数计算出来，如下面的代码计算欧拉函数 $\varphi(n)$.

```
void Euler(int n) {
    phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!vis[i]) {
            phi[i] = i - 1;
            vis[i] = true;
            pri[++cnt] = i;
        }
        for (int j = 1; j <= cnt && i * pri[j] <= n; j++) {
            vis[i * pri[j]] = true;
            if (i % pri[j] == 0) {
                phi[i * pri[j]] = phi[i] * pri[j];
                break;
            }
            phi[i * pri[j]] = phi[i] * phi[pri[j]];
        }
    }
}
```

2.1.2 杜教筛

$O(n^{\frac{2}{3}})$. 对于积性函数 $f(x)$ 的前缀和 $S(n)$，对于任意一个数论函数 $g(x)$，均有：

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

可以利用贝尔级数等找到 $g, f*g$ 都易于求和的 g.
快速求莫比乌斯函数前缀和.

```
int S_mu(int n) {
    if (n <= MAX_N) return mu[n];
    if (hsmu.count(n)) return hsmu[n];
    int res = 1;
```

```
    for (int i = 2, j; i <= n; i = j + 1) {
        j = n / (n / i);
        res -= (j - i + 1) * S_mu(n / i);
    }
    return hsmu[n] = res;
}
```

2.1.3 min25 筛

min25 筛主要分为两个步骤，首先自下而上计算一个素数函数前缀和，然后自上而下递归地计算积性函数前缀和，这个过程是类似埃氏筛的。
为了做到这一点，那么首先定义两个函数 $g(n,j)$ 和 $S(n,j)$：

$$g(n,j) = \sum_{i=1}^{n} f(i)[i \in P\text{或}i\text{的最小素因子} > p_j]$$
$$S(n,j) = \sum_{i=1}^{n} f(i)[i\text{的最小素因子} > p_j]$$

$f(x)$ 为要求前缀和的积性函数，p_j 表示第 j 个素数，P 为小于等于 n 的素数组成的集合，特别的我们定义 $p_0 = 1$。为什么要这么做呢？首先看第一个问题，要把素数函数的前缀和筛出来，如果利用埃氏筛，其时间复杂度为 $O(n \log\log n)$。仔细思考其过程，如果筛到一个素数，要把它所有的倍数全部划去，但是很显然如果这个素数大于 $\lfloor\sqrt{n}\rfloor$ 的时候，后面根本没有数要被划去！但是由于要筛出所有素数，这一步骤必须进行；再看现在的问题，如果要筛出前缀和，那么不需要处理出所有素数，也就是说，如果先把所有的数和算出来，然后用素数倍数一个个把不符合的划去，就可以计算出**素数函数的前缀和**了。而由于大于根号的素数对答案不会产生贡献（不会划去任何数），所以我们只要预处理前面 $\lfloor\sqrt{n}\rfloor$ 的素数就行了。利用素数分布的结果大概可以知道这个过程差不多是 $O(\frac{n}{\log n})$ 的。
接下来处理细节，考虑 $g(n,j)$ 的转移：由刚才的分析，$g(n,j)$ 显然要由 $g(n,j-1)$ 转移过来，前一个转态划去了所有素因子小于等于 p_{j-1} 的，接下来显然要把最小素因子为 p_j 的划去。因为需要划去的数其最小的素数已经是 p_j 了，所以我们可以发现这些数的和就是 $g(\frac{n}{p_j}, j-1)$，但是需要注意，这里面不止包含了素因子大于等于 p_j 的数，还有所有素数，因此需要把小于 p_j 的素数加回来。故有：

$$g(n,j) = g(n,j-1) - f(p_j) \times \left(g(\frac{n}{p_j}, j-1) - g(p_{j-1}, j-1)\right)$$

由于如果有必要转移的话（就是如果 $p_j^2 > n$ 时，$g(n,j) = g(n,j-1)$），所以 $g(p_{j-1}, j-1)$ 也就是前 $(j-1)$ 个素数函数前缀和是可以预处理的时候算出来的。所以转移需要用到的状态是 $g(n,j-1), g(\frac{n}{p_j}, j-1)$，那么很明显这个可以用**滚动数组**进行优化，并且注意到所有对答案有贡献的状态 $g(n',j)$ 中，n' 必然是某个 $\lfloor\frac{n}{i}\rfloor$，由**整除分块**的知识知道有用的转态只有 $O(\sqrt{n})$ 个。
考虑第二个问题，也是类似的，我们考虑它的转移。因为以我们的定义答案就是 $S(n,0)+1$，所以这个转移方向和 g 是相反的。考虑 $S(n,j)$，可以认为其值由两部分组成，一部分是素数的和，这个由前面已经计算出来的 g 值就可以直接统计出答案，所以问题变成合数部分如何求解。我们考虑该合数的最小素数，那么由 $S(n,j)$ 的定义，这个素数是 $p_{j+1}, p_{j+2}\ldots$，我们枚举它的幂次 e，暴力由 $S(\frac{n}{p_k^e}, k)$ 转移过来，但是需要注意，这时候 $p_k^e(e>1)$ 同样被删去了，把它加回来就行。故有：

$$S(n,j) = g(n,|P|) - \sum_{i=1}^{j} f(p_i) + \sum_{k=j+1}^{|P|\text{且}p_k^2 \le n} \sum_{e=1}^{\infty} \left(f(p_k^e) \times (S(\frac{n}{p_k^e}, k) + (e > 1))\right)$$

这个时间复杂度不会分析，据说是亚线性 $O(n^{1-\varepsilon})$。

$O(n^{1-\varepsilon})$.

```cpp
// luogu P5325 f(p^k) = p^k(p^k-1)
#include <bits/stdc++.h>
#define int long long

const int MOD = 1e9 + 7;
const int MAX_SQRT = 2e5 + 7;
int pri[MAX_SQRT], id1[MAX_SQRT << 1], id2[MAX_SQRT], w[MAX_SQRT], sp2[MAX_SQRT],
    sp1[MAX_SQRT], g2[MAX_SQRT], g1[MAX_SQRT];
bool vis[MAX_SQRT];
int n, cnt, sqr, tot;

inline int add(const int &x, const int &y) {
   return (x + y) >= MOD ? x + y - MOD : x + y;
}

inline int dec(const int &x, const int &y) {
   return (x - y) < 0 ? x - y + MOD : x - y;
}

void Euler(int n) {
   pri[0] = 1;
   for (int i = 2; i <= n; ++i) {
      if (!vis[i]) {
         pri[++cnt] = i;
         sp2[cnt] = add(sp2[cnt - 1], i * i % MOD);
         sp1[cnt] = add(sp1[cnt - 1], i);
      }
      for (int j = 1; j <= cnt && i * pri[j] <= n; ++j) {
         vis[i * pri[j]] = true;
         if (i % pri[j] == 0) break;
      }
   }
}

int f_pow(int base, int b, int mod = MOD) {
   int res = 1;
   while (b) {
      if (b & 1) res = res * base % mod;
      base = base * base % mod;
      b >>= 1;
   }
   return res;
}

inline int f(const int &p, const int &e) {
   int tmp = f_pow(p, e);
   return tmp * (tmp - 1) % MOD;
}

inline int g(const int &k) { return dec(g2[k], g1[k]); }
```

```cpp

inline int sp(const int &y) { return dec(sp2[y], sp1[y]); }

int S(int x, int y) {
    if (pri[y] >= x) return 0;
    int k = (x <= sqr) ? id1[x] : id2[n / x], res = dec(g(k), sp(y));
    for (int i = y + 1; i <= cnt && pri[i] * pri[i] <= x; ++i) {
        for (int e = 1, prod = pri[i]; prod <= x; ++e, prod *= pri[i]) {
            int tmp = prod % MOD;
            res = add(res, tmp * (tmp - 1) % MOD * (S(x / prod, i) + (e != 1)) % MOD);
            // res = add(res, f(pri[i], e) * (S(n / prod, i) + (e != 1)) % MOD);
        }
    }
    return res < 0 ? res + MOD : res;
}

int min_25(const int &n) {
    sqr = sqrt(n);
    Euler(sqr);
    int inv6 = f_pow(6ll, MOD - 2);
    for (int i = 1, j, tmp; i <= n; i = j + 1) {
        j = n / (n / i);
        w[++tot] = tmp = n / i;
        if (tmp >= MOD) tmp %= MOD;
        g1[tot] = dec(tmp * (tmp + 1) / 2 % MOD, 1);
        g2[tot] = dec(tmp * (tmp + 1) % MOD * (tmp + tmp + 1) % MOD * inv6 % MOD, 1);
        if (w[tot] <= sqr) id1[w[tot]] = tot;
        else id2[n / w[tot]] = tot;
    }
    for (int i = 1; i <= cnt; ++i) {
        for (int j = 1; j <= tot && pri[i] * pri[i] <= w[j]; ++j) {
            int k = (w[j] / pri[i] <= sqr) ? id1[w[j] / pri[i]] : id2[n / (w[j] / pri[i])];
            g1[j] = dec(g1[j], pri[i] * dec(g1[k], sp1[i - 1]) % MOD);
            g2[j] = dec(g2[j], pri[i] * pri[i] % MOD * dec(g2[k], sp2[i - 1]) % MOD);
        }
    }
    return add(S(n, 0), 1);
}

signed main() {
    scanf("%lld", &n);
    printf("%lld\n", min_25(n));
    return 0;
}
```

2.2 Euclid 算法

2.2.1 最大公约数算法

$O(\log n)$.

```cpp
int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
```

2.2.2 扩展欧几里得

$O(\log n)$，求裴蜀定理系数.

$$ax_0 + by_0 = d$$

则不定方程通解为:

$$\begin{cases} x = (x_0 + \frac{b}{d}t) \\ y = (y_0 - \frac{a}{d}t) \end{cases}$$

```cpp
int ex_gcd(int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int res = ex_gcd(b, a % b, x, y);
    int tmp = x; x = y; y = tmp - a / b * y;
    return res;
}
```

2.2.3 类欧几里得

求两个分数中间分子最小的分数.
$O(\log n)$.

```cpp
pii find(int a, int b, int c, int d) {
    if ((a + b - 1) / b <= c / d) return pii((a + b - 1) / b, 1);
    int t = a / d;
    pii tmp = find(d, c - d * t, b, a - t * b);
    return pii(tmp.second + t * tmp.first, tmp.first);
}
```

2.3 中国剩余定理 (CRT)

2.3.1 CRT

$n \log \mathrm{MAX}$.

```cpp
inline int crt(int a, int m, int M) {
    return a * inv(M / m, m) % M * (M / m) % M;
}
```

2.3.2 EX-CRT

对方程进行两两合并，由于若干个方程的解为一个特解的关于最小公倍数的剩余类，所以合并是正确的。对于当前要合并的两个方程:

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \end{cases}$$

也即

$$\begin{cases} x = m_1t_1 + a_1 \\ x = m_2t_2 + a_2 \end{cases}$$

两式相减即得

$$m_1t_1 - m_2t_2 = a_2 - a_1 = c$$

那么很明显，若 $(m_1, m_2) \nmid c$ 则方程无解。否则也就是要求解同余方程：

$$m_1t_1 \equiv c(\mod m_2)$$

扩展欧几里得即可。
$O(n \log \mathrm{MAX})$.

```
int ex_crt(int *a, int *m, int n) {
    int x, y, m1 = 1, ans = 0;
    for (int i = 1; i <= n; ++i) {
        int m2 = m[i], c = (a[i] - ans) % m2, _gcd = ex_gcd(m1, m2, x, y);
        if (c % _gcd) return -1;
        x = (x * (c / _gcd)) % m2;
        ans += x * m1;
        m1 *= m2 / _gcd;
        ans %= m1;
    }
    return ans < 0 ? ans + m1 : ans;
}
```

2.4 Lucas 定理

2.4.1 Lucas

定理 2.4.1 (Lucas's Theorem).

$$C_a^b \equiv C_{\lfloor \frac{a}{p} \rfloor}^{\lfloor \frac{b}{p} \rfloor} \cdot C_{a \mod p}^{b \mod p} (\mod p)$$

Lucas 定理揭示了一个组合数取模可以将 m, n 分别写成 p 进制数再进行计算。其主要原因是由于

$$p \mid C_p^n (n = 1, 2, \ldots, p-1)$$

因而有

$$(1+x)^p \equiv 1 + x^p (\mod p)$$

由带余数除法将 m, n 写成：

$$\begin{cases} m = pq_m + r_m \\ n = pq_n + r_n \end{cases}$$

于是有

$$\begin{aligned}(1+x)^m &= (1+x)^{pq_m+r_m}\\ &= [(1+x)^p]^{q_m}\cdot(1+x)^{r_m}\\ &\equiv (1+x^p)^{q_m}\cdot(1+x)^{r_m}\end{aligned}$$

考虑左边 x^n 的系数 C_m^n；而右边第二项由于 $r_m < p$，而第一项全是 p 的若干倍数，故只能是第一项取 q_n，第二项取 r_n，于是有：

$$C_m^n = C_{\lfloor \frac{m}{p} \rfloor}^{\lfloor \frac{n}{p} \rfloor}\cdot C_{m \mod p}^{n \mod p}$$

Lucas 定理另一形式：令 $m = m_0 + m_1 p + \ldots + m_d p^d, n = n_0 + n_1 p + \ldots + n_d p^d$，则：

$$C_m^n \equiv C_{m_0}^{n_0}\cdot C_{m_1}^{n_1}\cdot \ldots \cdot C_{m_d}^{n_d} (\mod p)$$

推论 2.4.1.

$$C_n^p \equiv \lfloor \frac{n}{p} \rfloor (\mod p)$$

定理 2.4.2 (Fine's Theorem).

$$n = n_0 + n_1 p + n_2 p^2 + \ldots + n_d p^d$$

则 $C_n^k (k = 0, 1, \ldots, n)$ 中，有 $\prod_{i=1}^d (1 + n_i)$ 个数不被 p 整除。

如果预处理前 p 个数的阶乘及其逆元，则时间复杂度为 $O(p)$.

```
void init(int p) {
    fac[0] = 1;
    for (int i = 1; i < p; ++i) fac[i] = fac[i - 1] * i % p;
    inv[p - 1] = f_pow(fac[p - 1], p - 2, p);
    for (int i = p - 1; i; --i) inv[i - 1] = i * inv[i] % p;
}

inline int C(int n, int m, int p) {
    return n >= m ? fac[n] * inv[n - m] % p * inv[m] % p : 0;
}

int lucas(int n, int m, int p) {
    return m ? C(n % p, m % p, p) * lucas(n / p, m / p, p) % p: 1;
}
```

2.4.2 EX-Lucas

扩展 Lucas 定理是将 p 进行分解：

$$p = \prod_{i=1}^{k} p_i^{\alpha_i}$$

然后计算 $C_n^m \equiv a(\mod p^{\alpha_i})$，最后中国剩余定理进行合并。考虑

$$\frac{n!}{m!(n-m)!} = \frac{\frac{n!}{p^a}}{\frac{m!}{p^b}\frac{(n-m)!}{p^c}}\cdot p^{a-b-c}$$

所以考虑

$$\frac{n!}{p^a}\%p^\alpha$$

稍加观察可以发现上式

$$\frac{n!}{p^a} \equiv \lfloor\frac{n}{p}\rfloor! \cdot \left(\prod_{i=1\text{且}(i,p)=1}^{p^\alpha} i\right)^{\lfloor\frac{n}{p^\alpha}\rfloor} \cdot \left(\prod_{i=1\text{且}(i,p)=1}^{(n\%p^\alpha)} i\right) \pmod{p^\alpha}$$

$O(p\log_p n)$

```
#include <bits/stdc++.h>
#define int long long

using namespace std;

int n, m, p;

inline int add(const int &x, const int &y, const int &mod) {
    return x + y >= mod ? x + y - mod : x + y;
}
inline int dec(const int &x, const int &y, const int &mod) {
    return x - y < 0 ? x - y + mod : x - y;
}

int f_pow(int base, int b, int mod) {
    int res = 1;
    while (b) {
        if (b & 1) res = res * base % mod;
        base = base * base % mod;
        b >>= 1;
    }
    return res;
}

int ex_gcd(int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int res = ex_gcd(b, a % b, x, y);
    int tmp = x; x = y; y = tmp - a / b * y;
    return res;
}

inline int inv(int a, int mod) {
    int x, y;
    ex_gcd(a, mod, x, y);
    return (x %= mod) < 0 ? x + mod : x;
}

int fac(int n, int p, int pk) {
    if (!n) return 1;
```

```
    int res = 1;
    for (int i = 2; i < pk; ++i) if (i % p)
        res = res * i % pk;
    res = f_pow(res, n / pk, pk);
    for (int i = 2; i <= n % pk; ++i) if (i % p)
        res = res * i % pk;
    return res * fac(n / p, p, pk) % pk;
}

int C(int n, int m, int pk, int p) {
    int k = 0;
    for (int i = n; i;) k += (i /= p);
    for (int i = m; i;) k -= (i /= p);
    for (int i = n - m; i;) k -= (i /= p);
    return fac(n, p, pk) * inv(fac(m, p, pk), pk) % pk * inv(fac(n - m, p, pk), pk)
         % pk * f_pow(p, k, pk) % pk;
}

inline int crt(int a, int m, int M) {
    return a * inv(M / m, m) % M * (M / m) % M;
}

int ex_lucas(int n, int m, int p) {
    if (n < m) return 0;
    int res = 0, t = p;
    for (int i = 2, pk; i * i <= p; ++i) {
        if (t % i) continue;
        pk = 1;
        while (t % i == 0) t /= i, pk *= i;
        res = add(res, crt(C(n, m, pk, i), pk, p), p);
    }
    if (t > 1) res = add(res, crt(C(n, m, t, t), t, p), p);
    return res;
}

signed main() {
    scanf("%lld%lld%lld", &n, &m, &p);
    printf("%lld\n", ex_lucas(n, m, p));
    return 0;
}
```

2.5 原根

寻找最小原根，$O(n^{0.25}\log n + \sqrt{n})$.

```
void getPrime(int n) {
    cntn = 0;
    int x = n;
    for (int i = 2; i * i <= n; i++) {
        if (x % i == 0) {
            npri[++cntn] = i;
            while (x % i == 0) x /= i;
```

```
        }
    }
    if (x > 1) npri[++cntn] = x;
}

bool chk(int g, int n) {
    if (f_pow(g, phi[n], n) != 1) return false;
    getPrime(phi[n]);
    for (int i = 1; i <= cntn; i++) {
        if (f_pow(g, phi[n] / npri[i], n) == 1) return false;
    }
    return true;
}

int findG(int n) {
    for (int i = 1; i < n; i++) {
        if (chk(i, n)) return i;
    }
    return -1;
}
```

寻找所有原根，$O(\varphi(n)log\varphi(n))$.

```
void findG(int g, int n) {
    int base = g, prod = g;
    for (int i = 2; i <= phi[n]; i++) {
        prod = g * prod % n;
        if (gcd(i, phi[n]) == 1) prt[++num] = prod;
    }
}
```

2.6 离散对数问题 (DLP)

2.6.1 BSGS(Baby-step Giant-step)

BSGS 算法，$O(\sqrt{p})$ 求解满足 $a^x \equiv b(\mod p), (a,p) = 1$ 的最小自然数解 x.
算法原理：注意到当 a，p 互素时，令 $x = A\lceil p\rceil - B(0 \leq B < \lceil p\rceil)$，则原问题等价于

$$a^{A*\lceil p\rceil} \equiv ba^B(\mod p)$$

先 $\sqrt{p}$ 下枚举 B，将 ba^B 模 p 存到 hash 里面，再 $\sqrt{p}$ 下枚举 A 即可.

```
int bsgs(int a, int p, int b) {
    int nsqrt = (long long)sqrt(p) + 1, base = f_pow(a, nsqrt, p);
    unordered_map <int, int> mp;
    for (int i = 0, prod = b; i < nsqrt; i++, prod = (prod * a) % p) {
        mp[prod] = i;
    }
    for (int i = 1, prod = base; i <= nsqrt; i++, prod = (prod * base) % p) {
        if (mp.count(prod)) {
            return i * nsqrt - mp[prod];
        }
    }
```

```
    return -1;
}
```

扩展 BSGS 算法，$O(\sqrt{p})$ 求离散对数，但是不要求 $(a,p)=1$. 找到最大的 k，使得 $(a^k,p)>1$，如果 b 不能被最大公约数整除，方程无解，否则把逆元乘到后面再做 BSGS 即可.

```
int ex_bsgs(int a, int p, int b) {
    int k = 0, d, x, y, down = 1;
    if (b == 1) return 0;
    while ((d = ex_gcd(a, p, x, y)) != 1) {
        if (b % d) return -1;
        k++, b /= d, p /= d, down = (down * a / d) % p;
        if (down == b) return k;
    }
    ex_gcd(down, p, x, y);
    b = (b * x % p + p) % p;
    return (d = bsgs(a, p, b)) < 0 ? -1 : d + k;
}
```

Chapter 3

Math

3.1 公式（Formula）

3.1.1 切/曼距离

切比雪夫距离转化成曼哈顿距离:

$$\begin{cases} x' = & \frac{x+y}{2} \\ y' = & \frac{x-y}{2} \end{cases}$$

曼哈顿距离转化成切比雪夫距离:

$$\begin{cases} x' = & x+y \\ y' = & x-y \end{cases}$$

3.1.2 贝尔级数

定义数论函数 f 在模素数 p 意义下的贝尔级数:

$$f_p(x) = \sum_{i=0}^{\infty} f(p^i)x^i$$

则有:

$$(f * g)_p = f_p \times g_p$$

即可以用级数乘法来刻画卷积，这点和拉普拉斯变换有点类似.

常用数论函数的贝尔级数

$$\begin{aligned}
&\mu_p(x) = 1 - x \\
&\varphi_p(x) = \frac{1-x}{1-px} \\
&1_p(x) = \frac{1}{1-x} \\
&\varepsilon_p(x) = 1 \\
&ID_p(x) = \frac{1}{1-px} \\
&ID_{k_p}(x) = \frac{1}{1-p^k x} \\
&\mu_p^2(x) = 1 + x \\
&(ID \cdot \mu) = 1 - px \\
&\sigma_p(x) = (1 * ID)_p(x) = (1_p \times ID_p)(x) = \frac{1}{(1-x)(1-px)} \\
&\sigma_{k_p}(x) = \frac{1}{(1-x)(1-p^k x)} \\
&\lambda_p(x) = \frac{1}{1+x}
\end{aligned}$$

3.2 数值积分

辛普森积分，O(能过)...

```
double simpson(double a, double b) {
    double c = (a + b) / 2;
    return (f(a) + f(b) + 4 * f(c)) * (b - a) / 6;
}
double asr(double a, double b, double eps, double A) {
    double c = (a + b) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if (fabs(L + R - A) < 15 * eps)
        return (L + R + (L + R - A) / 15.0);
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
double asr(double a, double b, double eps) { return asr(a, b, eps, simpson(a, b)); }
```

3.3 康拓展开

对于排列 $b_1 b_2 \dots b_n$，其排名：

$$X = \sum_{i=1}^{n} a_i (n-i)! + 1$$

a_i 代表后面有多少个小于当前元素的元素个数。

$O(n \log n)$。(树状数组优化)

```
int cantor(int *a, int n) {
    int res = 1;
```

```
    for (int i = 1; i <= n; ++i) {
        int cnt = get_sum(a[i] - 1);
        add(a[i], -1);
        res = (res + cnt * fac[n - i] % MOD) % MOD;
    }
    return res;
}
```

逆康托展开：$O(n\log^2 n)$，树状数组优化版本。（实际上一般 n 都很小，暴力即可）

```
vector<int> decantor(int x, int n) {
    vector<int> res;
    for (int i = 1; i <= n; ++i) {
        int pos = find_pos(x / fac[n - i]);
        x %= fac[n - 1];
        res.push_back(pos);
        add(pos, -1);
    }
    return res;
}
```

Chapter 4

Polynomial

4.1 快速傅立叶变换 FFT

$O(n \log n)$.

```
//luogu P3803 【模板】多项式乘法（FFT）
#include <iostream>
#include <cstdio>
#include <cstring>
#include <complex>
#include <algorithm>
#include <cmath>
#define PI 3.141592653589

using namespace std;

typedef complex <double> cd;
const int N = 3e6 + 7;
int rev[N];
cd f[N], g[N];
int n, m;

void fft(cd *a, int n, int dft) {
    for(int i = 0; i < n; i++) {
        if(i < rev[i]) swap(a[i], a[rev[i]]);
    }
    for(int i = 1; i < n; i <<= 1) {
        cd wn = exp(cd(0, 1.0 * dft * PI / i));
        for(int j = 0; j < n; j += (i << 1)) {
            cd wnk = cd(1, 0);
            for(int k = j; k < j + i; k++) {
                cd a1 = a[k], a2 = a[k + i];
                a[k] = a1 + wnk * a2;
                a[k + i] = a1 - wnk * a2;
                wnk *= wn;
            }
        }
    }
    if(dft == -1) {
        for(int i = 0; i < n; i++) a[i] /= n;
```

```
    }
}
int main() {
    scanf("%d%d", &n, &m);
    for(int i = 0; i <= n; i++) {
        int x; scanf("%d", &x);
        f[i] = x;
    }
    for(int i = 0; i <= m; i++) {
        int x; scanf("%d", &x);
        g[i] = x;
    }
    int N = 1, p = 0;
    while(N < (m + n + 1)) N <<= 1, p++;
    for(int i = 0; i < N; i++) rev[i] = ((rev[i >> 1] >> 1) | ((i & 1) << (p - 1)));
    fft(f, N, 1);
    fft(g, N, 1);
    for(int i = 0; i < N; i++) {
        f[i] *= g[i];
    }
    fft(f, N, -1);
    for(int i = 0; i <= (n + m); i++) {
        printf("%d ", (int)(f[i].real() + 0.5));
    }
    puts("");
    return 0;
}
```

4.2 快速数论变换 NTT

$O(n \log n)$.

```
//luogu P3803 【模板】多项式乘法（FFT）
#include <iostream>
#include <cstdio>
#include <cstring>
#include <complex>
#include <algorithm>
#include <cmath>
#define int long long
#define G 3

using namespace std;

const int N = 3e6 + 7;
const int MOD = 998244353;
int rev[N];
int f[N], g[N];
int n, m;

int f_pow(int base, int b, int mod) {
    int res = 1;
```

```
    while(b) {
        if(b & 1) res = res * base % mod;
        base = base * base % mod;
        b >>= 1;
    }
    return res;
}
void ntt(int *a, int n, int dft) {
    for(int i = 0; i < n; i++) {
        if(i < rev[i]) swap(a[i], a[rev[i]]);
    }
    for(int i = 1; i < n; i <<= 1) {
        int wn = f_pow(G, (MOD - 1) / (i << 1), MOD);
        if(dft < 0) wn = f_pow(wn, MOD - 2, MOD);
        for(int j = 0; j < n; j += (i << 1)) {
            int wnk = 1;
            for(int k = j; k < j + i; k++) {
                int a1 = a[k], a2 = a[k + i];
                a[k] = (a1 + wnk * a2 % MOD) % MOD;
                a[k + i] = (a1 - wnk * a2 % MOD) % MOD;
                wnk = wnk * wn % MOD;
            }
        }
    }
    if(dft == -1) {
        int inv = f_pow(n, MOD - 2, MOD);
        for(int i = 0; i < n; i++) a[i] = a[i] * inv % MOD;
    }
}
signed main() {
    scanf("%lld%lld", &n, &m);
    for(int i = 0; i <= n; i++) scanf("%lld", f + i);
    for(int i = 0; i <= m; i++) scanf("%lld", g + i);
    int N = 1, p = 0;
    while(N < (m + n + 1)) N <<= 1, p++;
    for(int i = 0; i < N; i++) rev[i] = ((rev[i >> 1] >> 1) | ((i & 1) << (p - 1)));
    ntt(f, N, 1);
    ntt(g, N, 1);
    for(int i = 0; i < N; i++) {
        f[i] *= g[i];
    }
    ntt(f, N, -1);
    for(int i = 0; i <= (n + m); i++) printf("%lld ", (f[i] + MOD) % MOD);
    puts("");
    return 0;
}
```

Chapter 5

Geometry

5.1 基础知识

5.1.1 Point 类

```
template<class T> int sgn(T x) { return (x > 0) - (x < 0); }
struct Point {
    double x, y;
    Point(double _x = 0, double _y = 0) : x(_x), y(_y) {}
    Point operator+(const Point &p) const {
        return Point(x + p.x, y + p.y);
    }
    Point operator-(const Point &p) const {
        return Point(x - p.x, y - p.y);
    }
    Point operator*(double w) const {
        return Point(x * w, y * w);
    }
    Point operator/(double w) const {
        return Point(x / w, y / w);
    }
    bool operator==(const Point &p) const {
        return (!sgn(x - p.x)) && (!sgn(y - p.y));
    }
    bool operator<(const Point &p) const {
        return (!sgn(x - p.x)) ? x < p.x : y < p.y;
    }
    Point unit() const { return *this / sqrt(x * x + y * y); }
    Point perp() const { return Point(-y, x); }
    Point normal() const { return perp().unit(); }
    void print() {
        printf("(%lf %lf)\n", x, y);
    }
};
double dist2(const Point &p) {
    return p.x * p.x + p.y * p.y;
}
double dist(const Point &p) {
    return sqrt(p.x * p.x + p.y * p.y);
```

```
}
double dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}
double cross(const Point &a, const Point &b) {
    return a.x * b.y - b.x * a.y;
}
Point rotate(const Point &a, double theta) {
    return Point(a.x * cos(theta) - a.y * sin(theta), a.y * cos(theta) + a.x * sin(
        theta));
}
double angle(const Point &p) {
    return atan2(p.y, p.x);
}
pair<int, Point> line_inter(Point &s1, Point &e1, Point &s2, Point &e2) {
    double d = cross(e1 - s1, e2 - s2);
    if (!sgn(d)) return {-(sgn(cross(e1 - s1, e2 - s1)) == 0), Point(0, 0)};
    double p = cross(e1 - s2, e2 - s2), q = cross(e2 - s2, s1 - s2);
    return {1, (s1 * p + e1 * q) / d};
}
```

5.1.2 定比分点（求两直线交点）

坐标上 A, B 连线上有一点 P 满足：

$$\overrightarrow{AP} = \lambda \overrightarrow{PB}$$

则有：

$$P = \frac{A + \lambda B}{1 + \lambda}$$

5.1.3 Line 类

5.1.4 距离

```
// 左正右负
double line_dist(Point &s, Point &e, Point &p) {
    return cross(e - s, p - s) / dist(e - s);
}

double seg_dist(Point &s, Point &e, Point &p) {
    if (s == e) return dist(p - s);
    double d = dist2(e - s), t = min(d, max(0.0, dot(p - s, e - s)));
    return dist((p - s) * d - (e - s) * t) / d;
}
```

5.1.5 判断点在线上

```
bool on_seg(Point &s, Point &e, Point &p) {
    return seg_dist(s, e, p) < EPS;
}

bool on_line(Point &s, Point &e, Point &p) {
```

```
    return line_dist(s, e, p) < EPS;
}
```

5.1.6 交点

```
vector<Point> seg_inter(Point &s1, Point &e1, Point &s2, Point &e2) {
    double oa = cross(e2 - s2, s1 - s2), ob = cross(e2 - s2, e1 - s2);
    double oc = cross(e1 - s1, s2 - s1), od = cross(e1 - s1, e2 - s1);
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0) {
        return {(s1 * ob - e1 * oa) / (ob - oa)};
    }
    set<Point> s;
    if (on_seg(s2, e2, s1)) s.insert(s1);
    if (on_seg(s2, e2, e1)) s.insert(e1);
    if (on_seg(s1, e1, s2)) s.insert(s2);
    if (on_seg(s1, e1, e2)) s.insert(e2);
    return {s.begin(), s.end()};
}

//1: 一个交点, -1: 无穷多, 0: 无交点
pair<int, Point> line_inter(Point &s1, Point &e1, Point &s2, Point &e2) {
    double d = cross(e1 - s1, e2 - s2);
    if (!sgn(d)) return {-(sgn(cross(e1 - s1, e2 - s1)) == 0), Point(0, 0)};
    double p = cross(e1 - s2, e2 - s2), q = cross(e2 - s2, s1 - s2);
    return {1, (s1 * p + e1 * q) / d};
}

Point line_inter(Line &l1, Line &l2) {
    return line_inter(l1[0], l1[1], l2[0], l2[1]).second;
}
```

5.1.7 与直线位置

左边返回 1，右边返回 -1，在直线上返回 0。

```
int side_of(const Point &s, const Point &e, const Point &p) {
    double a = cross(e - s, p - s), l = dist(e - s) * EPS;
    return (a > l) - (a < -l);
}
```

5.1.8 线性变换

```
Point linear_tran(Point &p0, Point &p1, Point &q0, Point &q1, Point &r) {
    Point dp = p1 - p0, dq = q1 - q0, num(cross(dp, dq), dot(dp, dq));
    return q0 + Point(cross(r - p0, num), dot(r - p0, num)) / dist2(dp);
}
```

5.1.9 对称

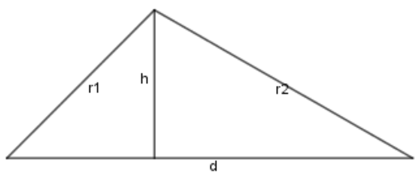
```cpp
// 轴对称
Point sysmmetry(Point &s, Point &e, Point &r) {
    if (s == e) return s * 2 - r;
    Point p = e - s, q = r - s, num(dot(p, q), cross(p, q));
    return s + Point(dot(num, p), cross(num, p)) / dist2(p);
}
// 中心对称
Point sysmmetry(Point &s, Point &r) {
    return sysmmetry(s, s, r);
}
```

5.2 圆（Circle）

5.2.1 Circle 类

```cpp
struct Circle {
    Point o; double r;
    Circle(Point _o = {0, 0}, double _r = 0) {
        o = _o, r = _r;
    }
    int position(const Point &p)const {
        return sgn(r - dist(p - o));
    }
};
```

5.2.2 两圆交点



如图有：

$$
\begin{aligned}
&\sqrt{r_1^2 - h^2} + \sqrt{r_2^2 - h^2} = d \\
&\Rightarrow r_1^2 - r_2^2 = d(x_1 - x_2) \\
&\Rightarrow x_1 + x_2 = d \\
&\Rightarrow x_1 = \frac{r_1^2 - r_2^2 + d^2}{2d}
\end{aligned}
$$

```
pair<bool, pair<Point, Point>> circle_inter(Circle &a, Circle &b) {
    if (a.o == b.o) return {false, {Point(0, 0), Point(0, 0)}};
    Point d = b.o - a.o;
    double d2 = dist2(d), sum = a.r + b.r, dif = a.r - b.r,
           p = (d2 + a.r * a.r - b.r * b.r) / (d2 * 2), h2 = a.r * a.r - p * p * d2;
    if (sum * sum < d2 || dif * dif > d2) return {false, {Point(0, 0), Point(0, 0)
        }};
    Point mid = a + d * p, per = d.perp() * sqrt(max(0.0, h2) / d2);
    return {true, {mid + per, mid - per}};
}
```

5.2.3 两圆公切线

可以返回 $0, 1, 2$ 三条切线，0 表示没有切线，1 表示两圆相切，2 会返回两条外公切线，并且当 $b.r < 0$ 时返回内公切线。公切线用切点表示。

```
vector<pair<Point, Point>> circle_tan(Circle &a, Circle &b) {
    Point d = b.o - a.o;
    double dr = a.r - b.r, d2 = dist2(d), h2 = d2 - dr * dr;
    if (d2 < EPS || h2 < 0) return {};
    vector<pair<Point, Point>> res;
    for (double sign : {-1, 1}) {
        Point v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
        res.push_back({a.o + v * a.r, b.o + v * b.r});
    }
    if (h2 < EPS) res.pop_back();
    return res;
}
```

5.2.4 三角形外接圆

张角定理可证。

```
Circle circumcircle(Point &A, Point &B, Point &C) {
    Point b = C - A, c = B - A;
    Point out = (b * dist2(c) - c * dist2(b)).perp() / cross(b, c) / 2;
    return Circle(A + out, dist(out));
}
```

5.2.5 最小圆覆盖（Minimum Enclosing Circle）

期望复杂度：$O(n)$。

```
Circle mec(vector<Point> ps) {
    shuffle(ps.begin(), ps.end(), mt19937(time(0)));
    Circle C(ps[0], 0);
    for (int i = 0; i < ps.size(); ++i) if (C.position(ps[i]) == -1) {
        C = {ps[i], 0};
        for (int j = 0; j < i; ++j) if (C.position(ps[j]) == -1) {
            C = {(ps[i] + ps[j]) / 2, dist(ps[j] - ps[i]) / 2};
            for (int k = 0; k < j; ++k) if (C.position(ps[k]) == -1) {
                C = circumcircle(ps[i], ps[j], ps[k]);
            }
        }
    }
    return C;
}
```

5.3 多边形（Polygon）

5.3.1 面积（Area）

$O(n)$。

```
// 逆正顺负
double get_area(vector<Point> &p) {
    double res = 0;
    int n = p.size();
    if (n < 3) return res;
    for (int i = 1; i < n - 1; ++i) {
        res += cross(p[i] - p[0], p[i + 1] - p[0]);
    }
    return res / 2;
}

double get_area(vector<Line> &L) {
    vector<Point> p;
    for (int i = 0; i < L.size(); ++i) {
        p.push_back(line_inter(L[i], L[(i + 1) % L.size()]));
    }
    return get_area(p);
}
```

5.3.2 凸包（Convex Hull）

Andrew 算法（$O(n \log n)$）。

```
vector<Point> convex_hull(vector<Point> p) {
    sort(p.begin(), p.end());
    vector<Point> h(p.size() + 1);
    h[0] = p[0];
    int top = 1;
    for (int i = 1; i < p.size(); ++i) {
        while (top > 1 && cross(h[top - 1] - h[top - 2], p[i] - h[top - 1]) <= 0) --
            top;
```

```cpp
        h[top++] = p[i];
    }
    int down = top;
    for (int i = p.size() - 2; i; --i) {
        while (top > down && cross(h[top - 1] - h[top - 2], p[i] - h[top - 1]) <= 0)
            --top;
        h[top++] = p[i];
    }
    return {h.begin(), h.begin() + top};
}
```

5.3.3 旋转卡壳

$O(n \log n)$。

```cpp
pair<Point, Point> hull_diameter(vector<Point> h) {
    int n = h.size(), j = n < 2 ? 0 : 1;
    pair<double, pair<Point, Point>> res({0, {h[0], h[0]}});
    for (int i = 0; i < j; ++i) {
        for (;; j = (j + 1) % n) {
            res = max(res, {dist2(h[i] - h[j]), {h[i], h[j]}});
            if (cross(h[(j + 1) % n] - h[j], h[i + 1] - h[i]) >= 0) break;
        }
    }
    return res.second;
}
```

5.3.4 半平面交

$O(n \log n)$。

```cpp
// 求半平面交 ，半平面是逆时针方向 ，输出按照逆时针
vector<Line> half_plane_inter(vector<Line> &L){
    sort(L.begin(), L.end());
    deque<Line> q;
    vector<Line> res;
    for (int i = 0; i < L.size(); ++i){
        if (i && same_dir(L[i], L[i - 1])) continue;
        while (q.size() > 1 && (!check_pos(q[q.size() - 2], q[q.size() - 1], L[i])))
            q.pop_back();
        while (q.size() > 1 && (!check_pos(q[1], q[0], L[i]))) q.pop_front();
        q.push_back(L[i]);
    }
    while (q.size() > 2 && (!check_pos(q[q.size() - 2], q[q.size() - 1], q[0]))) q.
     pop_back();
    while (q.size() > 2 && (!check_pos(q[1], q[0], q[q.size() - 1]))) q.pop_front();
    for (int i = 0; i < q.size(); ++i) res.push_back(q[i]);
    return res;
}
```

Chapter 6

Linear Algebra

6.1 异或线性基

$O(\log n)$。

```cpp
bool insert(int x) {
    for (int i = 63; i >= 0; --i) {
        if (x & (1ll << i)) {
            if (!b[i]) {
                b[i] = x;
                return true;
            }
            x ^= b[i];
        }
    }
    return false;
}
```

6.2 矩阵 (Matrix)

矩阵类：

```cpp
const int MAX_ML = 7;
struct Matrix{
    int m[MAX_ML][MAX_ML];
    int n;
    Matrix(int _n = 0) : n(_n) {
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
                m[i][j] = 0;
    }
    Matrix I() {
        Matrix res(n);
        for (int i = 1; i <= n; ++i) res[i][i] = 1;
        return res;
    }
    int* const operator [] (const int k) {
        return m[k];
    }
```

```
    Matrix operator * (const Matrix &M) {
        Matrix res(n);
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
                for (int k = 1; k <= n; ++k)
                    res[i][j] = (res[i][j] + m[i][k] * M.m[k][j]);
        return res;
    }
    Matrix& operator = (const Matrix &M) {
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j) m[i][j] = M.m[i][j];
        return *this;
    }
    Matrix f_pow(Matrix base, int b) {
        Matrix res(n); res = I();
        while (b) {
            if (b & 1) res = res * base;
            base = base * base;
            b >>= 1;
        }
        return res;
    }
};
```

6.3 高斯约当消元

$O(n^3)$.

```
bool Gauss() {
    for (int i = 1; i <= n; ++i) {
        int mpos = i;
        for (int j = i + 1; j <= n; ++j) {
            if (fabs(a[j][i]) > fabs(a[mpos][i])) mpos = j;
        }
        if (fabs(a[mpos][i]) < EPS) return false;
        for (int j = i; j <= n + 1; ++j) swap(a[i][j], a[mpos][j]);
        for (int j = 1; j <= n; ++j) {
            if (j == i) continue;
            double tmp = a[j][i] / a[i][i];
            for (int k = i + 1; k <= n + 1; ++k) {
                a[j][k] -= tmp * a[i][k];
            }
        }
    }
    return true;
}
```

Chapter 7

Graph

7.1 树

7.1.1 直径

树形 DP 做法，用 $dp[u]$ 表示当前结点向下最大距离，用最大及次大更新树的直径即可，$O(n)$.

```
void dfs(int u, int fa) {
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa) continue;
        dfs(v, u);
        dia = max(dia, dp[u] + dp[v] + 1);
        dp[u] = max(dp[u], dp[v] + 1);
    }
}
```

7.1.2 重心

定义
子树大小最大值最小的点.
性质
1. 重心子树的大小不会超过所有结点数目的一半.

Proof. 反证即可. □

2. 树中所有点到某一个点的距离和中，到重心的距离和最小.

Proof. 考虑树形 DP 的转移，记一个结点 u 的答案为 $dis[u]$，那么其儿子 v 可以由他转移

$$dis[v] = dis[u] + (n - 2 * siz[v])$$

也就是说，如果当前树的大小小于全部结点数 n 的一半，那么其父亲的答案肯定更小，以重心为根结点建立一颗树，则由上一条性质知此结论成立. □

3. 把两棵树连接，其重心在原重心路径上.

Proof. 若不再该路径上，由上一条结论证明中的方法，可以不断向上更新答案，矛盾. □

4. 在树上添加或删去一个叶子结点，重心最多移动一条边.
5. 重心之间有边相连.

Proof. 先确定一个重心，很显然，此重心最大的子树只有一个并且另一个重心在该子树中。如果其他子树大小都小于 $siz[son]-1$，那么显然最大子树根结点才是重心，故可以发现其他子树大小最大为 $siz[son]-1$，故另一个根结点即为最大子树的根. 故相连. □

6. 推论：树的重心最多有两个.

DFS 算法

选一个根 DFS，每次通过向上和向下更新当前答案即可.

7.1.3 树上差分

树上差分数组定义为

$$\mathrm{diff}[u] = w[u] - \sum_{v's\ father\ is\ u} w[v]$$

点差分：将结点 u 和 v 之间的所有点权值 $+x$，则操作为：

```
diff[u] += x;
diff[v] += x;
diff[lca] -= x;
diff[fa[lca]] -= x;
```

则每个结点的答案为其子树所有结点权值之和。

边差分：将结点 u 和 v 之间的所有边的权值 $+x$，则操作为：

```
diff[u] += x;
diff[v] += x;
diff[lca] -= 2 * x;
```

则当前子树的权值和是当前结点到其父亲结点的边的权值。

7.1.4 重链剖分

$O(n)$.

```
void dfs1(int u) {
   siz[u] = 1;
   for (int i = head[u]; i; i = e[i].next) {
      int v = e[i].to;
      if (v == fa[u]) continue;
      fa[v] = u;
      dep[v] = dep[u] + 1;
      dfs1(v);
      siz[u] += siz[v];
      if (siz[v] > siz[son[u]]) son[u] = v;
   }
}

void dfs2(int u, int tp) {
   top[u] = tp;
   dfn[u] = ++order;
```

```cpp
    rk[order] = u;
    if (son[u]) dfs2(son[u], tp);
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa[u] || v == son[u]) continue;
        dfs2(v, v);
    }
}
```

树链剖分求 LCA，$O(\log n)$。

```cpp
int lca(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]]) swap(u, v);
        v = fa[top[v]];
    }
    return dep[u] < dep[v] ? u : v;
}
```

7.1.5 树上启发式合并

某类计算每棵子树的答案的问题时，需要先由子树信息得到当前树的答案，但是需要清空去计算兄弟子树的答案导致时间复杂度变为 $O(n^2)$。注意到最后一棵子树不会再影响后面的兄弟子树，故其不需要清空，所以贪心地想必然选子树最大的作为最后一棵，也就是重儿子。算法设计过程如下：

1. 递归计算轻儿子的答案，并将记录的信息清空；
2. 计算重儿子的答案，不清空；
3. 合并其他轻儿子的答案。

复杂度分析：可以发现，每个结点被计算的次数即为从根走到当前结点的轻边数量加 1，基于重连剖分的性质，每个点被计算的次数为 $O(\log n)$，故总的时间复杂度为 $O(n \log n)$。

```cpp
//CF600E. Lomsat gelral
#include <iostream>
#include <cstdio>
#include <cstring>
#define int long long

using namespace std;

struct Edge {
    int to, next;
};

const int MAX_N = 1e5 + 7;
Edge e[MAX_N << 1];
int head[MAX_N], siz[MAX_N], cnt[MAX_N], son[MAX_N], ans[MAX_N], col[MAX_N];
int n, cnt_e, sum, mx;

void add(int u, int v) {
    e[++cnt_e].to = v;
```

```
    e[cnt_e].next = head[u];
    head[u] = cnt_e;
}

void dfs1(int u, int fa) {
    siz[u] = 1;
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa) continue;
        dfs1(v, u);
        siz[u] += siz[v];
        if (siz[v] > siz[son[u]]) son[u] = v;
    }
}

void del(int u, int fa) {
    --cnt[col[u]];
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa) continue;
        del(v, u);
    }
}

void calc(int u) {
    ++cnt[col[u]];
    if (mx < cnt[col[u]]) {
        sum = col[u];
        mx = cnt[col[u]];
    } else if (mx == cnt[col[u]]) {
        sum += col[u];
    }
}

void dfs2(int u, int fa) {
    calc(u);
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa) continue;
        dfs2(v, u);
    }
}

void dsu(int u, int fa) {
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa || v == son[u]) continue;
        dsu(v, u);
        del(v, u);
        sum = mx = 0;
    }
    if (son[u]) dsu(son[u], u);
    calc(u);
```

```
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa || v == son[u]) continue;
        dfs2(v, u);
    }
    ans[u] = sum;
}

signed main() {
    scanf("%lld", &n);
    for (int i = 1; i <= n; ++i) scanf("%lld", col + i);
    for (int i = 1; i < n; ++i) {
        int u, v;
        scanf("%lld%lld", &u, &v);
        add(u, v); add(v, u);
    }
    dfs1(1, 0);
    dsu(1, 0);
    for (int i = 1; i <= n; ++i) printf("%lld ", ans[i]);
    puts("");
    return 0;
}
```

7.1.6 虚树

$O(\sum k \log k)$.

```
bool cmp(const int &x, const int &y) {
    return dfn[x] < dfn[y];
}

void build() {
    sort(h + 1, h + 1 + k, cmp);
    st.push(1), vhead[1] = 0, cntve = 0;
    for (int i = 1, _lca; i <= k; i++) {
        if (h[i] == 1) continue;
        _lca = lca(h[i], st.top());
        if (_lca != st.top()) {
            int tp = st.top(); st.pop();
            while (dfn[_lca] < dfn[st.top()]) vadd(st.top(), tp), tp = st.top(), st.
                pop();
            if (dfn[_lca] > dfn[st.top()]) {
                vhead[_lca] = 0, vadd(_lca, tp), st.push(_lca);
            } else {
                vadd(_lca, tp);
            }
        }
        vhead[h[i]] = 0, st.push(h[i]);
    }
    int tp = st.top(); st.pop();
    while (!st.empty()) vadd(st.top(), tp), tp = st.top(), st.pop();
}
```

7.1.7 点分治

$O(n\log n)$.

```
//luogu P3806 【模板】点分治1
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#define INF 1e7

using namespace std;

struct Edge {
    int to, next, w;
    Edge() {}
};

const int MAX_N = 1e4 + 7;
const int MAX_K = 1e7 + 7;
const int MAX_M = 1e2 + 7;
Edge e[MAX_N << 1];
int head[MAX_N], siz[MAX_N], dp[MAX_N], dis[MAX_N], qu[MAX_N], k[MAX_M];
bool mp[MAX_K], vis[MAX_N], ans[MAX_M];
int n, m, rt, cnt, num;
queue<int> q;

void add(int u, int v, int w) {
    e[++cnt].to = v;
    e[cnt].next = head[u];
    head[u] = cnt;
    e[cnt].w = w;
}

void get_rt(int u, int fa, int size_all) {
    siz[u] = 1;
    dp[u] = 0;
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa || vis[v]) continue;
        get_rt(v, u, size_all);
        dp[u] = max(dp[u], siz[v]);
        siz[u] += siz[v];
    }
    dp[u] = max(dp[u], size_all - siz[u]);
    if (dp[u] < dp[rt]) rt = u;
}

void get_dis(int u, int fa) {
    qu[++num] = dis[u];
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa || vis[v]) continue;
        dis[v] = dis[u] + e[i].w;
```

```
        get_dis(v, u);
    }
}

void solve(int u) {
    vis[u] = mp[0] = true;
    q.push(0);
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (vis[v]) continue;
        num = 0; dis[v] = e[i].w;
        get_dis(v, u);
        for (int t = 1; t <= num; ++t) {
            for (int j = 1; j <= m; ++j) {
                if (qu[t] <= k[j]) ans[j] |= mp[k[j] - qu[t]];
            }
        }
        for (int t = 1; t <= num; ++t) if (qu[t] <= INF) q.push(qu[t]), mp[qu[t]] =
             true;
    }
    while (!q.empty()) mp[q.front()] = false, q.pop();
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (vis[v]) continue;
        rt = 0;
        get_rt(v, u, siz[v]);
        solve(rt);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i < n; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w); add(v, u, w);
    }
    for (int i = 1; i <= m; ++i) scanf("%d", k + i);
    rt = 0; dp[0] = INF;
    get_rt(1, 0, n);
    solve(rt);
    for (int i = 1; i <= m; ++i) {
        if (ans[i]) {
            puts("AYE");
        } else {
            puts("NAY");
        }
    }
    return 0;
}
```

luogu P4178 Tree：求树上距离小于等于 k 的点对数，如果仿照上一种方法，可以直接利用权值线段树进行区间查询单点修改，时间复杂度 $O(n \log n \log k)$；也可以基于桶排序的双指针法（需

要容斥)，时间复杂度 $O(max\{n,k\}\log n)$。

```
//luogu P4178 Tree
// 计算距离小于等于 k 的点对，利用双指针法
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#include <complex>
#include <vector>
#define INF 2e4

using namespace std;

struct Edge {
    int to, next, w;
    Edge() {}
};

const int MAX_N = 4e4 + 7;
const int MAX_K = 2e4 + 7;
Edge e[MAX_N << 1];
int head[MAX_N], siz[MAX_N], dp[MAX_N], dis[MAX_N], qu[MAX_N], box[MAX_K];
bool vis[MAX_N];
int n, m, rt, cnt, num, k, ans;
queue<int> q;

void add(int u, int v, int w) {
    e[++cnt].to = v;
    e[cnt].next = head[u];
    head[u] = cnt;
    e[cnt].w = w;
}

void get_rt(int u, int fa, int size_all) {
    siz[u] = 1;
    dp[u] = 0;
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa || vis[v]) continue;
        get_rt(v, u, size_all);
        dp[u] = max(dp[u], siz[v]);
        siz[u] += siz[v];
    }
    dp[u] = max(dp[u], size_all - siz[u]);
    if (dp[u] < dp[rt]) rt = u;
}

void get_dis(int u, int fa) {
    ++num;
    ++box[dis[u]];
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
```

```
            if (v == fa || vis[v]) continue;
            dis[v] = dis[u] + e[i].w;
            get_dis(v, u);
        }
}

int calc(int u, int w) {
    dis[u] = w; num = 0; get_dis(u, 0);
    for (int i = 0, j = 1; i <= MAX_K && j <= num; ++i) {
        while (box[i]) qu[j++] = i, --box[i];
    }
    int l = 1, r = num, res = 0;
    while (l < r) {
        qu[l] + qu[r] <= k ? res += r - l, ++l : --r;
    }
    return res;
}

void solve(int u) {
    vis[u] = true; ans += calc(u, 0);
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (vis[v]) continue;
        dis[v] = e[i].w;
        ans -= calc(v, dis[v]);
    }
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (vis[v]) continue;
        rt = 0;
        get_rt(v, u, siz[v]);
        solve(rt);
    }
}

signed main() {
    scanf("%d", &n);
    for (int i = 1; i < n; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w); add(v, u, w);
    }
    scanf("%d", &k);
    rt = 0; dp[0] = INF + INF;
    get_rt(1, 0, n);
    solve(rt);
    printf("%d\n", ans);
    return 0;
}
```

7.2 差分约束

$O(|V||E|)$.

```
bool bellman_ford(int s) {
    memset(dis, 0x3f, sizeof dis);
    dis[s] = 0;
    int cnt = 0;
    while(cnt <= n) {
        bool upd = false;
        for(int i = 1; i <= cntE; i++) {
            int x = e[i].from, y = e[i].to, w = e[i].w;
            if(dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                upd = true;
            }
        }
        if(!upd) return false;
        cnt++;
    }
    return true;
}
int main() {
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= m; i++) {
        int x, y, w;
        scanf("%d%d%d", &x, &y, &w);
        add(y, x, w);
    }
    for(int i = 1; i <= n; i++) {
        add(0, i, 0);
    }
    if(bellman_ford(0)) puts("NO");
    else {
        for(int i = 1; i <= n; i++) printf("%d ", dis[i]);
        puts("");
    }
    return 0;
}
```

7.3 强连通分量 SCC

7.3.1 tarjan 算法

$O(n)$.

```
void tarjan(int u) {
    dfn[u] = low[u] = ++order;
    st.push(u);
    in_st[u] = true;
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (!dfn[v]) {
```

```
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in_st[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        int tmp;
        num++;
        do {
            tmp = st.top();
            st.pop();
            scc[tmp] = num;
            in_st[tmp] = false;
        } while(tmp != u);
    }
}
```

7.4 双连通分量 BCC

7.4.1 点双

$O(n)$.

```
void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++order;
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa) continue;
        if (!dfn[v]) {
            int id = e[i].index;
            st.push(id);
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u]) {
                num++;
                mn[num] = INF;
                int tmp;
                do {
                    tmp = st.top();
                    belong[tmp] = num;
                    st.pop();
                    mn[num] = min(mn[num], tmp);
                } while(tmp != id);
            }
        } else if (dfn[v] < dfn[u]) {
            st.push(e[i].index);
            low[u] = min(low[u], dfn[v]);
        }
    }
}
```

7.4.2 割点

$O(n)$.

```
void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++order;
    int child = 0;
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa) continue;
        if (!dfn[v]) {
            child++;
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u] && fa) {
                if (!point[u]) point[u] = true, ans++;
            } else if (!fa && child > 1) {
                if (!point[u]) point[u] = true, ans++;
            }
        } else if (dfn[v] < dfn[u]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
}
```

7.4.3 边双连通分量/桥

$O(n)$.

```
void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++order;
    bool flag = true;
    st.push(u);
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (v == fa && flag) {
            flag = false;
            continue;
        }
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) {
                bridge[i] = 1;
                if (i & 1) bridge[i + 1] = 1;
                else bridge[i - 1] = 1;
            }
        } else if (dfn[v] < dfn[u]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        num++;
        int tmp;
```

```
        do{
            tmp = st.top();
            belong[tmp] = num;
            st.pop();
        } while(tmp != u);
    }
}
```

7.5 2-SAT

$O(n)$.

```
//HDU 3062 Party
#include <iostream>
#include <cstdio>
#include <cstring>
#include <stack>

using namespace std;

struct Edge {
    int to, next;
};

const int MAX_N = 2e3 + 7;
Edge e[MAX_N * MAX_N];
int head[MAX_N], dfn[MAX_N], low[MAX_N], scc[MAX_N];
bool vis[MAX_N];
int n, m, cnt, num, order;
stack <int> st;

void add(int x, int y) {
    e[++cnt].to = y; e[cnt].next = head[x]; head[x] = cnt;
}

void tarjan(int u) {
    dfn[u] = low[u] = ++order;
    st.push(u); vis[u] = true;
    for (int i = head[u]; i; i = e[i].next) {
        int v = e[i].to;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (vis[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        num++;
        int tmp;
        do {
            tmp = st.top();
```

```
            st.pop();
            scc[tmp] = num;
            vis[tmp] = false;
        } while(tmp != u);
    }
}

bool twoSat() {
    for (int i = 0; i < (n + n); i++) {
        if (!dfn[i]) tarjan(i);
    }
    for (int i = 0; i < n; i++) {
        if (scc[i] == scc[i + n]) return false;
    }
    return true;
}

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        cnt = order = num = 0;
        for (int i = 0; i < (n + n); i++) {
            dfn[i] = low[i] = head[i] = 0;
        }
        for (int i = 1; i <= m; i++) {
            int a1, a2, c1, c2;
            scanf("%d%d%d%d", &a1, &a2, &c1, &c2);
            add(a1 + c1 * n, a2 + (c2 ^ 1) * n);
            add(a2 + c2 * n, a1 + (c1 ^ 1) * n);
        }
        if (twoSat()) puts("YES");
        else puts("NO");
    }
    return 0;
}
```

7.6 斯坦纳树

$O(n \times 3^k + m \log m \times 2^k)$

```
//luogu P6192 【模板】最小斯坦纳树
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#define INF 0x3f3f3f3f

using namespace std;

struct Edge {
    int to, next, w;
    Edge() {}
};
```

```cpp
struct Node {
    int v, w;
    bool operator < (const Node & x) const {
        return x.w < w;
    }
    Node (int _v = 0, int _w = 0) : w(_w), v(_v) {}
}p;

const int MAX_N = 1e2 + 7;
const int MAX_S = 1 << 10;
int dp[MAX_N][MAX_S + 7], head[MAX_N];
bool vis[MAX_N];
Edge e[10 * MAX_N];
priority_queue <Node> q;
int n, m, k, cnt, key;

void add(int x, int y, int w) {
    e[++cnt].to = y; e[cnt].next = head[x]; head[x] = cnt; e[cnt].w = w;
}

void dijkstra(int s) {
    for (int i = 1; i <= n; i++) vis[i] = false;
    while (!q.empty()) {
        p = q.top(); q.pop();
        if (vis[p.v]) continue;
        vis[p.v] = true;
        for (int i = head[p.v]; i; i = e[i].next) {
            int v = e[i].to;
            if (dp[v][s] > dp[p.v][s] + e[i].w) {
                dp[v][s] = dp[p.v][s] + e[i].w;
                q.push(Node(v, dp[v][s]));
            }
        }
    }
}

int main() {
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 1; i <= m; i++) {
        int x, y, w;
        scanf("%d%d%d", &x, &y, &w);
        add(x, y, w); add(y, x, w);
    }
    memset(dp, 0x3f, sizeof dp);
    for (int i = 0; i < k; i++) {
        scanf("%d", &key);
        dp[key][1 << i] = 0;
    }
    for (int s = 1; s < (1 << k); s++) {
        for (int i = 1; i <= n; i++) {
            for (int subs = s & (s - 1); subs; subs = s & (subs - 1)) {
                dp[i][s] = min(dp[i][s], dp[i][subs] + dp[i][subs ^ s]);
            }
```

```
        if (dp[i][s] < INF) q.push(Node(i, dp[i][s]));
      }
      dijkstra(s);
   }
   printf("%d\n", dp[key][(1 << k) - 1]);
   return 0;
}
```

7.7 网络流

7.7.1 最大流/最小割

Ford-Fulkerson 算法

步骤一（贪心）
1. 找到一条由 s 到 t 的只经过 $f(e) < c(e)$ 的路径；
2. 如果不存在该路径，算法结束. 否则，沿该路径尽可能增加 $f(e)$，返回上一步.
步骤二 1. 利用残余网络寻找 s 到 t 的路径；2. 若不存在该路径，算法结束. 否则，沿该路径尽可能增加流，返回上一步.
残余网络定义了新的边，为原来边的反向边，其容量为：

$$c_f(e) = \begin{cases} f(e), e \notin E \\ c(e) - f(e), e \in E \end{cases}$$

即允许流量回流.

Dinic 算法

$O(|E||V|^2)$.

```
struct Edge {
   int to, cap, rev;
   Edge(int _to = 0, int _cap = 0, int _rev = 0) : to(_to), cap(_cap), rev(_rev) {}
};

const int MAX_N = 1e2 + 7;
vector <Edge> e[MAX_N];
int level[MAX_N], iter[MAX_N];
int n, m;

void add(int from, int to, int cap) {
   e[from].push_back((Edge){to, cap, (int)e[to].size()});
   e[to].push_back((Edge){from, 0, (int)e[from].size() - 1});
}

bool bfs(int s, int t) {
   memset(level, 0, sizeof level);
   queue <int> q;
   level[s] = 1;
   q.push(s);
   while (!q.empty()) {
      int v = q.front(); q.pop();
      for (int i = 0; i < e[v].size(); i++) {
```

```
        Edge &ed = e[v][i];
        if (ed.cap > 0 && !level[ed.to]) {
          level[ed.to] = level[v] + 1;
          q.push(ed.to);
        }
      }
    }
    return level[t] > 0;
}

int dfs(int v, int t, int f) {
    if (v == t) return f;
    for (int &i = iter[v]; i < e[v].size(); i++) {
      Edge &ed = e[v][i];
      if (ed.cap > 0 && level[v] < level[ed.to]) {
        int d = dfs(ed.to, t, min(f, ed.cap));
        if (d) {
          ed.cap -= d;
          e[ed.to][ed.rev].cap += d;
          return d;
        }
      }
    }
    return 0;
}

int dinic(int s, int t) {
    int flow = 0, f;
    while (bfs(s, t)) {
    memset(iter, 0, sizeof iter);
      while (f = dfs(s, t, INF)) {
        flow += f;
      }
    }
    return flow;
}
```

7.7.2 最小费用最大流 (MCMF)

```
struct Edge {
    int to, cap, cost, rev;
    Edge() {}
    Edge(int _to, int _cap, int _cost, int _rev)
      : to(_to), cap(_cap), cost(_cost), rev(_rev) {}
};

int dis[MAX_N];
int pre[MAX_N];
int tag[MAX_N];
bool vis[MAX_N];
vector<Edge> e[MAX_N];
queue<int> q;
```

```cpp

void add(int x, int y, int z, int w) {
    e[x].push_back(Edge(y, z, w, e[y].size()));
    e[y].push_back(Edge(x, 0, -w, e[x].size() - 1));
}

bool spfa(int s, int t) {
    memset(dis, 0x3f, sizeof(int) * (n + 5));
    dis[s] = 0, q.push(s), vis[s] = true;
    while (!q.empty()) {
        int x = q.front();
        q.pop(), vis[x] = false;
        for (int i = 0; i < e[x].size(); i++) {
            Edge &ed = e[x][i];
            if (ed.cap > 0 && dis[ed.to] > dis[x] + ed.cost) {
                dis[ed.to] = dis[x] + ed.cost;
                pre[ed.to] = x, tag[ed.to] = i;
                if (!vis[ed.to]) {
                    q.push(ed.to), vis[ed.to] = true;
                }
            }
        }
    }
    return dis[t] < INF;
}

pii mcmf(int s, int t) {
    int flow = 0, cost = 0;
    while (spfa(s, t)) {
        int f = INF;
        for (int i = t; i != s; i = pre[i]) {
            f = min(f, e[pre[i]][tag[i]].cap);
        }
        flow += f, cost += f * dis[t];
        for (int i = t; i != s; i = pre[i]) {
            Edge &ed = e[pre[i]][tag[i]];
            ed.cap -= f, e[i][ed.rev].cap += f;
        }
    }
    return pii(flow, cost);
}
```

Chapter 8

Data Structure

8.1 并查集 (Union-Find)

```
struct UF {
    vector<int> fa, rk;
    UF(int n = 0) {
        fa.resize(n + 1);
        rk.resize(n + 1);
        for (int i = 1; i <= n; ++i) fa[i] = i, rk[i] = 1;
    }
    int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
    int merge(int x, int y) {
        x = find(x), y = find(y);
        if (rk[x] < rk[y]) swap(x, y);
        fa[y] = x, rk[x] = max(rk[x], rk[y] + 1);
        return x;
    }
};
```

8.2 哈希表

```
const int HASHSIZE = 100033, KEYSIZE = 102000;
template<typename Key, typename Mapped>
struct HashMap {
    typedef pair<Key, Mapped> pii;
    pii *End, kv[KEYSIZE];
    int head[HASHSIZE], nxt[HASHSIZE], tot;

    HashMap() {
        clear(); End = new pii;
    }
    pii *end() {
        return End;
    }
    inline void clear() {
        memset(head, 0, sizeof head); tot = 0;
    }
```

```
    inline void insert(const Key &x, const Mapped &y) {
        kv[++tot] = pii(x, y);
        int xm = x % HASHSIZE;
        nxt[tot] = head[xm];
        head[xm] = tot;
    }
    inline pii *find(const Key &x) {
        int xm = x % HASHSIZE;
        for (int _p = head[xm]; _p; _p = nxt[_p]) {
            if (kv[_p].first == x) return kv + _p;
        }
        return End;
    }
    Mapped &operator[] (const Key &idx) {
        if (find(idx) == End) insert(idx, 0);
        return find(idx)->second;
    }
    int count(const Key &idx) {
        if (find(idx) == End) return 0;
        return 1;
    }
};
```

8.3 Splay Tree

$O(\log n)$。

```
struct Splay {
    vector<int> key, siz, cnt, fa, tag;
    vector<vector<int>> ch;
    int rt, tot;
    Splay(int len = 0) {
        rt = tot = 0;
        key.resize(len, 0);
        siz.resize(len, 0);
        cnt.resize(len, 0);
        fa.resize(len, 0);
        tag.resize(len, 0); // lazy tag when using interval operations
        ch.resize(len, vector<int> (2, 0));
    }

    void pushup(int x) {
        if (x) {
            siz[x] = cnt[x] + siz[ch[x][0]] + siz[ch[x][1]];
        }
    }
    inline int get(const int &x) { return ch[fa[x]][1] == x; }
    inline void clear(const int &x) {
        ch[x][0] = ch[x][1] = siz[x] = cnt[x] = fa[x] = key[x] = 0;
    }
    void pushdown(int x) { // pushdown the lazy tag
        if (x && tag[x]) {
```

```
        swap(ch[x][0], ch[x][1]);
        tag[x] = 0;
        tag[ch[x][0]] ^= 1;
        tag[ch[x][1]] ^= 1;
    }
}
// blanced BST
void rotate(int x) {
    int f = fa[x], ff = fa[f], p = get(x);
    ch[f][p] = ch[x][p ^ 1], fa[ch[f][p]] = f;
    fa[f] = x, ch[x][p ^ 1] = f, fa[x] = ff;
    if (ff) ch[ff][ch[ff][1] == f] = x;
    pushup(f), pushup(x);
}
void splay(int x, int g) {
    for (int f; (f = fa[x]) != g; rotate(x)) {
        if (fa[f] != g) rotate(get(x) == get(f) ? f : x);
    }
    if (!g) rt = x;
}
int find(int x) {
    if (!rt) return -1;
    int u = rt, ans = 1, flag;
    while (ch[u][flag = (x > key[u])] && key[u] != x) ans += (siz[ch[u][0]] + cnt
        [u]) * flag, u = ch[u][flag];
    ans += siz[ch[u][0]];
    splay(u, 0);
    return key[u] == x ? ans : -1;
}
void insert(int x) {
    int u = rt, f = 0;
    while (u && key[u] != x) f = u, u = ch[u][x > key[u]];
    if (u) {
        ++cnt[u];
    } else {
        u = ++tot;
        if (f) ch[f][x > key[f]] = u;
        ch[u][0] = ch[u][1] = 0;
        fa[u] = f, key[u] = x, cnt[u] = siz[u] = 1;
    }
    splay(u, 0);
}
int pre(int x) {
    find(x);
    int u = rt;
    if (key[u] < x) return u;
    for (u = ch[u][0]; ch[u][1];) u = ch[u][1];
    return u;
}
int nxt(int x) {
    find(x);
    int u = rt;
    if (key[u] > x) return u;
```

```
        for (u = ch[u][1]; ch[u][0];) u = ch[u][0];
        return u;
    }
    bool del(int x) {
        find(x);
        if (!cnt[rt]) return false;
        int of = rt;
        if (cnt[rt] > 1) {
            --cnt[rt];
        } else if (!ch[rt][0] && !ch[rt][1]) {
            clear(rt), rt = 0;
        } else if (!ch[rt][0]) {
            rt = ch[rt][1], fa[rt] = 0, clear(of);
        } else if (!ch[rt][1]) {
            rt = ch[rt][0], fa[rt] = 0, clear(of);
        } else {
            int l = pre(x);
            splay(l, 0);
            ch[rt][1] = ch[of][1], fa[ch[of][1]] = rt, clear(of);
            pushup(rt);
        }
        return true;
    }
    int kth(int x) {
        if (tot < x) return -1;
        int u = rt, tmp, left;
        while (true) {
            pushdown(u);
            left = ch[u][0];
            if (x <= siz[left]) {
                u = left;
            } else {
                tmp = siz[left] + cnt[u];
                if (tmp >= x) return u;
                x -= tmp;
                u = ch[u][1];
            }
        }
    }
    inline int pre_key(const int &x) { return key[pre(x)]; }
    inline int nxt_key(const int &x) { return key[nxt(x)]; }
    inline int kth_key(const int &x) { return key[kth(x)]; }
    // interval operations
    int build(int l, int r, int p) { // a little slow
        if (l > r) return 0;
        int mid = (l + r) >> 1, u = ++tot;
        key[u] = mid, fa[u] = p, ++cnt[u];
        ch[u][0] = build(l, mid - 1, u);
        ch[u][1] = build(mid + 1, r, u);
        pushup(u);
        return u;
    }
    void reverse(int l, int r) {
```

```
        if (l >= r) return;
        int L = kth(l), R = kth(r);
        splay(L, 0);
        splay(R, L);
        tag[ch[ch[rt][1]][0]] ^= 1;
    }
};
```

8.4 Treap (Tree-heap)

$O(\log n)$。

```
mt19937 rnd(time(0));
struct Treap {
    vector<int> siz, key, pri;
    vector<vector<int>> ch;
    int rt, tot;
    Treap(int len = 0) {
        siz.resize(len, 0);
        key.resize(len, 0);
        pri.resize(len, 0);
        ch.resize(len, vector<int> (2, 0));
        rt = tot = 0;
    }

    void pushup(int x) {
        if (x) {
            siz[x] = siz[ch[x][0]] + siz[ch[x][1]] + 1;
        }
    }
    int new_node(int x) {
        int u = ++tot;
        siz[u] = 1, key[u] = x, pri[u] = rnd() % 998244353;
        ch[u][0] = ch[u][1] = 0;
        return u;
    }
    void clear(int x) {
        siz[x] = key[x] = pri[x] = ch[x][0] = ch[x][1] = 0;
    }

    int merge(int u, int v) {
        if (!u || !v) return u + v;
        if (pri[u] > pri[v]) {
            ch[u][1] = merge(ch[u][1], v);
            pushup(u);
            return u;
        } else {
            ch[v][0] = merge(u, ch[v][0]);
            pushup(v);
            return v;
        }
    }
```

```
    int merge(pii p) { return merge(p.first, p.second); }
    pii split(int u, int x) { // ch[u][0] <= k, ch[u][1] > k
        if (!u) return pii(0, 0);
        if (x < key[u]) {
            pii o = split(ch[u][0], x);
            ch[u][0] = o.second;
            pushup(u);
            return pii(o.first, u);
        } else {
            pii o = split(ch[u][1], x);
            ch[u][1] = o.first;
            pushup(u);
            return pii(u, o.second);
        }
    }
    void insert(int x) {
        int u = new_node(x);
        pii o = split(rt, x);
        o.first = merge(o.first, u);
        rt = merge(o);
    }
    bool del(int x) {
        pii o = split(rt, x - 1), p = split(o.second, x);
        if (!p.first) return false;
        int u = merge(ch[p.first][0], ch[p.first][1]);
        clear(p.first);
        rt = merge(o.first, merge(u, p.second));
        return true;
    }
    int find(int x) {
        pii o = split(rt, x - 1);
        int res = siz[o.first] + 1;
        rt = merge(o);
        return res;
    }
    int kth(int rt, int x) {
        int u = rt;
        if (tot < x) return -1;
        while (siz[ch[u][0]] + 1 != x) {
            if (siz[ch[u][0]] >= x) {
                u = ch[u][0];
            } else {
                x -= siz[ch[u][0]] + 1;
                u = ch[u][1];
            }
        }
        return key[u];
    }
    int kth(int x) { return kth(rt, x); }
    int pre(int x) {
        pii o = split(rt, x - 1);
        int res = kth(o.first, siz[o.first]);
        rt = merge(o);
```

```
        return res;
    }
    int nxt(int x) {
        pii o = split(rt, x);
        int res = kth(o.second, 1);
        rt = merge(o);
        return res;
    }
};
```

Chapter 9

String Theory

9.1 AC 自动机

$$O(\sum_{i=1}^{n}|s_i| + n|\Sigma| + |T|)$$

```cpp
//luogu P3796 【模板】AC自动机（加强版）
#include <iostream>
#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>

using namespace std;

const int MAX_N = 157;
const int L = 77;
const int MAX_T = 1e6 + 7;

void clear(queue<int> &q) {
    queue<int> empty;
    swap(empty, q);
}

struct AC {
    int tot;
    vector<vector<int>> tr;
    vector<int> fail, id, val, cnt;
    queue<int> q;

    AC(int len, int N, int chSize) : tot(0) {
        fail.resize(len, 0);
        id.resize(len, 0);
        val.resize(len, 0);
        cnt.resize(N, 0);
        tr.resize(len, vector<int> (chSize, 0));
        clear(q);
    }

```

```
    void insert(char *s, int index) {
        int u = 0, n = strlen(s + 1);
        for (int i = 1; i <= n; ++i) {
            int e = s[i] - 'a';
            if (!tr[u][e]) tr[u][e] = ++tot;
            u = tr[u][e];
        }
        id[u] = index;
    }

    void build() {
        int u;
        for (int i = 0; i < 26; ++i) {
            if (tr[0][i]) q.push(tr[0][i]);
        }
        while (!q.empty()) {
            u = q.front(), q.pop();
            for (int i = 0; i < 26; ++i) {
                if (tr[u][i]) {
                    fail[tr[u][i]] = tr[fail[u]][i];
                    q.push(tr[u][i]);
                } else {
                    tr[u][i] = tr[fail[u]][i];
                }
            }
        }
    }

    int query(char *t) {
        int u = 0, res = 0, n = strlen(t + 1);
        for (int i = 1; i <= n; ++i) {
            u = tr[u][t[i] - 'a'];
            for (int j = u; j; j = fail[j]) ++val[j];
        }
        for (int i = 1; i <= tot; ++i) {
            if (id[i]) res = max(res, val[i]), cnt[id[i]] = val[i];
        }
        return res;
    }
};

char s[MAX_N][L], t[MAX_T];
int n, mx;

int main() {
    while (true) {
        scanf("%d", &n);
        if (!n) break;
        AC ac((n + 1) * L, n + 1, 27);
        for (int i = 1; i <= n; ++i) scanf("%s", s[i] + 1), ac.insert(s[i], i);
        ac.build();
        scanf("%s", t + 1);
        mx = ac.query(t);
```

```
        printf("%d\n", mx);
        for (int i = 1; i <= n; ++i) if (ac.cnt[i] == mx) puts(s[i] + 1);
    }
    return 0;
}
```

9.2 前缀函数（Prefix-Function）

$O(n)$ 计算 π 函数。

```
void calc_pi(string s) {
    int n = s.size();
    vector<int> pi(n);
    for (int i = 1; i < n; ++i) {
        int j = pi[i - 1];
        while (j && s[j] != s[i]) j = pi[j - 1];
        pi[i] = j + (s[i] == s[j]);
    }
}
```

9.3 KMP 自动机

$O(n|\Sigma|)$。

```
struct KMP {
    int n, st;
    vector<int> pi;
    vector<vector<int>> tr;

    KMP(int _n) : n(_n) {
        pi.resize(_n, 0);
        tr.resize(_n, vector<int> (26, 0));
    }

    void reset(int _st = 0) {
        st = _st;
    }

    void calc_pi(char *s) {
        for (int i = 1; i < n; ++i) {
            int j = pi[i - 1];
            while (j && s[i] != s[j]) j = pi[j - 1];
            pi[i] = j + (s[i] == s[j]);
        }
    }

    void build(char *s) {
        calc_pi(s);
        for (int i = 0; i < n; ++i) {
            for (int c = 0; c < 26; ++c) {
                if (i && ('A' + c != s[i])) {
```

```
                tr[i][c] = tr[pi[i - 1]][c];
            } else {
                tr[i][c] = i + ('A' + c == s[i]);
            }
        }
    }
}

int query(int c) {
    st = tr[st][c];
    return st;
}
};
```

9.4 后缀数组（SA）

9.4.1 后缀排序

倍增

基于倍增的排序，$O(n \log n)$。

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

void suffix_sort(char *s) {
    int m = 300, p;
    for (int i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
    for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
    for (int i = n; i; --i) sa[cnt[rk[i]]--] = i;
    for (int w = 1; w < n; w <<= 1, m = p) {
        p = 0;
        for (int i = n; i > n - w; --i) id[++p] = i;
        for (int i = 1; i <= n; ++i) {
            if (sa[i] > w) id[++p] = sa[i] - w;
        }
        for (int i = 0; i <= m; ++i) cnt[i] = 0;
        for (int i = 1; i <= n; ++i) ++cnt[px[i] = rk[id[i]]];
        for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
        for (int i = n; i; --i) sa[cnt[px[i]]--] = id[i];
        swap(oldrk, rk);
        p = 0;
        for (int i = 1; i <= n; ++i) {
            rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
        }
    }
}
```

DC3

9.4.2 最长公共前缀（LCP）

定义 9.4.1 (最长公共前缀). 对于两个字符串 S 和 T，其最长公共前缀（LCP）即为 x，x 是满足 $\forall 1 \le i \le x$，$S_i = T_i$ 的最大整数。
并记 $lcp(i, j)$ 表示后缀 i 和后缀 j 的 LCP，$LCP(i, j) = lcp(SA[i], SA[j])$。

性质 9.4.1. $LCP(i, j) = LCP(j, i)$。

性质 9.4.2. $LCP(i, i) = n - |SA[i]| + 1$。

引理 9.4.1 (LCP 引理). 对任意 $1 \le i < j < k \le n$，$LCP(i, k) = min\{LCP(i, j), LCP(j, k)\}$。

证明. 记 $LC(i, k) = p$，$SA[i]$ 表示的字符串为 AB，$SA[k]$ 表示 AD，$|A| = p$。然后根据 $SA[i, i+1, \ldots, k]$ 表示字符串有序再随便反证一下就行了。 □

由 LCP 引理立即得到 LCP 定理:

定理 9.4.1 (LCP 定理). 对任意 $1 \le i < j \le n$，有 $LCP(i, j) = min\{LCP(k-1, k) | i < k \le j\}$。

推论 9.4.1. 对 $i \le j < k$，有 $LCP(i, k) \le LCP(j, k)$。

由此进一步定义 $height$ 和 h 数组。

定义 9.4.2 (height 数组).

$$height[i] = \begin{cases} LCP(i-1, i), 1 < i \le n \\ 0, i = 1 \end{cases}$$

定义 9.4.3 (h 数组). $h[i] = height[Rank[i]]$，也即 $height[i] = h[SA[i]]$。

定理 9.4.2. 对任意 $i > 1$ 且 $Rank[i] > 1$，均有 $h[i] \ge h[i-1] - 1$。

证明. 仿照引理的证明方法。 □

根据上面定理，即可 $O(n)$ 暴力求解 $height$ 数组。

```
void calc_height(char *s) {
    for (int i = 1, k = 0; i <= n; ++i) {
        if (k) --k;
        int j = sa[rk[i] - 1];
        while (s[i + k] == s[j + k]) ++k;
        height[rk[i]] = k;
    }
}
```

9.5 后缀自动机（SAM）

$O(n)$。

```
struct SAM {
    int cnt, last;
    vector<int> len, link;
    vector<vector<int>> tr;
    // vector<map<int, int>> tr; // Use map but there'll be a log.

```

```cpp
    SAM(int strLen, int chSize) : cnt(1), last(1) {
        len.resize(strLen * 2, 0);
        link.resize(strLen * 2, 0);
        tr.resize(strLen * 2, vector<int> (chSize, 0));
        // tr.resize(strLen * 2, map<int, int> ());
    }

    void extend(int c) {
        int p, cur = ++cnt;
        len[cur] = len[last] + 1;
        for (p = last; p && (!tr[p][c]); p = link[p]) tr[p][c] = cur;
        if (!p) {
            link[cur] = 1;
        } else {
            int q = tr[p][c];
            if (len[q] == len[p] + 1) {
                link[cur] = q;
            } else {
                int clone = ++cnt;
                len[clone] = len[p] + 1, tr[clone] = tr[q], link[clone] = link[q];
                for (; p && tr[p][c] == q; p = link[p]) tr[p][c] = clone;
                link[cur] = link[q] = clone;
            }
        }
        last = cur;
    }
};
```

9.6 Manacher

$O(n)$。

```cpp
vector<int> manacher(string s) {
    n = s.size();
    vector<int> d(n);
    for (int i = 0, l = 0, r = -1; i < n; ++i) {
        int k = (i > r) ? 1 : min(d[l + r - i], r - i + 1);
        while ((i - k >= 0) && (i + k < n) && (s[i - k] == s[i + k])) ++k;
        d[i] = k--;
        if (i + k > r) l = i - k, r = i + k;
    }
    return d;
}
```