CSE 190, Great ideas in algorithms: Polynomial multiplication and FFT

1 Polynomial multiplication

A univariate polynomial is

$$f(x) = \sum_{i=0}^{n} f_i x^i.$$ 

The degree of a polynomial is the maximal $i$ such that $f_i \neq 0$. The product of two polynomials $f,g$ of degree $n$ each is given by

$$f(x)g(x) = \left( \sum_{i=0}^{n} f_i x^i \right) \left( \sum_{j=0}^{n} g_j x^j \right) = \sum_{i=0}^{n} \sum_{j=0}^{n} f_i g_j x^{i+j} = \sum_{i=0}^{2n} \left( \sum_{j=0}^{\min(i,n)} f_j g_{i-j} \right) x^i.$$ 

So, in order to compute the coefficients of $fg$, we need to compute $\sum_{j=0}^{\min(i,n)} f_j g_{i-j}$ for all $0 \leq i \leq 2n$. This trivially takes time $O(n^2)$. We will see how to do it in time $O(n \log n)$, using a variant of the Fast Fourier Transform (FFT).

1.1 FFT

Let $\omega_n \in \mathbb{C}$ be a primitive $n$-th root of unity, for example $\omega_n = e^{2\pi i/n} = \cos(2\pi/n) + i \sin(2\pi/n)$. The order $n$ Fourier matrix is given by

$$(M_n)_{i,j} = (\omega_n)^{ij} = (\omega_n)^{ij \mod n}$$

What is so special about it? well, as we will see soon, we can multiply it by a vector in time $O(n \log n)$, whereas for general matrices this takes time $O(n^2)$. To keep the description simple, we assume from now on that $n$ is a power of two.

**Theorem 1.1.** For any $x \in \mathbb{C}^n$, we can compute $(M_n)x$ using $O(n \log n)$ additions and multiplications.
Proof. Decompose \( M_n \) into four \( n/2 \times n/2 \) matrices as follows. First, reorder the rows to list first all \( n/2 \) even indices, then all \( n/2 \) odd indices. Let \( M'_n \) be the new matrix, with re-ordered rows. Decompose

\[
M'_n = \begin{pmatrix} A & B \\ C & D \end{pmatrix}.
\]

What are \( A, B, C, D \)? If \( 1 \leq a, b \leq n/2 \) then

- \( A_{a,b} = (M_n)_{2a,b} = (\omega_n)^{2ab} = (\omega_{n/2})^{ab} = (M_{n/2})_{a,b}. \)
- \( B_{a,b} = (M_n)_{2a,b+n/2} = (\omega_n)^{2ab+an} = (\omega_n)^{2ab} = (M_{n/2})_{a,b}. \)
- \( C_{a,b} = (M_n)_{2a+1,b} = (\omega_n)^{2ab+b} = (M_{n/2})_{a,b} \cdot (\omega_n)^b. \)
- \( D_{a,b} = (M_n)_{2a+1,b+n/2} = (\omega_n)^{2ab+b+an+n/2} = -(M_{n/2})_{a,b} \cdot (\omega_n)^b. \)

So, let \( P \) be the \( n/2 \times n/2 \) diagonal matrix with \( P_{a,b} = (\omega_n)^b \). Then

\[
A = B = M_{n/2}, \quad C = -D = M_{n/2}P.
\]

In order to compute \( (M_n)x \), decompose \( x = (x', x'') \) with \( x', x'' \in \mathbb{C}^{n/2} \). Then

\[
(M'_n)x = (Ax + By, Cx + Dy) = (M_{n/2}(x + y), M_{n/2}P(x - y)).
\]

Let \( T(n) \) be the number of additions and multiplications required to multiply \( M_n \) by a vector. Then to compute \( (M_n)x \) we need to: compute \( x + y, x - y, P(x - y) \), which takes time \( 3n \) since \( P \) is diagonal; multiply each of the vectors by \( M_{n/2} \), which takes time \( 2T(n/2) \); and finally reorder the rows back to compute \( (M_n)x \), which takes another \( n \) steps. So we obtain the recursion

\[
T(n) = 2T(n/2) + 4n.
\]

This recursion solves to \( T(n) \leq 4n \log n \):

\[
T(n) \leq 2 \cdot 4(n/2) \log(n/2) + 4n = 4n \log n - 4n + 4n.
\]

\[\square\]

1.2 Inverse FFT

The inverse of the Fourier matrix is the complex conjugate of the Fourier matrix, up to scaling.

Lemma 1.2. \( (M_n)^{-1} = \frac{1}{n} M_n^* \).

Proof. We have \( (M^*_n)_{a,b} = \overline{(\omega_n)^{ab}} = \omega_n^{-ab} \). So

\[
(M_n M_n^*)_{a,b} = \sum_{c=0}^{n-1} (\omega_n)^{(a-b)c}
\]

2
If $a = b$ then the sum equals $n$. We claim that when $a \neq b$ the sum is zero. To see that, let $S = \sum_{i=0}^{n-1} \omega_n^i c$, where $c \neq 0 \mod n$. Then

$$(\omega_n)^c \cdot S = \sum_{i=1}^{n} (\omega_n)^{ic} = \sum_{i=0}^{n-1} (\omega_n)^{ic} = S.$$ 

Since $\omega_n$ has order $n$, we have $\omega_n^c \neq 1$, and hence $S = 0$. \hfill \square

**Corollary 1.3.** For any $x \in \mathbb{C}^n$, we can compute $(M_n)^{-1}x$ using $O(n \log n)$ additions and multiplications.

### 1.3 Fast polynomial multiplication

Let $f(x)$ be a polynomial. Its order $n$ Fourier transform is defined as its evaluations on the $n$-th roots of unity:

$$\hat{f}(i) = f((\omega_n)^i).$$ 

**Lemma 1.4.** Let $f(x)$ be a polynomial of degree $\leq n - 1$. Its Fourier transform can be computed in time $O(n \log n)$.

**Proof.** Let $f(x) = \sum_{i=0}^{n-1} f_i x^i$. Then

$$\hat{f}(j) = \sum_{i=0}^{n-1} f_i (\omega_n)^{ij} = (M_n) \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{pmatrix}$$ 

\hfill \square

**Corollary 1.5.** Let $f$ be a polynomial of degree $\leq n - 1$. Given the evaluations of $f$ at the $n$-th roots of unity, we can recover the coefficients of $f$ in time $O(n \log n)$.

**Proof.** Compute $f = (M_n)^{-1}\hat{f}$. \hfill \square

The Fourier transform of a product has a simple formula:

$$\hat{(fg)}(i) = (fg)((\omega_n)^i) = f((\omega_n)^i) \cdot g((\omega_n)^i) = \hat{f}(i) \cdot \hat{g}(i).$$

So, we can multiply two polynomials as follows: compute their Fourier transform; multiply it coordinate-wise; and then perform the inverse Fourier transform. Note that if $f, g$ have degrees $d, e$, respectively, then $fg$ has degree $d + e$. So, we need to choose $n > d + e$ to compute their product correctly.
Fast polynomial multiplication

Input: two polynomials \( f, g \), of degrees \( d \) and \( e \), respectively, given as lists of coefficients
Output: their product \( fg \) as a list of coefficients

0. Let \( n \) be the smallest power of two, such that \( n \geq d + e + 1 \)
1. Pad \( f, g \) to length \( n \) if necessary (by adding zeros)
2. Compute \( x = (M_n) f \) \hspace{1cm} // \hspace{1cm} \( x = \hat{f} \)
3. Compute \( y = (M_n) g \) \hspace{1cm} // \hspace{1cm} \( y = \hat{g} \)
4. Compute \( z_i = x_iy_i \) for \( 1 \leq i \leq n \) \hspace{1cm} // \hspace{1cm} \( z = \hat{fg} \)
5. Return \( (M_n)^{-1} z \).

1.4 Multivariate polynomials

Let \( f, g \) be multivariate polynomials. For simplicity, let’s consider bivariate polynomials. Let 
\[
 f(x, y) = \sum_{i,j=0}^{n} f_{i,j} x^i y^j, \quad g(x, y) = \sum_{i,j=0}^{n} g_{i,j} x^i y^j.
\]
Their product is
\[
 (fg)(x, y) = \sum_{i,j,i',j'=0}^{n} f_{i,j} g_{i',j'} x^{i+i'} y^{j+j'} = \sum_{i,j=0}^{2n} \left( \sum_{i'=0}^{\min(n,i)} \sum_{j'=0}^{\min(n,j)} f_{i',j'} g_{i-i',j-j'} \right) x^i y^j.
\]
Our goal is to compute \( fg \) quickly. One approach is to define a two-dimensional FFT. Instead, we would reduce the problem of multiplying two bivariate polynomials of degree \( n \) in each variable, to the problem of multiplying two univariate polynomials of degree \( O(n^2) \), and then apply the algorithm using the standard FFT.

Let \( N \) be large enough to be determined later, and define the following univariate polynomials:
\[
 F(z) = \sum_{i,j=0}^{n} f_{i,j} z^{N_i+j}, \quad G(z) = \sum_{i,j=0}^{n} g_{i,j} z^{N_i+j}.
\]
We can clearly compute \( F, G \) from \( f, g \) in linear time, and as \( \deg(F), \deg(G) \leq (N + 1)n \), we can compute \( F \cdot G \) in time \( O((Nn) \log(Nn)) \). The only question is whether we can infer \( f \cdot g \) from \( F \cdot G \).

**Lemma 1.6.** Let \( N \geq 2n + 1 \). If \( H(z) = F(z) G(z) = \sum H_i z^i \) then
\[
 (fg)(x, y) = \sum_{i,j=0}^{2n} H_{N_i+j} x^i y^j.
\]
Proof. We have

\[ H(z) = F(z)G(z) = \left( \sum_{i=0}^{n} f_{i,j} z^{N_i + j} \right) \left( \sum_{i=0}^{n} g_{i',j'} z^{N_{i'} + j'} \right) \]

\[ = \sum_{i,j,i',j'=0}^{n} f_{i,j} g_{i',j'} z^{N(i+i')+(j+j')} . \]

We need to show that the only solutions for

\[ N(i + i') + (j + j') = N i^* + j^* \]

where 0 ≤ i, i', j, j' ≤ n and 0 ≤ i*, j* ≤ 2n, are these which satisfy i + i' = i*, j + j' = j*. As 0 ≤ j + j', j* ≤ 2n and N > 2n, if we compute the value modulo N we get that j + j' = j*, and hence also i + i' = i*. \[\square\]

Corollary 1.7. We can compute the product of two bivariate polynomials of degree n in time \(O(n^2 \log n)\).