Computer Algebra

David Stanovský

Charles University in Prague
Czech Republic

stanovsk@karlin.mff.cuni.cz
http://www.karlin.mff.cuni.cz/~stanovsk

Warszawa, April 2009

Computer Algebra

= computing in various algebraic domains

- polynomials
- linear algebra
- groups
- commutative algebra (fields)
- calculus (derivatives, primitive functions, etc.)
- etc.

numerical mathematics = approximative computing
computer algebra = precise computing

Software:

- Mathematica, Maple, Magma, ..., Sage, Singular, ...
- C++ libraries: GMP, NTL, ...

Polynomial Computer Algebra

My course in Prague: **polynomials**

1. *fast arithmetic*: multiplication, divison, GCD, factorization
2. *more advanced problems & applications*:
    - Gröbner bases
    - lattices and LLL

Domains: $\mathbb{Z}[x]$, $\mathbb{Q}[x]$, $\mathbb{Z}_p[x]$, multivariate polynomials

Arithmetic in $\mathbb{Z}$

$n, m =$ length of arguments, $n \geq m$

$\pm$	school	$\mathcal{O}(n)$
$\cdot$	school	$\mathcal{O}(mn)$
	Karacuba	$\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.585})$
	Toom-3	$\mathcal{O}(n^{\log_3 5}) = \mathcal{O}(n^{1.465})$
	Toom-k	$\mathcal{O}(n^{1+f(k)}),\ f(k) \to 0$
	Schönhage-Strassen	$\mathcal{O}(n \log n \log \log n)$
	Fürer	$n \log n 2^{\mathcal{O}(\log^* n)}$
div, mod	school	$\mathcal{O}(m(n-m+1))$
	Newton's method	$M(n)$
GCD	Eukleid's algorithm	$\mathcal{O}(mn)$

Slow: school methods
Faster: divide & conquer
Fast (very slow): FFT arithmetic

Arithmetic in $\mathbb{Z}$

$n, m =$ length of arguments, $n \geq m$

$\cdot$	school	$\mathcal{O}(mn)$
	Karacuba	$\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.585})$
	Toom-3	$\mathcal{O}(n^{\log_3 5}) = \mathcal{O}(n^{1.465})$
	Toom-k	$\mathcal{O}(n^{1+f(k)}),\ f(k) \to 0$
	Schönhage-Strassen	$\mathcal{O}(n \log n \log \log n)$
	Fürer	$n \log n 2^{\mathcal{O}(\log^* n)}$

Applicability (GMP, 32-bit digits):

school		Karacuba		Toom-3		SS
	10		30-100		300-1000	

Karacuba's multiplication

Divide & conquer. Let

$$a = rB^{n/2} + s \quad \text{and} \quad b = tB^{n/2} + u.$$

Then

$$a \cdot b = (rB^{n/2} + s)(tB^{n/2} + u) = rtB^n + (ru + st)B^{n/2} + su,$$

and we get $T(n) = 4T(n/2) + \mathcal{O}(n)$, i.e. $T(n) = O(n^2)$.

Karacuba's multiplication

Divide & conquer. Let

$$a = rB^{n/2} + s \quad \text{and} \quad b = tB^{n/2} + u.$$

Then

$$a \cdot b = (rB^{n/2} + s)(tB^{n/2} + u) = rtB^n + (ru + st)B^{n/2} + su,$$

and we get $T(n) = 4T(n/2) + \mathcal{O}(n)$, i.e. $T(n) = O(n^2)$.

But also

$$a \cdot b = rtB^n + (rt + su + (r - s)(u - t))B^{n/2} + su$$

and we get $T(n) = 3T(n/2) + \mathcal{O}(n)$, i.e. $T(n) = O(n^{\log_2 3})$.

Arithmetic in $\mathbf{R}[x]$

$n, m =$ degree of arguments, $n \geq m$

$\pm$	school	$\mathcal{O}(m)$
$\cdot$	school Karacuba FFT	$\mathcal{O}(mn)$ $\mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.585})$ $\mathcal{O}(n \log n)$
div, mod	school FFT + Newton's method	$\mathcal{O}(m(n-m+1))$ $\mathcal{O}(n \log n)$

Slow: school methods
Fast (really): Fast Fourier transform

(GCD: quadratic in $\mathbb{Z}_p[x]$, much worse in $\mathbb{Z}[x]$ — see later)

Modular representation

= an *epimorphism* $\mathbf{R} \to \mathbf{R}/M_1 \times \ldots \times \mathbf{R}/M_n$. Hence

$$\mathbf{R}/\bigcap M_i \simeq \mathbf{R}/M_1 \times \ldots \times \mathbf{R}/M_n.$$

1. (Chinese Remainder Theorem) for pairwise coprime m_i's
$$\mathbb{Z}/m_1 \cdots m_n \simeq \mathbb{Z}/m_1 \times \ldots \times \mathbb{Z}/m_n, \quad x \mapsto (\ldots, x \bmod m_i, \ldots)$$
2. (Interpolation Theorem) for pairwise distinct α_i's
$$\mathbf{F}[x]/(x-\alpha_1)\cdots(x-\alpha_n) \simeq \mathbf{F}[x]/(x-\alpha_1) \times \ldots \times \mathbf{F}[x]/(x-\alpha_n) \simeq \mathbf{F}^n$$
$$f \mapsto (\ldots, f(\alpha_i), \ldots)$$

Modular representation

= an *epimorphism* $\mathbf{R} \to \mathbf{R}/M_1 \times \ldots \times \mathbf{R}/M_n$. Hence

$$\mathbf{R}/\bigcap M_i \simeq \mathbf{R}/M_1 \times \ldots \times \mathbf{R}/M_n.$$

1. (Chinese Remainder Theorem) for pairwise coprime m_i's

$$\mathbb{Z}/m_1 \cdots m_n \simeq \mathbb{Z}/m_1 \times \ldots \times \mathbb{Z}/m_n, \quad x \mapsto (..., x \bmod m_i, ...)$$

2. (Interpolation Theorem) for pairwise distinct α_i's

$$\mathbf{F}[x]/(x-\alpha_1)\cdots(x-\alpha_n) \simeq \mathbf{F}[x]/(x-\alpha_1) \times \ldots \times \mathbf{F}[x]/(x-\alpha_n) \simeq \mathbf{F}^n$$

$$f \mapsto (..., f(\alpha_i), ...)$$

1-1: $m_i \mid x - y$, hence $m = m_1 \cdots m_n \mid x - y$
Onto: given $(u_1, \ldots, u_n)$, put

$$f = \sum u_i \cdot \prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

$$x = \sum u_i \cdot n_i r_i, \qquad \text{where } n_i = \prod_{j \neq i} m_j \text{ and } r_i = n_i^{-1} \pmod{m_i}$$

Fast multiplication

Recall with $m = (x - \alpha_1) \cdots (x - \alpha_n)$

$$\varphi : \mathbf{F}[x]/m \simeq \mathbf{F}^n, \quad f \mapsto (..., f(\alpha_i), ...)$$

Idea: If $n > \deg f + \deg g$, then

$$\varphi(f \cdot g) = \varphi(f \cdot g \mod m) = \varphi(f) \cdot \varphi(g).$$

Algorithm:

1. choose $\alpha_1, \ldots, \alpha_n$
2. $\bar{a} = (f(\alpha_1), \ldots, f(\alpha_n))$, $\bar{b} = (g(\alpha_1), \ldots, g(\alpha_n))$
3. $\bar{c} = \bar{a} \cdot \bar{b}$
4. find h such that $(h(\alpha_1), \ldots, h(\alpha_n)) = \bar{c}$

Step *3.* has complexity $\mathcal{O}(n)$!!!

So, how to choose $\alpha_1, \ldots, \alpha_n$ so that the rest is also subquadratic?

Fast Fourier transform

Choose $\alpha_i = \omega^i$, where ω is a primitive n-th root of 1.

Then

- interpolation $= \frac{1}{n}\cdot$ enumeration in ω^{-1}
- *enumeration is fast:* if $g = \sum a_{2i}x^i$ (even terms) and $h = \sum a_{2i+1}x^i$ (odd terms), then $f(\omega^i) = g(\omega^{2i}) + \omega^i h(\omega^{2i})$. Moreover, $\omega^{i+n/2} = -\omega^i$, hence enumeration of f in $\omega^0, \omega^1, \ldots, \omega^{n-1}$ can be recovered from enumeration of halfsize g, h in half points $\omega^0, \omega^2, \omega^{n-2}$. So

$$T(n) = 2T(n/2) + \mathcal{O}(n), \qquad \text{i.e. } T(n) = \mathcal{O}(n \log n).$$

$=$ *2 multiplication & 1 inverse formal power series*

With $h^* = x^{\deg h} h(x^{-1})$, the reciprocal,

$$\begin{aligned} f = gq + r \;\Leftrightarrow\; & f(x^{-1}) = g(x^{-1})q(x^{-1}) + r(x^{-1}) \\ \Leftrightarrow\; & f^* = g^* q^* + x^{n-\deg r} r^* \\ \Leftrightarrow\; & q^* = f^* \cdot (g^*)^{-1} - x^{n-\deg r} \cdot r^* \cdot (g^*)^{-1} \end{aligned}$$

Hence we need $u =$ first $n - m + 1$ terms of the power series $(g^*)^{-1}$ and

$$q^* = f^* \cdot u.$$

Newton's method

Example: compute $\frac{1}{2x+1}$, i.e. $\sum a_i x^i$ such that

$$(\sum a_i x^i) \cdot (2x+1) = 1 + 0x + 0x^2 + \dots .$$

So $a_0 = 1$, $2a_0 + a_1 = 0$, $2a_1 + a_2 = 0$, ... and so

$$h = \sum a_i x^i = \sum (-\frac{1}{2})^i x^i.$$

Newton's method: 1 step = double the number of terms

$$gh = 1 + x^n \cdot s$$
$$gh - 1 = x^n \cdot s$$
$$(gh-1)^2 = x^{2n} \cdot s^2$$
$$g \cdot \underbrace{(h(2-gh))} = 1 + x^{2n} \cdot s^2$$

Newton's method

Example: compute $\frac{1}{2x+1}$, i.e. $\sum a_i x^i$ such that

$$(\sum a_i x^i) \cdot (2x+1) = 1 + 0x + 0x^2 + \dots .$$

So $a_0 = 1$, $2a_0 + a_1 = 0$, $2a_1 + a_2 = 0$, ... and so

$$h = \sum a_i x^i = \sum (-\frac{1}{2})^i x^i .$$

Newton's method: 1 step = double the number of terms

$$gh = 1 + x^n \cdot s$$
$$gh - 1 = x^n \cdot s$$
$$(gh-1)^2 = x^{2n} \cdot s^2$$
$$g \cdot \underbrace{(h(2-gh))} = 1 + x^{2n} \cdot s^2$$

The same works for computing high precision $\frac{1}{a}$: put

$$x = 1/a + 10^{-n}s, \quad \text{i.e. } xa = 1 + 10^{-n}sa$$

Using CRT: Factoring polynomials

... effective in $\mathbb{Z}_p[x]$, difficult in $\mathbb{Z}[x]$

Algorithm:

1. factorize in $\mathbb{Z}_p[x]$ for a small p
2. use Hensel lifting to get factorization in $\mathbb{Z}_{p^k}[x]$ for a sufficiently big p^k
3. recover the result

Possible issue:

- $f = x^3 - 2$ irreducible in $\mathbb{Z}[x]$
- $f \bmod 3 = (x+1)^3$ in $\mathbb{Z}_3[x]$

Using CRT: GCD of polynomials

Issue: in Euclid's algorithm, coefficients grow exponentially fast:

$$x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$$
$$3x^6 + 5x^4 - 4x^2 - 9x + 21$$
$$-5/9x^4 + 1/9x^2 - 1/3$$
$$-117/25x^2 - 9x + 441/25$$
$$233150/19773x - 102500/6591$$
$$-1288744821/543589225$$

Cure: in $\mathbb{Z}_p[x]$, no growth over p

Algorithm: compute mod several p's, recover result with CRT

It follows from theory of *resultants*, that only few p's give a wrong result.

Using CRT: Fast Fourier transform

Issue: $\mathbb{Z}$ has no primitive roots of 1 !!!

Solution: compute mod p such that

1. $\mathbb{Z}_p$ contains the FFT roots of 1
2. $p >$ the max. coefficient of $f \cdot g$

(or, compute mod several small p's, and use CRT to recover)

Summary

- non-trivial mathematical results to solve "trivial problems" (by fast algorithms)
- *Modular method* for
  1. polynomials (interpolation) — fast multiplication
  2. coefficients (Chinese Remainder Theorem) — keep arithmetic fast (GCD), let it work (factorization, FFT)
- *Divide & Conquer* — a general tool for design of fast algorithms
- formal power series aren't just an algebra teacher's toy :-)