

# Recursividade e Percurso Árvore Binária

Códigos de Alta Performance

PROFa. PATRÍCIA MAGNA - [profpaticia.magna@fiap.com.br](mailto:profpaticia.magna@fiap.com.br)

**Recursividade** é um termo usado de maneira mais geral para descrever o processo de repetição de um objeto de um jeito similar ao que já fora mostrado. Um bom exemplo disso são as imagens repetidas que aparecem quando dois espelhos são apontados um para o outro.

<https://pt.wikipedia.org/wiki/Recursividade>



Alguns problemas têm como propriedade de ter sua solução facilitada a cada instância menor do mesmo problema. Diz-se que esses problemas possuem uma *estrutura recursiva*.

A resolução desse tipo de problema baseia-se na aplicação do seguinte método:

1. se a instância em questão for pequena, resolva-a diretamente;
2. Senão, *reduza-a* a uma instância menor do mesmo problema, aplique o método à instância menor, volte à instância original.

A aplicação desse método produz um algoritmo *recursivo*.

Algoritmo recursivos são normalmente implementados usando funções recursivas ou métodos recursivos.

Uma função é dita recursiva quando ela chama a ela mesma para resolver um determinado problema. Passando para a função chamada uma instância menor do problema.

Um exemplo de uso de recursividade é o percurso de todos os nós de uma árvore (veremos daqui a pouco...).

$$n! = \begin{cases} n * (n-1)!, & \text{se } n \neq 0 \\ 1, & \text{caso contrário} \end{cases}$$

$$n! = \begin{cases} n * (n-1)!, & \text{se } n \neq 0 \\ 1, & \text{caso contrário} \end{cases}$$

➤  $4! = 4 * 3!$

➤  $3! = 3 * 2!$

➤  $2! = 2 * 1!$

➤  $1! = 1 * 0!$

➤  $0! = 1$

```
public static void main(String[] args) {  
    int fat;  
    fat = fatorial (4);  
    System.out.println("Fatorial = " + fat);  
  
}  
public static int fatorial (int n) {  
    if (n != 0)  
        return ( n * fatorial (n-1) );  
    else  
        return (1);  
}
```



**Ativação 1**

`n = 4 => return (4 * fatorial (3))`

**Ativação 1**

**Ativação 2**

`n = 3 => return ( 3 * fatorial (2) )`

**Ativação 1**

**Ativação 2**

**Ativação 3**

$n = 2 \Rightarrow \text{return } (2 * \text{fatorial}(1))$

**Ativação 1**

**Ativação 2**

**Ativação 3**

**Ativação 4**

$n = 1 \Rightarrow \text{return } (1 * \text{fatorial}(0))$

**Ativação 1**

**Ativação 2**

**Ativação 3**

**Ativação 4**

**Ativação 5**

$n = 0 \Rightarrow \text{return } (1)$

**Ativação 1**

**Ativação 2**

**Ativação 3**

**Ativação 4**

$n = 1 \Rightarrow \text{return } (1 * \text{fatorial}(0))$

1

**Ativação 1**

**Ativação 2**

**Ativação 3**

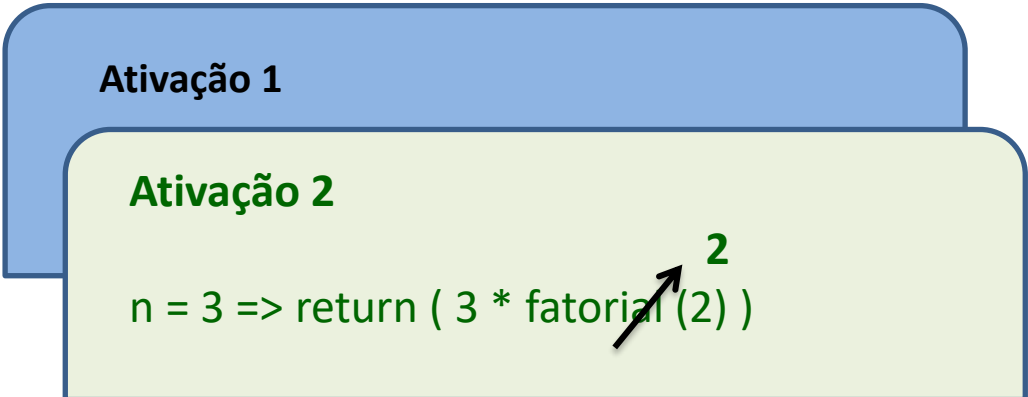
$n = 2 \Rightarrow \text{return } (2 * \text{fatorial}(1))$

1

**Ativação 1**

**Ativação 2**

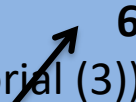
$n = 3 \Rightarrow \text{return} ( 3 * \text{fatorial}(2) )$






**Ativação 1**

$n = 4 \Rightarrow \text{return } (4 * \text{fatorial}(3))$

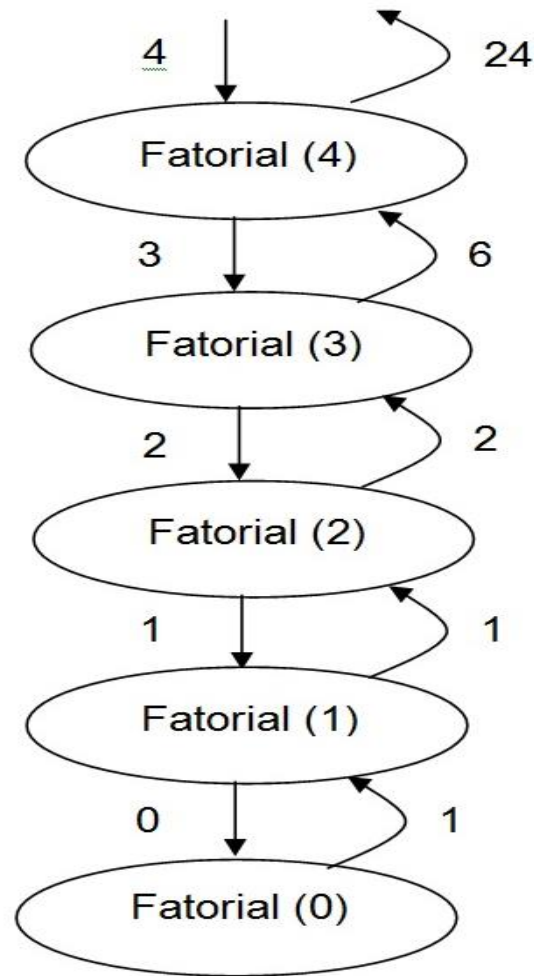


```
public static void main(String[] args) {  
    int fat;  
    fat = fatorial (4);  
    System.out.println("Fatorial = " + fat);  
}  
public static int fatorial (int n) {  
    if (n != 0)  
        return ( n * fatorial (n-1) );  
    else  
        return (1);  
}
```



# Outra Forma de Representar as ativações de função recursiva

FIA/P



```
public static int mdc(int n, int m){  
    if (n > m)  
        return(mdc(m,n));  
    else {  
        if (n == 0)  
            return(m);  
        else  
            return(mdc(n, m%n));  
    }  
}
```

```
public static int mdc(int n, int m){
    if (n > m)
        return(mdc(m,n));
    else {
        if (n == 0)
            return(m);
        else
            return(mdc(n,m%n));
    }
}

public static void main(String[] args) {
    int x = mdc (8,4);
    System.out.println(" MDC entre 8 e 4 = " + x);
}
```

**Ativação 1**

$n = 8, m=4 \Rightarrow n>m \Rightarrow \text{return}(\text{mdc}(4,8))$

**Ativação 1****Ativação 2**

$n=4$  e  $m=8 \Rightarrow n < m$  e  $n \neq 0 \Rightarrow \text{return}(\text{mdc}(4, 8\%4))$

**Ativação 1****Ativação 2**

$n=4$  e  $m=8 \Rightarrow n < m$  e  $n \neq 0 \Rightarrow \text{return}(\text{mdc}(4,0))$



**Ativação 1**

**Ativação 2**

**Ativação 3**

$n = 4$  e  $m = 0 \Rightarrow n > m \Rightarrow \text{return}(\text{mdc}(0, 4))$

**Ativação 1**

**Ativação 2**

**Ativação 3**

**Ativação 4**

$n=0$  e  $m=4 \Rightarrow n < m$  e  $n=0 \Rightarrow \text{return } (4)$

**Ativação 1**

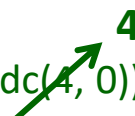
**Ativação 2**

**Ativação 3**

$n = 4$  e  $m = 0 \Rightarrow n > m \Rightarrow \text{return}(\text{mdc}(0, 4))$

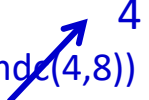
**Ativação 1****Ativação 2**

$n=4$  e  $m=8 \Rightarrow n < m$  e  $n \neq 0 \Rightarrow \text{return}(\text{mdc}(4, 0))$



**Ativação 1**

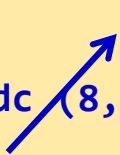
$n=8$  e  $m=4 \Rightarrow n > m \Rightarrow \text{return}(\text{mdc}(4,8))$



```
public static int mdc(int n, int m){
    if (n > m)
        return(mdc(m,n));
    else {
        if (n == 0)
            return(m);
        else
            return(mdc(n,m%n));
    }
}

public static void main(String[] args) {

    int x = mdc(8,4);
    System.out.println(" MDC entre 8 e 4 = " + x);
}
```



Escreva uma função recursiva que calcule o somatório:

$$\sum_0^n soma$$

$$\Sigma n = \begin{cases} n + \Sigma (n - 1), & \text{se } n \neq 0 \\ 0, & \text{caso contrário} \end{cases}$$



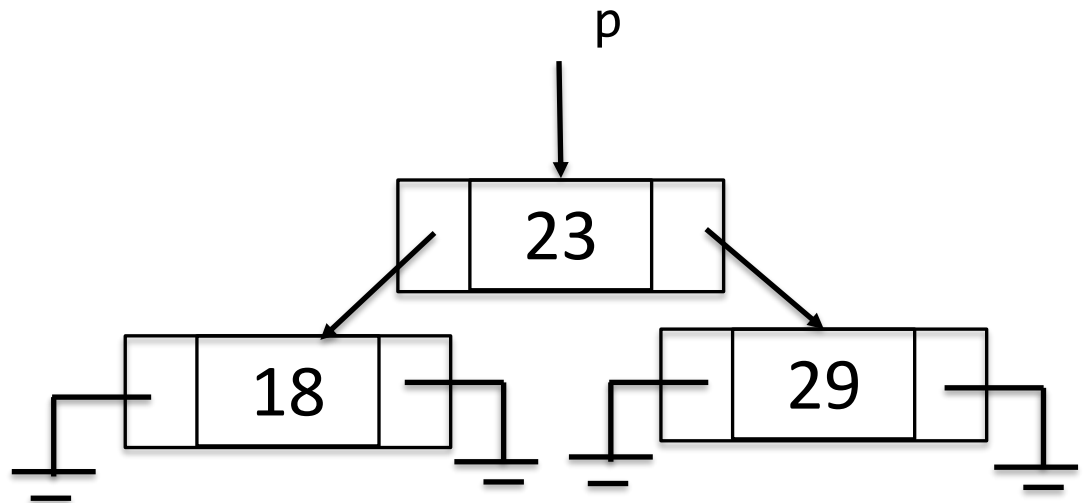
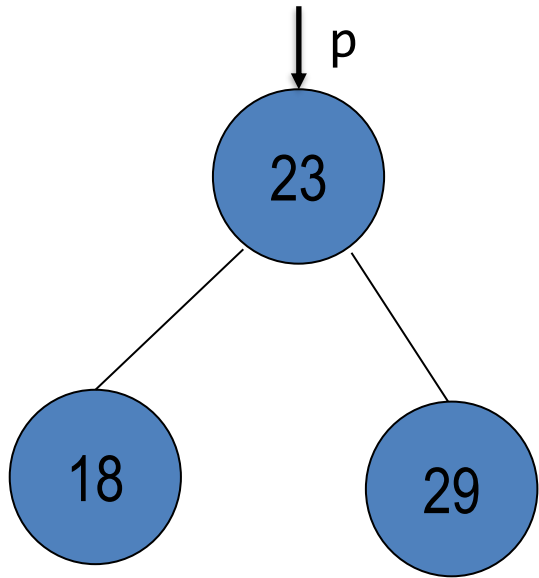
O problema de “percorrer” todos os nós de uma árvore binária de profundidade  $d$  é difícil.

A resolução desse tipo de problema baseia-se na aplicação do seguinte método:

1. se a instância estiver em um nó folha (nós sem filhos) basta “acessar ” o dado contido no nó;
2. Senão, passe para o nível do nó filho de um dos seus ramos , assim, reduzindo a uma instância menor do mesmo problema.

```
private class NO{  
    int dado;  
    NO esq;  
    NO dir;  
}
```

## Representação de Nó com ponteiros ...



# Percurso de Árvore Binária

Em Ordem

Pré Ordem

Pós Ordem

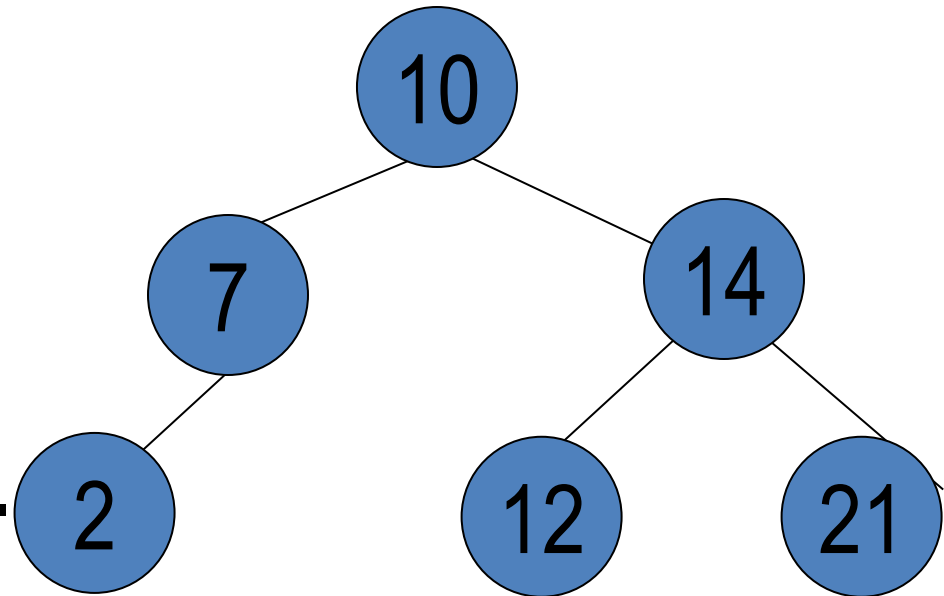
- EM Ordem (Ordem Simétrica)

R  
E  
C  
U  
R  
S  
I  
V  
O

1 – Percorremos a subárvore **esquerda** em ordem simétrica.

2 – Visitamos a **RAIZ**.

3 – Percorremos a subárvore **direita** em ordem simétrica.

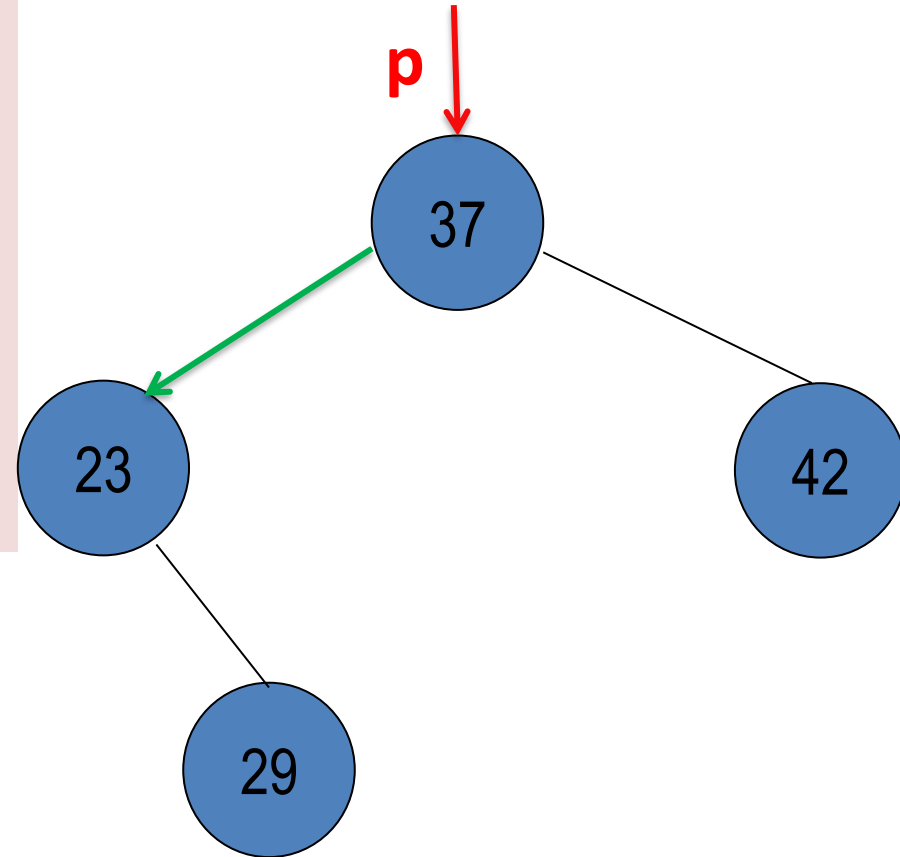


# Implementação Percorre Em Ordem

```
public static void inOrdem(NO p) {  
    if (p!=null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println("dado: " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}  
  
public static void main(String[] args) {  
    NO root = null;  
    // insere alguns nós...  
    inOrdem(root);  
    ...  
}
```

## ATIVACÃO 1

```
public static void inOrdem(NO p) {  
    if (p != null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println(" " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}
```

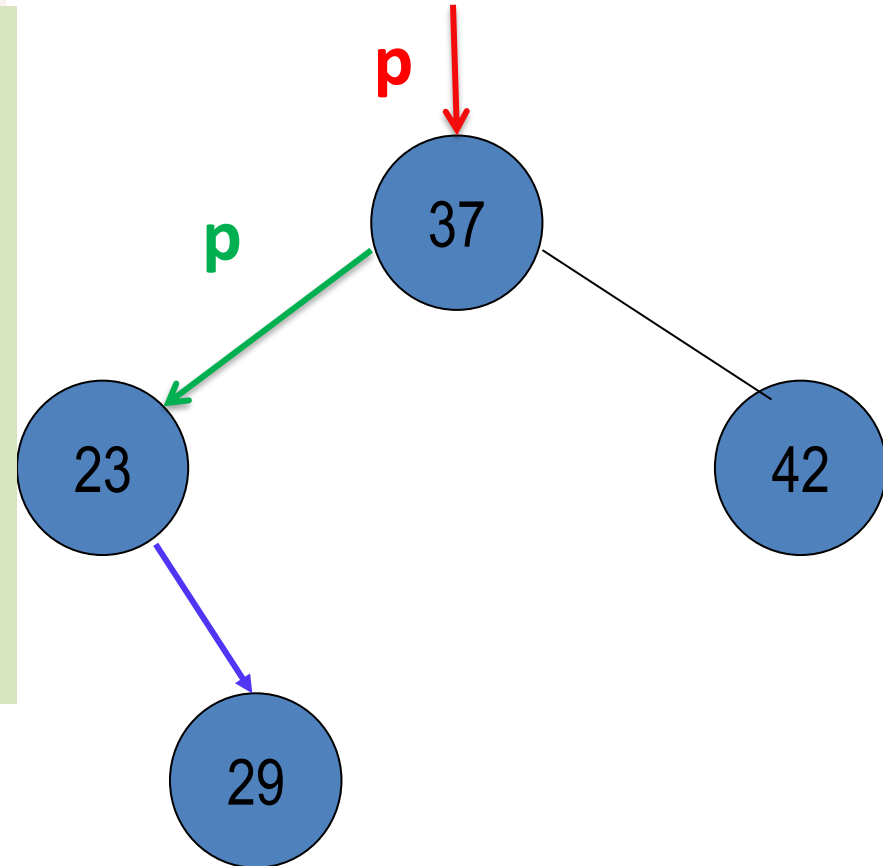


Tela de Saída:

## ATIVACÃO 1

## ATIVACÃO 2

```
public static void inOrdem(NO p) {  
    if (p != null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println(" " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}
```



Tela de Saída:

**23**

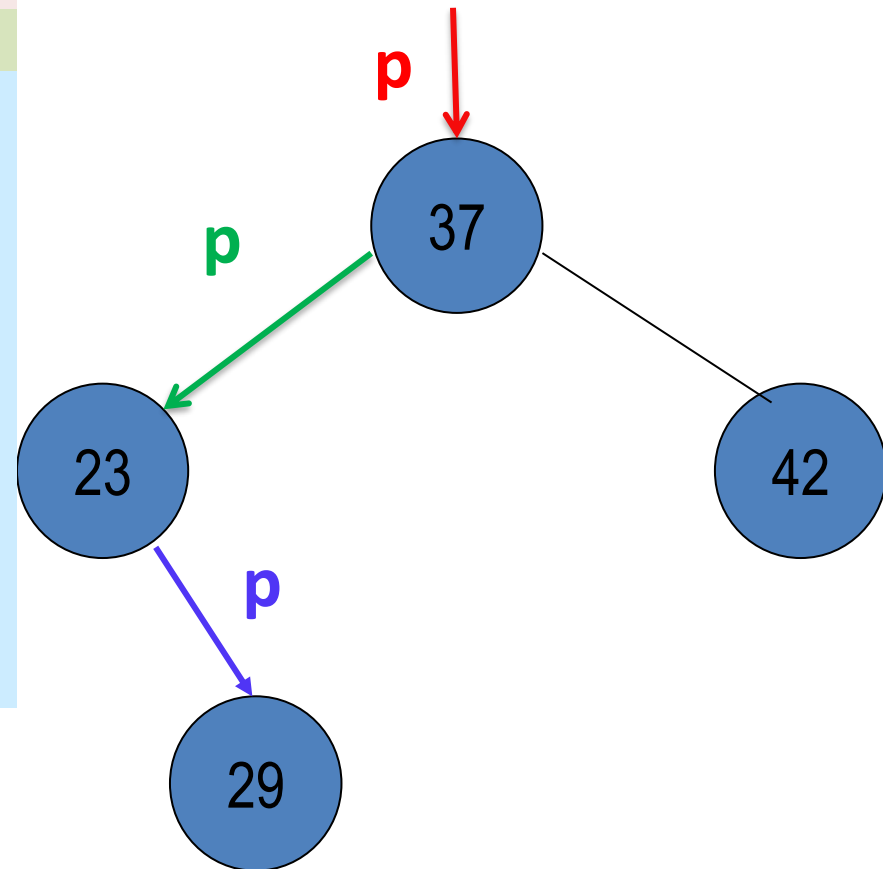


## ATIVACÃO 1

## ATIVACÃO 2

## ATIVACÃO 3

```
public static void inOrdem(NO p) {  
    if (p != null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println(" " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}
```



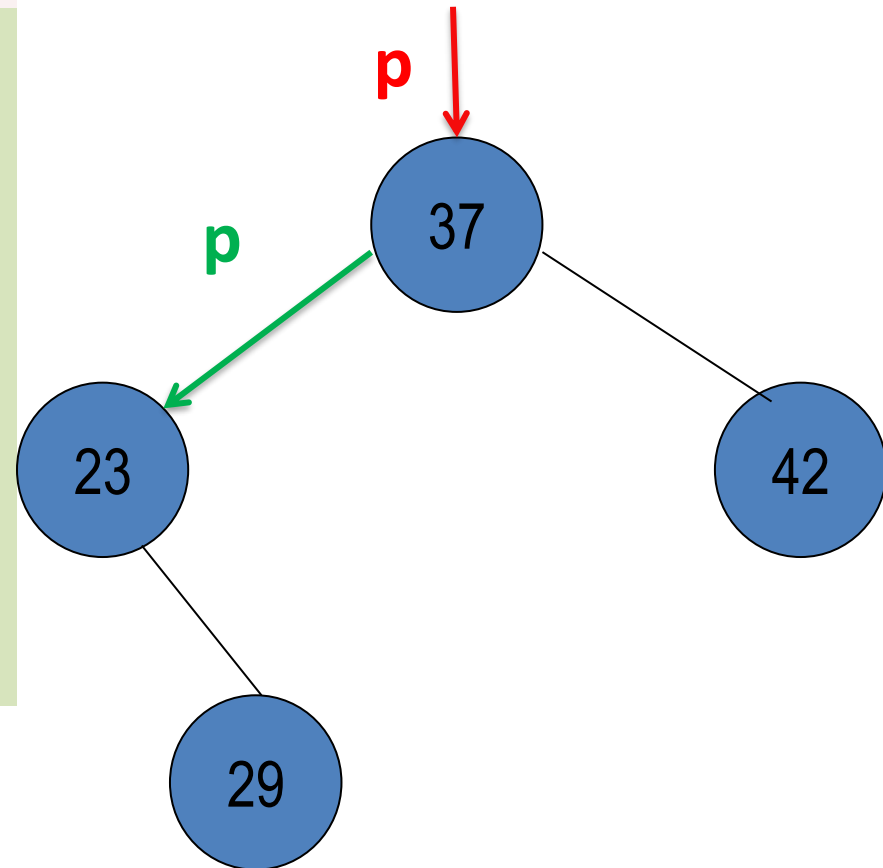
Tela de Saída:

**23 29**

## ATIVACÃO 1

## ATIVACÃO 2

```
public static void inOrdem(NO p) {  
    if (p != null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println(" " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}
```

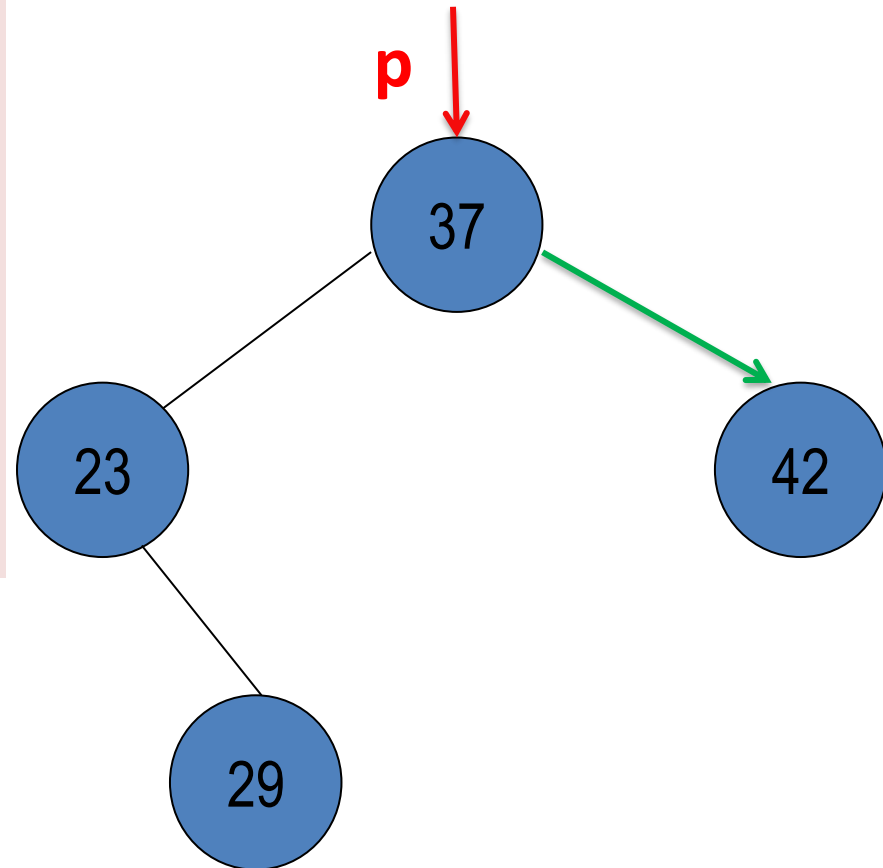


Tela de Saída:

**23 29**

## ATIVACÃO 1

```
public static void inOrdem(NO p) {  
    if (p != null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println(" " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}
```



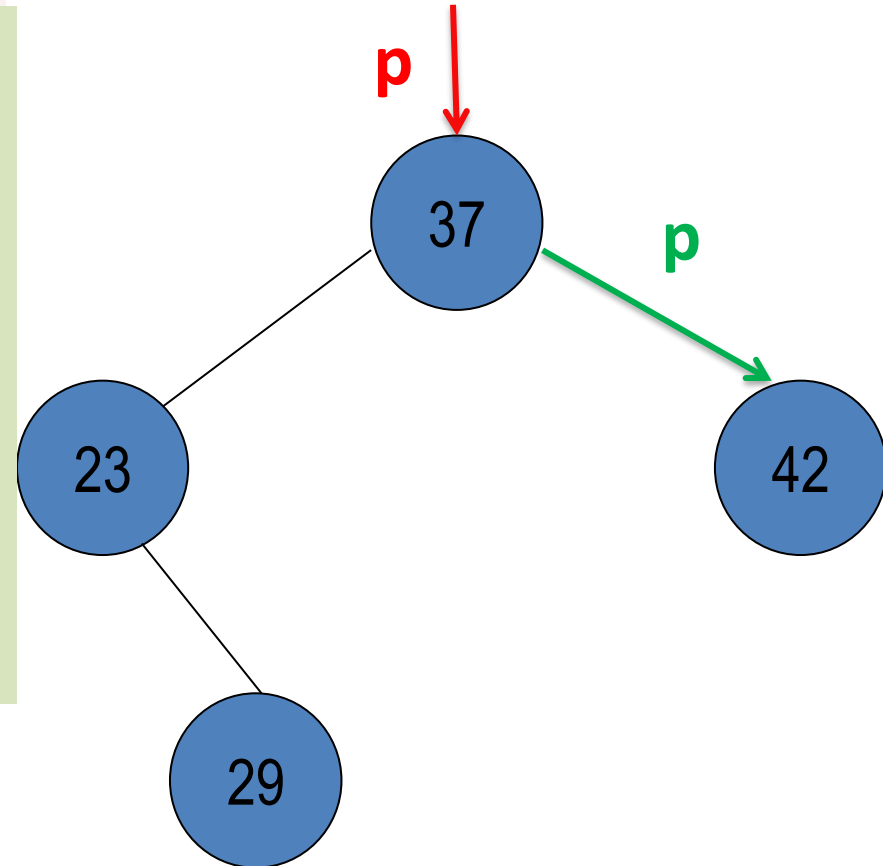
Tela de Saída:

**23 29 37**

## ATIVACÃO 1

## ATIVACÃO 2

```
public static void inOrdem(NO p) {  
    if (p != null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println(" " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}
```

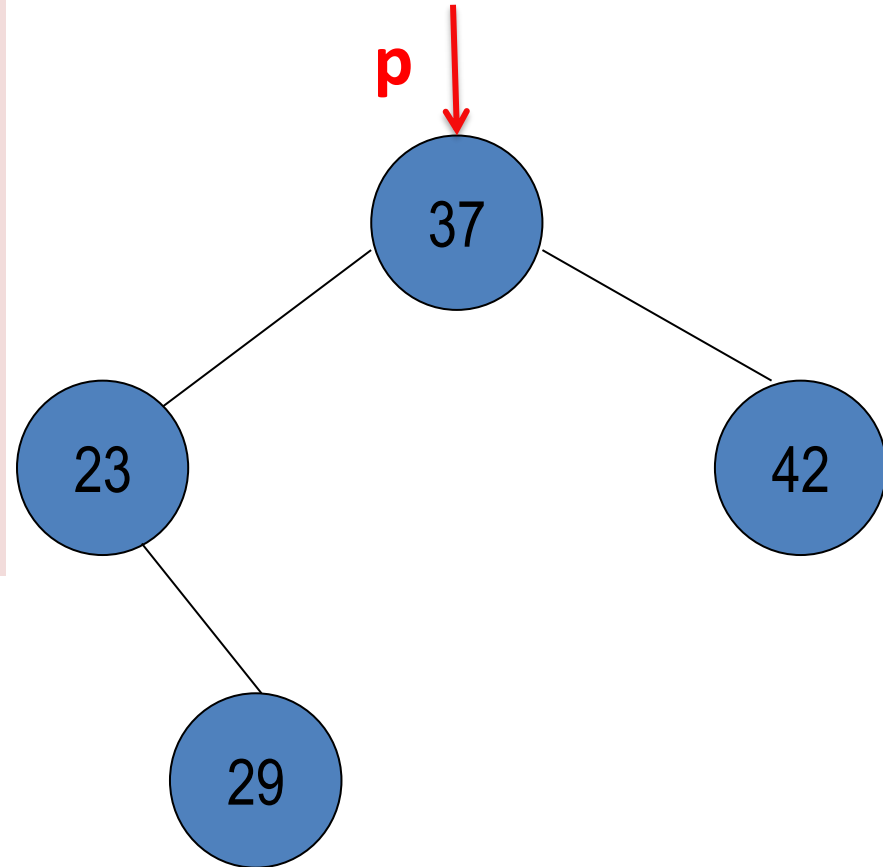


Tela de Saída:

**23 29 37 42**

## ATIVACÃO 1

```
public static void inOrdem(NO p) {  
    if (p != null) {  
        if (p.esq != null)  
            inOrdem(p.esq);  
        System.out.println(" " + p.dado);  
        if (p.dir != null)  
            inOrdem(p.dir);  
    }  
}
```



Tela de Saída:

**23 29 37 42**

**23 29 37 42**

Como a árvore apresentada é uma ABB o percurso em ordem apresenta valores em ordem crescente

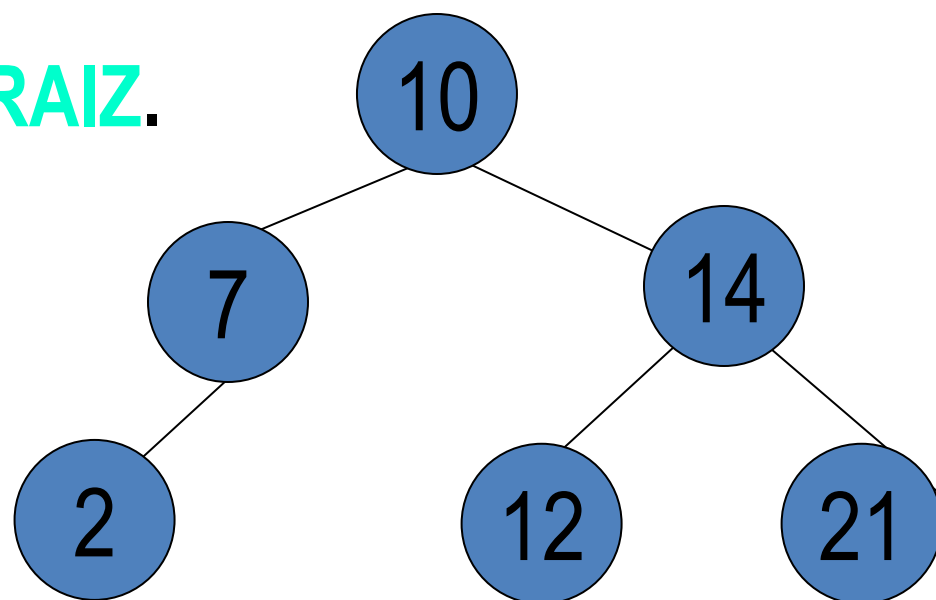
- Pré-Ordem (Percurso em Profundidade)

R  
E  
C  
U  
R  
S  
I  
V  
O

1 – Apresentamos a **RAIZ**.

2 – Percorremos a subárvore **esquerda** em ordem prévia.

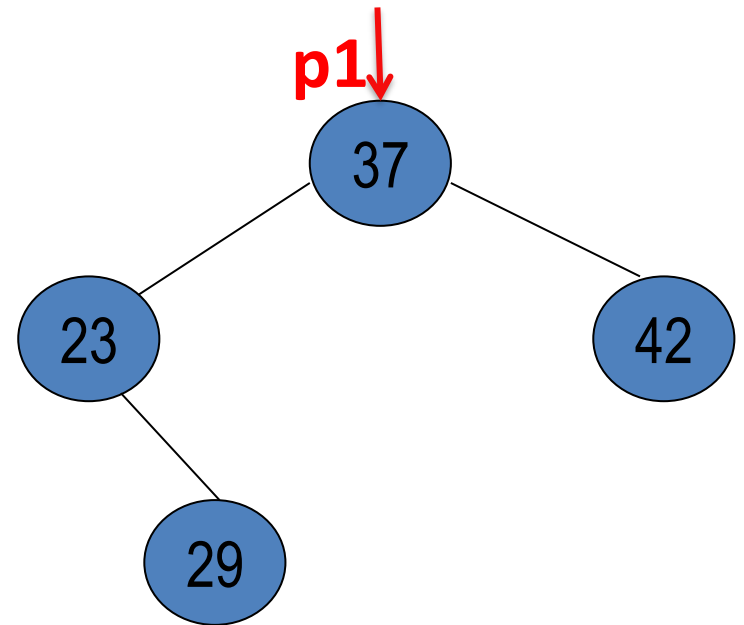
3 – Percorremos a subárvore **direita** em ordem prévia.



# Percurso de arvore Binária Pré Ordem FIAP

```
public static void preOrdem(NO p) {  
    if (p!=null) {  
        System.out.println("dado: " + p.dado);  
        if (p.esq != null)  
            preOrdem(p.esq);  
  
        if (p.dir != null)  
            preOrdem(p.dir);  
    }  
}
```

37 23 29 42





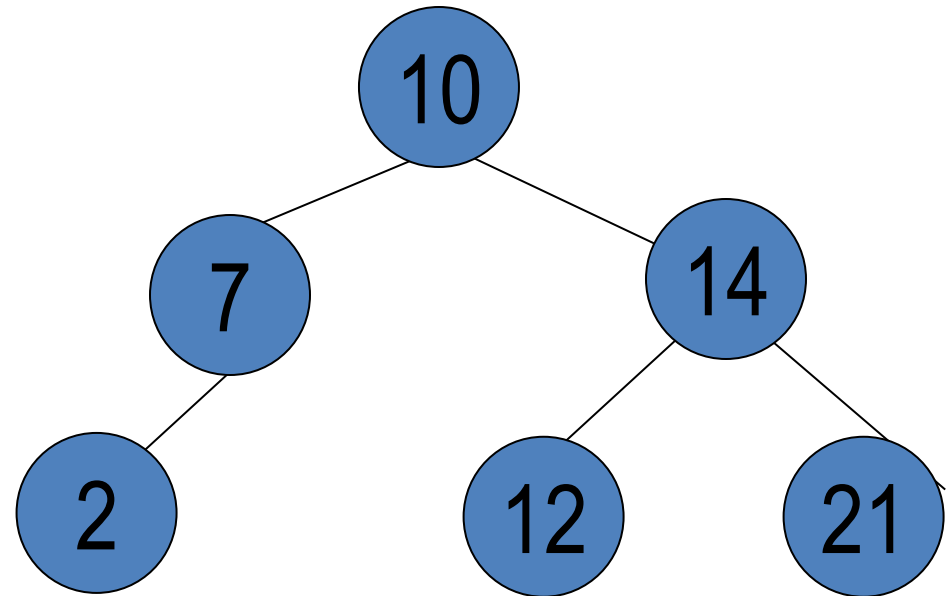
- Pós-Ordem

R  
E  
C  
U  
R  
S  
I  
V  
O

1 – Percorremos a subárvore **esquerda** em ordem posterior.

2 – Percorremos a subárvore **direita** em ordem posterior.

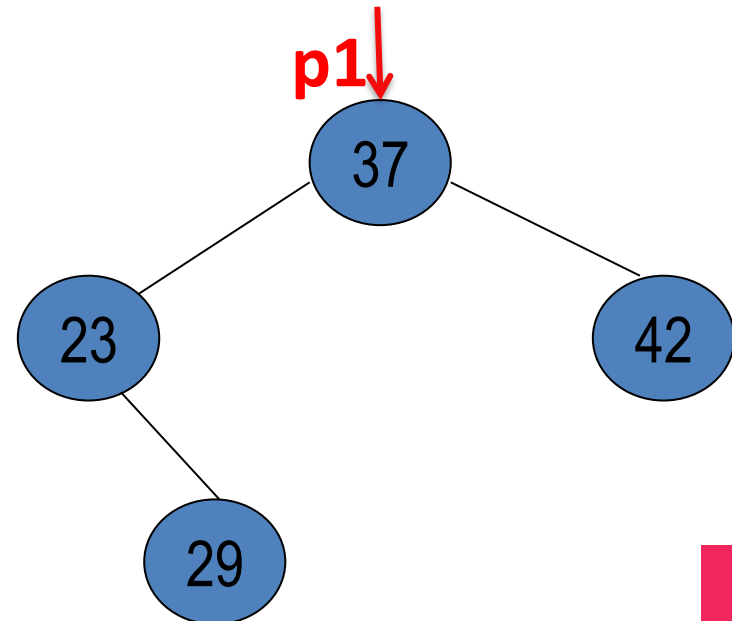
3 – Apresentamos a **RAIZ**.



# Percurso de árvore Binária Pós Ordem FIAP

```
public static void posOrdem(NO p) {  
    if (p!=null) {  
        if (p.esq != null)  
            posOrdem(p.esq);  
        if (p.dir != null)  
            posOrdem(p.dir);  
        System.out.println("dado: " + p.dado);  
    }  
}
```

29 23 42 37

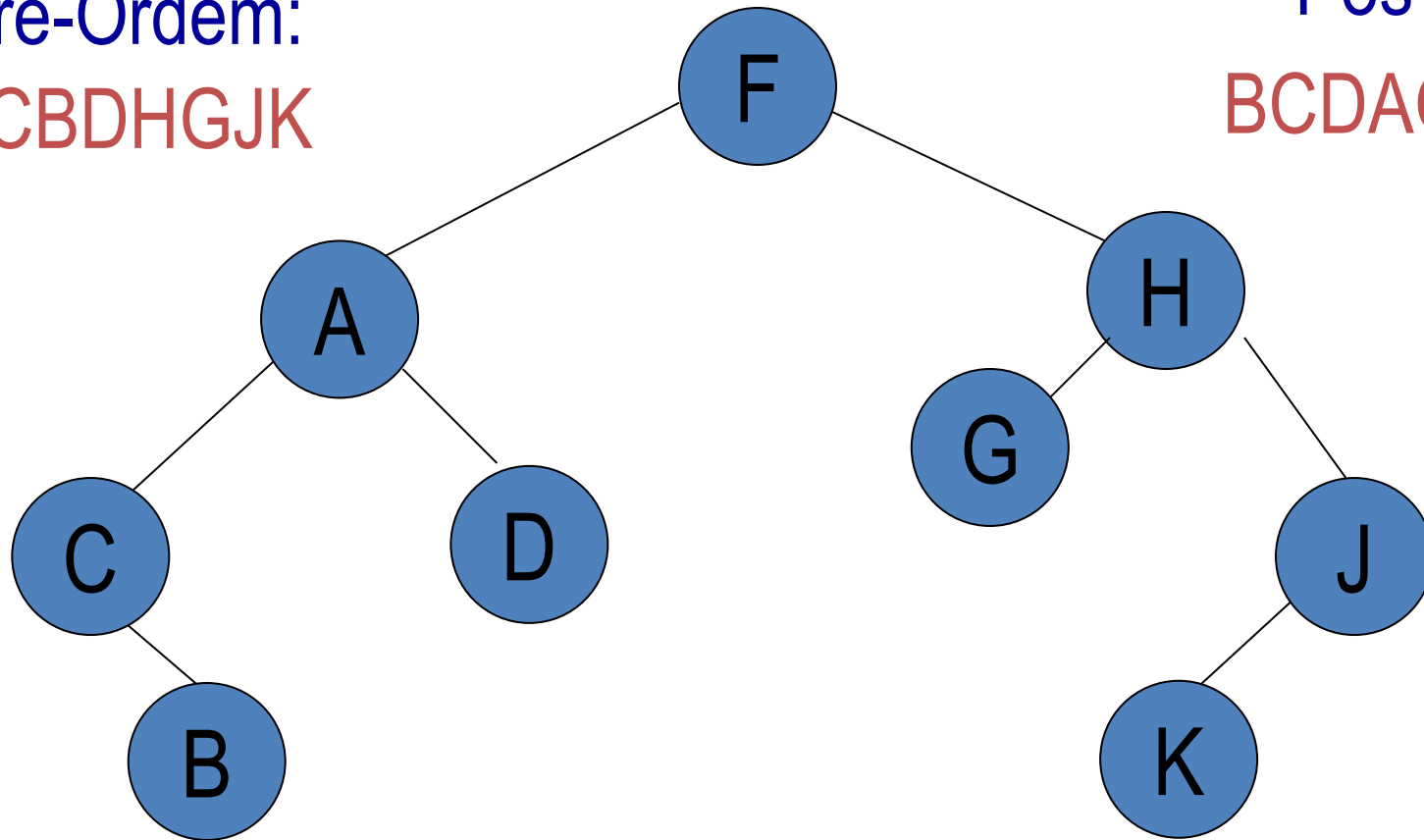


# Árvores Binárias – Exercício: Percurso

FIAP

Pré-Ordem:  
FACBDHGJK

Pós-Ordem:  
BCDAGKJHF



Em Ordem: CBADFGHKJ

# REFERÊNCIAS



- TENENBAUM, A.M. E outros - **Estruturas de Dados usando C**. Makron Books do Brasil Editora Ltda, SP.
- PEREIRA, S. L. **Estrutura de Dados Fundamentais**. São Paulo: Érica.
- FORBELLONE, A.L.V. & EBERSPÄCHER, H.F. – **Lógica de Programação: A Construção de Algoritmos e Estruturas de Dados**. Makron Books, São Paulo, SP
- ASCENCIO, A.F.G e ARAÚJO, G.S. – Estruturas de Dados: Algoritmos, Análise da Complexidade e Implementação em JAVA e C/C++

Copyright © 2022  
Profa. Patrícia Magna

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, dos professores.