Lecture 5

Yash Mehan

01 Sep 2021, Wed

Fast Fourier Transform

Background

Polynomial multiplication. We have two d degree polynomials, say, $A(x)$ and $B(x)$, we want to calculate $C(x) = A(x)B(x)$ in the quickest possible time. The idea of FFT emerges from the divide and conquer methodology.

$A(x) = a_0 + a_1x + \cdots + a_dx^d$

$B(x) = b_0 + b_1x + \cdots + b_dx^d$

$C(x) = c_0 + c_1x + \cdots + c_{2d}x^{2d}$, where the coefficients $c_k = \sum_{i=0}^{k} a_ib_{k-i}$

Naive algorithm: $O(d^2)$

FFT: $O(d \log d)$

An alternative representation of Polynomials

$A(x)$ can be represented as $a_0+a_1x+a_2x^2+\cdots+a_dx^d$ or in a list $[a_0, a_1, \cdots, a_d]$, where a_i are the coefficients. This is called the Coefficient Representation.

$A(x)$ can also be represented as list of $\{\alpha, A(\alpha)\}$ pairs. This list should have atleast $d + 1$ distinct pairs. (where d is the degree of $A(x)$). This is because a polynomial of degree d can be uniquely identified with $d + 1$ distinct pairs. (proof is trivial).

- Conversion from Coefficient representation to Value representation is called **Evaluation**.
- Conversion from value representation to coefficient is called **Interpolation**.

Plan

1. Given $A(x)$, evaluate it into $2d + 1$ such pairs
2. Do the same for $B(x)$
3. Obtain $C(x_i) = A(x_i)B(x_i)$ for $1 \leq i \leq 2d + 1$

4. Hence Value representation of $C(x)$ is known. Since we have $2d+1$ pairs, $C(x)$ can be uniquely ascertained.
5. Interpolate $C(x)$ back to Coefficient representation.

From the first glance, picking $2d+1$ values to evaluate A and B on, looks trivial enough. Conversion to value representation is $O(d^2)$ again. Interpolation is equally costly, maybe more? Naive interpolation is worse than $O(d^2)$ because it is matrix inversion and multiplication.

Can we create a specific matrix which aids in faster inversion and multiplication?

Evaluation by divide and conquer

Can we choose the points in such a way that there is a lot of *overlap in calculations*?

Suppose we had $P(x) = x^2$, pick evaluation points. We observe that picking $x = 1$ gives the same result as picking $x = -1$, similarly for $x = 2$ and -2, and so on.

Assume $A(x) = x^2 + 3x + 2$, $B(x) = 2x^2 + 1, C(x) = A(x)B(x)$

$C(x) = 3x^5 + 2x^4 + x^3 + 7x^2 + 5x + 1$

If we separate the odd and even powers, i.e.

$C(x) = (2x^4 + 7x^2 + 1) + x(3x^4 + x^2 + 5)$

let the former term be $e(x^2)$ and latter be $o(x^2)$

$\implies C(x) = e(x^2) + x \cdot o(x^2)$

$\implies C(-x) = e(x^2) - x \cdot o(x^2)$

We see a lot of *overlap in calculations.*

If we choose to evaluate $C(x)$ at $x = 1$ then at $x = -1$ isn't very expensive, and we are able to split the big polynomial into two smaller ones with degree $\frac{d}{2} - 1$.

$T(n) = 2T(\frac{n}{2}) + O(N)$

($O(N)$ time for adding)

Now we can evaluate $e(x^2)$ and $o(x^2)$ at $\frac{n}{2}$ points (because as we saw, evaluating at 1 and -1 are *overlapping*), and this becomes a recursive algorithm.

Or does it?

This trick works only for the this step of recursion.

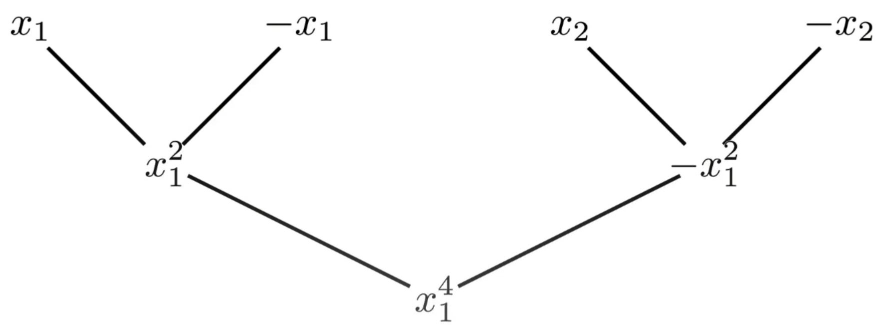
Say, we chose $x = +1, -1, +2, -2, +3, -3, +4, -4$ for evaluation. and now we need to evaluate. Now we need to evaluate $e(x^2) = 2x^4 + 7x^2 + 1$ at $x = 1, 4, 9, 16$. But these aren't $+/-$ pairs. Recursion won't work now. All these are square numbers, and thus can't be paired as $+/-$ pairs.

Or can they be?

Complex numbers

Which complex numbers to use as the evaluation points?

Suppose we have $P(x) = x^3 + x^2 - x - 1$ and need atleast 4 points.



let them be $\pm x_1, \pm x_2$. For the next level of recursion, x_2^2 should be equal to $-(x_1^2)$. We observe numbers bifurcate into their two square roots. Assume $x_1 = 1$, this is how nth roots of unity come into picture. Were n is an exact power of 2, and $n \geq 2d + 1$

This is how we reach the value representation of $C(x)$.

Pseudocode

```
function fft(Polynomial P, omega)
{
    if omega == 1
        return P(1)
    list e = [P[0], P[2], .. P[n-2]]
    list o = [P[1], P[3], .. P[n-1]]
    list y_e = fft(e, omega*omega)
    list y_o = fft(o, omega*omega)

    for j = 0 to n/2
        y[j] = y_e[j] + pow(omega, j)*y_o[j]

    for j = n/2 to n
        y[j] = y_e[j-n/2] - pow(omega, j)*y_o[j-n/2]

    return y
}
// y is the list conataing the values of value_represenation
//y[0] = P(omega^0)
//y[1] = P(omega^1)
```

```
//y[k] = P(omega^k)
```

This algorithm does the evaluation step: takes in the coefficients of a polynomial and evaluates it some special points which are the quickest to calculate.

Interpolation

$$P(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{n-1} x^{n-1}$$

$$\begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

We put x_i as ω^i (where $\omega = e^{\frac{2i\pi}{n}}$)

Evaluation is multiplication by M (the middle, vandermonde matrix), while interpolation is multiplication by M^{-1}

$$P(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{n-1} x^{n-1}$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{bmatrix}$$

Every ω in M^{-1} is $\frac{1}{n}\omega^{-1}$ now

Now, we have the coefficients of $C(x)$.