025089 – Projeto e Análise de Algoritmos

Aula 04

lizier@dc.ufscar.br

Análise de Algoritmos

- Revisão dos não-recursivos

- Algoritmos Recursivos:

Análise de Algoritmos Recursivos

- Escolha do parâmetro (ou parâmetros) de entrada que define o tamanho do problema (ou entrada)
- Identificação a operação básica (mais frequente) do algoritmo
- Contagem do número de vezes que a operação básica é executada, dependendo apenas do tamanho da entrada. Se depender também do tipo da entrada, analise o melhor e pior caso separadamente
- **Defina a relação de recorrência, com uma condição inicial apropriada, para a operação básica escolhida**
- **Resolva a relação de recorrência e classifique a ordem de crescimento do algoritmo**

Exemplo 1

- Fatorial:

$$n! = 1 \cdot \ldots \cdot (n-1) \cdot n = (n-1)! \cdot n \quad \text{for } n \geq 1$$

ALGORITHM $F(n)$

//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

Exemplo 1

- Fatorial:

$$F(n) = F(n-1)*n \qquad , \text{para todo } n > 0$$

$$F(0) = 1$$

- Relação de recorrência para a operação de multiplicação:

$$T(n) = T(n-1) + 1 \qquad , \text{para } n > 0$$

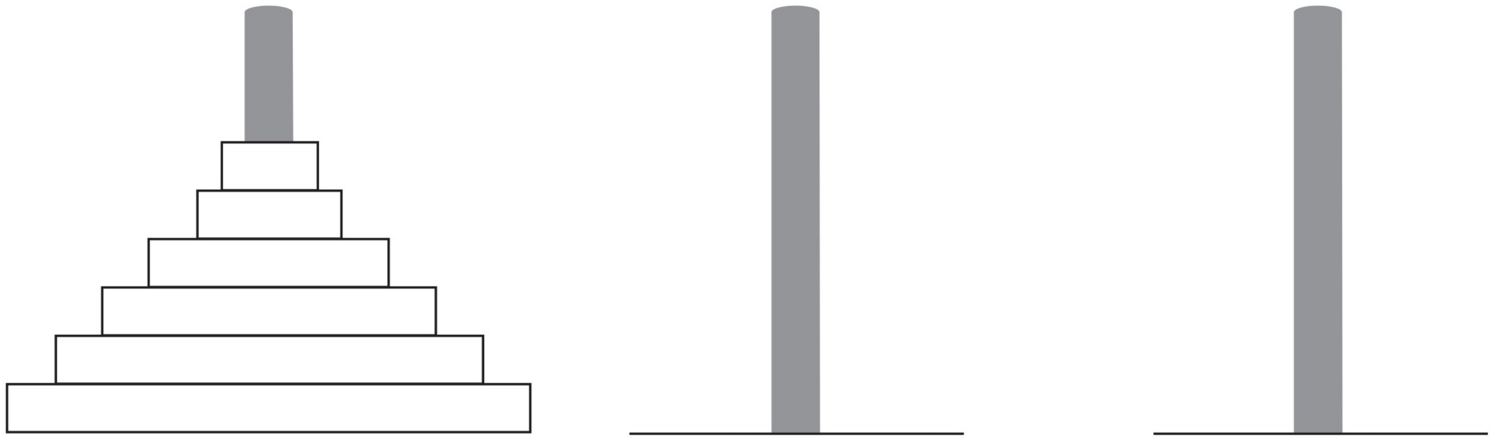
$$T(0) = 0$$

- Pelo método da substituição resolve

$$T(n) = T(n - i) + i \quad \rightarrow \quad i=n \quad \rightarrow \quad T(n - n) + n = T(0) + n = n$$

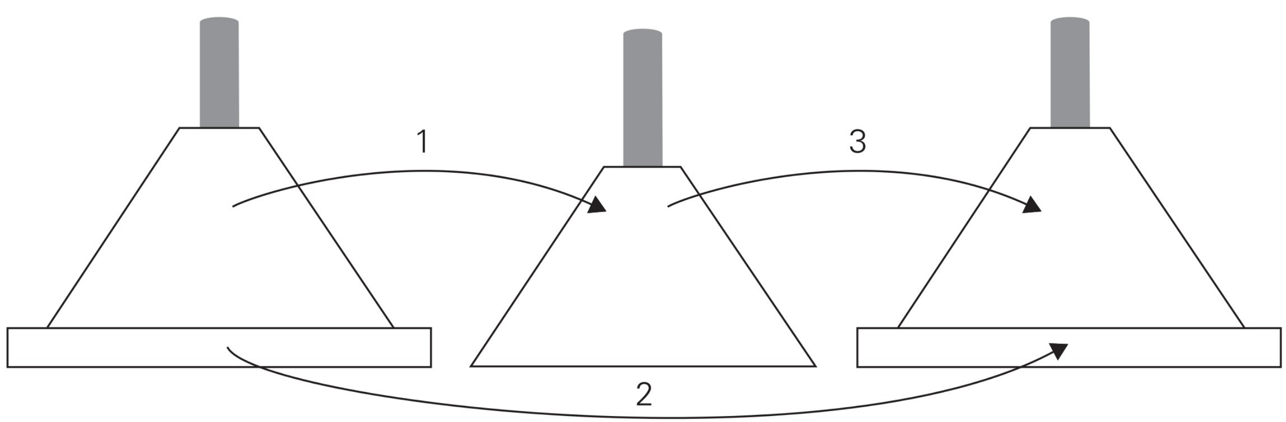
Exemplo 2

- Torres de Hanói:



Exemplo 2

- Torres de Hanói:
  - Solução recursiva:
    - Para mover *n* discos do pino 1 para o 3 (usando o pino 2 como auxiliar):
      - Mova *n-1* discos do pino 1 para o pino 2 (usando o pino 3 como auxiliar)
      - Mova o maior disco do pino 1 para o pino 3
      - Mova os *n-1* discos do pino 2 para o pino 3 (usando o pino 1 como auxiliar)



Exemplo 2

- Torres de Hanói:
  - Relação de recorrência para a operação de mover:

$$T(n) = T(n-1) + 1 + T(n-1) = 2*T(n-1) + 1 \quad , para\ n > 1$$

$$T(1) = 1$$

  - De novo o método da substituição resolve

Exemplo 2

- Torres de Hanói:
  - Relação de recorrência para a operação de mover:

$$T(n) = T(n-1) + 1 + T(n-1) = 2*T(n-1) + 1 \quad , para\ n > 1$$

$$T(1) = 1$$

  - De novo o método da substituição resolve

$$T(n) = 2^i * T(n - i) + 2^i - 1$$

$$\rightarrow i=n-1 \rightarrow 2^{(n-1)}*T(1) + 2^{(n-1)} - 1 = 2^n - 1$$

Exemplo 3

- Número de bits:

ALGORITHM $BinRec(n)$

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return $BinRec(\lfloor n/2 \rfloor) + 1$

- Relação de recorrência:

Exemplo 3

- Número de bits:

ALGORITHM $BinRec(n)$

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
if $n = 1$ **return** 1
else return $BinRec(\lfloor n/2 \rfloor) + 1$

- Relação de recorrência:

$T(n) = T(n/2)+1$, *para* $n > 1$

$T(1) = 0$

  - *Método da substituição de novo (detalhe: n=2^k)*

Regra da suavidade

- Funções suaves:

  Se e somente se: $f(2n) \in \Theta(f(n))$

  - Def.:

    $f(bn) \in \Theta(f(n))$, *iff* $c*f(n) \leq f(bn) \leq d*f(n)$, *para* $n \geq n0$
  - São suaves: lineares, lineares-logarítmicas, polinomiais, …
  - Não são suaves: exponenciais e fatoriais

- Regra da suavidade:
  - Se $f(n)$ é suave e $T(n) \in \Theta(f(n))$ para valores de n que são potências de b, com $b \geq 2$, então $T(n) \in \Theta(f(n))$ para qualquer valor de n

Relações de Recorrência

- Podem ser bem complicadas!
  - Linear ou não-linear
  - Ordem 1, 2, … k
  - Homogênea ou não-homogênea

- Método da substituição:
  - Até a condição inicial (*backward substitution*)
  - A partir da condição inicial (*forward substitution*)

- Teorema master
  - Recorrência por um fator constante

Teorema Master
(simplificado)

- Problemas de tamanho ***n***, divididos em ***a*** partes menores de ***n/b*** tamanhos, e combinados por ***f(n)***, sendo ***f(n)*** polinomial, e para a notação *Big-Oh:*

  Dada a recorrência $\boldsymbol{T(n) \leq aT(n/b) + cn^d}$, para alguma constante **c**, temos:

  Caso 1: $\mathbf{T(n) = O(n^d)}$, se $\mathbf{\log_b(a) < d}$

  Caso 2: $\mathbf{T(n) = O(n^d \log(n))}$, se $\mathbf{\log_b(a) = d}$

  Caso 3: $\mathbf{T(n) = O(n^{\log b(a)})}$, se $\mathbf{\log_b(a) > d}$

Recorrências

- Exemplos:
  - $T(n) = 9T(n/3) + n$
    - $O(n^2)$
  - $T(n) = T(2n/3) + 1$
    - $O(\log n)$
  - $T(n) = 2T(n/2) + n$
    - $O(n \log n)$

Merge Sort

  - $T(n) = 8T(n/2) + n^2$
    - $O(n^3)$

Exemplo 5

- Grafo completo

```
ALGORITHM GraphComplete(A[0..n − 1, 0..n − 1])
    //Input: Adjacency matrix A[0..n − 1, 0..n − 1]) of an undirected graph G
    //Output: 1 (true) if G is complete and 0 (false) otherwise
    if n = 1 return 1   //one-vertex graph is complete by definition
    else
        if not GraphComplete(A[0..n − 2, 0..n − 2]) return 0
        else for j ← 0 to n − 2 do
                if A[n − 1, j] = 0 return 0
            return 1
```