Lecture 19: Matrix Multiplication & Fast Modular Composition

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

March 24, 2021

Overview

- Fast Linear Algebra
- Matrix Multiplication
- Fast Modular Composition
- Conclusion

Importance of Fast Linear Algebra

- So far we have discussed how to get better algorithms for:
  1. integer & polynomial multiplication
  2. integer & polynomial division
  3. polynomial factoring
  4. polynomial interpolation
  5. integer & polynomial GCD
- Also saw algorithms for algebraic geometry problems and invariant theoretic problems

Importance of Fast Linear Algebra

- So far we have discussed how to get better algorithms for:
  1. integer & polynomial multiplication
  2. integer & polynomial division
  3. polynomial factoring
  4. polynomial interpolation
  5. integer & polynomial GCD
- Also saw algorithms for algebraic geometry problems and invariant theoretic problems
- In this part of the course, we turn our attention to performing fast linear algebra
  1. evaluation of determinant
  2. matrix multiplication
  3. linear system solving
  4. find rank, basis for null space, Jordan form, etc

Importance of Fast Linear Algebra

- So far we have discussed how to get better algorithms for:
  1. integer & polynomial multiplication
  2. integer & polynomial division
  3. polynomial factoring
  4. polynomial interpolation
  5. integer & polynomial GCD
- Also saw algorithms for algebraic geometry problems and invariant theoretic problems
- In this part of the course, we turn our attention to performing fast linear algebra
  1. evaluation of determinant
  2. matrix multiplication
  3. linear system solving
  4. find rank, basis for null space, Jordan form, etc
- These tasks are pervasive in symbolic computation (and in real life!)

Matrix Multiplication

square matrices

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$

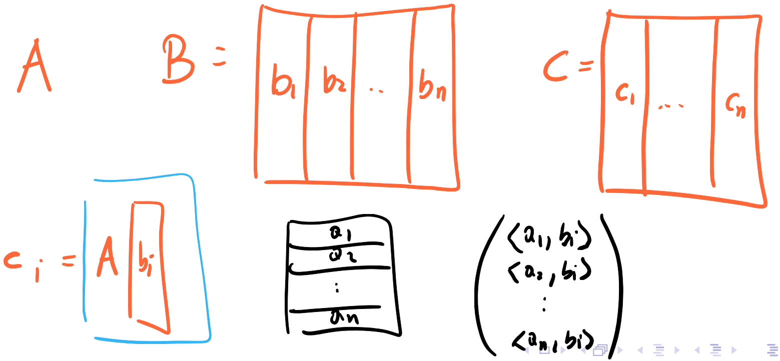
multiplying rectangular matrices $A \in \mathbb{F}^{m \times n}$, $B \in \mathbb{F}^{n \times p}$

is also quite important in theory and in practice!

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

  Compute n matrix vector multiplications.



Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

  Compute n matrix vector multiplications.
- Running time: $O(n^3)$

  Can we do better?

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

  Compute n matrix vector multiplications.
- Running time: $O(n^3)$

  Can we do better?
- Strassen 1969: YES!
- Idea: divide matrix into blocks, and *reduce number of multiplications* needed!

  Similar in spirit as Karatsuba's algorithm for polynomial multiplication!

Strassen's Algorithm

- Suppose that $n = 2^k$ (doesn't affect our asymptotic running time)
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

8 multiplications

$T(n) \leq 8 \cdot T(n/2) + c \cdot \left(\frac{n}{2}\right)^2$

Master thm

$T(n) = O(n^3)$

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

takes us $O((\frac{n}{2})^2)$ time to compute them.

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4 B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1 T_1,\ P_6 = S_2 T_2,\ P_7 = S_3 T_3$$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$

$$A_{11}B_{11} + (A_{12} + A_{11} - A_{21} - A_{22})B_{22} + (A_{21} + A_{22})(B_{12} - B_{11})$$
$$+ (A_{21} + A_{22} - A_{11})(B_{22} - B_{12} + B_{11})$$

$$A_{12}B_{22} + A_{11}B_{12}$$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4 B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1 T_1,\ P_6 = S_2 T_2,\ P_7 = S_3 T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22} = P_1 + P_5 + P_6 + P_7$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22} = P_1 + P_5 + P_6 + P_7$
- Correctness follows from the computations

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions — S_i, T_i's
  2. 7 multiplications — P_i's
  3. 10 additions — C_{ij}'s

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions *set up multiplications* — S_i, T_i's
  2. 7 multiplications — P_i's
  3. 10 additions *put everything together* — C_{ij}'s
- Recurrence:

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions — S_i, T_i's
  2. 7 multiplications — P_i's
  3. 10 additions — C_{ij}'s
- Recurrence:

$k = \log_2 n$

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

$$MM(2^k) \leq 7 \cdot MM(2^{k-1}) + 18 \cdot c \cdot 2^{2k-2}$$

$$MM(2^k) \leq 7^k \cdot 8 + 18 \cdot c \cdot \sum_{j=1}^{k-1} 4^j = O(7^k) = O(n^{\log_2 7})$$

$$\sum_{j=1}^{k-1} 4^j \to \frac{4^k - 1}{3}$$

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions — S_i, T_i's
  2. 7 multiplications — P_i's
  3. 10 additions — C_{ij}'s
- Recurrence:

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

$$MM(2^k) \leq 7 \cdot MM(2^{k-1}) + 18 \cdot c \cdot 2^{2k-2}$$

- Could also use Master theorem to get $MM(n) = O(n^{\log 7}) \approx O(n^{2.807})$

Matrix Multiplication Exponent

- We can define ω (or ω_{mult}) as the *matrix multiplication exponent*.
  1. If an algorithm for $n \times n$ matrix multiplication has running time $O(n^{\alpha})$, then $\omega \leq \alpha$.
  2. For any $\varepsilon > 0$, there is an algorithm for $n \times n$ matrix multiplication running in time $O(n^{\omega+\varepsilon})$

ω is the least (infimum) of all feasible exponents for matrix multiplication

We know: $\omega \geq 2$

Open question: is $\omega = 2$?

Matrix Multiplication Exponent

- We can define ω (or ω_{mult}) as the *matrix multiplication exponent*.
  1. If an algorithm for $n \times n$ matrix multiplication has running time $O(n^{\alpha})$, then $\omega \leq \alpha$.
  2. For any $\varepsilon > 0$, there is an algorithm for $n \times n$ matrix multiplication running in time $O(n^{\omega+\varepsilon})$
- As we will see later in the course, ω is a fundamental constant in computer science!

Matrix Multiplication Exponent

- We can define ω (or ω_{mult}) as the *matrix multiplication exponent*.
  1. If an algorithm for $n \times n$ matrix multiplication has running time $O(n^{\alpha})$, then $\omega \leq \alpha$.
  2. For any $\varepsilon > 0$, there is an algorithm for $n \times n$ matrix multiplication running in time $O(n^{\omega+\varepsilon})$
- As we will see later in the course, ω is a fundamental constant in computer science!
- Currently we know $\omega < 2.376$

Open Question

What is the right value of ω?

Historical Remarks

- Strassen's work is not only important because it gives a faster matrix multiplication algorithm, but because it startled the community that the trivial cubic algorithm could be improved!

Historical Remarks

- Strassen's work is not only important because it gives a faster matrix multiplication algorithm, but because it startled the community that the trivial cubic algorithm could be improved!
- Motivated work on better algorithms for all other linear algebraic problems

Historical Remarks

- Strassen's work is not only important because it gives a faster matrix multiplication algorithm, but because it startled the community that the trivial cubic algorithm could be improved!
- Motivated work on better algorithms for all other linear algebraic problems
- introduced complexity of computation of *bilinear functions* and the study of complexity of tensor decompositions

- Fast Linear Algebra
- Matrix Multiplication
- Fast Modular Composition
- Conclusion

Modular Composition

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that

$$\deg(g), \deg(h) < \deg(f) = n.$$

- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f

$$g(x) = x^2 + 1 \qquad h(x) = x^3 + 1$$

$$g(h) = (x^3+1)^2 + 1 = x^6 + 2x^3 + 2$$

$$f = x^4 \qquad g(h) \equiv 2x^3 + 2 \mod f$$

Modular Composition

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that

$$\deg(g), \deg(h) < \deg(f) = n.$$

- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f
- If we only use Horner's rule, this can be done with $O(n \cdot M(n))$ operations in $\mathbb{F}$

$M(n) :=$ time it takes to multiply two polynomials of degree $\leq n$.

Modular Composition

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that

$$\deg(g), \deg(h) < \deg(f) = n.$$

- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f
- If we only use Horner's rule, this can be done with $O(n \cdot M(n))$ operations in $\mathbb{F}$
- We can do much better by combining: *fast polynomial arithmetic* and *fast matrix arithmetic*

Fast Algorithm

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that

$$\deg(g), \deg(h) < \deg(f) = n = m^2$$

- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f

Fast Algorithm

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that
$$\deg(g), \deg(h) < \deg(f) = n = m^2$$
- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f
- Let $g(x) = \sum_{i=0}^{m-1} g_i(x) \cdot x^{mi}$, where $\deg(g_i) < m$

$n = 4 = 2^2$

$g(x) = 2x^3 + 3x^2 + 4x + 1$

high degree: $x^2 \cdot (2x + 3)$, where $2x + 3 = g_1$

lower: $4x + 1 = g_0$

$\deg(g_0) = 1$

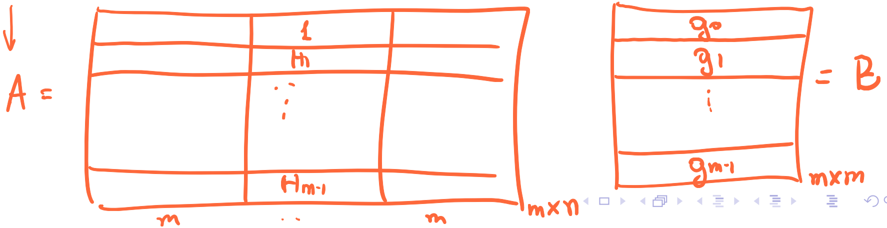
$\deg(g_1) = 1$

Fast Algorithm

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that
$$\deg(g), \deg(h) < \deg(f) = n = m^2$$
- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f
- Let $g(x) = \sum_{i=0}^{m-1} g_i(x) \cdot x^{mi}$, where $\deg(g_i) < m$
- Compute $H_i(x) := h^i \mod f$ for $1 \leq i \leq m$

Fast Algorithm

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that
$$\deg(g), \deg(h) < \deg(f) = n = m^2$$
- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f
- Let $g(x) = \sum_{i=0}^{m-1} g_i(x) \cdot x^{mi}$, where $\deg(g_i) < m$
- Compute $H_i(x) := h^i \mod f$ for $1 \leq i \leq m$
- Let $A \in \mathbb{F}^{m \times n}$ be the matrix whose rows are the coefficients of $1, H_1, H_2, \ldots, H_{m-1}$, and $B \in \mathbb{F}^{m \times m}$ be the matrix whose rows are the coefficients of $g_0, \ldots, g_{m-1}$.
- Compute $C = BA$ by doing m matrix multiplications of size $m \times m$



Fast Algorithm

- **Input:** polynomials $f, g, h \in \mathbb{F}[x]$ such that

$$\deg(g), \deg(h) < \deg(f) = n = m^2$$

- **Output:** $g(h) \mod f$ remainder of composition $g(h)$ modulo f
- Let $g(x) = \sum_{i=0}^{m-1} g_i(x) \cdot x^{mi}$, where $\deg(g_i) < m$
- Compute $H_i(x) := h^i \mod f$ for $1 \leq i \leq m$
- Let $A \in \mathbb{F}^{m \times n}$ be the matrix whose rows are the coefficients of $1, H_1, H_2, \ldots, H_{m-1}$, and $B \in \mathbb{F}^{m \times m}$ be the matrix whose rows are the coefficients of $g_0, \ldots, g_{m-1}$.
  Compute $C = BA$ by doing m matrix multiplications of size $m \times m$
- For $0 \leq i < m$, let $c_i(x) \in \mathbb{F}[x]$ be the polynomial corresponding to i^{th} row of C. Compute (using Horner's rule)

$$r(x) = \sum_{i=0}^{m-1} c_i \cdot (H_m)^i \mod f$$

and return r

Fast Algorithm in Picture

trying to beat $n \cdot M(n)$ → $m^2 \cdot M(m^2) \geq m^4$

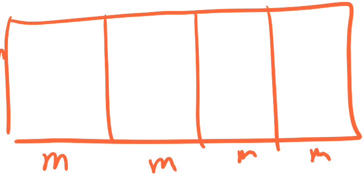
$H_i(x) = h^i \bmod f$ $0 \le i \le m$

(m+1 products of polynomials of degree $\le n$)

$(1+m) \cdot M(n)$

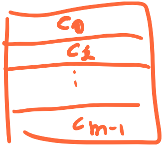
some savings here

$A :=$



BA $m \cdot MM(m)$

$C = BA$



$$r(x) = \sum_{i=0}^{m-1} c_i (H_m)^i \bmod f$$

Horner's rule $m \cdot M(n)$ to compute $r(x) \bmod f$

Total running time: $2(m+1)M(n) + m \cdot MM(m) = O(m M(m^2) + m^{3.8})$

$$g(x) = \sum_{i=0}^{m-1} g_i(x)\,(x^m)^i$$

$$g_i(x) = \sum_{j=0}^{m-1} g_{ij}\cdot x^j$$

$$H_i(x) = h(x)^i \bmod f \qquad 0 \le i \le m \quad (*)$$

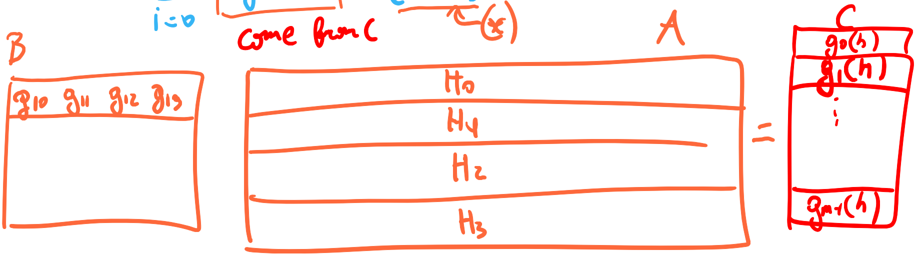
h^j

$g_{ij}\, H_j \bmod f$

$$g(h) = \sum_{i=0}^{m-1} g_i(h)\cdot (h^m)^i \equiv \bmod f$$

$$= \sum_{i=0}^{m-1} g_i(h)\cdot (H_m)^i$$

come from C ← (*)



Example of Fast Algorithm

- $f = x^4 - 1, g = x^3 + 1, h = x^2 + 1 \in \mathbb{F}_3[x]$

$m = 2$

$n = 4$

$$g(x) = 1 \cdot \underbrace{1}_{g_0(x)} + x^2 \cdot \underbrace{(x)}_{g_1(x)}$$

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}}_{C}$$

1	x	x^2	x^3
1	0	0	0
1	0	1	0

$h^0 \bmod f$

$h^1 \bmod f$

$h^2 \bmod f$

A comes from here

$g_0(h) = 1$ ←

$g_1(h) = 1 + x^2$ ←

1	x	x^2	x^3
1	0	0	0
1	0	1	0

$$r(x) = g_0(h)\cdot(H_2)^0 + g_1(h)\cdot(H_2)^1 \mod f$$

$$H_1 = h = x^2+1$$

$$H_2 = h^2 = x^4+2x^2+1 \equiv 2(x^2+1)$$

$$r(x) = 2(x^2+1)\cdot(1+x^2) + 1 \mod f$$

$$= 2\cdot 2(x^2+1) + 1$$

$$= 4x^2+5 \mod f$$ ← return.

Horner's rule

$$P(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_0$$

$$((((a_d x + a_{d-1}) x + a_{d-2}) x + a_{d-3}) x + \cdots$$

Conclusion

- Today we learned about Strassen's fast matrix multiplication algorithm
- Application to fast modular composition
- Next lectures: exploration of matrix multiplication exponent and fast linear algebra in "black box" model

References I

von zur Gathen, J. and Gerhard, J. 2013.
Modern Computer Algebra
Cambridge University Press

Chapter 12