Week3Lec1_Khush_2020101119

Khush Devendra Patel

Main ideas :

- **Idea: Fast Fourier Transform***
- **Idea: Interpolation***

Fast Fourier Theorem

Polynomial Multiplication : We have two d degree polynomials, say,$A(x)$ and $B(x)$.

We have to get C(x) = A(x)B(x) in the quickest possible time.

The concept of Fast Fourier Theorem emerges from the divide and conquer methodology.

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

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

$C(x) = c_0 + c_1x + \cdots + c_2 *_d \, x_2 *_d$

Naive algorithm : $O(d^2)$

FFT : $O(d * logd)$

Alternate Representation of Polynomials

- **Coefficient Representation** : 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.

- **Value Representation** : A(x) can also be represented as list of $(\alpha, A(\alpha))$ pairs. This list should have at least d + 1 distinct pairs. (where d is the degree of A(x)).
- Conversion from Coefficient representation to Value representation is called **Evaluation**.
- Conversion from value representation to coefficient is called **Interpolation**.

Algorithm

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 ≤ i ≤ 2d+1
4. Therefore value representation of C(x) is known.
5. Interpolate C(x) back to Coefficient representation.

Evaluation by divide and conquer

Suppose we had $P(x) = x^2$, pick evaluation points.

When x = 1 and x = -1, we get same results. Similarly for x = 2 and x = -2

Assume

- $A(x) = x2 + 3x + 2$
- $B(x) = 2x^2 + 1$
- $C(x) = A(x)B(x)$ = $3x^5 + 2x^4 + x^3 + 7x^2 + 5x + 1$

Separating odd and even powers,

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

or $C(x) = e(x^2) + x * o(x^2)$

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

There is a lot of overlap in calculations

If we have already calculated C(x) at x=1, then calculating at x=-1 isn't very expensive. We can split the big polynomial into two smaller units with degree d/2 - 1.

$T(n) = 2T(n/2) + O(N)$. [O(n) for adding]

We can evaluate $e(x^2)$ and $o(x^2)$ at $n/2$ points, and this becomes a recursive algorithm.

But this trick works only for the this step of recursion.

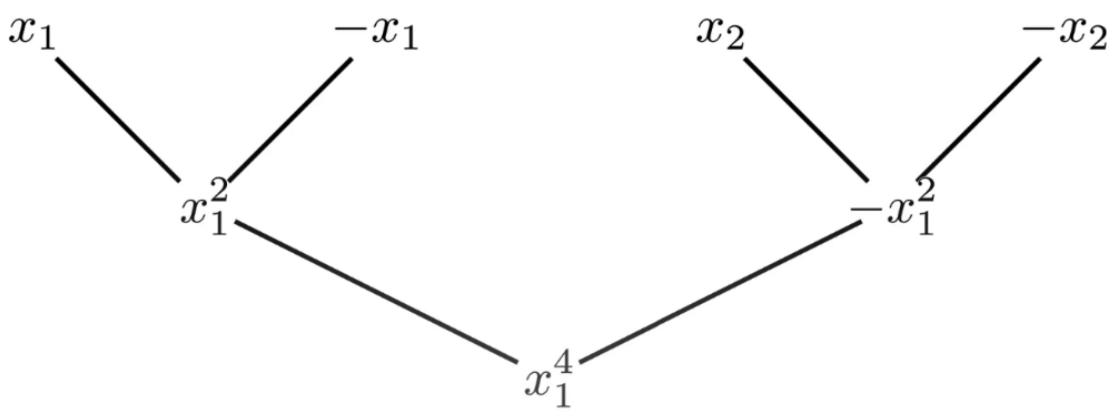
Lets say we need $e(x^2) = 2x^4 + 7x^2 + 1$ at x = 1, 4, 9, 16.

But these aren't positive/negative pairs. Recursions doesn't apply

We can solve it using complex numbers.

Complex Numbers

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



let them be ±x1,±x2. 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 ≥ 2d + 1

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

Algorithm :

function FFT(A, ω)

Input: Coefficient representation of a polynomial $A(x)$
of degree $\leq n-1$, where n is a power of 2
ω, an nth root of unity

Output: Value representation $A(\omega^0), \ldots, A(\omega^{n-1})$

if $\omega = 1$: return $A(1)$
express $A(x)$ in the form $A_e(x^2) + xA_o(x^2)$
call FFT(A_e, ω^2) to evaluate A_e at even powers of ω
call FFT(A_o, ω^2) to evaluate A_o at even powers of ω
for $j = 0$ to $n-1$:
   compute $A(\omega^j) = A_e(\omega^{2j}) + \omega^j A_o(\omega^{2j})$

return $A(\omega^0), \ldots, A(\omega^{n-1})$

Interpolation

After we have values for both polynomials at all n places, we can multiply each pair in linear time. It's just a matter of converting the value representation to the coefficient representation. Surprisingly, by slightly altering the formula, this may be solved in a similar way, taking advantage of the property of inverse of a DFT matrix.

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

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

We put x as ω_i (where ω=e^(2*i*pi/n))

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

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

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

w in M^(-1) is 1/n*(w^(-1))

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