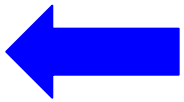
Unidade VI:
Árvores Binárias

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

Agenda

- **Definições e conceitos** 
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

Introdução

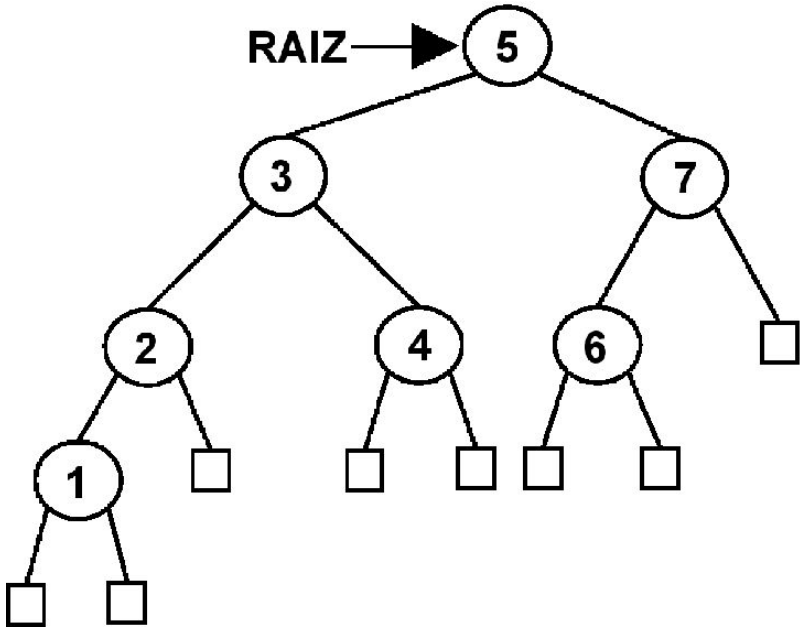
- A inserção, remoção e pesquisa nas estruturas de dados lineares (pilhas, filas e listas) têm custo de Θ(n) comparações

- O custo de pesquisa em listas estáticas ordenadas é de Θ(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 outro de arcos (arestas) que conectam os vértices

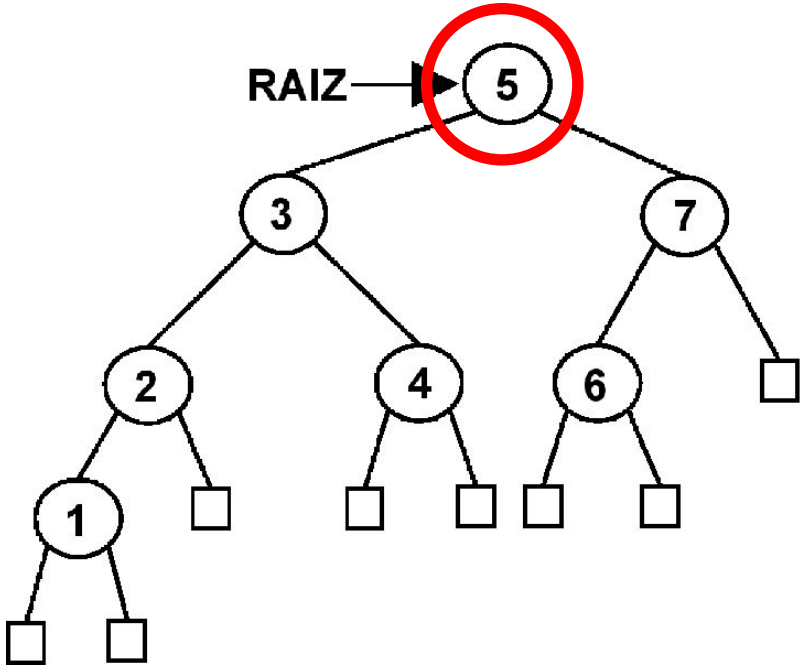


Árvore

- 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 outro de arcos (arestas) que conectam os vértices

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

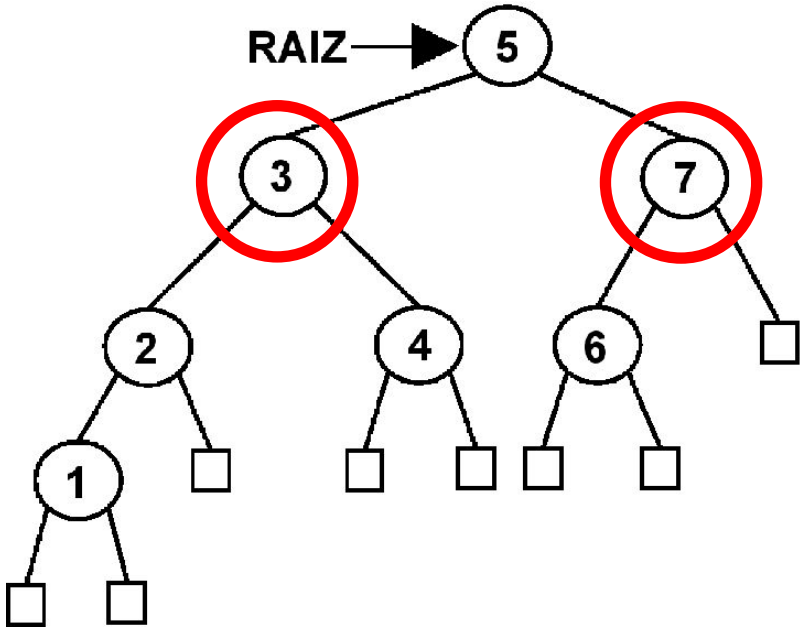


Árvore

- 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 outro de arcos (arestas) que conectam os vértices

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

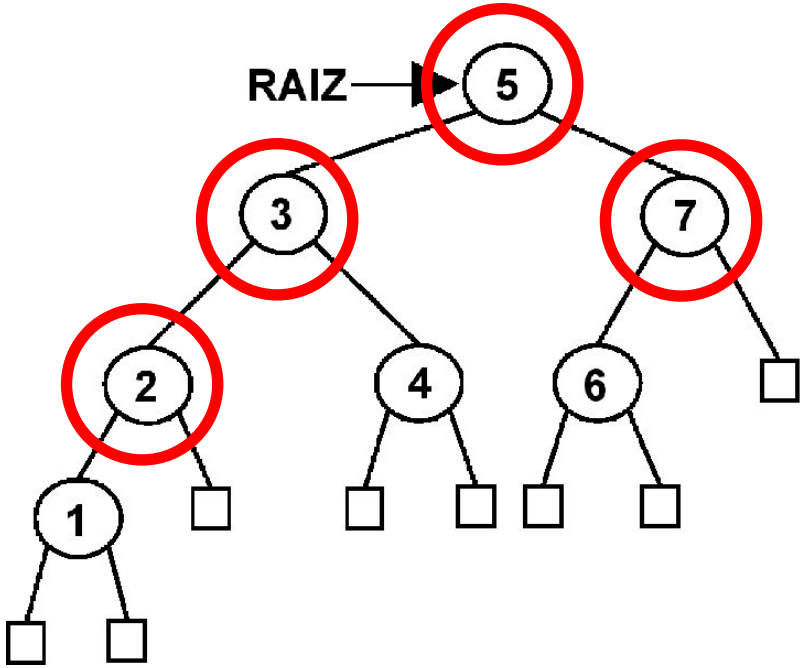


Árvore

- 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 outro de arcos (arestas) que conectam os vértices

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

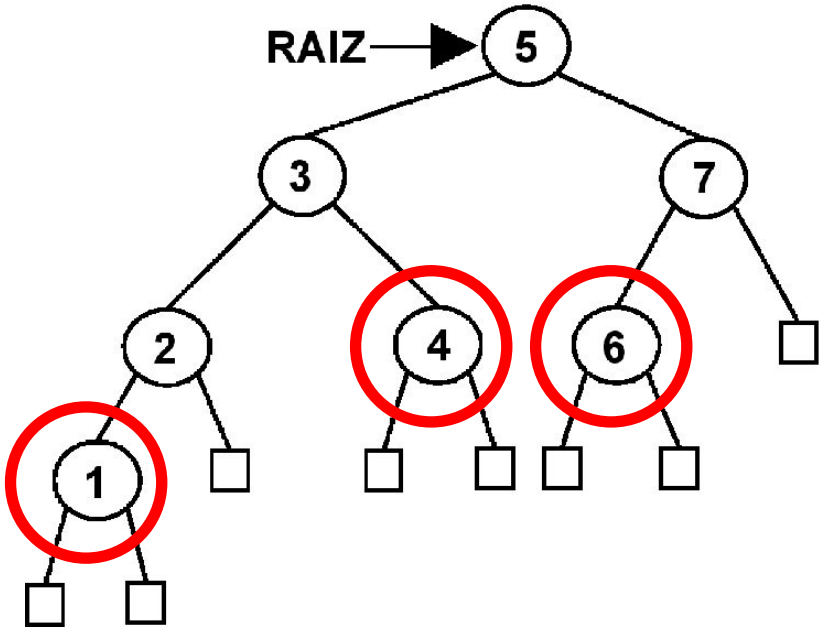


Árvore

- 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 outro de arcos (arestas) que conectam os vértices

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

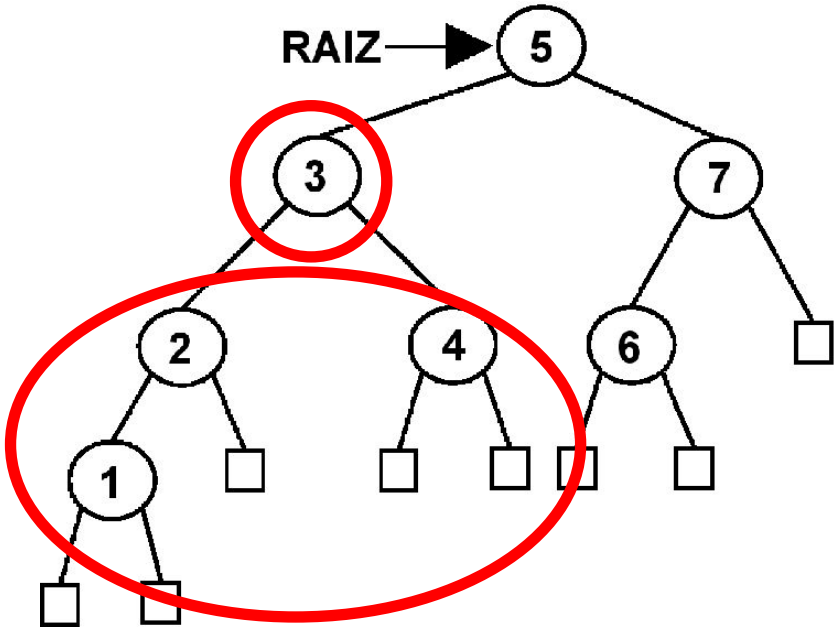


Árvore

- 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 outro de arcos (arestas) que conectam os vértices

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

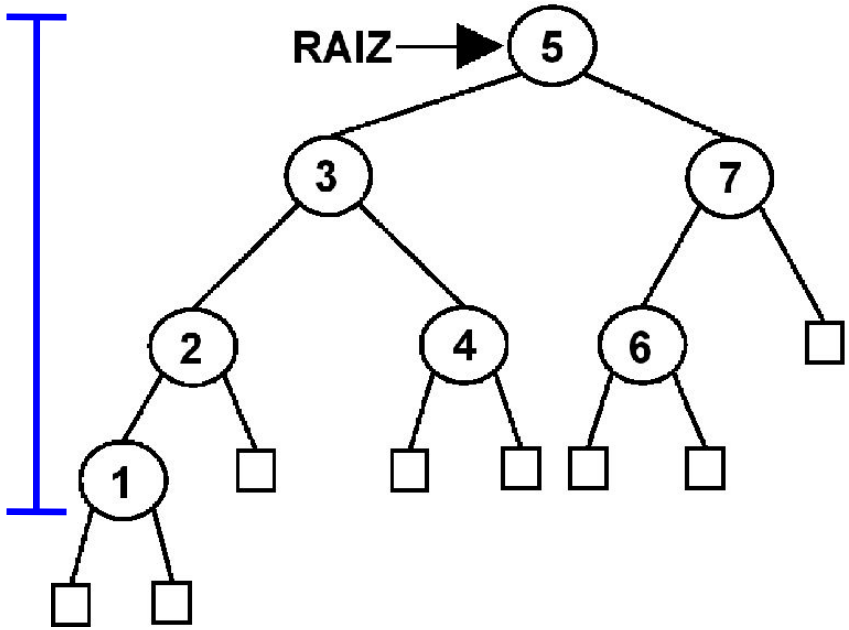


Árvore

- 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 outro 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

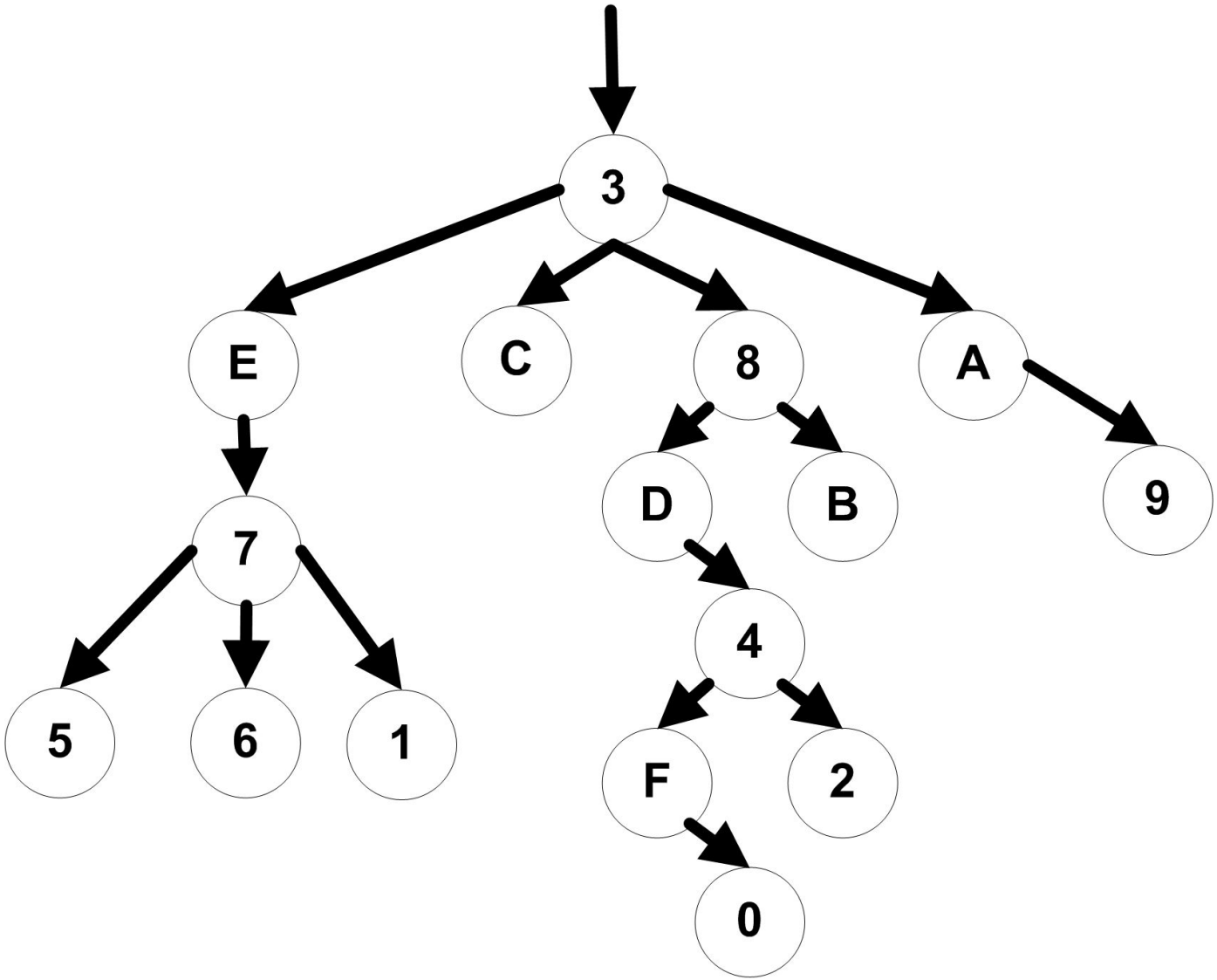


Exercício Resolvido (1)

- Crie uma árvore com os dígitos hexadecimais

Exercício Resolvido (1)

- Crie uma árvore com os dígitos hexadecimais



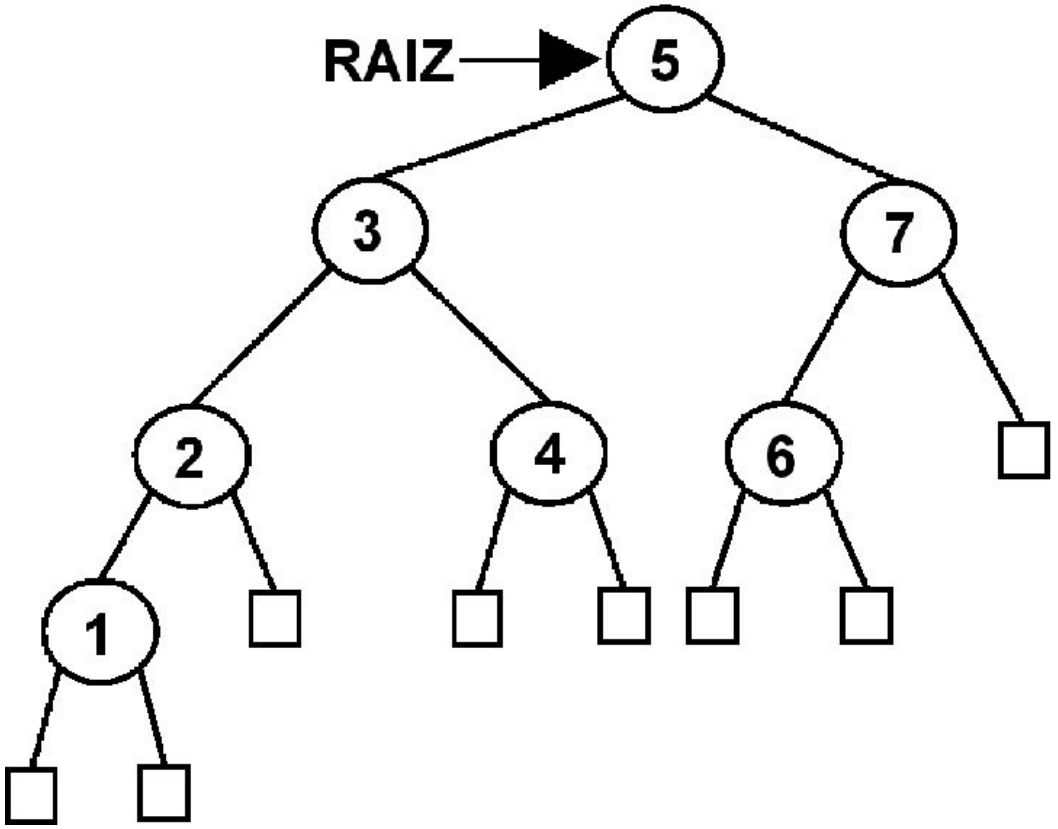
Árvore Binária

- 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

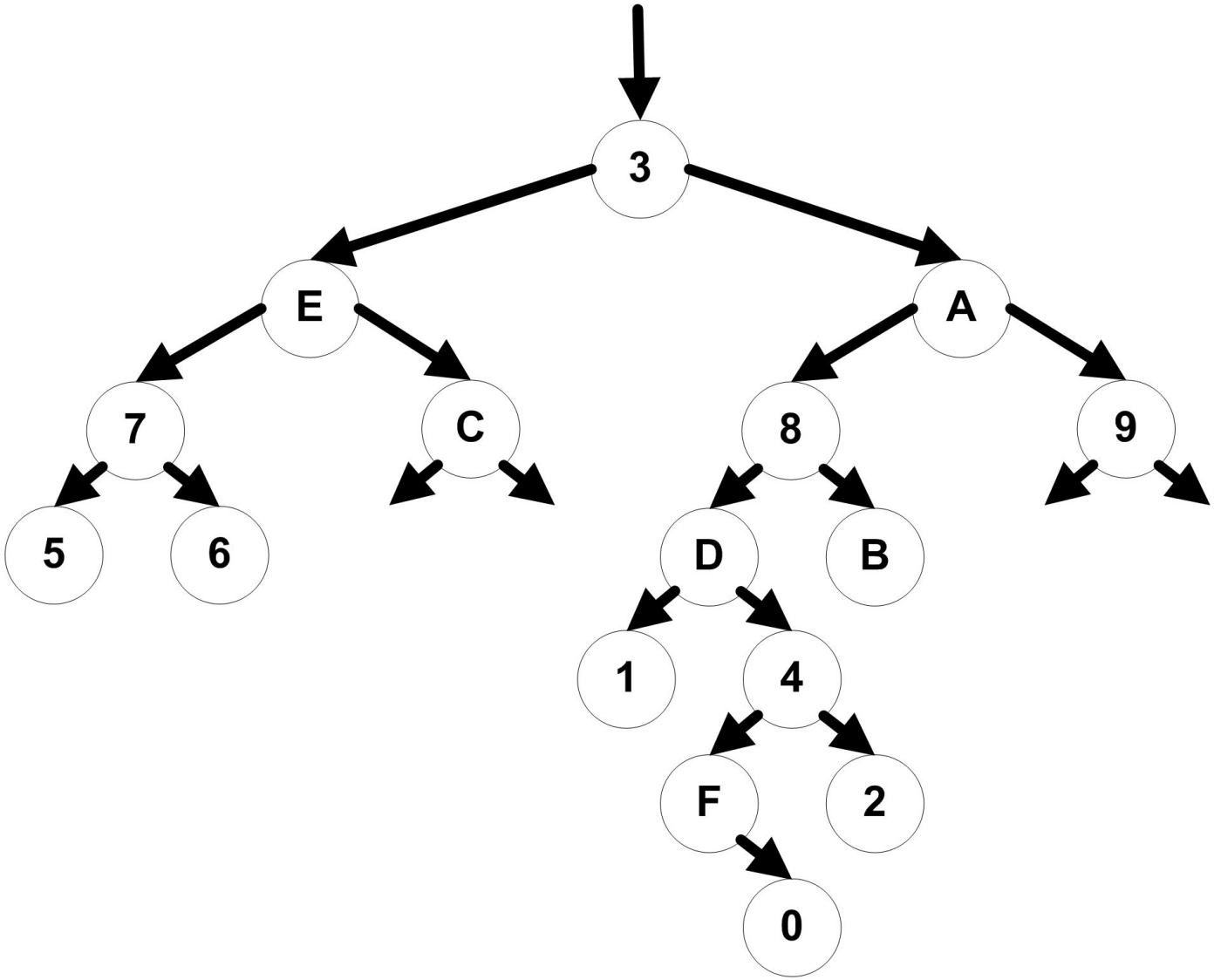


Exercício Resolvido (2)

- Crie uma árvore binária com os dígitos hexadecimais

Exercício Resolvido (2)

- Crie uma árvore binária com os dígitos hexadecimais



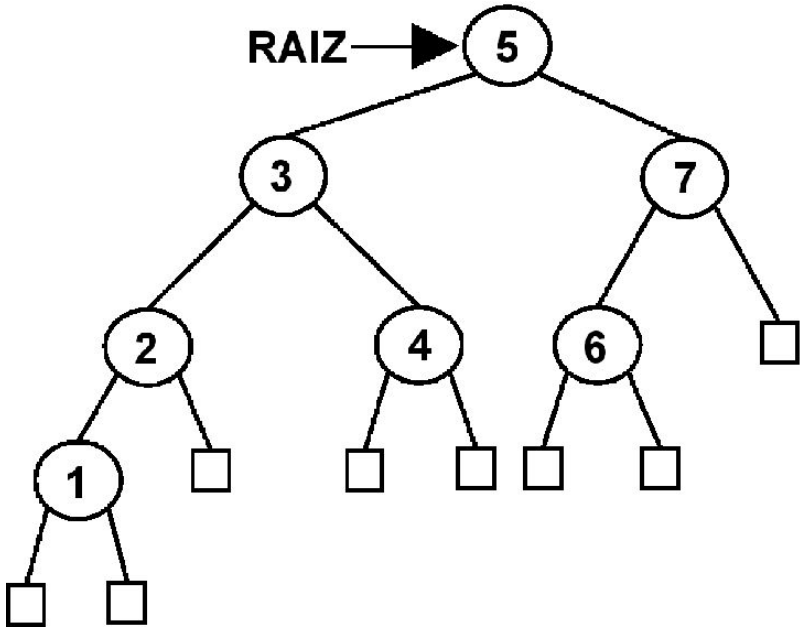
Árvore Binária de Pesquisa

- 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

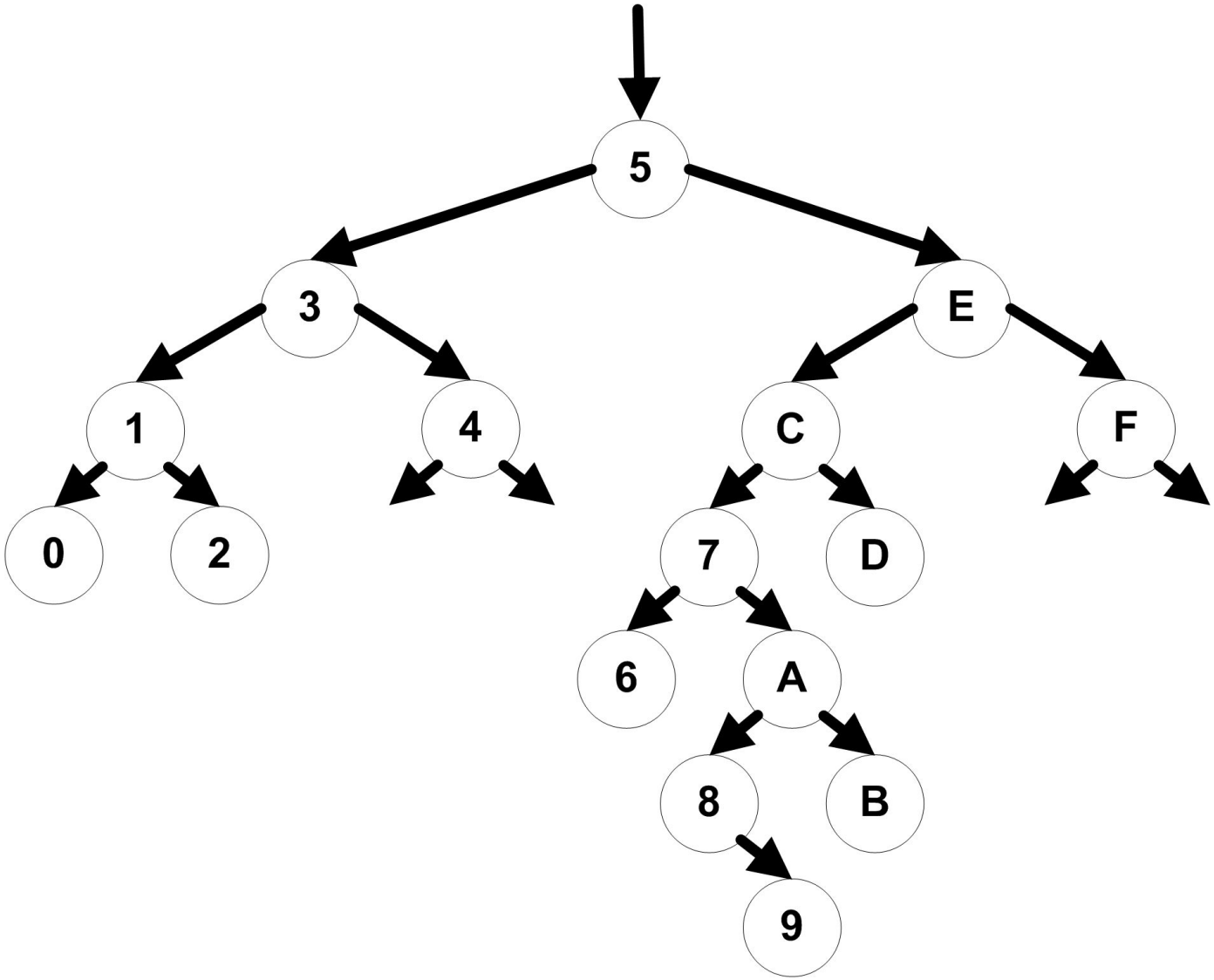


Exercício Resolvido (3)

- Crie uma árvore binária de pesquisa com os dígitos hexadecimais

Exercício Resolvido (3)

- 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 $2^h - 1 + 2^h = 2^{(h+1)} - 1$

Exercício Resolvido (4)

- Crie uma árvore binária completa com os dígitos hexadecimais

Exercício Resolvido (4)

- Crie uma árvore binária completa com os dígitos hexadecimais

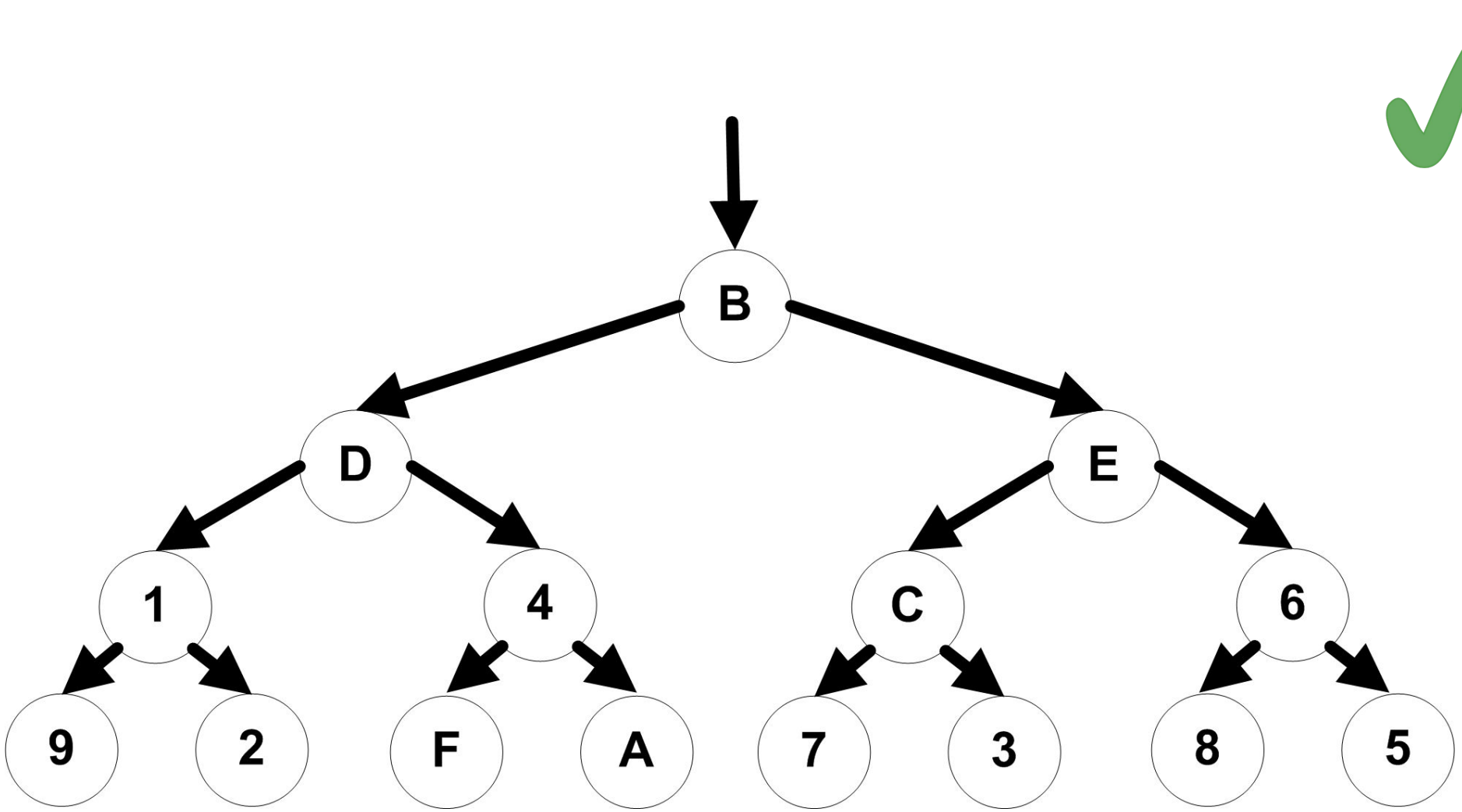
Impossível por que o número de nós não é uma potência de dois (está sobrando um nó)

Exercício Resolvido (5)

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

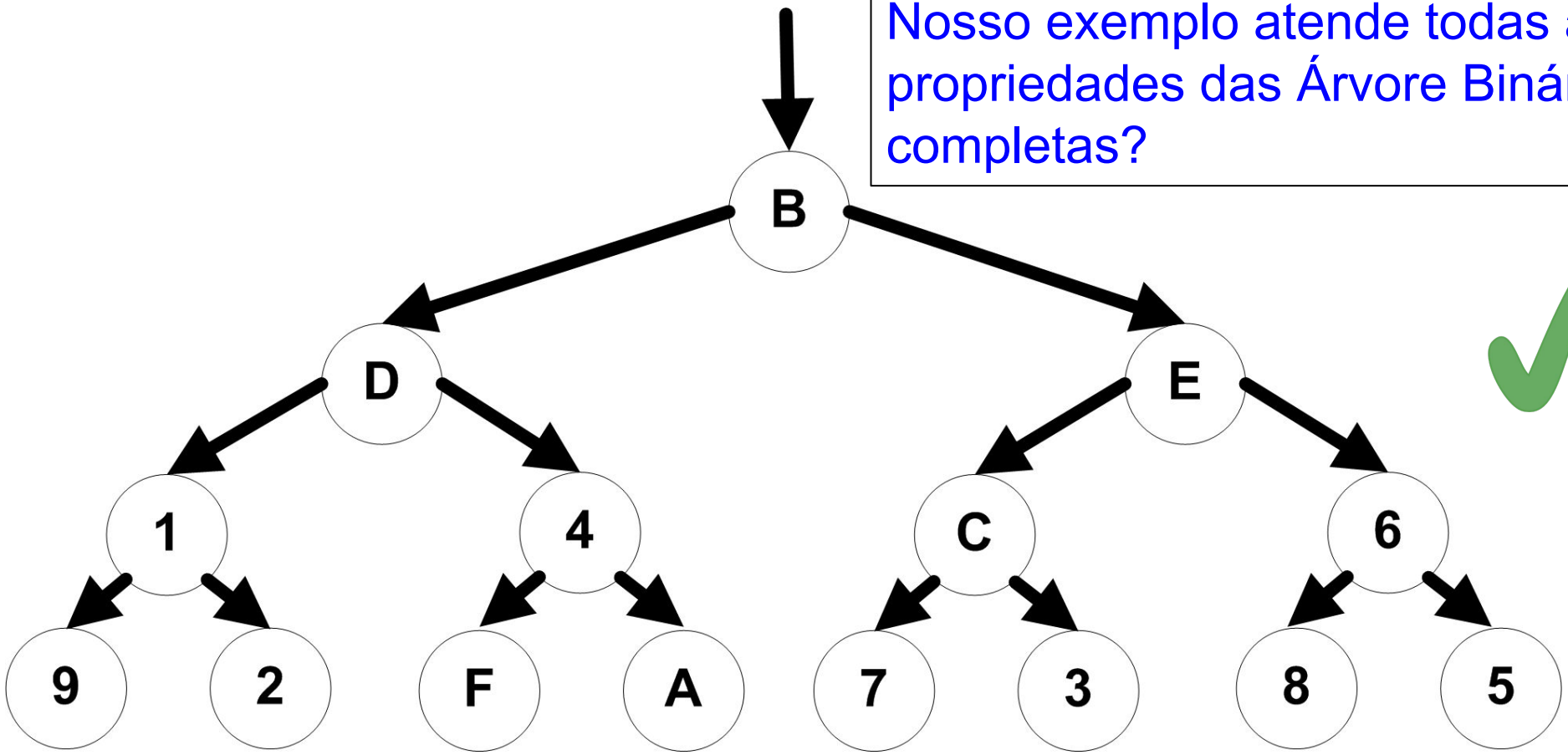
Exercício Resolvido (5)

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



Exercício Resolvido (5)

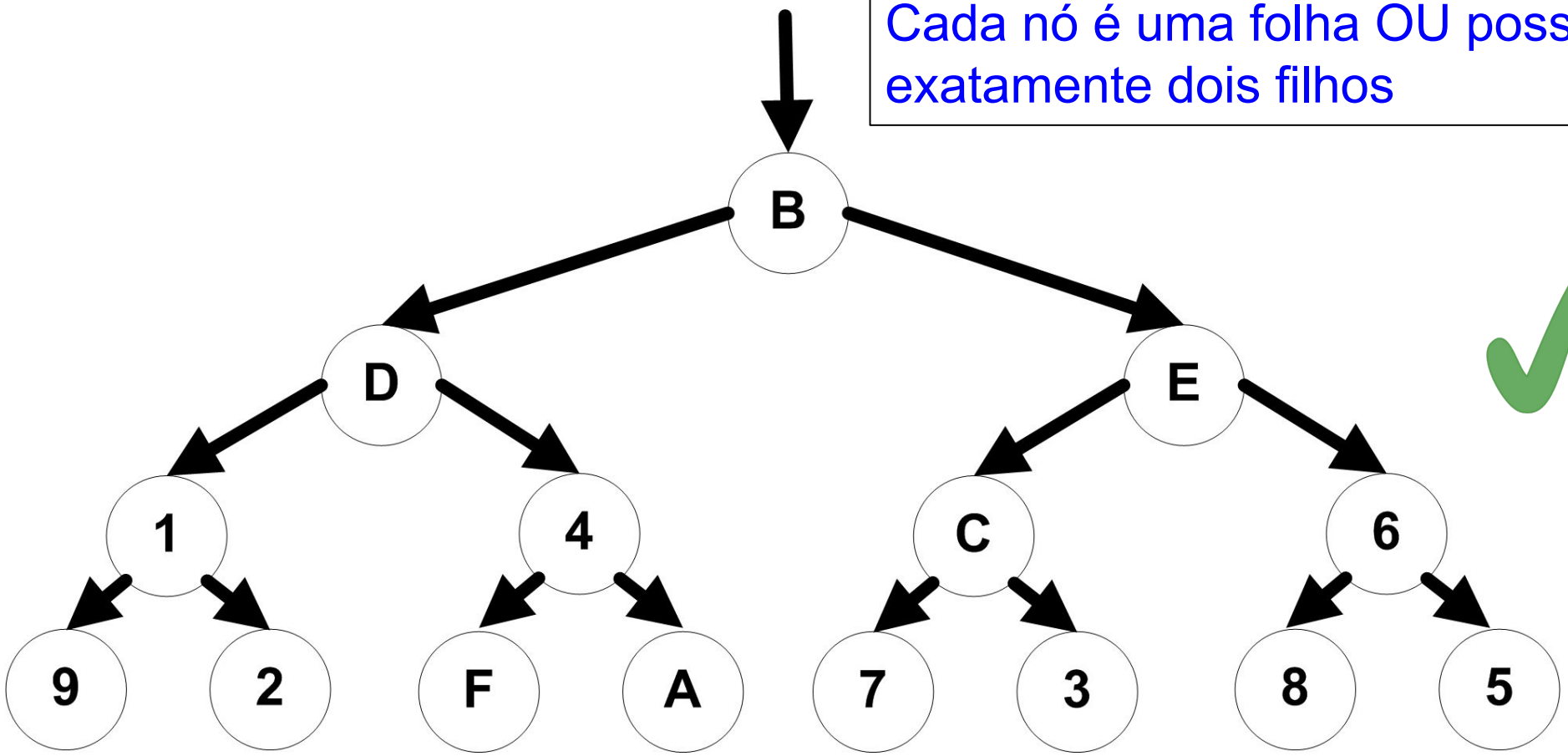
- Crie uma árvore binária completa com os dígitos hexadecimais não nulos



Exercício Resolvido (5)

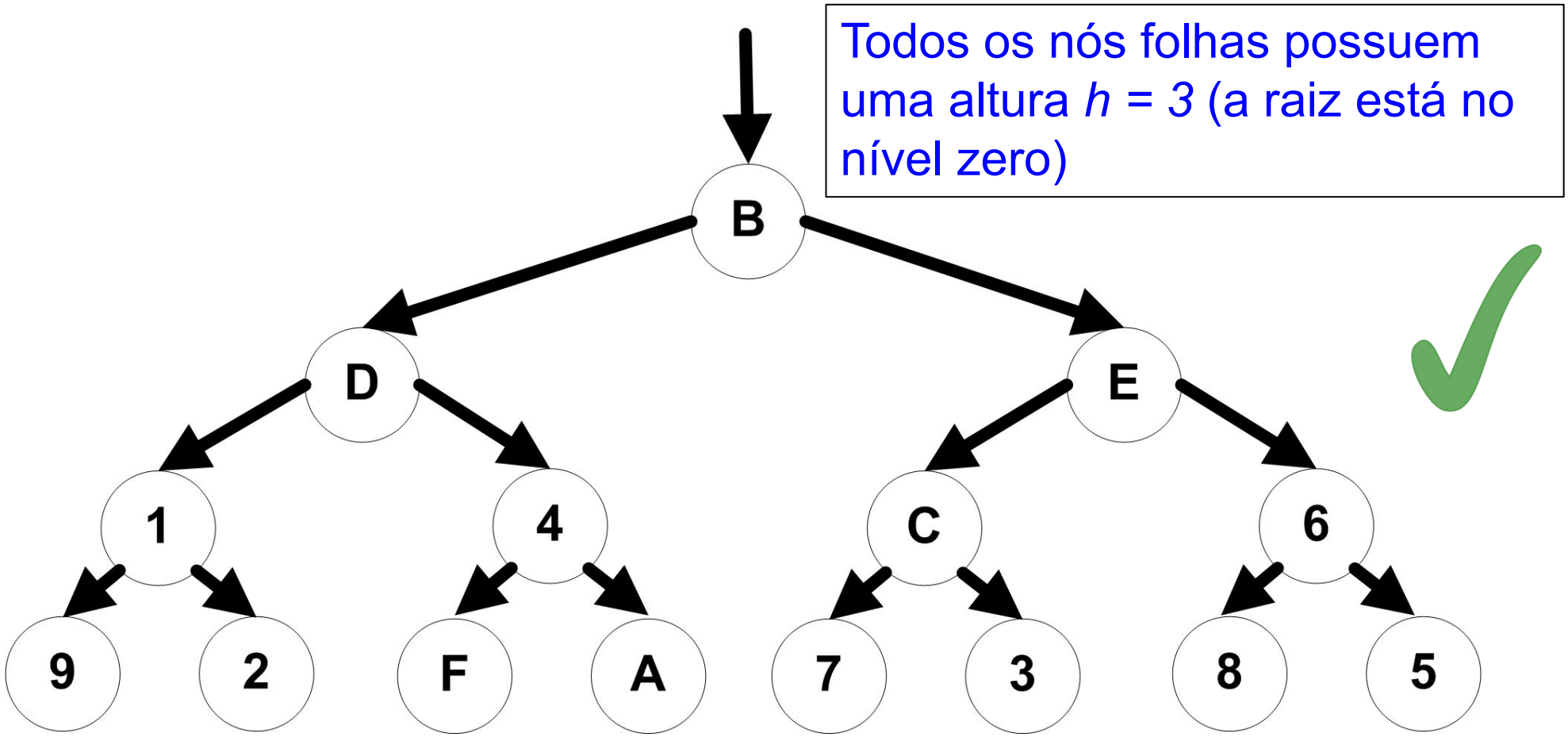
- Crie uma árvore binária completa com os dígitos hexadecimais não nulos

Cada nó é uma folha OU possui exatamente dois filhos



Exercício Resolvido (5)

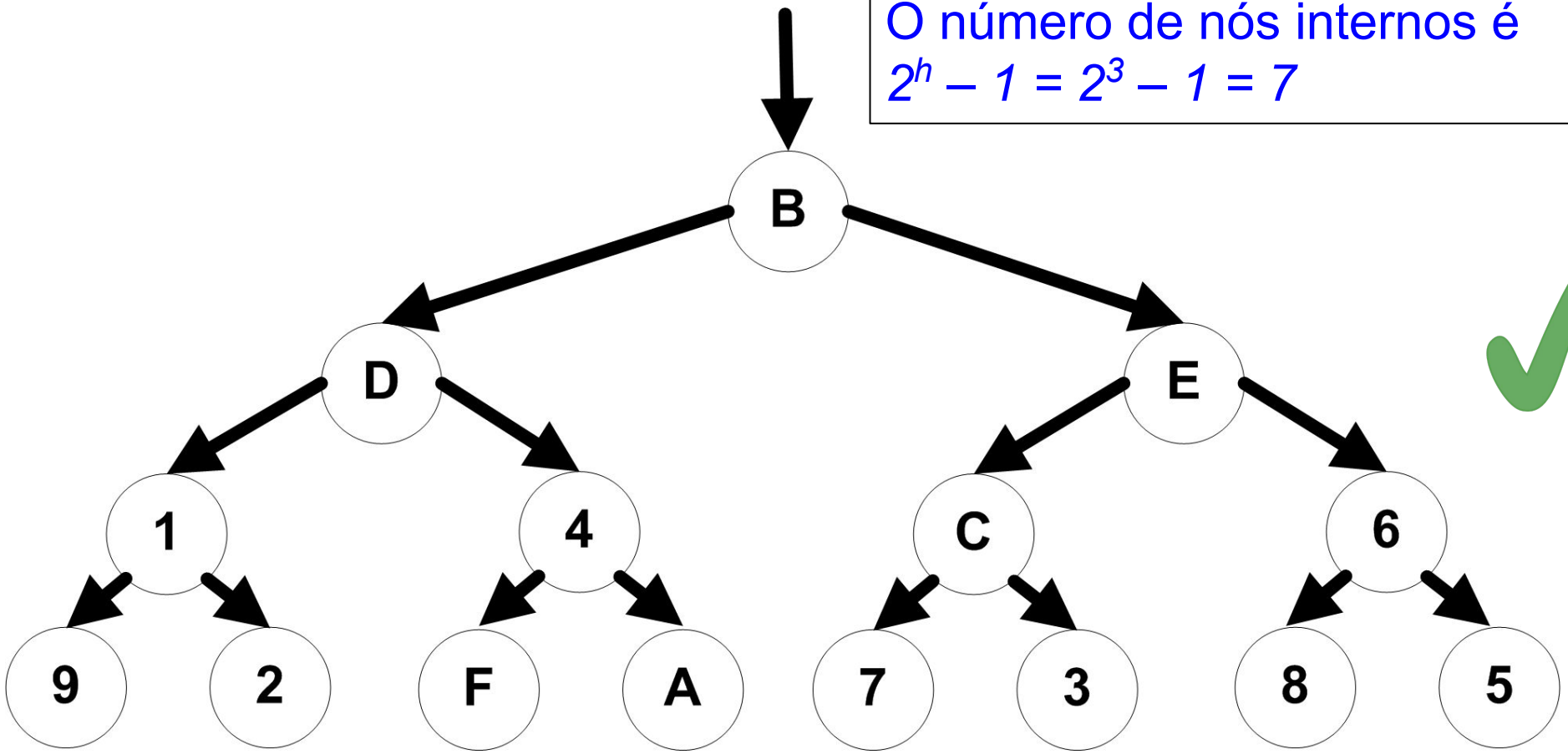
- Crie uma árvore binária completa com os dígitos hexadecimais não nulos



Exercício Resolvido (5)

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

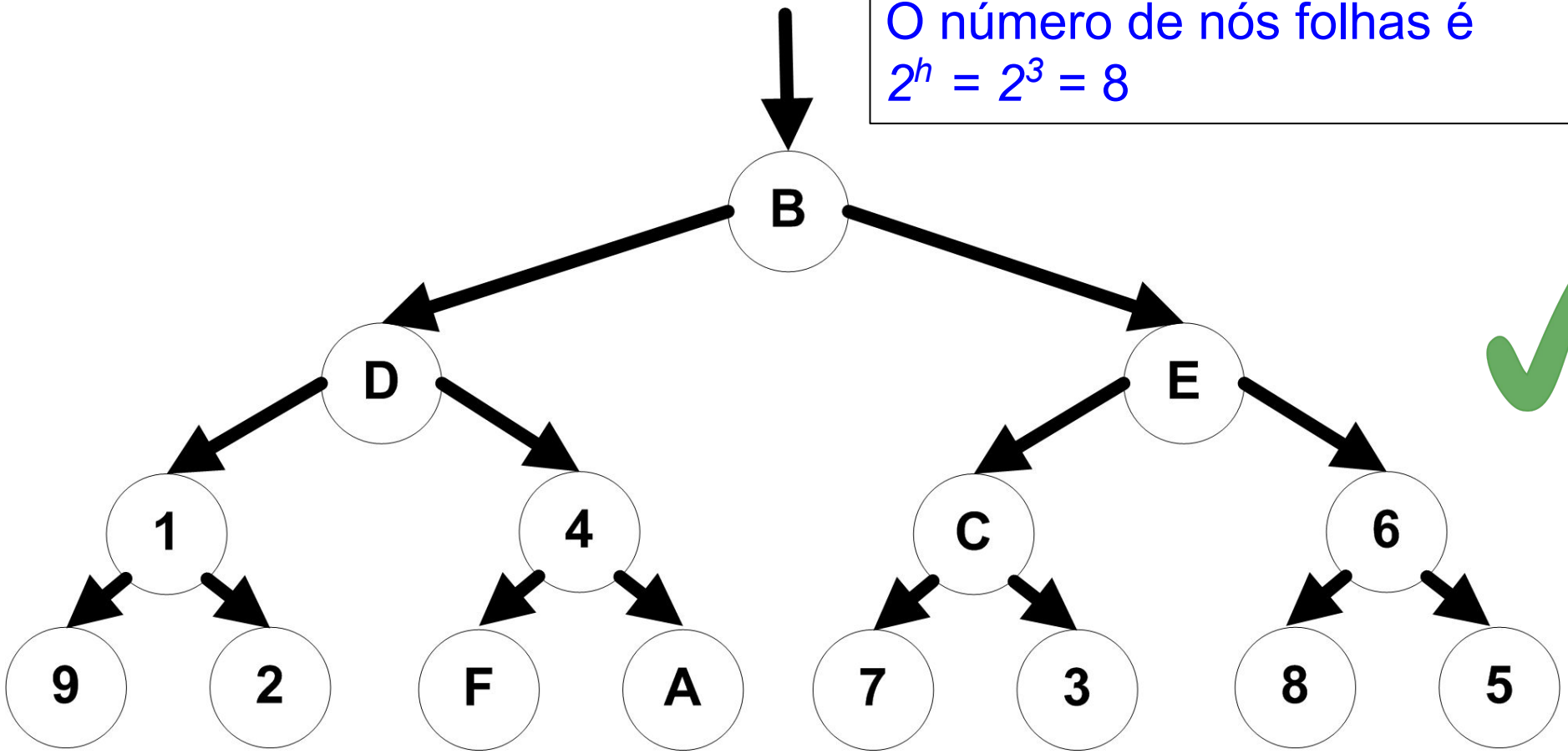
O número de nós internos é
$2^h - 1 = 2^3 - 1 = 7$



Exercício Resolvido (5)

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

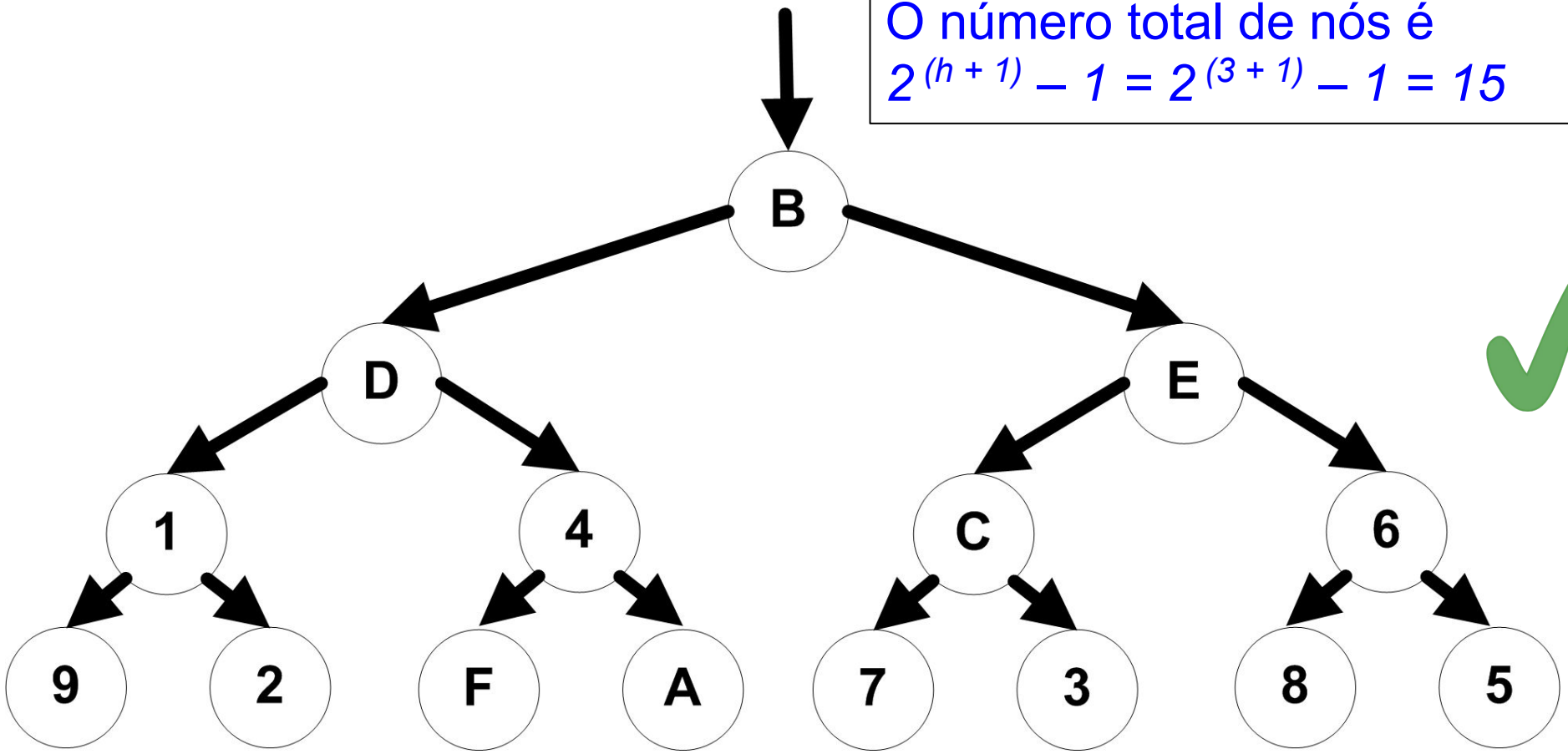
O número de nós folhas é
$2^h = 2^3 = 8$



Exercício Resolvido (5)

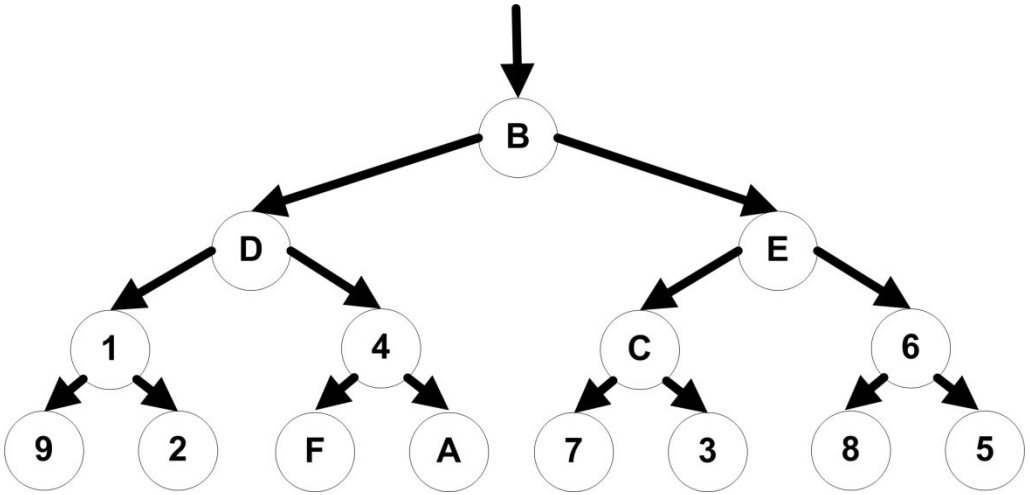
- Crie uma árvore binária completa com os dígitos hexadecimais não nulos

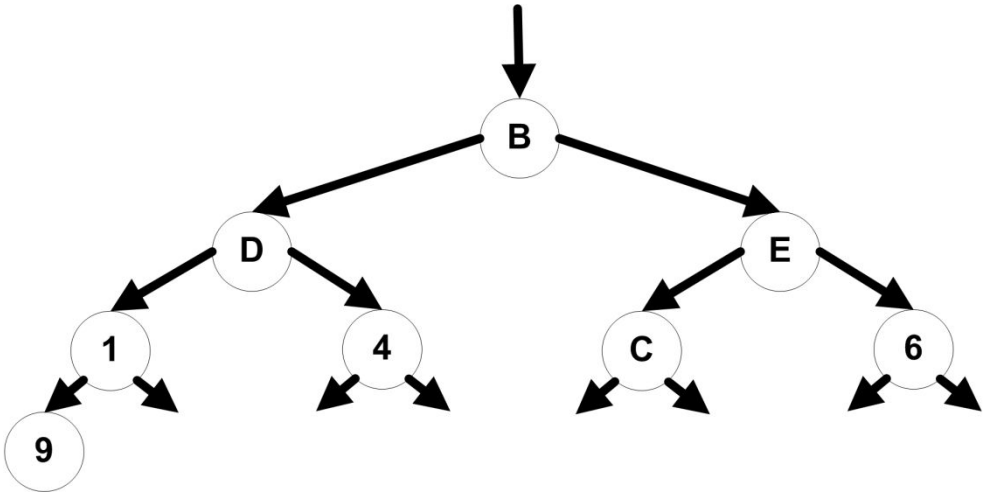
O número total de nós é
$2^{(h+1)} - 1 = 2^{(3+1)} - 1 = 15$

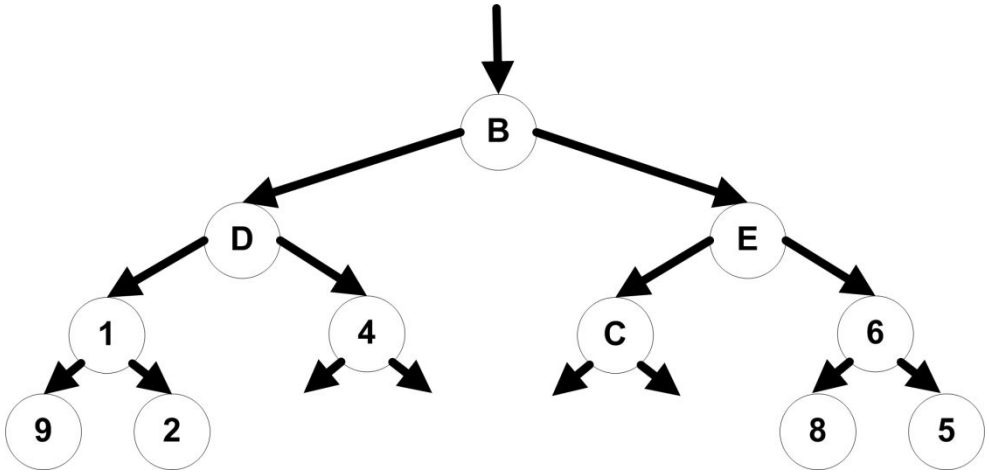


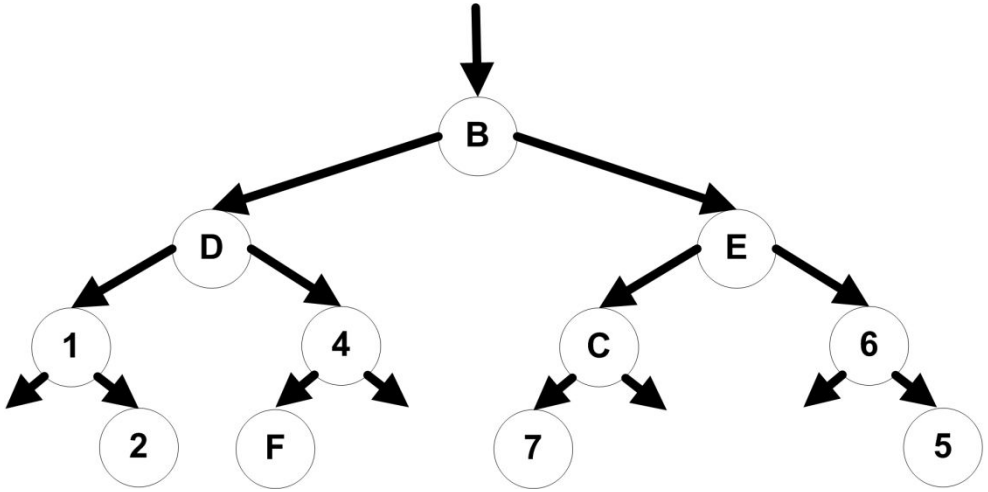
Árvore Balanceada

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







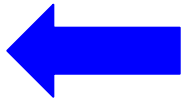


Consideração

- As Árvores Binárias de Pesquisa também são chamadas de Árvores Binárias de Busca

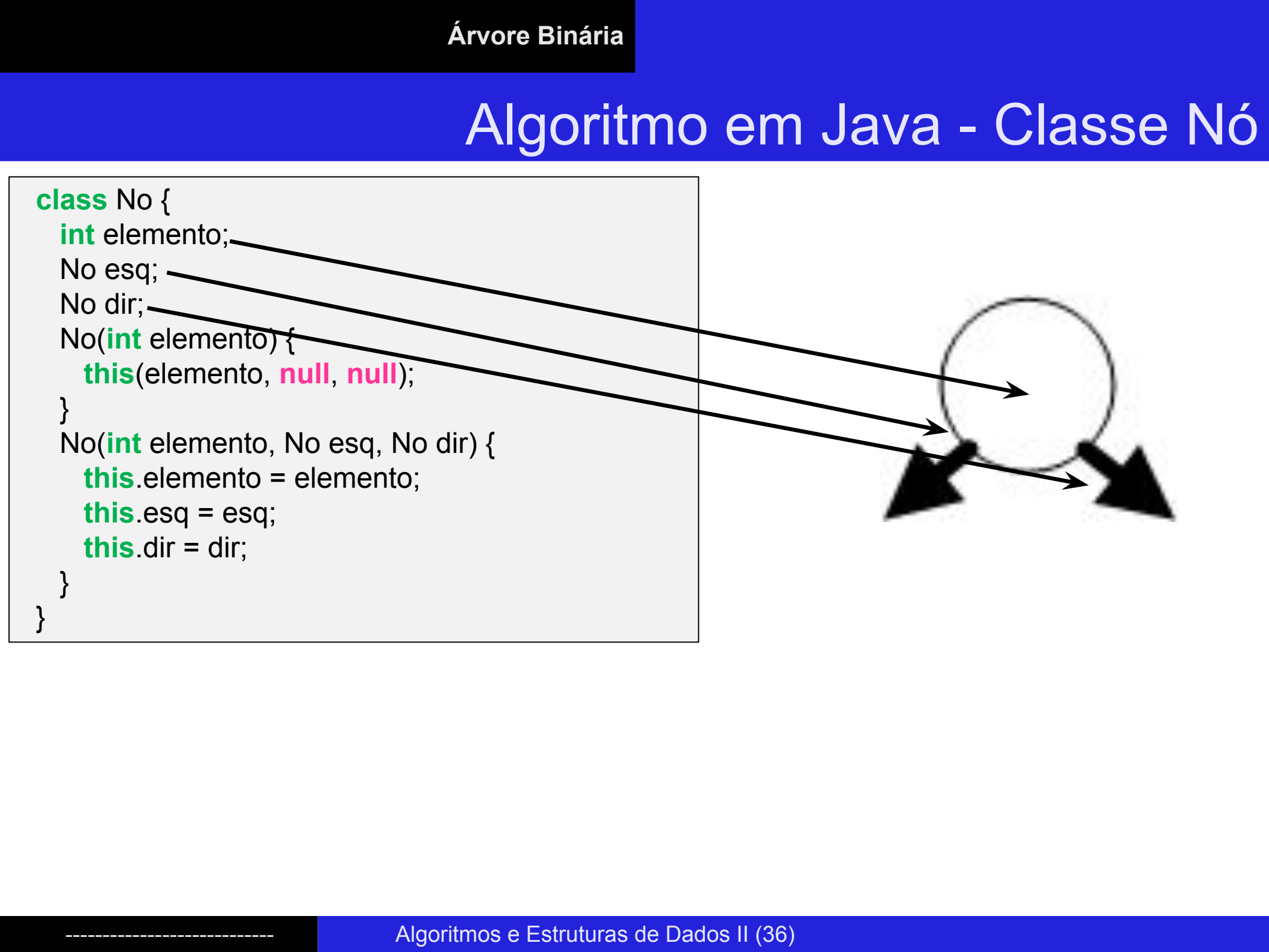
- A partir deste ponto, considera-se que todas as árvores binárias serão de pesquisa

Agenda

- Definições e conceitos
- **Classes Nó e Árvore Binária em Java** 
- Inserção
- Pesquisa
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

Algoritmo em Java - Classe Nó

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

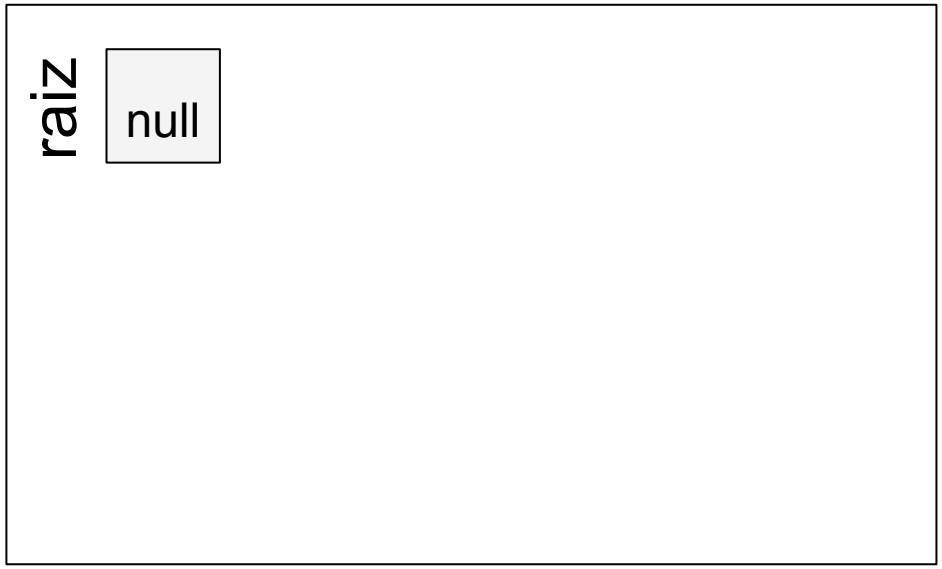


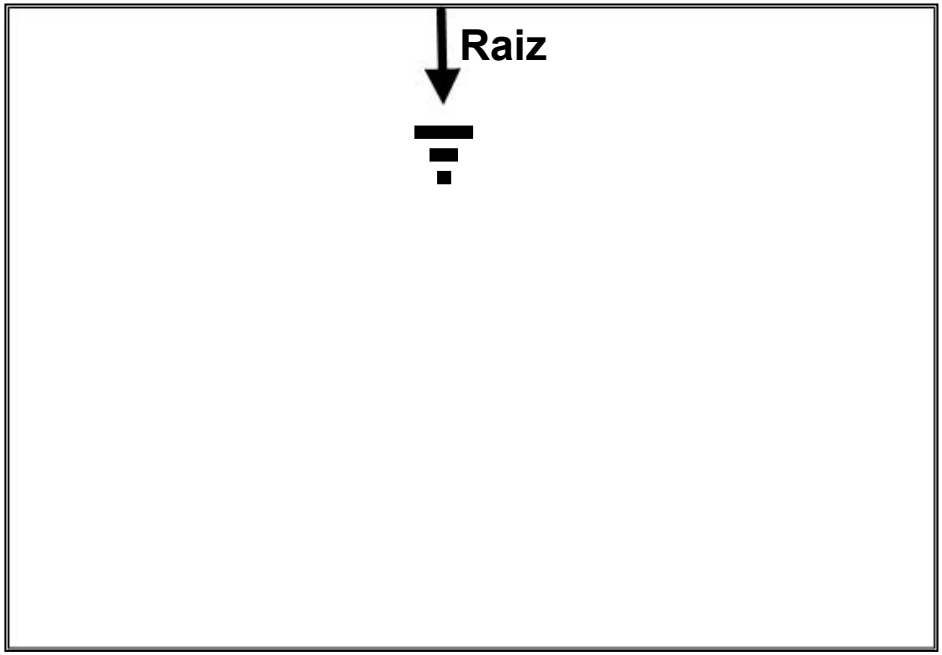
Classe Árvore Binária em Java

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    void inserirPai(int x) {  }
    boolean pesquisar(int x) {   }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
    void remover(int x) {  }
}
```

Classe Árvore Binária em Java

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    void inserirPai(int x) {  }
    boolean pesquisar(int x) {   }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
    void remover(int x) {  }
}
```





Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- **Inserção**
- Pesquisa
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passa
- Estruturas híbridas

- **Funcionamento básico**
- Exemplo
- Inserção em Java com retorno de referência
- Inserção em Java com passagem de pai
- Análise de Complexidade

Funcionamento Básico da Inserção

(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

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- **Inserção**
- Pesquisa
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passa
- Estruturas híbridas

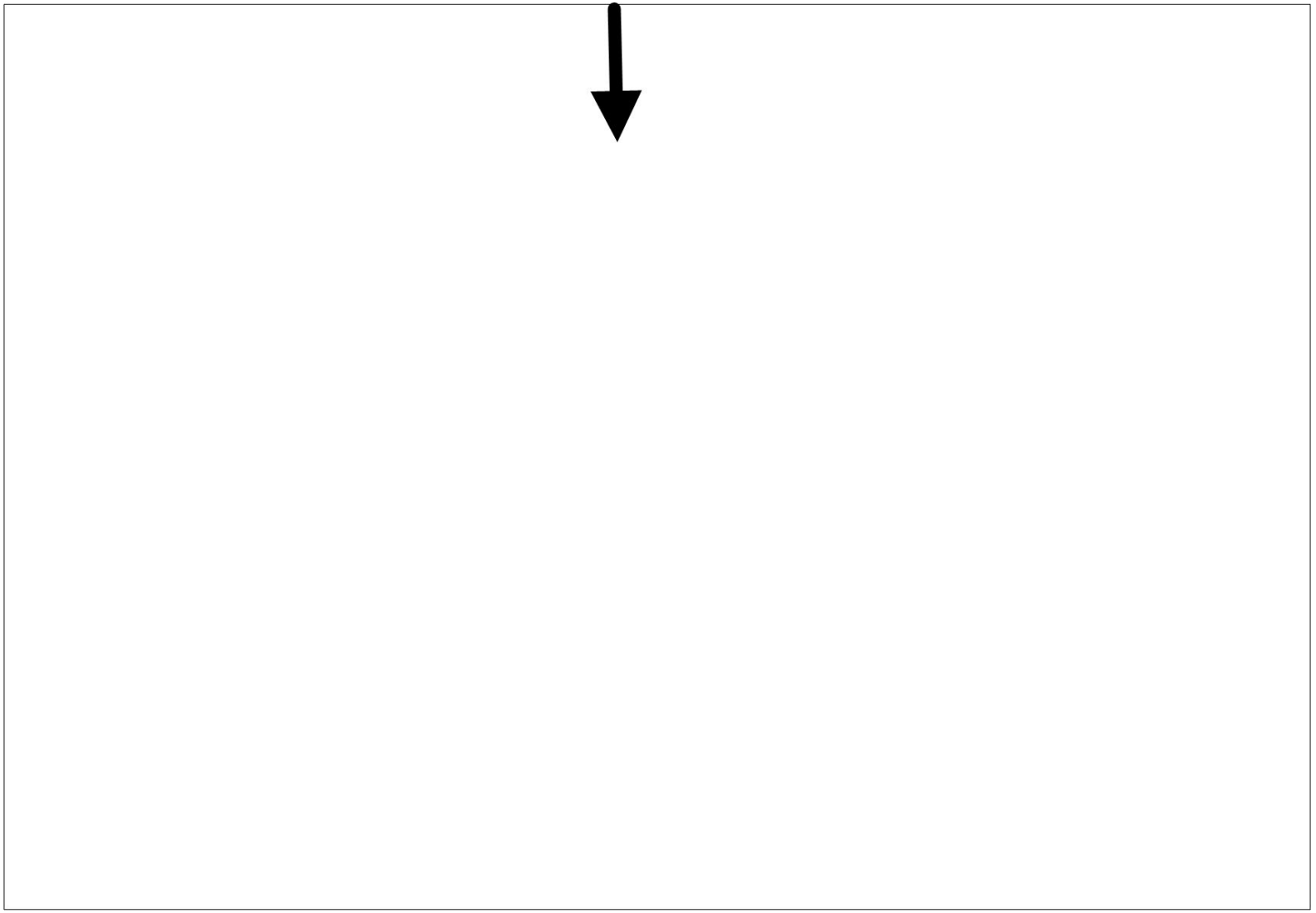
  - Funcionamento básico
  - **Exemplo**
  - Inserção em Java com retorno de referência
  - Inserção em Java com passagem de pai
  - Análise de Complexidade

Exemplo

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

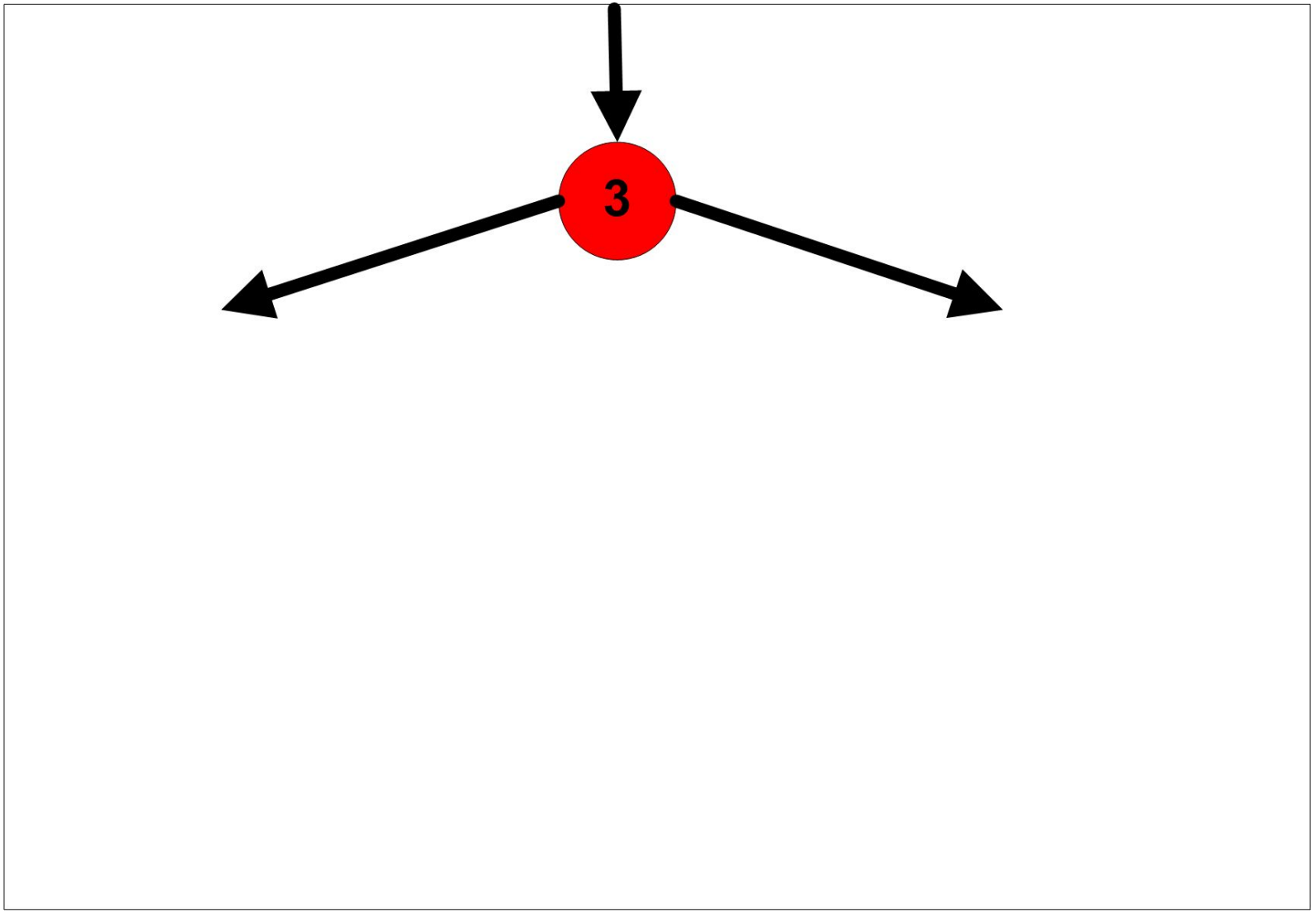
Exemplo

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



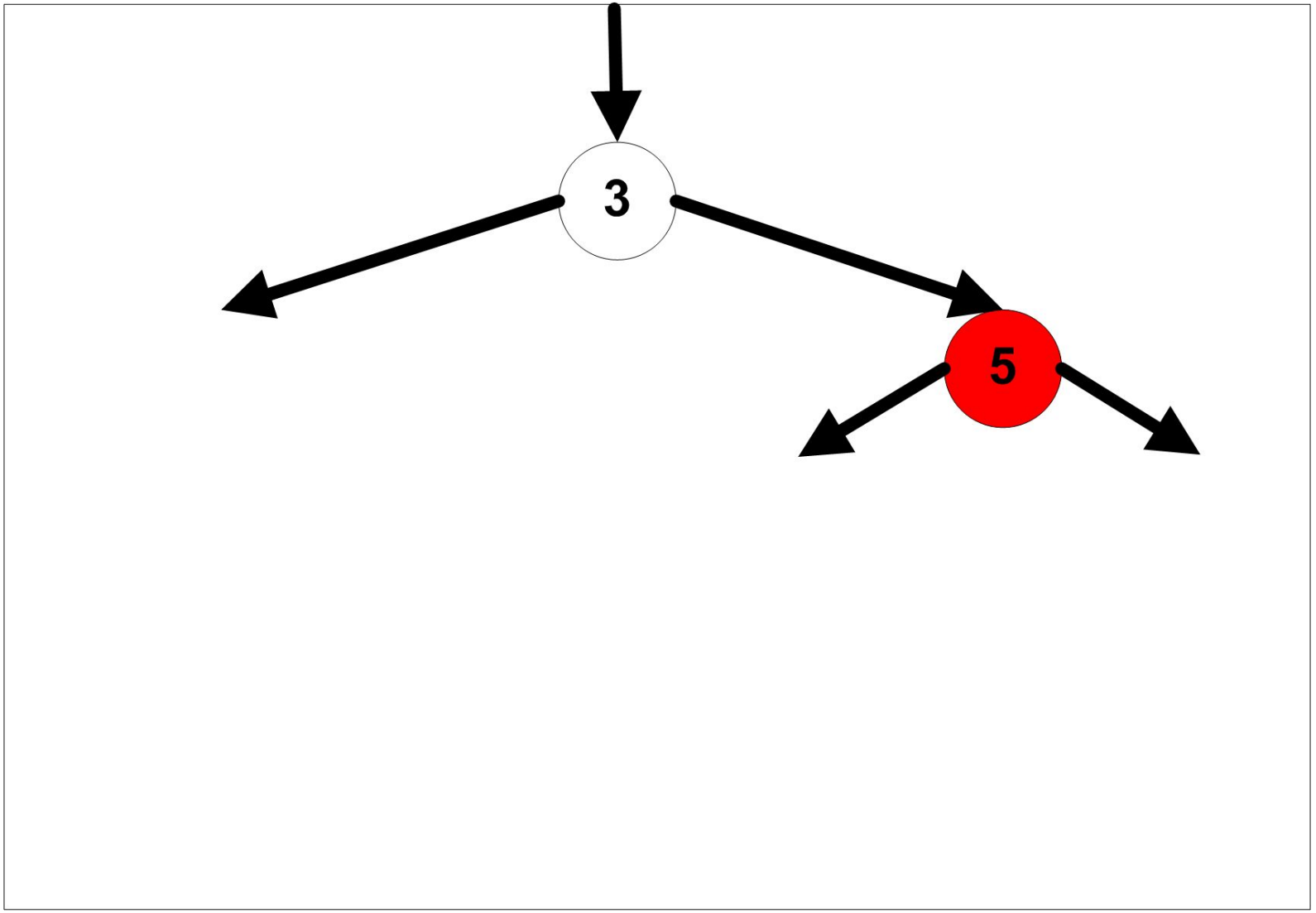
Exemplo

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



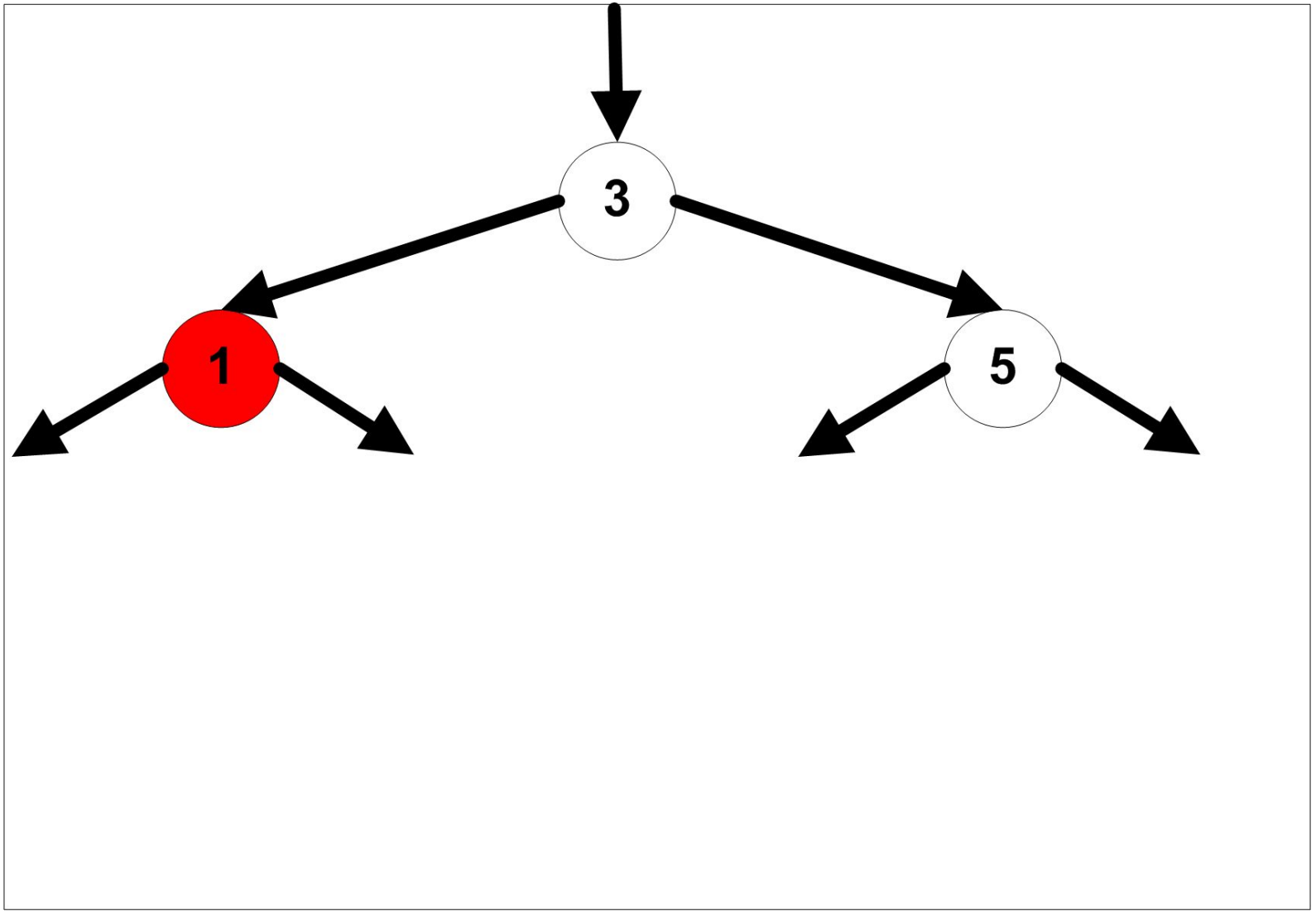
Exemplo

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



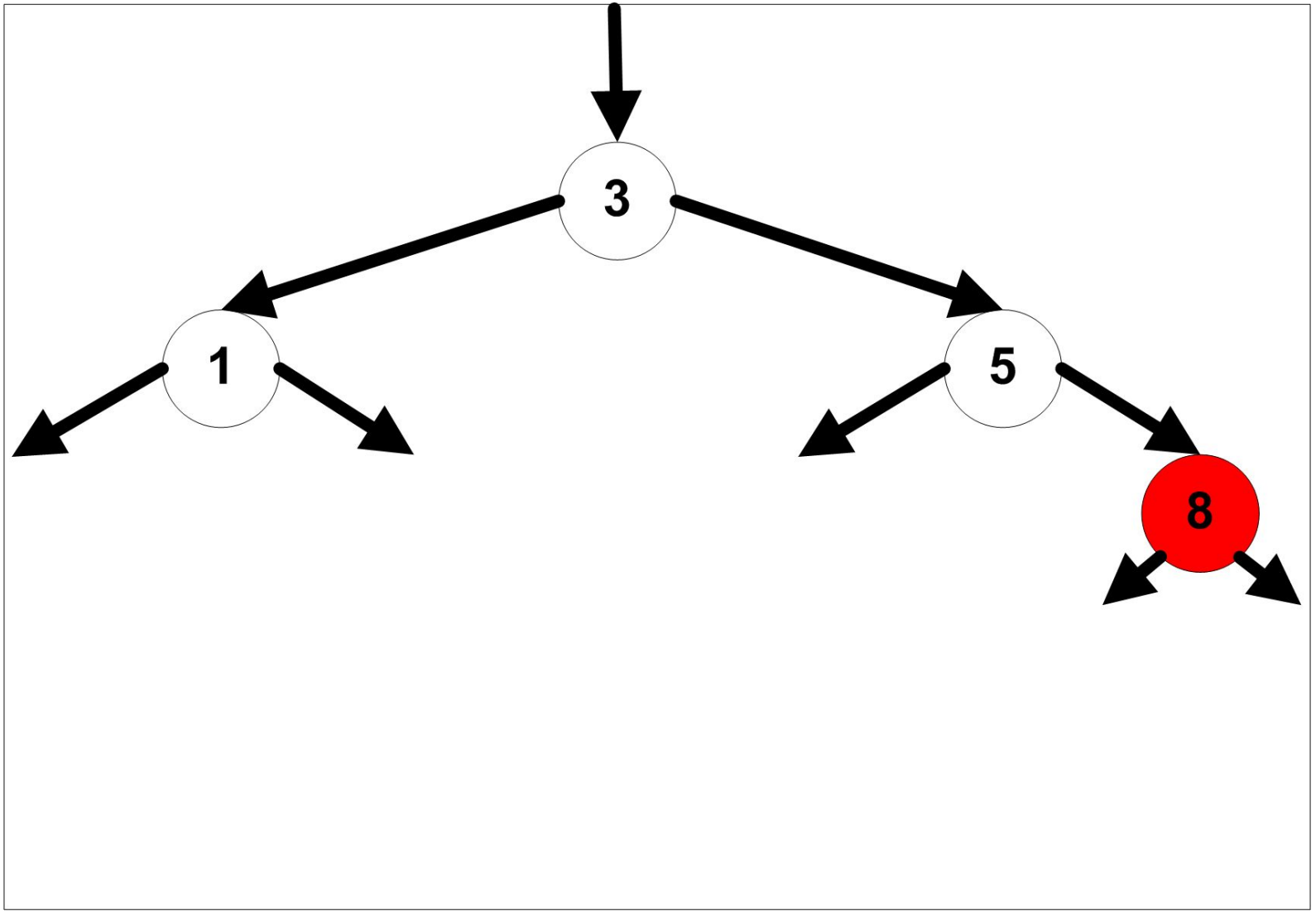
Exemplo

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



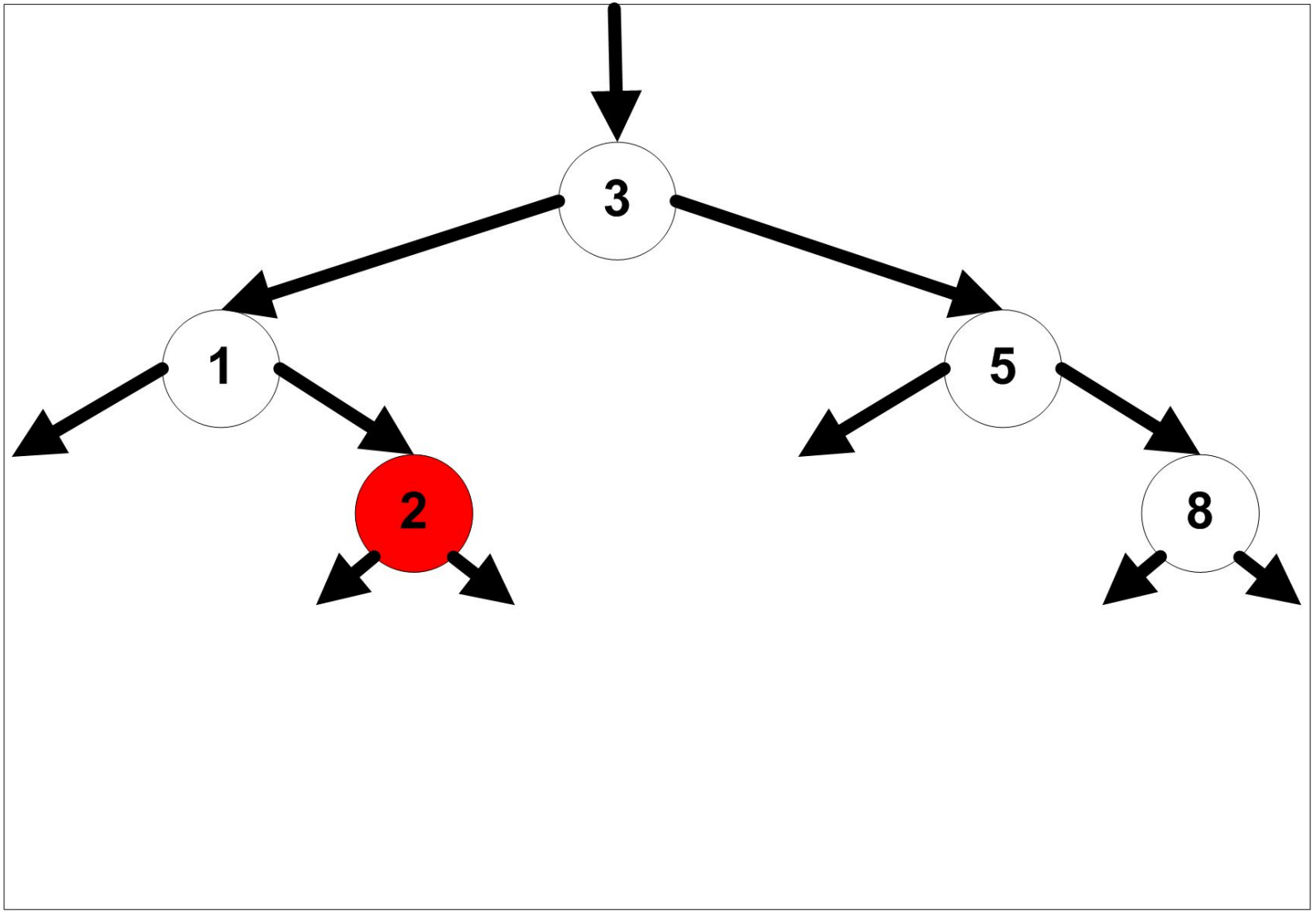
Exemplo

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



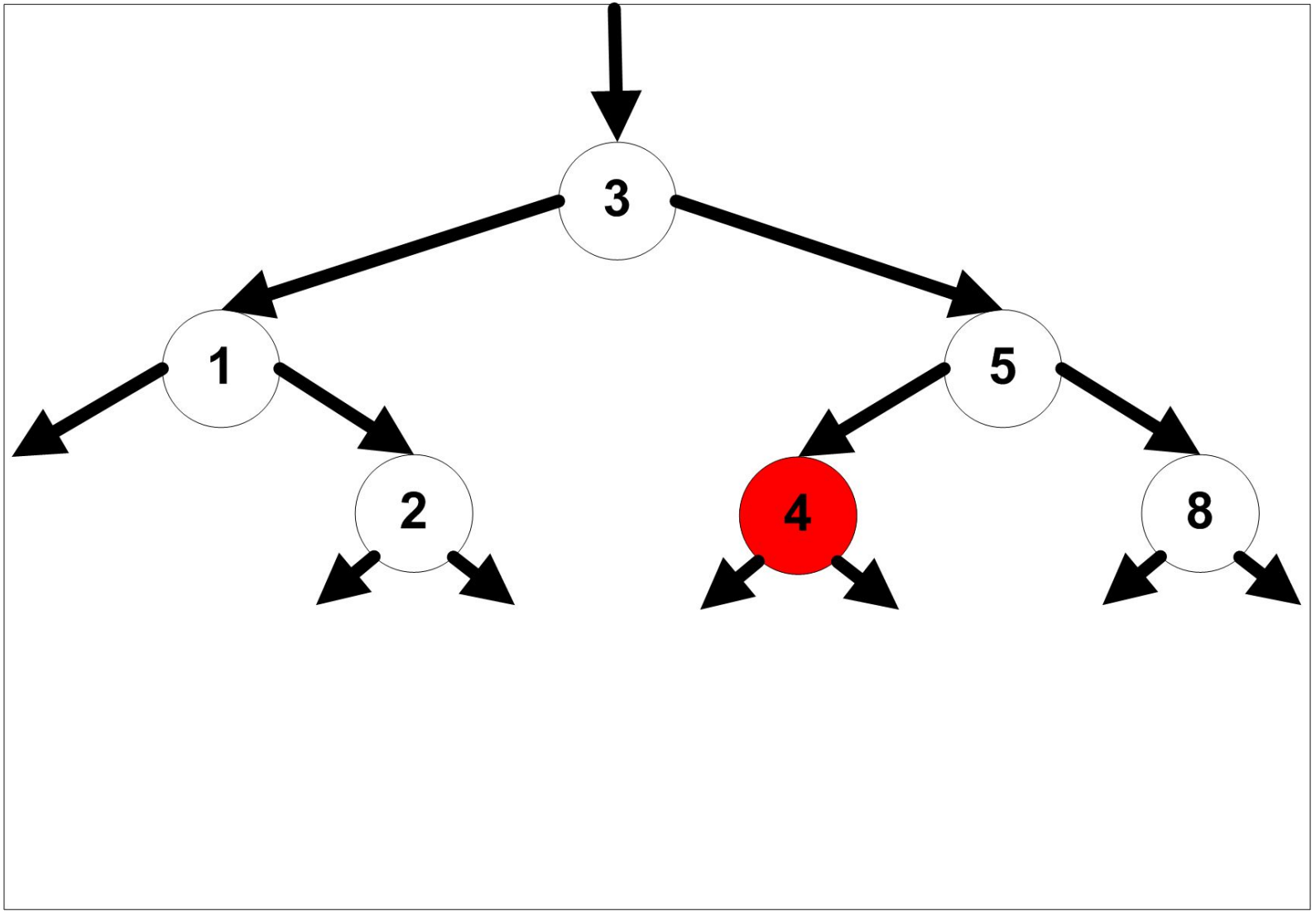
Exemplo

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



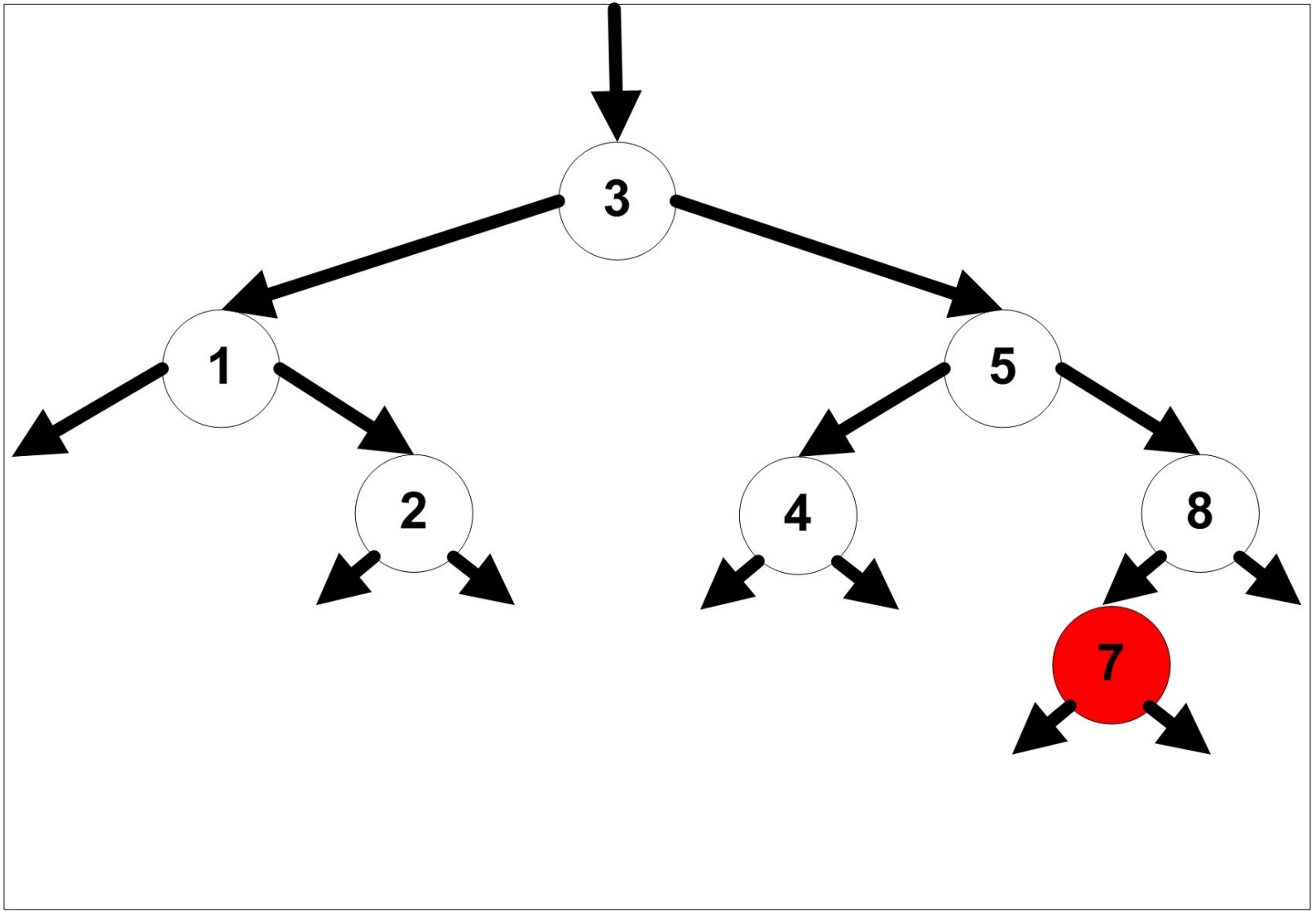
Exemplo

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



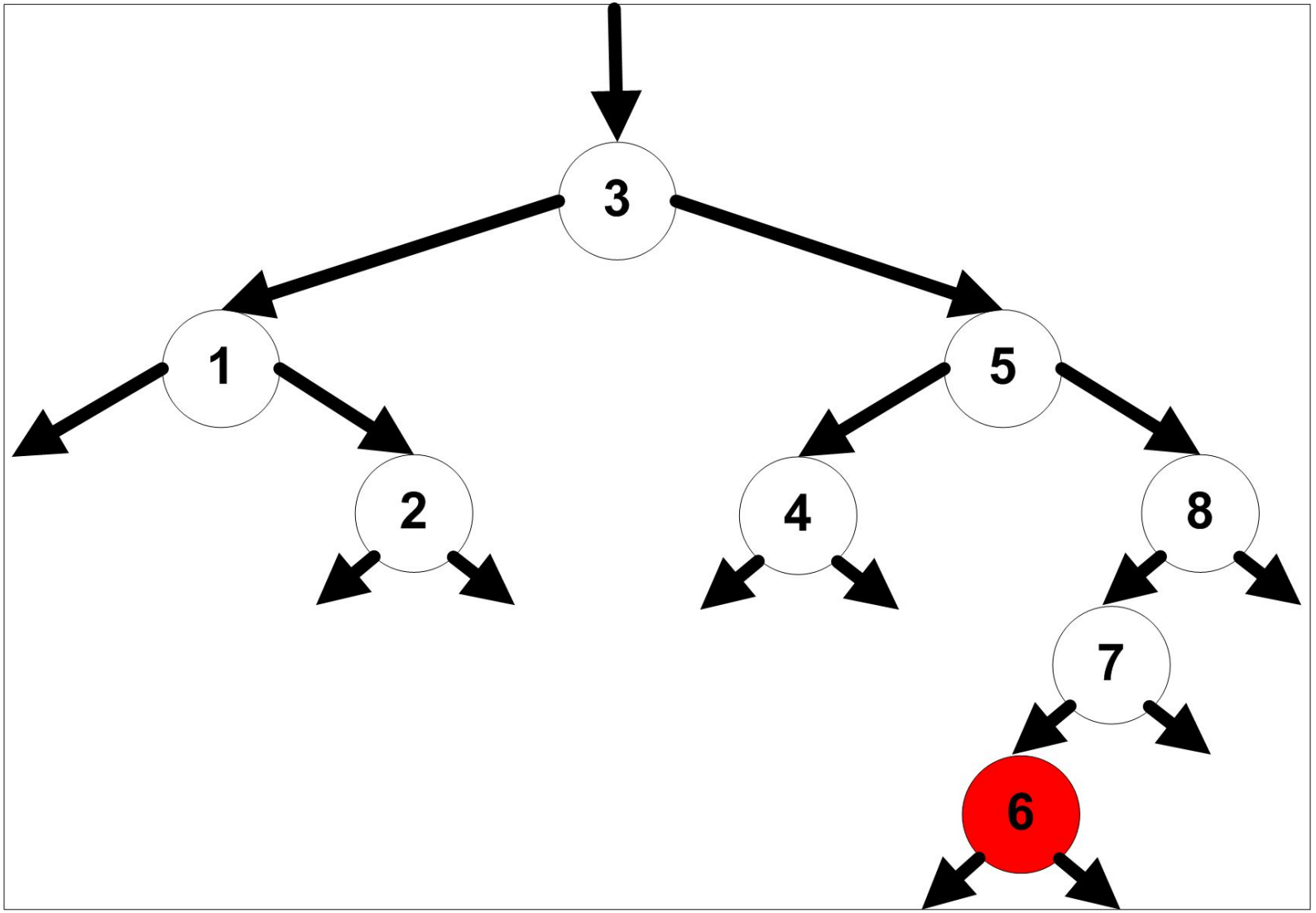
Exemplo

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



Exemplo

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



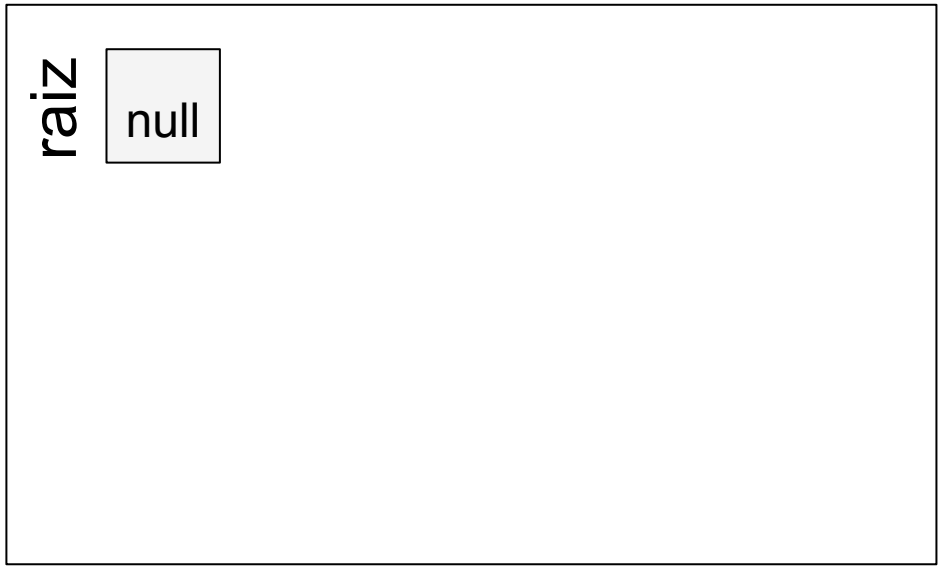
Agenda

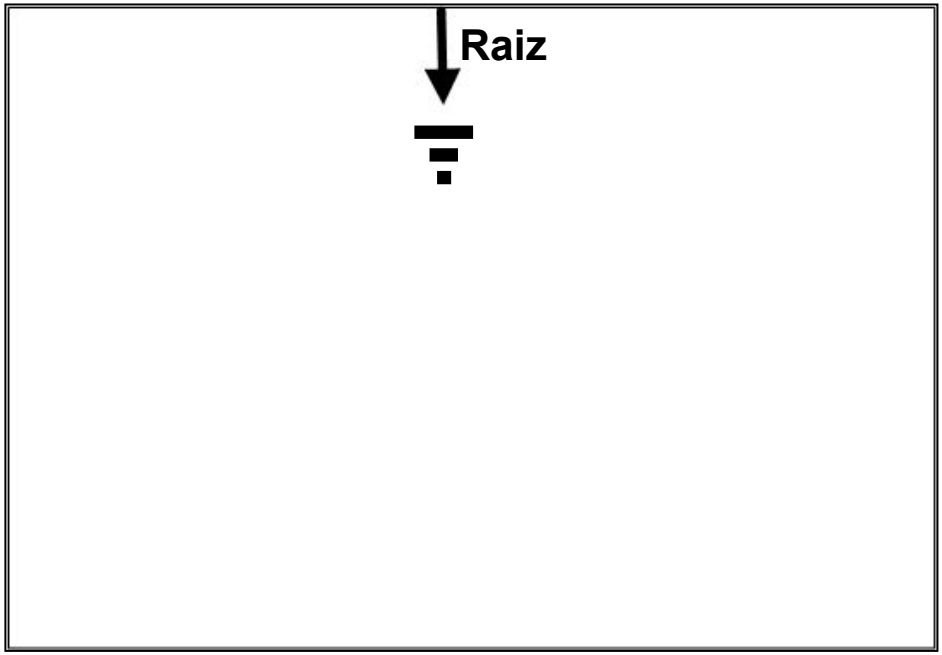
- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- **Inserção**
- Pesquisa
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passa
- Estruturas híbridas

  - Funcionamento básico
  - Exemplo
  - **Inserção em Java com retorno de referência**
  - Inserção em Java com passagem de pai
  - Análise de Complexidade

Inserção em Java com Retorno de Referência

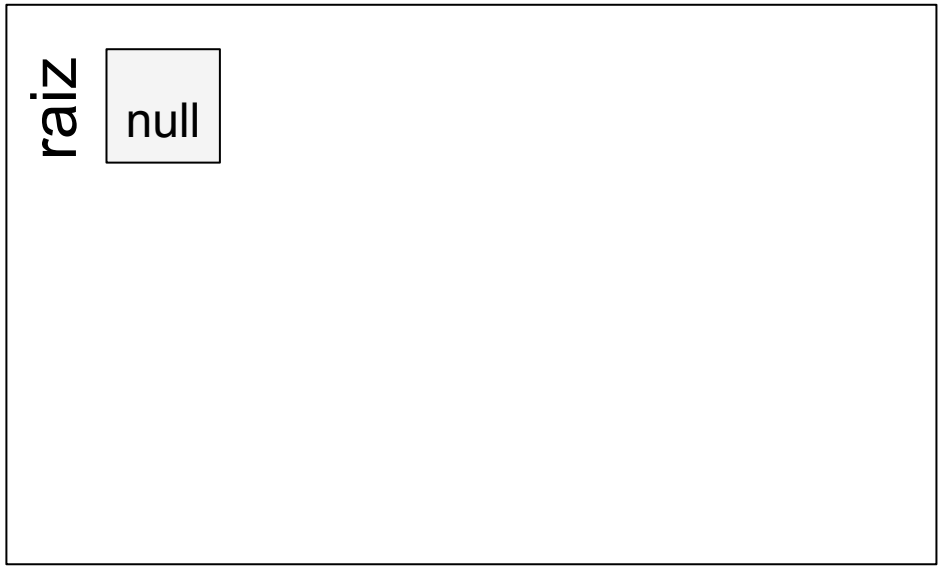
```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    void inserirPai(int x) {  }
    boolean pesquisar(int x) {   }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
    void remover(int x) {  }
}
```

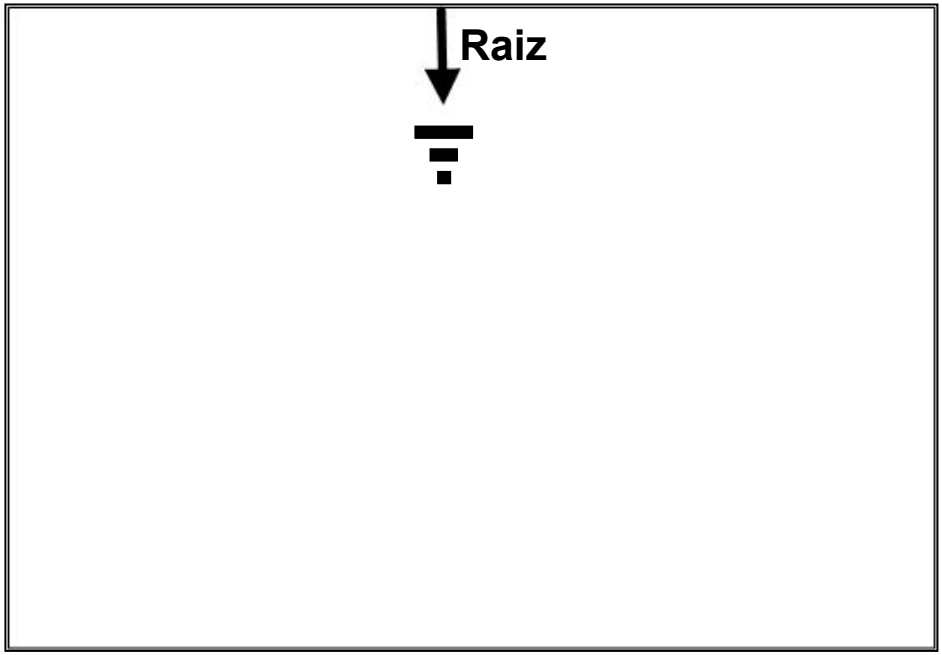




Inserção em Java com Retorno de Referência

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    void inserirPai(int x) {  }
    boolean pesquisar(int x) {   }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
    void remover(int x) {  }
}
```

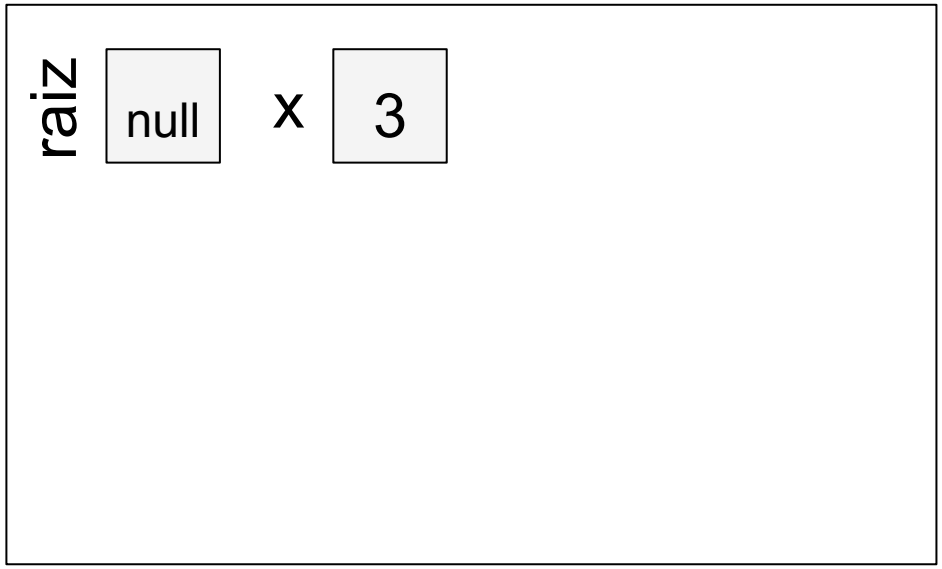


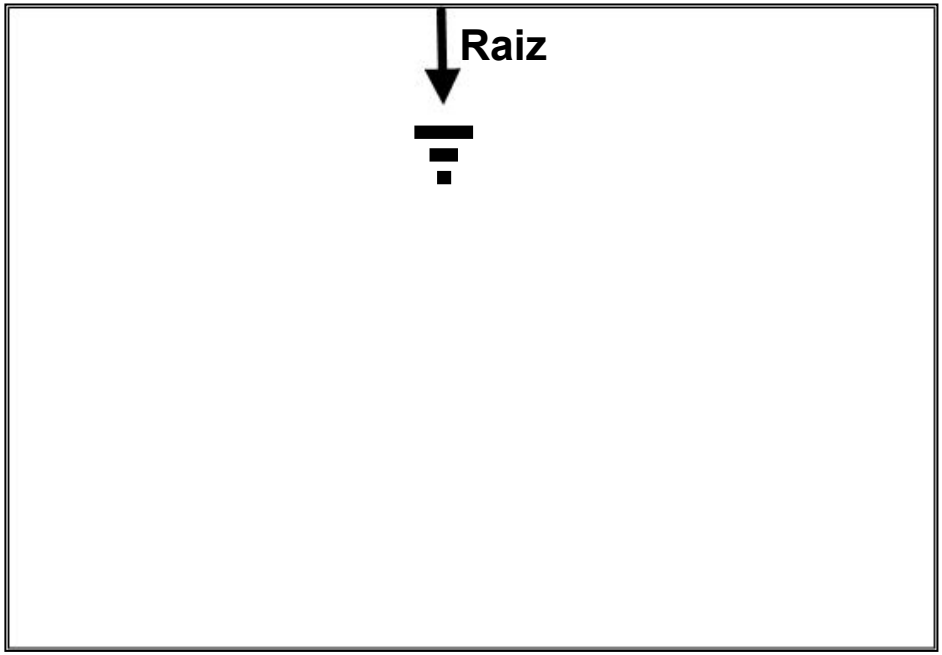


Vamos inserir os elementos
3, 5, 1, 8, 2, 4, 7 e 6
(várias chamadas do inserir)

Inserção em Java com Retorno de Referência

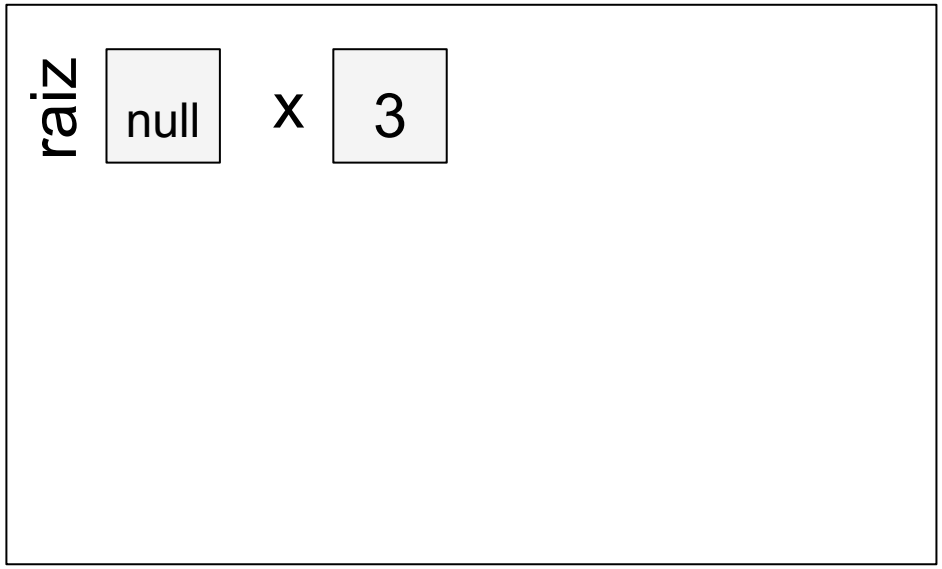
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

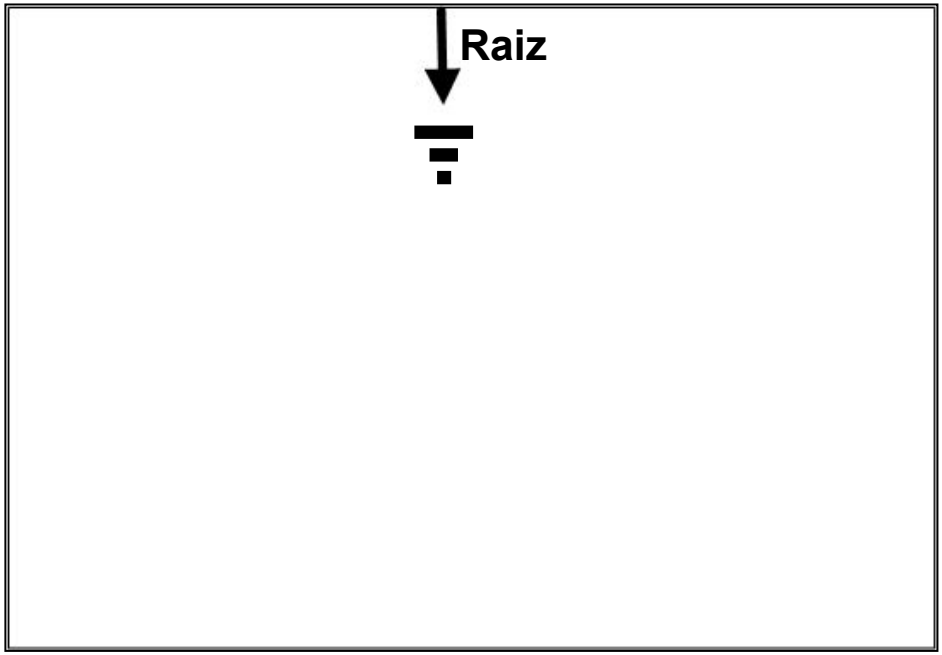




Inserção em Java com Retorno de Referência

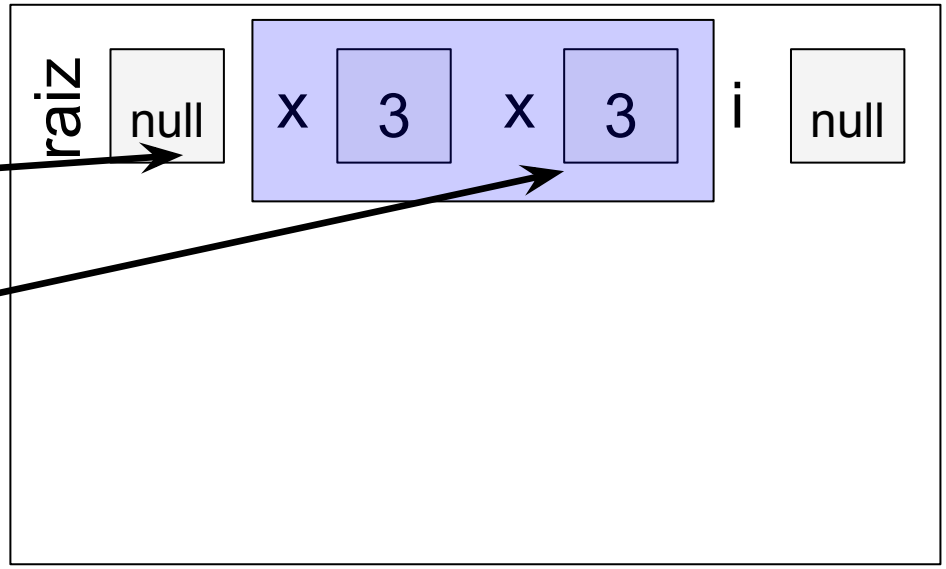
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

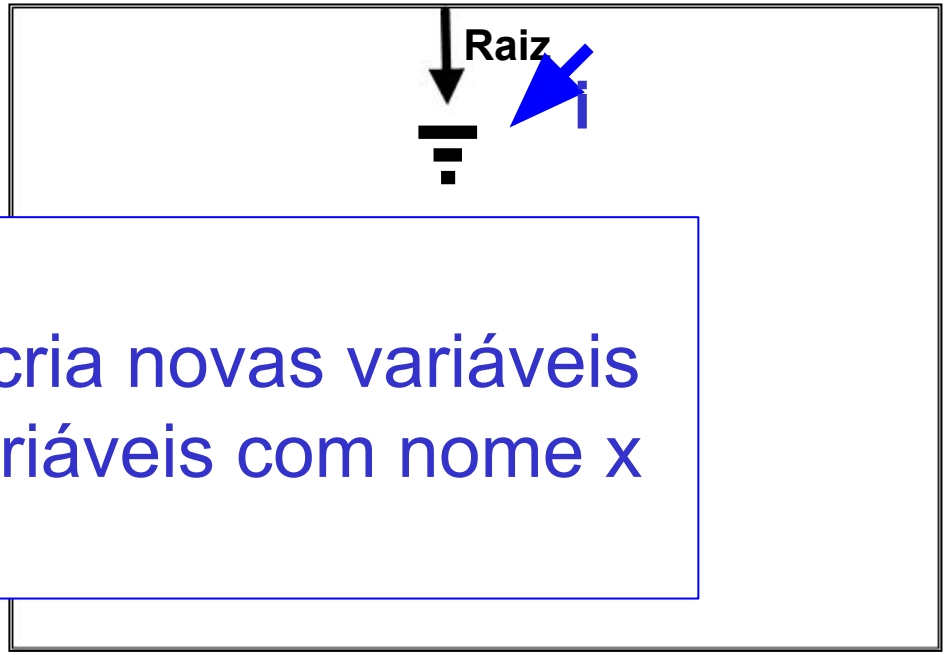




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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 {
      thro
   }
   return
}
```

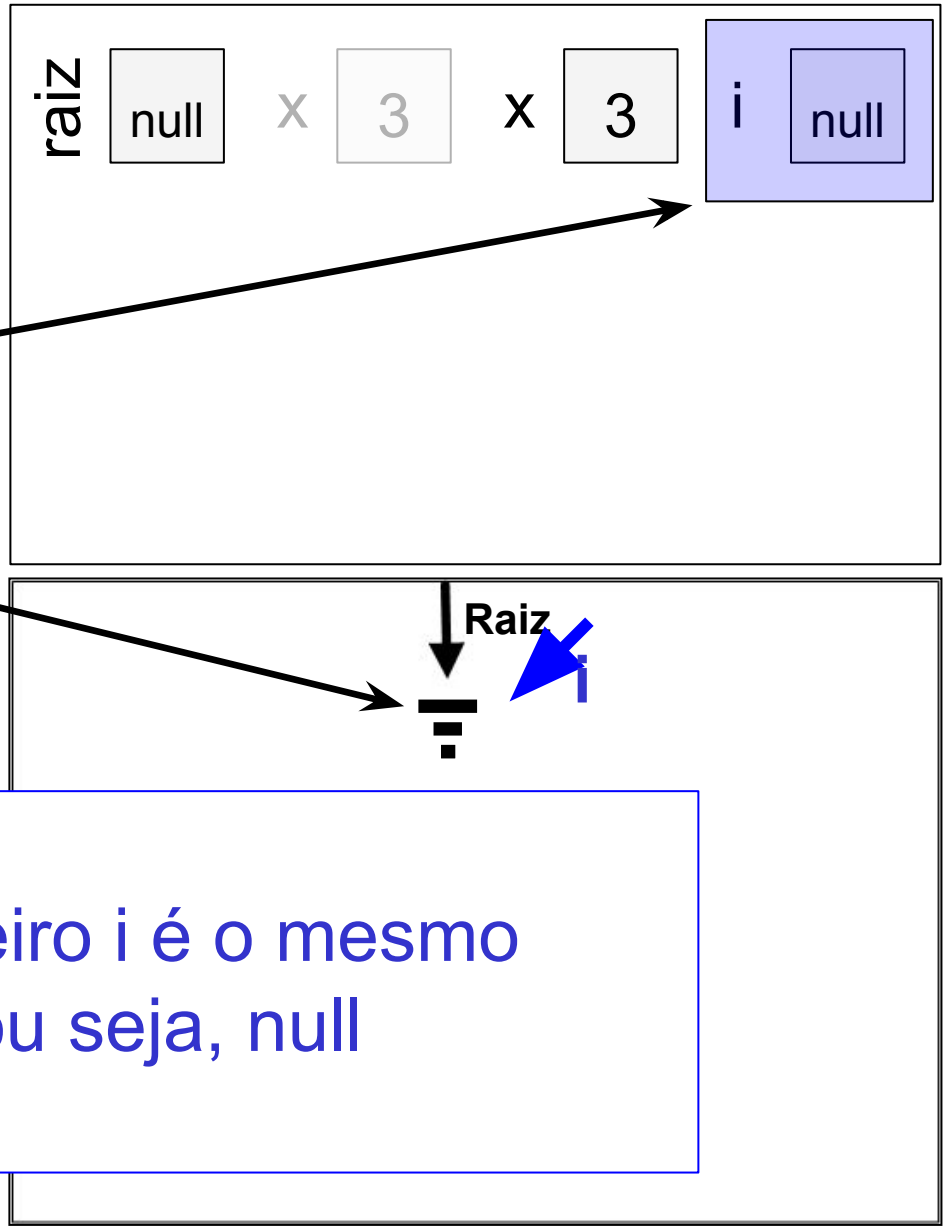




Cada chamada do inserir cria novas variáveis e, por isso, temos duas variáveis com nome x

Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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
   } else
      thro
   }
   returr
}
```

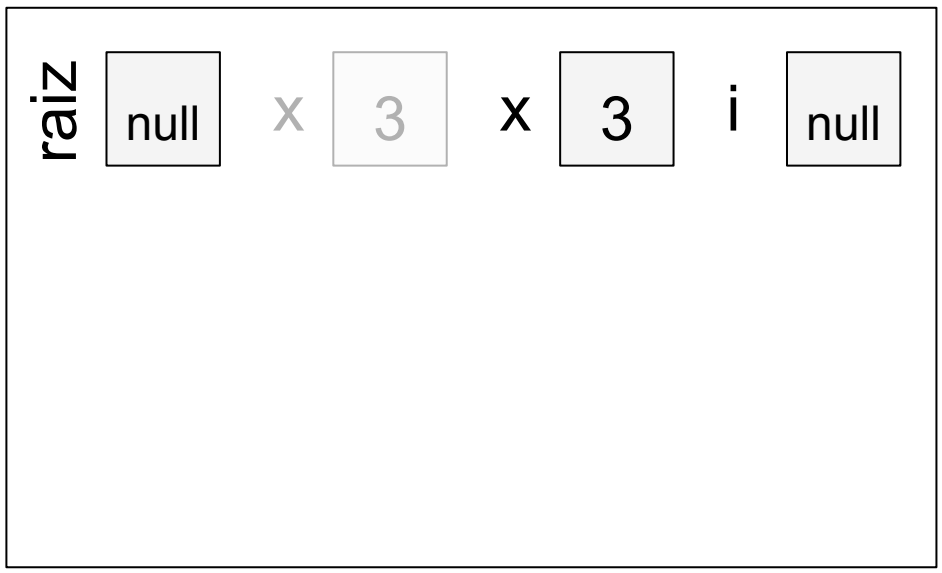


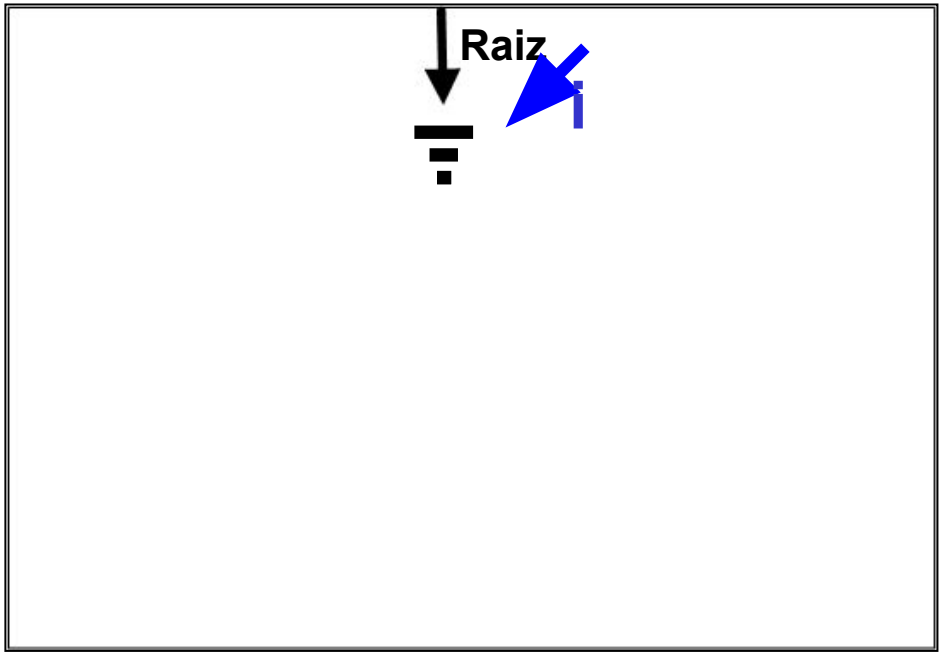
O valor inicial do ponteiro i é o mesmo do ponteiro raiz, ou seja, null

Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

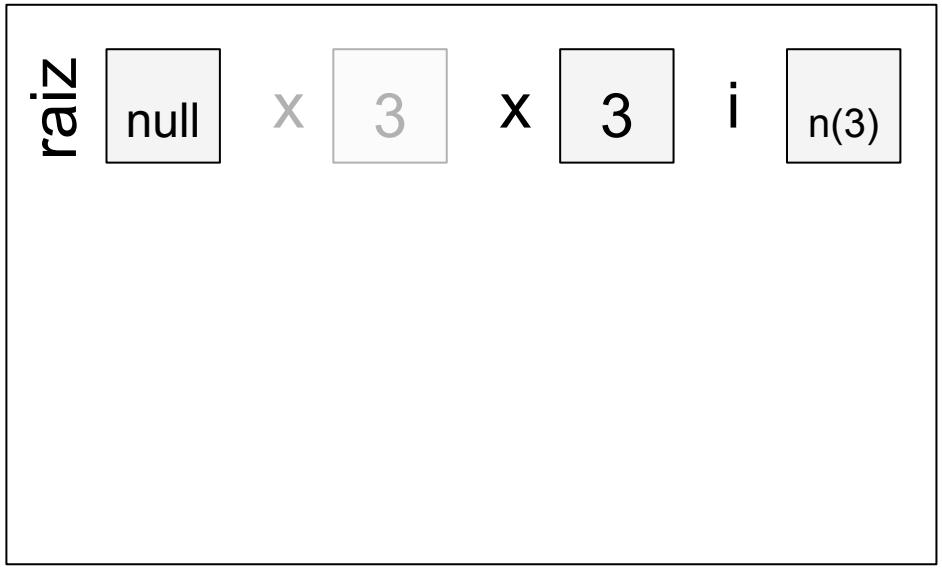
true: null == null

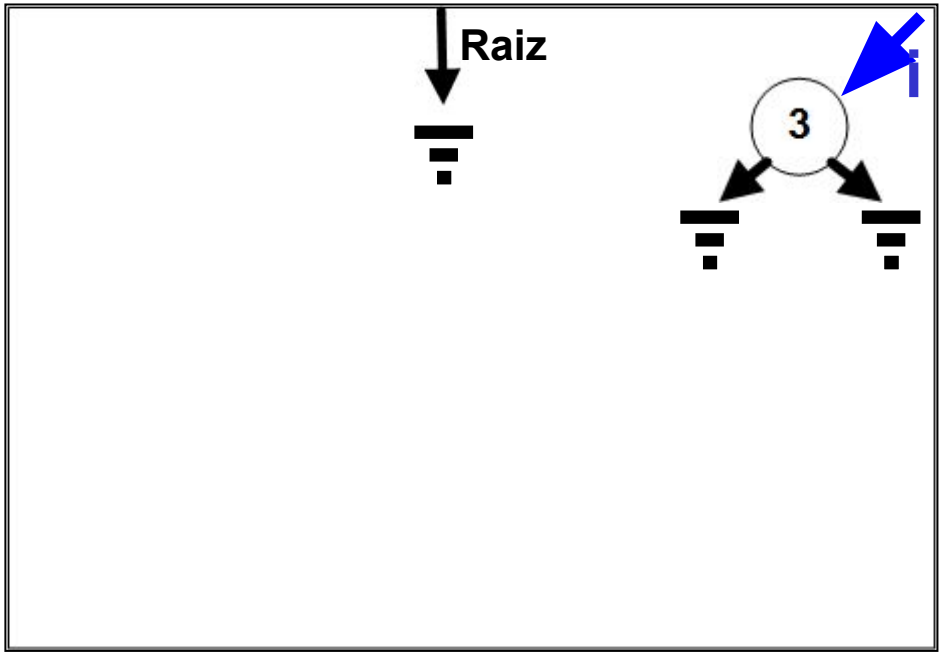




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

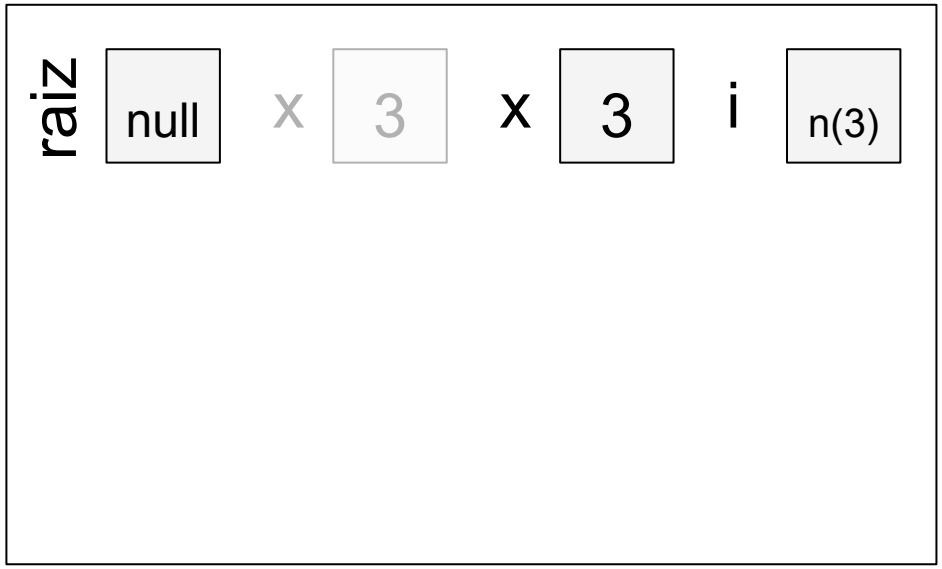


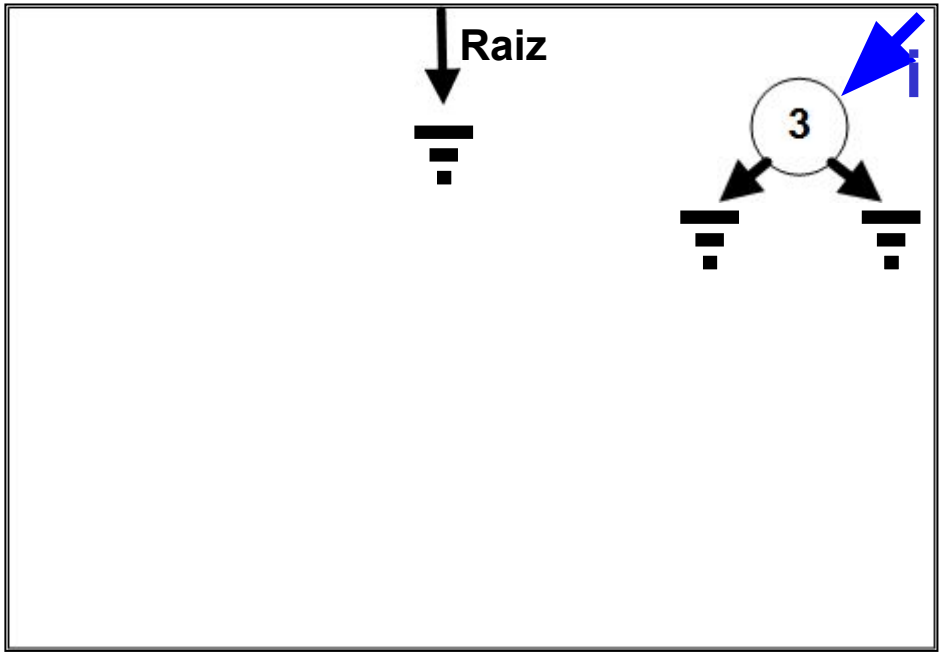


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

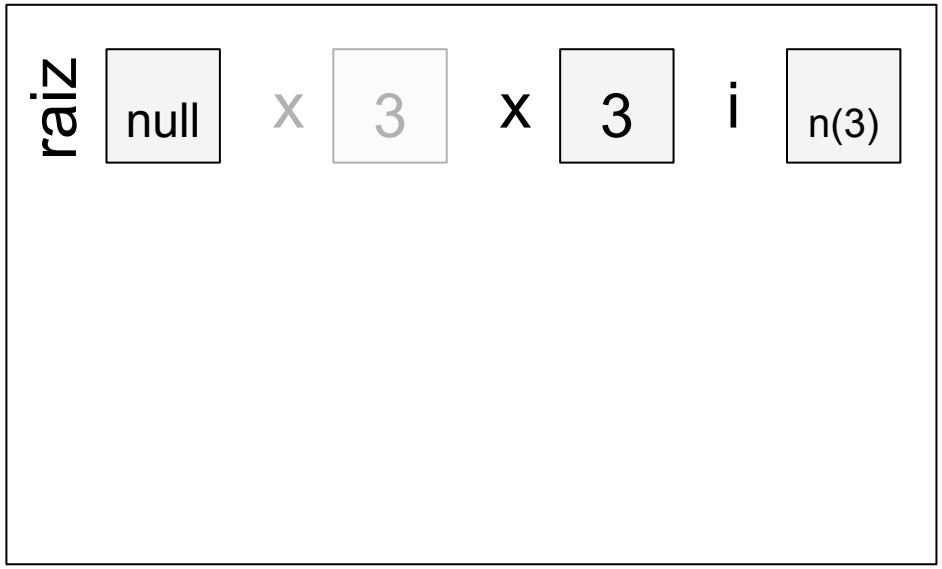
retorna o endereço de n(3)

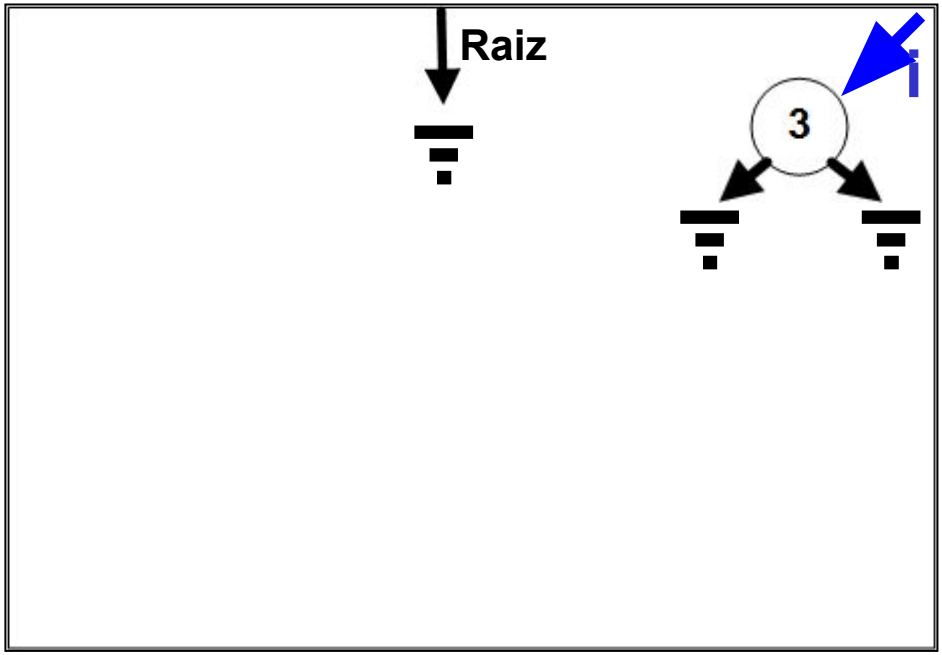




Inserção em Java com Retorno de Referência

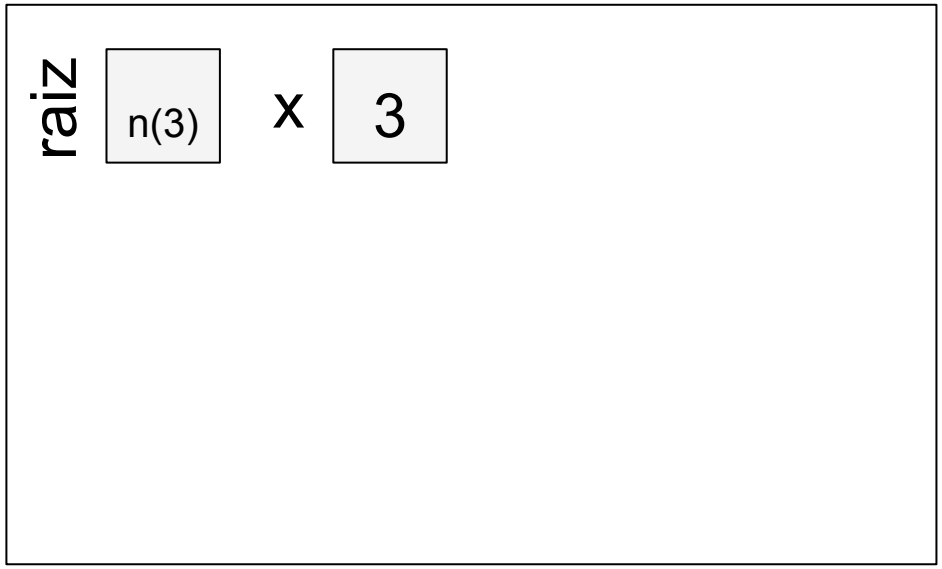
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

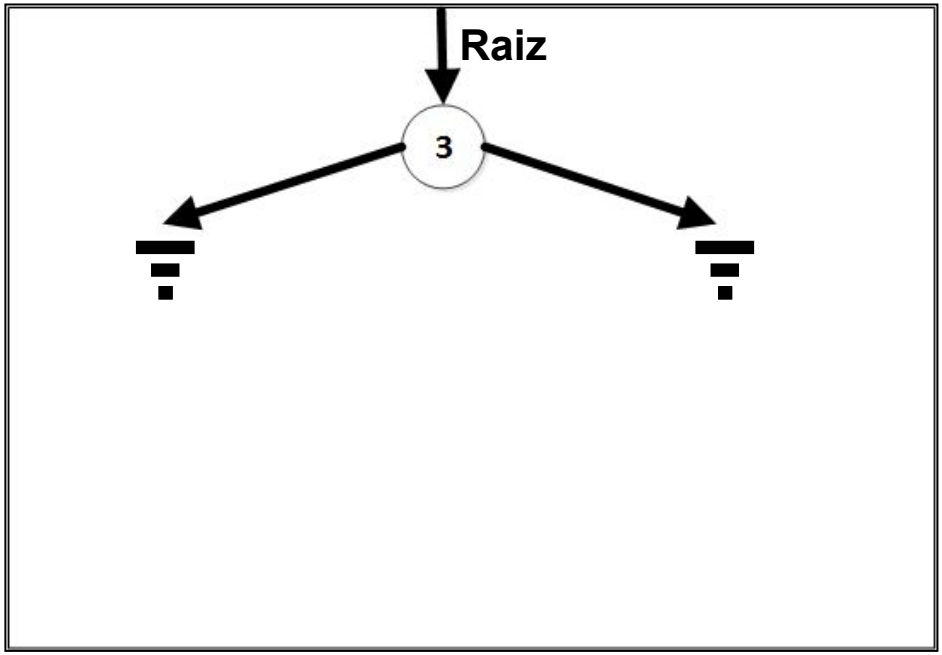




Inserção em Java com Retorno de Referência

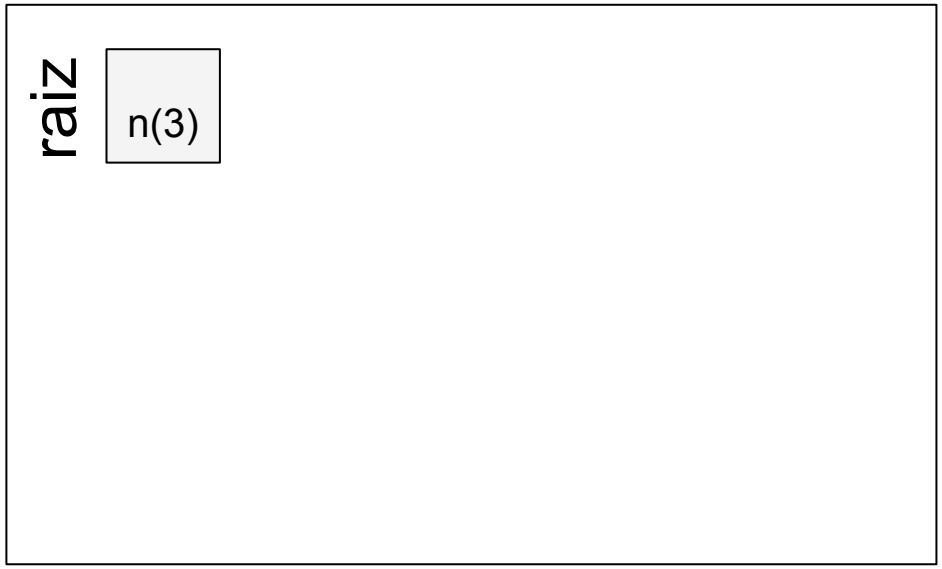
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

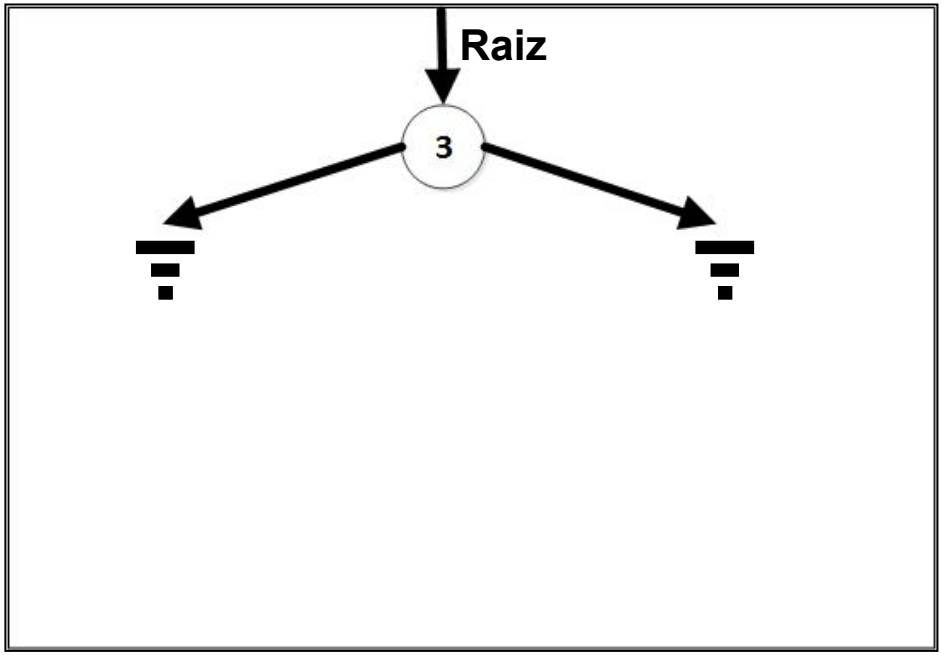




Inserção em Java com Retorno de Referência

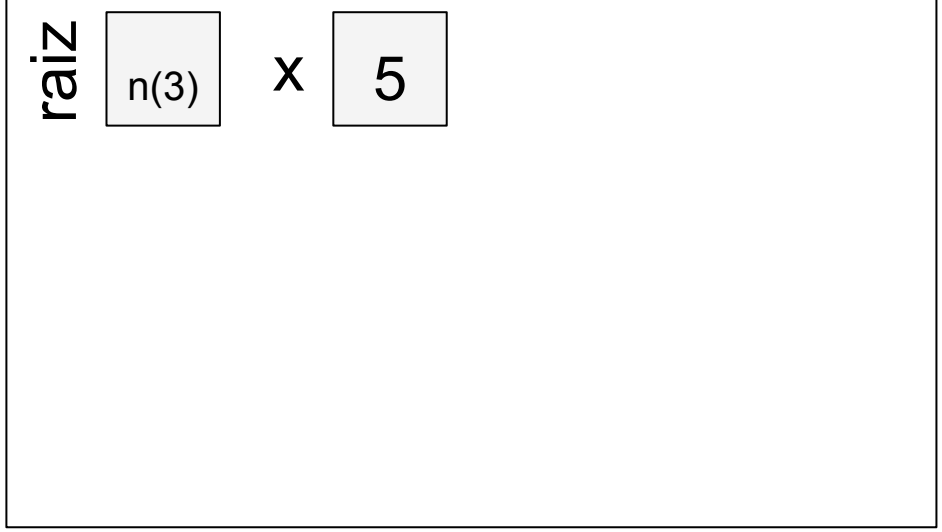
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

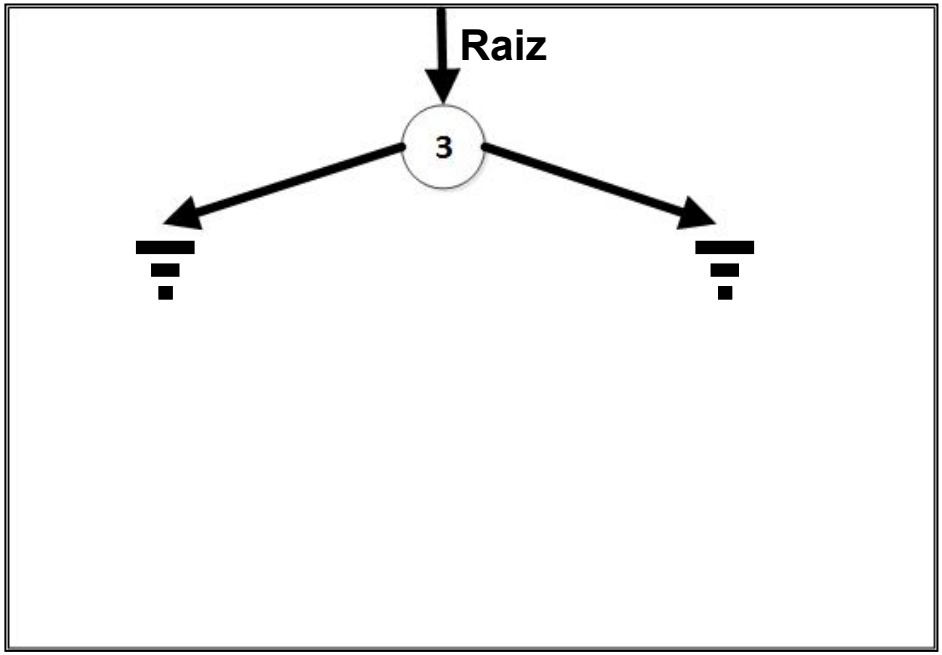




Inserção em Java com Retorno de Referência

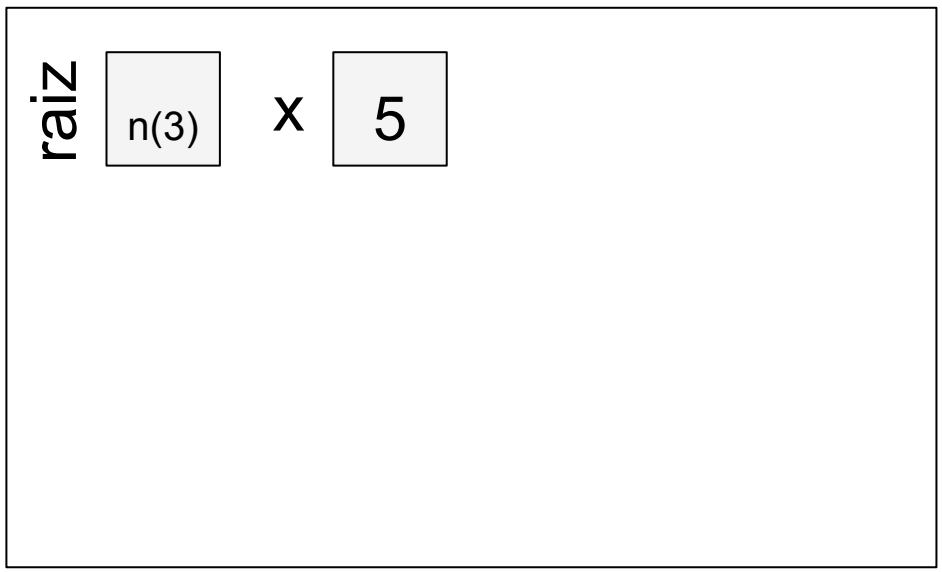
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

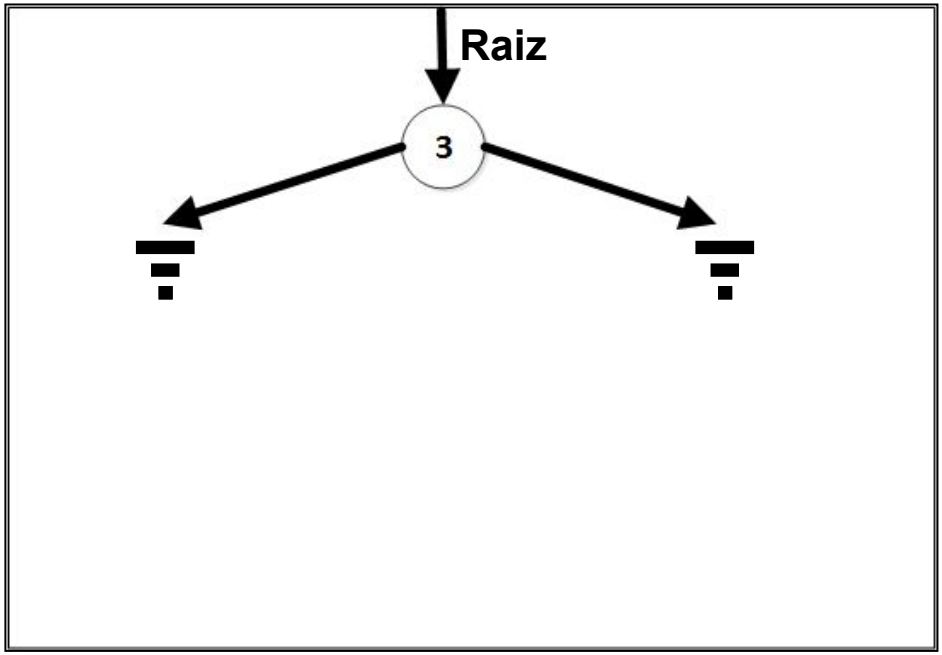




Inserção em Java com Retorno de Referência

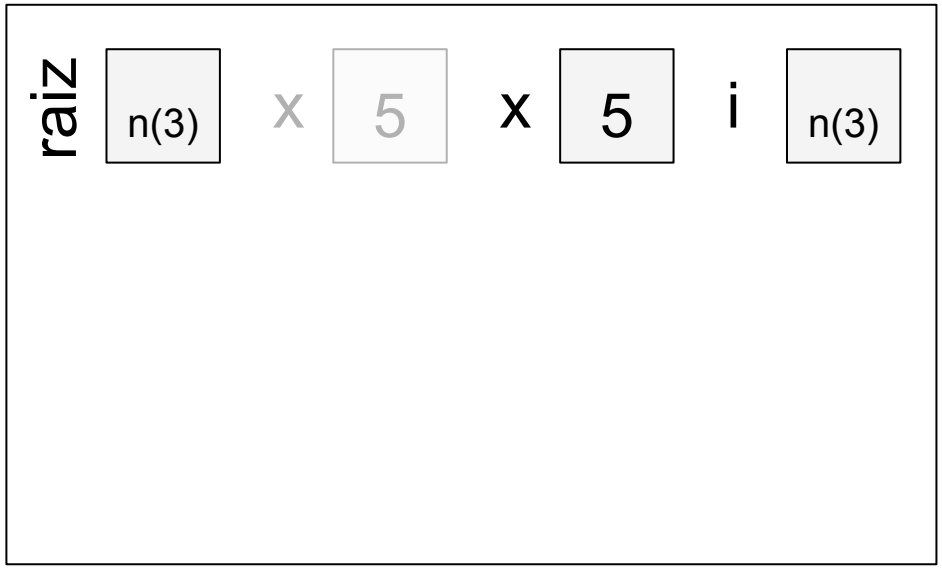
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

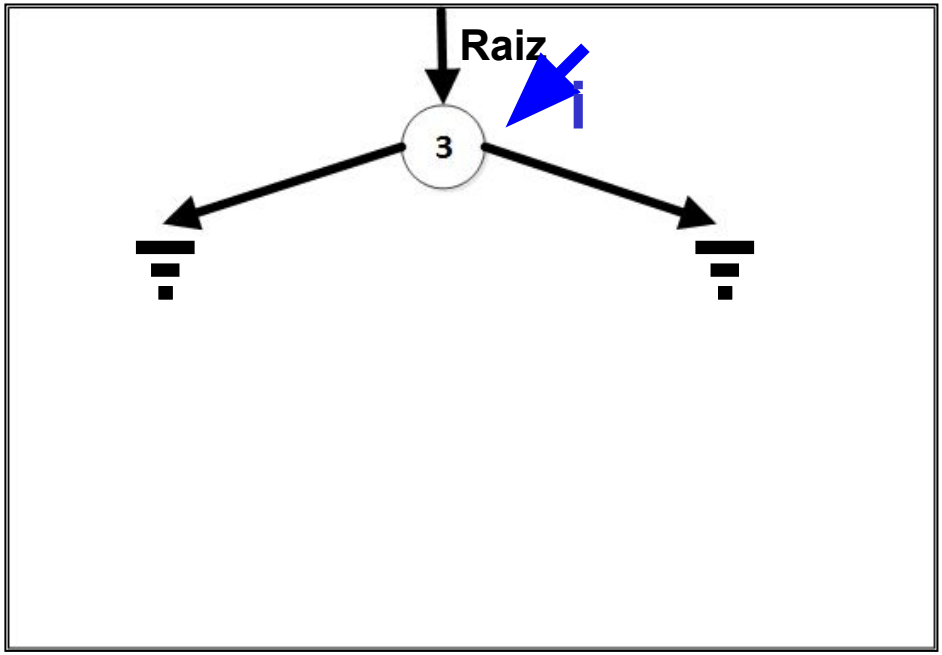




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

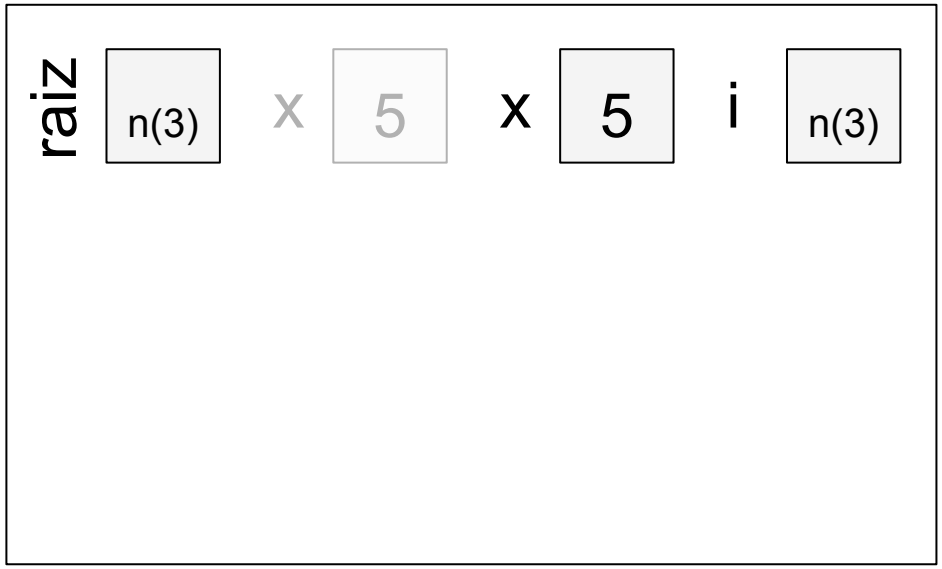


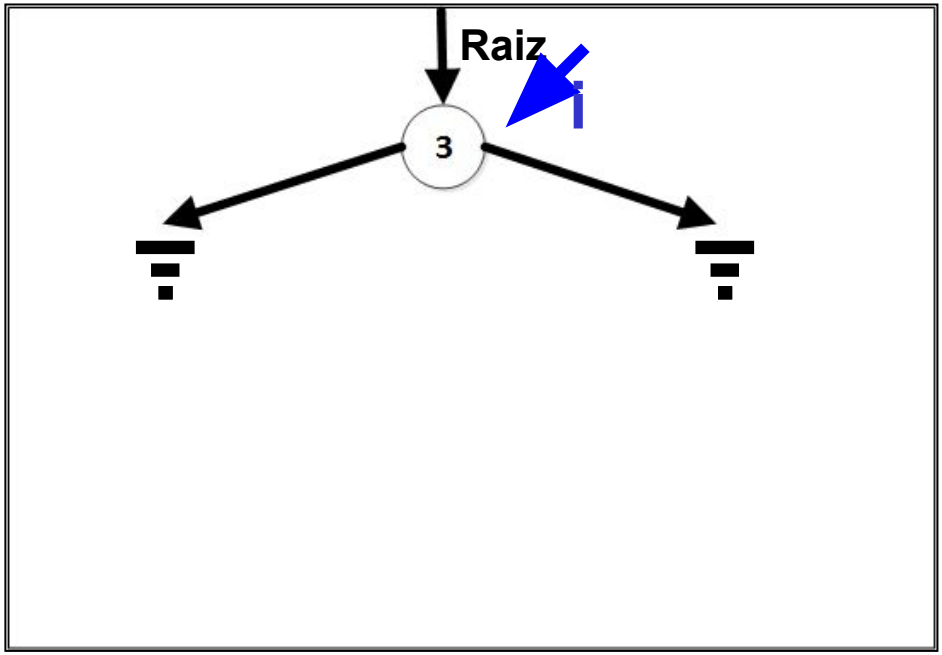


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

false: n(3) == null

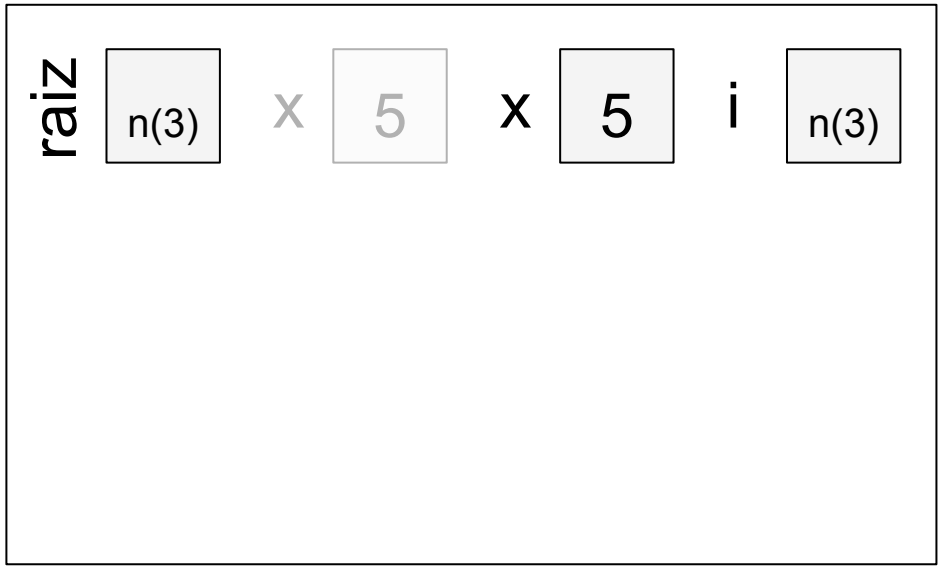


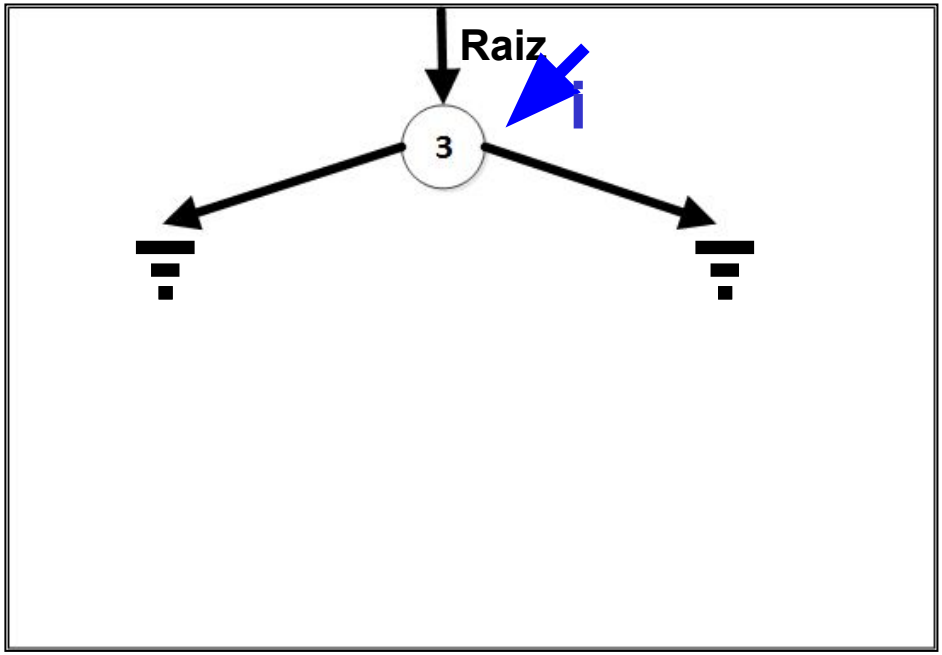


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

false: 5 < 3

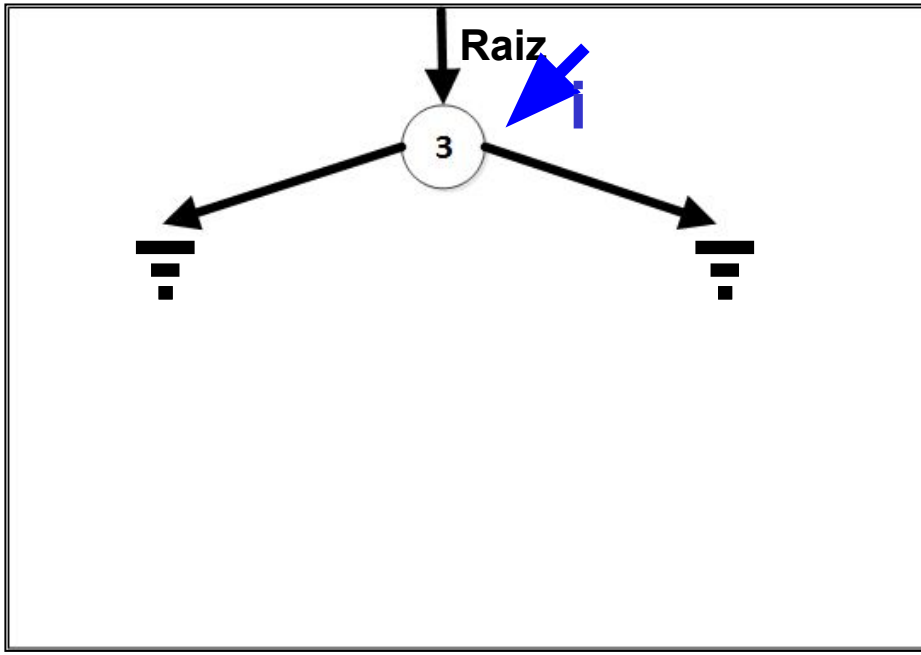




Inserção em Java com Retorno de Referência

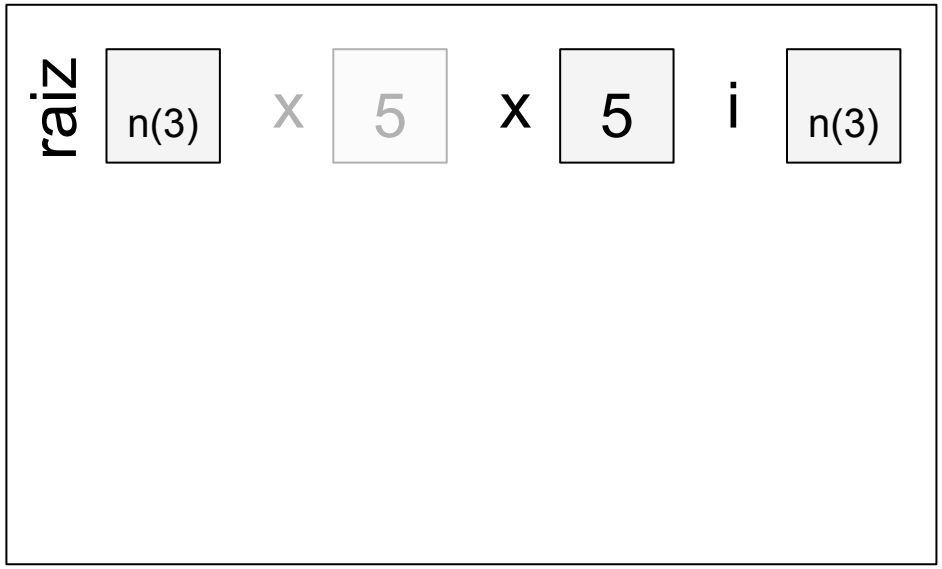
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

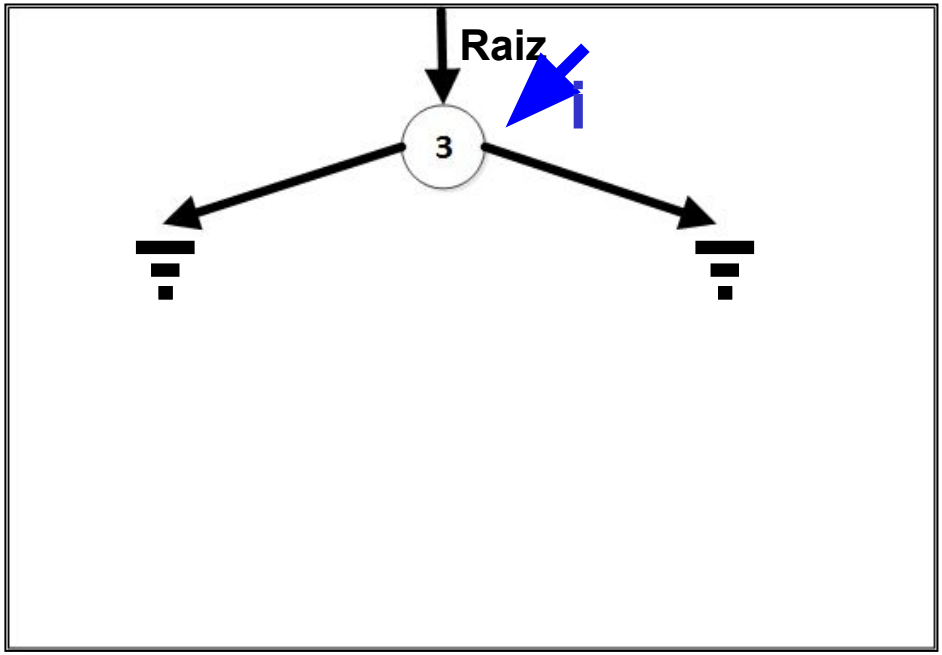
true: 5 > 3



Inserção em Java com Retorno de Referência

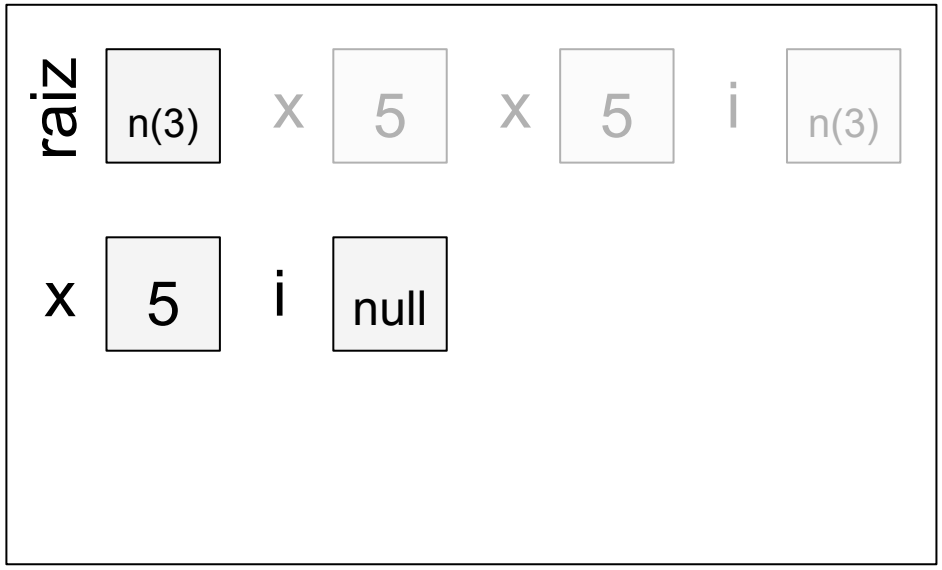
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

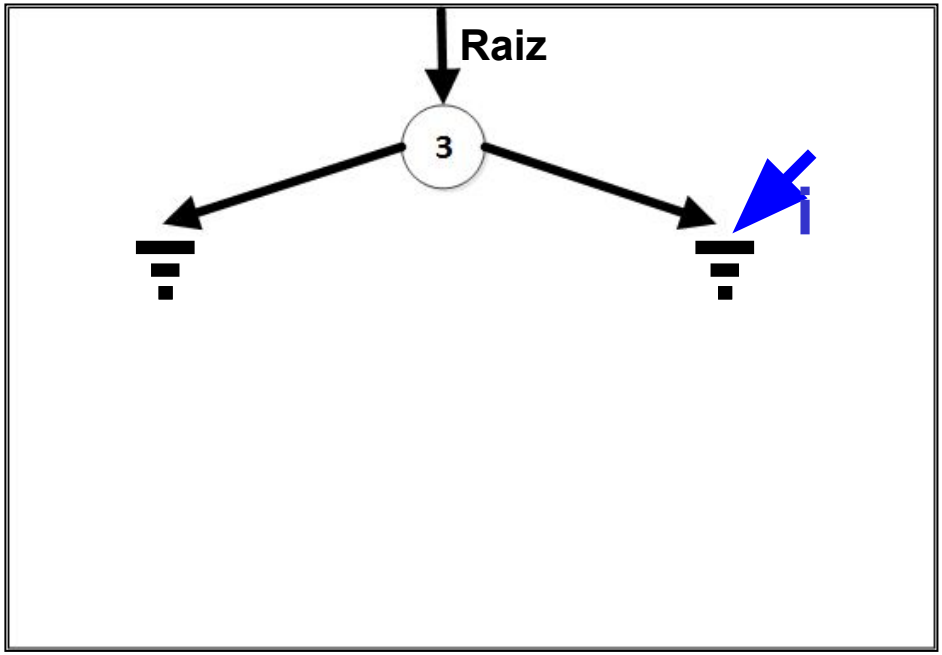




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

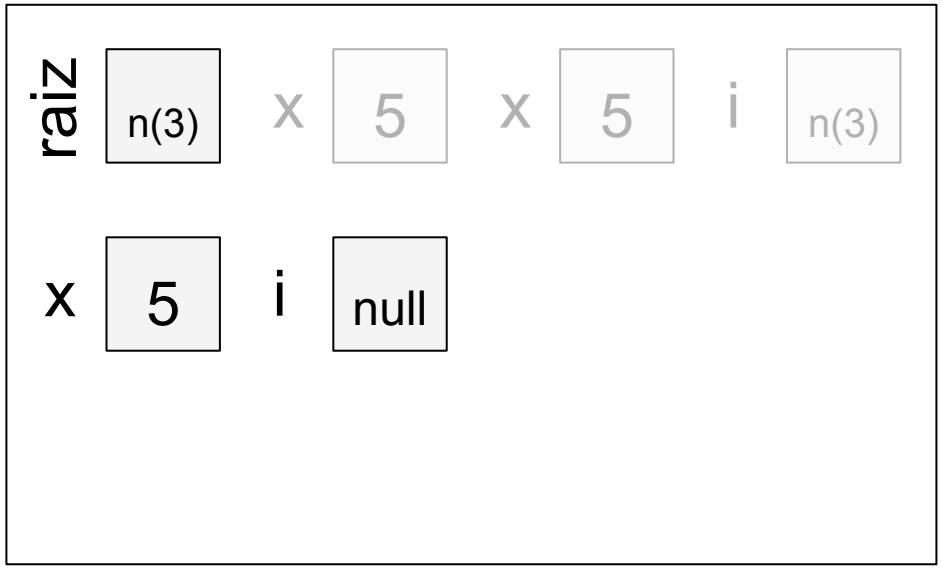


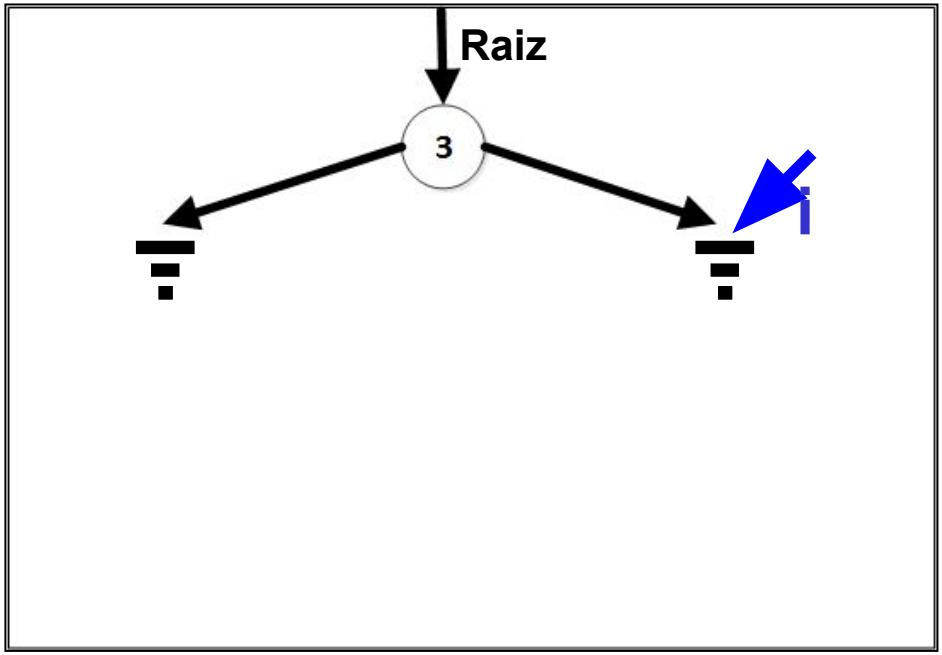


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

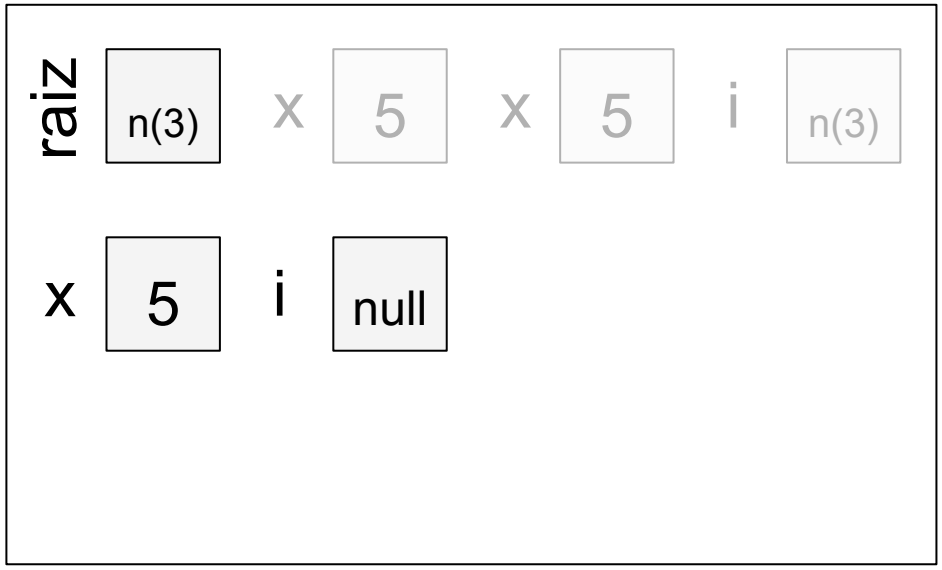
true: null == null

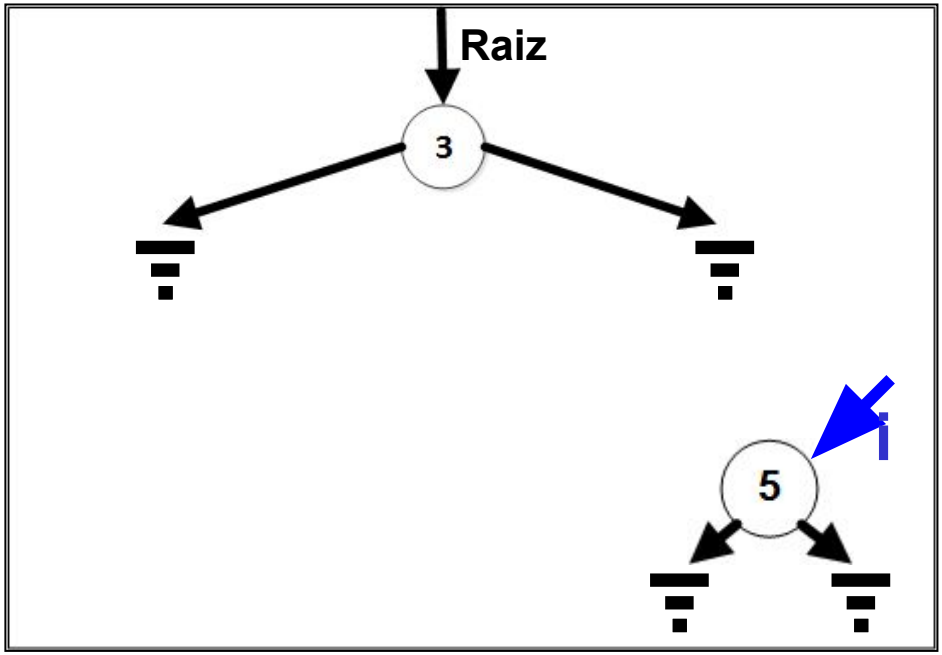




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

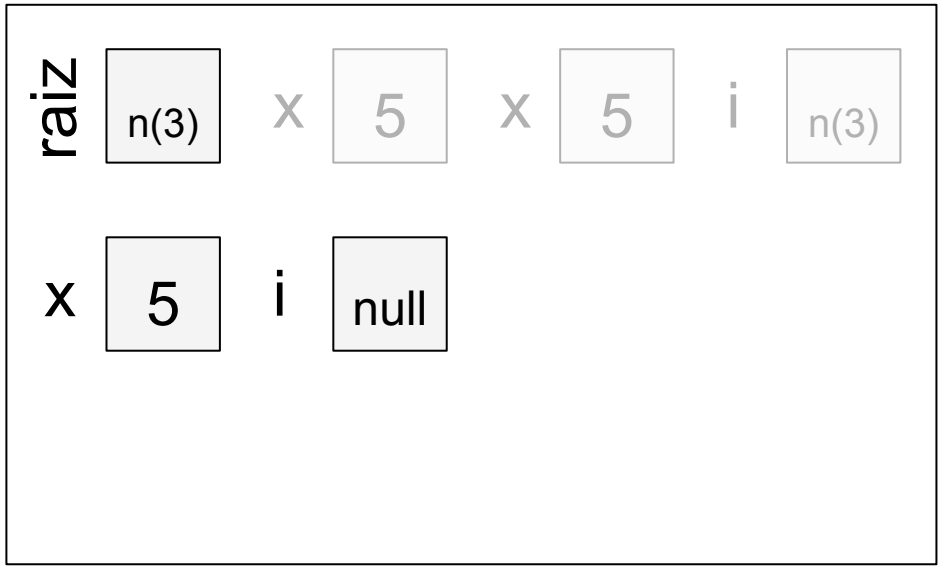


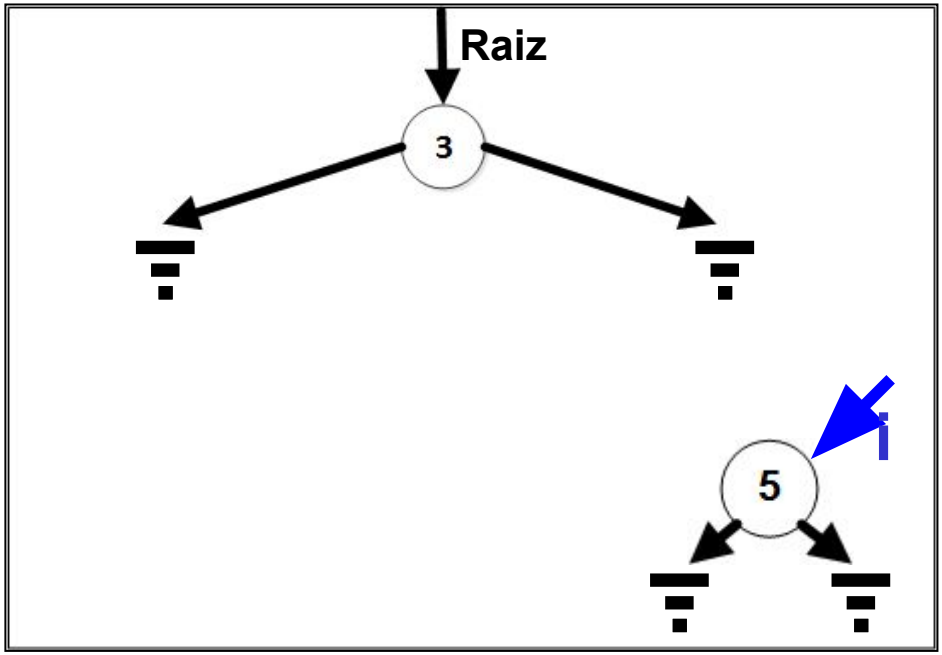


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

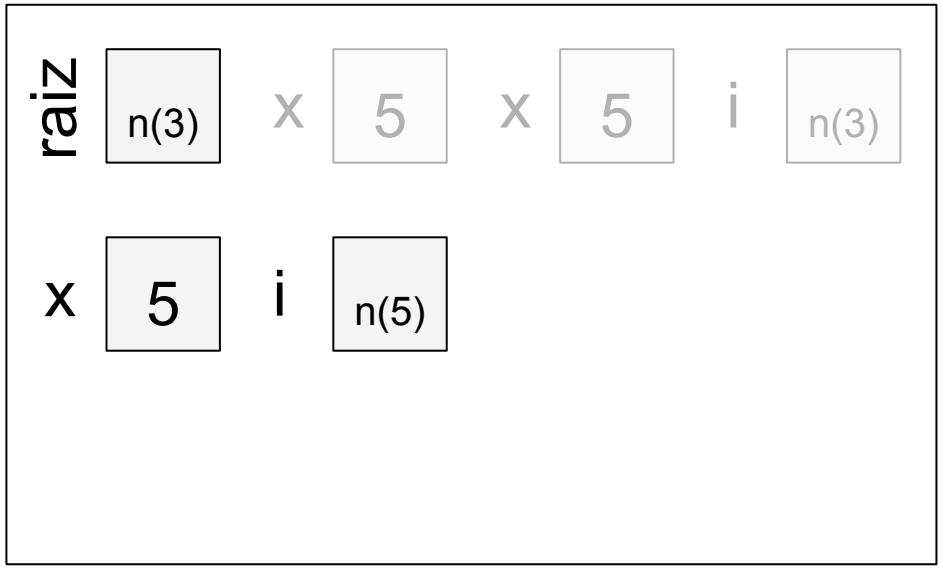
retorna o endereço de n(5)

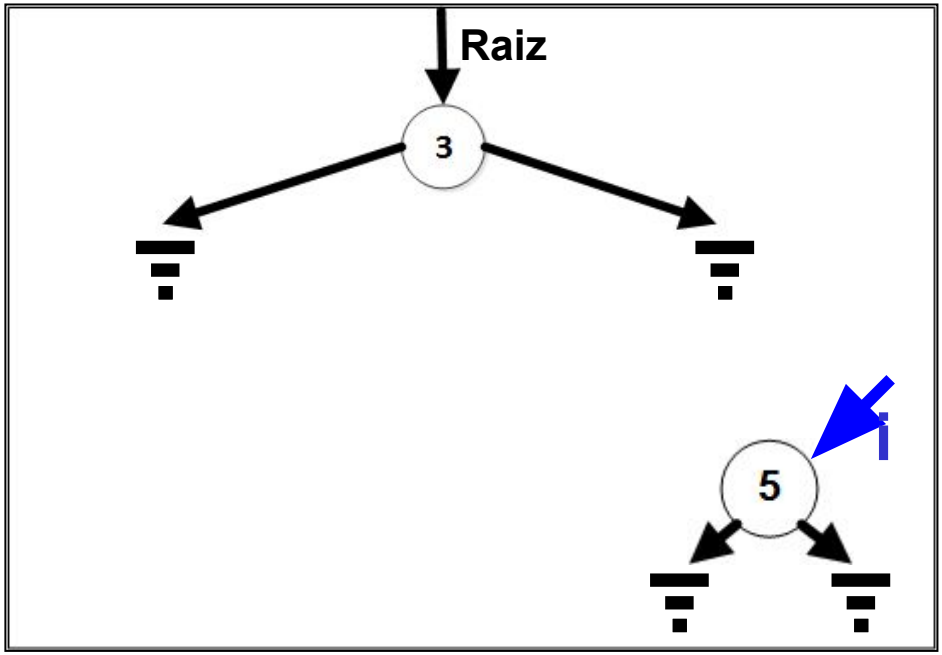




Inserção em Java com Retorno de Referência

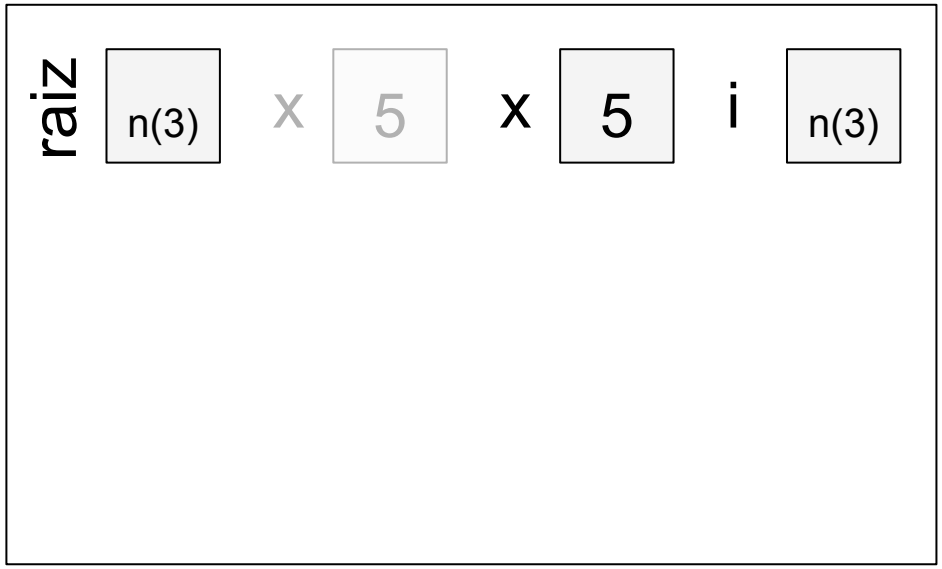
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

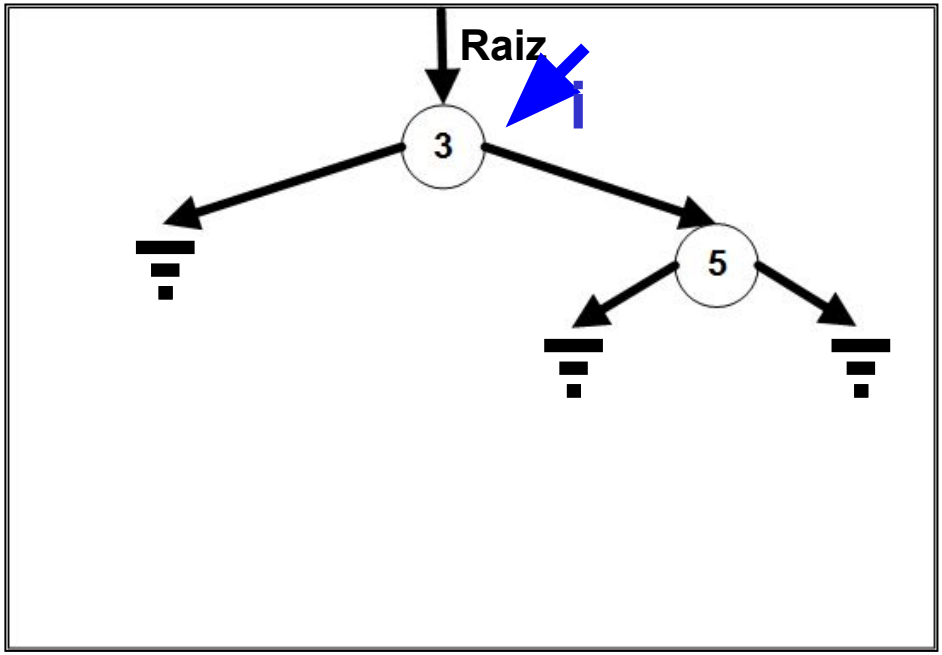




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

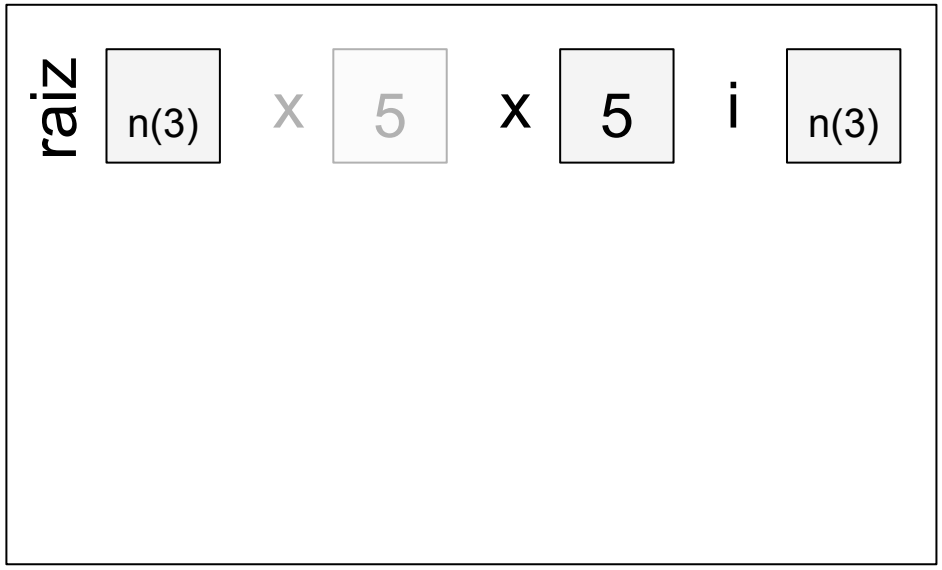


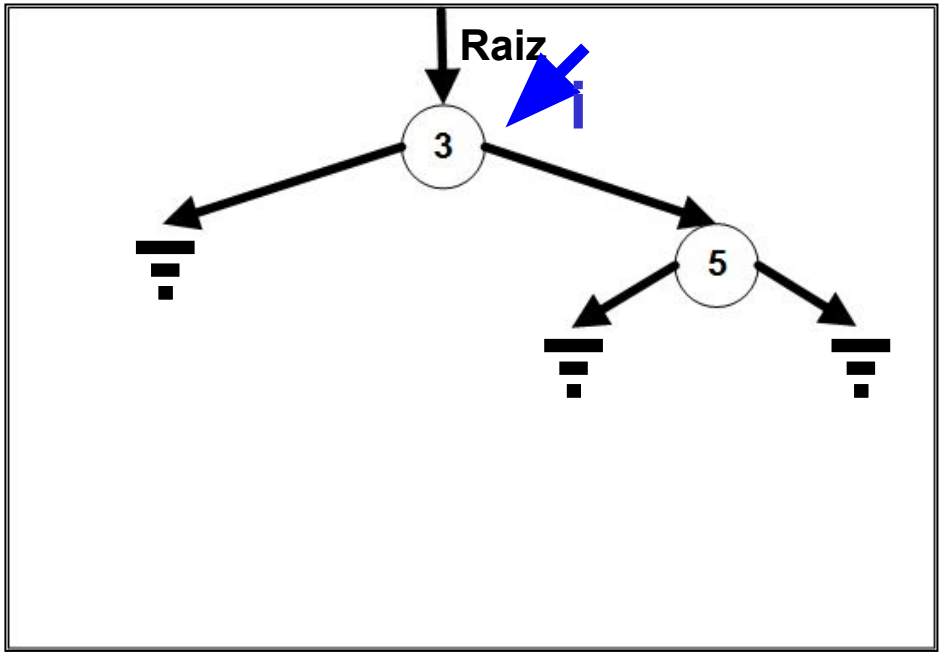


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

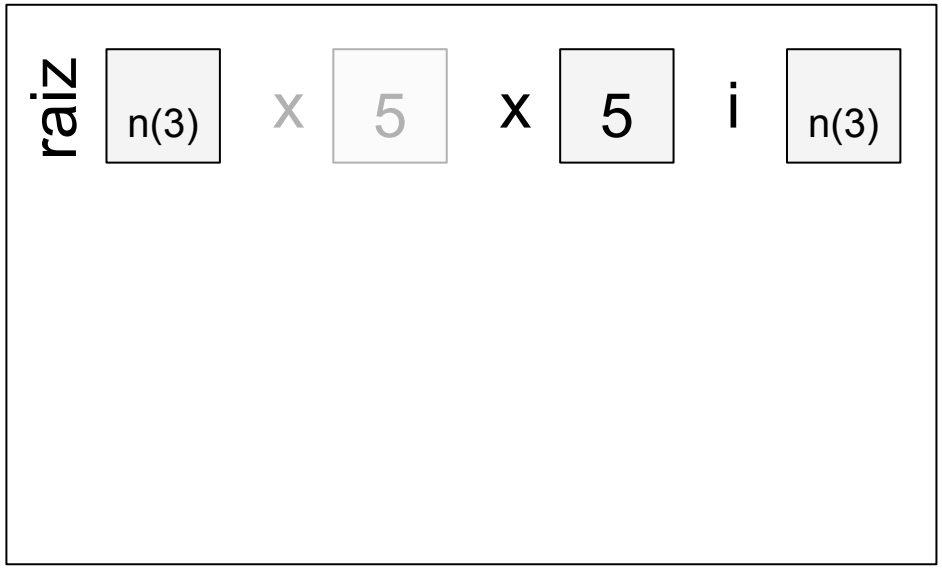
retorna o endereço de n(3)

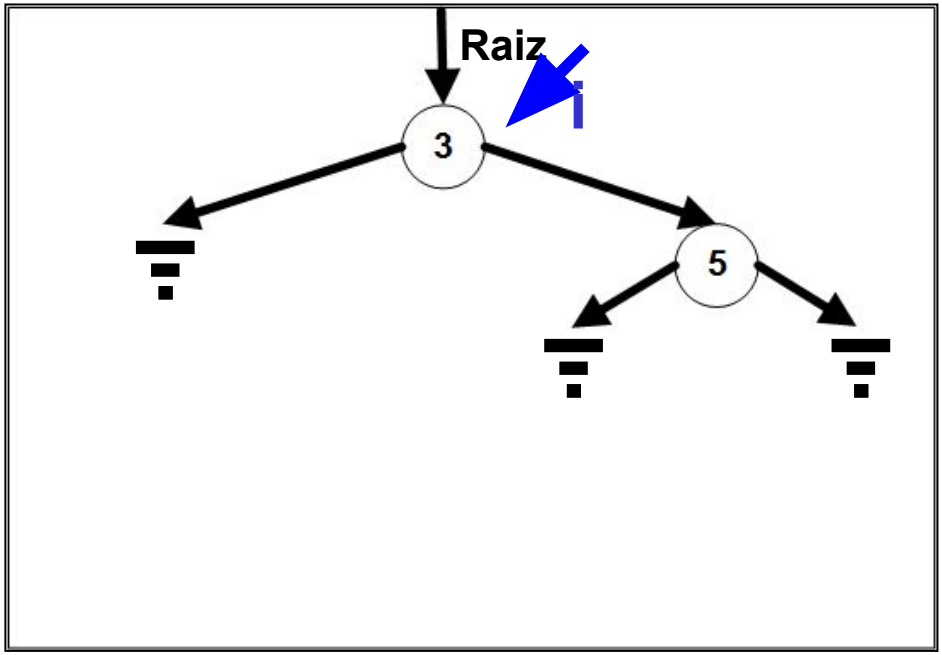




Inserção em Java com Retorno de Referência

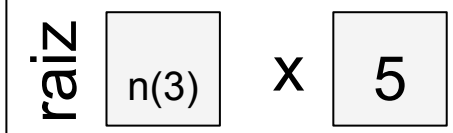
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

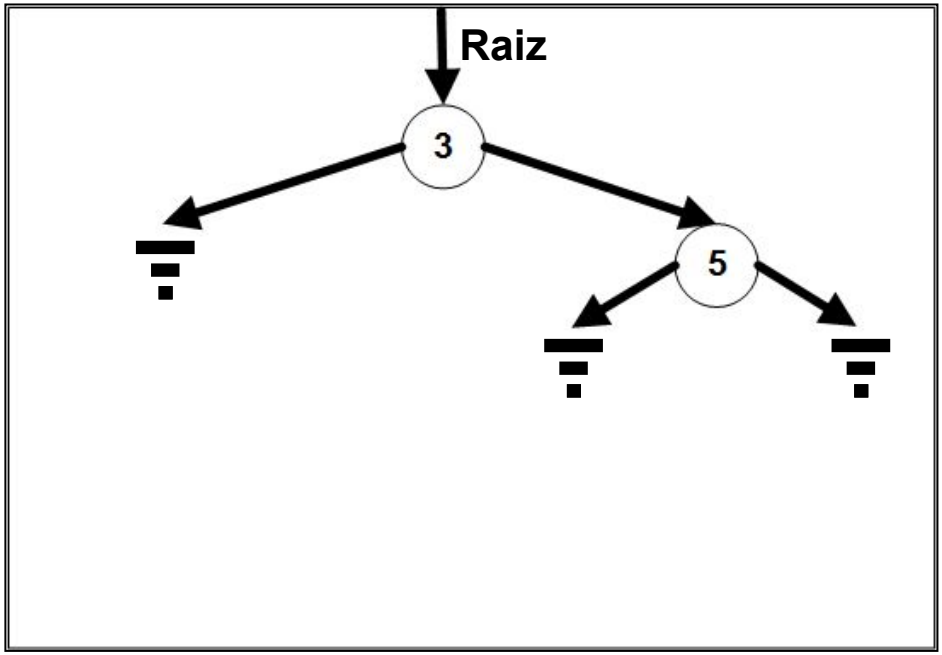




Inserção em Java com Retorno de Referência

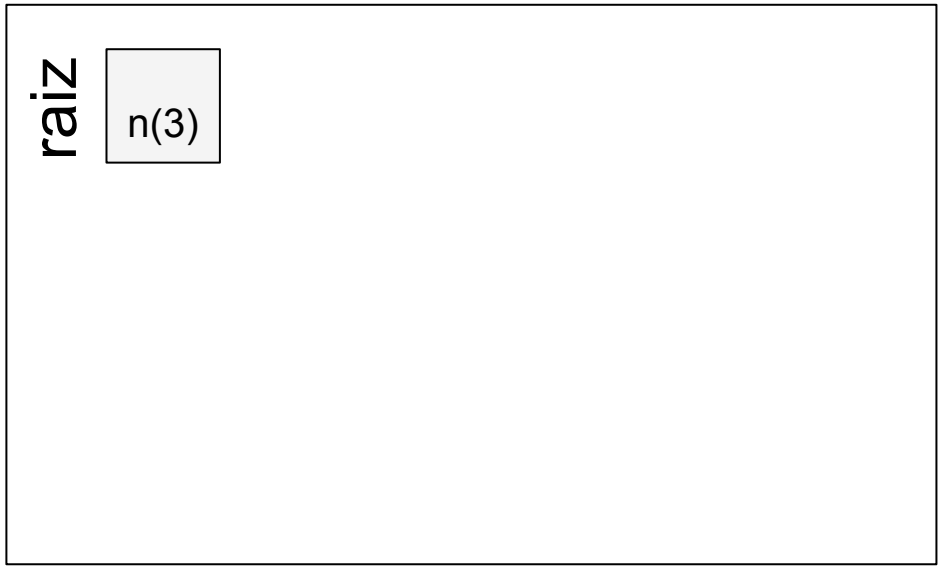
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

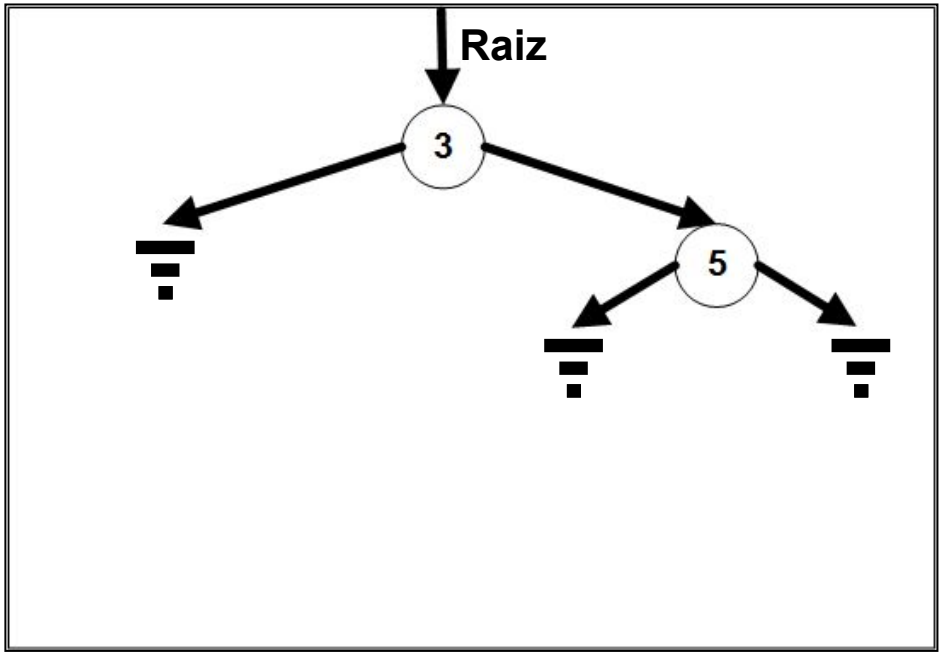




Inserção em Java com Retorno de Referência

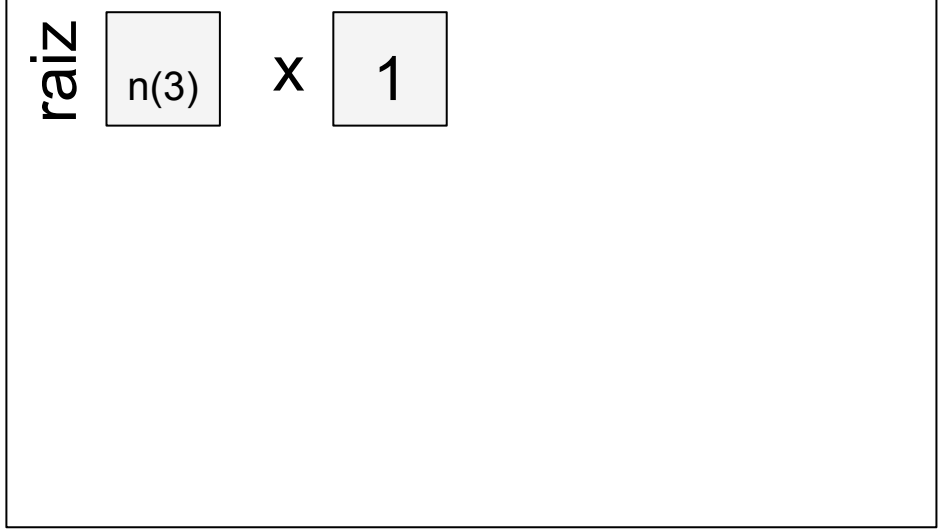
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

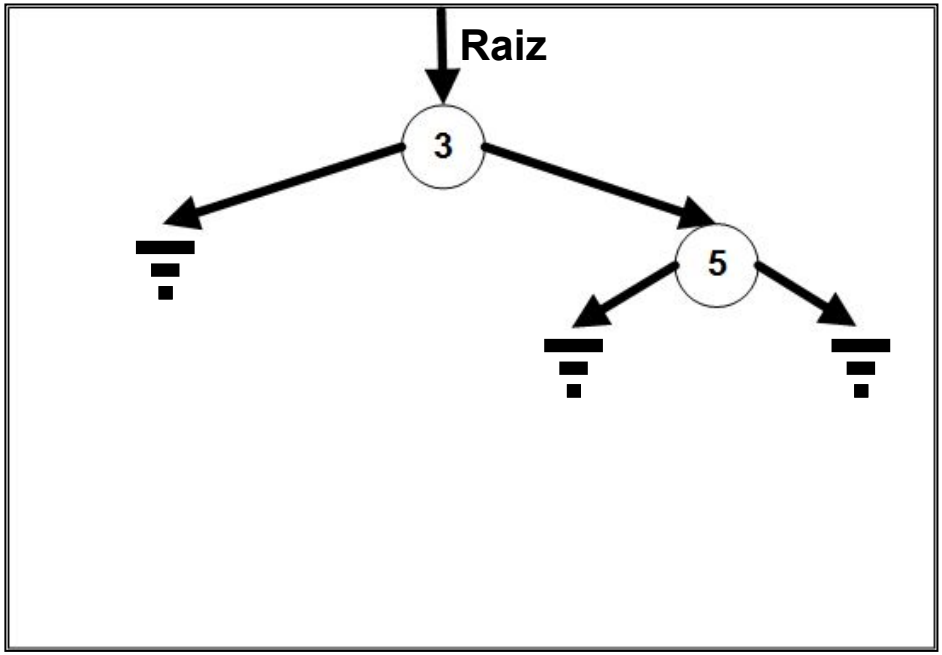




Inserção em Java com Retorno de Referência

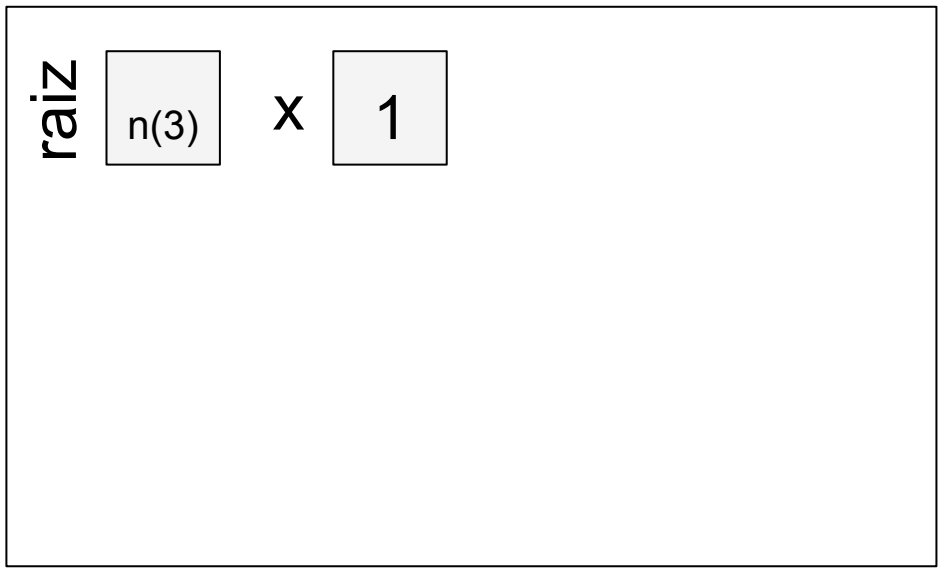
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

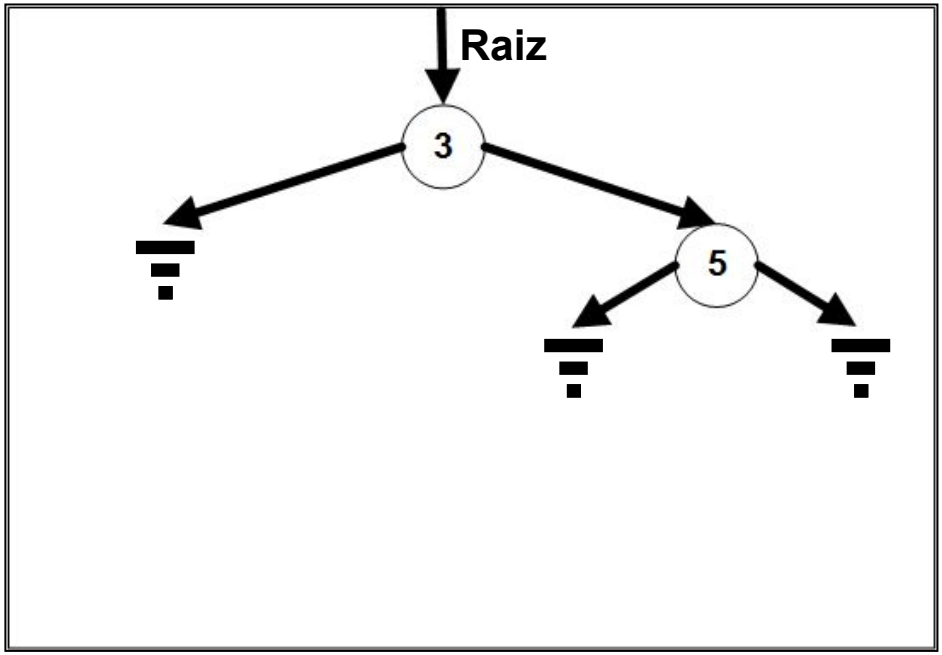




Inserção em Java com Retorno de Referência

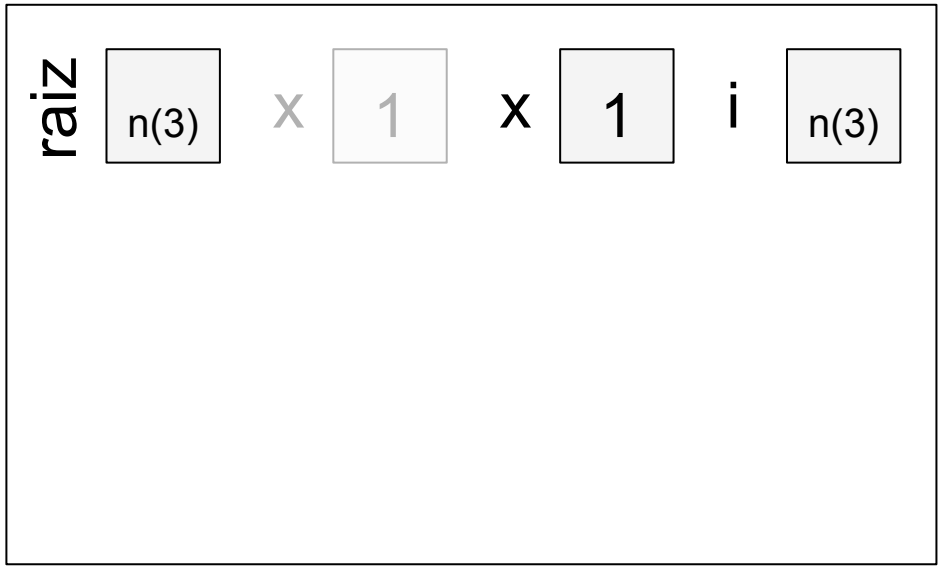
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

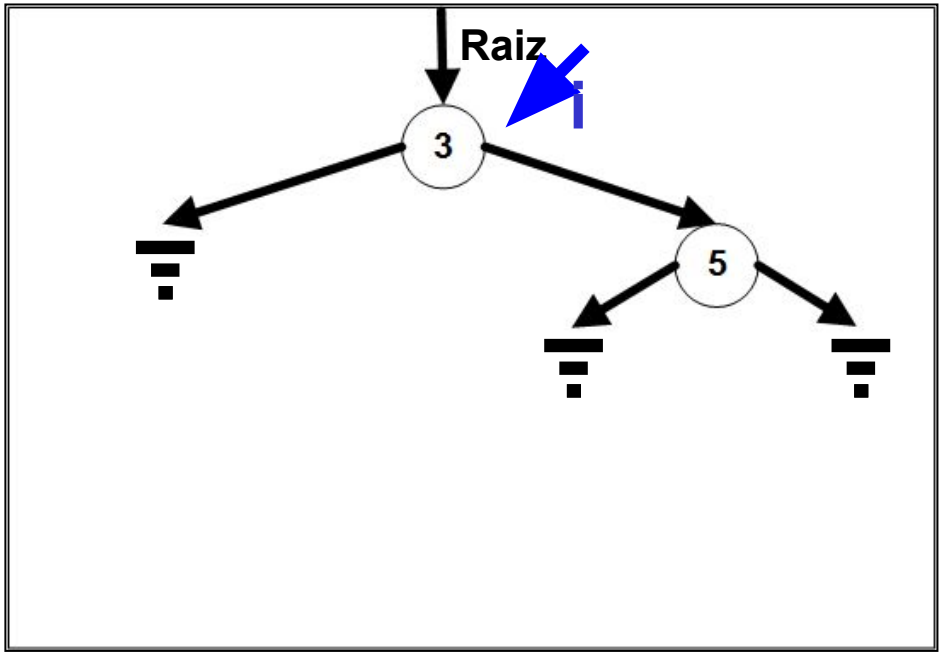




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

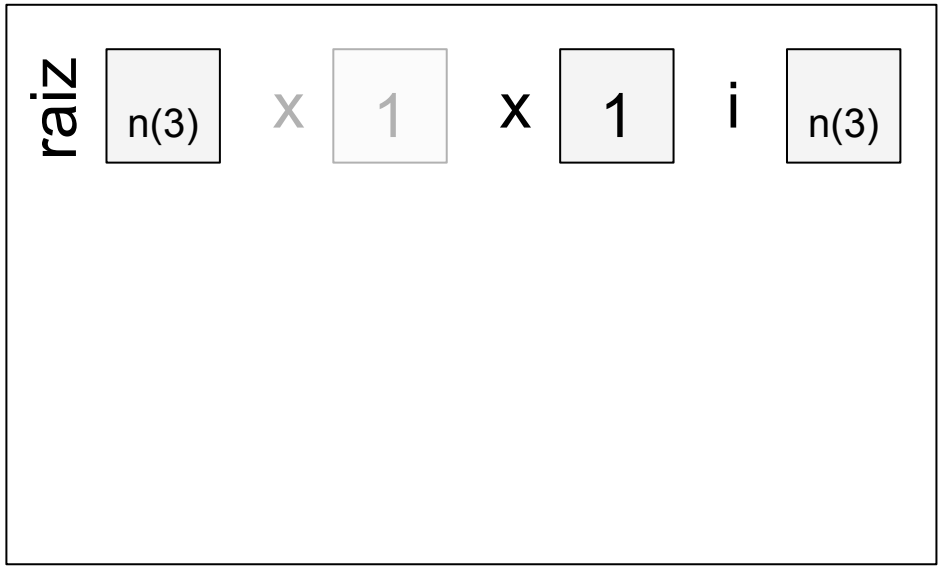


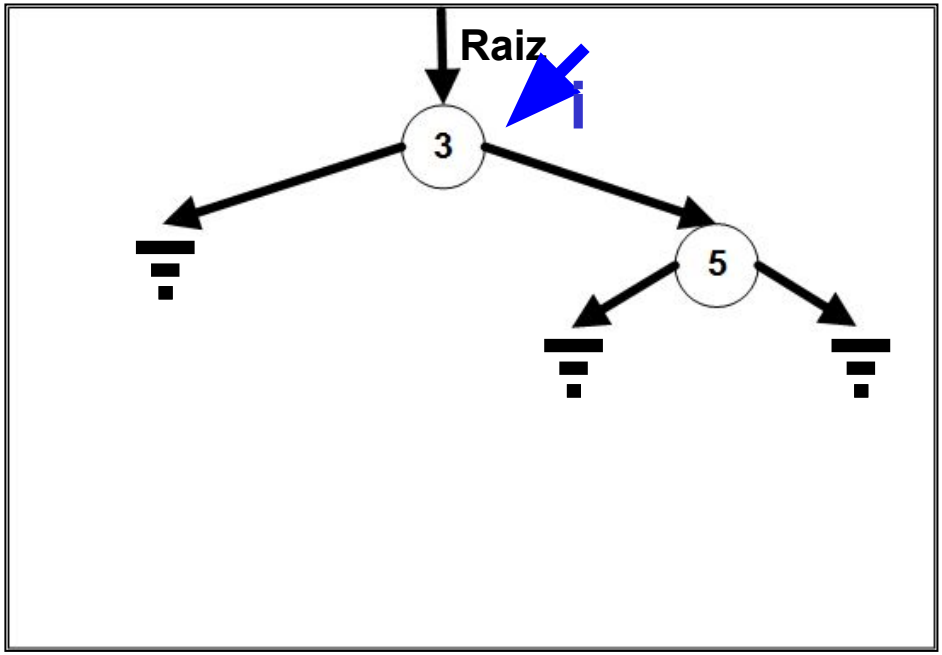


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

false: n(3) == null

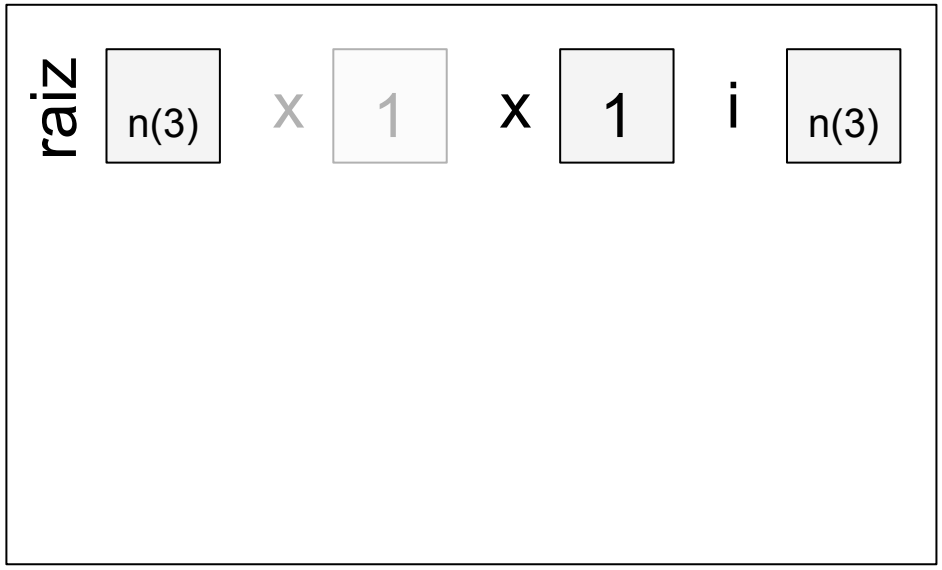


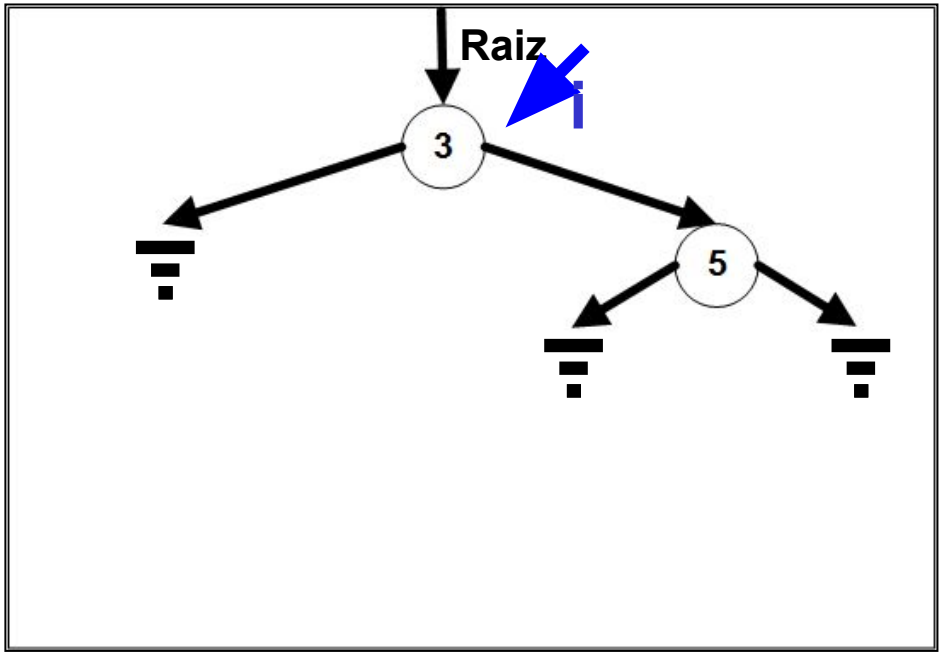


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

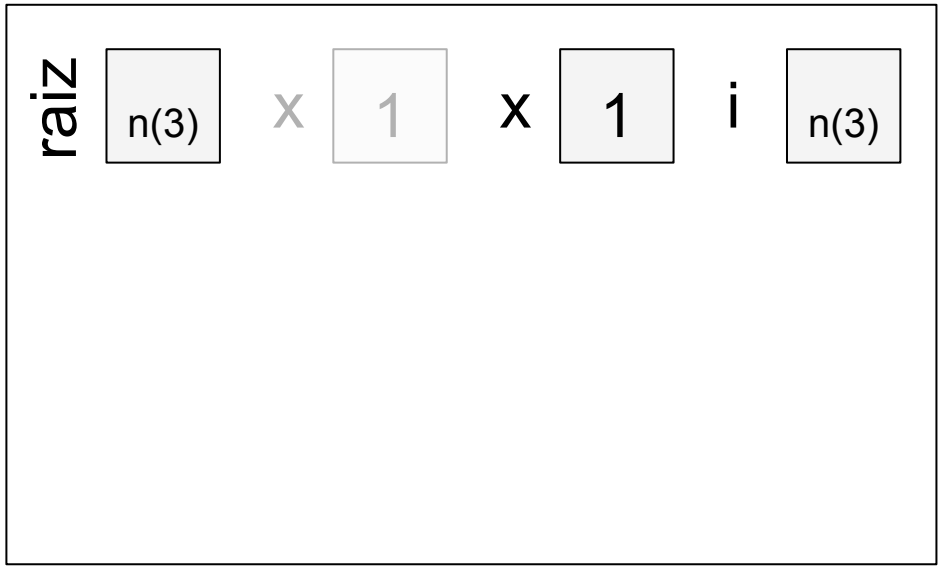
true: 1 < 3

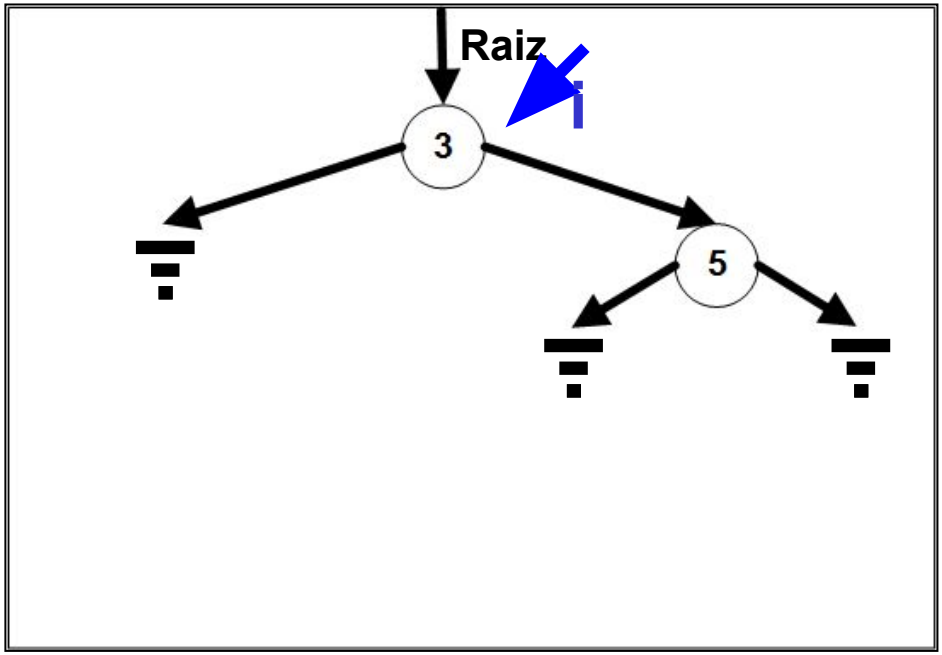




Inserção em Java com Retorno de Referência

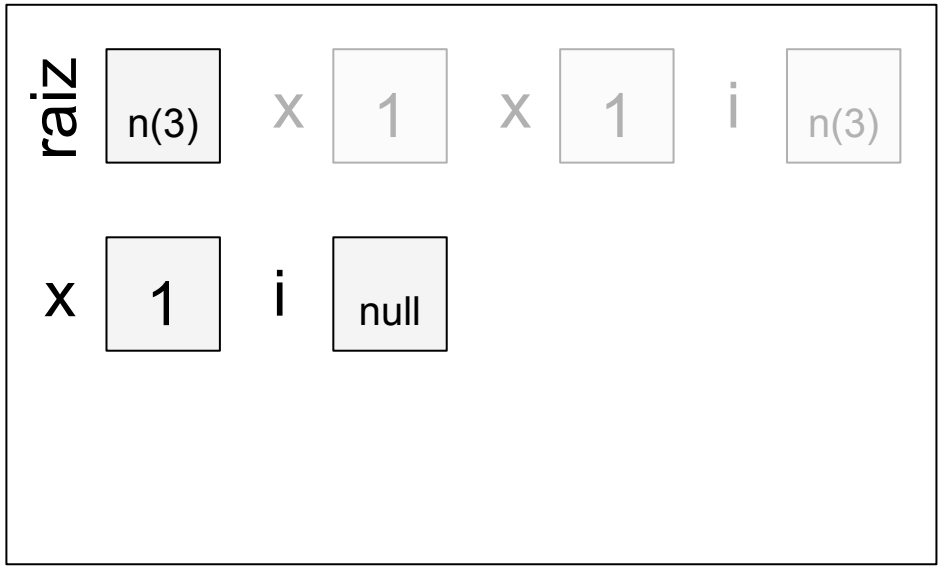
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

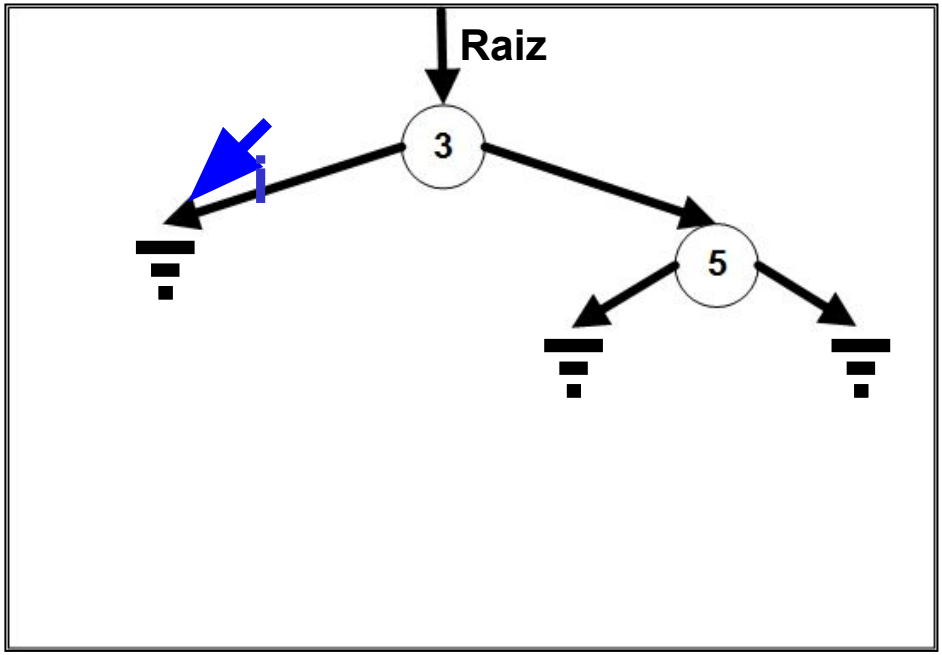




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

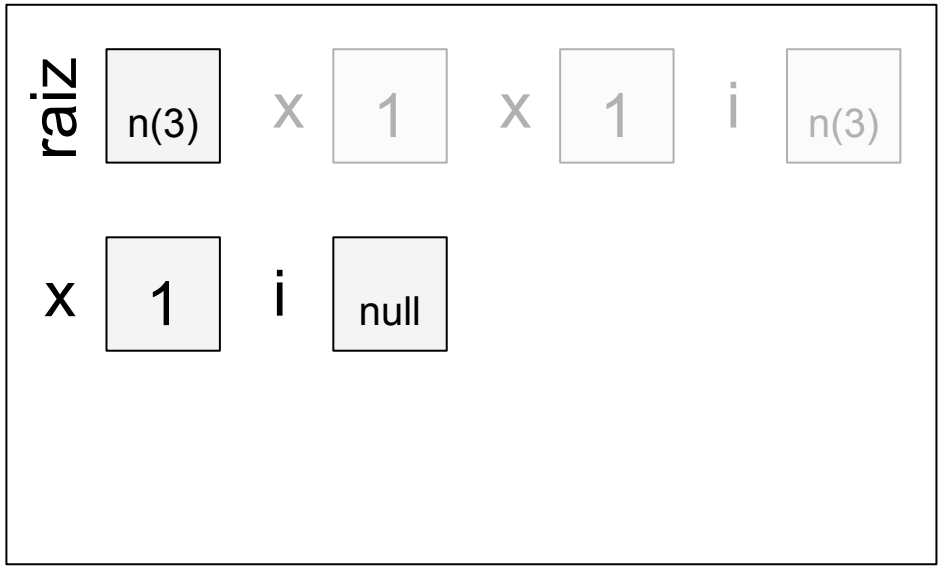


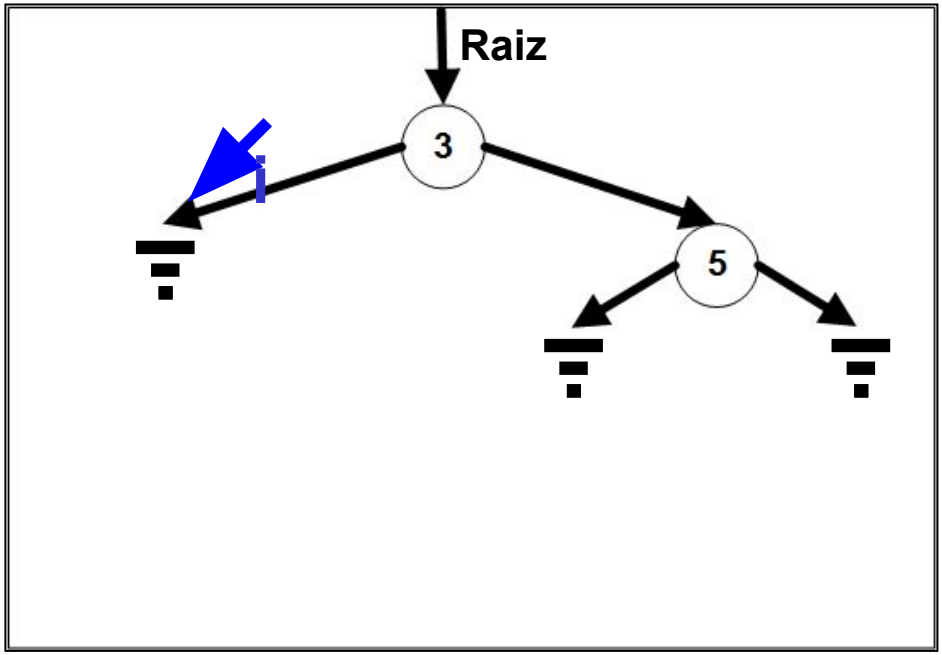


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

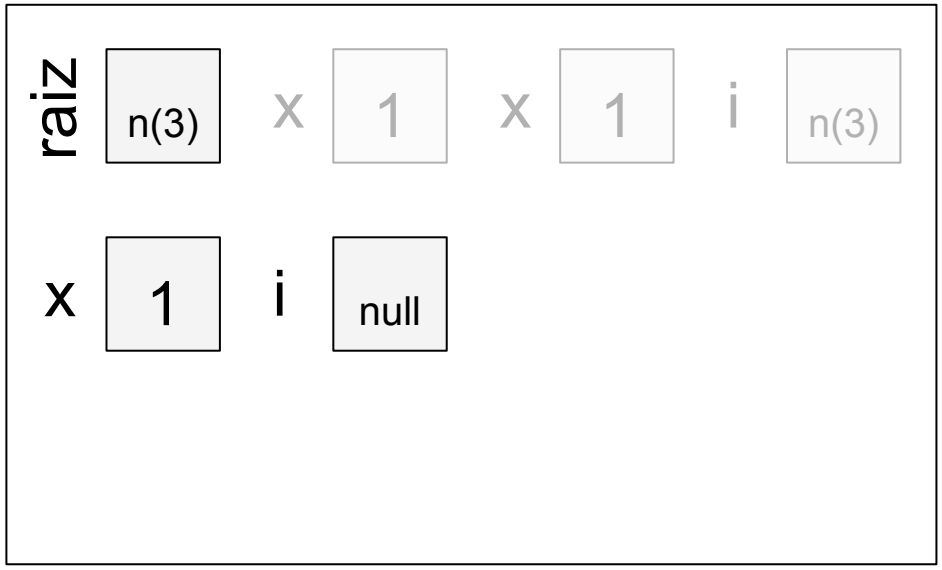
true: null == null

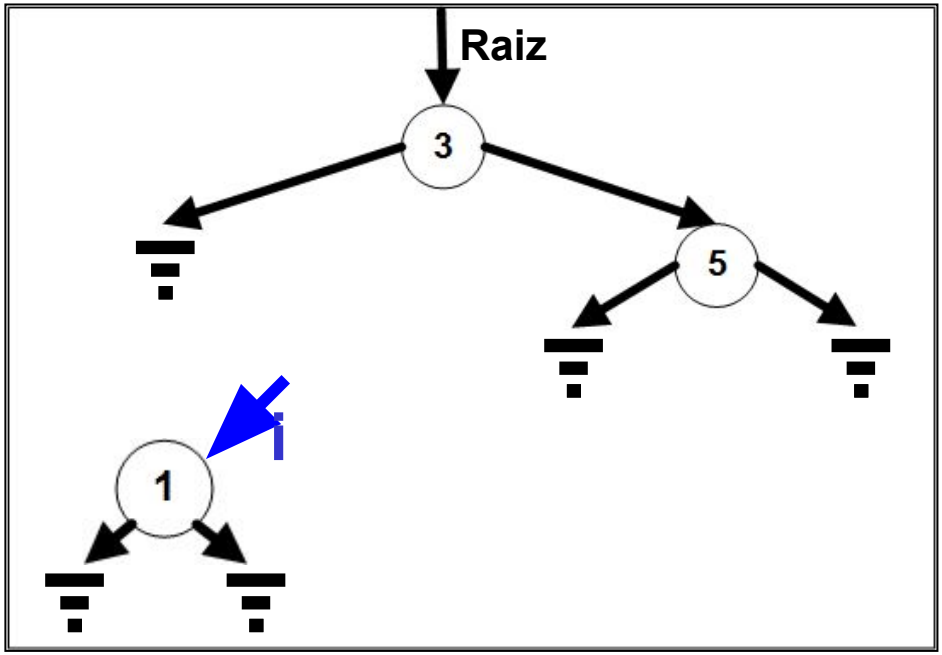




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

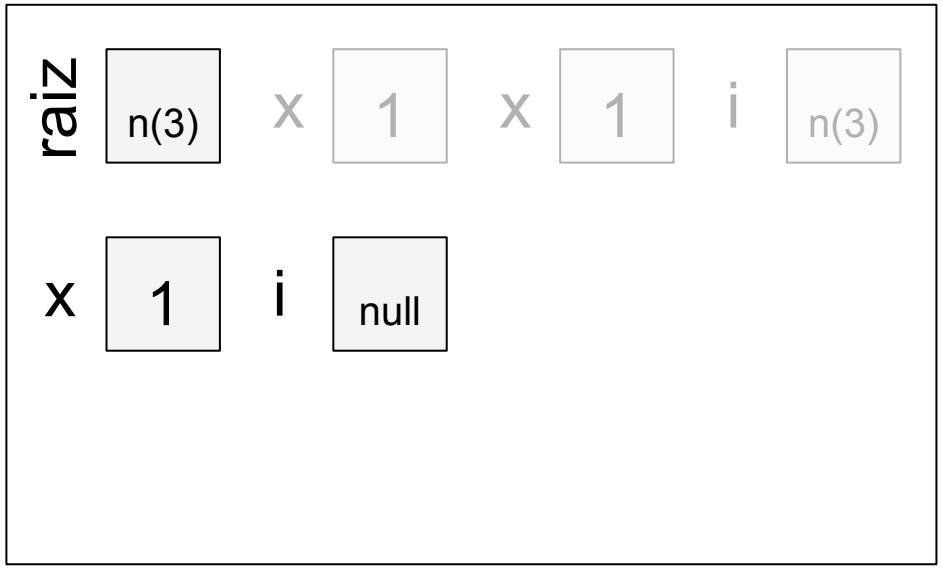


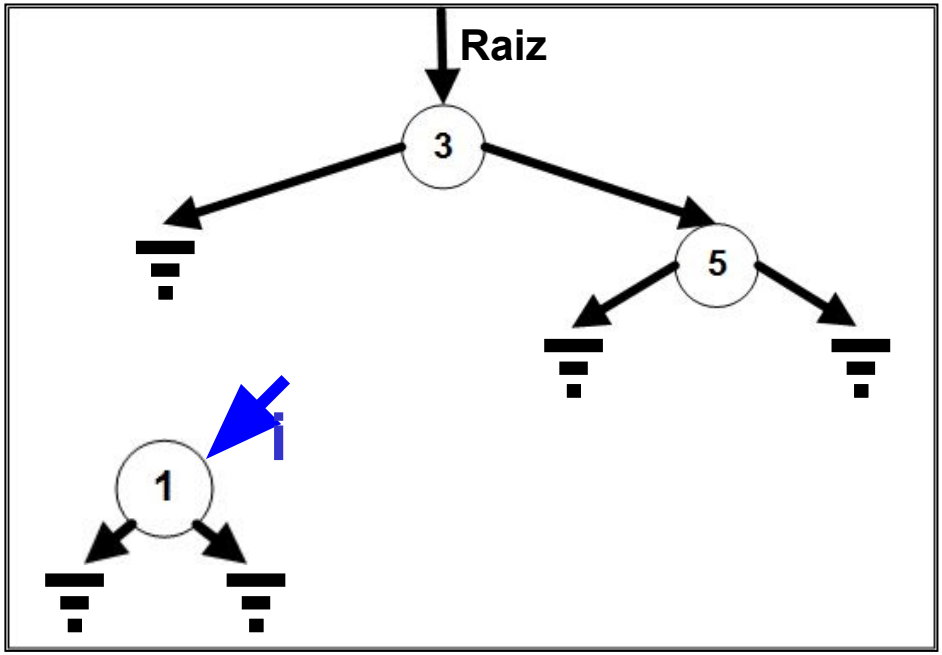


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

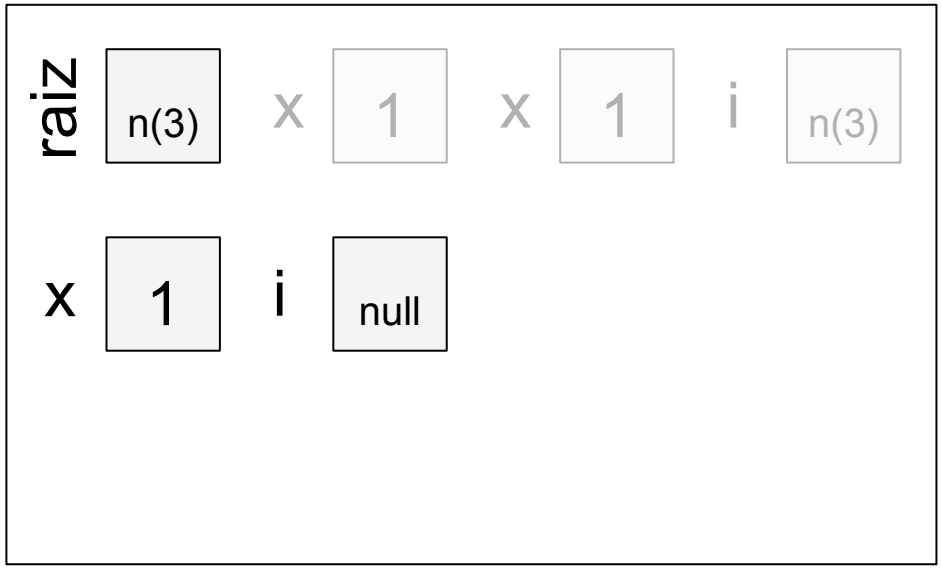
retorna o endereço de n(1)

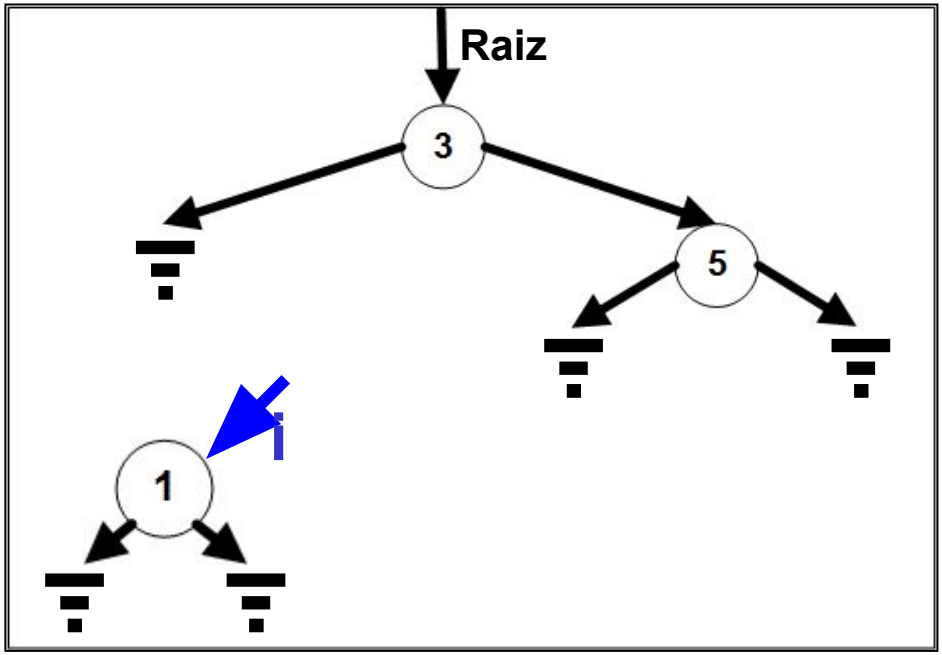




Inserção em Java com Retorno de Referência

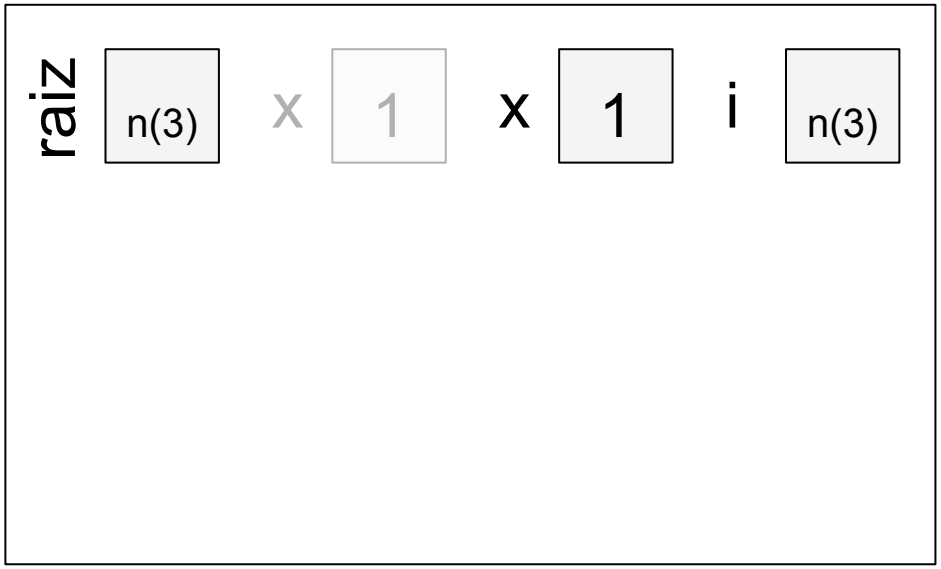
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

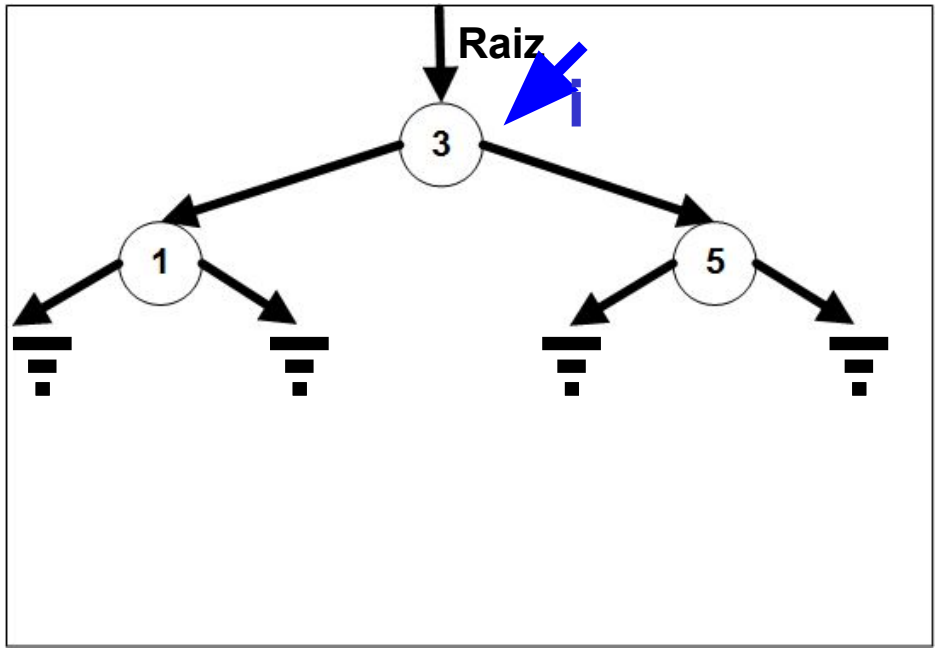




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

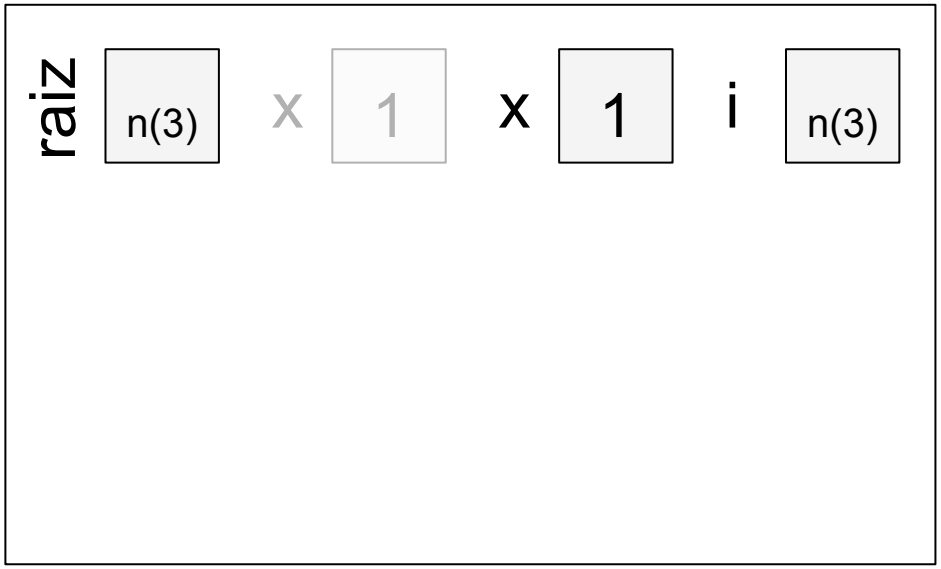


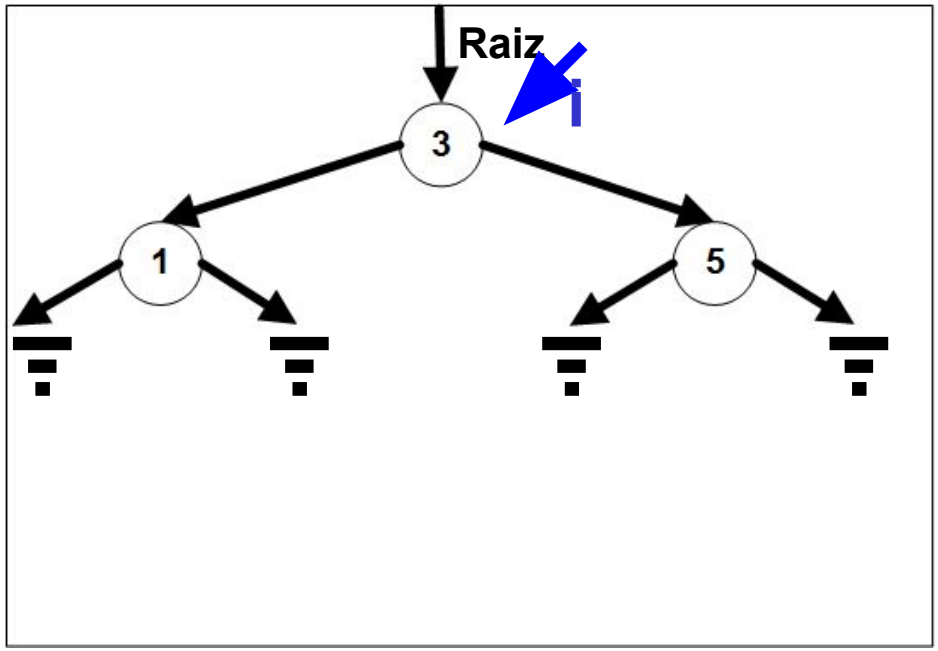


Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

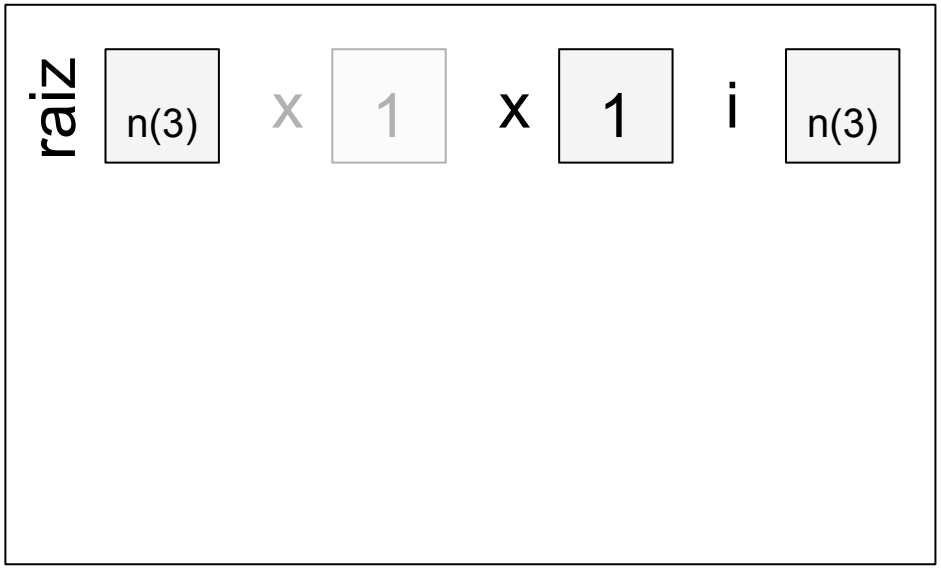
retorna o endereço de n(3)

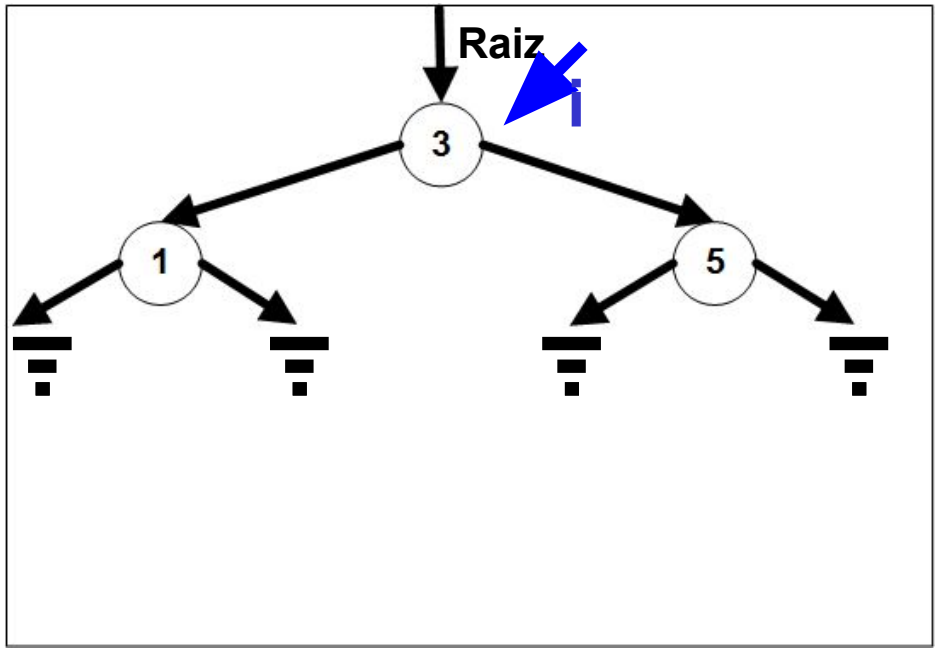




Inserção em Java com Retorno de Referência

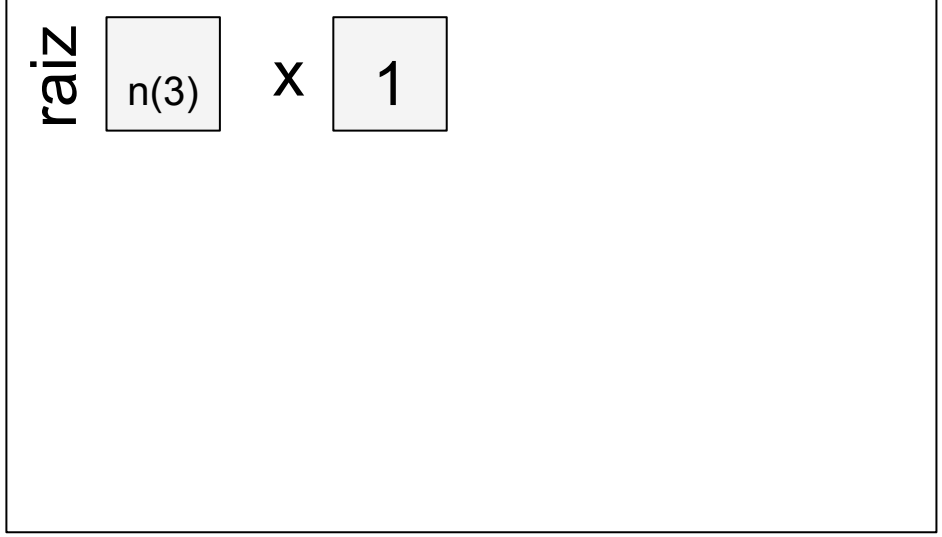
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

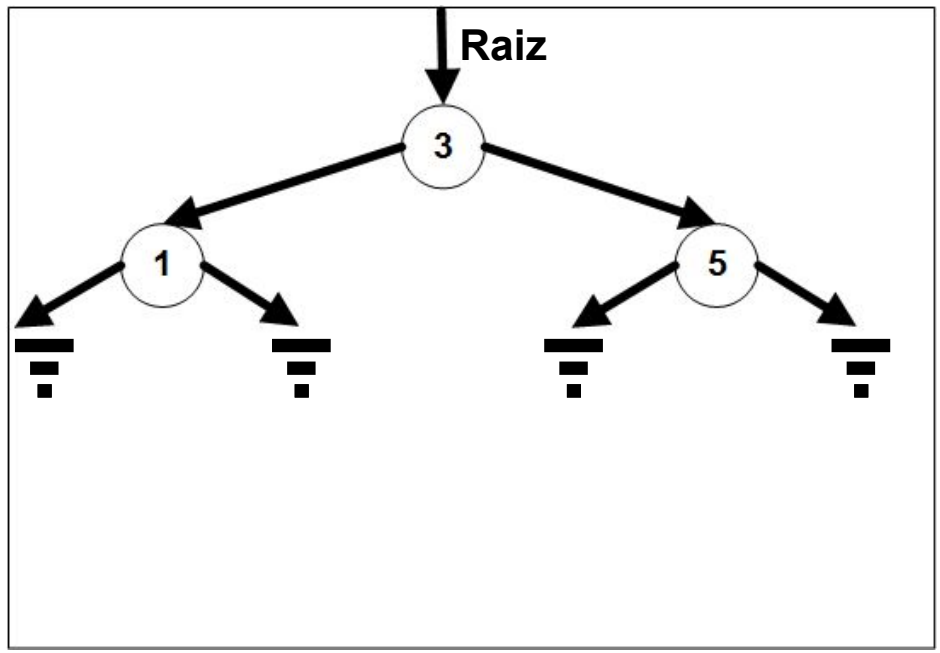




Inserção em Java com Retorno de Referência

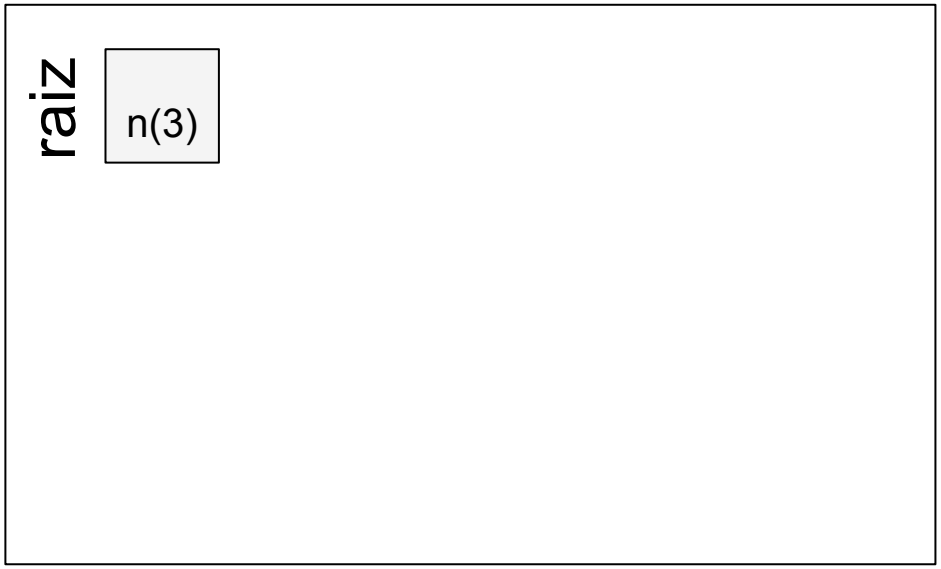
```
//Inserir 3, 5, 1, 8, 2, 4, 7, 6

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

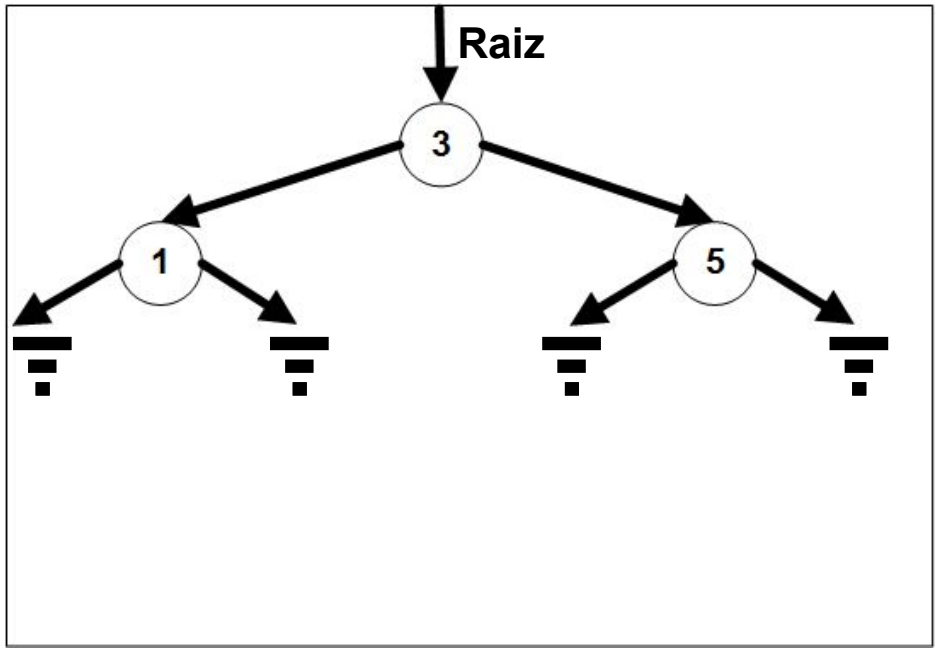




Inserção em Java com Retorno de Referência

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

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

No inserir(int x, No i) {
   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("Erro!");
   }
   return i;
}
```

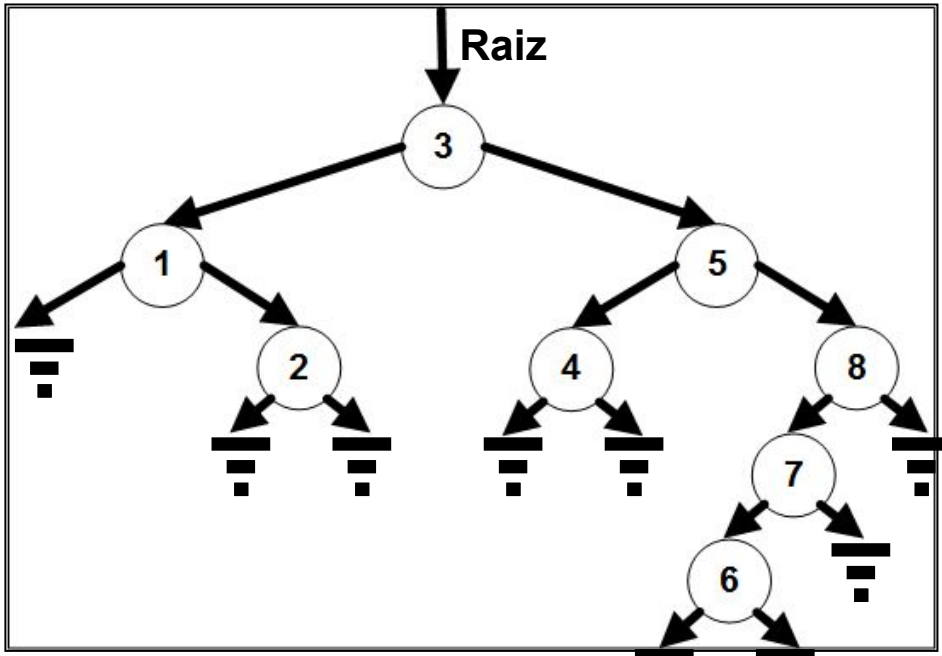




Inserção em Java com Retorno de Referência

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    void inserirPai(int x) {  }
    boolean pesquisar(int x) {   }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
    void remover(int x) {  }
}
```

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



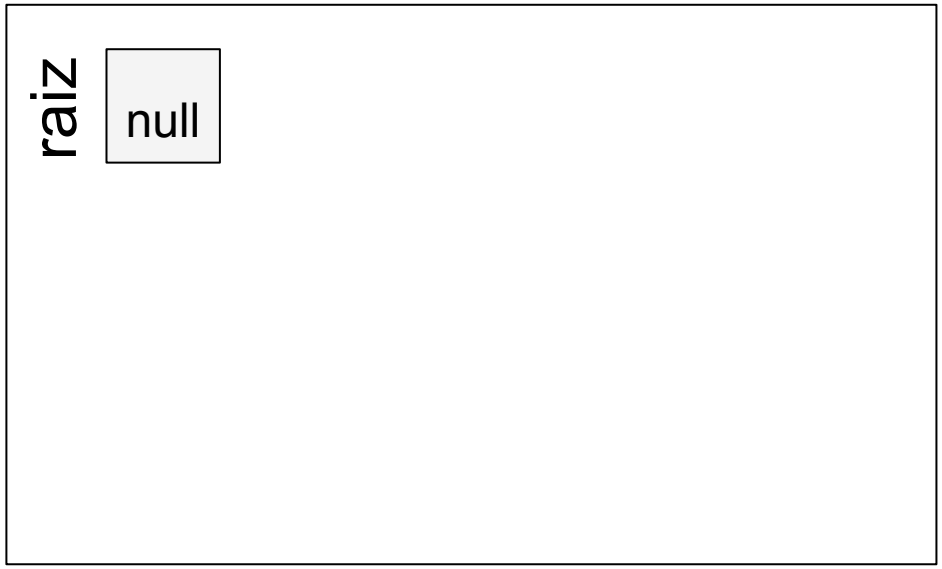
Agenda

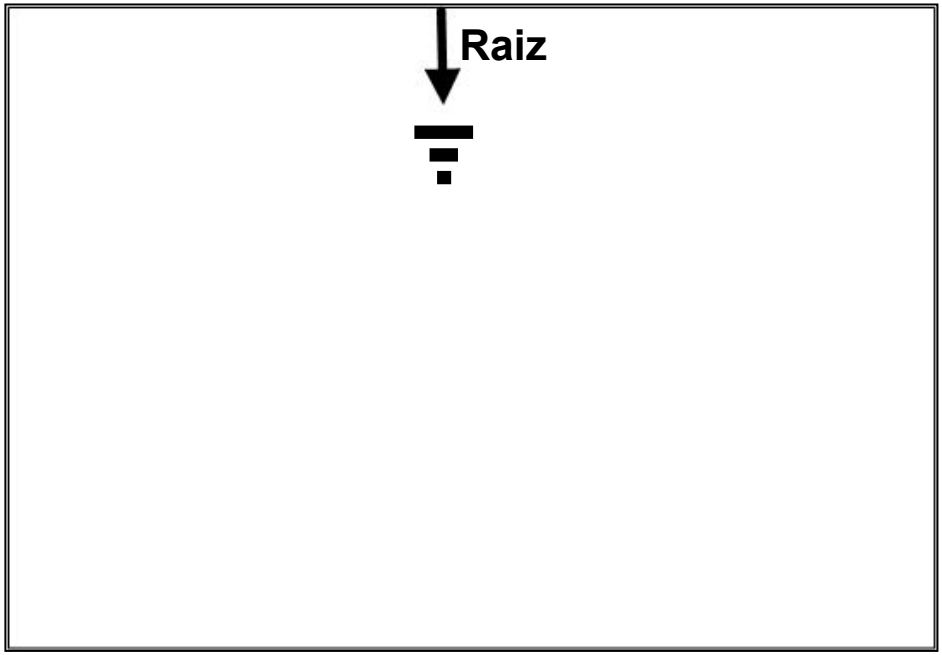
- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- Inserção em C com ponteiro
- Inserção em C++ com passa
- Estruturas híbridas

  - Funcionamento básico
  - Exemplo
  - Inserção em Java com retorno de referência
  - **Inserção em Java com passagem de pai**
  - Análise de Complexidade

Inserção em Java com Passagem de Pai

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    void inserirPai(int x) {  }
    boolean pesquisar(int x) {   }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
    void remover(int x) {  }
}
```

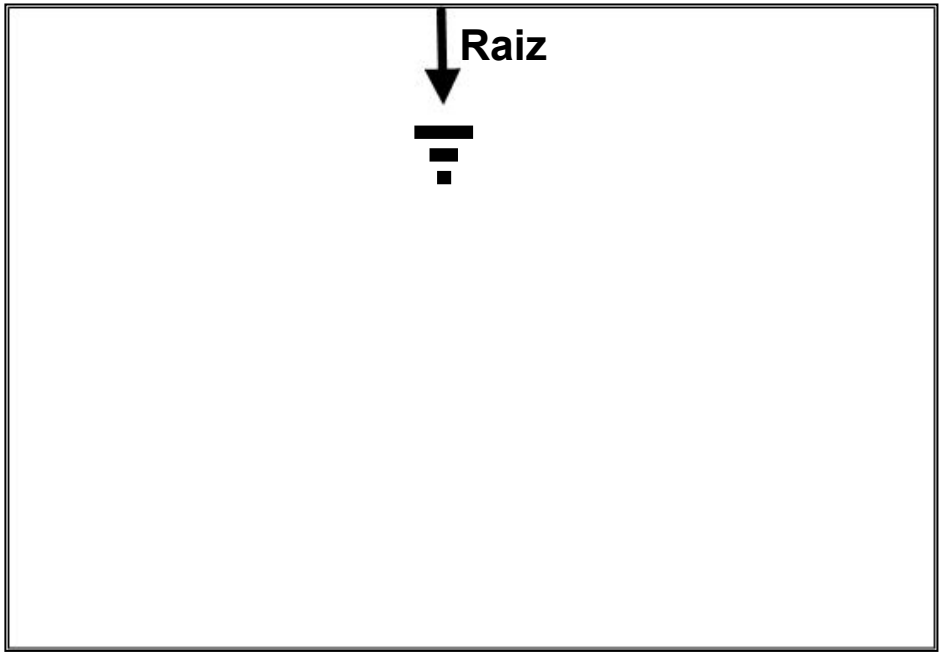




Vamos inserir os elementos
3, 5, 1 e 8
(várias chamadas do inserir)

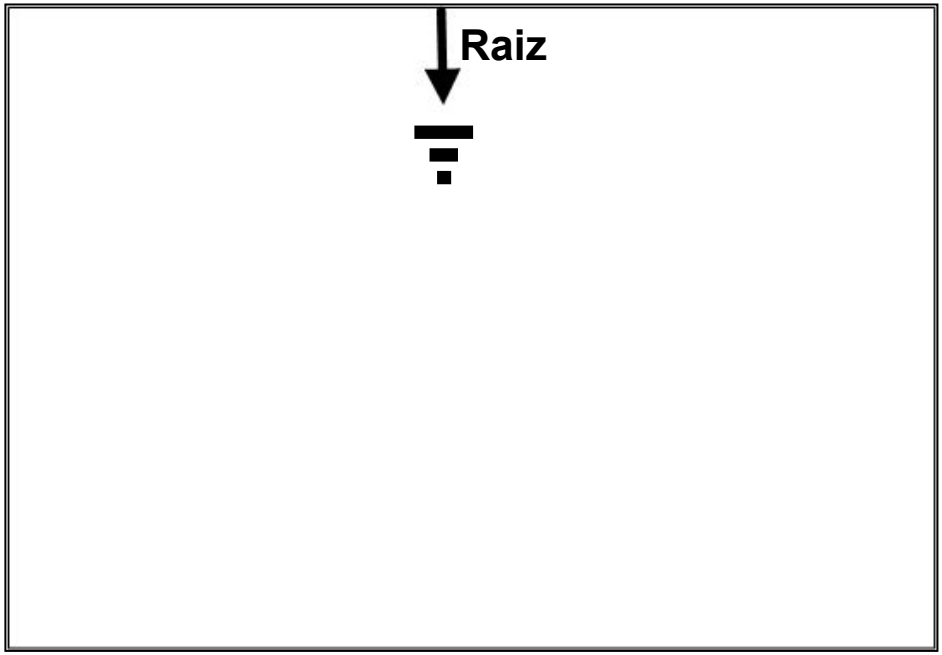
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■  ■  ■

}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■  ■  ■

}
```

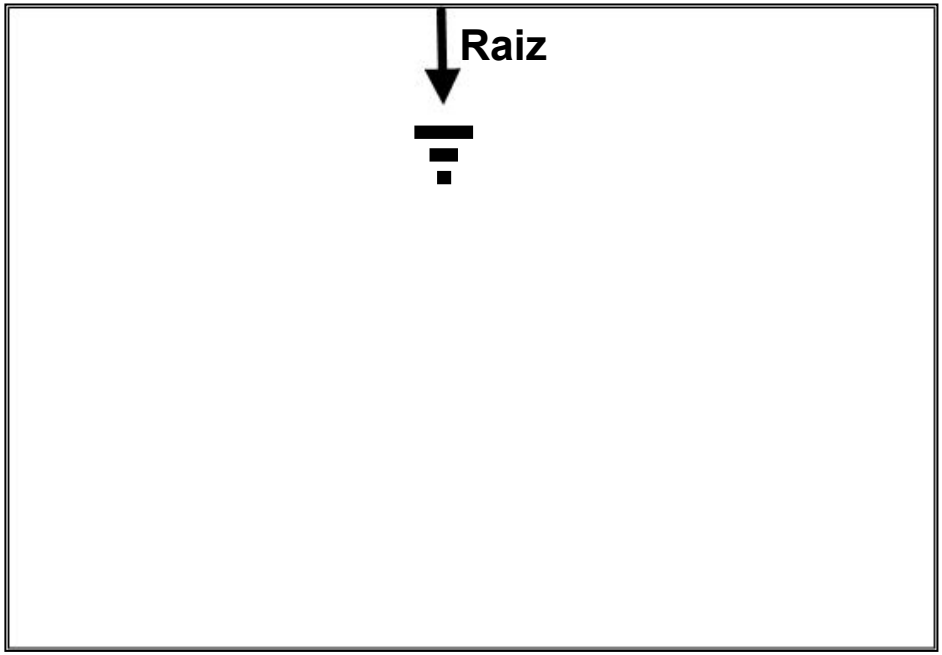


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

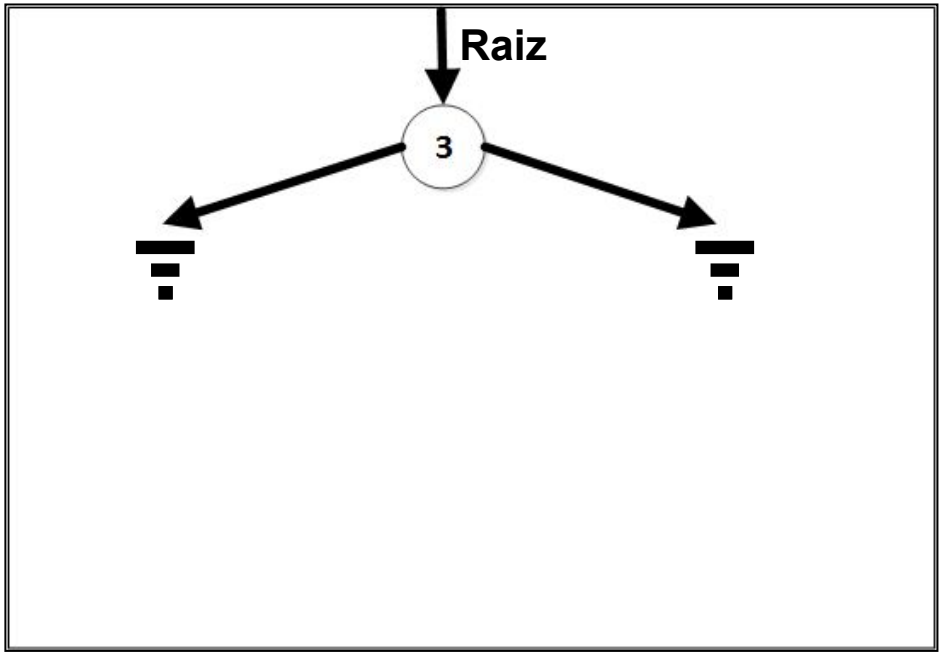
true: null == null

```
void inserirPai(int x, No i, No pai)  {

   ■  ■  ■

}
```



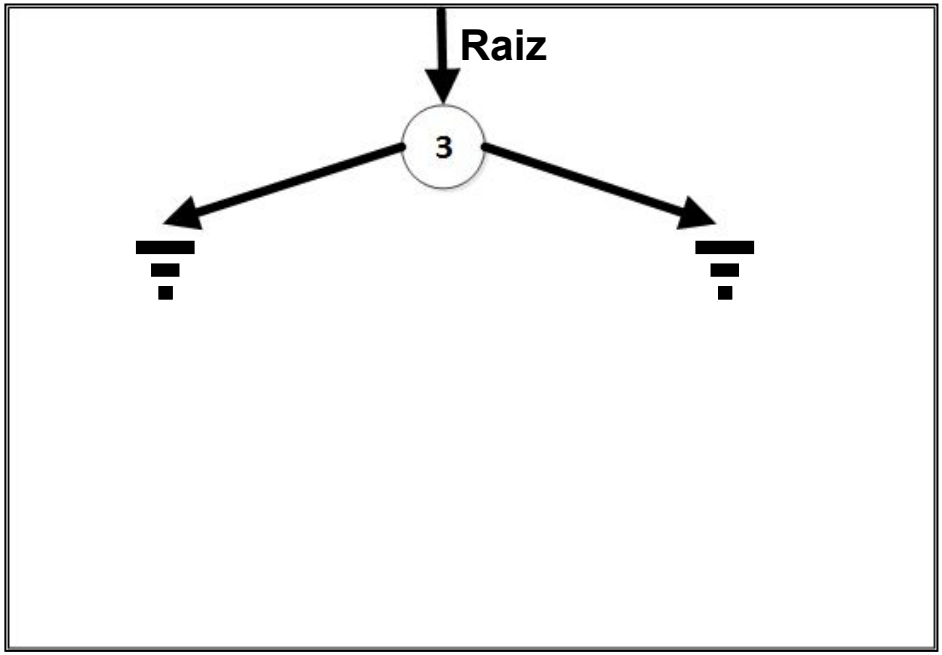
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■ ■ ■

}
```



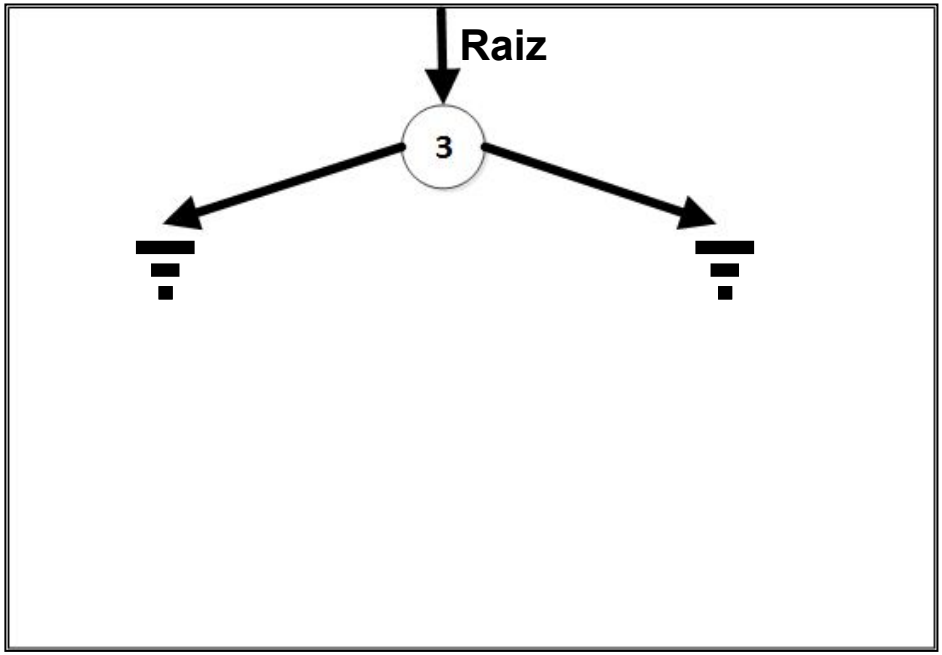
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■  ■  ■

}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■ ■ ■

}
```

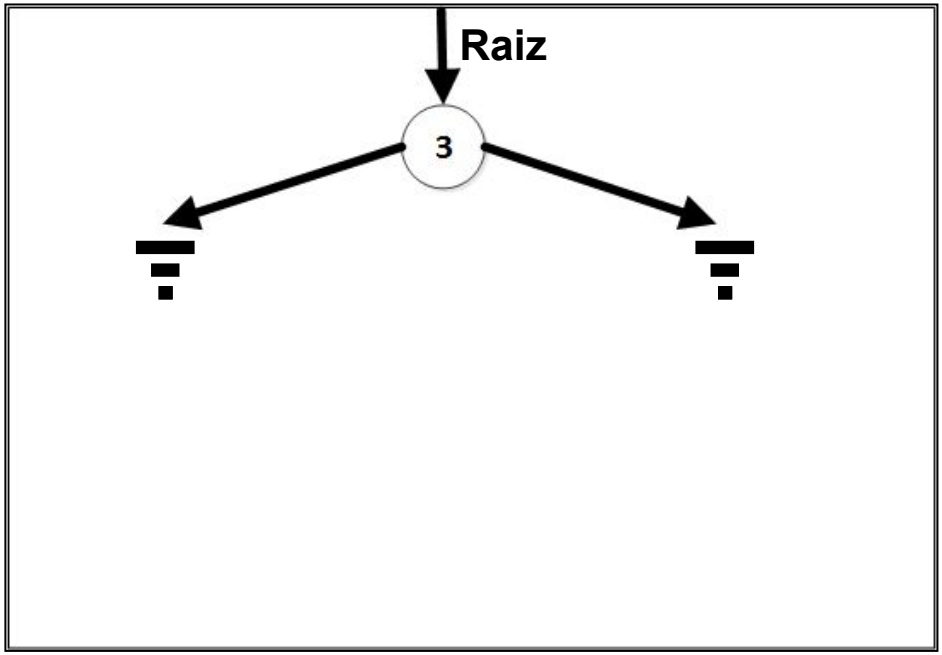


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

false: n(3) ≠ null

```
void inserirPai(int x, No i, No pai)  {

   ■  ■  ■

}
```

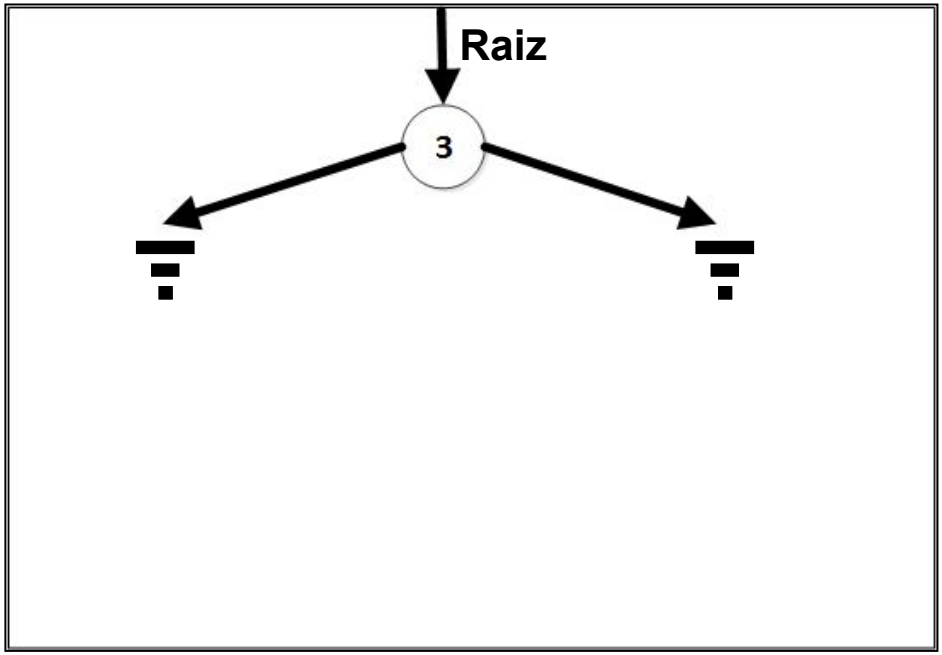


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

false: 5 < 3

```
void inserirPai(int x, No i, No pai)  {

   ■  ■  ■

}
```

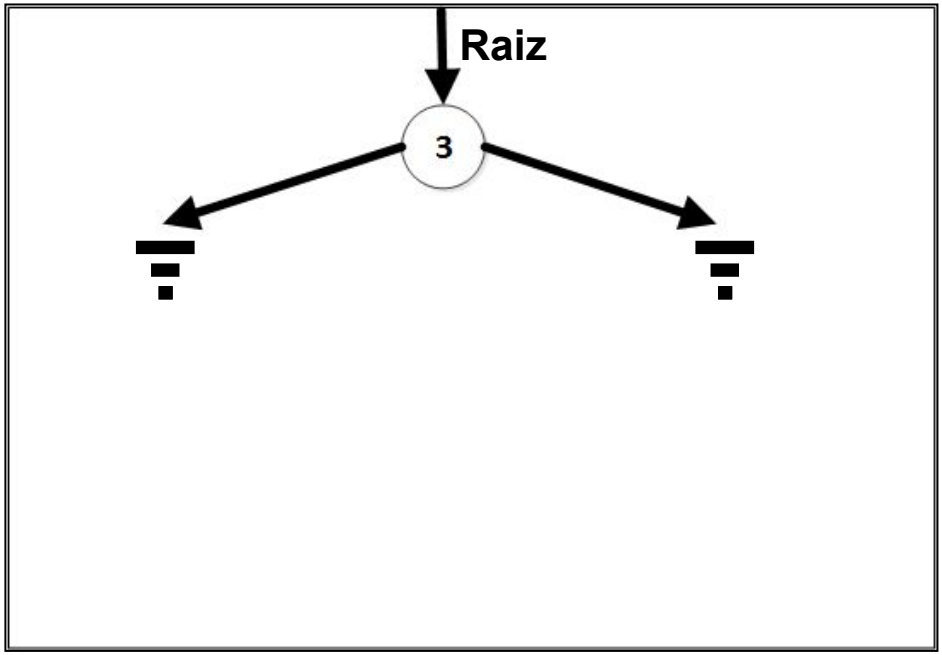


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

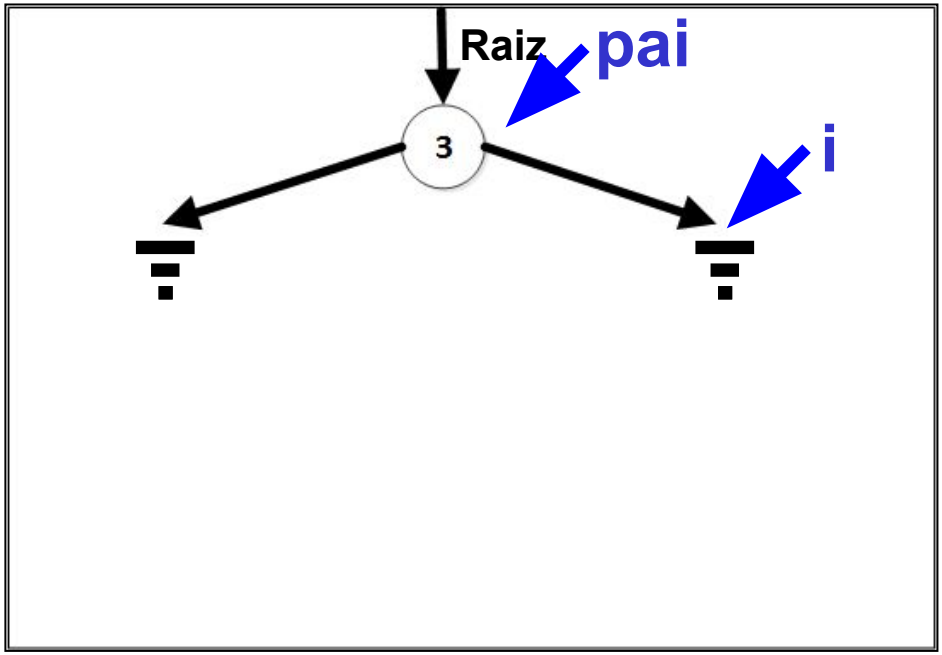
true: 5 > 3

```
 void inserirPai(int x, No i, No pai)  {

                  ■  ■  ■

}
```



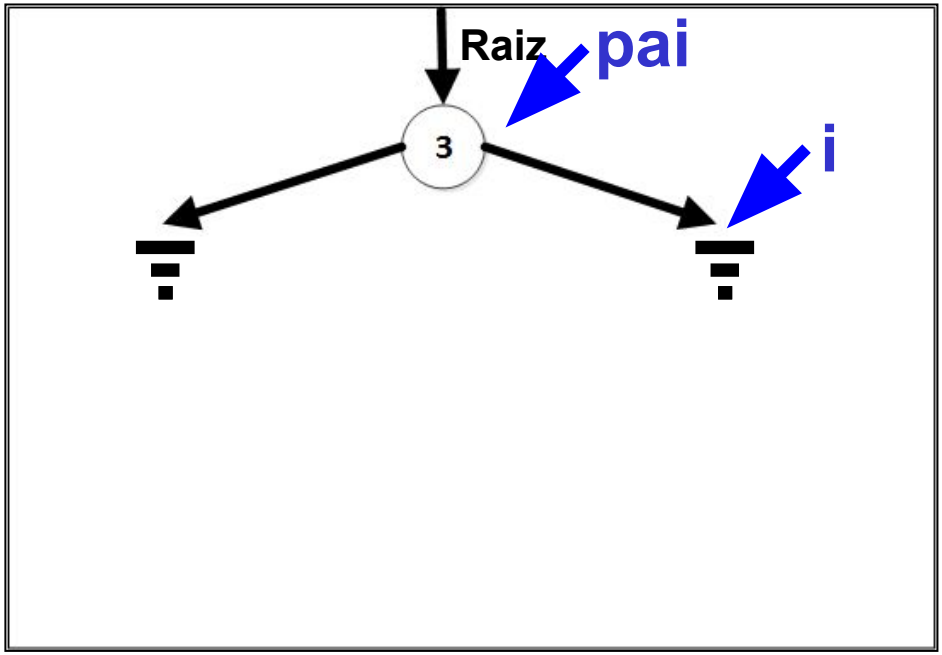
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                  ■  ■  ■

}
```



Inserção em Java com Passagem de Pai

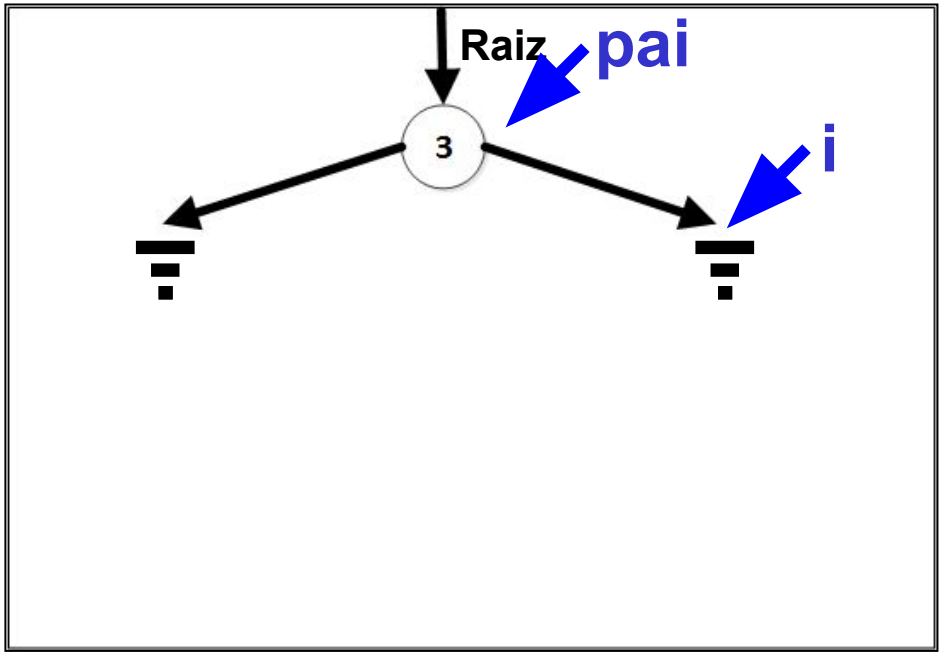
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
   if (i == null) {
      If (x < pai.elemento){
         pai.esq = new No(x);
      } else {
         pai.dir = new No(x);
      }
   } else if (x < i.elemento) {
      inserirPai(x, i.esq, i);
   } else if (x > i.elemento)  {
      inserirPai(x, i.dir, i);
   } else {
      throw new(“Erro!”);
   }
}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
     If (x < pai.elemento){
        pai.esq = new No(x);
     } else {
        pai.dir = new No(x);
     }
  } else if (x < i.elemento) {
     inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
     inserirPai(x, i.dir, i);
  } else {
     throw new("Erro!");
  }
}
```

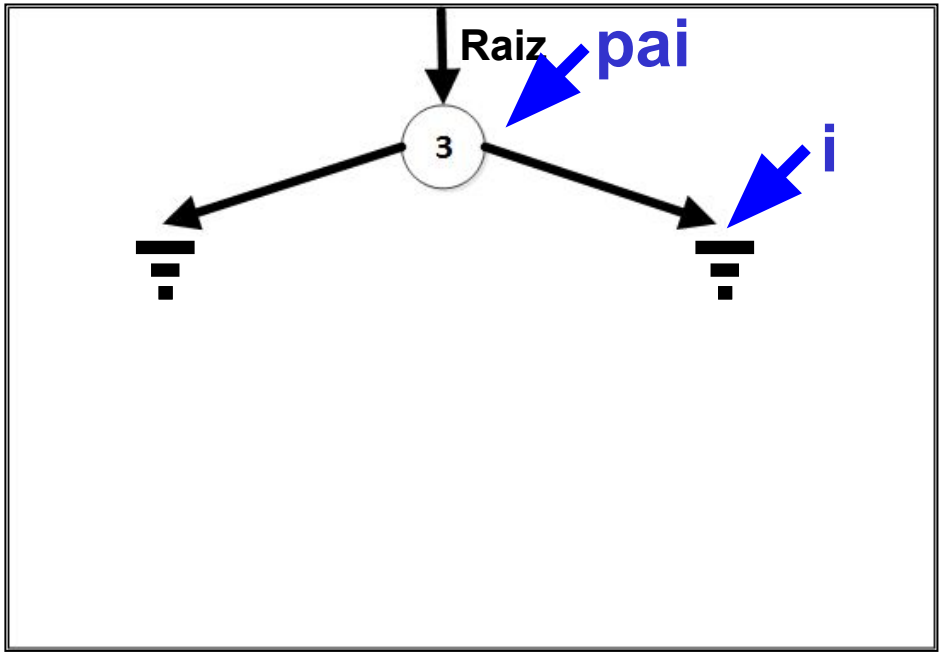
true: null == null



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```

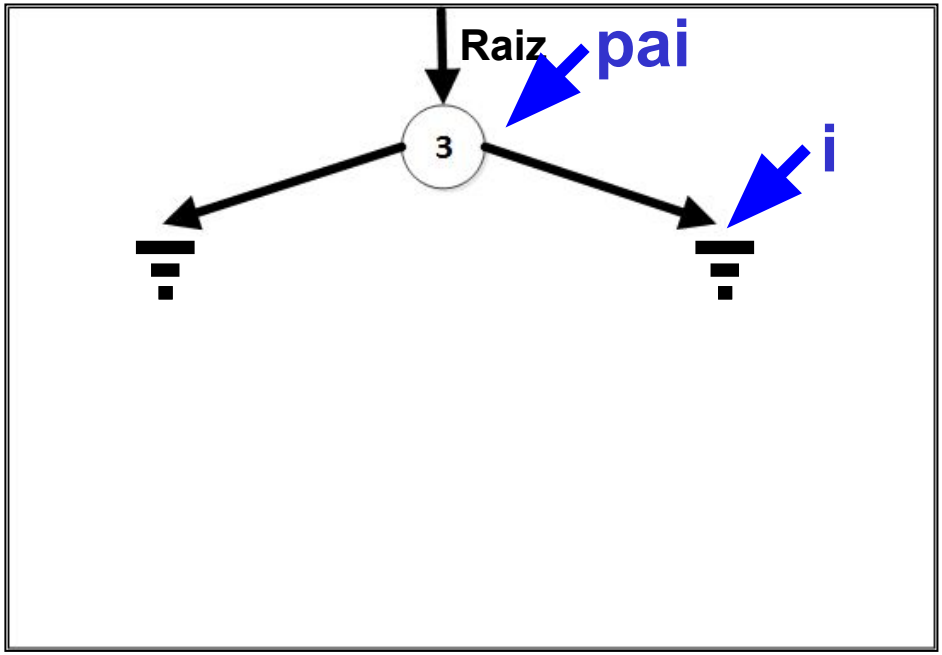
false: 5 < 3



Inserção em Java com Passagem de Pai

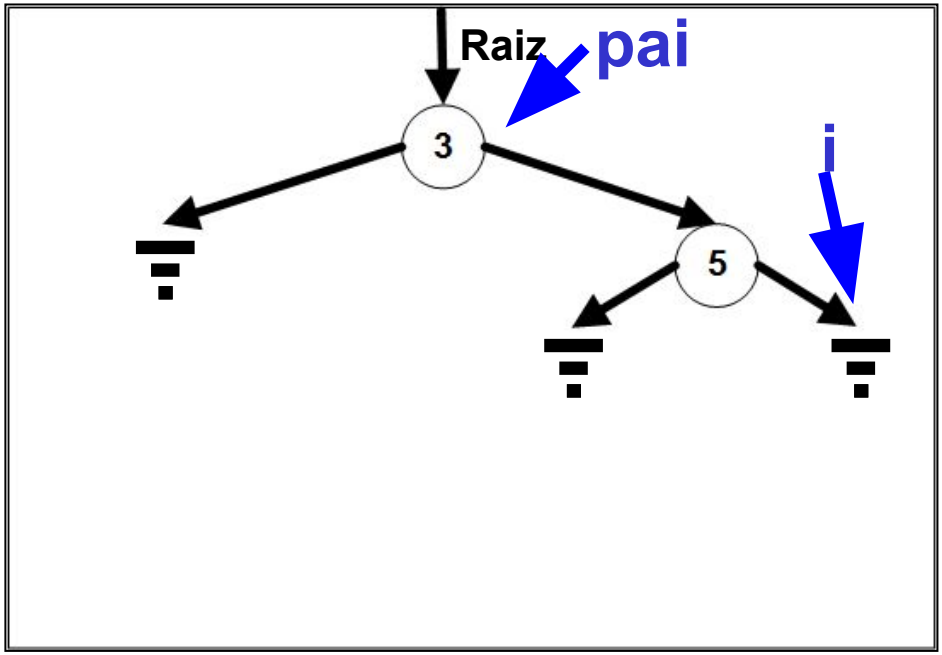
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```

true: 5 > 3



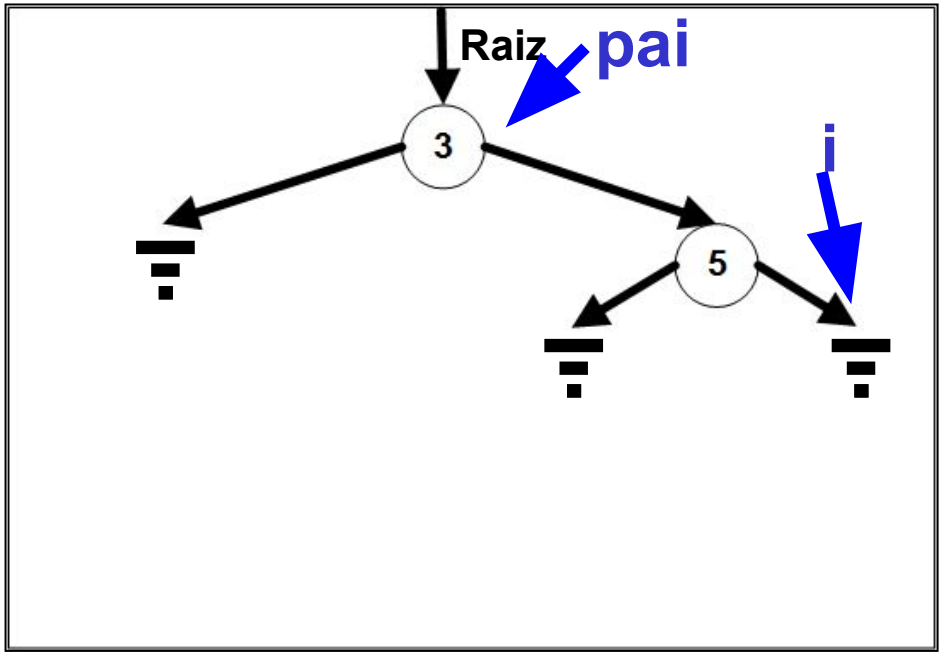
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```



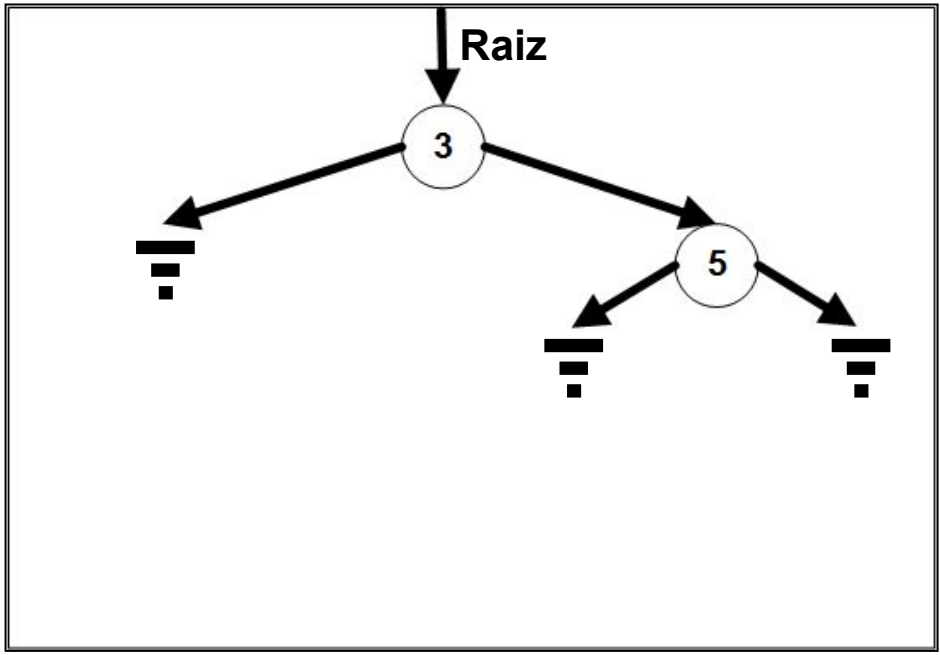
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
  if (i == null) {
     If (x < pai.elemento){
        pai.esq = new No(x);
     } else {
        pai.dir = new No(x);
     }
  } else if (x < i.elemento) {
     inserirPai(x, i.esq, i);
  } else if (x > i.elemento)  {
     inserirPai(x, i.dir, i);
  } else {
     throw new("Erro!");
  }
}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■ ■ ■

}
```

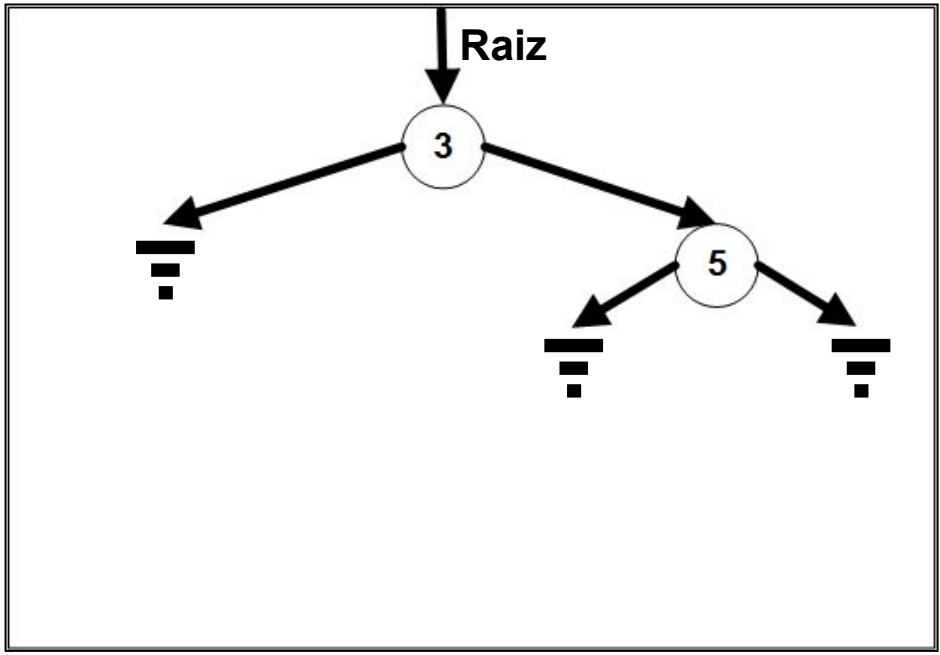


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

false: n(3) ≠ null

```
void inserirPai(int x, No i, No pai)  {

   ■  ■  ■

}
```

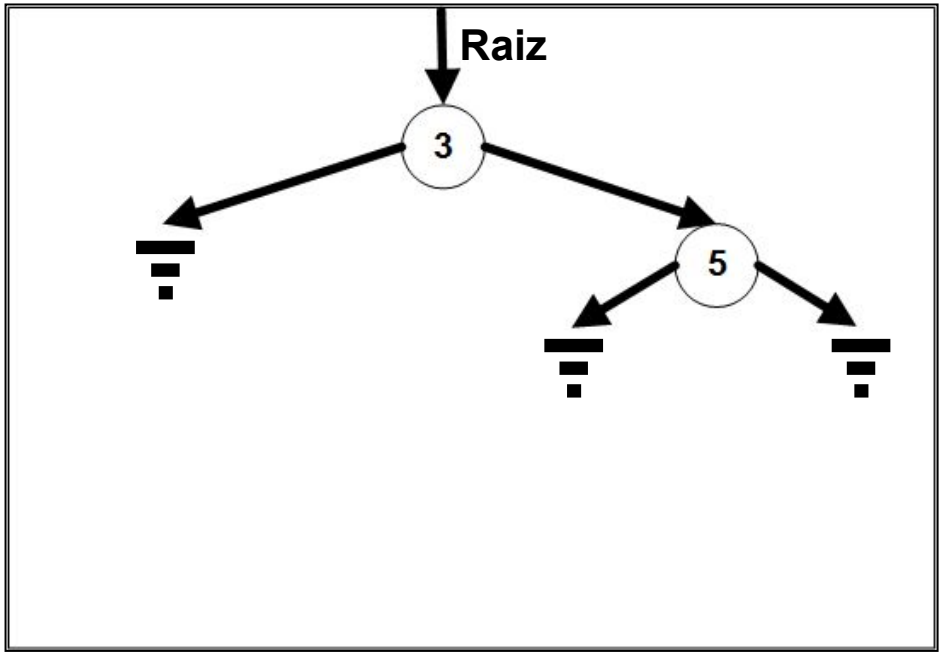


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

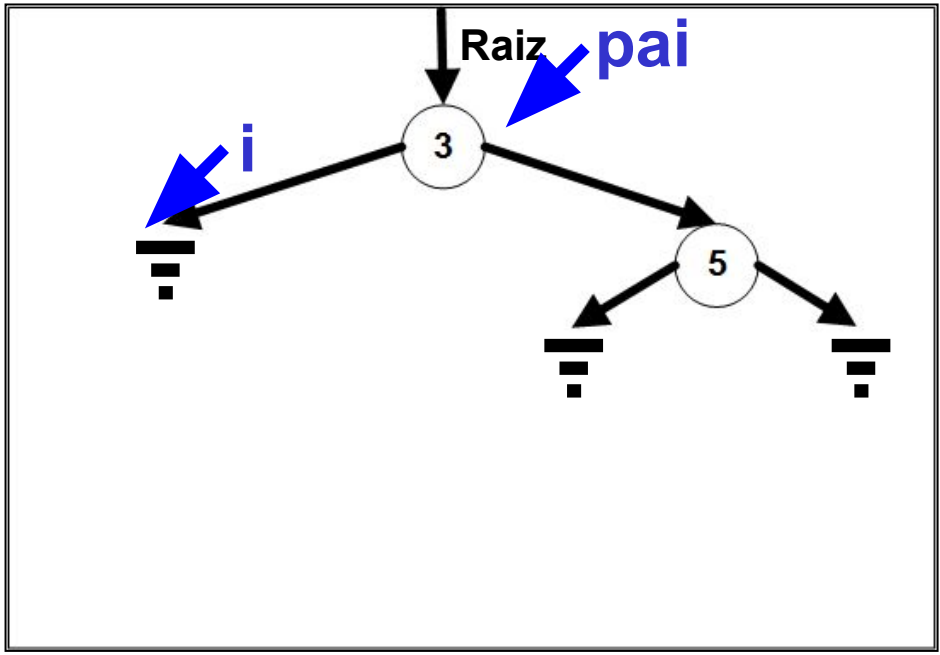
true: 1 < 3

```
void inserirPai(int x, No i, No pai)  {

}
```



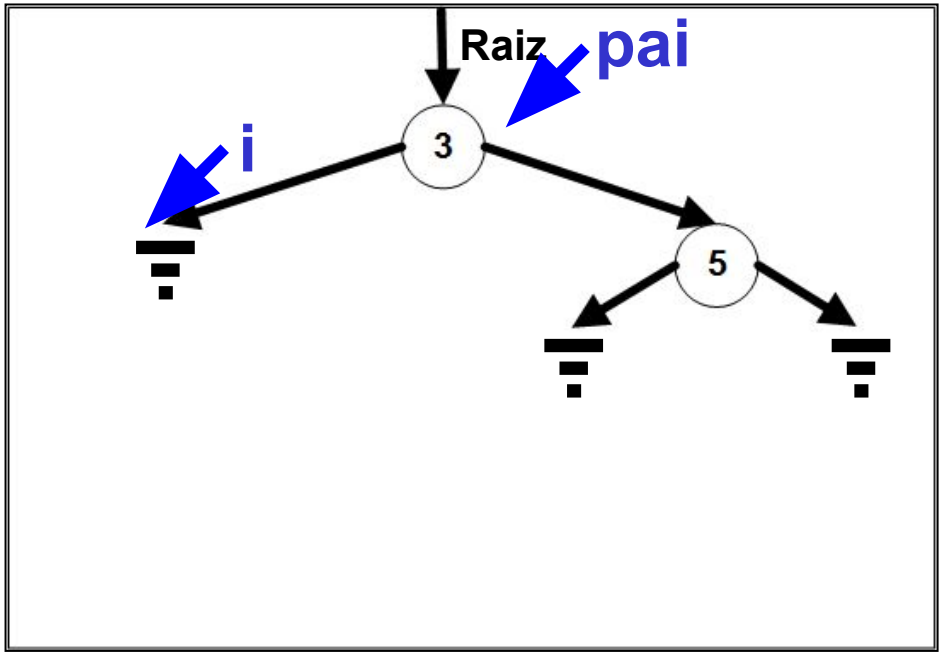
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■ ■ ■

}
```



Inserção em Java com Passagem de Pai

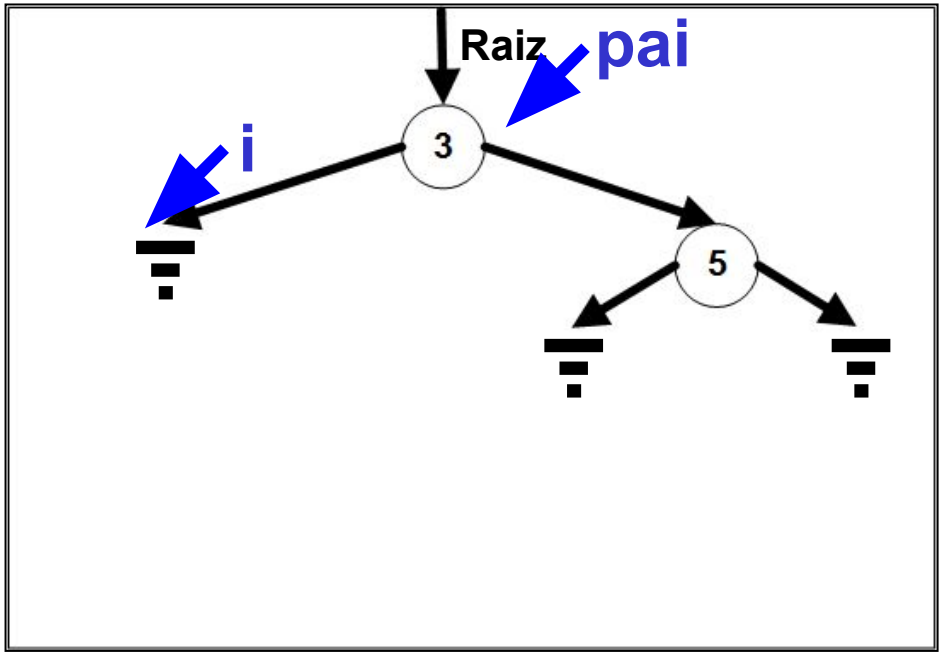
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
   if (i == null) {
      If (x < pai.elemento){
         pai.esq = new No(x);
      } else {
         pai.dir = new No(x);
      }
   } else if (x < i.elemento) {
      inserirPai(x, i.esq, i);
   } else if (x > i.elemento)  {
      inserirPai(x, i.dir, i);
   } else {
      throw new(“Erro!”);
   }
}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
     If (x < pai.elemento){
        pai.esq = new No(x);
     } else {
        pai.dir = new No(x);
     }
  } else if (x < i.elemento) {
     inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
     inserirPai(x, i.dir, i);
  } else {
     throw new("Erro!");
  }
}
```

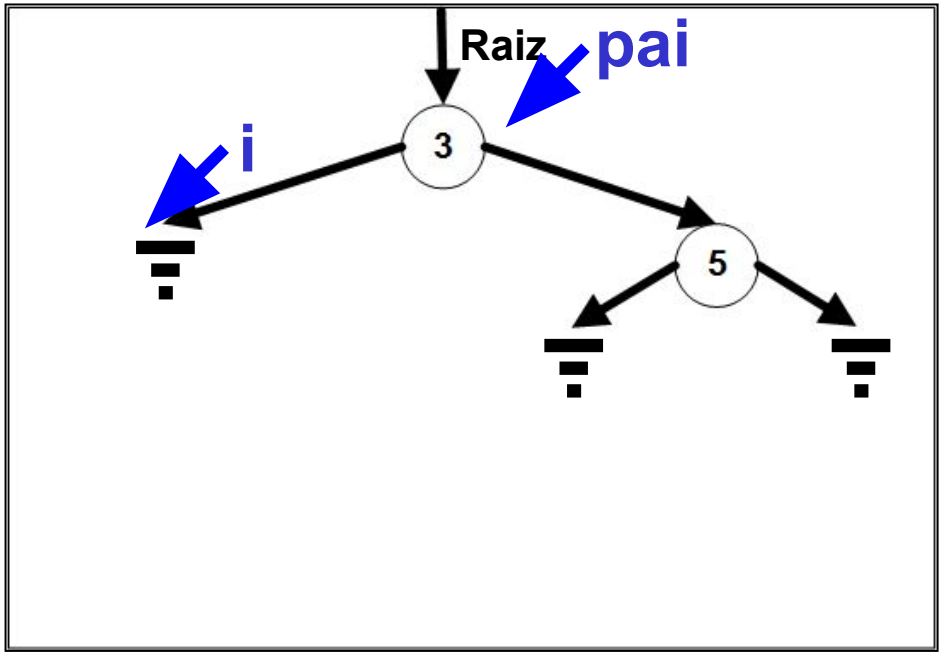
true: null == null



Inserção em Java com Passagem de Pai

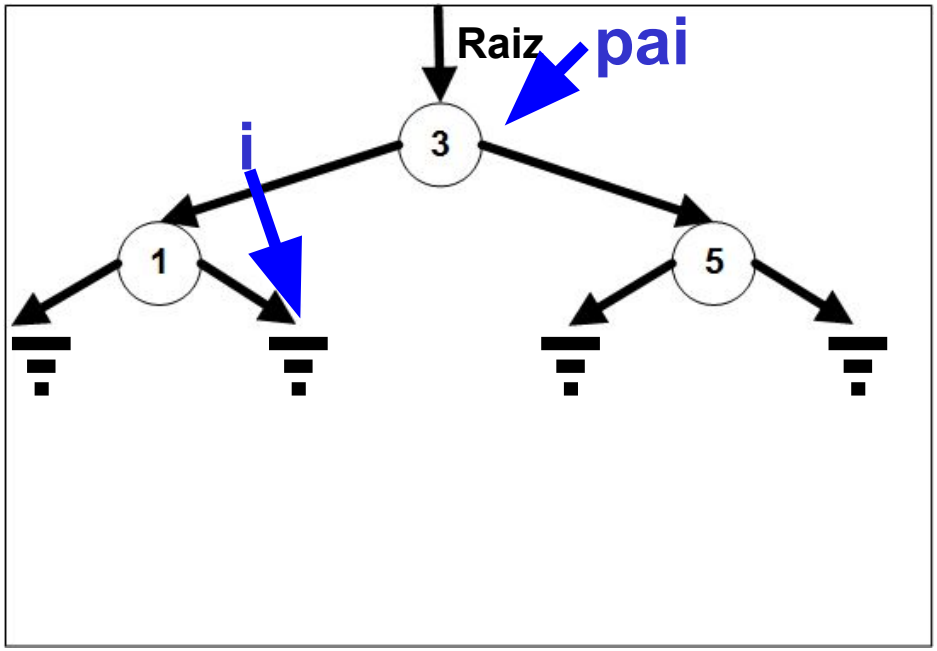
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```

true: 1 < 3



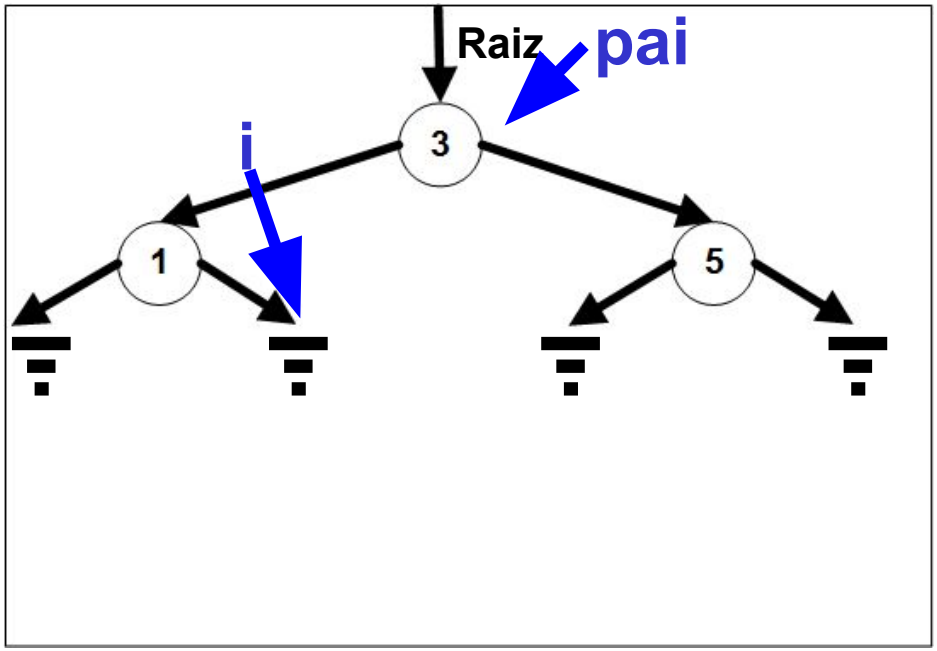
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```



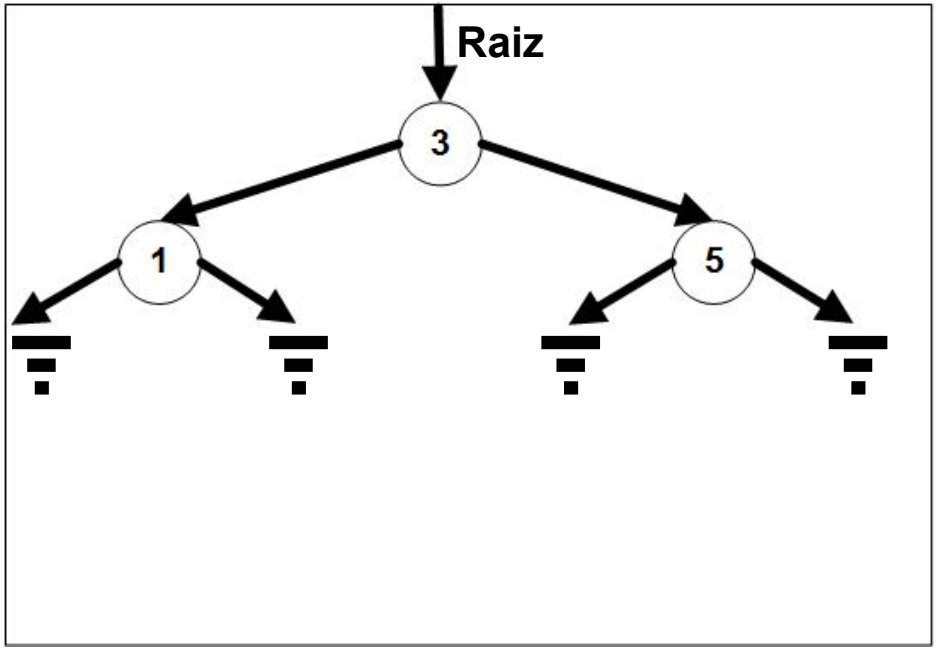
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento)  {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```



Inserção em Java com Passagem de Pai

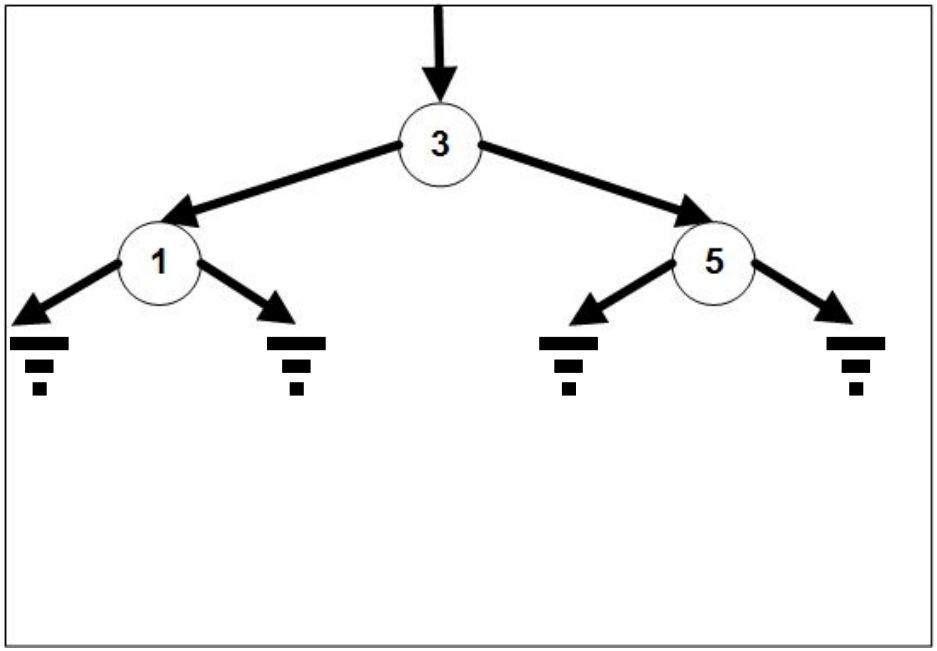
```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■ ■ ■

}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

void inserirPai(int x, No i, No pai)  {

}
```

false: n(3) ≠ null

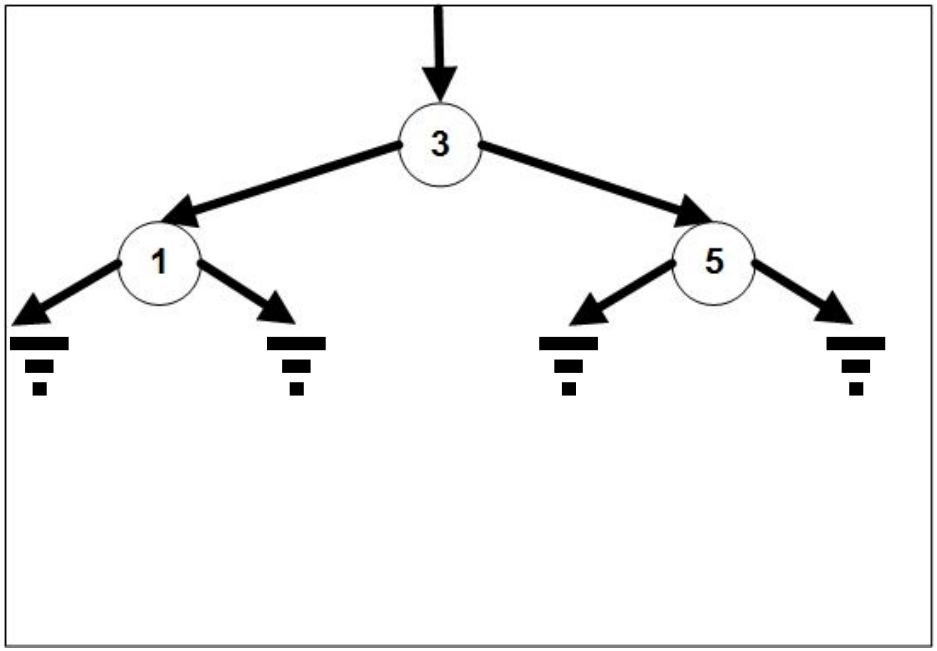


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

false: 8 < 3

```
void inserirPai(int x, No i, No pai)  {

   ■  ■  ■

}
```

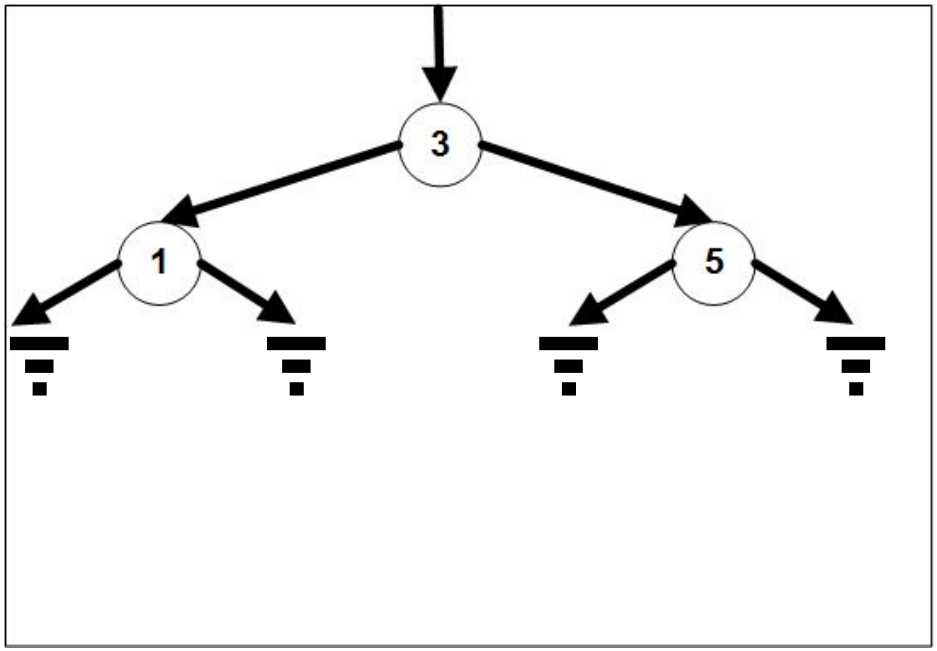


Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}
```

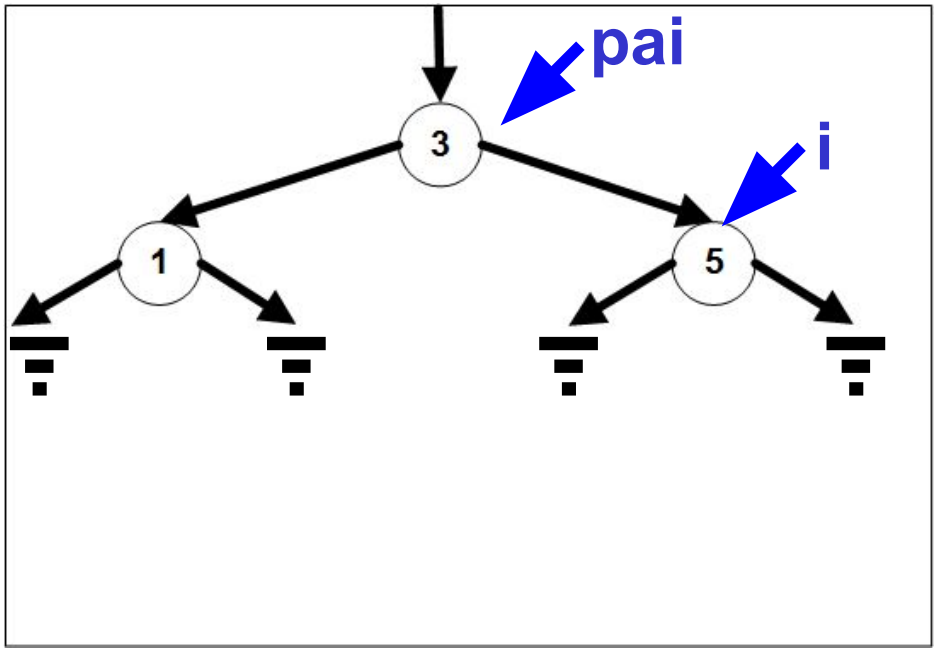
true: 8 > 3

```
void inserirPai(int x, No i, No pai)  {

   ■  ■  ■

}
```



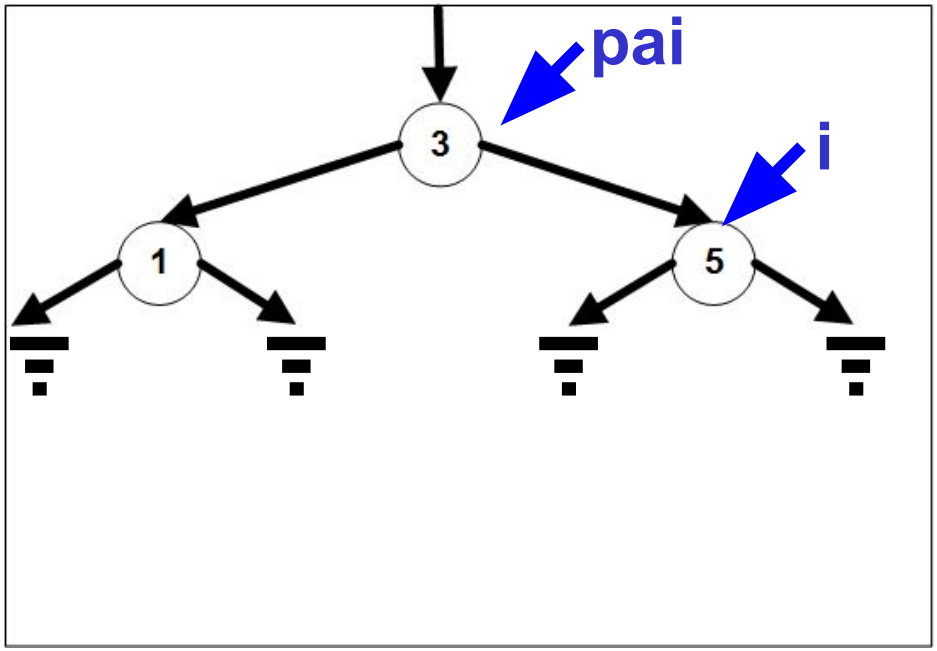
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {
      throw new("Erro!");
   }
}

 void inserirPai(int x, No i, No pai)  {

                    ■ ■ ■

}
```



Inserção em Java com Passagem de Pai

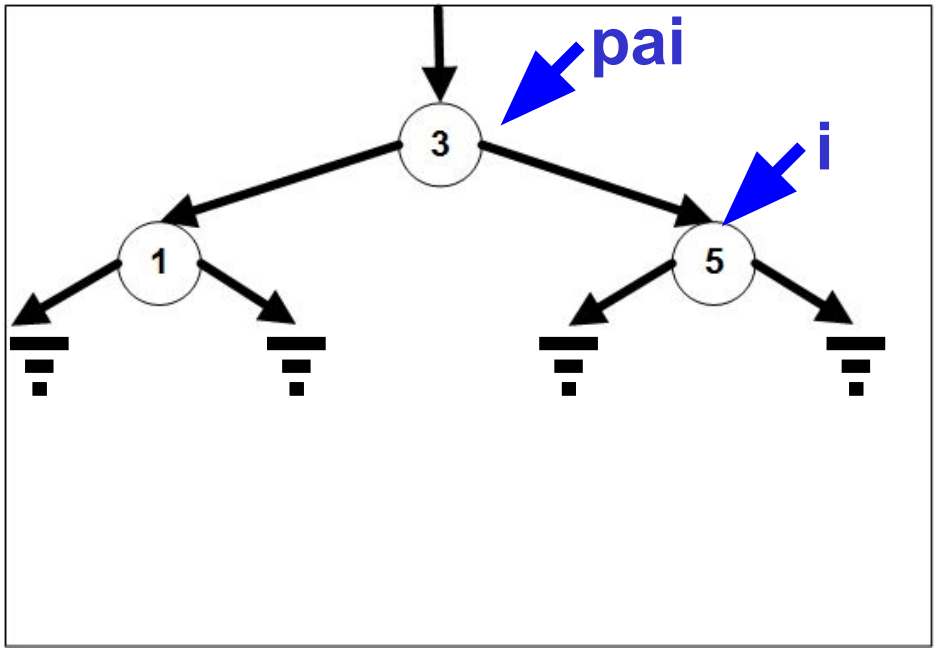
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
   if (i == null) {
      If (x < pai.elemento){
         pai.esq = new No(x);
      } else {
         pai.dir = new No(x);
      }
   } else if (x < i.elemento) {
      inserirPai(x, i.esq, i);
   } else if (x > i.elemento)  {
      inserirPai(x, i.dir, i);
   } else {
      throw new("Erro!");
   }
}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
   if (i == null) {
      If (x < pai.elemento){
         pai.esq = new No(x);
      } else {
         pai.dir = new No(x);
      }
   } else if (x < i.elemento) {
      inserirPai(x, i.esq, i);
   } else if (x > i.elemento)  {
      inserirPai(x, i.dir, i);
   } else {
      throw new("Erro!");
   }
}
```

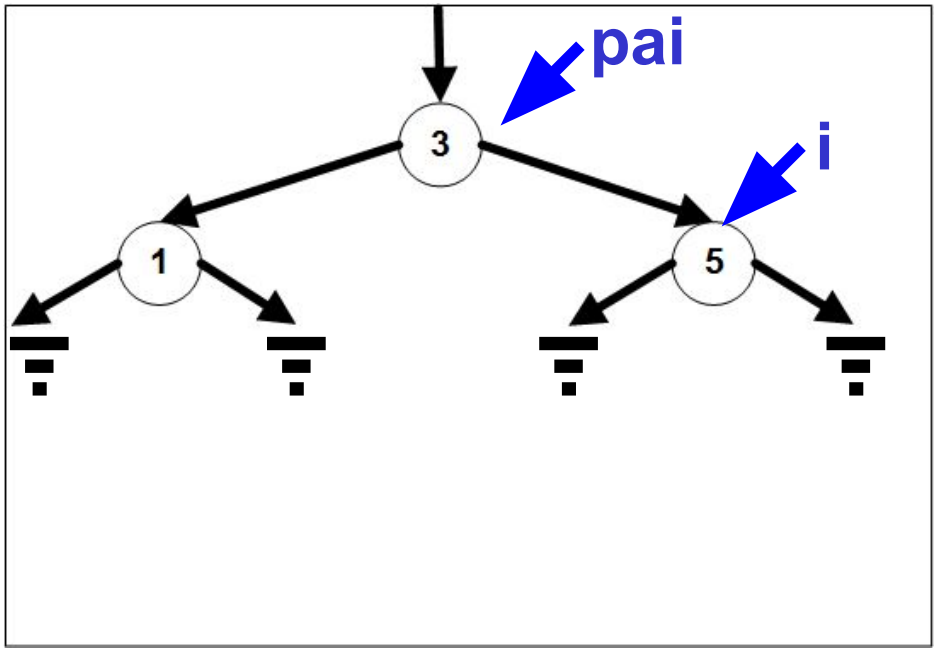
false: n(5) ≠ null



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```

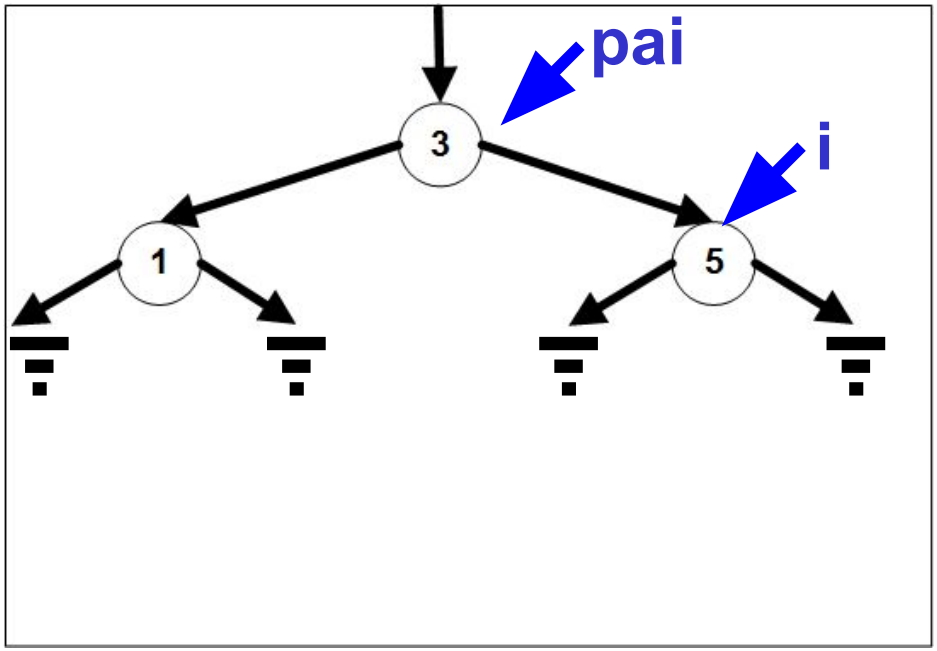
false: 8 < 5



Inserção em Java com Passagem de Pai

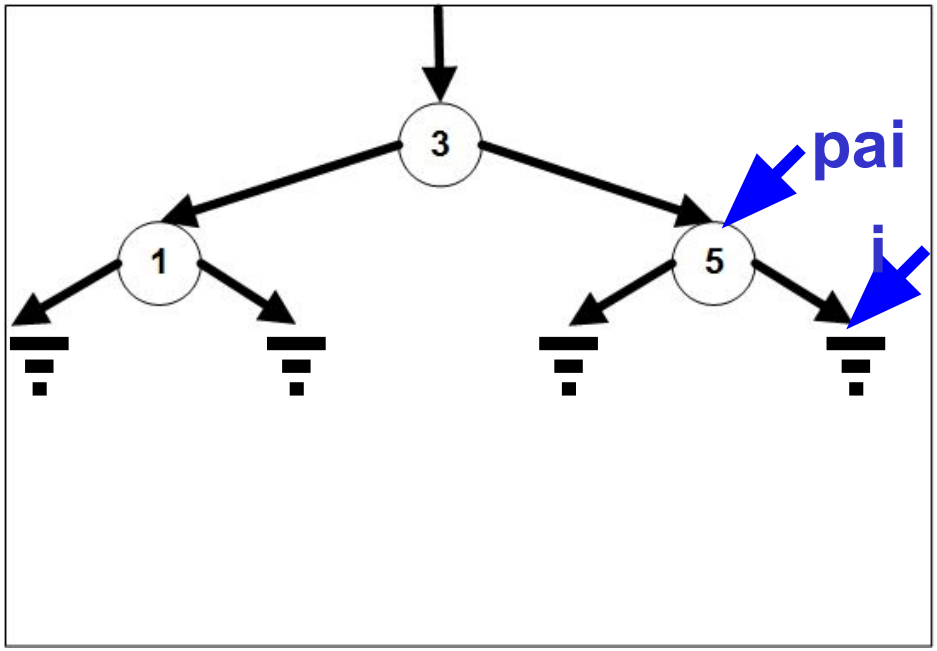
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```

true: 8 > 5



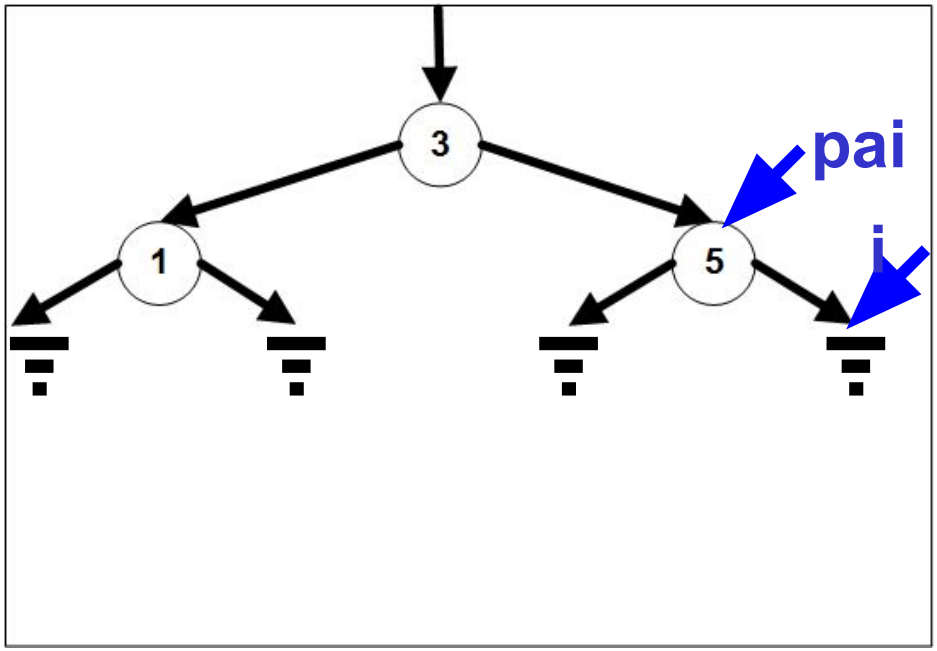
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```



Inserção em Java com Passagem de Pai

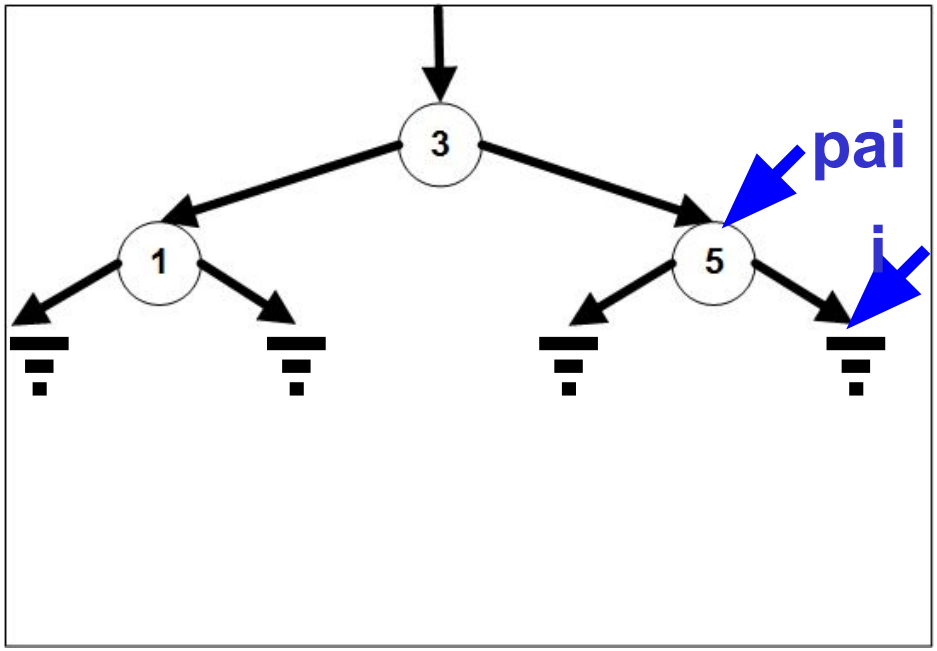
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
   if (i == null) {
      If (x < pai.elemento){
         pai.esq = new No(x);
      } else {
         pai.dir = new No(x);
      }
   } else if (x < i.elemento) {
      inserirPai(x, i.esq, i);
   } else if (x > i.elemento)  {
      inserirPai(x, i.dir, i);
   } else {
      throw new(“Erro!”);
   }
}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```

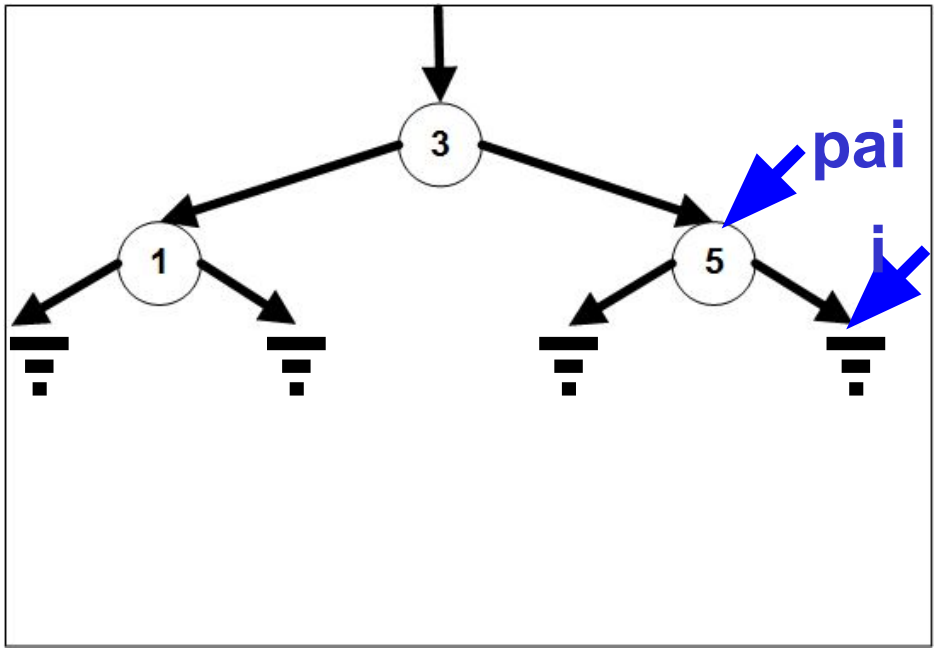
true: null == null



Inserção em Java com Passagem de Pai

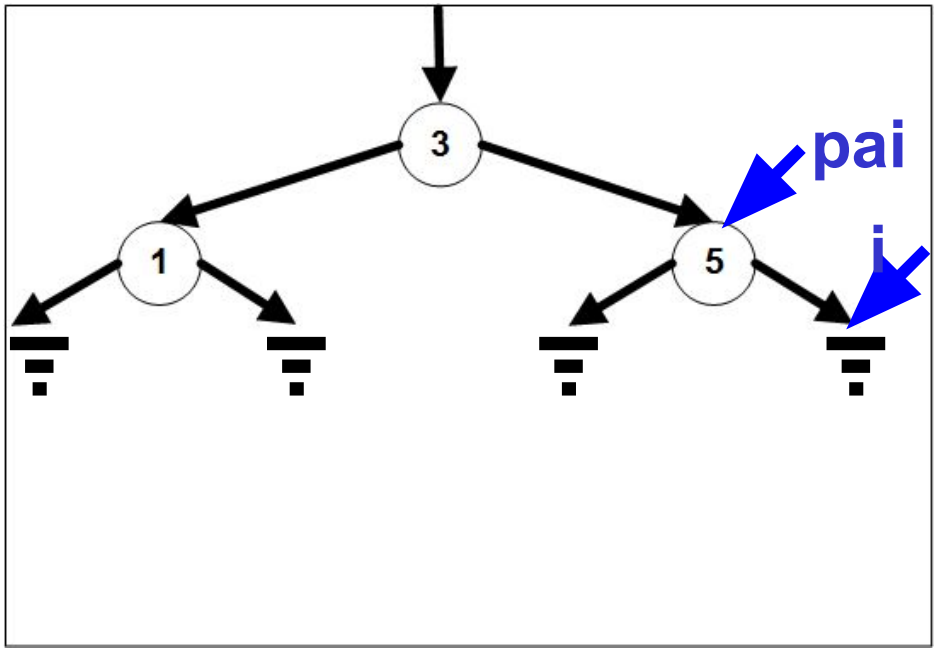
```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```

false: 8 < 5



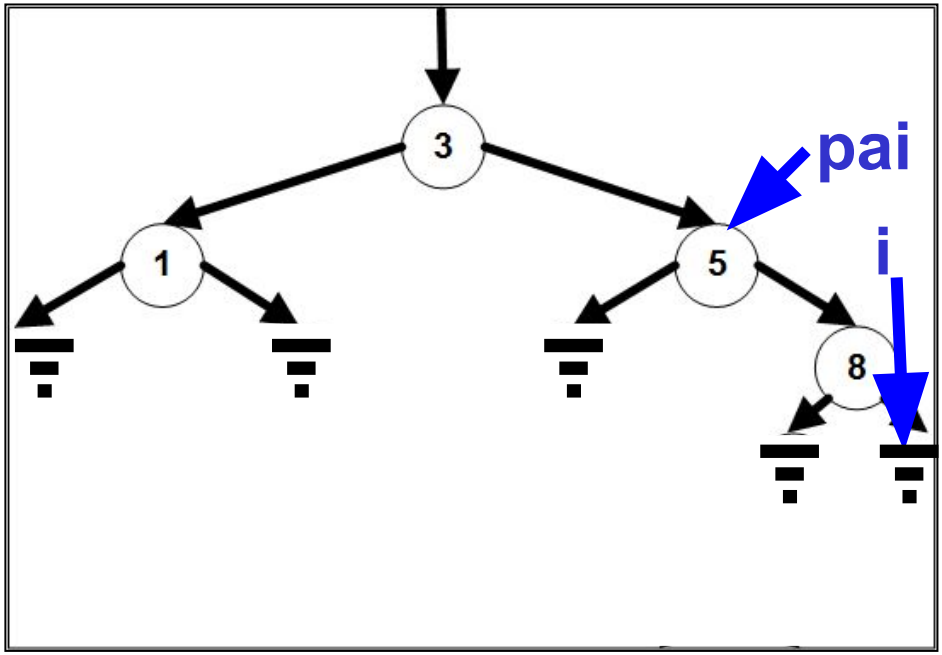
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```



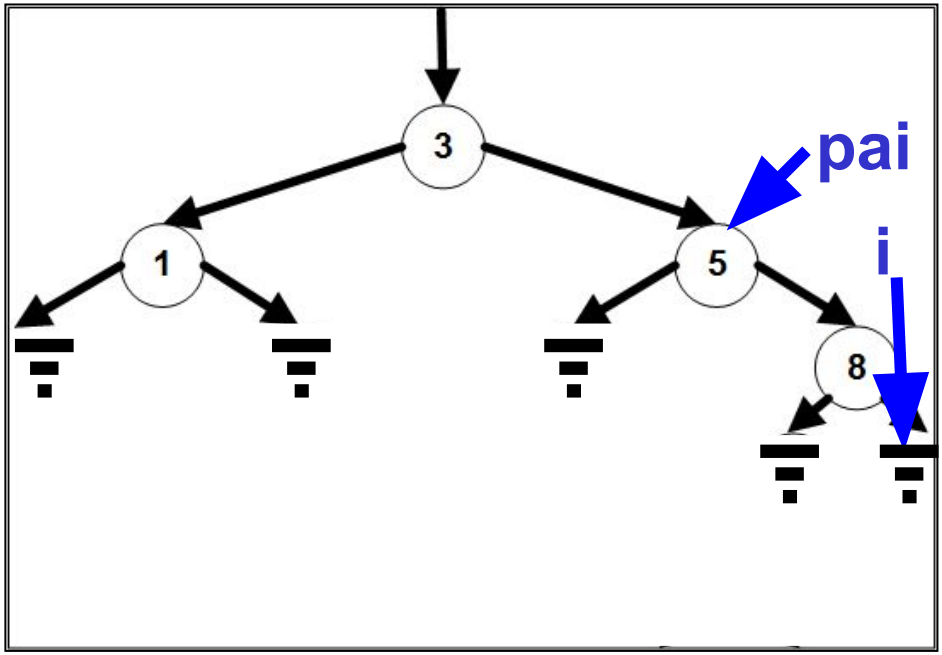
Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai)  {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento)  {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```



Inserção em Java com Passagem de Pai

```
//Inserir 3, 5, 1, 8

void inserirPai(int x, No i, No pai) {
  if (i == null) {
    If (x < pai.elemento){
      pai.esq = new No(x);
    } else {
      pai.dir = new No(x);
    }
  } else if (x < i.elemento) {
    inserirPai(x, i.esq, i);
  } else if (x > i.elemento) {
    inserirPai(x, i.dir, i);
  } else {
    throw new("Erro!");
  }
}
```



Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- **Inserção**
- Pesquisa
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passa
- Estruturas híbridas

  - Funcionamento básico
  - Exemplo
  - Inserção em Java com retorno de referência
  - Inserção em Java com passagem de pai
  - **Análise de Complexidade**

Análise de Complexidade da Inserção

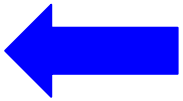
- Número de comparações no:
  - Melhor Caso: $\Theta(1)$
  - Pior Caso: $\Theta(n)$
  - Caso Médio: $\Theta(lg(n))$

Análise de Complexidade da Inserção

- 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)

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- **Pesquisa**
- Remoção
- Caminhamento
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas



- **Funcionamento básico**
- Algoritmo em Java
- Análise de Complexidade

Funcionamento Básico da Pesquisa

(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**

Agenda

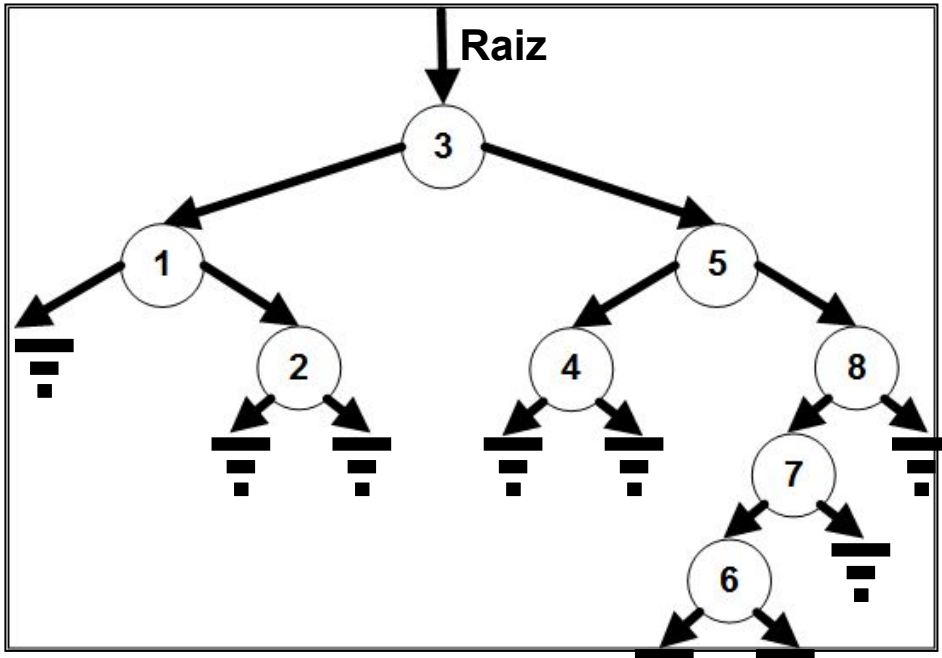
- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- **Pesquisa**
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

  - Funcionamento básico
  - **Algoritmo em Java**
  - Análise de Complexidade

Classe Árvore Binária em Java

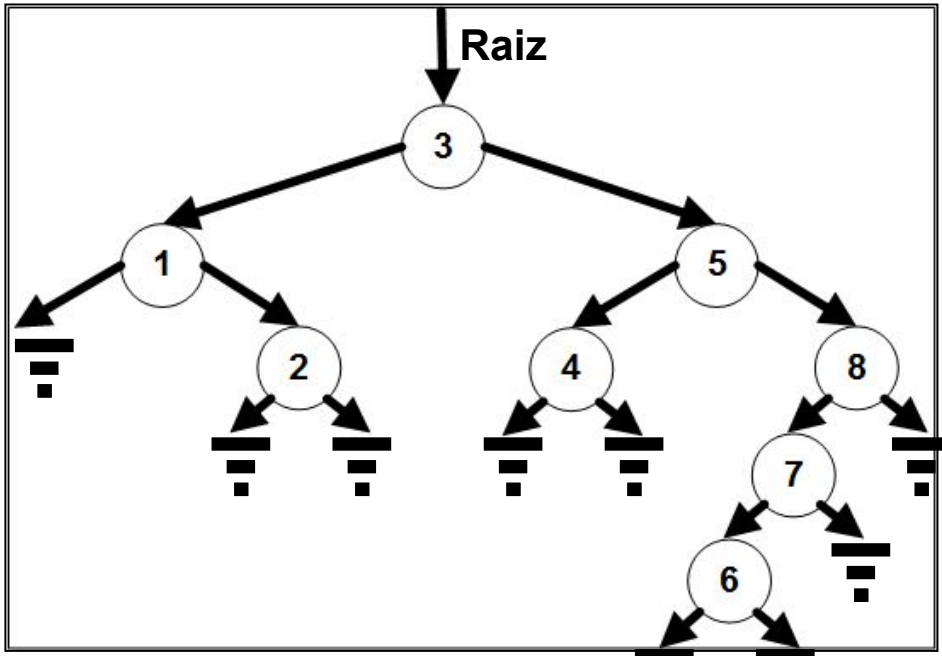
```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    boolean pesquisar(int x) {   }
    void remover(int x) {  }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
}
```

Vamos pesquisar se o
4 está em nossa árvore



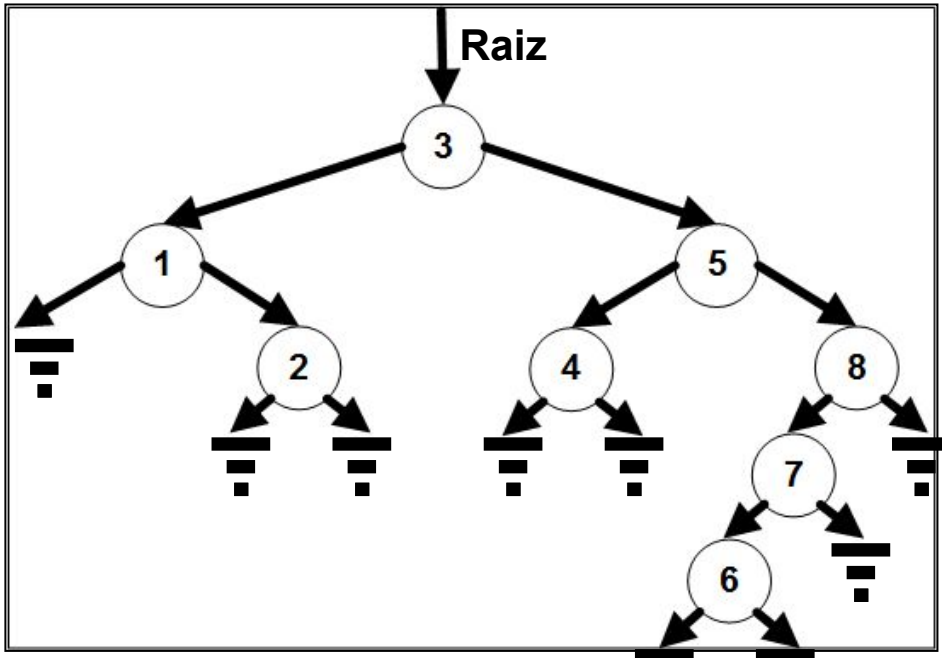
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



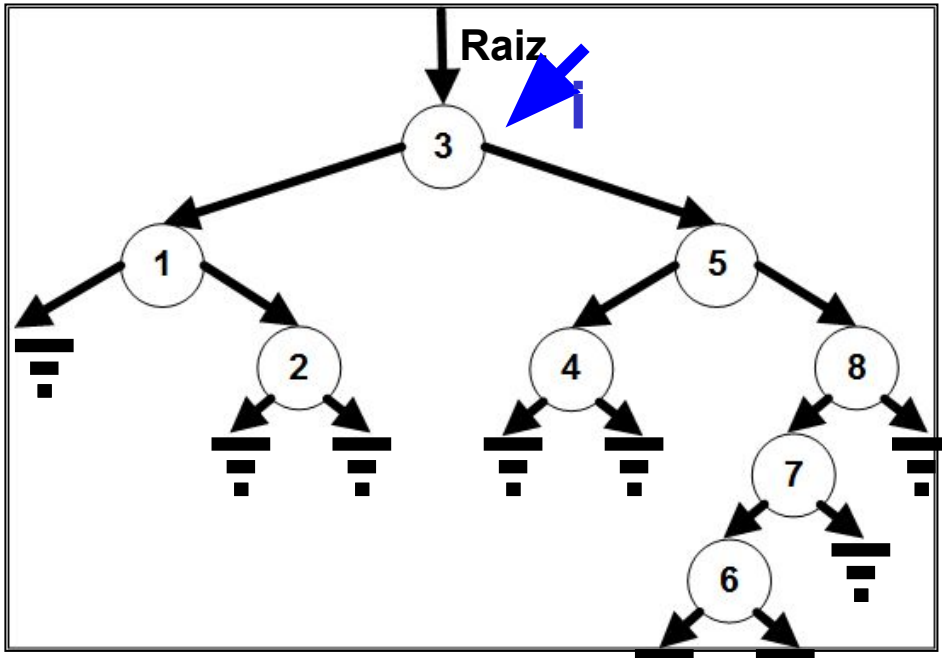
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



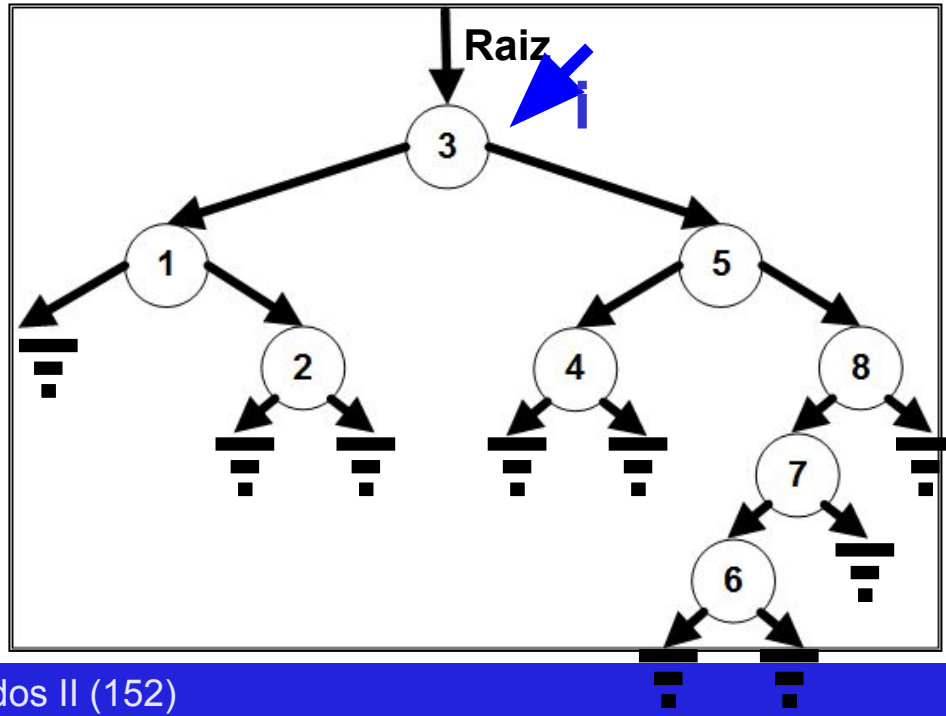
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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 de Pesquisa em Java

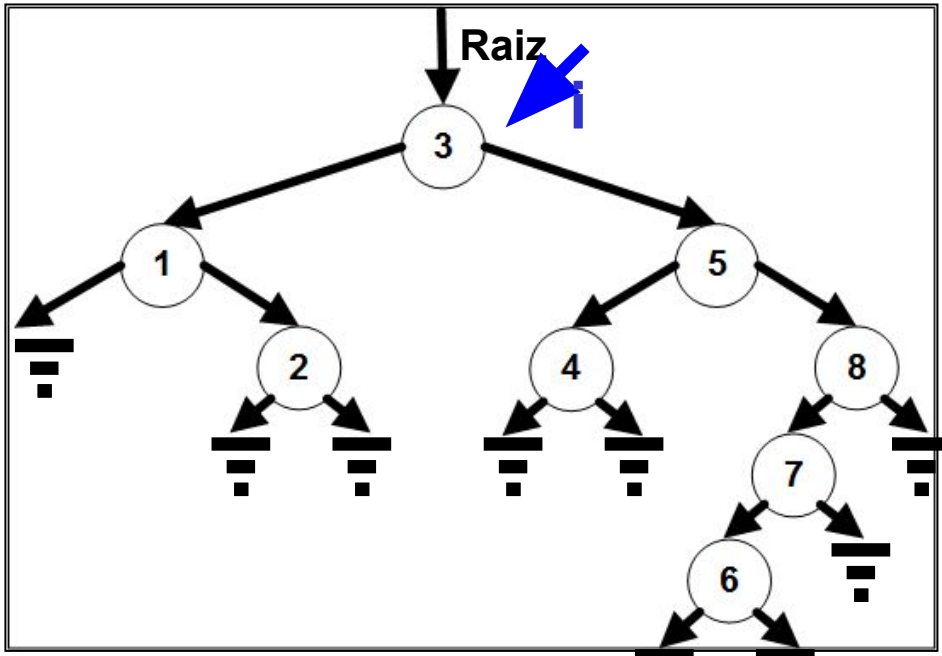
```
//Pesquisar  (4)

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

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 de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```

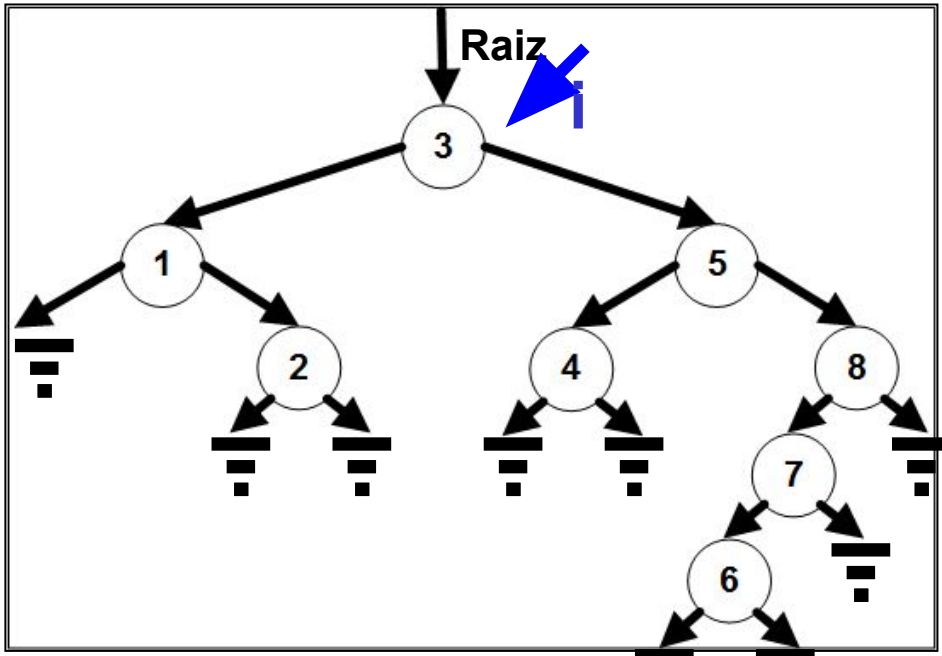
false: n(3) == null



Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```

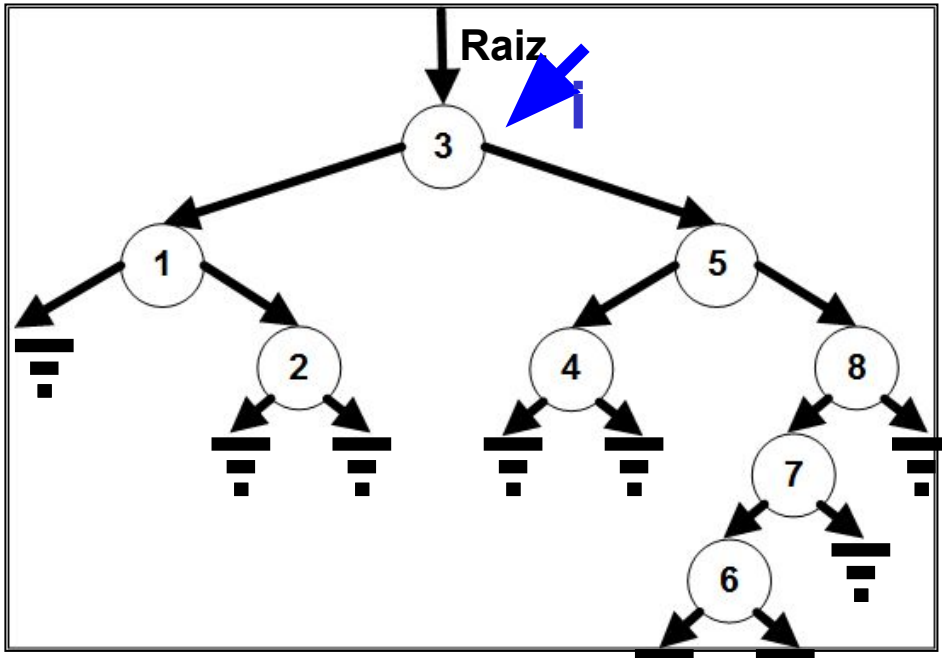
false: 4 == 3



Algoritmo de Pesquisa em Java

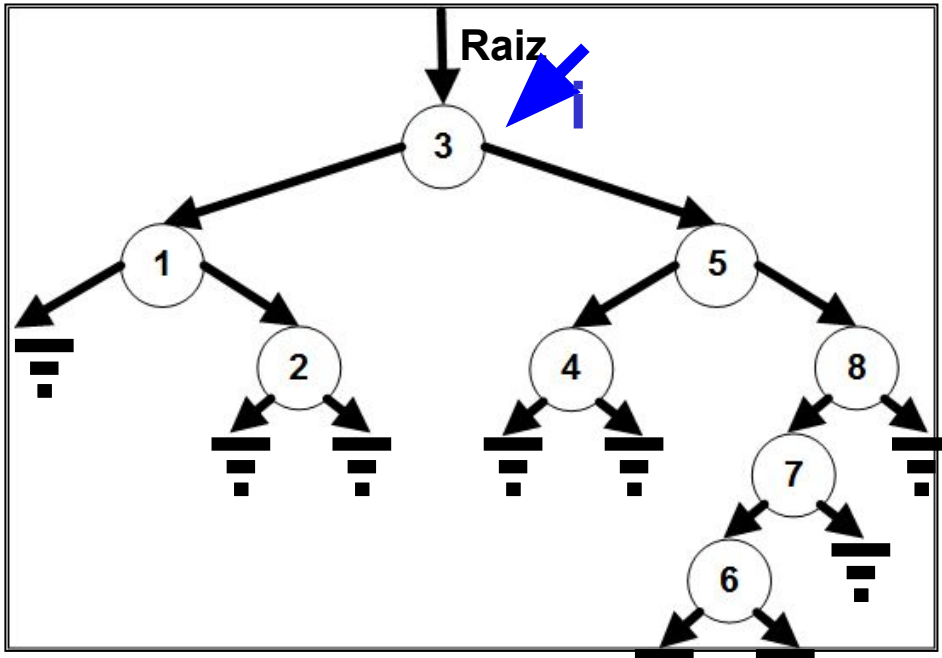
```
//Pesquisar  (4)

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

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



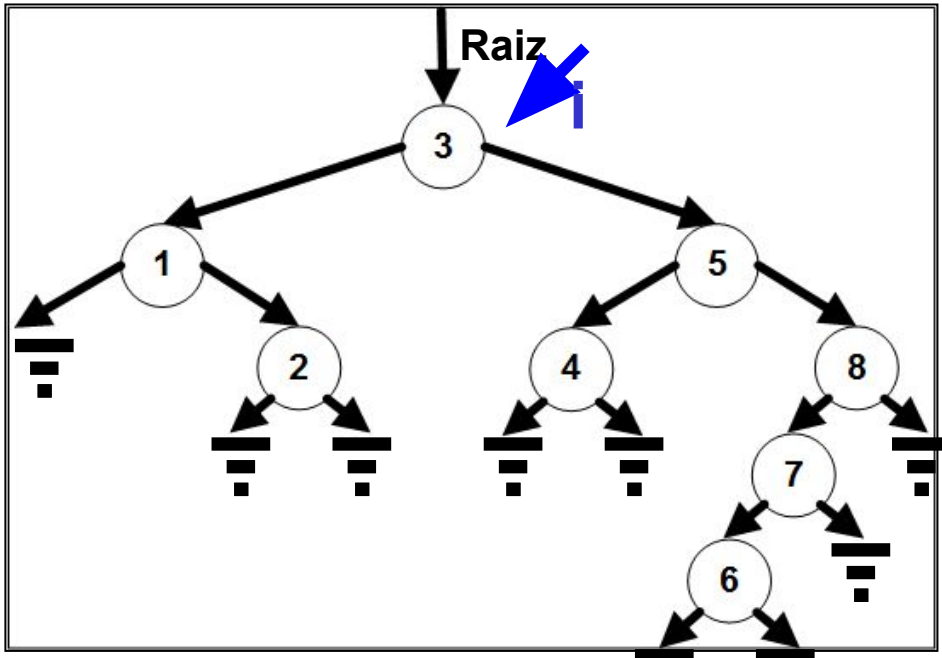
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



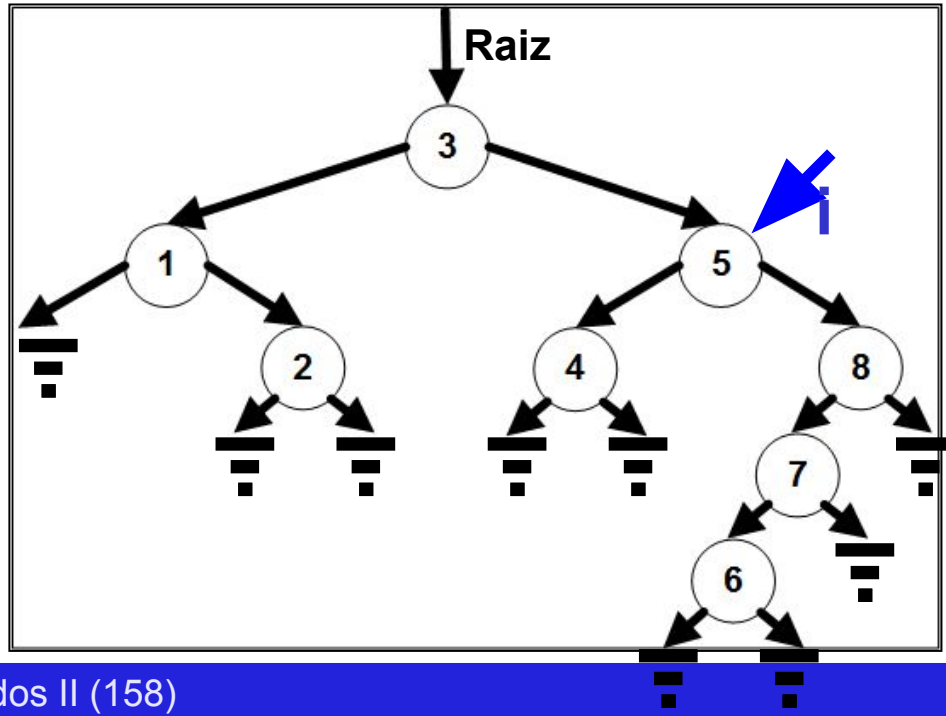
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



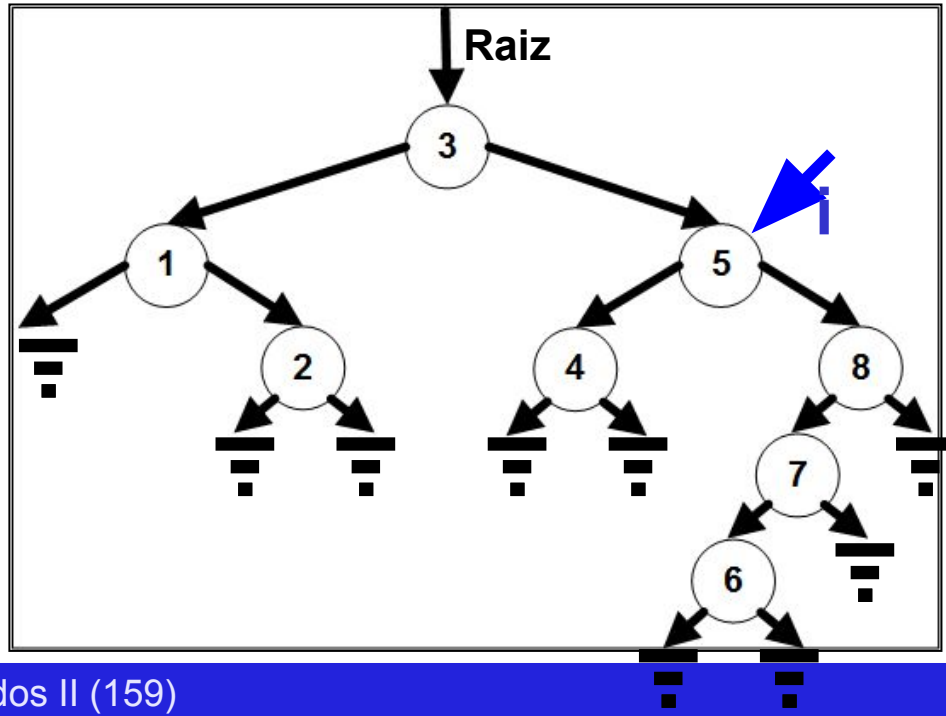
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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 de Pesquisa em Java

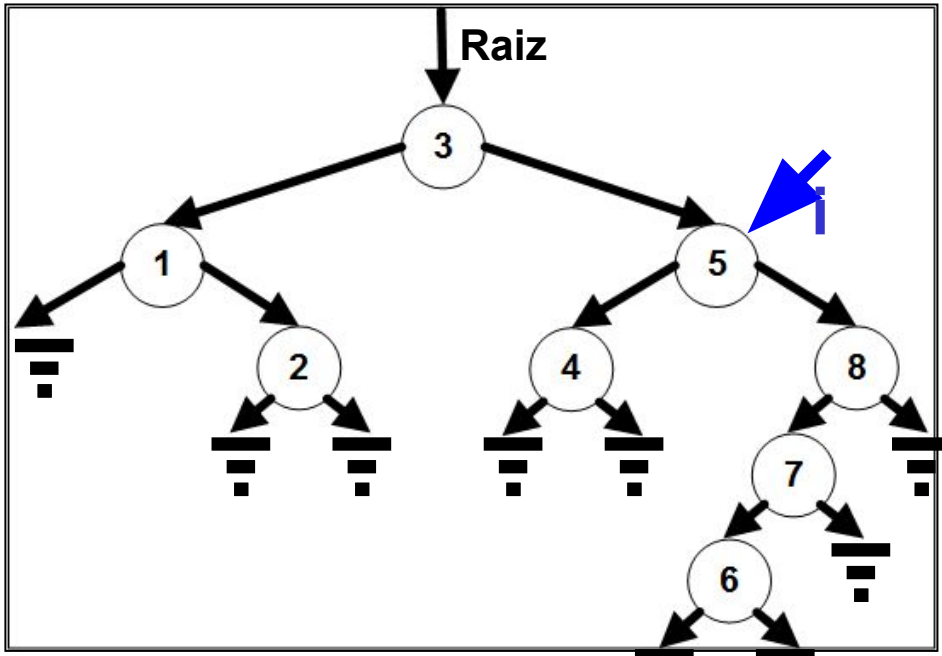
```
//Pesquisar  (4)

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

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 de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```

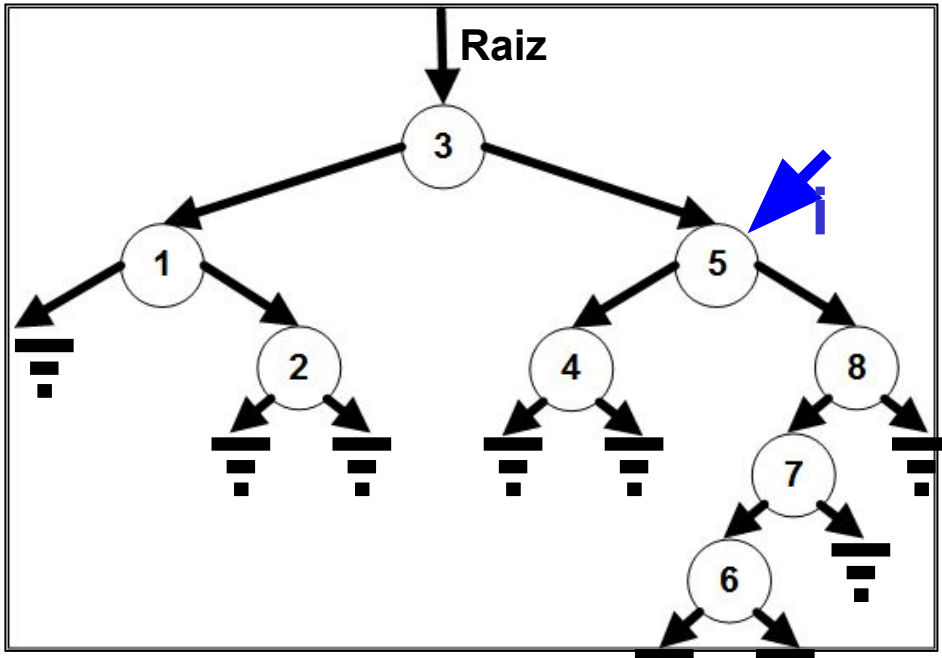
false: n(5) == null



Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```

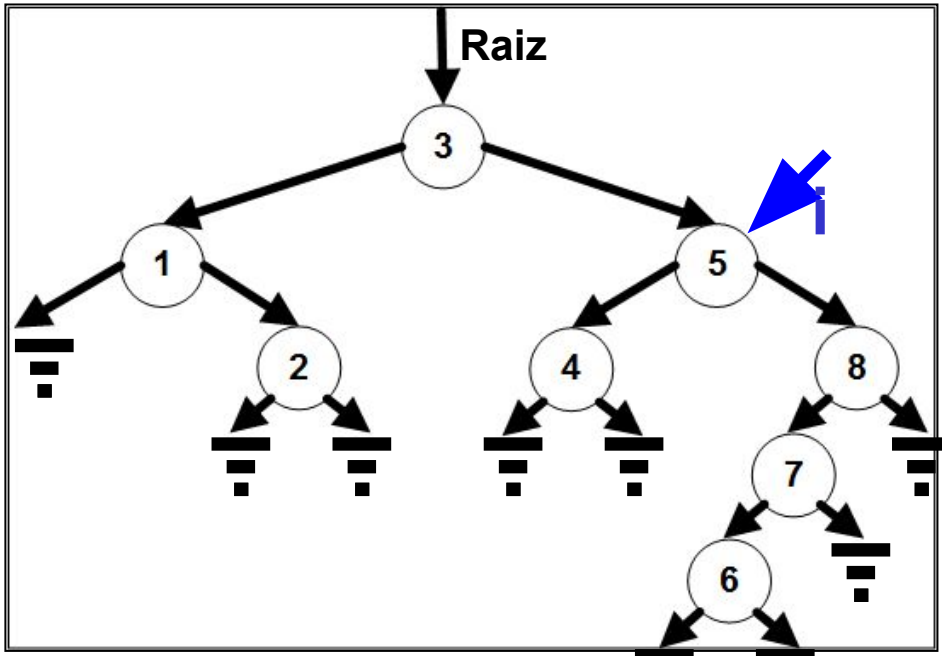
false: 4 == 5



Algoritmo de Pesquisa em Java

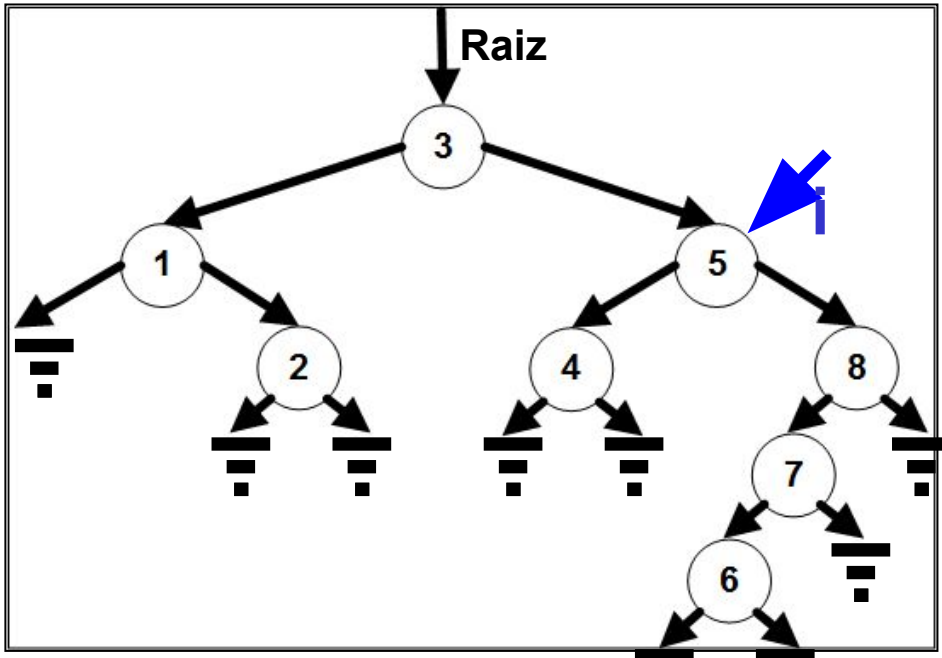
```
//Pesquisar  (4)

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

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



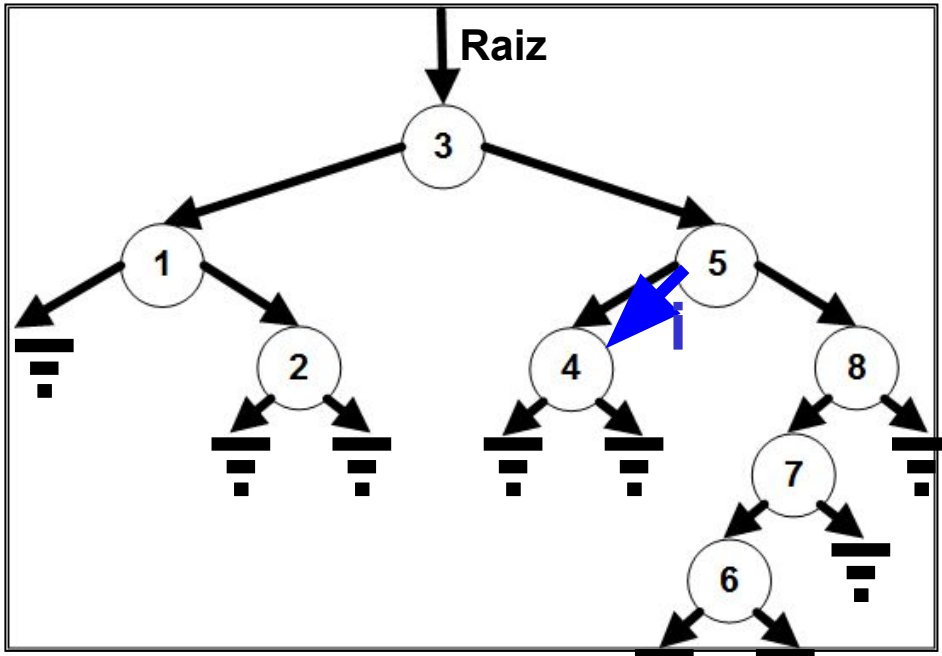
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



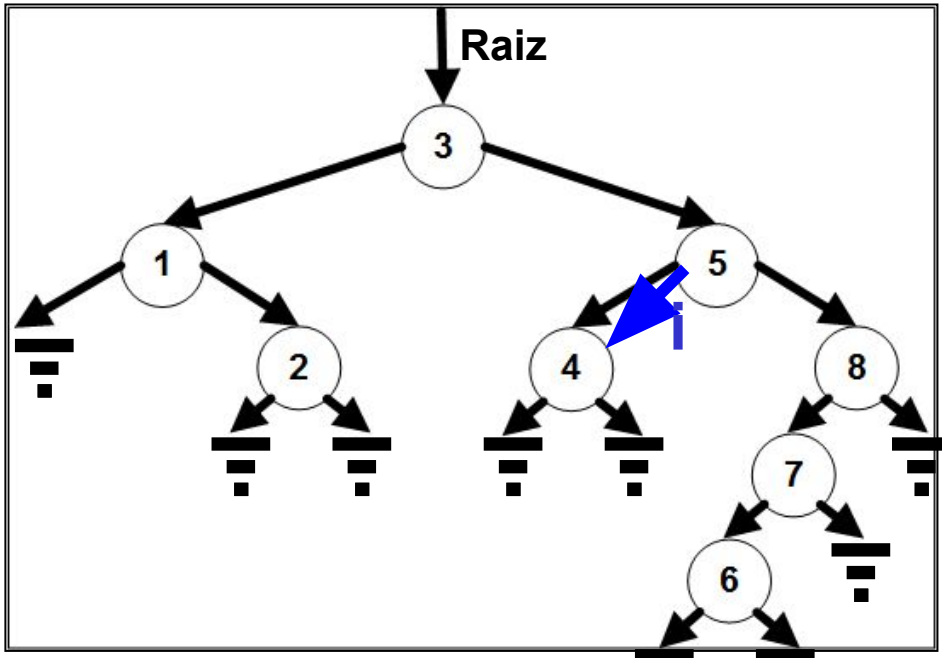
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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 de Pesquisa em Java

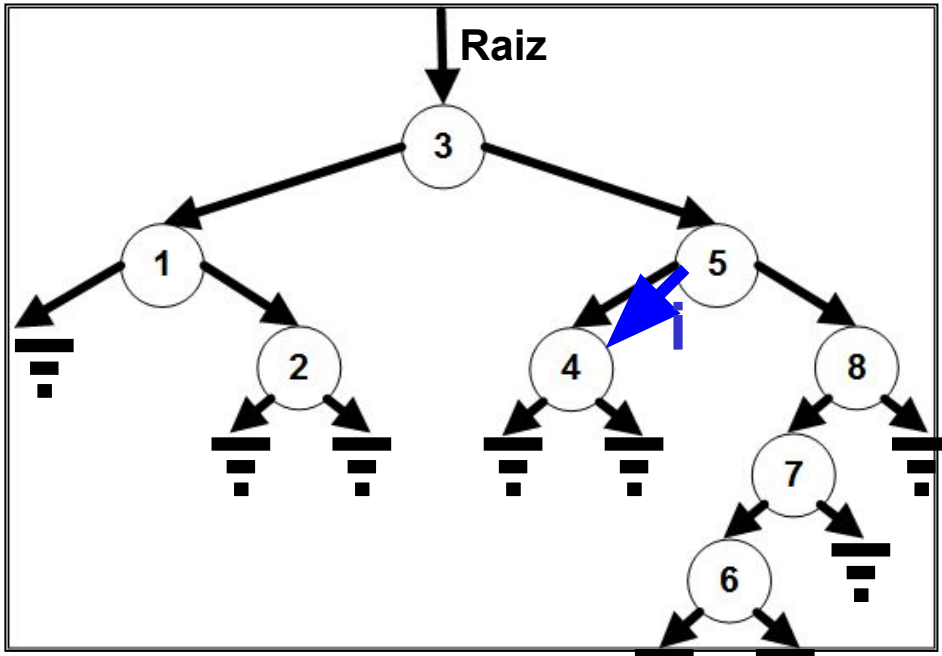
```
//Pesquisar  (4)

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

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 de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```

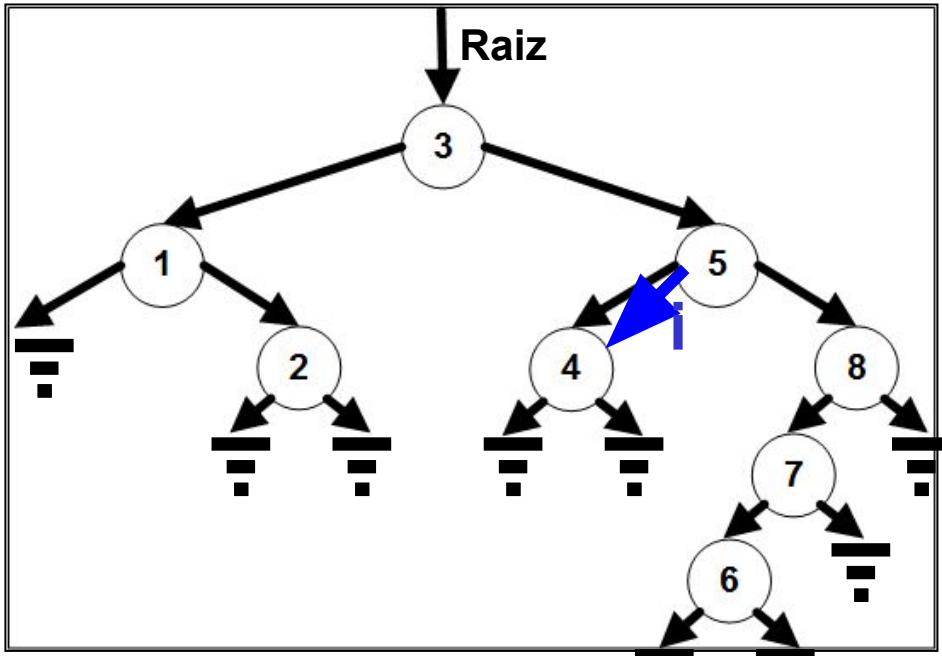
false: n(4) == null



Algoritmo de Pesquisa em Java

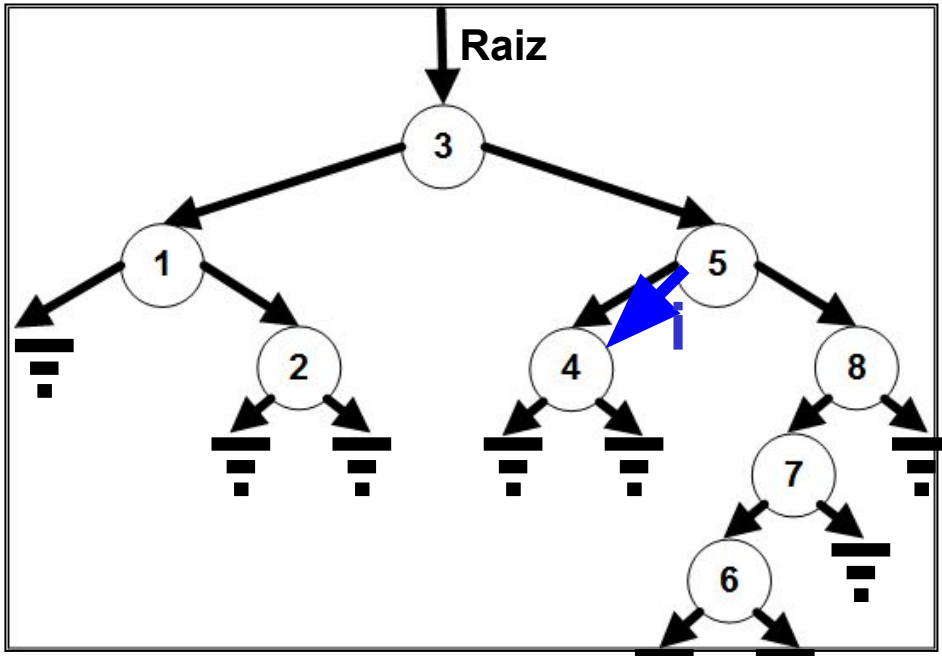
```
//Pesquisar  (4)

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

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



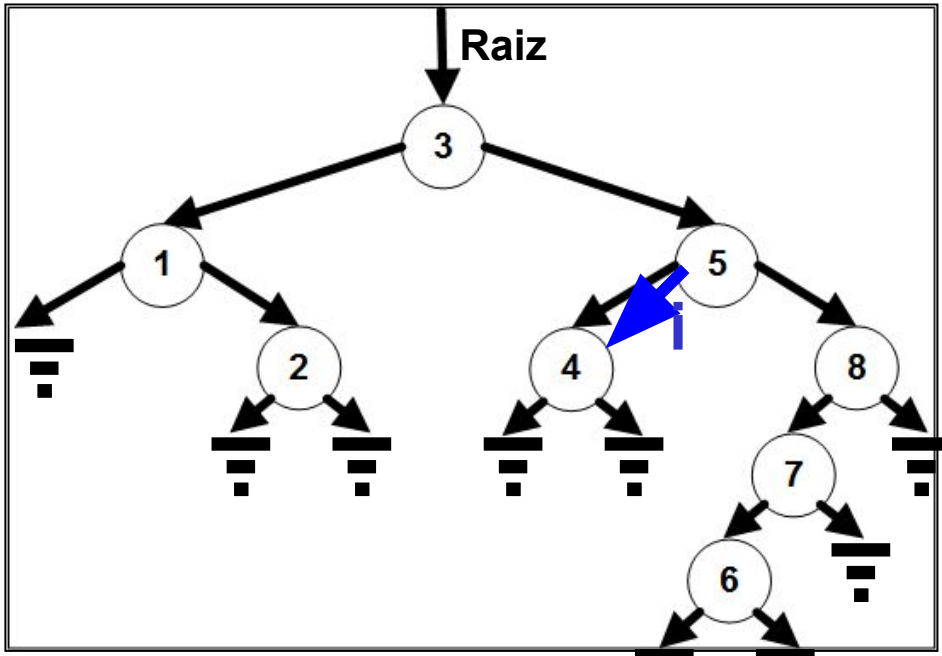
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



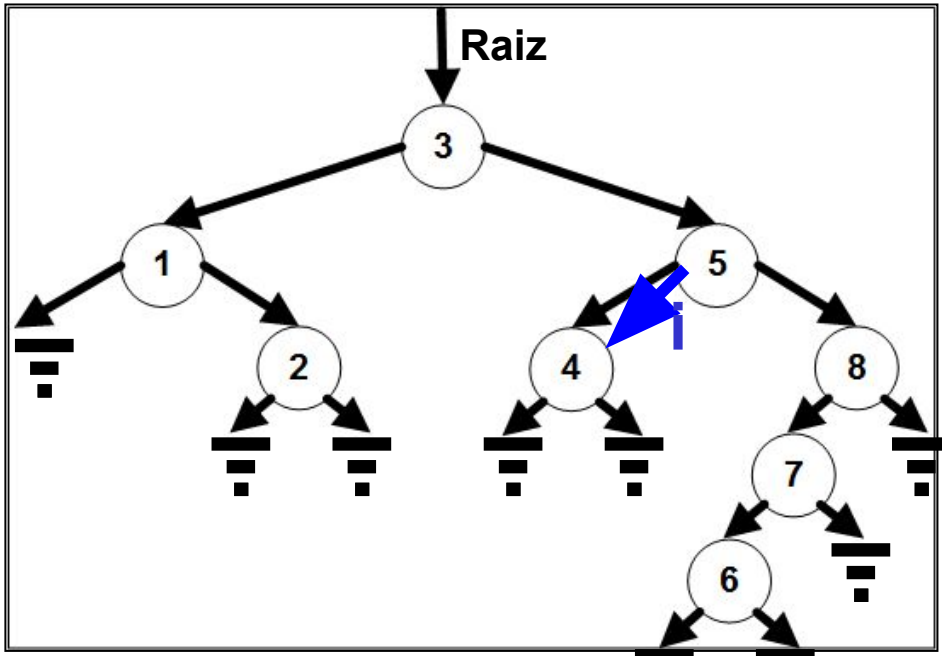
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



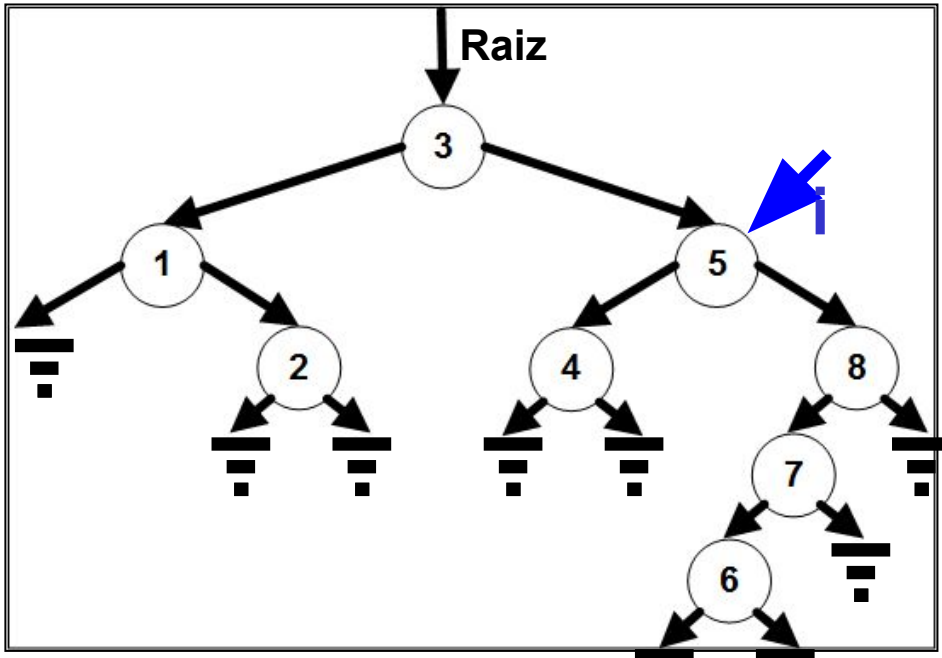
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



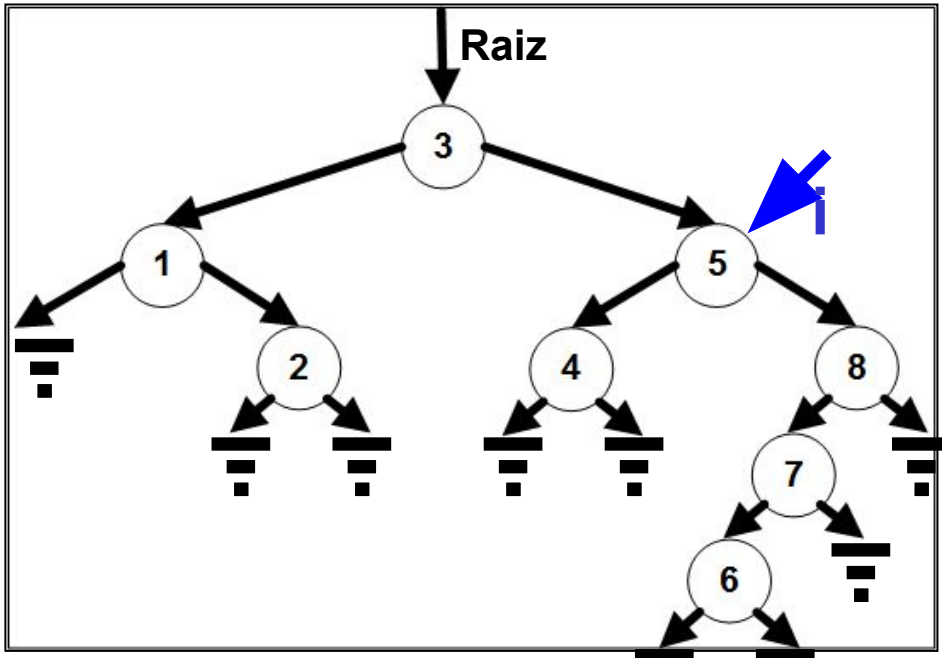
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



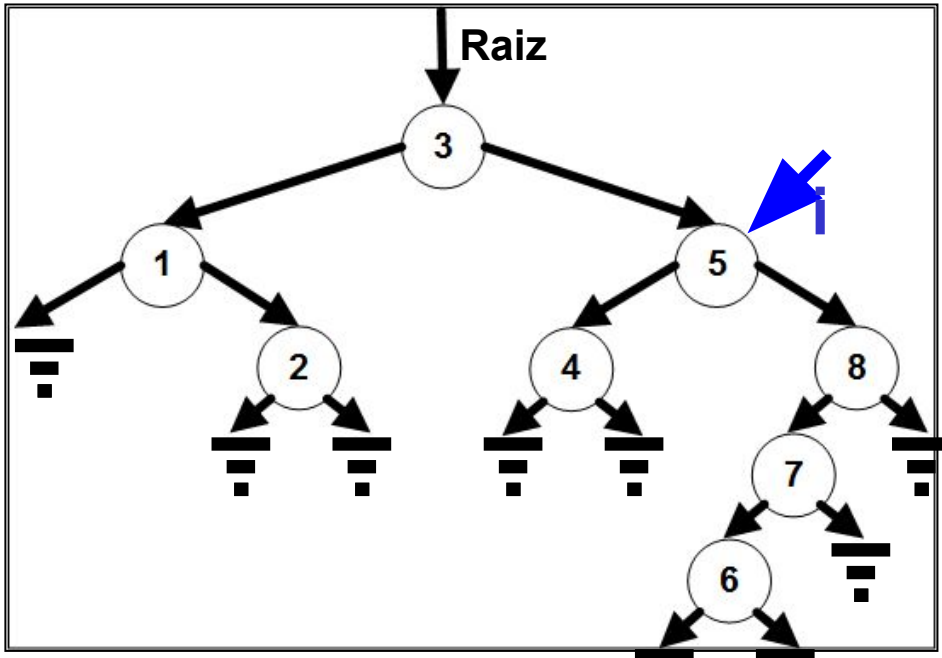
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



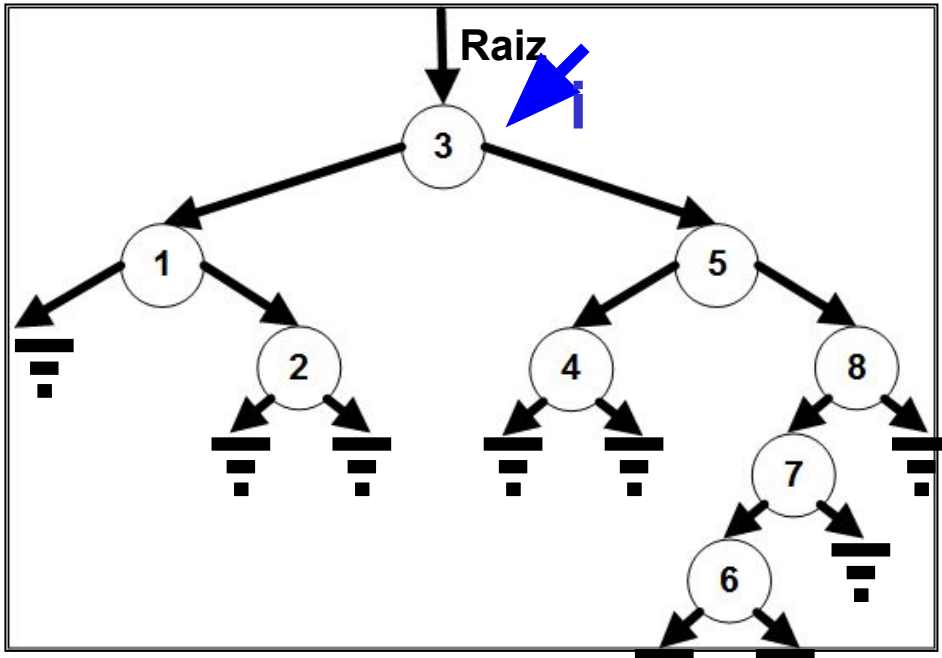
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



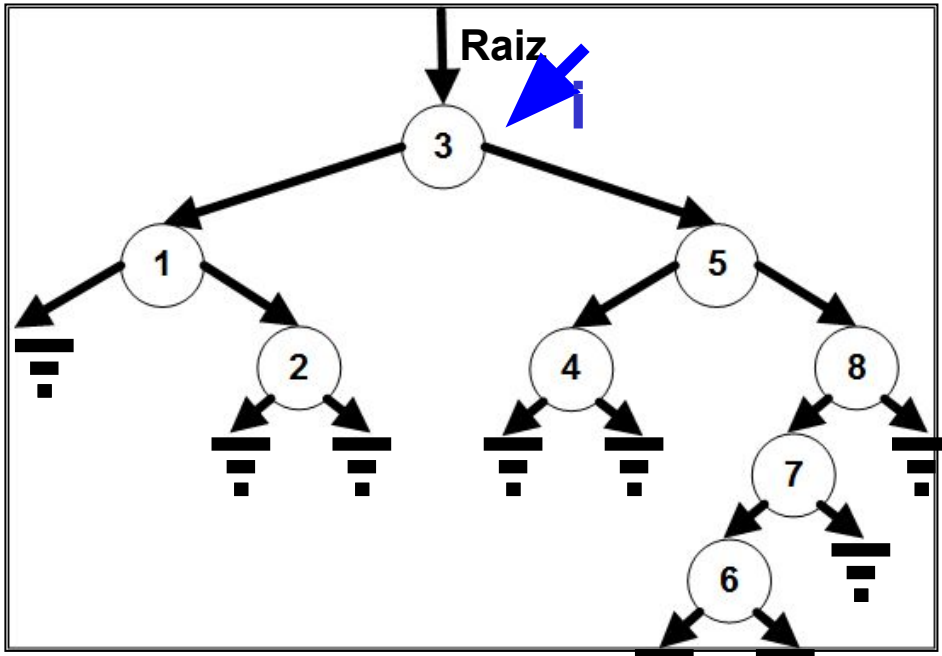
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



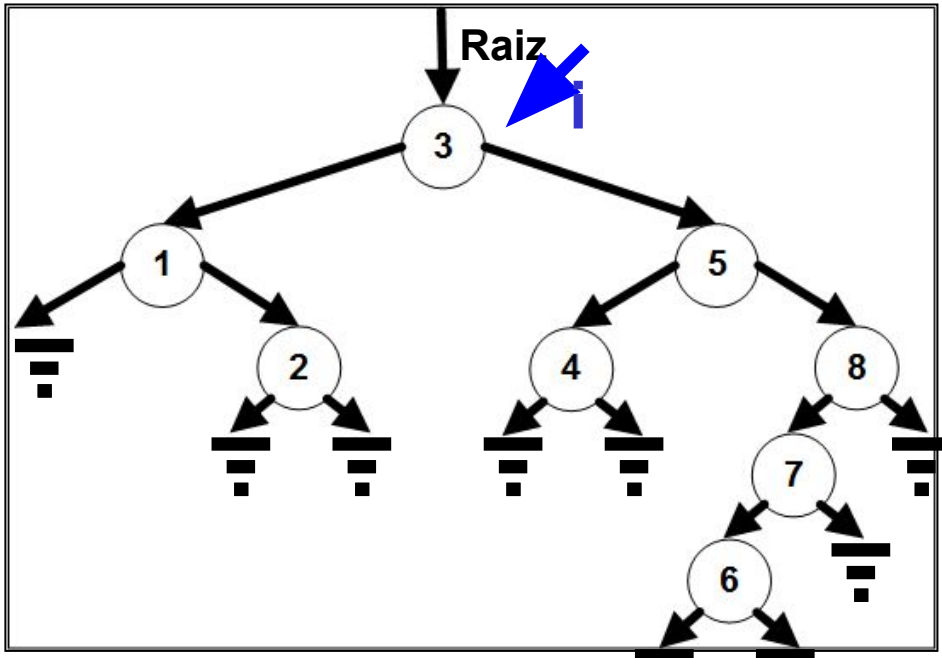
Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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 de Pesquisa em Java

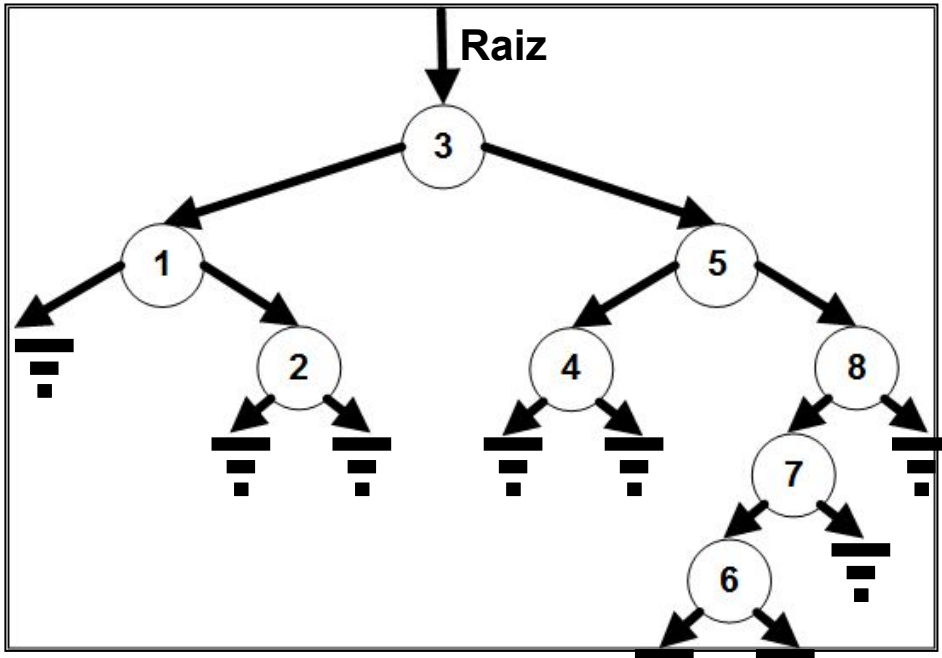
```
//Pesquisar  (4)

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

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 de Pesquisa em Java

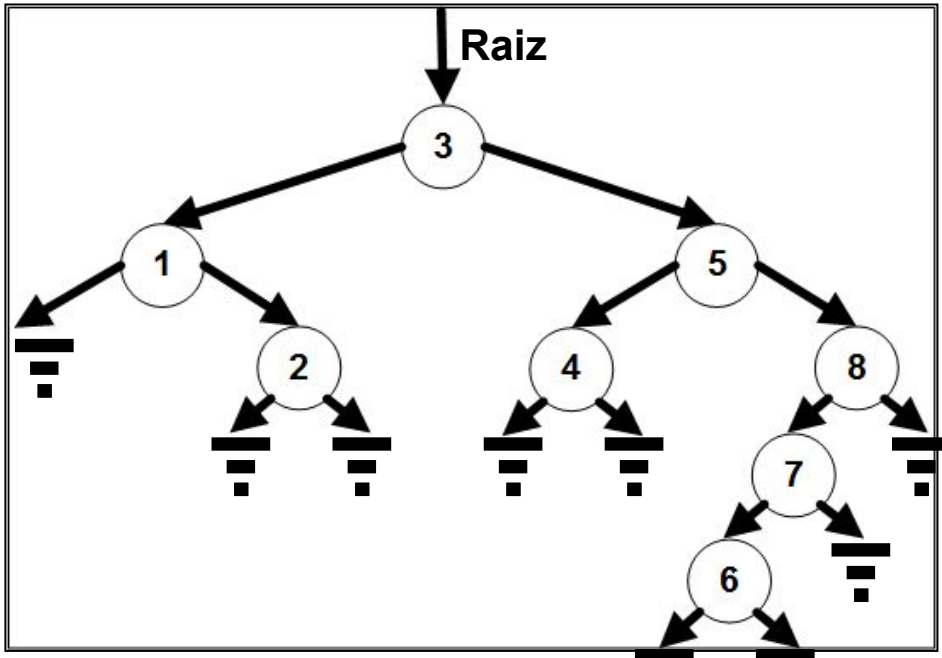
```
//Pesquisar  (4)

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

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;
}
```

retorna true



Algoritmo de Pesquisa em Java

```
//Pesquisar  (4)

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

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;
}
```



Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- **Pesquisa**
- Caminhamento
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

  - Funcionamento básico
  - Algoritmo em Java
  - **Análise de Complexidade**

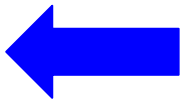
Análise de Complexidade da Pesquisa

- 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)

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- **Caminhamento** 
- Remoção
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

Caminhamento

- Consiste em “caminhar” por todos os nós da árvore
- Também chamado de percorrer, buscar, visitar, mostrar, ...
- Análise de complexidade: Θ(n) visitas

Alguns Caminhamento

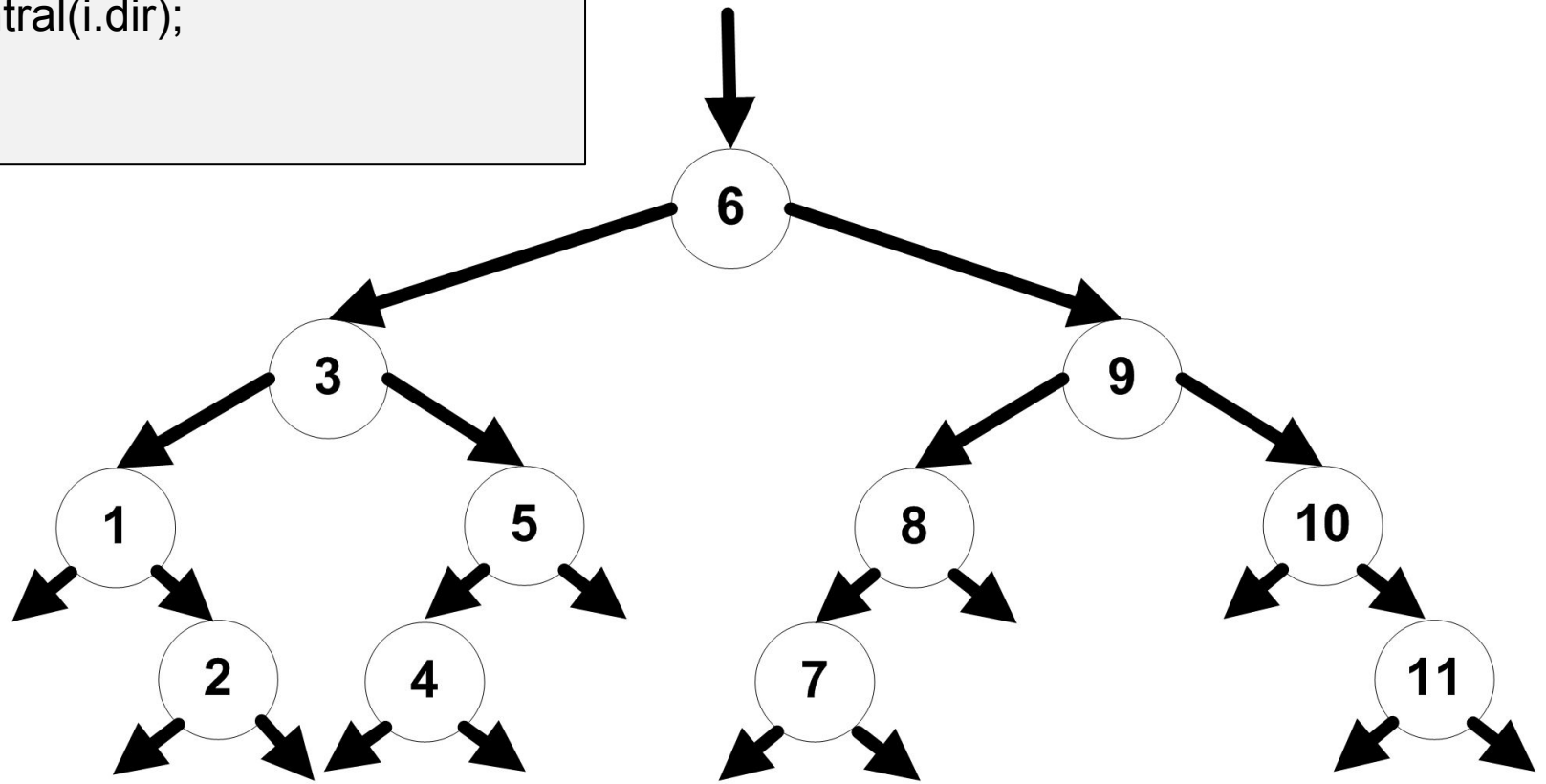
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
                    //central ou em ordem
```

```
void caminharPre(No i) {
  if (i != null) {
    System.out.print(i.elemento + " ");
    caminharPre(i.esq);
    caminharPre(i.dir);
  }
}
                    //pré-fixado ou pré-ordem
```

```
void caminharPos(No i) {
  if (i != null) {
    caminharPos(i.esq);
    caminharPos(i.dir);
    System.out.print(i.elemento + " ");
  }
}
                    //pós-fixado ou pós-ordem
```

Caminhamento Central ou Em Ordem

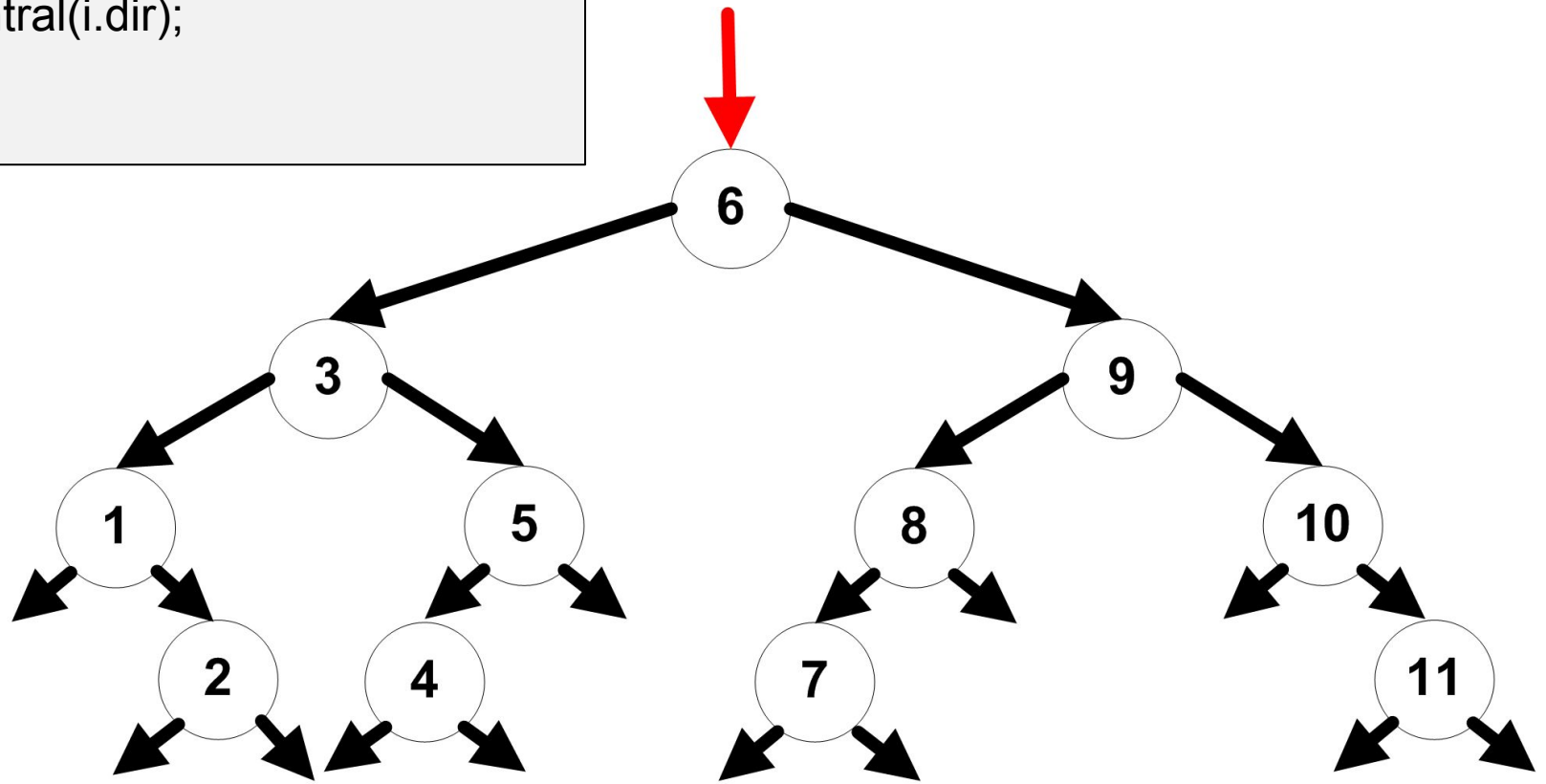
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	

Caminhamento Central ou Em Ordem

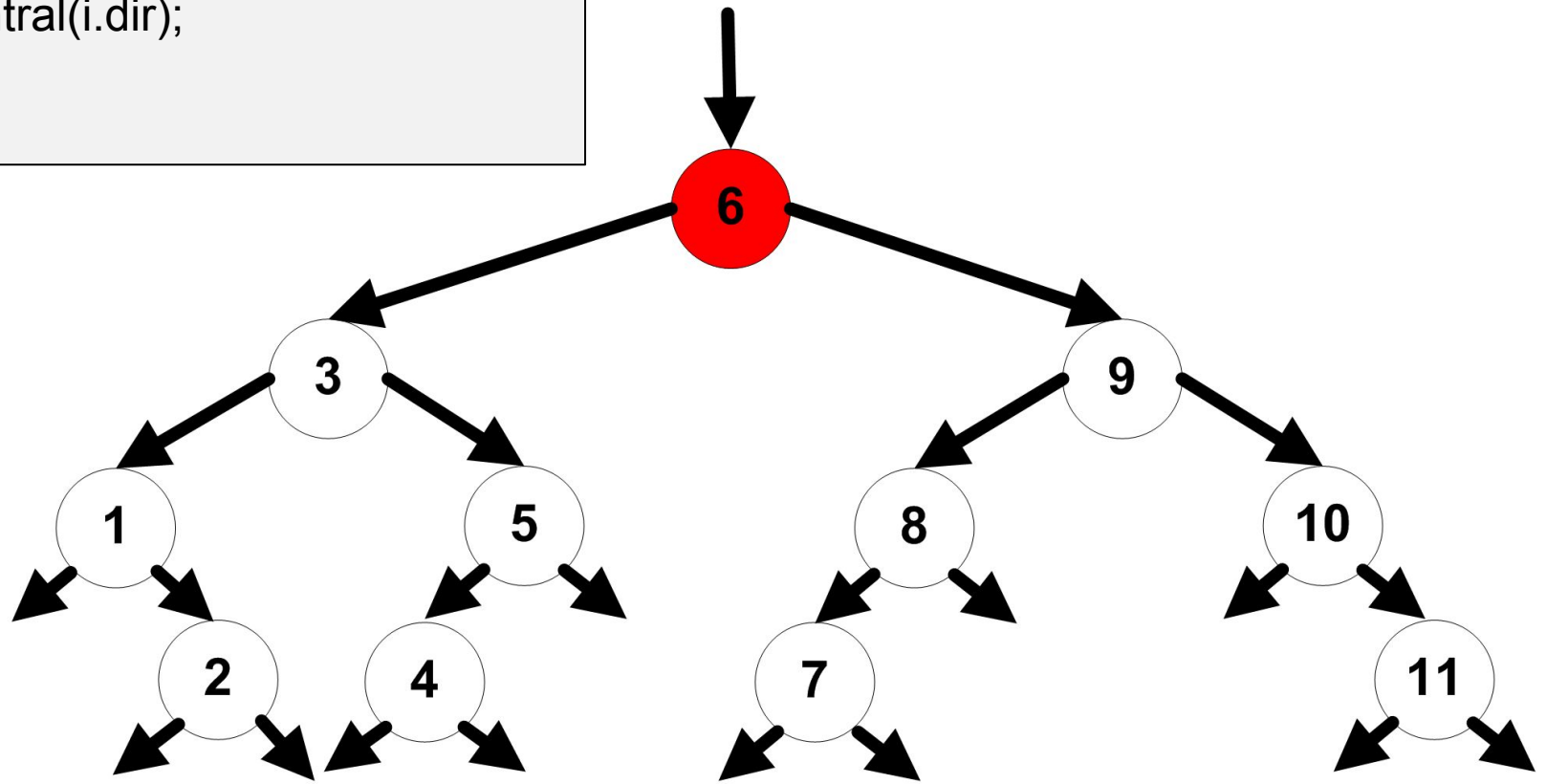
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	

Caminhamento Central ou Em Ordem

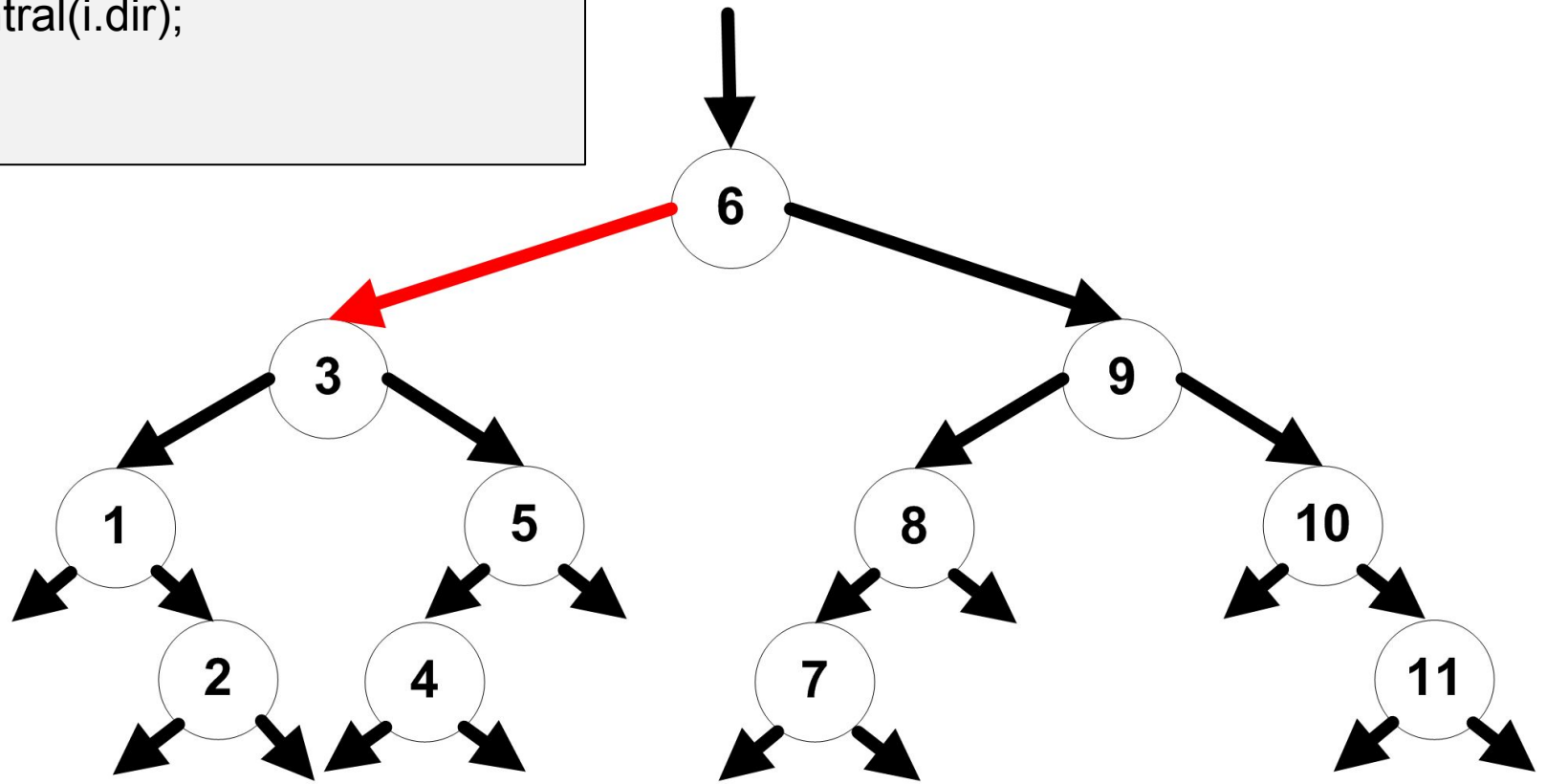
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	

Caminhamento Central ou Em Ordem

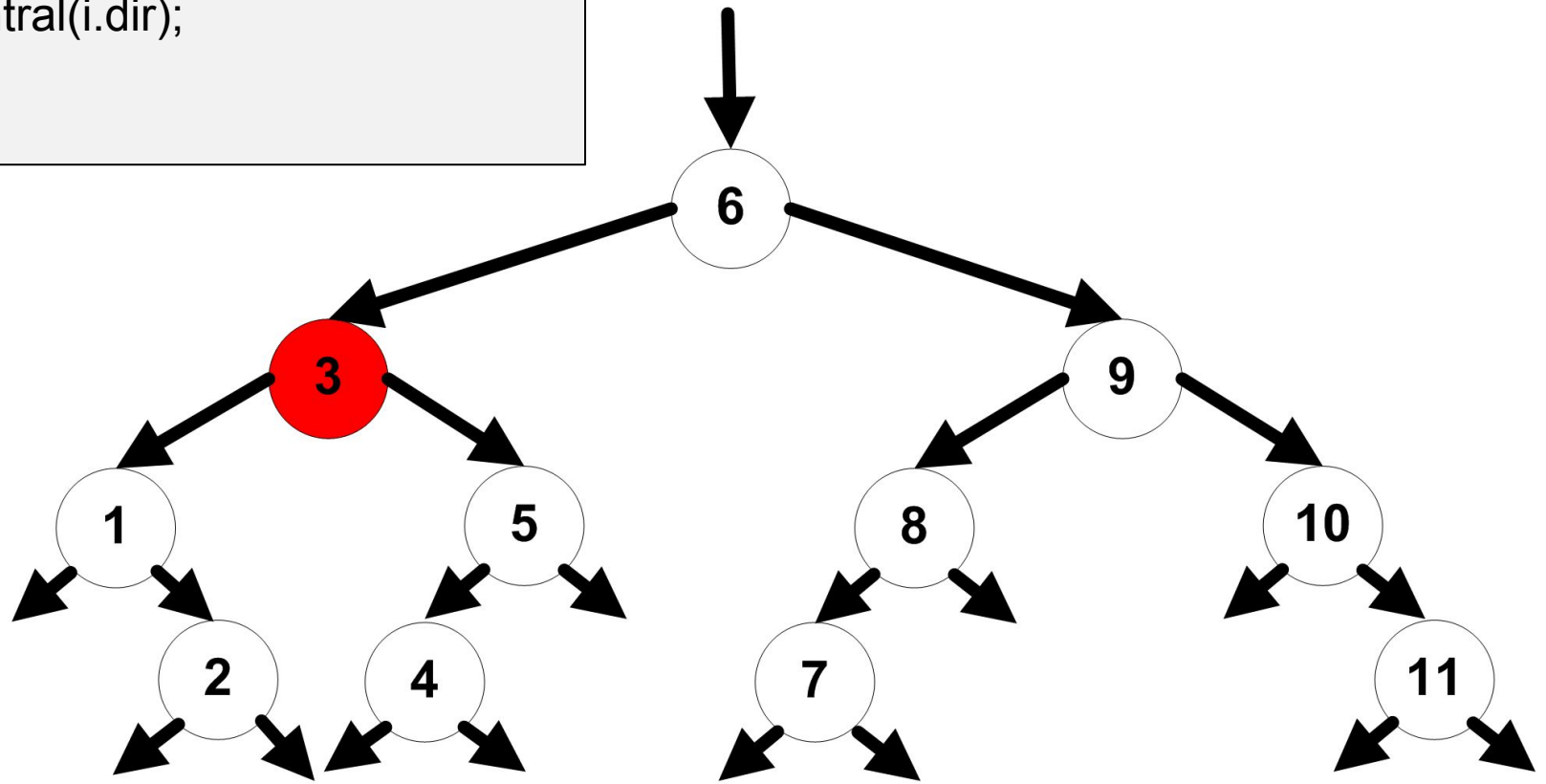
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	

Caminhamento Central ou Em Ordem

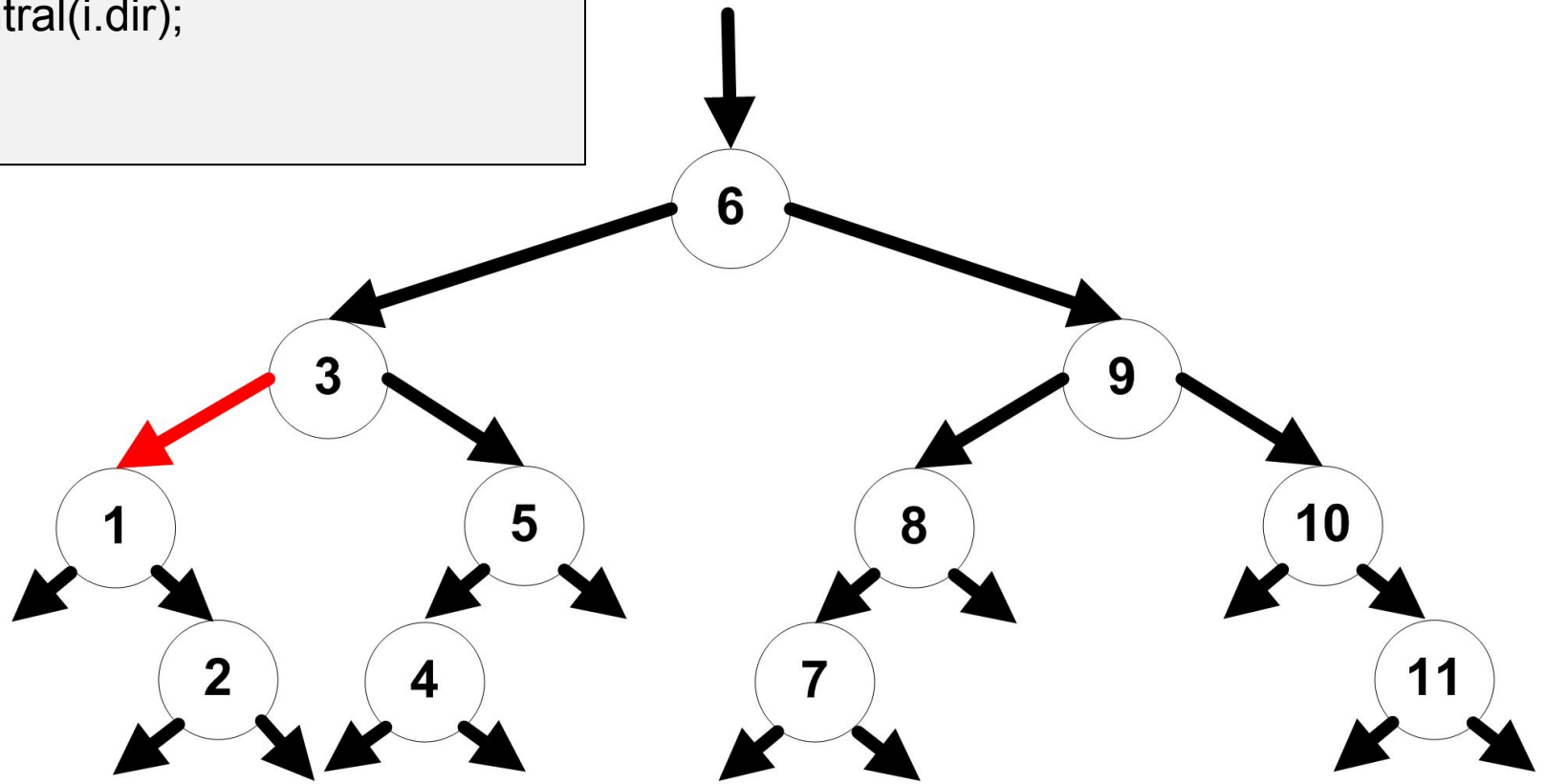
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	

Caminhamento Central ou Em Ordem

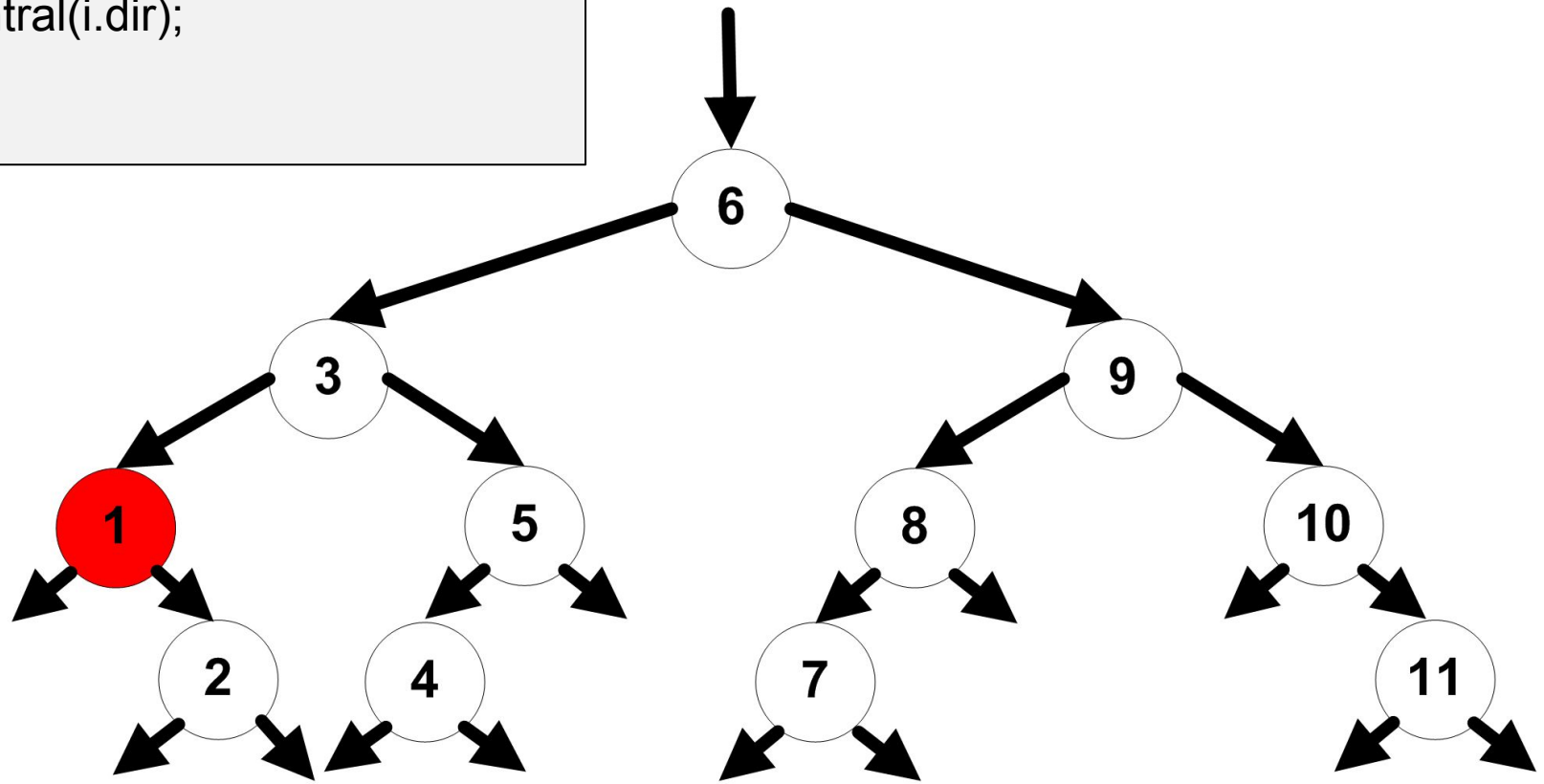
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	

Caminhamento Central ou Em Ordem

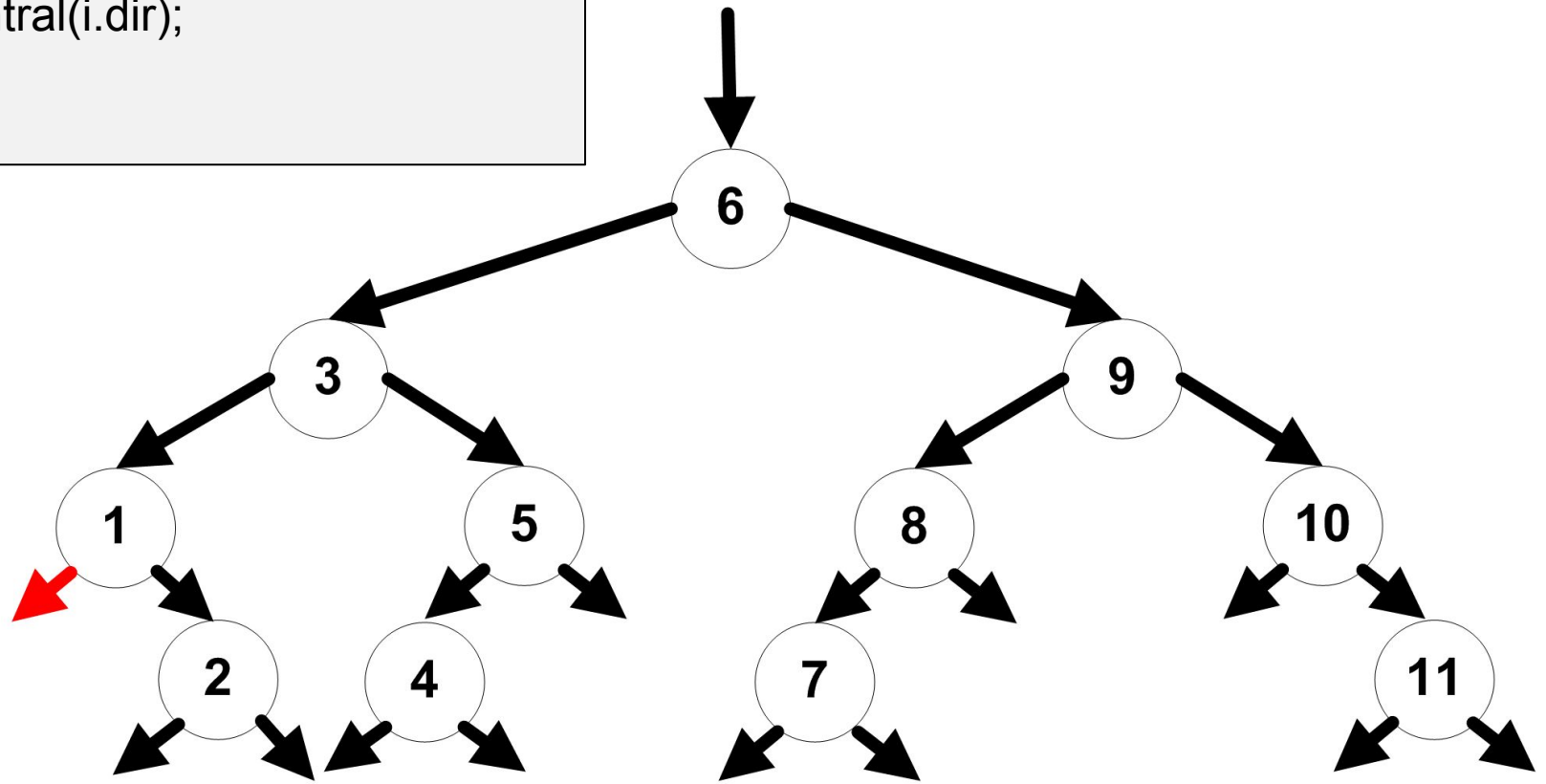
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	

Caminhamento Central ou Em Ordem

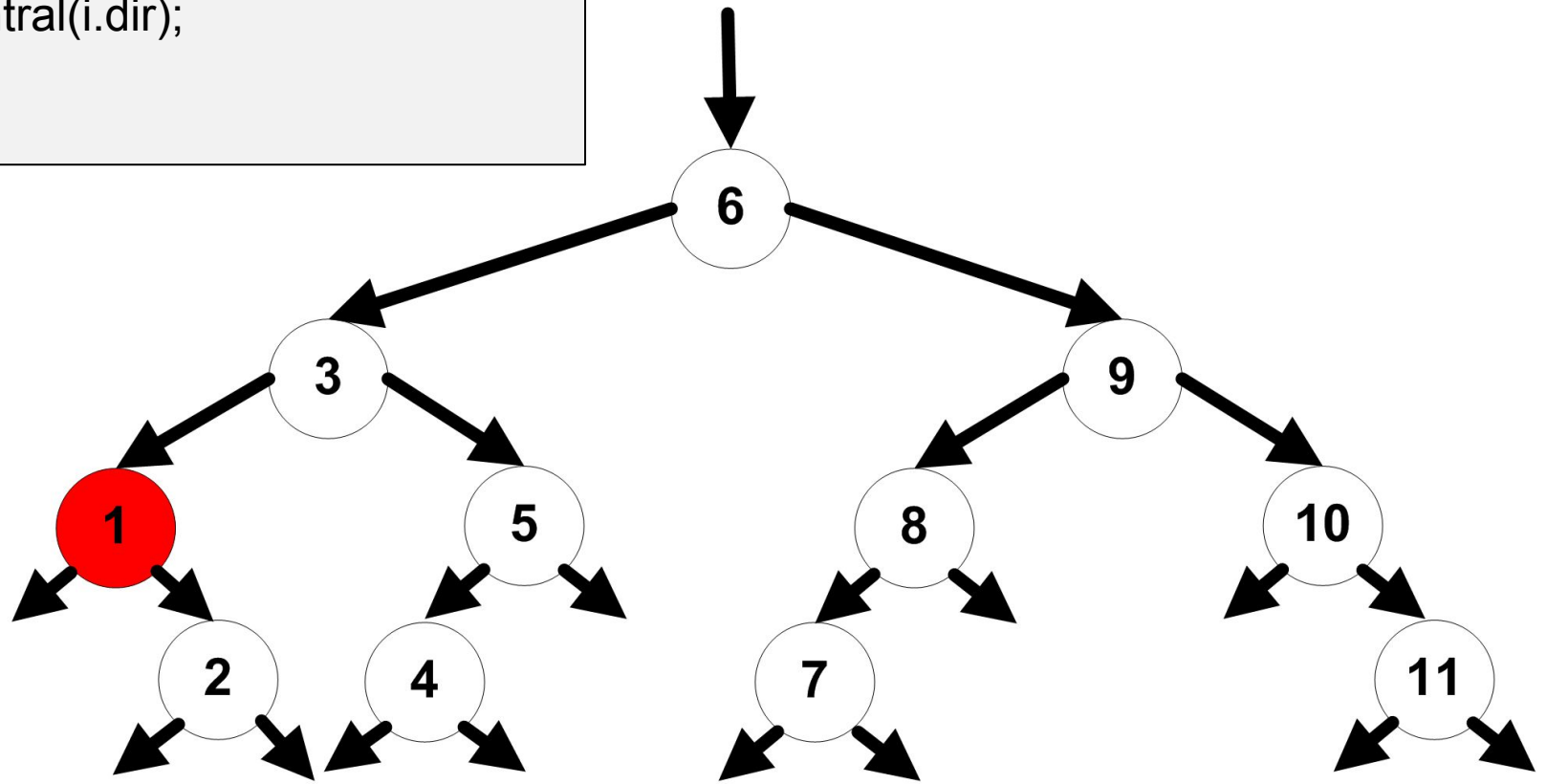
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	

Caminhamento Central ou Em Ordem

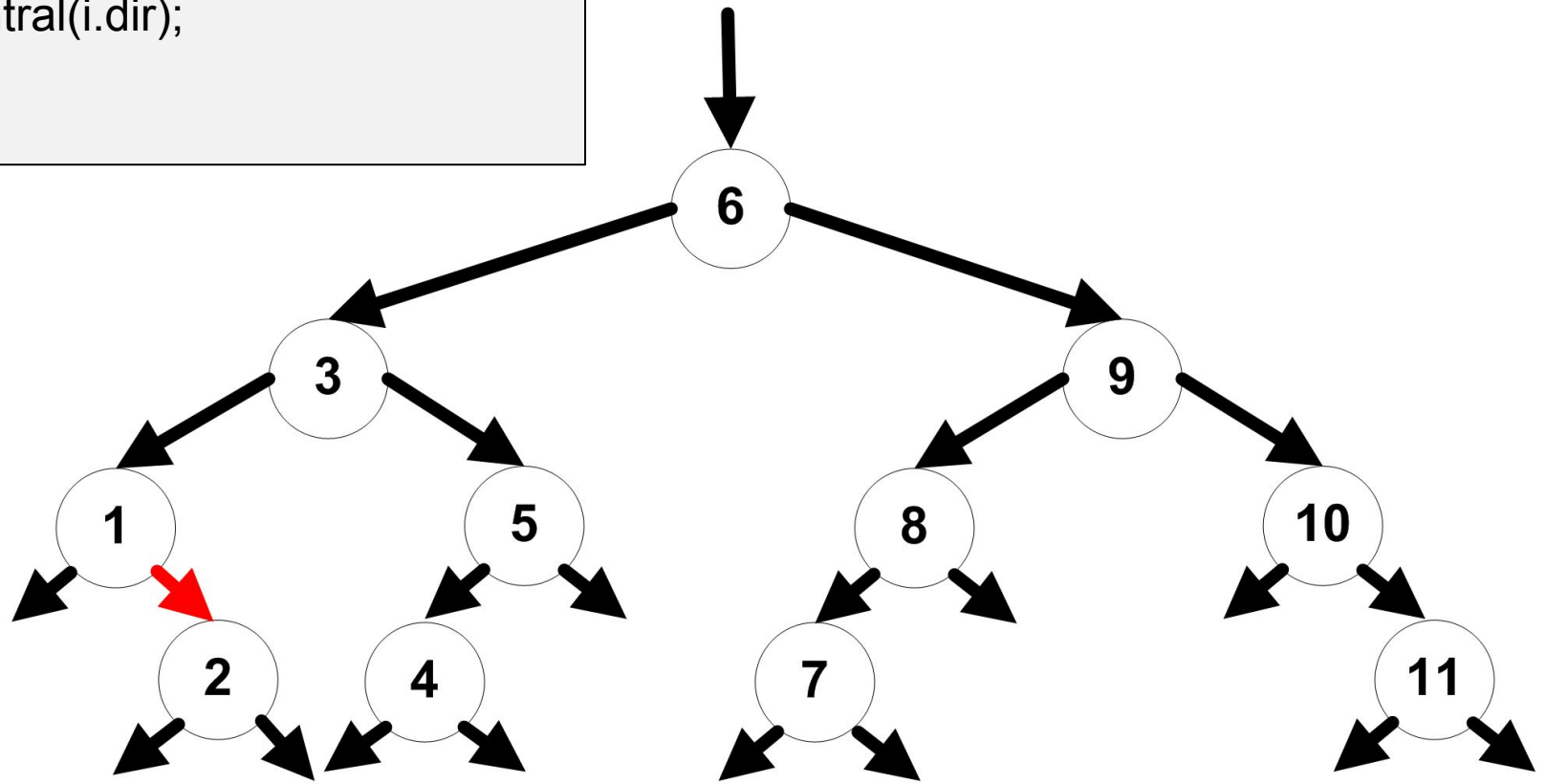
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1

Caminhamento Central ou Em Ordem

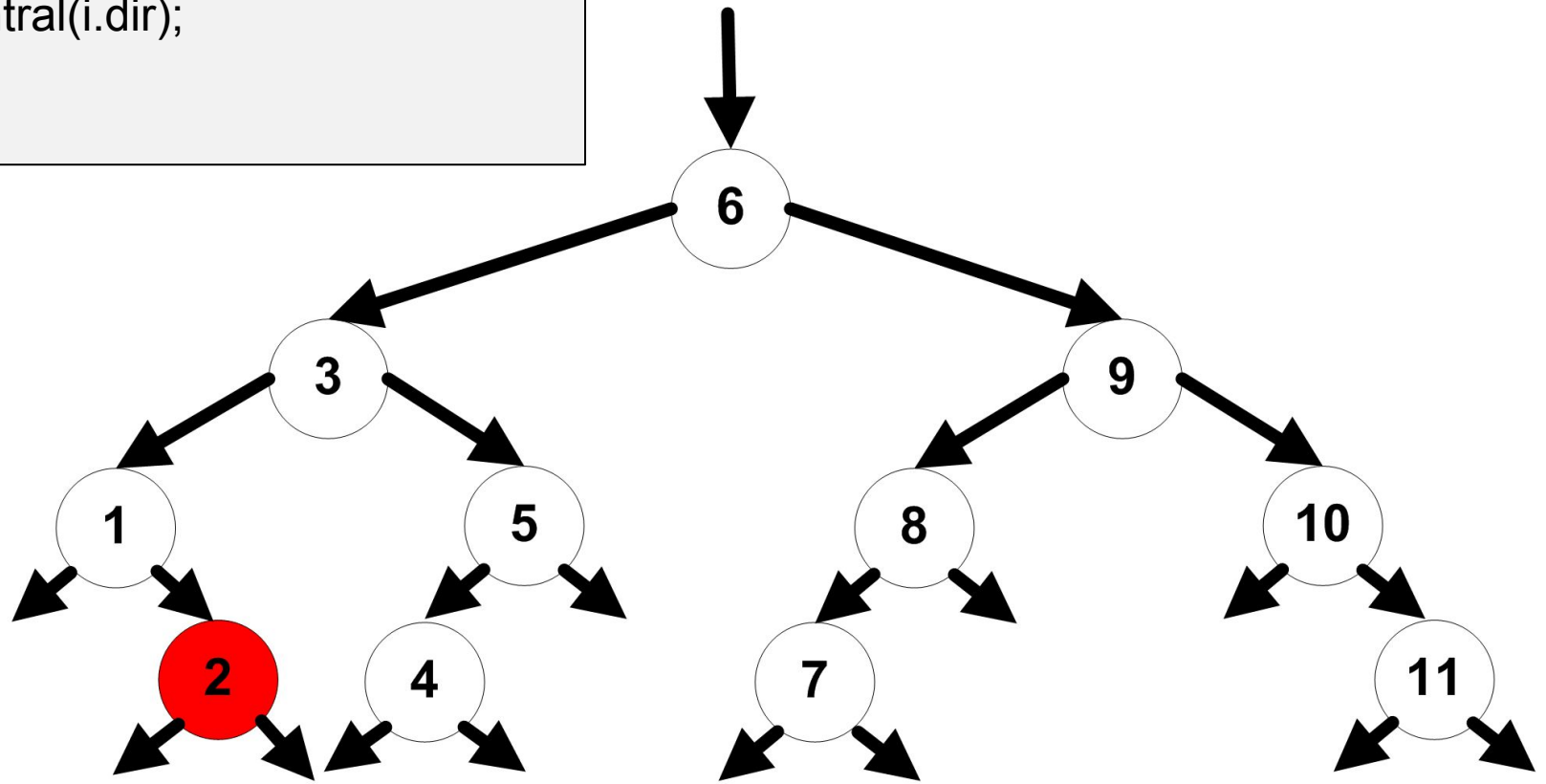
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1

Caminhamento Central ou Em Ordem

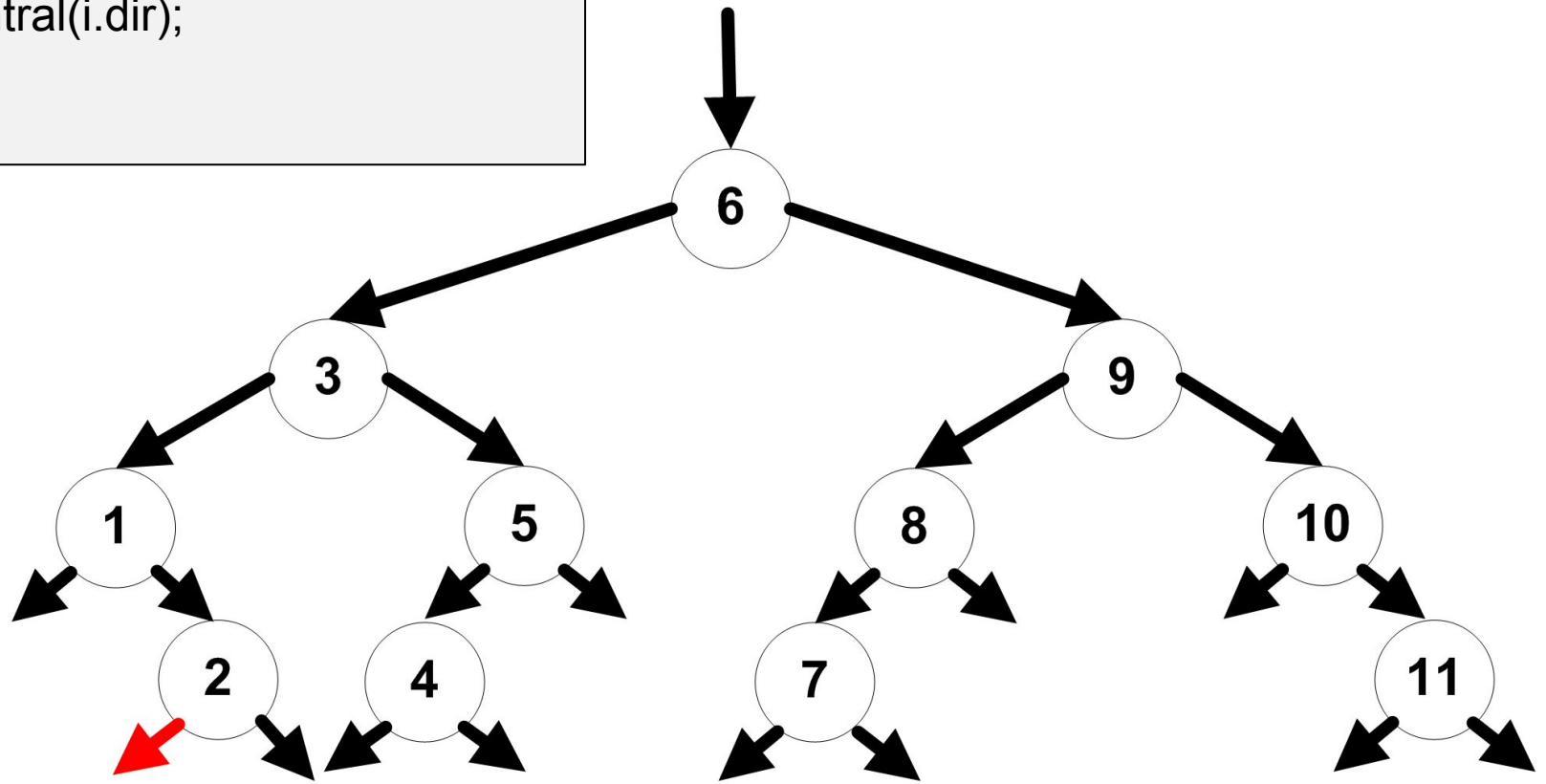
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1

Caminhamento Central ou Em Ordem

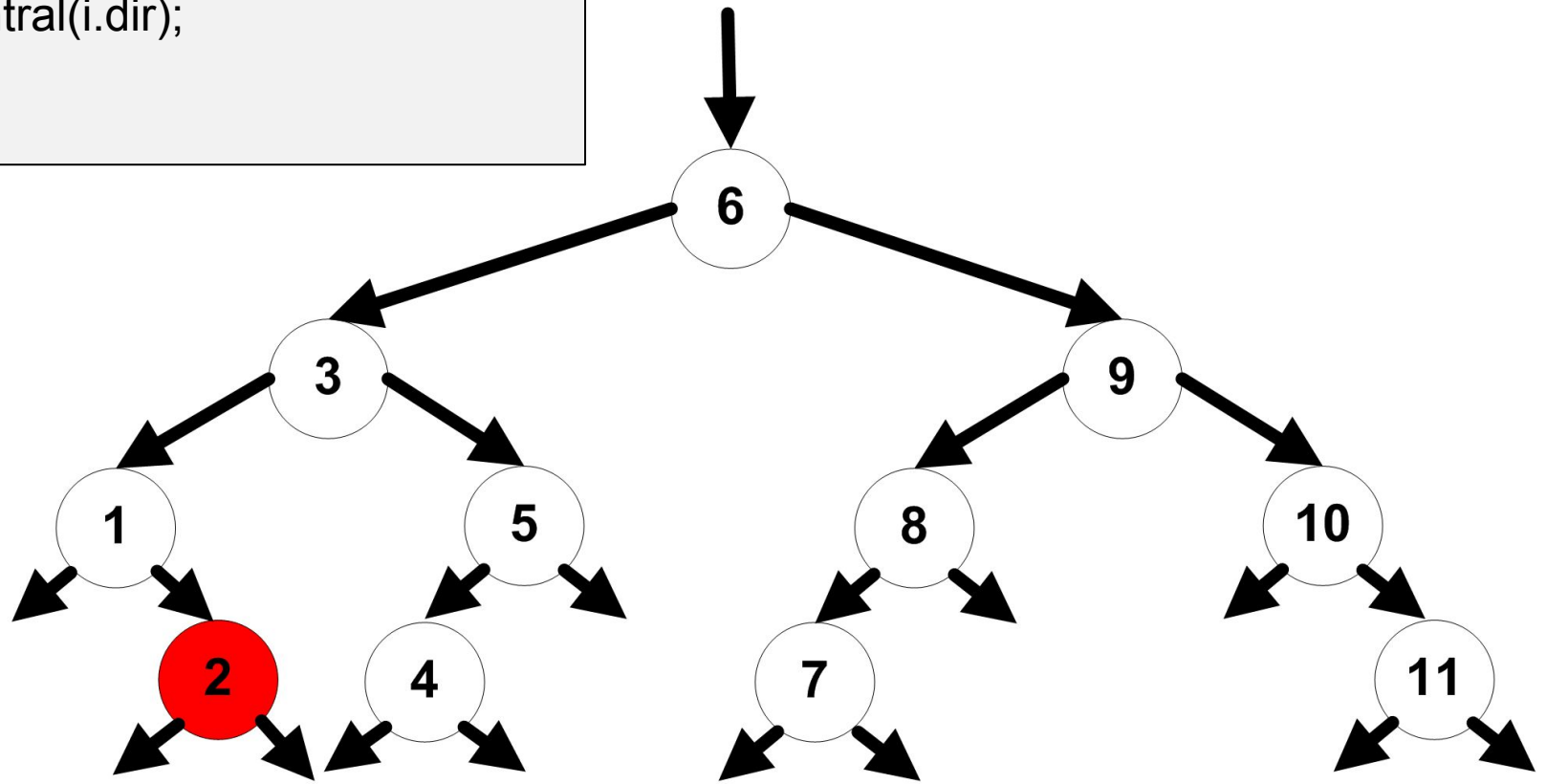
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1

Caminhamento Central ou Em Ordem

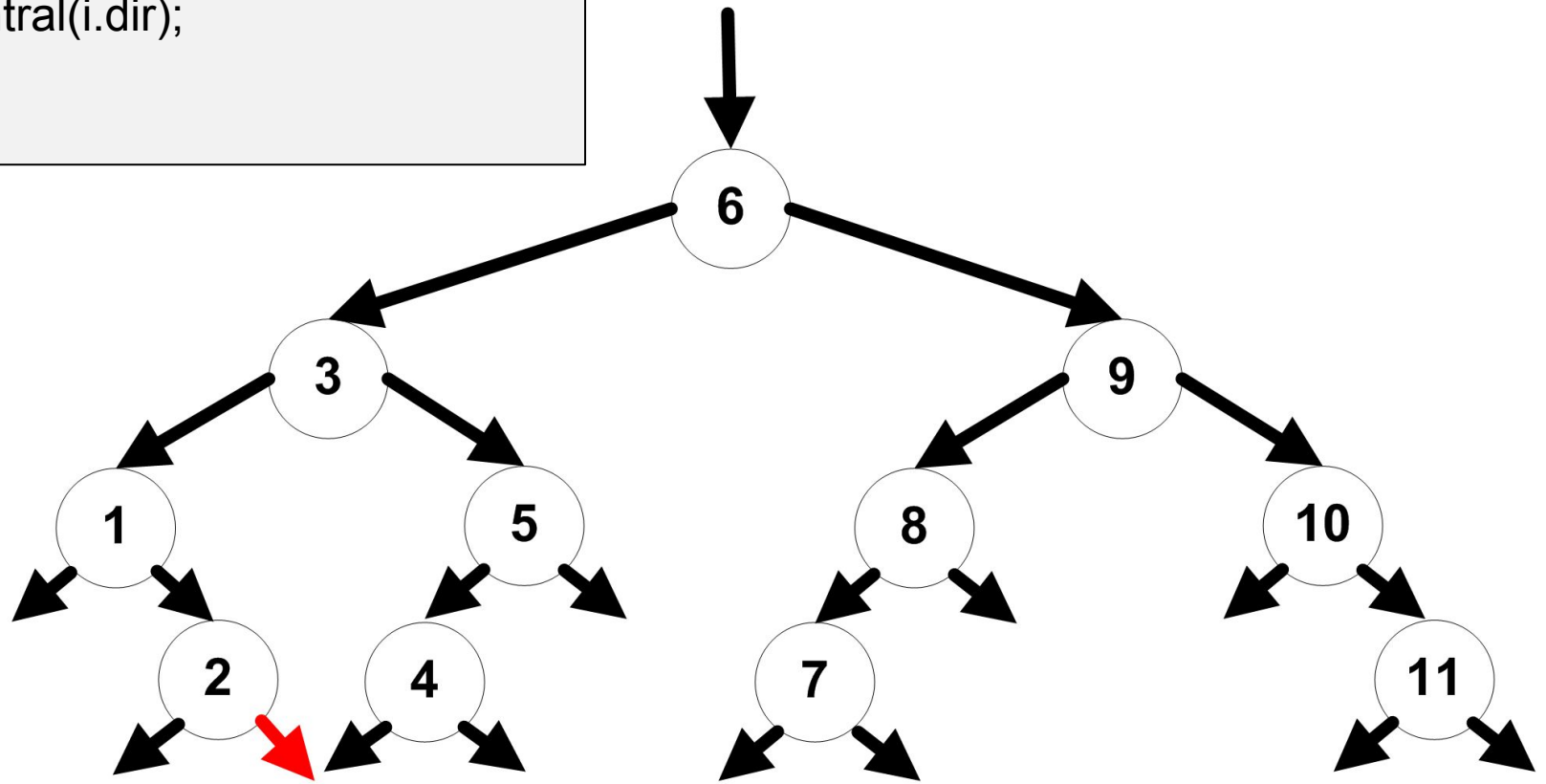
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2

Caminhamento Central ou Em Ordem

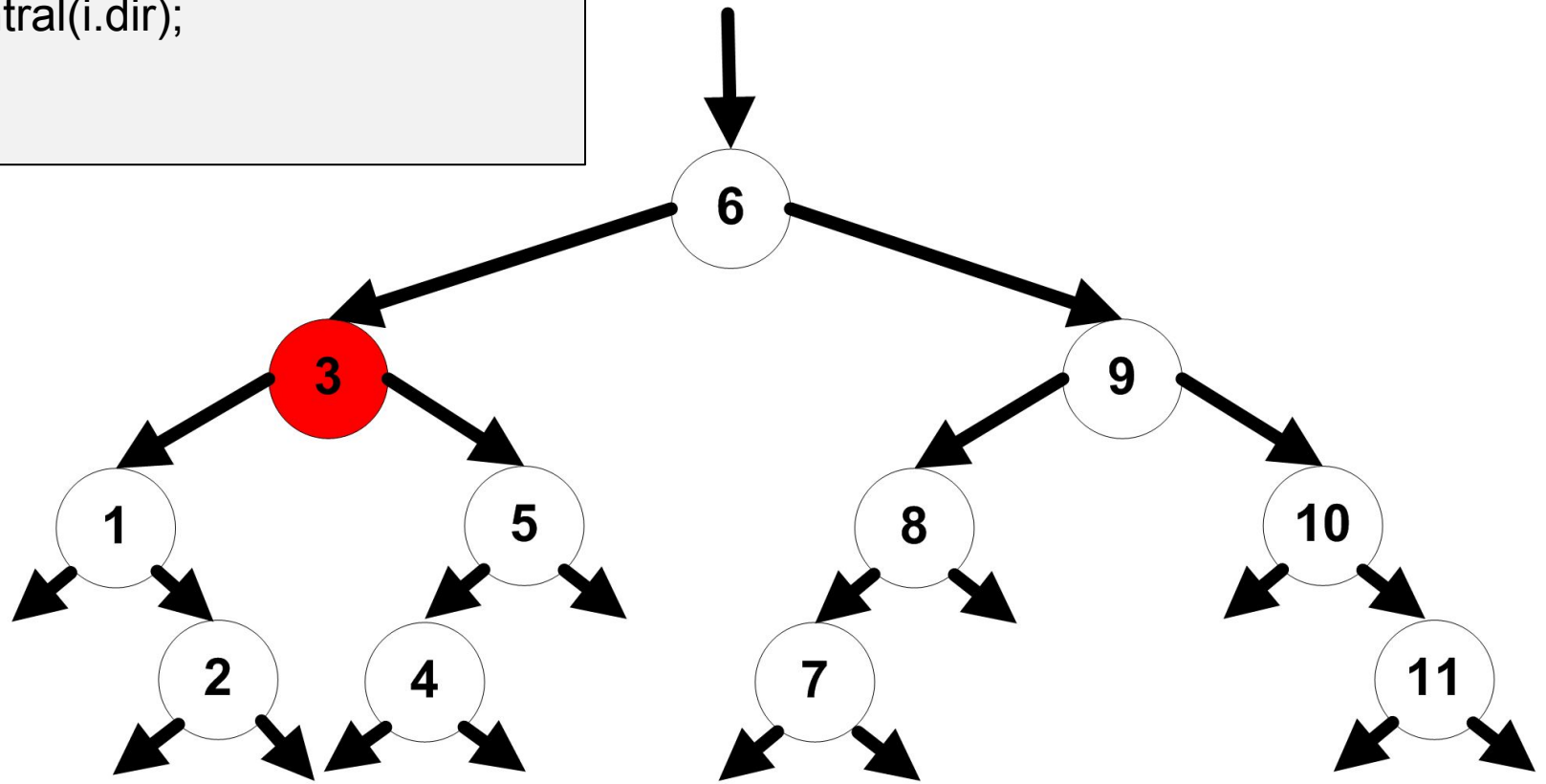
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2

Caminhamento Central ou Em Ordem

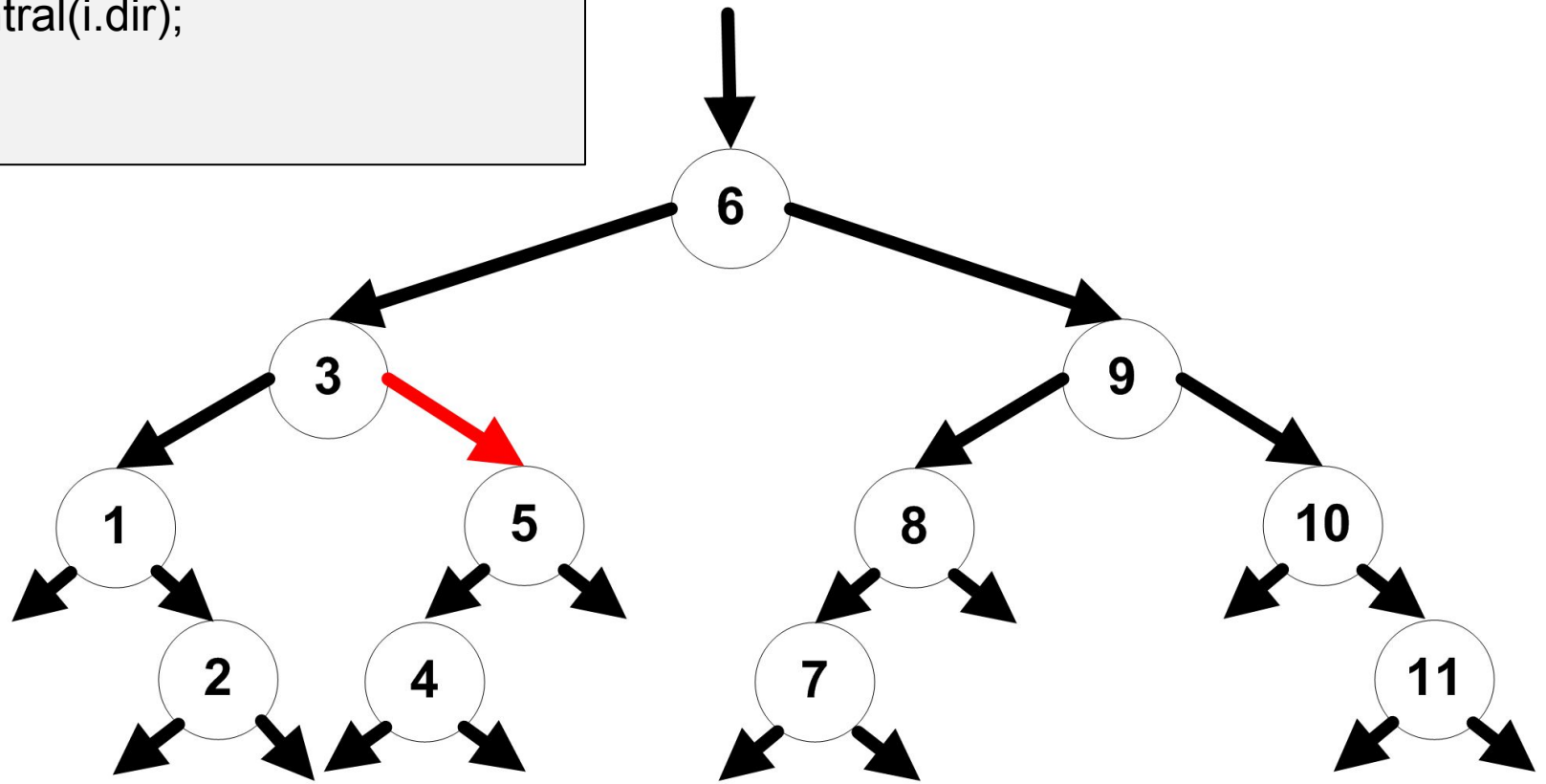
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3

Caminhamento Central ou Em Ordem

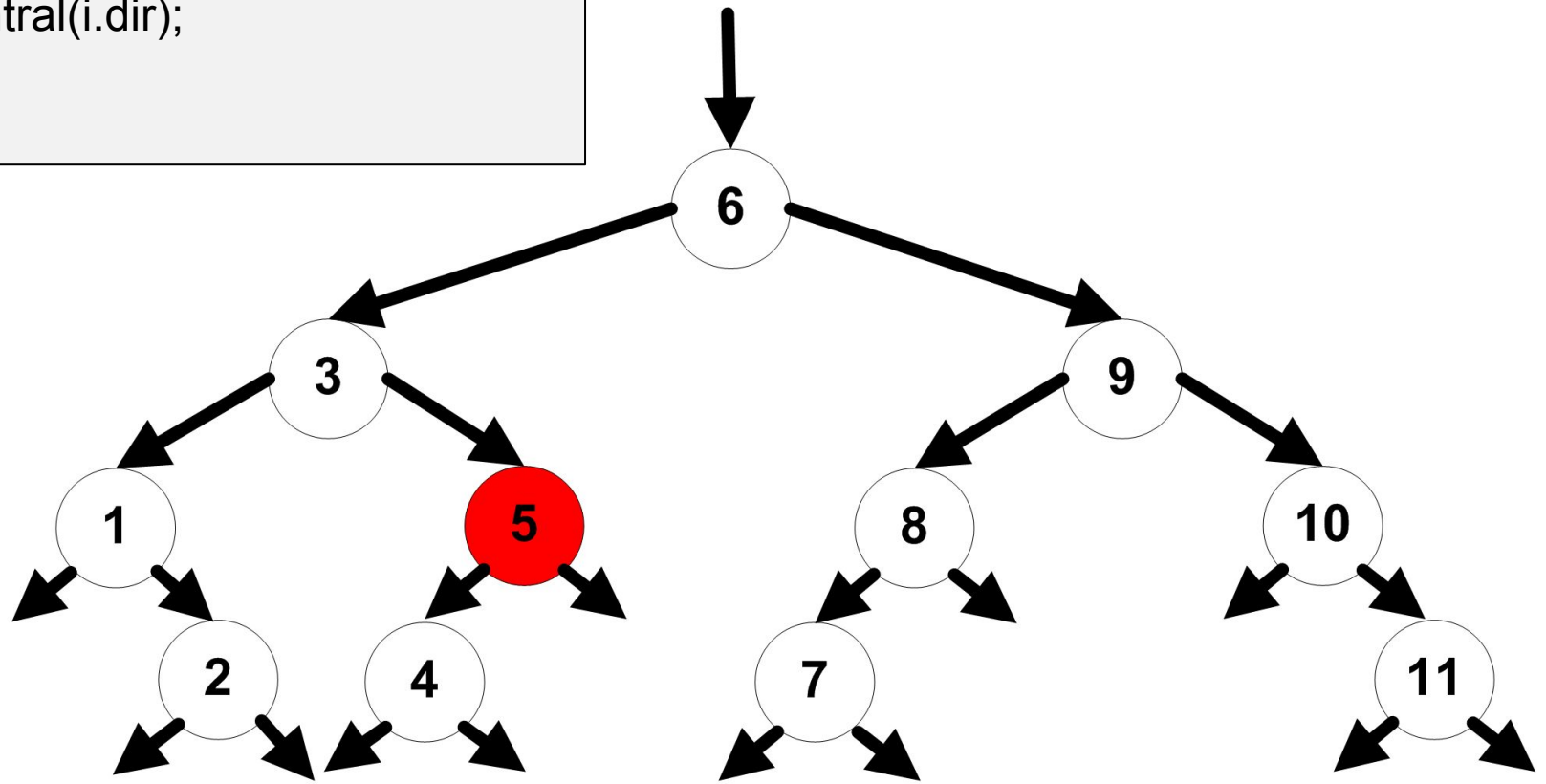
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3

Caminhamento Central ou Em Ordem

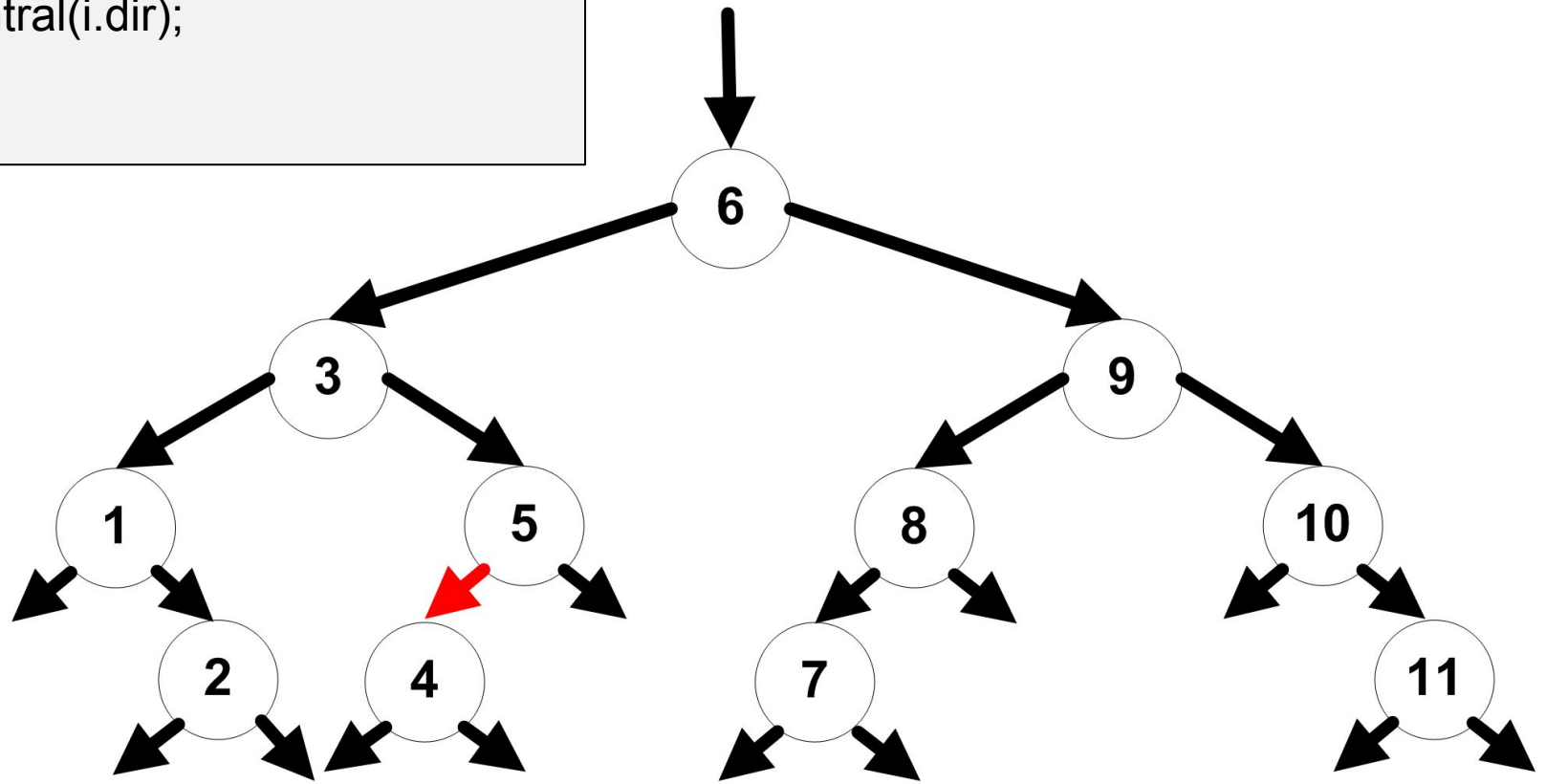
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3

Caminhamento Central ou Em Ordem

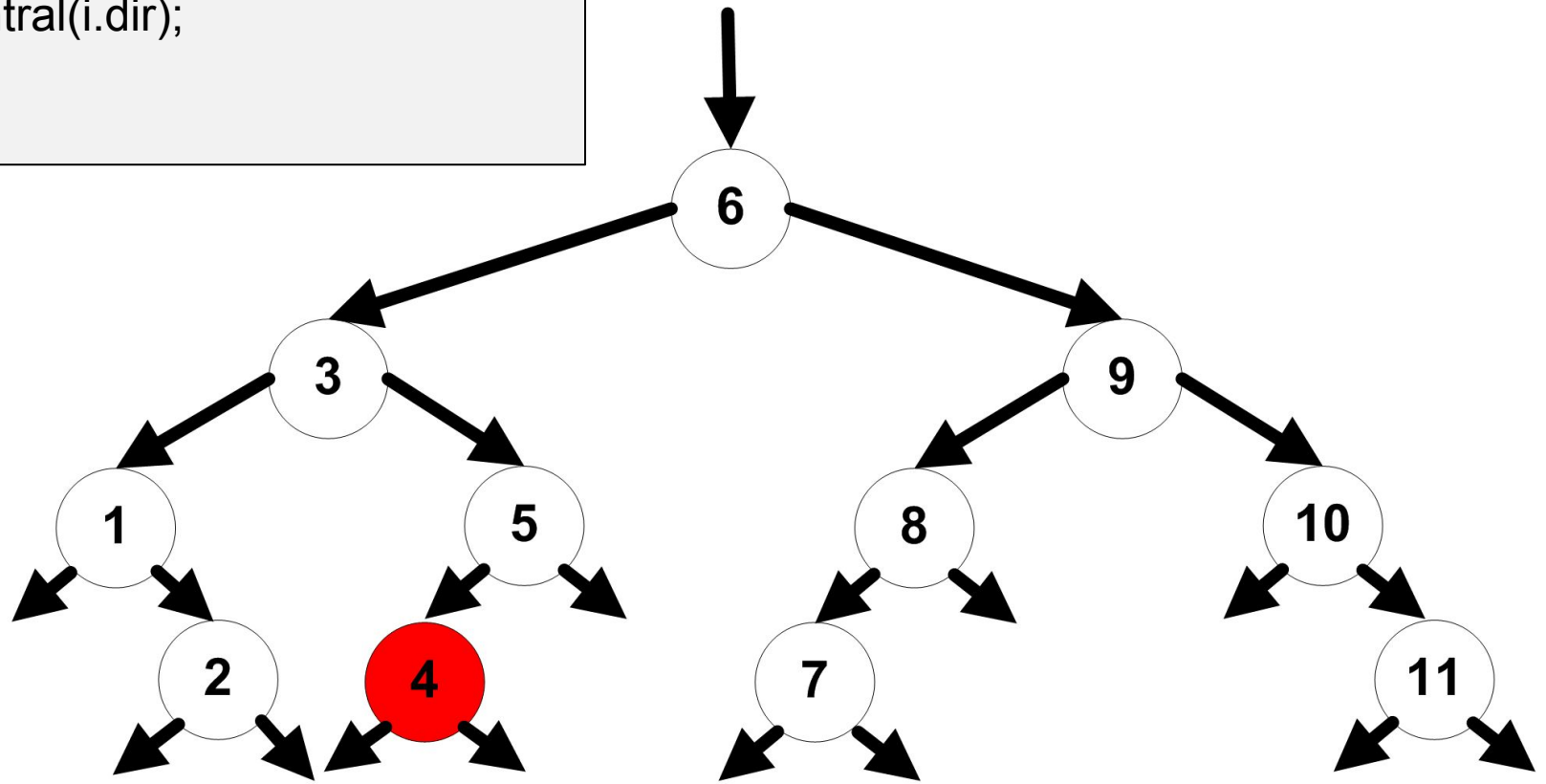
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	1 2 3

Caminhamento Central ou Em Ordem

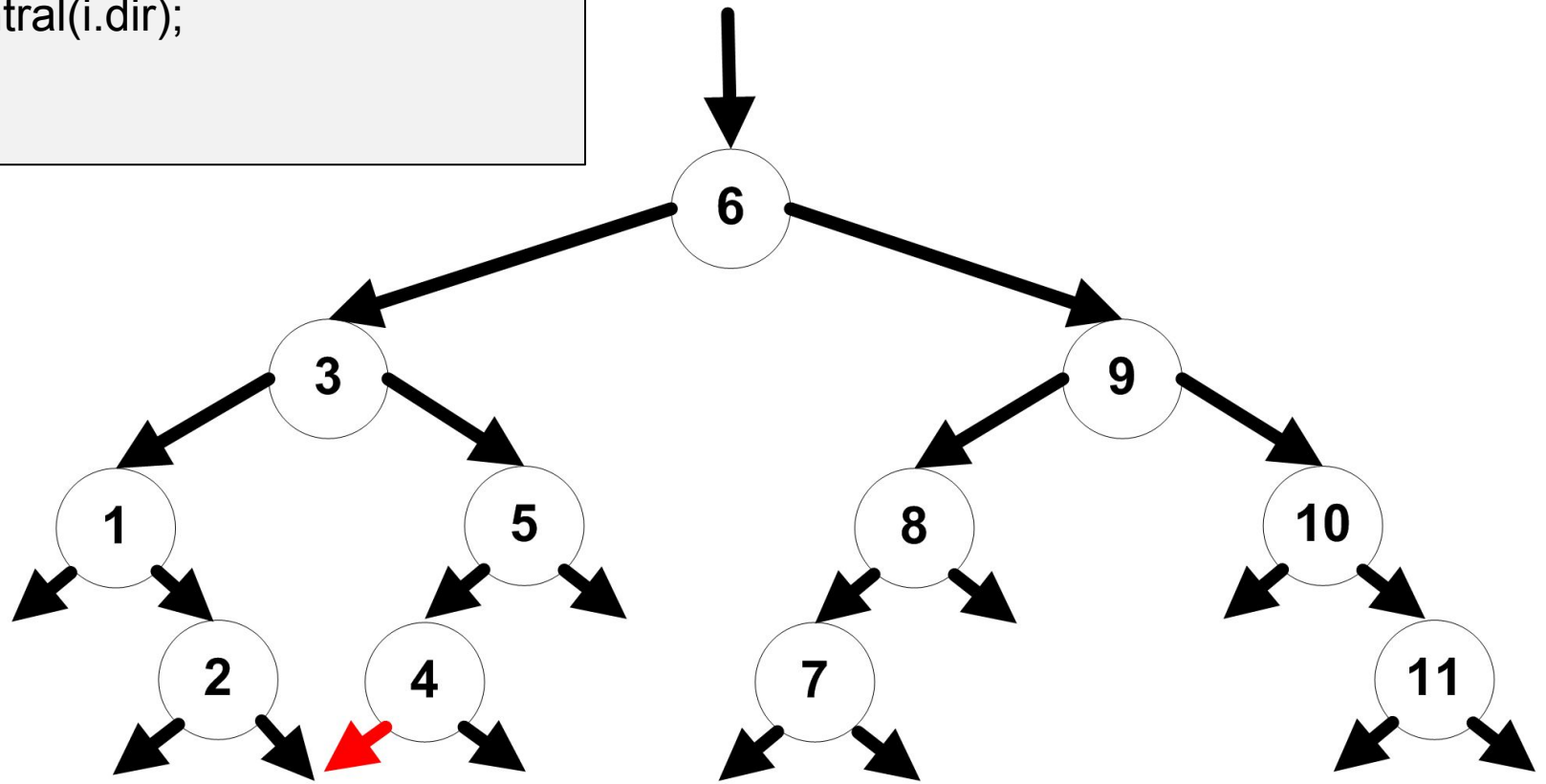
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3

Caminhamento Central ou Em Ordem

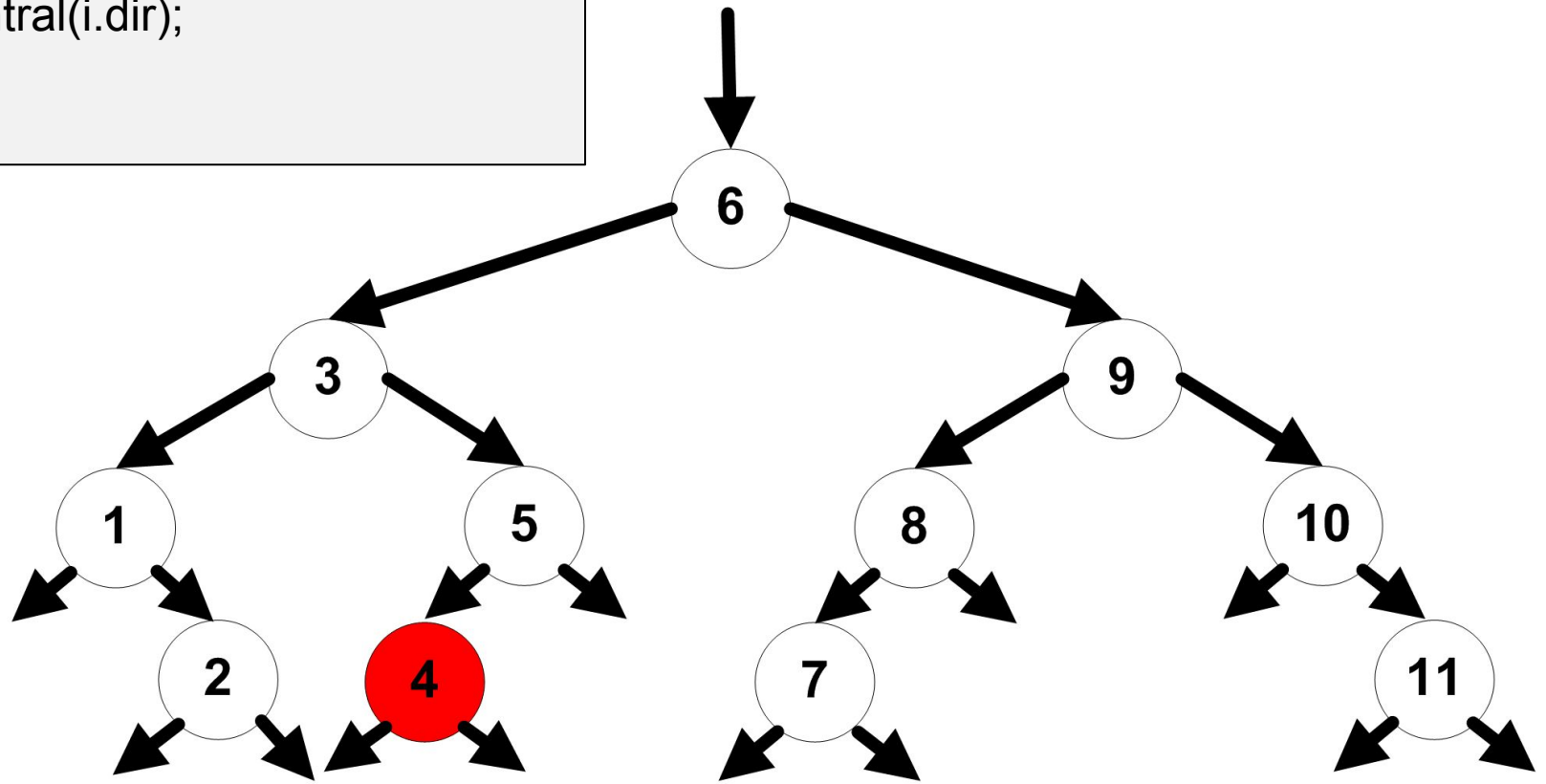
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3

Caminhamento Central ou Em Ordem

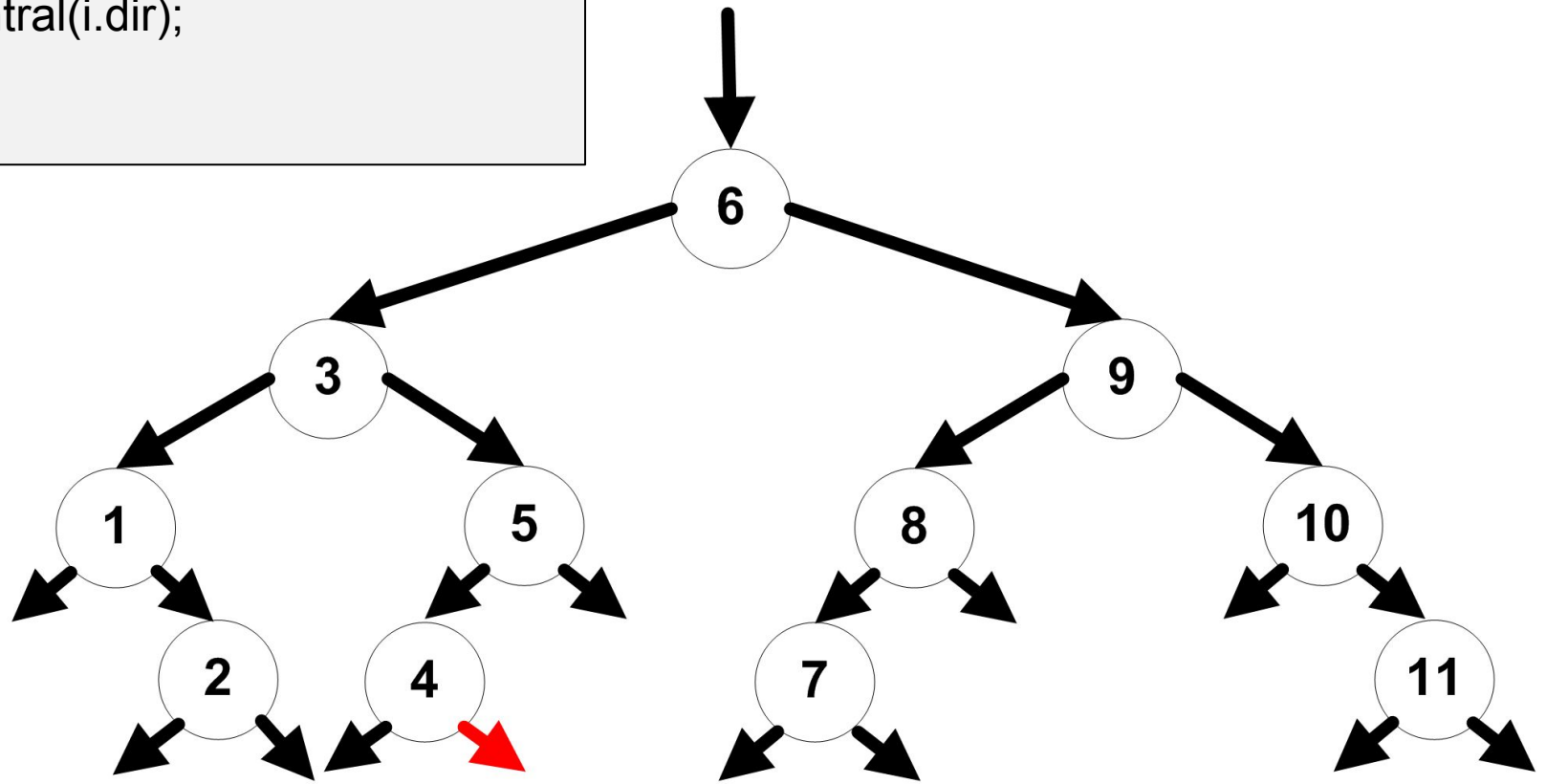
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3 4

Caminhamento Central ou Em Ordem

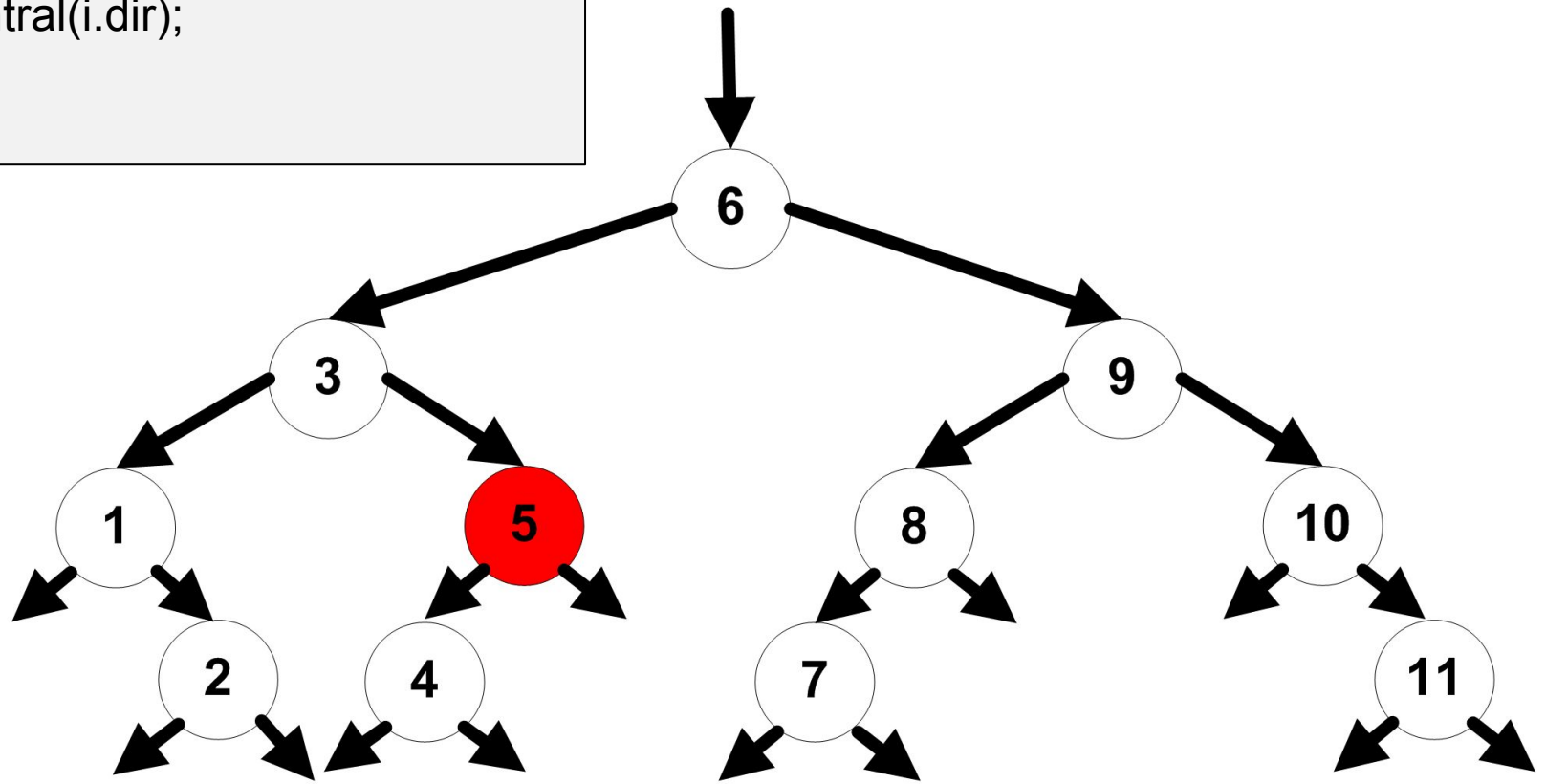
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3 4

Caminhamento Central ou Em Ordem

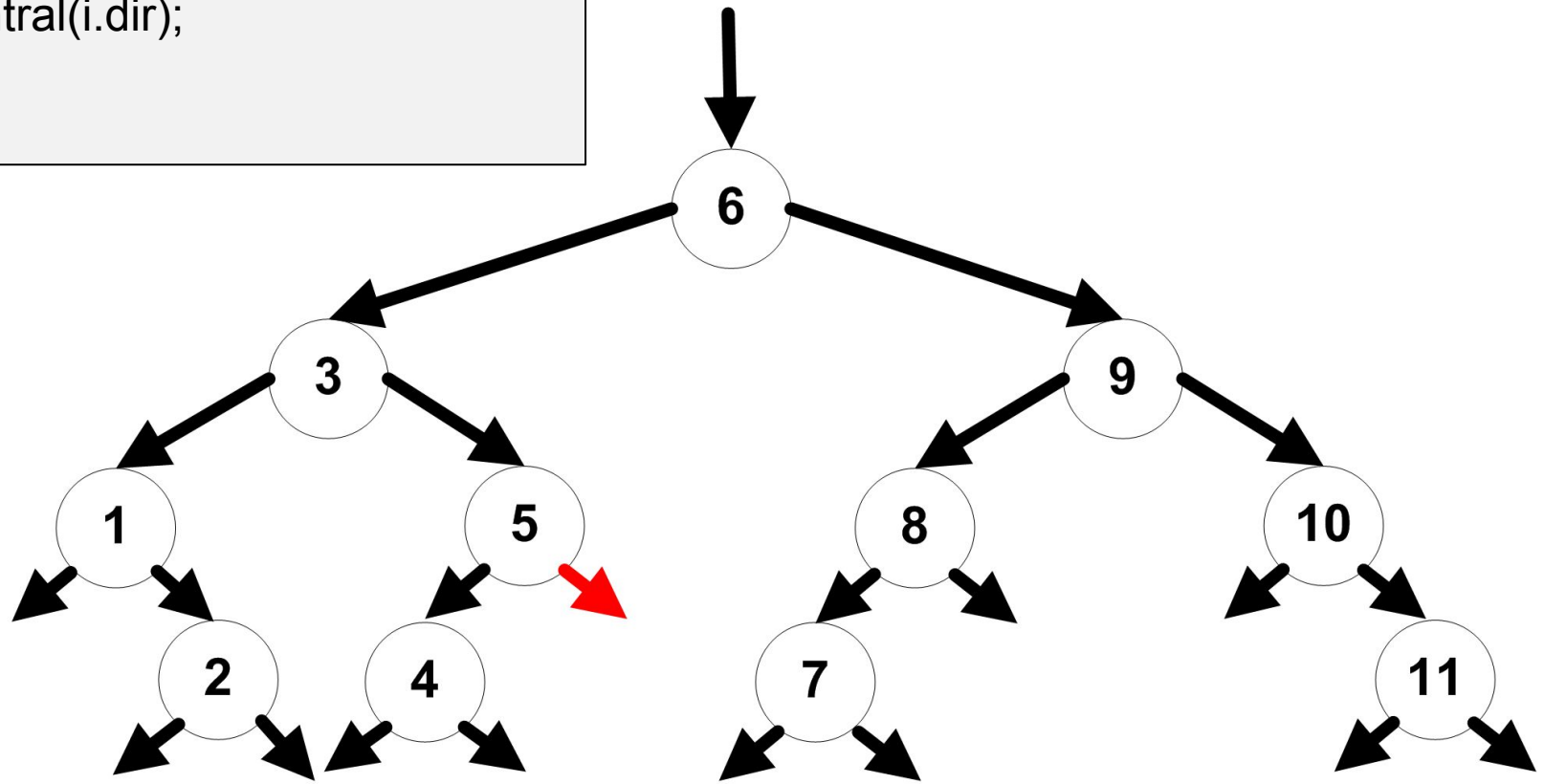
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	1 2 3 4 5

Caminhamento Central ou Em Ordem

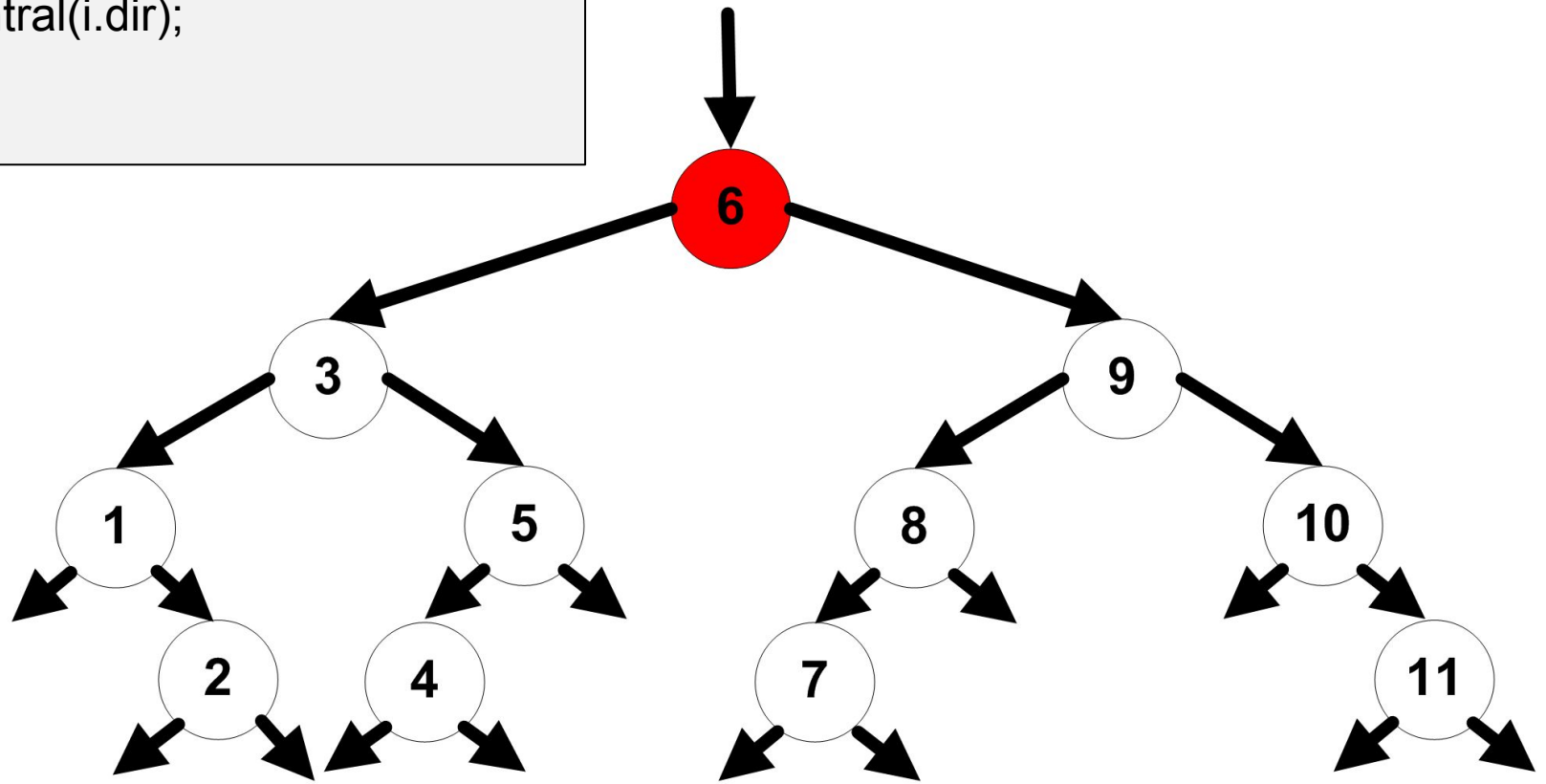
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	1 2 3 4 5

Caminhamento Central ou Em Ordem

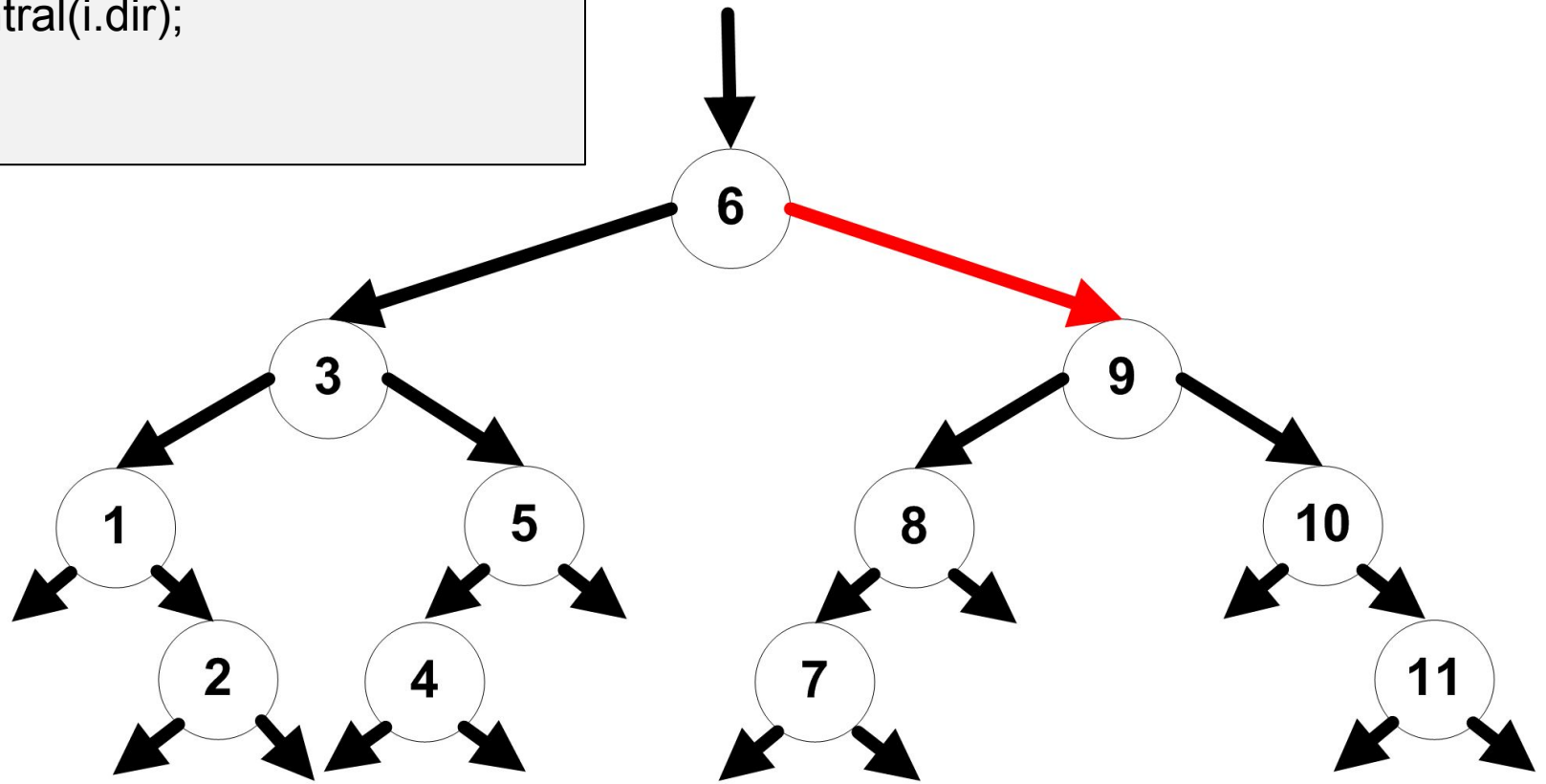
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6

Caminhamento Central ou Em Ordem

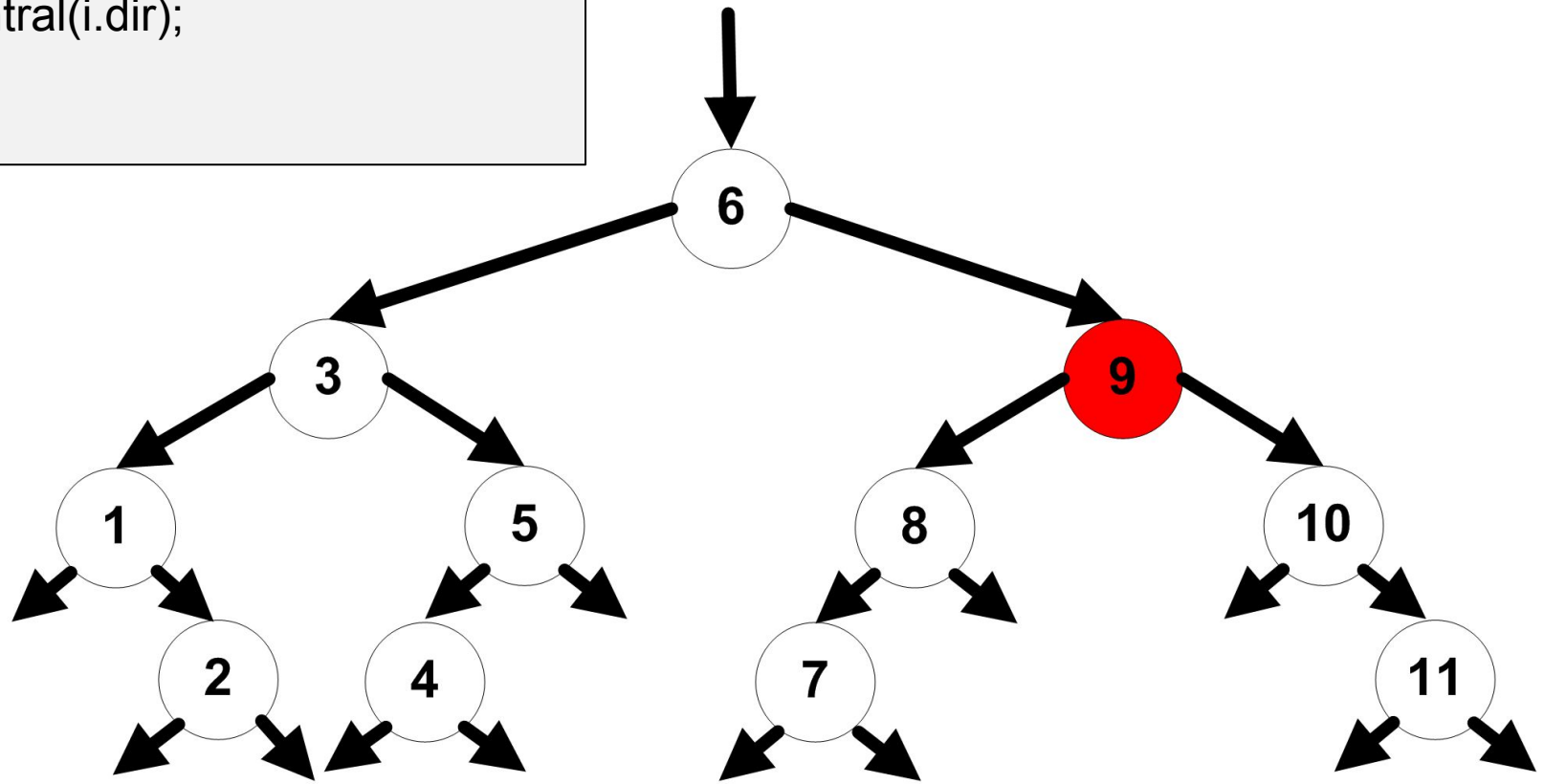
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6

Caminhamento Central ou Em Ordem

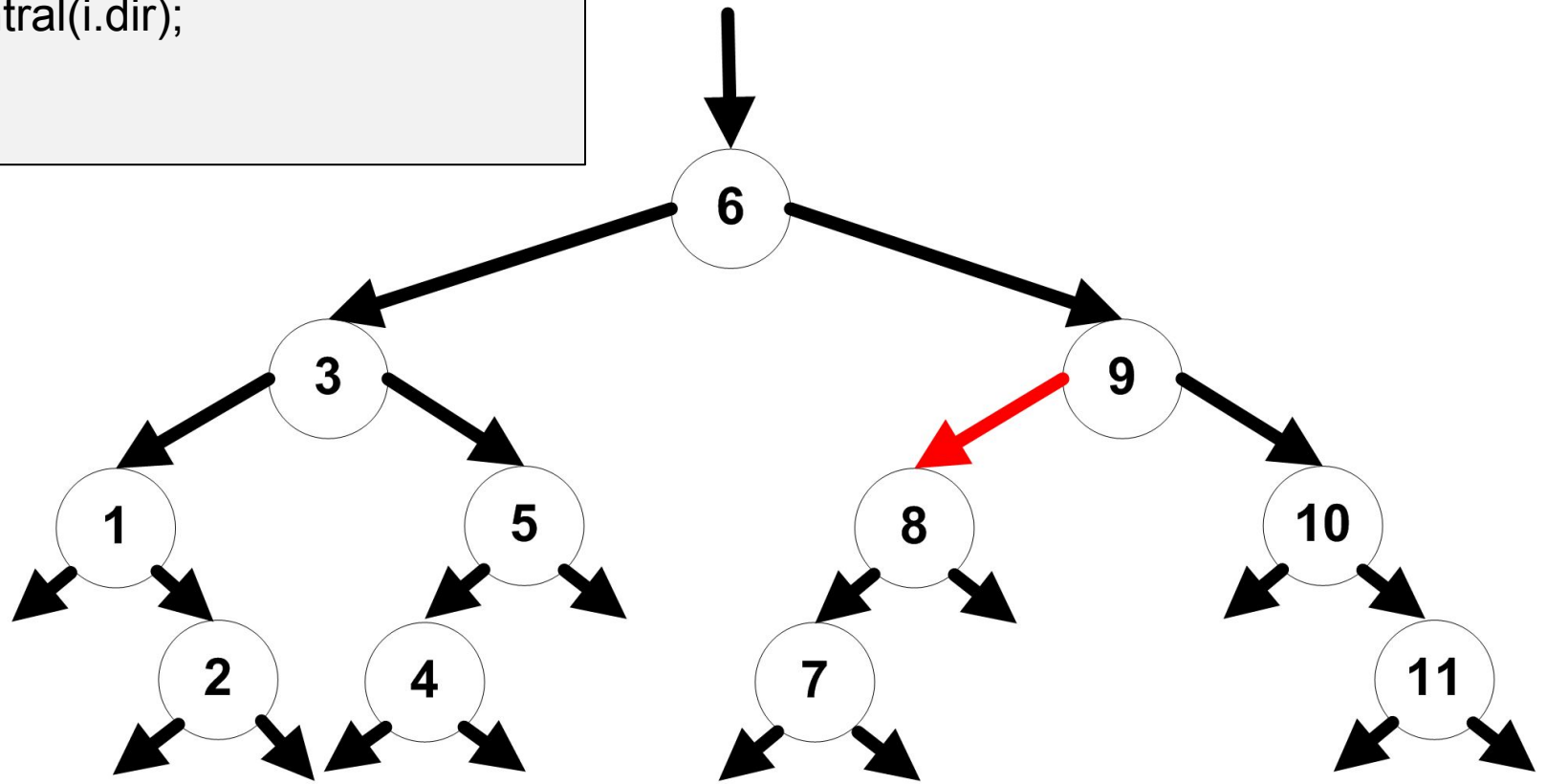
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3 4 5 6

Caminhamento Central ou Em Ordem

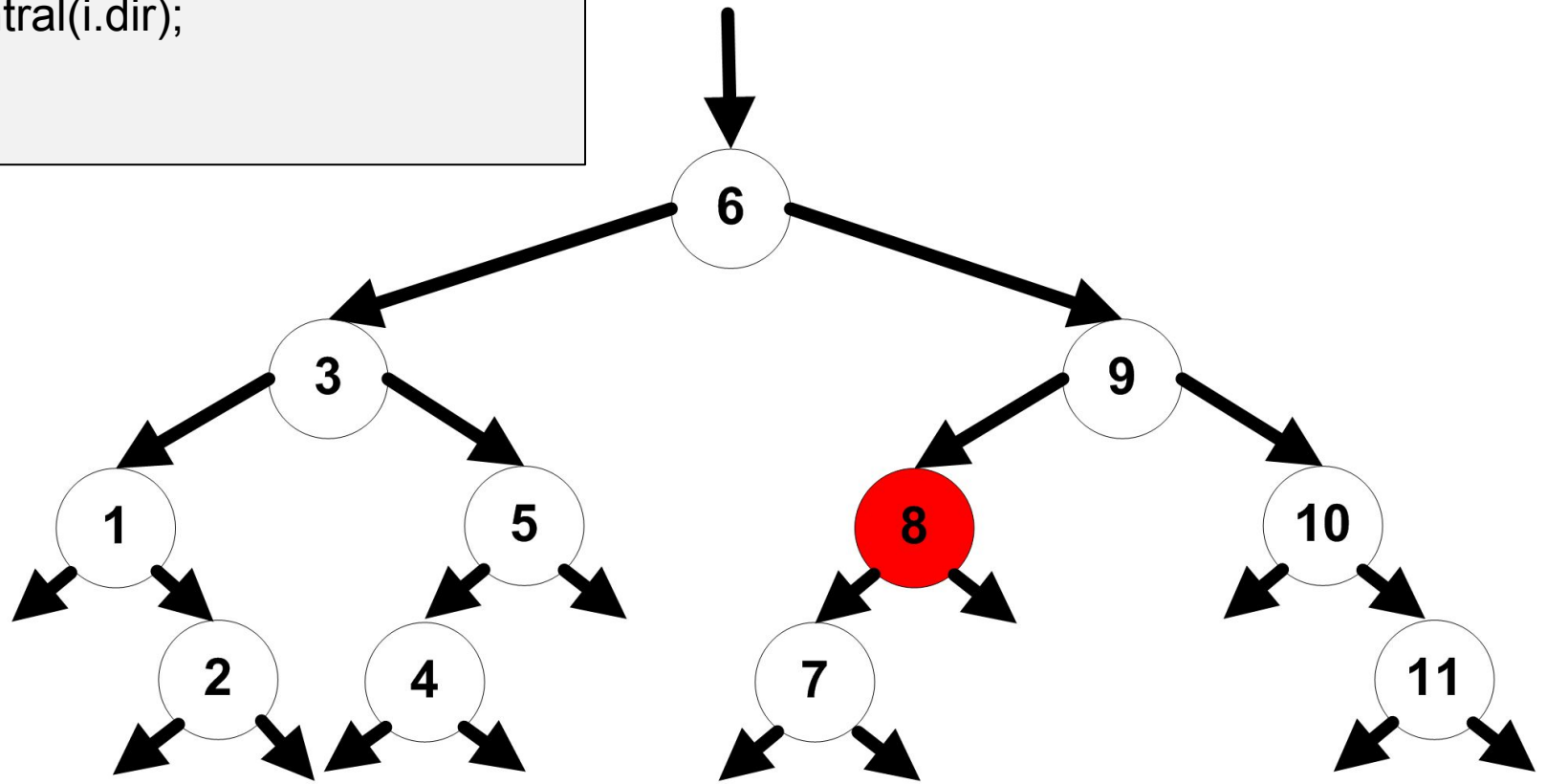
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6

Caminhamento Central ou Em Ordem

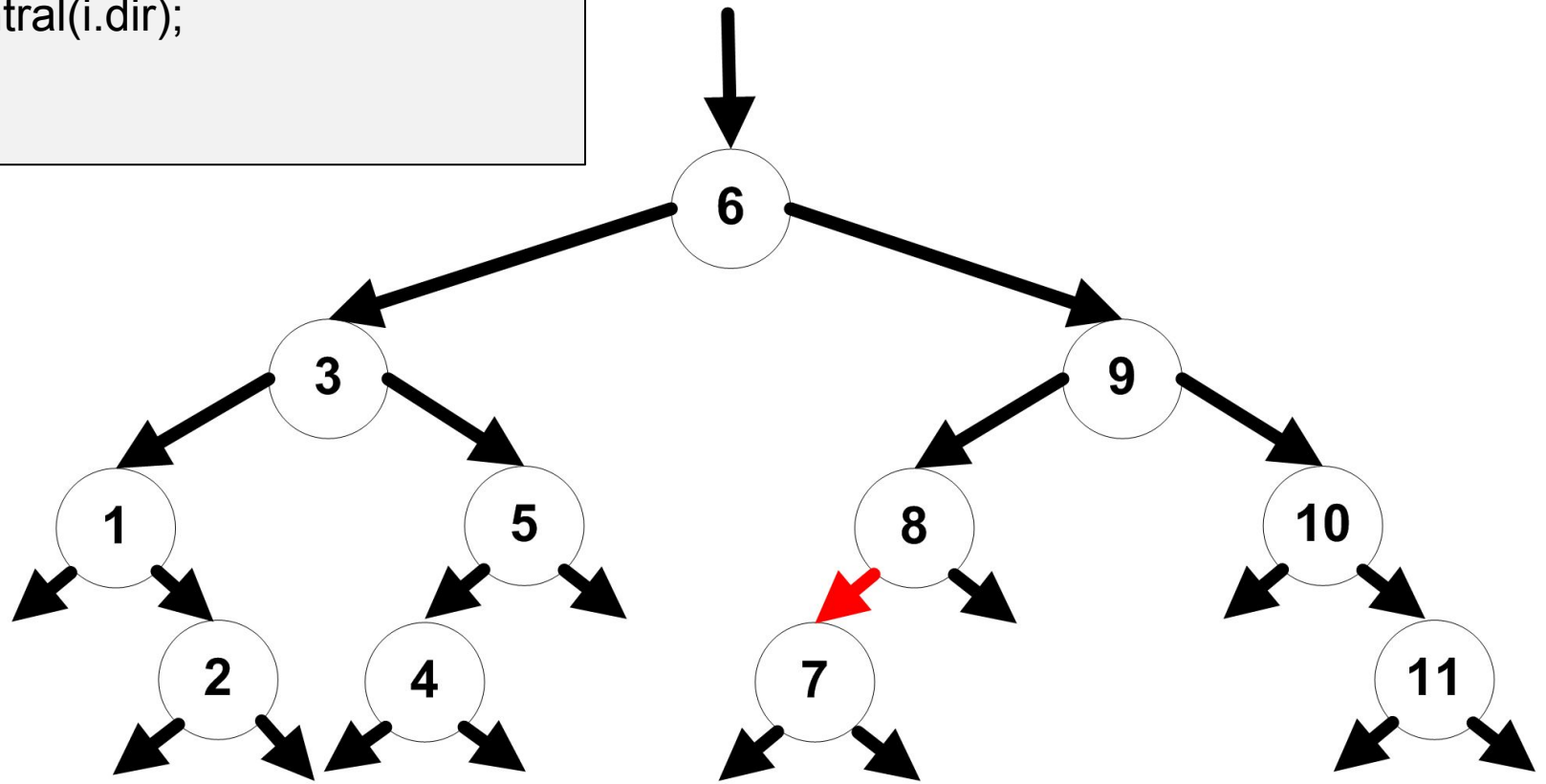
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	1	2	3	4	5	6

Caminhamento Central ou Em Ordem

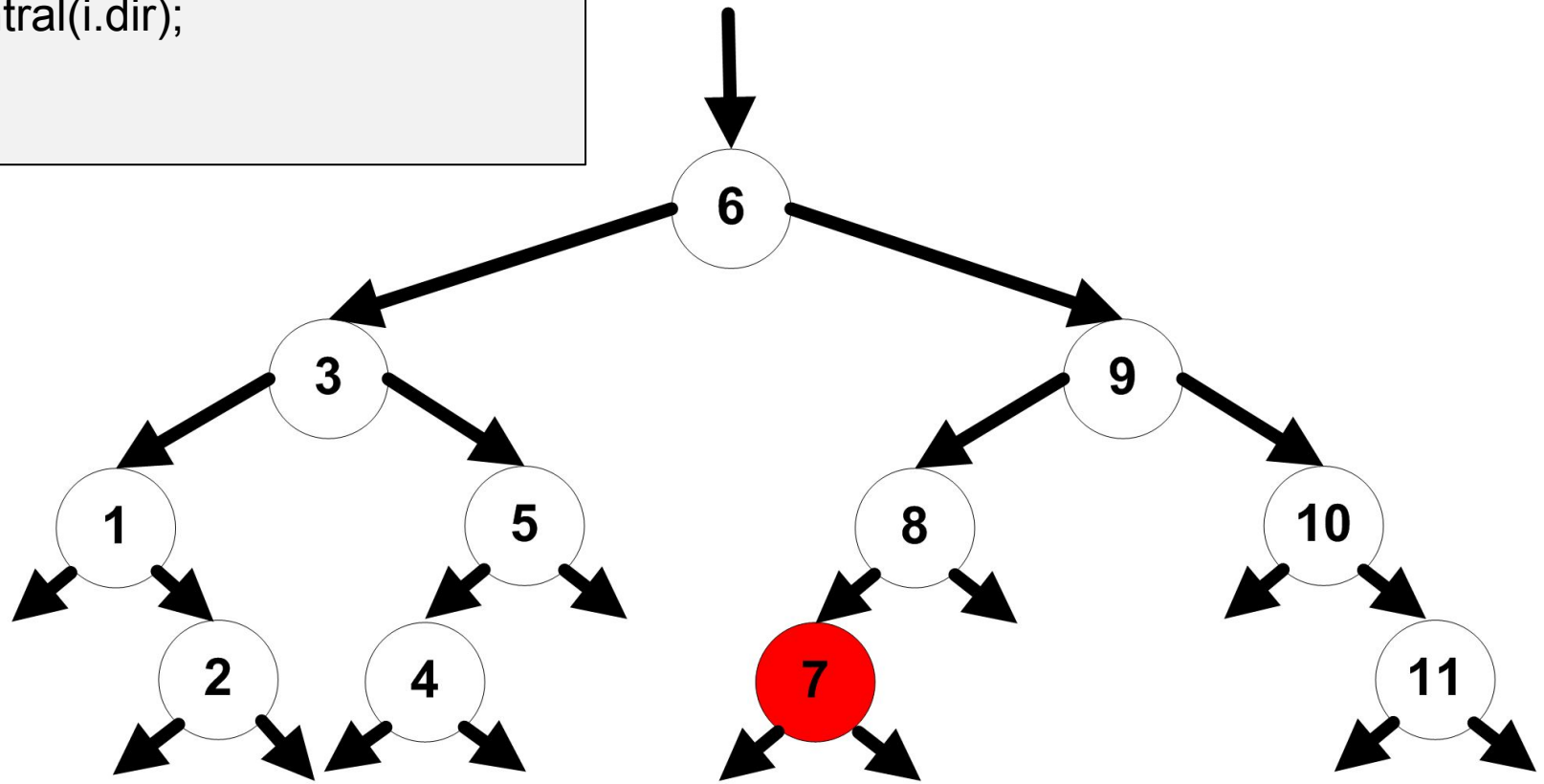
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6

Caminhamento Central ou Em Ordem

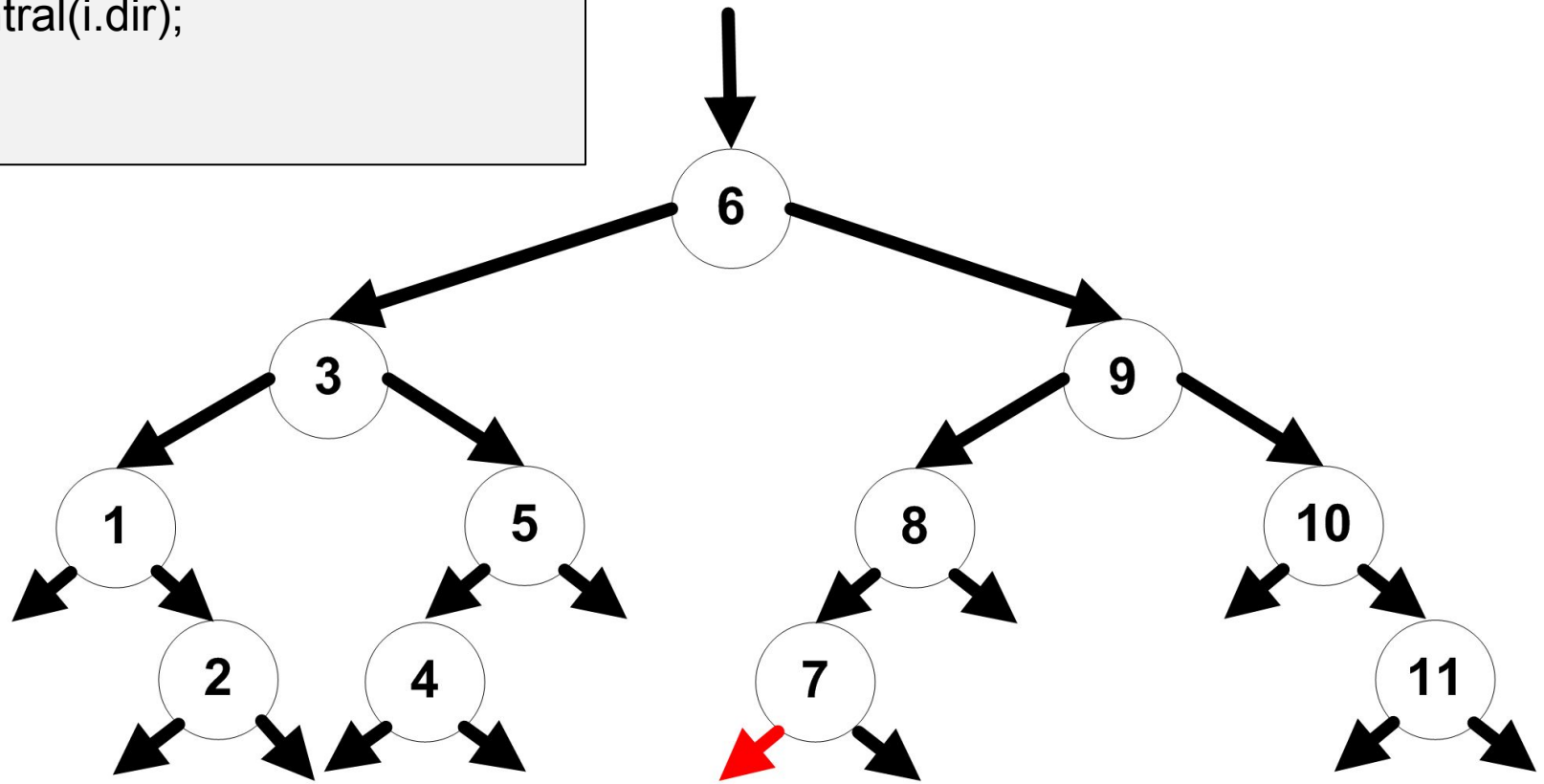
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1 2 3 4 5 6

Caminhamento Central ou Em Ordem

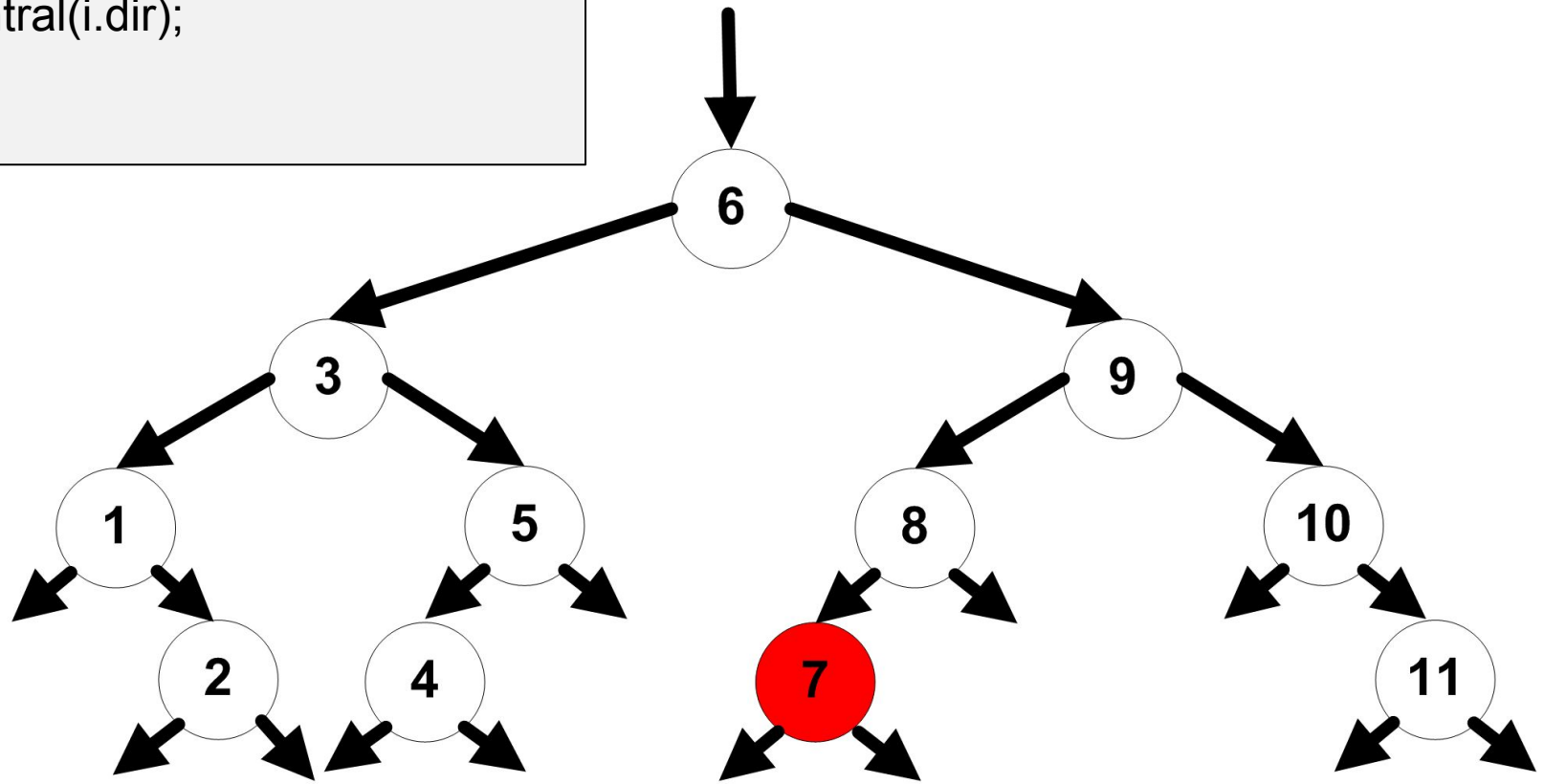
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6

Caminhamento Central ou Em Ordem

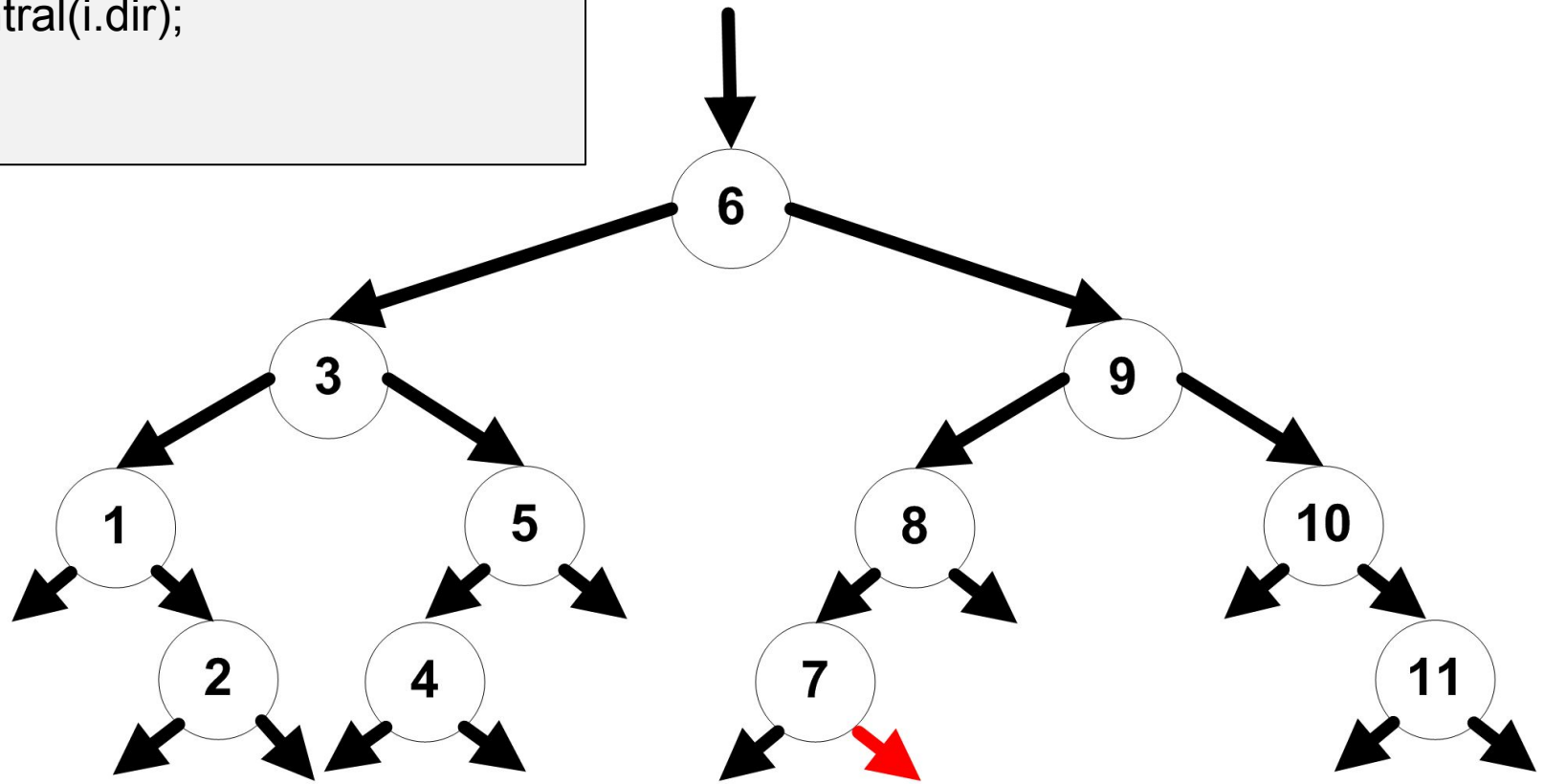
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7

Caminhamento Central ou Em Ordem

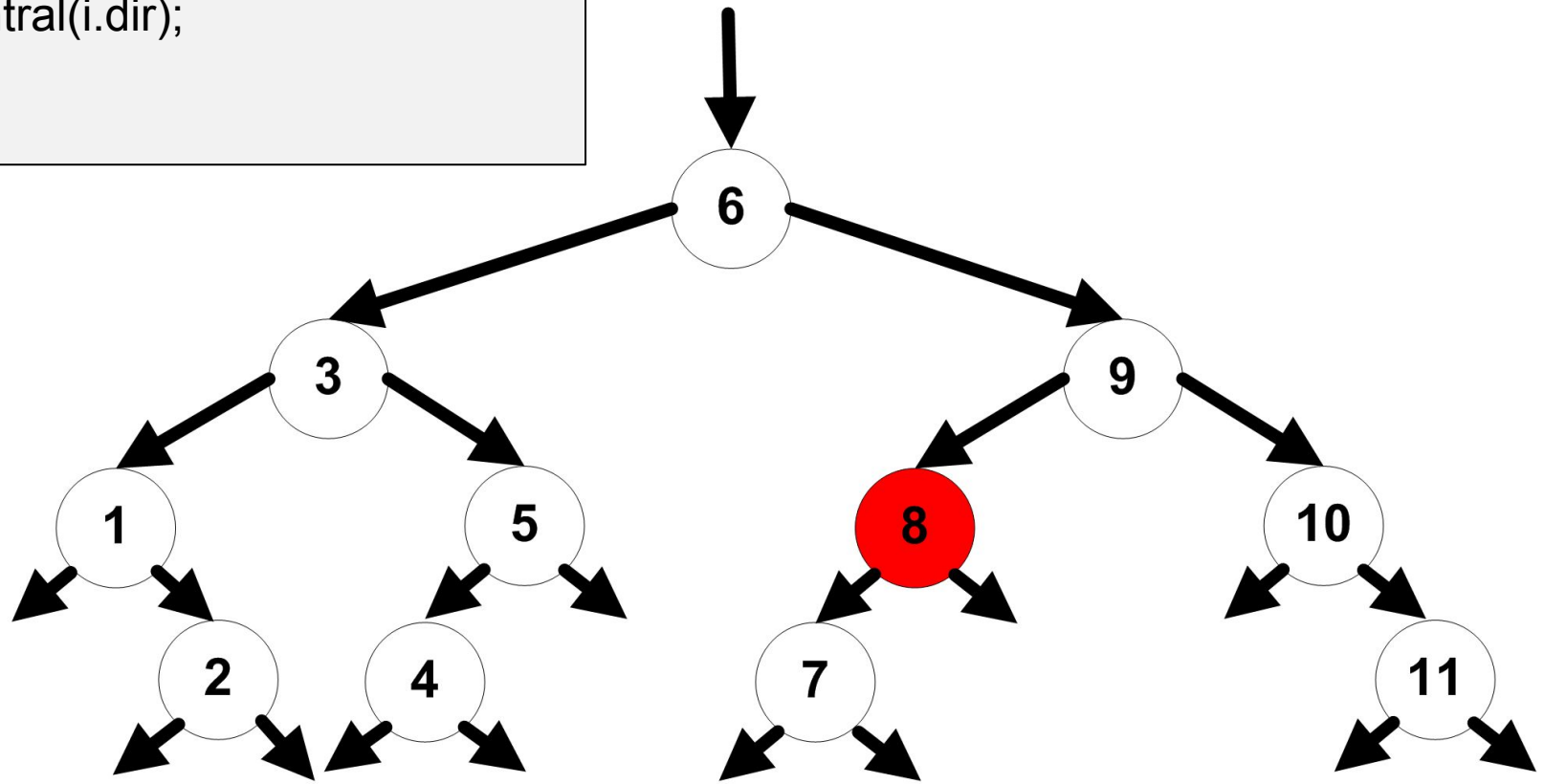
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7

Caminhamento Central ou Em Ordem

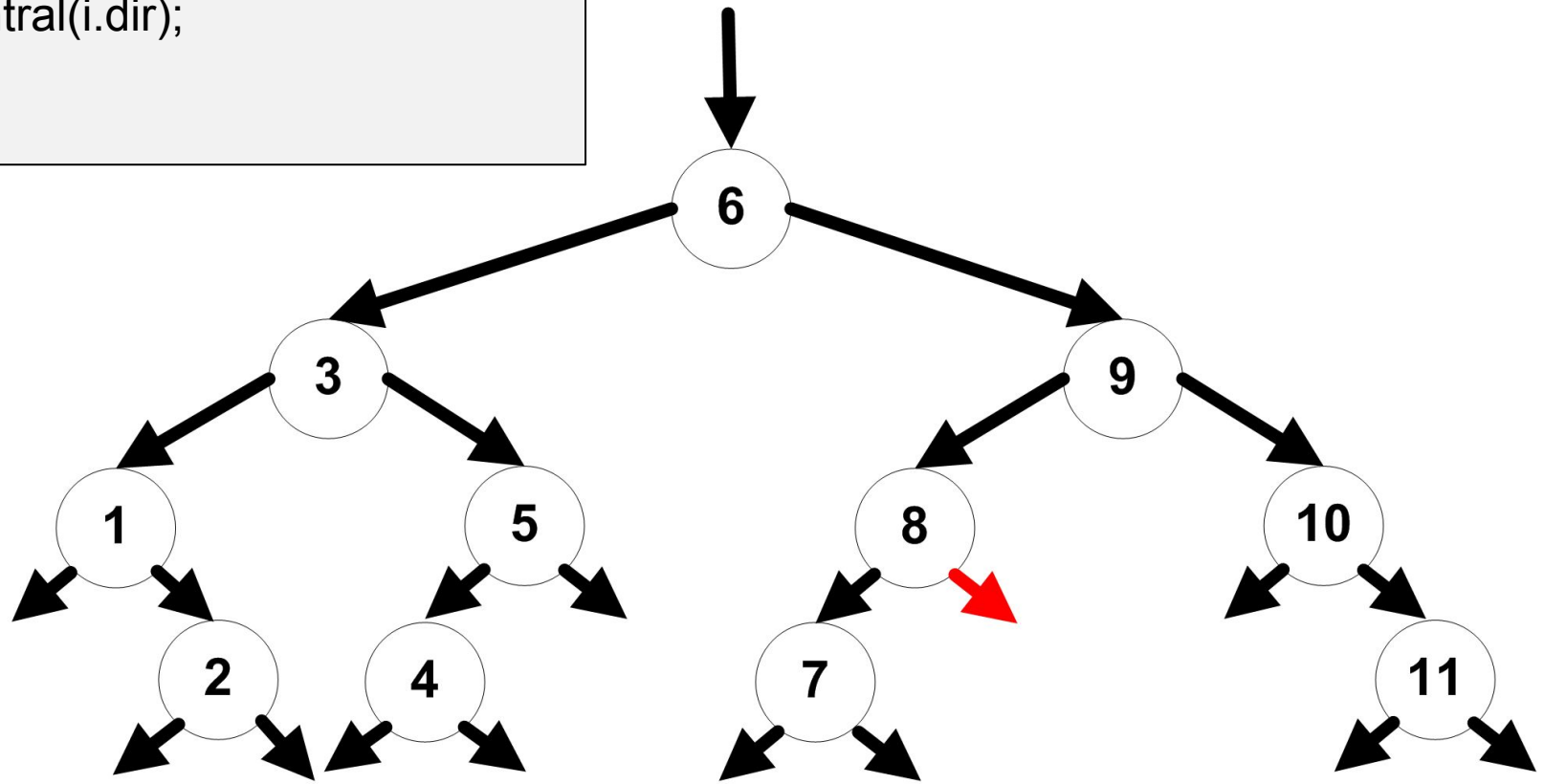
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8

Caminhamento Central ou Em Ordem

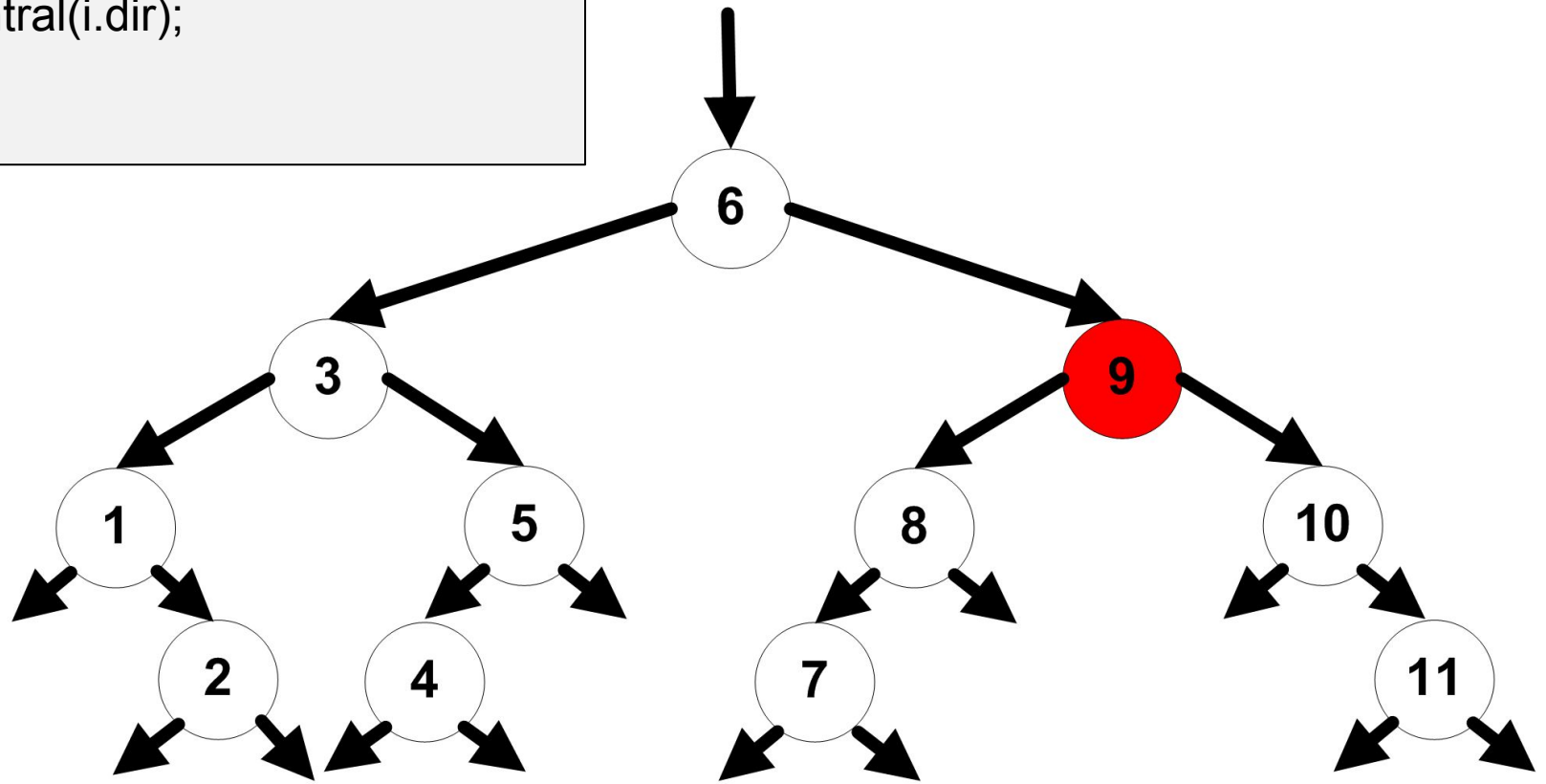
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8

Caminhamento Central ou Em Ordem

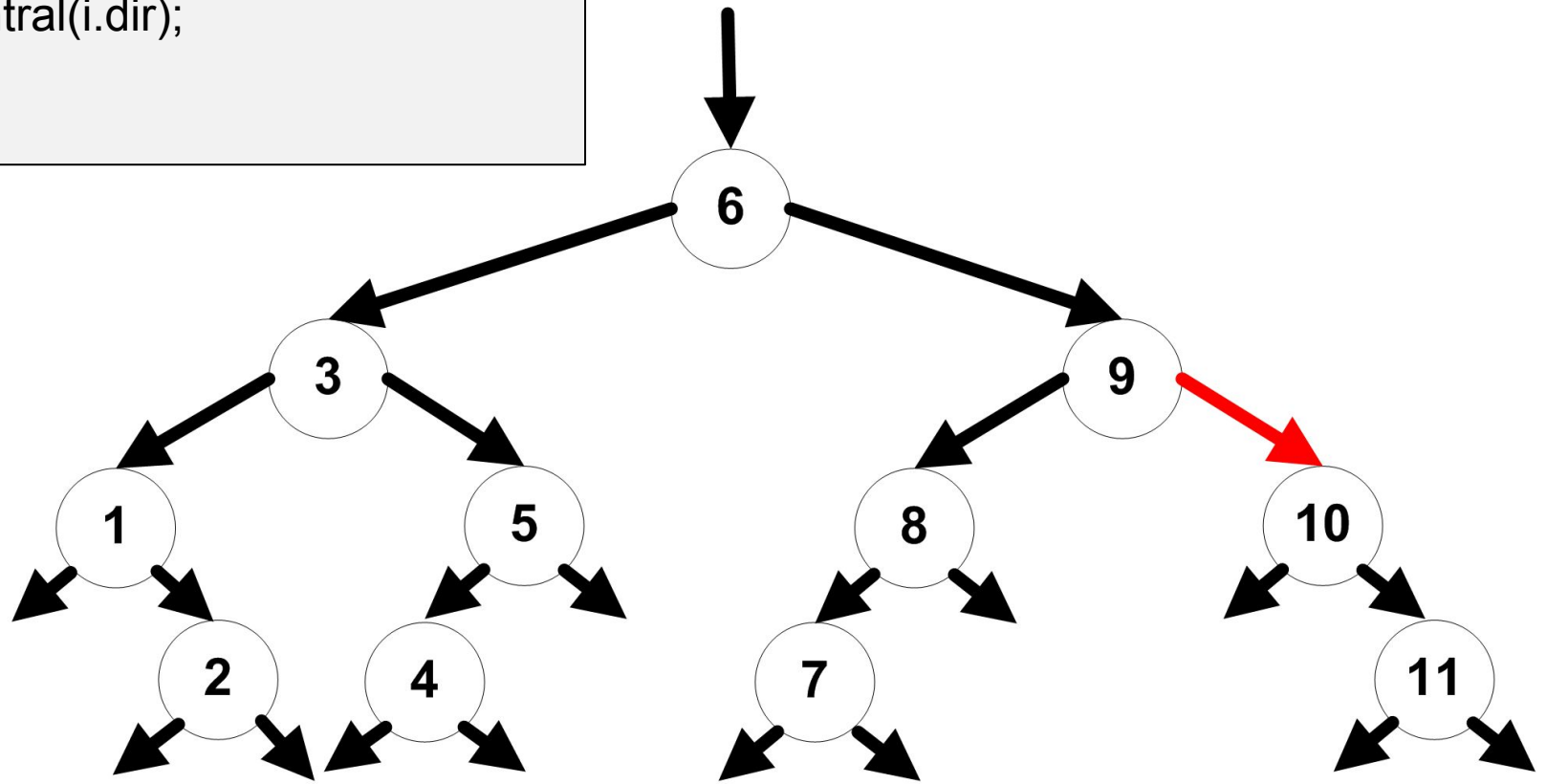
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	1	2	3	4	5	6	7	8	9

Caminhamento Central ou Em Ordem

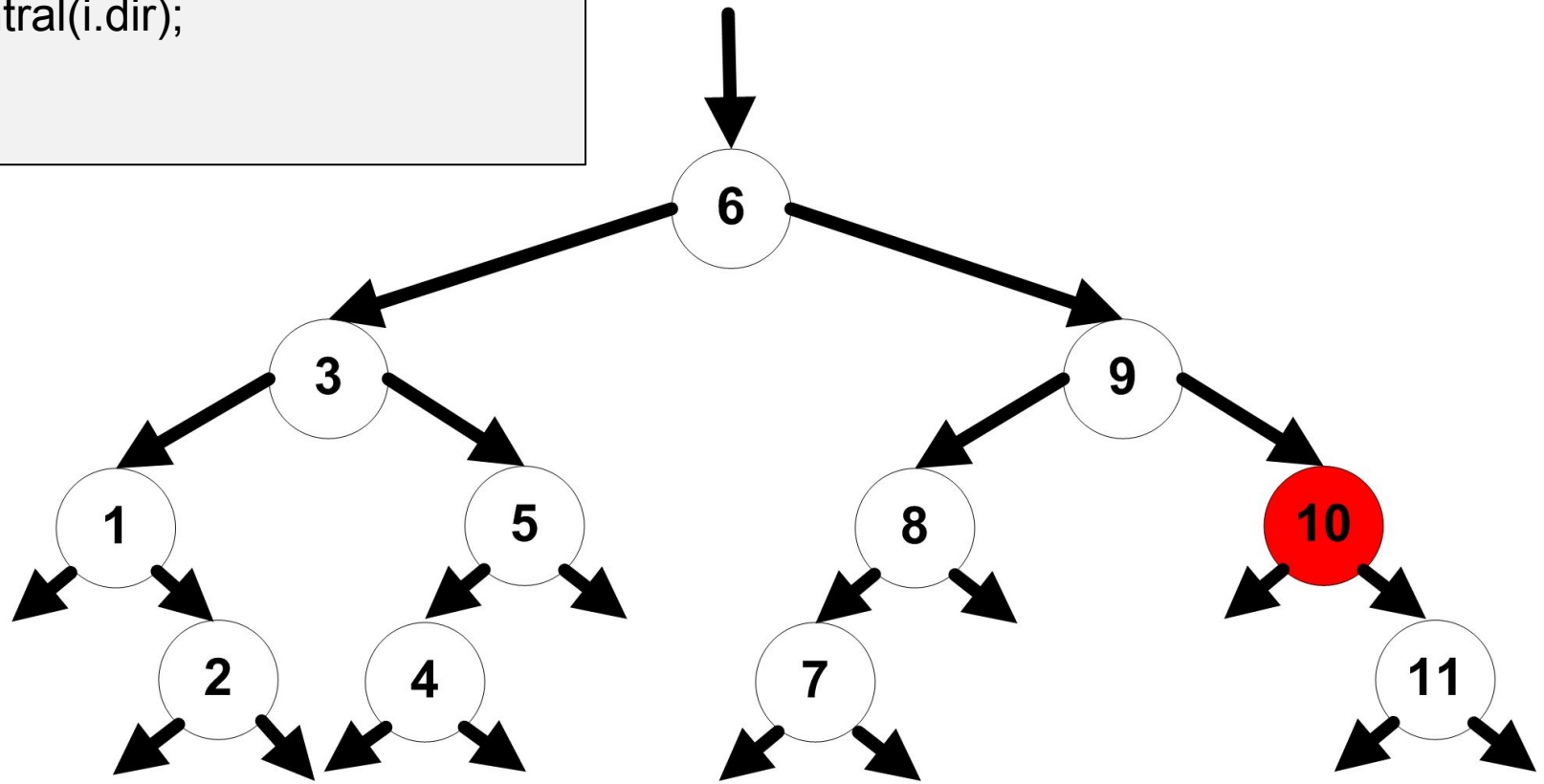
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



Tela	1	2	3	4	5	6	7	8	9

Caminhamento Central ou Em Ordem

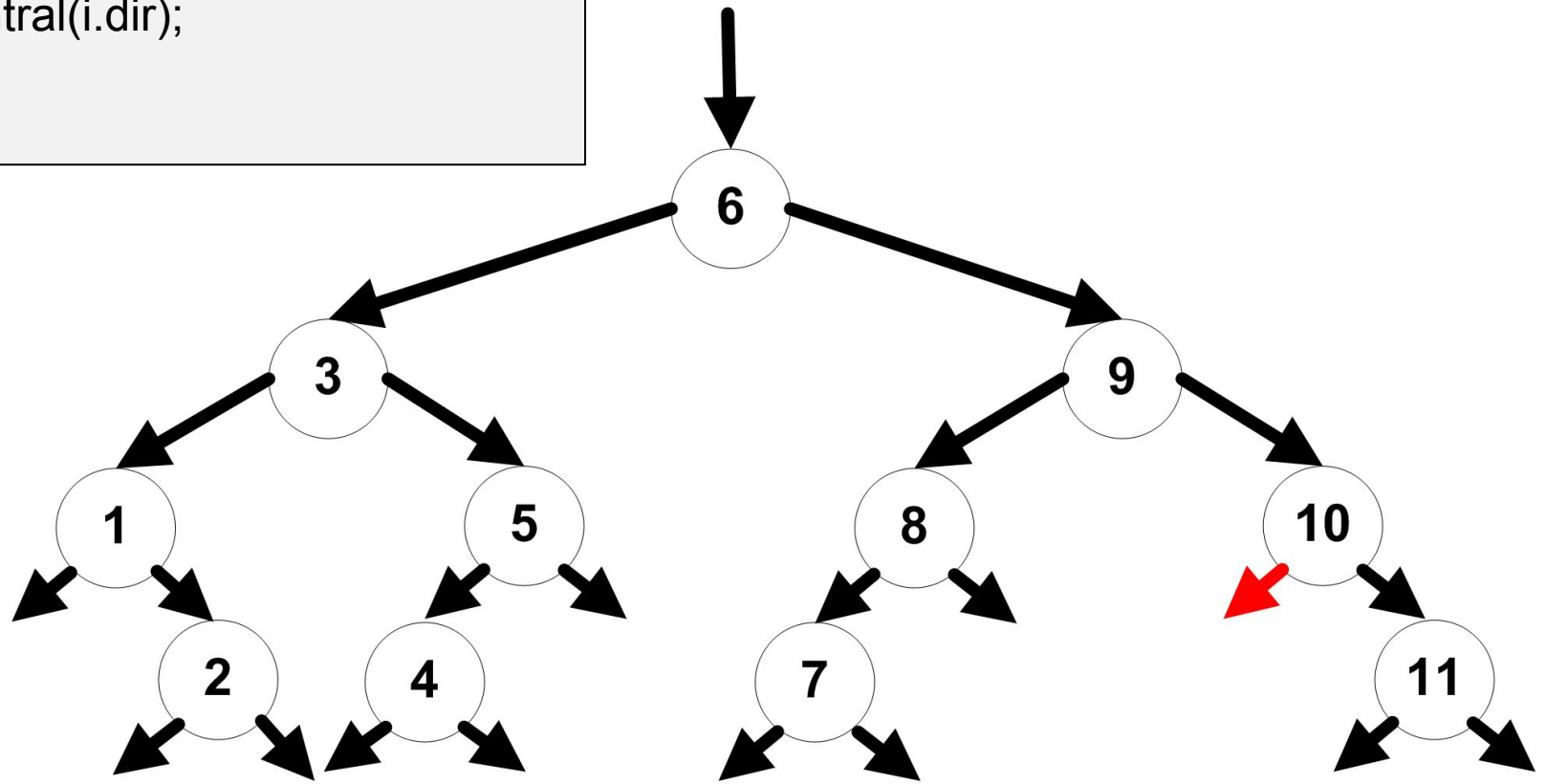
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8	9

Caminhamento Central ou Em Ordem

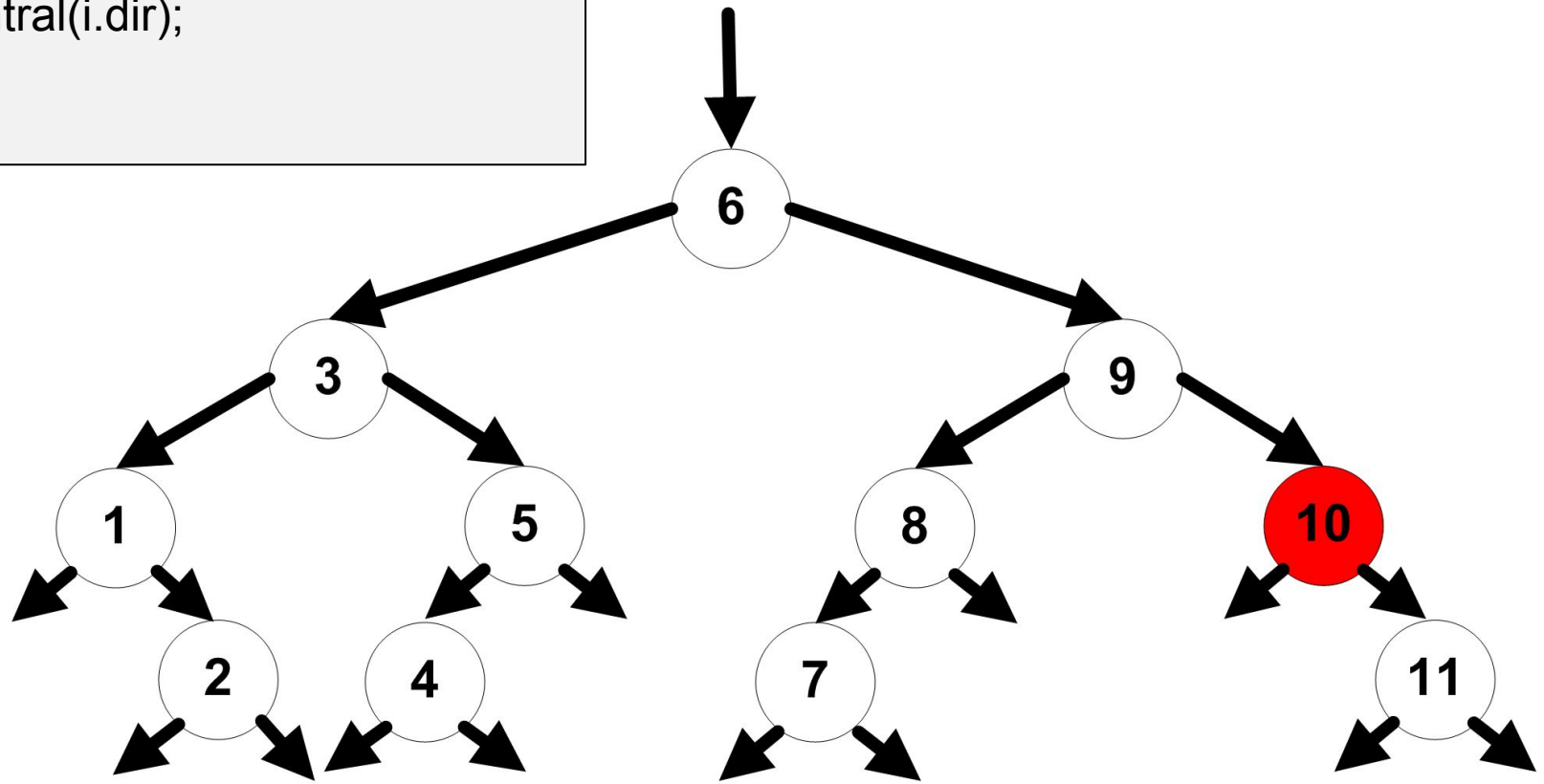
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8	9

Caminhamento Central ou Em Ordem

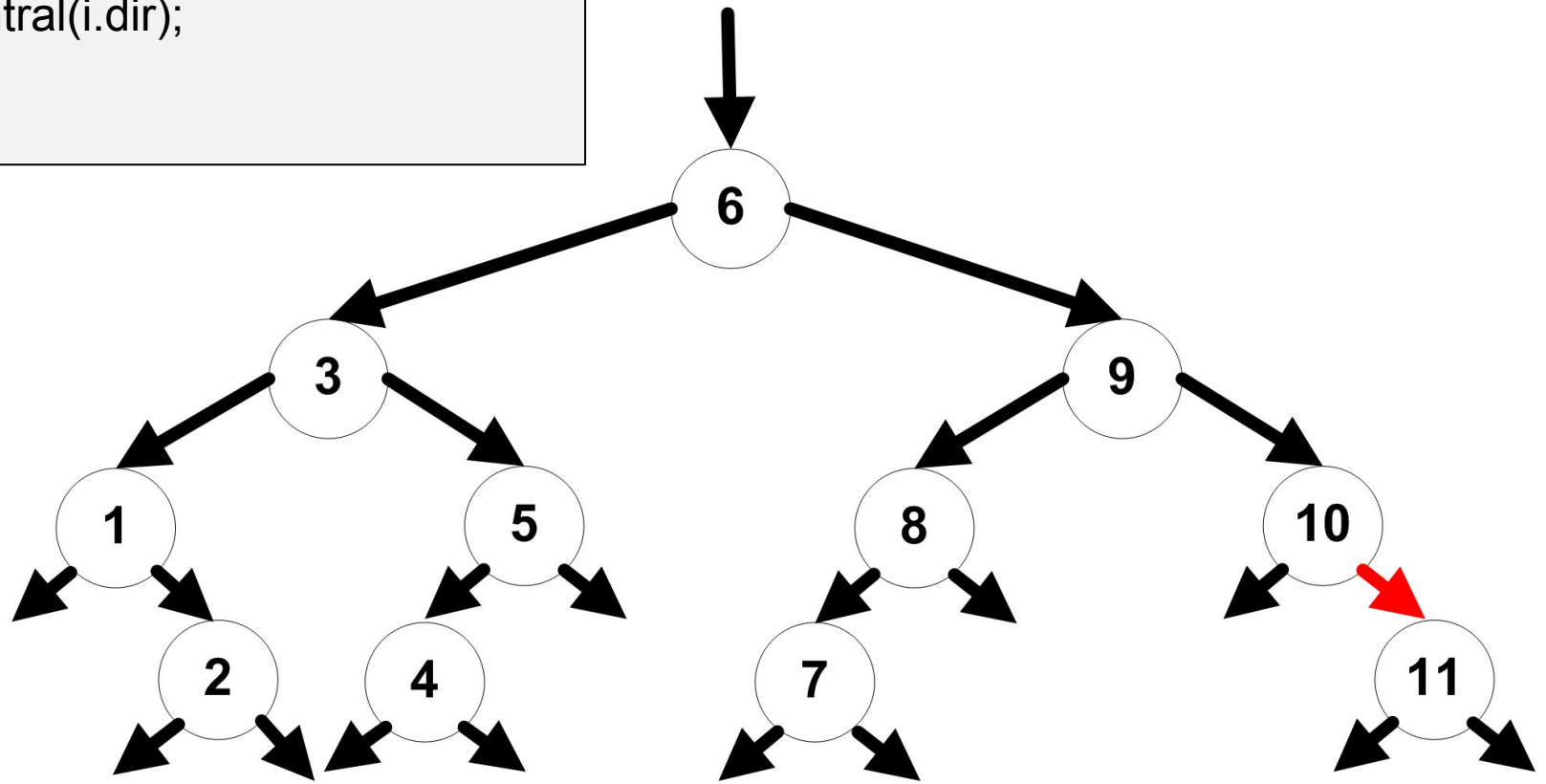
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8	9	10

Caminhamento Central ou Em Ordem

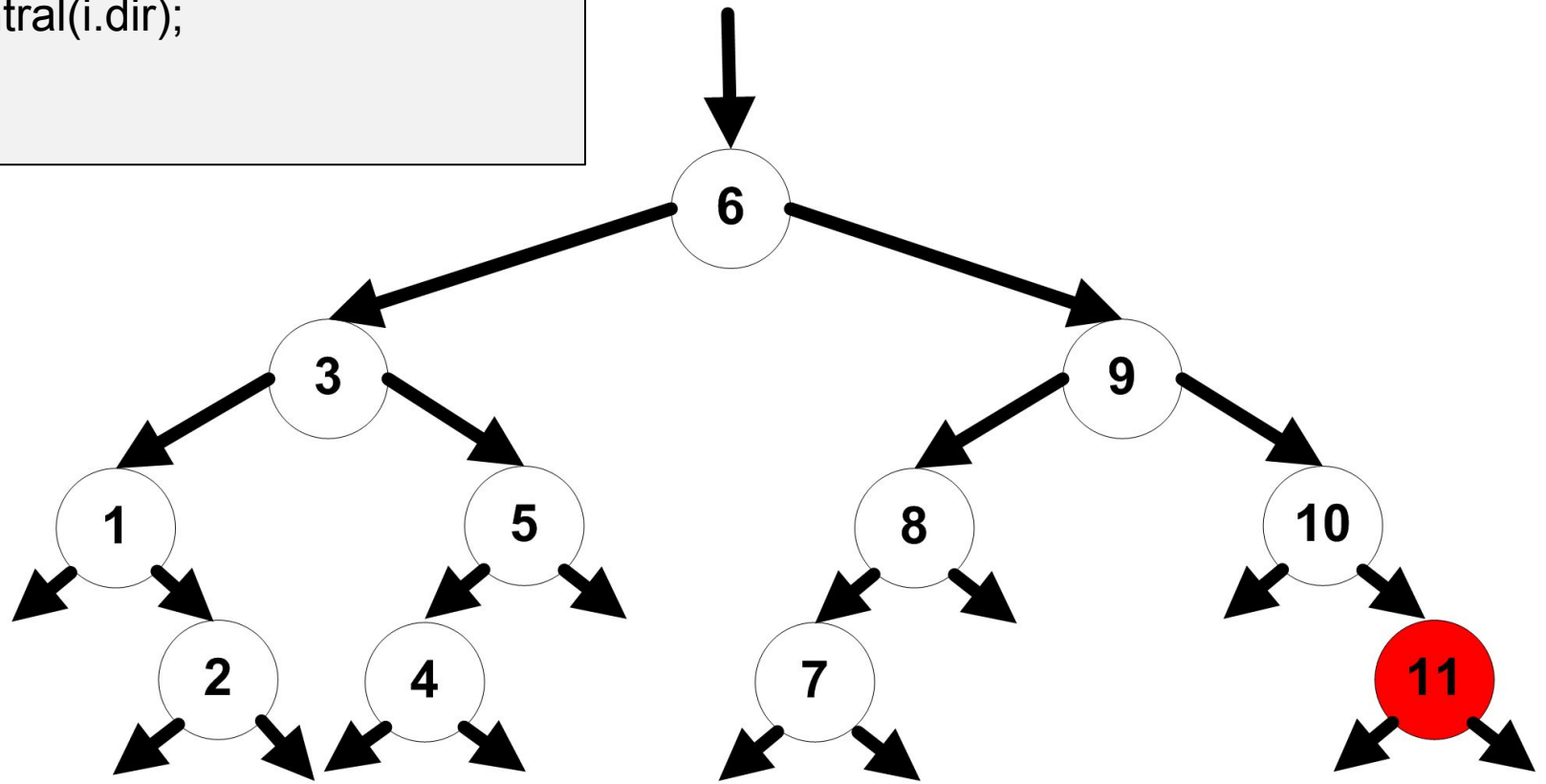
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8	9	10

Caminhamento Central ou Em Ordem

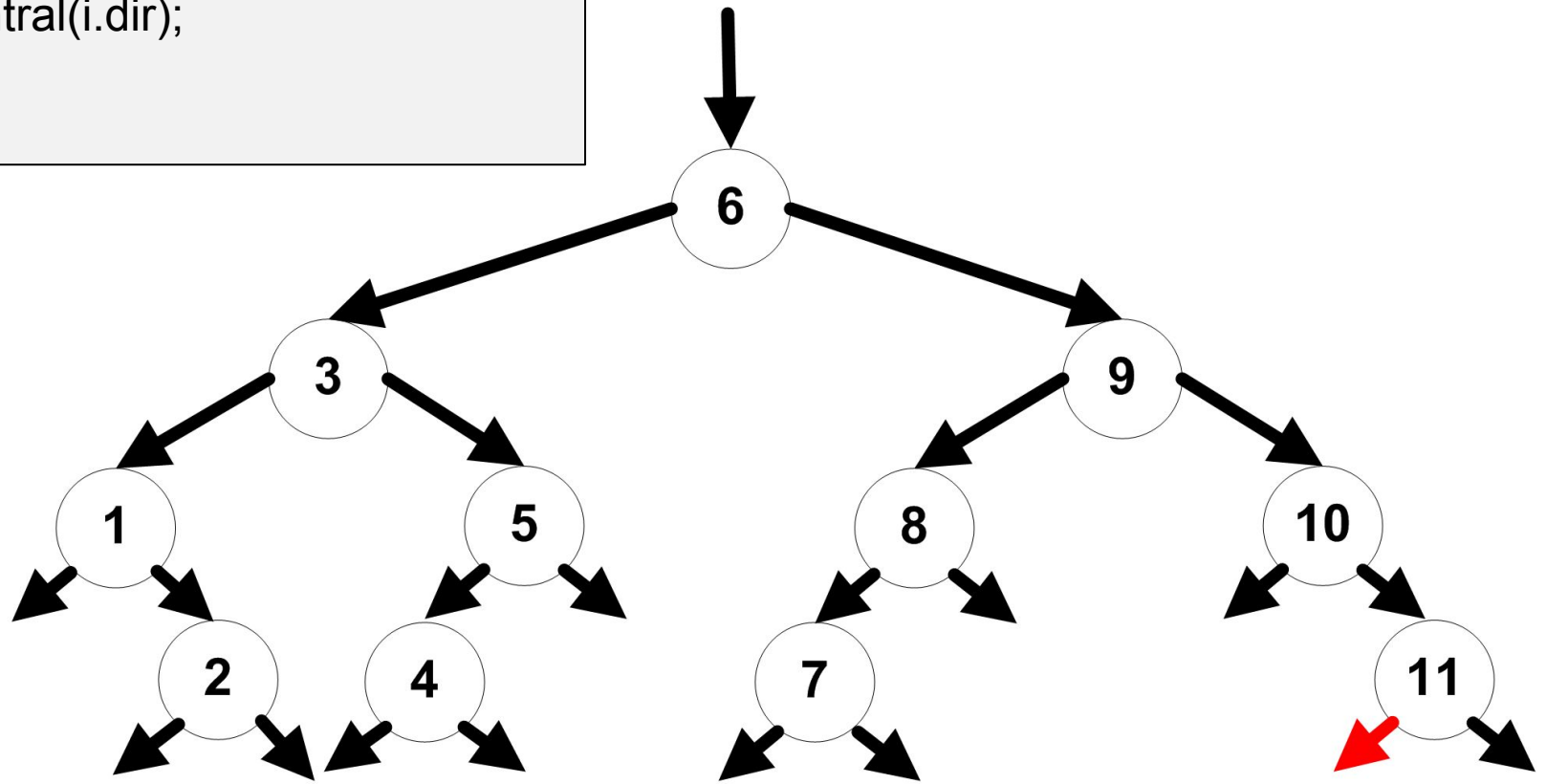
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8	9	10

Caminhamento Central ou Em Ordem

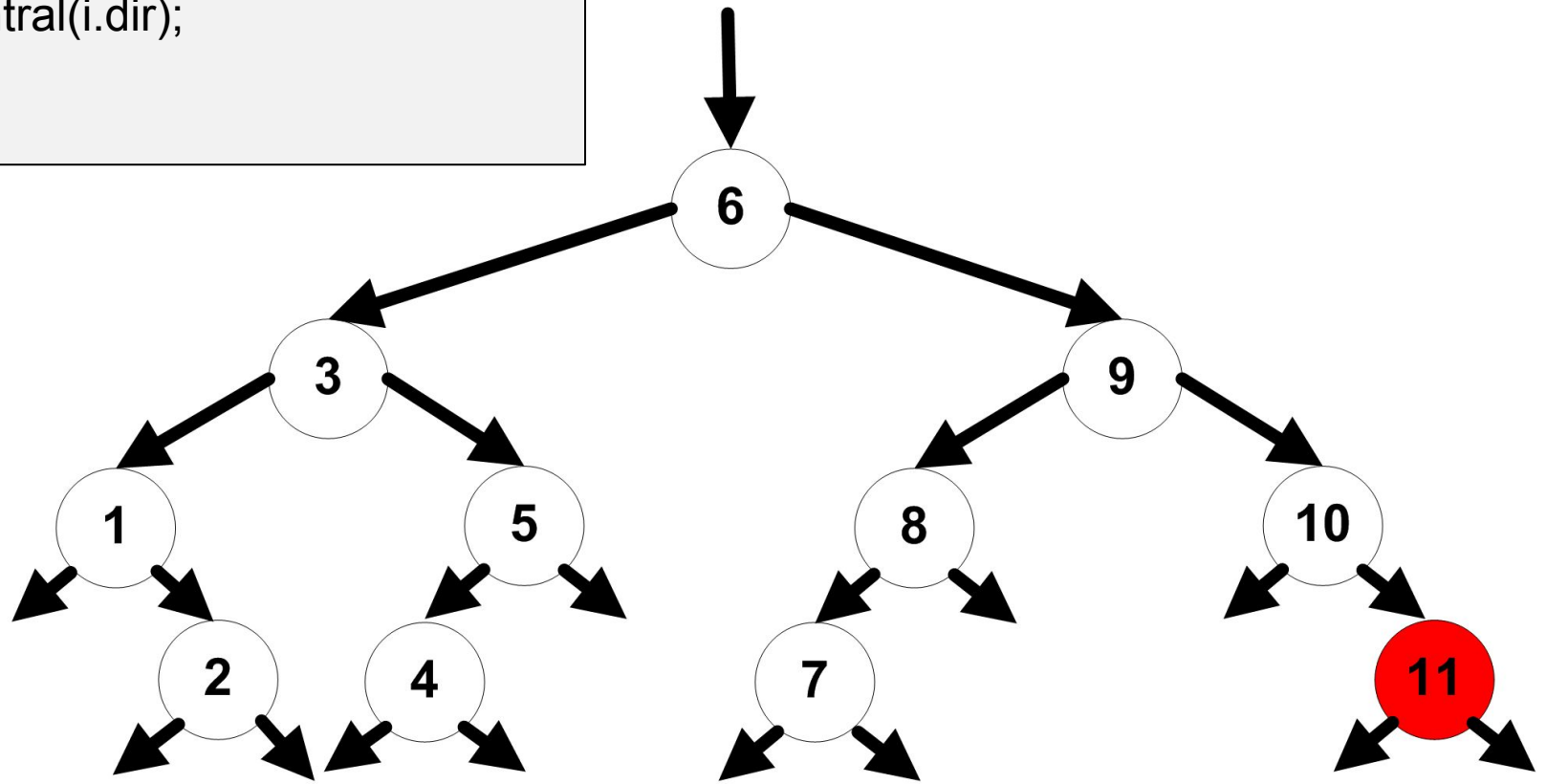
```
void caminharCentral(No i) {
   if (i != null) {
      caminharCentral(i.esq);
      System.out.print(i.elemento + " ");
      caminharCentral(i.dir);
   }
}
```



Tela	1	2	3	4	5	6	7	8	9	10

Caminhamento Central ou Em Ordem

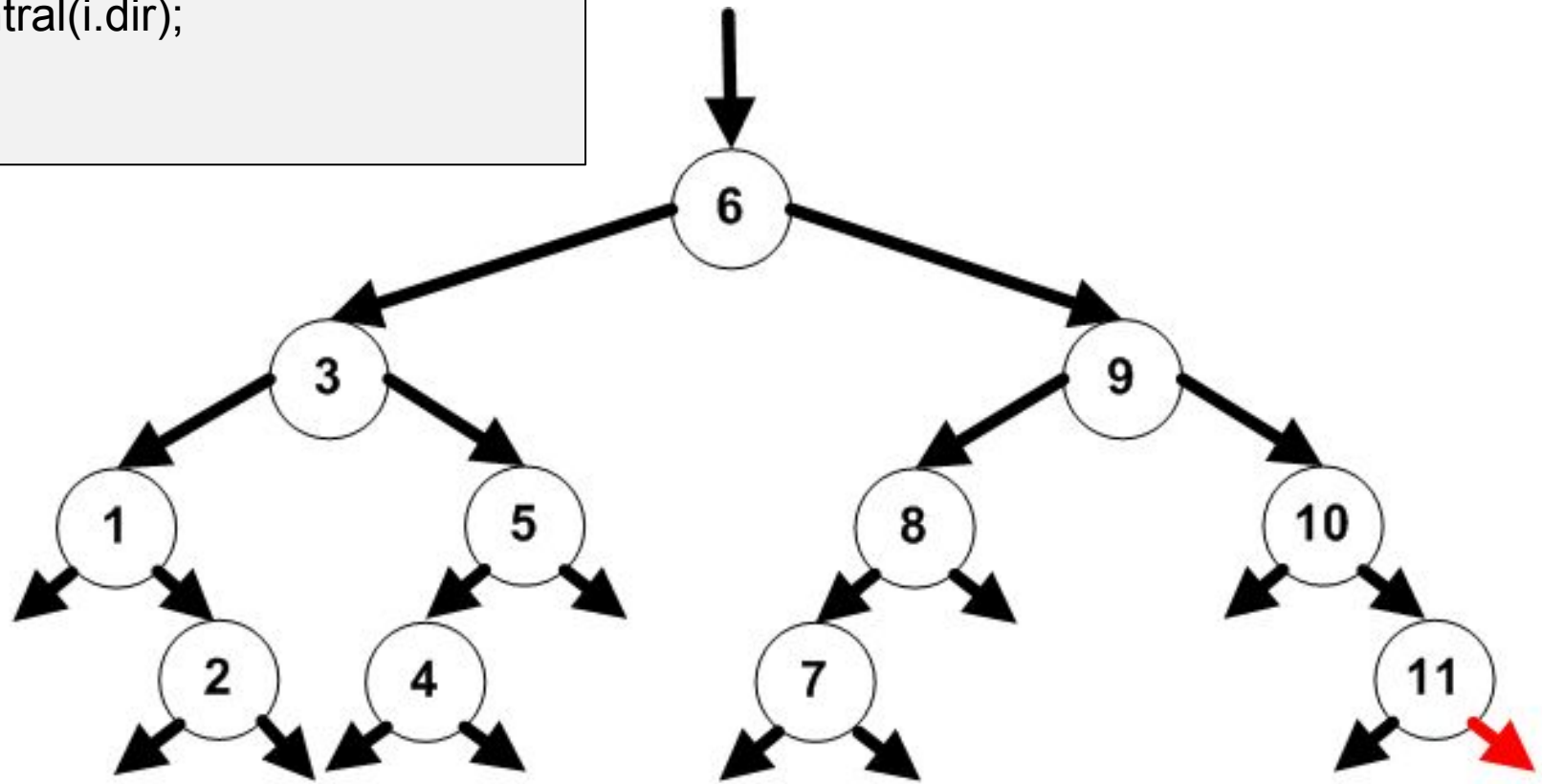
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



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

Caminhamento Central ou Em Ordem

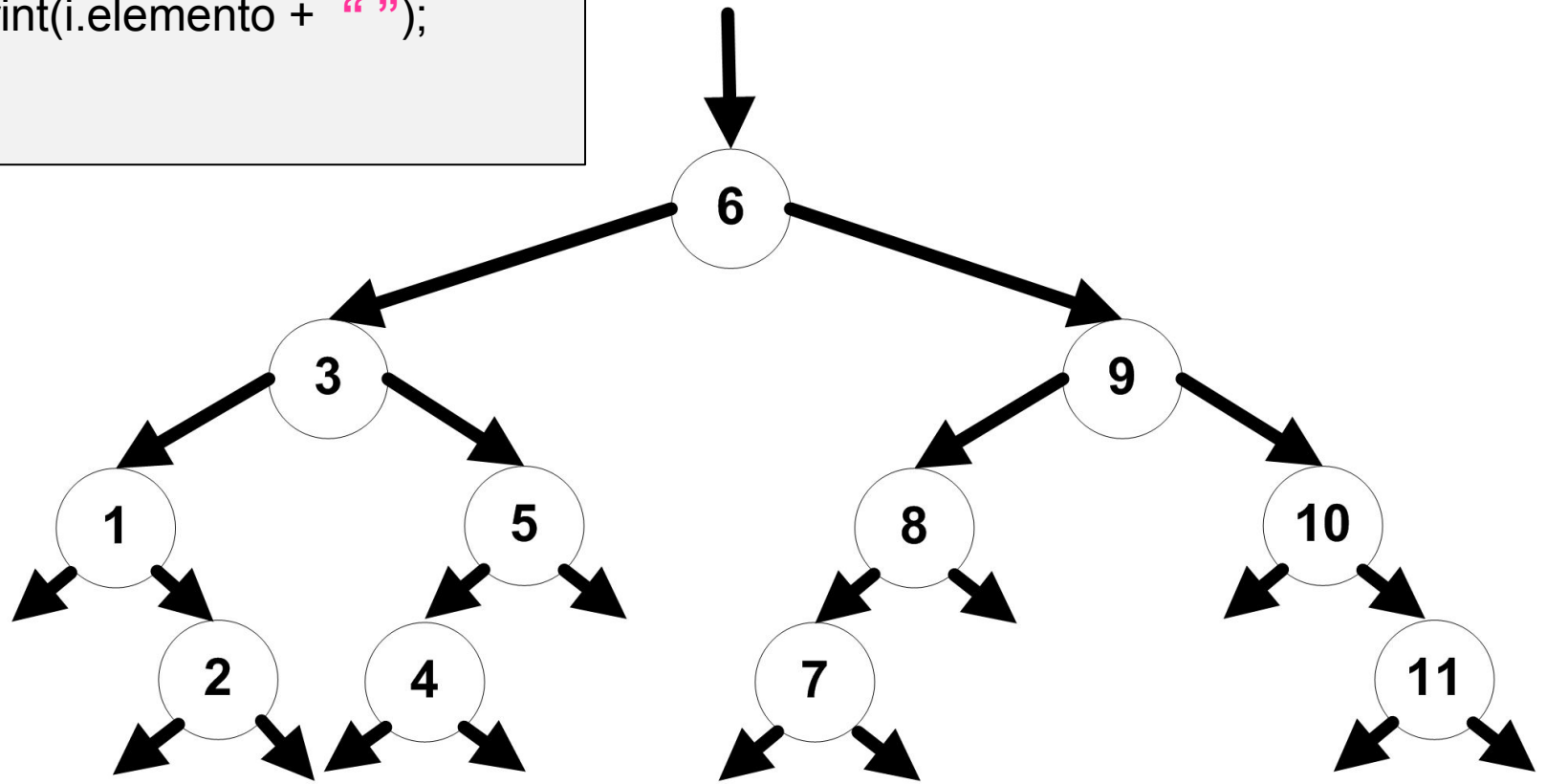
```
void caminharCentral(No i) {
  if (i != null) {
    caminharCentral(i.esq);
    System.out.print(i.elemento + " ");
    caminharCentral(i.dir);
  }
}
```



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

Caminhamento Pós-fixado ou Pós-ordem

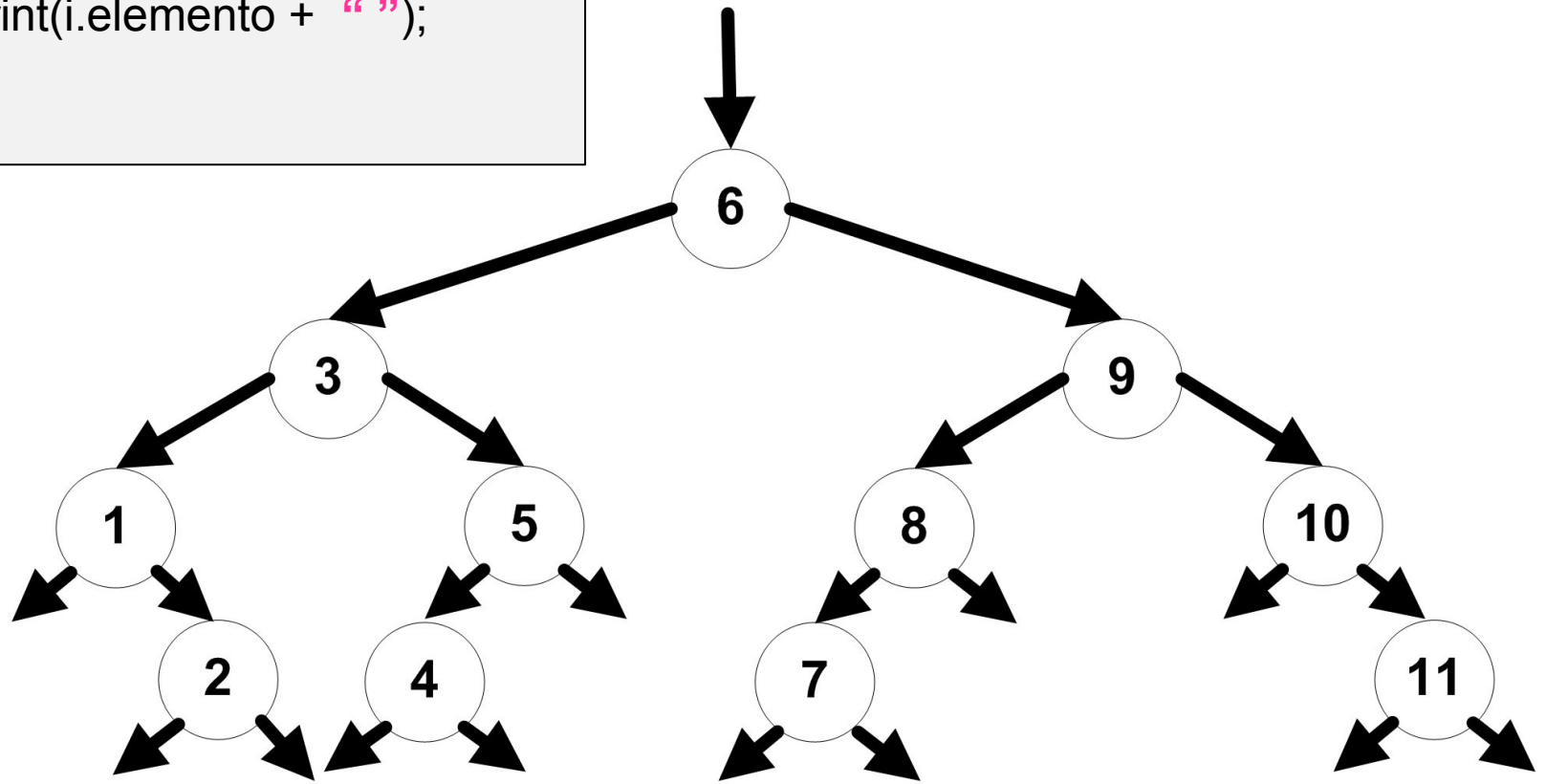
```
void caminharPos(No i) {
  if (i != null) {
    caminharPos(i.esq);
    caminharPos(i.dir);
    System.out.print(i.elemento + " ");
  }
}
```



Tela	

Caminhamento Pós-fixado ou Pós-ordem

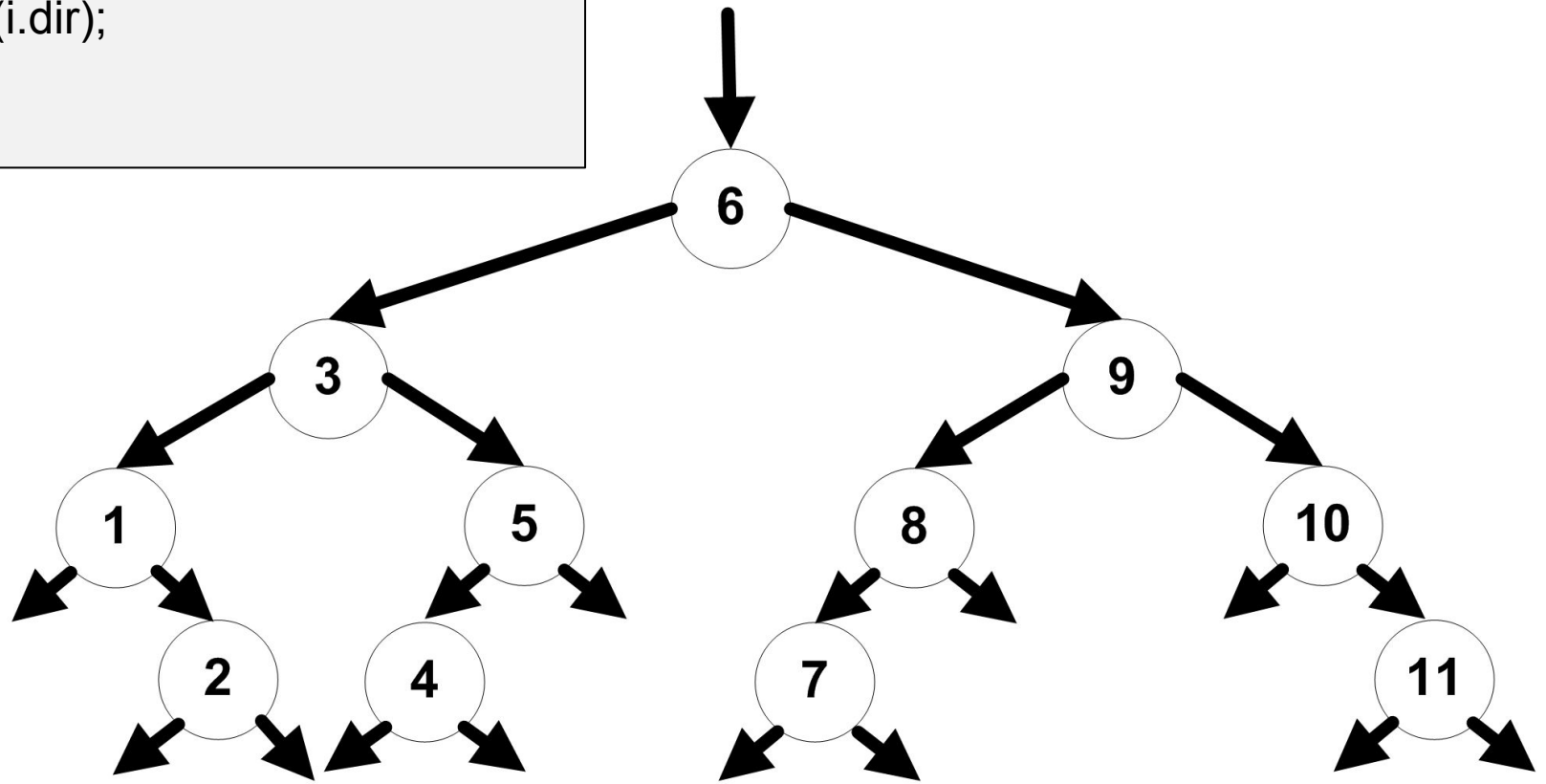
```
void caminharPos(No i) {
  if (i != null) {
    caminharPos(i.esq);
    caminharPos(i.dir);
    System.out.print(i.elemento + " ");
  }
}
```



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

Caminhamento Pré-fixado ou Pré-ordem

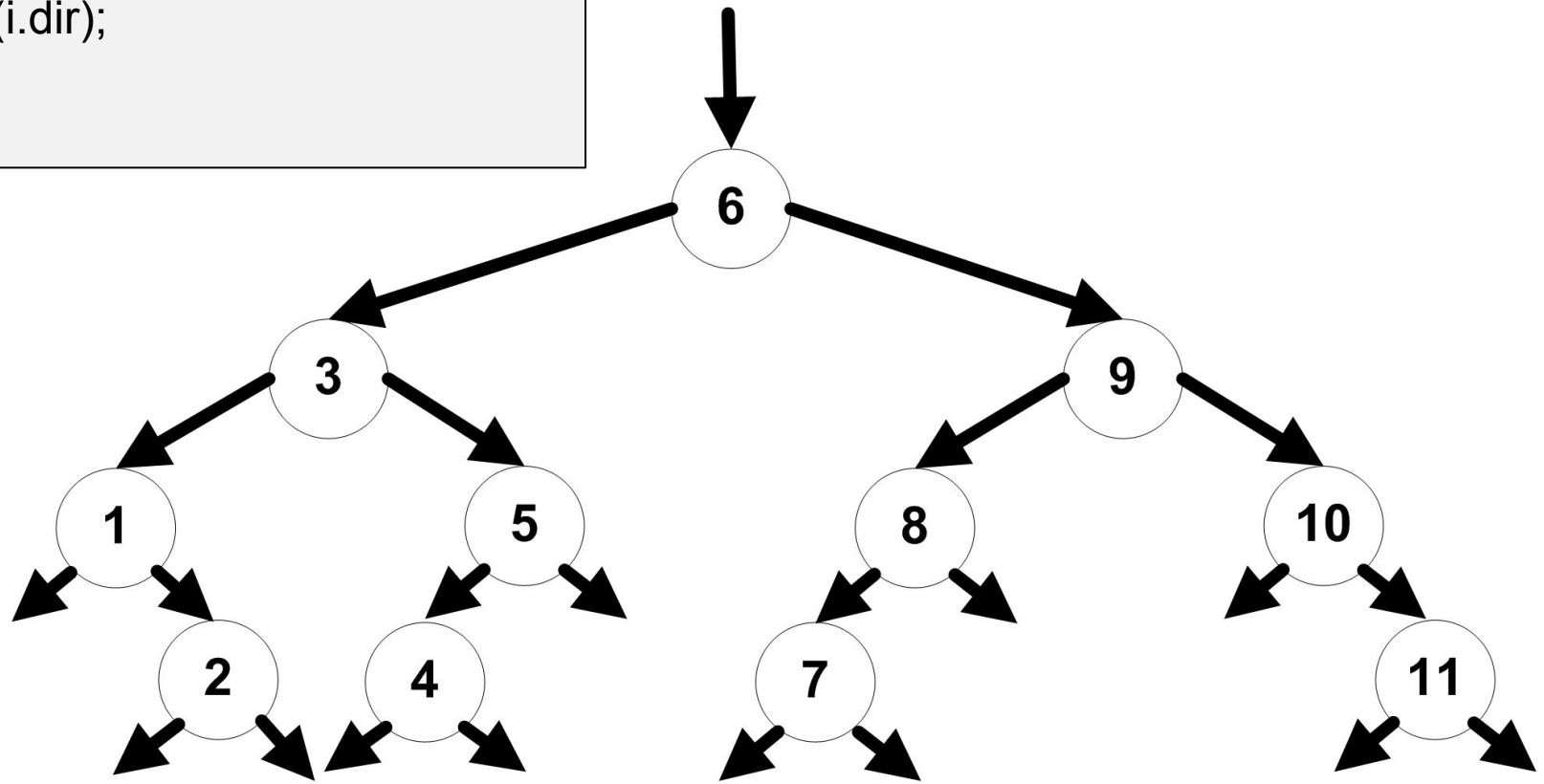
```
void caminharPre(No i) {
  if (i != null) {
    System.out.print(i.elemento + " ");
    caminharPre(i.esq);
    caminharPre(i.dir);
  }
}
```



Tela	

Caminhamento Pré-fixado ou Pré-ordem

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



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

Exercício (1)

- Faça um método que retorna a altura da árvore. Em seguida, insira vários elementos de forma aleatória. Para cada inserção, mostre na tela o número de elementos da árvore, o logaritmo (base 2) desse número e a altura

Exercício (2)

- Faça um método que retorne a soma dos elementos existentes na árvore

Exercício (3)

- Faça um método que retorne o número de elementos pares existentes na árvore.

Exercício (4)

- Faça um método estático que recebe dois objetos do tipo árvore binária e retorne um booleano indicando se as duas árvores são iguais.

Exercício (5)

- Faça um método que retorna ***true*** se a árvore contém algum número divisível por onze.

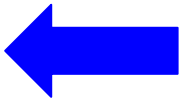
Exercício (6)

- Um algoritmo de ordenação é o *TreeSort* que insere os elementos do *array* em uma árvore binária e utiliza um "mostrar" para ordenar os elementos do *array*. Implemente o *TreeSort* e faça a análise de complexidade do mesmo.

Exercício (7)

- Faça o método ***No toArvoreBinaria(Celula primeiro, CelulaDupla primeiro)*** que recebe o nó cabeça de uma lista simples e o de outra dupla. Em seguida, crie uma árvore binária contendo os elementos intercalados das duas listas e retorne o endereço do nó raiz da árvore criada.

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Caminhamento
- **Remoção** 
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

  - **Funcionamento básico**
  - Algoritmo em Java
  - Análise de Complexidade

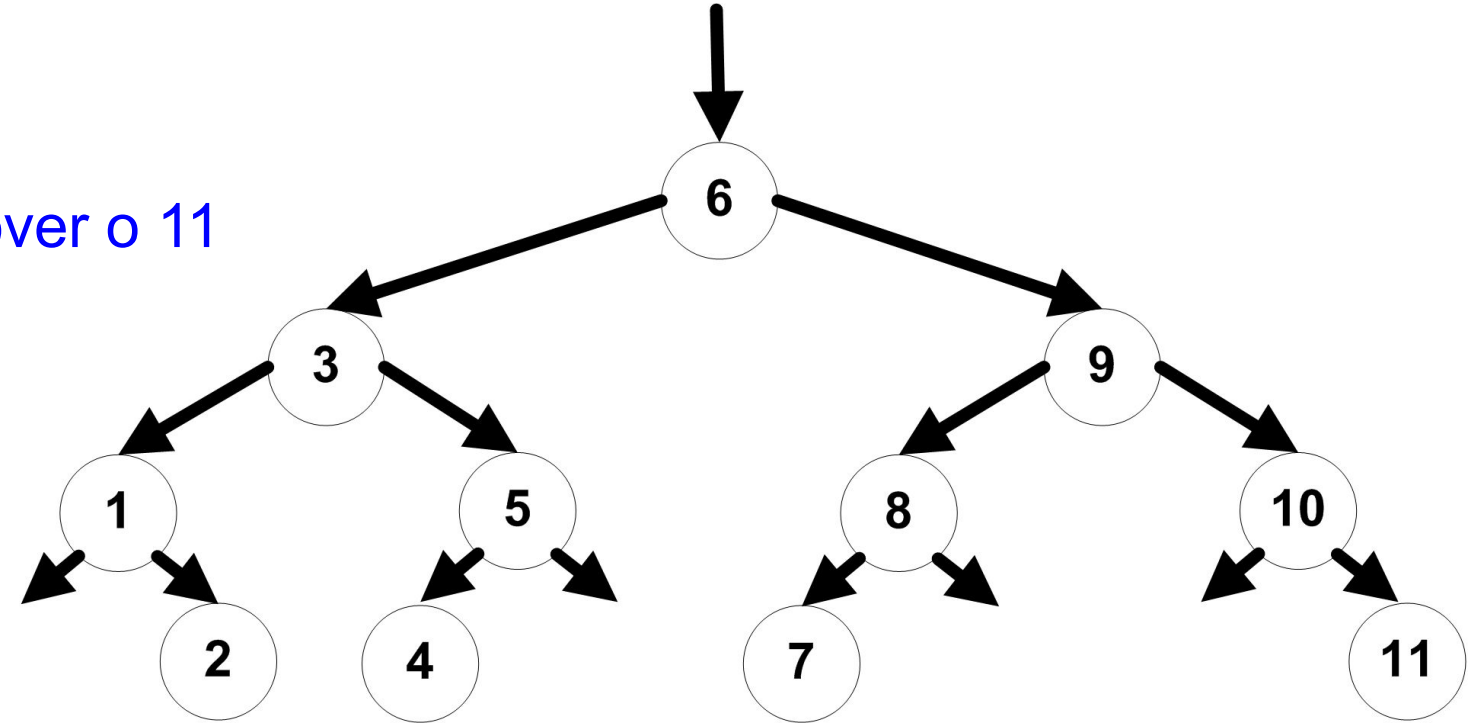
Funcionamento Básico da Remoção

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

Funcionamento Básico da Remoção

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

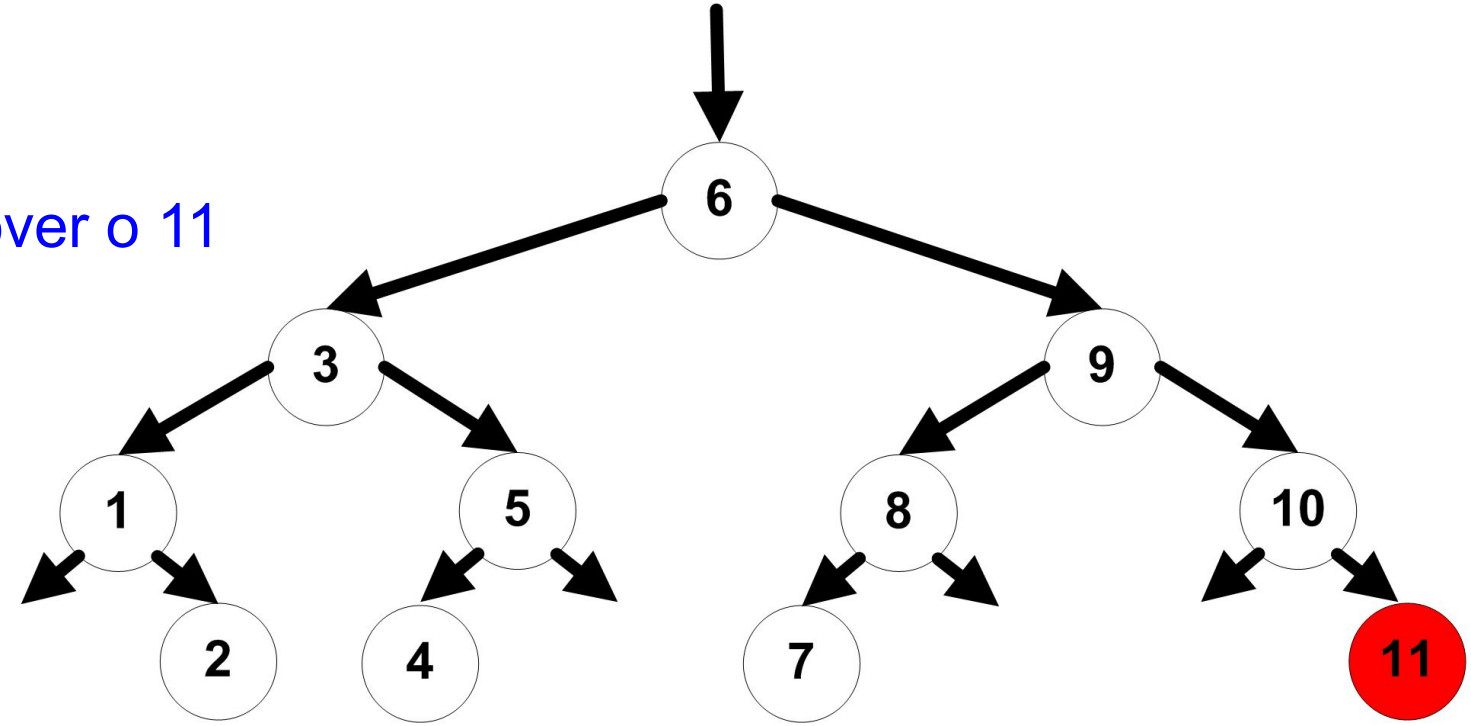
Exemplo: Remover o 11



Funcionamento Básico da Remoção

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

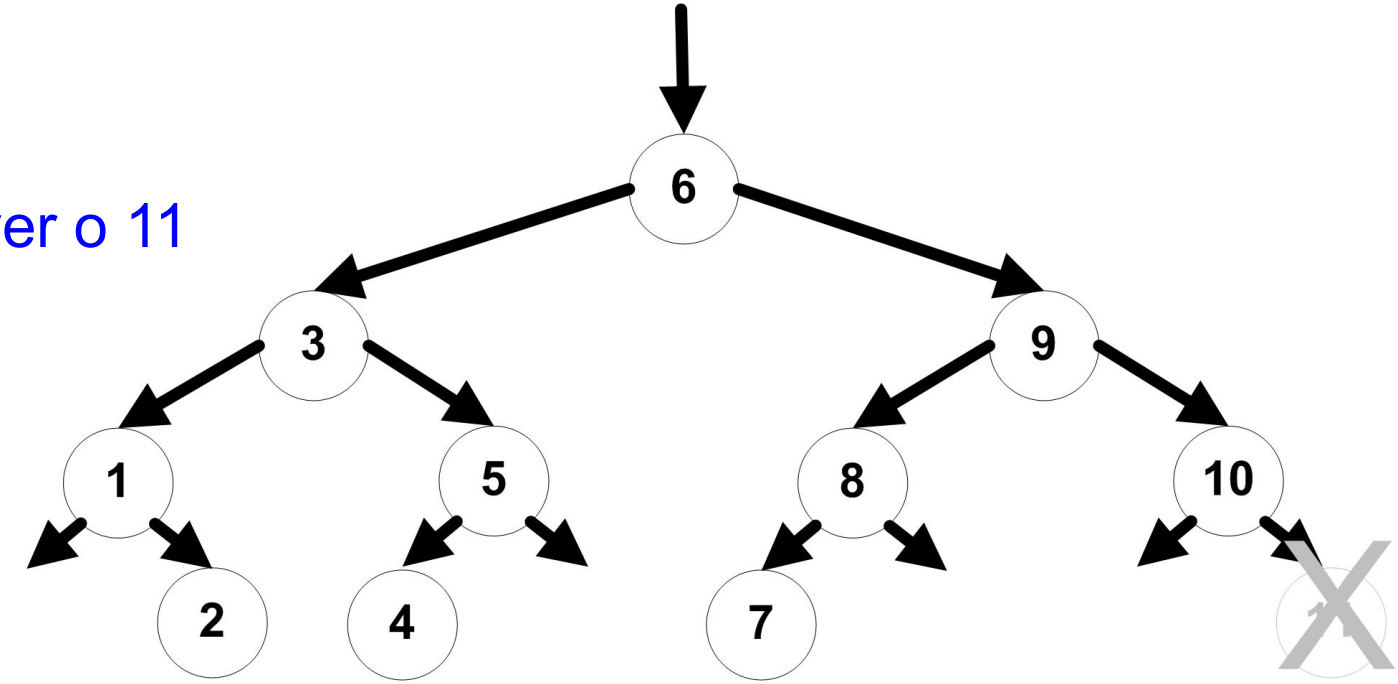
Exemplo: Remover o 11



Funcionamento Básico da Remoção

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

Exemplo: Remover o 11



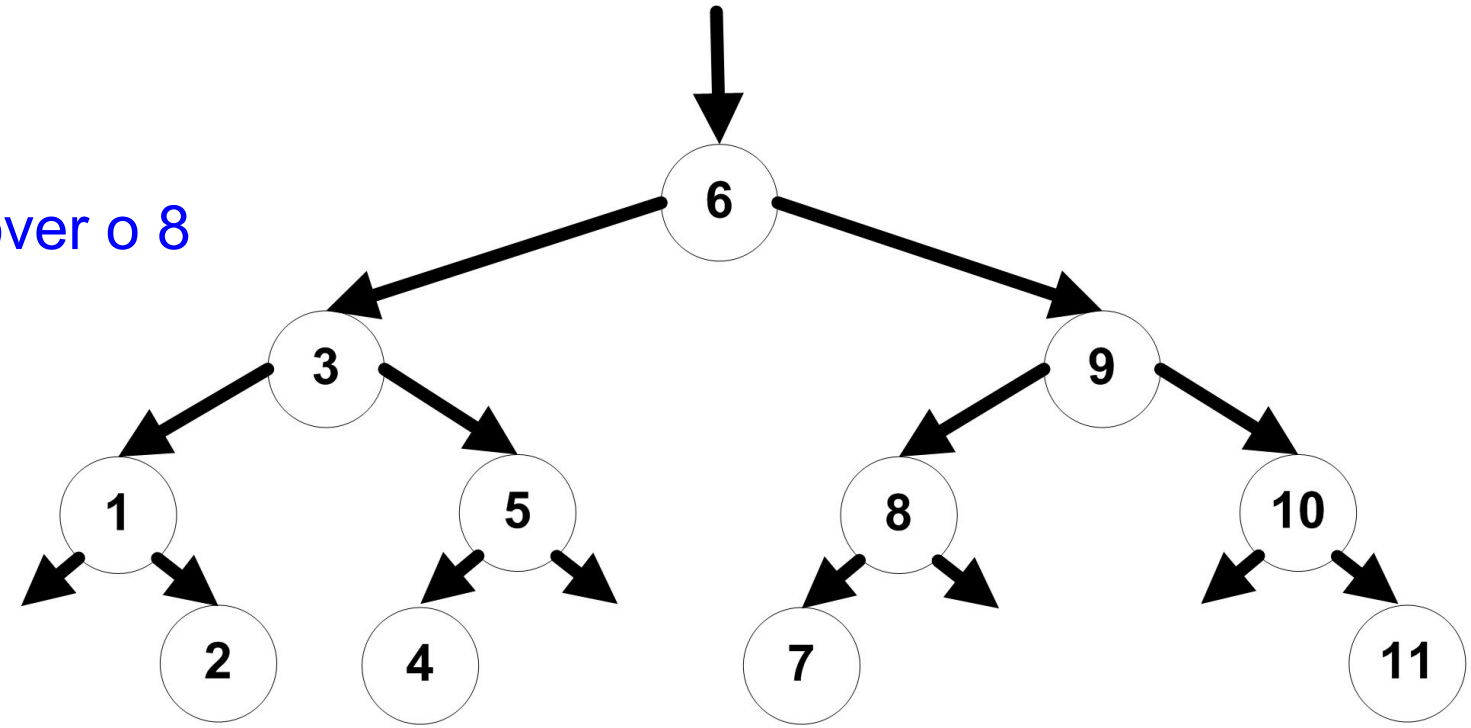
Funcionamento Básico da Remoção

(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

Funcionamento Básico da Remoção

(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

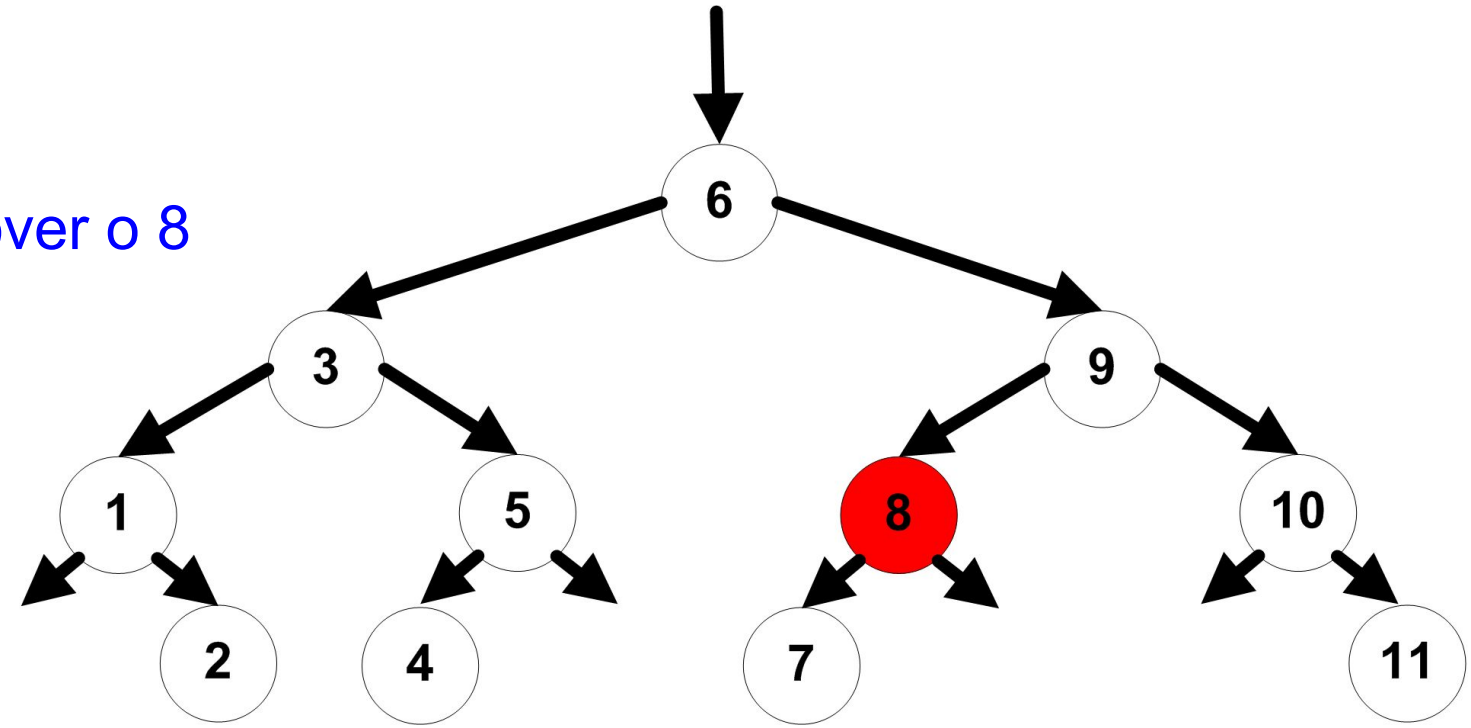
Exemplo: Remover o 8



Funcionamento Básico da Remoção

(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

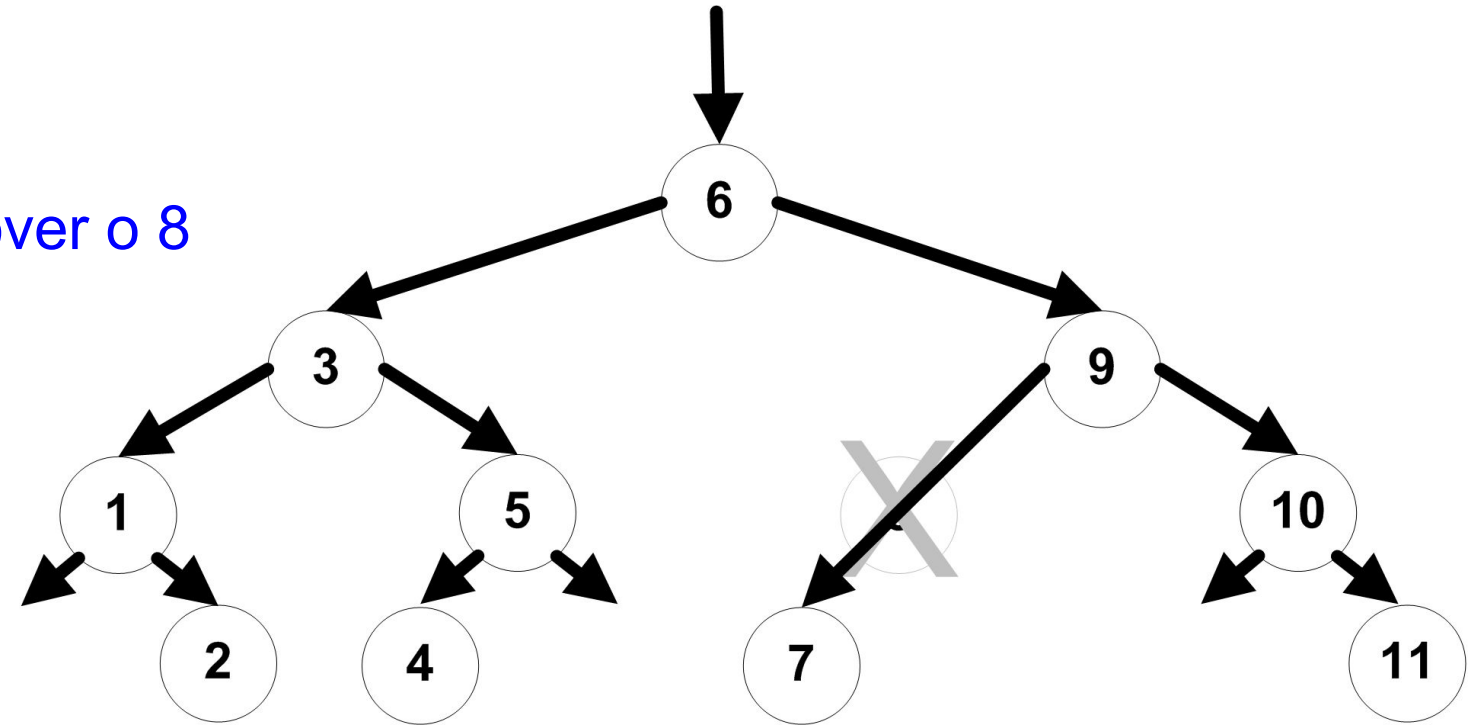
Exemplo: Remover o 8



Funcionamento Básico da Remoção

(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

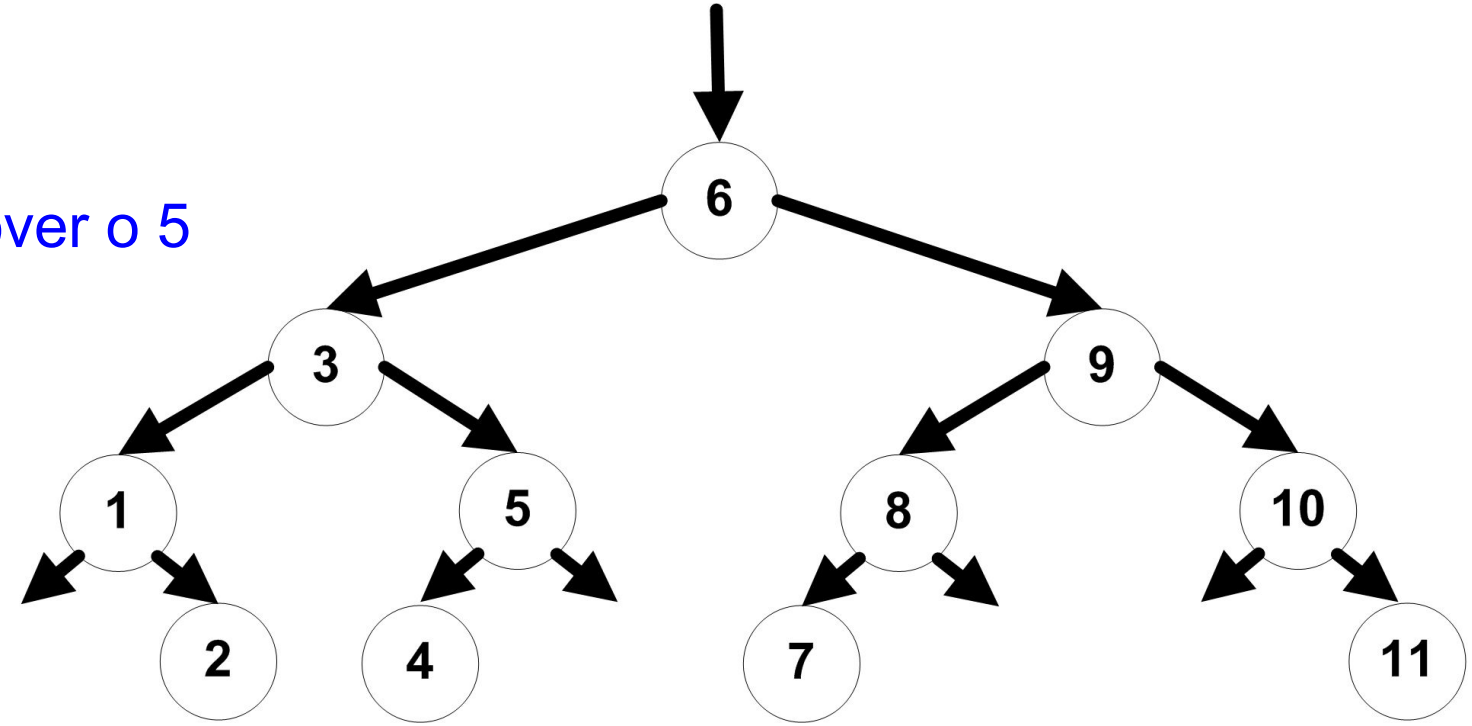
Exemplo: Remover o 8



Funcionamento Básico da Remoção

(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

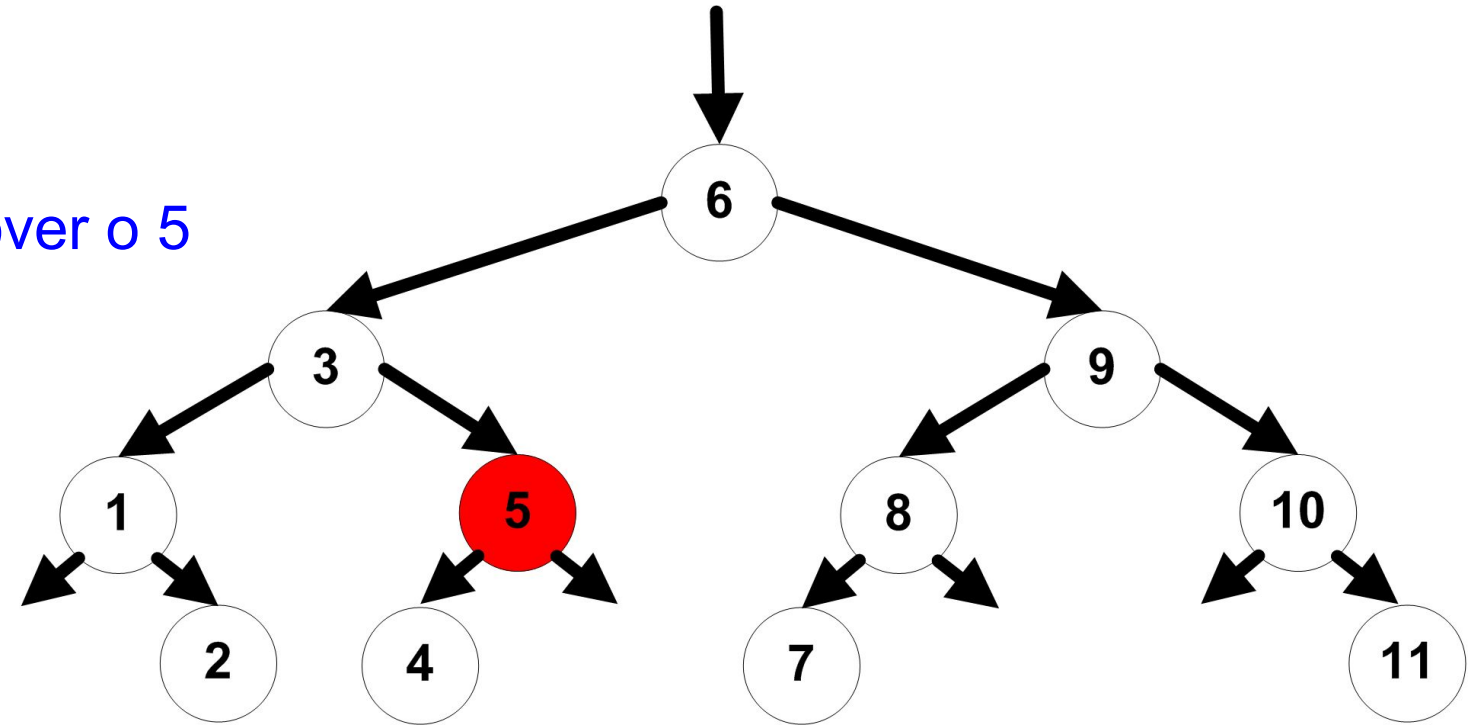
Exemplo: Remover o 5



Funcionamento Básico da Remoção

(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

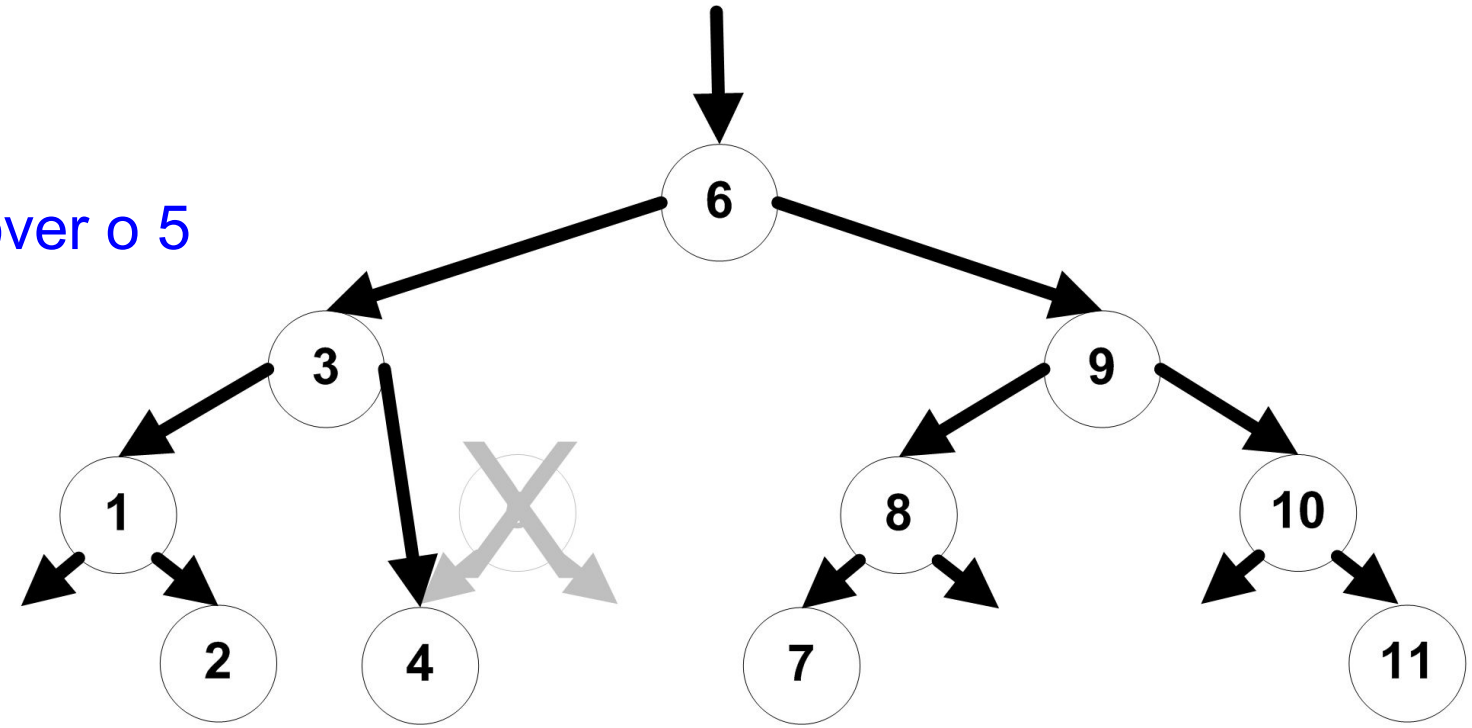
Exemplo: Remover o 5



Funcionamento Básico da Remoção

(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



Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

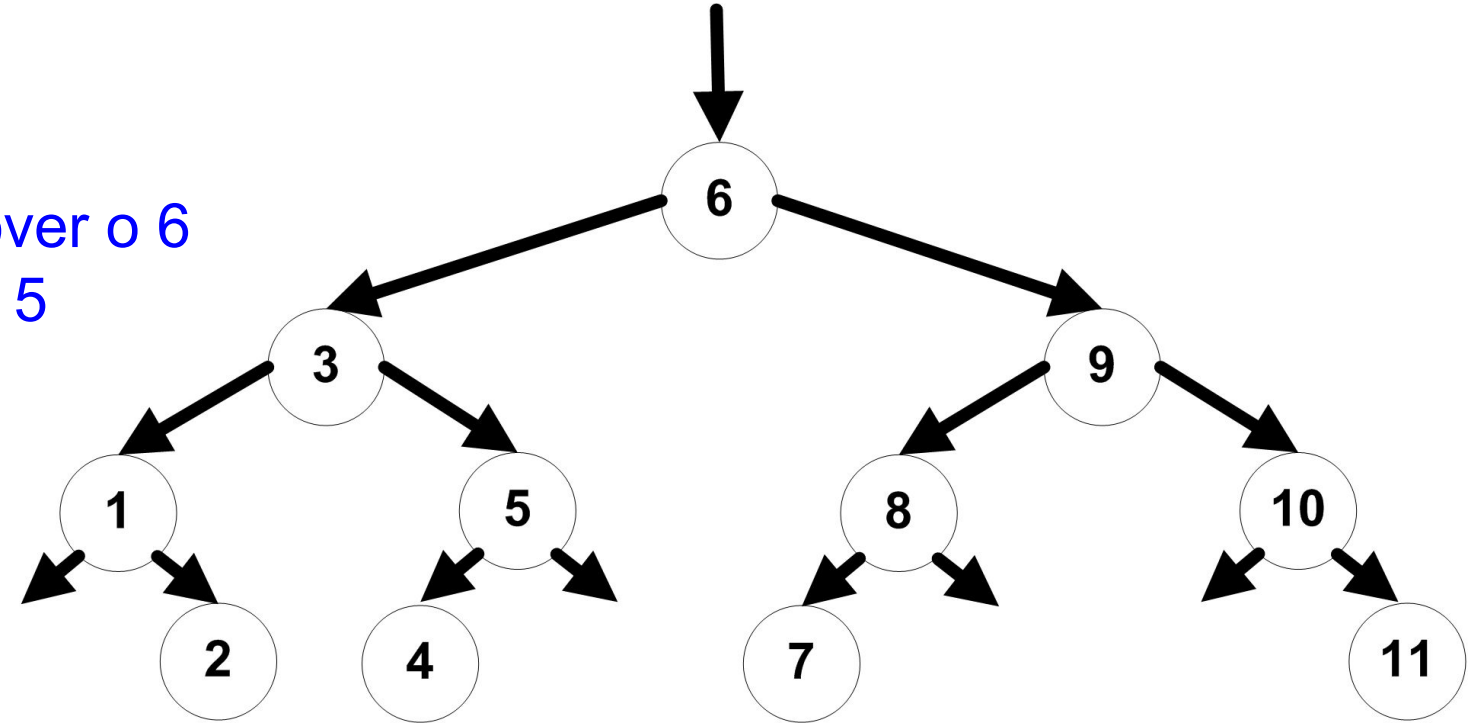
Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

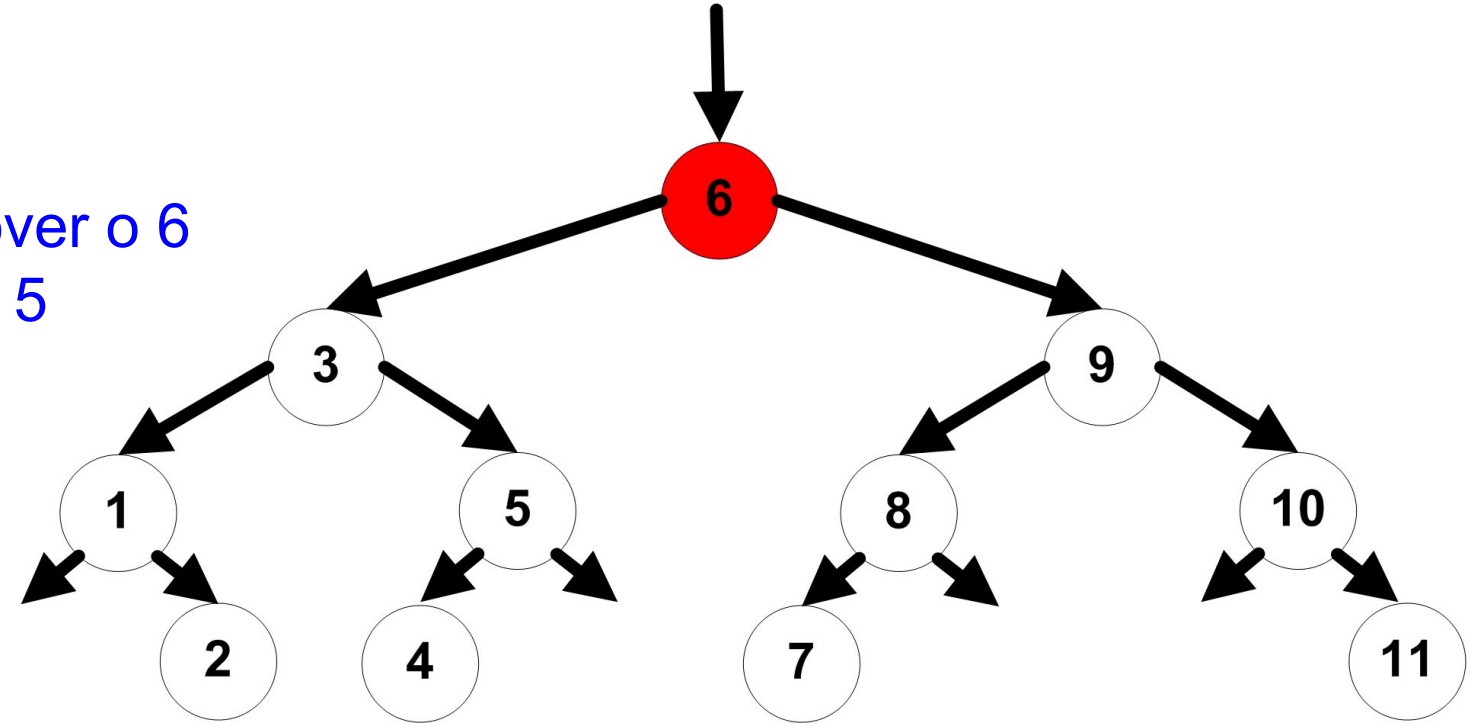
Exemplo: Remover o 6 e substituir pelo 5



Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

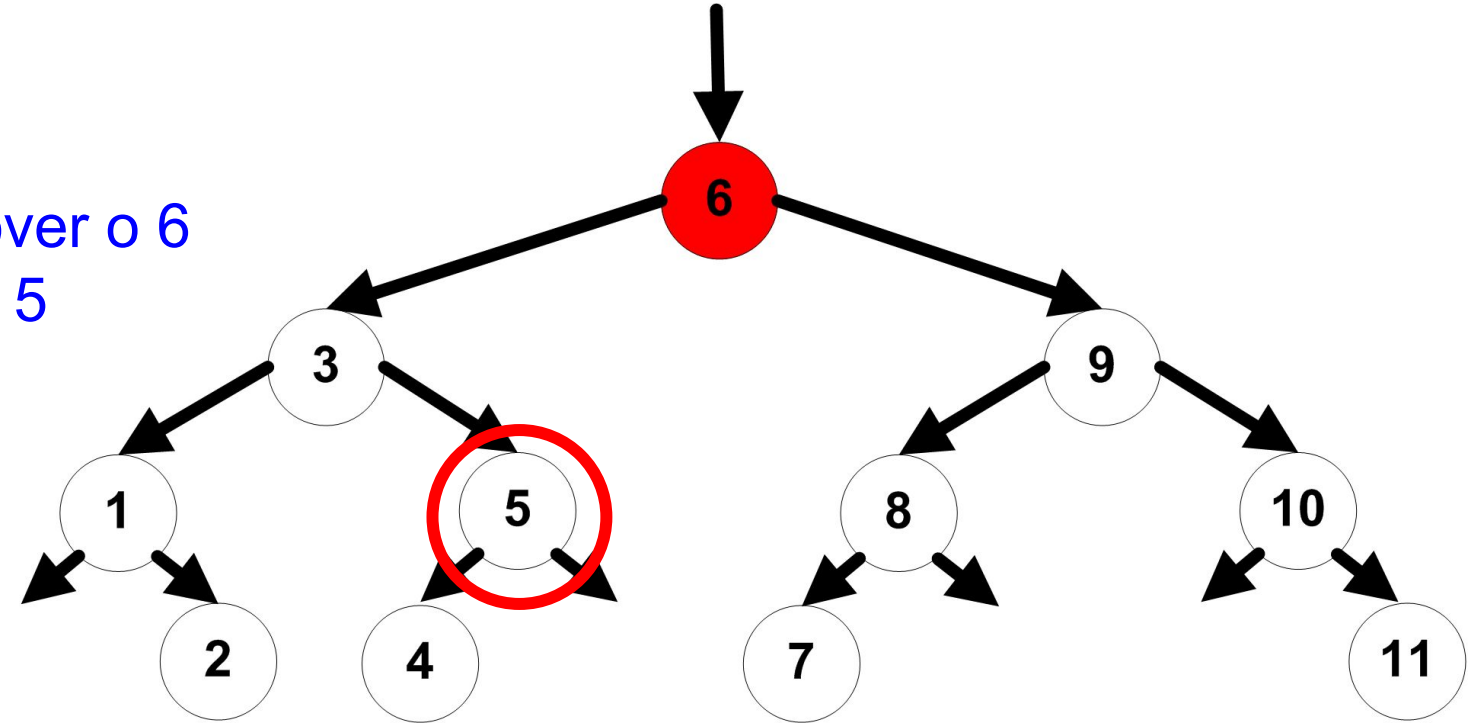
Exemplo: Remover o 6 e substituir pelo 5



Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

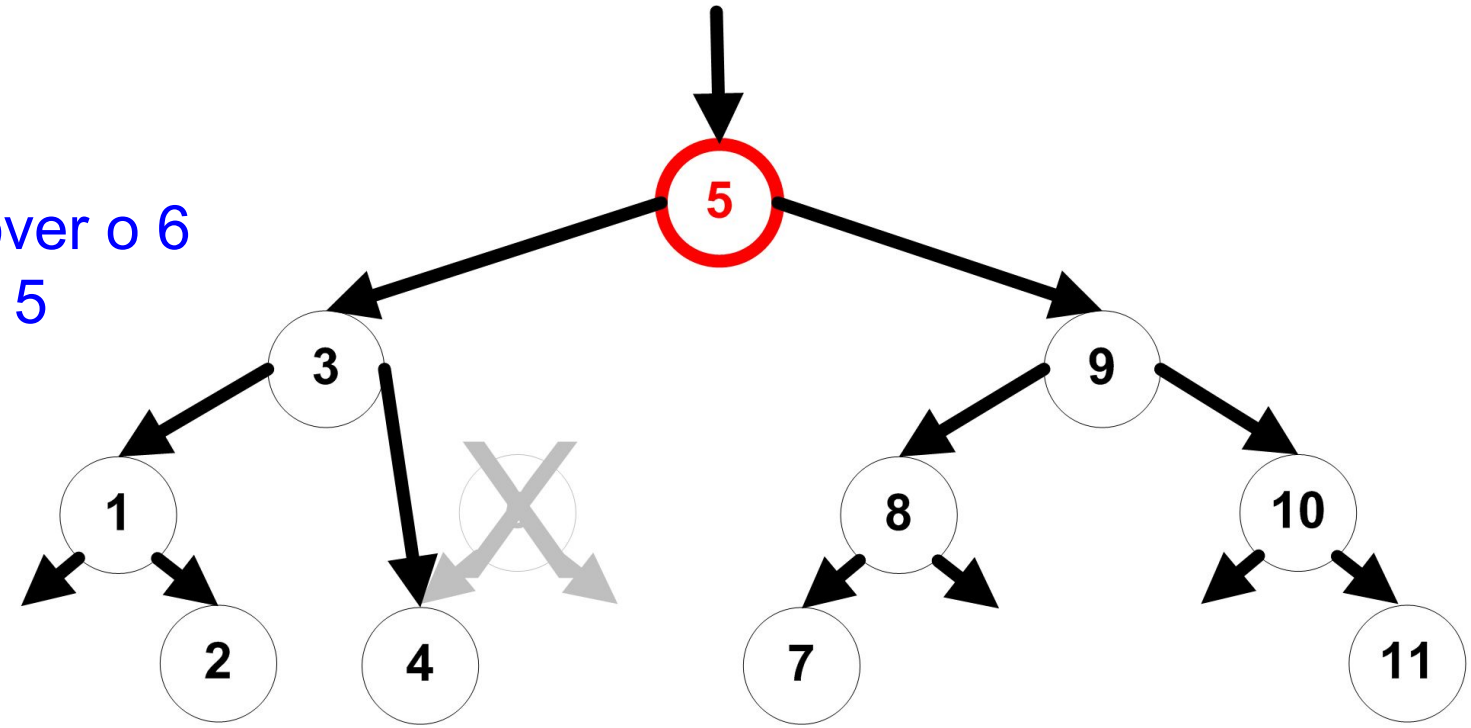
Exemplo: Remover o 6 e substituir pelo 5



Funcionamento Básico da Remoção

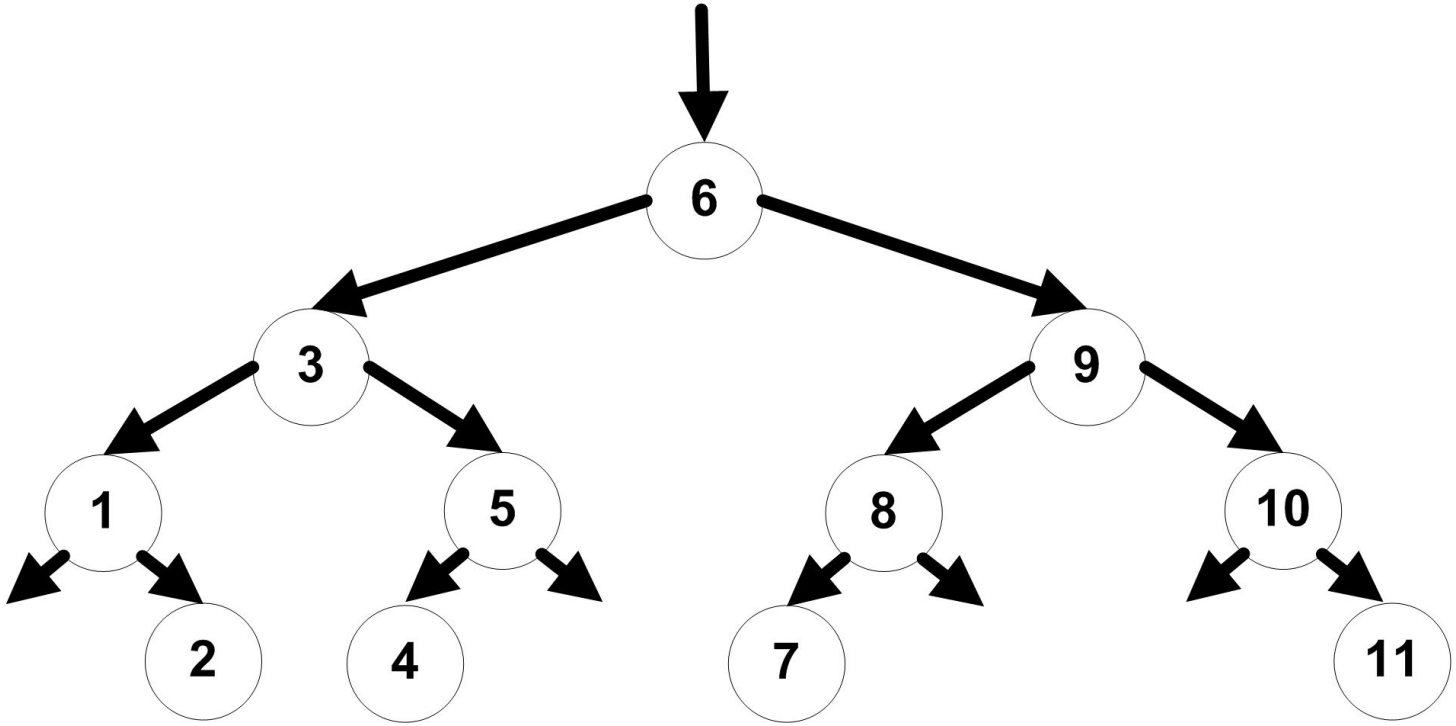
(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou **menor nó da subárvore à direita**

Exemplo: Remover o 6 e substituir pelo 5



Funcionamento Básico da Remoção

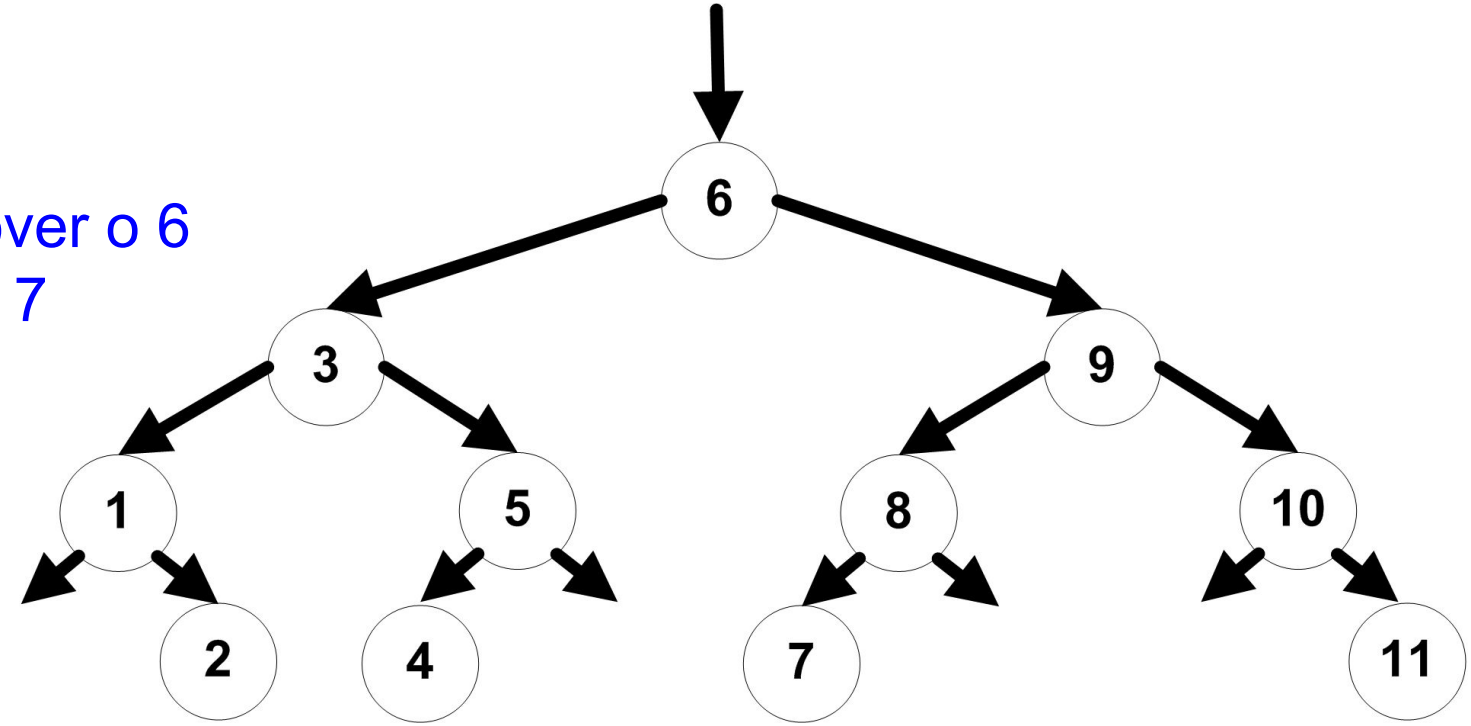
(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou **menor nó da subárvore à direita**



Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

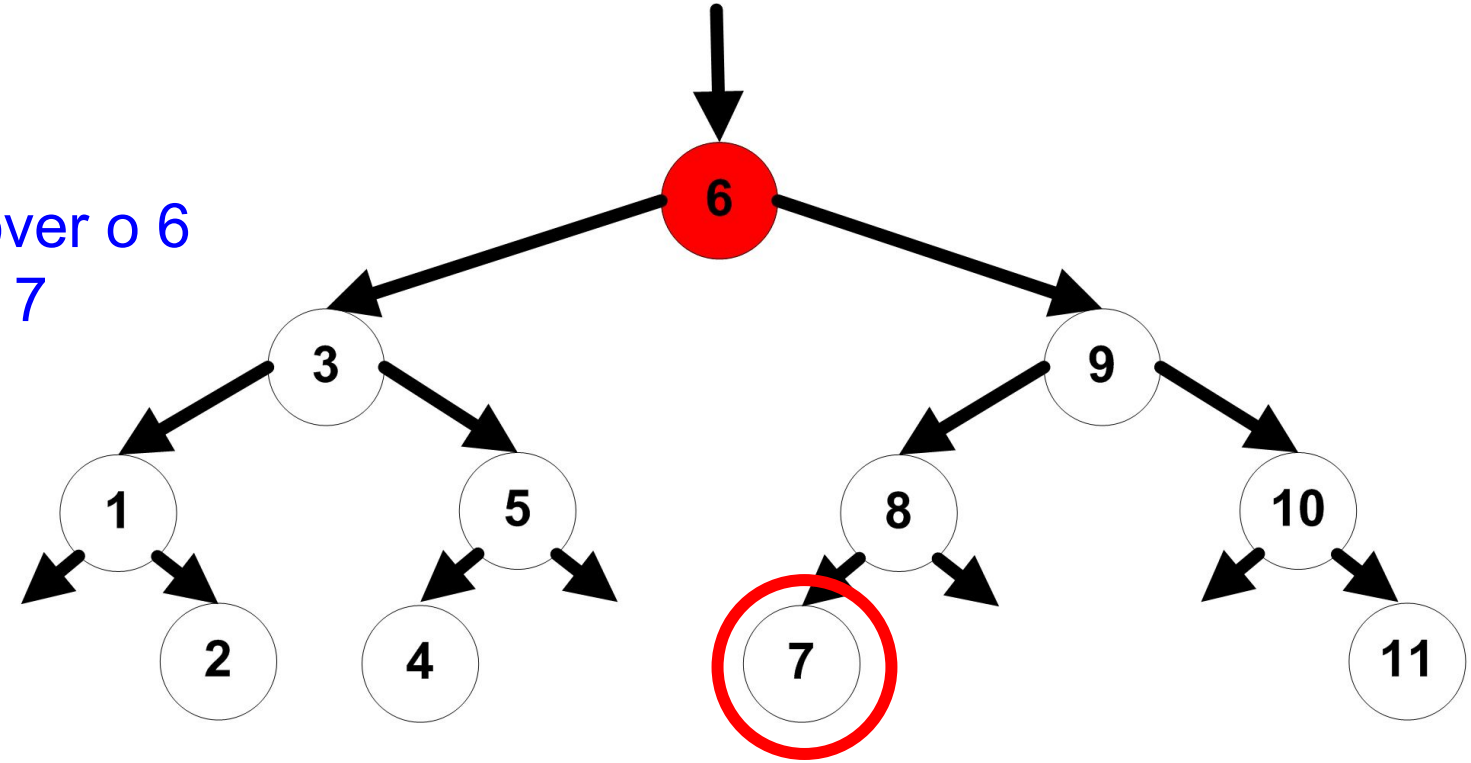
Exemplo: Remover o 6 e substituir pelo 7



Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

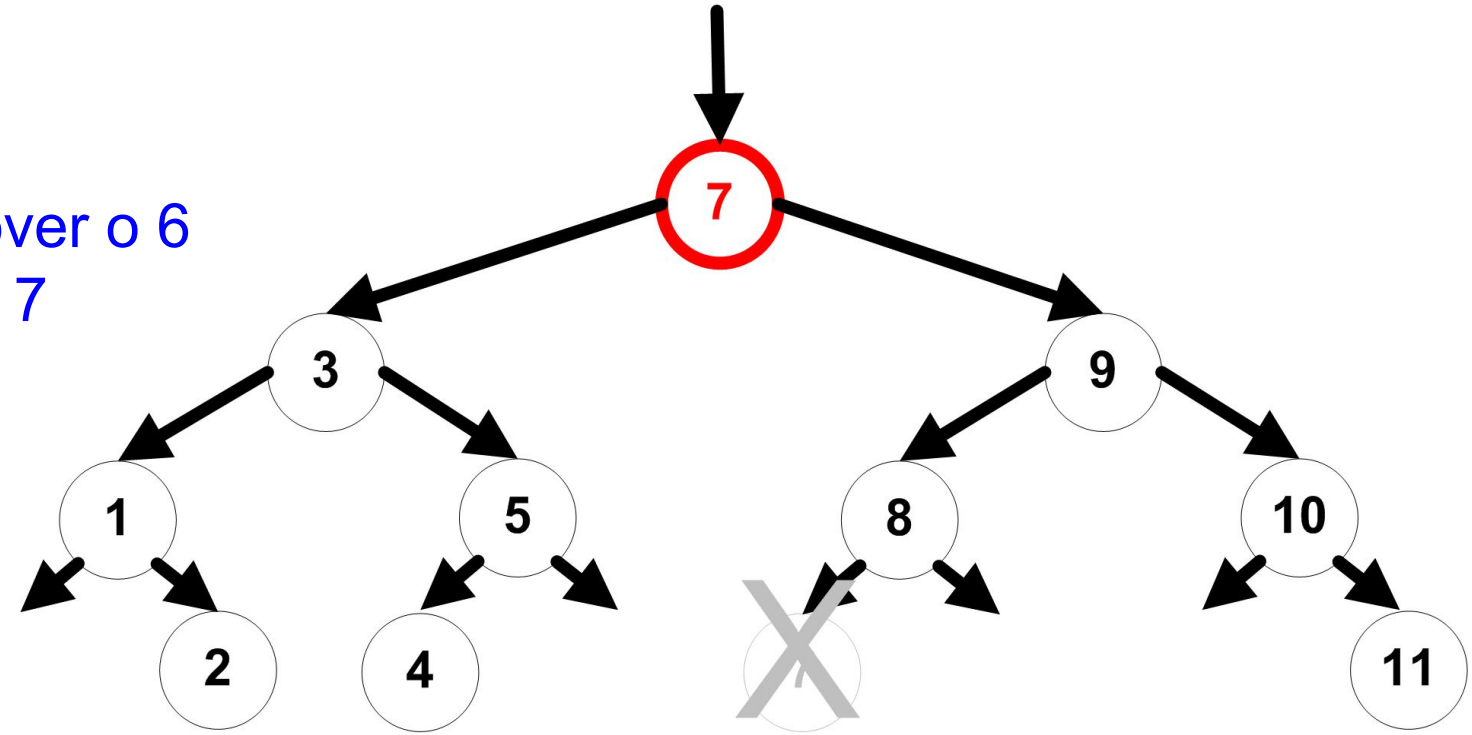
Exemplo: Remover o 6 e substituir pelo 7



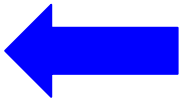
Funcionamento Básico da Remoção

(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 **menor nó da subárvore à direita**

Exemplo: Remover o 6 e substituir pelo 7



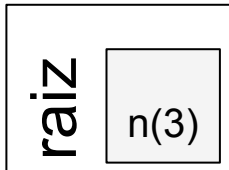
Agenda

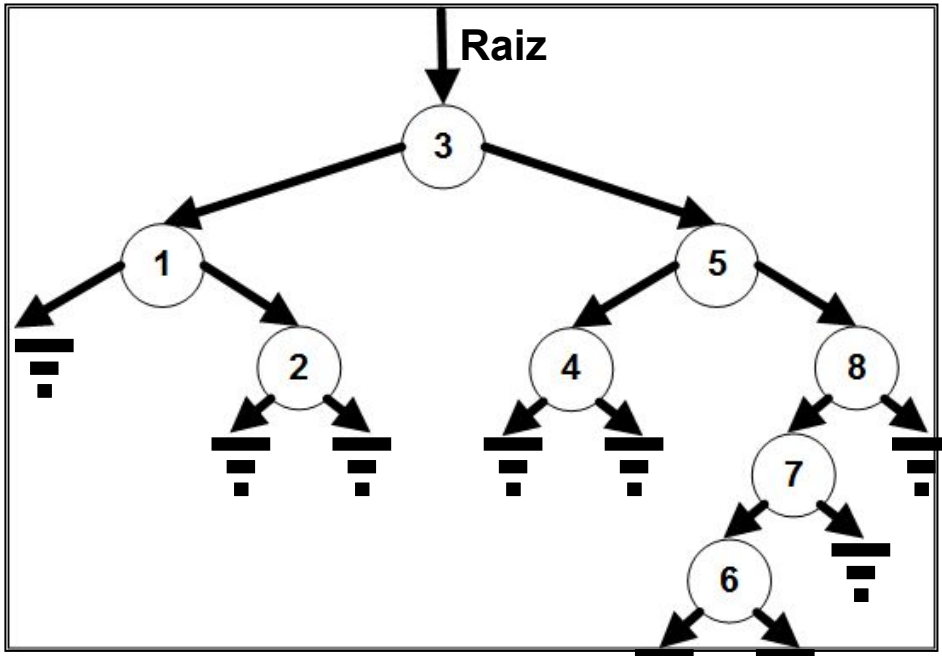
- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Caminhamento
- **Remoção** 
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

  - Funcionamento básico
  - **Algoritmo em Java**
  - Análise de Complexidade

Classe Árvore Binária em Java

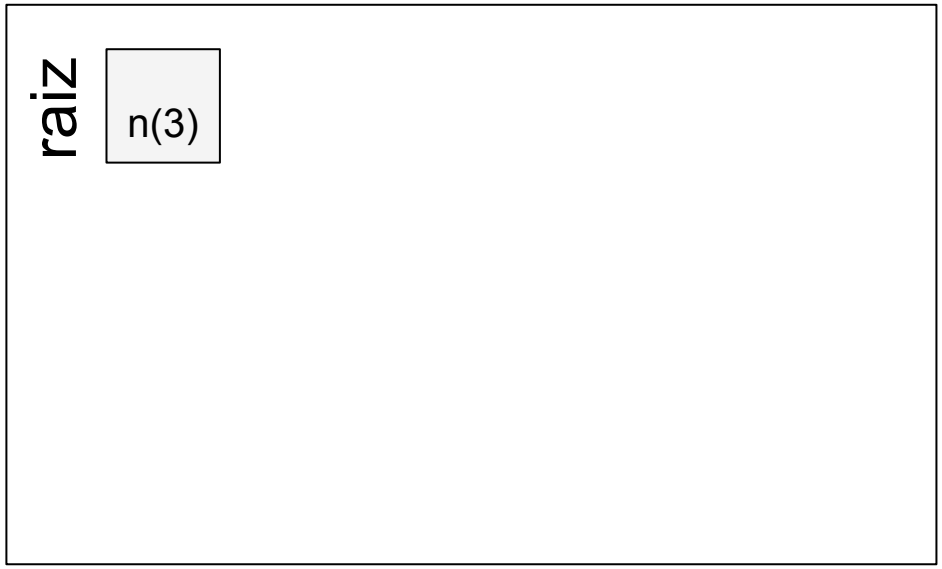
```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    boolean pesquisar(int x) {   }
    void remover(int x) {  }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
}
```



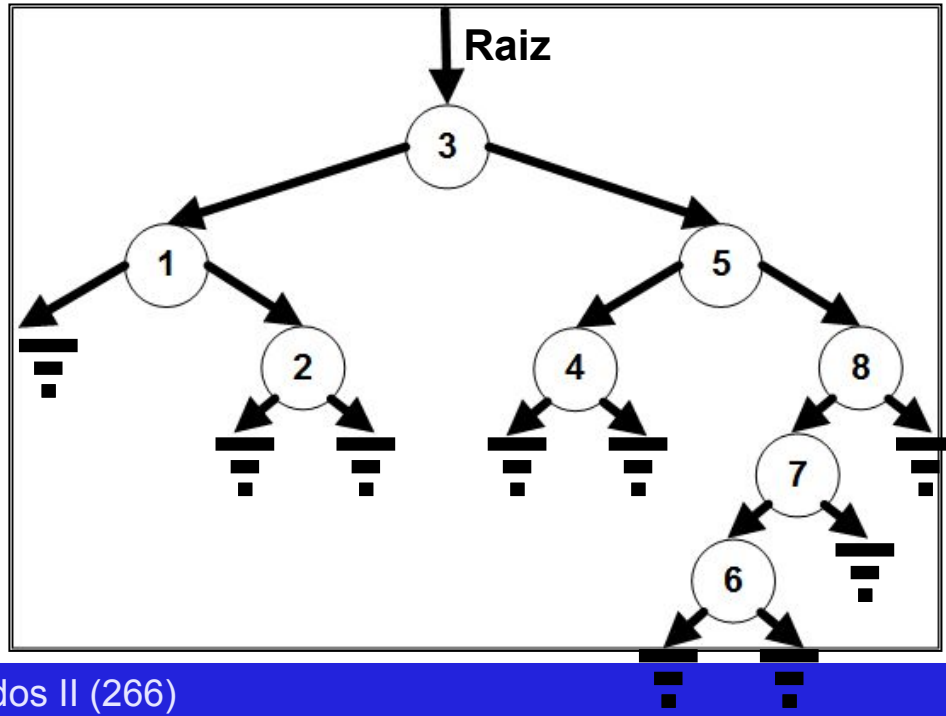


Algoritmo de Remoção em Java

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    boolean pesquisar(int x) {   }
    void remover(int x) {  }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
}
```



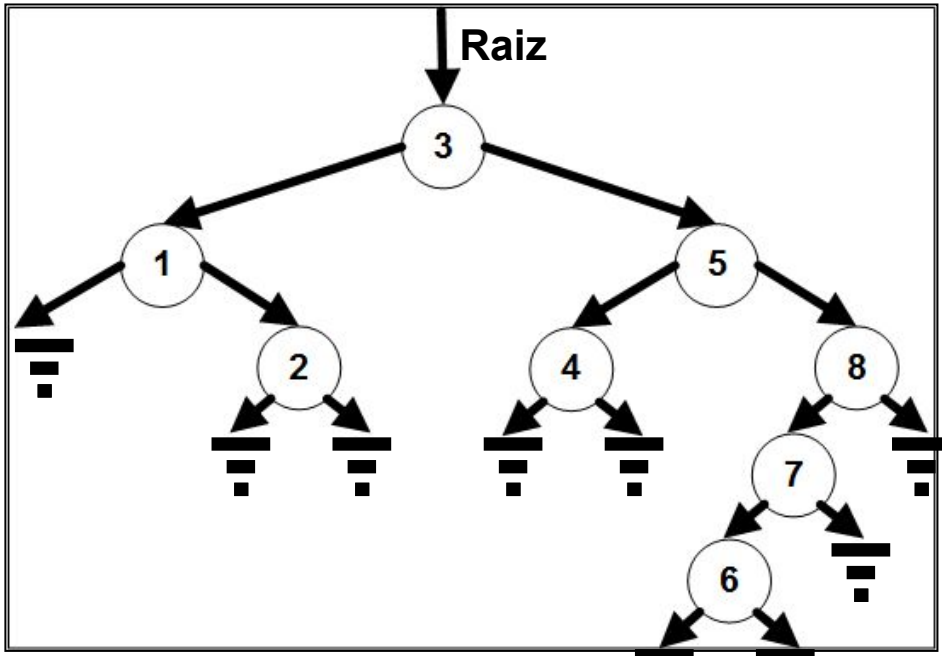
Vamos remover o 2 (uma folha) de nossa árvore



Algoritmo de Remoção em Java

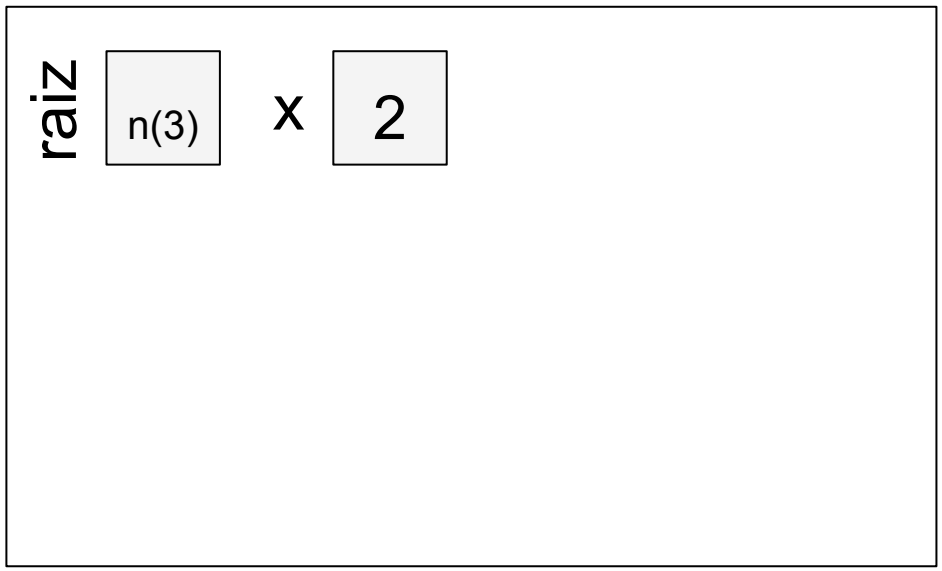
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

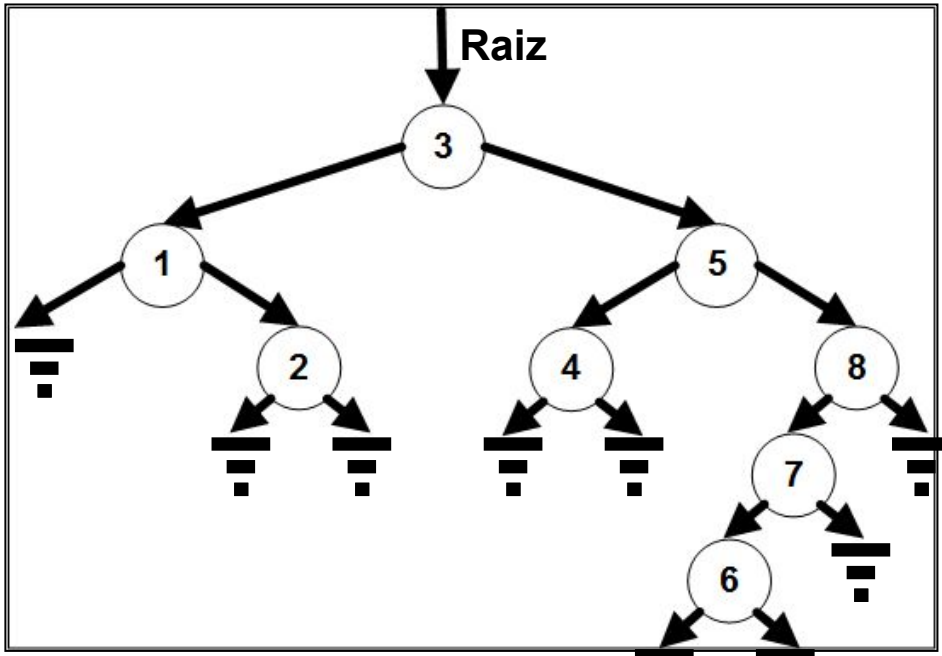
raiz | n(3) | x | 2



Algoritmo de Remoção em Java

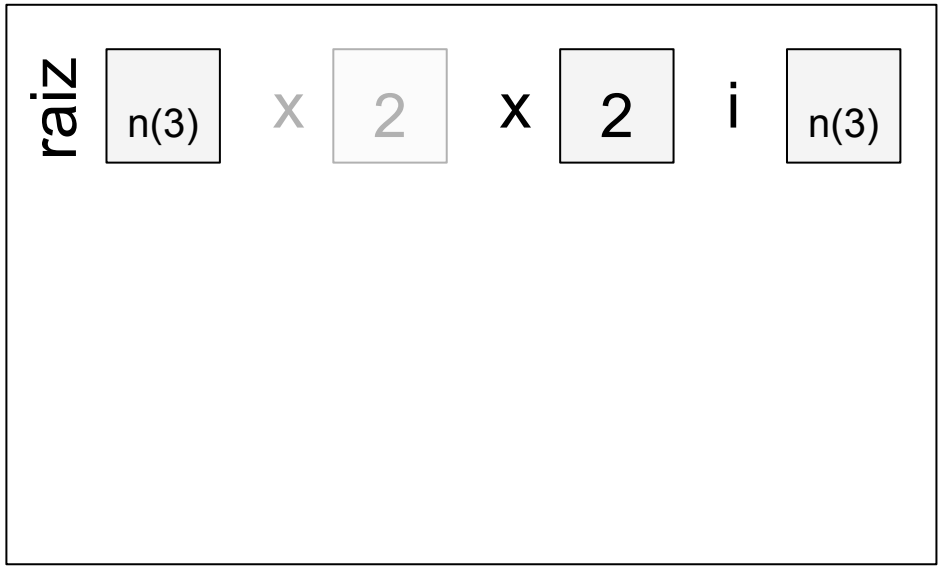
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

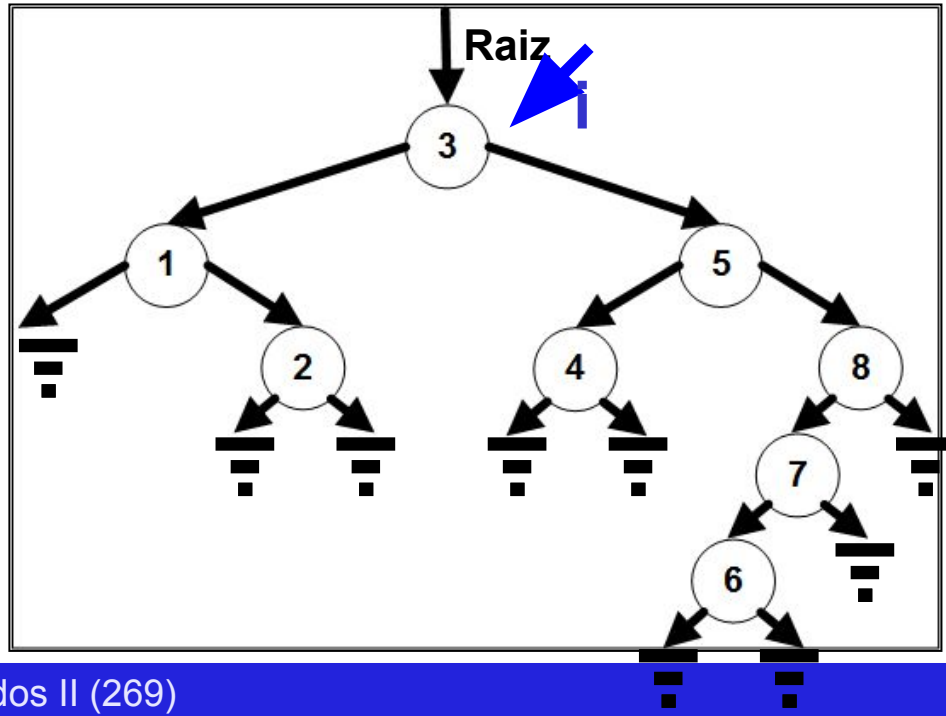




Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

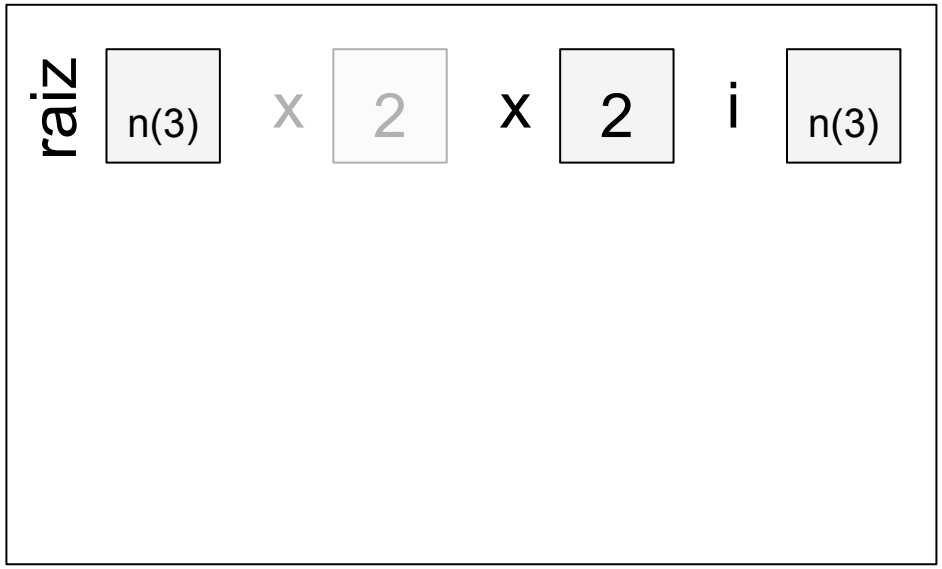


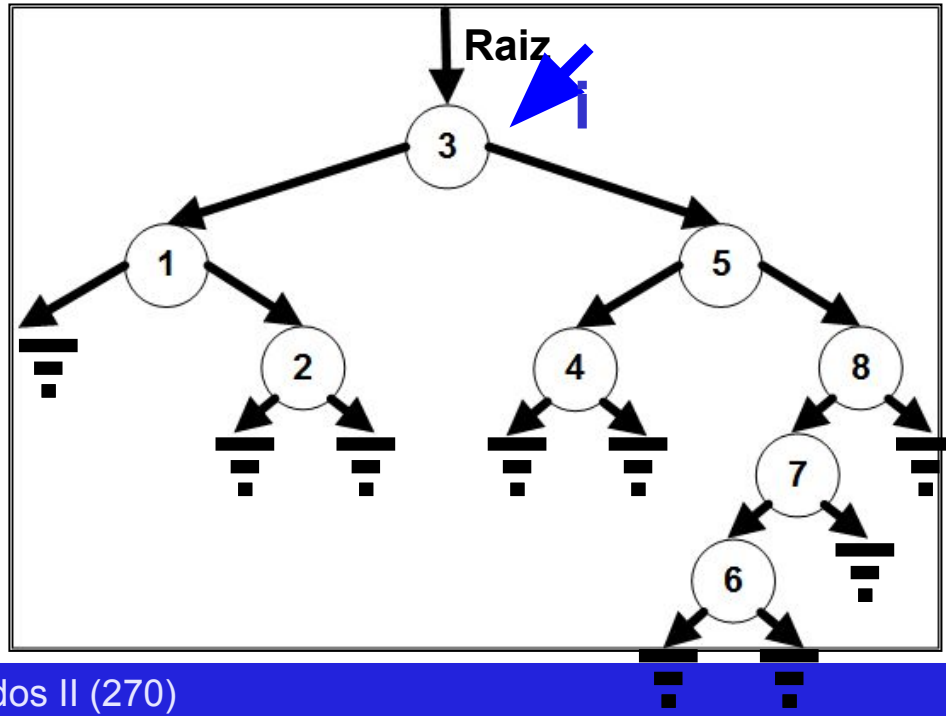


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {      throw new("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;
}

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;
}
```

false: n(3) == null

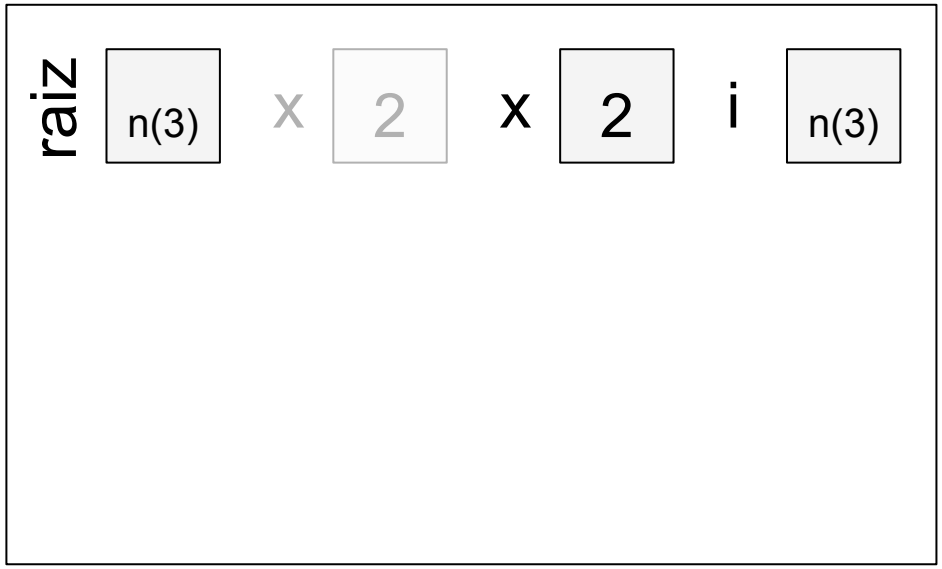


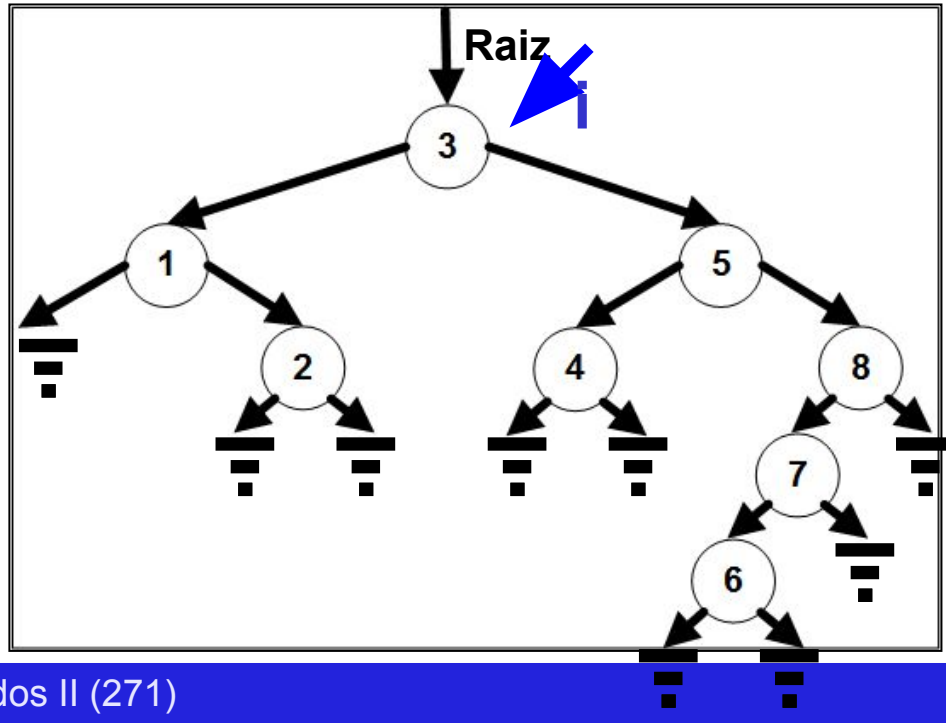


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

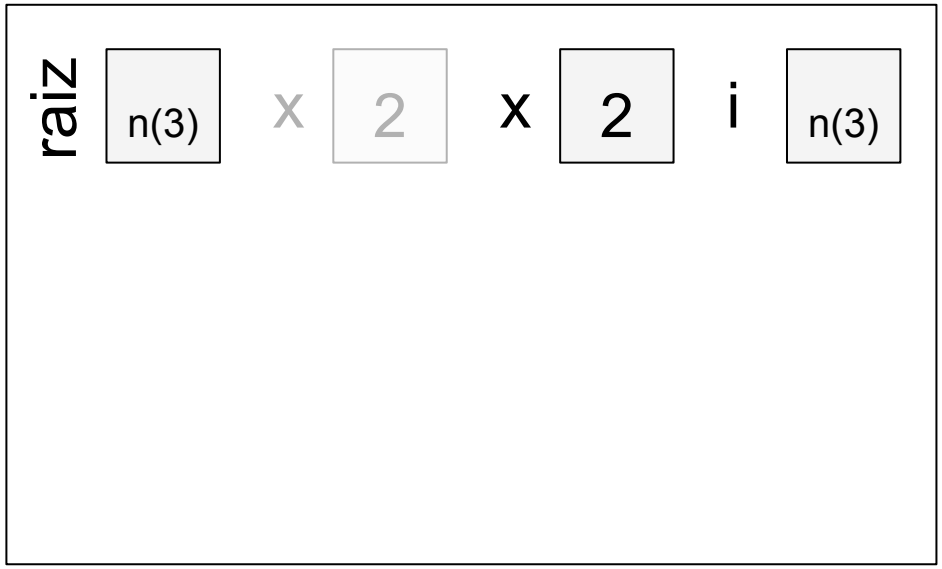
true: 2 < 3

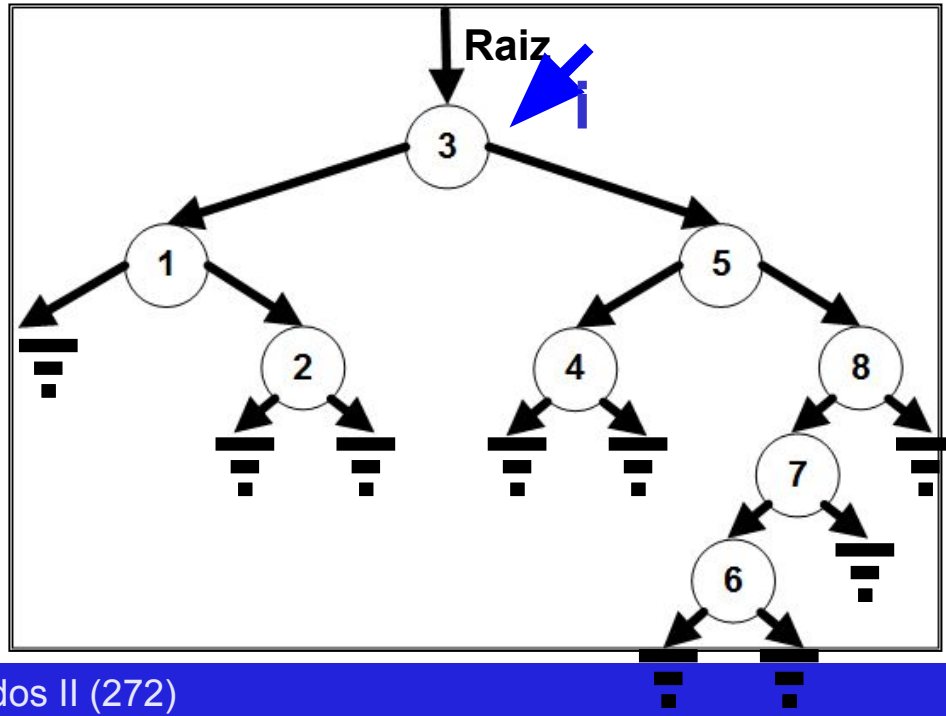




Algoritmo de Remoção em Java

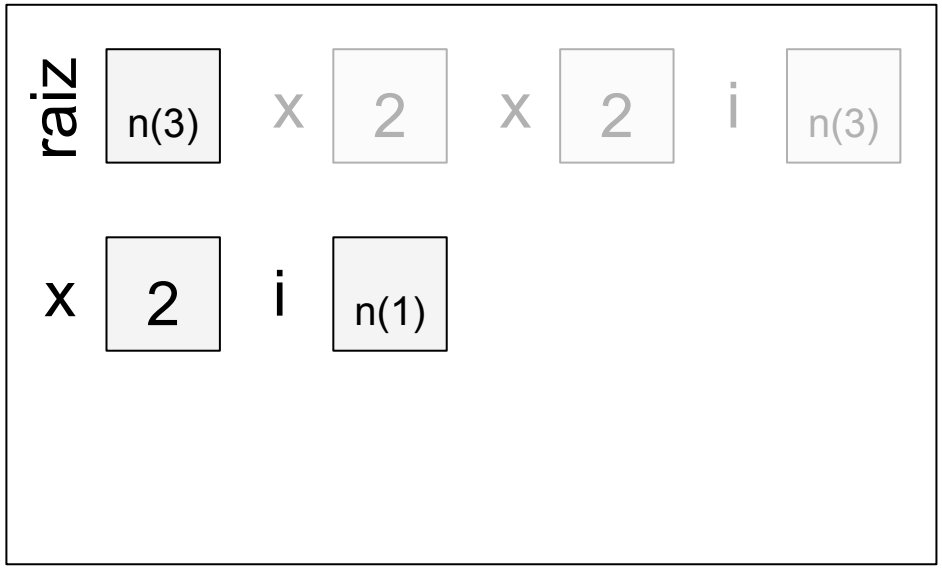
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

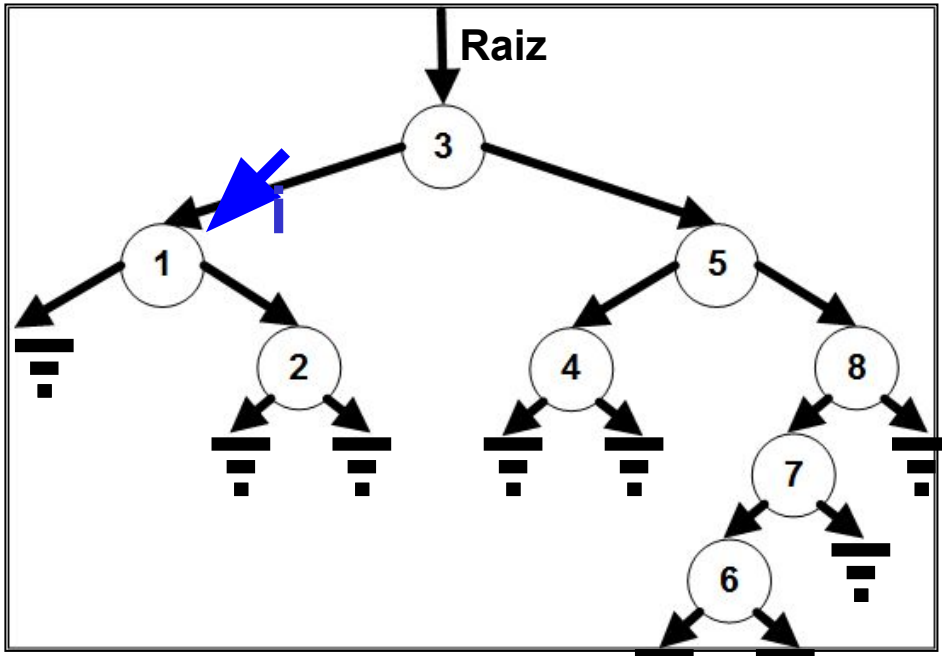




Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

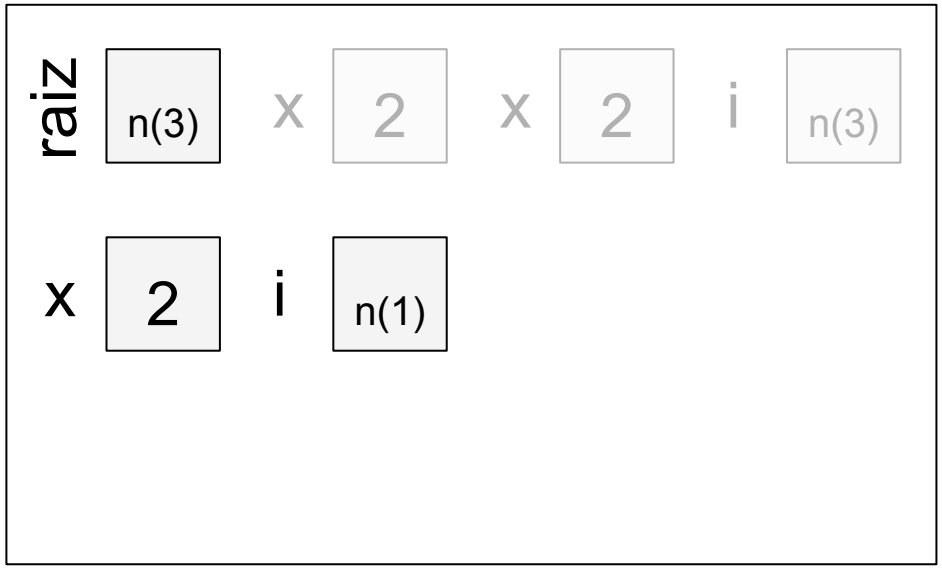


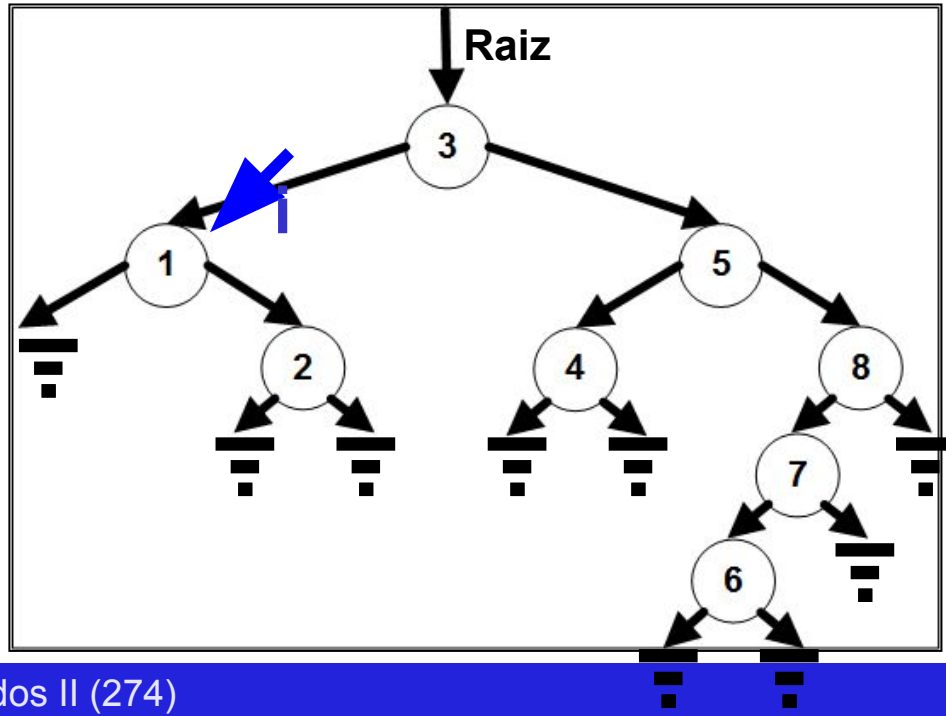


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {      throw new("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;
}

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;
}
```

false: n(1) == null

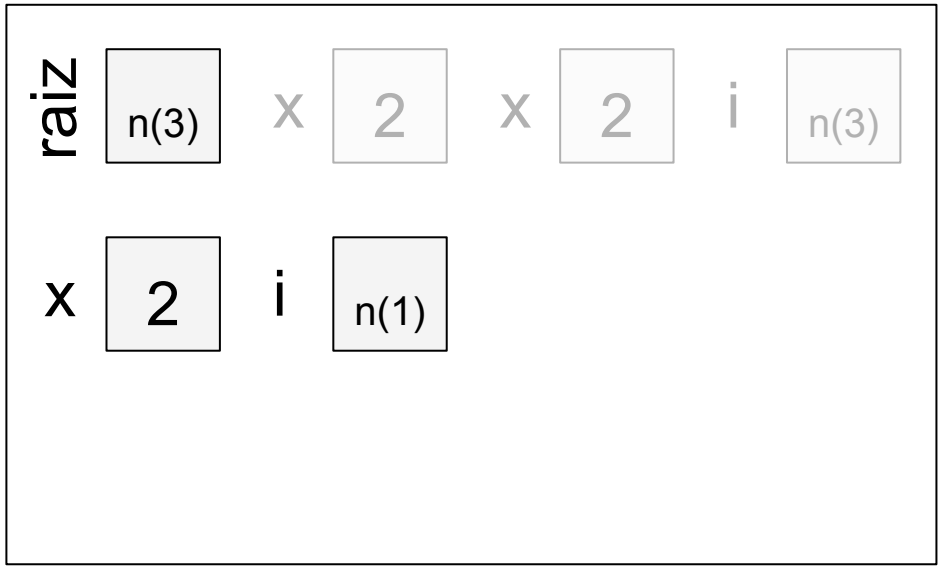


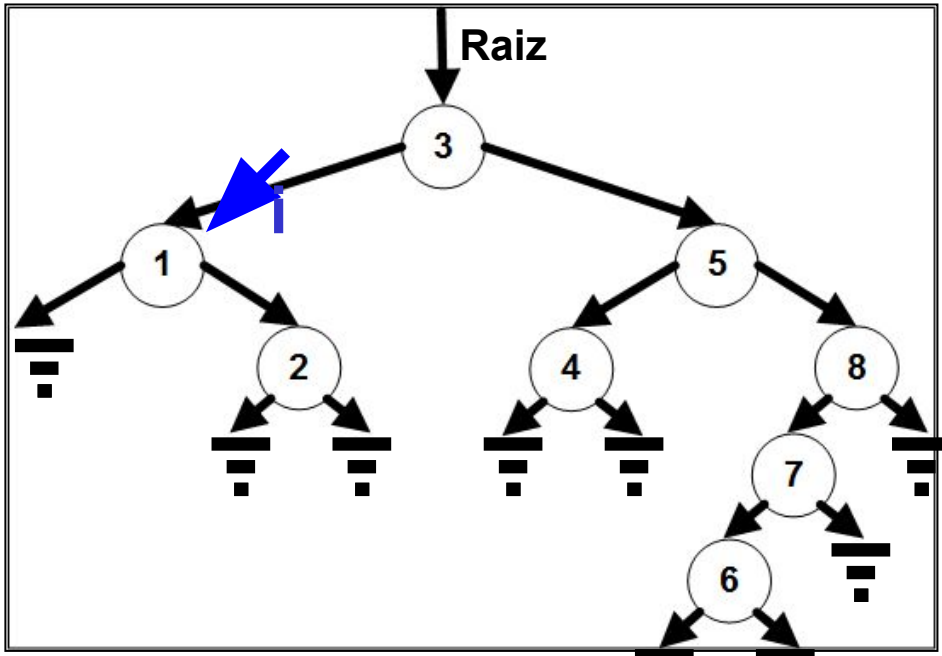


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: 2 < 1

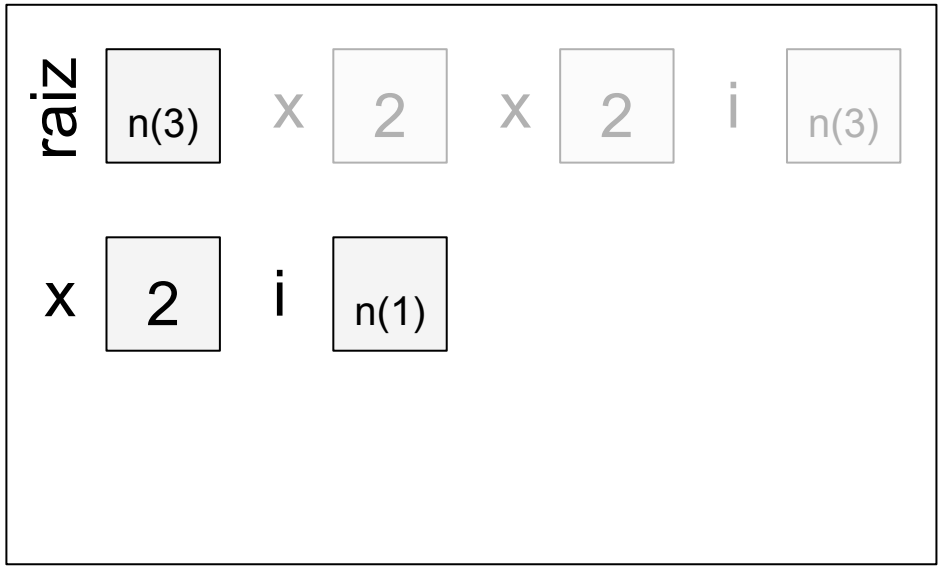


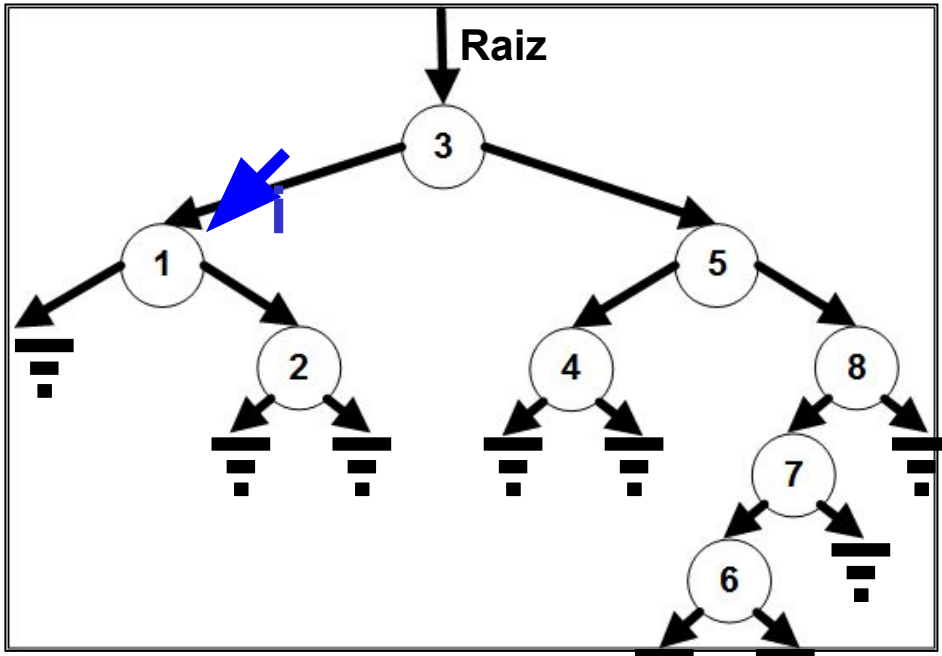


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

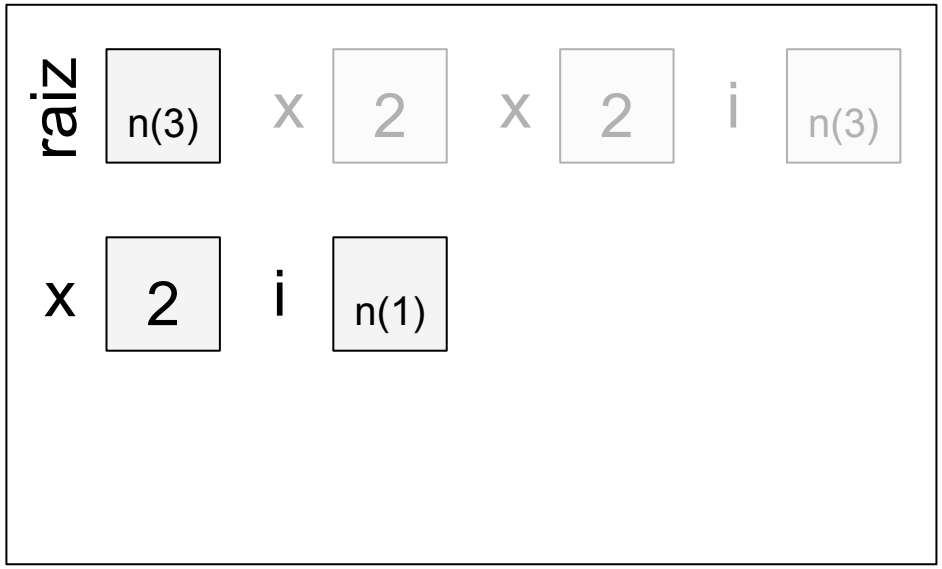
true: 2 > 1

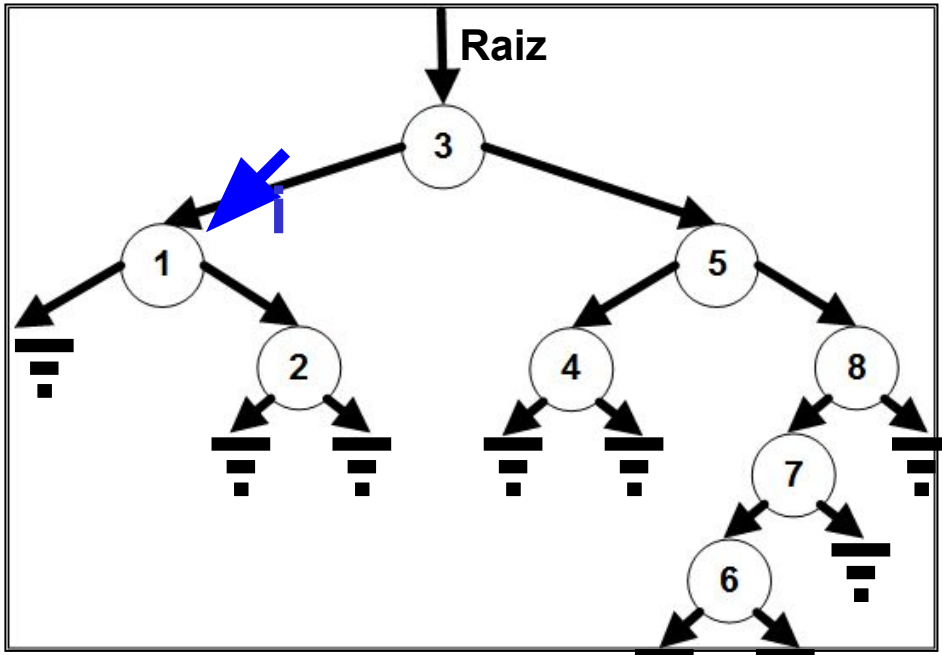




Algoritmo de Remoção em Java

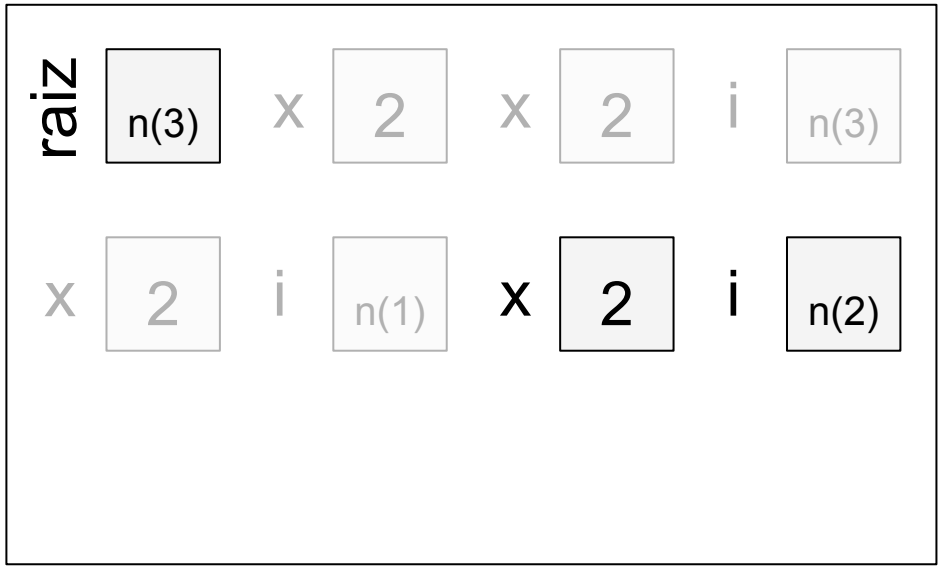
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

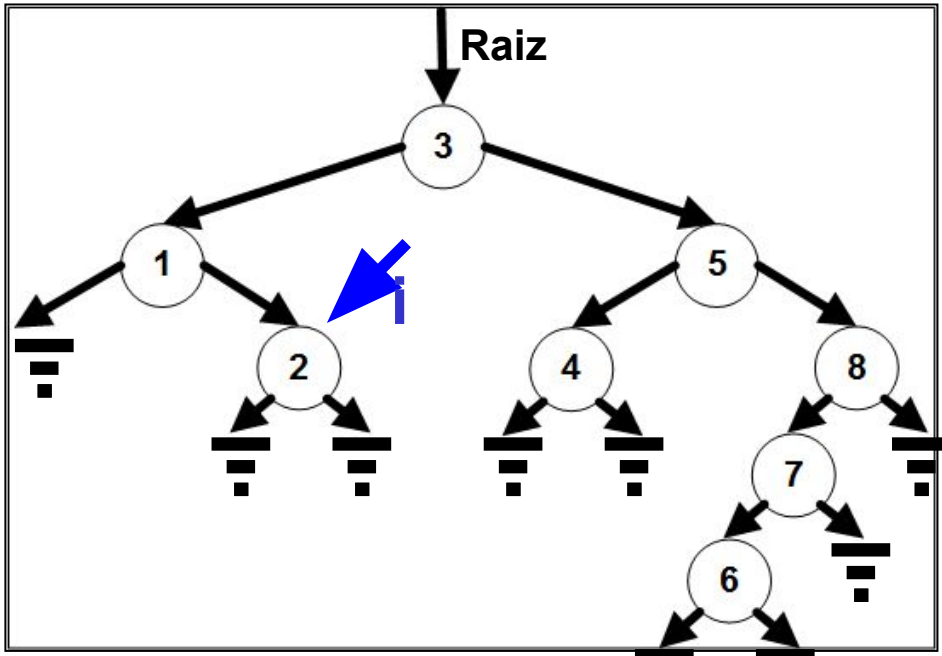




Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

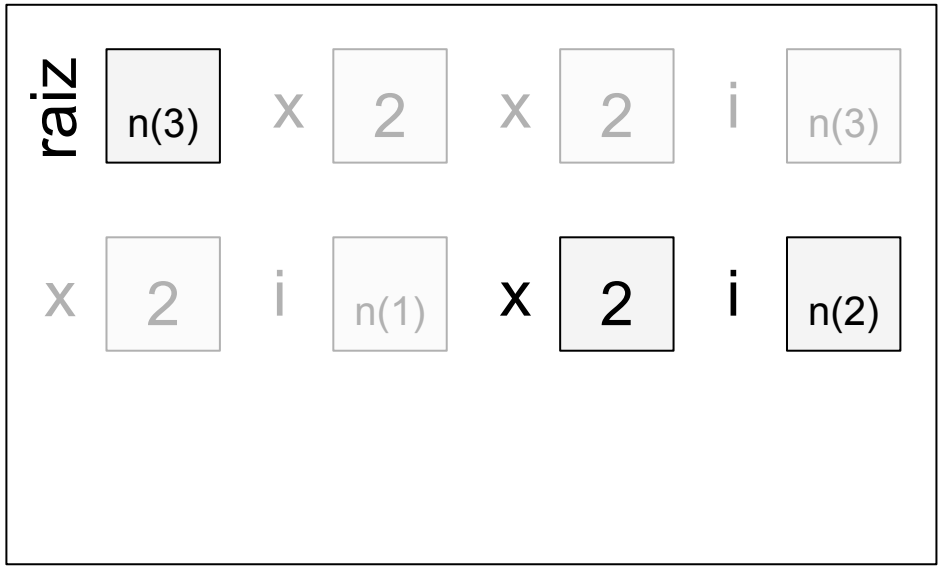


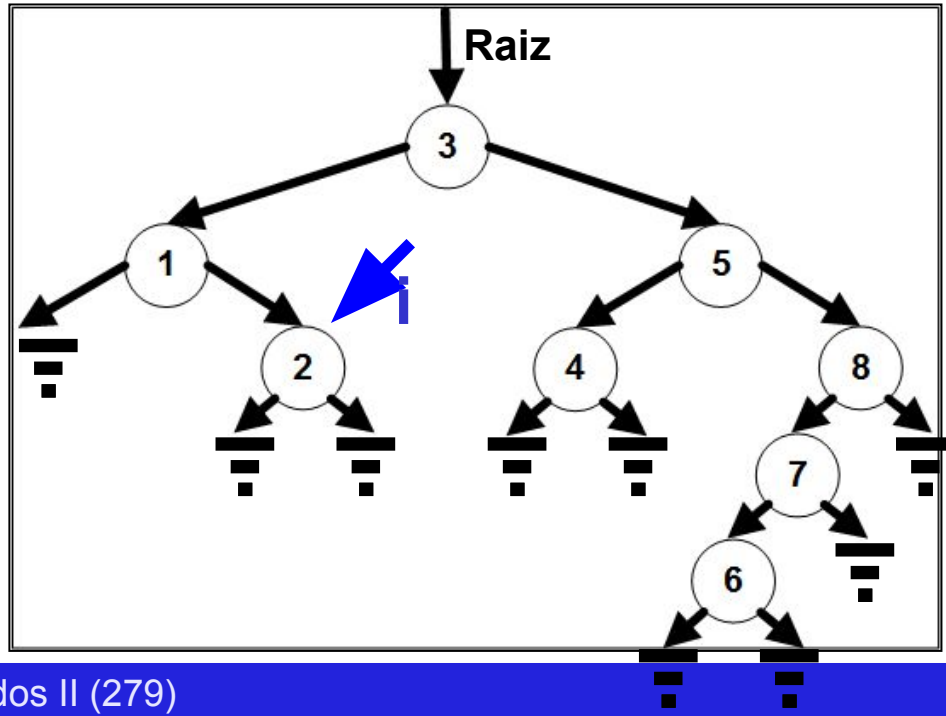


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {      throw new("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;
}

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;
}
```

false: n(2) == null

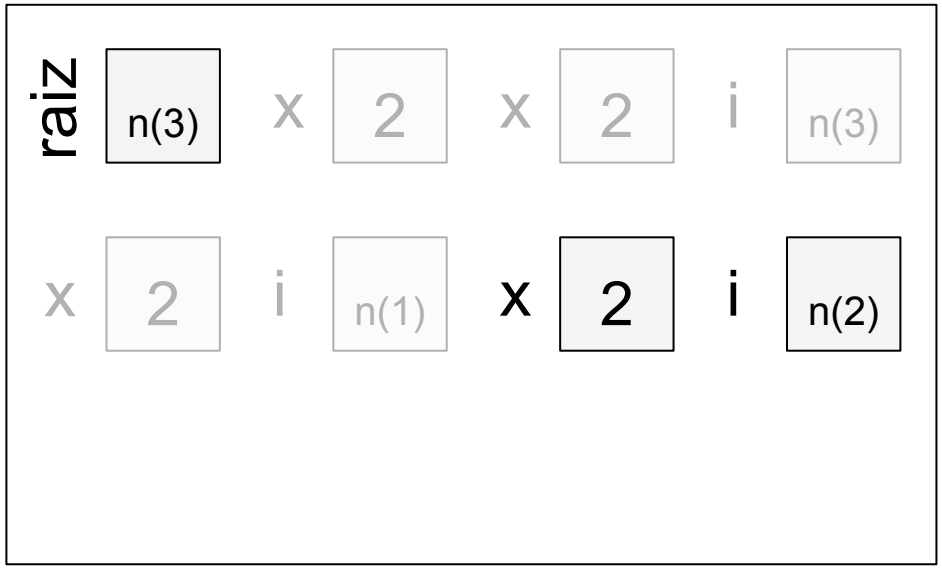


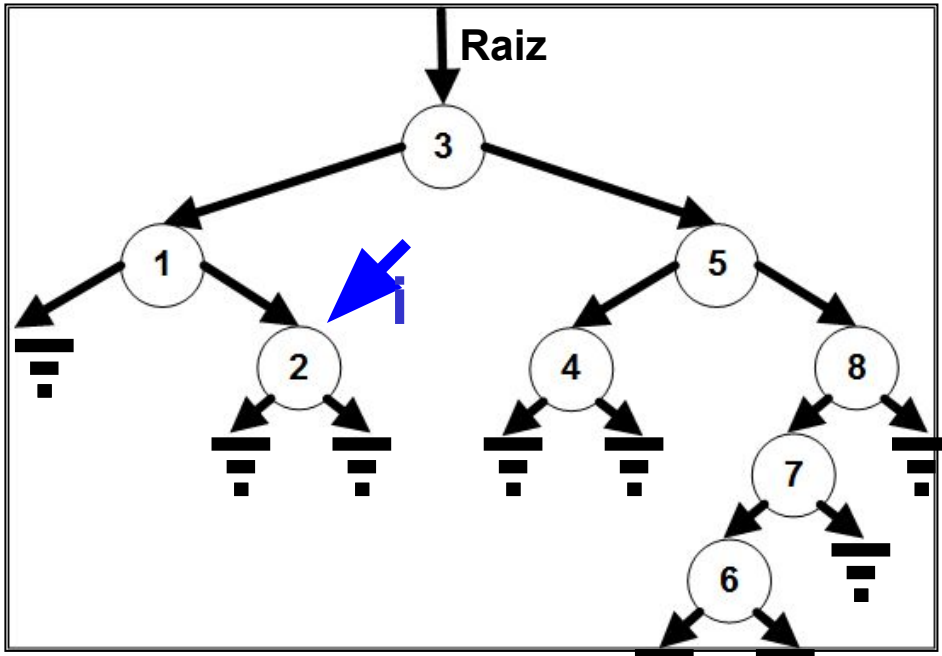


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: 2 < 2

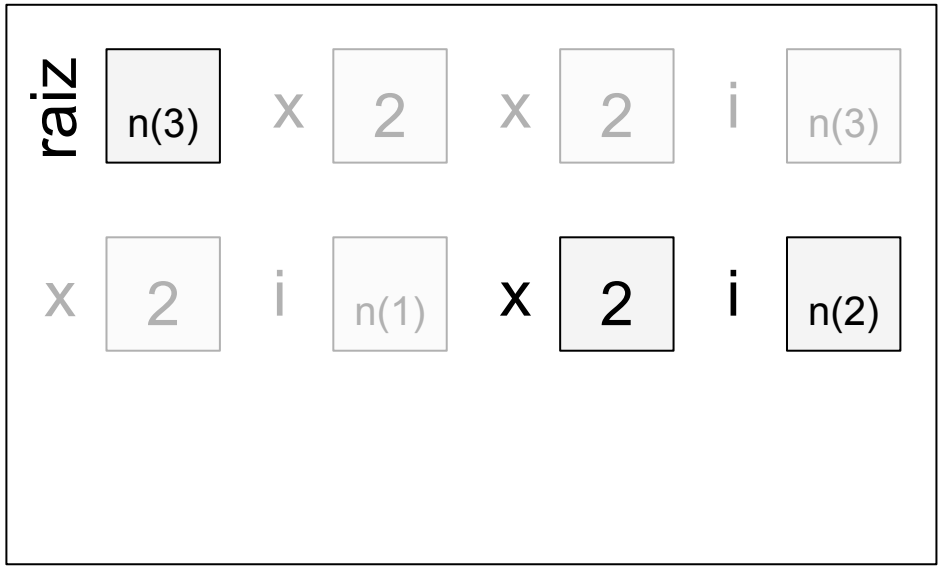


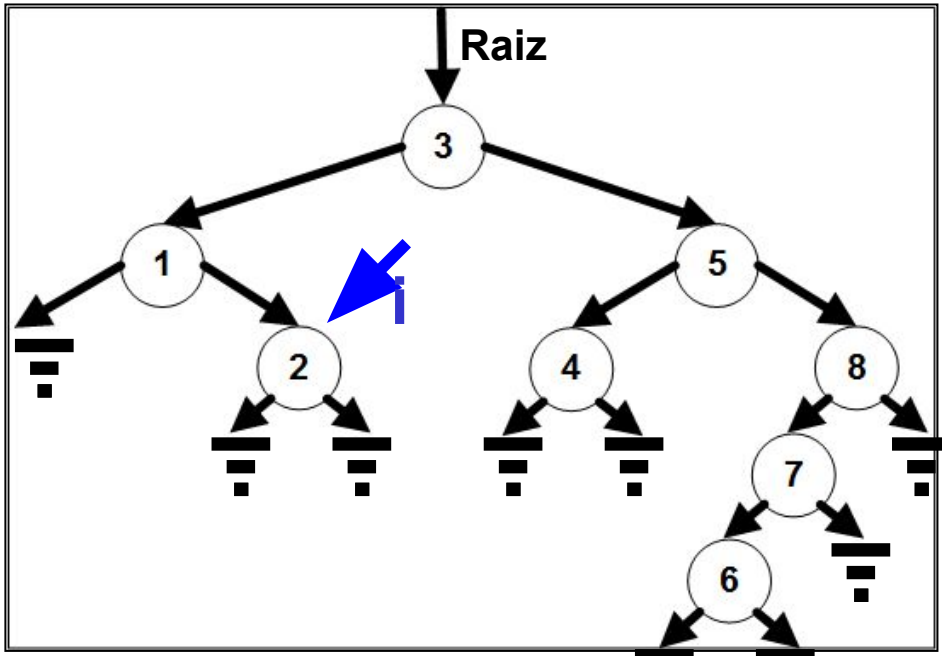


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: 2 > 2

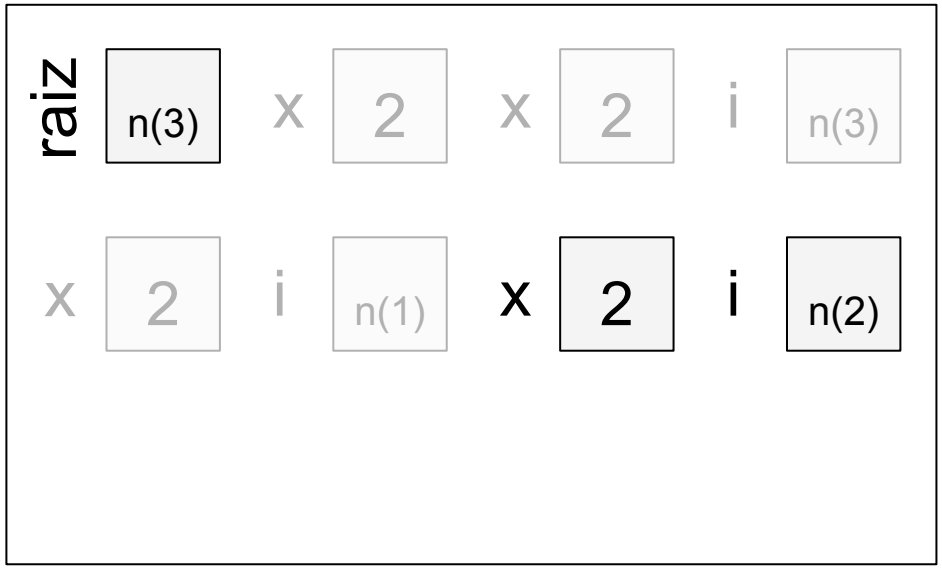


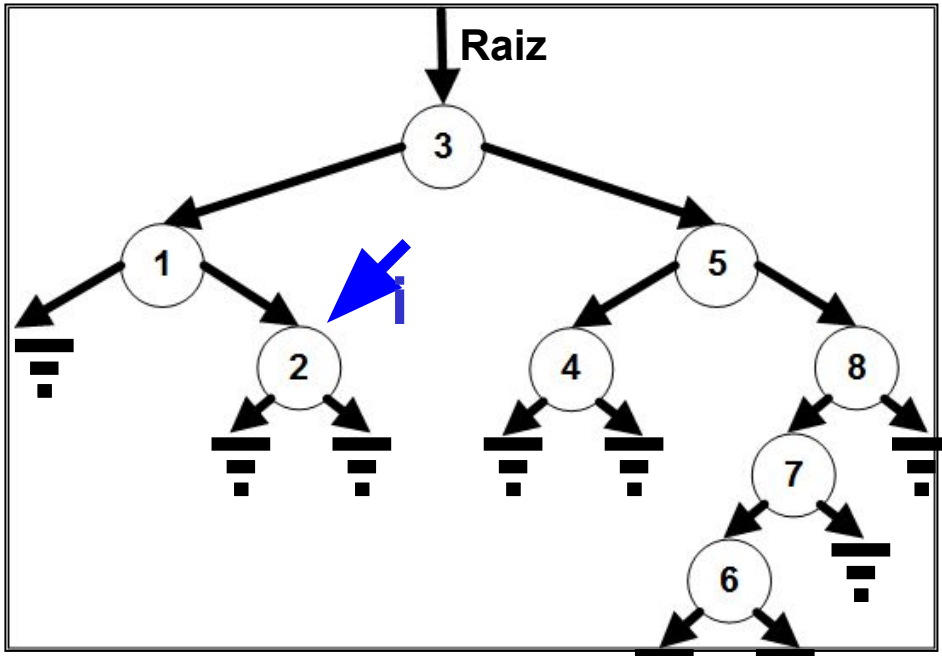


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

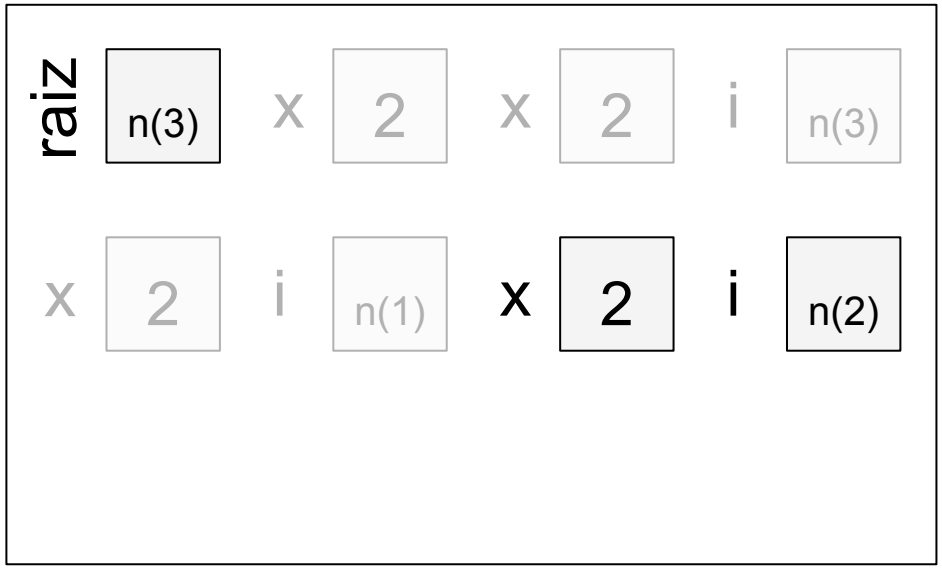
true: null == null

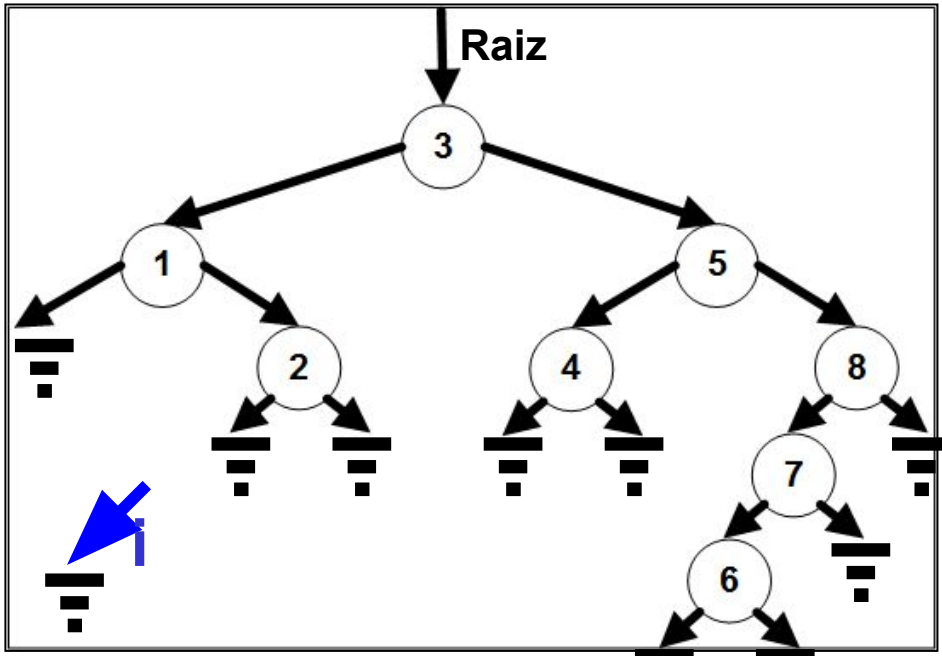




Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



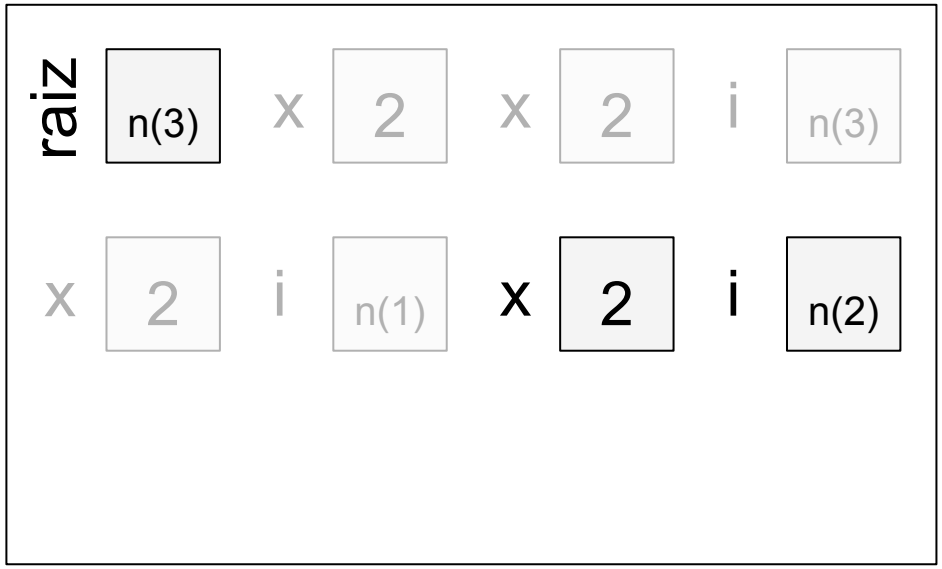


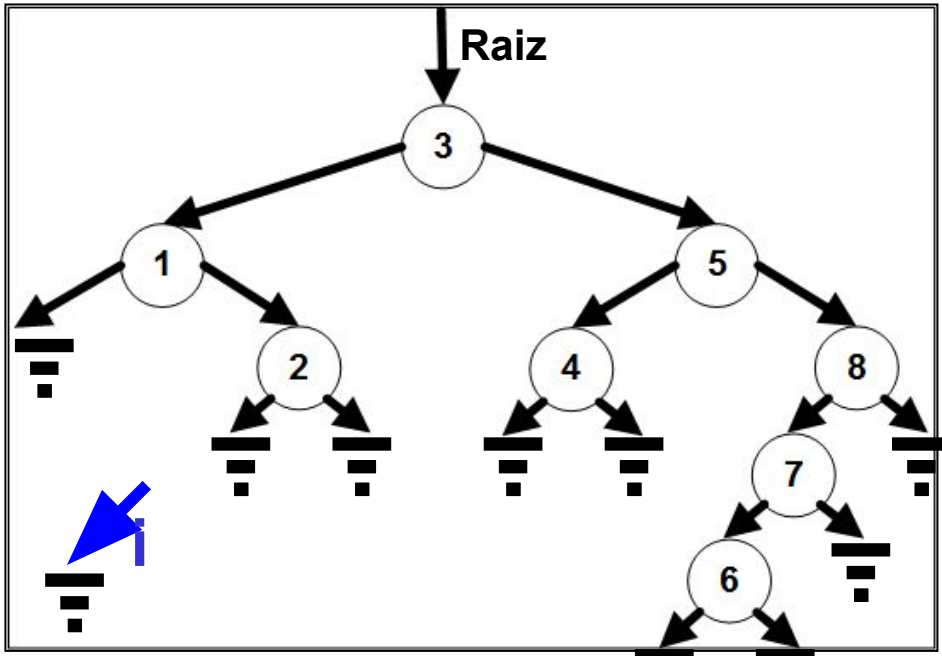
Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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

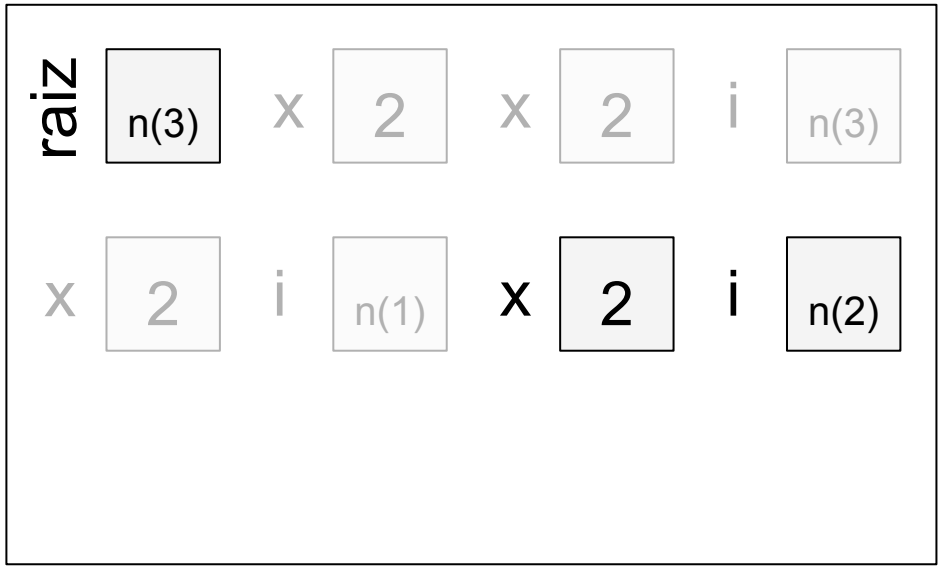
```
No anterior(No i, No j) {
   if (j.dir != null)  j.dir = anterior(i, j.dir);
   else {              i.elemento = j.elemento; j = j.esq; }
   return j;
}
```

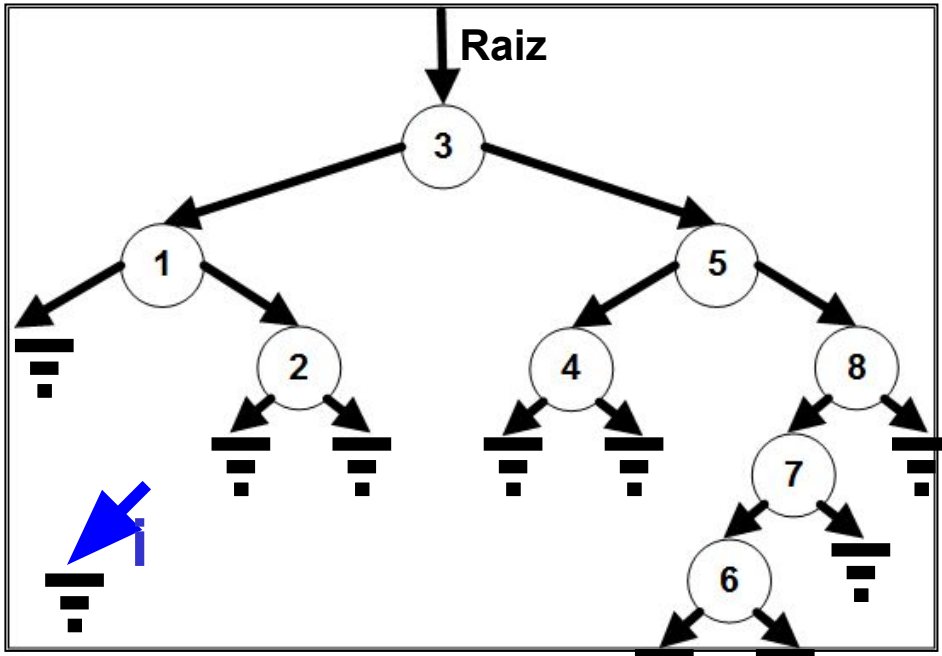




Algoritmo de Remoção em Java

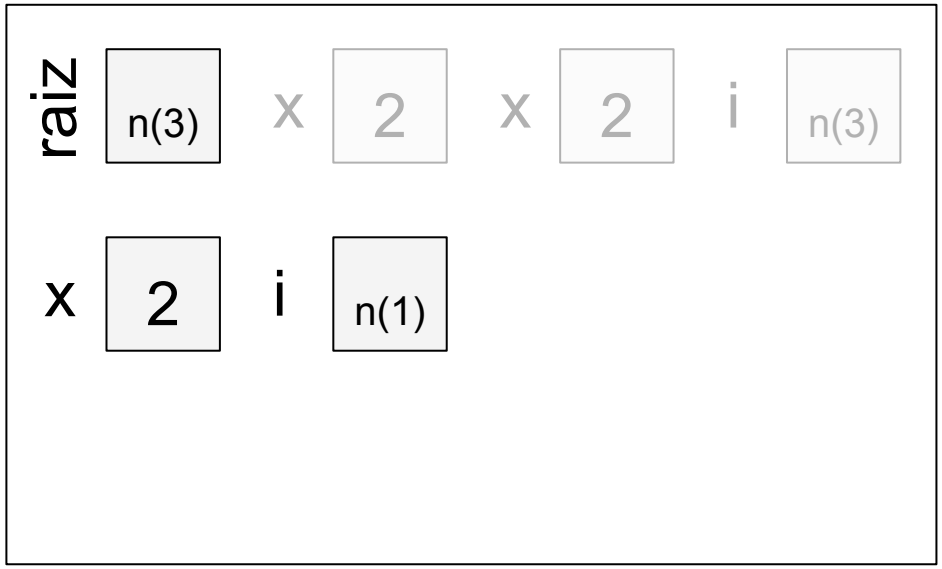
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

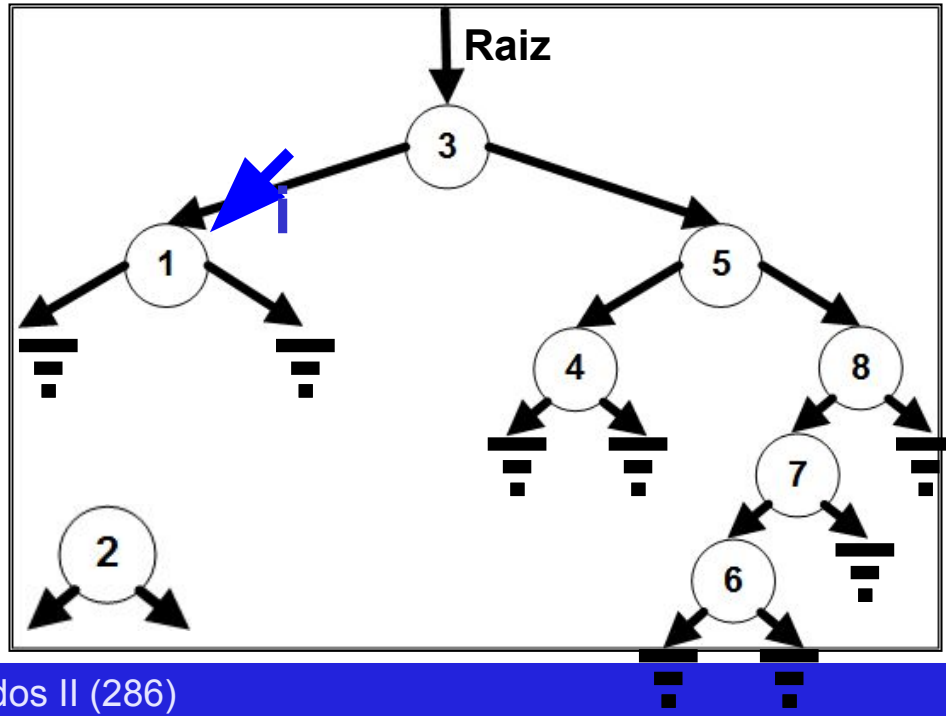




Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



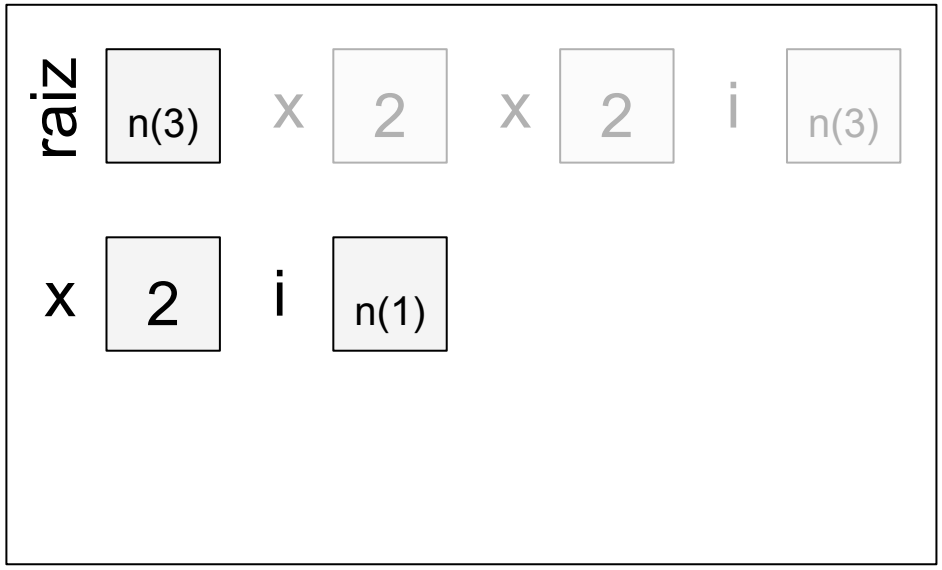


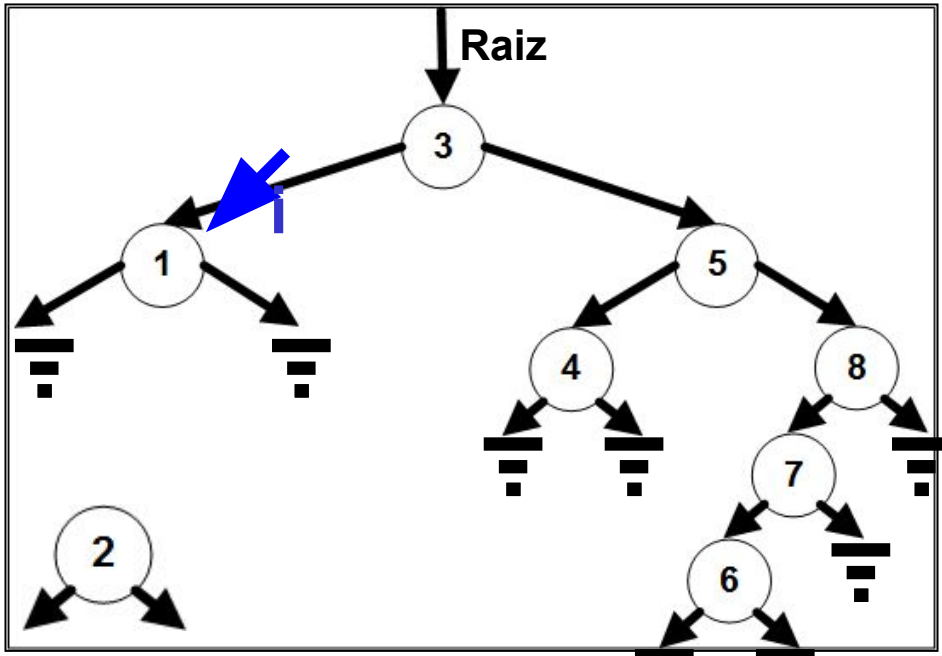
Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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(1)

```
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;
}
```



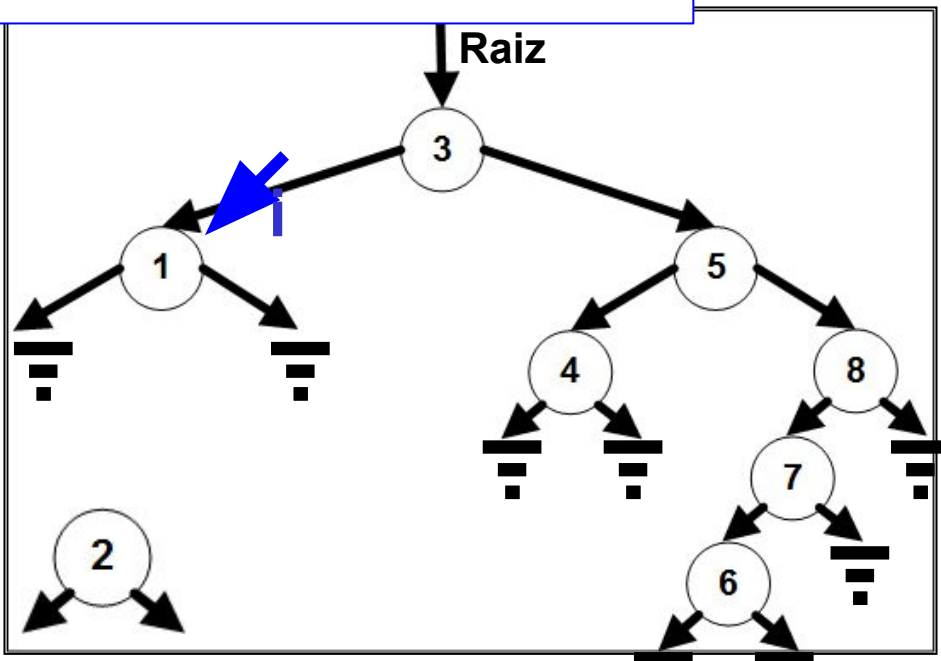


Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remo
   if (i ==
   } else
   } else
   } 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;
}

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;
}
```

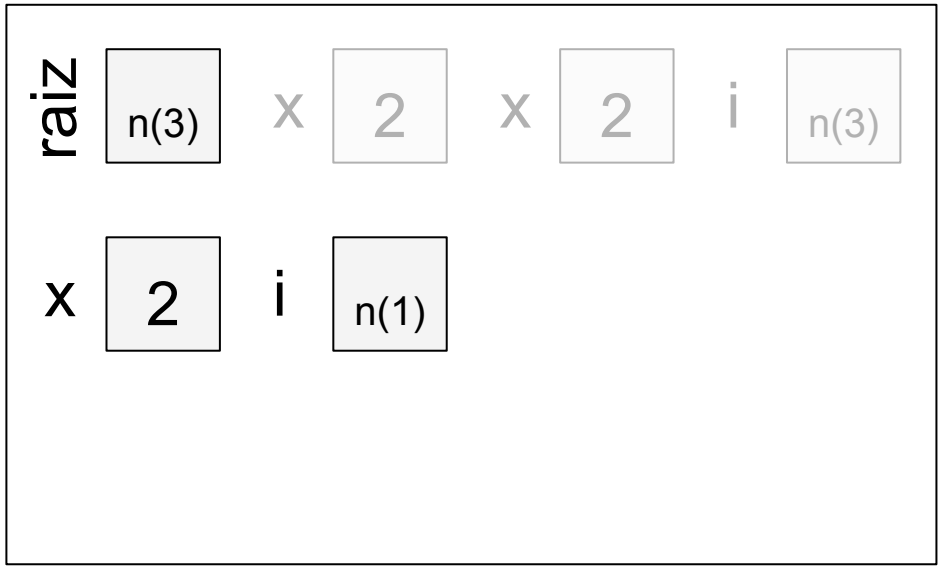
raiz | n(3) | x | 2 | x | 2 | i | n(3)

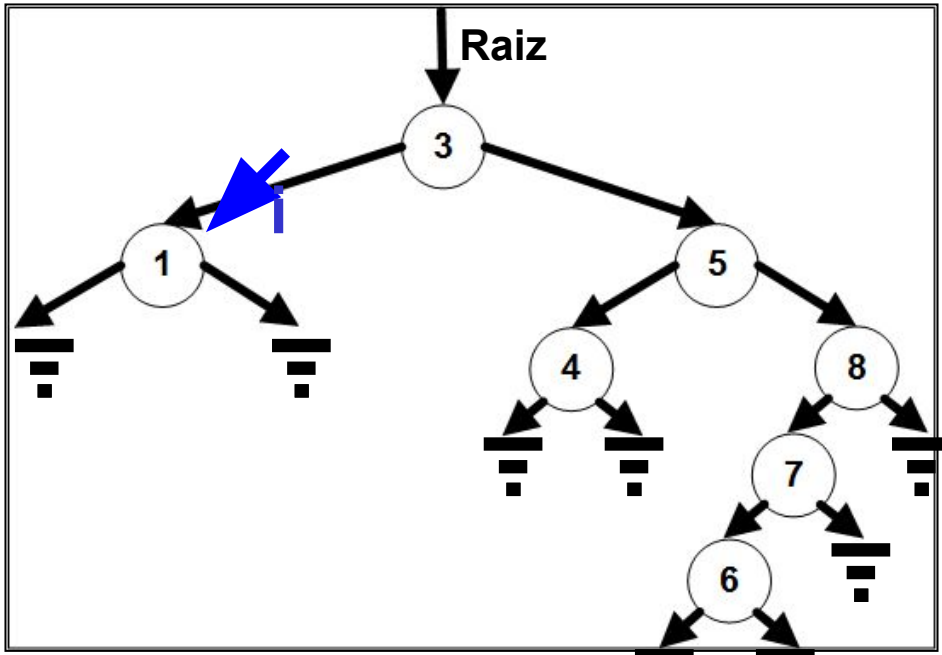
Após a coleta de lixo do Java
(que não controlamos quando ela acontece)...



Algoritmo de Remoção em Java

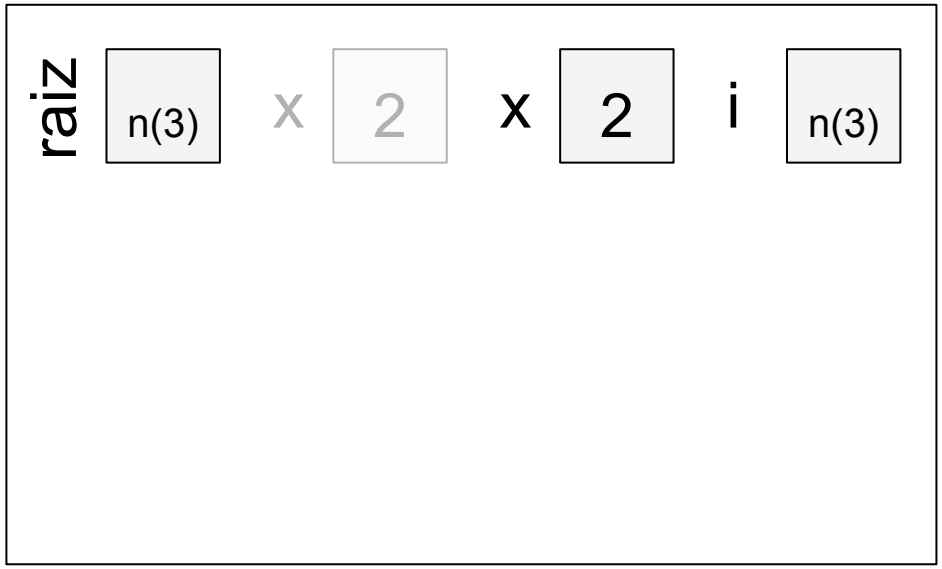
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

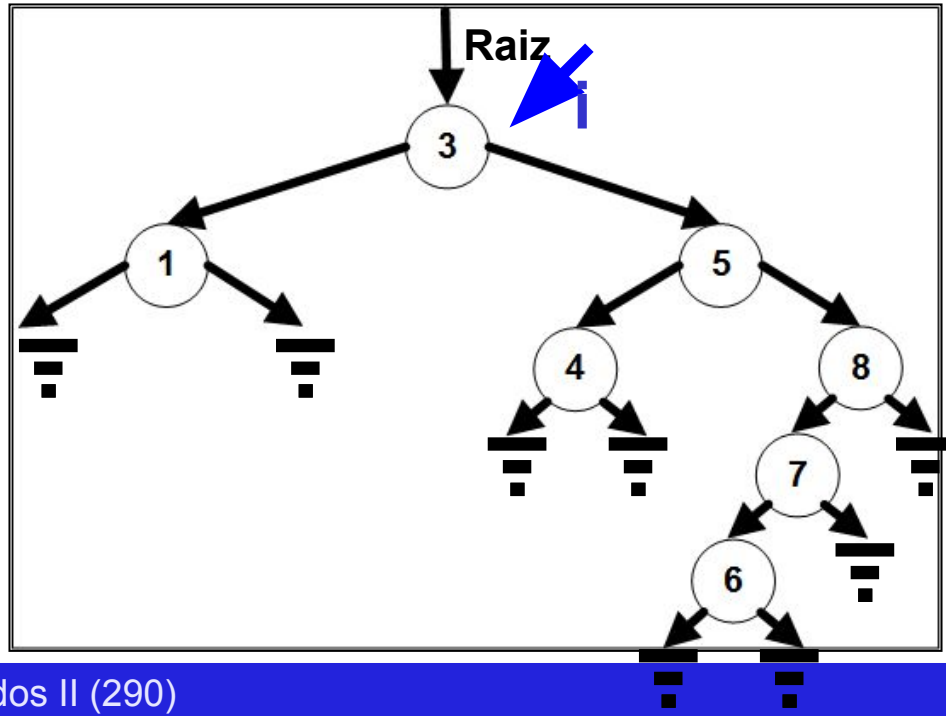




Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



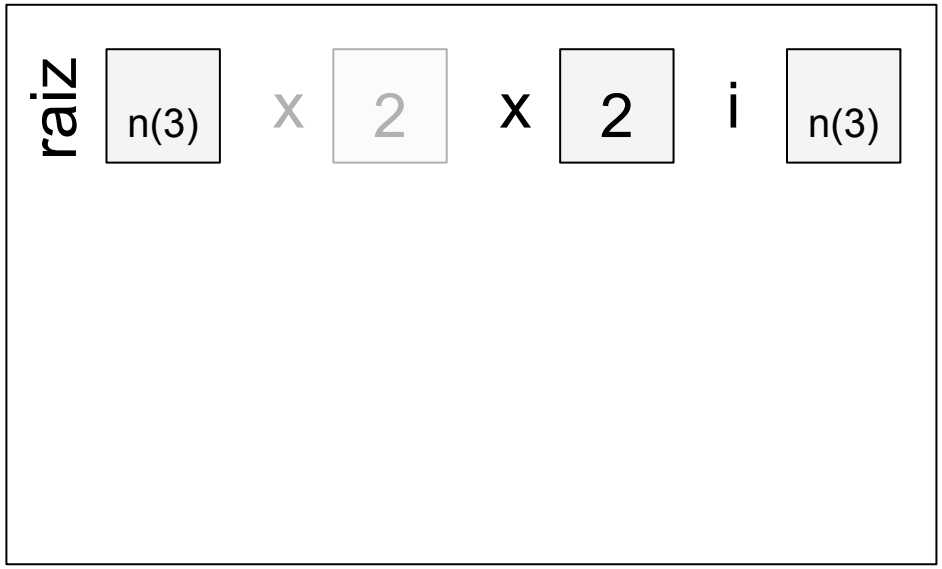


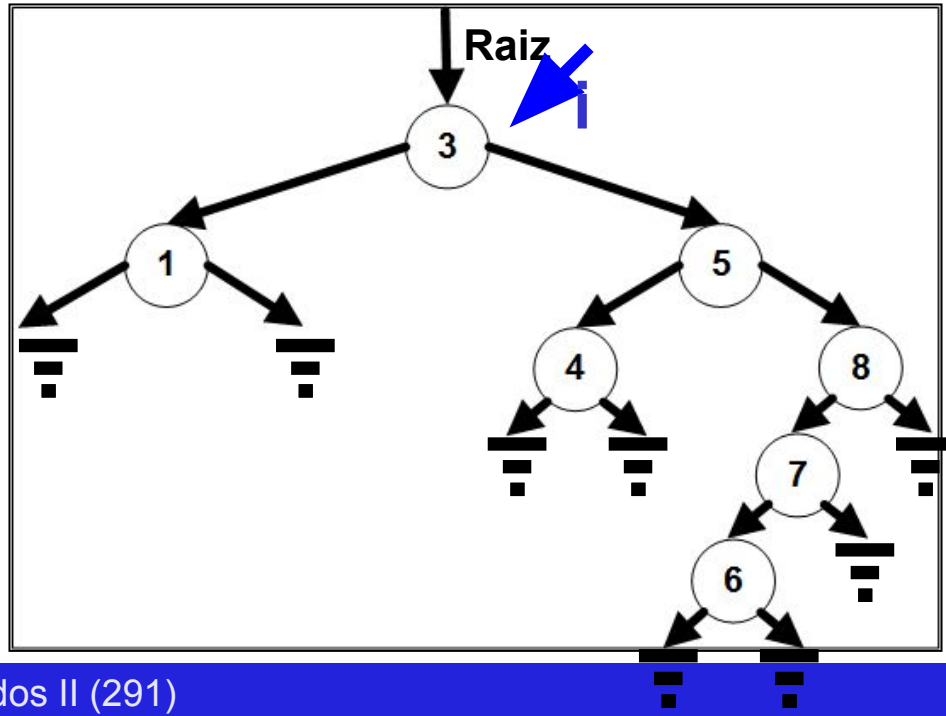
Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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)

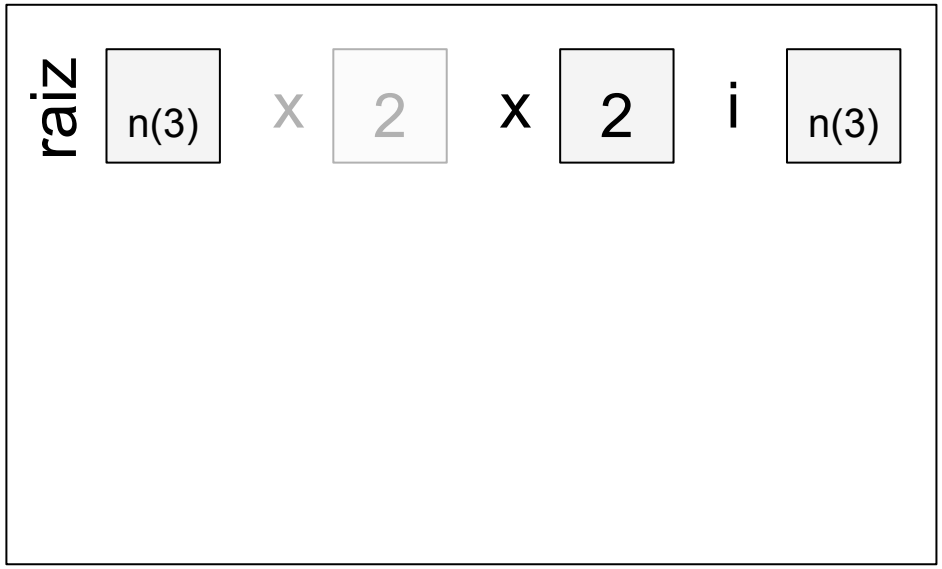
```
No anterior(No i, No j) {
   if (j.dir != null)  j.dir = anterior(i, j.dir);
   else {              i.elemento = j.elemento; j = j.esq; }
   return j;
}
```

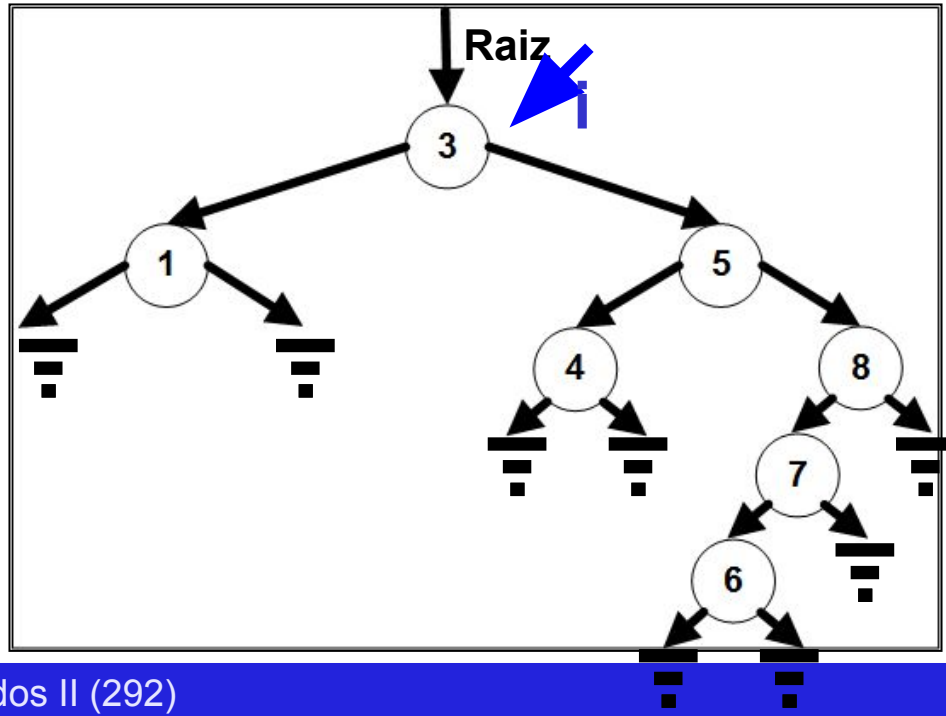




Algoritmo de Remoção em Java

```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

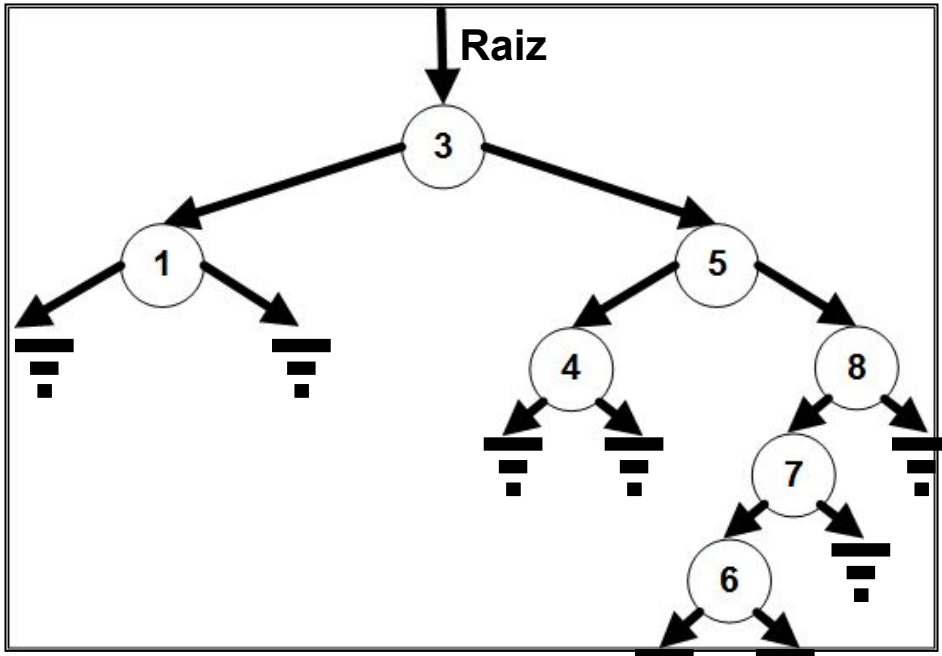




Algoritmo de Remoção em Java

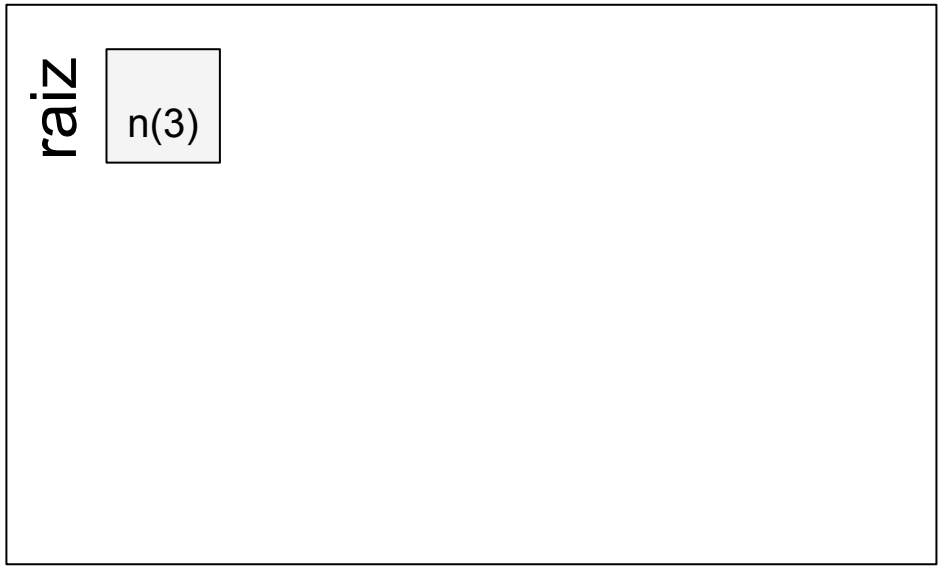
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

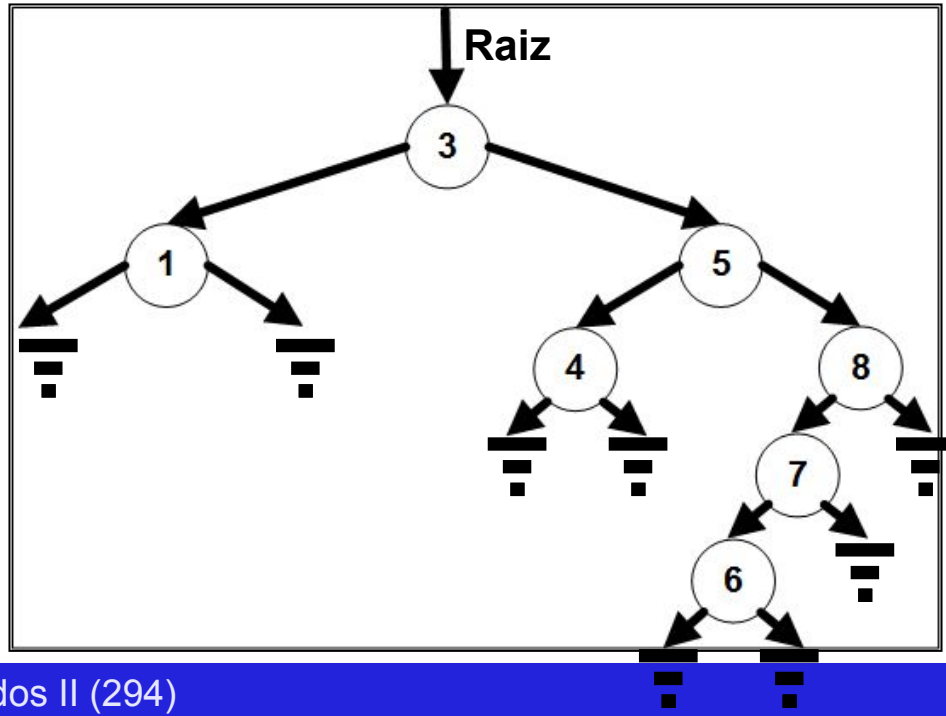
raiz | n(3) | x | 2



Algoritmo de Remoção em Java

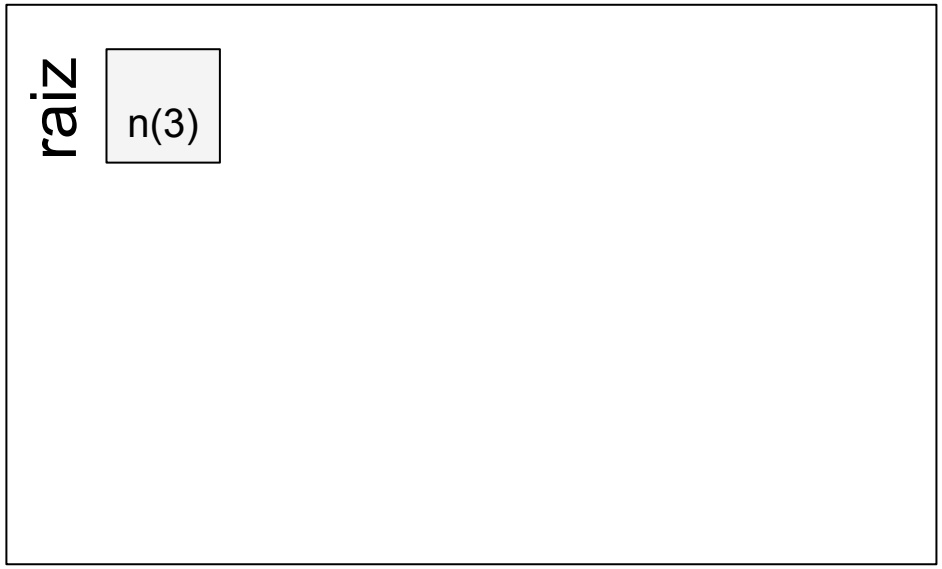
```
//remover(2), folha

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



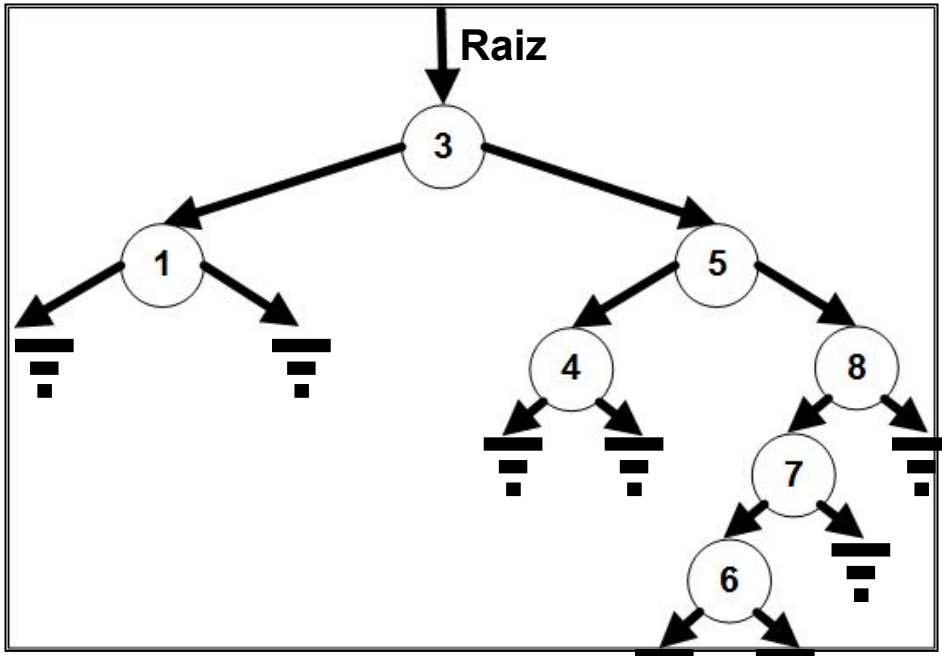


Algoritmo de Remoção em Java

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    boolean pesquisar(int x) {   }
    void remover(int x) {  }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
}
```

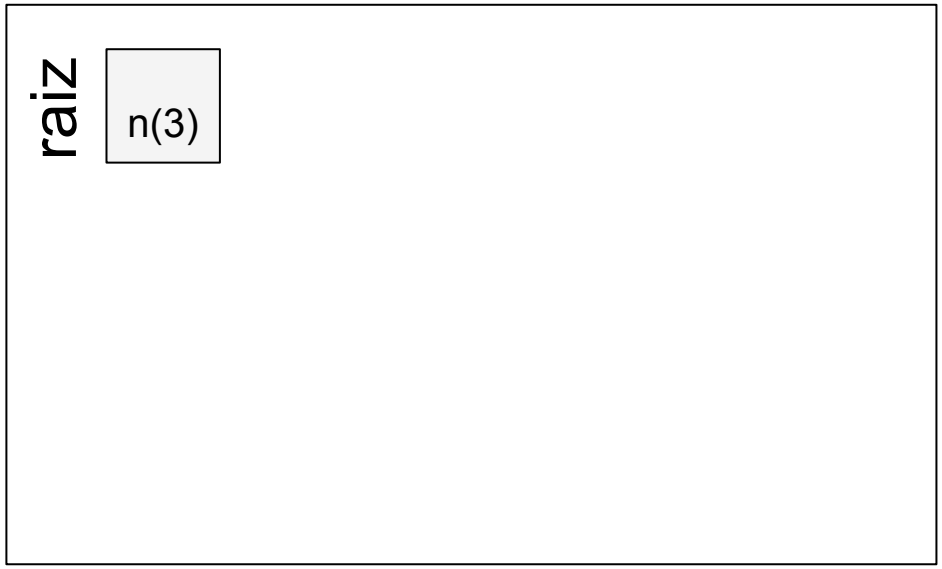


Voltando com o 2 antes
de fazer outra remoção

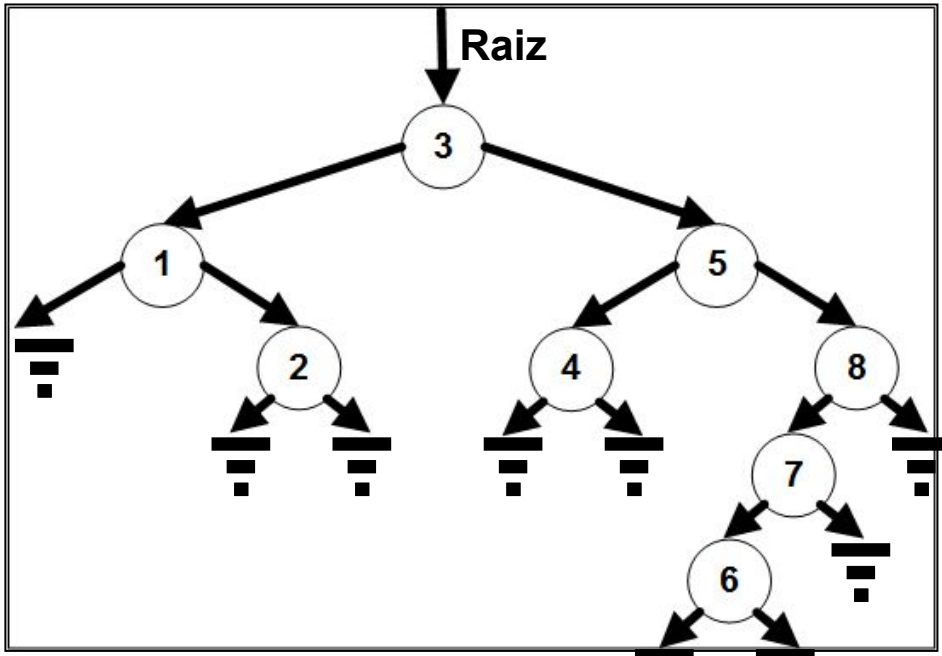


Algoritmo de Remoção em Java

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    boolean pesquisar(int x) {   }
    void remover(int x) {  }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
}
```

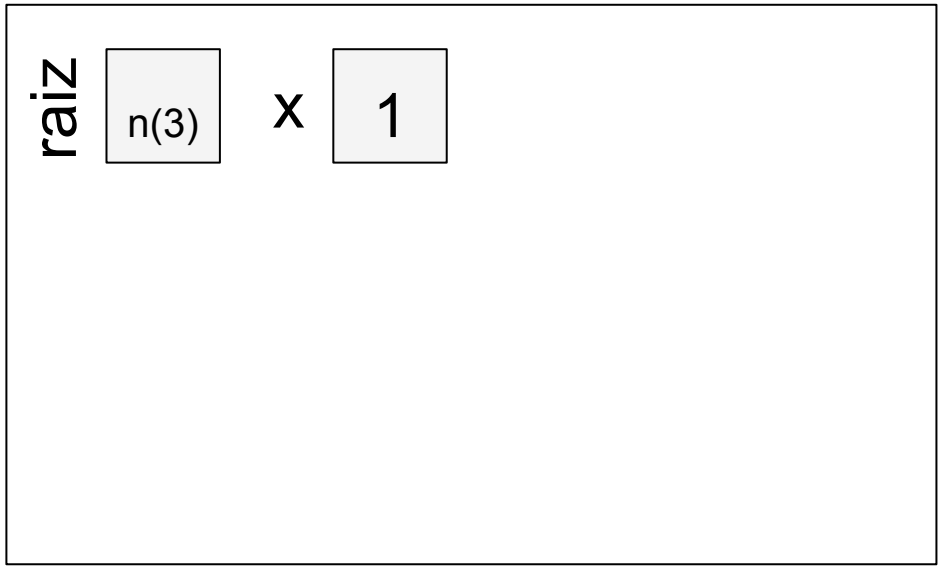


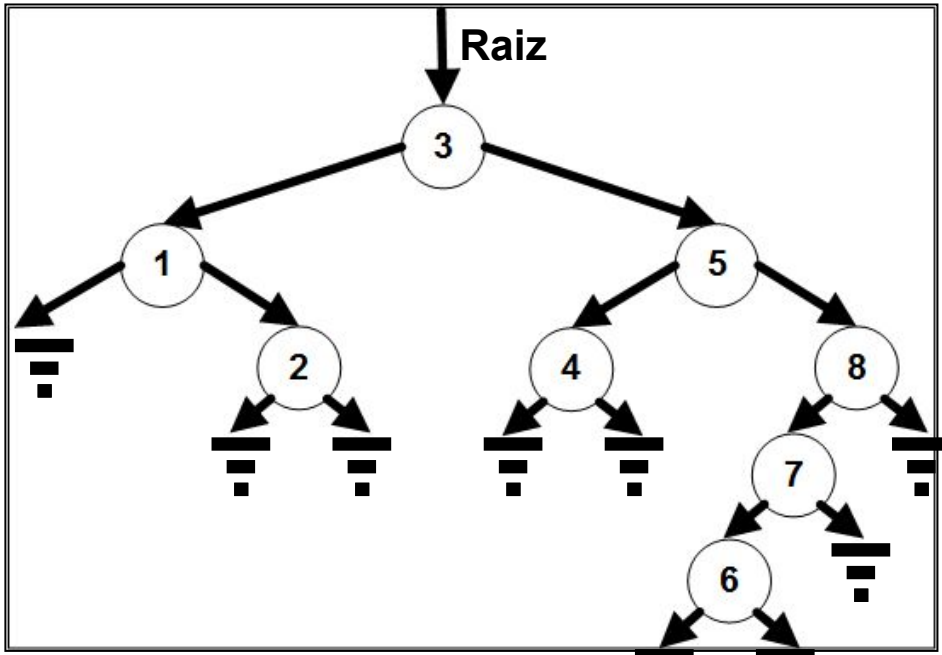
Vamos remover o 1 (tem um filho) de nossa árvore



Algoritmo de Remoção em Java

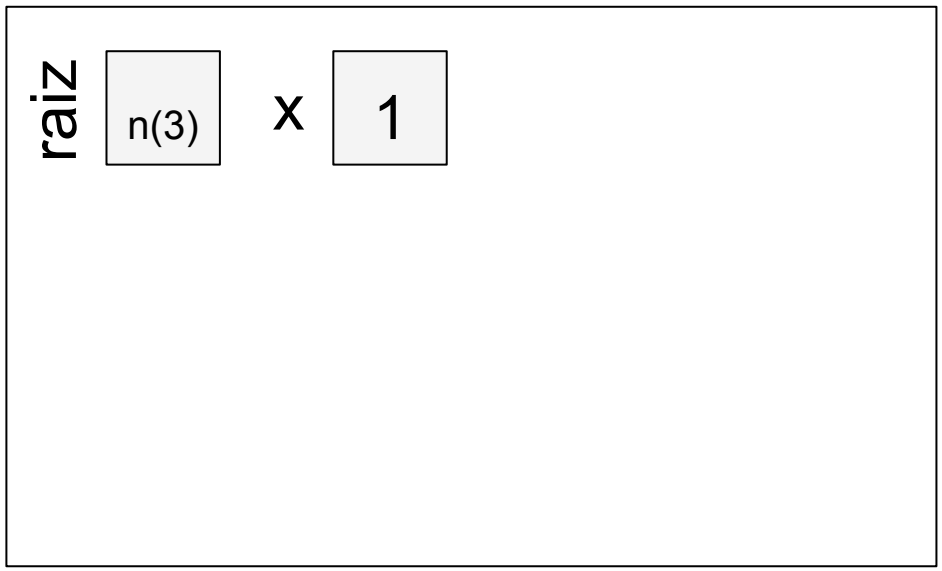
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

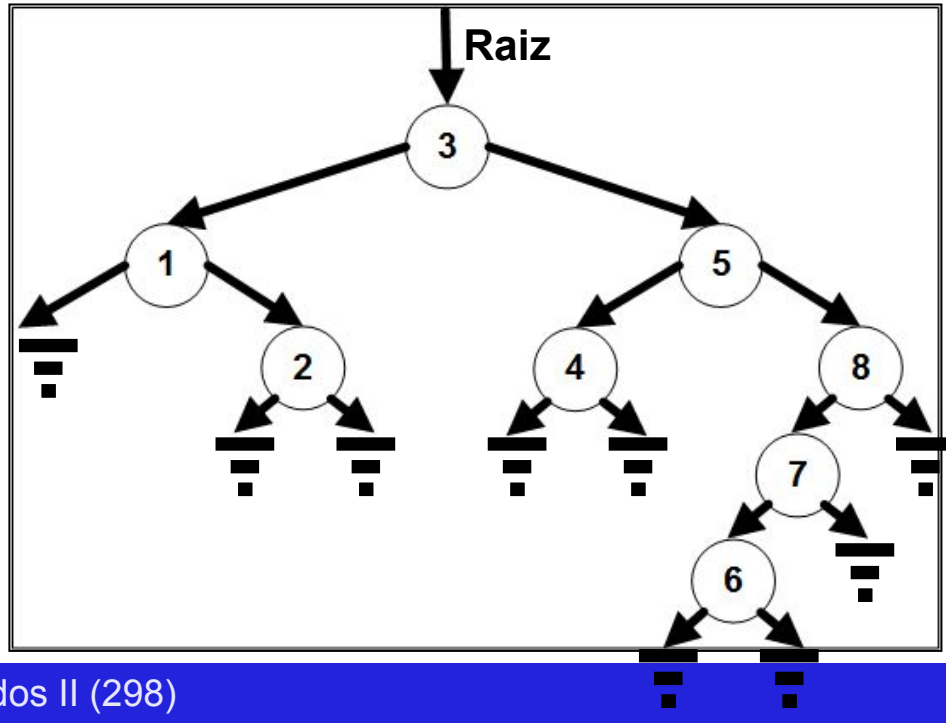




Algoritmo de Remoção em Java

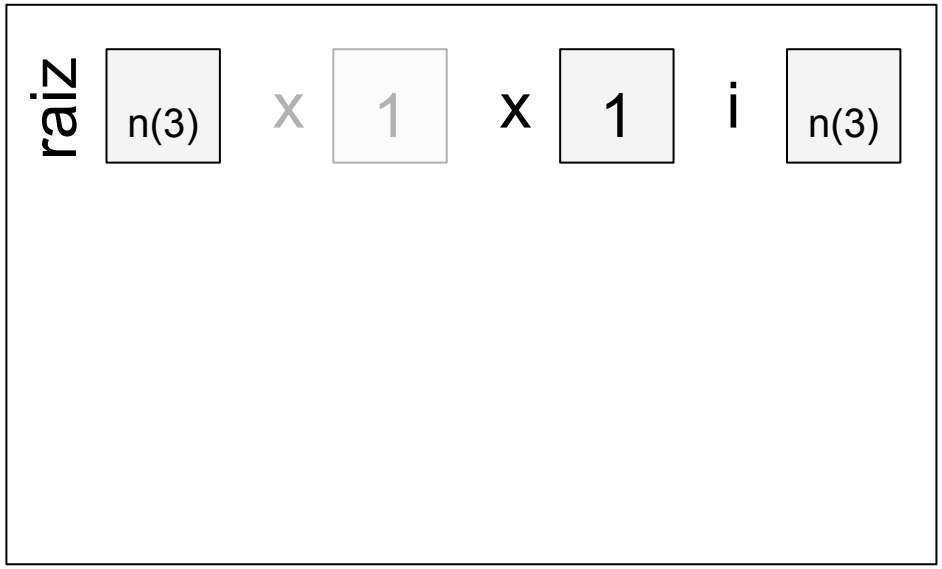
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

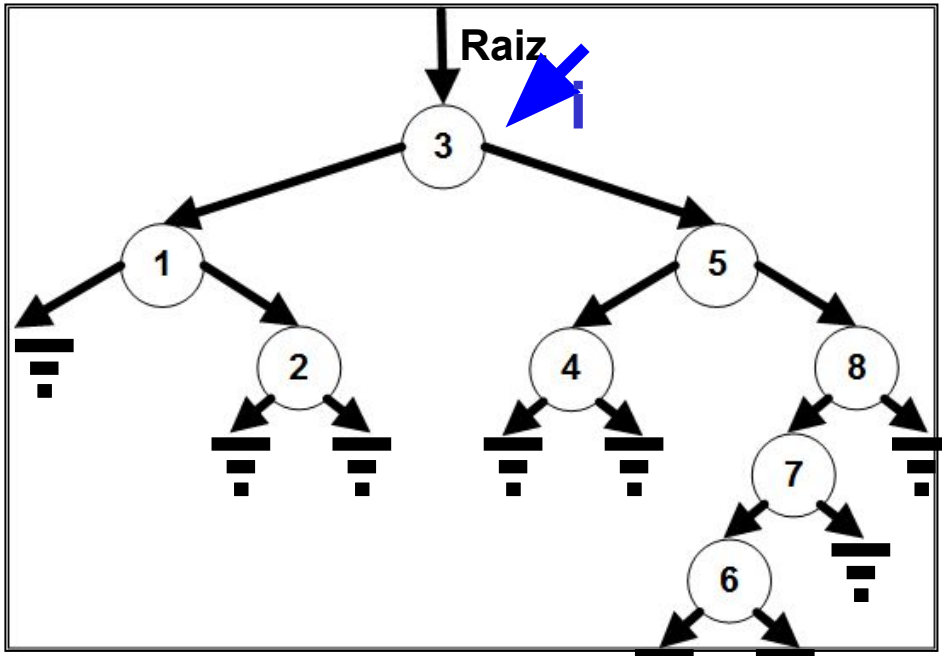




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

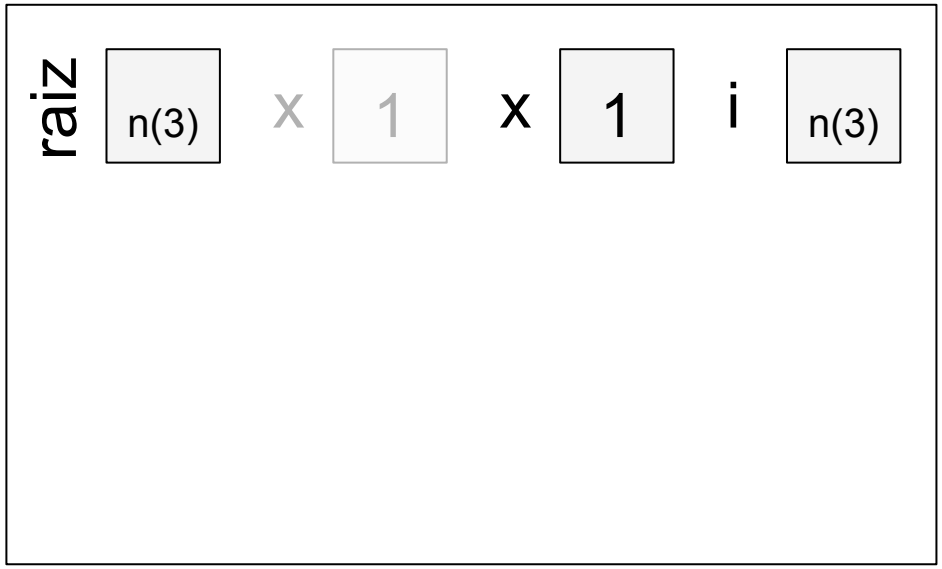


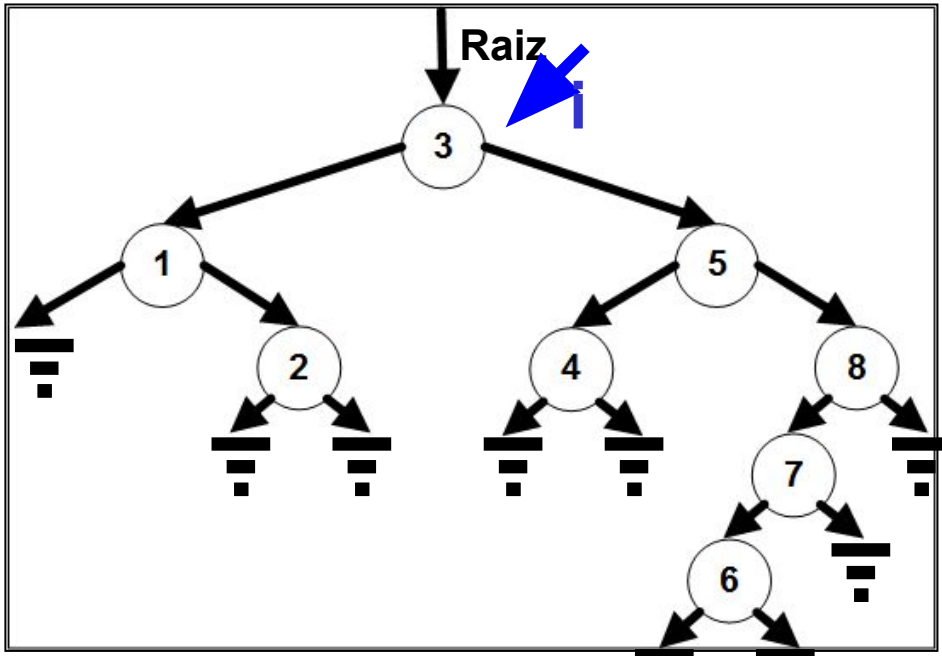


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {      throw new("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;
}

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;
}
```

false: n(3) == null

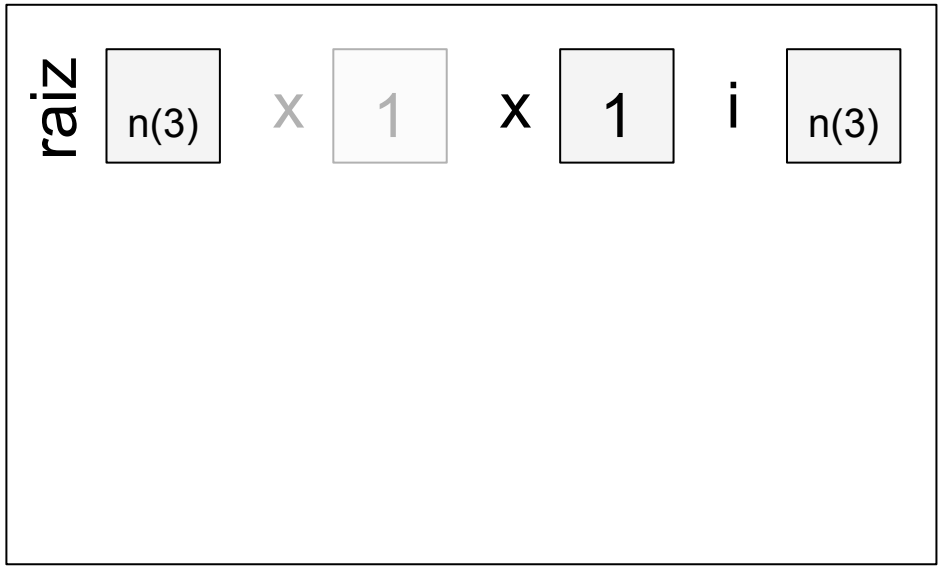


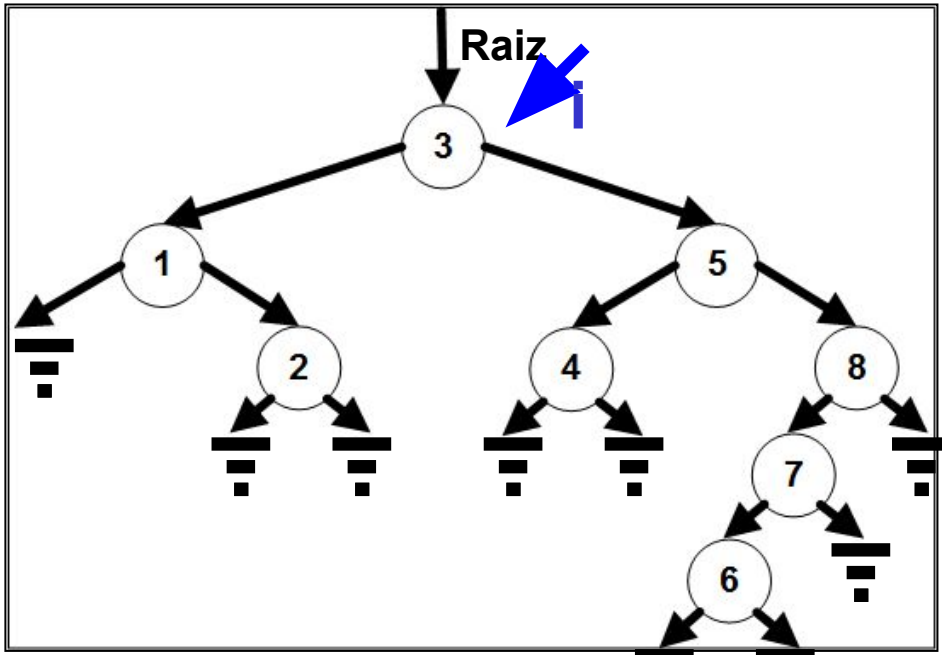


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

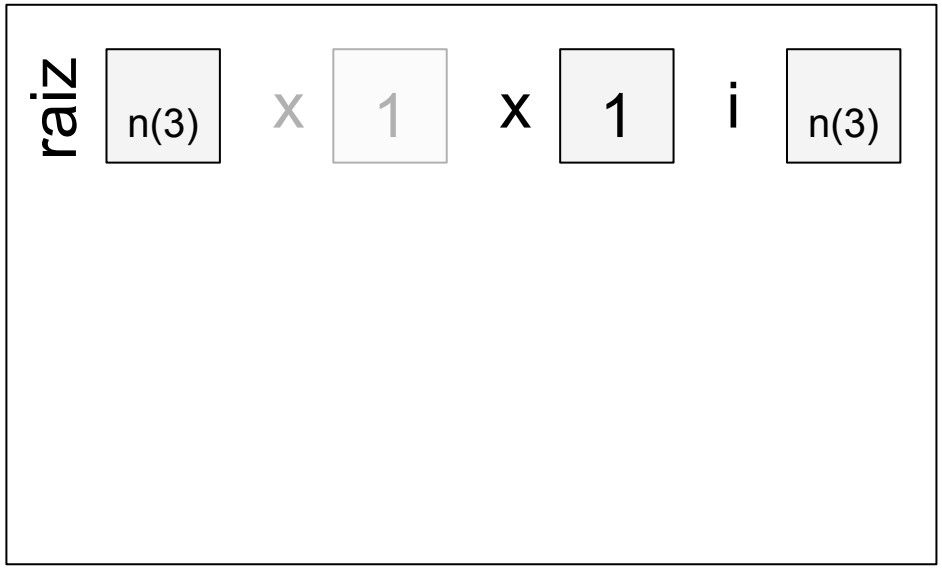
true: 1 < 3

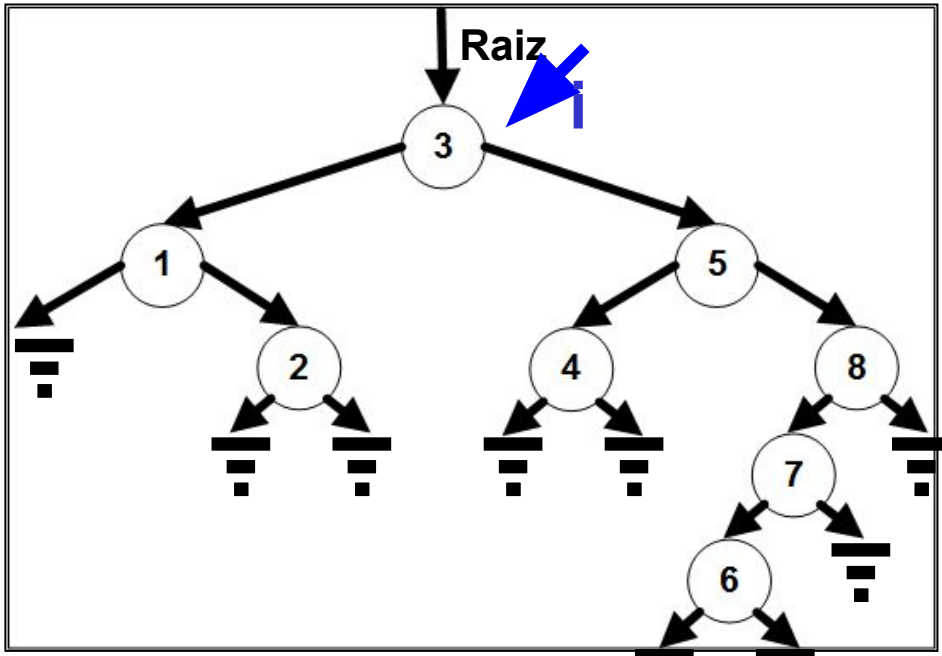




Algoritmo de Remoção em Java

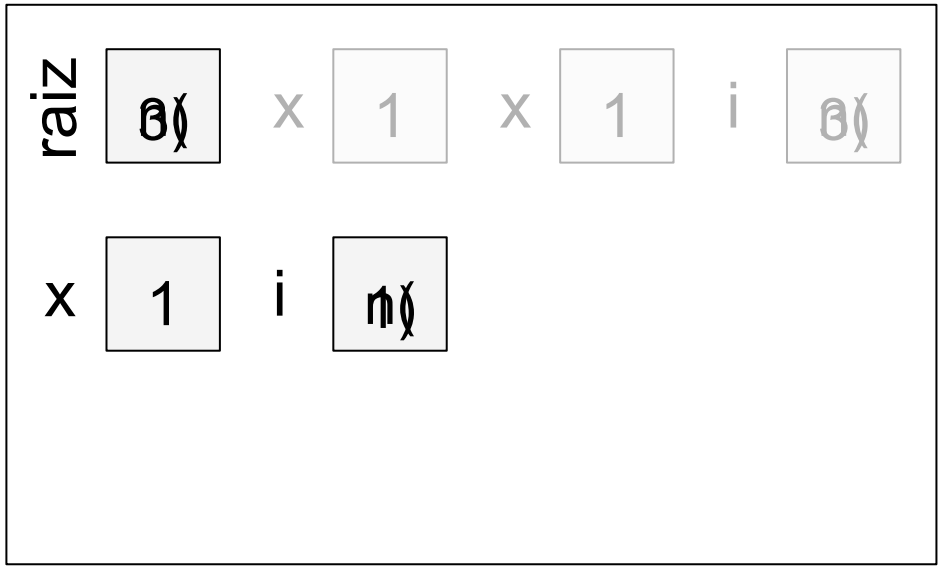
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

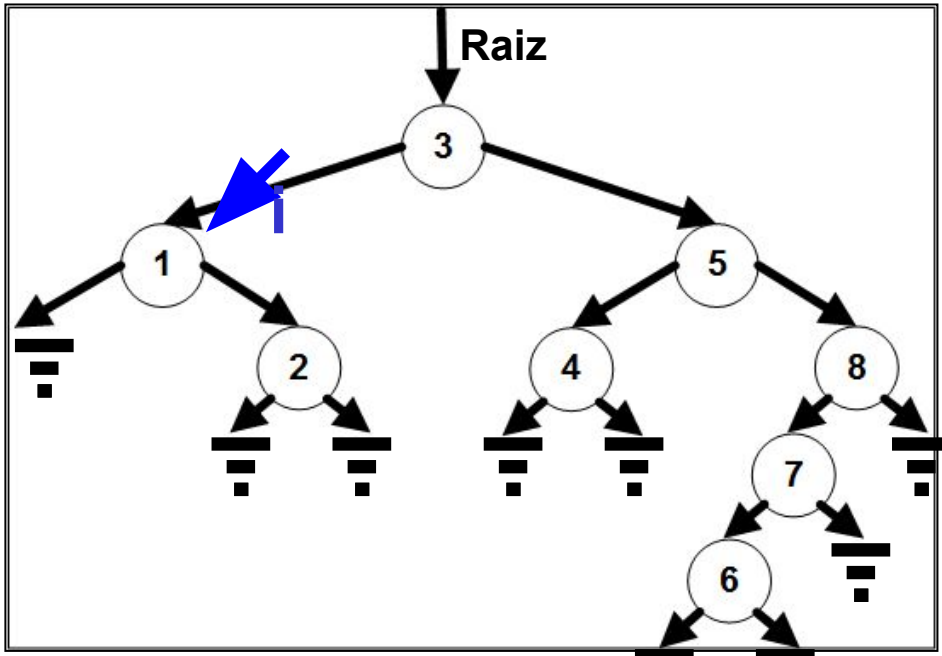




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

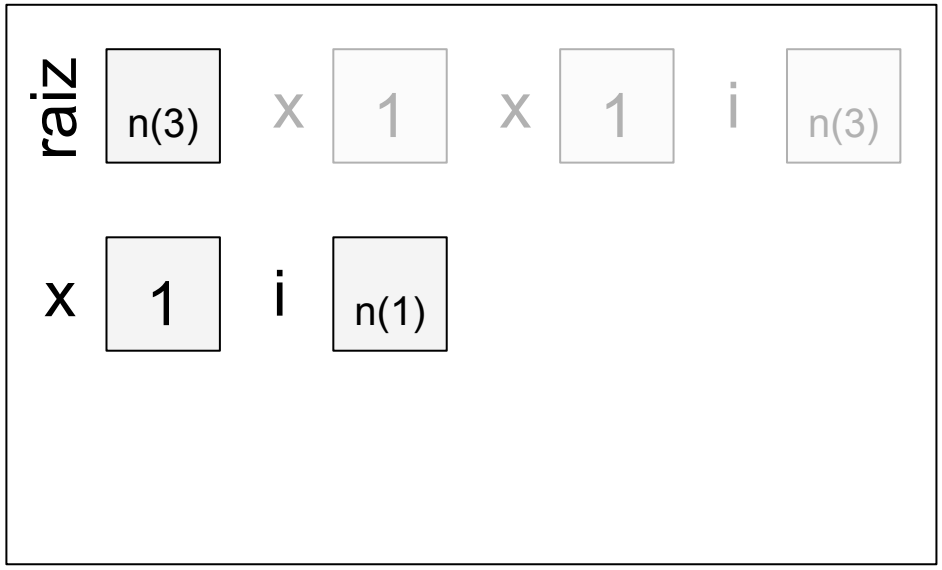


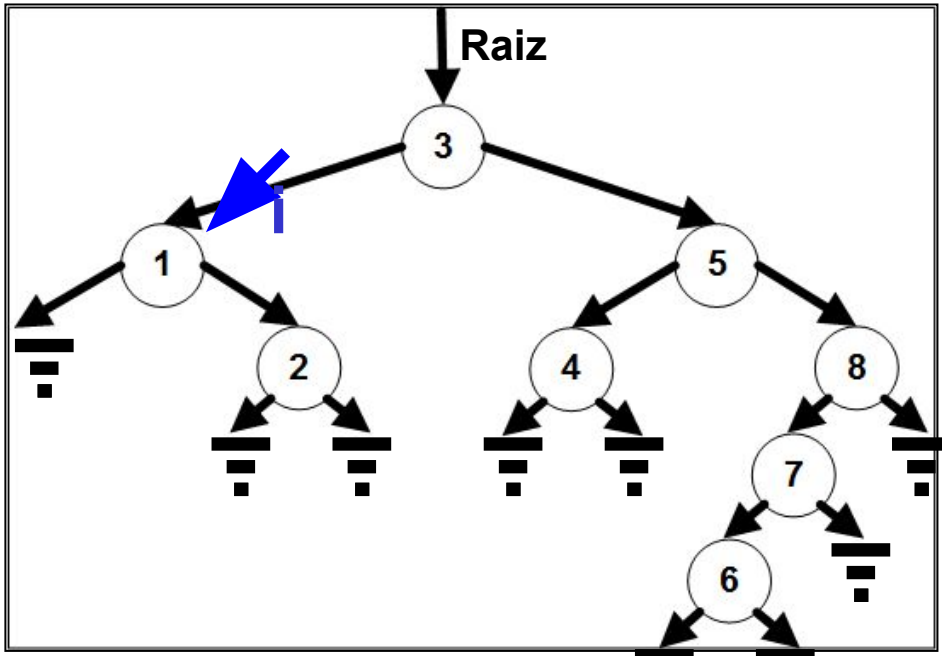


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {      throw new("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;
}

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;
}
```

false: n(1) == null

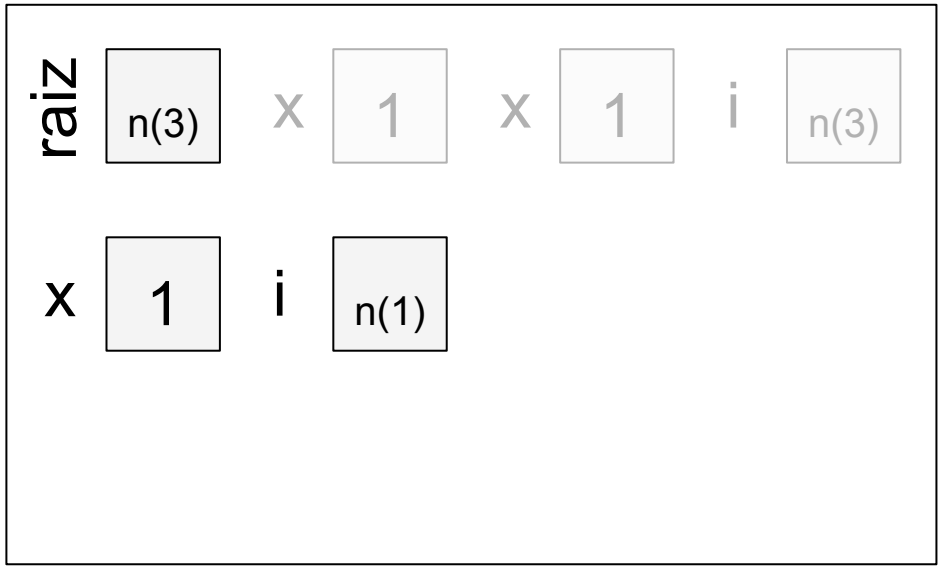


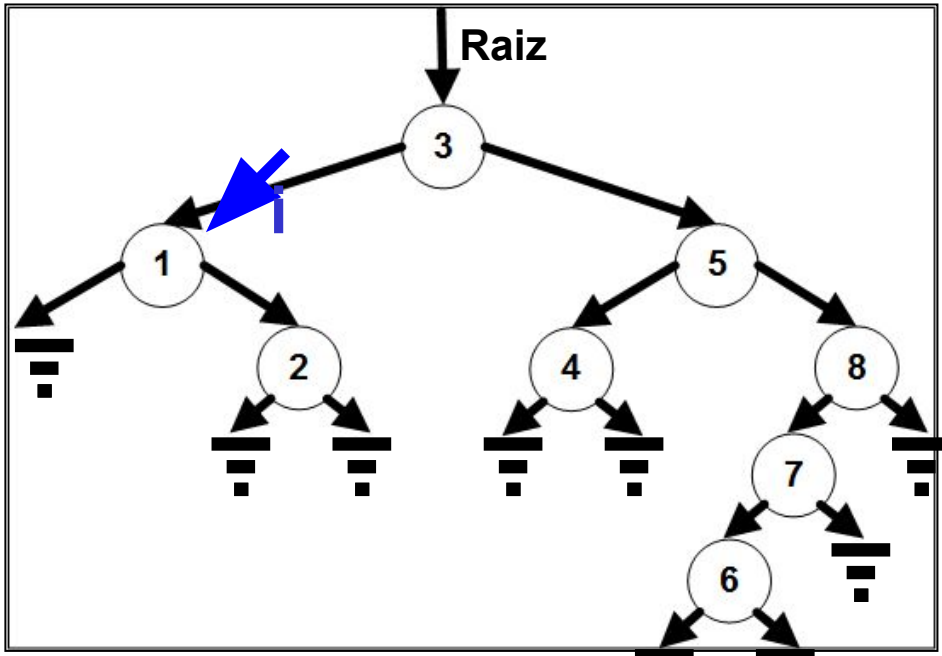


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: 1 < 1

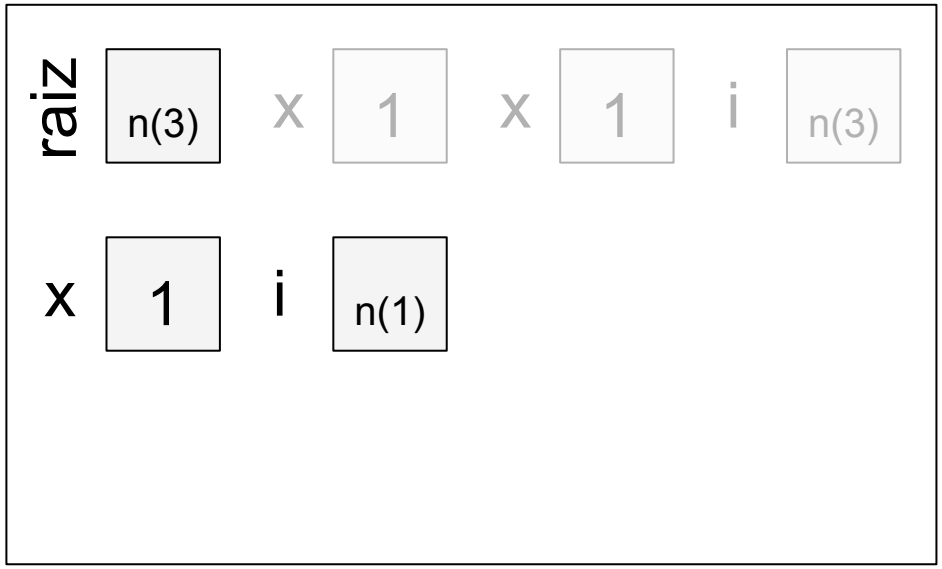


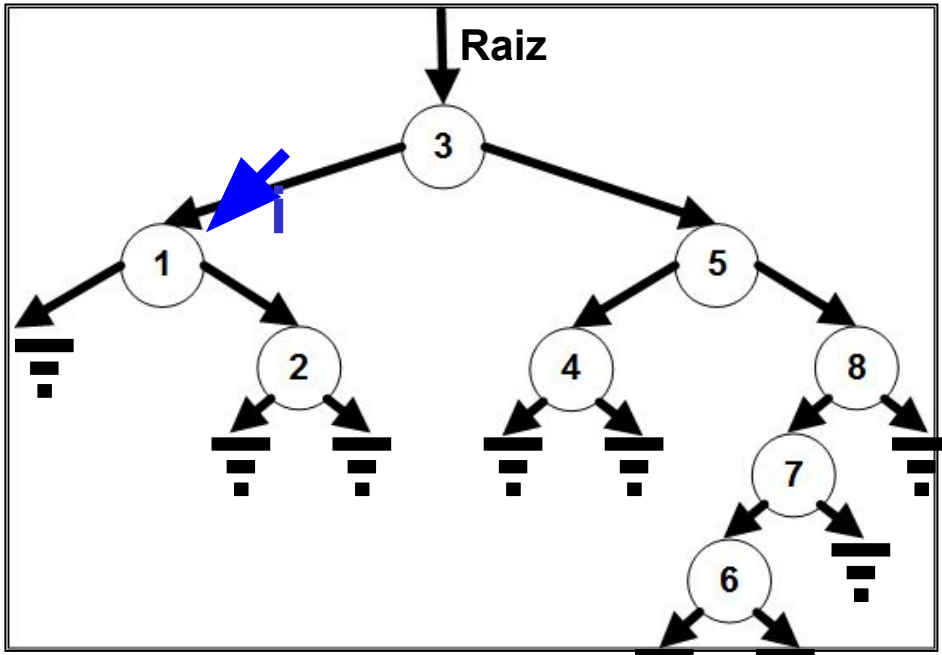


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: 1 > 1

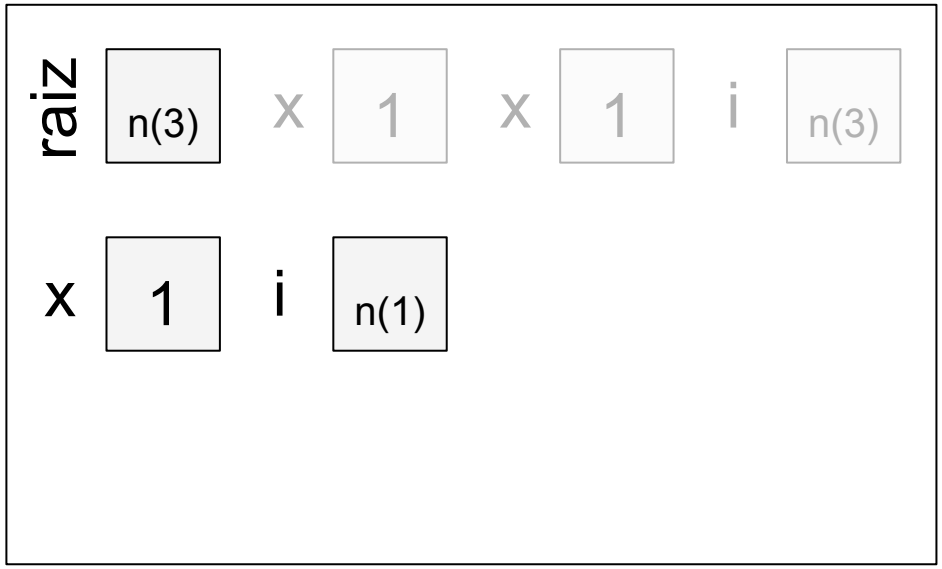


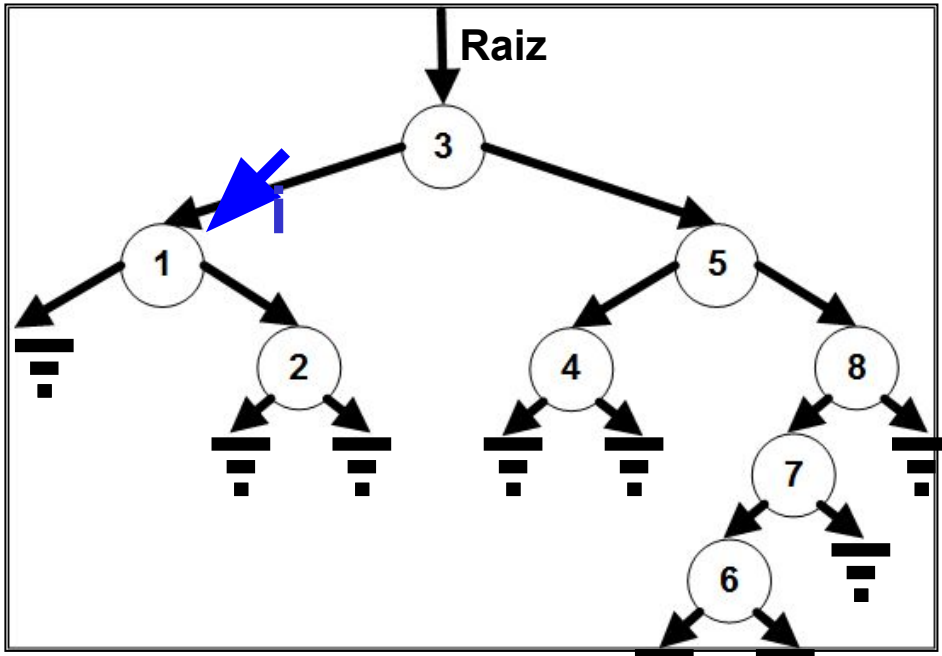


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: n(2) == null

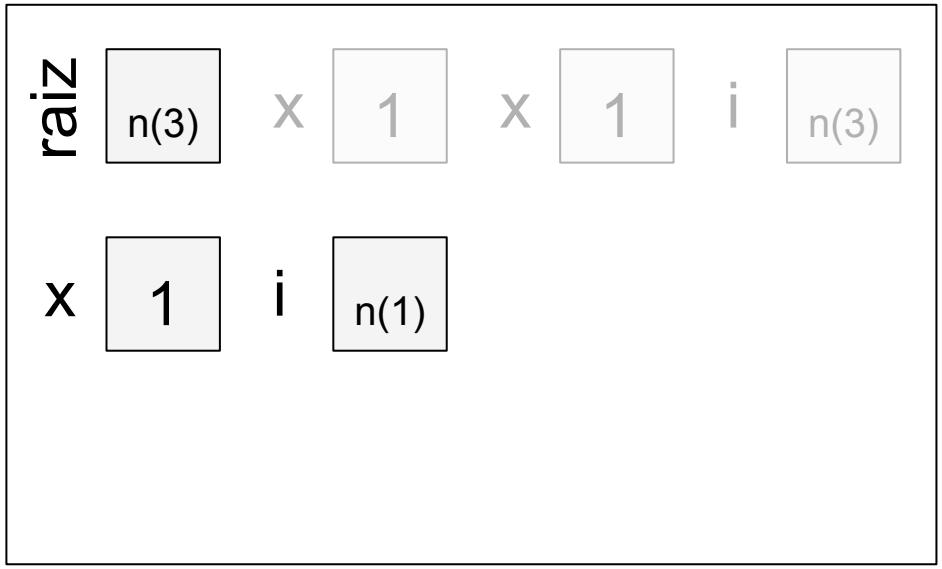


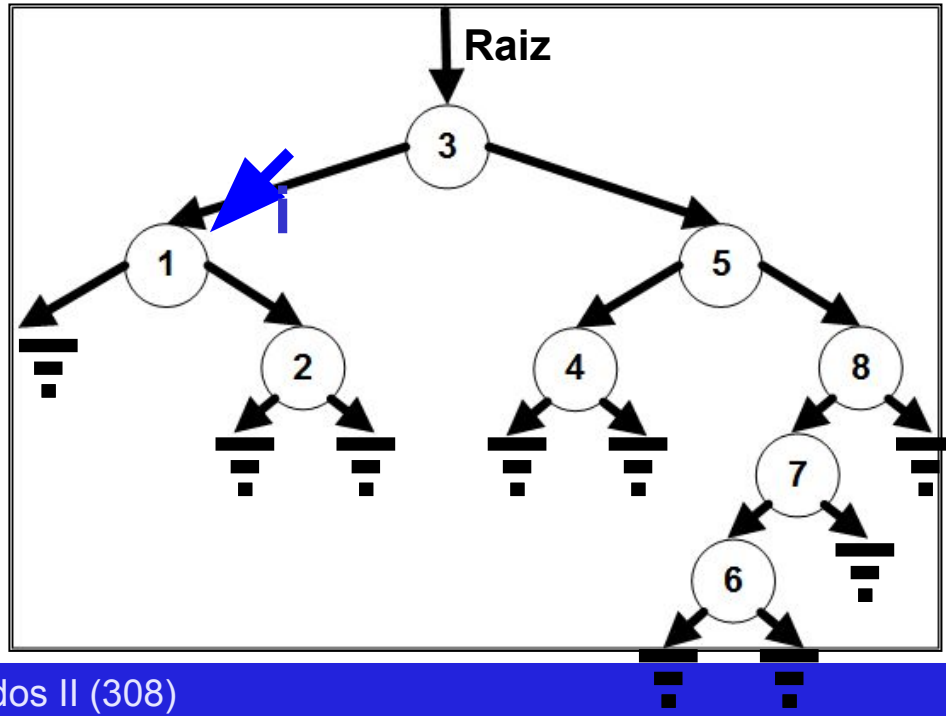


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

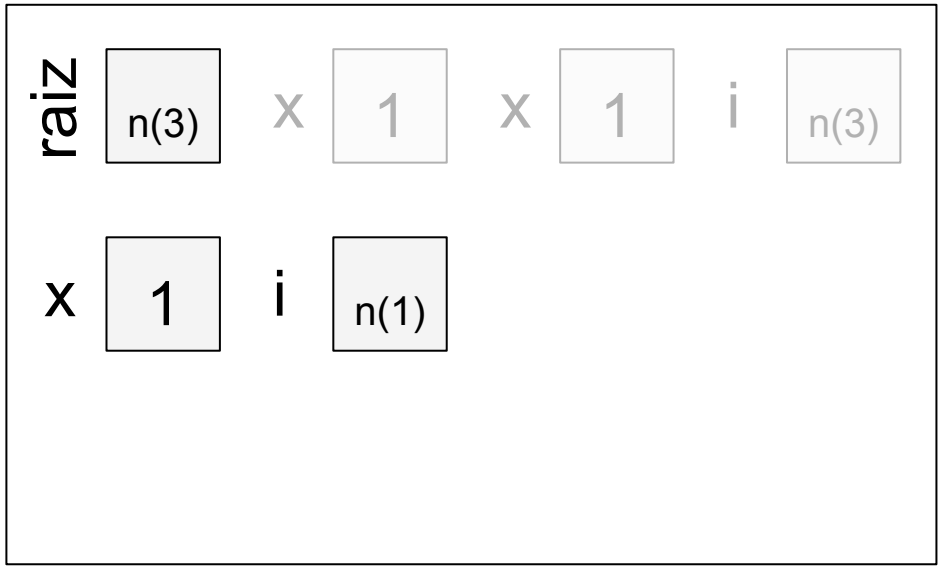
true: null == null

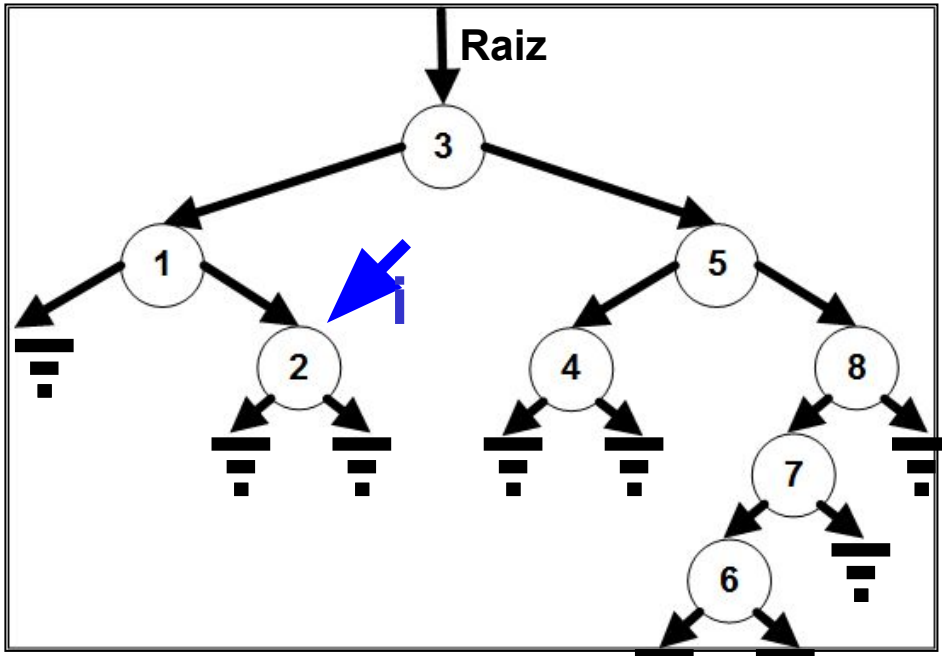




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



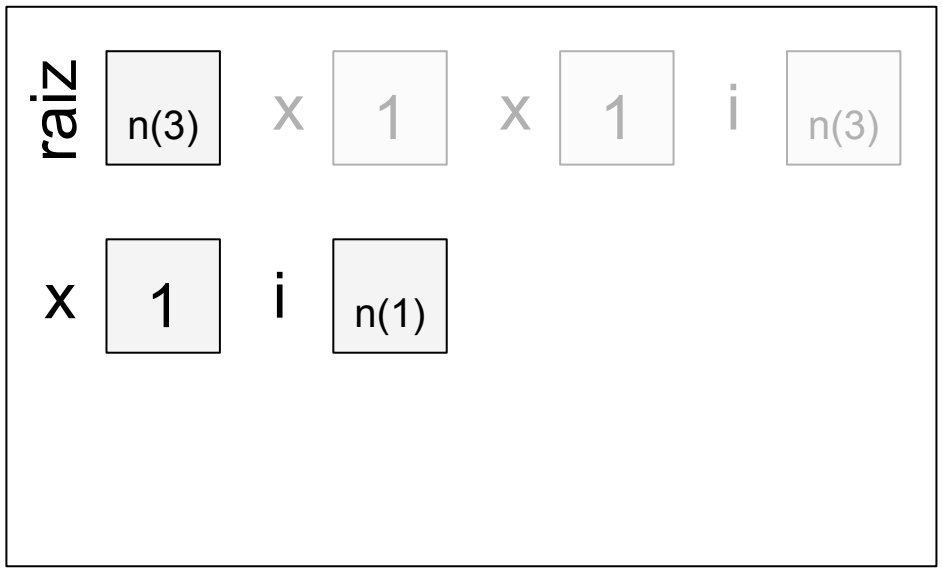


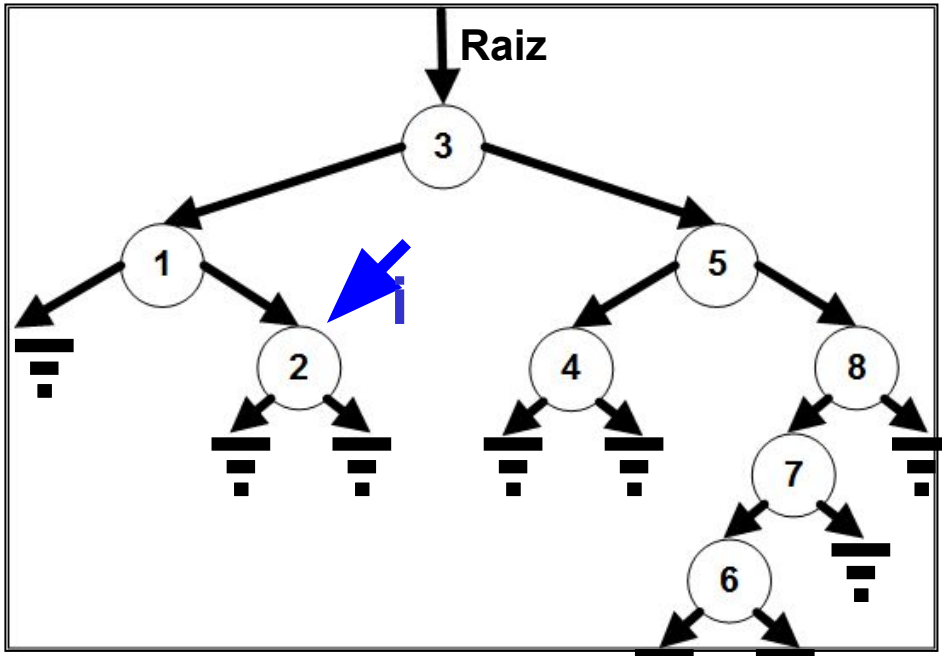
Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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)

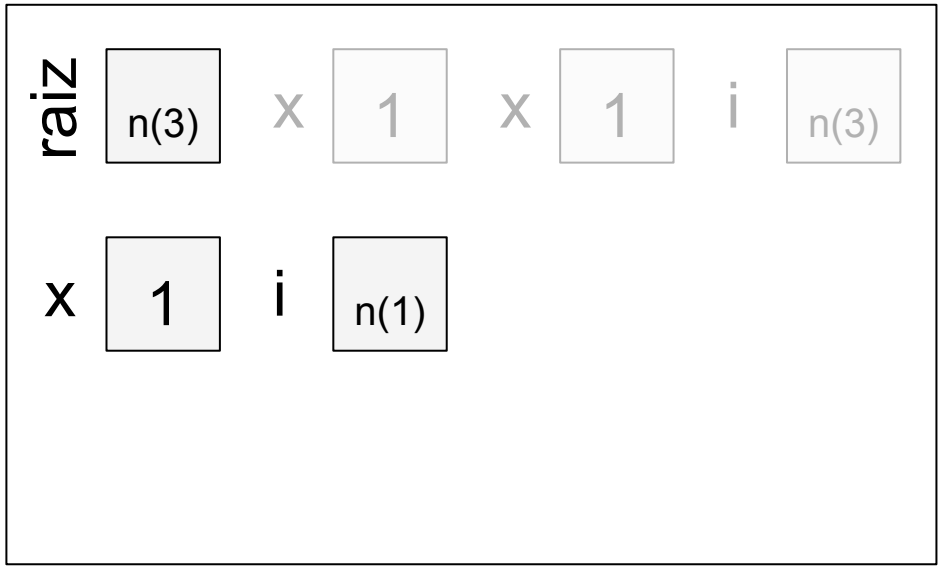
```
No anterior(No i, No j) {
   if (j.dir != null)  j.dir = anterior(i, j.dir);
   else {              i.elemento = j.elemento; j = j.esq; }
   return j;
}
```

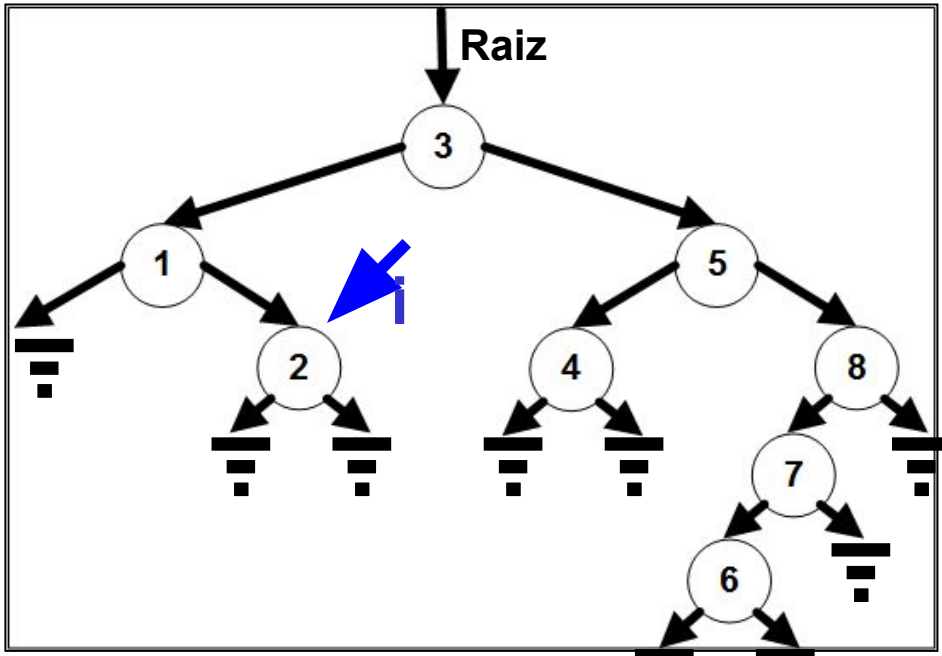




Algoritmo de Remoção em Java

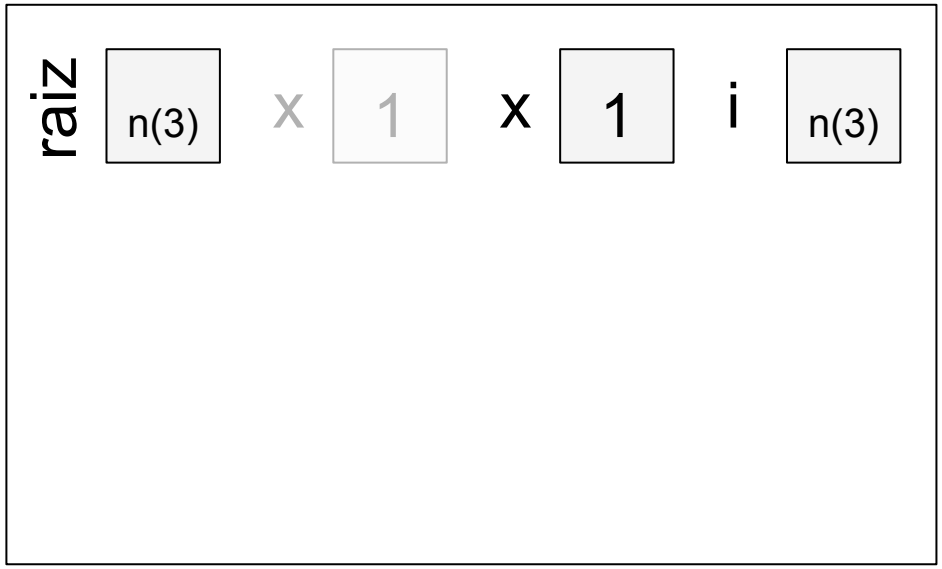
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

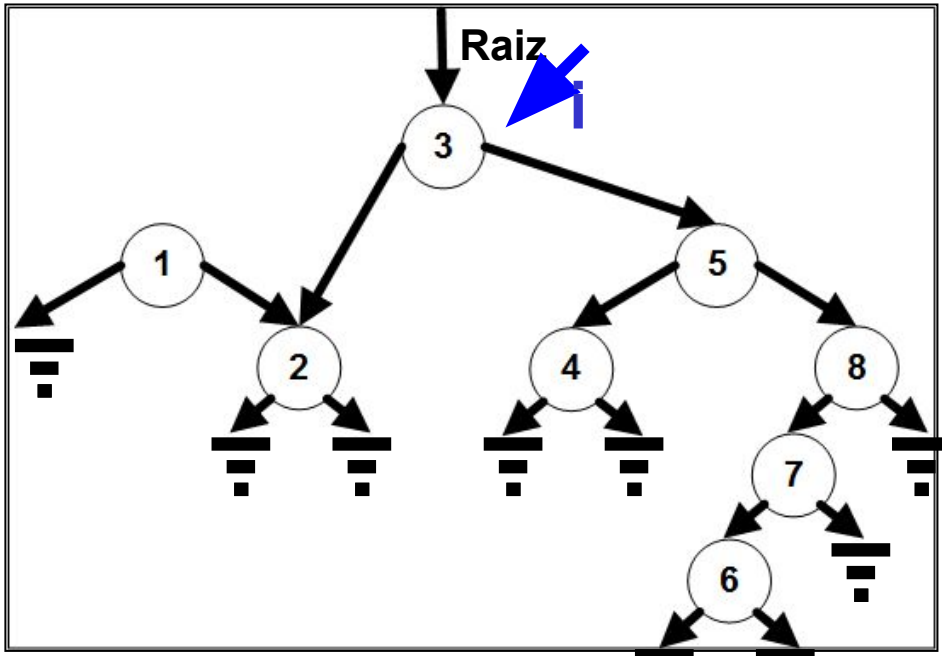




Algoritmo de Remoção em Java

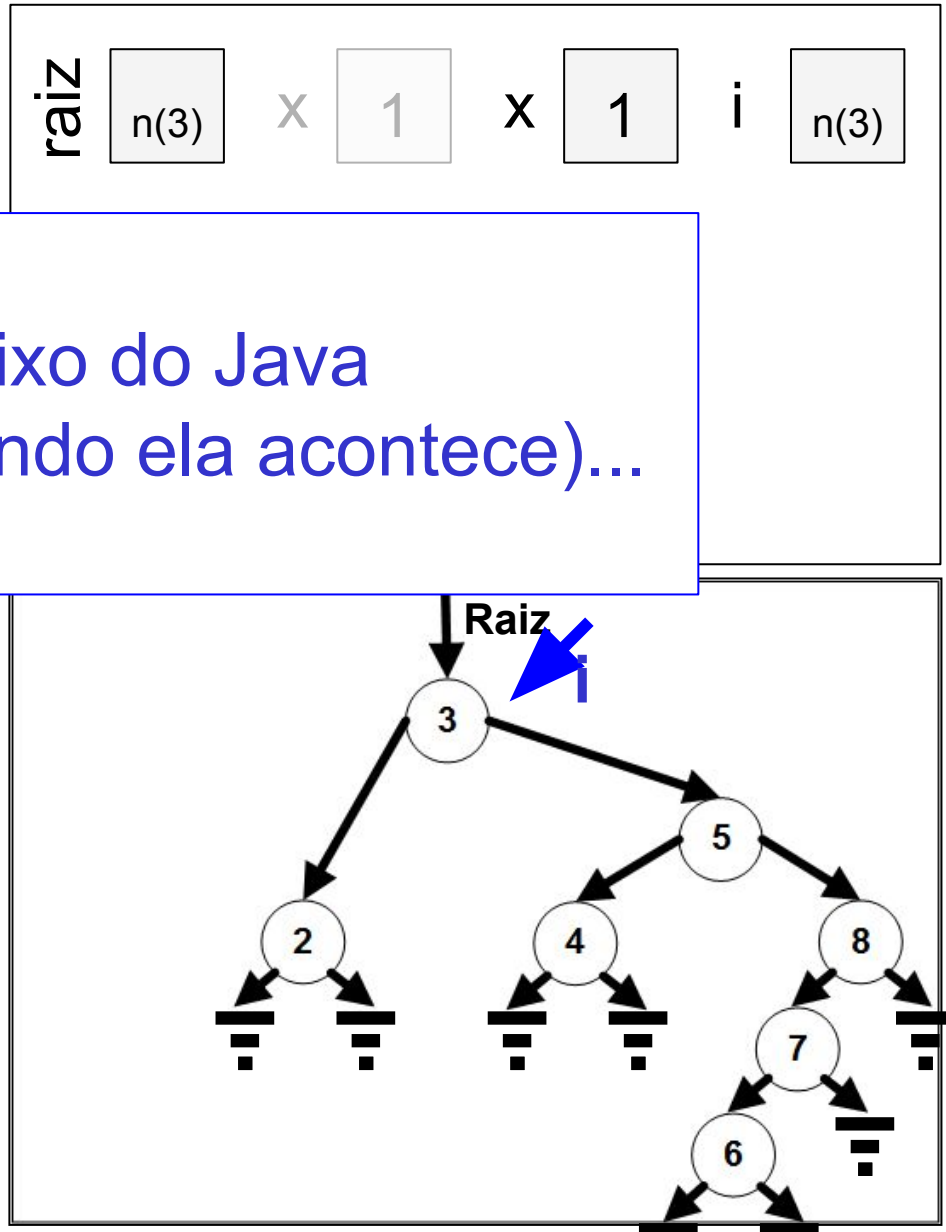
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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 de Remoção em Java

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

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

No remo
   if (i ==
   } else
   } else
   } 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;
}

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;
}
```

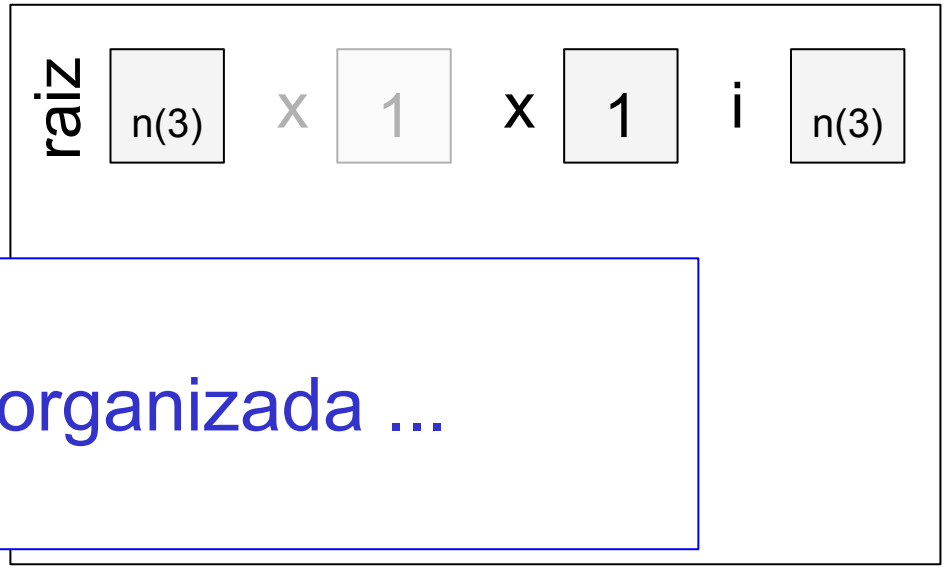


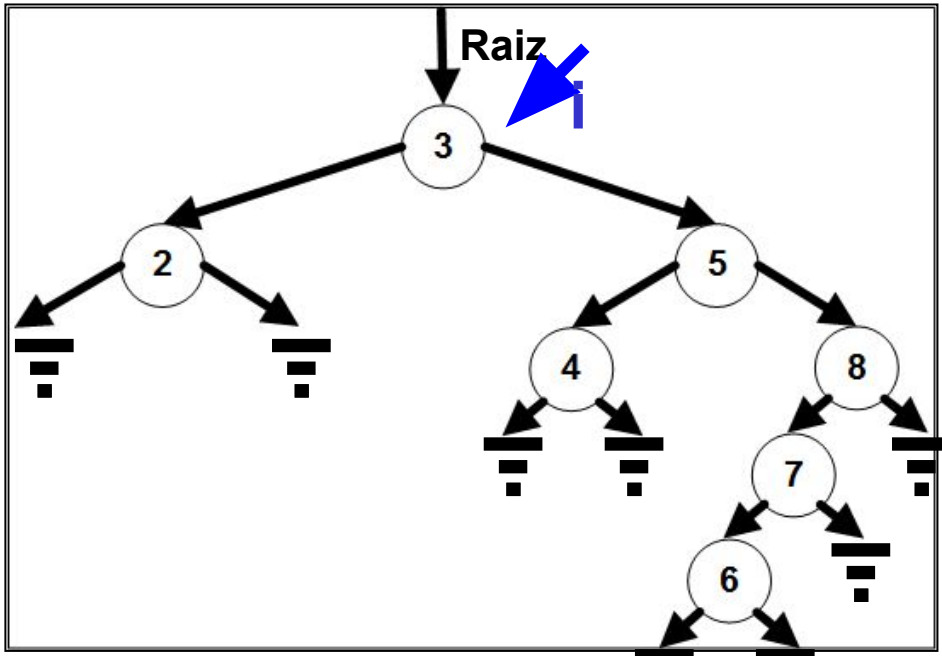
Após a coleta de lixo do Java
(que não controlamos quando ela acontece)...

Algoritmo de Remoção em Java

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

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

No remo
   if (i ==
   } else
   } else
   } 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;
}

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 ...



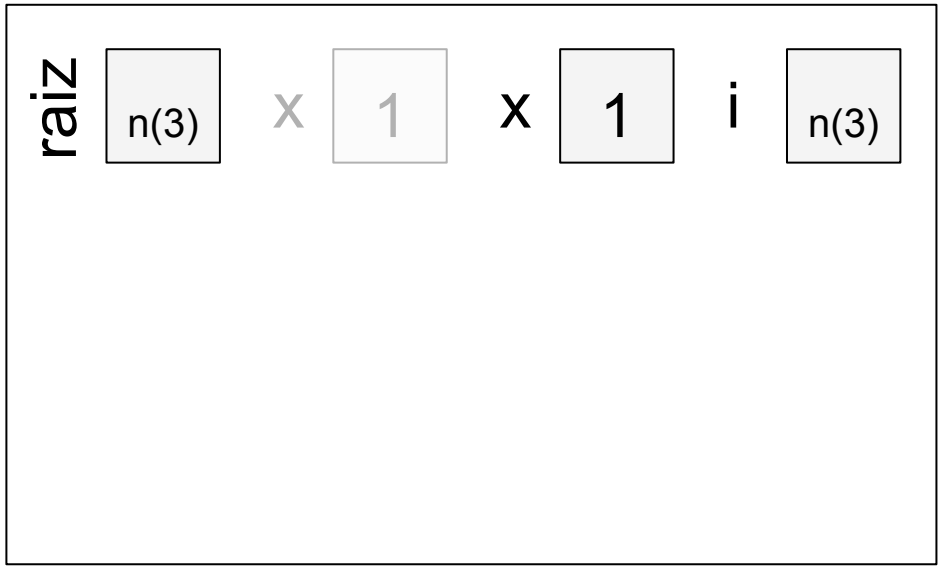


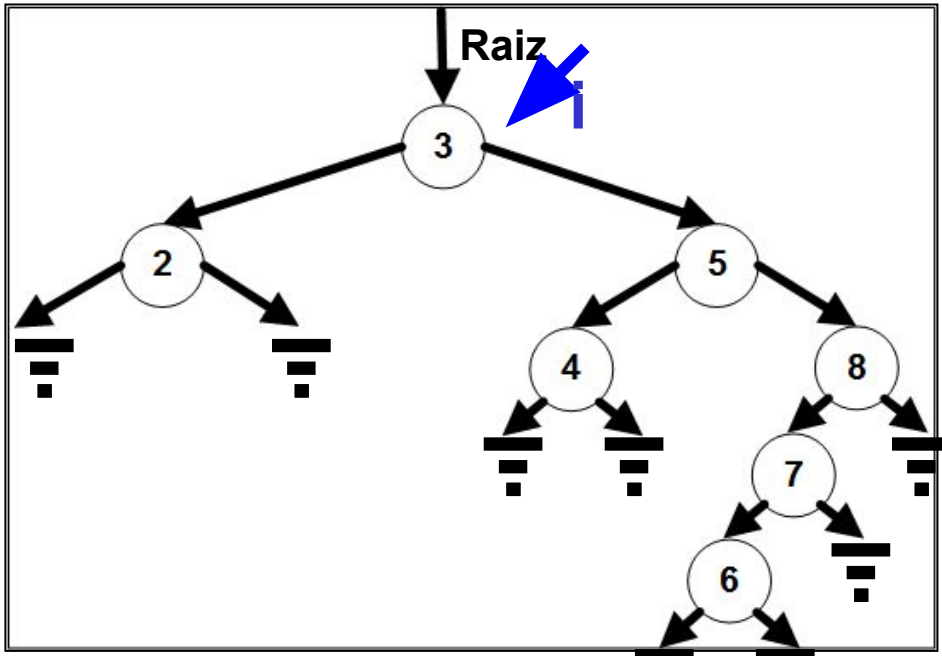
Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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)

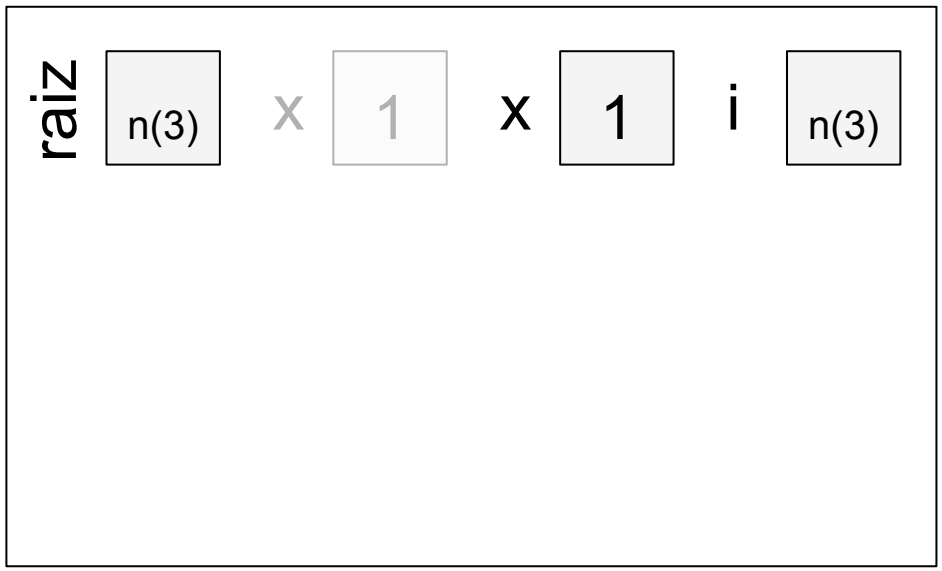
```
No anterior(No i, No j) {
   if (j.dir != null)  j.dir = anterior(i, j.dir);
   else {              i.elemento = j.elemento; j = j.esq; }
   return j;
}
```

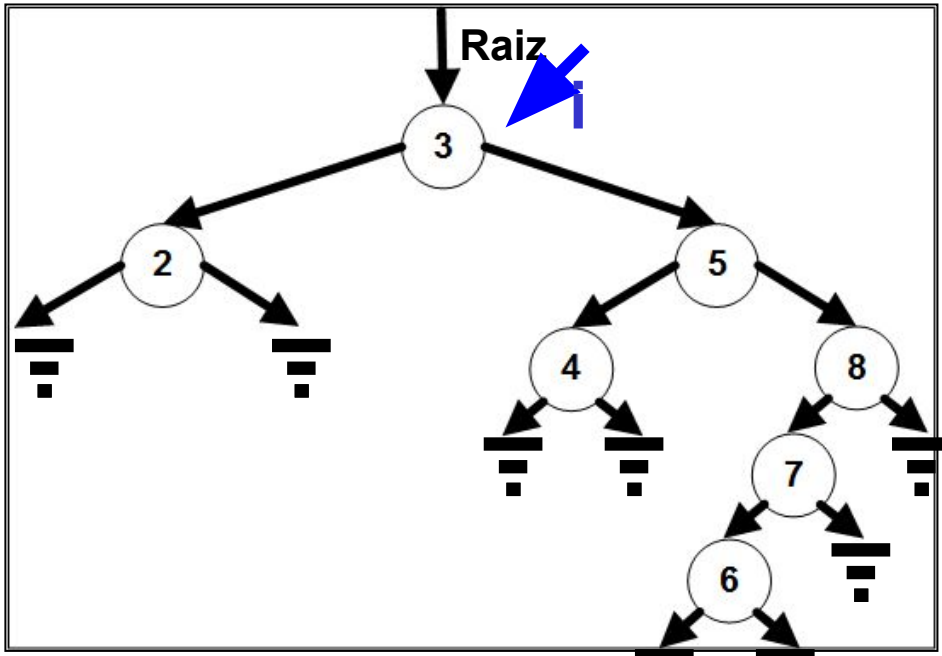




Algoritmo de Remoção em Java

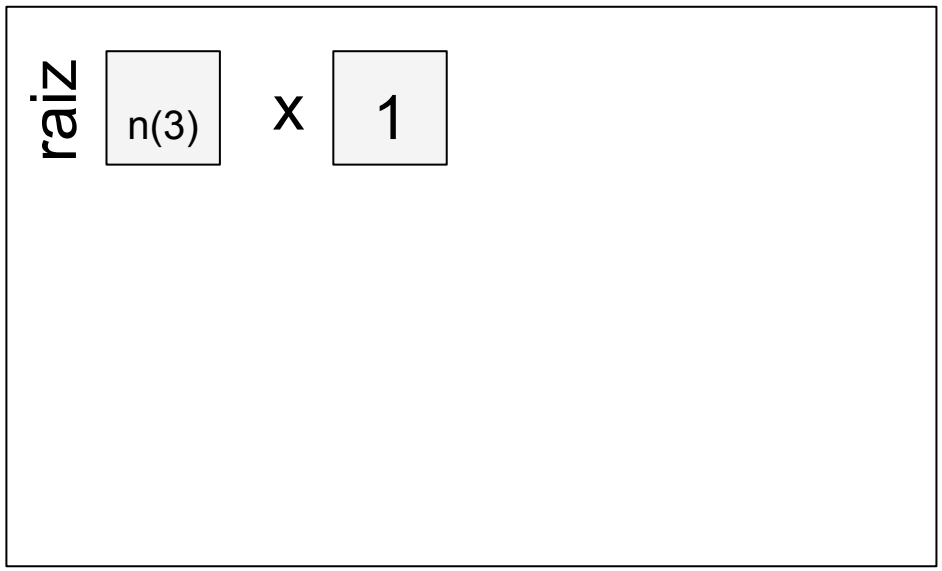
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

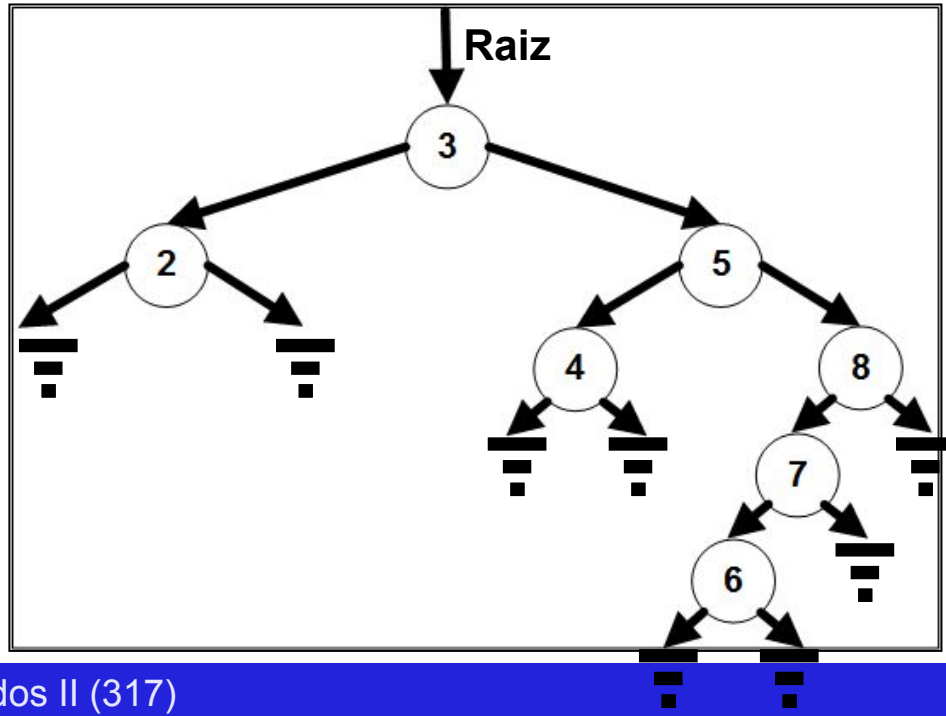




Algoritmo de Remoção em Java

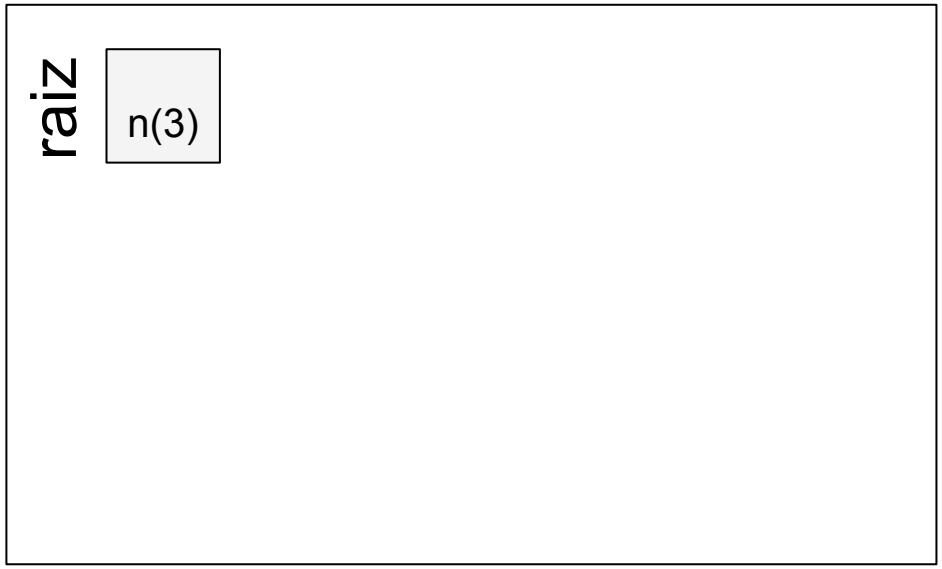
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

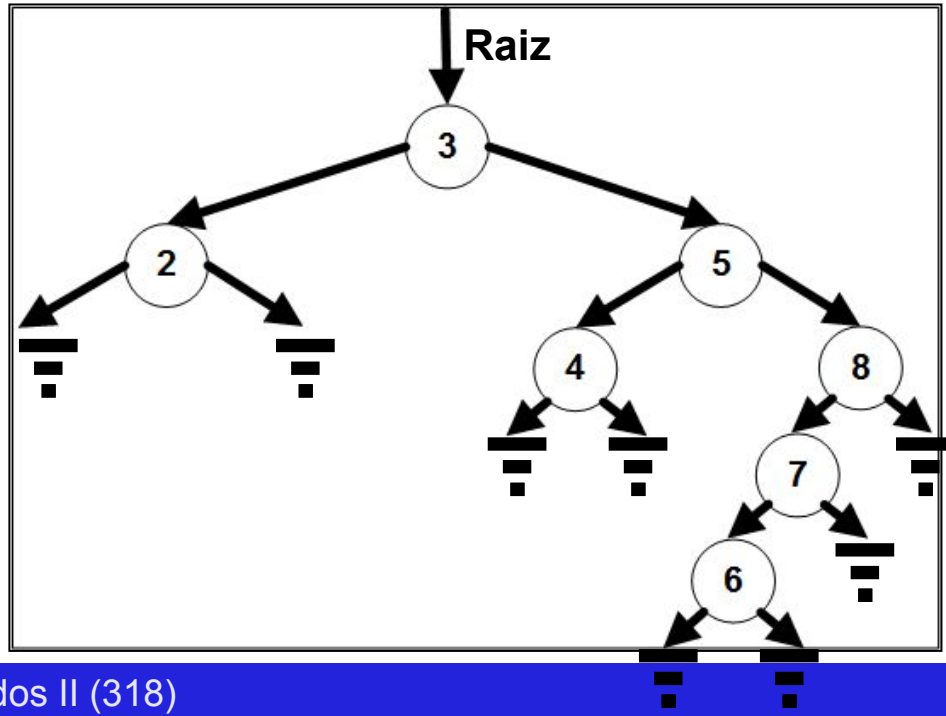




Algoritmo de Remoção em Java

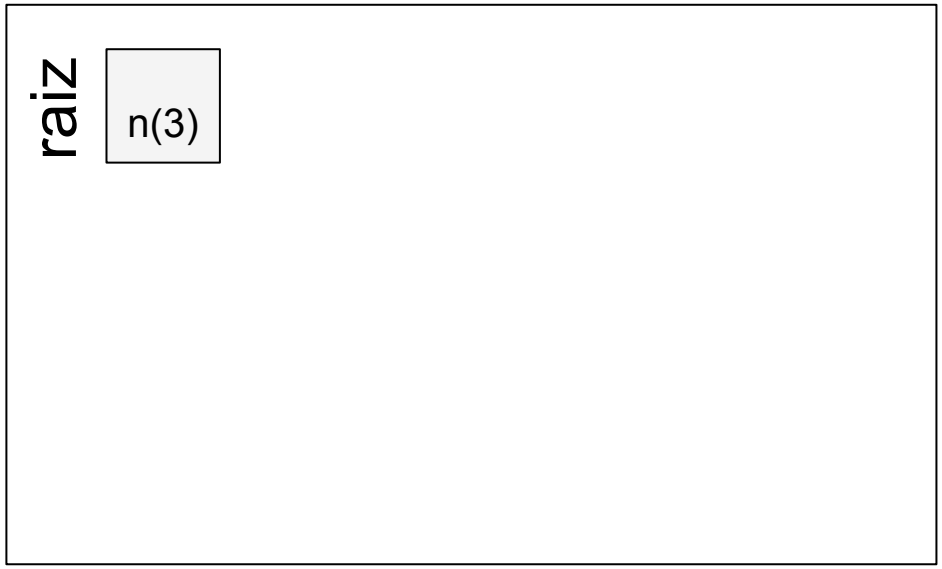
```
//remover(1), um filho

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



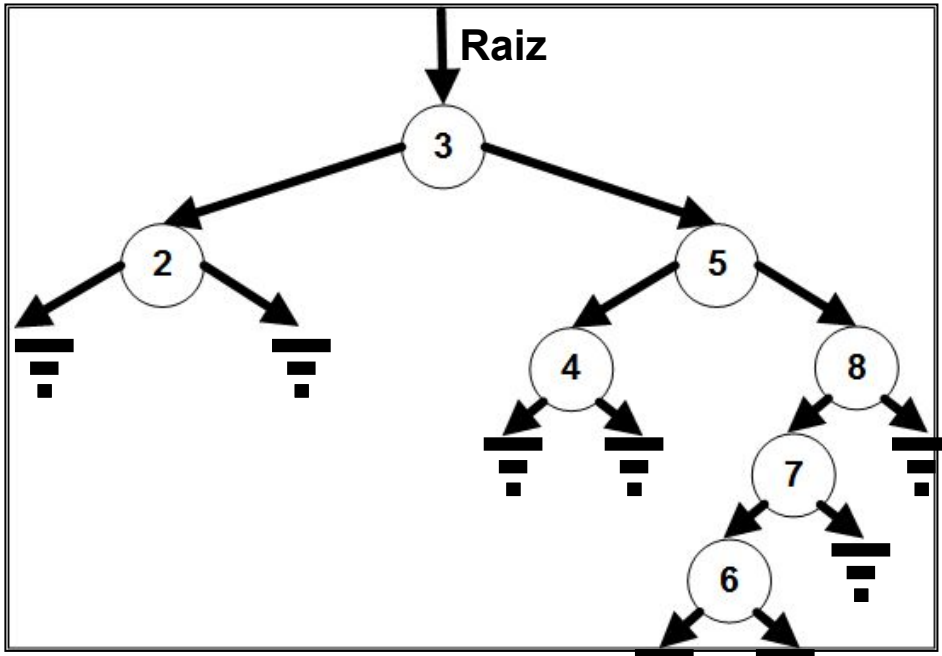


Algoritmo de Remoção em Java

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    boolean pesquisar(int x) {   }
    void remover(int x) {  }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
}
```

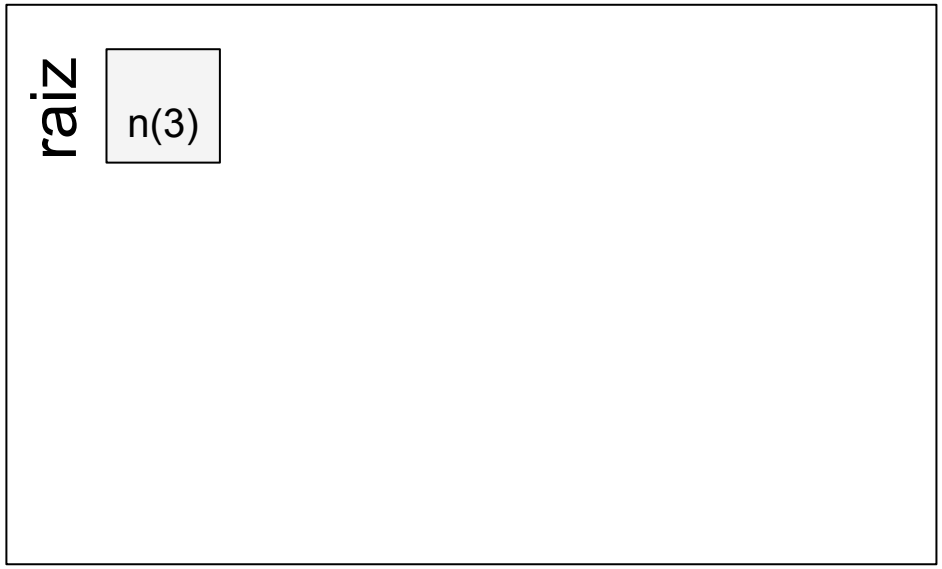


Voltando com o 1 antes
de fazer outra remoção

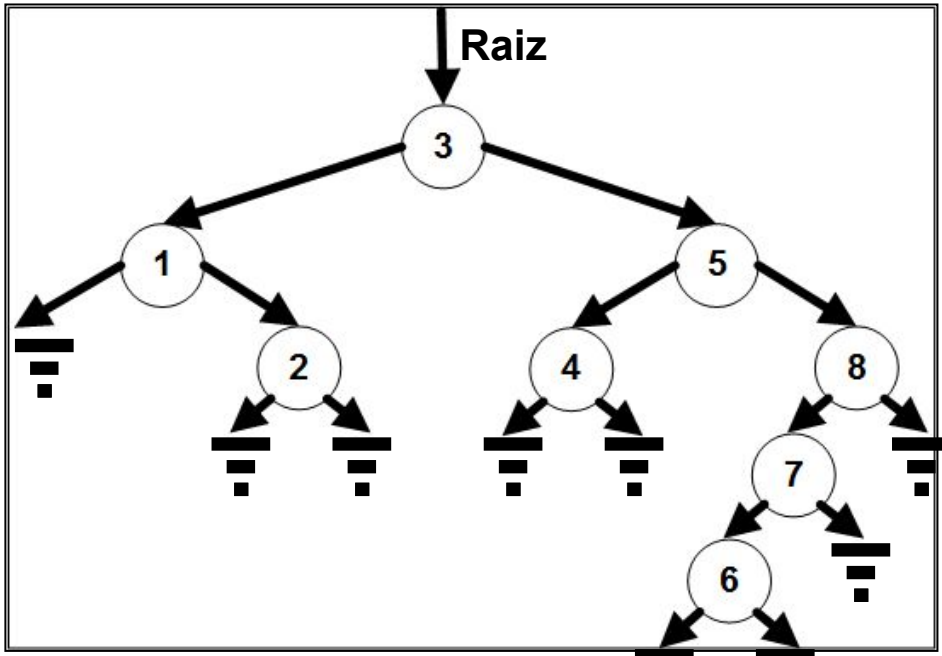


Algoritmo de Remoção em Java

```
class ArvoreBinaria {
    No raiz;
    ArvoreBinaria() { raiz = null; }
    void inserir(int x) {  }
    boolean pesquisar(int x) {   }
    void remover(int x) {  }
    void caminharCentral() {  }
    void caminharPre() {  }
    void caminharPos() {  }
}
```



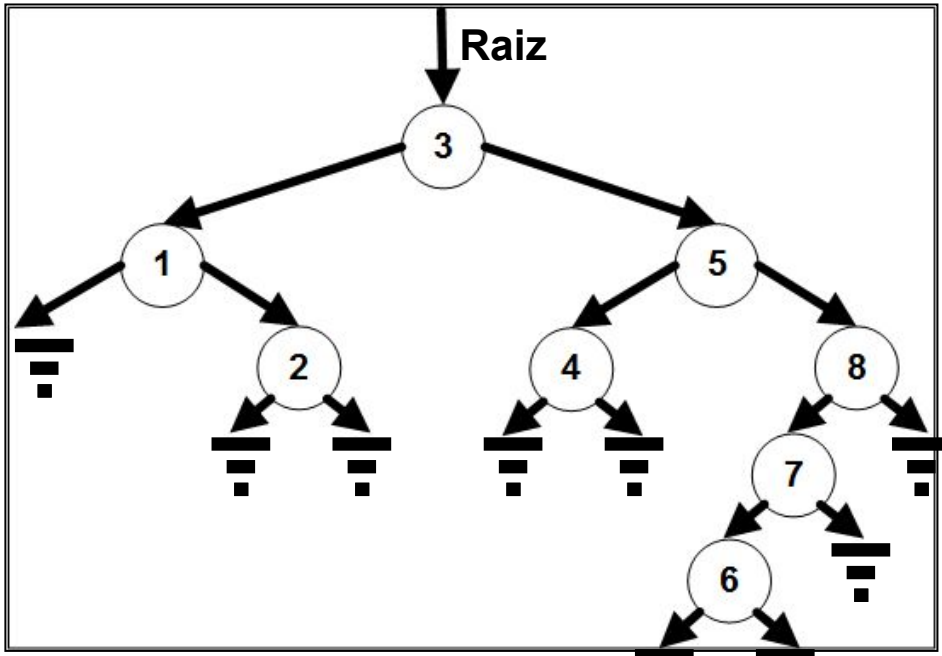
Vamos remover o 3 (tem dois filhos) de nossa árvore



Algoritmo de Remoção em Java

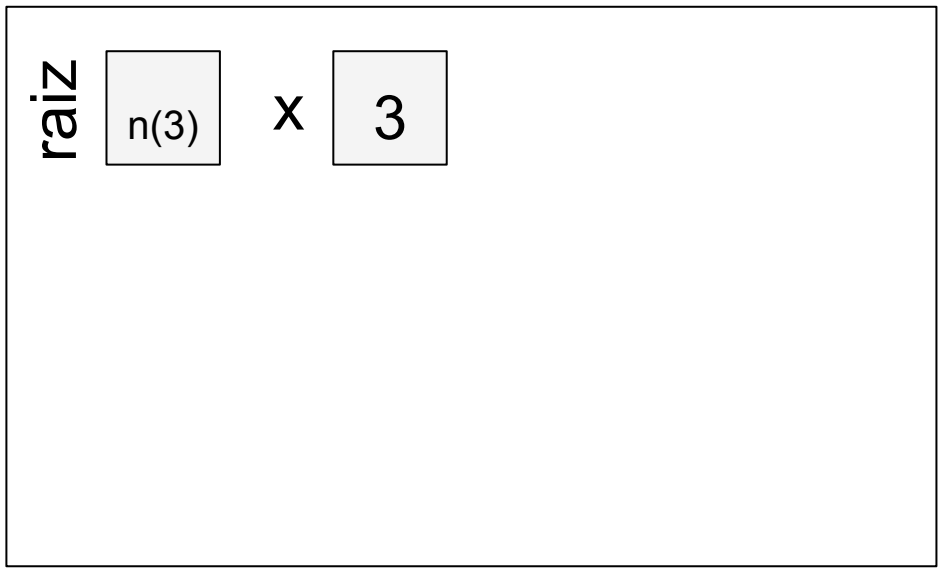
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

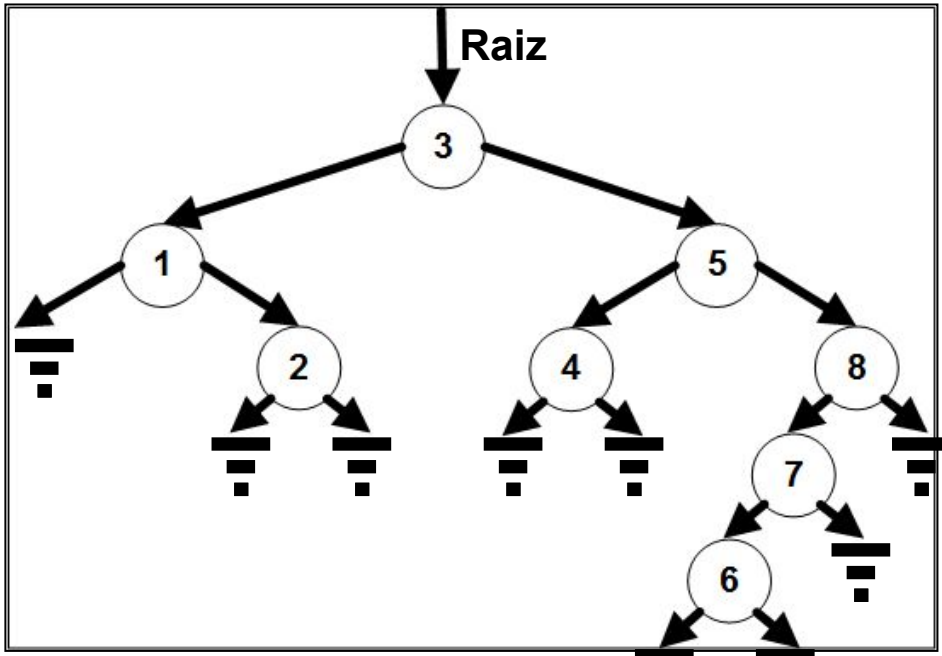
raiz n(3) x 3



Algoritmo de Remoção em Java

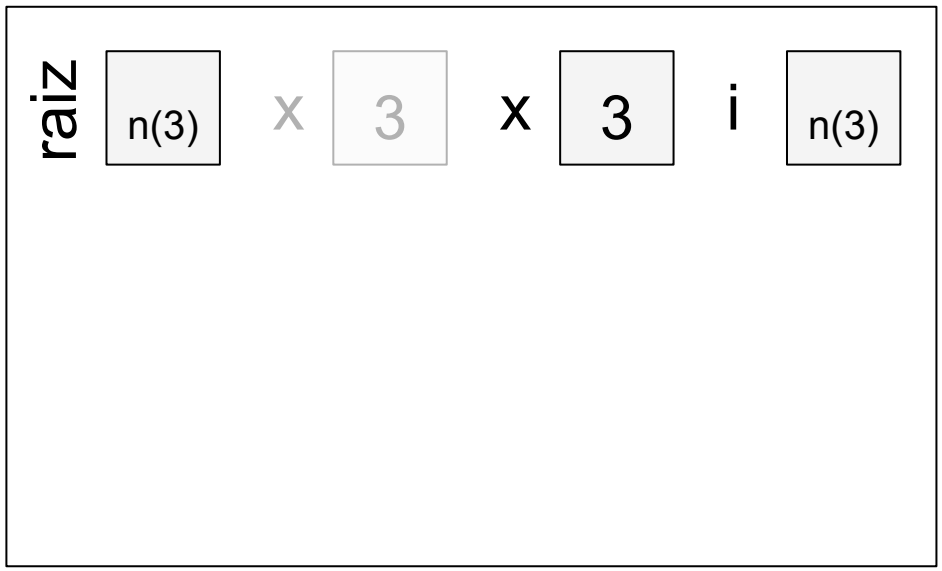
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

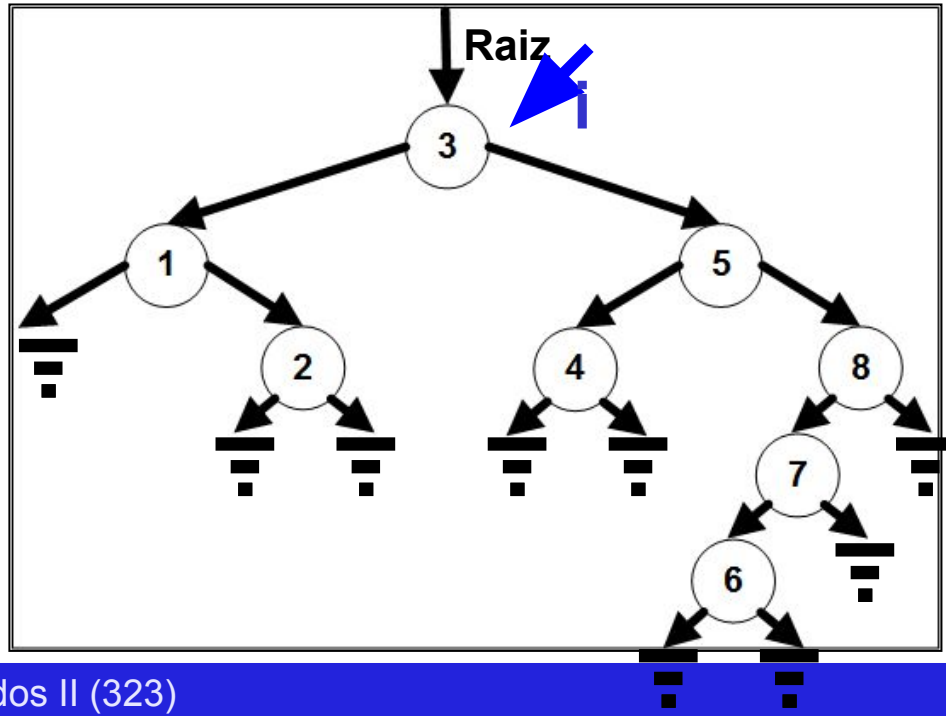




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

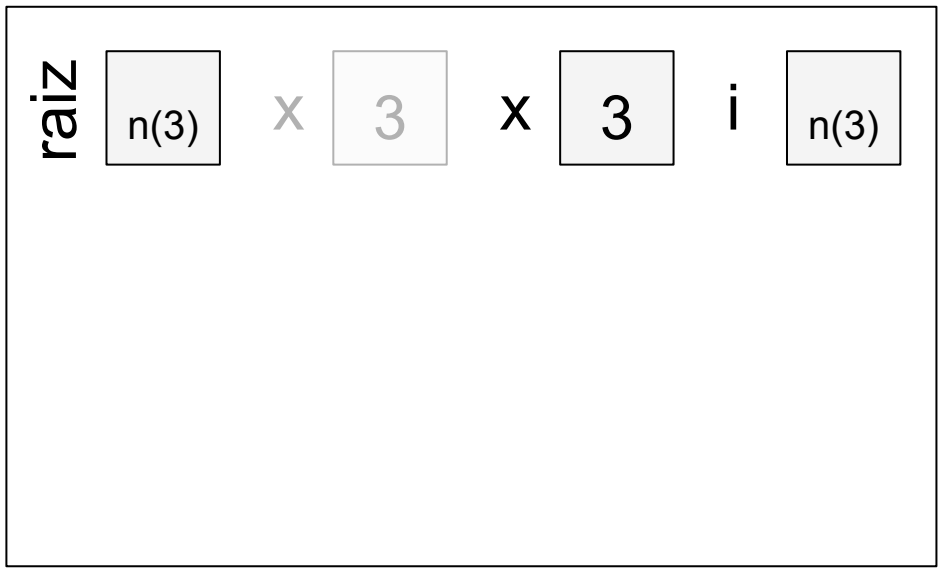


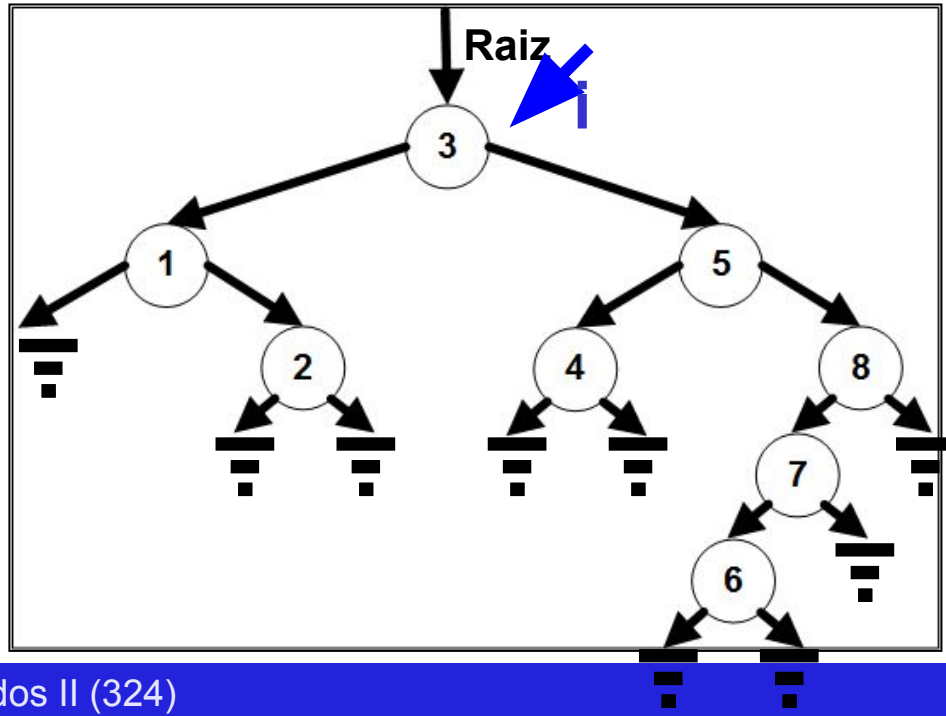


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {      throw new("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;
}

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;
}
```

false: n(3) == null

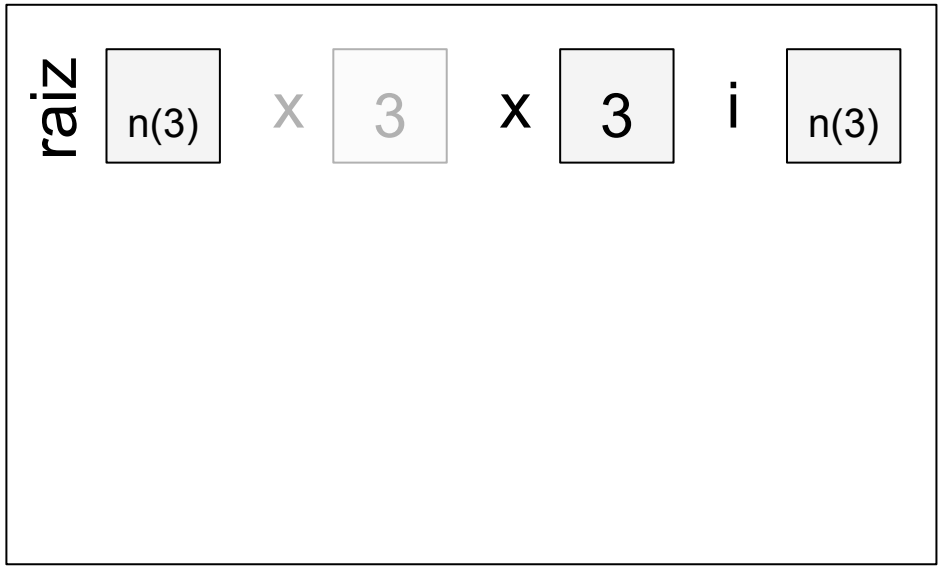


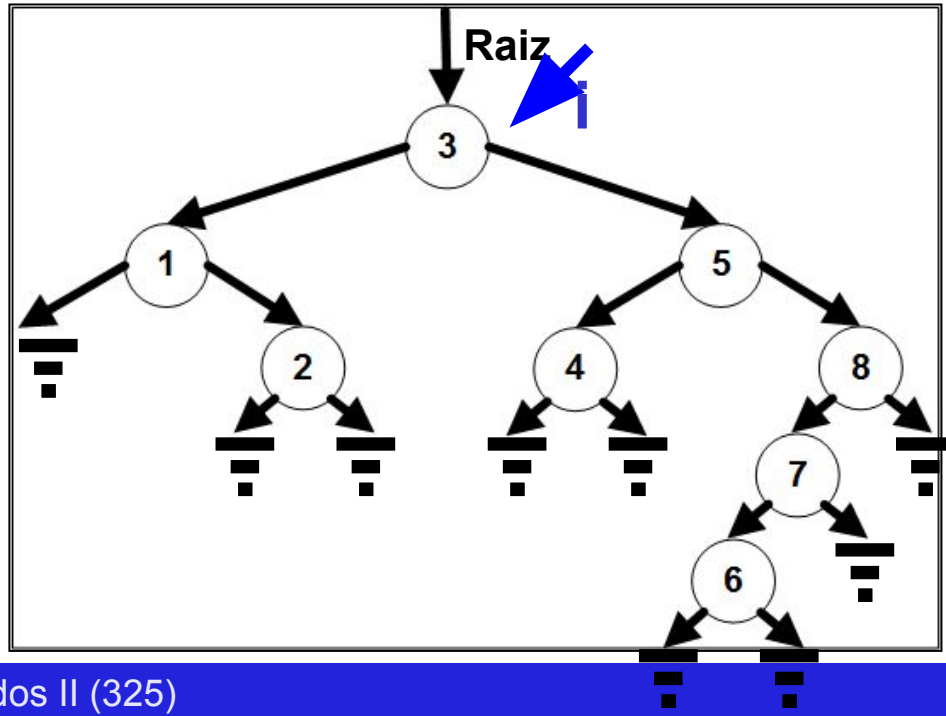


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: 3 < 3

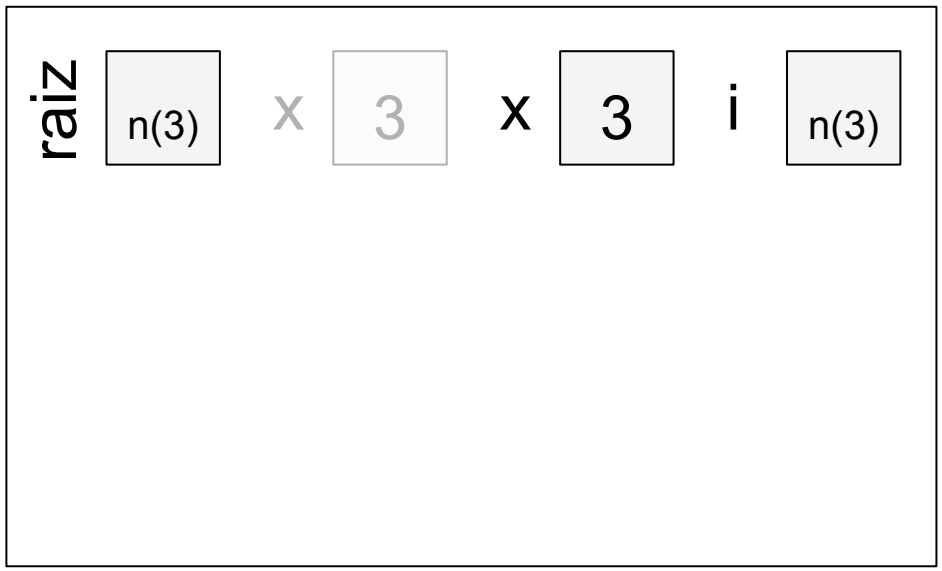


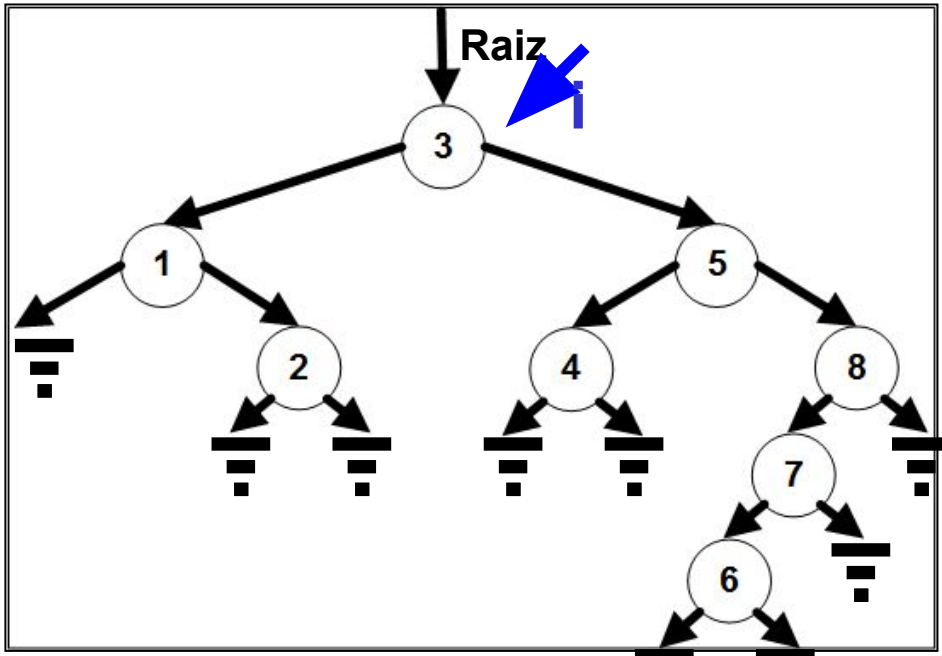


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: 3 > 3

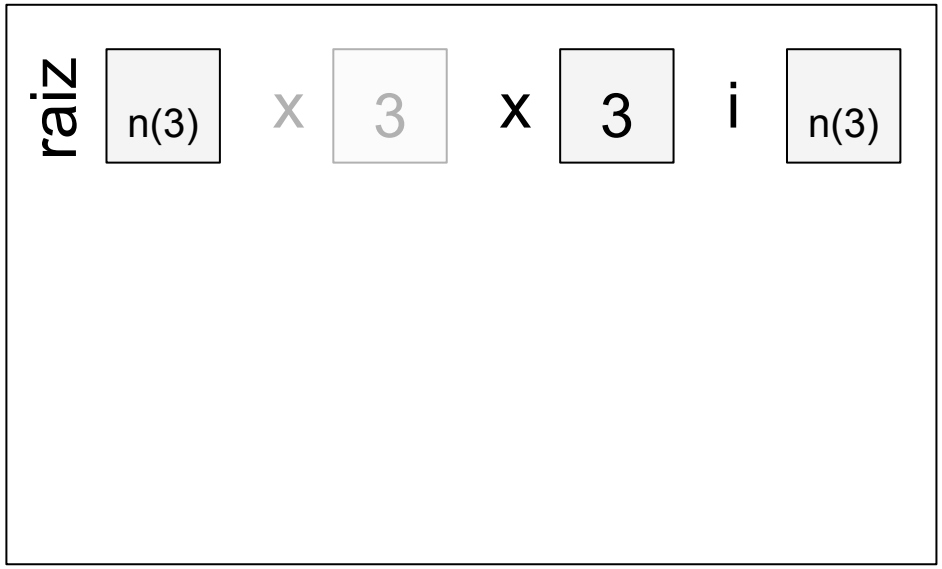


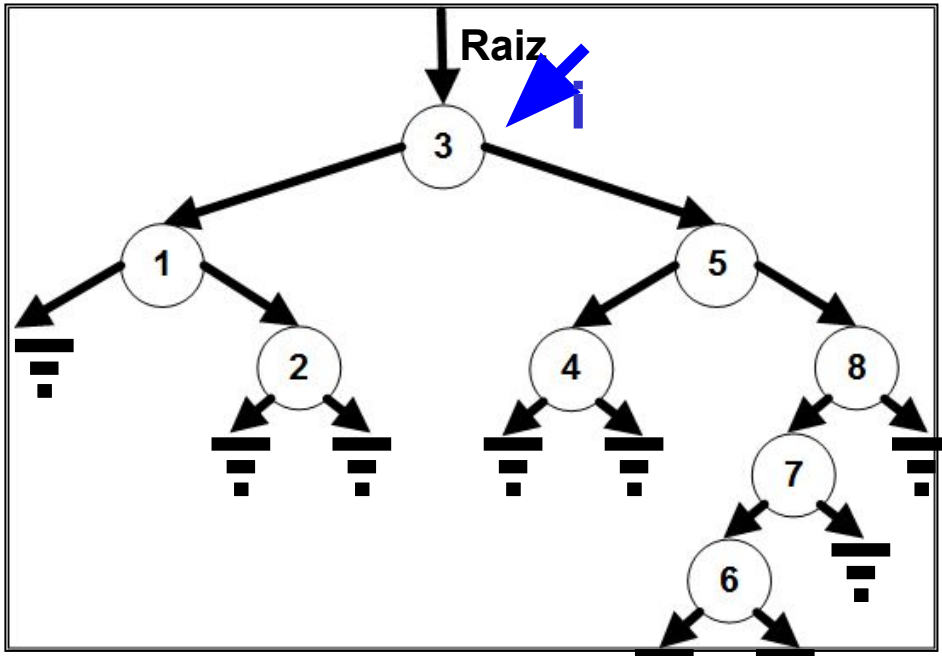


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

false: n(5) == false



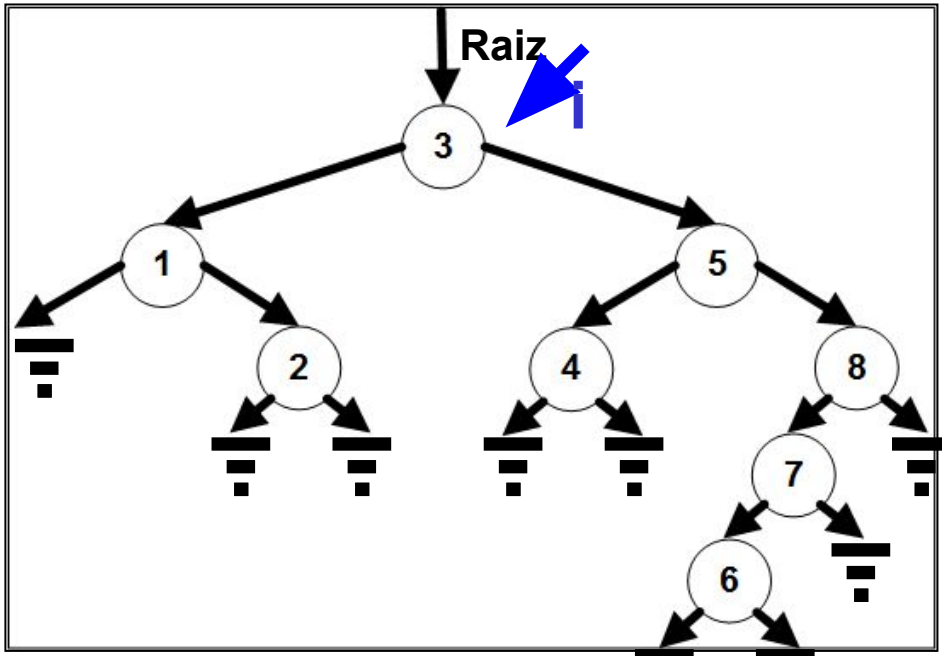


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

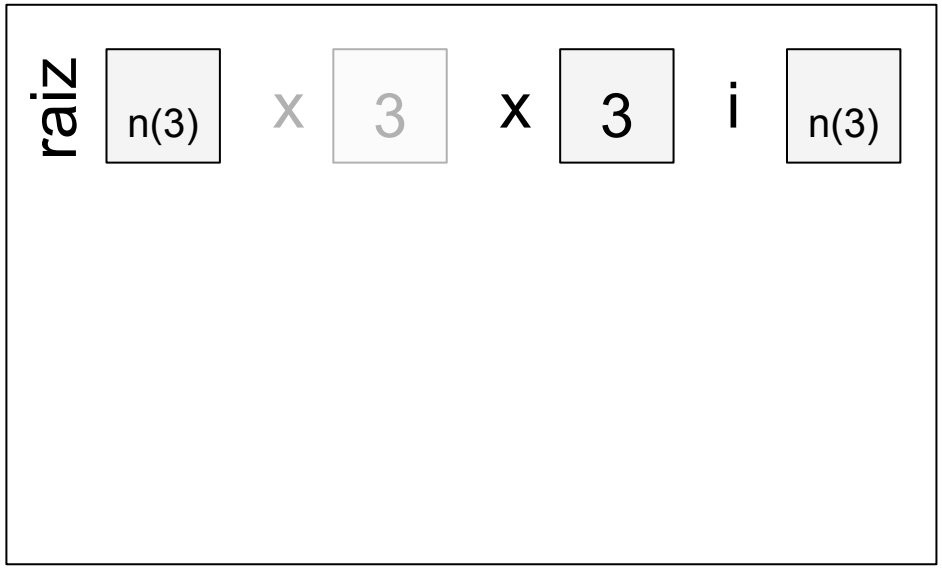
false: n(1) == false

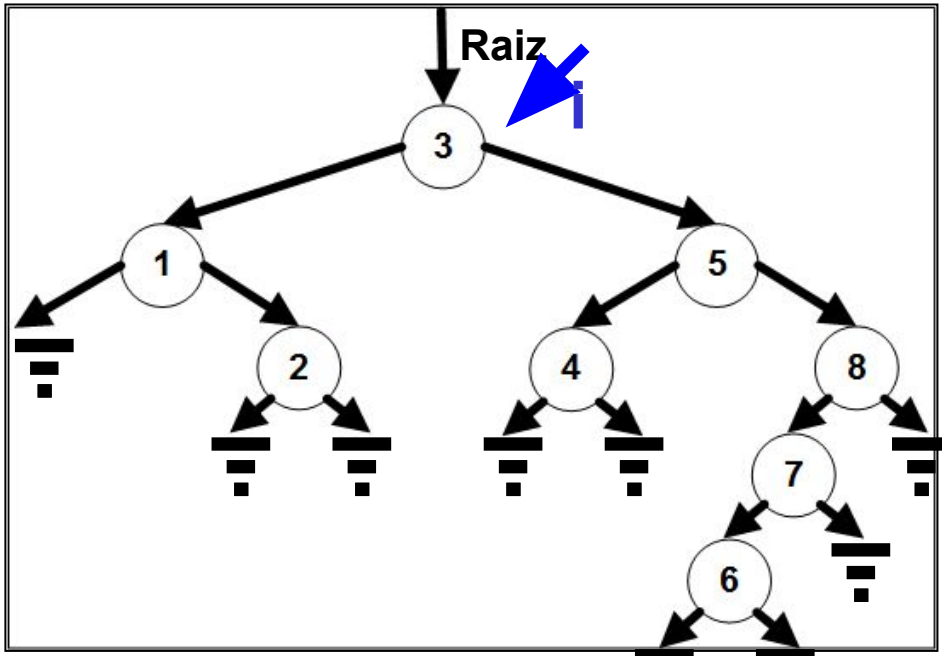
raiz n(3) x 3 x 3 i n(3)



Algoritmo de Remoção em Java

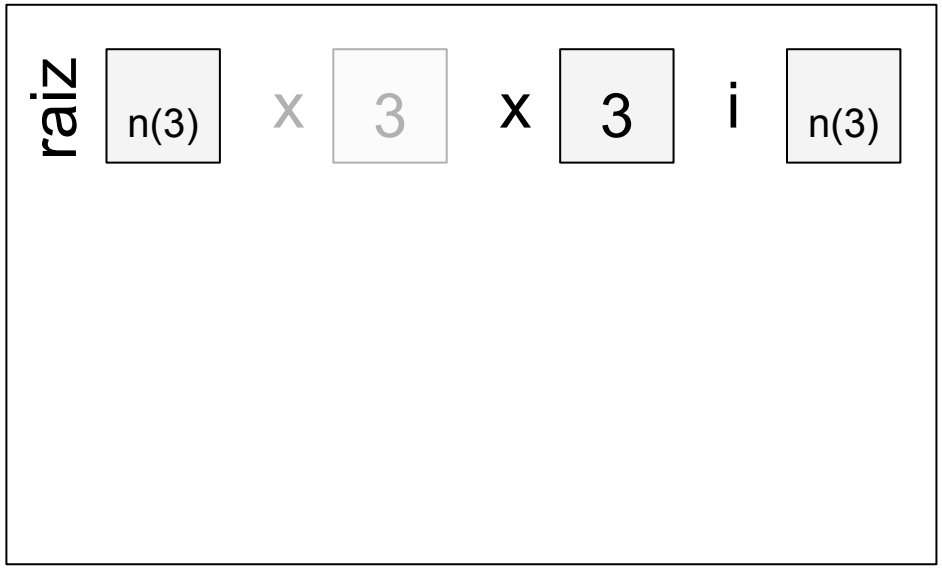
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

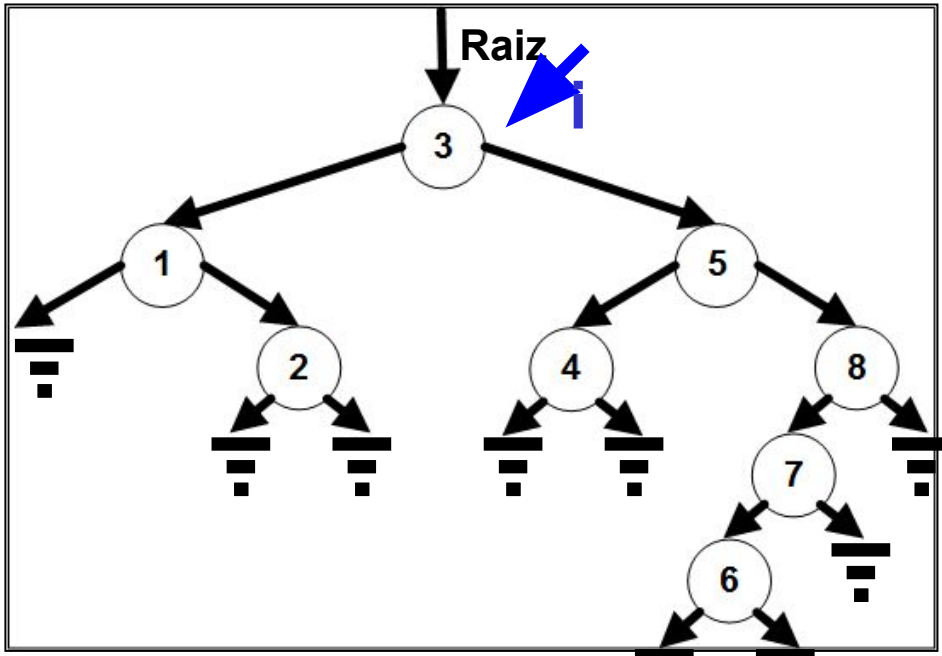




Algoritmo de Remoção em Java

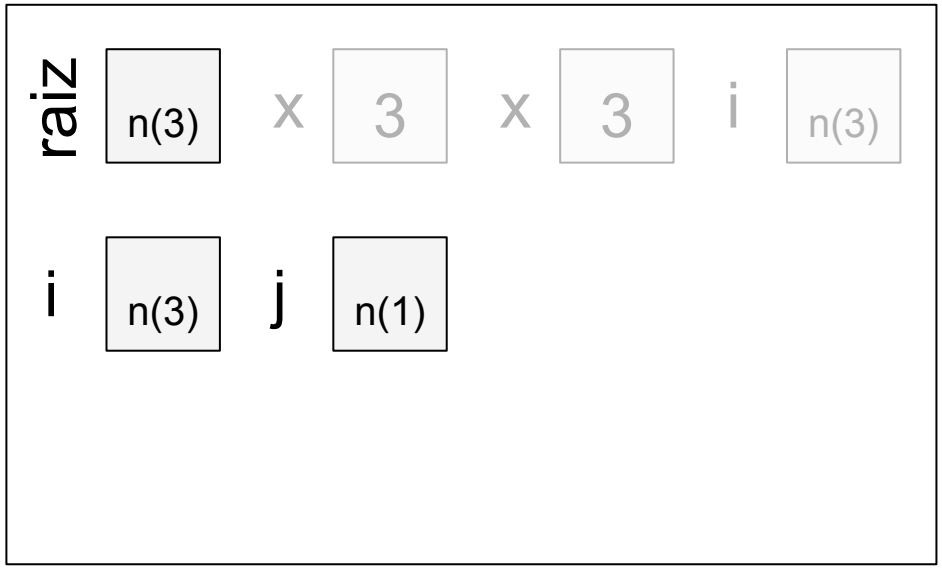
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

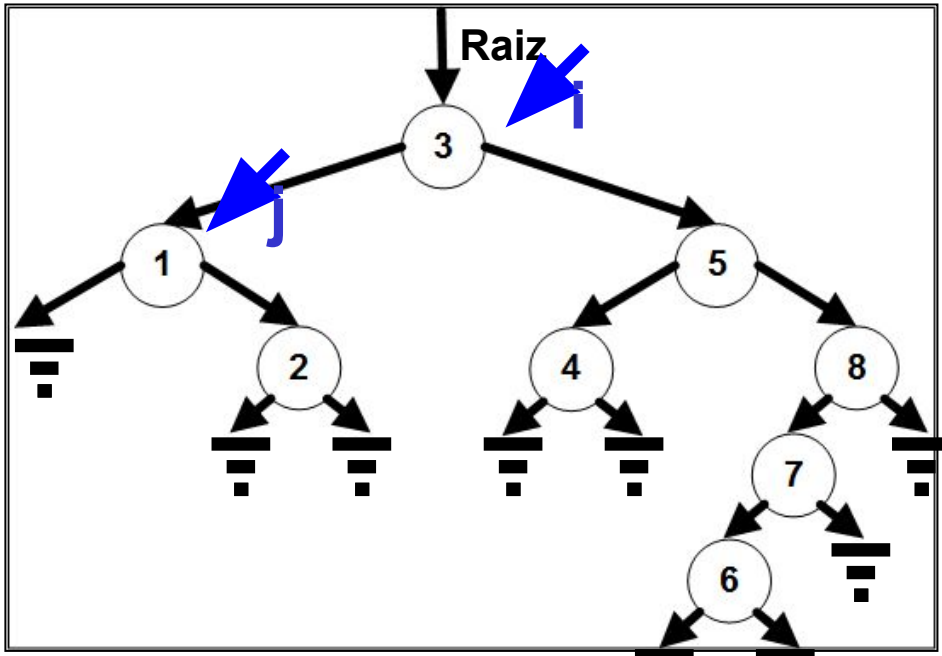




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

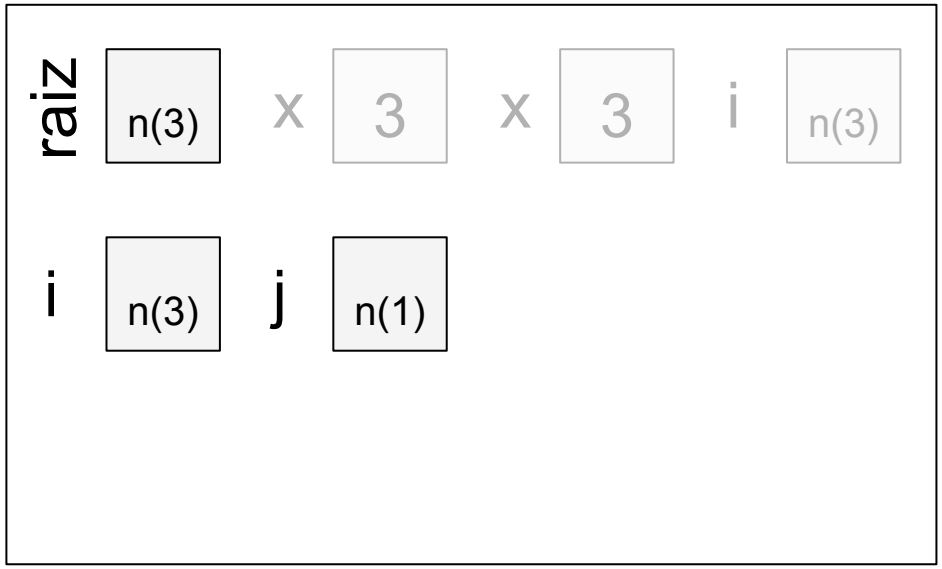


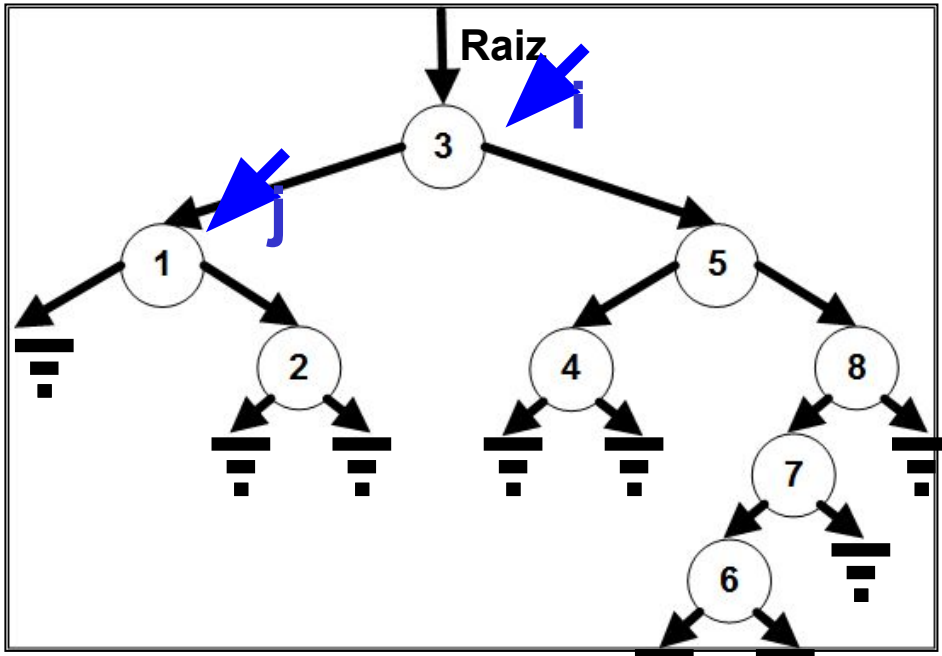


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

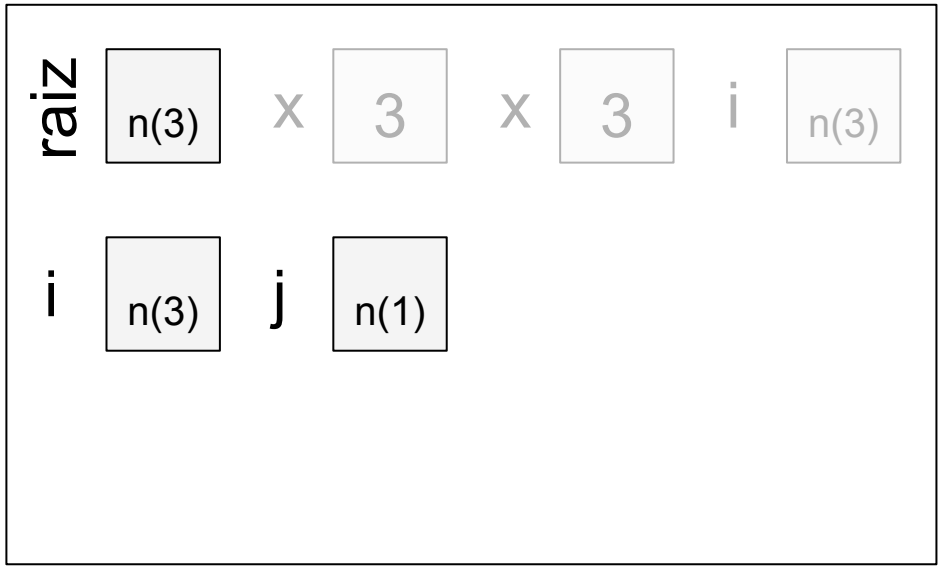
true: n(2) != null

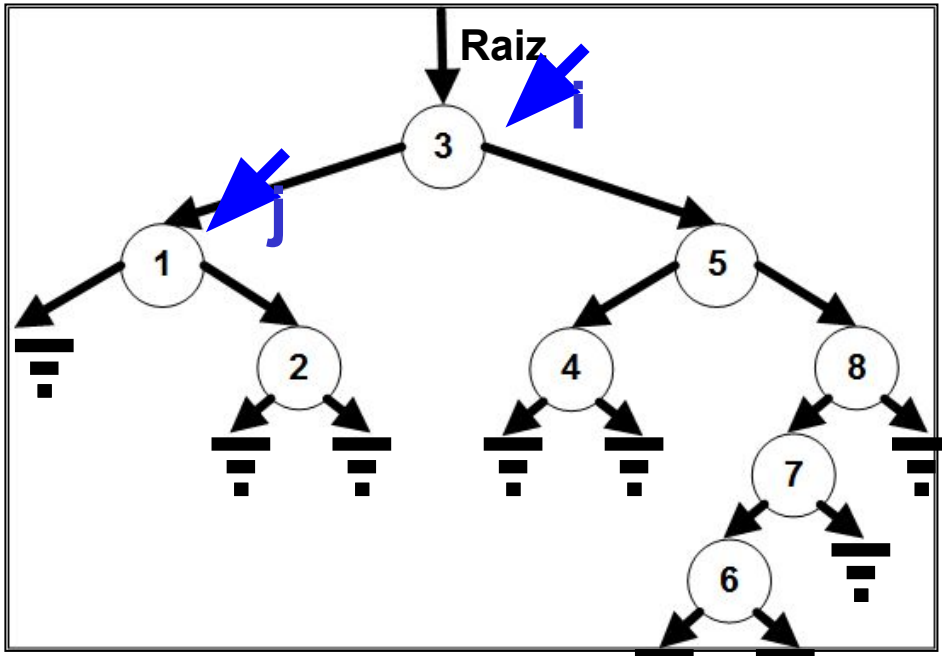




Algoritmo de Remoção em Java

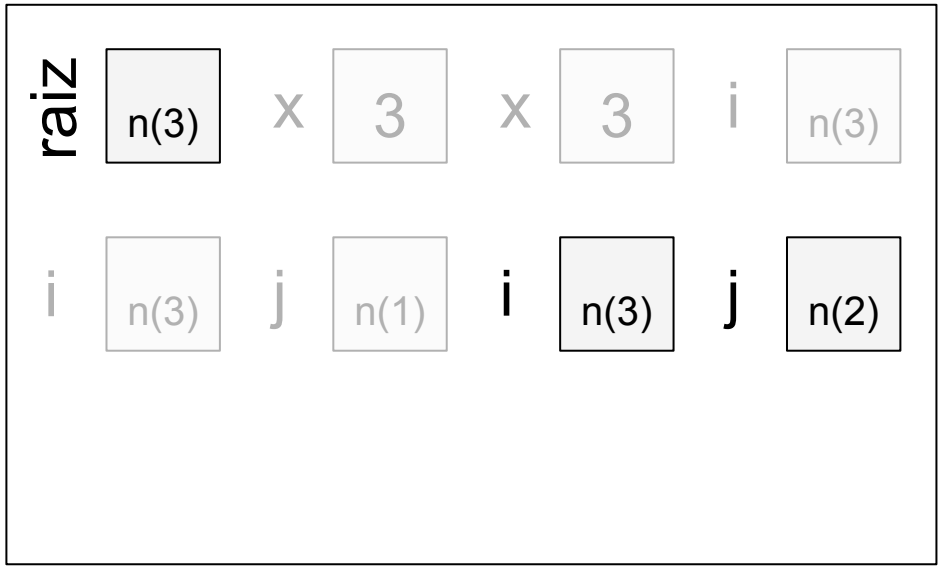
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

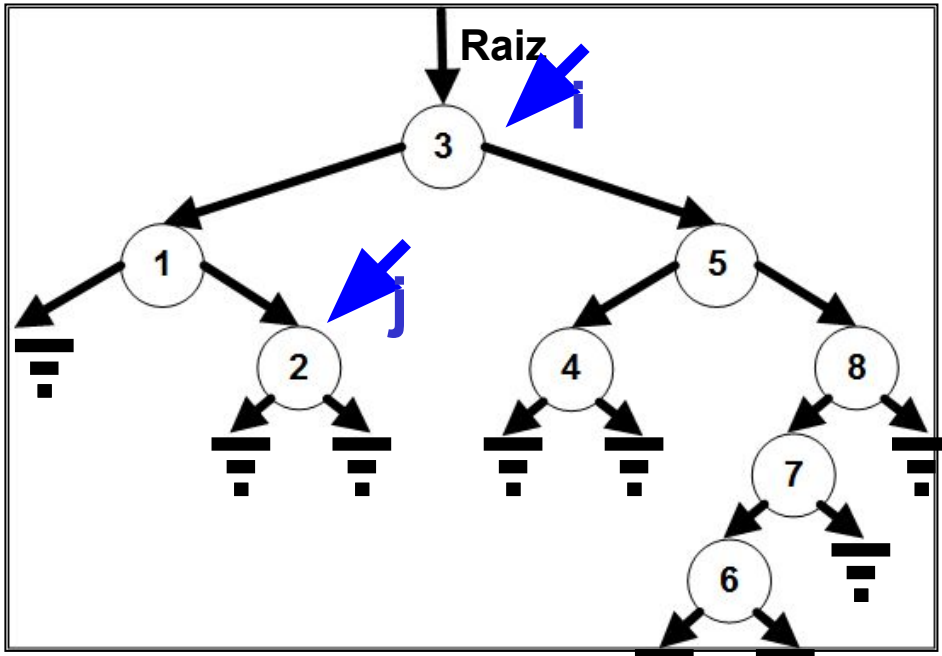




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

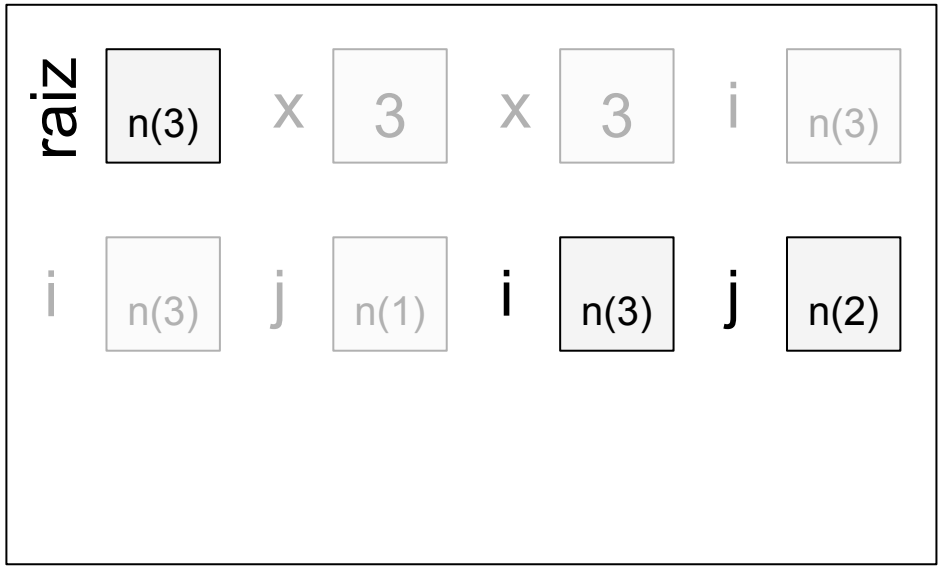


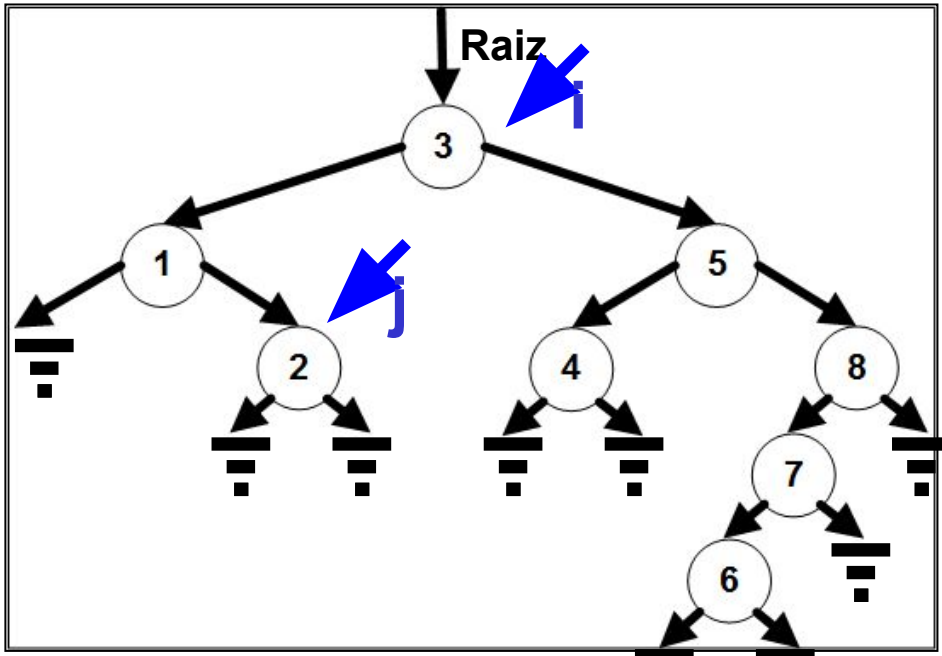


Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

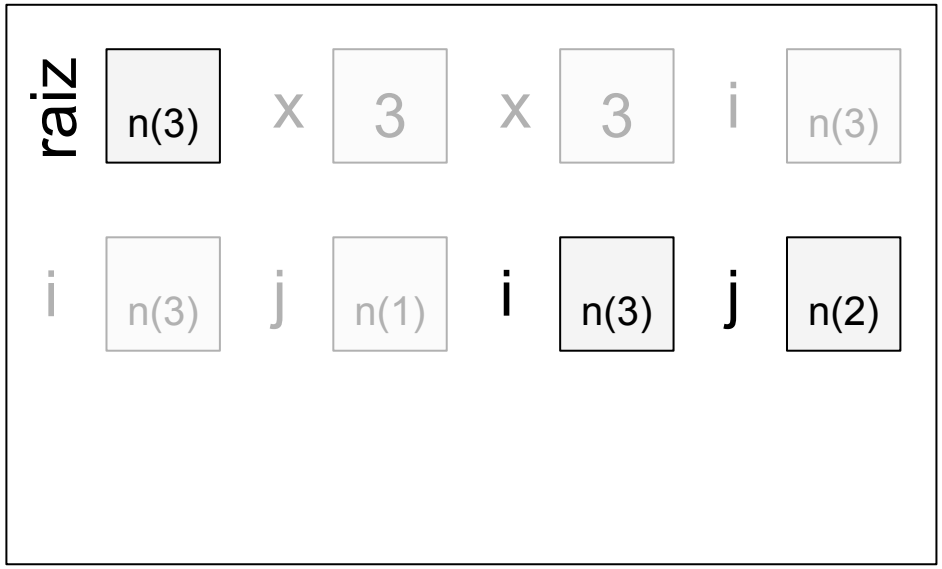
false: null != null

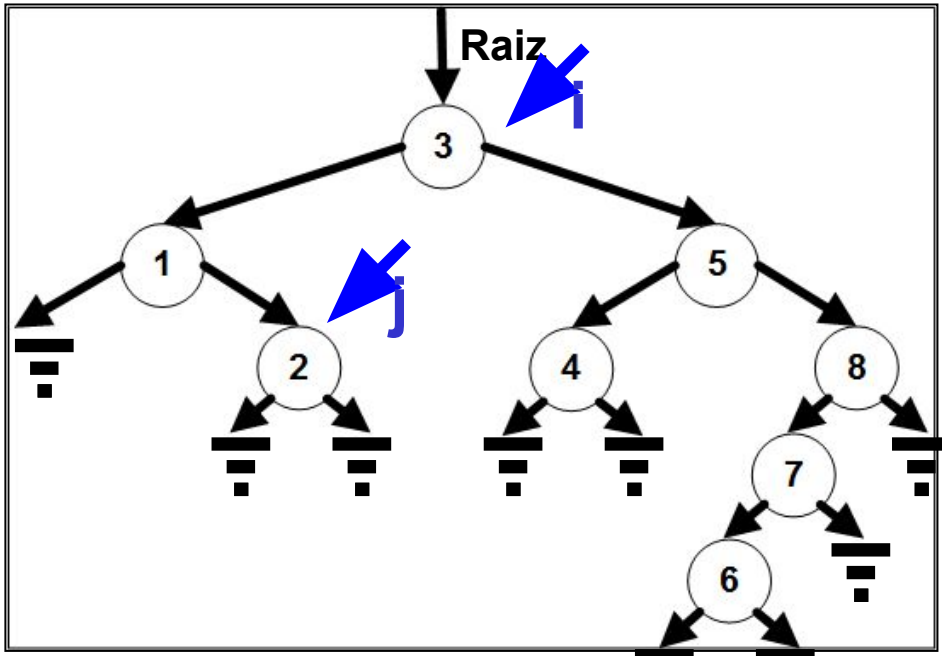




Algoritmo de Remoção em Java

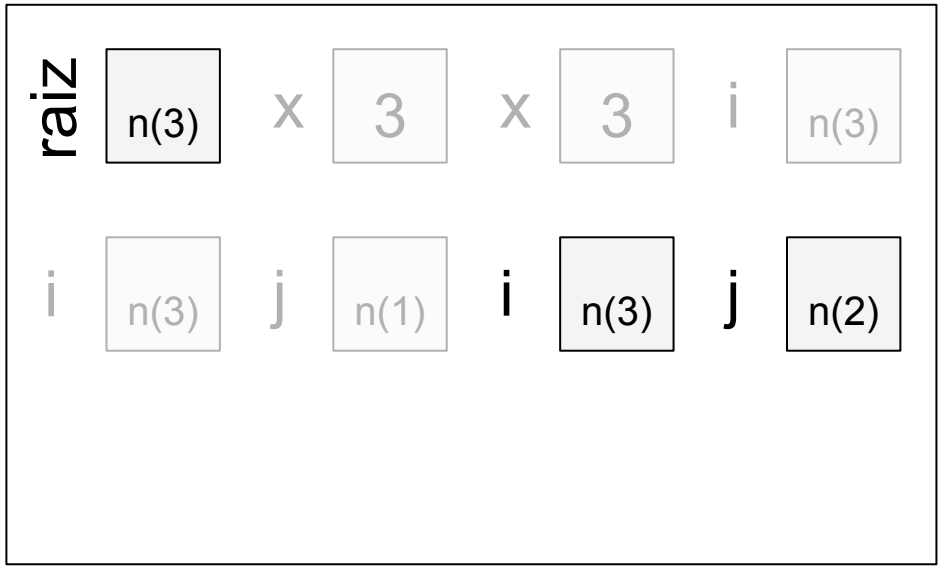
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

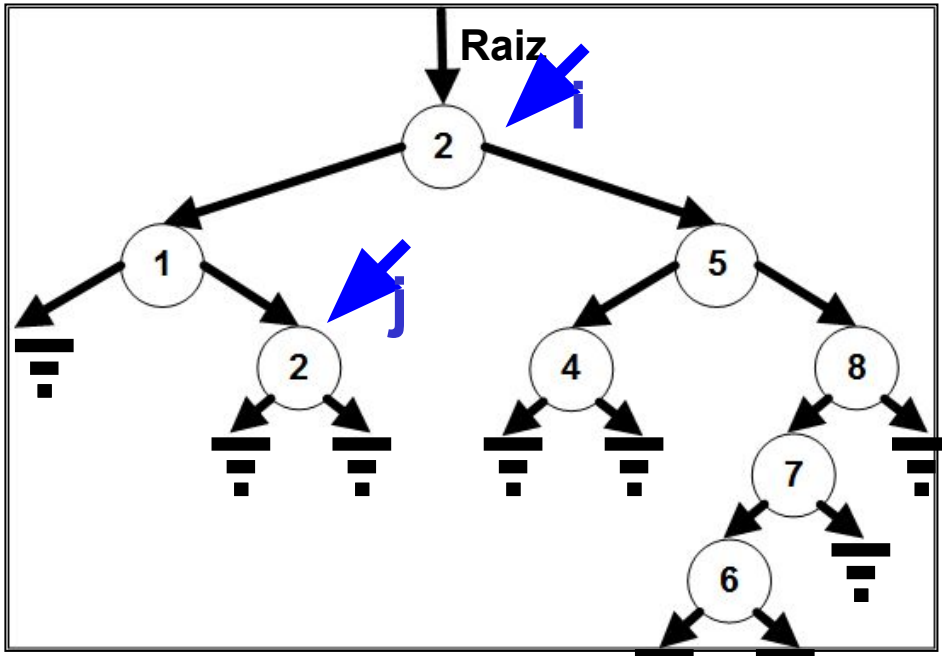




Algoritmo de Remoção em Java

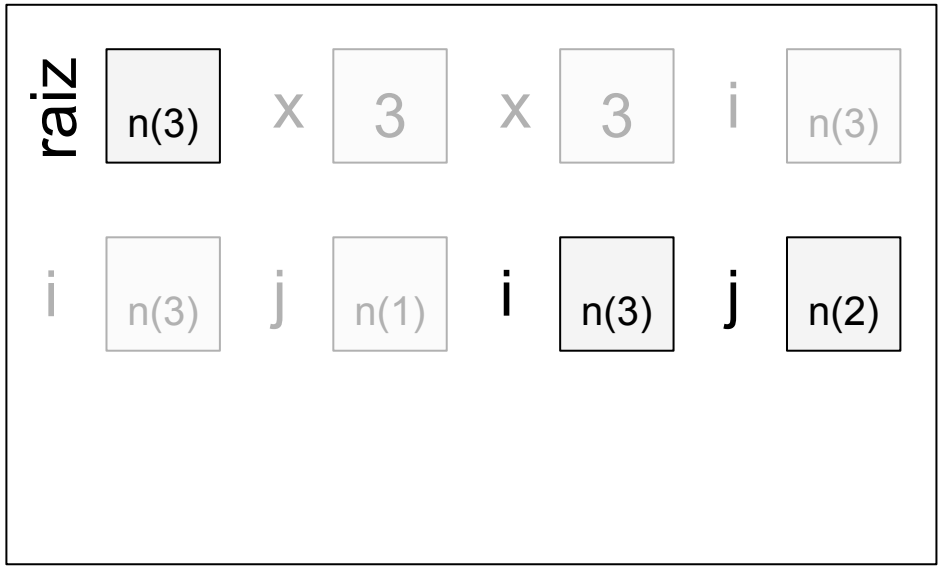
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

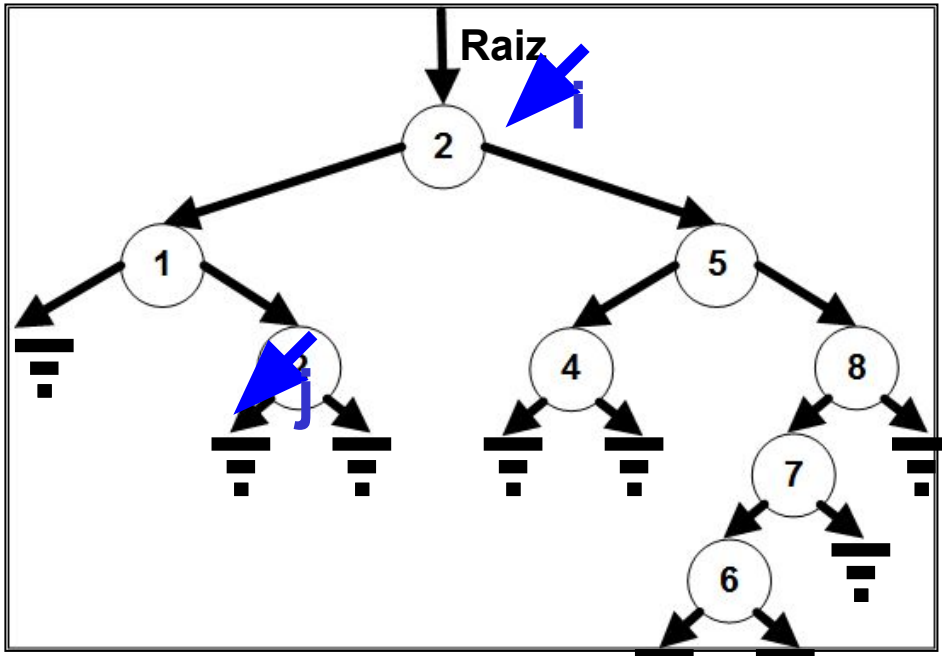




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



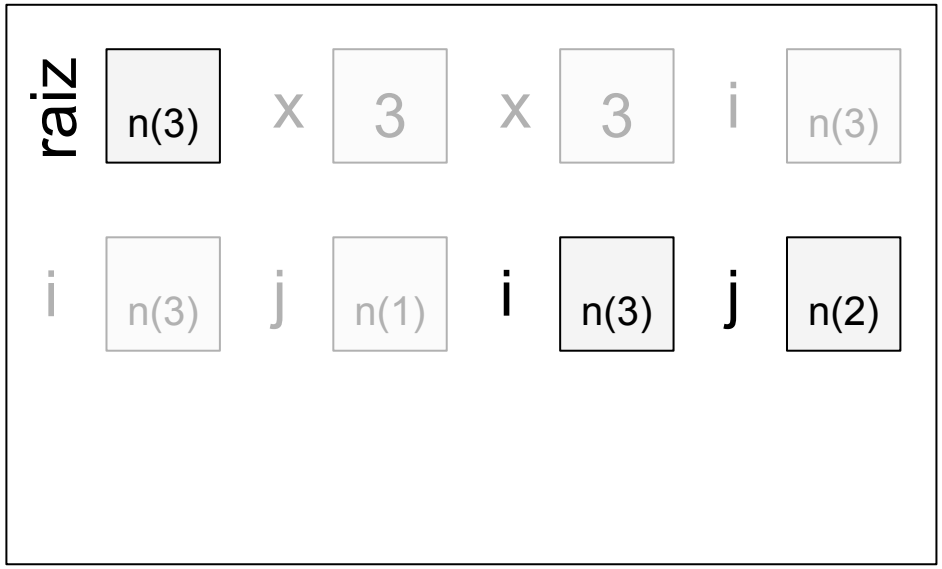


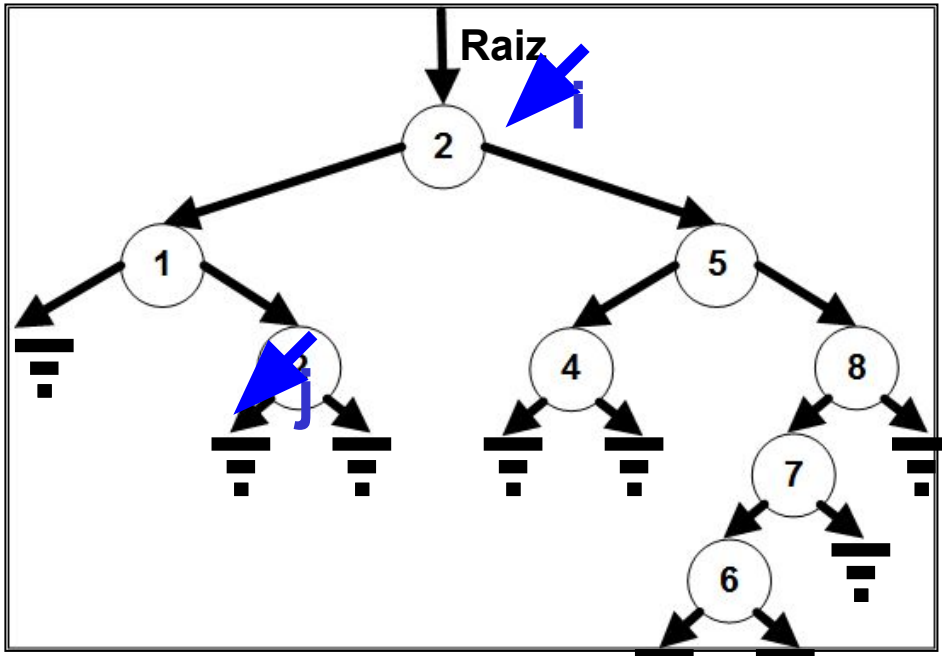
Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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

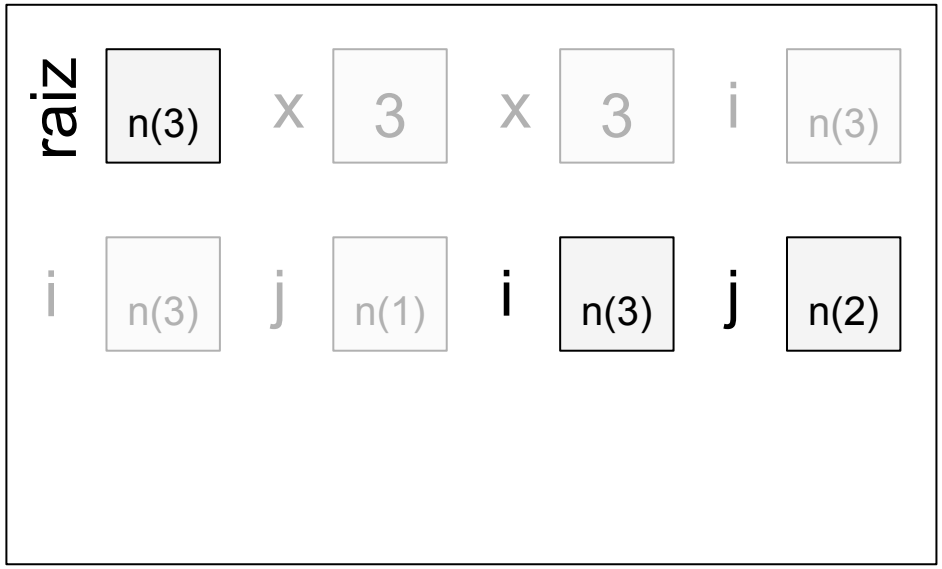
```
No anterior(No i, No j) {
   if (j.dir != null)  j.dir = anterior(i, j.dir);
   else {              i.elemento = j.elemento; j = j.esq; }
   return j;
}
```

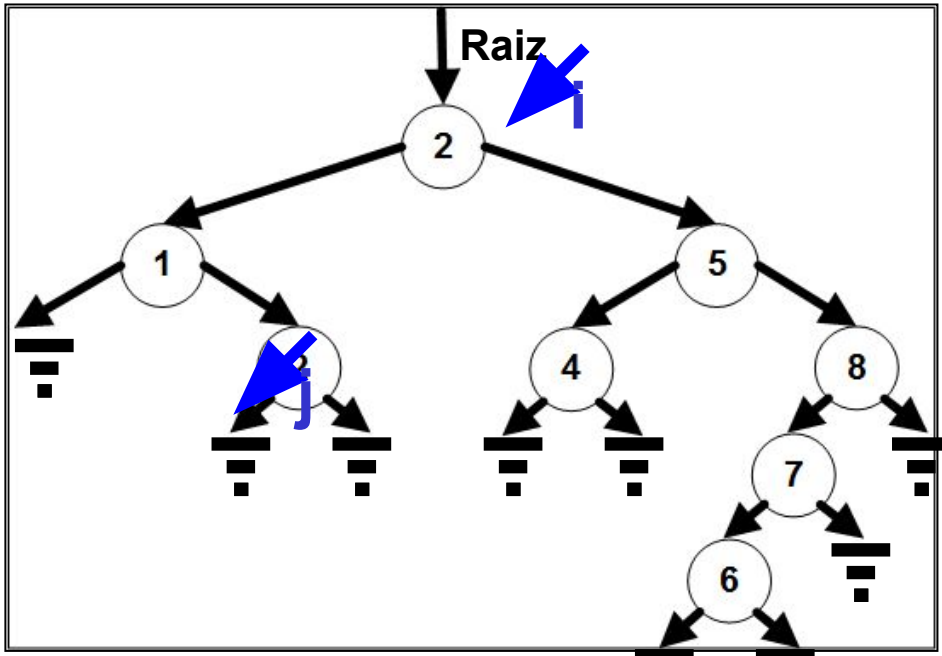




Algoritmo de Remoção em Java

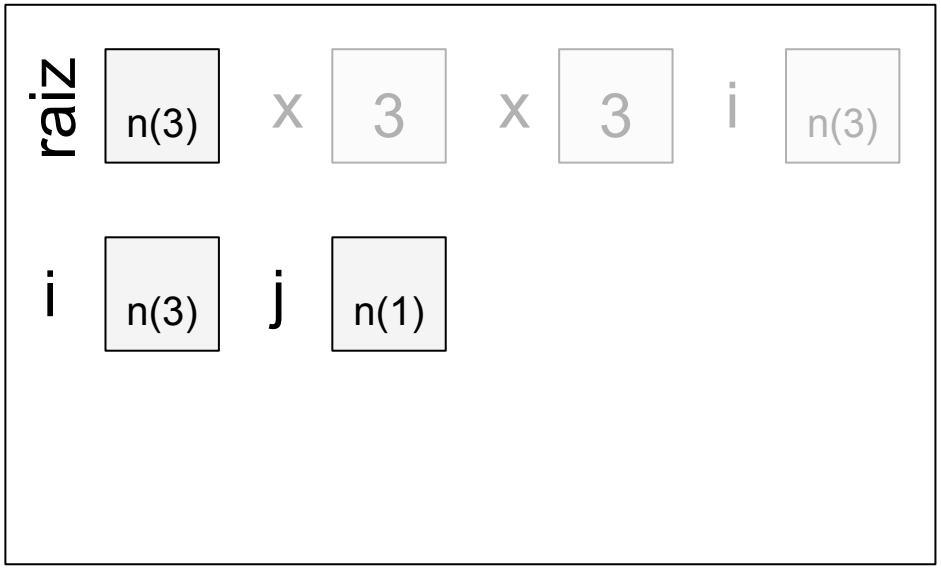
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

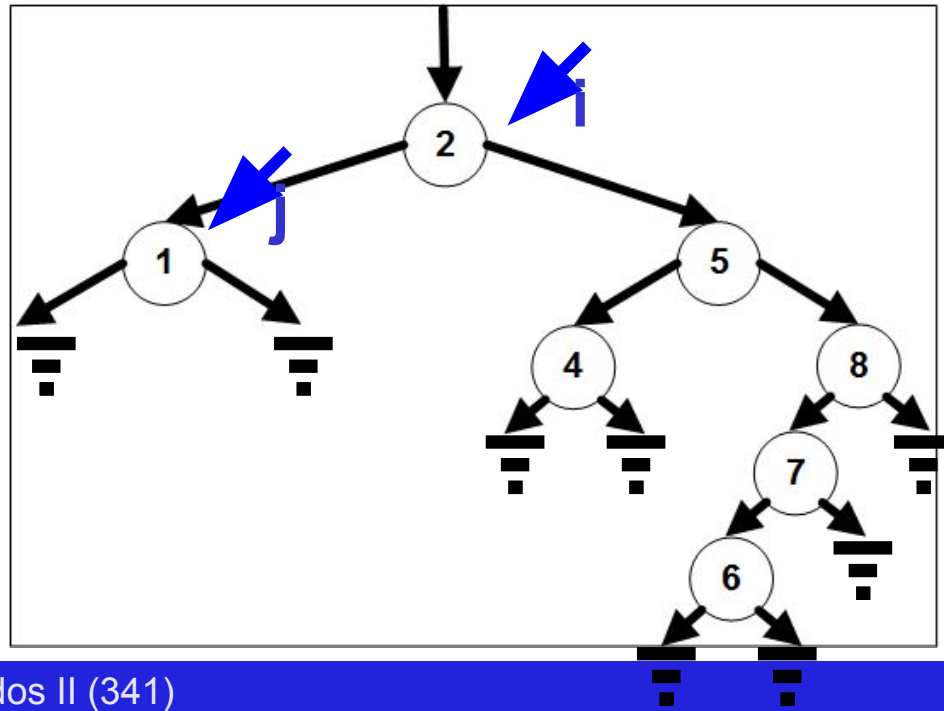




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```



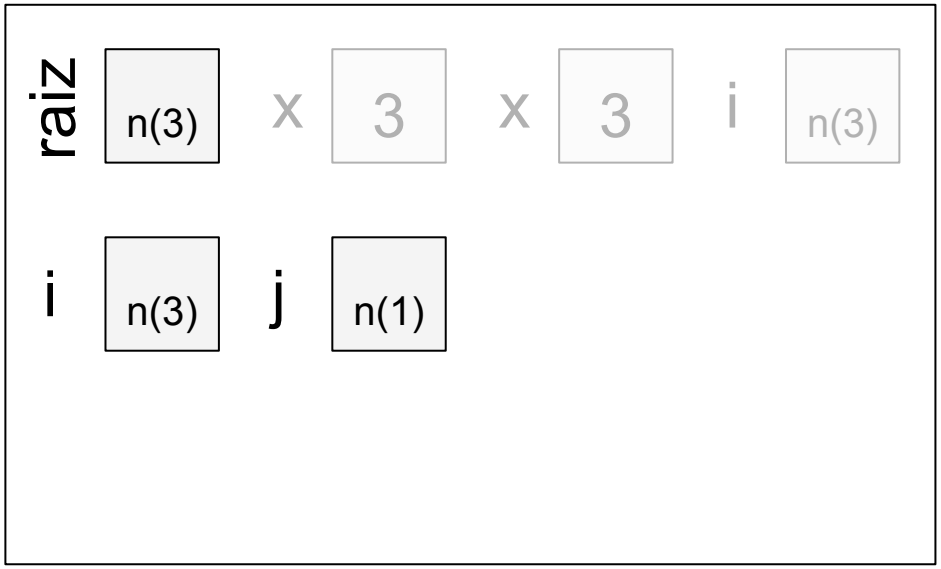


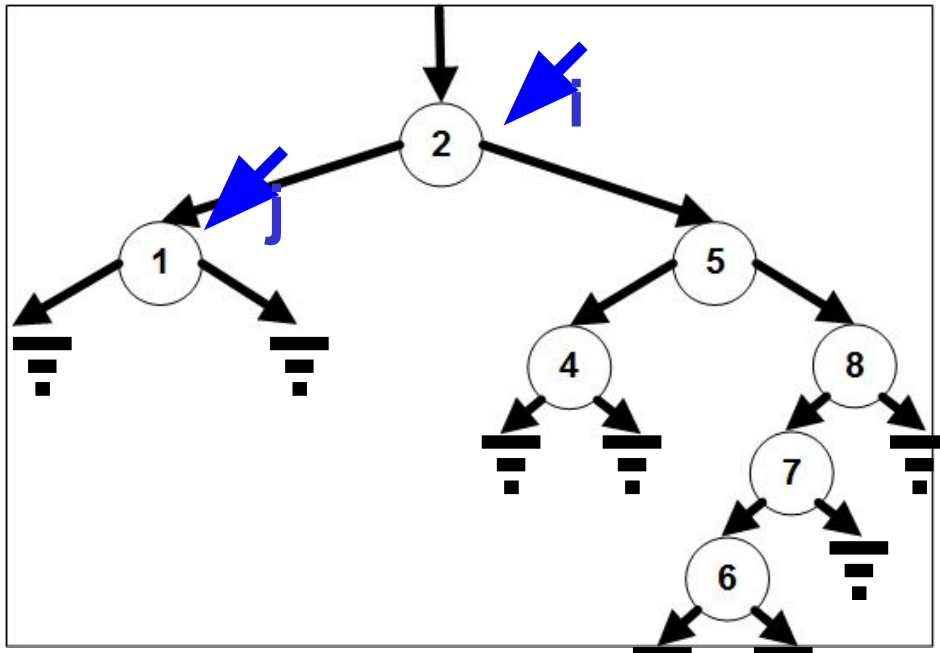
Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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)

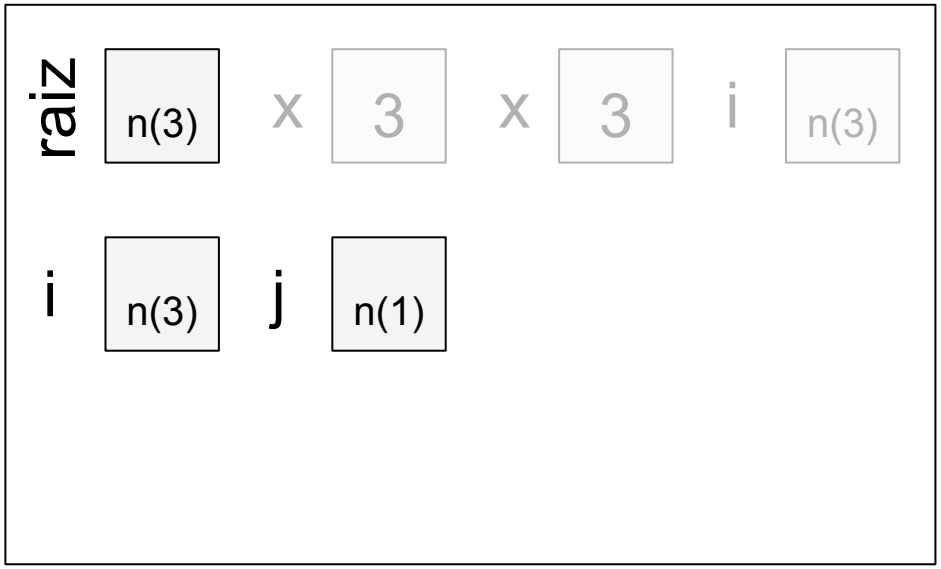
```
No anterior(No i, No j) {
   if (j.dir != null)  j.dir = anterior(i, j.dir);
   else {              i.elemento = j.elemento; j = j.esq; }
   return j;
}
```

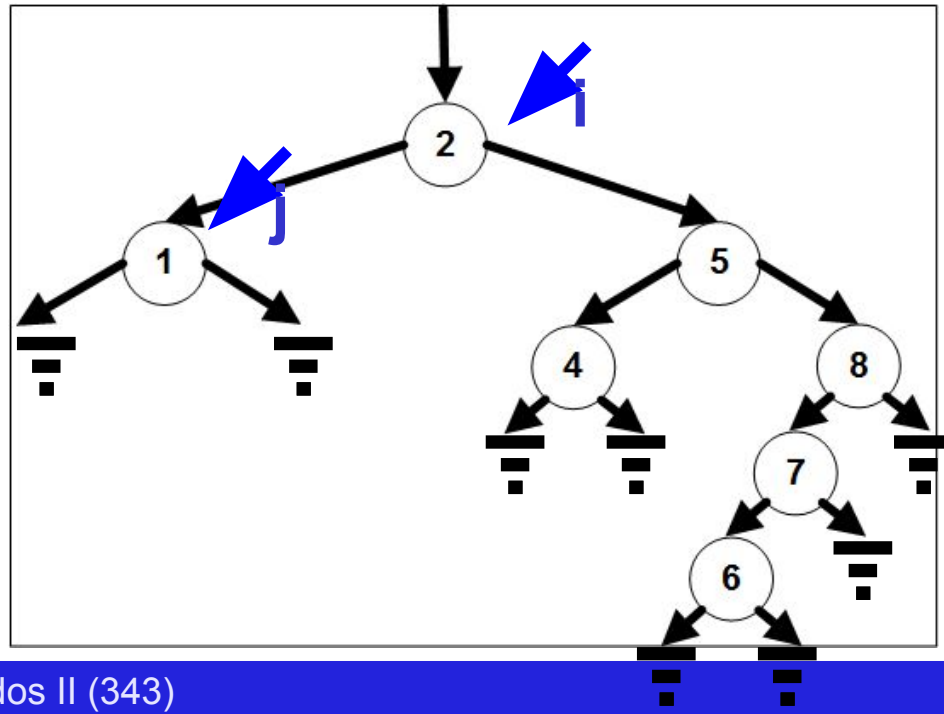




Algoritmo de Remoção em Java

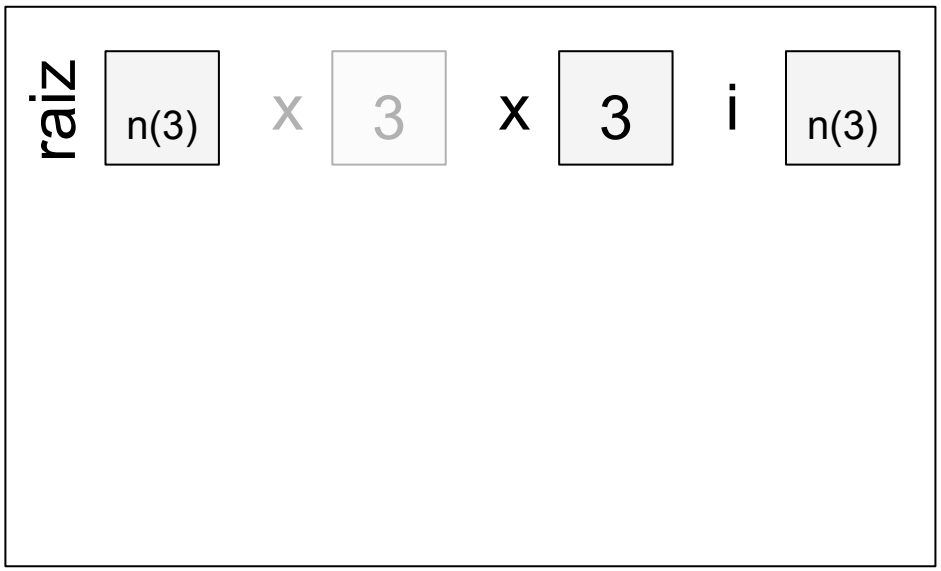
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

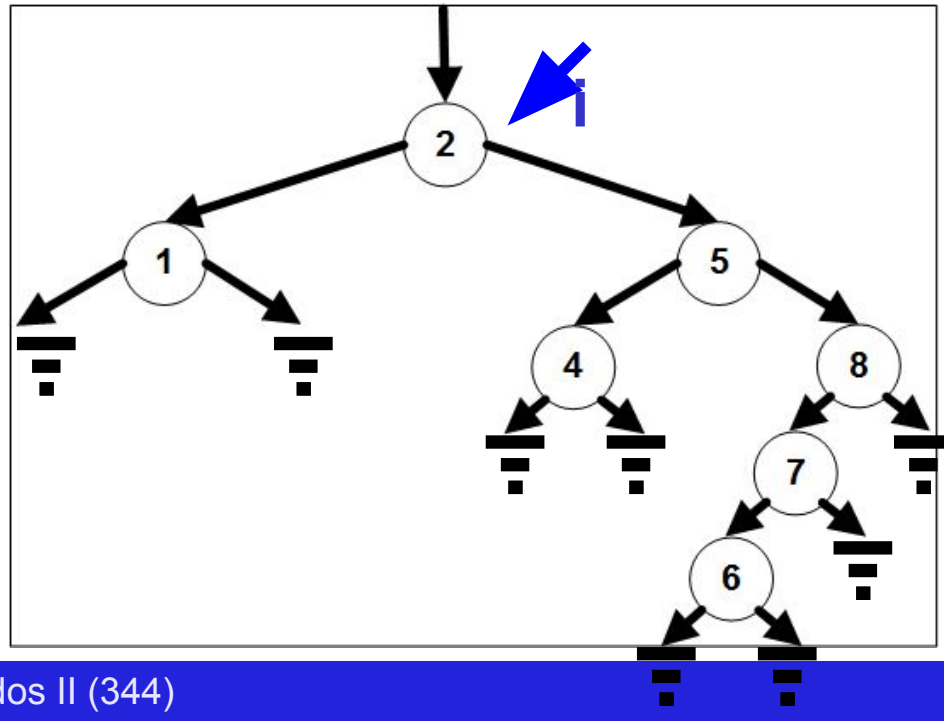




Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```





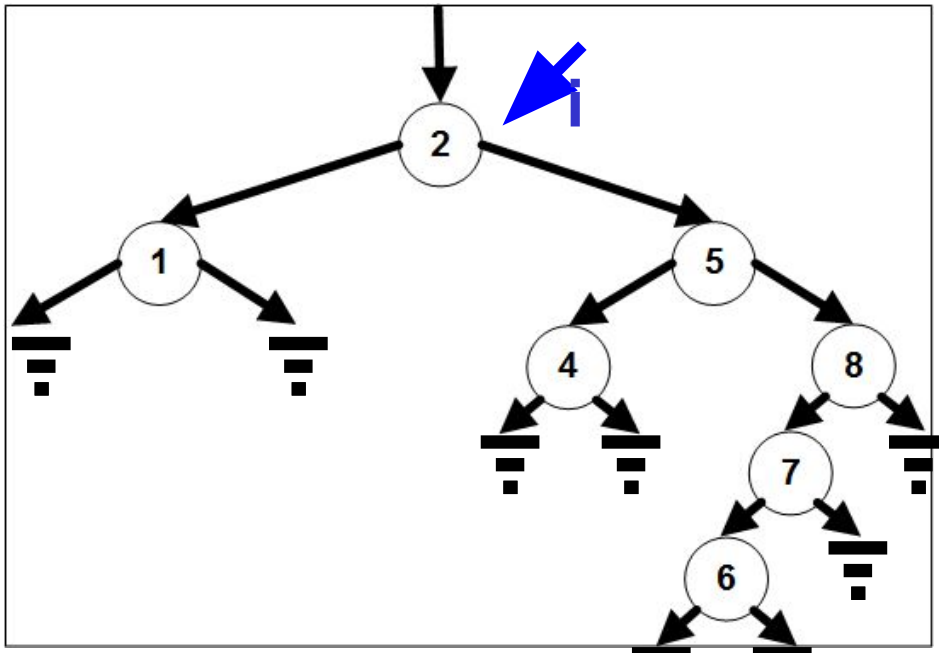
Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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)

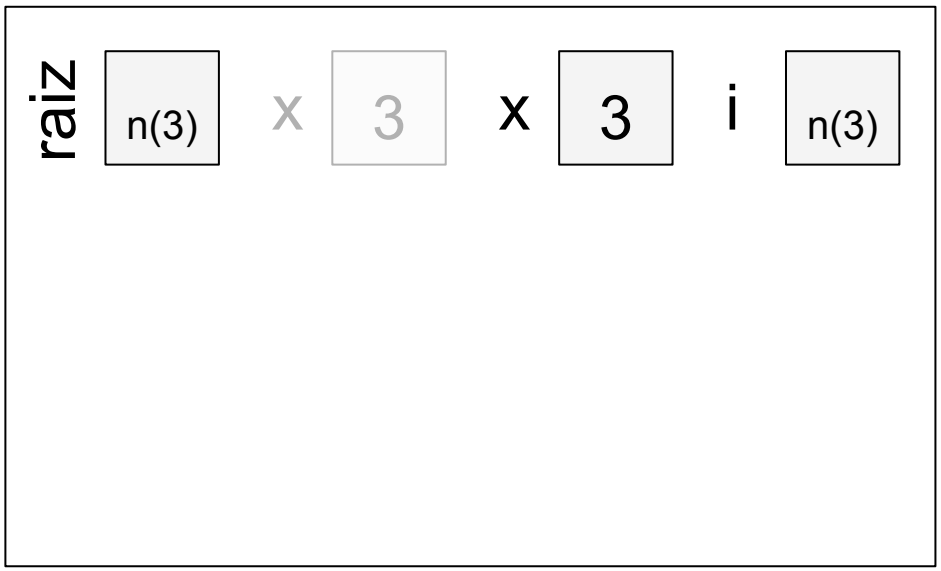
```
No anterior(No i, No j) {
   if (j.dir != null)  j.dir = anterior(i, j.dir);
   else {              i.elemento = j.elemento; j = j.esq; }
   return j;
}
```

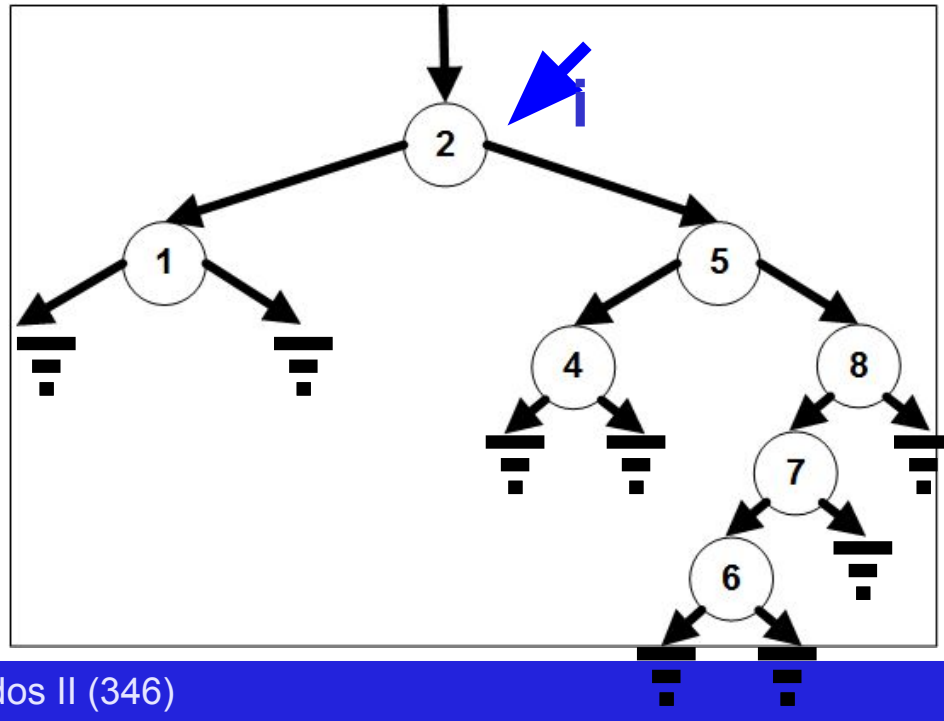
raiz | n(3) | x | 3 | x | 3 | i | n(3)



Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

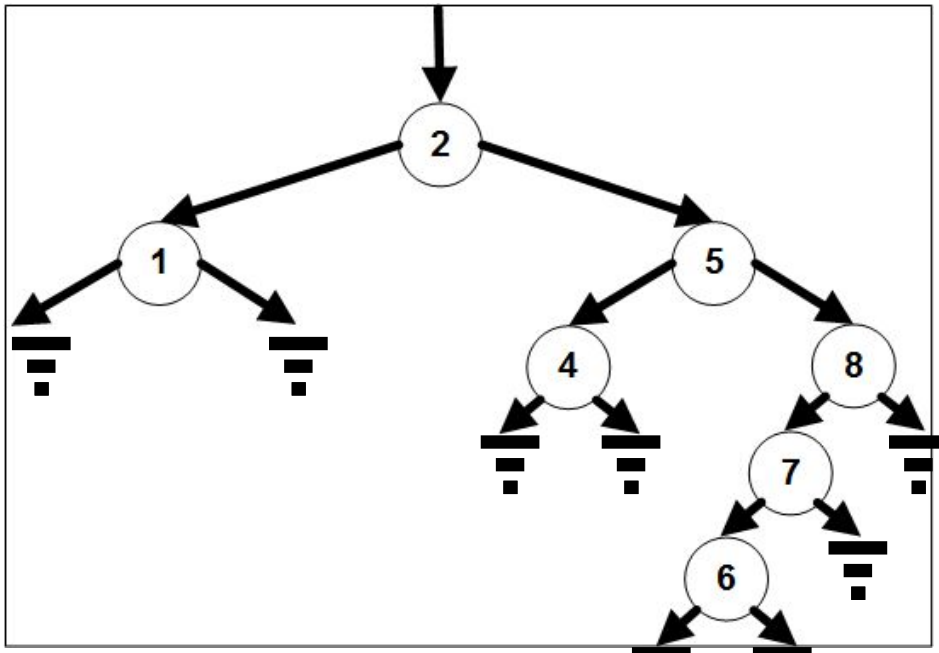




Algoritmo de Remoção em Java

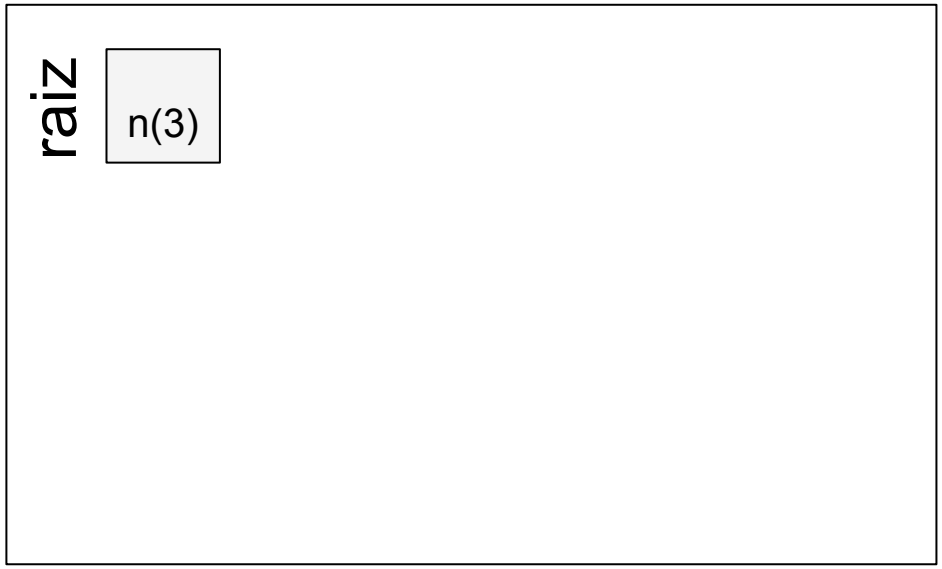
```
//remover(3), dois filhos

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```

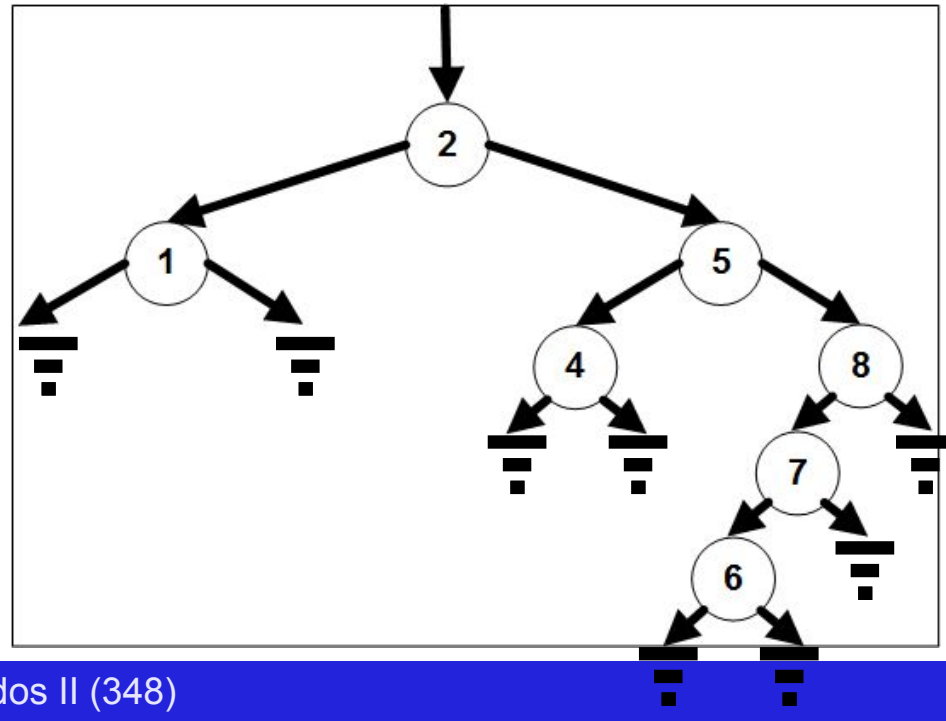
raiz n(3) x 3



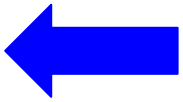
Algoritmo de Remoção em Java

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

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

No remover(int x, No i) {
   if (i == null) {       throw new("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;
}

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;
}
```





Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Caminhamento
- **Remoção** 
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- Estruturas híbridas

  - Funcionamento básico
  - Algoritmo em Java
  - **Análise de Complexidade**

Análise de Complexidade da Remoção

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

Análise de Complexidade da Remoção

- 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)

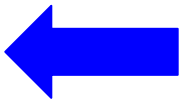
Exercício (8)

- O método ***remover*** privado e recursivo apresentado em nossa árvore recebe e um valor e retorna um No. Altere tal método para que o mesmo retorne ***void***.

Exercício (9)

- O método ***remover*** privado e recursivo apresentado em nossa árvore recebe e um valor e retorna um No. No exercício anterior, o ***remover2*** retorna *void*. Implemente um método ***remover3*** que retorna o elemento removido.

Agenda

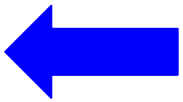
- Definições e conceitos
- Classes Nó e Árvore Bi
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- **Inserção em C com ponteiro** 
- Inserção em C++ com passagem por referência
- Estruturas híbridas

- **Estrutura arquivos**
- makefile
- Arquivos “no”
- Arquivos “arvorebinaria”

Estrutura de Arquivos

- no.h
- no.c
- arvorebinaria.c
- arvorebinaria.h
- principal.c
- makefile

Agenda

- Definições e conceitos
- Classes Nó e Árvore Bi
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- **Inserção em C com ponteiro** 
- Inserção em C++ com passagem por referência
- Estruturas híbridas

  - Estrutura arquivos
  - **makefile**
  - Arquivos “no”
  - Arquivos “arvorebinaria”

makefile

- Arquivo contendo um conjunto de diretivas usadas pela ferramenta de automação de compilação *make* para gerar um alvo / meta

- Nesse caso, os arquivos serão compilados digitando ***make***

makefile

```
1   all: exec
2
3   exec: principal.o arvorebinaria.o no.o
4       gcc -o exec principal.o arvorebinaria.o no.o
5
6   principal.o: principal.c
7       gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
8
9   arvorebinaria.o: arvorebinaria.c
10      gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
11
12  no.o: no.c
13      gcc -o no.o no.c -c -W -Wall -ansi -pedantic
14
15  clean:
16      rm -rf *.o *~ exec
17
18  limpa:
19      rm -rf *.o
```

makefile

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Possíveis execuções:
make
make alvo

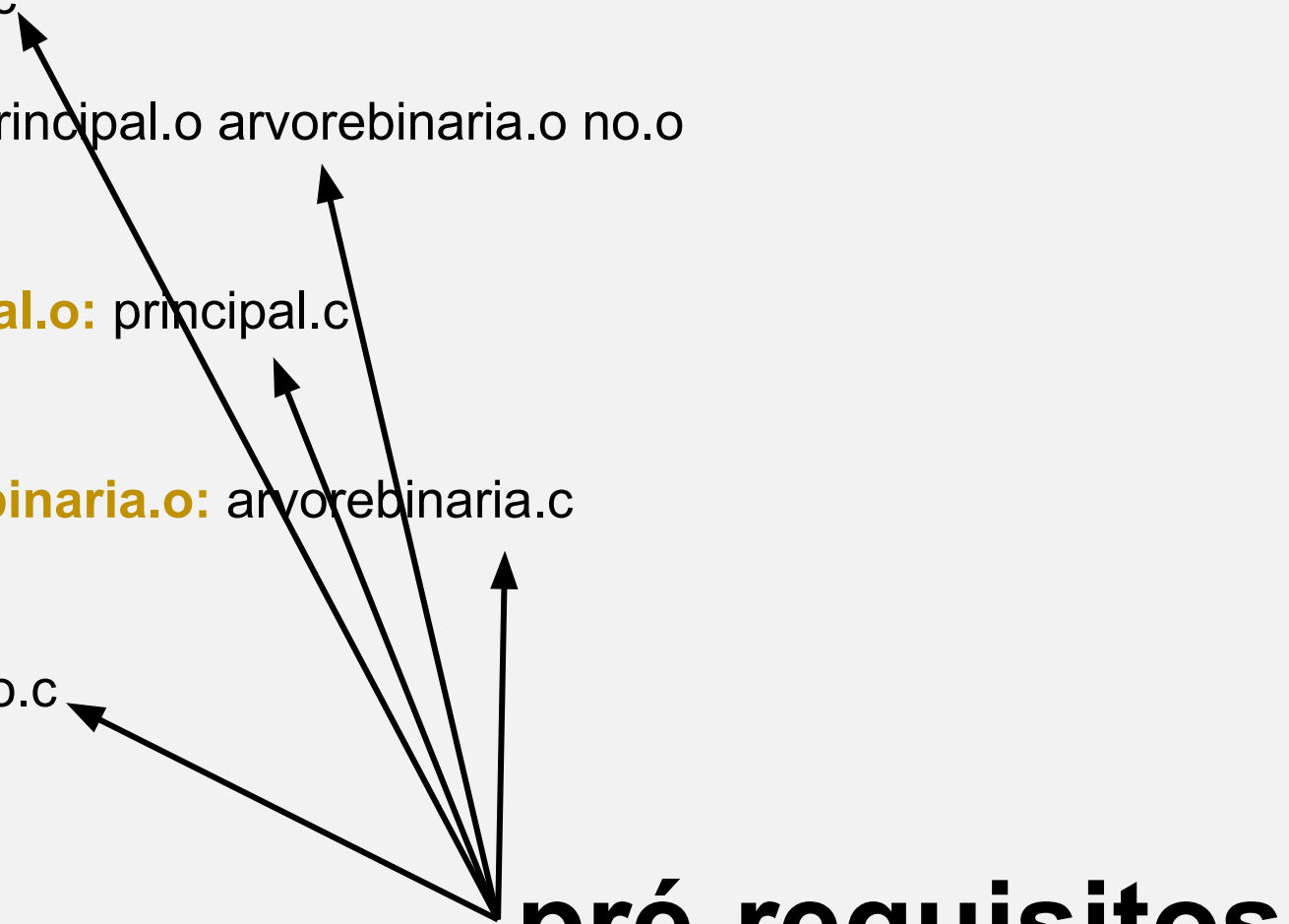
makefile

```
1   all:
2
3   exec:
4       gcc -o exec principal.o arvorebinaria.o no.o
5
6   principal.o:
7       gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
8
9   arvorebinaria.o:
10      gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
11
12  no.o:
13      gcc -o no.o no.c -c -W -Wall -ansi -pedantic
14
15  clean:
16      rm -rf *.o *~ exec
17
18  limpa:
19      rm -rf *.o
```

comandos

makefile

```
1   all: exec
2
3   exec: principal.o arvorebinaria.o no.o
4
5
6   principal.o: principal.c
7
8
9   arvorebinaria.o: arvorebinaria.c
10
11
12  no.o: no.c
13
14
15  clean:
16
17
18  limpa:
19
```



makefile

```
1   all: exec
2
3   exec: principal.o arvorebinaria.o no.o
4       gcc -o exec principal.o arvorebinaria.o no.o
5
6   principal.o: principal.c
7       gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
8
9   arvorebinaria.o: arvorebinaria.c
10      gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
11
12  no.o: no.c
13      gcc -o no.o no.c -c -W -Wall -ansi -pedantic
14
15  clean:
16      rm -rf *.o *~ exec
17
18  limpa:
19      rm -rf *.o
```

Exercício Resolvido (6)

- Na pasta binariaC, digite a sequência de comandos abaixo e explique a saída na tela

1) make all ; ls
2) make clean ; ls
3) make ; ls
4) make clean ; ls
5) make exec ; ls
6) make limpa ; ls
7) make no.o ; ls

Exercício Resolvido (6)

- Na pasta binariaC, digite a sequência de comandos abaixo e explique a saída na tela

1) **make all ; ls**
2) **make clean ; ls**
3) make ; ls
4) make clean ; ls
5) make exec ; ls
6) make limpa ; ls
7) make no.o ; ls

```
:$  make all ; ls

gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
gcc -o no.o no.c -c -W -Wall -ansi -pedantic
gcc -o exec principal.o arvorebinaria.o no.o

arvorebinaria.c  arvorebinaria.h  arvorebinaria.o  exec  makefile
no.c  no.h  no.o  principal.c  principal.o

:$  make clean ; ls

rm -rf *.o *~ exec

arvorebinaria.c  arvorebinaria.h  makefile  no.c  no.h  principal.c
```

Exercício Resolvido (6)

- Na pasta binariaC, digite a sequência de comandos abaixo e explique a saída na tela

1) make all ; ls
2) make clean ; ls
3) **make ; ls**
4) **make clean ; ls**
5) make exec ; ls
6) make limpa ; ls
7) make no.o ; ls

```
:$  make ; ls

gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
gcc -o no.o no.c -c -W -Wall -ansi -pedantic
gcc -o exec principal.o arvorebinaria.o no.o

arvorebinaria.c  arvorebinaria.h  arvorebinaria.o  exec  makefile
no.c  no.h  no.o  principal.c  principal.o

:$  make clean ; ls

rm -rf *.o *~ exec

arvorebinaria.c  arvorebinaria.h  makefile  no.c  no.h  principal.c
```

Exercício Resolvido (6)

- Na pasta binariaC, digite a sequência de comandos abaixo e explique a saída na tela

1) make all ; ls
2) make clean ; ls
3) make ; ls
4) make clean ; ls
5) **make exec ; ls**
6) **make limpa ; ls**
7) make no.o ; ls

```
:$  make exec ; ls

gcc -o principal.o principal.c -c -W -Wall -ansi -pedantic
gcc -o arvorebinaria.o arvorebinaria.c -c -W -Wall -ansi -pedantic
gcc -o no.o no.c -c -W -Wall -ansi -pedantic
gcc -o exec principal.o arvorebinaria.o no.o

arvorebinaria.c  arvorebinaria.h  arvorebinaria.o  exec  makefile
no.c  no.h  no.o  principal.c  principal.o

:$  make limpa ; ls

rm -rf *.o *

arvorebinaria.c  arvorebinaria.h  exec makefile  no.c  no.h
principal.c
```

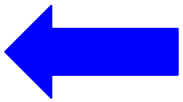
Exercício Resolvido (6)

- Na pasta binariaC, digite a sequência de comandos abaixo e explique a saída na tela

1) make all ; ls
2) make clean ; ls
3) make ; ls
4) make clean ; ls
5) make exec ; ls
6) make limpa ; ls
7) **make no.o ; ls**

```
:$  make no.o ; ls

gcc -o no.o no.c -c -W -Wall -ansi -pedantic

arvorebinaria.c  arvorebinaria.h  arvorebinaria.o  exec  makefile
no.c  no.h  no.o  principal.c  principal.o
```

Agenda

- Definições e conceitos
- Classes Nó e Árvore Bi
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- **Inserção em C com ponteiro** 
- Inserção em C++ com passagem por referência
- Estruturas híbridas

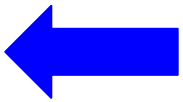
  - Estrutura arquivos
  - makefile
  - **Arquivos “no”**
  - Arquivos “arvorebinaria”

Arquivos “no”

```
//no.h

typedef struct No {
      int elemento;
   struct No *esq, *dir;
} No;

No* novoNo(int elemento);
```

```
//no.c

#include <stdlib.h>
#include "no.h"

No* novoNo(int elemento) {
   No* novo = (No*) malloc(sizeof(No));
   novo->elemento = elemento;
   novo->esq = NULL;
   novo->dir = NULL;
   return novo;
}
```

Agenda

- Definições e conceitos
- Classes Nó e Árvore Bi
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- **Inserção em C com ponteiro** 
- Inserção em C++ com passagem por referência
- Estruturas híbridas

  - Estrutura arquivos
  - makefile
  - Arquivos “no”
  - **Arquivos “arvorebinaria”**

Arquivos “arvorebinaria”

```
//arvorebinaria.h
#include "no.h"
#define bool   short
#define true   1
#define false  0

bool pesquisarRec(int, No*);
void caminharCentralRec(No*);
void caminharPreRec(No*);
void caminharPosRec(No*);
void inserirRec(int, No**);
void removerRec(int, No**);
void antecessor(No**, No**);

void start();
bool pesquisar(int);
void caminharCentral();
void caminharPre();
void caminharPos();
void inserir(int);
void remover(int);
```

```
//arvorebinaria.c
#include "no.h"
#include <err.h>
#include <stdlib.h>
#include <stdio.h>
#include "arvorebinaria.h"

No* raiz;

void start() {
   raiz = NULL;
}
```

■ ■ ■

Arquivos “arvorebinaria”

```
//arvorebinaria.h
#include "no.h"
#define bool   short
#define true   1
#define false  0

bool pesquisarRec(int, No*);
void caminharCentralRec(No*);
void caminharPreRec(No*);
void caminharPosRec(No*);
void inserirRec(int, No**);
void removerRec(int, No**);
void antecessor(No**, No**);

void start();
bool pesquisar(int);
void caminharCentral();
void caminharPre();
void caminharPos();
void inserir(int);
void remover(int);
```

Como o C tem apenas a passagem de parâmetros por valor, neste material, optamos por fazer a inserção usando o endereço de ponteiro

Poderíamos, também, usar as duas estratégias implementadas em nosso código Java

Implementação da Função Inserir

- Anteriormente, em Java, apresentamos duas implementações do inserir()

```
No inserir(int x, No i) //Java
void inserir(int x, No i, No pai) //Java
```

- As implementações correspondentes em C seriam:

```
No* inserir(int x, No* i) //C
void inserir(int x, No* i, No* pai) //C
```

Implementação da Função Inserir

- Anteriormente, em Java, apresentamos duas implementações do inserir()

No inserir(int x, No i) //Java

void inserir(int x, No i, No pai) //Java

- As implementações correspondentes em C seriam:

No* inserir(int x, No* i) //C

void inserir(int x, No* i, No* pai) //C

Primeira Opção para o Inserir em C/Java

```java
//código em Java

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

No inserir(int x, No i) {
   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(“Erro!”);
   }
   return i;
}
```

```c
//código em C

void inserir(int x) {
   raiz = inserirRec(x, raiz);
}

No* inserirRec(int x, No* i) {
   if (i == NULL) {
      i = novoNo(x);
   } else if (x < i->elemento) {
      i->esq = inserirRec(x, i->esq);
   } else if (x > i->elemento) {
      i->dir = inserirRec(x, i->dir);
   } else {
      errx(1, "Erro ao inserir!");
   }
   return i;
}
```

Implementação da Função Inserir

- Anteriormente, em Java, apresentamos duas implementações do inserir()

```
No inserir(int x, No i) //Java
void inserir(int x, No i, No pai) //Java
```

- As implementações correspondentes em C seriam:

```
No* inserir(int x, No* i) //C
void inserir(int x, No* i, No* pai) //C
```

Segunda Opção para o Inserir em C/Java

```java
//código em Java
void inserirPai(int x) {
   if (raiz == null) {
      raiz = new No(x);
   } else if (x < raiz.elemento) {
      inserirPai(x, raiz.esq, raiz);
   } else if (x > raiz.elemento)  {
      inserirPai(x, raiz.dir, raiz);
   } else {   throw new("Erro!");
} }

 void inserirPai(int x, No i, No pai)   {
   if (i == null) {
      If (x < pai.elemento){ pai.esq = new No(x);
      } else {                    pai.dir = new No(x);  }
   } else if (x < i.elemento) {
      inserirPai(x, i.esq, i);
   } else if (x > i.elemento)  {
      inserirPai(x, i.dir, i);
   } else {   throw new("Erro!");
}  }
```

```c
//código em C
void inserirPai(int x) {
   if(raiz == NULL){
      raiz = novoNo(x);
   } else if(x < raiz->elemento){
      inserirPaiRec(x, raiz->esq, raiz);
   } else if(x > raiz.elemento){
      inserirPaiRec(x, raiz->dir, raiz);
   } else {   errx(1, "Erro ao inserir!");
} }

void inserirPaiRec(int x, No* i, No* pai) {
   if (i == NULL) {
      if(x < i->elemento){ pai->esq = novoNo(x);
      } else {                    pai->dir = novoNo(x); }
   } else if (x < i->elemento) {
      inserirPaiRec(x, i->esq, i);
   } else if (x > i->elemento) {
      inserirPaiRec(x, i->dir, i);
   } else {   errx(1, "Erro ao inserir!");
}  }
```

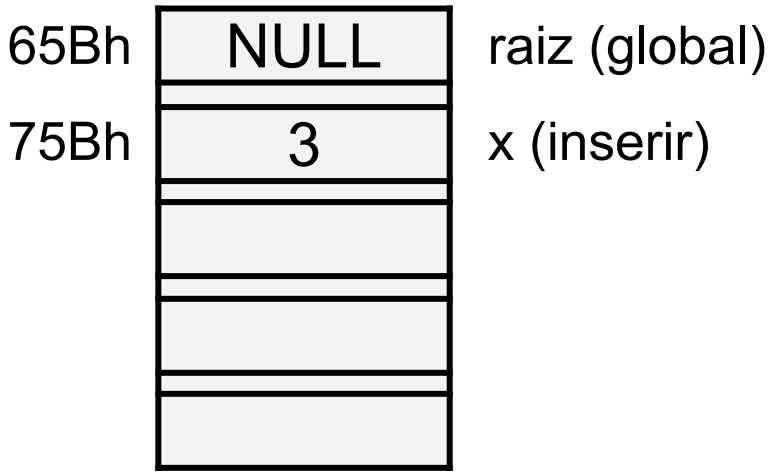
Arquivos “arvorebinaria”

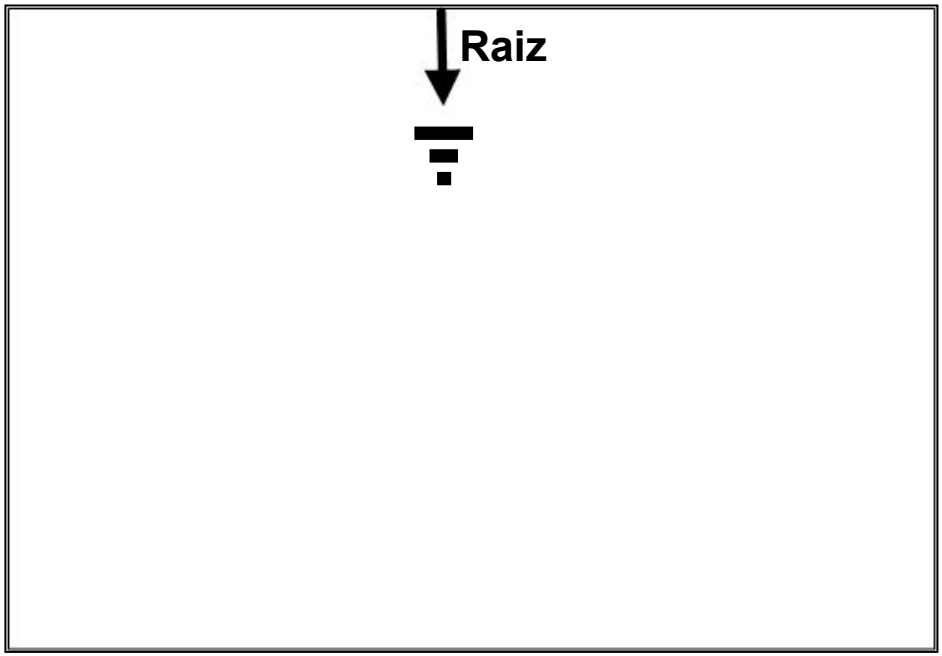
```
//arvorebinaria.h
#include "no.h"
#define bool   short
#define true   1
#define false  0

bool pesquisarRec(int, No*);
void caminharCentralRec(No*);
void caminharPreRec(No*);
void caminharPosRec(No*);
void inserirRec(int, No**);
void removerRec(int, No**);
void antecessor(No**, No**);

void start();
bool pesquisar(int);
void caminharCentral();
void caminharPre();
void caminharPos();
void inserir(int);
void remover(int);
```

```
//arvorebinaria.c
■ ■ ■

void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```

Arquivos “arvorebinaria”

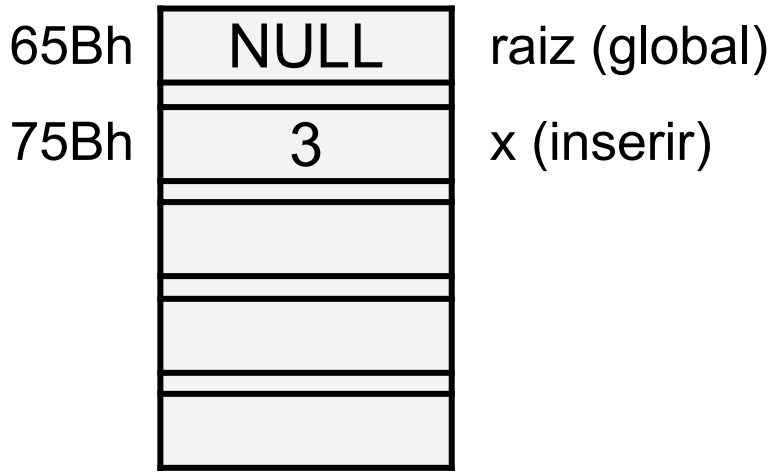
```
//arvorebinaria.c  (supondo inserir o 3)


void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```

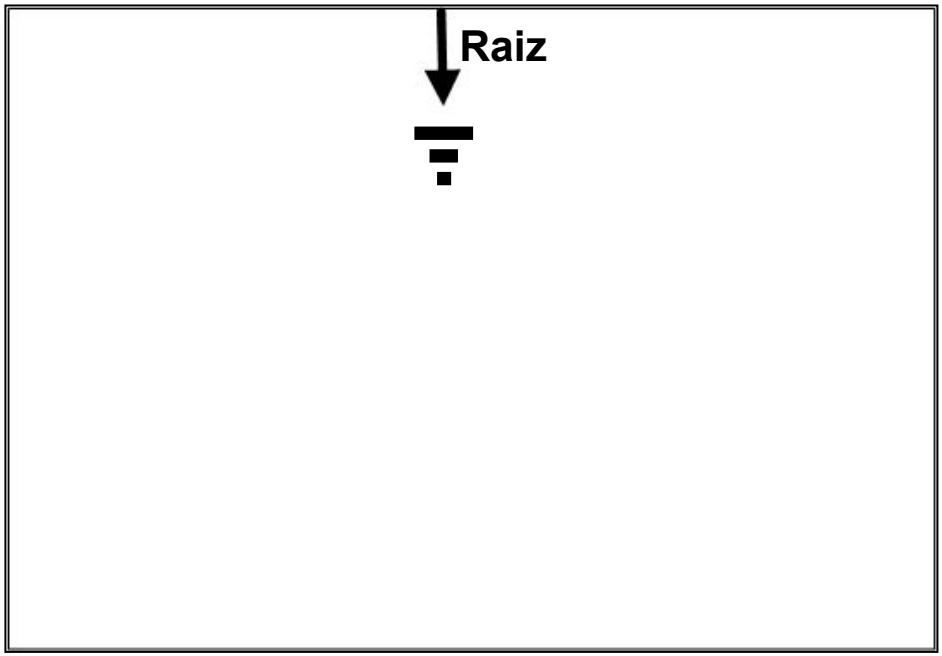




Arquivos “arvorebinaria”

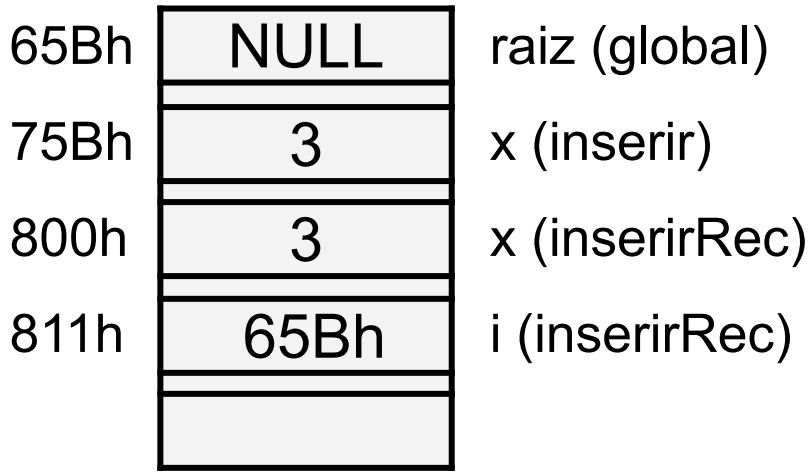
```
//arvorebinaria.c  (supondo inserir o 3)


void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```

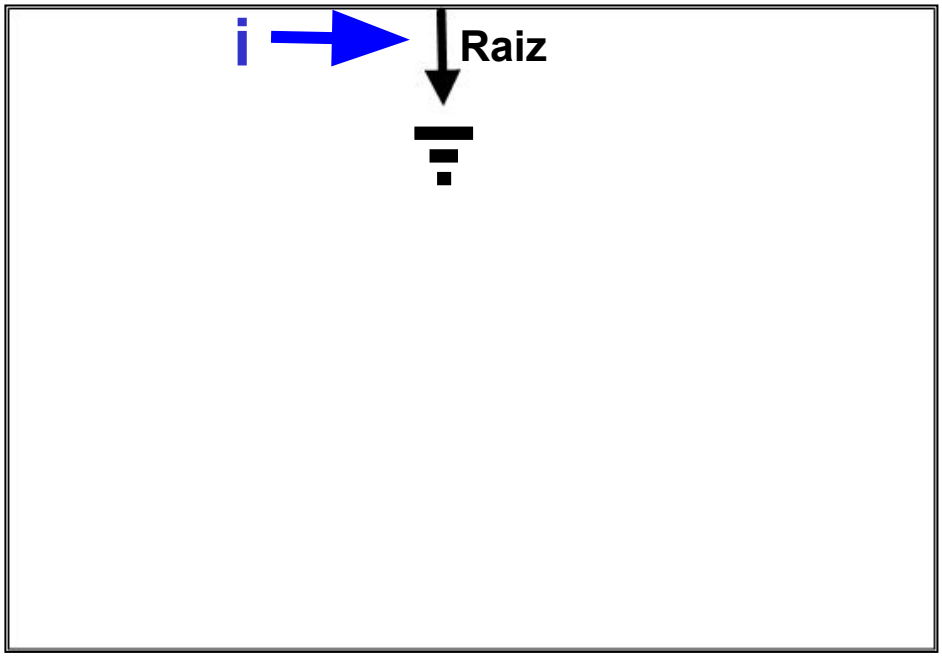




Arquivos “arvorebinaria”

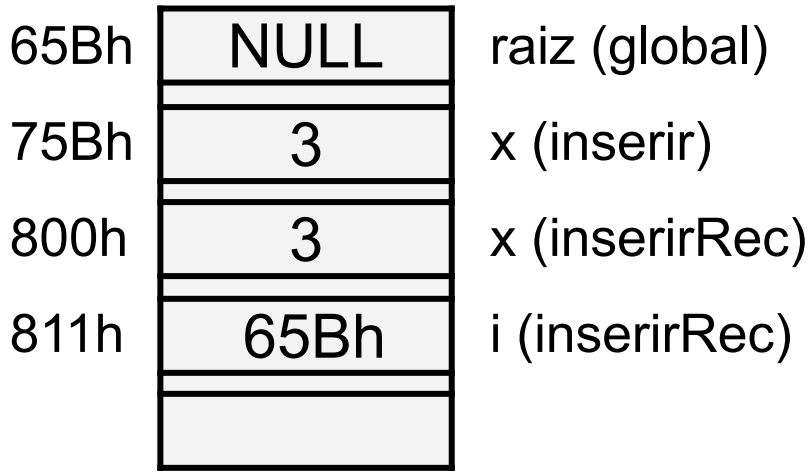
```
//arvorebinaria.c  (supondo inserir o 3)


void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```

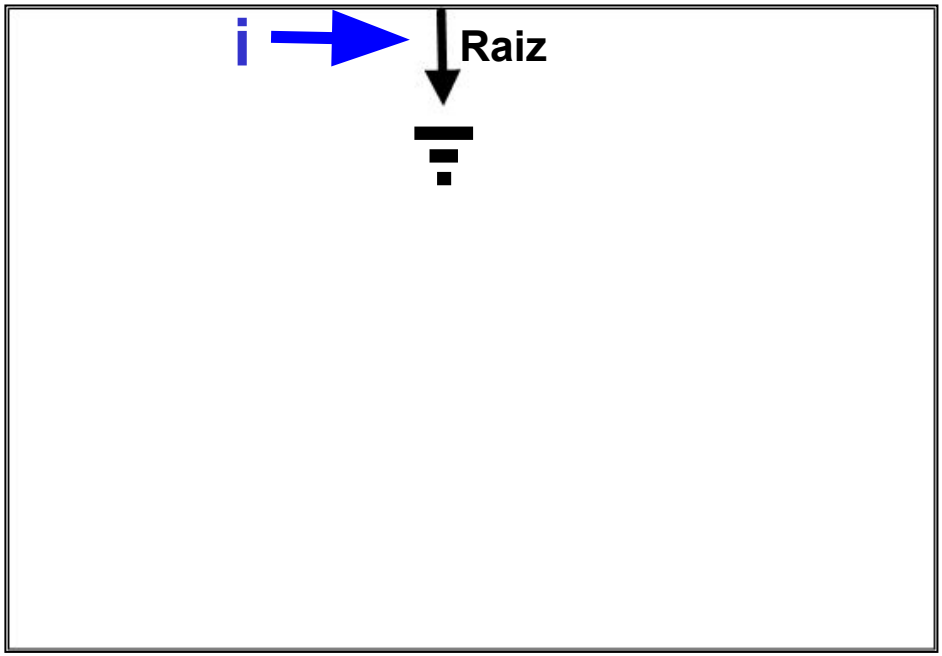




Arquivos “arvorebinaria”

```
//arvorebinaria.c  (supondo inserir o 3)


void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```

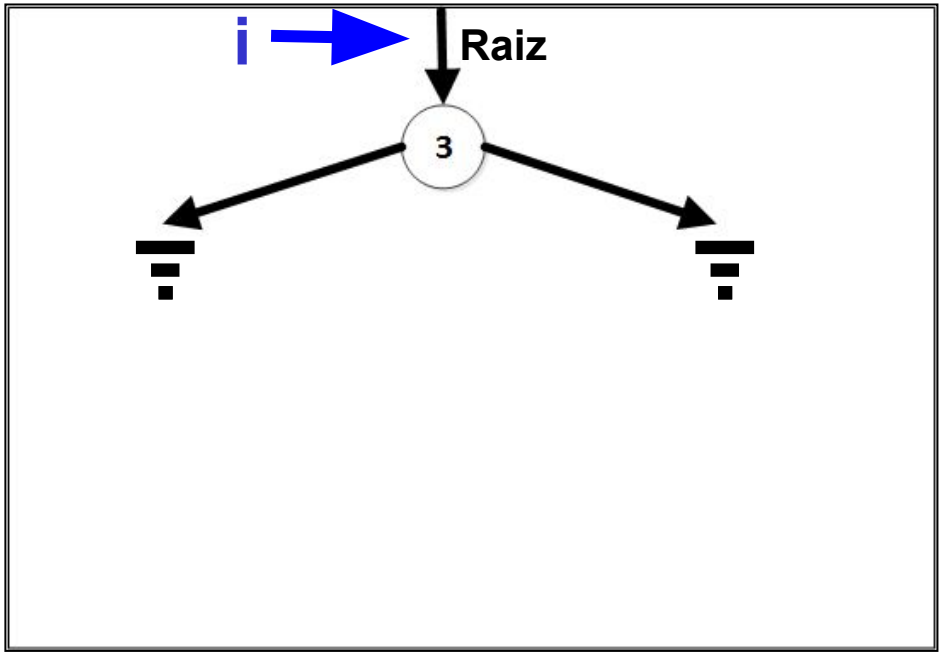




Arquivos “arvorebinaria”

```
//arvorebinaria.c  (supondo inserir o 3)


void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```

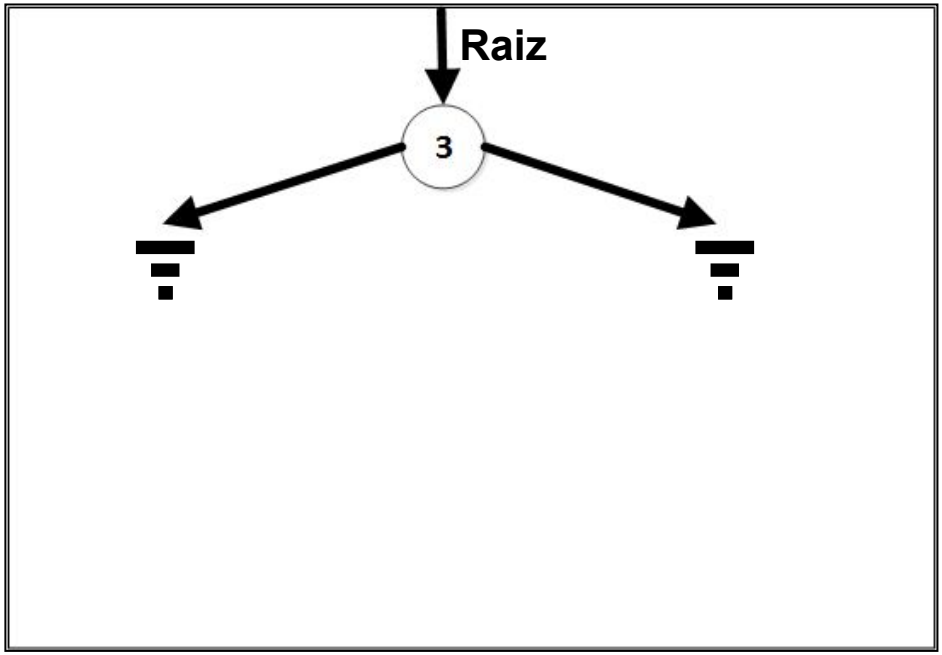
65Bh	922h	raiz (global)
75Bh	3	x (inserir)
800h	3	x (inserirRec)
811h	65Bh	i (inserirRec)
922h	3/null/null	(novoNo)



Arquivos “arvorebinaria”

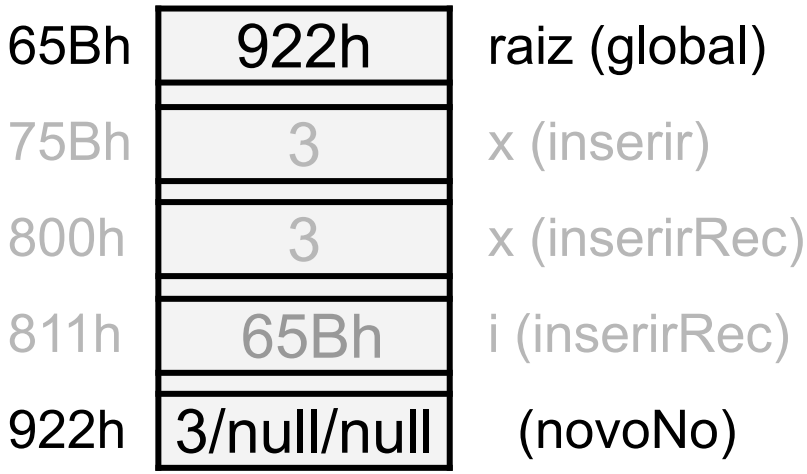
```
//arvorebinaria.c  (supondo inserir o 3)


void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```

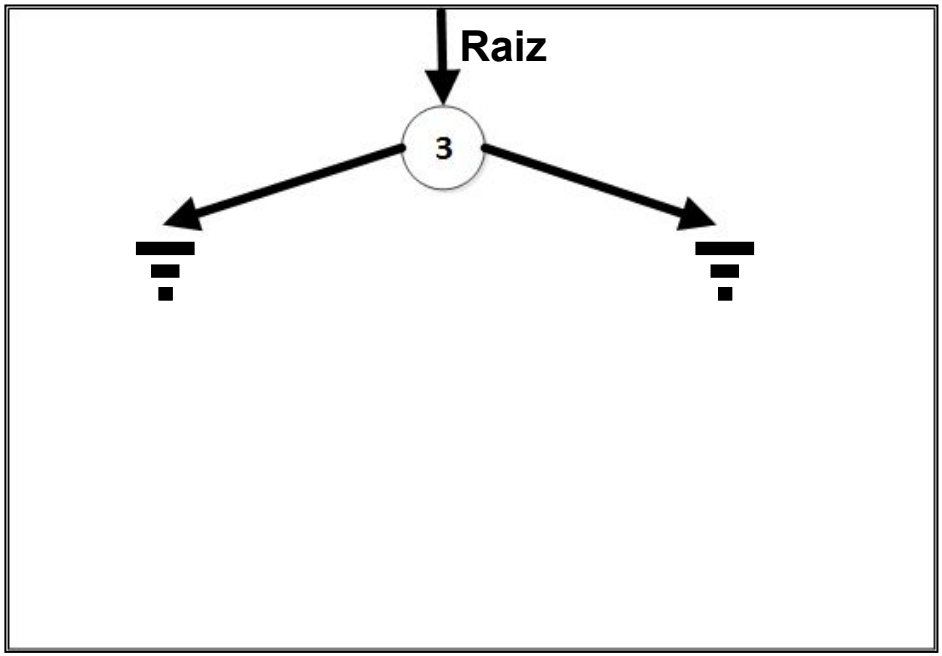
65Bh	922h	raiz (global)
75Bh	3	x (inserir)
800h	3	x (inserirRec)
811h	65Bh	i (inserirRec)
922h	3/null/null	(novoNo)



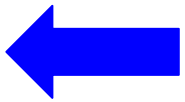
Arquivos “arvorebinaria”

```
//arvorebinaria.c  (supondo inserir o 3)


void inserir(int x) {
   inserirRec(x, &raiz);
}

void inserirRec(int x, No** i) {
   if (*i == NULL) {
      *i = novoNo(x);
   } else if (x < (*i)->elemento) {
      inserirRec(x, &((*i)->esq));
   } else if (x > (*i)->elemento) {
      inserirRec(x, &((*i)->dir));
   } else {
      errx(1, "Erro ao inserir!");
   }
}
```





Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- Inserção em C com ponteiro
- **Inserção em C++ com passagem por referência** 
- Estruturas híbridas

  - **Passagem de Parâmetros por Referência (C++)**
  - Estrutura dos Arquivos
  - Classe Nó
  - Classe ArvoreBinaria

Passagem de Parâmetro

- As linguagens de programação normalmente permitem as passagens de parâmetro por valor e por referência

- A Linguagem C (como o Java) permite somente a passagem de parâmetro por valor

- Na passagem de parâmetros por valor, passamos apenas o valor e qualquer alteração no método chamado não será refletida no que chama

Passagem de Parâmetro

- Na passagem por referência, passamos uma referência fazendo com que qualquer alteração no método chamado seja refletida no que chama

- Nesse caso, o argumento do método chamado ocupa a mesma área de memória da variável correspondente no método que chama

- Por exemplo, as linguagens C++ e C# possuem a passagem de parâmetros por referência

Passagem de Parâmetro

- Um erro comum na linguagem C (no Java também) é achar que ela tem a passagem de parâmetros por referência e essa confusão acontece quando o argumento é um ponteiro

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```

Exemplo de Passagem por Valor com Ponteiro

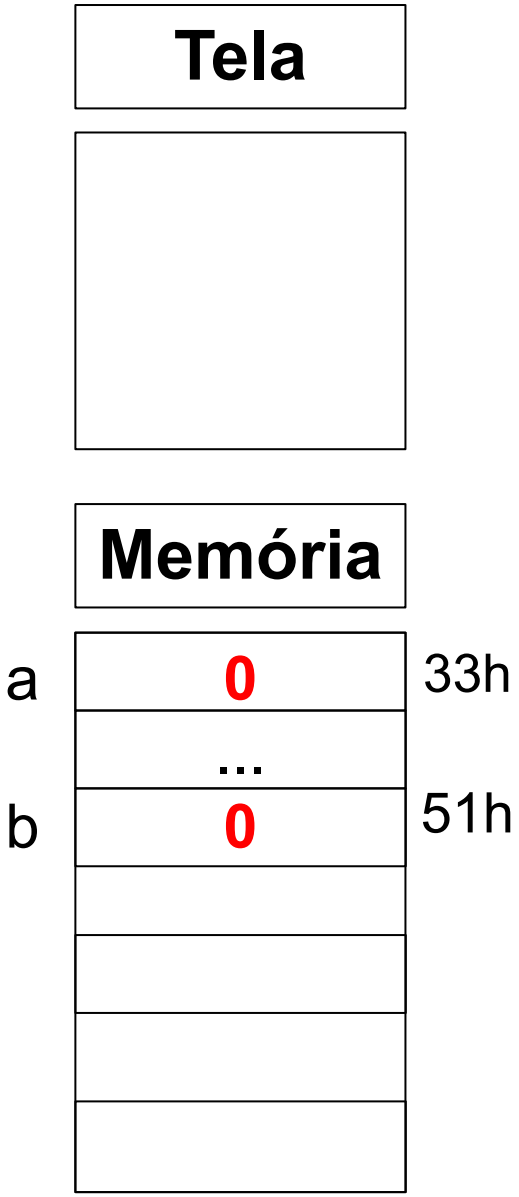
```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```

Tela

Memória

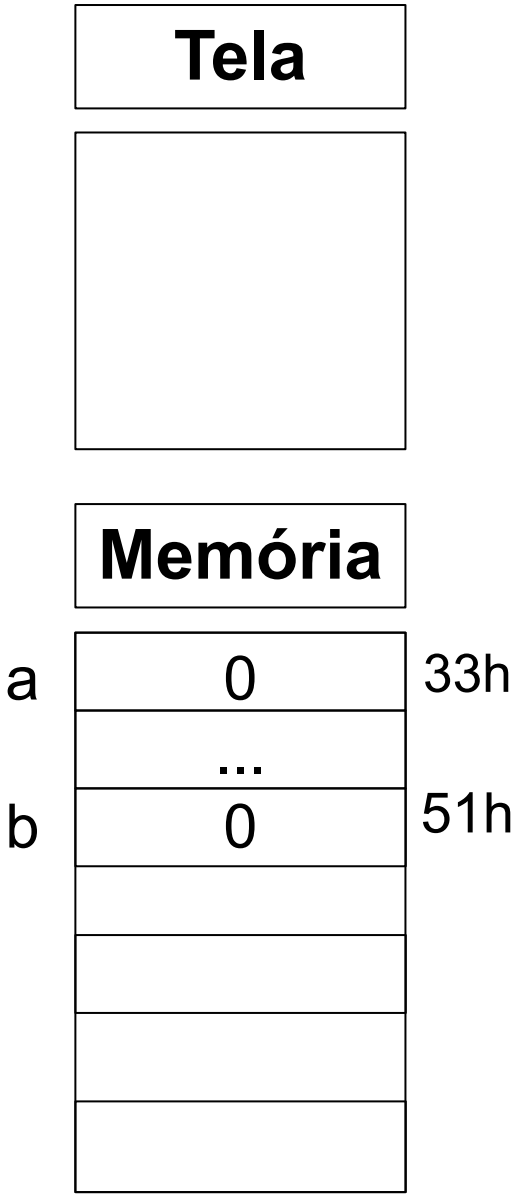
Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```



Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```



Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
        *a = *a + 1;
        b = b + 1;
        printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
        int a = 0, b = 0;
        funcao(&a, b);
        printf("\n(%i)  (%i)", a, b);
        return 0;
}
```

Tela

Memória

a	0	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	0	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```

Tela

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	0	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```

Tela

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	1	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```

Tela

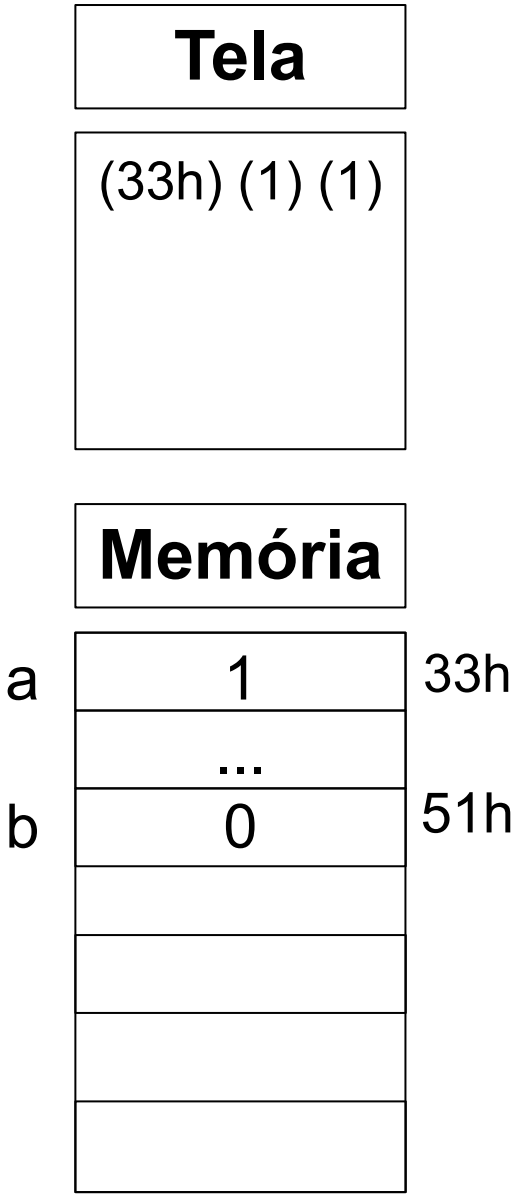
(33h) (1) (1)

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	1	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```



Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```

Tela

(33h) (1) (1)
(1)(0)

Memória

a	1	33h
	...	
b	0	51h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p)  (%i)  (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i)  (%i)", a, b);
    return 0;
}
```

Tela

(33h) (1) (1)
(1)(0)

Memória

a	1	33h
	...	
b	0	51h

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
      int z;
       printf(" %i %i ?", x, y);
      x = y = z = 2;
      printf("\n %i %i %d", x, y, z);
      return z;
}

int main(int argc, char **argv){
      int a, b, c;
      a = b = 1;
      printf(" %i %i ?", a, b);
      c = metodo(a, b);
       printf("\n %i %i %d", a, b, c);
}
```

Tela

Memória

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
     printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
     printf("\n %i %i %d", a, b, c);
}
```

Tela

Memória

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
      int z;
       printf(" %i %i ?", x, y);
      x = y = z = 2;
      printf("\n %i %i %d", x, y, z);
      return z;
}

int main(int argc, char **argv){
      int a, b, c;
      a = b = 1;
      printf(" %i %i ?", a, b);
      c = metodo(a, b);
       printf("\n %i %i %d", a, b, c);
}
```

Tela

Memória

a	?
b	?
c	?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
     printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
     printf("\n %i %i %d", a, b, c);
}
```

Tela

Memória

a	1
b	1
c	?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
     printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
     printf("\n %i %i %d", a, b, c);
}
```

Tela

1 1 ?

Memória

a	1
b	1
c	?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
      int z;
       printf(" %i %i ?", x, y);
      x = y = z = 2;
      printf("\n %i %i %d", x, y, z);
      return z;
}

int main(int argc, char **argv){
      int a, b, c;
      a = b = 1;
      printf(" %i %i ?", a, b);
      c = metodo(a, b);
       printf("\n %i %i %d", a, b, c);
}
```

Tela

1 1 ?

Memória

a	1
b	1
c	?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
        int z;
         printf(" %i %i ?", x, y);
        x = y = z = 2;
        printf("\n %i %i %d", x, y, z);
        return z;
}

int main(int argc, char **argv){
        int a, b, c;
        a = b = 1;
        printf(" %i %i ?", a, b);
        c = metodo(a, b);
         printf("\n %i %i %d", a, b, c);
}
```

Tela

1 1 ?

Memória

a/x	1
b	1
c	?
y	1

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
     printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
     printf("\n %i %i %d", a, b, c);
}
```

Tela

1 1 ?

Memória

a/x	1
b	1
c	?
y	1
z	?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
     printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
     printf("\n %i %i %d", a, b, c);
}
```

Tela

1	1	?
1	1	?

Memória

a/x	1
b	1
c	?
y	1
z	?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
      int z;
       printf(" %i %i ?", x, y);
      x = y = z = 2;
      printf("\n %i %i %d", x, y, z);
      return z;
}

int main(int argc, char **argv){
      int a, b, c;
      a = b = 1;
      printf(" %i %i ?", a, b);
      c = metodo(a, b);
       printf("\n %i %i %d", a, b, c);
}
```

Tela

1	1	?
1	1	?

Memória

a/x	2
b	1
c	?
y	2
z	2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
     printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
     printf("\n %i %i %d", a, b, c);
}
```

Tela

1	1	?
1	1	?
2	2	2

Memória

a/x	2
b	1
c	?
y	2
z	2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
     printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
     printf("\n %i %i %d", a, b, c);
}
```

Tela

1	1	?
1	1	?
2	2	2

Memória

a/x	2
b	1
c	?
y	2
z	2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
      int z;
       printf(" %i %i ?", x, y);
      x = y = z = 2;
      printf("\n %i %i %d", x, y, z);
      return z;
}

int main(int argc, char **argv){
      int a, b, c;
      a = b = 1;
      printf(" %i %i ?", a, b);
      c = metodo(a, b);
       printf("\n %i %i %d", a, b, c);
}
```

Tela

1	1	?
1	1	?
2	2	2

Memória

a/x	2
b	1
c	2
y	2
z	2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
      int z;
       printf(" %i %i ?", x, y);
      x = y = z = 2;
      printf("\n %i %i %d", x, y, z);
      return z;
}

int main(int argc, char **argv){
      int a, b, c;
      a = b = 1;
      printf(" %i %i ?", a, b);
      c = metodo(a, b);
       printf("\n %i %i %d", a, b, c);
}
```

Tela

1	1	?
1	1	?
2	2	2
2	1	2

Memória

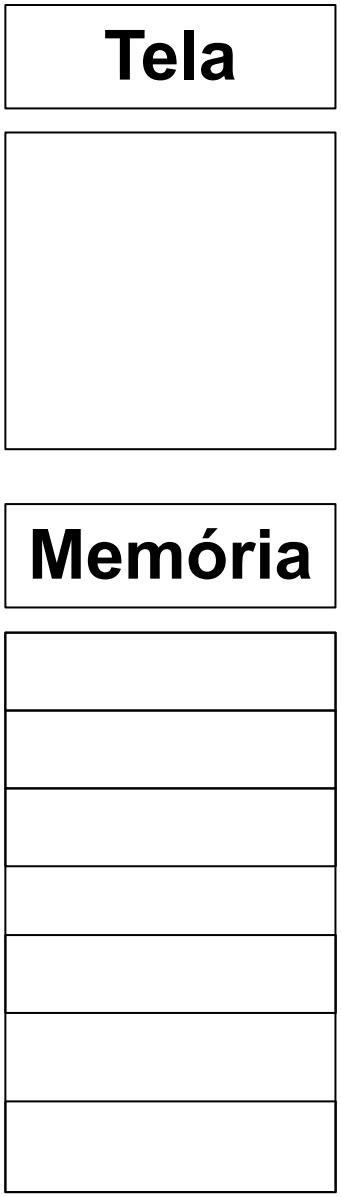
a/x	2
b	1
c	2
y	2
z	2

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```

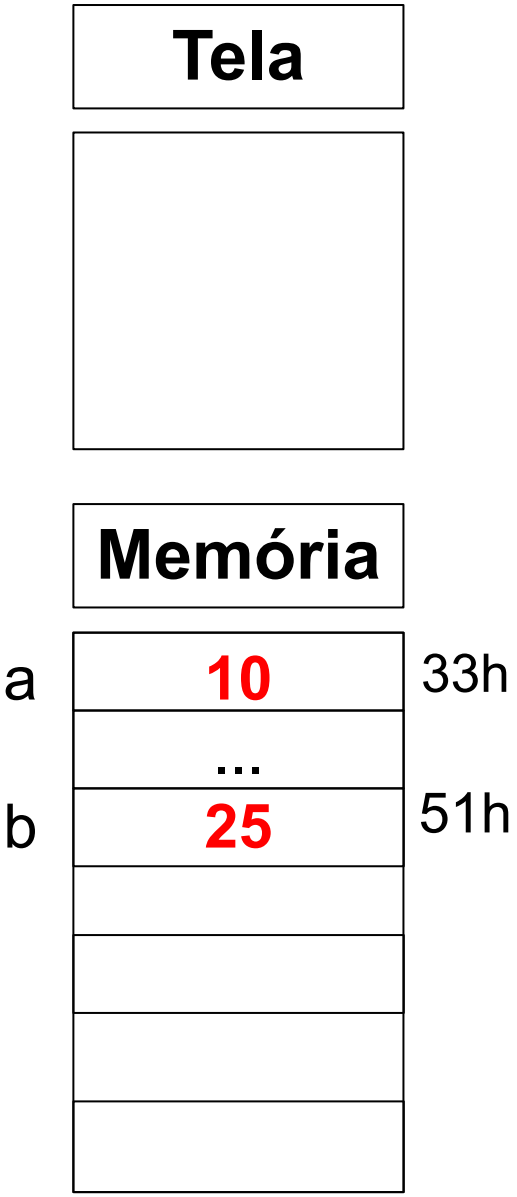
Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```



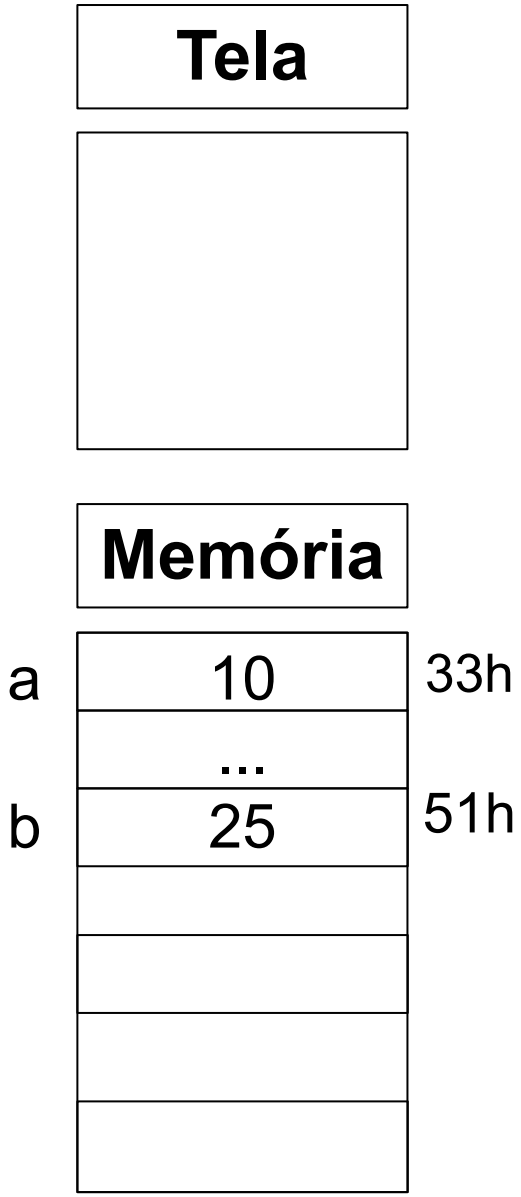
Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```



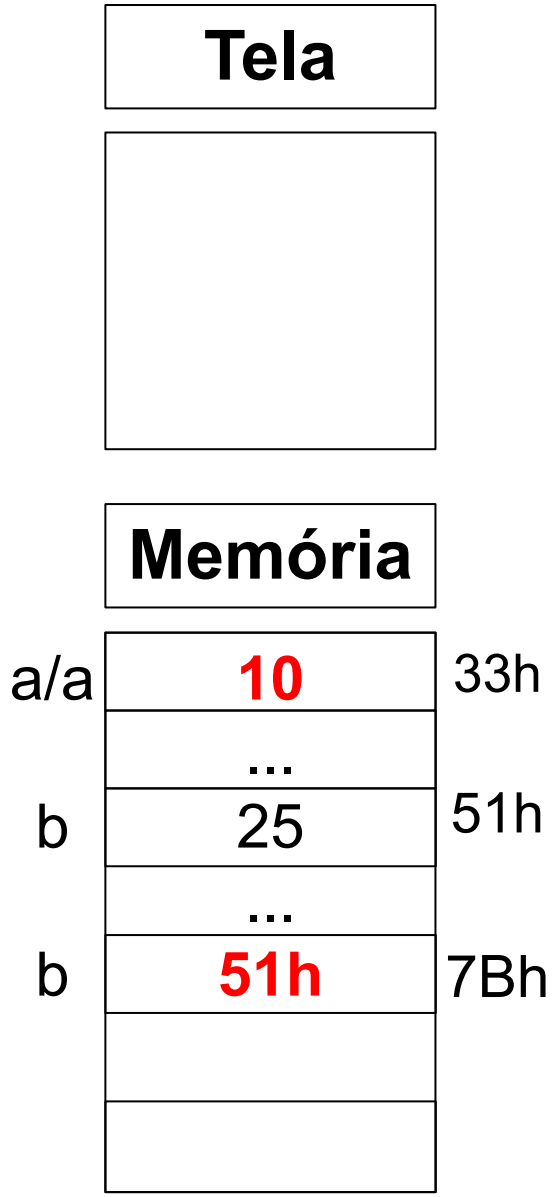
Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```



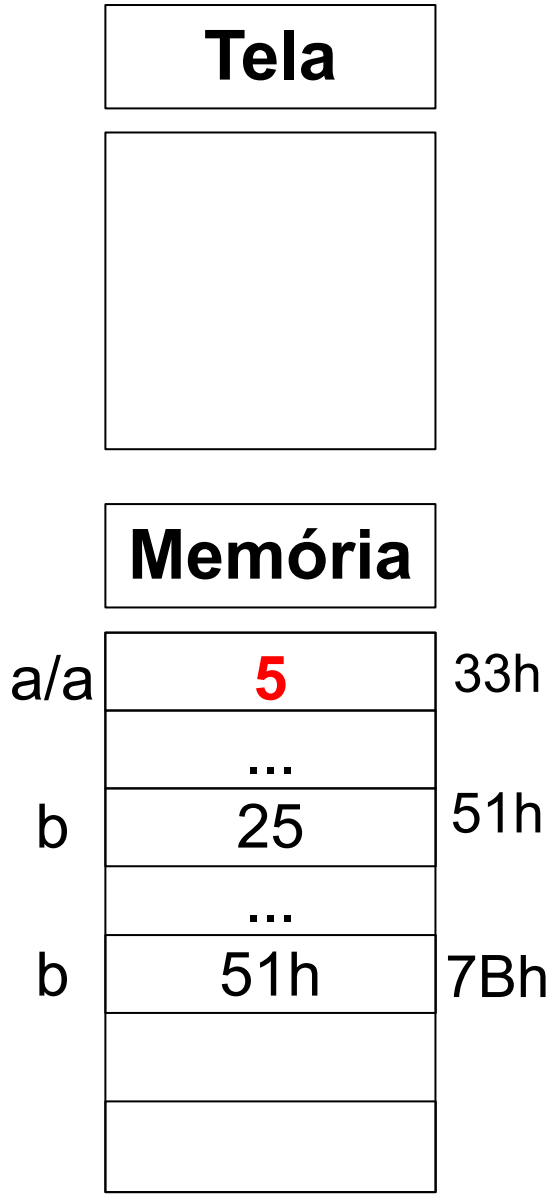
Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
        a = 5;
        *b = 6;
        int* resp = new int[3];
        resp[0] = 10; resp[1] = 20; resp[2] = 30;
        return resp;
    }

    int main(int argc, char **argv){
        int a = 10, b = 25;
        int* vet = metodo(a, &b);
        printf("%d %d %d", a, b);
        printf("%d %d %d", vet[0],  vet[1], vet[2]);
    }
```



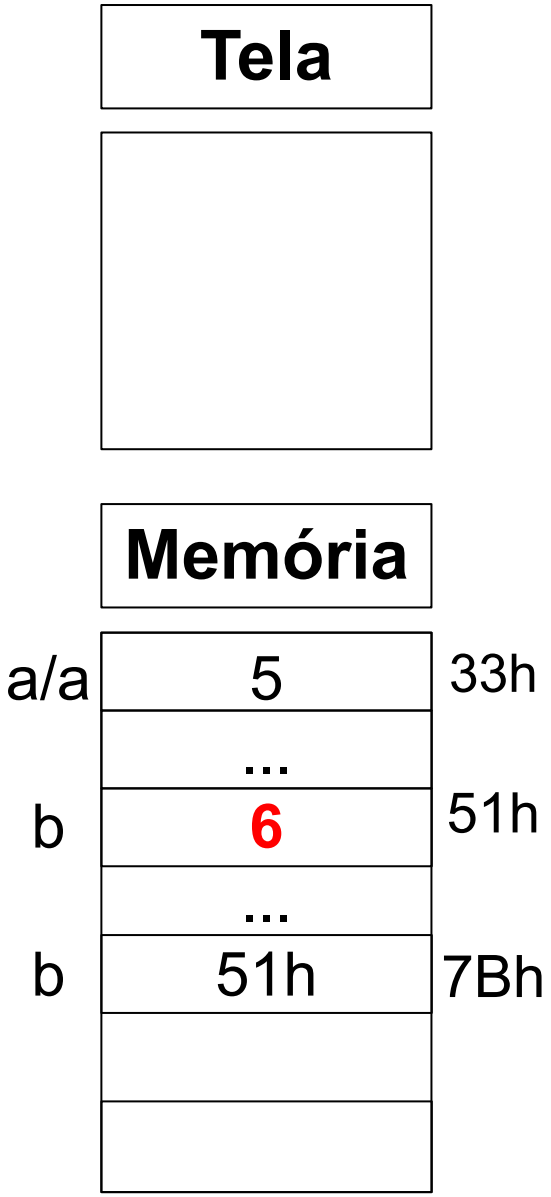
Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```



Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```



Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```

E9h

Tela

Memória

a/a	5	33h
	...	
b	6	51h
	...	
b	51h	7Bh
	...	
resp	E9h	C2h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```

10	20	30	E9h

Tela

Memória

a/a	5	33h
	...	
b	6	51h
	...	
b	51h	7Bh
	...	
resp	E9h	C2h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```

10	20	30	E9h

Tela

Memória

a/a	5	33h
	...	
b	6	51h
	...	
b	51h	7Bh
	...	
resp	E9h	C2h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```

10	20	30	E9h

Tela

Memória

a	5	33h
	...	
b	6	51h
	...	
vet	E9h	88h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```

10	20	30	E9h

Tela

5 6

Memória

a	5	33h
	...	
b	6	51h
	...	
vet	E9h	88h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){
    a = 5;
    *b = 6;
    int* resp = new int[3];
    resp[0] = 10; resp[1] = 20; resp[2] = 30;
    return resp;
}

int main(int argc, char **argv){
    int a = 10, b = 25;
    int* vet = metodo(a, &b);
    printf("%d %d %d", a, b);
    printf("%d %d %d", vet[0],  vet[1], vet[2]);
}
```

10	20	30	E9h

Tela

5 6
10 20 30

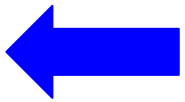
Memória

a	5	33h
	...	
b	6	51h
	...	
vet	E9h	88h

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- Inserção em C com ponteiro
- **Inserção em C++ com passagem por referência**
- Estruturas híbridas

  - Passagem de Parâmetros por Referência (C++)
  - **Estrutura dos Arquivos**
  - Classe Nó
  - Classe ArvoreBinaria



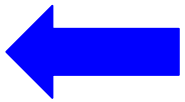
Estrutura de Arquivos

- no.h
- no.cc
- arvorebinaria.h
- arvorebinaria.cc
- principal.c
- makefile

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- Inserção em C com ponteiro
- **Inserção em C++ com passagem por referência**
- Estruturas híbridas

  - Passagem de Parâmetros por Referência (C++)
  - Estrutura dos Arquivos
  - **Classe Nó**
  - Classe ArvoreBinaria

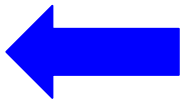


Classe Nó em C++

```
//no.h
#include <iostream>

using namespace std;

class No {
   public:
      int elemento;
      No *esq, *dir;
      No(int);
};
```

```
//no.cc
#include ”no.h”

No::No(int elemento) {
   this->elemento = elemento;
   this->esq = NULL;
   this->dir = NULL;
}
```

Agenda

- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- Inserção em C com ponteiro
- **Inserção em C++ com passagem por referência** 
- Estruturas híbridas

  - Passagem de Parâmetros por Referência (C++)
  - Estrutura dos Arquivos
  - Classe Nó
  - **Classe ArvoreBinaria**

Classe Arvore Binária em C++

```
//arvorebinaria.h
#include ”no.h”

class ArvoreBinaria {
   private:
      No* raiz;
      bool pesquisar(int, No*);
      void inserir(int, No* &);
      void caminharCentral(No*);
      void caminharPre(No*);
      void caminharPos(No*);
      void remover(int, No* &);
      void antecessor(No*, No* &);
   public:
      ArvoreBinaria();
      bool pesquisar(int);
      void inserir(int);
      void caminharCentral();
      void caminharPre();
      void caminharPos();
      void remover(int);
};
```

```
//arvorebinaria.cc
#include <err.h>
#include "arvorebinaria.h"

ArvoreBinaria::ArvoreBinaria() {
   raiz = NULL;
}
```

■ ■ ■

Classe Arvore Binária em C++

```
//arvorebinaria.h
#include ”no.h”

class ArvoreBinaria {
   private:
      No* raiz;
      bool pesquisar(int, No*);
      void inserir(int, No* &);
      void caminharCentral(No*);
      void caminharPre(No*);
      void caminharPos(No*);
      void remover(int, No* &);
      void antecessor(No*, No* &);
   public:
      ArvoreBinaria();
      bool pesquisar(int);
      void inserir(int);
      void caminharCentral();
      void caminharPre();
      void caminharPos();
      void remover(int);
};
```

As implementações apresentadas do inserir em Java/C poderiam ser usadas, contudo, neste material, exploraremos a passagem de parâmetros por referência

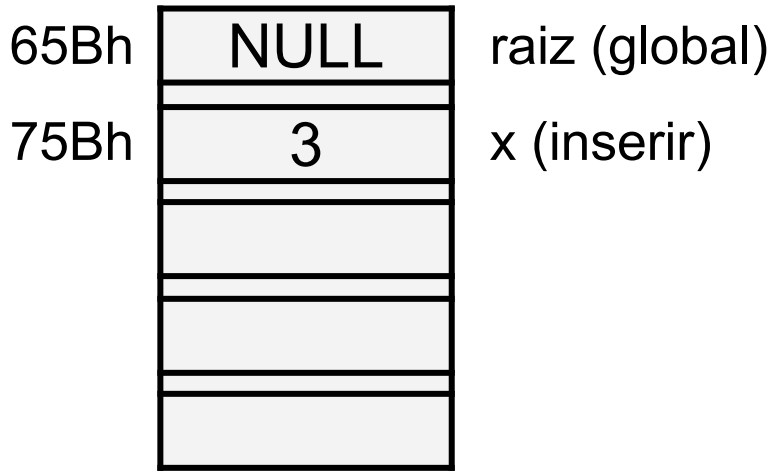
Classe Arvore Binária em C++

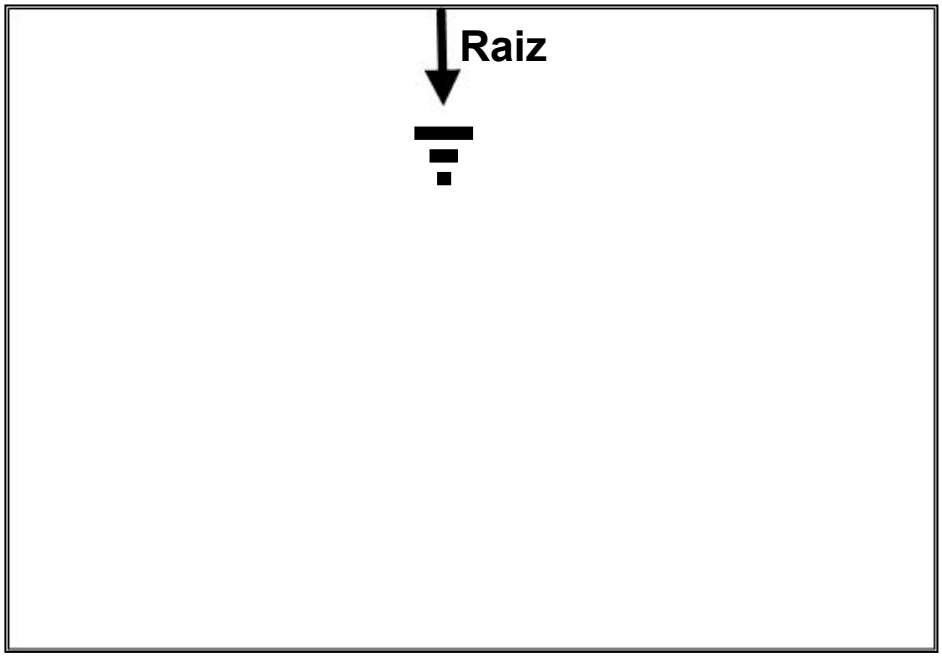
```
//arvorebinaria.h
#include ”no.h”

class ArvoreBinaria {
   private:
      No* raiz;
      bool pesquisar(int, No*);
      void inserir(int, No* &);
      void caminharCentral(No*);
      void caminharPre(No*);
      void caminharPos(No*);
      void remover(int, No* &);
      void antecessor(No*, No* &);
   public:
      ArvoreBinaria();
      bool pesquisar(int);
      void inserir(int);
      void caminharCentral();
      void caminharPre();
      void caminharPos();
      void remover(int);
};
```

```
//arvorebinaria.cc

■ ■ ■

void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```

Arquivos “arvorebinaria”

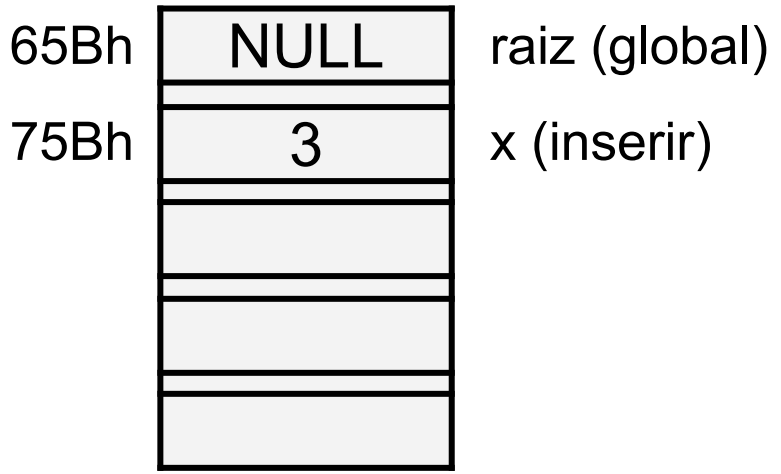
```
//arvorebinaria.cc  (supondo inserir o 3)


void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```

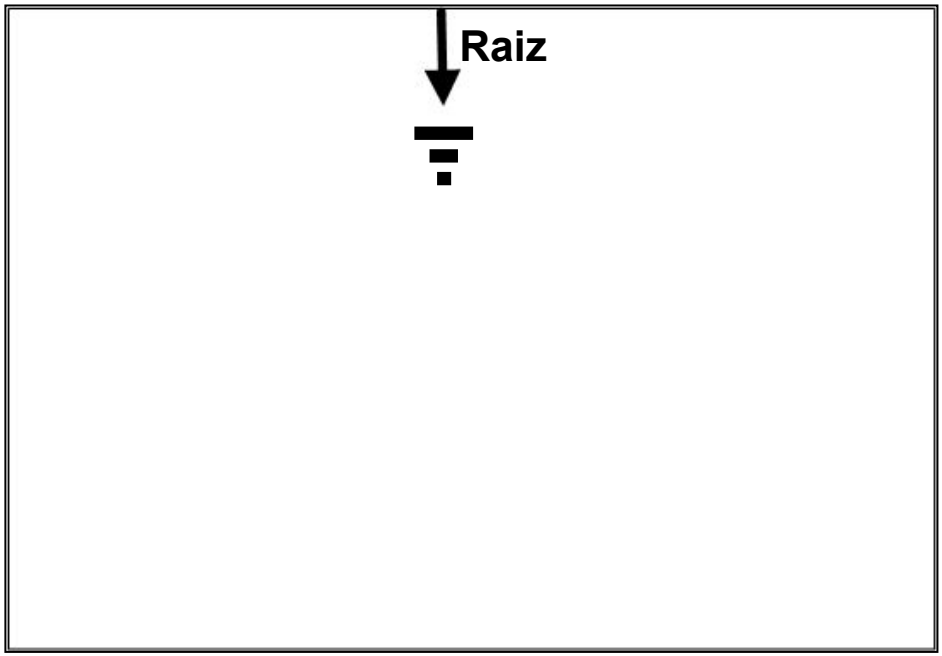




Arquivos “arvorebinaria”

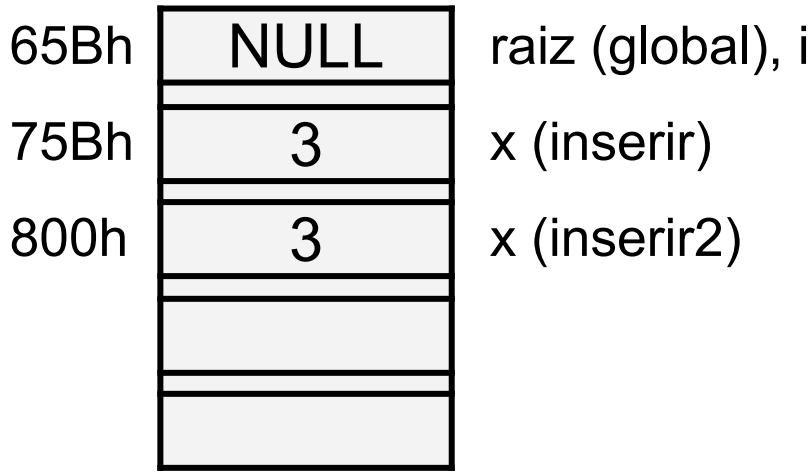
```
//arvorebinaria.cc  (supondo inserir o 3)


void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```

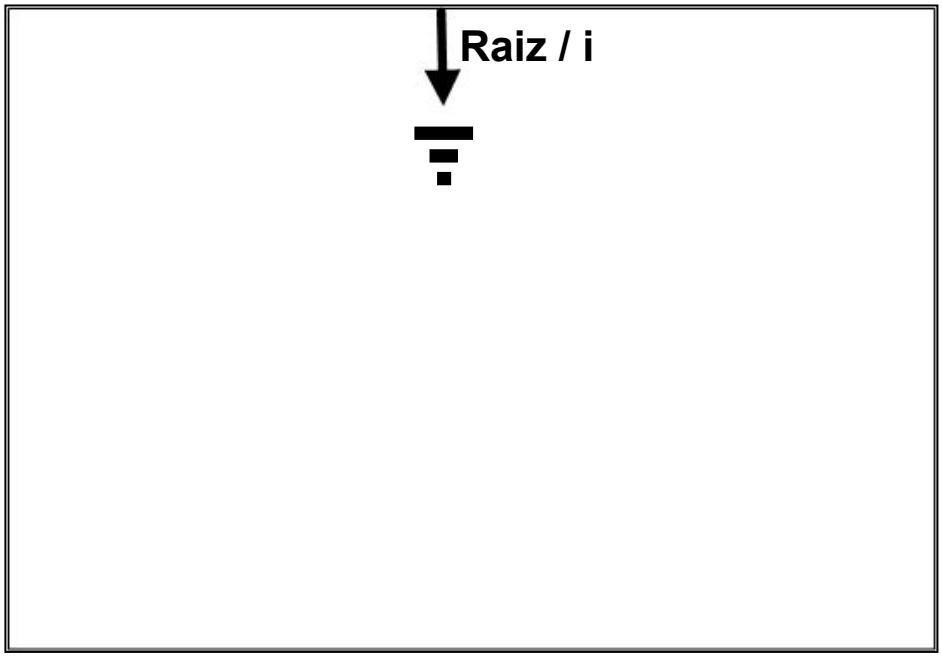




Arquivos “arvorebinaria”

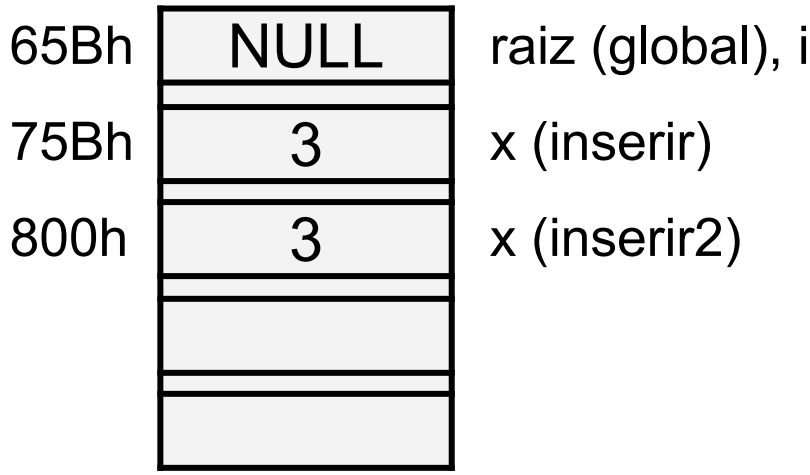
```
//arvorebinaria.cc  (supondo inserir o 3)


void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```

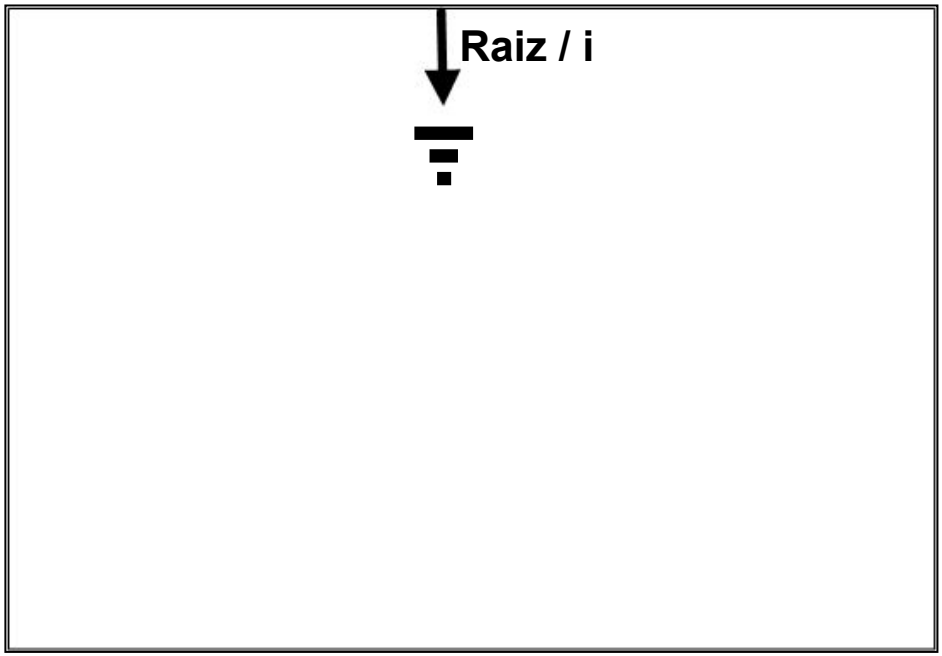




Arquivos “arvorebinaria”

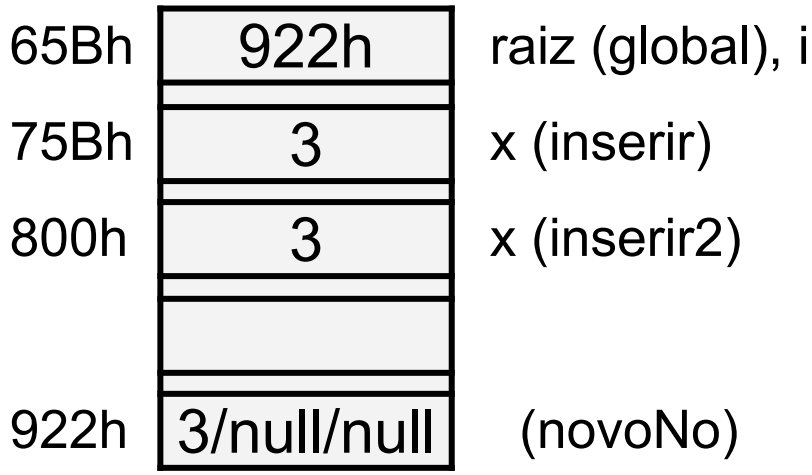
```
//arvorebinaria.cc  (supondo inserir o 3)


void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```

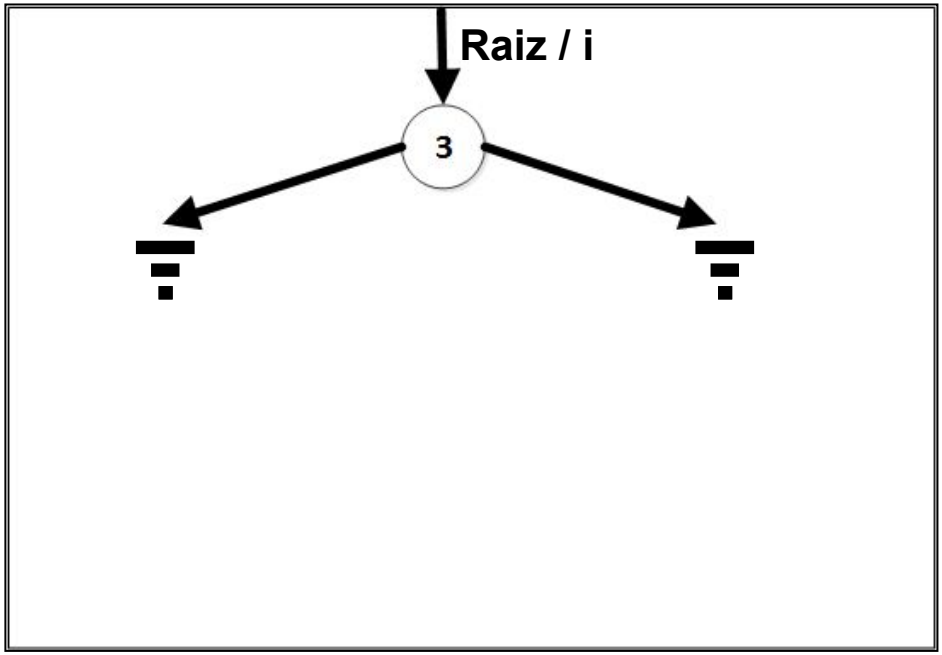




Arquivos “arvorebinaria”

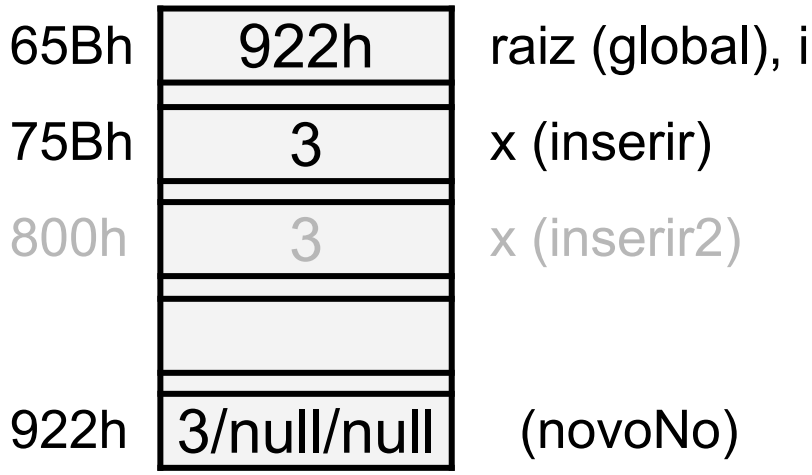
```
//arvorebinaria.cc  (supondo inserir o 3)


void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```

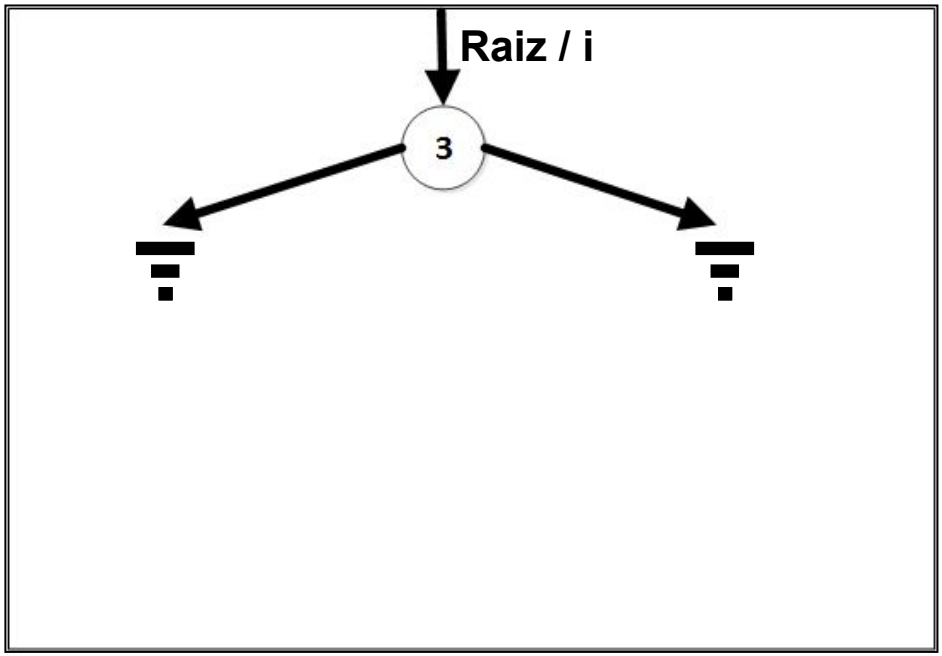




Arquivos “arvorebinaria”

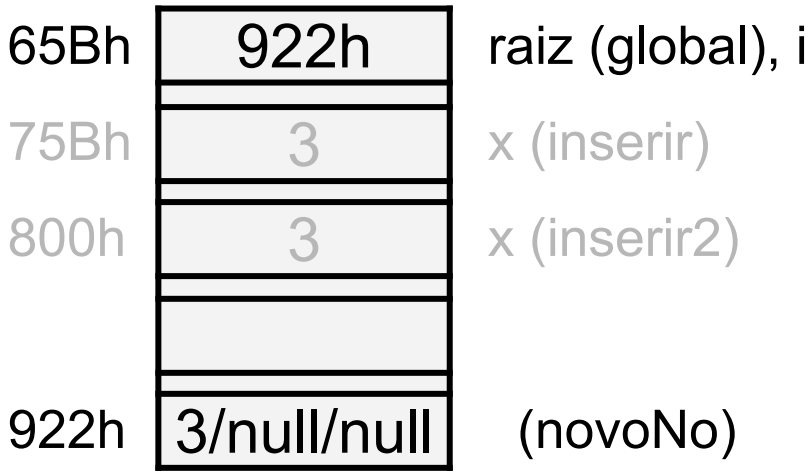
```
//arvorebinaria.cc  (supondo inserir o 3)


void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```

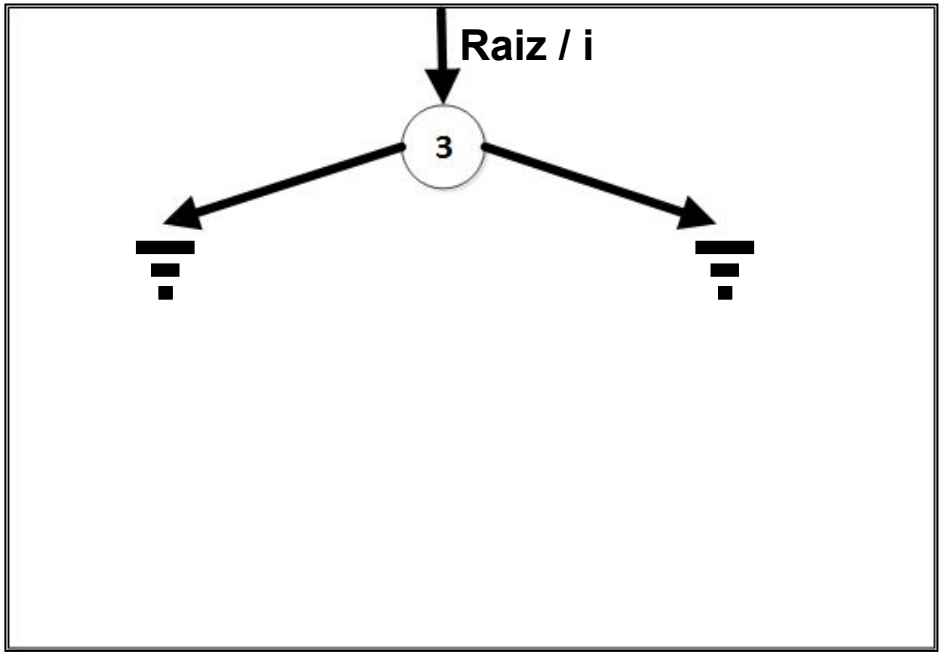




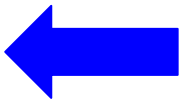
Arquivos “arvorebinaria”

```
//arvorebinaria.cc  (supondo inserir o 3)


void ArvoreBinaria::inserir(int x) {
   inserir(x, raiz);
}

void ArvoreBinaria::inserir(int x, No* &i) {
   if (i == NULL) {
      i = new No(x);

   } else if (x < i->elemento) {
      inserir(x, i->esq);

   } else if (x > i->elemento) {
         inserir(x, i->dir);

   } else {
         errx(1, "Erro ao inserir!");
   }
}
```





Agenda

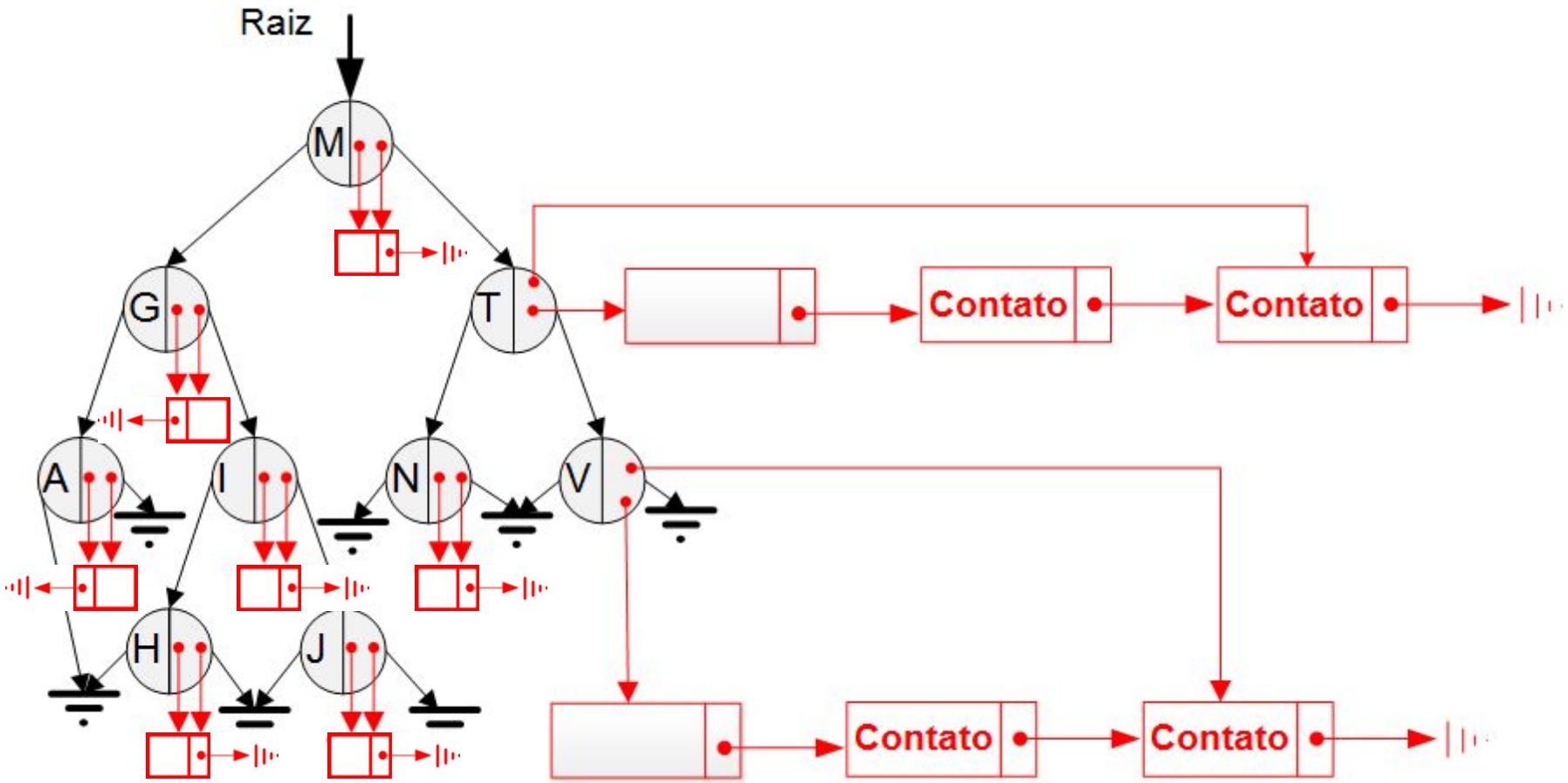
- Definições e conceitos
- Classes Nó e Árvore Binária em Java
- Inserção
- Pesquisa
- Remoção
- Caminhamento
- Inserção em C com ponteiro
- Inserção em C++ com passagem por referência
- **Estruturas híbridas** 

Exercício (10)

- Você foi contratado para desenvolver uma agenda de contatos (atributos nome, telefone, email e CPF) para um escritório de advocacia

Exercício (10)

- Um colega sugeriu implementar 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



Exercício (10)

- Crie uma classe Contato contendo os atributos String nome, telefone e email e int CPF

- Crie uma classe Celula contendo os atributos Contato contato e Celula prox

- Crie uma classe No contendo os atributos Celula primeiro e ultimo, No esq e dir, e char letra

- 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 (11)

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