

Recursividade

**Professora:
Fátima L. S. Nunes**



Recursividade

- O que é recursividade?

Recursividade

- O que é recursividade?
 - *sf (recursivo+i+dade) Ling* Propriedade sintática pela qual um elemento pode aparecer um número infinito de vezes numa derivação, introduzido sempre pela mesma regra. *Ex: O filho da irmã da mãe de José.*

Recursividade - Wikipedia

Na matemática e na ciência da computação, a recursão específica (ou constrói) uma classe de objetos ou métodos (ou um objeto de uma certa classe) definindo alguns poucos casos base ou métodos muito simples (frequentemente apenas um), e então definindo regras para formular casos complexos em termos de casos mais simples.

Por exemplo, segue uma definição recursiva da ancestralidade de uma pessoa:

Os pais de uma pessoa são seus antepassados (caso base);

Os pais de qualquer antepassado são também antepassados da pessoa em consideração (passo recursivo).

Definições como esta são frequentemente encontradas na matemática, por exemplo, a definição formal dos números naturais diz que 0 (zero) é um número natural, e todo número natural tem um sucessor, que é também um número natural.

Recursividade

- Em computação:
 - ação na qual um procedimento (método, função, ...) chama a si mesmo.
 - geralmente seu uso permite descrição mais clara e concisa dos algoritmos, principalmente quando o problema considerado tem natureza recursiva.
 - **Exemplo:**
 - calcular fatorial de um número natural maior que 0

$$1! = 1$$

$$2! = 2$$

$$n! = 2 * 3 * 4 * \dots * (n-1) * n, n > 2$$

Para $n > 1$:

$$n! = (n-1)! * n$$

Recursividade

- **Exemplo:**
 - O cálculo do fatorial pode ser implementado de duas formas:
 - iterativo
 - recursivo

Recursividade

- Exemplo:
 - Algoritmo iterativo:

Recursividade

- Exemplo:
 - Algoritmo iterativo:

```
fatorial (n)
```

```
fat = 1
```

```
para i = 2 até n
```

```
    fat = fat * i
```

```
fim para
```

```
retorna fat
```

Recursividade

- Exemplo:
 - Algoritmo recursivo:

fatorial (n)

se $n < 3$

retorna n

senão

retorna $n * \text{fatorial}(n-1)$

fim se

Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $x = \text{fatorial}(5)$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna n * fatorial (4)
```

Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna n * fatorial (4)
```

```
n = 4
retorna n * fatorial (3)
```



Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna n * fatorial (4)
```

```
n = 4
retorna n * fatorial (3)
```

```
n = 3
retorna n * fatorial (2)
```

Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

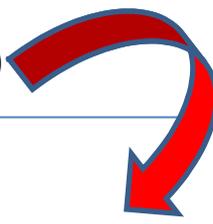
```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna n * fatorial (4)
```

```
n = 4
retorna n * fatorial (3)
```

```
n = 3
retorna n * fatorial (2)
```

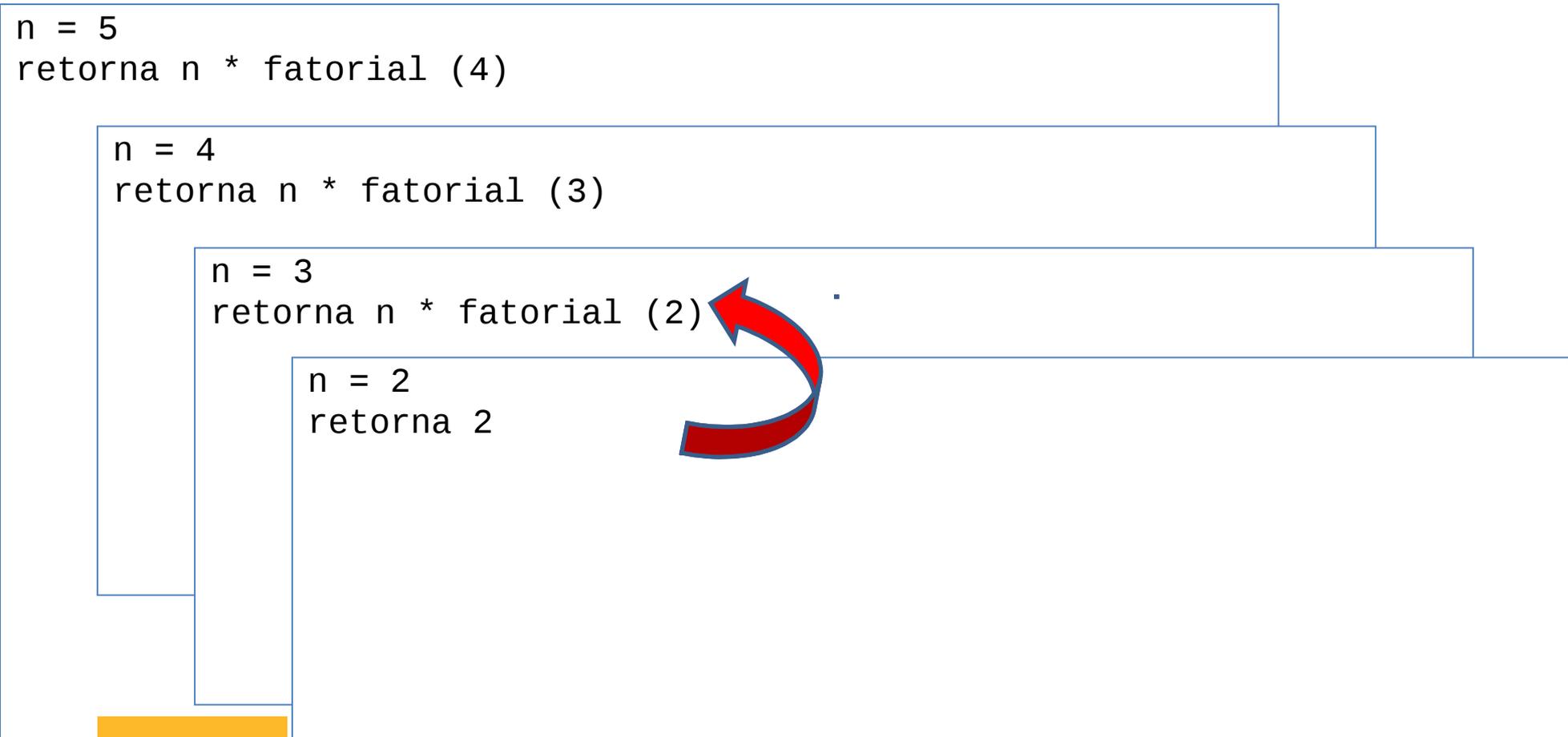
```
n = 2
retorna 2
```



Recursividade

- Exemplo:
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```



Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna n * fatorial (4)
```

```
n = 4
retorna n * fatorial (3)
```

```
n = 3
retorna n * 2 fatorial (2)
```



Recursividade

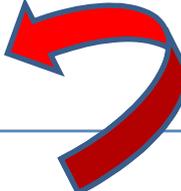
- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna n * fatorial (4)
```

```
n = 4
retorna n * fatorial (3)
```

```
n = 3
retorna 3 * 2 * fatorial (2)
```



Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
  se n < 3
    retorna n
  senão
    retorna n*fatorial (n-1)
  fim se
```

```
n = 5
retorna n * fatorial (4)
```

```
n = 4
retorna n * 6 fatorial (3)
```

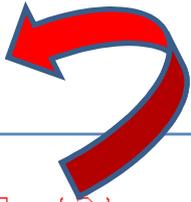
Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna n * fatorial (4)
```

```
n = 4
retorna 4 * 6 fatorial (3)
```



Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
  se n < 3
    retorna n
  senão
    retorna n*fatorial (n-1)
  fim se
```

```
n = 5
retorna n * 24 fatorial (4)
```

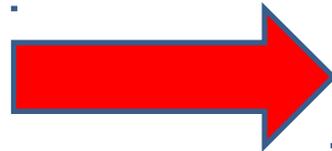
Recursividade

- **Exemplo:**
 - Algoritmo recursivo:
 - Simulação para $n = 5$

```
fatorial (n)
se n < 3
  retorna n
senão
  retorna n*fatorial (n-1)
fim se
```

```
n = 5
retorna 5 * fatorial (4)
```

24



RESULTADO = 120

Recursividade

- Implementação na linguagem C:

```
int fatorial (int n)
{
    if (n < 3)
        return n;
    else
        return n*fatorial (n-1);
}
```

Recursividade

- Como implementar recursividade:
 - Compilador usa uma *pilha*
 - estrutura de dados que armazena dados usados em cada chamada de um procedimento que ainda não terminou de processar;
 - o último dado a entrar é o primeiro a sair (LAST IN FIRST OUT);
 - o espaço de variáveis e parâmetros alocado para um método implementado em uma determinada linguagem é chamado de *registro de ativação*;
 - o *registro de ativação* é desalocado quando termina um método.
 - mais detalhes de pilha serão vistos na disciplina Estruturas de dados
 - Deve ser considerado o problema de **terminação**
 - no algoritmo de deve existir uma condição que encerre o processo de empilhamento e comece desempilhar os dados armazenados

Recursividade

```
int soma(int n) {  
    if ( n > 0 )  
        n += soma(n-1);  
    return n;  
}
```

Soma dos n inteiros não negativos

Recursividade

```
int soma(int n) {  
    if ( n > 0 )  
        n += soma(n-1);  
    return n;  
}
```

```
int soma(int n) {  
    int soma = 0;  
    while ( n > 0 )  
        soma += n--;  
    return soma;  
}
```

Recursividade

- **Exemplo:**
 - Números de Fibonacci, definidos pela seguinte equação de recorrência:

$$\left\{ \begin{array}{l} f_0 = 0, f_1 = 1, \\ f_n = f_{n-1} + f_{n-2}, n \geq 2 \end{array} \right.$$

Recursividade

- Exemplo:
 - Implemente um procedimento recursivo para calcular a sequência de Fibonacci, dado n

$$\left\{ \begin{array}{l} f_0 = 0, f_1 = 1, \\ f_n = f_{n-1} + f_{n-2}, n \geq 2 \end{array} \right\}$$

Recursividade

- Exemplo:
 - Implemente um procedimento recursivo para calcular a sequência de Fibonacci, dado n

$$\left\{ \begin{array}{l} f_0 = 0, f_1 = 1, \\ f_n = f_{n-1} + f_{n-2}, n \geq 2 \end{array} \right\}$$

fibonacci(n)

Recursividade

- Exemplo:
 - Implemente um procedimento recursivo para calcular a sequência de Fibonacci, dado n

$$\left\{ \begin{array}{l} f_0 = 0, f_1 = 1, \\ f_n = f_{n-1} + f_{n-2}, n \geq 2 \end{array} \right\}$$

fibonacci(n)

se $n < 2$

retorna n

senão

retorna $\text{fibonacci}(n-1) + \text{fibonacci}(n-2)$

fim se

Recursividade

- Exemplo:
 - Qual é o problema?

```
fibonacci(n)  
se  $n < 2$   
  retorna  $n$   
senão  
  retorna  $\text{fibonacci}(n-1) + \text{fibonacci}(n-2)$   
fim se
```

Recursividade

- Exemplo:
 - Qual é o problema?

```
fibonacci(n)  
se  $n < 2$   
  retorna  $n$   
senão  
  retorna  $\text{fibonacci}(n-1) + \text{fibonacci}(n-2)$   
fim se
```

- Para cada iteração, chama o procedimento recursivamente 2 vezes → **aumenta absurdamente a complexidade do algoritmo**

Recursividade versus Iteração

- Soluções recursivas são geralmente mais concisas que as iterativas, gerando programas menores e mais simples.
- Soluções iterativas em geral usam espaço definido de memória, enquanto soluções recursivas solicitam memória à medida que precisam.
- Cópia dos parâmetros a cada chamada recursiva é um custo adicional para as soluções recursivas.
- Programas recursivos que possuem chamadas no final do código são ditos terem recursividade de cauda. São facilmente transformáveis em uma versão não recursiva (exemplo números de Fibonacci)
- Projetista de algoritmos deve levar em consideração a complexidade (temporal e espacial), bem como os outros custos (e.g., facilidade de manutenção) para decidir por qual solução utilizar.

Indução Matemática

- Algoritmos recursivos podem ser definidos e estudados a partir da indução matemática.
- Seja T um teorema a ser provado
 - Consideremos T como tendo um número natural como parâmetro (n)
 - Em vez de tentar provar que T é válido para todos valores de n , basta provar duas condições:
 1. T é válido para $n = 1$;
 2. Para todo $n > 1$:
 - se T é válido para $n-1 \Rightarrow T$ é válido para n

Indução Matemática

- Exemplo: fatorial
- Para facilitar a indução, modificaremos um pouco o método usado anteriormente para calcular o fatorial, para que retorne uma constante, facilitando a definição do passo base:

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

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

- Qual é o passo base?

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

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

- Qual é o passo base?

Se $n=0$, então o fatorial = 1

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

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

- Qual é o passo base?
Se $n=0$, então o fatorial = 1
- Qual é o passo indutivo?

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

Recursividade e indução

- Definindo o cálculo do fatorial usando indução:

- Qual é o passo base?

Se $n=0$, então o fatorial = 1

- Qual é o passo indutivo?

- Devemos expressar a solução para $n > 0$, supondo que já sabemos a solução para algum caso mais simples (**$n-1$** , por exemplo)

$$n! = n * (n-1)!$$

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

Exercícios

1. Forneça uma solução recursiva para o problema de busca binária .
2. Como pode ser calculada a complexidade (temporal ou espacial) de um algoritmo recursivo?
3. Escreva um método recursivo que calcule a soma dos elementos positivos do vetor de inteiros $v[0..n-1]$. O problema faz sentido quando n é igual a 0? Quanto deve valer a soma nesse caso?
4. Escreva um método recursivo *maxmin* que calcule o valor de um elemento máximo e o valor de um elemento mínimo de um vetor $v[0..n-1]$. Quantas comparações envolvendo os elementos do vetor a sua função faz?
5. Escreva um método recursivo que calcule a soma dos elementos positivos do vetor $v[ini..fim-1]$. O problema faz sentido quando *ini* é igual a *fim*? Quanto deve valer a soma nesse caso?
6. Escreva um método recursivo que calcule a soma dos dígitos de um inteiro positivo n . A soma dos dígitos de 132, por exemplo, é 6.
7. Escreva um método recursivo *onde()*. Ao receber um inteiro x , um vetor v e um inteiro n , o método deve devolver j no intervalo fechado $0...n-1$ tal que $v[j] == x$. Se tal j não existe, o método deve devolver -1.
8. Escreva um método recursivo que recebe um inteiro x , um vetor v e inteiros *ini* e *fim* e devolve j tal que $ini \leq j \leq fim-1$ e $v[j] == x$. Se tal j não existe então devolve *ini-1*.

Alguns métodos recursivos (C)

Busca sequencial recursiva

```
int sequencial(int valor, int[] vetor, int n) {
    if(n == 1) {
        if(vetor[0] == valor) {
            return 0;
        }
        else return -1;
    }
    else {
        int index = sequencial(valor, vetor, n-1);
        if(index < 0) {
            if(vetor[n-1] == valor) {
                index = n-1;
            }
        }
        return index;
    }
}
```

Alguns métodos recursivos (C)

■ Busca Binária

Na busca binária, supõe-se que os elementos do vetor estão ordenados. A pesquisa inicia-se no elemento central. Se o elemento procurado for encontrado, o algoritmo termina. Se o elemento procurado for maior (menor) do que o elemento central a busca é repetida considerando-se apenas a metade superior (inferior) do vetor.

■ O processo continua até que o valor seja encontrado ou até que o tamanho do vetor a ser procurado seja 0.

Alguns métodos recursivos (C)

Busca Binária iterativa

```
int binaria(int valor, int vetor[], int n) {
    int esq, dir, meio;
    esq = 0; dir = n-1;
    while (esq <= dir) {
        meio = (esq + dir) / 2;
        if (vetor[meio] == valor) return meio;
        if (vetor[meio] < valor)
            esq = meio + 1;
        else
            dir = meio - 1;
    }
    return -1;
}
```


Alguns métodos recursivos (C)

Busca Binária recursiva

```
int binaria(int valor, int vetor[], int esq, int dir) {
    int meio = (esq + dir)/2;
    if (esq > dir ) return -1; /// não achou valor
    if(valor > vetor[meio]) {
        esq = meio + 1;
        return binaria(valor, vetor, esq, dir);
    }
    else
    if(valor < vetor[meio]) {
        dir = meio - 1;
        return binaria(valor, vetor, esq, dir);
    }
    else return meio;
}
```

Alguns métodos recursivos (C)

Torres de Hanói

- O jogo **Torres de Hanói** é um quebra-cabeça que consiste em um conjunto de N discos de tamanhos diferentes e 3 pinos verticais, nos quais os discos devem ser encaixados.
- Cada pino pode conter uma pilha com qualquer quantidade de discos, desde que cada disco não seja colocado sobre outro disco menor.
- Configuração inicial: todos os discos no pino 1.
- Objetivo: mover todos os discos para um dos outros pinos, sempre obedecendo a restrição de não colocar um disco sobre outro menor.



Alguns métodos recursivos (C)

Torres de Hanói

```
hanoi(A, B, C, n) {  
  se n = 1  
    Move de A para B  
  }  
  else {  
    hanoi(A, C, B, n-1)  
    Move de A para B  
    hanoi(C, B, A, n-1);  
  }  
}
```



Alguns métodos recursivos (C)

Torres de Hanói

```
void hanoi(char ori, char dst, char aux, int nro) {  
    if(nro == 1) {  
        printf("Move de %c para %c\n",ori, dst);  
    }  
    else {  
        hanoi(ori, aux, dst, nro-1);  
        printf("Move de %c para %c\n",ori, dst);  
        hanoi(aux, dst, ori, nro-1);  
    }  
}
```



Alguns métodos recursivos (C)

- Torres de Hanói
- <https://www.youtube.com/watch?v=0qMPHqrPdE0>



Referências

- C. Camarão & L. Figueiredo. Programação de Computadores em Java. Livros Técnicos e Científicos Editora, 2003.
- Nívio Ziviani. Projeto de Algoritmos com implementações em C e Pascal. Editora Thomson, 2a. Edição, 2004.
- Notas de aula – Prof. Delano Beder – EACH-USP

Recursividade

Professora:
Fátima L. S. Nunes