CDM

Recurrences and Fibonacci

Klaus Sutner

Carnegie Mellon University

20-fibonacci 2017/12/15 23:16

1. Recurrence Equations

- Second Order

- The Fibonacci Monoid

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

An equation of the form

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

is a linear recurrence equation of order k.

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

- linear: there are no terms u_i^2, $\sqrt{u_i}$, and so on
- order k: u_n depends on $u_{n-1}, u_{n-2}, \ldots, u_{n-k}$
- 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.

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:

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

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.

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

$$u_0 = 0$$
$$u_n = u_{n-1} + n$$

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

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.

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 $n = 2$:

$$\begin{aligned} c_0 + c_2 &= 0 \\ c_0(a-1) - c_1 + 1 &= 0 \\ c_0(a-1) + c_1(a-2) + 2 &= 0 \end{aligned}$$

with solutions

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

These coefficients produces the final solution

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

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

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

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 . . .

- Recurrence Equations

2. Second Order

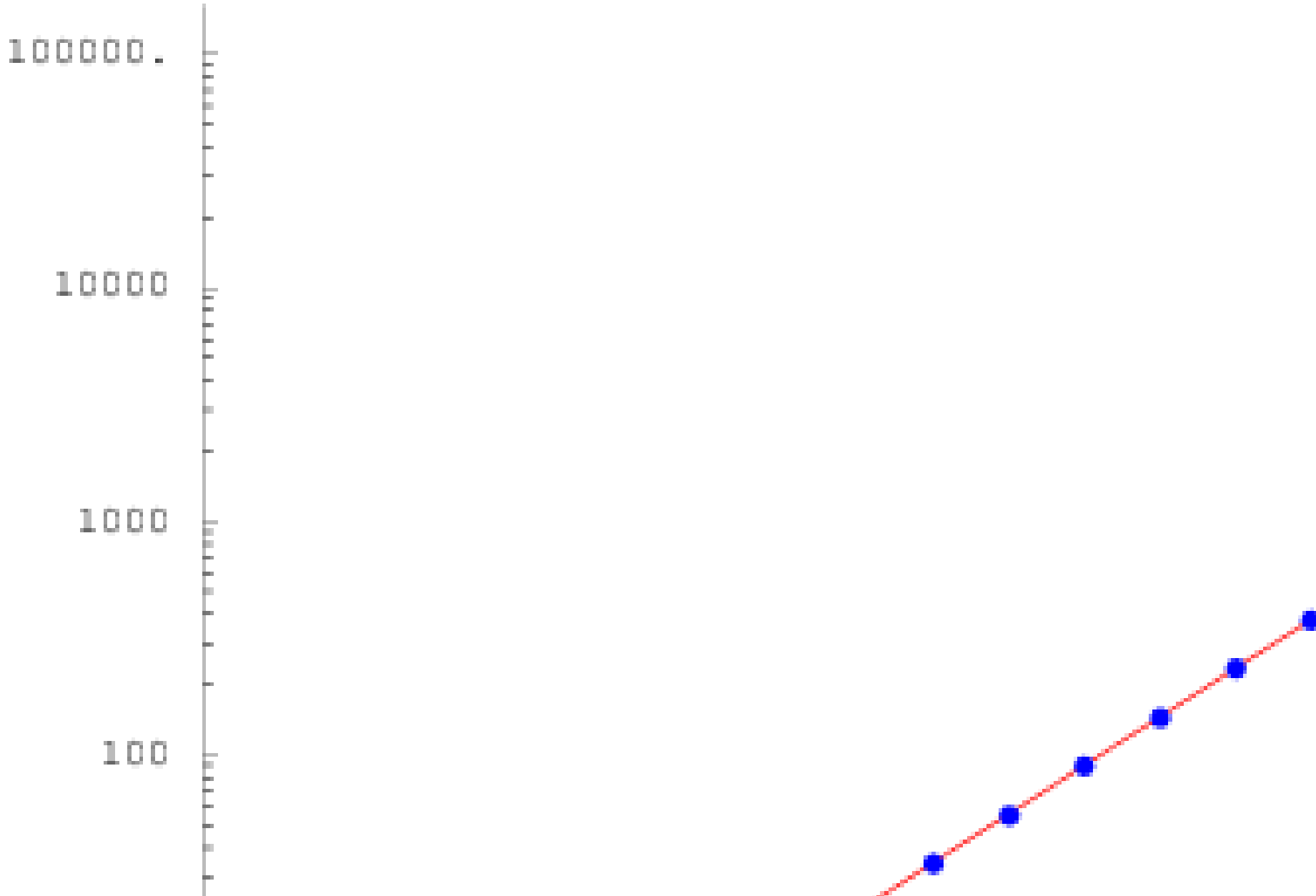
- The Fibonacci Monoid

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.


100000.
10000
1000
100

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

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.

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

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

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 $F_n = \Theta(\Phi^n)$.

But can we get a precise expression for F_n?

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

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

Yields linear equations

$$c_1 + c_2 = 0$$
$$c_1\Phi + c_2\widehat{\Phi} = 1$$

with solution

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

Again, this is the "guess-and-verify" approach.

Lemma

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 $F_n = G_n$. □

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.

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

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

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):

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

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 $\sum_{i \leq n} F_{2i}$.

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).

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?

The key is to extend the basic recurrence:

$$
\begin{aligned}
F_{n+1} &= 1 \cdot F_n + 1 \cdot F_{n-1} \\
F_{n+2} &= 2 \cdot F_n + 1 \cdot F_{n-1} \\
F_{n+3} &= 3 \cdot F_n + 2 \cdot F_{n-1} \\
F_{n+4} &= 5 \cdot F_n + 3 \cdot F_{n-1} \\
F_{n+5} &= 8 \cdot F_n + 5 \cdot F_{n-1} \\
F_{n+5} &= 13 \cdot F_n + 8 \cdot F_{n-1}
\end{aligned}
$$

Now it's easy to conjecture

Lemma

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

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.

But there is more.

Claim

F_n and F_{n-1} are coprime.

Proof. By tracing the Euclidean algorithm. □

Lemma

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

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

□

- Recurrence Equations
- Second Order

3. The Fibonacci Monoid

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

Definition (Fibonacci Monoid)

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

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

It is straightforward to check:

Claim

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

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

```
z = 1;                          // neutral element
while( n > 0 )
{
  if( n odd ) z = z * a;        // monoid multiplication
  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.

Here are some powers of $(0,1)$ in the Fibonacci monoid:

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

So we can compute F_{32} in just 5 monoid operations.

Of course, these operations involve several multiplications and additions of naturals.

The key fact is that the submonoid generated by $(0, 1)$ produces the Fibonacci numbers:

$$(x, y) * (0, 1) = (y, x + y)$$

A straightforward induction shows that

Claim

In the Fibonacci monoid, $(0, 1)^n = (F_{n-1}, F_n)$.

So we can compute F_{1024} and F_{1023} very quickly: 11 Fibonacci multiplications suffice.

But how about F_{1022}?

If we can use the standard algorithm we need 19 operations.

Wild Idea: Perhaps we can work backwards from F_{1024} as in

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

using just 13 operations?

Well, for this we need a group, not just a monoid. Any chance the Fibonacci monoid might be a group?

Given x and y we have to solve

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

for x' and y'. No problem:

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

. . . except that we are now dealing with rationals rather than naturals. Also note that $(0, 0)$ has no inverse, but that is the only problem: $x^2 + xy - y^2$ has no other integral solutions.

At any rate,

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

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

So we have a group on $\mathbb{Q} \times \mathbb{Q} - \{(0,0)\}$, but we are really interested in the subgroup generated by $(0,1)$ which does not involve rationals.

The question arises how many group operations are needed to compute F_n.

We can describe a computation like the one above for F_{1022} as a 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 the line number and $i, j < k$. We always start with

$$v_0 = (0, 1)$$

The result is the 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 turns out to be difficult even for fairly small values of n.

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

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

So length 8 suffices.

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

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

Is length 6 perhaps optimal?

Yes, but that's not so easy to show.

Nor is the solution unique.

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

Exercise

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

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

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

The same simplification works for all our SLPs.

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 $a \in G$.

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

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

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

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

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

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

So, in a sense, instead of computing $P(\mathcal{F}, (0, 1))$ we can just as well compute $P(\mathbb{Z}, 1)$:

$$f(P(\mathbb{Z}, 1)) = P(\mathcal{F}, (0, 1))$$

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

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

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

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

Note that there is a natural addition on $\mathbb{Q} \times \mathbb{Q}$: add vectors component-wise.

It is easy to check that Fibonacci multiplication coexists peaceful with this addition. For example, we have distributivity:

$$x * (y + z) = x * y + x * z.$$

Hence, we really have a commutative ring.

But we have already seen that other than zero every element has an inverse, so we are actually dealing with a field

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

the *Fibonacci field*.

Fields are a key concept in algebra, so one wonders if there is any simple description of this field. The DeMoivre-Binet formula suggest that $\sqrt{5}$ should play a role.

$(1,0)$ is the multiplicative neutral element, so $(n,0)$ corresponds to $n \in \mathbb{N}$.

The inverse $(1/n,0)$ corresponds to $1/n$ and $(m/n,0)$ to the rational m/n.

So $\mathbb{Q}$ embeds into $\mathbb{F}$ via $r \mapsto (r,0)$.

But what is our generator $a = (0,1)$?

$$1/a = (-1,1) = a - (1,0) = a - 1_{\mathbb{F}}$$

Hence a solves the equation $x^2 - x - 1 = 0$ in $\mathbb{F}$, corresponding to $1/2(1+\sqrt{5})$.

But then $(-1,2)$ corresponds to $\sqrt{5}$. If you are suspicious, note that $(-1,2)^2 = (5,0)$.

Proposition

The following map is a field isomorphism:

$$\begin{array}{rccc} f: & \mathbb{Q}[\sqrt{5}] & \longrightarrow & \mathbb{F} \\ & a+b\sqrt{5} & \longmapsto & (a-b, 2b) \end{array}$$

Exercise

Prove the proposition.

Base $B = 2$, digit set $D = \{-1, 0, 1\}$.

Claim

For every natural number n there are digits $d_i \in D$ such that $n = \sum d_i 2^i$ and $d_i d_{i+1} = 0$.

```
i = 0;
while( n > 0 )
{
      if( n odd )
          d[i] = 2 - (n mod 4);
      else
          d[i] = 0;
      n = (n - d[i])/2;
      i++;
}
```