多项式全家桶

NTT全家桶

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

int read() {
    int res = 0, flag = 1;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-') flag = -1;
        c = getchar();
    }
    while (isdigit(c)) {
        res = (res << 1) + (res << 3) + (c ^ 48);
        c = getchar();
    }
    return res * flag;
}

void write(int x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x >= 10) write(x / 10);
    putchar('0' + x % 10);
}

void write(int x, char c) {
    write(x);
    putchar(c);
}

// #define debug(x) cout<<"[debug]"#x<<"="<<x<<endl
typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
const double eps = 1e-8;
const int inf = 0x3f3f3f3f;

#ifndef ONLINE_JUDGE
#define debug(...)
// #include<debug>
#else
#define debug(...)
#endif

const int N = 1000006;
```

```cpp
namespace NTT {
typedef vector<ll> Poly;
const int mod = 998244353, G = 3, Gi = 332748118;
const int M = 2e6 + 1e5;
int bit, tot;
int rev[M];
int inv[M];
int init_inv = []() {
    inv[0] = inv[1] = 1;
    for (int i = 2; i < M; i++)
        inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
    return 0;
}();
ll qmi(ll a, ll b, ll p) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % p;

        a = a * a % p;
        b >>= 1;
    }
    return res;
}
void NTT(Poly &a, int inv) {
    for (int i = 0; i < tot; i++)
        if (i < rev[i]) swap(a[i], a[rev[i]]);

    for (int mid = 1; mid < tot; mid *= 2) {
        ll w1 = qmi(inv == 1 ? G : Gi, (mod - 1) / (2 * mid), mod);
        for (int i = 0; i < tot; i += mid * 2) {
            ll wk = 1;
            for (int j = 0; j < mid; j++, wk = wk * w1 % mod) {
                ll x = a[i + j];
                ll y = wk * a[i + j + mid] % mod;
                a[i + j] = (x + y) % mod;
                a[i + j + mid] = (x - y + mod) % mod;
            }
        }
    }

    if (inv == -1)  // 就不用后面除了
    {
        ll intot = qmi(tot, mod - 2, mod);
        for (int i = 0; i < tot; i++) {
            a[i] = a[i] * intot % mod;
        }
```

```cpp
    }
}
Poly mul(Poly a, Poly b)  // deg是系数的数量，所以有0~deg-1次项
{
    int deg = (int)a.size() + b.size() - 1;
    bit = 0;
    while ((1 << bit) < deg) bit++;  // 至少要系数的数量
    tot = 1 << bit;                  // 系数项为0~tot-1
    for (int i = 0; i < tot; i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));

    Poly c(tot);
    a.resize(tot), b.resize(tot);
    NTT(a, 1), NTT(b, 1);
    for (int i = 0; i < tot; i++) c[i] = a[i] * b[i] % mod;
    NTT(c, -1);
    // c.resize(1000001);
    return c;
}
Poly operator*(Poly &a, Poly &b) { return mul(a, b); }
Poly operator*(Poly &a, int &t) {
    Poly res;
    for (int i = 0; i < a.size(); i++) res.push_back(1ll * a[i] * t % mod);
    return res;
}
Poly operator*(Poly &a, ll &t) {
    Poly res;
    for (int i = 0; i < a.size(); i++) res.push_back(a[i] * t % mod);
    return res;
}
Poly operator+(Poly &a, Poly &b) {
    Poly res(a);
    res.resize(max(a.size(), b.size()));
    for (int i = 0; i < b.size(); i++) res[i] = (res[i] + b[i]) % mod;
    return res;
}
Poly operator-(Poly &a, Poly &b) {
    Poly res(a);
    res.resize(max(a.size(), b.size()));
    for (int i = 0; i < b.size(); i++) res[i] = (res[i] - b[i] + mod) % mod;
    return res;
}
Poly Inv(
    Poly &f,
    int deg)  // 多项式f对于x^deg的逆元(注意rev[]等数组要开到2*deg的空间级别,f[]要开deg级别)
{
```

```
    if (deg == 1) return Poly(1, qmi(f[0], mod - 2, mod));

    Poly B = Inv(f, (deg + 1) >> 1);  // 上一个逆元
    Poly A(f.begin(), f.begin() + deg);

    bit = 0;
    while ((1 << bit) < (deg << 1)) bit++;  // 至少要系数的数量
    tot = 1 << bit;                         // 系数项为0~tot-1
    for (int i = 0; i < tot; i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));

    A.resize(tot), B.resize(tot);
    NTT(A, 1), NTT(B, 1);
    for (int i = 0; i < tot; i++)
        A[i] = B[i] * (2 - A[i] * B[i] % mod + mod) % mod;
    NTT(A, -1);
    A.resize(deg);
    return A;
}
Poly cdq_ntt(int l, int r, vector<Poly> &f) {
    if (l == r) return f[l];
    int mid = l + r >> 1;
    return mul(cdq_ntt(l, mid, f), cdq_ntt(mid + 1, r, f));
}
namespace Cipolla {
int W;
struct cp {
    ll x, y;
    cp(ll x = 0, ll y = 0) : x(x), y(y) {}
    cp operator+(const cp _) const {
        return {(x + _.x) % mod, (y + _.y) % mod};
    }
    cp operator*(const cp _) const {
        return {(x * _.x % mod + y * _.y % mod * W % mod) % mod,
                (x * _.y + y * _.x) % mod};
    }
};
cp qmi(cp a, int b) {
    cp res = 1;
    while (b) {
        if (b & 1) res = res * a;

        a = a * a;
        b >>= 1;
    }
    return res;
```

```
}
int cipolla(int n) {
    if (qmi(n, (mod - 1) >> 1).x != 1) {
        return -1;
    }
    int a = rand() % mod;
    while (!a || qmi((1ll * a * a - n + mod) % mod, (mod - 1) >> 1).x == 1) {
        a = rand() % mod;
    }
    W = (1ll * a * a - n + mod) % mod;

    int x = qmi(cp(a, 1), (mod + 1) >> 1).x;
    if (x > mod - x) x = mod - x;
    return x;
}
}  // namespace Cipolla
Poly Sqrt(Poly &A, int deg) {
    if (deg == 1) {
        return {Cipolla::cipolla(A[0])};  // A[0]=1修改这里即可
    }

    Poly f0 = Sqrt(A, (deg + 1) >> 1);
    f0.resize(deg);
    Poly inf0 = Inv(f0, deg);
    Poly temp(A.begin(), A.begin() + deg);
    temp = mul(inf0, temp);
    temp.resize(deg);
    for (int i = 0; i < deg; i++) f0[i] = (f0[i] + temp[i]) * inv[2] % mod;

    return f0;
}
Poly Deriv(Poly f)  // 求导
{
    for (int i = 0; i < f.size(); i++) f[i] = f[i + 1] * (i + 1) % mod;
    f.pop_back();
    return f;
}
Poly Integ(Poly f)  // 积分
{
    f.push_back(0);
    for (int i = f.size() - 1; i >= 1; i--) f[i] = f[i - 1] * inv[i] % mod;
    f[0] = 0;
    return f;
}
Poly Ln(Poly f, int deg)  // f[0]=1
{
```

```
    f = mul(Deriv(f), Inv(f, deg));
    f.resize(deg - 1);
    return Integ(f);
}
Poly Exp(Poly &A, int deg)  // A[0]=0，记得至少开deg大小
{
    if (deg == 1) return {1};
    Poly f0 = Exp(A, (deg + 1) >> 1);
    Poly temp = f0;
    temp.resize(deg);
    temp = Ln(temp, deg);
    for (int i = 0; i < deg; i++) temp[i] = (A[i] - temp[i] + mod) % mod;
    temp[0] = (temp[0] + 1) % mod;
    temp = mul(f0, temp);
    temp.resize(deg);
    return temp;
}
Poly qmi_ntt(Poly A, int k, int deg = -1)  // A(x)^k且A[0]=1
{
    if (deg == -1) deg = (A.size() - 1) * k + 1;
    A.resize(deg);
    A = Ln(A, deg);
    A = A * k;
    A = Exp(A, deg);
    return A;
}
Poly qmi_ntt(Poly A, string s, int deg = -1)  // A(x)^k
{
    ll k = 0;   // mod (p)
    ll kk = 0;  // mod (p-1),即phi(p)
    bool ty = false;
    for (int i = 0; i < s.size(); i++) {
        k = (1ll * k * 10 + (s[i] - '0'));
        if (k >= mod) k %= mod, ty = true;
        kk = (1ll * kk * 10 + (s[i] - '0')) % (mod - 1);
    }
    if (deg == -1) deg = (A.size() - 1) * k + 1;
    A.resize(deg);
    int n = A.size();
    int t = 0;
    while (t < n && !A[t]) t++;

    if (t == n || ty && t >= 1) {
        return Poly(n, 0);
    }
    ll temp = qmi(A[t], mod - 2, mod);
```

```
    Poly B;
    for (int i = t; i < A.size(); i++) B.push_back(A[i] * temp % mod);
    B = Ln(B, B.size());
    B = B * k;
    B = Exp(B, B.size());
    B.resize(n);
    Poly res(n);
    temp = qmi(A[t], kk, mod);
    for (ll i = 1ll * t * k; i < n; i++)
        res[i] = B[i - 1ll * t * k] * temp % mod;
    return res;
}
Poly psqmi_ntt(Poly A, int k) {
    Poly res(1, 1);
    while (k) {
        if (k & 1) res = res * A;
        A = A * A;
        k >>= 1;
    }
    return res;
}
}  // namespace NTT
using namespace NTT;

int main() {
    /* P1919 A*B
    string a , b ;
    cin >> a >> b ;
    int n = a.size() , m = b.size() ;
    Poly A(n) , B(m) ;
    for(int i = 0 ; i < n ; i ++) A[i] = a[n - i - 1] - '0' ;
    for(int i = 0 ; i < m ; i ++) B[i] = b[m - i - 1] - '0' ;
    Poly c = A * B ;
    for(int i = 0 ; i < c.size() ; i ++){
            if(c[i] >= 10) c[i + 1] += c[i] / 10 , c[i] %= 10 ;
    }
    while(c.size() > 1 && c.back() == 0) c.pop_back() ;
    for(int i = c.size() - 1 ; i >= 0 ; i --)
            write(c[i]) ;
    */
/*
input:
    83517934
    327830610
output:
    27379735249159740
```

```
*/

    /* P4238 【模板】多项式乘法逆
            int n = read() ;
            Poly A(n) ;
            for(int i = 0 ; i < n ; i ++) A[i] = read() ;
            Poly B = Inv(A , n) ;
            for(int i = 0 ; i < n ; i ++) write(B[i] , ' ') ;
    */

/*
input:
    5
    1 6 3 4 9
output:
    1 998244347 33 998244169 1020
*/

    /* P5205 【模板】多项式开根
            int n = read() ;
            Poly A(n) ;
            for(int i = 0 ; i < n ; i ++) A[i] = read() ;
            Poly B = Sqrt(A , n) ;
            for(int i = 0 ; i < n ; i ++) write(B[i] , ' ') ;
    */
/*
input:
7
1 8596489 489489 4894 1564 489 35789489
output:
1 503420421 924499237 13354513 217017417 707895465 411020414
*/

    /* P4725 【模板】多项式对数函数（多项式 ln）
            int n = read() ;
            Poly A(n) ;
            for(int i = 0 ; i < n ; i ++) A[i] = read() ;
            Poly B = Ln(A , n) ;
            for(int i = 0 ; i < n ; i ++) write(B[i] , ' ') ;
    */
/*
input:
6
1 927384623 878326372 3882 273455637 998233543
output:
0 927384623 817976920 427326948 149643566 610586717
```

```
*/

    /* P4726 【模板】多项式指数函数（多项式 exp）
            int n = read() ;
            Poly A(n) ;
            for(int i = 0 ; i < n ; i ++) A[i] = read() ;
            Poly B = Exp(A , n) ;
            for(int i = 0 ; i < n ; i ++) write(B[i] , ' ') ;
    */
/*
input:
6
0 927384623 817976920 427326948 149643566 610586717
output:
1 927384623 878326372 3882 273455637 998233543
*/

    /* P5245 【模板】多项式快速幂
            int n = read() ;
            string k ;  cin >> k ;
            Poly A(n) ;
            for(int i = 0 ; i < n ; i ++) A[i] = read() ;
            Poly B = qmi_ntt(A , k , n) ;
            for(int i = 0 ; i < n ; i ++) write(B[i] , ' ') ;
    */
}
/*
input:
9 18948465
1 2 3 4 5 6 7 8 9
output:
1 37896930 597086012 720637306 161940419 360472177 560327751 446560856 524295016
*/
```

FFT模板

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

#define rep(i, a, b) for (int i = a; i < (int)b; i++)
#define mem(a, b) memset(a, b, sizeof(a))
typedef long long ll;

const int maxn = (1 << 17) + 9;
const long double pi = acos(-1.0L);
using C = complex<long double>;
void fft(C a[], int n, int ty) {
    static C w[maxn * 4];
    w[0].real(1);
    for (int i = 0, j = 0; i < n; i++) {
        if (i > j) swap(a[i], a[j]);
        for (int l = n / 2; (j ^= l) < l; l /= 2)
            ;
    }
    for (int i = 1; i < n; i *= 2) {
        C wn(cos(pi / i), ty * sin(pi / i));
        for (int j = (i - 2) / 2; ~j; j--)
            w[j * 2 + 1] = (w[j * 2] = w[j]) * wn;
        for (int j = 0; j < n; j += i * 2)
            for (int k = j; k < j + i; k++) {
                C x = a[k], y = a[k + i] * w[k - j];
                a[k] = x + y, a[k + i] = x - y;
            }
    }
    if (ty != 1)
        for (int i = 0; i < n; i++) a[i] /= n;
}
template <class ty>
void operator*=(vector<ty>& a, vector<ty>& b) {
    static C c[maxn * 4], d[maxn * 4];
    int n = a.size(), m = b.size(), len = 1 << int(ceil(log2(n + m)));
    rep(i, 0, n) c[i] = a[i];
    fill(c + n, c + len, C());
    rep(i, 0, m) d[i] = b[i];
    fill(d + m, d + len, C());
    fft(c, len, 1), fft(d, len, 1);
    rep(i, 0, len) c[i] *= d[i];
    fft(c, len, -1);
    a.resize(len = n + m - 1);
    rep(i, 0, len) a[i] = c[i].real() + 0.5;
}
```