Recursão
Algoritmos e Estrutura de Dados I

Instituto de Engenharia – UFMT

Roteiro

Objetivos

Esta aula tem como objetivos:

1. Apresentar processos recursivos;
2. Elucidar os prós e contras da implementação recursiva;
3. Explicitar quando é viável utilizar essa técnica, em termos de desempenho;
4. Exemplificar a implementação de algoritmos recursivos.

Introdução

Função recursiva

Segundo **??**), uma função é dita recursiva quando é definida em termos dela mesma.

- A recursão está relacionada à indução matemática, sendo formada por um (ou mais) caso base e um passo recursivo;
- O uso da recursão geralmente permite uma descrição mais clara e concisa dos algoritmos;

Introdução

- No entanto, mesmo a recursão sendo a forma mais prática de implementar algumas soluções, não é a técnica mais eficiente;
- É importante saber quando utilizar a recursão e quando não utilizar;
- Para entender a técnica, vamos começar por um exemplo simples.

Exemplo: Fatorial

Considere o fatorial de um inteiro positivo n, em que $n!$ é dado pela seguinte fórmula:

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

- Essa definição estabelece um processo recursivo para calcular o fatorial de um inteiro n.
- O caso trivial ocorre quando $n = 0$;
- Assim, usando-se esse processo recursivo, o cálculo de 5!, por exemplo, é feito como a seguir:

Exemplo: Fatorial

Considere o fatorial de um inteiro positivo n, em que $n!$ é dado pela seguinte fórmula:

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

- Essa definição estabelece um processo recursivo para calcular o fatorial de um inteiro n.
- O caso trivial ocorre quando $n = 0$;
- Assim, usando-se esse processo recursivo, o cálculo de 5!, por exemplo, é feito como a seguir:

Exemplo: Fatorial

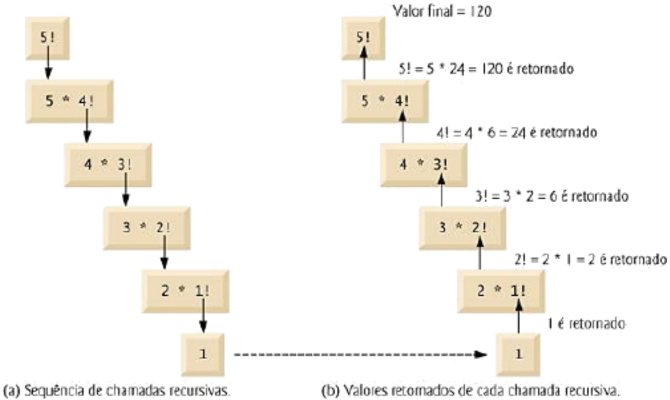


Figura: Adaptada de **??**).

Exemplo: Fatorial

Algoritmo 1: fatorial

Entrada: n
Saída: $n!$

```
início
    se (n = 0) então
        retorna 1
    senão
        retorna n × fatorial(n − 1)
```

Exemplo: Fatorial

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

Exemplo: Fatorial

Calcula-se o fatorial de 5 da seguinte forma:

$$
\begin{aligned}
fatorial(5) &= 5 \times fatorial(5-1) \\
&= 5 \times fatorial(4) \\
&= 5 \times 4 \times fatorial(4-1) \\
&= 5 \times 4 \times fatorial(3) \\
&= 5 \times 4 \times 3 \times fatorial(3-1) \\
&= 5 \times 4 \times 3 \times fatorial(2) \\
&= 5 \times 4 \times 3 \times 2 \times fatorial(2-1) \\
&= 5 \times 4 \times 3 \times 2 \times fatorial(1) \\
&= 5 \times 4 \times 3 \times 2 \times 1 \times fatorial(1-1) \\
&= 5 \times 4 \times 3 \times 2 \times 1 \times fatorial(0) \\
&= 5 \times 4 \times 3 \times 2 \times 1 \times 1 \\
&= 120
\end{aligned}
$$

Exemplo: Somatória

Implemente um algoritmo que calcule o valor da soma dos números de 1 a n de forma recursiva.

Exemplo: Somatória

```
int somatoria(int n) {
  if (n == 0)
    return 0;
  else
    return n + somatoria(n-1);
}
```

Exemplo: Somatória

Calcula-se a soma de 1 até 3, ou seja $somatoria(3)$, da seguinte forma

$$\begin{aligned} somatorio(3) &= 3 + somatorio(3-1) \\ &= 3 + somatorio(2) \\ &= 3 + 2 + somatorio(2-1) \\ &= 3 + 2 + 1 + somatorio(1-1) \\ &= 3 + 2 + 1 + 0 \\ &= 6. \end{aligned}$$

Exemplo: Multiplicação

Implemente um algoritmo que calcule o valor da multiplicação de dois números, x e y, de forma recursiva, utilizando apenas soma.

Exemplo: Multiplicação

```
int multiplicacao(int x, int y) {
  if (y == 0)
    return 0;
  else
    return x + multiplicacao(x, y-1);

}
```

Exemplo: Multiplicação

Calcula-se a multiplicação 4×3 conforme a equação a seguir.

$$\begin{aligned}
multiplicacao(4,3) &= 4 + multiplicacao(4, 3-1)\\
&= 4 + multiplicacao(4,2)\\
&= 4 + 4 + multiplicacao(4, 2-1)\\
&= 4 + 4 + multiplicacao(4,1)\\
&= 4 + 4 + 4 + multiplicacao(4, 1-1)\\
&= 4 + 4 + 4 + multiplicacao(4,0)\\
&= 4 + 4 + 4 + 0\\
&= 12
\end{aligned}$$

Exemplo: Potência

Implemente um algoritmo que calcule o valor da potência de x elevado a y, de forma recursiva.

Exemplo: Potência

```
int potencia(int x, int y) {
  if (y == 0)
    return 1;
  if (y == 1)
    return x;
  else
    return x*potencia(x,y-1);
}
```

Exemplo: Potência

Calcula-se a potência de 4 elevado a 3 conforme a equação a seguir.

$$\begin{aligned}
potencia(4,3) &= 4 \times potencia(4, 3-1) \\
&= 4 \times potencia(4,2) \\
&= 4 \times 4 \times potencia(4, 2-1) \\
&= 4 \times 4 \times potencia(4,1) \\
&= 4 \times 4 \times 4 \\
&= 64
\end{aligned}$$

Exemplo: Conta dígitos

Implemente um algoritmo recursivo que dado um número, n, informe quantos dígitos n possui.

Exemplo: Conta dígitos

```
int conta_digitos(int n) {
  if (n < 10)
    return 1;
  else
    return 1 + conta_digitos(n/10);
}
```

Exemplo: Conta dígitos

O cálculo da quantidade de dígitos do número 9876 pode ser verificado na equação abaixo.

$$
\begin{aligned}
conta\_digitos(9876) &= 1 + conta\_digitos(\frac{9876}{10}) \\
&= 1 + conta\_digitos(987) \\
&= 1 + (1 + conta\_digitos(\frac{987}{10})) \\
&= 1 + (1 + conta\_digitos(98)) \\
&= 1 + (1 + (1 + conta\_digitos(\frac{98}{10}))) \\
&= 1 + (1 + (1 + conta\_digitos(9))) \\
&= 1 + (1 + (1 + 1)) \\
&= 4
\end{aligned}
$$

Exemplo: Divisão

Implemente um algoritmo recursivo que dados dois números, x e y, calcule a divisão inteira de x por y.

Exemplo: Divisão

```
int divisao(int x, int y) {
  if (x < y)
    return 0;
  else
    return 1 + divisao(x-y,y);
}
```

Exemplo: Divisão

Calcula-se a divisão inteira $\frac{15}{4}$ conforme a equação a seguir.

$$\begin{aligned}
divisao(15,4) &= 1 + divisao(15-4,4)\\
&= 1 + divisao(11,4)\\
&= 1 + 1 + divisao(11-4,4)\\
&= 1 + 1 + divisao(7,4)\\
&= 1 + 1 + divisao(7-4,4)\\
&= 1 + 1 + 1 + divisao(3,4)\\
&= 1 + 1 + 1 + 0\\
&= 3
\end{aligned}$$

Exemplo: Resto da divisão (módulo)

Implemente um algoritmo recursivo que dados dois números, x e y, calcule o resto da divisão inteira de x por y.

Exemplo: Resto da divisão(módulo)

```
int resto(int x, int y) {
  if (x < y)
    return x;
  else
    return resto(x-y,y);
}
```

Exemplo: Resto da divisão

Calcula-se o resto da divisão $\frac{15}{4}$, conforme a equação abaixo.

$$\begin{aligned}
resto(15,4) &= resto(15-4,4) \\
&= resto(11,4) \\
&= resto(11-4,4) \\
&= resto(7,4) \\
&= resto(7-4,4) \\
&= resto(3,4) \\
&= 3
\end{aligned}$$

Exemplo: Raiz quadrada

Implemente um algoritmo recursivo que dado um número n, calcule a raiz quadrada inteira de n.

Exemplo: Raiz quadrada

```c
int raiz_quadrada(int n, int i) {
  if (i*i<=n)
    return raiz_quadrada(n, i+1);
  else
    return i-1;
}
```

Exemplo: Raiz quadrada

O cálculo da raiz quadrada inteira pode ser verificado na equação abaixo.

$$
\begin{aligned}
raiz\_quadrada(16,0) &= raiz\_quadrada(16,1) \\
&= raiz\_quadrada(16,2) \\
&= raiz\_quadrada(16,3) \\
&= raiz\_quadrada(16,4) \\
&= raiz\_quadrada(16,5) \\
&= 4
\end{aligned}
$$

Exemplo: Raiz n-ésima

Implemente um algoritmo recursivo que dados dois números inteiros, denominados m e n, calcule a raiz n-ésima inteira de m. Para isso, utilize a função recursiva para cálculo da potência.

Exemplo: Fibonacci

O número de Fibonacci F_n para $n \geq 0$ é definido da seguinte maneira:

$$F_n = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F_{n-1} + F_{n-2} & \text{se } n > 1 \end{cases}$$

- Os casos bases ocorrem quando $n = 0$ e $n = 1$;
- Utilizando o processo recursivo, o cálculo de F_3, por exemplo, é feito como a seguir:

Exemplo: Fibonacci

O número de Fibonacci F_n para $n \geq 0$ é definido da seguinte maneira:

$$F_n = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F_{n-1} + F_{n-2} & \text{se } n > 1 \end{cases}$$

- Os casos bases ocorrem quando $n = 0$ e $n = 1$;
- Utilizando o processo recursivo, o cálculo de F_3, por exemplo, é feito como a seguir:

Exemplo: Fibonacci

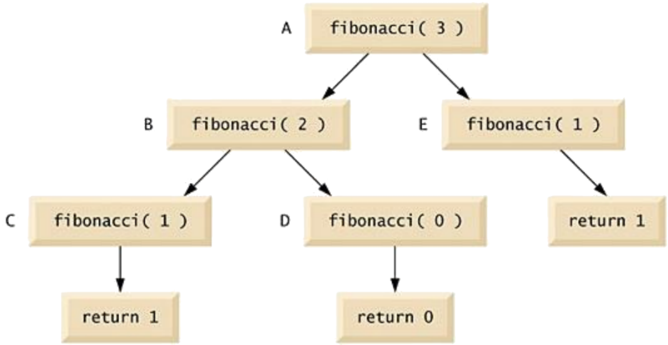


Figura: Chamadas de método feitas dentro da chamada fibonacci(3). Adaptada de **??**).

Exemplo: Fibonacci

Algoritmo 2: fibonacci

Entrada: n
Saída: número de Fibonacci F_n

```
início
    se (n = 0) então
        retorna 0
    se (n = 1) então
        retorna 1
    senão
        retorna fibonacci(n − 1) + fibonacci(n − 2)
```

Exemplo: Fibonacci

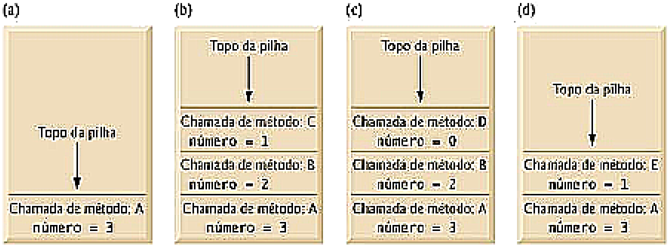


Figura: Chamadas de método na pilha de execução do programa. Adaptada de **??**).

Exemplo: Fibonacci

```c
int fibonacci(int n) {
  if (n == 0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Recursão *versus* Iteração

Semelhanças

- Tanto iteração quanto recursão se baseiam em um estrutura de controle:
  - a iteração utiliza estrutura de repetição e recursão utiliza instruções de seleção.
- Ambas envolvem repetição:
  - a iteração utiliza explicitamente estruturas de repetição e a recursão alcança a repetição por meio de chamadas repetidas de método.
- Ambas envolvem um teste de terminação:
  - a iteração quando a condição de repetição falha e a recursão quando o caso base é alcançado.
- Tanto uma quanto a outra podem ocorrer infinitamente.

Recursão *versus* Iteração

Soluções iterativas

Para ilustrar as diferenças, vamos examinar soluções iterativas para os problemas apresentados.

Exemplo iterativo: Fatorial

Algoritmo 3: fatorial-iterativo

Entrada: n

Saída: $n!$

início

  $valor \leftarrow 1$

  para $i \leftarrow n$ $até$ 1 **faça**

    $valor \leftarrow valor \times i$

  retorna $valor$

Exemplo iterativo: Fibonacci

Algoritmo 4: fibonacci-iterativo

Entrada: n

Saída: número de Fibonacci F_n

início

$\quad f[0] \leftarrow 0$

$\quad f[1] \leftarrow 1$

$\quad$ **para** $i \leftarrow 2$ *até* n **faça**

$\qquad f[i] \leftarrow f[i-1] + f[i-2]$

$\quad$ **retorna** $f[i]$

Recursão *versus* Iteração

A recursão tem muitas negativas:

- Invoca repetidamente o mecanismo, e consequentemente o *overhead*, das chamadas de método;
- Essa repetição pode ser cara tanto em termos de processador como de espaço de memória;
- Cada chamada recursiva faz com que outra cópia do método seja criada;
- Esse conjunto de cópias pode consumir espaço considerável de memória;
- Como a iteração ocorre dentro de um único método, as chamadas de métodos repetidas e a atribuição extra de memória são evitadas.

Recursão *versus* Iteração

Vantagem

Então por que escolher recursão?

Exemplo

Vamos verificar o porquê através de um exemplo.

Recursão *versus* Iteração

Vantagem

Então por que escolher recursão?

Exemplo

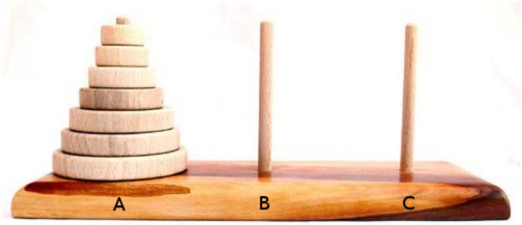
Vamos verificar o porquê através de um exemplo.

Exemplo: Torres de Hanói

Neste problema, deve-se mover uma pilha de discos de um pino para outro. Ao mover a pilha de um pino para outro deve-se ficar atento às restrições:

- um único disco é movido por vez;
- em nenhuma circunstância um disco maior pode ser colocado em cima de um disco menor.

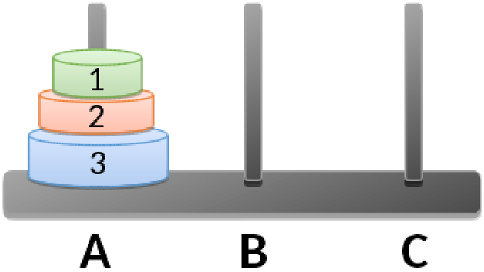
Três pinos são fornecidos e um deles é utilizado para armazenar discos temporariamente.



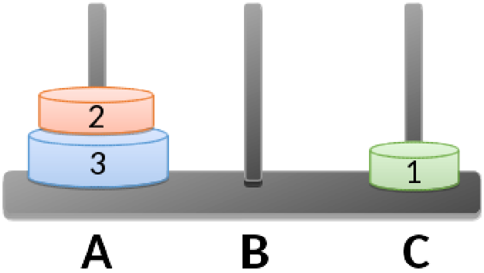
Exemplo: Torres de Hanói

- Vamos desenvolver um algoritmo que, dada uma entrada que indica a quantidade de discos, informa a sequência de movimentos e quantos são necessários para mover os discos do pino A para o pino C, utilizando o pino B para armazenamento temporário.
- Se o valor fornecido para o programa for $n = 3$, então a sequência de chamadas e as saídas geradas são:

Exemplo: Torres de Hanói

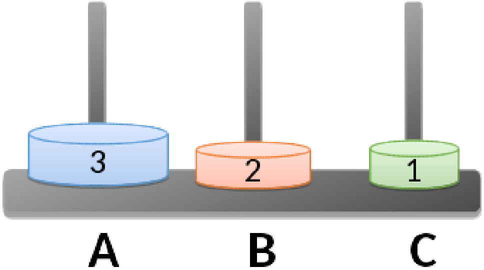


Exemplo: Torres de Hanói



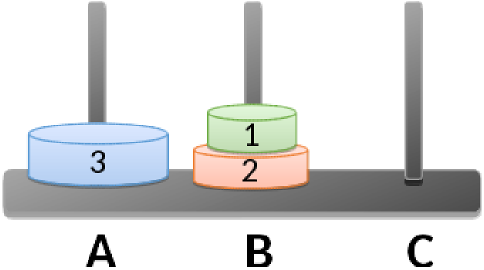
Movimento 1: disco 1 do pino A para o pino B.

Exemplo: Torres de Hanói



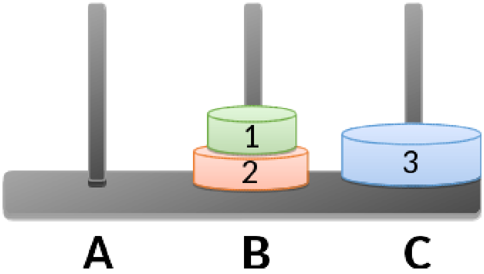
Movimento 2: disco 2 do pino A para o pino C.

Exemplo: Torres de Hanói



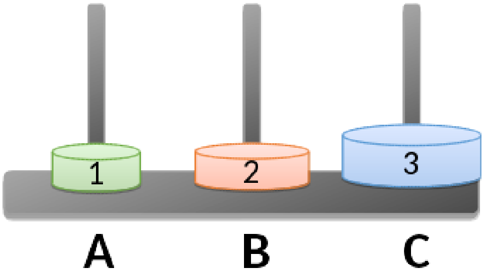
Movimento 3: disco 1 do pino C para o pino C.

Exemplo: Torres de Hanói



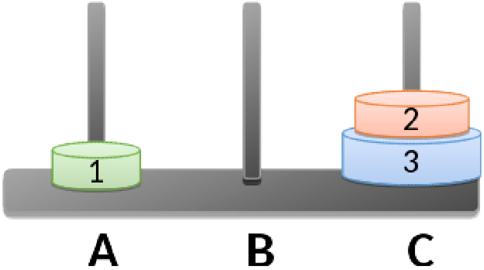
Movimento 4: disco 3 do pino A para o pino C.

Exemplo: Torres de Hanói



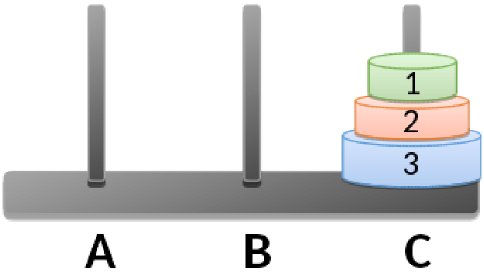
Movimento 5: disco 1 do pino B para o pino A.

Exemplo: Torres de Hanói



Movimento 6: disco 2 do pino B para o pino C.

Exemplo: Torres de Hanói



Movimento 7: disco 1 do pino A para o pino C.

Exemplo: Torres de Hanói

Mover n discos pode ser visualizado em termos de mover $n-1$ discos como a seguir:

- Mova $n-1$ discos do pino A para o pino B, utilizando o pino C para armazenamento temporário;
- Mova o último disco (o maior) do pino A para o pino C;
- Mova os $n-1$ discos do pino B para o pino C, utilizando o pino A para armazenamento temporário.

Exemplo: Torres de Hanói

Algoritmo 5: torres-hanoi(n, P_o, P_d, P_{tmp})

Entrada: quantidade n de discos, pino de origem P_o, pino de destino P_d e pino auxiliar P_{tmp}

Saída: sequência e quantidade de movimentos para mover os n discos do pino P_o para o pino P_d utilizando o pino auxiliar P_{tmp}.

```
início
    // Variável global que indica a ordem de cada movimento na sequência
       de resolução.
    global ordem ← 0
    se n = 1 então
        Imprimir: ordem: disco do pino P_o para o pino P_d
    torres-hanoi(n − 1, P_o, P_tmp, P_d)
    Imprimir: ordem: disco do pino P_o para o pino P_d
    torres-hanoi(n − 1, P_tmp, P_d, P_o)
```

Exemplo: Torres de Hanói

- A quantidade mínima de movimentos m é dada por: $m = 2^n - 1$, em que n é a quantidade de discos;
- Ao tentar encontrar uma solução iterativa para esse problema, é provável que nos encontremos irremediavelmente "amarrados" no gerenciamento dos discos;

Tarefa

Exemplo

Escrever um programa em linguagem C que calcula o movimento de n discos de acordo com as regras estabelecidas.

Referências bibliográficas

FIM

Fim

Fim