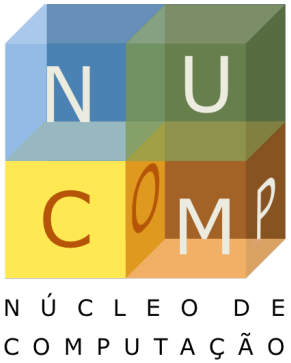
UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA
NÚCLEO DE COMPUTAÇÃO - NUCOMP



Algoritmos e Estruturas de Dados I

Árvores Rubro-Negras

Prof. Flávio José Mendes Coelho
fcoelho@uea.edu.br

Objetivos

Entender...

1. Definição de árvore rubro-negra
2. Busca em árvore rubro-negra
3. Inserção e balanceamento
4. Rotações em árvores rubro-negras

Árvore rubro-negra: definições e propriedades

Árvore rubro-negra (*red-black tree*)

ABB com as seguintes **propriedades**:

P1. Cada nó é VERMELHO ou PRETO.

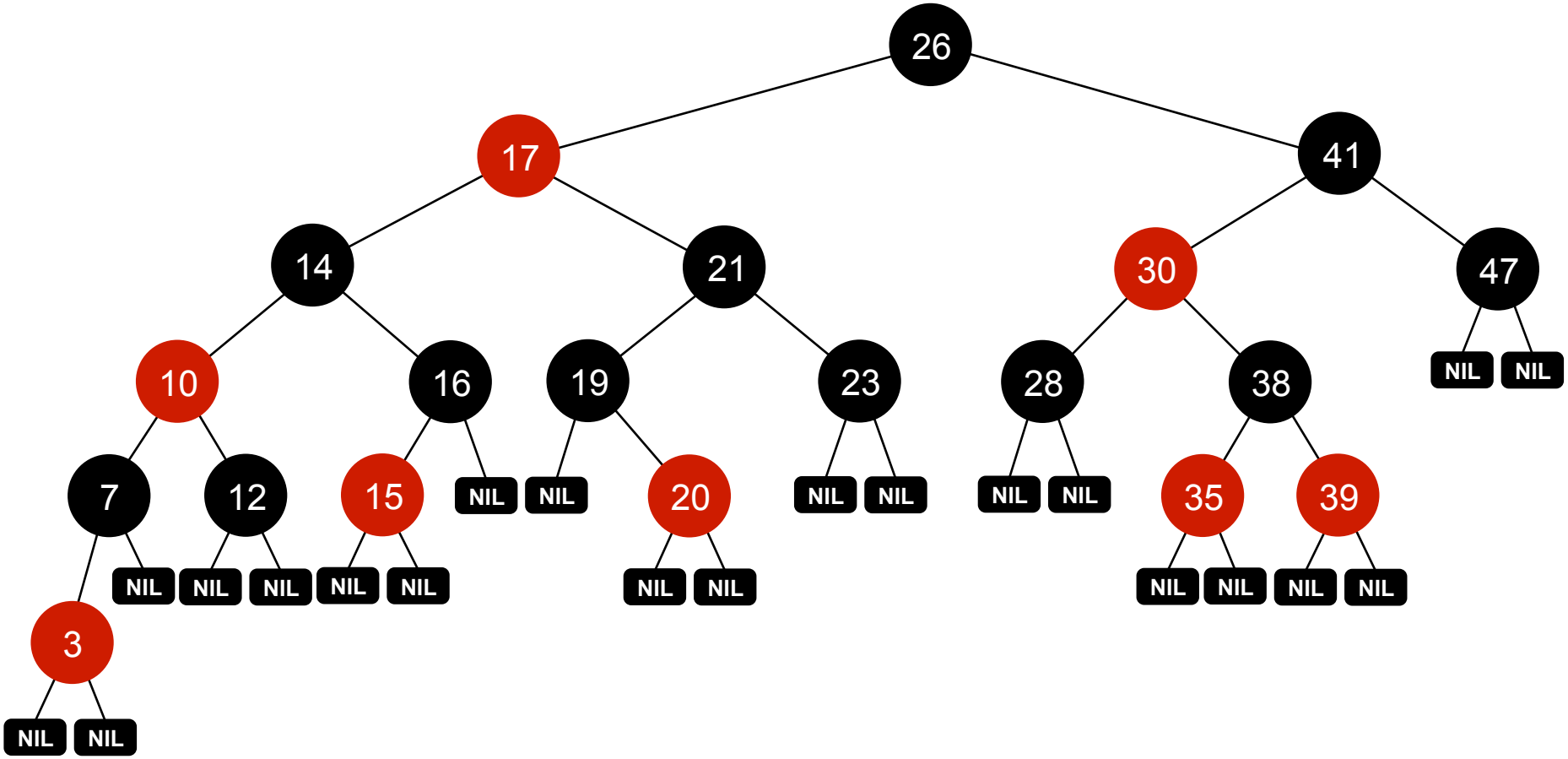
P2. A raiz é PRETA.

P3. Cada folha é PRETA.

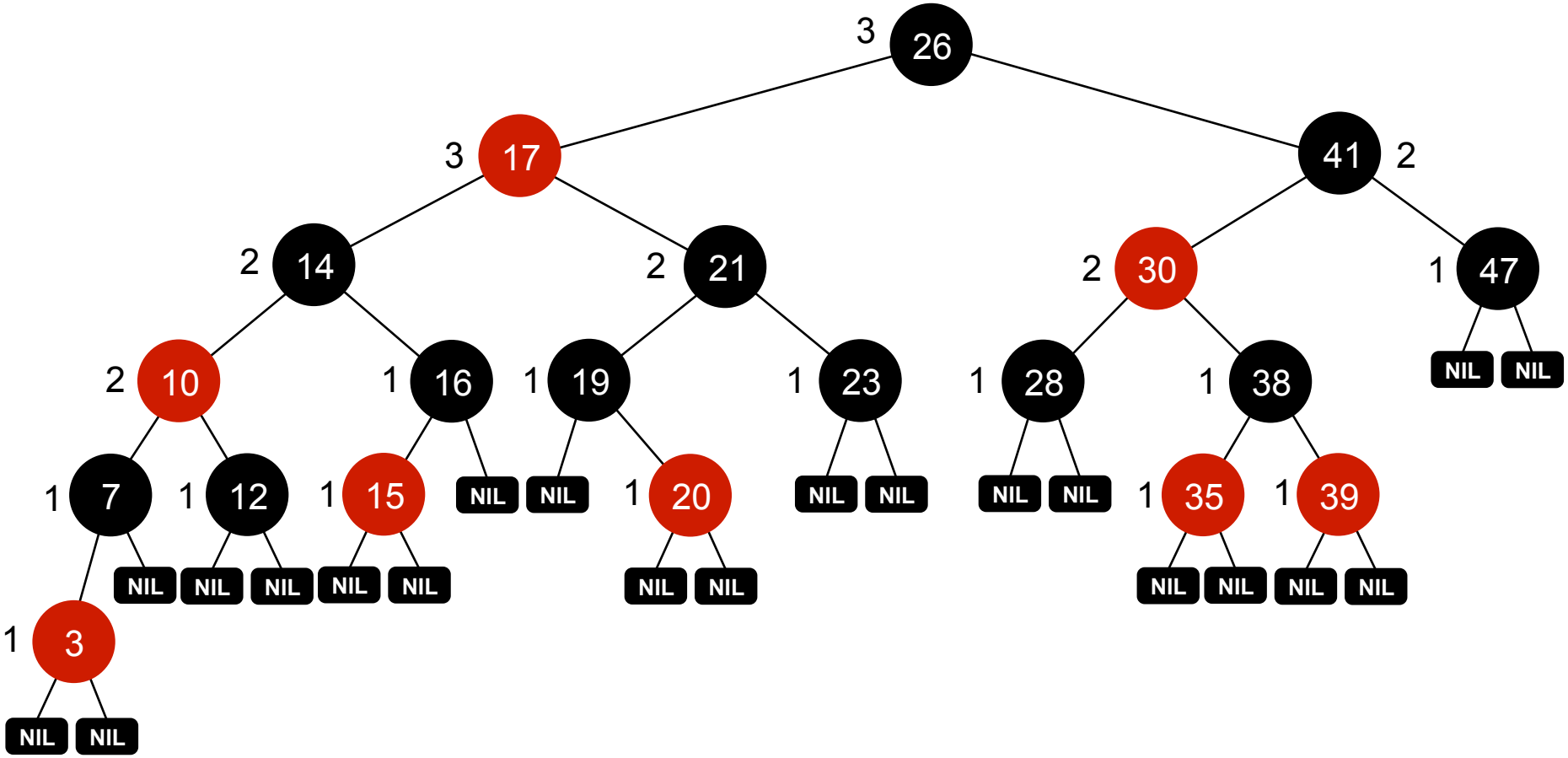
P4. Nó VERMELHO tem filhos PRETOS.

P5. O caminho simples de um nó qualquer até seus descentes folhas tem o mesmo número de nós PRETOS.

Árvore rubro-negra

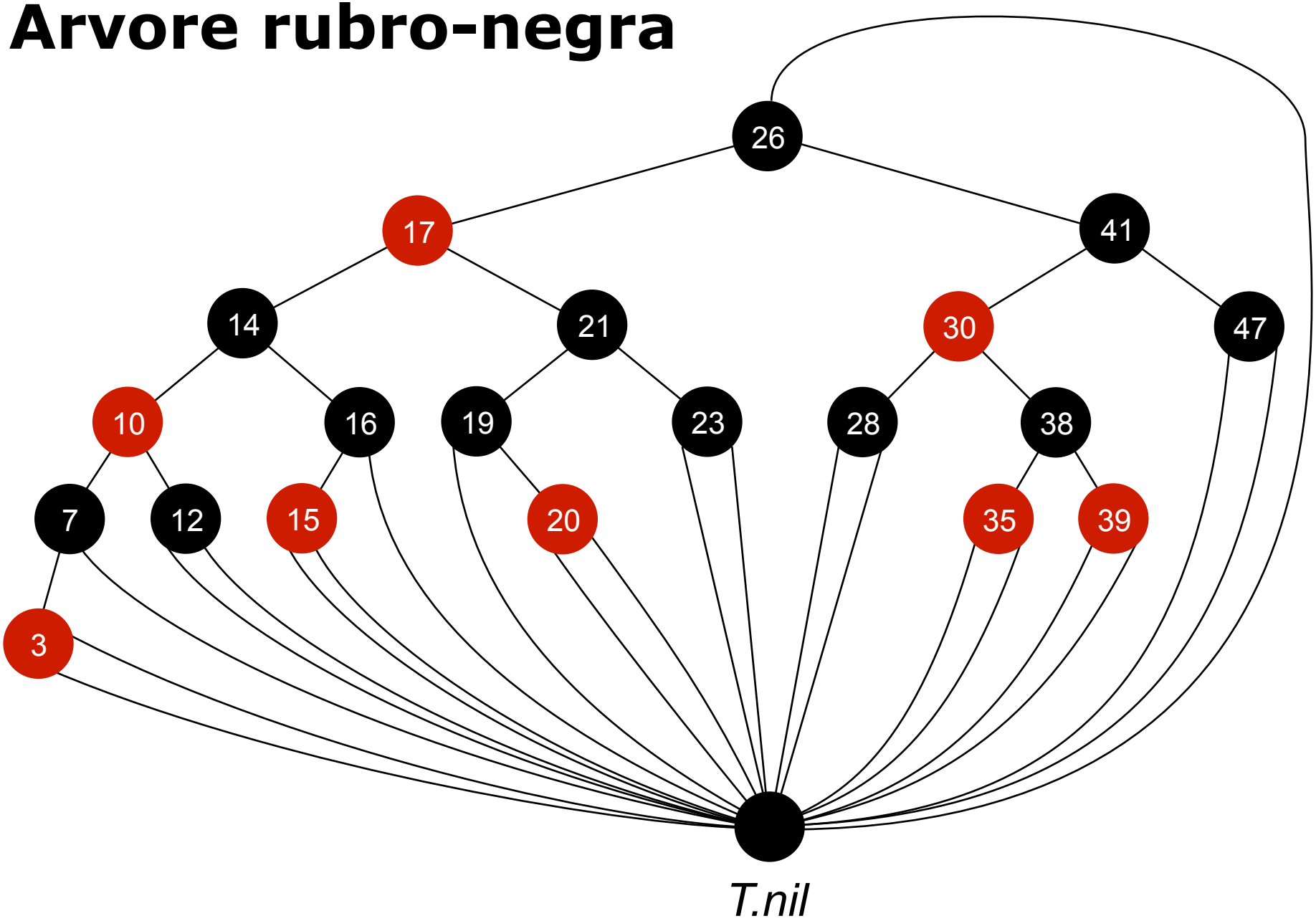


Árvore rubro-negra

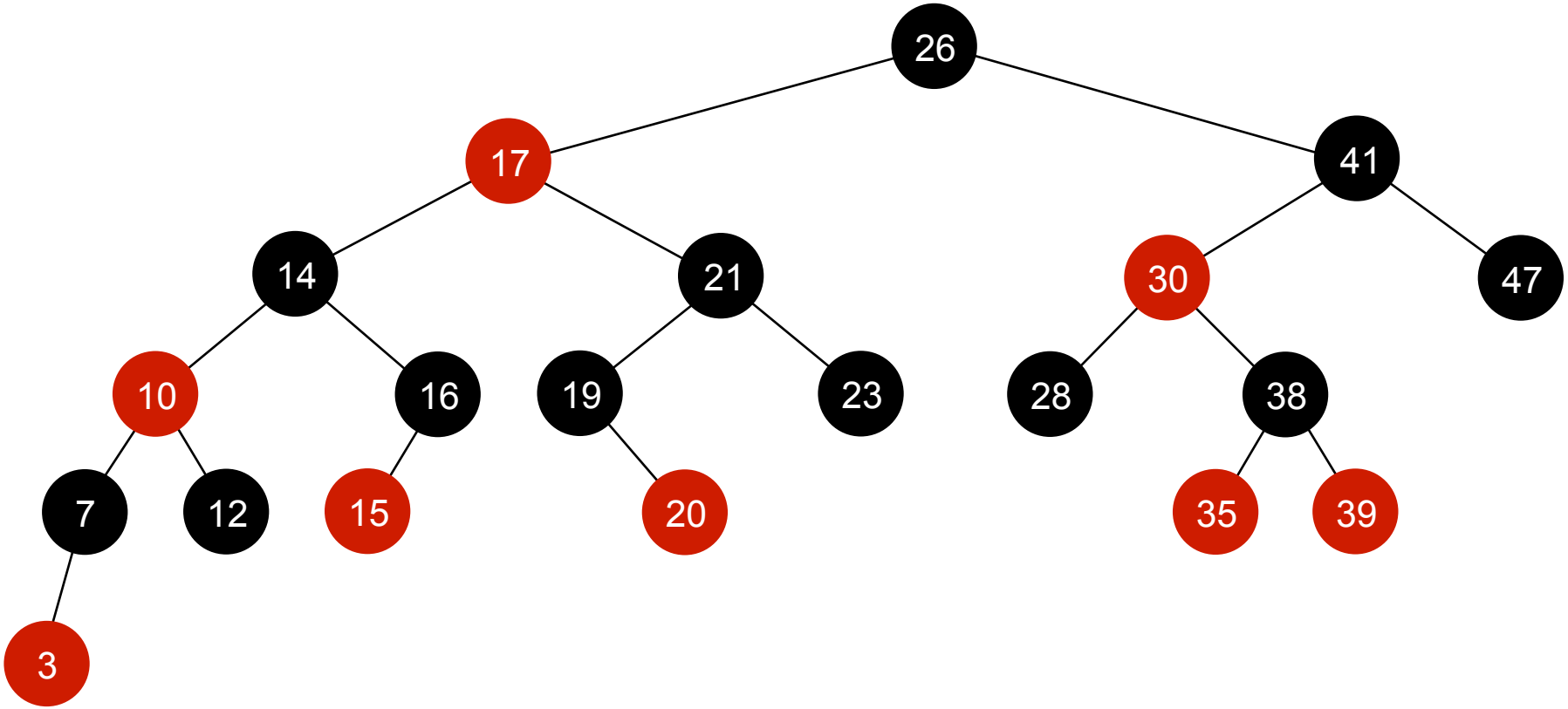


Altura-preta, ***ap*(*x*)**: número de nós pretos, não incluindo o *x*, até uma folha (NIL).

Árvore rubro-negra



Árvore rubro-negra



Árvore rubro-negra

As propriedades garantem que a árvore seja **aproximadamente balanceada**.

Nenhum caminho da raiz até uma folha é 2x mais longo do que outro qualquer.

Lema. Uma árvore rubro-negra com n nós internos tem altura no máximo $2\log(n + 1)$.

Árvore rubro-negra

Nó da árvore rubro-negra:

- **cor**: VERMELHO/PRETO.
- **item** ou **chave**: item contém chave.
- **dir**, **esq**: ponteiros para as sub-árvores esquerda e direita.
- ***pai***: ponteiro para o nó pai.

Busca na Árvore Rubro-Negra

Busca na árvore rubro-negra

Semelhante à busca em uma ABB.

Lembrar que folha na RN é nó *T.nil*.

```
// p é um ponteiro para No
RN-BUSCA(chave, No p)
1   se p == T.nil ou chave == p.item.chave
2       retorne p
3   senão se chave < p.item.chave
4       return RN-BUSCA(chave, p.esq)
5   senão
6       return RN-BUSCA(chave, p.dir)
```

Inserção na Árvore Rubro-Negra

Inserção na árvore rubro-negra

A inserção e remoção de itens desbalanceiam a árvore.

Procedimentos para balancear a árvore:

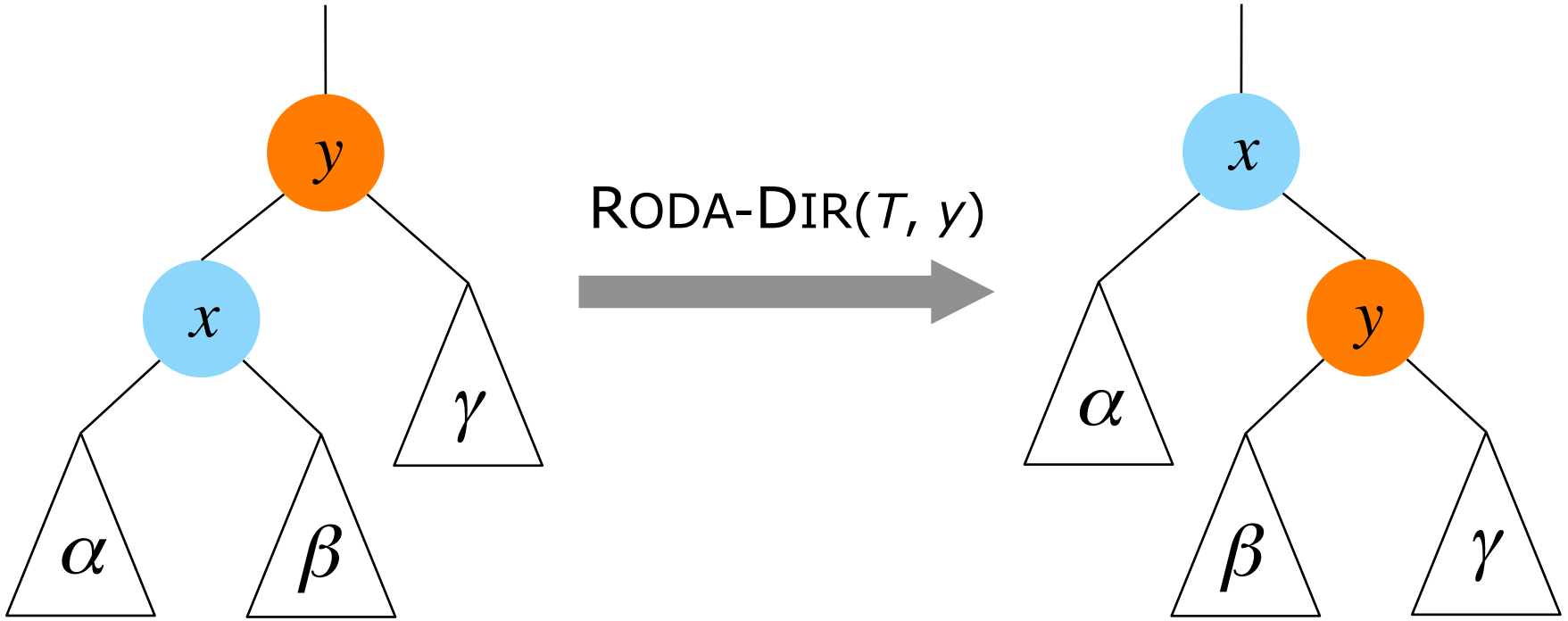
- **Recolorir** nós;
- Aplicar **rotações**:

  RODA-ESQ(T, x)

  RODA-DIR (T, y)

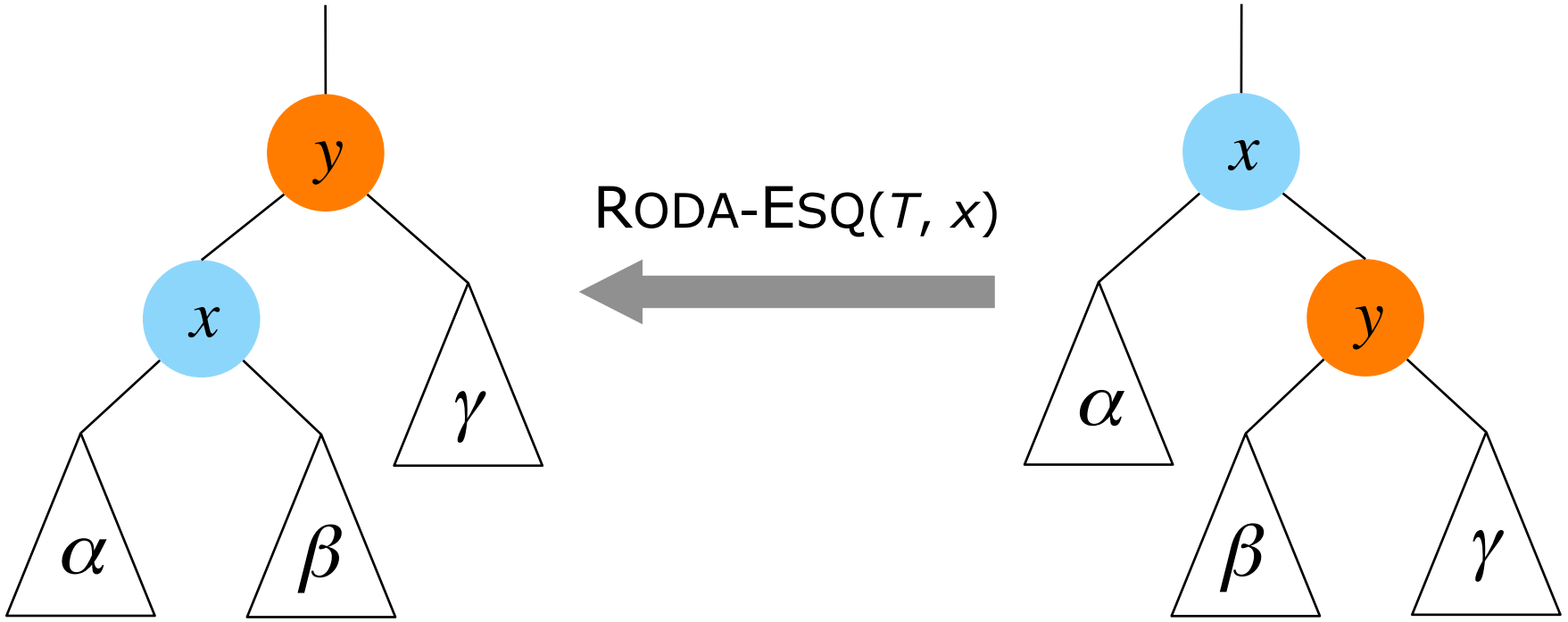
T, x e y são ponteiros para No.

Rotações



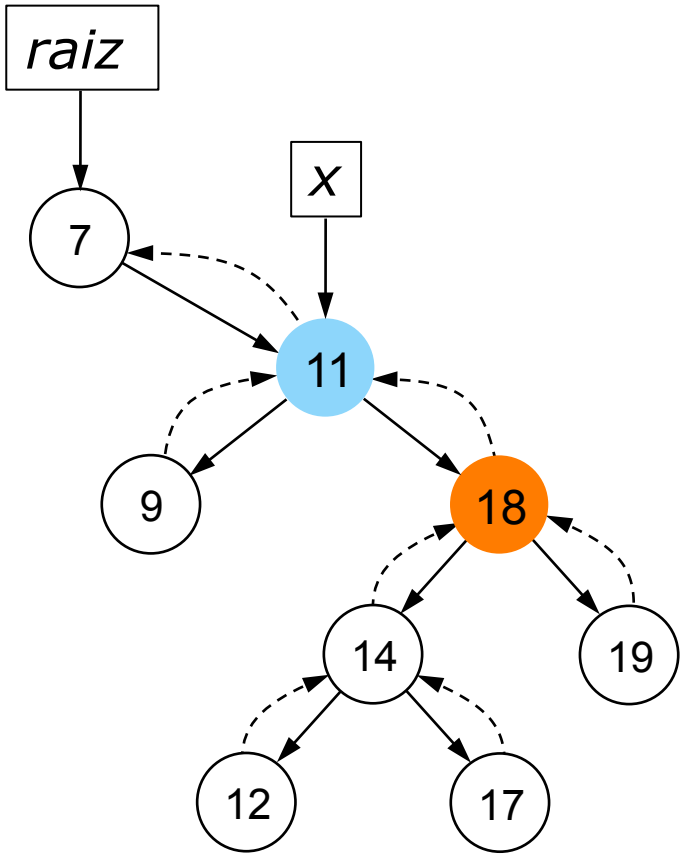
Nó *x* <u>não</u> deve ser *T.nil*.

Rotações

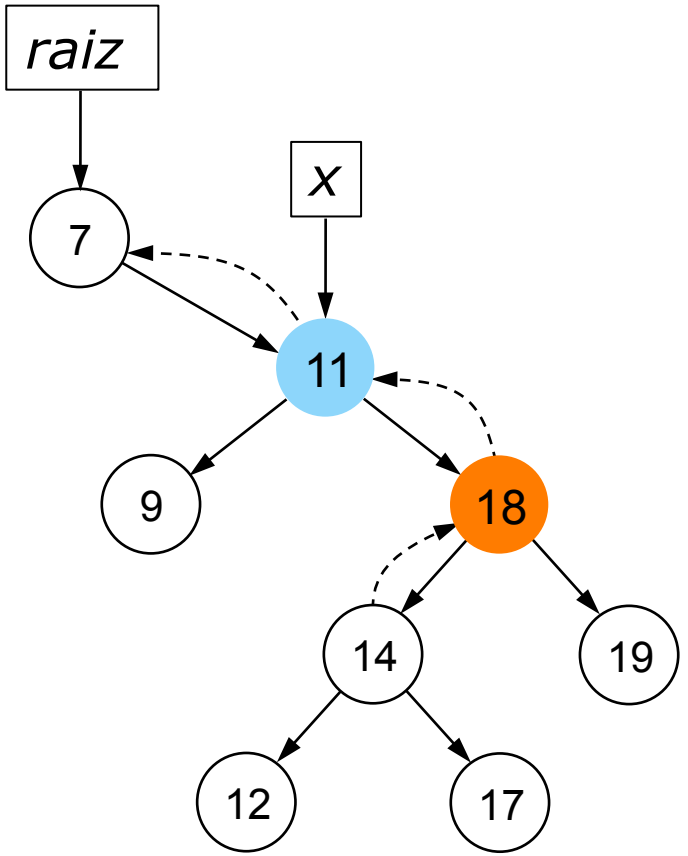


Nó *y* <u>não</u> deve ser *T.nil*.

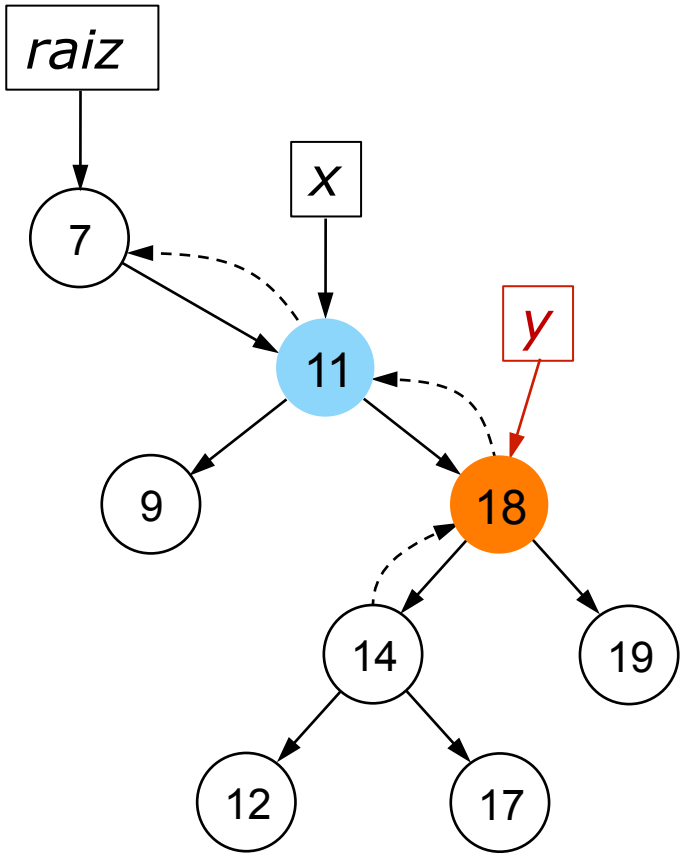
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



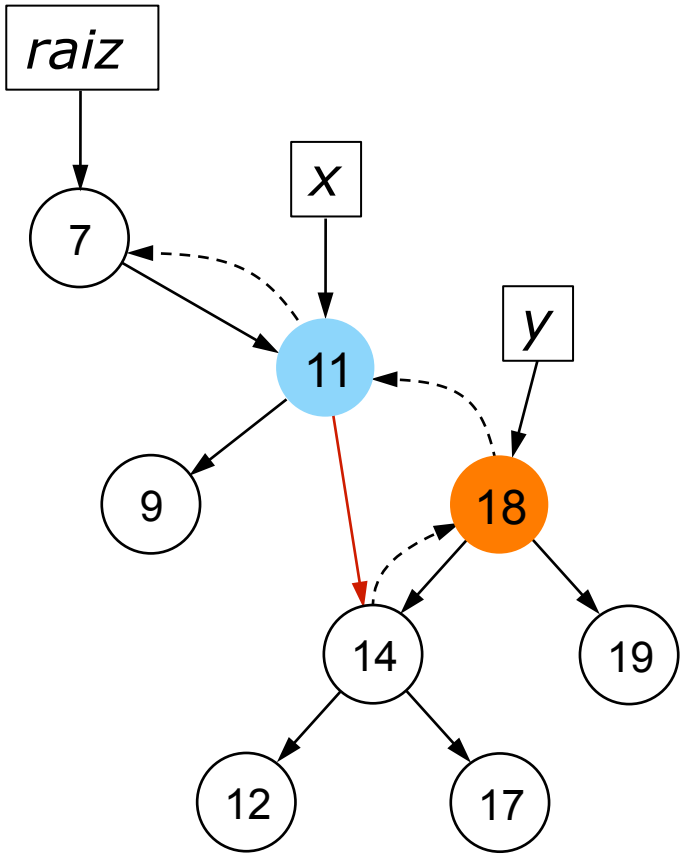
```
RODA-ESQ(T, x)
1     y = x.dir
2     x.dir = y.esq
3     se y.esq ≠ T.nil
4           y.esq.pai = x
5     y.pai = x.pai
6     se x.pai == T.nil
7           T.raiz = y
8     senão se x == x.pai.esq
9           x.pai.esq = y
10    senão
11          x.pai.dir = y
12    y.esq = x
13    x.pai = y
```



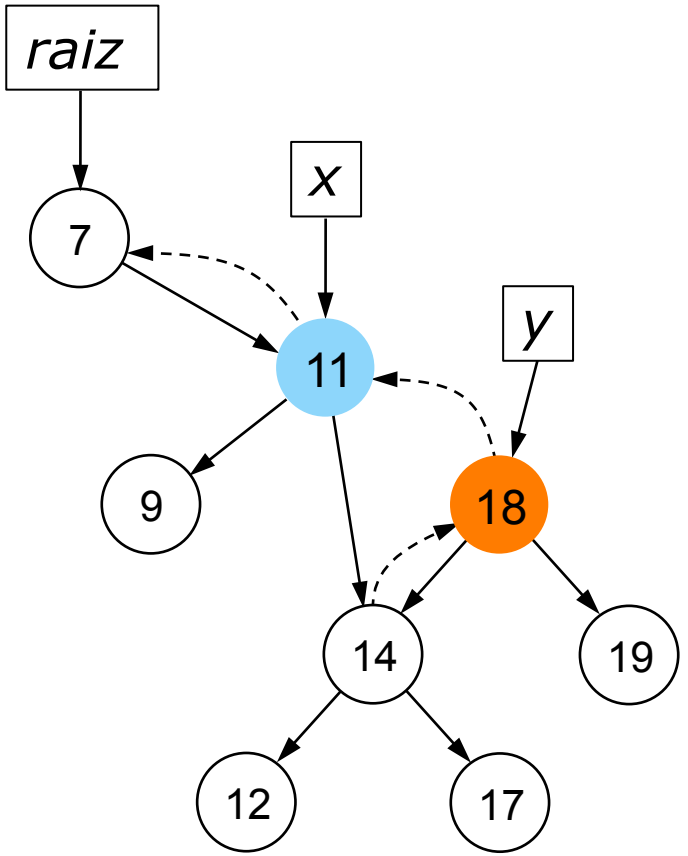
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



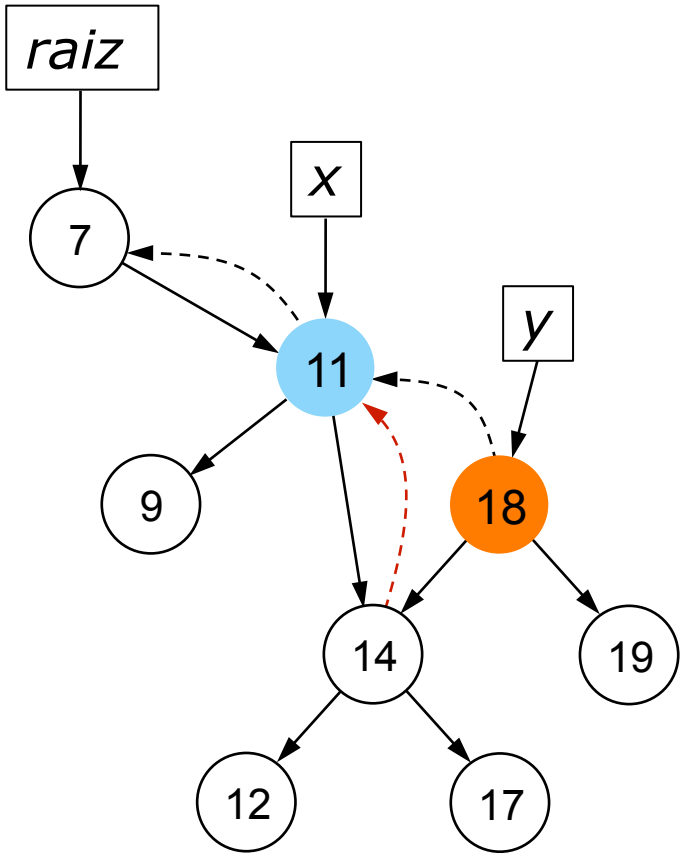
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



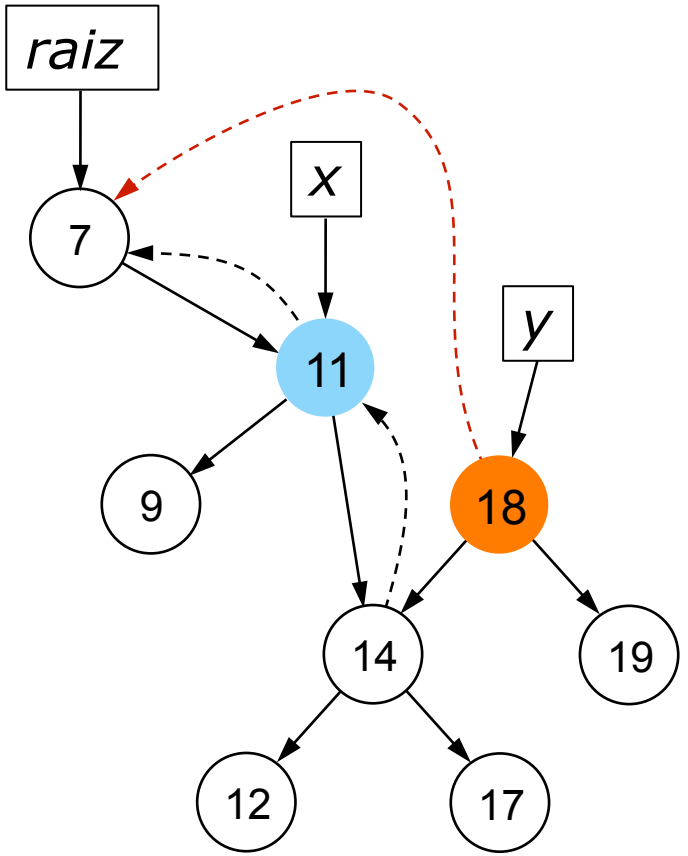
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



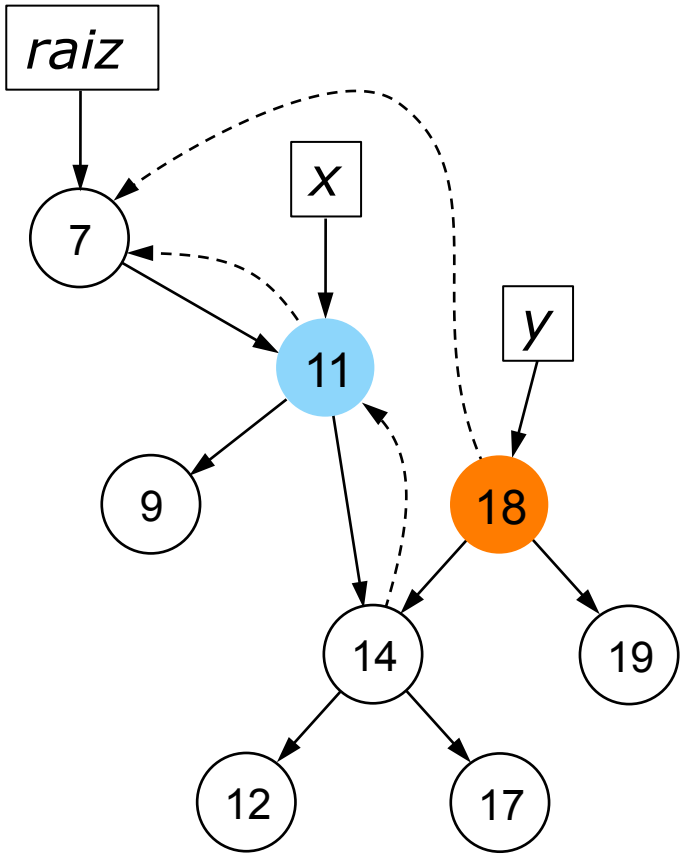
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



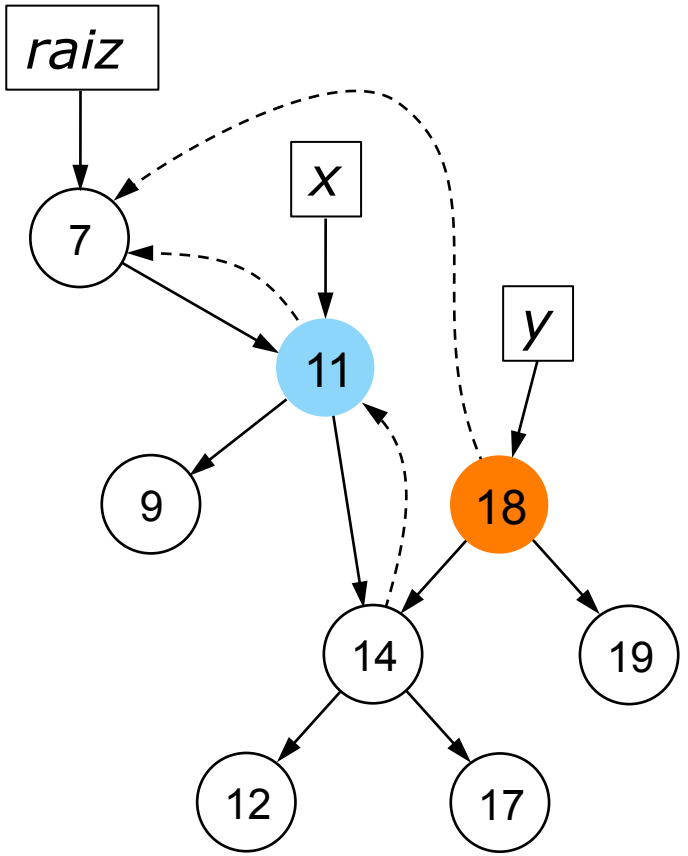
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



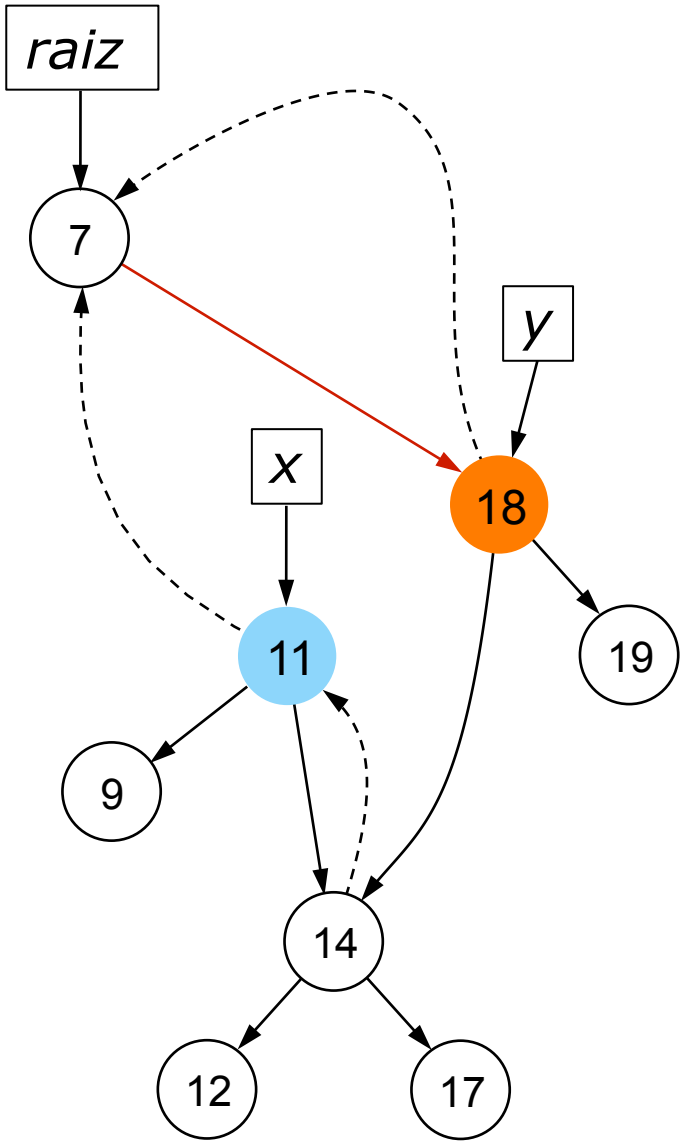
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



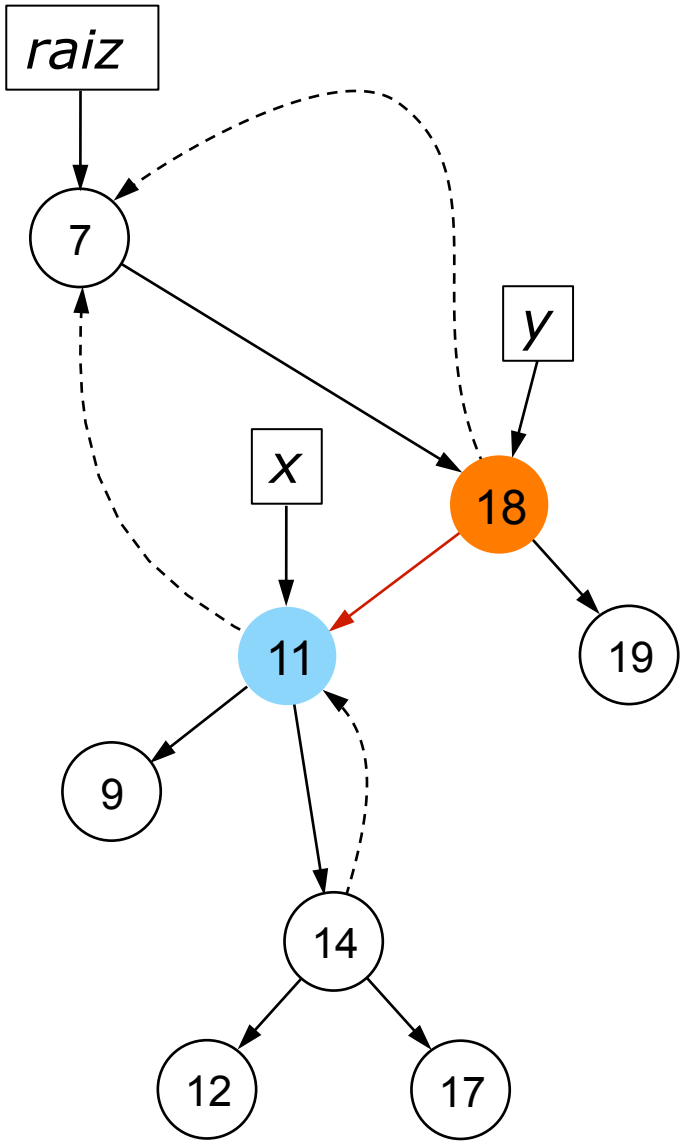
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



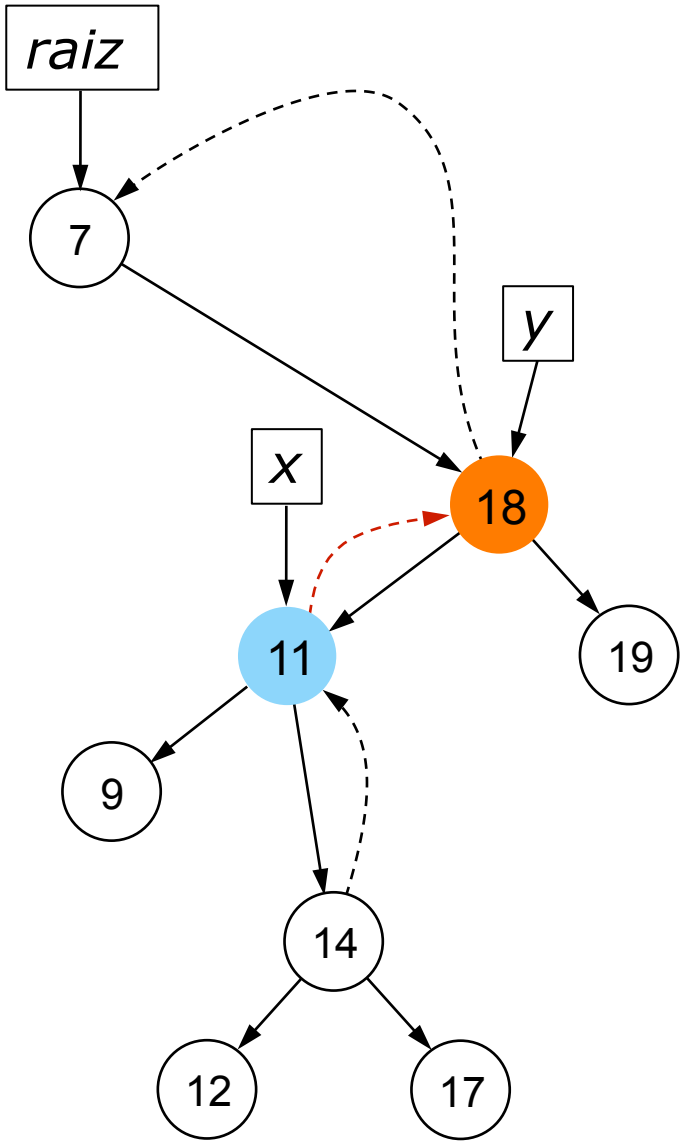
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



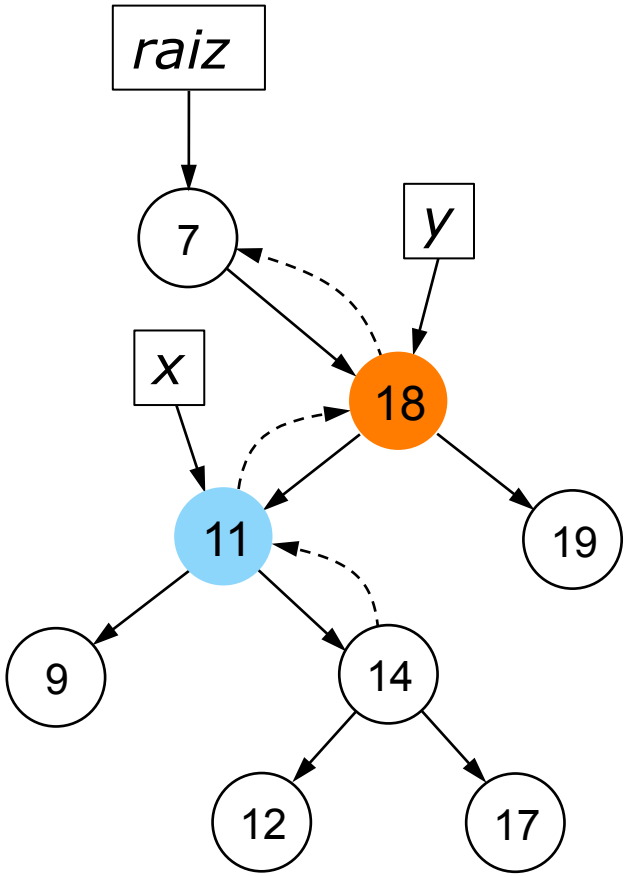
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



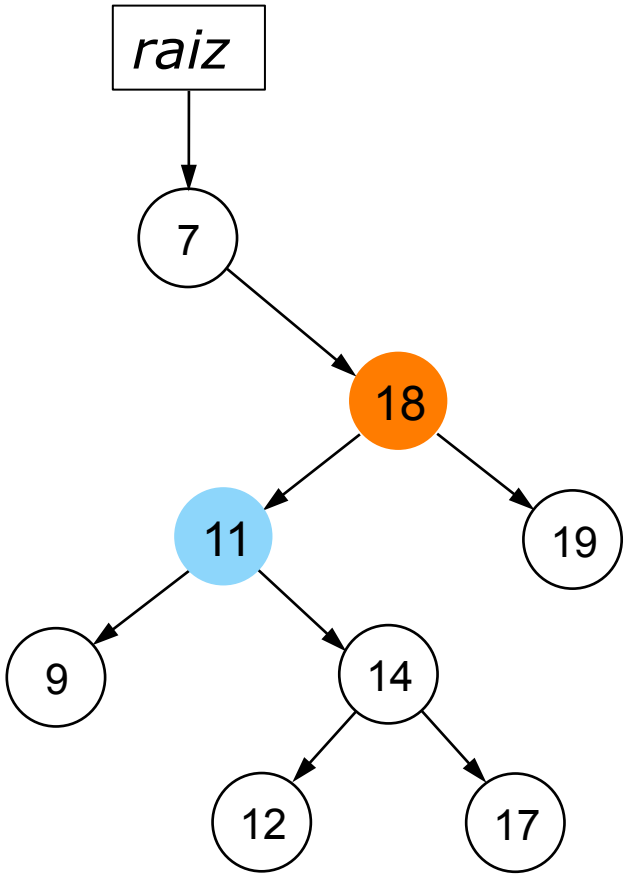
```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



```
RODA-ESQ(T, x)
1       y = x.dir
2       x.dir = y.esq
3       se y.esq ≠ T.nil
4               y.esq.pai = x
5       y.pai = x.pai
6       se x.pai == T.nil
7               T.raiz = y
8       senão se x == x.pai.esq
9               x.pai.esq = y
10      senão
11              x.pai.dir = y
12      y.esq = x
13      x.pai = y
```



Como seria o RODA-DIR?

RODA-DIR é simétrico a RODA-ESQ.

RODA-ESQ e RODA-DIR executam em tempo $O(1)$.

Somente ponteiros são modificados durante as operações; todos os demais atributos dos nós continuam os mesmos.

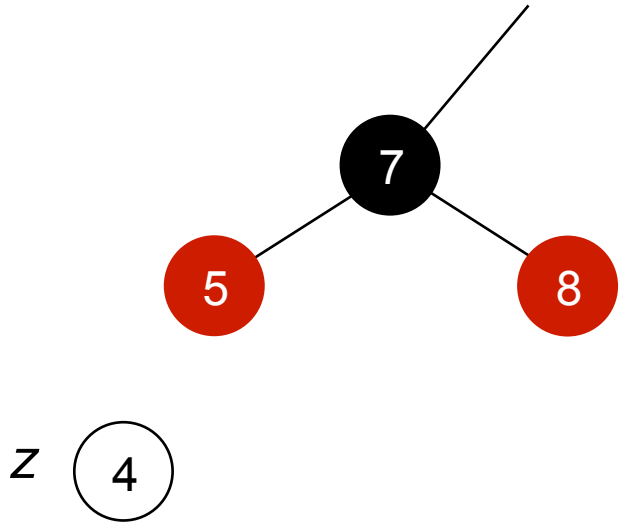
Inserção na árvore rubro-negra

É possível inserir um nó em uma árvore RN com n nós, em tempo $O(\log n)$.

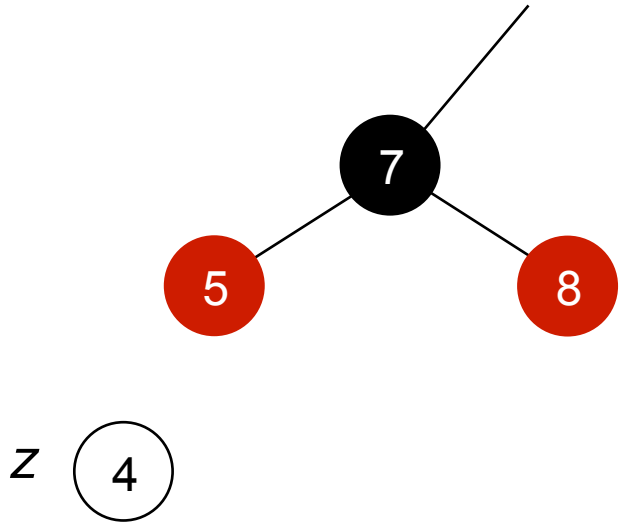
Inserção:

1. Nó z (contendo a nova chave) é inserido na árvore T como se faz em uma ABB, e z é pintado de VERMELHO.
2. Chama RN-INSERE-REPARA(T, z) para restaurar as propriedades da árvore (recolorindo nós e executando rotações).

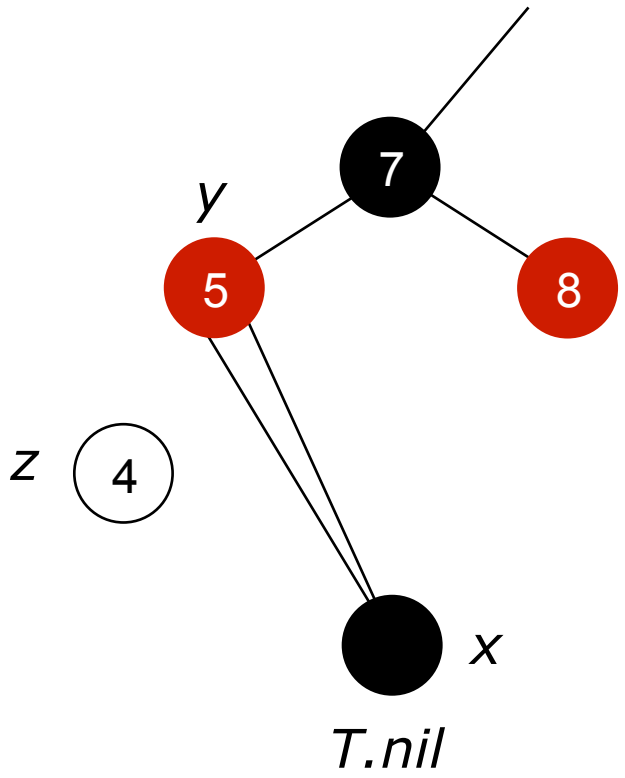
```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



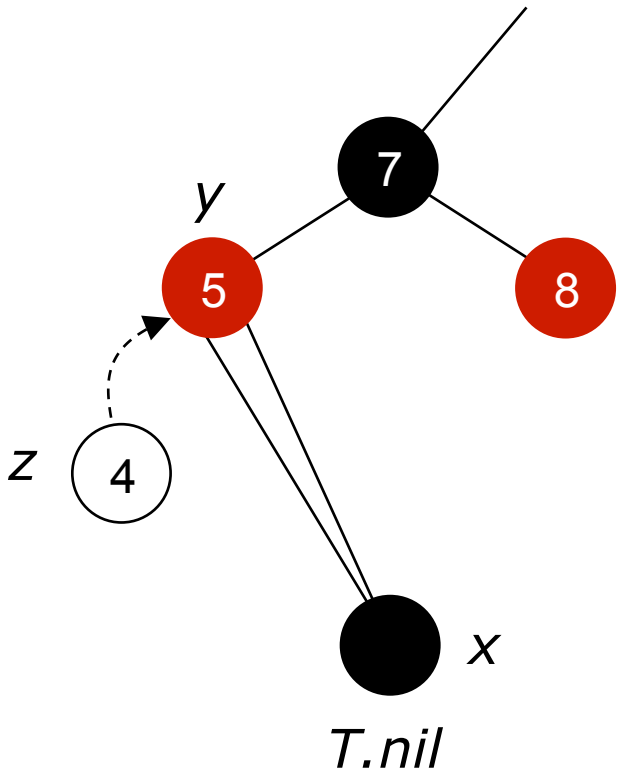
```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



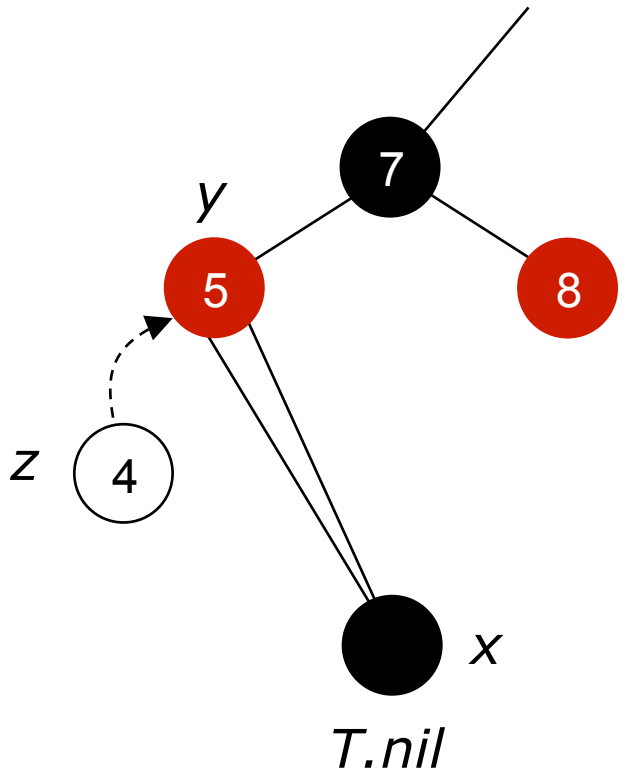
```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



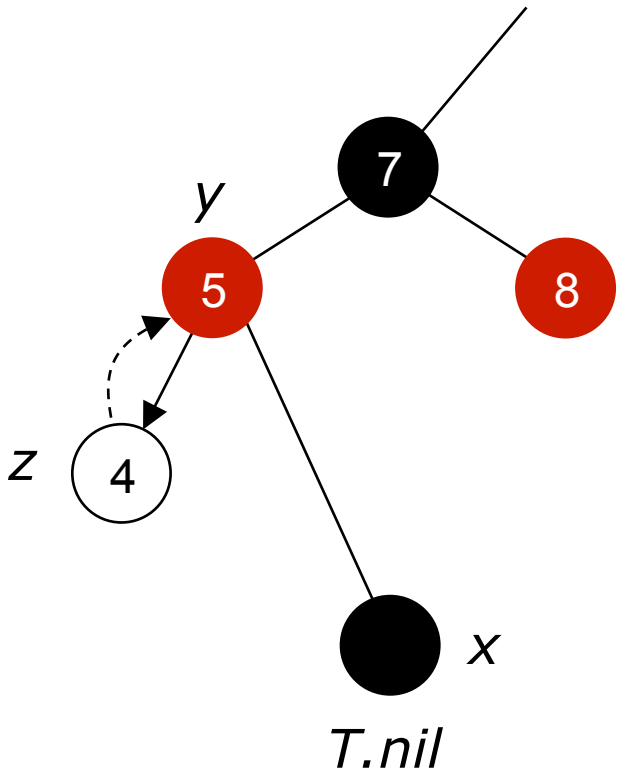
```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



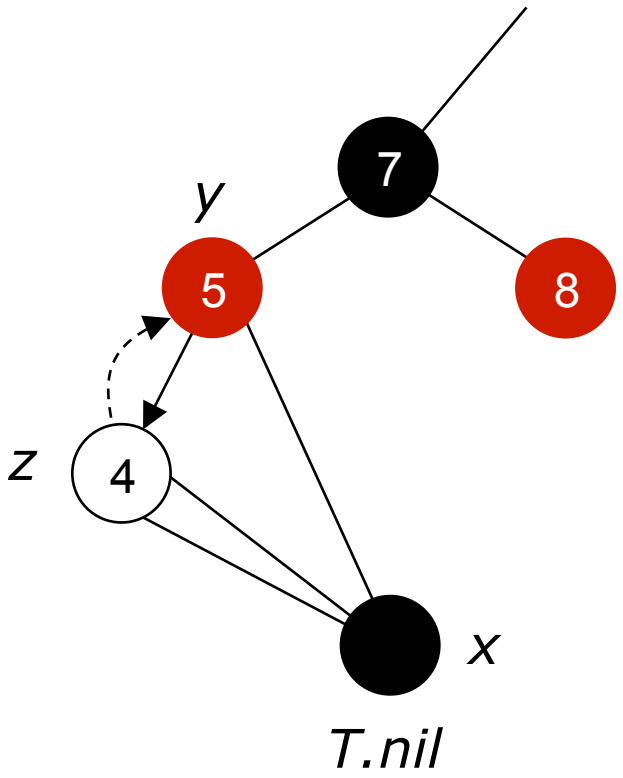
```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



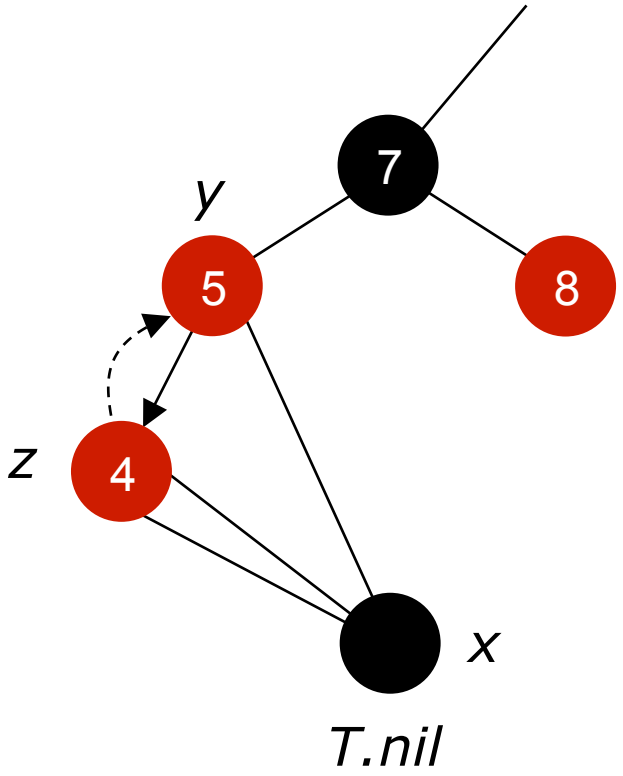
```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



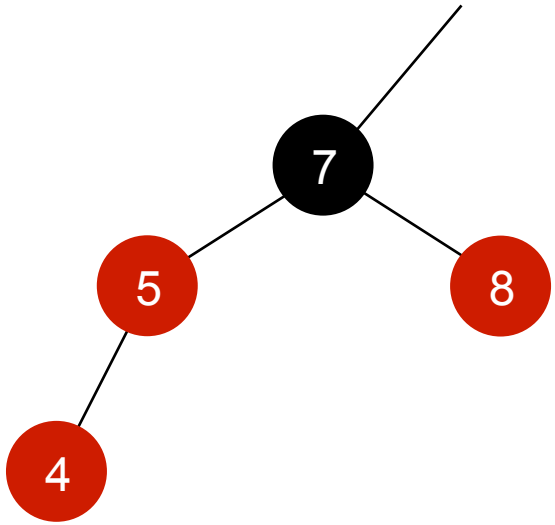
```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



```
RN-INSERE(T, z)
1       y = T.nil
2       x = T.raiz
3       while x ≠ T.nil
4               y = x
5               se z.chave < x.chave
6                       x = x.esq
7               senão x = x.dir
8       z.pai = y
9       se y == T.nil
10              T.raiz = z
11      senão se z.chave < y.chave
12              y.esq = z
13      senão y.dir = z
14      z.esq = z.esq = T.nil
15      z.cor = VERMELHO
16      RN-INSERE-RESTAURA(T, z)
```



```
RN-INSERE-RESTAURA(T, z)
1       enquanto z.pai.cor == VERMELHO
2           se z.pai == z.pai.pai.esq
3               y = z.pai.pai.dir
4               se y.cor == VERMELHO
5                   z.pai.cor = PRETO
6                   y.cor = PRETO
7                   z.pai.pai.cor = VERMELHO
8                   z = z.pai.pai
9               senão se z == z.pai.dir
10                      z = z.pai
11                      RODA-ESQ(T, z)
12                  z.pai.cor = PRETO
13                  z.pai.pai.cor = VERMELHO
14                  RODA-DIR(T, z.pai.pai)

15          senão ... // mesmo se com dir e esq trocados
16      T.raiz.cor = PRETO
```

```
RN-INSERE-RESTAURA(T, z)
1       enquanto z.pai.cor == VERMELHO
2           se z.pai == z.pai.pai.esq
3               y = z.pai.pai.dir
4               se y.cor == VERMELHO
5                   z.pai.cor = PRETO
6                   y.cor = PRETO
7                   z.pai.pai.cor = VERMELHO
8                   z = z.pai.pai
9               senão se z == z.pai.dir
10                  z = z.pai
11                  RODA-ESQ(T, z)
12              z.pai.cor = PRETO
13              z.pai.pai.cor = VERMELHO
14              RODA-DIR(T, z.pai.pai)
15          senão ... // mesmo se com dir e esq trocados
16      T.raiz.cor = PRETO
```

CASO 1

CASO 2

CASO 3

Propriedades:

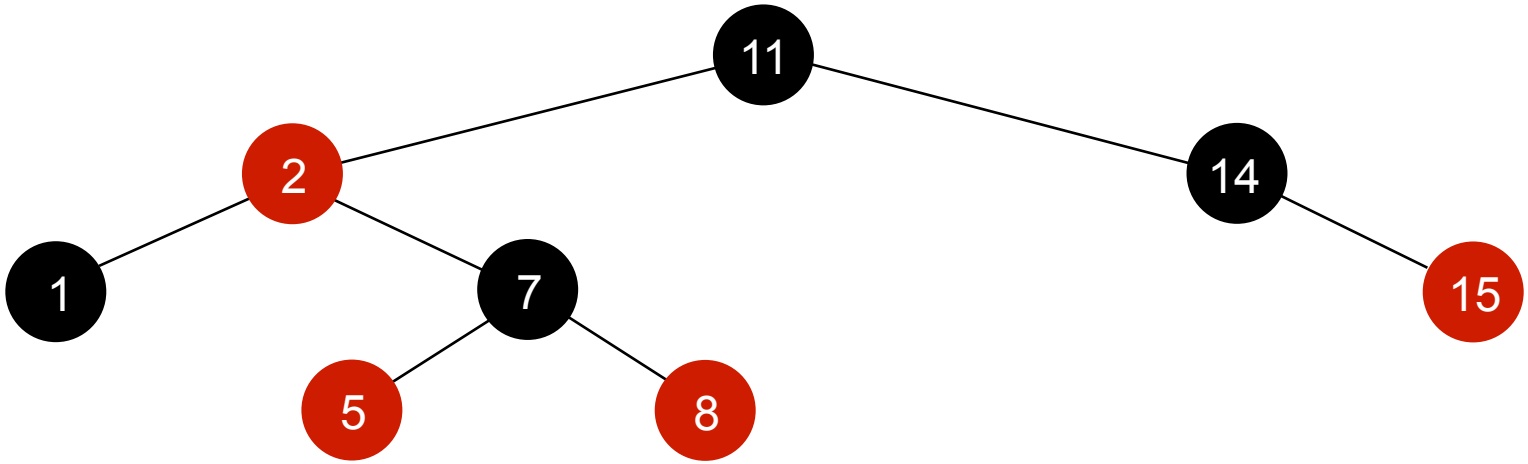
P1. Cada nó é VERMELHO ou PRETO.

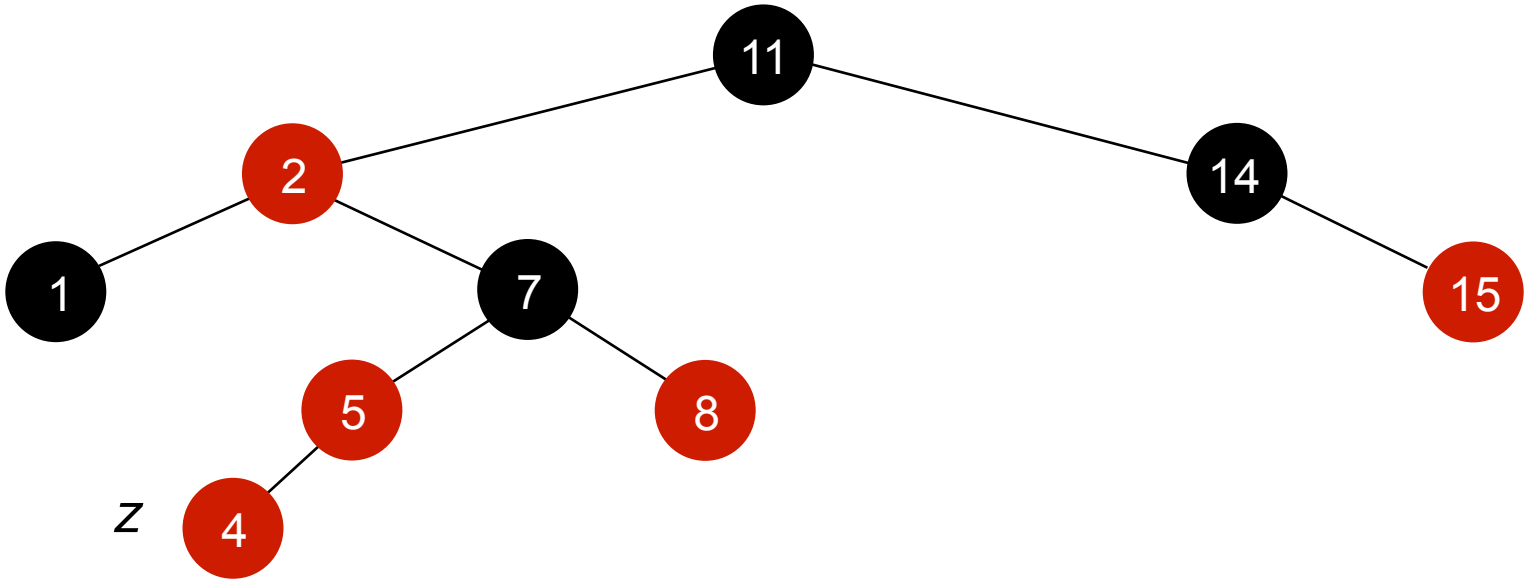
P2. A raiz é PRETA.

P3. Cada folha é PRETA.

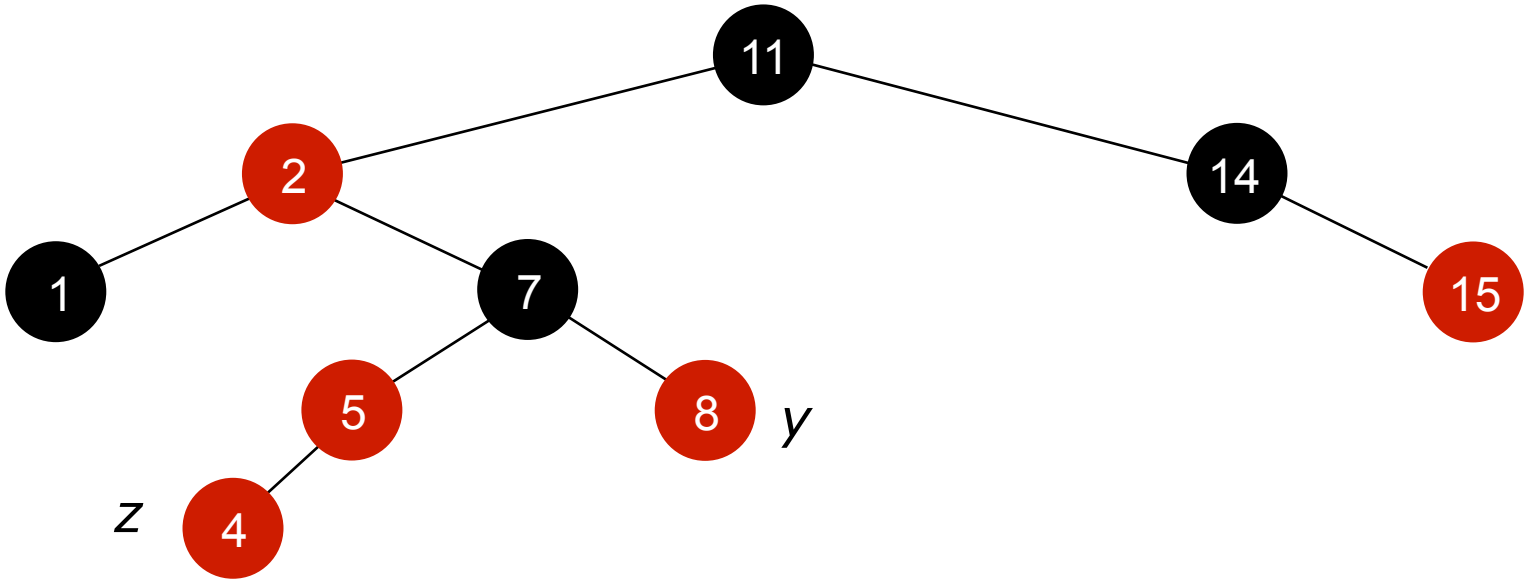
P4. Nó VERMELHO tem filhos PRETOS.

P5. O caminho simples de um nó qualquer até seus descentes folhas tem o mesmo número de nós PRETOS.


11
2
14
1
7
15
5
8



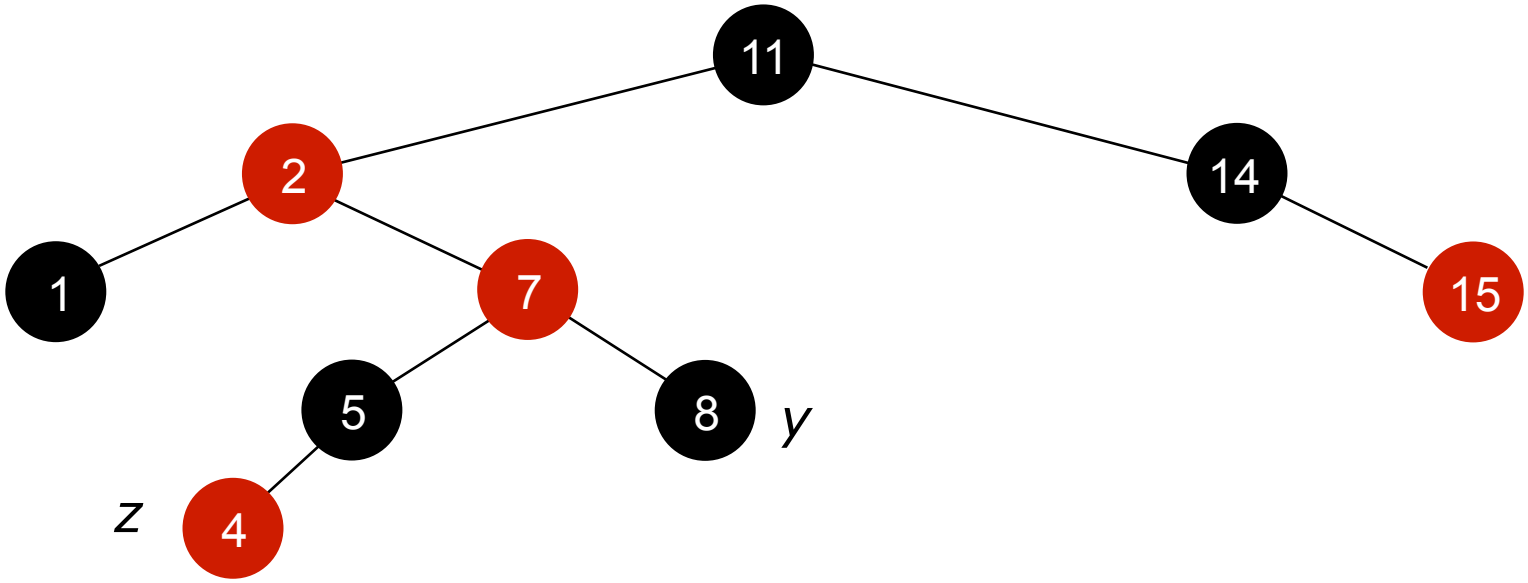
P1, **P2** e **P3** não são violadas



P4 é violada: *z* e seu pai são VERMELHOS

CASO 1: tio *y* de *z* é VERMELHO.

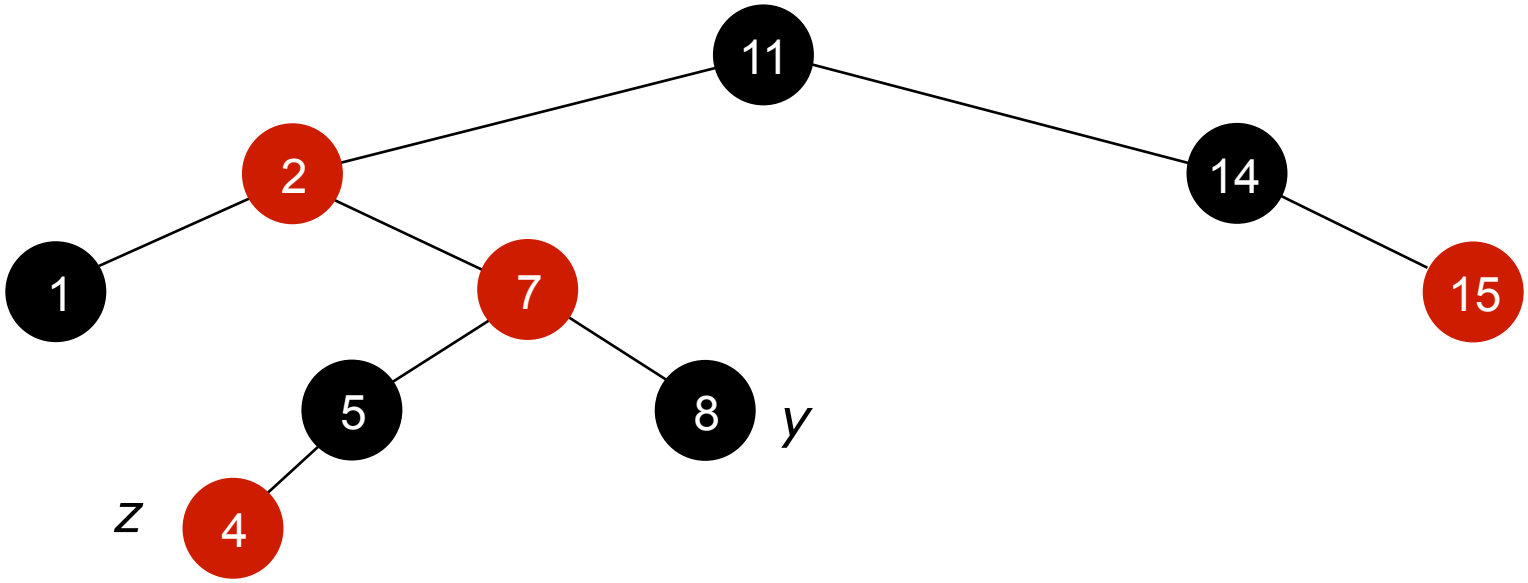
AÇÃO: (1) avô e seus filhos são recoloridos;



P4 é violada: *z* e seu pai são VERMELHOS

CASO 1: tio *y* de *z* é VERMELHO.

AÇÃO: (1) avô e seus filhos são recoloridos;

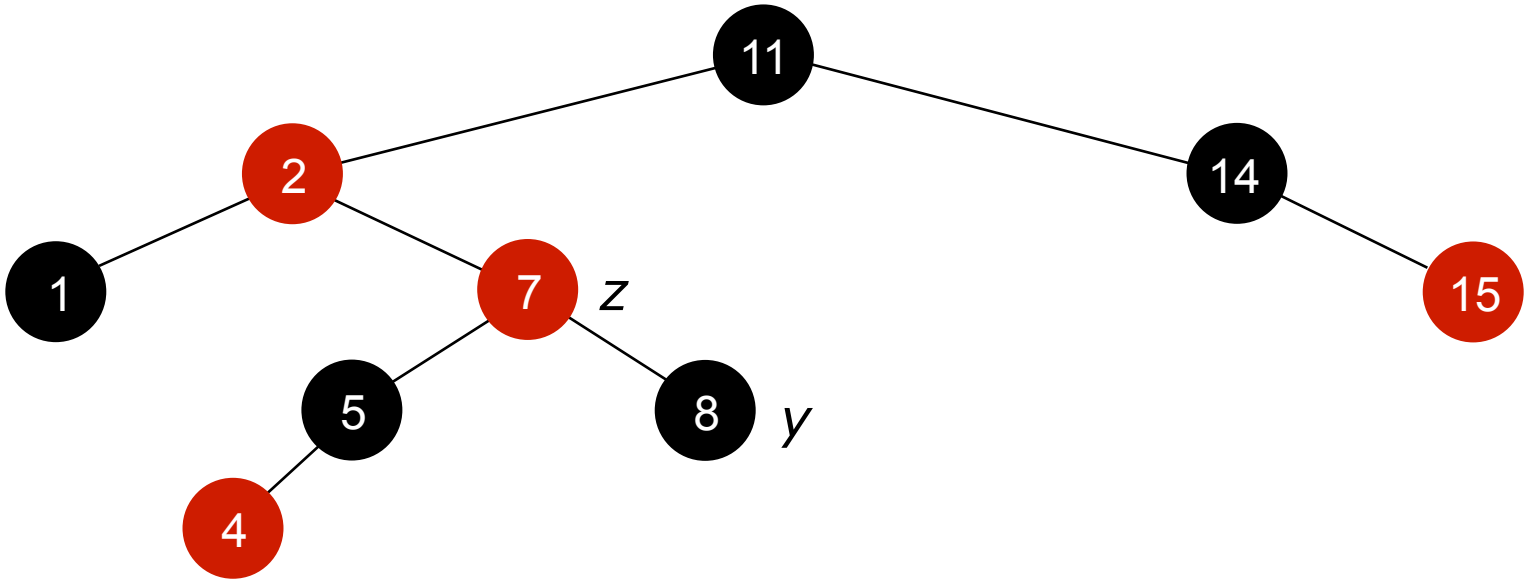


P4 é violada: *z* e seu pai são VERMELHOS

CASO 1: tio *y* de *z* é VERMELHO.

AÇÃO: (1) avô e seus filhos são recoloridos;

(2) ponteiro *z* sobe para o avô.

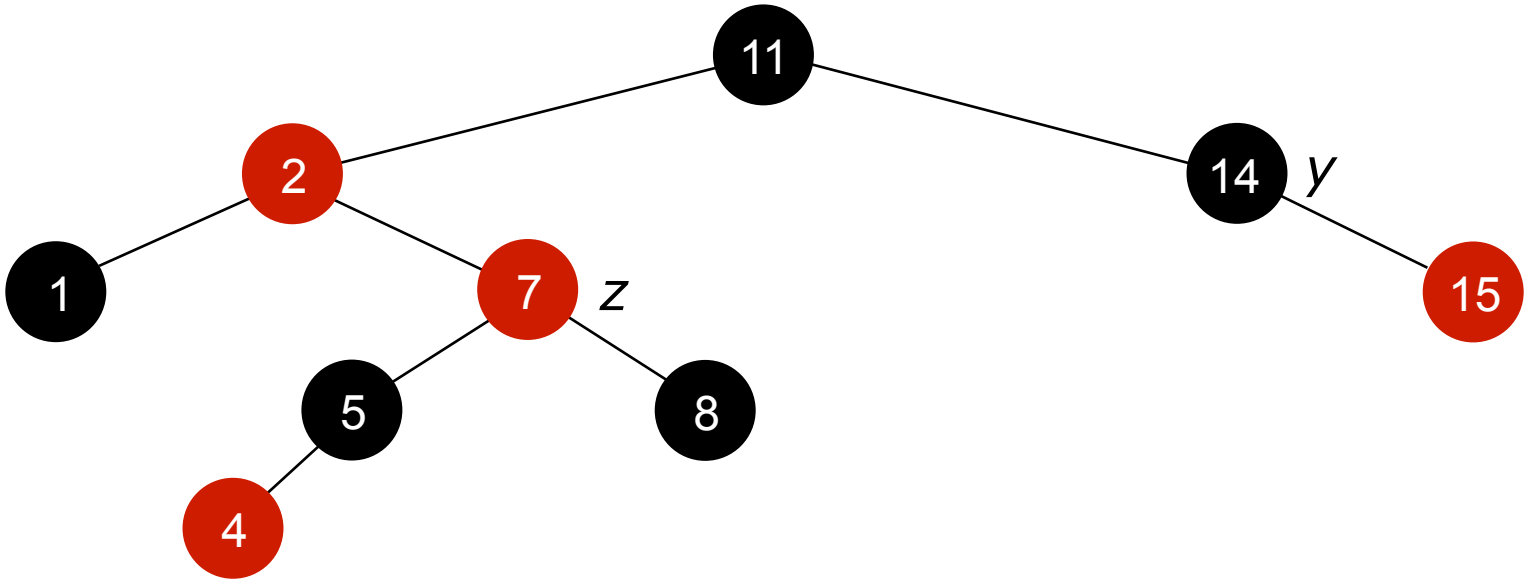


P4 é violada: *z* e seu pai são VERMELHOS

CASO 1: tio *y* de *z* é VERMELHO.

AÇÃO: (1) avô e seus filhos são recoloridos;

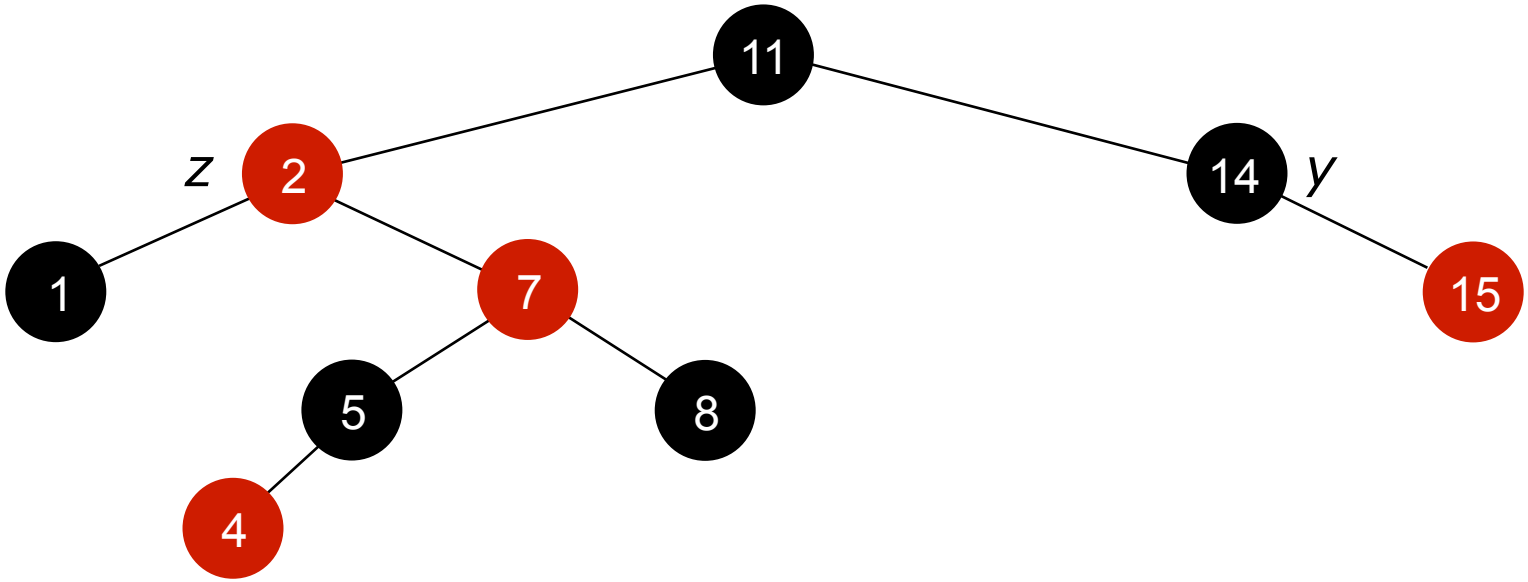
(2) ponteiro *z* sobe para o avô.



P4 é violada: *z* e seu pai são VERMELHOS

CASO 2: tio *y* é PRETO e z é filho direito.

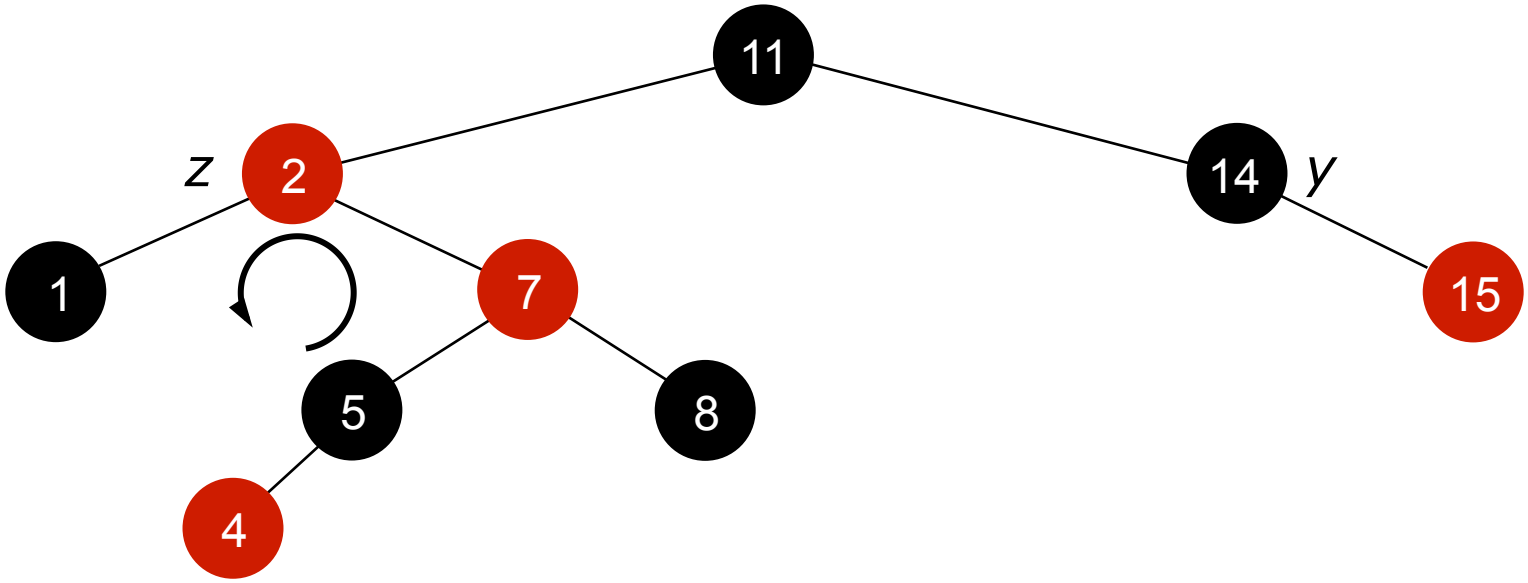
AÇÃO: (1) *z* passa a ser seu pai;



P4 é violada: *z* e seu pai são VERMELHOS

CASO 2: tio *y* é PRETO e z é filho direito.

AÇÃO: (1) *z* passa a ser seu pai;

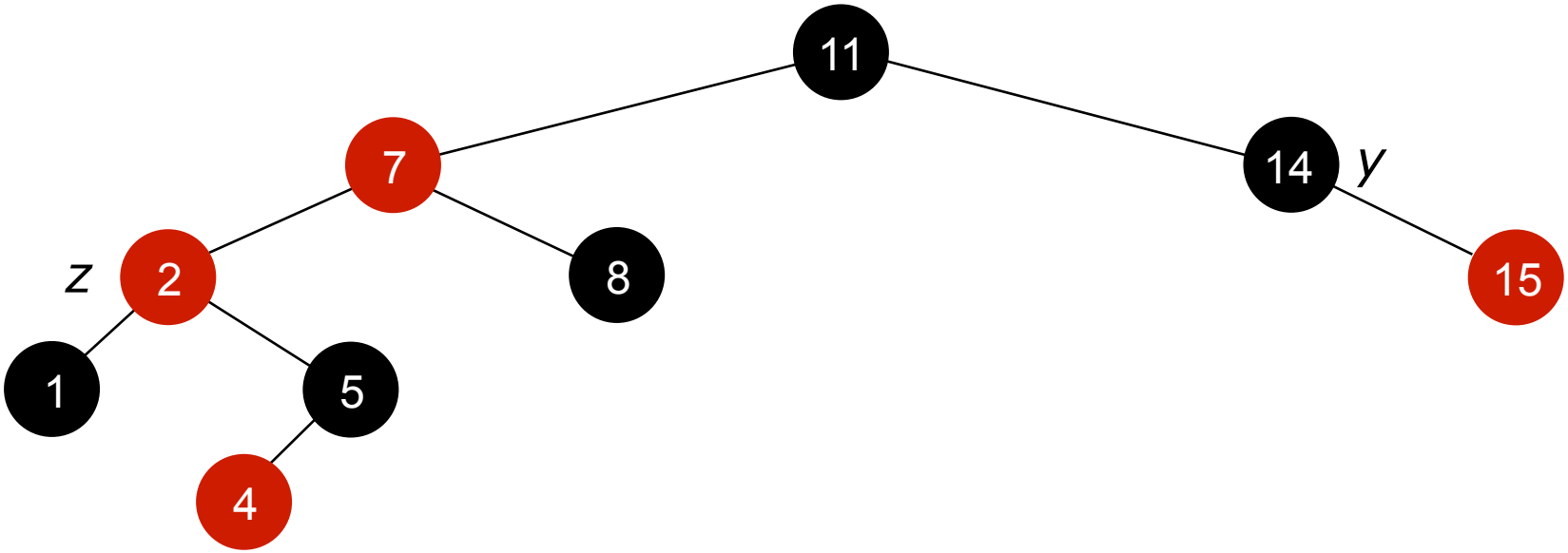


P4 é violada: *z* e seu pai são VERMELHOS

CASO 2: tio *y* é PRETO e z é filho direito.

AÇÃO: (1) *z* passa a ser seu pai;

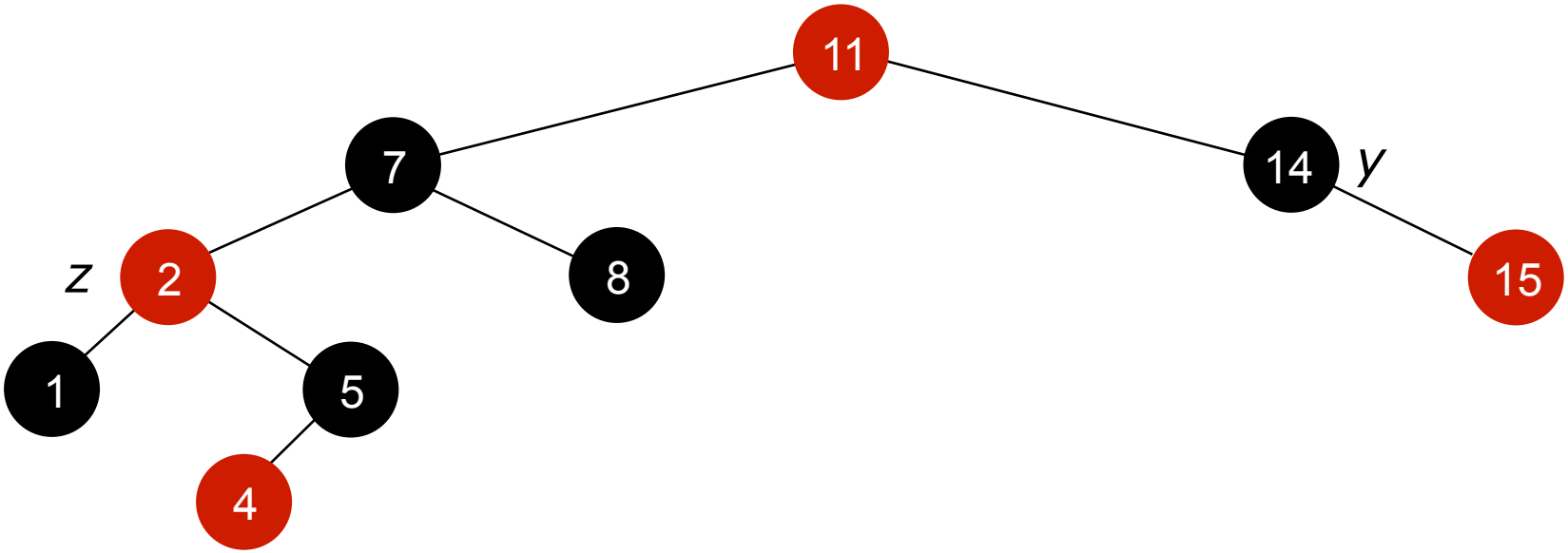
(2) rotação para a esquerda.



P4 ainda violada: *z* e seu pai são VERMELHOS.

CASO 3: tio y é PRETO e *z* é filho esquerdo.

AÇÃO: (1) recolorir o pai e o avô de *z*;

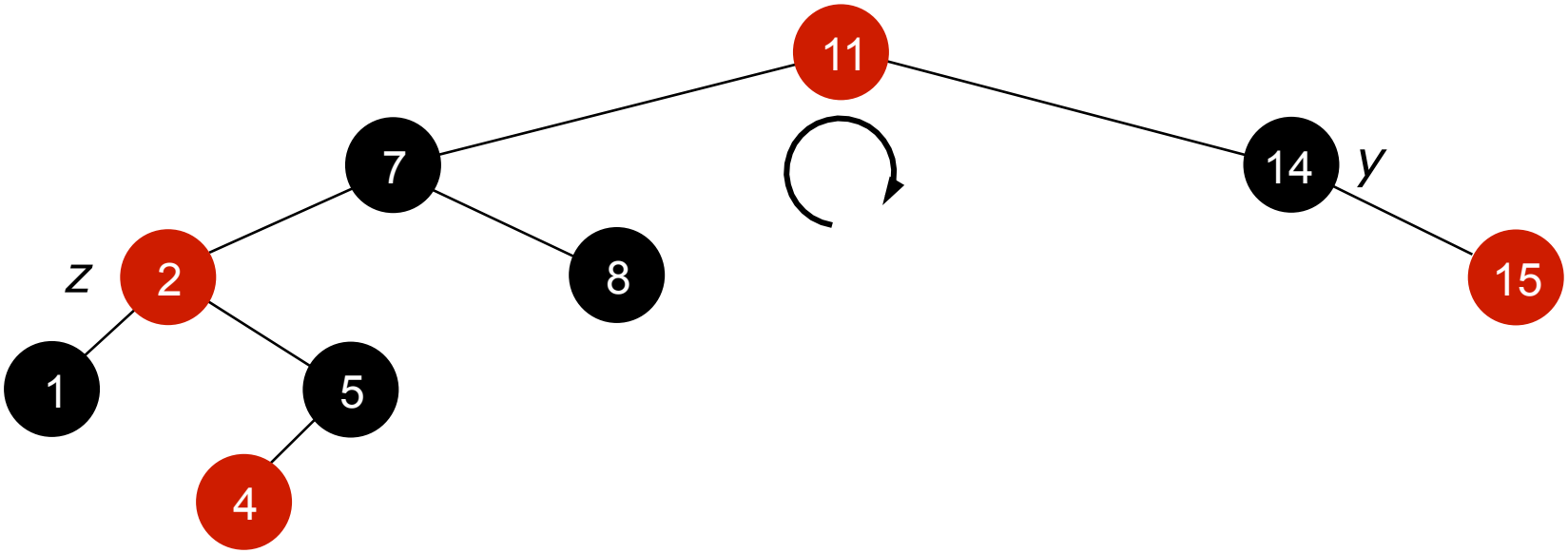


P4 ainda violada: *z* e seu pai são VERMELHOS.

CASO 3: tio y é PRETO e *z* é filho esquerdo.

AÇÃO: (1) recolorir o pai e o avô de *z*;

P2 foi violada.



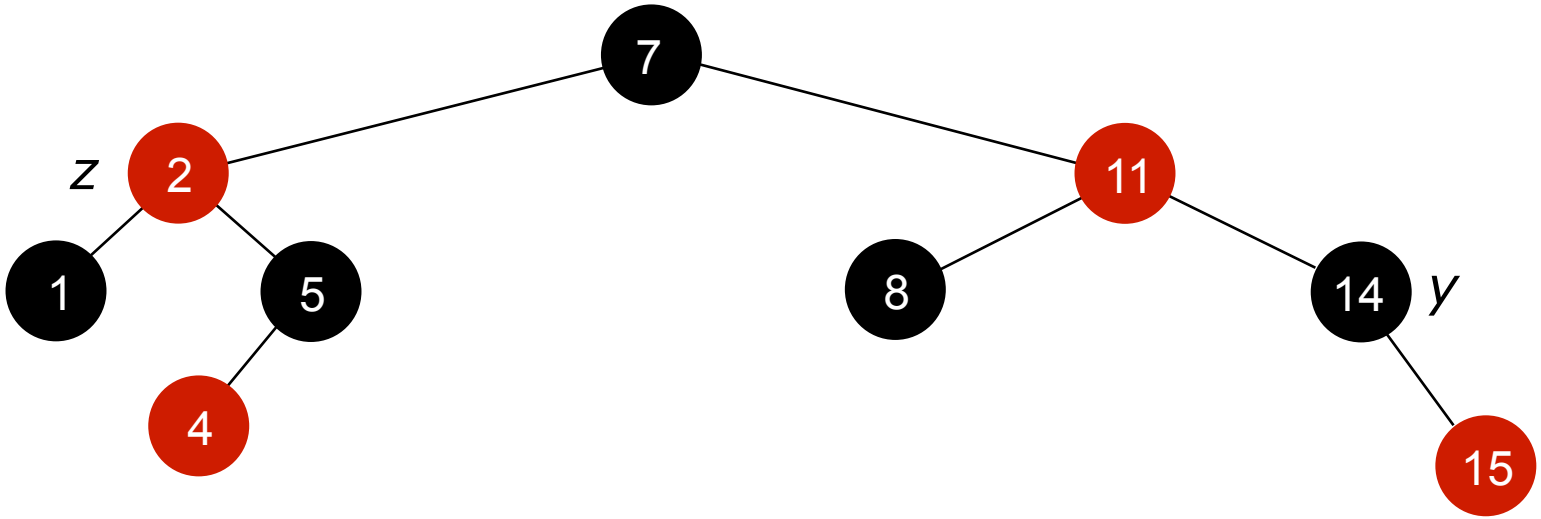
P4 ainda violada: *z* e seu pai são VERMELHOS.

CASO 3: tio y é PRETO e *z* é filho esquerdo.

AÇÃO: (1) recolorir o pai e o avô de *z*;

P2 foi violada.

(2) rotação para a direita.



Árvore volta a ser rubro-negra (balanceada).

Referências

- CORMEN, T. H., LEISERSON, C. E., RIVEST R. L., STEIN, C. Introduction do Algorithms. 3rd ed. MIT Press: 2009.
- SZWARCFITER, J. L., MARKENZON, L. Estruturas de Dados e seus Algoritmos. 3a. edição. Rio de Janeiro: LTC, 2010.