

# AED1 – Aula 3

## Recursividade

Prof. Dr. Aldo Díaz  
[aldo.diaz@ufg.br](mailto:aldo.diaz@ufg.br)

Instituto de Informática  
Universidade Federal de Goiás



03/11/2022

# Agenda

- 1 Introdução
- 2 Tipos de recursividade
- 3 Escrevendo funções recursivas
- 4 Vantagens e desvantagens
- 5 Saiba mais ...

# Introdução



Que características em comum têm estas imagens?

# Introdução



Extraído de [The developer's life.](#)



# Introdução

## Recursividade

- Mecanismo pelo qual é possível definir uma função em termos de si mesma.

### Exemplo – Fatorial

O fatorial de um número  $x \in \mathbb{N}$  é definido como:

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

# Introdução

## Fatorial v1.0 (iterativo)

```
1 unsigned long int factorial1(unsigned long int num) {  
2     unsigned long int i, fat=1;  
3  
4     for(i=num; i > 1; i--)  
5         // TODO: Me complete ... (=  
6  
7     return(fat);  
8 }
```

## Fatorial v2.0 (recursivo)

```
1 unsigned long int factorial2(unsigned long int num) {  
2     if(num == 0)  
3         // TODO: Caso trivial  
4  
5     else  
6         // TODO: Caso nao trivial  
7 }
```

# Introdução

## Recursividade – Destaques

- Cada chamada nova trabalha um problema *menor que* o problema da chamada anterior.
- Toda função recursiva deve prever *quando* o processo de recursão será interrompido.
- A versão recursiva não necessariamente é mais rápida. Ela pode ser até mais lenta, devido ao custo de manter as chamadas recursivas.
- A versão recursiva não necessariamente traz economia de memória. Isto porque os valores processados são mantidos em *pilhas*.

# Introdução

## Exemplo – Potenciação

A potência de um número  $x \in \mathbb{R}$ , é definida por:

$$x^n = \prod_{k=0}^{n-1} x$$

## Potenciação v1.0 (iterativa)

```
1 float potencia1(float x, int n) {  
2     float result = 1.0;  
3  
4     for(int i=0; i<n; i++)  
5         result *= x;  
6  
7     return(result);  
8 }
```

# Introdução

## Potenciação v2.0 (recursiva)

```
1 float potencia2(float x, int n) {  
2     if(n == 0)  
3         // TODO: Caso trivial 1  
4     else if(n == 1)  
5         // TODO: Caso trivial 2  
6     else  
7         // TODO: Caso nao trivial  
8 }
```

## Potenciação v3.0 (recursiva)

```
1 float potencia3(float x, int n) {  
2     if(n == 0)  
3         // TODO: Caso trivial  
4     else  
5         // TODO: Caso nao trivial  
6 }
```

# Introdução

## Recursividade – Diferenças

- `potencia2` possui 2 casos triviais,  $n = 0$  e  $n = 1$ .
- `potencia3` possui 1 caso trivial,  $n = 0$ .

Portanto, `potencia3` utilizará uma chamada recursiva a mais.

# Tipos de recursividade

1. Recursão direta
  - 1.1. Linear
  - 1.2. Aninhada
  - 1.3. Não linear (em árvore)
2. Recursão indireta

# Tipos de recursividade

## 1. Recursão direta

A rotina possui uma chamada para si mesma.

- **Exemplo:** As funções factorial e potencia.

### 1.1. Recursão direta – Linear

Há uma **única** chamada recursiva a ela.

### 1.2. Recursão direta – Aninhada

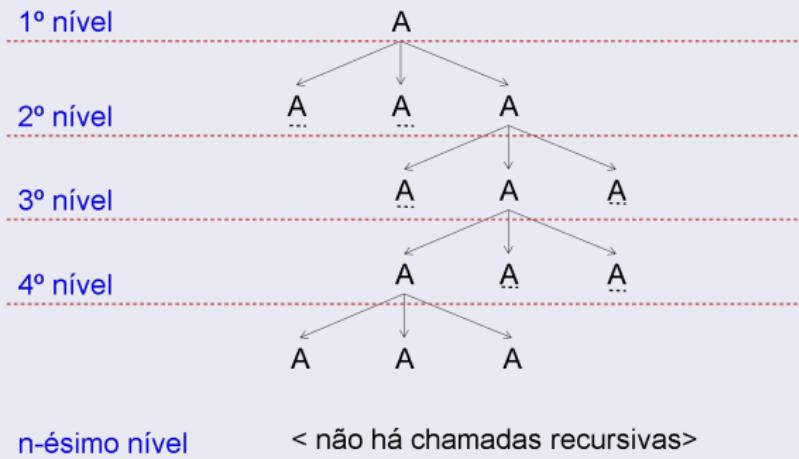
A chamada recursiva de uma função recebe como argumento outra chamada recursiva à própria função.

- **Exemplo:** A função `nested(x, nested(a, b))`.

# Tipos de recursividade

## 1.3. Recursão direta – Não linear (em árvore)

Há **duas ou mais** chamadas recursivas a ela. A sequência de chamadas se assemelha a uma “árvore”:



# Tipos de recursividade

## 2. Recursão indireta

A rotina possui uma chamada para outra rotina que, por sua vez, possui uma chamada para a rotina anterior. O círculo de chamadas deve ser interrompido em algum momento.

- **Exemplo:** As funções  $f$  e  $g$  definidas sobre os números naturais:

$$f(x) = \begin{cases} 2 & , \text{se } x \leq 1 \\ 2g(x+1) & , \text{se } x > 1 \end{cases}$$

$$g(y) = \begin{cases} 3 & , \text{se } y \geq 5 \\ 2f(y-2) & , \text{se } y < 5 \end{cases}$$

# Escrevendo funções recursivas

## Exemplo: Somatório

Forneça uma solução recursiva ao problema de somar todos os elementos de um vetor de números.

$$S = \sum_{k=0}^{n-1} v[k]$$

## Somatório v1.0 (iterativo)

```
1 unsigned long int somatorio1(unsigned long int vetor[], unsigned int n) {  
2     unsigned long int soma = 0;  
3  
4     for(int i=0; i<n; i++)  
5         soma += vetor[i];  
6  
7     return(soma);  
8 }
```

# Escrevendo funções recursivas

## Somatório v2.0 (recursivo)

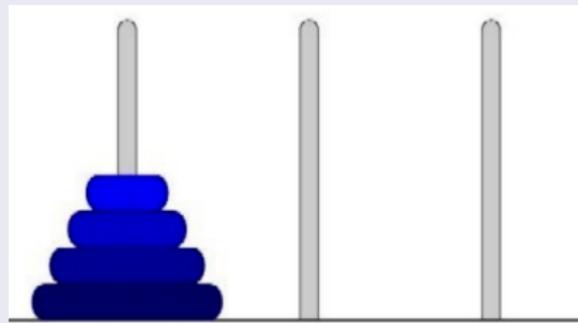
1. Determine os casos triviais e sua solução.
  - O menor problema a ser resolvido é somatorio(vetor, 1)
2. Determine o problema imediatamente menor
  - É o problema somatorio(vetor, n-1)
3. Use (2) para resolver o problema atual
  - $\text{vetor}[n-1] + \text{somatorio}(\text{vetor}, n-1)$

```
1 unsigned long int somatorio2(unsigned long int vetor[], unsigned int n) {  
2     if(n == 1)  
3         // TODO: Caso trivial  
4     else  
5         // TODO: Caso nao trivial  
6 }
```

# Escrevendo funções recursivas

## Exemplo: Torre de Hanói

Considerando que os  $n$  discos estão na primeira haste, A.



\* Entenda as regras [aqui](#).

# Escrevendo funções recursivas

Exemplo: Torre de Hanói (algoritmo recursivo)

1. Se  $n = 1$ , *mova* o disco da haste  $A$  para a  $B$  e **pare**
2. *Mova* os  $(n - 1)$  discos de  $A$  para  $C$ , usando  $B$  como auxiliar
3. *Mova* o disco remanescente de  $A$  para  $B$
4. *Mova* os  $(n - 1)$  discos de  $C$  para  $B$ , usando  $A$  como auxiliar

\* **Dica:** Você não precisa saber como mover ...

# Escrevendo funções recursivas

## Torre de Hanói v1.0 (recursivo)

```
1 #include <stdio.h>
2
3 void hanoi(int nDiscos, char deHaste, char paraHaste, char auxHaste) {
4     // Se um disco somente, mova-o e retorne
5     if(nDiscos==1) {
6         printf("Move disco 1 de %c para %c\n", deHaste, paraHaste);
7         return;
8     }
9     // Move (n-1) discos de A para C, usando B
10    hanoi(nDiscos-1, deHaste, auxHaste, paraHaste);
11    // Move disco remanescente de A para B
12    printf("Move disco %d de %c para %c\n", nDiscos, deHaste, paraHaste);
13    // Move (n-1) discos de C para B, usando A
14    hanoi(nDiscos-1, auxHaste, paraHaste, deHaste);
15 }
```

# Escrevendo funções recursivas

## Torres de Hanói v2.0 (iterativo)

```
1 // As hastes sao 0, 1 e 2
2 // Os discos estao inicialmente na haste 0
3 #include <stdio.h>
4
5 void main(void) {
6     int n;
7
8     printf("Quantos discos? ");
9     scanf("%d", &n);
10    printf("\n\n");
11    for(int x=1; x<(1 << n); x++)
12        printf("Mova da haste %i para a haste %i\n", (x&x-1)%3, ((x|x-1)+1)
13        %3);
14 }
```

# Escrevendo funções recursivas

## Torre de Hanói – Alguns questionamentos

1. Qual é o total de passos para mover os  $n$  discos?
2. Qual será mais rápida, a versão *recursiva* ou a *iterativa*?

# Escrevendo funções recursivas

Exemplo: Máximo divisor comum (algoritmo recursivo)

```
1 unsigned long int mdc(unsigned long int a, unsigned long int b) {  
2     if(b == 0)  
3         return(a);  
4     else  
5         return(mdc(b, (a % b)));  
6 }
```

# Escrevendo funções recursivas

Exemplo: Número binomial

Um número binomial,  $C_k^n$ , é definido como:

$$C_k^n = \frac{n!}{k!(n-k)!}$$

com  $n, k \in \mathbb{N}$  e  $n \geq k$ .

					1				
				1	1				
			1	2	1				
		1	3	3	1				
	1	4	6	4	1				
1	5	10	10	5	1				
1	6	15	20	15	6	1			
1	7	21	35	35	21	7	1		

Figura: Triângulo de Pascal.

# Escrevendo funções recursivas

## Exemplo: Número binomial (algoritmo recursivo)

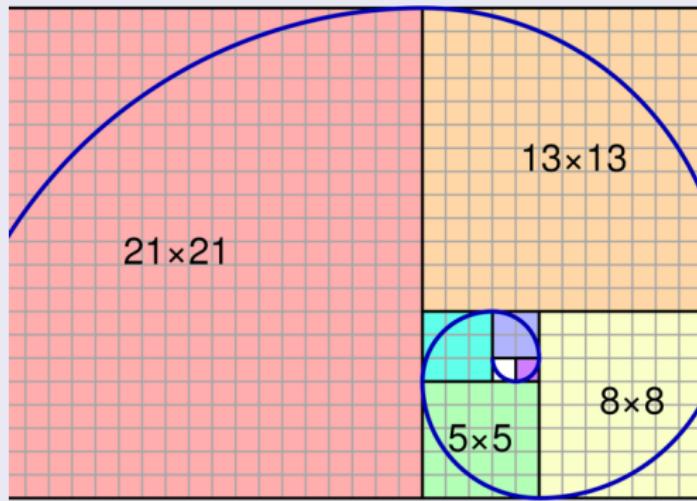
1. Se  $n = k$ , ou  $k = 0$ , retorne 1 (um) e pare
2. Retorne o valor da soma dos binomiais  $C_k^{n-1}$  e  $C_{k-1}^{n-1}$

```
1 unsigned long int binomial1(unsigned long int n, unsigned long int k) {  
2     if((n == k) || (k == 0))  
3         return(1);  
4     else  
5         return(binomial1(n-1, k) + binomial1(n-1, k-1));  
6 }
```

# Escrevendo funções recursivas

## Exemplo: Espiral de Fibonacci

Projete uma função recursiva para calcular o  $n$ -ésimo número da sequência de Fibonacci.



# Escrevendo funções recursivas

## Exemplo: Espiral de Fibonacci (algoritmo recursivo)

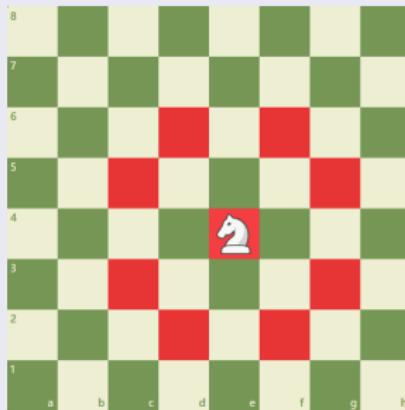
1. Se  $n = 1$ , ou  $n = 2$ , retorne o valor 1 e pare
2. Retorne a soma dos elementos nas posições  $(n - 1)$  e  $(n - 2)$

```
1 unsigned long int fibonacci(unsigned long int n) {  
2     if(n == 1 || n == 2)  
3         return(1);  
4     else  
5         return(fibonacci(n-1) + fibonacci(n-2));  
6 }
```

# Escrevendo funções recursivas

## Exemplo: Passeio do cavalo

O *passeio do cavalo* é um problema matemático envolvendo o movimento do cavalo no tabuleiro de xadrez.



# Escrevendo funções recursivas

## Exemplo: Passeio do cavalo

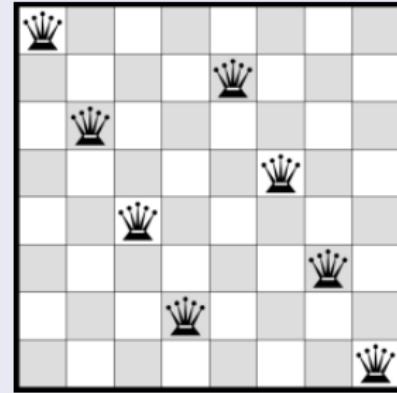
Um cavalo é colocado no tabuleiro vazio e, seguindo as regras do jogo de xadrez, precisa passar por todas as casas exatamente uma vez em movimentos consecutivos.

1. Pesquise possíveis algoritmos (iterativos e recursivos) para este problema
2. Implemente um deles
3. Qual é o número de movimentos necessários, no algoritmo, para resolver o problema?
4. Pense na generalização: E se o tabuleiro tivesse um tamanho arbitrário  $n \times n$ ,  $n \in \mathbb{N}$  com  $n \geq 3$ ?

# Escrevendo funções recursivas

## Exemplo: 8 rainhas

As 8 rainhas é um problema matemático envolvendo a colocação de 8 rainhas no tabuleiro de xadrez, de tal maneira que nenhuma delas seja capaz de capturar a outra.



# Escrevendo funções recursivas

## Exemplo: 8 rainhas

A *solução por força bruta* seria: “Gere todas as possíveis configurações do tabuleiro contendo 8 rainhas e, admita como solução aquela que obedeça à restrição imposta”.

O **problema com esta solução** é que o número de configurações possíveis é gigantesco!

1. Pesquise sobre os possíveis algoritmos (iterativos e recursivos) para resolver este problema
2. Pense na generalização: E se o tabuleiro tivesse um tamanho arbitrário  $n \times n$ ,  $n \in \mathbb{N}$ , devendo-se, portanto, serem inseridas  $n$  rainhas?

# Vantagens e desvantagens

## Vantagens

1. Algumas estruturas de dados são *naturalmente* recursivas:
  - Listas encadeadas
  - Árvores binárias
2. Problemas *naturalmente* recursivos:
  - Deslocamento em listas encadeadas
  - Caminho em árvores binárias
  - Avaliação de expressões

## Desvantagens

1. Em geral, as funções recursivas são *mais lentas* que as iterativas
2. Recursão excessiva pode estourar a *pilha de execução*
3. Funções recursivas podem ser complicadas de expressar textualmente

Saiba mais ...

youtu.be

## Canal do Prof. Wanderley de Souza Alencar (INF/UFG)

**Wanderley de Souza Alencar**  
327 suscriptores

**INICIO** VÍDEOS LISTAS COMUNIDAD CANALES INFORMACIÓN

**Resolução de exercícios envolvendo o conceito de "recursividade" para a elaboração de programas.**  
Wanderley de Souza Alencar • 204 visualizações • hace 2 años  
Nesta videoaula são resolvidos, em detalhe, três problemas que envolvem a utilização do conceito de recursividade. As soluções são programadas em C. (!) impressão de números naturais...

**Aula 02 - Recursividade (Parte 1)**  
Wanderley de Souza Alencar • 208 visualizações • hace 2 años  
Aula 02 - Recursividade Apresenta uma introdução à recursividade no contexto da programação de computadoras.

**Aula 02 - Recursividade (Parte 02)**  
Wanderley de Souza Alencar • 301 visualizações • hace 4 años  
Aula de Introdução à recursividade, abordando as diferenças tipos de recursão (direta e indireta), com exemplos de uso em C.

**Aula 02 - Recursividade (Parte 04)**  
Wanderley de Souza Alencar • 173 visualizações • hace 4 años  
Aula de Introdução à recursividade, onde serão abordados as vantagens e desvantagens da utilização do conceito de recursividade, bem como apresentada a ação "Saiba mais..." indicada.

**Aula 02 - Recursividade (Parte 03)**

# Saiba mais ...

## Livros

- CORMEN, T. H. et al. *Algoritmos* (tradução). 2 ed. Rio de Janeiro: Elsevier, 2002.
- FERRARI, R. et al. *Estruturas de dados com jogos*. 1 ed. São Paulo: Elsevier, 2014.
- GOODRICH, M. T.; TAMASSIA, R. *Estruturas de dados e algoritmos em Java*, 4 ed. São Paulo: Bookman, 2007.

## Saiba mais ...

### Artigos

- CONRAD, A. et al. *Solution of the knight's Hamiltonian path problem on chessboards*. In: Discrete Applied Mathematics, v. 50, p. 125-134, 1994.
- BAI, S. et al. *Generalized knight's tour on 3D chessboards*. In: Discrete Applied Mathematics, v. 158, p. 1727-1731, 2010.
- BELL, J.; STEVENS, B. *A survey of known results and research areas for n-queens*. In: Discrete Applied Mathematics, v. 309, p. 1-31, 2009.
- DEKKING, F. M. et al. *Queens in exile - non-attacking queens on infinite chess boards*.
- FUÁ, P.; LIS, K. *Comparing Python, Go and C++ on the n-queens problem*.