Recursão (e indução)

↳ "chama a si mesmo"

```
def fun(x):
    return fun(x)
```

PROBLEMA!

$$F: \mathbb{N} \longrightarrow \mathbb{N}$$

dada por

$$F(n) = \begin{cases} n \ , \text{ se } n \leq 1 \\ F(n-2) + F(n-1) \ , \text{ c.c.} \end{cases}$$

c.c. ← caso contrário

$$F(5) = F(3) + F(4)$$

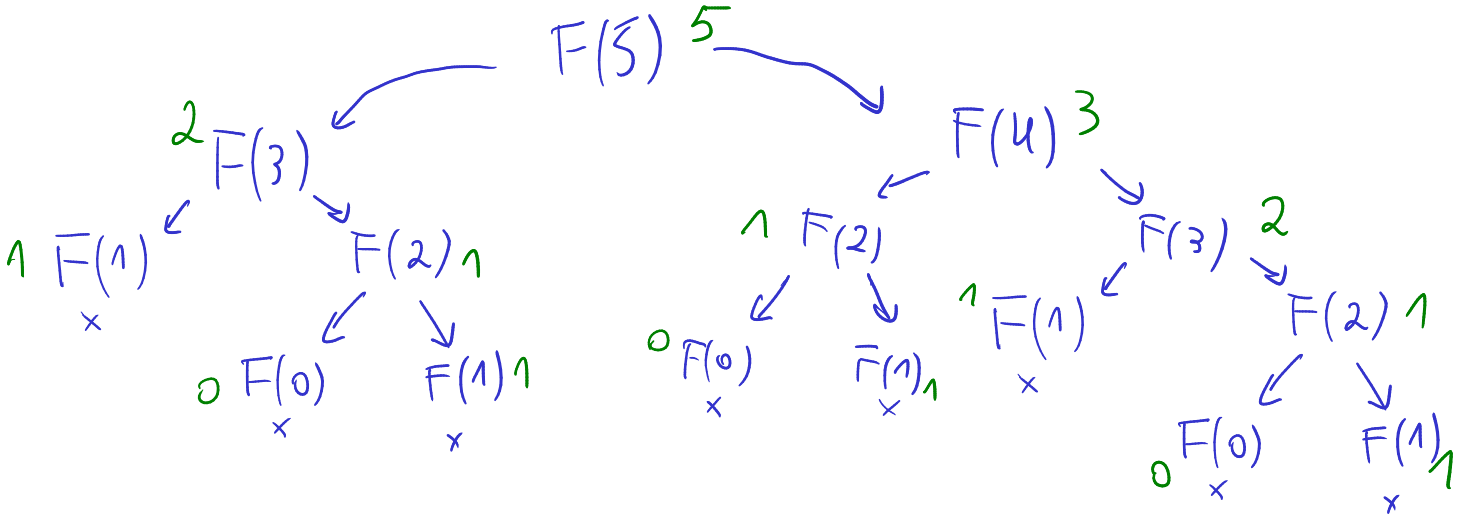
$$= \left(F(1) + F(2) \right) + \left(F(2) + F(3) \right)$$

$$= \left(F(1) + \left[F(0) + F(1) \right] \right) + \left(\left[F(0) + F(1) \right] + \left[F(1) + F(2) \right] \right)$$

$$= \left(\text{---------- (1 + 0 + 1 + 0 + 1) ----------} \right) + \left(F(0) + F(1) \right)$$

$$= 1 + 0 + 1 + 0 + 1 + 1 + 0 + 1$$

$$= 5$$


F(5) 5
2 F(3)
F(4) 3
1 F(1)
x
F(2) 1
1 F(2)
F(3) 2
0 F(0)
x
F(1) 1
x
0 F(0)
x
F(1) 1
x
1 F(1)
x
F(2) 1
0 F(0)
x
F(1) 1
x

Definições recursivas parecem ferir uma "lei de ouro" da matemática: só podemos usar, em um dado momento, coisas que fizemos antes deste momento!

Exs sempre funções $\mathbb{N} \to \mathbb{N}$

$$F_0(n) = \begin{cases} 8, & \text{se } n \leq 42 \\ 3 \cdot F_0(n+1), & \text{c.c.} \end{cases}$$

$$F_1(n) = \begin{cases} 8, & \text{se } n \geq 42 \\ 3 \cdot F_1(n+1), & \text{c.c.} \end{cases}$$

Ideia: funciona quando chamadas recursivas caminham em direção de, convergem para, se aproximam de chamadas básicas (não recursivas)

Ou, numa visão "temporal": funciona quando posso ver a definição não como uma, mas como várias, de

maneira que possam ser organizadas e respeitar a lei de ouro.

Conjectura de Collatz: a definição abaixo funciona?

Não sabemos!

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

$$C(n) = \begin{cases} 1, & \text{se } n \in 1 \\ C(n/2), & \text{se } n \geq 2 \text{ e } n \text{ é par} \\ C(3 \cdot n + 1), & \text{c.c.} \end{cases}$$

Outra forma mais visual:
para uma def. recursiva funcionar,
conceitualmente devo conseguir
escrever os casos em uma lista
de maneira que, se eu representar
"caso A depende do caso B" como
uma seta de A para B, todas
as setas têm direção para a esquerda.

Merge sort (altíssimo nível)

Dada lista L...

- se L tem no máx 1 elem.,
  retorne L
- senão, divida L em duas "metades"
  L', L''

- recursivamente ordene L' | L''
- faça a junção de L', L'' (MERGE)
  e retorne (onde "junção" é operação definida em Estruturas de Dados :)

"Análise"

Momento 1: caso base (listas L com $|L| \leq 1$)

" 2: listas de tamanho 2

" 3: " " " 3 ou 4

" 4 : " " " 5 a 8

⋮

" n : " " " $2^{n-2}+1$ a 2^{n-1} (?)

Mais "precisamente" (?)

Uma definição recursiva é "válida", "bem feita", quando:

- na verdade pode ser vista como diversos (talvez infinitos?) CASOS
- entre alguns casos há uma relação de DEPENDÊNCIA

- Os casos podem ser (todos) ORGANIZADOS de maneira que, para quaisquer casos A & B, se o caso A depende do caso B, então o caso A vem depois do caso B na organização

→ logo há casos que não dependem de outros (CASOS BASE)

→ os outros (CASOS RECURSIVOS)

Mais exemplos:

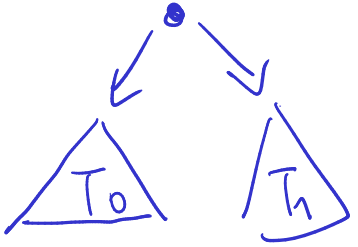
1) "Árvores estritamente binárias"

def "comum": árvores não-vazias onde nós internos (i.e., nós que não são folhas) têm exatamente 2 filhos

def recursiva:

Base: • é uma A.E.B. (só uma "raiz" e mais nada)

Recursivo:



onde T_0, T_1 são AEB

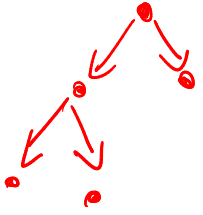
é AEB

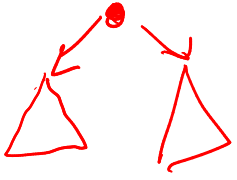
Rascunho

Momento 1 :

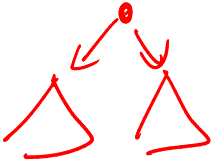
" 2 :

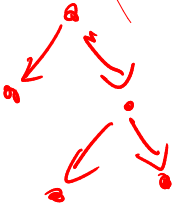
" 3 :



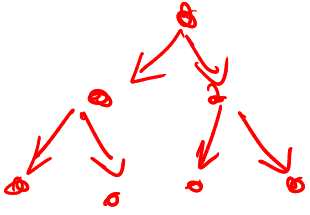


)





)



2) "Palavras" (no sentido de Ling. Formais) em um alfabeto Σ

def "comum": uma sequência finita de elementos de Σ

def recursiva 1:

Base: ε (a palavra vazia)

Recursivo: $w \frown \langle x \rangle$, onde w é palavra e $x \in \Sigma$

(sequência; concatenação)

def recursiva 2:

Base: . ε (a palavra vazia)

. $\langle x \rangle$ | onde $x \in \Sigma$

Recursivo: $w \cap u$ | onde w, u são palavras