Recurrence relations

A recurrence relation for a sequence $\{a_n\}$ is

an equation that expresses a_n in terms of

one or more of the previous terms of the sequence.

Sequence of Factorials

$$a_n = na_{n-1}$$

Geometric sequences

$$b_n = xb_{n-1}$$

Fibonacci sequence

$$f_n = f_{n-1} + f_{n-2}$$

Initial conditions of a recurrence relation

The initial conditions specify the terms

that precede the terms where

the recurrence relation can take effect.

Sequence of Factorials: $0! = 1$

$$a_n = na_{n-1} = n \cdot (n-1)! = n!$$

Geometric sequences: $x_0 = 1$

$$b_n = xb_{n-1} = x \cdot x^{n-1} = x^n$$

Fibonacci sequence: $f_0 = 0, f_1 = 1$

$$f_n = f_{n-1} + f_{n-2} = ?$$

Well-defined sequence

For recursively defined sequences, we might not have a closed formula for each term a_n. Still:

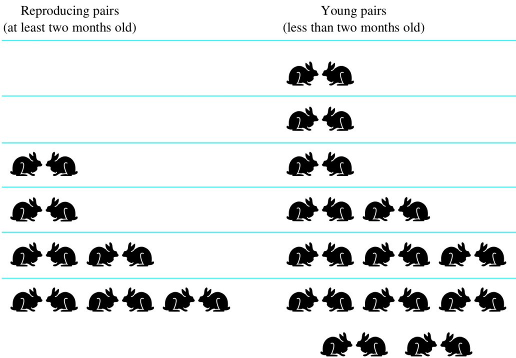
The sequence is well defined. In other words,

for every positive integer n, the value of a_n

is determined in an unambiguous way.

Fibonacci numbers

A pair of rabbits does not breed until they are 2 months old. After they are 2 months old, each pair of rabbits produces another pair each month. Assume that no rabbits ever die.



Month	Reproducing pairs	Young pairs	Total pairs
1	0	1	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8

FIGURE 1 **Rabbits on an Island.**

Codeword enumeration

A string of decimal digits is a valid codeword if it contains an even number of 0 digits.

- 1230407869 is valid,
- 120987045608 is not valid.

$$a_n = 9 \cdot a_{n-1} + (10^{n-1} - a_{n-1}) = 8a_{n-1} + 10^{n-1}$$

Idea: Take out first digit and count all possible substrings divided into: valid and not-valid.

Recursively defined structures

A rooted tree consists of a set of vertices containing a distinguished vertex called the root, and edges connecting these vertices (no loops or cycles allowed).

The set of rooted trees

BASIS STEP: A single vertex r is a rooted tree.

RECURSIVE STEP: Suppose that $T_1, T_2, \ldots, T_n$ are disjoint rooted trees, then the following is also a rooted tree:

A root r, which is not in any of $T_1, T_2, \ldots, T_n$, and add an edge from r to each of the roots of $T_1, T_2, \ldots, T_n$.

Recursively defined structures

RECURSIVE STEP: Suppose that $T_1, T_2, \ldots, T_n$ are disjoint rooted trees, then the following is also a rooted tree:

A root r, which is not in any of $T_1, T_2, \ldots, T_n$, and add an edge from r to each of the roots of $T_1, T_2, \ldots, T_n$.

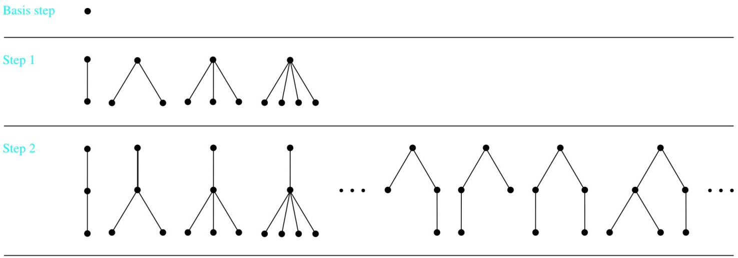


FIGURE 2 **Building Up Rooted Trees.**

Recursive and Iterative algorithms

RECURSIVE:

Successively reduce the desired computation
to the evaluation of the algorithm at smaller integers.

ITERATIVE:

Start with the output of the algorithm at the base cases;
and successively apply the recursive definition to
find the solution of the algorithm at larger integers.

Recursive and Iterative Fibonacci numbers

ALGORITHM 7 A Recursive Algorithm for Fibonacci Numbers.

procedure *fibonacci*(n: nonnegative integer)
if $n = 0$ **then return** 0
else if $n = 1$ **then return** 1
else return *fibonacci*$(n - 1)$ + *fibonacci*$(n - 2)$
{output is *fibonacci*(n)}

ALGORITHM 8 An Iterative Algorithm for Computing Fibonacci Numbers.

procedure *iterative fibonacci*(n: nonnegative integer)
if $n = 0$ **then return** 0
else
    $x := 0$
    $y := 1$
    for $i := 1$ to $n - 1$
        $z := x + y$
        $x := y$
        $y := z$
    return y
{output is the nth Fibonacci number}

Recursive vs. Iterative

- Use iterative algorithm:
  If you will compute all or most of previous terms to find solution.
  E.g. Fibonacci numbers.

- Use recursive algorithm:
  If you will need only need a few of previous solutions.
  E.g. Factorization of integers into prime factors.

Recursive algorithm for modular exponentiation

ALGORITHM 4 Recursive Modular Exponentiation.

procedure *mpower*(b, n, m: integers with $b > 0$ and $m \geq 2, n \geq 0$)
if $n = 0$ **then**
    return 1
else if n is even **then**
    return $mpower(b, n/2, m)^2$ **mod** m
else
    return $(mpower(b, \lfloor n/2 \rfloor, m)^2$ **mod** $m \cdot b$ **mod** m) **mod** m
{output is b^n **mod** m}

Proving correctness of a recursive algorithm

ALGORITHM 4 Recursive Modular Exponentiation.

procedure $mpower(b, n, m$: integers with $b > 0$ and $m \geq 2, n \geq 0)$
if $n = 0$ **then**
    return 1
else if n is even **then**
    return $mpower(b, n/2, m)^2$ **mod** m
else
    return $(mpower(b, \lfloor n/2 \rfloor, m)^2$ **mod** $m \cdot b$ **mod** $m)$ **mod** m
$\{$output is b^n **mod** $m\}$

Fix b and m integers with $b > 0$ and $m \geq 2$. Let

$P(n)$:Algorithm $mpower(b, n, m)$ outputs $b^n (\textbf{mod } m)$

To prove $\forall n \in \mathbb{N}\ P(n)$ is true, use strong induction.

Recall Strong Induction

BASIS STEP:

Verify that the proposition $P(1)$ is true.

INDUCTIVE STEP:

Show that the conditional statement

$$[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$$

is true for all k positive integer.