11 Aula 11: 17/SET/2019

Tópico de hoje: introdução à *recursão*

11.1 Hanoi

Problema: torres de hanoi.

Esse problema serve para introduzir *raciocínio* recursivo para resolução de problemas. execução.

Slides: `slides_hanoi.pdf`

Programas: `hanoi.py`, `hanoir.py`, `minimal_hanoi.py` (animação)

Começamos a aula descrevendo o problema e escrevendo o `main()`.

```
def main():
    no_discos = int(argv[1])
    hanoi (no_discos,'A','B','C')
```

Que produz para `no_discos == 3` a saída

```
    1: mova o disco 1 da torre A para a torre C.
    2: mova o disco 2 da torre A para a torre B.
    3: mova o disco 1 da torre C para a torre B.
    4: mova o disco 3 da torre A para a torre C.
    5: mova o disco 1 da torre B para a torre A.
    6: mova o disco 2 da torre B para a torre C.
    7: mova o disco 1 da torre A para a torre C.
```

O número do movimento é opcional. Talvez possamos fazer sem o contador e depois inserir o contador.

Damos tempo para eles e elas fazerem a função.

```
#-------------------------------------------------------------
def hanoi(no_discos, origem, auxiliar, destino, mov=0):
    '''(int,str,str,str, int) -> None

    Recebe o numero de discos 'no_discos', o pinos
    origem, auxiliar e destino e o número do proximo
    movimento e imprime as mesagens que resolvem o
    problema das torres de Hanoi.
    '''
    if no_discos > 0:
        hanoi(no_discos-1, origem, destino, auxiliar, mov)
        print("%d: mova o disco %d da torre %s para a torre %c."
          %(mov, n, origem, destino))
        mov += 1
        hanoi(no_discos-1, auxiliar, origem, destino, mov)
```

11.2 Estrutura e leis :-)

Estrutura típica (*pattern*) de uma função recursiva:

```
if instância é pequena:
     resolva-a diretamente
     retorne
reduza-a a uma instância 'menor' do mesmo problema
aplique o método à instância 'menor'
monte a solução da instância atual e retorne
```

Três leis de recursão :-), um algoritmo recursivo deve:

- ter um **caso base**
- alterar seu estado de maneira a se **aproximar do caso base**
- chamar a si mesmo direta ou indiretamente.

11.3 Fatorial

Problema: dado um inteiro não negativo `k` calcular `k!`.

Programas: `fibonacciI.py`, `fibonacciR.py`, `fibonacciR-s.py`

Nesse problema recursão é evidente e serve para nos familiarizarmos com o mecanismo recursivo em programação.

```
fatorial(3) = 6

fatorial(3)
  fatorial(2)
    fatorial(1)
      fatorial(0)
fatorial(3) = 6
```

```
def fatorialI(n):
    '''(int) -> int
    Recebe um inteiro n e retorna n!.
    '''
    fat = 1
    for i in range(2,n+1):
        fat *= i
    return fat

def fatorialR(n):
    '''(int) -> int

    Recebe um inteiro n e retorna n!.
    '''
    if n == 0: return 1
    return n * fatorial(n-1)
```

11.4 Máximo

Slide: `slides_maximo.pdf`

Programas: `maximoR0.py`, `maximoR1.py`, `maximoR2.py`, `maximoR3.py`

O objetivo aqui é exercitar o raciocínio recursivo.

```
def maximoR(n,v):
    '''(int,list) -> item

    Recebe um inteiro positivo n e uma lista v e retorna
    o maior elemento das n primeiras posições.
    '''
    if n == 1: return v[0]
    x = maximoR(n-1,v)
    if x > v[n-1]: return x
    return v[n-1]

def maximo(v):
    '''(list) -> item

    Recebe uma lista v e retorna o maior elemento da lista.
    '''
    return maximoR(len(v), v)
```