

CDM

Möbius and Fibonacci meet Algebra

Klaus Sutner
Carnegie Mellon University
www.cs.cmu.edu/~sutner

Battleplan

- Inverting Summations: Möbius
- An Algebraic Explanation
- Fibonacci Numbers
- The Fibonacci Monoid

Möbius Inversion

Necklaces

We already know how to deal with counting problems relating to necklaces and bracelets.

Definition 1.

A *k -ary necklace* is a word over a k symbol alphabet up to rotation. A *k -ary bracelet* is a word over a k symbol alphabet up to rotation and reflection.

If you don't like words, think colored beads.

If the words have length n then the appropriate actions are given by the cyclic group \mathbb{Z}_n and the dihedral group D_n , respectively.

No problem.

Primitive Words

But here is a very closely related notion that produces a new type of counting problem.

Definition 2. *A non-empty word w is **primitive** if it is not of the form x^i for any x shorter than w . The **root** of a word w is the shortest word x such that $x^i = w$ and the exponent i is then the **repetition factor** of w .*

Thus a primitive word is its own root and has repetition factor 1.

A word of prime length is primitive unless it is of the form a^n .

Primitive words pop up naturally when one tries to understand commutativity in words.

A Lemma

Lemma 1. *Suppose u and v are non-empty words such that $uv = vu$.*

Then u and v have the same root.

Proof.

Use induction on $|uv|$, the combined length of u and v .

And draw a picture.

□

In other words (NPI), words never commute except in the trivial case.

Counting Primitive Words

Even if n has many factors, random words of length n tend to be primitive.

In order to count primitive words, let $\text{prim}(n, k)$ denote the number of primitive words of length n over an alphabet of size k . Note that

$$k^n = \sum_{d|n} \text{prim}(d, k)$$

Wishful thinking: It would be nice if we could turn this equation around:

We would like to obtain an expression for $\text{prim}(n, k)$ in terms of k^n .

Möbius Function

There is a well-known method to tackle problems of this form based on the *Möbius function* μ which is defined by

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1, \\ (-1)^r & \text{if } n = p_1 \dots p_r, \text{ all distinct primes,} \\ 0 & \text{if } n \text{ is divisible by } p^2, p \text{ a prime.} \end{cases}$$



Herr Möbius' invention is a bit strange. Since $\mu(n)$ depends very much on the prime decomposition of n its behavior is rather erratic.

Möbius Inversion

Lemma 2. *Möbius Inversion Formula*

Let $f(n) = \sum_{d|n} g(d)$. Then

$$g(n) = \sum_{d|n} \mu(d) f(n/d) = \sum_{d|n} \mu(n/d) f(d).$$

In other words, if f is obtained from g by summation (over divisors), then g can in turn be expressed by summation over f .

Why On Earth???

Proposition 1.

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if } n = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Proof.

Suppose $n > 1$ with prime decomposition

$$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$$

The only divisors that matter in the sum are $d \mid p_1 p_2 \cdots p_k$.

But then the $\mu(d)$ cancel out.

□

Calculating

Compute away:

$$\begin{aligned}\sum_{d|n} \mu(d) f(n/d) &= \sum_{d|n} \mu(n/d) f(d) \\ &= \sum_{d|n} \mu(n/d) \sum_{e|d} g(e) \\ &= \sum_{e|d|n} \mu(n/d) g(e) \\ &= \sum_{e|n} \sum_{d|n/e} \mu(n/ed) g(e) \\ &= \sum_{e|n} \sum_{d|n/e} \mu(d) g(e) \\ &= g(n)\end{aligned}$$

Application: Multiplicative Subgroup

Recall Euler's totient function

$$\varphi(n) = \#(k \mid 1 \leq k < n, \gcd(k, n) = 1)$$

Then

$$\sum_{d|n} \varphi(d) = n$$

Hence

$$\varphi(n) = \sum_{d|n} \mu(d)n/d$$

No Free Lunch

Alas, computationally this is not a great step forward:

- To compute φ directly, we need prime factorization.
- To compute φ via μ , we need prime factorization.

This is inevitable in a way: since factorization is presumably hard, and division is cheap, the complexity has to go somewhere else: it hides in μ .

Computationally there is no free lunch, ever.

Application: Primitive Words

For the number of primitive words the Möbius inversion formula yields

$$\text{prim}(n, k) = \sum_{d|n} \mu(d) k^{n/d}$$

Over a binary alphabet the number of primitive words up to length 12 is

2, 2, 6, 12, 30, 54, 126, 240, 504, 990, 2046, 4020

Application: Necklaces

Define $\text{neck}(n, k)$ to be the number of necklaces of length n using beads of k colors. By stringing up primitive words into necklaces we get

$$\begin{aligned}
 \text{neck}(n, k) &= \sum_{d|n} \frac{1}{d} \text{prim}(d, k) \\
 &= \frac{1}{n} \sum_{d|n} \frac{n}{d} \sum_{e|d} \mu(e) k^{d/e} \\
 &= \frac{1}{n} \sum_{d|n} \sum_{e|n/d} \mu(e) d k^{n/de} \\
 &= \frac{1}{n} \sum_{D|n} \sum_{ed=D} \mu(e) D/e k^{n/D} \\
 &= \frac{1}{n} \sum_{D|n} \varphi(D) k^{n/D}
 \end{aligned}$$

Lyndon Words

In each necklace (considered as an equivalence class of words) we can pick the lexicographically first as a representative.

Definition 3. A **Lyndon word** is a representative for a necklace that is also primitive.

Proposition 2. The number of Lyndon words of length n over a k symbol alphabet is

$$\frac{1}{n} \text{prim}(n, k) = \frac{1}{n} \sum_{d|n} \mu(n/d) k^d.$$

Algebraic Proof

Möbius Meets Algebra

What is really going on? Our proof, while correct, offers no insights. A good proof should explain why the result holds, not just establish its formal correctness.

Definition 4. An **arithmetic function** is a function $f : \mathbb{N}^+ \rightarrow \mathbb{C}$ from the positive integers to the complex numbers.

The **Dirichlet product** or **convolution** of two arithmetic functions f and g is defined as

$$(f * g)(n) = \sum_{d|n} f(d)g(n/d) = \sum_{xy=n} f(x)g(y).$$

Thus, an arithmetic function is just an infinite sequence of numbers. Often the numbers are integers or rationals, but we might as well deal with the more general case.

Properties of the Dirichlet Product

Let ε be the arithmetic function defined by

$$\varepsilon(n) = \begin{cases} 1 & \text{if } n = 1, \\ 0 & \text{otherwise.} \end{cases}$$

The following is easy to see.

Proposition 3.

*The Dirichlet product is associative: $f * (g * h) = (f * g) * h$ and commutative: $f * g = g * f$.*

*Moreover, ε is a neutral element: $f * \varepsilon = \varepsilon * f = f$.*

So, we are dealing with a commutative monoid.

Inverse Elements

Which arithmetic functions f have an inverse f^{-1} such that For example $\varepsilon^{-1} = \varepsilon$.

Lemma 3. f has an inverse iff $f(1) \neq 0$.

Proof.

Suppose g is the inverse of f . Then

$$1 = (f * g)(1) = f(1)g(1)$$

so that $f(1) \neq 0$.

For the opposite direction let $f(1) \neq 0$. We can construct the inverse g directly by induction.

$$g(n) = \begin{cases} \frac{1}{f(1)} & \text{if } n = 1, \\ \frac{-1}{f(1)} \sum_{1 < d|n} f(d)g(n/d) & \text{otherwise.} \end{cases}$$

□

Zeta and Möbius Function

Definition 5. *The zeta function is defined by $\zeta(n) = 1$.*

Its inverse is called the Möbius function μ .

This is justified by the lemma; ζ is indeed invertible.

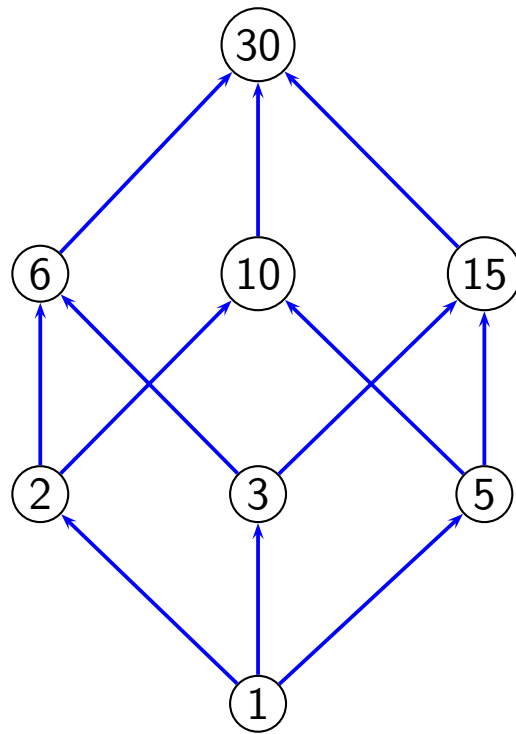
Note that convolution with ζ is essentially just a summation

$$(f * \zeta)(n) = \sum_{d|n} f(d)$$

so with $g = f * \zeta$ we have $f = g * \mu$: exactly what we wanted.

Partial Orders

Since we are not summing over intervals $i = 1, \dots, n$ but over the divisors of n we need to talk about partial orders.



Intervals

For any partial order P define *intervals* by

$$[a, b] = \{x \in P \mid a \leq x \leq b\}$$

and $\text{int}(P)$ the collection of all non-empty intervals over P .

Now consider two functions $f, g : \text{int}(P) \rightarrow \mathbb{R}$. The *convolution* of f and g is

$$(f * g)([a, b]) = \sum_{a \leq x \leq b} f([a, x])g([x, b])$$

Technicalities

For this to work we have to assume that P is *locally finite*: all intervals $[a, b]$ must be finite.

Also note that these intervals are not necessarily linearly ordered, the picture from above is the interval $[1, 30]$.

The notation $f([a, b])$ is technically correct, but really an atrocity.

One usually just writes $f(a, b)$ instead.

Incidence Algebra

Definition 6.

$\langle \text{int}(P), * \rangle$ is the **incidence algebra** (over the partial order P).

The **delta function** is defined to be $\delta(x, x) = 1$, and 0 otherwise.

Proposition 4. *The incidence algebra is a monoid: convolution is an associative operation, and δ is a neutral element.*

Proof is a standard exercise in summation.

Inverse Elements

One might wonder what elements possess an inverse in the incidence algebra:

$$f * g = g * f = \delta$$

and what these inverses might look like.

Lemma 4.

An element f of the incidence algebra is invertible iff $f(x, x) \neq 0$ for all x .

Proof. Necessity is easy.

$$1 = \delta(x, x) = (f * g)(x, x) = f(x, x)g(x, x).$$

Proof, contd.

For sufficiency we will define g explicitly.

$$g(x, y) = \begin{cases} \frac{1}{f(x, x)} & \text{if } x = y, \\ \frac{-1}{f(x, x)} \sum_{x < z \leq y} f(x, z)g(z, y) & \text{otherwise.} \end{cases}$$

This definition uses induction on the size of the intervals. By brute force, $f * g = \delta$.

Note that g also has the property from the lemma, so let h be its right inverse.

$$g * f = (g * f) * (g * h) = g * (f * g) * h = g * h = \delta.$$

So, g is also a left inverse.

□

Zeta Function

Definition 7. *The zeta function is defined by $\zeta(x, y) = 1$.*

From the lemma, ζ is invertible.

Its inverse is called the Möbius function μ of $\text{int}(P)$.

Note that convolution with ζ is essentially just a summation

$$f * \zeta(a, b) = \sum_{a \leq z \leq b} f(a, z)$$

so with $g = f * \zeta$ we have $f = g * \mu$: exactly what we wanted.

The Theorem

Theorem 1. *Let $g(x, y) = \sum_{x \leq z \leq y} f(x, z)$. Then*

$$f(x, y) = \sum_{x \leq z \leq y} \mu(z, y) \cdot g(x, z).$$

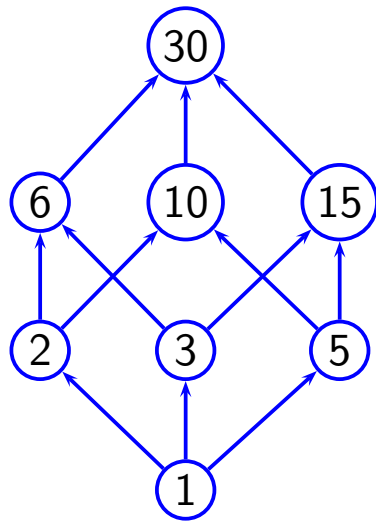
This is very pretty, but the problem is that we need to compute μ .

There are techniques to do this for interesting cases (using products of partial orders), but we won't go there.

For the divisibility order on the natural numbers we get the classical Möbius function – sort of.

Another Technicality

In our applications, the partial order is divisibility on the natural numbers.



However, we are really dealing with functions of the form $f_0 : P \rightarrow \mathbb{R}$: the argument ranges over the partial order, not the incidence algebra.

Easy to fix: set $f(x, y) = f_0(y)$ or $f(x, y) = f_0(x)$.

Fibonacci and Algebra

Recurrence Equations

We can define a sequence $(u_n)_{n \geq 0}$ in two ways:

- Explicitly: $u_n = n(n + 1)/2$.
- Inductively: $u_n = u_{n-1} + n$.

Inductively definitions are often much easier to find, e.g. in running time analysis. But we then want a explicit formula for the sequence, or at least an asymptotically correct explicit formula.

Definition 8. *An equation of the form*

$$u_n = a_1 u_{n-1} + a_2 u_{n-2} + \dots + a_k u_{n-k} + f(n)$$

is a linear recurrence equation of order k .

Solving *this recurrence means to find an explicit formula for u_n .*

Terminology

- linear: there are no terms u_i^2 , $\sqrt{u_i}$, and so on
- order k : u_n depends on $u_{n-1}, u_{n-2}, \dots, u_{n-k}$
- called *homogeneous* if $f(n) = 0$, non-homogeneous otherwise

Solving recurrences turns out to be rather difficult. Requires somewhat complicated machinery, and lot's of tricks.

Sometimes, the best approach is to make an educated guess and then verify. Of course, that requires experience.

Here are some basic ideas.

First-Order

How do we solve

$$u_n = a \cdot u_{n-1} + b$$

Substitute equation into itself, and hope for a pattern.

$$u_n = a^2 u_{n-2} + ab + b$$

$$u_n = a^3 u_{n-3} + a^2 b + ab + b$$

$$u_n = a^4 u_{n-4} + a^3 b + a^2 b + ab + b$$

Leads to conjecture:

$$u_n = a^n u_0 + b \sum_{i < n} a^i$$

Easy to prove by induction. So for $a \neq 1$ we have

$$u_n = a^n u_0 + b \frac{a^n - 1}{a - 1}$$

An Aside

Note that this “repeated substitution until a pattern becomes visible” approach actually works very well in the context of computer algebra: let the system do the substitutions and the necessary simplifications.

You just look for the patterns.

Asymptotics

If we are not interested in details (e.g., in running time analysis) we can simply say

$$u_n = \Theta(a^n) = \Theta(2^{n \log_2 a})$$

But note that for $a = 1$ we get

$$u_n = u_0 + bn$$

so that

$$u_n = \Theta(n).$$

In this case, asymptotic notation is just convenient to hide irrelevant detail, but sometimes it is the only way to get a reasonable answer.

Non-Constant

How about

$$u_n = a \cdot u_{n-1} + f(n)?$$

Easy to see

$$u_n = a^n u_0 + \sum_{i < n} a^i f(n - i)$$

Need to know more about f to proceed.

Example:

$$u_0 = 0$$

$$u_n = u_{n-1} + n$$

Since $a = 1$ we easily get $u_n = n(n + 1)/2$.

Case $a \neq 1$

How about

$$u_0 = 0$$

$$u_n = a \cdot u_{n-1} + n$$

This already makes an astonishing mess. To tackle the problem, first look at the homogeneous case:

$$u_n = a \cdot u_{n-1}$$

That's easy: $u_n = a^n u_0$.

Crazy Idea: Let's try something that looks like the homogeneous solution, plus a term similar to $f(n)$.

$$u_n = c_0 + c_1 n + c_2 a^n$$

This may or may not work, let's just try.

An Ansatz

We need

$$a(c_0 + c_1(n - 1) + c_2a^{n-1}) + n = c_0 + c_1n + c_2a^n$$

Use $u_0 = 0$, and substitute $n = 1$ and $n = 2$:

$$c_0 + c_2 = 0$$

$$c_0(a - 1) - c_1 + 1 = 0$$

$$c_0(a - 1) + c_1(a - 2) + 2 = 0$$

with solutions

$$c_0 = \frac{-a}{(a - 1)^2}$$

$$c_1 = \frac{-1}{a - 1}$$

which produces the final solution

$$u_n = \frac{a(a^n - 1) - n(a - 1)}{(a - 1)^2}.$$

It Works!

For example

$$u_0 = 0$$

$$u_n = 2 \cdot u_{n-1} + n$$

has solution

$$u_n = 2^{n+1} - n - 2 = \Theta(2^n)$$

The Flip-Side

Recall the fundamental principle of recursion:

Complicated problems often have simple recursive solutions.

Now we are dealing with the opposite problem:

Simple recursive problems often have complicated solutions.

There is no free lunch after all . . .

Second-Order: Fibonacci Numbers

The simplest second-order recurrence is the famous Fibonacci recurrence (homogeneous):

$$F_0 = 0,$$

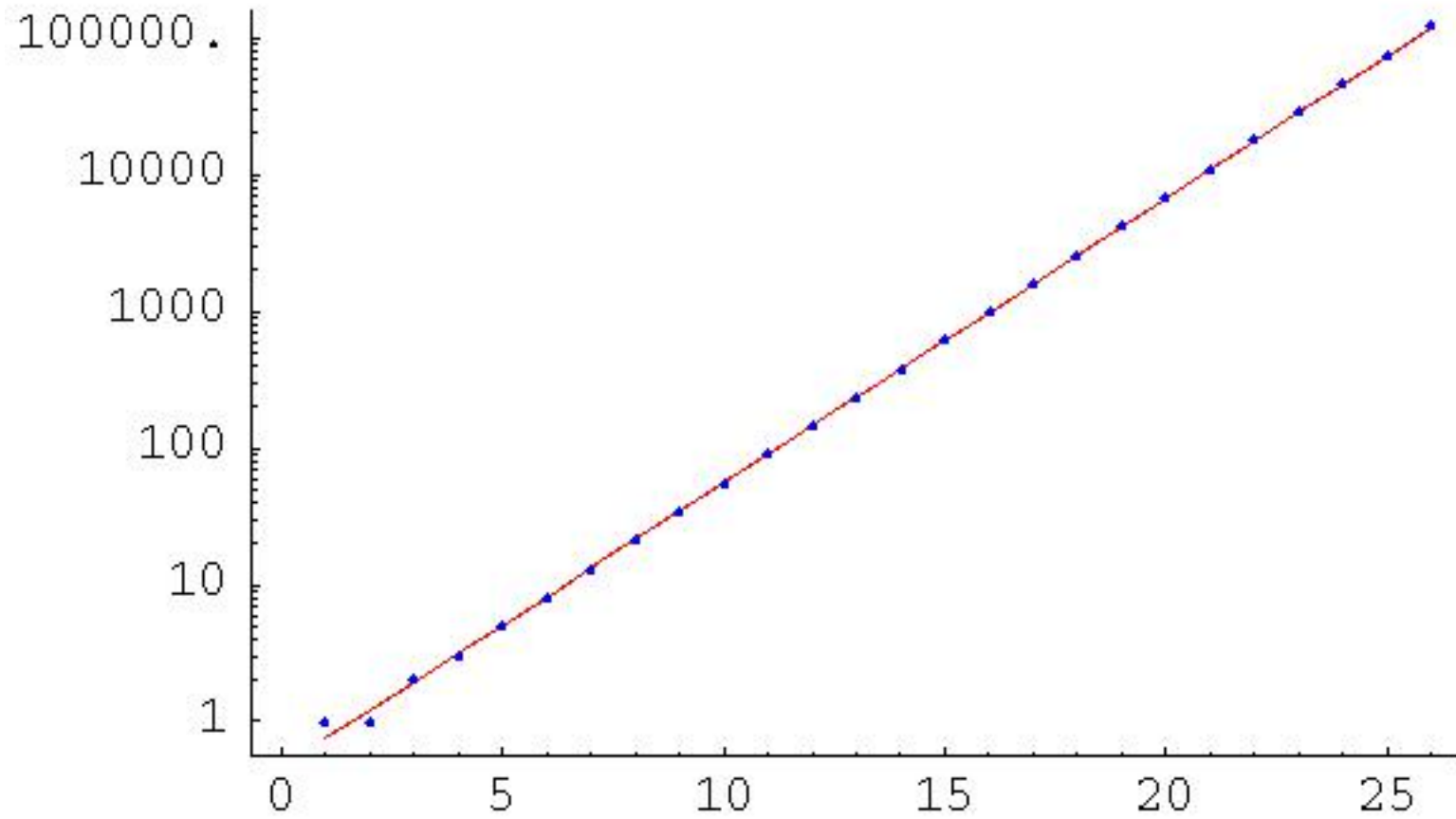
$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

What can we say about these numbers?

Clearly $F_n < 2^n$, but they grow pretty quickly: here's a log-plot.

Fibonacci Plot



Characteristic Equation

The plot suggests that $\log F_n$ is linear. Hence we should have $F_n \approx c \cdot x^n$.

What is c and x ? Need

$$cx^n = cx^{n-1} + cx^{n-2}$$

or

$$x^2 - x - 1 = 0$$

Definition 9. *This is the **characteristic equation** of the recurrence.*

Solutions of the characteristic equation:

$$x_{1/2} = \frac{1 \pm \sqrt{5}}{2}$$

We will construct a solution of the recurrence from these.

The Golden Ratio

We clearly need the root larger than 1, often called the *Golden Ratio*

$$\Phi = \frac{1 + \sqrt{5}}{2} \approx 1.61803$$

The second root is

$$\hat{\Phi} = \frac{1 - \sqrt{5}}{2} = 1 - \Phi.$$

If we set $c = 1/\sqrt{5}$ we get a fairly good approximation $F_n \approx \Phi^n / \sqrt{5}$.

E.g., $F_{20} = 6765$ and

$$\frac{\Phi^{20}}{\sqrt{5}} - F_{20} \approx 0.0000295639$$

So it looks like $F_n = \Theta(\Phi^n)$.

Precise Solution

► But can we get a precise expression for F_n ?

How about using both roots of the equation? We could try something like

$$F_n = c_1 \Phi^n + c_2 \hat{\Phi}^n$$

Yields linear equations

$$c_1 + c_2 = 0$$

$$c_1 \Phi + c_2 \hat{\Phi} = 1$$

with solution

$$c_1 = -c_2 = 1/\sqrt{5}.$$

► Again, this is the “guess-and-verify” approach.

DeMoivre-Binet

Claim. *DeMoivre-Binet Formula*

$$F_n = (\Phi^n - \hat{\Phi}^n) / \sqrt{5}$$

Proof. Write $G_n = (\Phi^n - \hat{\Phi}^n) / \sqrt{5}$. We know $G_0 = 0 = F_0$ and $G_1 = 1 = F_1$. But it is straightforward to check that

$$G_n = G_{n-1} + G_{n-2}.$$

But then by induction $F_n = G_n$. □

So

$$F_n = ((1 + \sqrt{5})^n - (1 - \sqrt{5})^n) / (2^n \sqrt{5}).$$

Note that this expression really yields an integer: the square roots of 5 all cancel out.

Computing with $\sqrt{5}$

Since $\hat{\Phi} \approx -0.618034$ we have $\lim \hat{\Phi}^n = 0$ quite rapidly, which explains why the second term is not so important.

Really a bit strange: we are computing in $\mathbb{Q}(\sqrt{5})$, i.e., with numbers of the form

$$x + y \cdot \sqrt{5} \quad \text{where } x, y \in \mathbb{Q}$$

in order to compute an integer.

Note that $\mathbb{Q}(\sqrt{5})$ is closed under addition and multiplication, so we can really do arithmetic. For example

$$(x + y \cdot \sqrt{5})^{-1} = (x - y\sqrt{5}) / (x^2 - 5y^2)$$

Application: Bit-Counts

We have

$$F_n \approx 0.45 \cdot 1.62^n \approx 2^{0.7n-1}$$

Hence the binary expansion of F_n should have about $0.7n$ digits. Quite close:

F_{1000} : 694 binary digits, F_{10000} : 6942 binary digits.

So we can easily estimate the amount of memory needed to store Fibonacci numbers.

By a simple error estimate we get:

$$F_n = \text{round}(\Phi^n / \sqrt{5})$$

Alternates (above/below actual integer value):

0.447, 0.724, 1.171, 1.894, 3.065, 4.960, 8.025, 12.985

Fibonacci Sums

Can we find a nice description of $\sum_{i \leq n} F_i$?

Claim.

$$\sum_{i \leq n} F_i = F_{n+2} - 1$$

Proof.

By induction on n . Let $S_n = \sum_{i \leq n} F_i$.

Base: $S_0 = 0 = F_2 - 1$.

Step:

$$S_{n+1} = S_n + F_{n+1} = F_{n+2} - 1 + F_{n+1} = F_{n+3} - 1.$$

Done. □

Exercise: Figure out $\sum_{i \leq n} F_{2i}$.

More Identities

Claim. *Sums of Squares*

$$\sum_{i \leq n} F_i^2 = F_n \cdot F_{n+1}.$$

Claim. *Cassini's Identity*

$$F_n^2 - F_{n-1} \cdot F_{n+1} = (-1)^n$$

Both identities are easy to prove by induction.

The point is finding them in the first place (relatively easy with a computer algebra system).

Fast Fibonacci Numbers

We can certainly compute F_n in $O(n)$ steps (assuming arithmetic is $O(1)$ which is a bit fishy).

Is there a better way?

The $\mathbb{Q}(\sqrt{5})$ computation is no better (and in fact worse).

Can we perhaps speed up the recursion? Yes!

Claim.

$$F_{2n} = F_n^2 + 2 \cdot F_n \cdot F_{n-1}$$
$$F_{2n+1} = F_{n+1}^2 + F_n^2$$

This can be proved brute-force by induction, but where on earth do these equations come from?

Extending the Recurrence

The key is to extend the basic recurrence:

$$F_{n+1} = 1 \cdot F_n + 1 \cdot F_{n-1}$$

$$F_{n+2} = 2 \cdot F_n + 1 \cdot F_{n-1}$$

$$F_{n+3} = 3 \cdot F_n + 2 \cdot F_{n-1}$$

$$F_{n+4} = 5 \cdot F_n + 3 \cdot F_{n-1}$$

$$F_{n+5} = 8 \cdot F_n + 5 \cdot F_{n-1}$$

$$F_{n+5} = 13 \cdot F_n + 8 \cdot F_{n-1}$$

Now it's easy to conjecture

Lemma 5.

$$F_{n+m} = F_{m+1}F_n + F_mF_{n-1}$$

Proof

Proof.

By induction on m .

Base $m = 0$ is trivial.

Step:

$$\begin{aligned}F_{n+m+1} &= F_{n+m} + F_{n+m-1} \\ &= F_{m+1}F_n + F_mF_{n-1} + F_mF_n + F_{m-1}F_{n-1} \\ &= F_{m+2}F_n + F_{m+1}F_{n-1}\end{aligned}$$

Done. □

Our claim follows from the lemma: set $m = n$ and $m = n, n = n + 1$, respectively.

GCDs of Fibonacci Numbers

But there is more.

Claim. F_n and F_{n-1} are coprime.

Proof. By tracing the Euclidean algorithm. □

Lemma 6. $\gcd(F_m, F_n) = F_{\gcd(n,m)}$.

Proof.

Let $d = \gcd(F_{n+m}, F_n)$. The lemma implies

$$0 = F_m \cdot F_{n-1} \pmod{d},$$

so d divides F_m .

Hence $\gcd(F_{m+n}, F_n) = \gcd(F_n, F_m)$.

□

A Wild Monoid

Our fast recurrence is one way to compute Fibonacci numbers quickly. Are there any other clever ways?

Recall that in any monoid we can compute x^n in $O(\log n)$ monoid multiplications (using the squaring trick).

Can we find a monoid and a special element a such that $a^n = F_n$?

Definition 10. *The Fibonacci Monoid*

Define the **Fibonacci product** $*$ on pairs of natural numbers by

$$(x, y) * (x', y') = (xx' + xy' + x'y, xx' + yy')$$

Bizarre, but why not? It is straightforward to check:

Claim. $\mathbb{N} \times \mathbb{N}$ with Fibonacci product and $(0, 1)$ forms a commutative monoid.

So???

k	$(1, 0)^{2^k}$	F_{2^k}
0	(1, 0)	1
1	(1, 1)	1
2	(3, 2)	3
3	(21, 13)	21
4	(987, 610)	987
5	(2178309, 1346269)	2178309

From the table, it looks like $a = (1, 0)$ works. Easy to show by induction

Claim. *In the Fibonacci monoid,*

$$(1, 0)^n = (F_n, F_{n-1}).$$

So we can compute F_{1024} and F_{1023} very quickly: 9 Fibonacci multiplications suffice.

Computing F_{1022}

To get F_{1022} we can still use our fast exponentiation algorithm.

Since $1022 = 1111111110_2$ we need $9 + 8 = 17$ operations.

But could we do a

$$F_{1022} = (1, 0)^{2^{10}} * (1, 0)^{-2}$$

in just $9 + 1 = 10$ operations?

Well, for this we need a **group**, not just a monoid.

Is the Fibonacci monoid a group?

Given x and y we have to solve

$$(xx' + xy' + x'y, xx' + yy') = (0, 1)$$

for x' and y' .

No Problem

$$x' = \frac{x}{x^2 - xy - y^2}$$

$$y' = \frac{-x - y}{x^2 - xy - y^2}$$

. . . except that we are now dealing with rationals!

Also, $(0, 0)$ has no inverse.

At any rate,

$$(1, 0)^{-1} = (1, -1)$$

so for the one element we are most interested in, there is no problem.

Exercise 1. *Show that all $(x, y) \neq (0, 0)$ have an inverse.*

Fast Exponentiation

Let's be a bit more careful about the fast exponentiation algorithm to compute a^n :

```
z = 1;
while ( n > 0 )
{
    if (n is odd) z = z * a;    // monoid operation
    a = a * a;
    n = n/2;
}
return z;
```

Proposition 5. *This uses $ds(n) + dc(n)$ monoid operations where $ds(n)$ is the **digit sum** of n (number of 1's in binary expansion), and $dc(n)$ is the total number of digits in the binary expansion.*

Not Always Optimal

Alas, that's not always optimal! E.g., we can get a^8 faster by 2 ops.

Worse yet, in a group we can sometimes shave off more:

$$\begin{aligned} 262128 &= 1111111111111110000_2 \\ &= 2^{18} - 2^4 \end{aligned}$$

Can be handled in 20 group operations (multiplication and division).

Challenge: Come up with a good algorithm that computes F_n in the least number of group operations.

Straight Line Arithmetic

We need to find an optimal *straight line program* (SLP), a sequence of instructions

$$v_k = v_i * v_j$$

$$v_k = v_i / v_j = v_i * v_j^{-1}$$

where k is incremented in each instruction, and $i, j < k$. We always start with

$$v_0 = (1, 0)$$

The result is the left value of the variable v_k in the last instruction. We call that k the *length* of the SLP.

Optimal then simply means: minimal length.

So, we want an algorithm that, on input n , determines the optimal SLP with output F_n .

This ain't easy even for just a few values of n !

Examples

Here is a SLP for F_{31} using only multiplication (no division).

$$\begin{array}{ll} v_0 = (1, 0) & F_1 \\ v_1 = v_0 * v_0 & F_2 \\ v_2 = v_1 * v_1 & F_4 \\ v_3 = v_2 * v_2 & F_8 \\ v_4 = v_3 * v_3 & F_{16} \\ v_5 = v_4 * v_3 & F_{24} \\ v_6 = v_5 * v_2 & F_{28} \\ v_7 = v_6 * v_1 & F_{30} \\ v_8 = v_7 * v_0 & F_{31} \end{array}$$

So length 8 suffices.

Going Back

A length 6 SLP for F_{31} that uses division.

$$\begin{array}{ll} v_0 = (1, 0) & F_1 \\ v_1 = v_0 * v_0 & F_2 \\ v_2 = v_1 * v_1 & F_4 \\ v_3 = v_2 * v_2 & F_8 \\ v_4 = v_3 * v_3 & F_{16} \\ v_5 = v_4 * v_4 & F_{32} \\ v_6 = v_4 / v_0 & F_{31} \end{array}$$

Is length 6 perhaps optimal?

Optimality

Yes, but that's not so easy to show. Nor is the solution unique.

$$\begin{array}{ll} v_0 = (1, 0) & F_1 \\ v_1 = v_0 * v_0 & F_2 \\ v_2 = v_1 * v_1 & F_4 \\ v_3 = v_2 * v_2 & F_8 \\ v_4 = v_3 * v_3 & F_{16} \\ v_5 = v_4 / v_0 & F_{15} \\ v_6 = v_5 * v_4 & F_{31} \end{array}$$

Exercise 2. *Show that length 6 is optimal for F_{31} : there is no shorter SLP that computes F_{31} .*

A Little Help

Looking at these examples, notice that we only worry about the n in F_n . We could have written

$$\begin{array}{ll} v_0 = 1 & 1 \\ v_1 = v_0 + v_0 & 2 \\ v_2 = v_1 + v_1 & 4 \\ v_3 = v_2 + v_2 & 8 \\ v_4 = v_3 + v_3 & 16 \\ v_5 = v_4 - v_0 & 15 \\ v_6 = v_5 + v_4 & 31 \end{array}$$

The same simplification works for all our SLPs.

What's going on?

First note that in order to execute an SLP we need two things:

- an arbitrary group G , and
- a special element $a \in G$.

We can then initialize $v_0 = a$ in line 0, and run through the other instructions interpreting multiplication and division in the group G .

The same program can be run over any group G , and with any starting value $a \in G$.

E.g., if we let $a = 1 \in G$, the output is always $1 \in G$.

Why? The whole computation takes place in

$$\langle a \rangle = \{ a^i \mid i \in \mathbb{Z} \} \subseteq G,$$

the subgroup generated by a .

SLP in a Group

So, given an SLP P , a group G , and $a \in G$, we can define

$$P(G, a) \in G$$

as the result of executing P over G with initial value a .

Now consider the two groups

$$\mathbb{Z} = \langle \mathbb{Z}, +, 0 \rangle$$

$$\mathbb{F} = \langle F, *, (0, 1) \rangle$$

where \mathbb{F} is the subgroup of the full Fibonacci group generated by $(1, 0)$.

These two groups are isomorphic via $f : \mathbb{Z} \rightarrow \mathbb{F}$:

$$f(n) = (1, 0)^n = (F_n, F_{n-1})$$

So, instead of computing $P(\mathbb{F}, (1, 0))$ we can just as well compute $P(\mathbb{Z}, 1)$.

Representation is Crucial

This is another example where an isomorphism makes life so much easier.

Everybody understands $\langle \mathbb{Z}, +, 0 \rangle$ very well intuitively.

But $\langle F, *, (0, 1) \rangle$ is a bit mysterious.

Doesn't matter, they are isomorphic, so we can argue in either one.

Summary

- Möbius inversion is another classical counting tool.
- The “best” proof uses algebra: it avoids magic, and the argument generalizes very nicely.