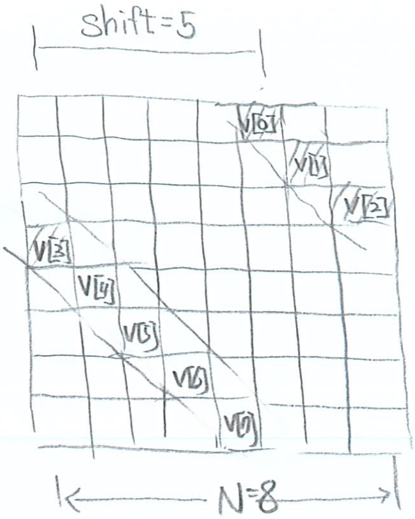
SparseMatrix



$$A[i][j] = \begin{cases} V[i] & \text{if } j-i \equiv \text{shift} \pmod N \\ 0 & \text{else} \end{cases}$$

or

$$V[i] = A[i][\mathrm{mod}(i+\text{shift}, N)]$$

```
template <int N, int NumDiagsA, int NumDigsB>
void matmul( const SparseComplexMatrix<N, NumDiagsA>& A,
             const SparseComplexMatrix<N, NumDiagsB>& B,
                   SparseComplexMatrix <N, NumDiagsA * NumDiagsB> & C ){

    for( int iA=0; iA< NumDiagsA; iA++)
    for( int iB=0; iB < NumDiagsB; iB++){
        int shiftA = A.shift [iA];
        int shiftB = A. shift [iB];

        int iC = iA* NumDigsB + iB;
        C.shift[iC]=(shiftA+shiftB)%N;
        for(int i=0; i<N; i++){
            int k=(i+shiftA)%N;
            C.diagr[iC][i] = A.diagr[iA][i]* B.diagr[iB][k]
                           - A.diagi[iA][i]* B.diagi[iB][k];
            C.diagi[iC][i] = A.diagr[iA][i]* B.diagi[iB][k]
                           + A.diagi[iA][i]* B.diagr[iB][k];
        }
    }

}.
```

$$C[i][j] = \sum_k A[i][k]\, B[k][j]$$

$$= A[i][k] \cdot B[k][j] \quad (k = i + \text{shift}_A)$$

$$= \begin{cases} V_A[i] \cdot V_B[k], & \text{if } j = k + \text{shift}_B \text{ or } \boxed{j = i + \text{shift}_A + \text{shift}_B} \\ 0 & \end{cases}$$

$C[i][j] = C.\mathrm{diag}[i]$
$A[i][k] = A.\mathrm{diag}[i]$
$B[k][j] = B.\mathrm{diag}[k]$

Conjugation: $z \in \mathbb{C}^{\frac{N}{2}} \xrightarrow{\quad\Delta\quad} pt \in R_Q$.

$$z[j] = \frac{pt}{\Delta}(\zeta) = \sum_{i=0}^{N-1} \frac{pt[i]}{\Delta} \cdot (\zeta^i) \quad , \quad \zeta : \text{a root of unity in } \mathbb{C}$$

$$\overline{z[j]} = \sum_{i=0}^{N-1} \frac{pt[i]}{\Delta} (\bar{\zeta}^i) = \frac{pt}{\Delta}(\bar{\zeta}) \quad , \quad \zeta \cdot \bar{\zeta} = 1, \quad \bar{\zeta} = \frac{1}{\zeta}$$

$$\therefore \ \bar{z} \in \mathbb{C}^{\frac{N}{2}} \xrightarrow{\quad\Delta\quad} pt^{(conj)} \in R_Q, \quad \text{where } pt^{(conj)}(x) = pt\left(\frac{1}{x}\right)$$

$$pt(x) = \sum_{i=0}^{N-1} pt[i]\, x^i$$

$$pt^{(conj)}(x) = \sum_{i=0}^{N-1} pt[i] \frac{1}{x^i} = -\sum_{i=0}^{N-1} pt[i] \frac{x^N}{x^i} \qquad x^N = -1$$

$$= \sum_{i=0}^{N-1} \underbrace{-pt[i]}_{\substack{\| \\ pt^{(conj)}[N-i]}} x^{N-i}$$

```
template<int N>
void conj( const int s[N], int s^(conj)[N] ){
    s^(conj)[0] = s[0];
    for(int i=1; i<N; i++)
        s^(conj)[N-i] = - s[i];
}
```

$s^{(conj)}[N-i] = -s[i]$

```
template<int N, int L>
void conj( const uint64_t q[L],
           const uint64_t a[L][N],
                 uint64_t a^(conj)[L][N] ){
    for(int i=0; i<L; i++){
        a^(conj)[i][0] = a[i][0];
        for(int j=1; j<N; j++)
            a^(conj)[i][N-j] = (q[i]-a[i][j])%q[i];
    }
}
```

$a^{(conj)}[i][N-j] = -a[i][j]$ in $\mathbb{Z}_{q[i]}$

CoeffToSlot: $z \in \mathbb{C}^{\frac{N}{2}} \xrightarrow{\Delta} pt \in R_Q$

- z and $\frac{pt}{\Delta}$ are related by the DFT matrix U_0.

$$z = U_0\left(\frac{pt^{1st}}{\Delta} + i\,\frac{pt^{2nd}}{\Delta}\right), \qquad pt = [\underbrace{pt^{1st}}_{\text{length } N/2} \mid \underbrace{pt^{2nd}}_{\text{length } N/2}]$$

- Using $\overline{U_0}^T U_0 = \frac{N}{2}\cdot \text{Identity}$,

  let $z_1 = \frac{1}{N}\overline{U_0}^T \times z$.

  Then $z_1 + \overline{z_1} = \frac{pt^{1st}}{\Delta}$ and $-i(z_1 - \overline{z_1}) = \frac{pt^{2nd}}{\Delta}$.

- U_0 is a full matrix and its multiplication involves $\frac{N}{2}$ Hadamard product. $(\frac{N}{2}\times\frac{N}{2})$

- FFT decomposition: $U_0 \cdot \underbrace{R}_{\text{a permutation matrix}} = E_2 E_4 \cdots E_{\frac{N}{2}}$, each E_i is a sparse matrix with 3 diagonals.

- The fastest implementation of CoeffToSlot is to multiply $\overline{E_i}^T$ sequentially, which requires $(3\times(\log N-1))$ Hadamard product. Each matrix multiplication needs to convert its real number elements to ~~an~~ integers, consuming a level. The fastest implementation thus consumes $(\log N-1)$ levels, that is too much.

- In practice, the sparse matrices are combined into 3 or 4, which we call matrix decomposition number, MDNUM.

- $U_0 R = \underbrace{E_2 E_4}_{A[0]} \cdots \underbrace{E_{\frac{N}{2}}}_{A[MDNUM-1]}$, $\quad \overline{U_0}^T = R^T\, \overline{A[MDNUM-1]}^T \times \cdots \times \overline{A[0]}^T$

  (N and MDNUM are assumed to satisfy $MDNUM \mid (\log N - 1)$.)

$z \in \mathbb{C}^{\frac{N}{2}} \xrightarrow{\Delta} pt \in R_Q$ — $ct \in R_Q^2$

$\frac{1}{N}\times\overline{A[MDNUM-1]}^T \cdots \times \overline{A[0]}^T$ | linear transform

$z_1 \xrightarrow{\Delta^{DNUM+1}}$ — $ct_1 \in R_Q^2$

bit-reverse permuted:

$pt^{1st}/\Delta \xrightarrow{\Delta^{DNUM+1}}$ — $ct_1 + \text{conj}(ct_1) \in R_Q^2$

$pt^{2nd}/\Delta \xrightarrow{\Delta^{DNUM+1}}$ — $pt_{-i} * [ct_1 - \text{conj}(ct_1)] \in R_Q^2$

$pt/\Delta \xrightarrow{\Delta}$ — RS — $\text{CoeffToSlot}(ct) \in \left[R_{Q/\Delta^{MDNUM}}\right]^2$

N/2 | MDNUM=3, 3^{MDNUM}

#4

```
void splitU_R_logN_10(SparseComplexMatrix<1<<9, 3*3*3> A[3]){
```

In the case $N=2^{10}$

```
    SparseComplexMatrix<1<<9 3> E[9];
    splitU_R<10>(E);
```

$U_oR = E[0] \times \cdots \times E[8]$

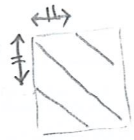
```
    for(int d=0; d<3; d++){
        SparseComplexMatrix<1<<9, 9> temp;
        matmul<1<<9, 3, 3>(E[3*d], E[3*d+1], temp);
        matmul<1<<9, 9, 3>(temp, E[3*d+2], A[d]);
    }
}
```

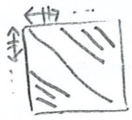
$U_oR = A[0] \cdot A[1] \cdot A[2]$

```
template<int N, int NumDigs>
void mat_conjtranspose(const SparseComplexMatrix<N, NumDigs>& A,
                       SparseComplexMatrix<N, NumDigs>& Ā^T){
    for(int d=0; d<NumDigs; d++){
        Ā^T.shift[d] = A.shift[d];
        for(int i=0; i<N; i++){
            Ā^T.diagr[d][i]=0;
            Ā^T.diagi[d][i]=0;
        }
    }
```

$\bar{A}^T$: same diagonal pattern as A

$\bar{A}^T = 0$

Each E[i] is of shape 

Each A[d] is of shape 

$\overline{A[d]}^T$ and A[d] have the same diagonal shapes.

For each shift[d], there should exist shift[d_] s.t. shift[d_]=(N−shift[d])

```
    for(int d=0; d<NumDigs; d++){
        int d_=0;
        for( ; d_<NumDigs; d_++)
            if(Ā^T.shift[d_] == (N-A.shift[d])%N))
                break;
        assert(d_ != NumDigs);
        int s = A.shift[d];
        for(int i=0; i<N; i++){
            Ā^T.diagr[d_][(i+s)%N] += A.diagr[d][i];
            Ā^T.diagi[d_][(i+s)%N] -= A.diagi[d][i];
        }
    }
}
```

$\bar{A}^T[i+s][i] = \overline{A[i][i+s]}$

$\bar{A}^T.diag[d_][i+s] = \overline{A.diag[d][i]}$

```
template<int L, int DNUM, int K>
void swkgen_logN_10( const uint64_t q[L],
                     const uint64_t P[K],   const int s[1<<10],
                  Complex
    const   SparseMatrix<1<<9, 3*3*3>& A[3],
        uint64_t   evk        [DNUM][2][DNUM*K+K][1<<10],
        uint64_t   ckey       [DNUM][2][    "    ][ " ],
        uint64_t   rkey [3][27][DNUM][2][DNUM*K+K][1<<10] ){

evk   int ss[1<<10];  conv<1<<10>(s, s, ss);
      swkgen<1<<10, L, DNUM>(ss, s, q, P, evk);

ckey  int sconj[1<<10];   conj<1<<10>(s, sconj);
      swkgen<1<<10, L, DNUM>(sconj, s, q, P, ckey);

    for(int d=0; d<3; d++)

rkey      rkey_gen<1<<10, L, DNUM, K, 27>(A[d], q, P, s, rkey[d]);

}
```

```
template<int L, int DNUM, int K>
void CoeffToSlot_logN_10 ( const uint64_t q[L],
                           const uint64_t p[K], uint64_t Delta,
    const SparseComplexMatrix<1<<9, 27> A[3],
    const uint64_t ckey^        [DNUM][2][DNUM*K+K][1<<10],
    const uint64_t rkey^ [3][27][  "  ][" ][   "  ][ " ],
    const uint64_t ct^ [2][L][1<<10],
          uint64_t ct^_CtS[2] [2][L-3][1<<10] ) {const int N= 1<<10;
```

```
SparseComplexMatrix<1<<9, 27> A-^T[3];
for(int n=0; n<3; n++){
    mat_conjtranspose<1<<9, 27>( A[n], A-^T[n]);
    for(int d=0; d<27; d++)
    for(int i=0; i<N/2; i++) {
        A-^T[n].diagr[d][i] /= (n==0)? 16 : 8 ;
        A-^T[n].diagi[d][i] /= (n==0)? 16 : 8 ;
    }
}
```

$$\frac{1}{N}\bar{U}_0^T = \frac{1}{2^{10}} \bar{A}[2]^T \cdot \bar{A}[1]^T \bar{A}[0]^T = \frac{\bar{A}[2]^T}{2^3} \cdot \frac{\bar{A}[1]^T}{2^3} \cdot \frac{\bar{A}[0]^T}{2^4}$$

$$\begin{array}{ccccc} Z & \xrightarrow{\Delta} & pt & - & ct \\ \Big| \frac{1}{N}\bar{U}_0^T & & \Big| & & \Big| \\ \frac{1}{2}\left(\frac{pt^{1st}}{\Delta} + i\,\frac{pt^{2nd}}{\Delta}\right) & \xrightarrow{\Delta^4} & & - & ct_1 \end{array}$$

```
uint64_t ct^_1 [2][L][N];
for(int i=0; i<2; i++)
for(int j=0; j<L; j++)
for(int k=0; k<N; k++) ct^_1[i][j][k] = ct^[i][j][k];
 for(int n=0; n<3; n++){
     uint64_t temp [2][L][N];
     lineartransform<N, 10, L, DNUM, K, 27>( A-^T[n], Delta, q, p,
                                  rkey^[n], ct^_1, temp);

     for(int i=0; i<2; i++)
     for(int j=0; j<L; j++)
     for(int k=0; k<N; k++)
         ct^_1 [i][j][k] = temp [i][j][k];
 }
```

```
uint64_t ĉt2[2][L][N];
{   uint64_t temp[2][L][N];
    intt<N,L>(g, ĉt1[0]);
    intt<N,L>(g, ĉt1[1]);
    conj<N,L>(g, ĉt1[0], temp[0]);
    conj<N,L>(g, ĉt1[1], temp[1]);
    ntt<N,L>(g, ĉt1[0]);    ntt<N,L>(g, temp[0]);
    ntt<N,L>(g, ĉt1[1]);    ntt<N,L>(g, temp[1]);
     RS<N,L,DNUM,K>(g, P, ĉkey, temp, ĉt2);
}
```

$ct_2 = conj(ct_1)$

```
for(int i=0; i<2; i++)
for(int j=0; j<L; j++)
for(int k=0; k<N; k++)
   ĉt1[i][j][k] = (ĉt1[i][j][k] + ĉt2[i][j][k])
                  % q[j];
 RS_hat<N,L,L-3>(ĉt1, ĉt_cts[0]);
```

$ct_{cts}[0] = RS(ct_1 + ct_2)$

```
uint64_t p̂tmi[L][N];
for(int i=0; i<L; i++){
    for(int j=0; j<N; j++)
        p̂tmi[i][j] = 0;
    p̂tmi[i][N/2] = q[i] - 1;
}
ntt<N,L>(g, p̂tmi);
```

$-i \in \mathbb{C}^{\frac{N}{2}}$ 1 —— $pt_{mi} \in \mathbb{Z}_Q$
$\quad = -x^{\frac{N}{2}}$

$ct_{cts}[1] = RS(pt_i * (ct_1 - ct_2))$

```
for(int i=0; i<2; i++)
for(int j=0; j<L; j++)
for(int k=0; k<N; k++)
   ĉt1[i][j][k] = mul_mod(p̂tmi[j][k], (ĉt1[i][j][k] +
                  mul_mod(q[j] - 2, ĉt2[i][j][k], q[j]))%q[j], q[j]);

RS_hat<N,L,L-3>(ĉt1, ĉt_cts[1]);
```

}

```
#define LOGN 10
    ...
void main() {
    double zr[N/2], zi[N/2];
        ...
    uint64_t pt[L][N];
        ...
    uint64_t ĉt[2][L][N];
        ...
```

default code with $N = 2^{10}$

```
SparseComplexMatrix <N/2, 2η> A[3];
splitUoR_logN_10 (A);
```

$U_0 R = A[0] \cdot A[1] \cdot A[2]$

```
uint64_t evk̂          [DNUM][2][DNUM*K+K][N];
uint64_t cKeŷ         [ " ][ " ][DNUM*K+K][N];
uint64_t rKeŷ [3][2η][ " ][ " ][DNUM*K+K][N];
swkgen_logN_10 <L,DNUM,K> (q, P, s, A, evk̂, cKeŷ, rKeŷ);
```

key generation

```
uint64_t ctĉts [2][2][L-3][N];
CoeffToSlot_logN_10 <L,DNUM,K>(q, P, Delta, A, cKeŷ, rKeŷ, ĉt, ctĉts);
```

```
uint64_t ptcts[2][L-3][N];
double wr[2][N/2], wi[2][N/2];
dec<N,L-3> (ctĉts[0], q, S, ptcts[0]);
dec<N,L-3> (ctĉts[1], q, S, ptcts[1]);
decode<N,logN,L-3>(ptcts[0], Delta, q, wr[0], wi[0]);
decode<N,logN,L-3>(ptcts[1], Delta, q, wr[1], wi[1]);
```

$z \in \mathbb{C}^{\frac{N}{2}} \xrightarrow{\Delta} pt \in R_Q$ —— $ct \in R_Q^2$

CtS

$\left[\frac{pt^{1st}}{\Delta}, \frac{pt^{2nd}}{\Delta}\right] \xrightarrow{\Delta}$ —— $ct_{cts} \in (R^2_{Q/\Delta^3})^2$

bit reversed by the matrix R

```
double pt_over_Delta [N];
ifft<N,logN>(zr, zi, pt_over_Delta);
```

$Z \in \mathbb{C}^{\frac{N}{2}} \xrightarrow{fft} \frac{pt}{\Delta} \xrightarrow{\Delta} pt \in R_Q$

```
double pt_over_Delta_bitReversed [N];
bitReverse <N/2> (pt_over_Delta, pt_over_Delta_bitReversed);
bitReverse <N/2> (pt_over_Delta + N/2, pt_over_Delta_bitReversed + N/2);
```

```
double er[2][N/2], ei[2][N/2];
for(int i=0; i<N/2; i++){
    er[0][i] = wr[0][i] - pt_over_Delta_bitReversed[i];
    ei[0][i] = wi[0][i] - 0;
    er[1][i] = wr[1][i] - pt_over_Delta_bitReversed[i+N/2];
    ei[1][i] = wi[1][i] - 0;
}
printf("%e, %e \n", norm_square_exp(er[0], ei[0]), norm_square_exp(er[1],ei[1]));
```

$$\left[\frac{pt^{1st}}{\Delta}, \frac{pt^{2nd}}{\Delta} \right]_{bitReversed}$$

$\updownarrow$ compare, error : $[2.98 \times 10^{-16}, 2.95 \times 10^{-16}]$.

$$[w[0], w[1]] \xleftarrow[\text{decode}]{\Delta} \xleftarrow[\text{dec.}]{} ct\,cts$$