

# Árvores Rubro-Negras

Estrutura de Dados Avançada — QXD0015



UNIVERSIDADE  
FEDERAL DO CEARÁ  
CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz  
gomes.atilio@ufc.br

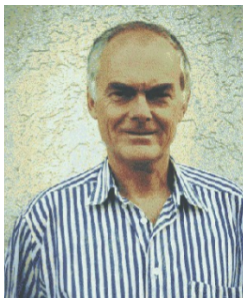
Universidade Federal do Ceará

1º semestre/2023



# Árvores rubro-negras (Red-Black trees)

- Originalmente criada por Rudolf Bayer em 1972.
  - Chamadas de Árvores Binárias Simétricas.
- Adquiriu o seu nome atual em um trabalho de Leonidas J. Guibas e Robert Sedgwick, em 1978.



R. Bayer



R. Sedgwick

# Árvores rubro-negras na prática

- Árvores rubro-negras são utilizadas em diversas aplicações e bibliotecas de linguagens de programação.
- Exemplos:
  - **Java:** `java.util.TreeMap`, `java.util.TreeSet`
  - **C++ STL:** `map`, `multiset`
  - **Linux kernel:** `completely fair scheduler`, `linux/rbtree.h`

# Árvores rubro-negras

- Uma **árvore rubro-negra** é uma árvore binária de busca.
- Cada nó de uma árvore rubro negra tem os seguintes campos:
  - **color** – Indica se o nó é **vermelho** ou **preto**.
  - **key** – Campo chave. Cada nó tem uma chave única.
  - **right** – Subárvore direita.
  - **left** – Subárvore esquerda.
  - **p** – Ponteiro para o pai do nó.

# Árvores rubro-negras

- Uma **árvore rubro-negra** é uma árvore binária de busca.
- Cada nó de uma árvore rubro negra tem os seguintes campos:
  - **color** – Indica se o nó é **vermelho** ou **preto**.
  - **key** – Campo chave. Cada nó tem uma chave única.
  - **right** – Subárvore direita.
  - **left** – Subárvore esquerda.
  - **p** – Ponteiro para o pai do nó.
- Se o filho ou o pai de um nó não existir, o campo aponta para um nó especial chamado **NIL** que possui cor **preta**.

# Árvores rubro-negras

- Uma **árvore rubro-negra** é uma árvore binária de busca.
- Cada nó de uma árvore rubro negra tem os seguintes campos:
  - **color** – Indica se o nó é **vermelho** ou **preto**.
  - **key** – Campo chave. Cada nó tem uma chave única.
  - **right** – Subárvore direita.
  - **left** – Subárvore esquerda.
  - **p** – Ponteiro para o pai do nó.
- Se o filho ou o pai de um nó não existir, o campo aponta para um nó especial chamado **NIL** que possui cor **preta**.
- Toda folha da árvore é um nó **NIL** e é também chamado **nó externo**. Os demais nós são chamados **nós internos**.

# Árvores rubro-negras

Uma árvore rubro-negra também satisfaz as seguintes **propriedades**:

# Árvores rubro-negras

Uma árvore rubro-negra também satisfaz as seguintes **propriedades**:

(P1) Todo nó da árvore ou é **vermelho** ou é **preto**.



# Árvores rubro-negras

Uma árvore rubro-negra também satisfaz as seguintes **propriedades**:

(P1) Todo nó da árvore ou é **vermelho** ou é **preto**.

(P2) A raiz é **preta**.

# Árvores rubro-negras

Uma árvore rubro-negra também satisfaz as seguintes **propriedades**:

(P1) Todo nó da árvore ou é **vermelho** ou é **preto**.

(P2) A raiz é **preta**.

(P3) Toda folha (NIL) é **preta**.

# Árvores rubro-negras

Uma árvore rubro-negra também satisfaz as seguintes **propriedades**:

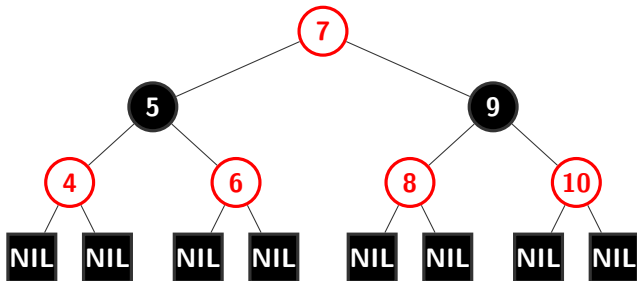
- (P1) Todo nó da árvore ou é **vermelho** ou é **preto**.
- (P2) A raiz é **preta**.
- (P3) Toda folha (NIL) é **preta**.
- (P4) Se um nó é **vermelho**, então ambos os filhos são **pretos**.

# Árvores rubro-negras

Uma árvore rubro-negra também satisfaz as seguintes **propriedades**:

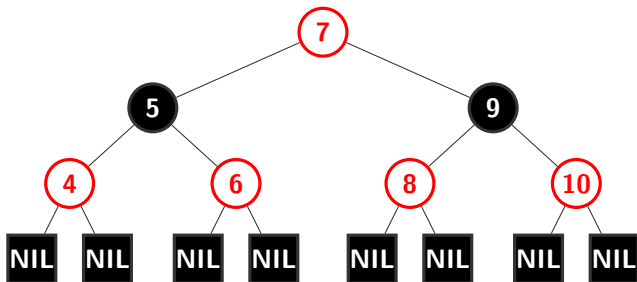
- (P1) Todo nó da árvore ou é **vermelho** ou é **preto**.
- (P2) A raiz é **preta**.
- (P3) Toda folha (NIL) é **preta**.
- (P4) Se um nó é **vermelho**, então ambos os filhos são **pretos**.
- (P5) Para todo nó, todos os caminhos do nó até as folhas descendentes contêm o **mesmo número de nós pretos**.

# Árvores rubro-negras – Reconhecendo 1



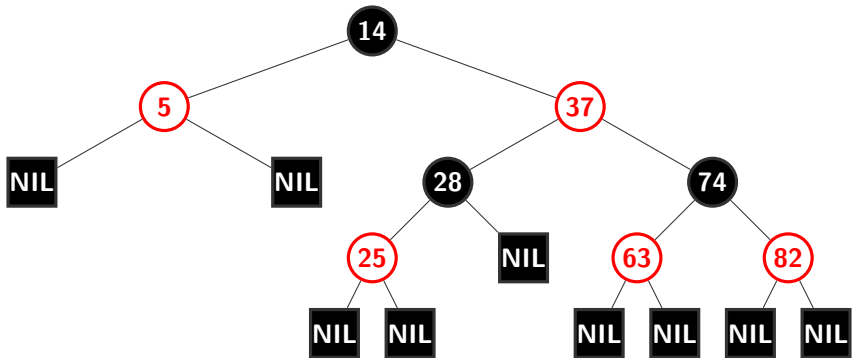
- É uma árvore rubro-negra?

# Árvores rubro-negras – Reconhecendo 1



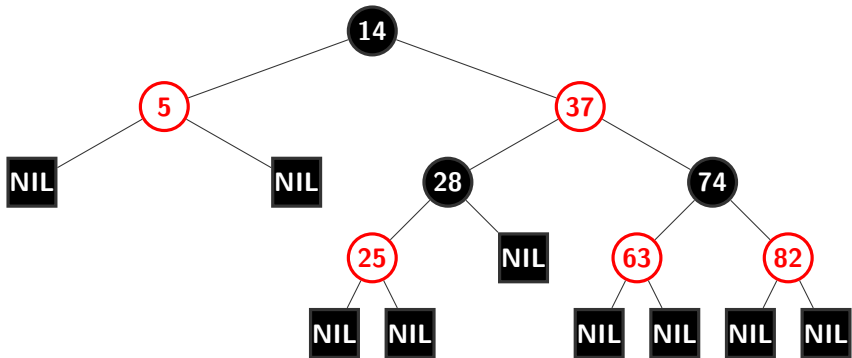
- É uma árvore rubro-negra?
- Não. Viola a Propriedade (2).

# Árvores rubro-negras – Reconhecendo 2



- É uma árvore rubro-negra?

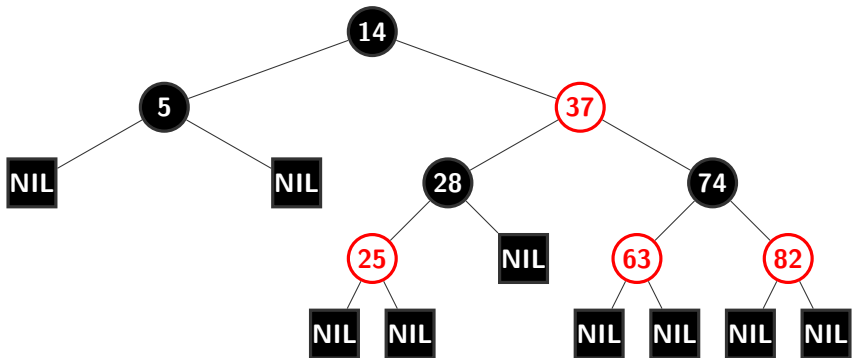
# Árvores rubro-negras – Reconhecendo 2



- É uma árvore rubro-negra?
- Não. Viola a Propriedade (5).

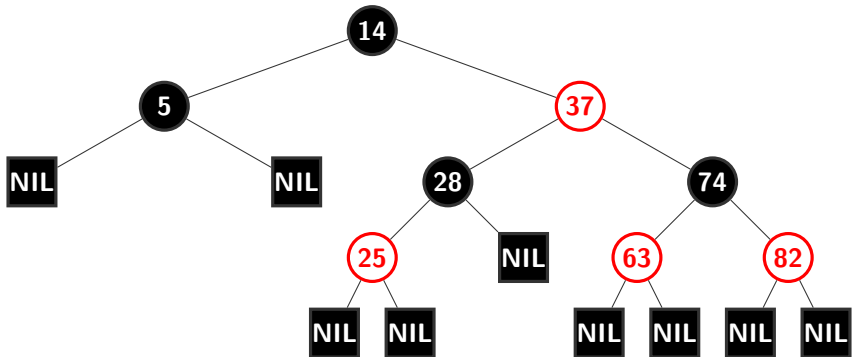


# Árvores rubro-negras – Reconhecendo 3



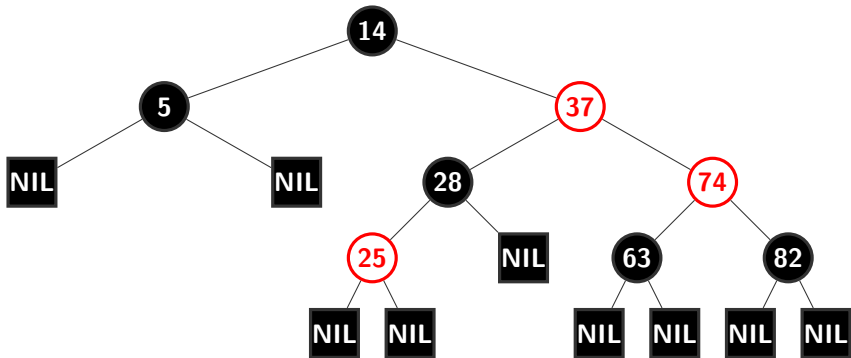
- É uma árvore rubro-negra?

# Árvores rubro-negras – Reconhecendo 3



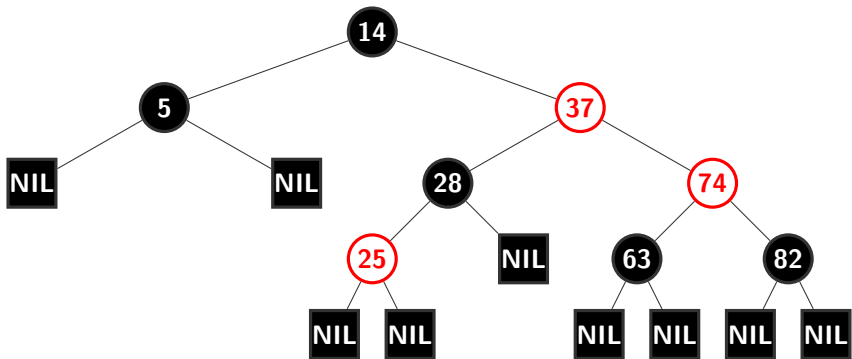
- É uma árvore rubro-negra?
- **SIM!!!**

# Árvores rubro-negras – Reconhecendo 4



- É uma árvore rubro-negra?

# Árvores rubro-negras – Reconhecendo 4



- É uma árvore rubro-negra?
- **NÃO**. Viola a Propriedade (4).

## Ideia por trás da definição

Qual a intuição por trás das propriedades da árvore rubro-negra? Como elas levam ao balanceamento?

## Ideia por trás da definição

Qual a intuição por trás das propriedades da árvore rubro-negra? Como elas levam ao balanceamento?

- Restringindo a maneira com que os nós podem ser coloridos do caminho da raiz até qualquer uma das suas folhas, as árvores rubro-negras:

## Ideia por trás da definição

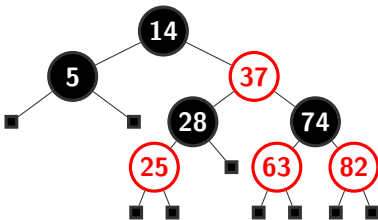
Qual a intuição por trás das propriedades da árvore rubro-negra? Como elas levam ao balanceamento?

- Restringindo a maneira com que os nós podem ser coloridos do caminho da raiz até qualquer uma das suas folhas, as árvores rubro-negras:
  - Garantem que nenhum dos caminhos será maior que **2 vezes** o comprimento de qualquer outro.

## Ideia por trás da definição

Qual a intuição por trás das propriedades da árvore rubro-negra? Como elas levam ao balanceamento?

- Restringindo a maneira com que os nós podem ser coloridos do caminho da raiz até qualquer uma das suas folhas, as árvores rubro-negras:
  - Garantem que nenhum dos caminhos será maior que **2 vezes** o comprimento de qualquer outro.
  - Garantem que a árvore é balanceada ( $h = O(\lg n)$ ).



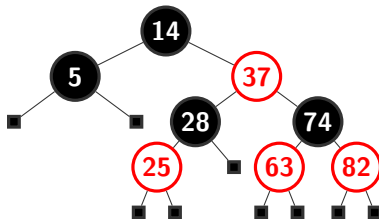


# Prova do balanceamento



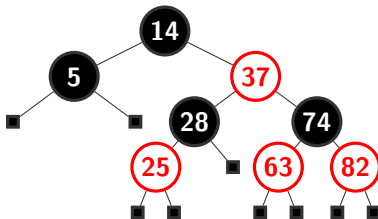
## Definição — Altura Negra

- A **altura negra** de um nó  $v$  é definida como o número de nós pretos visitados em qualquer caminho de  $v$  até as folhas (sem incluir o próprio  $v$ ).



## Definição — Altura Negra

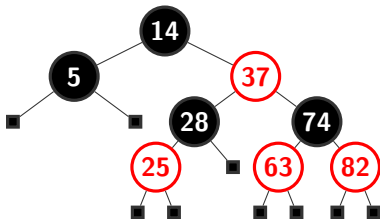
- A **altura negra** de um nó  $v$  é definida como o número de nós pretos visitados em qualquer caminho de  $v$  até as folhas (sem incluir o próprio  $v$ ).



- A altura negra do nó  $v$  é denotada por  $bh(v)$ .

## Definição — Altura Negra

- A **altura negra** de um nó  $v$  é definida como o número de nós pretos visitados em qualquer caminho de  $v$  até as folhas (sem incluir o próprio  $v$ ).



- A altura negra do nó  $v$  é denotada por  $bh(v)$ .
- Pela Propriedade (5),  $bh(v)$  é bem-definida para todo nó  $v$  da árvore. A altura negra da árvore rubro-negra é definida como sendo  $bh(root)$ .

# Lemas sobre árvores rubro-negras

## Lema 1

Seja  $v$  um nó com altura  $h$ . Então,  $bh(v) \geq \frac{h-1}{2}$ .

**Atenção:** a altura de um nó  $v$  é o número de nós no caminho de  $v$  até uma de suas folhas descendentes no nível de maior profundidade.

# Lemas sobre árvores rubro-negras

Demonstração do Lema 1:

# Lemas sobre árvores rubro-negras

## Demonstração do Lema 1:

Seja  $v$  um nó com altura  $h(v)$ .

Pela **Propriedade 5**, todos os caminhos de  $v$  até suas folhas descendentes contém o mesmo número de nós pretos.

# Lemas sobre árvores rubro-negras

## Demonstração do Lema 1:

Seja  $v$  um nó com altura  $h(v)$ .

Pela **Propriedade 5**, todos os caminhos de  $v$  até suas folhas descendentes contém o mesmo número de nós pretos.

Note que o maior caminho possível saindo de  $v$  até uma folha descendente deve intercalar nós vermelhos e nós pretos (**Propriedade 4**).



# Lemas sobre árvores rubro-negras

## Demonstração do Lema 1:

Seja  $v$  um nó com altura  $h(v)$ .

Pela **Propriedade 5**, todos os caminhos de  $v$  até suas folhas descendentes contém o mesmo número de nós pretos.

Note que o maior caminho possível saindo de  $v$  até uma folha descendente deve intercalar nós vermelhos e nós pretos (**Propriedade 4**).

**Caso 1:**  $v$  é vermelho. Neste caso, no maior caminho possível de  $v$  até uma de suas folhas descendentes, todo nó vermelho tem um filho preto, inclusive o próprio nó  $v$ . Assim,  $h(v) \leq 2 \cdot bh(v)$ .

# Lemas sobre árvores rubro-negras

## Demonstração do Lema 1:

Seja  $v$  um nó com altura  $h(v)$ .

Pela **Propriedade 5**, todos os caminhos de  $v$  até suas folhas descendentes contém o mesmo número de nós pretos.

Note que o maior caminho possível saindo de  $v$  até uma folha descendente deve intercalar nós vermelhos e nós pretos (**Propriedade 4**).

**Caso 1:**  $v$  é vermelho. Neste caso, no maior caminho possível de  $v$  até uma de suas folhas descendentes, todo nó vermelho tem um filho preto, inclusive o próprio nó  $v$ . Assim,  $h(v) \leq 2 \cdot bh(v)$ .

**Caso 2:**  $v$  é preto. Neste caso, no maior caminho possível de  $v$  até uma de suas folhas descendentes, todo nó vermelho tem um filho preto e o filho de  $v$  nesse caminho, chame-o  $w$ , é vermelho. Assim,  $h(v) \leq 2 \cdot bh(w) + 1 = 2 \cdot bh(v) + 1$ .

# Lemas sobre árvores rubro-negras

## Demonstração do Lema 1:

Seja  $v$  um nó com altura  $h(v)$ .

Pela **Propriedade 5**, todos os caminhos de  $v$  até suas folhas descendentes contém o mesmo número de nós pretos.

Note que o maior caminho possível saindo de  $v$  até uma folha descendente deve intercalar nós vermelhos e nós pretos (**Propriedade 4**).

**Caso 1:**  $v$  é vermelho. Neste caso, no maior caminho possível de  $v$  até uma de suas folhas descendentes, todo nó vermelho tem um filho preto, inclusive o próprio nó  $v$ . Assim,  $h(v) \leq 2 \cdot bh(v)$ .

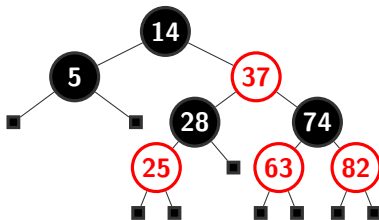
**Caso 2:**  $v$  é preto. Neste caso, no maior caminho possível de  $v$  até uma de suas folhas descendentes, todo nó vermelho tem um filho preto e o filho de  $v$  nesse caminho, chame-o  $w$ , é vermelho. Assim,  $h(v) \leq 2 \cdot bh(w) + 1 = 2 \cdot bh(v) + 1$ .

Dos Casos 1 e 2, concluímos que  $h(v) \leq 2 \cdot bh(v) + 1 \implies \frac{h-1}{2} \leq bh(v)$ .  $\square$

# Lemas sobre árvores rubro-negras

## Lema 2

A subárvore enraizada em um nó  $x$  qualquer contém pelo menos  $2^{bh(x)} - 1$  nós internos.



# Lemas sobre árvores rubro-negras

## Lema 2

A subárvore enraizada em um nó  $x$  qualquer contém pelo menos  $2^{bh(x)} - 1$  nós internos.

# Lemas sobre árvores rubro-negras

## Lema 2

A subárvore enraizada em um nó  $x$  qualquer contém pelo menos  $2^{bh(x)} - 1$  nós internos.

Demonstração:

# Lemas sobre árvores rubro-negras

## Lema 2

A subárvore enraizada em um nó  $x$  qualquer contém pelo menos  $2^{bh(x)} - 1$  nós internos.

### Demonstração:

- Por indução na altura  $h$  do vértice  $x$ .

# Lemas sobre árvores rubro-negras

## Lema 2

A subárvore enraizada em um nó  $x$  qualquer contém pelo menos  $2^{bh(x)} - 1$  nós internos.

### Demonstração:

- Por indução na altura  $h$  do vértice  $x$ .
- **Caso base:** Se a altura  $h$  de  $x$  é 1, então  $x$  é um nó externo (NIL). Então, a árvore enraizada em  $x$  contém pelo menos  $2^{bh(x)} - 1 = 2^0 - 1 = 0$  nós internos.



# Conclusão da demonstração

- **Passo indutivo:** Considere um nó  $x$  com altura  $h > 1$ . Note que  $x$  é um nó interno e possui dois filhos.

# Conclusão da demonstração

- **Passo indutivo:** Considere um nó  $x$  com altura  $h > 1$ . Note que  $x$  é um nó interno e possui dois filhos.
- Cada filho de  $x$  tem altura negra igual a  $bh(x)$  ou  $bh(x) - 1$ , dependendo se sua cor é **vermelha** ou **preta**, respectivamente.

# Conclusão da demonstração

- **Passo indutivo:** Considere um nó  $x$  com altura  $h > 1$ . Note que  $x$  é um nó interno e possui dois filhos.
- Cada filho de  $x$  tem altura negra igual a  $bh(x)$  ou  $bh(x) - 1$ , dependendo se sua cor é **vermelha** ou **preta**, respectivamente.
- Como a altura de um filho de  $x$  é menor que a altura de  $x$ , pela **hipótese de indução**, cada filho de  $x$  tem pelo menos  $2^{bh(x)-1} - 1$  nós internos.

## Conclusão da demonstração

- **Passo indutivo:** Considere um nó  $x$  com altura  $h > 1$ . Note que  $x$  é um nó interno e possui dois filhos.
- Cada filho de  $x$  tem altura negra igual a  $bh(x)$  ou  $bh(x) - 1$ , dependendo se sua cor é **vermelha** ou **preta**, respectivamente.
- Como a altura de um filho de  $x$  é menor que a altura de  $x$ , pela **hipótese de indução**, cada filho de  $x$  tem pelo menos  $2^{bh(x)-1} - 1$  nós internos.
- Seja  $n$  o número de nós internos da subárvore enraizada em  $x$ . Então,

$$n \geq (2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1.$$

e o resultado segue. ■

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2\lg(n + 1) + 1$ .

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n + 1) + 1$ .

Demonstração:

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2\lg(n+1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2\lg(n+1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .



# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n + 1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .
- Logo, pelo **Lema 2**, temos que  $n \geq 2^{(h-1)/2} - 1$ .

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n+1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .
- Logo, pelo **Lema 2**, temos que  $n \geq 2^{(h-1)/2} - 1$ .
- $n \geq 2^{(h-1)/2} - 1 \implies$

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n+1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .
- Logo, pelo **Lema 2**, temos que  $n \geq 2^{(h-1)/2} - 1$ .
- $n \geq 2^{(h-1)/2} - 1 \implies n + 1 \geq 2^{(h-1)/2} \implies$

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n+1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .
- Logo, pelo **Lema 2**, temos que  $n \geq 2^{(h-1)/2} - 1$ .
- $n \geq 2^{(h-1)/2} - 1 \implies n + 1 \geq 2^{(h-1)/2} \implies \lg(n+1) \geq \lg 2^{(h-1)/2}$   
 $\implies$

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n+1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .
- Logo, pelo **Lema 2**, temos que  $n \geq 2^{(h-1)/2} - 1$ .
- $n \geq 2^{(h-1)/2} - 1 \implies n+1 \geq 2^{(h-1)/2} \implies \lg(n+1) \geq \lg 2^{(h-1)/2} \implies \lg(n+1) \geq \frac{h-1}{2} \implies$

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n+1) + 1$ .

### Demonstração:

- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .
- Logo, pelo **Lema 2**, temos que  $n \geq 2^{(h-1)/2} - 1$ .
- $n \geq 2^{(h-1)/2} - 1 \implies n+1 \geq 2^{(h-1)/2} \implies \lg(n+1) \geq \lg 2^{(h-1)/2} \implies \lg(n+1) \geq \frac{h-1}{2} \implies h-1 \leq 2 \lg(n+1) \implies$

# Árvores rubro-negras são balanceadas

## Teorema 3

A altura de uma árvore rubro-negra com  $n$  nós internos é no máximo  $2 \lg(n+1) + 1$ .

### Demonstração:

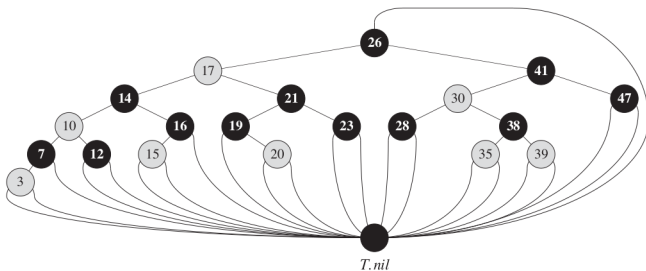
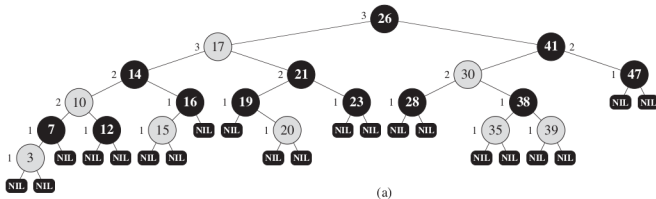
- Seja  $T$  uma árvore rubro-negra com altura  $h$ , com  $n$  nós, e seja  $v$  sua raiz.
- Pelo **Lema 1**,  $bh(v) \geq \frac{h-1}{2}$ .
- Logo, pelo **Lema 2**, temos que  $n \geq 2^{(h-1)/2} - 1$ .
- $n \geq 2^{(h-1)/2} - 1 \implies n+1 \geq 2^{(h-1)/2} \implies \lg(n+1) \geq \lg 2^{(h-1)/2} \implies \lg(n+1) \geq \frac{h-1}{2} \implies h-1 \leq 2 \lg(n+1) \implies h \leq 2 \lg(n+1) + 1$ . ■

## Corolário 4

As operações de Busca, Mínimo, Máximo, Sucessor e Predecessor podem ser efetuadas em tempo  $O(\lg(n))$  em uma árvore rubro-negra.  $\square$



# O nó T.NIL



Vamos considerar que cada **NIL** é substituído por um único nó chamado **T.NIL**, que sempre tem cor preta. O nó **T.NIL** também é o pai de **T.root**

# Rotações



# Rotações

- Antes de vermos como fazer inserções em uma árvore rubro-negra, é preciso apresentar o conceito de **rotações**.

# Rotações

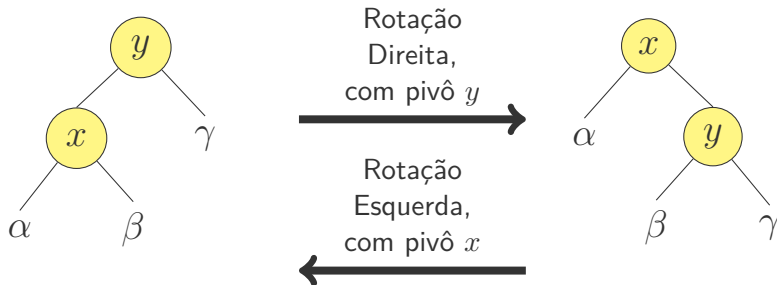
- Antes de vermos como fazer inserções em uma árvore rubro-negra, é preciso apresentar o conceito de **rotações**.
- Usamos as rotações para consertar **parte do estrago feito** pelas operações já conhecidas de inserção e remoção nas propriedades rubro-negras.

- Antes de vermos como fazer inserções em uma árvore rubro-negra, é preciso apresentar o conceito de **rotações**.
- Usamos as rotações para consertar **parte do estrago feito** pelas operações já conhecidas de inserção e remoção nas propriedades rubro-negras.
- O resto do estrago é consertado utilizando **recoloração de nós**.

# Rotações

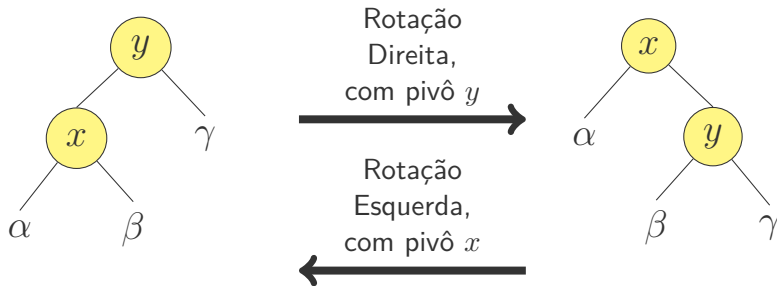
- Antes de vermos como fazer inserções em uma árvore rubro-negra, é preciso apresentar o conceito de **rotações**.
- Usamos as rotações para consertar **parte do estrago feito** pelas operações já conhecidas de inserção e remoção nas propriedades rubro-negras.
- O resto do estrago é consertado utilizando **recoloração de nós**.
- Rotações são operações **locais**: alteram um número **pequeno e constante de ponteiros**.

# Rotações Esquerda e Direita



- As letras  $\alpha$ ,  $\beta$ ,  $\gamma$  representam subárvores quaisquer, que podem ou não ser vazias.

# Rotações Esquerda e Direita



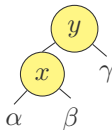
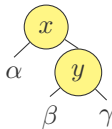
- As letras  $\alpha$ ,  $\beta$ ,  $\gamma$  representam subárvores quaisquer, que podem ou não ser vazias.
- Note que as duas rotações preservam a propriedade da árvore binária de busca.



# Rotação à esquerda — Pseudocódigo

## LEFT-ROTATE( $T, x$ )

```
1   $y = x.right$ 
2   $x.right = y.left$ 
3  if  $y.left \neq T.NIL$ 
4       $y.left.p = x$ 
5   $y.p = x.p$ 
6  if  $x.p == T.NIL$ 
7       $T.root = y$ 
8  elseif  $x == x.p.left$ 
9       $x.p.left = y$ 
10 else
11      $x.p.right = y$ 
12  $y.left = x$ 
13  $x.p = y$ 
```



Supomos que  $x \rightarrow right \neq T.NIL$  e que  $T.root.p == T.NIL$ .

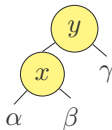
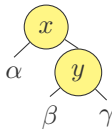
# Rotação à esquerda — Pseudocódigo

## LEFT-ROTATE( $T, x$ )

```

1  y = x.right
2  x.right = y.left
3  if y.left  $\neq$  T.NIL
4      y.left.p = x
5  y.p = x.p
6  if x.p == T.NIL
7      T.root = y
8  elseif x == x.p.left
9      x.p.left = y
10 else
11     x.p.right = y
12  y.left = x
13  x.p = y

```



Supomos que  $x \rightarrow \text{right} \neq \text{T.NIL}$  e que  $\text{T.root.p} == \text{T.NIL}$ .

- O pseudocódigo para **Right-Rotate( $T, y$ )** é simétrico e é deixado como exercício.

# Inserção



# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.

# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.

# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.
- Em uma árvore rubro-negra todos os nós estão equilibrados.

# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.
- Em uma árvore rubro-negra todos os nós estão equilibrados.

# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.
- Em uma árvore rubro-negra todos os nós estão equilibrados.
- Uma consequência direta das propriedades é que em qualquer caminho da raiz até uma folha **não existem dois nós vermelhos consecutivos**.



# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.
- Em uma árvore rubro-negra todos os nós estão equilibrados.
- Uma consequência direta das propriedades é que em qualquer caminho da raiz até uma folha **não existem dois nós vermelhos consecutivos**.

# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.
- Em uma árvore rubro-negra todos os nós estão equilibrados.
- Uma consequência direta das propriedades é que em qualquer caminho da raiz até uma folha **não existem dois nós vermelhos consecutivos**.
- Cada vez que uma operação de inserção/remoção for realizada na árvore, o conjunto de propriedades é **verificado em busca de violações**.

# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.
- Em uma árvore rubro-negra todos os nós estão equilibrados.
- Uma consequência direta das propriedades é que em qualquer caminho da raiz até uma folha **não existem dois nós vermelhos consecutivos**.
- Cada vez que uma operação de inserção/remoção for realizada na árvore, o conjunto de propriedades é **verificado em busca de violações**.

# Visão geral da inserção

- Um nó que satisfaz as regras de uma árvore rubro-negra é denominado **equilibrado**, caso contrário é dito **desequilibrado**.
- Em uma árvore rubro-negra todos os nós estão equilibrados.
- Uma consequência direta das propriedades é que em qualquer caminho da raiz até uma folha **não existem dois nós vermelhos consecutivos**.
- Cada vez que uma operação de inserção/remoção for realizada na árvore, o conjunto de propriedades é **verificado em busca de violações**.
- Caso alguma propriedade tenha sido violada, **realizamos rotações e ajustamos as cores** dos nós para que todas as propriedades continuem válidas.

# Inserções em árvores rubro-negras

- Árvores rubro-negras são árvores de busca binária com propriedades adicionais:

(P1) Um nó é **vermelho** ou é **preto**.

(P2) A raiz é **preta**.

(P3) Toda folha (NIL) é **preta**.

(P4) Se um nó é **vermelho** então ambos os seus filhos são **pretos**.

(P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos.

# Inserções em árvores rubro-negras

- Árvores rubro-negras são árvores de busca binária com propriedades adicionais:

(P1) Um nó é **vermelho** ou é **preto**.

(P2) A raiz é **preta**.

(P3) Toda folha (NIL) é **preta**.

(P4) Se um nó é **vermelho** então ambos os seus filhos são **pretos**.

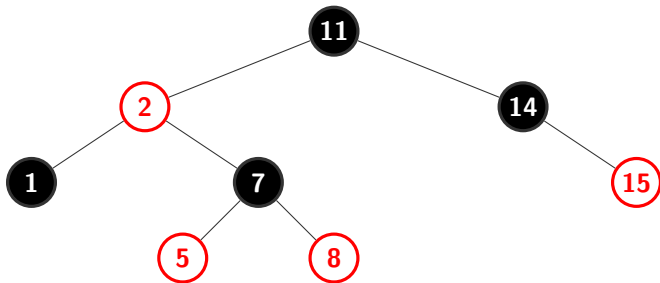
(P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos.

- Se utilizarmos o algoritmo que já conhecemos para a inserção em uma árvore rubro-negra, corremos o risco de **quebrar algumas dessas regras**.  
**Mas quais?**

# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

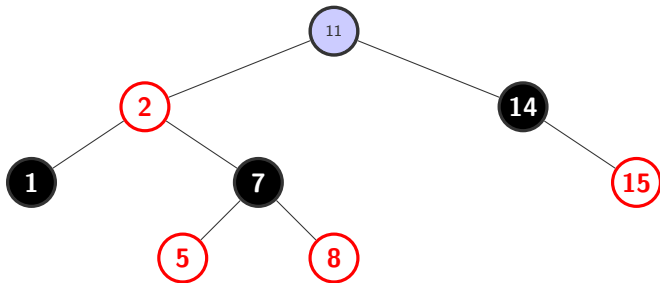
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

Exemplo: Inserindo a chave 4

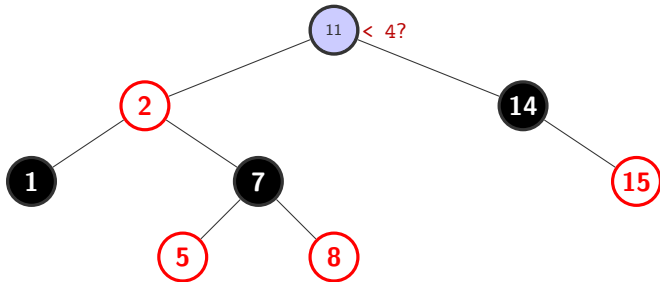




# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

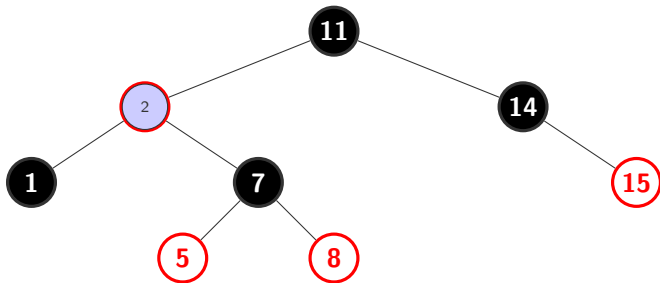
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

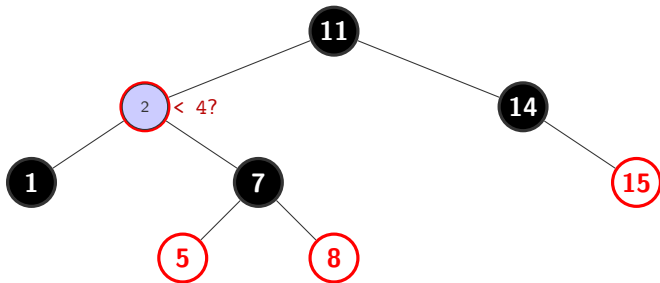
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

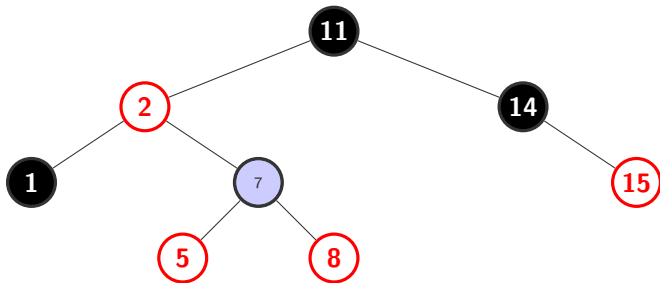
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

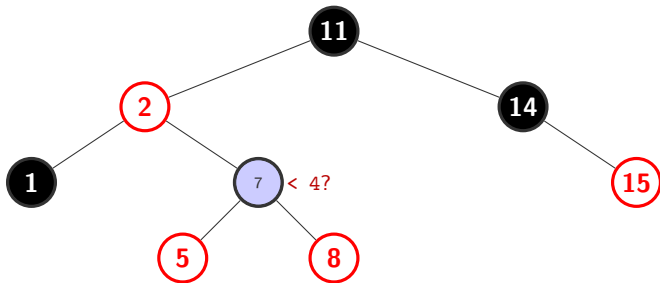
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

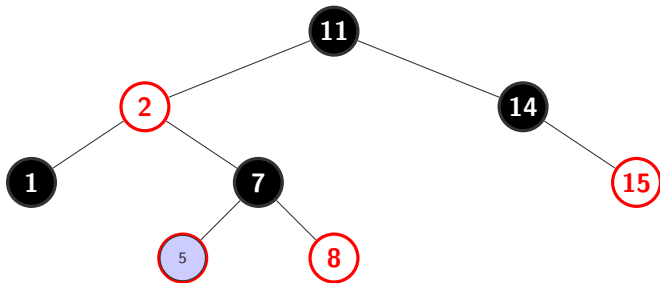
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

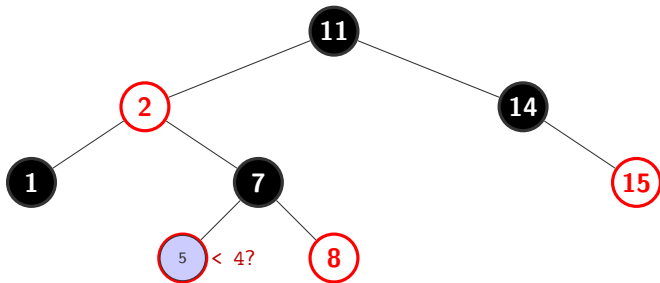
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

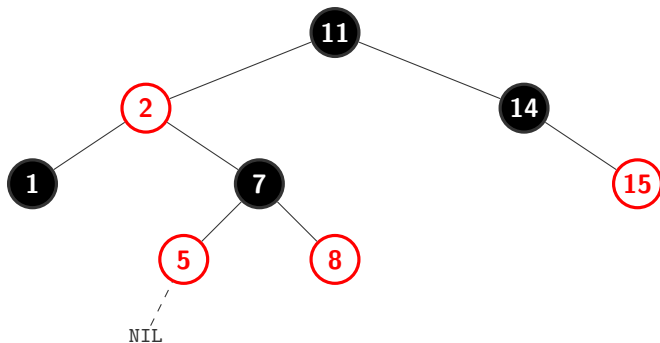
Exemplo: Inserindo a chave 4



# Inserções em árvores rubro-negras

Revisando a inserção em árvore binária de busca

Exemplo: Inserindo a chave 4

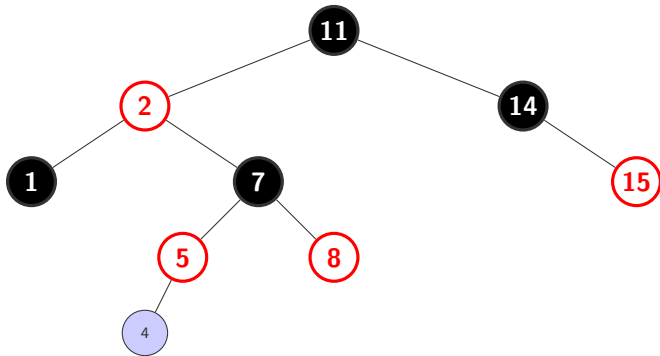




# Inserções em árvores rubro-negras

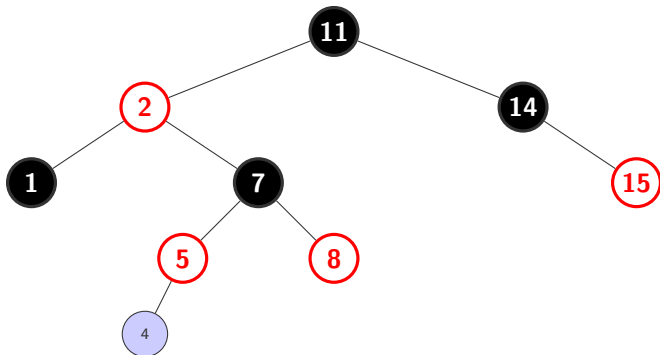
Revisando a inserção em árvore binária de busca

Exemplo: Inserindo a chave 4



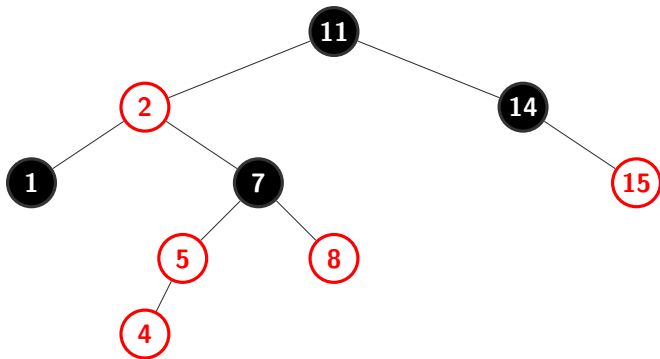
- Quebra P1 — Todo nó deve ser **vermelho** ou **preto**.

# Inserções em árvores rubro-negras



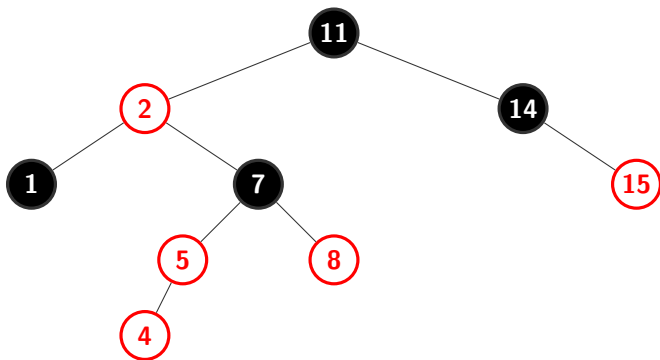
- É preciso decidir, qual faz menos mal, colocar um nó **vermelho** ou um **preto**?

# Inserções em árvores rubro-negras



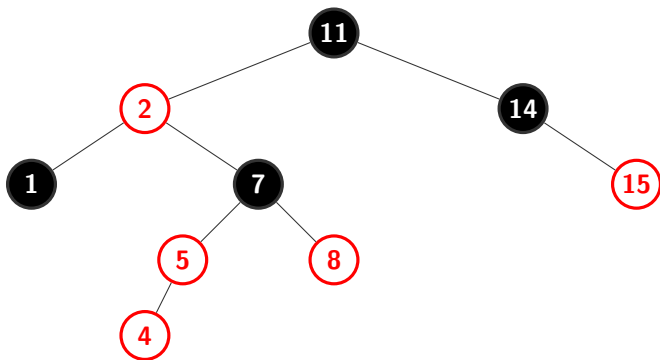
- É preciso decidir, qual faz menos mal, colocar um nó **vermelho** ou um **preto**?
  - Em algumas situações, o **vermelho** pode não quebrar nada.
  - O **preto** sempre vai desequilibrar a altura negra da raiz.

## Inserções em árvores rubro-negras



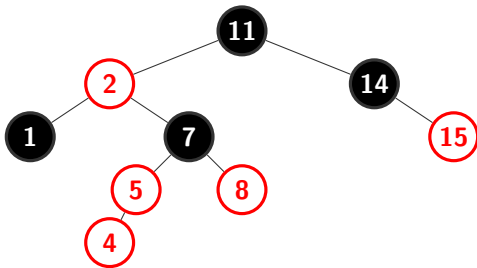
- Propriedade (1) satisfeita, **sempre insiro um nó com a cor vermelha.**

## Inserções em árvores rubro-negras



- Propriedade (1) satisfeita, **sempre insiro um nó com a cor vermelha.**
- E agora, **quais regras eu posso ter quebrado?**

# Avaliando quebras das propriedades



(P1) Um nó é **vermelho** ou é **preto**.

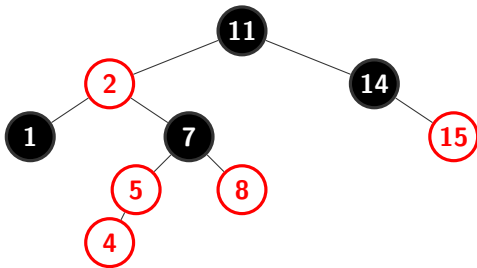
(P2) A raiz é **preta**.

(P3) Toda folha (NIL) é **preta**.

(P4) Se um nó é **vermelho** então ambos os seus filhos são **pretos**.

(P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos.

# Avaliando quebras das propriedades



(P1) Um nó é ~~vermelho~~ ou é **preto**.

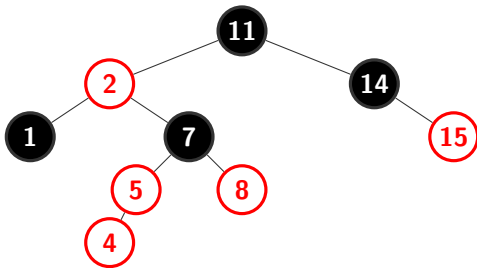
(P2) A raiz é **preta**.

(P3) Toda folha (NIL) é **preta**.

(P4) Se um nó é **vermelho** então ambos os seus filhos são **pretos**.

(P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos.

# Avaliando quebras das propriedades



(P1) Um nó é ~~vermelho~~ ou é **preto**.

(P2) A raiz é **preta**.

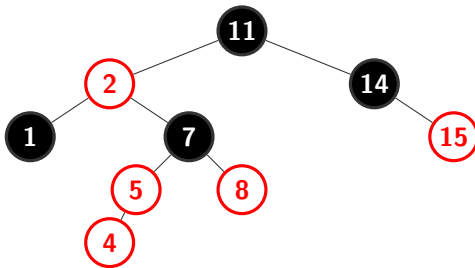
(P3) Toda folha ~~(NIL)~~ é **preta**.

(P4) Se um nó é **vermelho** então ambos os seus filhos são **pretos**.

(P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos.



# Avaliando quebras das propriedades



(P1) Um nó é ~~vermelho~~ ou é ~~preto~~.

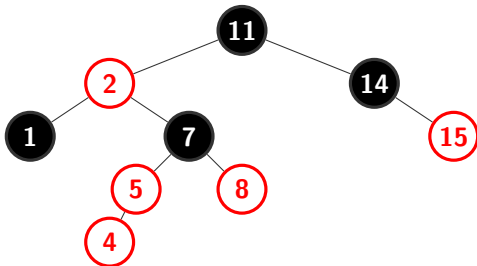
(P2) A raiz é **preta**.

(P3) Toda folha (NIL) é ~~preta~~.

(P4) Se um nó é **vermelho** então ambos os seus filhos são **pretos**.

(P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos.

# Avaliando quebras das propriedades



- Sabemos que podemos quebrar apenas as propriedades (P2) e (P4).
- Precisaremos de uma estratégia para recolorir os nós e rebalancear a árvore.
- Porém, já sabemos inserir o nó. Vamos ver o pseudocódigo da inserção!

# Inserindo um nó – Pseudocódigo

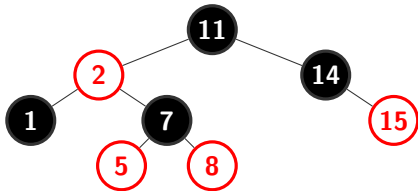
## RB-INSERT( $T, z$ )

```

1  x = T.root
2  y = T.NIL
3  while x  $\neq$  T.NIL
4      y = x
5      if z.key < x.key
6          x = x.left
7      elseif z.key > x.key
8          x = x.right
9      else
10         return
11  z.p = y
12  if y == T.NIL
13      T.root = z
14  elseif z.key < y.key
15      y.left = z
16  else
17      y.right = z
18  z.left = T.NIL
19  z.right = T.NIL
20  z.color = RED
21  RB-INSERT-FIXUP( $T, z$ )

```

**Obs.:** Esta função recebe um nó  $z$  como argumento e insere  $z$  na árvore  $T$ . O nó  $z$  foi criado fora desta função e já contém a chave a ser inserida na árvore.



## Resolvendo violação da Propriedade 2

Propriedade 2: A raiz é **preta**.

Raiz



4

## Resolvendo violação da Propriedade 2

Propriedade 2: A raiz é **preta**.



- Quando a raiz for vermelha, basta pintá-la de preto.



- Isto não quebra nenhuma outra propriedade. **Por quê?**

## Resolvendo violação da Propriedade 2

Propriedade 2: A raiz é **preta**.



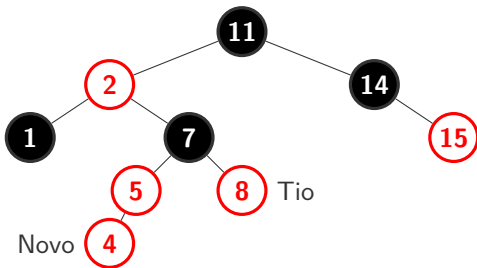
- Quando a raiz for vermelha, basta pintá-la de preto.



- Isto não quebra nenhuma outra propriedade. **Por quê?**
- Agora, vamos tentar garantir a **Propriedade 4**

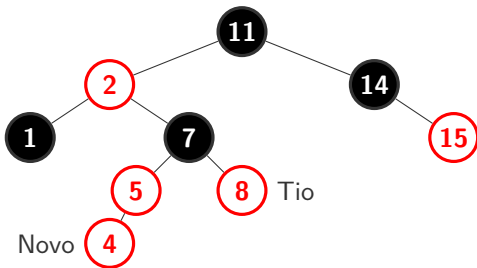
## Resolvendo violação da Propriedade 4

O pai do nó Novo é Vermelho. Então, Novo tem um avô e ele é preto.



## Resolvendo violação da Propriedade 4

O pai do nó Novo é **Vermelho**. Então, Novo tem um avô e ele é preto.



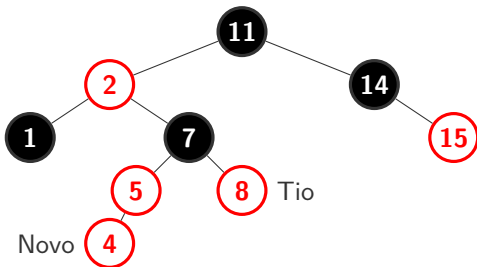
Existem 3 casos que precisamos tratar:

- **Caso 1:** O tio do nó Novo é **vermelho**.



## Resolvendo violação da Propriedade 4

O pai do nó Novo é **Vermelho**. Então, Novo tem um avô e ele é preto.

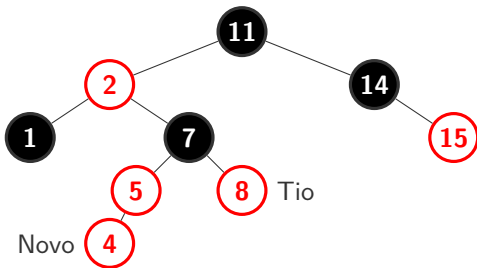


Existem 3 casos que precisamos tratar:

- **Caso 1:** O tio do nó Novo é **vermelho**.
- **Caso 2:** O tio do nó Novo é **preto** e Novo é filho da direita.

## Resolvendo violação da Propriedade 4

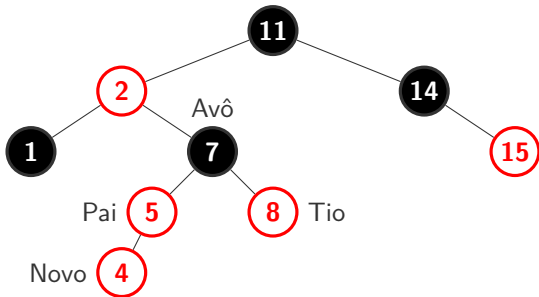
O pai do nó Novo é **Vermelho**. Então, Novo tem um avô e ele é preto.



Existem 3 casos que precisamos tratar:

- **Caso 1:** O tio do nó Novo é **vermelho**.
- **Caso 2:** O tio do nó Novo é **preto** e Novo é filho da direita.
- **Caso 3:** O tio do nó Novo é **preto** e Novo é filho da esquerda.

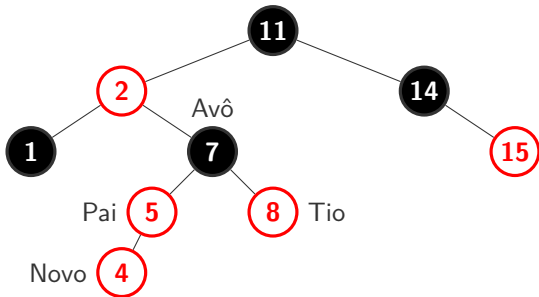
## Inserção — Caso 1



**Caso 1:** O tio do nó Novo é **vermelho**.

- Se o pai e o tio são vermelhos, então o avô é obrigatoriamente preto.

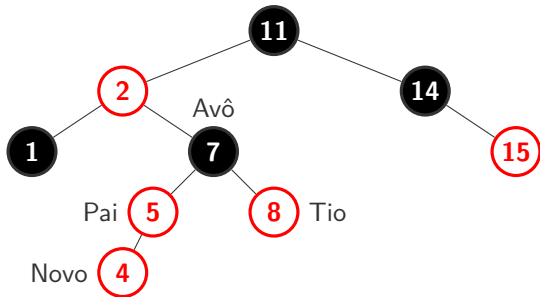
## Inserção — Caso 1



**Caso 1:** O tio do nó Novo é **vermelho**.

- Se o pai e o tio são vermelhos, então o avô é obrigatoriamente preto.
- Troque a cor do pai e do tio para **preto**.

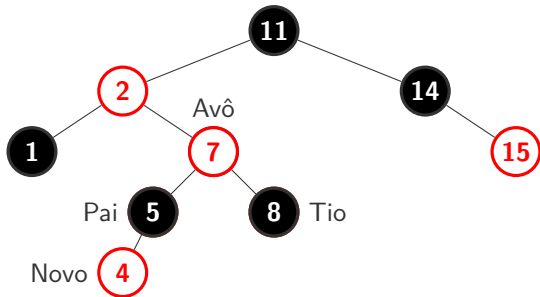
## Inserção — Caso 1



**Caso 1:** O tio do nó Novo é **vermelho**.

- Se o pai e o tio são vermelhos, então o avô é obrigatoriamente preto.
- Troque a cor do pai e do tio para **preto**.
- Troque a cor do avô para **vermelho**.

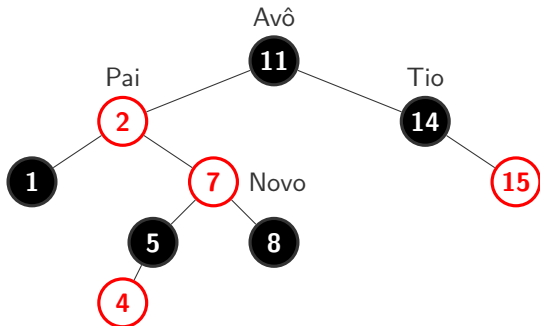
## Inserção — Caso 1



**Caso 1:** O tio do nó Novo é **vermelho**.

- Se o pai e o tio são vermelhos, então o avô é obrigatoriamente preto.
- Troque a cor do pai e do tio para **preto**.
- Troque a cor do avô para **vermelho**.
- Neste ponto, consertamos o problema do nó Novo, mas possivelmente estragamos o avô.

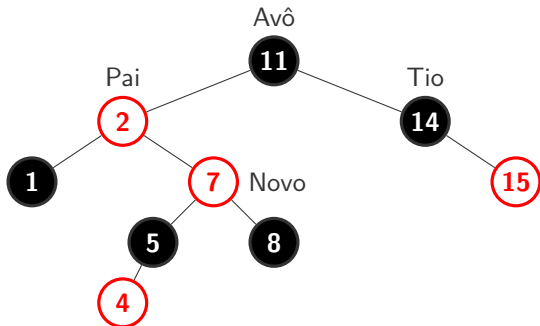
# Inserção em árvores rubro-negras



Continuamos o conserto, agora tomando como referência o antigo avô, o nó com o valor 7.

- **Caso 1:** O tio de Novo é **vermelho**.
- **Caso 2:** O tio de Novo é **preto** e Novo é filho da direita.
- **Caso 3:** O tio de Novo é **preto** e Novo é filho da esquerda.

# Inserção em árvores rubro-negras

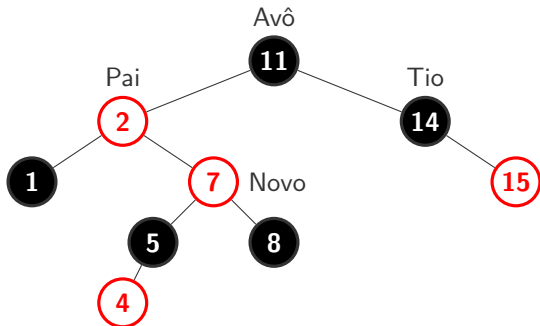


Continuamos o conserto, agora tomando como referência o antigo avô, o nó com o valor 7.

- ~~Caso 1: O tio de Novo é vermelho.~~
- Caso 2: O tio de Novo é **preto** e Novo é filho da direita.
- Caso 3: O tio de Novo é **preto** e Novo é filho da esquerda.



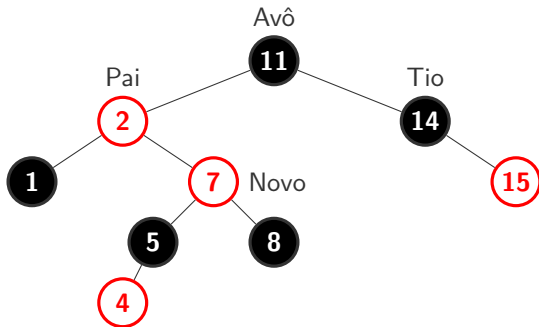
# Inserção em árvores rubro-negras



Continuamos o conserto, agora tomando como referência o antigo avô, o nó com o valor 7.

- ~~Caso 1: O tio de Novo é vermelho.~~
- Caso 2: O tio de Novo é **preto** e Novo é filho da direita.
- ~~Caso 3: O tio de Novo é preto e Novo é filho da esquerda.~~

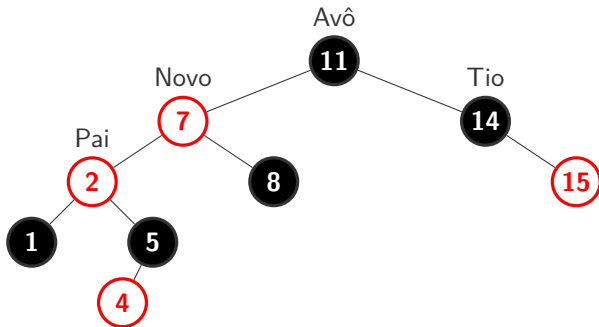
## Inserção — Caso 2



**Caso 2:** O tio de Novo é **preto** e Novo é filho da direita.

- Executa uma rotação à esquerda tendo como pivô o pai.

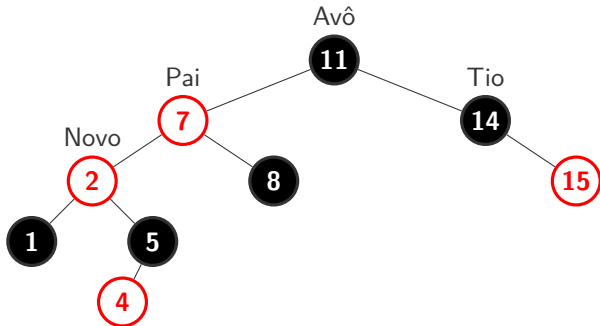
## Inserção — Caso 2



**Caso 2:** O tio de Novo é **preto** e Novo é filho da direita.

- Executa uma rotação à esquerda tendo como pivô o pai.
- Neste ponto, consertamos o problema no Novo, mas possivelmente estragamos o nó Pai.

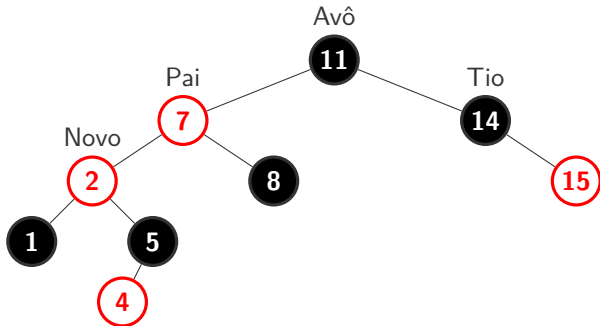
## Inserções em árvores rubro-negras



Continuamos o conserto, agora tomando como referência o antigo pai, o nó com a chave 2 na figura acima.

- **Caso 1:** O tio de Novo é **vermelho**.
- **Caso 2:** O tio de Novo é **preto** e Novo é filho da direita.
- **Caso 3:** O tio de Novo é **preto** e Novo é filho da esquerda.

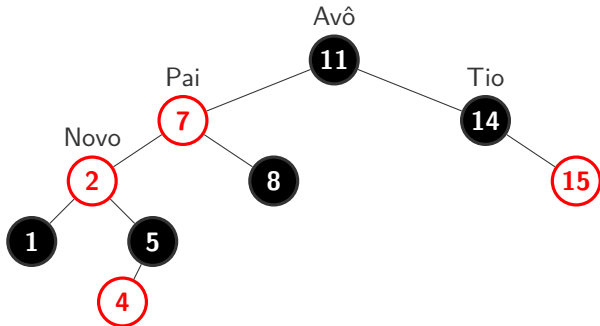
## Inserções em árvores rubro-negras



Continuamos o conserto, agora tomando como referência o antigo pai, o nó com a chave 2 na figura acima.

- ~~Caso 1: O tio de Novo é vermelho.~~
- Caso 2: O tio de Novo é **preto** e Novo é filho da direita.
- Caso 3: O tio de Novo é **preto** e Novo é filho da esquerda.

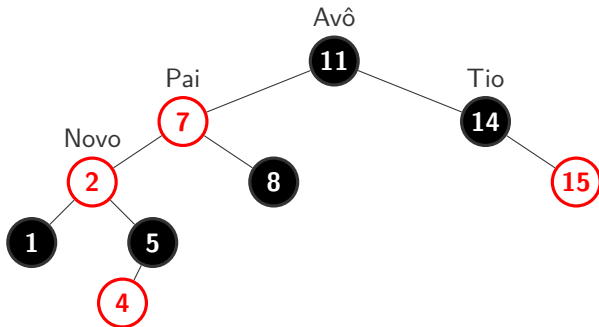
# Inserções em árvores rubro-negras



Continuamos o conserto, agora tomando como referência o antigo pai, o nó com a chave 2 na figura acima.

- ~~Caso 1: O tio de Novo é vermelho.~~
- ~~Caso 2: O tio de Novo é preto e Novo é filho da direita.~~
- ~~Caso 3: O tio de Novo é preto e Novo é filho da esquerda.~~

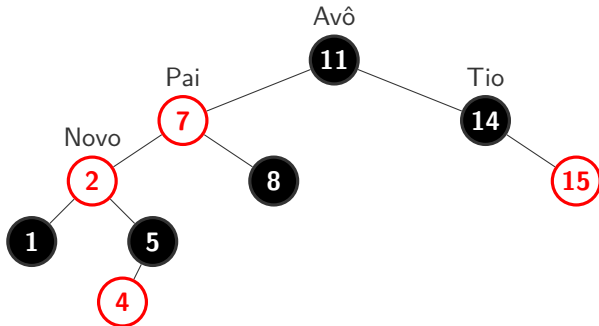
## Inserção — Caso 3



**Caso 3:** O tio de Novo é **preto** e Novo é filho da esquerda.

- Troca a cor do pai para **preto**.
- Troca a cor do avô para **vermelho**.
- Executa uma rotação à direita tendo como pivô o avô.
- Neste ponto a árvore voltou a ser uma árvore rubro-negra válida.

## Inserção — Caso 3

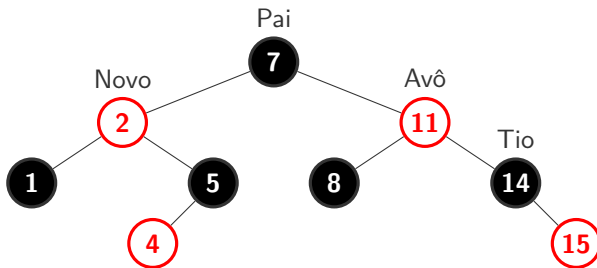


**Caso 3:** O tio de Novo é **preto** e Novo é filho da esquerda.

- Troca a cor do pai para **preto**.
- Troca a cor do avô para **vermelho**.
- Executa uma rotação à direita tendo como pivô o avô.
- Neste ponto a árvore voltou a ser uma árvore rubro-negra válida.



## Inserção — Caso 3



**Caso 3:** O tio de Novo é **preto** e Novo é filho da esquerda.

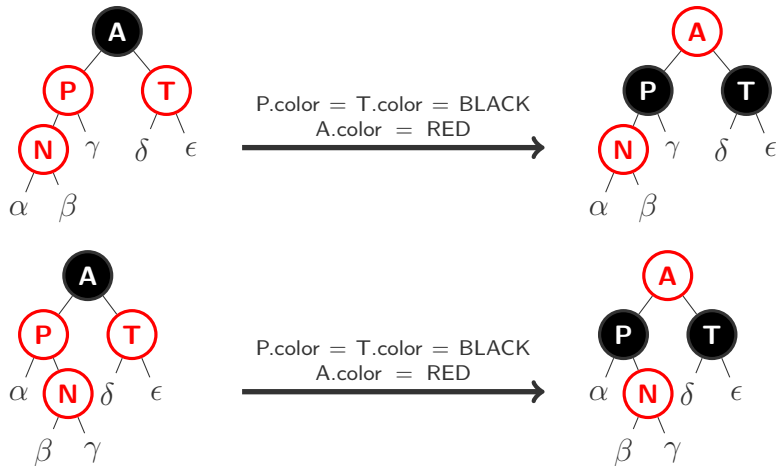
- Troca a cor do pai para **preto**.
- Troca a cor do avô para **vermelho**.
- Executa uma rotação à direita tendo como pivô o avô.
- Neste ponto a árvore voltou a ser uma árvore rubro-negra válida.

# Corretude do Caso 1

Vamos verificar que a recoloração realizada no Caso 1 mantém a altura negra da árvore!

- Vamos verificar apenas o caso em que o pai do nó Novo é filho esquerdo do avô de Novo.
- O caso em que o pai do nó Novo é filho direito do avô de Novo é simétrico.

## Corretude do Caso 1



**Propriedade 5** é mantida: Todos os caminhos saindo de um nó até uma de suas folhas possuem o mesmo número de nós pretos. Note que  $\alpha, \beta, \gamma, \delta, \epsilon$  possuem todos a mesma altura negra (antes e depois).

## Corretude dos Casos 2 e 3

O tio do nó inserido é preto

Vamos verificar que as rotações e recolorações realizadas nos Casos 2 e 3 mantêm a altura negra da árvore!

## Corretude dos Casos 2 e 3

O tio do nó inserido é preto

Vamos verificar que as rotações e recolorações realizadas nos Casos 2 e 3 mantêm a altura negra da árvore!

- Se o tio é preto, então existem 4 configurações possíveis:

## Corretude dos Casos 2 e 3

O tio do nó inserido é preto

Vamos verificar que as rotações e recolorações realizadas nos Casos 2 e 3 mantêm a altura negra da árvore!

- Se o tio é preto, então existem 4 configurações possíveis:
  1. Configuração **Esq-Dir** (pai é filho esquerdo do avô e Novo é filho direito de pai) — **Caso 2(a)**

## Corretude dos Casos 2 e 3

O tio do nó inserido é preto

Vamos verificar que as rotações e recolorações realizadas nos Casos 2 e 3 mantêm a altura negra da árvore!

- Se o tio é preto, então existem 4 configurações possíveis:
  1. Configuração **Esq-Dir** (pai é filho esquerdo do avô e Novo é filho direito de pai) — **Caso 2(a)**
  2. Configuração **Esq-Esq** (pai é filho esquerdo do avô e Novo é filho esquerdo de pai) — **Caso 3(a)**

## Corretude dos Casos 2 e 3

O tio do nó inserido é preto

Vamos verificar que as rotações e recolorações realizadas nos Casos 2 e 3 mantêm a altura negra da árvore!

- Se o tio é preto, então existem 4 configurações possíveis:
  1. Configuração **Esq-Dir** (pai é filho esquerdo do avô e Novo é filho direito de pai) — **Caso 2(a)**
  2. Configuração **Esq-Esq** (pai é filho esquerdo do avô e Novo é filho esquerdo de pai) — **Caso 3(a)**
  3. Configuração **Dir-Esq** (simétrico da 1) — **Caso 2(b)**



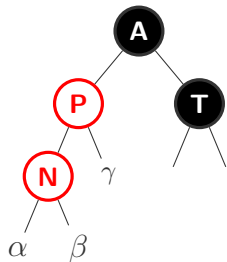
## Corretude dos Casos 2 e 3

O tio do nó inserido é preto

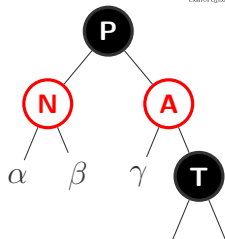
Vamos verificar que as rotações e recolorações realizadas nos Casos 2 e 3 mantêm a altura negra da árvore!

- Se o tio é preto, então existem 4 configurações possíveis:
  1. Configuração **Esq-Dir** (pai é filho esquerdo do avô e Novo é filho direito de pai) — **Caso 2(a)**
  2. Configuração **Esq-Esq** (pai é filho esquerdo do avô e Novo é filho esquerdo de pai) — **Caso 3(a)**
  3. Configuração **Dir-Esq** (simétrico da 1) — **Caso 2(b)**
  4. Configuração **Dir-Dir** (simétrico da 2) — **Caso 3(b)**

## Configuração Esq-Esq (Caso 3a)

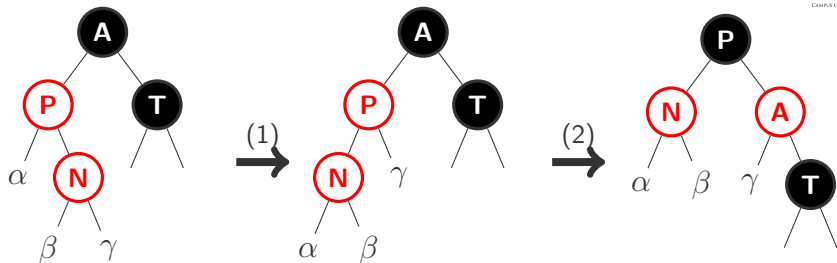


1. Rotação Direita, pivô A
2. Troca cores  $A \leftrightarrow P$



- A = Avô
- P = Pai
- T = Tio
- N = Novo
- $\alpha, \beta, \gamma$  são subárvores.
- As subárvores  $\alpha, \beta, \gamma$  e  $T$  possuem a mesma altura negra (antes e depois).

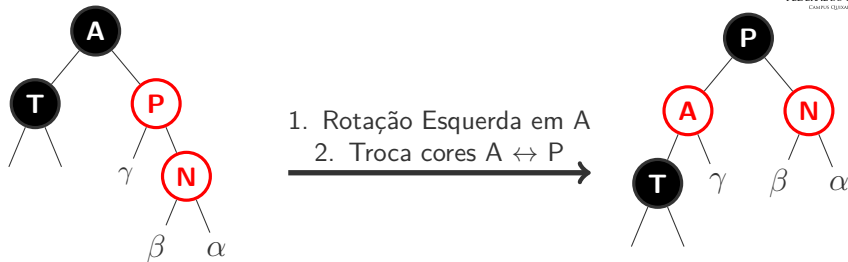
## Configuração Esq-Dir (Caso 2a + Caso 3a)



- A = Avô; P = Pai; T = Tio; N = Novo;
- $\alpha$ ,  $\beta$  e  $\gamma$  são subárvores.
- As subárvores  $\alpha$ ,  $\beta$ ,  $\gamma$  e  $T$  possuem a mesma altura negra (antes e depois).

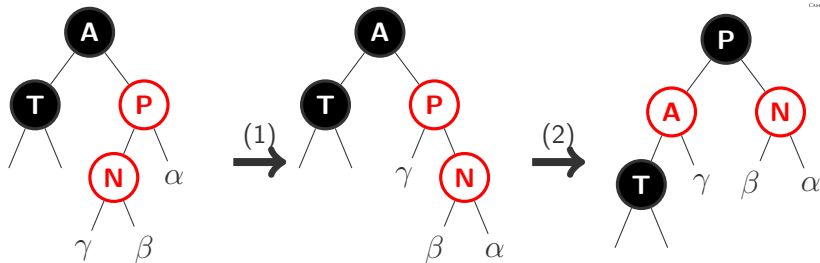
- (1) Efetua rotação à esquerda em torno do pai e troca  $P \leftrightarrow N$ .
- (2) Aplica a configuração **Esq-Dir** (Caso 3a).

## Configuração Dir-Dir (Caso 3b)



- A = Avô
- P = Pai
- T = Tio
- N = Novo
- $\alpha$ ,  $\beta$  e  $\gamma$  são subárvores.
- As subárvores  $\alpha$ ,  $\beta$ ,  $\gamma$  e  $T$  possuem a mesma altura negra (antes e depois).

## Configuração Dir-Esq (Caso 2b + Caso 3b)



- A = Avô; P = Pai; T = Tio; N = Novo;
- $\alpha$ ,  $\beta$  e  $\gamma$  são subárvores.
- As subárvores  $\alpha$ ,  $\beta$ ,  $\gamma$  e **T** possuem a mesma altura negra (antes e depois).

- (1) Efetua rotação à direita em torno do pai e troca  $N \leftrightarrow P$ .
- (2) Aplica a configuração **Dir-Dir** (Caso 3b).

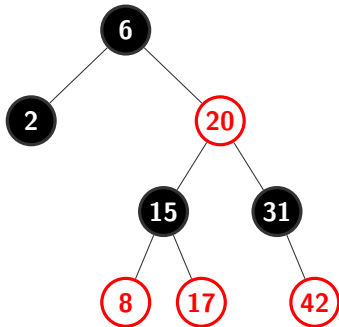
# Código (incompleto) do conserto pós-inserção

## RB-INSERT-FIXUP( $T, z$ )

```
1  while z.p.color == RED
2      if z.p == z.p.p.left
3          y = z.p.p.right
4          if y.color == RED
5              z.p.color = BLACK    // case 1
6              y.color = BLACK    // case 1
7              z.p.p.color = RED    // case 1
8              z = z.p.p
9          else
10             if z == z.p.right
11                 z = z.p            // case 2a
12                 LEFT-ROTATE(T,z)  // case 2a
13                 z.p.color = BLACK // case 3a
14                 z.p.p.color = RED  // case 3a
15                 RIGHT-ROTATE(T,z.p.p) // case 3a
16             else
17                 /* Simétrico ao código acima */
18  T.root.color = BLACK
```

# Exercício

Insira na árvore abaixo as seguintes chaves: 1, 5 e 19.



# Remoção





# Remoções em árvores rubro-negras

- Assim como no caso da inserção, nós utilizaremos rotações e recolorações para manter as propriedades da árvore rubro-negra.

# Remoções em árvores rubro-negras

- Assim como no caso da inserção, nós utilizaremos rotações e recolorações para manter as propriedades da árvore rubro-negra.
  - Contudo, a remoção de nós de uma árvore rubro-negra exige um pouco mais de trabalho.

## Remoções em árvores rubro-negras

- Assim como no caso da inserção, nós utilizaremos rotações e recolorações para manter as propriedades da árvore rubro-negra.
  - Contudo, a remoção de nós de uma árvore rubro-negra exige um pouco mais de trabalho.
- Durante a **inserção**, baseamos as nossas operações de correção na **cor do tio**.

# Remoções em árvores rubro-negras

- Assim como no caso da inserção, nós utilizaremos rotações e recolorações para manter as propriedades da árvore rubro-negra.
  - Contudo, a remoção de nós de uma árvore rubro-negra exige um pouco mais de trabalho.
- Durante a **inserção**, baseamos as nossas operações de correção na **cor do tio**.
  - Já durante a **remoção** nós nos baseamos na **cor do irmão** do nó para decidir qual caso aplicar.

# Remoções em árvores rubro-negras

- Assim como no caso da inserção, nós utilizaremos rotações e recolorações para manter as propriedades da árvore rubro-negra.
  - Contudo, a remoção de nós de uma árvore rubro-negra exige um pouco mais de trabalho.
- Durante a **inserção**, baseamos as nossas operações de correção na **cor do tio**.
  - Já durante a **remoção** nós nos baseamos na **cor do irmão** do nó para decidir qual caso aplicar.
- Durante a **inserção** o principal problema que enfrentamos era ter um pai e um filho ambos vermelhos.

# Remoções em árvores rubro-negras

- Assim como no caso da inserção, nós utilizaremos rotações e recolorações para manter as propriedades da árvore rubro-negra.
  - Contudo, a remoção de nós de uma árvore rubro-negra exige um pouco mais de trabalho.
- Durante a **inserção**, baseamos as nossas operações de correção na **cor do tio**.
  - Já durante a **remoção** nós nos baseamos na **cor do irmão** do nó para decidir qual caso aplicar.
- Durante a **inserção** o principal problema que enfrentamos era ter um pai e um filho ambos vermelhos.
  - Durante a **remoção**, se retirarmos um nó preto, estamos estragando a **propriedade de altura negra** da árvore.

## Revisão — Remoção em árvore binária de busca

**Problema:** dada uma árvore binária de busca  $T$  e uma chave  $k$ , remover o nó com chave  $k$  de  $T$  (se existir) de modo que árvore binária resultante continue sendo uma árvore binária de busca.

- Reduzimos este problema ao problema de remover a raiz de uma árvore binária de busca.

## Revisão — Remoção em árvore binária de busca

**Problema:** dada uma árvore binária de busca  $T$  e uma chave  $k$ , remover o nó com chave  $k$  de  $T$  (se existir) de modo que árvore binária resultante continue sendo uma árvore binária de busca.

- Reduzimos este problema ao problema de remover a raiz de uma árvore binária de busca.
- Caso o nó  $x$  a ser removido exista, dividimos o problema em 3 casos:



**Problema:** dada uma árvore binária de busca  $T$  e uma chave  $k$ , remover o nó com chave  $k$  de  $T$  (se existir) de modo que árvore binária resultante continue sendo uma árvore binária de busca.

- Reduzimos este problema ao problema de remover a raiz de uma árvore binária de busca.
- Caso o nó  $x$  a ser removido exista, dividimos o problema em 3 casos:
  - **Caso 1:** a raiz não tem filhos. A árvore torna-se vazia.

**Problema:** dada uma árvore binária de busca  $T$  e uma chave  $k$ , remover o nó com chave  $k$  de  $T$  (se existir) de modo que árvore binária resultante continue sendo uma árvore binária de busca.

- Reduzimos este problema ao problema de remover a raiz de uma árvore binária de busca.
- Caso o nó  $x$  a ser removido exista, dividimos o problema em 3 casos:
  - **Caso 1:** a raiz não tem filhos. A árvore torna-se vazia.
  - **Caso 2:** a raiz tem um único filho. Seu filho torna-se a nova raiz.

**Problema:** dada uma árvore binária de busca  $T$  e uma chave  $k$ , remover o nó com chave  $k$  de  $T$  (se existir) de modo que árvore binária resultante continue sendo uma árvore binária de busca.

- Reduzimos este problema ao problema de remover a raiz de uma árvore binária de busca.
- Caso o nó  $x$  a ser removido exista, dividimos o problema em 3 casos:
  - **Caso 1:** a raiz não tem filhos. A árvore torna-se vazia.
  - **Caso 2:** a raiz tem um único filho. Seu filho torna-se a nova raiz.
  - **Caso 3:** a raiz tem dois filhos. Neste caso, tomamos o sucessor da antiga raiz como a nova raiz.

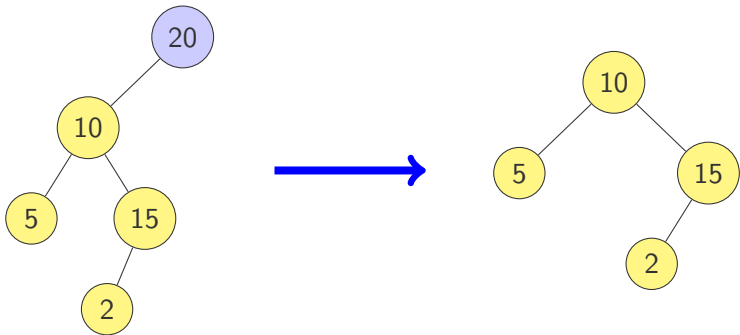
# Revisão — Remoção em árvore binária de busca

- **Caso 1:** a raiz não tem filhos. A árvore torna-se vazia.



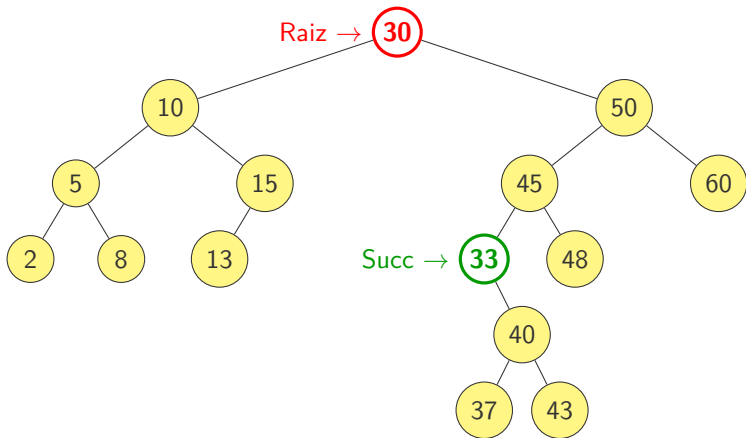
## Revisão — Remoção em árvore binária de busca

- **Caso 2:** a raiz tem um único filho. Seu filho torna-se a nova raiz.



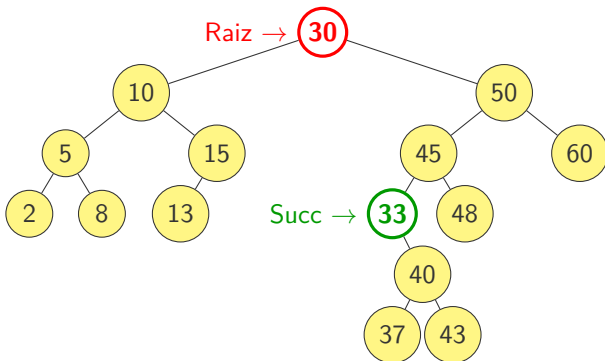
# Revisão — Remoção em árvore binária de busca

- **Caso 3:** a raiz tem ambos os filhos não-nulos. Neste caso, tomamos o **sucessor** da antiga raiz como a nova raiz.



# Revisão — Remoção em árvore binária de busca

- Como implementar o Caso 3: copiamos a informação do nó sucessor para a raiz e removemos o sucessor (recaindo no Caso 1 ou 2).



## Função-membro Remove

Esta função recebe como argumentos a árvore  $T$  e a chave  $k$  a ser removida. Ela faz uma busca pelo nó  $p$  contendo esta chave. Se o nó  $p$  existir, é chamada a função RB-Delete para removê-lo.

REMOVE( $T, k$ )

```
1:  $p = T.root$ 
2: while  $p \neq T.NIL$  and  $p.key \neq k$  do
3:   if  $k < p.key$  then
4:      $p = p.left$ 
5:   else
6:      $p = p.right$ 
7:   end if
8: end while
9: if  $p \neq T.NIL$  then
10:   RB-Delete( $T, p$ )
11: end if
```



# Função-membro RB-Delete

RB-DELETE( $T, z$ )

```
1  if z.left == T.NIL or z.right == T.NIL
2      y = z
3  else
4      y = Minimum(z.right)
5  if y.left != T.NIL
6      x = y.left
7  else
8      x = y.right
9  x.p = y.p
10 if y.p == T.NIL
11     T.root = x
12 else
13     if y == y.p.left
14         y.p.left = x
15     else
16         y.p.right = x
17 if y != z
18     z.key = y.key
19 if y.color == BLACK
20     RB-Delete-Fixup(T,x)
21 delete y
```

# Remoção em árvores rubro-negras — Problemas

Que problemas podemos criar ao removermos fisicamente um nó?

- **Remoção de um nó vermelho:** Não causa problemas no balanceamento.

# Remoção em árvores rubro-negras — Problemas

Que problemas podemos criar ao removermos fisicamente um nó?

- **Remoção de um nó vermelho:** Não causa problemas no balanceamento.
  - Nenhuma altura negra mudou.

# Remoção em árvores rubro-negras — Problemas

Que problemas podemos criar ao removermos fisicamente um nó?

- **Remoção de um nó vermelho:** Não causa problemas no balanceamento.
  - Nenhuma altura negra mudou.
  - Nenhum nó vermelho se tornou vizinho de outro vermelho.

# Remoção em árvores rubro-negras — Problemas

Que problemas podemos criar ao removermos fisicamente um nó?

- **Remoção de um nó vermelho:** Não causa problemas no balanceamento.
  - Nenhuma altura negra mudou.
  - Nenhum nó vermelho se tornou vizinho de outro vermelho.
  - Como o nó é vermelho, ele não era raiz e portanto a raiz permanece preta.

# Remoção em árvores rubro-negras — Problemas

Que problemas podemos criar ao removermos fisicamente um nó?

- **Remoção de um nó vermelho:** Não causa problemas no balanceamento.
  - Nenhuma altura negra mudou.
  - Nenhum nó vermelho se tornou vizinho de outro vermelho.
  - Como o nó é vermelho, ele não era raiz e portanto a raiz permanece preta.
- **Remoção de um nó preto:** Mais de uma propriedade da árvore rubro-negra será quebrada. Quais?

# Remoção em árvores rubro-negras — Problemas

Considere que o nó **y** a ser fisicamente removido é **preto**:

- (P1) Um nó é **vermelho** ou é **preto**.
- (P2) A raiz é preta.
- (P3) Toda folha (NULL) é preta.
- (P4) Se um nó é vermelho então ambos os seus filhos são pretos.
- (P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos

# Remoção em árvores rubro-negras — Problemas

Considere que o nó **y** a ser fisicamente removido é **preto**:

- Se **y** era raiz da árvore e um filho vermelho de **y** se torna raiz quebramos a **Propriedade 2**. (easy!)

(P1) Um nó é **vermelho** ou é **preto**.

(P2) A raiz é preta.

(P3) Toda folha (NULL) é preta.

(P4) Se um nó é vermelho então ambos os seus filhos são pretos.

(P5) Para cada nó  $p$ , todos os caminhos desde  $p$  até as folhas contêm o mesmo número de nós pretos



# Remoção em árvores rubro-negras — Problemas

Considere que o nó **y** a ser fisicamente removido é **preto**:

- Se **y** era raiz da árvore e um filho vermelho de **y** se torna raiz quebramos a **Propriedade 2**. (easy!)
- Se um filho vermelho de **y** toma seu lugar e o pai de **y** também é vermelho, então, agora violamos a **Propriedade 4**. (easy!)

(P1) Um nó é **vermelho** ou é **preto**.

(P2) A raiz é preta.

(P3) Toda folha (NULL) é preta.

(P4) Se um nó é vermelho então ambos os seus filhos são pretos.

(P5) Para cada nó *p*, todos os caminhos desde *p* até as folhas contêm o mesmo número de nós pretos

# Remoção em árvores rubro-negras — Problemas

Considere que o nó **y** a ser fisicamente removido é **preto**:

- Se **y** era raiz da árvore e um filho vermelho de **y** se torna raiz quebramos a **Propriedade 2**. (easy!)
- Se um filho vermelho de **y** toma seu lugar e o pai de **y** também é vermelho, então, agora violamos a **Propriedade 4**. (easy!)
- A remoção de **y** faz com que qualquer caminho que continha **y** anteriormente tenha um nó preto a menos. Desse modo quebramos a **Propriedade 5**.

(P1) Um nó é **vermelho** ou é **preto**.

(P2) A raiz é preta.

(P3) Toda folha (NULL) é preta.

(P4) Se um nó é vermelho então ambos os seus filhos são pretos.

(P5) Para cada nó **p**, todos os caminhos desde **p** até as folhas contêm o mesmo número de nós pretos

# Remoção — Primeiro passo do algoritmo

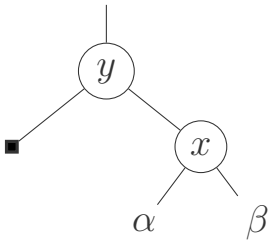
**Passo 1:** Execute a **remoção padrão** da Árvore Binária de Busca.

- Na remoção padrão, sempre excluimos um **nó que é folha ou que tem apenas um filho** (para um nó que tem dois filhos, copiamos a chave do seu sucessor nele e depois removemos o sucessor).

# Remoção — Primeiro passo do algoritmo

**Passo 1:** Execute a **remoção padrão** da Árvore Binária de Busca.

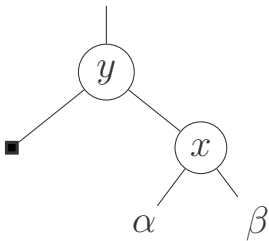
- Na remoção padrão, sempre excluimos um **nó que é folha ou que tem apenas um filho** (para um nó que tem dois filhos, copiamos a chave do seu sucessor nele e depois removemos o sucessor).
- Seja **y** o nó a ser excluído e seja **x** o filho que substitui **y**.



# Remoção — Primeiro passo do algoritmo

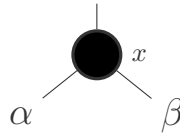
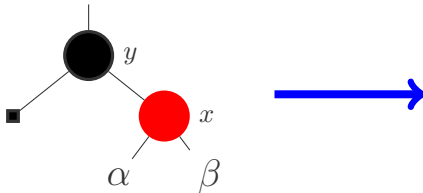
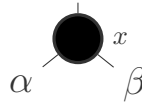
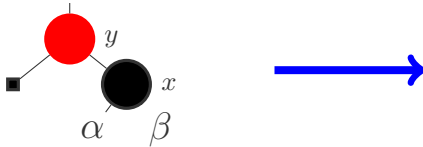
**Passo 1:** Execute a **remoção padrão** da Árvore Binária de Busca.

- Na remoção padrão, sempre excluimos um **nó que é folha ou que tem apenas um filho** (para um nó que tem dois filhos, copiamos a chave do seu sucessor nele e depois removemos o sucessor).
- Seja **y** o nó a ser excluído e seja **x** o filho que substitui **y**.
  - Note que **x** é **nil** quando **y** é uma folha e a cor de **nil** é **preta**.



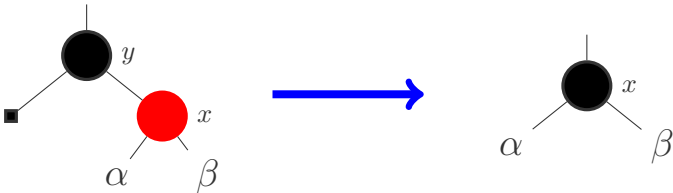
## Análise das cores de $y$ e $x$ — Caso Fácil

- Se  $y$  ou  $x$  forem **vermelhos**, fazemos  $x \rightarrow \text{color} = \text{BLACK}$  (não há alteração em nenhuma altura negra).



## Análise das cores de $y$ e $x$ — Caso Fácil

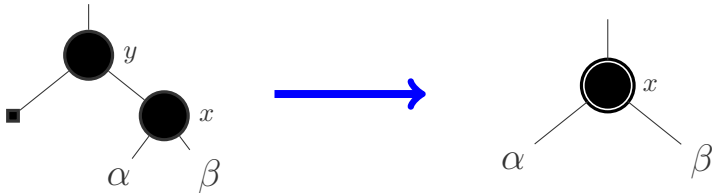
- Se  $y$  ou  $x$  forem **vermelhos**, fazemos  $x \rightarrow \text{color} = \text{BLACK}$  (não há alteração em nenhuma altura negra).



- $y$  e  $x$  não podem ser ambos **vermelhos**, pois pai e filho vermelhos não são permitidos na árvore rubro-negra.

## Análise das cores de $y$ e $x$ — Caso Ruim

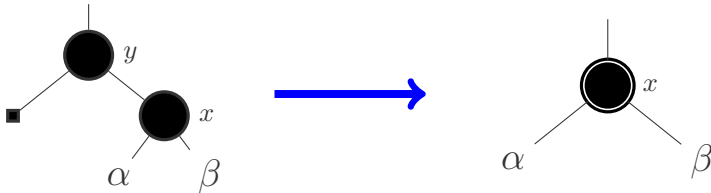
- O verdadeiro problema na remoção ocorre quando ambos  $y$  e  $x$  são pretos.





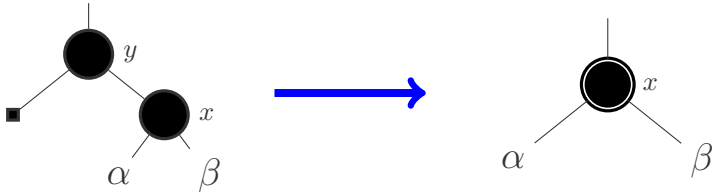
## Análise das cores de $y$ e $x$ — Caso Ruim

- O verdadeiro problema na remoção ocorre quando ambos  $y$  e  $x$  são **pretos**.
- Neste caso, ao remover o nó  $y$ , as alturas negras de todos os seus ancestrais ficam “corrompidas”.



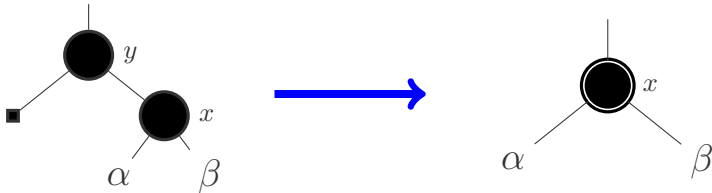
## Análise das cores de $y$ e $x$ — Caso Ruim

- O verdadeiro problema na remoção ocorre quando ambos  $y$  e  $x$  são **pretos**.
- Neste caso, ao remover o nó  $y$ , as alturas negras de todos os seus ancestrais ficam “**corrompidas**”.
- A fim de consertar este caso problemático, usamos a noção de **preto duplo**.



## Análise das cores de $y$ e $x$ — Caso Ruim

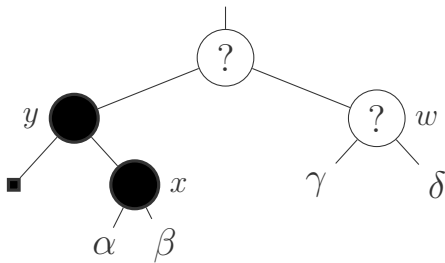
- O verdadeiro problema na remoção ocorre quando ambos  $y$  e  $x$  são **pretos**.
- Neste caso, ao remover o nó  $y$ , as alturas negras de todos os seus ancestrais ficam “**corrompidas**”.
- A fim de consertar este caso problemático, usamos a noção de **preto duplo**.
  - Quando um nó preto é excluído e substituído por um filho preto, o filho é marcado como **preto duplo**. A tarefa principal agora é converter esse **preto duplo** em **preto**.



## Segundo passo: Eliminação do preto duplo

Usamos a seguinte nomenclatura para os nós envolvidos no processo de remoção:

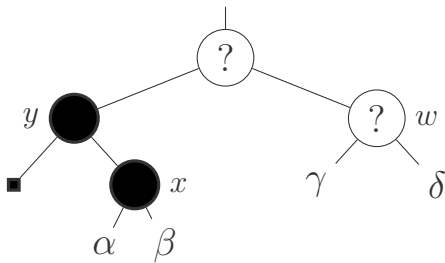
- $z$  — O nó cuja chave será removida.



## Segundo passo: Eliminação do preto duplo

Usamos a seguinte nomenclatura para os nós envolvidos no processo de remoção:

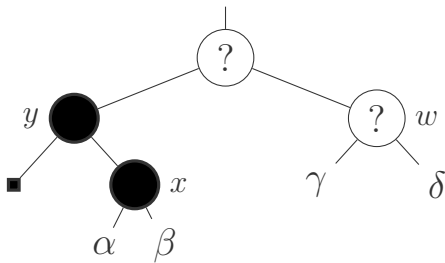
- $z$  — O nó cuja chave será removida.
- $y$  — O nó que será fisicamente removido. Onde  $y = z$  se  $z$  possui 0 ou 1 filho. Caso contrário,  $y = \text{sucessor}(z)$ .



## Segundo passo: Eliminação do preto duplo

Usamos a seguinte nomenclatura para os nós envolvidos no processo de remoção:

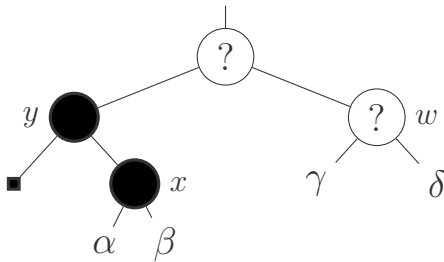
- **z** — O nó cuja chave será removida.
- **y** — O nó que será fisicamente removido. Onde  $y = z$  se  $z$  possui 0 ou 1 filho. Caso contrário,  $y = \text{sucessor}(z)$ .
- **x** — O único filho de **y** antes de qualquer modificação, ou **NIL** caso **y** não tenha filhos.



## Segundo passo: Eliminação do preto duplo

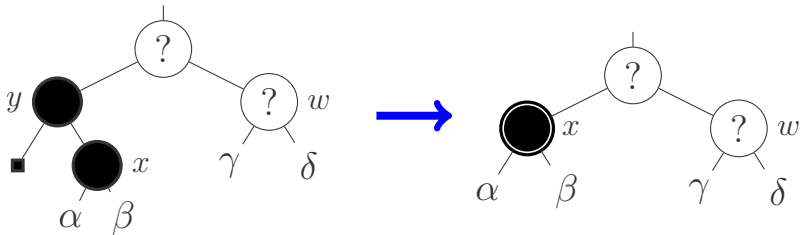
Usamos a seguinte nomenclatura para os nós envolvidos no processo de remoção:

- $z$  — O nó cuja chave será removida.
- $y$  — O nó que será fisicamente removido. Onde  $y = z$  se  $z$  possui 0 ou 1 filho. Caso contrário,  $y = \text{sucessor}(z)$ .
- $x$  — O único filho de  $y$  antes de qualquer modificação, ou  $NIL$  caso  $y$  não tenha filhos.
- $w$  — O tio de  $x$  (irmão de  $y$ ) antes da remoção de  $y$ .



## Segundo passo: Eliminação do preto duplo

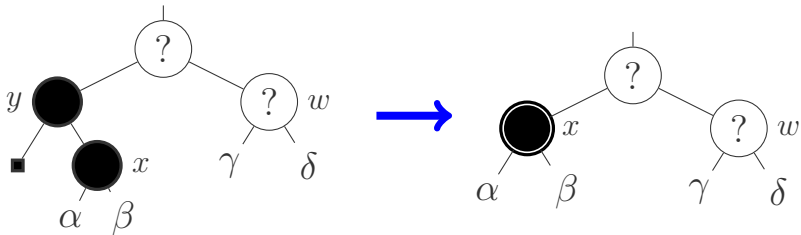
- Durante o conserto da árvore, quando considerarmos o nó  $x$  vamos contá-lo como preto duas vezes (**duplo preto**)





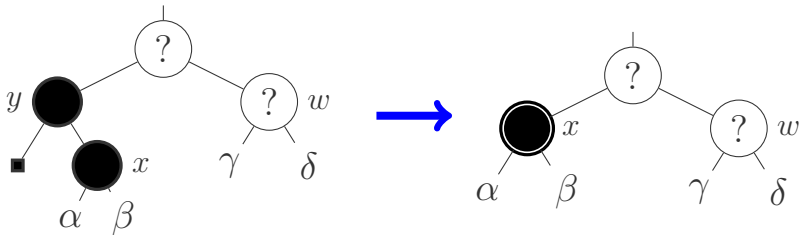
## Segundo passo: Eliminação do preto duplo

- Durante o conserto da árvore, quando considerarmos o nó  $x$  vamos contá-lo como preto duas vezes (**duplo preto**)
- A ideia é que o  $x$  possa receber sempre a contagem de preto do pai para manter a **Propriedade 5**.



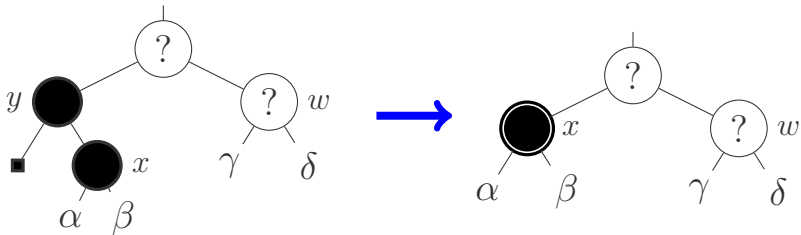
## Segundo passo: Eliminação do preto duplo

- Durante o conserto da árvore, quando considerarmos o nó  $x$  vamos contá-lo como preto duas vezes (**duplo preto**)
- A ideia é que o  $x$  possa receber sempre a contagem de preto do pai para manter a **Propriedade 5**.
- $x$  aponta para um **duplo preto**, o que nos diz que o nó  $w$  existe (**Por quê?**)



## Segundo passo: Eliminação do preto duplo

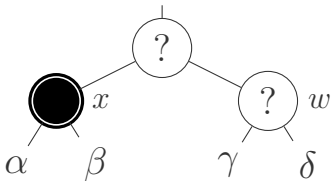
- Durante o conserto da árvore, quando considerarmos o nó  $x$  vamos contá-lo como preto duas vezes (**duplo preto**)
- A ideia é que o  $x$  possa receber sempre a contagem de preto do pai para manter a **Propriedade 5**.
- $x$  aponta para um **duplo preto**, o que nos diz que o nó  $w$  existe (**Por quê?**)
- No caso em que  $x$  era inicialmente **vermelho** (ainda assim o contamos como preto no cálculo da altura negra), basta pintar  $x$  de **preto** para finalizar o conserto da árvore.



## Eliminação do preto duplo — 4 Casos

Considere que o nó  $y$  foi removido. Agora,  $w$  é o irmão de  $x$ . Existem 4 casos a considerar:

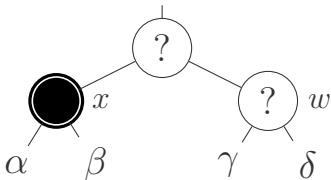
- Caso 1:  $w$  é vermelho.



## Eliminação do preto duplo — 4 Casos

Considere que o nó  $y$  foi removido. Agora,  $w$  é o irmão de  $x$ . Existem 4 casos a considerar:

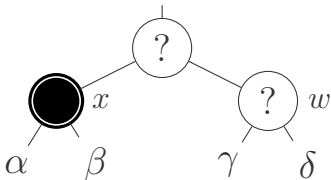
- Caso 1:  $w$  é **vermelho**.
- Caso 2:  $w$  é **preto** e seus dois filhos são **pretos**.



## Eliminação do preto duplo — 4 Casos

Considere que o nó  $y$  foi removido. Agora,  $w$  é o irmão de  $x$ . Existem 4 casos a considerar:

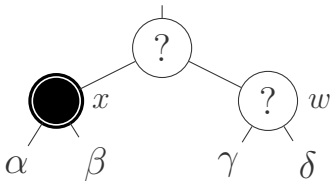
- Caso 1:  $w$  é **vermelho**.
- Caso 2:  $w$  é **preto** e seus dois filhos são **pretos**.
- Caso 3:  $w$  é **preto**, seu filho esquerdo é **vermelho** e seu filho direito é **preto**.



## Eliminação do preto duplo — 4 Casos

Considere que o nó  $y$  foi removido. Agora,  $w$  é o irmão de  $x$ . Existem 4 casos a considerar:

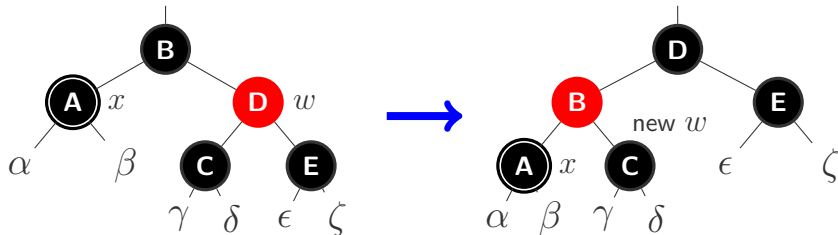
- Caso 1:  $w$  é **vermelho**.
- Caso 2:  $w$  é **preto** e seus dois filhos são **pretos**.
- Caso 3:  $w$  é **preto**, seu filho esquerdo é **vermelho** e seu filho direito é **preto**.
- Caso 4:  $w$  é **preto** e seu filho direito é **vermelho**.



## Caso 1: o nó $w$ é vermelho

Como  $w$  é **vermelho**, ambos os filhos são **pretos**. Logo:

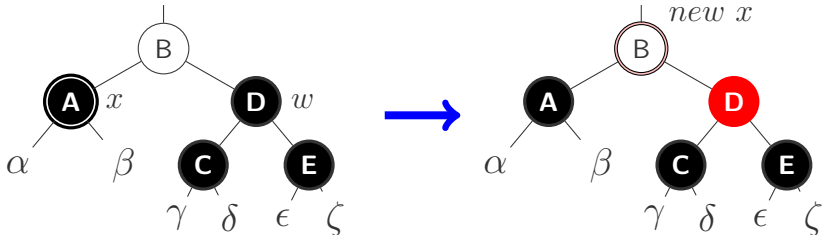
- Trocamos as cores de  $w$  e  $x.parent$ .
- Efetuamos uma rotação à esquerda tendo como pivô  $x.parent$ .
- Essas alterações não violam nenhuma propriedade da árvore rubro-negra.
- Mas transformam o Caso 1 no Caso 2, Caso 3 ou Caso 4.





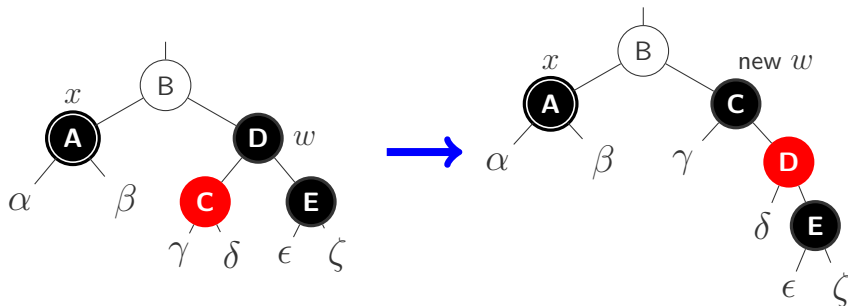
## Caso 2: o nó $w$ e seus dois filhos são pretos

- Retiramos um preto de  $x$  e um de  $w$  deixando  $x$  com apenas um **preto** e deixando  $w$  **vermelho**.
- Para compensar a remoção de um preto de  $x$  e de  $w$ , adicionamos um **preto extra** no  $x.parent$  que era originalmente **preto** ou **vermelho**.
- Note que se  $x.parent$  era vermelho, então terminou. Pintamo-lo de preto e acabou. Caso contrário, jogamos a bomba para cima, tratando o  $x.parent$  como sendo o novo  $x$ .



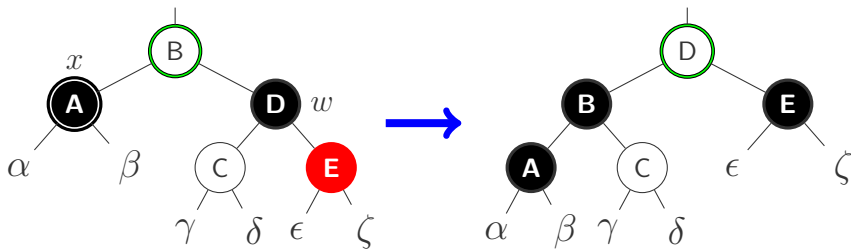
## Caso 3: $w$ e $w.\text{right}$ são pretos, $w.\text{left}$ é vermelho

- Trocamos as cores de  $w$  e de seu filho esquerdo.
- Rotaciona árvore à direita usando como pivô  $w$ .
- Essas operações não introduzem violações
- Neste ponto, o novo irmão  $w$  de  $x$  é um nó **preto** com um filho da direita **vermelho**. Estamos, portanto, no **Caso 4**.



## Caso 4: $w$ é preto e $w.\text{right}$ é vermelho

- $w$  é pintado com a cor de  $x.\text{parent}$
- Pinta  $x.\text{parent}$  de **preto**
- Pinta o filho direito de  $w$  de **preto**
- Rotaciona árvore à esquerda usando como pivô  $x.\text{parent}$



- Após a execução desse caso, nenhuma regulagem adicional será necessária.

# Pseudocódigo do conserto da Remoção



# Função-membro RB-Delete-FixUp

RB-DELETE-FIXUP( $T, x$ )

```
1  while  $x \neq T.root$  and  $x.color == BLACK$ 
2      if  $x == x.p.left$            // is  $x$  a left child?
3           $w = x.p.right$          //  $w$  is  $x$ 's sibling
4          if  $w.color == RED$ 
5               $w.color = BLACK$ 
6               $x.p.color = RED$ 
7              LEFT-ROTATE( $T, x.p$ )
8               $w = x.p.right$ 
9          if  $w.left.color == BLACK$  and  $w.right.color == BLACK$ 
10              $w.color = RED$ 
11              $x = x.p$ 
12         else
13             if  $w.right.color == BLACK$ 
14                  $w.left.color = BLACK$ 
15                  $w.color = RED$ 
16                 RIGHT-ROTATE( $T, w$ )
17                  $w = x.p.right$ 
18              $w.color = x.p.color$ 
19              $x.p.color = BLACK$ 
20              $w.right.color = BLACK$ 
21             LEFT-ROTATE( $T, x.p$ )
22              $x = T.root$ 
23     else // same as lines 3–22, but with “right” and “left” exchanged
24          $x.color = BLACK$ 
```

# AVL versus Rubro-Negra



# AVL $\times$ Rubro-negra

- Na teoria, possuem a mesma complexidade computacional

# AVL × Rubro-negra

- Na teoria, possuem a mesma complexidade computacional
  - Inserção, remoção e busca:  $O(\lg(n))$ .



# AVL × Rubro-negra

- Na teoria, possuem a mesma complexidade computacional
  - Inserção, remoção e busca:  $O(\lg(n))$ .
- Na prática, a árvore AVL é mais rápida na operação de busca, e mais lenta nas operações de inserção e remoção.

# AVL × Rubro-negra

- Na teoria, possuem a mesma complexidade computacional
  - Inserção, remoção e busca:  $O(\lg(n))$ .
- Na prática, a árvore AVL é mais rápida na operação de busca, e mais lenta nas operações de inserção e remoção.
  - A árvore AVL é mais balanceada do que a árvore rubro-negra, o que acelera a operação de busca.

# AVL × Rubro-negra

- Na teoria, possuem a mesma complexidade computacional
  - Inserção, remoção e busca:  $O(\lg(n))$ .
- Na prática, a árvore AVL é mais rápida na operação de busca, e mais lenta nas operações de inserção e remoção.
  - A árvore AVL é mais balanceada do que a árvore rubro-negra, o que acelera a operação de busca.
  - A árvore rubro-negra realiza menor quantidade de rotações ao inserir ou remover nós.

# AVL × Rubro-negra

- AVL: balanceamento mais rígido.
  - No pior caso, uma operação de remoção pode exigir  $O(\lg(n))$  rotações na árvore AVL, mas apenas 3 rotações na árvore rubro-negra.

# AVL × Rubro-negra

- AVL: balanceamento mais rígido.
  - No pior caso, uma operação de remoção pode exigir  $O(\lg(n))$  rotações na árvore AVL, mas apenas 3 rotações na árvore rubro-negra.
- Qual usar?
  - Operação de busca é a mais usada? Melhor usar uma árvore AVL.
  - Inserção ou remoção são mais usadas? Melhor usar uma árvore Rubro-negra.

# Exercícios



# Exercícios

- É possível ter uma árvore rubro-negra em que todos os seus nós sejam pretos?
- Prove ou dê um contraexemplo: toda árvore rubro-negra é uma árvore AVL.
- Prove ou dê um contraexemplo: toda árvore AVL é rubro-negra.

FIM

