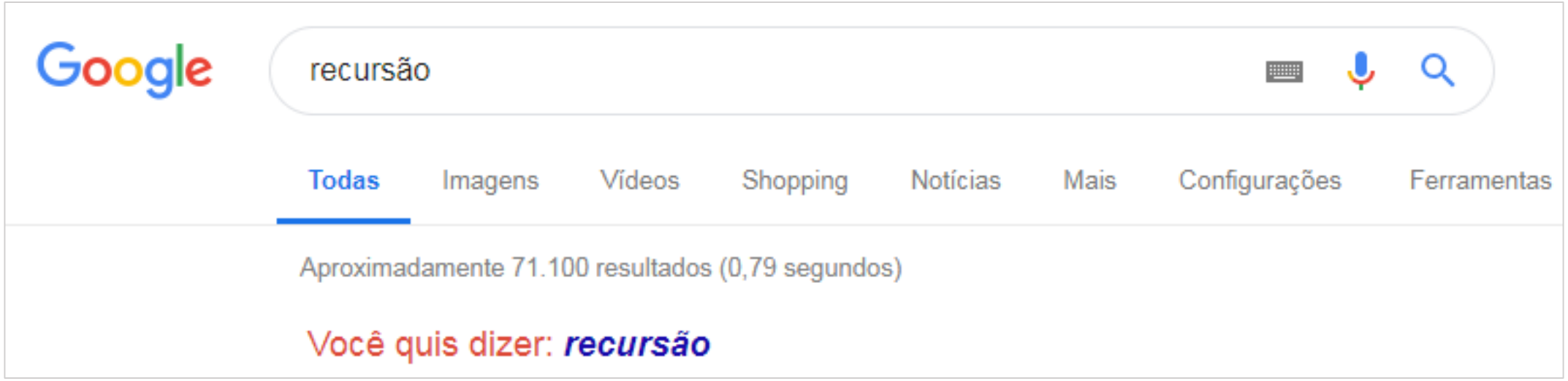
Recursividade

Pesquisa, Ordenação e Técnicas de Armazenamento

Prof. Me. Orlando da Silva Junior

O que é **recursão**?



O que é **recursão**?

Segundo o **dicionário Michaelis**:

https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/recursividade/

recursividade

re·cur·si·vi·da·de

sf

LING , LÓG Propriedade daquilo que se pode repetir indefinidamente.

ETIMOLOGIA

der de *recursivo+i+dade*, como *fr récursivité*.

O que é **recursão**?

Computacionalmente falando:

- Procedimento ou função que chama a si mesmo
- Um problema definido em termos de si mesmo
- Um método para resolver um problema que “quebre” esse problema em subproblemas cada vez menores de modo a tornar sua resolução trivial

Problema: como repetir N vezes uma mensagem X?

Solução 1: elabore um método chamado *exibeMensagem* que receba como entrada uma cadeia de caracteres TEXTO e um número inteiro N e imprima TEXTO em tela utilizando um laço de repetição que reproduza essa mensagem N vezes. Esse método não deverá devolver nenhum valor de retorno.

Resolução:

```
public void exibeMensagem(String texto, int n){
      for (int i = 0; i < n; i++){
            System.out.println(texto);
      }
}
```

Problema: como repetir N vezes uma mensagem X?

Solução 2: elabore um método recursivo chamado *exibeMensagem* que receba como entrada uma cadeia de caracteres TEXTO e um número inteiro N e imprima TEXTO em tela utilizando o mesmo método *exibeMensagem* com a mesma entrada de texto mas decrementando o valor N em passo 1 até que a entrada N seja igual a 0. Esse método não deverá devolver nenhum valor de retorno.

Resolução:

```
public void exibeMensagem(String texto, int n){
      if (n > 0) {
            System.out.println(texto);
            exibeMensagem(texto, n - 1);
      }
}
```

```
public void exibeMensagem(String texto, int n){
      if (n > 0) {
            System.out.println(texto);
            exibeMensagem(texto, n - 1);
      }
}
```

Isto é uma **chamada recursiva**!

No método principal ***main***, poderemos utilizar o método recursivo acima desta forma:

```
public static void main(String[] args) {
    exibeMensagem("Olá, Mundo Recursivo!", 5);
}
```

```
public static void exibeMensagem(String texto, int n){
      if (n > 0) {
            System.out.println(texto);
            exibeMensagem(texto, n - 1);
      }
}
```

Problema: como calcular o fatorial de um número inteiro positivo N?

Como calcular o fatorial de um número?

Para calcular o fatorial de um número inteiro positivo N, devemos realizar sucessivas multiplicações:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (\cdots) \cdot 3 \cdot 2 \cdot 1$$

Por exemplo:

- $3! = 3 \cdot 2 \cdot 1 = 6$
- $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$
- $1! = 1$
- $0! = 1$ (por definição!)

Problema: como calcular o fatorial de um número inteiro positivo N?

Solução 1: elabore um método iterativo chamado *fatorial* que receba como entrada um número inteiro N e multiplique o valor de N sucessivamente por seu antecessor até que o valor antecessor seja igual a 1.

Resolução:

```
public int fatorial(int n){
      int fatorial = 1;
      for (int i = 1; i <= n; i++){
            fatorial = fatorial * i;
      }
      return fatorial;
}
```

Problema: como calcular o fatorial de um número inteiro positivo N?

Solução 2: elabore um método recursivo chamado *fatorial* que receba como entrada um número inteiro N e multiplique o valor de N sucessivamente por seu antecessor até que o valor antecessor seja igual a 1.

Resolução:

```
public int fatorial(int n){
      if (n <= 1){
            return 1;
      }
      return n * fatorial(n - 1);
}
```

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

ONDE NA MEMÓRIA FICAM ARMAZENADAS AS **CHAMADAS RECURSIVAS**?

Vamos falar sobre...

SISTEMAS OPERACIONAIS

*Ao executar um **processo**, o sistema operacional aloca um espaço de memória destinado à execução desse processo, sendo que parte desse espaço de memória é destinado às chamadas que o processo faz aos subprocessos.*

Vamos falar sobre...

SISTEMAS OPERACIONAIS

*O método **main** é o ponto de entrada para a execução do processo.*

```
public static void main(String[] args) {


}
```

Vamos falar sobre...

SISTEMAS OPERACIONAIS

*O métodos que são chamados pelo método **main** têm como ponto de entrada o próprio método **main**.*

```
public static void main(String[] args) {
      int valor = fatorial(5);
}
```

Vamos falar sobre...

SISTEMAS OPERACIONAIS

*Quando o método **fatorial** é chamado, o método **main** fica "suspenso" até que o método **fatorial** finalize. Ao finalizar, o método **main** retoma o controle.*

```
public static void main(String[] args) {
      int valor = fatorial(5);
}
```

Vamos falar sobre...

SISTEMAS OPERACIONAIS

Quando um subprograma é chamado, linguagens de programação modernas, como Java e C#, criam um espaço na memória chamado ***Instância de Registro de Ativação*** *(IRA).*

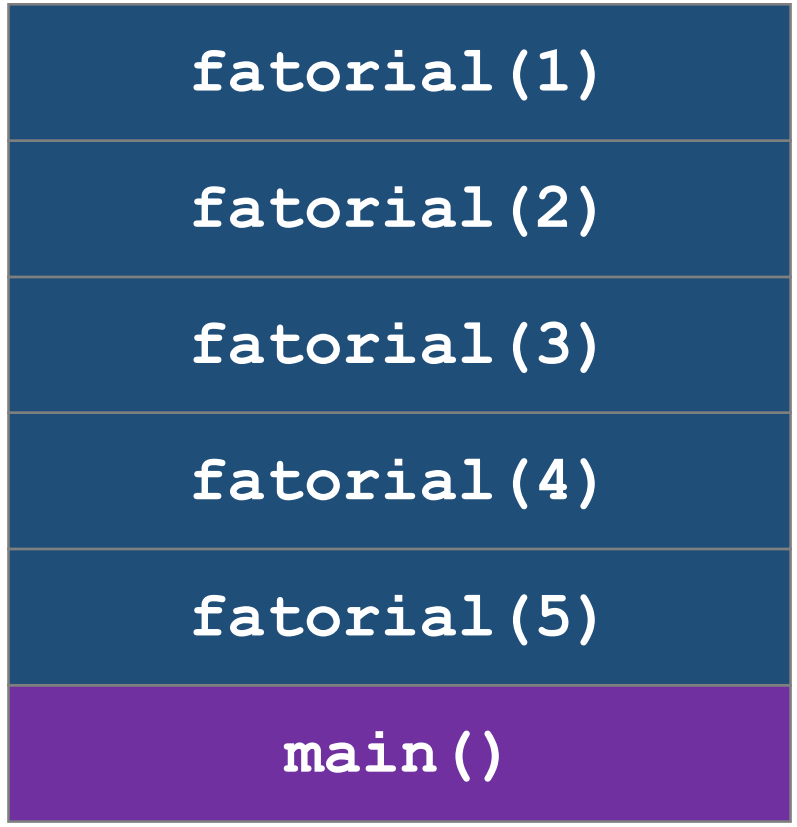
Vamos falar sobre...

SISTEMAS OPERACIONAIS

Sempre que um subprograma é chamado, uma IRA é criada e colocada no topo da pilha de chamadas.

Em chamadas recursivas, poderão existir diversas IRAs com o mesmo formato, mas com conteúdos diferentes.

*A **pilha de chamadas** (ou pilha de execução) é um espaço na memória que armazena informações dos subprogramas que são chamados pelo subprograma chamador ou pelo programa principal.*



Esse processo de *empilhar/desempilhar* é como o computador faz para manter o rastreamento de qual cópia do subprograma está em execução.

O **estouro da pilha de execução** acontece quando o programa consome mais memória do que a disponível.

As causas mais comuns para o estouro da pilha são chamadas infinitas e alocação excessiva de memória.

Soluções recursivas e iterativas

Tanto no programa de exibir uma mensagem N vezes quanto no programa para o cálculo do fatorial, você observou a construção de 2 soluções:

- Uma solução iterativa utilizando o laço **for**; e
- Uma solução recursiva utilizando o próprio método como laço de repetição.

Análise comparativa

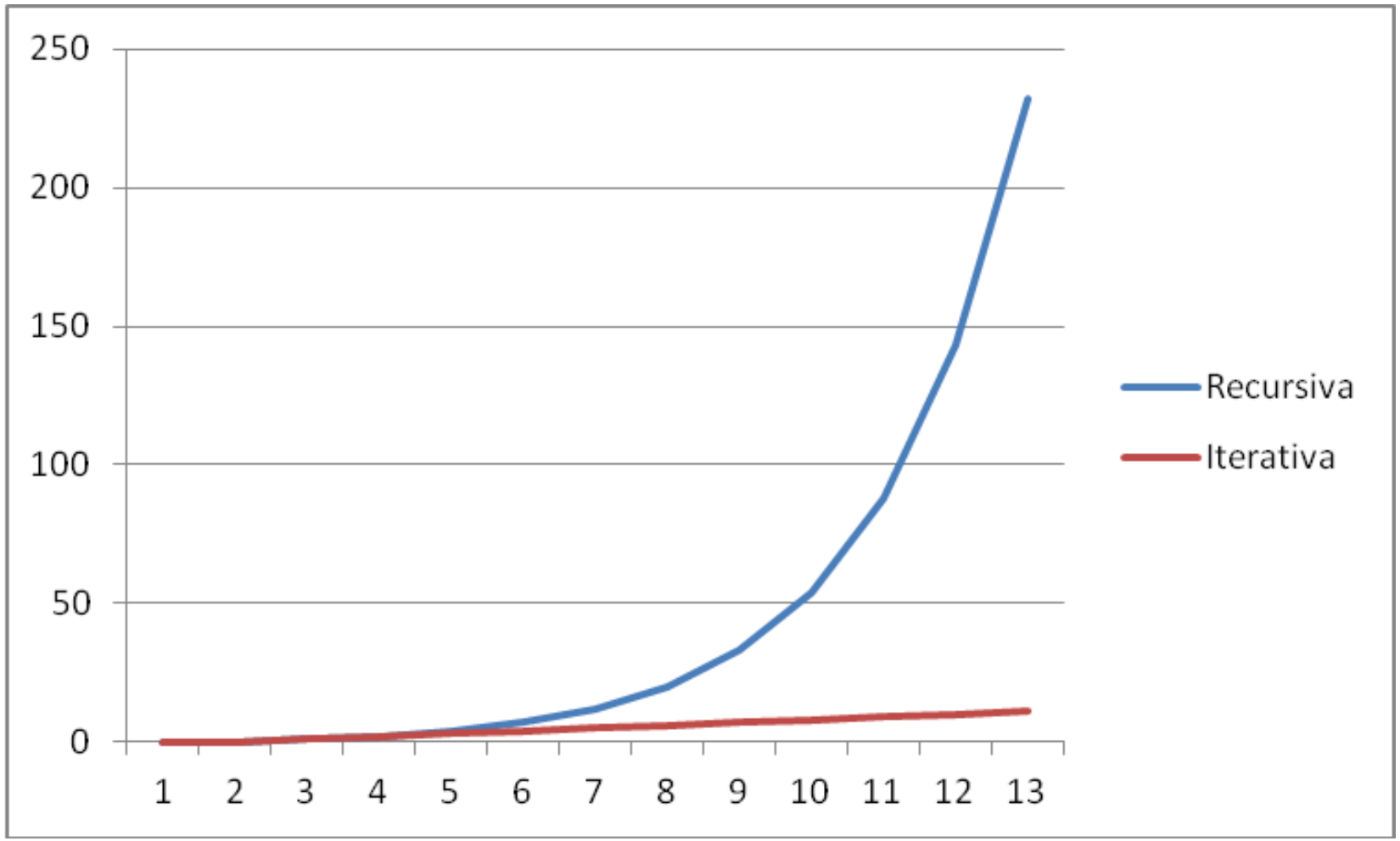


Imagem: http://algoritmoconcreta.blogspot.com/2013/07/numeros-de-fibonacci.html

*Podemos ainda construir soluções **diretamente** recursivas ou **indiretamente** recursivas.*

Na **recursão direta**, o procedimento chama a si mesmo para executar os comandos e concluir as operações computacionais.

Na **recursão indireta**, o procedimento chama outro procedimento que contém uma referência direta ou indireta ao primeiro procedimento.

Recursão **indireta**

*Um número N é **par** se, e somente se, N-1 é **ímpar**.*

*Um número N é **ímpar** se, e somente se, N-1 é **par**.*

Vantagens da recursividade

1. É a forma mais simples e direta de solução de problemas que têm definição recursiva. Nós verificamos que os problemas recursivos também podem ser construídos com soluções iterativas.

2. É excelente para acessar estruturas de dados implicitamente recursivas, como árvores.

3. Facilita a compreensão da lógica do subprograma.

4. O código recursivo tende a ser mais enxuto que o código não recursivo.

Desvantagens da recursividade

1. Necessita de memória extra durante a execução das chamadas recursivas.

2. Aumenta o tempo de processamento em razão das diversas chamadas recursivas.

3. É impraticável em problemas que só podem ser solucionados com um número muito elevado de chamadas.

4. Exige mais espaço na pilha de execução.

6 dicas para usar recursão

1. Utilize recursão para entender melhor o problema, mas evite considera-la quando o desempenho é uma exigência.

2. Defina sempre uma condição que permita a parada das chamadas recursivas.

3. É comum faltar memória para a execução. Fique atento ao número elevado de chamadas recursivas.

4. Considere estimar a quantidade de chamadas recursivas que o método realizará. Buscar um número em um vetor de 1000 elementos pode provocar até 999 chamadas recursivas.

5. Utilize mensagens no código para acompanhar as chamadas recursivas.

6. Teste valores limites para os parâmetros de entrada a fim de garantir a viabilidade do algoritmo.

Exercício

*Por volta do ano 1200, o matemático italiano Leonardo de Pisa percebeu a existência de uma curiosa sequência numérica. Chamada de **Sequência de Fibonacci**, ela tem como primeiros termos os números 0 e 1, sendo os termos subsequentes obtido pela soma dos dois termos predecessores:*

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

*Elabore um **algoritmo recursivo** em pseudocódigo ou linguagem Java para, dado um valor numérico inteiro como entrada correspondente à posição do número de Fibonacci na sequência, devolver como saída o número correspondente à posição na sequência.*

Exemplo:

- *Entrada: 0 Saída: 0*
- *Entrada: 1 Saída: 1*
- *Entrada: 3 Saída: 2*
- *Entrada: 8 Saída: 21*

Exercício

*Elabore a **versão iterativa** do algoritmo de Fibonacci.*

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

<u>Exemplo:</u>

- *Entrada: 0 Saída: 0*
- *Entrada: 1 Saída: 1*
- *Entrada: 3 Saída: 2*
- *Entrada: 8 Saída: 21*