Laboratorio di Algoritmi e Strutture Dati (Laboratory of Algorithms and Data Structures)

Guido Fiorino
`guido.fiorino@unimib.it`

Introduction to recursion
(Last Modified: 13-03-2018)

A First Example

$f : \mathbb{N} \to \mathbb{N}$

$$f(x) = \begin{cases} 5 & \text{if } x = 0 \\ f(x-1) + 3 & \text{if } x \geq 1 \end{cases}$$

Note that:

- f is defined only on natural numbers;
- there is a base case (namely, $x = 0$);
- according to the definition, to compute $f(x)$, for a given $x \geq 1$, we have to compute the values of f for smaller values of x.

A First Example - How to compute f

$f : \mathbb{N} \to \mathbb{N}$

$$f(x) = \begin{cases} 5 & \text{if } x = 0 \\ f(x-1) + 3 & \text{if } x \geq 1 \end{cases}$$

Let us suppose that we want compute the value of $f(4)$. By applying the definition of f, we have

$$\mathbf{f(4) = f(3) + 3}$$

that is, to compute $f(4)$ we must compute $f(3)$. By applying the definition of f, we have

$$\mathbf{f(3) = f(2) + 3}$$

that is, to compute $f(3)$ we must compute $f(2)$. By applying the definition of f, we have

$$\mathbf{f(2) = f(1) + 3}$$

that is, to compute $f(2)$ we must compute $f(1)$. By applying the definition of f, we have

$$\mathbf{f(1) = f(0) + 3}$$

that is, to compute $f(1)$ we must compute $f(0)$. By applying the definition of f, we have

$$\mathbf{f(0) = 5}.$$

A First Example - How to compute f

Note that

$$\begin{aligned}
\mathbf{f(4)} &= \mathbf{f(3)+3} \\
\mathbf{f(3)} &= \mathbf{f(2)+3} \\
\mathbf{f(2)} &= \mathbf{f(1)+3} \\
\mathbf{f(1)} &= \mathbf{f(0)+3} \\
\mathbf{f(0)} &= \mathbf{5}
\end{aligned}$$

is a stack of recursive calls. Now that we know $f(0)$ we calculate the values for $f(1), \ldots, f(4)$ as follows:

$$\begin{aligned}
\mathbf{f(1)} &= \mathbf{f(0)+3} &&= \mathbf{5+3} &&= \mathbf{8} \\
\mathbf{f(2)} &= \mathbf{f(1)+3} &&= \mathbf{8+3} &&= \mathbf{11} \\
\mathbf{f(3)} &= \mathbf{f(2)+3} &&= \mathbf{11+3} &&= \mathbf{14} \\
\mathbf{f(4)} &= \mathbf{f(3)+3} &&= \mathbf{14+3} &&= \mathbf{17}
\end{aligned}$$

Exercise (method `f`)

Write a method `int f(int n)` *that uses the recursive definition of the function f to return the value of $f(n)$.*

Fibonacci

$fib : \mathbb{N} \to \mathbb{N}$

$$fib(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x = 1 \text{ or } x = 2 \\ fib(x-1) + fib(x-2) & \text{if } x \geq 3 \end{cases}$$

Note that:

- *fib* is defined only on natural numbers;
- there are three base cases (namely, $x = 0$, $x = 1$ and $x = 2$);
- according to the definition, to compute $fib(x)$, for a given $x \geq 3$, we have to compute the values of *fib* for smaller values of x.

Fibonacci

$fib : \mathbb{N} \to \mathbb{N}$

$$fib(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x = 1 \text{ or } x = 2 \\ fib(x-1) + fib(x-2) & \text{if } x \geq 3 \end{cases}$$

Tree of the recursive calls on fib (6) obtained by applying the definition

```
fib(6)
      fib(5)
            fib(4)
                  fib(3)
                        fib(2)
                        fib(1)
                  fib(2)
            fib(3)
                  fib(2)
                  fib(1)
      fib(4)
            fib(3)
                  fib(2)
                  fib(1)
            fib(2)
```

Exercise (method `fib`)

Write a method `int fib(int n)` *that uses the recursive definition of the function fib to return the n-th Fibonacci number .*

The Ackermann Function

A recursive definition of the Ackermann function $ack : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is:

$$ack(x, y) = \begin{cases} y + 1 & \text{if } x = 0; \\ ack(x-1, 1) & \text{if } y = 0; \\ ack(x-1, ack(x, y-1)) & \text{otherwise.} \end{cases}$$

Exercise (method `ack`)

Write a method `long ack(long x, long y)` *that returns the value of the Ackermann function by using the definition given above.*

We remark that the Ackermann function grows quickly:

$ack(0, 0) = 1;$
$ack(1, 1) = 3;$
$ack(2, 2) = 7;$
$ack(3, 3) = 61;\ ack(3, 4) = 125;\ ack(3, 5) = 253;\ ack(3, 6) = 509;\ ack(3, 7) = 1021;\ ack(3, 8) = 2045;$
$ack(4, 0) = 13;\ ack(4, 4) = 2^{2^{2^{2^{16}}}} - 3.$

Greater Common Divisor (gcd)

The Greatest Common Divisor between two non-negative integers A and B, denoted with $\text{gcd}(A, B)$ is the largest integer D that divides both A and B.
We get a recursive definition of gcd by using the fact that if $A >= B$, then $\text{gcd}(A, B) = \text{gcd}(A - B, B)$. As a matter of fact, a recursive definition of gcd is:

$$\text{gcd}(A, B) = \begin{cases} A & \text{if } B = 0 \\ \text{gcd}(A - B, B) & \text{if } B > 0 \text{ and } A >= B \\ \text{gcd}(B, A) & \text{otherwise} \end{cases}$$

Tree of the recursive calls on $\text{gcd}(5, 13)$ obtained by applying the definition

```
GDC(5,13)
        GDC(13,5)
                GDC(8,5)
                        GDC(3,5)
                                GDC(5,3)
                                        GDC(2,3)
                                                GDC(3,2)
                                                        GDC(1,2)
                                                                GDC(2,1)
                                                                        GDC(1,1)
                                                                                GDC(0,1)
                                                                                        GDC(1,0)
```

Exercise (method `gcdSlow`)

Write a method `long gcdSlow(long A, long B)` *that returns the value of* $\text{gcd}(A, B)$ *by using the definition for* gcd *given above.*

Greater Common Divisor (gcd)

A better definition of gcd is

$$\text{gcd}(A, B) = \begin{cases} A & \text{if } B = 0 \\ \text{gcd}(B, A\%B) & \text{otherwise} \end{cases}$$

that gives rise to a faster implementation requiring less nested recursive calls.

Tree of the recursive calls on $\text{gcd}(5, 13)$ obtained by applying the definition

```
GDC(5,13)
         GDC(13,5)
                  GDC(5,3)
                          GDC(3,2)
                                  GDC(2,1)
                                          GDC(1,0)
```

Exercise (method `gcdFast`)

write a method `long gcdFast(long A, long B)` *that returns the value of* $\text{gcd}(A, B)$ *by using the definition for* gcd *given above.*

Integer power of a number

To calculate a^b, with $a \in \mathbb{R}$ and $b \in \mathbb{N}$ we can apply the well known definition:

$$a^b = \begin{cases} 1 & \text{if } b = 0 \\ \underbrace{a * \cdots * a}_{b \text{ times}} & \text{otherwise} \end{cases}$$

A recursive definition is:

$$a^b = \begin{cases} 1 & \text{if } b = 0 \\ a * a^{b-1} & \text{otherwise} \end{cases}$$

According to the previous definition, the value a^b is found by b recursive calls and $b - 1$ multiplications. The following definition is mathematically equivalent but gives rise to a faster algorithm:

$$a^b = \begin{cases} 1 & \text{if } b = 0 \\ a^{\frac{b}{2}} * a^{\frac{b}{2}} = (a * a)^{\frac{b}{2}} & \text{if } b > 0 \text{ and } b \text{ even} \\ a^{\frac{b}{2}} * a^{\frac{b}{2}} * a = (a * a)^{\frac{b}{2}} * a & \text{otherwise } (b > 0 \text{ and } b \text{ odd}) \end{cases}$$

Integer power of a number

Exercise (method `powerSlow`)

Write a method `long powerSlow(long a, long b)` *that given* $b \geq 0$*, calculates* a^b *by using the first version of the recursive definition.*

Exercise (method `powerFast`)

Write a method `long powerFast(long a, long b)` *that given* $b \geq 0$*, calculates* a^b *by using the second version of the recursive definition.*

Counting the number of digits of an `int`

We want to devise a recursive method to count the number of digits of an integer non-negative number. As an example, the number 239811 contains six digits.

Exercise (method `int count(int value)`)

Write a recursive method `int count(int value)` *that returns the number of digits of the number stored in the parameter* `value`*. Suppose that the number in value is not negative.*

Printing natural numbers a digit at a time

We want to print out a non-negative number N in decimal, one digit at a time. As an example to print the number 926, first we need to print the digit 9, then the digit 2 and finally the digit 6. This task can be described recursively:

Getting the digits of a decimal number

to print a number N whose digits are $d_1 \dots d_{n-1} d_n$, first print the digits $d_1 \dots d_{n-1}$, then the digit d_n. The base case is when the number N has one digit.

Given $N = d_1 \dots d_n$ we can split it in the head $d_1 \dots d_{n-1}$ and the tail d_n by using the operators / and %.

Exercise (method `printDecimal`)

Write a method `void printDecimal(int n)` *that prints out* `n` *as a decimal number one digit at a time.*

Exercise (method `printDigit`)

Write a method `int printDigit(int n, int k)` *that prints out the k-th digit of* `n` *(assume that $k = 0$ means LSD).*

Conversion from base 10 to any base b

A variant of the previous problem is to print out a decimal number N in a given base (for sake of concreteness we limit the given base to a value between 2 and 16). The task can be recursively stated as follows:

Conversion of a natural number from base 10 to any base $b \geq 2$

Given the decimal number N, convert in base b the number N/b, then concatenate to the result the digit $N\%b$. The base case is when $N < b$: the conversion coincides N.

Exercise (method `convertDecimal`)

Write a method `void convertDecimal(int N, int b)` *that prints* `N` *in base* `b`*, assume* $2 \leq$ `b` ≤ 16*.*

A variant is

Exercise (method `convertDecimal`)

Write a method `String convertDecimal(int N, int b)` *that returns a string corresponding to* `N` *in base* `b`*, assume* $2 \leq$ `b` ≤ 16*.*

Reverse of the content of an integer variable

A variant of the previous problems is

Exercise (method `reverseInt`)

Write a method `int reverseInt(int N)` *that returns the value that corresponds to read the value in* `N` *from right to left. Example if* `N` *contains 157, then* `reverseInt` *returns 751. If the value in N is a number ending with one or more zeros, then they are lost in the conversion, thus if* `N` *contains 157000, then* `reverseInt` *returns 751.*