

# CDM

## Recurrences and Fibonacci

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

### Battleplan

- Recurrence Equations
- First-Order
- Second-Order: Fibonacci Numbers
- DeMoivre-Binet Formula
- Some Fibonacci Identities
- Fast Fibonacci
- Real Fast Fibonacci

## Recurrence Equations

### Recurrence Equations

We can define a sequence  $(u_n)_{n \geq 0}$  in two standard 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 1.** 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 the equation

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

The natural attack is to substitute the equation into itself a number of times and hope for a pattern.

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

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

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

In this case there is an easy conjecture, which can be proved by induction:

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

## Solution and Asymptotics

So for  $a \neq 1$  we have

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

but for  $a = 1$  we get

$$u_n = u_0 + bn$$

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

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

in the case  $a \neq 1$  and  $u_n = \Theta(n)$  otherwise.

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

## Non-Constant

How about

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

It is easy to see that

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

We need to know more about  $f$  to proceed.

**Example 1.**

$$u_0 = 0$$

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

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

## Case $a \neq 1$

How about the slightly more complicated

$$u_0 = 0$$

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

This already makes an astonishing mess. To tackle the problem, first use a standard trick in mathematics: chop off offending parts and consider 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  $f(n)$ , something like

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

This may or may not work, we are just guessing here.

## Onward

We need

$$a(c_0 + c_1(n - 1) + c_2 a^{n-1}) + n = c_0 + c_1 n + c_2 a^n$$

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

$$c_0 = \frac{-a}{(a - 1)^2} \quad c_1 = \frac{-1}{a - 1}$$

These coefficients produces the final solution

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

## Example

For example

$$u_0 = 0$$

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

has solution

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

Not much different from our Hanoi solution:

$$u_0 = 0$$

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

with solution

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

## The Flip-Side

Recall from last time:

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

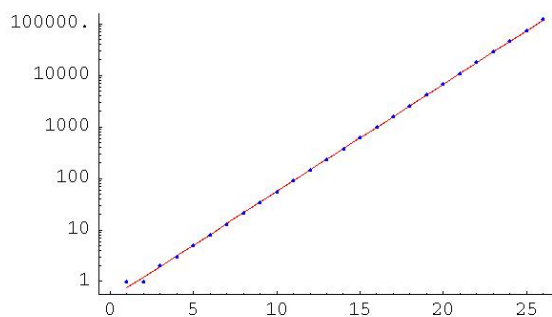
The simplest second-order recurrence is the famous Fibonacci recurrence (homogeneous):

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

What can we say about these numbers?

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

### Fibonacci Plot



A log-linear plot of the first 26 Fibonacci numbers; note the exponential scale.

### Characteristic Equation

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

What are the constants  $c$  and  $x$ ? We need

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

or

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

**Definition 2.** 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

**Lemma 1.** *DeMoivre-Binet Formula*

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

*Proof.* Write  $G_n = (\Phi^n - \widehat{\Phi}^n) / \sqrt{5}$ . 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  $F_n = G_n$ .  $\square$

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

This is really a bit strange: we are computing in the field extension  $\mathbb{Q}(\sqrt{5})$ , 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.  $\square$

**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 Mathematica).

### 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.**

$$\begin{aligned} F_{2n} &= F_n^2 + 2 \cdot F_n \cdot F_{n-1} \\ F_{2n+1} &= F_{n+1}^2 + F_n^2 \end{aligned}$$

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 2.**

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

### 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 3.**  $\gcd(F_m, F_n) = F_{\gcd(n,m)}$ .

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

### The Fibonacci Monoid

### 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 3.** *The Fibonacci Monoid*

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

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

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.

► Do you see why all  $(x, y) \neq (0, 0)$  do have an inverse?

At any rate,

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

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

## Fast Exponentiation

In a monoid, we can use our fast exponentiation algorithm to compute  $a^n$ :

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

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:

$$262128 = 111111111111110000_2$$

$$= 2^{18} - 2^4$$

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).

$$v_0 = (1, 0) \quad F_1$$

$$v_1 = v_0 * v_0 \quad F_2$$

$$v_2 = v_1 * v_1 \quad F_4$$

$$v_3 = v_2 * v_2 \quad F_8$$

$$v_4 = v_3 * v_3 \quad F_{16}$$

$$v_5 = v_4 * v_3 \quad F_{24}$$

$$v_6 = v_5 * v_2 \quad F_{28}$$

$$v_7 = v_6 * v_1 \quad F_{30}$$

$$v_8 = v_7 * v_0 \quad F_{31}$$

So length 8 suffices.

$F_{31}$ 

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?

 $F_{31}$ 

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 1.** 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  $a$ .

So, given an SPL  $P$ , 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

$$\begin{array}{l} \mathbb{Z} = \langle \mathbb{Z}, +, 0 \rangle \\ \mathbb{F} = \langle F, *, (0, 1) \rangle \end{array}$$

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.