Divide et Impera

Polynomials Multiplication & FFT

Francesco Andreussi

Bauhaus-Universität Weimar

7 June 2019

Bauhaus-Universität Weimar

Fakultät Medien

Polynomials Multiplication

How many operations are usually performed?

Polynomials Multiplication

How many operations are usually performed?

$$O(n^2)$$

Too many to be considered an efficient way of performing the operation, but a "DIVIDE ET IMPERA" algorithm can be applied: the **Karatsuba Algorithm**.

Karatsuba Algorithm I

The Core Idea

Divide $p(x)$ and $q(x)$ (polynomials of degree n) in two parts:
$p(x) = p_1 x^{n/2} + p_0$, $q(x) = q_1 x^{n/2} + q_0$.

Compute the multiplication using this alternative representation:

$$p(x)q(x) = p_1(x)q_1(x) \cdot x^n + \underline{(p_1(x)q_0(x) + p_0(x)q_1(x))} \cdot x^{n/2} + p_0(x)q_0(x)$$

This formula requires 4 multiplications, but the second term can be rewritten in order to be able to perform only 3 product operations (for a performance improvement):

$$r(x) = (p_0(x) + p_1(x)) \cdot (q_0(x) + q_1(x))$$

$$\begin{aligned} p(x)q(x) = {} & p_1(x)q_1(x) \cdot x^n + \\ & \underline{(r(x) - p_0(x)q_0(x) - p_1(x)q_1(x))} \cdot x^{n/2} + \\ & p_0(x)q_0(x) \end{aligned}$$

Karatsuba Algorithm II

Remarks

Both the products $p_1(x)q_1(x)$ and $r(x)$ can be performed applying the Karatsuba algorithm recursively if the factors have degree > 1.
Multiplications between polynomials of degree 1 cannot be simplified; they constitute the base case for the recursion.

In order to have nice numbers throughout all the recursive steps n should be a power of 2.
For more about the topic I suggest checking the wikipedia page.

There is another way of multiplying polynomials, which is even faster, but it is not strictly a "Divide and Conquer" algorithm. It uses the point-value representation of a polynomial, which can be obtained with the Discrete Fourier Transform ($O(n \ln n)$), because, in this representation, the operation is performed in **linear** time.

Recursive FFT Algorithm

Pseudocode

```
RECURSIVE-FFT(a)
n ← length[a]                        ▷ n is a power of 2.
if n = 1
   then return a
ω_n ← e^{2πi/n}
ω ← 1
a^[0] ← (a_0, a_2, ..., a_{n-2})
a^[1] ← (a_1, a_3, ..., a_{n-1})
y^[0] ← RECURSIVE-FFT(a^[0])
y^[1] ← RECURSIVE-FFT(a^[1])
for k ← 0 to n/2 − 1
     do y_k ← y_k^[0] + ω y_k^[1]
        y_{k+(n/2)} ← y_k^[0] − ω y_k^[1]
        ω ← ω ω_n
return y                             ▷ y is assumed to be column vector.
```

Image from Cormen, Leiserson, *Introduction to Algorithms*, 3rd Edition, MIT Press

The Math Underneath the FFT Algorithm I

The n-th Root of Unity

Definition

The **n-th root of unity** is a complex number ω_n such that $\omega_n^n = 1$, $k, n \in \mathbb{N}$

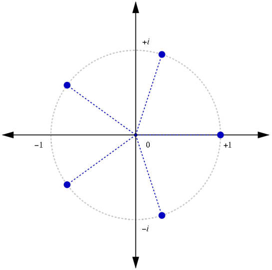

Fig. 1: In blue, the 5th roots of unity

Definition

$(\omega_n)^k = \omega_n^k \neq \omega_n^n$, if $k \neq n \wedge k \neq 0$

Definition

$\omega_n^k = e^{\frac{2\pi ik}{n}} = cos(\frac{2\pi k}{n}) + sin(\frac{2\pi k}{n})i$, with $k, n \in \mathbb{N} \wedge n \neq 0$

There are only $n - 1$ distinct powers of the n-th root:

Theorem

$\omega_n^k \omega_n^j = \omega_n^{k+j} = \omega_n^{(k+j) mod n}$, *with* $k, n, j \in \mathbb{N}$

Proof.

$\omega_n^j = \omega_n^{k+cn} = \omega_n^k \omega_n^{cn} = \omega_n^k (\omega_n^n)^c = \omega_n^k 1^c = \omega_n^k$, with $j > n$ and for every constant c □

Lemma (Cancellation Lemma)

$\omega_{cn}^{ck} = \omega_n^k$, *for every constant* c

FFT Analysis

Let's compute the FFT for [3,0,-5,-10,0,0,6,8]

Thanks for the Attention!