Programação em Python

Definições recursivas

2023

Departamento de Ciência de Computadores

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Conteúdo

Definições recursivas

Iteração e Recursão

Para resolver um problema complexo é comum partirmos esse problema em tarefas mais pequenas que são resolvidas, uma de cada vez, e no final, combinarmos os resultados obtidos.

Existem duas abordagens:

iteração repetimos uma sequência de operações um número variável de vezes até que o problema fique resolvido

recursão partimos o problema em problemas semelhantes mas mais pequenos que se consigam resolver, repetindo assim a sequência de operações e, com essas soluções, construimos a solução final.

Recursividade

Recursividade: definição de uma função, relação ou um processo em termos de si próprio.

Exemplos:

um antepassado é o pai ou a mãe ou um dos seus antepassados;

um diretório é uma estrutura que contém ficheiros e sub-diretórios;

uma árvore é uma folha ou um ramo que bifurca em duas ou mais sub-árvores.

Exemplos

Fatorial

Podemos definir o fatorial de n recursivamente:

$$
\begin{aligned}
0! &= 1 \\
n! &= n \times (n-1)! \qquad (n > 0)
\end{aligned}
$$

- A primeira equação define o caso base ($0!$)
- A segunda equação define o caso recursivo: $n!$ usando $(n-1)!$.

Assim fatorial fica definido para *todos* os inteiros não-negativos.

Fatorial (cont.)

Exemplo:

$$
\begin{aligned}
4! &= 4 \times 3! \\
&= 4 \times (3 \times 2!) \\
&= 4 \times (3 \times (2 \times 1!)) \\
&= 4 \times (3 \times (2 \times (1 \times 0!))) \\
&= 4 \times (3 \times (2 \times (1 \times 1))) \\
&= 24
\end{aligned}
$$

Fatorial (cont.)

Em Python: (Exemplo de execução no Python Tutor.)

```
def fatorial(n):
    if n==0:
        r=1                    # caso base
    else:
        r=n*fatorial(n-1)   # caso recursivo
    return r
```

- Uma função recursiva envolve a definição:
  - um ou mais casos base, para os quais a solução é conhecida
  - caso recursivo que chama a mesma função para resolver um problema mais simples e que se aproxima do(s) caso(s) base.

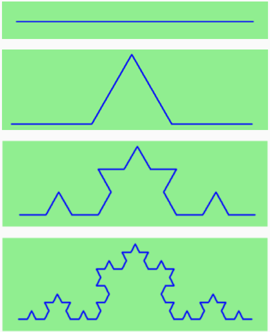
Fractal de Koch

$n = 0$

$n = 1$

$n = 2$

$n = 3$



A curva de ordem 0 é uma linha.

A curva ordem $n + 1$ consiste em quatro curvas de ordem n.

Fractal de Koch (cont.)

```python
from turtle import *

def koch(n, size):
    if n == 0:              # caso base: uma linha
        forward(size)
    else:                   # caso recursivo
        koch(n-1, size/3)   # 4 curvas com 1/3 do comprimento
        left(60)
        koch(n-1, size/3)
        right(120)
        koch(n-1, size/3)
        left(60)
        koch(n-1, size/3)
```

Fractal de Koch (cont.)

Podemos simplificar a definição anterior:

- `right(`α`)` é equivalente a `left(`$-\alpha$`)`;
- `left(0)` não altera a orientação.

Fractal de Koch (cont.)

```
def koch(n, size):
    if n == 0:                  # caso base: uma linha
        forward(size)
    else:                       # caso recursivo
        for angle in [60, -120, 60, 0]:
            koch(n-1, size/3) # avançar 1/3 do comprimento
            left(angle)
```

Números de Fibonacci

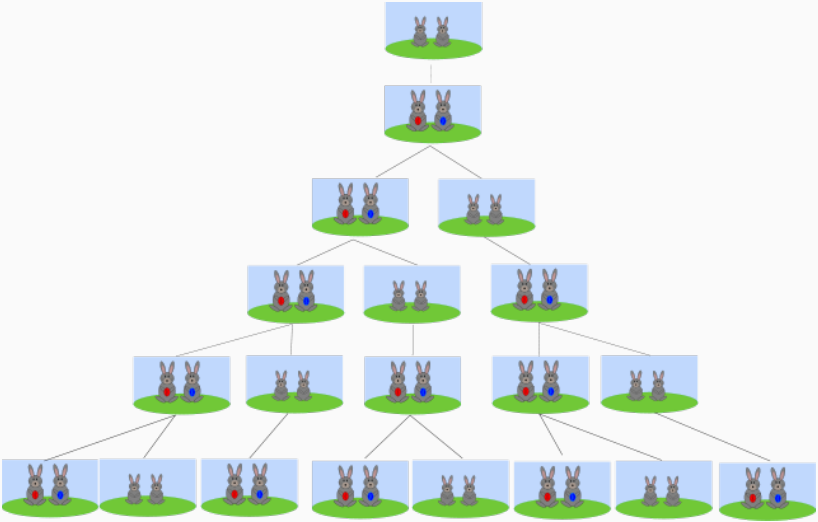
- Conhecidos na Índia (700 DC)
- Introduzidos na Europa por *Leonardo de Pisa* (1202 DC)
- Modelo simplificado para o crescimento de uma população de coelhos:
  - um par de coelhos são colocados num campo;
  - os coelhos podem acasalar com um mês;
  - no fim do segundo mês, nasce um novo par de coelhos;
  - os coelhos não morrem, e cada par produz sempre um novo par ao fim de um mês.

  Quantos pares de coelhos existem ao fim de um ano?

Números de Fibonacci (cont.)

- Ao fim de 1 mês ainda há só 1 par.
- Ao fim de 2 meses nasce um novo par, logo há 2 pares.
- Ao fim de 3 meses, nasce um segundo par da primeira fémea, logo há 3 pares.
- Ao fim de 4 meses, nasce o terceiro par da primeira fémea e o primeiro par da segunda fémea; logo há 5 pares.

Números de Fibonacci (cont.)



Números de Fibonacci (cont.)

O número de pares que existem ao fim de n meses é então a soma de:

1. o número de pares que existiam no mês anterior;
2. o número de pares que nascem (que é o número de pares no mês $n - 2$).

Simbolicamente

$$F_n = F_{n-1} + F_{n-2}$$

Os casos base são $F_0 = 1$ e $F_1 = 1$.

Implementação recursiva

Tradução direta para uma função recursiva.

```python
def fib(n):
    if n==0 or n==1:
        r=1
    else:
        r= fib(n-1) + fib(n-2)
    return r
```

Exemplo:

```
>>> fib(4)
5
```

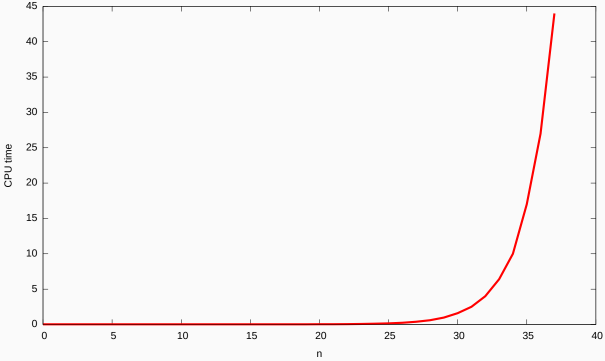
Contudo: esta implementação é pouco eficiente!

Eficiência

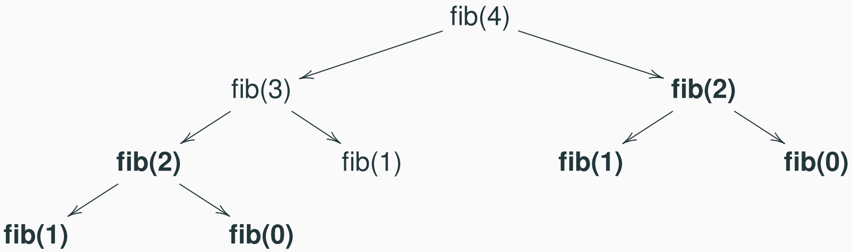
Construir uma tabela do tempo de computação em função de *n*.

```
import time
for n in range(1, 40):
    t0 = time.clock()
    fn = fib(n)
    t1 = time.clock()
    print("%d\t%f" % (n,t1-t0))
```

Eficiência (cont.)



Fibonacci: grafo de chamadas



A computação de `fib(2)` é repetida desnecessariamente…

Fibonacci com memorização

Vamos usar um dicionário para guardar resultados já calculados.

```
fib_memo = {0:1, 1:1} # dicionário global

def fastfib(n):
    if n in fib_memo:
       r = fib_memo[n]
    else:
       r = fastfib(n-1) + fastfib(n-2)
       fib_memo[n] =r
    return r
```

Fibonacci com memorização (cont.)

Utilização (resultados praticamente instantâneos):

```
>>> fastfib(40)
165580141
>>> fastfib(500)
225591516161936330872512695036072072046011324913758
190588638866418474627738686883405015987052796968498626L
```

Listas imbricadas

Vamos considerar listas imbricadas, i.e. listas cujos elementos são:

- números inteiros ou vírgula flutuante;
- outras listas imbricadas.

Exemplos:

```
[1, 2, 3]
[1, [2, 3], 4]
[[1,2],[3,4],[[5],6]]
```

Listas imbricadas (cont.)

Queremos definir uma função recursiva para somar *todos* os valores numa lista imbricada.

Note que a função pré-definida `sum` não serve (soma apenas listas de números).

```
>>> sum([1,2,3])
6
>>> sum([1, [2, 3], 4])
TypeError

>>> sum([[1,2],[3,4],[[5],6]]
TypeError
```

Listas imbricadas (cont.)

```python
def recsum(xs):
    # Somar todos os valores numa lista imbricada
    s = 0                   # acumulador da soma
    for x in xs:            # percorrer todos os elementos
        if type(x)==int or type(x)==float:
            s += x
        elif type(x)==list:
            s += recsum(x)  # somar sub-lista recursivamente
        else:
            raise TypeError("lista inválida")
    return s
```