Team notebook

January 13, 2020

Contents

1 FFT

```
// computes f  (k) = sum(a[x] exp(2 i  kx/N )) for all k.
// Useful for convolution: conv(a, b) = c, where c[x] =
   sum(a[i]b[x  i])
// convolution of complex numbers or more than two vectors:
   FFT, multiply
// pointwise, divide by n, reverse(start+1, end), FFT back.
   Rounding is safe if
// Otherwise, use long doubles/NTT/FFTMod.

#define IOS                             \
        ios_base::sync_with_stdio(false); \
        cin.tie(0);                     \
        cout.tie(0);                    \
        cin.exceptions(cin.failbit);

#define trav(a, x) for (auto &a : x)
#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C> &a)
{
        int n = sz(a), L = 31 - __builtin_clz(n);
        static vector<complex<long double>> R(2, 1);
        static vector<C> rt(2, 1);
        for (static int k = 2; k < n; k *= 2)
        {
                R.resize(n);
                rt.resize(n);
                auto x = polar(1.0L, M_PIl / k);
                rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i /
                    2] * x : R[i / 2];
        }
        vi rev(n);
        rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) /
            2;
        rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int k = 1; k < n; k *= 2)
```

```
			for (int i = 0; i < n; i += 2 * k)
				rep(j, 0, k)
				{
					C z = rt[j + k] * a[i + j + k];

					a[i + j + k] = a[i + j] - z;
					a[i + j] += z;
				}
}
vd conv(const vd &a, const vd &b)
{
	if (a.empty() || b.empty())
		return {};
	vd res(sz(a) + sz(b) - 1);
	int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
	vector<C> in(n), out(n);
	copy(all(a), begin(in));
	rep(i, 0, sz(b)) in[i].imag(b[i]);
	fft(in);
	trav(x, in) x *= x;
	rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
	fft(out);
	rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
	return res;
}

// CP ALGO

using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> &a, bool invert)
{
	int n = a.size();

	for (int i = 1, j = 0; i < n; i++)
	{
		int bit = n >> 1;
		for (; j & bit; bit >>= 1)
			j ^= bit;
		j ^= bit;

		if (i < j)
			swap(a[i], a[j]);
	}

	for (int len = 2; len <= n; len <<= 1)
	{
		double ang = 2 * PI / len * (invert ? -1 : 1);
		cd wlen(cos(ang), sin(ang));
		for (int i = 0; i < n; i += len)
		{
			cd w(1);
			for (int j = 0; j < len / 2; j++)
			{
				cd u = a[i + j], v = a[i + j +
				  len / 2] * w;
				a[i + j] = u + v;
				a[i + j + len / 2] = u - v;
				w *= wlen;
			}
		}
	}

	if (invert)
	{
		for (cd &x : a)
			x /= n;
	}
}
vector<int> multiply(vector<int> &a, vector<int> &b)
{
```

```
	vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
	    b.end());
	int n = 1;
	while (n < a.size() + b.size())
		n <<= 1;
	fa.resize(n);
	fb.resize(n);

	fft(fa, false);
	fft(fb, false);
	for (int i = 0; i < n; i++)
		fa[i] *= fb[i];
	fft(fa, true);

	vector<int> result(n);
	for (int i = 0; i < n; i++)
		result[i] = round(fa[i].real());
	return result;
}

// Description: Can be used for convolutions modulo specific
    nice primes of
// the form 2^a b + 1, where the convolution result has size
    at most 2 a . Inputs
// must be in [0, mod).
// Time: O (N log N )

const ll mod = (119 << 23) + 1, root = 62; // = 998244353
ll modpow(ll n, ll x)
{
	if (x == 0)
		return 1;
	ll z = modpow(n, x / 2);
	z *= z;
	z %= mod;
	if (x % 2)
		z *= n;
	z %= mod;
	return z;
}
// For p < 230 there i s also e . g . 5 << 25, 7 << 26, 479
    << 21
// and 483 << 21 (same root ) . The l a s t two are > 109.
typedef vector<ll> vl;
void ntt(vl &a, vl &rt, vl &rev, int n)
{
	rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
	for (int k = 1; k < n; k *= 2)
		for (int i = 0; i < n; i += 2 * k)
			rep(j, 0, k)
			{
				ll z = rt[j + k] * a[i + j + k]
				    % mod, &ai = a[i + j];
				a[i + j + k] = (z > ai ? ai - z
				    + mod : ai - z);
				ai += (ai + z >= mod ? z - mod
				    : z);
			}
}
vl conv(vl &a, vl &b)
{
	if (a.empty() || b.empty())
		return {};
	int s = sz(a) + sz(b) - 1, B = 32 -
	    __builtin_clz(s), n = 1 << B;
	vl L(a), R(b), out(n), rt(n, 1), rev(n);
	L.resize(n), R.resize(n);
	rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << B) /
	    2;
	ll curL = mod / 2, inv = modpow(n, mod - 2);
	for (int k = 2; k < n; k *= 2)
	{
```

```
            ll z[] = {1, modpow(root, curL /= 2)};
            rep(i, k, 2 * k) rt[i] = rt[i / 2] * z[i & 1]
                % mod;
        }
        ntt(L, rt, rev, n);
        ntt(R, rt, rev, n);
        rep(i, 0, n) out[-i & (n - 1)] = L[i] * R[i] % mod *
            inv % mod;
        ntt(out, rt, rev, n);
        return {out.begin(), out.begin() + s};
}

// Transform to a basis with fast convolutions of the form
    c[z] = a[x]  b[y], where  is one of AND, OR, XOR. The
    size
// of a must be a power of two.

void FST(vi &a, bool inv)
{
        for (int n = sz(a), step = 1; step < n; step *= 2)
        {
                for (int i = 0; i < n; i += 2 * step)
                        rep(j, i, i + step)
                        {
                                int &u = a[j], &v = a[j + step];
                                tie(u, v) =
                                    inv ? pii(v - u, u) :
                                        pii(v, u + v); // AND
                                inv ? pii(v, u - v) : pii(u +
                                    v, u); // OR
                                pii(u + v, u - v);
                                // XOR
                        }
        }
        if (inv)
                trav(x, a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b)
{
        FST(a, 0);
        FST(b, 0);
        rep(i, 0, sz(a)) a[i] *= b[i];
        FST(a, 1);
        return a;
}

// CRT.h
// Description: Chinese Remainder Theorem.
// crt(a, m, b, n) computes x such that x  a (mod m), x  b
    (mod n). If
// |a| < m and |b| < n, x will obey 0  x < lcm(m, n).
    Assumes mn < 2^62 .
// Time: log(n)

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