

Contents

1 Misc	1
1.1 Makefile	1
2 Math	1
2.1 Number Theory	1
2.1.1 Modular	1
2.1.2 Extended GCD	1
2.1.3 Chinese Remainder	1
2.1.4 Tonelli-Shanks	1
3 Numeric	1
3.1 long long Multiplication	1
3.2 Barrett Reduction	1
3.3 FFT	1

1 Misc

1.1 Makefile

.PRECIOUS: ./p%

```
%: p%
    ulimit -s unlimited && ./p%<
```

```
p%: p%.cpp
    g++ -std=gnu++17 -Wall -Wextra -Wshadow \
    -fsanitize=address -fsanitize=undefined \
    -o $@ $<
```

```
init:
    for i in a b c d e f g h; do \
        cp default.cpp "p$$i.cpp"; \
    done
```

2 Math

2.1 Number Theory

2.1.1 Modular

```
template<typename T> struct M {
    static T MOD;
    T v;
    M() : v(0) {}
    M(T x) {
        v = (-MOD <= x && x < MOD) ? x : x % MOD;
        if (v < 0) v += MOD;
    }
    explicit operator T() const { return v; }
    bool operator==(const M& b) const { return v == b.v; }
    bool operator!=(const M& b) const { return v != b.v; }
    M operator-() { return M(-v); }
    M operator+(M b) { return M(v + b.v); }
    M operator-(M b) { return M(v - b.v); }
    M operator*(M b) { return M((__int128)v * b.v % MOD); }
    M operator/(M b) { return *this * (b ^ (MOD - 2)); }
    friend M operator^(M a, ll b) {
        M ans(1);
        for (; b >= 1, a *= a; if (b & 1) ans *= a;
        return ans;
    }
    friend M& operator+=(M& a, M b) { return a = a + b; }
    friend M& operator-=(M& a, M b) { return a = a - b; }
    friend M& operator*=(M& a, M b) { return a = a * b; }
    friend M& operator/=(M& a, M b) { return a = a / b; }
};
using Mod = M<ll>;
template<> ll Mod::MOD = 1000000007;
```

```
/* Safe primes
 * 21673, 26497, 22621, 21817, 28393, 26821, 30181, 22093
 * 977680993, 971939533, 970479637, 910870273, 1041012121
 * 741266610070171837, 1110995545625882557
 * NTT prime | p - 1 | primitive root
 * 65537 | (2^16) | 3
 * 998244353 | (2^23)*119 | 3
 * 2748779069441 | (2^39)*5 | 3
 * 1945555039024054273 | (2^56)*27 | 5 */
```

2.1.2 Extended GCD

```
tuple<ll, ll, ll> extgcd(ll a, ll b) {
    if (b == 0) return { 1, 0, a };
    else {
        auto [p, q, g] = extgcd(b, a % b);
        return { q, p - q * (a / b), g };
    }
}
```

2.1.3 Chinese Remainder

```
ll crt(ll a, ll m, ll b, ll n) {
    if (n > m) swap(a, b), swap(m, n);
    auto [x, y, g] = extgcd(m, n);
    assert((a - b) % g == 0); // no solution
    x = ((b - a) / g * x) % (n / g) * m + a;
    return x < 0 ? x + m / g * n : x;
}
```

2.1.4 Tonelli-Shanks

```
int legendre(Mod a) {
    if (a == 0) return 0;
    return (a ^ ((a.MOD - 1) / 2)) == 1 ? 1 : -1;
}
Mod sqrt(Mod a) {
    assert(legendre(a) != -1); // no solution
    ll p = a.MOD, s = p - 1;
    if (a == 0) return 0;
    if (p == 2) return 1;
    if (p % 4 == 3) return a ^ ((p + 1) / 4);
    int r, m;
    for (r = 0; !(s & 1); r++) s >>= 1;
    Mod n = 2;
    while (legendre(n) != -1) n += 1;
    Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
    while (b != 1) {
        Mod t = b;
        for (m = 0; t != 1; m++) t *= t;
        Mod gs = g ^ (1LL << (r - m - 1));
        g = gs * gs, x *= gs, b *= g, r = m;
    }
    return x;
}
```

3 Numeric

3.1 long long Multiplication

```
using ull = unsigned long long;
using ll = long long;
using ld = long double;
// returns a * b % M
ull mult(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
```

3.2 Barrett Reduction

```
using ull = unsigned long long;
using uL = __uint128_t;
```

```
// very fast calculation of a % m
struct reduction {
    const ull m, d;
    reduction(ull m) : m(m), d(((uL)1 << 64) / m) {}
    inline ull operator()(ull a) const {
        ull q = (ull)(((uL)d * a) >> 64);
        return (a -= q * m) >= m ? a - m : a;
    }
};
```

3.3 FFT

```
template<typename T>
void work(int n, vector<T>& a, vector<T>& rt, bool inv) {
    for (int i = 1, r = 0; i < n; i++) {
        for (int bit = n; !(r & bit); bit >>= 1, r ^= bit);
        if (r > i) swap(a[i], a[r]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < len / 2; j++) {
                int pos = n / len * (inv ? len - j : j);
                T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
                a[i + j] = u + v, a[i + j + len / 2] = u - v;
            }
        }
    }
    if (inv) {
        T minv = T(1) / T(n);
        for (T& x : a) x *= minv;
    }
}

void FFT(vector<complex<double>>& a, bool inv) {
    int n = a.size();
    vector<complex<double>> rt(n + 1);
    double arg = acos(-1) * 2 / n;
    for (int i = 0; i <= n; i++)
        rt[i] = { cos(arg * i), sin(arg * i) };
    work(n, a, rt, inv);
}

void NTT(vector<Mod>& a, bool inv, Mod p_root) {
    int n = a.size();
    Mod root = p_root ^ (p_root.MOD - 1) / n;
    vector<Mod> rt(n + 1, 1);
    for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
    work(n, a, rt, inv);
}
```