CURSO DE PROGRAMAÇÃO EM JAVA

Aula 11

Recursividade

"

Um objeto é dito recursivo se pode ser definido em termos de si próprio.

“Para fazer iogurte, você precisa de leite e de um pouco de iogurte.”

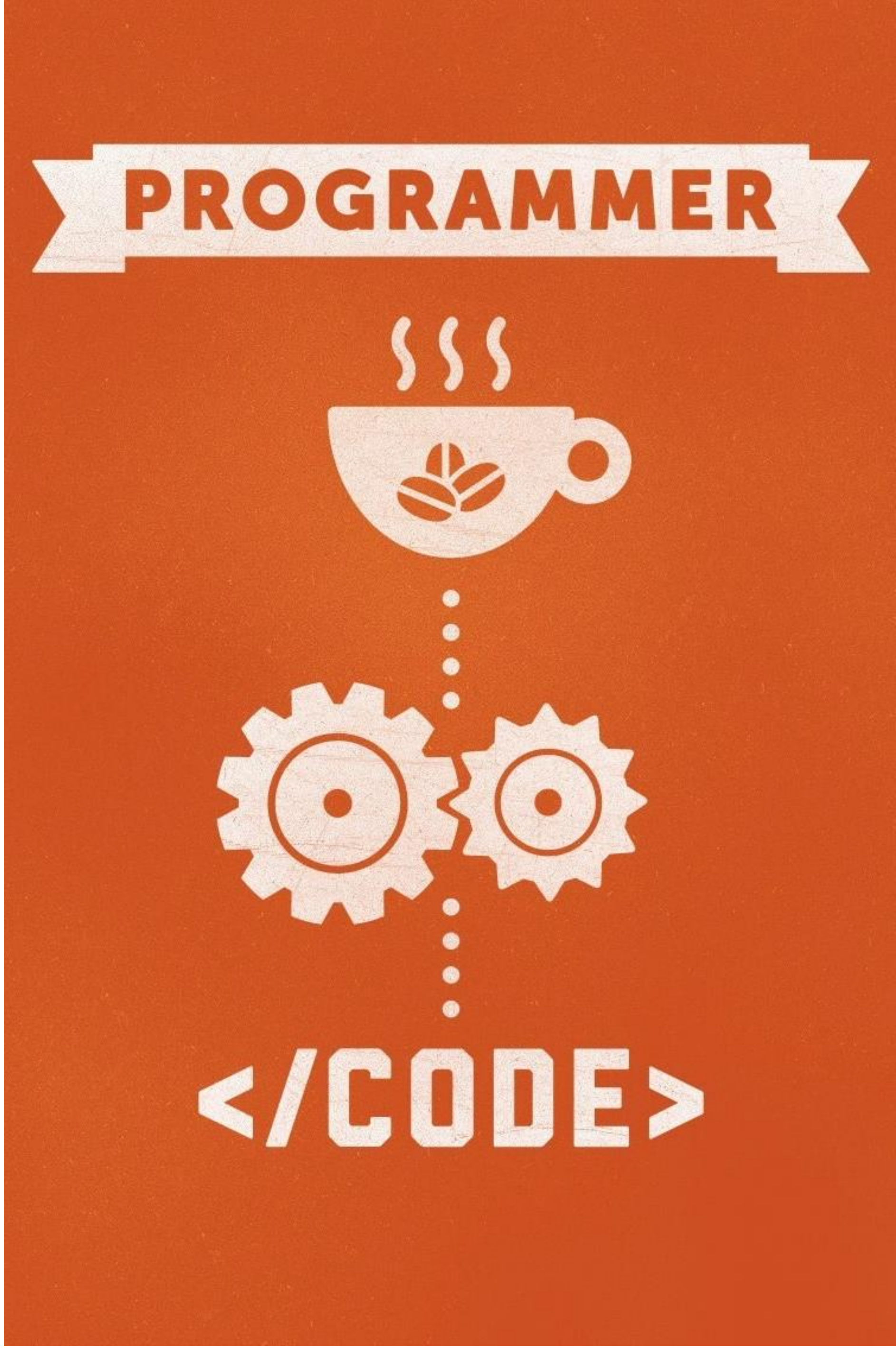
1.
Recursividade

Recursividade é a propriedade que uma função tem de chamar a si própria, diretamente ou não. Isto é usado para simplificar um problema.

Recursividade

- A recursão é uma forma interessante de resolver problemas, pois o divide em problemas menores de **mesma natureza**.

- Um processo recursivo consiste de duas partes:
  - O **caso trivial**, cuja solução é conhecida.
  - Uma **relação de recorrência** que reduz o problema a um ou mais problemas menores de mesma natureza.



Recursividade

- Um programa recursivo é um programa que chama a si mesmo, direta ou indiretamente.

- **Vantagens**
  - Redução do tamanho do código fonte
  - Permite descrever algoritmos de forma mais clara e Concisa
- **Desvantagens**
  - Redução do desempenho de execução devido ao tempo para gerenciamento de chamadas
  - Dificuldades na depuração de programas recursivos

Características das funções Recursivas

- As funções recursivas contêm duas partes fundamentais:

Ponto de Parada:
o ponto de parada da recursividade é resolvido sem utilização de recursividade, sendo este ponto geralmente um limite superior ou inferior da regra geral.

Relação de recorrência :
a relação de recorrência da recursividade reduz a resolução do problema através da invocação recursiva de casos menores e assim sucessivamente, até atingir o **ponto de parada** que finaliza o método.

Função fatorial

- A função fatorial de um inteiro não negativo pode ser definida como:

$$\text{fat}(n) = \begin{cases} 1, & \text{se } n = 1 \\ n * \text{fat}(n-1), & \text{se } n > 1 \end{cases}$$

- Esta definição estabelece um **processo recursivo** para calcular o fatorial de um inteiro **n**.
- Caso trivial: n=0.
- Método geral: n * (n-1)!.

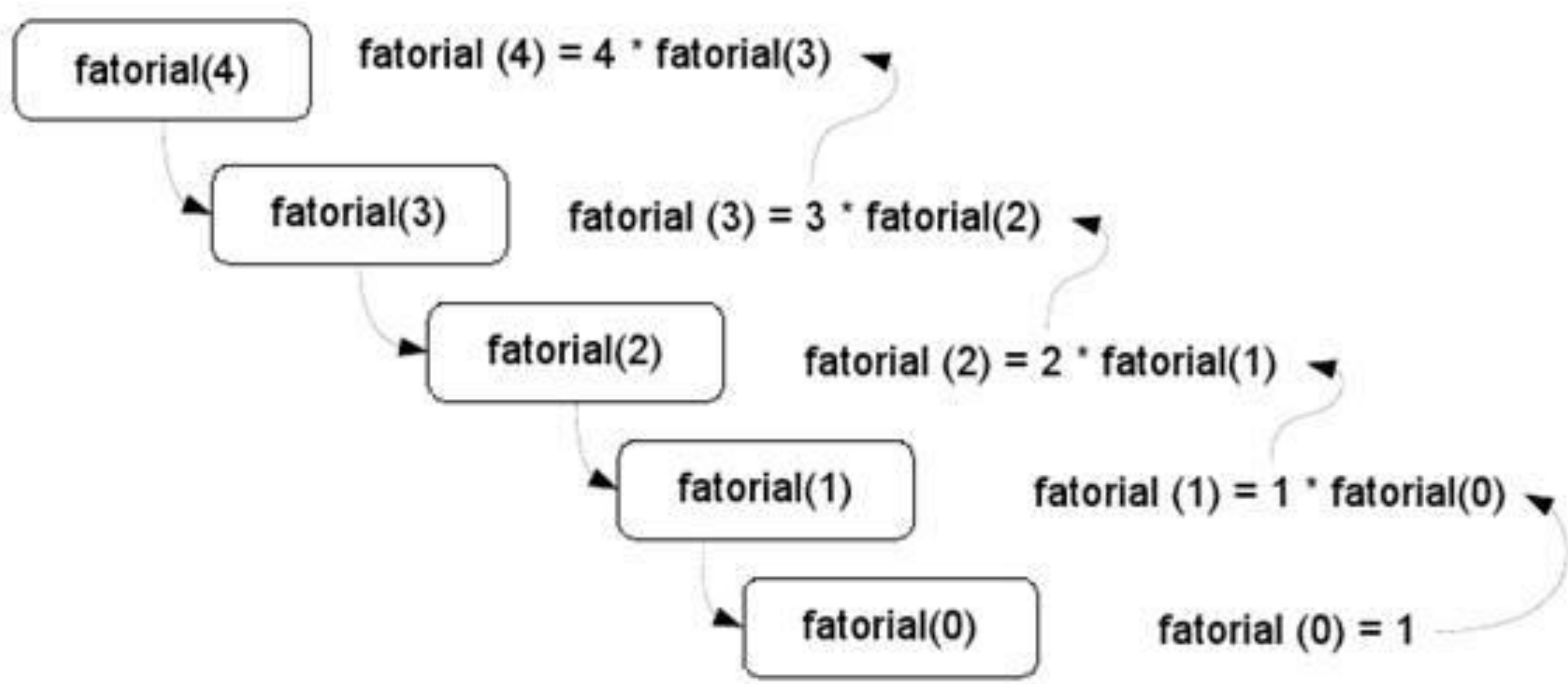
Função fatorial

Assim, usando-se este processo recursivo, o cálculo de 4!, por exemplo, é feito como a seguir:

```
private static int fatorial(int n){
        if(n==1)
                return n;
        return fatorial(n-1)*n;
}
```

4! = 4 * 3!
= 4 * (3 * 2!)
= 4 * (3 * (2 * 1!))
= 4 * (3 * (2 * (1 * 0!)))
= 4 * (3 * (2 * (1 * 1)))
= 4 * (3 * (2 * 1))
= 4 * (3 * 2)
= 4 * 6
= 24

Função fatorial



Obs: O "fatorial(4)" só pode ser descoberto depois que o "fatorial(3)" for descoberto, que por sua vez só poderá ser descoberto depois do fatorial(2) e assim por diante.

Função fatorial

```
factorial( n ):
if n == 1:
    return 1
else:
    return n * factorial(n-1):
                if n == 1:
                    return 1
                else:
```

factorial(n) =

www.penjee.com

Exemplo

```
//Soma elementos do vetor

public void somaElementos(int[] vet, n){
    if(n==0)
        retorna vet[0];
    else
        retorna vet[n]+somaElementos(vet, n-1);
}
```

DESAFIO

E aí, vamos praticar?

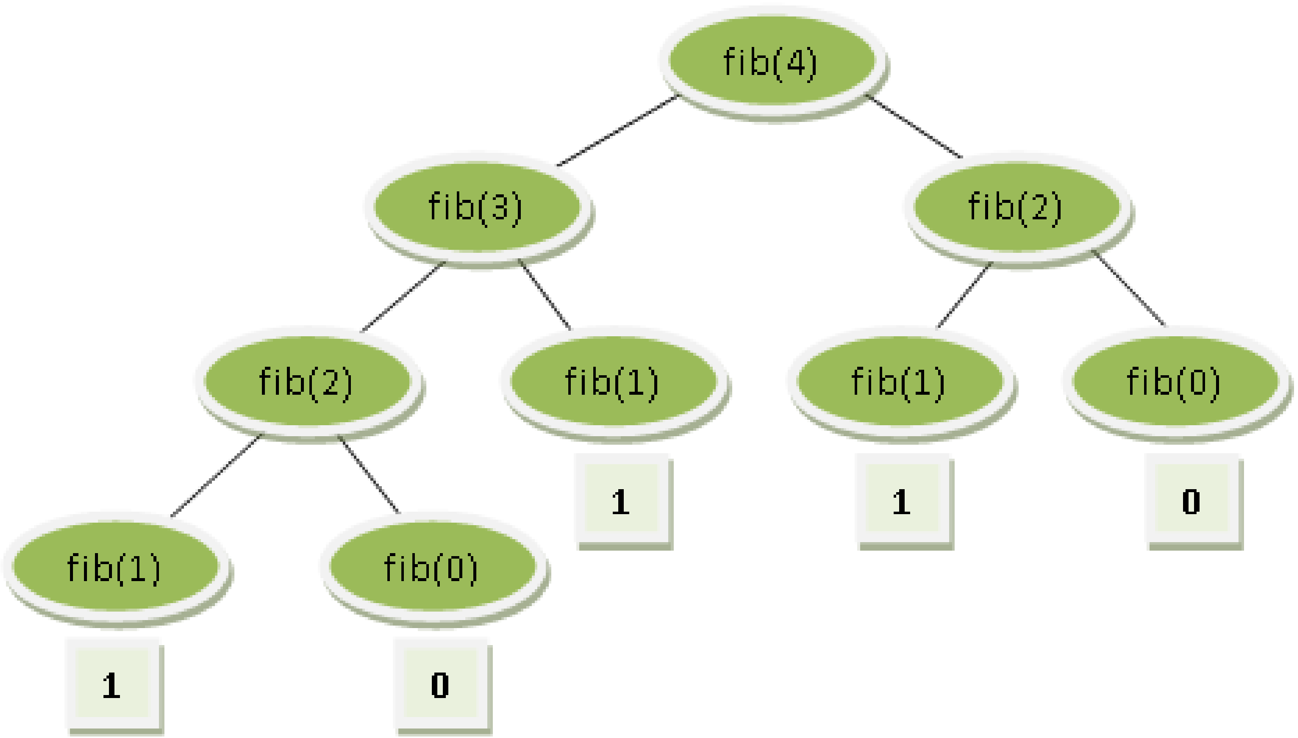
Fibonacci, Quantas Chamadas?

Quase todo estudante de Ciência da Computação recebe em algum momento no início de seu curso de graduação algum problema envolvendo a sequência de Fibonacci. Tal sequência tem como os dois primeiros valores 0 (zero) e 1 (um) e cada próximo valor será sempre a soma dos dois valores imediatamente anteriores. Por definição, podemos apresentar a seguinte fórmula para encontrar qualquer número da sequência de Fibonacci:

fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2);

Uma das formas de encontrar o número de Fibonacci é através de chamadas recursivas. Isto é ilustrado a seguir, apresentando a árvore de derivação ao calcularmos o valor fib(4), ou seja o 5º valor desta sequência:

Fibonacci, Quantas Chamadas?



Desta forma,
fib(4) = 1+0+1+1+0 = 3
Foram feitas 8 calls, ou seja, 8 chamadas recursivas.

Fibonacci, Quantas Chamadas?

Entrada: A primeira linha da entrada contém um único inteiro N, indicando o número de casos de teste. Cada caso de teste contém um inteiro X ($1 \leq X \leq 39$) .

Saída: Para cada caso de teste de entrada deverá ser apresentada uma linha de saída, no seguinte formato: fib(n) = num_calls calls = result, aonde num_calls é o número de chamadas recursivas, tendo sempre um espaço antes e depois do sinal de igualdade, conforme o exemplo abaixo.

Entrada:

```
2
5
4
```

Saída:

```
fib(5) = 14 calls = 5
fib(4) = 8 calls = 3
```

Figurinhas

Ricardo e Vicente são aficionados por figurinhas. Nas horas vagas, eles arrumam um jeito de jogar um “bafo” ou algum outro jogo que envolva tais figurinhas. Ambos também têm o hábito de trocarem as figuras repetidas com seus amigos e certo dia pensaram em uma brincadeira diferente. Chamaram todos os amigos e propuseram o seguinte: com as figurinhas em mãos, cada um tentava fazer uma troca com o amigo que estava mais perto seguindo a seguinte regra: cada um contava quantas figurinhas tinha. Em seguida, eles tinham que dividir as figurinhas de cada um em pilhas do mesmo tamanho, no maior tamanho que fosse possível para ambos. Então, cada um escolhia uma das pilhas de figurinhas do amigo para receber. Por exemplo, se Ricardo e Vicente fossem trocar as figurinhas e tivessem respectivamente 8 e 12 figuras, ambos dividiam todas as suas figuras em pilhas de 4 figuras (Ricardo teria 2 pilhas e Vicente teria 3 pilhas) e ambos escolhiam uma pilha do amigo para receber.

Figurinhas

Entrada: A primeira linha da entrada contém um único inteiro **N** (1 ≤ **N** ≤ 3000), indicando o número de casos de teste. Cada caso de teste contém 2 inteiros **F1** (1 ≤ **F1** ≤ 1000) e **F2** (1 ≤ **F2** ≤ 1000) indicando, respectivamente, a quantidade de figurinhas que Ricardo e Vicente têm para trocar.

Saída: Para cada caso de teste de entrada haverá um valor na saída, representando o tamanho máximo da pilha de figurinhas que poderia ser trocada entre dois jogadores.

Entrada:

3
8 12
9 27
259 111

Saída:

4
9
37

Obrigado!
Alguma pergunta?

Você pode nos contatar em:
ywassef@hotmail.com