Handouts

October 11, 2023 4:55 PM

Signed 64 bit integer

$F = \mathbb{Z}_p$ and $p^2 < 2^{63}$.

```
#define LONG long long int
int mulmod( int a, int b, int p ) {
    int t = (LONG) a * b % p;
    return t;
}
int addmod( int a, int b, int p ) {
    int t = a-p+b;
    t += (t>>31) & p; // if( t<0 ) t+= p;
    return t;
}
int submod( int a, int b, int p ) {
    int t = a-b;
    t += (t>>31) & p; // if( t<0 ) t+= p;
    return t;
}
```

It runs inplace.
It does not allocate memory.

```
void FFT1( int *A,   // [a0,a1,...,ad,0,...,0] of size n
     int n,     // n = 2^k
     int *W,    // [ powers of w, w^2, w^4, w^8, ... ]
     int p,     // prime < 2^31
     int *T )   // [ scratch array of size n ]
{
   int i,n2,t;
   if( n==1 ) return;
   n2 = n/2;

   for( i=0; i<n2; i++ ) {
      T[   i] = A[2*i];
      T[n2+i] = A[2*i+1];
   }

   FFT1( T,    n2, W+n2, p, A );
   FFT1( T+n2, n2, W+n2, p, A+n2 );

   for( i=0; i<n2; i++ ) {
      t = mulmod(W[i],T[n2+i],p);
      A[   i] = addmod(T[i],t,p);
      A[n2+i] = submod(T[i],t,p);
   }
   return;
}
```

$T+n/2$

$T = [\underbrace{a_0, a_2, a_4, \ldots, a_{n-2}}_{b(x)}, [\underbrace{a_1, a_3, \ldots, a_{n-1}}_{c(x)}]$

In parallel ??

$$W = [1, \omega, \omega^2, ..., \omega^{n/2-1}, 1, \omega^2, \omega^4, ..., \omega^{n/2-2}, 1, \omega^4, \omega^8, ..., \omega^{n/2-4}, ..., 1, 0]$$

In-place FFT routines with permutations and temporary storage T.

$$W = [1, \omega, \omega^2, ..., \omega^{n/2-1}, 1, \omega^2, \omega^4, ..., \omega^{n/2-2}, 1, \omega^4, \omega^8, ..., \omega^{n/2-4}, ..., 1, 0]$$

```
void FFT1( int *A, int n,
      int *W, int p, int *T )
{
  int i,n2,t;
  if( n==1 ) return;
  n2 = n/2;
  for( i=0; i<n2; i++ ) {
     T[   i] = A[2*i];
     T[n2+i] = A[2*i+1];
  }
  FFT1( T,    n2, W+n2, p, A );
  FFT1( T+n2, n2, W+n2, p, A+n2 );
  for( i=0; i<n2; i++ ) {
     t = mulmod(W[i],T[n2+i],p);
     A[   i] = addmod(T[i],t,p);
     A[n2+i] = submod(T[i],t,p);
  }
  return;
}
```

1 move

1 move.

```
void FFT2( int *A, int n,
      int *W, int p, int *T )
{
  int i,n2,t;
  if( n==1 ) return;
  n2 = n/2;
  for( i=0; i<n2; i++ ) {
     T[i] = addmod(A[i],A[n2+i],p);
     t = submod(A[i],A[n2+i],p);
     T[n2+i] = mulmod(t,W[i],p);
  }
  FFT2( T,    n2, W+n2, p, A );
  FFT2( T+n2, n2, W+n2, p, A+n2 );
  for( i=0; i<n2; i++ ) {
     A[ 2*i] = T[i];
     A[2*i+1] = T[n2+i];
  }
  return;
}
```

$A = [a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$

$T = [\overset{0}{a_0}, \overset{1}{a_2}, \overset{2}{a_4}, \overset{3}{a_6}, \underbrace{\overset{0}{a_1}, \overset{1}{a_3}, \overset{2}{a_5}, \overset{3}{a_7}}]$

$A = [\underbrace{\overset{0}{a_0} \overset{1}{a_4}} \underbrace{\overset{0}{a_2} \overset{1}{a_6}} \underbrace{\overset{0}{a_1} \overset{1}{a_5}} \underbrace{\overset{0}{a_3} \overset{1}{a_7}}]$

$T = [a_0\ a_4\ a_2\ a_6\ a_1\ a_5\ a_3\ a_7]$

000	001	010	011	100	101	110	111
0	1	2	3	4	5	6	7
0	4	2	6	1	5	3	7
000	100	010	110	001	101	011	111

$T = [\underbrace{\overset{0}{b_0}, \overset{1}{b_1}}, \underbrace{b_2\ b_3}\ \underbrace{b_4\ b_5}\ \underbrace{b_6\ b_7}]$

$A = [\underbrace{b_0 b_1\ b_2 b_3}\ \underbrace{b_4\ b_5\ b_6\ b_7}]$

$T = [\underbrace{\overset{0}{b_0} b_2 \overset{2}{b_1}\ b_3}\ \ \underbrace{b_4 b_6\ b_5\ b_7}]$

$A\ [\overset{0}{b_0} b_4 \overset{2}{b_2} b_6\ \overset{4}{b_1} b_5 \overset{6}{b_3}\ b_7]$

It's the same π_n.

In binary the permutation reverses the bits.

The permutation π_n is called the bit reversed permutation.

What is $(\pi_n \circ \pi_n)(a) = a$. ie. $\pi_n^{-1} = \pi_n$.

Let $C(n)$ be the # moves of A to T in FFT1.

$$C(n) = 2\frac{n}{2} + 2C(n/2) = n + 2C(\tfrac{n}{2}).$$

$$C(1) = 0$$

$$\Rightarrow C(n) = n \cdot \log_2 n. \text{ moves.}$$

$C(1) = 0$
$\Rightarrow C(n) = n \cdot \log_2 n.$ moves.
Moving data in modern hardware is expensive.

Multiplication in F[x] using the FFT

Input $a, b \in F[x]$ where $a = \sum_{i=0}^{d} a_i x^i$ and $b = \sum_{i=0}^{m} b_i x^i$.
Output $c = a \times b = \sum_{i=0}^{d+m} c_i x^i \in F[x]$.

$C(x) = a(x) \cdot b(x)$

interp. $\uparrow$ eval $\downarrow$ FFT $\downarrow$ eval

$C(\omega^i) = a(\omega^i) \cdot b(\omega^i)$

S1 Pick smallest $n = 2^k > d + m$.
Find $\omega \in F$ with $\omega^n = 1$ and $\omega^i \neq 1$ for $1 \leq i < n$.

Idea: interpolate $c(x)$ from $[\ c(\omega^i) = a(\omega^i)b(\omega^i)\ :\ 0 \leq i < n\ ]$.

S2 Compute $W = [\omega^i\ :\ 0 \leq i < n/2]$.

S3 $\text{FFT}_2(\ n, W, A = [a_0, a_1, \ldots, a_d, 0, \ldots, 0]\ )$ # $A = [a(1), a(\omega), \ldots, a(\omega^{n-1})]$

$\text{FFT}_2(\ n, W, B = [b_0, b_1, \ldots, b_m, 0, \ldots, 0]\ )$ # $B = [b(1), b(\omega), \ldots, b(\omega^{n-1})]$

S4 Compute $C = [\ A_i \times B_i\ :\ 0 \leq i < n\ ]$ # $C = [c(1), c(\omega), \ldots, c(\omega^{n-1})]$

S5 Compute $W = [\omega^{-i}\ :\ 0 \leq i < n/2]$.

$\text{FFT}_1(\ n, W^{-1}, C\ )$ # $C = n[c_0, c_1, \ldots, c_{d+m}, 0, \ldots, 0]$

S6 Return $\sum_{i=0}^{d+m} \frac{1}{n} C_i x^i \in F[x]$

Cost

If we use FFT2 for the forward transforms of A and B then FFT1 for the inverse transform of C the two permutations will cancel out so they can be omitted. In FFT1 and FFT2 below I've removed Π_n and we don't need the tempory array T anymore.

In-place FFT routines with permutation removed.

$W = [1, \omega, \omega^2, ..., \omega^{n/2-1}, 1, \omega^2, \omega^4, ..., \omega^{n/2-2}, 1, \omega^4, \omega^8, ..., \omega^{n/2-4}, ..., 1, 0]$

```
void FFT1( int *A, int n,
           int *W, int p )
{
  int i,n2,t;
  if( n==1 ) return;
  n2 = n/2;
  FFT1( A,    n2, W+n2, p );
  FFT1( A+n2, n2, W+n2, p );
  for( i=0; i<n2; i++ ) {
     s = A[i];
     t = mulmod(W[i],A[n2+i],p);
     A[   i] = addmod(s,t,p);
     A[n2+i] = submod(t,t,p);
  }
  return;
}
```

```
void FFT2( int *A, int n,
           int *W, int p )
{
  int i,n2,t;
  if( n==1 ) return;
  n2 = n/2;
  for( i=0; i<n2; i++ ) {
     s = addmod(A[i],A[n2+i],p);
     t = submod(A[i],A[n2+i],p);
     A[i] = s;
     A[n2+i] = mulmod(t,W[i],p);
  }
  FFT2( A,    n2 W+n2, p );
  FFT2( A+n2, n2, W+n2, p );
  return;
}
```