Árvores

Árvores *Red-Black* – Parte II: Remoção

Prof. Edson Alves - UnB/FGA

2018

Sumário

Remoção

Remoção em árvores red-black

- De forma semelhante ao que acontece com as árvores binárias de busca (pois as árvores *red-black* são árvores binárias de busca), a remoção deve ser tratada em três casos distintos
- Os casos dependente do número de filhos que não sejam folhas do nó N a ser removido: nenhum, um ou dois
- O caso de dois filhos pode ser removido a um dos dois casos anteriores, usando a mesma estratégia da remoção por cópia: basta substituir a informação do nó N pela informação do nó mais à direita D da sub-árvore à esquerda e proceder com a remoção fazendo $N = D$
- Assim, a partir deste ponto, será considerado que o nó N a ser removido tem, no máximo, um filho C que não seja folha

Remoção: caso trivial

- O caso trivial da remoção ocorre quando a informação a ser removida não consta na árvore
- Neste caso não há o que remover, e a rotina deve ser encerrada
- Para identificar tal caso, é necessário o auxílio de uma função auxiliar, que localiza o ponteiro do nó a ser removido
- Caso esta rotina não localize o nó, o ponteiro retornado será nulo e a remoção será encerrada
- Outra rotina auxiliar útil é a que reduz o caso 3 (o nó N a ser removido tem dois filhos que não são folhas) para o caso 1 ou o caso 2
- Esta rotina deve retornar o ponteiro do novo nó a ser removido

Implementação das rotinas auxiliares

```
142     Node * find(Node *node, const T& info)
143     {
144         if (node == nullptr or node->info == info)
145             return node;
146 
147         return info < node->info ? find(node->left, info) :
148             find(node->right, info);
149     }
150 
151     // Troca de informação com o nó mais à direita da sub-árvore
152     // à esquerda de N, e retorna um ponteiro para D
153     Node * swap_info(Node *N)
154     {
155         auto D = N->left;
156 
157         while (D->right)
158             D = D->right;
159 
160         std::swap(N->info, D->info);
161         return D;
162     }
```

Implementação das rotinas auxiliares

```
163
164     // Troca as posições do pai N e seu filho C
165     void swap_nodes(Node *N, Node *C)
166     {
167         auto P = parent(N);
168
169         if (C)
170             C->parent = P;
171
172         if (P == nullptr)
173             root = C;
174         else if (P->left == N)
175             P->left = C;
176         else
177             P->right = C;
178     }
179
```

Implementação da rotina de remoção

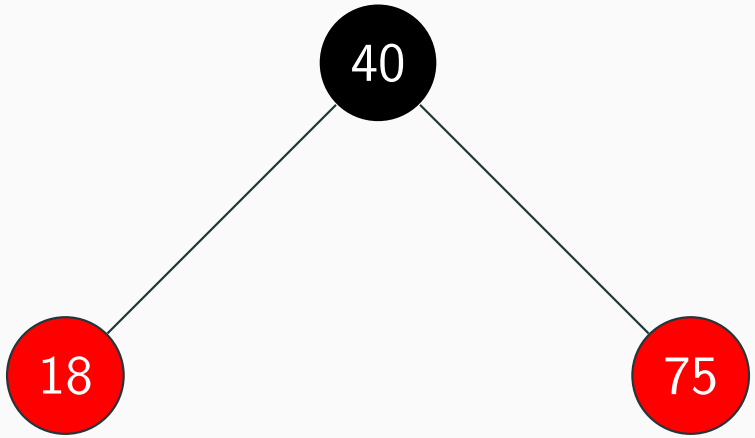
```
180 public:
181     void erase(const T& info)
182     {
183         Node *N = find(root, info);
184 
185         if (N == nullptr)       // Caso trivial
186             return;
187 
188         // Reduz o caso 3 ao caso 1 ou ao caso 2
189         if (N->left and N->right)
190             N = swap_info(N);
191 
192         erase(N);
193     }
194 
```

Remoção de nó vermelho

- Se N é vermelho, pela propriedade 2 seu filho C deve ser preto
- A substituição de N por C mantém as propriedades da árvore *red-black*
- Primeiramente, se C vir a ocupar a raiz da árvore, esta será preta, mantendo a propriedade 2
- O pai P de N é necessáriamente preto, uma vez que N é vermelho, de modo que a propriedade 4 não é violada
- Por fim, como o nó removido é vermelho, o número de nós pretos nos caminhos não se altera, mantendo a propriedade 5
- Observe que este caso só ocorre se ambos filhos de N são folhas (o caso de apenas uma folha violaria a propriedade 5 de uma árvore *red-black*)

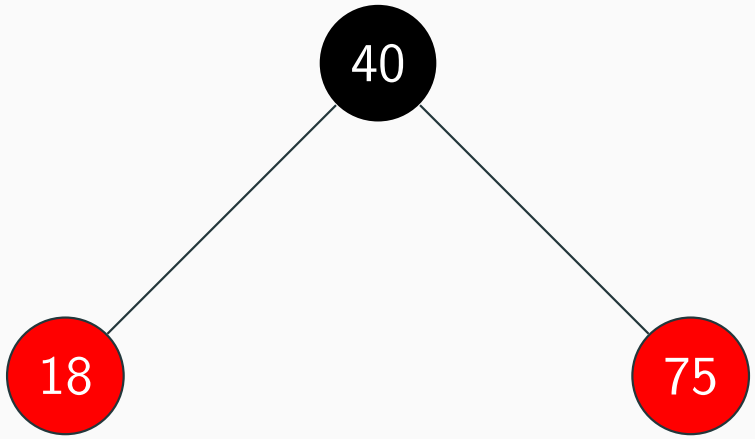
Exemplo de remoção de nó vermelho

Informação a ser removida: 40



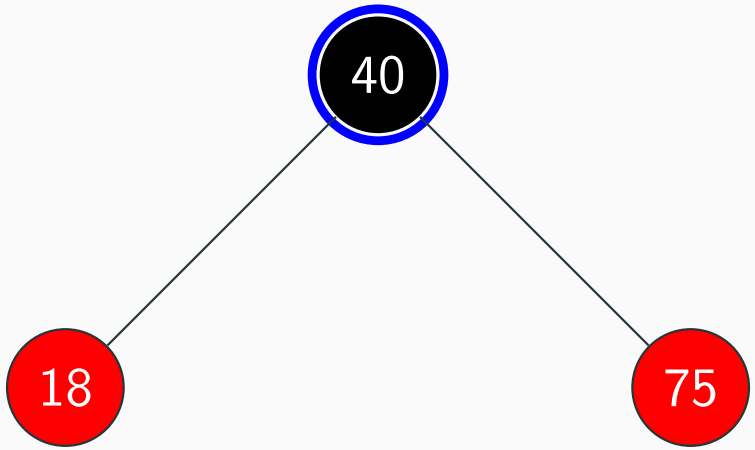
Exemplo de remoção de nó vermelho

40 tem dois filhos que não são folhas



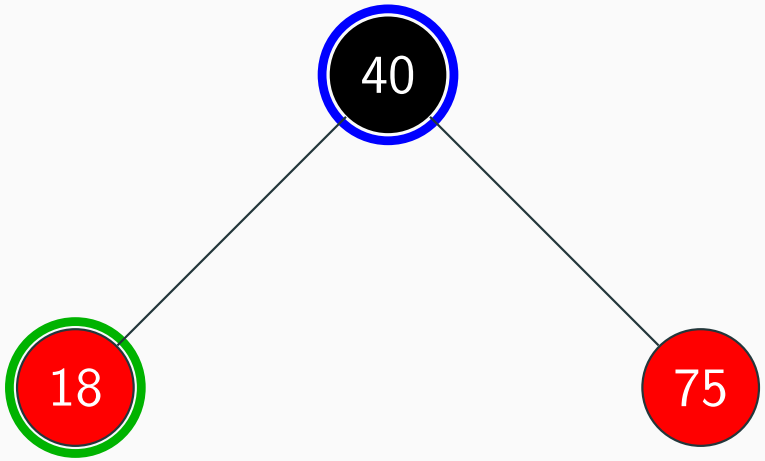
Exemplo de remoção de nó vermelho

Redução ao caso 1 ou ao caso 2



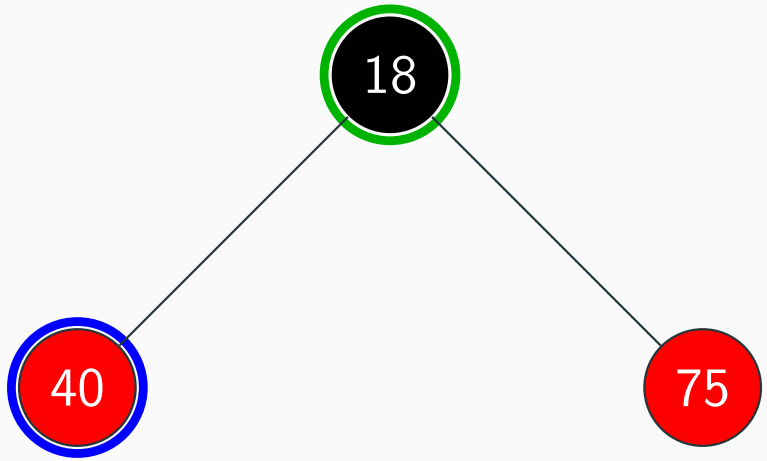
Exemplo de remoção de nó vermelho

Redução ao caso 1 ou ao caso 2



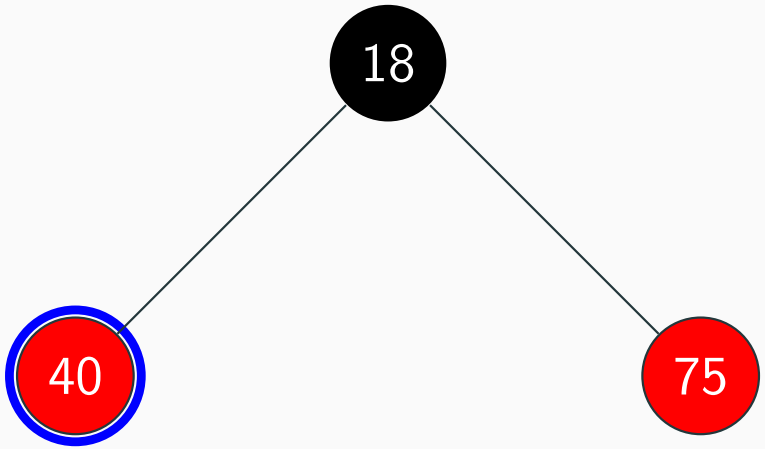
Exemplo de remoção de nó vermelho

Redução ao caso 1 ou ao caso 2



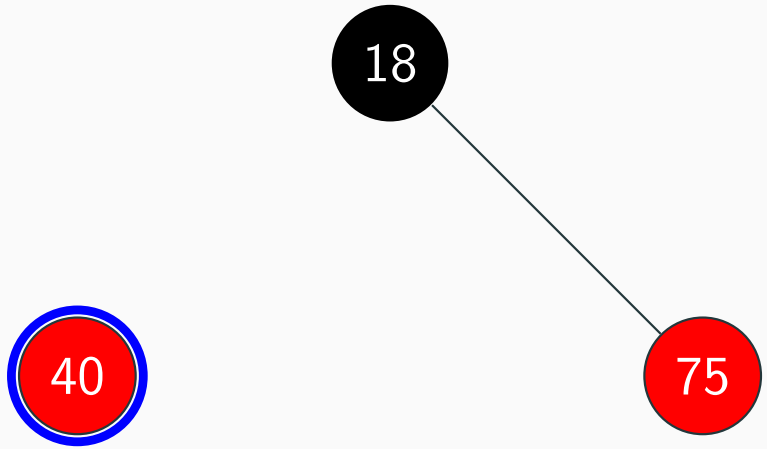
Exemplo de remoção de nó vermelho

Remoção de nó vermelho sem filhos que não sejam folhas



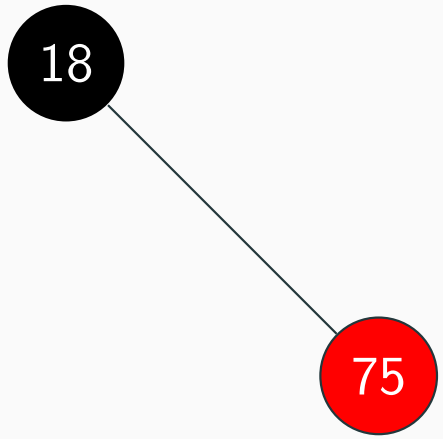
Exemplo de remoção de nó vermelho

Remoção de nó vermelho sem filhos que não sejam folhas



Exemplo de remoção de nó vermelho

Remoção de nó vermelho sem filhos que não sejam folhas



Implementação da remoção de nó vermelho

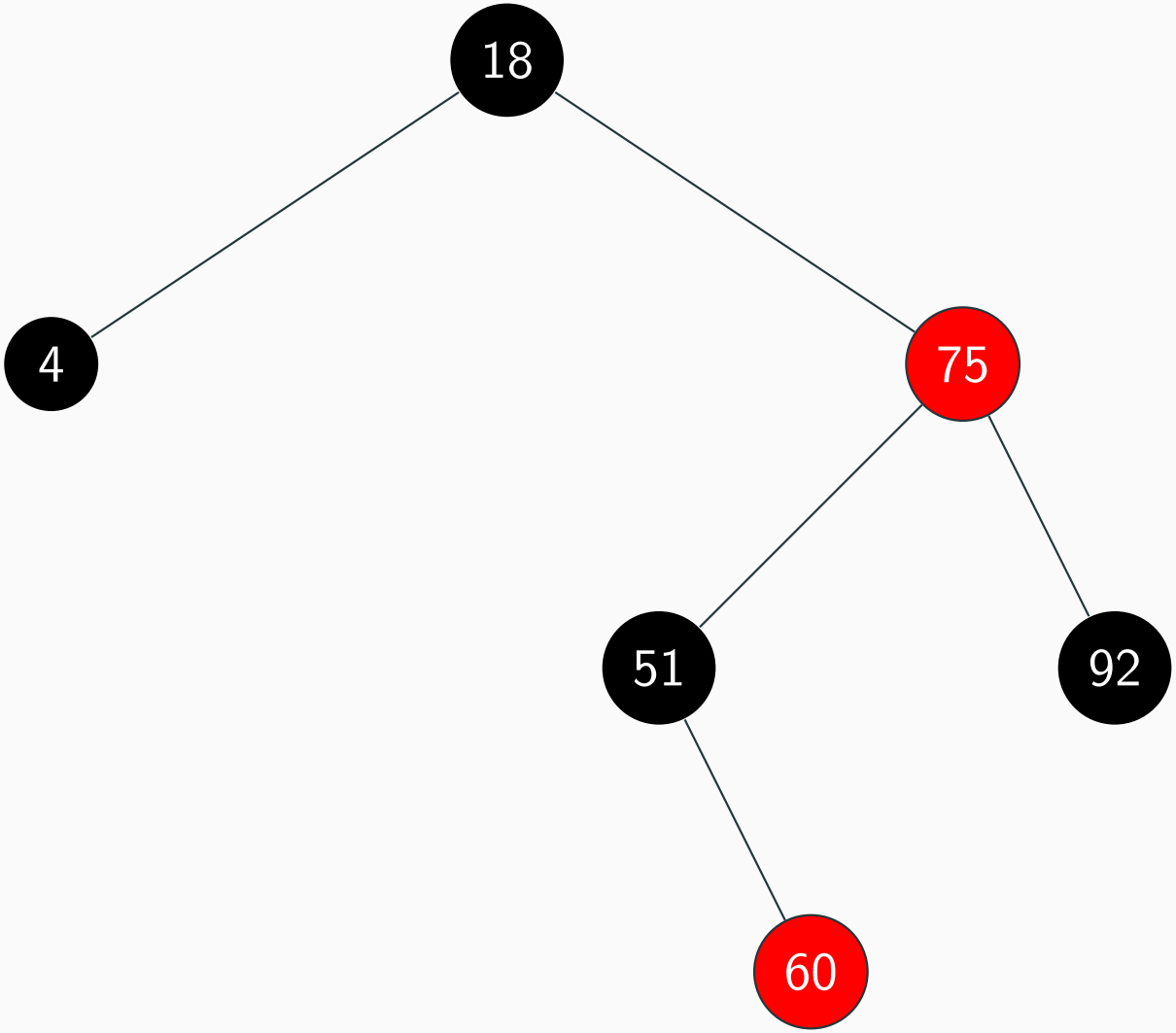
```
195 private:
196     void erase(Node *N)
197     {
198         // As relações de parentesco devem ser determinadas
199         // antes da troca de N por C
200 
201         auto P = parent(N);
202         auto S = sibling(N);
203         auto C = N->left ? N->left : N->right;
204 
205         swap_nodes(N, C);
206 
207         // Se N é vermelho não há nada mais a fazer
208 
```

Remoção de nó preto com filho vermelho

- Se N é preto e seu filho C é vermelho, a remoção de N viola a propriedade 5, devido a redução no número de nós pretos
- Além disso, a promoção de um nó vermelho pode levar a violação da propriedade 4
- Uma forma de preservar ambas propriedades é recolorir C como um nó preto
- Isto restaura a violação da propriedade 5, pois a perda de N agora é compensada com a adição de um novo nó preto
- O fato de C assumir a cor preta evita que a propriedade 4 seja violada

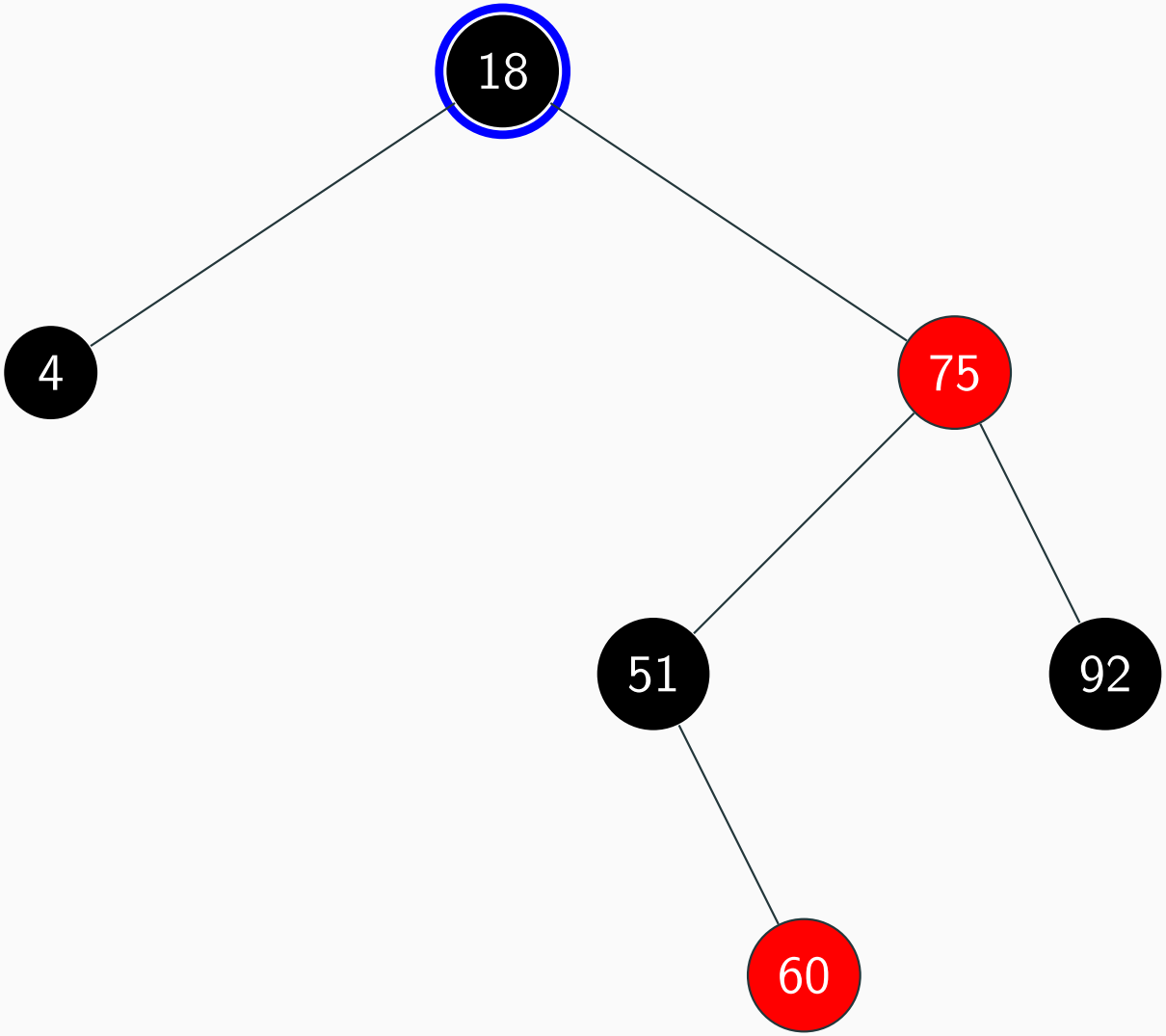
Exemplo de remoção de nó vermelho

Informação a ser removida: 51



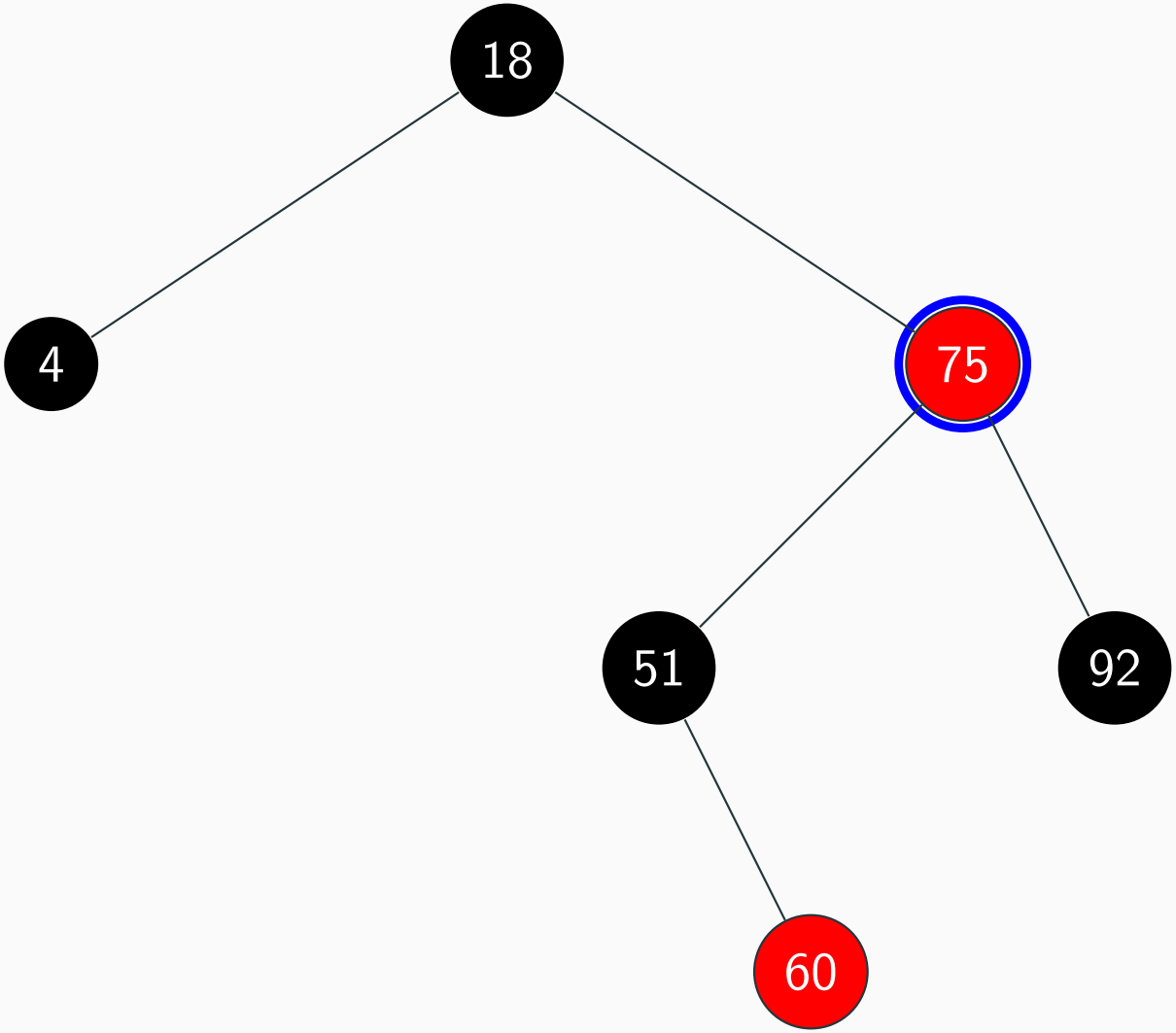
Exemplo de remoção de nó vermelho

Informação a ser removida: 51



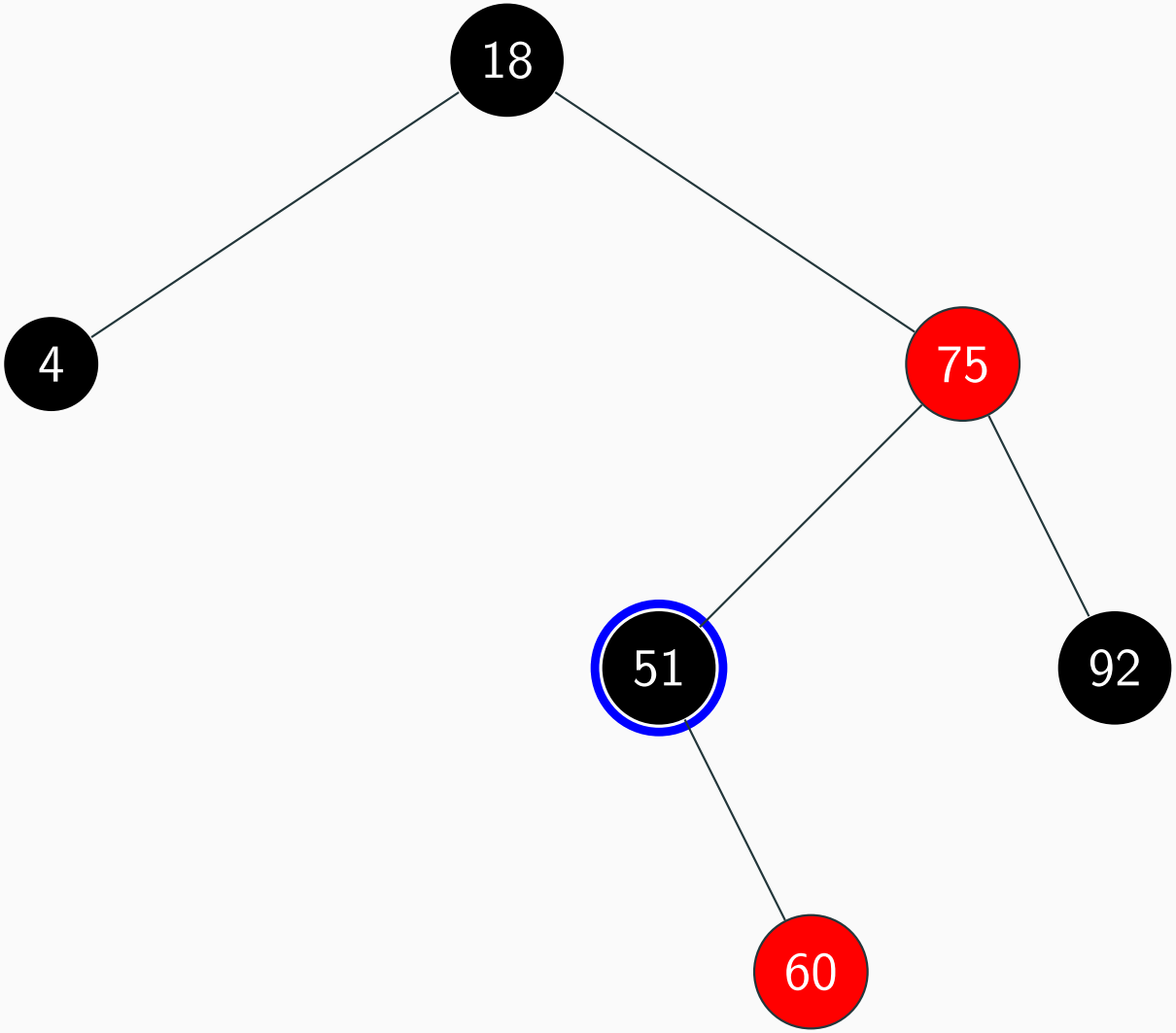
Exemplo de remoção de nó vermelho

Informação a ser removida: 51



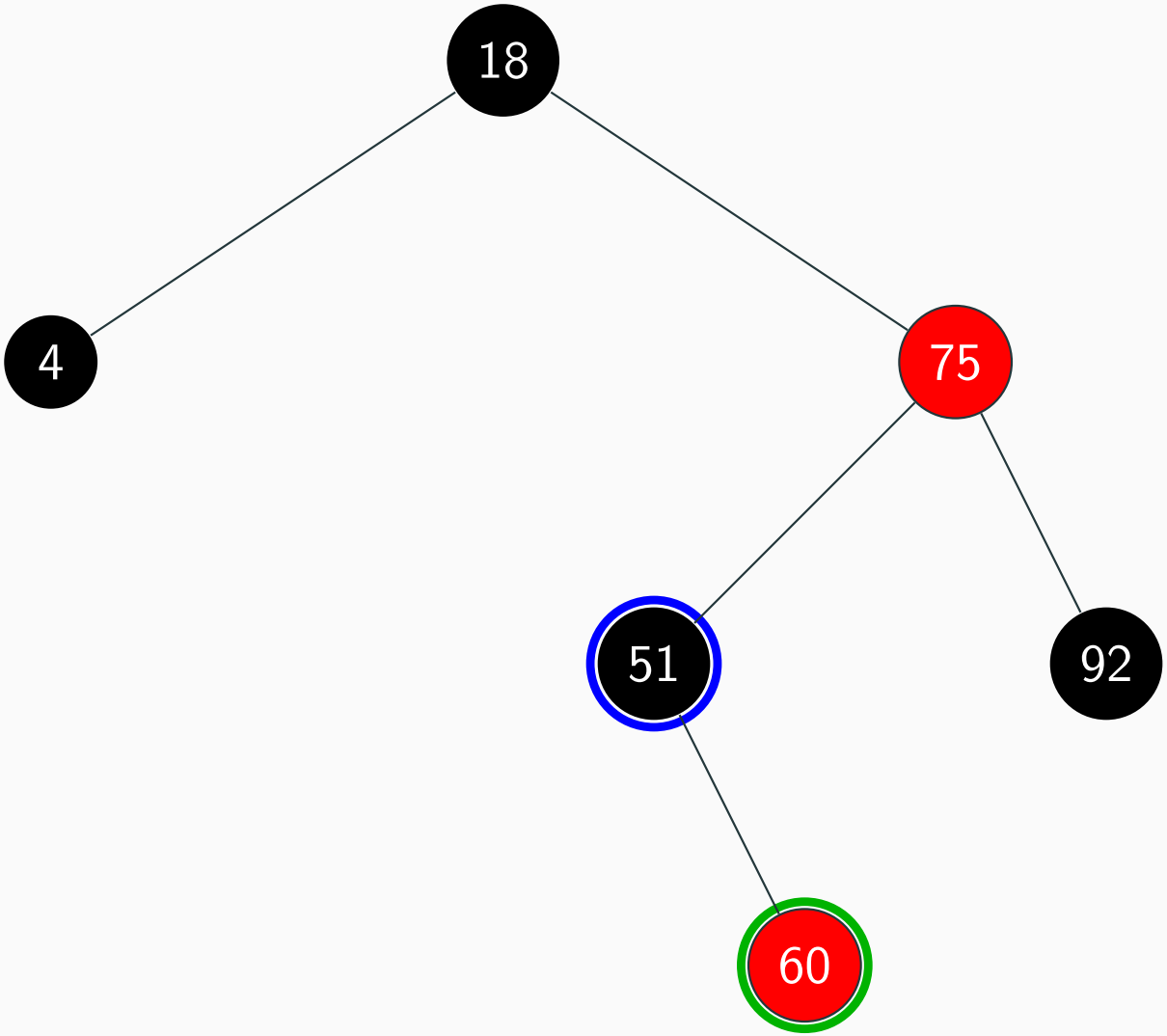
Exemplo de remoção de nó vermelho

Informação a ser removida: 51



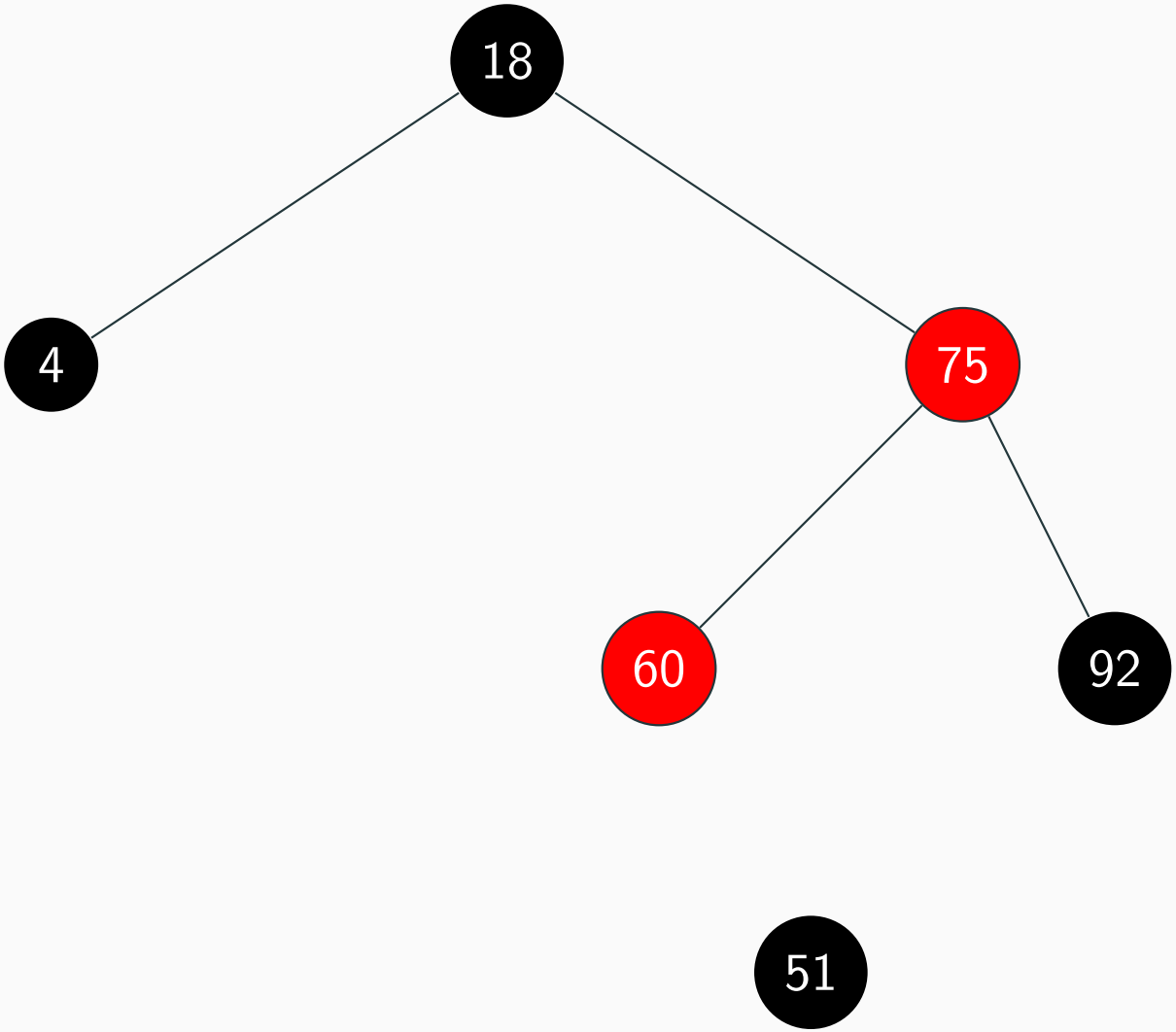
Exemplo de remoção de nó vermelho

Nó preto com filho vermelho



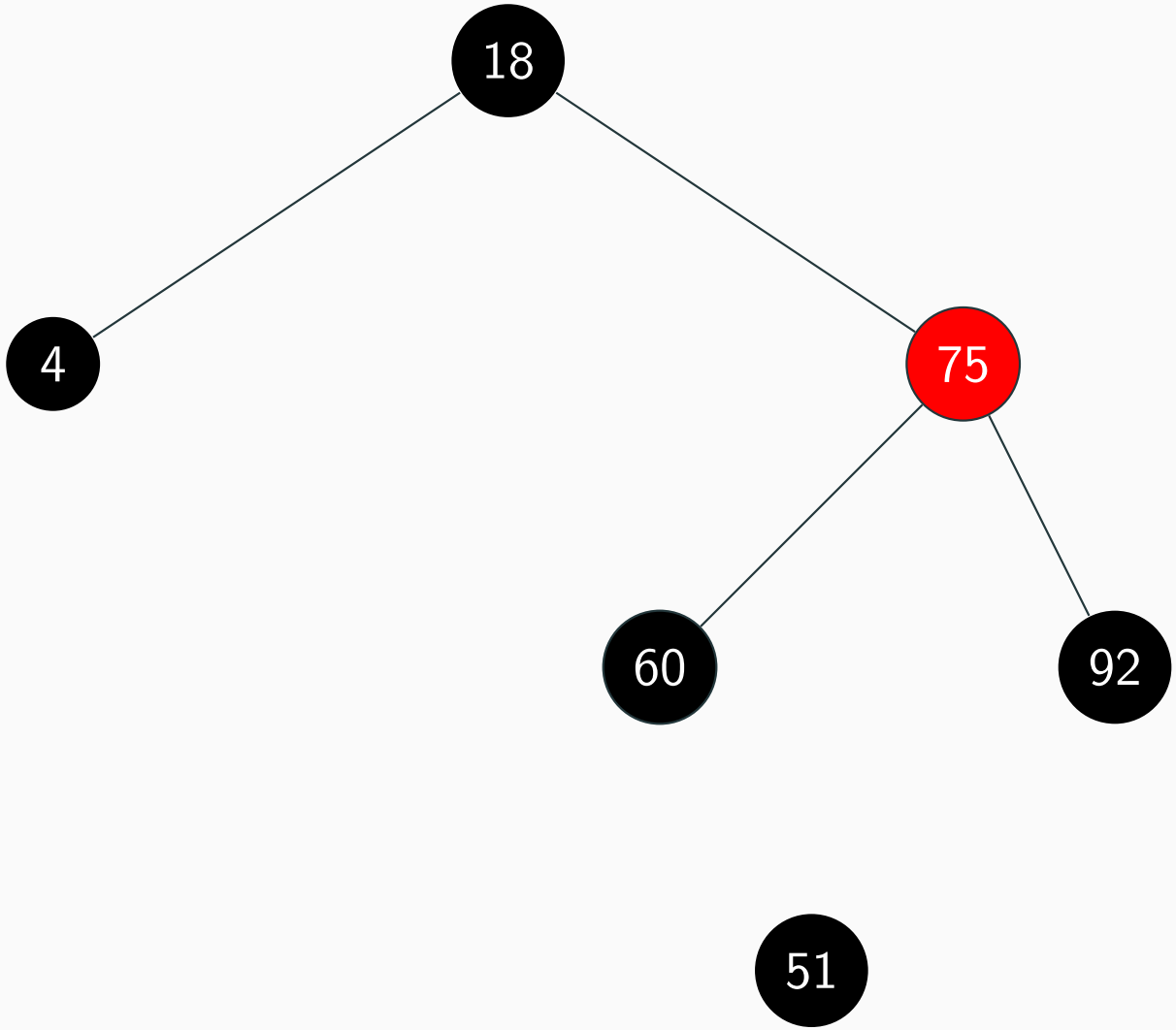
Exemplo de remoção de nó vermelho

Nó preto com filho vermelho



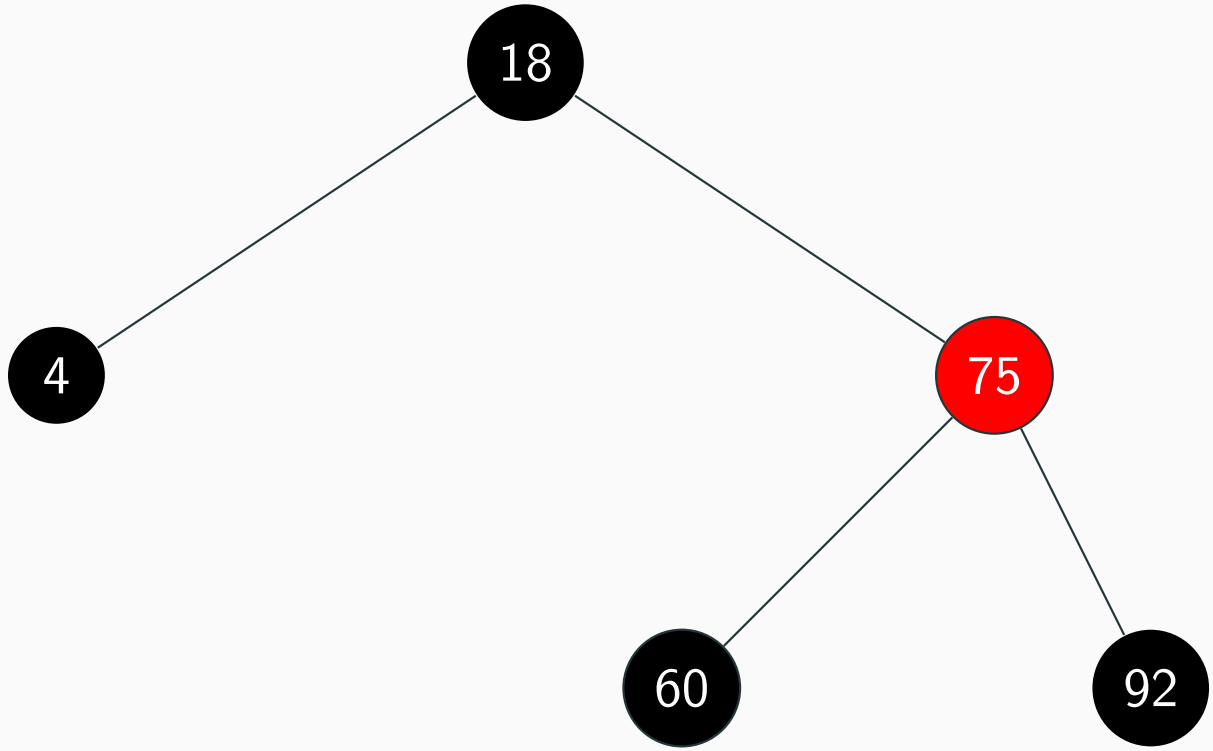
Exemplo de remoção de nó vermelho

Nó preto com filho vermelho



Exemplo de remoção de nó vermelho

Nó preto com filho vermelho



Implementação da remoção de nó preto com filho vermelho

```
209         if (N->color == Node::BLACK)
210         {
211             // N é preto e o filho não-folha C é vermelho
212             if (C and C->color == Node::RED)
213                 C->color = Node::BLACK;
214             else
215                 rebalancing(P, S, C);
216         }
217 
218         delete N;
219     }
220 
```

Nó preto com filho preto

- O caso onde ambos N e C são pretos é o mais complexo dentre todos os que envolvem um nó com, no máximo, um filho não-folha
- A remoção de um nó preto viola a propriedade 5 das árvores *red-black*
- Há múltiplos cenários possíveis, cada um tendo que ser tratado adequadamente, por meio do rebalanceamento da árvore
- Observe que este caso ocorre apenas quando ambos filhos de N são folhas
- Isto porque se N tivesse apenas uma folha preta, o fato do outro filho não ser folha violaria a propriedade 5

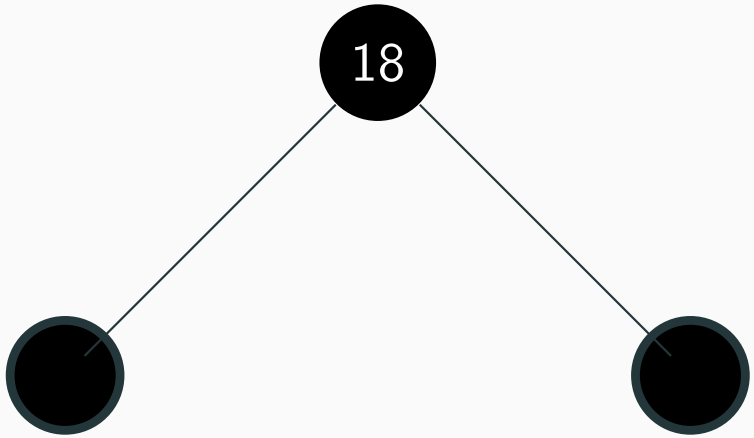
Cenário A: após a troca de N e C, C é raiz

- Este é o cenário mais simples
- A remoção de um nó preto da posição raiz subtrai igualmente uma unidade de todos os caminhos da raiz às folhas
- Assim a propriedade 5 fica preservada
- As demais propriedades também se mantém: como as folhas são pretas, a raiz também será preta

```
221     void rebalancing(Node *P, Node *S, Node *N)
222     {
223         // N é a raiz da árvore
224         if (P == nullptr)
225             return;
226
```

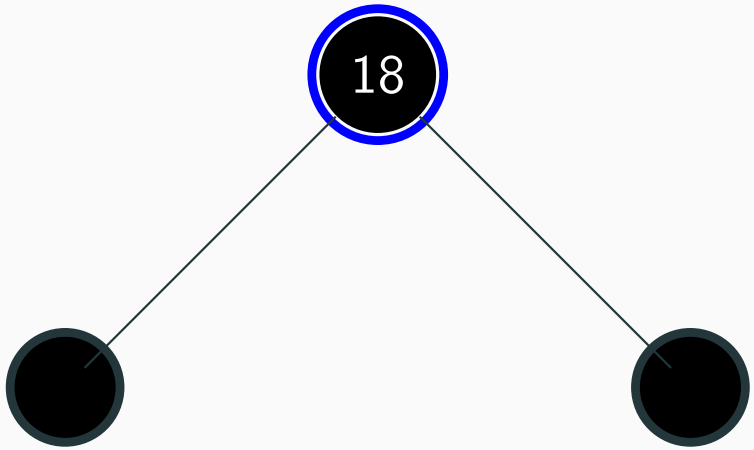
Exemplo de remoção no cenário A

Informação a ser removida: 18



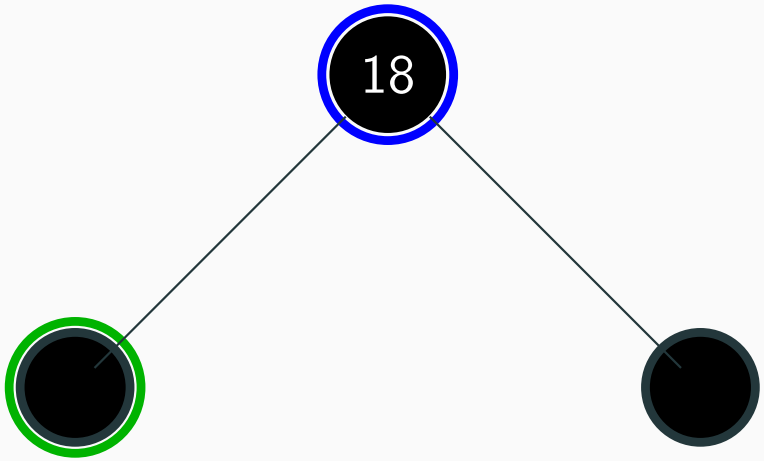
Exemplo de remoção no cenário A

Informação a ser removida: 18



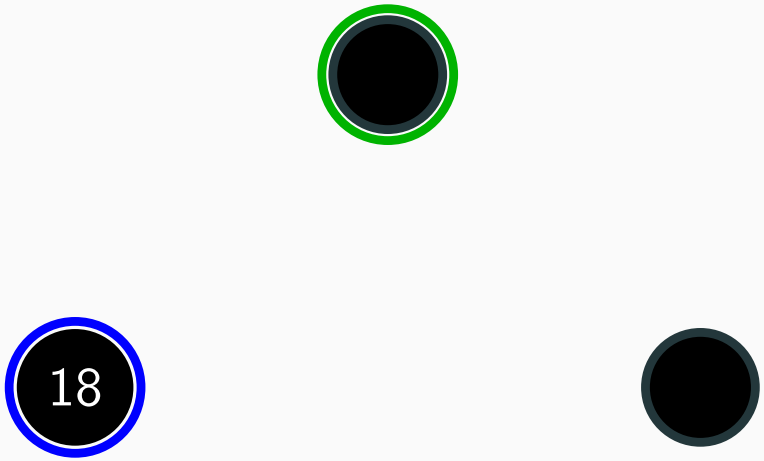
Exemplo de remoção no cenário A

Informação a ser removida: 18



Exemplo de remoção no cenário A

Informação a ser removida: 18



Exemplo de remoção no cenário A

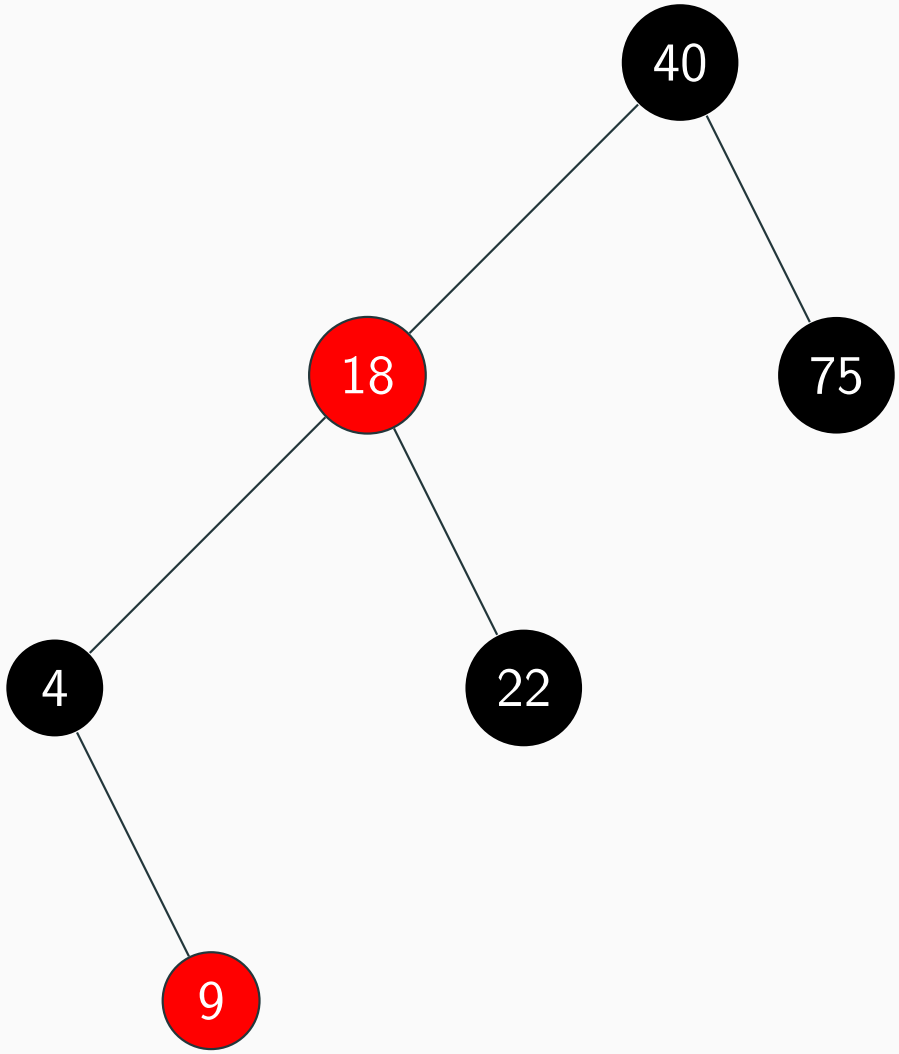
Informação a ser removida: 18

Cenário B: Irmão vermelho

- Se o nó a ser removido tem um irmão S vermelho, é preciso promover um reposicionamento dos nós, além de uma troca de cores entre o pai P e o irmão S
- Este cenário não pode ser resolvido diretamente: o resultado deste reposicionamento levará a um dos próximos cenários
- Primeiramente as cores de P e S devem ser trocadas
- Uma rotação de S em torno de P, o torna o novo avô de N
- Ao final deste processo uma das duas subárvores terá caminho da raiz até uma folha uma unidade menor do que a outra, violando a propriedade 5
- Porém o caso a ser tratado mudou: agora N tem pai vermelho, com irmão preto
- Este cenário será abordado mais adiante

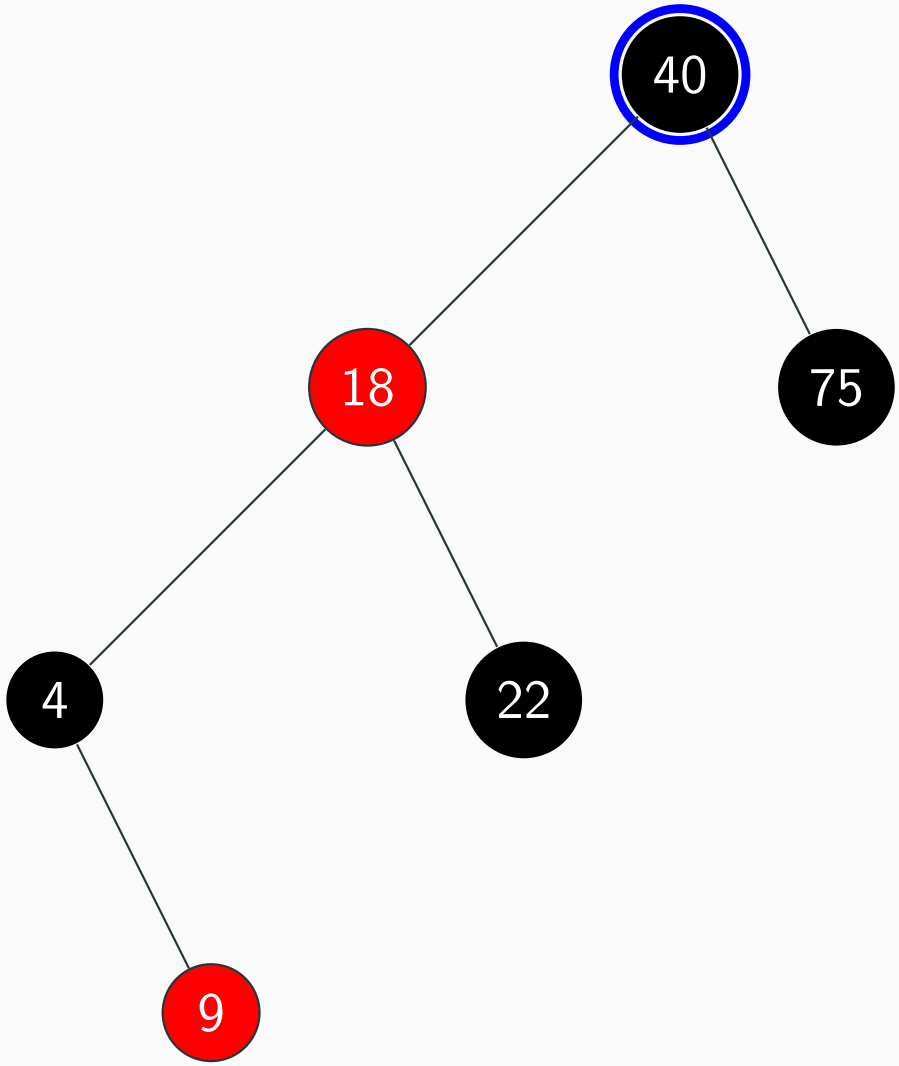
Exemplo de remoção no cenário B

Informação a ser removida: 75



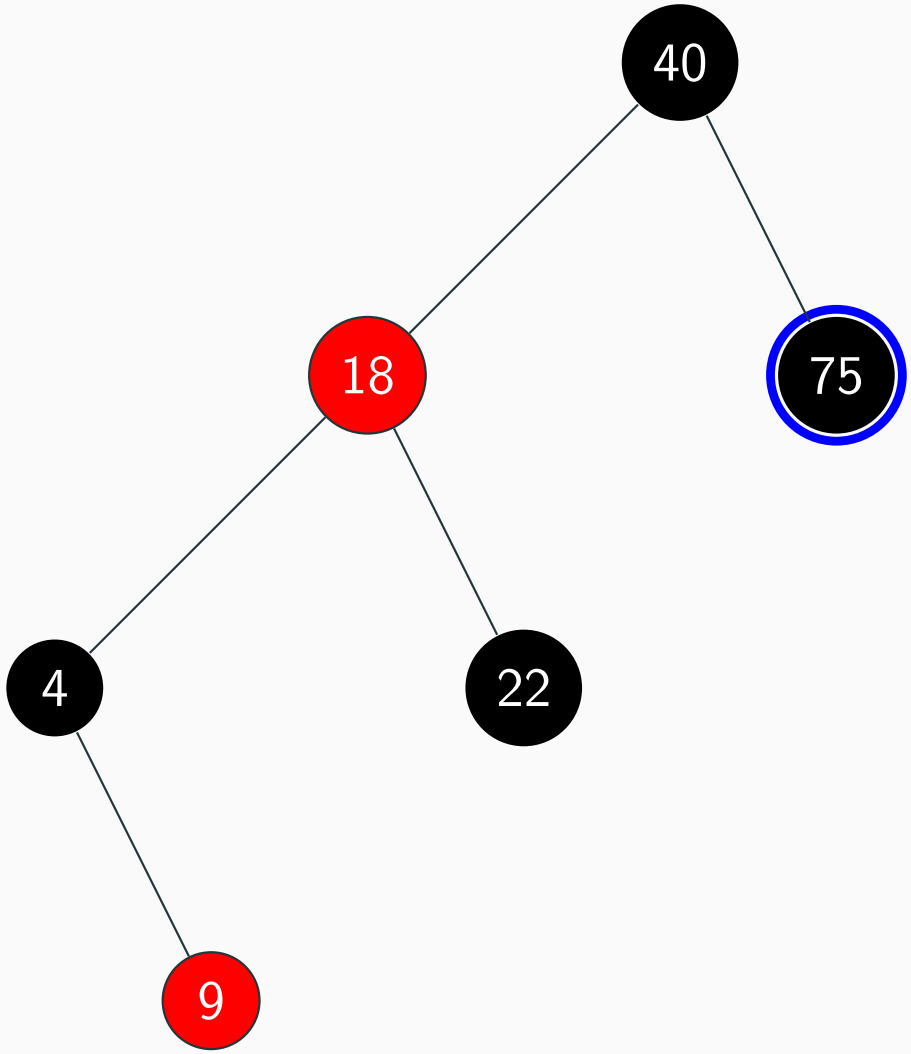
Exemplo de remoção no cenário B

Informação a ser removida: 75



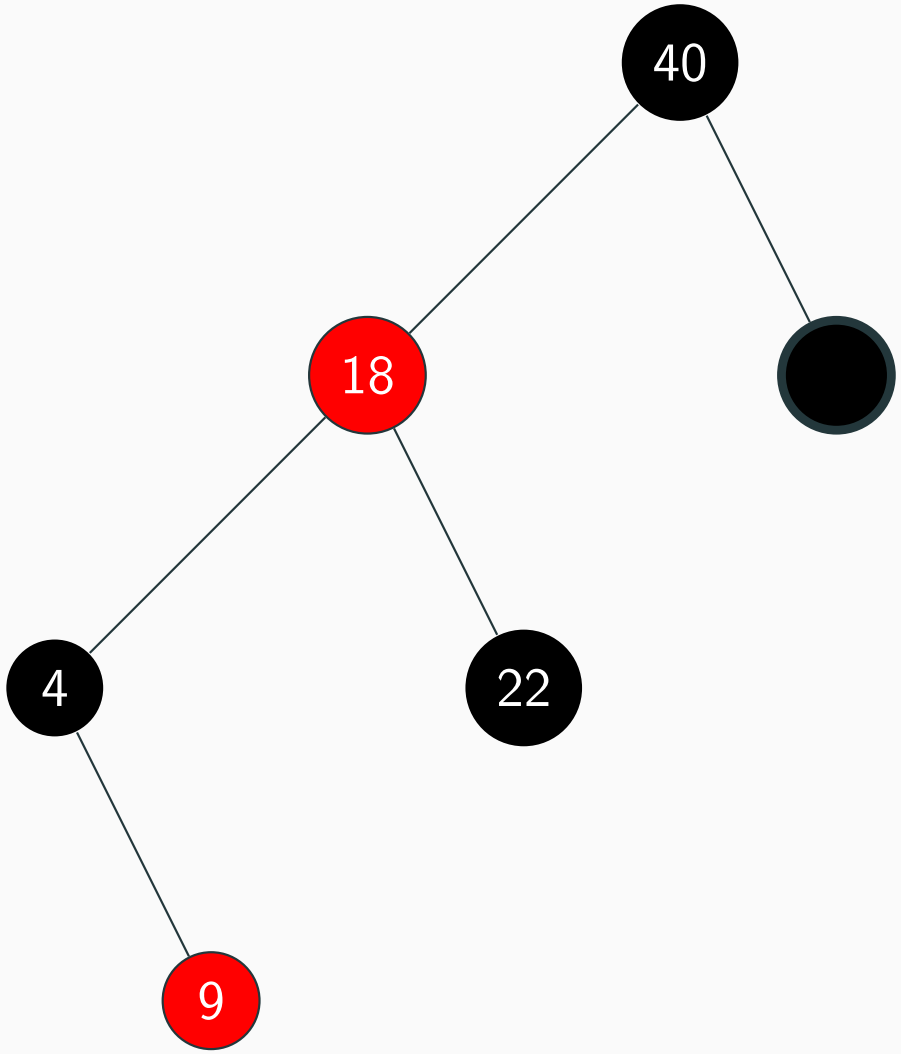
Exemplo de remoção no cenário B

Informação a ser removida: 75



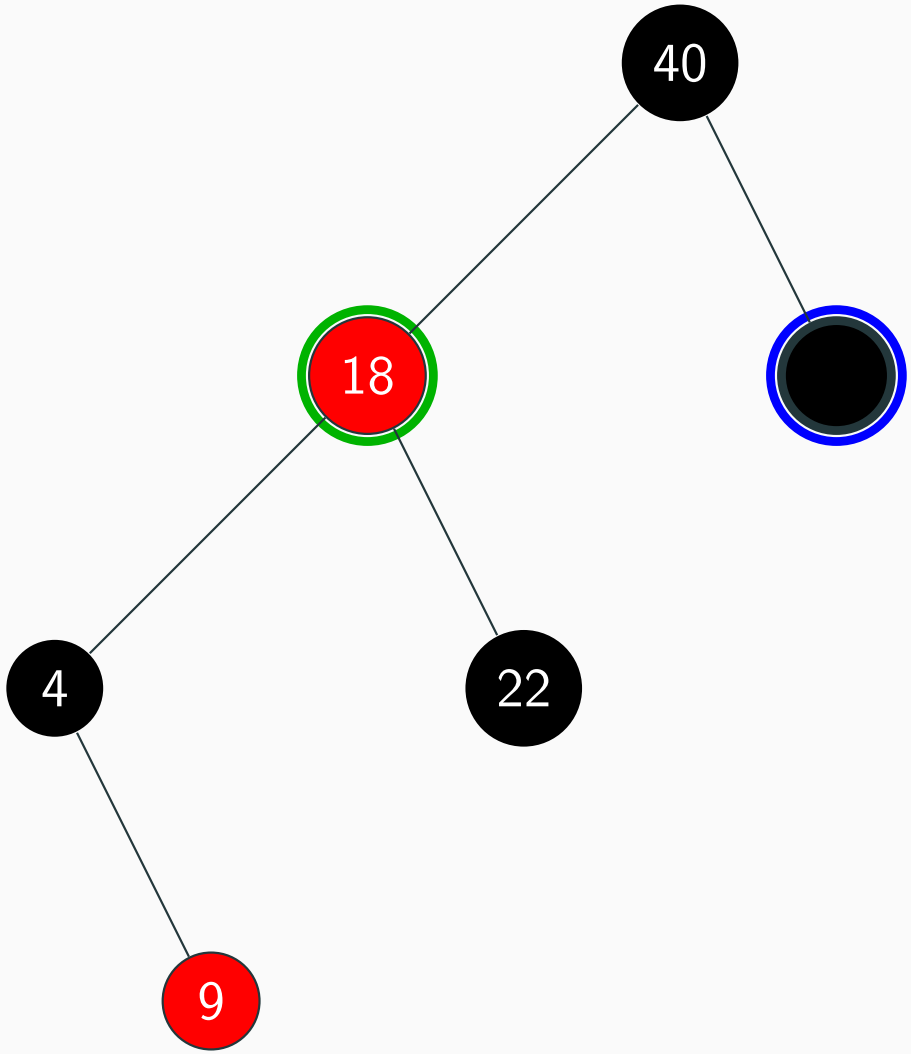
Exemplo de remoção no cenário B

Informação a ser removida: 75



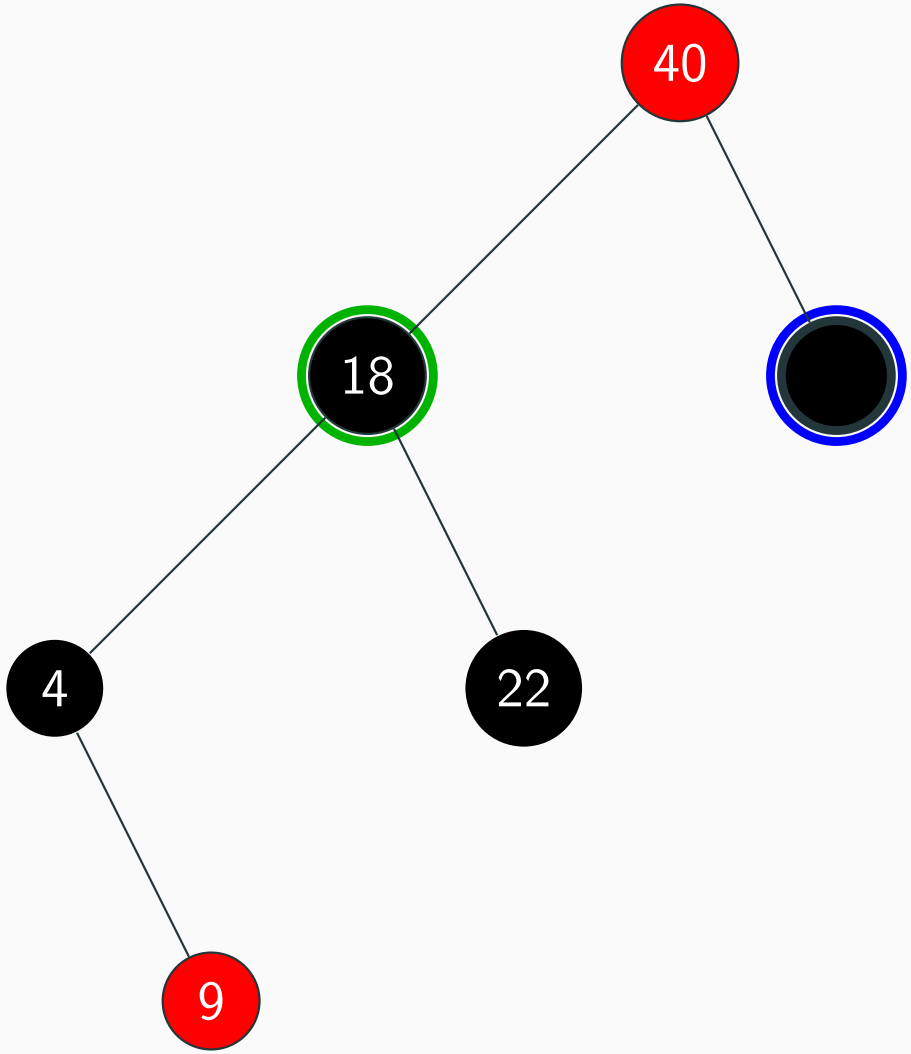
Exemplo de remoção no cenário B

Nó preto, irmão vermelho



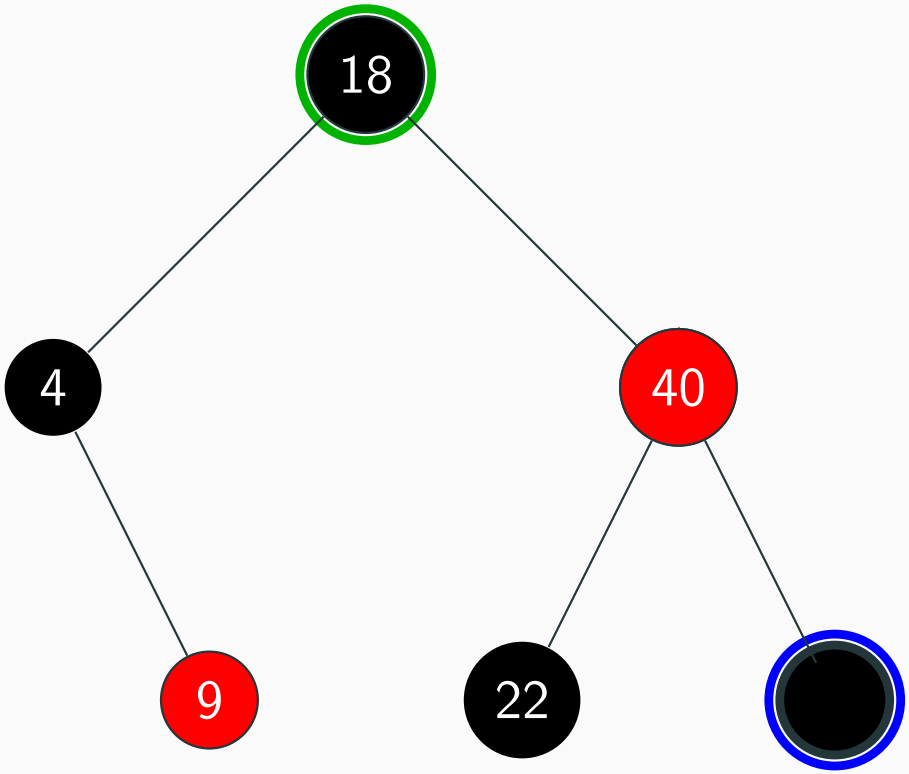
Exemplo de remoção no cenário B

Nó preto, irmão vermelho



Exemplo de remoção no cenário B

Nó preto, pai vermelho, irmão preto



Implementação do cenário B

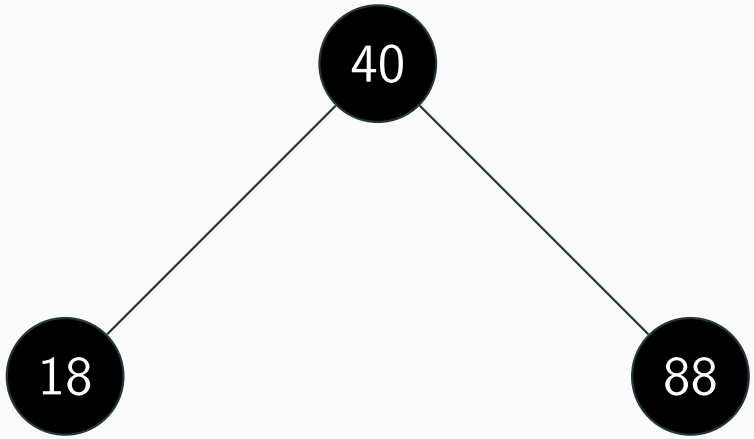
```
227         // N tem irmão S vermelho: este caso não se encerra no if
228         if (S and S->color == Node::RED)
229         {
230             P->color = Node::RED;
231             S->color = Node::BLACK;
232 
233             if (N == P->left)
234                 rotate_left(grandparent(S), P, S);
235             else
236                 rotate_right(grandparent(S), P, S);
237 
238             if (parent(S) == nullptr)
239                 root = S;
240 
241             // O irmão deve ser atualizado neste cenário
242             S = P->left == N ? P->right : P->left;
243         }
244 
```

Cenário C: Pai, irmão e sobrinhos pretos

- Neste caso é necessário recolorir S como vermelho
- Isto faz com que todos os caminhos que passem por S tenham um nó preto a menos
- Porém a remoção de N também reduz um nó preto dos caminhos que passavam por N
- Contudo, os caminhos que passam por P agora tem um nó preto a menos do que os caminhos que não passam, violando a propriedade 5
- Assim é preciso reiniciar o rebalanceamento em P

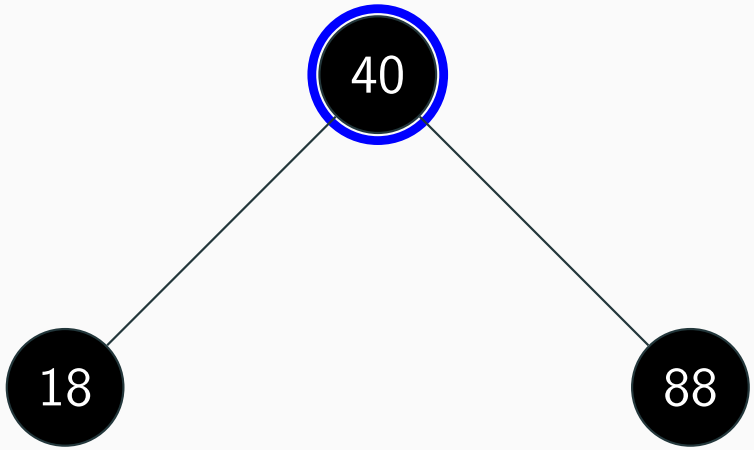
Exemplo de remoção no cenário C

Informação a ser removida: 88



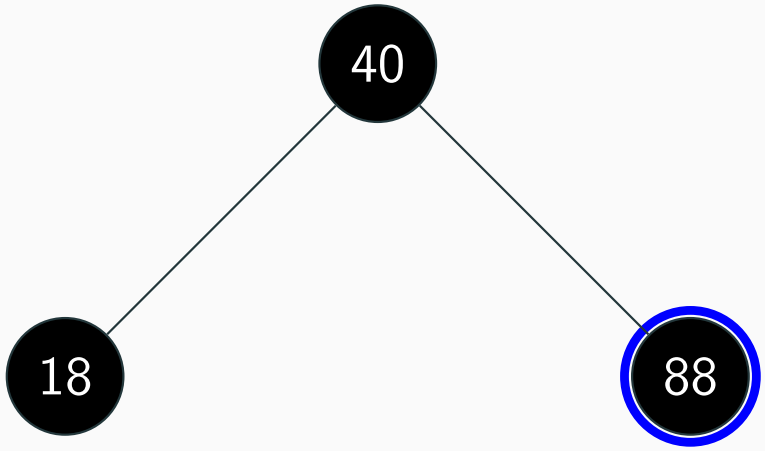
Exemplo de remoção no cenário C

Informação a ser removida: 88



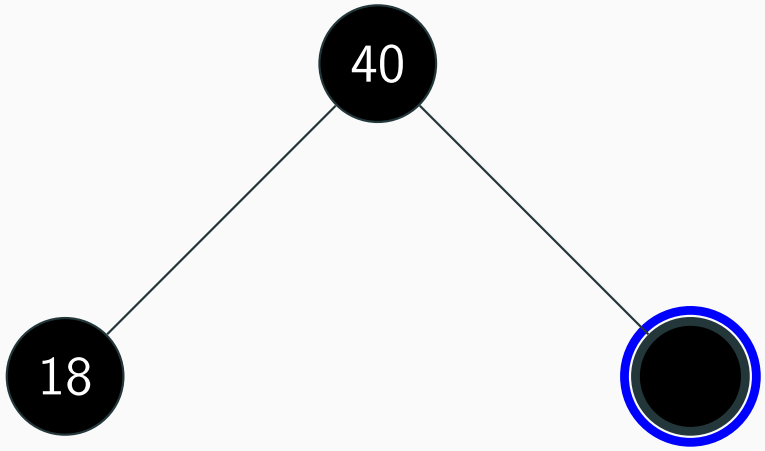
Exemplo de remoção no cenário C

Informação a ser removida: 88



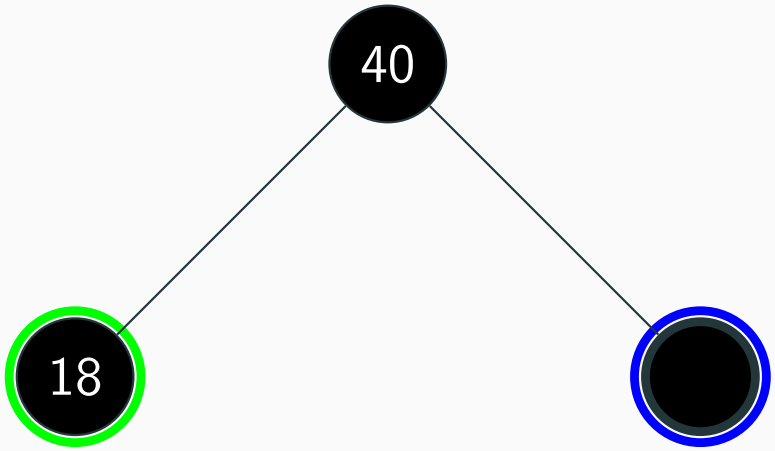
Exemplo de remoção no cenário C

Informação a ser removida: 88



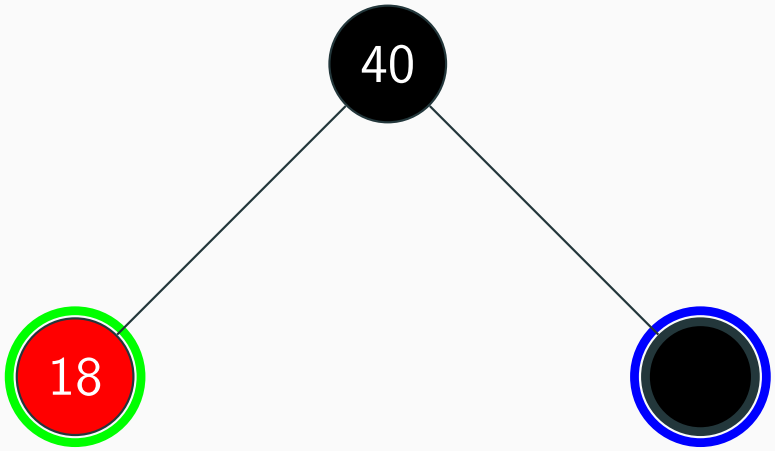
Exemplo de remoção no cenário C

Nó, pai, irmão e sobrinhos pretos



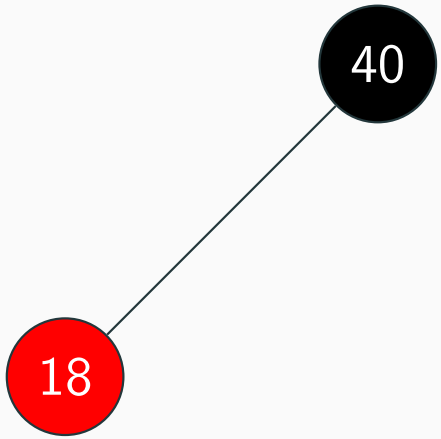
Exemplo de remoção no cenário C

Nó, pai, irmão e sobrinhos pretos



Exemplo de remoção no cenário C

Nó, pai, irmão e sobrinhos pretos



Implementação do cenário C

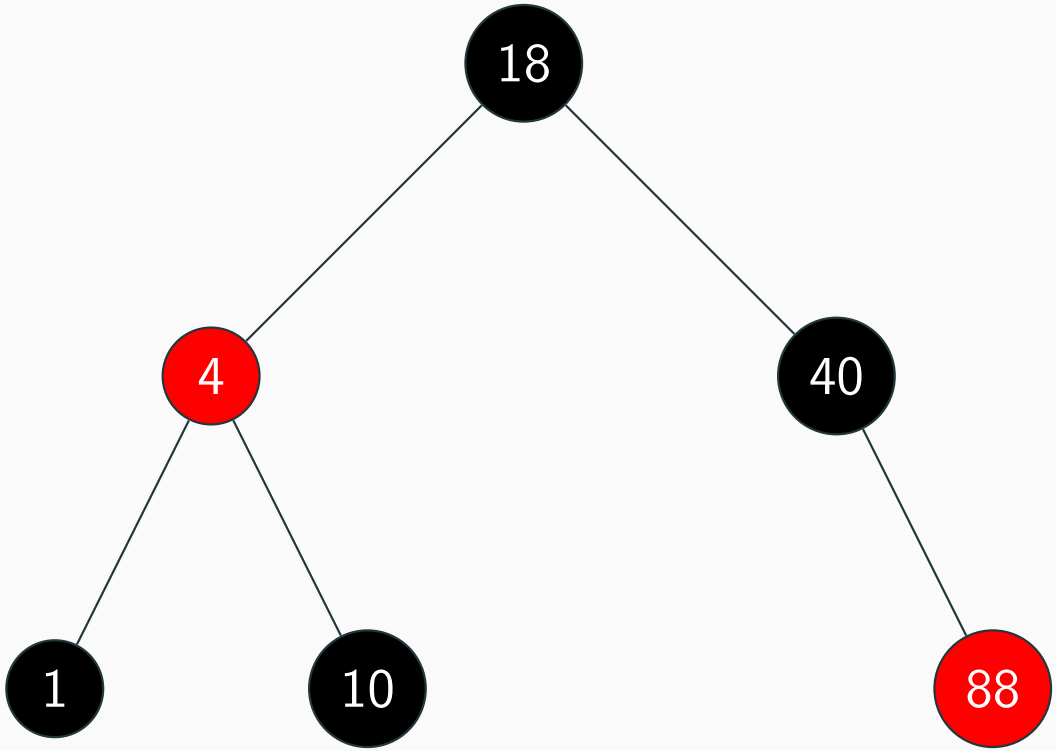
```
245         // N tem irmão, pai e sobrinhos pretos
246         if (P and P->color == Node::BLACK and S->color == Node::BLACK
247             and (S->left == nullptr or S->left->color == Node::BLACK)
248             and (S->right == nullptr or S->right->color == Node::BLACK))
249         {
250             S->color = Node::RED;
251             return rebalancing(parent(P), sibling(P), P);
252         }
253
```

Cenário D: Irmão e sobrinhos pretos, pai vermelho

- Este cenário é semelhante ao cenário C
- Contudo, a troca de cores em P e S, neste caso, não viola a propriedade 5, pois não há mudança no número de nós pretos nos caminhos que passam por S
- O número de nós pretos dos caminhos que passam por N aumenta em um, mas este número volta ao original após a remoção de N, o qual é preto
- Assim, este caso se encerra com esta troca de cores

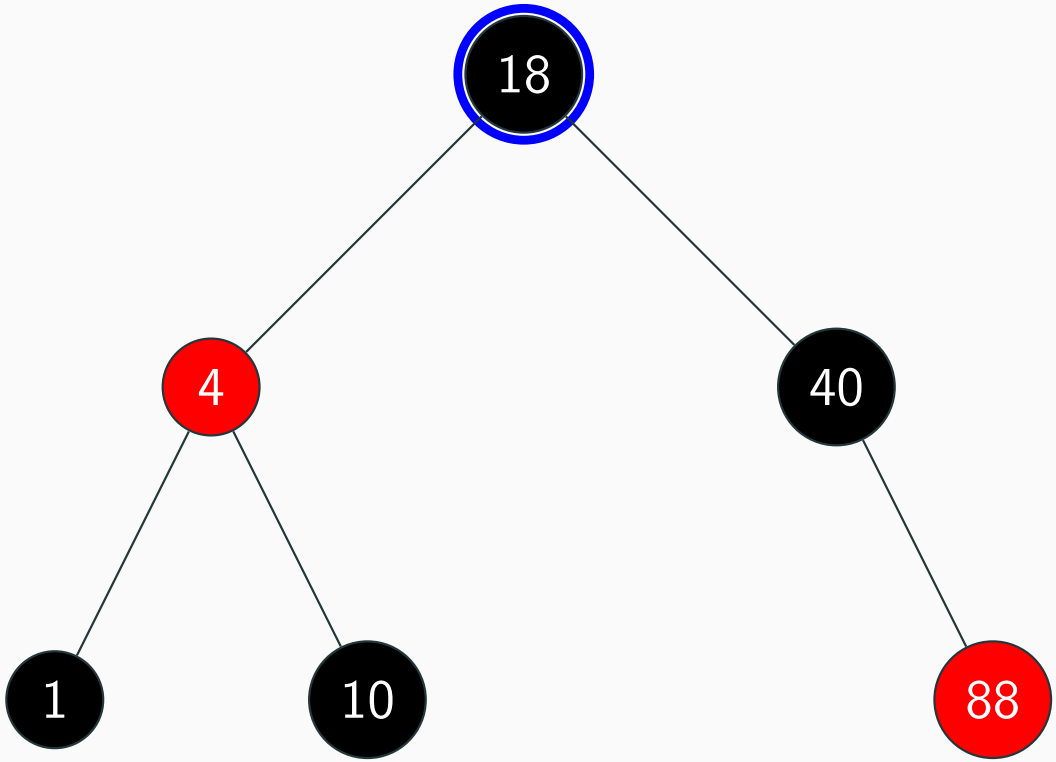
Exemplo de remoção no cenário D

Informação a ser removida: 10



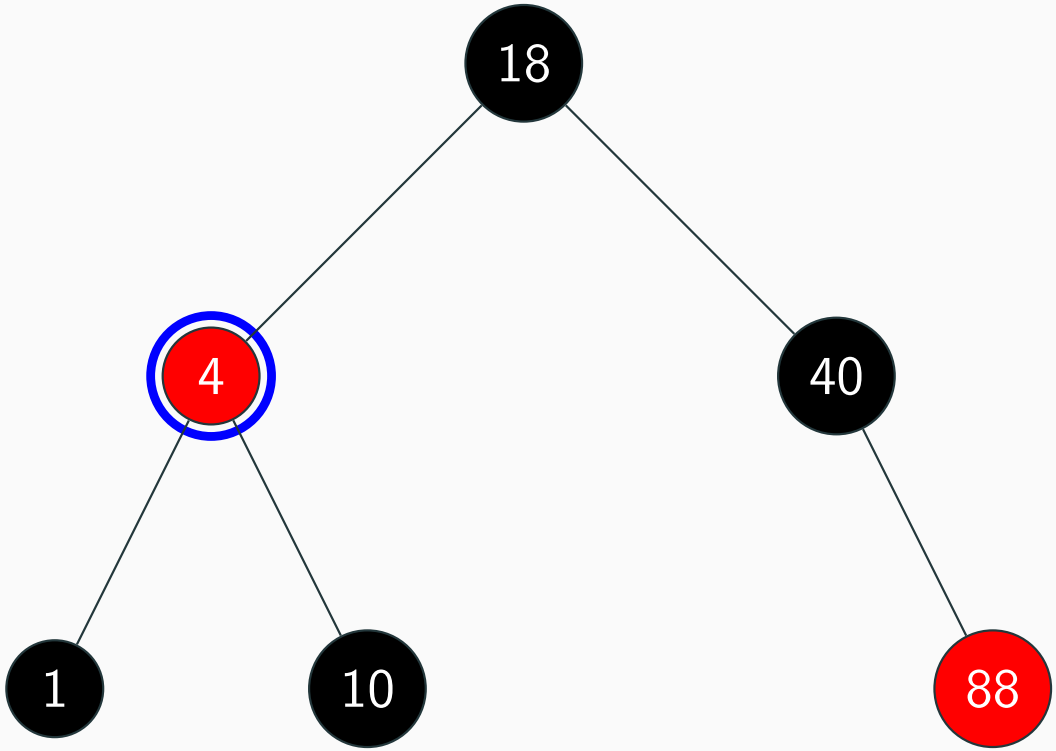
Exemplo de remoção no cenário D

Informação a ser removida: 10



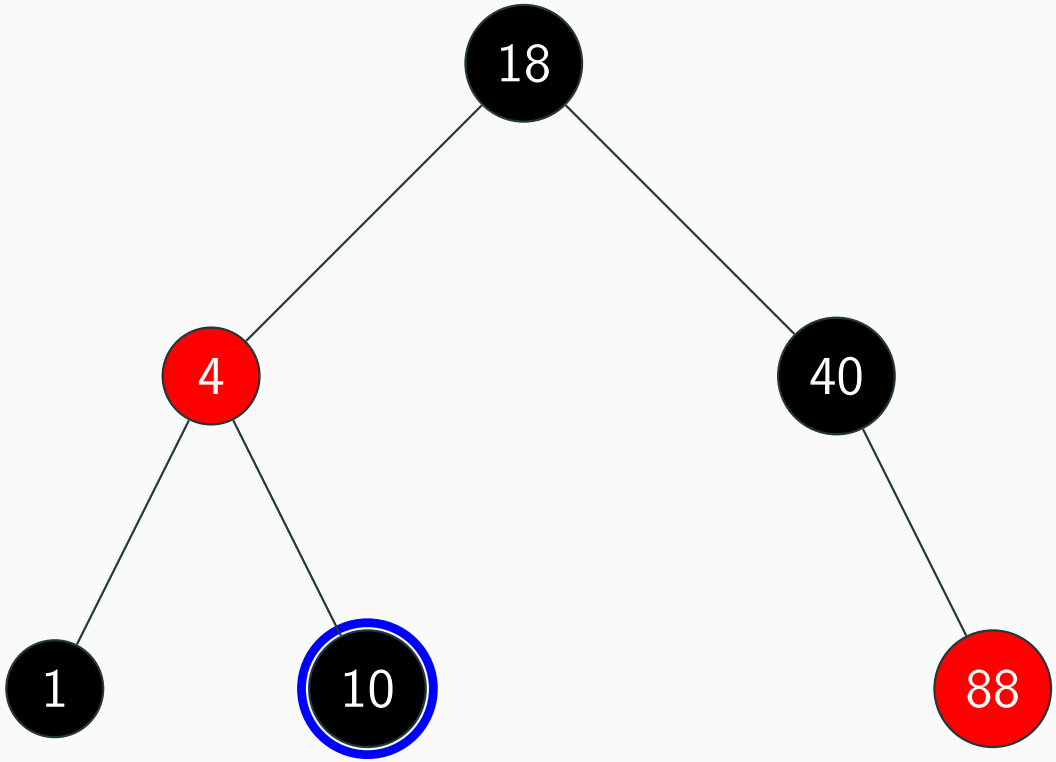
Exemplo de remoção no cenário D

Informação a ser removida: 10



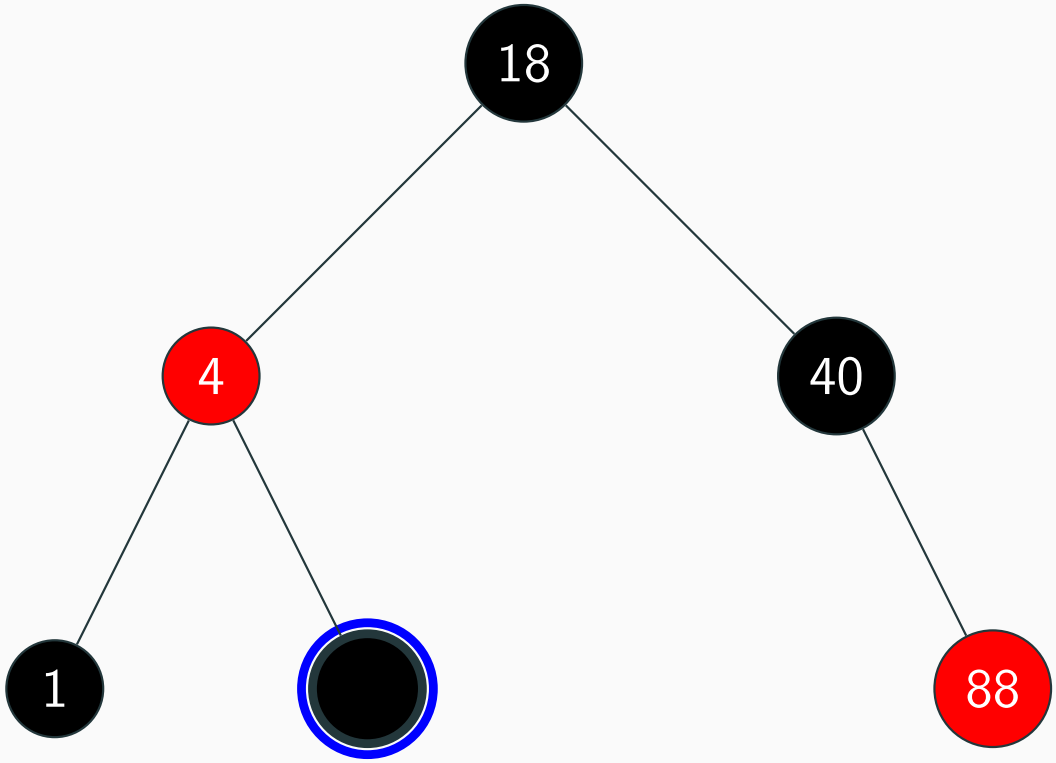
Exemplo de remoção no cenário D

Informação a ser removida: 10



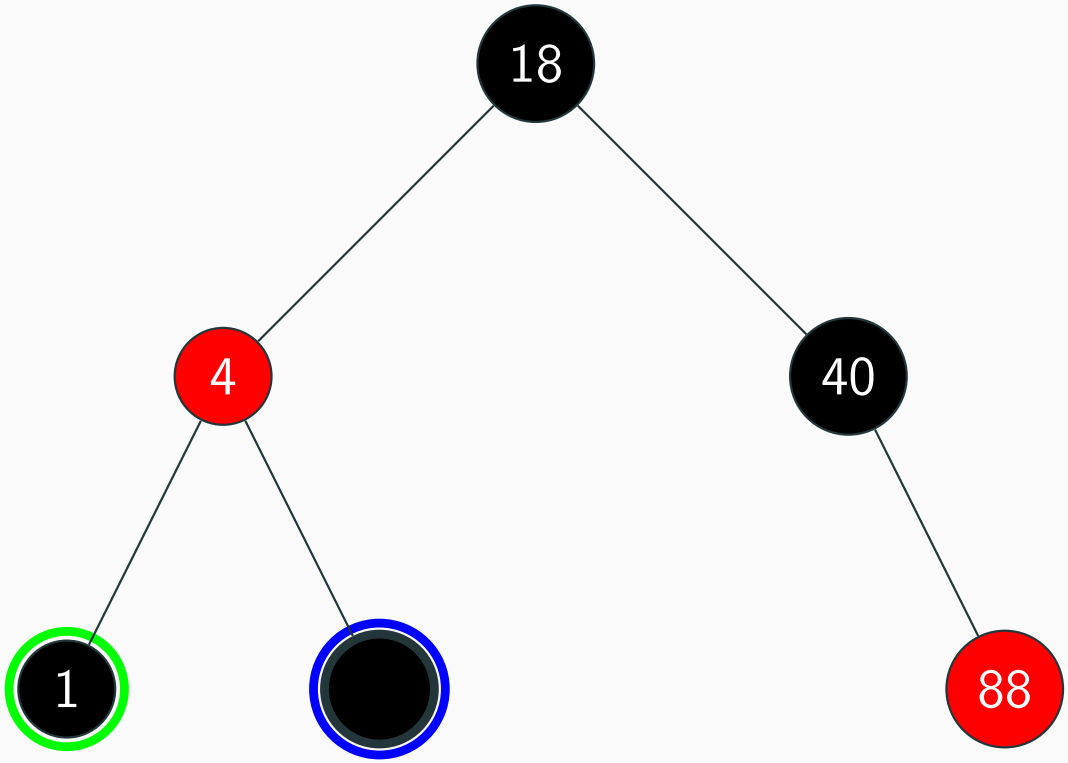
Exemplo de remoção no cenário D

Informação a ser removida: 10



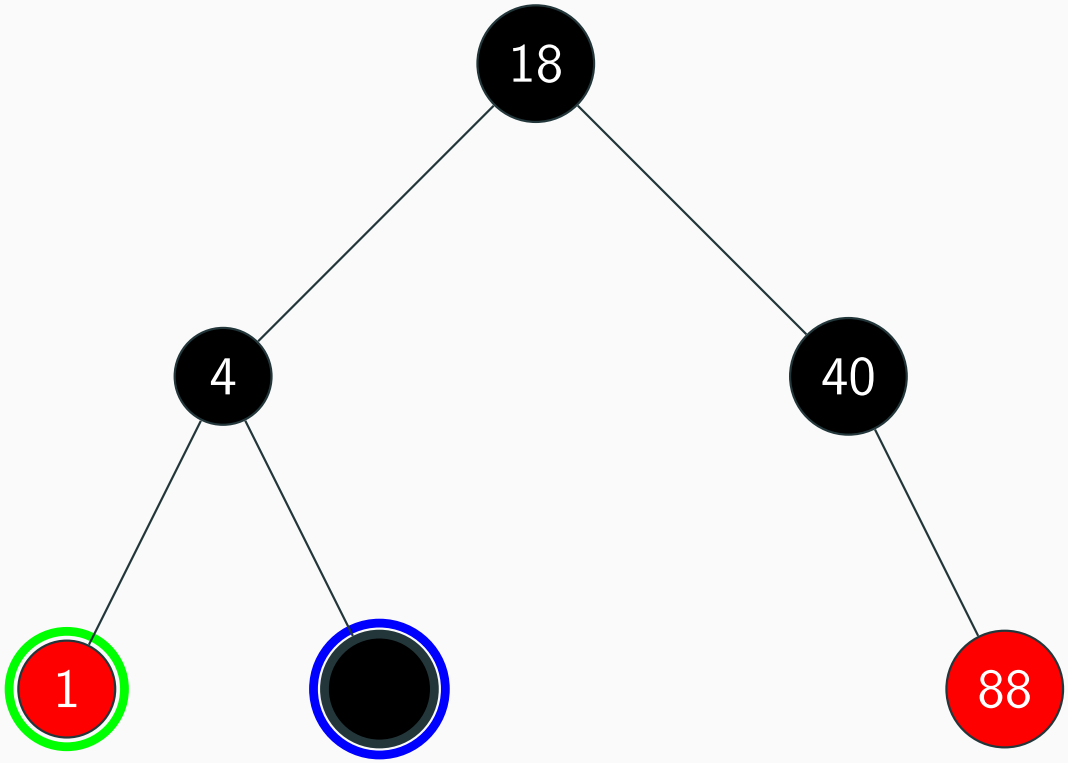
Exemplo de remoção no cenário D

Irmão e sobrinhos pretos, pai vermelho



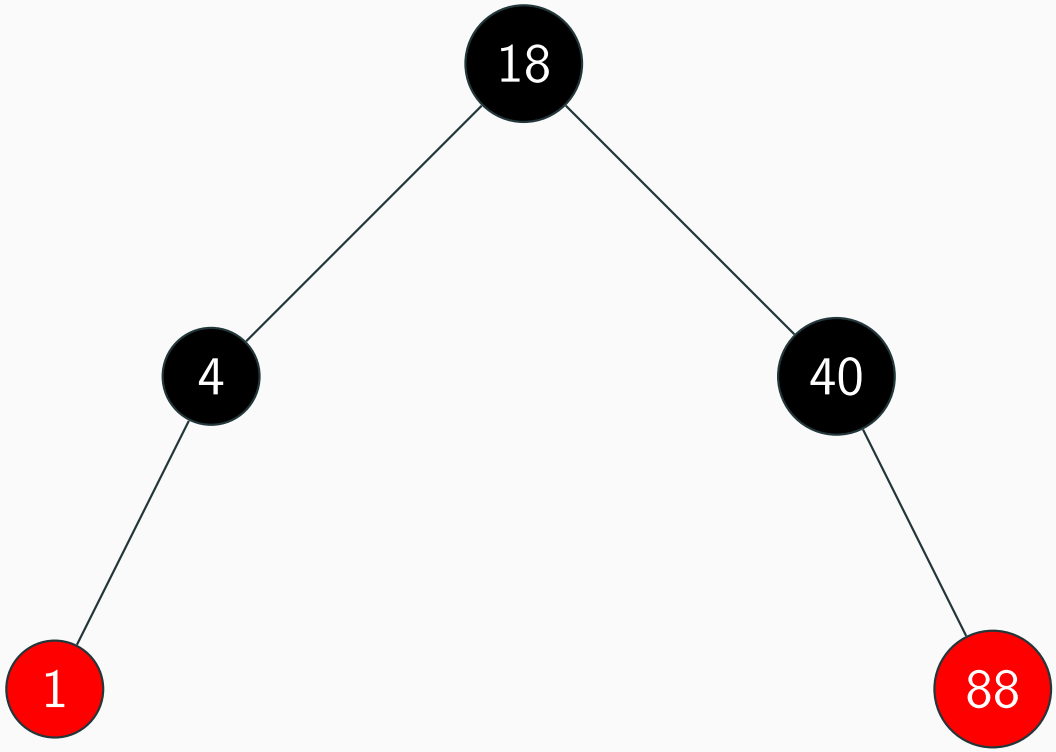
Exemplo de remoção no cenário D

Irmão e sobrinhos pretos, pai vermelho



Exemplo de remoção no cenário D

Irmão e sobrinhos pretos, pai vermelho



Implementação do cenário D

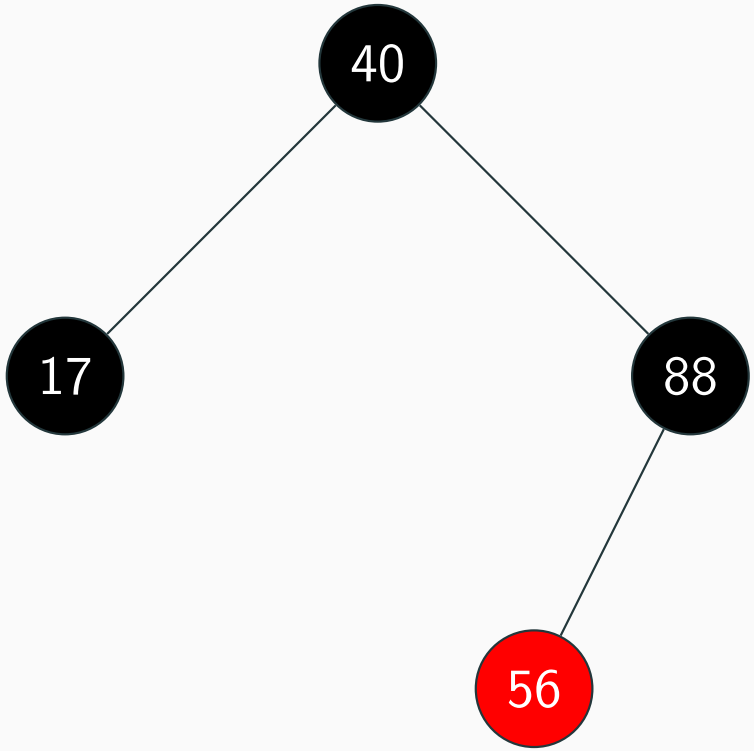
```
254         // N tem irmão e sobrinhos pretos, pai vermelho
255         if (P and P->color == Node::RED and S->color == Node::BLACK
256             and (S->left == nullptr or S->left->color == Node::BLACK)
257             and (S->right == nullptr or S->right->color == Node::BLACK))
258         {
259             S->color = Node::RED;
260             P->color = Node::BLACK;
261             return;
262         }
263
```

Cenário E: Irmão preto, sobrinho à esquerda vermelho

- Se N é o filho à esquerda de P, é necessário rotacionar o sobrinho à direita em torno de S
- Esta transformação torna o sobrinho à esquerda o novo irmão de N
- Antes da rotação, as cores de S e do sobrinho vermelho devem ser trocadas
- Estas modificações não violam a propriedade 5
- Agora, N tem irmão preto com sobrinho à direita vermelho, o que configura o sexto e último cenário
- Se N é o filho à direita de P e o sobrinho vermelho está à direita de S, a situação é simétrica
- Contudo, a rotação deve ser à esquerda em torno de S

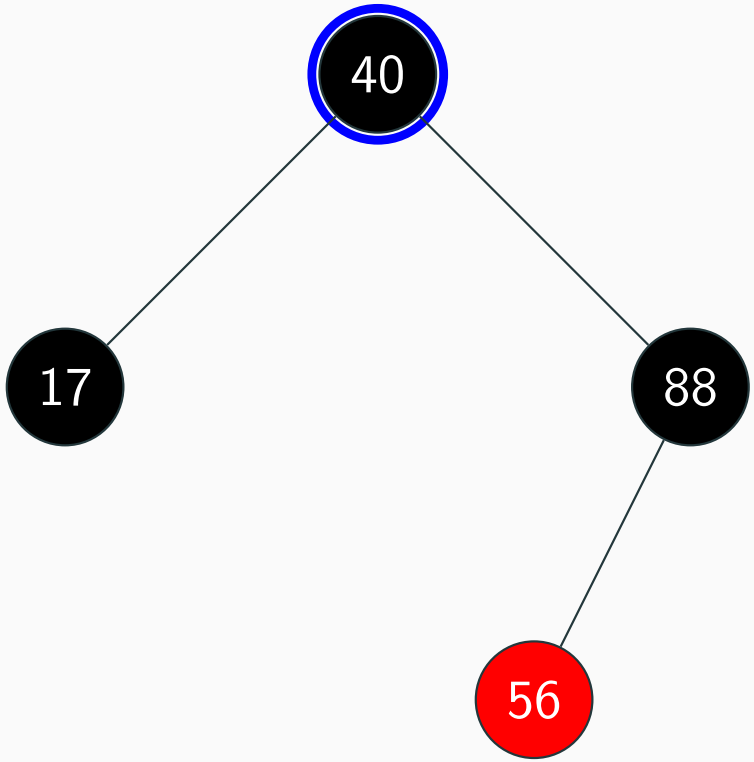
Exemplo de remoção no cenário E

Informação a ser removida: 17



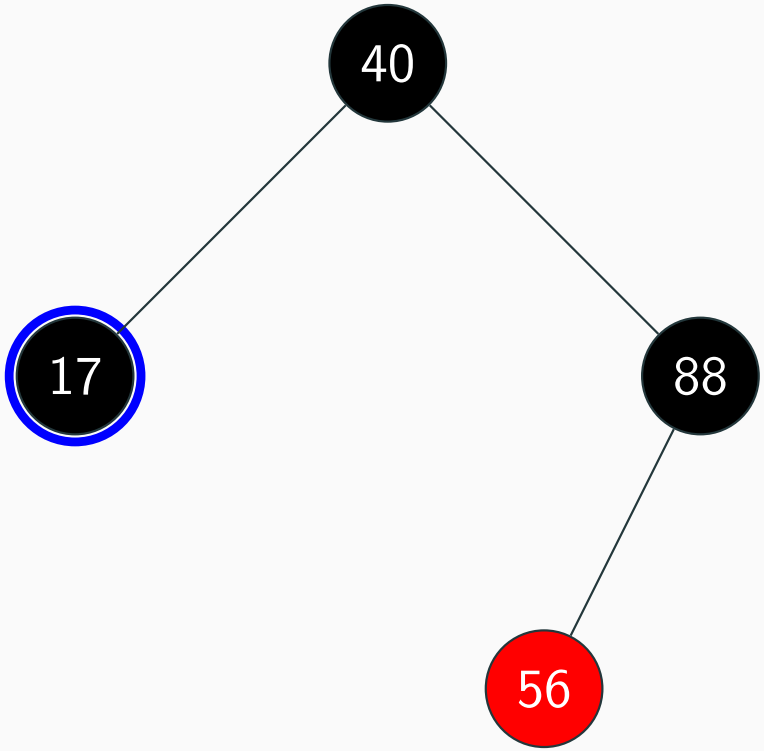
Exemplo de remoção no cenário E

Informação a ser removida: 17



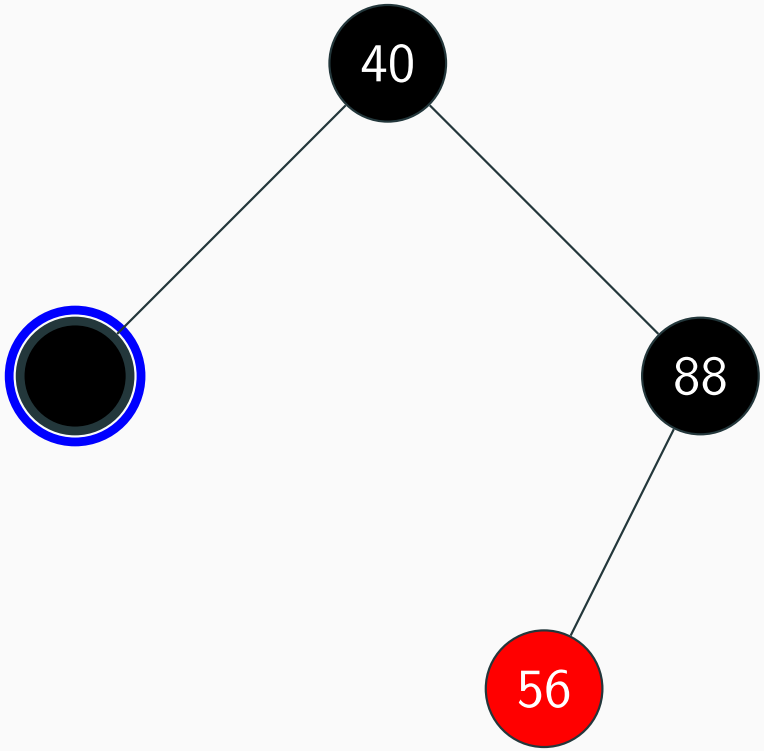
Exemplo de remoção no cenário E

Informação a ser removida: 17



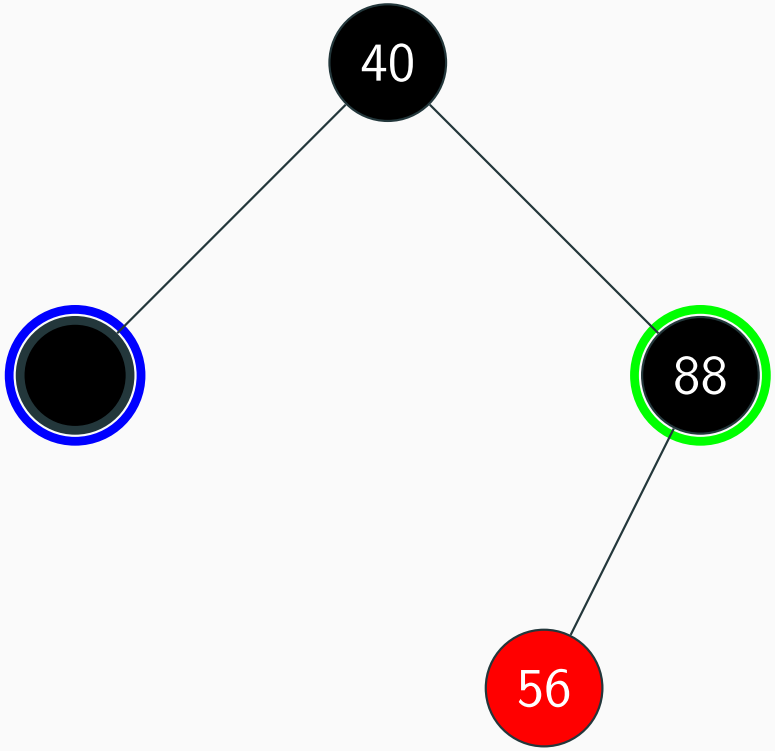
Exemplo de remoção no cenário E

Informação a ser removida: 17



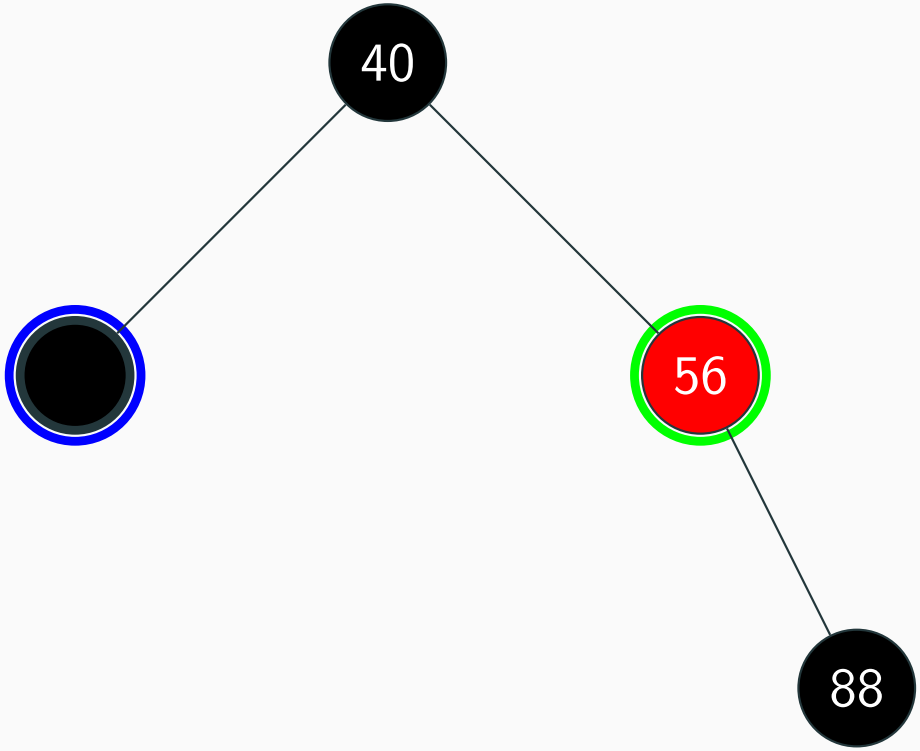
Exemplo de remoção no cenário E

Irmão preto com sobrinho à esquerda vermelho



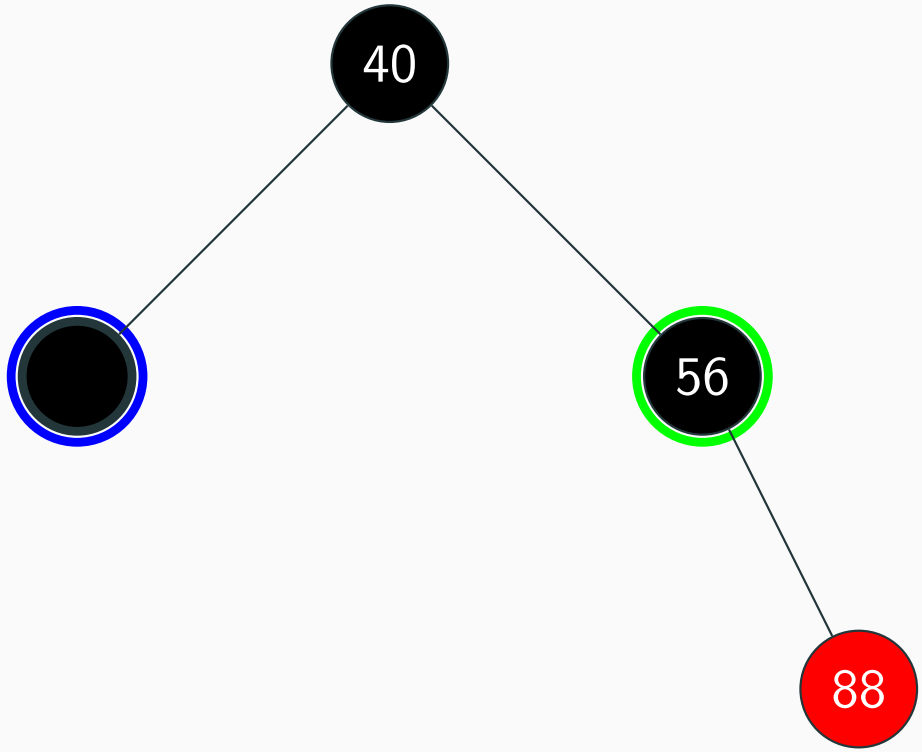
Exemplo de remoção no cenário E

Irmão preto com sobrinho à esquerda vermelho



Exemplo de remoção no cenário E

Cenário F: irmão preto, sobrinho à direita vermelho



Implementação do cenário E

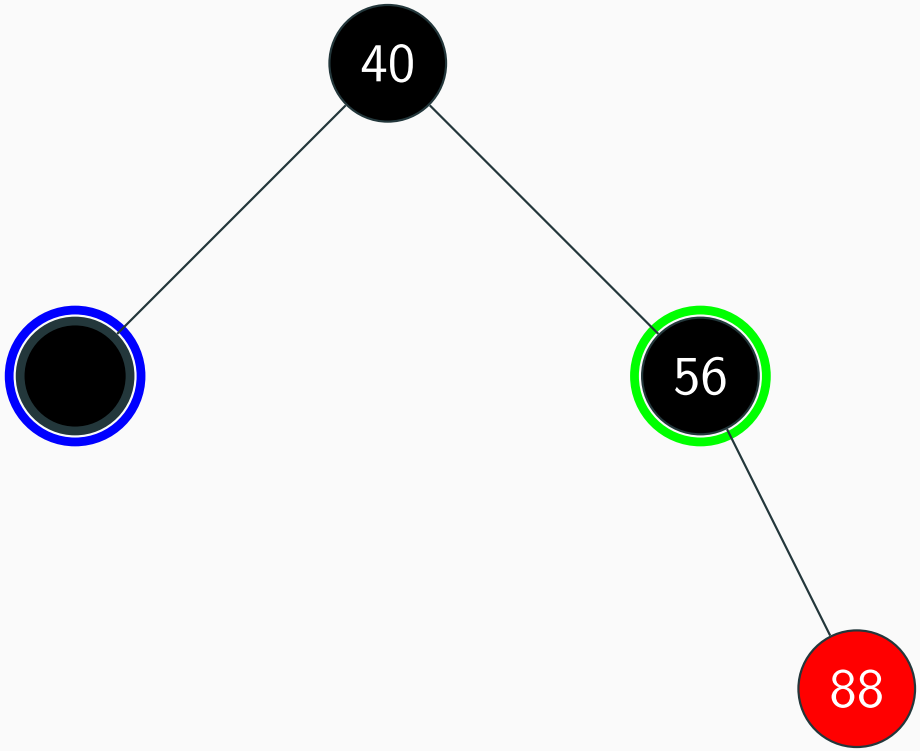
```
264         // N tem irmão preto com sobrinho à esquerda vermelho
265         if (S and S->color == Node::BLACK)
266         {
267             if (P and P->left == N
268                 and (S->left or S->left->color == Node::RED) and
269                 (S->right == nullptr or S->right->color == Node::BLACK))
270             {
271                 S->color = Node::RED;
272                 S->left->color = Node::BLACK;
273                 rotate_right(P, S, S->left);
274                 S = P->right;
275             } else if (P and P->right == N
276                 and (S->left == nullptr or S->left->color == Node::BLACK)
277                 and (S->right or S->right->color == Node::RED))
278             {
279                 S->color = Node::RED;
280                 S->right->color = Node::BLACK;
281                 rotate_left(P, S, S->right);
282                 S = P->left;
283             }
284         }
```

Cenário F: Irmão preto, sobrinho à direita vermelho

- Neste cenaŕio, N é o filho à esquerda de P, S é preto e o sobrinho à direita é vermelho
- O rebalanceamento consiste em rotacionar S à esquerda em torno de P, trocar as cores de P e S e tornar o sobrinho preto
- A subárvore manterá a cor de sua raiz, preservando a propriedade 4
- A mudança de cores não viola a propriedade 5, mas é preciso observar que, após a rotação, N tem um ascentral preto a mais, e N será removido, fazendo com que a contagem de nós pretos original se mantenha
- É possível observar que os caminhos que não passam por N preservam o mesmo número de nós pretos que tinham antes da rotação
- Assim este caso se encerra restaurando as propriedades da árvore *red-black*

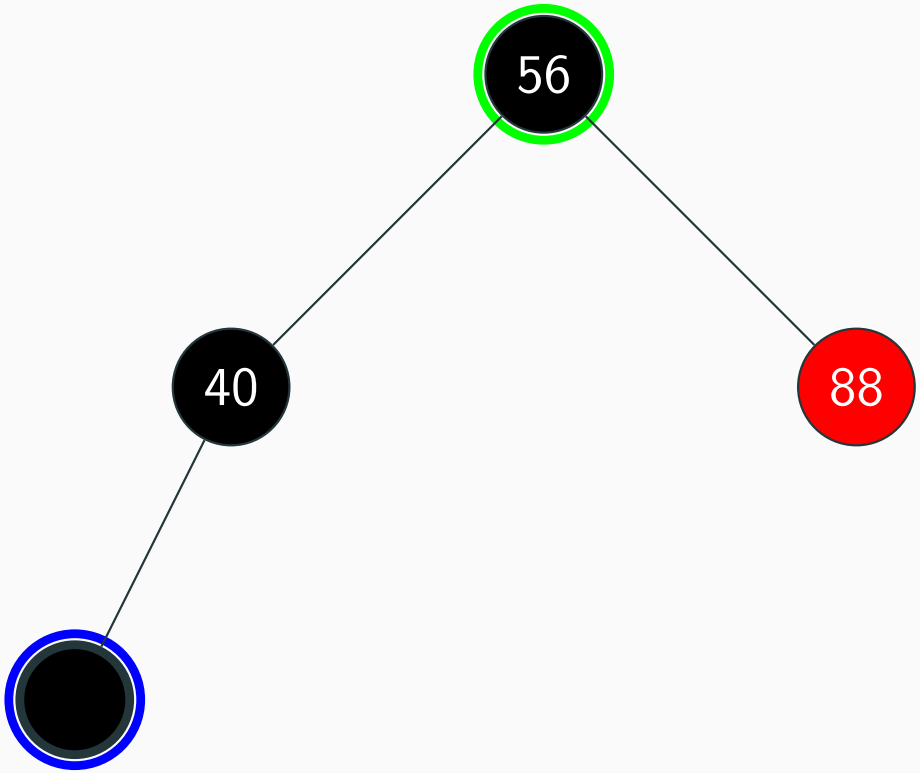
Exemplo de remoção no cenário F

Continuação do cenário anterior



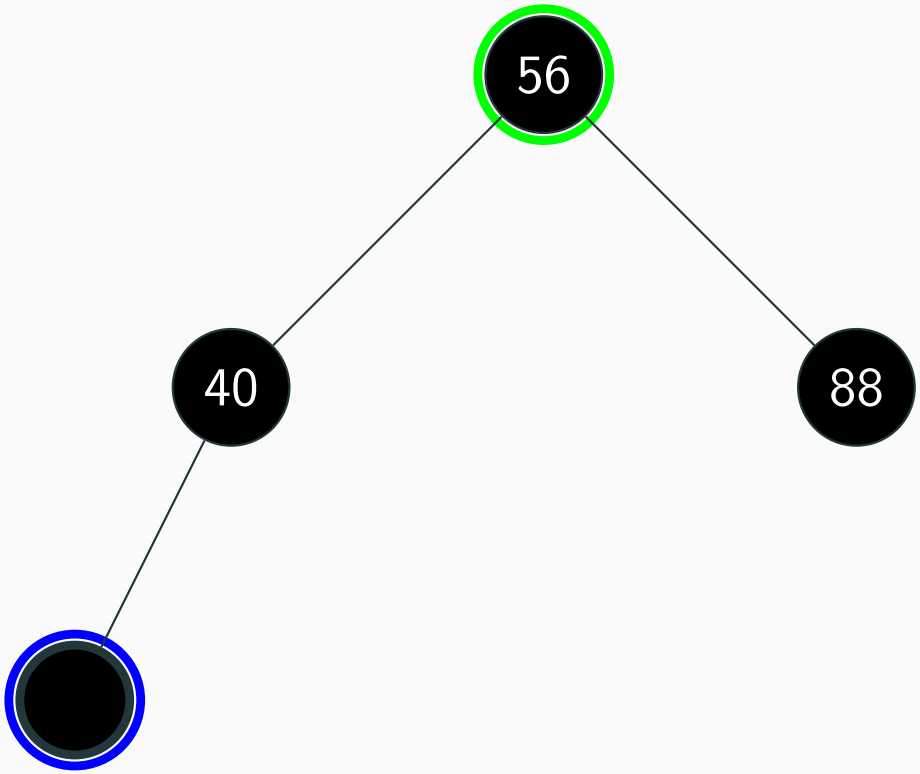
Exemplo de remoção no cenário F

Rotação de S em torno de P e troca de cores



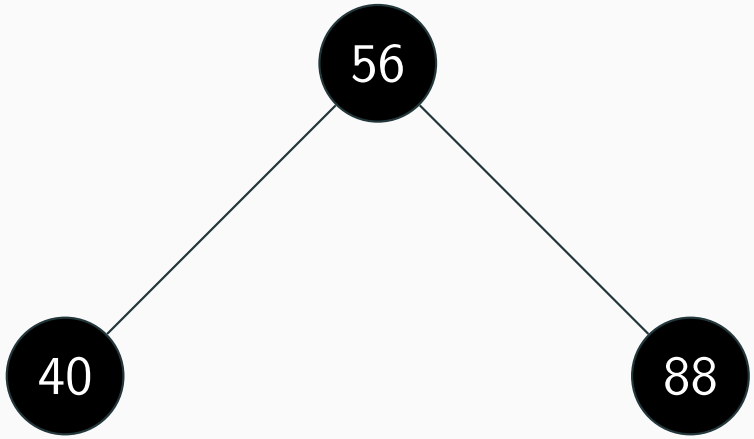
Exemplo de remoção no cenário F

O sobrinho à direita se torna preto



Exemplo de remoção no cenário F

Conclusão da remoção



Implementação do cenário F

```
285
286         // N tem irmão preto com sobrinho à direita vermelho
287         S->color = P->color;
288         P->color = Node::BLACK;
289
290         if (N == P->left)
291         {
292             S->right->color = Node::BLACK;
293             rotate_left(grandparent(P), P, S);
294         } else
295         {
296             S->left->color = Node::BLACK;
297             rotate_right(grandparent(P), P, S);
298         }
299
300         if (parent(S) == nullptr)
301             root = S;
302     }
303
```

Referências

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

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