Implementation of NTT

https://youtu.be/7Sjn5FxroWA

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

1. HAWK 소개

HAWK 소개는 세미나
https://youtu.be/dX0X1oMu7YE

```
ng_hawk Hawk keygen ftimer print
ftimer_ntt = 0.000273
fs->rng = 0.000021
fs->Hawk_regen_fg_t = 0.000808
fs->mp_mkgmigm_t = 0.000015
fs->mp_add_t = 0.000300
fs->mp_montymul = 0.000340
fs->mp_norm = 0.000300
fs->mp_mkgm_t = 0.000013
fs->vect_FFT_t = 0.000307
fs->solve_NTRU_t = 0.057962
```

```
ng_ntru Hawk keygen FTIMER_ntru print
solve_NTRU_deepest = 0.007041
solve_NTRU_intermediate = 0.049215
solve_NTRU_depth0 = 0.001656
poly_mp_set_small_t = 0.000000
mp_mkgm_t = 0.000000
mp_NTT_t = 0.000000
mp_mkigm_t = 0.000000
zint_rebuild_CRT_t = 0.000000
```

```c
/*
 * Memory layout: we keep Ft, Gt, ft and gt; we append_ntru:
 *   gm     NTT support (n)
 *   igm    iNTT support (n)
 *   fx     temporary f mod p (NTT) (n)
 *   gx     temporary g mod p (NTT) (n)
 */
uint32_t *gm = t1;
uint32_t *igm = gm + n;
uint32_t *fx = igm + n;
uint32_t *gx = fx + n;
mp_mkgmigm(logn, gm, igm, PRIMES[u].g, PRIMES[u].ig, p, p0i);
if (u < slen) {
    memcpy(fx, ft + u * n, n * sizeof *fx);
    memcpy(gx, gt + u * n, n * sizeof *gx);
    mp_iNTT(logn, ft + u * n, igm, p, p0i);
    mp_iNTT(logn, gt + u * n, igm, p, p0i);
} else {
    uint32_t Rx = mp_Rx31((unsigned)slen, p, p0i, R2);
    for (size_t v = 0; v < n; v ++) {
        fx[v] = zint_mod_small_signed(ft + v, slen, n,
            p, p0i, R2, Rx);
        gx[v] = zint_mod_small_signed(gt + v, slen, n,
            p, p0i, R2, Rx);
    }
    mp_NTT(logn, fx, gm, p, p0i);
    mp_NTT(logn, gx, gm, p, p0i);
}

/*
 * We have (F,G) from deeper level in Ft and Gt, in
 * RNS. We apply the NTT modulo p.
 */
uint32_t *Fe = Ft + u * n;
uint32_t *Ge = Gt + u * n;
mp_NTT(logn - 1, Fe + hn, gm, p, p0i);
mp_NTT(logn - 1, Ge + hn, gm, p, p0i);
```

```c
uint32_t *Fe = Ft + u * n;
uint32_t *Ge = Gt + u * n;
mp_NTT(logn - 1, Fe + hn, gm, p, p0i);
mp_NTT(logn - 1, Ge + hn, gm, p, p0i);

/*
 * Compute F and G (unreduced) modulo p.
 */
for (size_t v = 0; v < hn; v ++) {
    uint32_t fa = fx[(v << 1) + 0];
    uint32_t fb = fx[(v << 1) + 1];
    uint32_t ga = gx[(v << 1) + 0];
    uint32_t gb = gx[(v << 1) + 1];
    uint32_t mFp = mp_montymul(Fe[v + hn], R2, p, p0i);
    uint32_t mGp = mp_montymul(Ge[v + hn], R2, p, p0i);
    Fe[(v << 1) + 0] = mp_montymul(gb, mFp, p, p0i);
    Fe[(v << 1) + 1] = mp_montymul(ga, mFp, p, p0i);
    Ge[(v << 1) + 0] = mp_montymul(fb, mGp, p, p0i);
    Ge[(v << 1) + 1] = mp_montymul(fa, mGp, p, p0i);
}

/*
 * We want the new (F,G) in RNS only (no NTT).
 */
mp_iNTT(logn, Fe, igm, p, p0i);
mp_iNTT(logn, Ge, igm, p, p0i);
```

키 생성 과정 중에 solve_NTRU에서 많은 시간이 소요되는 것을 확인하고, 해당 연산을 확인했을 때, NTT가 많이 사용되었고, **NTT를 병렬 구현함으로써 최적화 구현**

2. NTT

- Number Theoretic Transform은 DFT의 일종으로 정수 연산을 수행하기 위해 설계됨.
  - 복소수 대신 정수 모듈러 연산을 사용
  - 원시 단위근 ω_n - 차수가 n만큼의 주기성을 가지는 원소
    - 원시 단위근이 되려면 아래 조건을 만족해야함.
    - $g^k \equiv 1 \, mod \, p$ ($\mathrm{k} = 1, 2, \dots, \mathrm{p} - 1$) 이때, 모든 값은 서로 다른 값을 가져야함
    - $g^{p-1} \equiv 1 \, mod \, p$ 이걸 만족하며, 이때 p-1은 오일러 피 함수의 값
    - $g^{p-1/n} \, mod \, p$ 차수 n에 대한 원시 단위근

$$f_k = \sum_{i \in [0,n)} x_i \cdot \omega_n^{i \cdot k} \qquad \omega_n = e^{\frac{2 \cdot \pi \cdot i}{n}}$$

where ω_n is a primitive n-th root of unity.

$$\omega_n = g^{\frac{p-1}{n}}$$

- DFT 대신 NTT를 사용하는 이유 (틀릴 수도 있음)
  - DFT는 복소수를 사용 -〉 실수랑 허수를 나누어서 연산해야함
  - 컴퓨터에서 부동소수점 연산은 오차가 있을 수 있음 -〉 누적되면 정확도가 떨어짐

2. NTT

- 원시 단위근 원시 단위근 ω_n 예
  - p=7 g=3

$$g^k \mod p \ \ (k = 1, 2, \ldots, p-1)$$
$$g^{p-1} \equiv 1 \mod p$$

$$3^1 \mod 7 = 3 \mod 7 = 3$$
$$3^2 \mod 7 = 9 \mod 7 = (7 * 1) + 2 \mod 7 = 2$$
$$3^3 \mod 7 = 27 \mod 7 = (7 * 3) + 6 \mod 7 = 6$$
$$3^4 \mod 7 = 81 \mod 7 = (7 * 11) + 4 \mod 7 = 4$$
$$3^5 \mod 7 = 243 \mod 7 = (7 * 34) + 5 \mod 7 = 5$$
$$3^6 \mod 7 = 729 \mod 7 = (7 * 104) + 1 \mod 7 = 1$$

$a(x) = 6x^5 + 5x^4 + 4x^3 + 3x^2 + 2x + 1$ n = 6 $\omega_n = g^{\frac{p-1}{n}}$ $\omega_6 = 3^{\frac{6}{6}} = 3$

2. NTT

$$p = 5 \,, \omega = 2$$
$$a(x) \ = \ 4x^3 + 3x^2 + 2x + 1$$

$$\omega^0 \ mod \ p = 2^0 \ mod \ 5 = 1$$
$$\omega^1 \ mod \ p = 2^1 \ mod \ 5 = 2$$
$$\omega^2 \ mod \ p = 2^2 \ mod \ 5 = 4$$
$$\omega^3 \ mod \ p = 2^3 \ mod \ 5 = 3$$

$$f_k = \sum_{i \in [0,n)} x_i \cdot \omega_n^{i \cdot k}$$

$$f_0 = x_0 \cdot \omega^0 + x_1 \cdot \omega^0 + x_2 \cdot \omega^0 + x_3 \cdot \omega^0$$
$$f_1 = x_0 \cdot \omega^0 + x_1 \cdot \omega^1 + x_2 \cdot \omega^2 + x_3 \cdot \omega^3$$
$$f_2 = x_0 \cdot \omega^0 + x_1 \cdot \omega^2 + x_2 \cdot \omega^4 + x_3 \cdot \omega^6$$
$$f_3 = x_0 \cdot \omega^0 + x_1 \cdot \omega^3 + x_2 \cdot \omega^6 + x_3 \cdot \omega^9$$

$$\mathrm{a}(0) = (1 \cdot 1 + 2 \cdot 1 + 3 \cdot 1 + 4 \cdot 1) \ mod \ 5 = 0$$
$$\mathrm{a}(1) = (1 \cdot 1 + 2 \cdot 2 + 3 \cdot 4 + 4 \cdot 3) \ mod \ 5 = 4$$
$$\mathrm{a}(2) = (1 \cdot 1 + 2 \cdot 4 + 3 \cdot 1 + 4 \cdot 4) \ mod \ 5 = 3$$
$$\mathrm{a}(3) = (1 \cdot 1 + 2 \cdot 3 + 3 \cdot 4 + 4 \cdot 2) \ mod \ 5 = 2$$

[0, 4, 3, 2]

$$a'(x) = NTT\big(a(x)\big) -> [0,4,3,2]$$
$$b'(x) = NTT\big(b(x)\big) -> [0,4,3,2]$$
$$a(x) * b(x) = INTT(a'(x) * b'(x))$$

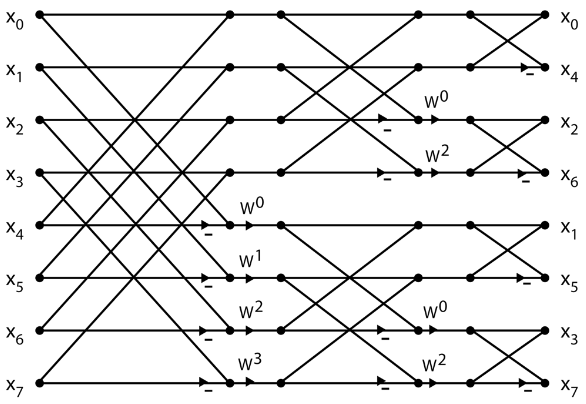
[0, 4, 3, 2]
[0, 4, 3, 2] 각 행끼리 곱
[0, 16, 9, 4] mod 5 -> [0, 1, 4, 4]
[0, 1, 4, 4] 를 INTT 하면
최종 $a(x) * b(x)$ 결과

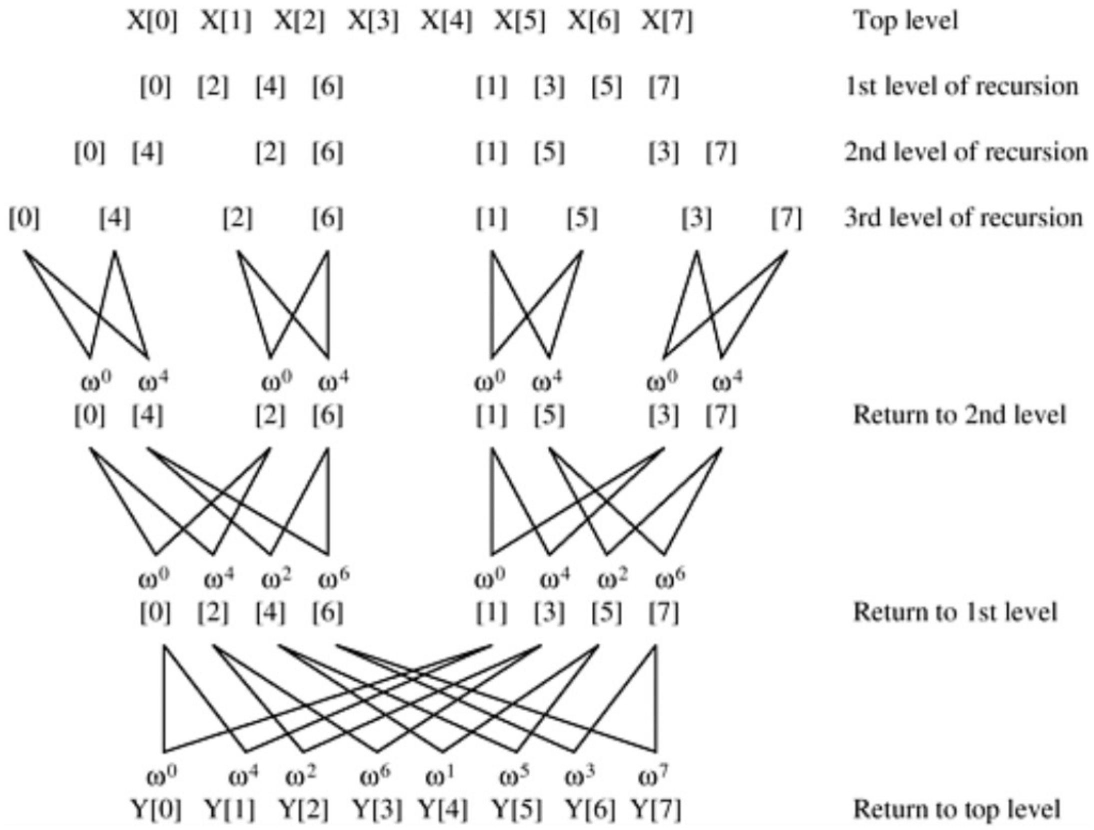
예시이기 때문에 위 값은 틀림.

2. NTT

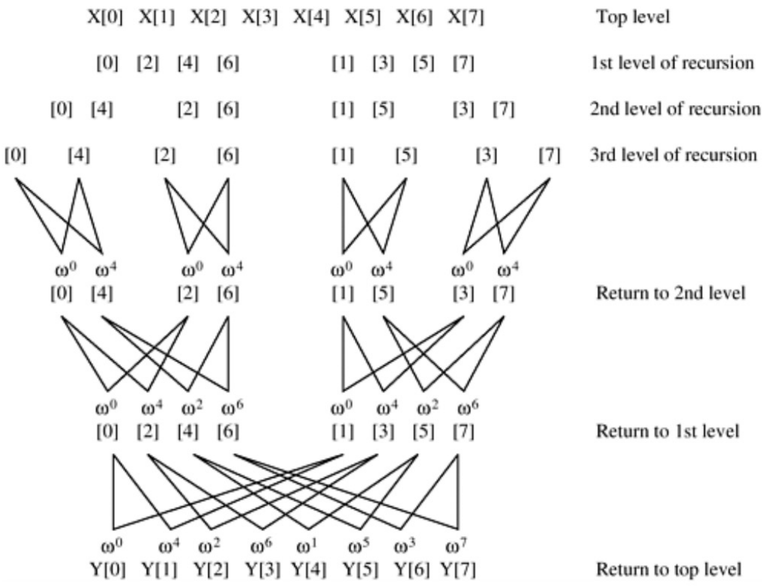
- Cooley–Tukey 방식

$$
\begin{aligned}
X_k &= E_k + e^{-\frac{2\pi i}{N}k} O_k \\
X_{k+\frac{N}{2}} &= E_k - e^{-\frac{2\pi i}{N}k} O_k
\end{aligned}
$$





3. HAWK NTT 최적화 구현




```
k1 = 0 / k2 = 4
k1 = 1 / k2 = 5
k1 = 2 / k2 = 6
k1 = 3 / k2 = 7

k1 = 0 / k2 = 2
k1 = 1 / k2 = 3

k1 = 4 / k2 = 6
k1 = 5 / k2 = 7

k1 = 0 / k2 = 1

k1 = 2 / k2 = 3

k1 = 4 / k2 = 5

k1 = 6 / k2 = 7
```

```c
void
mp_NTT(unsigned logn, uint32_t *restrict a, const uint32_t *restrict gm,
    uint32_t p, uint32_t p0i)
{
    if (logn == 0) {
        return;
    }
    size_t t = (size_t)1 << logn;
    for (unsigned lm = 0; lm < logn; lm ++) {
        size_t m = (size_t)1 << lm;
        size_t ht = t >> 1;
        size_t v0 = 0;
        for (size_t u = 0; u < m; u ++) {
            uint32_t s = gm[u + m];
            for (size_t v = 0; v < ht; v ++) {
                size_t k1 = v0 + v;
                size_t k2 = k1 + ht;
                uint32_t x1 = a[k1];
                uint32_t x2 = mp_montymul(a[k2], s, p, p0i);
                a[k1] = mp_add(x1, x2, p);
                a[k2] = mp_sub(x1, x2, p);
            }
            v0 += t;
        }
        t = ht;
    }
}
```

감 사 합 니 다