# PhD course: Finite Fields
# Some slides for 3rd Lecture

Diego Ruano

Department of Mathematical Sciences
Aalborg University
Denmark

14-12-2012

# Conway Polynomials from Second lecture

# Another representation of the finite field's elements

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{\overline{0}, \cdots, \overline{p-1}\}$$

Usually we represent the elements of $\mathbb{F}_p$ by

$$0, \ldots p-1$$

However, it can be practical to consider

$$-\frac{p-1}{2}, \ldots, \frac{p-1}{2}$$

Consider a computation over $\mathbb{Z}$ which solution $S$ is bounded by $-p/2 < S < p/2$, then we can perform the computations in over $\mathbb{F}_p$ and then reconstruct the solution uniquely over $\mathbb{Z}$.

# Berlekamp's algorithm

Wikipedia

How do we factorize polynomials in Sage?

# Horner's rule

Let $f = f_0 + f_1 X + \cdots + f_n X^n$ a polynomial in $F[X]$. How many

additions and multiplications do we need to evaluate $f$ at a single point of $F$?

Can we do this in more clever way?

## Horner's rule

$$f(u) = (\cdots(f_n u + f_{n-1})u + \cdots + f_1)u + f_0$$

How many additions and multiplications do we need now?

Polynomials can be considered as functions

Actually, any function over a finite field can be represented by a polynomial, thanks to Lagrange interpolation

The Lagrange interpolant

$$l_i = \prod_{0 \le j < n, j \ne i} \frac{x - u_j}{u_i - u_j}$$

has the property that $l_i(u_j) = 0$ if $i \ne j$ and $l_i(u_i) = 1$.

For arbitrary $v_0, \dots v_{n-1}$, the Lagrange polynomial

$$f = \sum_{0 \le i < n} v_i l_i$$

verifies $f(u_i) = v_i$ for all $i$.

## Complexity

Evaluating a polynomial $f \in F[X]$ of degree less than $n$ at $n$ distinct points $u_0, \ldots, u_{n-1}$ takes $2n^2 - 2n$ operations and the Lagrange interpolation takes $7n^2 - 8n + 1$ operations.

One can also understand evaluation as the $F$-linear map:

$$(f_0, \ldots, f_{n-1}) \mapsto \left( \sum_{0 \le j < n} f_j u_0^j, \ldots \sum_{0 \le j < n} f_j u_{n-1}^j \right)$$

that can be represented using the Vandermonde matrix

$$\begin{pmatrix} 1 & u_0 & u_0^2 & \cdots & u_0^{n-1} \\ 1 & u_1 & u_1^2 & \cdots & u_1^{n-1} \\ 1 & u_2 & u_2^2 & \cdots & u_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & u_{n-1} & u_{n-1}^2 & \cdots & u_{n-1}^{n-1} \end{pmatrix}$$

and interpolation by the inverse matrix.

The naive multiplication algorithm for polynomials has quadratic complexity. Can we do it faster?, yes. For instance we have Karatsuba's algorithm (with complexity $O(n^{\log 3})$) and, if we have a primitive $n$-th root of unity, we have DFT and FFT.

Let $\omega \in F$ (field), we say that it is a primitive $n$-th root of unity if $\omega^n = 1$ and $w^\ell - 1 \neq 0$ for all $1 \leq \ell < n$.

**Lemma**

For a prime power $q$ and $n \in \mathbb{N}$, a finite field $\mathbb{F}_q$ contains a primitive $n$-th root of unity if and only if $n$ divides $q - 1$.

# Discrete Fourier Transform

The following $F$-linear map is called Discrete Fourier Transform:

$$\begin{array}{rcl} \mathrm{DFT}_\omega : F^n & \to & F^n \\ f & \mapsto & (f(1), f(\omega), \ldots, f(\omega^{n-1})) \end{array}$$

The convolution of two polynomials $f, g \in F[X]$ is

$$f *_n g := fg \;\mathrm{rem}\; X^n - 1$$

# Fast convolution algorithm

- INPUT: $f, g$ of degree less than $n = 2^k$, $\omega$ primitive $n$-th root of unity
- OUTPUT: $f * g$

1. Compute $\omega^2, \ldots, \omega^{n-1}$
2. $\alpha = \mathrm{DFT}_\omega(f)$, $\beta = \mathrm{DFT}_\omega(g)$
3. $\gamma = \alpha \cdot \beta$
4. RETURN $\mathrm{DFT}_\omega^{-1}$ = (1/n) $\mathrm{DFT}_{\omega^{-1}}$

To evaluate $f$ at the powers $\omega, \ldots, \omega^{n-1}$, we divide $f$ by $x^{n/2} - 1$ and $x^{n/2} + 1$ with remainder

$$f = q_0(x^{n/2} - 1)r_0 = q_1(x^{n/2} + 1) + r_1$$

- We do not need $q_0$ and $q_1$.
- We can easily rompute $r_0$ and $r_1$. If $f = F_1 x^{n/2} + F_0$:

$$r_0 = F_0 + F_1 \text{ and } r_1 = F_0 - F_1$$

$$f(\omega^{2\ell}) = q_0(\omega^{2\ell})(\omega^{n\ell} - 1) + r_0(\omega^{2\ell}) = r_0(\omega^{2\ell})$$
$$f(\omega^{2\ell+1}) = q_1(\omega^{2\ell+1})(\omega^{n\ell}\omega^{n/2} - 1) + r_1(\omega^{2\ell}) = r_1(\omega^{2\ell})$$

But $r_1(\omega^{2\ell}) = r_1^*(\omega^{2\ell})$ for $r_1^* = (\omega x)$.

# Fast Fourier Transform

- INPUT: $f$ of degree less than $n = 2^k$, $\omega, \ldots, \omega^{n-1}$.
- OUTPUT: $DFT_\omega(f) = (f(1), f(\omega), \ldots, f(w^{n-1}))$

1. IF $n = 1$, RETURN $f_0$
2. $$r_0 = \sum_{0 \leq j < n/2} (f_j + f_{j+n/2}) x^j$$ $$r_1^* = \sum_{0 \leq j < n/2} (f_j - f_{j+n/2}) \omega x^j$$
3. Call algorithm recursively to evaluate $r_0$ and $r_1^*$ at the powers of $\omega^2$
4. RETURN $(r_0(1), r_1^*(1), r_0(\omega^2), r_1^*(\omega^2), \ldots, r_0(\omega^{n-2}), r_1^*(\omega^{n-2}))$

# Complexity

- FFT $((3/2)n \log n$ field operations
- Fast convolution: $(9/2)n \log n + O(n)$ field operations
  1. $n - 2$ multiplications
  2. $2n \log n$ additions and $n \log n$ multiplications by powers of $\omega$
  3. $n$ multiplications
  4. $n \log n$ additions, $(1/2)n \log n$ multiplications by powers of $\omega$ and $n$ divisions by $n$.

We reduce the complexity from $O(n^2)$ to $O(n \log n)$.