C-recursive sequences: Nth term. Application to power series composition

Alin Bostan

MPRI

COMPALG

8 October 2025

Exercise 1

Let $\mathbb{K}$ be a field of characteristic zero. Consider $F \in \mathbb{K}[[x]]$ with $F(0) = 1$.

(a) What is the complexity of computing $\sqrt{F}$, by using $\sqrt{F} = \exp(\frac{1}{2}\log F)$?

(b) Describe a Newton iteration that directly computes $\sqrt{F}$, without appealing to successive logarithm and exponential computations.

(c) Estimate the complexity of the algorithm in (b).

Exercise 1

Let $\mathbb{K}$ be a field of characteristic zero. Consider $F \in \mathbb{K}[[x]]$ with $F(0) = 1$.

(a) What is the complexity of computing $\sqrt{F}$, by using $\sqrt{F} = \exp(\frac{1}{2}\log F)$?

(b) Describe a Newton iteration that directly computes $\sqrt{F}$, without appealing to successive logarithm and exponential computations.

(c) Estimate the complexity of the algorithm in (b).

(a) $O(\mathsf{M}(N))$ for computing the first N terms

Exercise 1

Let $\mathbb{K}$ be a field of characteristic zero. Consider $F \in \mathbb{K}[[x]]$ with $F(0) = 1$.

(a) What is the complexity of computing $\sqrt{F}$, by using $\sqrt{F} = \exp(\frac{1}{2}\log F)$?

(b) Describe a Newton iteration that directly computes $\sqrt{F}$, without appealing to successive logarithm and exponential computations.

(c) Estimate the complexity of the algorithm in (b).

(a) $O(\mathsf{M}(N))$ for computing the first N terms

(b) $\Phi(F, G) := G^2 - F$ to get $G = \sqrt{F}$ provides $\mathcal{N}(G) = (G + F/G)/2$.
If $G = \sqrt{F} + O(X^n)$, then $\exists H,\ F = G^2(1 + X^n H)$, so $\mathcal{N}(G) = G + \frac{1}{2}GX^n H$.
Next, $\sqrt{F} = G(1 + \frac{1}{2}X^n H + O(X^{2n}))$, so $\mathcal{N}(G) = \sqrt{F} + O(X^{2n})$.

Computationally: given $F = T + O(X^N)$, recursively compute $\sqrt{F} = U + O(X^{\lceil N/2 \rceil})$, then return $U + \mathrm{rem}(T/U, X^N)/2$.

Exercise 1

Let $\mathbb{K}$ be a field of characteristic zero. Consider $F \in \mathbb{K}[[x]]$ with $F(0) = 1$.

(a) What is the complexity of computing $\sqrt{F}$, by using $\sqrt{F} = \exp(\frac{1}{2}\log F)$?

(b) Describe a Newton iteration that directly computes $\sqrt{F}$, without appealing to successive logarithm and exponential computations.

(c) Estimate the complexity of the algorithm in (b).

(a) $O(\mathsf{M}(N))$ for computing the first N terms

(b) $\Phi(F, G) := G^2 - F$ to get $G = \sqrt{F}$ provides $\mathcal{N}(G) = (G + F/G)/2$.
If $G = \sqrt{F} + O(X^n)$, then $\exists H,\ F = G^2(1 + X^n H)$, so $\mathcal{N}(G) = G + \frac{1}{2}GX^n H$.
Next, $\sqrt{F} = G(1 + \frac{1}{2}X^n H + O(X^{2n}))$, so $\mathcal{N}(G) = \sqrt{F} + O(X^{2n})$.

Computationally: given $F = T + O(X^N)$, recursively compute $\sqrt{F} = U + O(X^{\lceil N/2 \rceil})$, then return $U + \mathrm{rem}(T/U, X^N)/2$.

(c) $C(N) \leq C(N/2) + O(\mathsf{M}(N))$ leads to $C(N) = O(\mathsf{M}(N))$.

Exercise 2

Let $\mathbb{K}$ be as before. Let f and g in $\mathbb{K}[x,y]$ have degrees $\leq (d_x, d_y)$ in (x,y).

(a) Show that it is possible to compute the product $h = fg$ using $O(\mathsf{M}(d_x d_y))$ arithmetic operations in $\mathbb{K}$.
Hint: Use substitution $x \leftarrow y^{2d_y+1}$ to reduce to univariate computations.

(b) Improve this result by proposing an evaluation-interpolation scheme allowing the computation of h in $O(d_x\,\mathsf{M}(d_y) + d_y\,\mathsf{M}(d_x))$ ops. in $\mathbb{K}$.

Exercise 2

Let $\mathbb{K}$ be as before. Let f and g in $\mathbb{K}[x,y]$ have degrees $\leq (d_x, d_y)$ in (x,y).

(a) Show that it is possible to compute the product $h = fg$ using $O(\mathsf{M}(d_x d_y))$ arithmetic operations in $\mathbb{K}$.
Hint: Use substitution $x \leftarrow y^{2d_y+1}$ to reduce to univariate computations.

(b) Improve this result by proposing an evaluation-interpolation scheme allowing the computation of h in $O(d_x\,\mathsf{M}(d_y) + d_y\,\mathsf{M}(d_x))$ ops. in $\mathbb{K}$.

(a) ▷ Write $h(x,y) = h_0(y) + xh_1(y) + \cdots + x^{2d_x} h_{2d_x}(y)$ with $\deg_y h_i \leq 2d_y$.
Observe that in the specialization $h(y^{2d_y+1}, y)$, the terms $y^{(2d_y+1)i} h_i(y)$ have distinct monomial supports.

▷ So one gets $h(x,y)$ from $h(y^{2d_y+1}, y)$ in no arithmetic operation.

▷ Similarly, $f(y^{2d_y+1}, y)$ is obtained from $f(x,y)$ with no calculation.
The same holds for g.

▷ One only needs to compute $h(y^{2d_y+1}, y) = f(y^{2d_y+1}, y) \times g(y^{2d_y+1}, y)$, which requires $O(\mathsf{M}(d_x d_y))$ ops. in $\mathbb{K}$. □

Exercise 2

Let $\mathbb{K}$ be as before. Let f and g in $\mathbb{K}[x,y]$ have degrees $\leq (d_x, d_y)$ in (x,y).

(a) Show that it is possible to compute the product $h = fg$ using $O(\mathsf{M}(d_x d_y))$ arithmetic operations in $\mathbb{K}$.
Hint: Use substitution $x \leftarrow y^{2d_y+1}$ to reduce to univariate computations.

(b) Improve this result by proposing an evaluation-interpolation scheme allowing the computation of h in $O(d_x\,\mathsf{M}(d_y) + d_y\,\mathsf{M}(d_x))$ ops. in $\mathbb{K}$.

(b)
- $\deg_y h_i \leq 2d_y$ so $h_i(y)$ can be interpolated from $2d_y+1$ points in $\mathbb{K}$.
- Use $(1, q, q^2, \ldots, q^{2d_y})$ and get evaluations of all $h_i(y)$ simultaneously.
- Write $f(x,y) = f_0(y) + x f_1(y) + \cdots + x^{d_x} f_{d_x}(y)$ with $\deg_y f_i \leq d_y$.
- Write $g(x,y) = g_0(y) + x g_1(y) + \cdots + x^{d_x} g_{d_x}(y)$ with $\deg_y g_i \leq d_y$.
  - For $0 \leq i \leq d_x$, evaluate $f_i(y)$ and $g_i(y)$ at $(q^j)_{0\leq j\leq 2d_y}$. $O(d_x \mathsf{M}(d_y))$
  - For $0 \leq j \leq 2d_y$, do:
    - compute $f(x,q^j) = \sum_{i=0}^{d_x} x^i f_i(q^j)$;
    - compute $g(x,q^j) = \sum_{i=0}^{d_x} x^i g_i(q^j)$;
    - compute $h(x,q^j) = f(x,q^j) \times g(x,q^j)$. $O(d_y \mathsf{M}(d_x))$
  - For $0 \leq i \leq 2d_x$, interpolate $(h_i(q^j))_{0\leq j\leq 2d_y}$ to get $h_i(y)$. $O(d_x \mathsf{M}(d_y))$
  - Return $h(x,y) = \sum_{i=0}^{2d_x} x^i h_i(y)$. □

COMPUTING TERMS OF RECURRENT SEQUENCES

Goal, motivation, examples, main results

Based on [B., Mori, SOSA 2021] and [Kinoshita, Li, FOCS 2024]

A Simple and Fast Algorithm for Computing the N-th Term of a Linearly Recurrent Sequence

Alin Bostan* and Ryuhei Mori†
*Inria, Palaiseau, France and †Tokyo Institute of Technology, Japan
alin.bostan@inria.fr, mori@c.titech.ac.jp

Abstract

We present a simple and fast algorithm for computing the N-th term of a given linearly recurrent sequence. Our new algorithm uses $O(\mathsf{M}(d)\log N)$ arithmetic operations, where d is the order of the recurrence, and $\mathsf{M}(d)$ denotes the number of arithmetic operations for computing the product of two polynomials of degree d. The state-of-the-art algorithm, due to Fiduccia (1985), has the same arithmetic complexity up to a constant factor. Our algorithm is simpler, faster and obtained by a totally different method. We also discuss several algorithmic applications, notably to polynomial modular exponentiation and powering of matrices.

Keywords: Algebraic Algorithms; Computational Complexity; Linearly Recurrent Sequence; Rational Power Series; Fast Fourier Transform

1 Introduction

1.1 General context Computing efficiently selected terms in sequences is a basic and fundamental algorithmic problem, whose applications are ubiquitous, for instance in theoretical computer science [65, 49], algebraic complexity theory [59, 73], computer algebra [28, 71, 48], cryptography [31, 32, 29, 33], algorithmic number theory [72, 1], effective algebraic geometry [12, 36], numerical analysis [52, 51] and computational biology [56].

In simple terms, the problem can be formulated as follows:

> Given a sequence $(u_n)_{n\geq 0}$ in an effective ring[0] R, and given a positive integer $N \in \mathbb{N}$, compute the term u_N as fast as possible.

Here, the input $(u_n)_{n\geq 0} \in R^{\mathbb{N}}$ is assumed to be a recurrent sequence, specified by a data structure consisting in a *recurrence relation* and sufficiently many *initial terms* that uniquely determine its terms.

Efficiency is measured in terms of *ring operations* (algebraic model), or of *bit operations* (Turing machine model). The cost of an algorithm is respectively estimated in terms of *arithmetic complexity* or of *binary complexity*. Both measures have their own usefulness: the algebraic model is relevant when ring operations have essentially unit cost (typically, if R is a finite ring such as the prime field $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$), while the bit complexity model is relevant when elements of R have a variable bitsize, and thus ring operations in R have variable cost (typically, when R is the ring $\mathbb{Z}$ of integer numbers).

The recurrence relation satisfied by the input sequence $(u_n)_{n\geq 0}$ might be of several types:

(C) *linear with constant coefficients*, that is of the form,

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

for some given $c_0, \ldots, c_{d-1}$ in R. In this case we simply say that $(u_n)_{n\geq 0}$ is *linearly recurrent* (or, *C-recursive*). The most basic examples are the geometric sequence $(q^n)_{n\geq 0}$, for $q \in R$, and the Fibonacci sequence $(F_n)_n$ with $F_{n+2} = F_{n+1} + F_n$ for $n \geq 0$ and $F_0 = 0, F_1 = 1$.

(P) *linear with polynomial coefficients*, of the form,

$$u_{n+d} = c_{d-1}(n)u_{n+d-1} + \cdots + c_0(n)u_n, \quad n \geq 0,$$

for some given rational functions $c_0(x), \ldots, c_{d-1}(x)$ in $R(x)$. In this case the sequence is called *holonomic* (or, *P-recursive*). Among the most basic examples, other than the C-recursive ones, there is the factorial sequence $(n!)_{n\geq 0} = (1, 1, 2, 6, 24, 120, \ldots)$ and the Motzkin sequence $(u_n)_{n\geq 0} = (1, 1, 2, 4, 9, 21, 51, \ldots)$ specified by the recurrence $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ and the initial conditions $u_0 = u_1 = 1$.

(Q) *linear with polynomial coefficients in q and q^n*, that is of the form,

$$u_{n+d} = c_{d-1}(q, q^n)u_{n+d-1} + \cdots + c_0(q, q^n)u_n, \; n \geq 0,$$

[0] The ring R is assumed to be commutative with unity and *effective* in the sense that its elements are represented using some data structure, and there exist algorithms for performing the basic ring operations $(+, -, \times)$ and for testing equality of elements in R.

2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)

Power Series Composition in Near-Linear Time

Yasunori Kinoshita
Tokyo Institute of Technology
Tokyo, Japan
kinoshita.y.aq@m.titech.ac.jp

Baitian Li
Institute for Interdisciplinary Information Sciences
Tsinghua University
Beijing, China
lbt21@mails.tsinghua.edu.cn

***Abstract*—We present an algebraic algorithm that computes the composition of two power series in softly linear time complexity. The previous best algorithms are $O(n^{1+o(1)})$ non-algebraic algorithm by Kedlaya and Umans (FOCS 2008) and an $O(n^{1.43})$ algebraic algorithm by Neiger, Salvy, Schost and Villard (JACM 2023).**

Our algorithm builds upon the recent Graeffe iteration approach to manipulate rational power series introduced by Bostan and Mori (SOSA 2021).

***Index Terms*—algebraic algorithms, computations on polynomials**

I. Introduction

Let $\mathbb{A}$ be a commutative ring and let $f(x), g(x)$ be polynomials in $\mathbb{A}[x]$ of degrees less than m and n, respectively. The problem of *power series composition* is to compute the coefficients of $f(g(x)) \bmod x^n$. The terminology stems from the idea that $g(x)$ can be seen as a truncated formal power series. This is a fundamental problem in computer algebra [21, Section 4.7], and has applications in various areas such as combinatorics [24] and cryptography [4].

A textbook algorithm for power series composition is due to Brent and Kung [9], which computes the composition in $O(\mathsf{M}(n)(n\log n)^{1/2})$ time complexity. Here $\mathsf{M}(n)$ denotes the complexity of computing the product of two polynomials of degree less than n.

The current best known algorithm for power series composition is due to Kedlaya and Umans [19], [20], which computes the composition in $(n\log q)^{1+o(1)}$ bit operations, where $\mathbb{A}$ is the finite field $\mathbb{F}_q$. Van der Hoeven and Lecerf [32] gave a detailed analysis of the subpolynomial term, and showed that the algorithm has a time complexity of $\tilde{O}(n2^{O(\sqrt{\log n \log\log n})}\log q)$.

In this paper, we present a simple algorithm that reduces the complexity of power series composition to near-linear time. Furthermore, our algorithm works over arbitrary commutative rings.

Theorem 1: Given polynomials $f(x), g(x) \in \mathbb{A}[x]$ with degree less than m and n, respectively, the power series composition $f(g(x)) \bmod x^n$ can be computed in $O(\mathsf{M}(n)\log m + \mathsf{M}(m))$ arithmetic operations.

We remark that our algorithm also has consequences for other computation models measured by bit complexity, such as boolean circuits and multitape Turing machines.

Yasunori Kinoshita is partially supported by Japan Science and Technology Agency (JST) as part of Adopting Sustainable Partnerships for Innovative Research Ecosystem (ASPIRE), Grant Number JPMJAP2302.

a) Technical Overview.: One of our key ingredients is Graeffe's root squaring method, which was first time introduced by Schönhage [26] in a purely algebraic setting to compute reciprocals of power series. Recently, Bostan and Mori [5] applied the Graeffe iteration to compute the coefficient of x^N of the series expansion of a rational power series $P(x)/Q(x)$, achieving time complexity $O(\mathsf{M}(n)\log N)$ where n is the degree bound of $P(x)$ and $Q(x)$.

In a nutshell, their algorithm works as follows. Considering multiplying $Q(-x)$ on both the numerator and the denominator, one obtains $[x^N]\frac{P(x)}{Q(x)} = [x^N]\frac{P(x)Q(-x)}{Q(x)Q(-x)}$. By the symmetry of $Q(x)Q(-x)$, one can rewrite the expression to $[x^N]\frac{U(x)}{V(x^2)}$ where $U(x) = P(x)Q(-x)$ and $V(x^2) = Q(x)Q(-x)$. Furthermore, one can split $U(x)$ into even and odd parts as $U(x) = U_e(x^2) + xU_o(x^2)$, and reduce the problem into $[x^{N/2}]\frac{U_e(x)}{V(x)}$ or $[x^{(N-1)/2}]\frac{U_o(x)}{V(x)}$ depending on the parity of N. Since the iteration reduces N by half each time, the algorithm has time complexity $O(\mathsf{M}(n)\log N)$ by using polynomial multiplication.

Despite their algorithm has the same time complexity as the classical algorithm due to Fiduccia [13], their algorithm is based on a totally different idea, and it is worth to get a closer look at their algorithm when $N \leq n$. In that case, the terms greater than N are irrelevant, hence the size of the problem can be reduced to N, compared with n. Since the size of the problem is reduced by half in each iteration, the total time complexity is $O(\mathsf{M}(N))$.

We extend the Graeffe iteration approach to a special kind of bivariate rational power series $P(x)/Q(x, y)$, where $Q(x, y) = 1/(1 - yg(x))$. The goal is still to compute the coefficient of x^n of the series expansion of $P(x)/Q(x, y)$, but in this case, the output is a polynomial in y, instead of a single $\mathbb{A}$-coefficient.

At the beginning of our algorithm, both P and Q have a degree of n with respect to x, and a degree at most 1 with respect to y. In each iteration, we halve the degree with respect to x, and double the degree with respect to y, therefore the size of the problem stays at n. Since there are $\log n$ iterations, the total time complexity is $O(\mathsf{M}(n)\log n)$.

We observe that the above algorithm, actually already solved the power projection problem, which is the transposed problem of power series composition. By the transposition principle —

[1] We use $[x^N]$ to denote coefficient extraction, i.e., for a power series $F(x)$, $[x^N]F(x)$ is the coefficient of x^N in $F(x)$.

2575-8454/24/$31.00 ©2024 IEEE
DOI 10.1109/FOCS61266.2024.00127

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

▷ Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms
- Efficiency is measured in terms of ring operations, or of bit operations

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms
- Efficiency is measured in terms of ring operations, or of bit operations

---

Two variants:

Given $(u_n)_n$ in $R^{\mathbb{N}}$ and $(N_1, \ldots, N_s) \in \mathbb{N}^s$, compute $(u_{N_1}, \ldots, u_{N_s})$ fast

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms
- Efficiency is measured in terms of ring operations, or of bit operations

---

Two variants:

Given $(u_n)_n$ in $R^{\mathbb{N}}$ and $(N_1, \ldots, N_s) \in \mathbb{N}^s$, compute $(u_{N_1}, \ldots, u_{N_s})$ fast

and

Given $(u_n)_n$ in $\mathbb{Z}^{\mathbb{N}}$ and $(N_\ell)_{\ell=1}^s \in \mathbb{N}^s$, compute $(u_{N_\ell} \bmod N_\ell)_{\ell=1}^s$ fast

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$,

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$,

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$,

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$,
  i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$,
  i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$
- $\sum_{k=0}^{n-1} q^{k^2}$: $u_{n+1} - u_n = q^{2n-1}(u_n - u_{n-1})$ with $u_0 = 0, u_1 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$, i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$
- $\sum_{k=0}^{n-1} q^{k^2}$: $u_{n+1} - u_n = q^{2n-1}(u_n - u_{n-1})$ with $u_0 = 0, u_1 = 1$

q-holonomic

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$, i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$
- $\sum_{k=0}^{n-1} q^{k^2}$: $u_{n+1} - u_n = q^{2n-1}(u_n - u_{n-1})$ with $u_0 = 0, u_1 = 1$

q-holonomic

- Göbel: $u_{n+1} = \frac{1}{n} \cdot (1 + u_0^2 + u_1^2 + \cdots + u_{n-1}^2)$ with $u_0 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$, i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$
- $\sum_{k=0}^{n-1} q^{k^2}$: $u_{n+1} - u_n = q^{2n-1}(u_n - u_{n-1})$ with $u_0 = 0, u_1 = 1$

q-holonomic

- Göbel: $u_{n+1} = \frac{1}{n} \cdot (1 + u_0^2 + u_1^2 + \cdots + u_{n-1}^2)$ with $u_0 = 1$
- Somos: $u_{n+5} = \frac{u_{n+4} \cdot u_{n+1} + u_{n+3} \cdot u_{n+2}}{u_n}$ with $u_0 = \cdots = u_4 = 1$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$, i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$
- $\sum_{k=0}^{n-1} q^{k^2}$: $u_{n+1} - u_n = q^{2n-1}(u_n - u_{n-1})$ with $u_0 = 0, u_1 = 1$

q-holonomic

- Göbel: $u_{n+1} = \frac{1}{n} \cdot (1 + u_0^2 + u_1^2 + \cdots + u_{n-1}^2)$ with $u_0 = 1$
- Somos: $u_{n+5} = \frac{u_{n+4}\cdot u_{n+1} + u_{n+3}\cdot u_{n+2}}{u_n}$ with $u_0 = \cdots = u_4 = 1$

non-linear

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$, i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$
- $\sum_{k=0}^{n-1} q^{k^2}$: $u_{n+1} - u_n = q^{2n-1}(u_n - u_{n-1})$ with $u_0 = 0, u_1 = 1$

q-holonomic

- Göbel: $u_{n+1} = \frac{1}{n} \cdot (1 + u_0^2 + u_1^2 + \cdots + u_{n-1}^2)$ with $u_0 = 1$
- Somos: $u_{n+5} = \frac{u_{n+4} \cdot u_{n+1} + u_{n+3} \cdot u_{n+2}}{u_n}$ with $u_0 = \cdots = u_4 = 1$

non-linear

- Katz: $u_{n+1} = \frac{\partial u_n}{\partial x} - M \cdot u_n$ with $M \in \mathcal{M}_r(\mathbb{F}_p(x))$, $u_0 = I_r$

Examples

Given a sequence $(u_n)_{n\geq 0}$ in a ring R, and $N \in \mathbb{N}$, compute u_N fast

- geometric: $u_n = q^n$, i.e., $u_{n+1} = q \cdot u_n$ with $u_0 = 1$
- Fibonacci: $u_{n+2} = u_{n+1} + u_n$ with $u_0 = u_1 = 1$

C-recursive

- factorial: $u_n = n!$, i.e., $u_{n+1} = (n+1) \cdot u_n$ with $u_0 = 1$
- Motzkin: $u_{n+1} = \frac{2n+3}{n+3} \cdot u_n + \frac{3n}{n+3} \cdot u_{n-1}$ with $u_0 = u_1 = 1$

P-recursive (holonomic)

- q-factorial: $u_n = [n]_q! := (1+q)\cdots(1+q+\cdots+q^{n-1})$, i.e., $u_{n+1} = (1+q+\cdots+q^n) \cdot u_n$ with $u_0 = 1$
- $\sum_{k=0}^{n-1} q^{k^2}$: $u_{n+1} - u_n = q^{2n-1}(u_n - u_{n-1})$ with $u_0 = 0, u_1 = 1$

q-holonomic

- Göbel: $u_{n+1} = \frac{1}{n} \cdot (1 + u_0^2 + u_1^2 + \cdots + u_{n-1}^2)$ with $u_0 = 1$
- Somos: $u_{n+5} = \frac{u_{n+4} \cdot u_{n+1} + u_{n+3} \cdot u_{n+2}}{u_n}$ with $u_0 = \cdots = u_4 = 1$

non-linear

- Katz: $u_{n+1} = \frac{\partial u_n}{\partial x} - M \cdot u_n$ with $M \in \mathcal{M}_r(\mathbb{F}_p(x))$, $u_0 = I_r$

p-curvature

Motivations

- algebraic complexity theory
  - *evaluation of polynomials*: x^N and $\sum_{\ell=0}^{N} 2^\ell x^\ell$ vs. $\sum_{\ell=0}^{N} 2^{2^\ell} x^\ell$ [Strassen, 1974]
- basic computer algebra questions
  - *matrix powering* M^N; more generally, $P(M)$ [Giesbrecht, 1995]
  - *Graeffe polynomials* $\prod_{P(\alpha)=0} (x - \alpha^N)$ [B., Flajolet, Salvy, Schost, 2006]
  - *modular polynomial exponentiation* $P^N \bmod Q$ [B., Mori, 2021]
  - *power series composition* $f \circ g \bmod x^N$ [Kinoshita, Li, 2024]
- more involved computer algebra questions
  - *polynomial linear algebra* [Storjohann, 2003]
  - *factoring in* $\mathbb{F}_q[x]$ [Berlekamp, 1970; Cantor, Zassenhaus, 1981; Shoup, 1995]
- algorithmic number theory
  - *primality tests* [Solovay, Strassen, 1977; Miller, 1976; Rabin, 1980; Atkin, Morain, 1994; Agrawal, Kayal, Saxena, 2004]
- effective algebraic geometry
  - *counting points on elliptic curves* over $\mathbb{F}_q$ [Schoof-Elkies-Atkin, 1992–1998]

Motivations

- algebraic complexity theory
  - *evaluation of polynomials*: x^N and $\sum_{\ell=0}^{N} 2^\ell x^\ell$ vs. $\sum_{\ell=0}^{N} 2^{2^\ell} x^\ell$ [Strassen, 1974]
- basic computer algebra questions
  - *matrix powering* M^N; more generally, $P(M)$ [Giesbrecht, 1995]
  - *Graeffe polynomials* $\prod_{P(\alpha)=0} (x - \alpha^N)$ [B., Flajolet, Salvy, Schost, 2006]
  - *modular polynomial exponentiation* $P^N \bmod Q$ [B., Mori, 2021]
  - *power series composition* $f \circ g \bmod x^N$ [Kinoshita, Li, 2024]
- more involved computer algebra questions
  - *polynomial linear algebra* [Storjohann, 2003]
  - *factoring in* $\mathbb{F}_q[x]$ [Berlekamp, 1970; Cantor, Zassenhaus, 1981; Shoup, 1995]
- algorithmic number theory
  - *primality tests* [Solovay, Strassen, 1977; Miller, 1976; Rabin, 1980; Atkin, Morain, 1994; Agrawal, Kayal, Saxena, 2004]
- effective algebraic geometry
  - *counting points on elliptic curves* over $\mathbb{F}_q$ [Schoof-Elkies-Atkin, 1992–1998]

Overview: naive algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(N)$	iterative algorithm	N	$\tilde{O}(N^2)$	iterative algorithm

† assuming quasi-optimal (FFT-based) integer multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: naive algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(N)$	iterative	N	$\tilde{O}(N^2)$	iterative
F_N	1	$O(N)$	algorithm	N	$\tilde{O}(N^2)$	algorithm

† assuming quasi-optimal (FFT-based) integer multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: naive algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(N)$	iterative	N	$\tilde{O}(N^2)$	iterative
F_N	1	$O(N)$	algorithm	N	$\tilde{O}(N^2)$	algorithm
$N!$	1	$O(N)$	iterative algorithm	$N \log N$	$\tilde{O}(N^2)$	iterative algorithm

† assuming quasi-optimal (FFT-based) integer multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: naive algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(N)$	iterative	N	$\tilde{O}(N^2)$	iterative
F_N	1	$O(N)$	algorithm	N	$\tilde{O}(N^2)$	algorithm
$N!$	1	$O(N)$	iterative	$N \log N$	$\tilde{O}(N^2)$	iterative
M_N	1	$O(N)$	algorithm	$N \log N$	$\tilde{O}(N^2)$	algorithm

† assuming quasi-optimal (FFT-based) integer multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: naive algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(N)$	iterative	N	$\tilde{O}(N^2)$	iterative
F_N	1	$O(N)$	algorithm	N	$\tilde{O}(N^2)$	algorithm
$N!$	1	$O(N)$	iterative	$N \log N$	$\tilde{O}(N^2)$	iterative
M_N	1	$O(N)$	algorithm	$N \log N$	$\tilde{O}(N^2)$	algorithm
$[N]_q!$	1	$O(N)$	iterative algorithm	N^2	$\tilde{O}(N^3)$	iterative algorithm

† assuming quasi-optimal (FFT-based) integer multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: naive algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(N)$	iterative	N	$\tilde{O}(N^2)$	iterative
F_N	1	$O(N)$	algorithm	N	$\tilde{O}(N^2)$	algorithm
$N!$	1	$O(N)$	iterative	$N \log N$	$\tilde{O}(N^2)$	iterative
M_N	1	$O(N)$	algorithm	$N \log N$	$\tilde{O}(N^2)$	algorithm
$[N]_q!$	1	$O(N)$	iterative	N^2	$\tilde{O}(N^3)$	iterative
$\sum_{n=0}^{N} q^{n^2}$	1	$O(N^2)$	algorithm	N^2	$\tilde{O}(N^3)$	algorithm

† assuming quasi-optimal (FFT-based) integer multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary powering	N	$\tilde{O}(N)$	binary powering

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary	N	$\tilde{O}(N)$	binary
F_N	1	$O(\log N)$	powering	N	$\tilde{O}(N)$	powering

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary	N	$\tilde{O}(N)$	binary
F_N	1	$O(\log N)$	powering	N	$\tilde{O}(N)$	powering
$N!$	1	$\tilde{O}(\sqrt{N})$	baby-steps / giant-steps	$N \log N$	$\tilde{O}(N)$	binary splitting

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary	N	$\tilde{O}(N)$	binary
F_N	1	$O(\log N)$	powering	N	$\tilde{O}(N)$	powering
$N!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	$N \log N$	$\tilde{O}(N)$	binary
M_N	1	$\tilde{O}(\sqrt{N})$	giant-steps	$N \log N$	$\tilde{O}(N)$	splitting

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary	N	$\tilde{O}(N)$	binary
F_N	1	$O(\log N)$	powering	N	$\tilde{O}(N)$	powering
$N!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	$N \log N$	$\tilde{O}(N)$	binary
M_N	1	$\tilde{O}(\sqrt{N})$	giant-steps	$N \log N$	$\tilde{O}(N)$	splitting
$[N]_q!$	1	$\tilde{O}(\sqrt{N})$	baby-steps / giant-steps	N^2	$\tilde{O}(N^2)$	binary splitting

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary	N	$\tilde{O}(N)$	binary
F_N	1	$O(\log N)$	powering	N	$\tilde{O}(N)$	powering
$N!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	$N \log N$	$\tilde{O}(N)$	binary
M_N	1	$\tilde{O}(\sqrt{N})$	giant-steps	$N \log N$	$\tilde{O}(N)$	splitting
$[N]_q!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	N^2	$\tilde{O}(N^2)$	binary
$\sum_{n=0}^{N-1} q^{n^2}$	1	$\tilde{O}(\sqrt{N})$	giant-steps	N^2	$\tilde{O}(N^2)$	splitting

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary	N	$\tilde{O}(N)$	binary
F_N	1	$O(\log N)$	powering	N	$\tilde{O}(N)$	powering
$N!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	$N \log N$	$\tilde{O}(N)$	binary
M_N	1	$\tilde{O}(\sqrt{N})$	giant-steps	$N \log N$	$\tilde{O}(N)$	splitting
$[N]_q!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	N^2	$\tilde{O}(N^2)$	binary
$\sum_{n=0}^{N-1} q^{n^2}$	1	$\tilde{O}(\sqrt{N})$	giant-steps	N^2	$\tilde{O}(N^2)$	splitting

▷ For $R = \mathbb{F}_p$: $M_N \in R$ in $O(\log N)$ ops. in R; same for any u_N with *algebraic* GF $\sum_n u_n x^n$ [B., Christol, Dumas, 2016], [B., Caruso, Christol, Dumas, 2019]

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

Overview: best algorithms

Seq. Term	Arith. size	Arith. cost	Method	Bit size	Bit cost	Method
q^N	1	$O(\log N)$	binary	N	$\tilde{O}(N)$	binary
F_N	1	$O(\log N)$	powering	N	$\tilde{O}(N)$	powering
$N!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	$N \log N$	$\tilde{O}(N)$	binary
M_N	1	$\tilde{O}(\sqrt{N})$	giant-steps	$N \log N$	$\tilde{O}(N)$	splitting
$[N]_q!$	1	$\tilde{O}(\sqrt{N})$	baby-steps /	N^2	$\tilde{O}(N^2)$	binary
$\sum_{n=0}^{N-1} q^{n^2}$	1	$\tilde{O}(\sqrt{N})$	giant-steps	N^2	$\tilde{O}(N^2)$	splitting

- First part of this course focusses on the first two rows
- 12/11/2025: two middle rows

† assuming FFT-based integer and polynomial multiplication $\mathsf{M}(N) = \tilde{O}(N)$

I.

COMPUTING TERMS OF C-RECURSIVE SEQUENCES

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a field $\mathbb{K}$, and $N \in \mathbb{N}$, compute u_N fast

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a field $\mathbb{K}$, and $N \in \mathbb{N}$, compute u_N fast

▷ Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a field $\mathbb{K}$, and $N \in \mathbb{N}$, compute u_N fast

- Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms
- Efficiency measured in nb. of ops. $(\pm, \times, \div)$ in $\mathbb{K}$ (arithmetic complexity)

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a field $\mathbb{K}$, and $N \in \mathbb{N}$, compute u_N fast

- Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms
- Efficiency measured in nb. of ops. $(\pm, \times, \div)$ in $\mathbb{K}$ (arithmetic complexity)

Today: input sequence is C-recursive, given by initial terms $u_0, \ldots, u_{d-1}$ and a linear recurrence with constant coefficients $(c_0, \ldots, c_{d-1}) \in \mathbb{K}^d$

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0.$$

Main question

Given a sequence $(u_n)_{n\geq 0}$ in a field $\mathbb{K}$, and $N \in \mathbb{N}$, compute u_N fast

▷ Input $(u_n)_{n\geq 0}$ is assumed to be a recurrent sequence, and it is specified by a recurrence relation and enough initial terms

▷ Efficiency measured in nb. of ops. $(\pm, \times, \div)$ in $\mathbb{K}$ (arithmetic complexity)

Today: input sequence is C-recursive, given by initial terms $u_0, \ldots, u_{d-1}$ and a linear recurrence with constant coefficients $(c_0, \ldots, c_{d-1}) \in \mathbb{K}^d$

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0.$$

▷ Def. $\Gamma(x) := x^d - \sum_{i=0}^{d-1} c_i x^i$ is called *characteristic polynomial* for $(u_n)_{n\geq 0}$

Example: N-th term of the Fibonacci sequence

L'INTERMÉDIAIRE

DES

MATHÉMATICIENS

DIRIGÉ PAR

C.-A. LAISANT, DOCTEUR ÈS SCIENCES, | ÉMILE LEMOINE, INGÉNIEUR CIVIL,

ANCIENS ÉLÈVES DE L'ÉCOLE POLYTECHNIQUE.

TOME VI. — 1899.

PARIS,

GAUTHIER-VILLARS, IMPRIMEUR-LIBRAIRE

DU BUREAU DES LONGITUDES, DE L'ÉCOLE POLYTECHNIQUE,

55, Quai des Grands-Augustins, 55.

—

1899

Je serais également heureux d'avoir des renseignements bibliographiques (postérieurs à Delambre) sur les études *scientifiques* consacrées aux cadrans verticaux dans l'antiquité. PAUL TANNERY.

1539. **[H1b]** Il peut arriver qu'une équation différentielle admette une intégrale particulière imaginaire. La connaissance de celle-ci peut-elle être de quelque utilité pour l'intégration de l'équation donnée? H. BROCARD.

1540. **[I2b]** b étant un nombre composé, quelles sont les valeurs de b qui rendent le produit $1.2.3\ldots(b-1)$ non divisible par b? G. DE ROCQUIGNY.

1541. **[I25a]** Quel est le procédé le plus expéditif pour calculer un terme très éloigné dans la série de Fibonacci :

0, 1, 1, 2, 3, 5, 8, ...?

G. DE ROCQUIGNY.

1542. **[E1a]** Est-il exact, et dans ce cas comment pourrait-on le démontrer, que :

1° L'expression

$$\Phi_n(x) = n^{x-1} - \frac{x}{1}(n-1)^{x-1} + \frac{x(x-1)}{1.2}(n-2)^{x-1} - \ldots,$$

où n désigne un entier et x une quantité positive quelconque, tend vers zéro en même temps que n vers l'infini;

2° La loi de décroissance des quantités $\Phi_n(x)$ est assez rapide pour que la série

$$\Phi_1(x) + \Phi_2(x) + \Phi_3(x) + \ldots + \Phi_n(x) + \ldots$$

soit convergente;

3° La somme de cette série a pour limite la fonction $\Gamma(x)$?

Tout cela étant, la fonction eulérienne de deuxième $\Gamma(1+x)$ se présenterait comme la limite de l'expression

$$n^x - \frac{x}{1}(n-1)^x + \frac{x(x-1)}{1.2}(n-2)^x - \ldots.$$

E.-A. Majol.

Binary powering

Problem: Given a ring R, $a \in R$ and $N \geq 1$, compute a^N

Binary powering

Problem: Given a ring R, $a \in R$ and $N \geq 1$, compute a^N

▷ Naive (iterative) algorithm: $O(N)$ ops. in R

Binary powering

Problem: Given a ring R, $a \in R$ and $N \geq 1$, compute a^N

▷ Naive (iterative) algorithm: $O(N)$ ops. in R

▷ Better algorithm [Pingala, 200 BC]: $O(\log N)$ ops. in R

Compute a^N recursively, using square-and-multiply

$$a^N = \begin{cases} (a^{N/2})^2, & \text{if } N \text{ is even,} \\ a \cdot (a^{\frac{N-1}{2}})^2, & \text{else.} \end{cases}$$

Binary powering

Problem: Given a ring R, $a \in R$ and $N \geq 1$, compute a^N

▷ Naive (iterative) algorithm: $O(N)$ ops. in R

▷ Better algorithm [Pingala, 200 BC]: $O(\log N)$ ops. in R

Compute a^N recursively, using square-and-multiply

$$a^N = \begin{cases} (a^{N/2})^2, & \text{if } N \text{ is even,} \\ a \cdot (a^{\frac{N-1}{2}})^2, & \text{else.} \end{cases}$$

▷ Application to *fast matrix exponentiation*, with $R = \mathcal{M}_d(\mathbb{K})$:

- if $M \in \mathcal{M}_d(\mathbb{K})$, then M^N in $O(d^\theta \log N)$ ops. in $\mathbb{K}$, where
  - $\theta =$ *matrix multiplication exponent*
  - $2 \leq \theta \leq 2.371339$ [Alman, Duan, Vassilevska Williams, Xu, Xu, Zhou, 2025]

Binary powering

Problem: Given a ring R, $a \in R$ and $N \geq 1$, compute a^N

▷ Naive (iterative) algorithm: $O(N)$ ops. in R

▷ Better algorithm [Pingala, 200 BC]: $O(\log N)$ ops. in R

Compute a^N recursively, using square-and-multiply

$$a^N = \begin{cases} (a^{N/2})^2, & \text{if } N \text{ is even,} \\ a \cdot (a^{\frac{N-1}{2}})^2, & \text{else.} \end{cases}$$

▷ Application to *fast modular polynomial exponentiation*, with $R = \mathbb{K}[x]/(Q)$:

- if $P, Q \in \mathbb{K}[x]_{<d}$, then $P^N \bmod Q$ in $O(\mathsf{M}(d) \log N)$ ops. in $\mathbb{K}$, where
  - $\mathsf{M}(d) =$ *complexity of multiplication in* $\mathbb{K}[x]_{<d}$
    $= O(d \cdot \log d \cdot \log\log d) = \tilde{O}(d)$ via FFT [Schönhage, Strassen, 1971]
  - product in $R = \mathbb{K}[x]/(Q)$ via Newton iteration in $O(\mathsf{M}(d))$ [Strassen, 1973]

Binary powering

Problem: Given a ring R, $a \in R$ and $N \geq 1$, compute a^N

▷ Naive (iterative) algorithm: $O(N)$ ops. in R

▷ Better algorithm [Pingala, 200 BC]: $O(\log N)$ ops. in R

Compute a^N recursively, using square-and-multiply

$$a^N = \begin{cases} (a^{N/2})^2, & \text{if } N \text{ is even,} \\ a \cdot (a^{\frac{N-1}{2}})^2, & \text{else.} \end{cases}$$

▷ Application to *fast modular integer exponentiation*, with $R = \mathbb{Z}/A\mathbb{Z}$:

- N-th decimal of $\frac{1}{A}$ via $(10^{N-1} \bmod A)$ in $O(\mathsf{M}_{\mathbb{Z}}(\log A) \log N)$ bit ops.

Example: N-th decimal of a rational number

What is the 10^{10^6}-th decimal of $A = \frac{1}{2039}$?

Example: N-th decimal of a rational number

What is the 10^{10^6}-th decimal of $A = \frac{1}{2039}$?

```
> N:=10^(10^6): A:=2039:
> iquo(10*(irem(10^(N-1),A)), A);
```

Example: N-th decimal of a rational number

What is the 10^{10^6}-th decimal of $A = \frac{1}{2039}$?

```
> N:=10^(10^6): A:=2039:
> iquo(10*(irem(10^(N-1),A)), A);
```

Error, numeric exception: overflow

Example: N-th decimal of a rational number

What is the 10^{10^6}-th decimal of $A = \frac{1}{2039}$?

```
> N:=10^(10^6): A:=2039:
> iquo(10*(irem(10^(N-1),A)), A);
```

Error, numeric exception: overflow

```
> st:=time(): iquo(10*(`&^`(10,N-1) mod A), A), time()-st;
```

Example: N-th decimal of a rational number

What is the 10^{10^6}-th decimal of $A = \frac{1}{2039}$?

```
> N:=10^(10^6): A:=2039:
> iquo(10*(irem(10^(N-1),A)), A);
```

Error, numeric exception: overflow

```
> st:=time(): iquo(10*(`&^`(10,N-1) mod A), A), time()-st;
```

6, 0.037

Example: N-th decimal of a rational number

What is the 10^{10^6}-th decimal of $A = \frac{1}{2039}$?

```
> N:=10^(10^6): A:=2039:
> iquo(10*(irem(10^(N-1),A)), A);
```

Error, numeric exception: overflow

```
> st:=time(): iquo(10*(`&^`(10,N-1) mod A), A), time()-st;
```

6, 0.037

▷ The following also computes the right answer. Can you see why?

```
> n := irem(N,A-1);
> iquo(10*(irem(10^(n-1),A)), A);
```

6

Schönhage's Golden Rule 1

RULE 1: *Do care about the size of* $\mathbf{0}$ *!*

Fast polynomial division and modular exponentiation [Strassen, 1973]

Pb: Given $F \in \mathbb{K}[x]_{<2d}$ and $Q \in \mathbb{K}[x]_d$ compute (U, R) in Euclidean division

$$F = UQ + R$$

Fast polynomial division and modular exponentiation [Strassen, 1973]

Pb: Given $F \in \mathbb{K}[x]_{<2d}$ and $Q \in \mathbb{K}[x]_d$ compute (U, R) in Euclidean division

$$F = UQ + R$$

Naive algorithm: $O(d^2)$

Fast polynomial division and modular exponentiation [Strassen, 1973]

Pb: Given $F \in \mathbb{K}[x]_{<2d}$ and $Q \in \mathbb{K}[x]_d$ compute (U, R) in Euclidean division

$$F = UQ + R$$

Naive algorithm: $O(d^2)$

Idea: when $\mathbb{K} = \mathbb{R}$, look at $F = UQ + R$ from infinity: $U \sim_{+\infty} F/Q$

Fast polynomial division and modular exponentiation [Strassen, 1973]

Pb: Given $F \in \mathbb{K}[x]_{<2d}$ and $Q \in \mathbb{K}[x]_d$ compute (U, R) in Euclidean division

$$F = UQ + R$$

Naive algorithm: $O(d^2)$

Idea: when $\mathbb{K} = \mathbb{R}$, look at $F = UQ + R$ from infinity: $U \sim_{+\infty} F/Q$

Formalization: Let $D = \deg(F)$. Then $\deg(U) = D - d < d$, $\deg(R) < d$ and

$$\underbrace{F(1/x)x^D}_{\mathrm{rev}(F)} = \underbrace{Q(1/x)x^d}_{\mathrm{rev}(Q)} \cdot \underbrace{U(1/x)x^{D-d}}_{\mathrm{rev}(U)} + \underbrace{R(1/x)x^{\deg(R)}}_{\mathrm{rev}(R)} \cdot x^{D-\deg(R)}$$

Fast polynomial division and modular exponentiation [Strassen, 1973]

Pb: Given $F \in \mathbb{K}[x]_{<2d}$ and $Q \in \mathbb{K}[x]_d$ compute (U, R) in Euclidean division

$$F = UQ + R$$

Naive algorithm: $O(d^2)$

Idea: when $\mathbb{K} = \mathbb{R}$, look at $F = UQ + R$ from infinity: $U \sim_{+\infty} F/Q$

Formalization: Let $D = \deg(F)$. Then $\deg(U) = D - d < d$, $\deg(R) < d$ and

$$\underbrace{F(1/x)x^D}_{\mathrm{rev}(F)} = \underbrace{Q(1/x)x^d}_{\mathrm{rev}(Q)} \cdot \underbrace{U(1/x)x^{D-d}}_{\mathrm{rev}(U)} + \underbrace{R(1/x)x^{\deg(R)}}_{\mathrm{rev}(R)} \cdot x^{D-\deg(R)}$$

Algorithm: Complexity

1. Compute $A = 1/\mathrm{rev}(Q) \mod x^{D-d+1}$ — $3\,\mathsf{M}(d) + O(d)$
2. Compute $\mathrm{rev}(U) = \mathrm{rev}(F) \cdot A \mod x^{D-d+1}$ — $\mathsf{M}(d)$
3. Recover U and deduce $R = F - U \cdot Q$ — $\mathsf{M}(d) + O(d)$

Fast polynomial division and modular exponentiation [Strassen, 1973]

Pb: Given $F \in \mathbb{K}[x]_{<2d}$ and $Q \in \mathbb{K}[x]_d$ compute (U, R) in Euclidean division

$$F = UQ + R$$

Naive algorithm: $O(d^2)$

Idea: when $\mathbb{K} = \mathbb{R}$, look at $F = UQ + R$ from infinity: $U \sim_{+\infty} F/Q$

Formalization: Let $D = \deg(F)$. Then $\deg(U) = D - d < d$, $\deg(R) < d$ and

$$\underbrace{F(1/x)x^D}_{\mathrm{rev}(F)} = \underbrace{Q(1/x)x^d}_{\mathrm{rev}(Q)} \cdot \underbrace{U(1/x)x^{D-d}}_{\mathrm{rev}(U)} + \underbrace{R(1/x)x^{\deg(R)}}_{\mathrm{rev}(R)} \cdot x^{D-\deg(R)}$$

Algorithm: Complexity

1. Compute $A = 1/\mathrm{rev}(Q) \mod x^{D-d+1}$ — $3\,\mathsf{M}(d) + O(d)$
2. Compute $\mathrm{rev}(U) = \mathrm{rev}(F) \cdot A \mod x^{D-d+1}$ — $\mathsf{M}(d)$
3. Recover U and deduce $R = F - U \cdot Q$ — $\mathsf{M}(d) + O(d)$

▷ Step 1 based on formal Newton iteration; it depends only on Q (not on F)

Fast polynomial division and modular exponentiation [Strassen, 1973]

Pb: Given $F \in \mathbb{K}[x]_{<2d}$ and $Q \in \mathbb{K}[x]_d$ compute (U, R) in Euclidean division

$$F = UQ + R$$

Naive algorithm: $O(d^2)$

Idea: when $\mathbb{K} = \mathbb{R}$, look at $F = UQ + R$ from infinity: $U \sim_{+\infty} F/Q$

Formalization: Let $D = \deg(F)$. Then $\deg(U) = D - d < d$, $\deg(R) < d$ and

$$\underbrace{F(1/x)x^D}_{\mathrm{rev}(F)} = \underbrace{Q(1/x)x^d}_{\mathrm{rev}(Q)} \cdot \underbrace{U(1/x)x^{D-d}}_{\mathrm{rev}(U)} + \underbrace{R(1/x)x^{\deg(R)}}_{\mathrm{rev}(R)} \cdot x^{D-\deg(R)}$$

Algorithm: Complexity

1. Compute $A = 1/\mathrm{rev}(Q) \mod x^{D-d+1}$ — $3\,\mathsf{M}(d) + O(d)$
2. Compute $\mathrm{rev}(U) = \mathrm{rev}(F) \cdot A \mod x^{D-d+1}$ — $\mathsf{M}(d)$
3. Recover U and deduce $R = F - U \cdot Q$ — $\mathsf{M}(d) + O(d)$

▷ Step 1 based on formal Newton iteration; it depends only on Q (not on F)

▷ Corollary: Modular exponentiation $x^N \bmod Q$ in $\sim 3\,\mathsf{M}(d) \log N$ ops. in $\mathbb{K}$

Application: fast modular exponentiation

Pb: Given $P, Q \in \mathbb{K}[x]_{<d}$ compute $P^N \bmod Q$

Application: fast modular exponentiation

Pb: Given $P, Q \in \mathbb{K}[x]_{<d}$ compute $P^N \bmod Q$

Naive algorithm: $O(Nd^2)$

Application: fast modular exponentiation

Pb: Given $P, Q \in \mathbb{K}[x]_{<d}$ compute $P^N \bmod Q$

Naive algorithm: $O(Nd^2)$

Better algorithm: binary powering in $R = \mathbb{K}[x]/(Q)$ $O(\log N)$ ops. in R

Application: fast modular exponentiation

Pb: Given $P, Q \in \mathbb{K}[x]_{<d}$ compute $P^N \bmod Q$

Naive algorithm: $O(Nd^2)$

Better algorithm: binary powering in $R = \mathbb{K}[x]/(Q)$ $O(\log N)$ ops. in R

Algorithm: Complexity

1. Precompute $A = 1/\mathrm{rev}(Q) \bmod x^d$ $3\,\mathsf{M}(d) + O(d)$
2. Perform $\lfloor \log N \rfloor$ square-and-multiply modulo Q; for each $V^2 \bmod Q$:
   - compute the square $F := V^2$ $\mathsf{M}(d)$
   - compute the remainder $F \bmod Q$:
     - Compute $\mathrm{rev}(U) = \mathrm{rev}(F) \cdot A \bmod x^d$ $\mathsf{M}(d)$
     - Recover U and deduce $R = F - U \cdot Q$ $\mathsf{M}(d) + O(d)$

Application: fast modular exponentiation

Pb: Given $P, Q \in \mathbb{K}[x]_{<d}$ compute $P^N \bmod Q$

Naive algorithm: $O(Nd^2)$

Better algorithm: binary powering in $R = \mathbb{K}[x]/(Q)$ $O(\log N)$ ops. in R

Algorithm: Complexity

1. Precompute $A = 1/\mathrm{rev}(Q) \bmod x^d$ $3\,\mathsf{M}(d) + O(d)$
2. Perform $\lfloor \log N \rfloor$ square-and-multiply modulo Q; for each $V^2 \bmod Q$:
   - compute the square $F := V^2$ $\mathsf{M}(d)$
   - compute the remainder $F \bmod Q$:
     - Compute $\mathrm{rev}(U) = \mathrm{rev}(F) \cdot A \bmod x^d$ $\mathsf{M}(d)$
     - Recover U and deduce $R = F - U \cdot Q$ $\mathsf{M}(d) + O(d)$

▷ $P^N \bmod Q$ in $3\,\mathsf{M}(d)\,(1 + \lfloor \log N \rfloor) + O(d \log N) \sim 3\,\mathsf{M}(d) \log N$ ops. in $\mathbb{K}$

Application: fast modular exponentiation

Pb: Given $P, Q \in \mathbb{K}[x]_{<d}$ compute $P^N \bmod Q$

Naive algorithm: $O(Nd^2)$

Better algorithm: binary powering in $R = \mathbb{K}[x]/(Q)$ $O(\log N)$ ops. in R

Algorithm: Complexity

1. Precompute $A = 1/\mathrm{rev}(Q) \bmod x^d$ $3\,\mathsf{M}(d) + O(d)$
2. Perform $\lfloor \log N \rfloor$ square-and-multiply modulo Q; for each $V^2 \bmod Q$:
   - compute the square $F := V^2$ $\mathsf{M}(d)$
   - compute the remainder $F \bmod Q$:
     - Compute $\mathrm{rev}(U) = \mathrm{rev}(F) \cdot A \bmod x^d$ $\mathsf{M}(d)$
     - Recover U and deduce $R = F - U \cdot Q$ $\mathsf{M}(d) + O(d)$

▷ $P^N \bmod Q$ in $3\,\mathsf{M}(d)\,(1 + \lfloor \log N \rfloor) + O(d \log N) \sim 3\,\mathsf{M}(d) \log N$ ops. in $\mathbb{K}$

▷ A bit optimistic (did not count "-and-multiply"...); OK if $P = x$

Schönhage's Golden Rule 2

RULE 2: *Do not waste a factor of two !*

Computing the N-th term of a C-recursive sequence

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

Computing the N-th term of a C-recursive sequence

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

rewrites

$$\underbrace{\begin{bmatrix} u_N \\ u_{N+1} \\ \vdots \\ u_{N+d-1} \end{bmatrix}}_{v_N} = \underbrace{\begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ c_0 & c_1 & \cdots & c_{d-1} \end{bmatrix}}_{C^T} \underbrace{\begin{bmatrix} u_{N-1} \\ u_N \\ \vdots \\ u_{N+d-2} \end{bmatrix}}_{v_{N-1}} = (C^T)^N \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{d-1} \end{bmatrix}}_{v_0}, \qquad N \geq 1.$$

Computing the N-th term of a C-recursive sequence

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

rewrites

$$\underbrace{\begin{bmatrix} u_N \\ u_{N+1} \\ \vdots \\ u_{N+d-1} \end{bmatrix}}_{v_N} = \underbrace{\begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ c_0 & c_1 & \cdots & c_{d-1} \end{bmatrix}}_{C^T} \underbrace{\begin{bmatrix} u_{N-1} \\ u_N \\ \vdots \\ u_{N+d-2} \end{bmatrix}}_{v_{N-1}} = (C^T)^N \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{d-1} \end{bmatrix}}_{v_0}, \qquad N \geq 1.$$

▷ [Miller, Spencer Brown, 1966]: binary powering in $\mathcal{M}_d(\mathbb{K})$ $\quad O(d^\theta \log(N))$

Computing the N-th term of a C-recursive sequence

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

rewrites

$$\underbrace{\begin{bmatrix} u_N \\ u_{N+1} \\ \vdots \\ u_{N+d-1} \end{bmatrix}}_{v_N} = \underbrace{\begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ c_0 & c_1 & \cdots & c_{d-1} \end{bmatrix}}_{C^T} \underbrace{\begin{bmatrix} u_{N-1} \\ u_N \\ \vdots \\ u_{N+d-2} \end{bmatrix}}_{v_{N-1}} = (C^T)^N \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{d-1} \end{bmatrix}}_{v_0}, \qquad N \geq 1.$$

- ▷ [Miller, Spencer Brown, 1966]: binary powering in $\mathcal{M}_d(\mathbb{K})$ $\quad O(d^\theta \log(N))$
- ▷ [Fiduccia, 1985] binary powering in $\mathbb{K}[x]/(\Gamma)$, with $\Gamma = x^d - \sum_{i=0}^{d-1} c_i x^i$

$$u_N = e \cdot v_N = e \cdot (C^T)^N \cdot v_0 = v_0^T \cdot C^N \cdot e^T = \langle v_0,\ x^N \bmod \Gamma \rangle,$$

where $e = [1 \quad 0 \quad \cdots \quad 0]$. $\qquad \sim 3\,\mathsf{M}(d)\log N$

Computing the N-th term of a C-recursive sequence

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

rewrites

$$\underbrace{\begin{bmatrix} u_N \\ u_{N+1} \\ \vdots \\ u_{N+d-1} \end{bmatrix}}_{v_N} = \underbrace{\begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ c_0 & c_1 & \cdots & c_{d-1} \end{bmatrix}}_{C^T} \underbrace{\begin{bmatrix} u_{N-1} \\ u_N \\ \vdots \\ u_{N+d-2} \end{bmatrix}}_{v_{N-1}} = (C^T)^N \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{d-1} \end{bmatrix}}_{v_0}, \qquad N \geq 1.$$

- ▷ [Miller, Spencer Brown, 1966]: binary powering in $\mathcal{M}_d(\mathbb{K})$ $\quad O(d^\theta \log(N))$
- ▷ [Fiduccia, 1985] binary powering in $\mathbb{K}[x]/(\Gamma)$, with $\Gamma = x^d - \sum_{i=0}^{d-1} c_i x^i$

$$u_N = e \cdot v_N = e \cdot (C^T)^N \cdot v_0 = v_0^T \cdot C^N \cdot e^T = \langle v_0,\ x^N \bmod \Gamma \rangle,$$

where $e = [1 \quad 0 \quad \cdots \quad 0]$. $\quad \sim 3\,\mathsf{M}(d)\log N$

- ▷ [B., Mori, 2021]: different ideas / algorithms (upcoming) $\quad \sim 2\,\mathsf{M}(d)\log N$

Example: N-th term of the Fibonacci sequence

Fiduccia's algorithm (1985): binary powering in the ring $\mathbb{K}[x]/(x^2-x-1)$:

$$C^n = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \text{matrix of} \quad (x^n \bmod x^2-x-1)$$

$$\Longrightarrow \quad F_{n-2} + xF_{n-1} = \quad x^n \bmod x^2-x-1$$

Cost: $O(\log N)$ products in $\mathbb{K}[x]/(x^2-x-1) \longrightarrow O(\log N)$ ops. for F_N

Example: N-th term of the Fibonacci sequence

Fiduccia's algorithm (1985): binary powering in the ring $\mathbb{K}[x]/(x^2-x-1)$:

$$C^n = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \text{matrix of} \quad (x^n \bmod x^2-x-1)$$

$$\Longrightarrow \quad F_{n-2} + xF_{n-1} = \quad x^n \bmod x^2-x-1$$

Cost: $O(\log N)$ products in $\mathbb{K}[x]/(x^2-x-1) \longrightarrow O(\log N)$ ops. for F_N

Explains Shortt's algorithm (1978):

$$F_{2n-2} + xF_{2n-1} = (F_{n-2} + xF_{n-1})^2 \bmod x^2-x-1$$

Example: N-th term of the Fibonacci sequence

Fiduccia's algorithm (1985): binary powering in the ring $\mathbb{K}[x]/(x^2-x-1)$:

$$C^n = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \text{matrix of} \quad (x^n \bmod x^2-x-1)$$

$$\Longrightarrow \quad F_{n-2} + xF_{n-1} = \quad x^n \bmod x^2-x-1$$

Cost: $O(\log N)$ products in $\mathbb{K}[x]/(x^2-x-1) \longrightarrow O(\log N)$ ops. for F_N

Explains Shortt's algorithm (1978):

$$F_{2n-2} + xF_{2n-1} = (F_{n-2} + xF_{n-1})^2 \bmod x^2-x-1$$

$$\text{implies} \begin{cases} F_{2n-2} &= F_{n-2}^2 + F_{n-1}^2 \\ F_{2n-1} &= F_{n-1}^2 + 2F_{n-1}F_{n-2} \end{cases}$$

$$(F_0, F_1) \to (F_2, F_3) \to (F_6, F_7) \to (F_{14}, F_{15}) \to \dots$$

Cost: $3\times$ and $3+$ per arrow

Example: N-th term of the Fibonacci sequence

L'INTERMÉDIAIRE

DES

MATHÉMATICIENS

DIRIGÉ PAR

C.-A. LAISANT, DOCTEUR ÈS SCIENCES, | ÉMILE LEMOINE, INGÉNIEUR CIVIL,

ANCIENS ÉLÈVES DE L'ÉCOLE POLYTECHNIQUE.

TOME VII. — 1900.

PARIS,

GAUTHIER-VILLARS, IMPRIMEUR-LIBRAIRE

DU BUREAU DES LONGITUDES, DE L'ÉCOLE POLYTECHNIQUE,

55, Quai des Grands-Augustins, 55.

1900

cise en prenant un plus grand nombre de décimales, nous paraît, en effet, l'un des procédés de calcul les meilleurs. Elle permet de trouver des termes, exactement, jusqu'à une limite assez éloignée; et pour des termes très lointains et impossibles à écrire, elle a le mérite de faire connaître le nombre des chiffres. C.-A. LAISANT.

La suite de Fibonacci peut être exprimée par une autre loi que celle qui est ordinairement employée. En effet, en considérant les termes de cette suite comme provenant du calcul des réduites d'une fraction continue périodique symétrique dont tous les quotients incomplets sont égaux à l'unité, on trouve, en appliquant les formules de Lucas (*Théorie des nombres*) :

$$u_{2n} = u_n^2 + u_{n-1}^2,$$
$$u_{2n-1} = u_{n-1}(u_n + u_{n-2}) = u_n^2 - u_{n-2}^2,$$

le point de départ étant

$$u_0 = 1, \qquad u_1 = 1, \qquad u_2 = 2.$$

Ces formules permettent de calculer assez rapidement les termes de cette suite. En moins d'un quart d'heure j'ai pu calculer

$$u_{100} = 573\,147\,844\,013\,817\,084\,101.$$

Accessoirement ces formules montrent que tous les termes de rang impair sont des nombres composés sauf le terme $u_3 = 3$ et que tous les nombres premiers, que l'on ne peut rencontrer qu'aux rangs pairs, sont tous de la forme $4u + 1$ puisqu'ils sont la somme de deux carrés premiers entre eux. G. PICOU.

Les valeurs de u_{100} données par MM. *Rosace* et Picou sont toutes deus exactes, seulement M. *Rosace* prend pour premiers termes 0, 1, 1, 2, 3, ... et M. Picou 1, 1, 2, 3, ... E. LEMOINE.

1549. (**1899**, 150) (E. DUPORCQ). — *Sur une propriété nouvelle de l'ovale de Descartes.* — Soient R le rayon de courbure, θ, θ', θ'' les angles que font les rayons vecteurs avec la normale; selon que l'ovale est rapporté en coordonnées bipolaires aux foyers φ, φ'; φ, φ''; φ', φ'' on a pour le rayon de courbure les formes

$$(1) \quad R = \frac{\cos\theta + m\cos\theta'}{\frac{\cos^2\theta}{\rho} + m\frac{\cos^2\theta'}{\rho'}} = \frac{\cos\theta' + p\cos\theta''}{\frac{\cos^2\theta'}{\rho'} + p\frac{\cos^2\theta''}{\rho''}} = \frac{\cos\theta'' + q\cos\theta'}{\frac{\cos^2\theta''}{\rho''} + q\frac{\cos^2\theta'}{\rho'}}$$

Computing the N-th Taylor coefficient of a rational function

Duality lemma (link between C-recursive sequences and rational functions)
Let $A(x) = \sum_{n \geq 0} u_n x^n \in \mathbb{K}[[x]]$ be the generating function of $(u_n)_{n \geq 0}$.
The following assertions are equivalent:

(i) $(u_n)_{n \geq 0}$ is C-recursive, with characteristic polynomial Γ of degree d;

(ii) $A(x)$ is rational, $A = \frac{P}{Q}$ with $P \in \mathbb{K}[x]_{<d}$ and $Q = \mathsf{rev}(\Gamma) := \Gamma(\frac{1}{x})x^d$.

Computing the N-th Taylor coefficient of a rational function

Duality lemma (link between C-recursive sequences and rational functions)
Let $A(x) = \sum_{n\geq 0} u_n x^n \in \mathbb{K}[[x]]$ be the generating function of $(u_n)_{n\geq 0}$.
The following assertions are equivalent:

(i) $(u_n)_{n\geq 0}$ is C-recursive, with characteristic polynomial Γ of degree d;

(ii) $A(x)$ is rational, $A = \frac{P}{Q}$ with $P \in \mathbb{K}[x]_{<d}$ and $Q = \mathsf{rev}(\Gamma) := \Gamma(\frac{1}{x})x^d$.

▷ The *denominator* of A encodes a *recurrence* for $(u_n)_{n\geq 0}$; the *numerator* encodes *initial conditions*.

Computing the N-th Taylor coefficient of a rational function

Duality lemma (link between C-recursive sequences and rational functions)
Let $A(x) = \sum_{n\geq 0} u_n x^n \in \mathbb{K}[[x]]$ be the generating function of $(u_n)_{n\geq 0}$.
The following assertions are equivalent:

(i) $(u_n)_{n\geq 0}$ is C-recursive, with characteristic polynomial Γ of degree d;

(ii) $A(x)$ is rational, $A = \frac{P}{Q}$ with $P \in \mathbb{K}[x]_{<d}$ and $Q = \mathsf{rev}(\Gamma) := \Gamma(\frac{1}{x})x^d$.

▷ The *denominator* of A encodes a *recurrence* for $(u_n)_{n\geq 0}$; the *numerator* encodes *initial conditions*.

▷ Generating function of $(F_n)_{n\geq 0}$ given by $F_0 = a, F_1 = b, F_{n+2} = F_{n+1} + F_n$ is $(a + (b-a)x)/(1 - x - x^2)$. Here $\Gamma = x^2 - x - 1$ and $P = a + (b-a)x$.

Computing the N-th Taylor coefficient of a rational function

Duality lemma (link between C-recursive sequences and rational functions)
Let $A(x) = \sum_{n\geq 0} u_n x^n \in \mathbb{K}[[x]]$ be the generating function of $(u_n)_{n\geq 0}$.
The following assertions are equivalent:

(i) $(u_n)_{n\geq 0}$ is C-recursive, with characteristic polynomial Γ of degree d;

(ii) $A(x)$ is rational, $A = \frac{P}{Q}$ with $P \in \mathbb{K}[x]_{<d}$ and $Q = \mathsf{rev}(\Gamma) := \Gamma(\frac{1}{x})x^d$.

▷ The *denominator* of A encodes a *recurrence* for $(u_n)_{n\geq 0}$; the *numerator* encodes *initial conditions*.

▷ Generating function of $(F_n)_{n\geq 0}$ given by $F_0 = a, F_1 = b, F_{n+2} = F_{n+1} + F_n$ is $(a + (b-a)x)/(1 - x - x^2)$. Here $\Gamma = x^2 - x - 1$ and $P = a + (b-a)x$.

Corollary (of Fiduccia's algorithm + Duality lemma)
N-th Taylor coeff. of $\frac{P}{Q} \in \mathbb{K}(x)_d$ in $O(\mathsf{M}(d)\log N) = \tilde{O}(d \cdot \log N)$ ops. in $\mathbb{K}$

Computing the first N coefficients of a C-recursive sequence

Problem: Given $d, N \in \mathbb{N}$ with $N \gg d$ and the first d terms $u_0, \ldots, u_{d-1}$ of a C-recursive sequence of order d, compute the next terms $u_d, \ldots, u_N$

Computing the first N coefficients of a C-recursive sequence

Problem: Given $d, N \in \mathbb{N}$ with $N \gg d$ and the first d terms $u_0, \ldots, u_{d-1}$ of a C-recursive sequence of order d, compute the next terms $u_d, \ldots, u_N$

Naive algorithm: unroll the recurrence $O(dN) \subseteq O(N^2)$

Computing the first N coefficients of a C-recursive sequence

Problem: Given $d, N \in \mathbb{N}$ with $N \gg d$ and the first d terms $u_0, \ldots, u_{d-1}$ of a C-recursive sequence of order d, compute the next terms $u_d, \ldots, u_N$

Naive algorithm: unroll the recurrence $O(dN) \subseteq O(N^2)$

▷ By duality lemma: $\sum_{i \geq 0} u_i x^i$ is rational $P(x)/Q(x)$, with Q given by the input recurrence, and $\deg(P) < \deg(Q) = d$

Computing the first N coefficients of a C-recursive sequence

Problem: Given $d, N \in \mathbb{N}$ with $N \gg d$ and the first d terms $u_0, \ldots, u_{d-1}$ of a C-recursive sequence of order d, compute the next terms $u_d, \ldots, u_N$

Naive algorithm: unroll the recurrence $O(dN) \subseteq O(N^2)$

▷ By duality lemma: $\sum_{i \geq 0} u_i x^i$ is rational $P(x)/Q(x)$, with Q given by the input recurrence, and $\deg(P) < \deg(Q) = d$

Example (Fibonacci): $F_{i+2} = F_{i+1} + F_i \iff \sum_i F_i x^i = \dfrac{F_0 + (F_1 - F_0)x}{1 - x - x^2}$

Computing the first N coefficients of a C-recursive sequence

Problem: Given $d, N \in \mathbb{N}$ with $N \gg d$ and the first d terms $u_0, \ldots, u_{d-1}$ of a C-recursive sequence of order d, compute the next terms $u_d, \ldots, u_N$

Naive algorithm: unroll the recurrence $O(dN) \subseteq O(N^2)$

▷ By duality lemma: $\sum_{i \geq 0} u_i x^i$ is rational $P(x)/Q(x)$, with Q given by the input recurrence, and $\deg(P) < \deg(Q) = d$

Example (Fibonacci): $F_{i+2} = F_{i+1} + F_i \iff \sum_i F_i x^i = \dfrac{F_0 + (F_1 - F_0)x}{1 - x - x^2}$

A first algorithm:

- Compute (P, Q) from recurrence and $u_0, \ldots, u_{d-1}$ $O(\mathsf{M}(d))$
- Expand P/Q modulo x^{N+1} using Newton iteration $O(\mathsf{M}(N))$

Computing the first N coefficients of a C-recursive sequence

Problem: Given $d, N \in \mathbb{N}$ with $N \gg d$ and the first d terms $u_0, \ldots, u_{d-1}$ of a C-recursive sequence of order d, compute the next terms $u_d, \ldots, u_N$

Naive algorithm: unroll the recurrence $O(dN) \subseteq O(N^2)$

▷ By duality lemma: $\sum_{i \geq 0} u_i x^i$ is rational $P(x)/Q(x)$, with Q given by the input recurrence, and $\deg(P) < \deg(Q) = d$

Example (Fibonacci): $F_{i+2} = F_{i+1} + F_i \iff \sum_i F_i x^i = \dfrac{F_0 + (F_1 - F_0)x}{1 - x - x^2}$

A first algorithm:

- Compute (P, Q) from recurrence and $u_0, \ldots, u_{d-1}$ $O(\mathsf{M}(d))$
- Expand P/Q modulo x^{N+1} using Newton iteration $O(\mathsf{M}(N))$

A faster algorithm [Shoup, 1991]:

- Compute (P, Q) from recurrence and $u_0, \ldots, u_{d-1}$ $O(\mathsf{M}(d))$
- Compute $R(x) := 1/Q \bmod x^d$; set $c_0 := \sum_{j=0}^{d-1} u_j x^j$ $O(\mathsf{M}(d))$
- For $s = 0, \ldots, \lceil N/d \rceil - 1$ compute $c_{s+1} := -R \cdot [Q \cdot c_s]_d^{2d-1}$ $O\left(\frac{N}{d}\mathsf{M}(d)\right)$
- Return $\sum_{s=0}^{\lceil N/d \rceil} c_s(x) x^{sd} \bmod x^N$

Computing the N-th coefficient of a rational function, revisited

Pb: Given $P, Q \in \mathbb{K}[x]$ with $\deg(P) < \deg(Q) =: d$ and $N \in \mathbb{N}$, compute

$$u_N = [x^N] \frac{P(x)}{Q(x)}$$

Computing the N-th coefficient of a rational function, revisited

Pb: Given $P, Q \in \mathbb{K}[x]$ with $\deg(P) < \deg(Q) =: d$ and $N \in \mathbb{N}$, compute

$$u_N = [x^N] \frac{P(x)}{Q(x)}$$

▷ [Fiduccia, 1985] + duality lemma: fast algorithm $\sim 3\,\mathsf{M}(d) \log N$

Computing the N-th coefficient of a rational function, revisited

Pb: Given $P, Q \in \mathbb{K}[x]$ with $\deg(P) < \deg(Q) =: d$ and $N \in \mathbb{N}$, compute

$$u_N = [x^N] \frac{P(x)}{Q(x)}$$

- ▷ [Fiduccia, 1985] + duality lemma: fast algorithm $\sim 3\,\mathsf{M}(d)\log N$
- ▷ [B., Mori, 2021]: (direct) faster algorithm $\sim 2\,\mathsf{M}(d)\log N$

Computing the N-th coefficient of a rational function, revisited

Pb: Given $P, Q \in \mathbb{K}[x]$ with $\deg(P) < \deg(Q) =: d$ and $N \in \mathbb{N}$, compute

$$u_N = [x^N] \frac{P(x)}{Q(x)}$$

▷ [Fiduccia, 1985] + duality lemma: fast algorithm $\sim 3\,\mathsf{M}(d)\log N$
▷ [B., Mori, 2021]: (direct) faster algorithm $\sim 2\,\mathsf{M}(d)\log N$

Idea ("Graeffe iteration"): if $U(x) := P(x)Q(-x)$ and $V(x^2) := Q(x)Q(-x)$,

$$u_N = [x^N] \frac{P(x)Q(-x)}{Q(x)Q(-x)} = [x^N] \frac{U(x)}{V(x^2)}.$$

Computing the N-th coefficient of a rational function, revisited

Pb: Given $P, Q \in \mathbb{K}[x]$ with $\deg(P) < \deg(Q) =: d$ and $N \in \mathbb{N}$, compute

$$u_N = [x^N] \frac{P(x)}{Q(x)}$$

- ▷ [Fiduccia, 1985] + duality lemma: fast algorithm $\sim 3\,\mathsf{M}(d) \log N$
- ▷ [B., Mori, 2021]: (direct) faster algorithm $\sim 2\,\mathsf{M}(d) \log N$

Idea ("Graeffe iteration"): if $U(x) := P(x)Q(-x)$ and $V(x^2) := Q(x)Q(-x)$,

$$u_N = [x^N] \frac{P(x)Q(-x)}{Q(x)Q(-x)} = [x^N] \frac{U(x)}{V(x^2)}.$$

- ▷ Writing $U(x) = U_{\mathrm{e}}(x^2) + xU_{\mathrm{o}}(x^2)$, we have

$$u_N = \begin{cases} [x^N] \frac{U_{\mathrm{e}}(x^2)}{V(x^2)}, & \text{if } N \text{ is even} \\ [x^N] \frac{xU_{\mathrm{o}}(x^2)}{V(x^2)}, & \text{else.} \end{cases}$$

$$= \begin{cases} [x^{N/2}] \frac{U_{\mathrm{e}}(x)}{V(x)}, & \text{if } N \text{ is even} \\ [x^{(N-1)/2}] \frac{U_{\mathrm{o}}(x)}{V(x)}, & \text{else.} \end{cases}$$

Computing the N-th coefficient of a rational function, revisited

Pb: Given $P, Q \in \mathbb{K}[x]$ with $\deg(P) < \deg(Q) =: d$ and $N \in \mathbb{N}$, compute

$$u_N = [x^N] \frac{P(x)}{Q(x)}$$

- [Fiduccia, 1985] + duality lemma: fast algorithm $\sim 3\,\mathsf{M}(d)\log N$
- [B., Mori, 2021]: (direct) faster algorithm $\sim 2\,\mathsf{M}(d)\log N$

Idea ("Graeffe iteration"): if $U(x) := P(x)Q(-x)$ and $V(x^2) := Q(x)Q(-x)$,

$$u_N = [x^N] \frac{P(x)Q(-x)}{Q(x)Q(-x)} = [x^N] \frac{U(x)}{V(x^2)}.$$

- Writing $U(x) = U_{\mathrm{e}}(x^2) + xU_{\mathrm{o}}(x^2)$, we have

$$u_N = \begin{cases} [x^N] \frac{U_{\mathrm{e}}(x^2)}{V(x^2)}, & \text{if } N \text{ is even} \\ [x^N] \frac{xU_{\mathrm{o}}(x^2)}{V(x^2)}, & \text{else.} \end{cases}$$

$$= \begin{cases} [x^{N/2}] \frac{U_{\mathrm{e}}(x)}{V(x)}, & \text{if } N \text{ is even} \\ [x^{(N-1)/2}] \frac{U_{\mathrm{o}}(x)}{V(x)}, & \text{else.} \end{cases}$$

- Algorithm: repeat this reduction until $N \geq 1$ $\quad 2\,\mathsf{M}(d)\lceil\log(N+1)\rceil$

Algorithm 1 OneCoeff

Input: $P(x)$, $Q(x)$, N

Output: $[x^N] \frac{P(x)}{Q(x)}$

Assumptions: $Q(0)$ invertible and $\deg(P) < \deg(Q) =: d$

while $N \geq 1$ **do**
  $U(x) \leftarrow P(x)Q(-x)$ ▷ $U = \sum_{i=0}^{2d-1} U_i x^i$
  if N is even **then**
    $P(x) \leftarrow \sum_{i=0}^{d-1} U_{2i} x^i$
  else
    $P(x) \leftarrow \sum_{i=0}^{d-1} U_{2i+1} x^i$
  end if
  $A(x) \leftarrow Q(x)Q(-x)$ ▷ $A = \sum_{i=0}^{2d} A_i x^i$
  $Q(x) \leftarrow \sum_{i=0}^{d} A_{2i} x^i$
  $N \leftarrow \lfloor N/2 \rfloor$
end while
return $P(0)/Q(0)$

Computing the N-th term of a C-recursive sequence, revisited

Pb: Given $N \in \mathbb{N}$, $u_0, \ldots, u_{d-1} \in \mathbb{K}$, and the recurrence

$$u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

compute u_N

Computing the N-th term of a C-recursive sequence, revisited

Pb: Given $N \in \mathbb{N}$, $u_0, \ldots, u_{d-1} \in \mathbb{K}$, and the recurrence

$$u_{n+d} = c_{d-1} u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

compute u_N

▷ [Fiduccia, 1985] binary powering in $\mathbb{K}[x]/(\Gamma)$, with $\Gamma = x^d - \sum_{i=0}^{d-1} c_i x^i$
$\sim 3\,\mathsf{M}(d) \log N$

Computing the N-th term of a C-recursive sequence, revisited

Pb: Given $N \in \mathbb{N}$, $u_0, \ldots, u_{d-1} \in \mathbb{K}$, and the recurrence

$$u_{n+d} = c_{d-1} u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

compute u_N

- ▷ [Fiduccia, 1985] binary powering in $\mathbb{K}[x]/(\Gamma)$, with $\Gamma = x^d - \sum_{i=0}^{d-1} c_i x^i$ $\sim 3\,\mathsf{M}(d) \log N$
- ▷ [B., Mori, 2021]: Use $[x^N] \frac{P(x)}{Q(x)}$ and duality lemma $\sim 2\,\mathsf{M}(d) \log N$
  - Appropriate choice is: $Q = \mathrm{rev}(\Gamma)$, and P such that $\frac{P}{Q} = \sum_{i=0}^{d-1} u_i x^i \bmod x^d$

Computing the N-th term of a C-recursive sequence, revisited

Pb: Given $N \in \mathbb{N}$, $u_0, \ldots, u_{d-1} \in \mathbb{K}$, and the recurrence

$$u_{n+d} = c_{d-1} u_{n+d-1} + \cdots + c_0 u_n, \qquad n \geq 0,$$

compute u_N

- ▷ [Fiduccia, 1985] binary powering in $\mathbb{K}[x]/(\Gamma)$, with $\Gamma = x^d - \sum_{i=0}^{d-1} c_i x^i$ $\sim 3\,\mathsf{M}(d) \log N$
- ▷ [B., Mori, 2021]: Use $[x^N] \frac{P(x)}{Q(x)}$ and duality lemma $\sim 2\,\mathsf{M}(d) \log N$
  - Appropriate choice is: $Q = \mathrm{rev}(\Gamma)$, and P such that $\frac{P}{Q} = \sum_{i=0}^{d-1} u_i x^i \bmod x^d$
- ▷ Even for Fibonacci seq, [B., Mori, 2021] is competitive with state-of-the-art

Algorithm 2 OneTerm

Input: rec. $u_{n+d} = c_{d-1}u_{n+d-1} + \cdots + c_0 u_n$, $(n \geq 0)$, and $u_0, \ldots, u_{d-1}$, N
Output: u_N
Assumptions: $\Gamma(x) = x^d - \sum_{i=0}^{d-1} c_i x^i$ with $c_0 \neq 0$

1: $Q(x) \leftarrow x^d \Gamma(1/x)$
2: $P(x) \leftarrow (u_0 + \cdots + u_{d-1}x^{d-1}) \cdot Q(x) \bmod x^d$
3: **return** $[x^N]P(x)/Q(x)$ ▷ using Algorithm 1

▷ Advantage: faster than Fiduccia's algorithm

▷ in FFT-mode, $\sim \frac{2}{3}\,\mathrm{M}(d)\log N$ versus $\sim \frac{5}{3}\,\mathrm{M}(d)\log N$ [Shoup, NTL, 1995] and $\sim \frac{13}{12}\,\mathrm{M}(d)\log N$ [Mihăilescu, 2008]

▷ Drawback: computes a single u_N, while Fiduccia computes a whole slice

Application: new algorithm for the Fibonacci numbers

$$F_0 = 0, F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n,\ n \geq 0.$$

Application: new algorithm for the Fibonacci numbers

$$F_0 = 0, F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n,\ n \geq 0.$$

- Generating function $\sum_{n \geq 0} F_n x^n$ is $x/(1 - x - x^2)$.

Application: new algorithm for the Fibonacci numbers

$$F_0 = 0, F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n,\ n \geq 0.$$

- Generating function $\sum_{n\geq 0} F_n x^n$ is $x/(1-x-x^2)$.
- Thus, the coefficient $F_N = [x^N]\ \frac{x}{1-x-x^2}$ is equal to

$$[x^N]\ \frac{x(1+x-x^2)}{1-3x^2+x^4} = \begin{cases} [x^{\frac{N}{2}}]\ \frac{x}{1-3x+x^2}, & \text{if } N \text{ is even,} \\ [x^{\frac{N-1}{2}}]\ \frac{1-x}{1-3x+x^2}, & \text{else.} \end{cases}$$

Application: new algorithm for the Fibonacci numbers

$$F_0 = 0, F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n,\ n \geq 0.$$

- Generating function $\sum_{n\geq 0} F_n x^n$ is $x/(1-x-x^2)$.
- Thus, the coefficient $F_N = [x^N]\ \frac{x}{1-x-x^2}$ is equal to

$$[x^N]\ \frac{x(1+x-x^2)}{1-3x^2+x^4} = \begin{cases} [x^{\frac{N}{2}}]\ \frac{x}{1-3x+x^2}, & \text{if } N \text{ is even,} \\ [x^{\frac{N-1}{2}}]\ \frac{1-x}{1-3x+x^2}, & \text{else.} \end{cases}$$

- The computation of F_N is reduced to that of a coefficient of the form

$$[x^N]\ \frac{a+bx}{1-cx+x^2} = [x^N]\ \frac{(a+bx)(1+cx+x^2)}{1-(c^2-2)x^2+x^4}$$

which is equal to

$$\begin{cases} [x^{\frac{N}{2}}]\ \frac{a+(bc+a)x}{1-(c^2-2)x+x^2}, & \text{if } N \text{ is even,} \\ [x^{\frac{N-1}{2}}]\ \frac{(ac+b)+bx}{1-(c^2-2)x+x^2}, & \text{else.} \end{cases}$$

Algorithm 3 NewFibo

Input: N
Output: F_N
Assumptions: $N \geq 2$

1: $c \leftarrow 3$
2: **if** N is even **then**
3: $\quad [a, b] \leftarrow [0, 1]$
4: **else**
5: $\quad [a, b] \leftarrow [1, -1]$
6: **end if**
7: $N \leftarrow \lfloor N/2 \rfloor$
8: **while** $N > 1$ **do**
9: $\quad$ **if** N is even **then**
10: $\quad\quad b \leftarrow a + b \cdot c$
11: $\quad$ **else**
12: $\quad\quad a \leftarrow b + a \cdot c$
13: $\quad$ **end if**
14: $\quad c \leftarrow c^2 - 2$
15: $\quad N \leftarrow \lfloor N/2 \rfloor$
16: **end while**
17: **return** $b + a \cdot c$

Computation of $F_{43} = 433\,494\,437$ using the new algorithm

N	a	b	c
21	1	-1	3
10	$1 \times 3 - 1 = 2$		$3^2 - 2 = 7$
5		$(-1) \times 7 + 2 = -5$	$7^2 - 2 = 47$
2	$2 \times 47 - 5 = 89$		$47^2 - 2 = 2207$
1		$(-5) \times 2207 + 89$	$2207^2 - 2 = 4870847$
		$= -10946$	
0	$89 \times 4870847 - 10946$		
	$= 433494437$		

Algorithm 4 NewFiboPowerOfTwo

Input: N
Output: F_N
Assumptions: $N \geq 2$ and N is a power of 2

1: $[b, c] \leftarrow [1, 3]$
2: $N \leftarrow \lfloor N/2 \rfloor$
3: **while** $N > 2$ **do**
4: $\quad b \leftarrow b \cdot c$
5: $\quad c \leftarrow c^2 - 2$
6: $\quad N \leftarrow \lfloor N/2 \rfloor$
7: **end while**
8: **return** $b \cdot c$

▷ This is exactly [Cull, Holloway, 1989, Fig. 6], also [Knuth, 1969]

```
fib(n)
    f ← 1
    l ← 3
    for i = 2 to (log n − 1)
        f ← f * l
        l ← l * l − 2
    f ← f * l
    return f
```

Example: N-th term of the Fibonacci sequence

$u_{20} = 6765$; puis, pour $p = 20$, la même formule donnera

$$u_{100} = 354\,224\,848\,179\,261\,915\,075.$$

Les relations particulières qui précèdent pourraient être déduites directement de formules analogues à la formule (2), mais plus générales, telles que les suivantes :

$$u_{p+q+r-2} = u_{p-1}u_q u_r + u_p u_{q-1} u_r + u_p u_q u_{r-1} + u_{p-2} u_{q-2} u_{r-2},$$
$$u_{p+q+r-2} = u_p u_q u_r + u_p u_{q-1} u_{r-1} + u_{p-1} u_q u_{r-1} + u_{p-1} u_{q-1} u_r - u_{p-1} u_{q-1} u_{r-1};$$

Il suffit de faire $r = 1$ dans cette dernière pour retrouver la relation (2).

Rosace.

Soient les deux séries P_n (de Fibonacci) et Q_n, dont le $n^{\text{ième}}$ terme est au-dessous de l'indice n :

n	0	1	2	3	4	5	6	7	8	9	10
P_n	0	1	1	2	3	5	8	13	21	34	55
Q_n	2	1	3	4	7	11	18	29	47	76	123

La série Q_n procède des deux premiers termes 2, 1, de la même façon que la série P_n procède des termes 0 et 1. Les termes correspondants de ces deux séries présentent une remarquable analogie avec les fonctions trigonométriques *sinus* et *cosinus* :

$$P_n = \frac{1}{\sqrt{5}}[\alpha^n - (-1)^n \beta^n], \qquad Q_n = \alpha^n + (-1)^n \beta^n,$$

où $\alpha = \frac{\sqrt{5}+1}{2}$, $\beta = \frac{\sqrt{5}-1}{2}$.

De ces formules, on déduit les relations suivantes, correspondant à celles qui existent entre le *sinus* et le *cosinus* :

(1) $$Q_n^2 - 5P_n^2 = 4(-1)^n,$$
(2) $$2P_{m+n} = P_m Q_n + Q_m P_n,$$
(3) $$2P_{m-n} = (-1)^n (P_m Q_n - Q_m P_n),$$
(4) $$P_{2m} = P_m Q_m,$$
(5) $$2Q_{m+n} = Q_m Q_n + 5P_m P_n,$$
(6) $$2Q_{m-n} = (-1)^n (Q_m Q_n - 5P_m P_n),$$
(7) $$P_{-m} = (-1)^{m+1} P_m.$$
(8) $$Q_{-m} = (-1)^m Q_m,$$
(9) $$2Q_{2m} = Q_m^2 + 5P_m^2,$$
(10) $$Q_{2m} = Q_m^2 - 2(-1)^m.$$

Par combinaison de (2) et (3), nous avons

(11) $$P_{m+n} = P_m Q_n - (-1)^n P_{m-n}.$$

A l'aide de ces formules, notamment de (2), (4), (10) et (11), nous pourrons calculer très rapidement un terme P_n, d'une manière analogue à celle du sinus d'un arc multiple. Par exemple,

$$Q_{2^n} = 1, 3, 7, 47, \ldots,$$

chaque terme étant le carré du précédent, moins 2.

Pour une étude de ces séries et de quelques autres de ce genre, *voir* E. Lucas, *Théorie des fonctions numériques simplement périodiques* (*A. J. M.*, t. I, et *M. A. B.*, 1878).

E.-B. Escott (Grand Rapids, Mich.).

La série de Fibonacci, comme toute suite récurrente, est susceptible de deux modes de calcul : l'un le mode ordinaire, continu, qui n'est que l'application répétée de la formule même de récurrence; l'autre discontinu, auquel, dans le cas particulier envisagé, on est conduit par les considérations suivantes :

En partant des identités :

$$u_n + u_{n+1} - u_{n+2} = 0, \qquad \ldots, \qquad u_{n+m} + u_{n+m-1} - u_{n+m-1} = 0,$$

on a facilement $u_{n+m+1} = u_{n+1} u_{m+1} + u_n u_m$. Spécialement, pour $m = n$, j'aurai $u_{2n+1} = u_{n+1}^2 + u_n^2$ (égalité qui donne la décomposition en deux carrés d'un terme quelconque de rang impair de la série de Fibonacci), et de même, pour $m = n + 1$,

$$u_{2n+2} = u_{n+1}(u_{n+1} + 2u_n).$$

Ainsi les deux termes u_n et u_{n+1}, calculés par un moyen quelconque, suffisent à faire connaître les deux termes u_{2n+1} et u_{2n+2}, ceux-ci à faire connaître u_{4n+3} et u_{4n+4}, d'où l'on passera à u_{8n+7} et u_{8n+8}, etc. De proche en proche, on arrivera donc aux termes d'indices $(n+1)2^k - 1$ et $(n+1)2^k$, où n et k représentent des entiers à notre choix. Pour fixer les idées, soit à calculer la valeur numérique du $900^{\text{ième}}$ terme de la série. Le nombre 900 est égal à $7 \times 2^7 + 4$, j'aurai à chercher, de la façon qui vient d'être indiquée, u_{895} et u_{896}, puis, par le moyen de l'échelle de relation, je passerai successivement aux termes u_{897}, u_{898}, u_{899} et enfin u_{900}. Ainsi, les valeurs $u_{13} = 233$, $u_{14} = 377$ étant prises comme point de départ, j'en tirerai $u_{27} = 196418$, $u_{28} = 317811$, puis

$$u_{55} = 139583862445, \qquad u_{56} = 225851433717.$$

An exercise for next time (15/10/2025)

(a) Show that if $P \in \mathbb{K}[x]$ has degree d, then the sequence $(P(n))_{n\geq 0}$ is C-recursive, and admits $(x-1)^{d+1}$ as a characteristic polynomial.

(b) Deduce that P can be evaluated at the $N \gg d$ points $1, 2, \ldots, N$ in $O(N\,\mathsf{M}(d)/d)$ operations in $\mathbb{K}$.

II.

INTERMEZZO

Tellegen's transposition principle

Tellegen's transposition principle

Let $\mathbf{M}$ be an $m \times n$ matrix, with no zero rows and no zero columns. Any linear algorithm of complexity L that computes the matrix-vector product $\mathbf{M} \cdot \mathbf{v}$ can be transformed into a linear algorithm of complexity

$$L - n + m$$

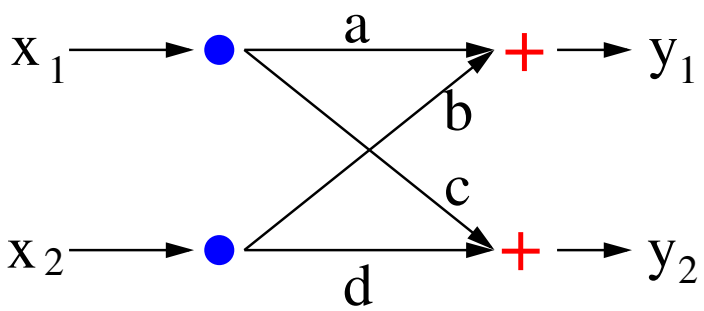
that computes the transposed matrix-vector product $\mathbf{M}^{\mathbf{T}} \cdot \mathbf{w}$.

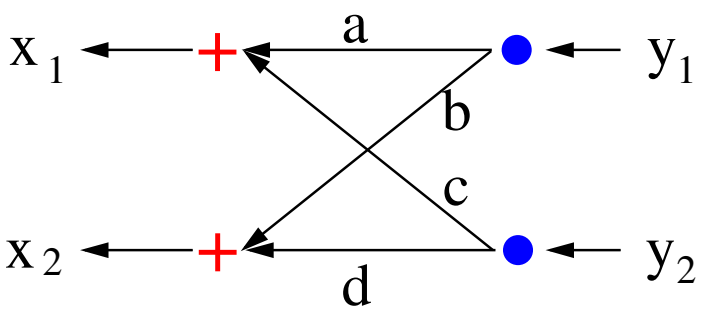
Tellegen's transposition principle

Let $\mathbf{M}$ be an $m \times n$ matrix, with no zero rows and no zero columns. Any linear algorithm of complexity L that computes the matrix-vector product $\mathbf{M} \cdot \mathbf{v}$ can be transformed into a linear algorithm of complexity

$$L - n + m$$

that computes the transposed matrix-vector product $\mathbf{M}^{\mathbf{T}} \cdot \mathbf{w}$.





$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \qquad \begin{bmatrix} a & c \\ b & d \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Tellegen's transposition principle

Let $\mathbf{M}$ be an $m \times n$ matrix, with no zero rows and no zero columns. Any linear algorithm of complexity L that computes the matrix-vector product $\mathbf{M} \cdot \mathbf{v}$ can be transformed into a linear algorithm of complexity

$$L - n + m$$

that computes the transposed matrix-vector product $\mathbf{M}^{\mathbf{T}} \cdot \mathbf{w}$.

- Precise formulations depend on the *model of computation* (DAG, SLP, RAM)
- Originates from electrical networks theory [Tellegen, '52; Bordewijk, '56]
- Introduced in computer algebra by [Fiduccia, '72; Hopcroft, Musinski, '73]
- In complexity theory, rediscovered by [Kaminski, Kirkpatrick, Bshouty, '88]
- Reverse mode in automatic differentiation [Canny, Kaltofen, Yagati '89]
- Automatic program transposition [B., Lecerf, Salvy '03; De Feo, Schost, '10]

A particular case

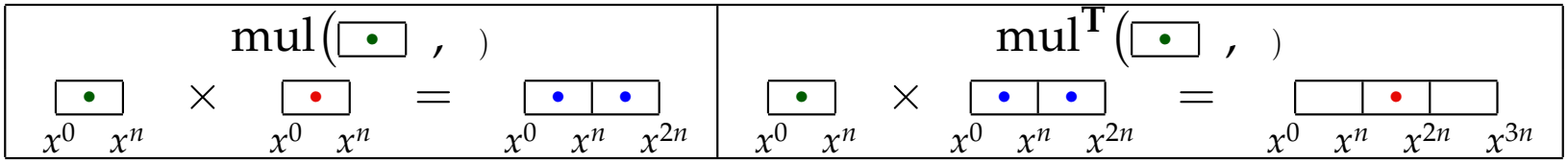
Suppose that the naive algorithm $\mathcal{N}$ is used to compute $M \cdot v$.
Define its dual $\mathcal{N}^T$ as the naive algorithm for computing $M^T \cdot w$.
Then:

$$\begin{aligned} \mathcal{N} &\text{ uses } m(n-1) \text{ ops. } \pm \text{ and } mn \text{ ops. } \times \\ \mathcal{N}^T &\text{ uses } n(m-1) \text{ ops. } \pm \text{ and } mn \text{ ops. } \times. \end{aligned}$$

$\longrightarrow$ Tellegen's theorem is trivially true for generic matrices

$\longrightarrow$ it becomes interesting when M is structured or sparse

Transposed polynomial multiplication = middle product (MP)



Example:

$$\mathrm{mul}\,(2+3x, a+bx) \qquad \mathrm{mul}^{\mathbf{T}}\left(3+2x, a+bx+cx^2\right)$$

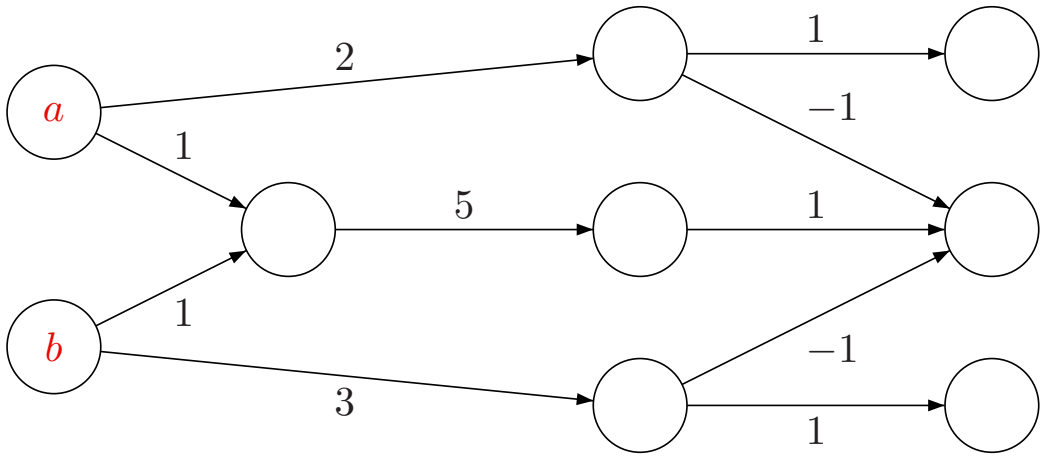
$$\begin{bmatrix} 2 & 0 \\ 3 & 2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2a \\ 3a+2b \\ 3b \end{bmatrix} \qquad \begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a+3b \\ 2b+3c \end{bmatrix}$$

Theorem [Hanrot-Quercia-Zimmerman, 2002]
From any linear algorithm for multiplication in degree n of cost $\mathsf{M}(\mathbf{n})$, one can derive a linear algorithm for the $(n, 2n)$ MP, of cost $\mathsf{M}(\mathbf{n}) + \mathbf{O}(\mathbf{n})$.

▷ Particular instance of Tellegen's theorem, for Toeplitz-band matrices

A DAG computing $(2+3x)(a+bx)$ *à la Karatsuba*

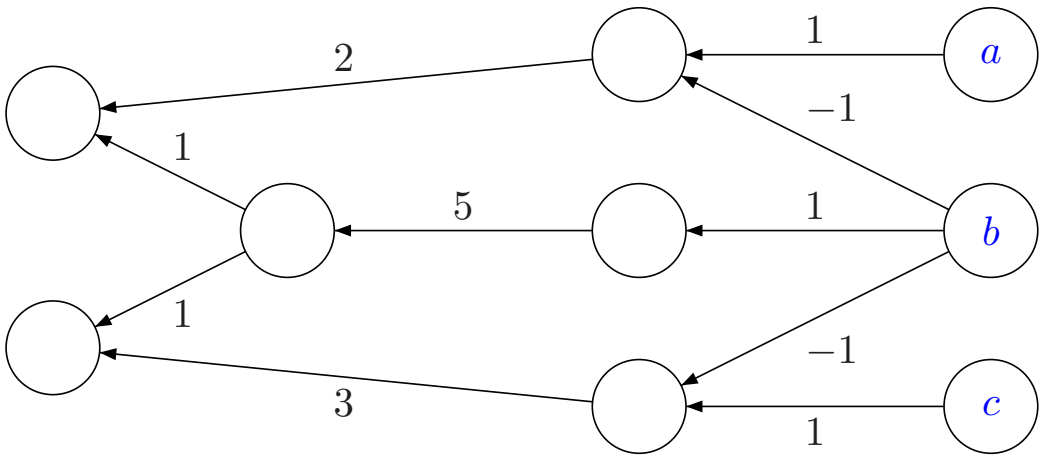
$$\mathrm{mul}\,(2+3x, a+bx)$$



$$\begin{bmatrix} 2 & 0 \\ 3 & 2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2a \\ 3a+2b \\ 3b \end{bmatrix} = \begin{bmatrix} 2a \\ 5(a+b)-2a-3b \\ 3b \end{bmatrix}$$

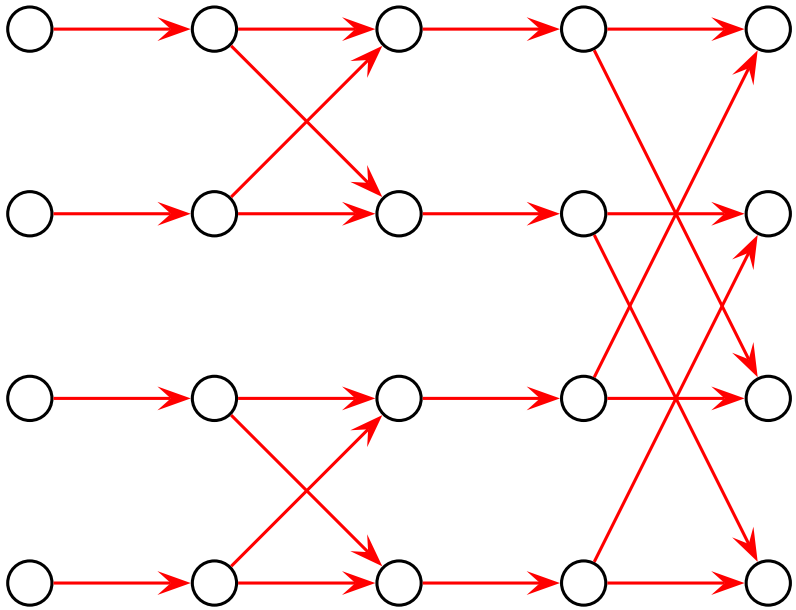
The transposed DAG computes the middle product of $3+2x$ and $a+bx+cx^2$ *à la Karatsuba*

$$\mathrm{mul}^{\mathbf{T}}\left(3+2x, a+bx+cx^2\right)$$



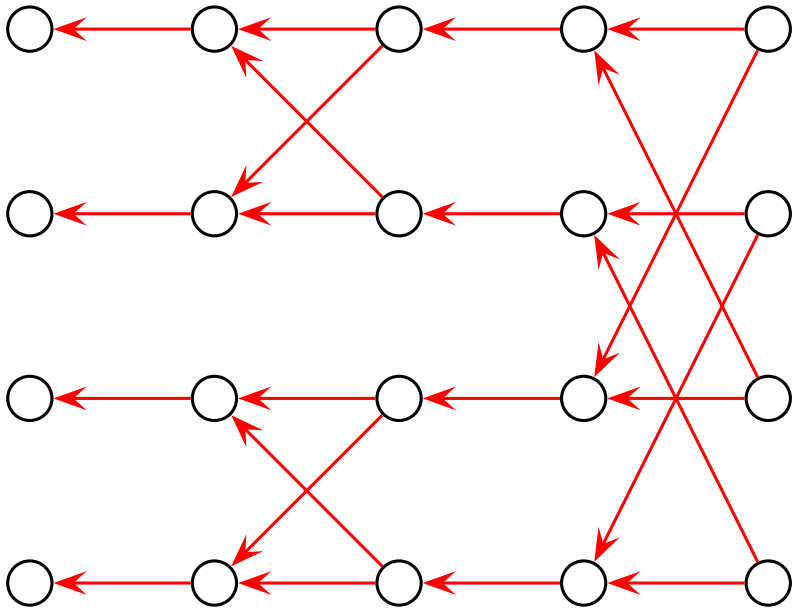
$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a+3b \\ 2b+3c \end{bmatrix} = \begin{bmatrix} 2(a-b)+5b \\ 3(b-c)+5b \end{bmatrix}$$

Duality between two classes of FFT algorithms



The Cooley-Tukey decimation-in-time DFT, on 4 points

Duality between two classes of FFT algorithms



The Gentleman-Sande decimation-in-frequency DFT, on 4 points

Toy example: automatic discovery of Horner's rule

$$[p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Toy example: automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$
$$[p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Toy example: automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Toy example: automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Input x_0.
$p_0 \leftarrow x_0$;
for j **from** 1 **to** n **do**
  $p_j \leftarrow p_{j-1}$;
  $p_j \leftarrow a p_j$;
Output $p = [p_0, \ldots, p_n]$.

Toy example: automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Input x_0.	**Input** $p = [p_0, \ldots p_n]$.
$p_0 \leftarrow x_0$;	
for j **from** 1 **to** n **do**	
$p_j \leftarrow p_{j-1}$;	
$p_j \leftarrow a p_j$;	
Output $p = [p_0, \ldots, p_n]$.	**Output** x_0.

Toy example: automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Input x_0.
$p_0 \leftarrow x_0$;
for j **from** 1 **to** n **do**
  $p_j \leftarrow p_{j-1}$;
  $p_j \leftarrow a p_j$;
Output $p = [p_0, \ldots, p_n]$.

Input $p = [p_0, \ldots p_n]$.
for j **from** n **downto** 1 **do**

Output x_0.

Toy example: automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Input x_0.
$p_0 \leftarrow x_0$;
for j **from** 1 **to** n **do**
  $p_j \leftarrow p_{j-1}$;
  $p_j \leftarrow a p_j$;
Output $p = [p_0, \ldots, p_n]$.

Input $p = [p_0, \ldots p_n]$.
for j **from** n **downto** 1 **do**
  $p_j \leftarrow a p_j$;

Output x_0.

Toy example: automatic discovery of Horner's rule

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Input x_0.
$p_0 \leftarrow x_0$;
for j **from** 1 **to** n **do**
$p_j \leftarrow p_{j-1}$;
$p_j \leftarrow a p_j$;
Output $p = [p_0, \ldots, p_n]$.

Input $p = [p_0, \ldots p_n]$.
for j **from** n **downto** 1 **do**
$p_j \leftarrow a p_j$;
$p_{j-1} \leftarrow p_j + p_{j-1}$;
Output x_0.

Toy example: automatic discovery of Horner's rule

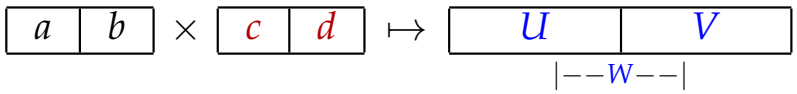
$$\mathbf{M} = [1, a, \ldots, a^n]$$

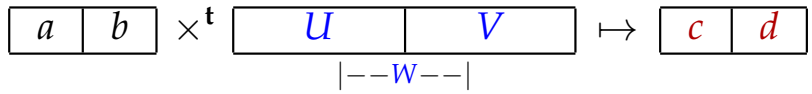
$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

Input x_0.
$p_0 \leftarrow x_0$;
for j **from** 1 **to** n **do**
$p_j \leftarrow p_{j-1}$;
$p_j \leftarrow a p_j$;
Output $p = [p_0, \ldots, p_n]$.

Input $p = [p_0, \ldots p_n]$.
for j **from** n **downto** 1 **do**
$p_j \leftarrow a p_j$;
$p_{j-1} \leftarrow p_j + p_{j-1}$;
$x_0 \leftarrow p_0$;
Output x_0.

Karatsuba's algorithm and its transpose





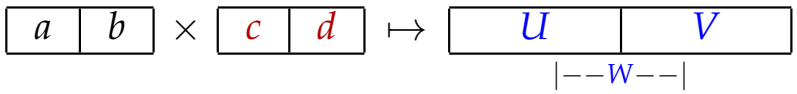
mul

Input (c, d).

$e \leftarrow c + d$;
$U \leftarrow \mathrm{mul}(a, c)$;
$V \leftarrow \mathrm{mul}(b, d)$;
$W \leftarrow \mathrm{mul}(a + b, e)$;
$W \leftarrow W - U - V$;

Output (U, V, W).

Karatsuba's algorithm and its transpose

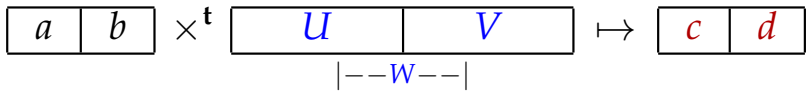


mul

Input (c, d).

$e \leftarrow c + d$;
$U \leftarrow \mathrm{mul}(a, c)$;
$V \leftarrow \mathrm{mul}(b, d)$;
$W \leftarrow \mathrm{mul}(a + b, e)$;
$W \leftarrow W - U - V$;

Output (U, V, W).



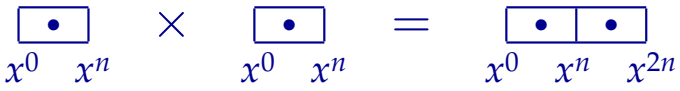
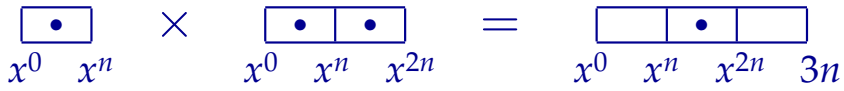
$\mathrm{mul}^{\mathbf{T}}$

Input (U, V, W).

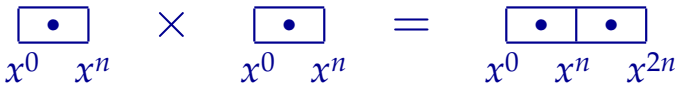
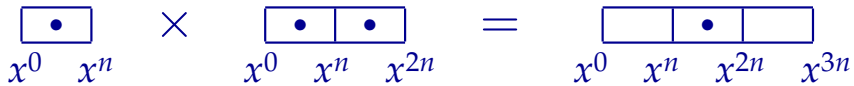
$V \leftarrow V - W$;
$U \leftarrow U - W$;
$e \leftarrow \mathrm{mul}^{\mathbf{T}}(a + b, W)$;
$d \leftarrow \mathrm{mul}^{\mathbf{T}}(b, V)$;
$c \leftarrow \mathrm{mul}^{\mathbf{T}}(a, U)$;
$c \leftarrow c + e$;
$d \leftarrow d + e$;

Output (c, d).

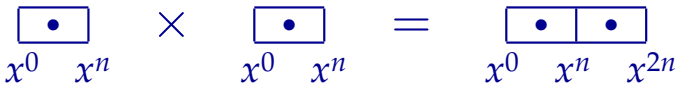
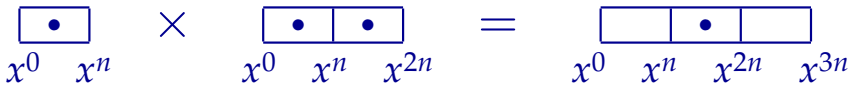
“Tellegen’s Polynomial Dictionary”

direct problem	transposed problem
multiplication	middle product
	

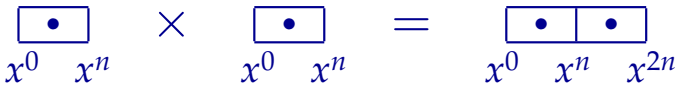
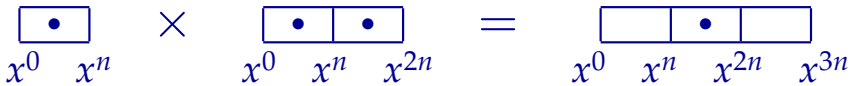
"Tellegen's Polynomial Dictionary"

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \bmod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$

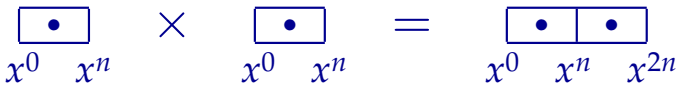
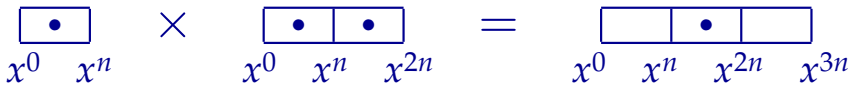
"Tellegen's Polynomial Dictionary"

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \bmod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$
multipoint evaluation	generalized power sums
$P \mapsto (P(a_0), \ldots, P(a_{n-1}))$	$(p_0, \ldots, p_{n-1}) \mapsto (\sum p_i, \ldots, \sum p_i a_i^{n-1})$

“Tellegen’s Polynomial Dictionary”

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \bmod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$
multipoint evaluation	generalized power sums
$P \mapsto (P(a_0), \ldots, P(a_{n-1}))$	$(p_0, \ldots, p_{n-1}) \mapsto (\sum p_i, \ldots, \sum p_i a_i^{n-1})$
shift of polynomials	evaluation in falling factorial basis
$P(x) \mapsto P(x+1)$	$P = \sum a_i x^{\underline{i}} \mapsto (P(0), \ldots, P(n-1))$

"Tellegen's Polynomial Dictionary"

direct problem	transposed problem
multiplication	middle product
	
division with remainder	extension of recurrences
$A \mapsto A \bmod P$	$(a_0, \ldots, a_{n-1}) \mapsto (a_0, \ldots, a_{2n-1})$
multipoint evaluation	generalized power sums
$P \mapsto (P(a_0), \ldots, P(a_{n-1}))$	$(p_0, \ldots, p_{n-1}) \mapsto (\sum p_i, \ldots, \sum p_i a_i^{n-1})$
shift of polynomials	evaluation in falling factorial basis
$P(x) \mapsto P(x+1)$	$P = \sum a_i x^{\underline{i}} \mapsto (P(0), \ldots, P(n-1))$
composition of power series	power projection
$f(x) \mapsto f(g(x)) \bmod x^n$	$\ell \in (\mathbb{K}[x]_{<n})^\star \mapsto (\ell(1), \ell(g), \ldots, \ell(g^{n-1}))$
$\vdots$	$\vdots$

III.

POWER SERIES COMPOSITION

Composition of series, first remarks

Pb: Given $N \in \mathbb{N}$, $f \in \mathbb{K}[[x]]$ and $g \in x\mathbb{K}[[x]]$ compute $f(g(x)) \bmod x^N$

- Naive approach (by Horner scheme) $O(N\,\mathsf{M}(N)) = \tilde{O}(N^2)$
- [Paterson and Stockmeyer, 1973] $O\left(N^{\frac{\theta+1}{2}}\right) = O(N^{1.68})$

  Baby-steps giant-steps technique: split f in chunks of length $\sqrt{N}$
- [Brent and Kung, 1978] $O\left(\sqrt{N \log N}\,\mathsf{M}(N)\right) = \tilde{O}(N^{1.5})$

  Similar splitting + Taylor's formula

Quasi-optimal composition of power series, in special cases

Pb: Given $N \in \mathbb{N}$, $f \in \mathbb{K}[[x]]$ and $g \in x\mathbb{K}[[x]]$ compute $f(g(x)) \bmod x^N$

- $f = 1/(1-x)$, $f = \exp(x)$, $f = \log(1-x)$ (by Newton iteration) $O(\mathsf{M}(N))$
- If g is a polynomial in $x\mathbb{K}[x]$ [Brent and Kung, 1978] $O(\mathsf{M}(N)\log N)$
- More generally if g is algebraic [van der Hoeven, 2002] $O(\mathsf{M}(N)\log N)$
- If $g = \exp(x) - 1$ or $g = \log(1+x)$ [Gerhard, 2000] $O(\mathsf{M}(N)\log N)$
- Many other cases [B., Salvy, Schost, 2008] $O(\mathsf{M}(N)\log N)$
  (via Tellegen's transposition principle)

A more general problem: Modular Composition

Problem: Given $f, g, h \in \mathbb{K}[x]_n$ compute $f(g(x)) \bmod h(x)$

Faster Modular Composition

VINCENT NEIGER, Sorbonne Université, France
BRUNO SALVY, Inria, France
ÉRIC SCHOST, University of Waterloo, Canada
GILLES VILLARD, CNRS, France

A new Las Vegas algorithm is presented for the composition of two polynomials modulo a third one, over an arbitrary field. When the degrees of these polynomials are bounded by n, the algorithm uses $O(n^{1.43})$ field operations, breaking through the 3/2 barrier in the exponent for the first time. The previous fastest algebraic algorithms, due to Brent and Kung in 1978, require $O(n^{1.63})$ field operations in general, and $n^{3/2+o(1)}$ field operations in the special case of power series over a field of large enough characteristic. If cubic-time matrix multiplication is used, the new algorithm runs in $n^{5/3+o(1)}$ operations, while previous ones run in $O(n^2)$ operations.

Our approach relies on the computation of a matrix of algebraic relations that is typically of small size. Randomization is used to reduce arbitrary input to this favorable situation.

CCS Concepts: • **Computing methodologies** → **Algebraic algorithms**; • **Theory of computation** → **Algebraic complexity theory;**

Additional Key Words and Phrases: Symbolic computation, algorithm, complexity, modular composition, multivariate polynomial, multivariate relation

ACM Reference Format:
Vincent Neiger, Bruno Salvy, Éric Schost, and Gilles Villard. 2024. Faster Modular Composition. *Journal of the ACM* 71, 2, Article 11 (April 2024), 79 pages. https://doi.org/10.1145/3638349

▷ $O(n^\kappa)$, where $4/3 \leq \kappa < 1.43$ depends on (rectangular) matrix exponent

Quasi-optimal modular composition, in a special case

Problem: Given $f, g, h \in \mathbb{F}_q[x]_{<n}$ ($q = p^k$), compute $f(g(x)) \bmod h(x)$

2008 49th Annual IEEE Symposium on Foundations of Computer Science

Fast modular composition in any characteristic

Kiran S. Kedlaya*
Department of Mathematics
MIT

Christopher Umans†
Department of Computer Science
Caltech

Abstract

We give an algorithm for modular composition of degree n univariate polynomials over a finite field $\mathbb{F}_q$ requiring $n^{1+o(1)} \log^{1+o(1)} q$ bit operations; this had earlier been achieved in characteristic $n^{o(1)}$ by Umans (2008). As an application, we obtain a randomized algorithm for factoring degree n polynomials over $\mathbb{F}_q$ requiring $(n^{1.5+o(1)} + n^{1+o(1)} \log q) \log^{1+o(1)} q$ bit operations, improving upon the methods of von zur Gathen & Shoup (1992) and Kaltofen & Shoup (1998). Our results also imply algorithms for irreducibility testing and computing minimal polynomials whose running times are best-possible, up to lower order terms.

backbone of numerous algorithms for computing with polynomials over finite fields, most notably the asymptotically fastest methods for polynomial factorization.

In contrast to other basic modular operations on polynomials (e.g modular multiplication), it is *not* possible to obtain an asymptotically fast algorithm for modular composition with fast algorithms for each step in the natural two step procedure (i.e., first compute $f(g(x))$, then reduce modulo $h(x)$). This is because $f(g(x))$ has n^2 terms, while we hope for a modular composition algorithm that uses only about $O(n)$ operations. Not surprisingly, it is by considering the overall operation (and beating n^2) that asymptotic gains are made in algorithms that employ modular composition.

Perhaps because nontrivial algorithms for modular com-

▷ $(n \cdot \log(q))^{1+o(1)}$ bit operations, but $(n \cdot p)^{1+o(1)}$ operations in $\mathbb{F}_q$

Quasi-optimal power series composition, general case

Pb: Given $N \in \mathbb{N}$, $f \in \mathbb{K}[[x]]$ and $g \in x\mathbb{K}[[x]]$ compute $f(g(x)) \bmod x^N$

Power Series Composition in Near-Linear Time

Yasunori Kinoshita * Baitian Li †

Abstract

We present an algebraic algorithm that computes the composition of two power series in $\tilde{O}(n)$ time complexity. The previous best algorithms are $O(n^{1+o(1)})$ by Kedlaya and Umans (FOCS 2008) and an $O(n^{1.43})$ algebraic algorithm by Neiger, Salvy, Schost and Villard (JACM 2023).

Our algorithm builds upon the recent Graeffe iteration approach to manipulate rational power series introduced by Bostan and Mori (SOSA 2021).

▷ $O(\mathsf{M}(N)\log N)$ operations in $\mathbb{K}$, without any restrictions on f, g or $\mathbb{K}$
▷ The algorithm is amazingly "simple" and beautiful
▷ It relies on (a bivariate extension of) [B., Mori, 2021] and on Tellegen's transposition principle

Quasi-optimal power series composition: proof strategy

Pb: Given $N \in \mathbb{N}$, $f \in \mathbb{K}[[x]]$ and $g \in x\mathbb{K}[[x]]$ compute $f(g(x)) \bmod x^N$

$O(\mathsf{M}(N) \log N)$ [Kinoshita, Li, 2024]

• Idea 1: Composition = (Power Projection)$^\mathsf{T}$; therefore, by Tellegen's principle, it is enough to solve PowerProjection in $O(\mathsf{M}(N) \log N)$

• Idea 2: Solving PowerProjection is equivalent to solving

$$[x^{N-1}] \frac{P(x)}{1 - yg(x)}$$

• Idea 3: Last problem can be solved using [B., Mori, 2021] and quasi-optimal multiplication in $\mathbb{K}[x, y]$ (e.g. via Kronecker substitution)

Quasi-optimal power series composition: proof strategy

Pb: Given $N \in \mathbb{N}$, $f \in \mathbb{K}[[x]]$ and $g \in x\mathbb{K}[[x]]$ compute $f(g(x)) \bmod x^N$

$O(\mathsf{M}(N)\log N)$ [Kinoshita, Li, 2024]

• Idea 1: Composition = (Power Projection)$^\mathsf{T}$; therefore, by Tellegen's principle, it is enough to solve PowerProjection in $O(\mathsf{M}(N)\log N)$

• Idea 2: Solving PowerProjection is equivalent to solving

$$[x^{N-1}]\frac{P(x)}{1-yg(x)}$$

• Idea 3: Last problem can be solved using [B., Mori, 2021] and quasi-optimal multiplication in $\mathbb{K}[x,y]$ (e.g. via Kronecker substitution)

▷ *Open question: can this algorithm be generalized to modular composition?*

Composition = (Power Projection)$^{\mathsf{T}}$ and Power Projection = N-th Coeff

Write $f(x) = u_0 + \cdots + u_{N-1}x^{N-1}$ and $\mathbf{u} = [u_0 \cdots u_{N-1}]^T$

Composition = (Power Projection)$^{\mathsf{T}}$ and Power Projection = N-th Coeff

Write $f(x) = u_0 + \cdots + u_{N-1}x^{N-1}$ and $\mathbf{u} = [u_0 \cdots u_{N-1}]^T$
Consider the $N \times N$ matrix A_g containing the coefficients of the powers of g:

$$A_g := \begin{bmatrix} [x^0]g(x)^0 & \cdots & [x^0]g(x)^{N-1} \\ \vdots & \ddots & \vdots \\ [x^{N-1}]g(x)^0 & \cdots & [x^{N-1}]g(x)^{N-1} \end{bmatrix}$$

Composition = (Power Projection)^T and Power Projection = N-th Coeff

Write $f(x) = u_0 + \cdots + u_{N-1}x^{N-1}$ and $\mathbf{u} = [u_0 \cdots u_{N-1}]^T$
Consider the $N \times N$ matrix A_g containing the coefficients of the powers of g:

$$A_g := \begin{bmatrix} [x^0]g(x)^0 & \cdots & [x^0]g(x)^{N-1} \\ \vdots & \ddots & \vdots \\ [x^{N-1}]g(x)^0 & \cdots & [x^{N-1}]g(x)^{N-1} \end{bmatrix}$$

▷ Let $\mathbf{v} = [v_0 \cdots v_{N-1}]^T$ be such that $\mathbf{v} = A_g \cdot \mathbf{u}$. Then

$$v_j = \sum_{i=0}^{N-1} u_i \cdot [x^j]g(x)^i = [x^j](f \circ g)$$

Composition = (Power Projection)^T and Power Projection = N-th Coeff

Write $f(x) = u_0 + \cdots + u_{N-1}x^{N-1}$ and $\mathbf{u} = [u_0 \cdots u_{N-1}]^T$
Consider the $N \times N$ matrix A_g containing the coefficients of the powers of g:

$$A_g := \begin{bmatrix} [x^0]g(x)^0 & \cdots & [x^0]g(x)^{N-1} \\ \vdots & \ddots & \vdots \\ [x^{N-1}]g(x)^0 & \cdots & [x^{N-1}]g(x)^{N-1} \end{bmatrix}$$

▷ Let $\mathbf{v} = [v_0 \cdots v_{N-1}]^T$ be such that $\mathbf{v} = A_g \cdot \mathbf{u}$. Then

$$v_j = \sum_{i=0}^{N-1} u_i \cdot [x^j]g(x)^i = [x^j](f \circ g)$$

▷ Let's look at the transposed map $\mathbf{v} \mapsto \mathbf{u} := A_g^T \cdot \mathbf{v}$

$$u_i = \sum_{j=0}^{N-1} v_j \cdot [x^j]g(x)^i = [x^{N-1}]\, P(x) \cdot g(x)^i,$$

where $P(x) := \sum_{j=0}^{N-1} v_j x^{N-1-j}$.

Composition = (Power Projection)^T and Power Projection = N-th Coeff

Write $f(x) = u_0 + \cdots + u_{N-1}x^{N-1}$ and $\mathbf{u} = [u_0 \cdots u_{N-1}]^T$
Consider the $N \times N$ matrix A_g containing the coefficients of the powers of g:

$$A_g := \begin{bmatrix} [x^0]g(x)^0 & \cdots & [x^0]g(x)^{N-1} \\ \vdots & \ddots & \vdots \\ [x^{N-1}]g(x)^0 & \cdots & [x^{N-1}]g(x)^{N-1} \end{bmatrix}$$

▷ Let $\mathbf{v} = [v_0 \cdots v_{N-1}]^T$ be such that $\mathbf{v} = A_g \cdot \mathbf{u}$. Then

$$v_j = \sum_{i=0}^{N-1} u_i \cdot [x^j]g(x)^i = [x^j](f \circ g)$$

▷ Let's look at the transposed map $\mathbf{v} \mapsto \mathbf{u} := A_g^T \cdot \mathbf{v}$

$$u_i = \sum_{j=0}^{N-1} v_j \cdot [x^j]g(x)^i = [x^{N-1}]\, P(x) \cdot g(x)^i,$$

where $P(x) := \sum_{j=0}^{N-1} v_j x^{N-1-j}$. Therefore,

$$\sum_{i=0}^{N-1} u_i y^i = [x^{N-1}]\, P(x) \cdot \sum_{i=0}^{N-1} g(x)^i y^i = [x^{N-1}]\, \frac{P(x)}{1 - yg(x)} \bmod y^N$$

Problem: Given $n = 2^s < N \leq 2n$, $P \in \mathbb{K}[x]_{<N}$ and $g \in \mathbb{K}[x]_{<N}$, compute

$$[x^n] \frac{P(x)}{1 - yg(x)} \bmod y^N$$

- [B., Mori, 2021] compute

$$[x^n] \frac{P_0(x,y)}{Q_0(x,y)} \longleftarrow [x^{n/2}] \frac{P_1(x,y)}{Q_1(x,y)} \longleftarrow \cdots \longleftarrow [x^1] \frac{P_s(x,y)}{Q_s(x,y)},$$

where $P_0 := P(x)$, $Q_0 := 1 - yg(x)$ and

$Q_i(x^2,y) = Q_{i-1}(x,y) \cdot Q_{i-1}(-x,y)$ and $P_i(x^2,y) + xR_i(x^2,y) = P_{i-1}(x,y) \cdot Q_{i-1}(-x,y)$

- (P_0, Q_0) have bidegree $(n,1)$, then (P_1, Q_1) have bidegree $(n/2, 2)$, etc
- Total cost: $O\left(\mathsf{M}(n \times 1) + \mathsf{M}\left(\frac{n}{2} \times 2\right)\right) + \cdots + \mathsf{M}(1 \times n)\right) = O(\mathsf{M}(n) \log n)$

Problem: Given $g \in \mathbb{K}[x]_{<N}$ and $\ell : \mathbb{K}[x]/(x^N) \to \mathbb{K}$, compute $\left(\ell(g^i)\right)_{i=0}^{N-1}$

Algorithm 1 (PowProj)
Input: n, m, $P(x,y), Q(x,y) \in \mathbb{A}[x,y]$
Output: $[x^{n-1}](P(x,y)/Q(x,y)) \bmod y^m$

Require: $[x^0y^0]Q(x,y) = 1$
1: **while** $n > 1$ **do**
2: $\quad U(x,y) \leftarrow P(x,y)Q(-x,y) \bmod x^n \bmod y^m$
3: $\quad$ **if** $n-1$ is even **then**
4: $\quad\quad P(x,y) \leftarrow \sum_{i=0}^{\lceil n/2 \rceil - 1} x^i [s^{2i}] U(s,y)$
5: $\quad$ **else**
6: $\quad\quad P(x,y) \leftarrow \sum_{i=0}^{\lceil n/2 \rceil - 1} x^i [s^{2i+1}] U(s,y)$
7: $\quad$ **end if**
8: $\quad A(x,y) \leftarrow Q(x,y)Q(-x,y) \bmod x^n \bmod y^m$
9: $\quad Q(x,y) \leftarrow \sum_{i=0}^{\lceil n/2 \rceil - 1} x^i [s^{2i}] A(s,y)$
10: $\quad n \leftarrow \lceil n/2 \rceil$
11: **end while**
12: **return** $(P(0,y)/Q(0,y)) \bmod y^m$

▷ $O(\mathsf{M}(N) \log N)$ operations in $\mathbb{K}$, without any restrictions on ℓ, g or $\mathbb{K}$

Problem: Given $f \in \mathbb{K}[x]_{<N}$ and $g \in \mathbb{K}[x]_{<N}$ compute $f(g(x)) \bmod x^N$

▷ Writing $\mathcal{F}_{a,b}(\sum_{i\geq 0} c_i(x)y^i) := \sum_{i=a}^{b-1} c_i(x)y^i$, we have

$$f(g(x)) \bmod x^N = \left[x^{N-1}\right] \frac{f(1/y)\cdot y^{N-1}}{1-y\cdot g(x)} = \mathcal{F}_{N-1,N}\left(\frac{f(1/y)\cdot y^{N-1}}{1-y\cdot g(x)}\right)$$

Algorithm 2 (Comp)
Input: n, d, m, $P(y) \in \mathbb{A}[y]$, $Q(x,y) \in \mathbb{A}[x,y]$
Output: $\mathcal{F}_{d,m}(P(y)/Q(x,y)) \bmod x^n$

Require: $[x^0y^0]Q(x,y) = 1$
if $n = 1$ **then**
  $C(y) \leftarrow (P(y)/Q(0,y)) \bmod y^m$
  return $\sum_{i=d}^{m-1} y^{i-d}[t^i]C(t)$
else
  $A(x,y) \leftarrow Q(x,y)Q(-x,y) \bmod x^n \bmod y^m$
  $V(x,y) \leftarrow \sum_{i=0}^{\lceil n/2\rceil-1} x^i[s^{2i}]A(s,y)$
  $e \leftarrow \max\{0, d - \deg_y Q(x,y)\}$
  $W(x,y) \leftarrow \mathsf{Comp}(\lceil n/2\rceil, e, m, P(y), V(x,y))$
  $B(x,y) \leftarrow W(x^2,y)Q(-x,y)$
  return $\sum_{i=d-e}^{m-e-1} y^{i-(d-e)}[t^i]B(x,t) \bmod x^n$
end if

▷ $O(\mathsf{M}(N)\log N)$ operations in $\mathbb{K}$, without any restrictions on f, g or $\mathbb{K}$

Takeaways

▷ High-order terms of C-recursive sequences are classically computed in *quasi-optimal complexity* by binary powering

Takeaways

▷ High-order terms of C-recursive sequences are classically computed in *quasi-optimal complexity* by binary powering

▷ The *best known solution before 2020* (Fiduccia's algorithm) reduces this problem to fast modular polynomial exponentiation

Takeaways

▷ High-order terms of C-recursive sequences are classically computed in *quasi-optimal complexity* by binary powering

▷ The *best known solution before 2020* (Fiduccia's algorithm) reduces this problem to fast modular polynomial exponentiation

▷ Today we saw a *better algorithmic solution* (B.-Mori algorithm), based on duality and on computing $[x^N]P/Q$ by Graeffe iteration

Takeaways

▷ High-order terms of C-recursive sequences are classically computed in *quasi-optimal complexity* by binary powering

▷ The *best known solution before 2020* (Fiduccia's algorithm) reduces this problem to fast modular polynomial exponentiation

▷ Today we saw a *better algorithmic solution* (B.-Mori algorithm), based on duality and on computing $[x^N]P/Q$ by Graeffe iteration

▷ Moreover, bivariate B.-Mori algorithm + Tellegen's transposition principle allow *quasi-optimal composition of power series* (Kinoshita-Li algorithm)

Schönhage's Golden Rule 8

RULE 8: *The development of fast algorithms is slow !*