A Fast Recurrence for the Fibonacci Sequence

M. J. Saltzman

September 14, 2018

[This formula was communicated to me by Dr. Neil Calkin.]
The classical recursive definition of the Fibonacci sequence is given by

$$\begin{aligned} F_0 &= 0; \\ F_1 &= 1; \\ F_n &= F_{n-1} + F_{n-2} \qquad\qquad \forall n > 1. \end{aligned}$$

A different recurrence is given by

$$\begin{aligned} F_0 &= 0; \\ F_1 = F_2 &= 1; \\ F_n &= \begin{cases} F_{(n+1)/2}^2 + F_{(n-1)/2}^2 & \text{if } n \text{ is odd} \\ F_{n/2+1}^2 - F_{n/2-1}^2 & \text{if } n \text{ is even} \end{cases} \qquad \forall n > 2. \end{aligned}$$

This recursion has depth proportional to $\log_2 n$ rather than n.

When I first experimented with this method in 2000 using Maple, the iterative computation of F_{100000} took approximately 45 seconds on a 650MHz Intel Pentium III processor. The recursive computation above took about 15 seconds. Maple offers a feature `options remember` that caches intermediate results. Using `options remember`, F_{100000} was computed almost instantaneously.

To see why this works, we can first derive that the nth Fibonacci term (suitably reindexed) counts the number of binary strings of length n with no repeating ones. To see this, we first observe that there is one such string of length zero (the empty string `''` has no consecutive ones) and two such strings of length one (`'0'` and `'1'`). Now suppose that the result holds for strings of length k or less. We can extend a string of length k by adding a bit on the left end. How many strings of length $k+1$ with no consecutive ones are there?

If the new leftmost bit is `0`, than the rest of the string can be any of the strings of length k. If the new leftmost bit is `1`, then the following bit must be `0` and the remaining bits can be any of the strings of length $k-2$. Thus the total number of strings of length $k+1$ is

$$F_{k+1} = F_k + F_{k-1},$$

the Fibonacci sequence.

Now let's consider a different way of computing the count of such strings. Consider odd-length strings of length $2k+1$. We can count the number of strings without consecutive ones as follows: Look at the middle bit. If this bit is `0`, then we can construct acceptable strings using any acceptable string of length k on the left and any acceptable string of length k on the right. If the middle bit is `1` then the bits on either side must be `0`, and the remaining bits can be taken from any pair of acceptable strings of length $k-1$. So the total number of such strings is

$$F_{2k+1} = F_k^2 + F_{k-1}^2.$$

For even-length strings, we take one of the two middle bits and follow the same logic to get

$$F_{2k} = F_k F_{k-1} + F_{k-1} F_{k-2}.$$

Observing that $F_{k-1} = F_k - F_{k-2}$, we see that

$$\begin{aligned} F_{2k} &= F_k[F_k - F_{k-2}] + [F_k - F_{k-2}]F_{k-2} \\ &= F_k^2 - F_{k-2}^2. \end{aligned}$$

Renaming the indices appropriately gives the formula above.

This idea illustrates that recursive computation can be much more efficient than iterative computation. The trick is to use the recursive viewpoint to find ways to divide the work of computing a particular result into subtasks that are different from those that come to mind when one thinks about iterating. Other results in the same vein include Karatsuba's method for multiplying polynomials and Strassen's method for multiplying matrices.