

# Recursividade

---

Fabio Lubacheski  
fabio.aglubacheski@sp.senac.br

# Recursividade

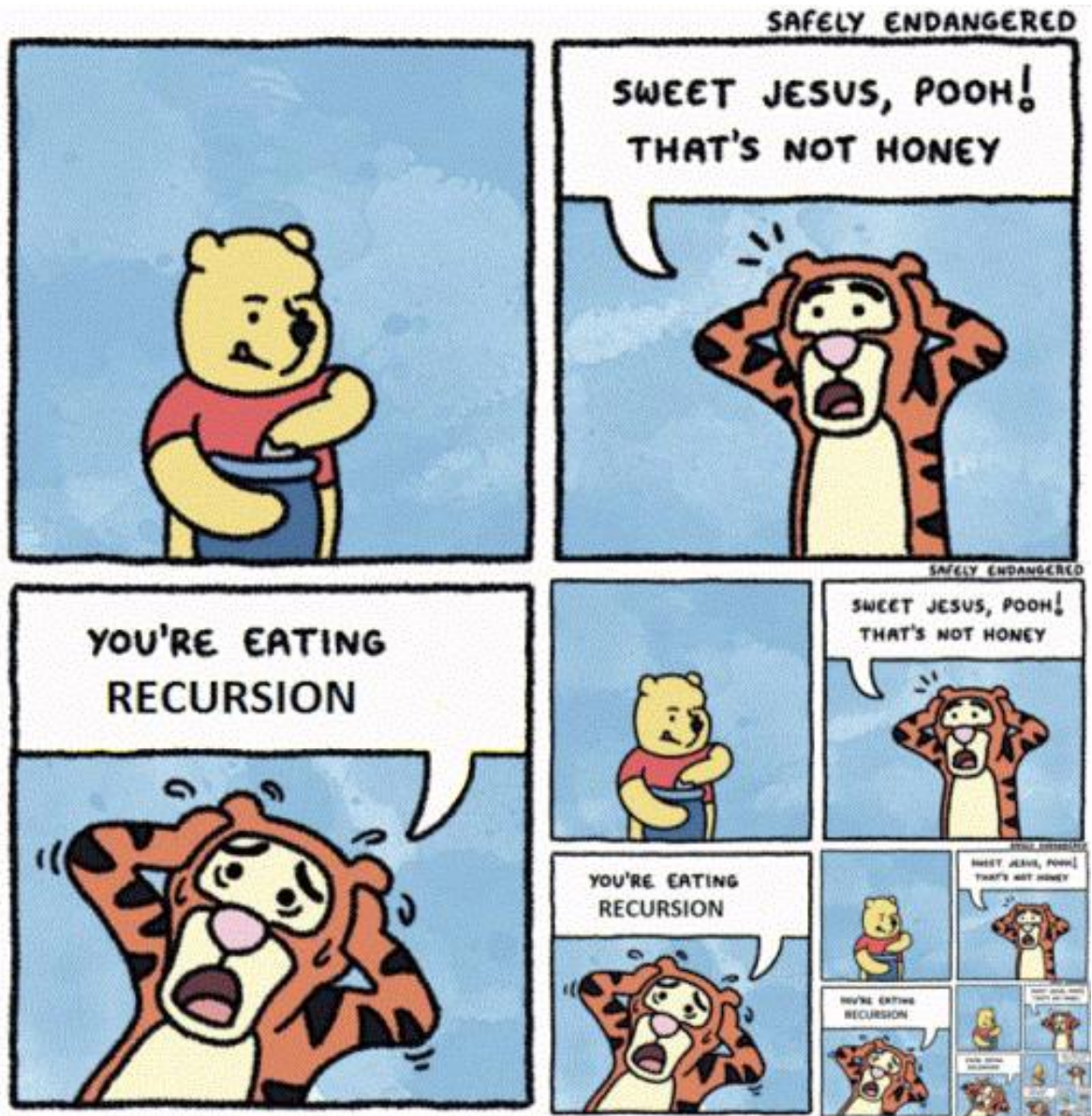


*"To understand recursion,  
we must first understand  
recursion."*  
— anônimo.




# Recursividade

*"Para fazer um  
procedimento recursivo  
é preciso ter fé. "*

*— prof. Siang Wun Song*



# Pesquisa por recursividade no Google

[All](#) [Images](#) [Videos](#) [Apps](#) [Maps](#) [More ▼](#) [Search tools](#)

---

About 7,740,000 results (0.28 seconds)

Did you mean: ***recursion***

# Introdução

Considere a **recorrência** (fórmula matemática) do fatorial:

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

A implementação de uma função que calcula fatorial com os conhecimentos que temos até agora poderia ser descrita assim:

```
public static int fatorial( int n ){  
    int fat = 1;  
    for( int termo = 1; termo <= n; termo++ ){  
        fat = fat * termo;  
    }  
    return fat;  
}
```

# Recursividade

A recursividade permite solucionar um problema a partir de uma **instância conhecida do problema**, e as soluções para as outras instâncias são encontradas a partir desta.

Um problema é dito **recursivo** se ele for definido em **termos de si próprio**, ou seja, recursivamente. Por exemplo a definição do fatorial.

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

# Recursividade

Dado que o problema é recursivo podemos escrever um algoritmo recursivo com a seguinte estratégia:

**se chegou instância conhecida do problema (base da recursão) então**

retorne o valor que sabemos calcular de forma direta.

**senão,**

Reduza a entrada do problema (parâmetros).

Faça a chamada da função novamente (chamada recursiva).

Use o retorno da função para calcular o resultado da solução completa



# Importância da base da recursão

Nunca se esqueça de escrever o caso base (**base da recursão**), caso contrário o resultado será uma recursão infinita





# Recursividade

A estrutura recursiva do cálculo do fatorial fica evidente. Assim, teremos a seguinte implementação recursiva.

```
public static int fatorial( int n ){  
    //base da recursão (condição de parada)  
    if( n == 0 )  
        return 1;  
    else  
        return n * fatorial(n-1) ;  
}
```

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

Uma implementação recursiva de uma se caracteriza por conter no corpo da função uma chamada para si mesma. A chamada de si mesma é dita **chamada recursiva**.

# Trabalhando com vetores e recursividade

Considere o algoritmo de busca linear iterativa visto nas aulas anteriores, a função implementada devolve a posição em que o elemento  $x$  se encontra no vetor, caso ele esteja lá, ou um valor inválido ( $-1$ ) que represente o insucesso da busca caso o elemento não esteja no vetor. Caso tenhamos elementos repetidos é retornado o índice da primeira ocorrência.

```
public static int buscaLinear(int v[], int x) {  
    for( int k=0; k < v.length; k++ ){  
        if( v[k] == x )  
            return k;  
    }  
    return -1;  
}
```

# Trabalhando com vetores e recursividade

Como implementar uma versão recursiva da função busca linear ??

A nova versão não poderá ter loops, ou seja, toda a avaliação dos elementos do vetor deverá ocorrer durante as chamadas recursivas da função. Para isso o índice  $k$  deverá ser passado por parâmetro.

# Trabalhando com vetores e recursividade

Como implementar uma versão recursiva da função busca linear ??

A nova versão não poderá ter loops, ou seja, toda a avaliação dos elementos do vetor deverá ocorrer durante as chamadas recursivas da função. Para isso o índice k deverá ser passado por parâmetro.

```
public static int buscaLinear(int v[], int x, int k) {  
    if ( k == v.length )  
        return -1;  
    if ( v[k] == x )  
        return k  
  
    return buscaLinear(v,x,k+1);  
}
```

# Recursividade

## Outros exemplos:

Cálculo do fatorial recursivo

<https://www.cs.usfca.edu/~galles/visualization/RecFact.html>

Invertendo uma palavra recursivamente

<https://www.cs.usfca.edu/~galles/visualization/RecReverse.html>

Problema das 8 rainhas:

<https://www.cs.usfca.edu/~galles/visualization/RecQueens.html>

Video da Torre de Hanoi

<https://www.youtube.com/watch?v=hLnuMXO95f8>

Como testar e entender um algoritmo recursivo ?

# Recursividade - Entendimento

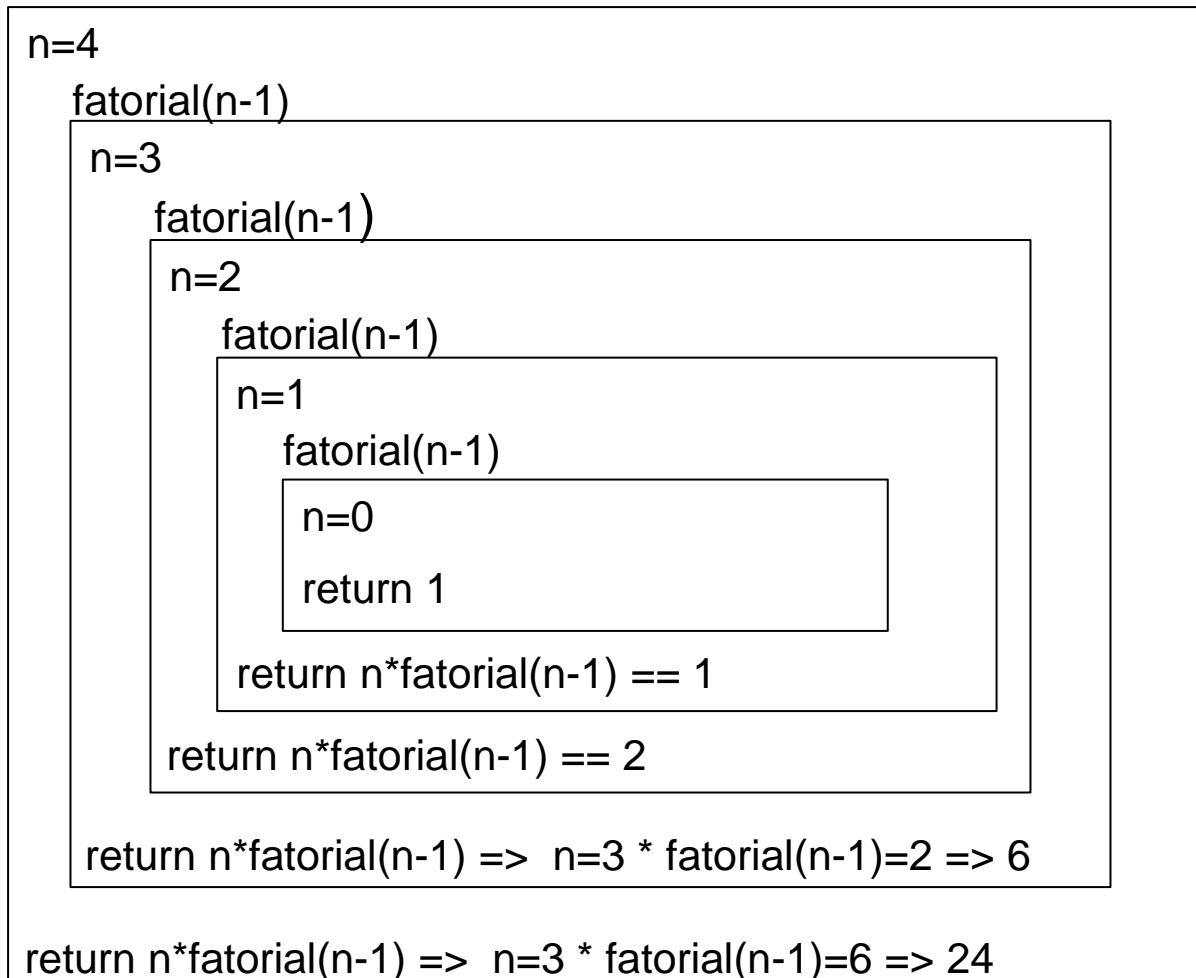
O **diagrama de execução** é um processo manual que é utilizado para validar e simular a lógica de uma **função recursiva**, e funciona da seguinte forma:

- Cada chamada recursiva da função é representada por um retângulo;
- Na parte superior (dentro) de cada retângulo são representadas as variáveis locais e parâmetros da função.
- Uma nova chamada recursiva resulta em um novo retângulo que estará encaixado no retângulo anterior (dentro), e assim sucessivamente.
- Na parte inferior é representado o retorno da função, ou seja, o momento que o valor calculado pela chamada é atribuída a uma variável do retângulo anterior.
- No diagrama de execução não são representadas as chamadas de outras funções iterativas (que não são recursivas).
- Convenções adicionais serão apresentadas nos exemplos que se seguem.

# Diagrama de execução

Diagrama de execução para `fatorial(4)`:

`fatorial(4)`





# Testar funções recursivas

[https://cscircles.cemc.uwaterloo.ca/java\\_visualize/#mode=display](https://cscircles.cemc.uwaterloo.ca/java_visualize/#mode=display)

# Profundidade da Recursão

O número de **chamadas recursivas** de uma função é denominado **Profundidade da Recursão**,

No caso anterior, esse **número é quatro**. Quando essa profundidade cresce muito, em geral há limitações de espaço de memória para guardar os valores intermediários.

Cada chamada recursiva irá empilhar dados na pilha de execução do programa (Stack) e, caso esta pilha não esteja dimensionada corretamente, podemos ter um “estouro de pilha”([stackoverflow](#)). Além disto, o processo de empilhamento pode ter um impacto significativo no desempenho do programa.

# Recursivo x Iterativo

## OBSERVAÇÕES IMPORTANTES:

1. Todo algoritmo **recursivo** tem uma versão **iterativa**, mas nem todo algoritmo **iterativo** tem uma versão **recursiva**
2. Para alguns problemas, se compararmos as implementações iterativa e recursiva, notaremos que que a **versão recursiva é mais simples que iterativa**.
3. A **versão iterativa** de uma algoritmo **é sempre mais rápida** que sua versão recursiva.
4. A complexidade (**nr passos**) de um algoritmo iterativo é a mesma que sua versão recursiva.

# Sequência de Fibonacci

A sequência [0, 1, 1, 2, 3, 5, 8, 13, 21, ...] é conhecida como sequência ou série de Fibonacci, e tem aplicações teóricas e práticas, na medida em que alguns padrões na natureza parecem segui-la. Pode ser obtida através da recorrência abaixo:

$$\text{Fib}(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ \text{Fib}(n - 1) + \text{Fib}(n - 2) & \text{se } n > 1 \end{cases}$$

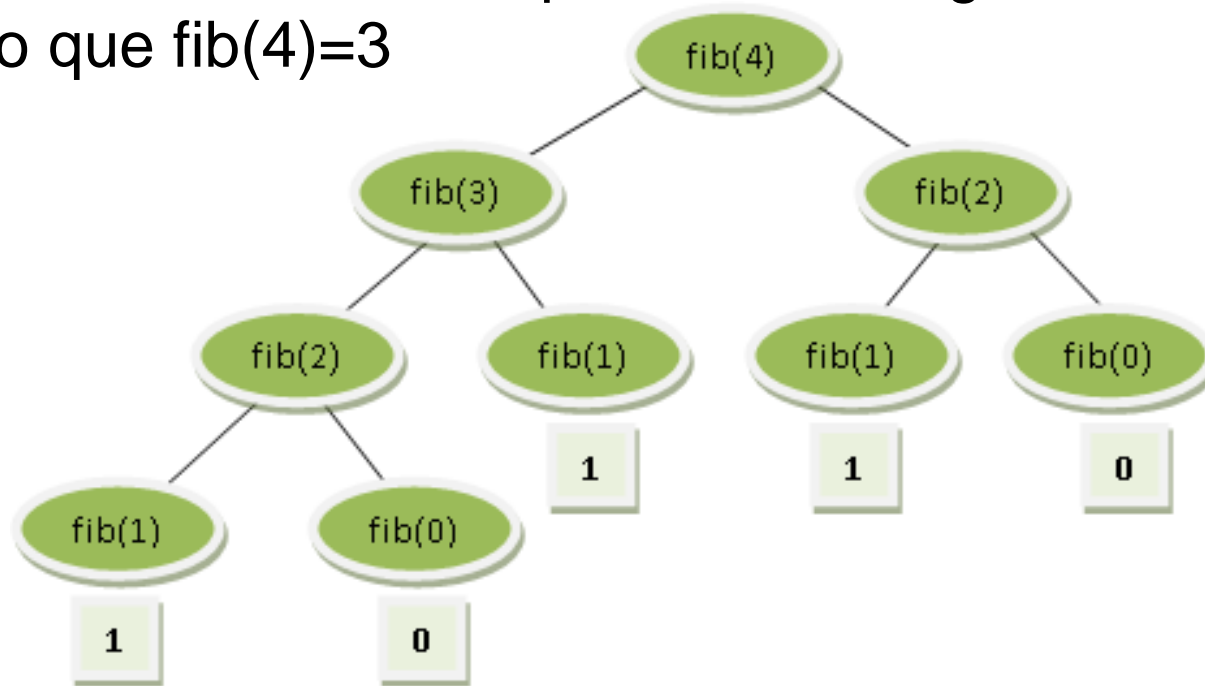
Como ficaria a implementação de uma função recursiva que calcula o número de Fibonacci nessa posição  $\text{Fib}(n)$ ?

Como ficaria o diagrama de execução para  $\text{Fib}(4)$  ?

# Fibonacci – profundidade da recursão

Para o Fibonacci de  $n=4$  foram feitas 8 chamadas recursivas, ou seja, o Fibonacci  $n=4$  tem profundidade igual a 8.

Lembrando que  $\text{fib}(4)=3$



Modifique o algoritmo para contar e retornar não mais o valor do número de Fibonacci na posição  $n$  e sim a profundidade da recursão para o número passado por parâmetro.

<https://www.urionlinejudge.com.br/judge/pt/problems/view/1029>

# Busca binária

Considere o algoritmo de busca binária iterativa:

```
public static int buscaBinaria( int v[], int x ){
    int i, m, f;
    i = 0; f = v.length-1;
    while (i <= f){
        m = (i + f)/2;
        if (v[m] == x)
            return m;
        if (x < v[m]) // esquerda
            f = m - 1;
        else // x > v[m] // direita
            i = m + 1;
    }
    return -1;
}
```

# Busca binária recursiva

Para implementarmos a busca binária recursiva será necessário *generalizar* ligeiramente o problema, trocando  $v[0..n-1]$  por  $v[i..f]$ . Assim teríamos que a função recebe um número  $x$  e um vetor em ordem crescente  $v[i..f]$ . Ele devolve um índice  $m$  tal que  $v[m] == x$  ou devolve  $-1$  se tal  $m$  não existe. A declaração da função ficaria:

```
public static int busca(int x, int v[], int i, int f ){  
    . . . . .  
}
```



# Exercícios

1) Acesse o site: <https://codingbat.com/java/Recursion-1> e tentem resolver os seguintes problemas:

Fatorial: <https://codingbat.com/prob/p154669>

Orelha dos coelhos: <https://codingbat.com/prob/p183649>

Fibonacci: <https://codingbat.com/prob/p120015>

Triângulo: <https://codingbat.com/prob/p194781>

Soma dígitos: <https://codingbat.com/prob/p163932>

Conta 7: <https://codingbat.com/prob/p101409>

.....

# Exercícios

2) Considere a seguinte função abaixo:

```
public static int result(int n){  
    if (n == 1)  
        return 2;  
    else  
        return 2 * result(n - 1);  
}
```

Faça o diagrama de execução para `result(5)`

3) Faça o diagrama de execução para `recursao(27)`

```
public static int recursao (int n){  
    if (n <= 10)  
        return n * 2;  
  
    else  
        return recursao(recursao(n/3)) ;  
}
```

# Exercícios

- 4) Qual a "**profundidade da recursão**" da função que realiza a busca binária recursiva ? Ou seja, quantas vezes `busca()` chama a si mesma?
- 5) Implemente uma função recursiva para calcular a potência  $a^n$ , supondo que tanto  $a$  quanto  $n$  sejam números inteiros positivos.
- 6) Dada um vetor de números inteiros positivos, descreva funções recursivas para :
  - a) Fazer a busca linear de um elemento no vetor;
  - b) Encontrar o menor elemento no vetor;
  - c) Fazer a soma dos elementos no vetor;
  - d) Calcular a média aritmética dos elementos no vetor;

# Exercícios

7) Faça o diagrama de execução para a função recursiva abaixo com  $N=7$ .

```
public static int fusc( int n )
{
    int aux;

    if( n <= 1 )
        return 1;
    if( n % 2 == 0 )
        return fusc( n/2 );

    return ( fusc((n-1)/2 ) + fusc((n+1)/2 ) );
}
```

# Exercícios

8) A função abaixo calcula o máximo divisor comum dos inteiros positivos m e n. Escreva uma função recursiva.

```
public static int mdc( int m, int n )
{
    int r;

    do{
        r = m % n;
        m = n;
        n = r;
    }while( r != 0 );

    return m;
}
```

# Exercícios

- 9) Escreva uma função recursiva que calcula o produto de  $a * b$ , em que  $a$  e  $b$  são inteiros maiores que zero. considere que o produto pode ser definido como  $a$  somado a si mesmo  $b$  vezes, usando uma definição recursiva temos

$$a * b = a \quad \text{se } b = 1$$

$$a * b = a * (b - 1) + a \quad \text{se } b > 1$$

- 10) Escreva uma função recursiva que gera todos as permutações (combinação simples sem repetição de valores) dígitos 1, 2, 3 .. n armazenados no vetor  $A[0.. n-1]$ . Para de  $n=3$  teríamos os seguintes dígitos 1, 2 e 3 armazenados no vetor  $A[]$  e como saída 6 permutações.

1= 1 2 3

2= 1 3 2

3= 2 1 3

4= 2 3 1

5= 3 1 2

6= 3 2 1

Como dica use um vetor auxiliar para gerar as permutações.

# Exercícios

11) A multiplicação Russa consiste em:

- Escrever os números A e B, que se deseja multiplicar na parte superior das colunas.
- Dividir A por 2, sucessivamente, ignorando o resto até chegar à unidade, escrever os resultados da coluna A.
- Multiplicar B por 2 tantas vezes quantas se haja dividido A por 2, escrever os resultados sucessivos na coluna B.
- Somar todos os números da coluna B que estejam ao lado de um número ímpar da coluna A.

Exemplo:  $27 \times 82 = \mathbf{2214}$

A	B	Parcelas
27	82	82
13	164	164
6	328	–
3	656	656
1	1312	1312

**Soma: 2214**

Programar em Java uma função recursiva que permita fazer à multiplicação russa de 2 entradas;



# Exercícios

- 12) A expressão abaixo converge para a raiz quadrada de A, sendo  $A > 0$ .  
Calcule um valor aproximado da raiz quadrada de um número dado A

$$x_n = \frac{1}{2} \left( x_{n-1} + \frac{A}{x_{n-1}} \right), \quad x_0 = 1, n \in N$$

Escreva uma função recursiva que calcule a raiz quadrada de um número usando o algoritmo acima, considere que sua função terá profundidade igual n, onde n é informado como um dos parâmetros da função

Fim