

# Unidade VII: Árvore Binária - Remoção



**PUC Minas**

Instituto de Ciências Exatas e Informática  
Departamento de Ciência da Computação

## Exercício Resolvido (1)

- Faça o método **int getMaior()** que retorna o maior elemento de uma Árvore Binária

## Exercício Resolvido (1)

- Faça o método **int getMaior()** que retorna o maior elemento de uma Árvore Binária

```
public int getMaior(){  
    int resp = -1;  
  
    if(raiz != null){  
        No i;  
        for(i = raiz; i.dir != null; i = i.dir);  
        resp = i.elemento;  
    }  
  
    return resp;  
}
```

## Exercício Resolvido (2)


- Faça o método **int getMenor()** que retorna o maior elemento de uma Árvore Binária

## Exercício Resolvido (2)

- Faça o método **int getMenor()** que retorna o maior elemento de uma Árvore Binária

```
public int getMenor(){  
    int resp = -1;  
  
    if(raiz != null){  
        No i;  
        for(i = raiz; i.esq != null; i = i.esq);  
        resp = i.elemento;  
    }  
  
    return resp;  
}
```

- Funcionamento básico
- Algoritmo em C#
- Análise de complexidade

- **Funcionamento básico** 
- Algoritmo em C#
- Análise de complexidade

# Funcionamento Básico da Remoção

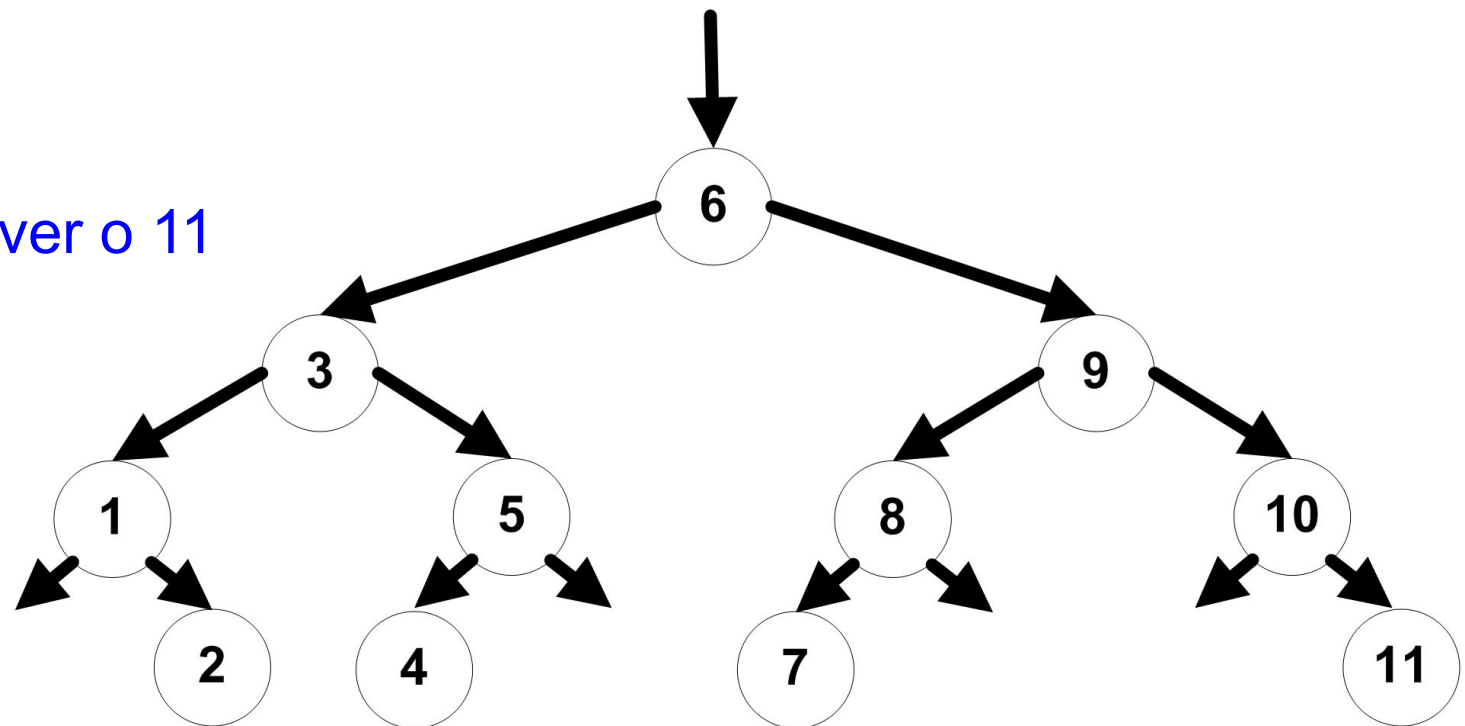
(1) Se o elemento estiver em uma **folha**, removê-la



# Funcionamento Básico da Remoção

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

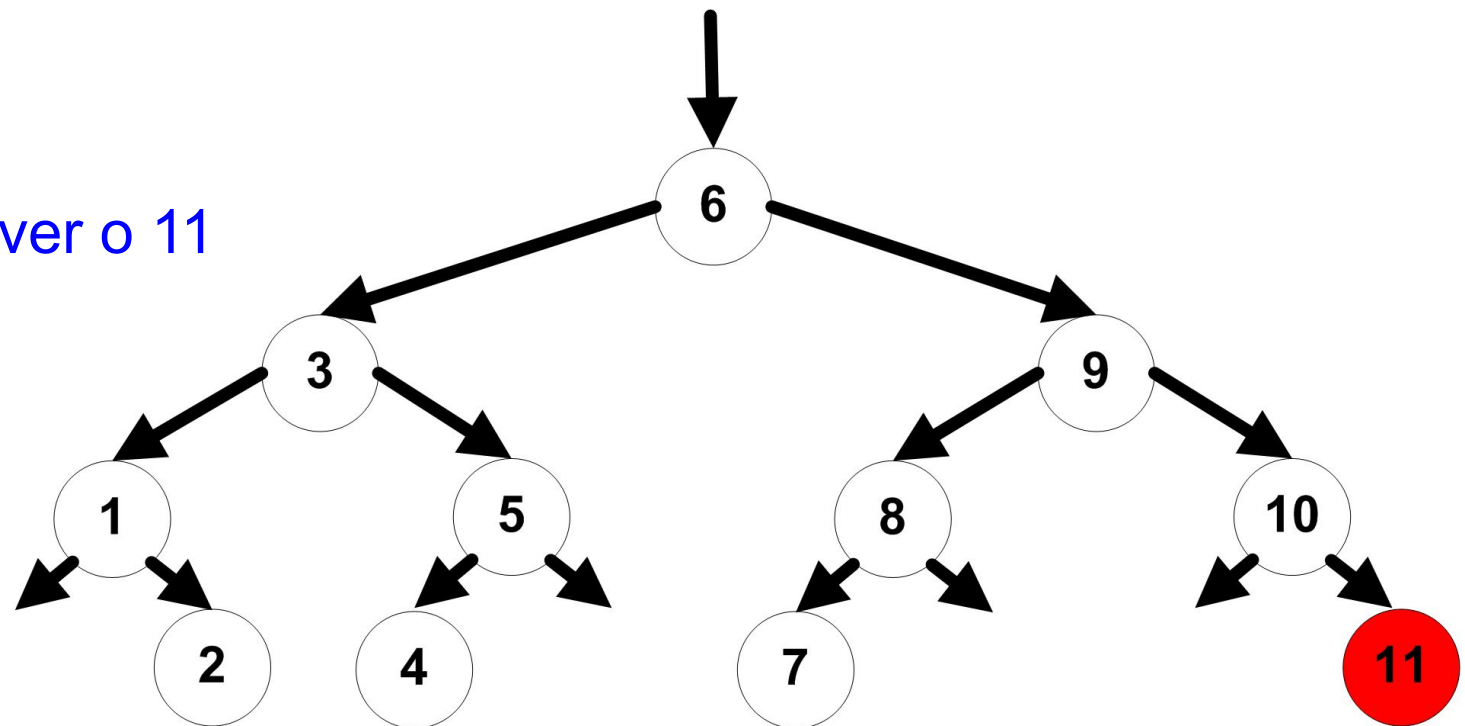
Exemplo: Remover o 11



# Funcionamento Básico da Remoção

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

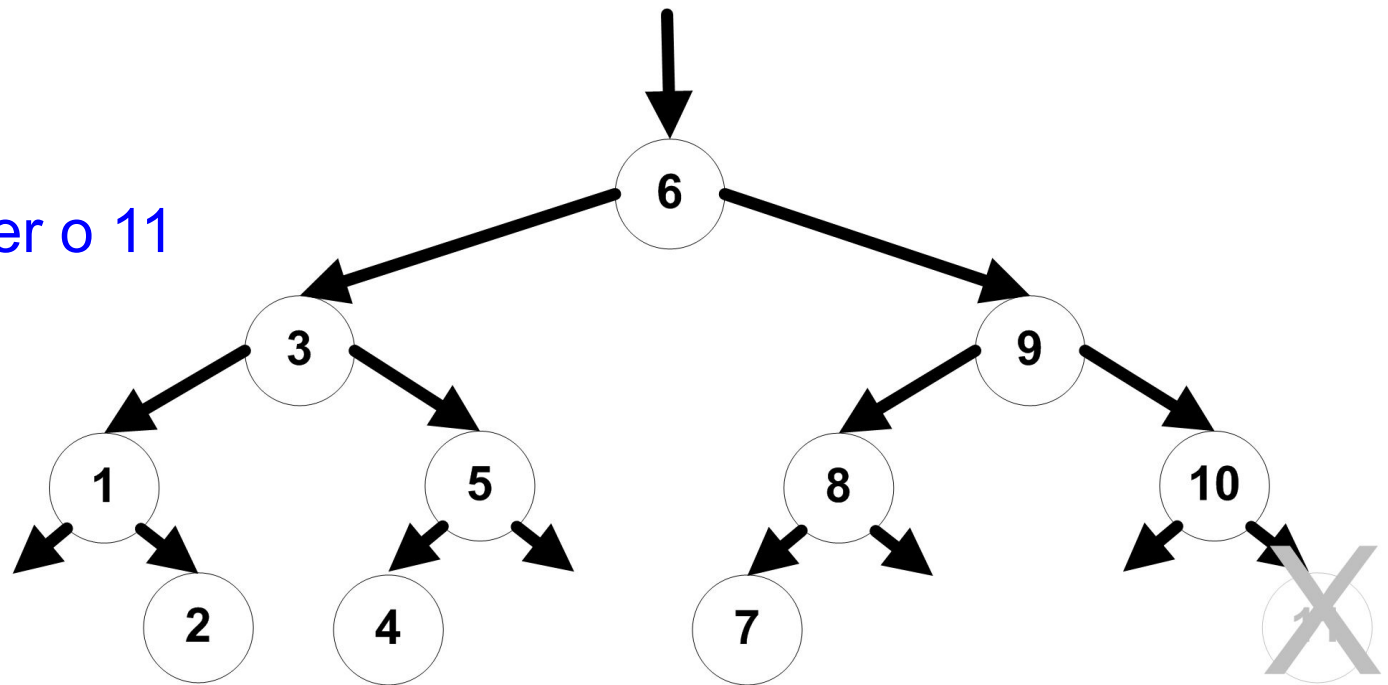
Exemplo: Remover o 11



# Funcionamento Básico da Remoção

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

Exemplo: Remover o 11



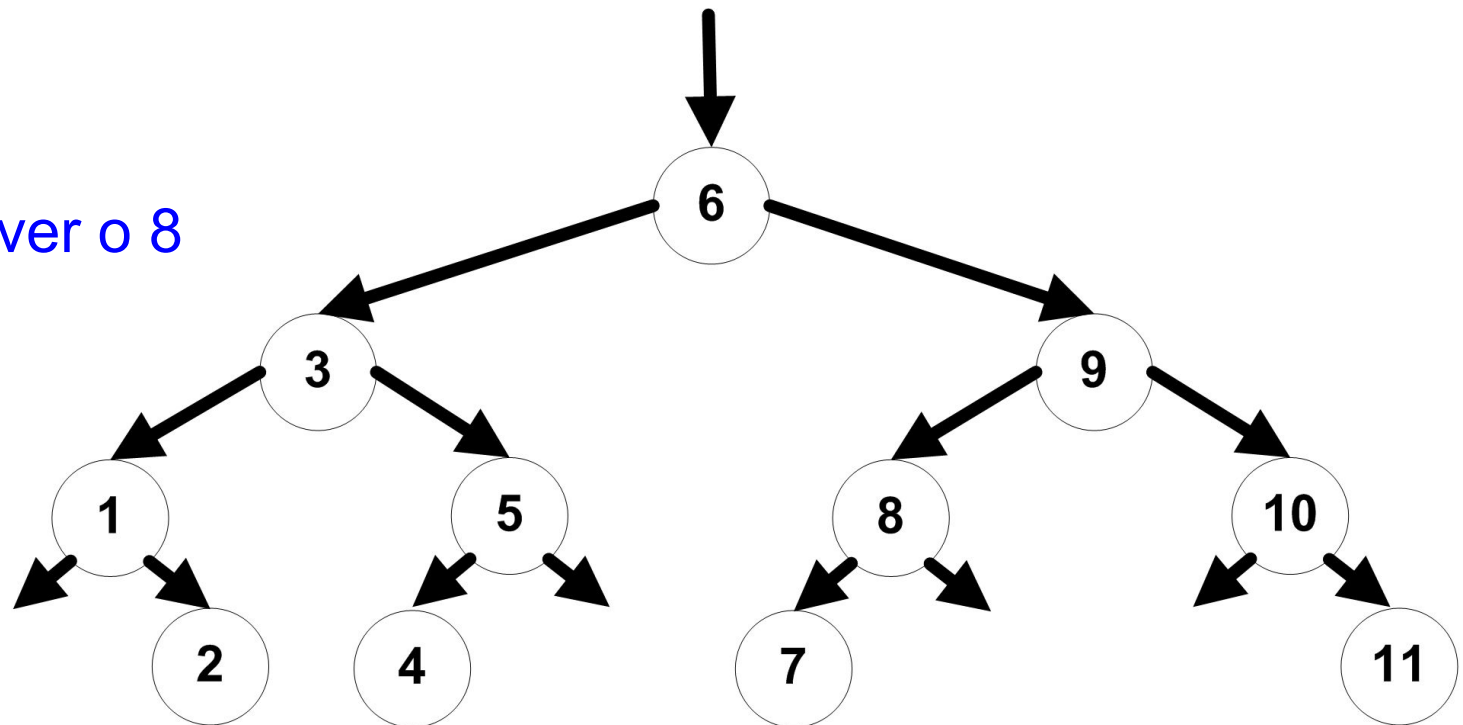
# Funcionamento Básico da Remoção

(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

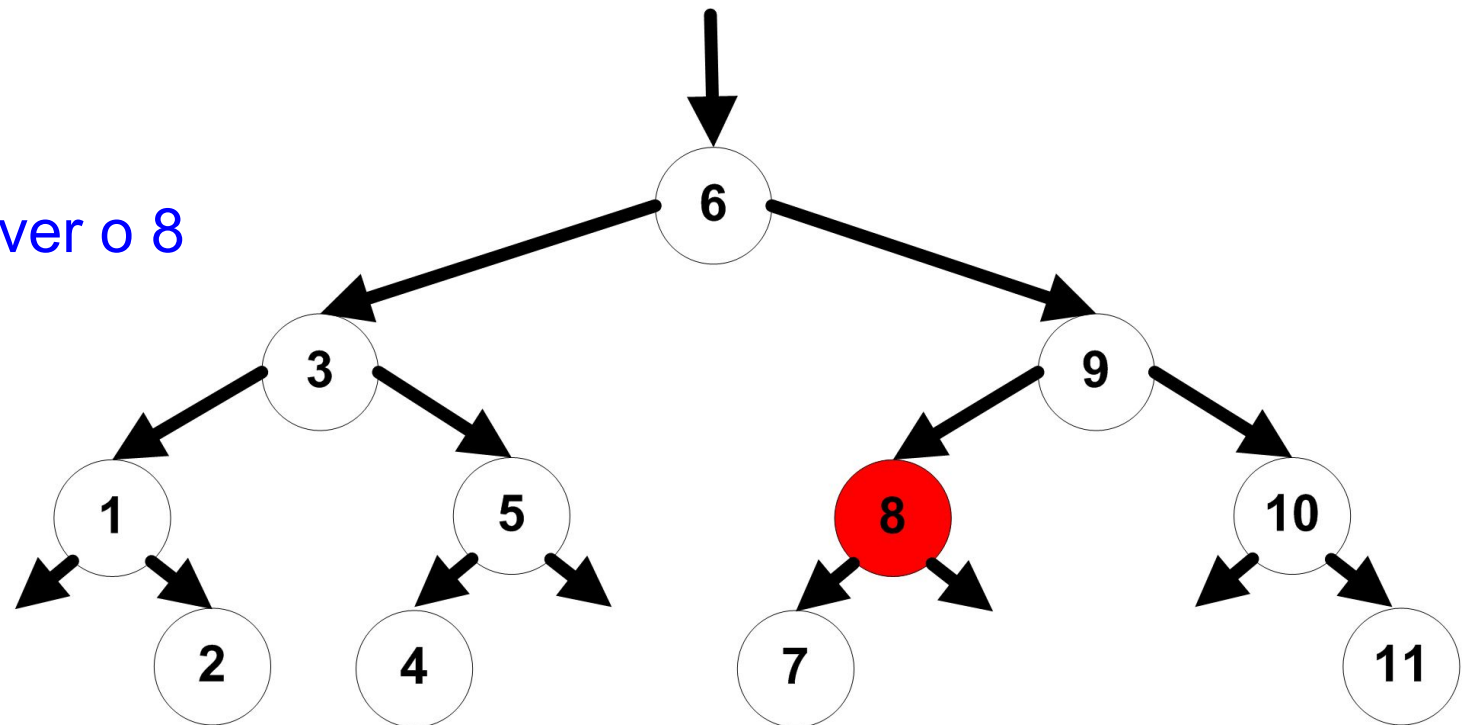
Exemplo: Remover o 8



# 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

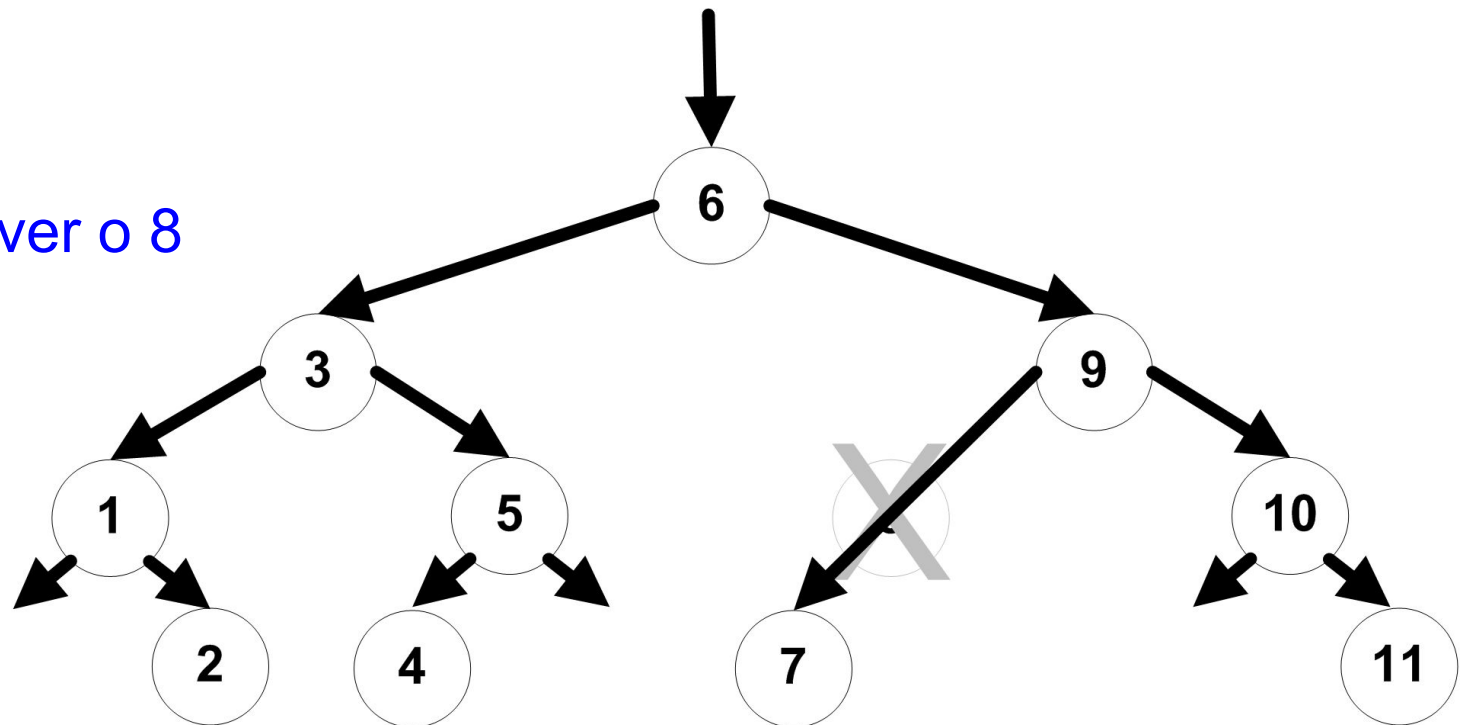
Exemplo: Remover o 8



# 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

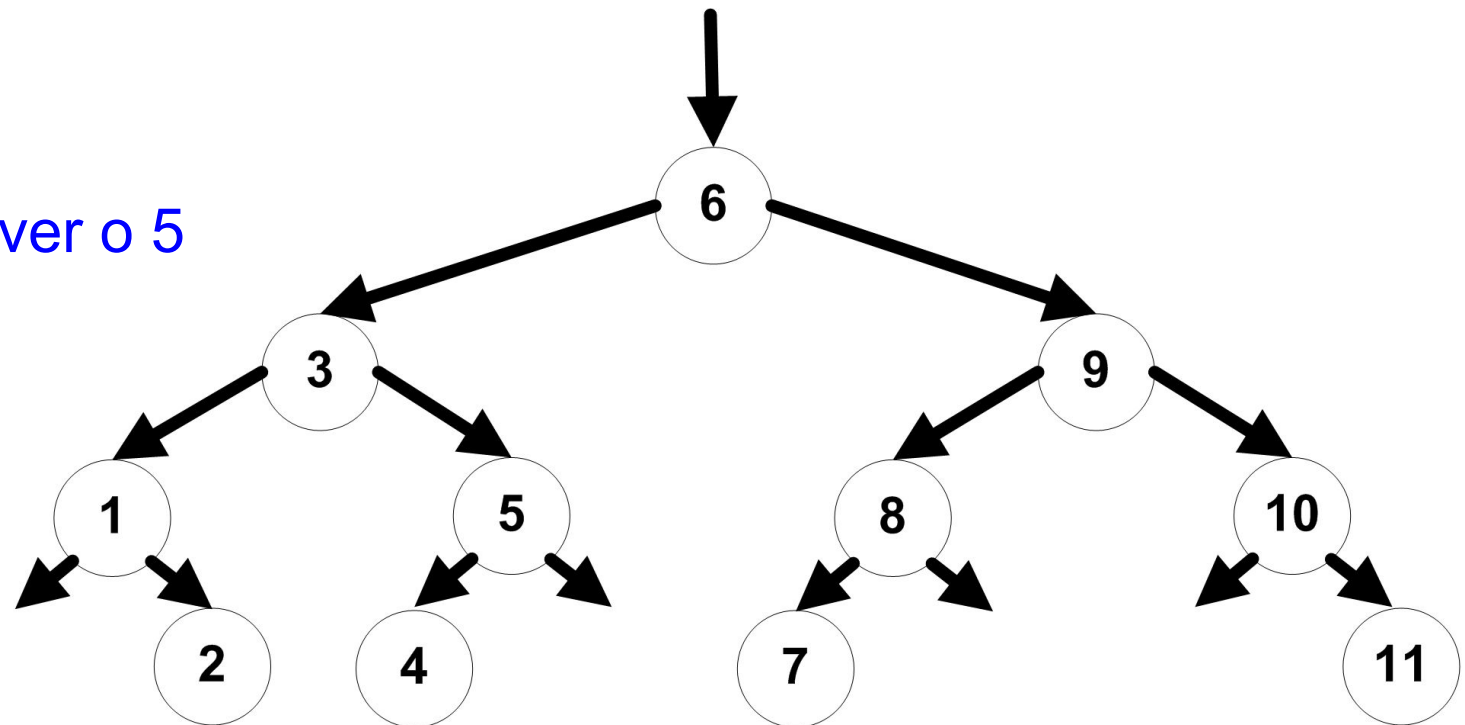
Exemplo: Remover o 8



# 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

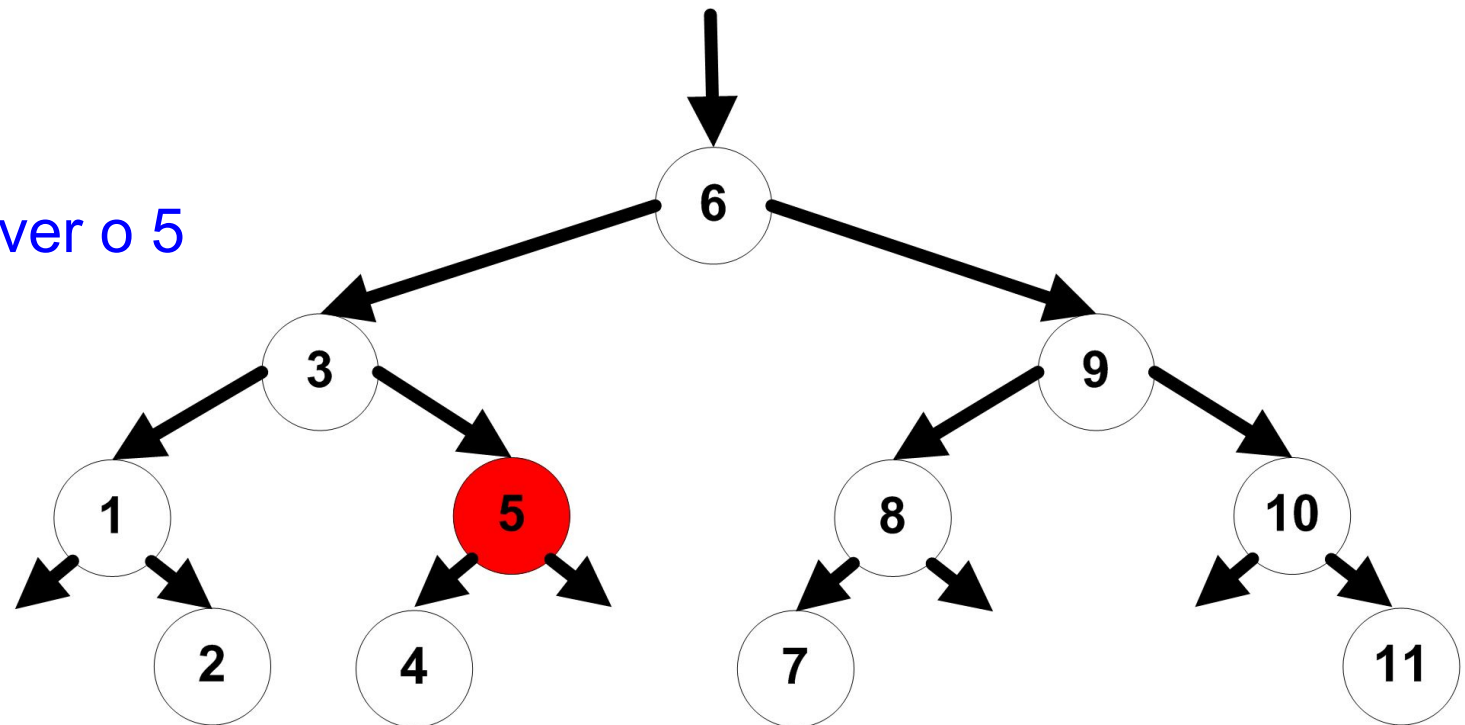




# Funcionamento Básico da Remoção

(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

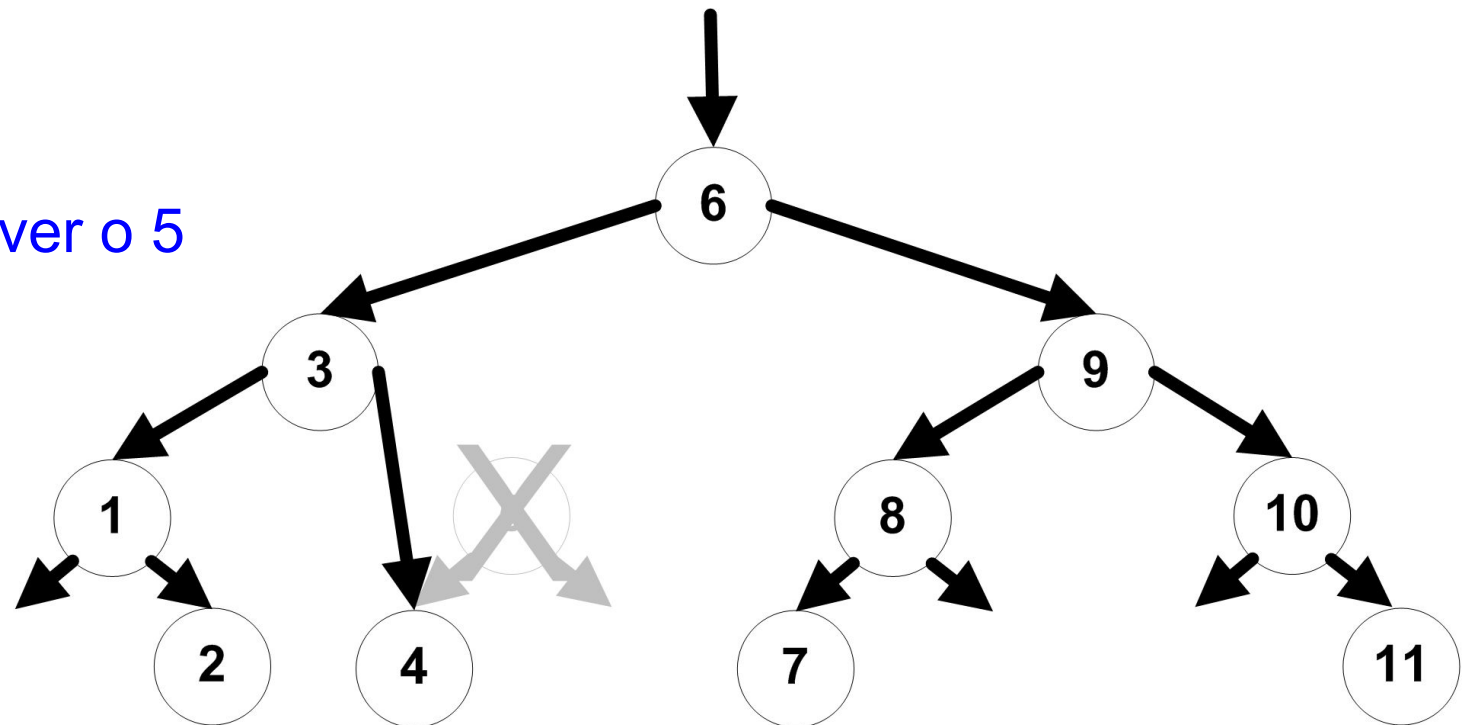
Exemplo: Remover o 5



# 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**

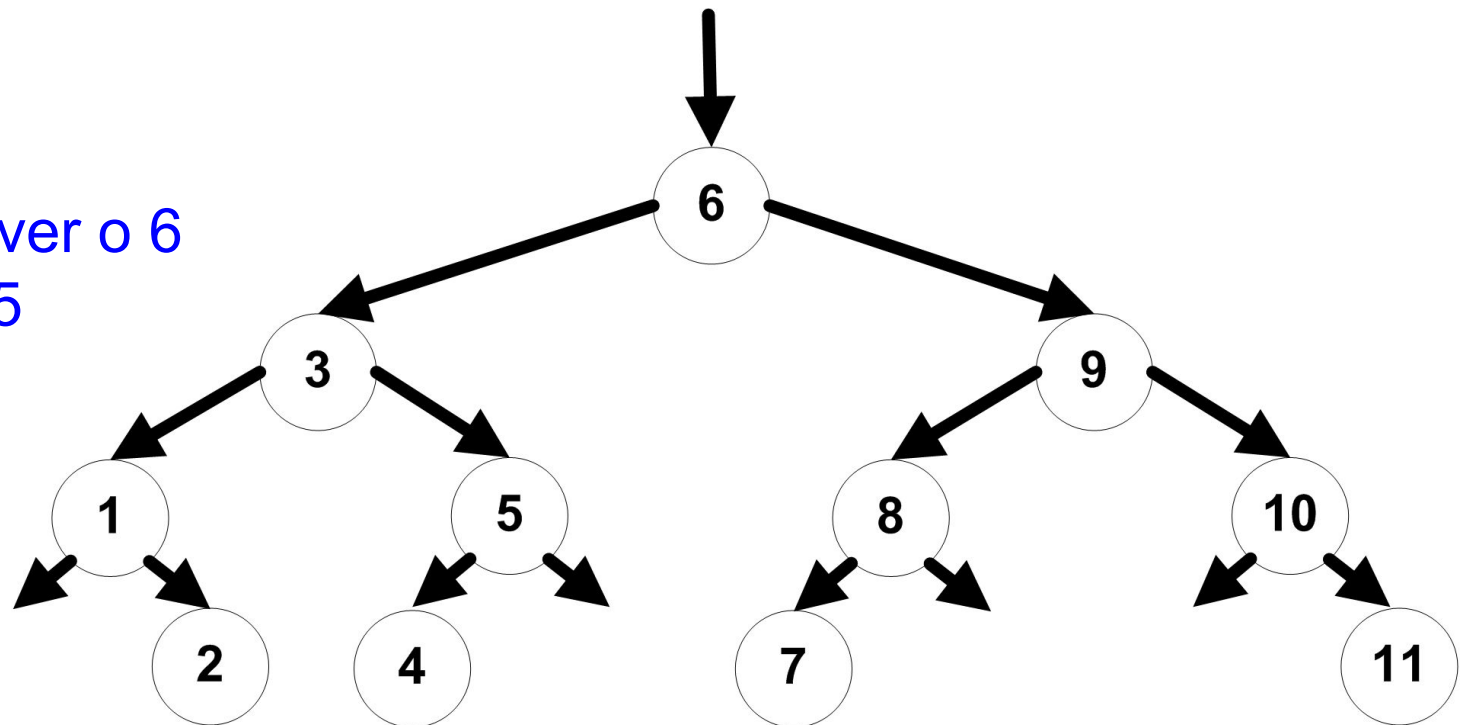
# Funcionamento Básico da Remoção

(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**

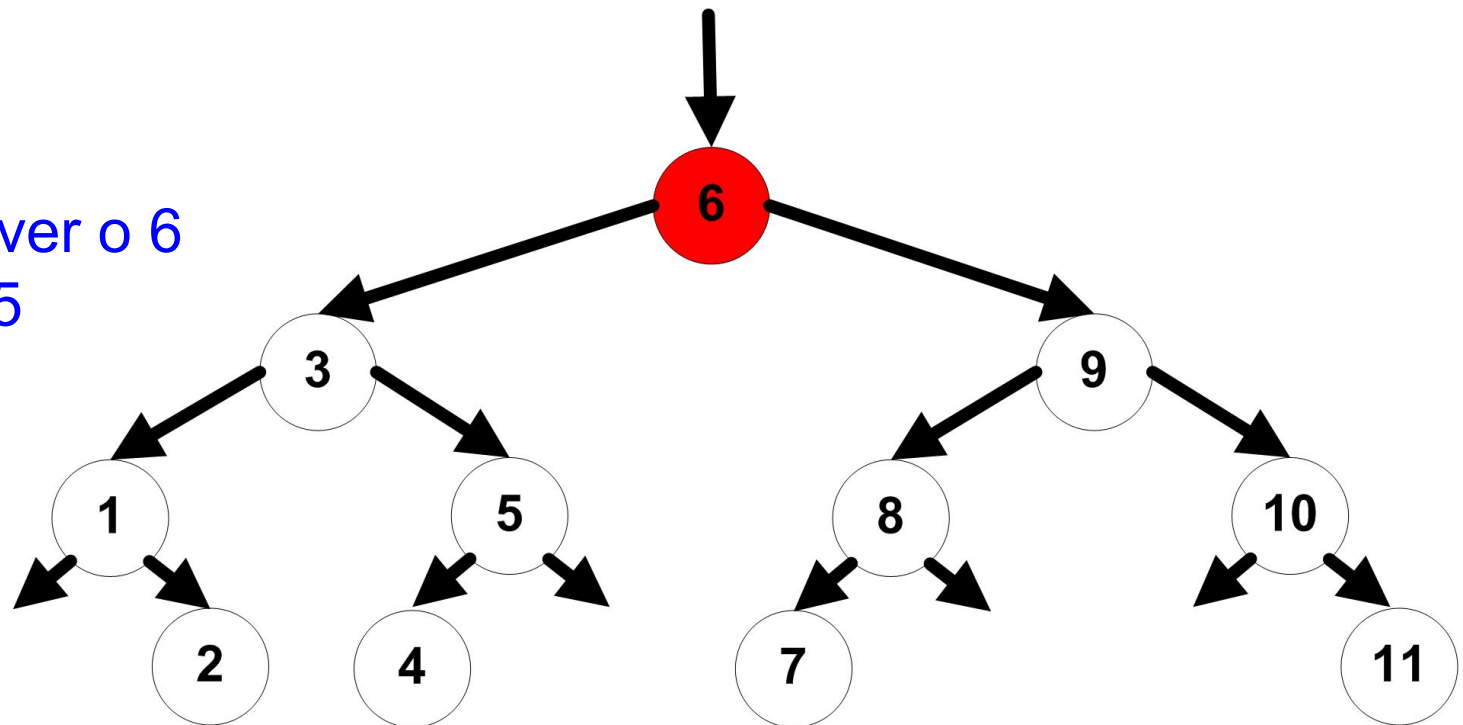
Exemplo: Remover o 6 e substituir pelo 5



# 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**

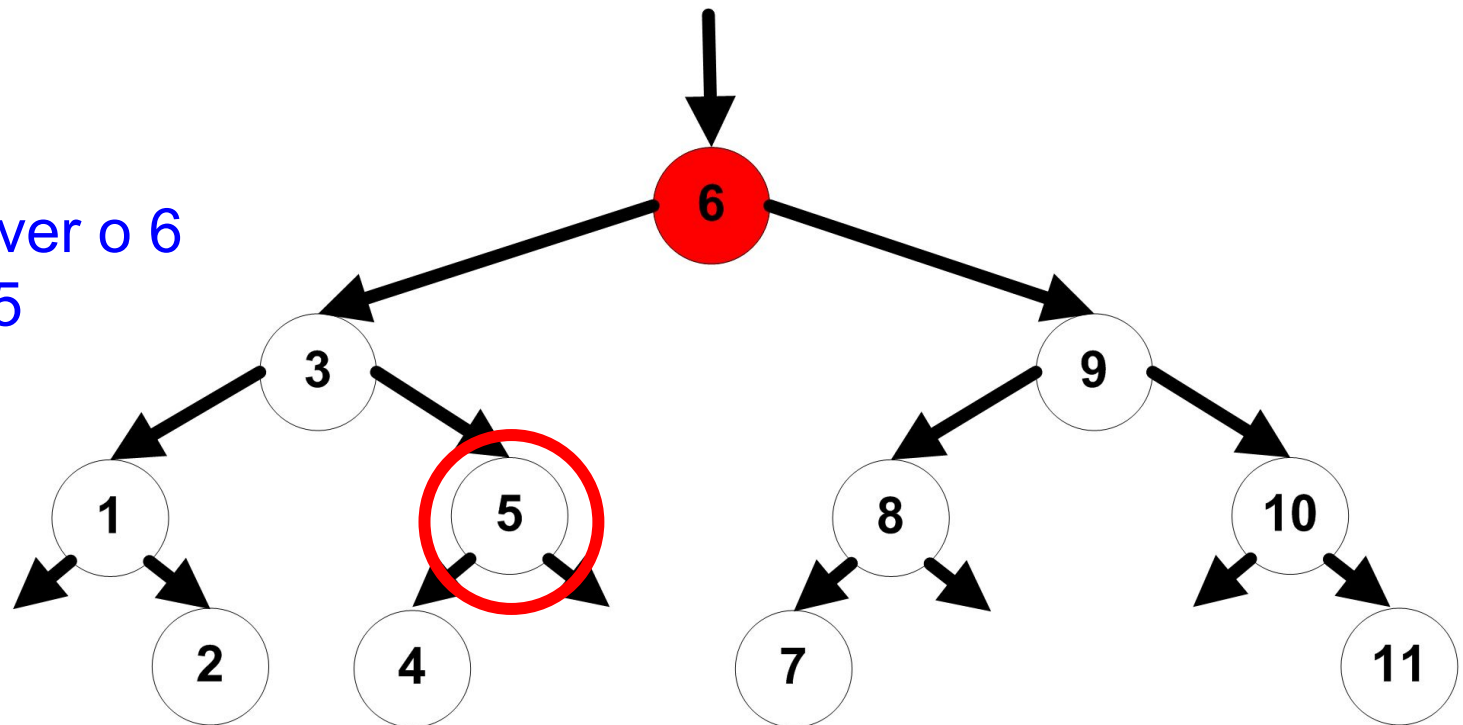
Exemplo: Remover o 6 e substituir pelo 5



# 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**

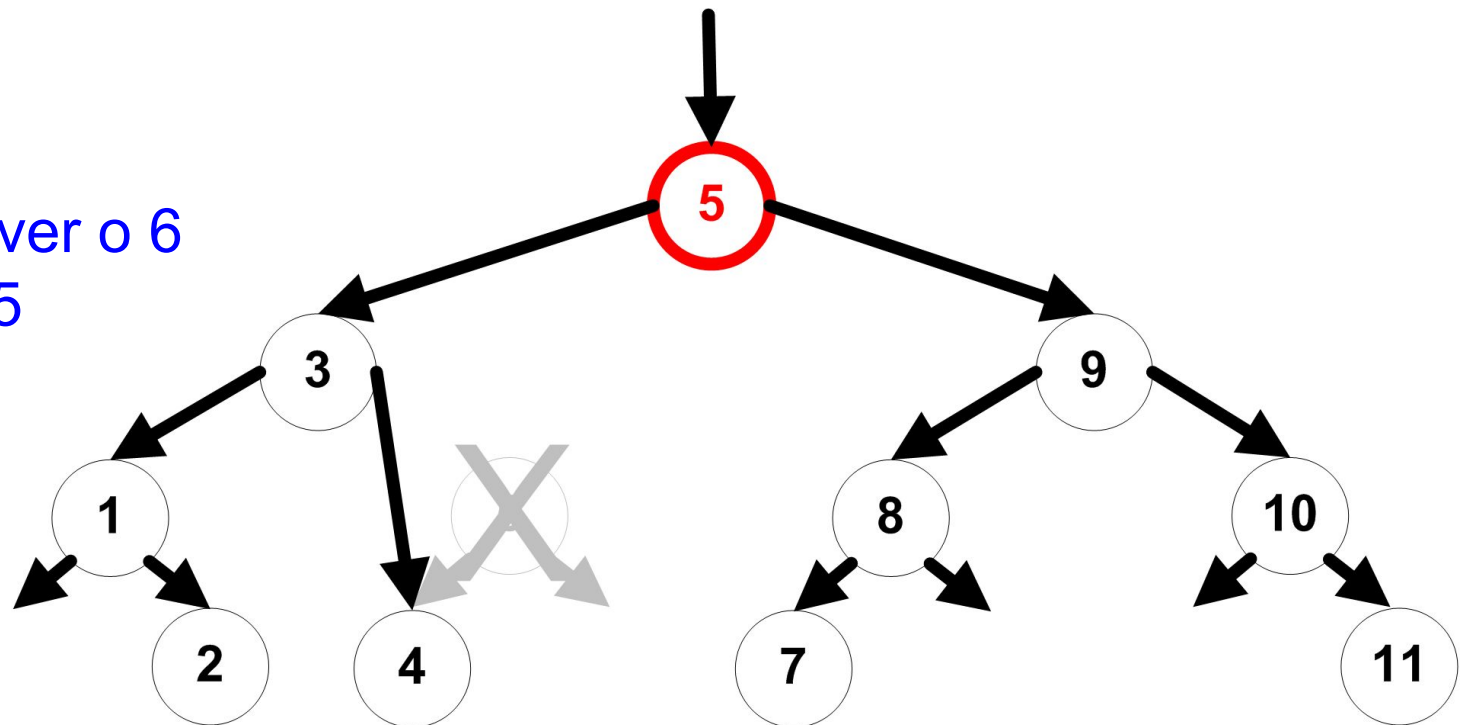
Exemplo: Remover o 6 e substituir pelo 5



# 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**

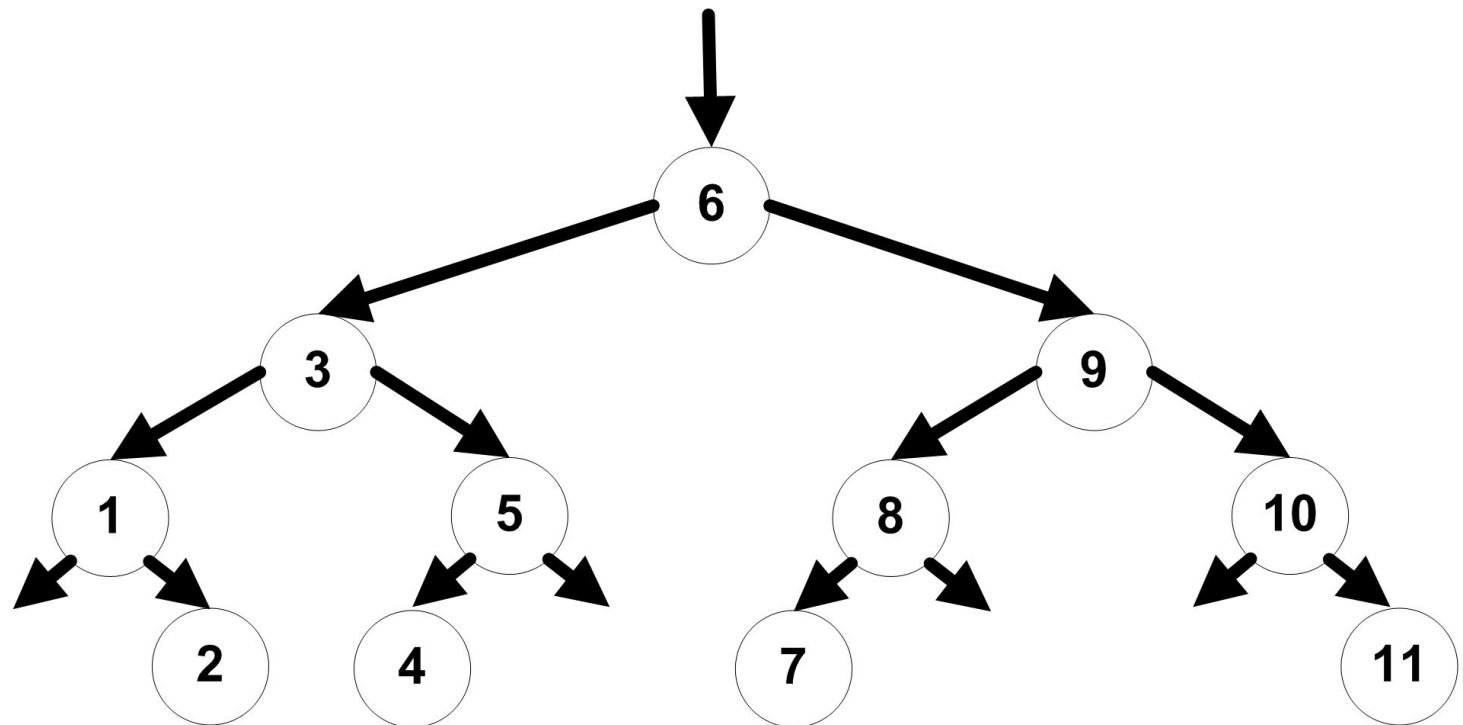
Exemplo: Remover o 6 e substituir pelo 5





# Funcionamento Básico da Remoção

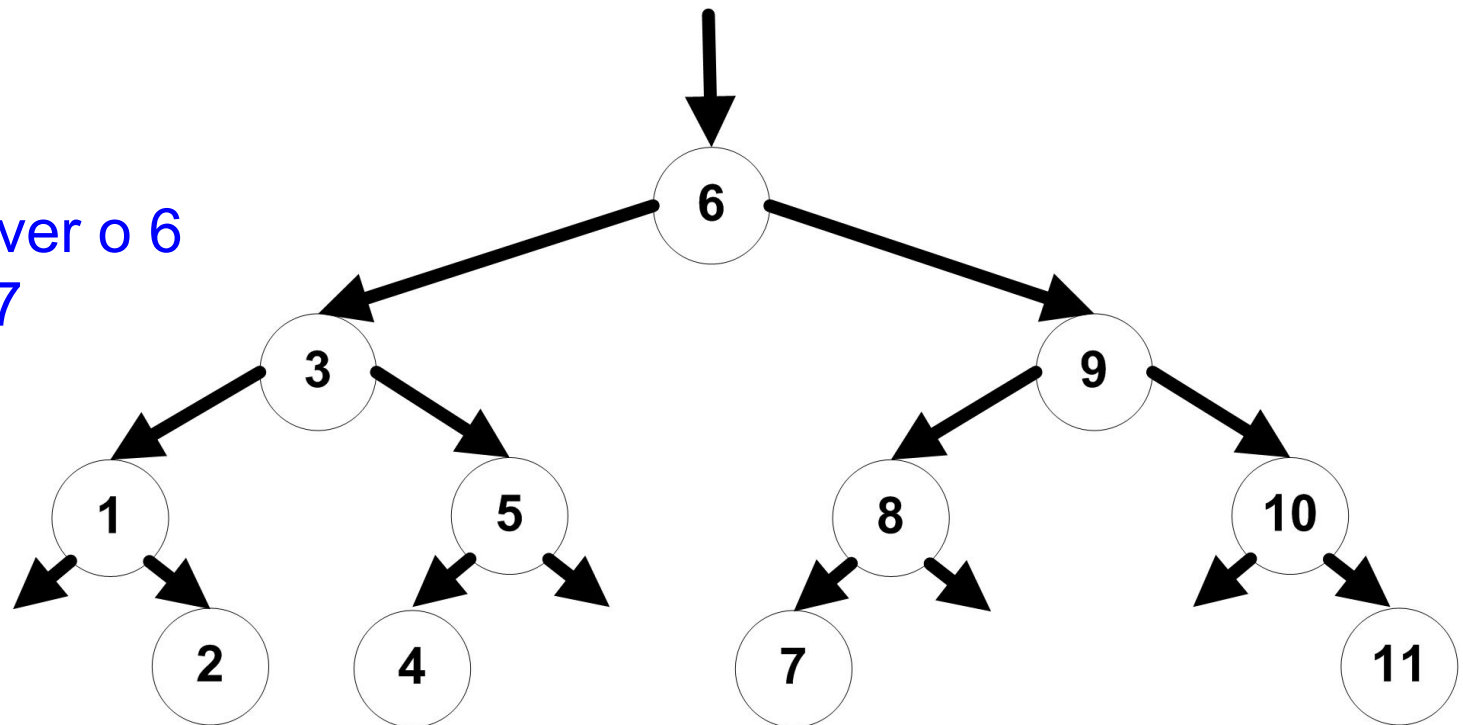
(3) Senão, se o elemento estiver em um **nó intermediário com dois filhos**, o elemento a ser removido deve ser substituído ou pelo **maior nó da subárvore à esquerda** ou **menor nó da subárvore à direita**



# 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**

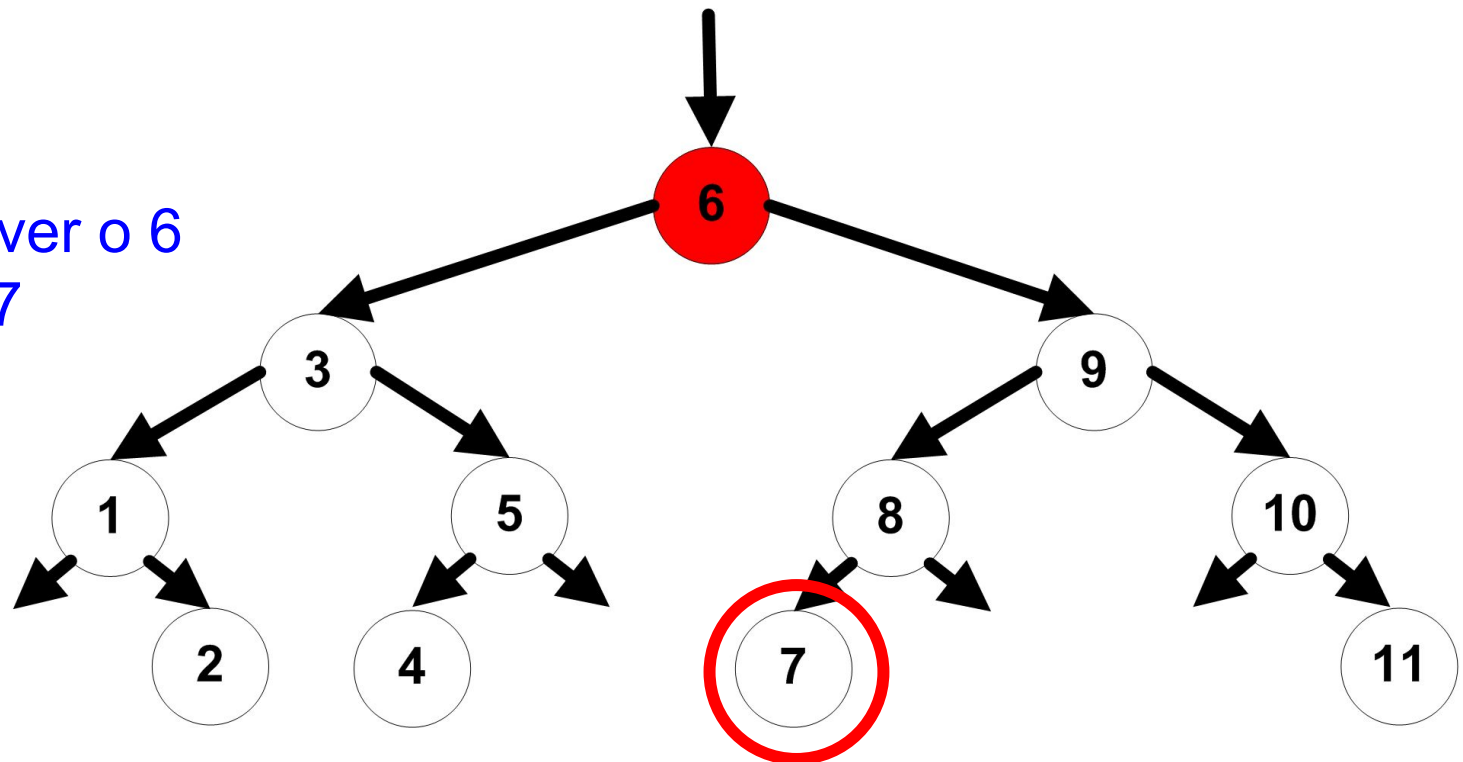
Exemplo: Remover o 6 e substituir pelo 7



# 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**

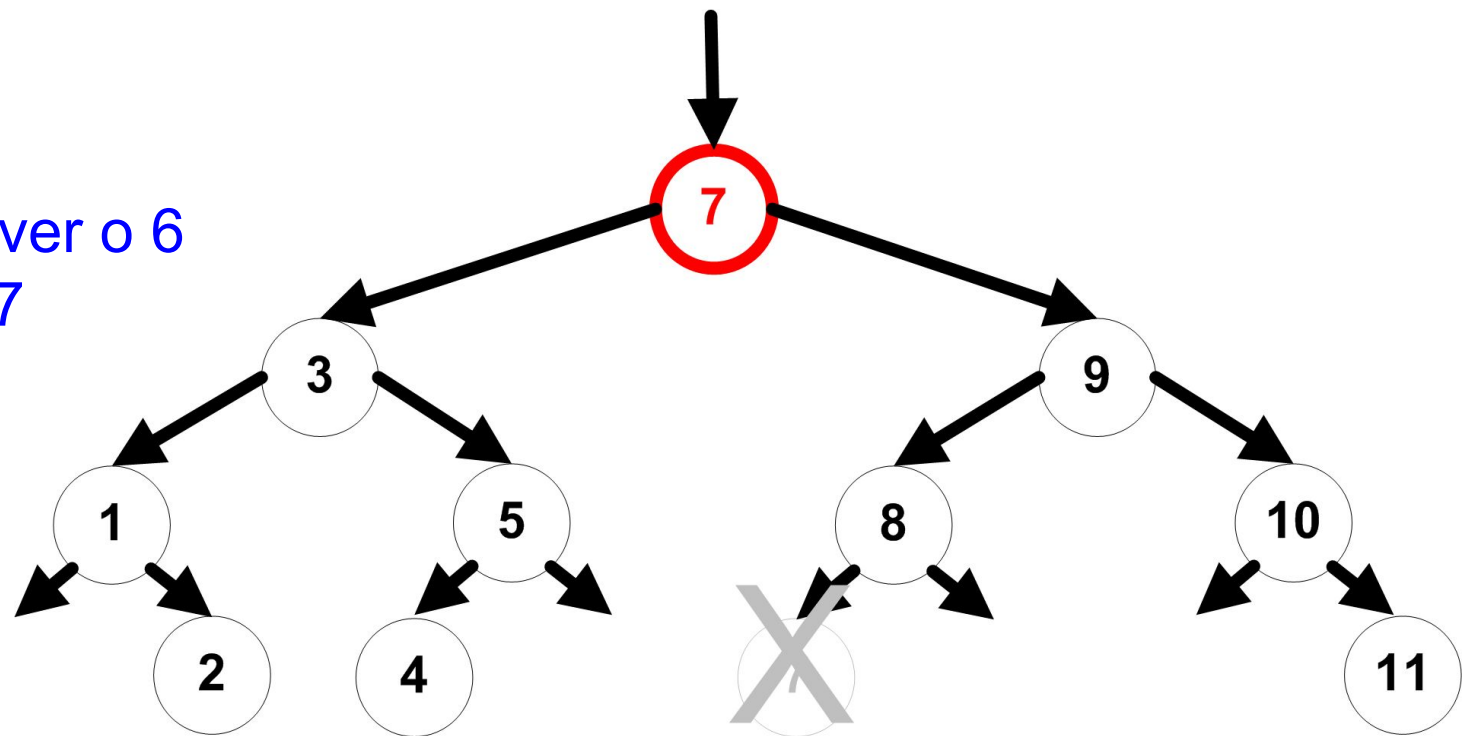
Exemplo: Remover o 6 e substituir pelo 7




# 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



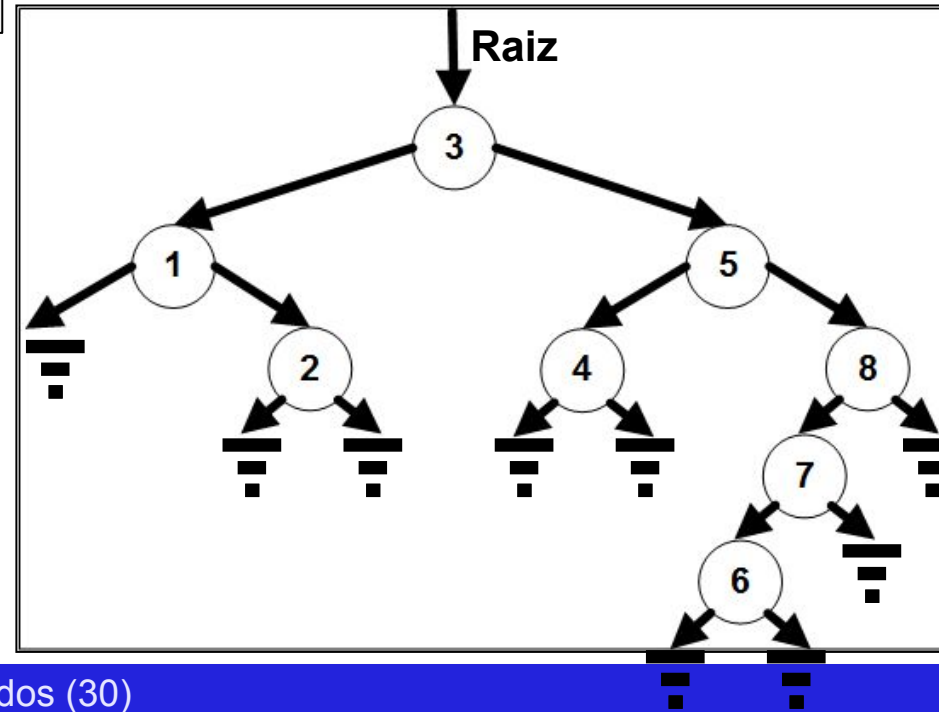
- Funcionamento básico
- **Algoritmo em C#** 
- Análise de complexidade

# Classe Árvore Binária: Remoção em C#

```
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() { }
}
```

raiz

n(3)



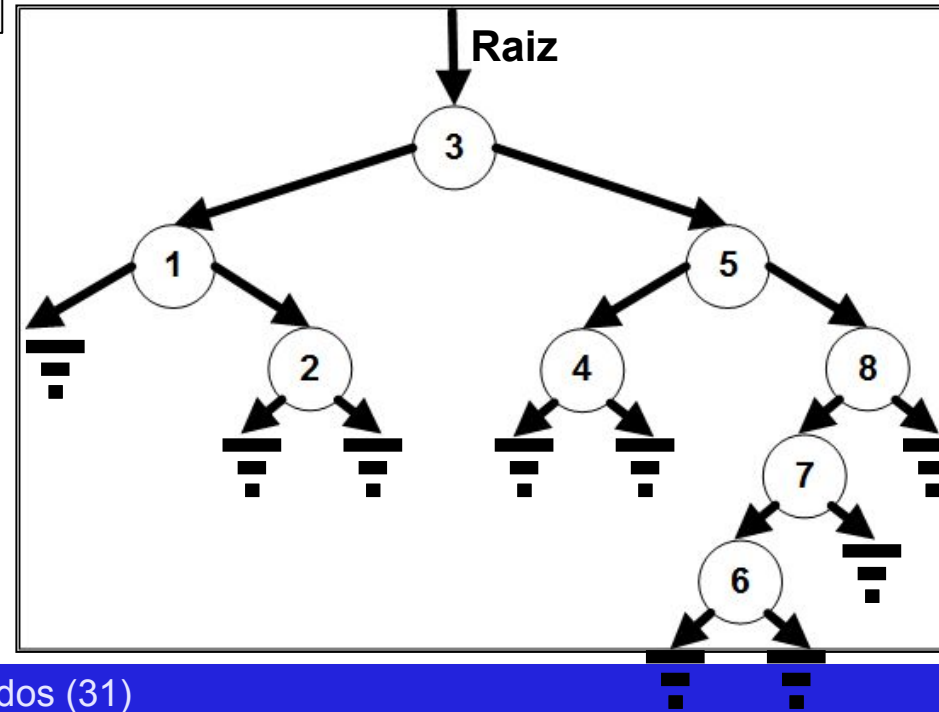
# Algoritmo de Remoção em C#

```
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() { }
}
```

raiz

n(3)

Vamos remover o 2 (uma folha) de nossa árvore



# Algoritmo de Remoção em C#

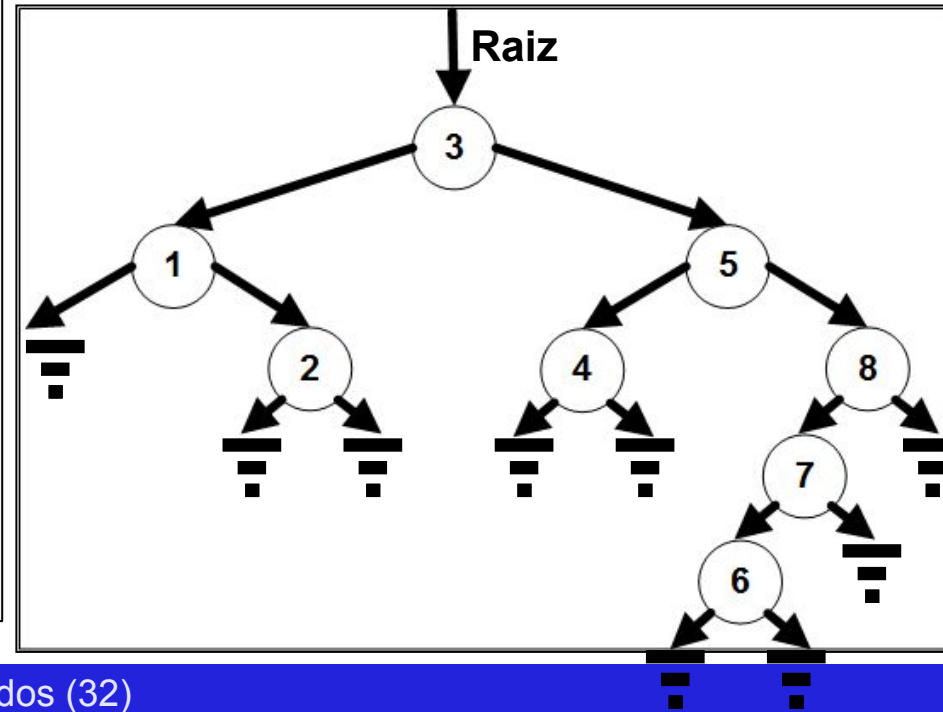
//remover(2), folha

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

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

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

raiz n(3) x 2





# Algoritmo de Remoção em C#

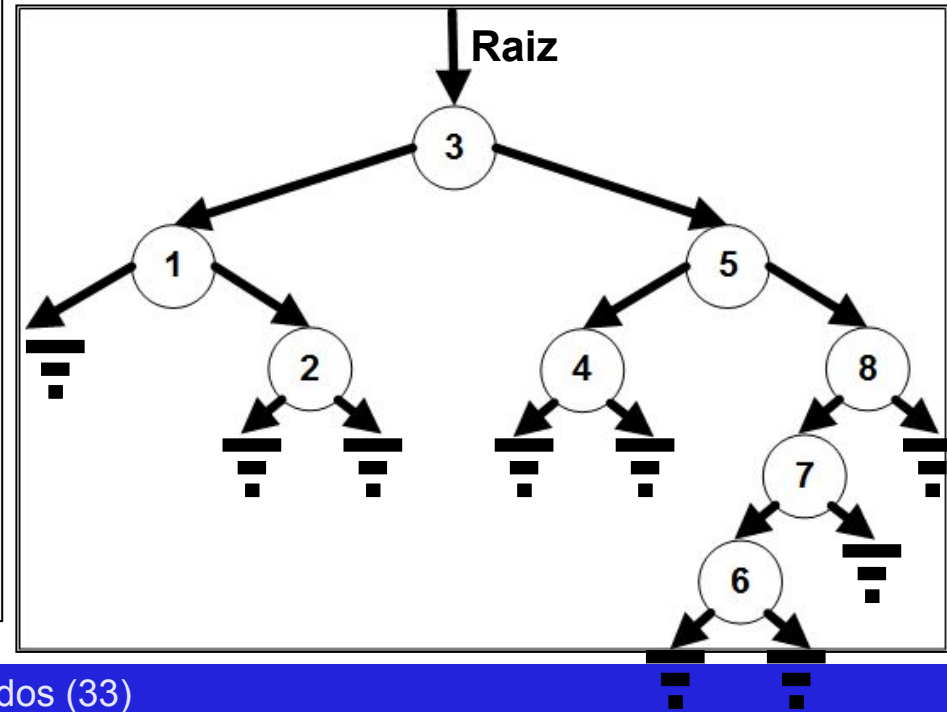
//remover(2), folha

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

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

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

raiz    n(3)    x    2



# Algoritmo de Remoção em C#

//remover(2), folha

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

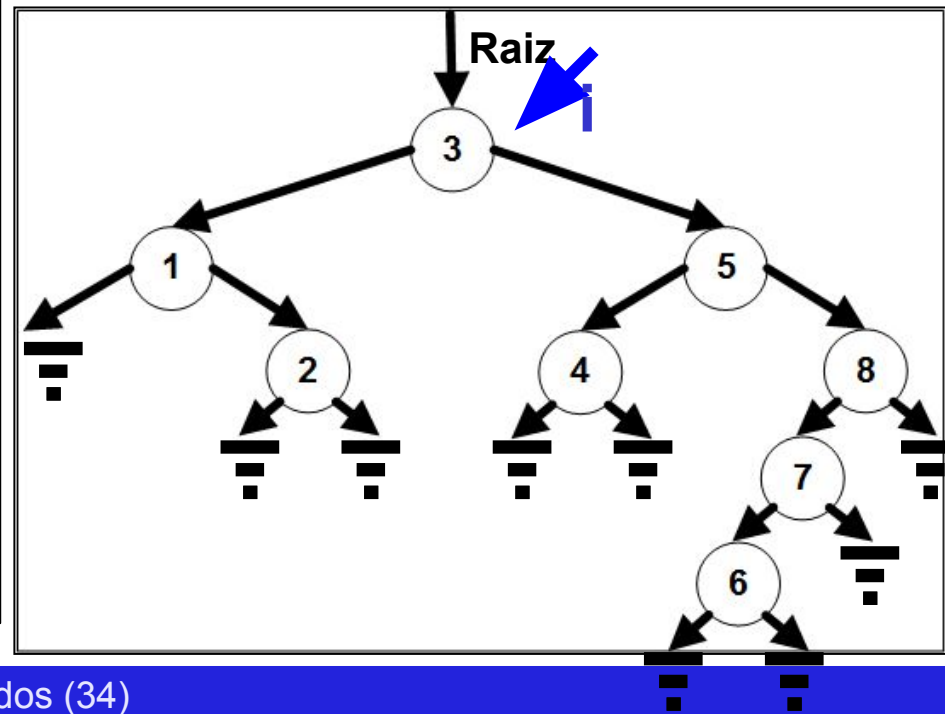
No remover(int x, No i) throws Exception {

```
    if (i == null) { throw new Exception("Erro!");
    } else if (x < i.elemento) { i.esq = remover(x, i.esq);
    } else if (x > i.elemento) { i.dir = remover(x, i.dir);
    } else if (i.dir == null) { i = i.esq;
    } else if (i.esq == null) { i = i.dir;
    } else {
        i.esq = maiorEsq(i, i.esq);
    }
    return i;
}
```

No maiorEsq(No i, No j) {

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

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



# Algoritmo de Remoção em C#

//remover(2), folha

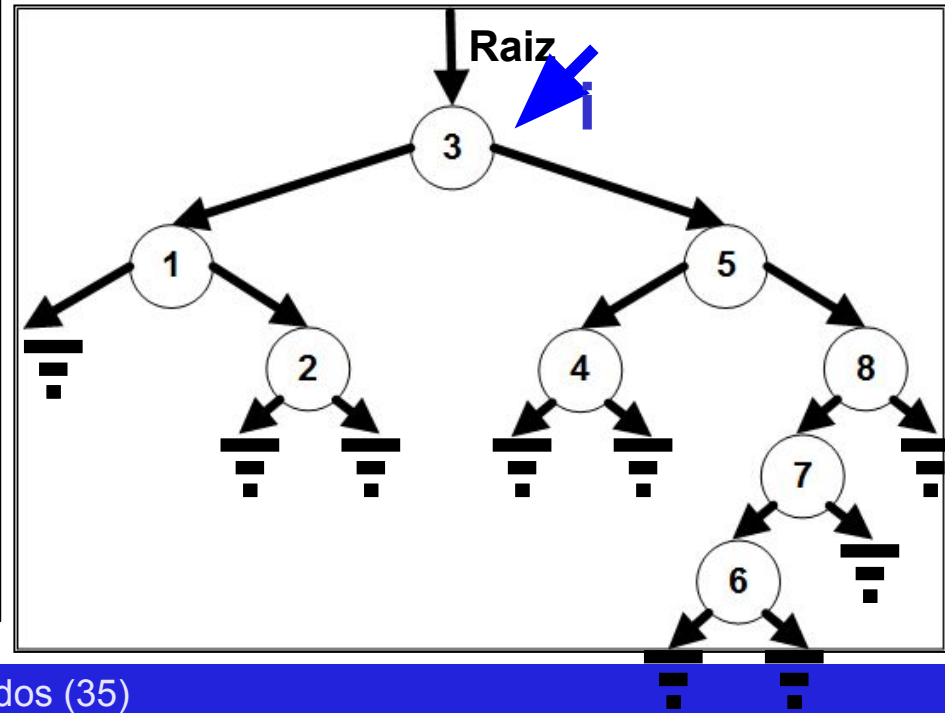
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

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

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

false: n(3) == null

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



# Algoritmo de Remoção em C#

//remover(2), folha

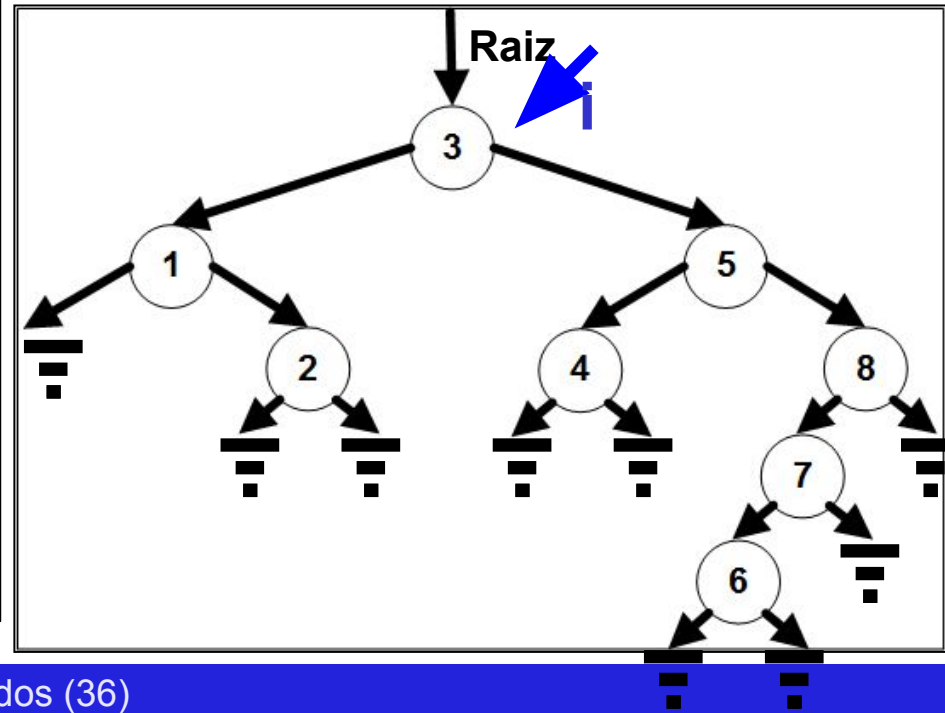
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

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

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

true: 2 < 3

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



# Algoritmo de Remoção em C#

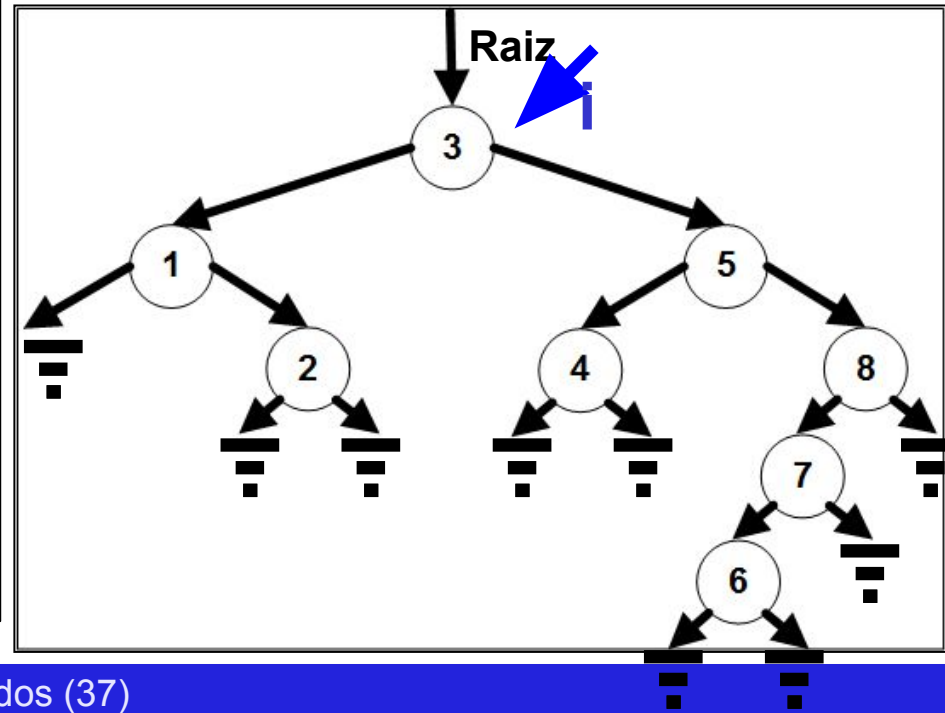
//remover(2), folha

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

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

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

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



# Algoritmo de Remoção em C#

//remover(2), folha

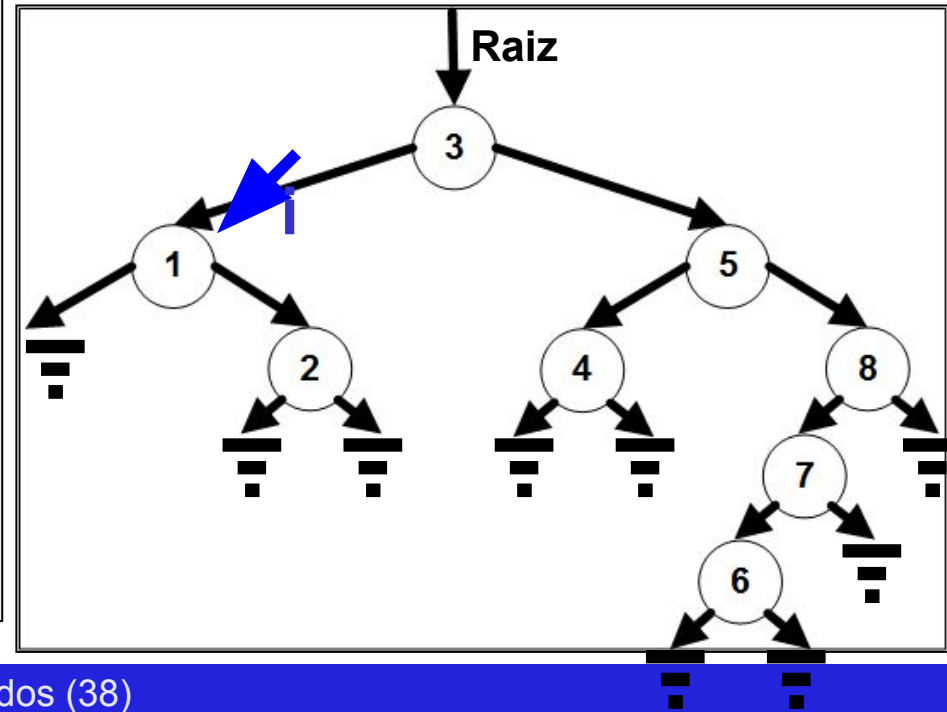
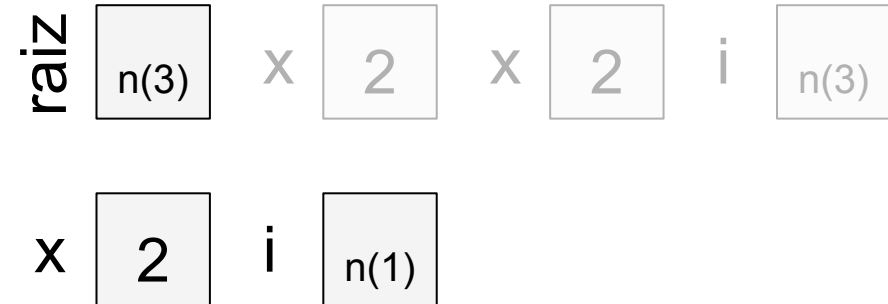
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

```
    if (i == null) { throw new Exception("Erro!");
    } else if (x < i.elemento) { i.esq = remover(x, i.esq);
    } else if (x > i.elemento) { i.dir = remover(x, i.dir);
    } else if (i.dir == null) { i = i.esq;
    } else if (i.esq == null) { i = i.dir;
    } else {
        i.esq = maiorEsq(i, i.esq);
    }
    return i;
}
```

No maiorEsq(No i, No j) {

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



# Algoritmo de Remoção em C#

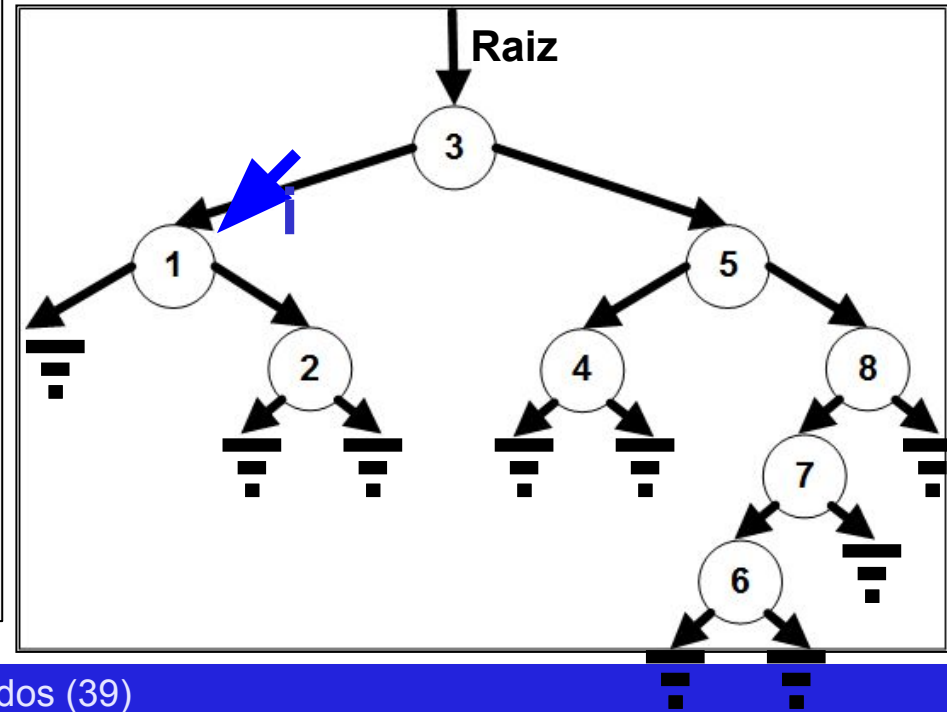
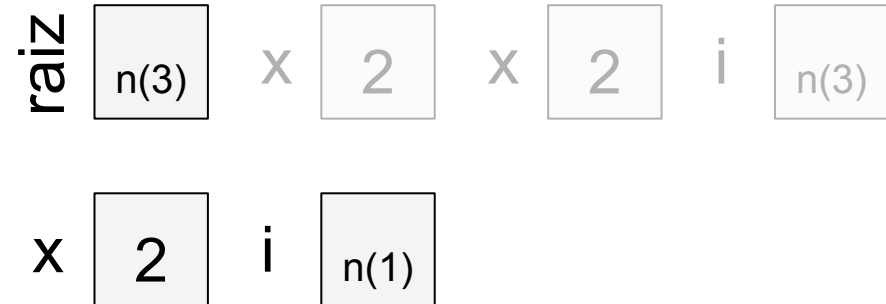
//remover(2), folha

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

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

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

false: n(1) == null



# Algoritmo de Remoção em C#

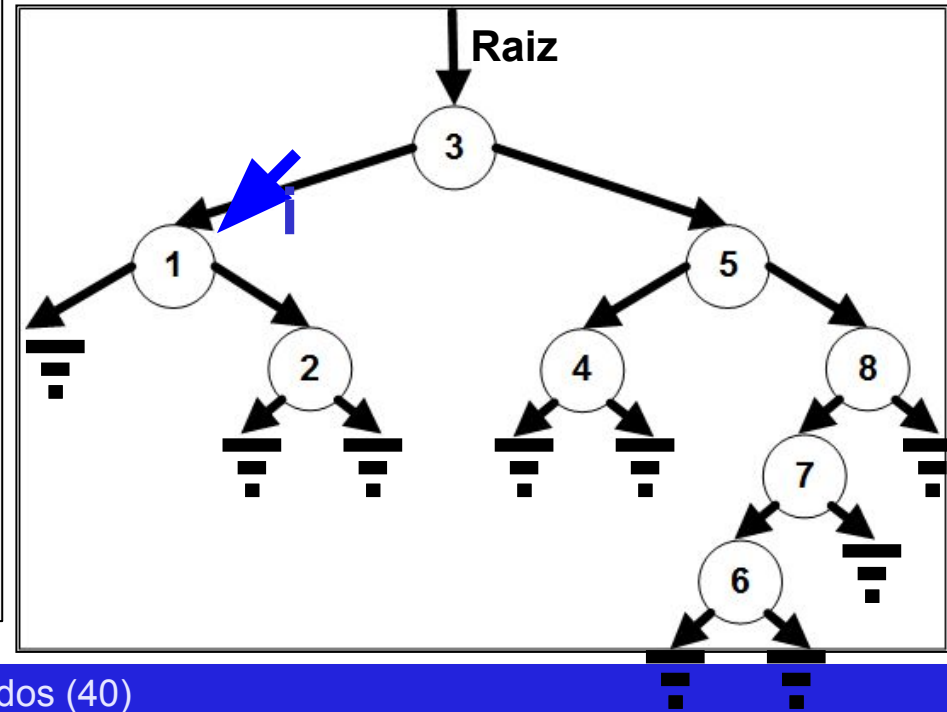
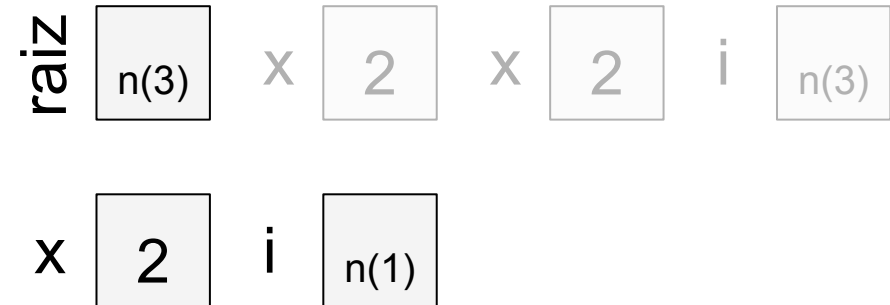
//remover(2), folha

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

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

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

false: 2 < 1





# Algoritmo de Remoção em C#

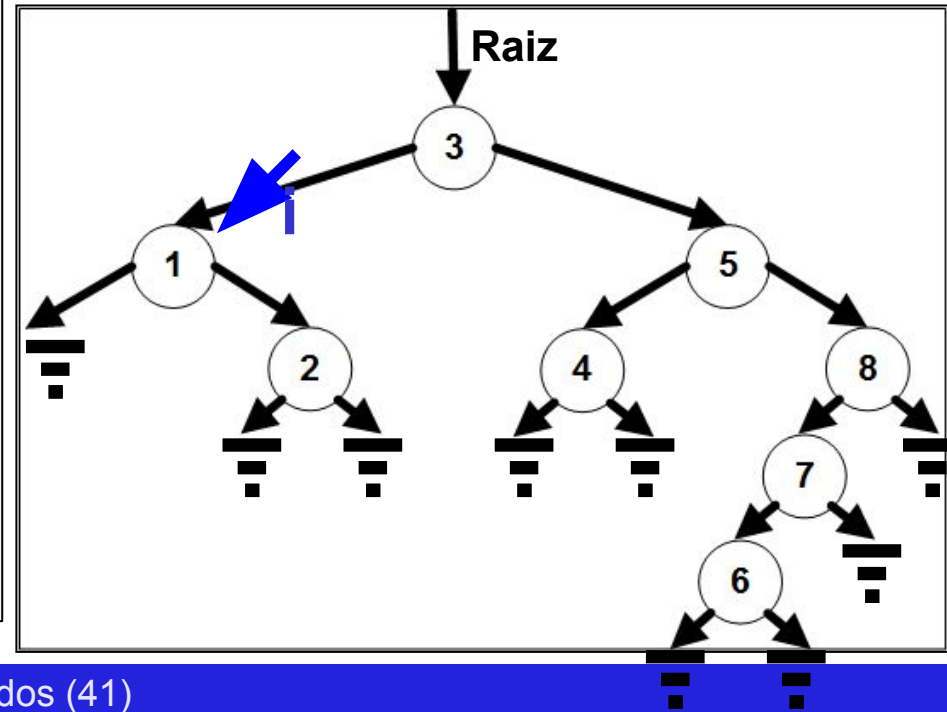
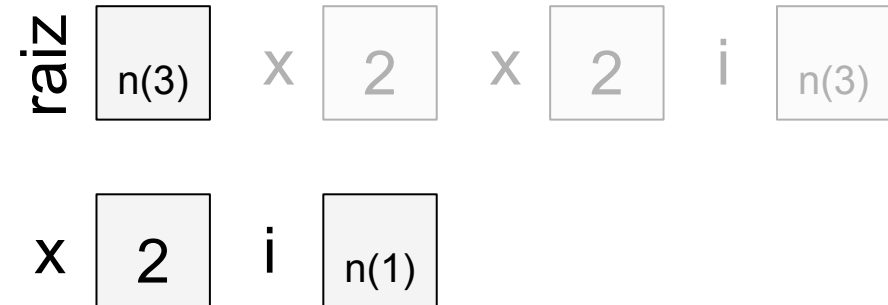
//remover(2), folha

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

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

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

true: 2 > 1



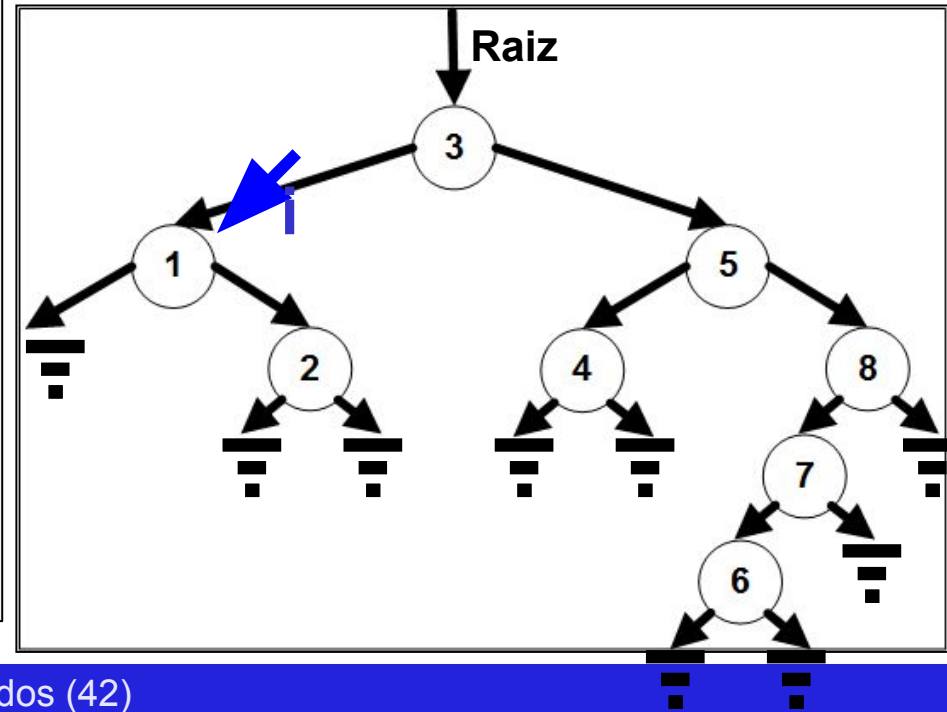
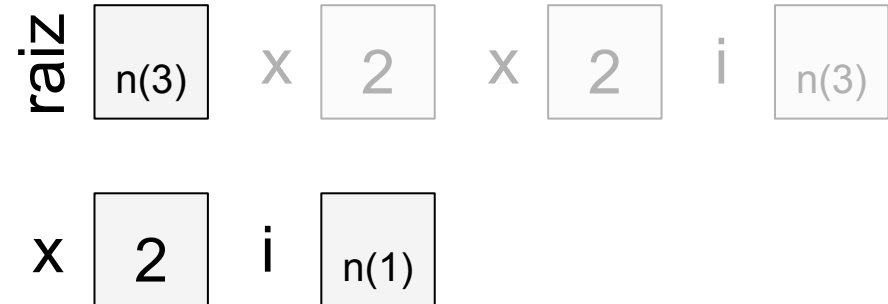
# Algoritmo de Remoção em C#

//remover(2), folha

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

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

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



# Algoritmo de Remoção em C#

//remover(2), folha

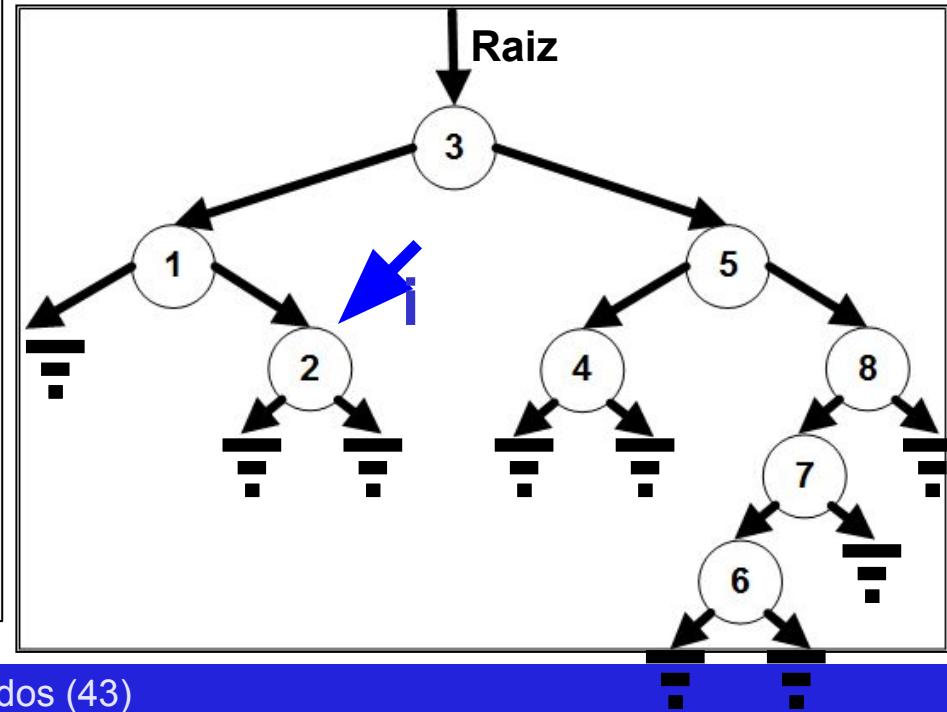
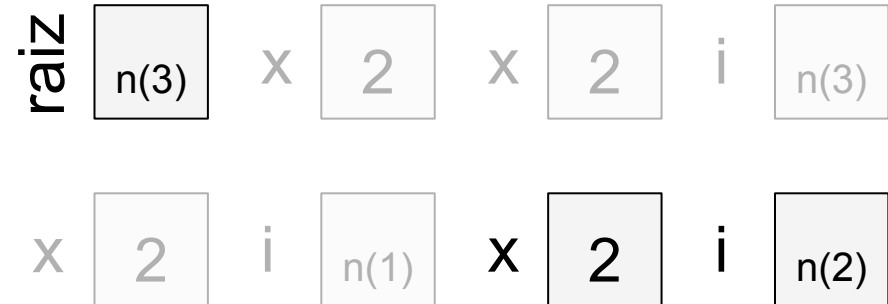
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

```
    if (i == null) { throw new Exception("Erro!");
    } else if (x < i.elemento) { i.esq = remover(x, i.esq);
    } else if (x > i.elemento) { i.dir = remover(x, i.dir);
    } else if (i.dir == null) { i = i.esq;
    } else if (i.esq == null) { i = i.dir;
    } else {
        i.esq = maiorEsq(i, i.esq);
    }
    return i;
}
```

No maiorEsq(No i, No j) {

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



# Algoritmo de Remoção em C#

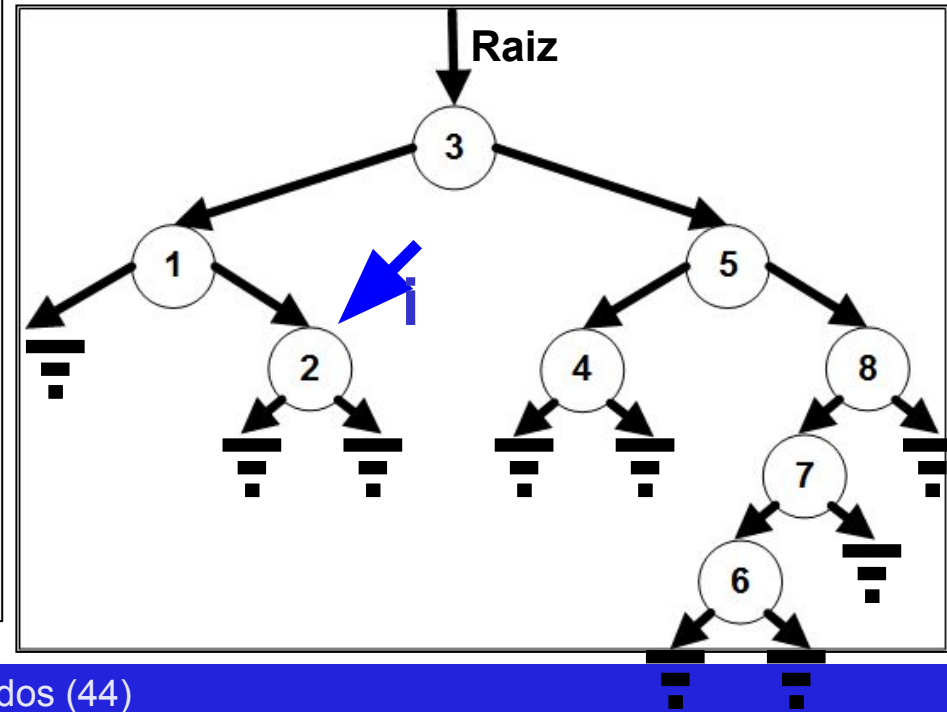
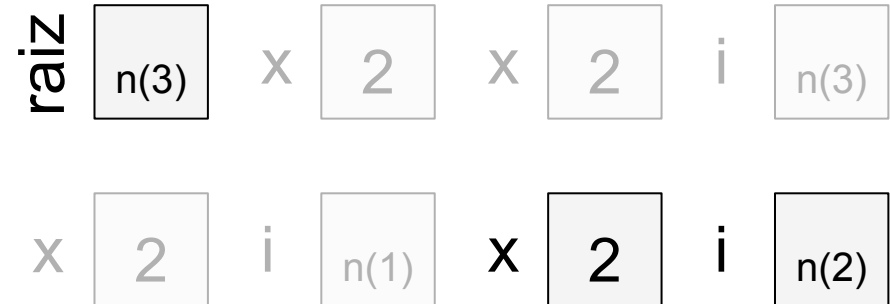
//remover(2), folha

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

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

false: n(2) == null
```

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



# Algoritmo de Remoção em C#

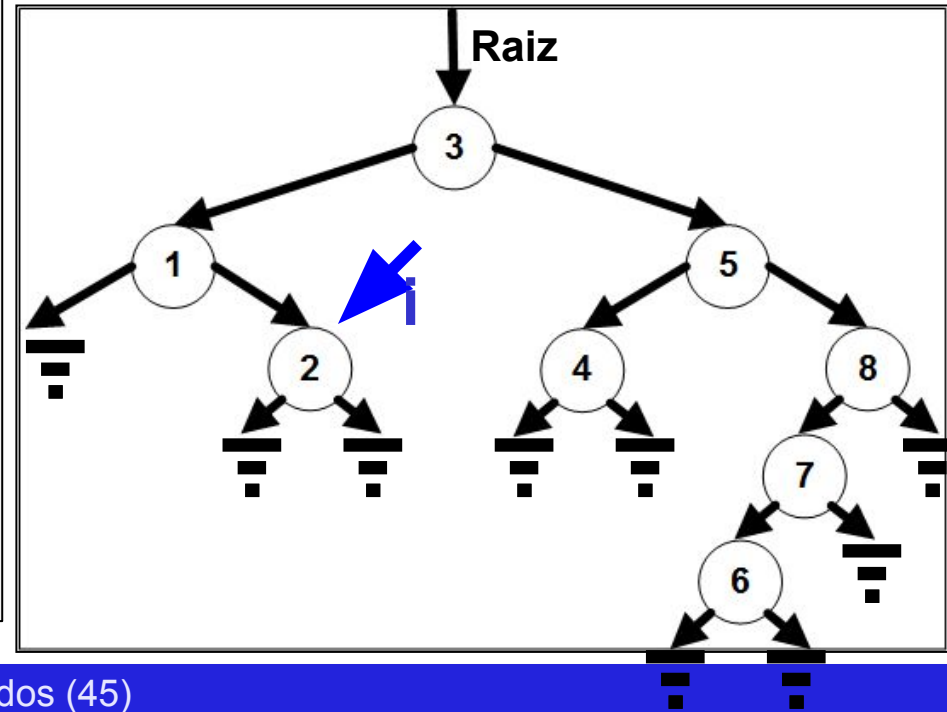
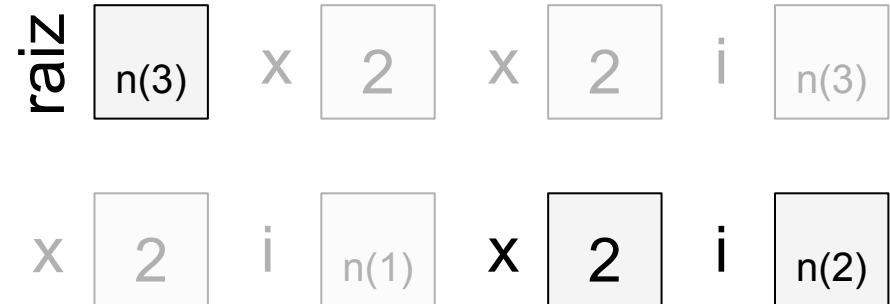
//remover(2), folha

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

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

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

false:  $2 < 2$



# Algoritmo de Remoção em C#

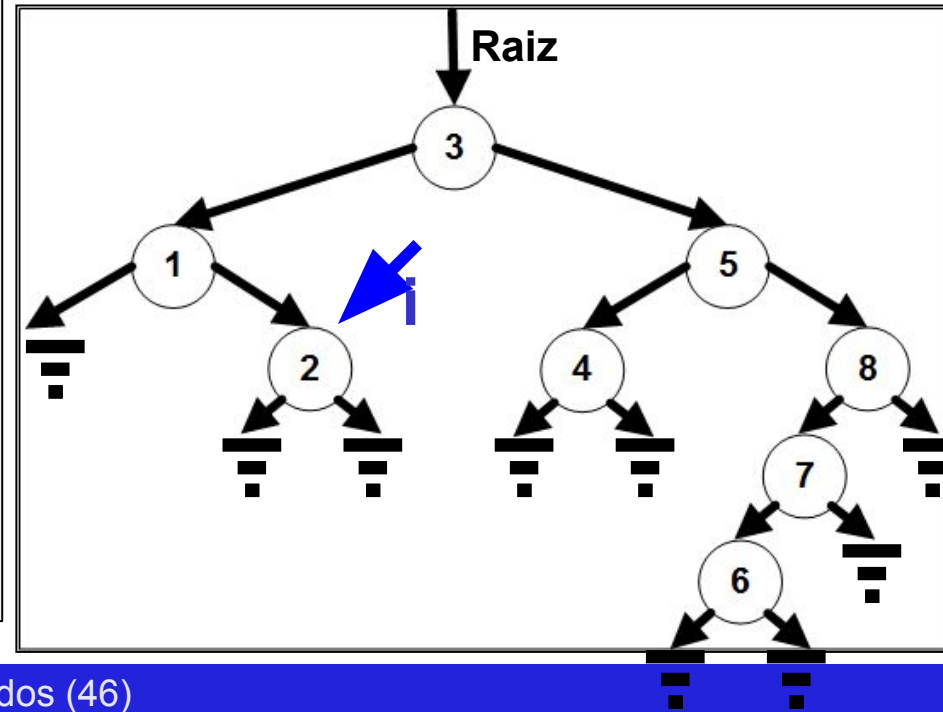
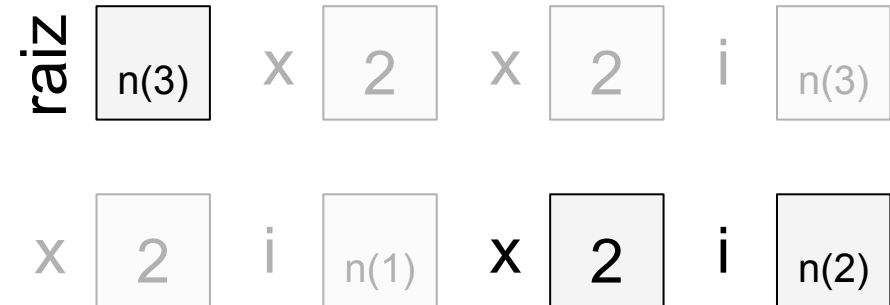
//remover(2), folha

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

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

false: 2 > 2

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



# Algoritmo de Remoção em C#

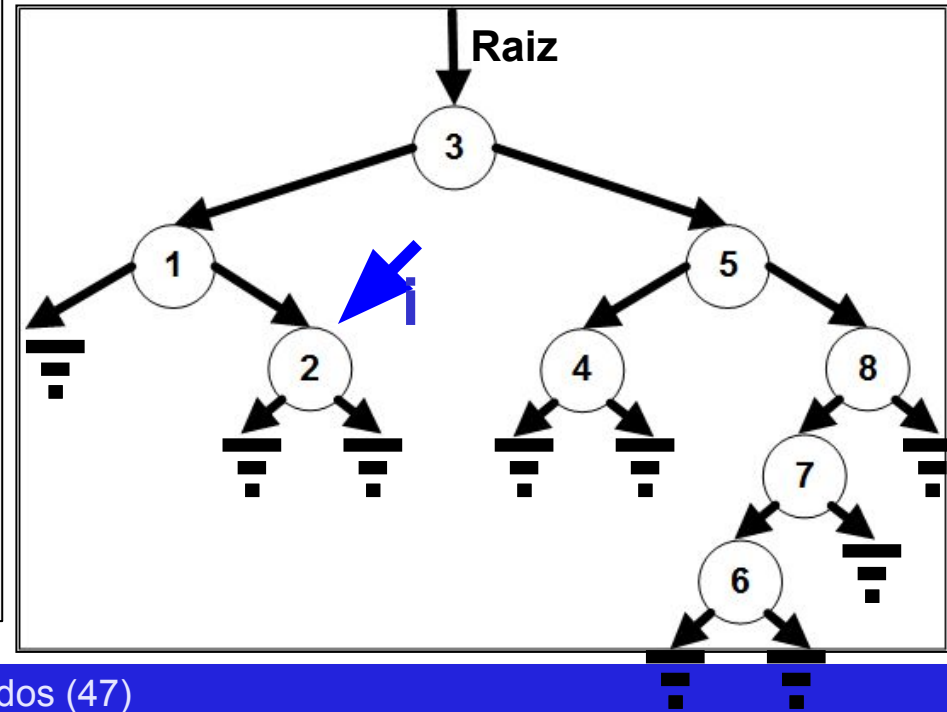
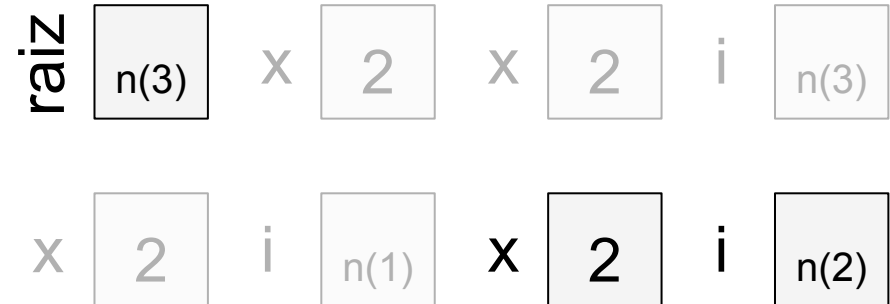
//remover(2), folha

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

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

true: null == null

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



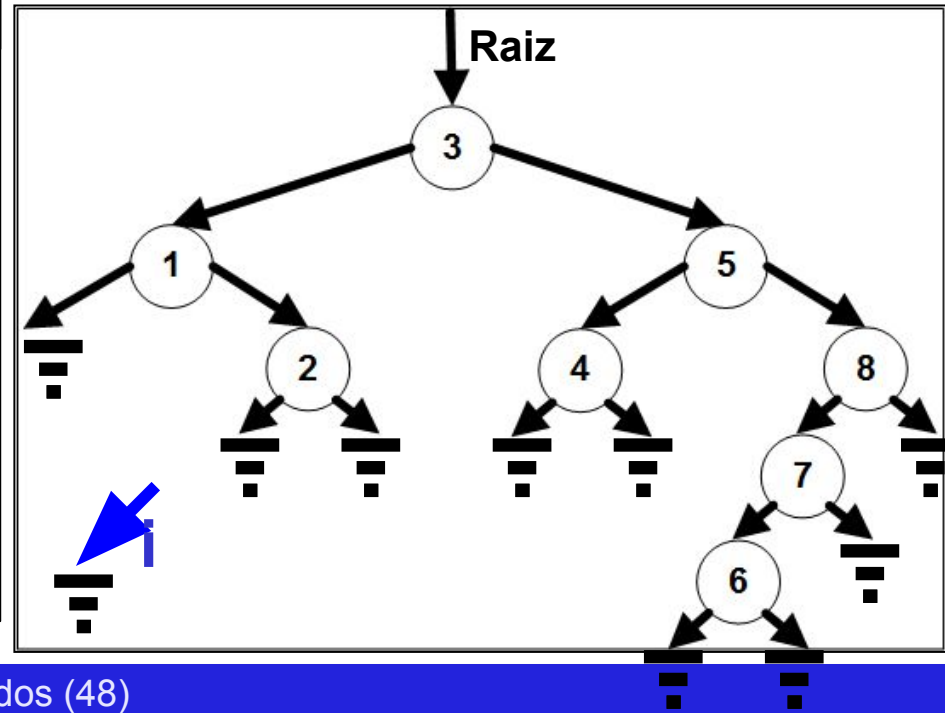
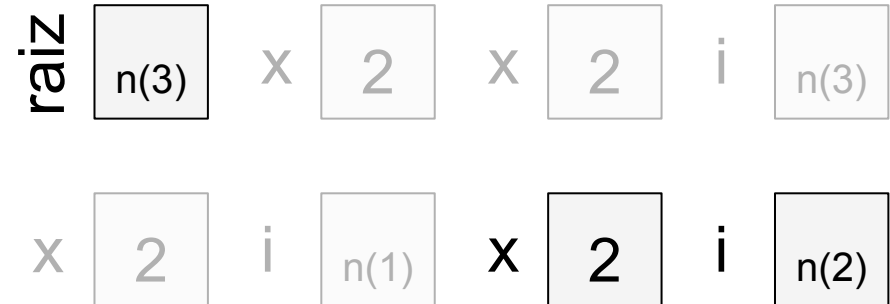
# Algoritmo de Remoção em C#

//remover(2), folha

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

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

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





# Algoritmo de Remoção em C#

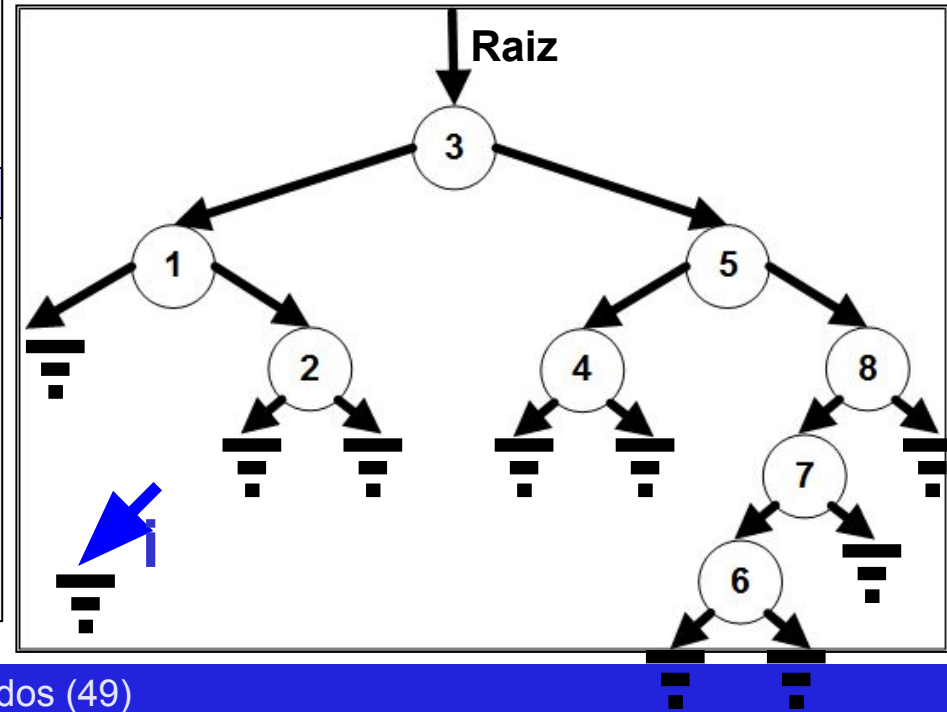
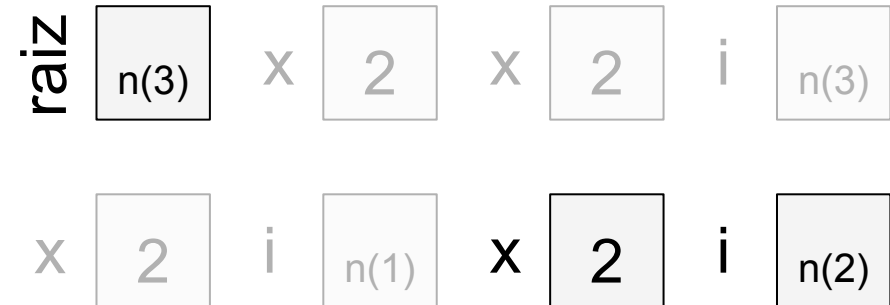
//remover(2), folha

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

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

Retorna null

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



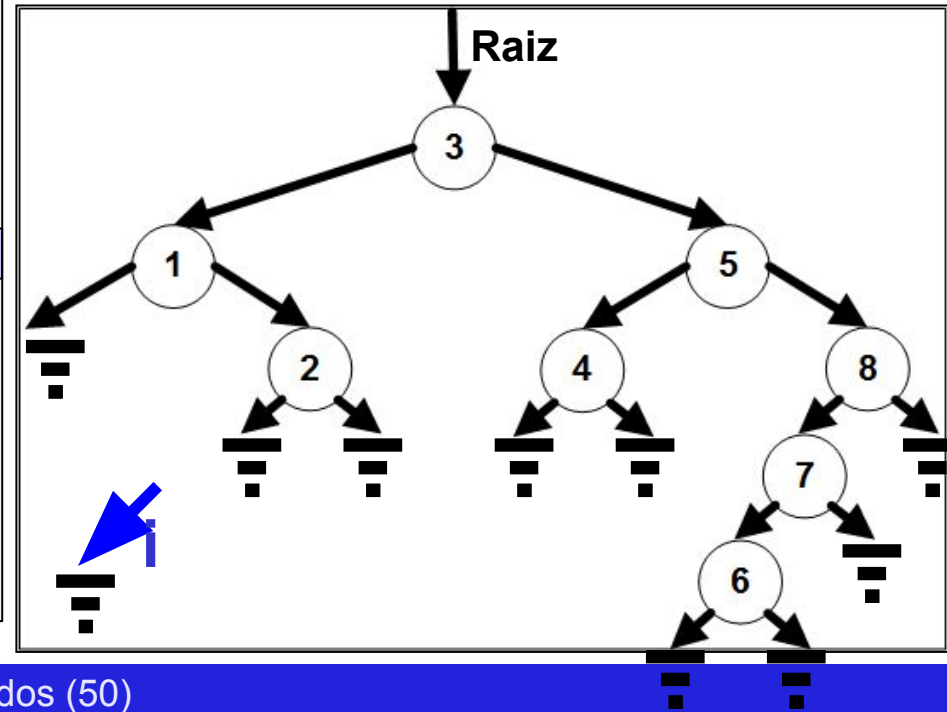
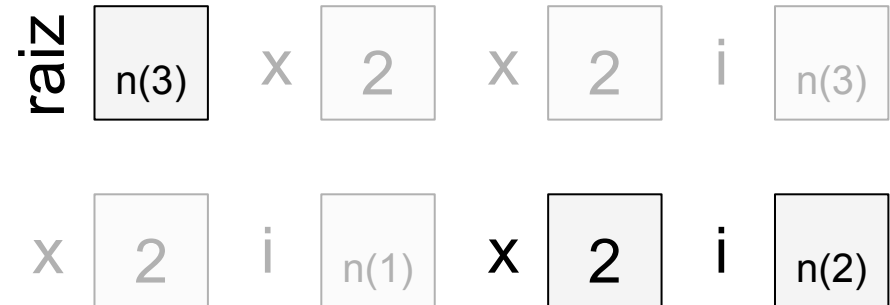
# Algoritmo de Remoção em C#

//remover(2), folha

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

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

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



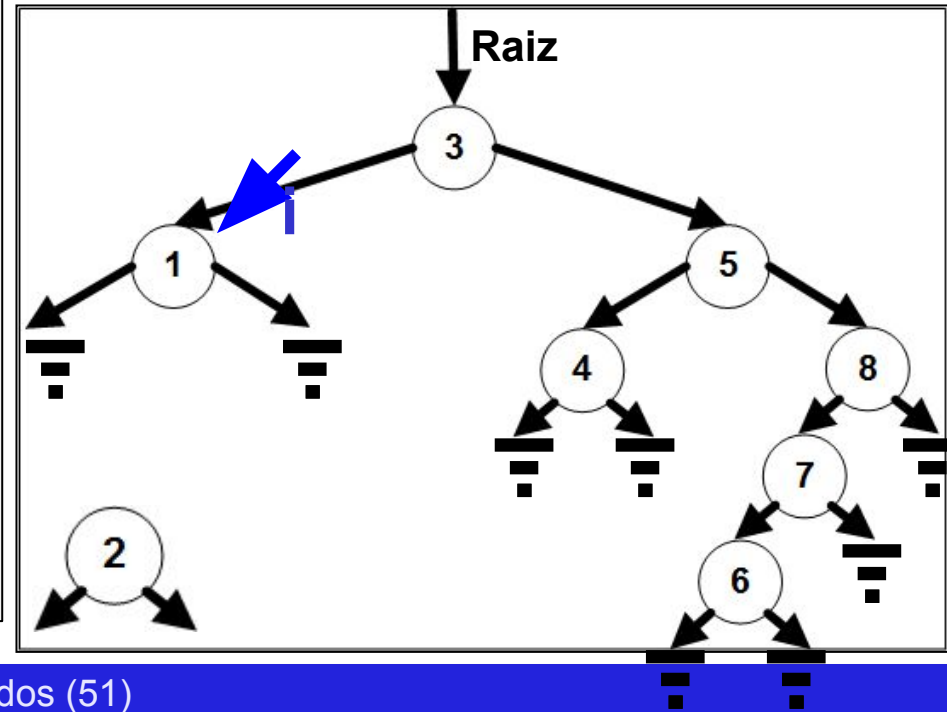
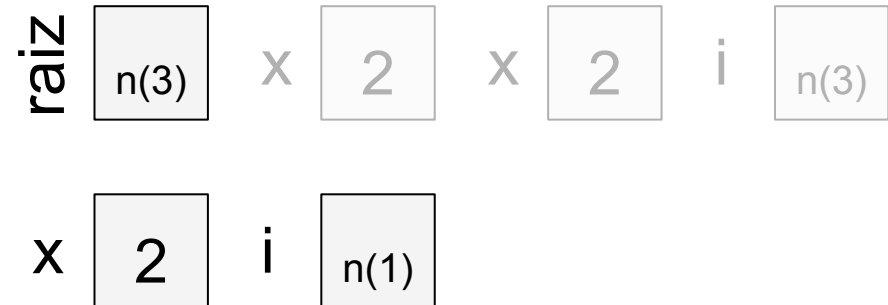
# Algoritmo de Remoção em C#

//remover(2), folha

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

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

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



# Algoritmo de Remoção em C#

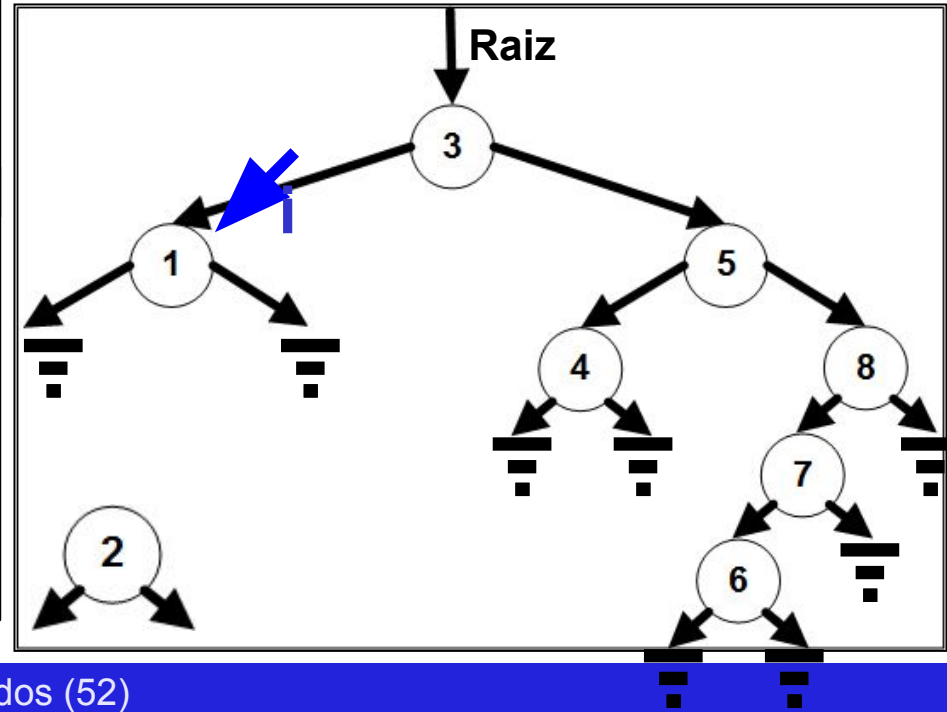
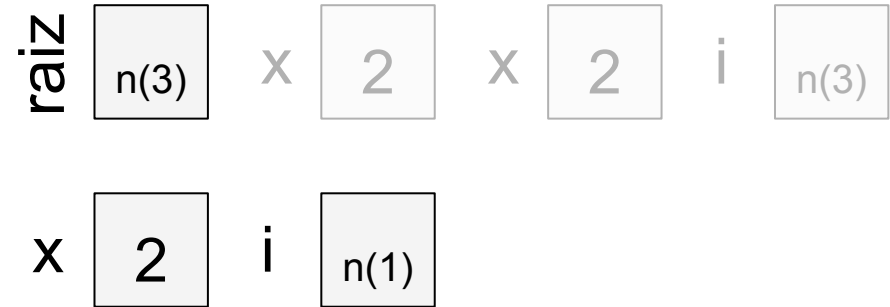
```
//remove(2), folha
```

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

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

Retorna  $n(1)$

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



# Algoritmo de Remoção em C#

```
//remover(2), folha
```

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

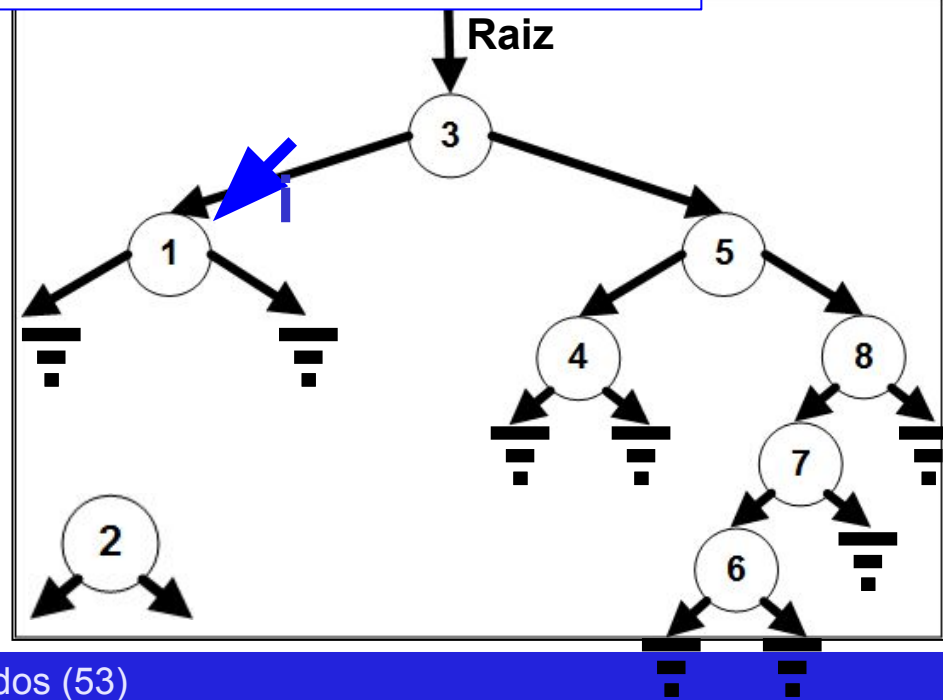
No remover

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

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

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

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



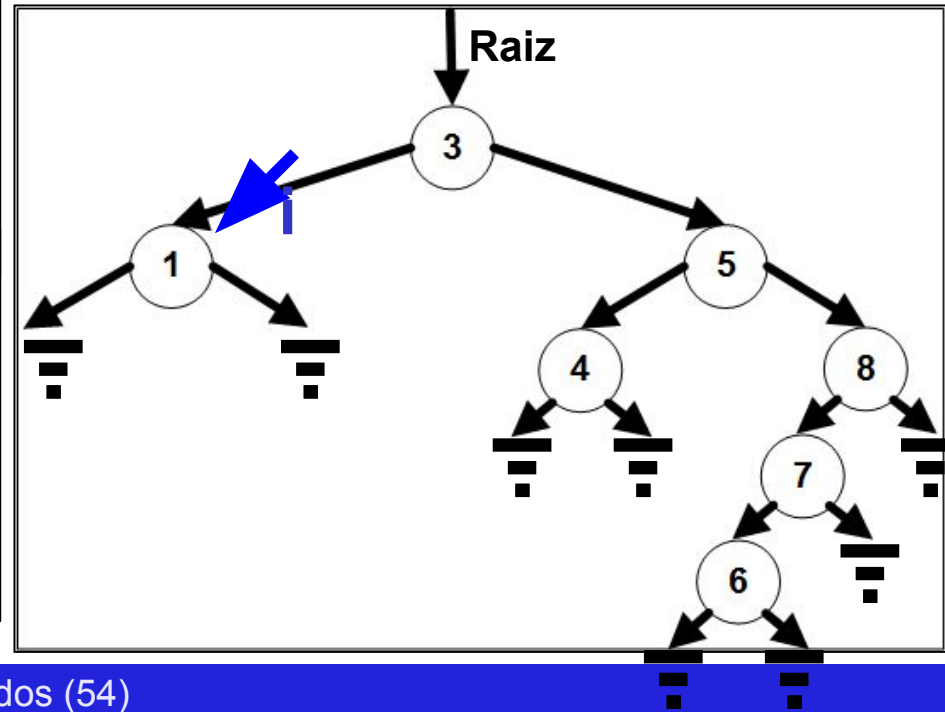
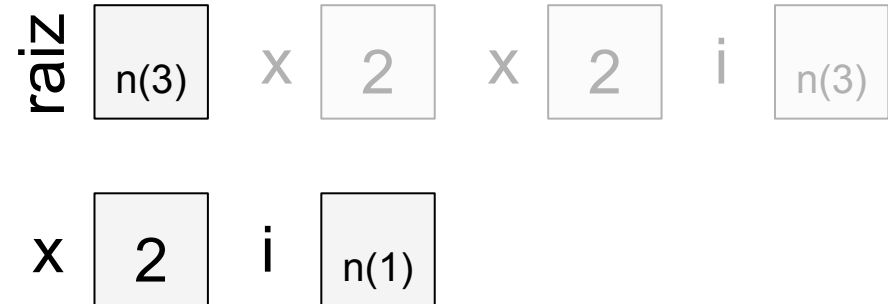
# Algoritmo de Remoção em C#

//remover(2), folha

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

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

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



# Algoritmo de Remoção em C#

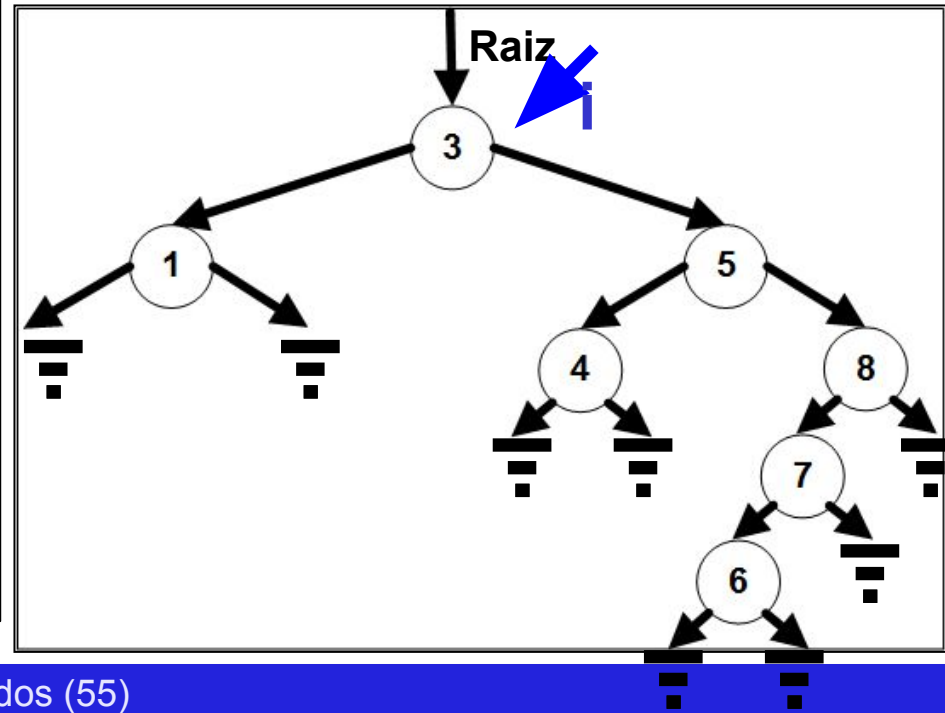
//remover(2), folha

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

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

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

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



# Algoritmo de Remoção em C#

//remover(2), folha

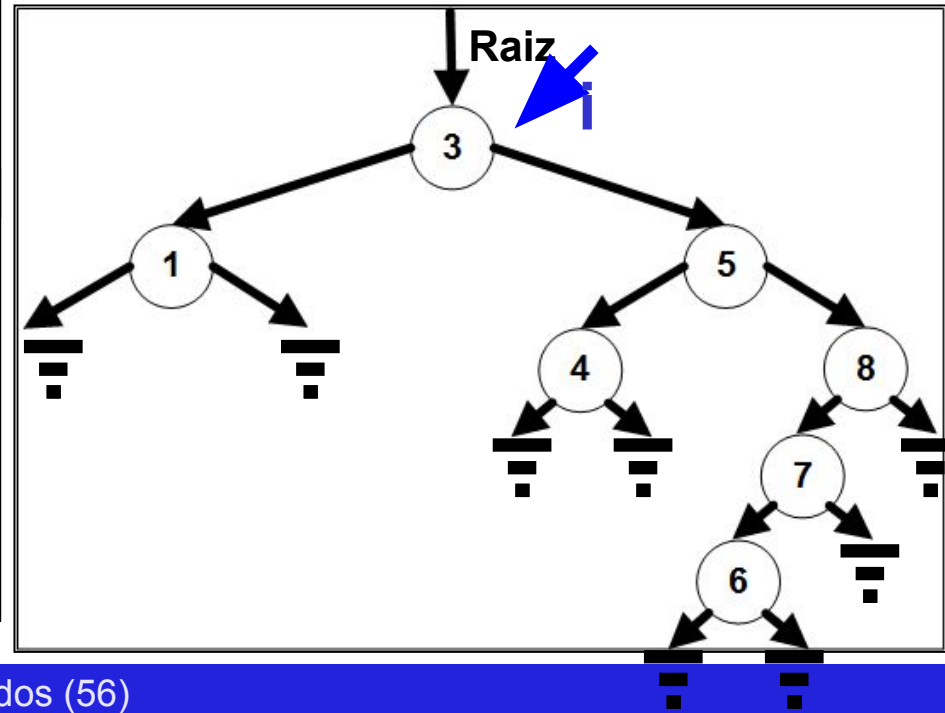
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

Retorna n(3)

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

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





# Algoritmo de Remoção em C#

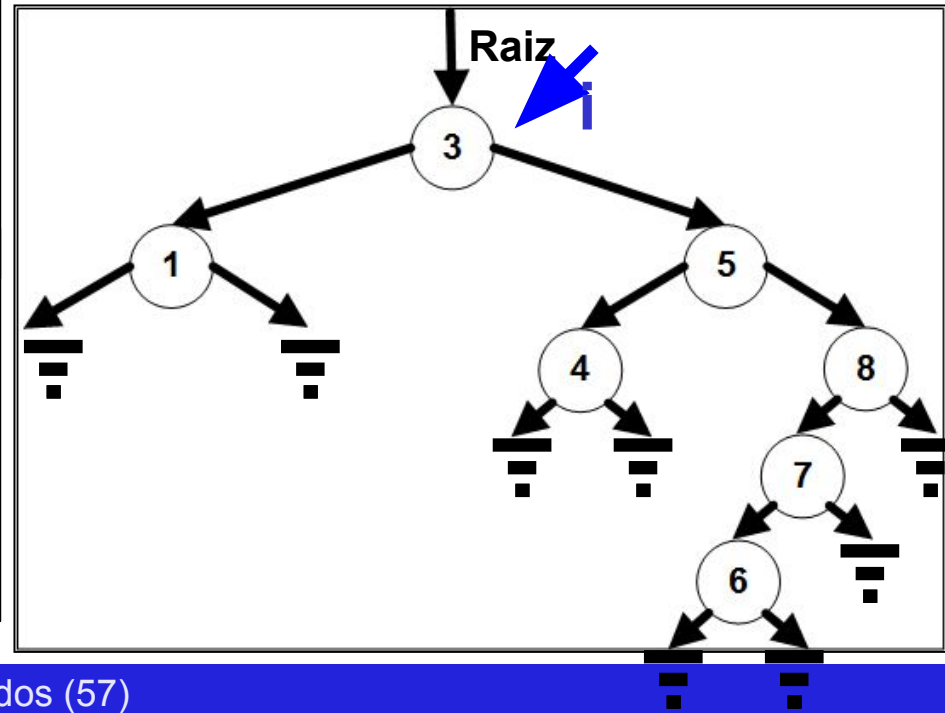
//remover(2), folha

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

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

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

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



# Algoritmo de Remoção em C#

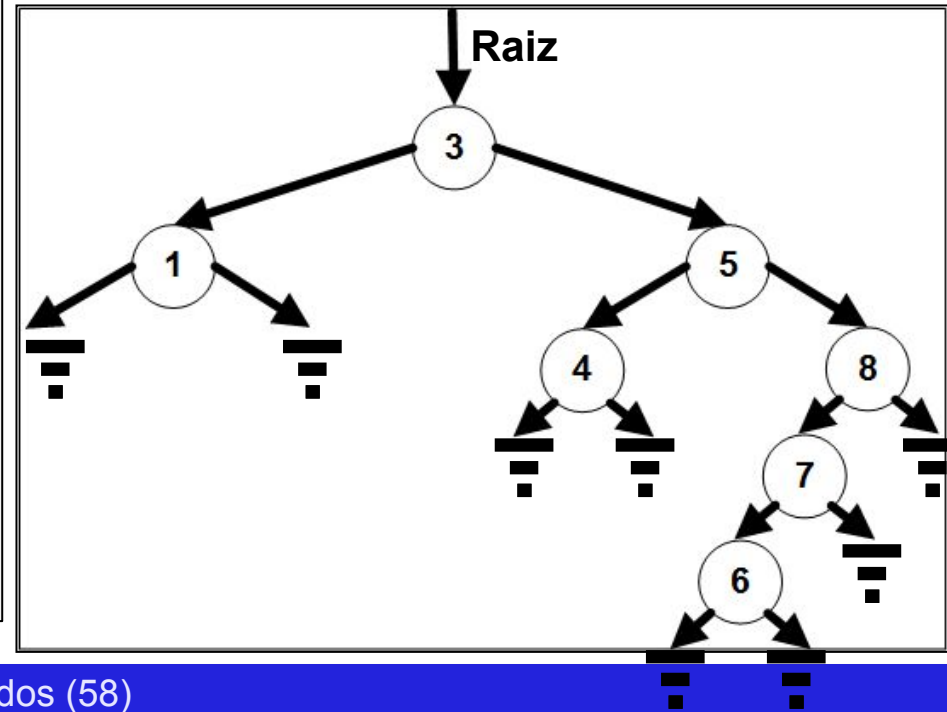
//remover(2), folha

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

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

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

raiz    n(3)    x    2



# Algoritmo de Remoção em C#

//remover(2), folha

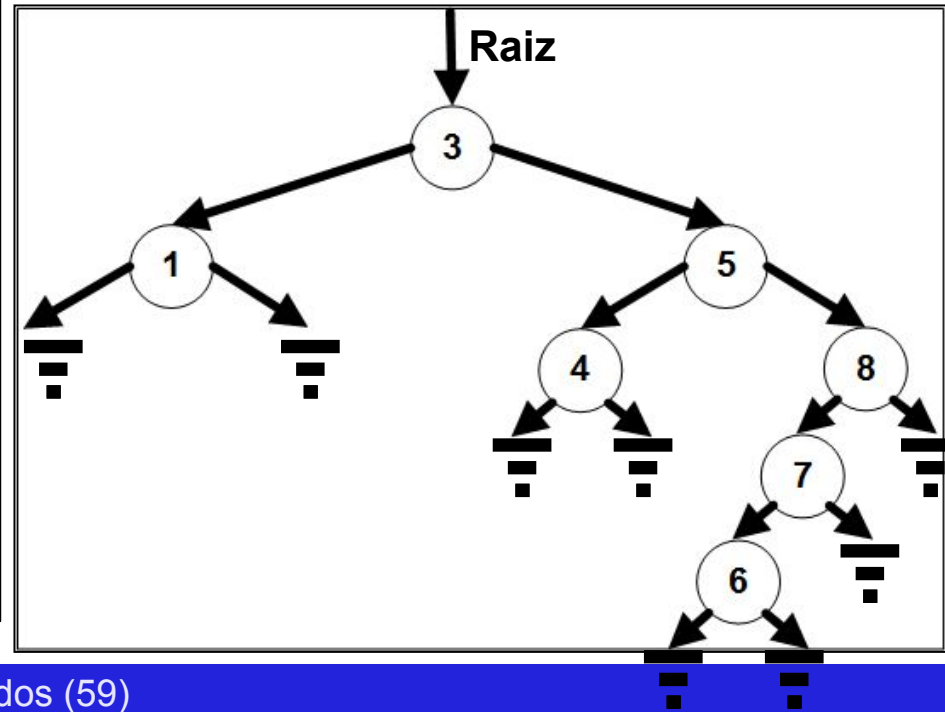
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

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

raiz

n(3)



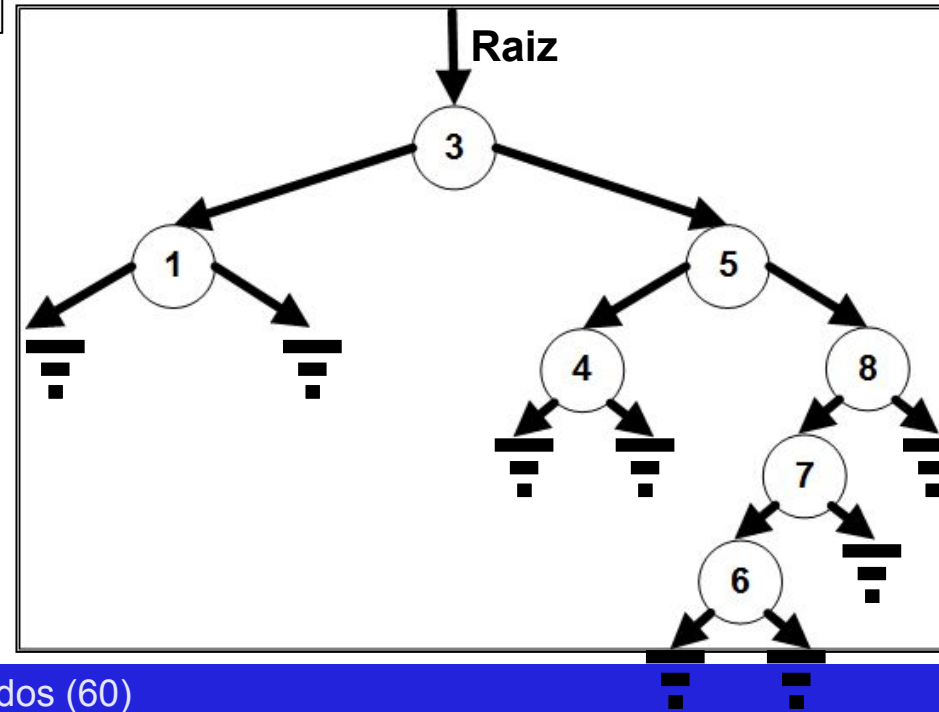
# Algoritmo de Remoção em C#

```
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() { }
}
```

raiz

n(3)

Voltando com o 2 antes  
de fazer outra remoção



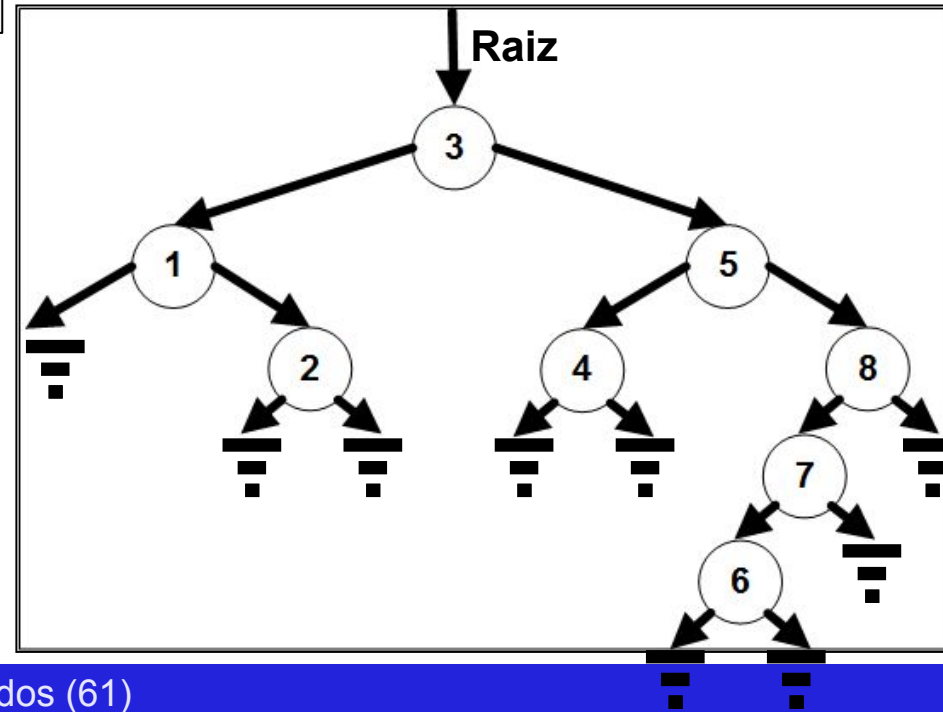
# Algoritmo de Remoção em C#

```
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() { }
}
```

raiz

n(3)

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



# Algoritmo de Remoção em C#

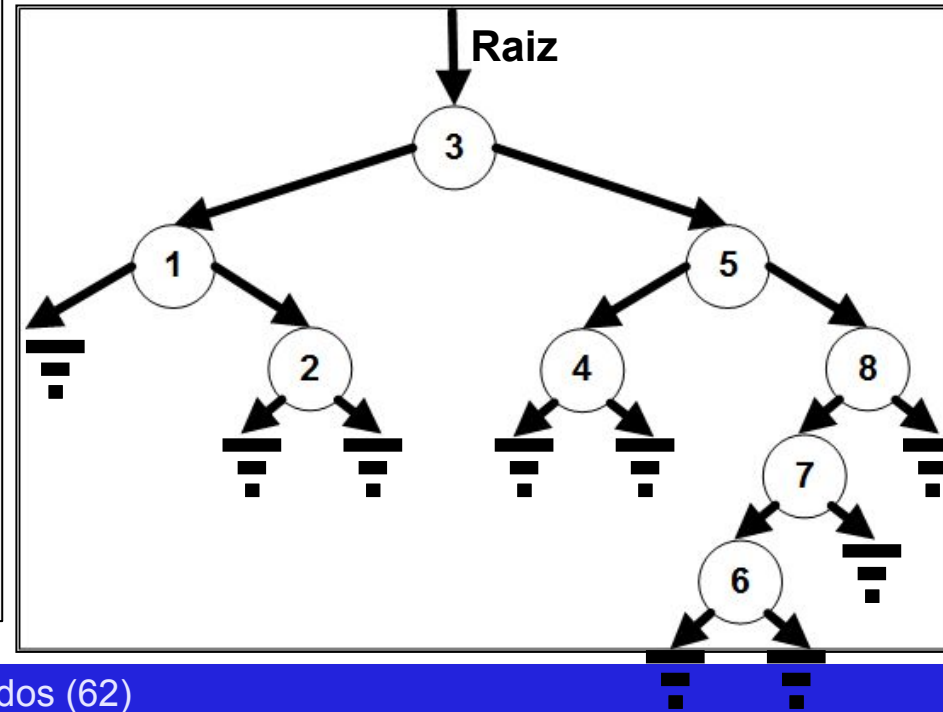
//remover(1), um filho

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

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

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

raiz     $n(3)$      $\times$      $1$



# Algoritmo de Remoção em C#

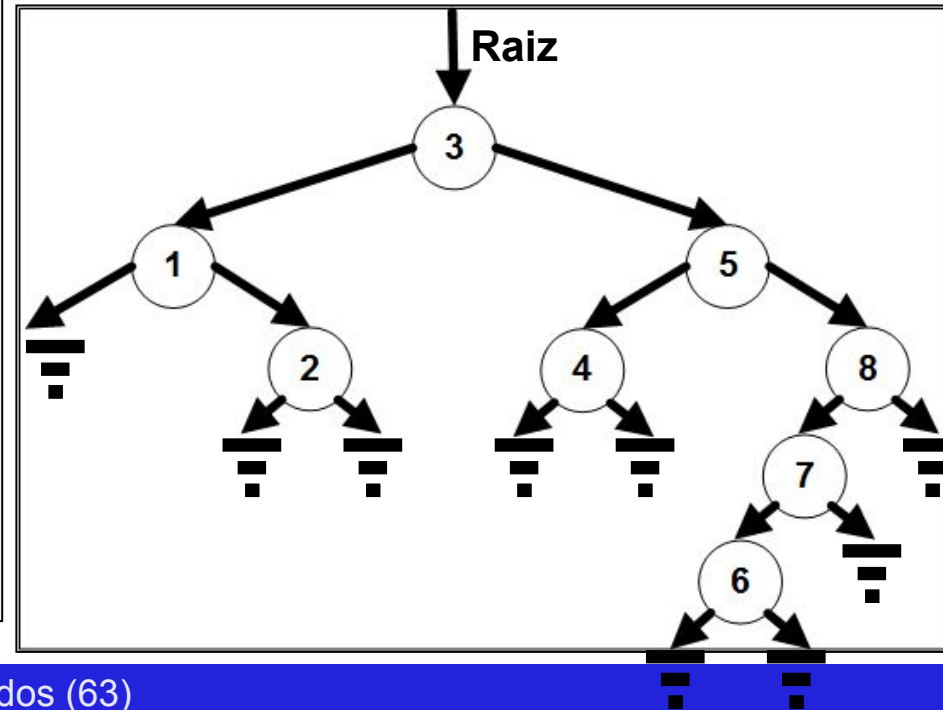
//remover(1), um filho

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

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

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

raiz    n(3)    x    1



# Algoritmo de Remoção em C#

//remover(1), um filho

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

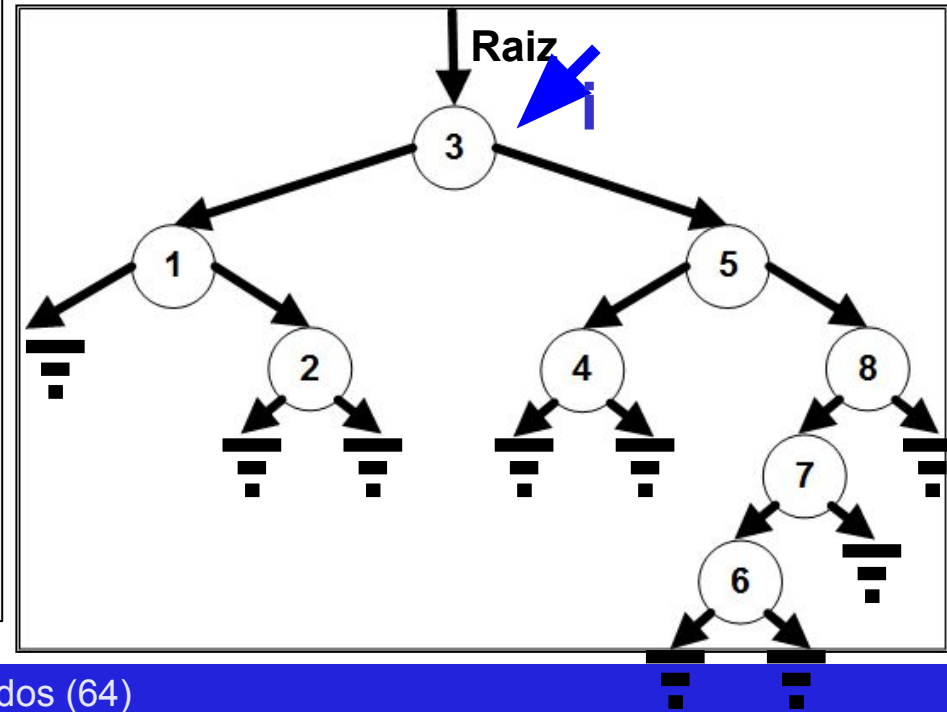
No remover(int x, No i) throws Exception {

```
    if (i == null) {        throw new Exception("Erro!");
    } else if(x < i.elemento) { i.esq = remover(x, i.esq);
    } else if(x > i.elemento) { i.dir = remover(x, i.dir);
    } else if(i.dir == null) {    i = i.esq;
    } else if(i.esq == null) {    i = i.dir;
    } else {
        i.esq = maiorEsq(i, i.esq);
    }
    return i;
}
```

No maiorEsq(No i, No j) {

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

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





# Algoritmo de Remoção em C#

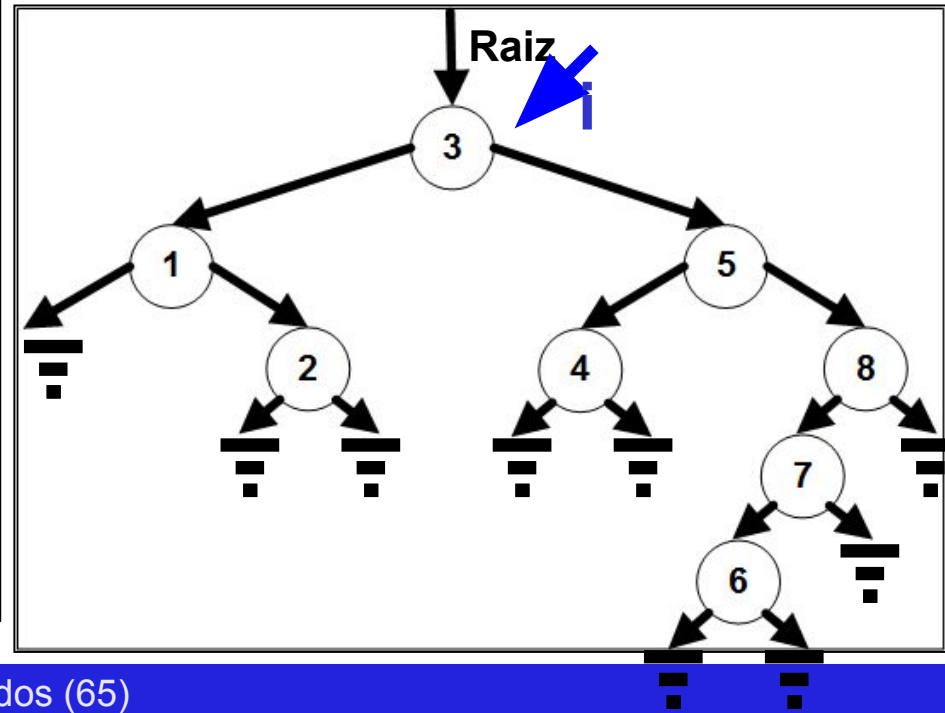
//remover(1), um filho

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

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

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

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



# Algoritmo de Remoção em C#

//remover(1), um filho

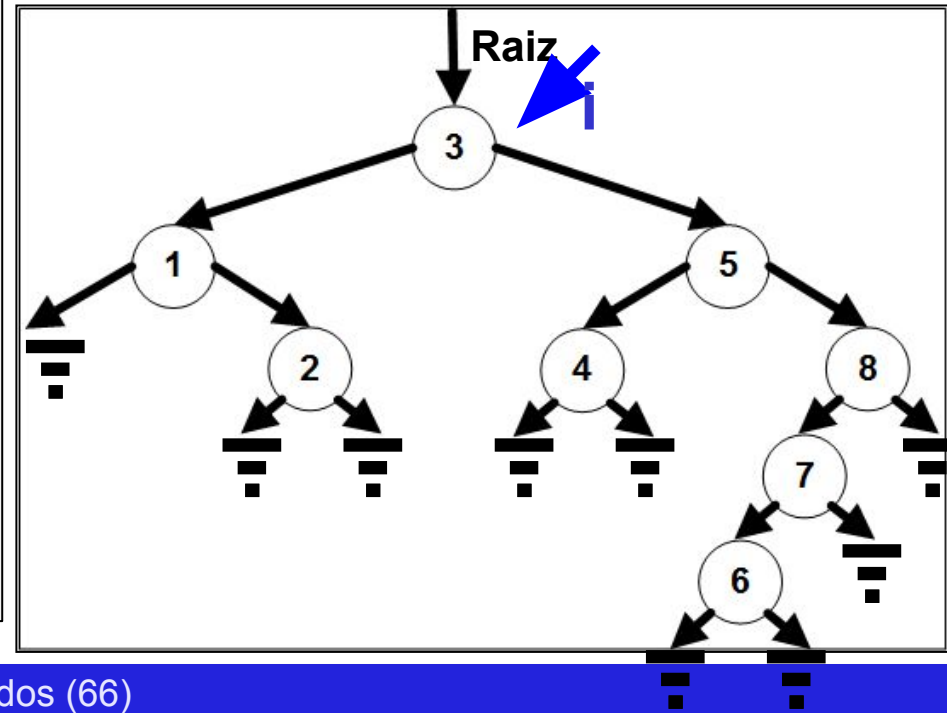
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

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

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

true: 1 < 3

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



# Algoritmo de Remoção em C#

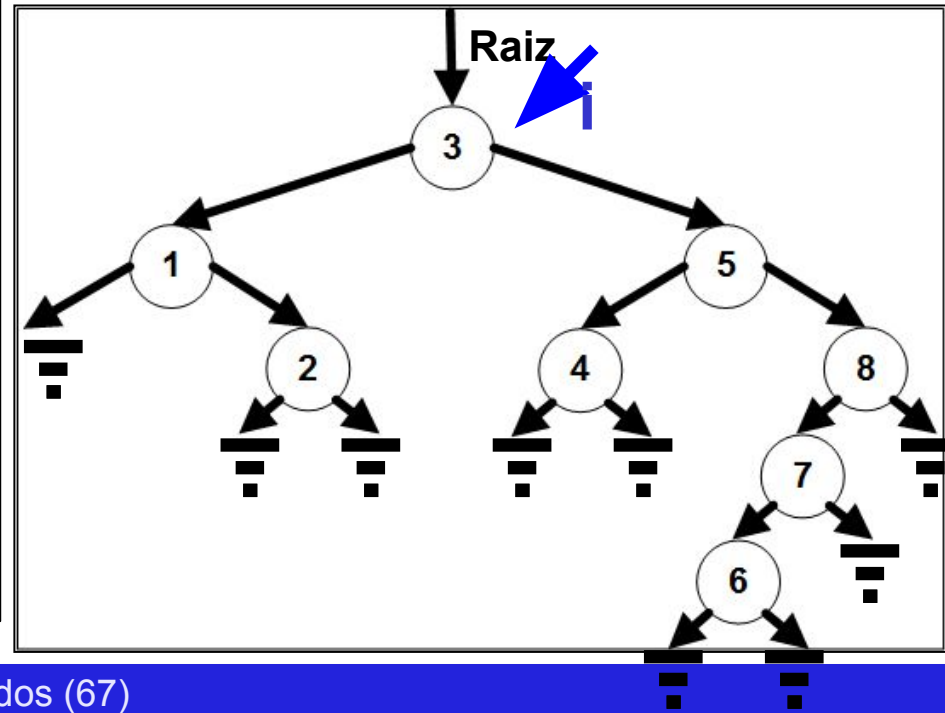
//remover(1), um filho

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

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

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

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



# Algoritmo de Remoção em C#

//remover(1), um filho

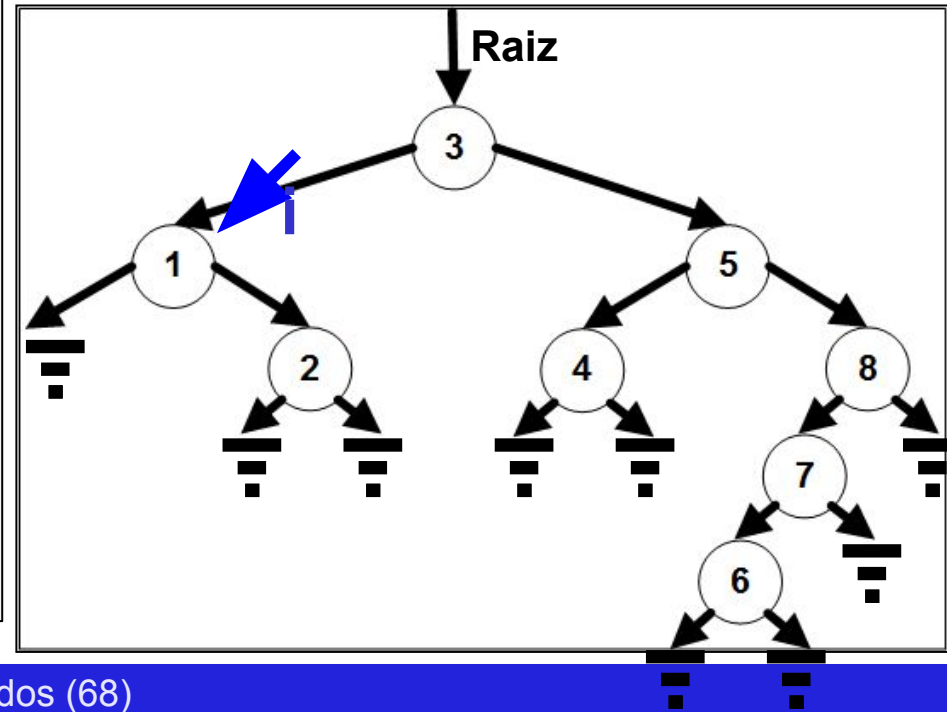
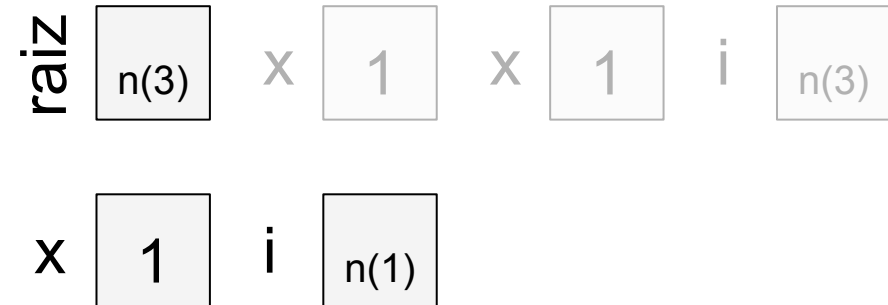
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

```
    if (i == null) { throw new Exception("Erro!");
    } else if (x < i.elemento) { i.esq = remover(x, i.esq);
    } else if (x > i.elemento) { i.dir = remover(x, i.dir);
    } else if (i.dir == null) { i = i.esq;
    } else if (i.esq == null) { i = i.dir;
    } else {
        i.esq = maiorEsq(i, i.esq);
    }
    return i;
}
```

No maiorEsq(No i, No j) {

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



# Algoritmo de Remoção em C#

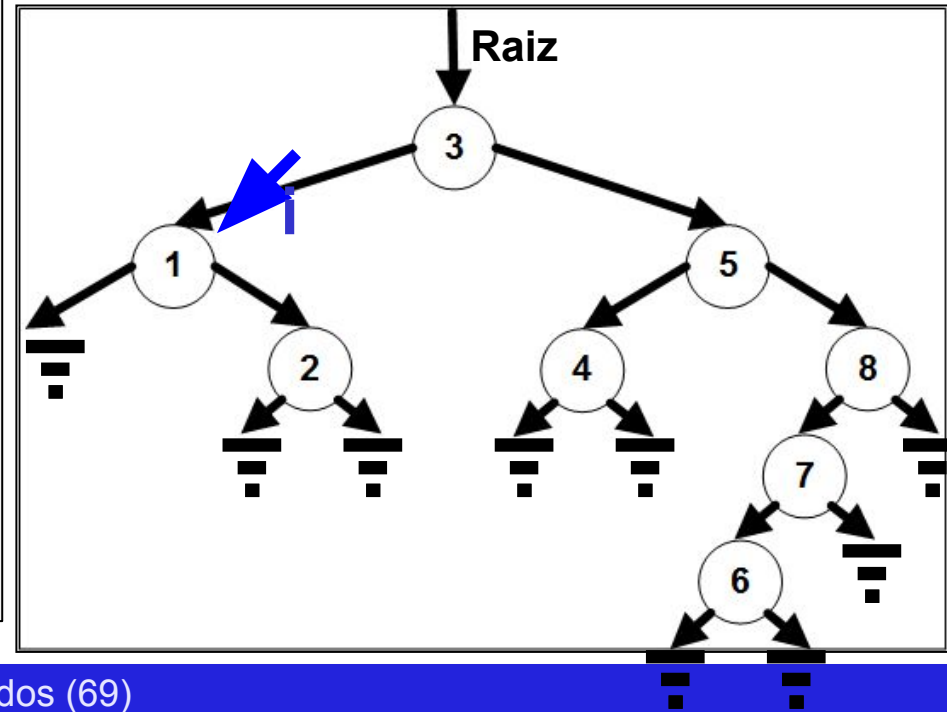
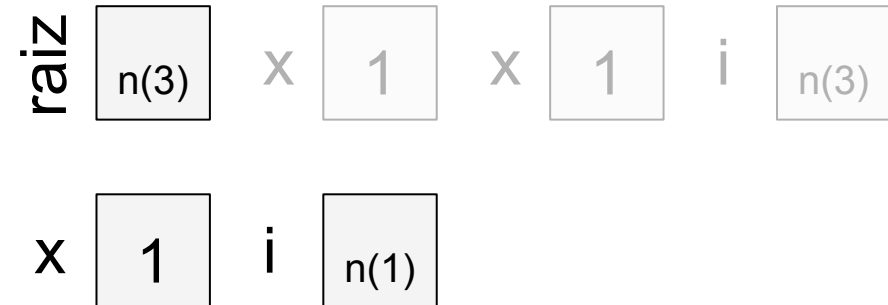
//remover(1), um filho

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

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

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

false: n(1) == null



# Algoritmo de Remoção em C#

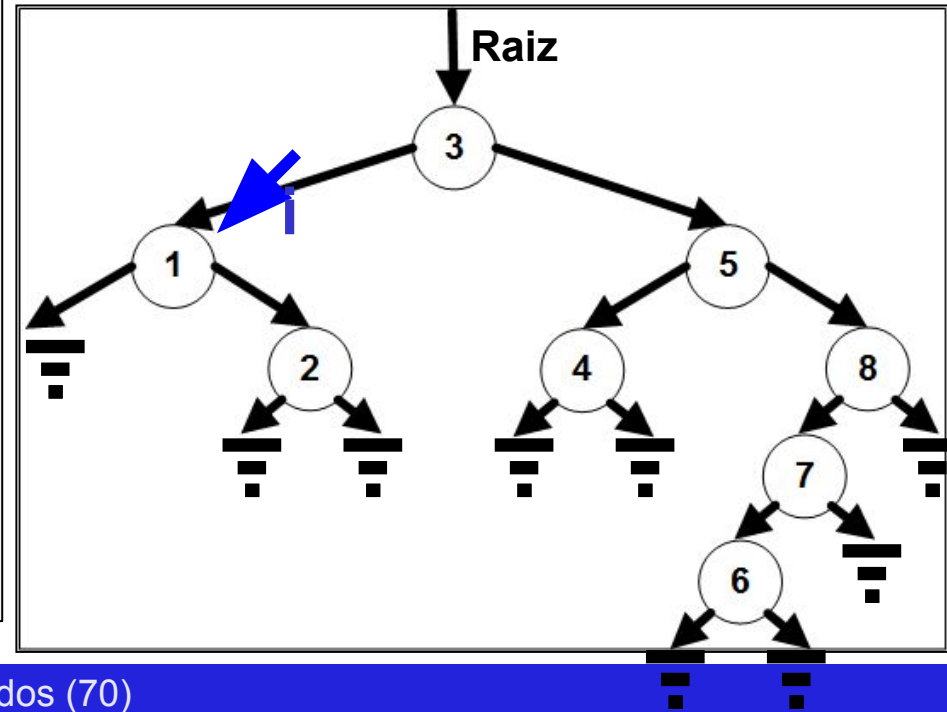
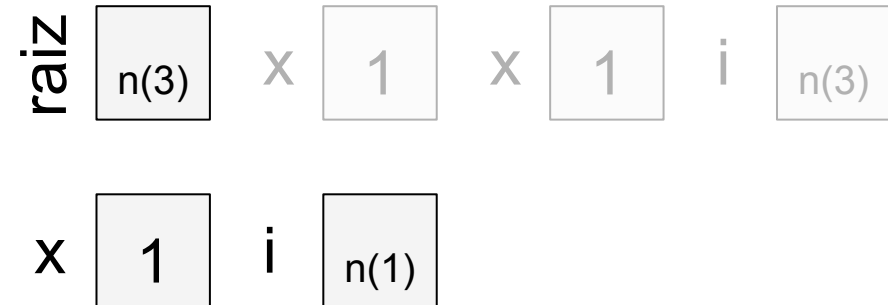
//remover(1), um filho

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

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

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

false: 1 < 1



# Algoritmo de Remoção em C#

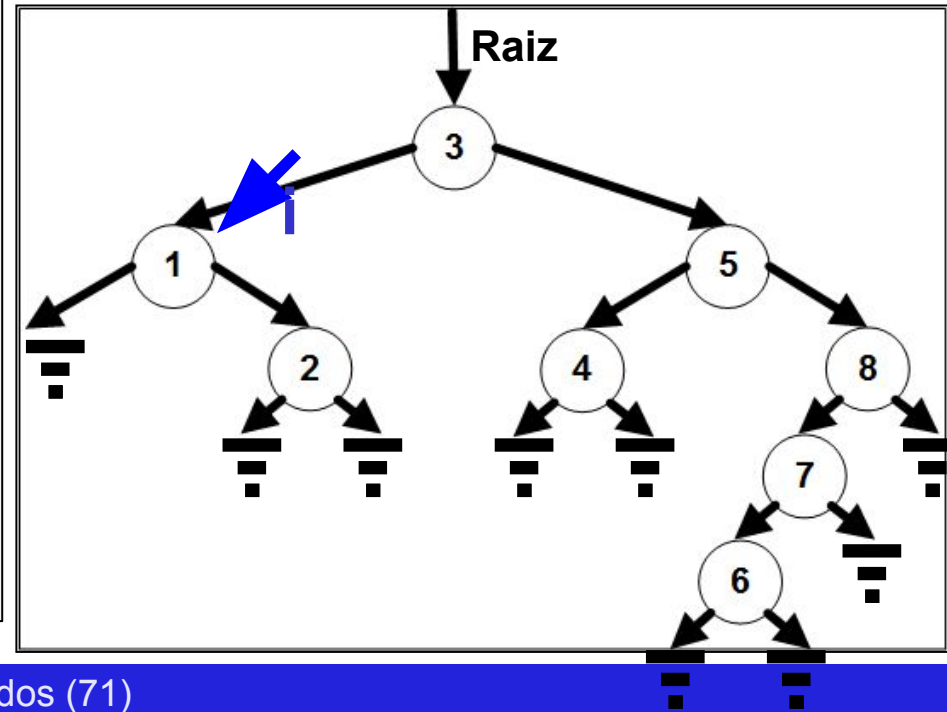
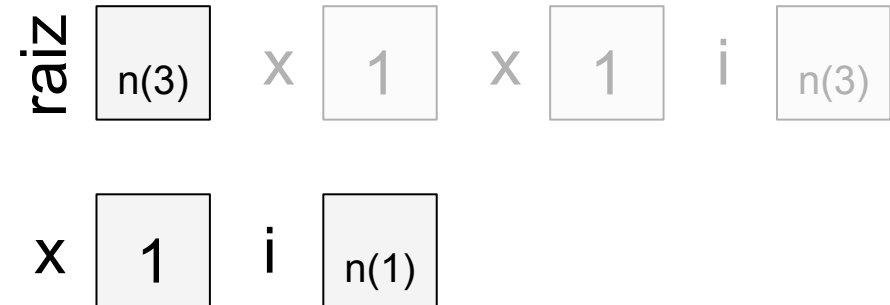
//remover(1), um filho

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

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

false: 1 > 1

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



# Algoritmo de Remoção em C#

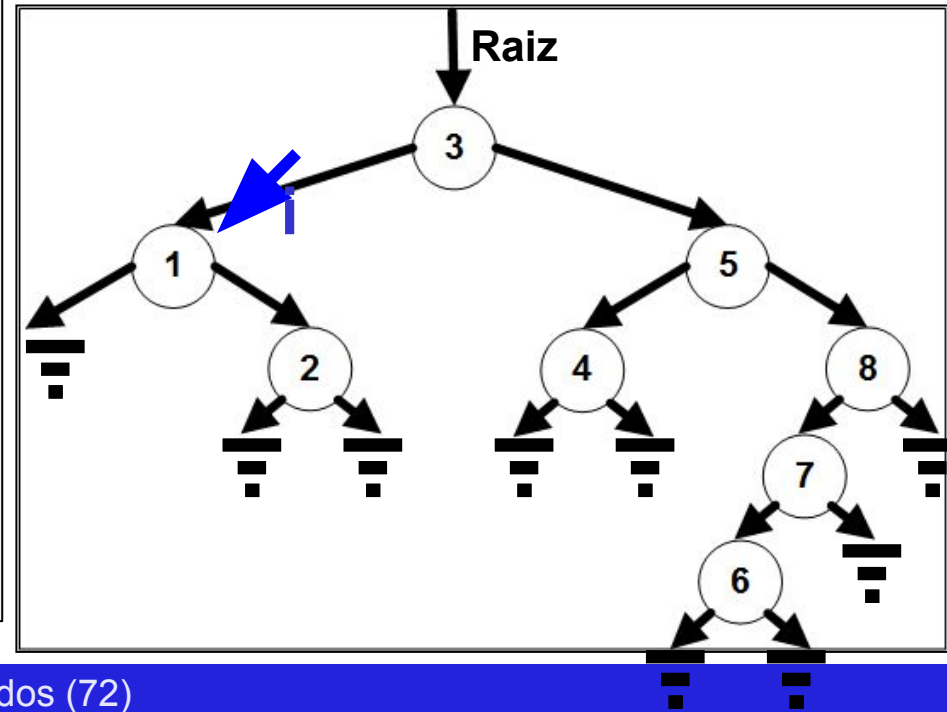
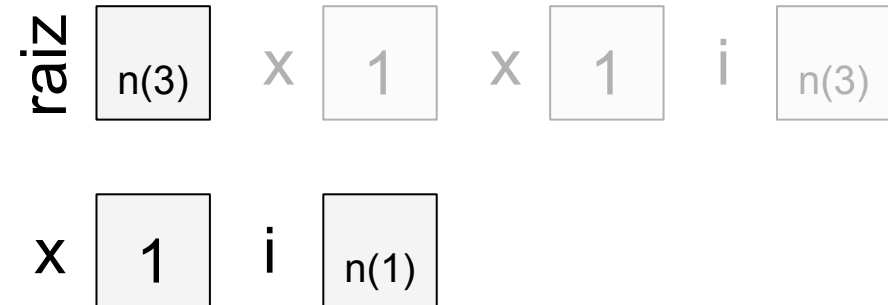
//remover(1), um filho

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

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

false: n(2) == null

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





# Algoritmo de Remoção em C#

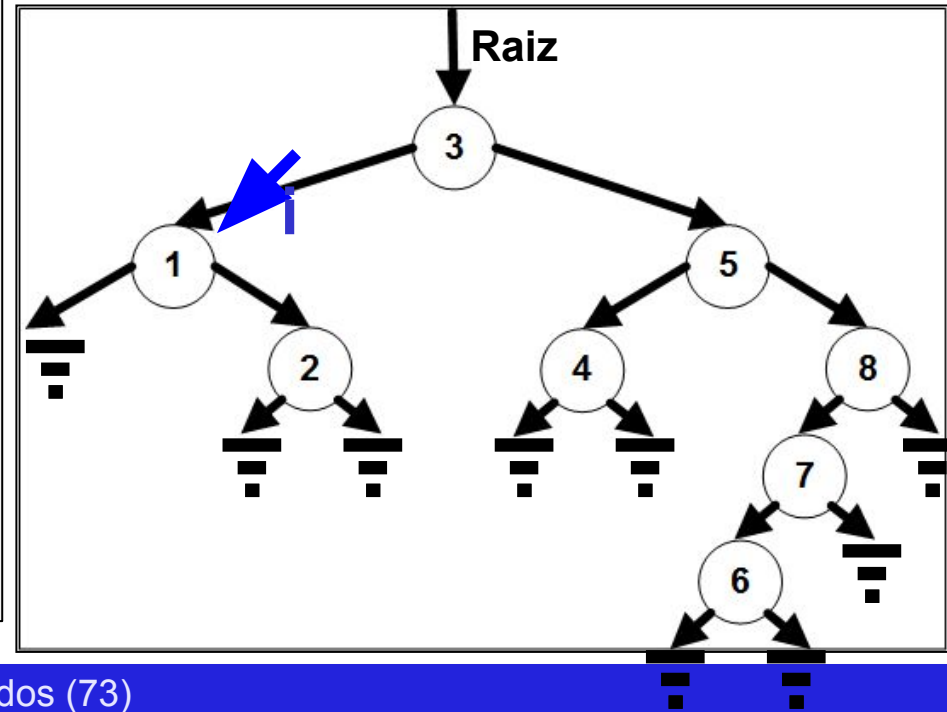
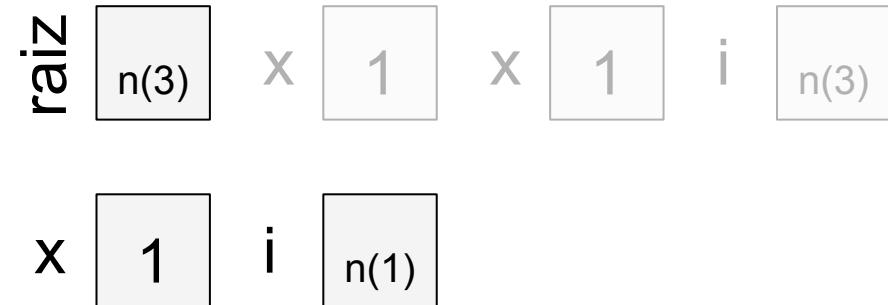
```
//remove(1), um filho
```

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

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

true: null == null

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



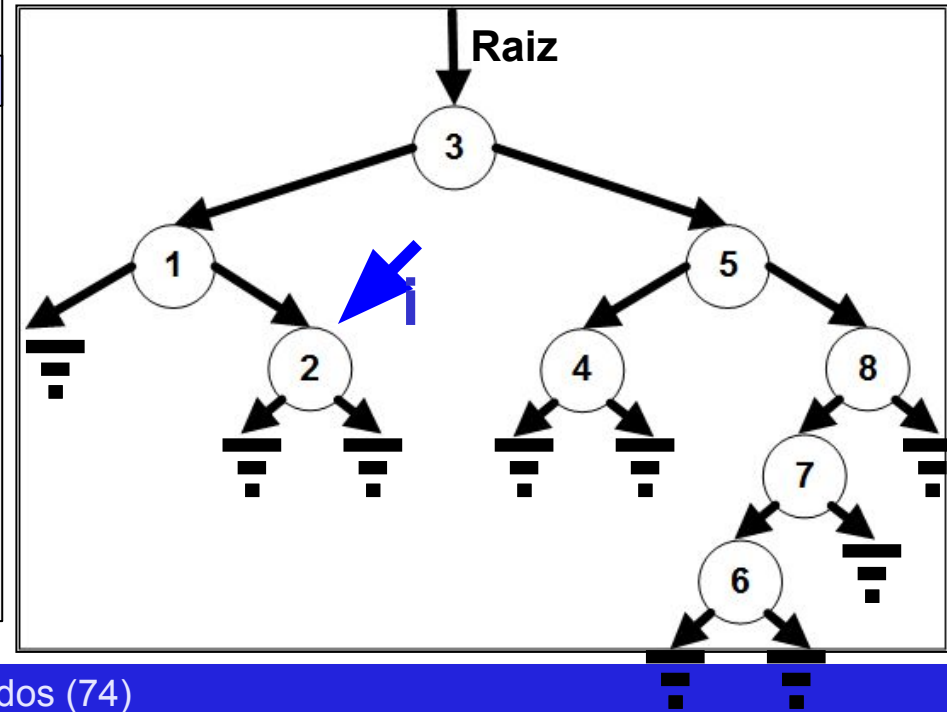
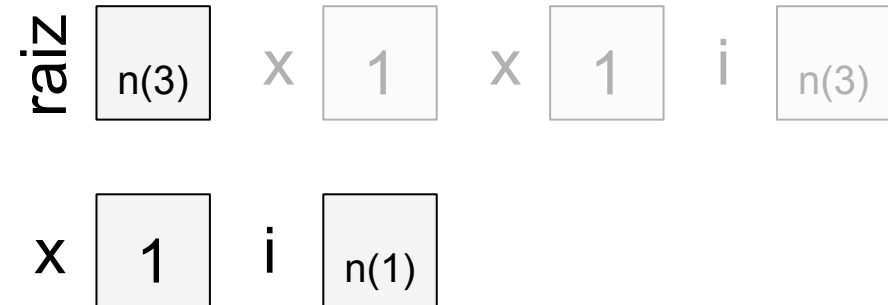
# Algoritmo de Remoção em C#

//remover(1), um filho

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

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

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



# Algoritmo de Remoção em C#

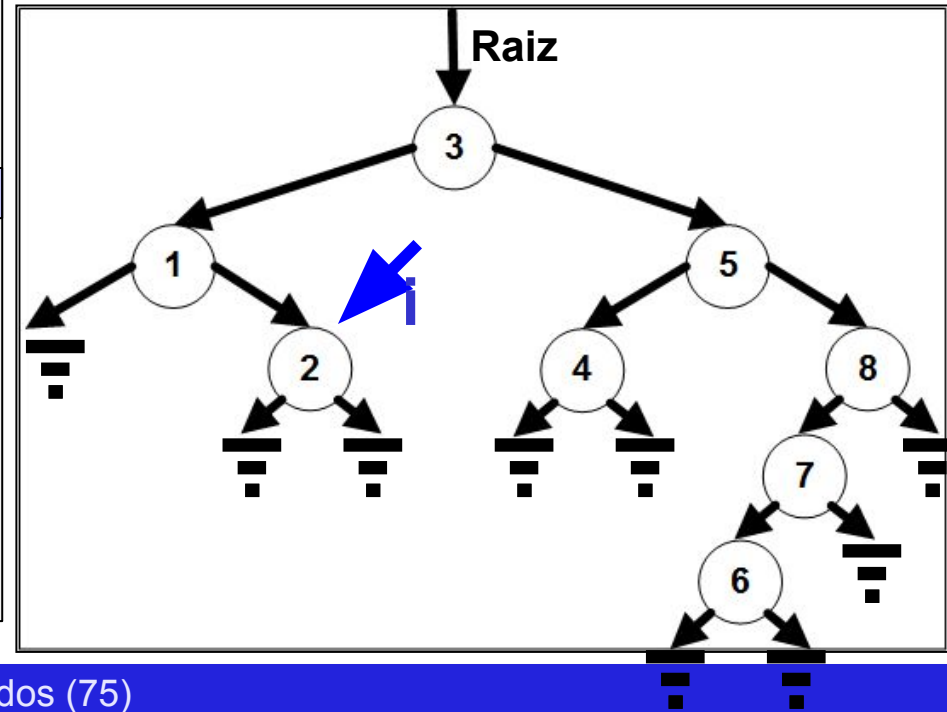
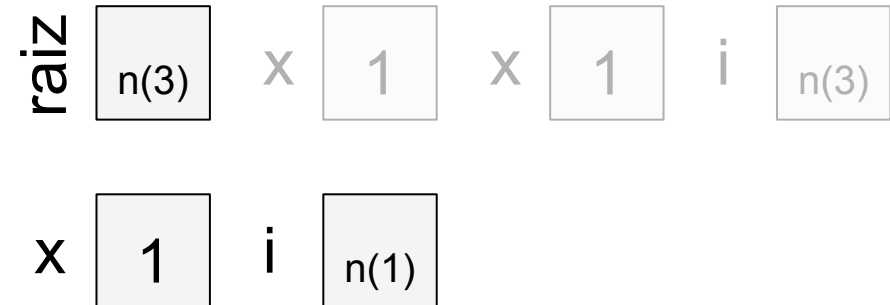
//remover(1), um filho

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

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

Retorna n(2)

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



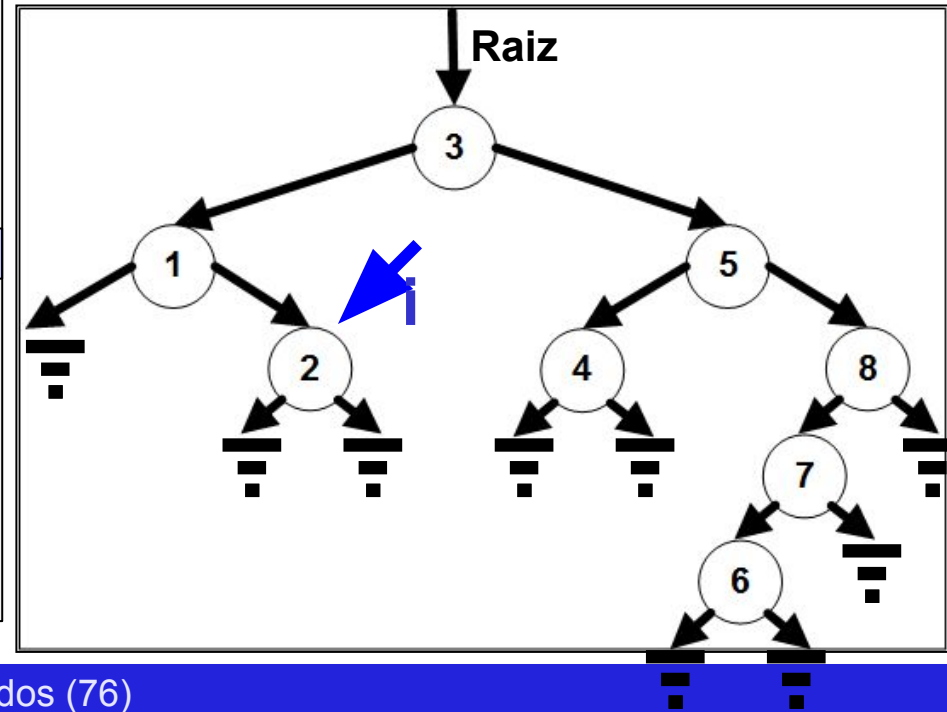
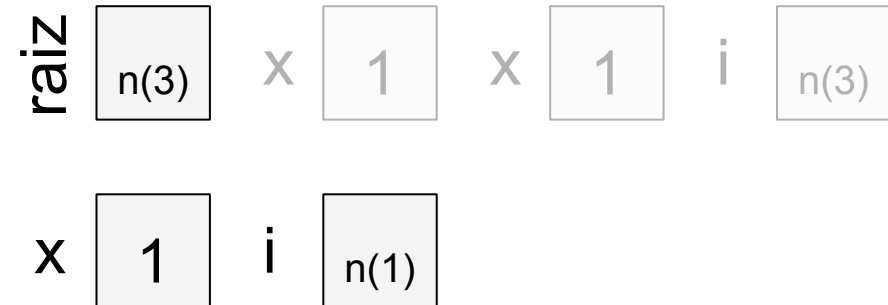
# Algoritmo de Remoção em C#

//remover(1), um filho

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

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

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



# Algoritmo de Remoção em C#

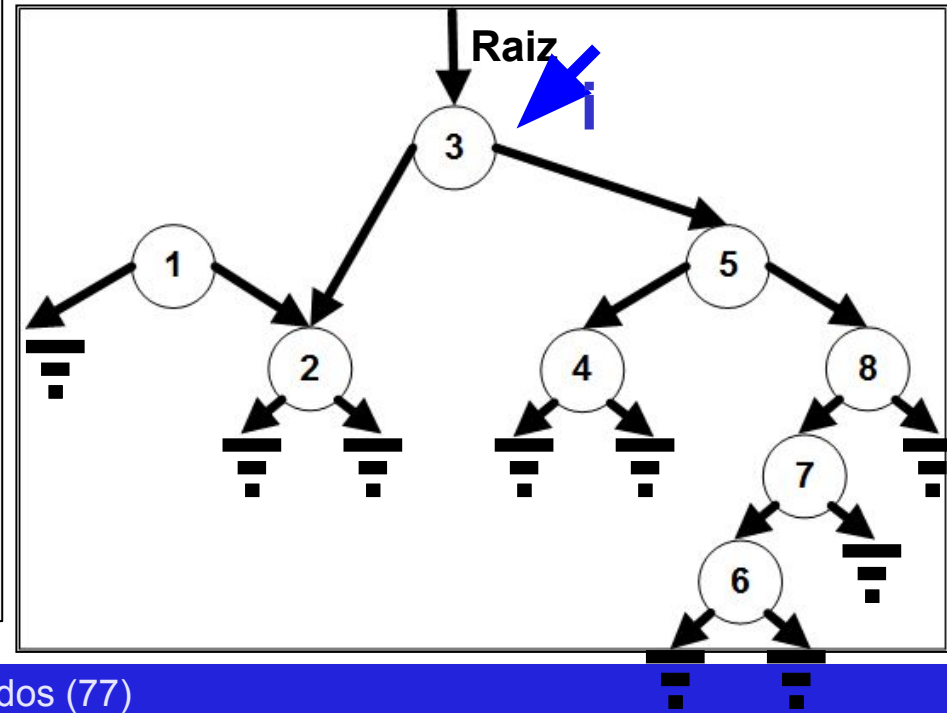
//remover(1), um filho

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

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

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

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



# Algoritmo de Remoção em C#

//remover(1), um filho

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

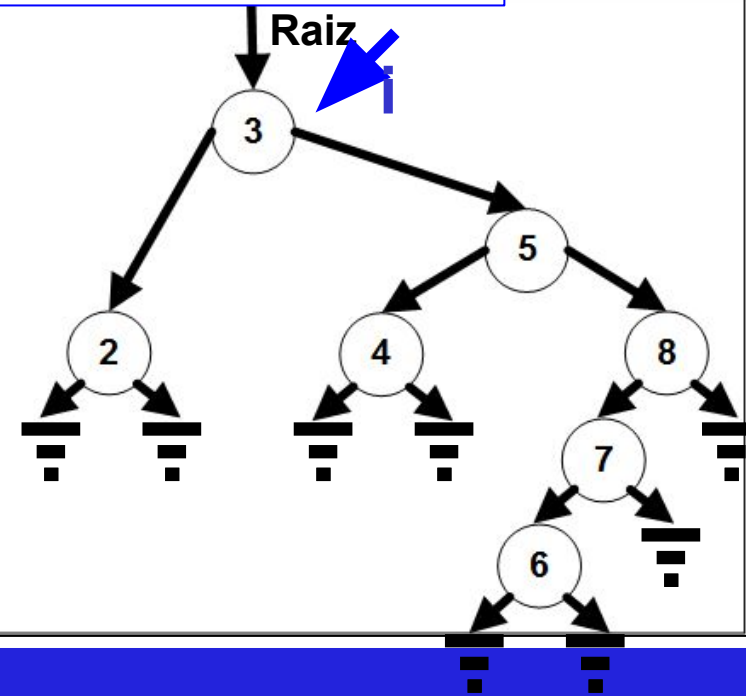
No remove

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

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

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

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



# Algoritmo de Remoção em C#

//remover(1), um filho

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

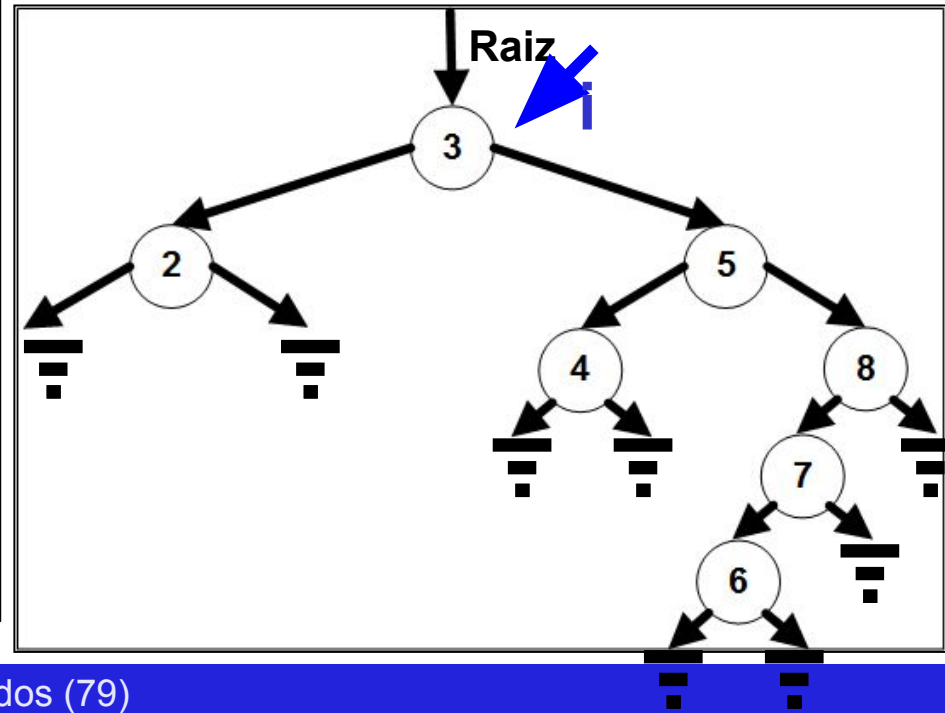
No remove

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

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

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

De uma forma mais organizada ...



# Algoritmo de Remoção em C#

//remover(1), um filho

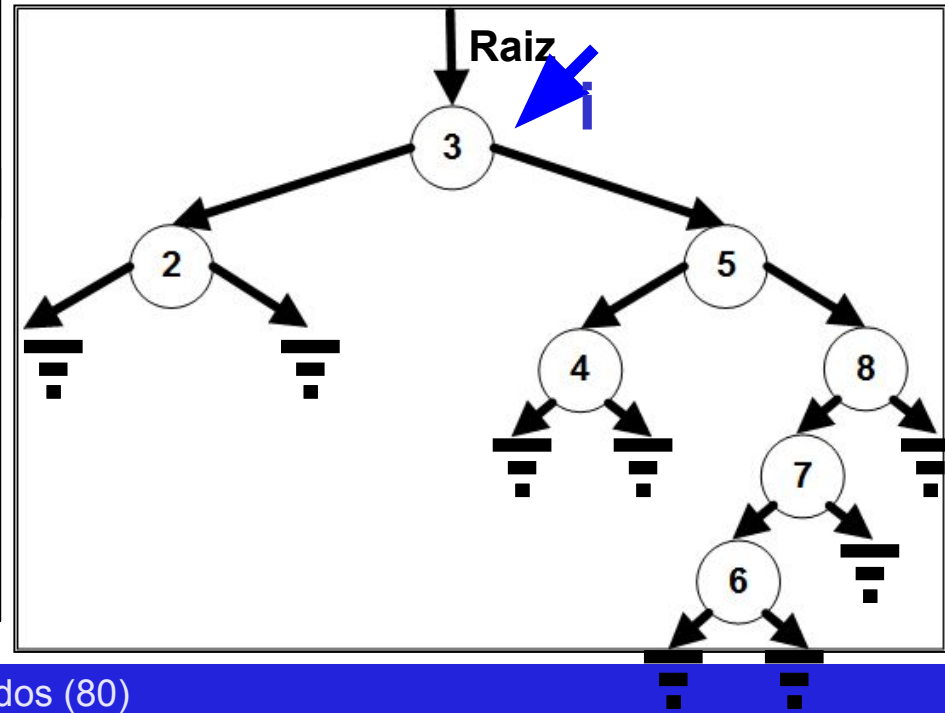
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

Retorna n(3)

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

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





# Algoritmo de Remoção em C#

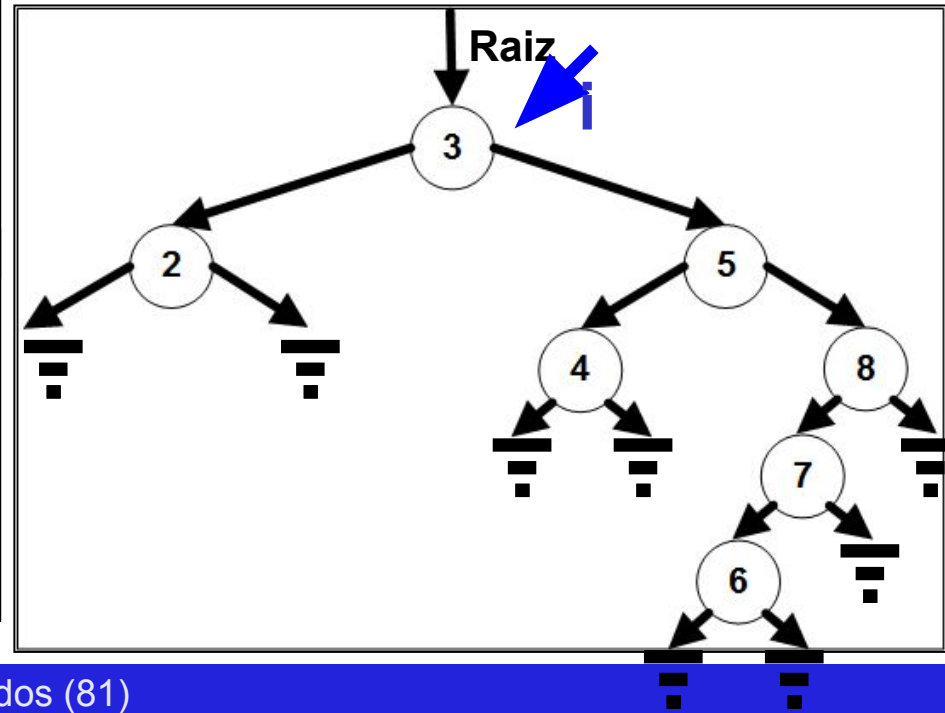
//remover(1), um filho

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

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

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

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



# Algoritmo de Remoção em C#

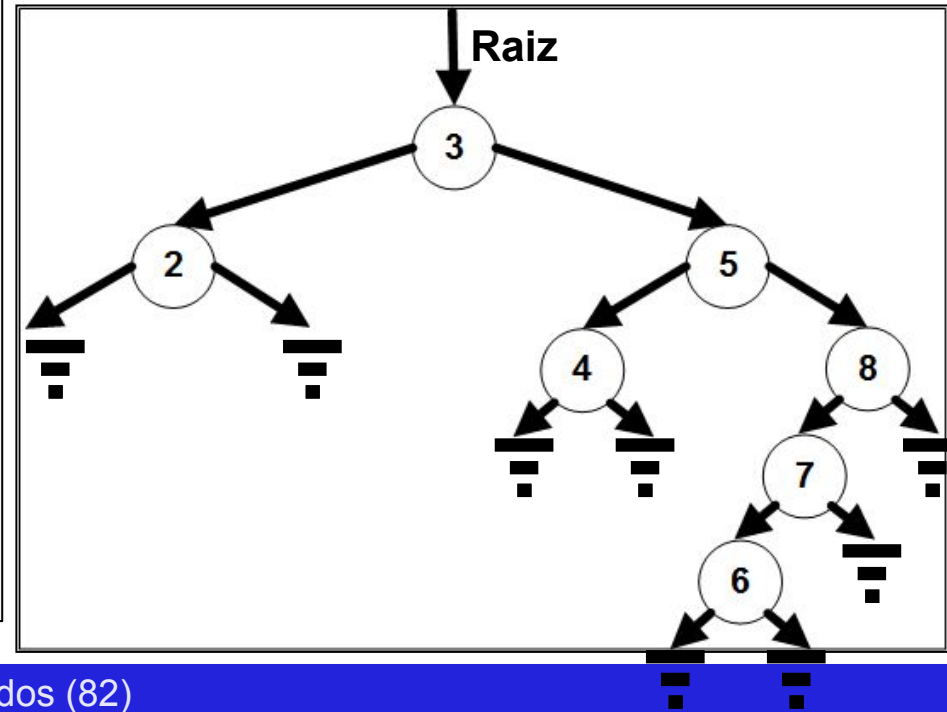
//remover(1), um filho

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

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

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

raiz    n(3)    x    1



# Algoritmo de Remoção em C#

//remover(1), um filho

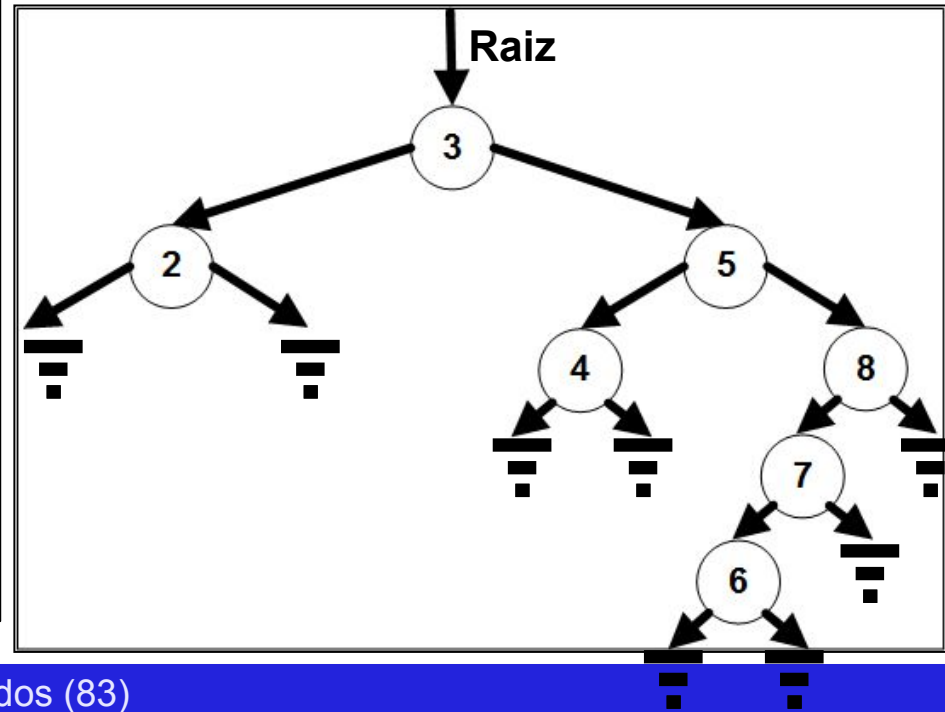
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

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

raiz

n(3)



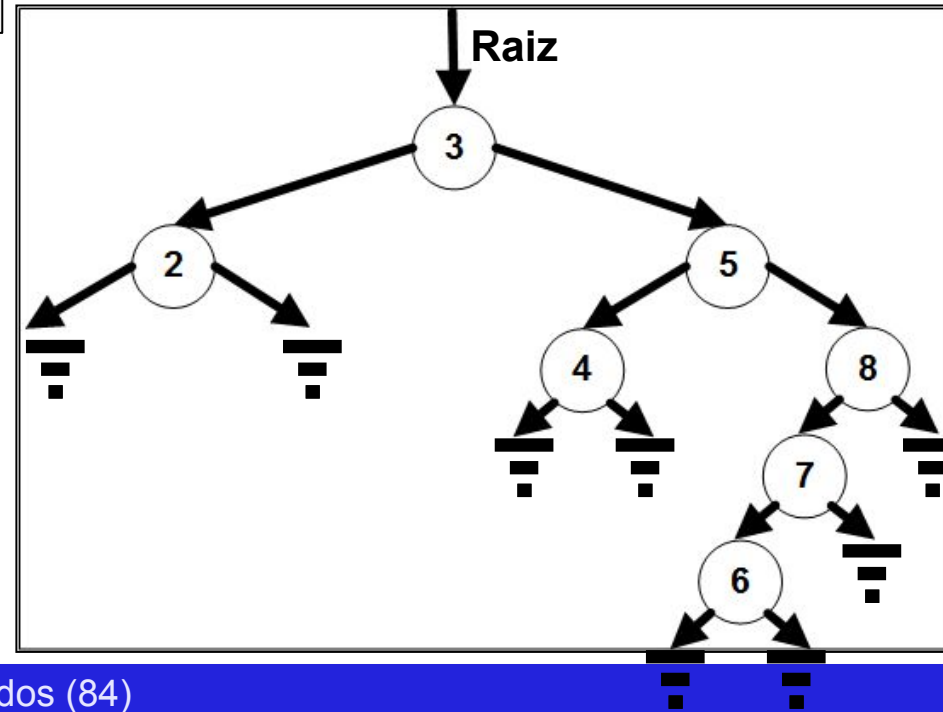
# Algoritmo de Remoção em C#

```
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() { }
}
```

raiz

n(3)

Voltando com o 1 antes  
de fazer outra remoção



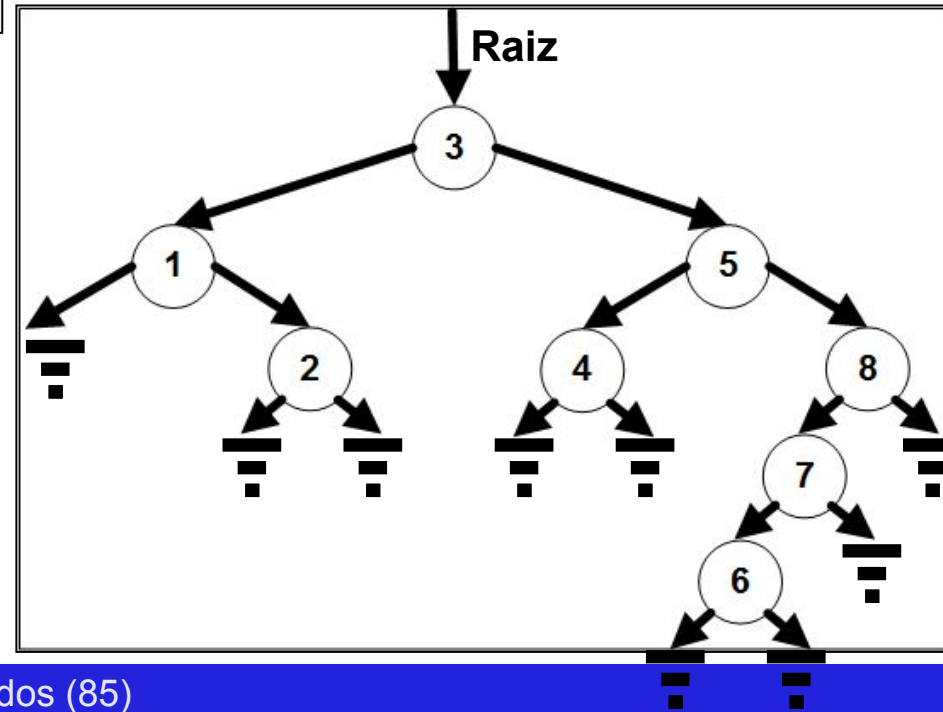
# Algoritmo de Remoção em C#

```
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() { }
}
```

raiz

n(3)

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



# Algoritmo de Remoção em C#

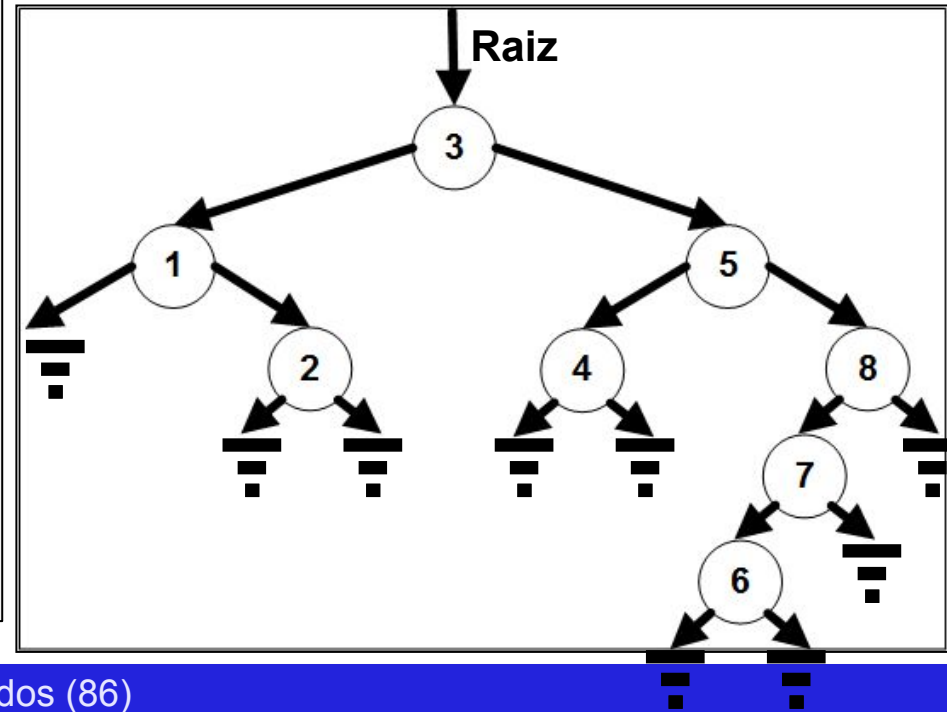
//remover(3), dois filhos

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

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

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

raiz    n(3)    x    3



# Algoritmo de Remoção em C#

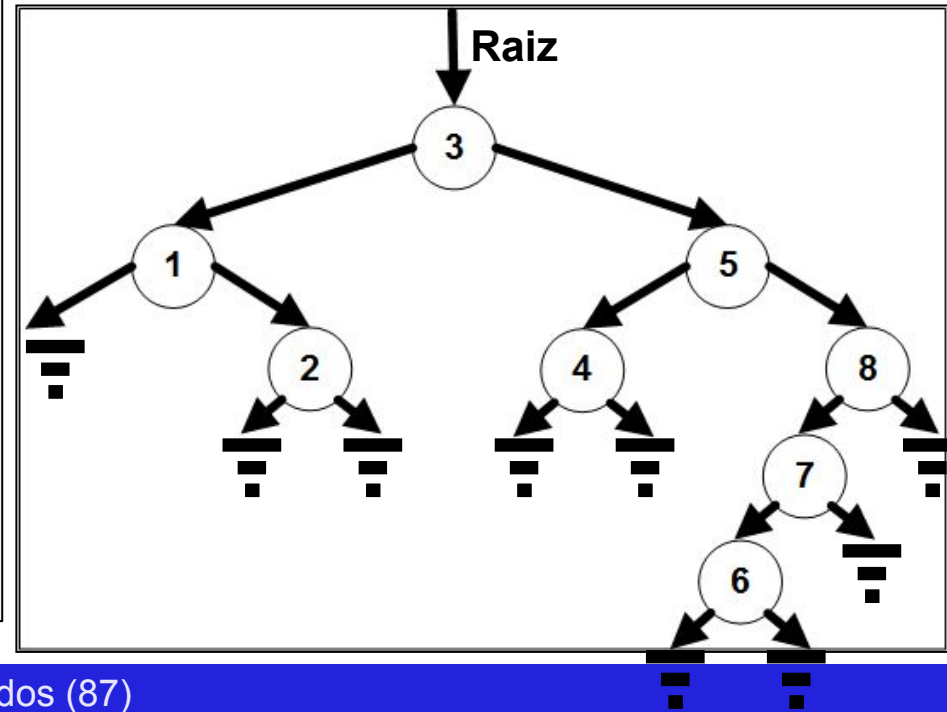
//remover(3), dois filhos

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

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

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

raiz    n(3)    x    3



# Algoritmo de Remoção em C#

//remover(3), dois filhos

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

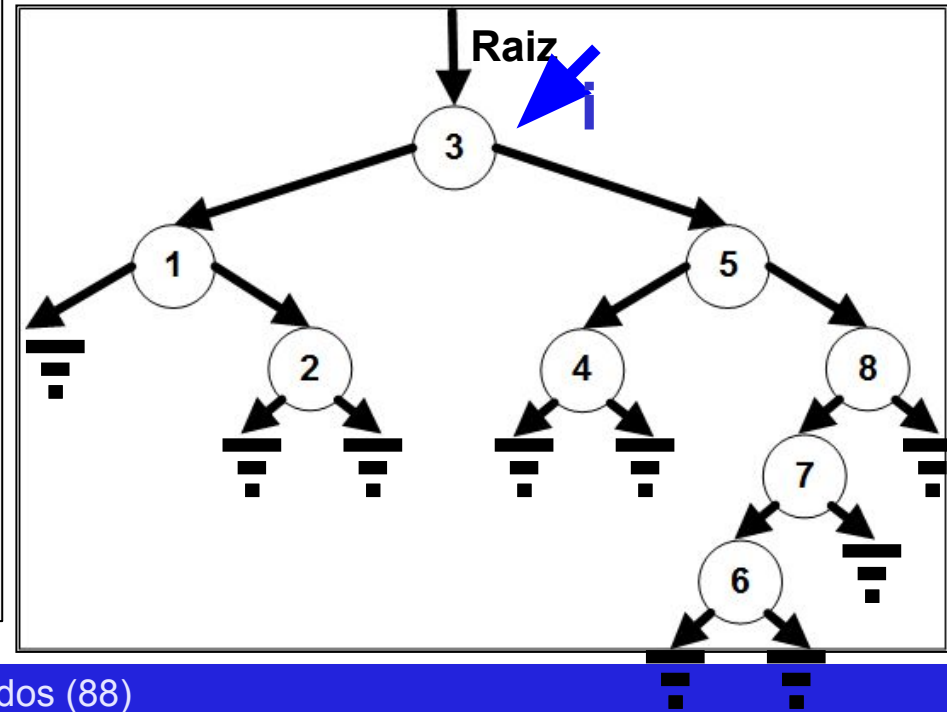
No remover(int x, No i) throws Exception {

```
    if (i == null) { throw new Exception("Erro!");
    } else if (x < i.elemento) { i.esq = remover(x, i.esq);
    } else if (x > i.elemento) { i.dir = remover(x, i.dir);
    } else if (i.dir == null) { i = i.esq;
    } else if (i.esq == null) { i = i.dir;
    } else {
        i.esq = maiorEsq(i, i.esq);
    }
    return i;
}
```

No maiorEsq(No i, No j) {

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

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





# Algoritmo de Remoção em C#

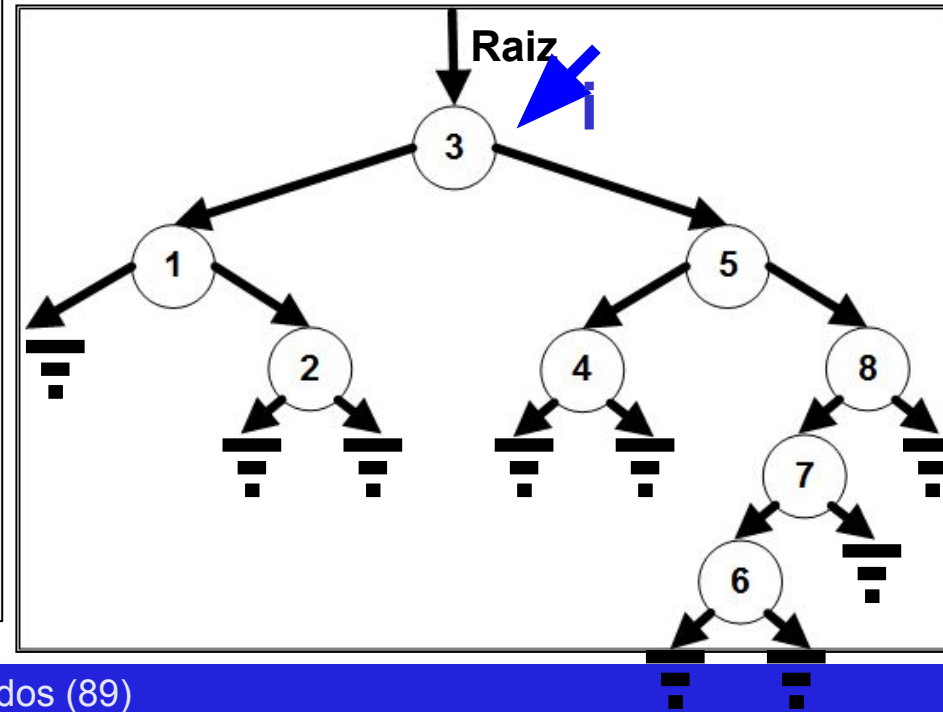
//remover(3), dois filhos

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

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

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

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



# Algoritmo de Remoção em C#

//remover(3), dois filhos

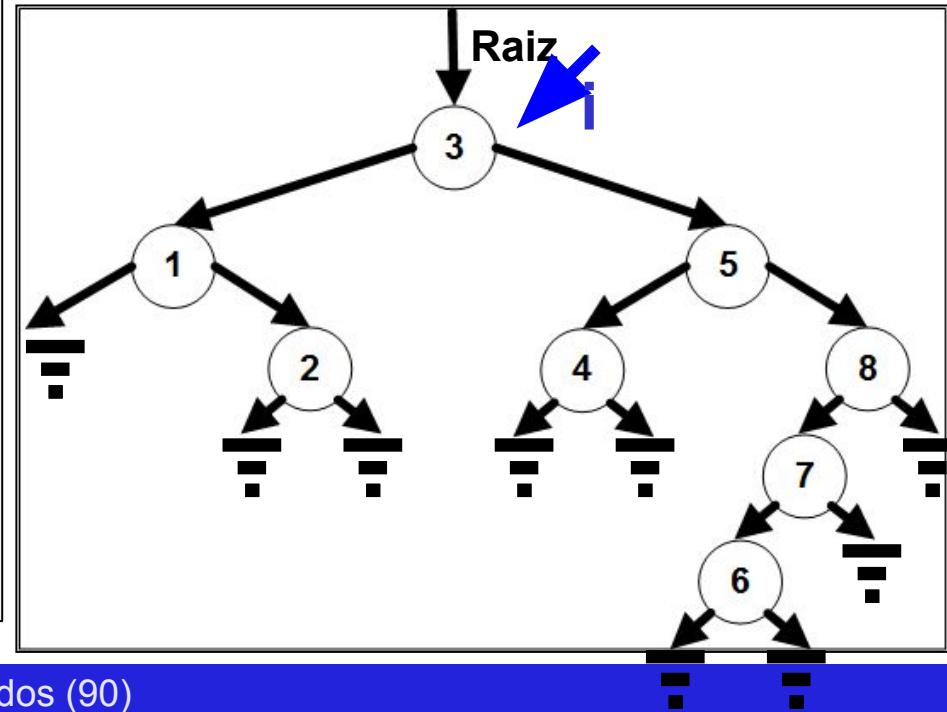
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

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

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

false: 3 < 3

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



# Algoritmo de Remoção em C#

//remover(3), dois filhos

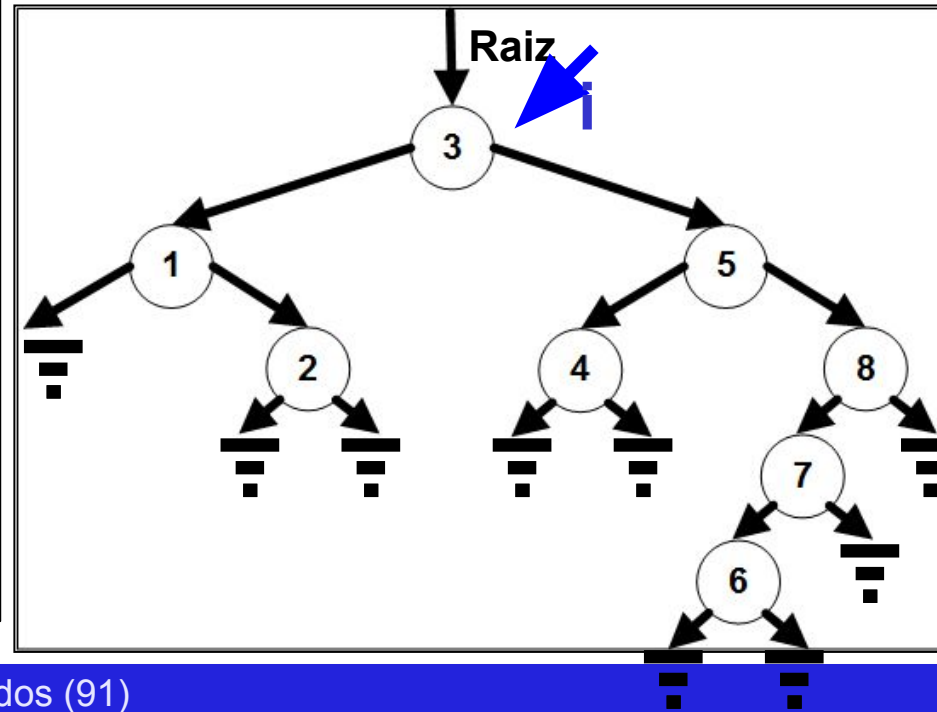
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

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

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

false: 3 > 3

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



# Algoritmo de Remoção em C#

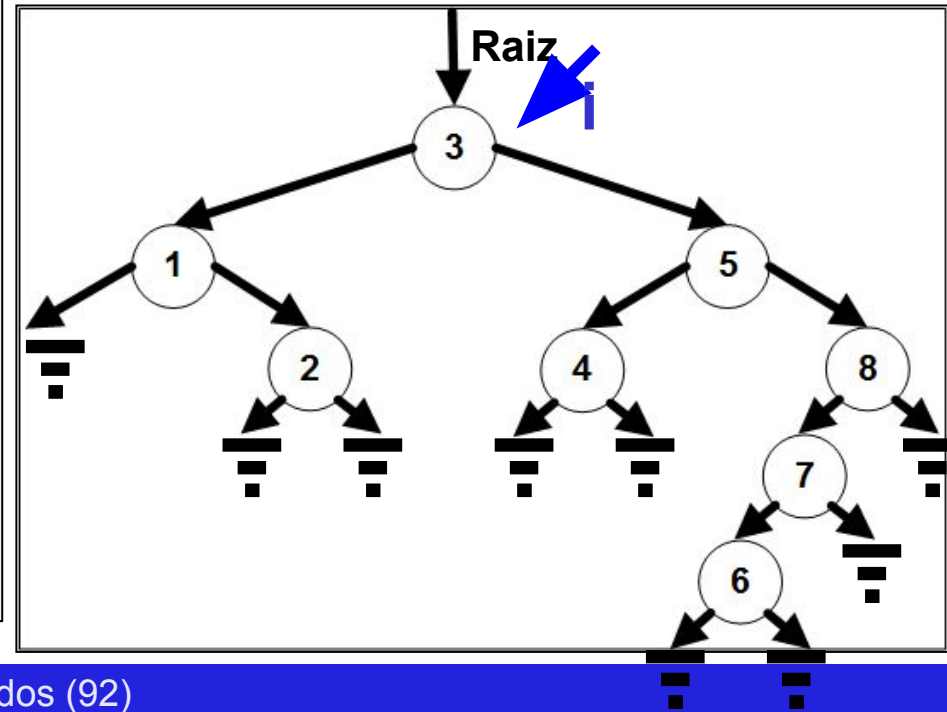
//remover(3), dois filhos

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

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

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

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



# Algoritmo de Remoção em C#

//remover(3), dois filhos

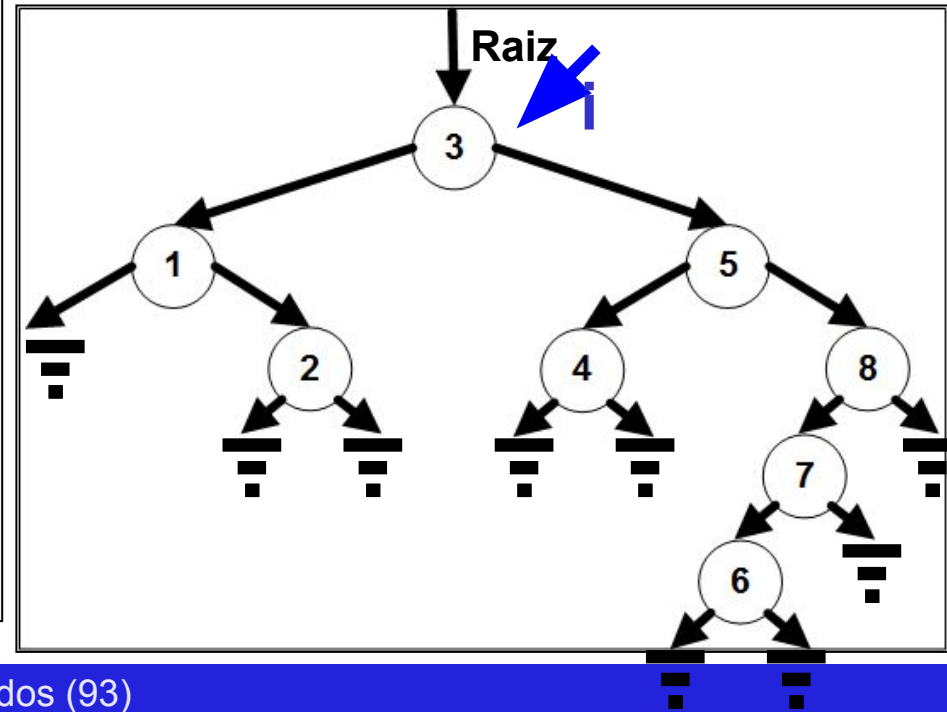
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}

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

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

false: n(1) == false

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



# Algoritmo de Remoção em C#

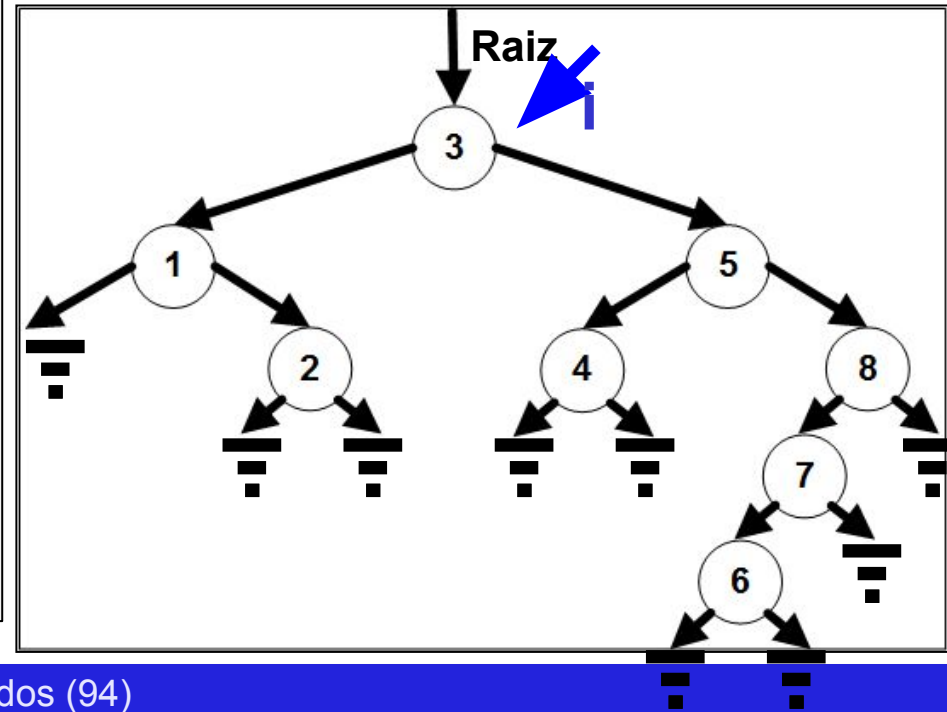
//remover(3), dois filhos

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

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

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

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



# Algoritmo de Remoção em C#

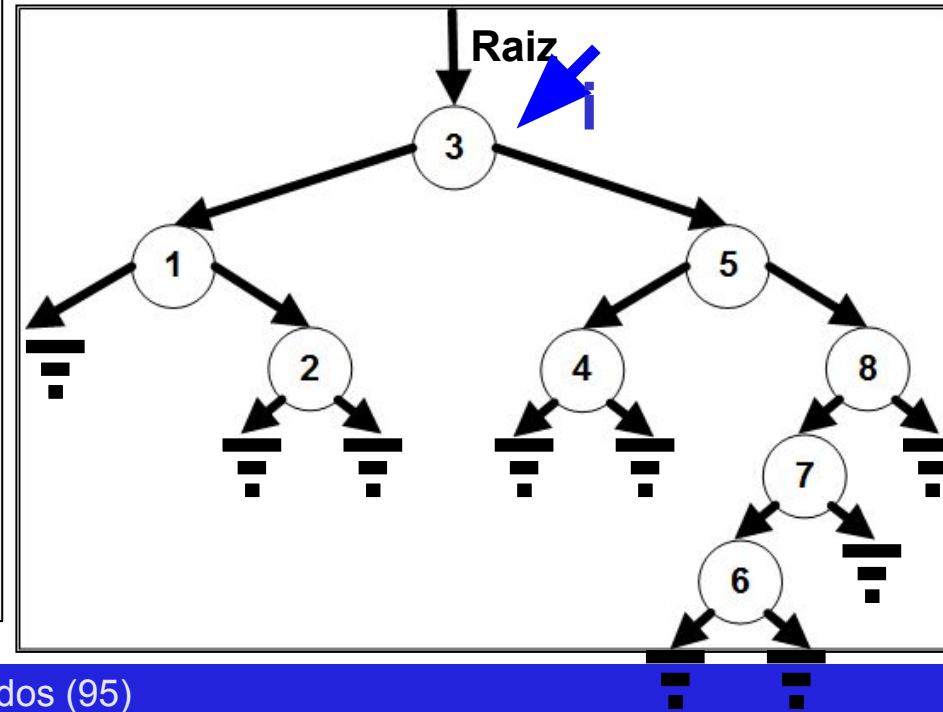
//remover(3), dois filhos

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

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

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

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



# Algoritmo de Remoção em C#

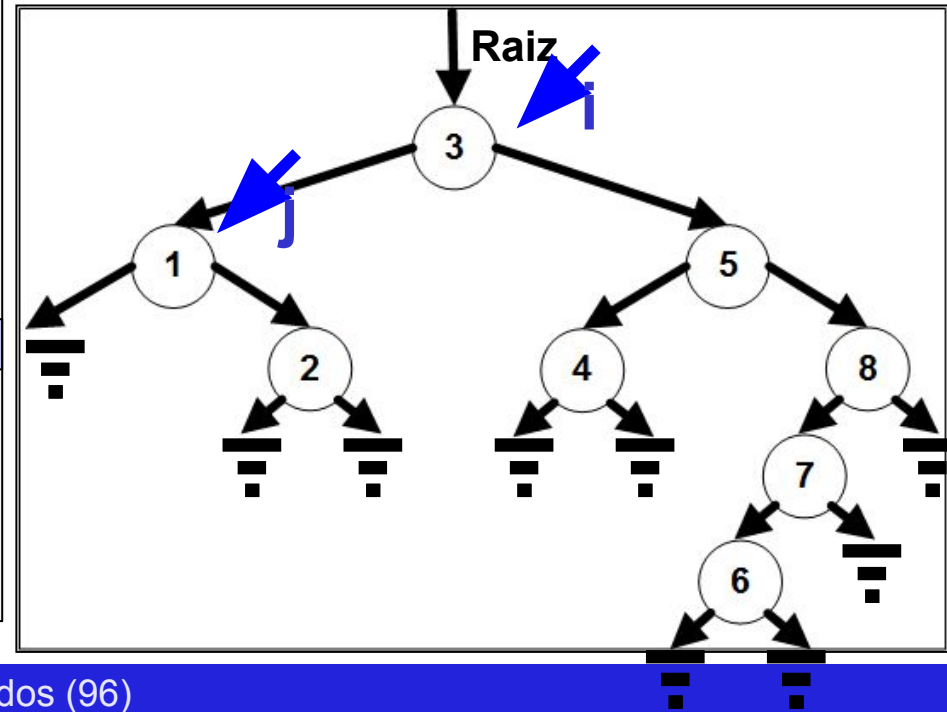
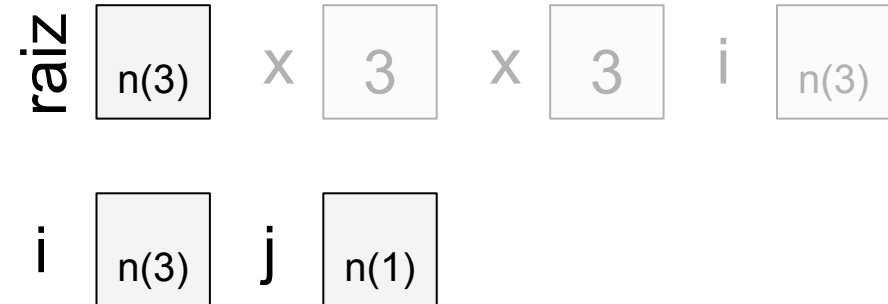
//remover(3), dois filhos

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

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

No maiorEsq(No i, No j) {

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





# Algoritmo de Remoção em C#

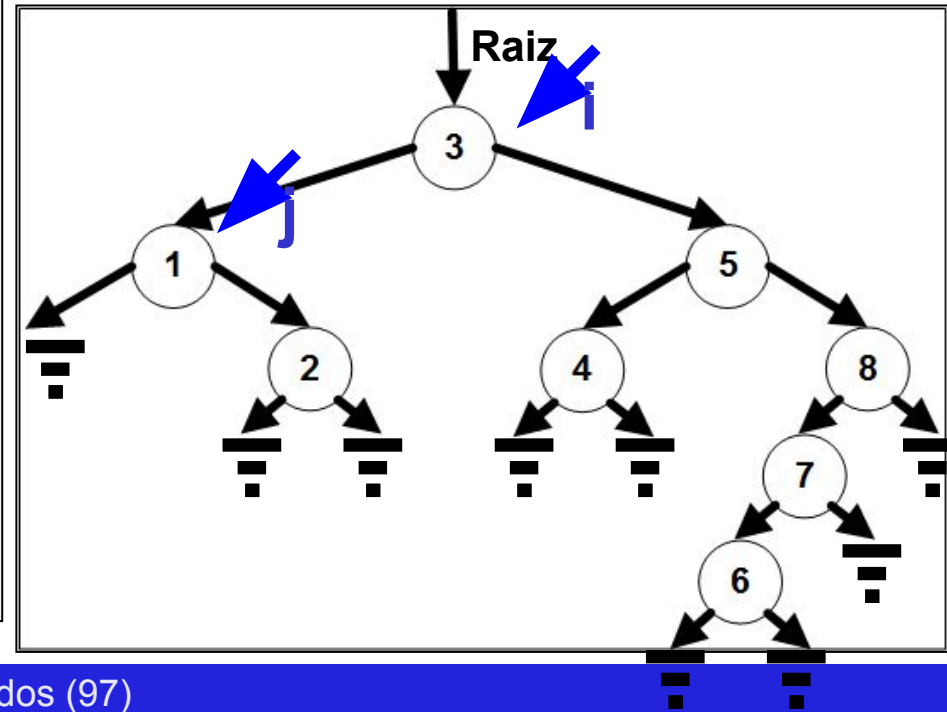
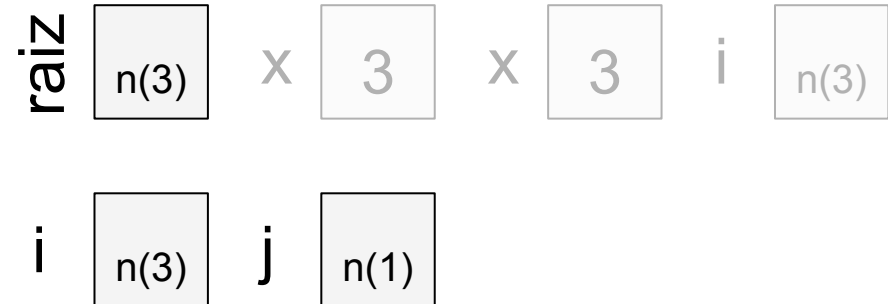
//remover(3), dois filhos

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

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

false: n(2) == null
```

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



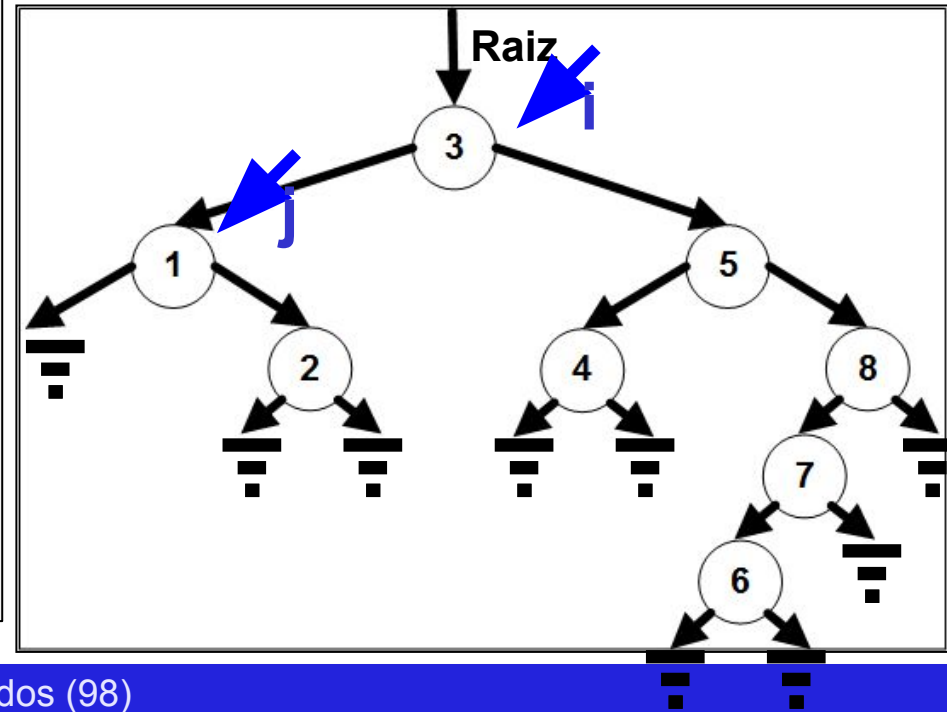
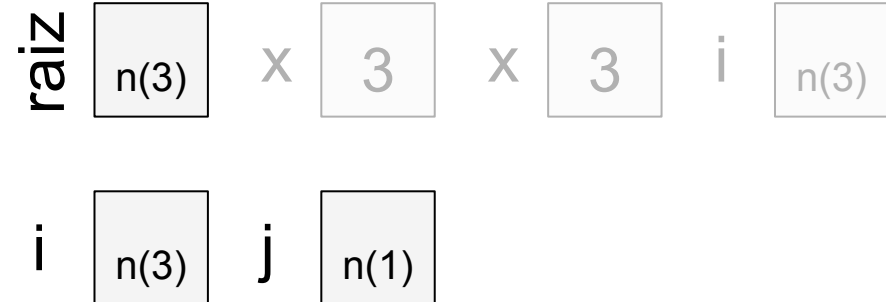
# Algoritmo de Remoção em C#

//remover(3), dois filhos

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

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

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



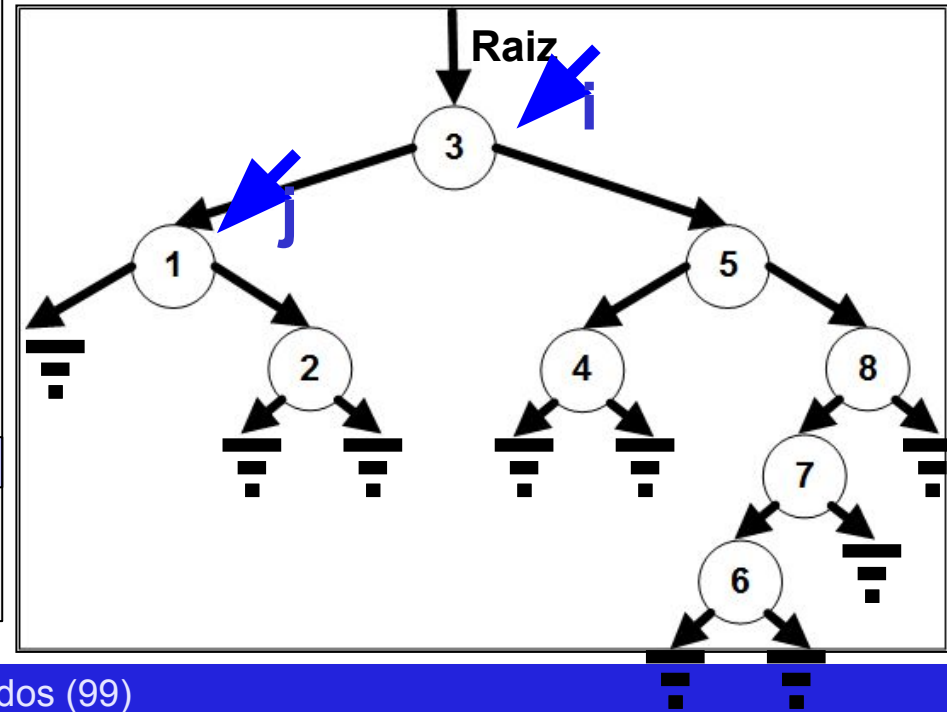
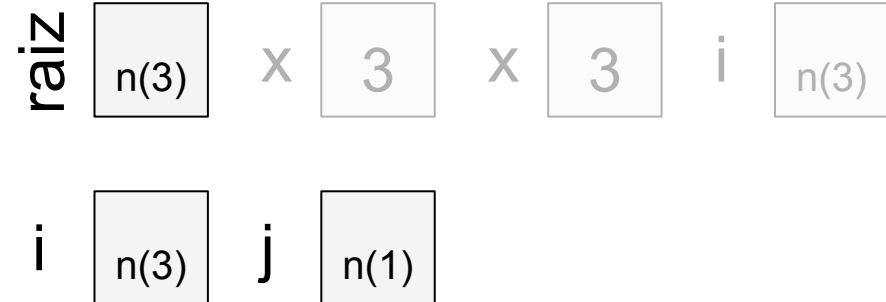
# Algoritmo de Remoção em C#

//remover(3), dois filhos

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

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

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



# Algoritmo de Remoção em C#

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

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

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

```
No maiorEsq(No i, No j) {
```

```

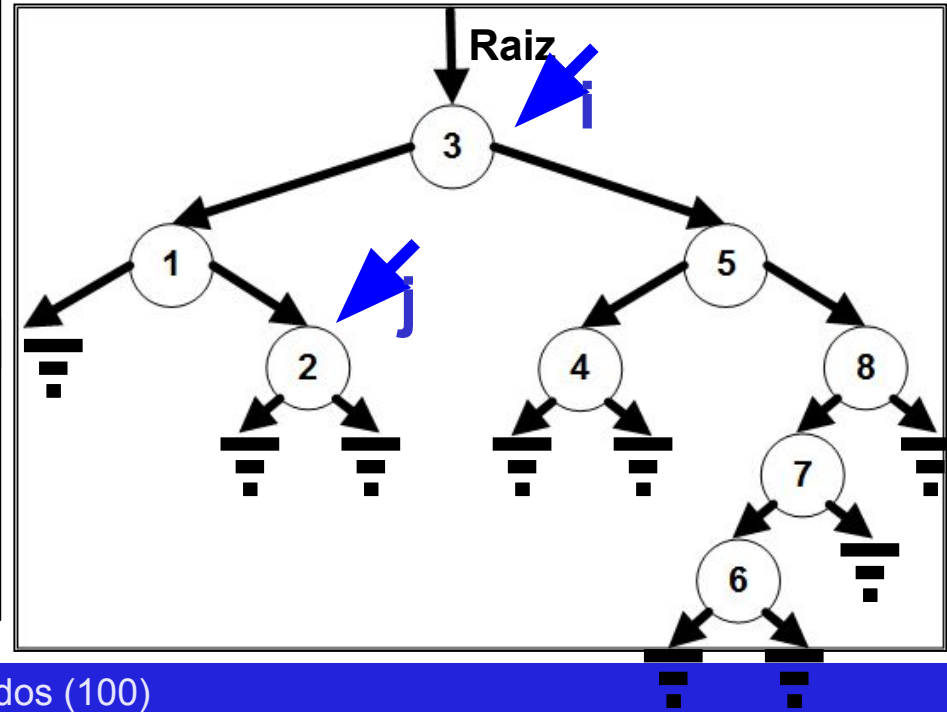
if (j.dir == null){ i.elemento=j.elemento; j=j.esq; }
else {           j.dir = maiorEsq(i, j.dir);           }
return j;
}

```

raiz

$n(3)$   $\times$   $3$   $\times$   $3$   $i$   $n(3)$

$i$   $n(3)$   $j$   $n(1)$   $i$   $n(3)$   $j$   $n(2)$



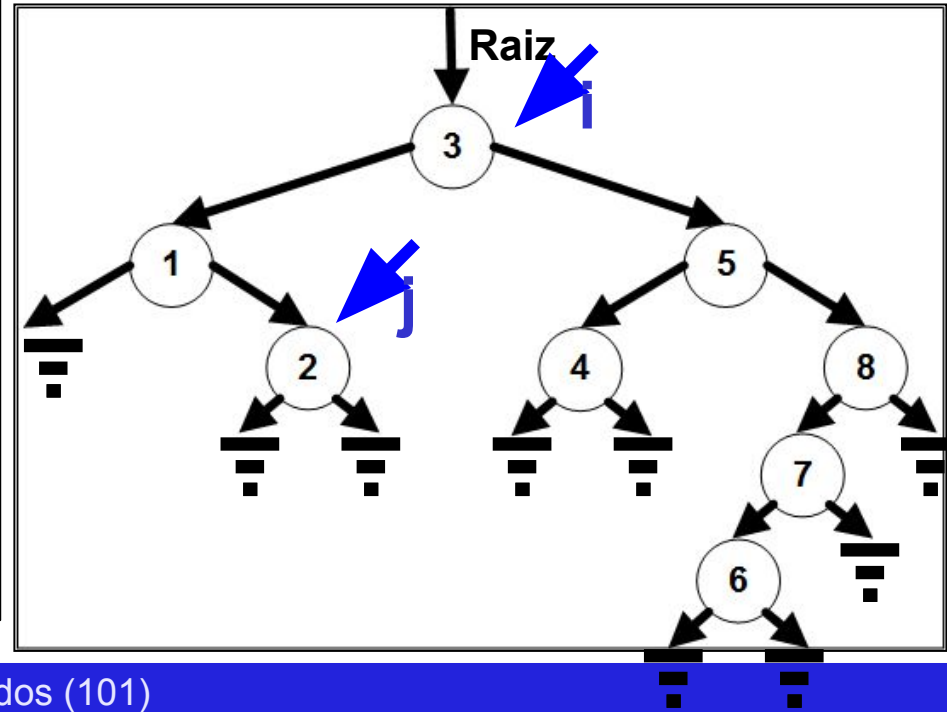
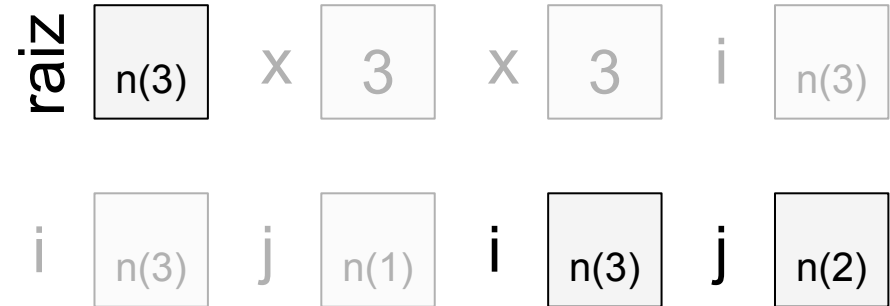
# Algoritmo de Remoção em C#

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

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

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

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



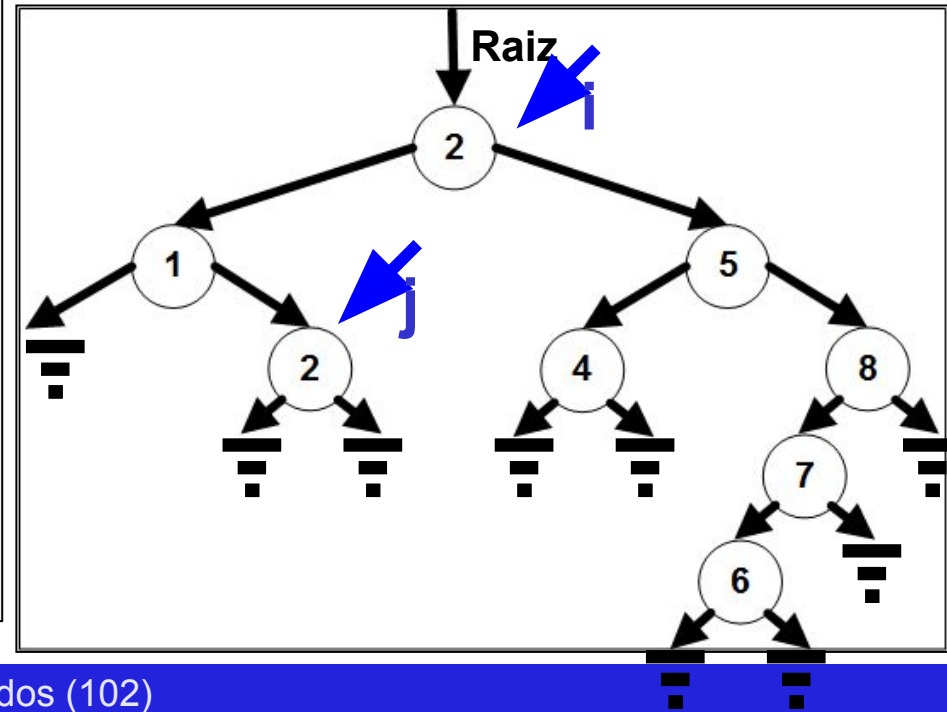
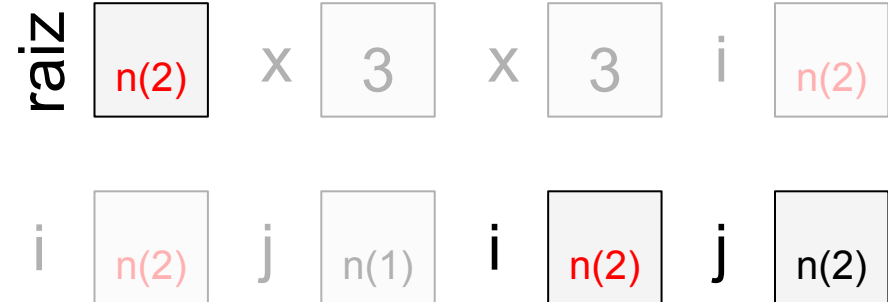
# Algoritmo de Remoção em C#

//remover(3), dois filhos

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

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

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



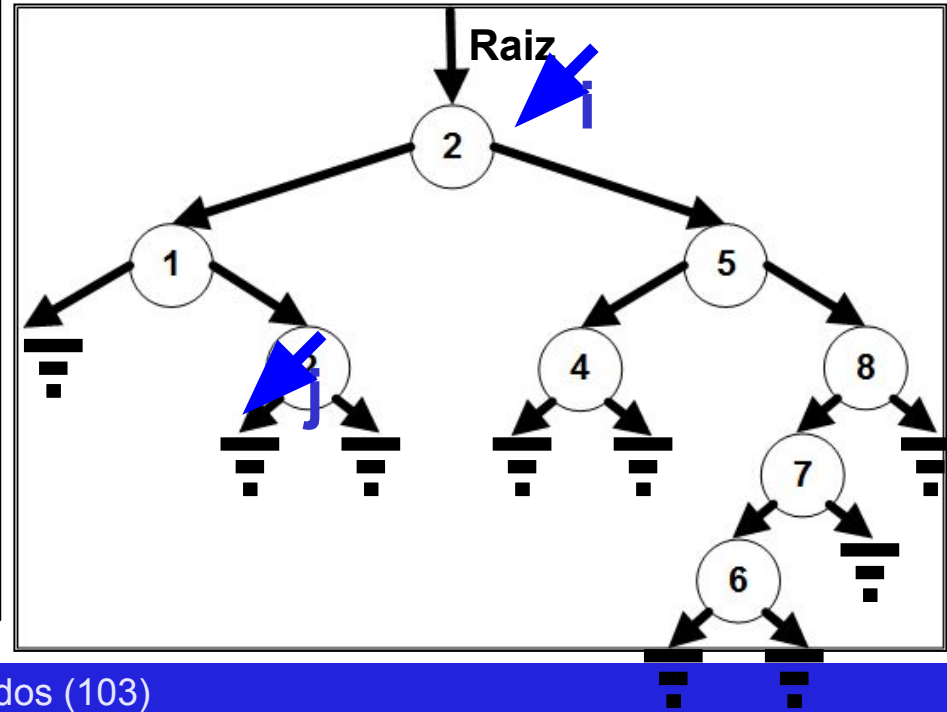
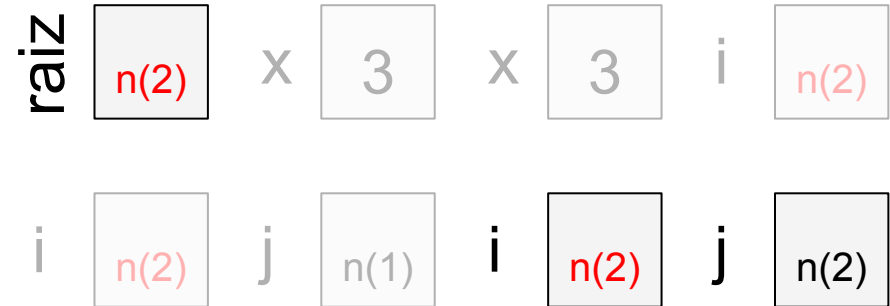
# Algoritmo de Remoção em C#

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

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

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

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



# Algoritmo de Remoção em C#

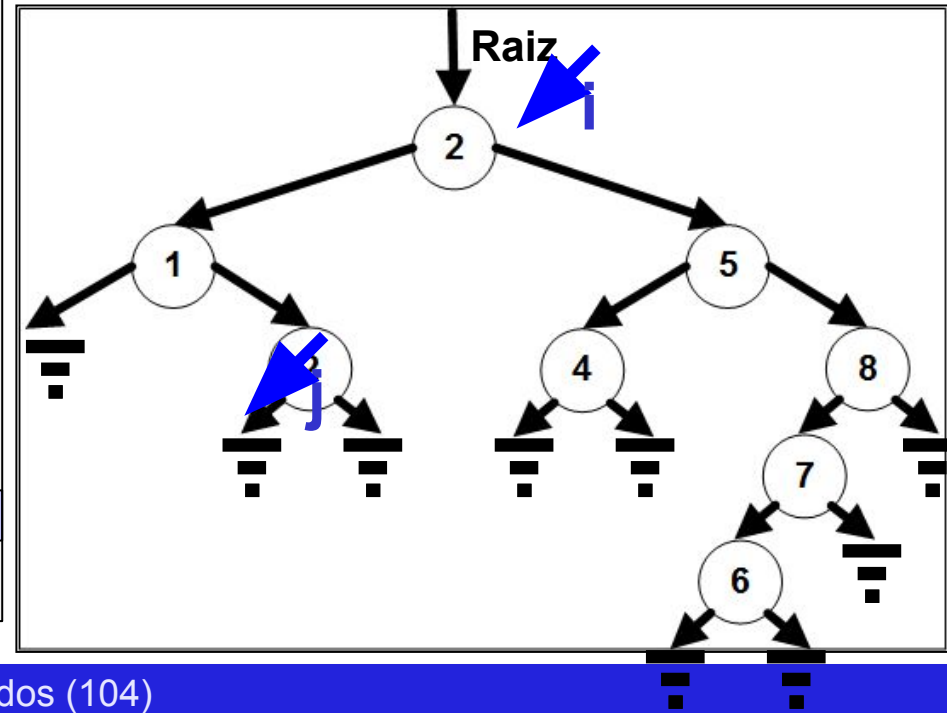
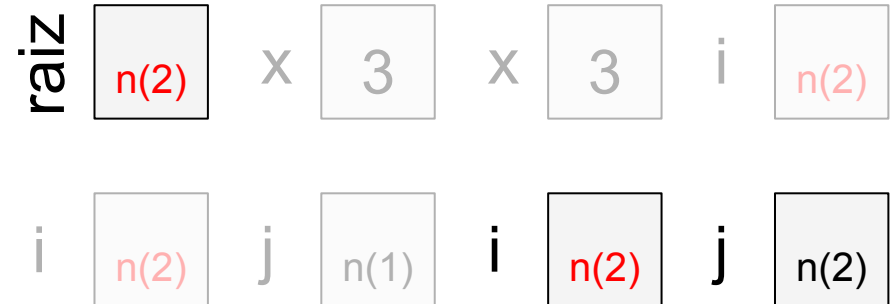
//remover(3), dois filhos

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

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

Retornando null

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





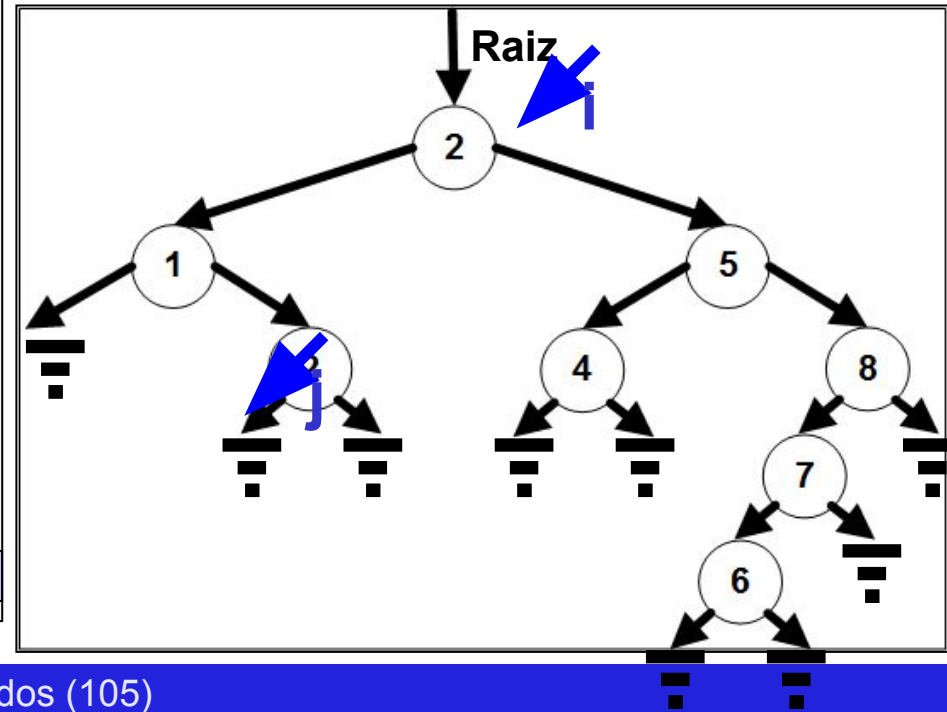
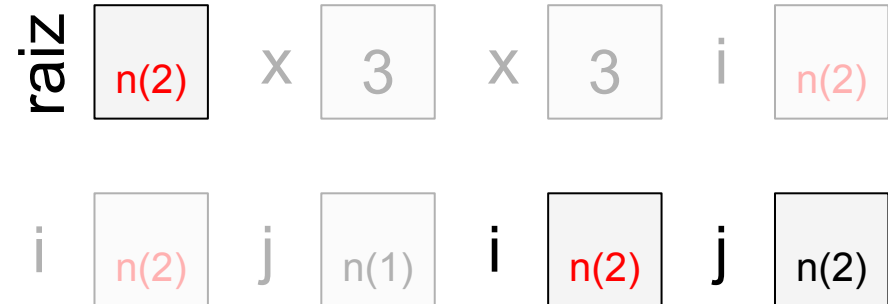
# Algoritmo de Remoção em C#

//remover(3), dois filhos

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

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

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



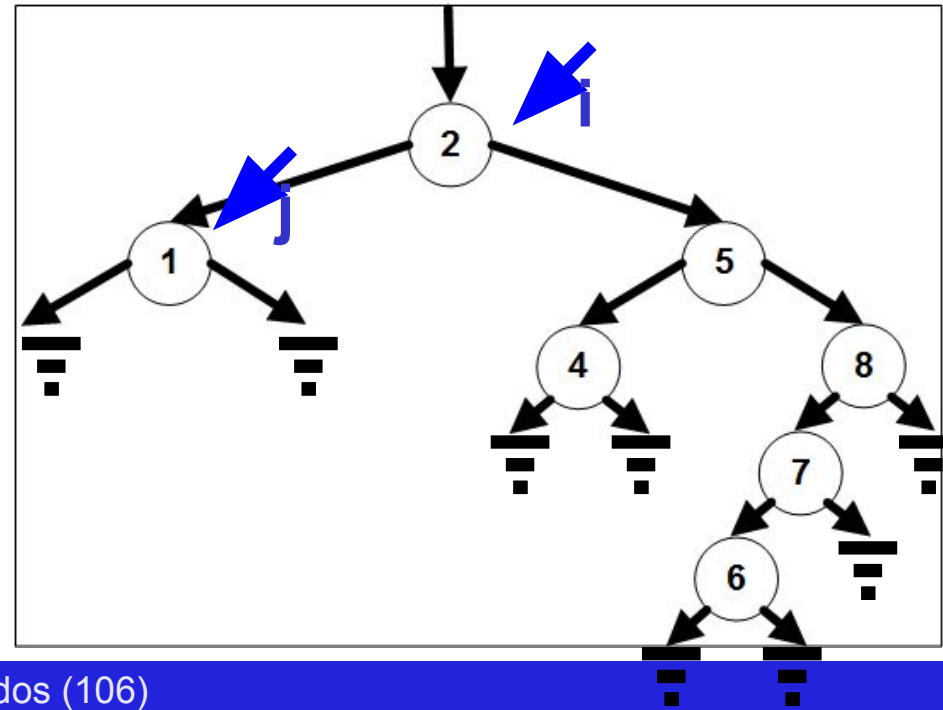
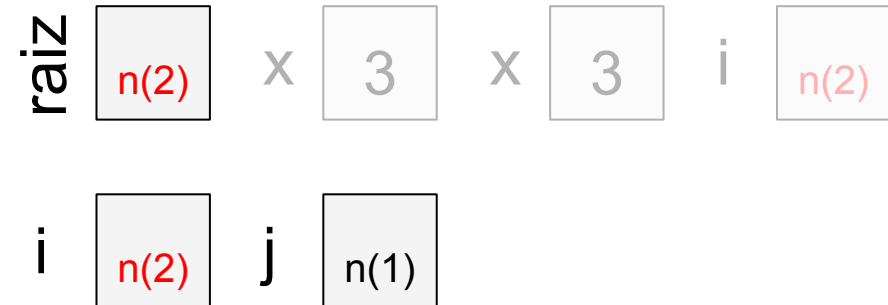
# Algoritmo de Remoção em C#

//remover(3), dois filhos

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

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

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



# Algoritmo de Remoção em C#

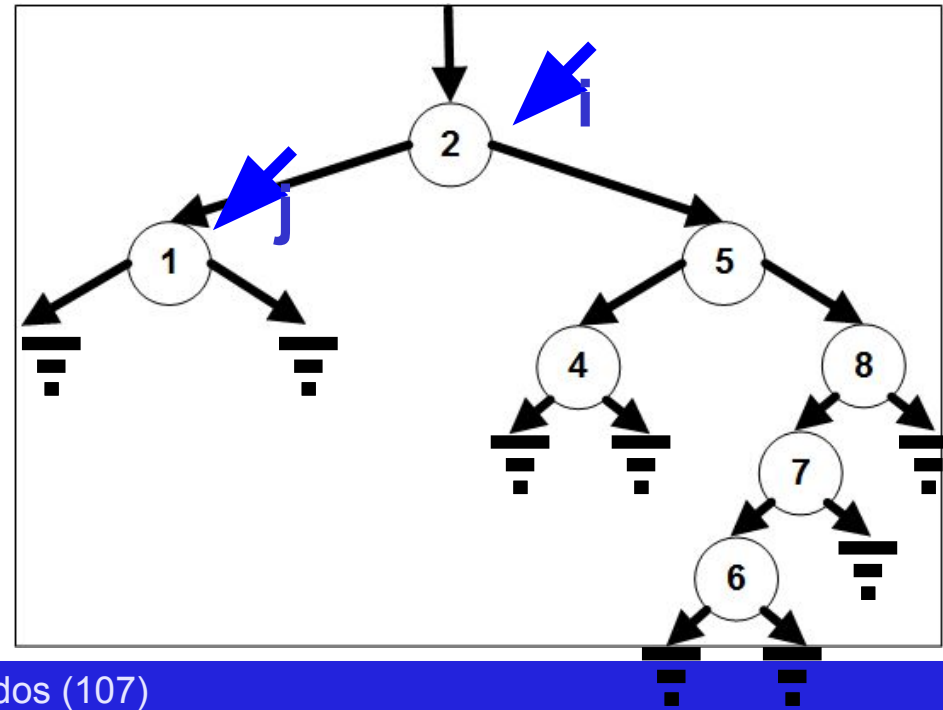
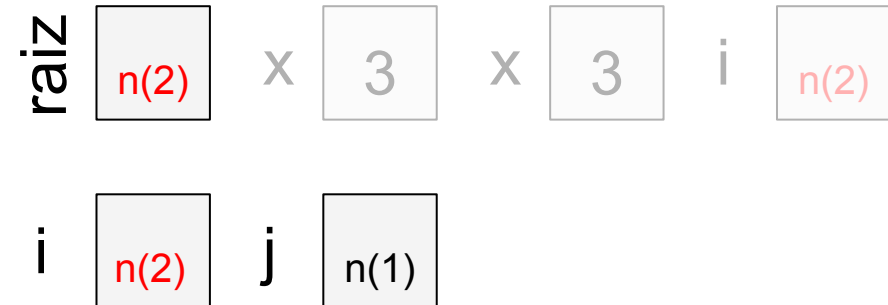
//remover(3), dois filhos

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

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

Retornando n(1)

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



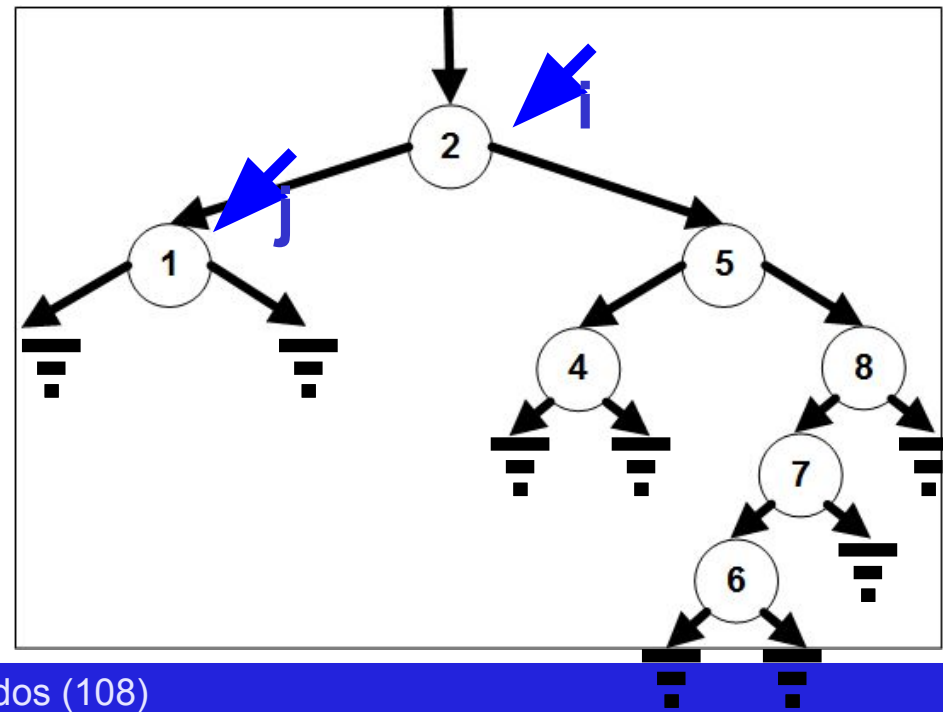
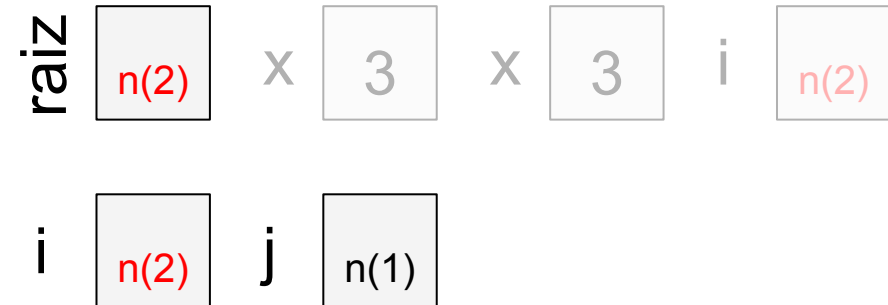
# Algoritmo de Remoção em C#

//remover(3), dois filhos

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

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

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



# Algoritmo de Remoção em C#

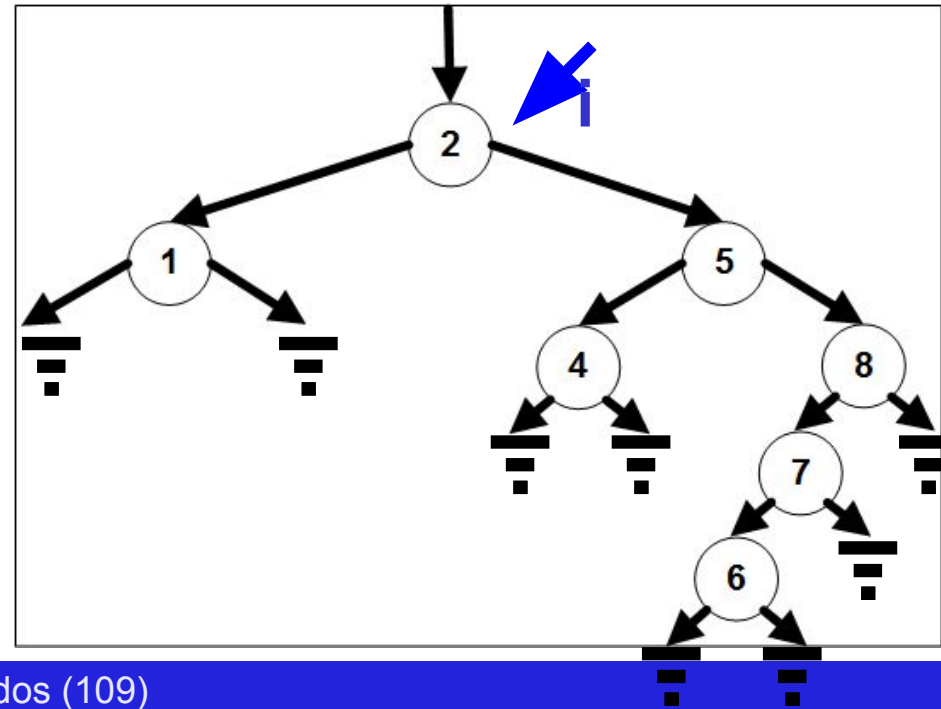
//remover(3), dois filhos

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

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

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

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



# Algoritmo de Remoção em C#

//remover(3), dois filhos

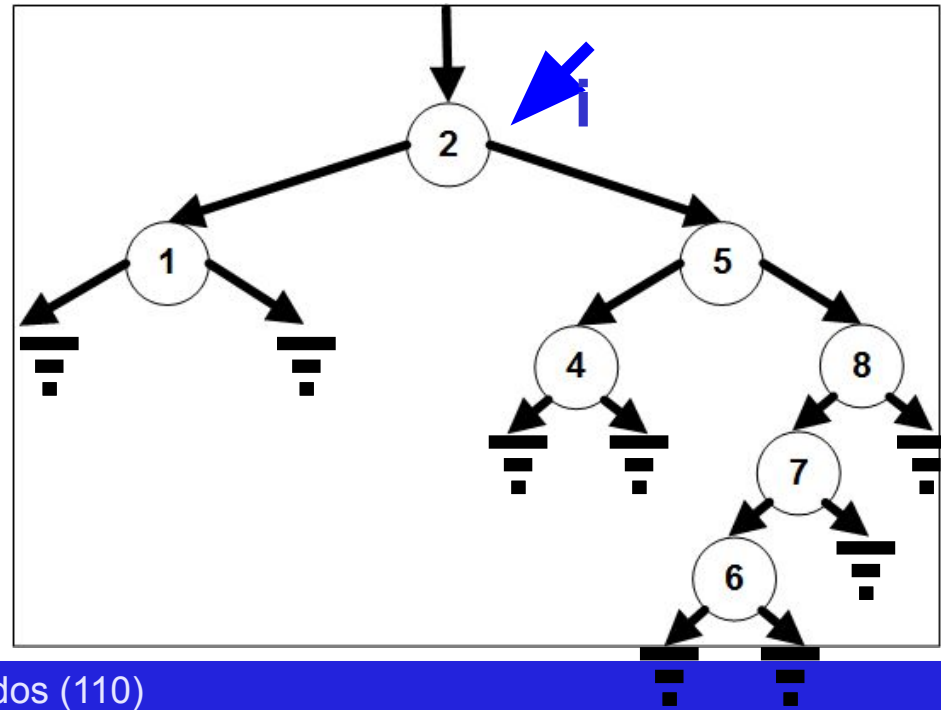
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```

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

Retorna n(2)

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

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



# Algoritmo de Remoção em C#

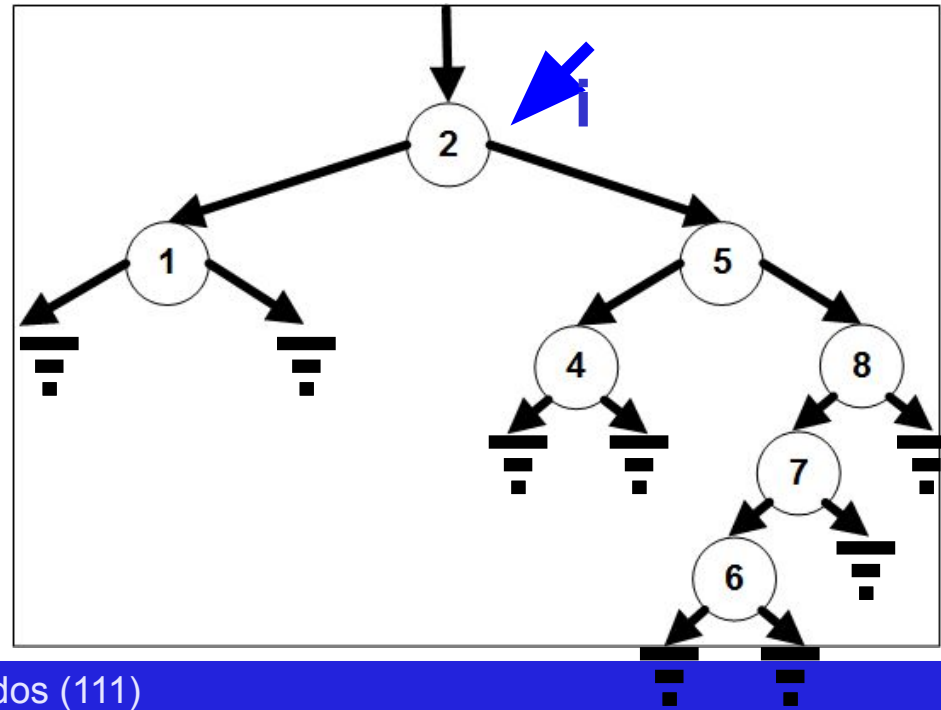
//remover(3), dois filhos

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

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

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

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



# Algoritmo de Remoção em C#

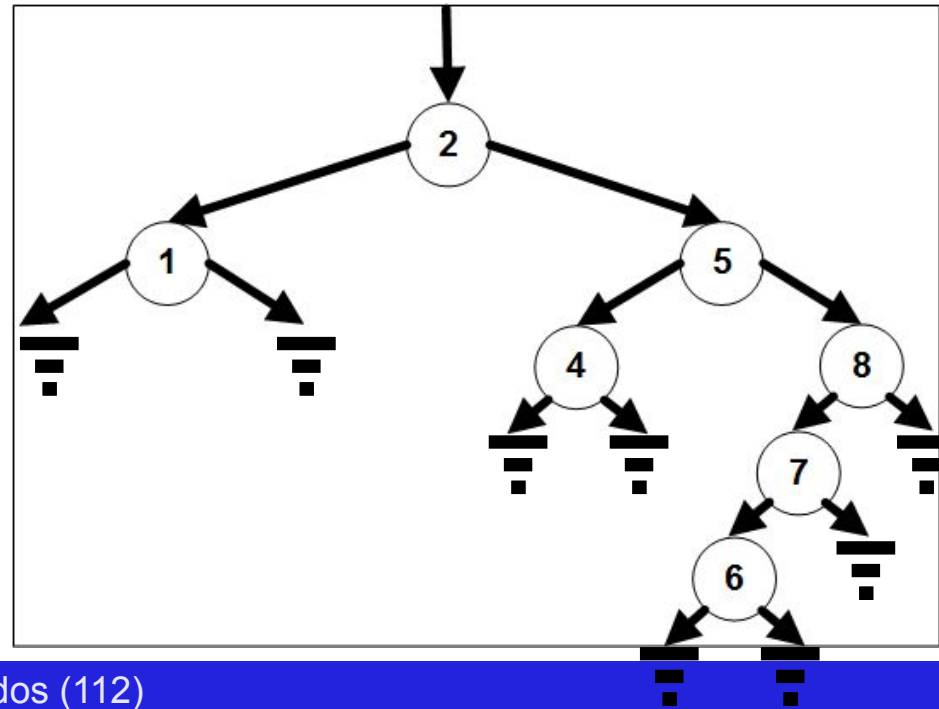
//remover(3), dois filhos

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

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

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

raiz n(2) x 3





# Algoritmo de Remoção em C#

//remover(3), dois filhos

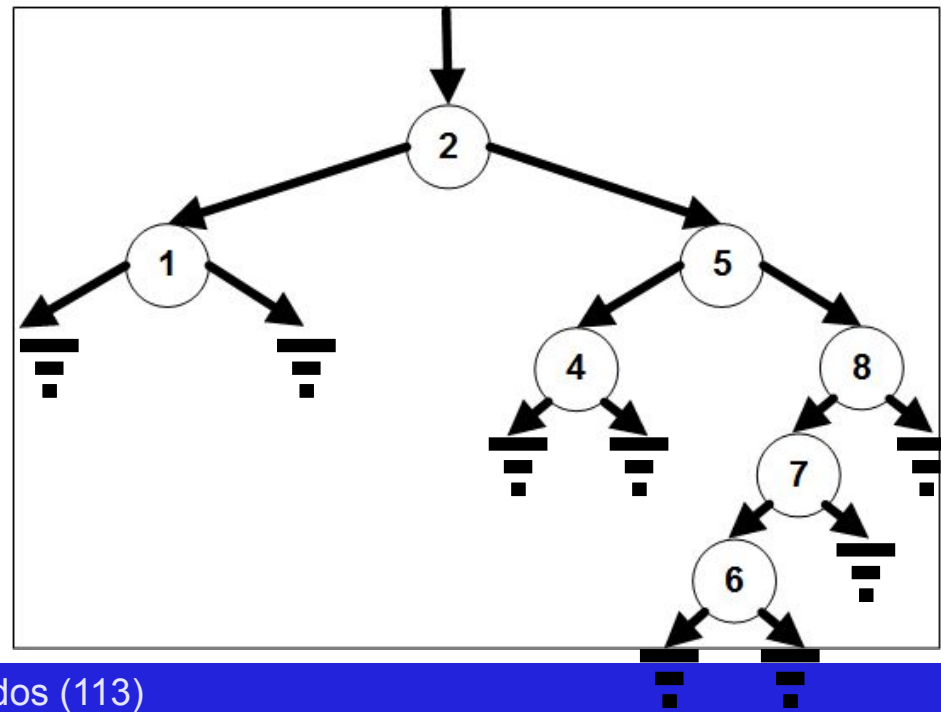
```
void remover(int x) throws Exception {
    raiz = remover(x, raiz);
}
```


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

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

raiz

n(2)



- Funcionamento básico
- Algoritmo em C#
- **Análise de complexidade** 

# Análise de complexidade da Remoção

- **Melhor Caso:**  $\Theta(1)$  comparações e acontece, por exemplo, na raiz
- **Pior Caso:**  $\Theta(n)$  comparações e acontece, por exemplo, quando inserimos os elementos em ordem e o elemento desejado remover na folha
- **Caso Médio:**  $\Theta(\lg(n))$  comparações e acontece, por exemplo, quando a árvore está balanceada e desejamos remover um elemento localizado em uma das folhas

## Exercício Resolvido (3)

- O método ***remove*** 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 Resolvido (3)

- 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**.

```
public void remover2(int x) throws Exception {
    if (raiz == null) {
        throw new Exception("Erro ao remover2!");
    } else if (x < raiz.elemento) {
        remover2(x, raiz.esq, raiz);
    } else if (x > raiz.elemento) {
        remover2(x, raiz.dir, raiz);
    } else if (raiz.dir == null) {
        raiz = raiz.esq;
    } else if (raiz.esq == null) {
        raiz = raiz.dir;
    } else {
        raiz.esq = antecessor(raiz, raiz.esq);
    }
}
```

```
private void remover2(int x, No i, No pai) throws Exception {
    if (i == null) {
        throw new Exception("Erro ao remover2!");
    } else if (x < i.elemento) {
        remover2(x, i.esq, i);
    } else if (x > i.elemento) {
        remover2(x, i.dir, i);
    } else if (i.dir == null) {
        if (x < pai.elemento) {
            pai.esq = i.esq;
        } else {
            pai.dir = i.esq;
        }
    } else if (i.esq == null) {
        if (x < pai.elemento) {
            pai.esq = i.dir;
        } else {
            pai.dir = i.dir;
        }
    } else {
        i.esq = antecessor(i, i.esq);
    }
}
```

## Exercício (1)

- O método ***remove*** privado e recursivo apresentado em nossa árvore recebe um valor e retorna um No. No exercício anterior, o ***remove2*** retorna *void*. Implemente um método ***remove3*** que retorna o elemento removido.