Fast Fourier Transform (FFT)

CP League Track 2
Web Enthusiasts’ Club NITK

Multiply 2 polynomials

$$A(x) = \sum_{i=0}^{n-1} a_i * x^i, B(x) = \sum_{i=0}^{n-1} b_i * x^i, C(x) = A(x) * B(x)$$

Brute - O(n^2)
FFT - O($nlogn$)

Point Value Form of a Polynomial

$A(x) = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots, (x_{n-1}, y_{n-1})\}$, where $y_k = A(x_k)$ and all the x_k are distinct.

How many points are needed for a polynomial of degree n?

The Idea...

1. Convert $A(x)$ and $B(x)$ from coefficient form to point value form. (FFT)
2. Now do the $O(n)$ convolution in point value form to obtain $C(x)$ in point value form, i.e. basically $C(x) = A(x) * B(x)$ in point value form.
3. Now convert $C(x)$ from point value from to coefficient form (Inverse FFT).

Coefficient to Point Value Form

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix}$$

Point Value to Coefficient Form

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix}$$

Degree of $C(x)$?

How many points to consider for $A(x)$ and $B(x)$?

What points to consider?

```
+-------------------------------------------+  +-----+     +--------+
|  1   1    1     1    ...    1             |  | a_0 |     | A(1)   |
|  1   w    w^2   w^3  ... w^{m-1}          |  | a_1 |     | A(w)   |
|  1   w^2  w^4   w^6  ... w^{2(m-1)}       |  | a_2 | ==  | A(w^2)|
|  1   w^3  w^6   w^9  ... w^{3(m-1)}       |  | a_3 |     | A(w^3)|
|         .....          .....              |  | ... |     | ...    |
|  1 w^{-1} w^{-2} w^{-3} ... w             |  |a_m-1|     | ...    |
+-------------------------------------------+  +-----+     +--------+
```

The FFT Algorithm

```
FFT(A, m, w)
{
  if (m==1) return vector (a_0)
```

The FFT Algorithm

```
FFT(A, m, w)
{
  if (m==1) return vector (a_0)
  else {
    A_even = (a_0, a_2, ..., a_{m-2})
    A_odd  = (a_1, a_3, ..., a_{m-1})
```

The FFT Algorithm

```
FFT(A, m, w)
{
  if (m==1) return vector (a_0)
  else {
    A_even = (a_0, a_2, ..., a_{m-2})
    A_odd  = (a_1, a_3, ..., a_{m-1})
    F_even = FFT(A_even, m/2, w^2)    //w^2 is a primitive m/2-th root of unity
    F_odd = FFT(A_odd, m/2, w^2)
    F = new vector of length m
```

The FFT Algorithm

```
FFT(A, m, w)
{
  if (m==1) return vector (a_0)
  else {
    A_even = (a_0, a_2, ..., a_{m-2})
    A_odd  = (a_1, a_3, ..., a_{m-1})
    F_even = FFT(A_even, m/2, w^2)    //w^2 is a primitive m/2-th root of unity
    F_odd = FFT(A_odd, m/2, w^2)
    F = new vector of length m
    x = 1
    for (j=0; j < m/2; ++j) {
      F[j] = F_even[j] + x*F_odd[j]
      F[j+m/2] = F_even[j] - x*F_odd[j]
      x = x * w
  }
  return F
}
```

Inverse of the below matrix?

```
+--------------------------------------------+
|  1   1     1      1    ...     1            |
|  1   w     w^2    w^3  ... w^{m-1}          |
|  1   w^2  w^4    w^6  ... w^{2(m-1)}        |
|  1   w^3  w^6    w^9  ... w^{3(m-1)}        |
|             .....          .....           |
|  1 w^{-1} w^{-2} w^{-3} ... w               |
+--------------------------------------------+
```

The Final Algorithm

```
Let F_A = FFT(A, m, w)                                 // time O(n log n)
Let F_B = FFT(B, m, w)                                 // time O(n log n)
For i=1 to m, let F_C[i] = F_A[i]*F_B[i]               // time O(n)
Output C = 1/m * FFT(F_C, m, w^{-1}).                  // time O(n log n)
```

Array Convolution using FFT

Problems

1. Problem H : https://codeforces.com/gym/101667/attachments
2. https://codeforces.com/problemset/problem/528/D
3. https://codeforces.com/problemset/problem/632/E