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 through

$$A = BQ + R,$$

with

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

**Notation**

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

**Remark**

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

Let $M(d)$ denote the cost of polynomial multiplication in degree $d$:

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

Technically, we ask $M(d + d') \geq M(d) + M(d')$. 
Main results

Prop.

• Suppose that

\[ \text{deg}(A) = m, \quad \text{deg}(B) = n, \quad m \geq n. \]

• One can compute

\[ A \div B \quad \text{and} \quad A \rem B \]

using

- \( O(m^2) \) operations \hspace{1cm} \text{naive algo}

- \( O(M(m)) \) operations \hspace{1cm} \text{Newton iteration}
Reminder: integer multiplication

Let $M_{\text{int}}(d)$ denote the cost of integer multiplication in size $d$:

- $M_{\text{int}}(d) \in O(d^2)$ for a naive algorithm
- $M_{\text{int}}(d) \in O(d^{1.59})$ for Karatsuba algorithm
- $M_{\text{int}}(d) \in O(d \log(d) 2^{\log^*(d)})$ using Fast Fourier Transform.

Technically, we ask $M_{\text{int}}(d + d') \geq M_{\text{int}}(d) + M_{\text{int}}(d')$. 
Definition for integers

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

$$A = BQ + R,$$

with

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

**Notation**

- $Q = A \text{ div } B$
- $R = A \text{ rem } B$
Main results

Prop.

- Suppose that
  \[ \text{size}(A) = m, \quad \text{size}(B) = n, \quad m \geq n. \]

- One can compute
  \[ A \div B \quad \text{and} \quad A \rem B \]
  using
  - \(O(m^2)\) operations \(\text{naive algo}\)
  - \(O(M_{\text{int}}(m))\) operations \(\text{Newton iteration}\)
Rules and examples
Some easy special cases

When \( \deg B = 1 \), that is, \( B = x - b \). 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

\[(A_1 \text{ rem } B) + (A_2 \text{ rem } B) = (A_1 + A_2) \text{ rem } B.\]

- multiplication

\[((A_1 \text{ rem } B) \times (A_2 \text{ rem } B)) \text{ rem } B = (A_1 \times A_2) \text{ rem } B.\]

Both proofs are consequences of the uniqueness.
Other easy cases

Application: \( B = x^d - b. \)

- \( x^d \mod B = b. \)
  
  **Proof:**
  
  \[ x^d = 1 \cdot (x^d - b) + b. \]

- \( (x^d A) \mod B = b \cdot (A \mod B). \)
  
  **Proof:** multiplication rule.

- \( x^{kd} \mod B = b^k. \)
  
  **Proof:** multiplication rule.

- **to get** \( A \mod B \), replace all \( x^d \) by \( b. \)
  
  **Proof:** work it out, multiplication + addition rules.
Important application: modular computations
Some notation

A multi-level construction:

- if $R$ is the ring of coefficients . . .
  \[ \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{F}_2, \ldots \]
- then $R[x]$ is the ring of polynomials with coefficients in $R$
  it is a ring too
- then $R[x]/P$ is the ring of polynomials with coefficients in $R$ modulo $P$. 
Operations in $R[x]/P$

Let $n = \deg(P)$.

**Elements of $R[x]/P$**

- polynomials of degree less than $n$

**Addition in $R[x]/P$**

- normal addition

**Multiplication in $R[x]/P$**

- normal multiplication, followed by reduction modulo $P$. 
Complex numbers

Let $\mathbb{R} = \mathbb{R}$ 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 $aa' + (a'b + ab')x + bb'x^2$
  - remainder: replace $x^2$ by $-1$ $(aa' - bb') + (a'b + ab')x$
  - this is the complex multiplication.
Complexity of computing in $\mathbb{R}[x]/P$

Prop. Let $n = \text{deg}(P)$.

- addition in $\mathbb{R}[x]/P$
  - $O(n)$, easy.

- multiplication in $\mathbb{R}[x]/P$
  - $O(n^2)$, naively.
  - $O(M(n))$, fast Euclidean division.

- inversion in $\mathbb{R}[x]/P$
  - 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.
  - $O(M_{\text{int}}(n))$, fast Euclidean division.

- inversion in $\mathbb{Z}/N\mathbb{Z}$
  - not always possible;
  - more tricky.
Naive algorithm
An easy recursive presentation

**Algorithm** reduction($A, B$)

1. if deg($A$) < deg($B$) return $A$

2. write

   $$A = a_0 + \cdots + a_mx^m, \quad B = b_0 + \cdots + x^n$$

   by assumption, $m \geq n$

3. compute

   $$A' = A - a_mx^{m-n}B$$

   remark that $A'$ looks like

   $$A = a'_0 + \cdots + a'_{m-1}x^{m-1}$$

4. return reduction($A', B$)
Analysis

Correctness

- $A \text{ rem } B = A' \text{ rem } B$.

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

\[ \sum_{i \geq 0} a_i x^i. \]

(of course, there is no idea of convergence in the usual sense for these things.)

Computationnally, we only handle **truncated power series**

\[ \sum_{i \geq 0} a_i x^i \quad \text{rem} \quad x^d = \sum_{i < d} a_i x^i \]

which are then 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 $M(n)$
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 $\text{constant\_coefficient}(P) \neq 0$, there exists a unique series $Q$ with

$$PQ = 1.$$ 

• one can compute $Q_n = Q \text{ rem } x^n$ in $O(M(n))$ operations.

Next lecture.

• then, $PQ_n \text{ rem } x^n = 1$, 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 $\text{deg}(A) = m$ and $\text{deg}(B) = n$.

Prop.

- The cost is $O(M(m))$.

How?

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

Rewrite the equality as

\[
\frac{A}{B} = Q + \frac{R}{B}.
\]

**Idea:** let \( x \to \infty \).

- Remember \( \text{deg}(R) < \text{deg}(B) \).
- So \( R/B \to 0 \).
- So the expansion of \( A/B \) at \( \infty \) 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

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

Now, remark that

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

So doing the same with the others, we get

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

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

So

$$q_{m-n} + q_{m-n-1}y \cdots + q_0 y^{m-n} = \frac{a_m + a_{m-1}y + \cdots + a_0 y^m}{b_n + b_{n-1}y + \cdots + b_0 y^n} \mod y^{m-n+1}.$$
Algorithm reduction($A, B$)

1. if $\text{deg}(A) < \text{deg}(B)$ return $A$
2. let
   \[ B^* = b_n + \cdots + b_0y^n, \quad A^* = a_m + \cdots + a_0y^m \]
3. compute $S = 1/B^* \mod y^{m-n+1}$
4. compute $Q^* = AS \mod y^{m-n+1}$
5. deduce $Q$
6. deduce $R$