N-th term of a linearly recurrent sequence

Alin Bostan

informatics mathematics
Inria

MPRI

C-2-22

4 November 2024

Exercise 1

The aim of this exercise is to prove algorithmically the following identity:

$$\sqrt[3]{\sqrt[3]{2}-1} = \sqrt[3]{\frac{1}{9}} - \sqrt[3]{\frac{2}{9}} + \sqrt[3]{\frac{4}{9}}.$$

Let $a = \sqrt[3]{2}$ and $b = \sqrt[3]{1/9}$.

(a) Determine a polynomial annihilating $c = 1 - a + a^2$, using a resultant.

(b) Deduce a polynomial annihilating the RHS, using another resultant.

(c) Show that the polynomial computed in (b) also annihilates the LHS.

(d) Conclude.

Exercise 1(a)

The aim of this exercise is to prove algorithmically the following identity:

$$\sqrt[3]{\sqrt[3]{2}-1} = \sqrt[3]{\frac{1}{9}} - \sqrt[3]{\frac{2}{9}} + \sqrt[3]{\frac{4}{9}}.$$

Let $a = \sqrt[3]{2}$ and $b = \sqrt[3]{1/9}$.

(a) Determine a polynomial annihilating $c = 1 - a + a^2$, using a resultant.

▷ Maple solution (all resultant computations *can* be done by hand):

```
> (* (a): a = 2^(1/3), c = 1 - 2^(1/3) + 4^(1/3) *)
> Pa := A^3 - 2:
> Pc := resultant(Pa, C - (1-A+A^2), A);
```

$$C^3 - 3\,C^2 + 9\,C - 9$$

Exercise 1(b)

The aim of this exercise is to prove algorithmically the following identity:

$$\sqrt[3]{\sqrt[3]{2}-1} = \sqrt[3]{\frac{1}{9}} - \sqrt[3]{\frac{2}{9}} + \sqrt[3]{\frac{4}{9}}.$$

Let $a = \sqrt[3]{2}$ and $b = \sqrt[3]{1/9}$.

(b) Deduce a polynomial annihilating the RHS, using another resultant.

```
> (* (a): b = (1/9)^(1/3), c = 1 - 2^(1/3) + 4^(1/3), RHS = b*c *)
> Pb:=B^3-1/9;
> numer(subs(B = RHS/C, Pb));
```

$$-C^3 + 9\,\mathrm{RHS}^3$$

```
> PR:=primpart(resultant(Pc, %, C));
```

$$\mathrm{RHS}^9 + 3\,\mathrm{RHS}^6 + 3\,\mathrm{RHS}^3 - 1$$

Exercise 1(c)

The aim of this exercise is to prove algorithmically the following identity:

$$\sqrt[3]{\sqrt[3]{2}-1} = \sqrt[3]{\frac{1}{9}} - \sqrt[3]{\frac{2}{9}} + \sqrt[3]{\frac{4}{9}}.$$

Let $a = \sqrt[3]{2}$ and $b = \sqrt[3]{1/9}$.

(c) Show that the polynomial computed in (b) also annihilates the LHS.

```
> (* (c): LHS = (a-1)^(1/3) *)
> PL := resultant(x - 1, subs(A = LHS^3 + x, Pa), x);
```

$$\mathrm{LHS}^9 + 3\,\mathrm{LHS}^6 + 3\,\mathrm{LHS}^3 - 1$$

```
> evalb(subs(LHS = x, PL) = subs(RHS = x, PR));
```

true

Exercise 1(d)

The aim of this exercise is to prove algorithmically the following identity:

$$\sqrt[3]{\sqrt[3]{2}-1} = \sqrt[3]{\frac{1}{9}} - \sqrt[3]{\frac{2}{9}} + \sqrt[3]{\frac{4}{9}}.$$

Let $a = \sqrt[3]{2}$ and $b = \sqrt[3]{1/9}$.

(d) Conclude.

```
> (* (d): PL is non-decreasing *)
> factor(diff(PL, LHS));
```

$$9\,\mathsf{LHS}^2\left(\mathsf{LHS}+1\right)^2\left(\mathsf{LHS}^2-\mathsf{LHS}+1\right)^2$$

- The polynomial annihilating both LHS and RHS has exactly 1 real root

Exercise 2

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 2

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).

1. $O(\mathsf{M}(N))$ for computing the first N terms
2. $\Phi(F, G) := G^2 - F$ 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 2

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).

1. $O(\mathsf{M}(N))$ for computing the first N terms
2. $\Phi(F, G) := G^2 - F$ 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 $U + \mathrm{rem}(T/U, X^N)/2$.
3. $C(N) \leq C(N/2) + O(\mathsf{M}(N))$ leads to $O(\mathsf{M}(N))$.

COMPUTING TERMS OF RECURRENT SEQUENCES

Goal, motivation, examples, main results

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

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]

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

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]

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]

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]

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]

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

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]

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]

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

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]

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

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
- 27/01/2025: two middle rows

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

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 is measured in terms of field operations (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 is measured in terms of field operations (arithmetic complexity)

First part: 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 is measured in terms of field operations (arithmetic complexity)

First part: 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 $(u_n)_{n\geq 0}$

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: modular exponentiation in $R = \mathcal{M}_d(\mathbb{K})$:

- if $M \in \mathcal{M}_d(\mathbb{K})$, then M^N in $O(d^\theta \log N)$ ops. in $\mathbb{K}$

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: modular exponentiation in $R = \mathbb{K}[x]/(Q)$:

- if $P, Q \in \mathbb{K}[x]_{<d}$, then $P^N \bmod Q$ in $O(\mathsf{M}(d) \log N)$ ops. in $\mathbb{K}$

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: modular exponentiation in $R = \mathbb{Z}/A\mathbb{Z}$:

- 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 Rules

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 Rules

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 for $(C^T)^N$ $\quad O(d^\theta \log(N))$

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 for $(C^T)^N$ $\quad O(d^\theta \log(N))$
- ▷ [Fiduccia, 1985] binary powering in $\mathbb{K}[x]/(\Gamma)$, with $\Gamma = x^d - \sum_{i=0}^{d-1} c_i x^i$

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

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

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 for $(C^T)^N$ $\quad O(d^\theta \log(N))$
- ▷ [Fiduccia, 1985] binary powering in $\mathbb{K}[x]/(\Gamma)$, with $\Gamma = x^d - \sum_{i=0}^{d-1} c_i x^i$

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

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

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

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.

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 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, having Γ as characteristic polynomial of degree d;

(ii) $A(x)$ is rational, of the form $A = P/Q$ for some $P \in \mathbb{K}[x]_{<d}$, where $Q := \mathsf{rev}_d(\Gamma) = \Gamma(\frac{1}{x})x^d$.

Computing the N-th 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, having Γ as characteristic polynomial of degree d;

(ii) $A(x)$ is rational, of the form $A = P/Q$ for some $P \in \mathbb{K}[x]_{<d}$, where $Q := \mathsf{rev}_d(\Gamma) = \Gamma(\frac{1}{x})x^d$.

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

Computing the N-th 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, having Γ as characteristic polynomial of degree d;

(ii) $A(x)$ is rational, of the form $A = P/Q$ for some $P \in \mathbb{K}[x]_{<d}$, where $Q := \mathsf{rev}_d(\Gamma) = \Gamma(\frac{1}{x})x^d$.

▷ 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 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, having Γ as characteristic polynomial of degree d;

(ii) $A(x)$ is rational, of the form $A = P/Q$ for some $P \in \mathbb{K}[x]_{<d}$, where $Q := \mathsf{rev}_d(\Gamma) = \Gamma(\frac{1}{x})x^d$.

▷ 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: N-th Taylor coeff. of $\frac{P}{Q} \in \mathbb{K}(x)_d$ in $\sim 3\,\mathsf{M}(d)\log N$ 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: With $U(x) := P(x)Q(-x)$ 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: With $U(x) := P(x)Q(-x)$ 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: With $U(x) := P(x)Q(-x)$ 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 $N \geq 1$ $2\,\mathsf{M}(d)\lceil \log(N+1) \rceil$

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 $\sim 2\,\mathsf{M}(d)\log N$

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 $\frac{P}{Q} = \sum_{i=0}^{d-1} u_i x^i \bmod x^d$

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 (14/11/2024)

(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 $\mathbb{K}$.

Tellegen's transposition principle

Tellegen's transposition principle

Let $\mathbf{M}$ be a $\mathsf{m} \times \mathsf{n}$ 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

$$\mathsf{L} - \mathsf{n} + \mathsf{m}$$

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

▷ A precise formulation depends on the *model of computation*.

A particular case

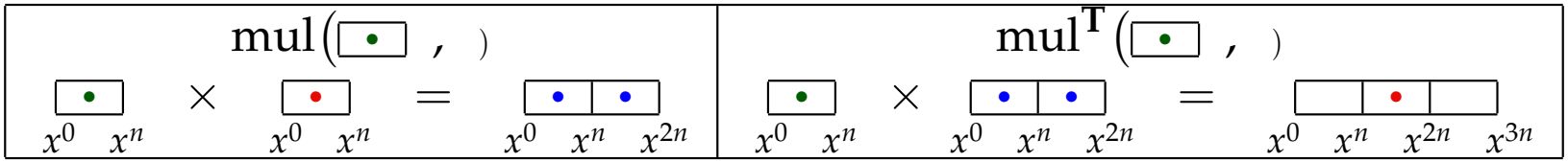
Suppose that the naive algorithm $\mathcal{N}$ is used to compute $M \cdot v$.
Define its dual $\mathcal{N}^T$ as the naive algorithm for computing $M^T \cdot w$.
Then:

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

$\longrightarrow$ 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:

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

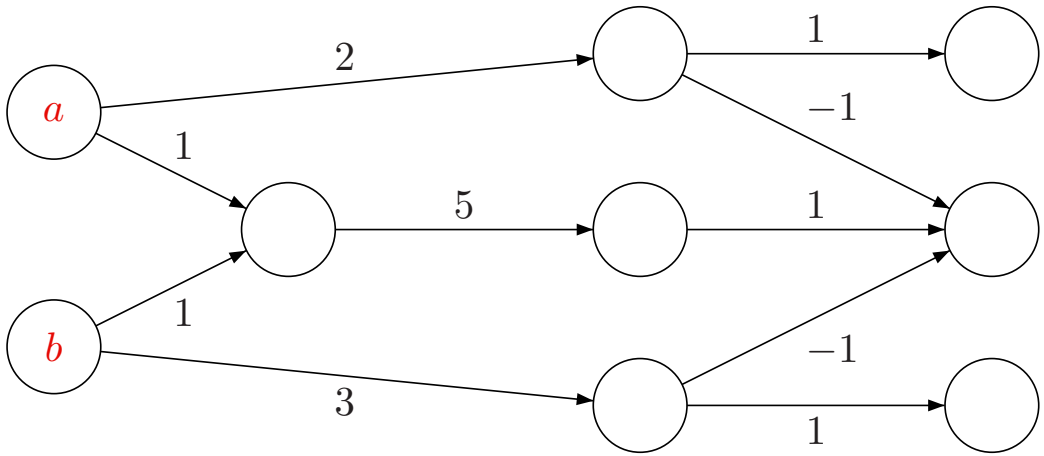
$$\begin{bmatrix} 2 & 0 \\ 3 & 2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2a \\ 3a+2b \\ 3b \end{bmatrix} \qquad \begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a+3b \\ 2b+3c \end{bmatrix}$$

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*

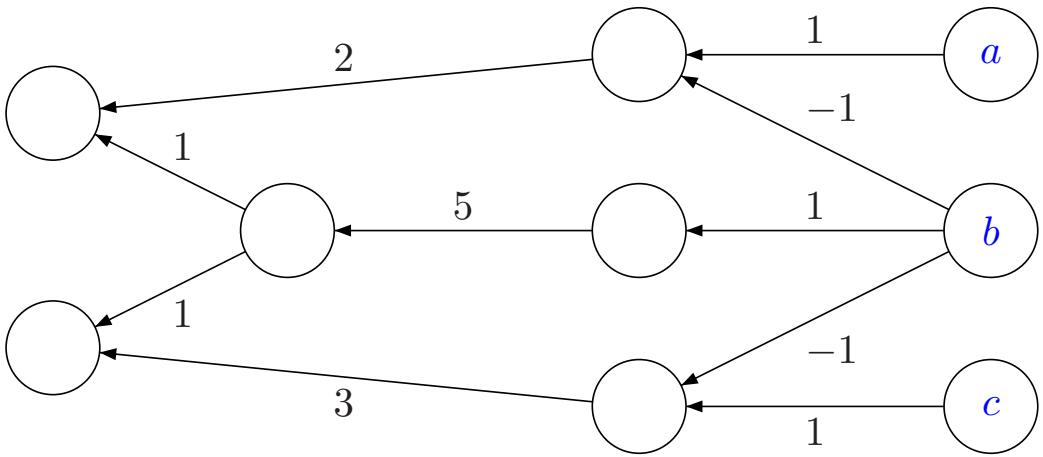
$$\mathrm{mul}\,(2+3x, a+bx)$$



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

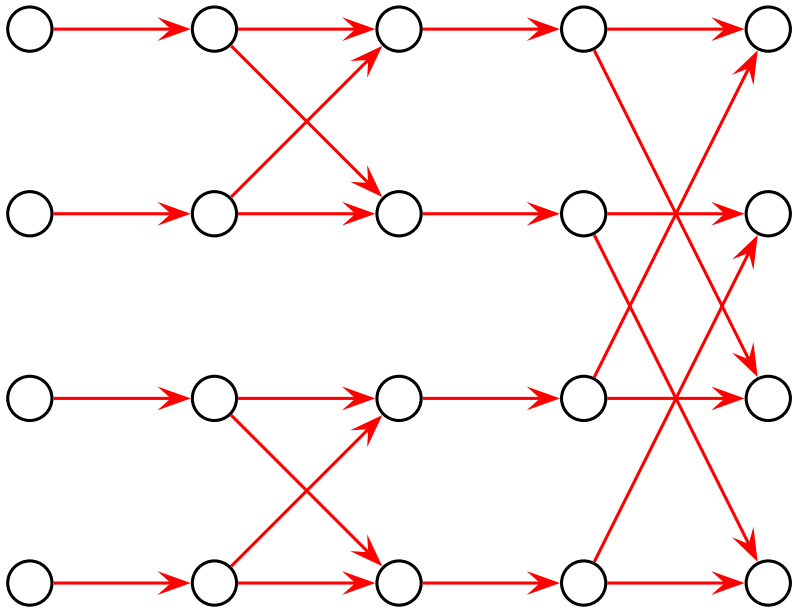
The transposed DAG computes the middle product of $3+2x$ and $a+bx+cx^2$ *à la Karatsuba*

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



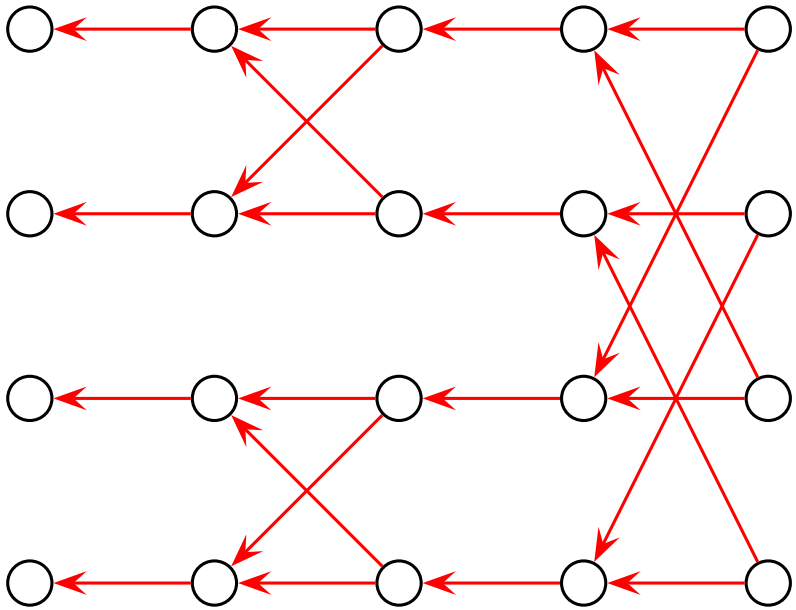
$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2a+3b \\ 2b+3c \end{bmatrix} = \begin{bmatrix} 2(a-b)+5b \\ 3(b-c)+5b \end{bmatrix}$$

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

Automatic discovery of Horner's rule

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

Automatic discovery of Horner's rule

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

Automatic discovery of Horner's rule

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

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

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 $p = [p_0, \ldots, p_n]$.

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.

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.

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.

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.

Automatic discovery of Horner's rule

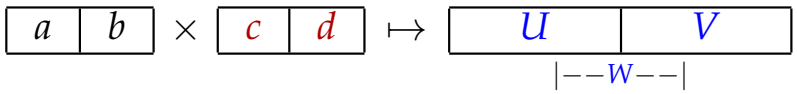
$$\mathbf{M} = [1, a, \ldots, a^n]$$

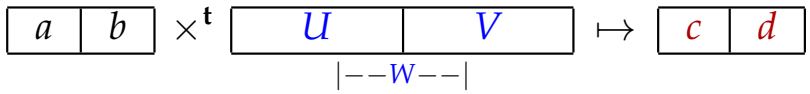
$$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0] \qquad [p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$$

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





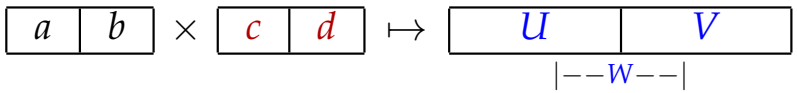
mul

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

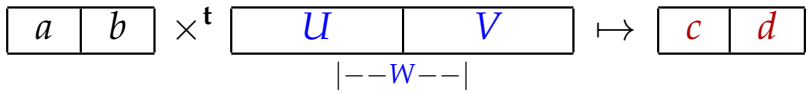


mul

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).



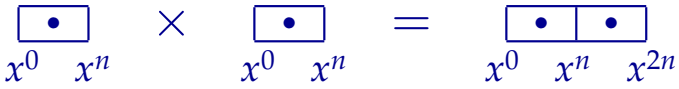
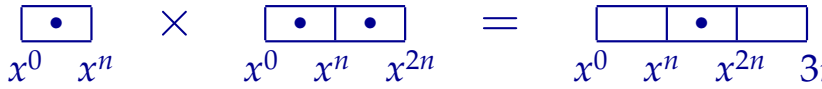
$\mathrm{mul}^{\mathbf{T}}$

Input (U, V, W).

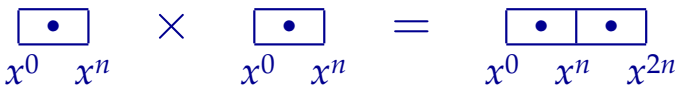
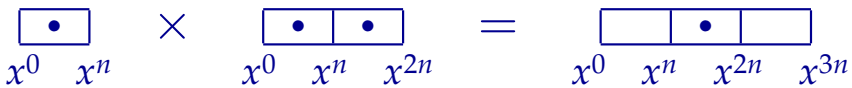
$V \leftarrow V - W$;
$U \leftarrow U - W$;
$e \leftarrow \mathrm{mul}^{\mathbf{T}}(a + b, W)$;
$d \leftarrow \mathrm{mul}^{\mathbf{T}}(b, V)$;
$c \leftarrow \mathrm{mul}^{\mathbf{T}}(a, U)$;
$c \leftarrow c + e$;
$d \leftarrow d + e$;

Output (c, d).

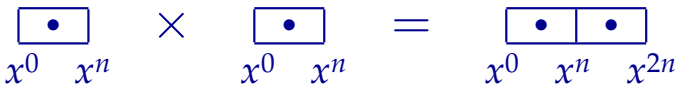
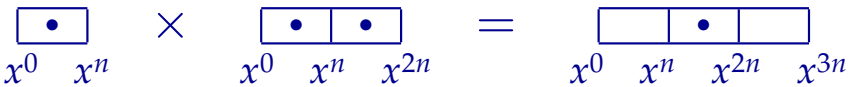
Tellegen's Polynomial Dictionary

direct problem	transposed problem
multiplication	middle product
	

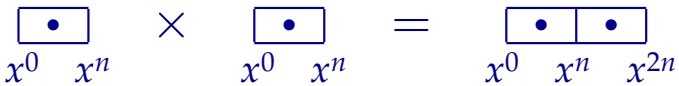
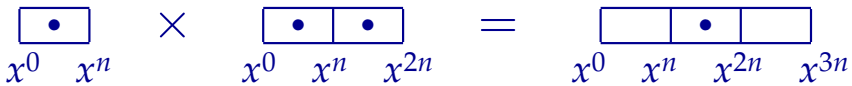
Tellegen's Polynomial Dictionary

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

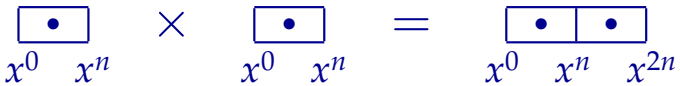
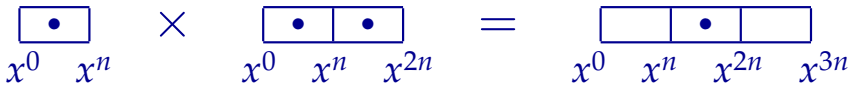
Tellegen's Polynomial Dictionary

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
$P(x) \mapsto P(x+1)$	$P = \sum a_i x^{\underline{i}} \mapsto (P(0), \ldots, P(n-1))$
modular composition	power projection
$\ldots$	$\ldots$

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) $O(N\,\mathsf{M}(N))$

▷ [Paterson and Stockmeyer, 1973] $O(\sqrt{N}(\mathsf{M}(N) + \mathsf{MM}(\sqrt{N})))$

By Shanks' 1969 baby-steps giant-steps technique: split polynomials in chunks of length $\sqrt{N}$, matrices in blocks of size $\sqrt{N} \times \sqrt{N}$.

▷ [Brent and Kung, 1978] $O(\sqrt{N \log N}\,\mathsf{M}(N))$

Similar splitting + Taylor formula.

Composition of series, 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) $O(\mathsf{M}(N))$
- $g \in x\mathbb{K}[x]$ [Brent and Kung, 1978] $O(\mathsf{M}(N)\log N)$
- More generally 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 particular case of power series over a field of large enough characteristic. If using cubic-time matrix multiplication, 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: composition of polynomials, complexity

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

Modular Composition: 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

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 fast bivariate multiplication (e.g. via Kronecker substitution, or evaluation/interpolation)

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 fast bivariate multiplication (e.g. via Kronecker substitution, or evaluation/interpolation)

▷ 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 $n = 2^s < N$, $P \in \mathbb{K}[x]_{<N}$ 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

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

▷ (P_0, Q_0) have bidegree $(n,1)$, then (P_1, Q_1) have bidegree $(n/2,2)$, etc

▷ Total cost: $C \cdot (\mathsf{M}(n,1) + \mathsf{M}(n/2,2) + \cdots + \mathsf{M}(1,n)) = O(\mathsf{M}(n) \log n)$, using e.g. Kronecker's substitution for bivariate polynomial products

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

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

Schönhage's Golden Rules

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