

Tecnologia em Sistemas para Internet



Algoritmos e Programação

Prof. Dr. Bruno Queiroz Pinto

Recursividade

UÉ, POR QUE VOCÊ ESTÁ AÍ HÁ UM TEMPO OLHANDO PRA PAREDE?

ESTOU AQUI PENSANDO EM COMO POSSO RESOLVER UM PROBLEMA DE RECURSIVIDADE

MUITO DIFÍCIL O PROBLEMA?

UM TANTO...

JOGUE O LIXO NO LIXO

Introdução

- Um algoritmo é recursivo quando ele chama a si mesmo ou chama uma sequência de outros algoritmos, e um deles chama novamente o primeiro algoritmo.

```
public static void funcaoRecursiva() {  
    //:  
    funcaoRecursiva();  
    //:  
}
```

```
public static void funcaoRecursiva() {  
    //:  
    funcao1();  
    //:  
}  
  
public static void funcao1() {  
    //:  
    funcao2();  
    //:  
}  
  
public static void funcao2() {  
    //:  
    funcaoRecursiva();  
    //:  
}
```

Geralmente substitui estruturas de repetição.

Introdução

○ Vantagens

- Redução do tamanho do código fonte.
- Poucas chamadas recursivas.
- Em alguns casos(?) é mais eficiente em ambiente com multiprocessadores.
- Vários problemas são naturalmente recursivos (modelados matematicamente através de funções recursivas). Geralmente geram algoritmos iterativos (estruturas de repetição) complexos.

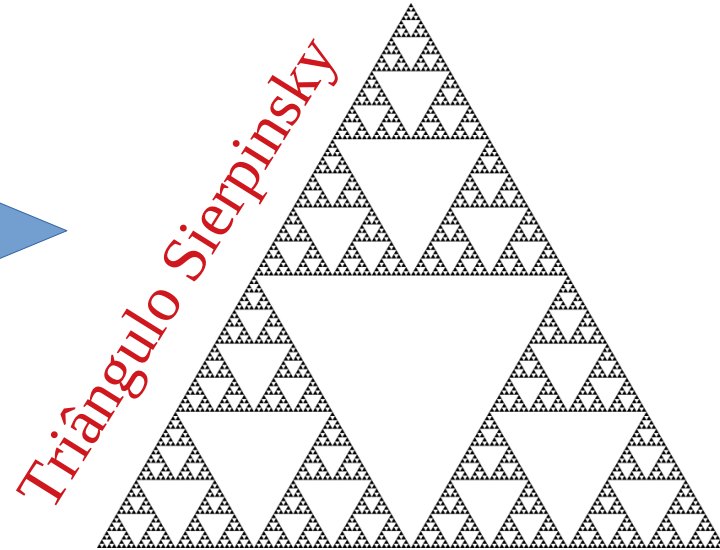
○ Desvantagens (Não em linguagens funcionais)

- Redução do desempenho de execução devido ao tempo para gerenciamento de chamadas.
- Dificuldades na depuração de programas recursivos, especialmente se a recursão for muito profunda.

Tema da aula

Recursividade

Código no Classroom



Um problema pode ser resolvido recursivamente quando ele pode ser dividido em 1 ou vários subproblemas com características similares

Introdução

- Uma função recursivo é uma função que chama a si mesmo, direta ou indiretamente.

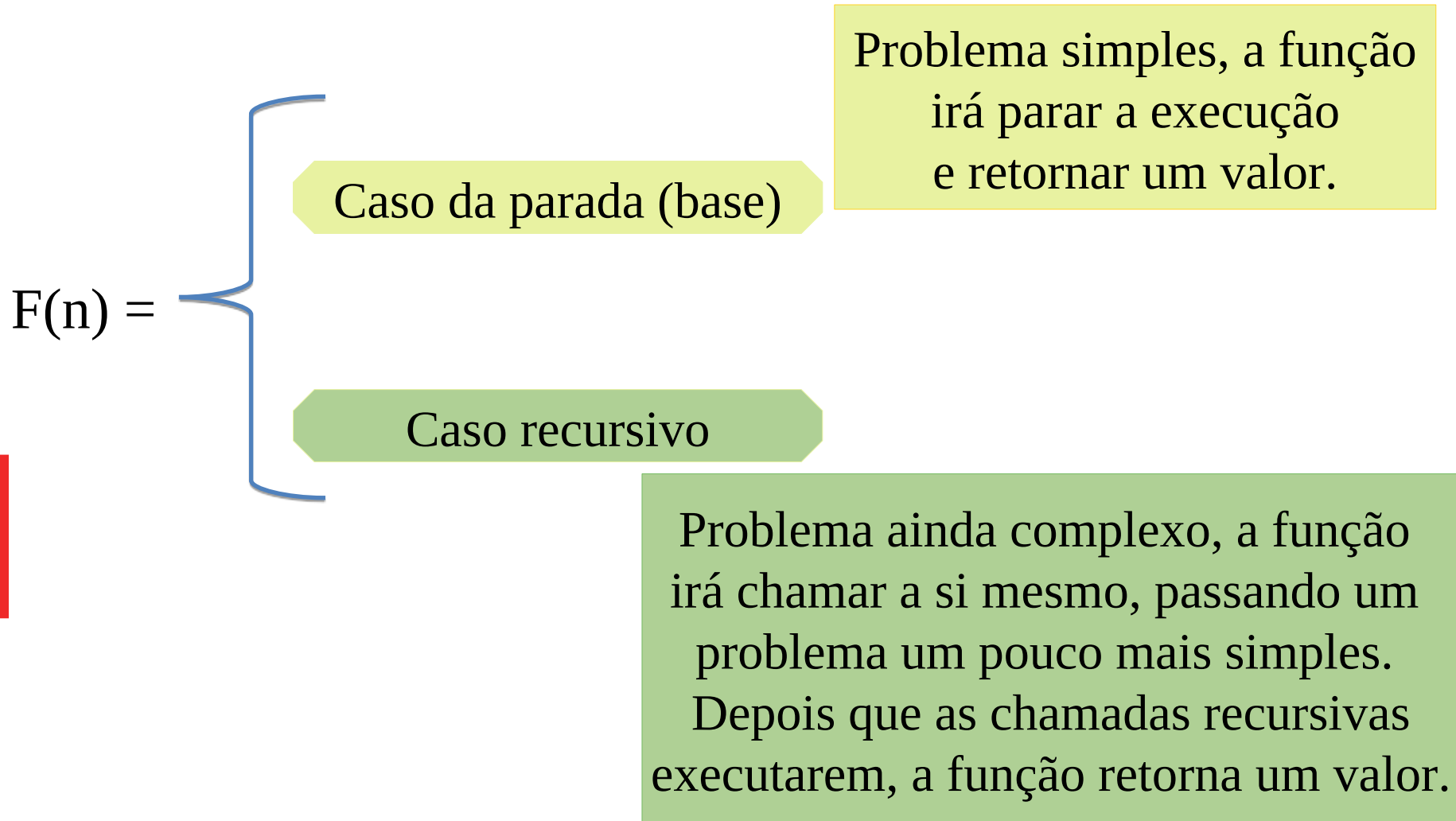
Sequência de Fibonacci

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

A sequência de Fibonacci consiste em uma série de números, tais que, definindo seus dois primeiros números como sendo 0 e 1, os números seguintes são obtidos através da soma dos seus dois antecessores. Exemplo da sequência: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

Função com retorno

Na matemática



Na matemática

Caso da parada(base)

$$F(n) = \begin{cases} C, \\ C' + F(?n?), \end{cases}$$

Caso recursivo

quando uma condição de parada for verdadeira
(1 ou mais casos da parada)

quando uma condição recursiva for verdadeira
(1 ou mais casos recursivos)

Na matemática

Sequência de Fibonacci

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

Implementação

○ Estrutura Básica

Sem retorno

Com retorno

Procedimento/funcao nome(parametros){
 if(condição){

Condição de parada (caso base)

 }
 else{

Condição de continuação (caso de recursão)
 Chamada de **nome(*parametros*)**

 }
}

Sequência de Fibonacci

Implementação

```
long fibonacci(int n){  
    if(condição){  
          
    }  
    else{  
          
    }  
}
```

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

$F(n) \rightarrow$ nomeado como fibonacci(int n)

Sequência de Fibonacci

Implementação

```
public static long fibonacci(int n) {  
    if (n==0) { return 0; }  
    if (n==1) { return 1; }  
    return fibonacci(n-1) + fibonacci(n-2);  
}
```

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

```
public static long fibonacci(int n) {  
    if (n<=1) {  
        return n;  
    } else {  
        return fibonacci(n-1) + fibonacci(n-2);  
    }  
}
```

Código não recursivo

```
public static long fibo(int n) {  
    int F = 0;      // atual  
    int ant = 0;    // anterior  
    for (int i = 1; i <= n; i++) {  
        if (i == 1) {  
            F = 1;  
            ant = 0;  
        } else {  
            F += ant;  
            ant = F - ant;  
        }  
    }  
    return F;  
}
```

Bem diferente

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

Função fatorial

- Definição de uma Função Fatorial

Na matemática o número seguido do símbolo de exclamação (!) é conhecido como fatorial, por exemplo, $x!$... O fatorial de um número natural n é a multiplicação de n pelos seus antecessores.

$$n! = \begin{cases} 1, & \text{se } n \leq 1, \\ n \times (n-1)!, & \text{caso contrário.} \end{cases}$$

Fatorial recursivo

$$\text{Fatorial}(N) = \text{Fatorial}(N-1) * N$$

b) Recursiva:

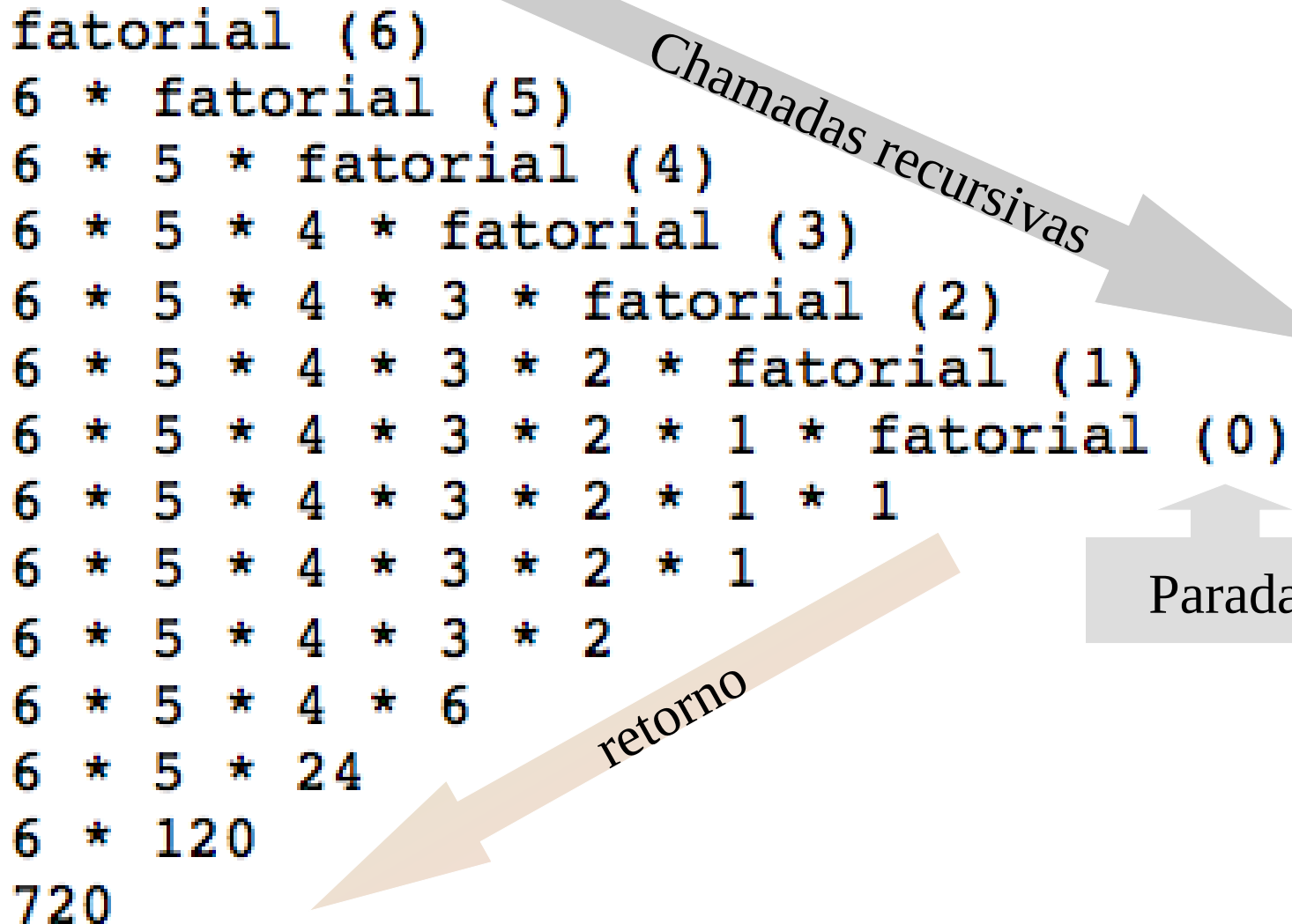
$$F(N) = \begin{cases} 1, & \text{para } N=0; \\ N * F(N-1), & \text{para } N \geq 1; \end{cases}$$

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

Caso base

Caso de
recursivo

Resultado da recursão - fatorial



```
fatorial (6)
6 * fatorial (5)
6 * 5 * fatorial (4)
6 * 5 * 4 * fatorial (3)
6 * 5 * 4 * 3 * fatorial (2)
6 * 5 * 4 * 3 * 2 * fatorial (1)
6 * 5 * 4 * 3 * 2 * 1 * fatorial (0)
6 * 5 * 4 * 3 * 2 * 1 * 1
6 * 5 * 4 * 3 * 2 * 1
6 * 5 * 4 * 3 * 2
6 * 5 * 4 * 3
6 * 5 * 4 * 6
6 * 5 * 24
6 * 120
720
```

Chamadas recursivas

Parada

retorno

Função Somatório


Podemos converter códigos não recursivos em recursivos.

```
public int somatorio(int N) {  
    int i, resp = 0;  
    for( i = 1; i <= N; i++ )  
        resp += i;  
    return resp;  
}
```

Refaça o exercício anterior, mas desta vez não utilize nenhum tipo de laço (for, while, do while, etc...).

Função Somatório recursiva

```
public int somatorio(int N) {  
    if( N == 1 )  
        return 1;  
    else  
        return (N + somatorio(N - 1));  
}
```




Função contar dígitos

//Quantidade de Dígitos

```
public int digitos(int N ) {  
    int cont = 1;  
    while( N >= 10) {  
        N = N / 10;  
        cont++;  
    }  
    return cont;  
}
```

Função contar dígitos recursivo

```
public int digitos(int N) {  
    if( N < 10 )  
        return 1;  
    else  
        return (1 + digitos(N / 10));  
}
```

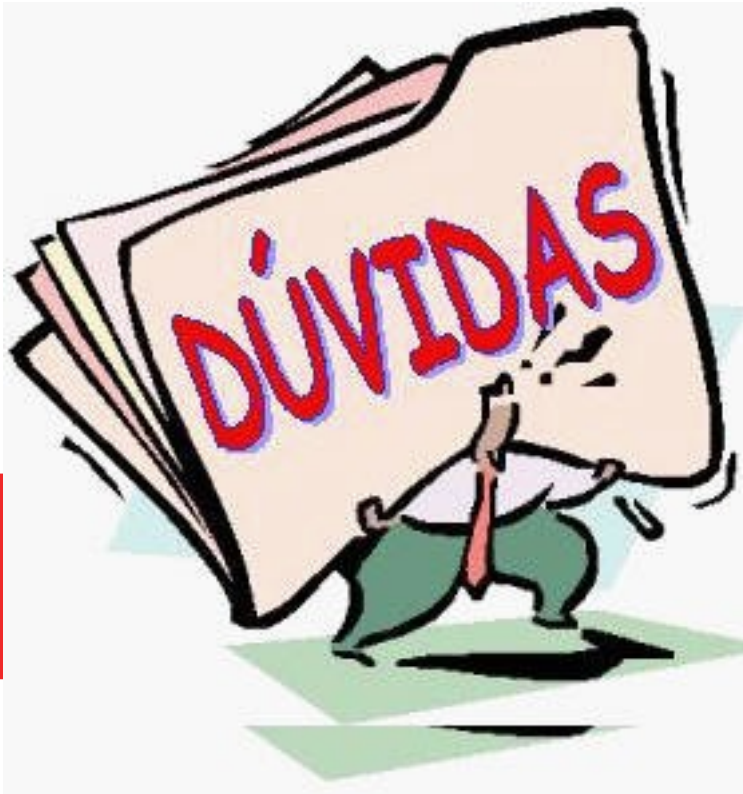


Função sem retorno

- Uma função recursiva sem retorno (void).
- Faça uma função que imprima os elementos do vetor.

```
public void imprimeVetor(int v[], int i) {  
    if( i == 0 )  
        System.out.print(v[i] + " ");  
    else{  
        System.out.print(v[i] + " ");  
        imprimeVetor(v, i-1);  
    }  
}
```

Fim da aula.....



"Para fazer um procedimento recursivo é preciso ter fé."—prof. Siang Wun Song, USP.

"Ao tentar resolver o problema, encontrei obstáculos dentro de obstáculos. Por isso, adotei uma solução recursiva." —aluno S.Y., USP, 1998

bruno.queiroz@iftm.edu.br