Introdução à recursão

Prof. Ítalo Assis

Ajude a melhorar este material =]

Encontrou um erro? Tem uma sugestão?

Envie e-mail para italo.assis@ufersa.edu.br

Agenda

- Funções
- Conceito de recursão
- Métodos recursivos
- Recursão *versus* iteração
- Processamento da recursão
- Recursão indireta

Funções

- Sendo Java uma linguagem orientada a objetos, é mais comum utilizarmos a nomenclatura método em vez de função
  - Veremos mais adiante no curso qual é a diferença
- Para definir uma “função” que poderá ser chamada diretamente pela função *main*, devemos definí-la da seguinte maneira:

```
public static [tipoDoRetorno] [nomeDaFuncao]([lista de parâmetros]) {
    ...
    return [valor de retorno];
}
```

Exemplo

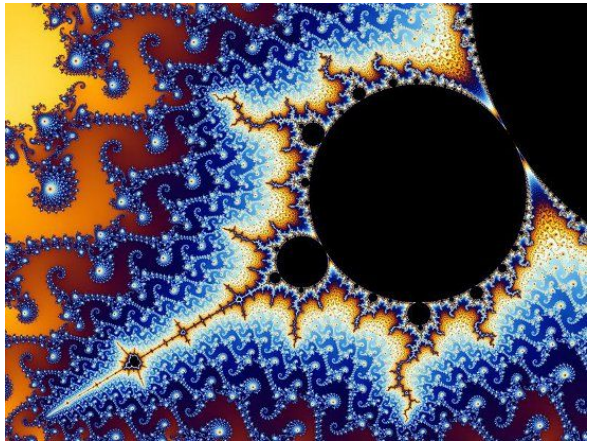
- Escreva um programa que leia 3 números *double* e informe sua média
  - O cálculo da média deve ser feito através de uma função

Recursão

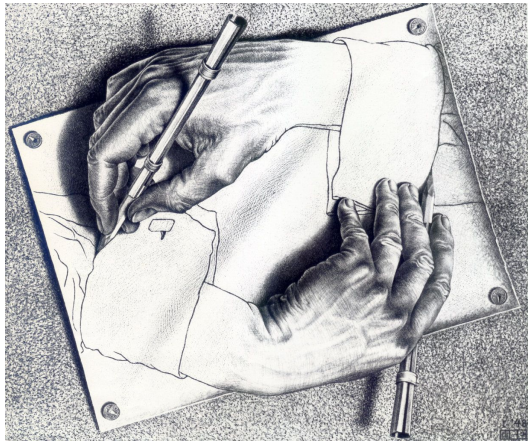
Recursão ocorre quando algo é definido a partir de sua própria definição

"*Recursividade é algo recursivo*" 😁

$$\mathbb{N} = \begin{cases} 0 \\ n+1 & \text{se } n \in \mathbb{N} \end{cases}$$



M.C.Escher



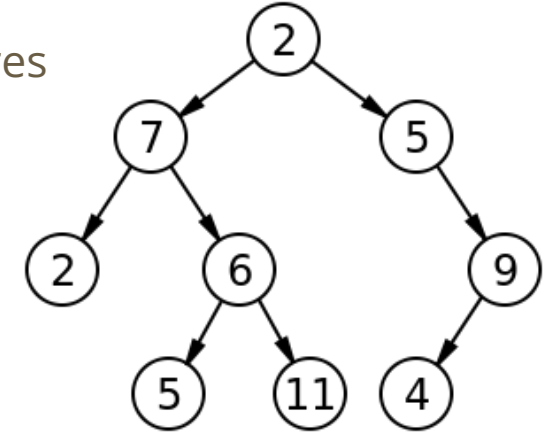
Recursividade em computação

Técnica de resolução de problemas

- Redução de um problema maior em um menor
- As vezes fica mais fácil resolver problemas menores

Áreas

- Computação gráfica (subdivisões espaciais)
- Algoritmos (ordenação)
- Estruturas de dados (árvores de busca)
- ...



Métodos recursivos

1. **Caso base**
   Define quando a recursão deve parar.
   Se não houver, haverá uma recursão infinita!
   No exemplo, a recursão para quando n = 0
2. **Caso recursivo**
   Define quando o caso pode ser resolvido usando casos menores.
   As chamadas recursivas devem conduzir ao caso base, senão haverá recursão infinita!
   No exemplo, há chamada recursiva quando n > 0

Fatorial

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

Exemplo

- Implemente uma função recursiva para calcular o fatorial de um número *n*
- Teste a função criada

Fatorial

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

Recursão versus iteração

- O uso de recursão oferece soluções elegantes para alguns problemas
- Estas soluções não são necessariamente mais velozes, econômicas (em relação ao uso de memória) ou mesmo, em alguns casos, mais claras.
- Se houver uma solução que use laços, sem recursão, e se velocidade ou uso de memória forem críticos, a solução sem recursão deve ser usada.

Recursão *versus* iteração

- Alguns problemas são **naturalmente recursivos**
- Nesses casos, uma solução recursiva é mais simples de implementar
- Exemplo: Série de Fibonacci.

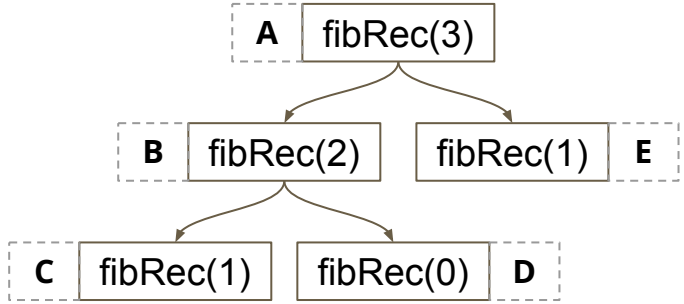
$$f(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ f(n-1) + f(n-2) & \text{se } n > 1 \end{cases}$$

Exemplo

- Vamos criar e teste duas funções para calcular o *n*-ésimo elemento da série de Fibonacci:
  - uma utilizando recursividade;
  - e a outra utilizando uma solução iterativa.

Recursão versus iteração

- Pilha de chamadas de métodos
  - As chamadas são “empilhadas” na memória
  - Para cada chamada um novo escopo é criado

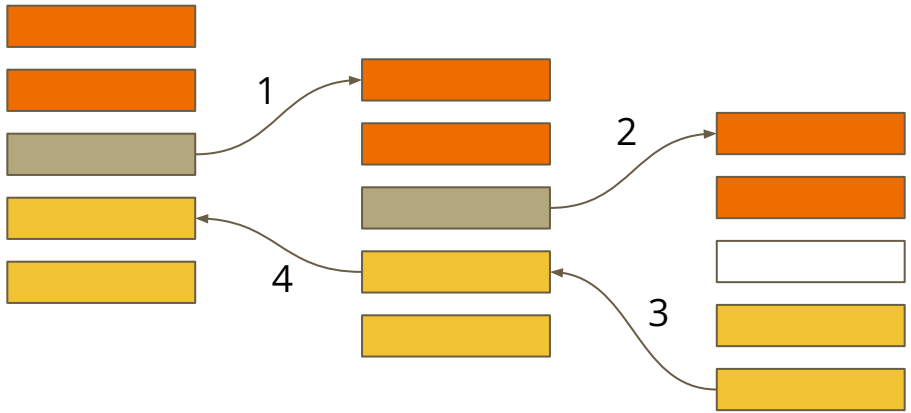


tempo →

Chamada A n = 3	Chamada B n = 2	Chamada C n = 1	Chamada B n = 2	Chamada D n = 0	Chamada B n = 2	Chamada A n = 3	Chamada E n = 1	Chamada A n = 3
	Chamada A n = 3	Chamada B n = 2	Chamada A n = 3	Chamada B n = 2	Chamada A n = 3		Chamada A n = 3	
		Chamada A n = 3		Chamada A n = 3				

Processamento da recursão

- Pode haver processamento:
  - Antes da chamada recursiva
  - Depois da chamada recursiva
- Dependendo de quando a chamada é feita o resultado é diferente

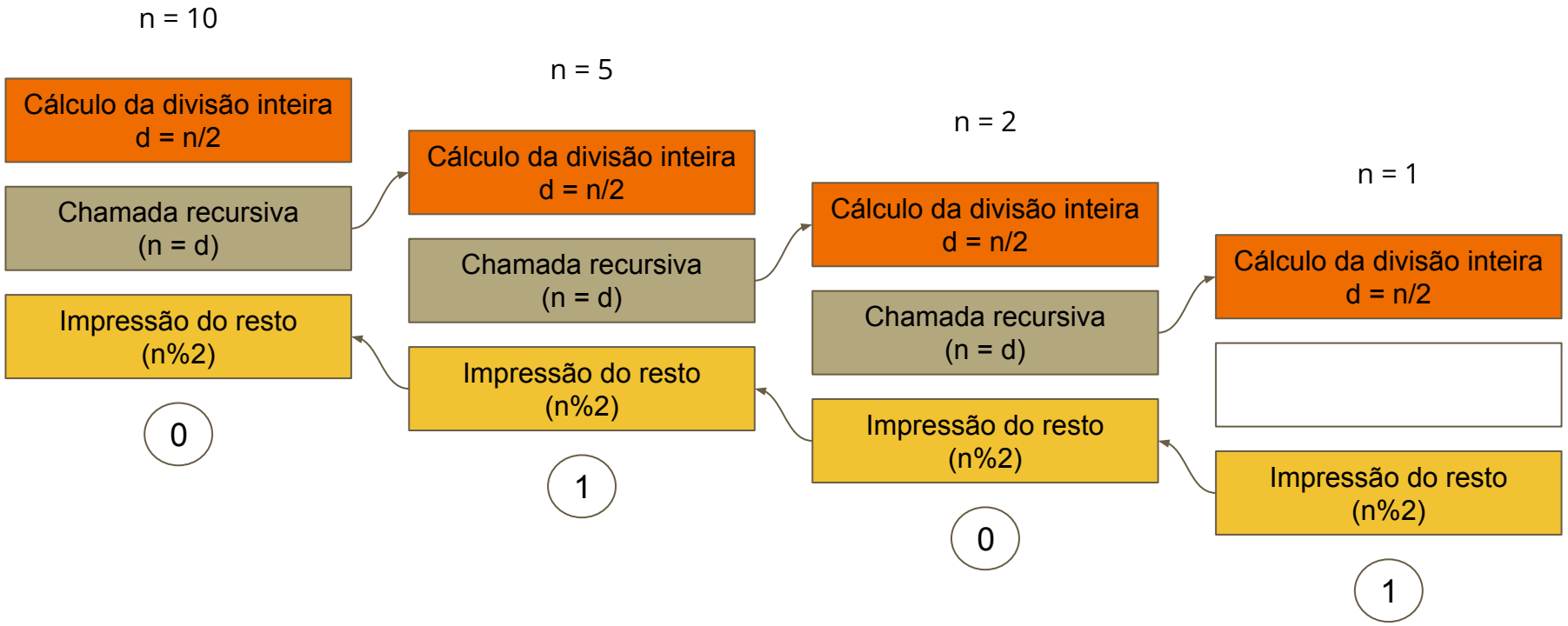


Exemplo

- Vamos escrever e testar uma função que recebe um número inteiro positivo decimal e apresenta seu equivalente binário.
  - Ideia de solução: dividir o valor por 2, pegar o resto e aplicar a mesma solução para a parte inteira do resultado da divisão
    - Ex: para valor 10
      - 10 / 2 = 5 (resta 0)
      - 5 / 2 = 2 (resta 1)
      - 2 / 2 = 1 (resta 0)
      - 1 / 2 = 0 (resta 1)

        O resultado é a sequência de restos na ordem invertida: 1010

Exemplo


n = 10
Cálculo da divisão inteira
d = n/2
Chamada recursiva
(n = d)
Impressão do resto
(n%2)
0
n = 5
Cálculo da divisão inteira
d = n/2
Chamada recursiva
(n = d)
Impressão do resto
(n%2)
1
n = 2
Cálculo da divisão inteira
d = n/2
Chamada recursiva
(n = d)
Impressão do resto
(n%2)
0
n = 1
Cálculo da divisão inteira
d = n/2
Impressão do resto
(n%2)
1

Recursão indireta (ou múltipla)

Uma recursão pode ser realizada usando mais de uma função

Por exemplo:

- Um número natural é par se ele for 0 ou se seu antecessor for ímpar;
- Um número natural é ímpar se ele não for 0 e seu antecessor for par.

Exemplo

- Utilizando recursão indireta, vamos criar e testar funções que recebem um número natural *num* como argumento:
  - *ehImpar*
    - Retorna *true* se *num* for ímpar e *false* caso contrário
  - e *ehPar*
    - Retorna *true* se *num* for par e *false* caso contrário

Os códigos relacionados a esta aula estão disponíveis em

https://github.com/italoaug/Programacao-Orientada-a-Objetos/tree/main/codigos/recursao

Referências

SANTOS, R. **Introdução à programação orientada a objetos usando JAVA**. 2. ed. Rio de Janeiro: Campus, 2013. 336p.

DEITEL, Paul; DEITEL, Harvey. **Java: como programar**. 10. ed. São Paulo: Pearson Education do Brasil, 2017.