12 Aula 12: 19/SET/2019

12.1 Aula passada

Problema: torres de hanoi.

Esse problema serviu para introduzir *raciocínio* recursivo para resolução de problemas. Fizemos a implementação e no final a simulação com diagrama de execução.

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

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

Comentamos as **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.

Também comentamos sobre a estrutura *típica* de uma solução recursiva

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

12.2 Aula de hoje

Possíveis passos para esta aula são:

- revisão da aula passada
- analise da solução do problema das torres de hanoi: recorrências, consumo de tempo exponencial
- `finonacciR()` e fibonacciI(): comparação do consumo de tempo das soluções recursivas e iterativas. Solução recursiva **resolve o mesmo subproblema várias vezes** se não for usada *memorização*

12.3 Rever recursão e Hanoi

Slides: `slides_hanoi_intro.pdf` e 'slides_hanoi_epilogo.pdf"

Programas: `hanoi.py`, `hanoir.py`, `minimal_hanoi.py`,

O objetivos e revisarmos recursão e analisarmos o consumo de tempo da solução. Nas contas aparece uma recorrência.

12.4 Fibonacci

Slides: `slides_fibonacci.pdf`

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

Fazer análise experimental de `fibonacciI()` e `fibonacciR()`:

```
% time python fibonacciI.py 4
fibonacci(4) = 3

real        0m0.040s
user        0m0.028s
sys         0m0.008s

% python fibonacciR.py 4
fibonacci(4) = 3

% python fibonacciR-s.py 4
fibonacciR(4)
  fibonacciR(3)
    fibonacciR(2)
      fibonacciR(1)
      fibonacciR(0)
    fibonacciR(1)
  fibonacciR(2)
    fibonacciR(1)
    fibonacciR(0)
fibonacci(4) = 3
```

Observar que o alto consumo de tempo de `fibonacciR()` não tem nada a ver com a recursão, mas sim com resolver o mesmo subproblema várias vezes.