

# ED62A-COM2A

# ESTRUTURAS DE DADOS

Aula 06b - Remoção em  
Árvores Binárias

Prof. Rafael G. Mantovani

14/10/2019

# Roteiro

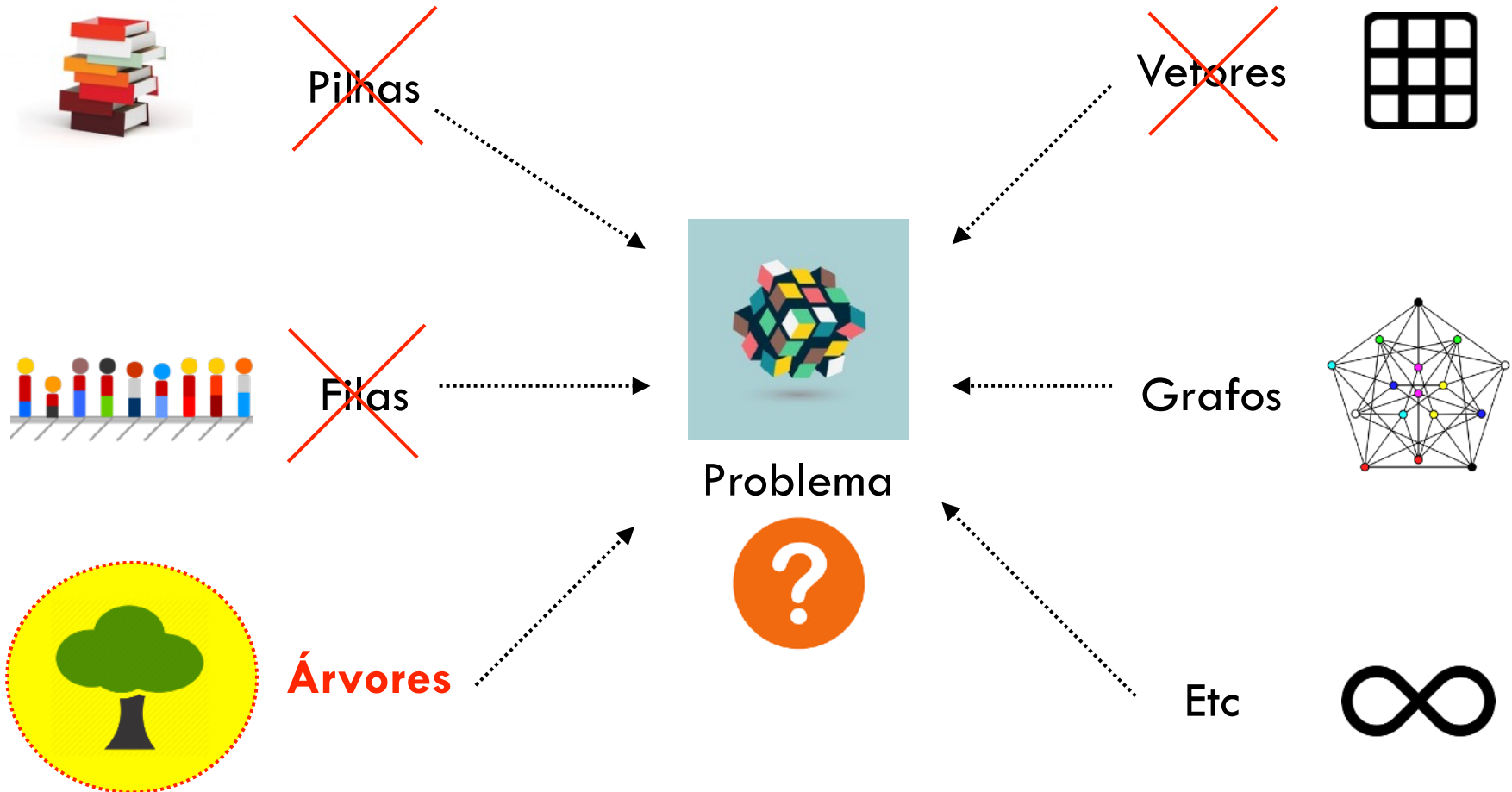


- 1** Tópicos já vistos anteriormente
- 2** Remoção em Árvore Binárias
- 3** Próximos conteúdos
- 4** Referências

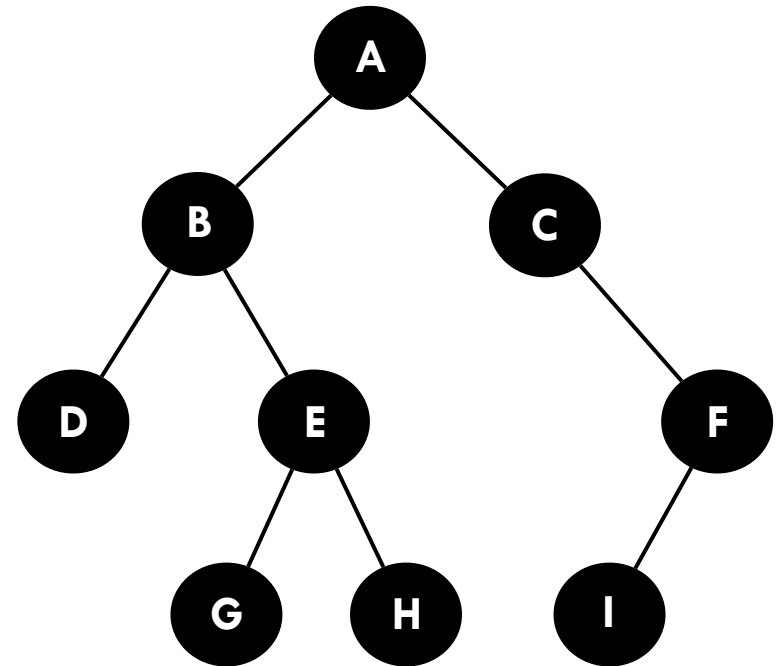
# Roteiro

- 1** Tópicos já vistos anteriormente
- 2** Remoção em Árvore Binárias
- 3** Próximos conteúdos
- 4** Referências

# Introdução

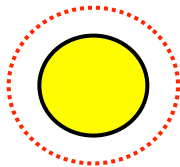


# Introdução



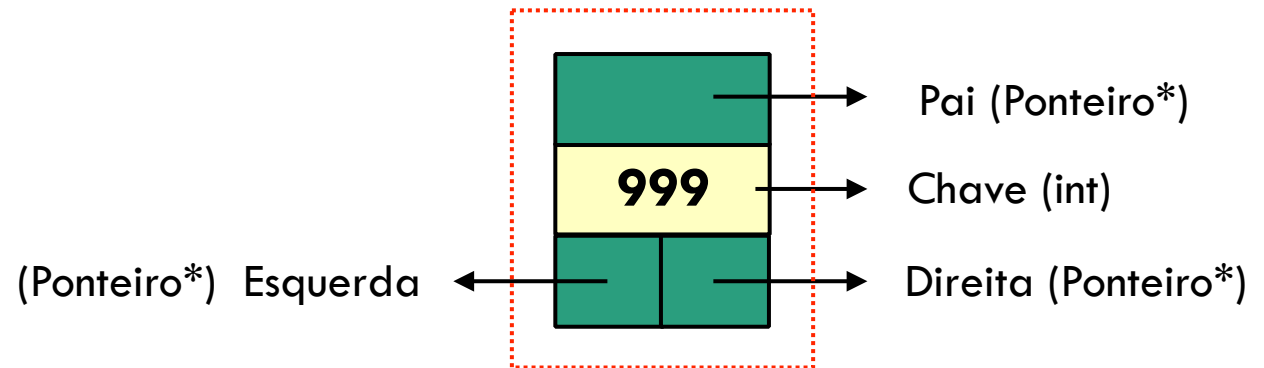
# Árvore Binária

NoArvore



Abstração

Struct



Tipo Abstrato  
de Dados

# Tipos de Árvore Binária

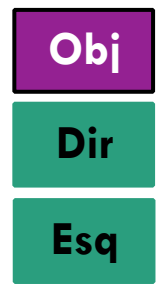
```
typedef struct {  
    int chave;  
} Objeto;  
  
typedef struct NoArvore *Ponteiro;  
  
typedef struct NoArvore{  
    Objeto obj;  
    Ponteiro direita;  
    Ponteiro esquerda;  
} NoArvore;  
  
/* Definir arvore binária na main */  
  
Ponteiro raiz; /* como definir e  
                usar árvore */
```

**Raiz**

**\*NoArvore**

**Arvore Binária**

tipo **NoArvore**



**Item armazenado**

**\*NoArvore (direita)**

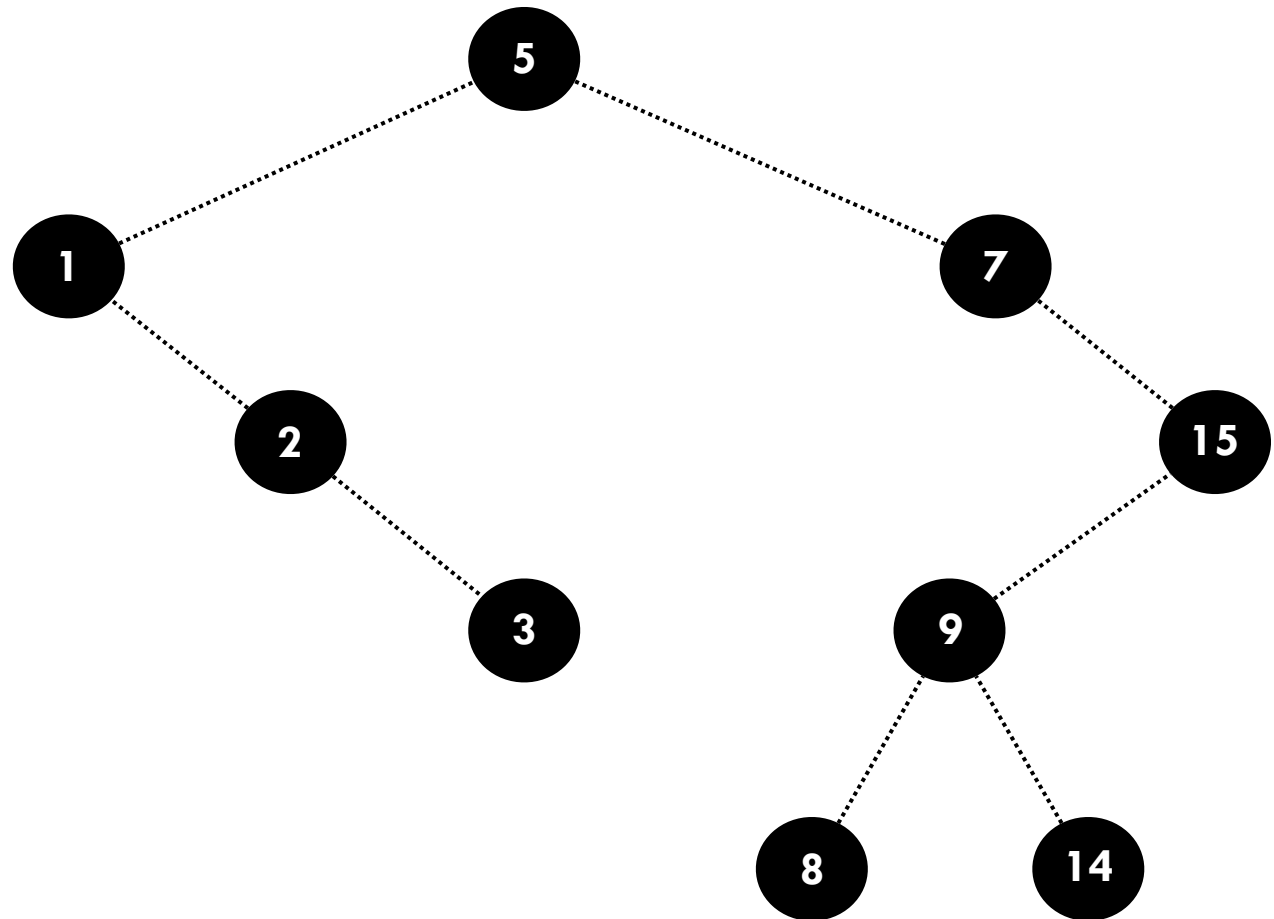
**\*NoArvore (esquerda)**

# Roteiro

- 1 Tópicos já vistos anteriormente
- 2 Remoção em Árvore Binárias
- 3 Próximos conteúdos
- 4 Referências

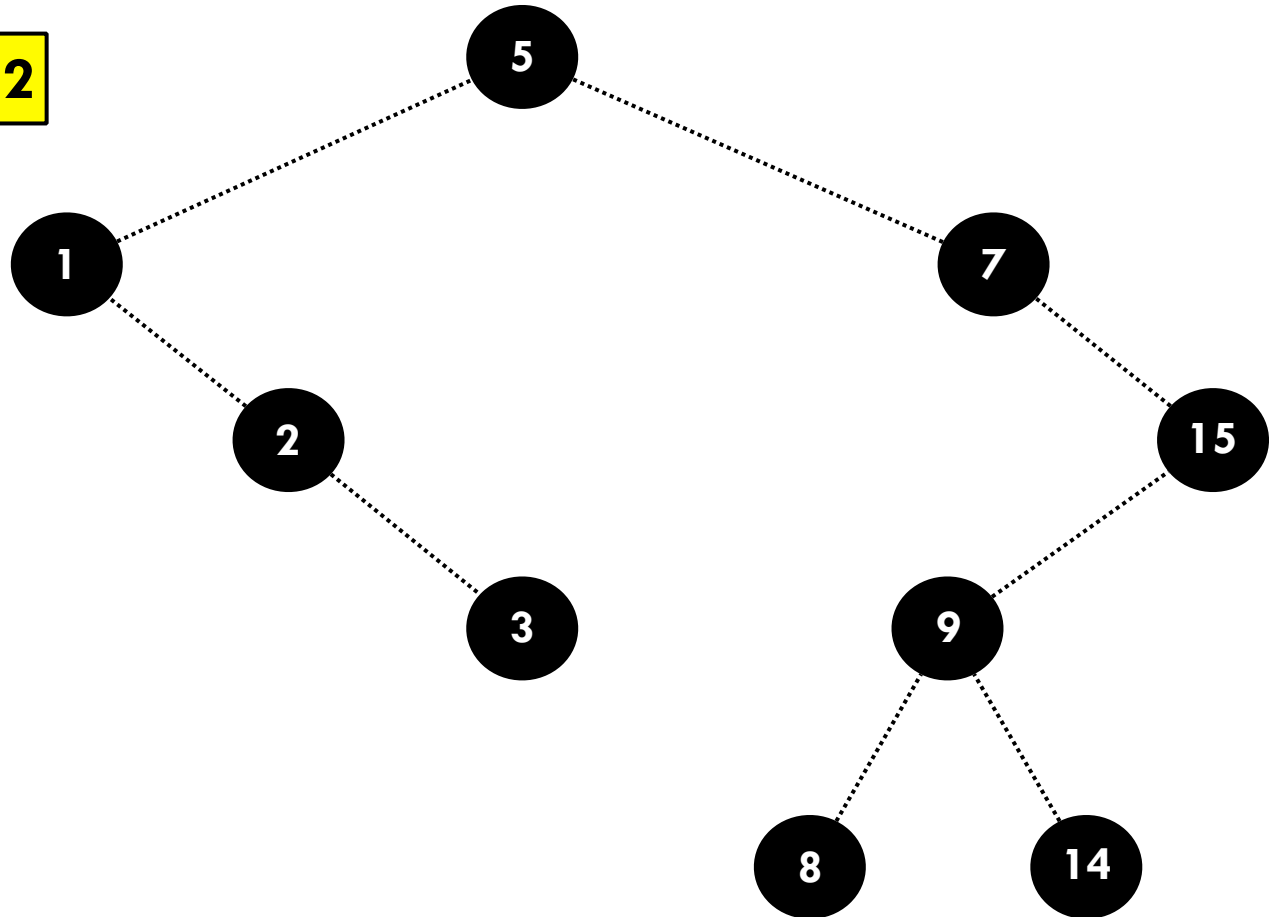


# Remoção



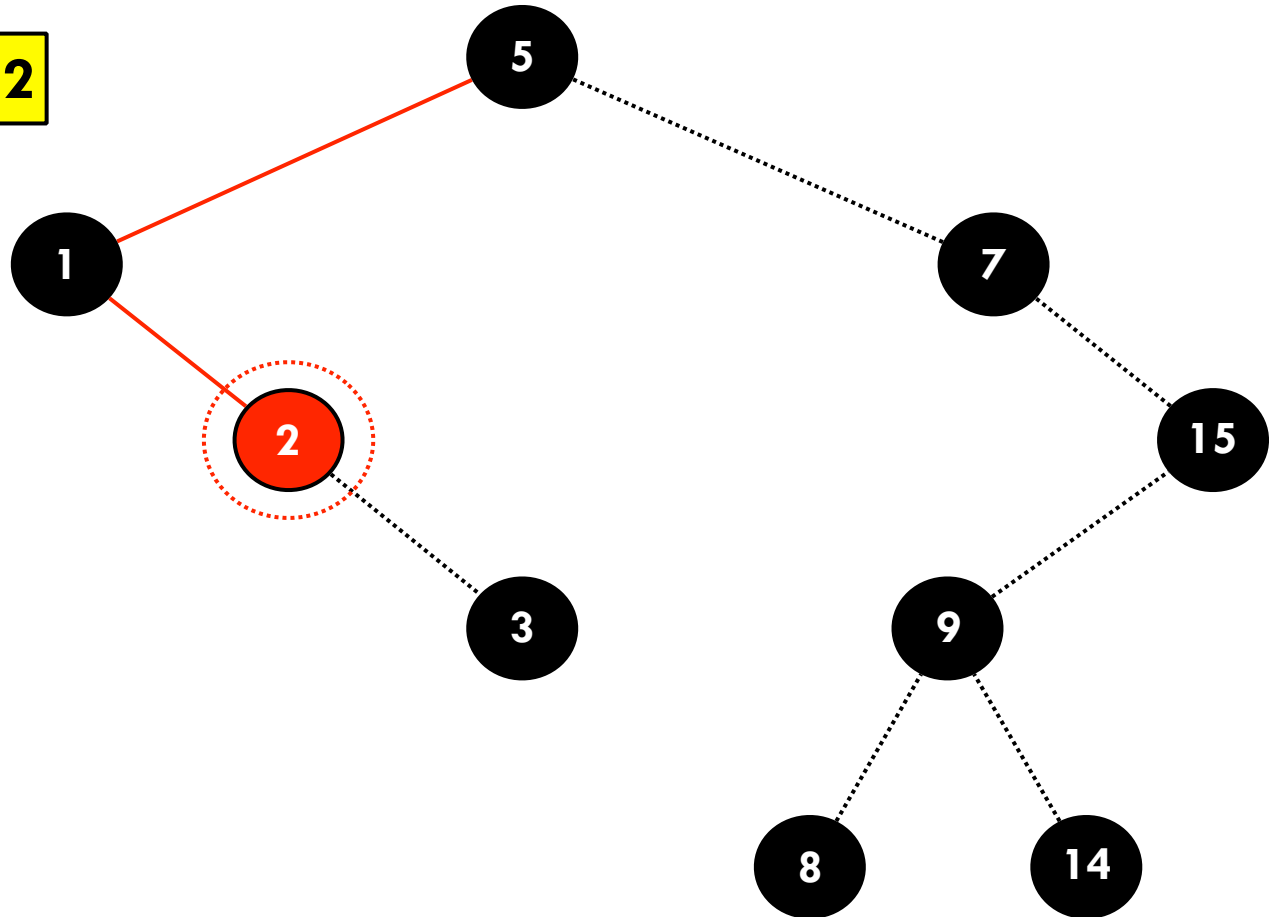
# Remoção

Remover  $x = 2$



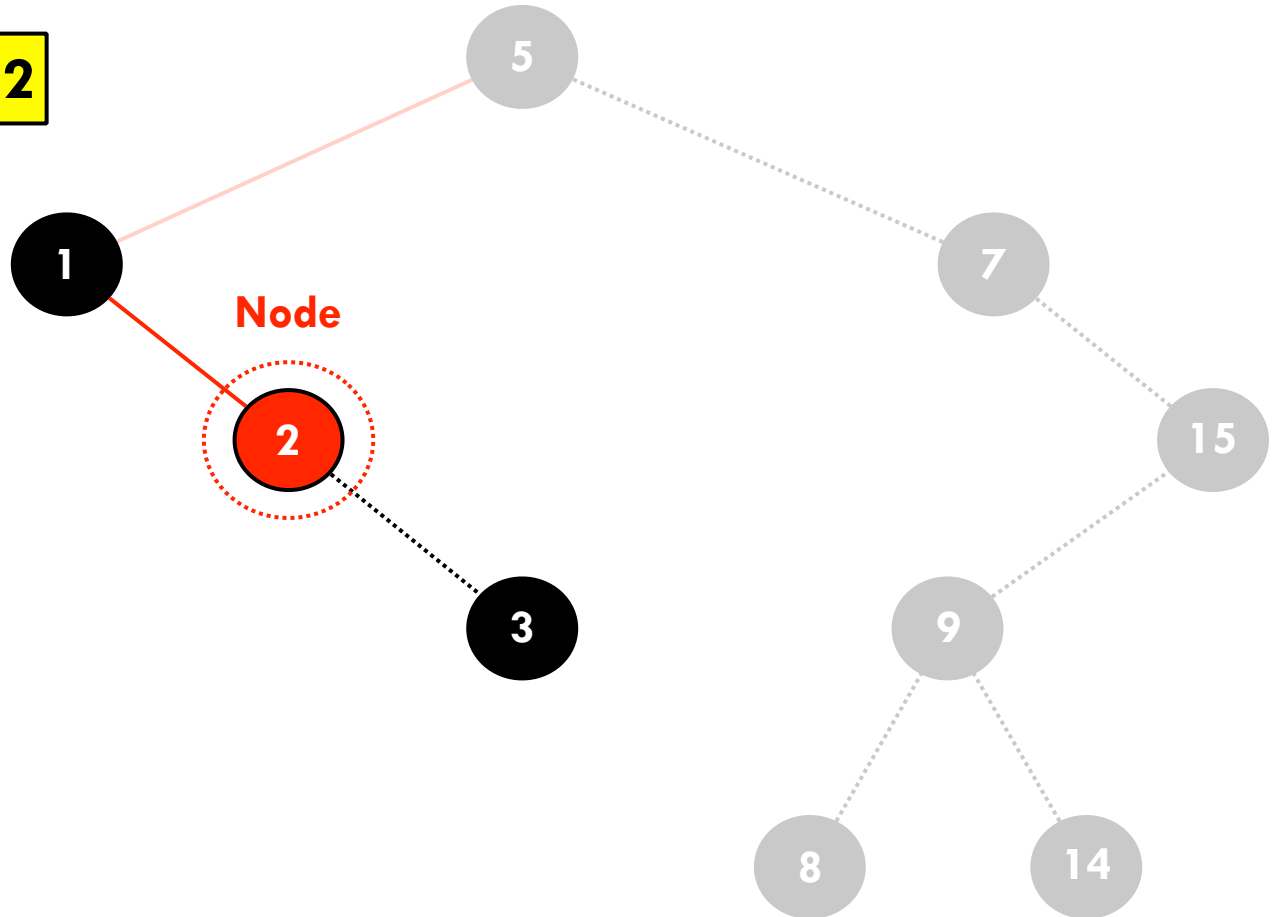
# Remoção

Remover  $x = 2$



# Remoção

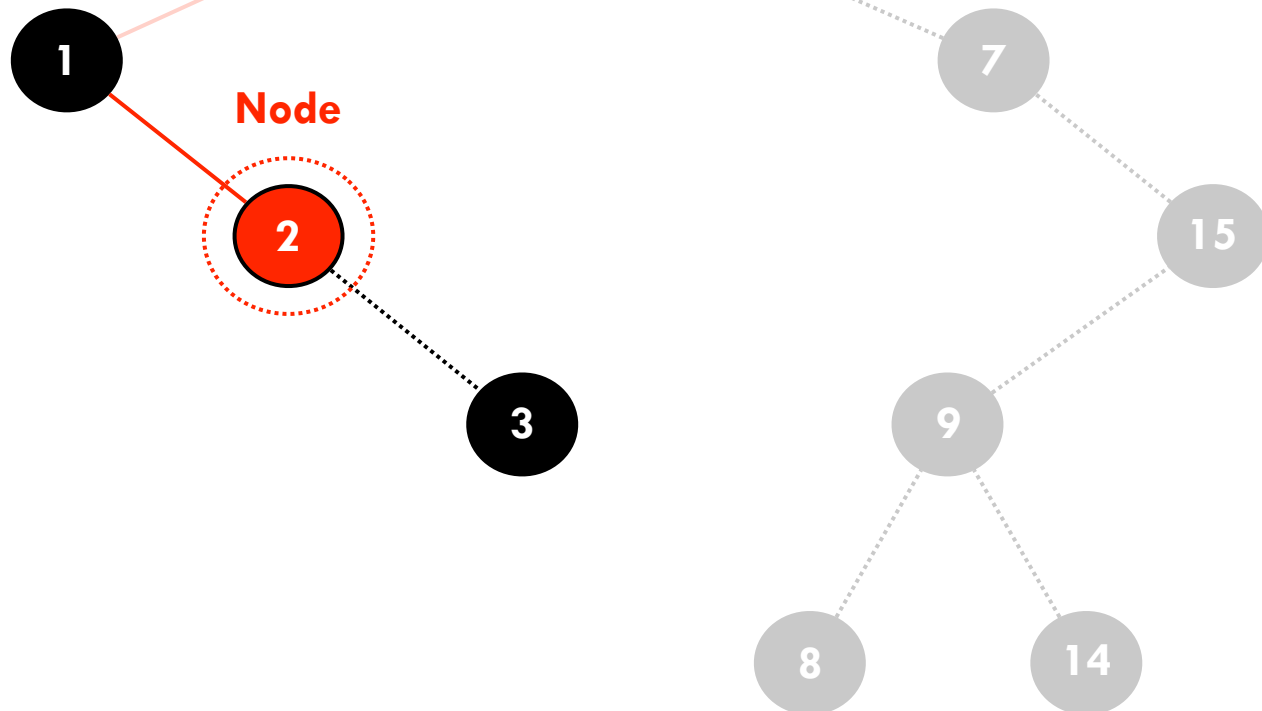
Remover  $x = 2$



# Remoção

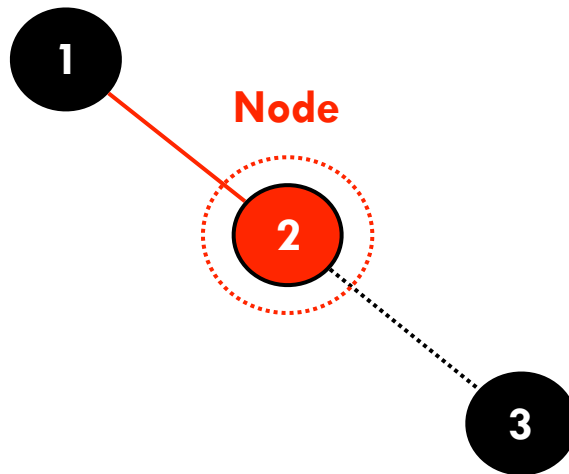
Remover  $x = 2$

Caso 1: sub-arvore esquerda é nula  
\* `node->esquerda == NULL`  
`node = node->direita`

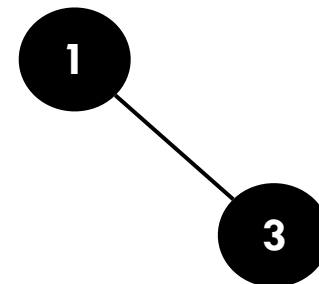


# Remoção

**Caso 1:** sub-arvore esquerda é nula  
\* node->esquerda == **NULL**  
node = node->direita

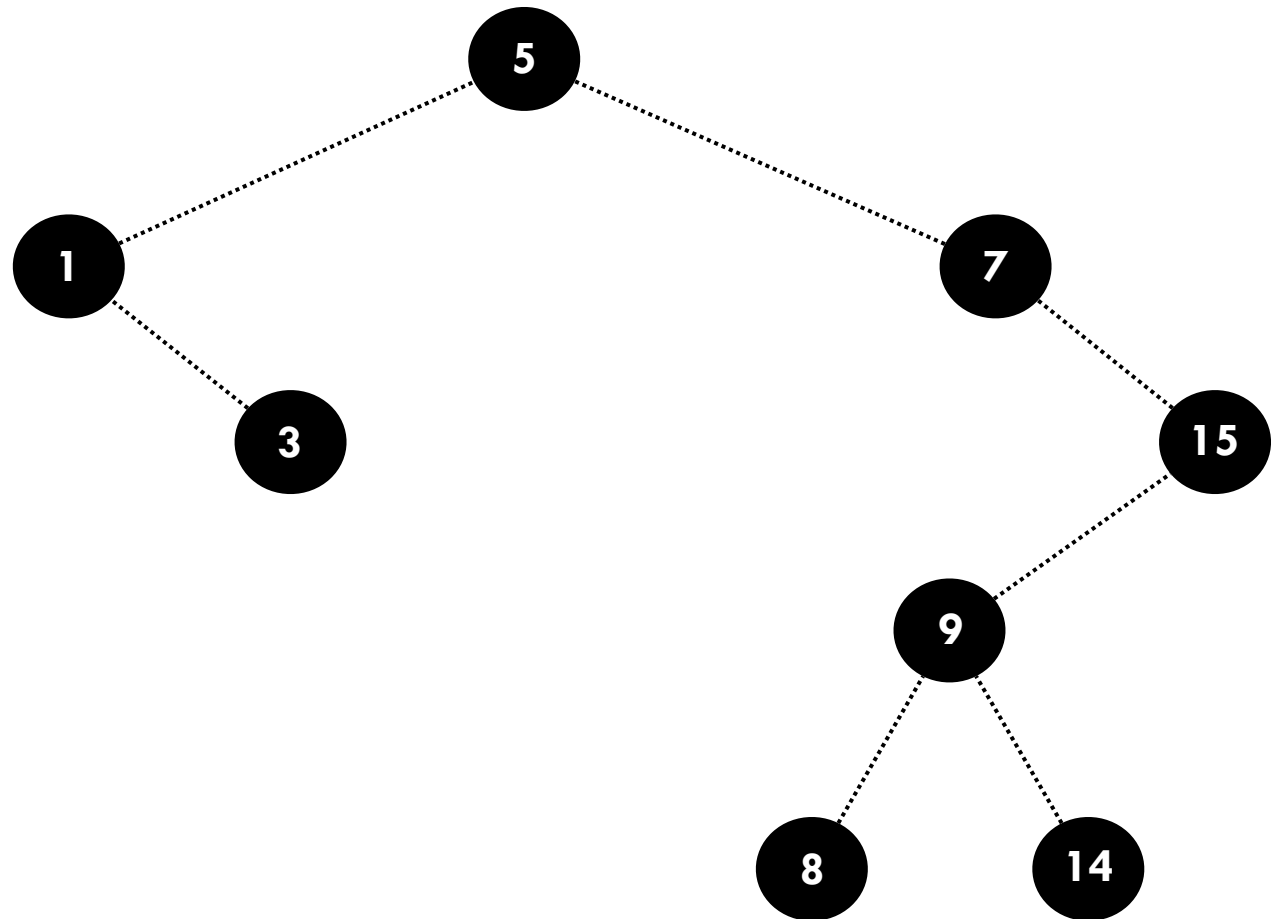


Antes da remoção



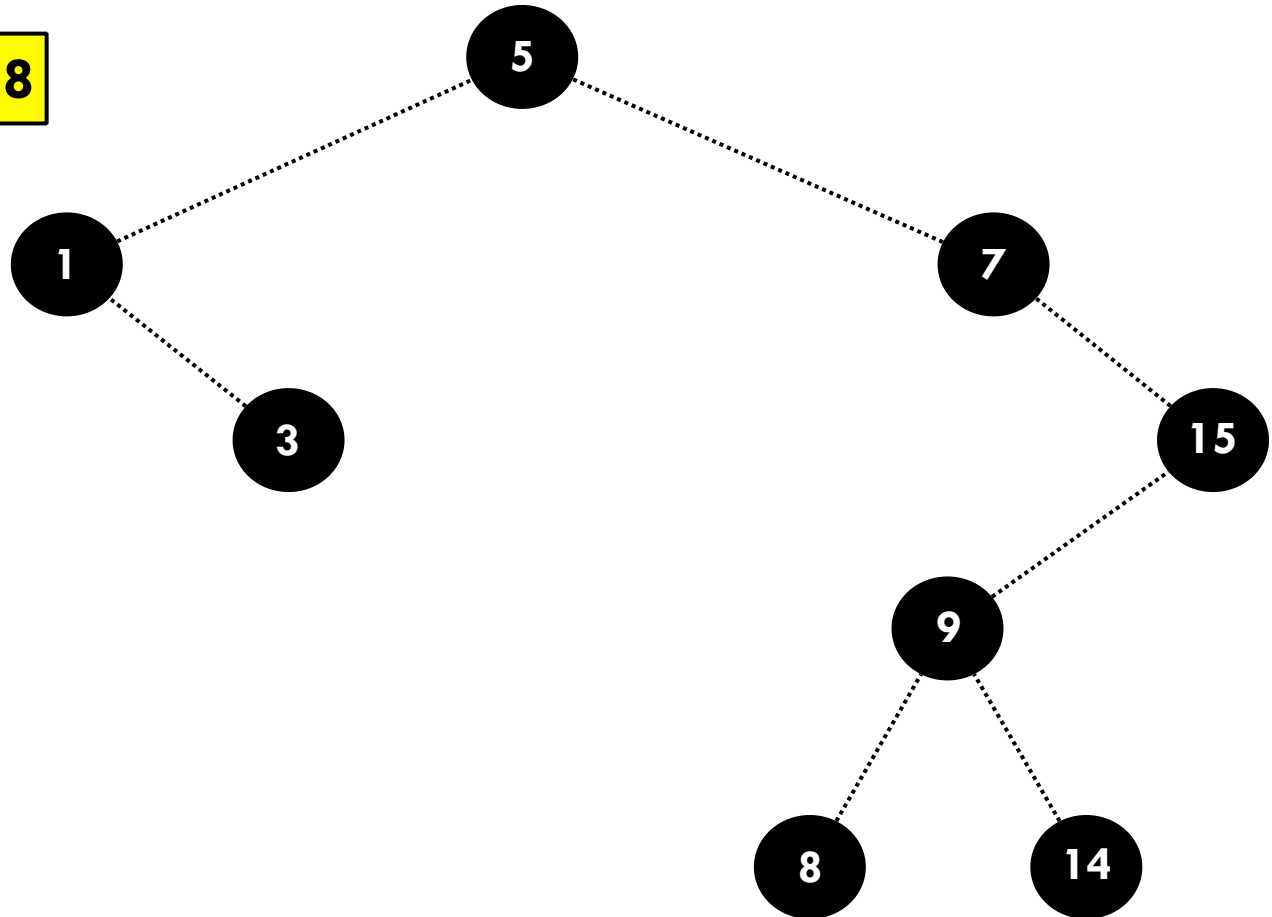
Depois da remoção

# Remoção



# Remoção

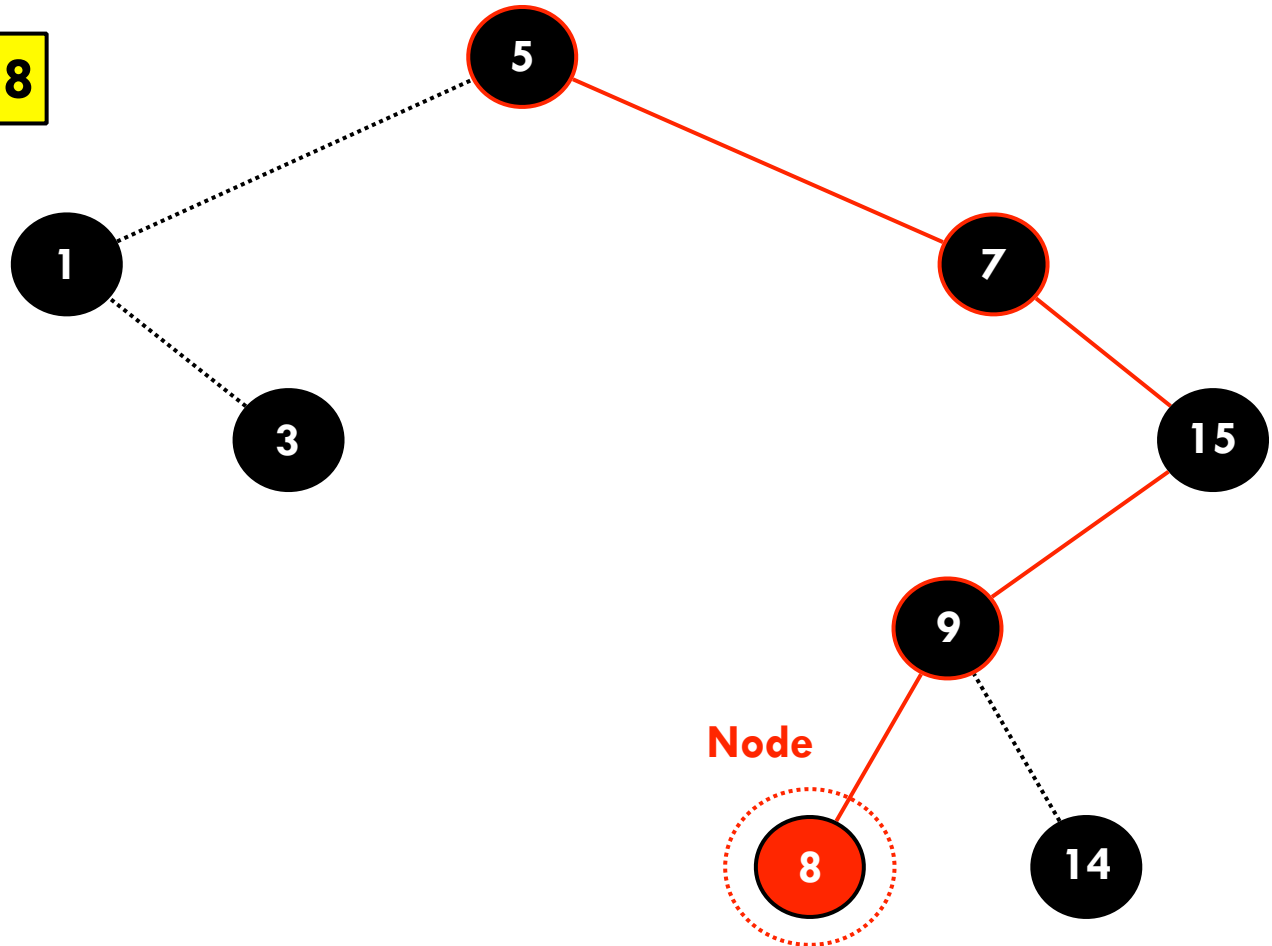
Remover  $x = 8$





# Remoção

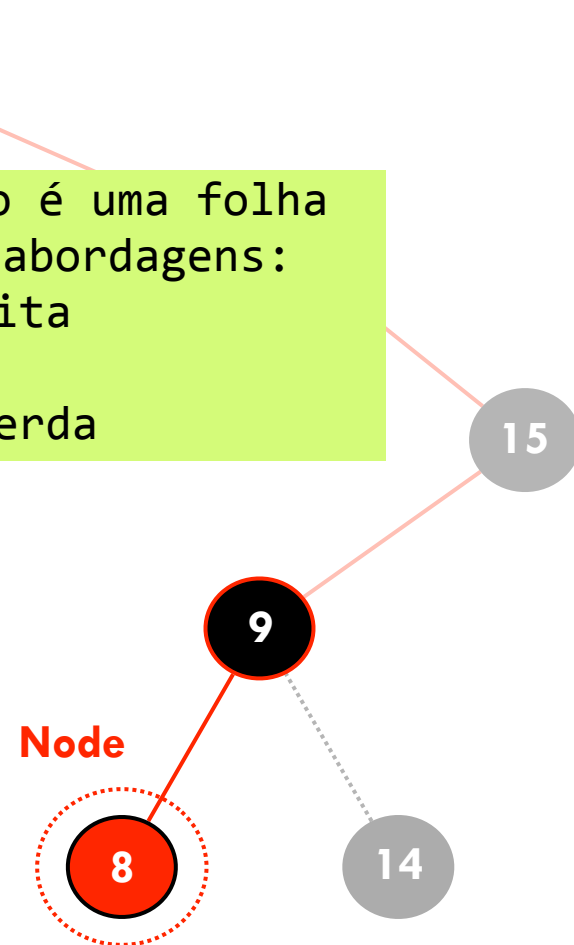
Remover  $x = 8$



# Remoção

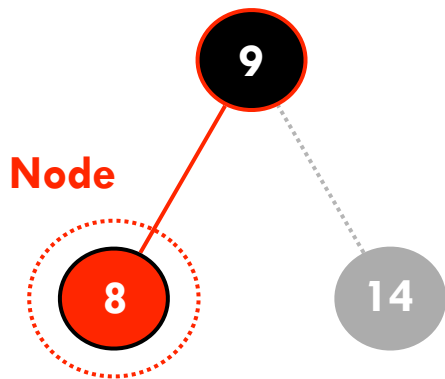
Remover  $x = 8$

Caso 2: nó a ser removido é uma folha  
\* podemos fazer duas abordagens:  
node = node->direita  
ou  
node = node->esquerda

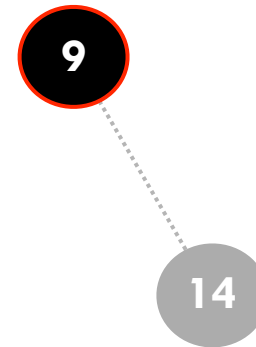


# Remoção

Caso 2: nó a ser removido é uma folha  
\* podemos fazer duas abordagens:  
    `node = node->direita`  
ou  
    `node = node->esquerda`



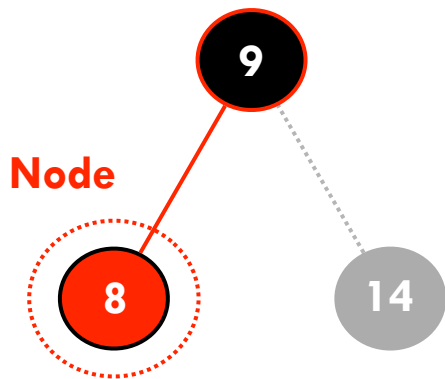
Antes da remoção



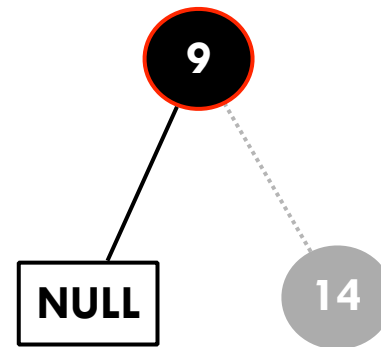
Depois da remoção

# Remoção

Caso 2: nó a ser removido é uma folha  
\* podemos fazer duas abordagens:  
    `node = node->direita`  
ou  
    `node = node->esquerda`

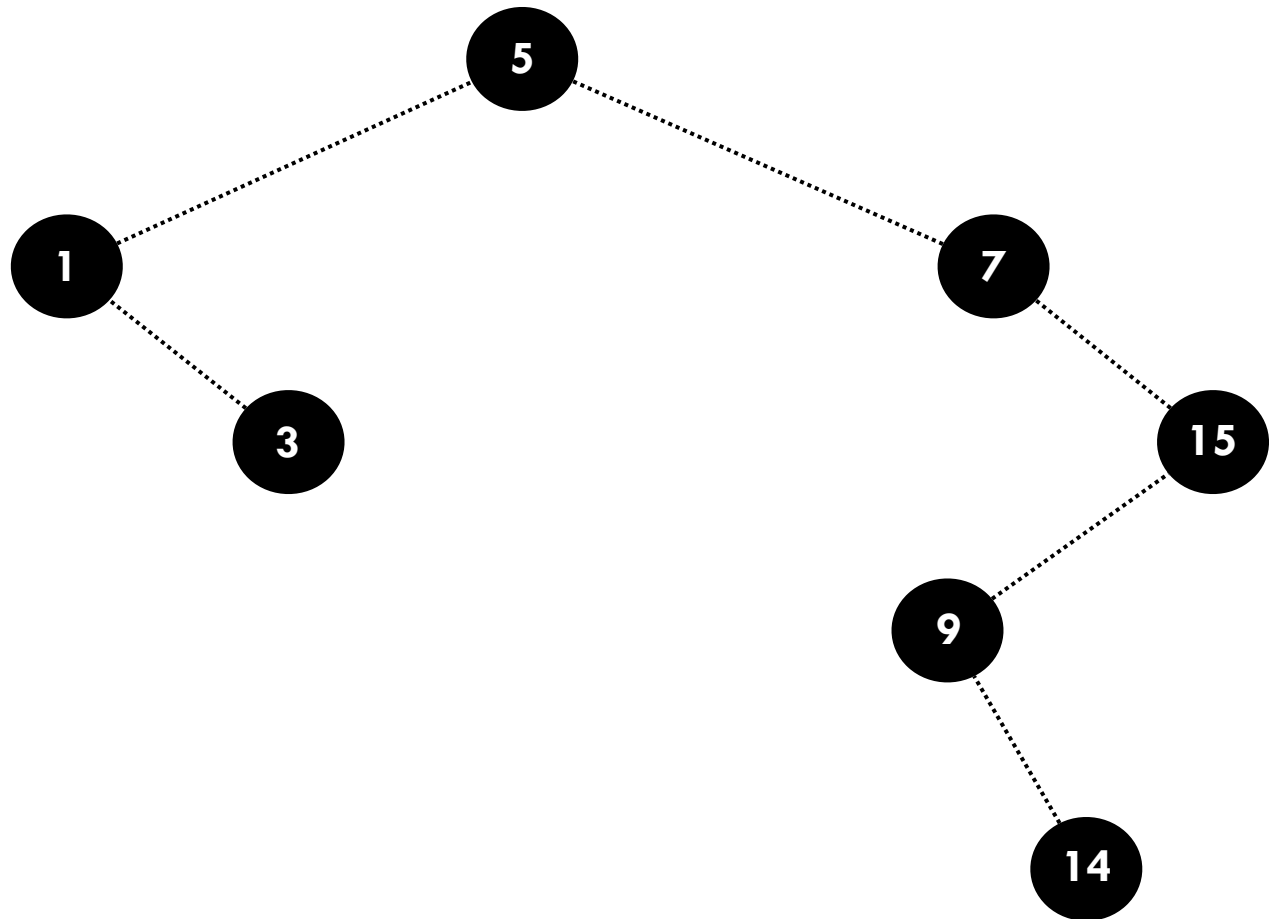


Antes da remoção



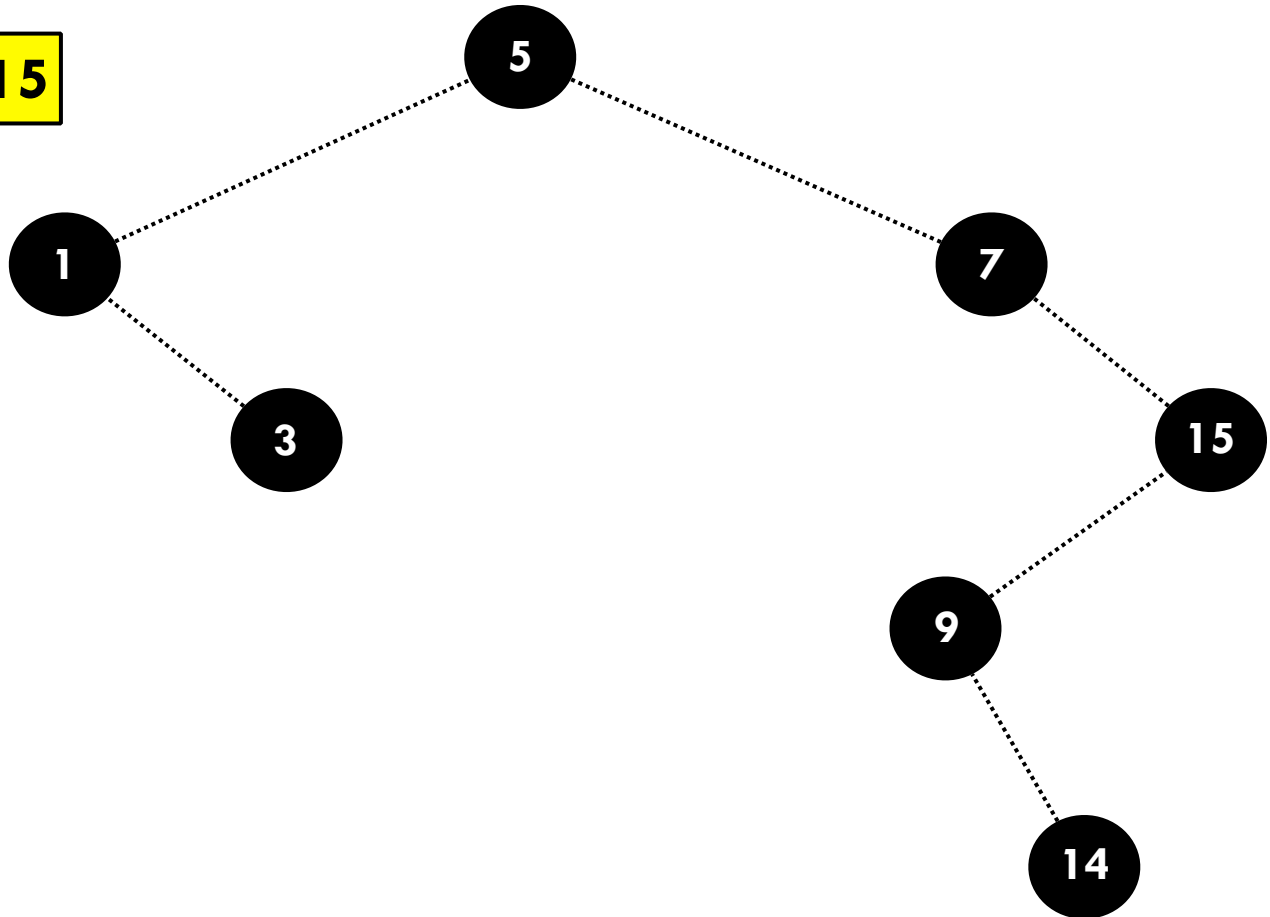
Depois da remoção

# Remoção



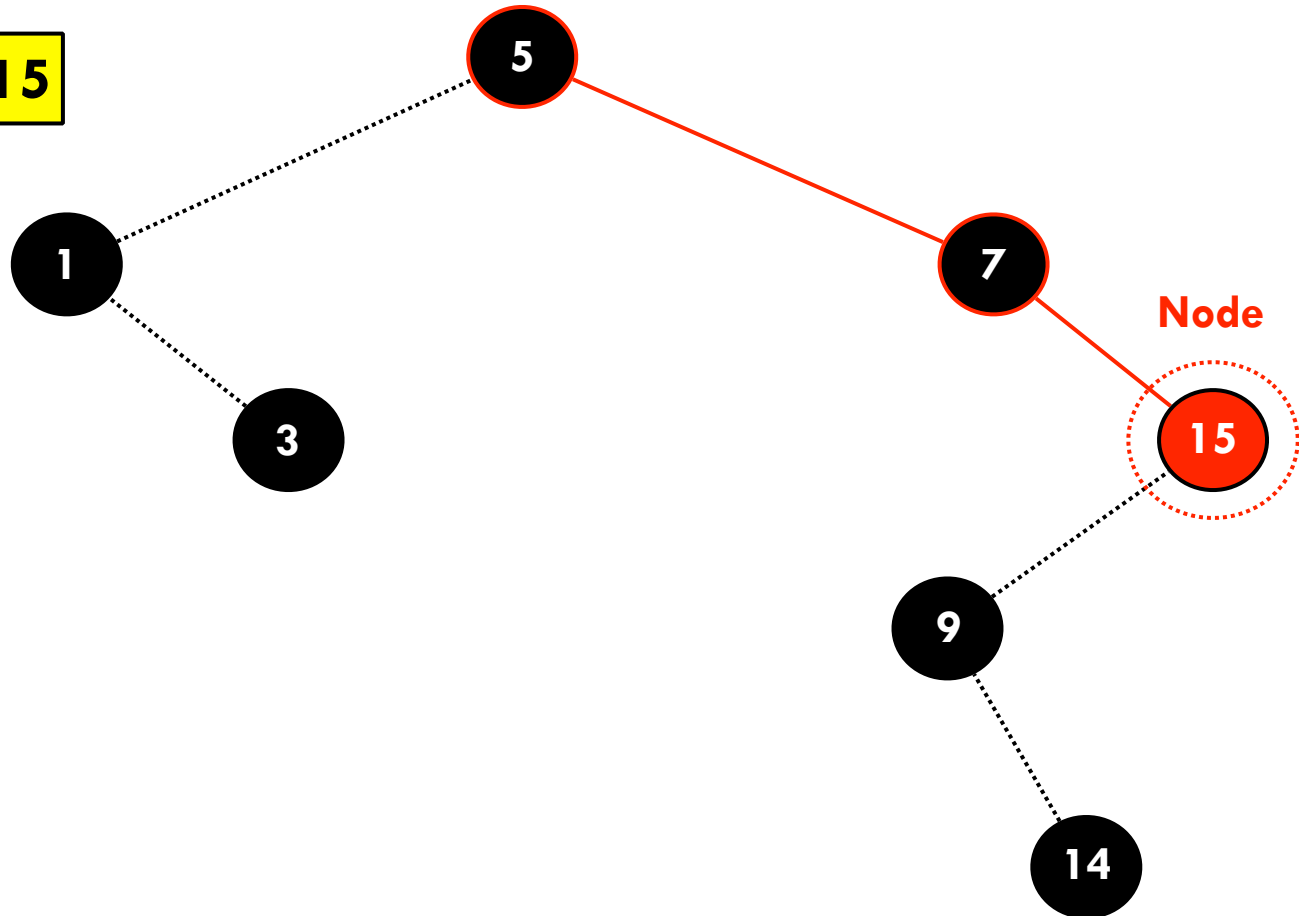
# Remoção

Remover  $x = 15$



# Remoção

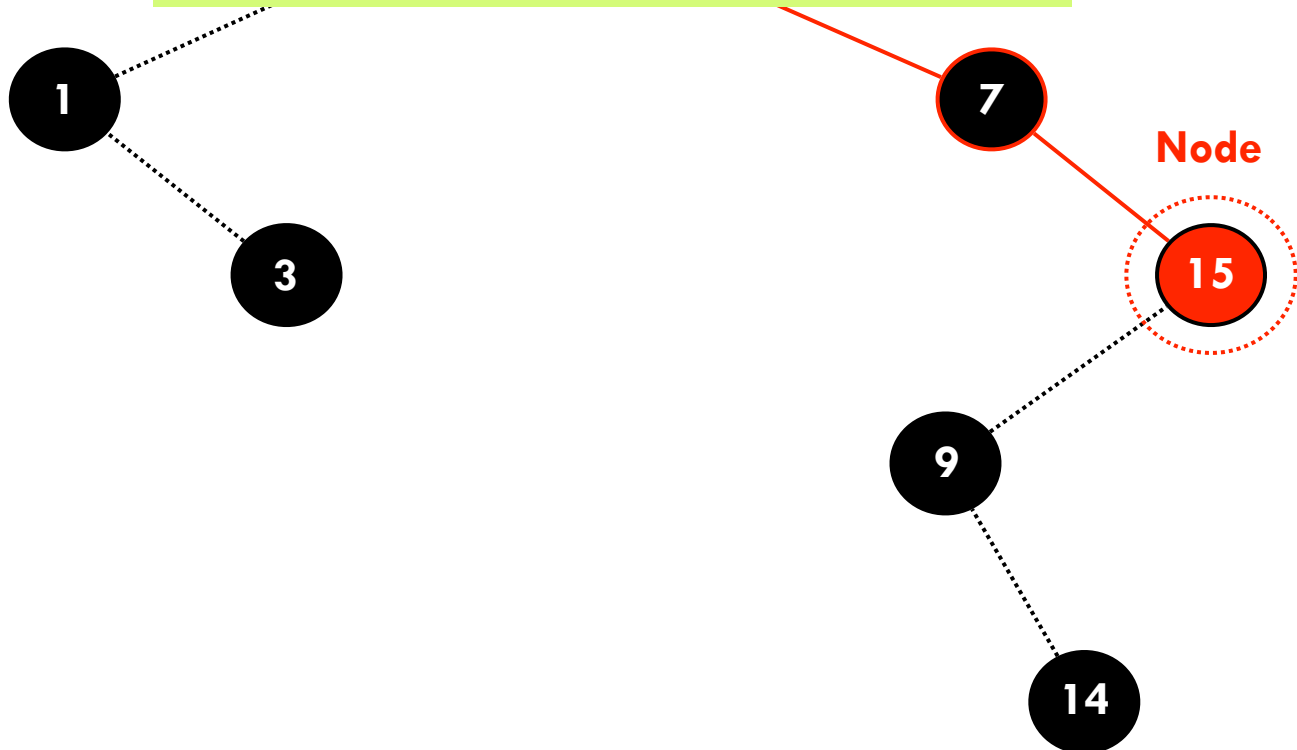
Remover  $x = 15$



# Remoção

Remover  $x = 15$

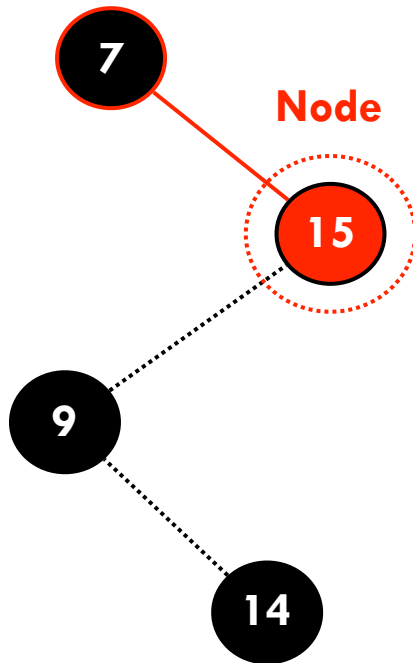
Caso 3: sub-arvore direita é nula  
\*  $\text{node} \rightarrow \text{direita} == \text{NULL}$   
node = node->esquerda



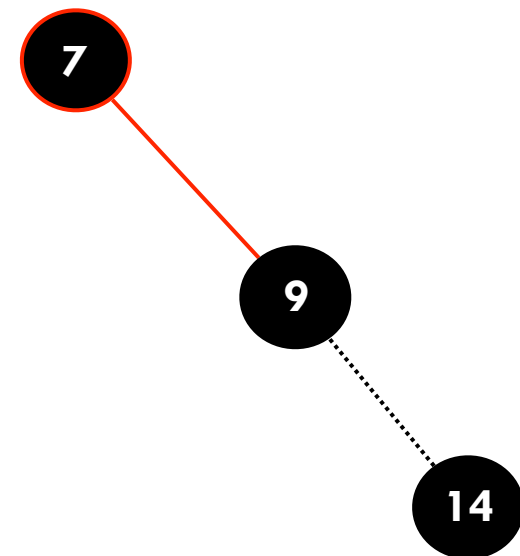


# Remoção

Caso 3: sub-arvore direita é nula  
\* node->direita == NULL  
node = node->esquerda

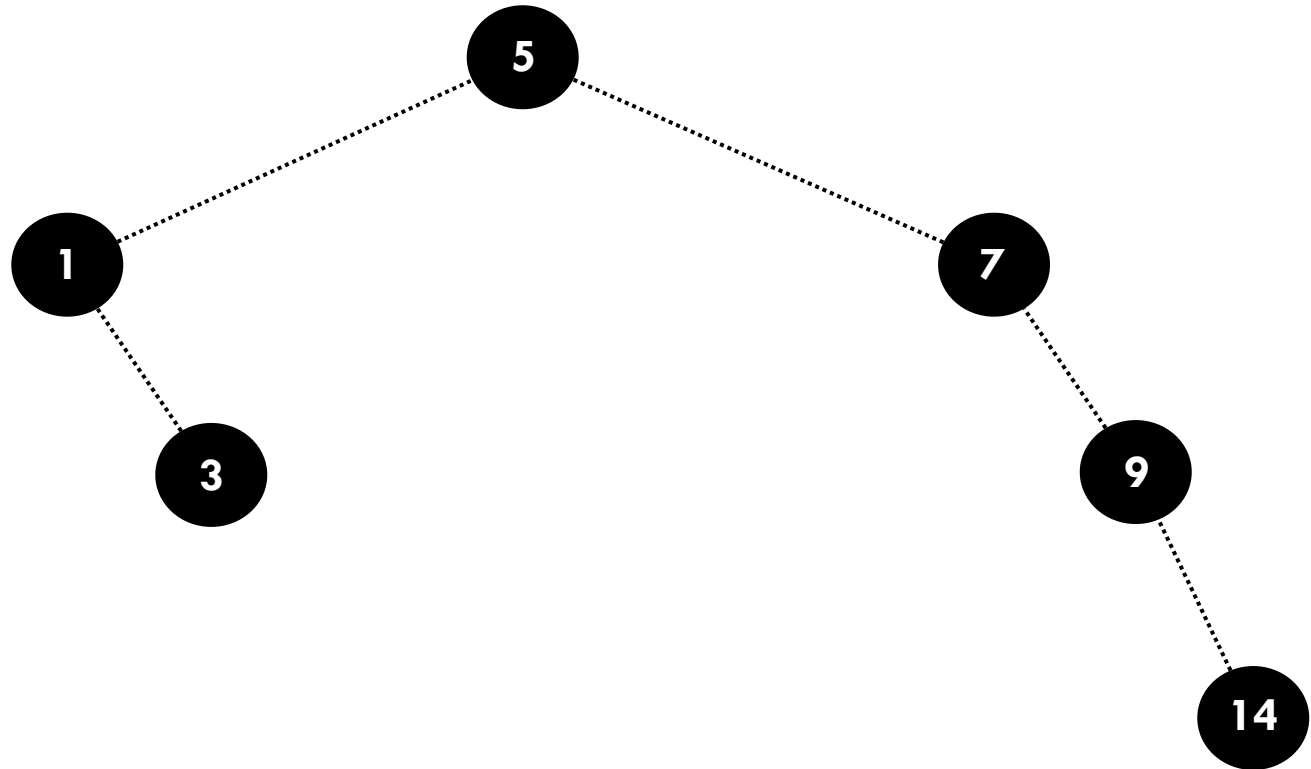


Antes da remoção



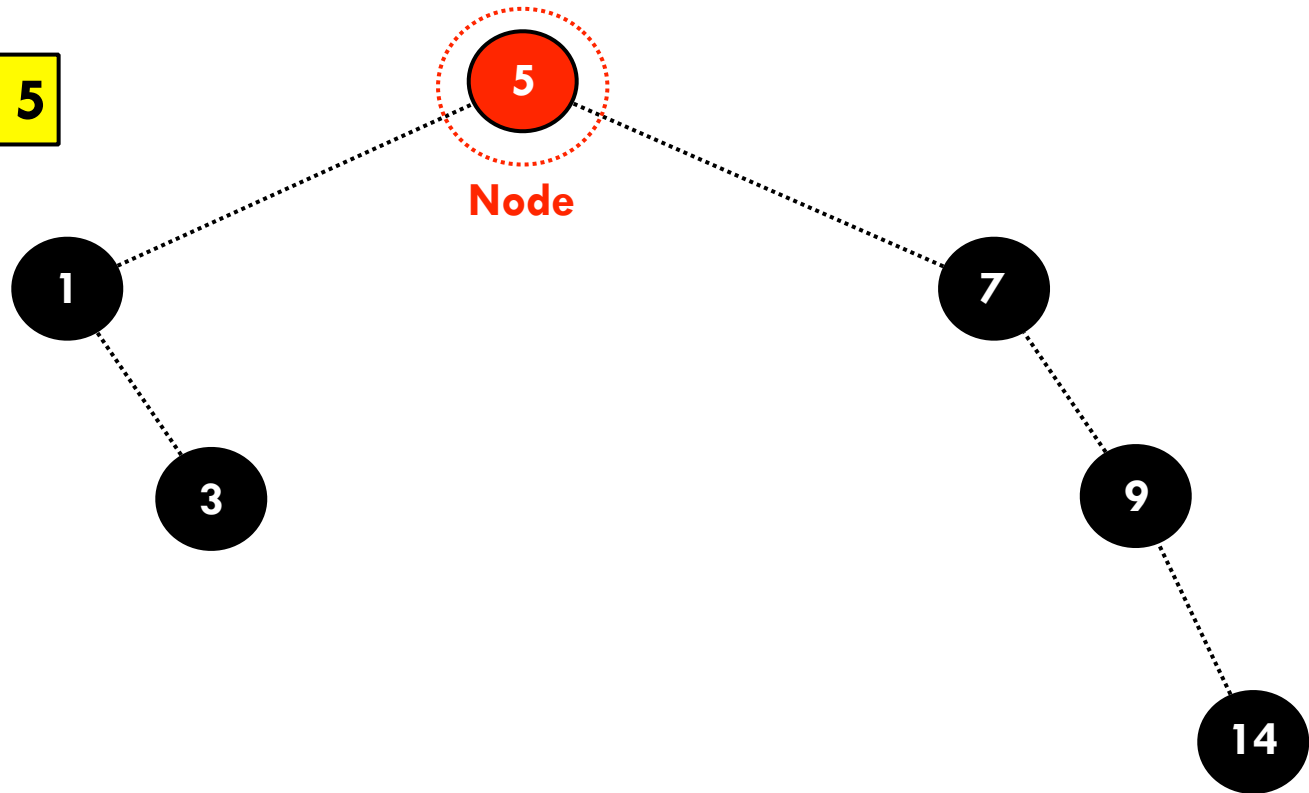
Depois da remoção

# Remoção



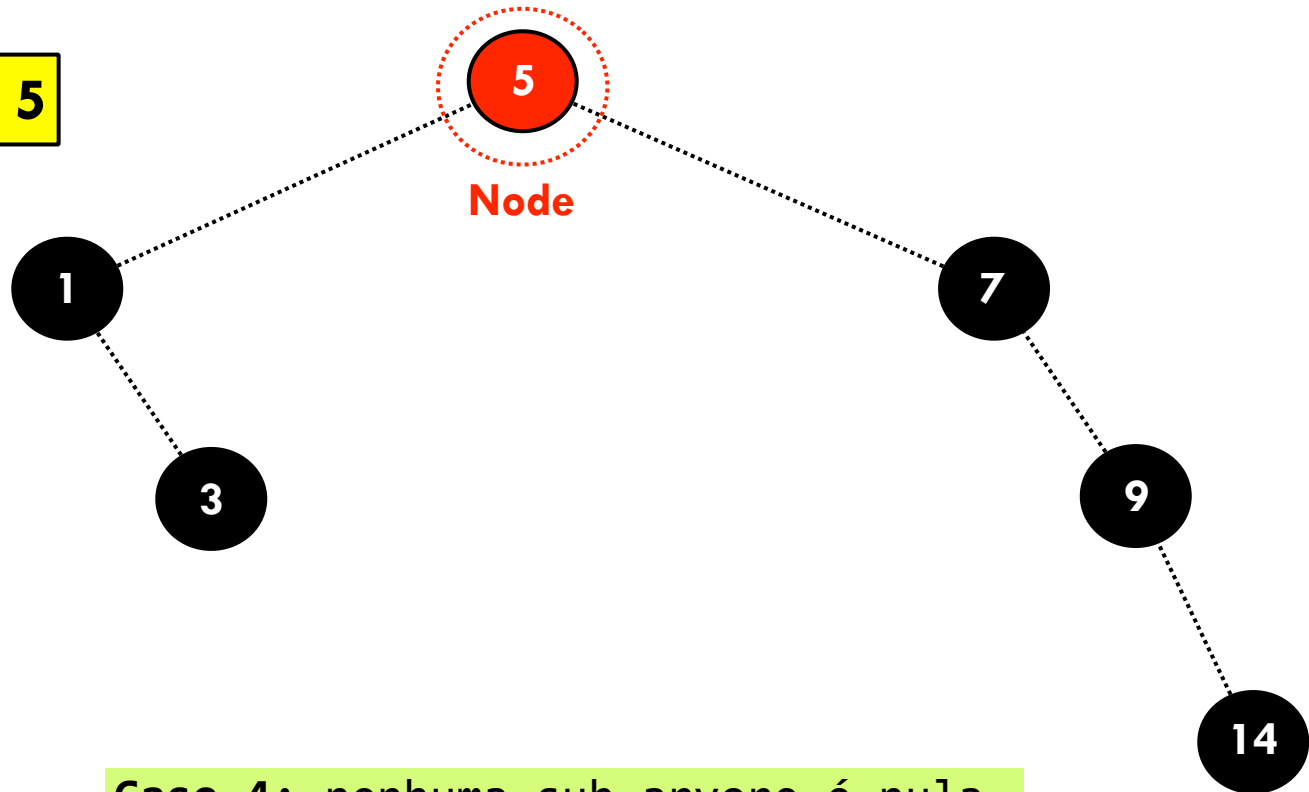
# Remoção

Remover  $x = 5$



# Remoção

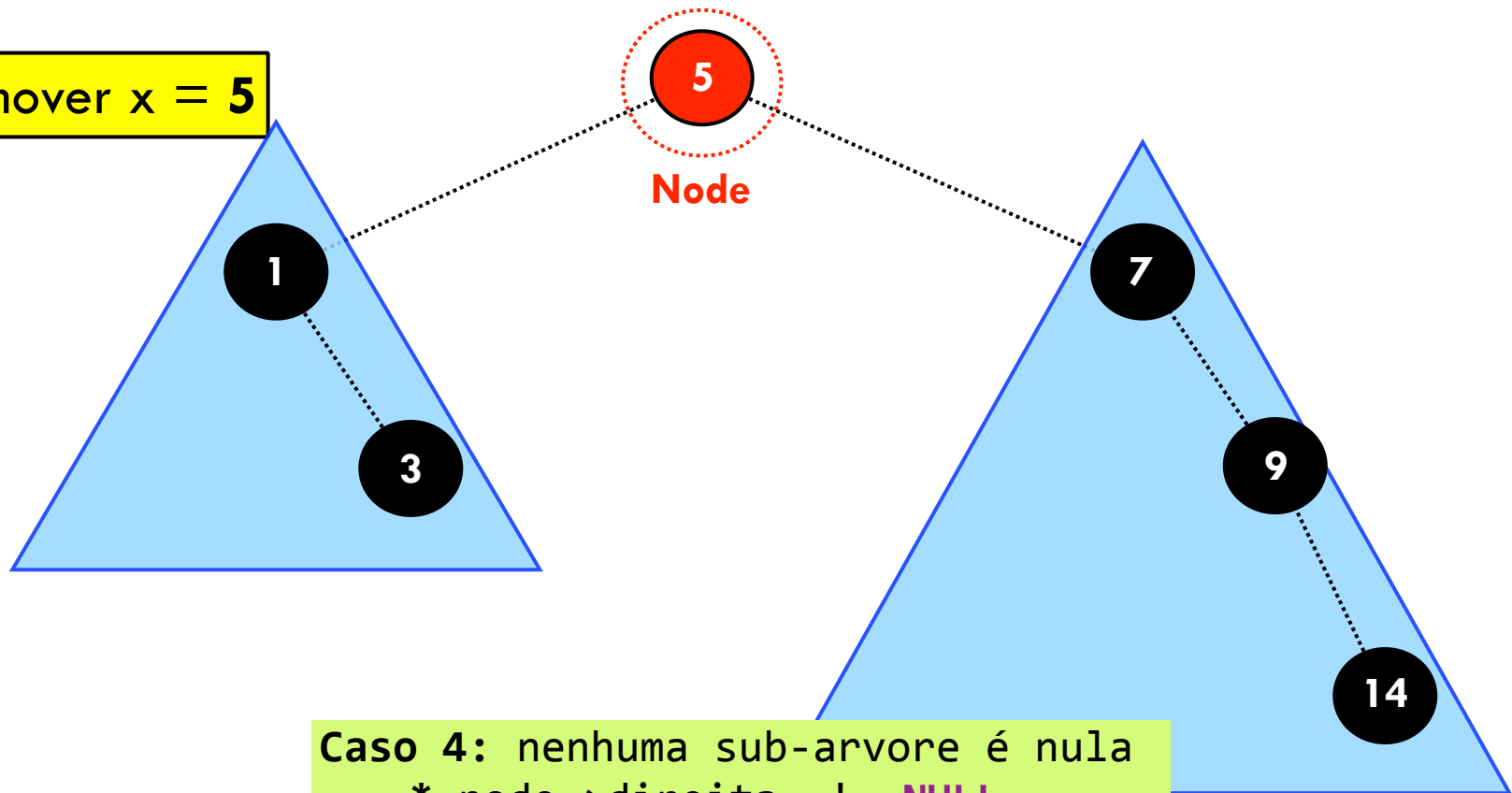
Remover  $x = 5$



Caso 4: nenhuma sub-arvore é nula  
\* node->direita != NULL  
\* node->esquerda != NULL

# Remoção

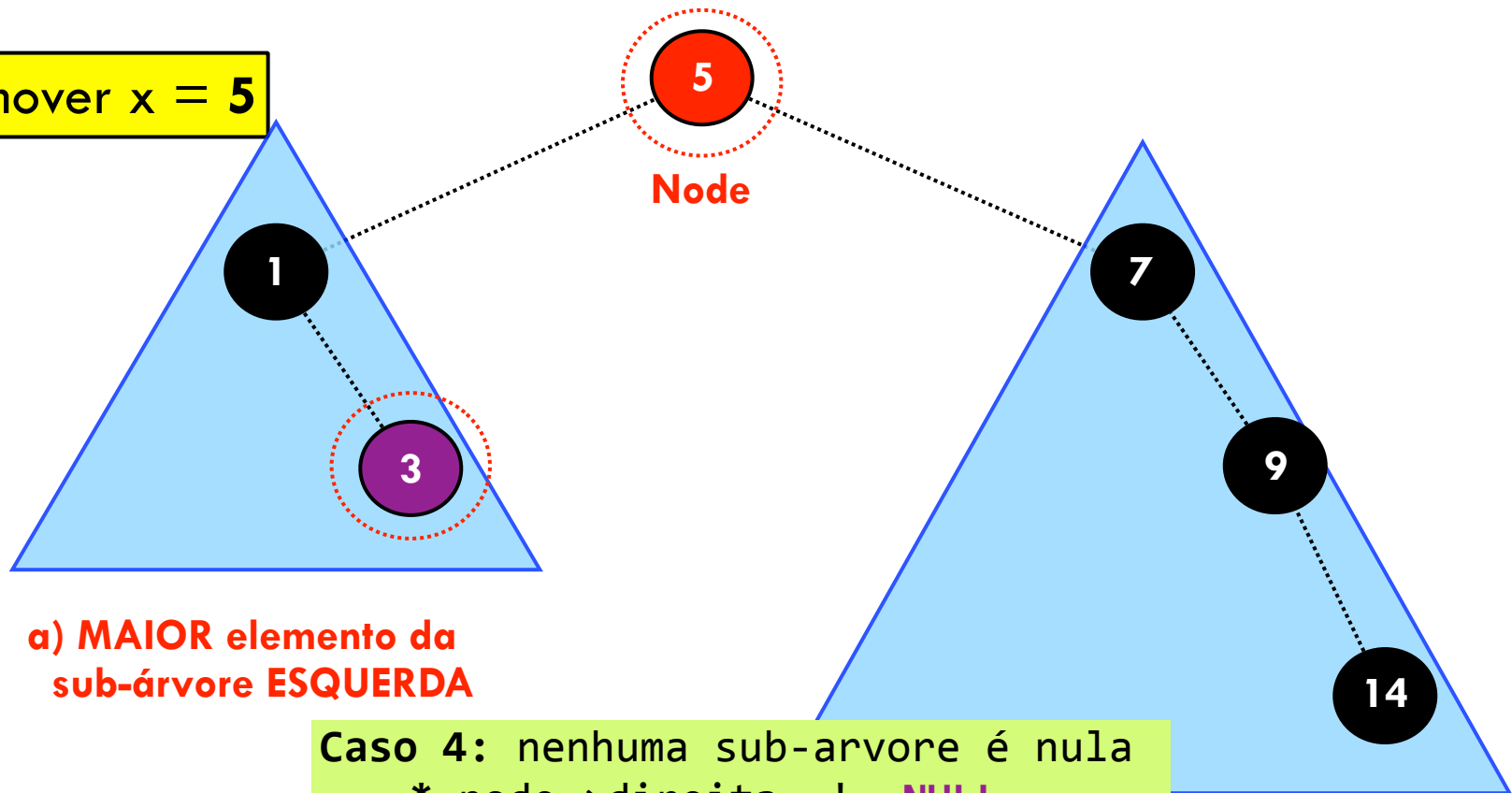
Remover  $x = 5$



Caso 4: nenhuma sub-arvore é nula  
\* `node->direita`  $\neq$  NULL  
\* `node->esquerda`  $\neq$  NULL

# Remoção

Remover  $x = 5$

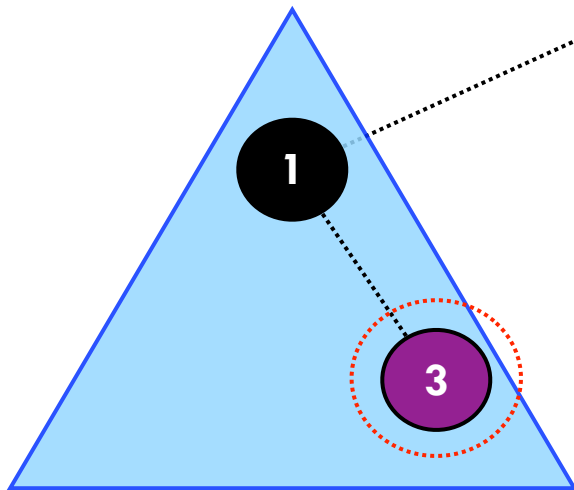


a) **MAIOR** elemento da sub-árvore **ESQUERDA**

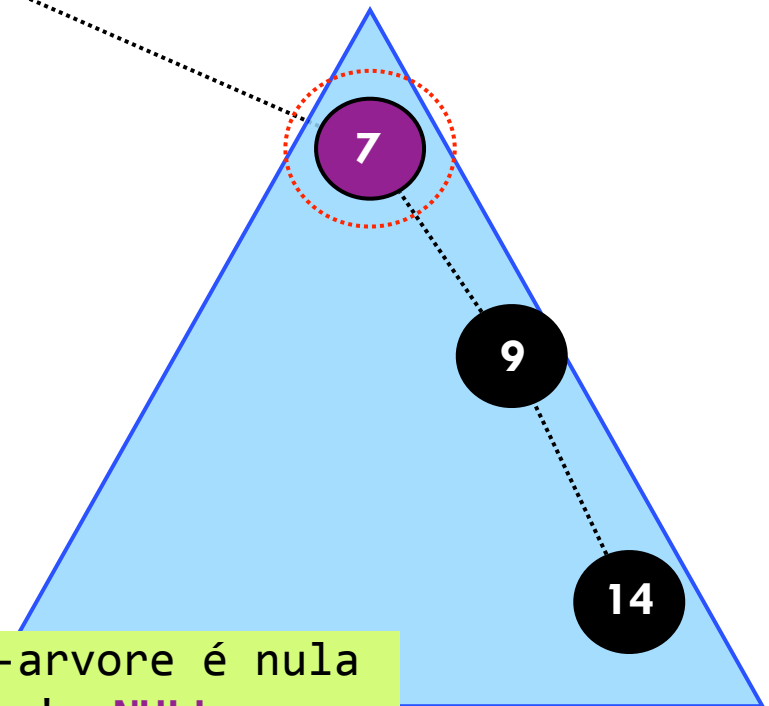
Caso 4: nenhuma sub-arvore é nula  
\* node->direita != NULL  
\* node->esquerda != NULL

# Remoção

a) MAIOR elemento da sub-árvore ESQUERDA



b) MENOR elemento da sub-árvore DIREITA

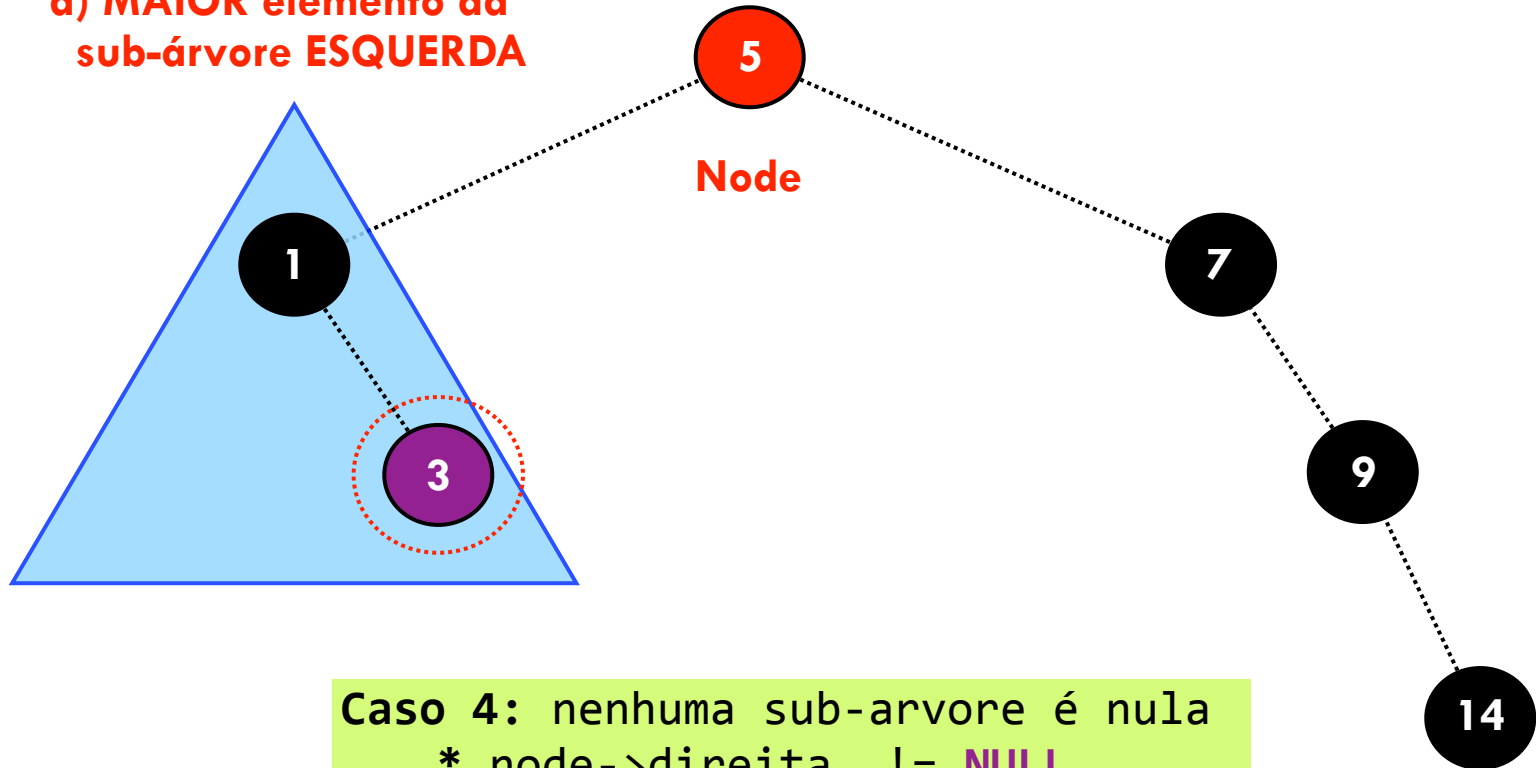


5  
Node

Caso 4: nenhuma sub-arvore é nula  
\* node->direita != NULL  
\* node->esquerda != NULL

# Remoção

a) MAIOR elemento da sub-árvore ESQUERDA

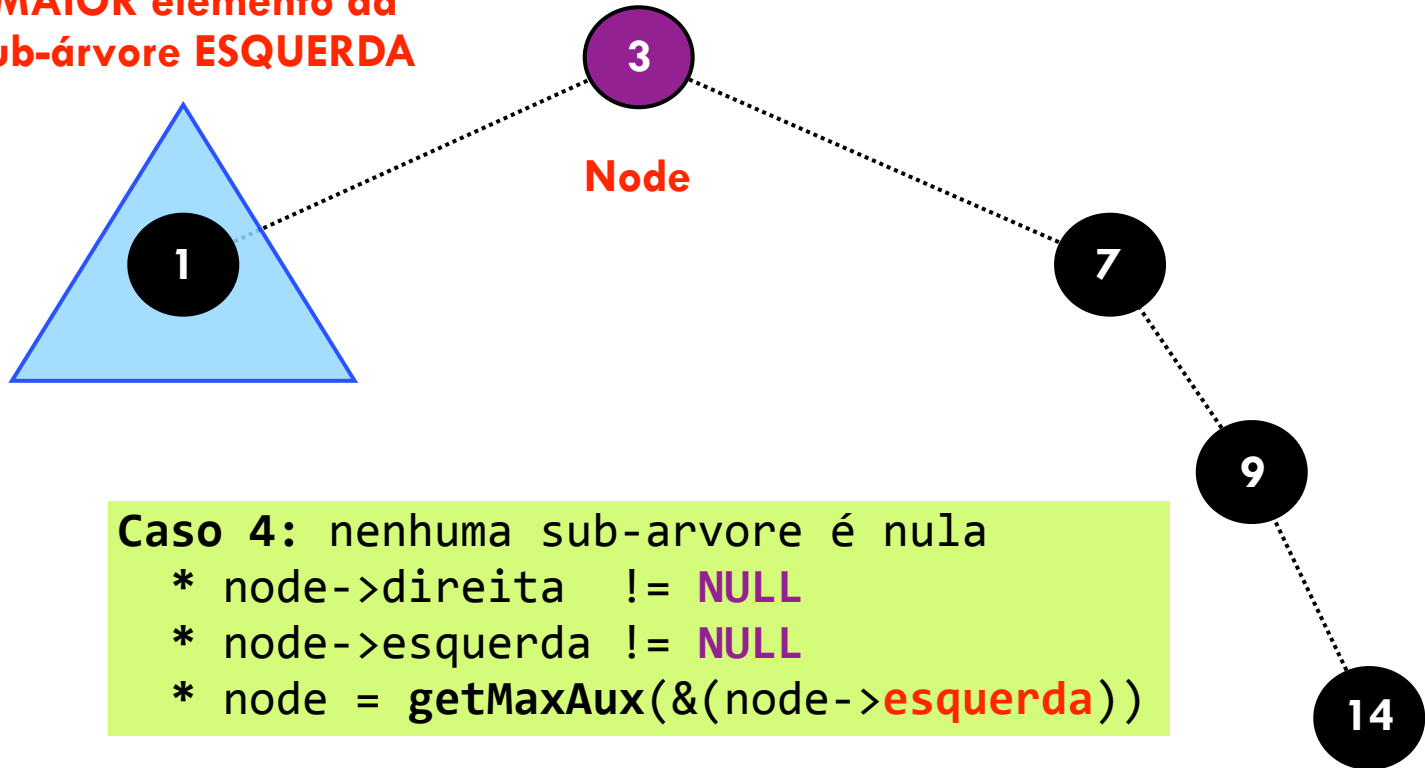


Caso 4: nenhuma sub-arvore é nula  
\* node->direita != NULL  
\* node->esquerda != NULL



# Remoção

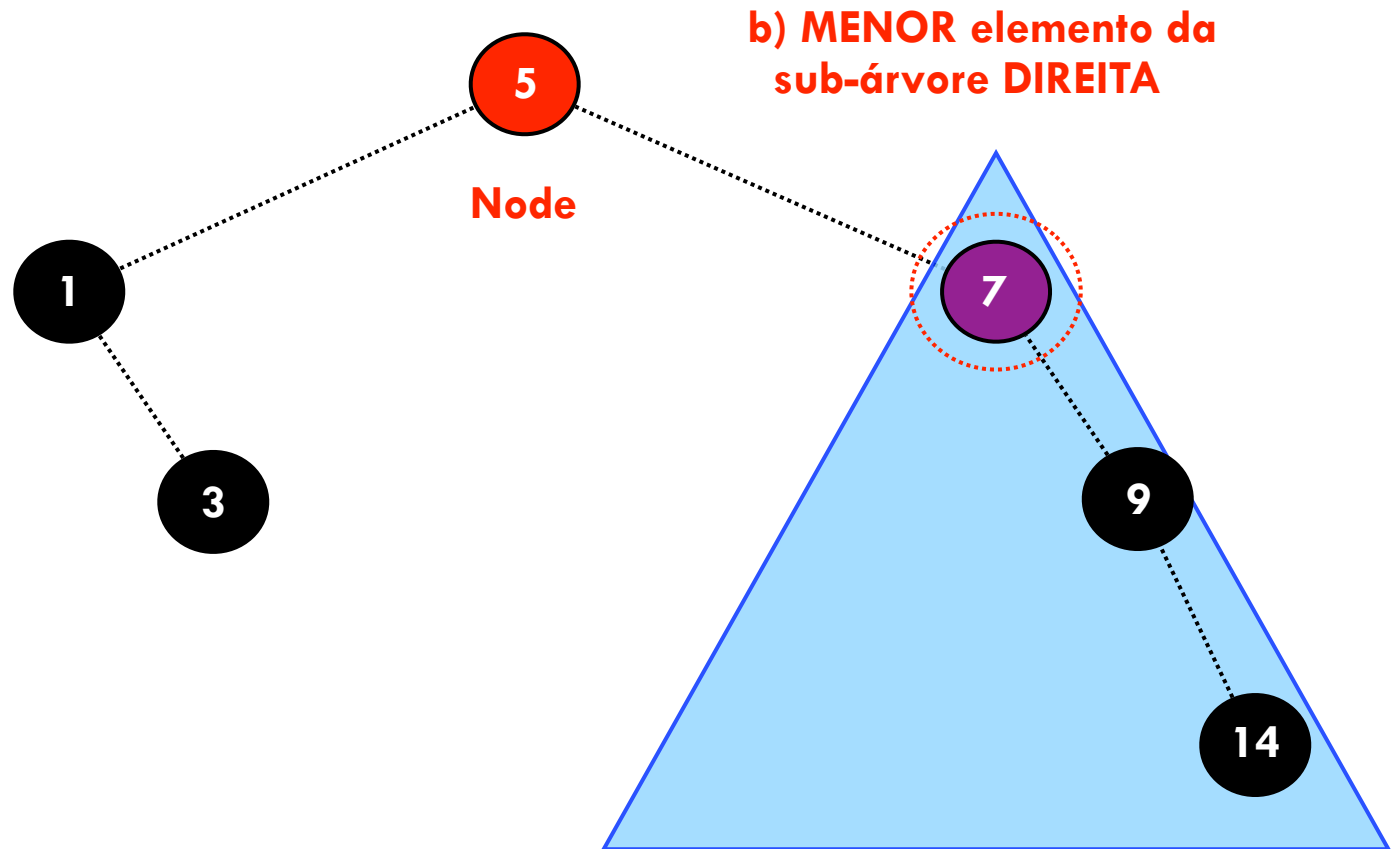
a) MAIOR elemento da sub-árvore ESQUERDA



Caso 4: nenhuma sub-arvore é nula

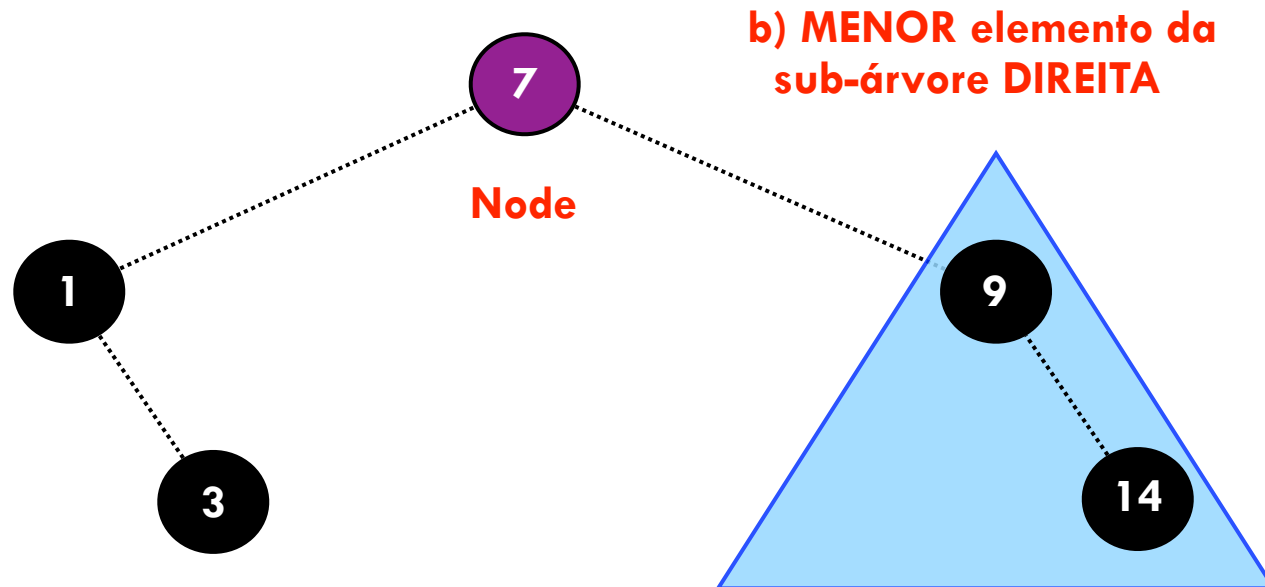
```
* node->direita != NULL  
* node->esquerda != NULL  
* node = getMaxAux(&(node->esquerda))
```

# Remoção



Caso 4: nenhuma sub-árvore é nula  
\* node->direita != NULL  
\* node->esquerda != NULL

# Remoção



Caso 4: nenhuma sub-arvore é nula

```
* node->direita != NULL  
* node->esquerda != NULL  
* node = getMinAux(&(node->direita))
```

# Remoção

Remoção (Ponteiro \*node, Item x)

1. se \*node == NULL, não existe elemento a ser removido
  1. return (**false**);
2. se (\*node)->elemento.chave == x.chave  
*// existe e tem q remover, verificar qual caso é*
  1. Caso1: sub-arvore esquerda nula
  2. Caso2: sub-arvore direita nula
  3. Caso3: nó folha
  4. Caso4: existem ambas as sub-arvores
  5. return(**true**);
3. se (\*node)->elemento.chave > x.chave
  1. return( Remoção recursivamente p filho da esquerda);
4. senão
  1. return( Remoção recursivamente p filho da direita);

# Exercício 01



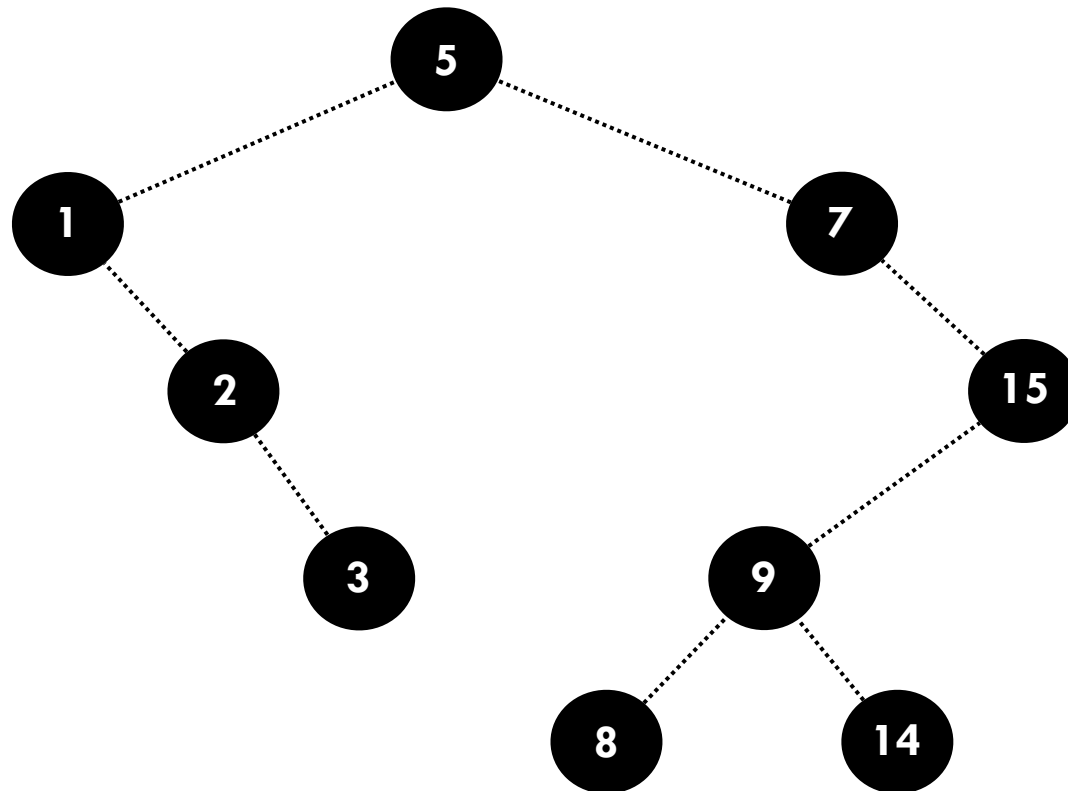
- Implementar a operação de remoção de árvores binárias;

# Roteiro

- 1 Tópicos já vistos anteriormente
- 2 Remoção em Árvore Binárias
- 3 Próximos conteúdos
- 4 Referências

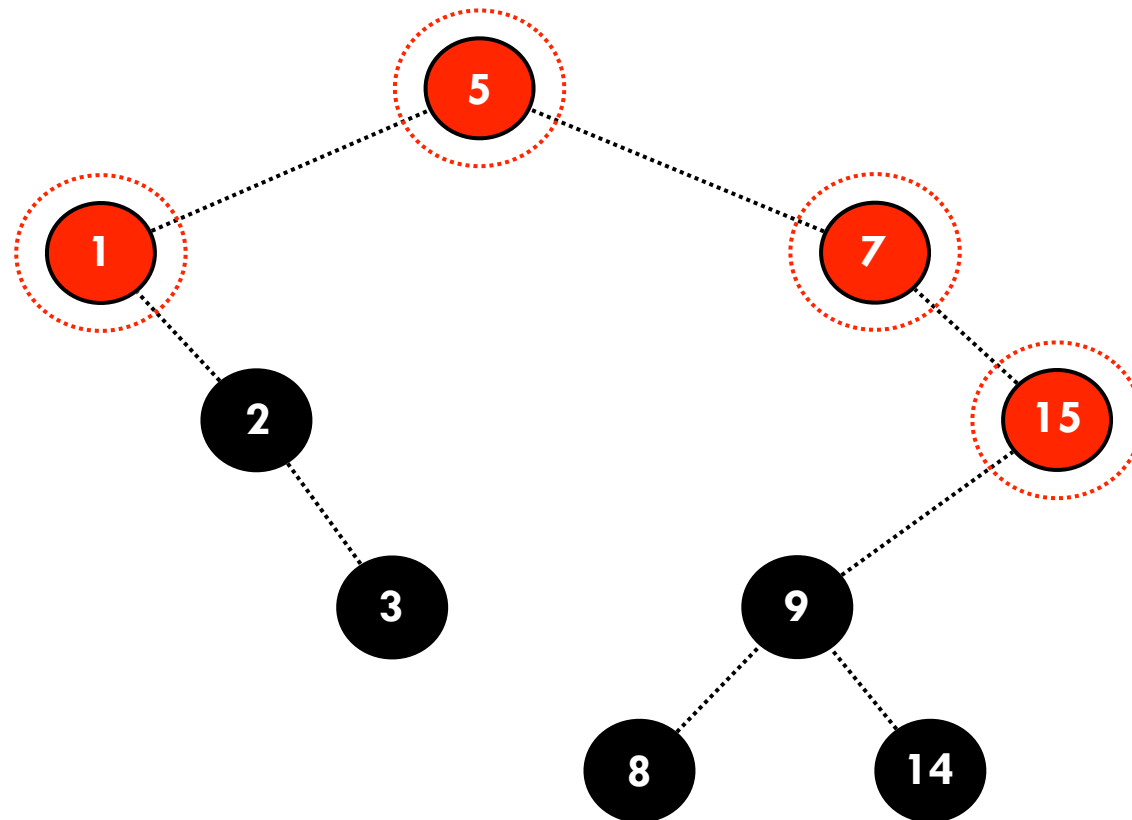
# Próximas aulas

## □ Desbalanceamentos



# Próximas aulas

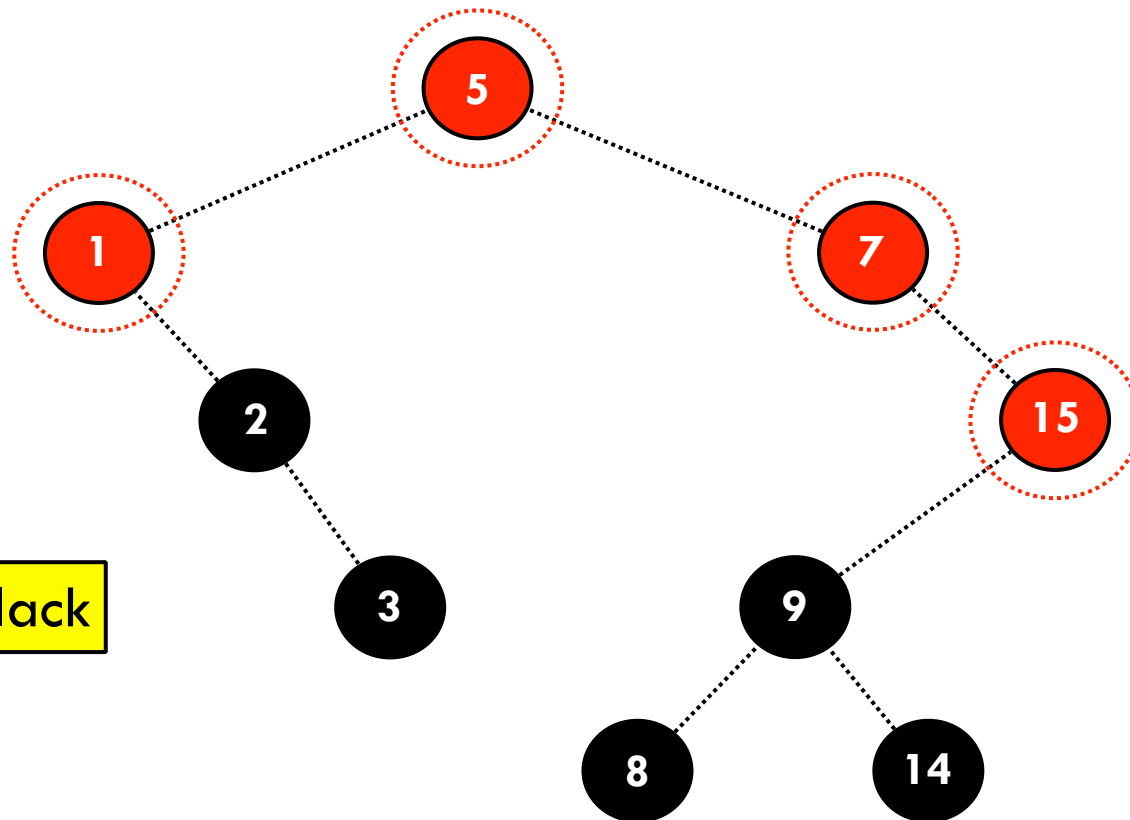
## □ Desbalanceamentos





# Próximas aulas

## □ Desbalanceamentos



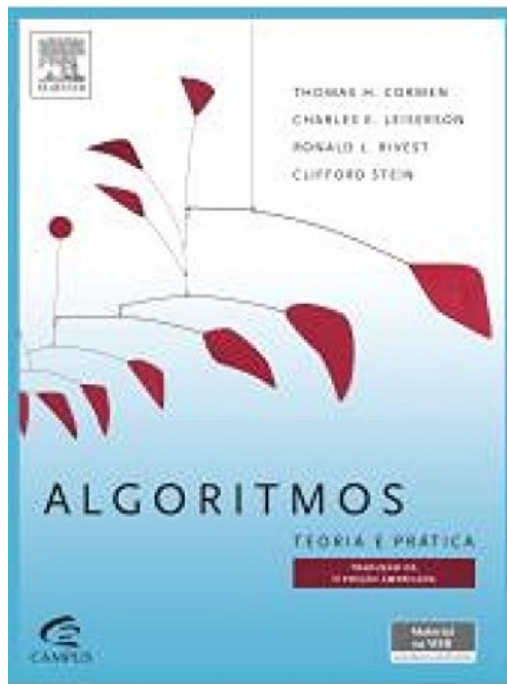
Árvores AVLs

Árvores Red-black

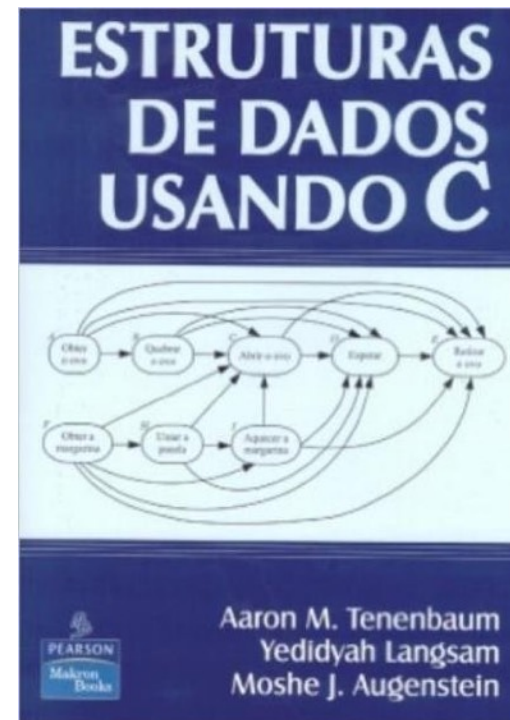
# Roteiro

- 1 Tópicos já vistos anteriormente
- 2 Remoção em Árvore Binárias
- 3 Próximos conteúdos
- 4 Referências

# Referências sugeridas

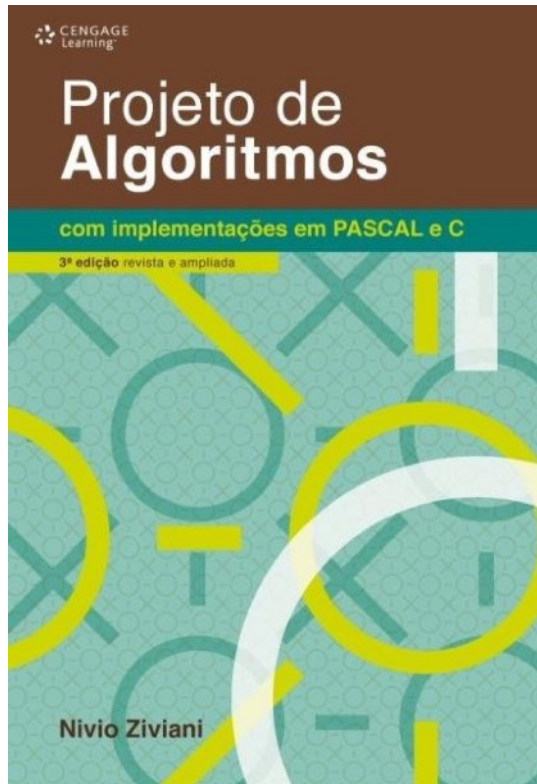


[Cormen et al, 2018]



[Tenenbaum et al, 1995]

# Referências sugeridas



[Ziviani, 2010]



[Drozdek, 2017]

# Perguntas?

Prof. Rafael G. **Mantovani**

[rafaelmantovani@utfpr.edu.br](mailto:rafaelmantovani@utfpr.edu.br)