Estruturas de Dados

Cristina Gomes Fernandes

ABBs rubro-negras

Uma ABB é rubro-**negra** se

1. todo nó é rubro ou **negro**
2. toda folha (NIL) é **negra**
3. se um nó é rubro, então seus dois filhos são **negros**
4. todo caminho de um nó x até uma folha sua descendente tem o mesmo número de nós **negros**.

ABBs rubro-negras

Uma ABB é rubro-**negra** se

1. todo nó é rubro ou **negro**
2. toda folha (NIL) é **negra**
3. se um nó é rubro, então seus dois filhos são **negros**
4. todo caminho de um nó x até uma folha sua descendente tem o mesmo número de nós **negros**.

$rn(x)$: número de nós **negros** no caminho de um filho de x até uma folha descendente de x.

ABBs rubro-negras

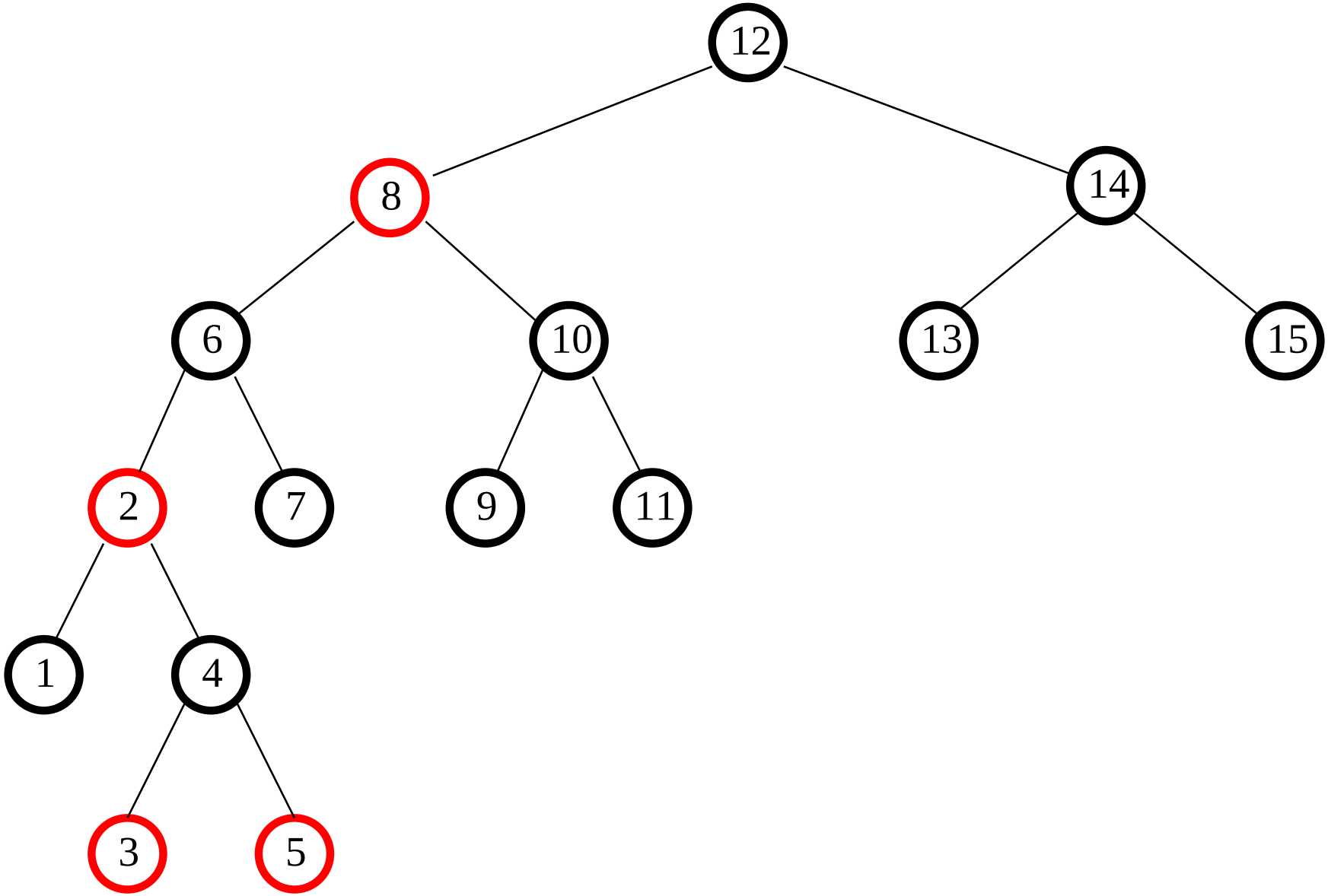
Uma ABB é rubro-**negra** se

1. todo nó é rubro ou **negro**
2. toda folha (NIL) é **negra**
3. se um nó é rubro, então seus dois filhos são **negros**
4. todo caminho de um nó x até uma folha sua descendente tem o mesmo número de nós **negros**.

$rn(x)$: número de nós **negros** no caminho de um filho de x até uma folha descendente de x.

Lema: Uma ABB rubro-**negra** com n nós internos tem altura no máximo $2\lg(n+1)$.

Exemplo de ABB rubro-negra



Rotações

O nó p é tal que $dir(p) \neq \text{NIL}$ e $cor(dir(p)) = \text{RUBRO}$.

ROTACIONEESQ (p)
1 $q \leftarrow dir(p)$
2 $dir(p) \leftarrow esq(q)$
3 $esq(q) \leftarrow p$
4 $cor(q) \leftarrow cor(p)$
5 $cor(p) \leftarrow$ RUBRO
6 **devolva** q

Rotações

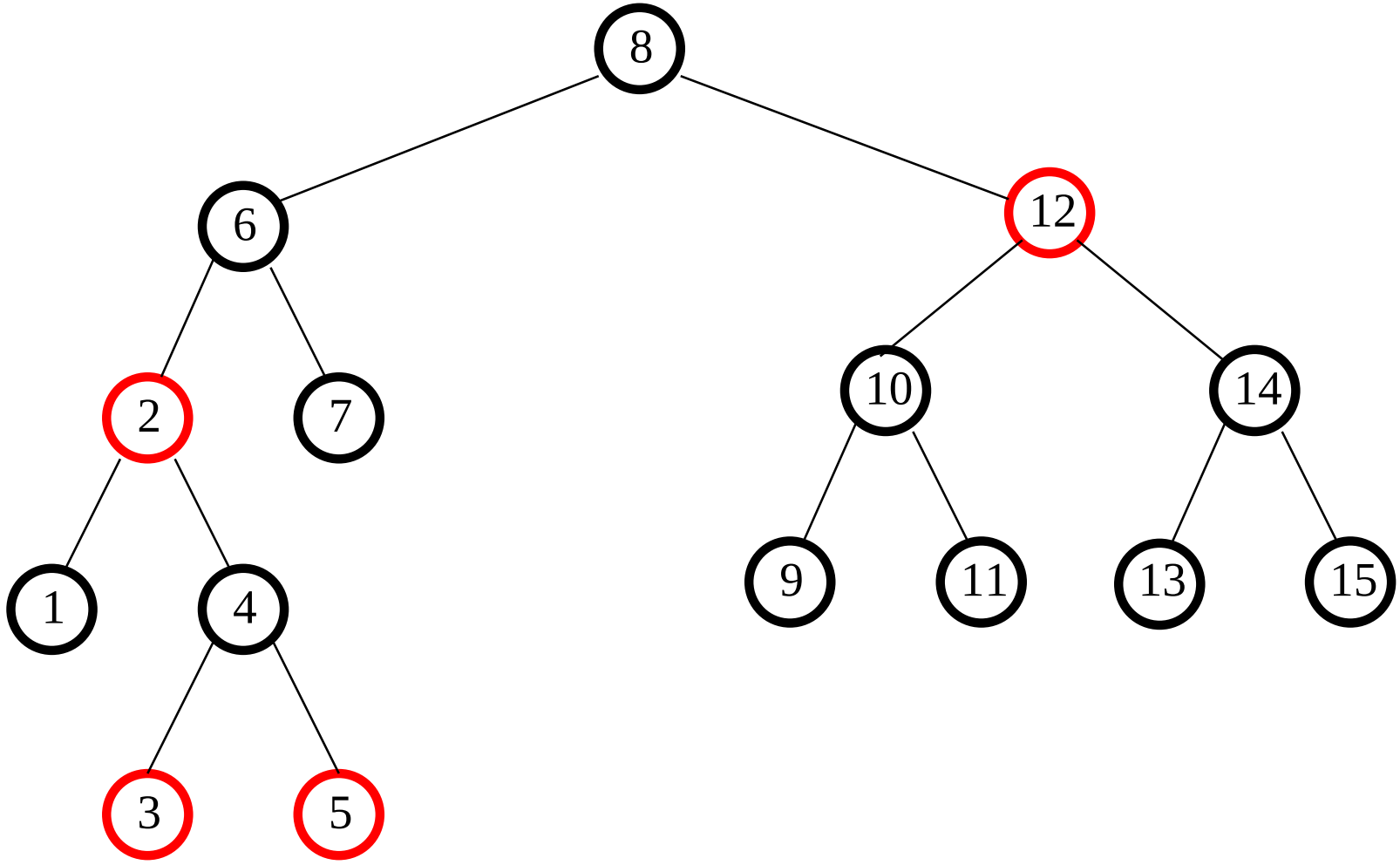
O nó p é tal que $dir(p) \neq \text{NIL}$ e $cor(dir(p)) = \text{RUBRO}$.

ROTACIONEESQ (p)
1 $q \leftarrow dir(p)$
2 $dir(p) \leftarrow esq(q)$
3 $esq(q) \leftarrow p$
4 $cor(q) \leftarrow cor(p)$
5 $cor(p) \leftarrow$ RUBRO
6 **devolva** q

Exercício: Escreva o ROTACIONEDIR.

Exercício: Ajuste estas rotinas para que mantenham o campo pai.

Rotação à direira na raiz



Ajusta cores

O nó p é interno.

TROQUECORES (p)
1 $cor(p) \leftarrow$ OUTRACOR$(cor(p))$
2 $cor(esq(p)) \leftarrow$ OUTRACOR$(cor(esq(p)))$
3 $cor(dir(p)) \leftarrow$ OUTRACOR$(cor(dir(p)))$

Ajusta cores

O nó p é interno.

TROQUECORES (p)
1 $cor(p) \leftarrow$ OUTRACOR$(cor(p))$
2 $cor(esq(p)) \leftarrow$ OUTRACOR$(cor(esq(p)))$
3 $cor(dir(p)) \leftarrow$ OUTRACOR$(cor(dir(p)))$

OUTRACOR (c)
1 **se** $c =$ RUBRO
2 **então devolva** NEGRO
3 **senão devolva** RUBRO

Inserção em ABB rubro-negra

```
RUBRO (p)
1  se p = NIL
2     então devolva FALSO
3     senão se cor(p) = RUBRO
4                 então devolva VERDADE
5                 senão devolva FALSO

NEGRO (p)
1  devolva não RUBRO(p)
```

Inserção em ABB rubro-negra

```
RUBRO (p)
1  se p = NIL
2     então devolva FALSO
3     senão se cor(p) = RUBRO
4                 então devolva VERDADE
5                 senão devolva FALSO

NEGRO (p)
1  devolva não RUBRO(p)

INSIRA (T, x)
 1  T ← INSIRAREC(T, x)
 2  cor(T) ← NEGRO     ▷ a raiz é sempre negra
```

Árvores 2-3

Uma árvore 2-3 é uma extensão de uma ABB:

1. todo nó tem uma ou duas chaves, sendo que um nó interno com k chaves tem $k+1$ filhos;
2. todas as folhas estão no mesmo nível;
3. um nó interno com uma única chave c_1 tem como filhos p_1 e p_2, onde p_1 é a raiz de uma subárvore 2-3 cujas chaves são todas menores que c_1, e p_2 é a raiz de uma subárvore 2-3 cujas chaves são todas maiores que c_1.
4. um nó interno com duas chaves c_1 e c_2 tem como filhos p_1, p_2 e p_3, onde p_1 é a raiz de uma subárvore 2-3 cujas chaves são todas menores que c_1, p_2 é a raiz de uma subárvore 2-3 cujas chaves são todas maiores que c_1 e menores que c_2, e p_3 é a raiz de uma subárvore 2-3 cujas chaves são todas maiores que c_2.

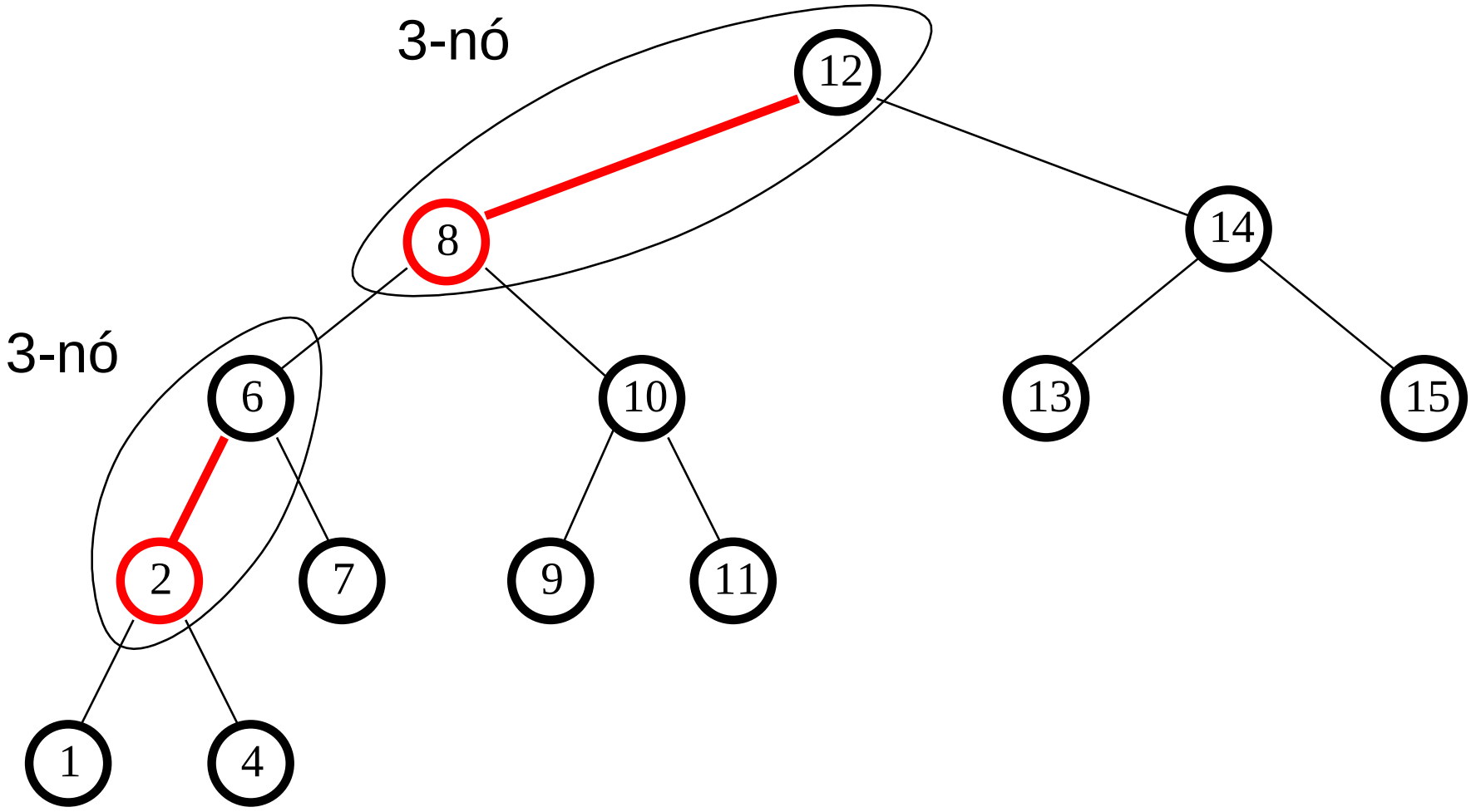
Árvores 2-3 $\times$ rubro-negras

Um nó de uma árvore 2-3 com duas chaves é chamado de 3-nó, e um com uma, é chamado de 2-nó.

Uma árvore 2-3 pode ser representada por uma árvore rubro-negra em que todo nó negro tem no máximo um filho rubro.

A representação é caída para a esquerda (*left leaning*) se, quando um nó negro tem um filho rubro, este é sempre o esquerdo.

Árvores 2-3 × rubro-negras



Inserção em ABB rubro-negra 2-3

Esta é a inserção para árvores 2-3:

```
INSIRAREC (T, x)
 1  se T = NIL
 2     então q ← NOVACÉLULA(x, NIL, NIL, RUBRO)
 3           devolva q
 4  se x < info(T)
 5     então esq(T) ← INSIRAREC(esq(T), x)
 6     senão dir(T) ← INSIRAREC(dir(T), x)
 7  se RUBRO(dir(T)) e NEGRO(esq(T))
 8     então T ← ROTACIONEESQ(T)
 9  se RUBRO(esq(T)) e RUBRO(esq(esq(T)))
10     então T ← ROTACIONEDIR(T)
11  se RUBRO(esq(T)) e RUBRO(dir(T))
12     então TROQUECORES(T)
13  devolva T
```

Inserção em ABB rubro-negra 2-3

Mostre por indução na altura de T que,
no INSIRAREC anterior,
se a árvore T for rubro-negra 2-3 exceto pela raiz,
que pode ser rubra, então a árvore T devolvida satisfaz:

- se a raiz de T era negra, então T é rubro-negra 2-3 exceto pela raiz, que pode ser rubra.
- se a raiz de T era rubra, então T é rubro-negra 2-3 exceto pela raiz, que pode ser rubra, e pode ter o filho esquerdo rubro.

(Fizemos isso na aula. Refaça para ver se você entendeu e se é capaz de reproduzir.)

Conclua que, se a T for uma árvore rubro-negra 2-3, então a árvore devolvida por INSIRA(T, x) é rubro-negra 2-3.

Árvores 2-3-4

Uma árvore 2-3-4 é uma extensão de uma árvore 2-3 em que os nós podem ter uma, duas ou três chaves e, se internos, respectivamente dois, três ou quatro filhos.

Uma árvore rubro-negra nada mais é do que uma implementação por uma ABB de uma árvore 2-3-4.

Um nó negro com dois filhos rubros é um 4-nó.

Mantenha isso em mente quando estudar os algoritmos sobre ABBs rubro-negras, pois isso ajuda. Por exemplo, note a seguir como a inserção em ABB rubro-negra difere da inserção em árvore 2-3 apenas pela mudança de duas linhas de código do lugar.

Inserção em ABB rubro-negra

Esta é a inserção para árvores 2-3-4:

```
INSIRAREC (T, x)
 1  se T = NIL
 2     então q ← NOVACÉLULA(x, NIL, NIL, RUBRO)
 3           devolva q
 4  se RUBRO(esq(T)) e RUBRO(dir(T))
 5     então TROQUECORES(T)
 6  se x < info(T)
 7     então esq(T) ← INSIRAREC(esq(T), x)
 8     senão dir(T) ← INSIRAREC(dir(T), x)
 9  se RUBRO(dir(T)) e NEGRO(esq(T))
10     então T ← ROTACIONEESQ(T)
11  se RUBRO(esq(T)) e RUBRO(esq(esq(T)))
12     então T ← ROTACIONEDIR(T)
13  devolva T
```

Inserção em ABB rubro-negra

Faça uma análise da correção da segunda versão do INSIRAREC, como fizemos para a primeira versão.

Conclua da sua análise que, se a T for uma árvore rubro-negra 2-3-4, então a árvore devolvida por INSIRA(T, x) é rubro-negra 2-3-4.

(Fizemos isso na aula. Refaça.)

Dica: ajuste as duas condições da prova por indução anterior para árvores rubro-negras 2-3-4, fortalecendo a primeira. Refaça a análise de casos cuidadosamente, para ver se os seus ajustes estão corretos. (Em outras palavras, faça a prova por indução.)

Remoção em ABBs

REMOVA (T, q)

```
 1  se esq(q) = NIL ou dir(q) = NIL
 2     então p ← q
 3     senão p ← MÍNIMO(dir(q))      info(q) ← info(p)
 4  se esq(p) = NIL
 5     então f ← dir(p)
 6     senão f ← esq(p)
 7  se f ≠ NIL
 8     então pai(f) ← pai(p)
 9  se pai(p) = NIL
10     então T ← f
11     senão se p = esq(pai(p))
12                 então esq(pai(p)) ← f
13                 senão dir(pai(p)) ← f
14  LIBERACÉLULA(p)
```

Versão recursiva da remoção

Supõe que x está presente na ABB.

```
REMOVA (T, x)
 1  T ← REMOVAREC(T, x)

REMOVAREC (T, x)
 1  se x < info(T)
 2     então esq(T) ← REMOVAREC(esq(T), x)
 3     senão se x > info(T)
 4              então dir(T) ← REMOVAREC(dir(T), x)
 5              senão se dir(T) = NIL
 6                       então devolva esq(T)
 7                     p ← MÍNIMO(dir(T))
 8                     info(T) ← info(p)
 9                     dir(T) ← REMOVAMIN(dir(T))
10  devolva T
```

Versão recursiva da remoção

```
REMOVAMIN (T)
1  se esq(T) = NIL
2     então devolva dir(T)
3     senão esq(T) ← REMOVAMIN(esq(T))
4           devolva T
```

Versão recursiva da remoção

```
RemovaMin (T)
1  se esq(T) = NIL
2     então devolva dir(T)
3     senão esq(T) ← RemovaMin(esq(T))
4           devolva T
```

Exercício: Modifique as rotinas acima para que a remoção funcione mesmo que x não esteja na ABB dada.

Exercício: Ajuste estas rotinas para que mantenham o campo pai atualizado.

Exercício: A remoção iterativa do Remova pode resultar em algo diferente da versão recursiva apresentada. Modifique o RemovaRec para que ela faça a remoção exatamente como a iterativa.

RemovaMin em árvores 2-3

```
RemovaMin (T)
 1  se esq(T) = NIL
 2     então devolva NIL      ▷ dir(T) = NIL aqui
 3  se Negro(esq(T)) e Negro(esq(esq(T)))
 4     então T ← MovaRubroEsq(T)
 5  esq(T) ← RemovaMin(esq(T))
▷ Fixup
 6  se Rubro(dir(T)) e Negro(esq(T))
 7     então T ← RotacioneEsq(T)
 8  se Rubro(esq(T)) e Rubro(dir(T))
 9     então TroqueCores(T)
10  devolva T
```

RemovaMin em árvores 2-3

```
MovaRubroEsq (p)
 1  TroqueCores(p)
 2  se Rubro(esq(dir(p)))
 3     então dir(p) ← RotacioneDir(dir(p))
 4           p ← RotacioneEsq(p)
 5           TroqueCores(p)
 6  devolva p
```

RemovaMin em árvores 2-3

```
MovaRubroEsq (p)
 1  TroqueCores(p)
 2  se Rubro(esq(dir(p)))
 3     então dir(p) ← RotacioneDir(dir(p))
 4           p ← RotacioneEsq(p)
 5           TroqueCores(p)
 6  devolva p
```

Mostre que o RemovaMin funciona quando T é uma árvore rubro-negra 2-3 exceto pela raiz, que pode ser rubra, ou é uma árvore rubro-negra em que o filho esquerdo da raiz rubro.

(Fizemos isso na aula. Refaça.)