Árvores

Árvores *Red-Black* – Parte I: Definição e Inserção

Prof. Edson Alves - UnB/FGA

2018

Sumário

Definição

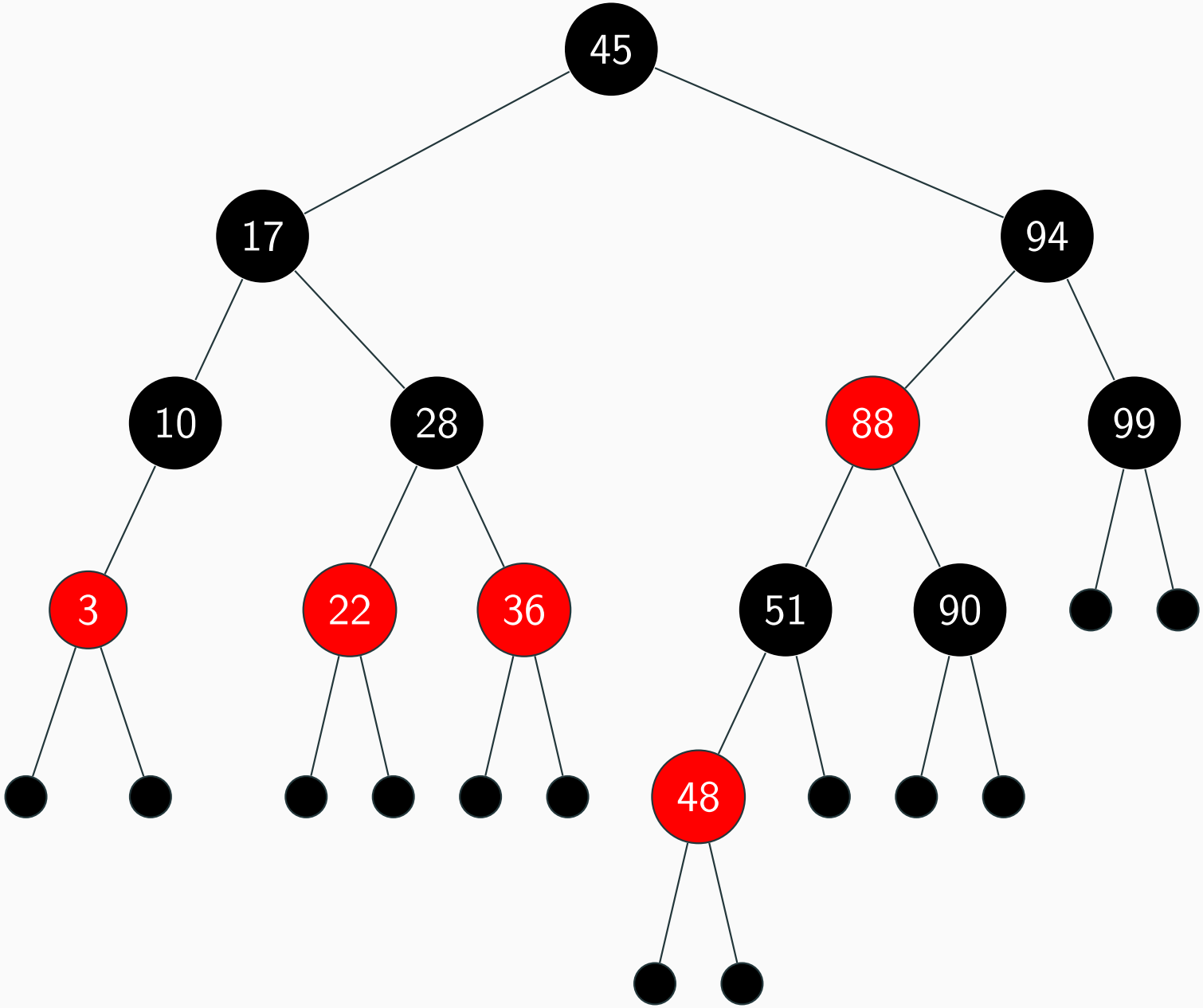
Árvores Red-Black

- Uma árvore *red-black* é uma árvore binária de busca auto-balanceável
- Foi proposta em 1978 por Leonidas J. Guibas e Robert Sedgewick
- A cada um de seus nós é atribuída uma cor: vermelha ou preta
- São estabelecidas 5 propriedades que relacionam os nós e suas cores
- Estas propriedades garantem que a altura h da árvore seja proporcional a $\log N$, onde N é o tamanho da árvore
- As rotinas de inserção e remoção devem preservar as propriedades das árvores *red-black* e, consequentemente, o balanceamento de sua altura

Propriedades de uma árvore red-black

1. Cada nó ou é vermelho ou é preto
2. A raiz é tem a cor preta
3. Todas as folhas são nulas e tem a cor preta
4. Se um nó é vermelho, então todos os seus filhos são pretos
5. Dado um nó n, todos os caminhos de n até um de seus descendentes nulos tem o mesmo número de nós pretos

Exemplo de árvore red-black



Observações sobre árvores red-black

- Cormen et. al propõem e demonstram o seguinte lema: "*A red-black tree with N internal nodes has height at most* $2\log(N+1)$"
- Este lema garante complexidade $O(\log N)$ para as operações de busca, inserção e remoção
- Como uma árvore *red-black* é uma árvore binária de busca, o algoritmo de busca é idêntico ao utilizado em árvores binárias de busca
- As inserções e remoções devem tratar as possíveis violações às propriedades das árvores *red-black*, de modo que as árvores resultantes sejam efetivamente árvores *red-black*
- Para implementar tais operações, é útil manter um ponteiro para o pai de cada nó
- É útil também implementar funções auxiliares que permitam acessar os ponteiros do avó, do tio e do irmão de um dado nó

Definição de uma árvore red-black em C++

```cpp
#include <bits/stdc++.h>

template<typename T>
class RBTree {
private:
    struct Node {
        T info;
        enum { RED, BLACK } color;
        Node *left, *right, *parent;
    };

    Node *root;

public:
    RBTree() : root(nullptr) {}

```

Funções auxiliares

```
17 private:
18     Node * parent(Node *node)
19     {
20         return node ? node->parent : nullptr;
21     }
22
23     Node * grandparent(Node* node)
24     {
25         return parent(parent(node));
26     }
27
28     Node* sibling(Node* node)
29     {
30         auto p = parent(node);
31         return p ? (node == p->left ? p->right : p->left) : nullptr;
32     }
33
34     Node * uncle(Node* node)
35     {
36         return sibling(parent(node));
37     }
```

Funções auxiliares

```
38
39      void rotate_left(Node *G, Node *P, Node *C)
40      {
41          if (G != nullptr)
42              G->left == P ? G->left = C : G->right = C;
43
44          P->right = C->left;
45          C->left = P;
46
47          C->parent = G;
48          P->parent = C;
49
50          if (C->left)
51              C->left->parent = P;
52      }
```

Funções auxiliares

```
53
54      void rotate_right(Node *G, Node *P, Node *C)
55      {
56          if (G != nullptr)
57              G->left == P ? G->left = C : G->right = C;
58
59          P->left = C->right;
60          C->right = P;
61
62          C->parent = G;
63          P->parent = C;
64
65          if (C->right)
66              C->right->parent = P;
67      }
68
```

Inserção

Inserção em árvores red-black

- A inserção em uma árvore *red-black* consiste em duas etapas
- A primeira é a inserção de um nó vermelho, nos mesmos moldes da inserção em uma árvore binária de busca
- A segunda etapa consiste em corrigir possíveis violações às propriedades de uma árvore *red-black*
- Observe que as propriedade 1 e 3 sempre serão verdadeiras
- As demais propriedades podem ser violadas, a depender do local onde a inserção foi realizada
- São quatro cenários possíveis

Rotinas básicas de inserção

```
69 public:
70     void insert(const T& info) {
71         auto node = insert(&root, nullptr, info);
72         restore_properties(node);
73     }
74
75 private:
76     Node * insert(Node **node, Node *parent, const T& info)
77     {
78         if (*node == nullptr) {
79             *node = new Node { info, Node::RED, nullptr, nullptr, parent };
80             return *node;
81         }
82
83         if ((*node)->info == info)
84             return *node;
85         else if (info < (*node)->info)
86             return insert(&(*node)->left, *node, info);
87         else
88             return insert(&(*node)->right, *node, info);
89     }
```

Cenário A: inserção em árvore vazia

- Neste caso, a inserção é trivial, mas a propriedade 2 é violada
- A correção consiste em pintar a raiz com a cor preta
- Este processo adicionará uma unidade ao tamanho de todos os caminhos que partem da raiz às folhas
- Assim, a propriedade 5 ficará preservada
- Segue abaixo uma visualização da inserção da informação 40 em uma árvore vazia:

Restauração das propriedades no cenário A

```
90
91      void restore_properties(Node *node)
92      {
93          if (parent(node) == nullptr)        // Cenário A: node é a raiz
94              node->color = Node::BLACK;
```

Cenário B: o pai do nó inserido é preto

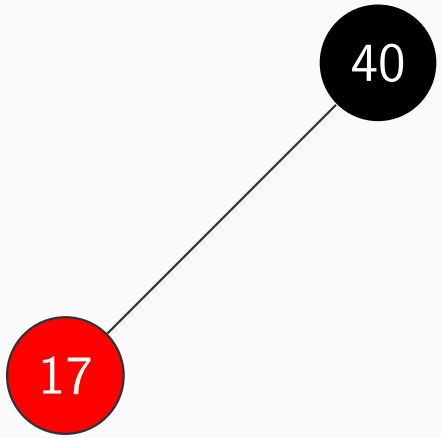
- Como o pai do nó inserido n é preto, não há violação da propriedade 4
- Além disso, como n é vermelho, o caminho da raiz até um de seus filhos mantém o mesmo número de nós pretos que haviam até a posição onde n foi inserido
- Deste modo, não há violação da propriedade 5
- Como n tem um pai preto, ele não é a raiz (pois a raiz não tem pai)
- Assim, não há violação da propriedade 2
- De fato, neste cenário não há necessidade de nenhuma correção após a inserção

Exemplo de inserções no cenário B

Informação a ser inserida: 17

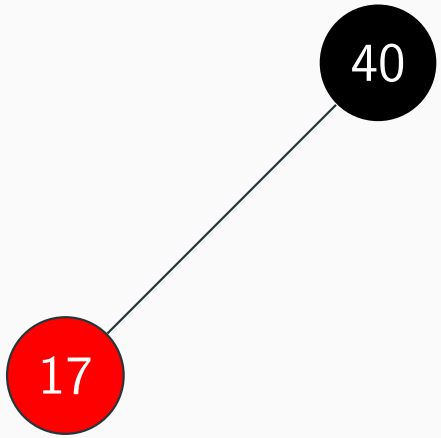
Exemplo de inserções no cenário B

Informação a ser inserida: 17



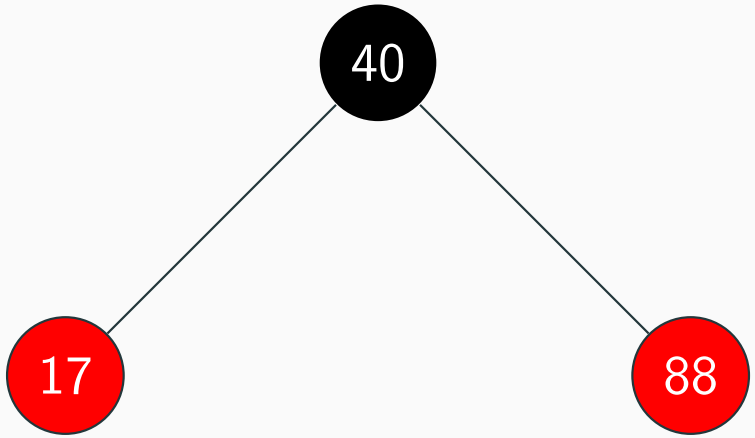
Exemplo de inserções no cenário B

Informação a ser inserida: 88



Exemplo de inserções no cenário B

Informação a ser inserida: 88

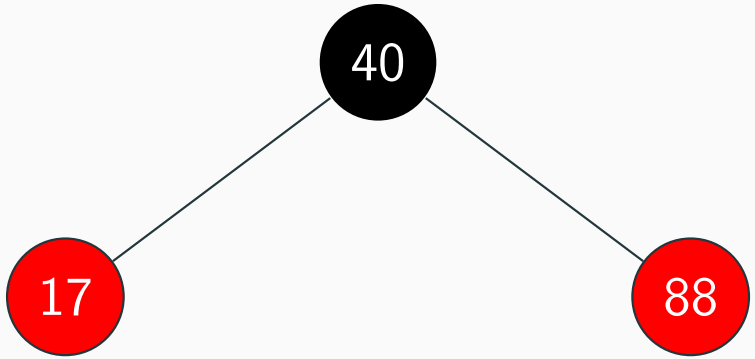


Cenário C: o pai e o tio do nó inserido são vermelhos

- Neste cenário, o pai e o filho do nó inserido n são ambos vermelhos
- Como n também é vermelho, há uma violação da propriedade 4
- Se o avô se tornar vermelho, e o pai e o tio se tornarem pretos, a violação da propriedade 4 é corrigida
- Esta mudança também não viola a propriedade 5, pois o número de nós pretos nos caminhos muda
- Contudo, se o avô for a raiz, a propriedade 2 passa a ser violada
- Caso contrário, pode existir uma violação da propriedade 4, se o bisavô for vermelho
- Para evitar tais violações, é preciso reiniciar a rotina de correção no avô

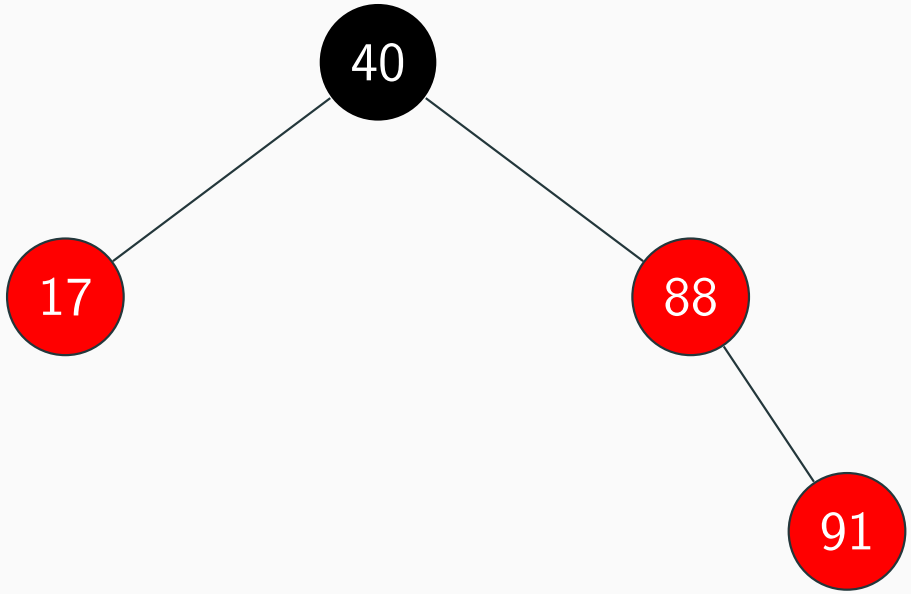
Exemplo de inserções no cenário C

Informação a ser inserida: 91



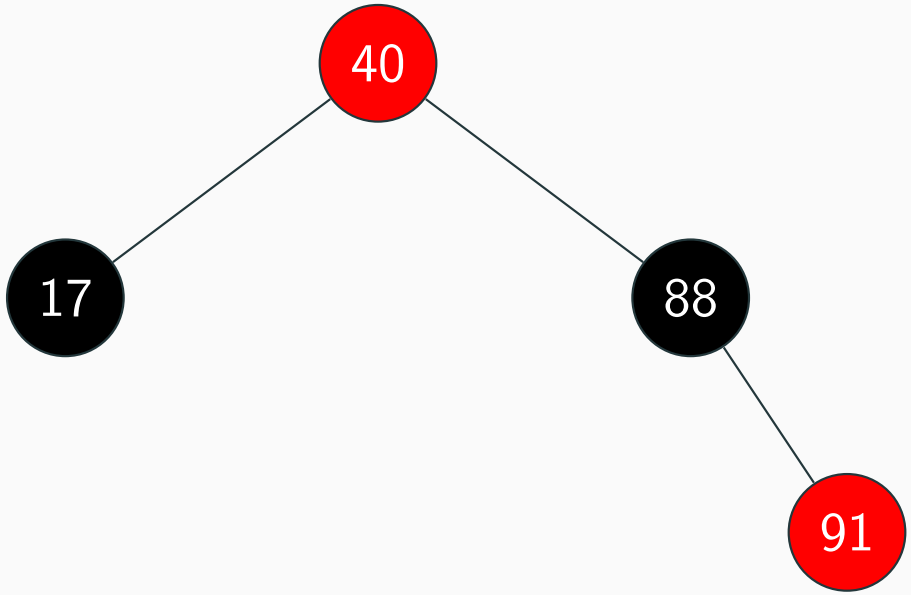
Exemplo de inserções no cenário C

Informação a ser inserida: 91



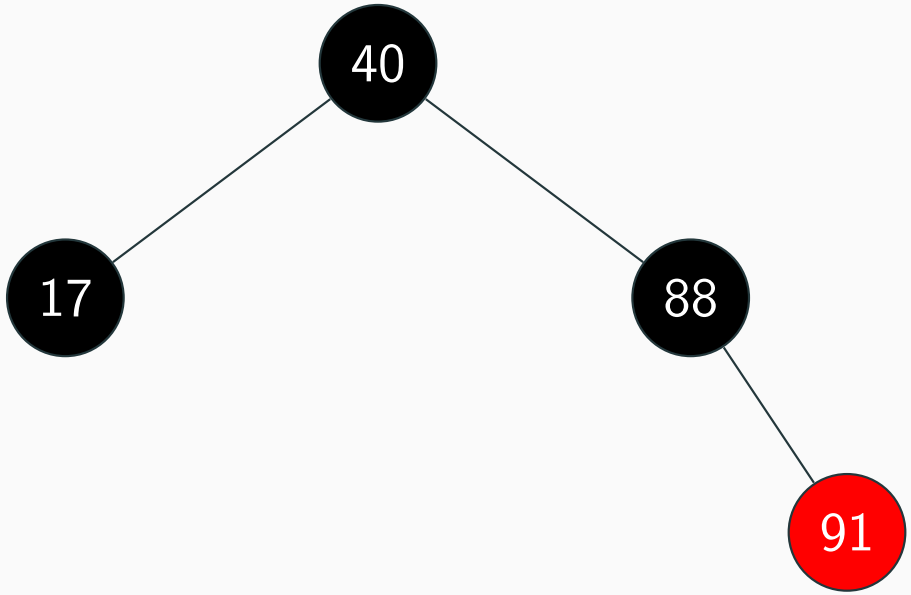
Exemplo de inserções no cenário C

Informação a ser inserida: 91



Exemplo de inserções no cenário C

Informação a ser inserida: 91



Restauração das propriedades nos cenários B e C

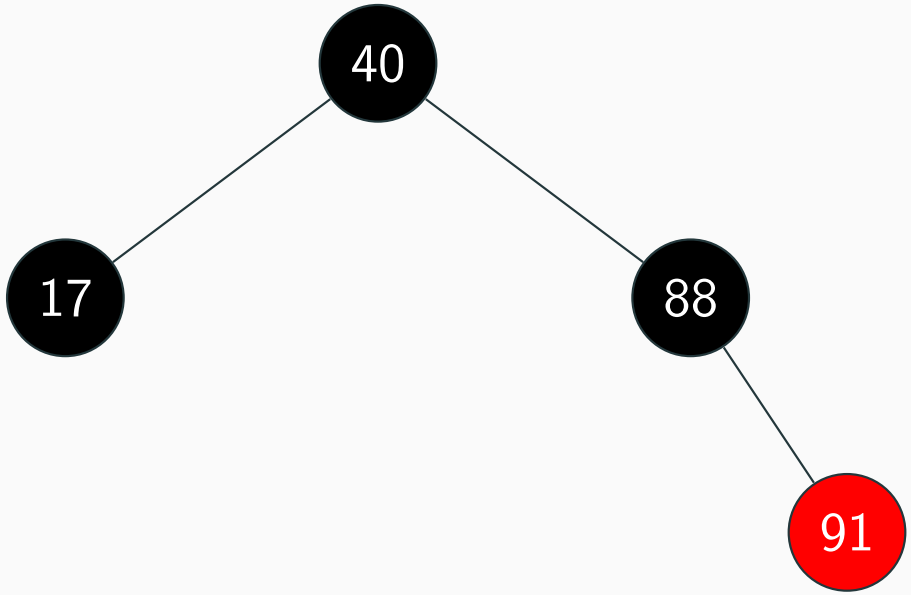
```
95          else if (parent(node)->color == Node::BLACK)    // Cenário B
96              return;
97          else if (uncle(node) and uncle(node)->color == Node::RED)
98          {
99              // Cenário C: pai e tio vermelhos
100             parent(node)->color = Node::BLACK;
101             uncle(node)->color = Node::BLACK;
102             grandparent(node)->color = Node::RED;
103
104             // Como o pai é vermelho, o avô não é nulo
105             restore_properties(grandparent(node));
```

Cenário D: pai vermelho, tio preto

- Neste último cenário, é necessário utilizar rotações de modo a reposicionar o pai na posição do avô
- A direção da rotação é definida de acordo com a posição do pai em relação ao avô: se for o filho à esquerda, a rotação é para a direita, e vice-versa
- Também é necessária a troca de cores entre o pai e o avô após a rotação
- Esta troca restaura as violação à propriedade 4 e não viola a propriedade 5
- Há, porém, um caso especial: se o filho tiver em direção oposta a que o pai ocupa em relação ao avô
- Neste caso, é necessária uma rotação para colocar o filho na posição do pai
- Deste modo, ambos passaram a ter a mesma direção em relação ao avô

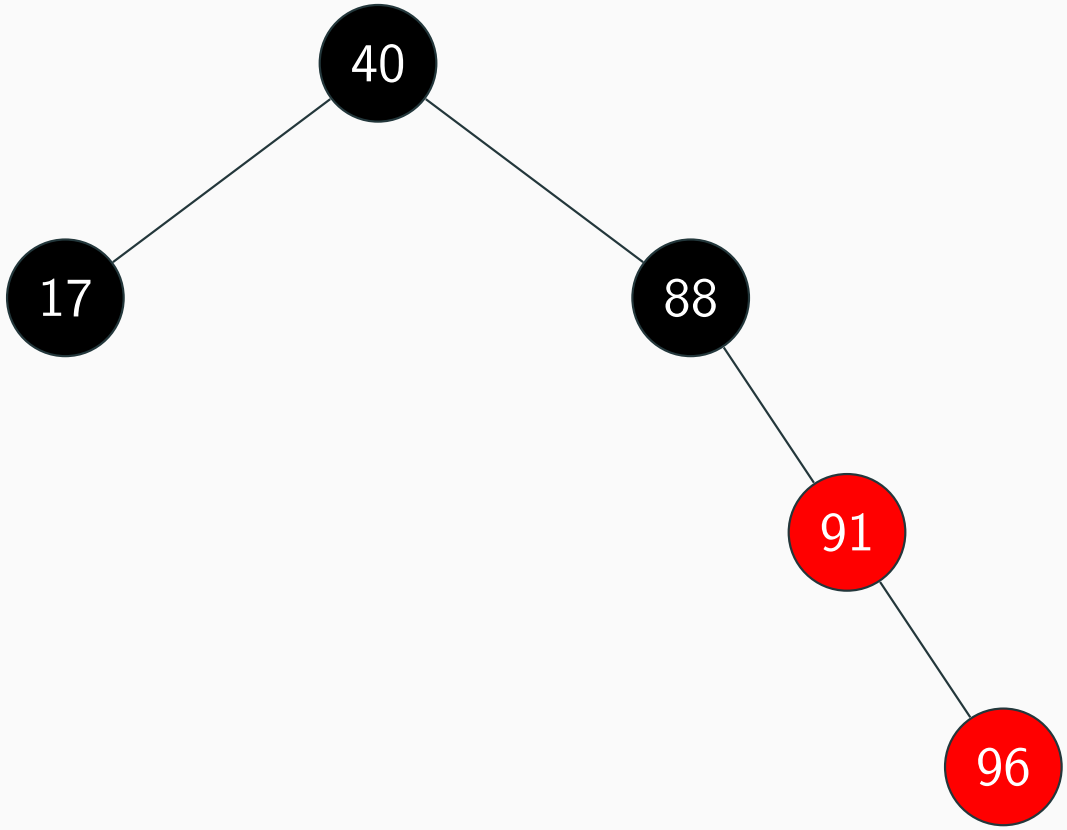
Exemplo de inserções no cenário D

Informação a ser inserida: 96



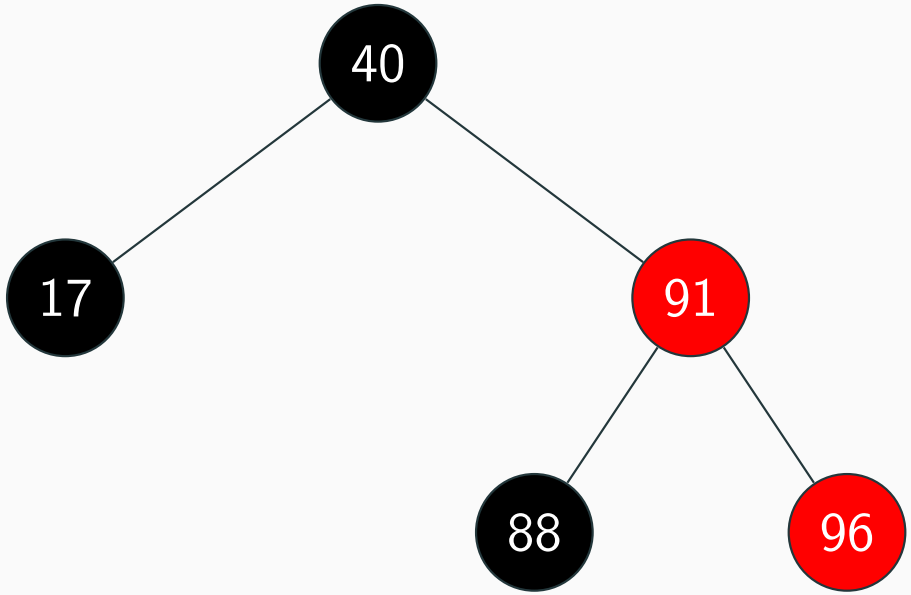
Exemplo de inserções no cenário D

Informação a ser inserida: 96



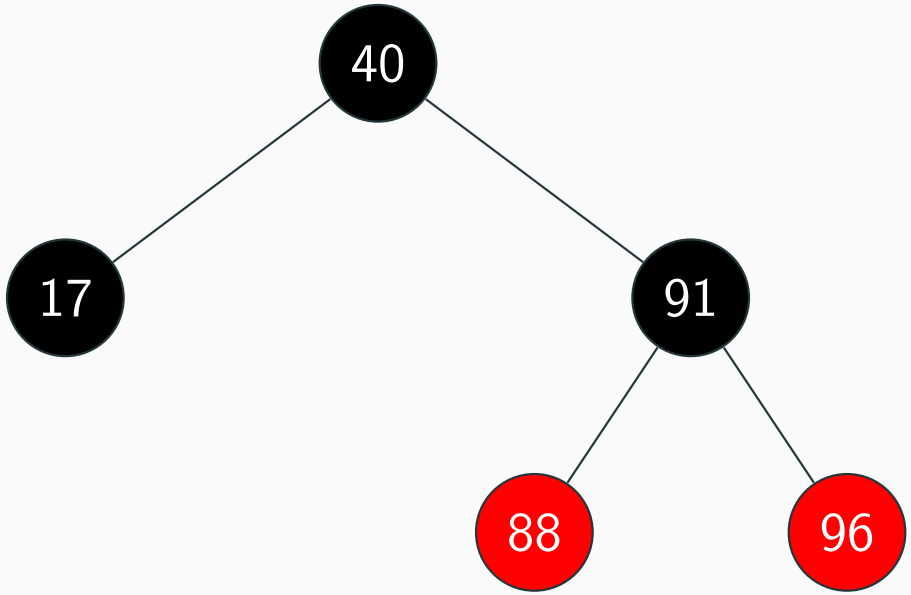
Exemplo de inserções no cenário D

Informação a ser inserida: 96



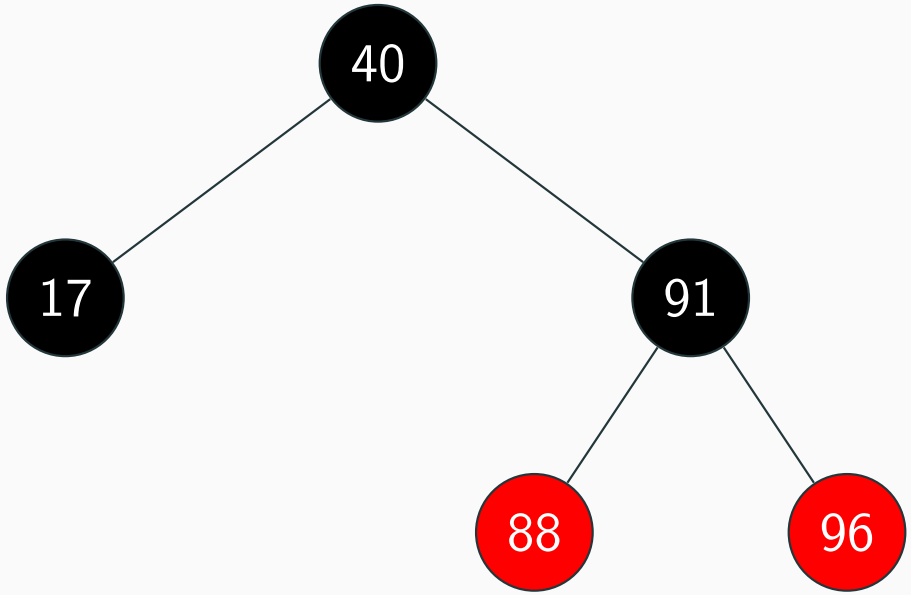
Exemplo de inserções no cenário D

Informação a ser inserida: 96



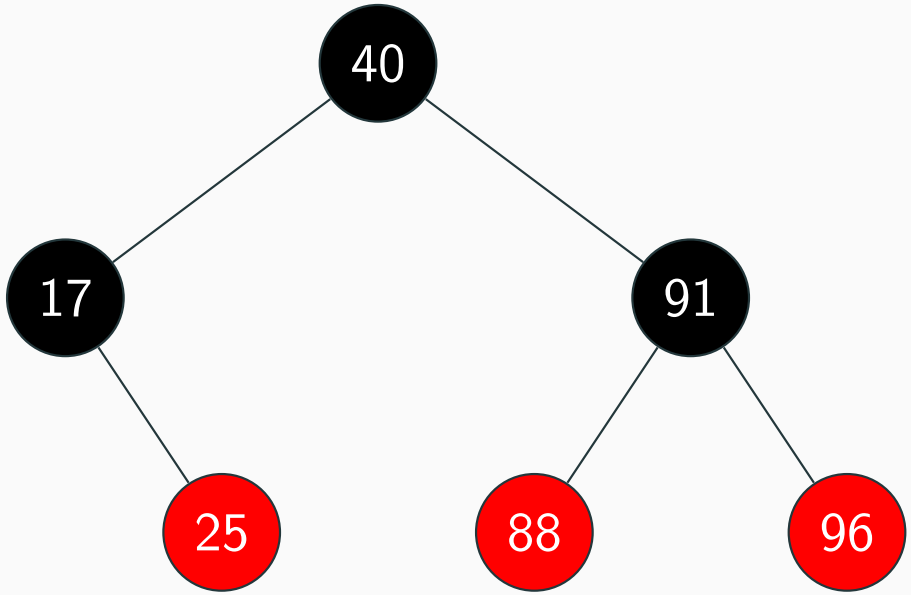
Exemplo de inserções no cenário D

Informação a ser inserida: 25



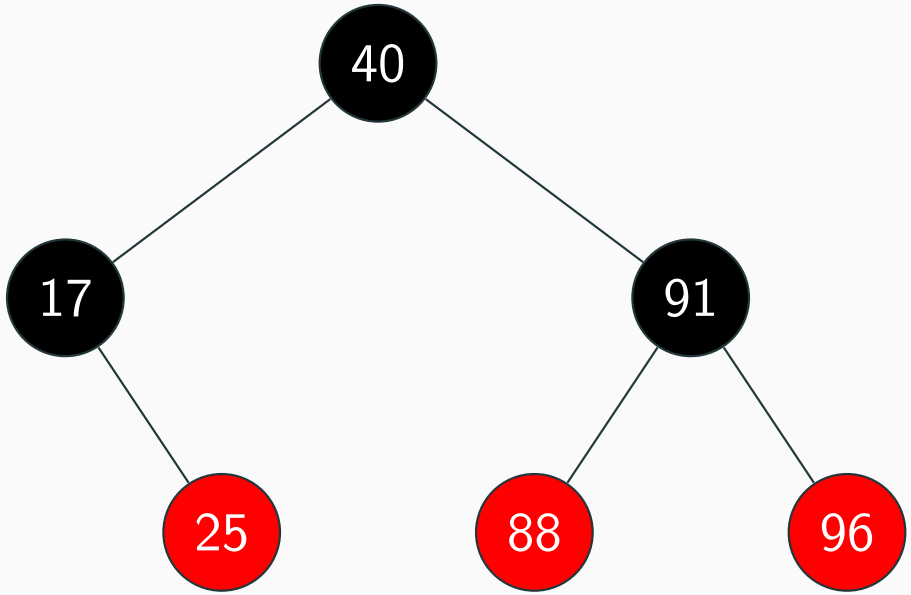
Exemplo de inserções no cenário D

Informação a ser inserida: 25



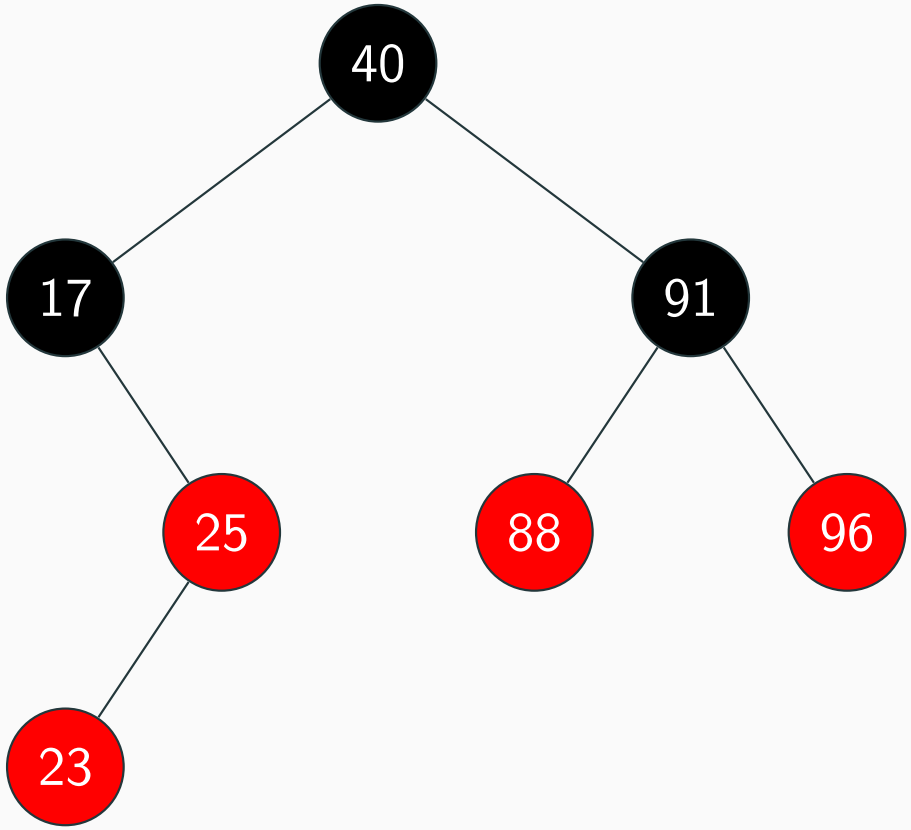
Exemplo de inserções no cenário D

Informação a ser inserida: 23



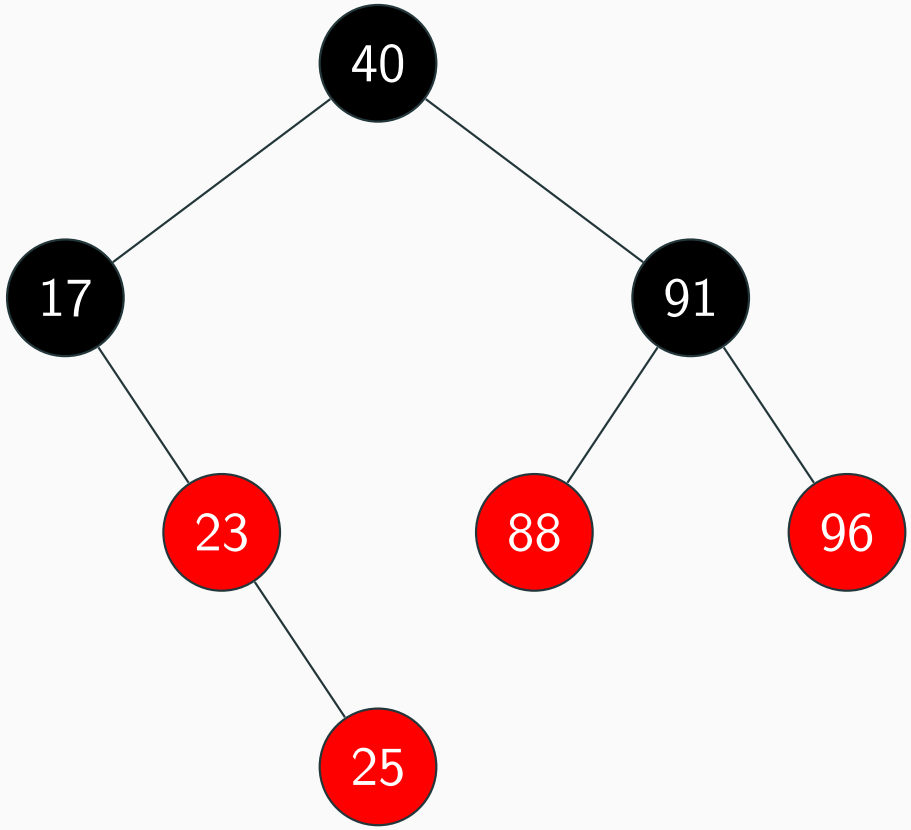
Exemplo de inserções no cenário D

Informação a ser inserida: 23



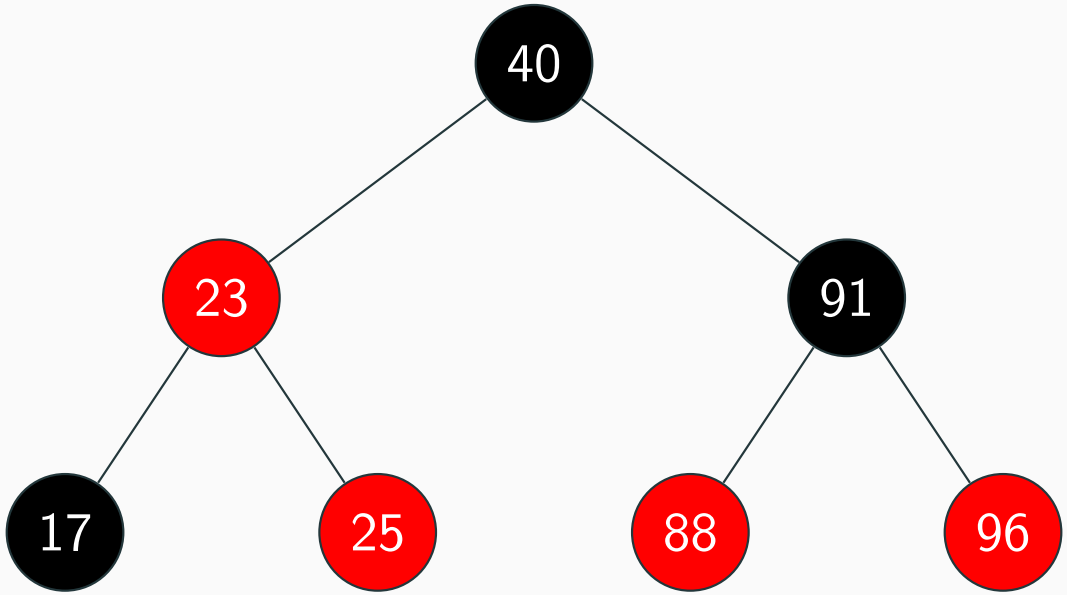
Exemplo de inserções no cenário D

Informação a ser inserida: 23



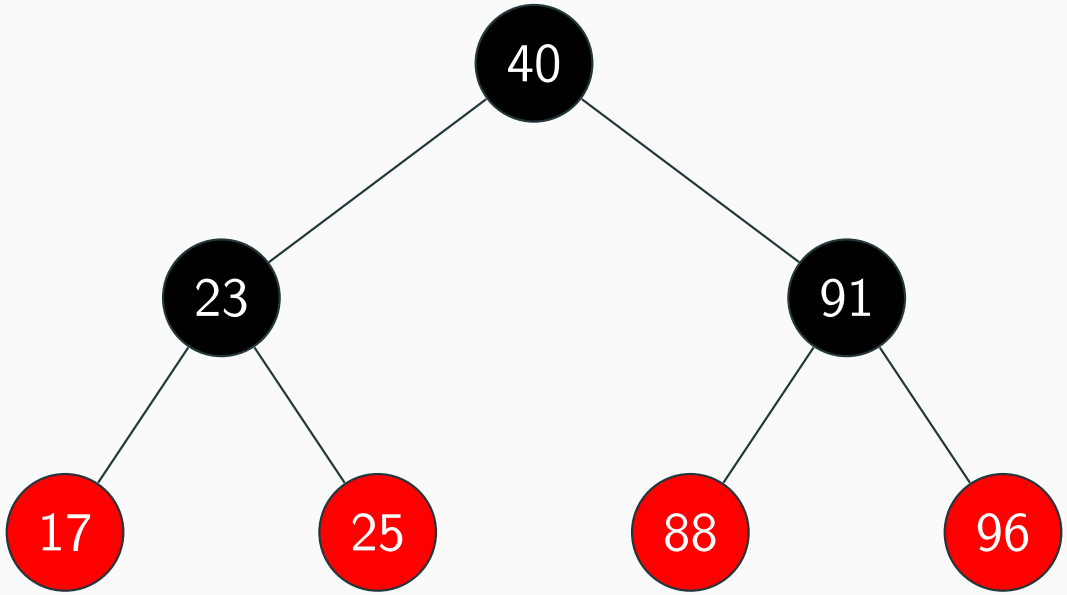
Exemplo de inserções no cenário D

Informação a ser inserida: 23



Exemplo de inserções no cenário D

Informação a ser inserida: 23



Restauração das propriedades no cenário D

```
106         } else
107         {
108             // Cenário D: pai vermelho, tio preto
109             auto C = node;
110             auto P = parent(node);
111             auto G = grandparent(node);
112 
113             if (C == P->right and P == G->left)
114             {
115                 rotate_left(G, P, C);
116                 P = C;
117             } else if (node == P->left and P == G->right)
118             {
119                 rotate_right(G, P, C);
120                 P = C;
121             }
122 
```

Restauração das propriedades no cenário D

```
123             C = P;
124             P = G;
125             G = parent(P);
126
127             if (C == P->left)
128                 rotate_right(G, P, C);
129             else
130                 rotate_left(G, P, C);
131
132             // Corner case: após a rotação C é a nova raiz
133             if (G == nullptr)
134                 root = C;
135
136             C->color = Node::BLACK;
137             P->color = Node::RED;
138         }
139     }
140
```

Referências

1. Red-Black Trees, acesso em 27/03/2019.
2. 8.2 Red-Black Trees, acesso em 27/03/2019.
3. Wikipédia. *Red-Black Tree*, acesso em 27/03/2019.[1]

[1] https://en.wikipedia.org/wiki/Red-black_tree