CS 487 / · · ·

Introduction to Symbolic Computation

University of Waterloo
Éric Schost
eschost@uwaterloo.ca
GCD, XGCD
Algorithms over rings

So far, we discussed

- multiplication of polynomials
- Euclidean division by a **monic** polynomial
  (monic: leading coefficient is 1)

When can we apply them?

- Karatsuba, Newton iteration work over any **ring**
  \( \mathbb{Z}/2\mathbb{Z} = \{0, 1\}, \mathbb{Z}/3\mathbb{Z} = \{0, 1, 2\}, \mathbb{Z}/4\mathbb{Z} = \{0, 1, 2, 3\}, \mathbb{Q}, \mathbb{Z}, \ldots \)
- the FFT we presented does not
- but a more complicated version of it does.

The algorithms do not “branch”

- trace of the algorithms is always the same
A **field** is a structure where we can **divide**.

**Def.**
- A ring $R$ is a field if for every non-zero element $\alpha$ of $R$, there exists $\beta$ in $R$ such that $\alpha \beta = 1$.
- $\beta$ is the inverse of $\alpha$.

**Examples of fields**
- $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}/2\mathbb{Z}$, $\mathbb{Z}/3\mathbb{Z}$

**Examples of not fields**
- $\mathbb{Z}$, $\mathbb{Z}/4\mathbb{Z}$

**Notation:** $R$ for rings, $K$ for fields (Körper)
Why divisions?

In this chapter:

• to make it possible to do Euclidean division by non-monic polynomials

Examples

• \((x^2 + 2x + 2) \mod (2x + 1) = \frac{5}{4} \text{ over } \mathbb{Q}\)

• \((x^2 + 2x + 2) \mod (2x + 1) = 2 \text{ over } \mathbb{Z}/3\mathbb{Z}\)

• \((x^2 + 2x + 2) \mod (2x + 1) = ??? \text{ over } \mathbb{Z}/4\mathbb{Z}\)

How?

• make it monic

• \((x^2 + 2x + 2) \mod (2x + 1) = (x^2 + 2x + 2) \mod (x + \frac{1}{2})\)
GCD of polynomials

**Definition**

- Let $A$ and $B$ be in $K[x]$.
  - $K[x]$ is the ring of polynomials with coefficients in field $K$

- A **Greatest Common Divisor** of $A$ and $B$ is a polynomial $G$ such that
  - $G$ divides $A$
  - $G$ divides $B$
  - if $C$ divides both $A$ and $B$, it divides $G$. 
GCD of polynomials

Definition

• Let $A$ and $B$ be in $K[x]$. $K[x]$ is the ring of polynomials with coefficients in field $K$

• A **Greatest Common Divisor** of $A$ and $B$ is a polynomial $G$ such that
  
  • $G$ divides $A$
  • $G$ divides $B$
  • if $C$ divides both $A$ and $B$, it divides $G$.

• If $G$ and $H$ are GCD’s of $A$ and $B$, then $G = \ell H$, for some constant $\ell \neq 0$.

• So usually we say that **THE GCD** is the one with leading coefficient =1.
Another definition of the GCD

- Any polynomial can be factored into a product of irreducibles
- The GCD of $A$ and $B$ is obtained by finding their common irreducible factors, and keeping the minimum of their exponents in $A$ and $B$.
- Example: $A = x(x^2 + 1)$ and $B = (x - 1)(x + 1)(x^2 + 1)^2$ in $\mathbb{Q}[x]$ gives $\text{GCD}(A, B) = (x^2 + 1)^1$. 
Facts

• The second definition does not lead to an easy algorithm.
• To do better: Euclid's algorithm.

Complexity

• The naive version of Euclid’s algorithm takes $O(n^2)$ for polynomials of degree $n$.
• The fast version takes $O(M(n) \log (n))$. 
A few useful rules

Prop.

- \( \gcd(A, B) = \gcd(B, A) \).
  The definition is symmetric.

- \( \gcd(A, 0) = A / \text{leading coefficient}(A) \).
  \( A \) divides \( A \), and \( A \) divides \( 0 \), so \( A \) divides their GCD.
  Conversely, the GCD divides \( A \). So the GCD is a constant times \( A \).

- \( \gcd(A, c) = 1 \) if \( c \) is a non-zero constant.
  Any polynomial that divides \( c \) is a constant.
Prop.

• For all $A, B$ in $K[x]$,

\[
gcd(A, B) = gcd(A, B \mod A) = gcd(B, A \mod B).
\]
The main idea

Prop.

• For all $A, B$ in $K[x],$

\[ \gcd(A, B) = \gcd(A, B \mod A) = \gcd(B, A \mod B). \]

Proof.

• Let $R = B \mod A$. Then

\[ R = B - AQ. \]

• Let $G = \gcd(A, B)$ and $H = \gcd(A, R)$.

• $G$ divides $A$ and $B$, so $G$ divides $R$.

Property of the GCD for $H$: $G$ divides $H$.

• $H$ divides $A$ and $R$, so $H$ divides $B$.

Property of the GCD for $G$: $H$ divides $G$. 
Euclid’s algorithm

\[ \text{gcd}(A, B) \]

- if \( \text{deg}(A) < \text{deg}(B) \) then return \( \text{gcd}(B, A) \).

so now we assume that \( \text{deg}(A) \geq \text{deg}(B) \)

- if \( B = 0 \) then return \( A / \text{leading coefficient}(A) \).

second rule

- return \( \text{gcd}(B, A \mod B) \)

previous slide
Towards the iterative algorithm

Setup.

- We rewrite $A_0 = A$, $A_1 = B$.
- We assume $\deg(A_0) \geq \deg(A_1)$ (otherwise, swap them).

Steps.

- $\gcd(A_0, A_1) = \gcd(A_1, A_2)$
- $A_2 = A_0 \mod A_1$
- $\gcd(A_1, A_2) = \gcd(A_2, A_3)$
- $A_3 = A_1 \mod A_2$
- $\ldots$
- $\gcd(A_i, A_{i+1}) = \gcd(A_{i+1}, A_{i+2})$
- $A_{i+2} = A_i \mod A_{i+1}$
- $\ldots$
- $\gcd(A_N, 0) = A_N$/leading coefficient($A_N$).
The iterative algorithm

Setup.

- We rewrite $A_0 = A$, $A_1 = B$.
- We assume $\deg(A_0) \geq \deg(A_1)$ (otherwise, swap them).

**gcd($A_0, A_1$)**

- $i = 1$
- while $A_i \neq 0$
  - $A_{i+1} = A_{i-1} \mod A_i$
  - $i++$
- return $A_{i-1}/\text{leading coefficient}(A_{i-1})$
Complexity

Setup.

- \( n = \deg(A_0) \)
- then, all polynomials have degree \( \leq n \).

Naive analysis.

- We do at most \( n + 1 \) Euclidean divisions.
- Euclidean division in degree \( \leq n \) takes \( O(n^2) \) operations.
- So the total cost is \( O(n^3) \).

Correct result, but we will do much better.
Over \( \mathbb{Z}/3\mathbb{Z} \): 

\[
A_0 = 1 + 2x + x^2 + x^3 + 2x^4 \\
A_1 = 1 + 2x + x^2 + x^3 \\
A_2 = 2 + 2x + x^2 \\
A_3 = 2x \\
A_4 = 2 \\
A_5 = 0
\]
Let $\deg(A_0) = n$ and $\deg(A_1) = m$

For fixed $n$ and $m$, the input of the GCD algorithm is a vector of $(n + 1) + (m + 1)$ coefficients $a_{0,0}, \ldots, a_{0,n}, a_{1,0}, \ldots, a_{1,m}$ in $K$.

We want to describe the shape of the remainder sequence for “general” input.

**Similar questions**

- A “general” $2 \times 2$ (or $n \times n$) matrix is invertible.

- A “general” monic polynomial of degree 2 has no multiple root.
Genericity for algebraic algorithms

Introduce \textbf{indeterminates} that stand for the coefficients of the polynomials / entries of the matrix / . . . .

Matrices.
There exists a not-identically-zero polynomial

\[ \Delta(A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2}) = A_{1,1}A_{2,2} - A_{2,1}A_{1,2} \]

such that for all \( a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2} \) in \( K \), if the value

\[ \Delta(a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}) \]

is not zero,

matrix \[
\begin{bmatrix}
a_{1,1} & a_{1,2} \\
a_{2,1} & a_{2,2}
\end{bmatrix}
\]
is invertible.

In particular, if \( K = \mathbb{R} \), the bad guys have measure zero.
Introduce indeterminates that stand for the coefficients of the polynomials / entries of the matrix / . . . .

Polynomials.
There exists a not-identically-zero polynomial

$$\Gamma(A_0, A_1) = A_1^2 - 4A_0$$

such that for all $a_0, a_1$ in $K$, if the value

$$\Gamma(a_0, a_1)$$

is not zero,

polynomial $x^2 + a_1 x + a_0$ has no double root.

In particular, if $K = \mathbb{R}$, the bad guys have measure zero.
There exists a non-zero polynomial

$$\Lambda(A_{0,0}, \ldots, A_{0,n}, A_{1,0}, \ldots, A_{1,m})$$

such that for all $a_{0,0}, \ldots, a_{0,n}, a_{1,0}, \ldots, a_{1,m}$ in $K$, if the value

$$\Lambda(a_{0,0}, \ldots, a_{0,n}, a_{1,0}, \ldots, a_{1,m})$$

is not zero, the degrees of the polynomials in the remainder sequence are

$$n, m, m - 1, m - 2, \ldots, 2, 1, 0.$$

**Example:** with $n = 3, m = 2$,

$$\Lambda(A_{0,0}, A_{0,1}, A_{0,2}, A_{0,3}, A_{1,0}, A_{1,1}, A_{1,2}) = A_{1,2} \times (A_{0,1}A_{1,2}^2 - A_{0,2}A_{1,1}A_{1,2} - A_{0,3}A_{1,0}A_{1,2} + A_{0,3}A_{1,1}^2)$$

$$\times (A_{0,0}^2A_{1,2}^3 + (11 \text{ terms}) + A_{0,3}^2A_{1,0}^3).$$
There exists a non-zero polynomial

$$\Lambda(A_{0,0}, \ldots, A_{0,n}, A_{1,0}, \ldots, A_{1,m})$$

such that for all $a_{0,0}, \ldots, a_{0,n}, a_{1,0}, \ldots, a_{1,m}$ in $K$, if the value

$$\Lambda(a_{0,0}, \ldots, a_{0,n}, a_{1,0}, \ldots, a_{1,m})$$

is not zero, the degrees of the polynomials in the remainder sequence are

$$n, m, m - 1, m - 2, \ldots, 2, 1, 0.$$

**Consequence:** lower bound $\Omega(m^2)$ ops for the GCD algorithm.
A more careful analysis of Euclidean division

Prop.

• If \( \deg(A) = n \) and \( \deg(B) = m \), we can compute the quotient and remainder of \( A \) by \( B \) in at most

\[
2(n - m + 1)(m + 1)
\]

operations.

Proof (sketch).

• We do at most \( n - m + 1 \) reduction steps.
• Each takes at most \( 2(m + 1) \) operations.
Prop.

• The total cost of the gcd algorithm is $O(n^2)$.

Proof. Let $n_i = \deg(A_i)$ be the degrees of the successive remainders.

• Then the cost of computing $A_{i+1}$ is at most

$$2(n_{i-1} - n_i + 1)(n_i + 1) \leq 2(n_{i-1} - n_i + 1)(n + 1).$$

• So the total cost is at most

$$\sum_{i=1}^{N-1} 2(n_{i-1} - n_i + 1)(n + 1) \leq 2(n + 1) \sum_{i=1}^{N-1} (n_{i-1} - n_i + 1)$$

• The sum simplifies into $n_0 - n_{N-1} + (N - 1) \leq 2n$

• So the total cost is at most $2(n + 1)n = O(n^2)$. 

XGCD
Extended gcd

Prop.

- Given $A$ and $B$, one can compute $G = \gcd(A, B)$, as well as Bézout coefficients $U$, $V$ such that

\[ AU + BV = G, \quad \deg(U) < \deg(B), \quad \deg(V) < \deg(A) \]

by a small modification of Euclid’s algorithm.

Special case.

- We say that $A$ and $B$ are coprime if $\gcd(A, B) = 1$.
- In that case the Bézout coefficients satisfy

\[ AU + BV = 1. \]
Example: complex numbers

How to compute with complex numbers

- **complex multiplication** is multiplication modulo \( 1 + x^2 \);
- **complex inversion** is extended gcd with \( 1 + x^2 \).

How?

- suppose \( z = a + bi \)
- compute \( G = \gcd(a + bx, 1 + x^2) \) and the Bézout coefficients \( U(x), V(x) \)
- \( G = 1, \deg(U) < 2 \) and \( \deg(V) < 1 \), so \( U = u_0 + u_1x \) and \( V = v_0 \).
- then \( (u_0 + u_1x)(a + bx) + v_0(1 + x^2) = 1 \)
- evaluating at \( x = i \) gives \( (u_0 + u_1i)(a + bi) = 1 \)
More general example (we’ll get back to it)

Suppose that \( P \) in \( K[x] \) is **irreducible**: it has no divisor, other than constants or itself.

Then for \( A \) in \( K[x] \):

- either \( P \) divides \( A \), and then \( \gcd(A, P) = P \)
- or \( \gcd(A, P) = 1 \).
More general example (we’ll get back to it)

Suppose that $P$ in $K[x]$ is irreducible: it has no divisor, other than constants or itself.

Then for $A$ in $K[x]$:

- either $P$ divides $A$, and then $\gcd(A, P) = P$
- or $\gcd(A, P) = 1$.

Remember that we defined $K[x]/P$ as

- the set of all polynomials of degree less than $\deg(P)$
- with addition and multiplication defined modulo $P$. 
More general example (we’ll get back to it)

Suppose that \( P \) in \( K[x] \) is \textit{irreducible}: it has no divisor, other than constants or itself.

Then for \( A \) in \( K[x] \):

- either \( P \) divides \( A \), and then \( \gcd(A, P) = P \)
- or \( \gcd(A, P) = 1 \).

Remember that we defined \( K[x]/P \) as

- the set of all polynomials of degree less than \( \deg(P) \)
- with addition and multiplication defined \textit{modulo} \( P \).

Now we also have \textit{inversion} modulo \( P \):

- for \( A \neq 0 \) in \( K[x]/P \), \( \gcd(A, P) = 1 \)
- so there exists \( U, V \) with \( AU + PV = 1 \) (as polynomials)
- so \( AU = 1 \) in \( K[x]/P \).
Getting the quotients.

- replace the step

\[ A_{i+1} = A_{i-1} \mod A_i \]

by

\[ Q_i = A_{i-1} \div A_i \]

and

\[ A_{i+1} = A_{i-1} - Q_i A_i \]
Towards the extended Euclidean algorithm

Getting the quotients.

• replace the step

\[ A_{i+1} = A_{i-1} \mod A_i \]

by

\[ Q_i = A_{i-1} \div A_i \]

and

\[ A_{i+1} = A_{i-1} - Q_i A_i \]

• remark that we still have

\[ A_{i+1} = A_{i-1} \mod A_i \]

• the algorithm is still \( O(n^2) \)
The extended Euclidean algorithm

Additionnally to \((A_i)\), we also compute sequence \((U_i)\) and \((V_i)\) with

\[
U_0 = 1, \quad U_1 = 0, \quad U_{i+1} = U_{i-1} - Q_i U_i
\]

and

\[
V_0 = 0, \quad V_1 = 1, \quad V_{i+1} = V_{i-1} - Q_i V_i
\]
The extended Euclidean algorithm

Additionnally to \((A_i)\), we also compute sequence \((U_i)\) and \((V_i)\) with

\[
U_0 = 1, \quad U_1 = 0, \quad U_{i+1} = U_{i-1} - Q_i U_i
\]

and

\[
V_0 = 0, \quad V_1 = 1, \quad V_{i+1} = V_{i-1} - Q_i V_i
\]

Prop.

- For \(0 \leq i \leq N\), we have

\[
A_0 U_i + A_1 V_i = A_i
\]

Proof.

- By induction \((i = 0\) and \(1\) initiate the induction).
The extended Euclidean algorithm

Additionally to \((A_i)\), we also compute sequence \((U_i)\) and \((V_i)\) with

\[
U_0 = 1, \quad U_1 = 0, \quad U_{i+1} = U_{i-1} - Q_i U_i
\]

and

\[
V_0 = 0, \quad V_1 = 1, \quad V_{i+1} = V_{i-1} - Q_i V_i
\]

Prop.

• For \(0 \leq i \leq N\), we have

\[
A_0 U_i + A_1 V_i = A_i
\]

Proof.

• By induction \((i = 0 \text{ and } 1 \text{ initiate the induction})\).

Prop.

• For \(i = N\) (when we get the gcd), we have

\[
A_0 U_N + A_1 V_N = A_N.
\]
Example (same as before)

Over $\mathbb{Z}/3\mathbb{Z}$:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A_i$</th>
<th>$U_i$</th>
<th>$V_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$2x^4 + x^3 + x^2 + 2x + 1$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$x^3 + x^2 + 2x + 1$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$x^2 + 2x + 2$</td>
<td>1</td>
<td>$x + 1$</td>
</tr>
<tr>
<td>3</td>
<td>$2x$</td>
<td>$2x + 1$</td>
<td>$2x^2 + 2$</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>$2x^2 + 2x$</td>
<td>$2x^3 + x^2 + 2$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>$x^3 + x^2 + 2x + 1$</td>
<td>$x^4 + 2x^3 + 2x^2 + x + 2$</td>
</tr>
</tbody>
</table>
Degrees and complexity

Roughly speaking

- the degrees of the $U_i$’s and $V_i$’s **increase**;
- the degrees of the $A_i$’s **decrease**.

Precisely (by an annoying induction)

- $\deg(U_i) = \deg(Q_2) + \cdots + \deg(Q_{i-1})$ \quad $i \geq 2$
- $\deg(V_i) = \deg(Q_1) + \cdots + \deg(Q_{i-1})$ \quad $i \geq 2$

But $\deg(Q_i) = \deg(A_{i-1}) - \deg(A_i)$ so

- $\deg(U_i) = \deg(A_1) - \deg(A_{i-1}) \leq n$ \quad $i \geq 2$
- $\deg(V_i) = \deg(A_0) - \deg(A_{i-1}) \leq n$ \quad $i \geq 2$

**Consequence:** the complexity is still $O(n^2)$. 
An application
With Newton iteration, given polynomials $N(x)$ and $D(x)$, we can expand

$$S(x) = \frac{N(x)}{D(x)} = s_0 + s_1 x + s_2 x^2 + \cdots$$

Assuming you know sufficiently many terms, it is possible to go backwards and recover $N(x)/D(x)$.

**Prop.**

- This is a problem of linear algebra, so it can be solved in theory.
- the XGCD gives a better algorithm.
Sketch of the algorithm

Suppose that:

- we know that \( \deg(N) \leq n \) and \( \deg(D) \leq d \);
- we know \( s_0, \ldots, s_{n+d} \).

We run the extended Euclidean algorithm with input \( A_0 = x^{n+d+1} \) and \( A_1 = s_0 + \cdots + s_{n+d} x^{n+d} \).
Suppose that:

• we know that \( \deg(N) \leq n \) and \( \deg(D) \leq d \);
• we know \( s_0, \ldots, s_{n+d} \).

We run the extended Euclidean algorithm with input \( A_0 = x^{n+d+1} \) and \( A_1 = s_0 + \cdots + s_{n+d}x^{n+d} \).

• For \( i = 0 \), let \( U_0 = 1, V_0 = 0 \).
• For \( i = 1 \), let \( U_1 = 0, V_1 = 1 \).
• For \( i \geq 2 \)
  • \( Q_i = A_{i-1} \text{ div } A_i \)
  • \( A_{i+1} = A_{i-1} - Q_i A_i \)
  • \( U_{i+1} = U_{i-1} - Q_i U_i \)
  • \( V_{i+1} = V_{i-1} - Q_i V_i \)
At each step, we maintain the invariant

\[ U_i x^{n+d+1} + V_i (s_0 + s_1 x + \cdots + s_{n+d} x^{n+d}) = A_i. \]

Moreover:

- the degrees of the \( A_i \) decrease;
- the degrees of the \( V_i \) increase.

Prop.

- Let \( i \) be the first index with \( \deg (A_i) \leq n \).
- Then \( \deg (V_i) = n + d + 1 - \deg (A_{i-1}) \leq d \).
- Hence, \( A_i / V_i = N / D \).
Problem: find the next term.

\[ U : \quad 1, 1, 1, 1, 1, 1, 1, 1 \]

\[ V : \quad 0, 1, 1, 2, 3, 5, 8, 13 \]

\[ W : \quad 12, 134, 222, 21, -3898, -40039, -347154, -2929918, -24657854 \]

Answer: 1, 21 and −207605083.

How? The sequences \( U, V, W \) satisfy linear recurrences with constant coefficients:

\[ U_{n+1} = U_n, \]

\[ V_{n+2} = V_{n+1} + V_n, \]

\[ W_{n+4} = 12W_{n+3} - 33W_{n+2} + 22W_{n+1} + 19W_n. \]
Generating series

Given a sequence \((s_0, s_1, \ldots)\), we can construct the series

\[
S = \sum_{i \geq 0} s_i x^i.
\]

This is the **generating series** of \(u\).

- The properties of \(u\) (recurrence) translate to properties of \(S\).

**Simple case**

- \(s_n = 2^n\) (equivalently, \(s_0 = 1\) and \(s_{n+1} - 2s_n = 0\))
- generating series

\[
S = \sum_{i} 2^i x^i = \frac{1}{1 - 2x}
\]
The generating series of the previous example is **rational**.

**Prop.**

- The generating series $S$ is rational:

\[
S = \frac{N(x)}{D(x)}, \quad \text{with}
\]

\[
D(x) = 1 + a_{k-1}x + \cdots + a_1x^{k-1} + a_0x^k \quad \text{and} \quad \deg(N) < \deg(D)
\]

if and only if the sequence $(s_n)_{n \geq 0}$ satisfies the recurrence

\[
s_{n+k} + a_{k-1}s_{n+k-1} + \cdots + a_1s_{n+1} + a_0s_n = 0, \quad a_0 \neq 0
\]

**rational series $\iff$ recurrence with constant coefficients**
Proof on an example

We check this for recurrences of order 2, with

\[ s_0 = \alpha, \quad s_1 = \beta, \quad s_{n+2} + a s_{n+1} + b s_n = 0 \]

and

\[ S = \sum_{i \geq 0} s_i x^i. \]

1. Multiply the recurrence relation by \( x^{n+2} \):

\[ s_{n+2} x^{n+2} + a s_{n+1} x^{n+2} + b s_n x^{n+2} = 0. \]

2. Sum, for \( n \geq 0 \):

\[ S - (\alpha + \beta x) + ax(S - \alpha) + bx^2 S = 0. \]

3. Rearrange

\[ S = \frac{\alpha + (\beta + \alpha a)x}{1 + ax + bx^2}. \]
Suppose that you know that a sequence \((s_n)_{n \geq 0}\) satisfies a recurrence of order \(k\):

- set \(n = k - 1\), \(d = k\)
- you need \(s_0, \ldots, s_{n+d}\), so up to \(s_{2k-1}\).
- you apply the Extended Euclidean Algorithm to

\[
A_0 = x^{2k}, A_1 = s_0 + \cdots + s_{2k-1}x^{2k-1}
\]

- you stop at the first \(i\) with \(\text{deg}(A_i) \leq k - 1\).
Example: Fibonacci numbers

Take \((s_n)_{n \geq 0} = (1, 1, 2, 3, 5, 8, \ldots)\), and suppose we know it satisfies a recurrence of order \(k = 2\). We apply the XGCD algorithm to \(A_0 = x^4\) and \(A_1 = 1 + x + 2x^2 + 3x^3\).

<table>
<thead>
<tr>
<th>(i)</th>
<th>(A_i)</th>
<th>(U_i)</th>
<th>(V_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x^4)</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>(1 + x + 2x^2 + 3x^3)</td>
<td>?</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>(\frac{1}{9}(2 - x + x^2))</td>
<td>?</td>
<td>(\frac{1}{9}(2 - 3x))</td>
</tr>
<tr>
<td>3</td>
<td>(-9)</td>
<td>?</td>
<td>(9(-1 + x + x^2))</td>
</tr>
</tbody>
</table>

So we find that

\[
1 + x + 2x^2 + 3x^3 + 5x^4 + \cdots = \frac{-9}{9(-1 + x + x^2)} = \frac{1}{1 - x - x^2}
\]

and the recurrence

\[
s_{n+2} - s_{n+1} - s_n = 0.
\]
The fast algorithm (sketch)
Matrices in Euclid’s algorithm

Notation as before:

- \( A_0, A_1, \ldots \) the successives remainders
- \( Q_1, Q_2, \ldots \) the quotients.

We can write the transformation \((A_{i-1}, A_i) \rightarrow (A_i, A_{i+1})\) in a matrix way:

\[
\begin{bmatrix}
  A_i \\
  A_{i+1}
\end{bmatrix} = \begin{bmatrix}
  0 & 1 \\
  1 & -Q_i
\end{bmatrix} \begin{bmatrix}
  A_{i-1} \\
  A_i
\end{bmatrix}.
\]

Multiplying matrices, we see that for all \( i \), we can write

\[
\begin{bmatrix}
  A_i \\
  A_{i+1}
\end{bmatrix} = R_i \begin{bmatrix}
  A_0 \\
  A_1
\end{bmatrix},
\]

\[
R_i = \begin{bmatrix}
  0 & 1 \\
  1 & -Q_i
\end{bmatrix} \cdots \begin{bmatrix}
  0 & 1 \\
  1 & -Q_1
\end{bmatrix}
\]

Main idea

• to compute the (x)gcd, it is too costly to compute all remainders;
• we are going to do big steps, to skip a lot of them.

Half-gcd

• suppose $\deg(A_0) = n$, $\deg(A_1) = n - 1$, ..., $\deg(A_j) = n - j$, ... and $A_n = \text{constant}$, $n$ is a power of 2
• the half-gcd algorithm computes the matrix $R_{n/2}$:

$$
\begin{bmatrix}
A_{n/2} \\
A_{n/2+1}
\end{bmatrix}
= R_{n/2}
\begin{bmatrix}
A_0 \\
A_1
\end{bmatrix}.
$$

• the GCD matrix is the matrix $R_{n-1}$

$$
\begin{bmatrix}
A_{n-1} \\
A_n
\end{bmatrix}
= R_{n-1}
\begin{bmatrix}
A_0 \\
A_1
\end{bmatrix}.
$$

If we find it, we can get the Bézout coefficients (second row).
Recursive algorithm for computing the GCD matrix.

\[
gcd\text{-matrix}(A, B)
\]

- \textbf{if} \, \text{deg}(B) = 0, \textbf{return} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
- \text{let} \, R = hgcd\text{-matrix}(A, B)
- \text{let} \begin{bmatrix} U \\ V \end{bmatrix} = R \begin{bmatrix} A \\ B \end{bmatrix}
- \textbf{return} \, gcd\text{-matrix}(U, V)R
Notation

• Let $G(n)$ be the cost of \texttt{gcd\_matrix} in degree $n$
• Let $H(n)$ be the cost of \texttt{hgcd\_matrix} in degree $n$.

Fact: $H(n) = O(M(n) \log(n))$.

Recurrence

$$G(n) = G(n/2) + O(M(n) \log(n))$$

Solving it gives

$$G(n) = O(M(n) \log(n))$$
Main idea of the HGCD

In **Euclidean division**

- when you divide two polynomials (of high degree),
- the **remainder** does depend on all coefficients
- but the **quotient** depends only on the high-degree ones.

You can see it:

- in the **slow algorithm**, you construct $Q$ using the high-degree terms only
- in the **fast algorithm**, you construct $Q$ by a truncated series product.
Truncating the polynomials

Given

\[ A_0 = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_0 \]
\[ A_1 = a'_n X^{n-1} + a'_{n-2} X^{n-2} + \cdots + a'_0 \]

define, for some \( m \leq n \),

\[ \tilde{A}_0 = A_0 \text{ div } X^{n-m} \]
\[ = a_n X^m + a_{n-1} X^{m-1} + \cdots + a_{n-m} \]
\[ \tilde{A}_1 = A_1 \text{ div } X^{n-m} \]
\[ = a'_n X^{m-1} + a'_{n-2} X^{m-2} + \cdots + a'_{n-m} \]

Then \( R_{m/2} = \text{hgcd\_matrix}(\tilde{A}_0, \tilde{A}_1) \)
The algorithm

\texttt{hgcd\_matrix}(A_0, A_1) \quad \text{deg}(A_0) = n

- if \( n = 2 \), return \( \begin{bmatrix} 0 & 1 \\ 1 & -Q \end{bmatrix} \), \( Q \) = quotient of \( A_0 \) by \( A_1 \)
The algorithm

\[ \text{hgcd\_matrix}(A_0, A_1) \quad \text{deg}(A_0) = n \]

- if \( n = 2 \), return \( \begin{bmatrix} 0 & 1 \\ 1 & -Q \end{bmatrix} \), \( Q \) = quotient of \( A_0 \) by \( A_1 \)
- let \( \tilde{A}_0 = A_0 \text{ div } X^{n/2} \), \( \tilde{A}_1 = A_1 \text{ div } X^{n/2} \)  \( \text{deg}(\tilde{A}_0) = n/2 \)
The algorithm

\texttt{hgcd\_matrix}(A_0, A_1) \quad \text{deg}(A_0) = n

- if \( n = 2 \), return \( \begin{bmatrix} 0 & 1 \\ 1 & -Q \end{bmatrix} \), \( Q \) = quotient of \( A_0 \) by \( A_1 \)
- let \( \tilde{A}_0 = A_0 \div X^{n/2} \), \( \tilde{A}_1 = A_1 \div X^{n/2} \) \quad \text{deg}(\tilde{A}_0) = n/2
- compute \( M = \text{hgcd}\_\text{matrix}(\tilde{A}_0, \tilde{A}_1) \) \quad \( M = R_{n/4} \)
The algorithm

\[
\text{hgcd\_matrix}(A_0, A_1) \quad \text{deg}(A_0) = n
\]

- if \( n = 2 \), return \[
\begin{bmatrix}
0 & 1 \\
1 & -Q
\end{bmatrix}
\]
  \( Q \) = quotient of \( A_0 \) by \( A_1 \)

- let \( \tilde{A}_0 = A_0 \ \text{div} \ X^{n/2} \), \( \tilde{A}_1 = A_1 \ \text{div} \ X^{n/2} \) \( \text{deg}(\tilde{A}_0) = n/2 \)

- compute \( M = \text{hgcd\_matrix}(\tilde{A}_0, \tilde{A}_1) \) \( M = R_{n/4} \)

- let \[
\begin{bmatrix}
A_{n/4} \\
A_{n/4+1}
\end{bmatrix}
= M \begin{bmatrix}
A_0 \\
A_1
\end{bmatrix}
\]
  \( \text{deg}(A_{n/4}) = 3n/4 \)
The algorithm

\texttt{hgcd\_matrix}(A_0, A_1) \quad \text{deg}(A_0) = n

- if \( n = 2 \), return \( \begin{bmatrix} 0 & 1 \\ 1 & -Q \end{bmatrix} \), \( Q = \)quotient of \( A_0 \) by \( A_1 \)
- let \( \tilde{A}_0 = A_0 \div X^{n/2} \), \( \tilde{A}_1 = A_1 \div X^{n/2} \) \quad \text{deg}(\tilde{A}_0) = n/2
- compute \( M = \text{hgcd\_matrix}(\tilde{A}_0, \tilde{A}_1) \) \quad \( M = R_{n/4} \)
- let \( \begin{bmatrix} A_{n/4} \\ A_{n/4+1} \end{bmatrix} = M \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} \) \quad \text{deg}(A_{n/4}) = 3n/4
- let \( \tilde{A}_{n/4} = A_{n/4} \div X^{n/4} \) and \( \tilde{A}_{n/4+1} = A_{n/4+1} \div X^{n/4} \) \quad \text{deg}(\tilde{A}_{n/4}) = n/2
The algorithm

```
hgcd_matrix(A_0, A_1)
```

- if \( n = 2 \), return \[
\begin{bmatrix}
0 & 1 \\
1 & -Q
\end{bmatrix}
\]
where \( Q \) is the quotient of \( A_0 \) by \( A_1 \)
- \( \text{deg}(A_0) = n \)
- let \( \tilde{A}_0 = A_0 \text{ div } X^{n/2} \), \( \tilde{A}_1 = A_1 \text{ div } X^{n/2} \)
- \( \text{deg}(\tilde{A}_0) = n/2 \)
- \( \text{deg}(A_{n/4}) = 3n/4 \)

```
\begin{bmatrix}
A_{n/4} \\
A_{n/4+1}
\end{bmatrix} = M \begin{bmatrix}
A_0 \\
A_1
\end{bmatrix}
```

- compute \( M = hgcd_matrix(\tilde{A}_0, \tilde{A}_1) \)
- \( M = R_{n/4} \)
- let \( \tilde{A}_{n/4} = A_{n/4} \text{ div } X^{n/4} \)
- and \( \tilde{A}_{n/4+1} = A_{n/4+1} \text{ div } X^{n/4} \)
- \( \text{deg}(\tilde{A}_{n/4}) = n/2 \)
- \( \text{deg}(\tilde{A}_{n/4+1}) = n/2 \)
- compute \( N = hgcd_matrix(\tilde{A}_{n/4}, \tilde{A}_{n/4+1}) \)
- \( \text{return } NM \)
The algorithm does:

- 2 recursive calls in degree $n/2$
- some products of $2 \times 2$ matrices in degree $\leq n$

**Recurrence**

$$H(n) = 2H(n/2) + O(M(n))$$

Solving it gives

$$H(n) \in O(M(n) \log(n)).$$