多项式全家桶

```
#include <bits/stdc++.h>
using namespace std;
const int D = 18, mod = 998244353, G = 3;
typedef unsigned long long ull;
typedef unsigned long long ll;

int power(int a, int b) {
    long long res = a, ans = 1;
    for (; b; b >>= 1, res = res * res % mod) if (b & 1) ans = ans * res % mod;
    return ans;
}
int Mod(int x) {return x >= mod ? x - mod : x;}
void SMod(int &x) { if (x >= mod) x -= mod; }
struct mint {
    int x;
    mint() {x = 0;}
    mint(int y) {x = y;}
    mint inv() const { return mint{power(x, mod - 2)}; }
    explicit operator int() const { return x; }
    int operator == (const mint &b) const { return x == b.x; }
    int operator != (const mint &b) const { return x != b.x; }
};
mint operator + (mint a, mint b) { return Mod(a.x + b.x); }
mint operator - (mint a, mint b) { return Mod(a.x + mod - b.x); }
mint operator * (mint a, mint b) { return 1ll * a.x * b.x % mod; }
mint operator - (mint a) { return Mod(mod - a.x); }
mint power(mint a, int b) {
    mint ans = 1;
    for (; b; b >>= 1, a = a * a) if (b & 1) ans = ans * a;
    return ans;
}

// mint msqrt(mint x) {     // 二次剩余
//     if (power(x, (mod - 1) / 2) != 1) return -1;
//     while (1) {
//         mint cur = rand() % mod;
//         if (power(cur * cur - x, (mod - 1) / 2) == 1) continue;
//         pair < mint, mint > res(cur, 1), ans(1, 0);
//         cur = cur * cur - x;
//         auto mult = [&](pair <mint, mint> a, pair <mint, mint> b) {
//             return make_pair(a.first * b.first + a.second * b.second * cur, a.second
* b.first + a.first * b.second);
//         };
//         for (int b = (mod + 1) / 2; b; b >>= 1, res = mult(res, res)) if (b & 1) ans
= mult(ans, res);
//         return min(ans.first.x, mod - ans.first.x);
//     }
// }
```

```
mint fac[1 << D | 10], facinv[1 << D | 10], inv[1 << D | 10];
namespace Poly {
    typedef vector <mint> poly;
    vector <int> gpower[D];
    ull nttf[1 << D | 10];
    int rev[1 << D | 10];
    int len(const poly &x) {return x.size();}
    void init() {
        for (int i = 0; i < D; i++) {
            gpower[i].resize(2 << i);
            int wn = power(G, (mod - 1) >> i + 1);
            for (int j = 0, w = 1; j < 2 << i; j++, w = 1ll * w * wn % mod) gpower[i]
[j] = w;
        }
        inv[1] = 1;
        for (int i = 2; i <= 1 << D; i++) inv[i] = (mod - mod / i) * inv[mod % i];
        fac[0] = facinv[0] = 1;
        for (int i = 1; i <= 1 << D; i++) fac[i] = fac[i - 1] * i, facinv[i] = facinv[i
- 1] * inv[i];
    }
    void get_rev(int l) {
        static int lstl = -1;
        if (l == lstl) return;
        lstl = l;
        for (int i = 1; i < 1 << l; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << l -
1);
    }
    void NTT(ull *a, int n, int type) {
        for (int i = 0; i < n; i++) if (rev[i] < i) swap(a[i], a[rev[i]]);
        for (int p = 1, d = 0; p < n; p <<= 1, d++) {
            if (d == 17) for (int i = 0; i < n; i++) a[i] %= mod;
            for (int s = 0; s < n; s += p << 1) {
                for (int *w = gpower[d].data(), i = s; i < s + p; i++) {
                    int h1 = *w++ * a[i + p] % mod;
                    a[i + p] = a[i] + mod - h1;
                    a[i] += h1;
                }
            }
        }
        if (type == -1) {
            int inv = power(n, mod - 2);
            for (int i = 0; i < n; i++) a[i] = a[i] * inv % mod;
            reverse(a + 1, a + n);
        }
        else for (int i = 0; i < n; i++) a[i] %= mod;
    }
    void NTT(poly &a, int l, int type) {
        get_rev(l);
        for (int i = 0; i < 1 << l; i++) nttf[i] = a[i].x;
        NTT(nttf, 1 << l, type);
        for (int i = 0; i < 1 << l; i++) a[i].x = nttf[i];
    }
```

```cpp
poly operator * (poly a, poly b) {
    int n = len(a), m = len(b);
    int l = 0;
    for (; 1 << l < n + m - 1; l++);
    a.resize(1 << l), b.resize(1 << l);
    NTT(a, l, 1), NTT(b, l, 1);
    for (int i = 0; i < 1 << l; i++) a[i] = a[i] * b[i];
    NTT(a, l, -1);
    a.resize(n + m - 1);
    return a;
}
// 48 - 107 必抄
poly operator + (poly a, const poly &b) {
    a.resize(max(len(a), len(b)));
    for (int i = 0; i < len(b); i++) a[i] = a[i] + b[i];
    return a;
}
poly operator - (poly a, const poly &b) {
    a.resize(max(len(a), len(b)));
    for (int i = 0; i < len(b); i++) a[i] = a[i] - b[i];
    return a;
}
poly operator * (poly a, mint b) {
    for (auto &i : a) i = i * b;
    return a;
}

poly Inv(const poly &a, int size) {    // 逆
    poly ans;
    ans.push_back(a[0].inv());
    for (int l = 0; 1 << l < size; l++) {
        poly b = ans;
        b.resize(2 << l);
        poly tmp(a.begin(), min(a.end(), a.begin() + (2 << l)));
        tmp.resize(2 << l);
        NTT(tmp, l + 1, 1), NTT(b, l + 1, 1);
        for (int i = 0; i < 2 << l; i++) tmp[i] = tmp[i] * b[i];
        NTT(tmp, l + 1, -1);
        for (int i = 0; i < 1 << l; i++) tmp[i] = 0;
        NTT(tmp, l + 1, 1);
        for (int i = 0; i < 2 << l; i++) tmp[i] = tmp[i] * b[i];
        NTT(tmp, l + 1, -1);
        ans.resize(2 << l);
        for (int i = 1 << l; i < 2 << l; i++) ans[i] = 0 - tmp[i];
    }
    ans.resize(size);
    return ans;
}

poly Der(poly a) { // 求导
    for (int i = 1; i < len(a); i++) a[i - 1] = i * a[i];
    a.pop_back();
    return a;
    }
    poly Int(poly a) {
        a.push_back(0);
        for (int i = len(a); i --> 1; ) a[i] = a[i - 1] * inv[i];
        a[0] = 0;
        return a;
    }
    poly Ln(const poly &a, int size) {
        poly ans = Int(Inv(a, size) * Der(a));
        ans.resize(size);
        return ans;
    }

    poly Exp(const poly &a, int size) {
        poly ans;
        ans.push_back(1);
        for (int l = 0; 1 << l < size; l++) {
            poly b = ans, tmp(a.begin(), min(a.end(), a.begin() + (2 << l)));
            b.resize(2 << l), tmp.resize(2 << l);
            tmp = tmp - Ln(ans, 2 << l);
            NTT(b, l + 1, 1), NTT(tmp, l + 1, 1);
            for (int i = 0; i < 2 << l; i++) tmp[i] = tmp[i] * b[i];
            NTT(tmp, l + 1, -1);
            ans.resize(2 << l);
            for (int i = 1 << l; i < 2 << l; i++) ans[i] = tmp[i];
        }
        ans.resize(size);
        return ans;
    }
    poly T(poly a) {
        reverse(a.begin(), a.end());
        return a;
    }
    poly Div(const poly &a, const poly &b) {
        if (len(a) < len(b)) return poly();
        int l = len(a) - len(b) + 1;
        poly ans = T(a) * Inv(T(b), l);
        ans.resize(l);
        return T(ans);
    }
    poly Mod(const poly &a, const poly &b) {
        poly ans = a - Div(a, b) * b;
        ans.resize(len(b) - 1);
        return ans;
    }
    mint calcVal(const poly &a, mint b) {
        mint ans = 0;
        for (int i = len(a); i --> 0; ) ans = ans * b + a[i];
        return ans;
    }
    poly Pow(poly a, mint b, int size) {
        int n = len(a);
        return Exp(Ln(a, size) * b, size);
    }
    poly Sqrt(poly a, int size) {
        mint st = msqrt(a[0]), sti = a[0].inv();
        for (auto &i : a) i = i * sti;
        a = Pow(a, inv[2], size);
        for (auto &i : a) i = i * st;
        return a;
    }
    namespace QuickCalc {
        mint *x, *y;
        poly *p;
        void build(int cur, int l, int r) {
            if (l == r) return void(p[cur] = poly{1, -x[l]});
            int mid = l + r >> 1;
            build(cur << 1, l, mid);
            build(cur << 1 | 1, mid + 1, r);
            p[cur] = p[cur << 1] * p[cur << 1 | 1];
        }
        poly Tmult(poly a, poly b) {
            b = T(b);
            int l = 0, n = len(a), m = len(b);
            for (; 1 << l < n; l++);
            a.resize(1 << l), b.resize(1 << l);
            NTT(a, l, 1), NTT(b, l, 1);
            for (int i = 0; i < 1 << l; i++) a[i] = a[i] * b[i];
            NTT(a, l, -1);
            int ansl = n - m + 1;
            poly ans(ansl);
            for (int i = 0; i < ansl; i++) ans[i] = a[m - 1 + i];
            return ans;
        }
        void calc(const poly &cur, int x, int l, int r) {
            if (l == r) return void(y[l] = cur[0]);
            int mid = l + r >> 1;
            calc(Tmult(cur, p[x << 1 | 1]), x << 1, l, mid);
            calc(Tmult(cur, p[x << 1]), x << 1 | 1, mid + 1, r);
        }
        void quickeva(poly f, int m, mint *X, mint *Y) {
            if (m == 0) return;
            int l = 0, n = len(f);
            for (; 1 << l < m; l++);
            p = new poly[(2 << l) + 1];
            x = X, y = Y;
            build(1, 0, m - 1);
            f.resize(n + m - 1);
            calc(Tmult(f, Inv(p[1], n)), 1, 0, m - 1);
            delete[] p;
            x = y = nullptr;
        }
        poly calc2(int x, int l, int r) {
            if (l == r) return poly({y[l]});
            int mid = l + r >> 1;
```

```cpp
            return calc2(x << 1, l, mid) * p[x << 1 | 1] + calc2(x << 1 | 1, mid + 1,
r) * p[x << 1];
        }
        poly quickint(int n, mint *X, mint *Y) {
            if (n == 0) return poly();
            int l = 0;
            for (; 1 << l < n; l++);
            p = new poly[(2 << l) + 1];
            x = X;
            y = new mint[n];
            build(1, 0, n - 1);
            poly f = Der(T(p[1]));
            f.resize(n + n - 1);
            calc(Tmult(f, Inv(p[1], n)), 1, 0, n - 1);
            for (int i = 0; i < n; i++) y[i] = Y[i] * y[i].inv();
            poly ans = calc2(1, 0, n - 1);
            delete[] p;
            delete[] y;
            x = y = nullptr;
            return T(ans);
        }
    }
    using QuickCalc :: quickeva;
    using QuickCalc :: quickint;
}
using Poly :: poly;
using Poly :: operator *;
using Poly :: operator +;
using Poly :: operator -;

int main() {
    Poly :: init(); // 必写
    int n, k;
    scanf("%d%d", &n, &k);
    Poly :: poly x;
    x.resize(++n);
    for (int i = 0; i < n; i++) scanf("%d", &x[i].x);
    Poly :: poly s = Poly :: Exp(Poly :: Int(Poly :: Inv(Poly :: Sqrt(x, n), n)), n);
    x[0] = 2;
    x = Poly :: Ln(x - s, n);
    x[0] = 1;
    x = Poly :: Der(Poly :: Pow(x, k, n));
    for (int i = 0; i < n - 1; i++) printf("%d%c", x[i].x, " \n"[i == n - 2]);
}
```