```cpp
#include "../bits/stdc++.h"
/**
 * FFT における ω = exp(-2*pi*i/N) を ω^N = 1 (mod P)
 * となるような 1 の原始 n 乗根に拡張
 *
 * sum_{k=0}^{n-1}{ω^((l-j)*k)} = (l==j) ? n : 0
 *
 * e.g. P = 998244353 = 7 * 17 * 2^23 + 1 -> ω = 3, 31...
 *      P = 163577857 = 39 * 2^22 + 1 -> 23...
 *      P = 167772161 = 5 * 2^25 + 1 -> 3,...,
 *      P = 469762049 = 7 * 2^26 + 1 -> 3,...,
 *
 * E に対する n-1 次多項式fを定義する. (nは2の冪乗)
 * f(E) = a_{n-1}*E^{n-1} + ... + a_1*E + a_0
 *
 * NTT における順変換
 * f_k = a_{n-1}*ω^{(n-1)*k} + ... + a_1*ω^k + a_0
 *     = sum_{l=0}^{n-1}{a_l*ω^(l*k)} (k = 0,1,...,n-1)
 * となる f_k を求める
 *
 * 逆変換
 * a_j = (1/n) * sum_{k=0}f_k*ω^(-kj) (j = 0,1,...,n-1)
 */
// verified: http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=3384943
class NTT
{
    using ll = long long;
    // mod = 998244353
    const int mod = 7 * 17 * (1 << 23) + 1;

    static const int omega_max = 23;
    static const int root = 31;
    std::vector<int> omega;

    int mod_inv(ll a, ll m)
    {
        ll b = mod, u = 1, v = 0;
        while (b > 0)
        {
            ll t = a / b;
            a -= t * b;
            std::swap(a, b);
            u -= t * v;
            std::swap(u, v);
        }
        return (u % m + m) % m;
    }
    std::vector<int> fft(std::vector<int> v, bool inv)
    {
        const int n = v.size();
        // assert(n == 2冪);
        int _logn = 0;
        while ((1 << _logn) < n)
            _logn++;
        assert(1 << _logn == n);
        int ww = omega[_logn];
        if (inv)
            ww = mod_inv(ww, mod);
        for (int m = n; m >= 2; m >>= 1)
        {
            const int mh = m >> 1;
            int w = 1;
            for (int i = 0; i < mh; i++)
            {
                for (int j = i; j < n; j += m)
                {
                    const int k = j + mh;
                    int x = v[j] - v[k];
                    if (x < 0)
                        x += mod;
                    v[j] += v[k];
                    if (v[j] >= mod)
                        v[j] -= mod;
                    v[k] = (1LL * w * x) % mod;
                }
                w = (1LL * w * ww) % mod;
            }
            ww = (1LL * ww * ww) % mod;
        }

        int i = 0;
        for (int j = 1; j < n - 1; j++)
        {
            for (int k = n >> 1; k > (i ^= k); k >>= 1)
                ;
            if (j < i)
                std::swap(v[i], v[j]);
        }
        if (inv)
        {
            const int inv_n = mod_inv(n, mod);
            for (auto &x : v)
            {
                x = (1LL * x * inv_n) % mod;
            }
```

```
        }
        return v;
    }

  public:
    NTT()
    {
        omega.resize(omega_max + 1);
        int x = root;
        for (int i = omega_max; i >= 0; i--)
        {
            omega[i] = x;
            x = (1LL * x * x) % mod;
        }
    }
    std::vector<int> convolution(std::vector<int> f, std::vector<int> g)
    {
        int sz = 1;
        const int m = f.size() + g.size() - 1;
        while (sz < m)
            sz *= 2;
        f.resize(sz), g.resize(sz);
        f = this->fft(std::move(f), false);
        g = this->fft(std::move(g), false);
        for (int i = 0; i < sz; i++)
        {
            f[i] = (1LL * f[i] * g[i]) % mod;
        }
        return this->fft(std::move(f), true);
    }
};

/**
 * 任意 mod 拡張
 * 異なる mod で NTT を複数回計算し, garner のアルゴリズムを適用する
 * 2^30 程度なら一意に結果を復元できる
 */

```