CSC006P1M: Design and Analysis of Algorithms *Lecture* 13 *(Polynomial Multiplication Using Fast Fourier Transform (FFT))*

Sumit Kumar Pandey

September 21, 2022

Polynomials

- Consider a polynomial of degree 1, say $a_1x + a_0$. How many points do we need to uniquely define this polynomial? 2.
- For $a_2x^2 + a_1x + a_0$? 3.
- For $a_3x^3 + a_2x^2 + a_1x + a_0$? 4.
- For $a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$? $n + 1$.

Example:
The second-degree polynomial $p(x) = x^2 + 3x + 1$ is defined by the points $(1, 5)$, $(2, 11)$ and $(3, 19)$ and it is the **only** second degree polynomial that includes all these points.

Remark:
These three points (above) are not the only three points that define this polynomial; any three points on the corresponding curve will do.

Polynomial Multiplication

- We can represent an n-degree polynomial by its $n+1$ points.
- This representation is attractive for polynomial multiplication because multiplying the values of points is easy.

For Example:

- The polynomial $p(x) = x^2 + 3x + 1$ can be represented by $(1,5)$, $(2,11)$ and $(3,19)$.
- The polynomial $q(x) = 2x^2 - x + 3$ can be represented by $(1,4)$, $(2,9)$ and $(3,18)$.
- The product of the polynomial $p(x)q(x)$ can be represented by $(1,20)$, $(2,99)$ and $(3,342)$.
- What is $p(x)q(x)$?
- Three points are not sufficient to determine $p(x)q(x)$. Why?
- The degree of $p(x)q(x)$ is 4. So we need 5 points to determine $p(x)q(x)$.

Example contd...

- Evaluate $p(x)$ and $q(x)$ at two more points.
- Add $(0, 1)$ and $(-1, -1)$ to $p(x)$, and $(0, 3)$ and $(-1, 6)$ to $q(x)$.
- Now, we have five points which belong to the product. These are $(1, 20)$, $(2, 99)$, $(3, 342)$, $(0, 3)$ and $(-1, -6)$.
- We need five scalar multiplications.
- The representation for $p(x)q(x)$ is now $(1, 20)$, $(2, 99)$, $(3, 342)$, $(0, 3)$ and $(-1, -6)$.
- Using this idea, we can compute the product of two polynomials of degree n, given in this representation, with only $\Theta(n)$ multiplications.

Polynomial Multiplication

Is this new representation okay?

- How to evaluate $p(x)q(x)$ at some new points?
- How to multiply $p(x)q(x)$ with some new polynomial $r(x)$?

A better way is to convert coordinates into polynomial again. This process is called **interpolation**. See Lagrange's interpolation.

Time Complexity

- Converting from polynomial to coordinates can be done by **polynomial evaluation**. We can compute the value of a polynomial $p(x)$, given by its list of coefficients, at any given point by Horner's rule using n multiplications.
- We need to evaluate $p(x)$ at n arbitrary points, so we require n^2 multiplications.
- Converting from points to coefficients is called **interpolation**, and it also requires $\Theta(n^2)$ operations.
- So the time complexity is $\Theta(n^2)$.

It's worse!!!!!

The key idea is that we do not have to use n **arbitrary** points.

We are free to choose **any** set of n distinct points we want.

The Fast Fourier Transform chooses a very special set of points such that both steps, evaluation and interpolation, can be done quickly.

The Forward Fourier Transform

The Forward Fourier Transform

- We need to evaluate two $n-1$ degree polynomials each at $2n-1$ points, so that their product, which is a $2n-2$ degree polynomial, can be interpolated.
- However, we can always represent an $n-1$ degree polynomial as a $2n-2$ degree polynomial by setting the first $n-1$ (leading) coefficients to zero.
- So, without loss of generality, we assume that the problem is to evaluate an arbitrary polynomial $P = \sum_{i=0}^{n-1} a_i x^i$ of degree $n-1$ at n distinct points.

> We want to find n points for which the polynomials are easy to evaluate.

We assume, for simplicity, that n is a power of 2.

The Forward Fourier Transform

We use matrix terminology to simplify the notation. The evaluation of the polynomial P above for the n points $x_0, x_1, \cdots, x_{n-1}$ can be represented as the following matrix by vector multiplication:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n-1}) \end{bmatrix}$$

- Consider two arbitrary rows i and j.
- We would like to make them as similar as possible to save multiplications.
- We cannot take $x_i = x_j$. (No gain).
- But, we can take $x_i = -x_j$.

The Forward Fourier Transform

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n/2-1} & x_{n/2-1}^2 & \cdots & x_{n/2-1}^{n-1} \\ 1 & -x_0 & (-x_0)^2 & \cdots & (-x_0)^{n-1} \\ 1 & -x_1 & (-x_1)^2 & \cdots & (-x_1)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & -x_{n/2-1} & (-x_{n/2-1})^2 & \cdots & (-x_{n/2-1})^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n/2-1} \\ a_{n/2} \\ a_{n/2+1} \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n/2-1}) \\ P(-x_0) \\ P(-x_1) \\ \vdots \\ P(-x_{n/2-1}) \end{bmatrix}$$

- The $n \times n$ matrix is divided into two submatrices, each of size $n/2 \times n$. These two matrices are very similar.
- For each i such that $0 \leq i \leq n/2$, we have $x_i = -x_{n/2+i}$.
- The coefficients of the even powers are exactly the same in both submatrices, so they need to be computed only once.
- The coefficients of the odd powers are not the same, but they are exactly the negation of each other.

The Forward Fourier Transform

Let $P(x)$ be a polynomial of degree n which is a power of 2.

$$P(x) = E + O = \sum_{i=0}^{n/2-1} a_{2i}x^{2i} + \sum_{i=0}^{n/2-1} a_{2i+1}x^{2i+1}.$$

The even polynomial (E) can be written as a regular polynomial of degree $n/2 - 1$ with the even coefficients of P:

$$E = \sum_{i=0}^{n/2-1} a_{2i}(x^2)^i = P_e(x^2).$$

The odd polynomial (O) can be written in the same way:

$$O = x\sum_{i=0}^{n/2-1} a_{2i+1}(x^2)^i = xP_o(x^2).$$

So, overall, we have the following expression:

$$P(x) = P_e(x^2) + xP_o(x^2),$$

where P_e and P_o are $n/2 - 1$ degree polynomials with the coefficients of the even and odd powers of P respectively.

When we substitute $-x$ for x, we get

$$P(-x) = P_e(x^2) + (-x)P_o(x^2).$$

The Forward Fourier Transform

- $P(x) = P_e(x^2) + xP_o(x^2)$ and
- $P(-x) = P_e(x^2) + (-x)P_o(x^2)$.

We are interested in this matrix.

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n/2-1} & x_{n/2-1}^2 & \cdots & x_{n/2-1}^{n-1} \\ 1 & -x_0 & (-x_0)^2 & \cdots & (-x_0)^{n-1} \\ 1 & -x_1 & (-x_1)^2 & \cdots & (-x_1)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & -x_{n/2-1} & (-x_{n/2-1})^2 & \cdots & (-x_{n/2-1})^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n/2-1} \\ a_{n/2} \\ a_{n/2+1} \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n/2-1}) \\ P(-x_0) \\ P(-x_1) \\ \vdots \\ P(-x_{n/2-1}) \end{bmatrix}$$

- To evaluate $P(x_i)$ and $P(-x_i)$ for $0 \leq i \leq n/2$, we need to compute only $n/2$ values of $P_e(x^2)$ and $P_o(x^2)$.
- And need to perform $n/2$ additions, $n/2$ subtractions, and $n/2$ multiplications.

Time Complexity:

$T(n) = 2T(n/2) + \Theta(n)$.
$T(n) = \Theta(n \lg n)$.

A big improvement.

The Forward Fourier Transform

Let

$$P(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$$

be a 7 degree polynomial.

$$P(x) = (a_0 + a_2x^2 + a_4x^4 + a_6x^6) + x(a_1 + a_3x^2 + a_5x^4 + a_7x^6)$$

So,

$$P_e(x^2) = a_0 + a_2x^2 + a_4x^4 + a_6x^6; \quad P_o(x^2) = a_1 + a_3x^2 + a_5x^4 + a_7x^6.$$

and

$$P(x) = P_e(x^2) + xP_o(x^2)$$

The Forward Fourier Transform

- We want $P(x)$ to be evaluated at 8 points. Let these 8 points be $x_0, x_1, x_2 \cdots, x_7$.
- But these 8 points have a relation, namely $x_0 = -x_4$, $x_1 = -x_5$, $x_2 = -x_6$ and $x_3 = -x_7$.
- Then only, we can exploit this relation $P(x) = P_e(x^2) + xP_o(x^2)$.

What are we expecting from P_e and P_o?

- $P_e(y_0), P_e(y_1), P_e(y_2)$ and $P_e(y_3)$ where $y_0 = x_0^2$, $y_1 = x_1^2$, $y_2 = x_2^2$ and $y_3 = x_3^2$. And similarly,
- $P_o(y_0), P_o(y_1), P_o(y_2)$ and $P_o(y_3)$ where $y_0 = x_0^2$, $y_1 = x_1^2$, $y_2 = x_2^2$ and $y_3 = x_3^2$.

Any arbitrary x_0, x_1, x_2 and x_3 sufficient?

The Forward Fourier Transform

We want $P(x)$ to be evaluated at 8 points. These 8 points are $x_0, x_1, x_2, x_3, -x_0, -x_1, -x_2, -x_3$.

Any arbitrary x_0, x_1, x_2 and x_3 sufficient?

Let us look at the recurrence relation for the time complexity closely - $T(n) = 2T(n/2) + \Theta(n)$.

- $T(n)$ denotes the time complexity to evaluate P at n points. **The important point is that the first half $n/2$ points are negation of other half $n/2$ points**.
- $2T(n/2)$ denotes the time complexity to evaluate P_e and P_o at $n/2$ points. But again **the important point is that the first half $n/4$ points must be the negation of other half $n/4$ points**. Then only, we can use the recurrence relation.

The Forward Fourier Transform

Let us come to our example. What are we expecting from P_e and P_o?

- $P_e(y_0), P_e(y_1), P_e(y_2)$ and $P_e(y_3)$ where $y_0 = x_0^2$, $y_1 = x_1^2$, $y_2 = x_2^2$ and $y_3 = x_3^2$. And similarly,
- $P_o(y_0), P_o(y_1), P_o(y_2)$ and $P_o(y_3)$ where $y_0 = x_0^2$, $y_1 = x_1^2$, $y_2 = x_2^2$ and $y_3 = x_3^2$.

Any arbitrary x_0, x_1, x_2 and x_3 sufficient? No.

- $y_0 = -y_2$ and $y_1 = -y_3$.
- So, $x_0^2 = -x_2^2$ and $x_1^2 = -x_3^2$.
- It is not possible when x_0, x_1, x_2 and x_3 are all real because x_0^2 is positive whereas $-x_2^2$ is negative. Similarly, x_1^2 is positive and $-x_3^2$ is negative.
- So, $x_2 = \iota x_0$ and $x_3 = \iota x_1$ where $\iota^2 = -1$.

The Forward Fourier Transform

- Let us refresh our points - $x_0, x_1, \iota x_0, \iota x_1, -x_0, -x_1, -\iota x_0, -\iota x_1$.
- Are x_0 and x_1 arbitrary? No.
- Similar argument shows that $x_0^4 = -x_1^4$ which implies $x_0 = \iota^{1/2} x_1$.

So, our points are
$x_0, \iota^{1/2} x_0, \iota x_0, \iota \cdot \iota^{1/2} x_0, -x_0, -\iota^{1/2} x_0, -\iota x_0, -\iota \cdot \iota^{1/2} x_0$.

- Let $\omega = \iota^{1/2}$ or $\omega^2 = \iota$ or $\omega^8 = \iota^4 = 1$.
- Then our points are $x_0, \omega x_0, \omega^2 x_0, \omega^3 x_0, \omega^4 x_0, \omega^5 x_0, \omega^6 x_0, \omega^7 x_0$.
- Take $x_0 = 1$. Then our final points are $1, \omega, \omega^2, \omega^3, \omega^4, \omega^5, \omega^6, \omega^7$.

Let us generalise. Let $P(x)$ be a polynomial of degree $n-1$ where n is a power of 2.

- Evaluate $P(x)$ at points $1, \omega, \omega^2, \omega^3, \cdots, \omega^{n-1}$ where $\omega^n = 1$.

Now we are interested in this matrix.

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n-1}) \end{bmatrix}$$

or,

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \cdots & \omega^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{(n-1)\cdot 2} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

The Forward Fourier Transform

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \cdots & \omega^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{(n-1)\cdot 2} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

This product is called the **Fourier Transform** of $(a_0, a_1, \cdots, a_{n-1})$.

FastFourierTransform($n, a_0, a_1, \cdots, a_{n-1}, \omega,\ V$)

Input: n (a power of 2), $a_0, a_1, \cdots, a_{n-1}$ and ω ($\omega^n = 1$).

Output: V (an array in the range $[0 \cdots n-1]$ of output elements).

begin

    if $n = 1$ **then**

        $V[0] := a_0$;

    else

        FastFourierTransform($n/2, a_0, a_2, \cdots, a_{n-2}, \omega^2, U$);

        FastFourierTransform($n/2, a_1, a_3, \cdots, a_{n-1}, \omega^2, W$);

        for $j := 0$ **to** $n/2 - 1$ **do**

            $V[j] := U[j] + \omega^j W[j]$;

            $V[j + n/2] := U[j] - \omega^j W[j]$;

end

The Inverse Fourier Transform

The Inverse Fourier Transform

- The algorithm for the Fast Fourier Transform solves only half of our problem.
- We can evaluate the two given polynomials $p(x)$ and $q(x)$ at the points $1, \omega, \omega^2, \cdots, \omega^{n-1}$, multiply the resulting values, and find the values of the product polynomial $p(x) \cdot q(x)$ at these points.
- Fortunately, the interpolation problem turns out to be very similar to the evaluation problem, and an almost identical algorithm can solve it.

The Inverse Fourier Transform

- Let $h(x) = p(x)q(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}$.
- We evaluate $h(1) = p(1)q(1)$, $h(\omega) = p(\omega)q(\omega)$, $h(\omega^2) = p(\omega^2)q(\omega^2)$, $\cdots$, $h(\omega^{n-1}) = p(\omega^{n-1})q(\omega^{n-1})$.

Consider the following product

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \cdots & \omega^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{(n-1)\cdot 2} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} h(1) \\ h(\omega) \\ h(\omega^2) \\ \vdots \\ h(\omega^{n-1}) \end{bmatrix}$$

We want $(b_0, b_1, b_2, \cdots, b_{n-1})$.

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \cdots & \omega^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{(n-1)} & \omega^{(n-1)\cdot 2} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} h(1) \\ h(\omega) \\ h(\omega^2) \\ \vdots \\ h(\omega^{n-1}) \end{bmatrix}$$

In short,

$$V(\omega)\mathbf{b} = \mathbf{h}$$

So,

$$\mathbf{b} = V(\omega)^{-1}\mathbf{h}.$$

if $V(\omega)$ has an inverse.

$$\mathbf{b} = V(\omega)^{-1}\mathbf{h}.$$

- $V(\omega)$ has an inverse.
- The inverse of an $n \times n$ matrix requires $O(n^3)$ operations.
- But, $V(\omega)^{-1}$ has a very simple form and can be computed in very less operations.

Theorem

$$V(\omega)^{-1} = \frac{1}{n} V(\omega^{-1})$$

The Inverse Fourier Transform

- $\mathbf{b} = V(\omega)^{-1}\mathbf{h}$ can be computed using the same algorithm FastFourierTransform$(n, h_0, h_1, \cdots, h_{n-1}, \omega^{-1}, V)$ in $\Theta(n \lg n)$ operations.
- The above process is called the Inverse Fourier Transform.
- So overall, the product of two polynomials can be computed with $\Theta(n \lg n)$ operations.
- Notice that we need to be able to add and multiply complex numbers.

Thank You