Programação I
Aula 15 — Definições recursivas

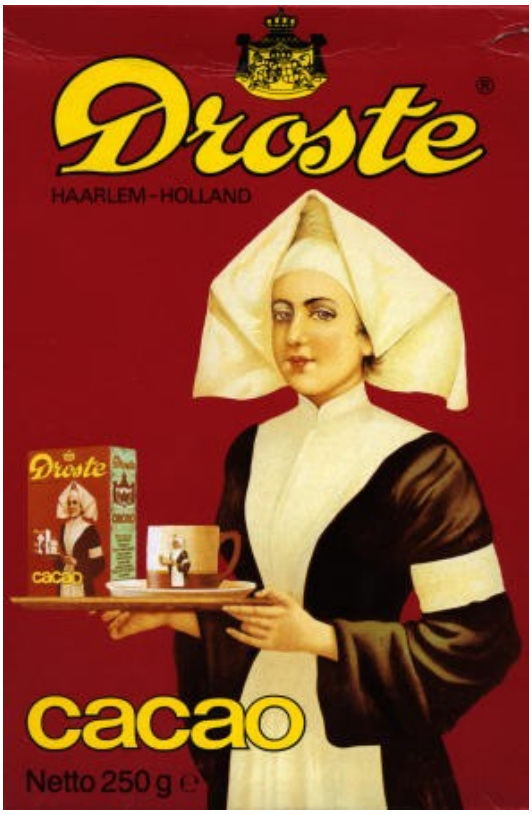
Pedro Vasconcelos
DCC/FCUP

Nesta aula

Recursividade

Uma definição diz-se recursiva se usamos na definição o próprio objeto que está ser definido.

Apesar de parecerem paradoxais, as definições recursivas são fundamentais em linguística, matemática e computação.



Algoritmos recursivos

Estratégia dividir para conquistar:

- exprimimos diretamente a solução do(s) *caso(s) base*;
- caso geral: resolvemos recursivamente *sub-problema(s) menor(es)* de forma a obter a solução do problema original.

Factorial

Podemos calcular o factorial recursivamente:

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

- A primeira equação define o caso base: factorial de 0
- A segunda equação define o caso geral: calcular $fact(n)$ usando $fact(n-1)$.

Exemplo

A definição anterior permite calcular o factorial de qualquer inteiro não-negativo.

Exemplo:

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

Factorial recursivo em Python

```
def fact(n: int) -> int:
    if n==0:            # caso base
        return 1
    else:               # caso geral
        return n*fact(n-1)
```

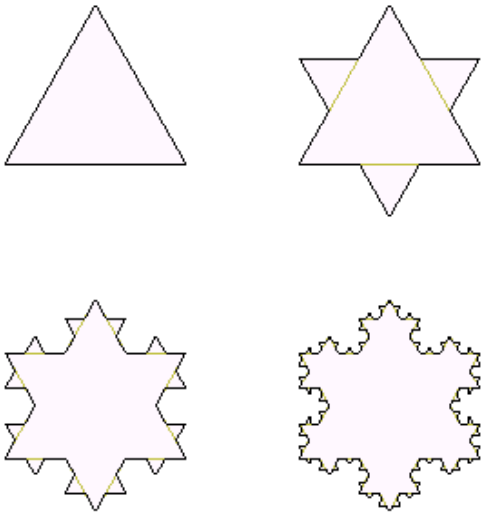

Exemplos

```
>>> fact(0)
1
>>> fact(4)
24
>>> fact(5)
120
>>> fact(-1)
maximum recursion depth exceeded
```

- Experimente visualizar usando o `http://pythontutor.com`
- O que significa o erro do último exemplo?

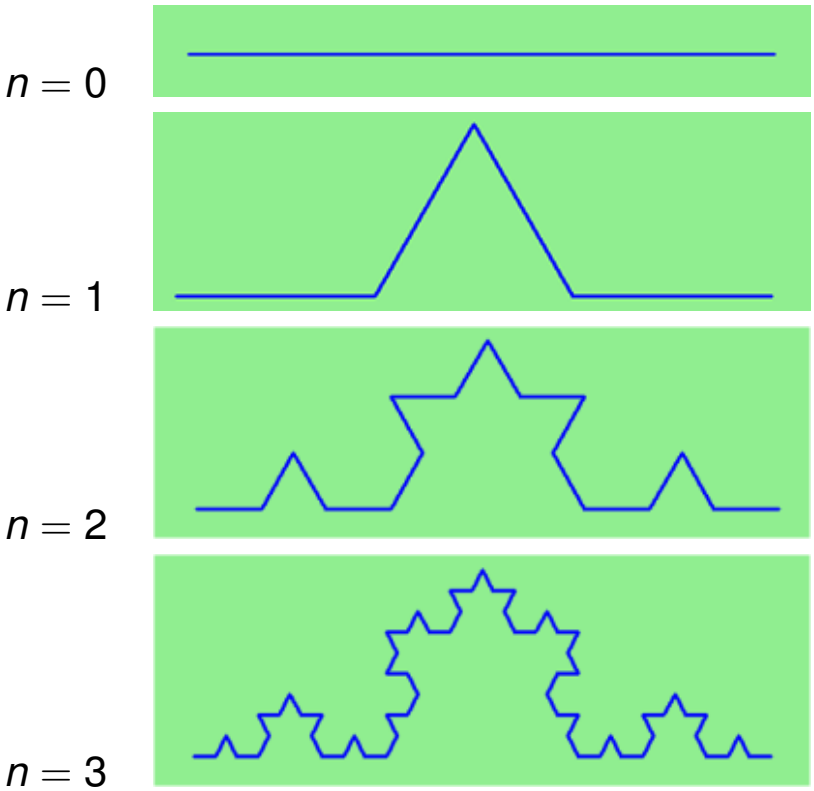
Floco de neve de Koch

- Uma curva fechada proposta em 1904 pelo matemático Helve von Koch
- Um dos primeiros exemplos de um figura *fractal*
- Vamos escrever um programa para desenhar aproximações desta figura



Fonte: `https://pt.wikipedia.org/wiki/Curva_de_Koch`

Curva de Koch



Curva de Koch (cont.)

Vamos definir um procedimento `koch(n,size)` para desenhar a curva de ordem *n*:

- a curva de ordem 0 é uma linha
- a curva de ordem *n* contém 4 curvas de ordem $n-1$

Curva de Koch (cont.)

```
from turtle import *

def koch(n: int, size: float):
    if n == 0:      # caso base: apenas uma linha
        forward(size)
    else:
        # caso geral: 4 curvas de ordem n-1 e 1/3 tamanho
        koch(n-1, size/3)
        left(60)
        koch(n-1, size/3)
        right(120)
        koch(n-1, size/3)
        left(60)
        koch(n-1, size/3)
```

Curva de Koch (cont.)

Vamos simplificar a definição anterior:

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

Curva de Koch (cont.)

```
def koch(n: int, size: float):
    if n == 0:          # caso base
        forward(size)
    else:               # caso geral
        for angle in [60, -120, 60, 0]:
            koch(n-1, size/3)
            left(angle)
```

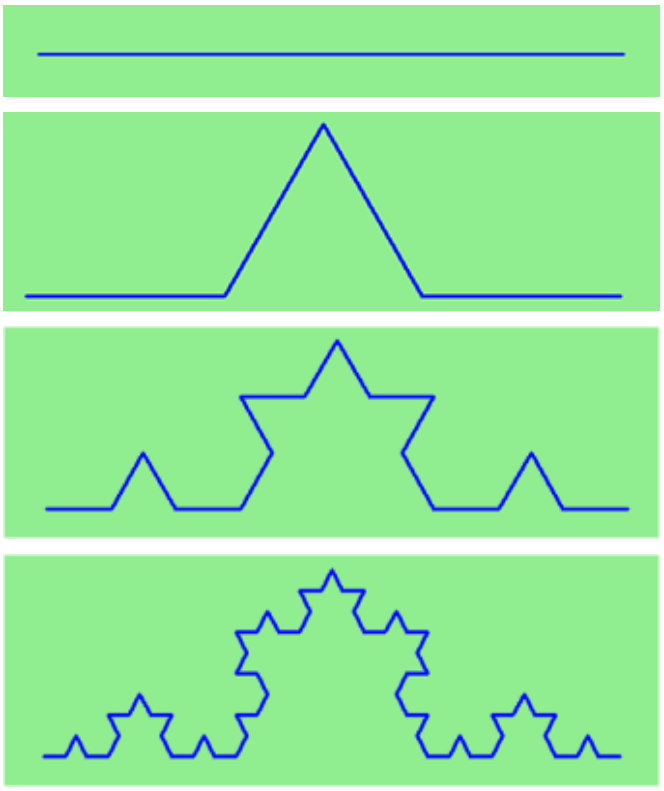
Exemplos

`koch(0,300)`

`koch(1,300)`

`koch(2,300)`

`koch(3,300)`

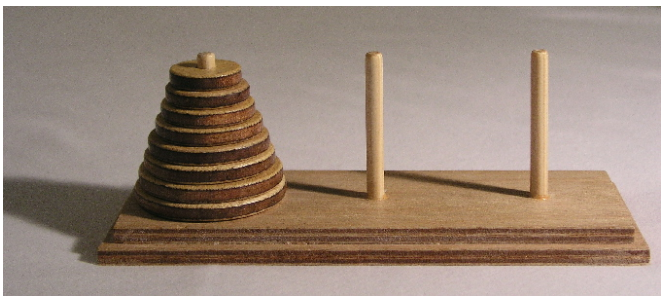


Exemplos (cont.)

- O procedimento `koch` desenha apenas 1/3 do “floco de neve”
- Para desenhar a figura completa: repetimos `koch`, rodando a tartaruga entre cada curva (exercício da folha 9)

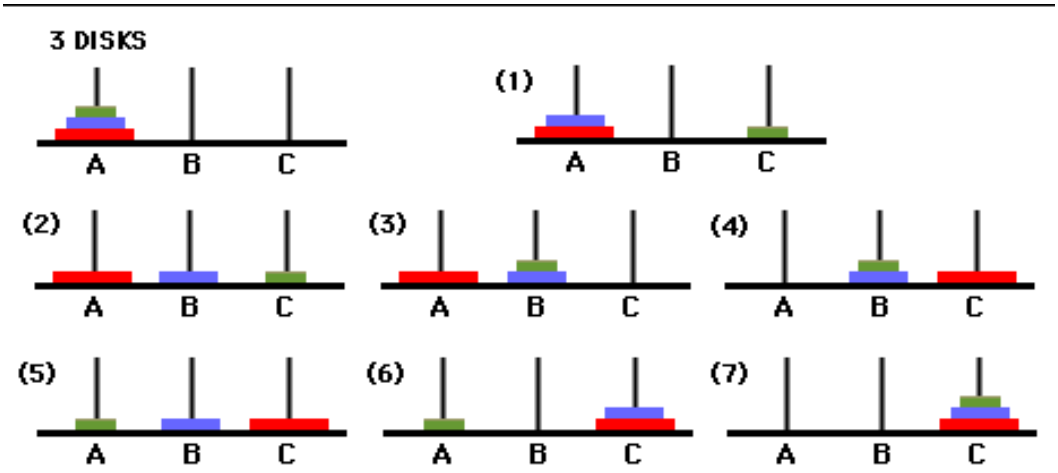
Torre de Hanoi

- Começamos com *N* discos empilhados com diametros crescentes
- Objetivo: mover todos os discos do primeiro para o terceiro pino
- Só podemos mover um disco de cada vez
- Não podemos colocar um disco sobre outro de diâmetro inferior



Fonte: `http://en.wikipedia.org/wiki/Tower_of_Hanoi`

Exemplo (3 discos)



Solução: A $\rightarrow$ C; A $\rightarrow$ B; C $\rightarrow$ B; A $\rightarrow$ C; B $\rightarrow$ A; B $\rightarrow$ C; A $\rightarrow$ C.

Solução recursiva

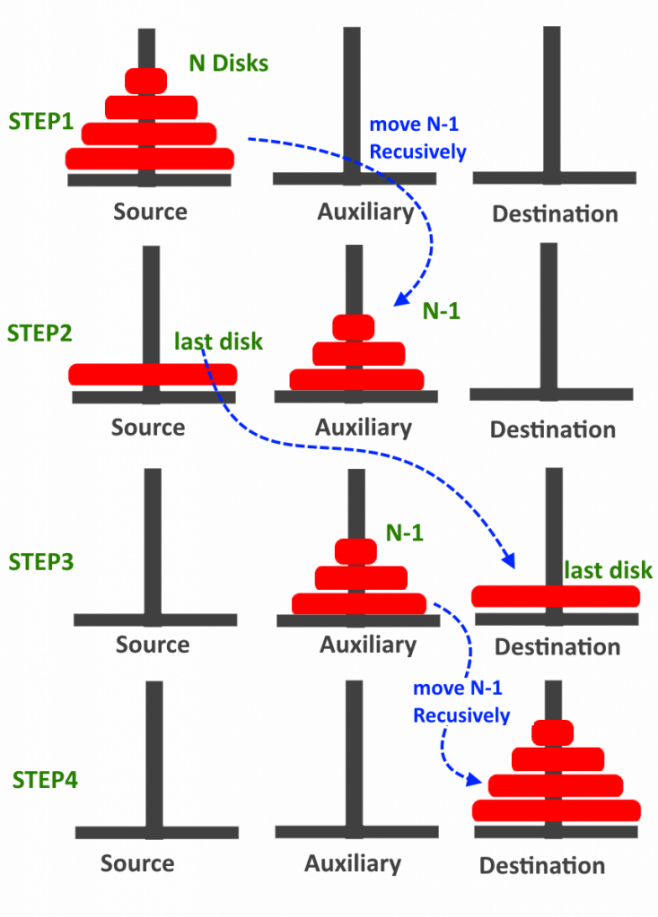
Para mover n discos da pilha X para Y usando Z como auxiliar:

se $n = 1$: (caso base) mover um único disco de X para Y

se $n > 1$: (caso geral)

1. recursivamente mover $n - 1$ discos de X para Z usando Y como auxiliar
2. mover um disco no topo de X para Y
3. recursivamente mover $n - 1$ discos de Z para Y usando X como auxiliar

Solução recursiva (cont.)



Implementação

```
def hanoi(n: int, X: str, Y: str, Z: str):
    '''Mover n discos da pilha X para Y
       usando Z como auxiliar.'''
    if n == 1:          # caso base
        print(X, '->', Y)
    else:               # caso geral (n>1)
        # mover n - 1 discos da origem para o auxiliar
        hanoi(n - 1, X, Z, Y)
        # mover um disco de X para Y
        print(X, '->', Y)
        # mover n - 1 discos do auxiliar para o destino
        hanoi(n - 1, Z, Y, X)
```


Solução para três discos

```
>>> hanoi(3,"A","C","B")
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

```
>>> hanoi(2,"A","B","C")
A -> C
A -> B
C -> B

>>> hanoi(2,"B","C","A")
B -> A
B -> C
A -> C
```

A solução para 3 discos contém as soluções de dois sub-problemas de 2 discos.

Observações

- O número de movimentos T para resolver o *puzzle* cresce bastante quando o número de discos N aumenta

N	2	3	4	5	6	7	8	9	10
T	3	7	15	31	63	127	255	511	1023

- Porquê. . . ?

Observações (cont.)

- Para tamanho 1 efetuamos 1 movimento
- Para tamanho $N > 1$:
  1. resolvemos 2 sub-problemas de tamanho $N - 1$;
  2. e efetuamos mais 1 movimento
- Obtemos uma *equação de recorrência*

$$T(1) = 1$$
$$T(N) = 2 \times T(N-1) + 1$$

  A solução é $T(N) = 2^N - 1$
- O número de movimentos cresce de forma exponencial com o número de discos

Crescimento exponencial

Se cada movimento demorar 1 segundo:

- 10 discos demoram $2^{10} - 1$ segundos $\approx$ 17 minutos
- 20 discos demoram $2^{20} - 1$ segundos $\approx$ 12 dias
- 30 discos demoram $2^{30} - 1$ segundos $\approx$ 34 anos
- 40 discos demoram $2^{40} - 1$ segundos $\approx$ 35 mil anos
- 60 discos demoram mais do que idade atual do universo ($\approx 1.1 \times 10^{12}$ anos)!

Só podemos resolver problemas com crescimento exponencial para tamanhos pequenos!