Lecture 26: Recursion

Just as the structure of the natural numbers supports induction as a method of proof, it supports induction as a method of definition or of computation.

When used in this way, induction is usually called recursion, and one speaks of a *recursive definition* or a *recursive algorithm*.

Recursive Definitions

Many well known functions $f(n)$ are most easily defined in the "base step, induction step" format, because $f(n+1)$ depends in some simple way on $f(n)$.

The induction step in the definition is more commonly called the *recurrence relation* for f, and the base step the *initial value*.

Example. The factorial $f(n) = n!$
Initial value. $0! = 1$.
Recurrence relation. $(k+1)! = (k+1) \times k!$.

Many programming languages allow this style of definition, and the value of the function is then computed by a descent to the initial value. For example, to compute 4!, the machine successively computes

$$\begin{aligned} 4! &= 4 \times 3! \\ &= 4 \times (3 \times 2!) \\ &= 4 \times (3 \times (2 \times (1!))) \\ &= 4 \times (3 \times (2 \times (1 \times 0!))) \end{aligned}$$

which can finally be evaluated since $0! = 1$.

The numbers $4, 3, 2, 1$ have to be stored on a "stack" before the program reaches the initial value $0! = 1$ which finally enables it to evaluate 4!.
Thus a recursive program, though short, may run slowly and even cause "stack overflow."

Example. The Fibonacci sequence $0, 1, 1, 2, 3, 5, 8, \ldots$.
The n^{th} number $F(n)$ in this sequence is defined by
Initial values. $F(0) = 0, F(1) = 1$.
Recurrence relation. $F(k+1) = F(k) + F(k-1)$.

Using a recursive program to compute Fibonacci numbers can easily lead to stack overflow, because each value depends on two previous values (each of which depends on another two, and so on).

A more efficient way to use the recursive definition is to use three variables to store $F(k+1)$, $F(k)$ and $F(k-1)$. The new values of these variables, as k increases by 1, depend only on the three stored values, not on all the previous values.

Properties of recursively defined functions

These are naturally proved by induction, using a base step and induction step which parallel those in the definition of the function.

Example. For $n \geqslant 5$, 10 divides $n!$

Proof. *Base step.*

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 10 \times 4 \times 3,$$

hence 10 divides 5!.

Induction step. We have to show

$$10 \text{ divides } k! \implies 10 \text{ divides } (k+1)!$$

Since $(k+1)! = (k+1) \times k!$ by the recurrence relation for factorial, the induction step is clear, and hence the induction is complete.

Example. $F(0) + F(1) + \cdots + F(n) = F(n+2) - 1$

Proof. *Base step.* $F(0) = 0 = F(2) - 1$ because $F(2) = 1$

Induction step. We have to show

$$\begin{aligned} & F(0) + F(1) + \cdots + F(k) = F(k+2) - 1 \\ \Rightarrow \quad & F(0) + F(1) + \cdots + F(k+1) = F(k+3) - 1 \end{aligned}$$

Well,

$$\begin{aligned} & F(0) + F(1) + \cdots + F(k) = F(k+2) - 1 \\ \Rightarrow \quad & F(0) + F(1) + \cdots + F(k+1) \\ & = F(k+2) + F(k+1) - 1, \\ & \text{by adding } F(k+1) \text{ to both sides} \\ \Rightarrow \quad & F(0) + F(1) + \cdots + F(k+1) = F(k+3) - 1 \\ & \text{since } F(k+2) + F(k+1) = F(k+3) \\ & \text{by the Fibonacci recurrence relation} \end{aligned}$$

This completes the induction.

Questions

26.1 A function $s(n)$ is defined recursively by
Initial value: s(0)=0
Recurrence relation: s(n+1)=s(n)+2n+1
Write down the first few values of $s(n)$, and guess what function s is.

$s(1) = s(0) + 2 \times 0 + 1 = 1$. $s(2) = s(1) + 2 \times 1 + 1 = 4$.
$s(3) = s(2) + 2 \times 2 + 1 = 9$. $s(4) = s(3) + 2 \times 3 + 1 = 16$.
Guess that maybe $s(n) = n^2$?

26.2 Check that the function s you guessed in Question 26.1 satisfies $s(0) = 0$ and $s(n + 1) = s(n) + 2n + 1$

If $s(n) = n^2$ then $s(0) = 0^2 = 0$ and
$s(n) + 2n + 1 = n^2 + 2n + 1 = (n + 1)^2 = s(n + 1)$.

(This proves by induction that our guess was correct.)

Problems with recursive solutions

Sometimes a problem about n reduces to a problem about $n-1$ (or smaller numbers), in which case the solution may be a known recursively defined function.

Example. Find how many n-digit binary numerals contain no two consecutive 0s.
We can divide this problem into two cases.

1. Numerals which end in 1, e.g. 1101101. In this case, the numeral before the 1 (110110 here) can be any $(n-1)$-digit numeral with no consecutive 0s.

2. Numerals which end in 0, e.g. 1011010. In this case, the numeral must actually end in 10, to avoid consecutive 0s, but the numeral before the 10 (10110 here) can be any $(n-2)$-digit numeral with no consecutive 0s.

Thus among numerals with no consecutive 0s we find

1. Those with n digits ending in 1 correspond to those with $(n-1)$ digits.
2. Those with n digits ending in 0 correspond to those with $(n-2)$ digits.

Hence if we let $f(n)$ be the number of such numerals with n digits we have

$$f(n) = f(n-1) + f(n-2).$$

This is the Fibonacci recurrence relation.
It can also be checked that

$$f(1) = 2 = F(3) \text{ and } f(2) = 3 = F(4),$$

hence it follows (by induction) that

$$\begin{aligned} f(n) = &\text{ number of } n\text{-digit numerals} \\ &\text{ with no consecutive 0s} \\ = &\ F(n+2). \end{aligned}$$

Questions

26.3 If a sequence satisfies the Fibonacci recurrence relation,

$$f(n) = f(n-1) + f(n-2),$$

must it agree with the Fibonacci sequence from some point onward?

Answer: NO!

To match the Fibonacci sequence it has to satisfy the same recurrence relation, but that is not enough. It also must agree with enough consecutive terms to force all subsequent terms to agree. For the Fibonacci sequence "enough terms" means two.

To be concrete, the constant sequence $0, 0, 0, 0, \ldots$ does satisfy the recurrence $f(n) = f(n-1) + f(n-2)$; however, it doesn't match any two terms of the Fibonacci sequence. There are many, many other examples. e.g. 4,4,8,12,20,32,52,84,... does match a term of the Fibonacci sequence (which?) but never two in a row.