

# MC-102 — Aula 16

## Introdução à Recursão

Instituto de Computação – Unicamp

Segundo Semestre de 2007

# Roteiro

- 1 Recursão
- 2 Aspectos técnicos da recursão
- 3 Exemplos

# Recursividade

## Recursividade

Um objeto é dito recursivo se pode ser definido em termos de si próprio.

# Componentes da Recursão

Toda recursão é composta por

- **Um caso base**, uma instância do problema que pode ser solucionada facilmente. Por exemplo, é trivial fazer a soma de uma lista com um único elemento.
- **Uma ou mais chamadas recursivas**, onde o objeto define-se em termos de si próprio, tentando convergir para o caso base. A soma de uma lista de  $n$  elementos pode ser definida a partir da lista da soma de  $n - 1$  elementos.

# Recursão na matemática

Como definir recursivamente a soma abaixo?

$$\sum_{k=m}^n k = m + (m + 1) + \cdots + (n - 1) + n$$

# Recursão na matemática

Primeira definição recursiva

$$\sum_{k=m}^n k = \begin{cases} m & \text{se } n = m \\ \sum_{k=m}^{n-1} k + n & \text{se } n > m \end{cases}$$

# Recursão na matemática

Segunda definição recursiva

$$\sum_{k=m}^n k = \begin{cases} m & \text{se } n = m \\ m + \sum_{k=m+1}^n k & \text{se } n > m \end{cases}$$

# Recursão na computação

```
int soma(int m, int n) {  
    if (m == n)  
        return n;  
    else  
        return m + soma(m+1, n);  
}
```

Veja o código: `soma.c`

# Pilha de execução

- A cada chamada de função o sistema reserva espaço para parâmetros, variáveis locais e valor de retorno.

main	s
	ret: ??
soma	m: 5 n: 10
	ret: ??
soma	m: 6 n: 10
	...

## Estouro de pilha de execução

“To understand recursion you must first understand recursion.”

- O que acontece se a função não tiver um caso base?
- O sistema de execução não consegue implementar infinitas chamadas. (Lembre-se, somente Chuck Norris conta até o infinito).

Veja o código `rec-infinita.c`

# Fatorial

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{se } n > 0 \end{cases}$$

```
int fatorial(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n * fatorial (n-1);  
}
```

## Potência

$$x^n = \begin{cases} \frac{1}{x^{(-n)}} & \text{se } n < 0 \\ 1 & \text{se } n = 0 \\ x \cdot x^{(n-1)} & \text{se } n > 0 \end{cases}$$

```
double pot(double x, int n) {  
    if (n == 0) return 1;  
    else if (n < 0)  
        return 1/pot(x, -n);  
    else  
        return x*pot(x, n-1);  
}
```

## Questões de desempenho

- É sempre mais simples usar recursão?
- É mais eficiente?

Veja pot-iterativa.c

# Torres de Hanói



- Os discos devem ser movidos da origem para o destino.
- Nenhum disco pode ser colocado sobre outro menor.

## Torres de Hanói

- Problema inventado pelo matemático francês Édouard Lucas em 1883.
- Também conhecido como Torres de Brahma.

*No grande templo de Brahma em Benares, numa bandeja de metal sob a cúpula que marca o centro do mundo, três agulhas de diamante servem de pilar a sessenta e quatro discos de ouro puro. Incasavelmente, os sacerdotes transferem os discos um de cada vez, de agulha para agulha, obedecendo sempre à lei imutável de Brahma: nenhum disco se poderá sobrepor a um menor.*

## Torres de Hanói

*No início do mundo, todos os sessenta e quatro discos de ouro foram dispostos na primeira das três agulhas, constituindo a Torre de Brahma. No momento em que o menor dos discos for colocado de tal modo que se forme uma vez mais a Torre de Brahma numa agulha diferente da inicial, tanto a torre como o templo serão transformados em pó e o ribombar de um trovão assinalará o fim do mundo.*

# Torres de Hanói



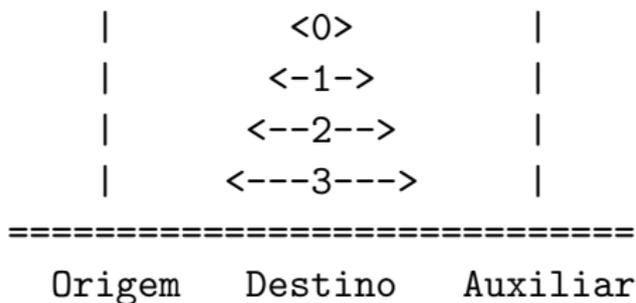
- Podemos mover n-1 discos para a torre auxiliar

# Torres de Hanói



- Podemos mover o disco maior para a torre destino

# Torres de Hanói



- Podemos mover n-1 discos da torre auxiliar para a torre destino.

## Torres de Hanói

```
void move_hanoi (int n, int origem, int destino,
                 int auxiliar) {
    if (n == 0) return;
    if (n == 1) {
        move_disco(origem, destino);
        return;
    }
    move_hanoi(n-1, origem, auxiliar, destino);
    move_disco(origem, destino);
    move_hanoi(n-1, auxiliar, destino, origem);
}
```

## Torres de Hanói

Seja  $T(n)$  o número de movimentos necessários para resolver o problema com  $n$  discos.

Pelo algoritmo anterior,  $T(n)$  é dado pela seguinte recursão:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2T(n-1) + 1 & \text{se } n \geq 2 \end{cases}$$

Fazendo alguns cálculos, chegamos à solução seguinte:

$$T(n) = 2^n - 1, \text{ para } n \geq 1.$$

Quanto tempo para resolver o problema com 64 discos?