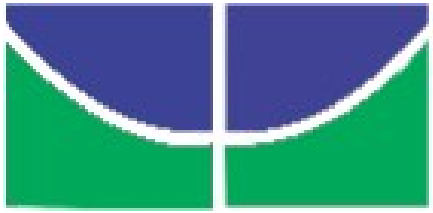


Árvores vermelho-preto

Prof. Dr Wesin Ribeiro

Neste capítulo

- Introdução
- Propriedades
- Rotação
- Inserção
- Eliminação
- revisão

Introdução

Uma árvore vermelho-preto é uma evolução da árvore de busca binária vista no capítulo anterior. Ela executa as mesmas operações de busca, inserção, eliminação, mínimo máximo, sucessor, ou predecessor, porém, por ser uma árvore balanceada, ela é mais rápida. A característica principal é que cada nó apresenta uma coloração e possui a complexidade O(log n) no pior caso.

Mais atributos que ABB

Uma ***árvore vermelho-preto*** é uma árvore de busca binária com um bit extra de armazenamento por nó: sua ***cor***.

- Restringe as cores
- Aproximadamente balanceada
- Valores nulos para nós inexistentes
- Contém os atributos cor, chave, direita, esquerda e pai

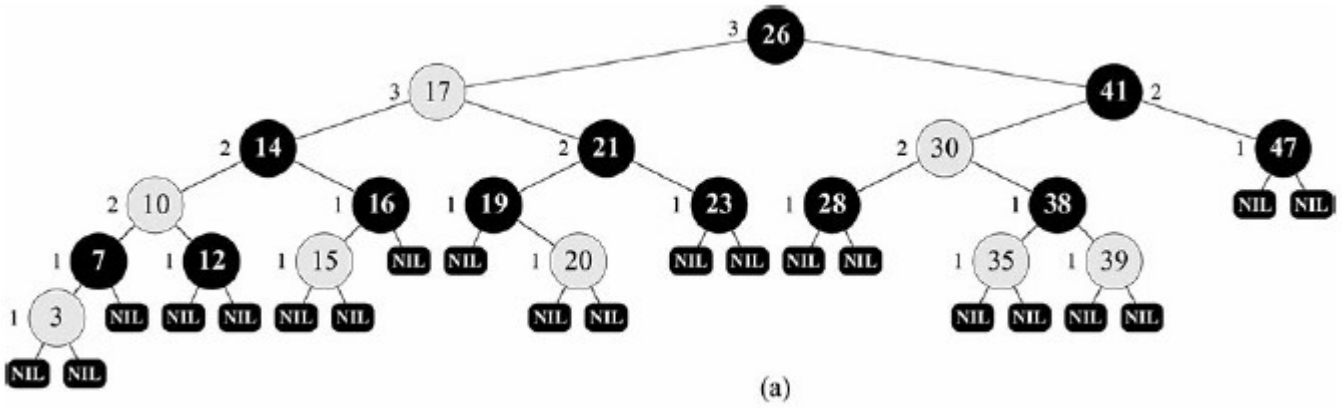
Propriedades

Uma árvore vermelho-preto é uma árvore de busca binária que satisfaz as seguintes ***propriedades vermelho-preto***:

1. Todo nó é vermelho ou preto
2. A raiz sempre será preta
3. Toda folha (Nó nulo) é preta
4. Um nó vermelho só pode ter filhos preto
5. Para cada nó, todos os caminhos do nó até as folhas descendentes contém o mesmo número de nós pretos.

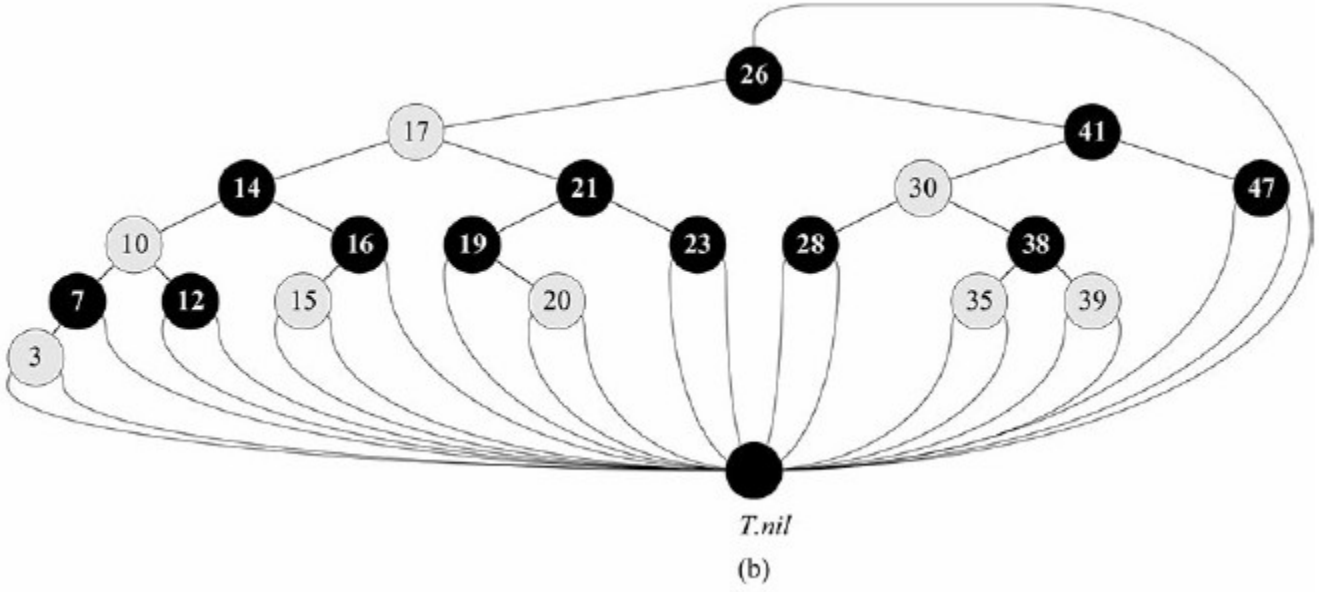
Altura preta

O número de nós pretos em qualquer caminho simples de um nó *x*, sem incluir esse nó, até uma folha, é chamado de ***altura preta***.



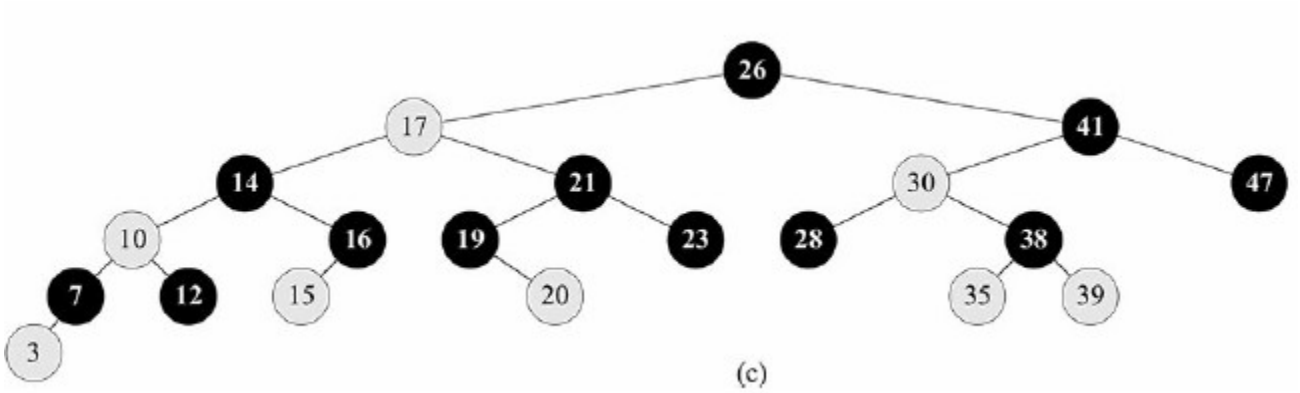
O que fazer com as folhas nulas?

Todos os ponteiros nulos de uma árvore vermelho-preto podem ser substituídos por uma sentinela a fim de economizar espaço.



Estrutura equivalente

Em geral, limitamos nosso interesse aos nós internos de uma árvore vermelho-preto, já que eles contêm os valores de chaves.



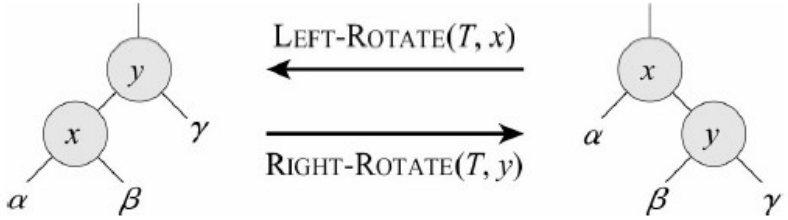
Exercício

Desenhe, no estilo da Figura (a) , a árvore de busca binária completa de altura 3 nas chaves {1, 2, ...,15}. Adicione as folhas NIL e cores aos nós de três maneiras diferentes, de tal modo que as alturas pretas das árvores vermelho-preto resultantes sejam 2, 3 e 4.

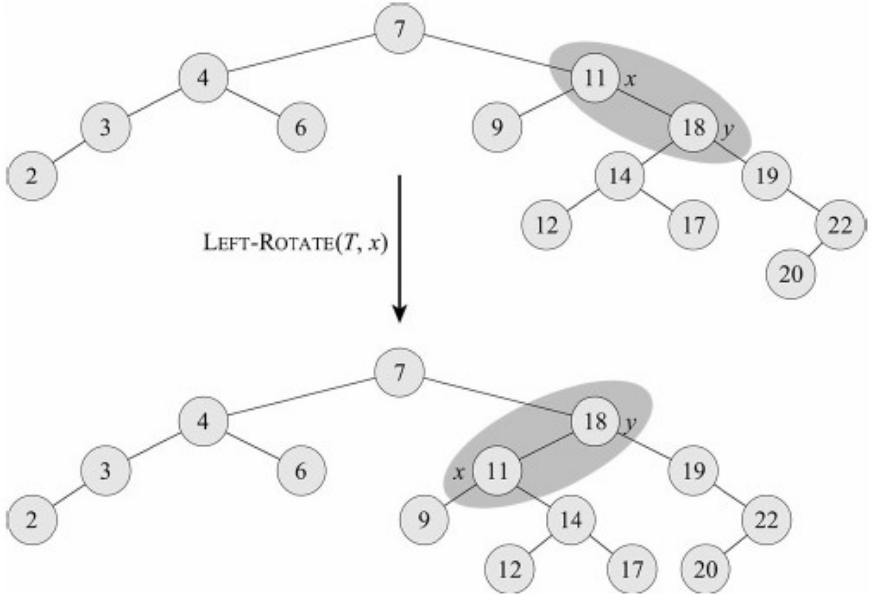
Desenhe a árvore vermelho-preto que resulta após a chamada a TREE -INSERT na árvore da Figura (a) com chave 36. Se o nó inserido for vermelho, a árvore resultante é uma árvore vermelho-preto? E se ele for preto?

Rotações em árvores

Ao executar uma inserção ou eliminação em uma árvore vermelho-preto, as propriedades da árvore podem ser violadas pela operação.



Tempo é O(1)



Pseudo código da rotação à esquerda

```
LEFT-ROTATE(T; x)
1  y = x.direita            // define y
2  x.direita = y.esquerda   // transforma a subárvore à esquerda de y na subárvore à direita de x
3  if y.esquerda ≠ T.nil
4        y.esquerda.p = x
5  y.p = x.p                // liga o pai de x a y
6  if x.p == T.nil
7        T.raiz = y
8  elseif x == x.p.esquerda
9        x.p. esquerda = y
10 else x.p.direita = y
11 y.esquerda = x           // coloca x à esquerda de y
12 x.p = y
```

Inserção

Para inserir o nó *z* na árvore *T,* usamos o método de inserção como se ela fosse uma árvore de busca binária comum e depois colorimos *z* de vermelho.

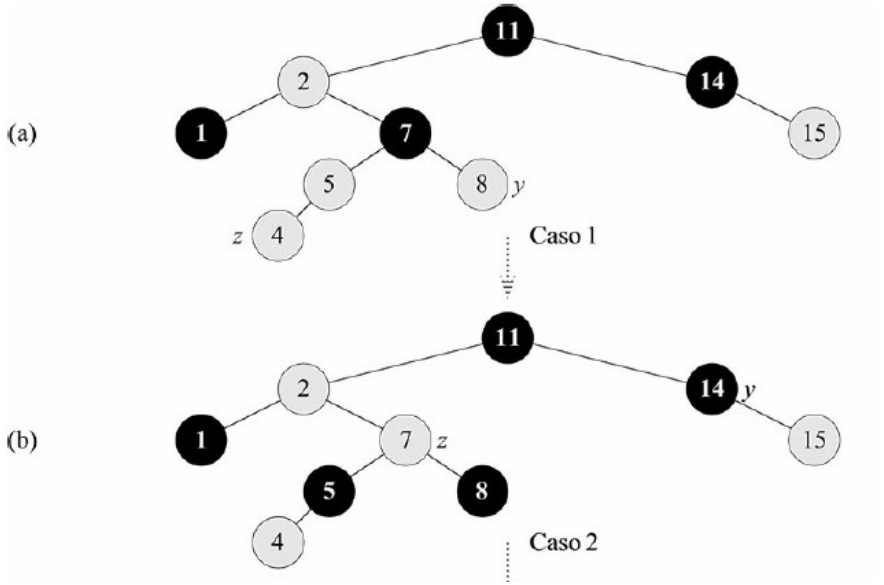
- Usa uma sentinela para objetos nulos
- O filho da esquerda e direita de z aponta para o sentinela nulo
- O nó z é colorido de vermelho
- Restaura as propriedades da árvore vermelho-preto

Casos que podem ocorrer

Após a inserção de um nó vermelho, certamente as propriedades 2 e 4 serão violadas e devem ser corrigidas.

Caso 1: O tio y de z é vermelho

Caso 2: O tio y de z é preto e z é um filho a direita

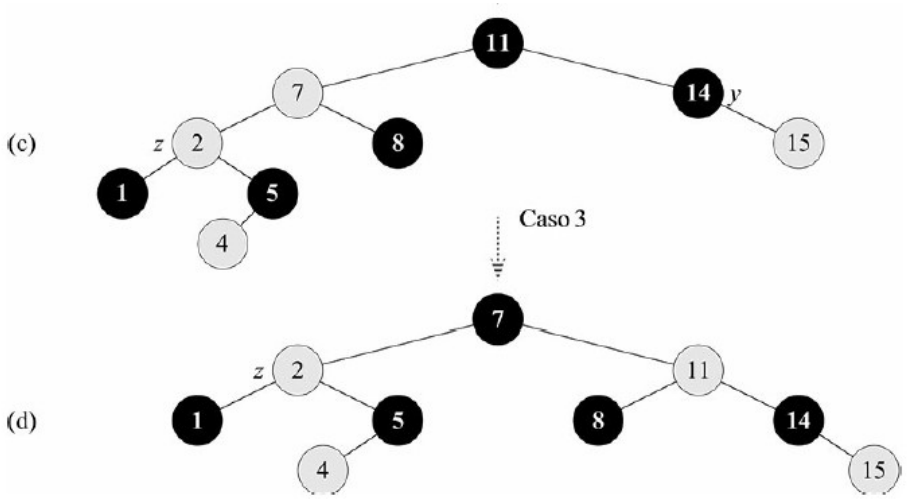


Casos que podem ocorrer

Após a inserção de um nó vermelho, certamente as propriedades 2 e 4 serão violadas e devem ser corrigidas.

Caso 3: O tio y de z é preto e z é um fiilho a esquerda

A figura (d) é um caso válido



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

Pseudo código da inserção

Pseudo código da inserção (correção)

```
RB-INSERT-FIXUP(T, z)
 1    while z.p.cor == VERMELHO
 2          if z.p == z.p.p.esquerda
 3                y = z.p.p.direita
 4                if y.cor = VERMELHO
 5                      z.p.cor = PRETO                          // caso 1
 6                      y.cor = PRETO                            // caso 1
 7                      z.p.p.cor = VERMELHO                     // caso 1
 8                      z = z.p.p                                // caso 1
 9                else if z = z.p.direita
10                            z = z.p                            // caso 2
11                            LEFT-ROTATE(T, z)                  // caso 2
12                      z.p.cor = PRETO                          // caso 3
13                      z.p.p.cor = VERMELHO                     // caso 3
14                      RIGHT-ROTATE(T, z.p.p)                   // caso 3
15          else (igual à cláusula then
                      com "direita" e "esquerda" trocadas)
16    T.raiz.cor = PRETO
```

Eliminação

Eliminar um nó de uma árvore vermelho-preto é um pouco mais complicado que inserir um nó.

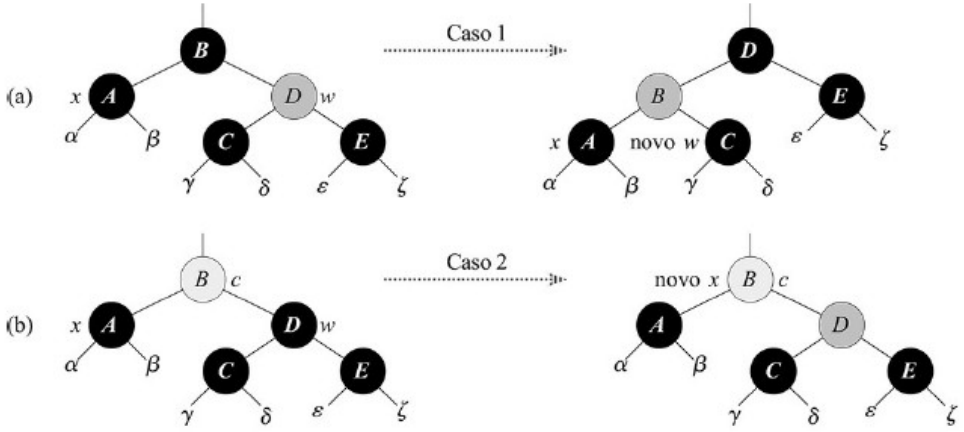
- Quando o nó z tem menos que dois filhos
- Quando o nó z tem dois filhos
- Guardamos a cor de y antes de ser eliminado ou passar para dentro
- Rastreamos o nó x que assume a posição original de y

Casos que podem ocorrer

Após a eliminação de um nó z, certamente as propriedades de árvores vermelho-preto serão violadas e devem ser corrigidas.

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

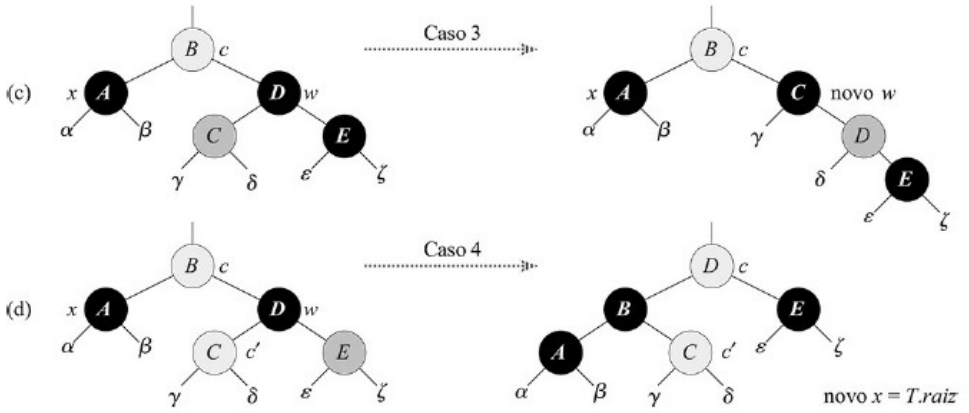


Casos que podem ocorrer

Após a eliminação de um nó z, certamente as propriedades de árvores vermelho-preto serão violadas e devem ser corrigidas.

Caso 3: O irmão w de x é preto, o filho a esquerda de w é vermelho e o filho a direita é preto

Caso 4: O irmão w de x é preto, e o filho a direita de w é vermelho.



Pseudo código da eliminação

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

```
RB-TRANSPLANT(T, u, v)
1   if u.p == T.nil
2           T.raiz = v
3   elseif u == u.p.esquerda
4           u.p.esquerda = v
5   else u.p.direita = v
6   v.p = u.p
```

Pseudo código da eliminação (correção)

```
RB-DELETE-FIXUP(T; x)
1    while x ≠ T.raiz and x.cor == PRETO
2        if x == x.p.esquerda
3        w = x.p.direita
4        if w.cor == VERMELHO
5            w.cor = PRETO                                  // caso 1
6            x.p.cor = VERMELHO                             // caso 1
7            LEFT-ROTATE(T, x.p)                            // caso 1
8            w = x.p.direita                                // caso 1
9        if w.esquerda.cor == PRETO and w.direita.cor == PRETO
10           w.cor = VERMELHO                               // caso 2
11           x = x.p                                        // caso 2
12       else if w.direita.cor == PRETO
13           w.esquerda.cor = PRETO                         // caso 3
14           w.cor = VERMELHO                               // caso 3
15           RIGHT-ROTATE(T,w)                              // caso 3
16           w = x.p.direita                                // caso 3
```

Revisão

Atenção, chegou a hora da revisão.

- Caso 1: O irmão w de x é vermelho