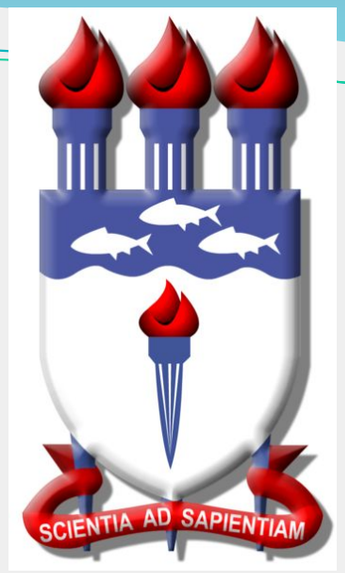


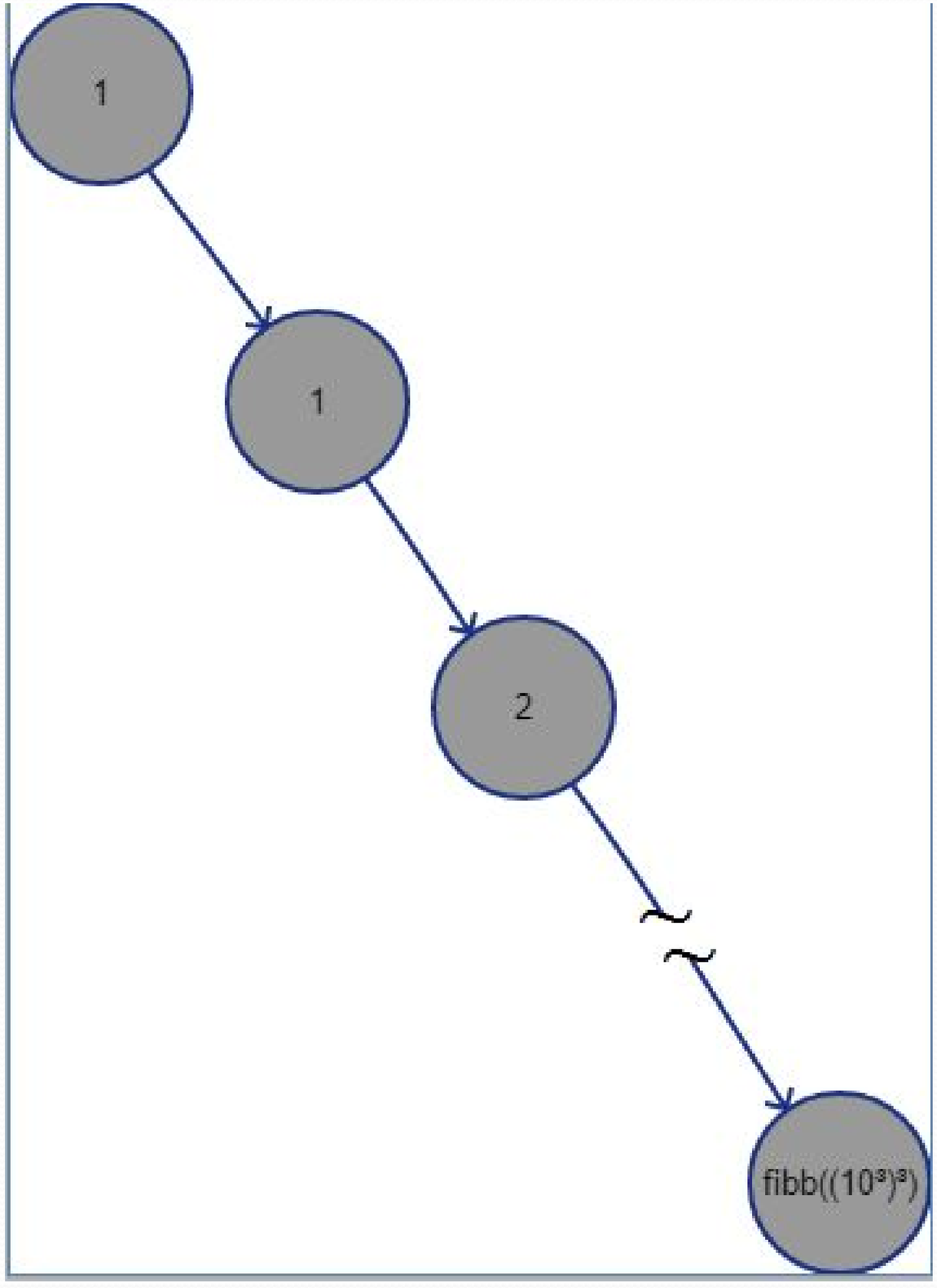
Árvore Red-Black

Alvino Lessa
Arthur Monteiro
Edvonaldo Horácio

https://github.com/4rthurmonteiro/Huffman

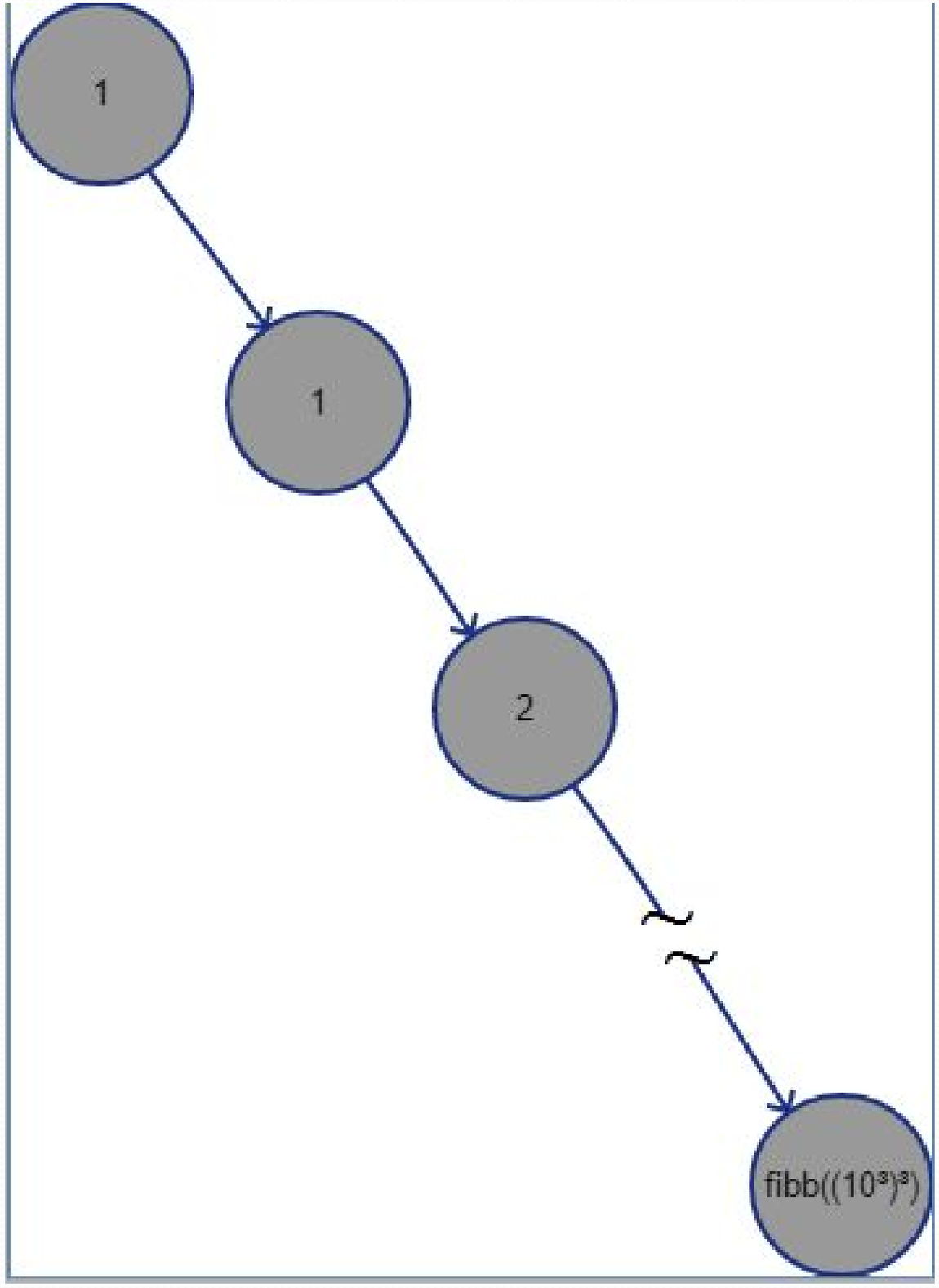
Introdução

- Inserir o termo 1 milhão na sequência 1, 1, 2, 3, 5, 7, 11 ..., inseridos nessa ordem em uma Árvore de Busca Binária.
- Qual problema nessas inserções?



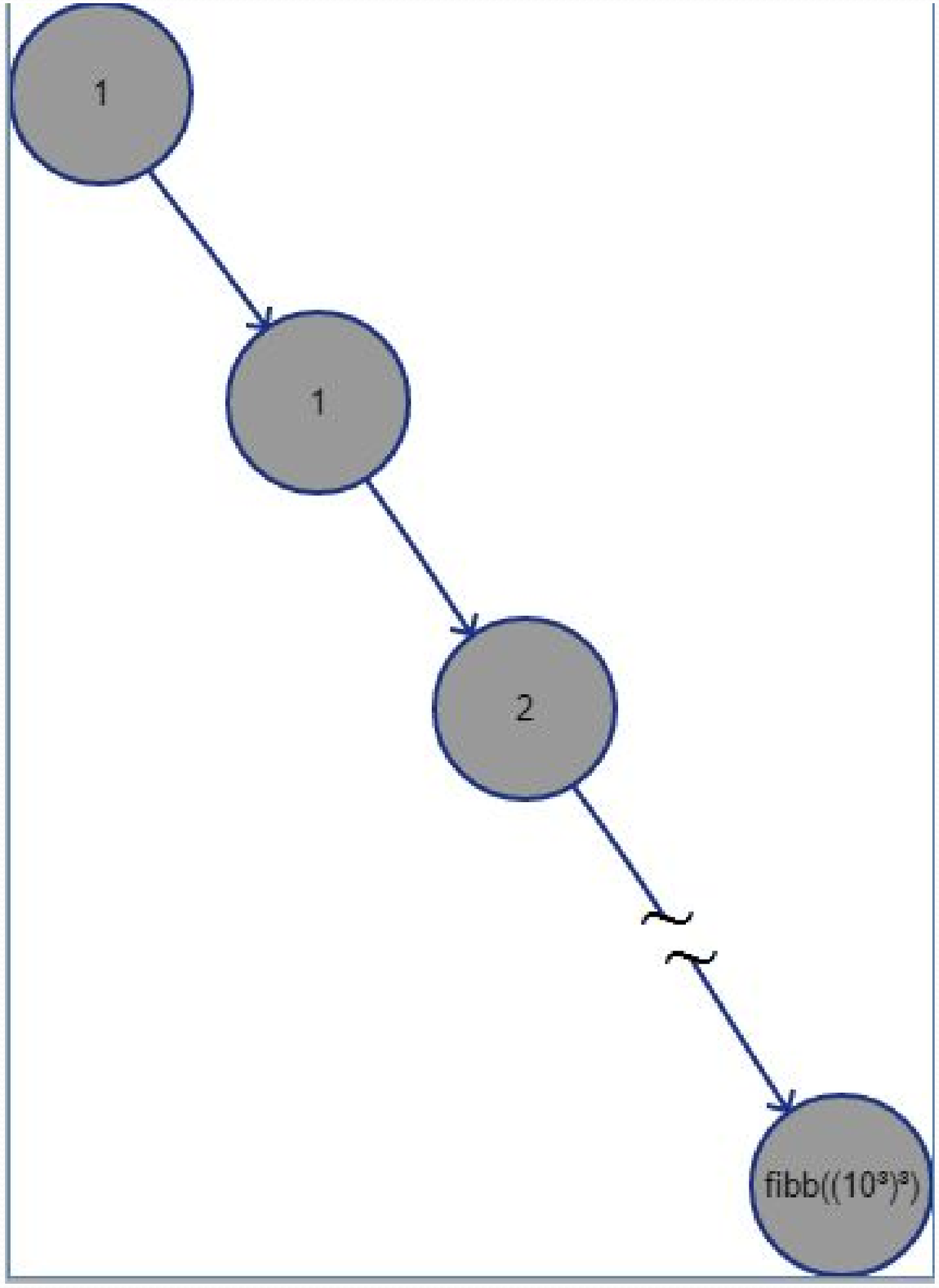
Introdução

- Deletar o termo 1 milhão na sequência 1, 1, 2, 3, 5, 7, 11 ..., inseridos nessa ordem em uma Árvore de Busca Binária.
- Qual o problema nessas deleções?



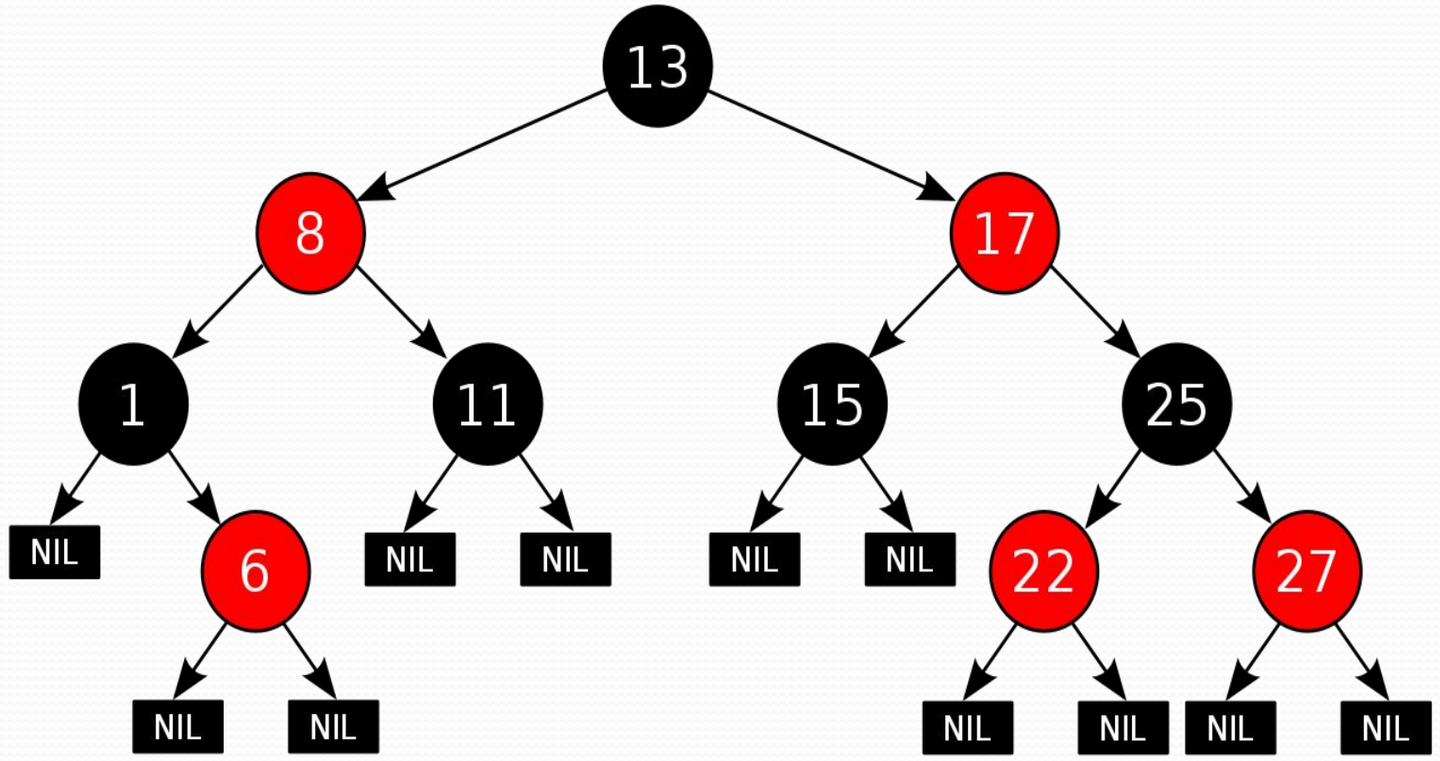
Introdução

- Buscar o termo 1 milhão na sequência 1, 1, 2, 3, 5, 7, 11 ..., inseridos nessa ordem em uma Árvore de Busca Binária.
- Qual o problema dessa busca?



Árvore Red-Black (Rubro-Negra)

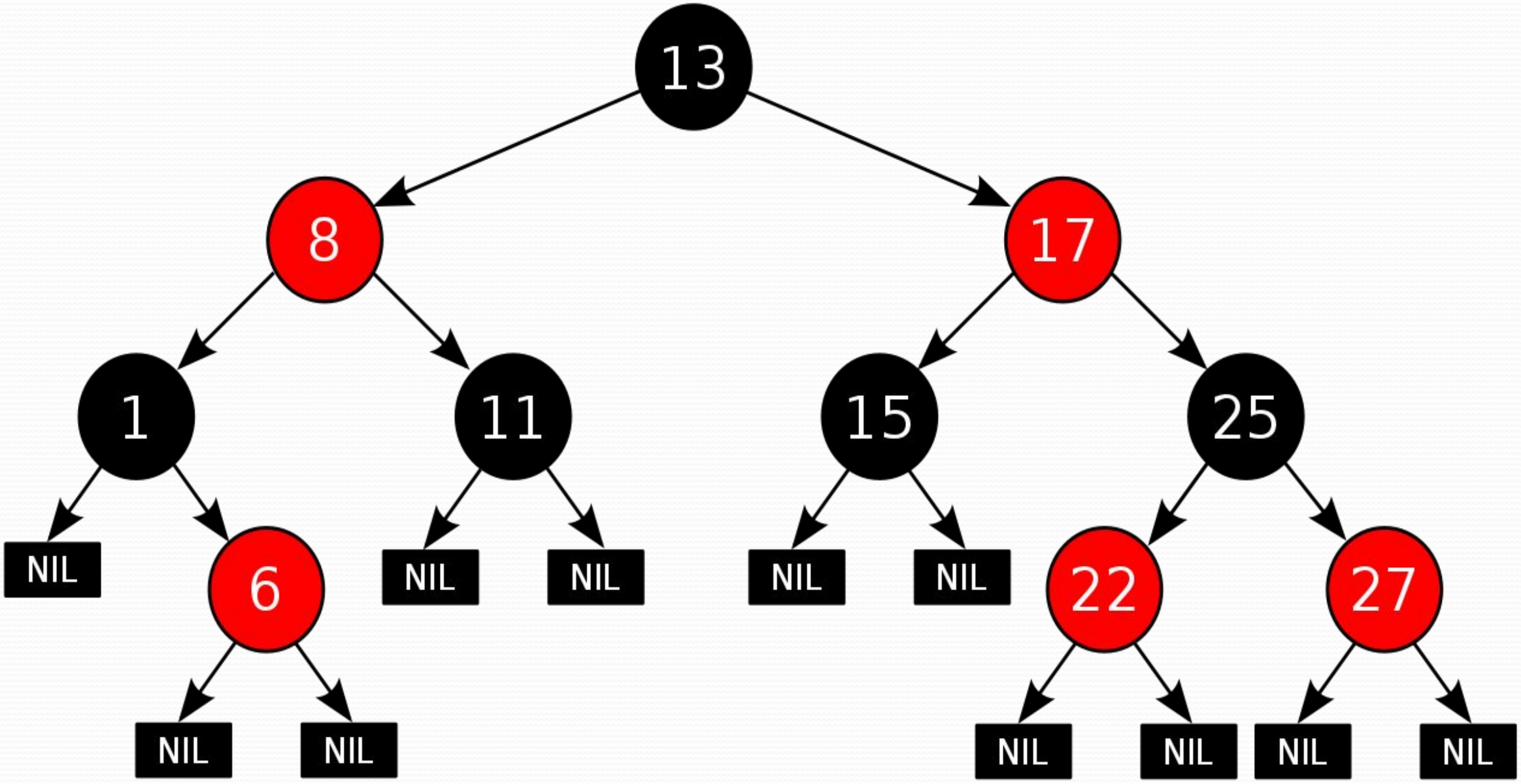
- Árvore de busca binária balanceada com algumas características especiais.
- Criada em 1972 por Rudolf Bayer (Árvores Binárias B simétricas).
- Inserção e remoção mais rápidas que em uma AVL.



Propriedades da Árvore Red-Black

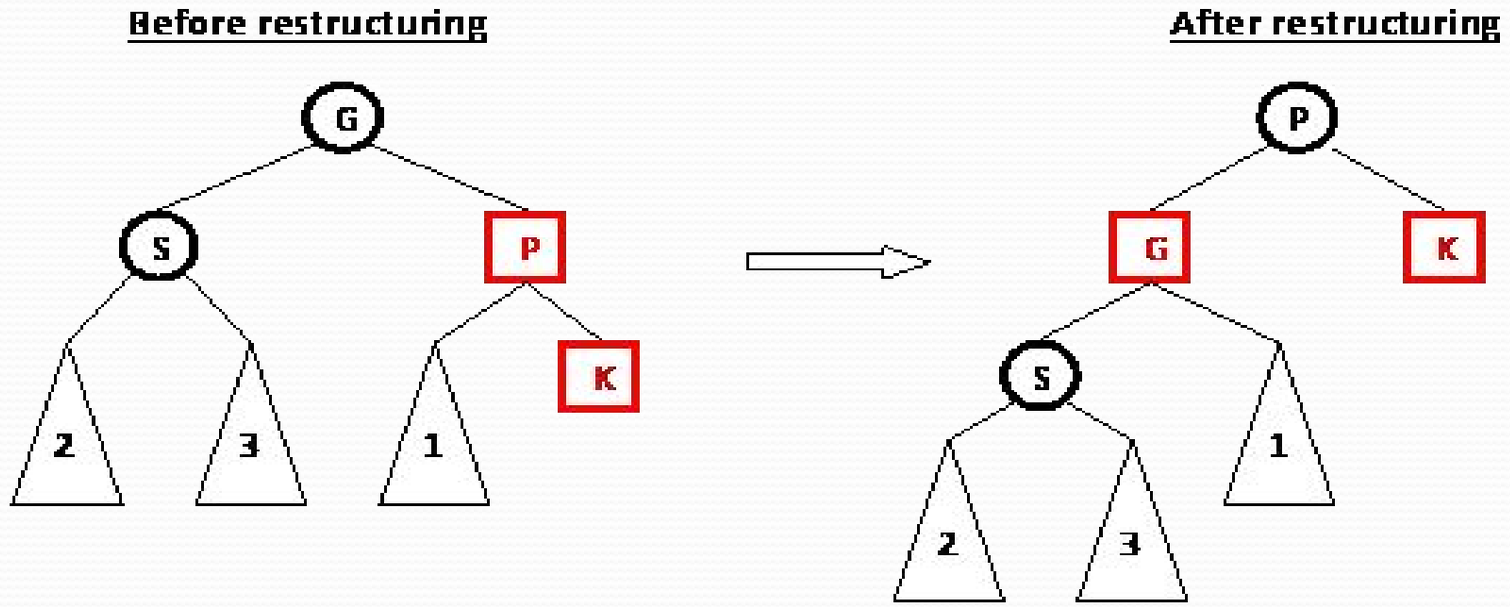
1. Todo nó é vermelho ou preto.
2. A raiz é preta.
3. Toda a folha (NIL) é preta.
4. Se um nó é vermelho, então os seus filhos são pretos.
5. Para cada nó, todos os caminhos simples do nó até folhas descendentes contêm o mesmo número de nós pretos.

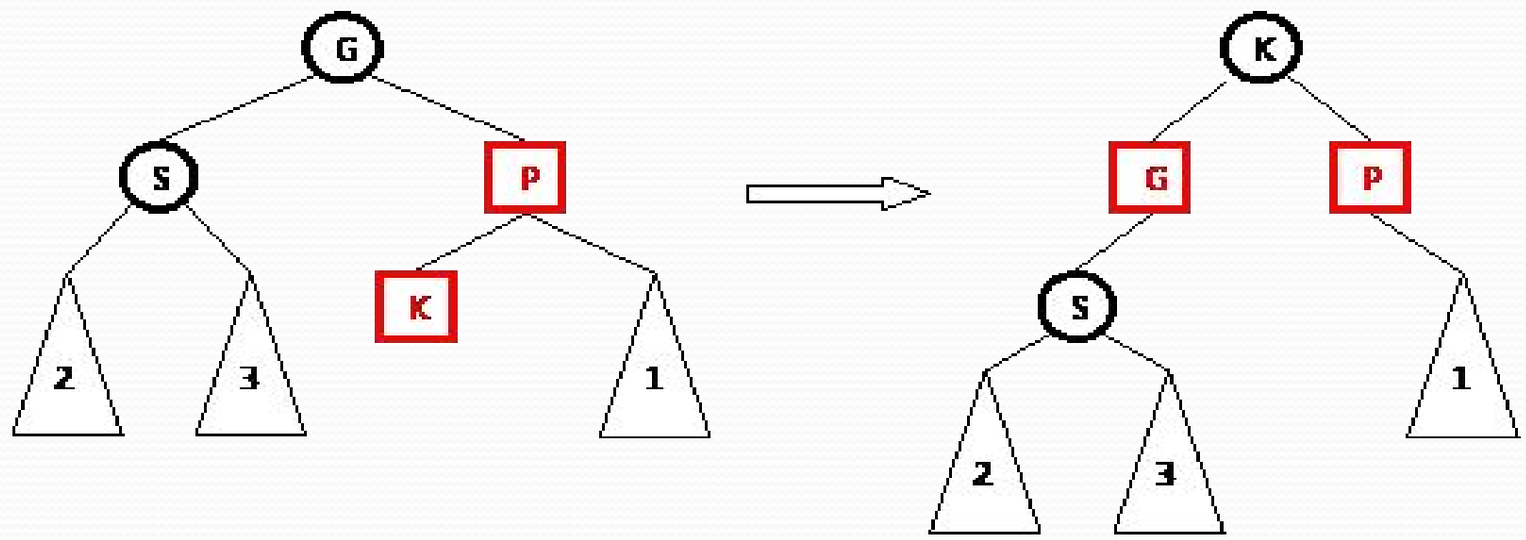
Árvore Red-Black



Operações na Red-Black

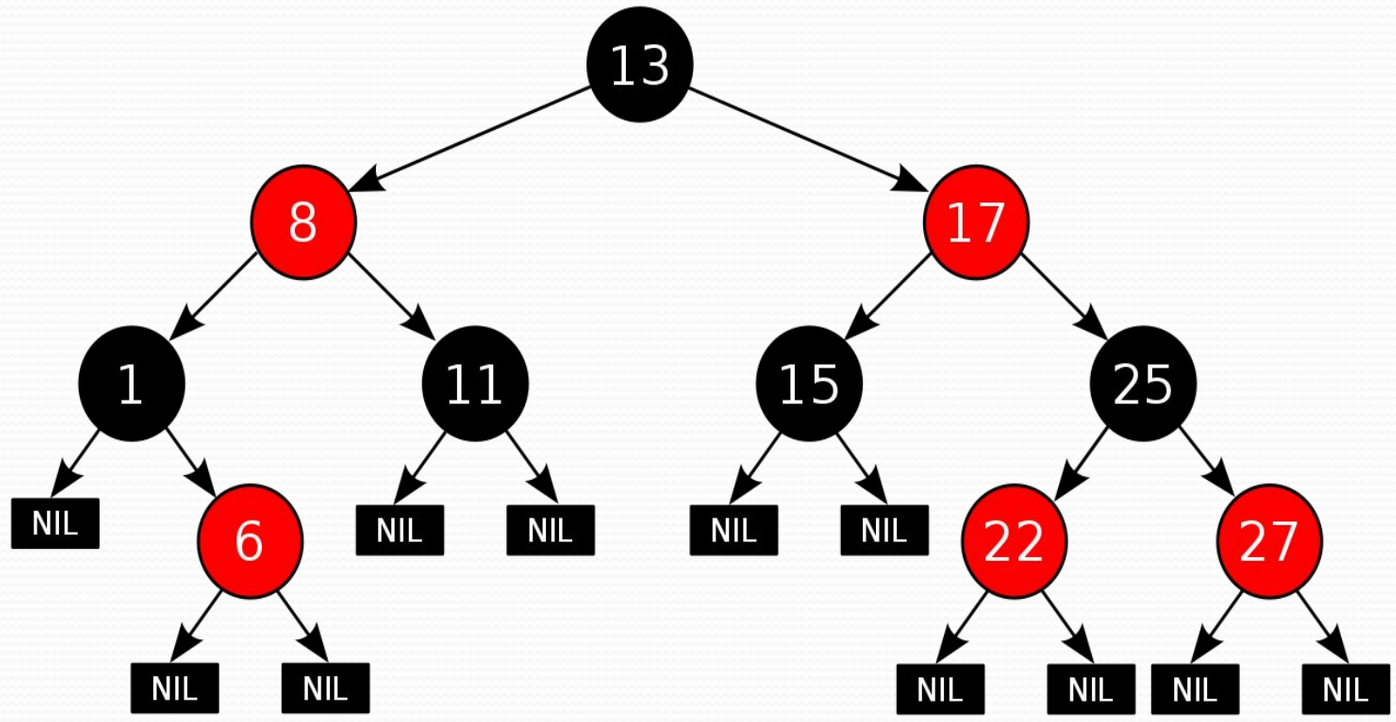
- Operações
  - **Rotação**
  - **Busca**
  - **Inserção**
  - **Remoção**





Árvore Red-Black

Lema: Uma árvore red-black com *n* nós internos têm no máximo uma altura *2lg(n+1)*.



Árvore Red-Black

- **Demonstração:**

  -Altura preta bh(x)

  -Hipótese: n >= 2^(bh(x)) -1

  1. Caso base: T.nil

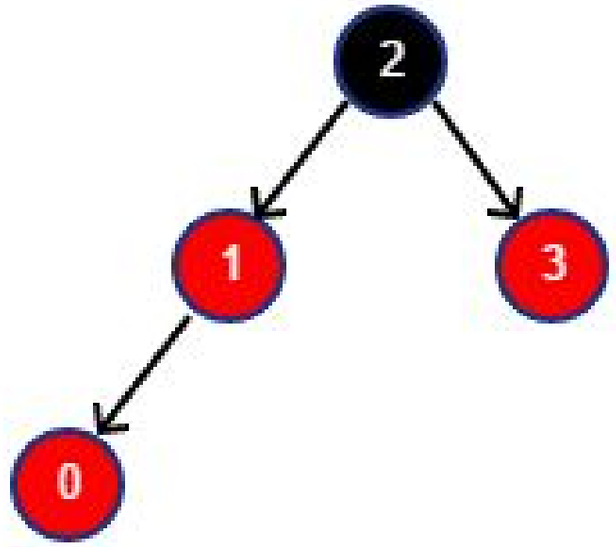
  2. Hipótese: nó x com subárvores de altura bh(x) -1, com 2^((bh(x)-1) -1 nós.

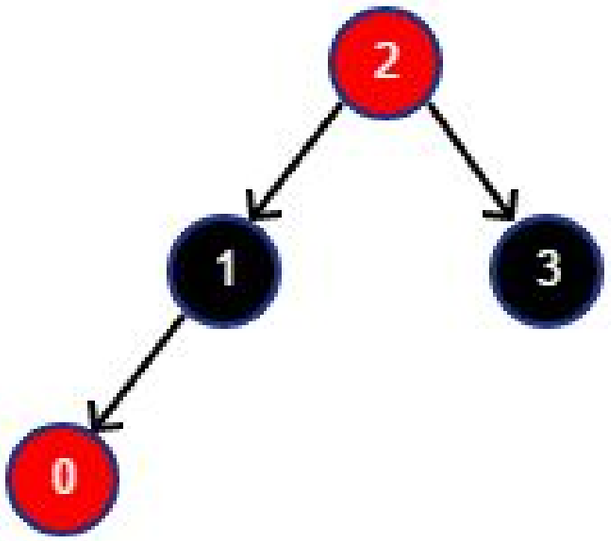
  3. Duas subárvores, então 2*(2^((bh(x)-1) -1) + 1(T.nil é pai de x).

  4. Manipular n >= n >= 2^(bh(x)) -1.

Troca de cor

```
/*
     *
     *Função para trocar de cor na Red-Black Tree
     *
*/
void change_color(node_tree_rb *node)
{
     node->color = !node->color;
     if(node->left != NULL)
     {
          node->left->color = !node->left->color;
     }
     if(node->right != NULL)
     {
          node->right->color = !node->right->color;
     }
}
```

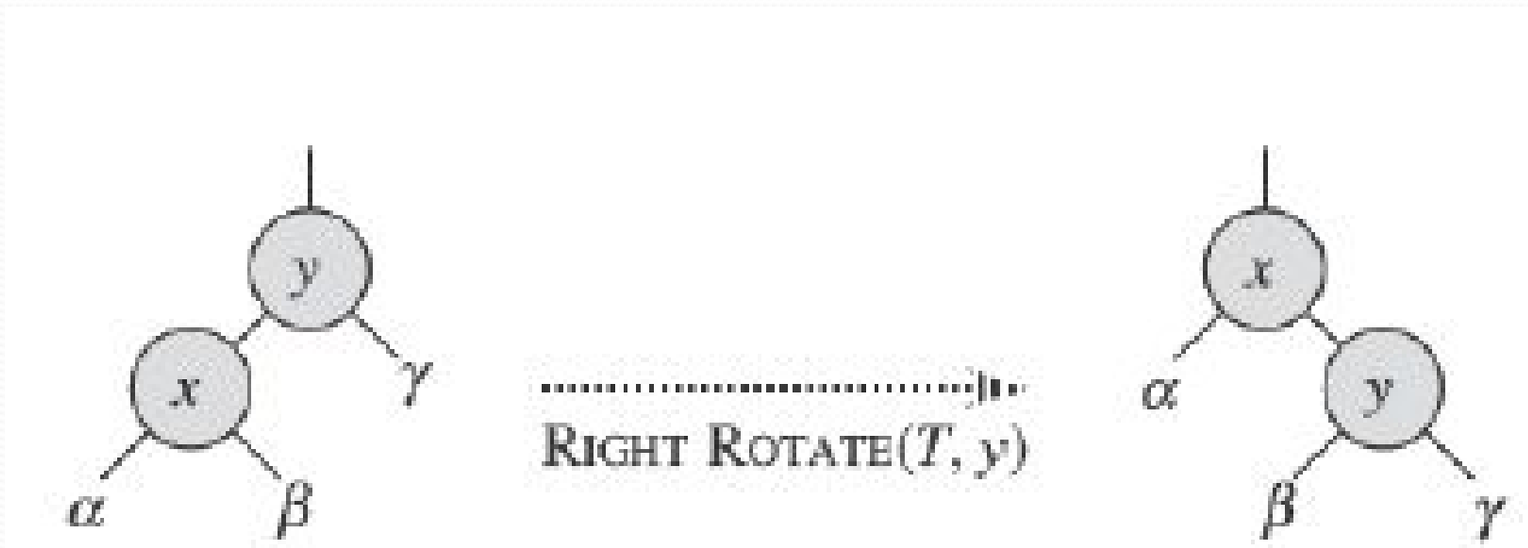




Rotação

1. Rotação à direita

```
RIGHT-ROTATE(T; y)
 1  x = y.left
 2  y.left = x.right
 3  if  x.right != T.nil
 4      x.right.p = y
 5  x.p = y.p
 6  if  y.p == T.nil
 7      T.root = x
 8  elseif  y == y.p.right
 9      y.p.right = x
10  else y.p.left = x
11  x.right = y
12  y.p = x
```



Rotação

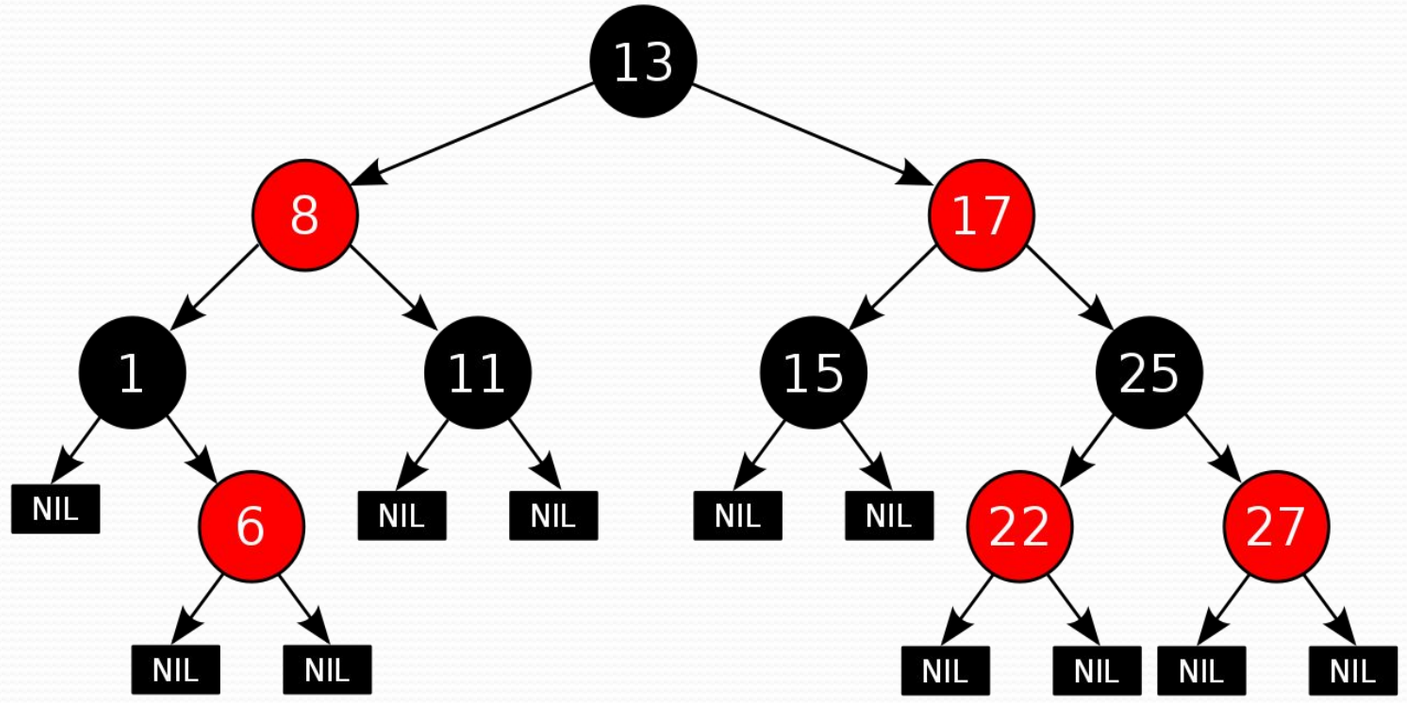
2. Rotação à esquerda.

```
LEFT-ROTATE(T, x)
 1  y = x.right
 2  x.right = y.left
 3  if y.left ≠ 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 x.p.right = y
11  y.left = x
12  x.p = y
```

Busca

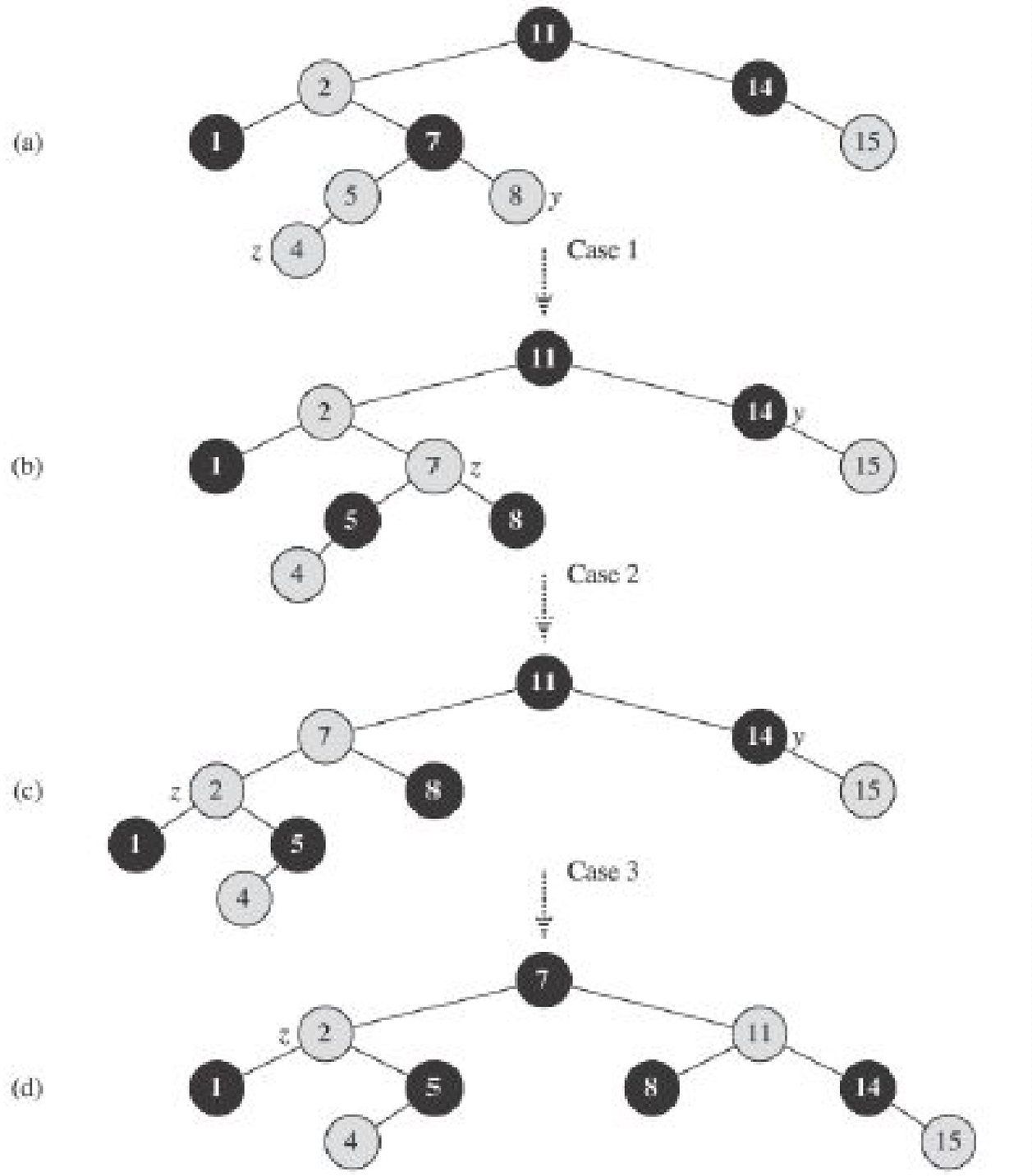
- Equivalente à uma busca numa Árvore de Busca Binária.

```
RB-TREE-SEARCH (x, k)
1 if x == T.nil or k == x.key
2       return x
3 if k < x.key
4       return RB-TREE-SEARCH (x.left, k)
5 else return RB-TREE-SEARCH (x.right, k)
```



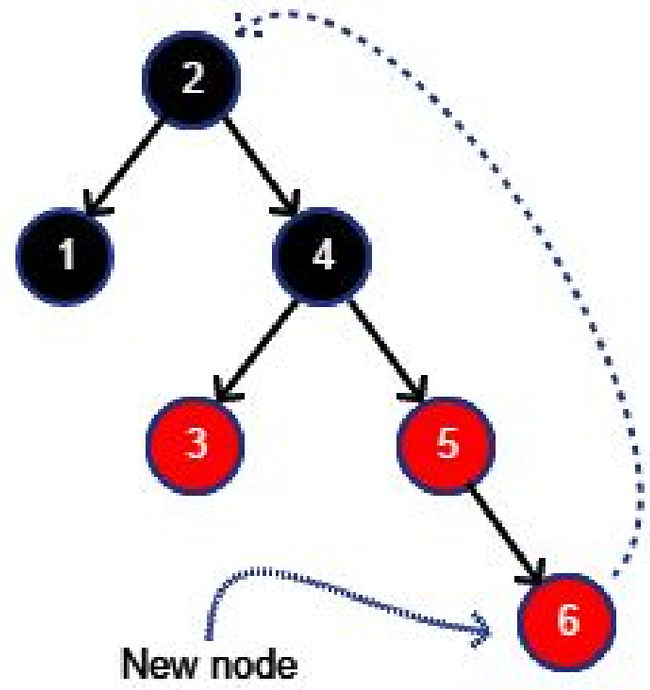
Inserção

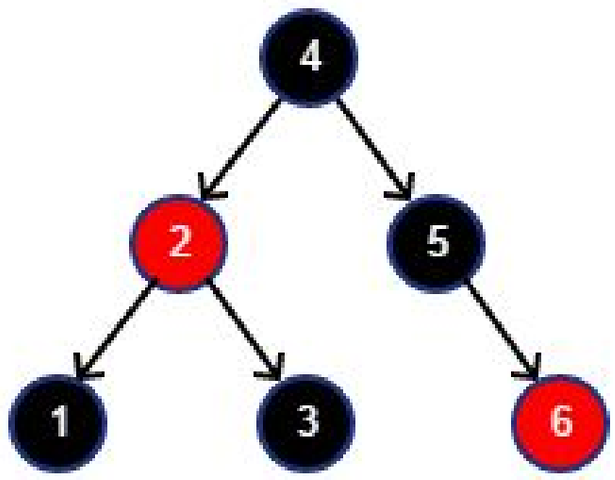
- Caso 1: o tio de z é y e vermelho;
- Caso 2: o tio de z é y e preto e z é um filho à direita;
- Caso 3: o tio de z é y e preto e z é um filho à esquerda;



Inserção

```
RB-INSERT(T, z)
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4        y = x
 5        if z.key < x.key
 6              x = x.left
 7        else x = x.right
 8  z.p = y
 9  if y == T.nil
10        T.root = z
11  elseif z.key < y.key
12        y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```





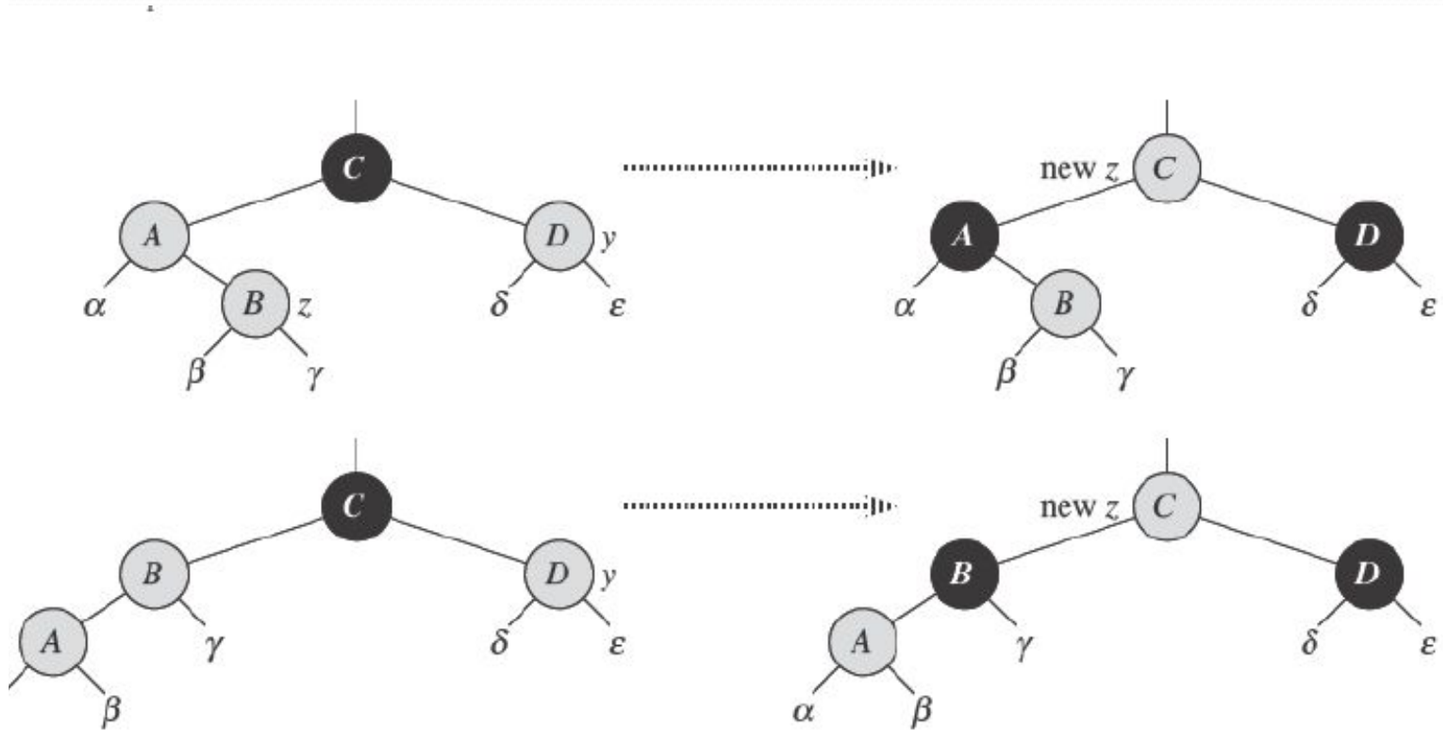
Inserção

- Como manter as propriedades da Red-Black?
- RB-Insert-Fixup(T,z) !

```
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                            // case 1
 9          else if z == z.p.right
10                  z = z.p                          // case 2
11                  LEFT-ROTATE(T, z)                // case 2
12              z.p.color = BLACK                    // case 3
13              z.p.p.color = RED                    // case 3
14              RIGHT-ROTATE(T, z.p.p)               // case 3
15      else (same as then clause
                with "right" and "left" exchanged)
16  T.root.color = BLACK
```

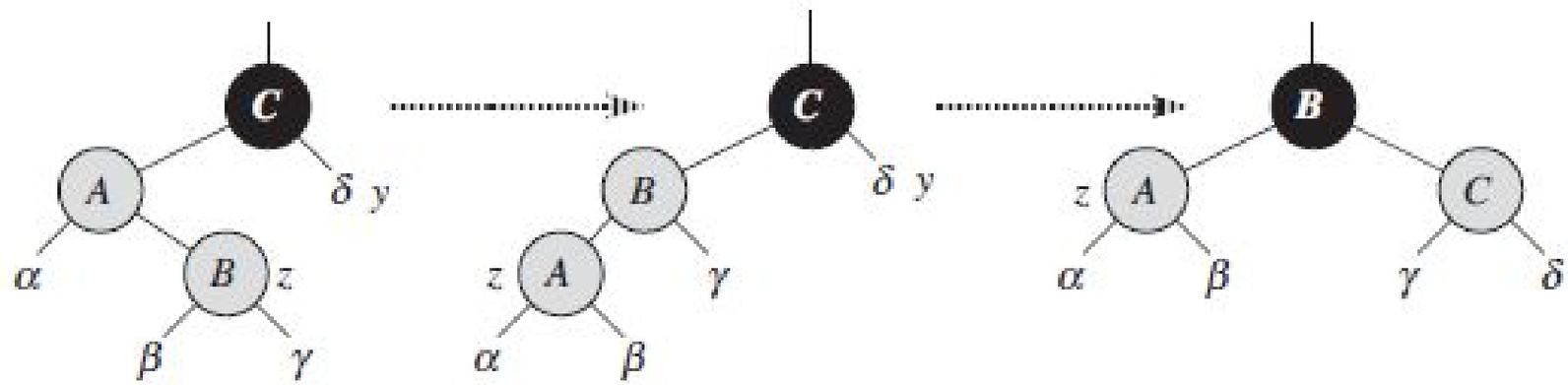
Inserção - Caso 1

```
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                  //Case 1
 9          else if z == z.p.right
10              z = z.p
11              LEFT-ROTATE(T, z)
12          z.p.color = BLACK
13          z.p.p.color = RED
14          RIGHT-ROTATE(T, z.p.p)
15      else (same as then clause
                with "right" and "left" exchanged)
16  T.root.color = BLACK
```



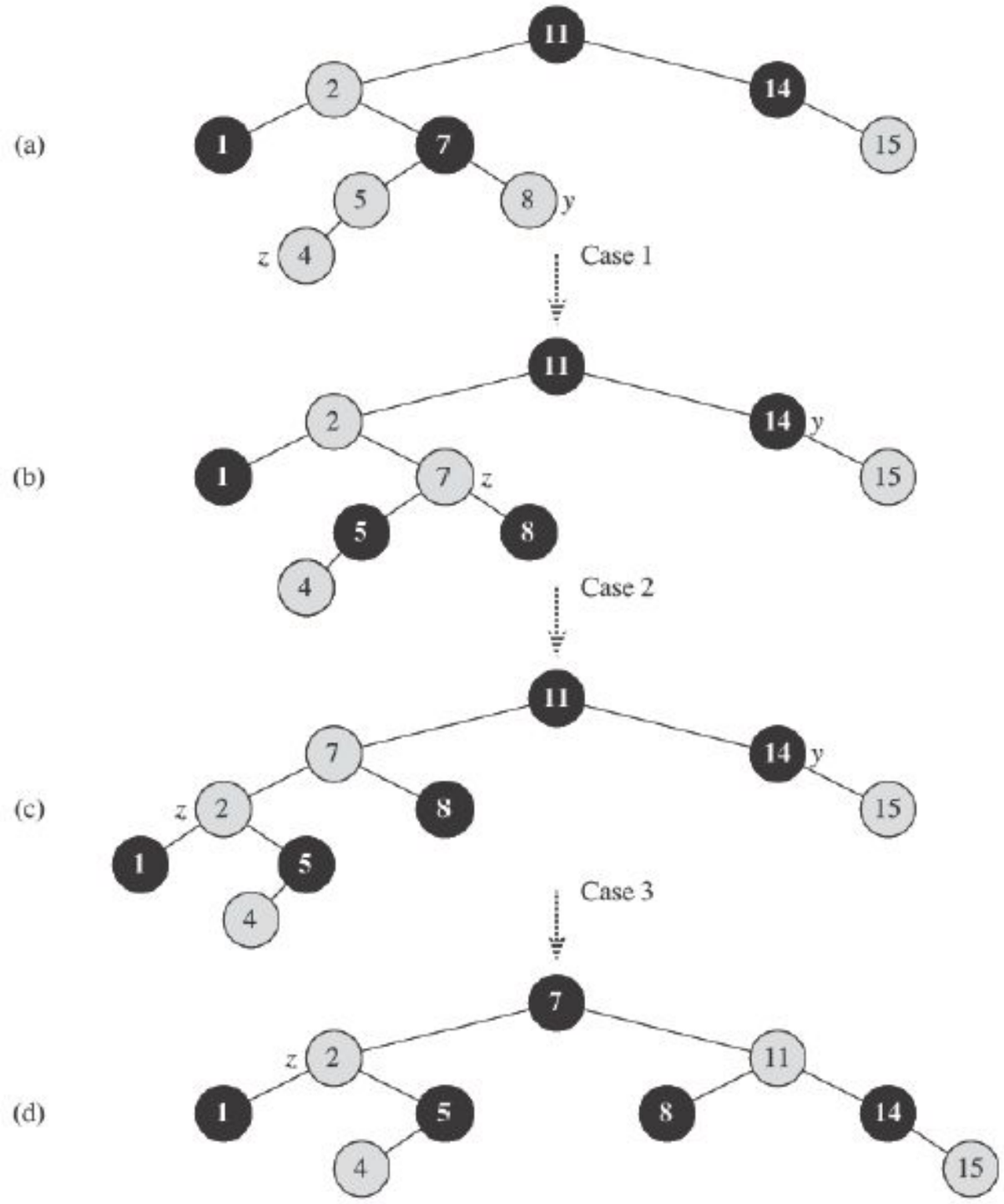
Inserção - Caso 2 e 3

```
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
 6              y.color = BLACK
 7              z.p.p.color = RED
 8              z = z.p.p
 9          else if z == z.p.right
10                  z = z.p                    //Case 2
11                  LEFT-ROTATE(T, z)          //Case 2
12              z.p.color = BLACK              //Case 3
13              z.p.p.color = RED              //Case 3
14              RIGHT-ROTATE(T, z.p.p)         //Case 3
15      else (same as then clause
                with "right" and "left" exchanged)
16  T.root.color = BLACK
```



Case 2 Case 3

Inserção - Casos 1 a 3



Remoção

- Caso 1: o irmão w de x é vermelho;
- Caso 2: o irmão w de x é preto e os filhos de w são pretos;
- Caso 3: o irmão w de x é preto, o filho à esquerda de w é vermelho e o filho à direita de w é preto;
- Caso 4: o irmão w de x é preto e o filho à direita de w é vermelho.

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

Remoção

- Troca os pais das subárvores *u* e *v*.
- O pai de *u* será o pai de *v* e vice-versa.

```
RB-TRANSPLANT(T, u, v)
1   if u.p == T.nil
2       T.root = v
3   elseif u == u.p.left
4       u.p.left = v
5   else u.p.right = v
6   v.p = u.p
```

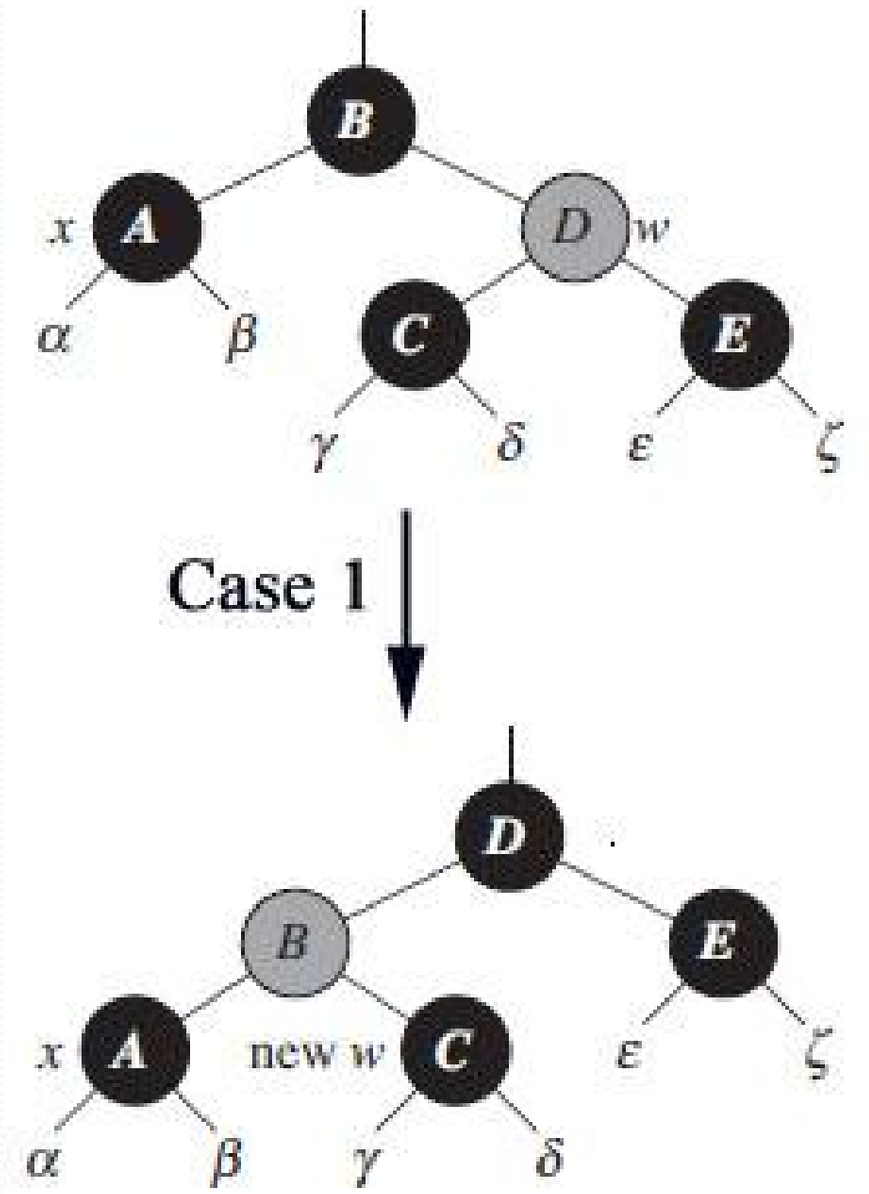
Remoção

- Como manter as propriedades da Red-Black?
- RB-Delete-Fixup(T,z)!

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 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 if w.right.color == BLACK
13                  w.left.color = BLACK
14                  w.color = RED
15                  RIGHT-ROTATE(T, w)
16                  w = x.p.right
17              w.color = x.p.color
18              x.p.color = BLACK
19              w.right.color = BLACK
20              LEFT-ROTATE(T, x.p)
21              x = T.root
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```

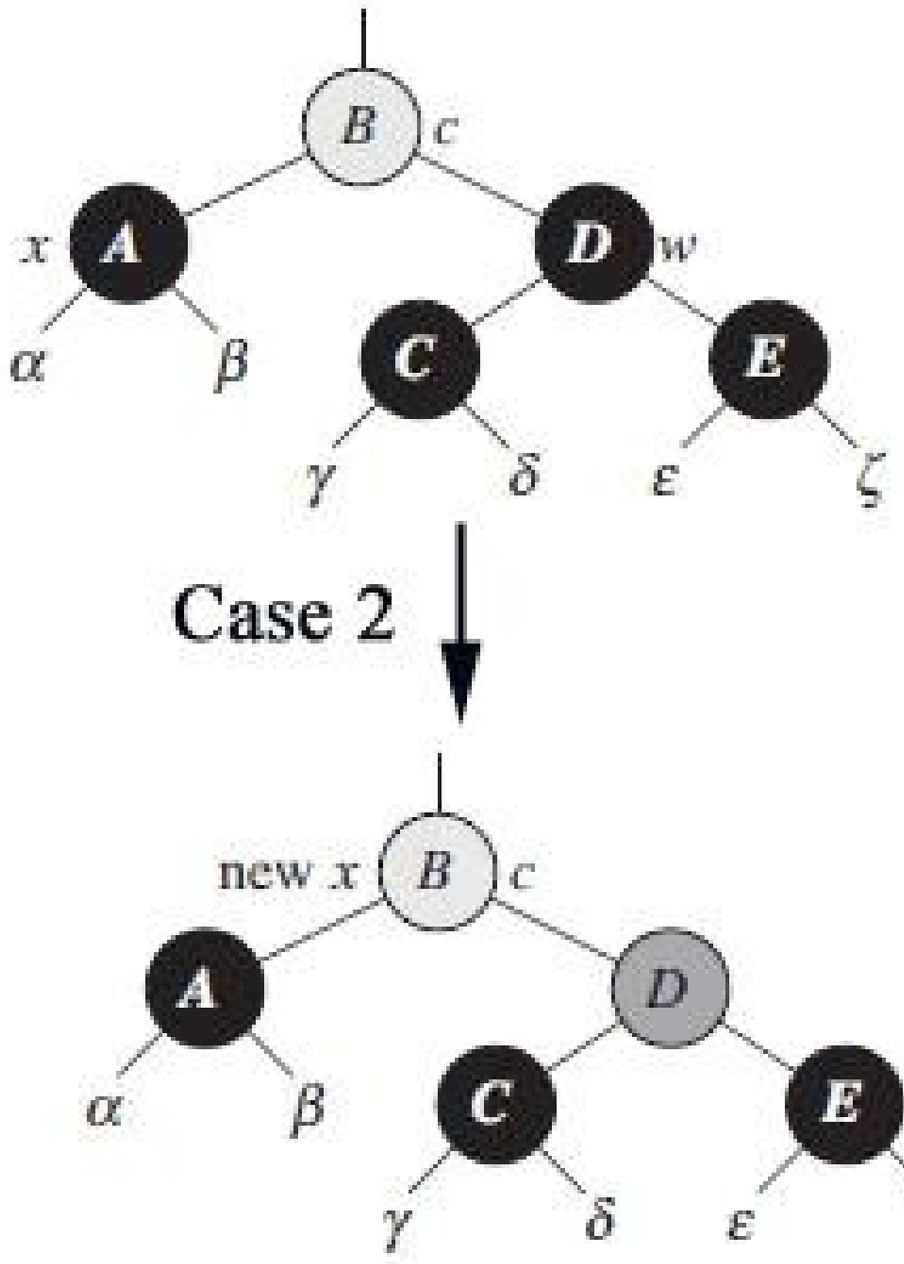
Remoção - Caso 1

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 4          if w.color == RED
 5              w.color = BLACK                          //Case 1
 6              x.p.color = RED                          //Case 1
 7              LEFT-ROTATE(T, x.p)                      //Case 1
 8              w = x.p.right                            //Case 1
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED
11              x = x.p
12          else if w.right.color == BLACK
13                  w.left.color = BLACK
14                  w.color = RED
15                  RIGHT-ROTATE(T, w)
16                  w = x.p.right
17              w.color = x.p.color
18              x.p.color = BLACK
19              w.right.color = BLACK
20              LEFT-ROTATE(T, x.p)
21              x = T.root
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```



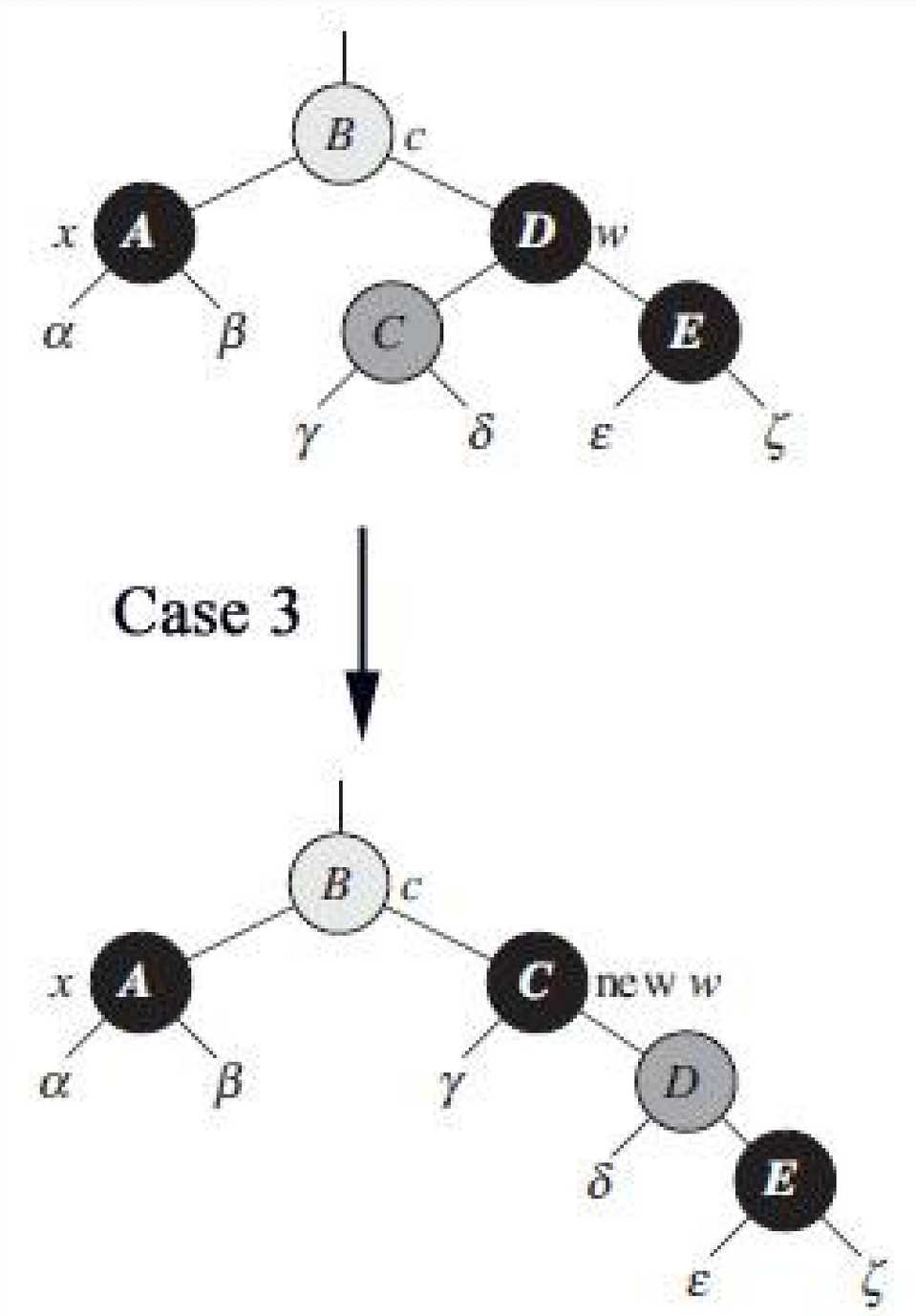
Remoção - Caso 2

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 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                                  //Case 2
11              x = x.p                                        //Case 2
12          else if w.right.color == BLACK
13                  w.left.color = BLACK
14                  w.color = RED
15                  RIGHT-ROTATE(T, w)
16                  w = x.p.right
17              w.color = x.p.color
18              x.p.color = BLACK
19              w.right.color = BLACK
20              LEFT-ROTATE(T, x.p)
21              x = T.root
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```



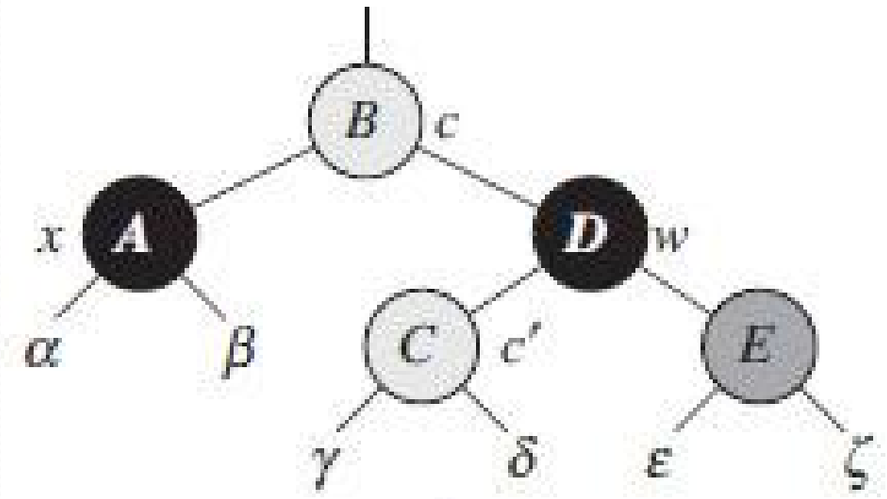
Remoção - Caso 3

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 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 if w.right.color == BLACK
13                  w.left.color = BLACK      //Case 3
14                  w.color = RED             //Case 3
15                  RIGHT-ROTATE(T, w)        //Case 3
16                  w = x.p.right             //Case 3
17              w.color = x.p.color
18              x.p.color = BLACK
19              w.right.color = BLACK
20              LEFT-ROTATE(T, x.p)
21              x = T.root
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```

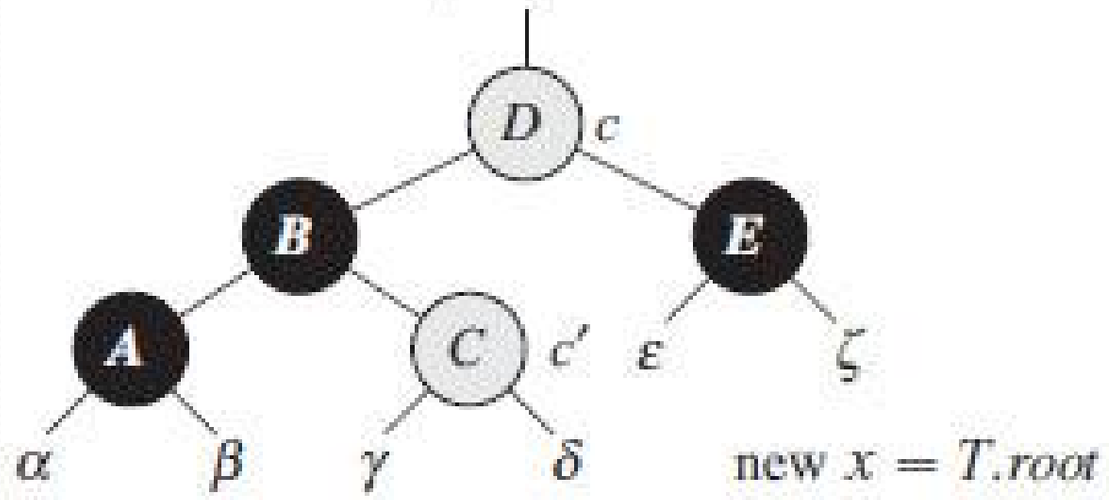


Remoção - Caso 4

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 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 if w.right.color == BLACK
13                  w.left.color = BLACK
14                  w.color = RED
15                  RIGHT-ROTATE(T, w)
16                  w = x.p.right
17              w.color = x.p.color
18              x.p.color = BLACK                 //Case 4
19              w.right.color = BLACK             //Case 4
20              LEFT-ROTATE(T, x.p)               //Case 4
21              x = T.root                        //Case 4
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```



Case 4



Exemplos

- Inserção
- Remoção
- Busca

- https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Árvore Red-Black (Rubro-Negra)

- Inserção: O(lgN)
- Remoção: O(lgN)
- Busca: O(lgN)