Counting rabbits

An introduction to recursion

Stuart Allen

CM1103

School of Computer Science & Informatics
Cardiff University

Counting rabbits

How many pairs of rabbits can be produced from a single pair in a year if every month each pair begets a new pair which from the second month on becomes productive?

Problem formulisation

Let D_r be the total number of *non-breeding* pairs at the end of the rth month

Let E_r be the total number of *breeding* pairs at the end of the rth month

Let F_r be the total number of pairs, where:

$$F_r = D_r + E_r$$

for $r \geq 1$

What is the value of F_r?

Example

- We know that $D_1 = 1$ (we start with one non-breeding pair)
- We know that $E_1 = 0$ (we start with no breeding pairs)

$\underline{1}$ $\underline{2}$

$\underline{8\ 8}$

Recursive solution

For $r \geq 2$:

$$E_r = E_{r-1} + D_{r-1} \text{ and } D_r = E_{r-1}$$

So:

$$E_r = F_{r-1}$$

and (for $r \geq 3$):

$$D_r = E_{r-1} = F_{r-2}$$

so that:

$$F_r = E_r + D_r = F_{r-1} + F_{r-2}$$

I.e. each value depends on the two preceding values.

Fibonacci numbers

The sequence of numbers defined by:

$$\begin{aligned} F_1 &= 1 \\ F_2 &= 1 \\ F_r &= F_{r-1} + F_{r-2} \qquad \text{for } r \geq 3 \end{aligned}$$

are known as the **Fibonacci numbers** and date back to 1202.

Calculating Fibonacci numbers: recursive functions

The definition of the Fibonacci numbers leads to a natural *recursive* implementation

Definition (recursion)

Recursion is the process of calling the function that is currently executing

Definition (recursive function)

A **recursive function** is a function that calls itself to solve a problem

Definition (base case)

The **base case** is a conditional branch in a recursive function that *does not* make a recursive call

Factorials

The factorial of an integer is defined as follows:

$$\begin{aligned} 0! &= 1 \\ n! &= n.(n-1).(n-2).\dots.1 \end{aligned}$$

Iterative solution

We can also calculate the Fibonacci numbers iteratively

Fibonacci numbers by iteration: Requires: integer n

```
If {$n$ == 1 OR $n$ == 2}
    Return 1
Else
    $q \leftarrow 1$ (Store the last value)
    $p \leftarrow 1$ (Store the $2^{nd}$ last value)
    For {$i$ = 3 to $n$}
        $temp \leftarrow p + q$
        $p \leftarrow q$
        $q \leftarrow temp$
    End for
    Return $p + q$
End if
```

Analytic solution

There is an analytic formula for F_n:

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

Proof

Can we prove this is true (easily)?

Proof by induction

Definition (Mathematical induction (weak))

To prove a propositional function $P(n)$ is true for all positive integers n, we complete two steps:

Basis step: Show that $P(1)$ is true

Inductive step: Show that $P(k)$ is true if $P(k-1)$ is true.

Definition (Propositional function)

A *propositional function* $P(x)$ returns either *true* or *false* depending on the value of the parameter x.

Example

Theorem

The sum of the first n odd numbers is n^2.

Proof:

Example *(continued)*

Example: Fibonacci sequence

Theorem

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Proof:

Example *(continued)*

Greatest common divisor

The *greatest common divisor* ($GCD(a, b)$) of two positive integers a and b is the largest divisor common to both a and b. I.e. the largest value d such that:

$$\begin{aligned} a &= dx \\ b &= dy \end{aligned}$$

for some positive integers x, y.

Example

- If $a = 3$ and $b = 5$, then $GCD(a, b) = 1$
- If $a = 12, b = 60$, then $GCD(a, b) = 12$
- $GCD(12, 90) = 6$

Recursive algorithm

Euclid's algorithm (from 300 B.C.) gives a *recursive* method to find the GCD.

Euclid's algorithm: Require: positive integers a and b

```
if $b = 0$ the GCD is $a$
otherwise the GCD is the GCD of $b$ and the remainder from dividing $a$
    by $b$
```

When to use recursion

- *Never, never, never* use recursion to solve problems such as calculating Fibonacci numbers or factorials (even though almost every textbook will give them as an example)
- Use recursion if:
    - it significantly simplifies the algorithm/implementation; **and**
    - it has no significant impact on the run time.

Fibonacci & GCD

Using Euclid's algorithm to find the GCD of F_{n+2} and F_{n+1} requires *exactly* n steps.

Tower of Hanoi

The *Tower of Hanoi* is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

Tower of Hanoi: recursive solution

Summary

- There are often many ways to solve a problem – you need to choose your algorithm carefully to balance simplicity, ease of implementation & speed
- Recursive methods work by calling themselves until they reach a trivial case
- Read sections 5.8 – 5.10 and 6.5 – 6.8 of *Think Python!*
- Define, understand and use recursion and proof by induction

Optional further work

- Use Python to verify that:
  Using Euclid's algorithm to find the GCD of F_{n+2} and F_{n+1} requires exactly n steps.
- Write Python code to time how long each of the 3 methods for calculating the Fibonacci numbers takes (for various values of n) [Hint: `import time` and call the `time.time()` function before and after calling the Fibonacci code. The difference between these two values will be the time taken in seconds.] You could try to use the `matplotlib` module to plot your results.