CS 4424
Euclidean division

eschost@uwo.ca

Definition

Consider two polynomials A, B; highest coefficient of B is 1.

The quotient and remainder of the division of A by B are defined by

$$A = BQ + R,$$

with

$$\deg(R) < \deg(B).$$

Notation

- $Q = A \text{ div } B$
- $R = A \bmod B$

Remark

- Q and R are unique;
- If $\deg(A) < \deg(B)$, then $Q = 0$ and $R = A$.

Reminder: polynomial multiplication

Let $\mathsf{M}(d)$ denote the cost of polynomial multiplication in degree d:

- $\mathsf{M}(d) \in O(d^2)$ for a naive algorithm
- $\mathsf{M}(d) \in O(d^{1.59})$ for Karatsuba algorithm
- $\mathsf{M}(d) \in O(d \log(d))$ using Fast Fourier Transform (if you have roots of 1)
- $\mathsf{M}(d) \in O(d \log(d) \log\log(d))$ using Fast Fourier Transform in general.

For technical reasons, we ask $\mathsf{M}(d + d') \geq \mathsf{M}(d) + \mathsf{M}(d')$.

Main results

Prop.

- Suppose that
$$\deg(A) = m, \quad \deg(B) = n, \quad m \geq n.$$
- One can compute
$$A \text{ div } B \quad \text{and} \quad A \bmod B$$
using
  - $O(m^2)$ operations — naive algo
  - $O(\mathsf{M}(m))$ operations — Newton iteration

Reminder: integer multiplication

Let $\mathsf{M}_{\mathrm{int}}(d)$ denote the cost of integer multiplication in size d:

- $\mathsf{M}_{\mathrm{int}}(d) \in O(d^2)$ for a naive algorithm
- $\mathsf{M}_{\mathrm{int}}(d) \in O(d^{1.59})$ for Karatsuba algorithm
- $\mathsf{M}_{\mathrm{int}}(d) \in O(d \log(d)\, 2^{\log^*(d)})$ using Fast Fourier Transform.

Technically, we ask $\mathsf{M}_{\mathrm{int}}(d + d') \geq \mathsf{M}_{\mathrm{int}}(d) + \mathsf{M}_{\mathrm{int}}(d')$.

Definition for integers

Consider two integer numbers A, B. The quotient and remainder of the division of A by B are defined by

$$A = BQ + R,$$

with

$$0 \leq R < |B|.$$

Notation

- $Q = A \text{ div } B$
- $R = A \text{ mod } B$

Main results

Prop.

- Suppose that
$$\text{size}(A) = m, \quad \text{size}(B) = n, \quad m \geq n.$$
- One can compute
$$A \text{ div } B \quad \text{and} \quad A \bmod B$$
using
  - $O(m^2)$ operations — naive algo
  - $O(\mathsf{M}_{\text{int}}(m))$ operations — Newton iteration

Rules and examples

An easy special case

When $\deg B = 1$, that is, $B = x - b$, we have $R = A(b)$.

Write

$$A = QB + R \;=\; Q(x - b) + R.$$

Constraint:

$$0 \leq \deg(R) < \deg(B) \implies \deg(R) = 0.$$

So R is a constant. Evaluate at b:

$$A(b) = Q(b) \cdot 0 + R(b) \implies R(b) = A(b).$$

But R is a constant so $R = A(b)$.

Important rules

Basic operations.

- addition

$$(A_1 \bmod B) \;+\; (A_2 \bmod B) \;=\; (A_1 + A_2) \bmod B.$$

- multiplication

$$\big((A_1 \bmod B) \;\times\; (A_2 \bmod B)\big) \;\bmod B \;=\; (A_1 \times A_2) \bmod B.$$

Both proofs are consequences of the uniqueness.

Other easy cases

Application: $B = x^d - b$.

- $x^d \bmod B = b$.

  Proof:

$$x^d = 1 \cdot (x^d - b) + b.$$

- $(x^d A) \bmod B = b(A \bmod B)$.

  Proof: multiplication rule.

- $x^{kd} \bmod B = b^k$.

  Proof: multiplication rule.

Conclusion:

- to get $A \bmod B$, replace all x^d by b.

  Proof: work it out, multiplication + addition rules.

Modular computations

Some notation

A multi-level construction:

- if R is the ring of coefficients ...

  $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{F}_2$, ...

- then $\mathsf{R}[x]$ is the ring of polynomials with coefficients in R

- then $\mathsf{R}[x]/P$ is the ring of polynomials with coefficients in R modulo P.

Operations in $\mathsf{R}[x]/P$

Let $n = \deg(P)$.

Elements of $\mathsf{R}[x]/P$

- polynomials of degree less than n

Addition in $\mathsf{R}[x]/P$

- normal addition

Multiplication in $\mathsf{R}[x]/P$

- normal multiplication, followed by reduction modulo P.

Complex numbers

Let $\mathsf{R} = \mathbb{R}$ and $P = x^2 + 1$, so $n = 2$.

Elements of $\mathbb{R}[x]/(x^2+1)$

- polynomials of degree less than 2: $a + bx$

Addition in $\mathbb{R}[x]/(x^2+1)$

- $(a + bx) \;+\; (a' + b'x) \;=\; (a + a') \;+\; (b + b')x.$

Multiplication in $\mathbb{R}[x]/(x^2+1)$

- $(a + bx) \;\times\; (a' + b'x) \;= \;?$
  - normal multiplication $\qquad aa' + (a'b + ab')x + bb'x^2$
  - remainder: replace x^2 by -1 $\qquad (aa' - bb') \;+\; (a'b + ab')x$
  - this is the complex multiplication.

Complexity of computing in $\mathsf{R}[x]/P$

Prop. Let $n = \deg(P)$.

- addition in $\mathsf{R}[x]/P$
  - $O(n)$, easy.
- multiplication in $\mathsf{R}[x]/P$
  - $O(n^2)$, naively.
  - $O(\mathsf{M}(n))$, fast Euclidean division.
- inversion in $\mathsf{R}[x]/P$
  - not always possible;
  - more tricky.

Similar constructions for integers

Given a positive integer N, we define $\mathbb{Z}/N\mathbb{Z}$ as we did for polynomials:

Elements of $\mathbb{Z}/N\mathbb{Z}$

- integers in $\{0, \ldots, N-1\}$

Addition in $\mathbb{Z}/N\mathbb{Z}$

- normal addition, followed by reduction mod N

Multiplication in $\mathbb{Z}/N\mathbb{Z}$

- normal multiplication, followed by reduction modulo N

Remark: when N is prime, we also write $\mathbb{F}_N$.

Complexity of computing in $\mathbb{Z}/N\mathbb{Z}$

Prop. Let $n = \text{size}(N)$.

- addition in $\mathbb{Z}/N\mathbb{Z}$
  - $O(n)$, easy.
- multiplication in $\mathbb{Z}/N\mathbb{Z}$
  - $O(n^2)$, naively.
  - $O(\mathsf{M}_{\text{int}}(n))$, fast Euclidean division.
- inversion in $\mathbb{Z}/N\mathbb{Z}$
  - not always possible;
  - more tricky.

Applications

RSA

- a public-key crypto procotol used behind the scenes when you use `ssh`, `https`, ...
- the security relies on the difficulty of factoring numbers

Basics

- I keep a private key (an integer d)
- I publish a public key (two integers n and e)
- messages are represented by integers

Encryption and decryption

- to send me a message m, you compute $c = m^e \bmod n$
- using my private key, (only) I can find $m = c^d \bmod n$

Naive algorithm

A recursive presentation

Algorithm reduction(A, B)

1. if $\deg(A) < \deg(B)$ return A
2. write
$$A = a_0 + \cdots a_m x^m, \quad B = b_0 + \cdots + x^n$$
by assumption, $m \geq n$
3. compute
$$A' = A - a_m x^{m-n} B$$
remark that A' looks like
$$A = a'_0 + \cdots a'_{m-1} x^{m-1}$$
4. return reduction(A', B)

Analysis

Correctness

- $A \bmod B \;=\; A' \bmod B$.

  Proof.

  - definition of $A' = A - a_m x^{m-n} B$
  - use addition and multiplication rules.

Complexity

- $T(m) = T(m-1) + (\text{time for computing } A')$
- A' can be computed in linear time $O(m)$
- so $T(m) \in O(m^2)$.

Remark: we could (and will) refine the analysis.

Power series

Power series

A power series is a formal sum of the form

$$A = \sum_{i \geq 0} a_i x^i.$$

Computationally we only handle truncated power series

$$A \bmod x^d = \sum_{i < d} a_i x^i,$$

which are plain polynomials.

However,thinking in terms of series often makes things clearer.

Operations on series

Addition, multiplication are defined like those of polynomials.

- addition is done term-by-term;
- multiplication is done using the same formulas as polynomials.

Algorithmically

- you only represent truncations,
- then the algorithms are the same as those for polynomials
  - addition at precision n is $O(n)$
  - multiplication at precision n is $\mathsf{M}(n)$

Inverse of a power series

Main difference with polynomials:

- many functional relations have solutions that series, but not polynomials
- first example: inversion.

Easy case: let $P(x) = x - 1$.

- there is no polynomial $Q(x)$ such that

$$PQ = 1$$

- but there exists a series $Q(x)$ such that $PQ = 1$:

$$Q = 1 + x + x^2 + x^3 + x^4 + \cdots$$

Inverse in general

Prop.

- for any series P, with constant coefficient(P) $\neq 0$, there exists a unique series Q with
$$PQ = 1.$$

- Exercise: give an algorithm of complexity $O(n^2)$ to compute $Q_n = Q \bmod x^n$

- Better: one can compute $Q_n = Q \bmod x^n$ in $O(\mathsf{M}(n))$ operations.
Next lecture.

- then, $PQ_n \bmod x^n = 1$, which means
$$PQ_n = 1 + 0 \cdot x + \cdots + 0 \cdot x^{n-1} + r_n x^n + \cdots$$

The fast algorithm

Euclidean division

We want to compute the quotient and remainder

$$A = BQ + R,$$

with $\deg(A) = m$ and $\deg(B) = n$.

Prop.

- The cost is $O(\mathsf{M}(m))$.

How?

- We reduce this to power series inversion.
- We get Q first.
- Once Q is known, we get $R = A - BQ$ $O(\mathsf{M}(m))$

Details

Rewrite the equality as

$$\frac{A}{B} = Q + \frac{R}{B}.$$

Idea: let $x \to \infty$.

- Remember $\deg(R) < \deg(B)$.
- So $R/B \to 0$.
- So the expansion of A/B at ∞ gives Q.

Formally, replace x by $1/y$, giving

$$\frac{A(1/y)}{B(1/y)} = Q(1/y) + \frac{R(1/y)}{B(1/y)}.$$

More details

From $A = a_0 + a_1 x + \cdots + a_m x^m$, we get

$$A(\frac{1}{y}) = a_0 + \frac{a_1}{y} + \cdots + \frac{a_m}{y^m}.$$

Now, remark that

$$\deg(Q) = m - n, \quad \deg(R) \leq n - 1.$$

So doing the same with the others, we get

$$\frac{a_0 + \frac{a_1}{y} + \cdots + \frac{a_m}{y^m}}{b_0 + \frac{b_1}{y} + \cdots + \frac{b_n}{y^n}} = q_0 + \frac{q_1}{y} + \cdots + \frac{q_{m-n}}{y^{m-n}} + \frac{r_0 + \frac{r_1}{y} + \cdots + \frac{r_{n-1}}{y^{n-1}}}{b_0 + \frac{b_1}{y} + \cdots + \frac{b_n}{y^n}}$$

Conclusion

Multiply by y^{m-n}.

1. On the left, we get

$$\frac{a_m + a_{m-1}y + \cdots + a_0 y^m}{b_n + b_{n-1}y + \cdots + b_0 y^n}$$

2. On the right, we get

$$q_{m-n} + q_{m-n-1}y \cdots + q_0 y^{m-n} \quad + \quad y^{m-n+1} \, \frac{r_{n-1} + r_{n-2}y + \cdots + r_0 y^{n-1}}{b_n + b_{n-1}y + \cdots + b_0 y^n}.$$

So

$$q_{m-n} + q_{m-n-1}y \cdots + q_0 y^{m-n} \quad = \quad \frac{a_m + a_{m-1}y + \cdots + a_0 y^m}{b_n + b_{n-1}y + \cdots + b_0 y^n} \bmod y^{m-n+1}.$$

Algorithm

Algorithm reduction(A, B)

1. if $\deg(A) < \deg(B)$ return A
2. let
$$B^\star = b_n + \cdots + b_0 y^n, \quad A^\star = a_m + \cdots + a_0 y^m$$
3. compute $S = 1/B^\star \bmod y^{m-n+1}$
4. compute $Q^\star = A^\star S \bmod y^{m-n+1}$
5. deduce Q
6. deduce R