ccpc桂林更新 2023.10.26

fft

```cpp
const int N=1e6+10;
const long double Pi=acos(-1);
int n,m,limit;
int l, r[1<<20];

const int EX = 20;
const long double pi = 3.1415926535897932384626433832795028841971;
// Complex number
pair<long double, long double> operator+(const pair<long double, long double> &a, const pair<long double, long double> &b) {
    return {a.x + b.x, a.y + b.y};
}
pair<long double, long double> operator-(const pair<long double, long double> &a, const pair<long double, long double> &b) {
    return {a.x - b.x, a.y - b.y};
}
pair<long double, long double> operator*(const pair<long double, long double> &a, const pair<long double, long double> &b) {
    return {a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x};
}
void FFT(pair<long double, long double> *a,int op){
    for(int i=0; i<limit; i++){
        if(i<r[i]) swap(a[i],a[r[i]]);
    }
    for(int mid=1; mid<limit; mid<<=1){
        pair<long double, long double> W(cos(Pi/mid),op*sin(Pi/mid));
        for(int r=mid<<1,j=0; j<limit; j+=r){
            pair<long double, long double> w(1,0);
            for(int l=0; l<mid; l++,w=w*W){
                pair<long double, long double> x=a[j+l],y=w*a[j+mid+l];
                a[j+l]=x+y; a[j+mid+l]=x-y;
            }
        }
    }
}

vector<int> operator*(const vector<int> &a, const vector<int> &b) {
    int n = a.size(), m = b.size();
    vector<int> ans(n + m - 1);
    n--, m--;
    limit = 1;
    l = 0;
    while (limit <= n + m)
        limit <<= 1, l++;
    for (int i = 1; i <= limit; i++)
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    pair<long double, long double> *c = new pair<long double, long double>[limit]();
    for (int i = 0; i <= n; i++)
        c[i].x = a[i];
    for (int i = 0; i <= m; i++)
        c[i].y = b[i];
    FFT(c, 1);
    for (int i = 0; i < limit; i++)
        c[i] = c[i] * c[i];
    FFT(c, -1);
    for (int i = 0; i <= n + m; i++)
        ans[i] = c[i].y / (limit << 1) + 0.5;
    delete[] c;
    return ans;
}
```

jsjh

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

struct Point { double x, y; };        // 点
using Vec = Point;                    // 向量
struct Line { Point P; Vec v; };      // 直线（点向式）
struct Seg { Point A, B; };           // 线段（存两个端点）
struct Circle { Point O; double r; }; // 圆（存圆心和半径）

const Point O = {0, 0};                        // 原点
const Line Ox = {O, {1, 0}}, Oy = {O, {0, 1}}; // 坐标轴
const double PI = acos(-1), EPS = 1e-9;

bool eq(double a, double b) { return abs(a - b) < EPS; } // ==
bool gt(double a, double b) { return a - b > EPS; }      // >
bool lt(double a, double b) { return a - b < -EPS; }     // <
bool ge(double a, double b) { return a - b > -EPS; }     // >=
bool le(double a, double b) { return a - b < EPS; }      // <=

Vec r90a(Vec v) { return {-v.y, v.x}; }                    // 逆时针旋转90度的向量
Vec r90c(Vec v) { return {v.y, -v.x}; }                    // 顺时针旋转90度的向量
Vec operator+(Vec u, Vec v) { return {u.x + v.x, u.y + v.y}; }   // 向量加向量
Vec operator-(Vec u, Vec v) { return {u.x - v.x, u.y - v.y}; }   // 向量减向量
Vec operator*(double k, Vec v) { return {k * v.x, k * v.y}; }    // 数乘
double operator*(Vec u, Vec v) { return u.x * v.x + u.y * v.y; } // 点乘
double operator^(Vec u, Vec v) { return u.x * v.y - u.y * v.x; } // 叉乘
double len(Vec v) { return sqrt(v.x * v.x + v.y * v.y); }        // 向量长度
double slope(Vec v) { return v.y / v.x; }                        // 斜率 // NOTE 不要用isinf判断斜率不存在，用后面的paral_y

// 两向量的夹角余弦
// DEPENDS len, V*V
double cos_t(Vec u, Vec v) { return u * v / len(u) / len(v); }

// 归一化向量（与原向量方向相同的单位向量）
// DEPENDS len
Vec norm(Vec v) { return {v.x / len(v), v.y / len(v)}; }

```

```cpp
// 与原向量平行且横坐标大于等于0的单位向量
// DEPENDS d*V, len
Vec pnorm(Vec v) { return (v.x < 0 ? -1 : 1) / len(v) * v; }

// 线段的方向向量
// DEPENDS V-V
// NOTE 直线的方向向量直接访问属性v
Vec dvec(Seg l) { return l.B - l.A; }

// 两点式直线
// DEPENDS V-V
Line line(Point A, Point B) { return {A, B - A}; }

// 斜截式直线
Line line(double k, double b) { return {{0, b}, {1, k}}; }

// 点斜式直线
Line line(Point P, double k) { return {P, {1, k}}; }

// 线段所在直线
// DEPENDS V-V
Line line(Seg l) { return {l.A, l.B - l.A}; }

// 给定直线的横坐标求纵坐标
// NOTE 请确保直线不与y轴平行
double at_x(Line l, double x) { return l.P.y + (x - l.P.x) * l.v.y / l.v.x; }

// 给定直线的纵坐标求横坐标
// NOTE 请确保直线不与x轴平行
double at_y(Line l, double y) { return l.P.x - (y + l.P.y) * l.v.x / l.v.y; }

// 点到直线的垂足
// DEPENDS V-V, V*V, d*V
Point pedal(Point P, Line l) { return l.P - (l.P - P) * l.v / (l.v * l.v) * l.v; }

// 过某点作直线的垂线
// DEPENDS r90c
Line perp(Line l, Point P) { return {P, r90c(l.v)}; }

// 角平分线
// DEPENDS V+V, len, norm
Line bisec(Point P, Vec u, Vec v) { return {P, norm(u) + norm(v)}; }

// 线段的方向向量
// DEPENDS V-V
// NOTE 直线的方向向量直接访问属性v
Vec dvec(Seg l) { return l.B - l.A; }

// 线段中点
Point midp(Seg l) { return {(l.A.x + l.B.x) / 2, (l.A.y + l.B.y) / 2}; }

// 线段中垂线
// DEPENDS r90c, V-V, midp
Line perp(Seg l) { return {midp(l), r90c(l.B - l.A)}; }

```

```cpp
// 向量是否互相垂直
// DEPENDS eq, V*V
bool verti(Vec u, Vec v) { return eq(u * v, 0); }

// 向量是否互相平行
// DEPENDS eq, V^V
bool paral(Vec u, Vec v) { return eq(u ^ v, 0); }

// 向量是否与x轴平行
// DEPENDS eq
bool paral_x(Vec v) { return eq(v.y, 0); }

// 向量是否与y轴平行
// DEPENDS eq
bool paral_y(Vec v) { return eq(v.x, 0); }

// 点是否在直线上
// DEPENDS eq
bool on(Point P, Line l) { return eq((P.x - l.P.x) * l.v.y, (P.y - l.P.y) * l.v.x); }

// 点是否在线段上
// DEPENDS eq, len, V-V
bool on(Point P, Seg l) { return eq(len(P - l.A) + len(P - l.B), len(l.A - l.B)); }

// 两个点是否重合
// DEPENDS eq
bool operator==(Point A, Point B) { return eq(A.x, B.x) && eq(A.y, B.y); }

// 两条直线是否重合
// DEPENDS eq, on(L)
bool operator==(Line a, Line b) { return on(a.P, b) && on(a.P + a.v, b); }

// 两条线段是否重合
// DEPENDS eq, P==P
bool operator==(Seg a, Seg b) { return (a.A == b.A && a.B == b.B) || (a.A == b.B && a.B == b.A); }

// 以横坐标为第一关键词、纵坐标为第二关键词比较两个点
// DEPENDS eq, lt
bool operator<(Point A, Point B) { return lt(A.x, B.x) || (eq(A.x, B.x) && lt(A.y, B.y)); }

// 直线与圆是否相切
// DEPENDS eq, V^V, len
bool tangency(Line l, Circle C) { return eq(abs((C.O ^ l.v) - (l.P ^ l.v)), C.r * len(l.v)); }

// 圆与圆是否相切
// DEPENDS eq, V-V, len
bool tangency(Circle C1, Circle C2) { return eq(len(C1.O - C2.O), C1.r + C2.r); }

// 两点间的距离
// DEPENDS len, V-V
double dis(Point A, Point B) { return len(A - B); }

// 点到直线的距离
// DEPENDS V^V, len
double dis(Point P, Line l) { return abs((P ^ l.v) - (l.P ^ l.v)) / len(l.v); }
```

```cpp

// 平行直线间的距离
// DEPENDS d*V, V^V, len, pnorm
// NOTE 请确保两直线是平行的
double dis(Line a, Line b) { return abs((a.P ^ pnorm(a.v)) - (b.P ^ pnorm(b.v))); }

// 平移
// DEPENDS V+V
Line operator+(Line l, Vec v) { return {l.P + v, l.v}; }
Seg operator+(Seg l, Vec v) { return {l.A + v, l.B + v}; }

// 旋转
// DEPENDS V+V, V-V
Point rotate(Point P, double rad) { return {cos(rad) * P.x - sin(rad) * P.y, sin(rad) * P.x + cos(rad) * P.y}; }
Point rotate(Point P, double rad, Point C) { return C + rotate(P - C, rad); }                  // DEPENDS ^1
Line rotate(Line l, double rad, Point C = O) { return {rotate(l.P, rad, C), rotate(l.v, rad)}; }  // DEPENDS ^1, ^2
Seg rotate(Seg l, double rad, Point C = O) { return {rotate(l.A, rad, C), rotate(l.B, rad, C)}; } // DEPENDS ^1, ^2

// 对称
// 关于点对称
Point reflect(Point A, Point P) { return {P.x * 2 - A.x, P.y * 2 - A.y}; }
Line reflect(Line l, Point P) { return {reflect(l.P, P), l.v}; }           // DEPENDS ^1
Seg reflect(Seg l, Point P) { return {reflect(l.A, P), reflect(l.B, P)}; } // DEPENDS ^1
// 关于直线对称
// DEPENDS V-V, V*V, d*V, pedal
// NOTE 向量和点在这里的表现不同，求向量关于某直线的对称向量需要用reflect_v
Point reflect(Point A, Line ax) { return reflect(A, pedal(A, ax)); }             // DEPENDS ^1
Vec reflect_v(Vec v, Line ax) { return reflect(v, ax) - reflect(O, ax); }        // DEPENDS ^1, ^4
Line reflect(Line l, Line ax) { return {reflect(l.P, ax), reflect_v(l.v, ax)}; } // DEPENDS ^1, ^4, ^5
Seg reflect(Seg l, Line ax) { return {reflect(l.A, ax), reflect(l.B, ax)}; }     // DEPENDS ^1, ^4

// 直线与直线交点
// DEPENDS eq, d*V, V*V, V+V, V^V
vector<Point> inter(Line a, Line b)
{
    double c = a.v ^ b.v;
    if (eq(c, 0)) return {};
    Vec v = 1 / c * Vec{a.P ^ (a.P + a.v), b.P ^ (b.P + b.v)};
    return {{v * Vec{-b.v.x, a.v.x}, v * Vec{-b.v.y, a.v.y}}};
}

// 直线与圆交点
// DEPENDS eq, gt, V+V, V-V, V*V, d*V, len, pedal
vector<Point> inter(Line l, Circle C)
{
    Point P = pedal(C.O, l);
    double h = len(P - C.O);
    if (gt(h, C.r)) return {};
    if (eq(h, C.r)) return {P};
    double d = sqrt(C.r * C.r - h * h);
    Vec vec = d / len(l.v) * l.v;
    return {P + vec, P - vec};
}

// 圆与圆的交点
// DEPENDS eq, gt, V+V, V-V, d*V, len, r90c
vector<Point> inter(Circle C1, Circle C2)
{
    Vec v1 = C2.O - C1.O, v2 = r90c(v1);
    double d = len(v1);
    if (gt(d, C1.r + C2.r) || gt(abs(C1.r - C2.r), d)) return {};
    if (eq(d, C1.r + C2.r) || eq(d, abs(C1.r - C2.r))) return {C1.O + C1.r / d * v1};
    double a = ((C1.r * C1.r - C2.r * C2.r) / d + d) / 2;
    double h = sqrt(C1.r * C1.r - a * a);
    Vec av = a / len(v1) * v1, hv = h / len(v2) * v2;
    return {C1.O + av + hv, C1.O + av - hv};
}

// 三角形的重心
Point barycenter(Point A, Point B, Point C)
{
    return {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};
}

// 三角形的外心
// DEPENDS r90c, V*V, d*V, V-V, V+V
// NOTE 给定圆上三点求圆，要先判断是否三点共线
Point circumcenter(Point A, Point B, Point C)
{
    double a = A * A, b = B * B, c = C * C;
    double d = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
    return 1 / d * r90c(a * (B - C) + b * (C - A) + c * (A - B));
}

// 三角形的内心
// DEPENDS len, d*V, V-V, V+V
Point incenter(Point A, Point B, Point C)
{
    double a = len(B - C), b = len(A - C), c = len(A - B);
    double d = a + b + c;
    return 1 / d * (a * A + b * B + c * C);
}

// 三角形的垂心
// DEPENDS V*V, d*V, V-V, V^V, r90c
Point orthocenter(Point A, Point B, Point C)
{
    double n = B * (A - C), m = A * (B - C);
    double d = (B - C) ^ (A - C);
    return 1 / d * r90c(n * (C - B) - m * (C - A));
}
```

mo

```cpp
const int N=2e5+10;
const int mod=998244353;
template<const int T>
struct ModInt {
    const static int mod = T;
    int x;
    ModInt(int x = 0) : x(x % mod) {}
    ModInt(long long x) : x(int(x % mod)) {}
    int val() { return x; }
    ModInt operator + (const ModInt &a) const { int x0 = x + a.x; return ModInt(x0 < mod ? x0 : x0 - mod); }
    ModInt operator - (const ModInt &a) const { int x0 = x - a.x; return ModInt(x0 < 0 ? x0 + mod : x0); }
    ModInt operator * (const ModInt &a) const { return ModInt(1LL * x * a.x % mod); }
    ModInt operator / (const ModInt &a) const { return *this * a.inv(); }
    void operator += (const ModInt &a) { x += a.x; if (x >= mod) x -= mod; }
    void operator -= (const ModInt &a) { x -= a.x; if (x < 0) x += mod; }
    void operator *= (const ModInt &a) { x = 1LL * x * a.x % mod; }
    void operator /= (const ModInt &a) { *this = *this / a; }
    friend ostream &operator<<(ostream &os, const ModInt &a) { return os << a.x;}

    ModInt pow(int64_t n) const {
        ModInt res(1), mul(x);
        while(n){
            if (n & 1) res *= mul;
            mul *= mul;
            n >>= 1;
        }
        return res;
    }

    ModInt inv() const {
        int a = x, b = mod, u = 1, v = 0;
        while (b) {
            int t = a / b;
            a -= t * b; swap(a, b);
            u -= t * v; swap(u, v);
        }
        if (u < 0) u += mod;
        return u;
    }

};
typedef ModInt<998244353> mint;
```

ntt

```cpp
// vector²»±ä´ó
using i64 = long long;
constexpr int mod = 998244353;
int norm(int x) {
    if (x < 0) {
        x += mod;
    }
    if (x >= mod) {
        x -= mod;
    }
    return x;
}
template<class T>
T power(T a, int b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(mod - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, mod - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % mod;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend istream &operator>>(istream &is, Z &a) {
        i64 v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend ostream &operator<<(ostream &os, const Z &a) {
        return os << a.val();
    }
};
vector<int> rev;
vector<Z> roots{0, 1};
void dft(vector<Z> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i ++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }
    for (int i = 0; i < n; i ++) {
        if (rev[i] < i) {
            swap(a[i], a[rev[i]]);
        }
    }
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            Z e = power(Z(3), (mod - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i ++) {
                roots[i << 1] = roots[i];
                roots[i << 1 | 1] = roots[i] * e;
            }
            k ++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j ++) {
                Z u = a[i + j], v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v, a[i + j + k] = u - v;
            }
        }
    }
}
```

```
void idft(vector<Z> &a) {
    int n = a.size();
    reverse(a.begin() + 1, a.end());
    dft(a);
    Z inv = (1 - mod) / n;
    for (int i = 0; i < n; i ++) {
        a[i] *= inv;
    }
}
struct Poly {
    vector<Z> a;
    Poly() {}
    Poly(const vector<Z> &a) : a(a) {}
    Poly(const initializer_list<Z> &a) : a(a) {}
    int size() const {
        return a.size();
    }
    void resize(int n) {
        a.resize(n);
    }
    Z operator[](int idx) const {
        if (idx < size()) {
            return a[idx];
        } else {
            return 0;
        }
    }
    Z &operator[](int idx) {
        return a[idx];
    }
    Poly mulxk(int k) const {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return Poly(b);
    }
    Poly modxk(int k) const {
        k = min(k, size());
        return Poly(vector<Z>(a.begin(), a.begin() + k));
    }
    Poly divxk(int k) const {
        if (size() <= k) {
            return Poly();
        }
        return Poly(vector<Z>(a.begin() + k, a.end()));
    }
    friend Poly operator+(const Poly &a, const Poly &b) {
        vector<Z> res(max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i ++) {
            res[i] = a[i] + b[i];
        }
        return Poly(res);
    }
    friend Poly operator-(const Poly &a, const Poly &b) {
        vector<Z> res(max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i ++) {
            res[i] = a[i] - b[i];
        }
        return Poly(res);
    }
    friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot) {
            sz *= 2;
        }
        a.a.resize(sz);
        b.a.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; i ++) {
            a.a[i] = a[i] * b[i];
        }
        idft(a.a);
        a.resize(tot);
        return a;
    }
    friend Poly operator*(Z a, Poly b) {
        for (int i = 0; i < int(b.size()); i ++) {
            b[i] *= a;
        }
        return b;
    }
    friend Poly operator*(Poly a, Z b) {
        for (int i = 0; i < int(a.size()); i ++) {
            a[i] *= b;
        }
        return a;
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    Poly deriv() const {
        if (a.empty()) {
            return Poly();
        }
        vector<Z> res(size() - 1);
        for (int i = 0; i < size() - 1; i ++) {
            res[i] = (i + 1) * a[i + 1];
        }
        return Poly(res);
    }
    Poly integr() const {
        vector<Z> res(size() + 1);
        for (int i = 0; i < size(); i ++) {
            res[i + 1] = a[i] / (i + 1);
        }
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x{a[0].inv()};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const {
        return (deriv() * inv(m)).integr().modxk(m);
    }
    Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly pow(int k, int m) const {
        int i = 0;
        while (i < size() && a[i].val() == 0) {
            i ++;
        }
        if (i == size() || 1LL * i * k >= m) {
            return Poly(vector<Z>(m));
        }
        Z v = a[i];
        auto f = divxk(i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
    }
    Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((mod + 1) / 2);
        }
        return x.modxk(m);
    }
    Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
};

// vectorä½šå•˜å¤§ï¼Œæœ•å¤šä¸•èƒ½è¶…è¿‡15æ¬¡ntt
#define int long long
const int md = 998244353;

inline void add(int &x, int y) {
    x += y;
    if (x >= md) {
        x -= md;
    }
}

inline void sub(int &x, int y) {
    x -= y;
    if (x < 0) {
        x += md;
    }
}

inline int mul(int x, int y) {
    return (long long) x * y % md;
}

inline int power(int x, int y) {
    int res = 1;
    for (; y; y >>= 1, x = mul(x, x)) {
        if (y & 1) {
        res = mul(res, x);
        }
    }
    return res;
}

inline int inv(int a) {
    a %= md;
    if (a < 0) {
        a += md;
    }
    int b = md, u = 0, v = 1;
    while (a) {
        int t = b / a;
        b -= t * a;
        swap(a, b);
        u -= t * v;
        swap(u, v);
    }
    if (u < 0) {
        u += md;
    }
    return u;
}

namespace ntt {
    int base = 1, root = -1, max_base = -1;
    vector<int> rev = {0, 1}, roots = {0, 1};

    void init() {
        int temp = md - 1;
        max_base = 0;
        while (temp % 2 == 0) {
            temp >>= 1;
            ++max_base;
        }
        root = 2;
        while (true) {
            if (power(root, 1 << max_base) == 1 && power(root, 1 << (max_base - 1)) != 1) {
                break;
            }
            ++root;
        }
    }

    void ensure_base(int nbase) {
        if (max_base == -1) {
            init();
        }
        if (nbase <= base) {
            return;
        }
        assert(nbase <= max_base);
        rev.resize(1 << nbase);
        for (int i = 0; i < 1 << nbase; ++i) {
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (nbase - 1));
        }
        roots.resize(1 << nbase);
        while (base < nbase) {
            int z = power(root, 1 << (max_base - 1 - base));
            for (int i = 1 << (base - 1); i < 1 << base; ++i) {
            roots[i << 1] = roots[i];
            roots[i << 1 | 1] = mul(roots[i], z);
            }
            ++base;
        }
    }

    void dft(vector<int> &a) {
        int n = a.size(), zeros = __builtin_ctz(n);
        ensure_base(zeros);
        int shift = base - zeros;
        for (int i = 0; i < n; ++i) {
            if (i < rev[i] >> shift) {
            swap(a[i], a[rev[i] >> shift]);
            }
        }
        for (int i = 1; i < n; i <<= 1) {
            for (int j = 0; j < n; j += i << 1) {
                for (int k = 0; k < i; ++k) {
                    int x = a[j + k], y = mul(a[j + k + i], roots[i + k]);
                    a[j + k] = (x + y) % md;
                    a[j + k + i] = (x + md - y) % md;
                }
            }
        }
    }

    vector<int> multiply(vector<int> a, vector<int> b) {
        int need = a.size() + b.size() - 1, nbase = 0;
        while (1 << nbase < need) {
            ++nbase;
        }
        ensure_base(nbase);
        int sz = 1 << nbase;
        a.resize(sz);
        b.resize(sz);
        bool equal = a == b;
        dft(a);
        if (equal) {
            b = a;
        } else {
            dft(b);
        }
        int inv_sz = inv(sz);
        for (int i = 0; i < sz; ++i) {
            a[i] = mul(mul(a[i], b[i]), inv_sz);
        }
        reverse(a.begin() + 1, a.end());
        dft(a);
        a.resize(need);
        return a;
    }

    vector<int> inverse(vector<int> a) {
        int n = a.size(), m = (n + 1) >> 1;
        if (n == 1) {
            return vector<int>(1, inv(a[0]));
        }
        else
        {
            vector<int> b = inverse(vector<int>(a.begin(), a.begin() + m));
            int need = n << 1, nbase = 0;
            while (1 << nbase < need) {
                ++nbase;
            }
            ensure_base(nbase);
            int sz = 1 << nbase;
            a.resize(sz);
            b.resize(sz);
            dft(a);
            dft(b);
            int inv_sz = inv(sz);
            for (int i = 0; i < sz; ++i) {
                a[i] = mul(mul(md + 2 - mul(a[i], b[i]), b[i]), inv_sz);
            }
            reverse(a.begin() + 1, a.end());
            dft(a);
            a.resize(n);
            return a;
        }
    }
}

using ntt::multiply;
using ntt::inverse;

vector<int>& operator += (vector<int> &a, const vector<int> &b) {
    if (a.size() < b.size()) {
        a.resize(b.size());
    }
    for (int i = 0; i < b.size(); ++i) {
        add(a[i], b[i]);
    }
    return a;
}

vector<int> operator + (const vector<int> &a, const vector<int> &b) {
    vector<int> c = a;
    return c += b;
}

vector<int>& operator -= (vector<int> &a, const vector<int> &b) {
    if (a.size() < b.size()) {
        a.resize(b.size());
    }
    for (int i = 0; i < b.size(); ++i) {
        sub(a[i], b[i]);
    }
    return a;
}

vector<int> operator - (const vector<int> &a, const vector<int> &b) {
    vector<int> c = a;
    return c -= b;
}

vector<int>& operator *= (vector<int> &a, const vector<int> &b) {
    if (min(a.size(), b.size()) < 128) {
        vector<int> c = a;
        a.assign(a.size() + b.size() - 1, 0);
        for (int i = 0; i < c.size(); ++i) {
        for (int j = 0; j < b.size(); ++j) {
            add(a[i + j], mul(c[i], b[j]));
        }
        }
    } else {
        a = multiply(a, b);
    }
    return a;
}
```

```

vector<int> operator * (const vector<int> &a, const vector<int> &b) {
    vector<int> c = a;
    return c *= b;
}

vector<int>& operator /= (vector<int> &a, const vector<int> &b) {
    int n = a.size(), m = b.size();
    if (n < m) {
        a.clear();
    } else {
        vector<int> c = b;
        reverse(a.begin(), a.end());
        reverse(c.begin(), c.end());
        c.resize(n - m + 1);
        a *= inverse(c);
        a.erase(a.begin() + n - m + 1, a.end());
        reverse(a.begin(), a.end());
    }
    return a;
}

vector<int> operator / (const vector<int> &a, const vector<int> &b) {
    vector<int> c = a;
    return c /= b;
}

vector<int>& operator %= (vector<int> &a, const vector<int> &b) {
    int n = a.size(), m = b.size();
    if (n >= m) {
        vector<int> c = (a / b) * b;
        a.resize(m - 1);
        for (int i = 0; i < m - 1; ++i) {
        sub(a[i], c[i]);
        }
    }
    return a;
}

vector<int> operator % (const vector<int> &a, const vector<int> &b) {
    vector<int> c = a;
    return c %= b;
}

vector<int> derivative(const vector<int> &a) {
    int n = a.size();
    vector<int> b(n - 1);
    for (int i = 1; i < n; ++i) {
        b[i - 1] = mul(a[i], i);
    }
    return b;
}

vector<int> primitive(const vector<int> &a) {
    int n = a.size();
    vector<int> b(n + 1), invs(n + 1);
    for (int i = 1; i <= n; ++i) {
        invs[i] = i == 1 ? 1 : mul(md - md / i, invs[md % i]);
        b[i] = mul(a[i - 1], invs[i]);
    }
    return b;
}

vector<int> logarithm(const vector<int> &a) {
    vector<int> b = primitive(derivative(a) * inverse(a));
    b.resize(a.size());
    return b;
}

vector<int> exponent(const vector<int> &a) {
    vector<int> b(1, 1);
    while (b.size() < a.size()) {
        vector<int> c(a.begin(), a.begin() + min(a.size(), b.size() << 1));
        add(c[0], 1);
        vector<int> old_b = b;
        b.resize(b.size() << 1);
        c -= logarithm(b);
        c *= old_b;
        for (int i = b.size() >> 1; i < b.size(); ++i) {
            b[i] = c[i];
        }
    }
    b.resize(a.size());
    return b;
}

vector<int> power(vector<int> a, int m) {
    int n = a.size(), p = -1;
    vector<int> b(n);
    for (int i = 0; i < n; ++i) {
        if (a[i]) {
        p = i;
        break;
        }
    }
    if (p == -1) {
        b[0] = !m;
        return b;
    }
    if ((long long) m * p >= n) {
        return b;
    }
    int mu = power(a[p], m), di = inv(a[p]);
    vector<int> c(n - m * p);
    for (int i = 0; i < n - m * p; ++i) {
        c[i] = mul(a[i + p], di);
    }
    c = logarithm(c);
    for (int i = 0; i < n - m * p; ++i) {
        c[i] = mul(c[i], m);
    }
    c = exponent(c);
    for (int i = 0; i < n - m * p; ++i) {
        b[i + m * p] = mul(c[i], mu);
    }
    return b;
}

vector<int> sqrt(const vector<int> &a) {
    vector<int> b(1, 1);
    while (b.size() < a.size()) {
        vector<int> c(a.begin(), a.begin() + min(a.size(), b.size() << 1));
        vector<int> old_b = b;
        b.resize(b.size() << 1);
        c *= inverse(b);
        for (int i = b.size() >> 1; i < b.size(); ++i) {
        b[i] = mul(c[i], (md + 1) >> 1);
        }
    }
    b.resize(a.size());
    return b;
}

vector<int> multiply_all(int l, int r, vector<vector<int>> &all) {
    if (l > r) {
        return vector<int>();
    } else if (l == r) {
        return all[l];
    } else {
        int y = (l + r) >> 1;
        return multiply_all(l, y, all) * multiply_all(y + 1, r, all);
    }
}

vector<int> evaluate(const vector<int> &f, const vector<int> &x) {
    int n = x.size();
    if (!n) {
        return vector<int>();
    }
    vector<vector<int>> up(n * 2);
    for (int i = 0; i < n; ++i) {
        up[i + n] = vector<int>{(md - x[i]) % md, 1};
    }
    for (int i = n - 1; i; --i) {
        up[i] = up[i << 1] * up[i << 1 | 1];
    }
    vector<vector<int>> down(n * 2);
    down[1] = f % up[1];
    for (int i = 2; i < n * 2; ++i) {
        down[i] = down[i >> 1] % up[i];
    }
    vector<int> y(n);
    for (int i = 0; i < n; ++i) {
        y[i] = down[i + n][0];
    }
```

```cpp
    return y;
}

vector<int> interpolate(const vector<int> &x, const vector<int> &y) {
    int n = x.size();
    vector<vector<int>> up(n * 2);
    for (int i = 0; i < n; ++i) {
        up[i + n] = vector<int>{(md - x[i]) % md, 1};
    }
    for (int i = n - 1; i; --i) {
        up[i] = up[i << 1] * up[i << 1 | 1];
    }
    vector<int> a = evaluate(derivative(up[1]), x);
    for (int i = 0; i < n; ++i) {
        a[i] = mul(y[i], inv(a[i]));
    }
    vector<vector<int>> down(n * 2);
    for (int i = 0; i < n; ++i) {
        down[i + n] = vector<int>(1, a[i]);
    }
    for (int i = n - 1; i; --i) {
        down[i] = down[i << 1] * up[i << 1 | 1] + down[i << 1 | 1] * up[i << 1];
    }
    return down[1];
}
```