

# **Unidade VI:**

# **Árvore Binária**

**Prof. Max do Val Machado**



**PUC Minas**

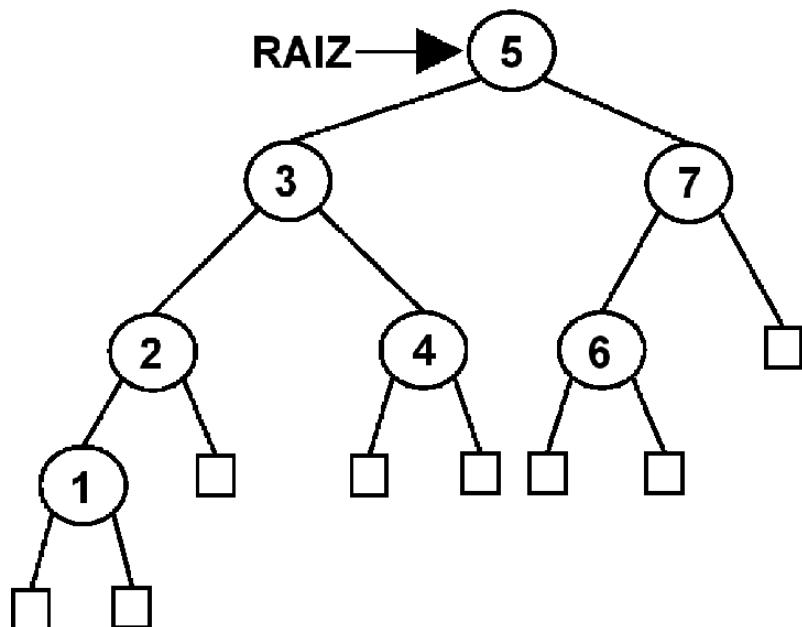
Instituto de Ciências Exatas e Informática  
Curso de Ciência da Computação

# Introdução

- A inserção, remoção e pesquisa nas estruturas de dados lineares (pilhas, filas e listas) têm custo de  $\Theta(n)$  comparações
- O custo de pesquisa em listas estáticas ordenadas é de  $\Theta(\lg(n))$  comparações. Contudo, as operações de inserção e remoção não são tão eficientes

## Árvore

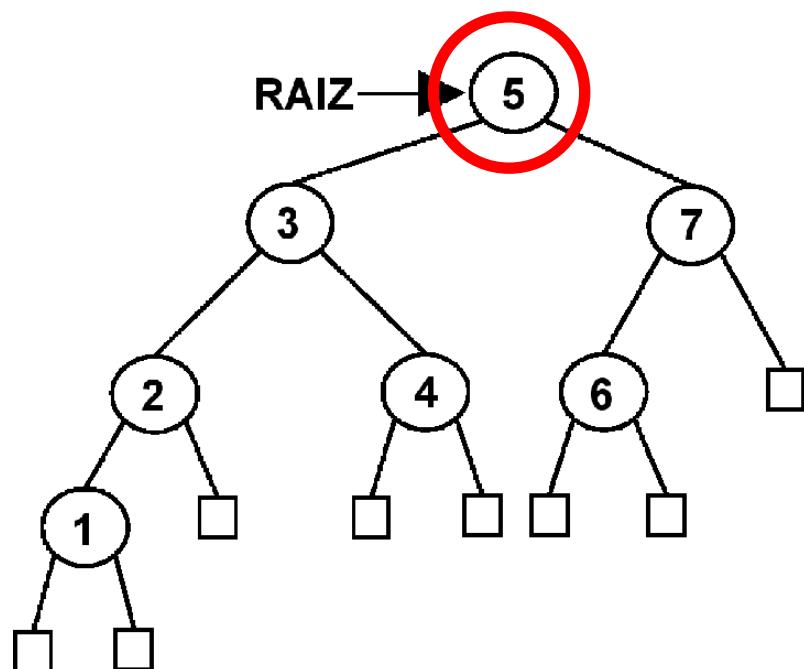
- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outros de arcos (arestas) que conectam os vértices



## Árvore

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outros de arcos (arestas) que conectam os vértices

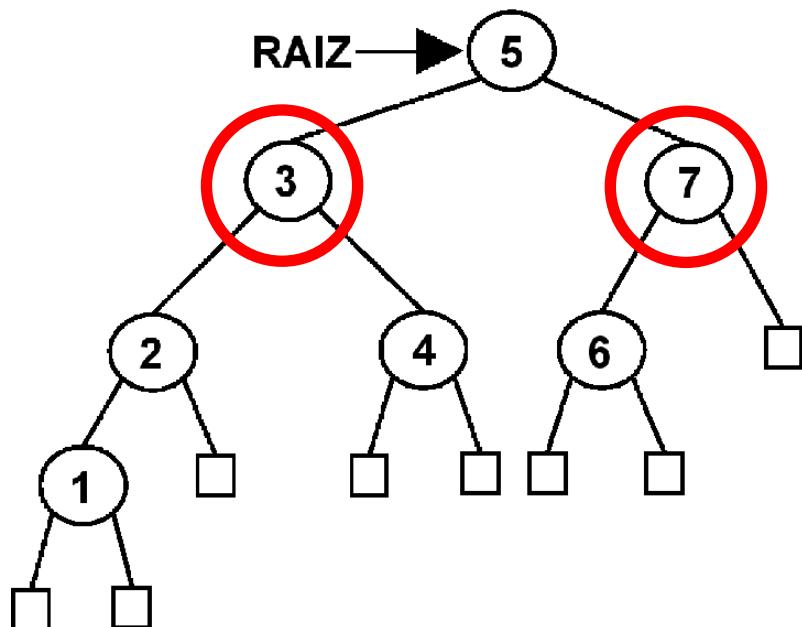
O nó 5 é denominado nó raiz e ele está no nível 0



## Árvore

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outros de arcos (arestas) que conectam os vértices

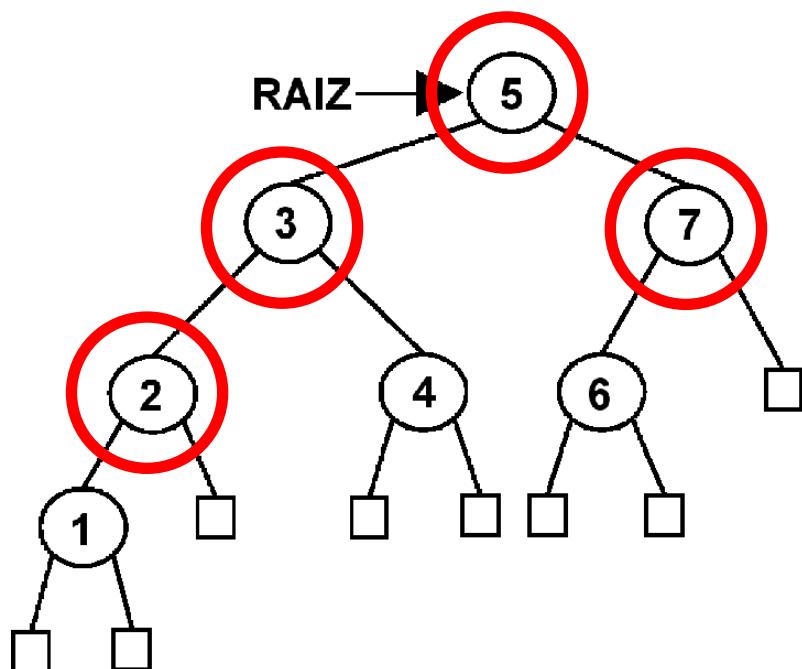
Os nós 3 e 7 são filhos do 5 e esse é pai dos dois primeiros



## Árvore

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outros de arcos (arestas) que conectam os vértices

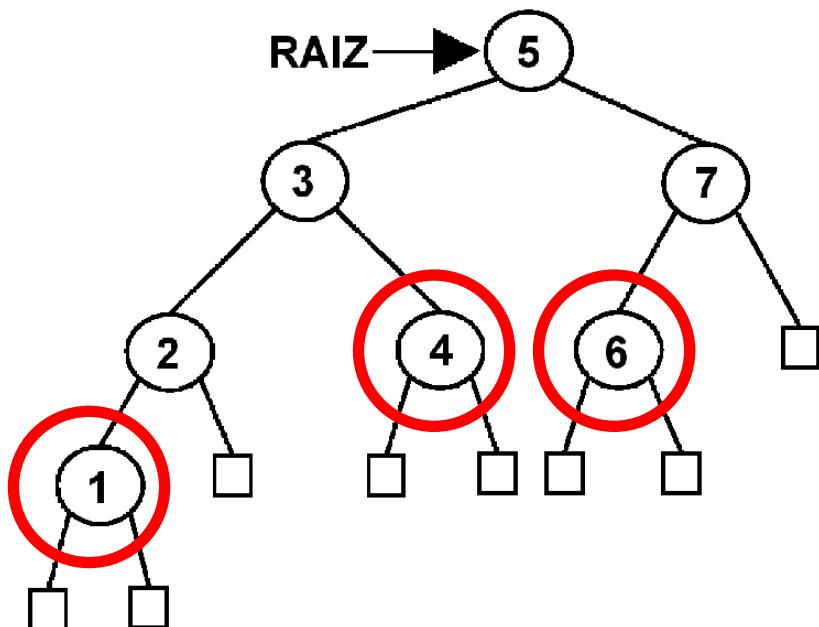
Um nó com filho(s) é chamado de **nó interno** e outro sem, de folha



## Árvore

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outros de arcos (arestas) que conectam os vértices

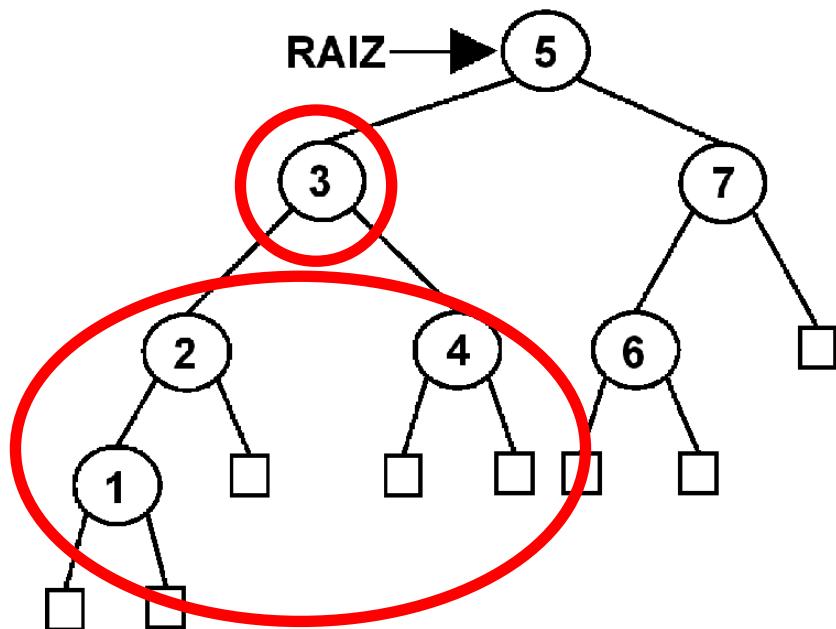
Um nó com filho(s) é chamado de nó interno e outro sem, de **folha**



## Árvore

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outros de arcos (arestas) que conectam os vértices

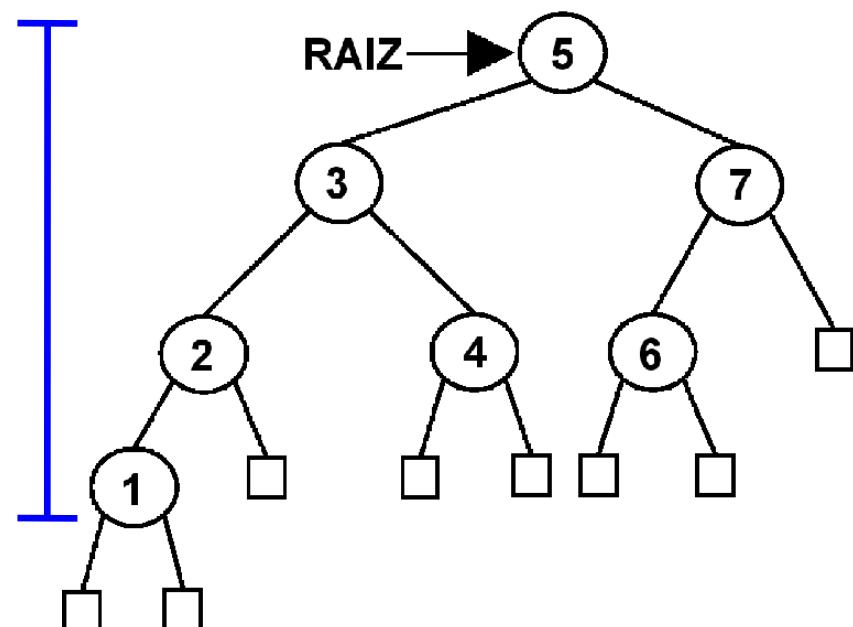
Os nós 1, 2 e 4 formam uma subárvore com raiz no nó 3



## Árvore

- Estrutura de dados cujas operações de inserção, remoção e substituição possuem a mesma eficiência
- Estrutura de dados que contém um conjunto finito de vértices (nós) e outros de arcos (arestas) que conectam os vértices

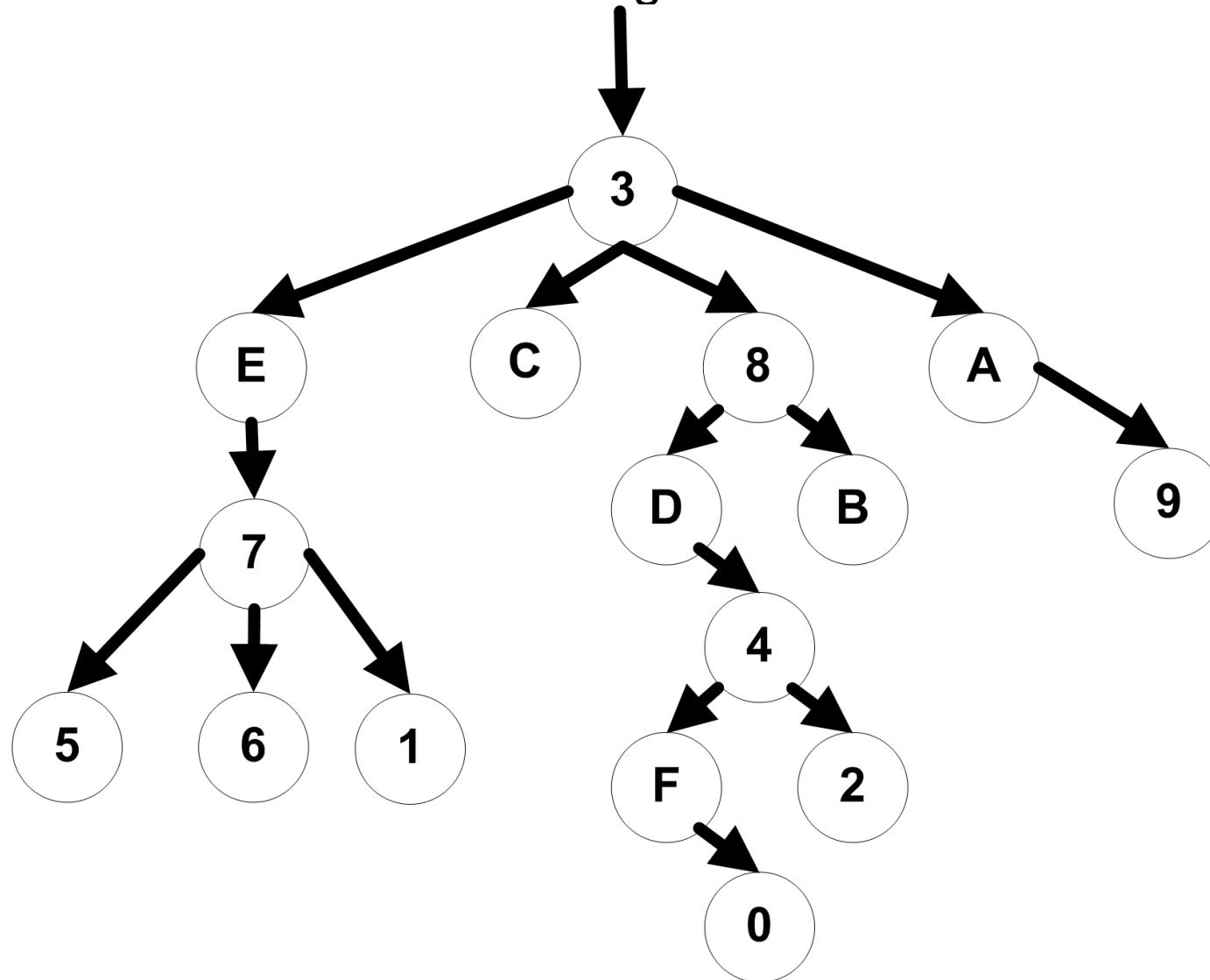
A altura de uma árvore é a maior distância de um nó em relação à raiz. Nesse caso, a altura será 3



# Árvore

- Exercício: Crie uma árvore com os dígitos hexadecimais

- Exercício: Crie uma árvore com os dígitos hexadecimais



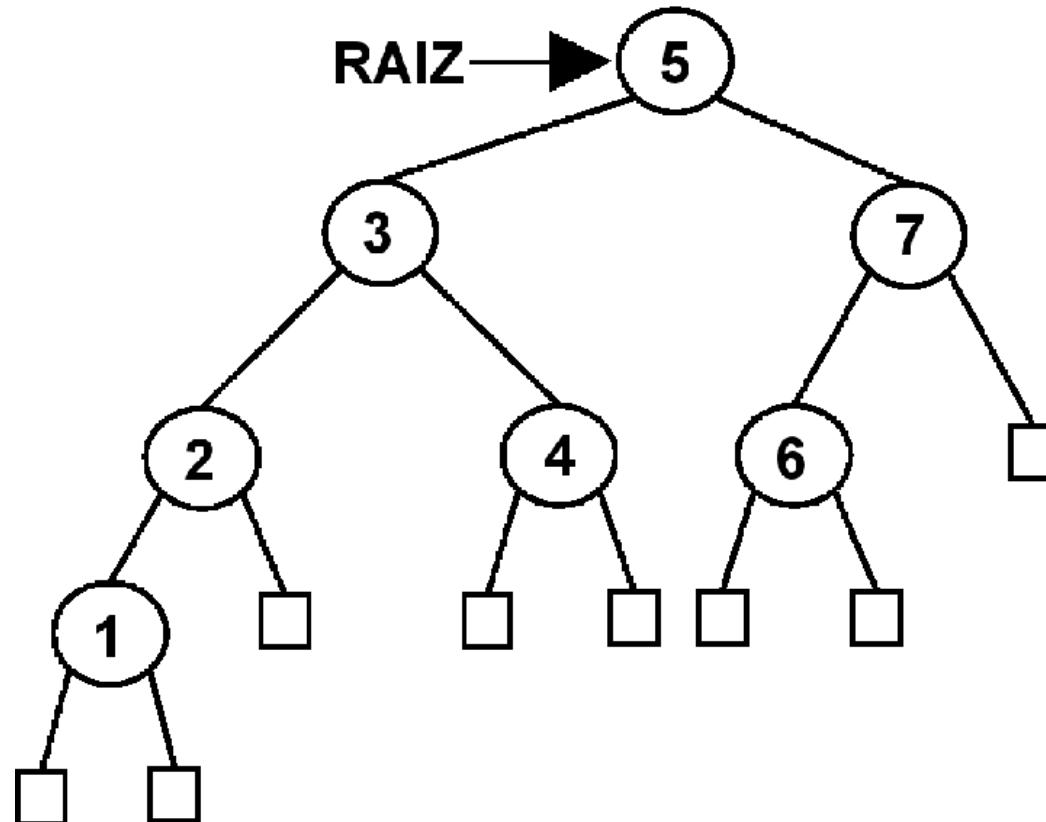
# Árvore Binária

- Árvore em que cada nó possui no máximo dois filhos

# Árvore Binária

- Árvore em que cada nó possui no máximo dois filhos

- Nosso primeiro exemplo é uma árvore binária

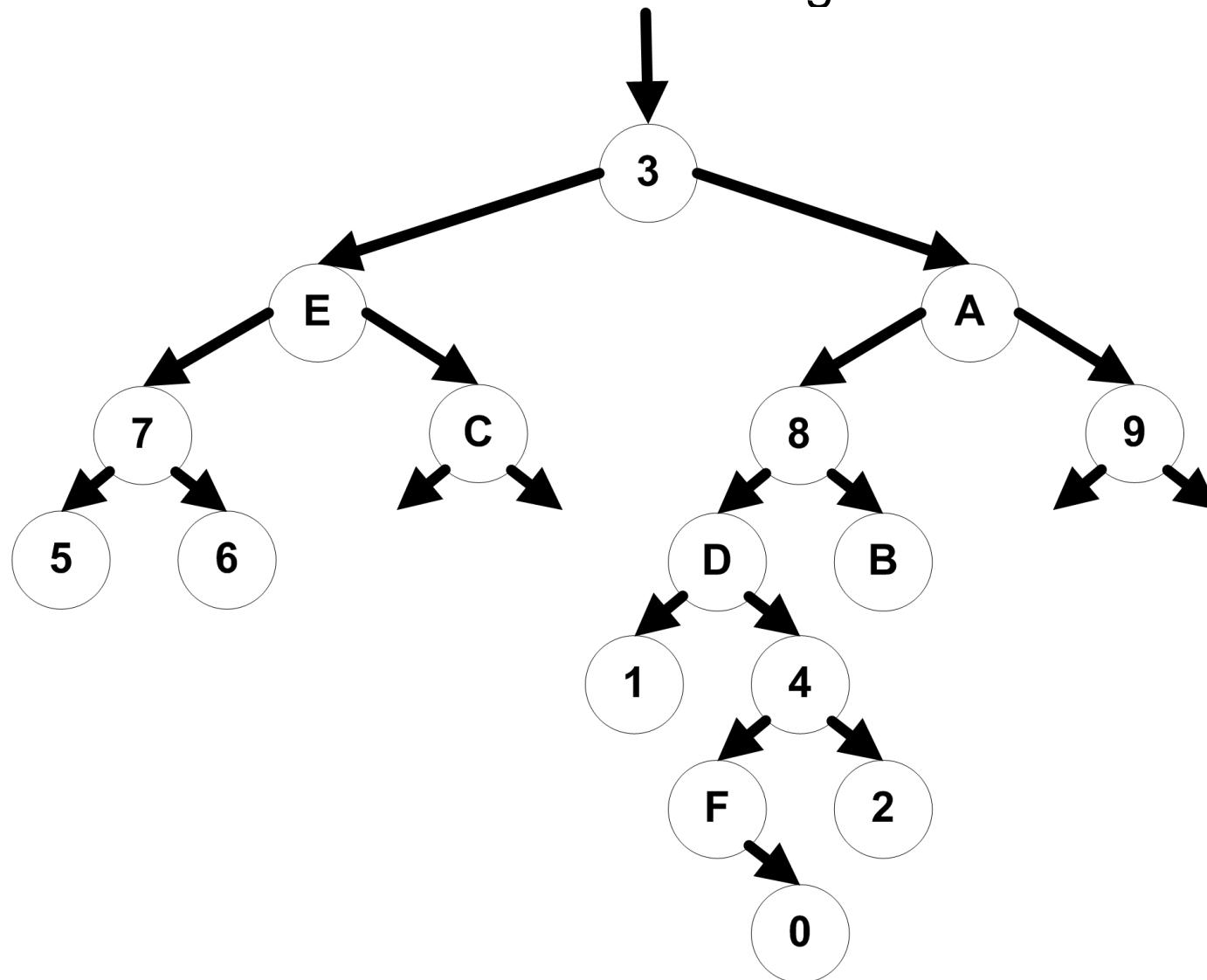


# Árvore Binária

- Exercício: Crie uma árvore binária com os dígitos hexadecimais

## Árvore Binária

- Exercício: Crie uma árvore binária com os dígitos hexadecimais



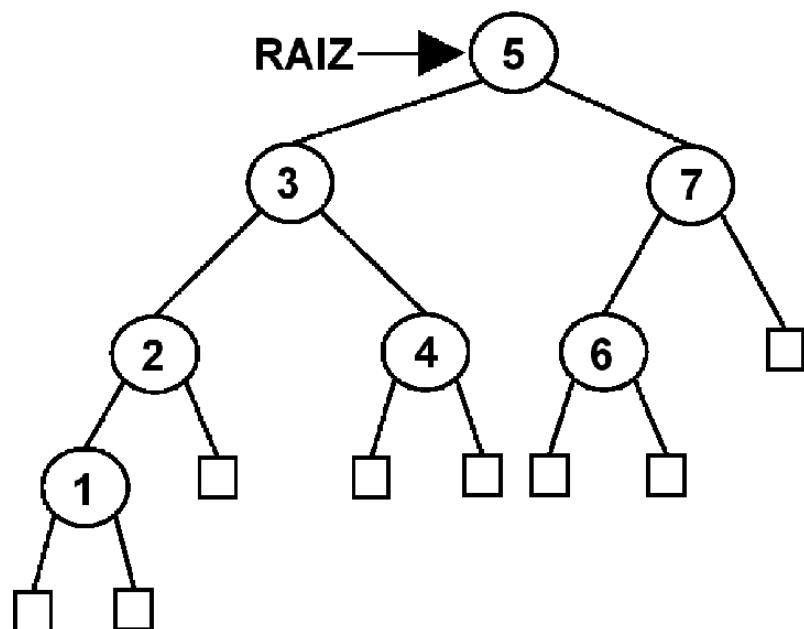
# Árvore Binária de Pesquisa

- Árvore binária em que cada nó é:
  - maior que todos seus vizinhos à esquerda
  - menor que todos seus vizinhos à direita

# Árvore Binária de Pesquisa

- Árvore binária em que cada nó é:
  - maior que todos seus vizinhos à esquerda
  - menor que todos seus vizinhos à direita

Nosso exemplo é uma árvore binária de pesquisa

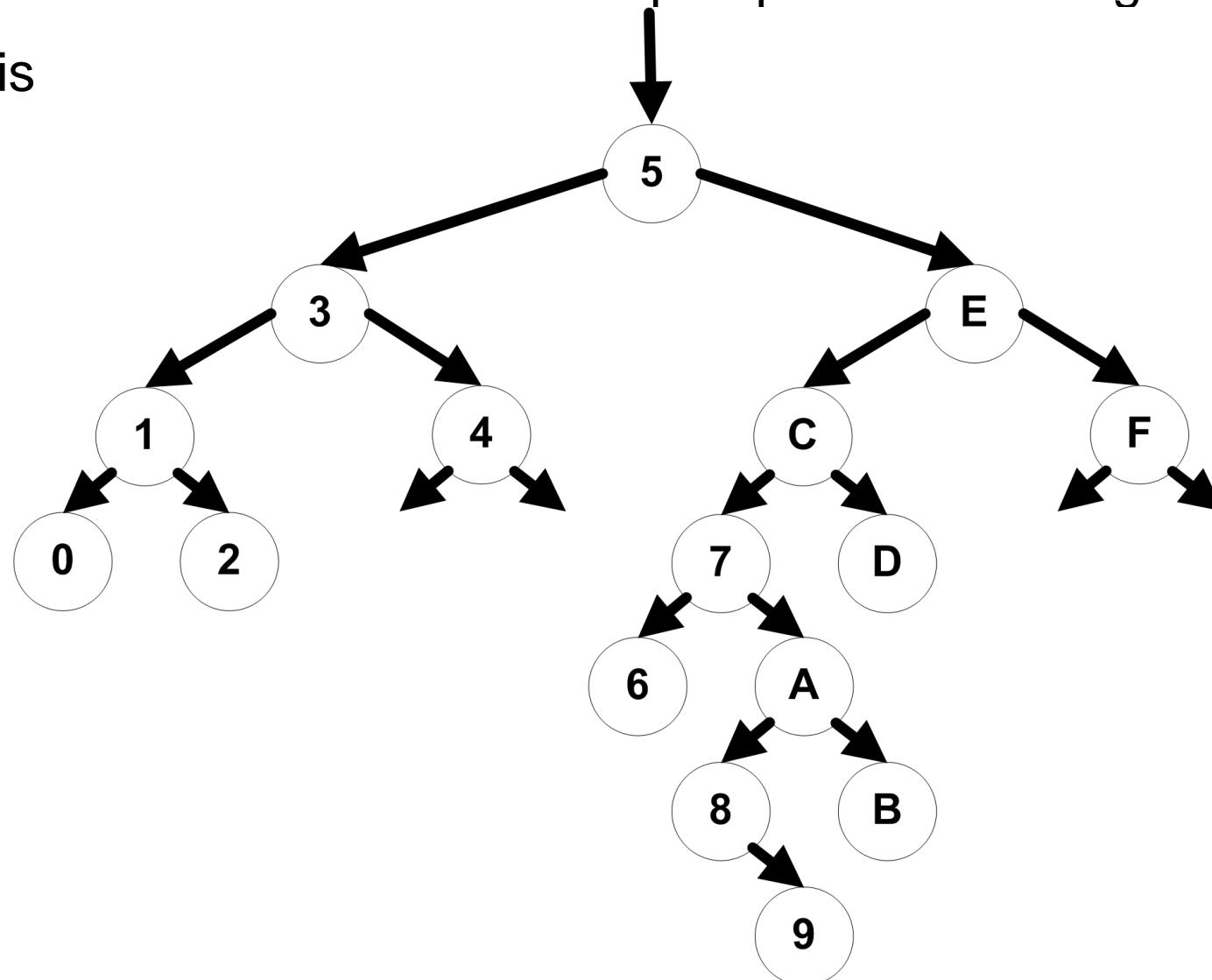


# Árvore Binária de Pesquisa

- Exercício: Crie uma árvore binária de pesquisa com os dígitos hexadecimais

# Árvore Binária de Pesquisa

- Exercício: Crie uma árvore binária de pesquisa com os dígitos hexadecimais



# Árvore Binária Completa

- Árvore binária em que:
  - Cada nó é uma folha OU possui exatamente dois filhos
  - Todos os nós folhas possuem uma altura  $h$
  - O número de nós internos é  $2^h - 1$
  - O número de nós folhas é  $2^h$
  - O número total de nós é  $2^h - 1 + 2^h = 2^{(h+1)} - 1$

# Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais

# Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais

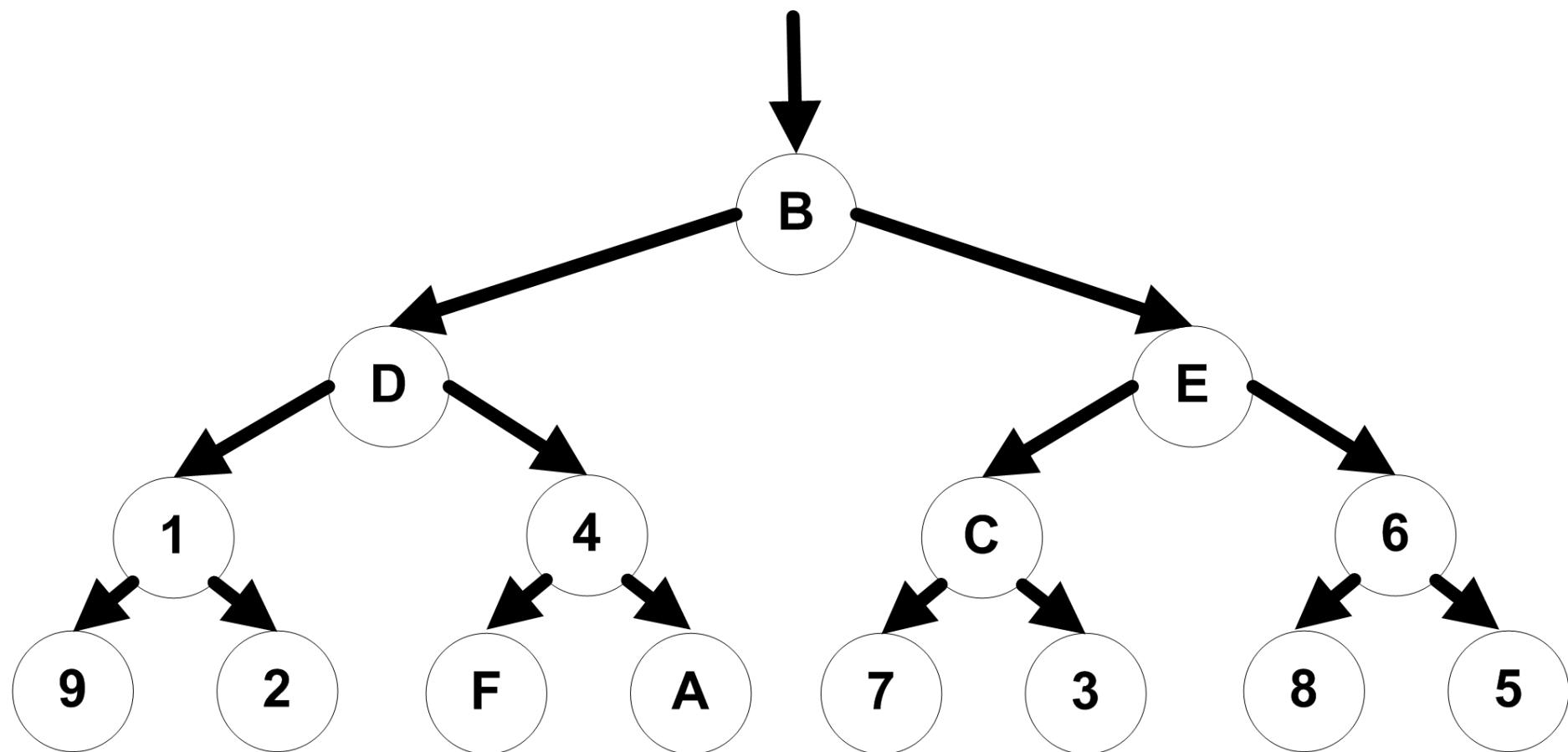
Impossível por que o número de nós não é uma potência de dois (está sobrando um nó)

# Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos

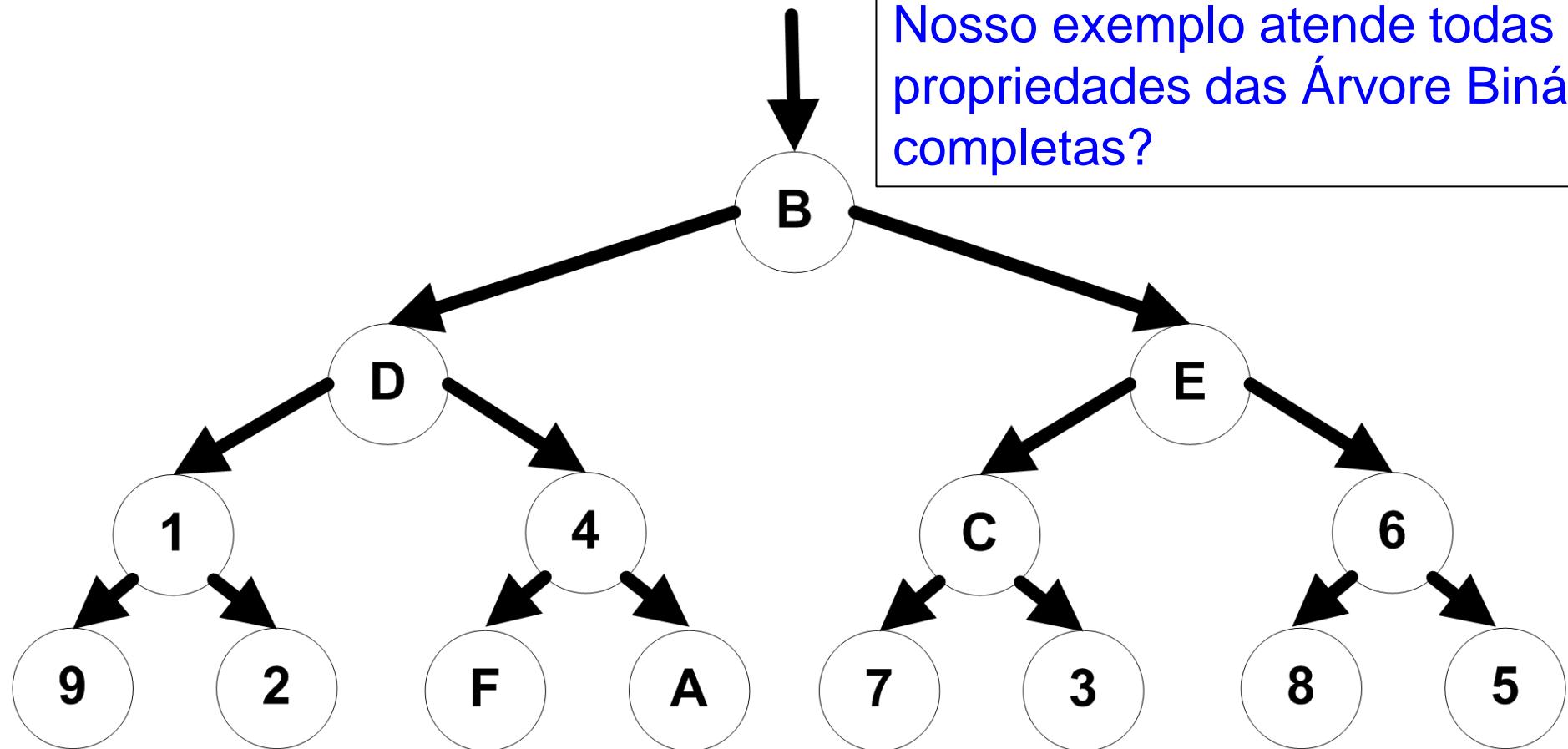
## Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos



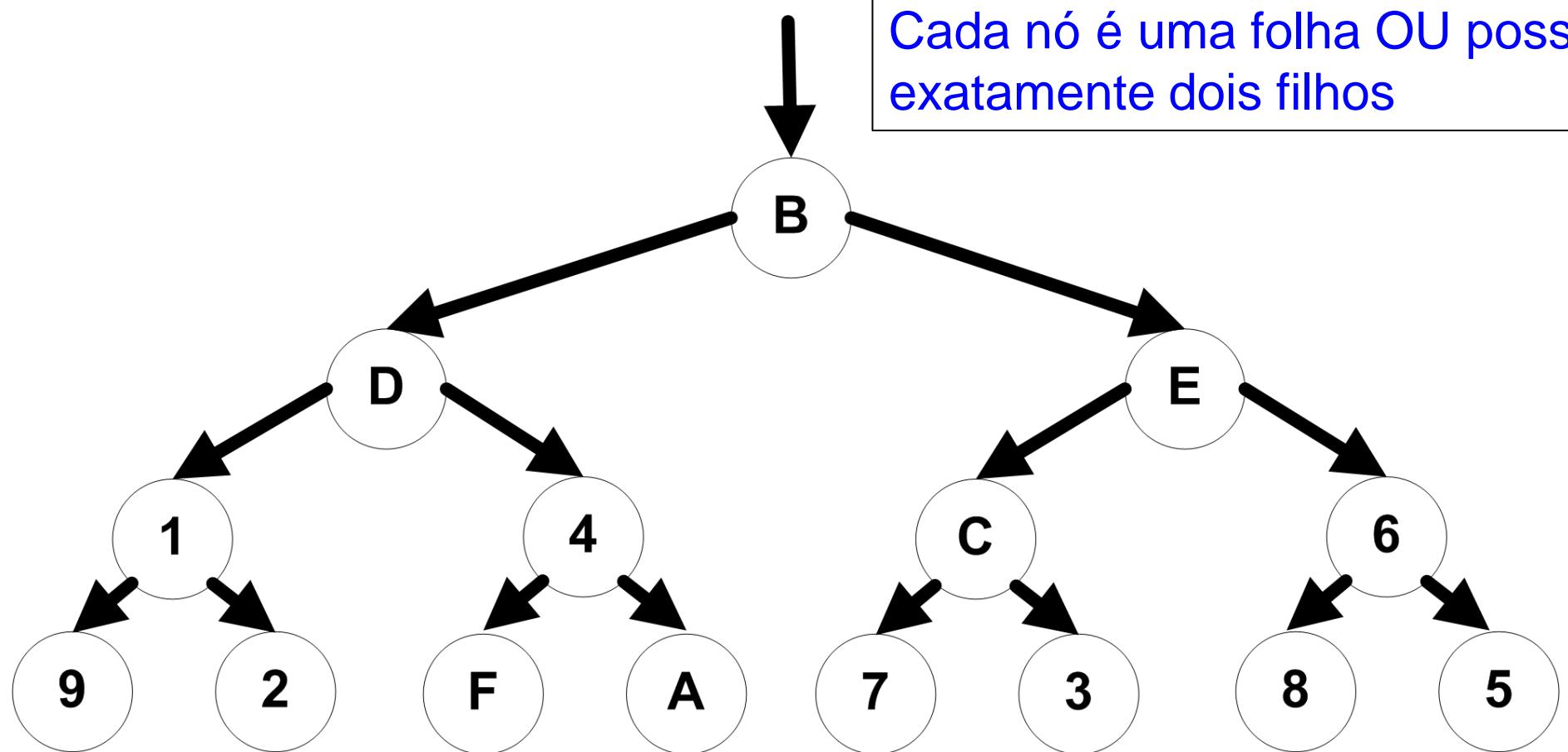
# Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos



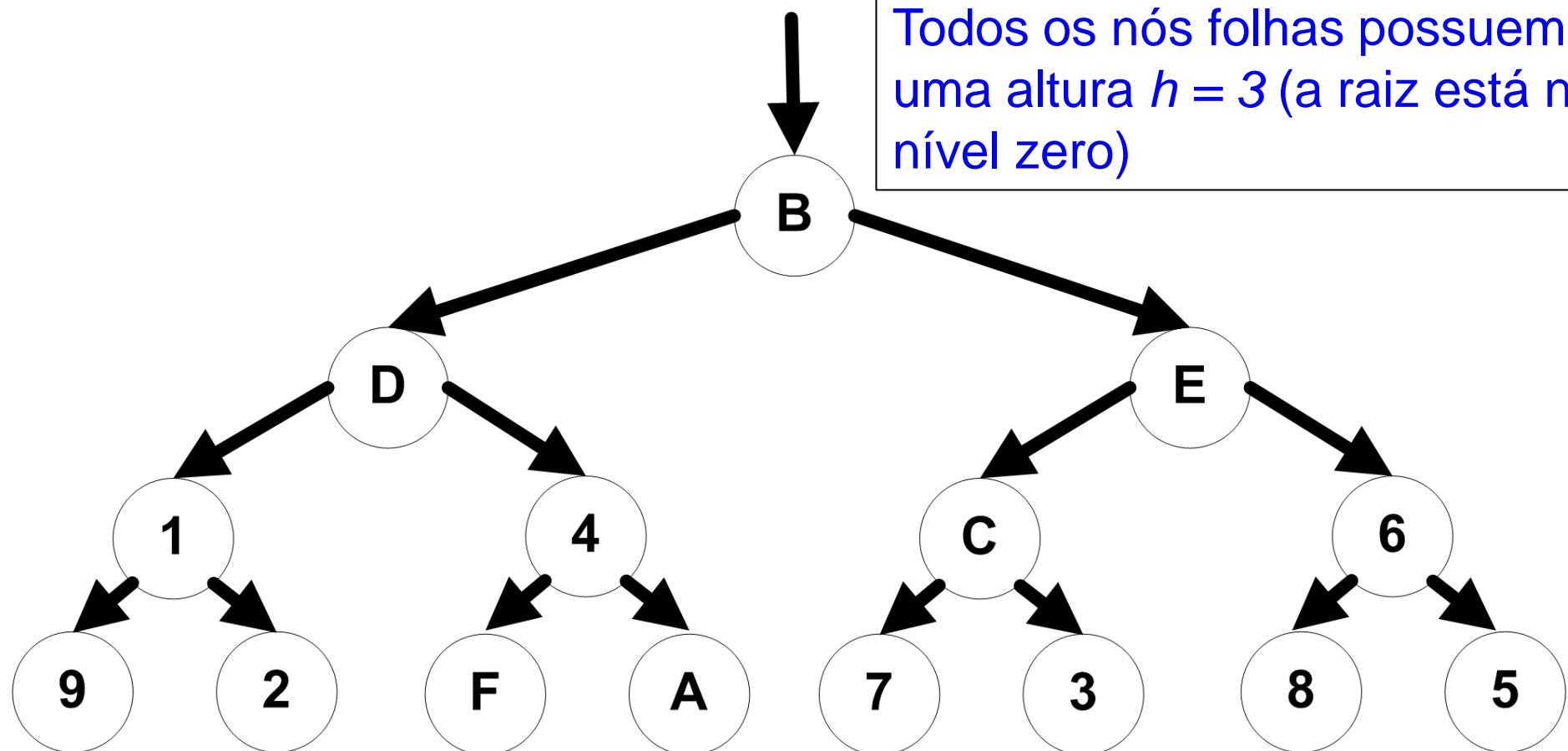
# Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos



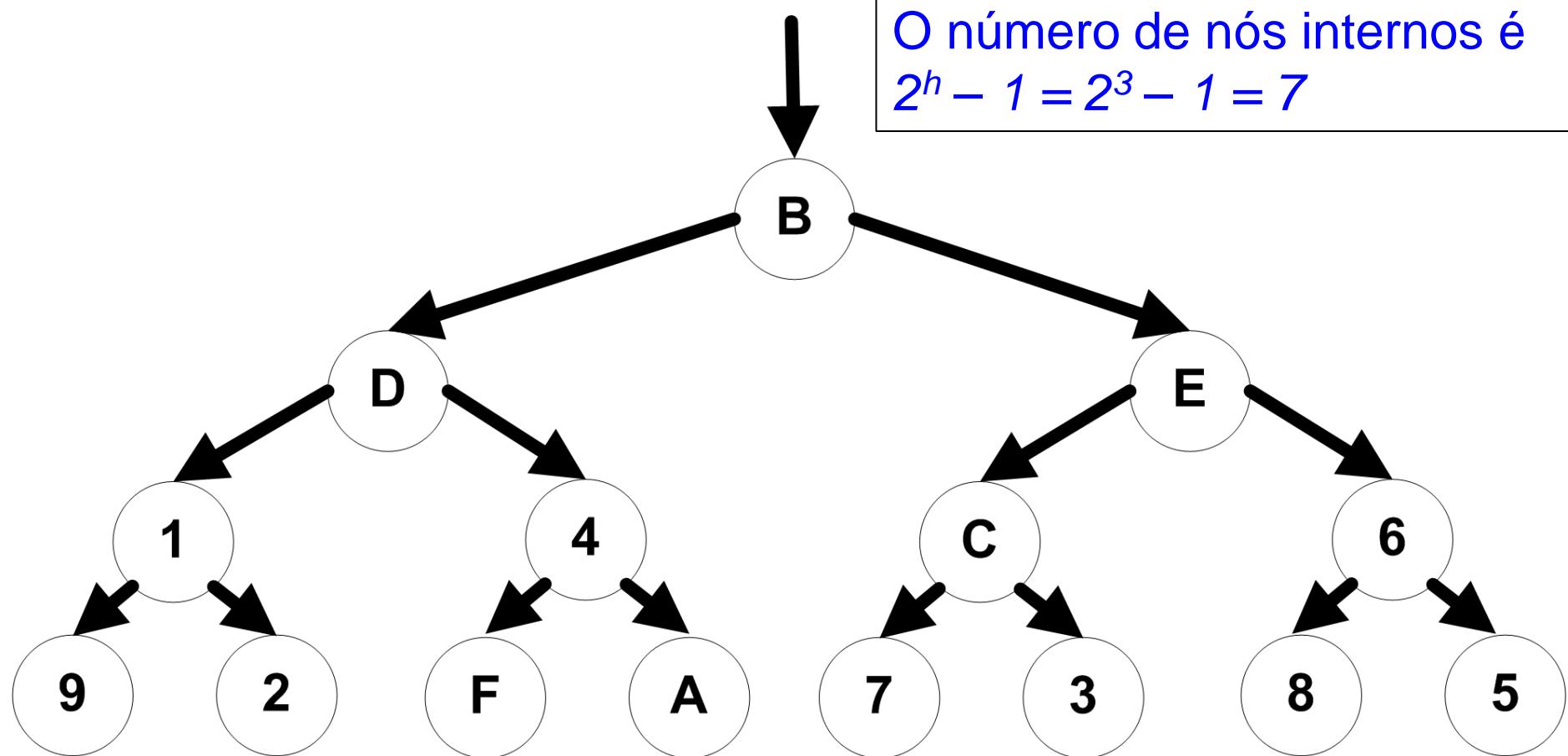
## Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos



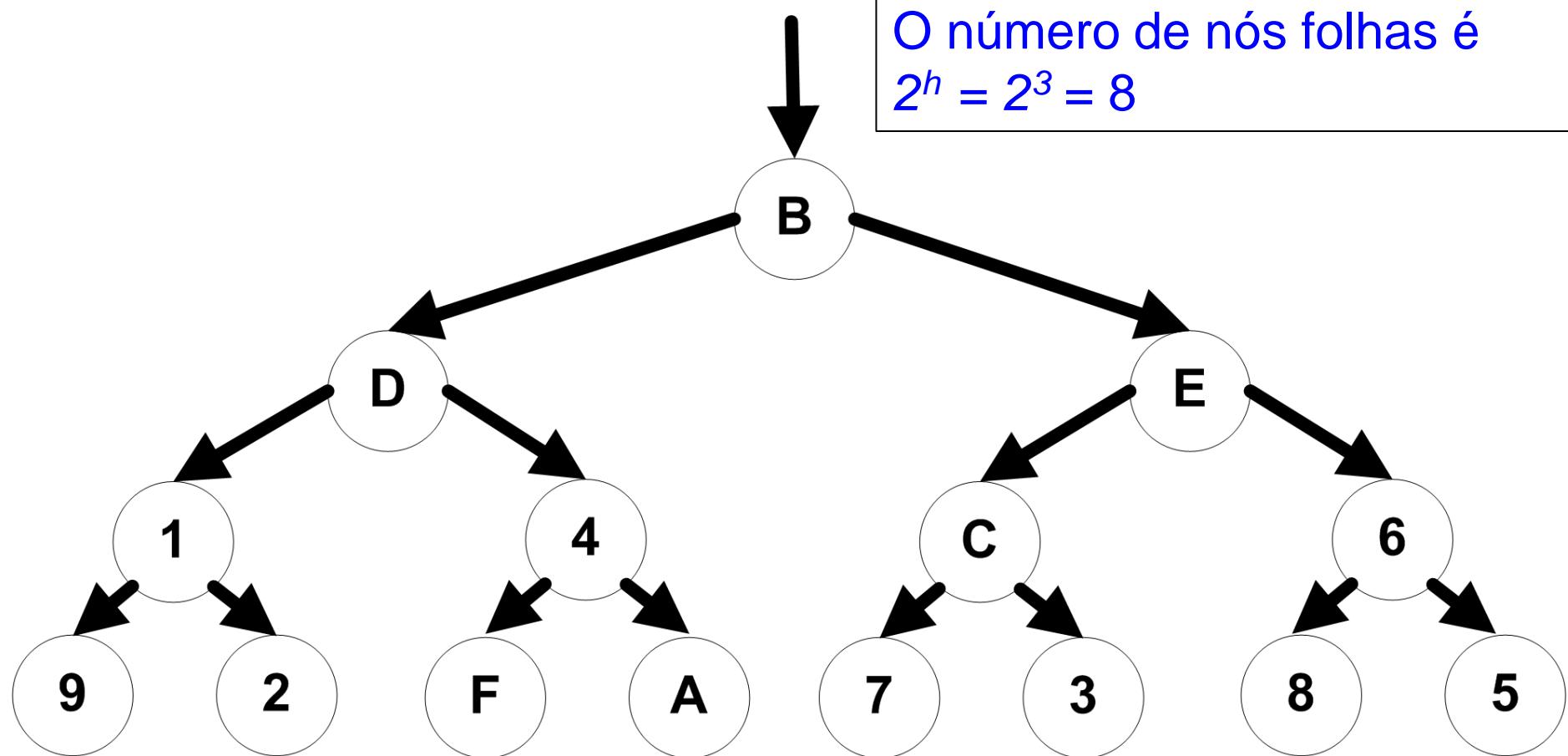
## Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos



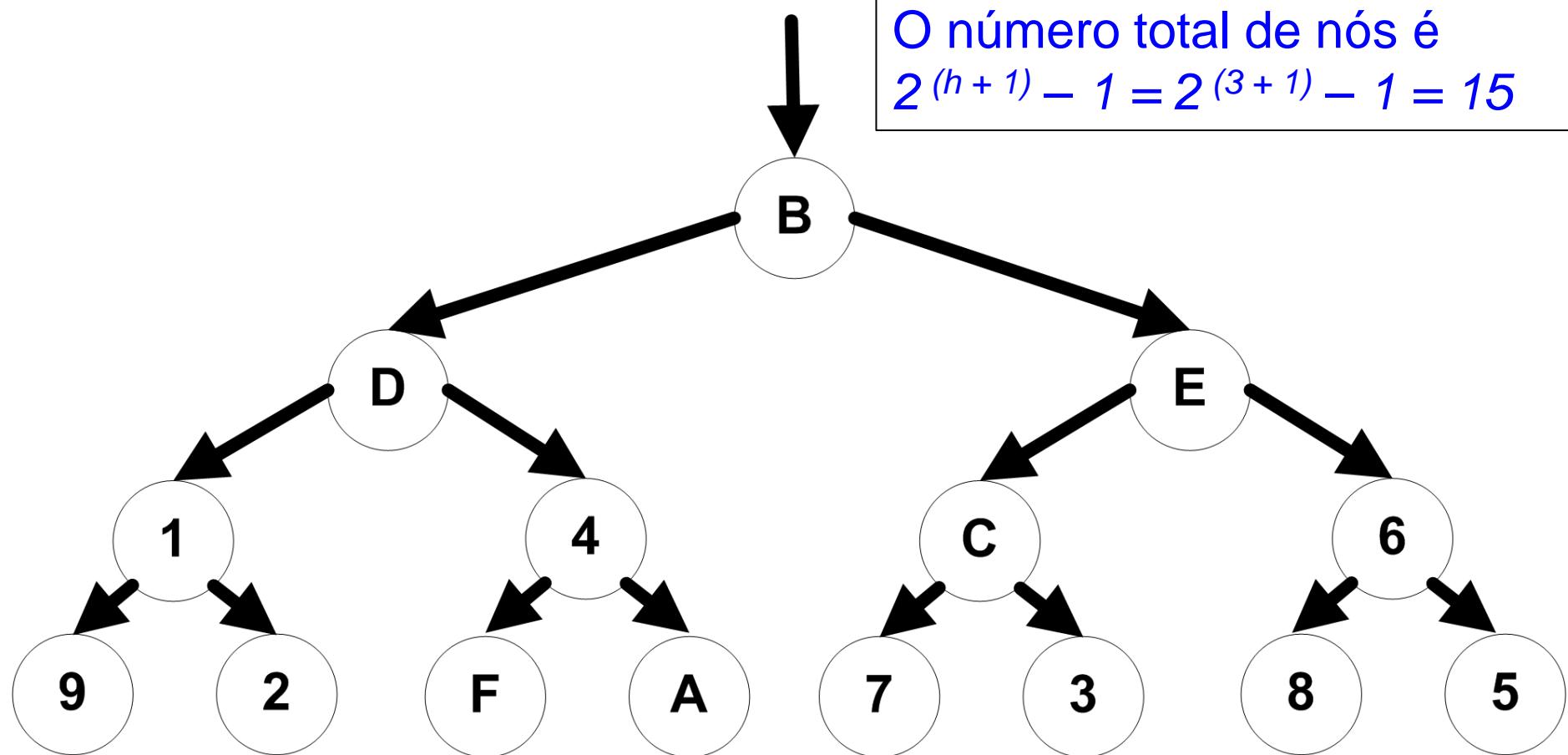
## Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos



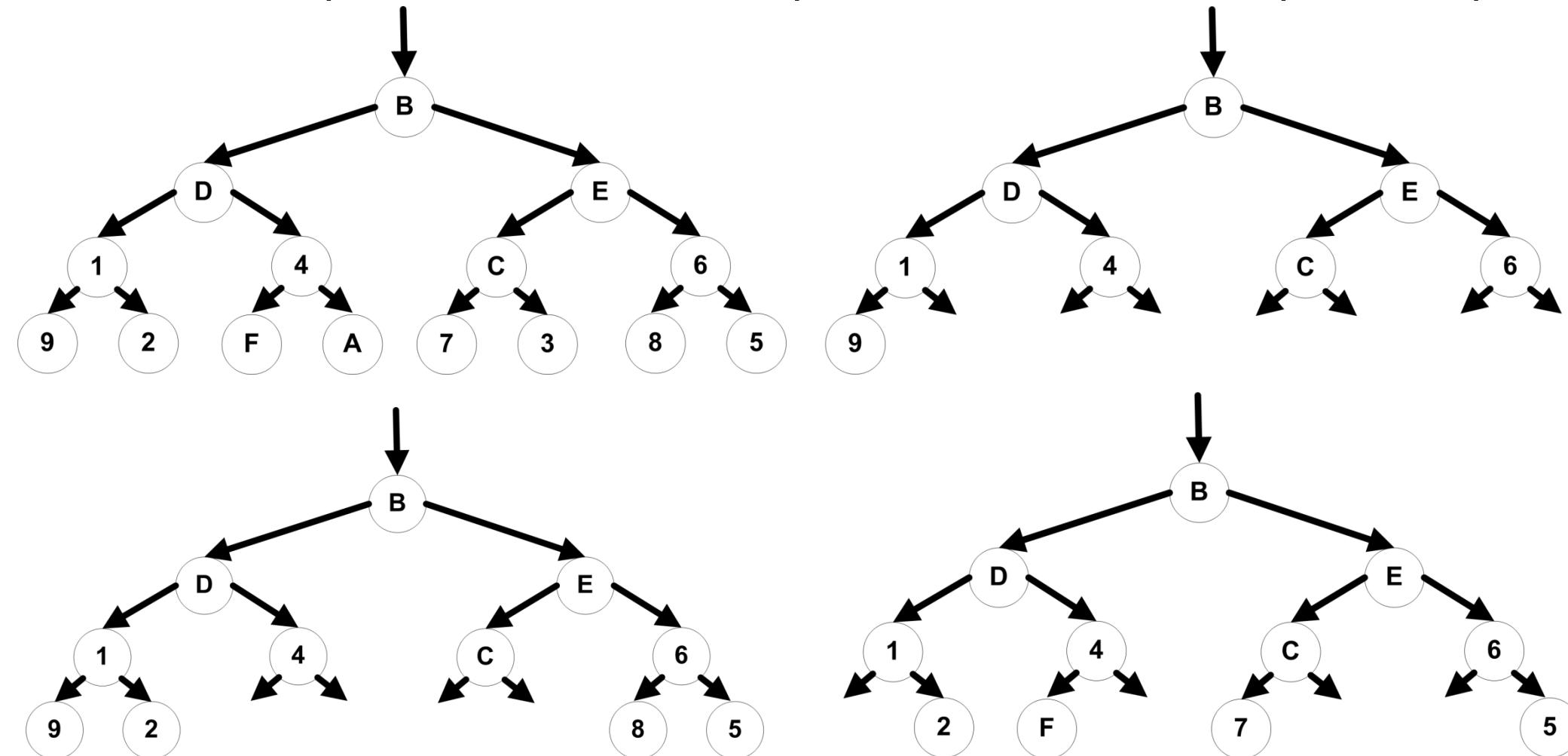
## Árvore Binária Completa

- Exercício: Crie uma árvore binária completa com os dígitos hexadecimais não nulos



## Árvore Balanceada

- Árvore em que para **TODOS** os nós, a diferença entre a altura de suas árvores da esquerda e da direita sempre será **0** ou  **$\pm 1$**  como, por exemplo:



# Consideração

- A partir deste ponto, considera-se que todas as Árvore Binária são de pesquisa

# Inserção em Árvore Binária

- Funcionamento básico:

**(1)** Se a raiz estiver vazia, insere-se o elemento nela

**(2)** Senão, se o novo elemento for **menor** que o da raiz, chama-se recursivamente a inserção para a subárvore da **esquerda**

**(3)** Senão, se o novo elemento for **maior** que o da raiz, chama-se recursivamente a inserção para a subárvore da **direita**

**(4)** Senão, se o novo elemento for **igual** ao da raiz, não inserir um elemento repetido

# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, 8, 2, 4, 7, 6

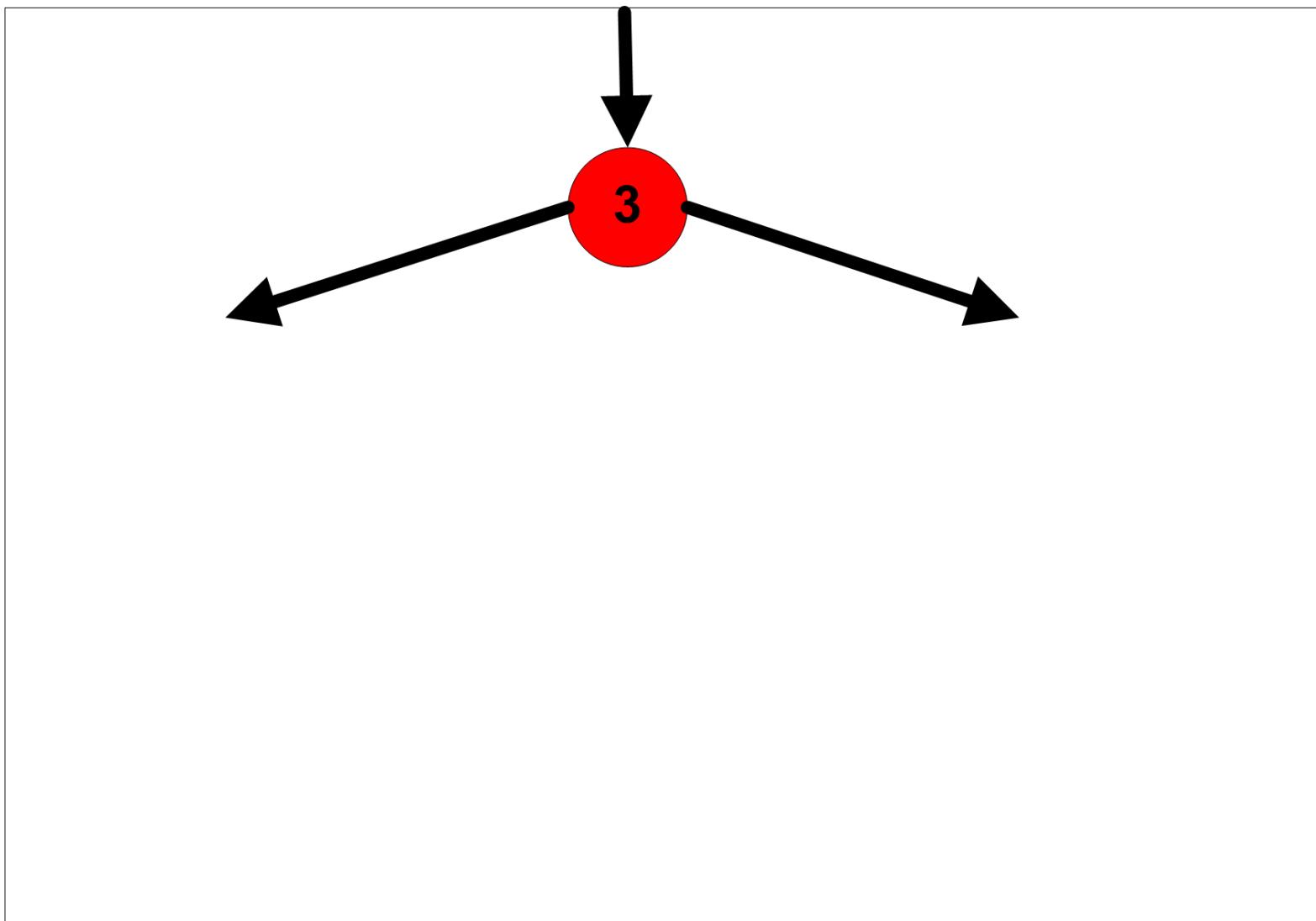
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, 8, 2, 4, 7, 6



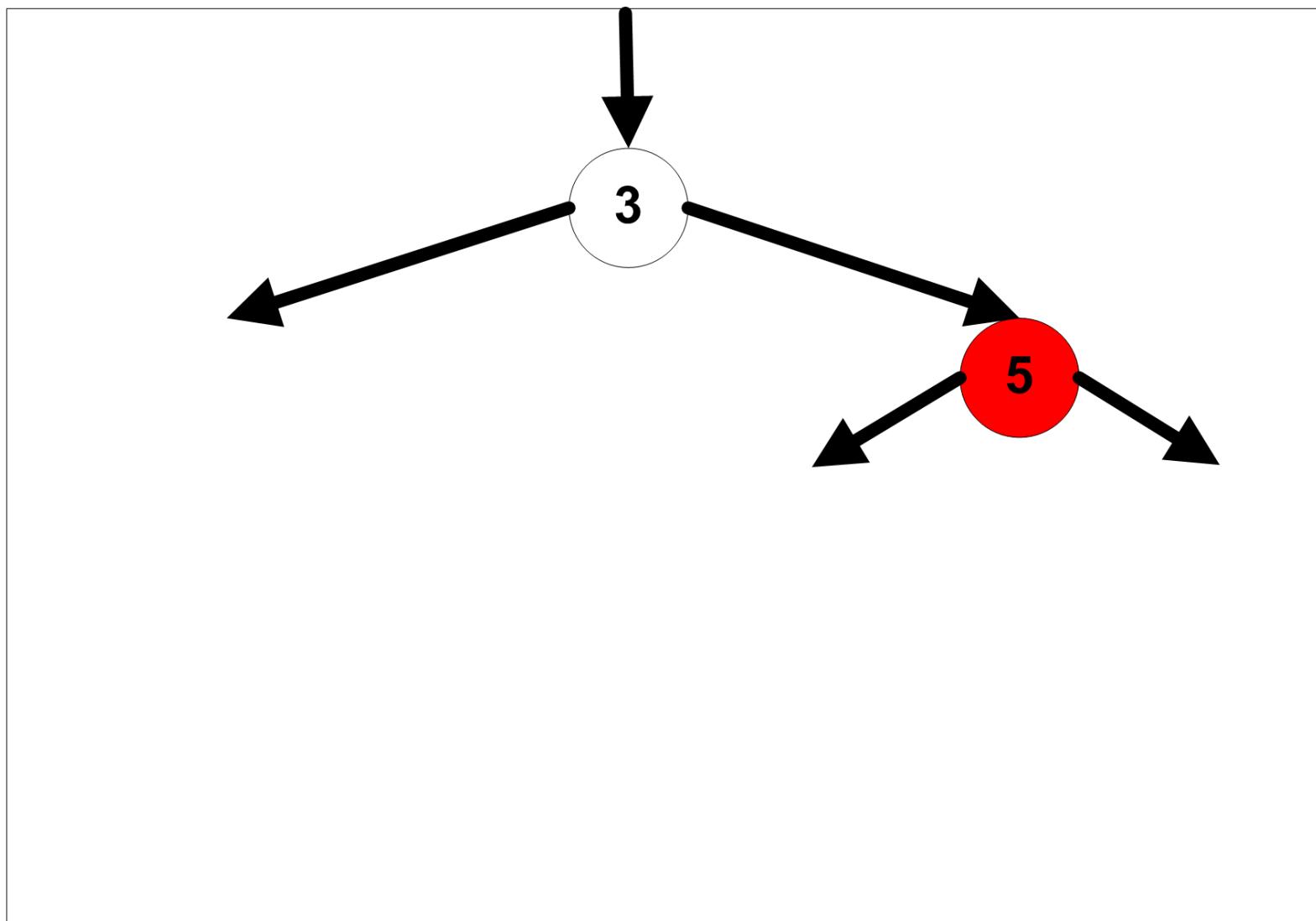
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos **3, 5, 1, 8, 2, 4, 7, 6**



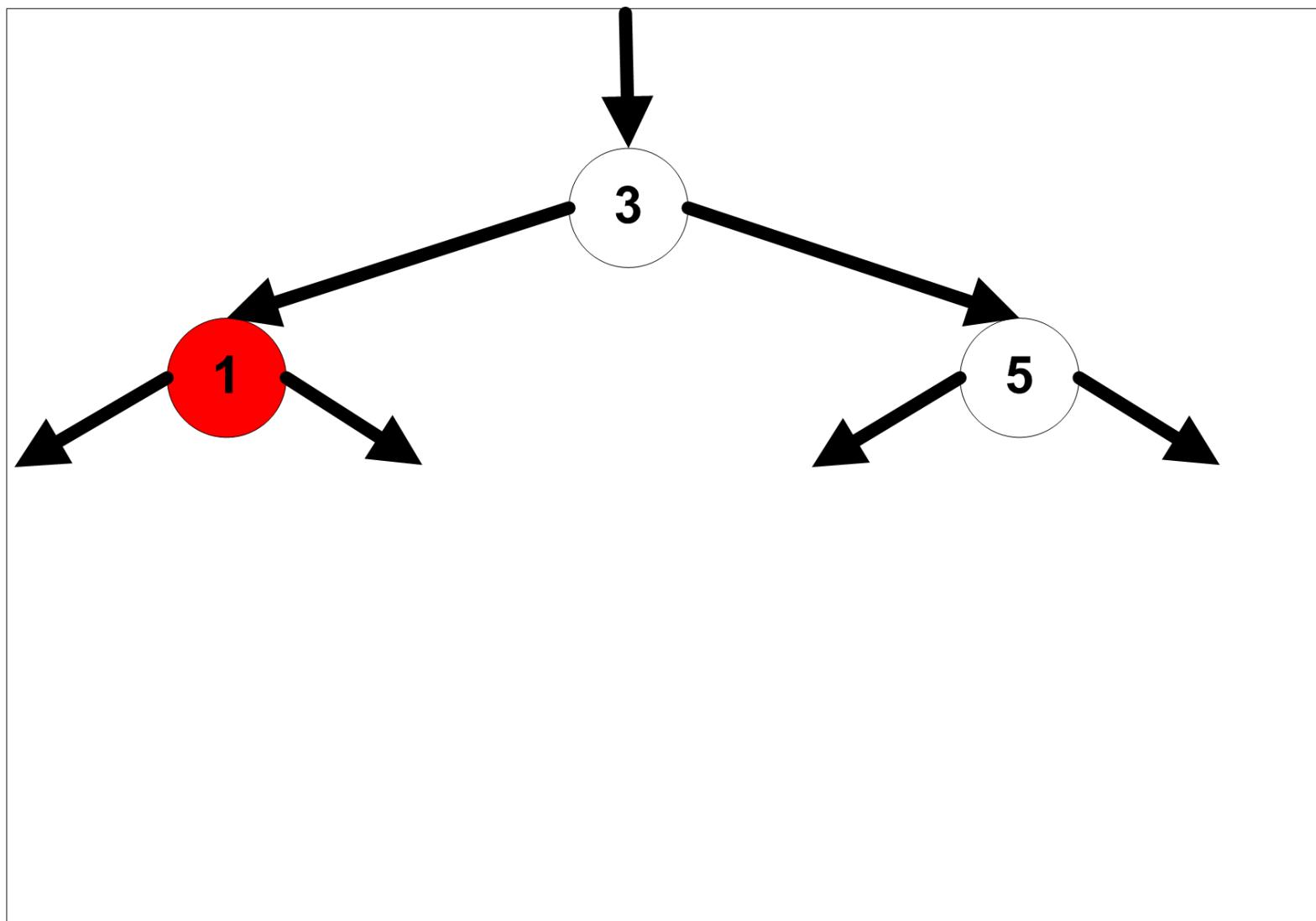
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, **5**, 1, 8, 2, 4, 7, 6



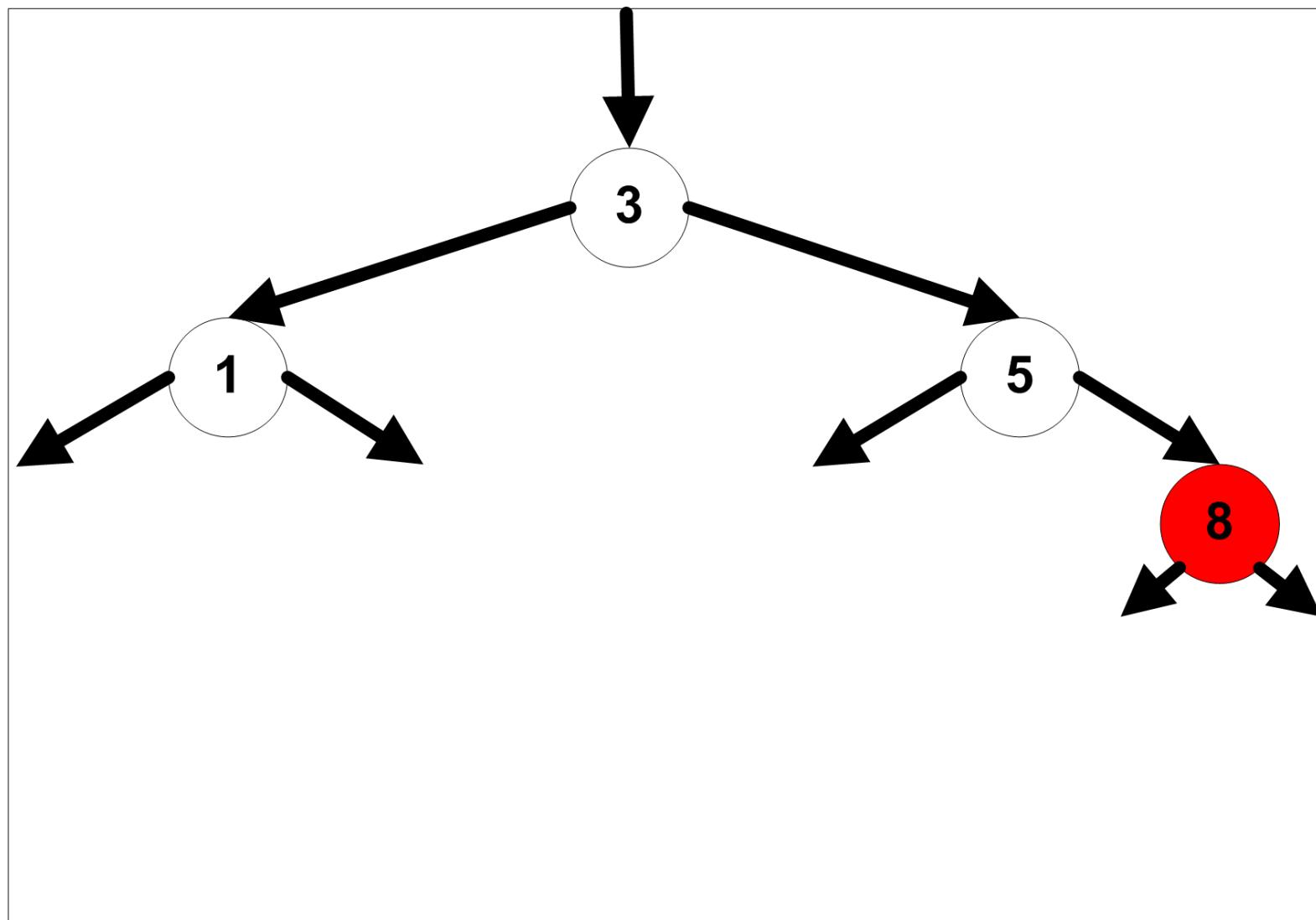
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, 8, 2, 4, 7, 6



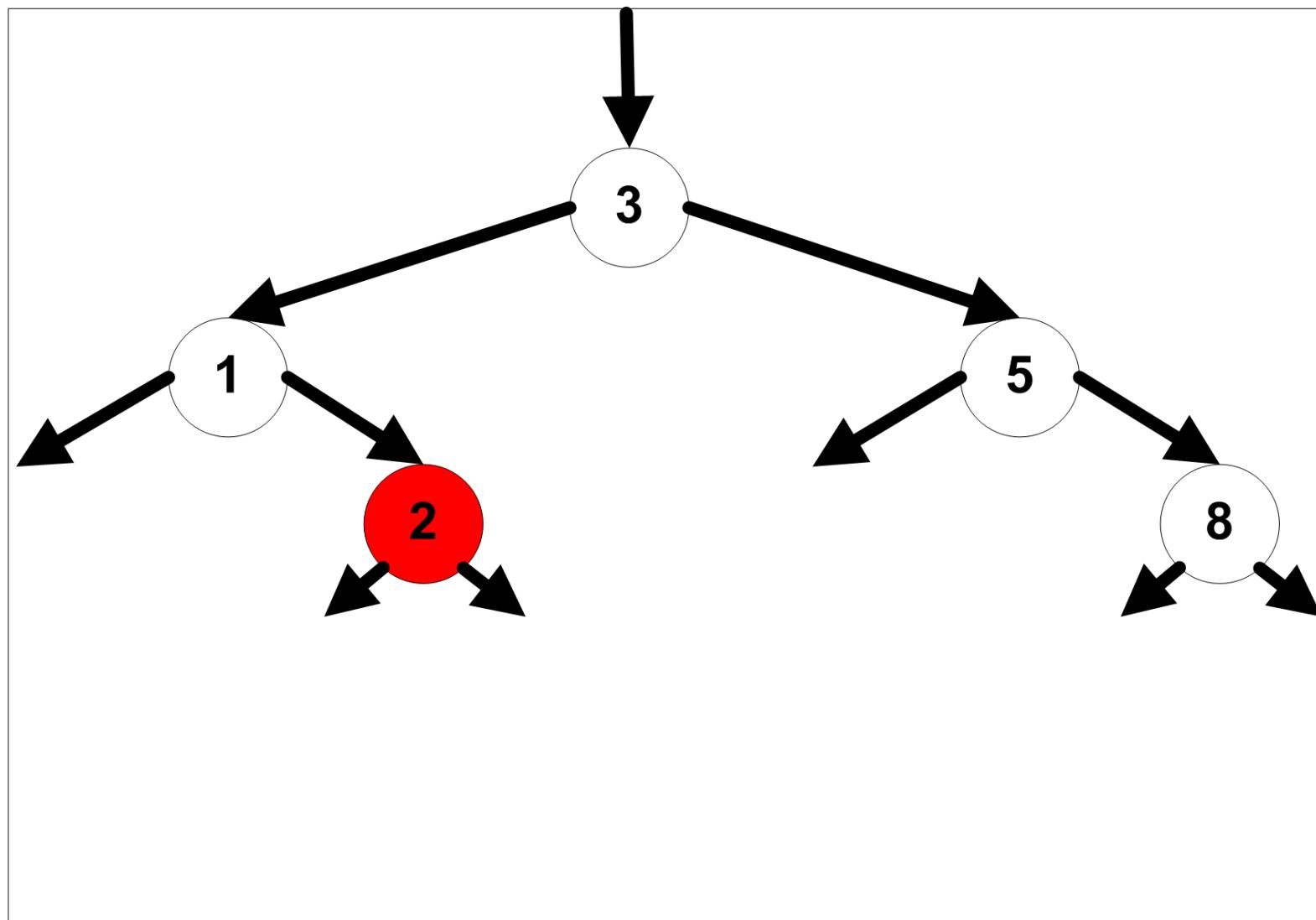
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, **8**, 2, 4, 7, 6



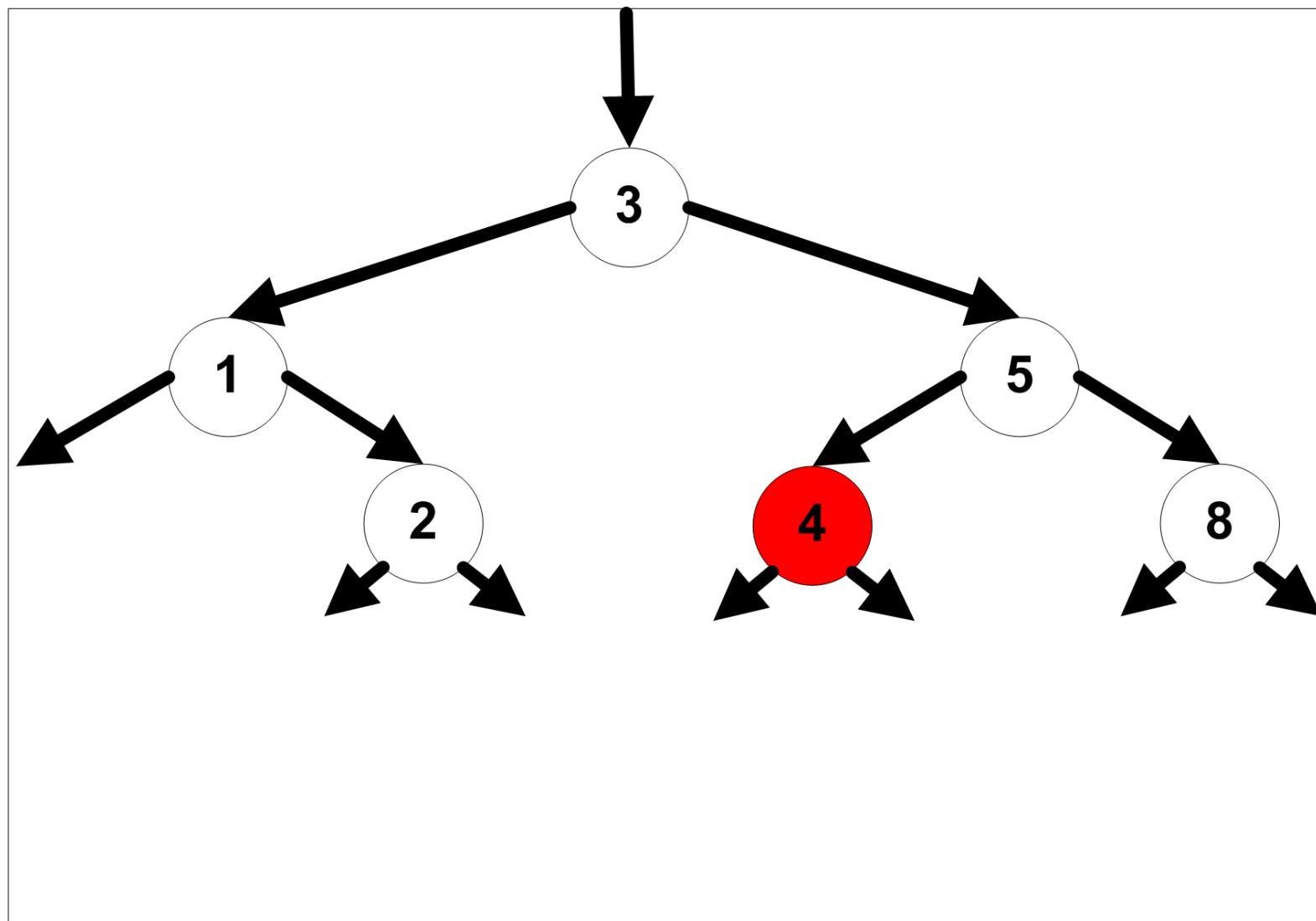
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, 8, **2**, 4, 7, 6



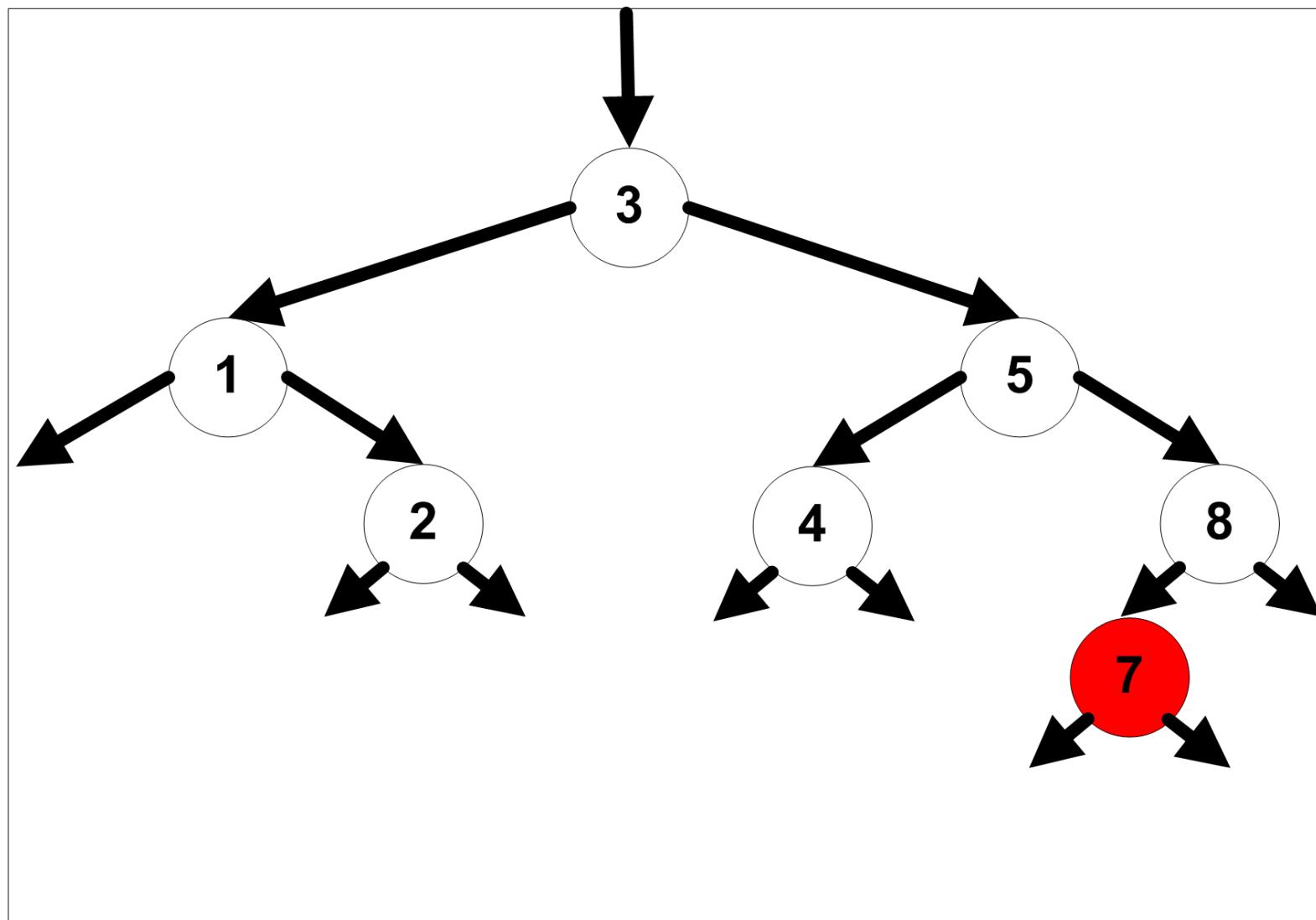
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, 8, 2, **4**, 7, 6



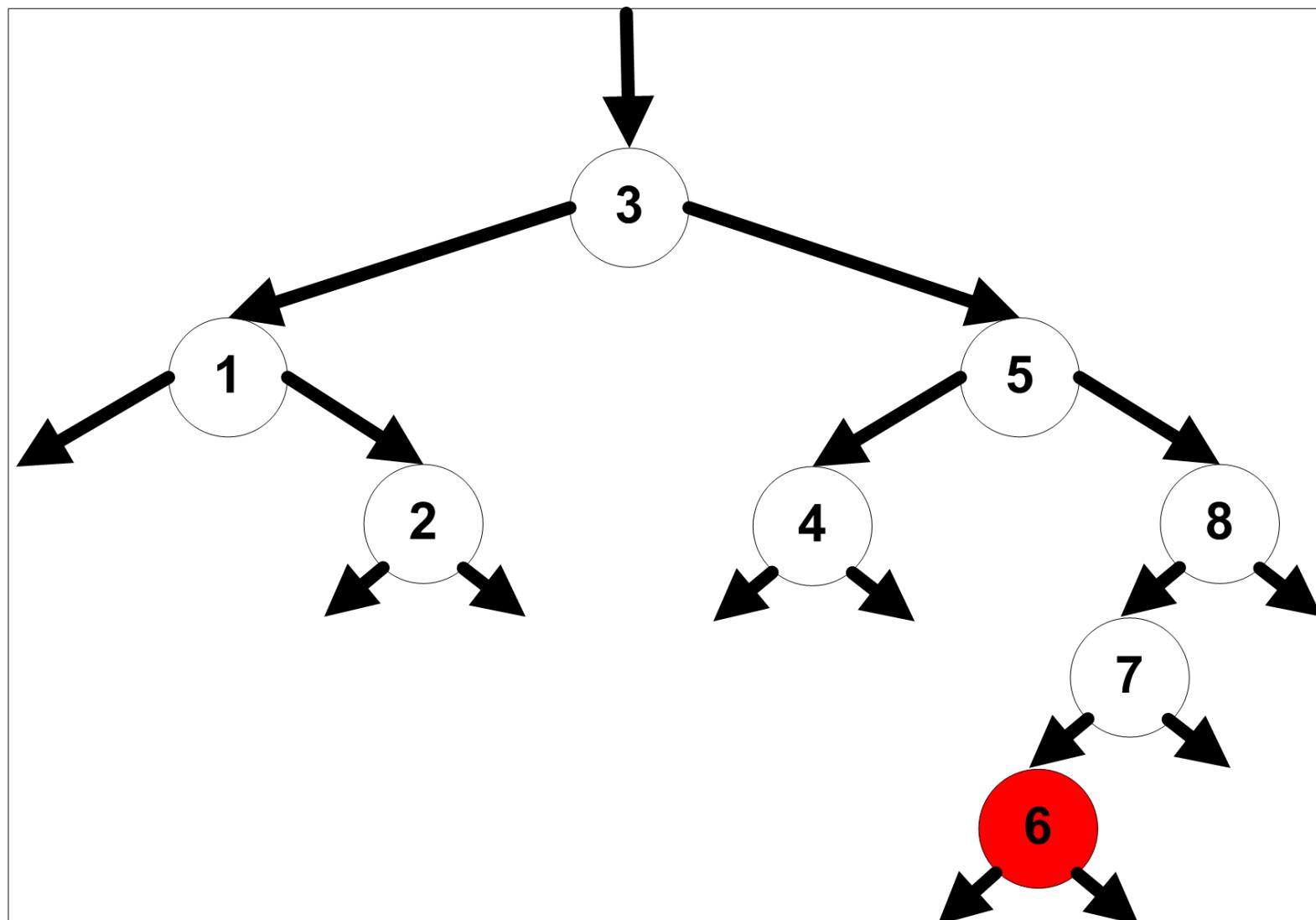
# Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, 8, 2, 4, **7**, 6



## Inserção em Árvore Binária

- Exemplo: Inserir, na ordem, os elementos 3, 5, 1, 8, 2, 4, 7, **6**



# Pesquisa em Árvore Binária

- Funcionamento básico:

(1) Se a raiz estiver vazia, retornar uma **pesquisa negativa**

(2) Senão, se o elemento procurado for **igual** ao da raiz, retornar uma **pesquisa positiva**

(3) Senão, se o elemento procurado for **menor** que o da raiz, chamar o método de pesquisa para a subárvore da **esquerda**

(4) Senão (elemento procurado é **maior** que o da raiz), chamar o método de pesquisa para a subárvore da **direita**

# Análise de Complexidade da Pesquisa

- Número de comparações em uma pesquisa com sucesso:
  - Melhor Caso:  $\Theta(1)$
  - Pior Caso:  $\Theta(n)$
  - Caso Médio:  $\Theta(\lg(n))$

# Análise de Complexidade da Pesquisa

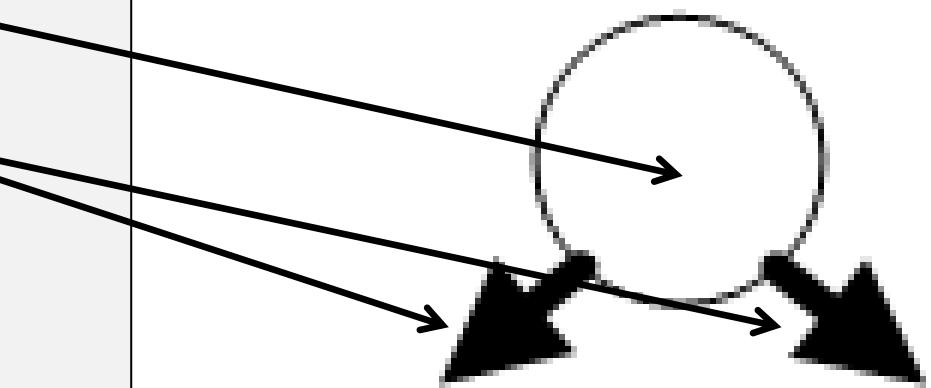
- Dependência do formato da árvore:
  - O pior caso pode ser obtido, por exemplo, através da inserção de elementos em ordem crescente ou decrescente
  - Na inserção aleatória, o número esperado de comparações é de aproximadamente  $1,39 \lg(n)$
  - **Exercício:** Em nosso código da AB, crie o método *int getAltura()* que retorna a altura de nossa AB. Em seguida, insira vários elementos de forma aleatória e veja o tamanho de nossa AB

# Análise de Complexidade da Inserção



# Algoritmo em Java - Classe Nó

```
class No {  
    public int elemento;  
    public No esq;  
    public No dir;  
    public No(int elemento) {  
        this(elemento, null, null);  
    }  
    public No(int elemento, No esq, No dir) {  
        this.elemento = elemento;  
        this.esq = esq;  
        this.dir = dir;  
    }  
}
```

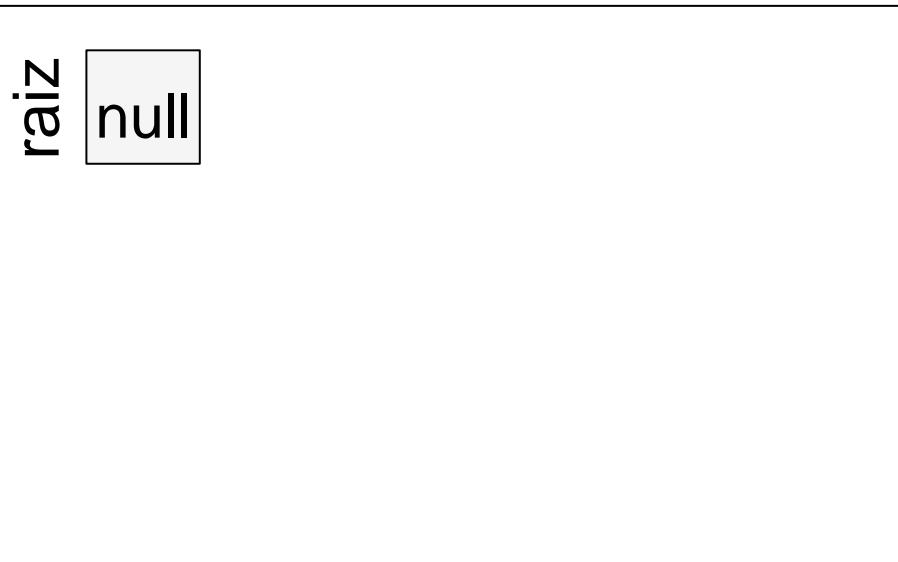


# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

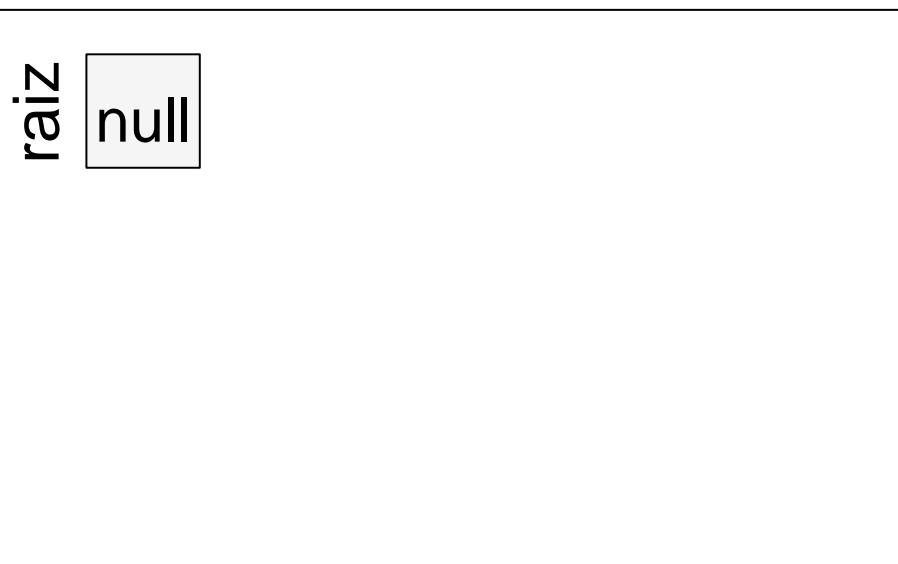
# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```



# Algoritmo em Java - Classe Árvore Binária

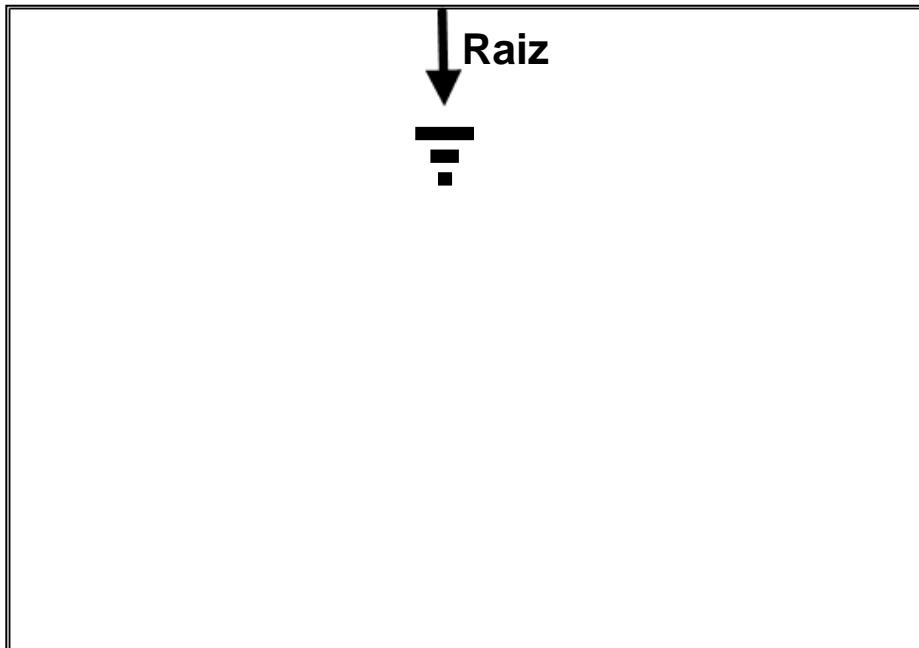
```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```



# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz      null



Vamos inserir os elementos

3, 5, 1, 8, 2, 4, 7 e 6

(várias chamadas do inserir)

# Algoritmo em Java - Classe Árvore Binária

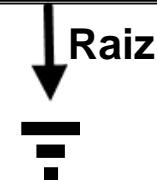
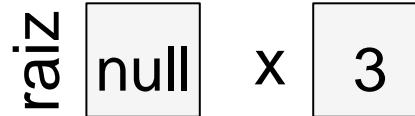
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
```

```
    raiz = inserir(x, raiz);  
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```



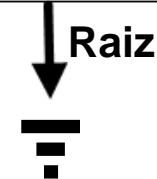
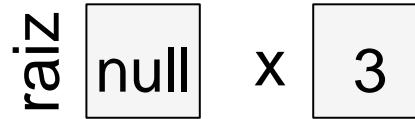
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);
```

```
}
```

```
private No inserir(int x, No i) throws Exception {  
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

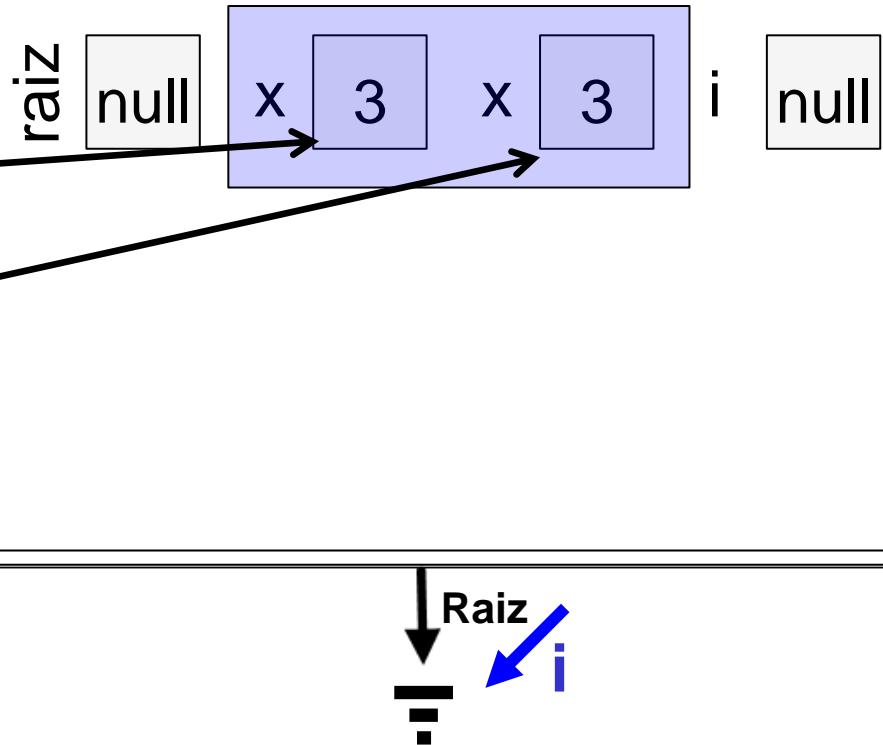


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else
        throw ...
    return i;
}
```



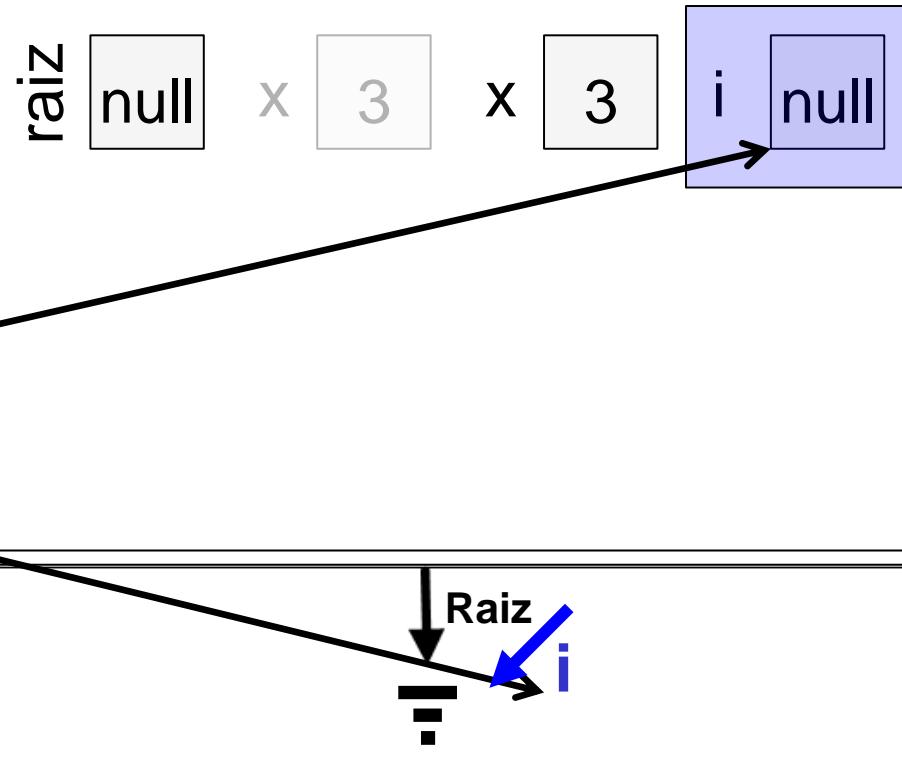
Cada chamada do inserir cria novas variáveis  
e, por isso, temos duas variáveis com nome x

# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else
        throw new Exception("Valor duplicado");
    return i;
}
```



O valor inicial do ponteiro *i* é o mesmo  
do ponteiro *raiz*, ou seja, null

# Algoritmo em Java - Classe Árvore Binária

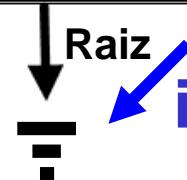
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

true: null == null



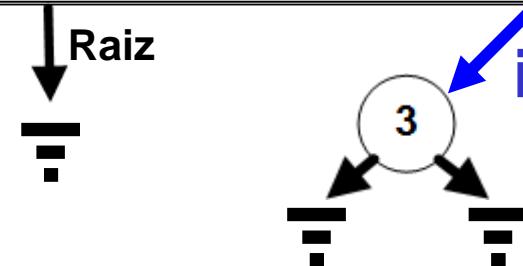
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz    null    x    3    x    3    i    n(3)



# Algoritmo em Java - Classe Árvore Binária

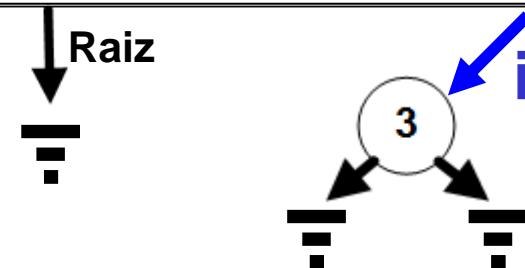
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

retorna o endereço de n(3)

raiz    null    x    3    x    3    i    n(3)



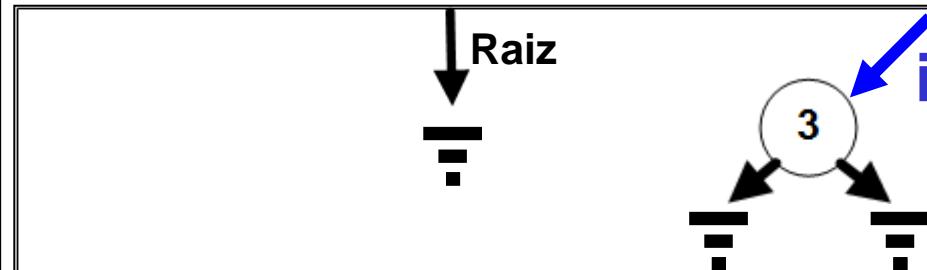
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz    null    x    3    x    3    i    n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

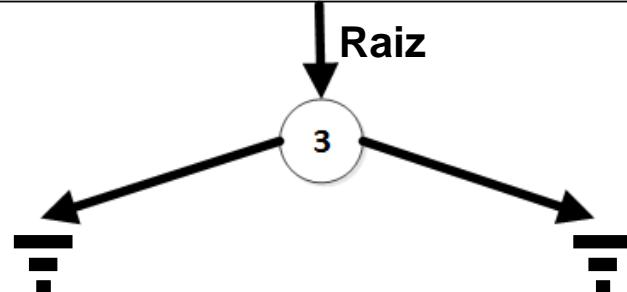
```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);
```

```
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz n(3) x 3



# Algoritmo em Java - Classe Árvore Binária

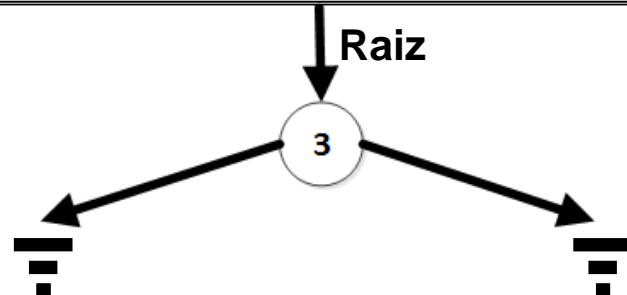
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);
```

```
}
```

```
private No inserir(int x, No i) throws Exception {  
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz  
n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
```

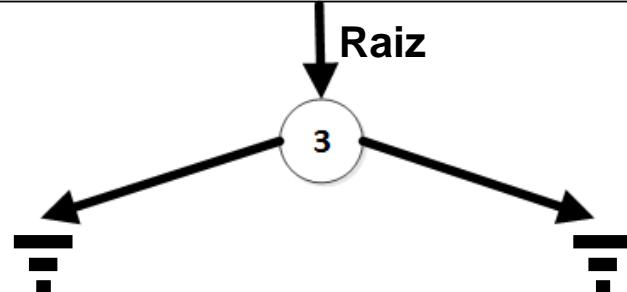
```
    raiz = inserir(x, raiz);  
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    }
```

```
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz n(3) x 5



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

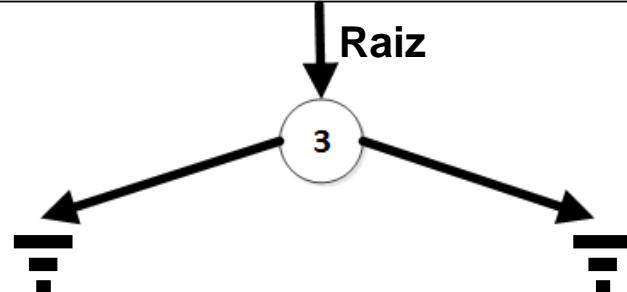
```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);
```

```
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz n(3) x 5



# Algoritmo em Java - Classe Árvore Binária

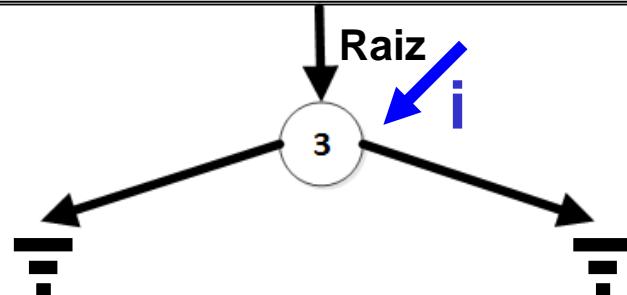
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3)   x 5   x 5   i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

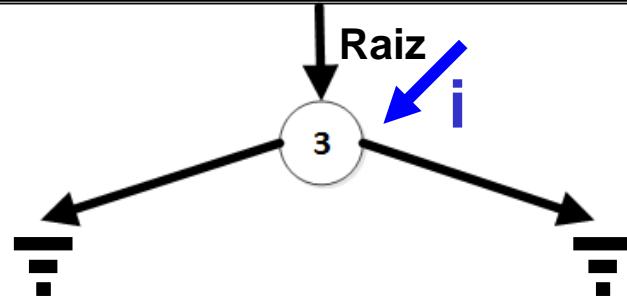
```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
if (i == null) {
    i = new No(x);
} else if (x < i.elemento) {
    i.esq = inserir(x, i.esq);
} else if (x > i.elemento) {
    i.dir = inserir(x, i.dir);
} else {
    throw new Exception("Erro!");
}
return i;
}
```

false: n(3) == null

raiz n(3) x 5 x 5 i n(3)



# Algoritmo em Java - Classe Árvore Binária

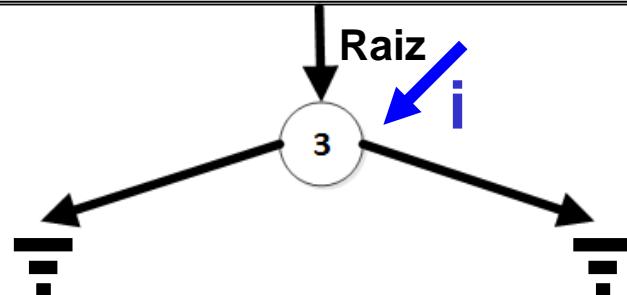
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

false:  $5 < 3$

raiz n(3)   x 5   x 5   i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    }
```

```
    } else if (x > i.elemento) {
```

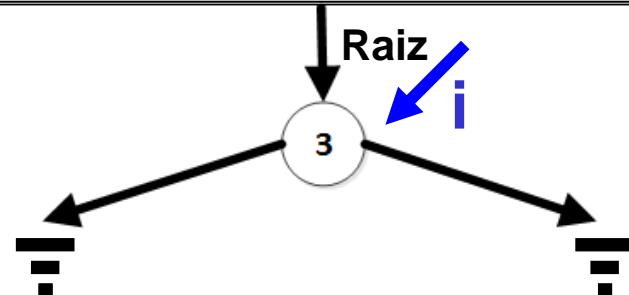
```
        i.dir = inserir(x, i.dir);
    } else {
```

```
        throw new Exception("Erro!");
    }
```

```
    return i;
}
```

true: 5 > 3

raiz n(3) x 5 x 5 i n(3)



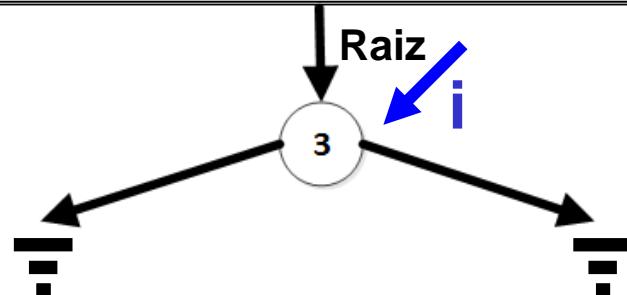
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3) x 5 x 5 i n(3)

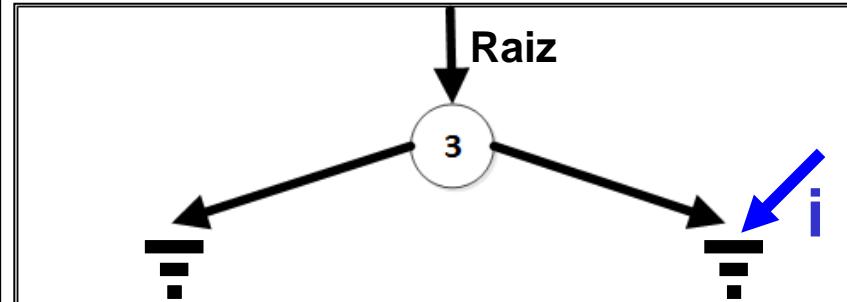
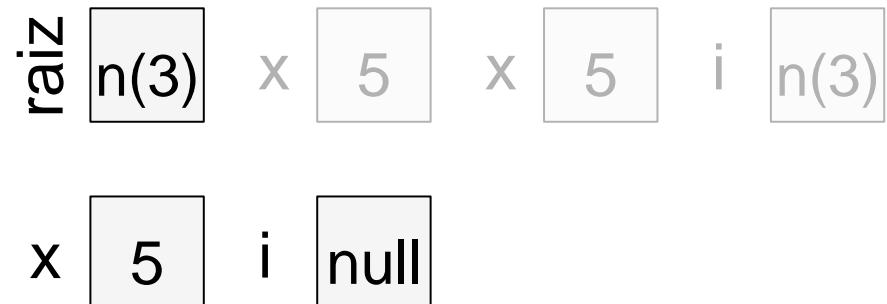


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}

private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

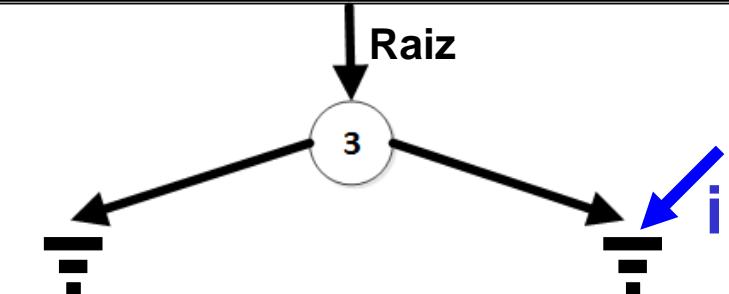
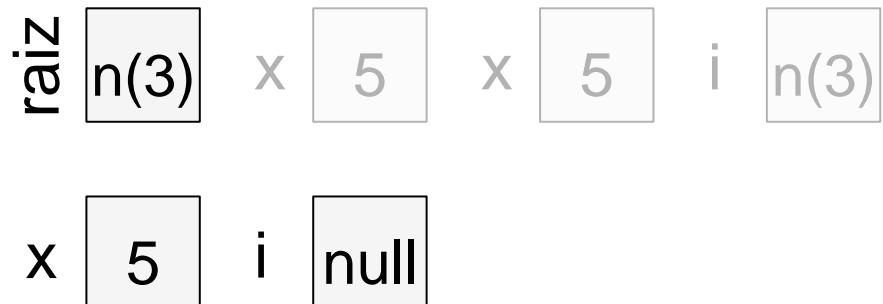


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}

private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
true: null == null
```



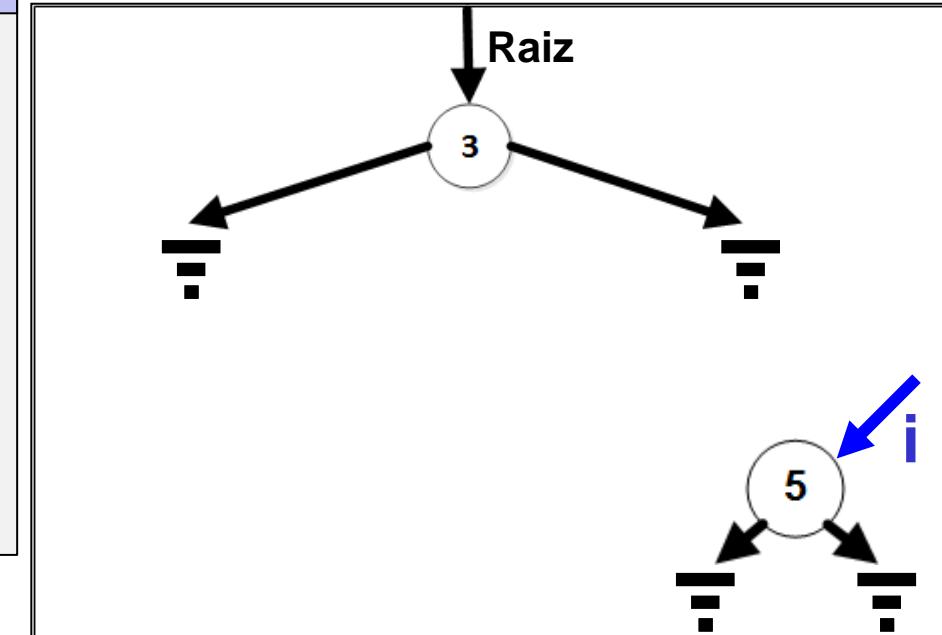
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
```

```
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

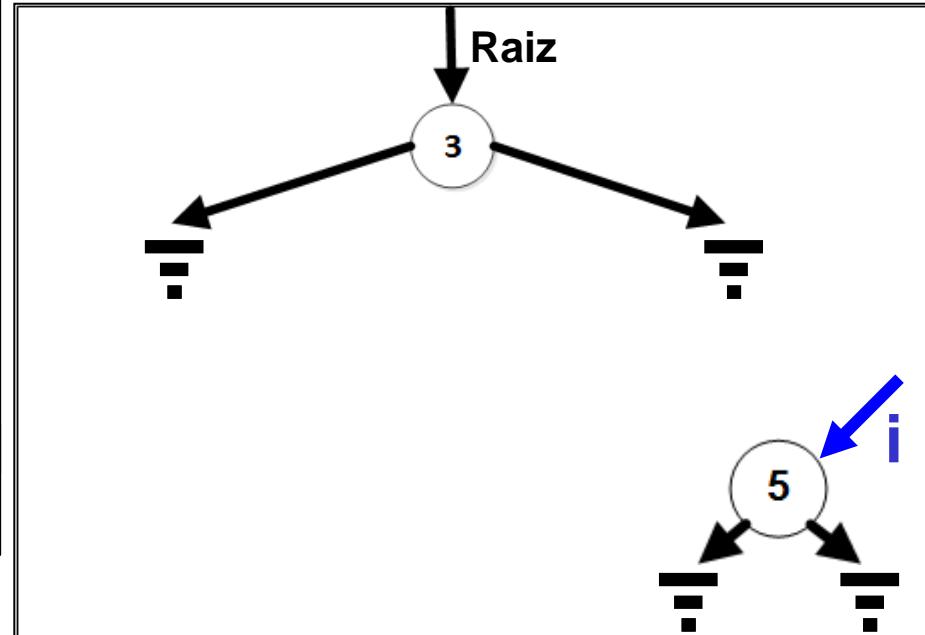
```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

retorna o endereço de n(5)

raiz n(3) x 5 x 5 i n(3)

x 5 i n(5)

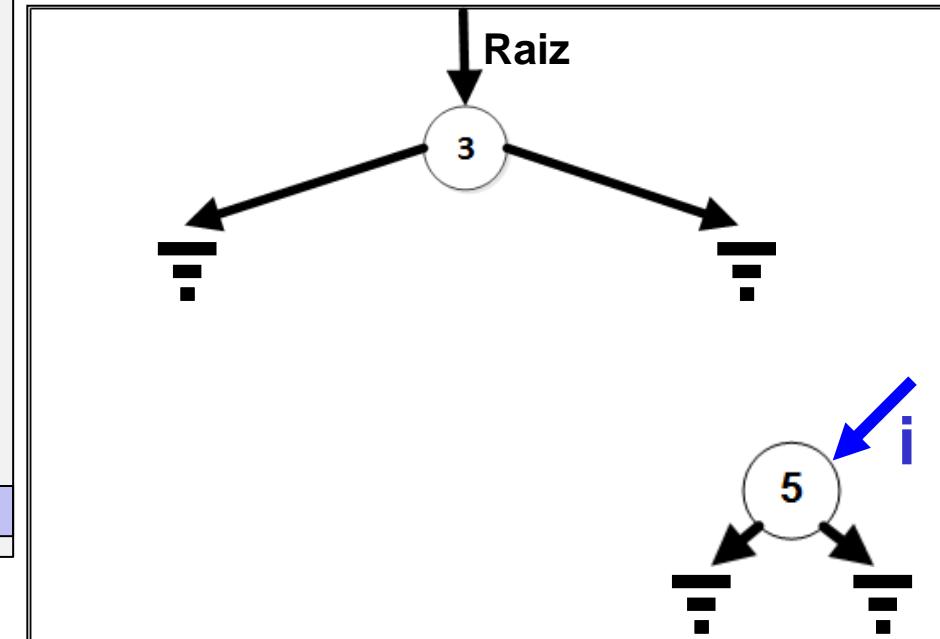


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}

private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

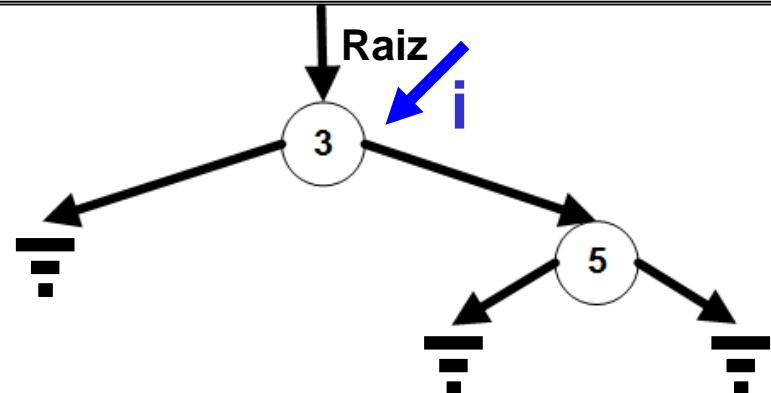
```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
```

```
        i.dir = inserir(x, i.dir);
    } else {
```

```
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3)   x 5   x 5   i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

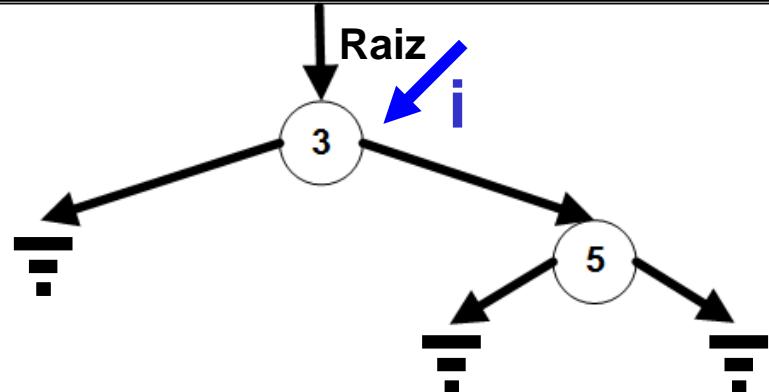
```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
```

```
        throw new Exception("Erro!");
    }
}
```

```
return i;
```

retorna o endereço de n(3)

raiz n(3)   x 5   x 5   i n(3)



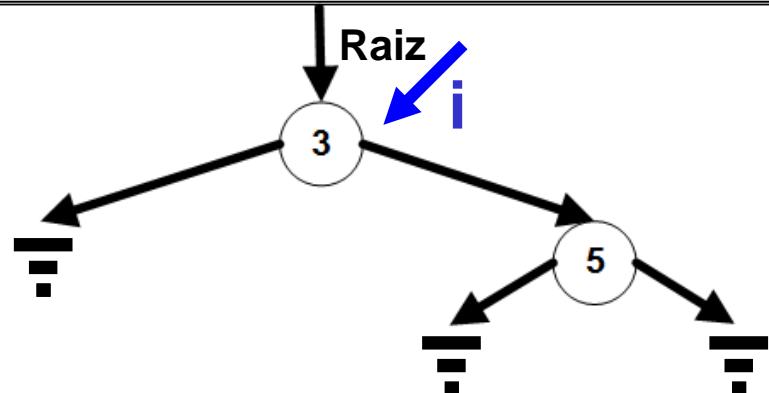
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3) x 5 x 5 i n(3)



# Algoritmo em Java - Classe Árvore Binária

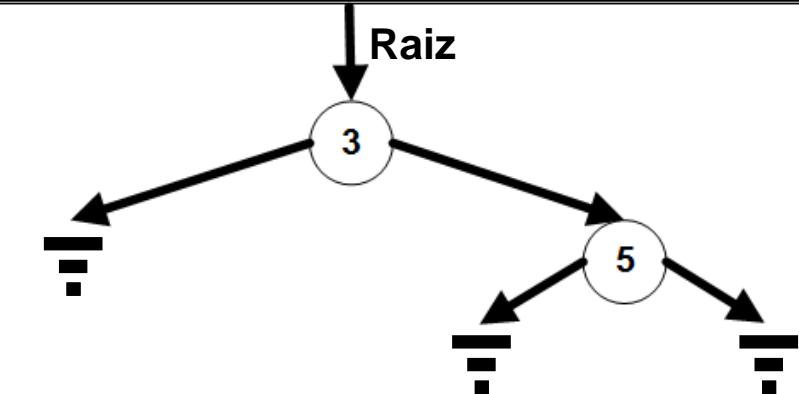
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);
```

```
}
```

```
private No inserir(int x, No i) throws Exception {  
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz n(3) x 5



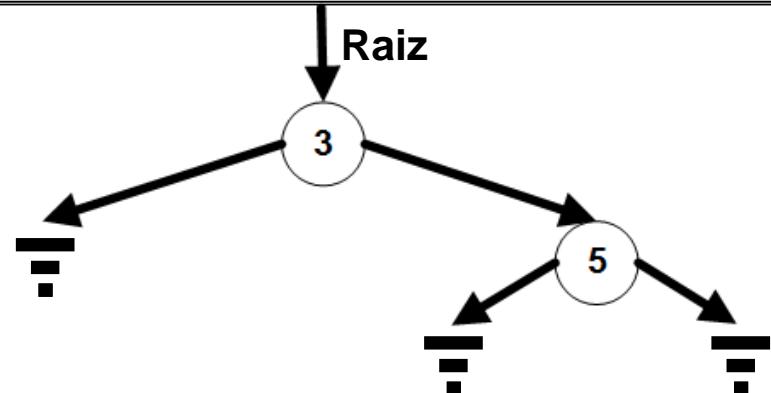
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);  
}
```

```
private No inserir(int x, No i) throws Exception {  
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz  
n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

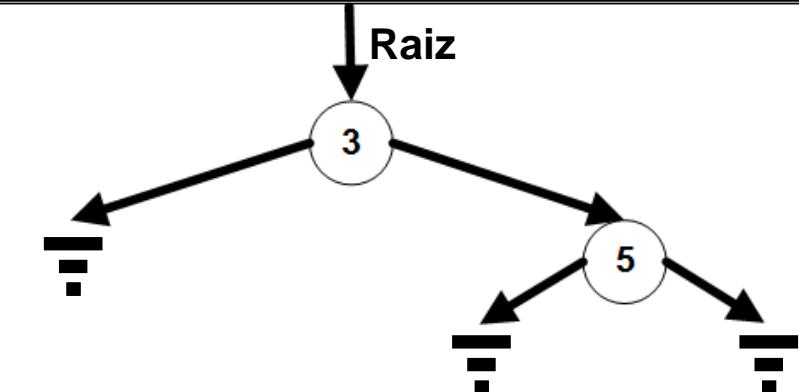
```
public void inserir(int x) throws Exception {
```

```
    raiz = inserir(x, raiz);  
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz n(3) x 1



# Algoritmo em Java - Classe Árvore Binária

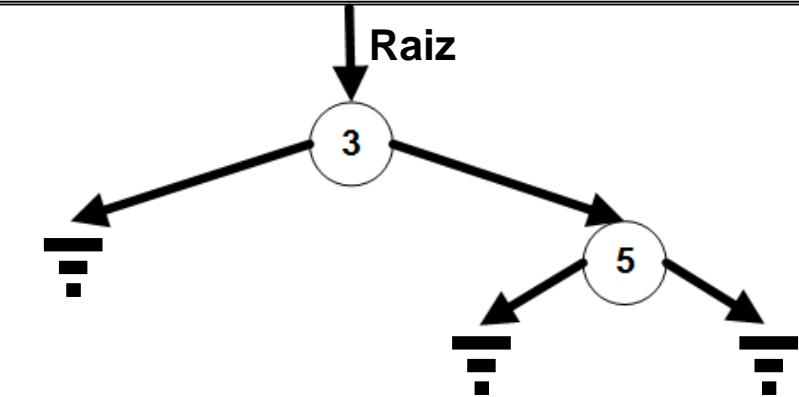
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);
```

```
}
```

```
private No inserir(int x, No i) throws Exception {  
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz n(3) x 1



# Algoritmo em Java - Classe Árvore Binária

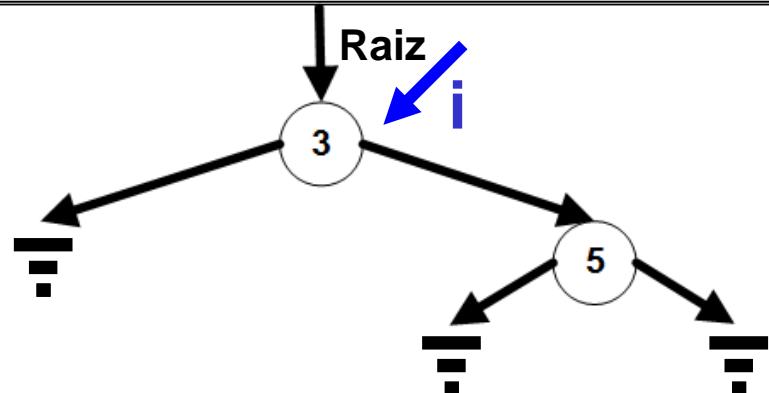
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3)   x 1   x 1   i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

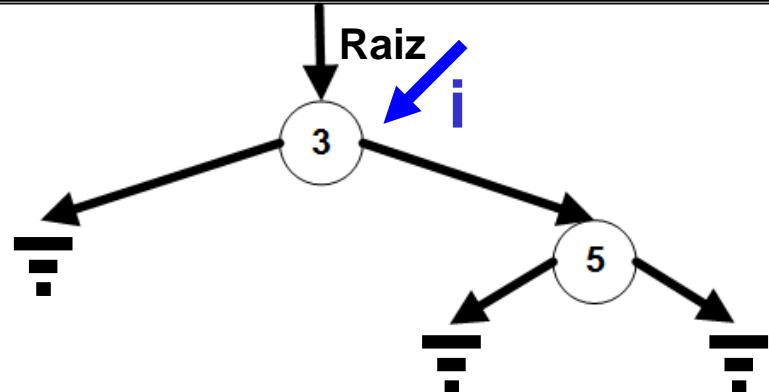
```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

false: n(3) == null

raiz n(3)   x 1   x 1   i n(3)



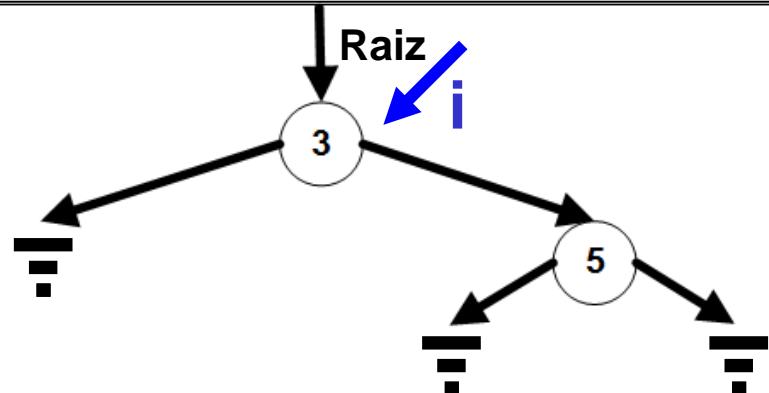
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
true: 1 < 3
```

raiz n(3) x 1 x 1 i n(3)



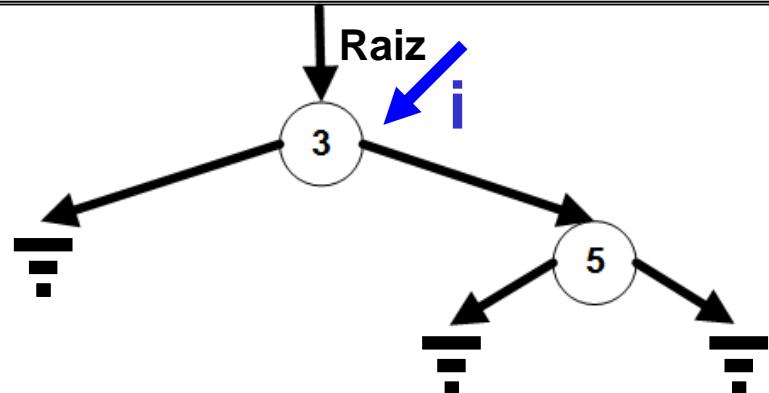
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

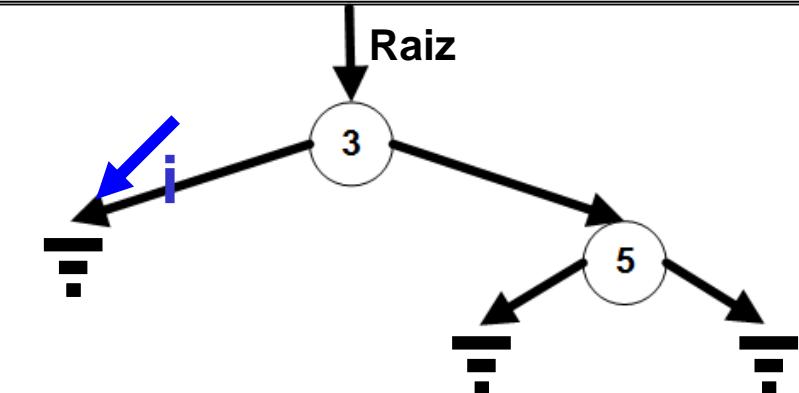
```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3) x 1 x 1 i n(3)

x 1 i null

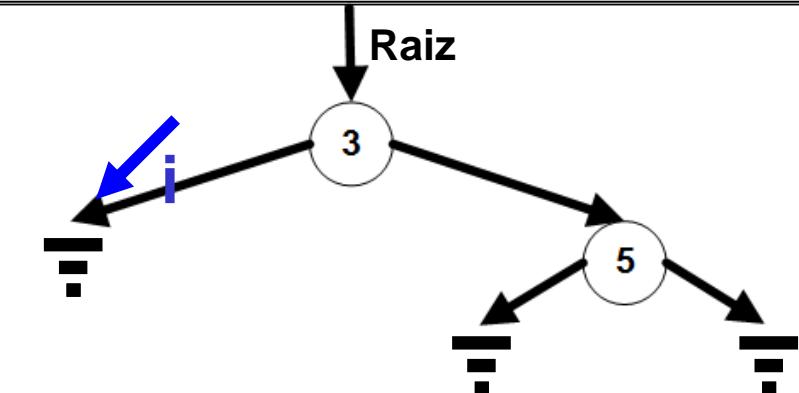
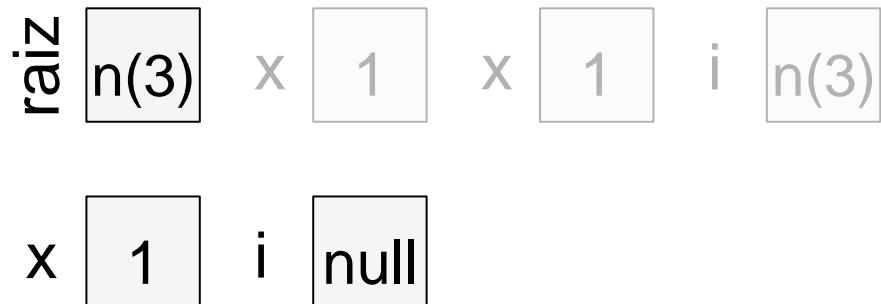


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}

private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
true: null == null
```

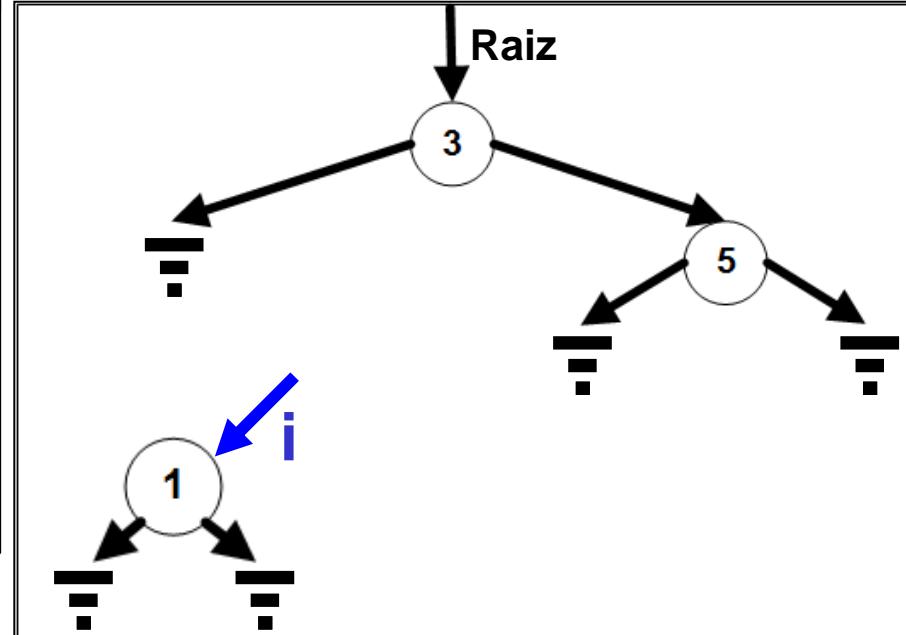
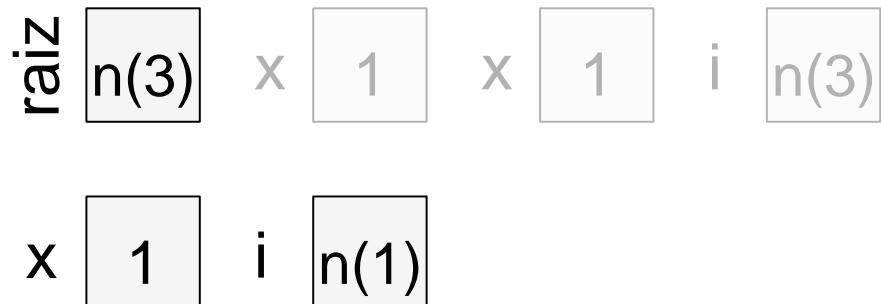


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

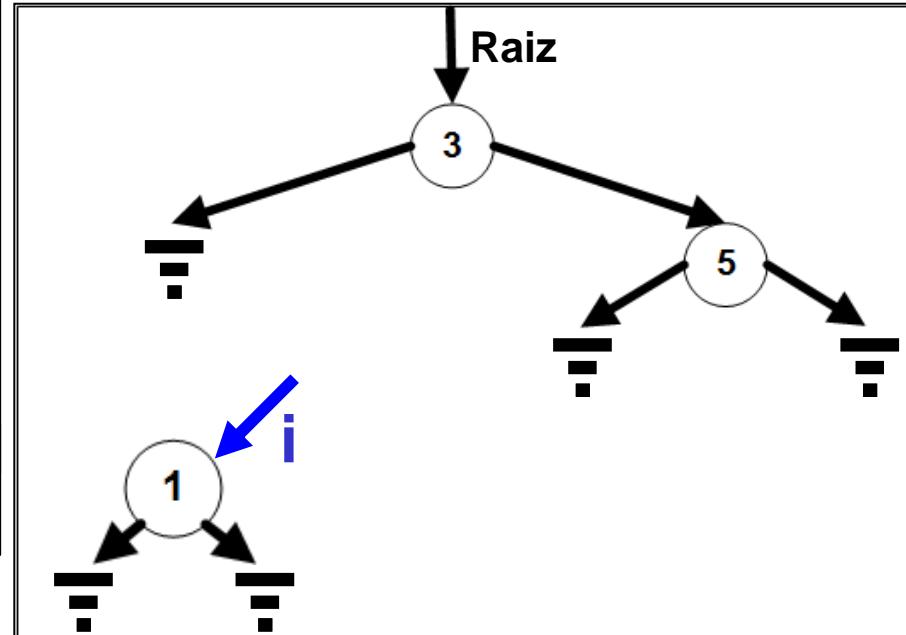
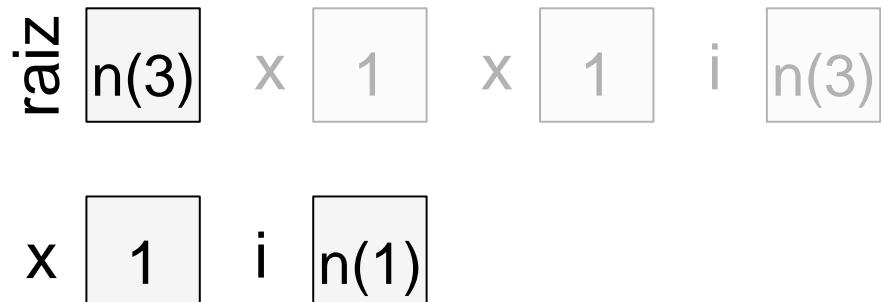


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}

private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

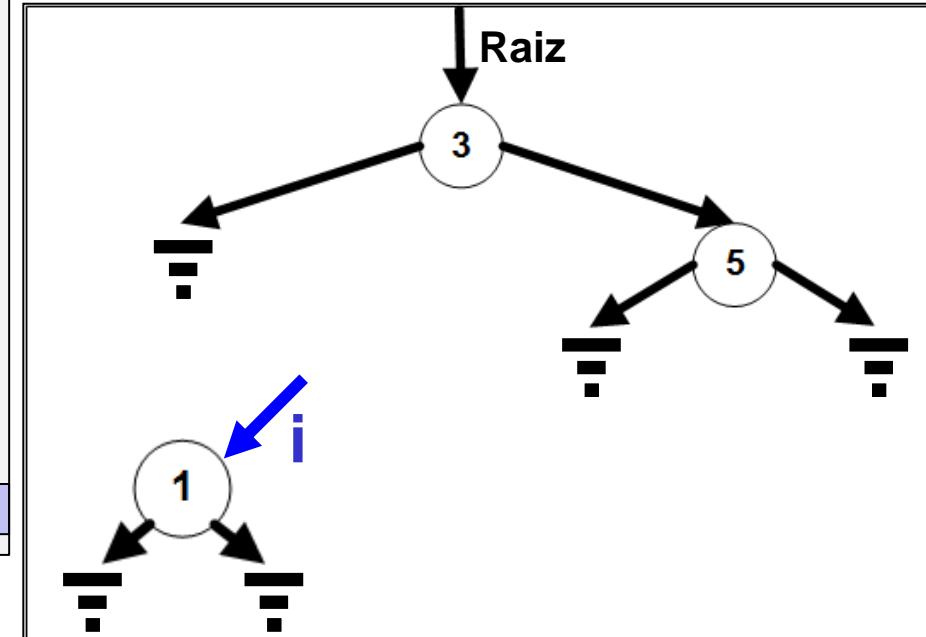
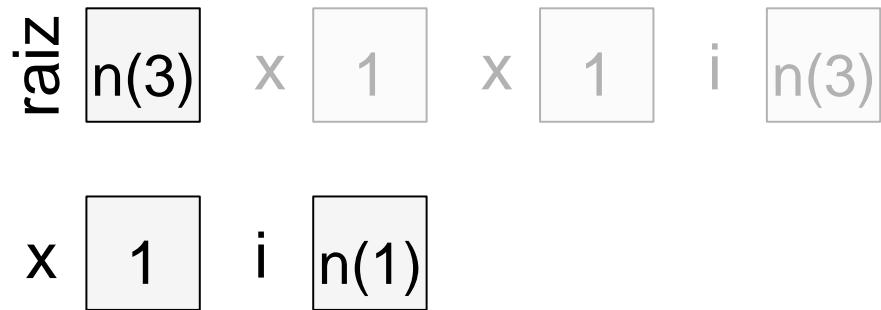


# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}

private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```



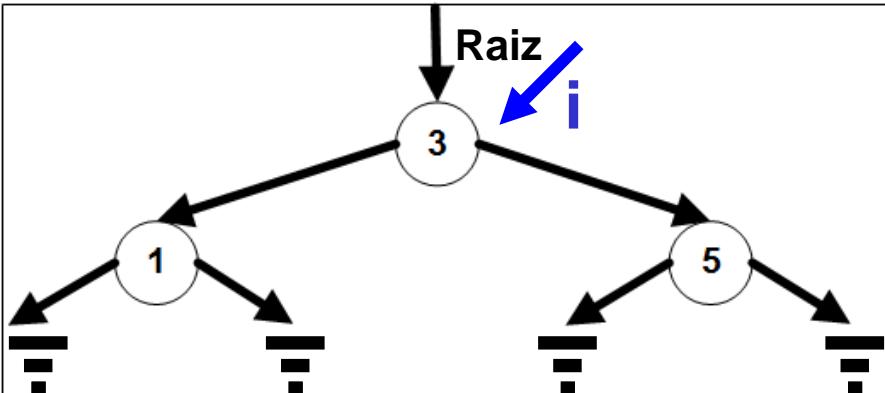
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3)   x 1   x 1   i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
```

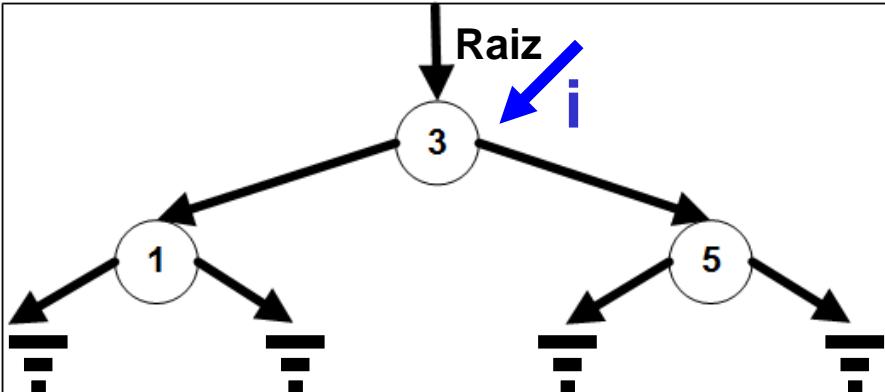
```
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
```

```
        throw new Exception("Erro!");
    }
}
```

```
return i;
```

retorna o endereço de n(3)

raiz n(3)   x 1   x 1   i n(3)



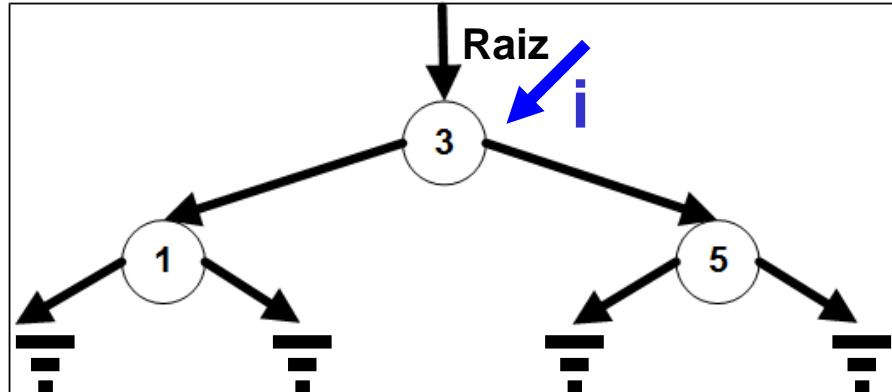
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3)   x 1   x 1   i n(3)



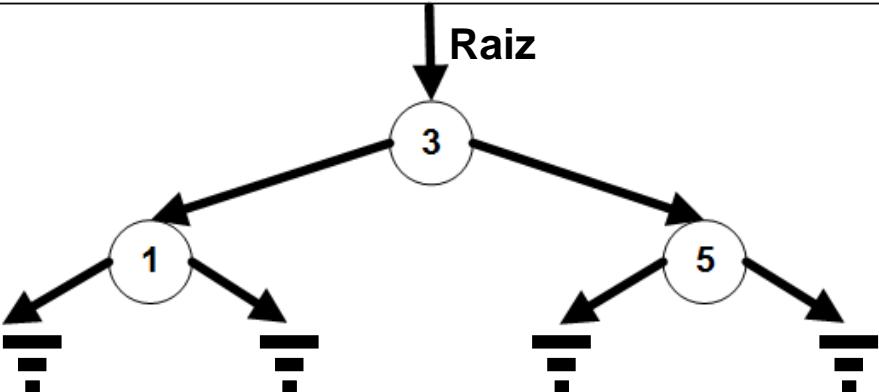
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {
    raiz = inserir(x, raiz);
}
```

```
private No inserir(int x, No i) throws Exception {
    if (i == null) {
        i = new No(x);
    } else if (x < i.elemento) {
        i.esq = inserir(x, i.esq);
    } else if (x > i.elemento) {
        i.dir = inserir(x, i.dir);
    } else {
        throw new Exception("Erro!");
    }
    return i;
}
```

raiz n(3) x 1



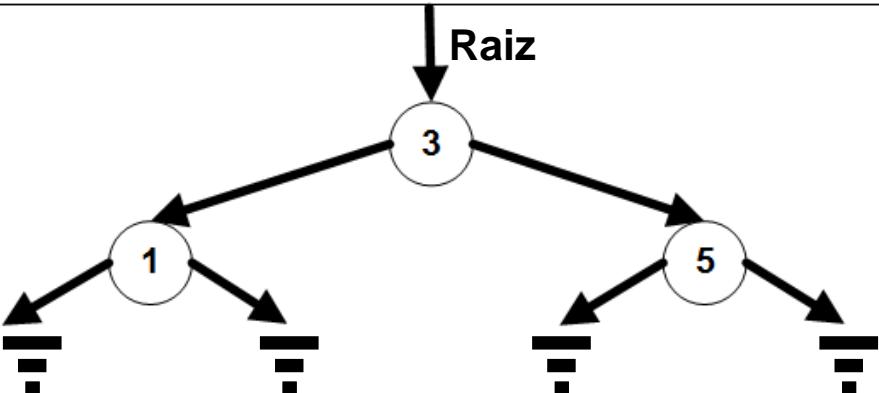
# Algoritmo em Java - Classe Árvore Binária

```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6
```

```
public void inserir(int x) throws Exception {  
    raiz = inserir(x, raiz);  
}
```

```
private No inserir(int x, No i) throws Exception {  
    if (i == null) {  
        i = new No(x);  
    } else if (x < i.elemento) {  
        i.esq = inserir(x, i.esq);  
    } else if (x > i.elemento) {  
        i.dir = inserir(x, i.dir);  
    } else {  
        throw new Exception("Erro!");  
    }  
    return i;  
}
```

raiz  
n(3)



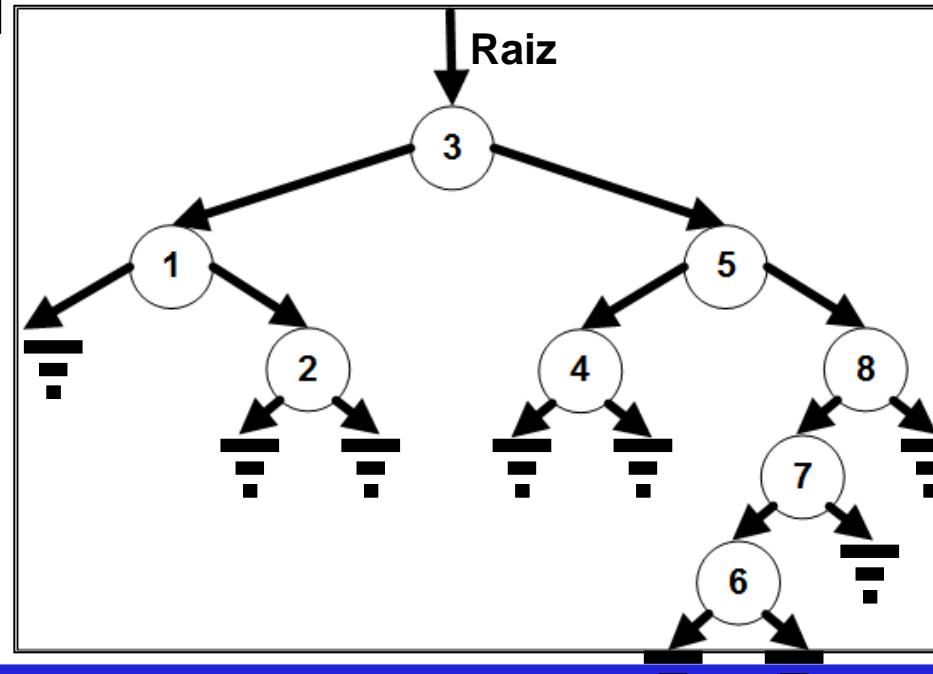
# Algoritmo em Java - Classe Árvore Binária

Após a inserção do 8, 2, 4, 7 e 6, temos:

# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)

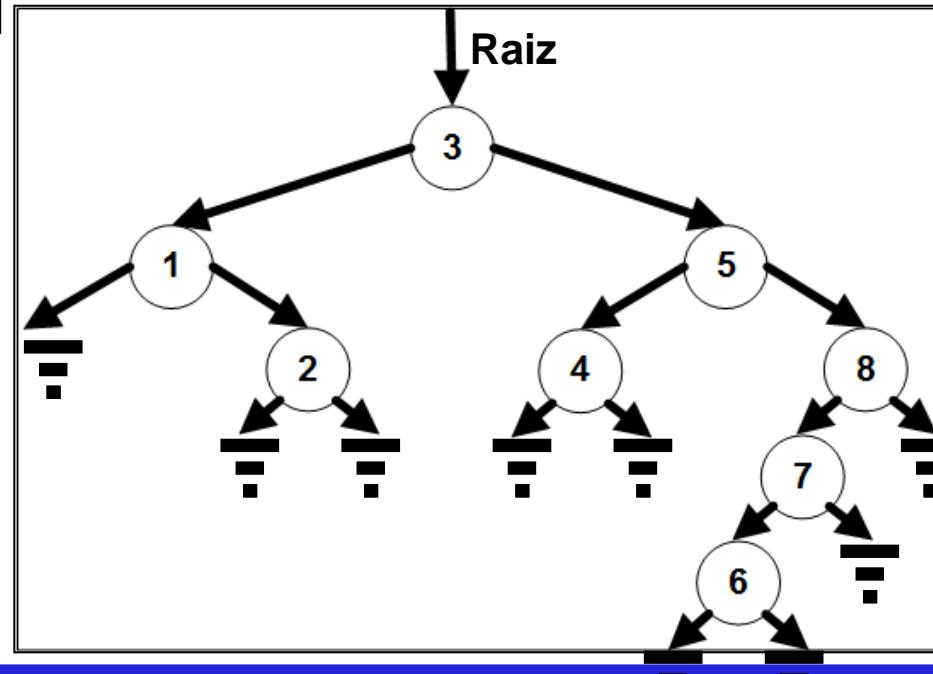


# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)

Vamos pesquisar se o  
4 está em nossa árvore



# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
```

```
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
```

```
    boolean resp;
```

```
    if (i == null) {
```

```
        resp = false;
```

```
    } else if (x == i.elemento) {
```

```
        resp = true;
```

```
    } else if (x < i.elemento) {
```

```
        resp = pesquisar(x, i.esq);
```

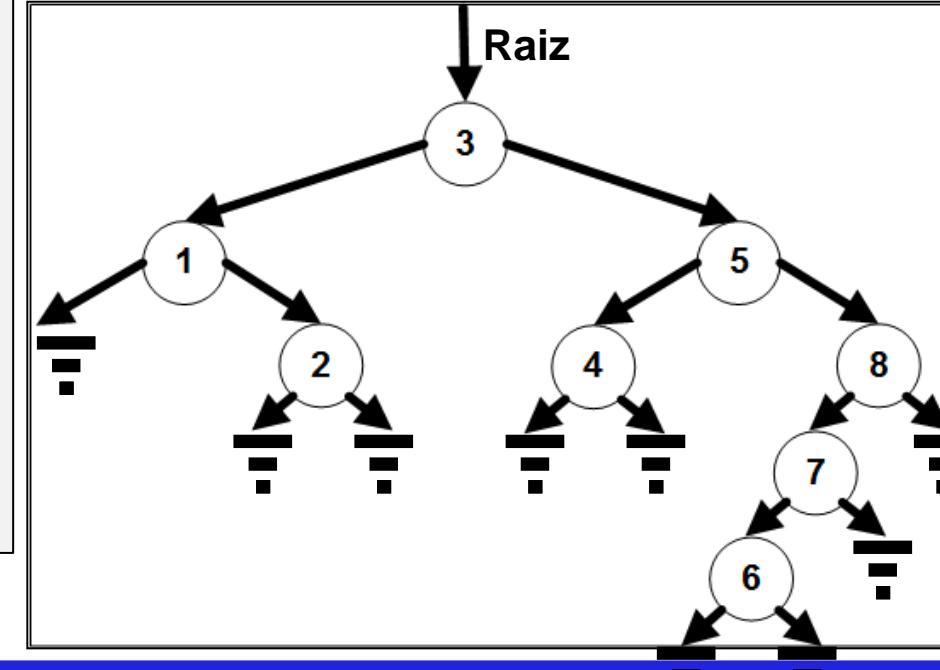
```
    } else {
```

```
        resp = pesquisar(x, i.dir);
```

```
    }
```

```
    return resp;
}
```

raiz n(3) x 4



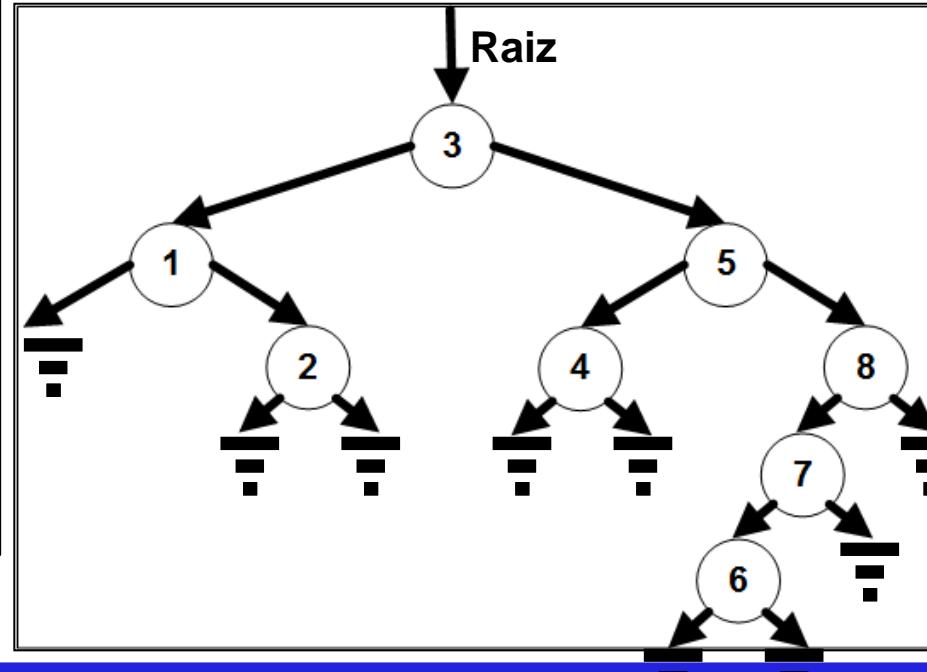
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3) x 4



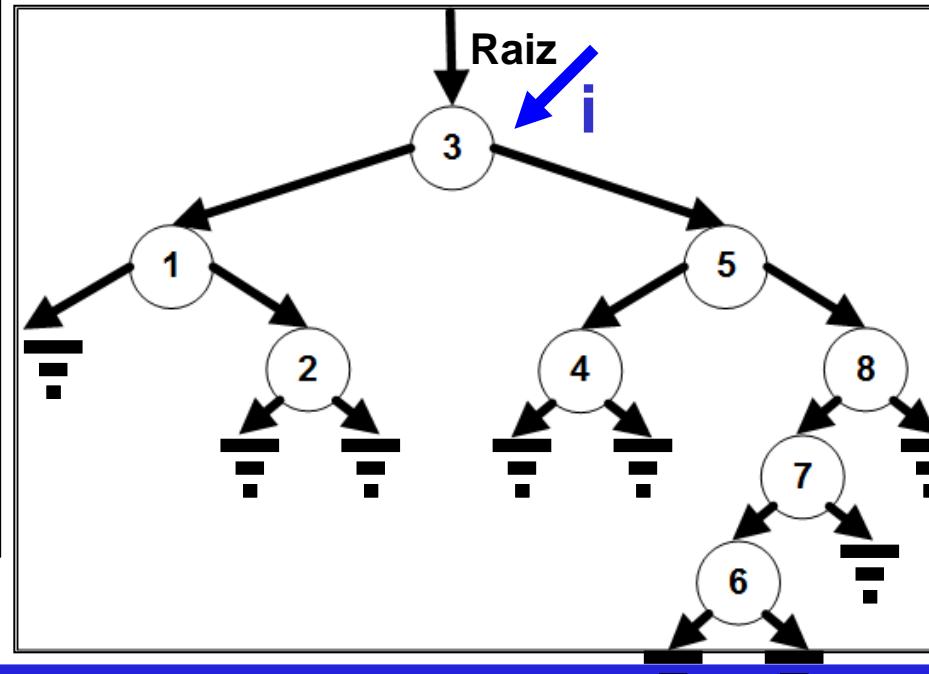
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3) x 4 x 4 i n(3)



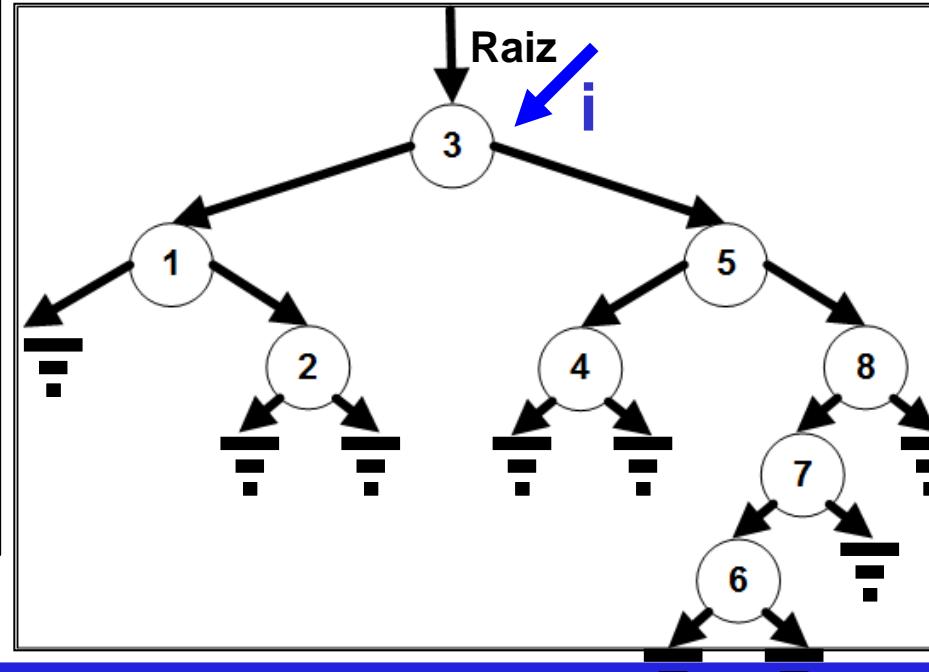
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3)   x 4   x 4   i n(3)  
 resp ?



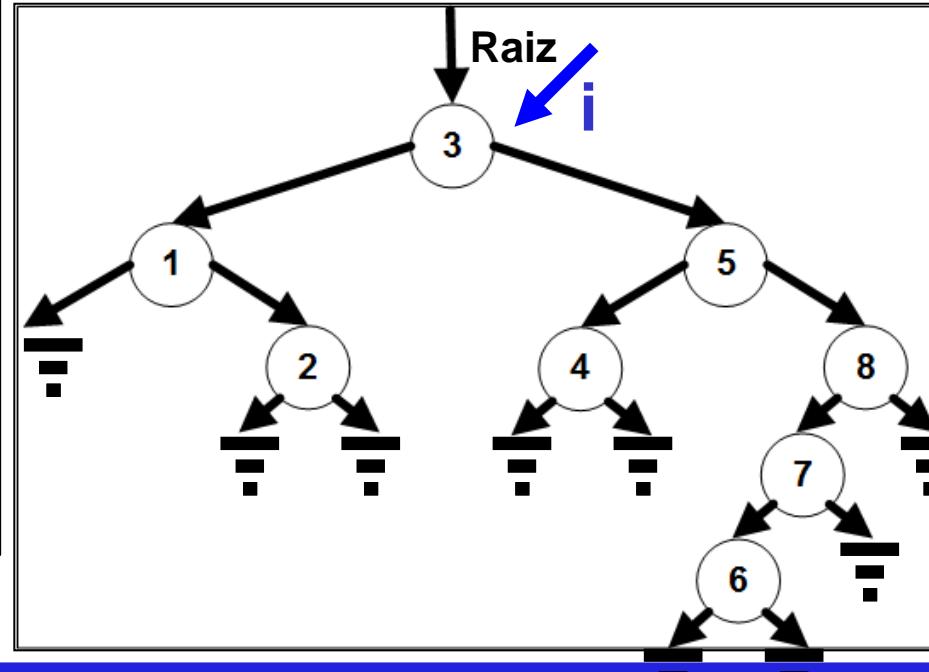
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
false: n(3) == null
```

raiz n(3)    x 4    x 4    i n(3)  
 resp ?



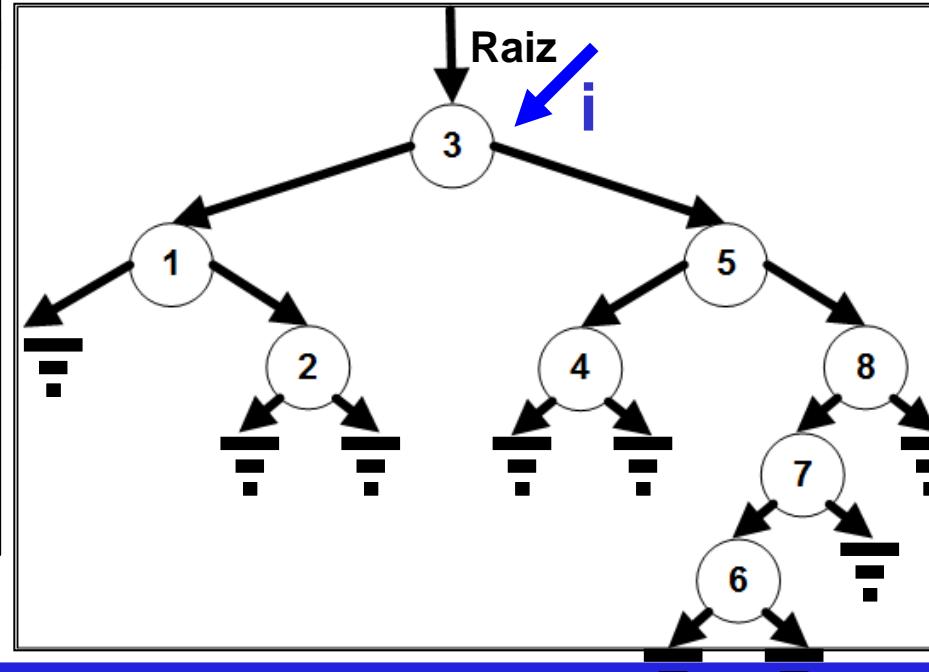
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
false: 4 == 3
```

raiz n(3) x 4 x 4 i n(3)  
 resp ?



# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

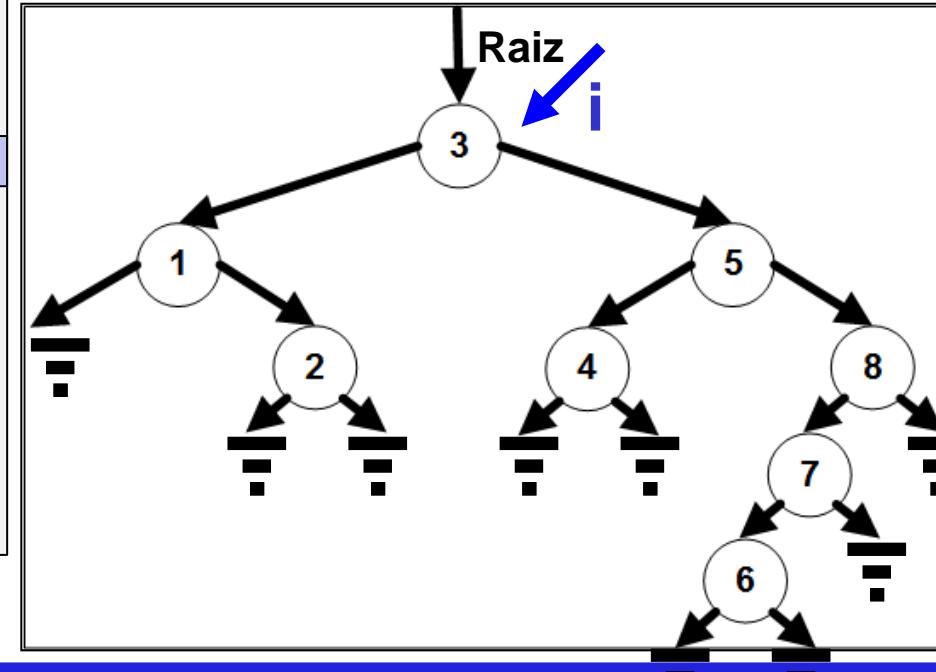
```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

false: 4 < 3

raiz n(3) x 4 x 4 i n(3)

resp ?



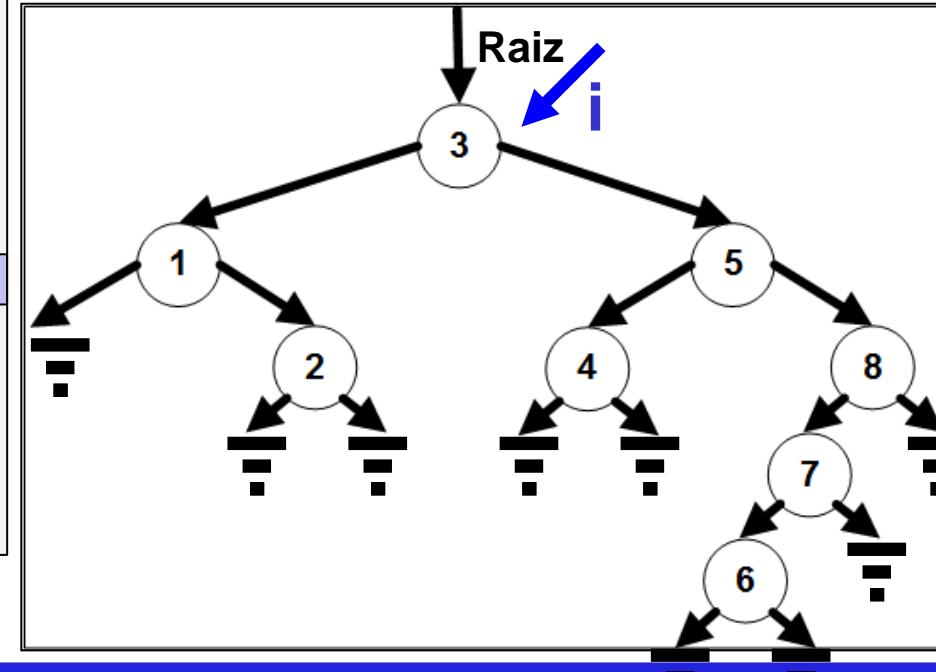
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3)   x 4   x 4   i n(3)  
 resp ?



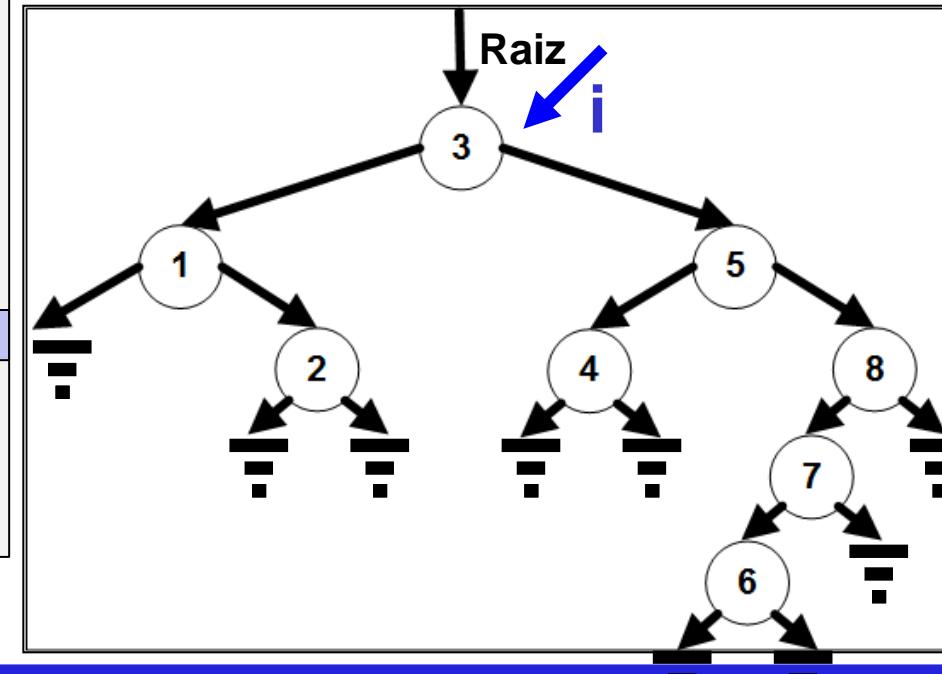
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3)   x 4   x 4   i n(3)  
 resp ?



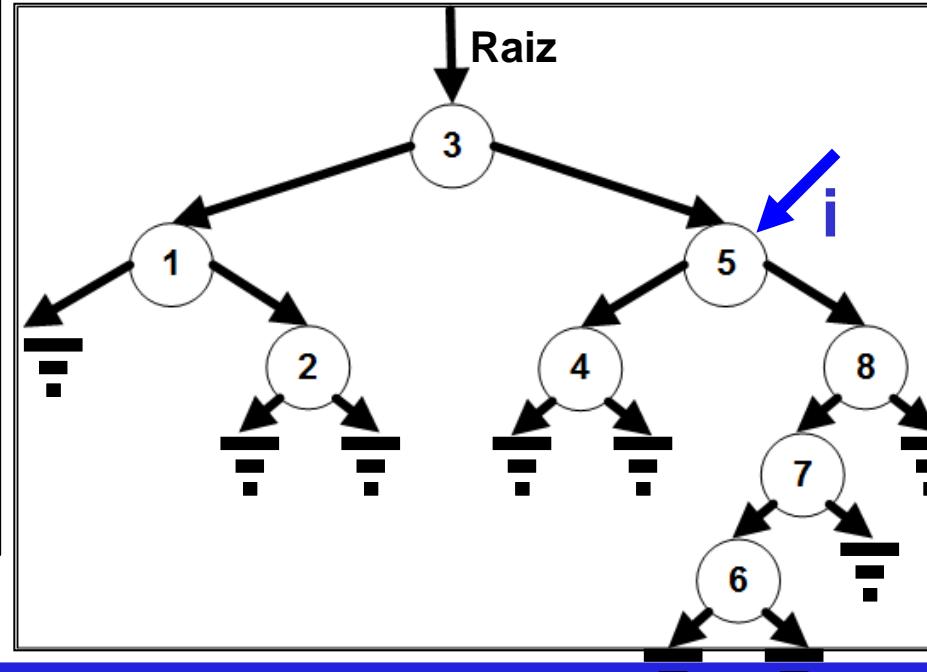
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3)   x 4   x 4   i n(3)  
 resp ?   x 4   i n(5)



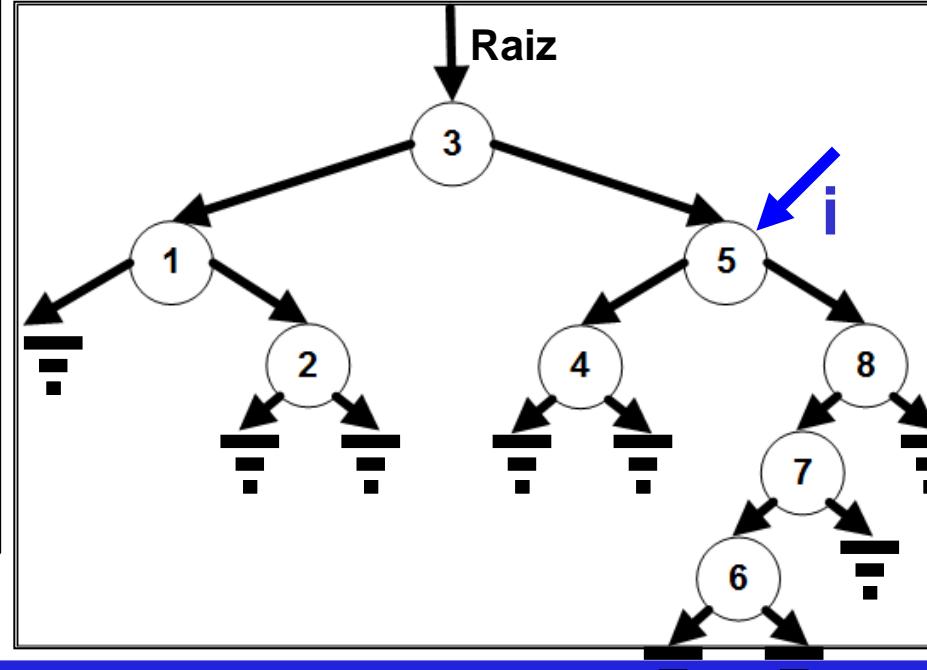
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3)   x 4   x 4   i n(3)  
 resp ?   x 4   i n(5)   resp ?

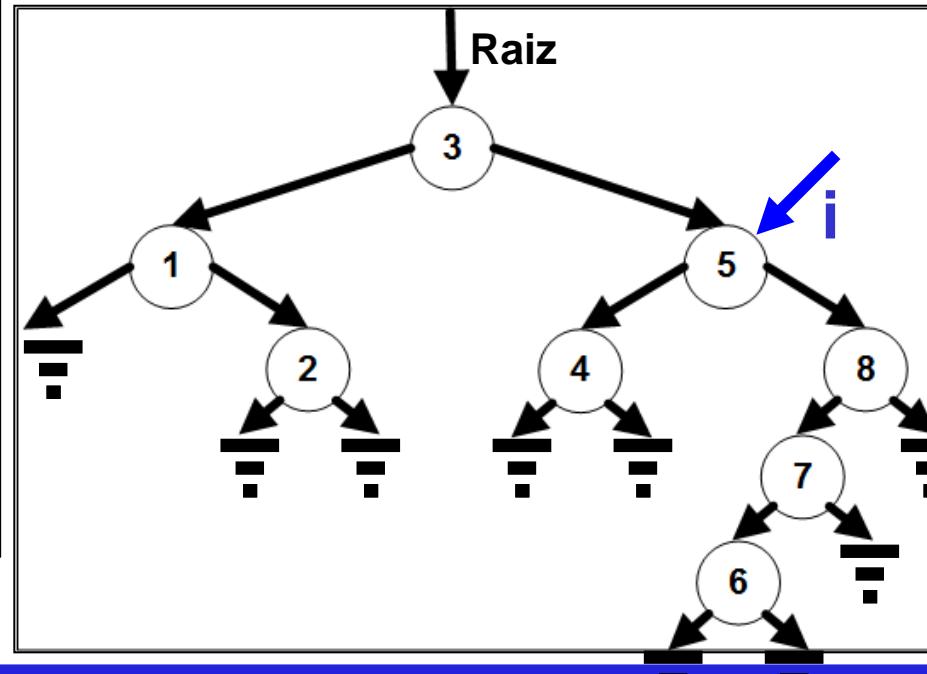
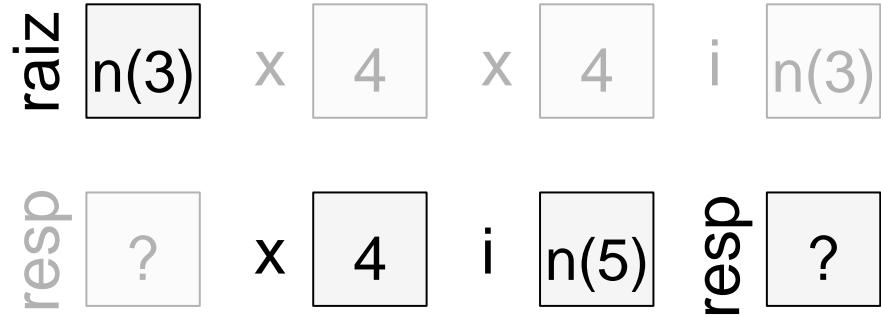


# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
false: n(5) == null
```



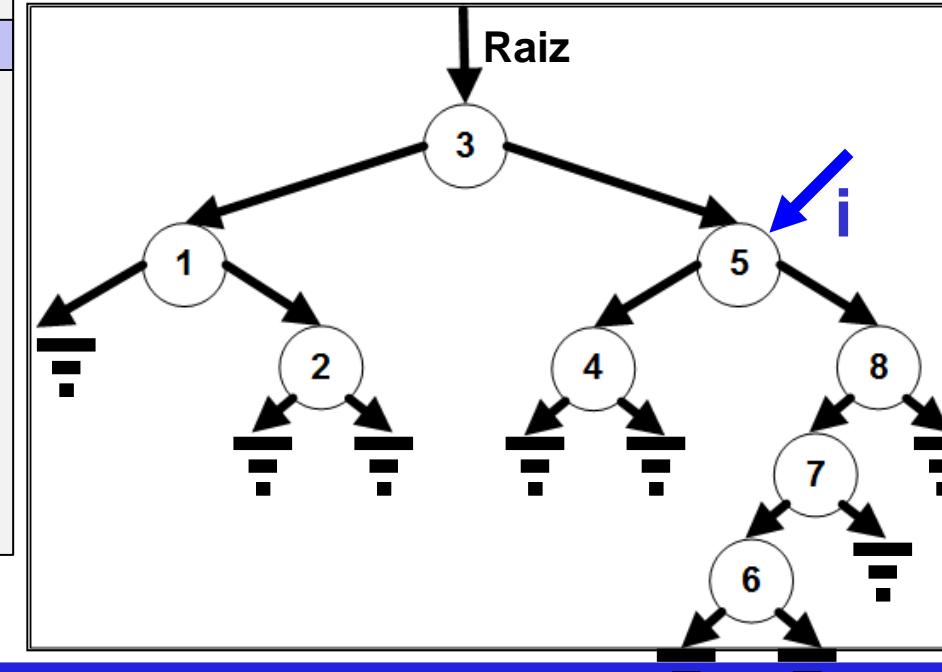
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
false: 4 == 5
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?



# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
```

```
    if (i == null) {
        resp = false;
```

```
    } else if (x == i.elemento) {
        resp = true;
```

```
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
```

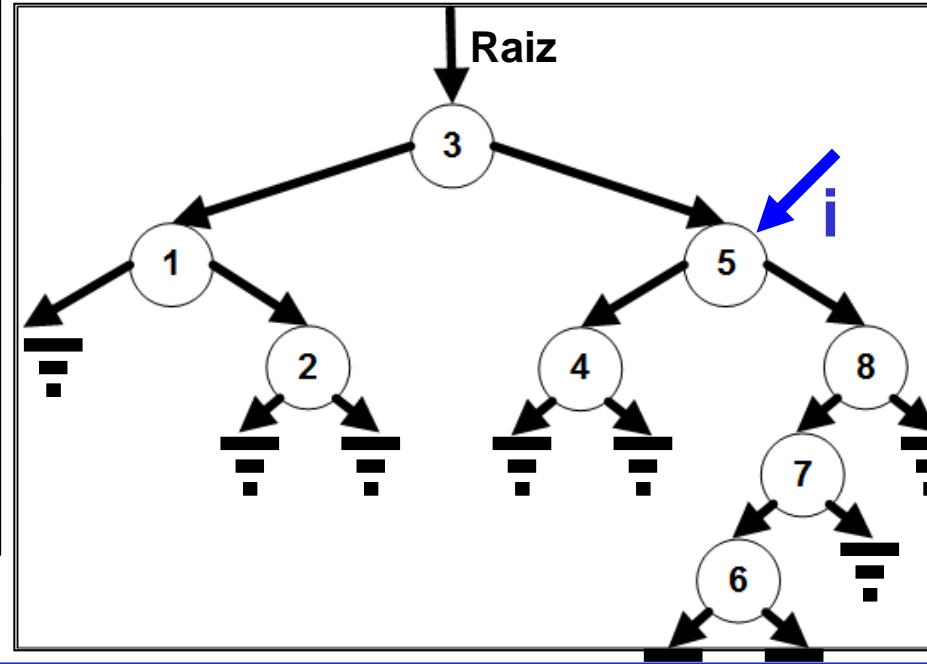
```
    } else {
        resp = pesquisar(x, i.dir);
```

```
    }
    return resp;
}
```

true: 4 < 5

raiz n(3) x 4 x 4 i n(3)

resp ? x 4 i n(5) resp ?



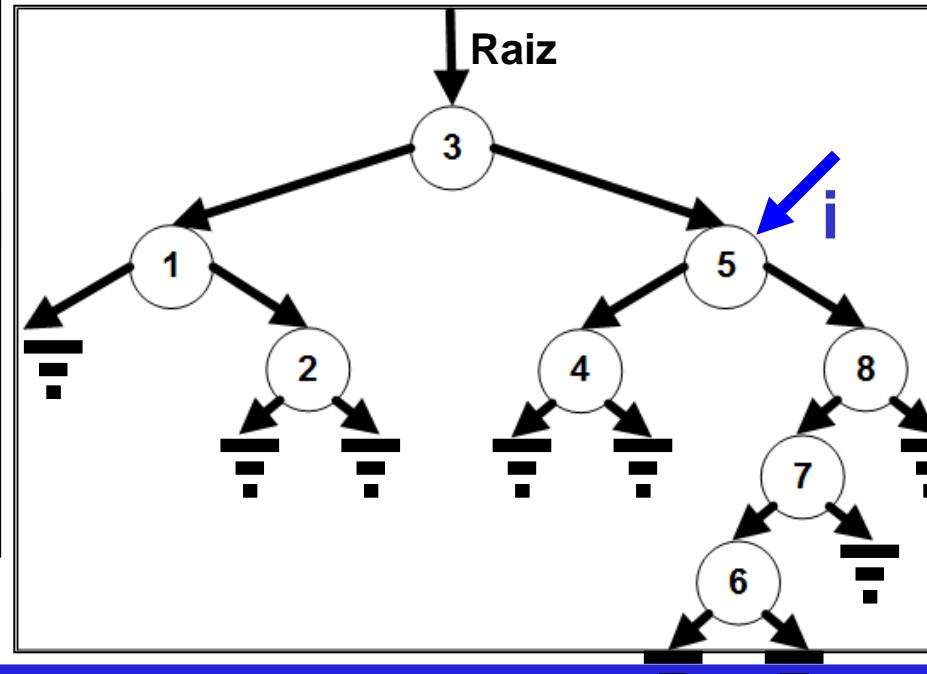
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?

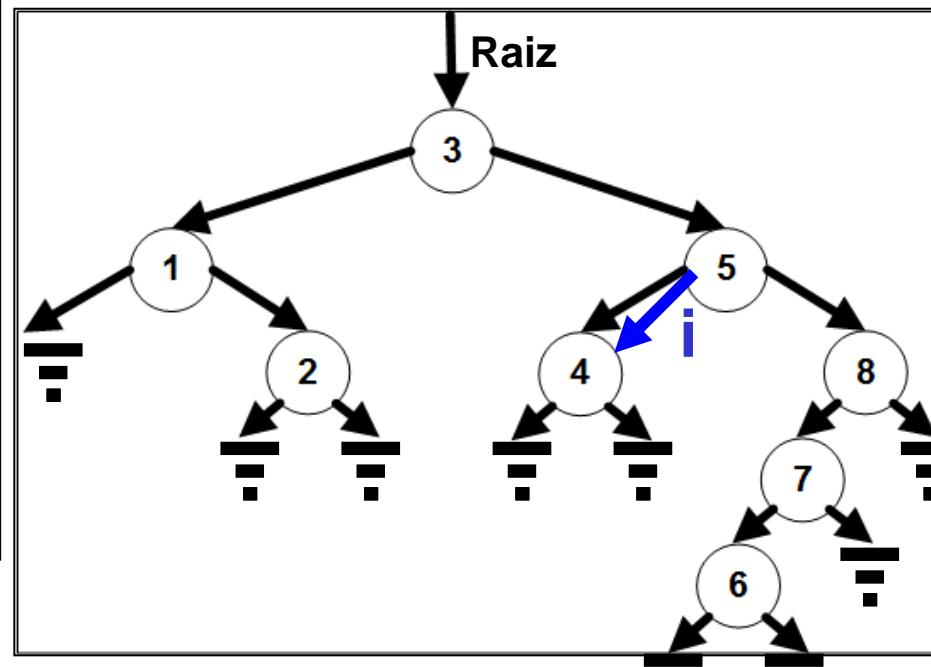
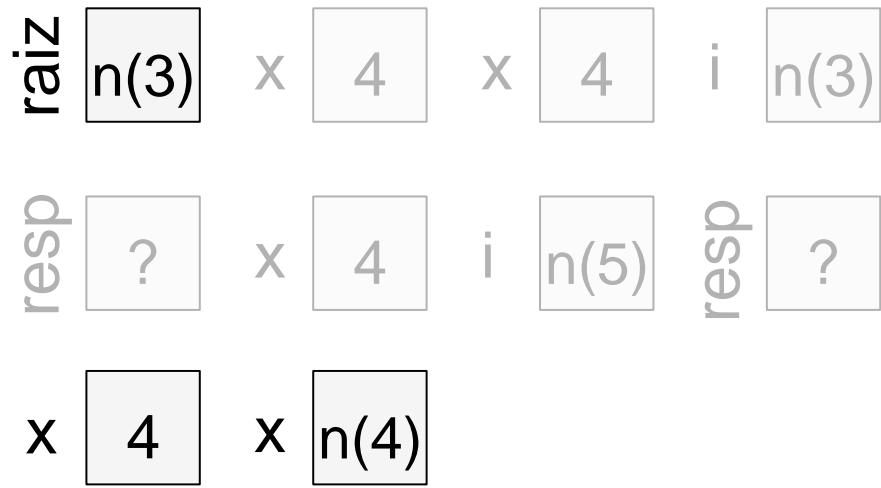


# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```



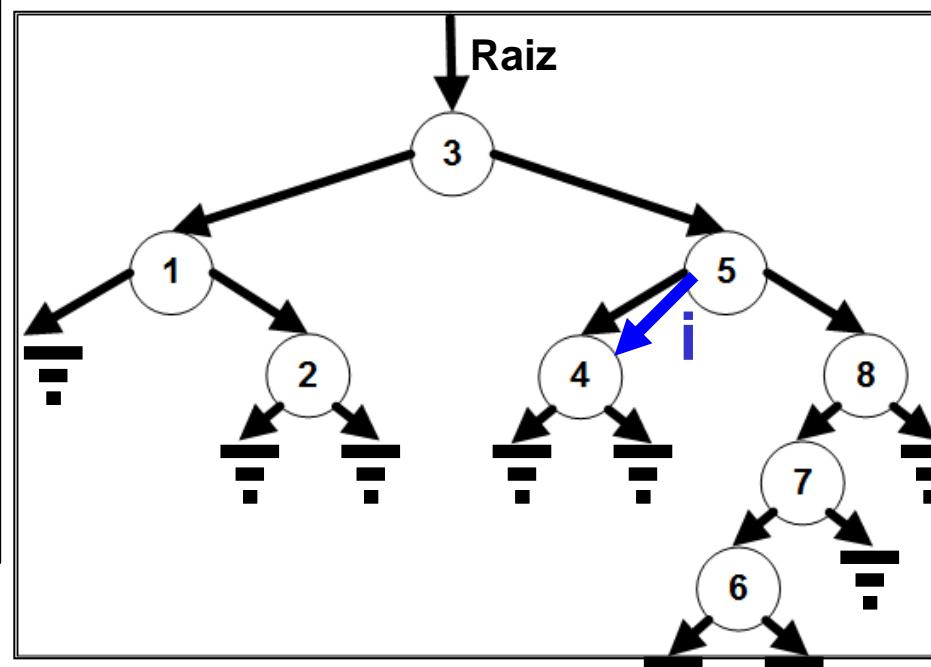
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?
x	4	x	n(4)	resp	?		



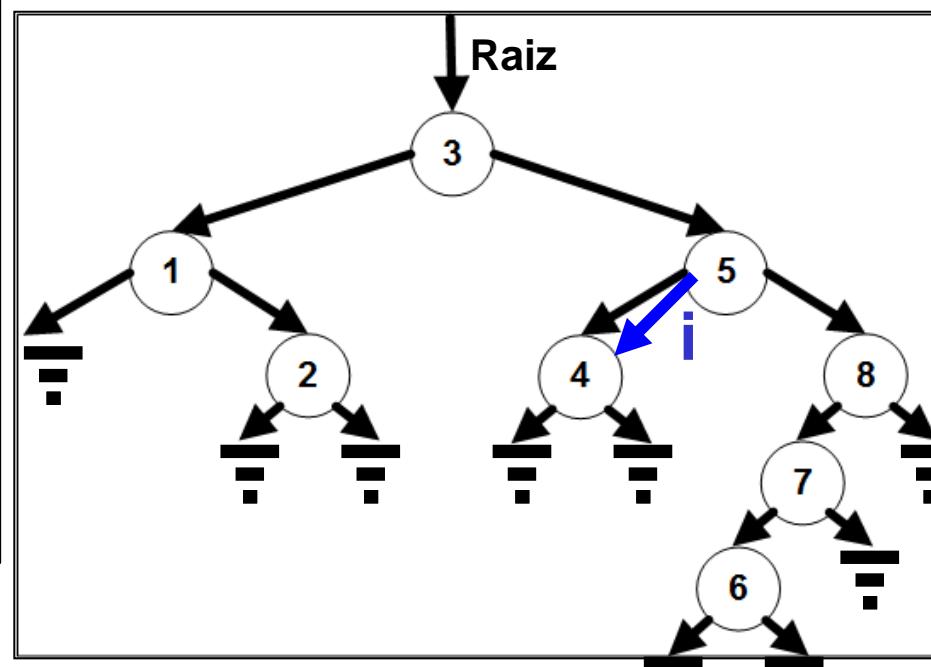
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
false: n(4) == null
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?
x	4	x	n(4)	resp	?		



# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
```

```
    boolean resp;
    if (i == null) {
        resp = false;
```

```
} else if (x == i.elemento) {
```

```
    resp = true;
```

```
} else if (x < i.elemento) {
    resp = pesquisar(x, i.esq);
```

```
} else {
```

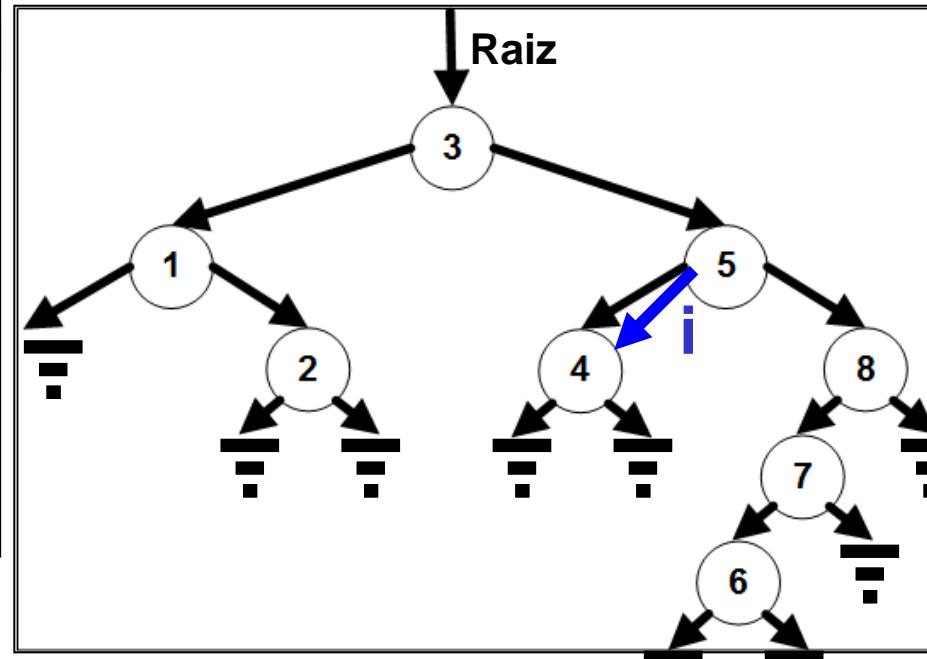
```
    resp = pesquisar(x, i.dir);
```

```
}
```

```
return resp;
```

true: 4 == 4

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?
x	4	x	n(4)	resp	?		



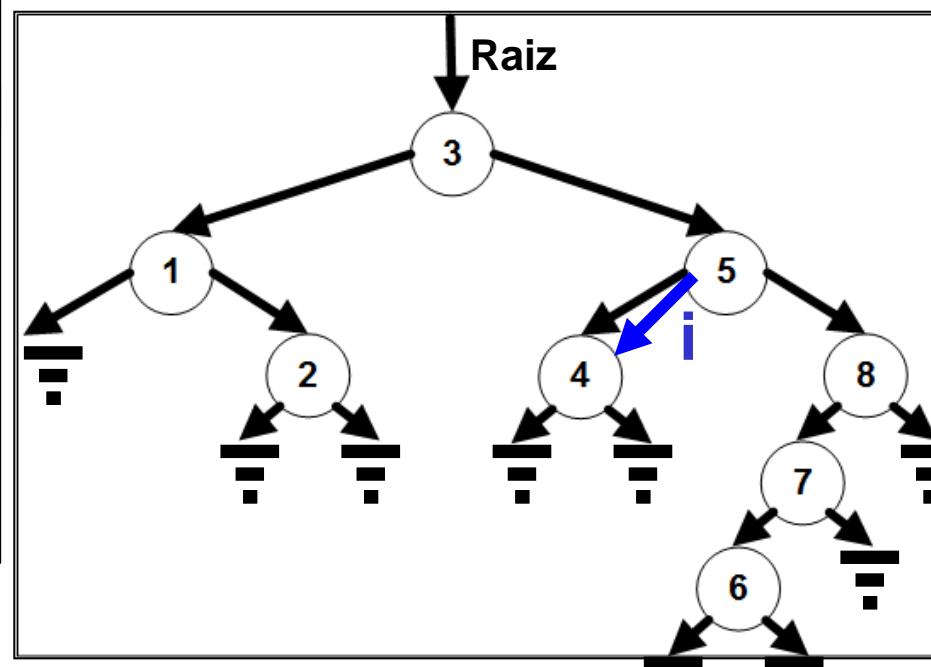
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?
x	4	x	n(4)	resp	true		



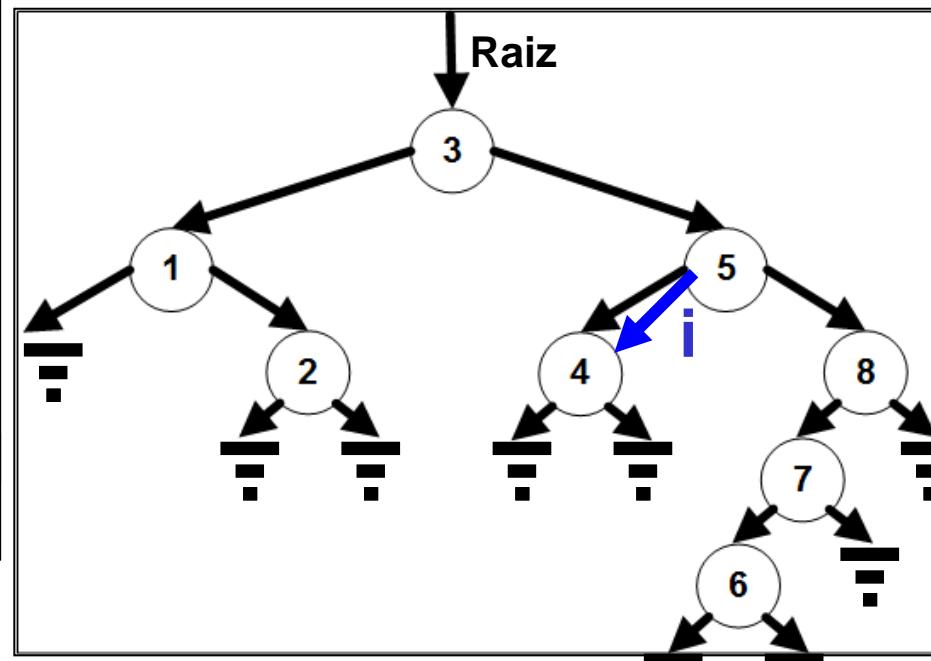
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?
x	4	x	n(4)	resp	true		



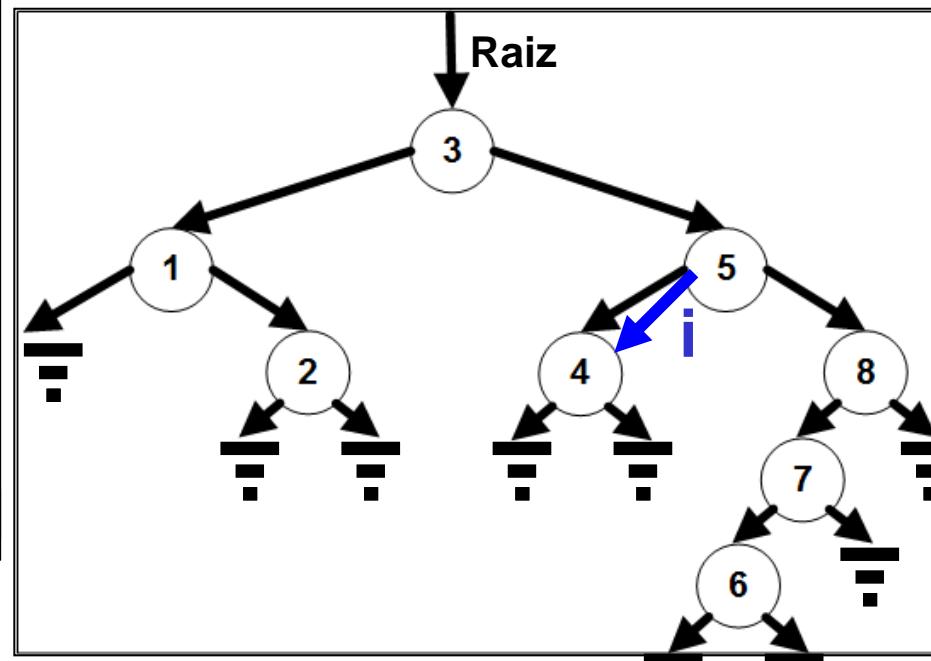
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	?
x	4	x	n(4)	resp	true		



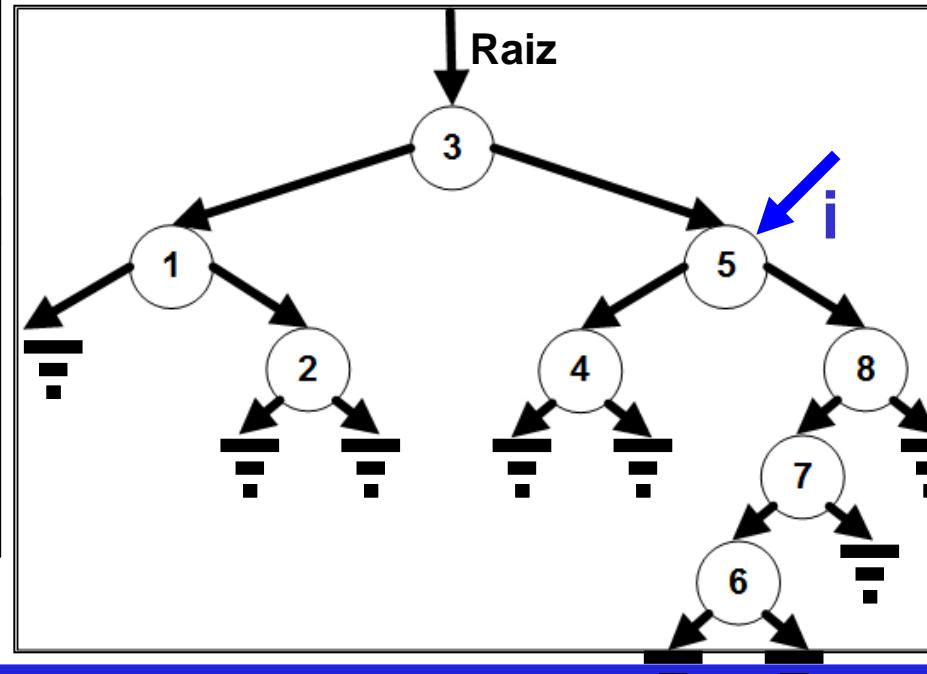
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	true



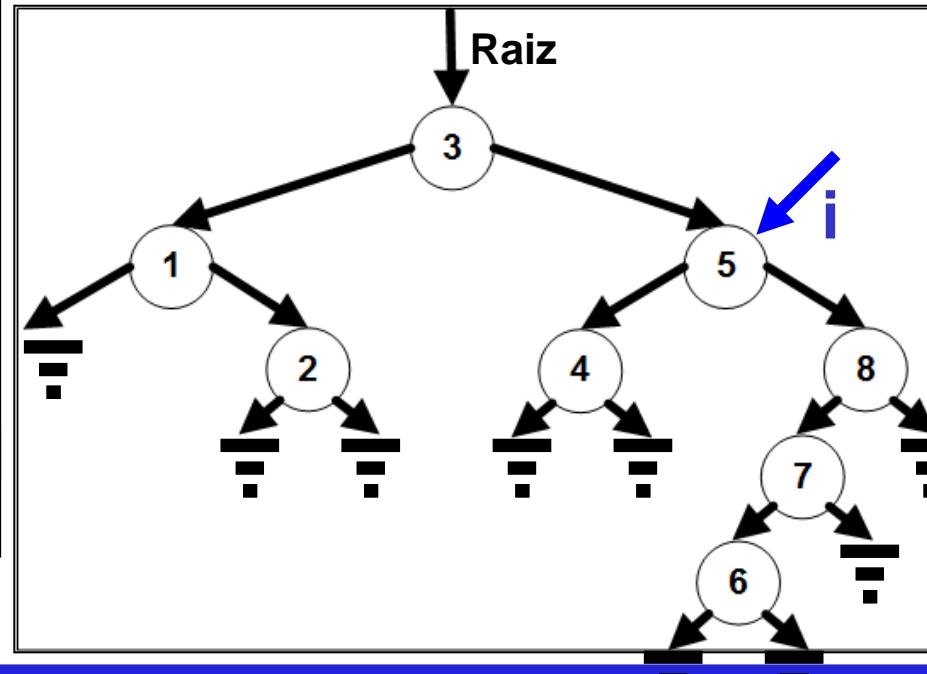
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	true



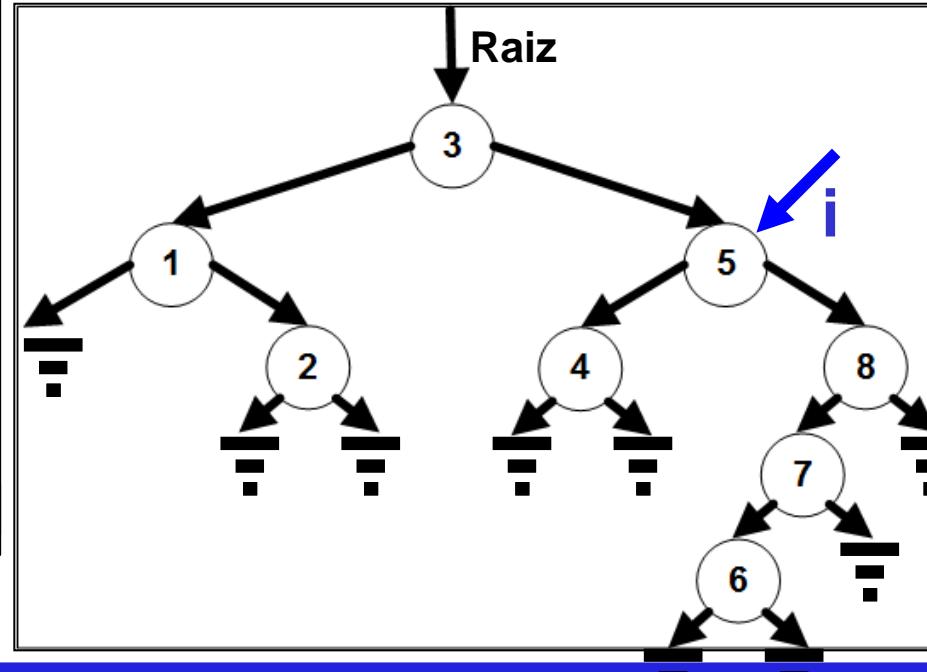
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	n(3)	x	4	x	4	i	n(3)
resp	?	x	4	i	n(5)	resp	true



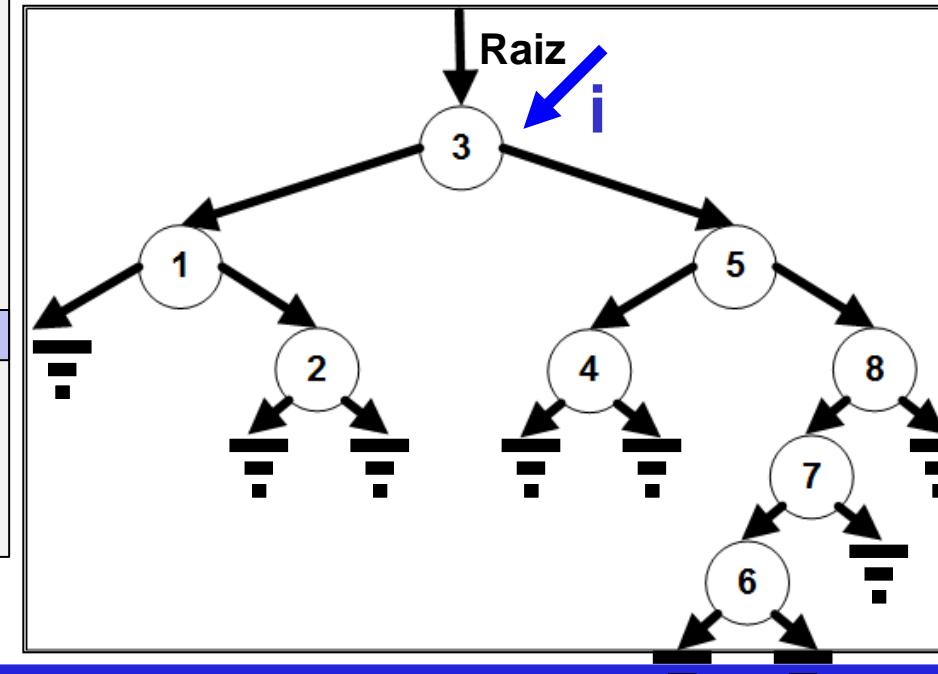
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3)   x 4   x 4   i n(3)  
 resp true



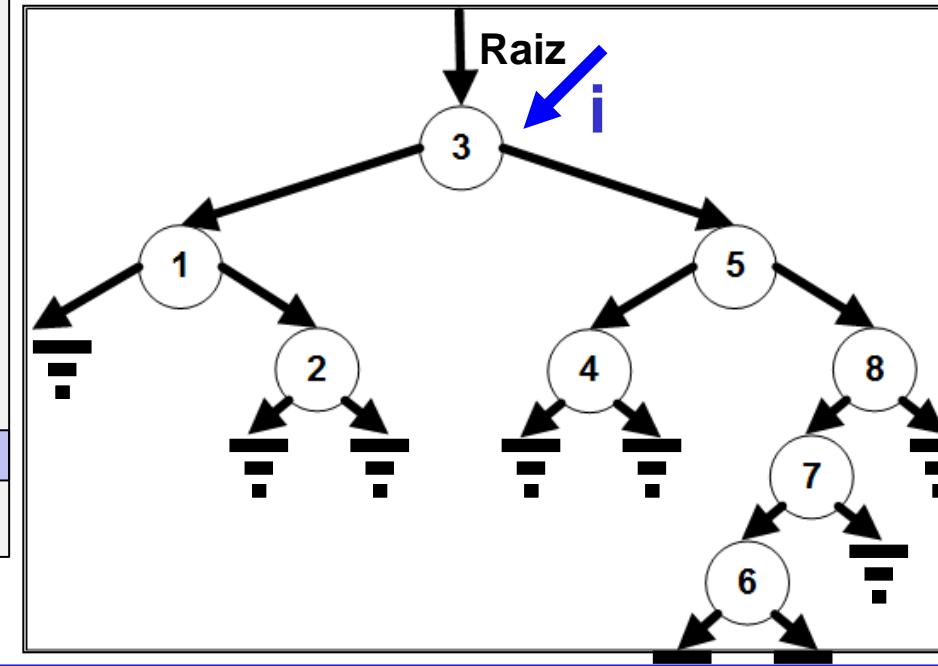
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}
```

```
private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3)    x 4    x 4    i n(3)  
resp true



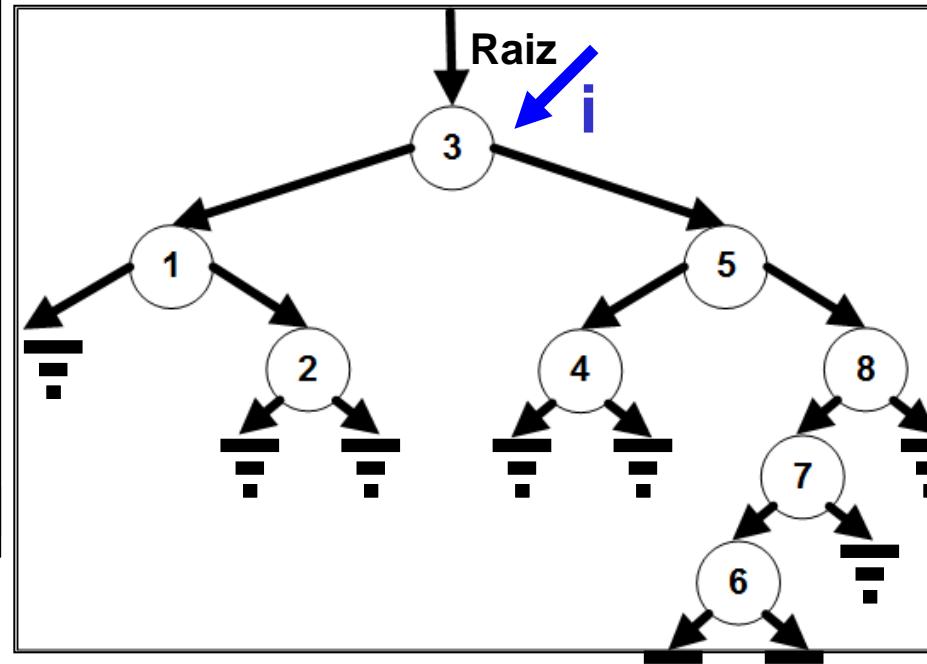
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz	<input type="text" value="n(3)"/>	x	<input type="text" value="4"/>	x	<input type="text" value="4"/>	i	<input type="text" value="n(3)"/>
resp	<input type="text" value="true"/>						



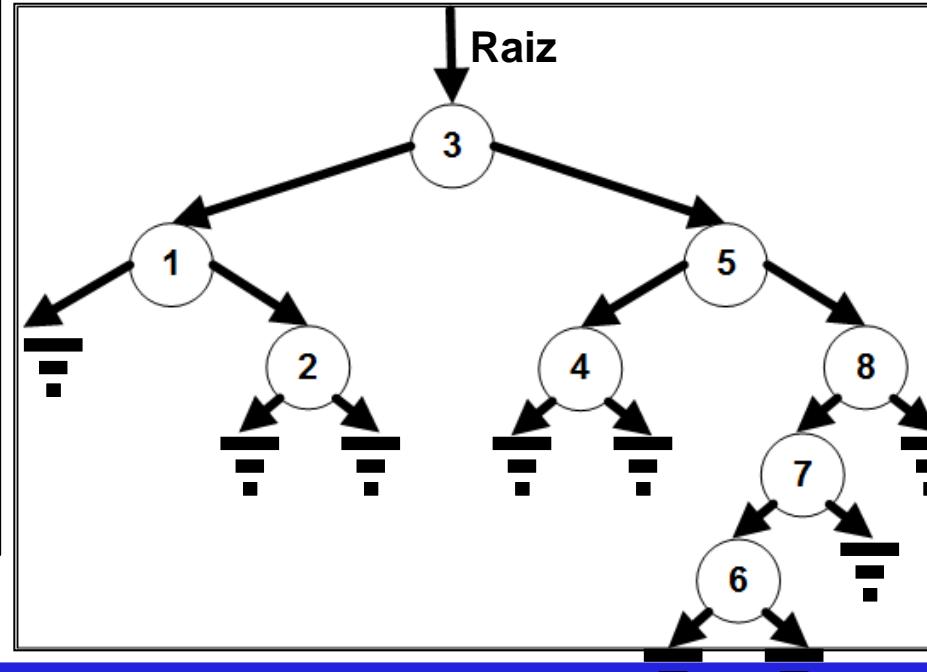
# Algoritmo em Java - Classe Árvore Binária

```
//Pesquisar (4)

public boolean pesquisar(int x) {
    return pesquisar(x, raiz);
}

private boolean pesquisar(int x, No i) {
    boolean resp;
    if (i == null) {
        resp = false;
    } else if (x == i.elemento) {
        resp = true;
    } else if (x < i.elemento) {
        resp = pesquisar(x, i.esq);
    } else {
        resp = pesquisar(x, i.dir);
    }
    return resp;
}
```

raiz n(3) x 4



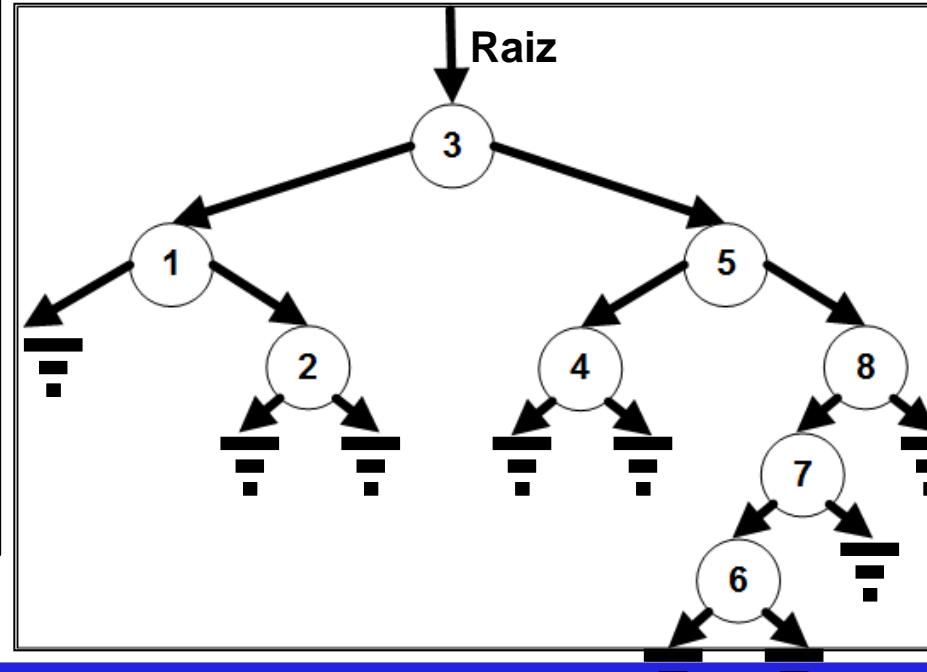
# Algoritmo em Java - Classe Árvore Binária

//Pesquisar (4)

```
public boolean pesquisar(int x) {  
    return pesquisar(x, raiz);  
}
```

```
private boolean pesquisar(int x, No i) {  
    boolean resp;  
    if (i == null) {  
        resp = false;  
    } else if (x == i.elemento) {  
        resp = true;  
    } else if (x < i.elemento) {  
        resp = pesquisar(x, i.esq);  
    } else {  
        resp = pesquisar(x, i.dir);  
    }  
    return resp;  
}
```

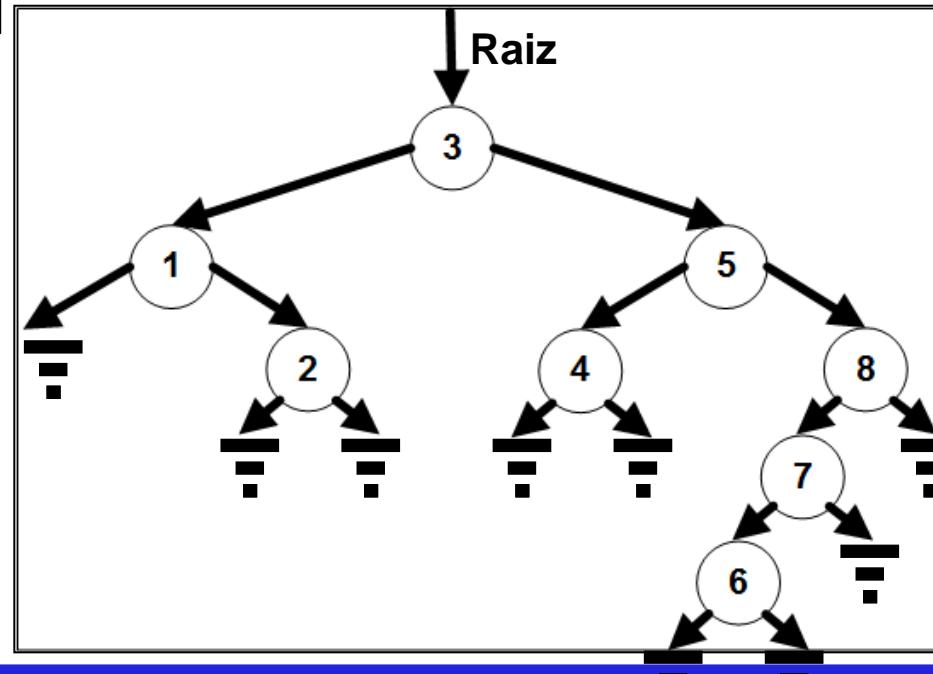
raiz  
n(3)



# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)



# Remoção em Árvore Binária

- Funcionamento básico:

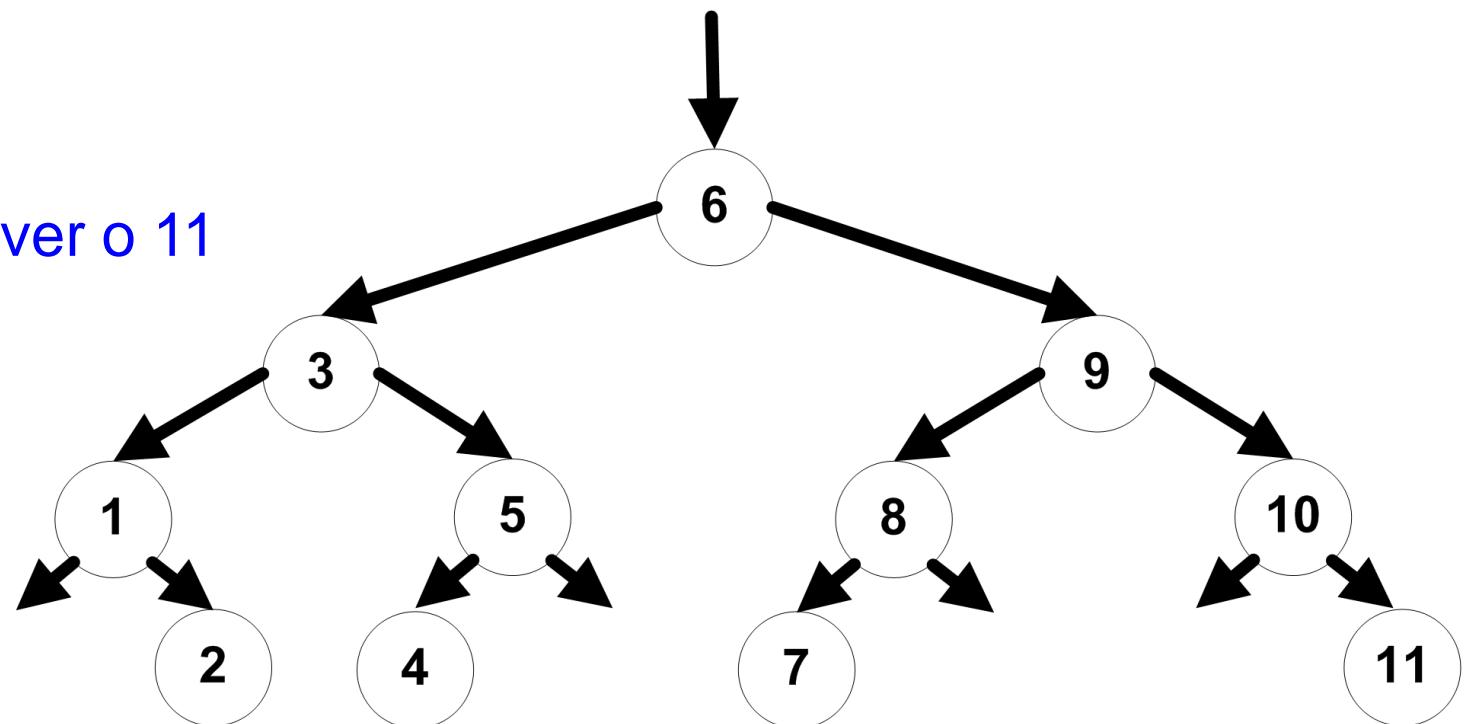
**(1)** Se o elemento estiver em uma **folha**, removê-la

# Remoção em Árvore Binária

- Funcionamento básico:

(1) Se o elemento estiver em uma **folha**, removê-la

Exemplo: Remover o 11

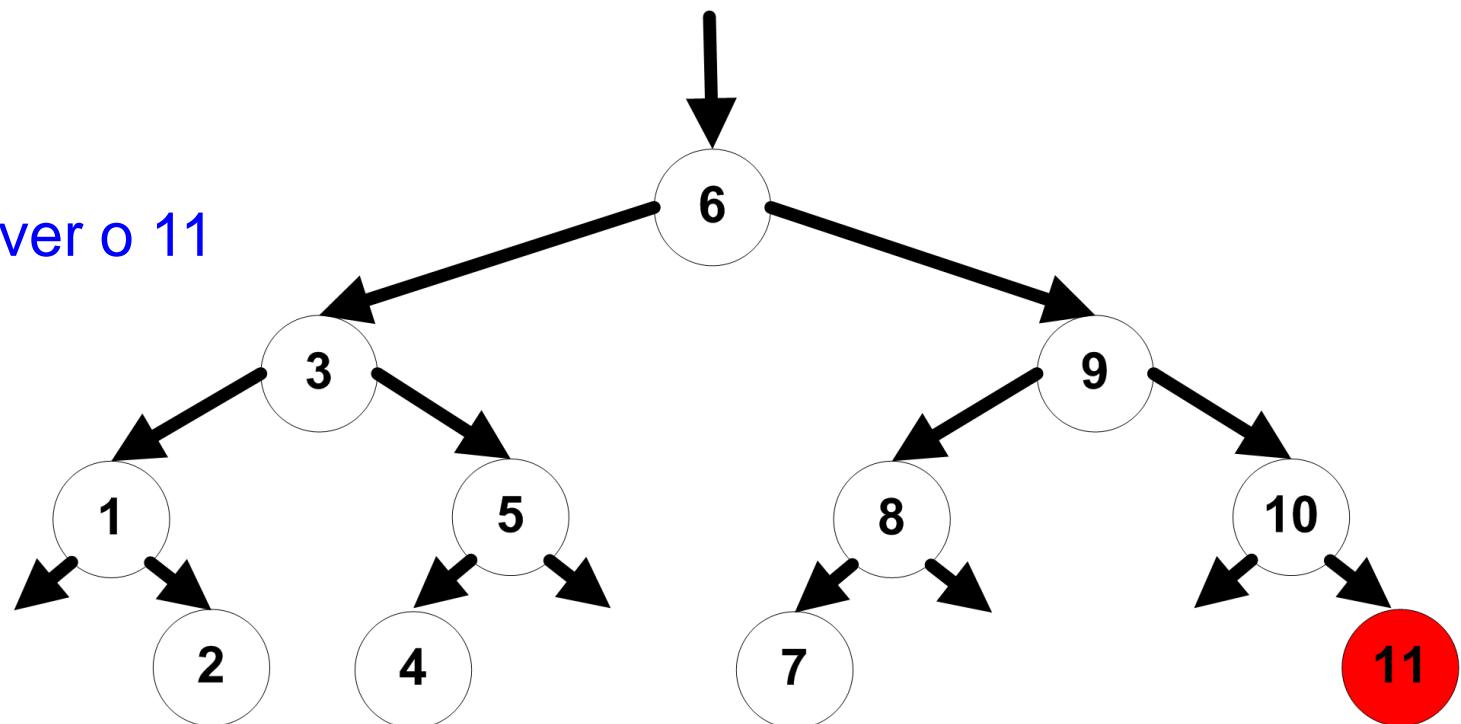


# Remoção em Árvore Binária

Funcionamento básico:

- (1) Se o elemento estiver em uma **folha**, removê-la

Exemplo: Remover o 11

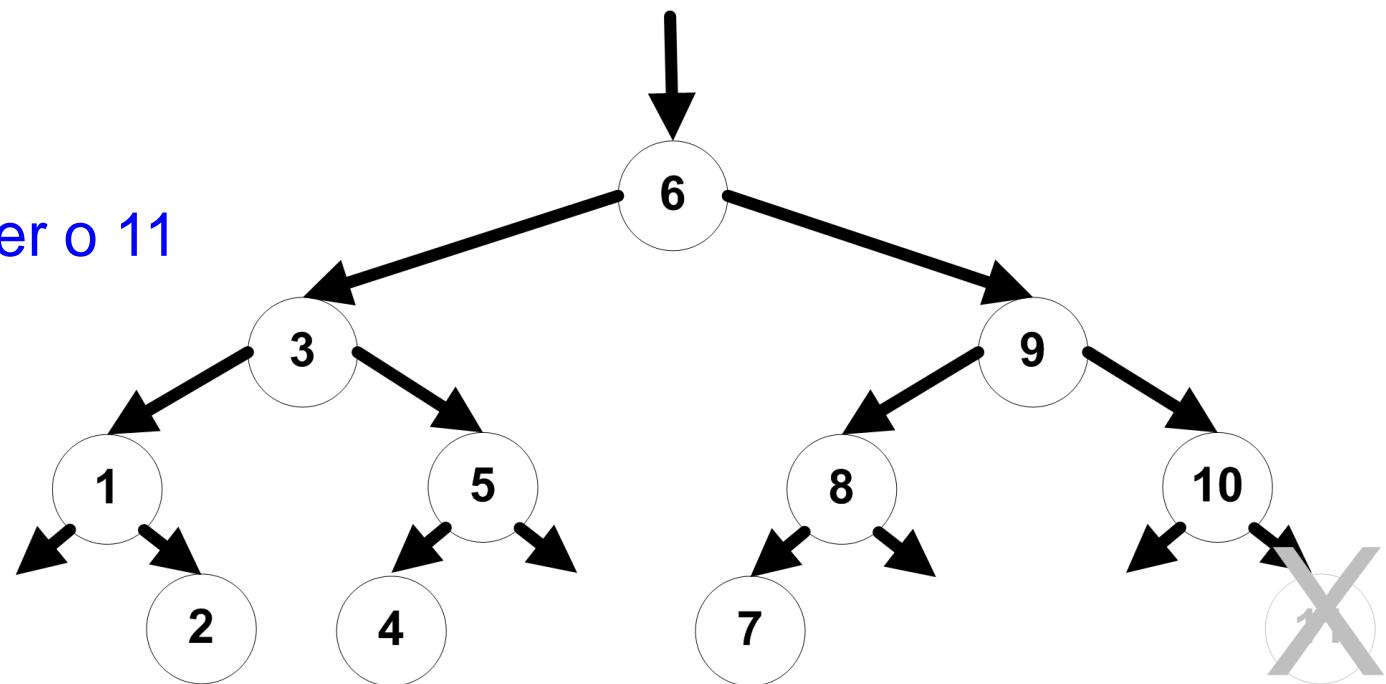


# Remoção em Árvore Binária

- Funcionamento básico:

**(1)** Se o elemento estiver em uma **folha**, removê-la

Exemplo: Remover o 11



# Remoção em Árvore Binária

- Funcionamento básico:

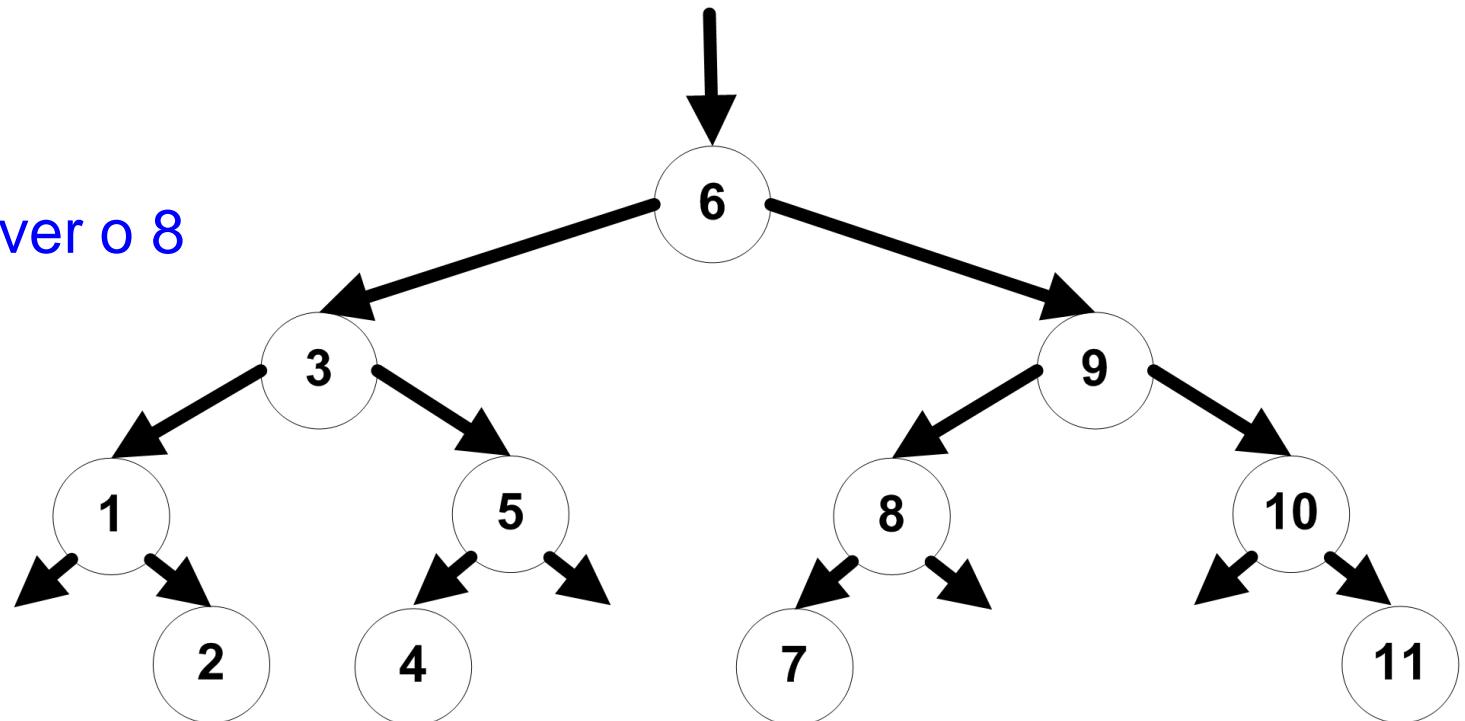
**(2)** Senão, se o elemento estiver em um **nó intermediário com um único filho**, remover o nó e fazer com que seu pai aponte para seu filho

# Remoção em Árvore Binária

- Funcionamento básico:

(2) Senão, se o elemento estiver em um **nó intermediário com um único filho**, remover o nó e fazer com que seu pai aponte para seu filho

Exemplo: Remover o 8

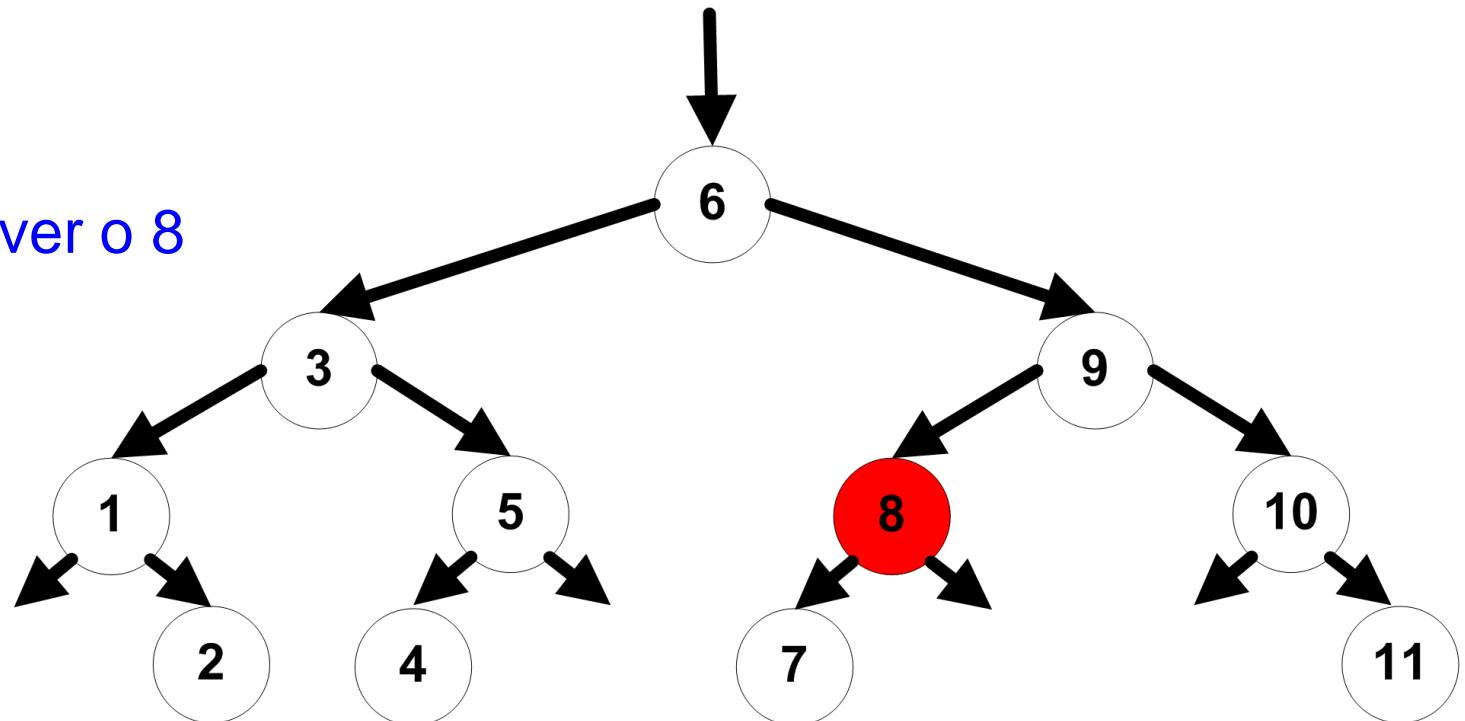


# Remoção em Árvore Binária

- Funcionamento básico:

(2) Senão, se o elemento estiver em um **nó intermediário com um único filho**, remover o nó e fazer com que seu pai aponte para seu filho

Exemplo: Remover o 8

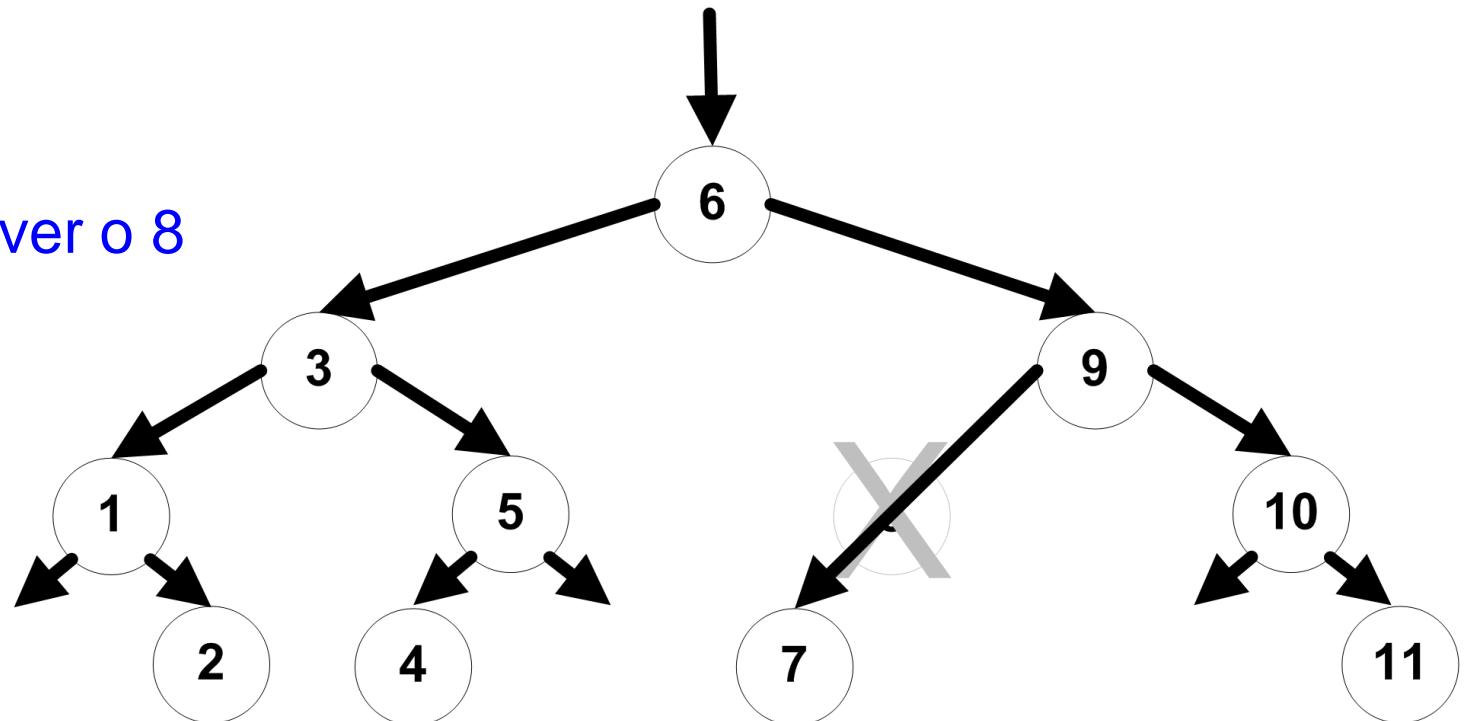


# Remoção em Árvore Binária

- Funcionamento básico:

(2) Senão, se o elemento estiver em um **nó intermediário com um único filho**, remover o nó e fazer com que seu pai aponte para seu filho

Exemplo: Remover o 8

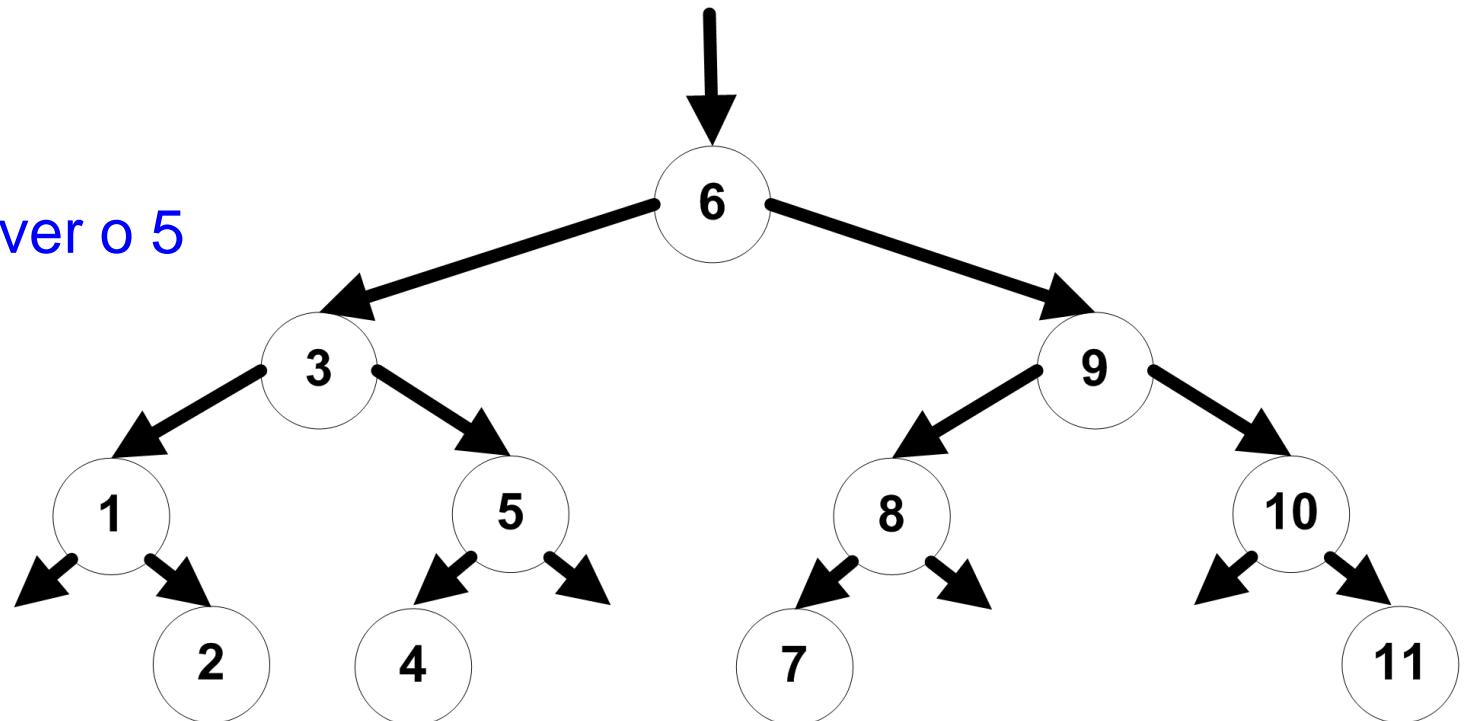


# Remoção em Árvore Binária

- Funcionamento básico:

(2) Senão, se o elemento estiver em um **nó intermediário com um único filho**, remover o nó e fazer com que seu pai aponte para seu filho

Exemplo: Remover o 5

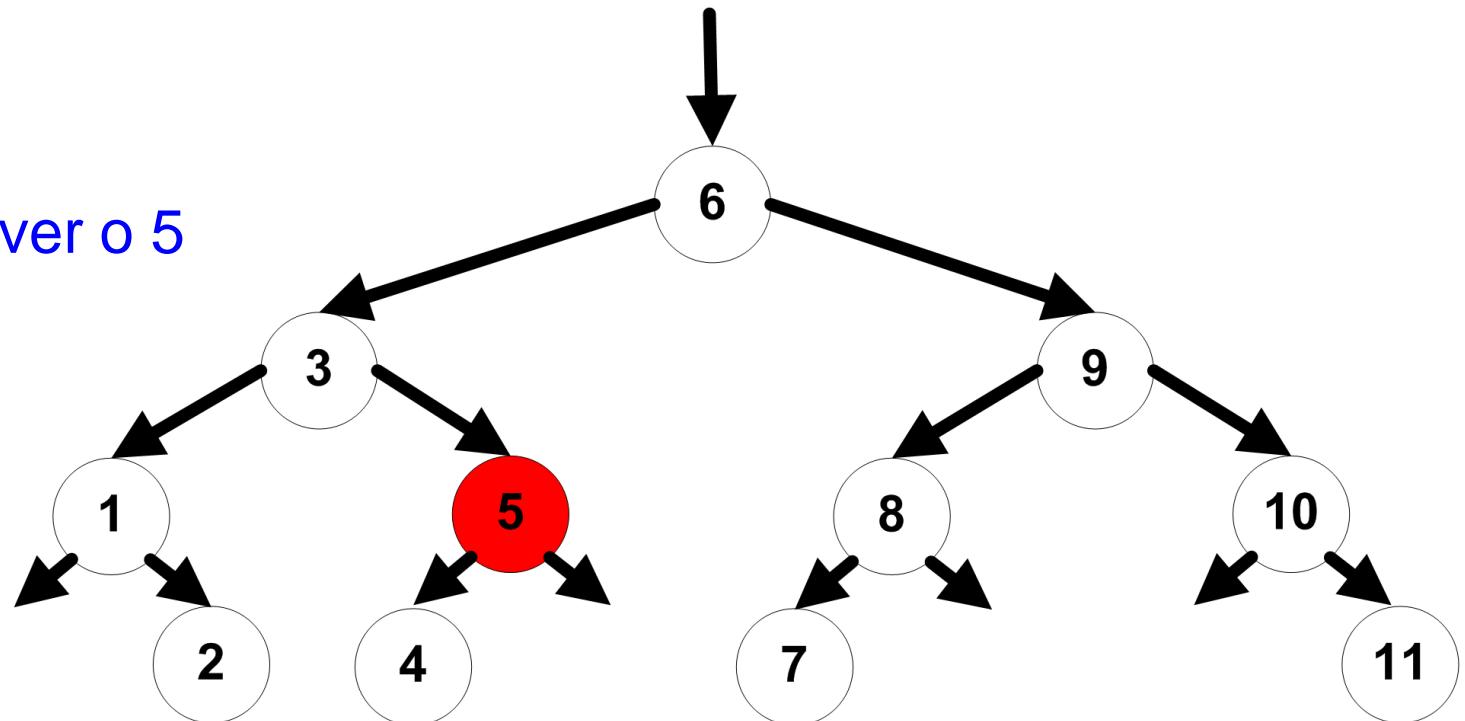


# Remoção em Árvore Binária

- Funcionamento básico:

(2) Senão, se o elemento estiver em um **nó intermediário com um único filho**, remover o nó e fazer com que seu pai aponte para seu filho

Exemplo: Remover o 5

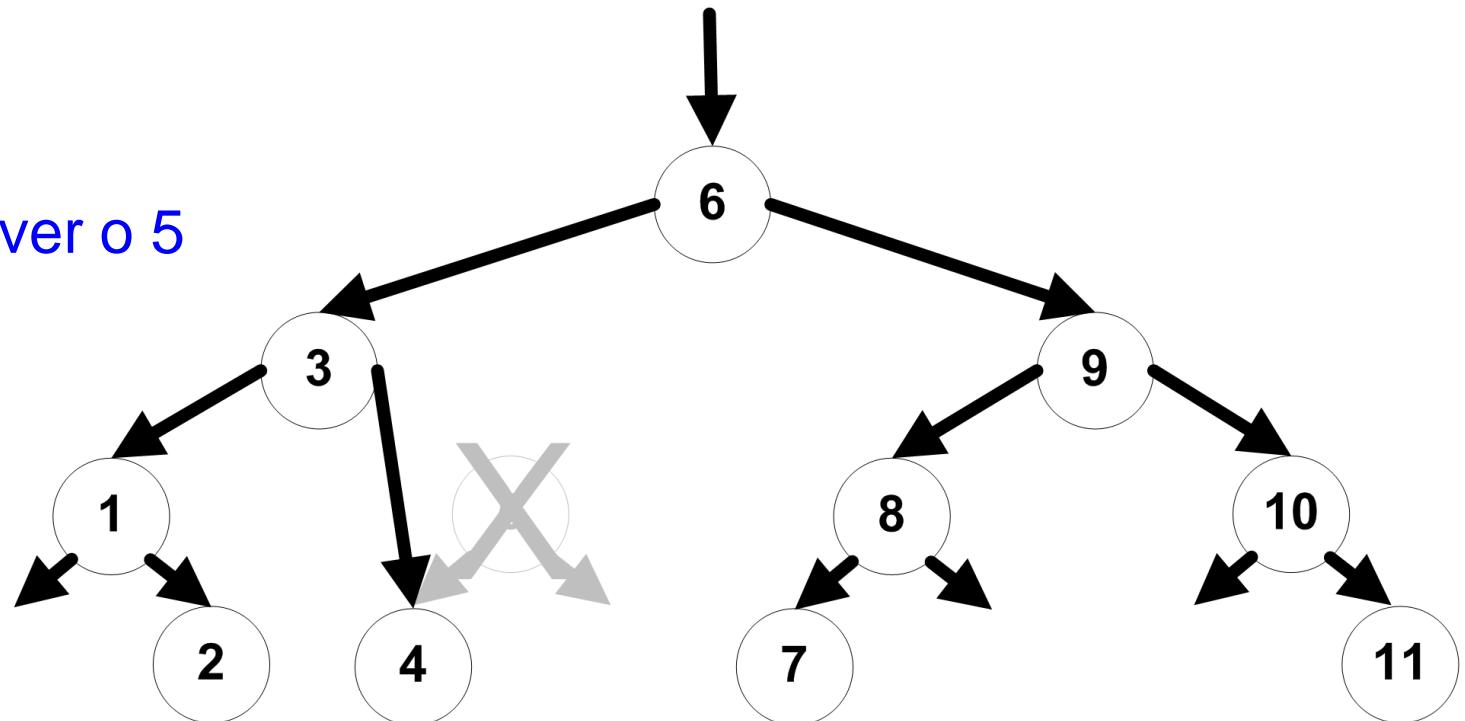


# Remoção em Árvore Binária

- Funcionamento básico:

(2) Senão, se o elemento estiver em um **nó intermediário com um único filho**, remover o nó e fazer com que seu pai aponte para seu filho

Exemplo: Remover o 5



# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

# Remoção em Árvore Binária

- Funcionamento básico:

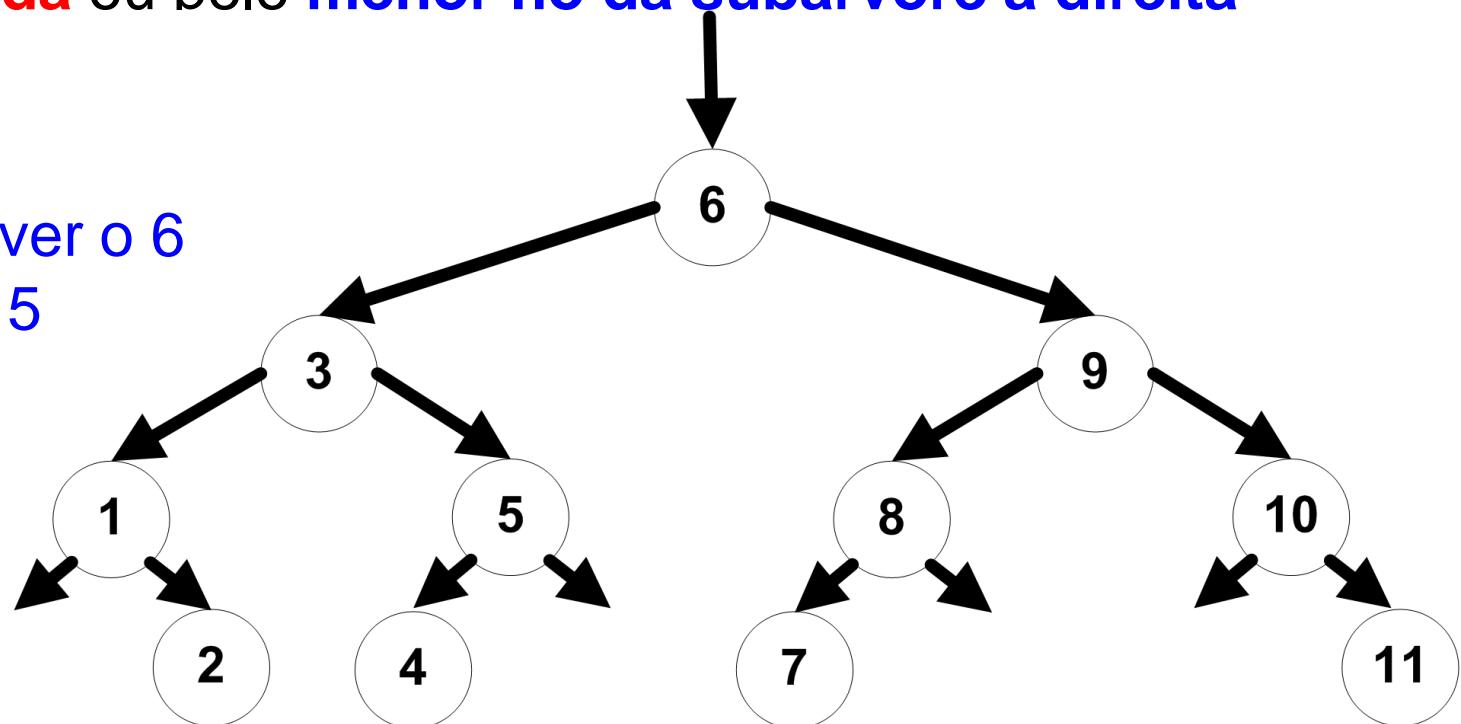
(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

Exemplo: Remover o 6 e substituir pelo 5

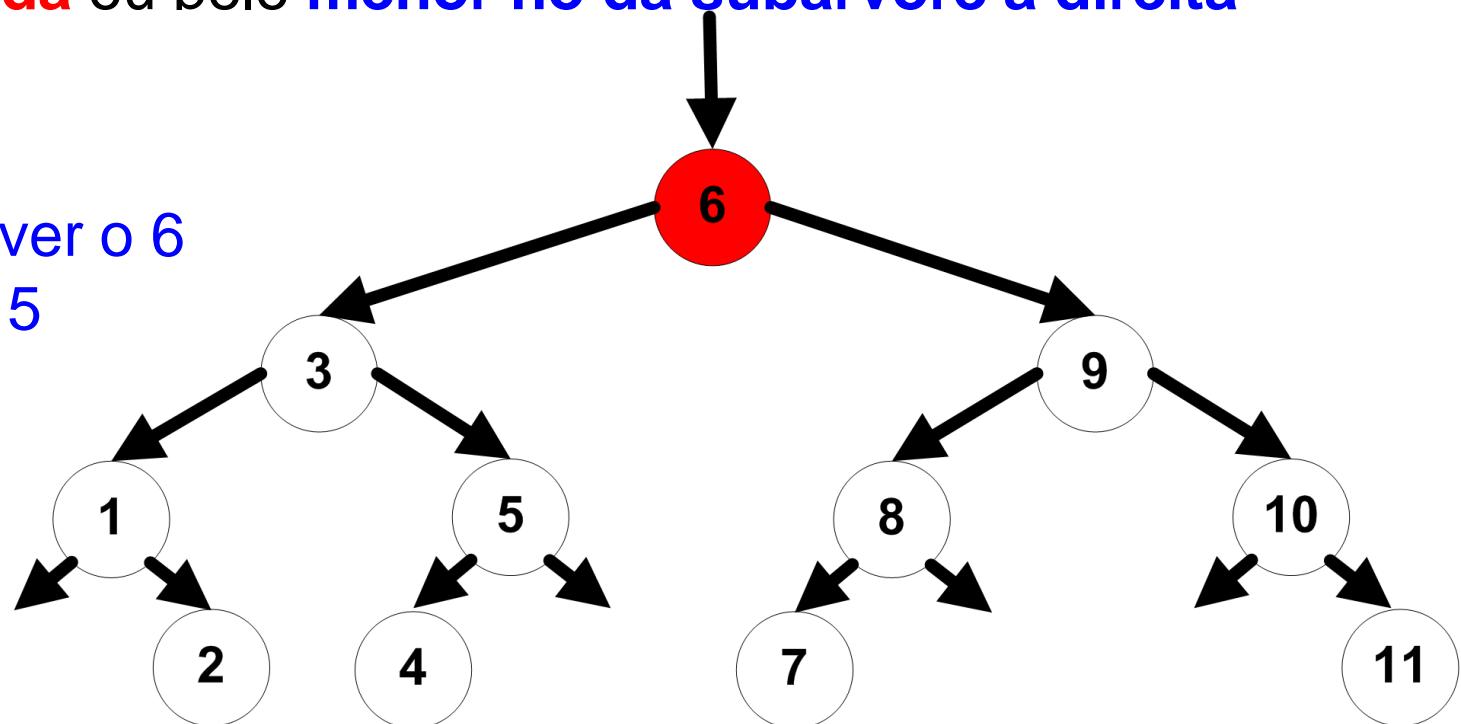


# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

Exemplo: Remover o 6 e substituir pelo 5

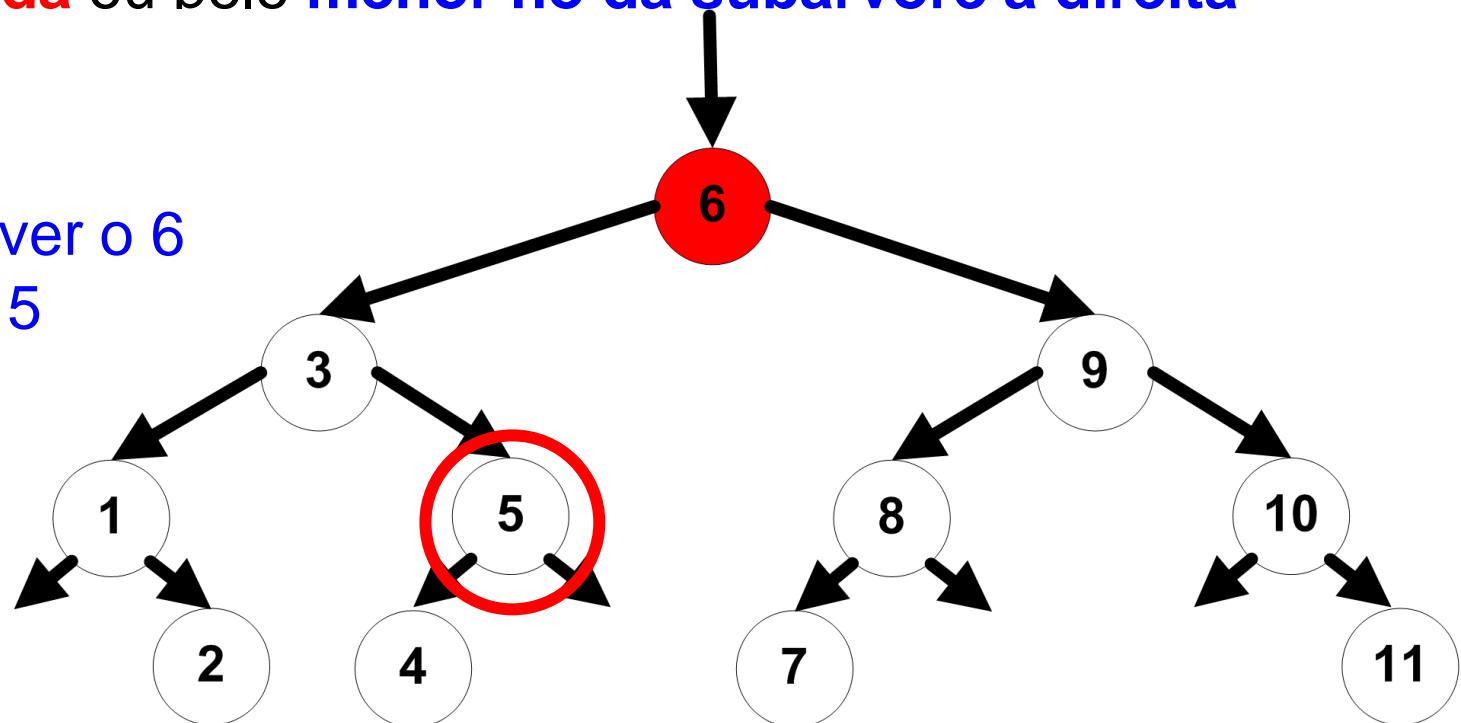


# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

Exemplo: Remover o 6 e substituir pelo 5

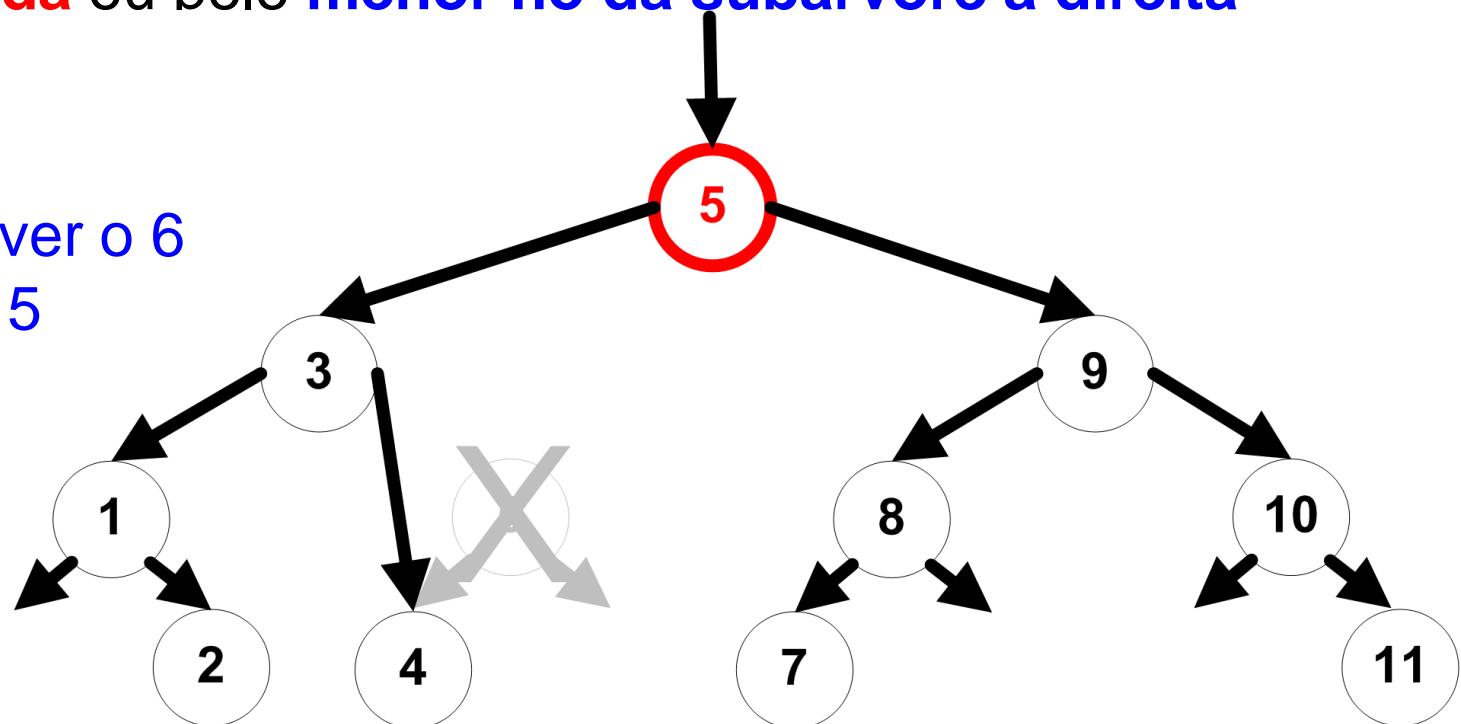


# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

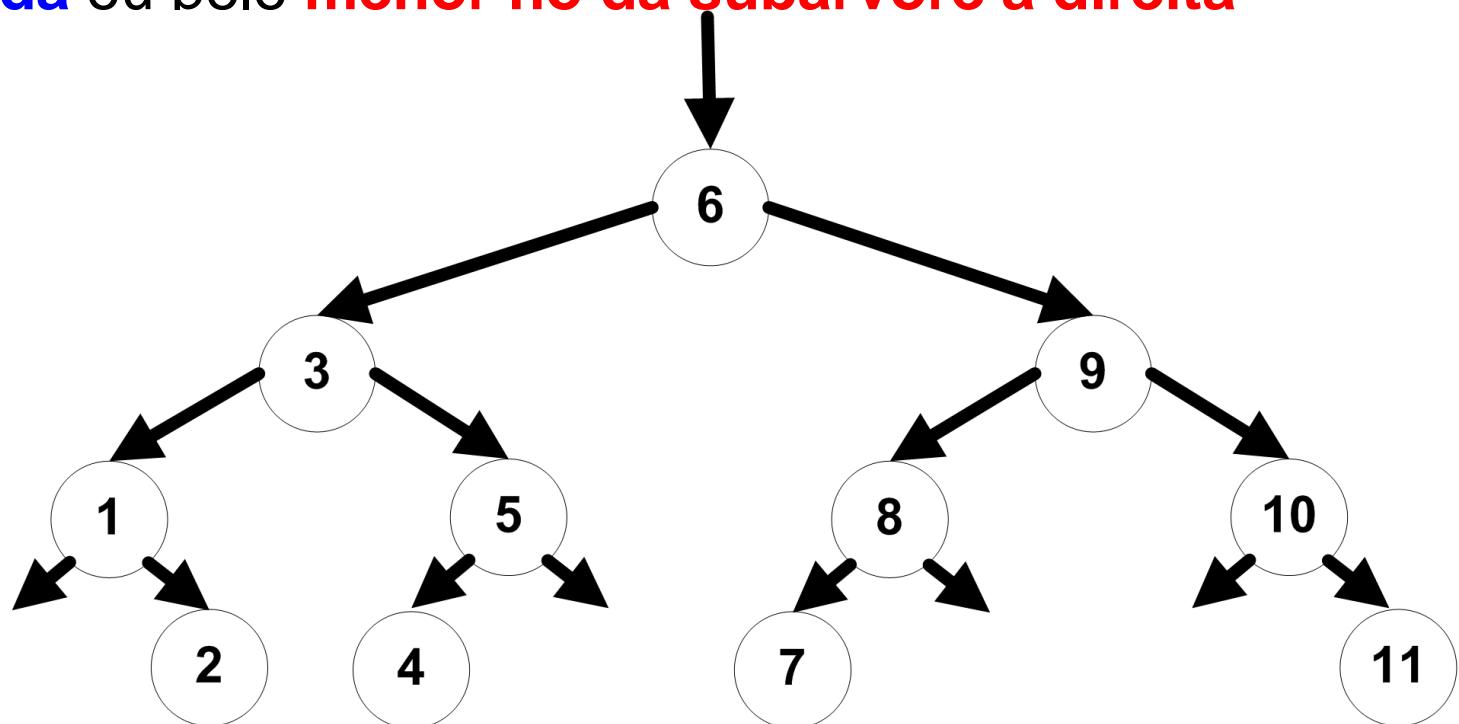
Exemplo: Remover o 6 e substituir pelo 5



# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

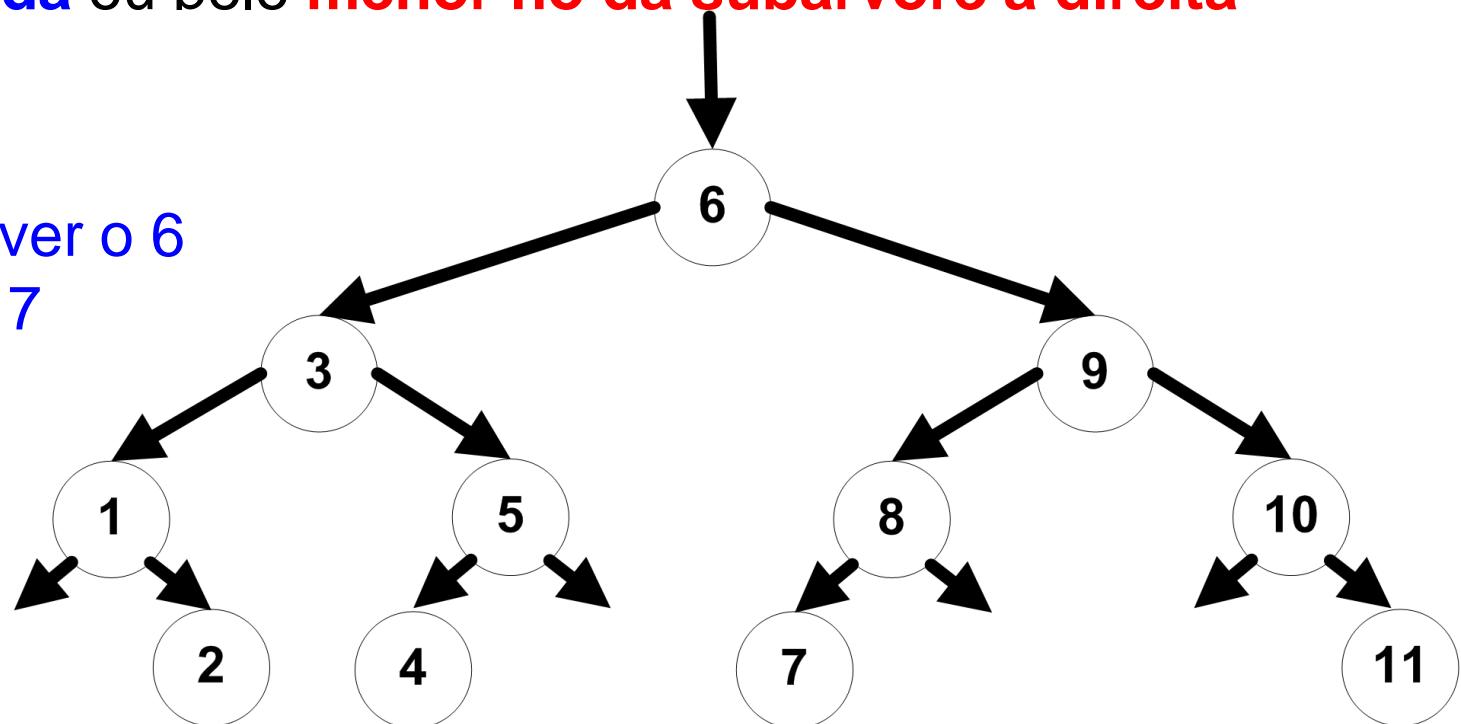


# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

Exemplo: Remover o 6 e substituir pelo 7

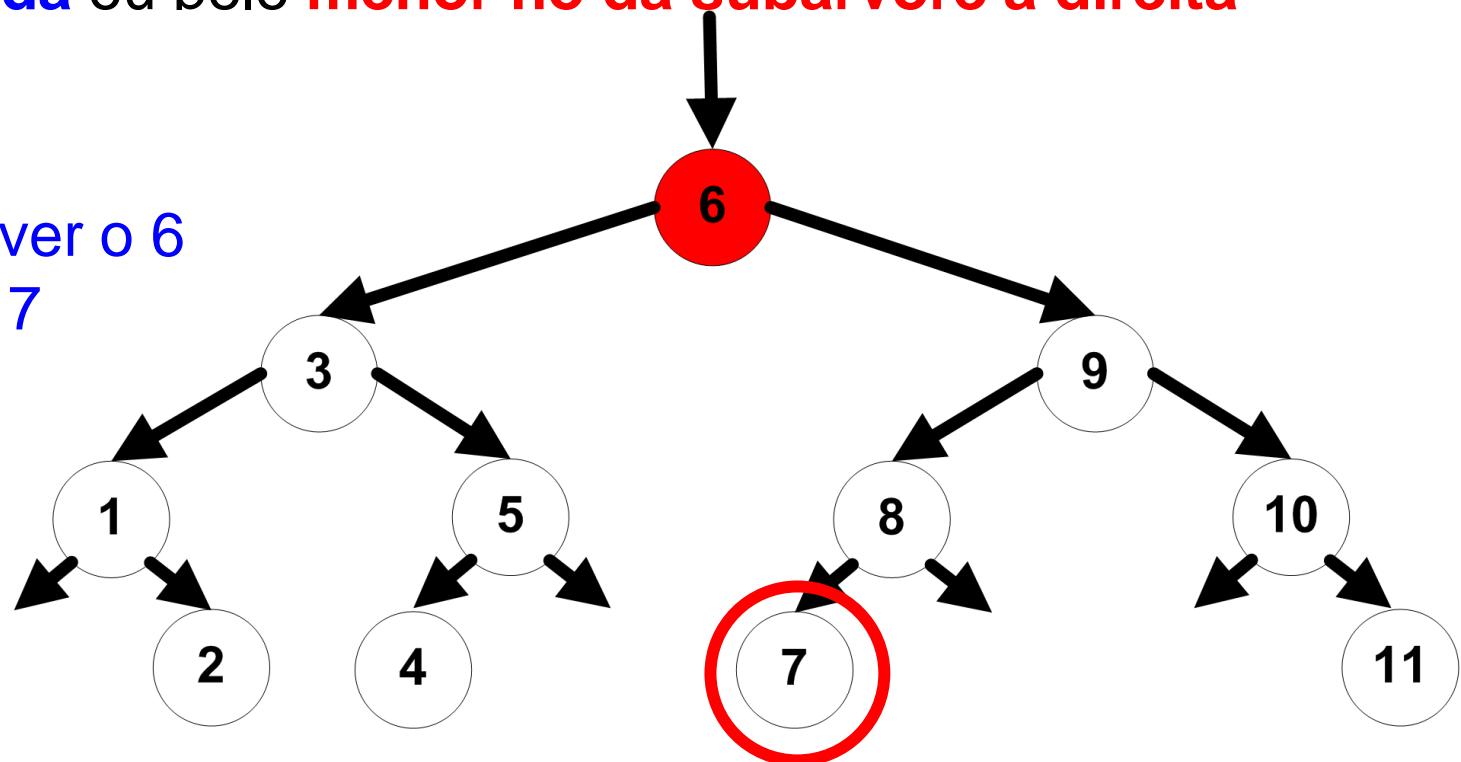


# Remoção em Árvore Binária

- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

Exemplo: Remover o 6 e substituir pelo 7

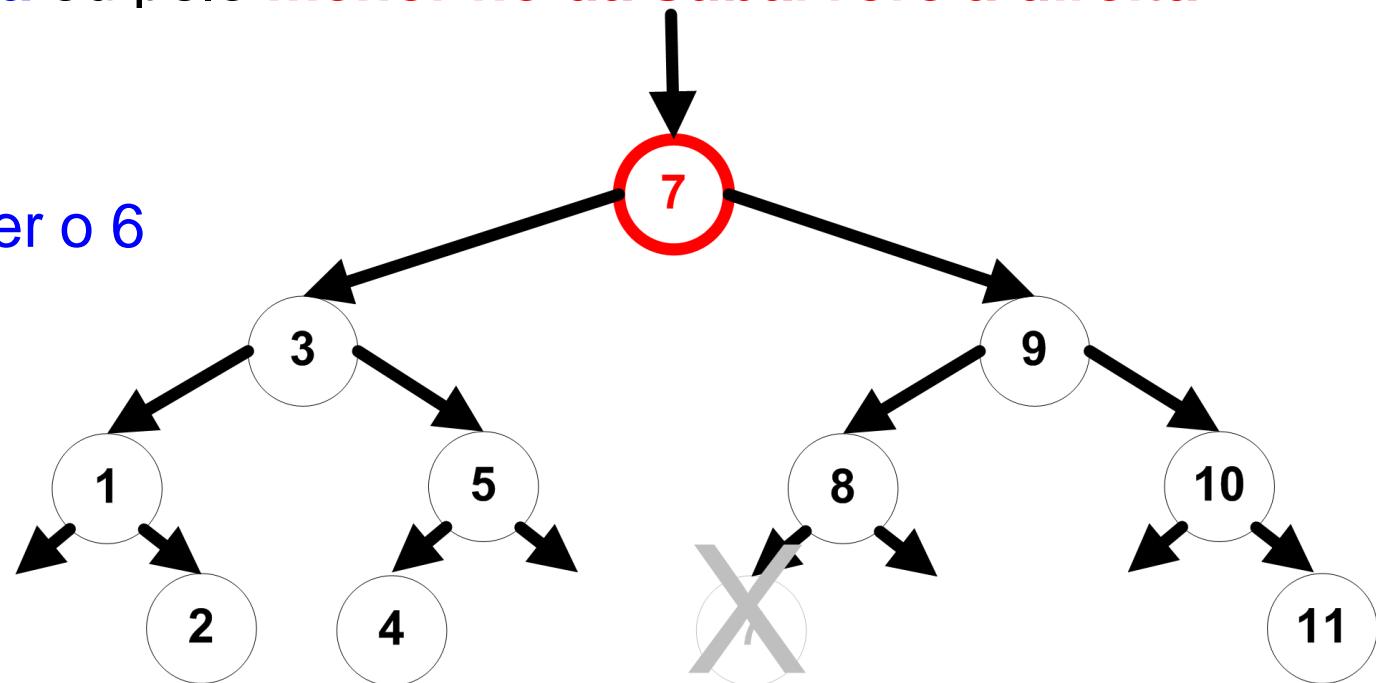


# Remoção em Árvore Binária

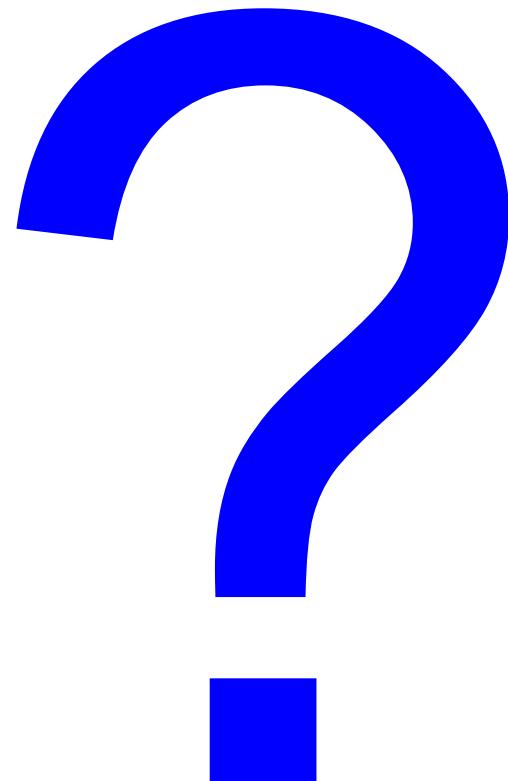
- Funcionamento básico:

(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou pelo **menor nó da subárvore à direita**

Exemplo: Remover o 6  
e substituir pelo 7



# Análise de Complexidade da Remoção

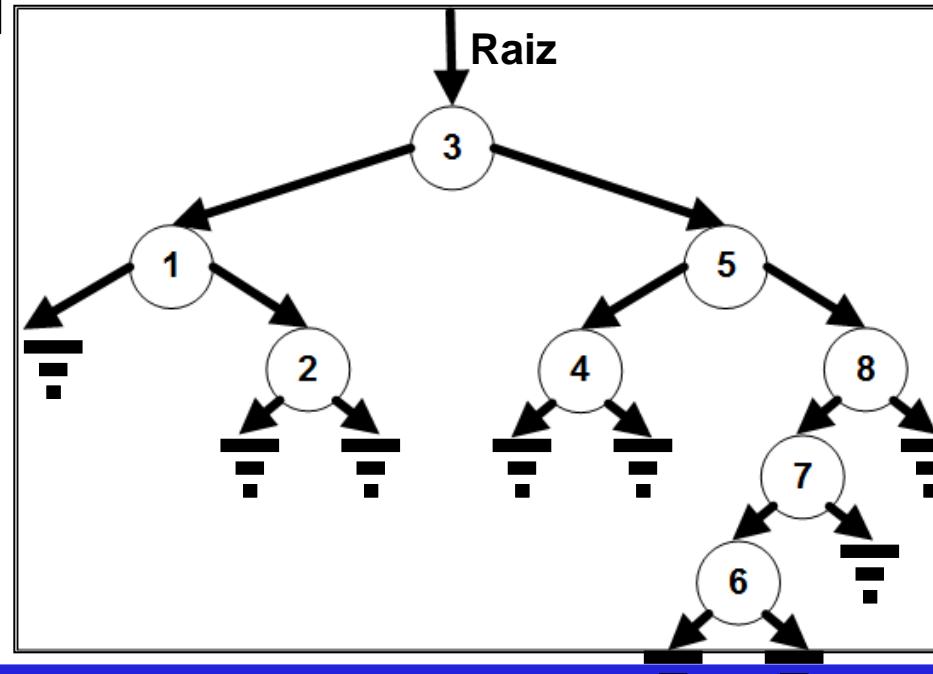


# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)

Vamos remover o 2 (uma folha) de nossa árvore



# Algoritmo em Java - Classe Árvore Binária

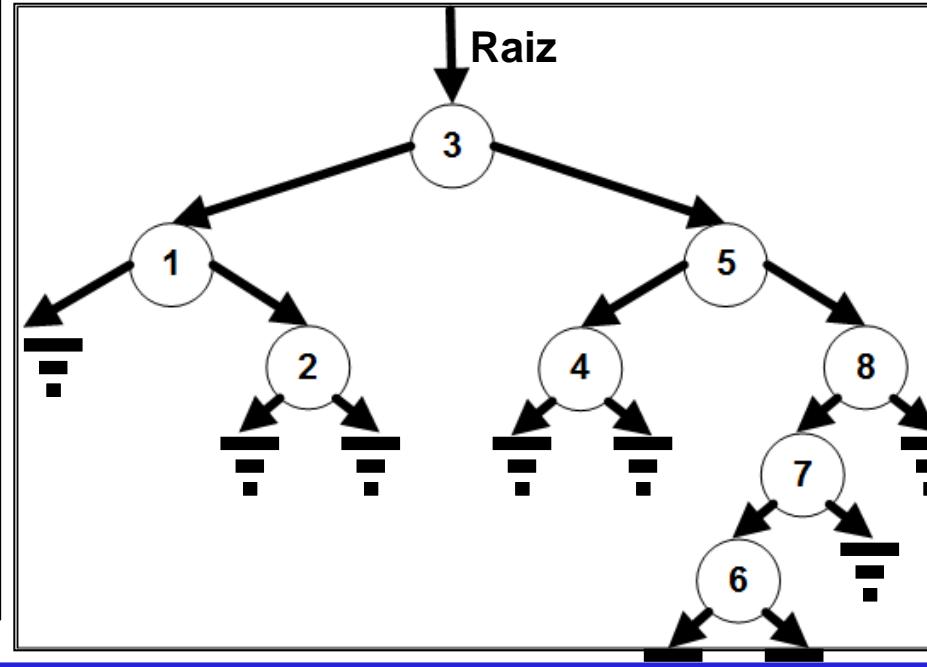
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}
```

raiz n(3) x 2



# Algoritmo em Java - Classe Árvore Binária

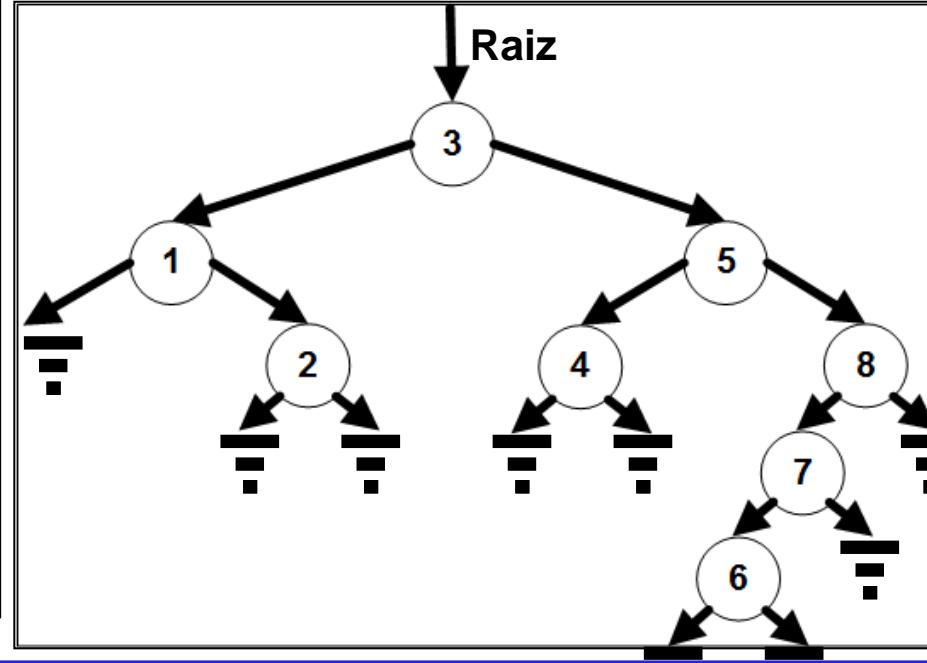
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2



# Algoritmo em Java - Classe Árvore Binária

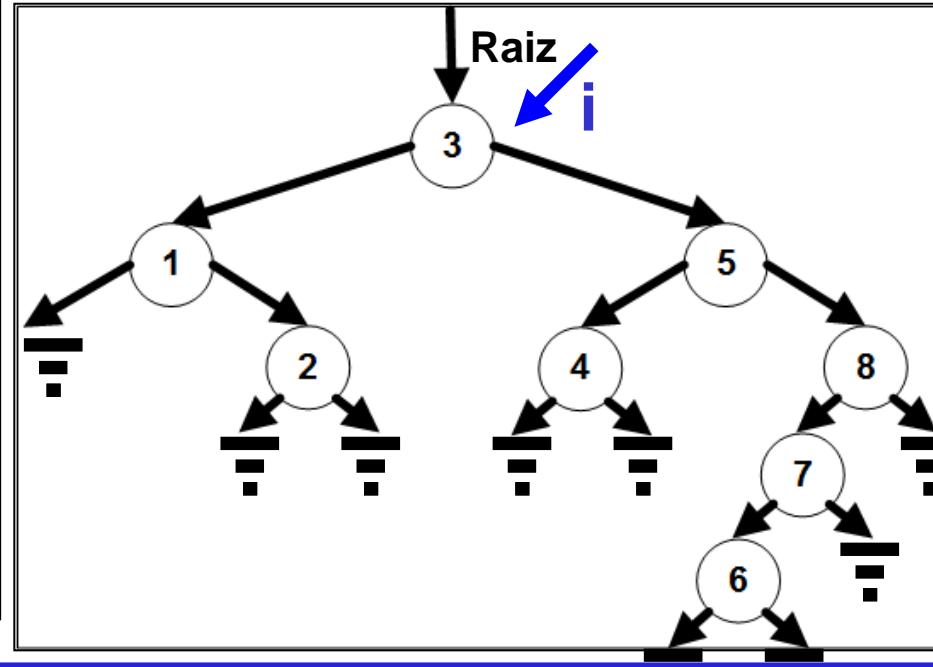
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)



# Algoritmo em Java - Classe Árvore Binária

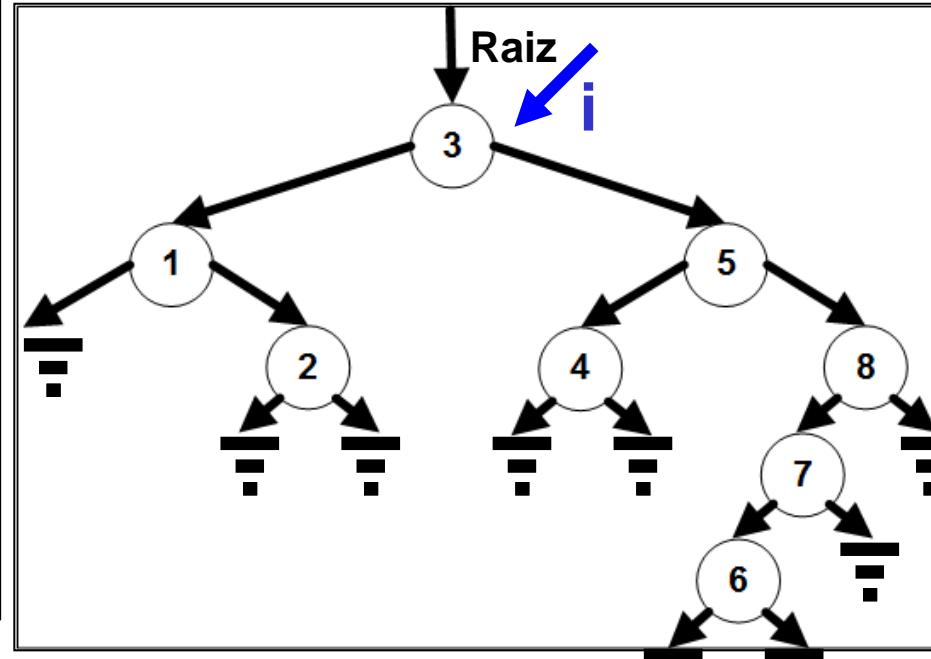
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
false: n(3) == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)



# Algoritmo em Java - Classe Árvore Binária

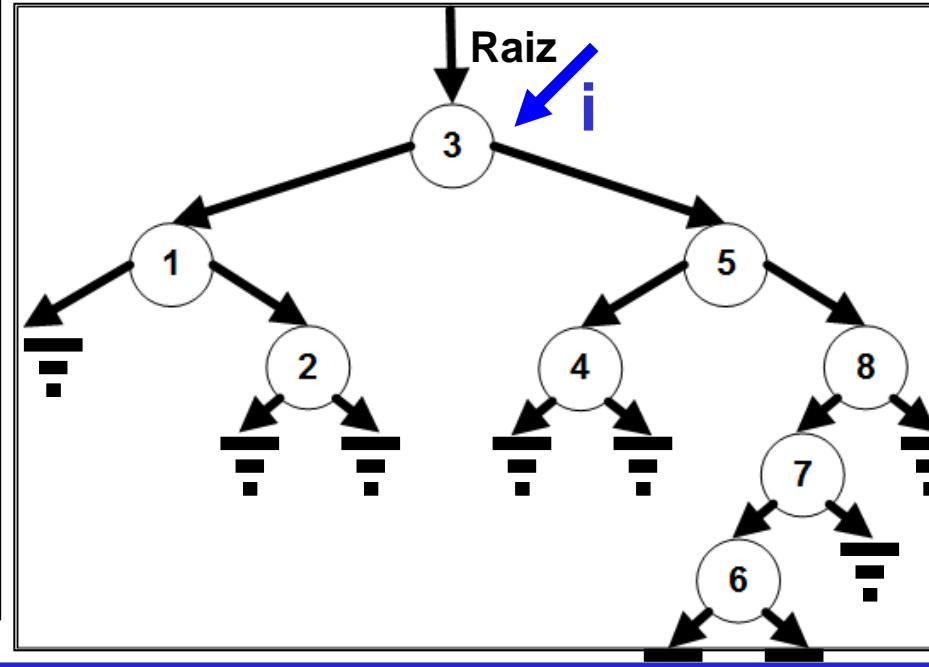
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento){ i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}
true: 2 < 3

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)



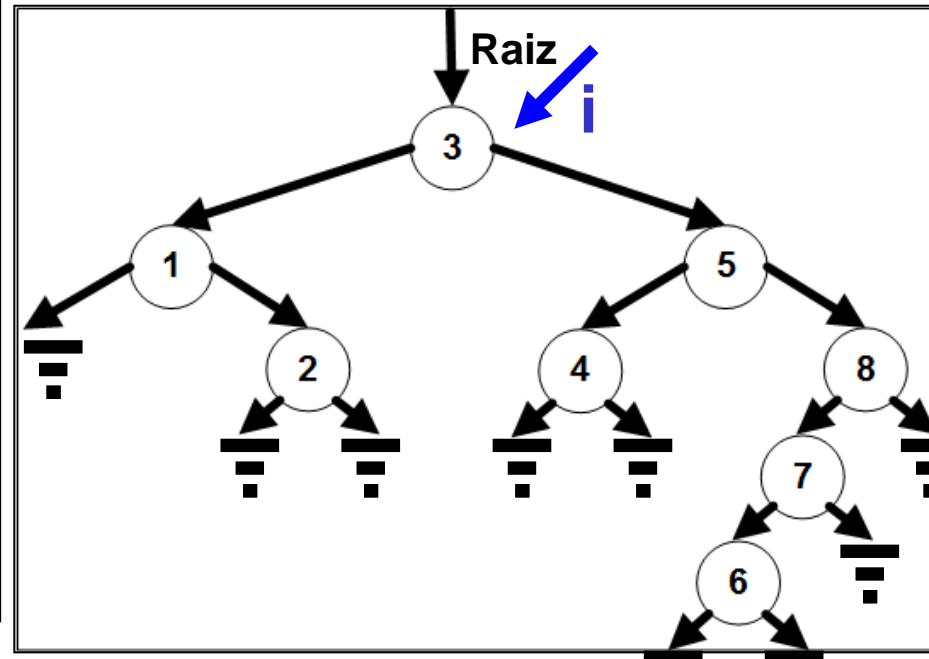
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) {      throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq = remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) {   i = i.esq;
    } else if(i.esq == null) {   i = i.dir;
    } else {                  i.esq = anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {                  i.elemento = j.elemento; j = j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

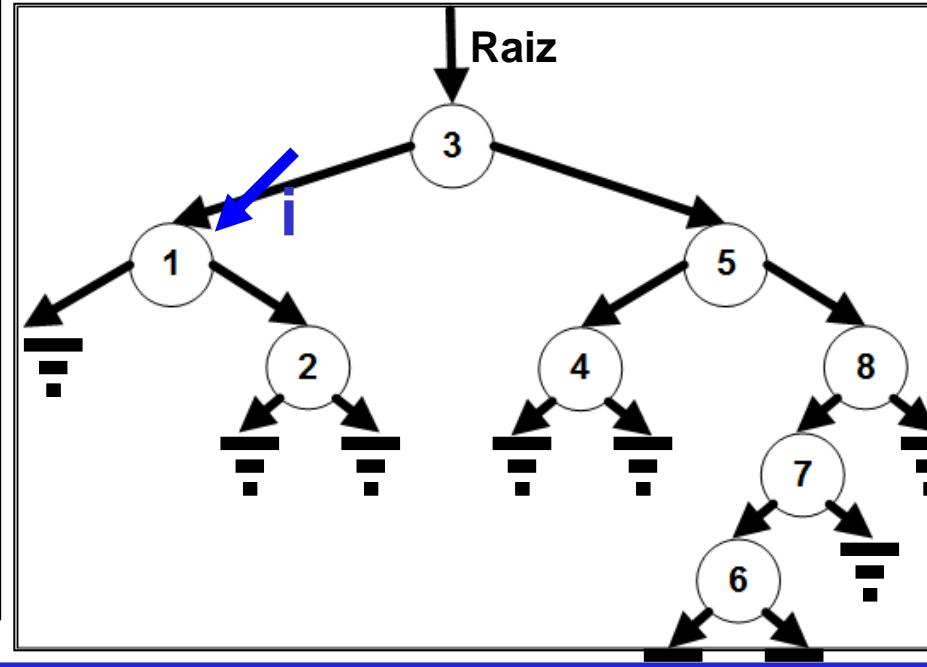
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)

x 2 i n(1)



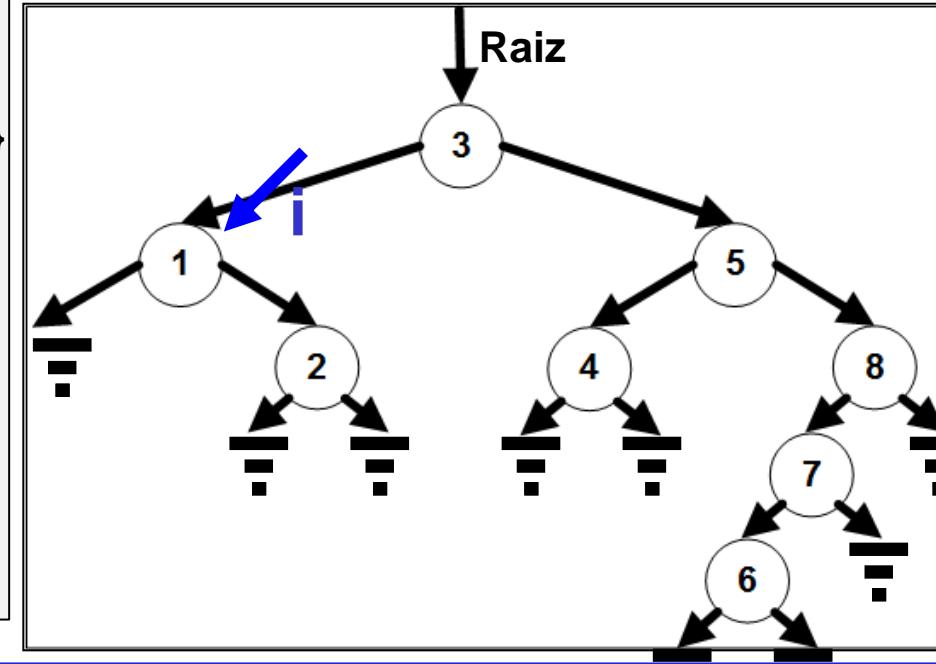
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}
false: n(1) == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

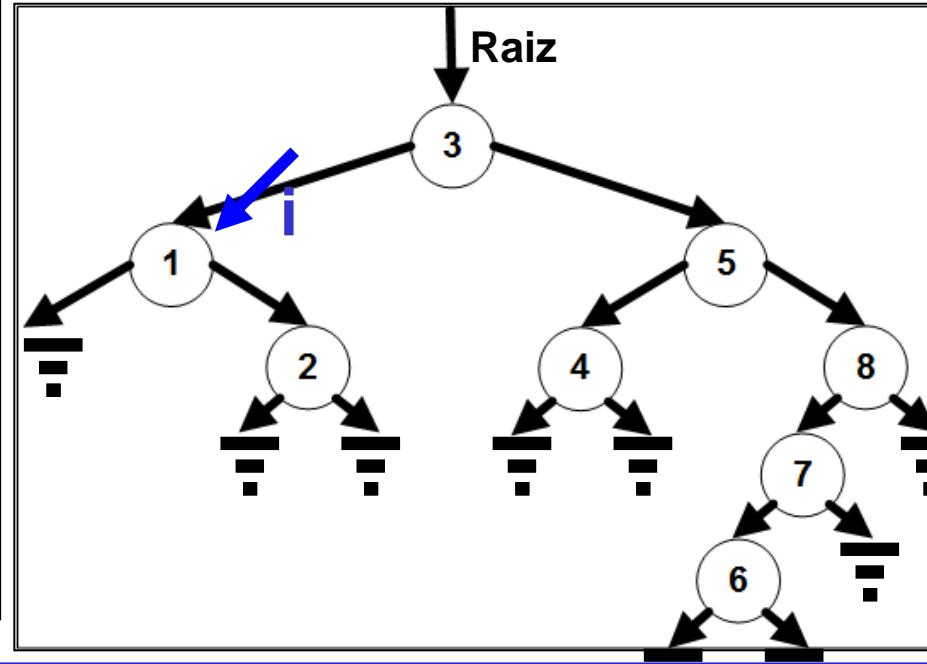
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento){ i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}
false: 2 < 1

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)

x 2 i n(1)



# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

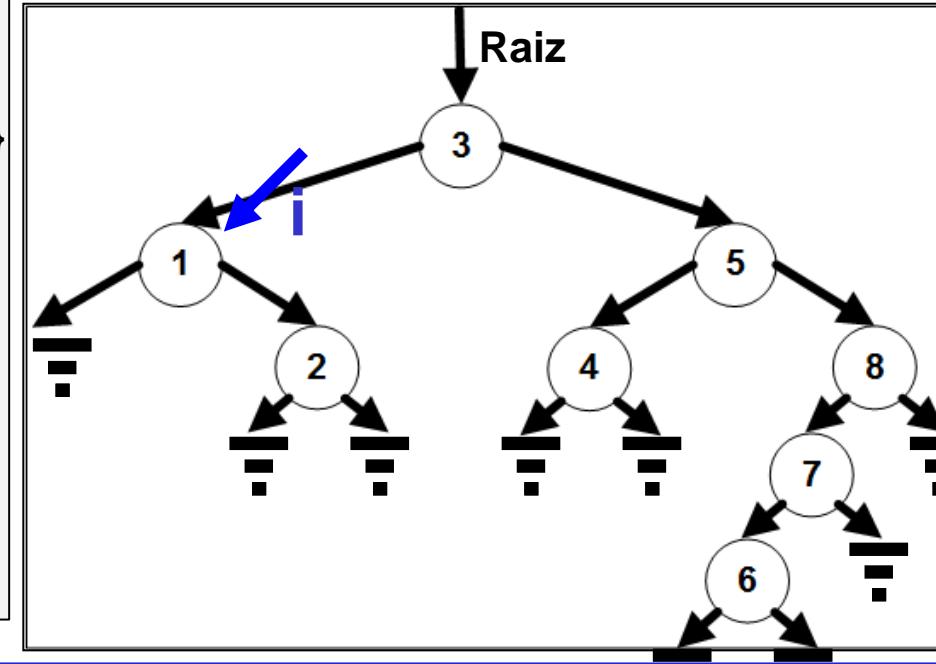
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
true: 2 > 1

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)

x 2 i n(1)



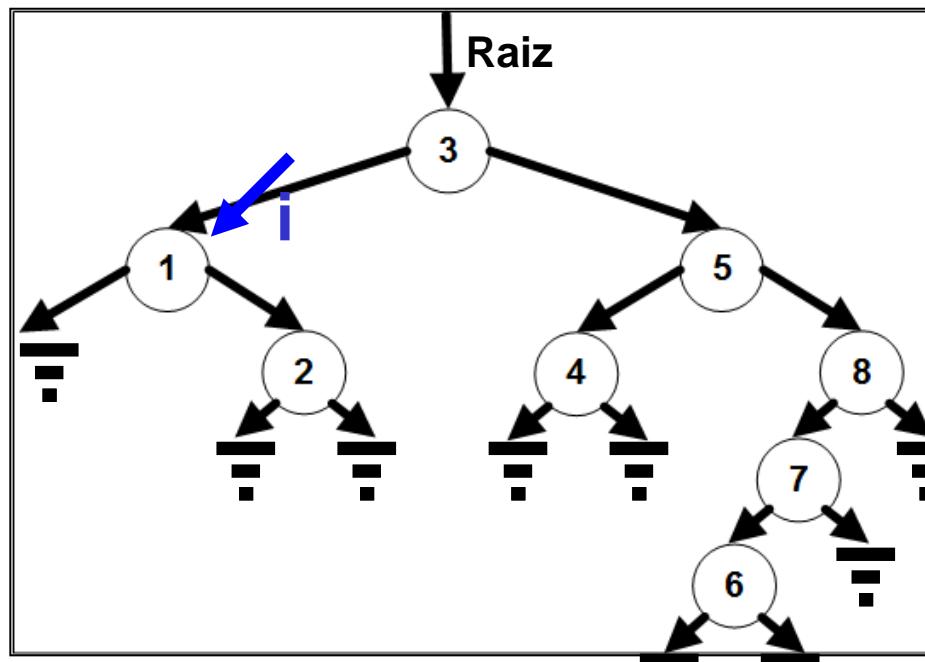
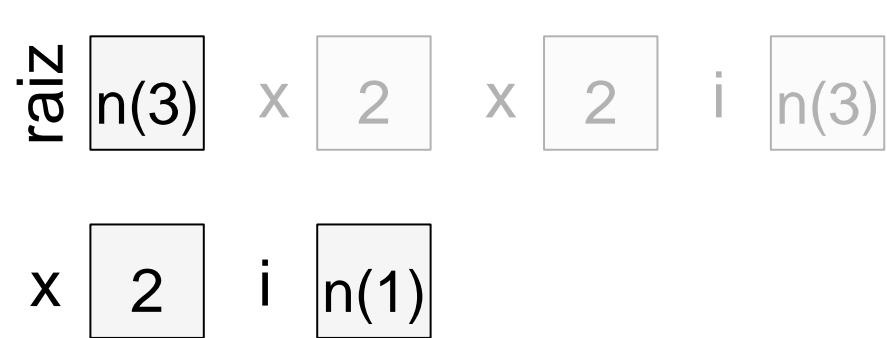
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) {      throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq = remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) {   i = i.esq;
    } else if(i.esq == null) {   i = i.dir;
    } else {                  i.esq = anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {                  i.elemento = j.elemento; j = j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

```

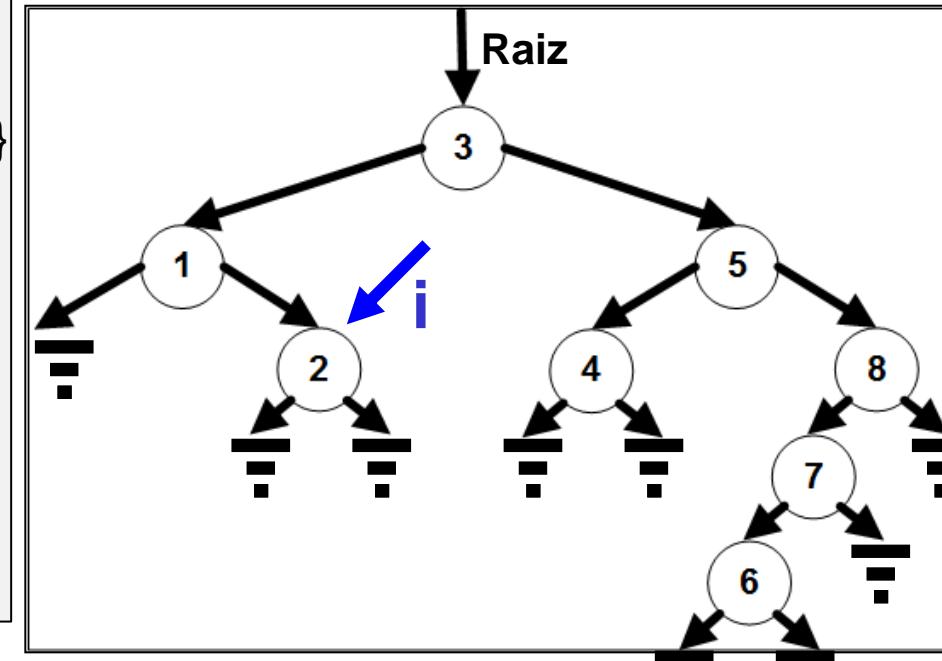
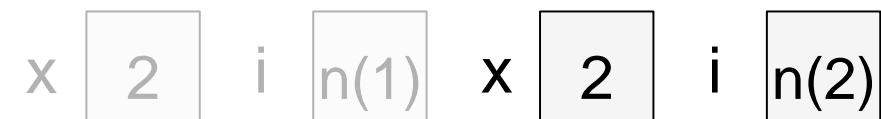
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}

```



# Algoritmo em Java - Classe Árvore Binária

```

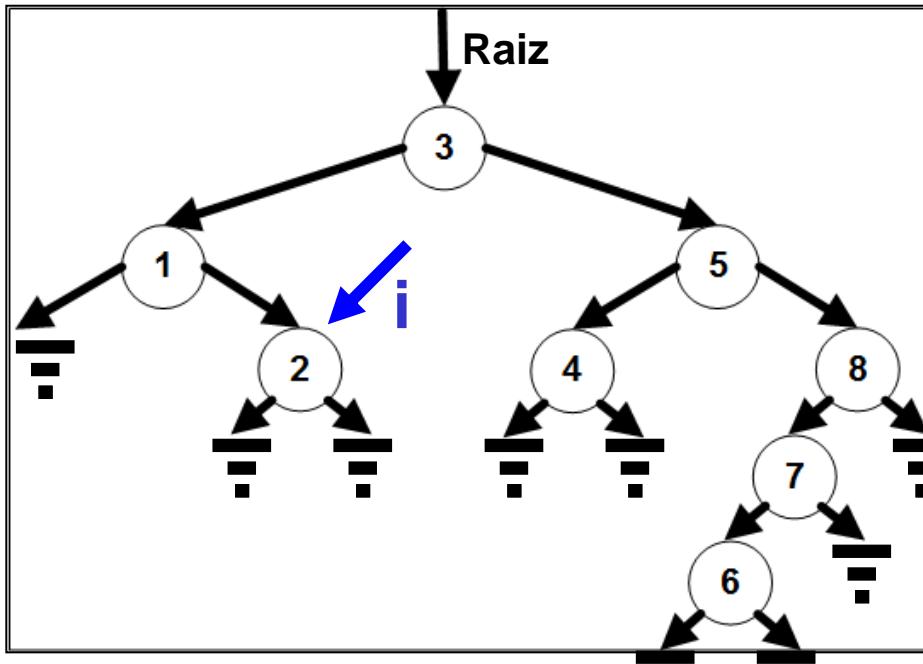
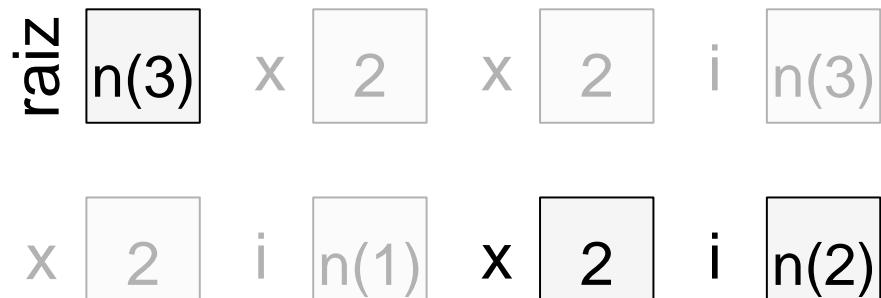
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}                                false: n(2) == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}

```



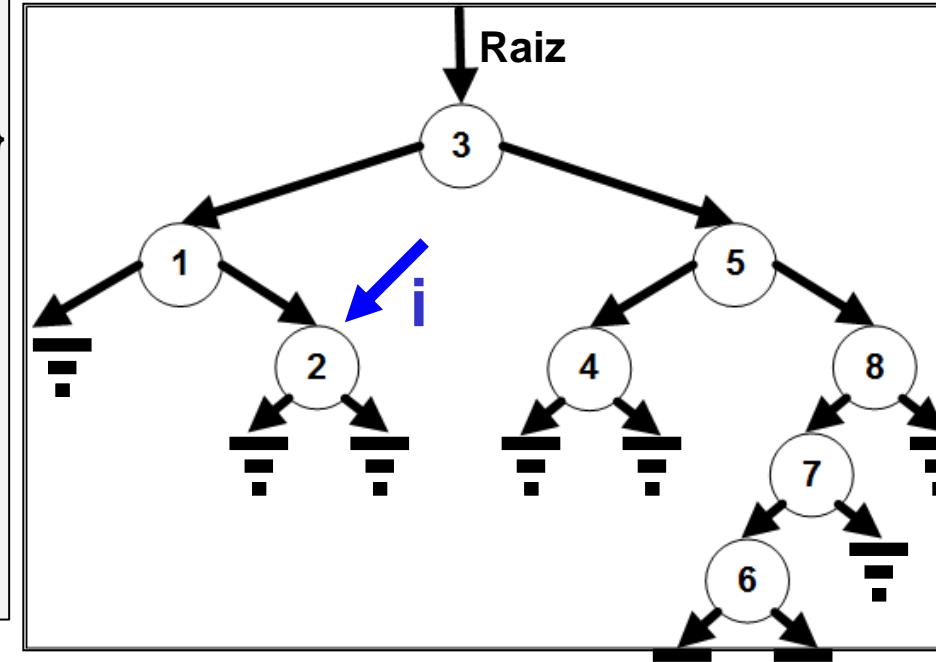
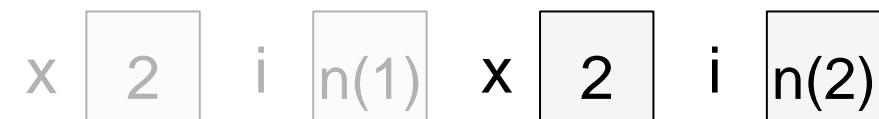
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento){ i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}
false: 2 < 2

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



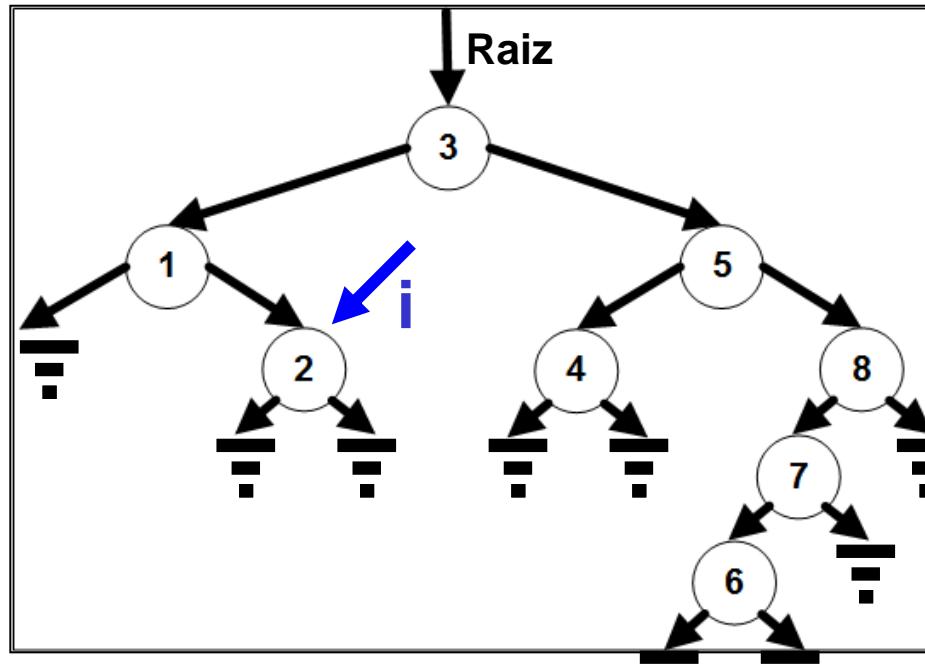
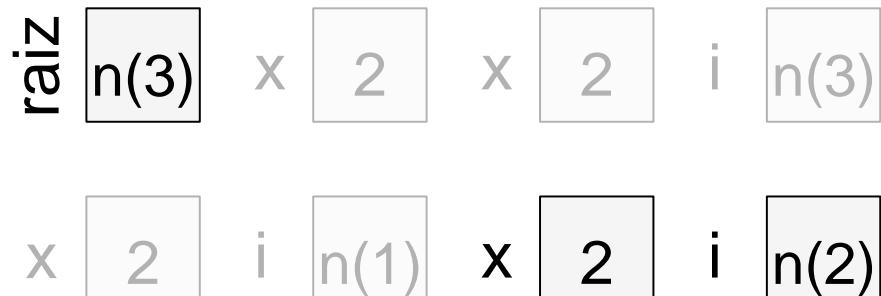
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
    false: 2 > 2

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

```

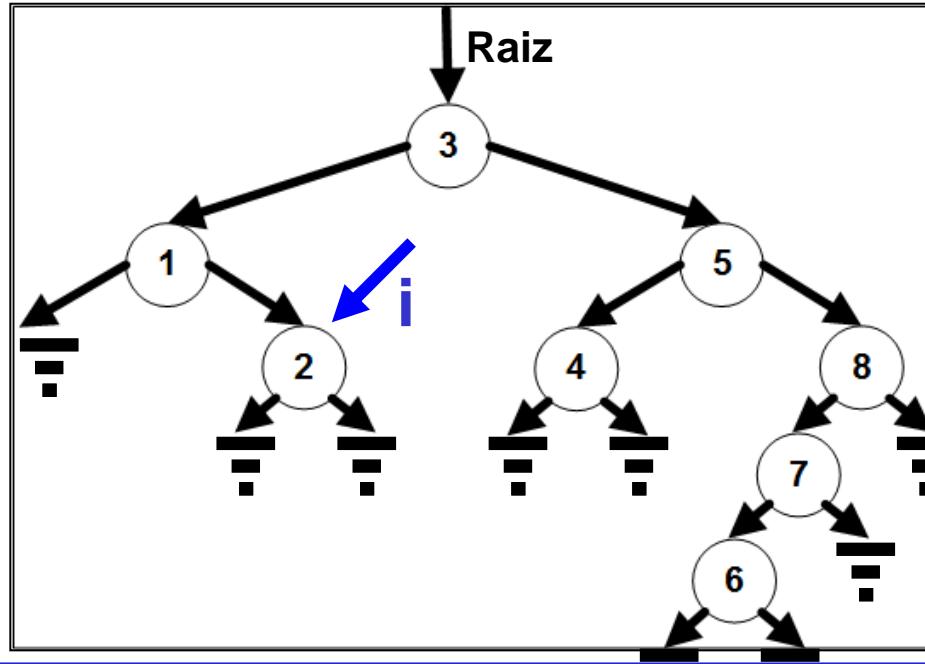
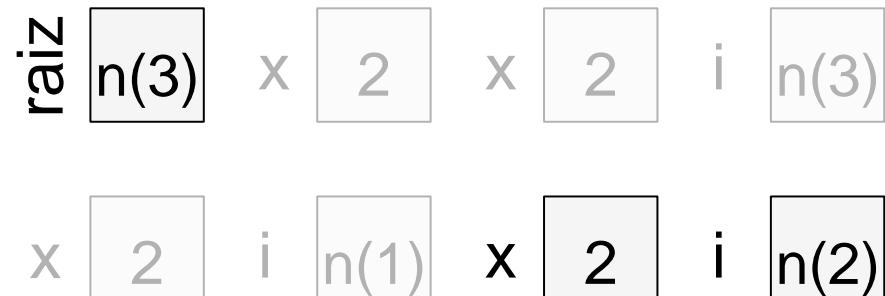
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
true: null == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}

```



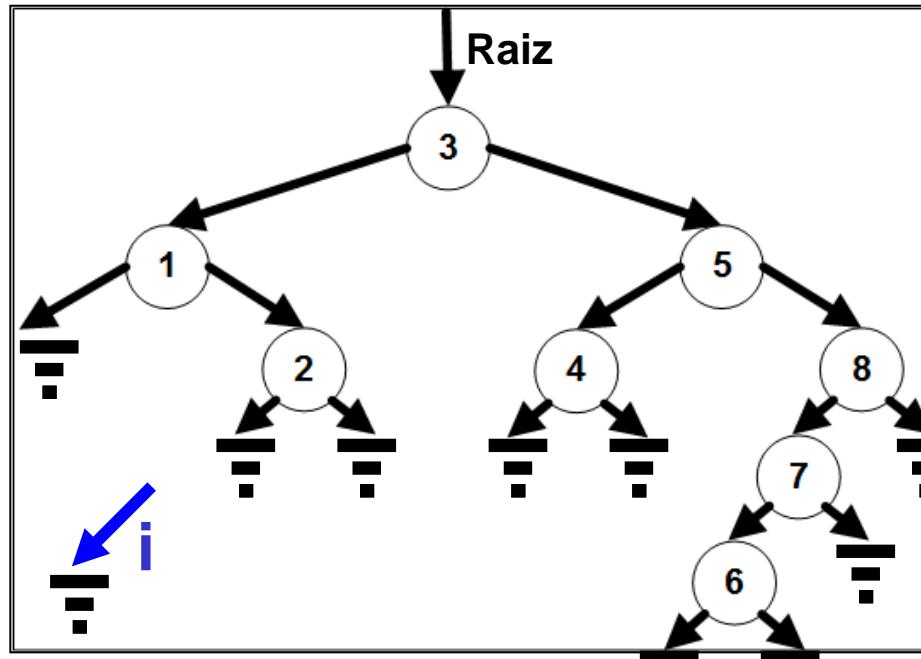
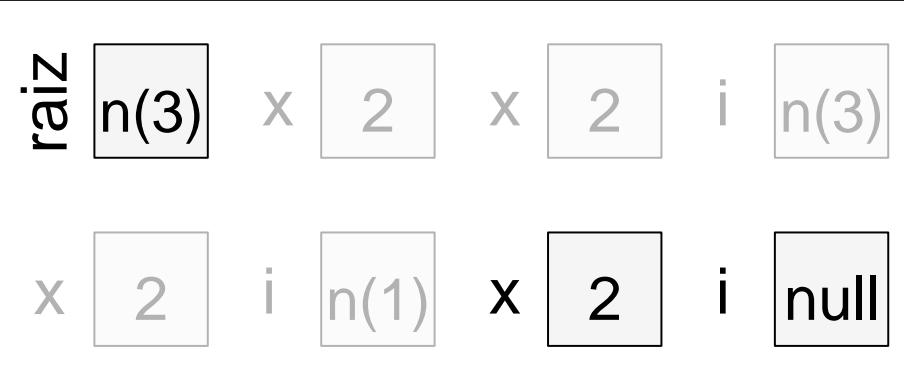
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq = remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) { i = i.esq;
    } else if(i.esq == null) { i = i.dir;
    } else { i.esq = anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento = j.elemento; j = j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

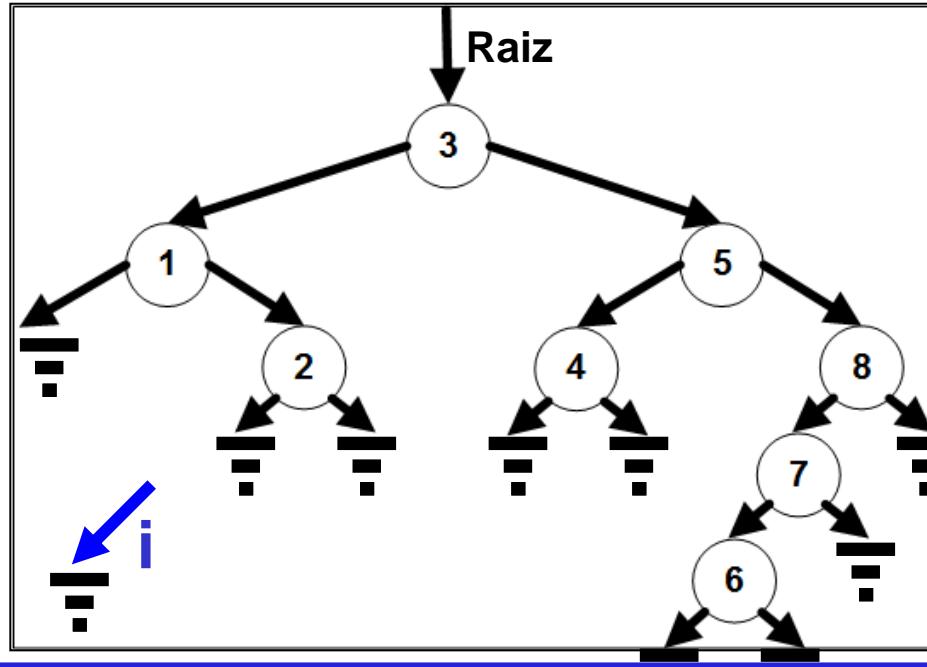
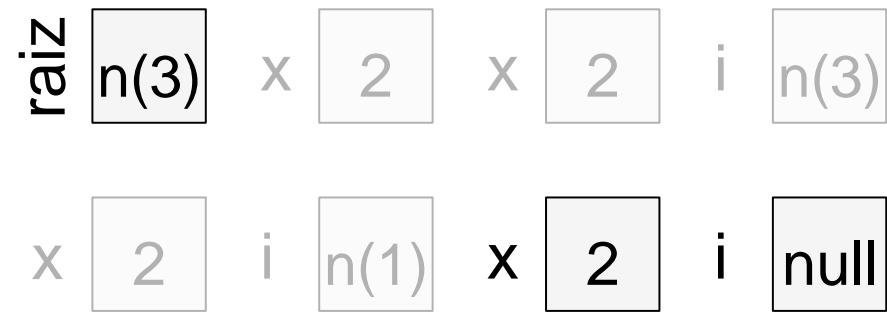
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

Retorna null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



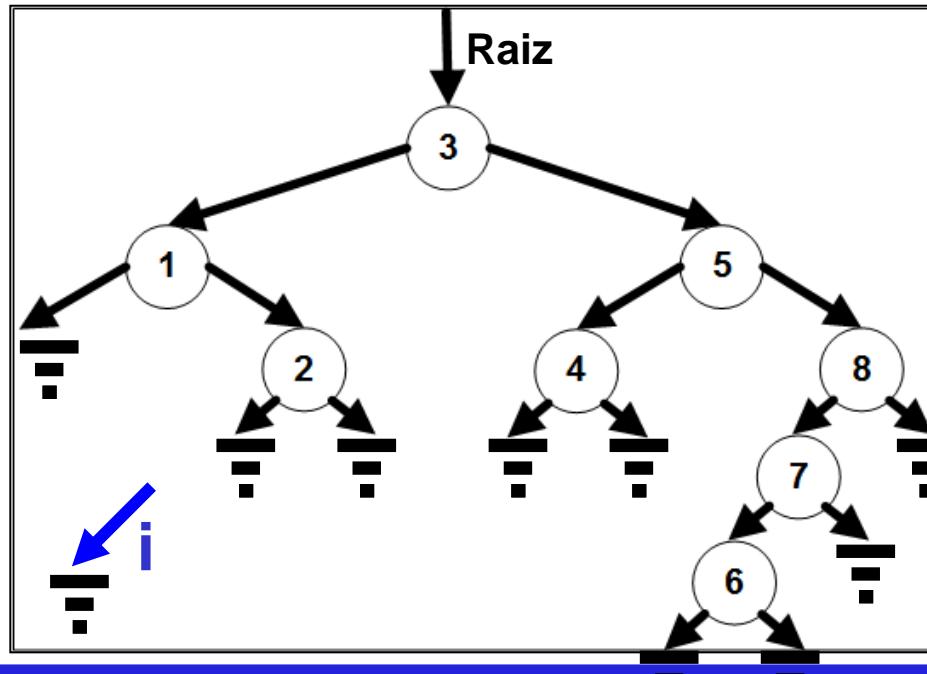
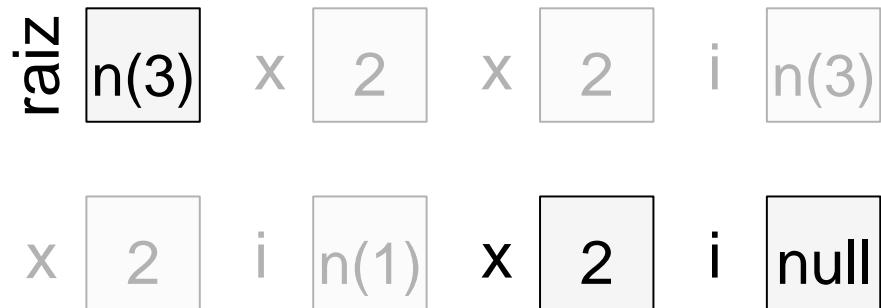
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



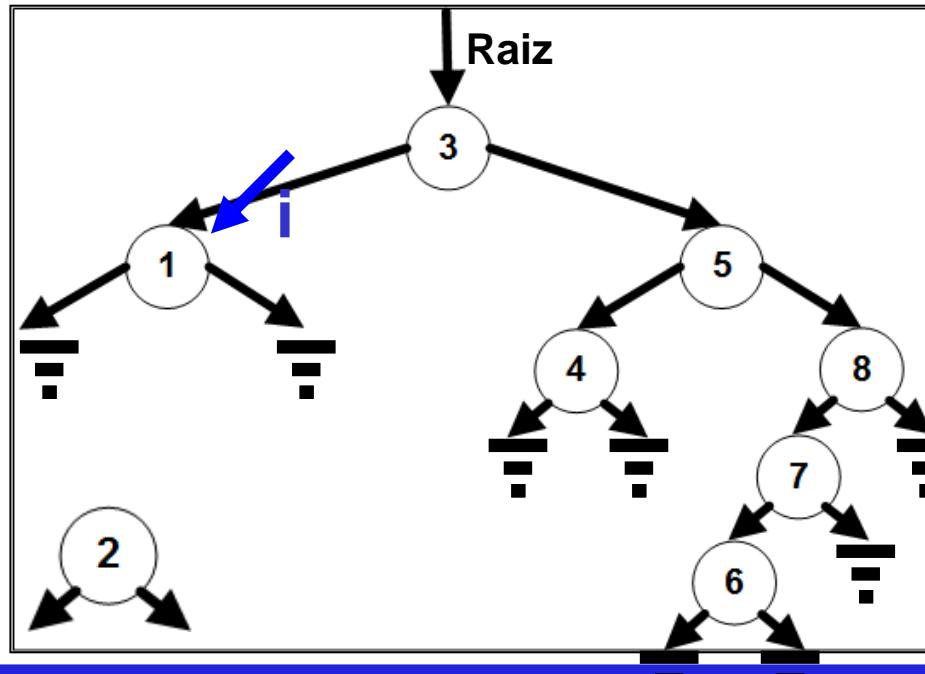
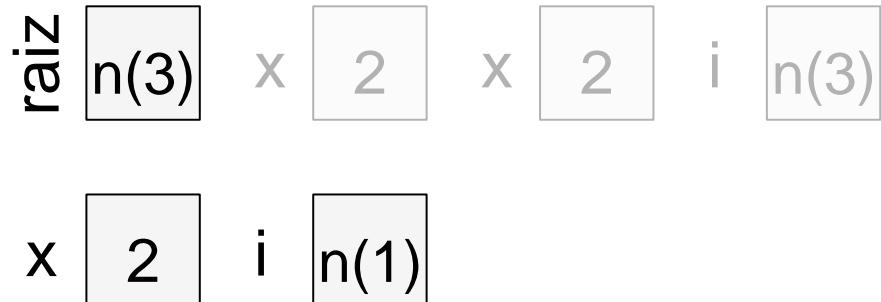
# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

```

//remover(2), folha

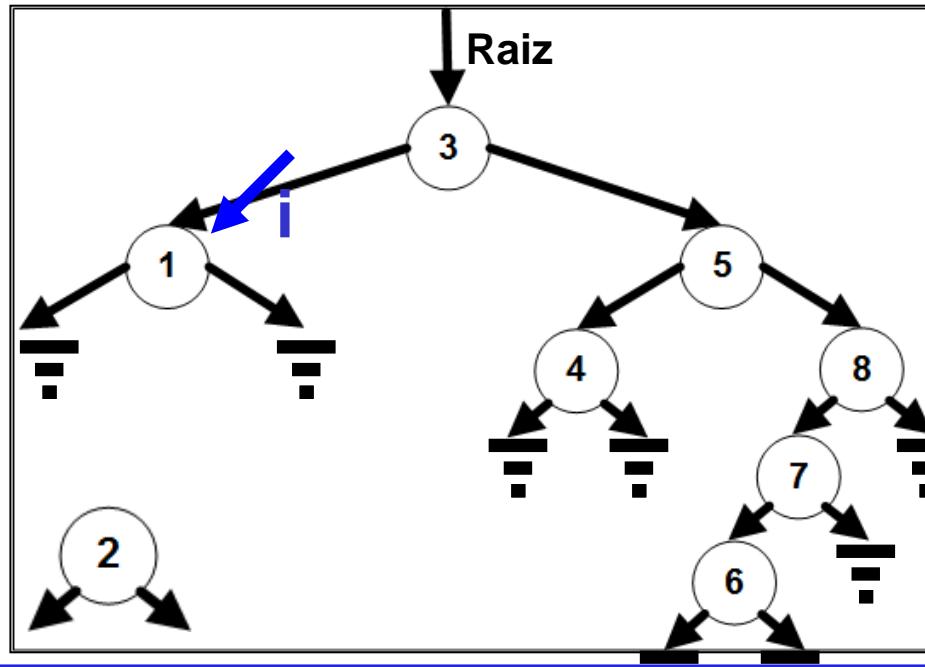
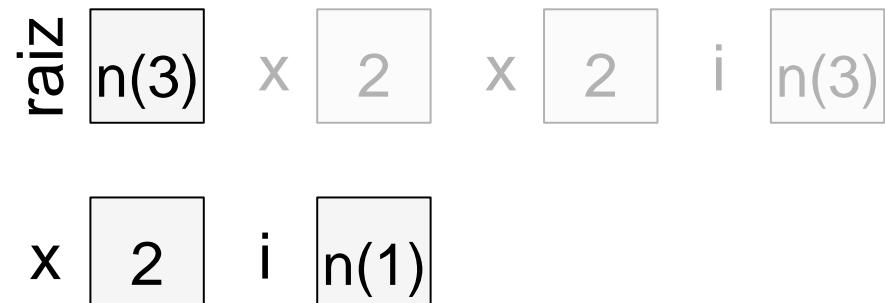
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}

```

**Retorna n(1)**



# Algoritmo em Java - Classe Árvore Binária

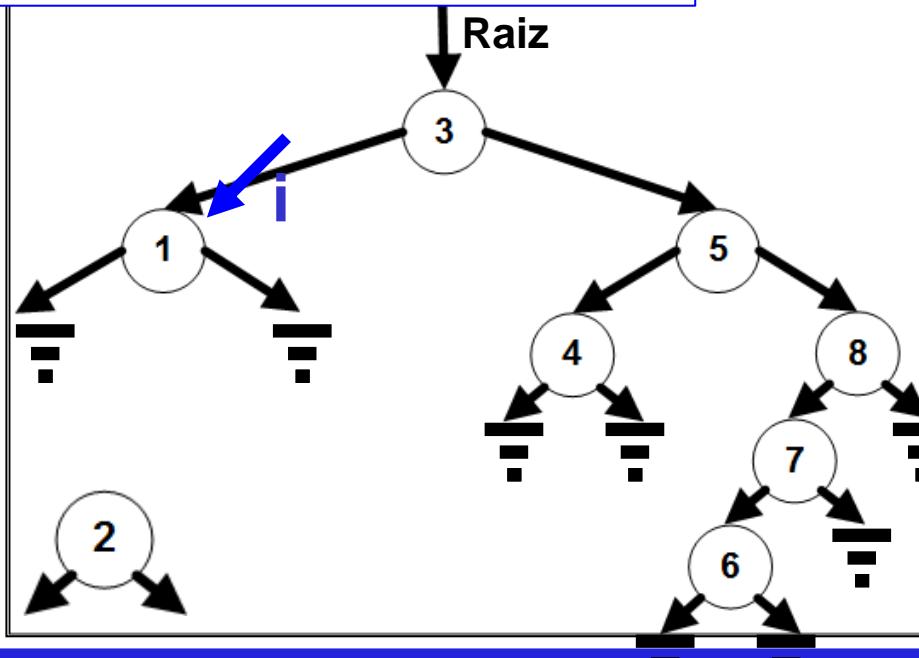
```
//remover(2), folha
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

```
private
if (i ==
} else
} else
} else if(i.esq == null) { i = i.esq;
} else if(i.esq == null) { i = i.dir;
} else {
    i.esq=anterior(i, i.esq);
return i;
}
```

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq;
    }
}
```

raiz n(3) x 2 x 2 i n(3)

Após a coleta de lixo do Java  
(que não controlamos quando ela acontece)...



# Algoritmo em Java - Classe Árvore Binária

```
//remover(2), folha

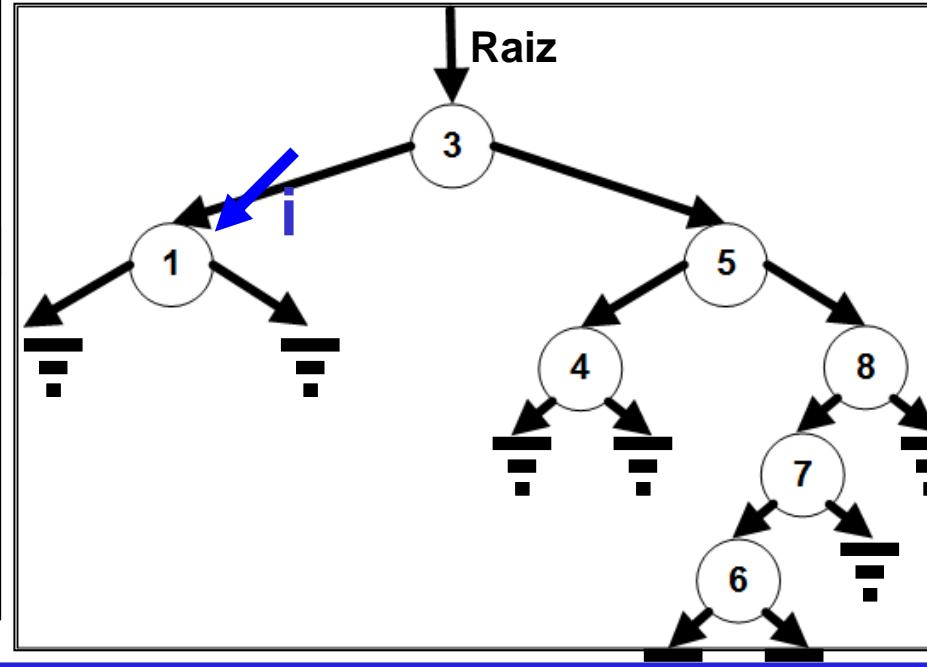
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)

x 2 i n(1)



# Algoritmo em Java - Classe Árvore Binária

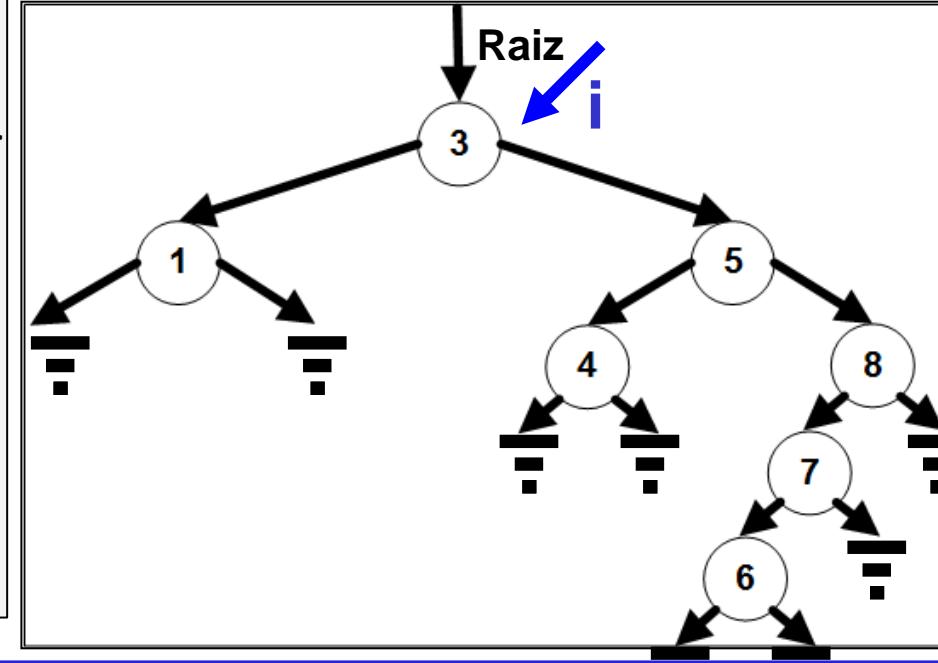
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2 x 2 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```

//remover(2), folha

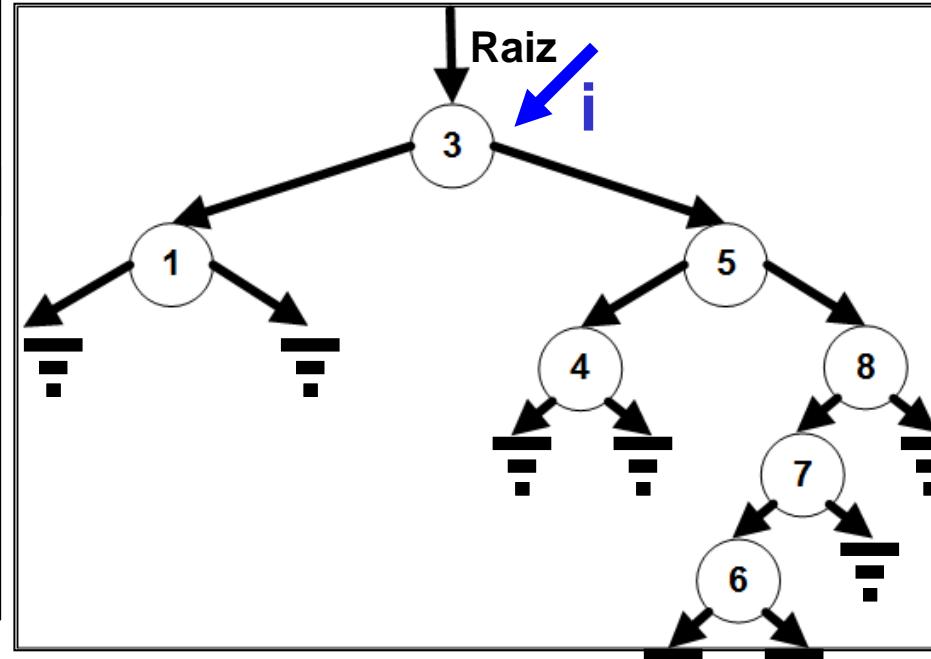
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    else if(i.dir == null) { i = i.esq; }
    else if(i.esq == null) { i = i.dir; }
    else { i.esq=anterior(i, i.esq); }
    return i;
} Retorna n(3)

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}

```

raiz n(3) x 2 x 2 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```

//remover(2), folha

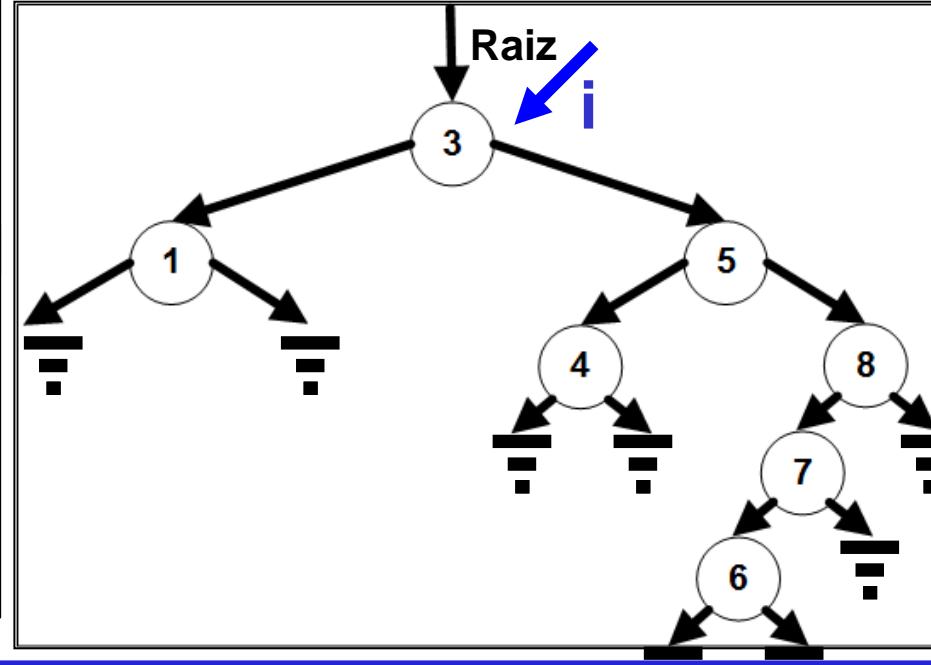
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    else if(i.dir == null) { i = i.esq; }
    else if(i.esq == null) { i = i.dir; }
    else { i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}

```

raiz **n(3)**    x **2**    x **2**    i **n(3)**



# Algoritmo em Java - Classe Árvore Binária

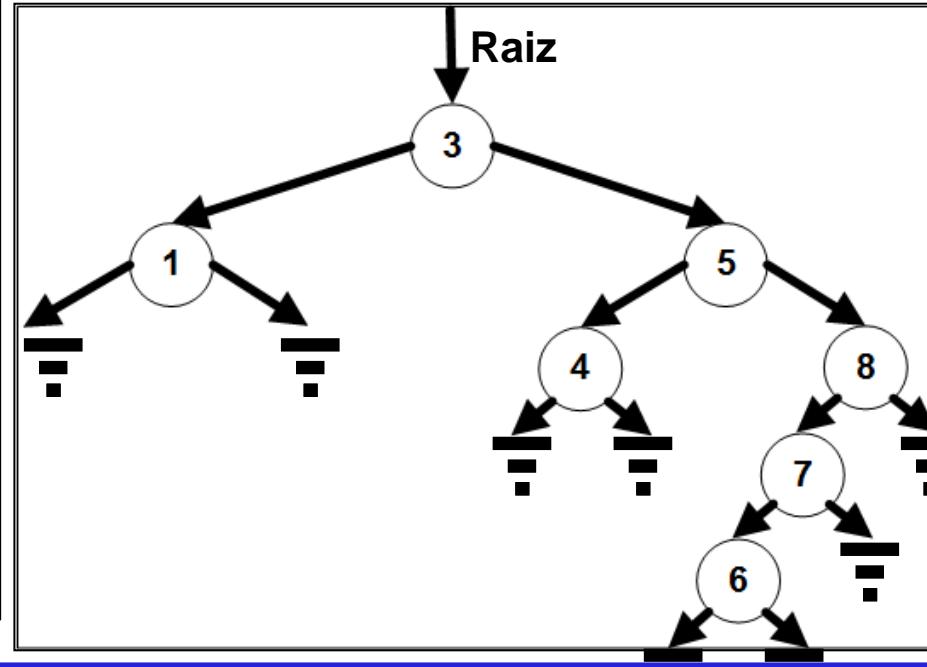
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 2



# Algoritmo em Java - Classe Árvore Binária

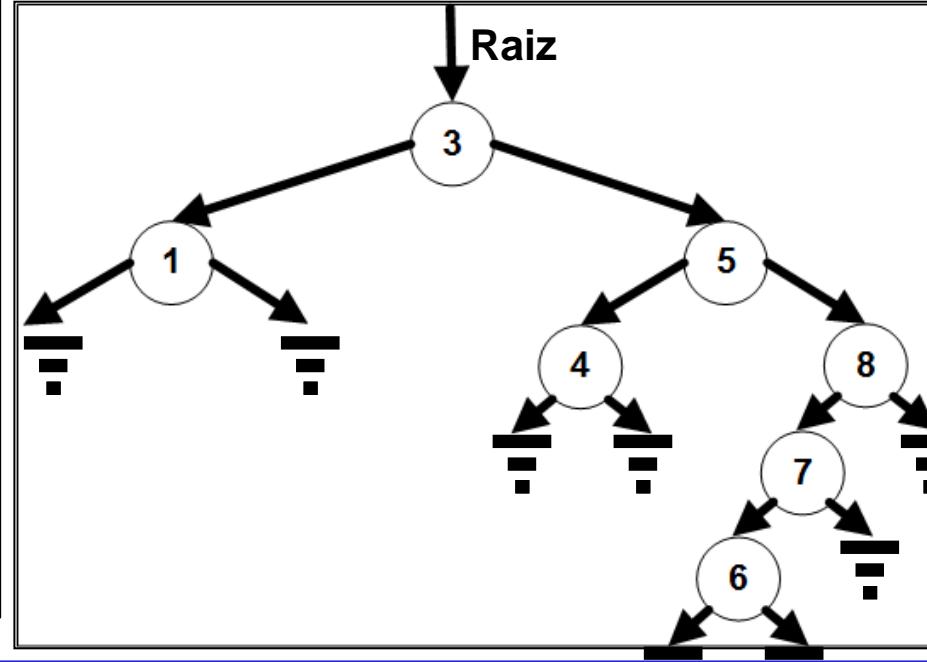
```
//remover(2), folha

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz  
n(3)

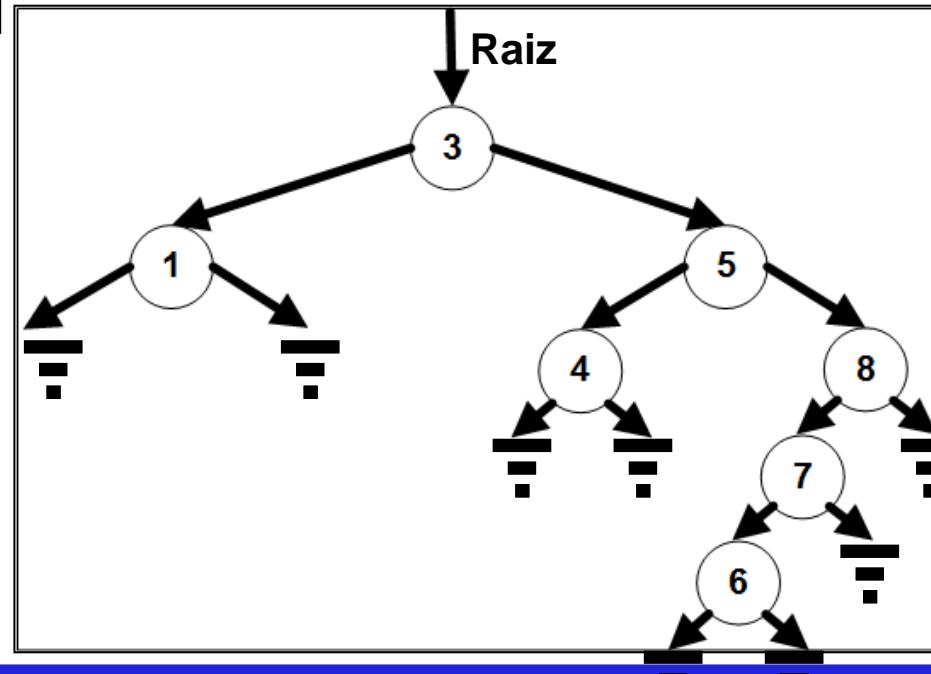


# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)

Voltando com o 2 antes  
de fazer outra remoção

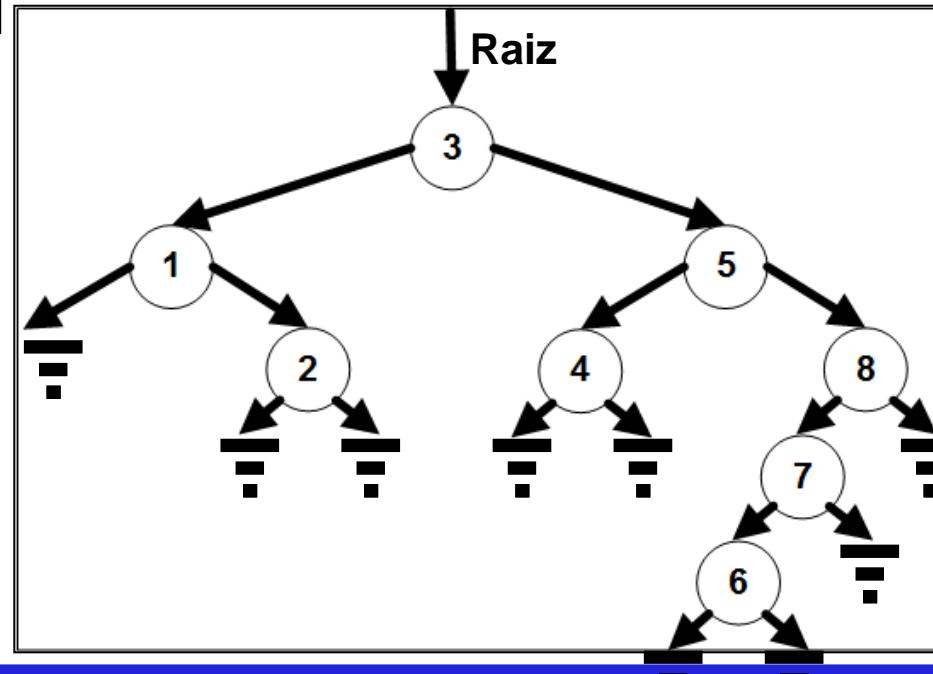


# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)

Vamos remover o 1 (tem um filho) de nossa árvore



# Algoritmo em Java - Classe Árvore Binária

//remover(1), um filho

```

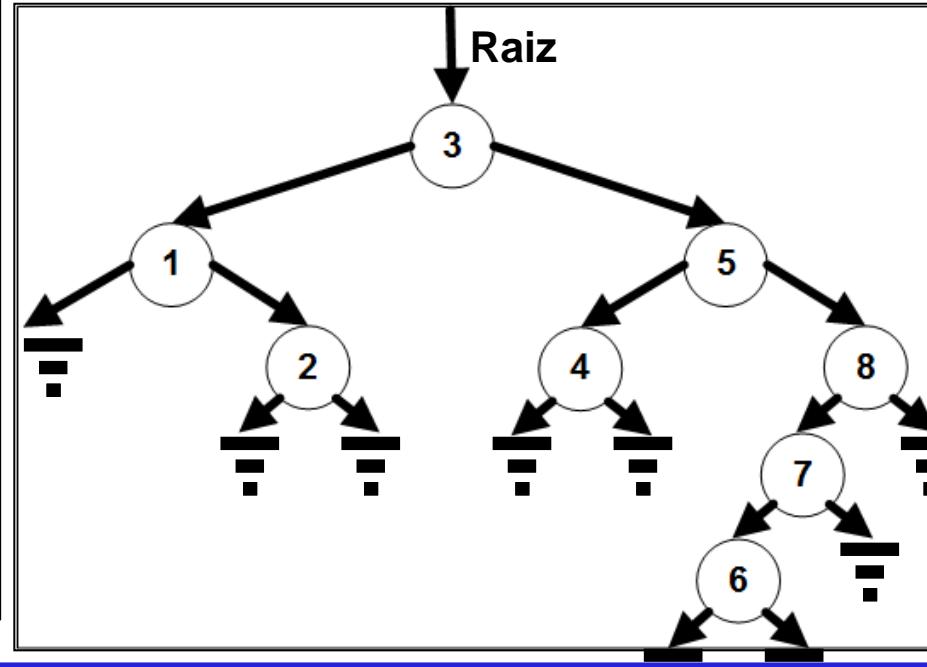
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq=remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) { i = i.esq;
    } else if(i.esq == null) { i = i.dir;
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}

```

raiz n(3) x 1



# Algoritmo em Java - Classe Árvore Binária

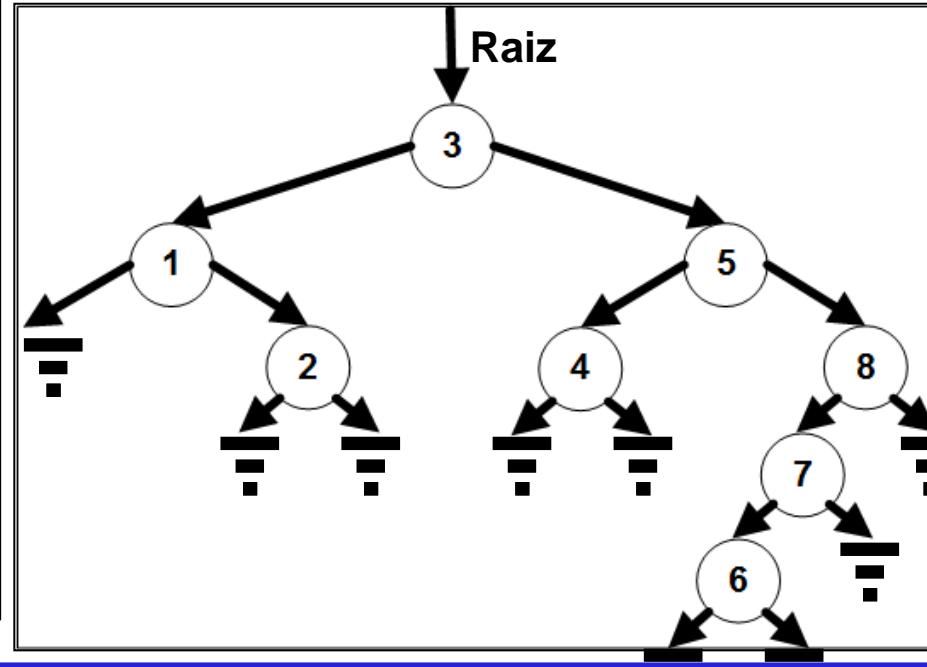
```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1



# Algoritmo em Java - Classe Árvore Binária

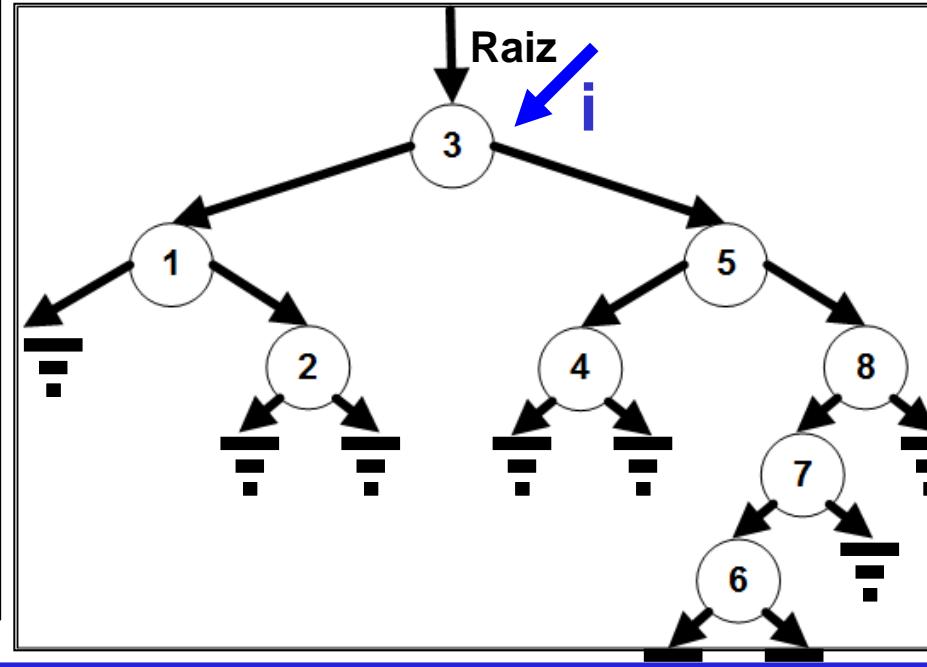
```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

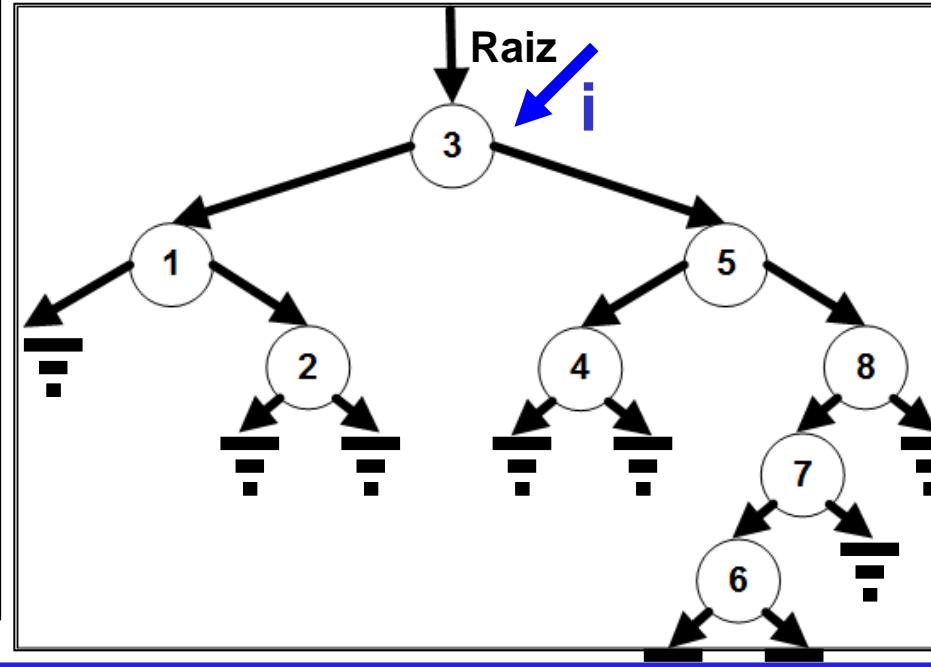
```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}                                false: n(3) == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

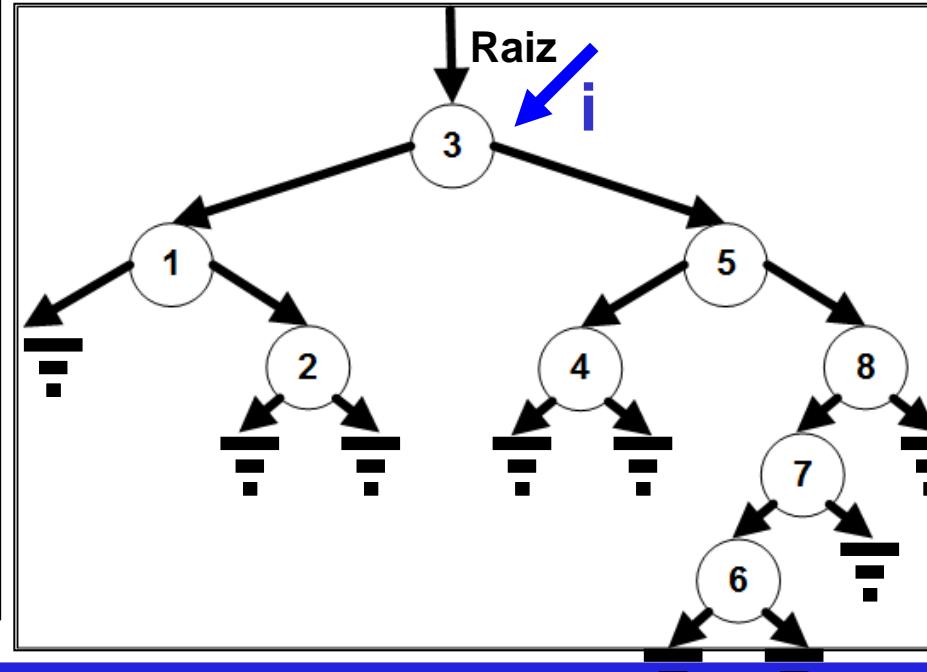
```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento){ i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}
true: 1 < 3

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

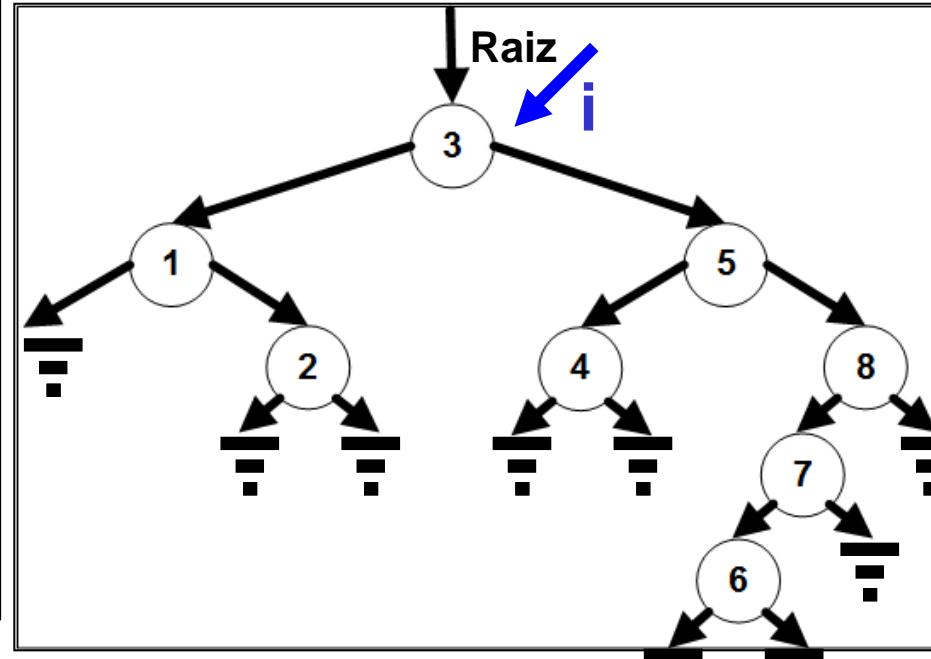
```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```

//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

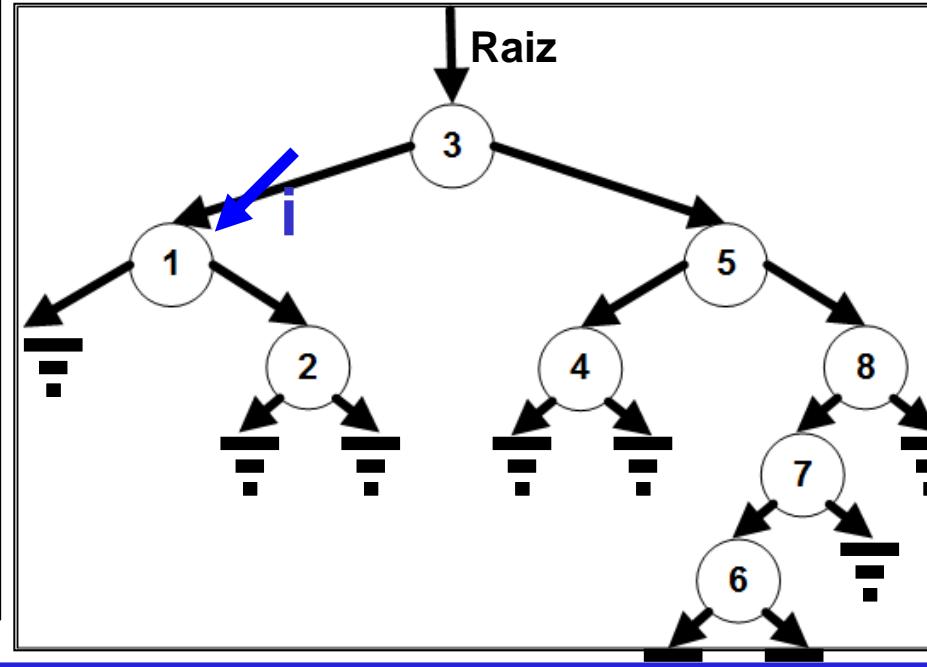
private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    else if(i.dir == null) { i = i.esq; }
    else if(i.esq == null) { i = i.dir; }
    else { i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}

```

raiz n(3) x 1 x 1 i n(3)

x 1 i n(1)



# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho

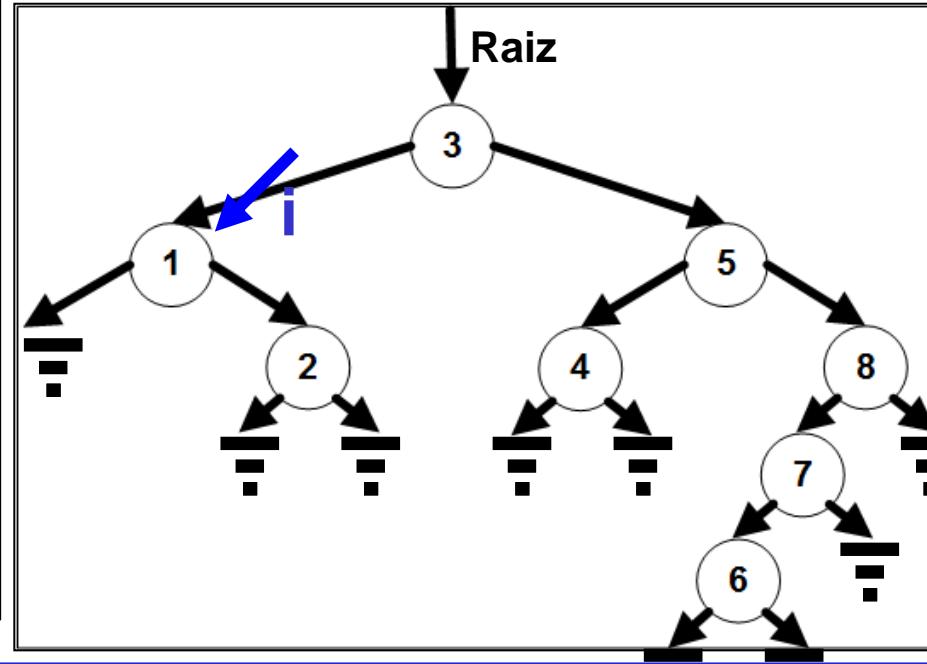
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
false: n(1) == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)

x 1 i n(1)



# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho

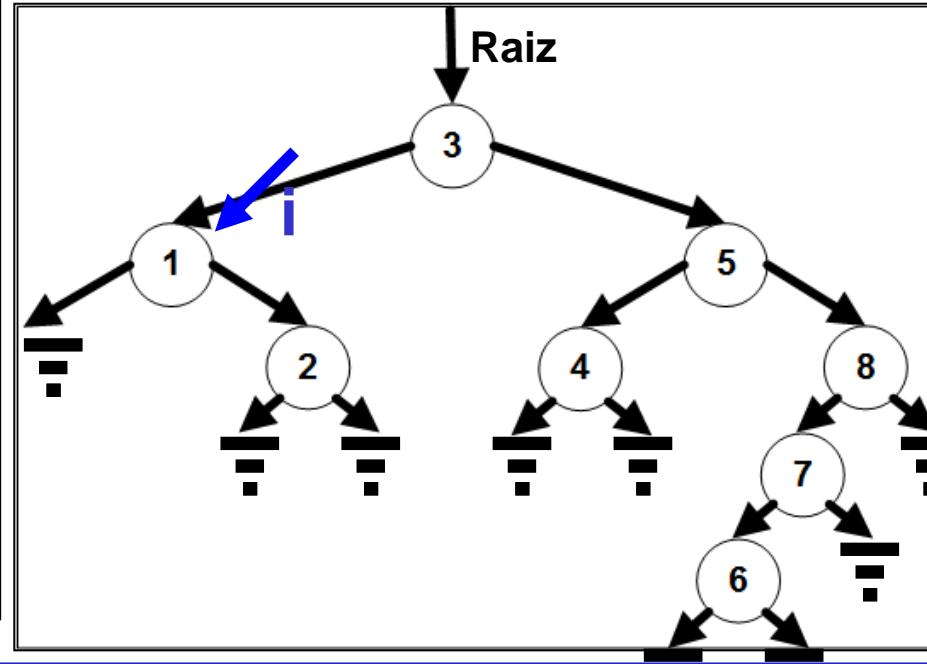
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento){ i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
false: 1 < 1

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)

x 1 i n(1)



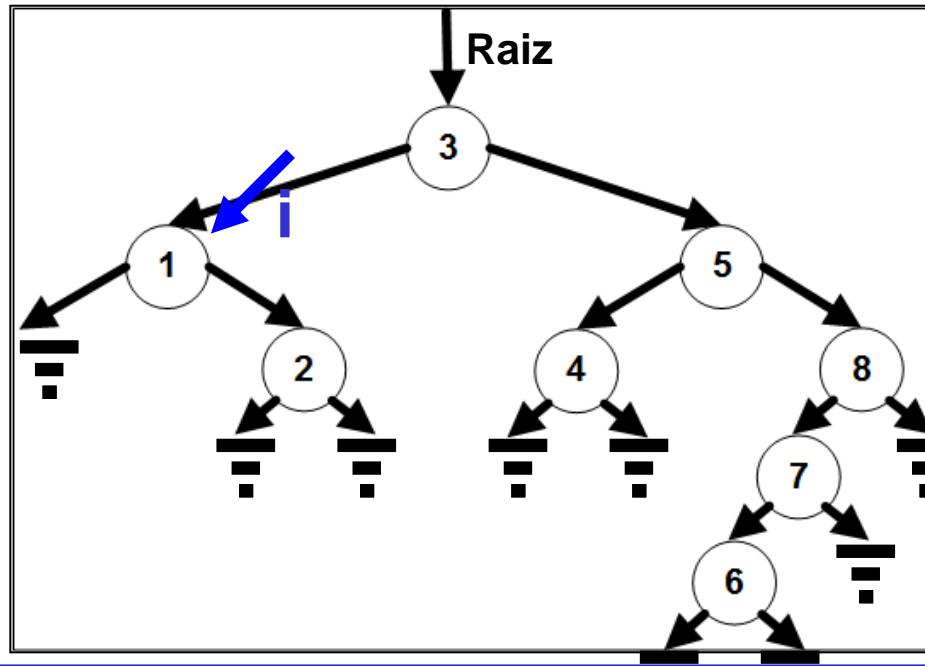
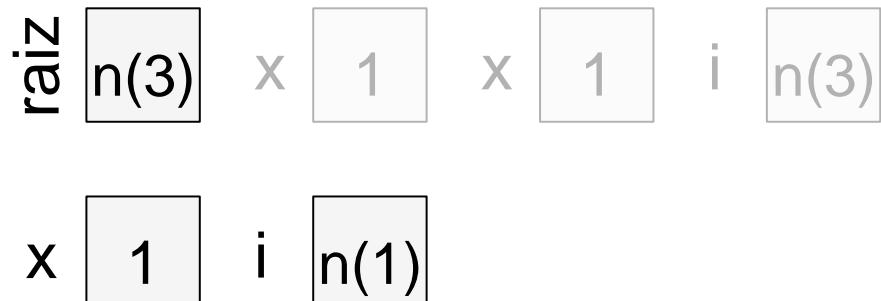
# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
false: 1 > 1

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho

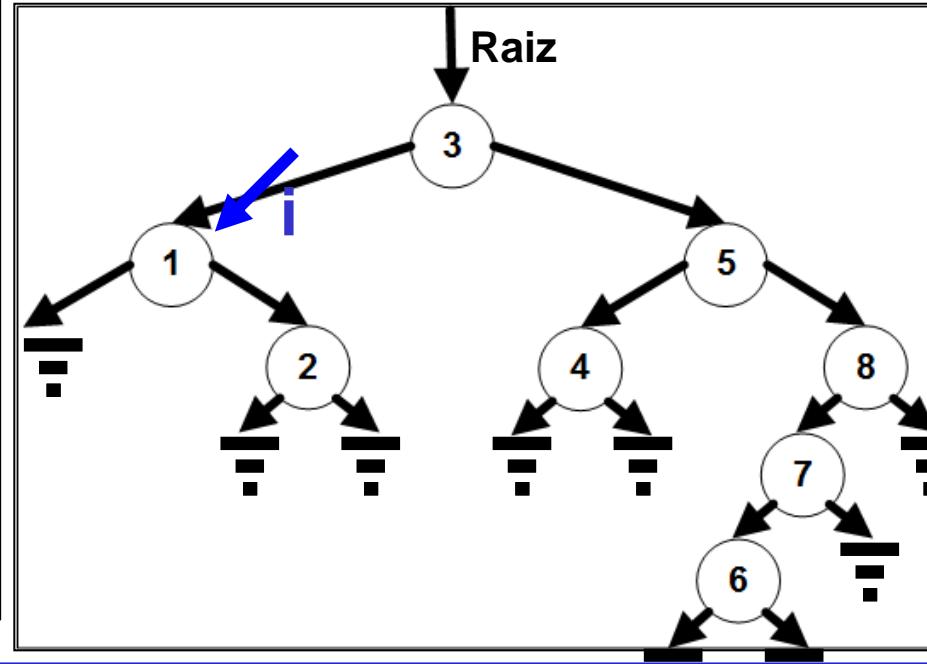
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
        return i;
    }
    false: n(2) == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}
}
```

raiz n(3) x 1 x 1 i n(3)

x 1 i n(1)



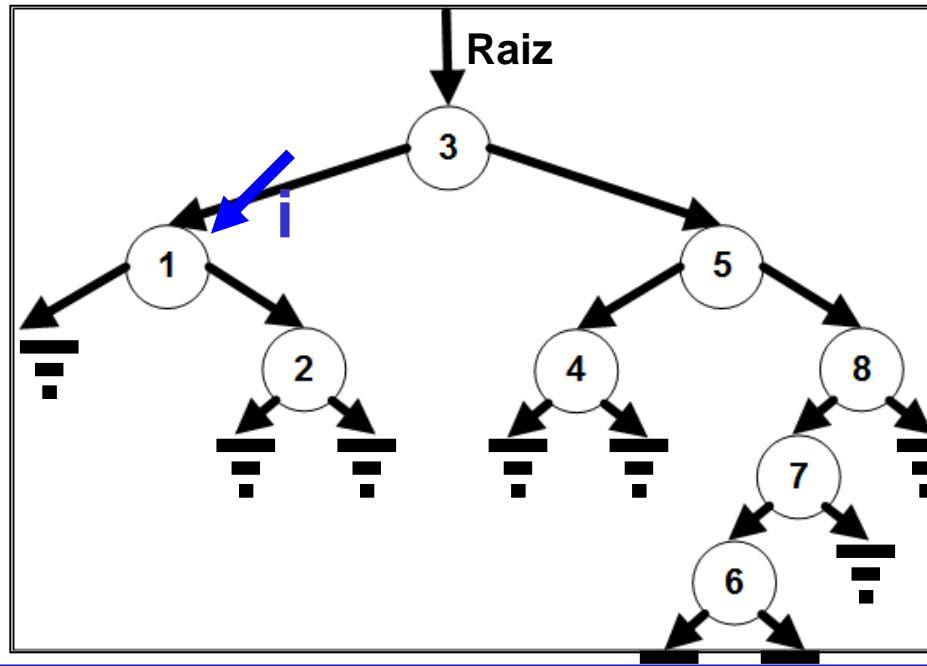
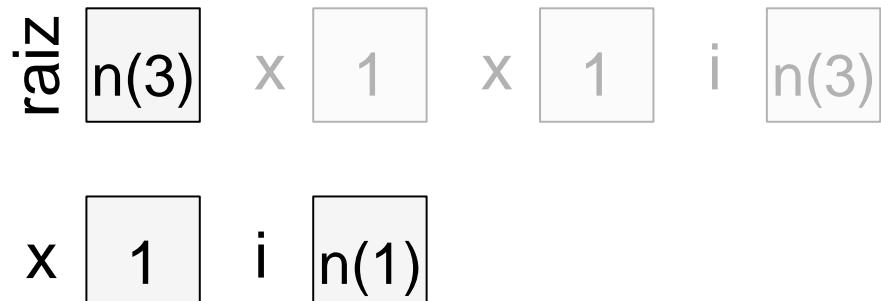
# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir;
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
true: null == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}
```



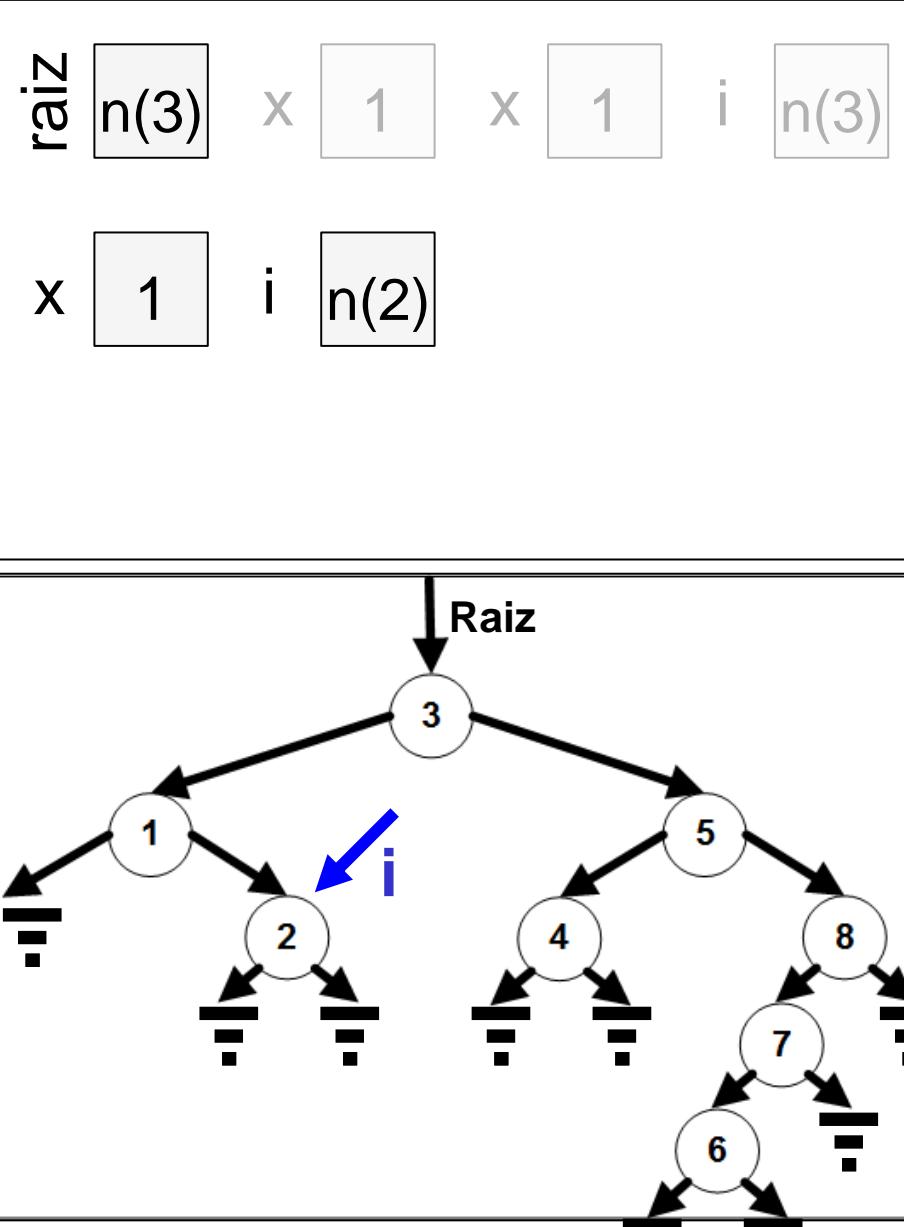
# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) {      throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq=remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) {   i = i.esq;
    } else if(i.esq == null) {   i = i.dir;
    } else {                  i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {                  i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

//remover(1), um filho

```
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

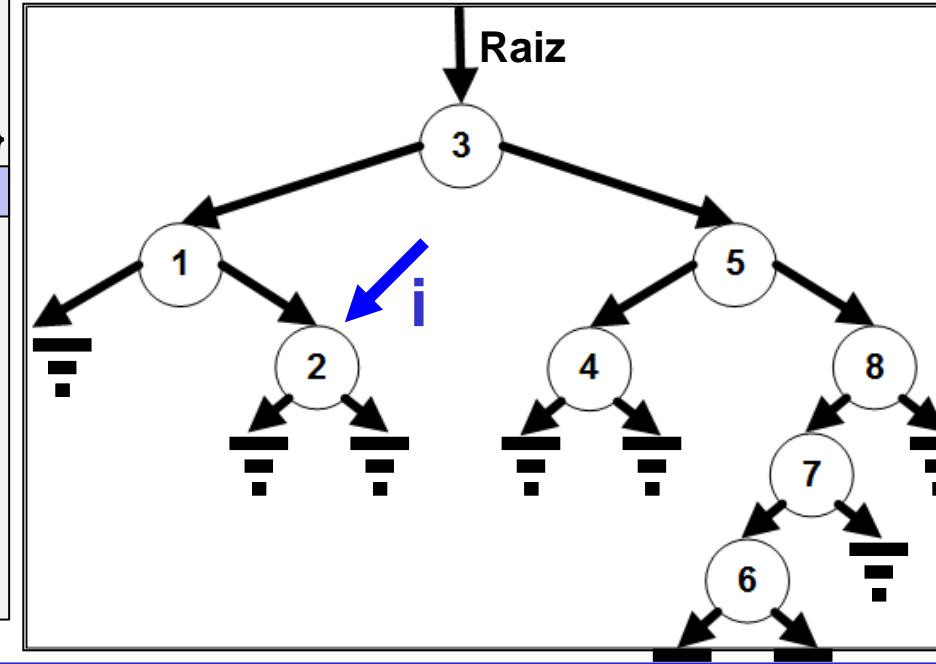
private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
```

Retorna n(2)

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)

x 1 i n(2)



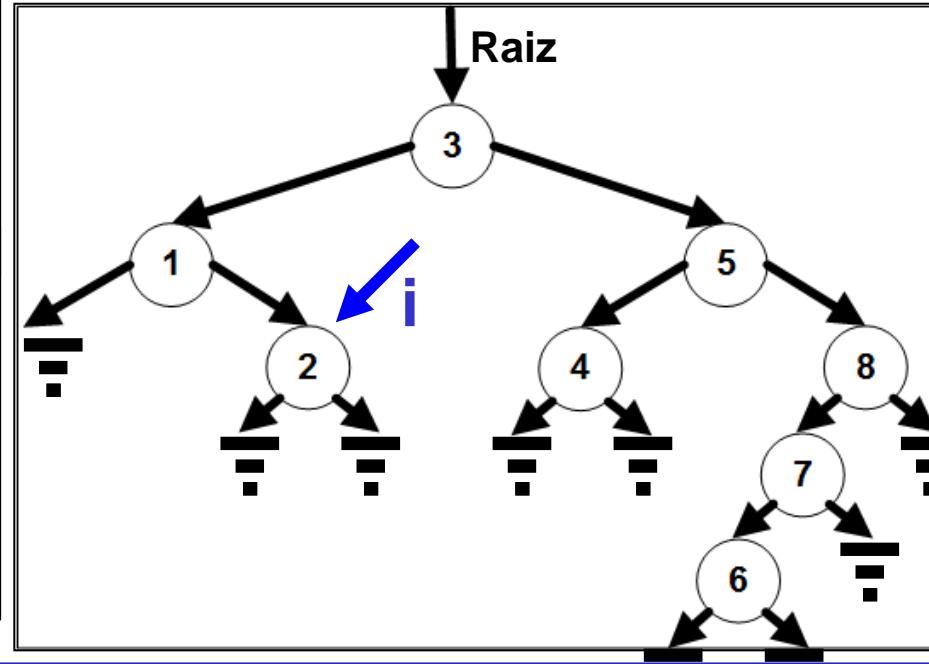
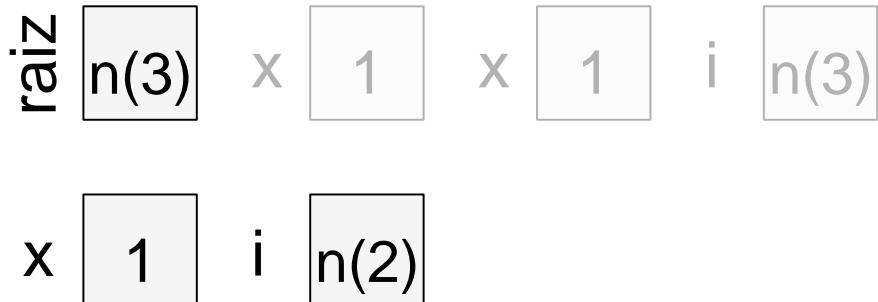
# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho
```

```
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
```

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

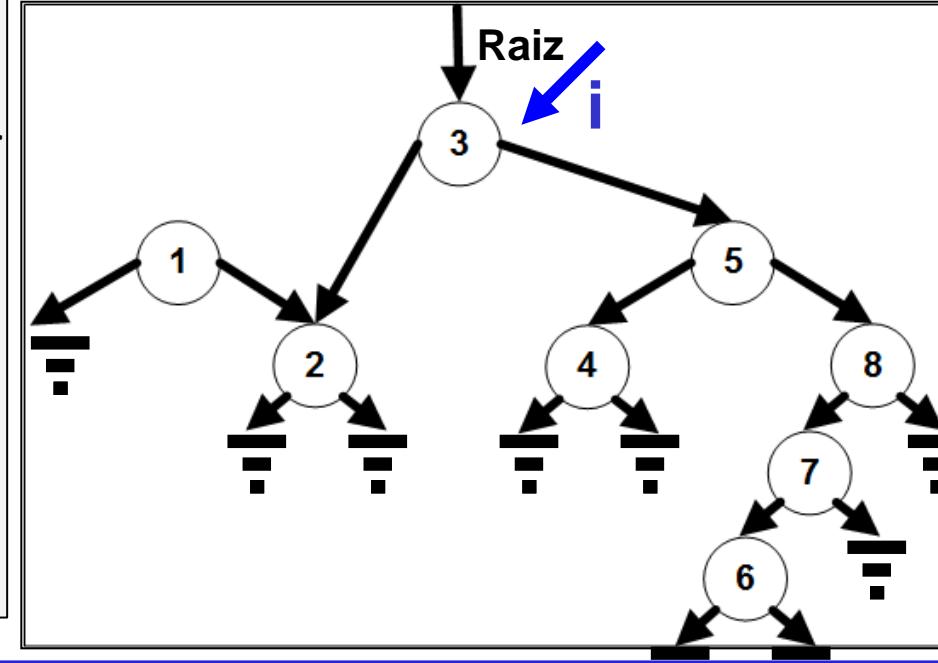
```
//remover(1), um filho

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho
```

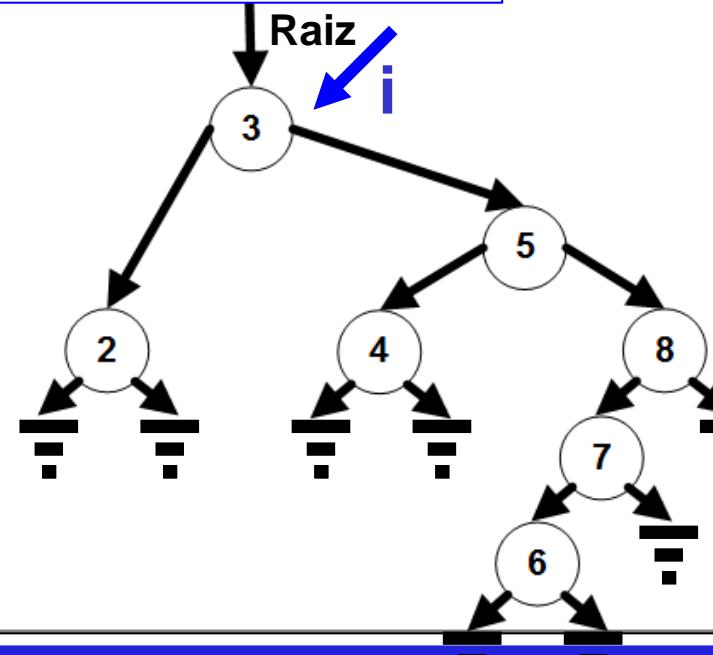
```
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

Após a coleta de lixo do Java  
 (que não controlamos quando ela acontece)...

```
if (i == null) return i;
else if (i.esq == null) { i = i.dir;
} else { i.esq = anterior(i, i.esq); }
return i;
```

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento = j.elemento; j = j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho
```

```
public void remover(int x) throws Exception {
```

```
    raiz = remover(x, raiz);
```

```
}
```

```
private
```

```
    if (i ==
```

```
    } else
```

```
    } else
```

```
        i.esq = anterior(i.esq, i.dir);
```

```
    } else if(i.dir == null) { i = i.esq;
```

```
    } else if(i.esq == null) { i = i.dir;
```

```
    } else { i.esq = anterior(i, i.esq); }
```

```
    return i;
```

```
}
```

```
private No anterior(No i, No j) {
```

```
    if (j.dir != null) j.dir = anterior(i, j.dir);
```

```
    else { i.elemento = j.elemento; j = j.esq; }
```

```
    return j;
```

```
}
```

De uma forma mais organizada ...



# Algoritmo em Java - Classe Árvore Binária

//remover(1), um filho

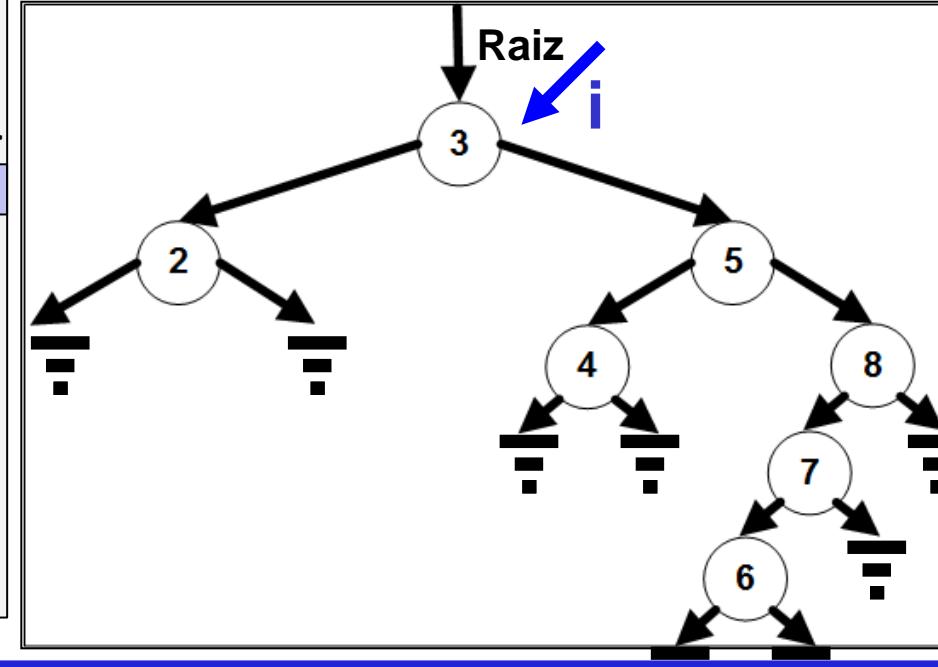
```
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
```

Retorna n(3)

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

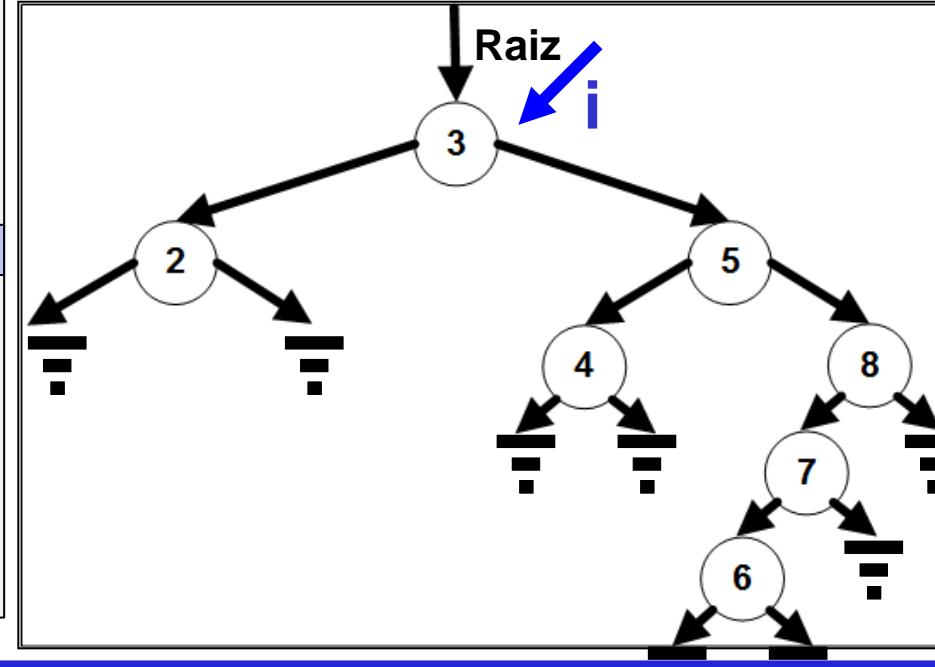
```
//remover(1), um filho
```

```
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
```

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    }
    return j;
}
```

raiz n(3) x 1 x 1 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//remover(1), um filho
```

```
public void remover(int x) throws Exception {
```

```
    raiz = remover(x, raiz);
```

```
}
```

```
private No remover(int x, No i) throws Exception {
```

```
    if (i == null) { throw new Exception("Erro!"); }
```

```
    } else if(x < i.elemento) { i.esq = remover(x, i.esq); }
```

```
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
```

```
    } else if(i.dir == null) { i = i.esq;
```

```
    } else if(i.esq == null) { i = i.dir;
```

```
    } else { i.esq = anterior(i, i.esq); }
```

```
    return i;
```

```
}
```

```
private No anterior(No i, No j) {
```

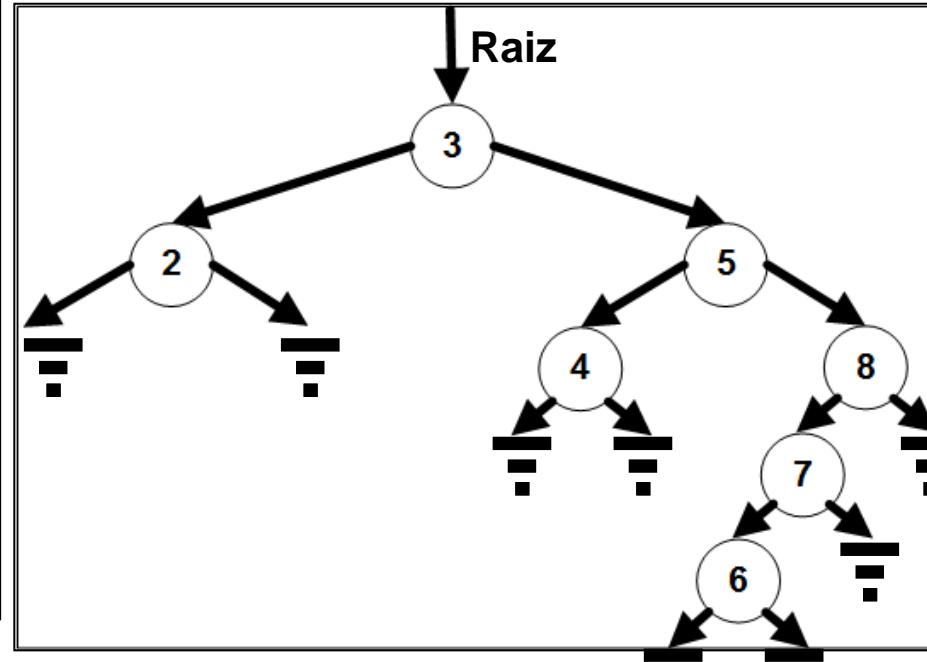
```
    if (j.dir != null) j.dir = anterior(i, j.dir);
```

```
    else { i.elemento = j.elemento; j = j.esq; }
```

```
    return j;
```

```
}
```

raiz n(3) x 1



# Algoritmo em Java - Classe Árvore Binária

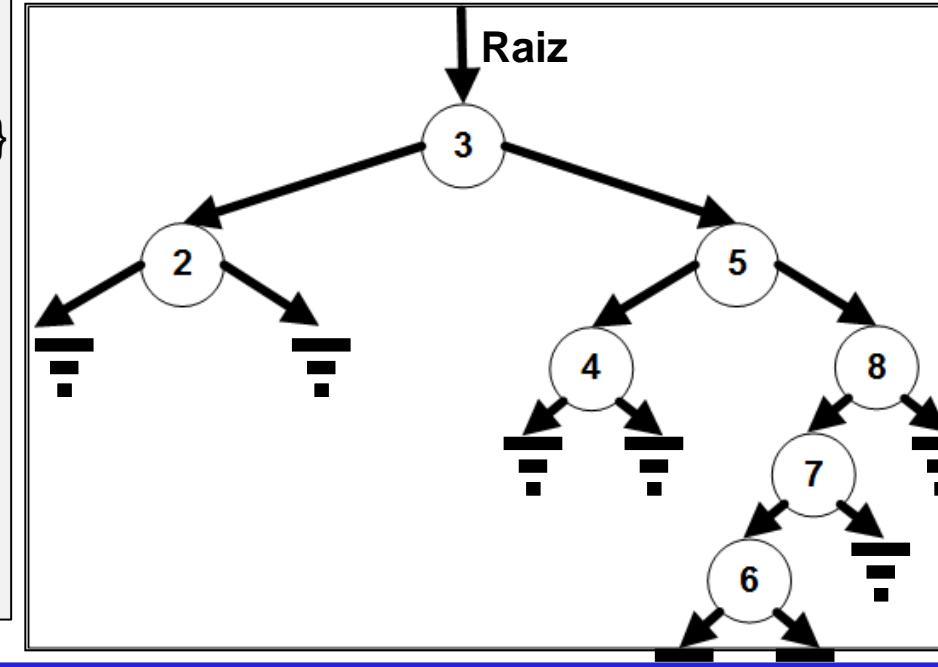
```
//remover(1), um filho
```

```
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

```
private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}
```

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz  
n(3)

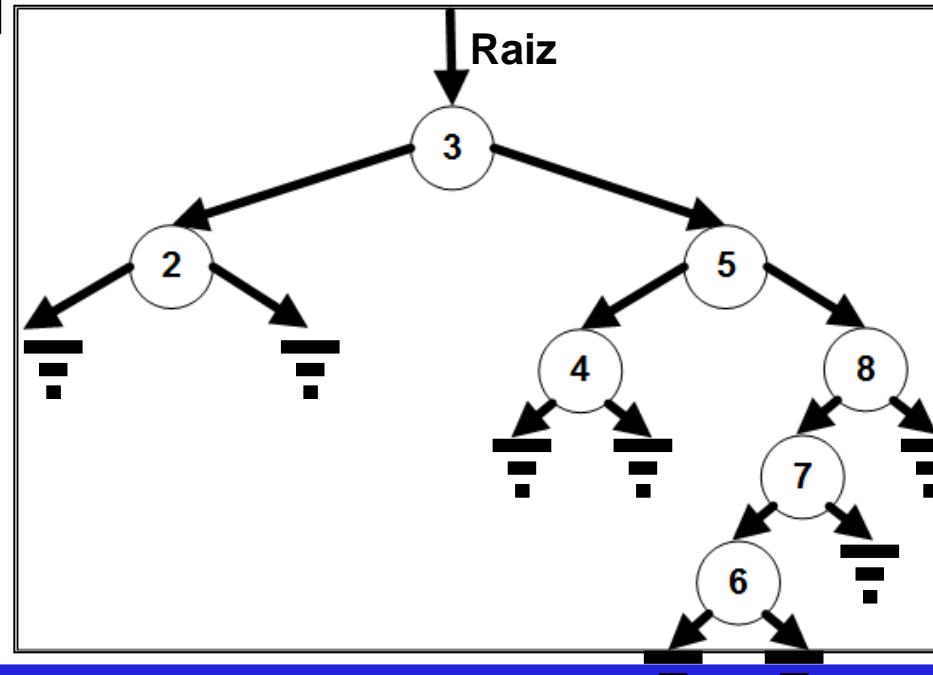


# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)

Voltando com o 1 antes  
de fazer outra remoção

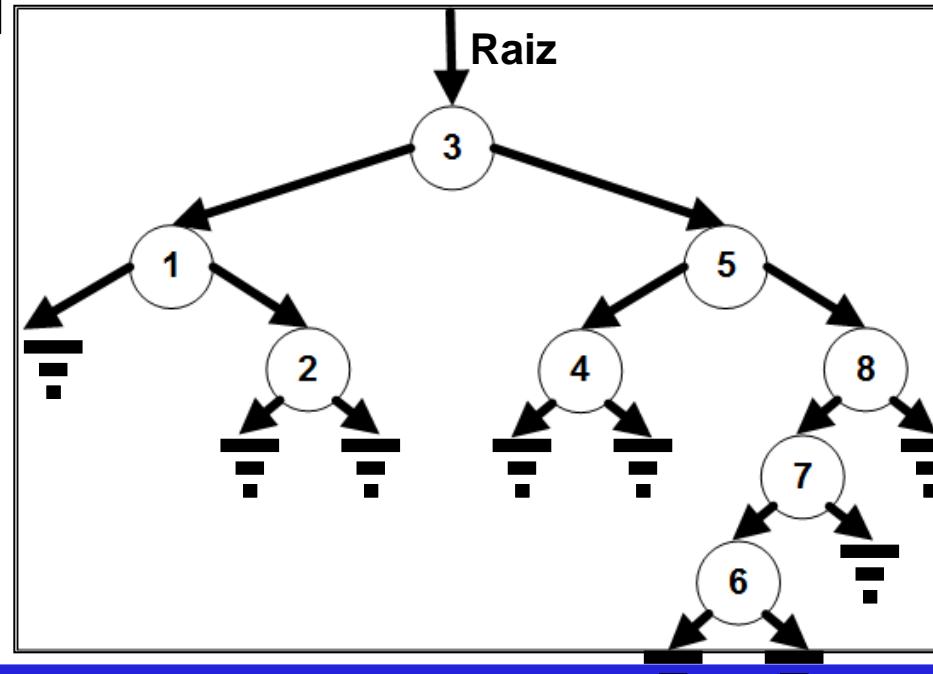


# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

raiz  
n(3)

Vamos remover o 3 (tem dois filhos) de nossa árvore



# Algoritmo em Java - Classe Árvore Binária

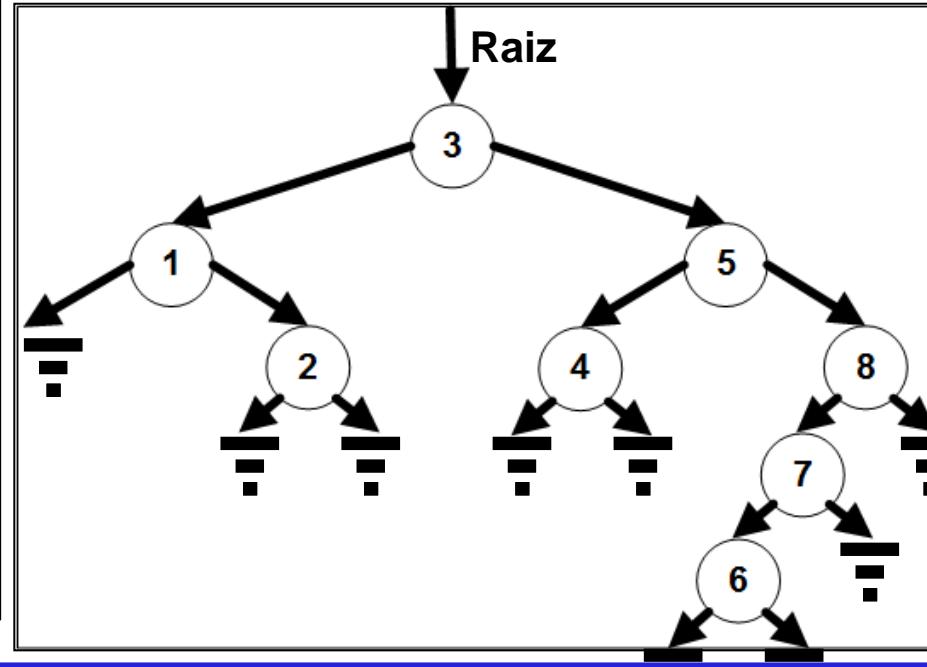
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3



# Algoritmo em Java - Classe Árvore Binária

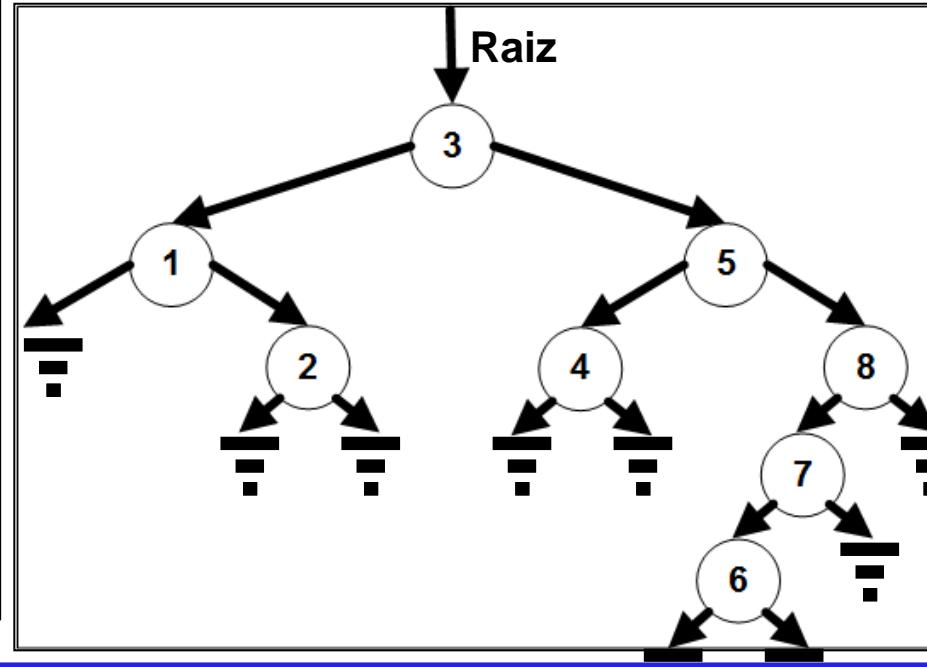
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3



# Algoritmo em Java - Classe Árvore Binária

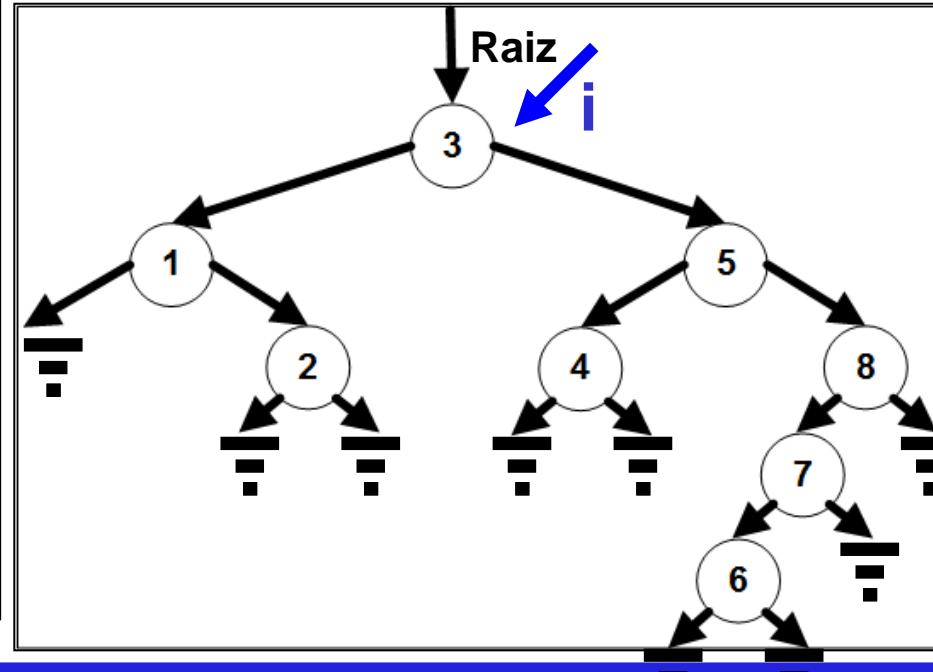
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

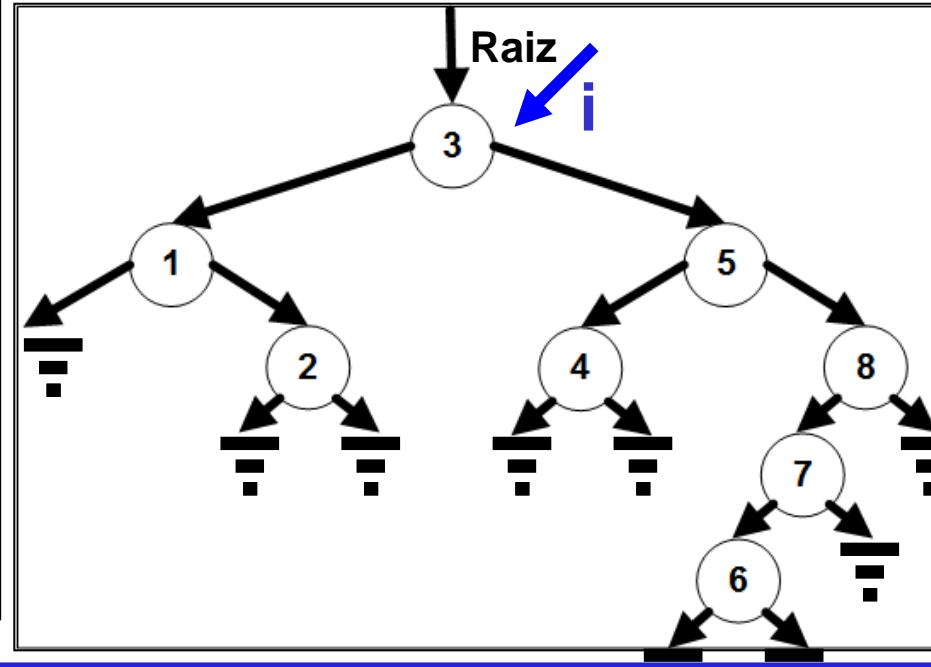
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}                                false: n(3) == null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

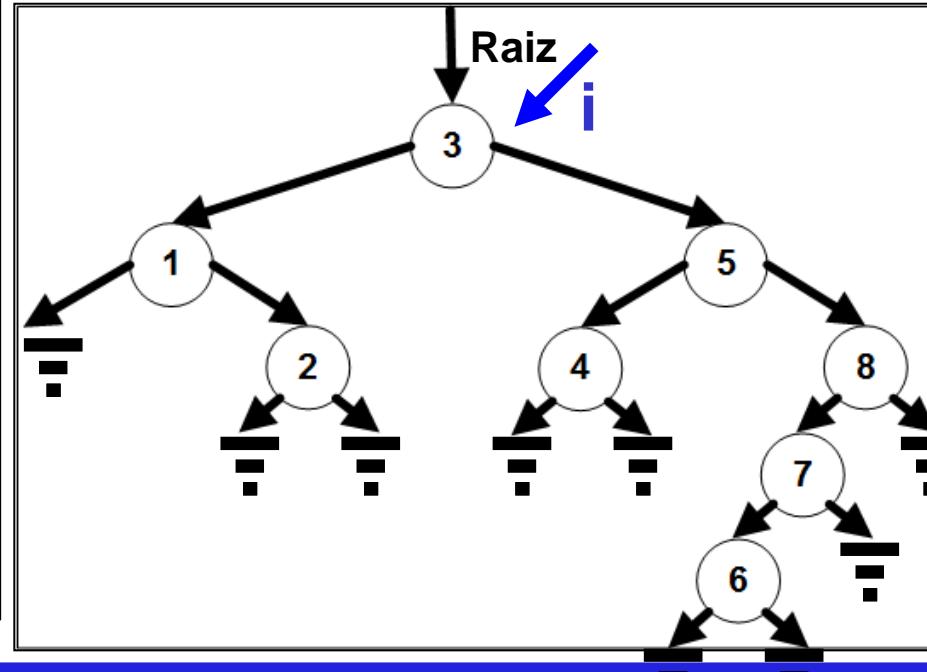
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento){ i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
    false: 3 < 3

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

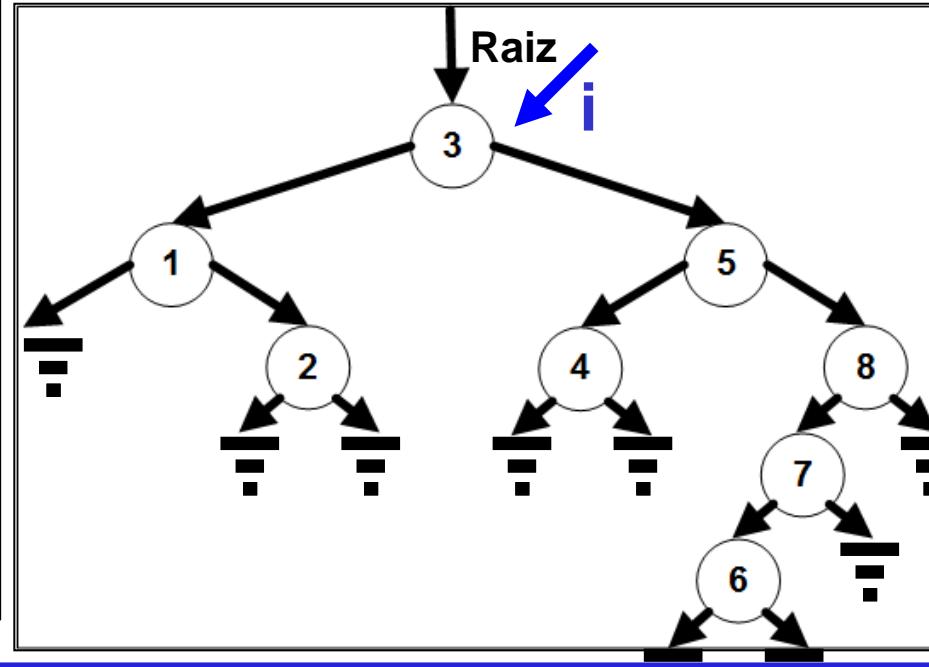
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}                                false: 3 > 3

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

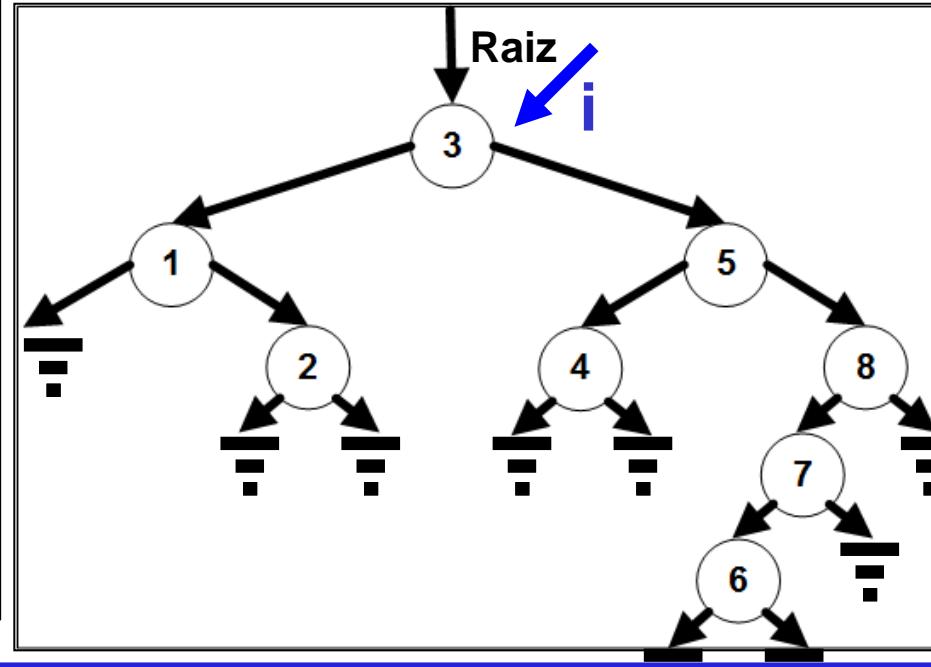
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
false: n(5) == false

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

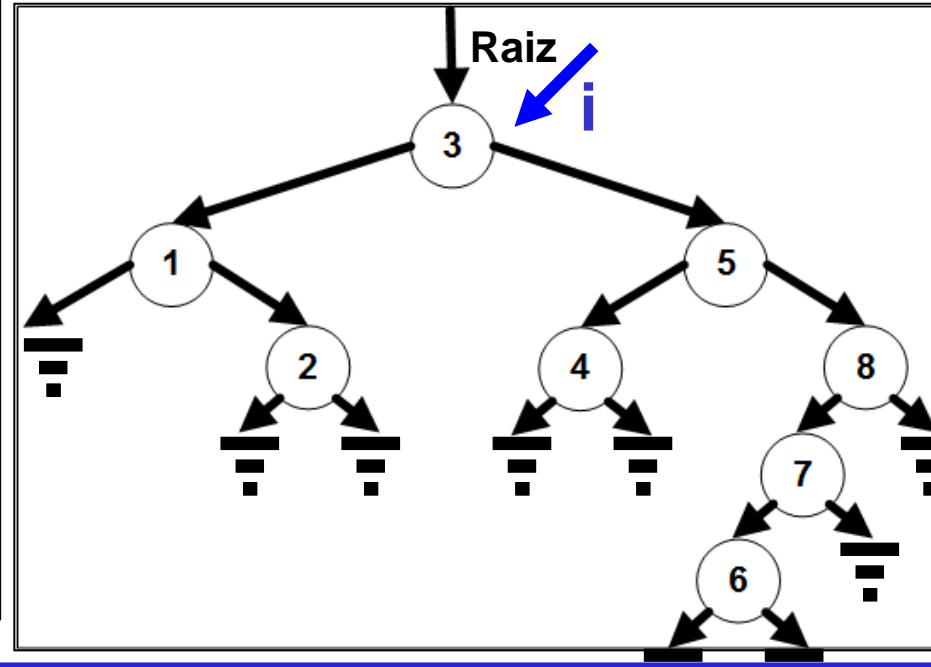
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir;
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
false: n(1) == false

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

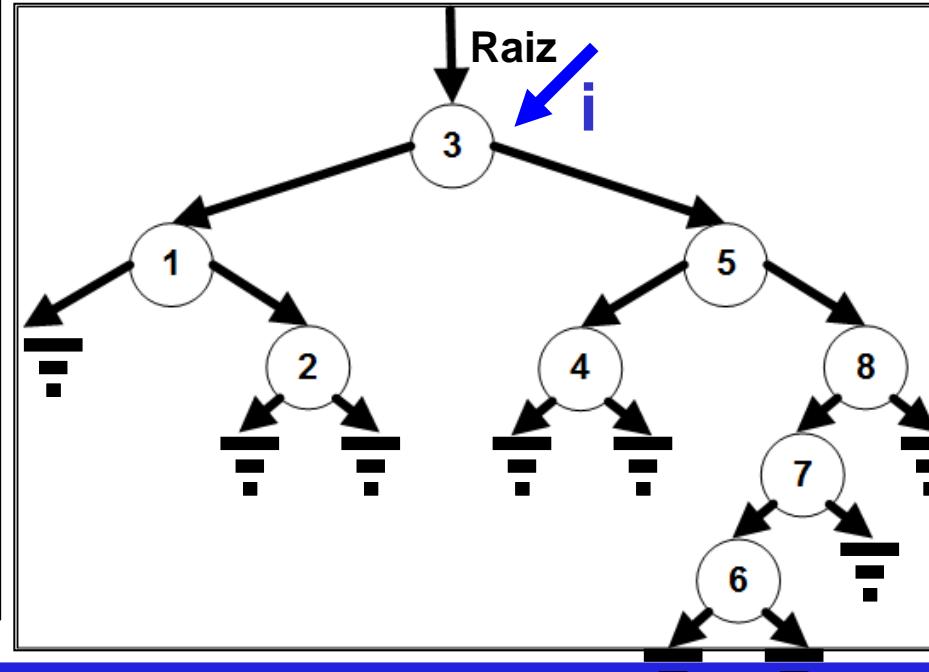
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

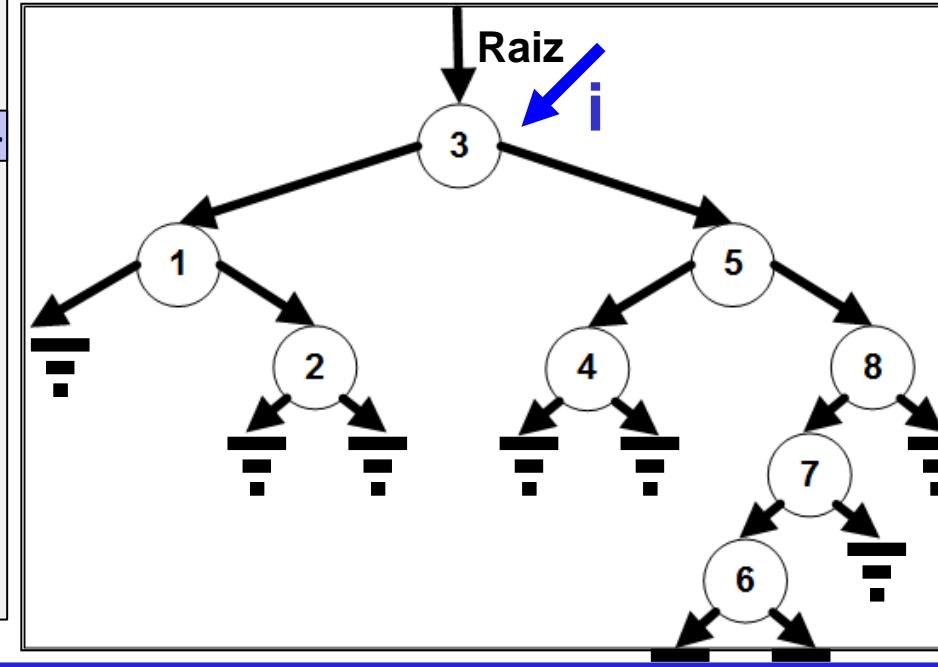
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```

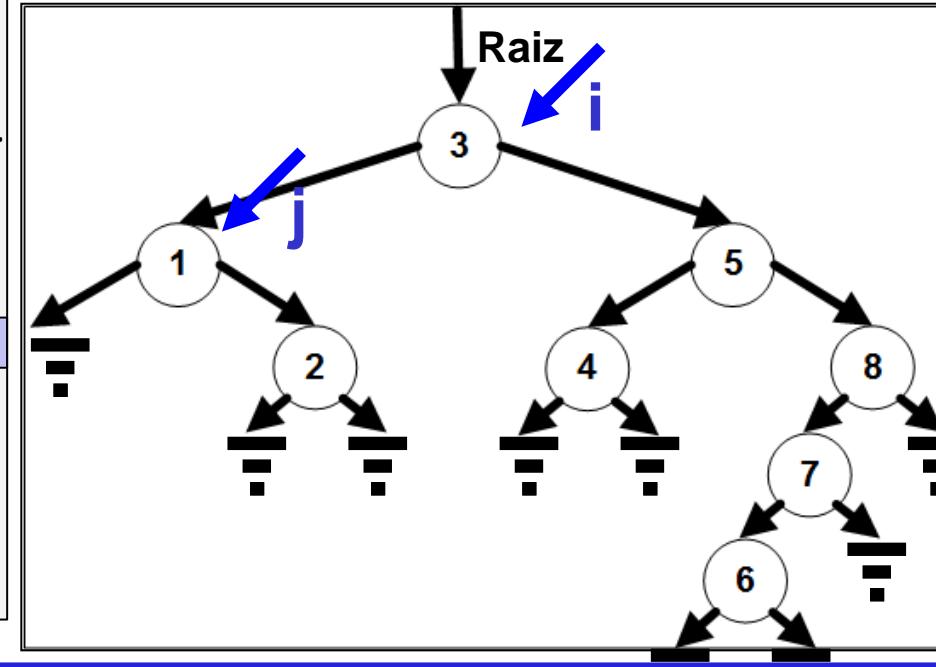
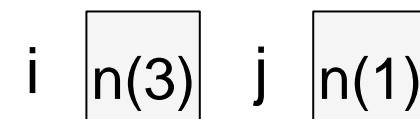
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    else if(i.dir == null) { i = i.esq; }
    else if(i.esq == null) { i = i.dir; }
    else { i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}

```



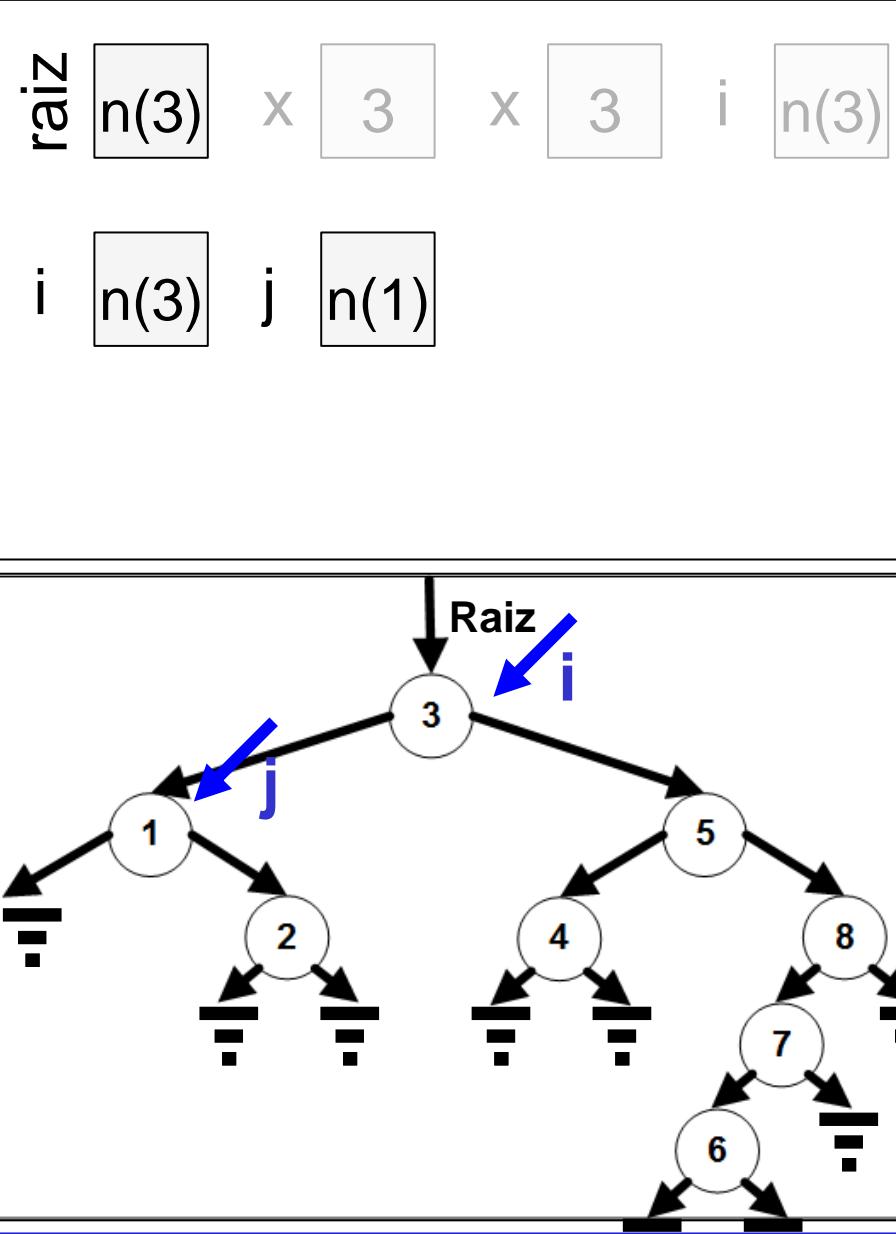
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
true: n(2) != null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    }
    return j;
}
```



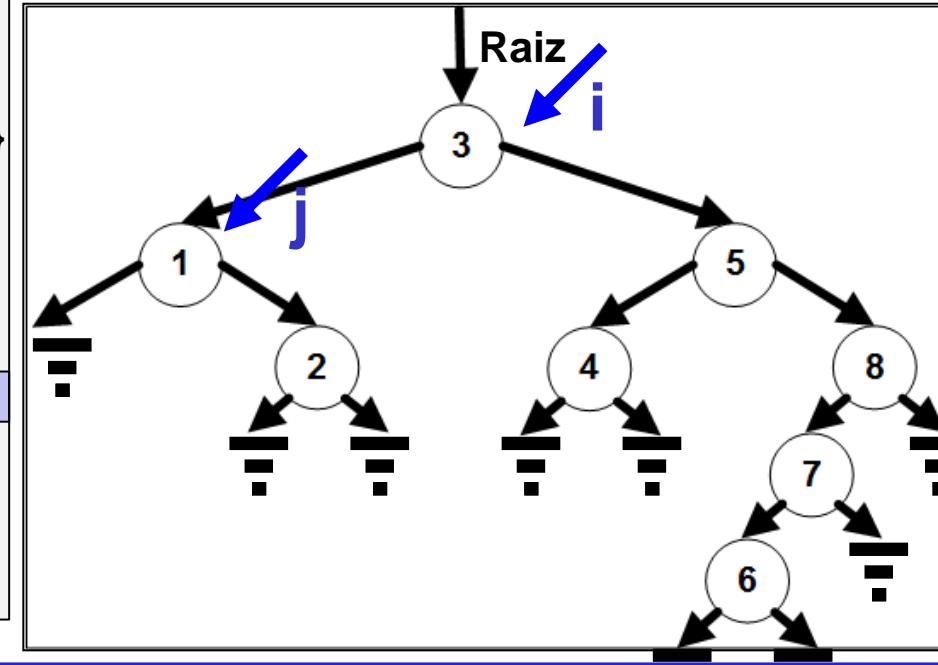
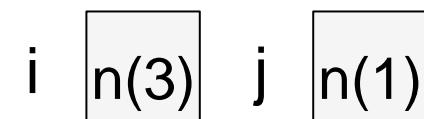
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



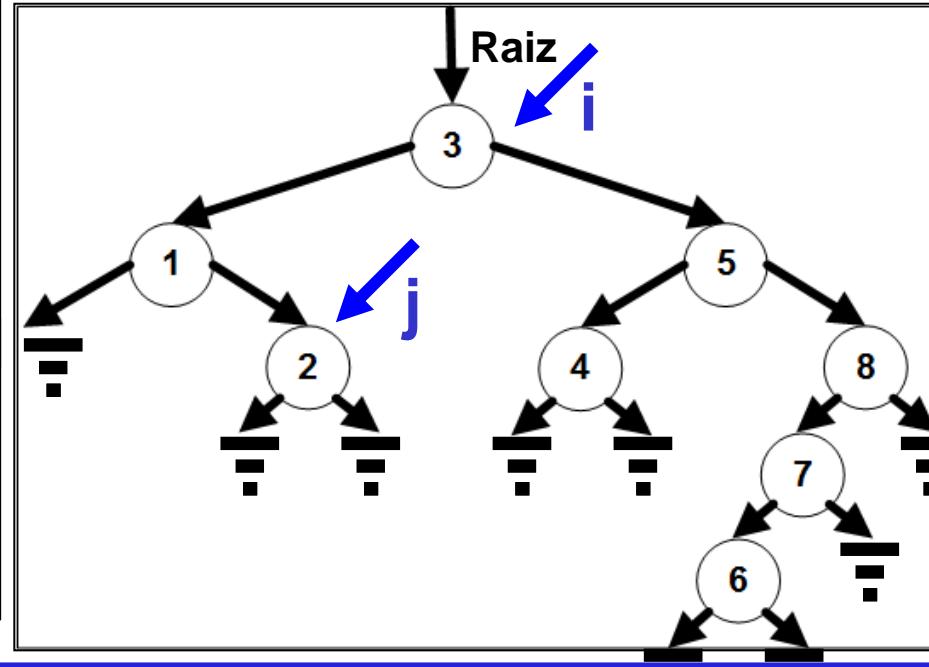
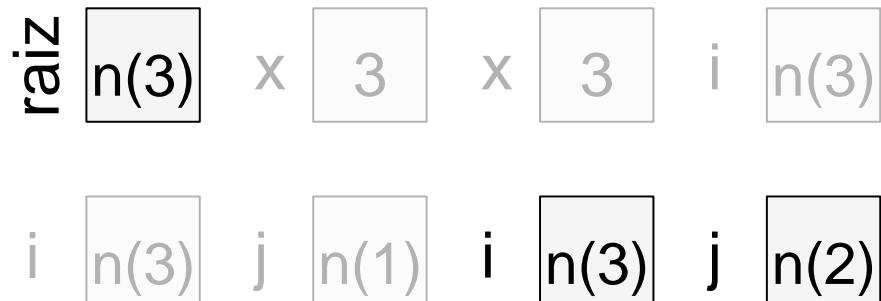
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



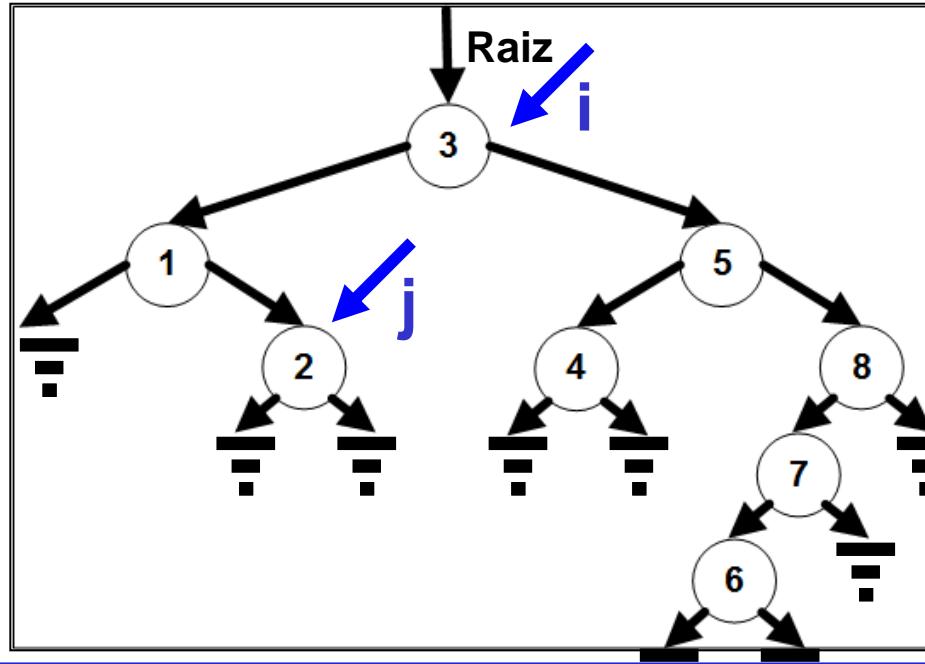
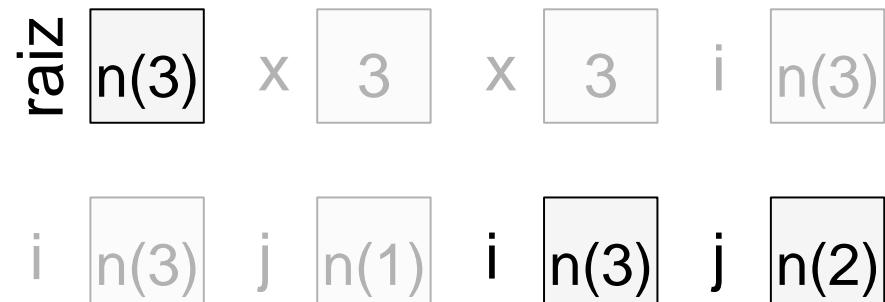
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}
false: null != null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



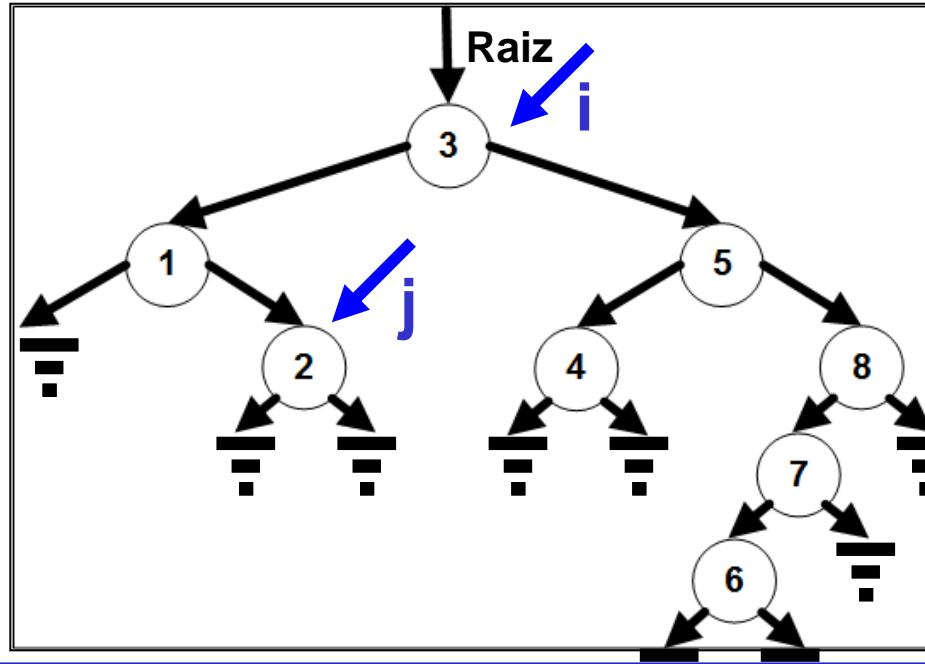
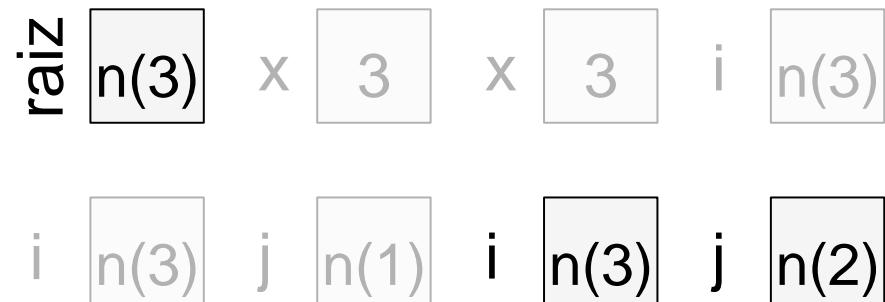
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq;
    }
    return j;
}
```



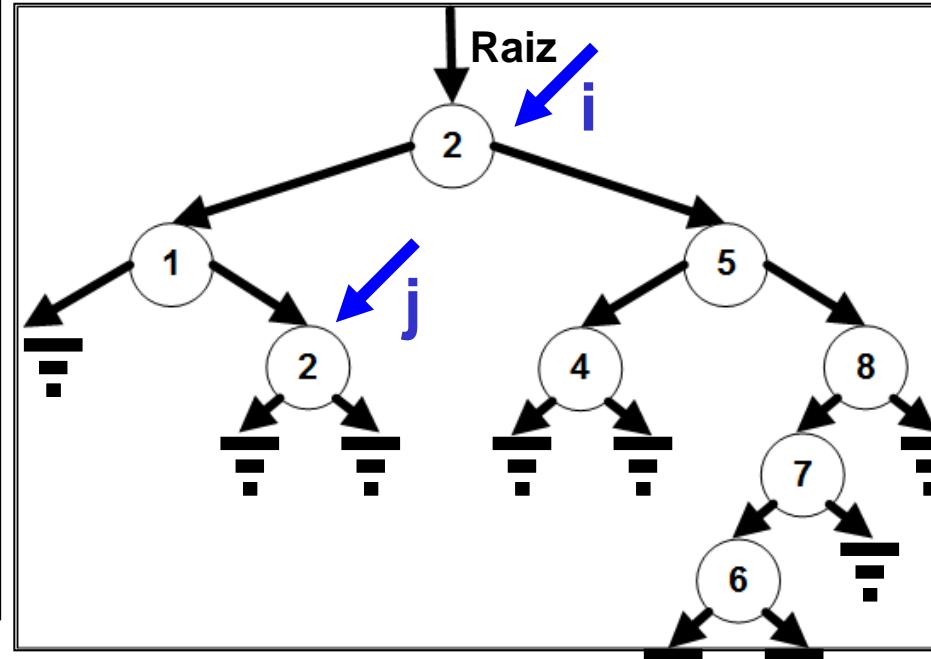
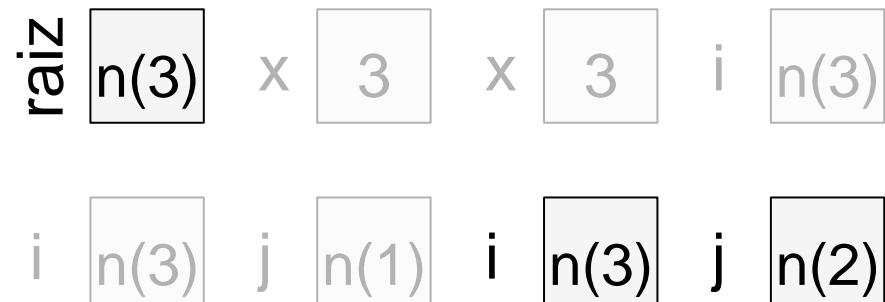
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



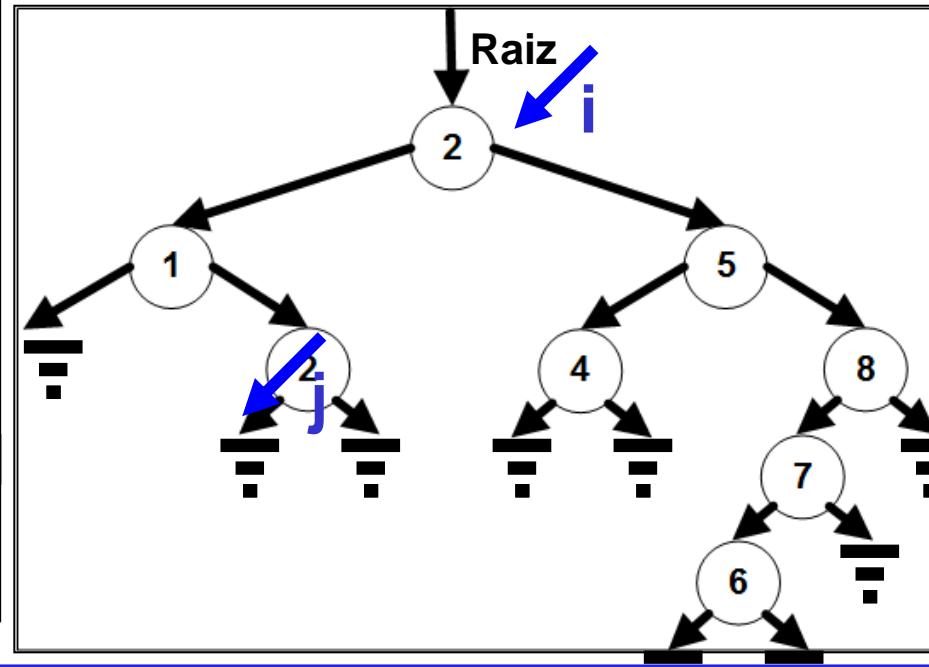
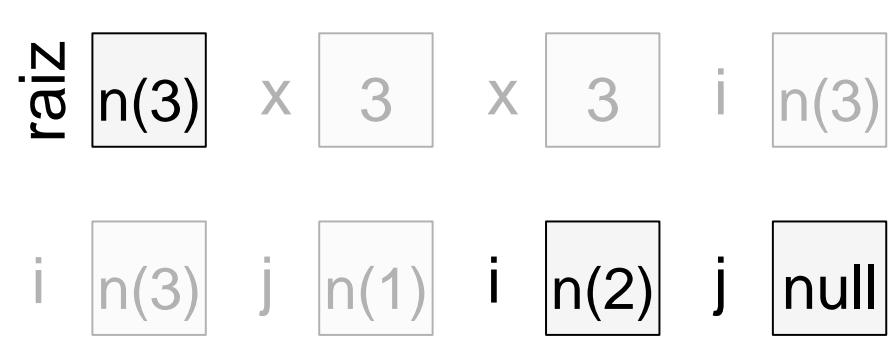
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento;j=j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

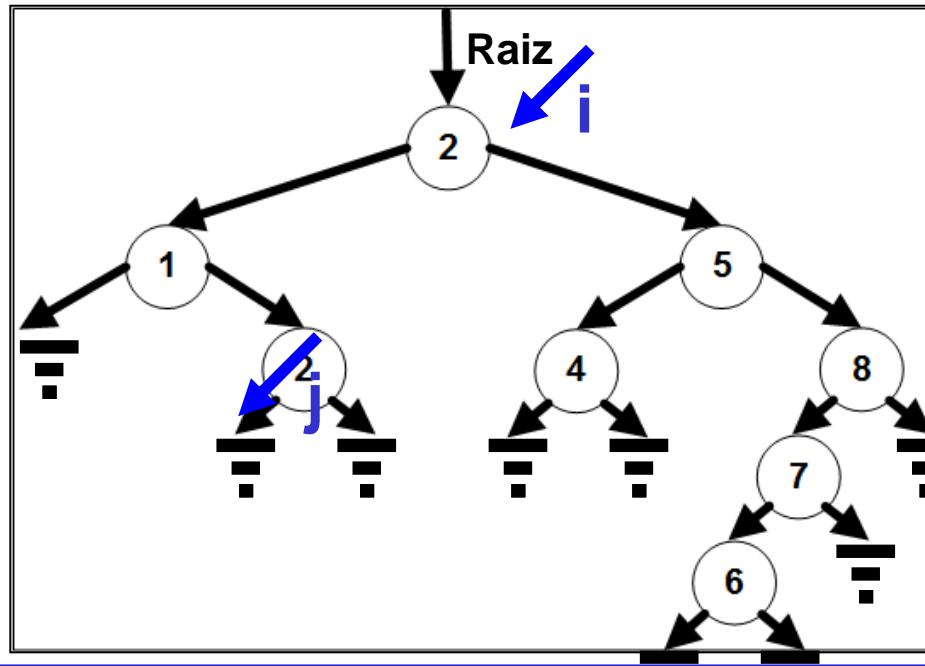
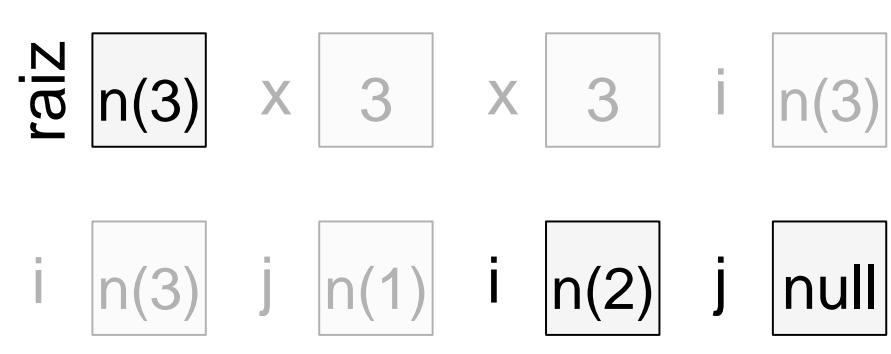
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

Retornando null

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



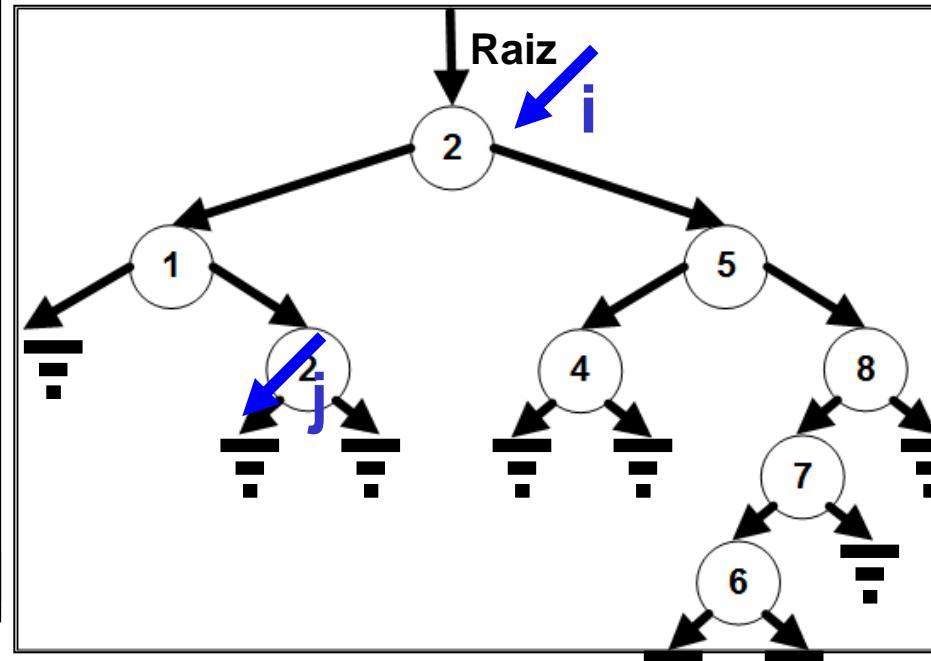
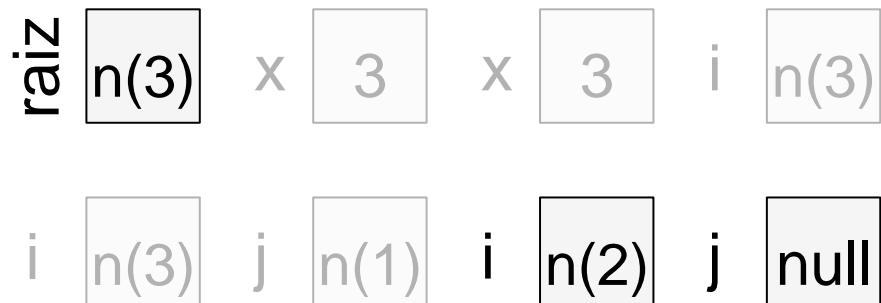
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
}
```



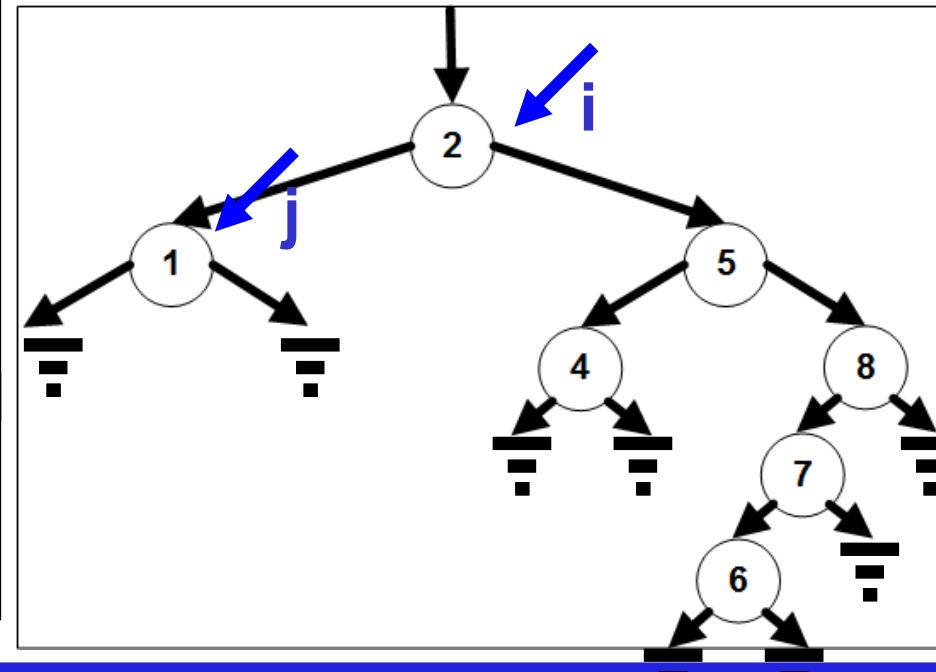
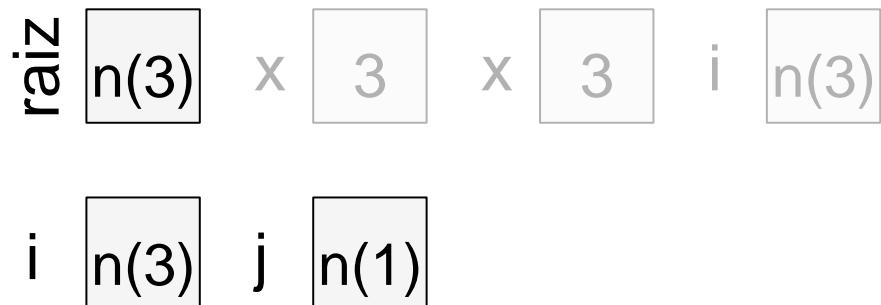
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

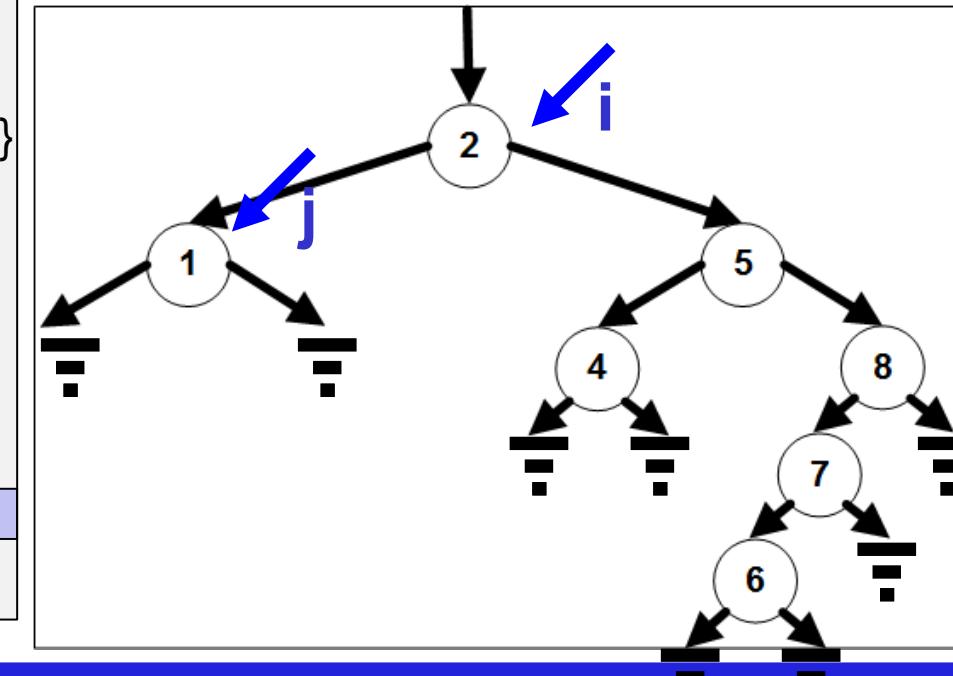
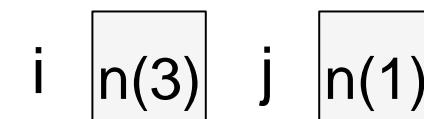
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

Retornando n(1)

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



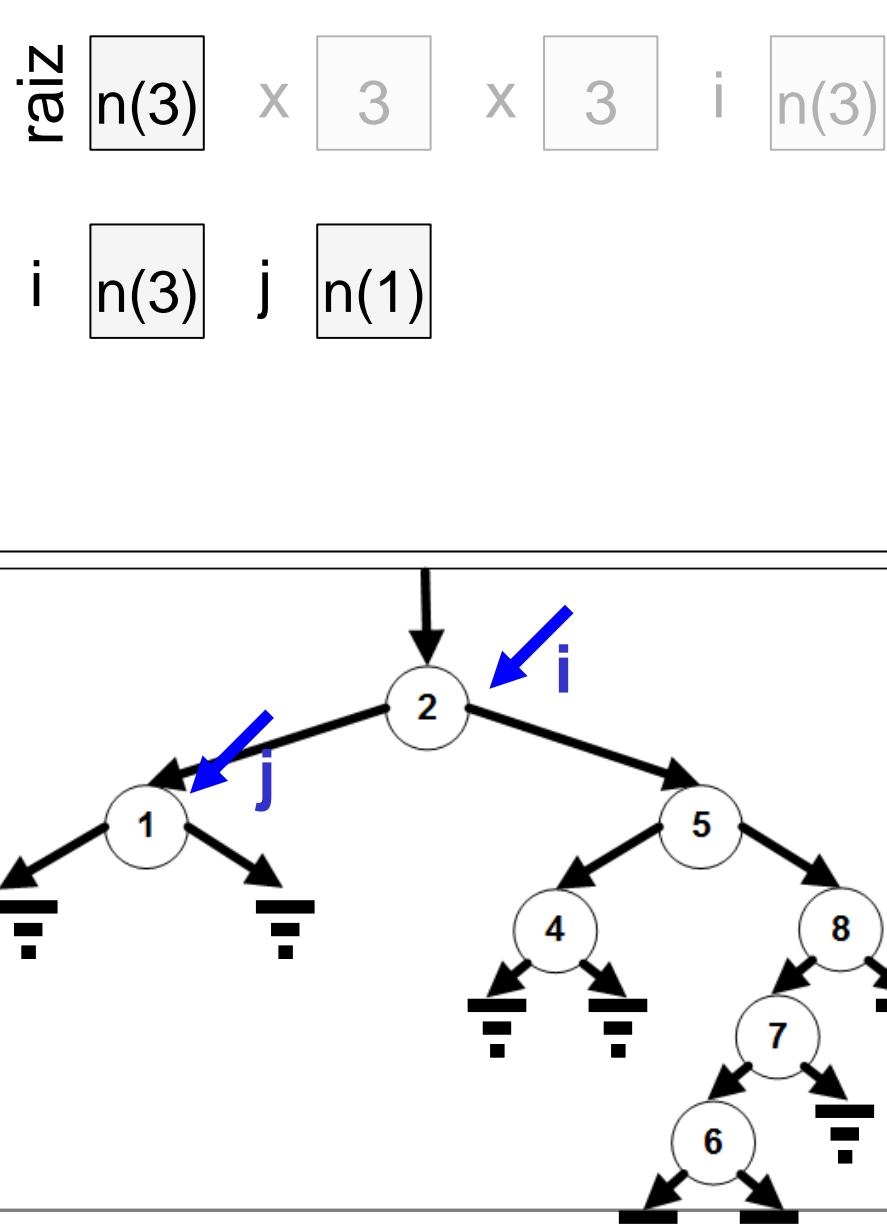
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) {      throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq=remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) {   i = i.esq;
    } else if(i.esq == null) {   i = i.dir;
    } else {                  i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {                  i.elemento=j.elemento; j=j.esq; }
    return j;
}
```



# Algoritmo em Java - Classe Árvore Binária

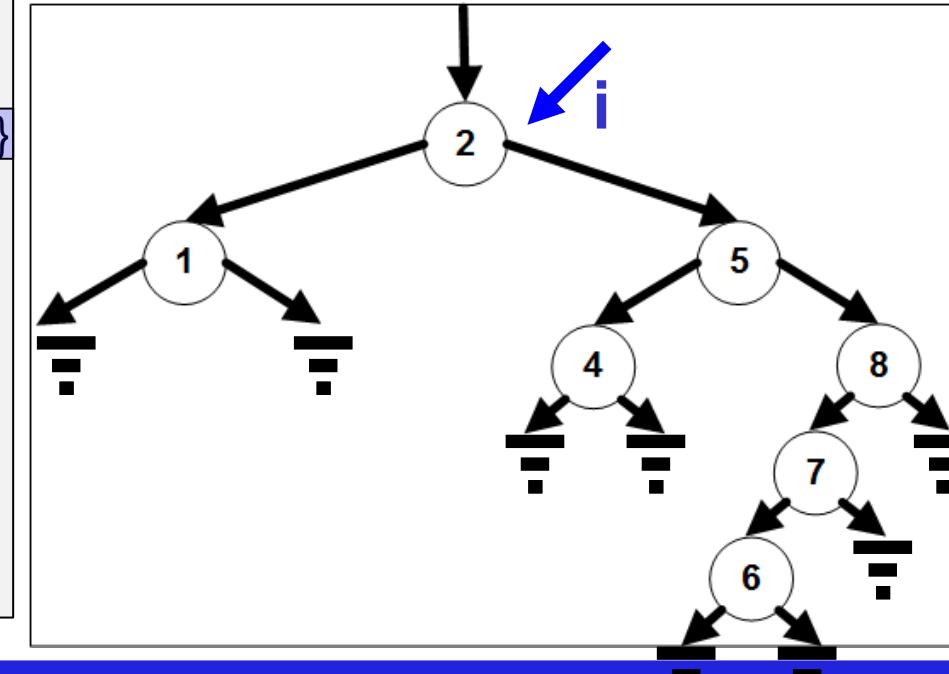
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else { i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3 x 3 i n(3)



# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

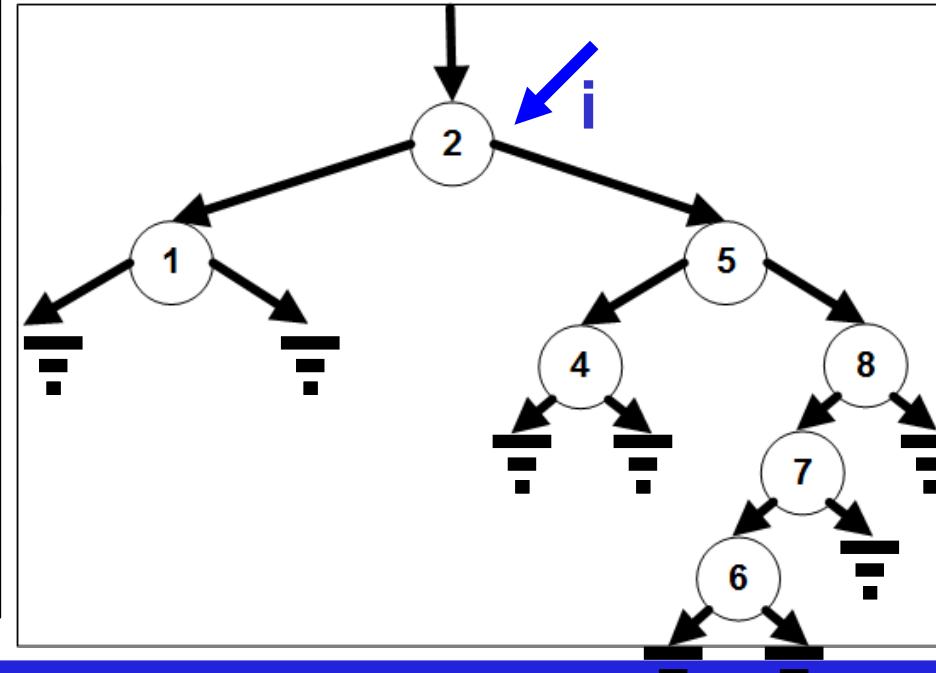
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq);
    }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

**Retorna n(3)**

raiz n(3) x 3 x 3 i n(3)



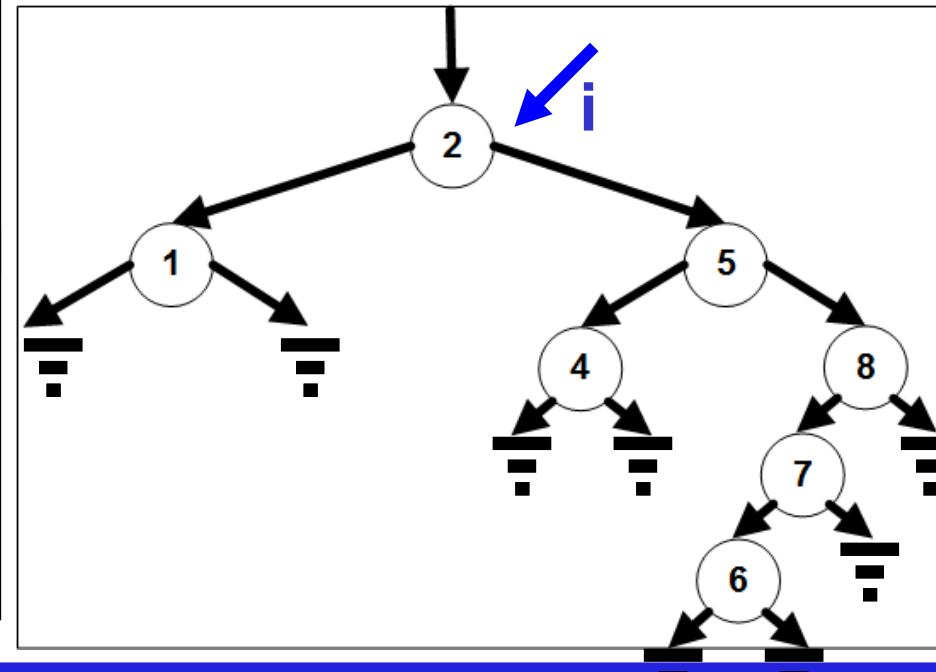
# Algoritmo em Java - Classe Árvore Binária

```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) {      throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq = remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) {   i = i.esq;
    } else if(i.esq == null) {   i = i.dir;
    } else {                      i.esq = anterior(i, i.esq); }
    return i;
}
```

```
private No anterior(No i, No j) {  
    if (j.dir != null) j.dir = anterior(i, j.dir);  
    else {                i.elemento=j.elemento; j=j.esq; }  
    return j;  
}
```



# Algoritmo em Java - Classe Árvore Binária

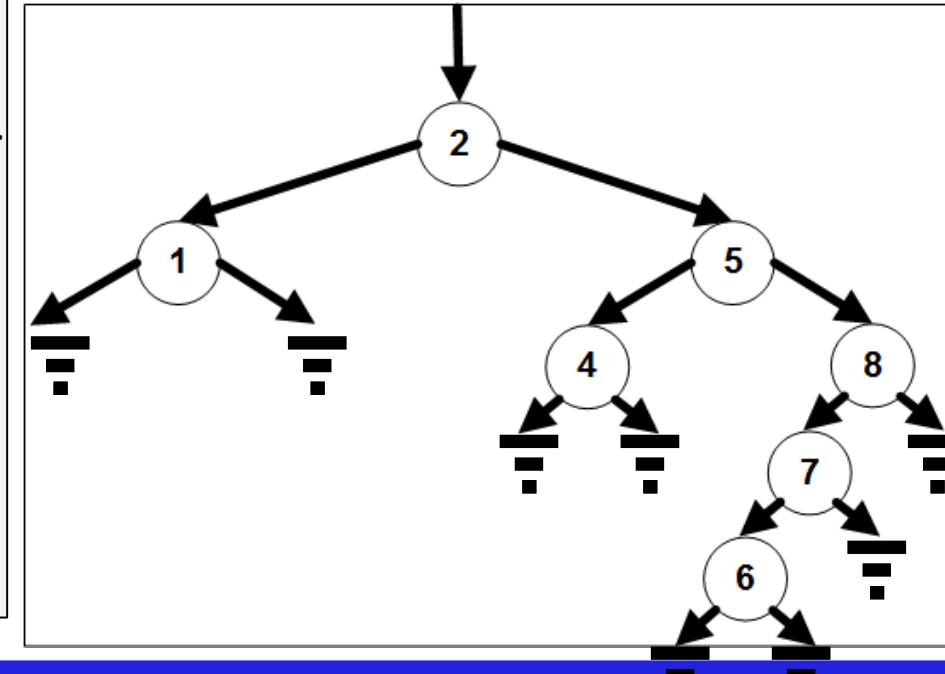
```
//remover(3), dois filhos

public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}

private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz n(3) x 3



# Algoritmo em Java - Classe Árvore Binária

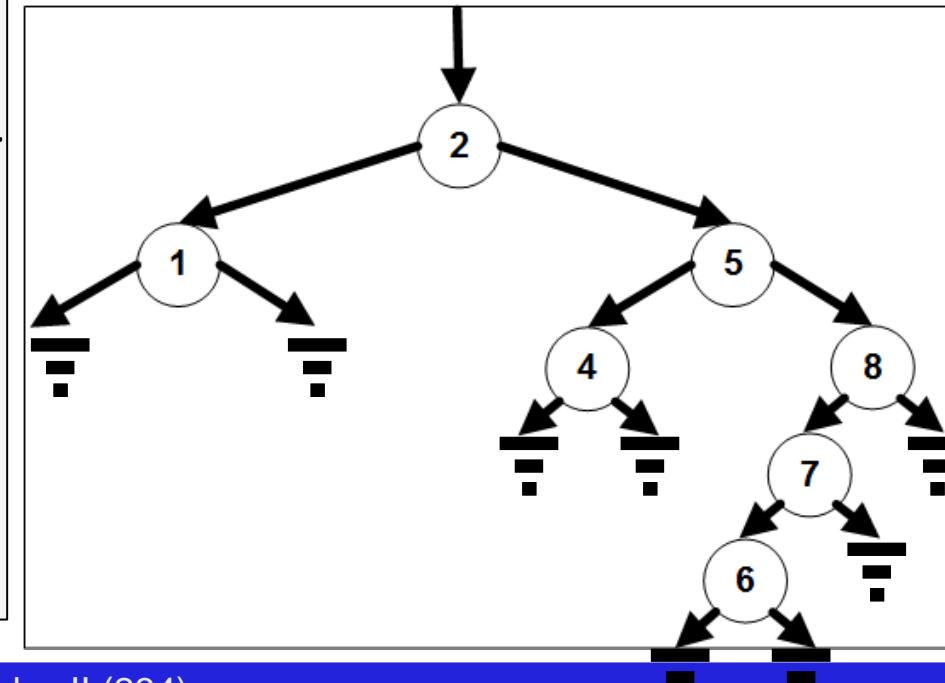
```
//remover(3), dois filhos
```

```
public void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

```
private No remover(int x, No i) throws Exception {
    if (i == null) { throw new Exception("Erro!"); }
    } else if(x < i.elemento) { i.esq=remover(x, i.esq); }
    } else if(x > i.elemento) { i.dir = remover(x, i.dir); }
    } else if(i.dir == null) { i = i.esq; }
    } else if(i.esq == null) { i = i.dir; }
    } else {
        i.esq=anterior(i, i.esq); }
    return i;
}
```

```
private No anterior(No i, No j) {
    if (j.dir != null) j.dir = anterior(i, j.dir);
    } else {
        i.elemento=j.elemento; j=j.esq; }
    return j;
}
```

raiz  
n(3)



# Algoritmo em Java - Classe Árvore Binária

```
public class ArvoreBinaria {  
    private No raiz;  
    public ArvoreBinaria() { raiz = null; }  
    public void inserir(int x) throws Exception { }  
    public boolean pesquisar(int x) { }  
    public void remover(int x) throws Exception { }  
    public void mostrarCentral() { }  
    public void mostrarPre() { }  
    public void mostrarPos() { }  
}
```

Cada um dos métodos de  
Mostrar mostra todos os  
elementos da árvore em  
uma determinada ordem

# Algoritmo em Java - Classe Árvore Binária

```
public void mostrarCentral() {  
    mostrarCentral(raiz);  
}
```

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

```
public void mostrarPos() {  
    mostrarPos(raiz);  
}
```

```
private void mostrarPos(No i) {  
    if (i != null) {  
        mostrarPos(i.esq);  
        mostrarPos(i.dir);  
        System.out.print(i.elemento + " ");  
    }  
}
```

```
public void mostrarPre() {  
    mostrarPre(raiz);  
}
```

```
private void mostrarPre(No i) {  
    if (i != null) {  
        System.out.print(i.elemento + " ");  
        mostrarPre(i.esq);  
        mostrarPre(i.dir);  
    }  
}
```

## Exercício

- Mostre a ordem em que os elementos da árvore abaixo são visitados pelos métodos de mostrar (a) central, (b) pós-fixado e (c) pré-fixado

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

```
private void mostrarPos(No i) {  
    if (i != null) {  
        mostrarPos(i.esq);  
        mostrarPos(i.dir);  
        System.out.print(i.elemento + " ");  
    }  
}
```

```
private void mostrarPre(No i) {  
    if (i != null) {  
        System.out.print(i.elemento + " ");  
        mostrarPre(i.esq);  
        mostrarPre(i.dir);  
    }  
}
```

## Exercício

- Mostre a ordem em que os elementos da árvore abaixo são visitados pelos métodos de mostrar (a) central, (b) pós-fixado e (c) pré-fixado

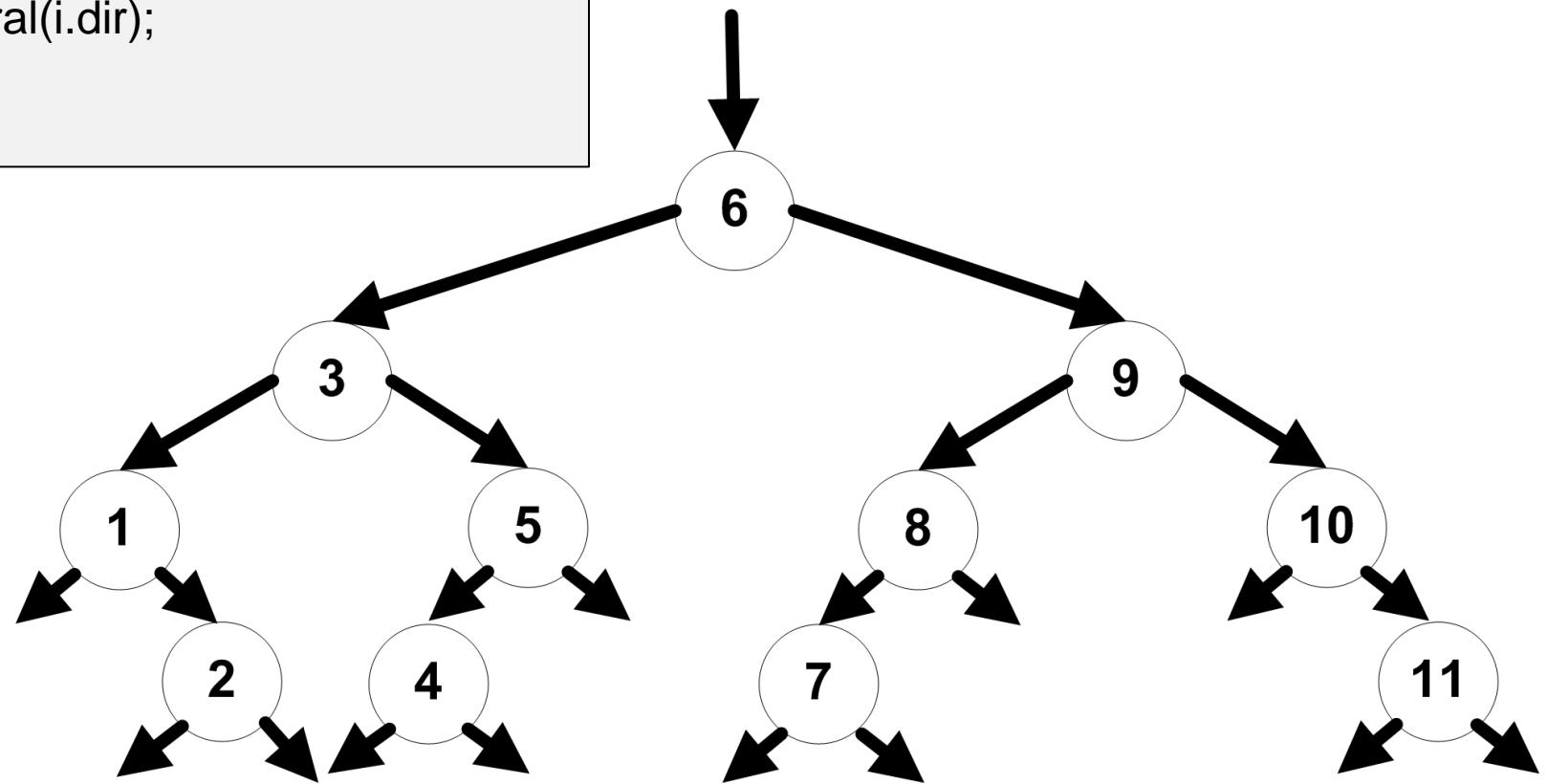
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

```
private void mostrarPos(No i) {  
    if (i != null) {  
        mostrarPos(i.esq);  
        mostrarPos(i.dir);  
        System.out.print(i.elemento + " ");  
    }  
}
```

```
private void mostrarPre(No i) {  
    if (i != null) {  
        System.out.print(i.elemento + " ");  
        mostrarPre(i.esq);  
        mostrarPre(i.dir);  
    }  
}
```

## Exercício

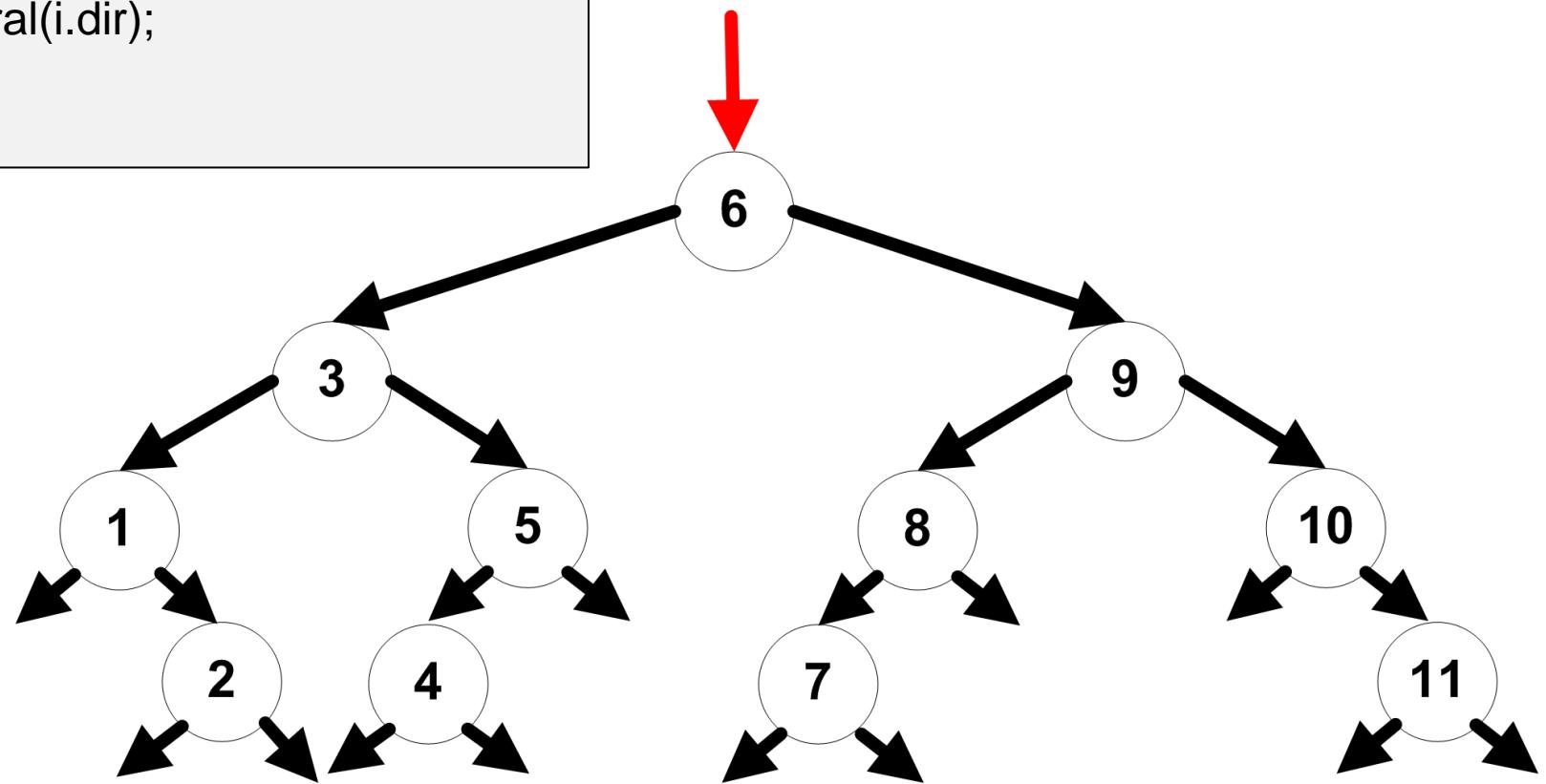
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

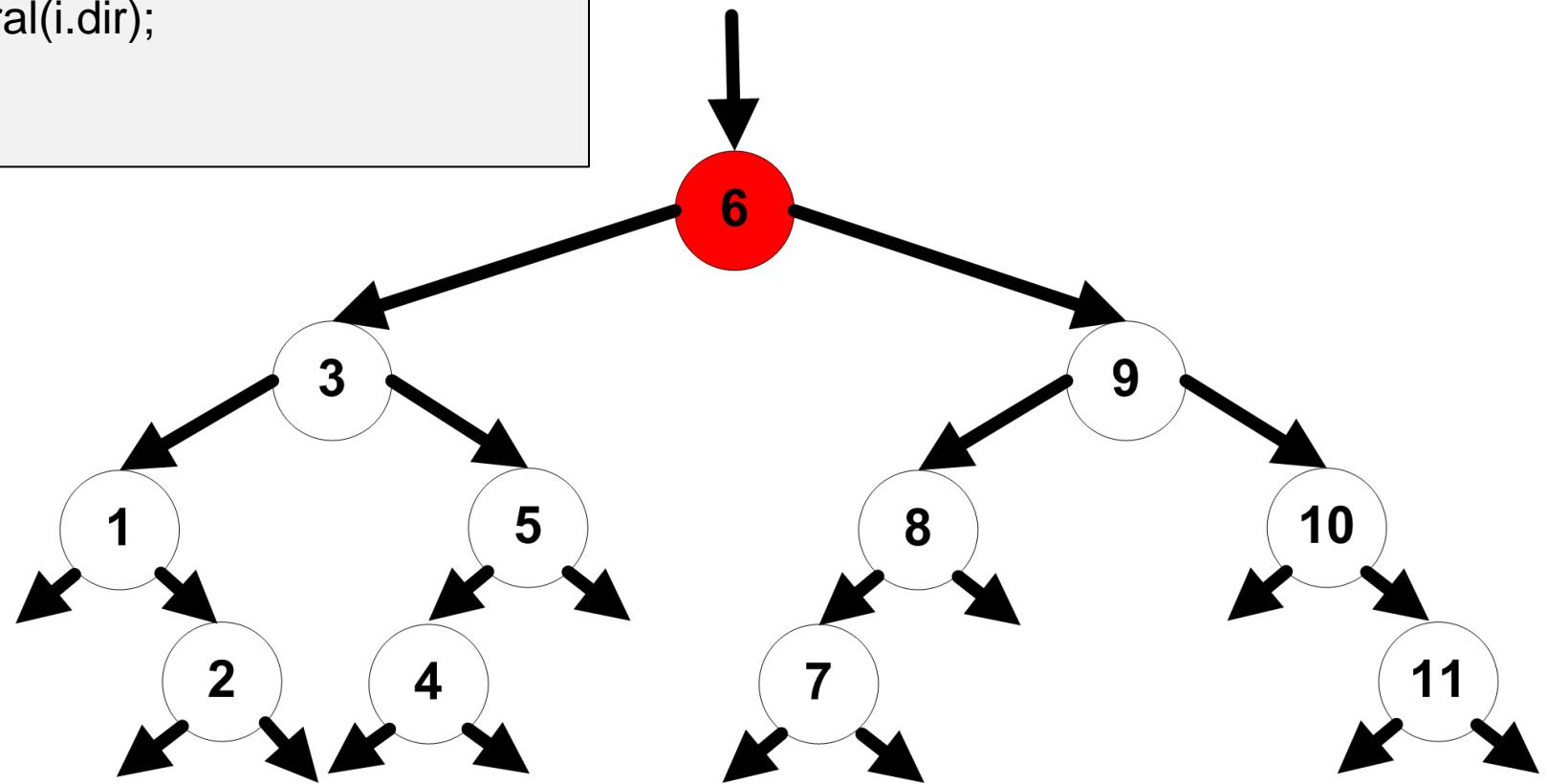
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

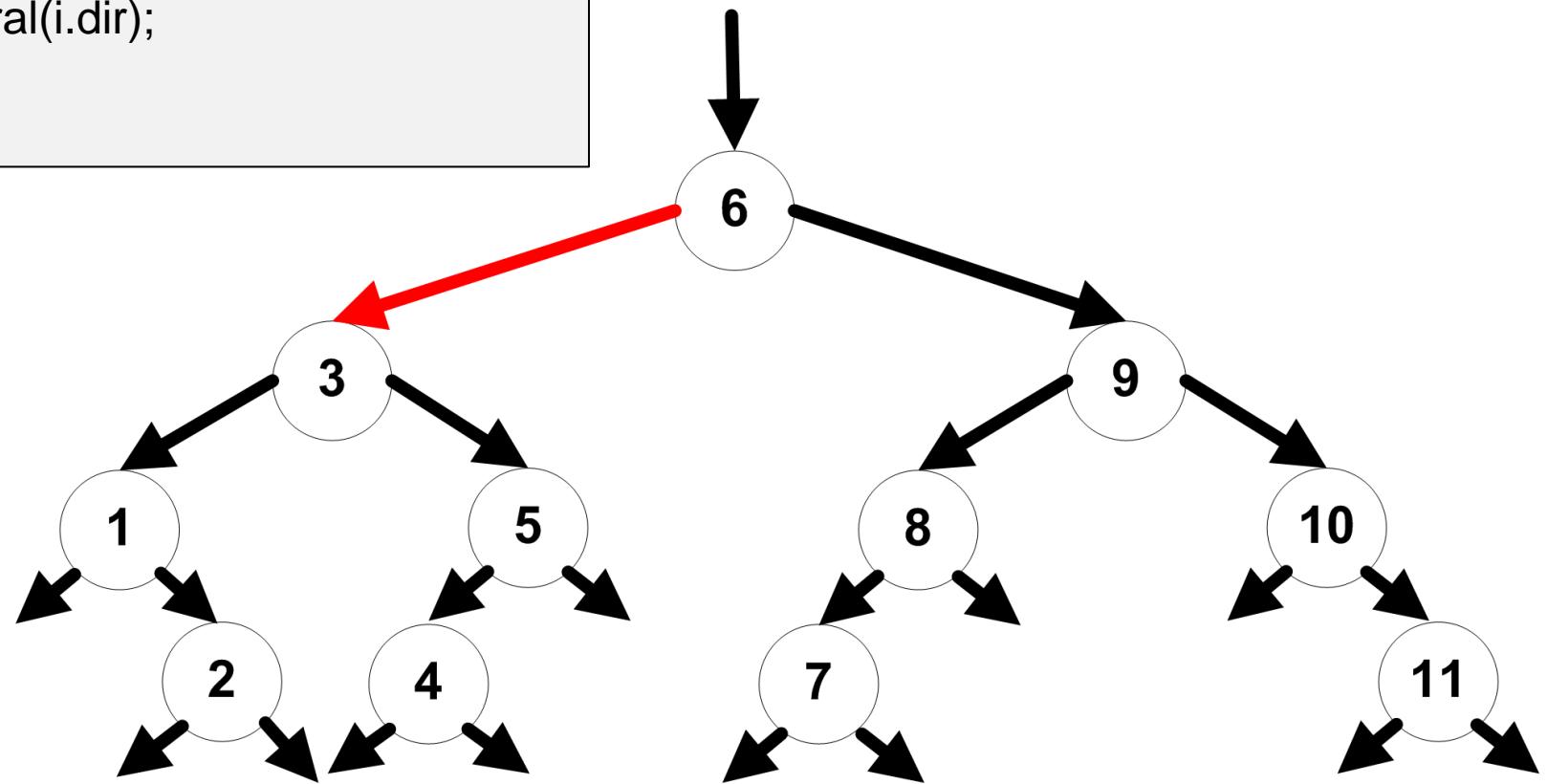
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

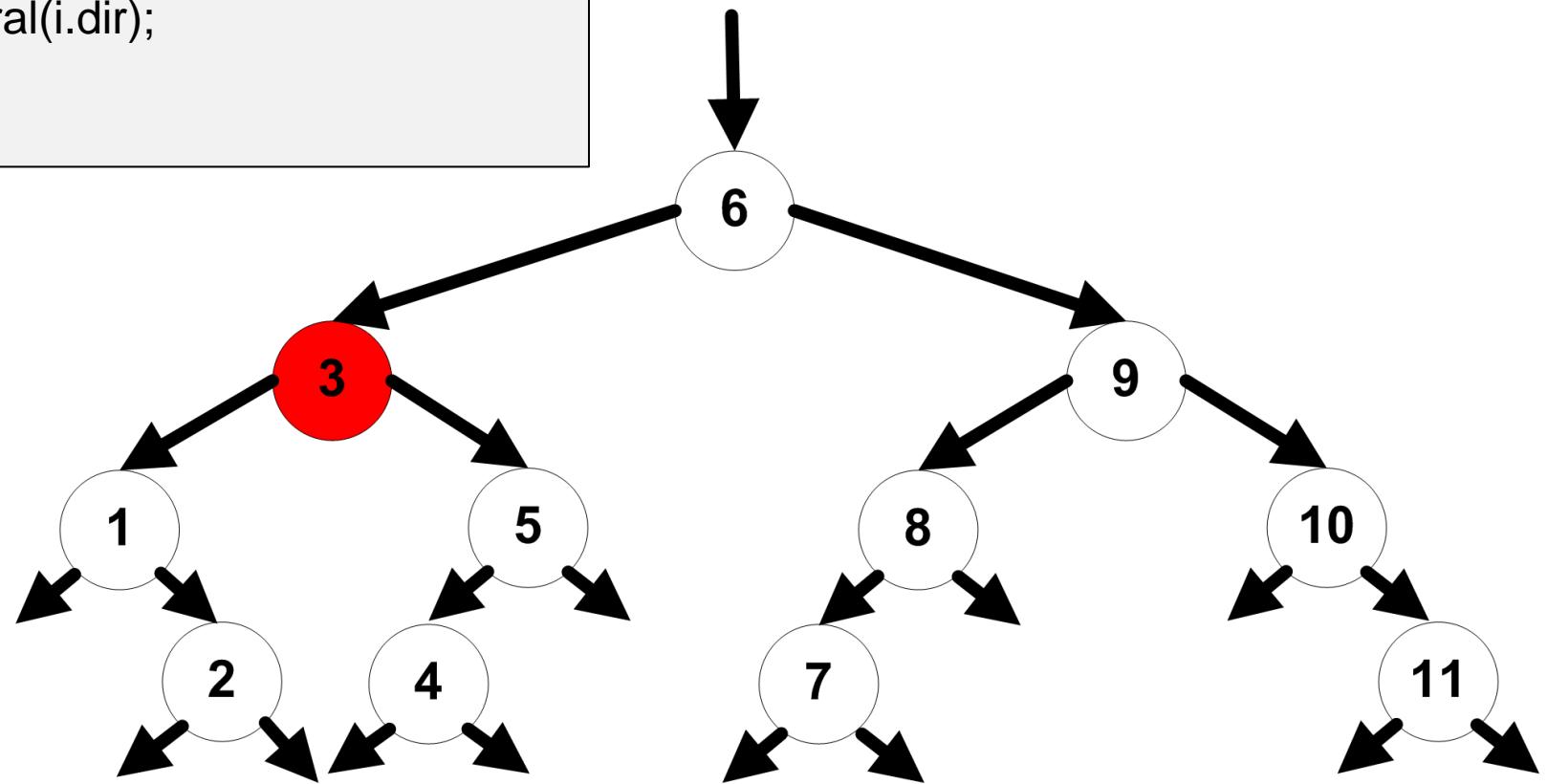
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

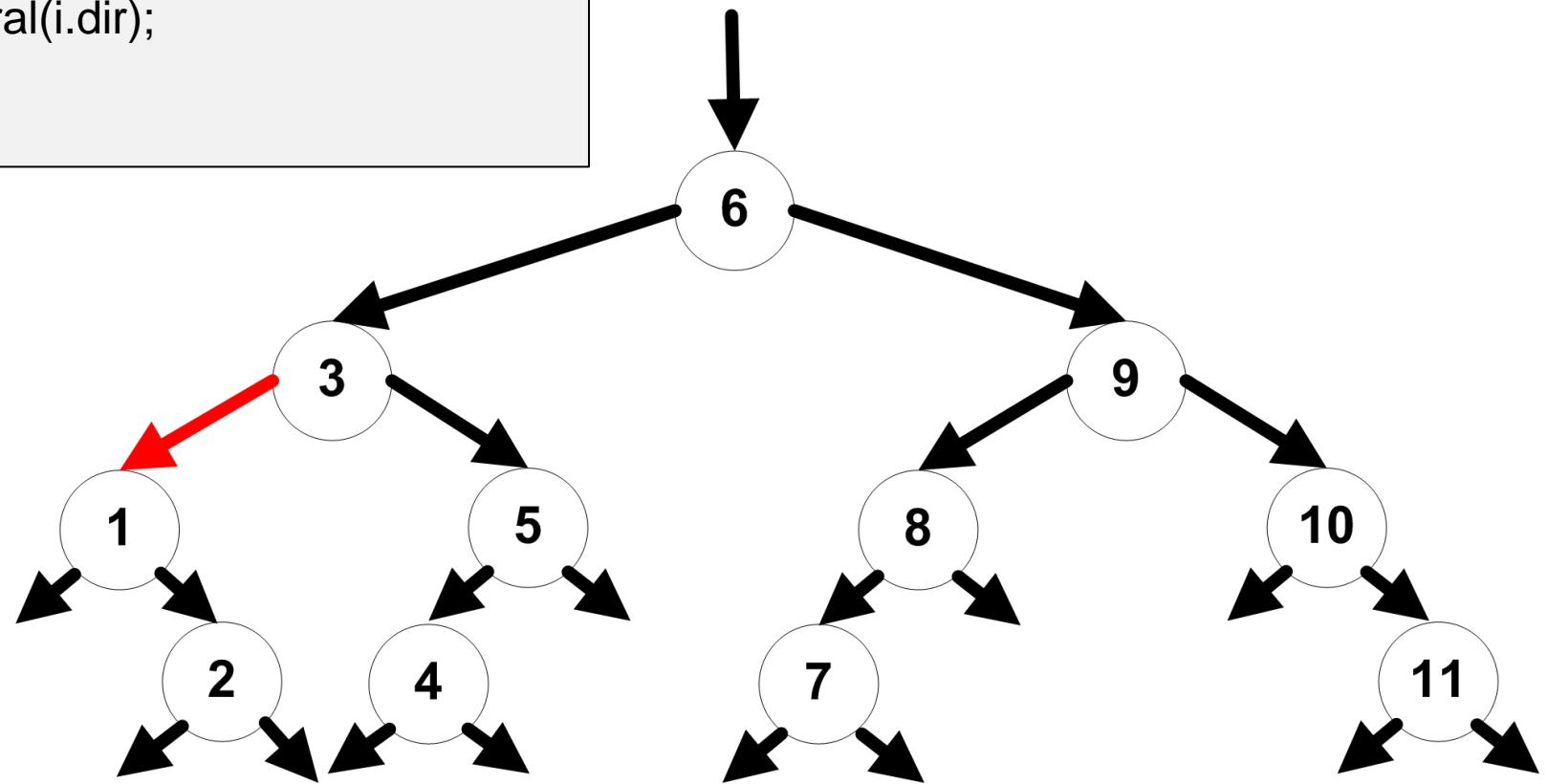
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

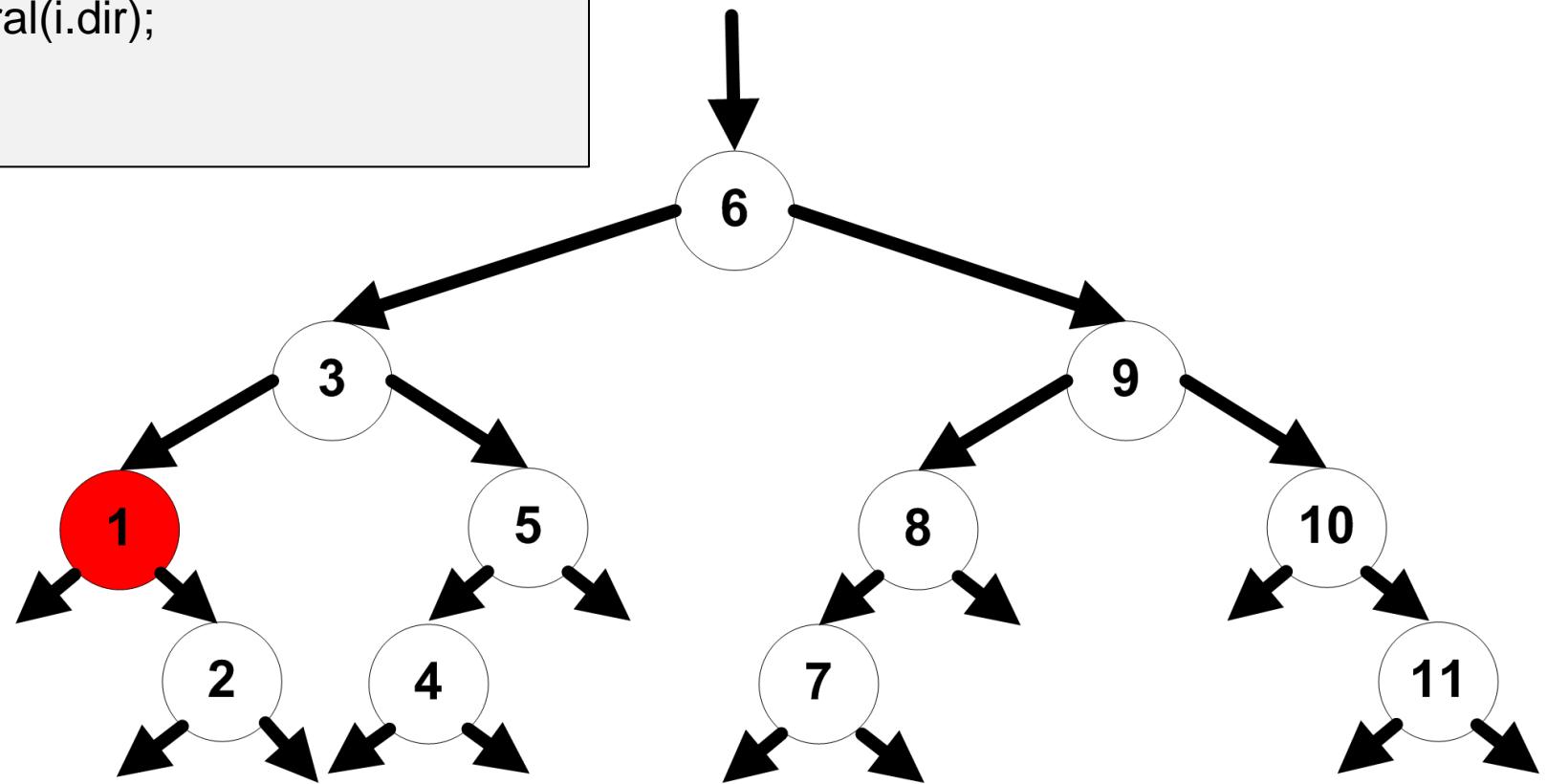
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

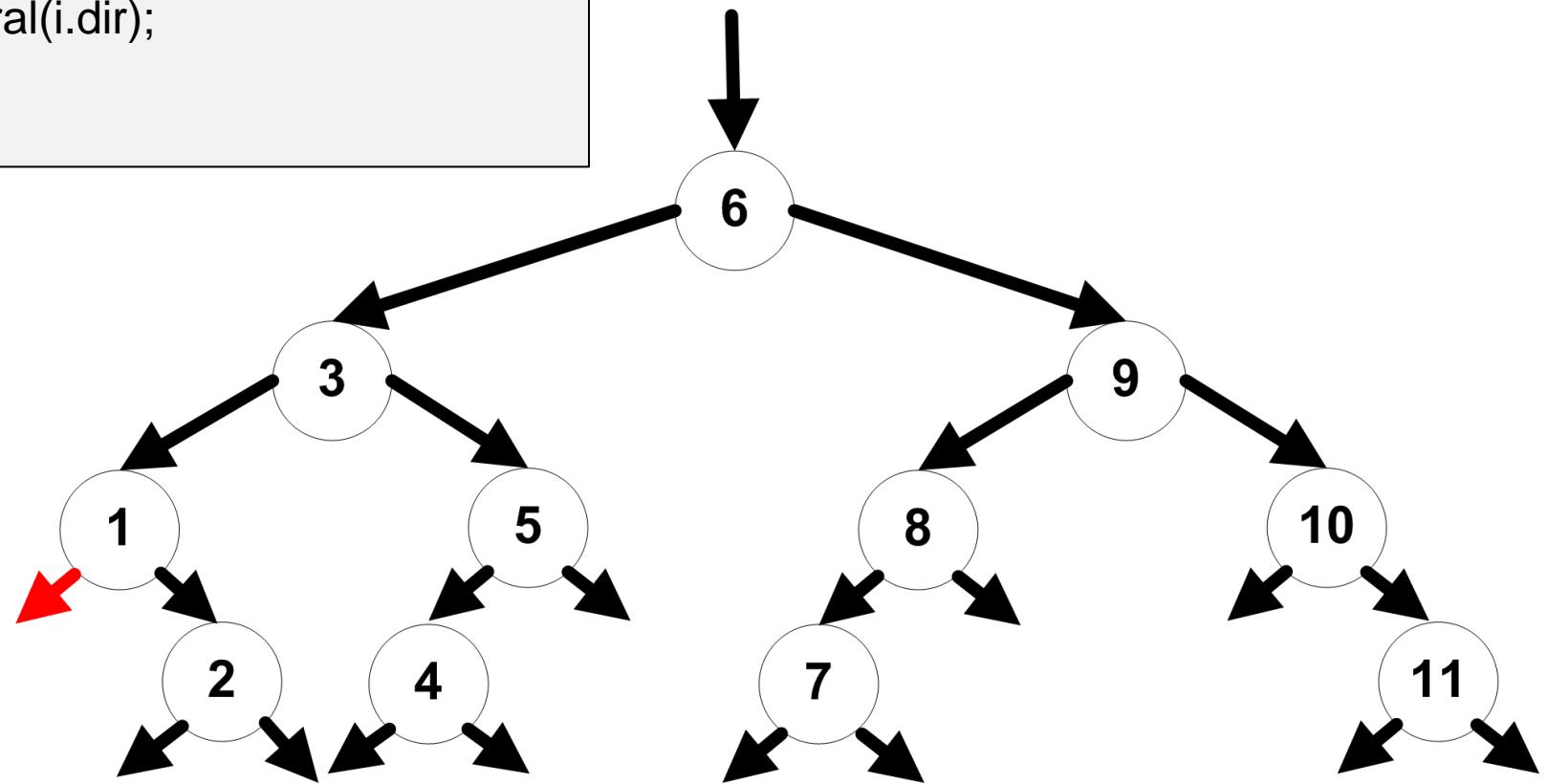
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

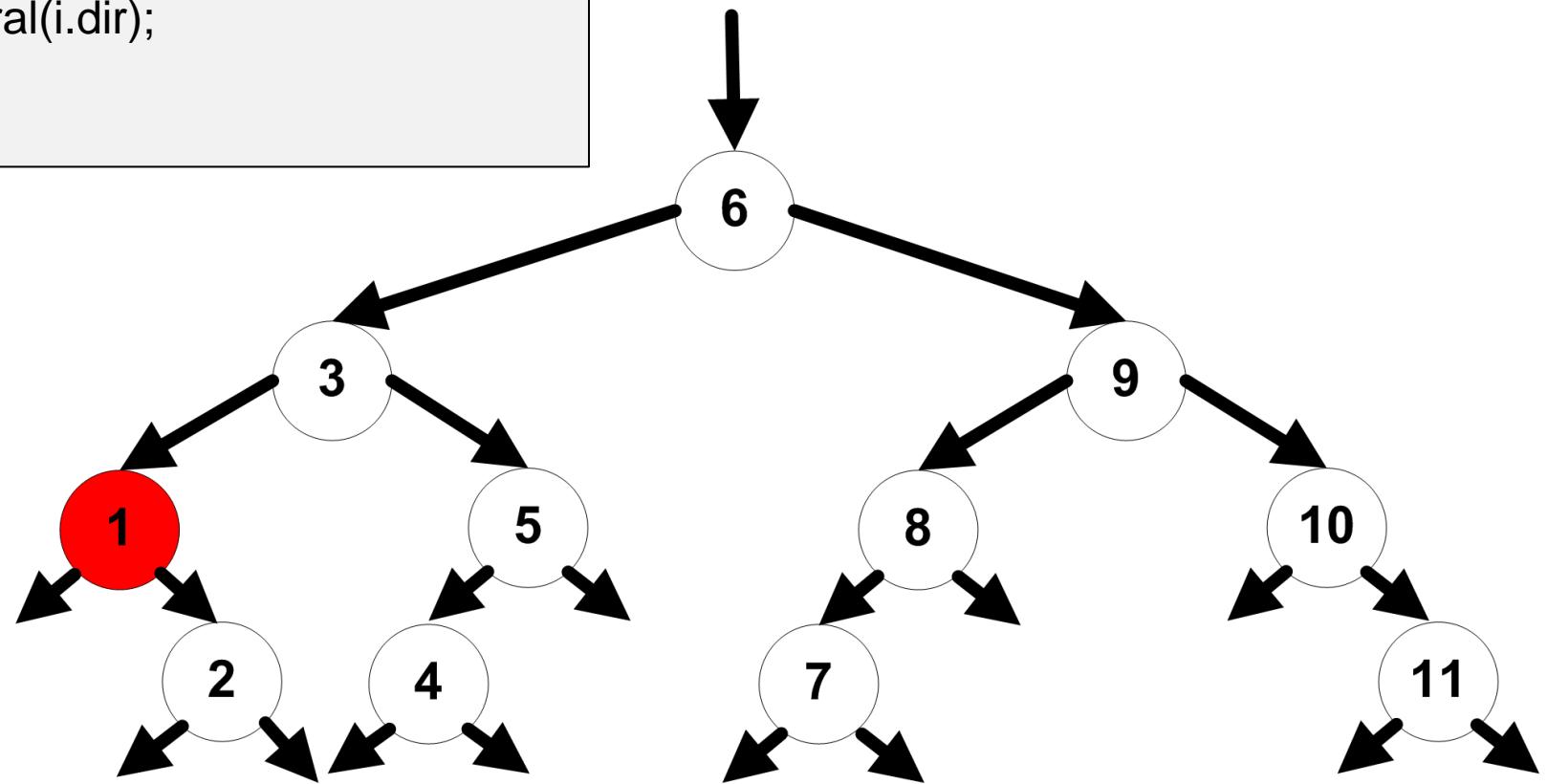
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

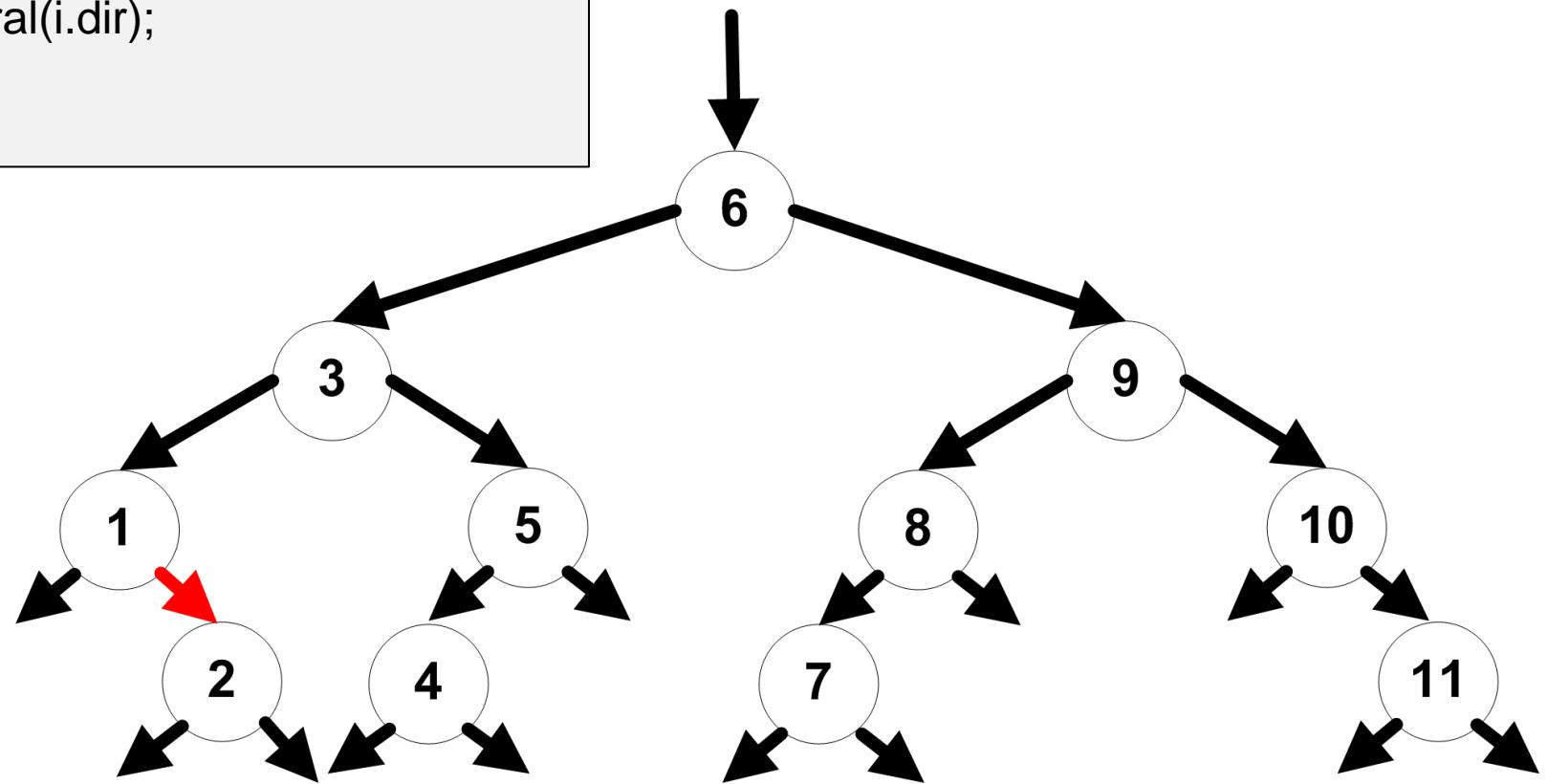


Tela

1

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

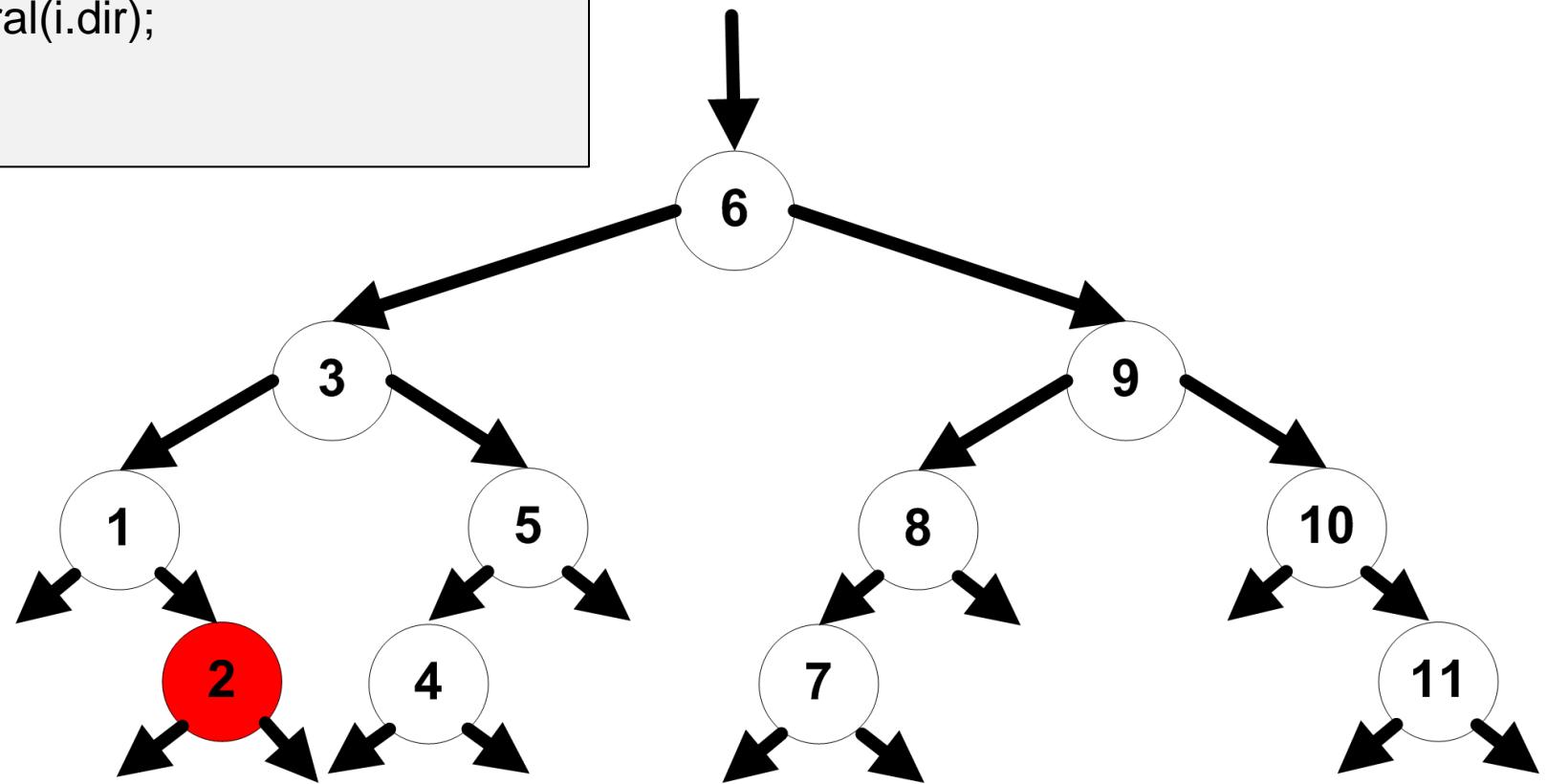


Tela

1

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

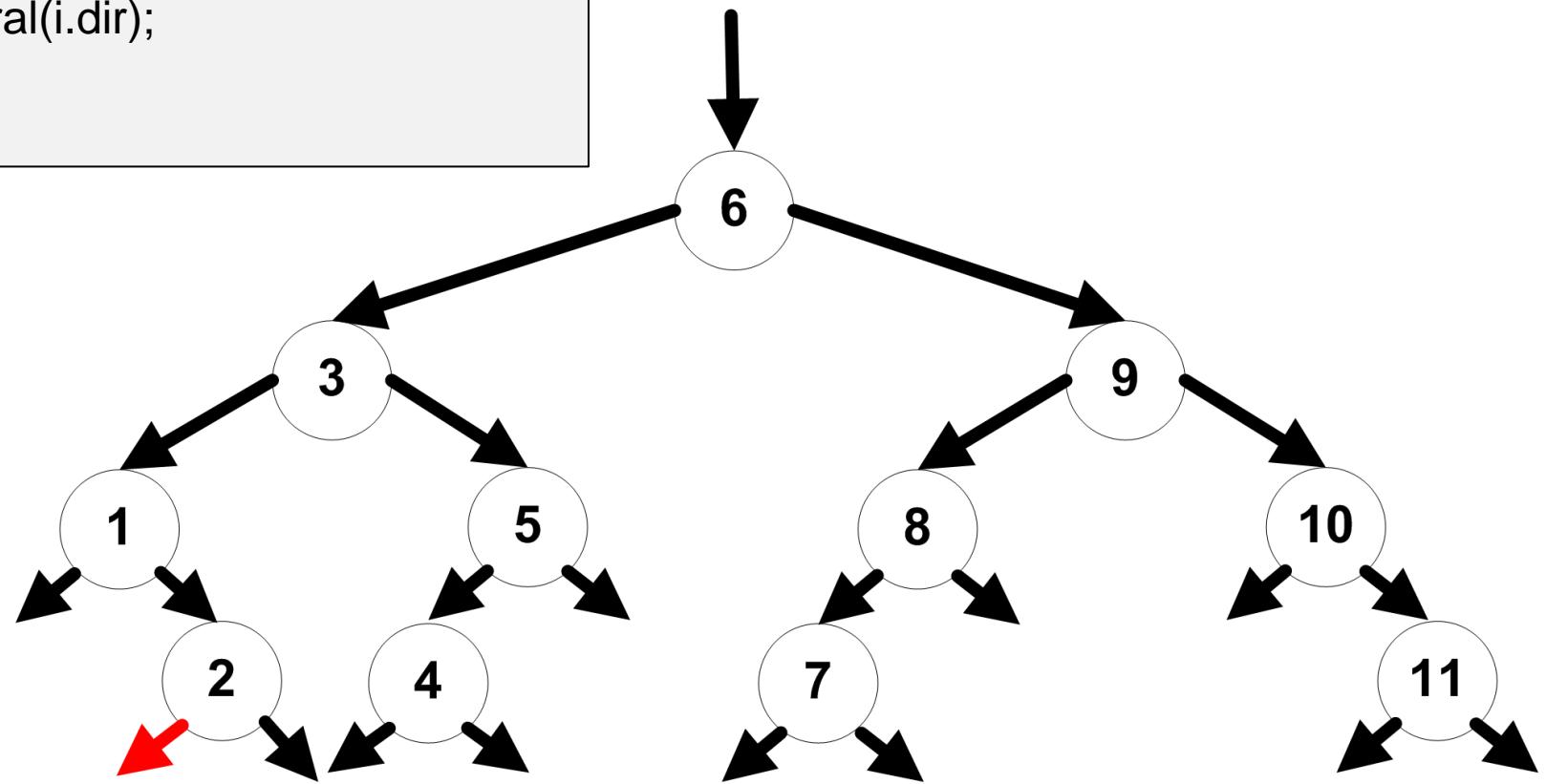


Tela

1

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

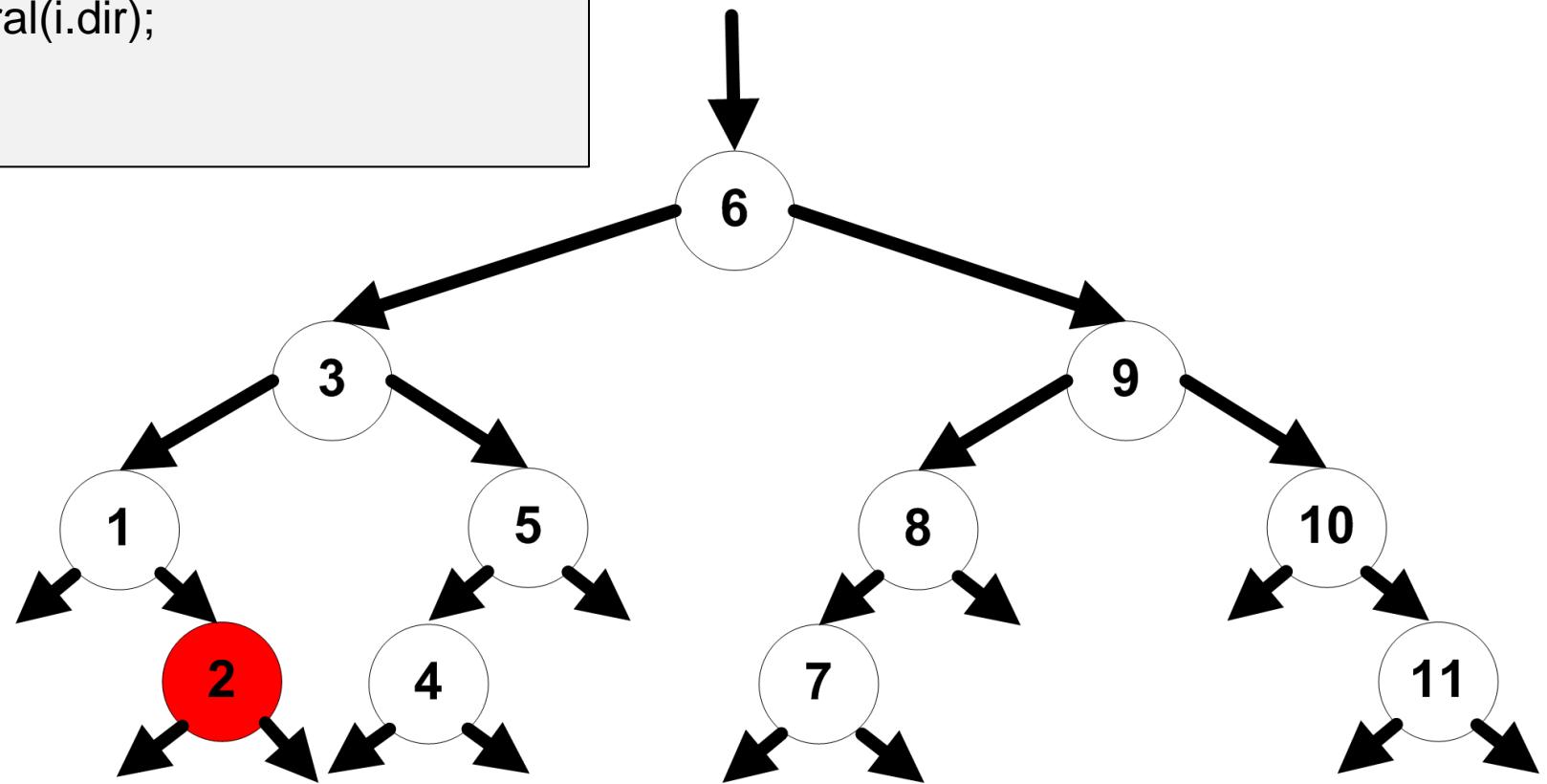


Tela

1

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

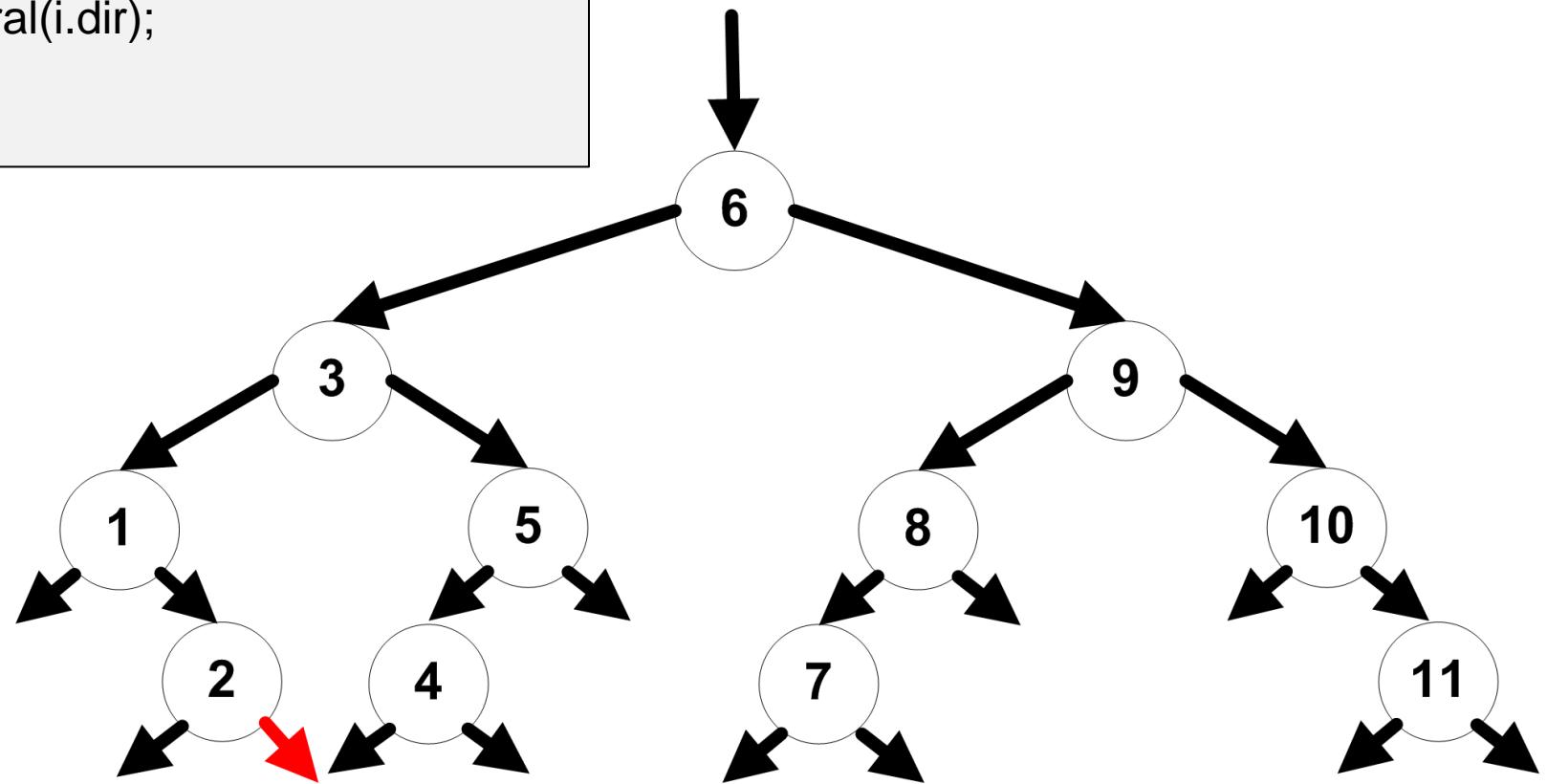


Tela

1 2

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

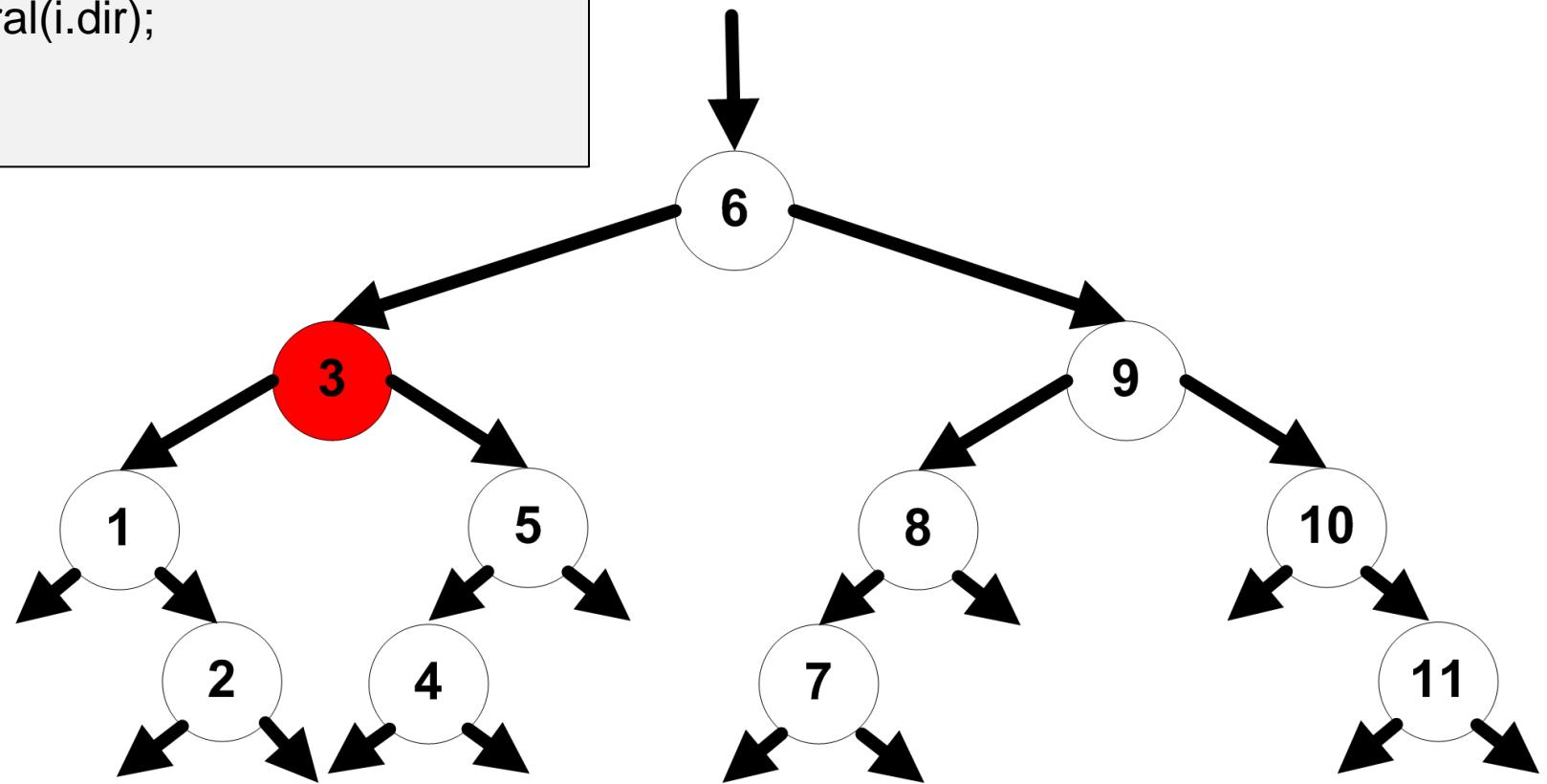


Tela

1 2

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

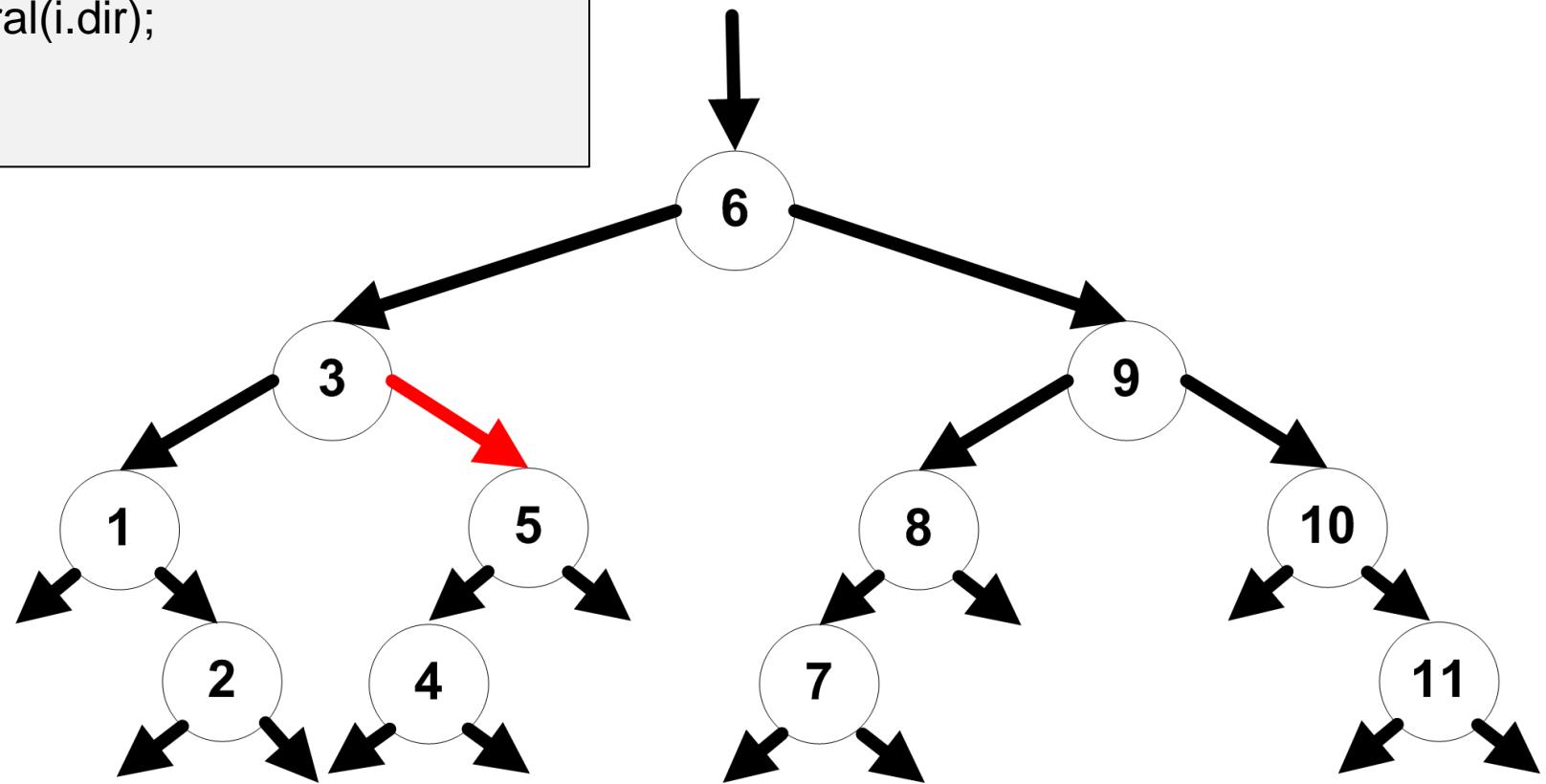


Tela

1 2 3

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

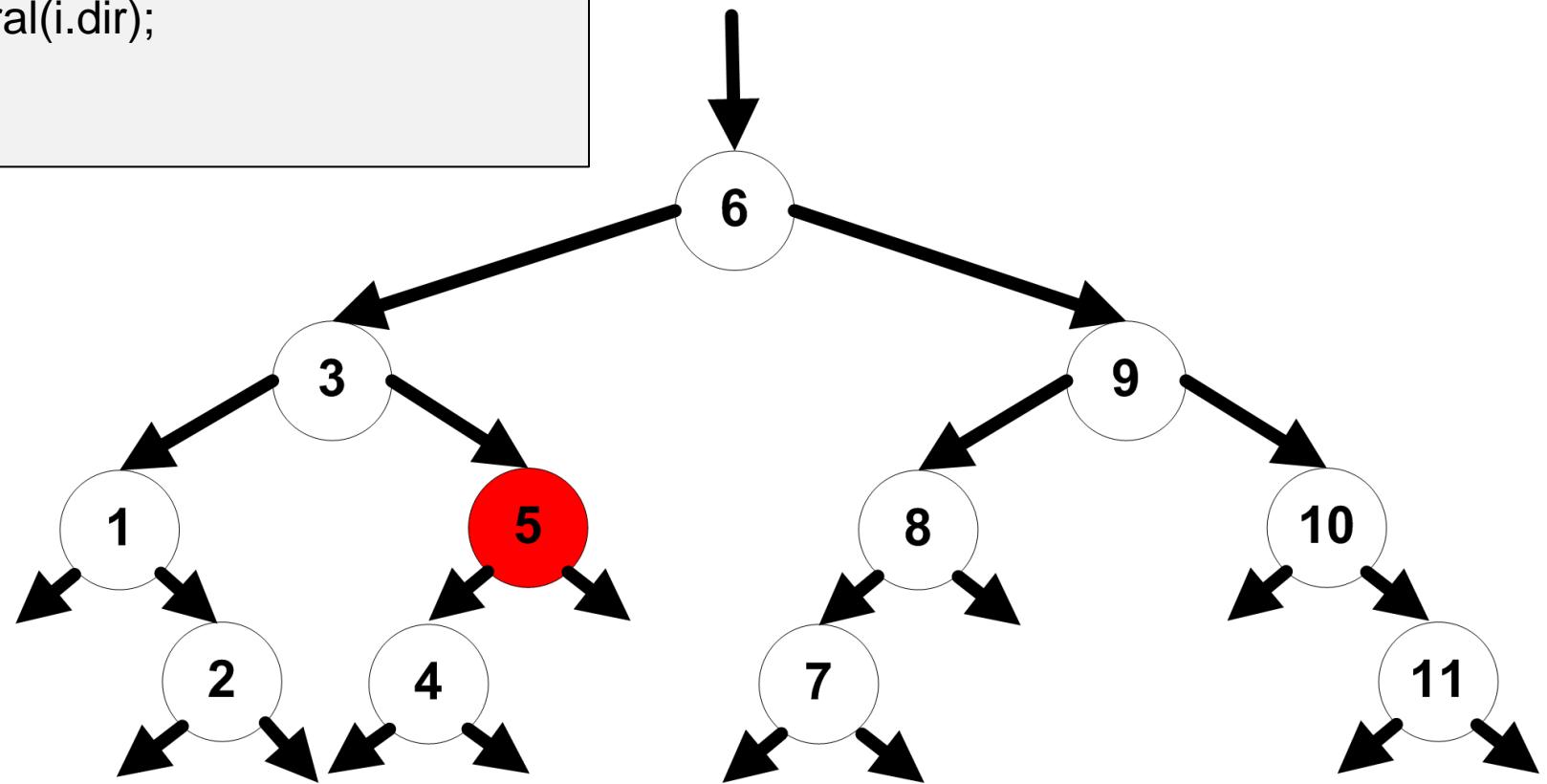


Tela

1 2 3

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

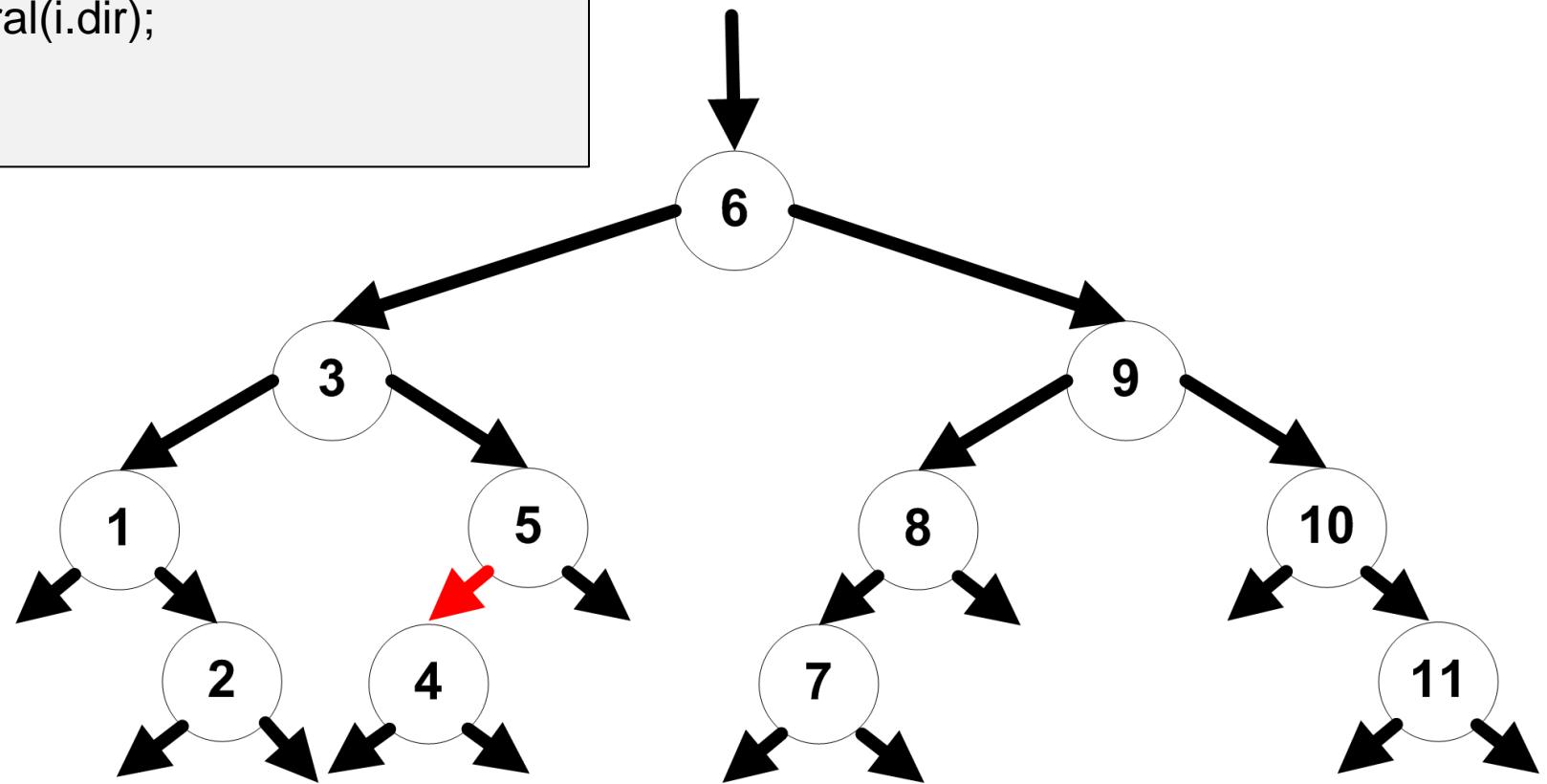


Tela

1 2 3

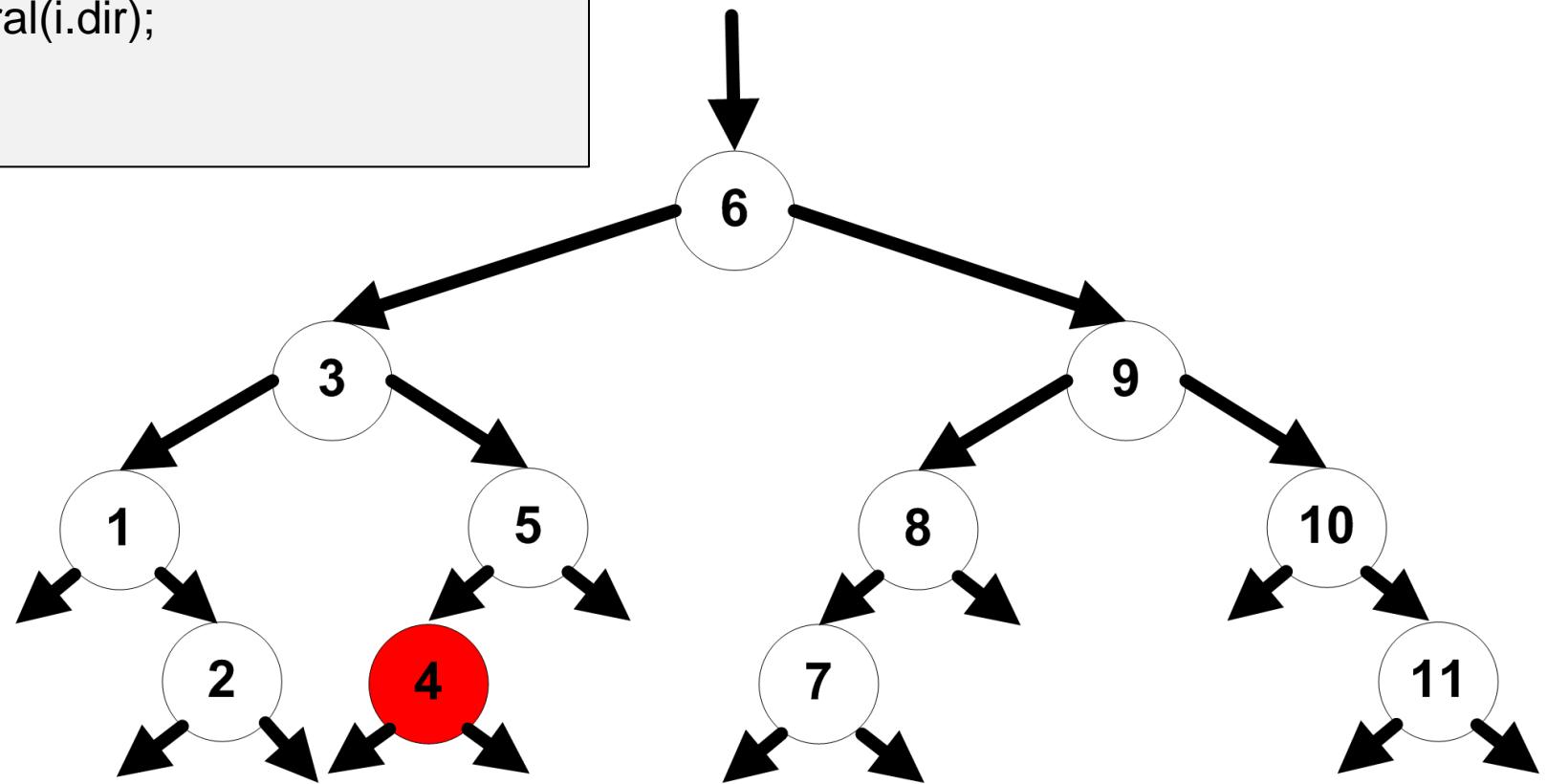
## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

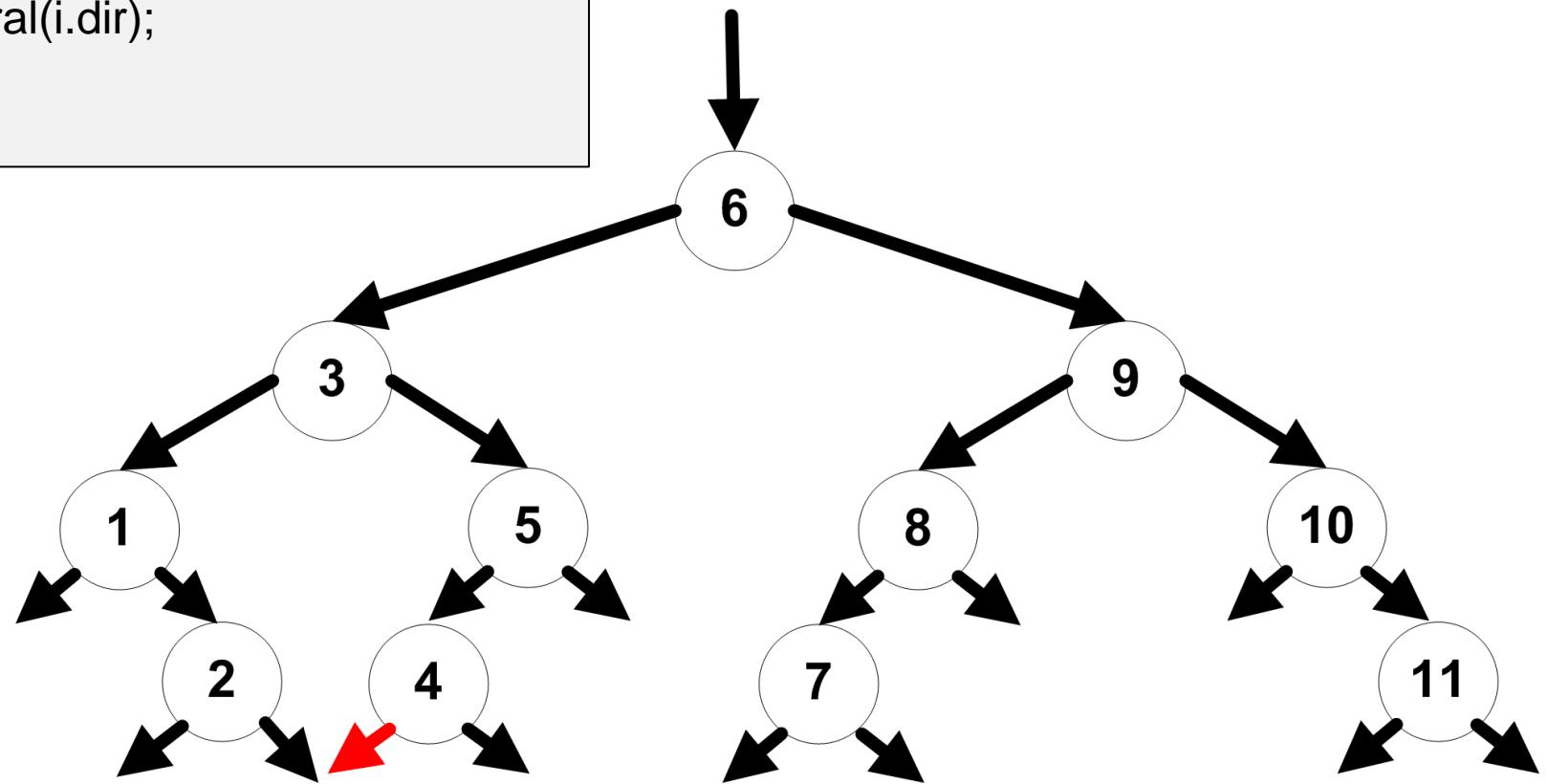


Tela

1 2 3

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

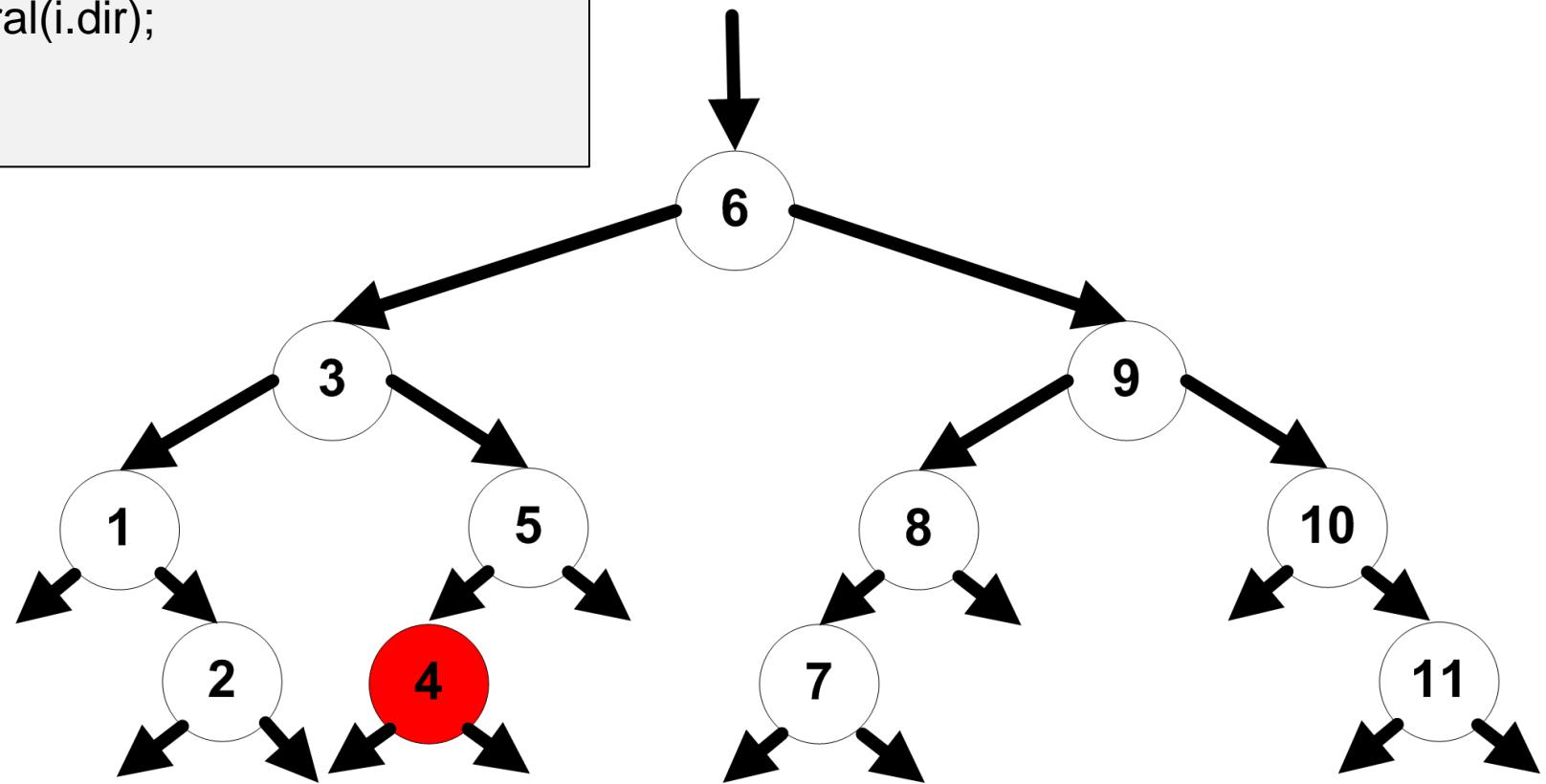


Tela

1 2 3

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

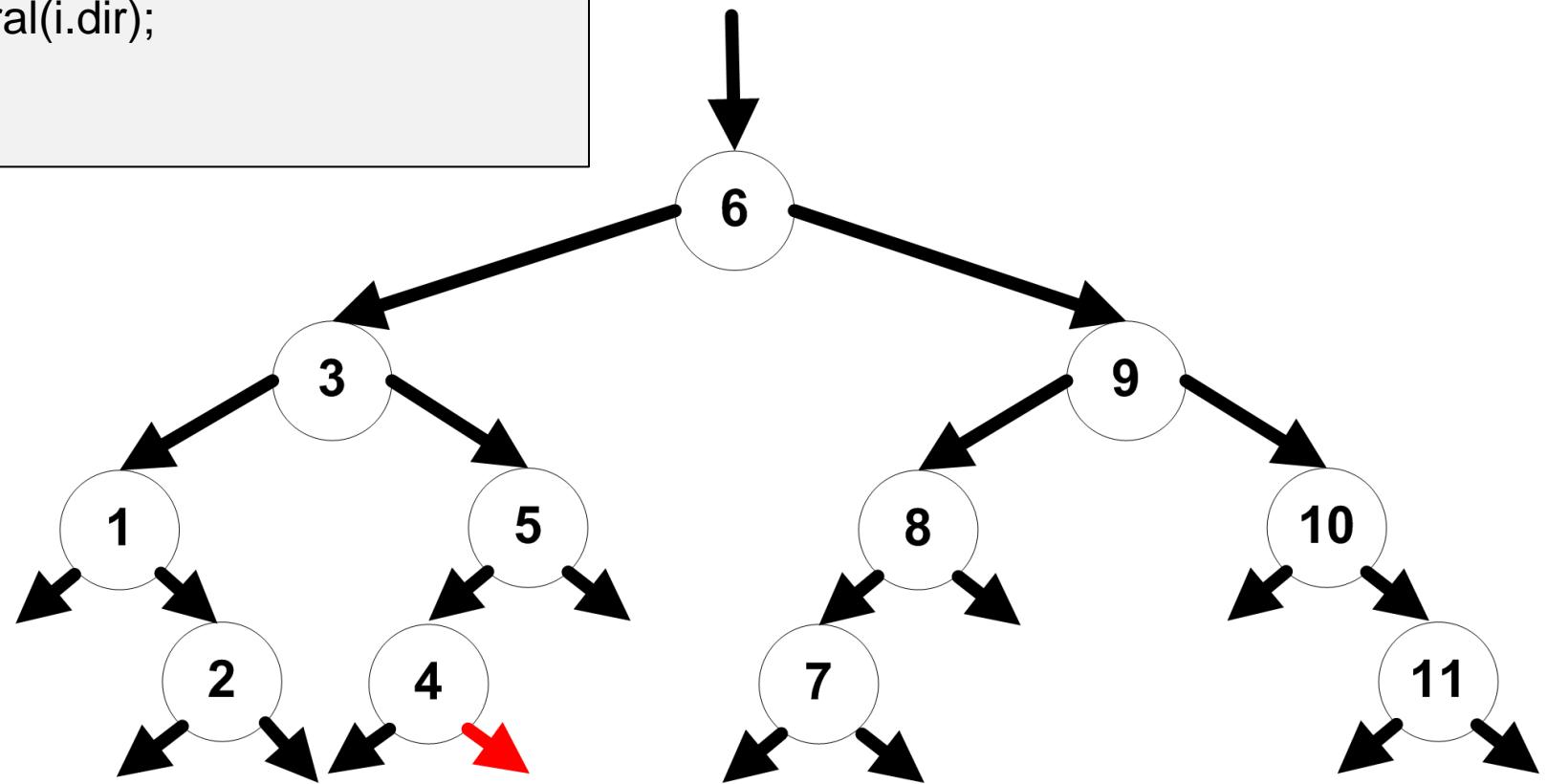


Tela

1 2 3 4

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

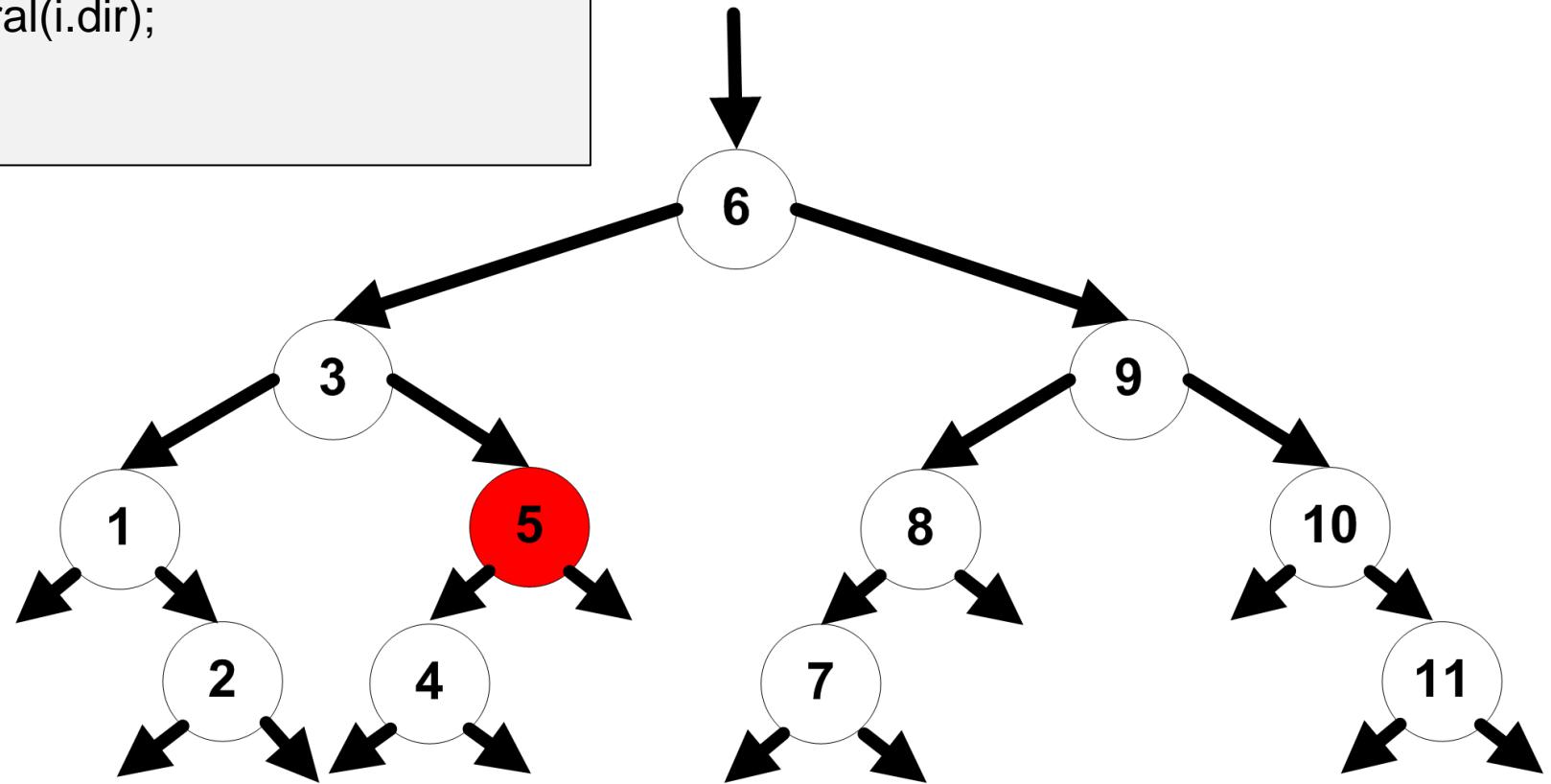


Tela

1 2 3 4

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

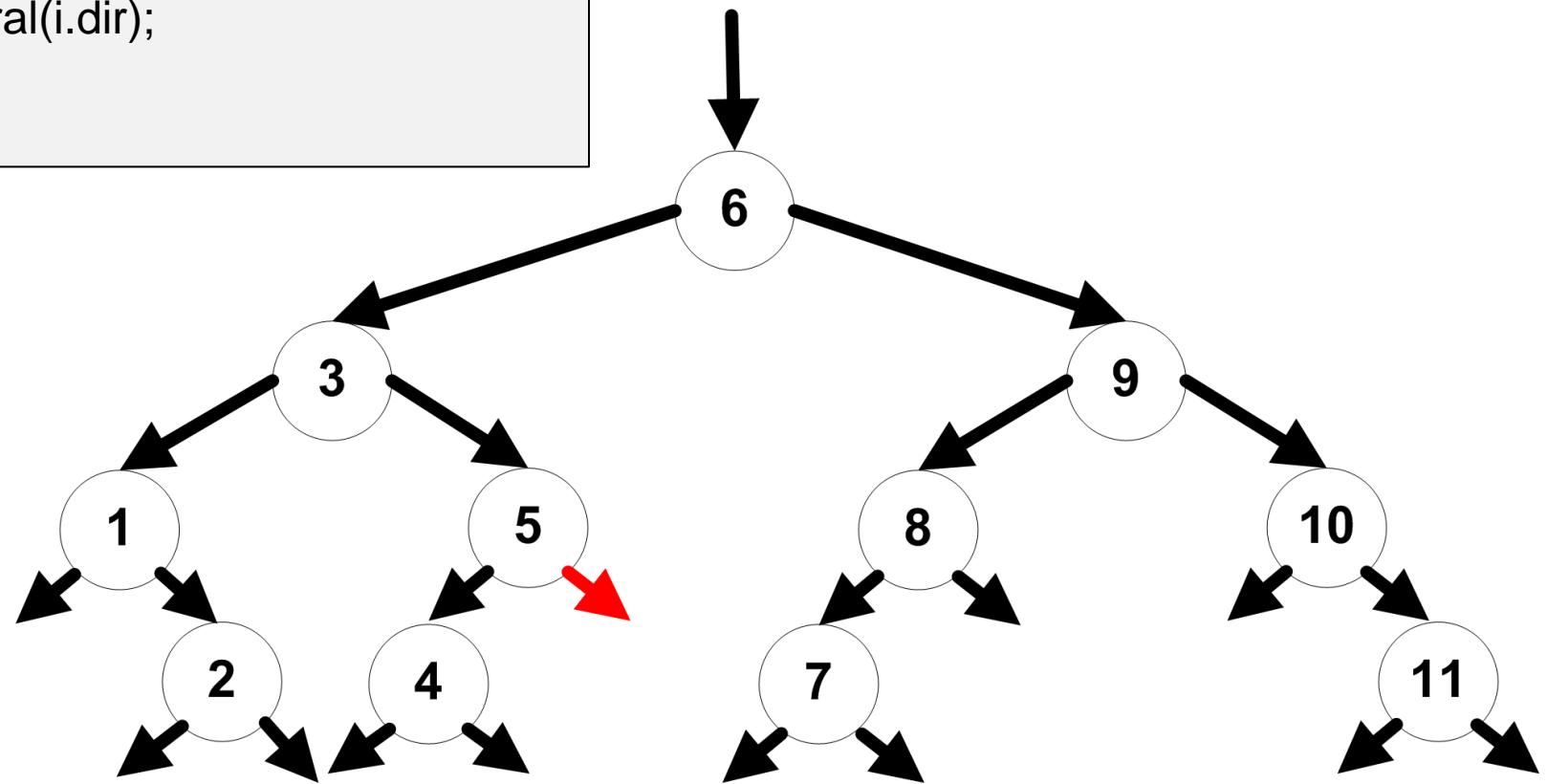


Tela

1 2 3 4 5

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

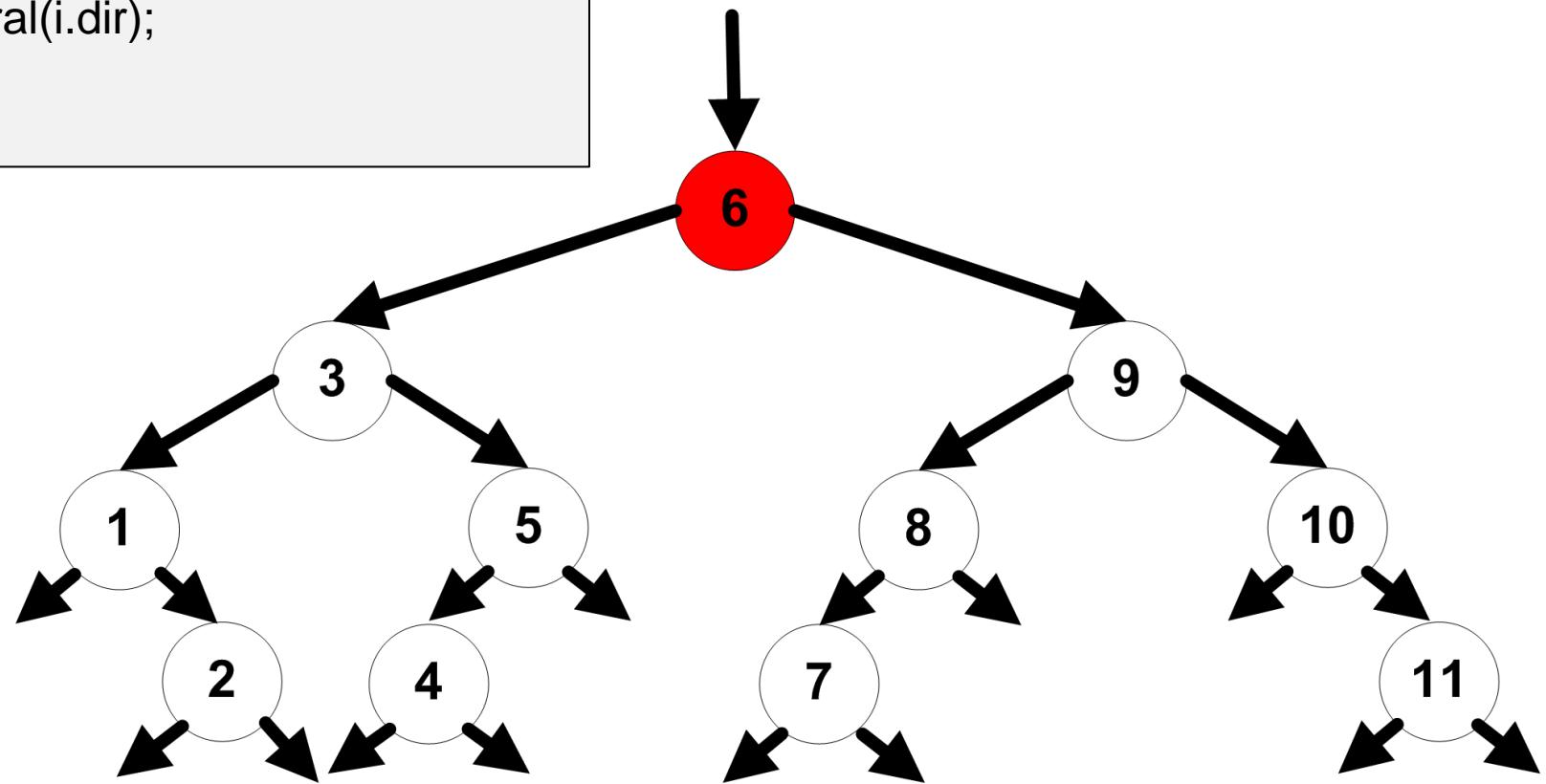


Tela

1 2 3 4 5

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

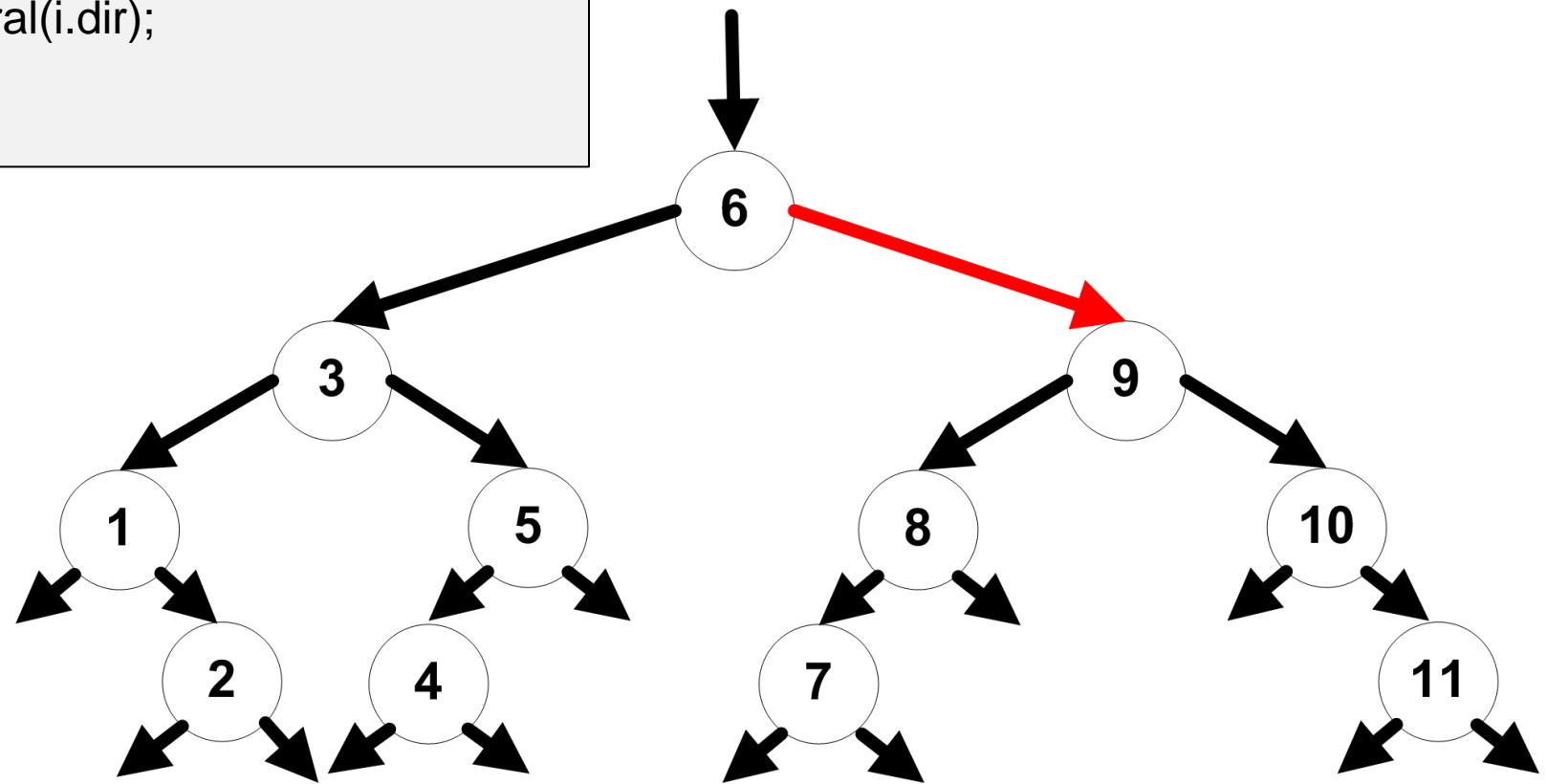


Tela

1 2 3 4 5 6

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela

1 2

3 4

5 6

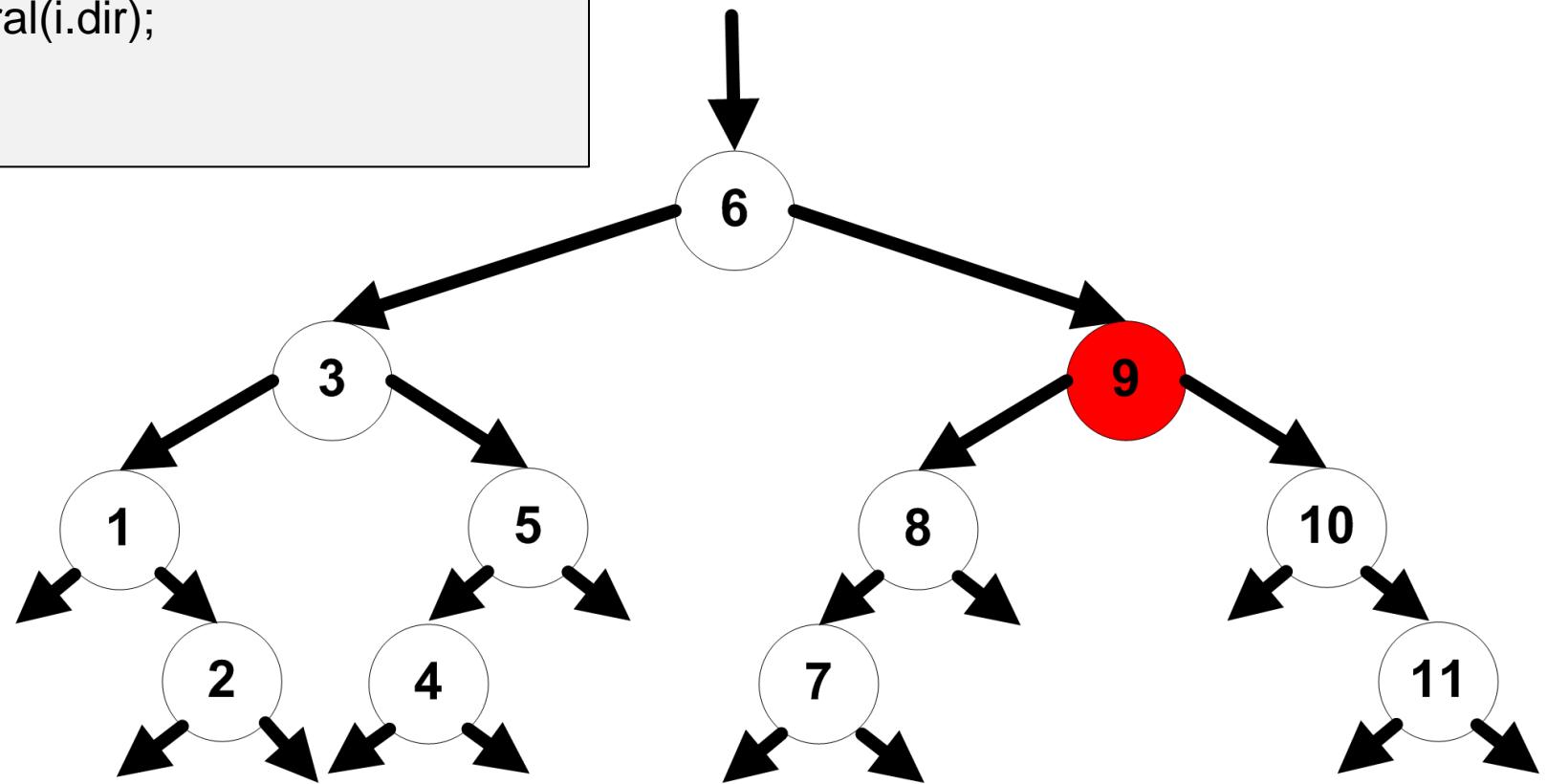
7 8

9 10

11

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

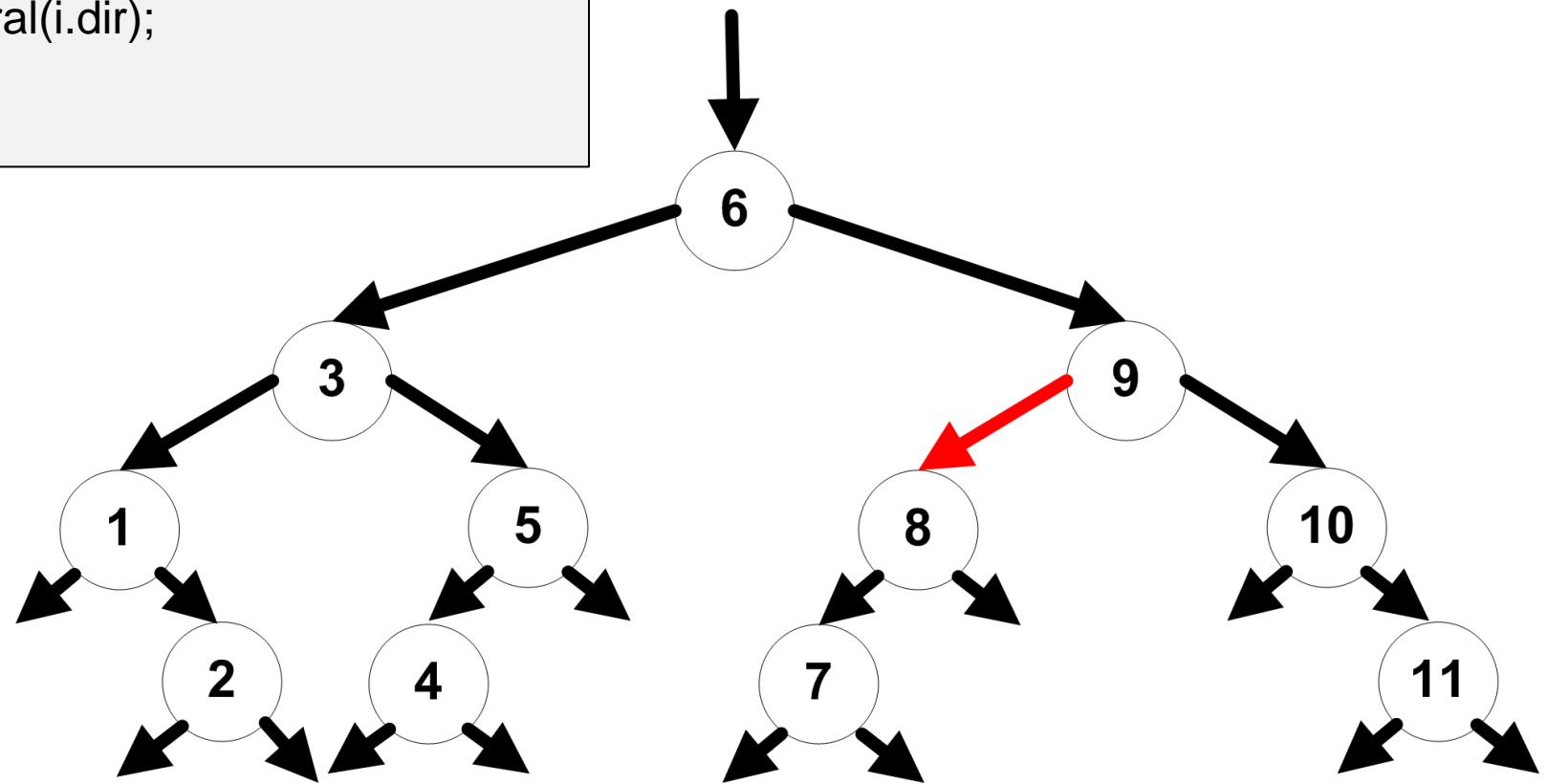


Tela

1 2 3 4 5 6

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

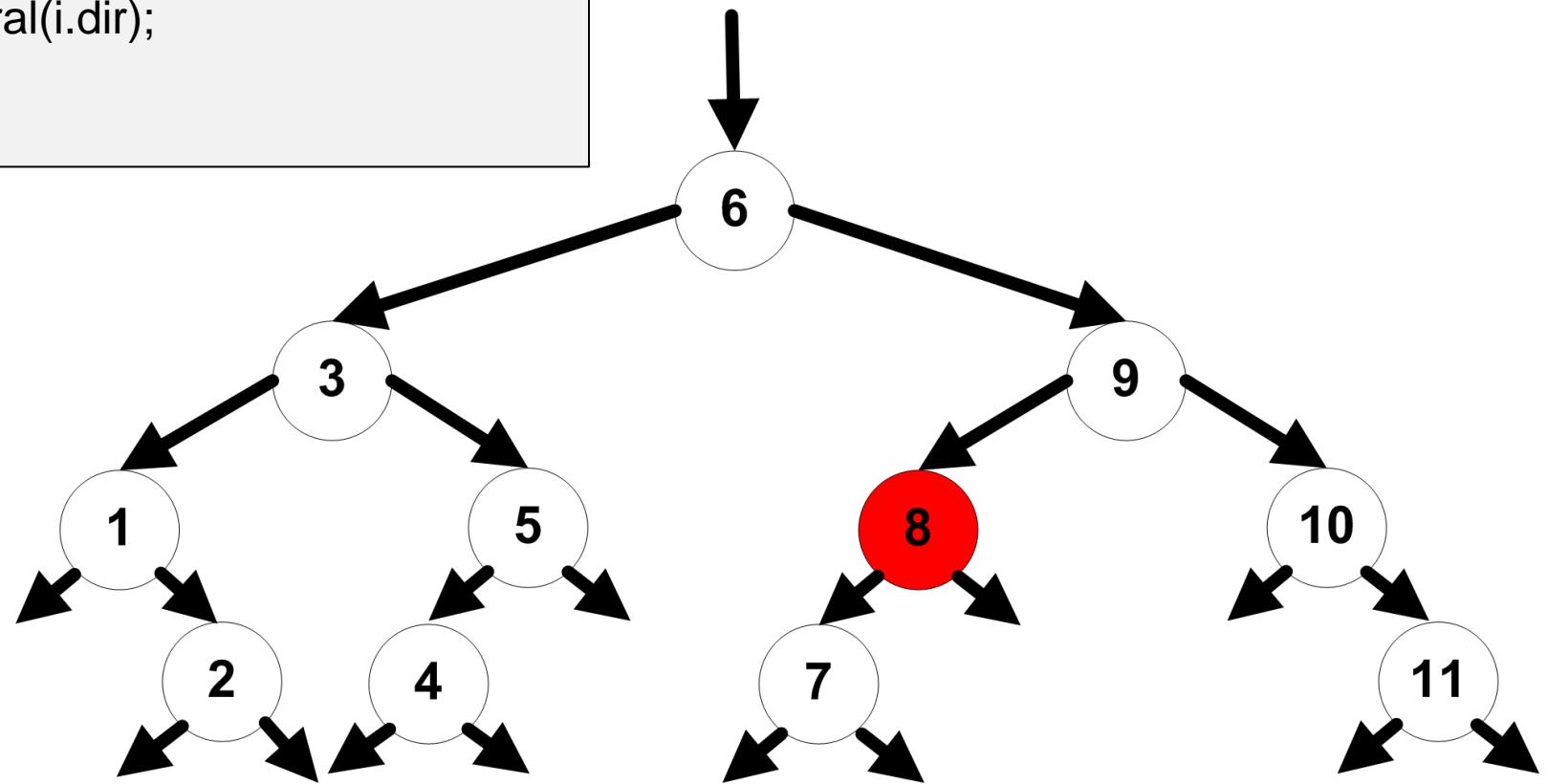


Tela

1 2 3 4 5 6

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

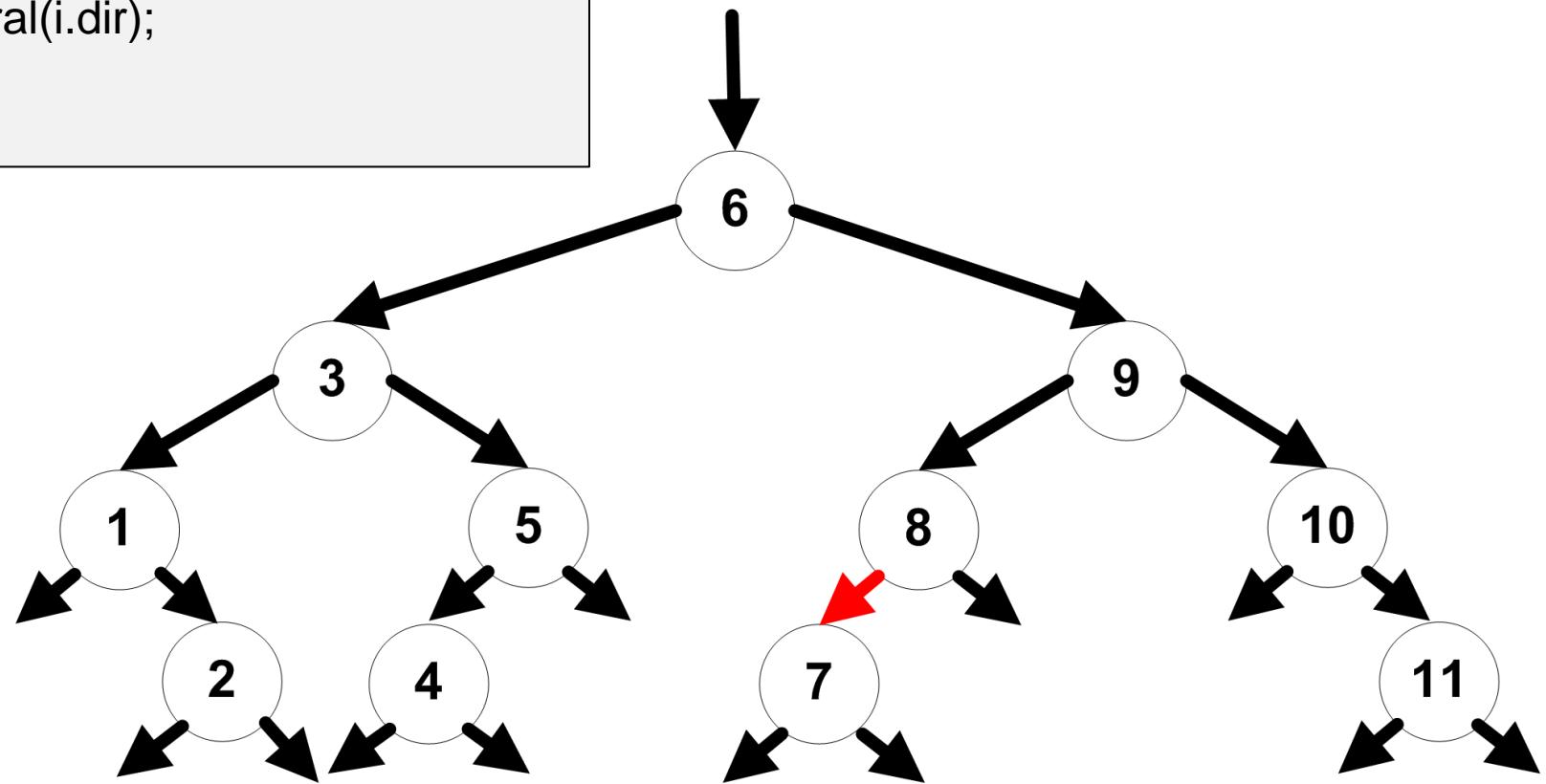


Tela

1 2 3 4 5 6

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

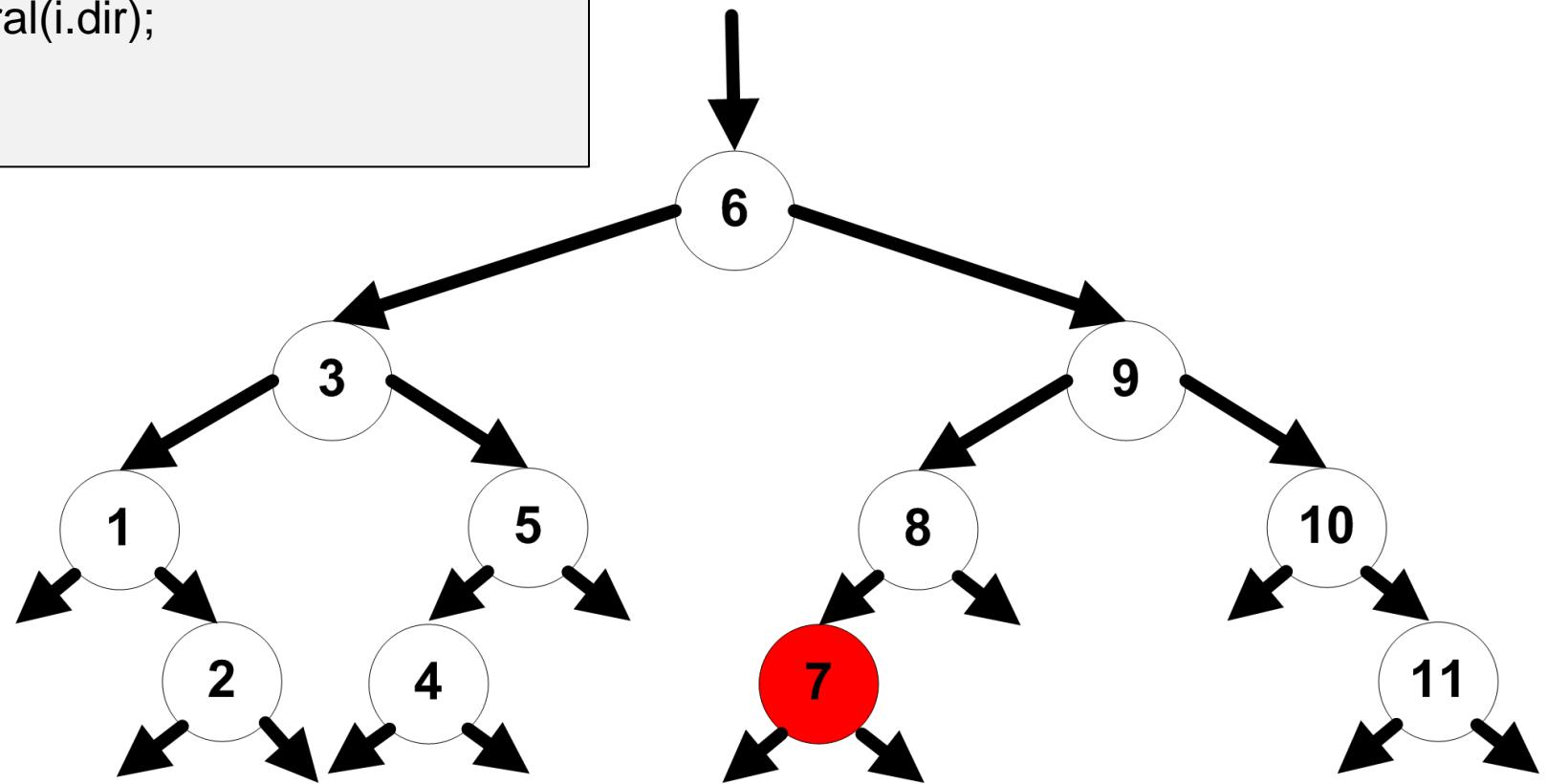


Tela

1 2 3 4 5 6

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

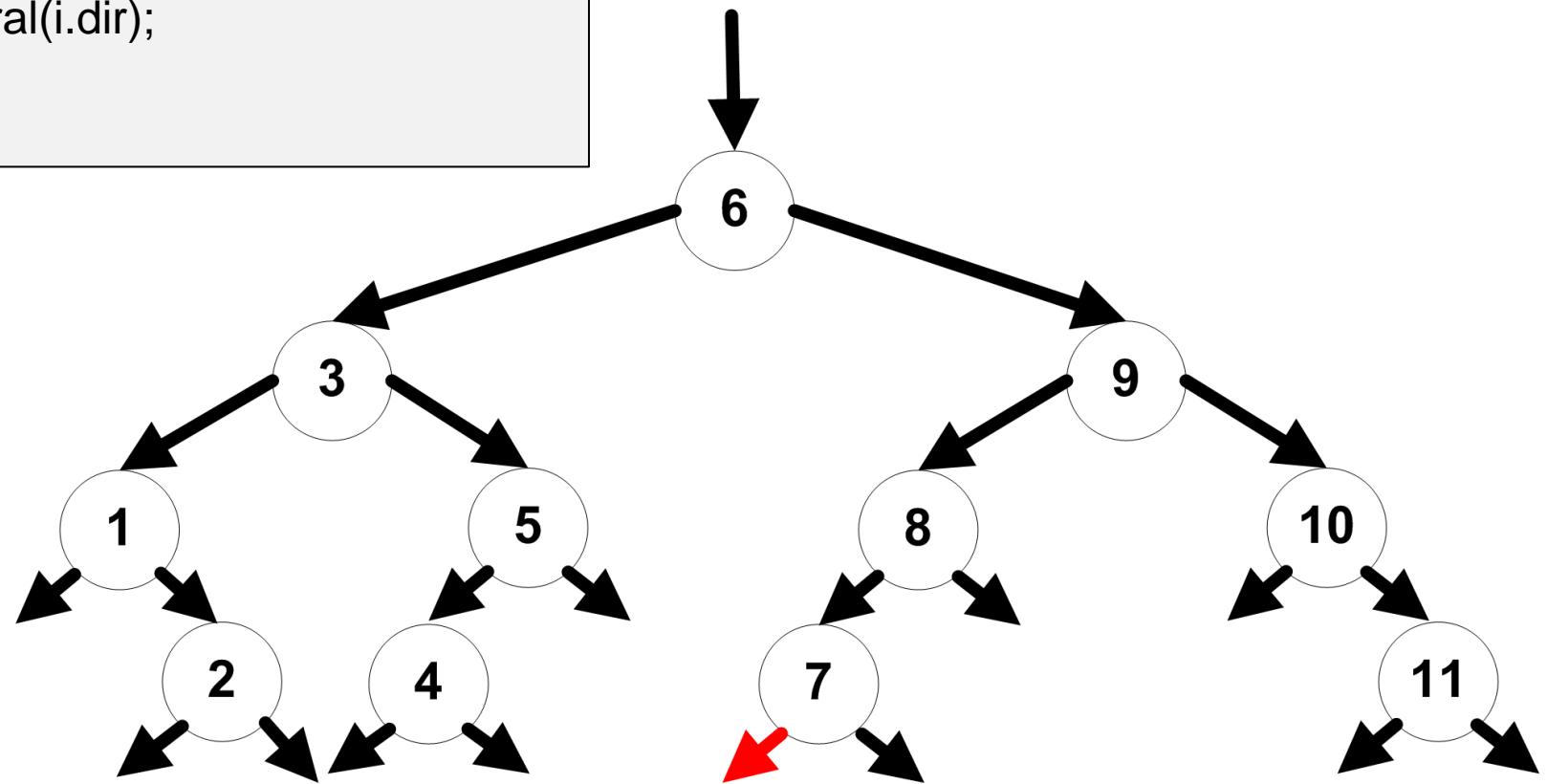


Tela

1 2 3 4 5 6

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

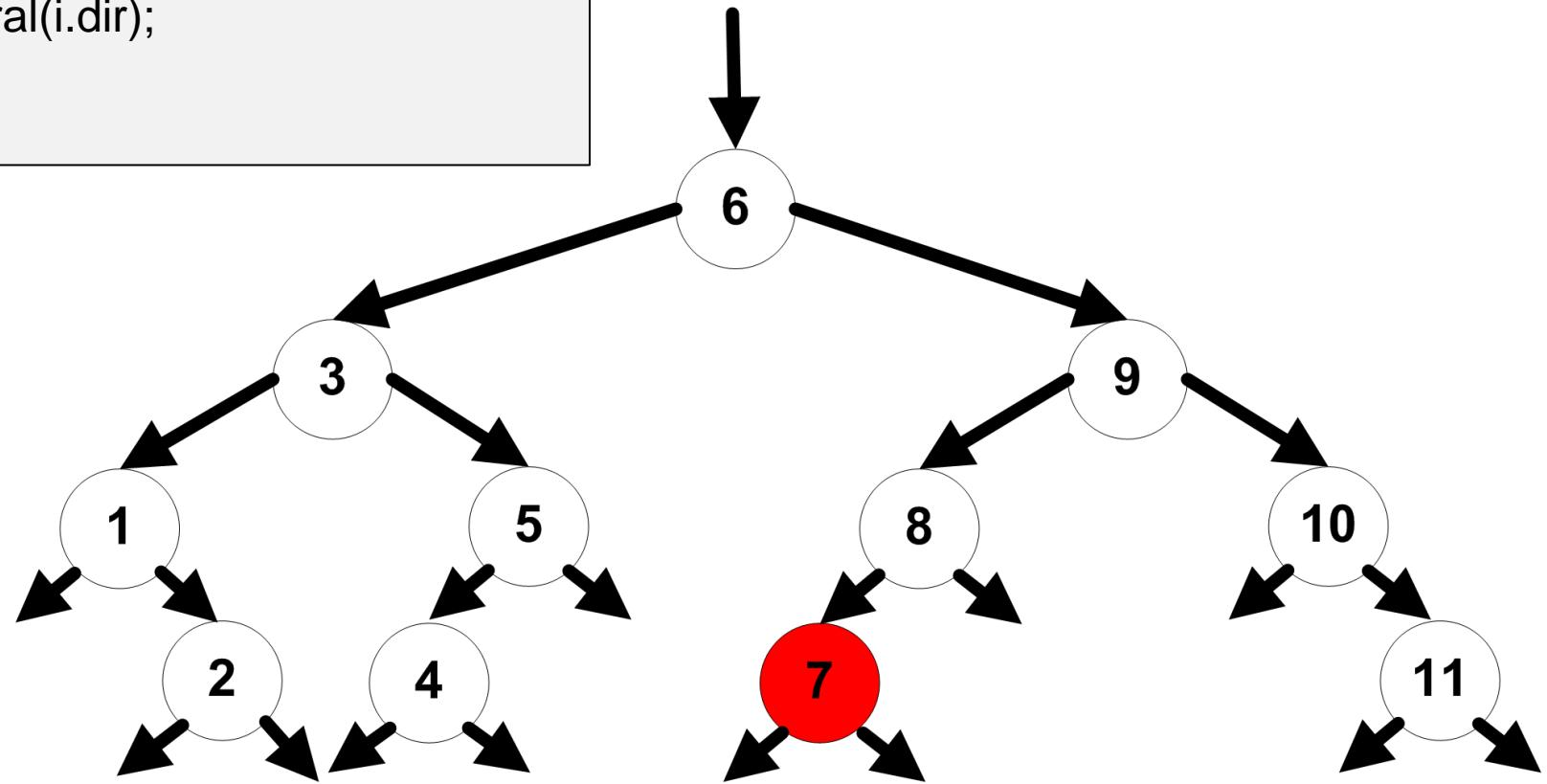


Tela

1 2 3 4 5 6

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

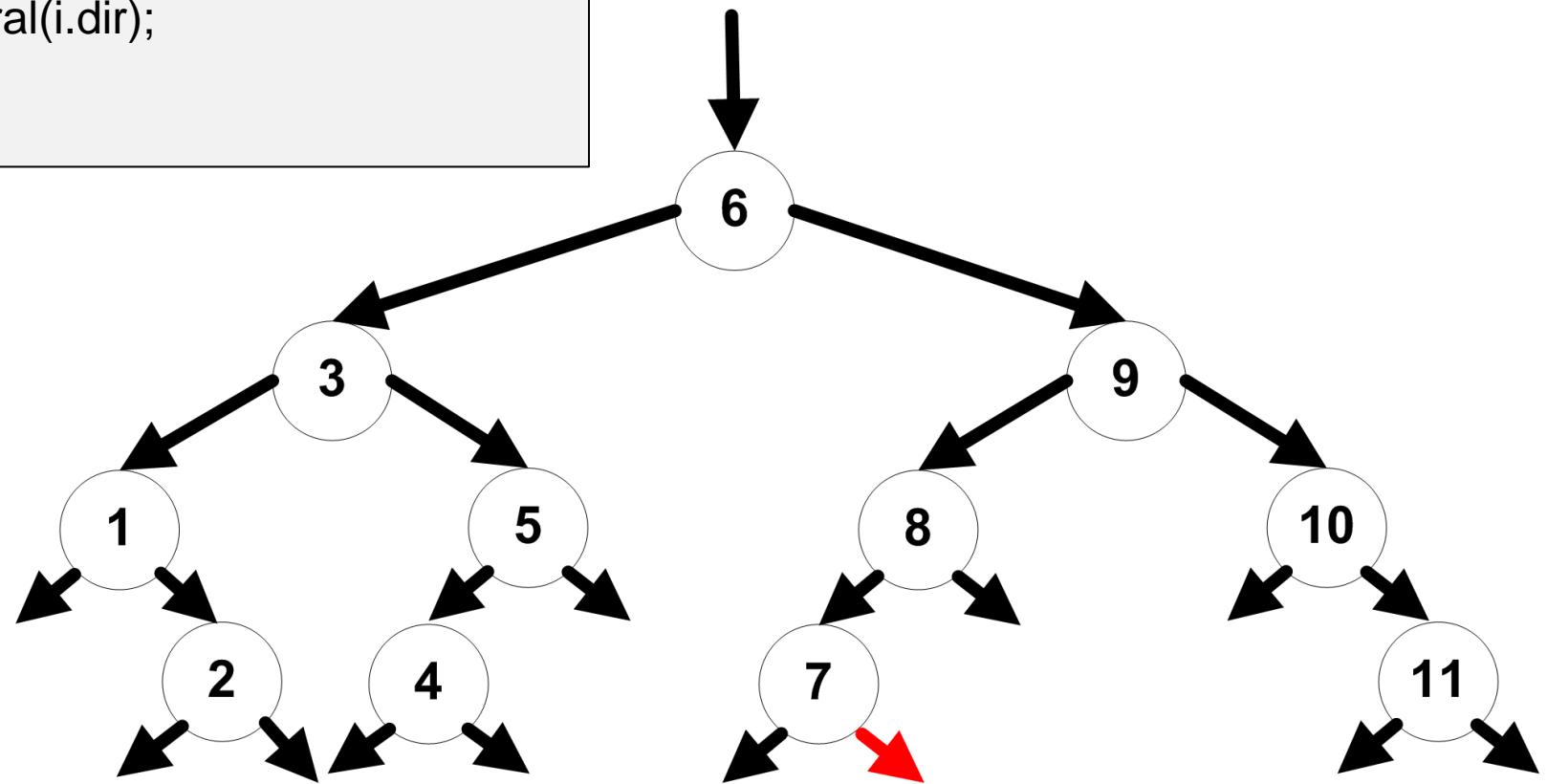


Tela

1 2 3 4 5 6 7

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

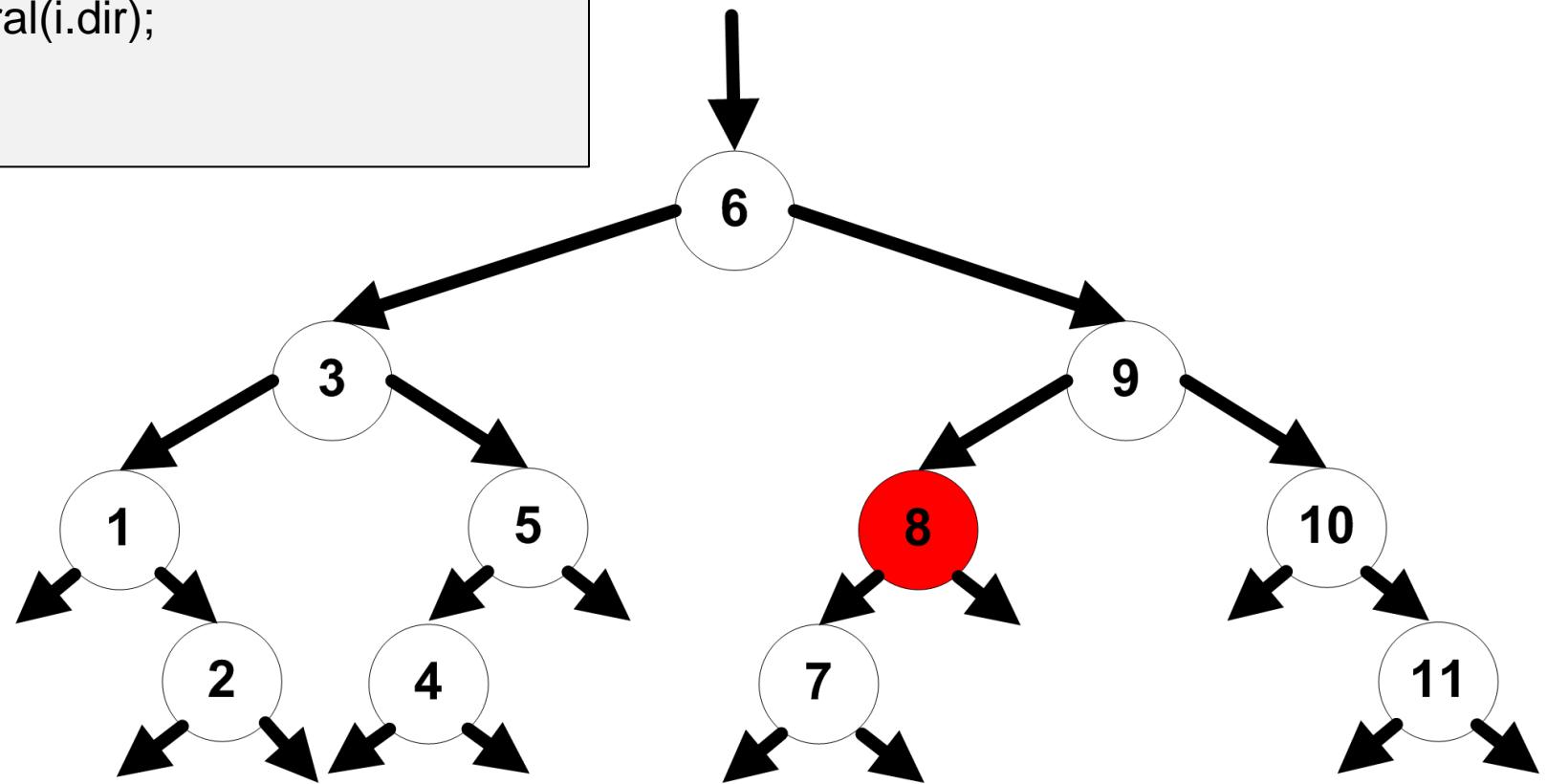


Tela

1 2 3 4 5 6 7

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

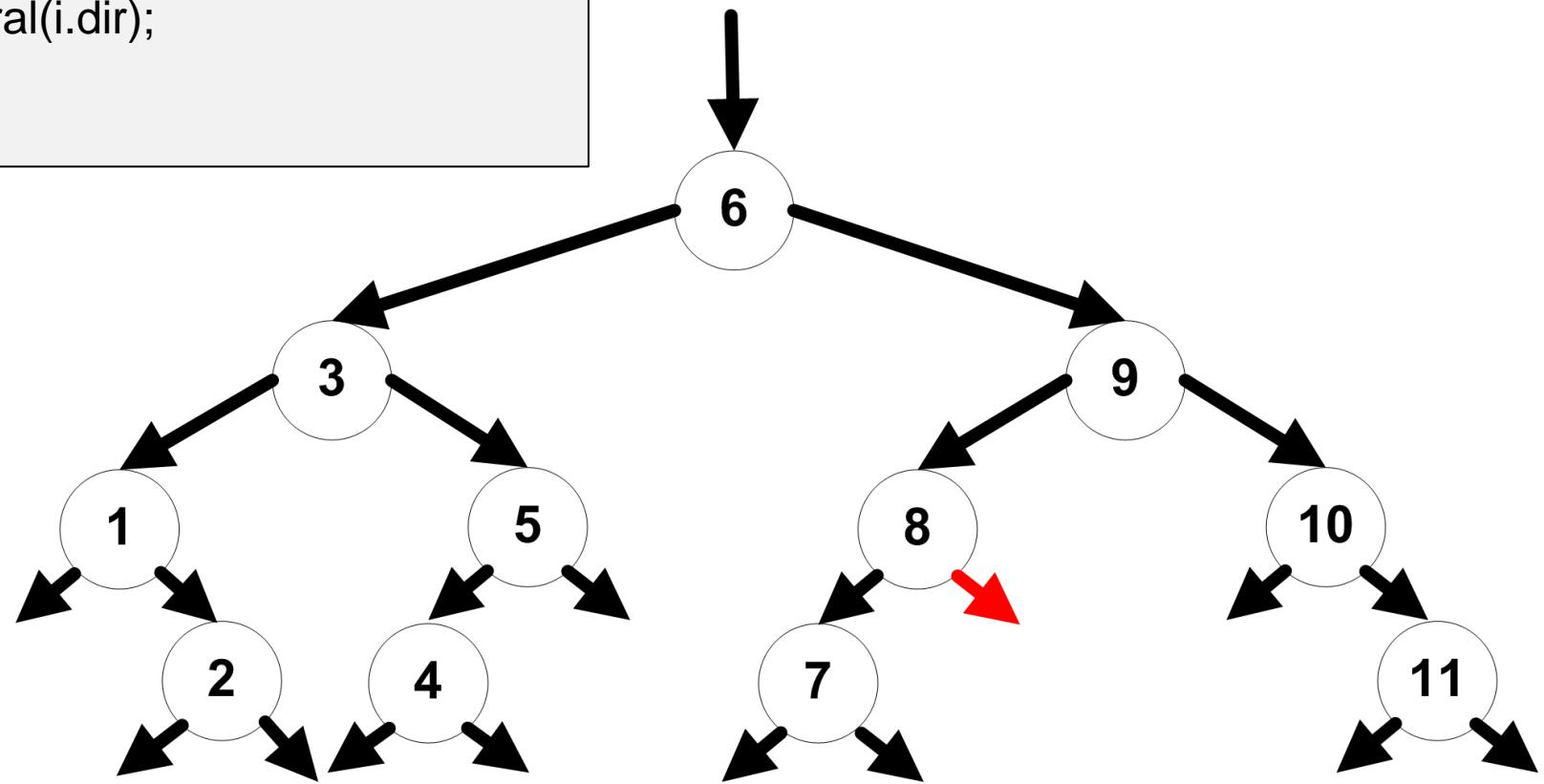


Tela

1 2 3 4 5 6 7 8

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

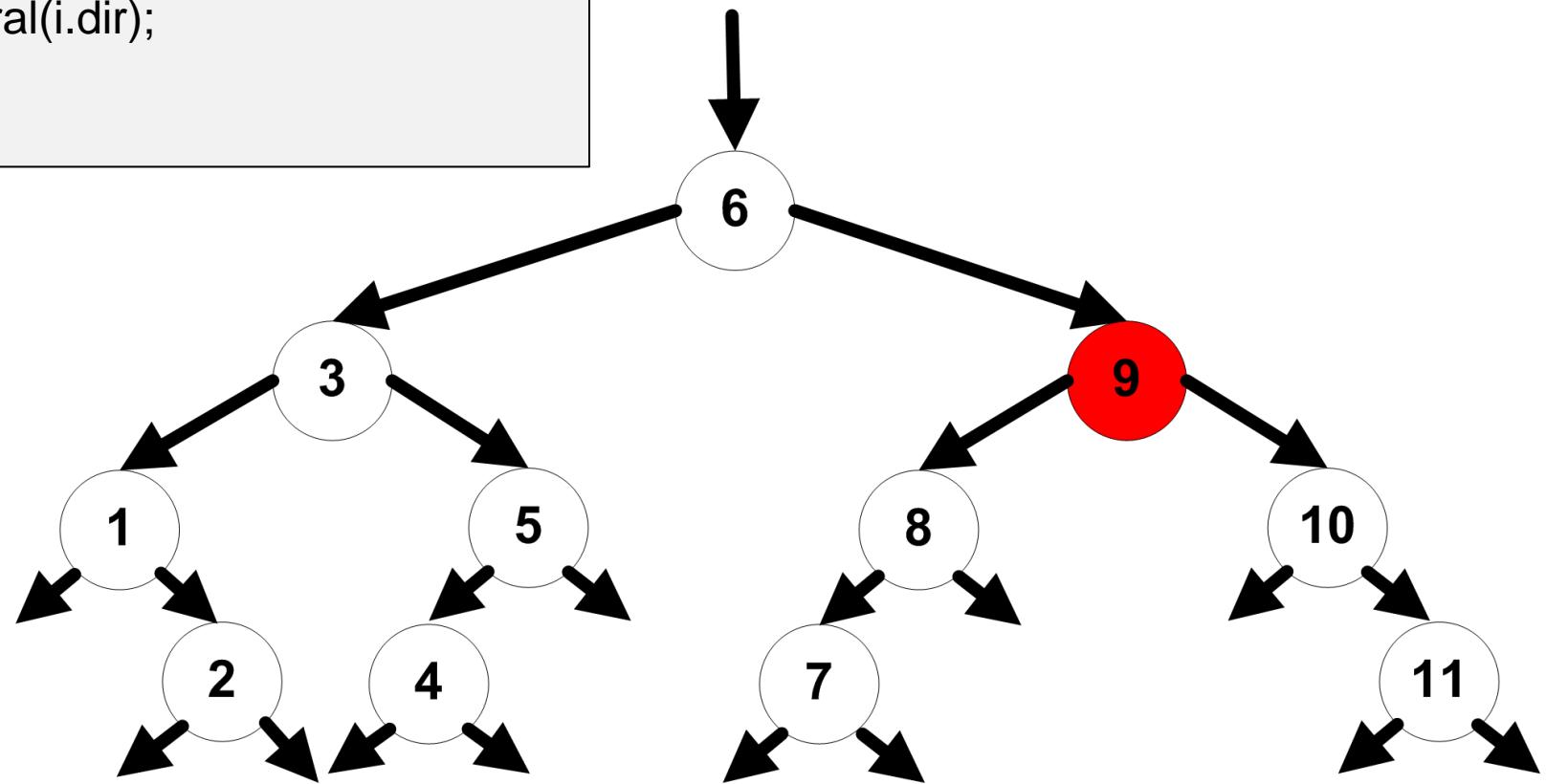


Tela

1 2 3 4 5 6 7 8

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

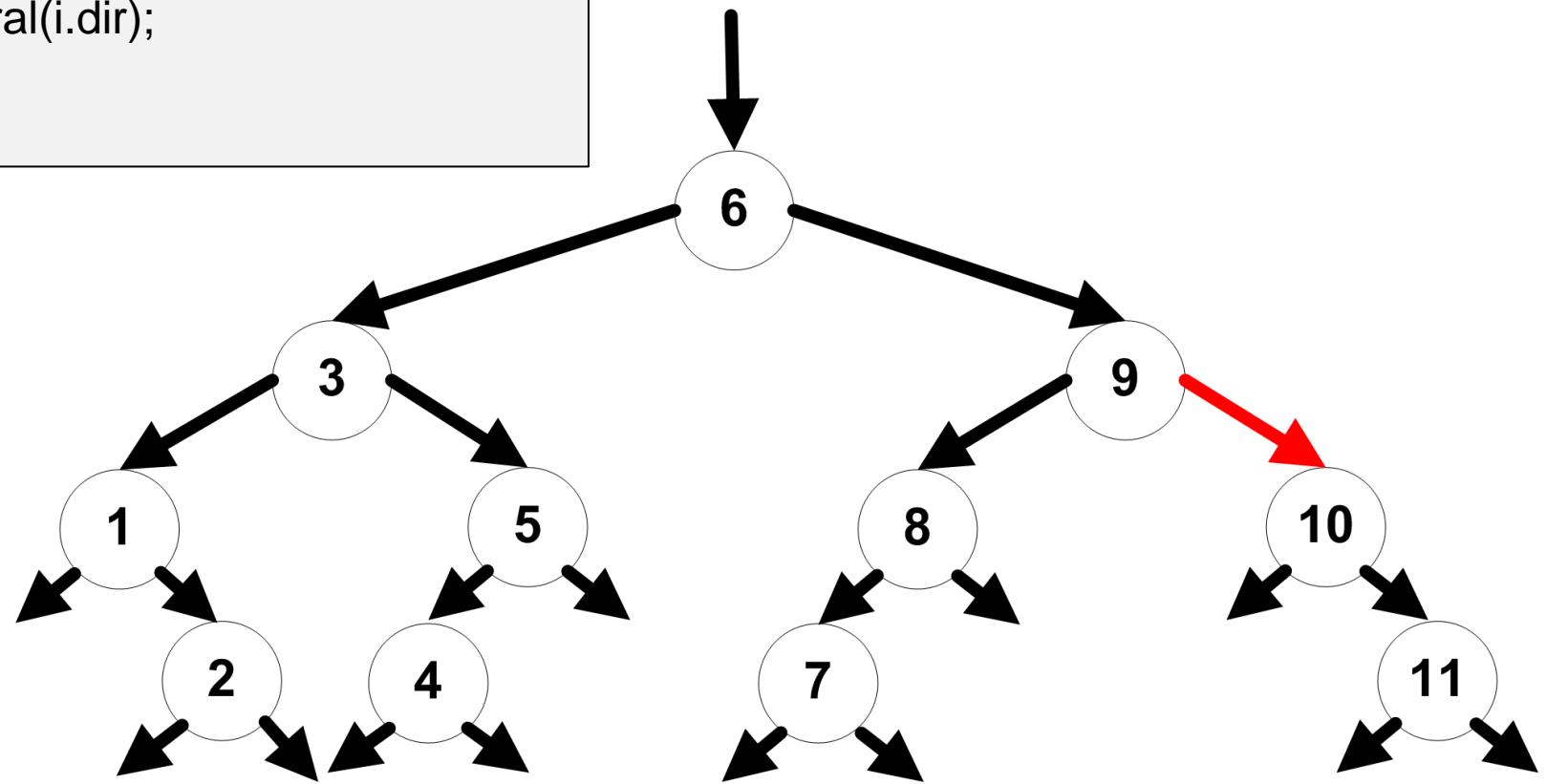


Tela

1 2 3 4 5 6 7 8 9

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

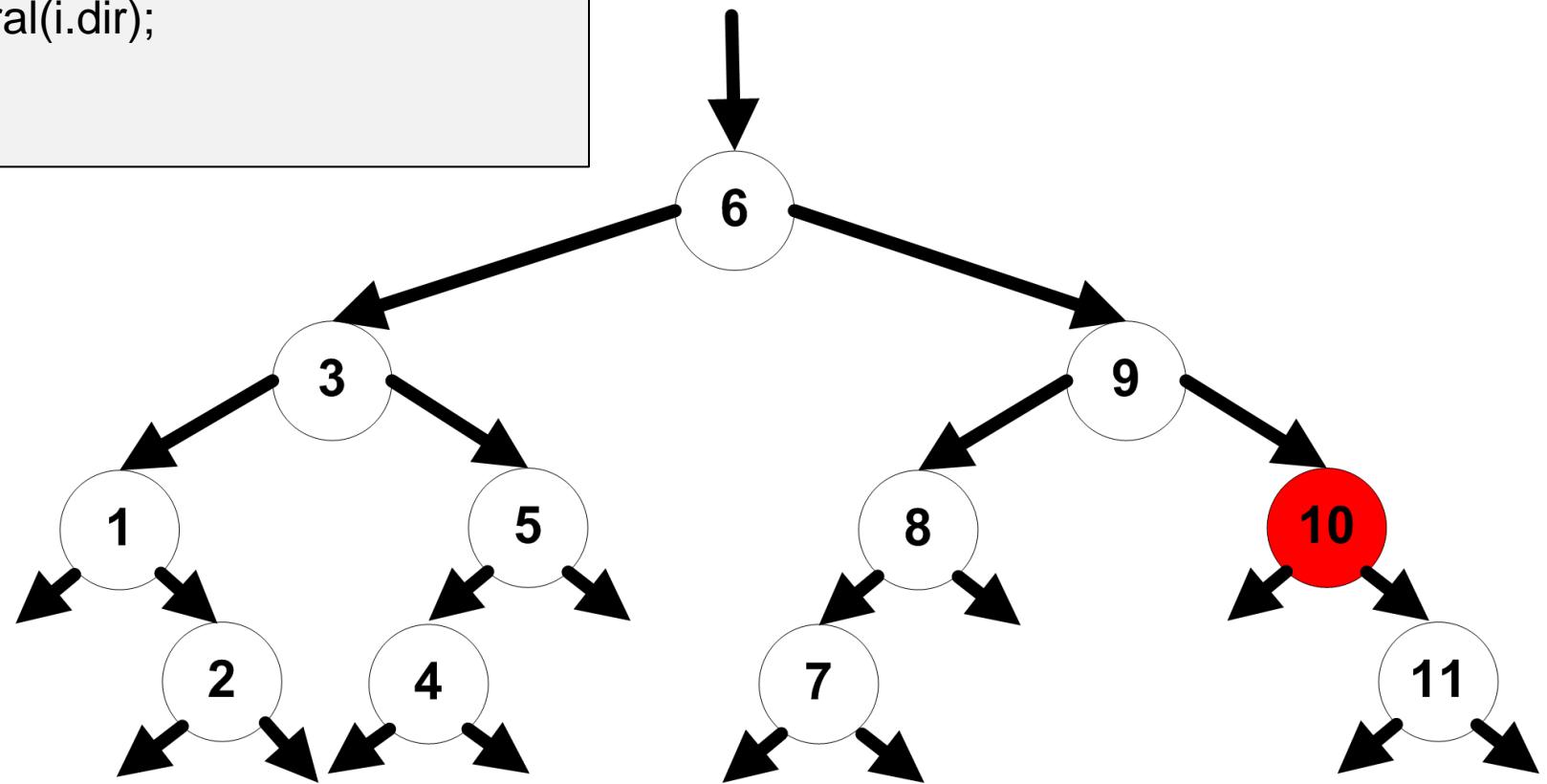


Tela

1 2 3 4 5 6 7 8 9

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

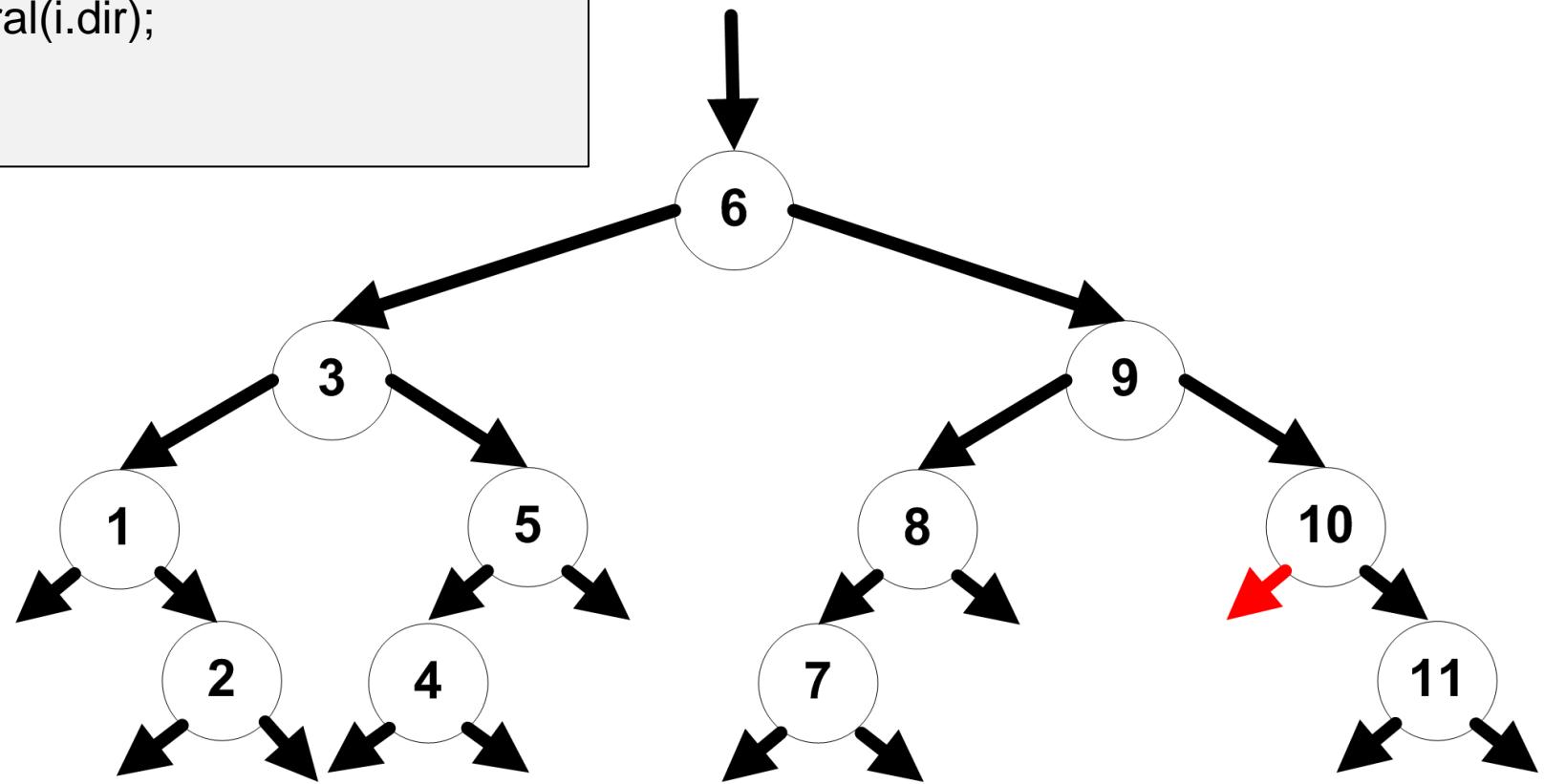


Tela

1 2 3 4 5 6 7 8 9

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

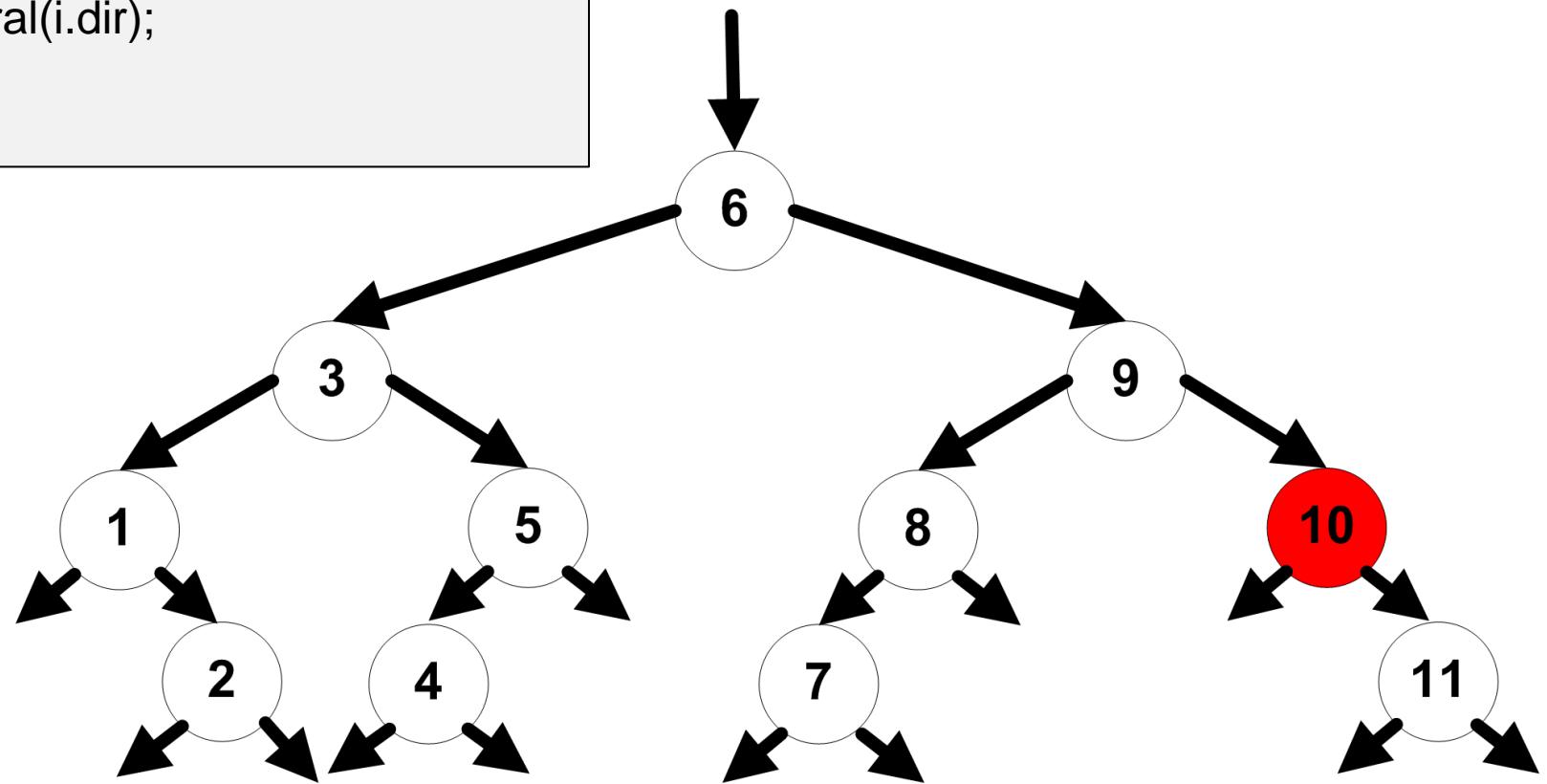


Tela

1 2 3 4 5 6 7 8 9

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

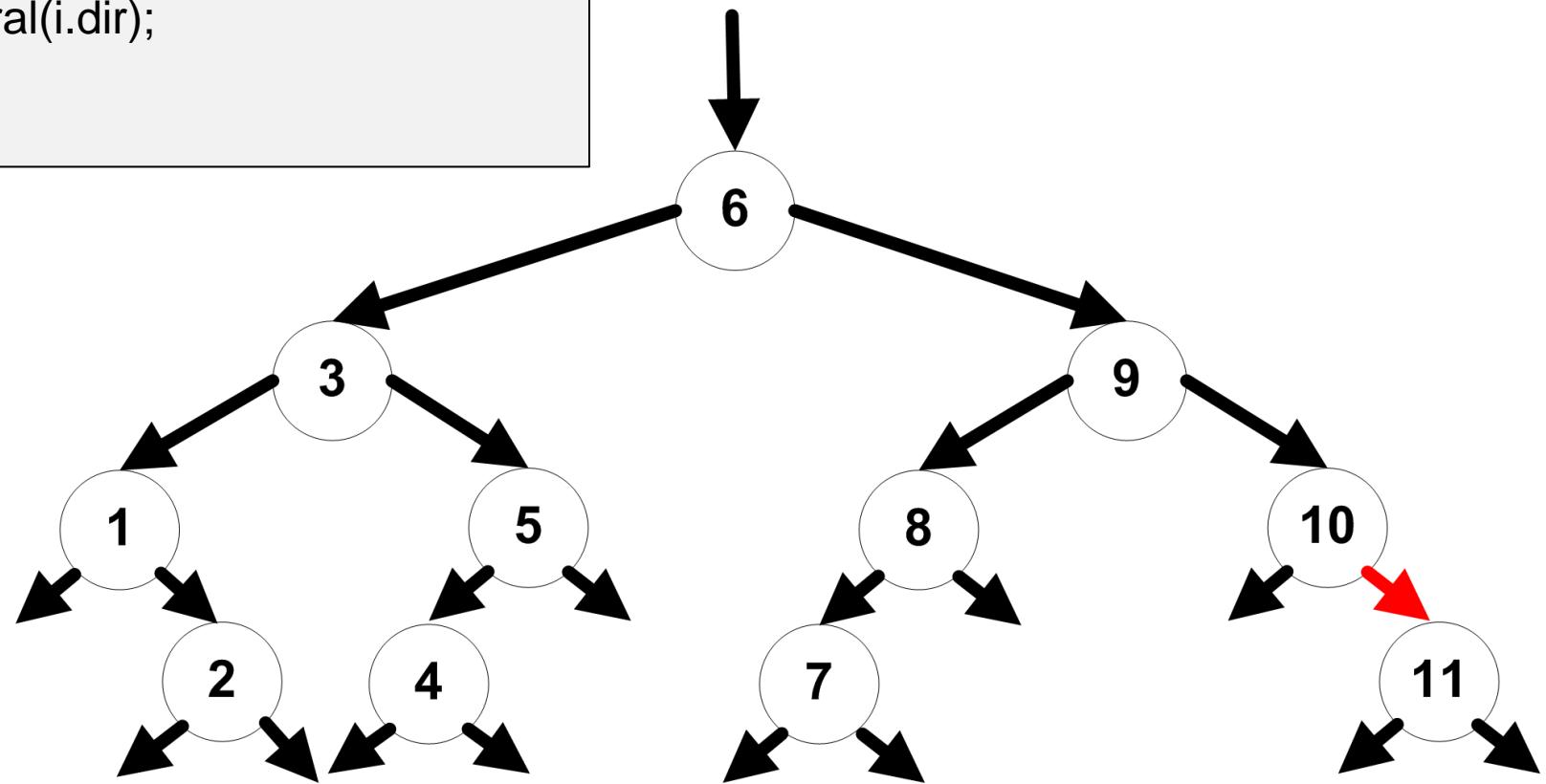


Tela

1 2 3 4 5 6 7 8 9 10

## Exercício

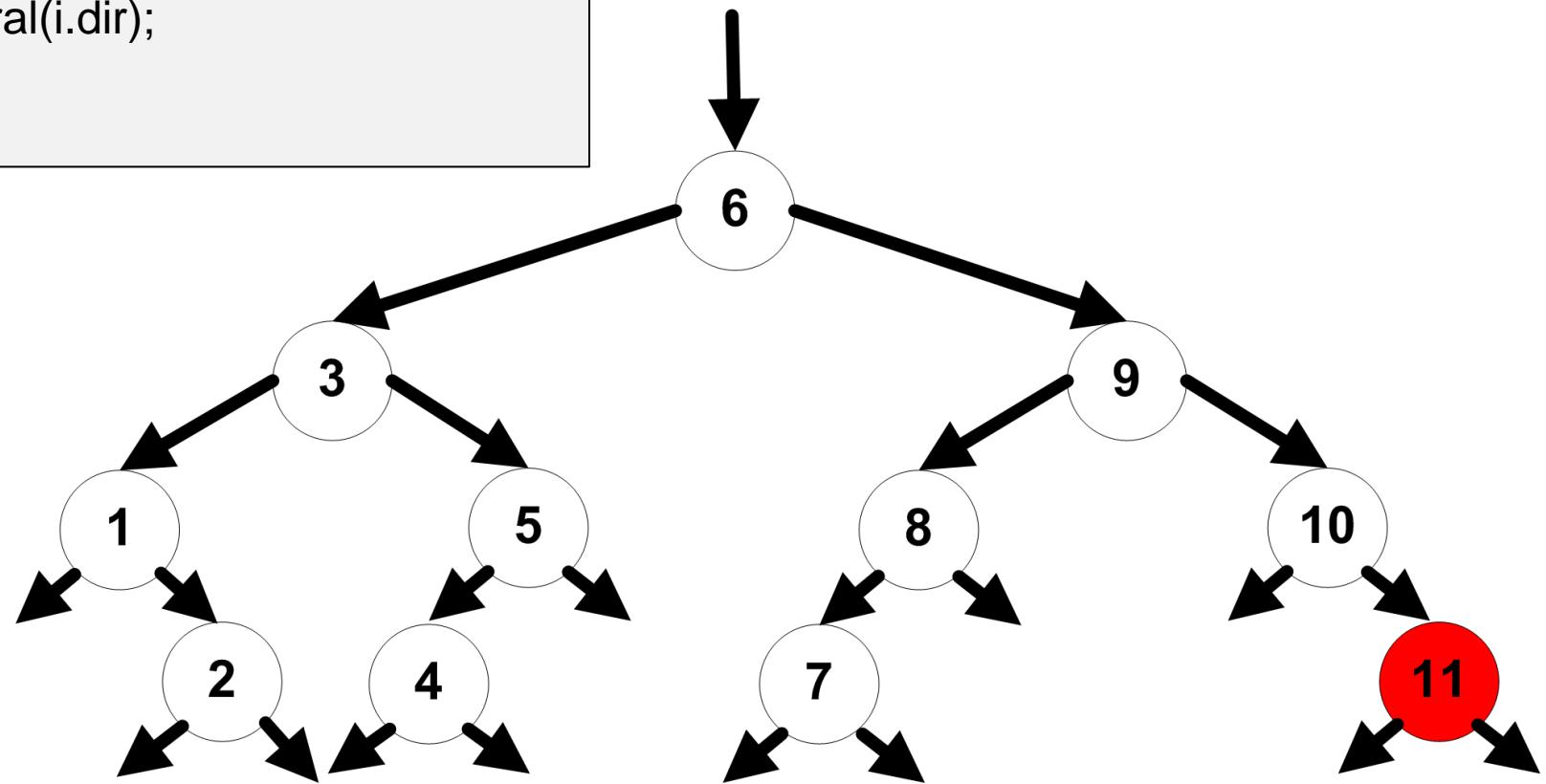
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela	1	2	3	4	5	6	7	8	9	10
------	---	---	---	---	---	---	---	---	---	----

## Exercício

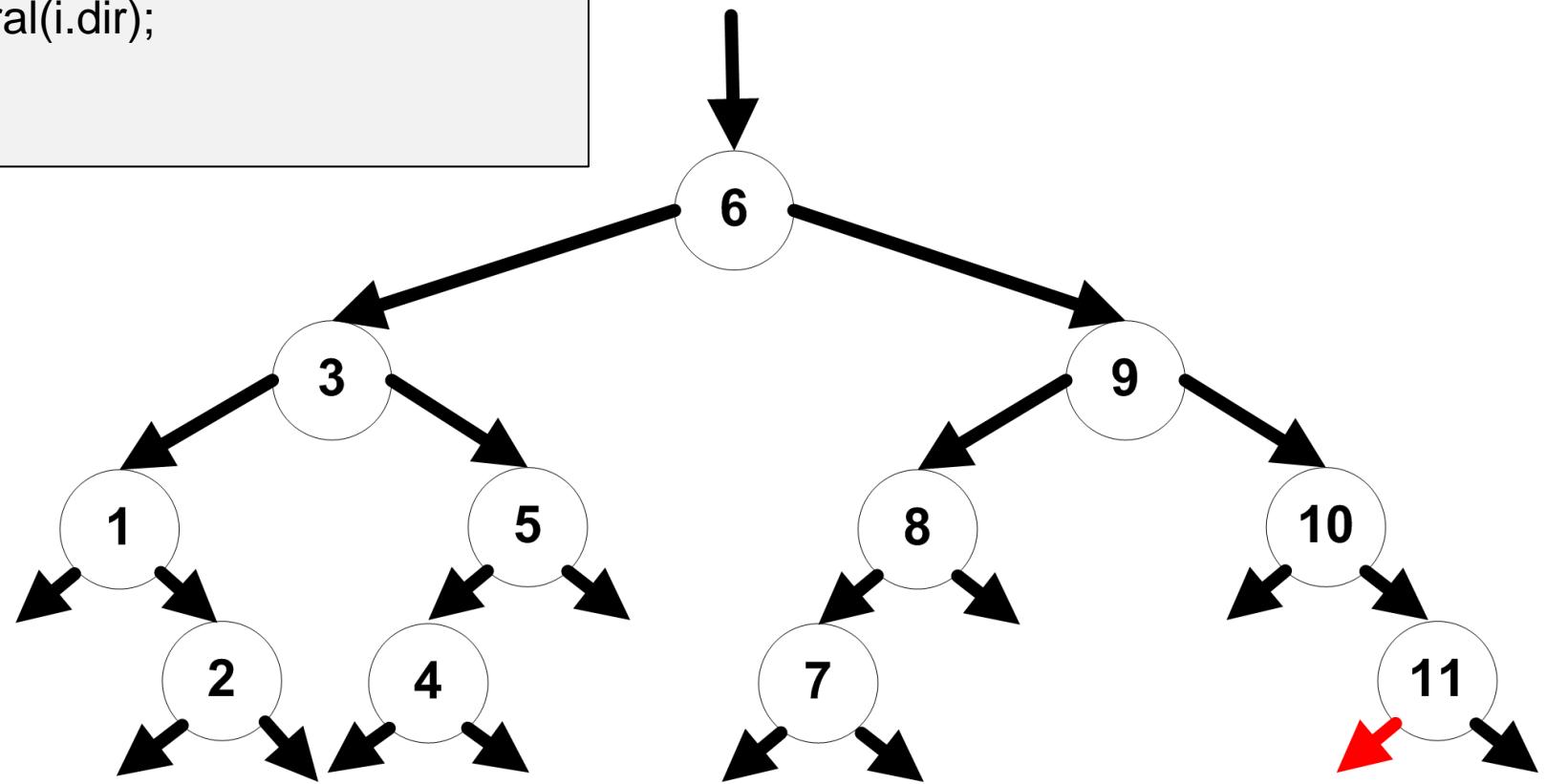
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela	1	2	3	4	5	6	7	8	9	10
------	---	---	---	---	---	---	---	---	---	----

## Exercício

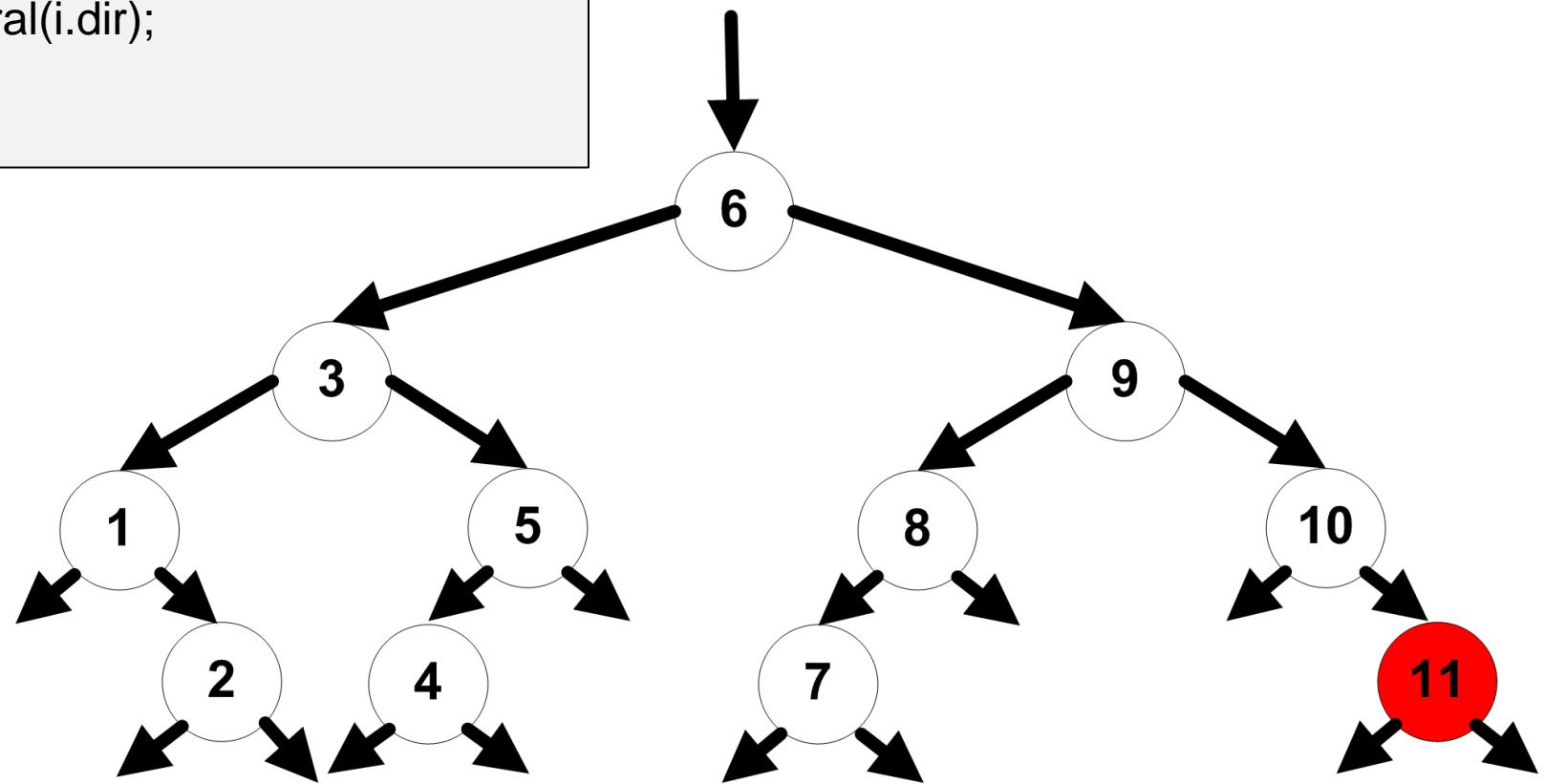
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela	1	2	3	4	5	6	7	8	9	10
------	---	---	---	---	---	---	---	---	---	----

## Exercício

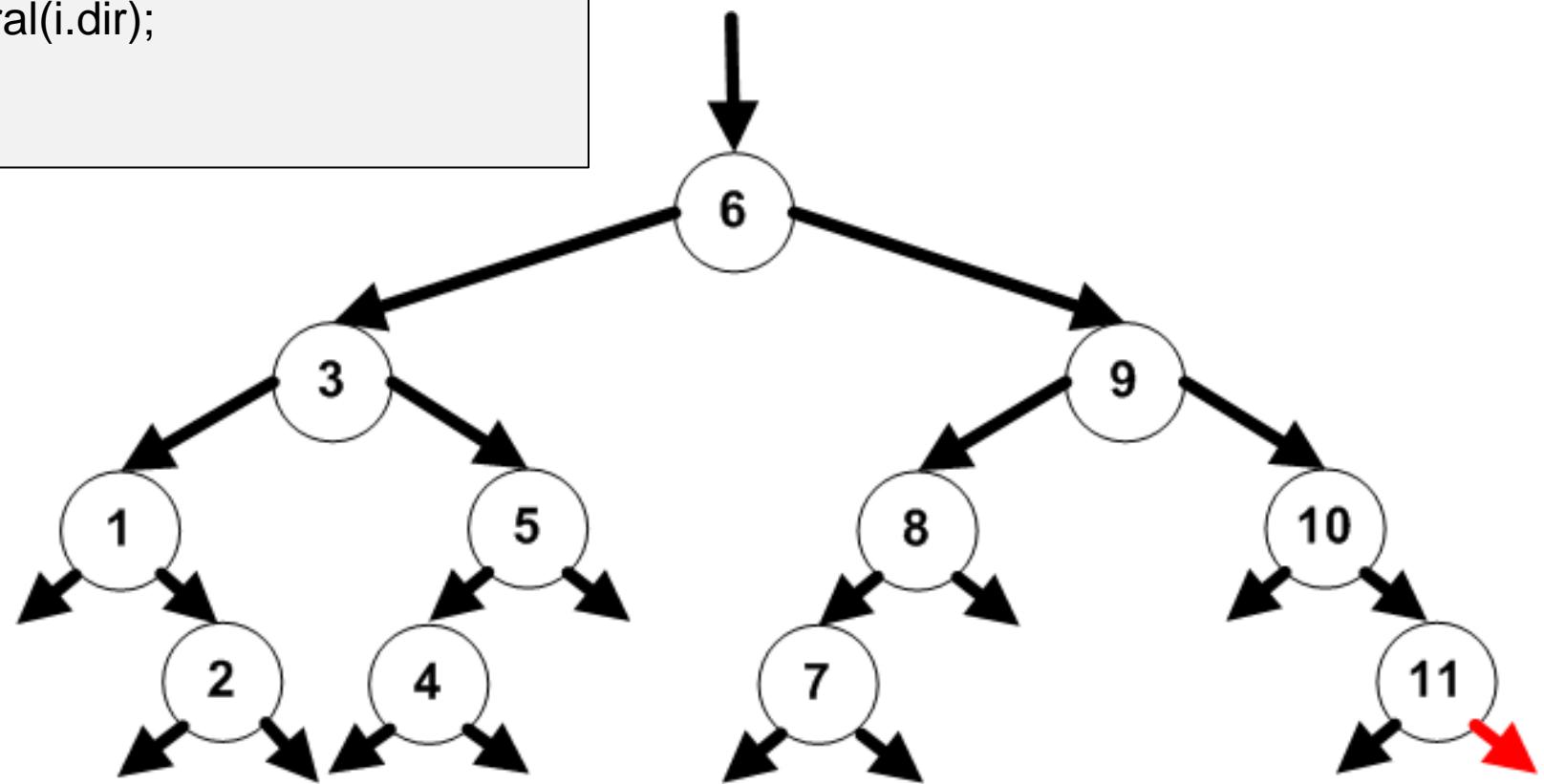
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela	1	2	3	4	5	6	7	8	9	10	11

## Exercício

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```



Tela	1	2	3	4	5	6	7	8	9	10	11

## Exercício

- Mostre a ordem em que os elementos da árvore abaixo são visitados pelos métodos de mostrar (a) central, (b) pós-fixado e (c) pré-fixado

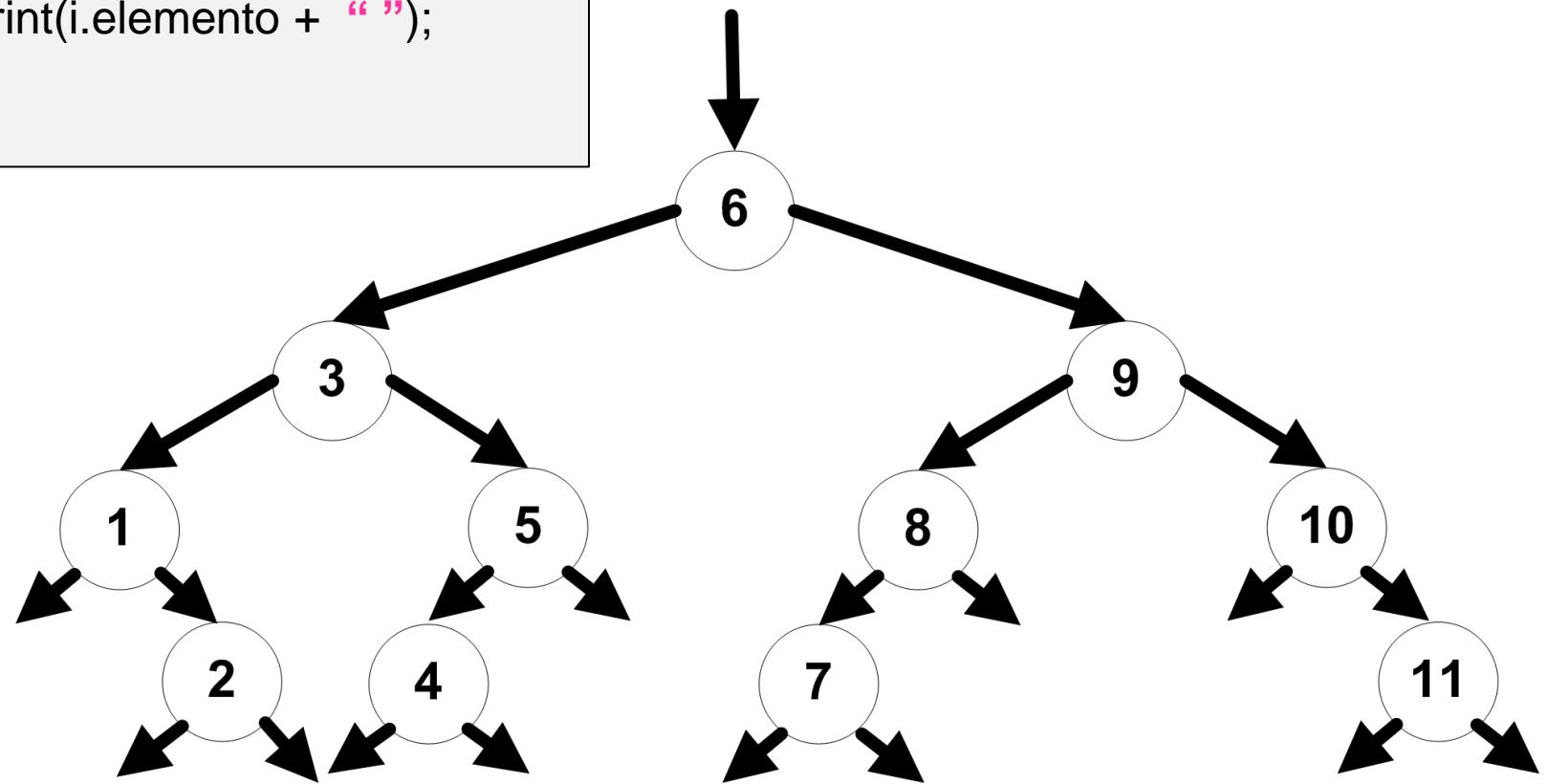
```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

```
private void mostrarPos(No i) {  
    if (i != null) {  
        mostrarPos(i.esq);  
        mostrarPos(i.dir);  
        System.out.print(i.elemento + " ");  
    }  
}
```

```
private void mostrarPre(No i) {  
    if (i != null) {  
        System.out.print(i.elemento + " ");  
        mostrarPre(i.esq);  
        mostrarPre(i.dir);  
    }  
}
```

## Exercício

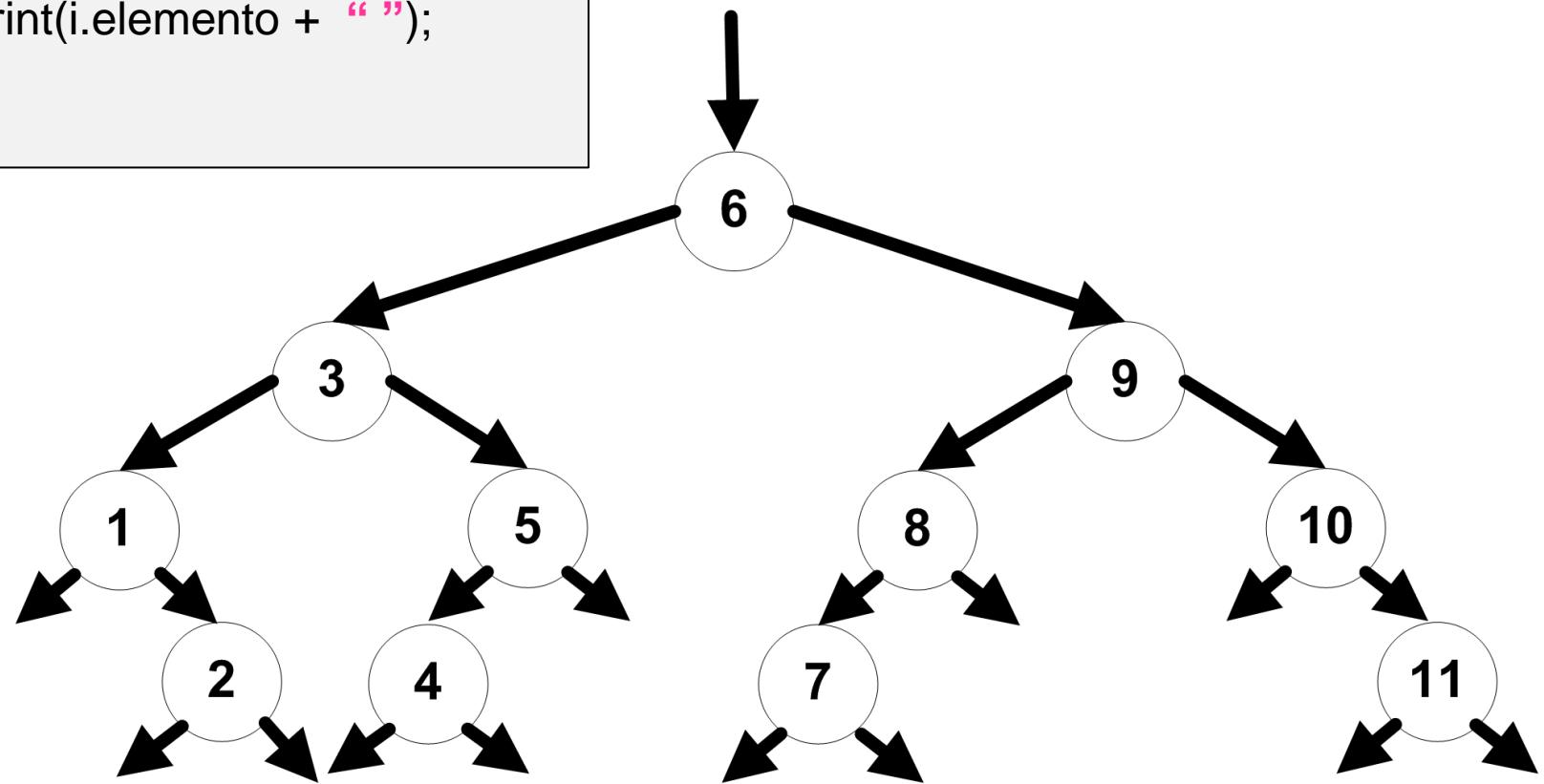
```
private void mostrarPos(No i) {  
    if (i != null) {  
        mostrarPos(i.esq);  
        mostrarPos(i.dir);  
        System.out.print(i.elemento + " ");  
    }  
}
```



Tela

## Exercício

```
private void mostrarPos(No i) {  
    if (i != null) {  
        mostrarPos(i.esq);  
        mostrarPos(i.dir);  
        System.out.print(i.elemento + " ");  
    }  
}
```



Tela

2 1 4 5 3 7 8 11 10 9 6

## Exercício

- Mostre a ordem em que os elementos da árvore abaixo são visitados pelos métodos de mostrar (a) central, (b) pós-fixado e (c) pré-fixado

```
private void mostrarCentral(No i) {  
    if (i != null) {  
        mostrarCentral(i.esq);  
        System.out.print(i.elemento + " ");  
        mostrarCentral(i.dir);  
    }  
}
```

```
private void mostrarPos(No i) {  
    if (i != null) {  
        mostrarPos(i.esq);  
        mostrarPos(i.dir);  
        System.out.print(i.elemento + " ");  
    }  
}
```

```
private void mostrarPre(No i) {  
    if (i != null) {  
        System.out.print(i.elemento + " ");  
        mostrarPre(i.esq);  
        mostrarPre(i.dir);  
    }  
}
```

## Exercício

```
private void mostrarPre(No i) {
```

```
    if (i != null) {
```

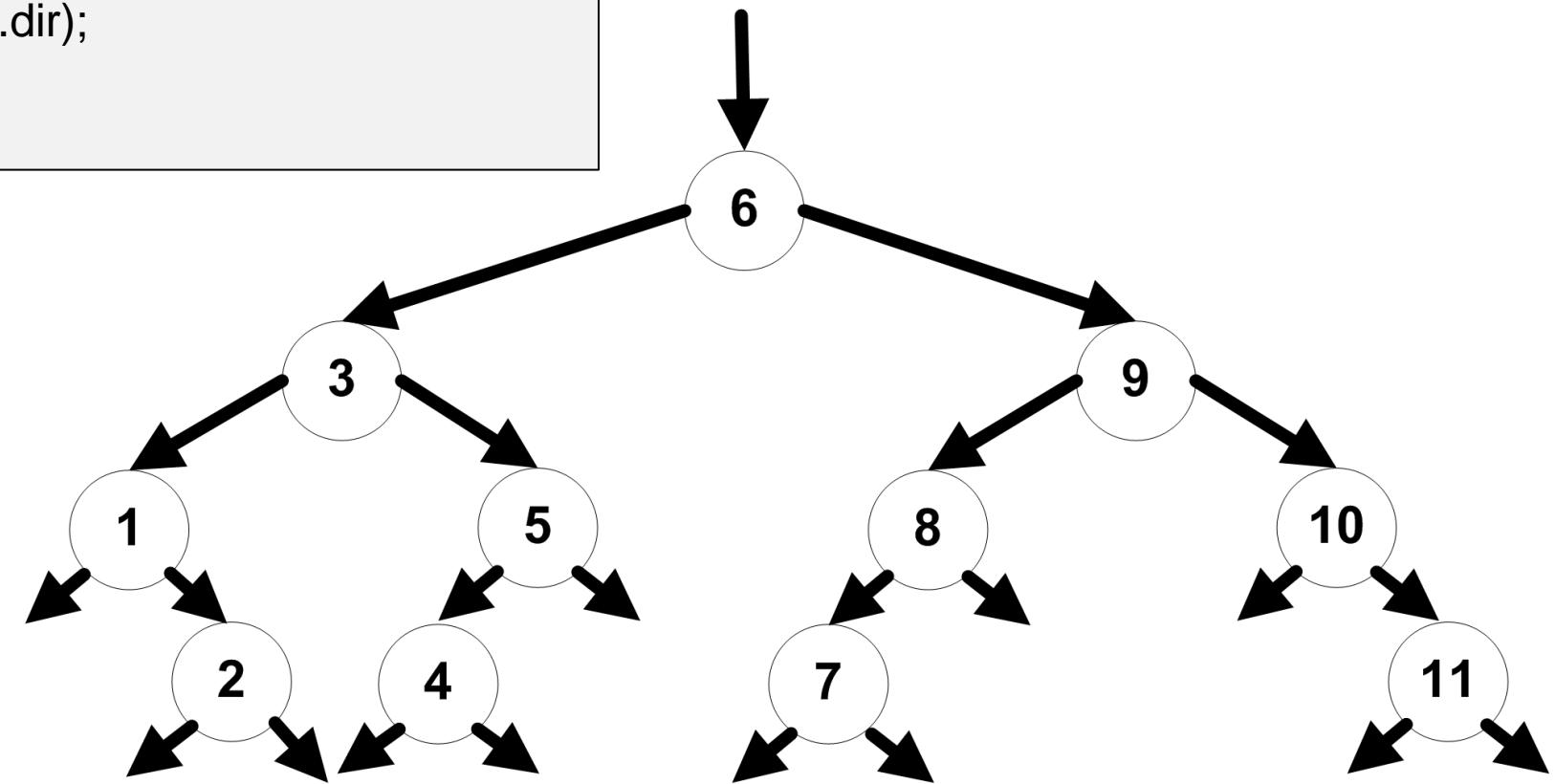
```
        System.out.print(i.elemento + " ");
```

```
        mostrarPre(i.esq);
```

```
        mostrarPre(i.dir);
```

```
}
```

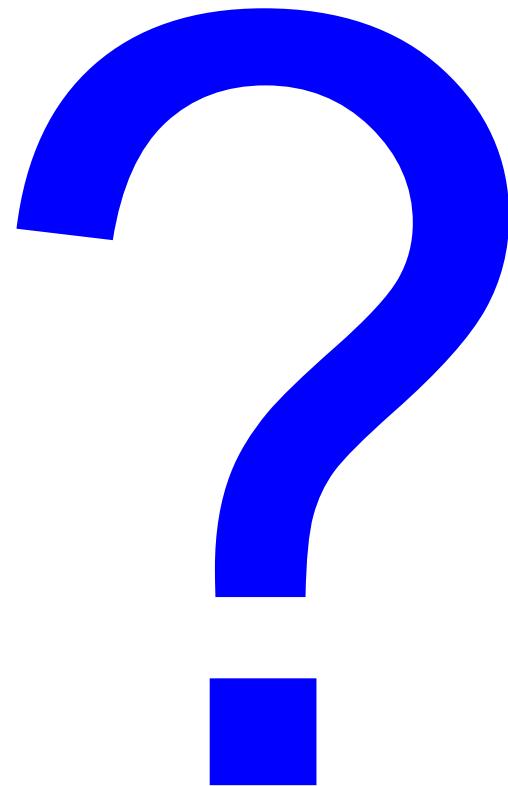
```
}
```



Tela	6	3	1	2	5	4	9	8	7	10	11
------	---	---	---	---	---	---	---	---	---	----	----

6	3	1	2	5	4	9	8	7	10	11
---	---	---	---	---	---	---	---	---	----	----

# Análise de Complexidade do Mostrar



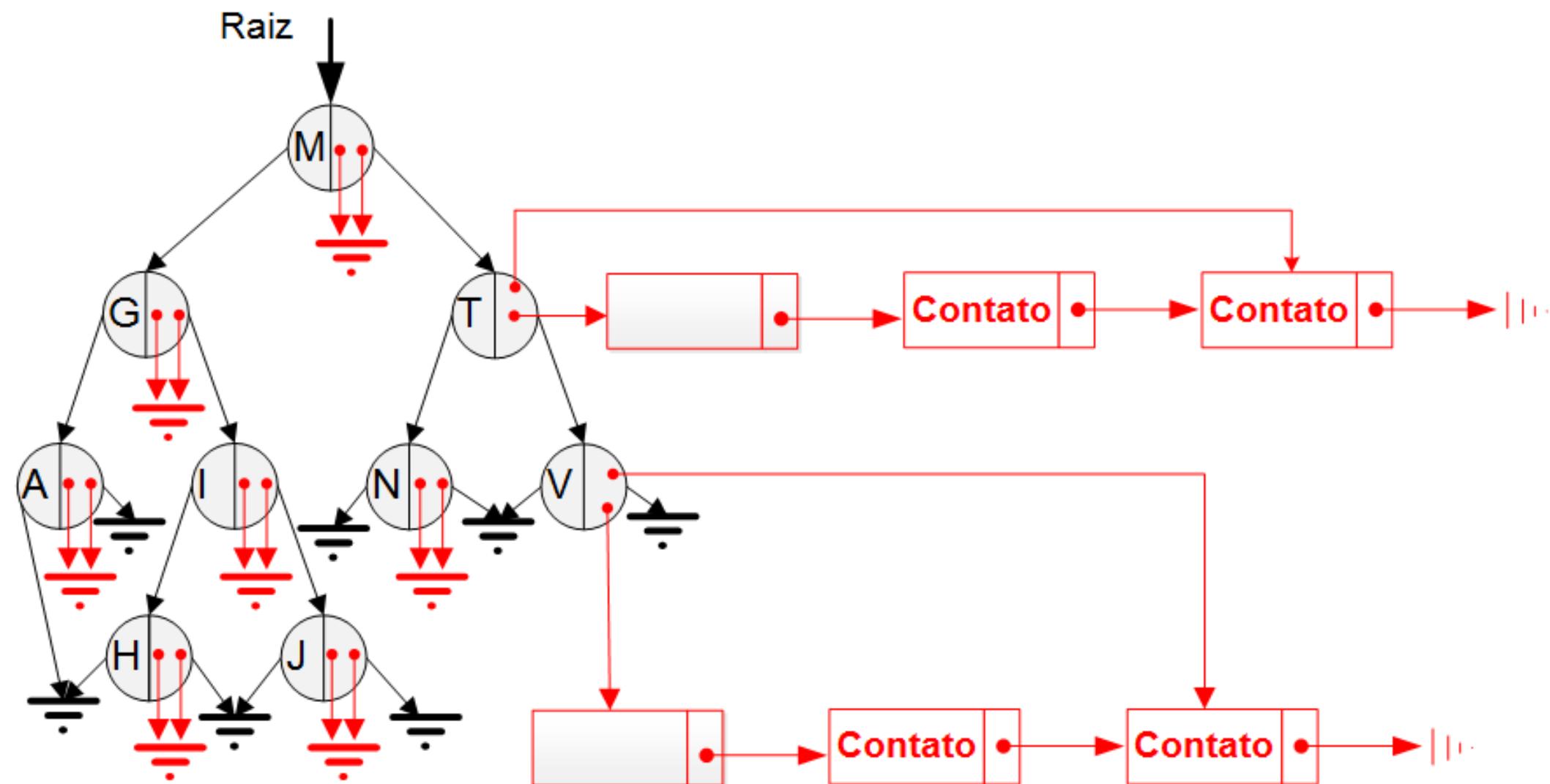
# Exercício (1)

- Você foi contratado para desenvolver uma agenda de contatos (atributos nome, telefone, email e CPF) para um escritório de advocacia
- Um colega sugeriu que você implementasse uma árvore de binária de listas em que a pesquisa na árvore acontece pela primeira letra do nome e, quando encontramos a letra, temos uma pesquisa em uma lista de contatos
- Crie uma classe Agenda contendo o atributo No raiz, os métodos inserir(Contato contato), remover(String nome), pesquisar(String nome) e pesquisar(int cpf). Para cada método, mostre o melhor e pior caso

# Exercício (1)

- Crie uma classe No contendo os atributos Celula primeiro e ultimo, No esq e dir, e char letra
- Crie uma classe Celula contendo os atributos Contato contato e Celula prox
- Crie uma classe Contato contendo os atributos String nome, telefone e email e int CPF

## Exercício (1)



## Exercício (2)

- Implemente os métodos pesquisar, inserir, remover para a estrutura abaixo:

