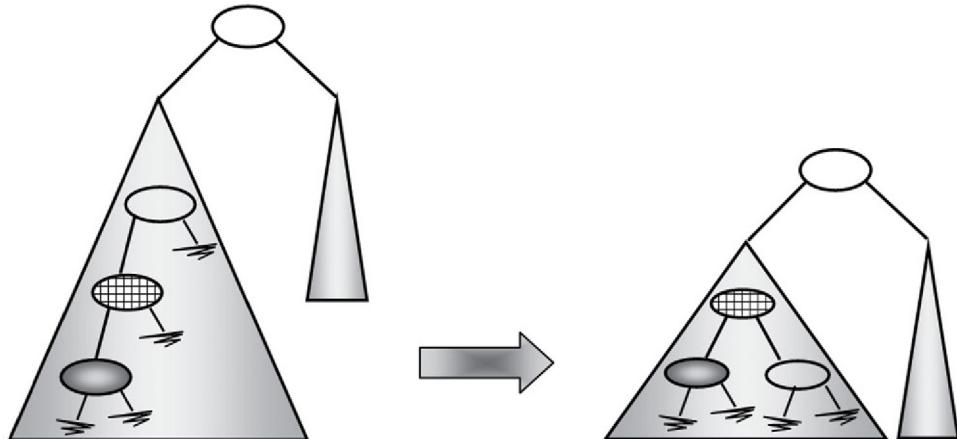


Capítulo 9

Árvores Balanceadas



Seus objetivos neste capítulo

- Entender o conceito de Balanceamento e sua importância para a eficiência das Árvores Binárias de Busca.
- Desenvolver habilidade para elaborar algoritmos sobre Árvores Binárias de Busca Balanceadas e para adaptar a lógica do Balanceamento a novas situações, se necessário.

9.1 Conceito de Balanceamento

O excelente desempenho de uma Árvore Binária de Busca só é atingido se os Nós estiverem uniformemente distribuídos ao longo de toda a Árvore. A Árvore não pode crescer para apenas um dos lados, esquerdo ou direito. É preciso crescer equilibradamente, para ambos os lados: para cada Nós, as alturas das Subárvores Esquerda e Direita precisam ser iguais ou, pelo menos, parecidas.

Os algoritmos para inserção e eliminação de Nós em uma ABB que elaboramos no Capítulo 8 não garantem que haja equilíbrio entre as alturas das Subárvores. Assim, o equilíbrio e o desempenho podem se degradar a cada operação.

Exercício 9.1 Inserir sequência de valores

Aplique o algoritmo Insere, desenvolvido no Capítulo 8, e insira em uma ABB de Raiz R1, inicialmente vazia, os seguintes valores, na sequência A: 50, 20, 40, 12, 90, 75, 120. Em seguida, em uma segunda Árvore de Raiz R2, também inicialmente vazia, insira os mesmos valores, mas na sequência B: 12, 20, 40, 50, 75, 90, 120.

A [Figura 9.1](#) mostra como ficariam as Árvores R1 e R2, dada a utilização do algoritmo Insere que desenvolvemos no Capítulo 8, para a inserção das sequências de valores A e B do Exercício 9.1.

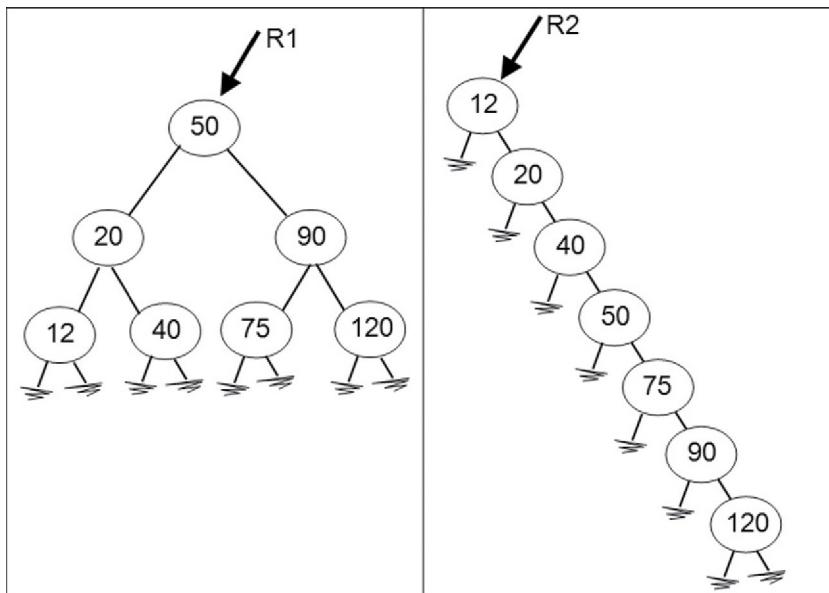


Figura 9.1 Valores inseridos em sequências diferentes.

Foram inseridos os mesmos valores em R1 e em R2, mas em R1 os valores foram inseridos na sequência A e em R2 foram inseridos na sequência B.

As Árvores R1 e R2 da [Figura 9.1](#) ilustram uma deficiência dos algoritmos de inserção e eliminação em uma ABB que elaboramos no Capítulo 8: dependendo da ordem de inserção dos elementos, as Árvores podem ficar equilibradas, como R1 na [Figura 9.1a](#), ou desequilibradas, como R2 na [Figura 9.1b](#). Por Árvore “equilibrada” entendemos uma Árvore em que os tamanhos de suas Subárvore Esquerda e Direita são iguais ou compatíveis.

Refletindo sobre a eficiência das Árvores Binárias de Busca no Capítulo 8, chegamos à conclusão de que, se uma ABB *uniformemente distribuída* tiver 20 níveis, ela terá cerca de um milhão de elementos, e poderemos saber se um elemento X está ou não nessa Árvore visitando, no máximo, 20 Nós. A conclusão traz como ressalva que esse desempenho é obtido em “uma ABB uniformemente distribuída”, como R1, na [Figura 9.1a](#).

A Árvore R2, na [Figura 9.1b](#), não é uniformemente distribuída. R2 é uma árvore desequilibrada ou ainda desbalanceada, pois a Subárvore Direita é muito maior que a Subárvore Esquerda. Logo, devido a esse desequilíbrio, em uma Árvore como R2 não teríamos a mesma eficiência que obtemos em Árvores Balanceadas, como R1. Se uma Árvore desbalanceada como R2 tiver um milhão de elementos, no pior caso teremos que visitar um milhão de Nós para ter certeza de que um valor X está na Árvore.

Visitar, no máximo, 20 Nós ou, no máximo, um milhão de Nós: que diferença de desempenho!

Árvores Binárias de Busca Balanceadas

Uma Árvore Binária de Busca está Balanceada se, para cada Nós, as alturas de suas Subárvores diferem de, no máximo, 1. Se chamarmos a altura da Subárvore Direita de

Definição: Árvore Binária de Busca Balanceada (ABBB)

Uma Árvore Binária de Busca é dita Balanceada se e somente se, para cada Nó da Árvore, as alturas de suas Subárvore diferem de, no máximo, 1.

Figura 9.2 Definição de Árvore Binária de Busca Balanceada.

Árvore R de Hd, e a altura da Subárvore Esquerda de R de He, podemos dizer que R está balanceada se $He = Hd$ ou se $Hd = He + 1$, ou ainda se $Hd = He - 1$. Esses valores precisam ser válidos para cada Nó da Árvore R. Em qualquer outra circunstância, a Árvore não estará Balanceada.

Exercício 9.2 As Árvores estão Balanceadas?

Analice as Árvores da [Figura 9.3](#) e verifique se estão Balanceadas, de acordo com a definição da [Figura 9.2](#).

A Árvore R1 está Balanceada, pois para cada Nó da Árvore as alturas de suas Subárvore diferem de, no máximo, 1. Mas a Árvore R2 não está Balanceada. Você saberia explicar por quê? Você saberia indicar em qual Nó de R2 o desbalanceamento fica evidente?

Ao analisarmos, em R2, o Nó que contém o valor 50, vemos que a altura de sua Subárvore Direita Hd é 2, e a altura da Subárvore Esquerda He é 3; assim, o critério de balanceamento não é quebrado. Hd e Hd diferem, no máximo, em 1 em todos os Nós de R2, exceto o Nó que contém o valor 20. Para esse Nó, Hd = 2 e He = 0. A diferença entre Hd e He para esse Nó difere em 2 e, assim, o critério de balanceamento é quebrado. Se o critério é quebrado em um dos Nós da Árvore, então a Árvore como um todo não está balanceada.

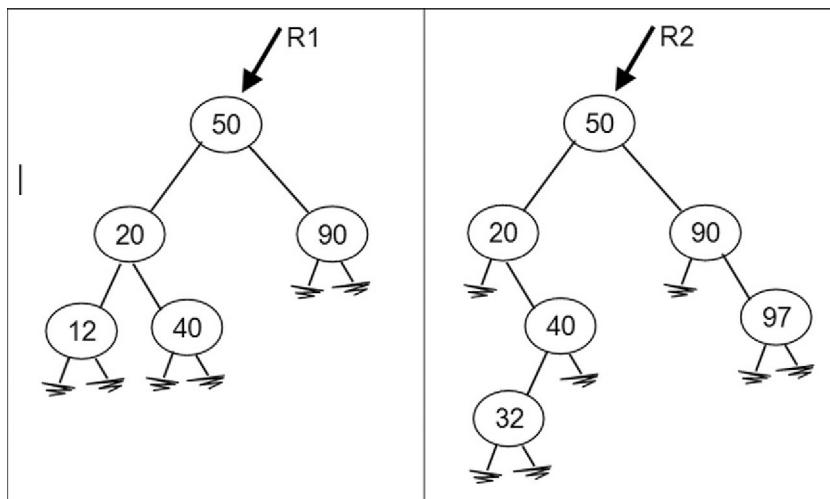


Figura 9.3 Estão Balanceadas?

Como manter uma Árvore Balanceada?

Se utilizarmos os algoritmos para inserção e eliminação que elaboramos no Capítulo 8, a Árvore Binária de Busca perderá gradativamente seu equilíbrio e seu desempenho. Em algum momento poderemos promover uma reorganização geral de todos os valores que fazem parte da Árvore, para torná-la novamente Balanceada. O bom desempenho seria mantido apenas por algum tempo: com a execução de operações para inserir e eliminar valores, o desempenho iria diminuir gradativamente, até que uma nova reorganização geral dos elementos seja executada.

Uma estratégia melhor seria não permitir o desbalanceamento da Árvore, em nenhum momento. Dessa forma, o bom desempenho seria garantido durante todo o tempo, e não apenas logo após uma reorganização geral. Para manter a Árvore constantemente Balanceada, precisamos alterar os algoritmos de inserção e eliminação. Os novos algoritmos deverão: monitorar o Balanceamento da Árvore e, quando necessário, desencadear ações de Rebalanceamento.

Estratégia para manter uma ABB Balanceada

Alterar os algoritmos de inserção e eliminação de modo a:

- a) Monitorar o Balanceamento da Árvore.
- b) Desencadear ações de rebalanceamento, sempre que necessário.

Figura 9.4 Estratégia para manter uma ABB Balanceada.

Essa estratégia de manter a Árvore sempre balanceada foi proposta por [Adelson-Velskii e Landis \(1962\)](#), dando origem ao nome “Árvore AVL” (AVL são as iniciais de Adelson-Velskii e Landis). Posteriormente, as Árvores AVL foram tratadas em livros como os de [Langsam, Augenstein e Tenembaum \(1996\)](#), e [Drozdek \(2002\)](#), e em materiais didáticos como os de Devadas et al. (2009), Leser (2011), Buricea (links 1, 2 e 3) e Camargo.

Para monitorar o Balanceamento, vamos manter, para cada Nô, o Fator de Balanceamento (nome adotado por [Drozdek, 2002](#)), ou Bal, definido como o valor da Altura da Subárvore Direita (Hd) menos o valor da Altura da Subárvore Esquerda (He). Pela definição de Árvores Balanceadas da [Figura 9.2](#), o Fator de Balanceamento poderá assumir os valores: -1, 0 e 1. Com outros valores, a Árvore estará desbalanceada.

Fator de Balanceamento: $\text{Bal} = \text{Hd} - \text{He}$

Altura da Subárvore Direita menos a altura da Subárvore Esquerda.

Figura 9.5 Definição: Fator de Balanceamento.

Exercício 9.3 Calcular o Fator de Balanceamento

Calcule o Fator de Balanceamento para cada Nô das Árvores R1 e R2 da [Figura 9.3](#).

9.2 Inserir elementos em uma ABB Balanceada

Seja uma Árvore R com Subárvore Esquerda SE e Subárvore Direita SD, com alturas He e Hd, respectivamente. Se um novo elemento é inserido em SE, causando aumento na altura He, três casos podem ocorrer, conforme ilustra a [Figura 9.6](#).

Se, antes da inserção, He era menor que Hd em 1, o Fator de Balanceamento de R era +1. Após a inserção, com o aumento da altura He, o Fator de Balanceamento passará a

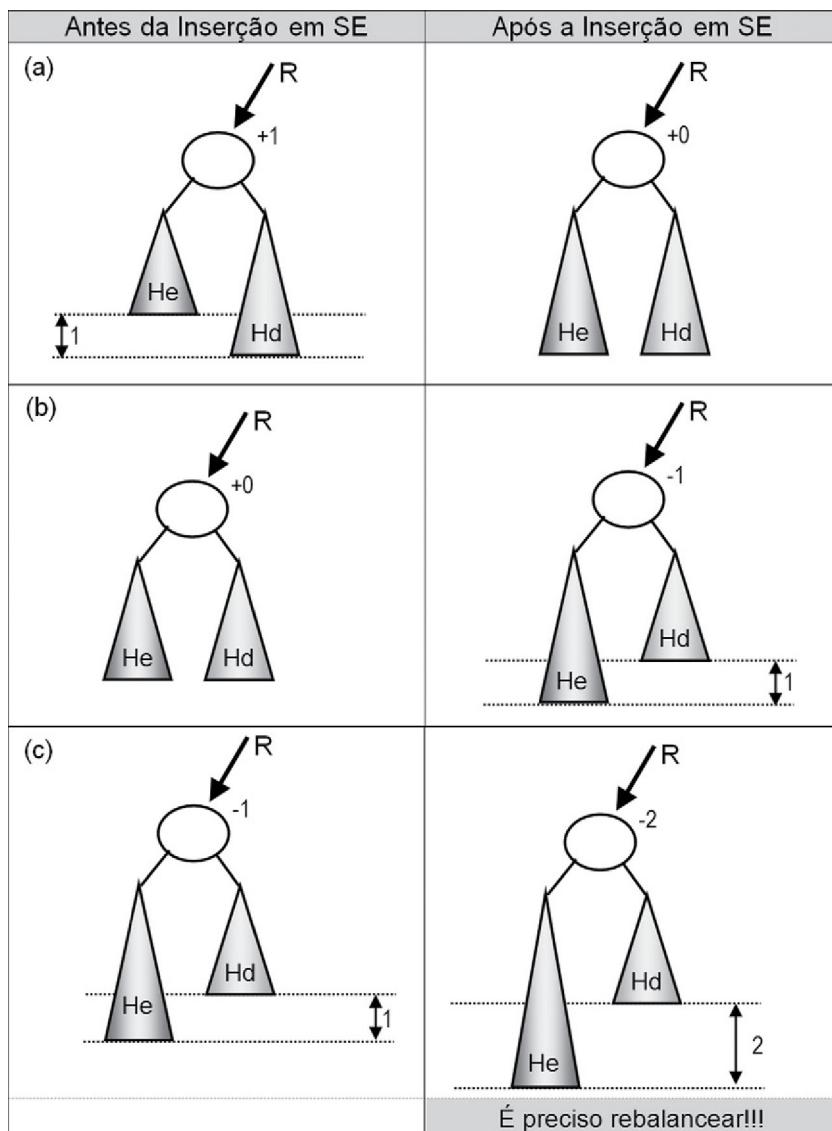


Figura 9.6 Monitorando o Balanceamento na inserção, com aumento da Altura He.

ser 0, pois He passará a ser igual a Hd. O critério de balanceamento não será quebrado nesse caso ([Figura 9.6a](#)).

Se, antes da inserção, as alturas das Subárvores Esquerda e Direita eram iguais, o Fator de Balanceamento era 0. Após a inserção, com aumento da altura He, o Fator de Balanceamento de R passará a ser -1, pois Hd será menor que He em 1, mas o critério de balanceamento não será quebrado ([Figura 9.6b](#)).

Se, antes da inserção, He era maior que Hd em 1, o balanceamento era -1. Após a inserção, com aumento da altura He, He será maior que Hd em 2. O critério de balanceamento, nesse caso, é violado, ou seja, a inserção de um elemento está causando desbalanceamento, e a Árvore precisará ser ajustada ([Figura 9.6c](#)).

Observe ainda, nas situações da [Figura 9.6](#), a altura total da Árvore. Na [Figura 9.6a](#), antes da inserção de um novo valor em He, a altura da Árvore R era Hd + 1 (altura da Subárvore Direita mais um nível, em função do Nô apontado por R). Após a inserção do novo valor, a altura total da Árvore continua sendo Hd + 1. Na [Figura 9.6b](#), na situação inicial a altura é Hd + 1 e na situação final a altura é Hd + 2. Na [Figura 9.6c](#), a altura inicial era Hd + 2 e a altura final é Hd + 3. Nos casos 9.6b e 9.6c, a inserção de um novo valor em SE fez crescer He e também a Árvore como um todo. Na situação da [Figura 9.6a](#), a inserção de um novo valor em SE fez crescer He, mas a altura da Árvore como um todo permaneceu a mesma.

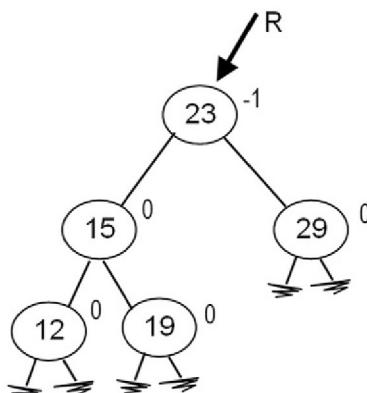


Figura 9.7 Monitorando o Balanceamento na Inserção, com aumento da Altura He.

Exercício 9.4 Causaria desbalanceamento?

Considerando como situação inicial a Árvore da [Figura 9.7](#), verifique:

- A inserção do valor 25 causaria desbalanceamento?
- A inserção do valor 40 causaria desbalanceamento? Considere a Árvore exatamente como mostra a [Figura 9.7](#), sem a chave 25.
- A inserção do valor 9 causaria desbalanceamento? Considere a Árvore exatamente como mostra a [Figura 9.7](#) (sem 25 e 40).
- A inserção do valor 13 causaria desbalanceamento? Considere a Árvore como mostra a [Figura 9.7](#) (sem 9, 25 e 40).

- A inserção do valor 17 causaria desbalanceamento? Considere a Árvore como mostra a [Figura 9.7](#) (sem 9, 13, 25 e 40).
- A inserção do valor 21 causaria desbalanceamento? Considere a Árvore como mostra a [Figura 9.7](#) (sem 9, 13, 17, 25 e 40).

A inserção dos valores 25 ou 40 não causaria desbalanceamento, pois eles seriam inseridos logo abaixo do Nό que contém o valor 29. Na verdade, a Árvore ficaria até mais equilibrada com a inserção de 25 ou 40. Já a inserção dos valores 9, 13, 17 ou 21 causaria desbalanceamento, pois eles seriam inseridos logo abaixo dos Nόs de valor 12 ou 19, aumentando o tamanho da Subárvore Esquerda de R. Como a Subárvore Esquerda de R já era maior do que a Subárvore Direita, com a inserção de 9, 13, 17 ou 21 o critério de balanceamento seria violado.

A inserção dos valores 9, 13, 17 ou 21 exemplifica a situação genérica da [Figura 9.6c](#), em que a Subárvore Esquerda já é maior, e cresce ainda mais, causando desbalanceamento. Conforme nossa estratégia para manter a Árvore balanceada ([Figura 9.4](#)), ao detectar que a inserção de um novo valor causou desbalanceamento, precisamos ajustar a Árvore para que volte a ser balanceada. Chamamos a esse processo de rebalanceamento.

Exercício 9.5 Como rebalancear?

Na [Figura 9.8](#), o valor 12 acabou de ser inserido e causou desbalanceamento. Como essa Árvore pode ser rebalanceada? Não se esqueça de que é preciso respeitar o critério de balanceamento (a diferença das alturas pode ser de, no máximo, 1) e também o critério que define uma Árvore Binária de Busca (valores da Subárvore Esquerda devem ser menores, valores da Subárvore Direita devem ser maiores, para cada Nό da Árvore).

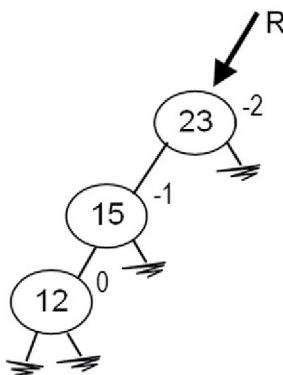


Figura 9.8 Inserção do valor 12 causou desbalanceamento.

Casos de Rebalanceamento: caso 1 — Rotação Simples EE

O Exercício 9.5 é um exemplo do caso de rebalanceamento Rotação Simples EE ou, ainda, Rotação Simples do tipo Esquerda-Esquerda. Essa forma de denominar os casos de rebalanceamento — adotada neste livro — é compatível com a adotada no material didático de Camargo.

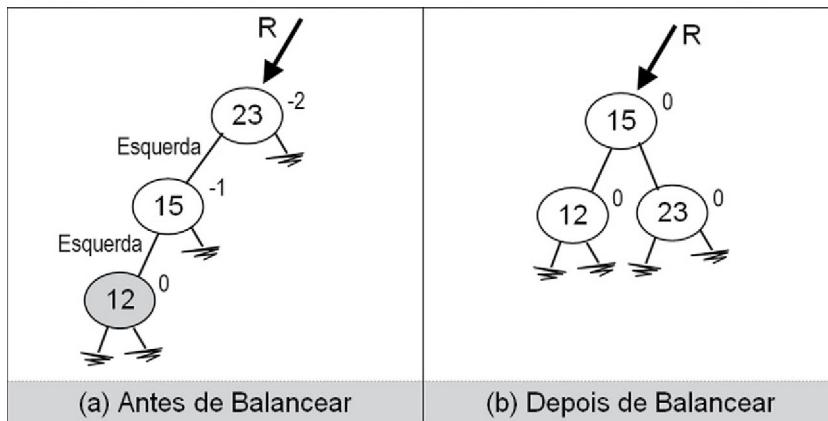


Figura 9.9 Insere em ABBB: caso 1 — Rotação Simples EE. Exemplo com apenas três valores.

A [Figura 9.9](#) mostra a situação inicial ([Figura 9.9a](#)): o Nó que contém o valor 12 acabou de ser inserido e causou o desbalanceamento. O nome Rotação Simples Esquerda-Esquerda reflete a seguinte situação: a partir do Nó em que foi detectado o desbalanceamento (Nó apontado por R), em direção ao Nó que causou o desbalanceamento, temos que seguir para a Subárvore Esquerda e depois outra vez para a Subárvore Esquerda.

A [Figura 9.9b](#) mostra a situação final da Árvore, após ter sido rebalanceada. Você consegue achar outra configuração para uma Árvore, com esses mesmos três valores, que respeite o critério de uma ABB e também o critério de balanceamento? Tente achar outra configuração, que não a da [Figura 9.9b](#).

Não há outra configuração que atenda a ambos os critérios. A situação da [Figura 9.9b](#) é a única solução. A lógica de balanceamento nesse exemplo com apenas três valores é bastante intuitiva. Lembre-se desse exemplo com apenas três valores se estiver em dúvida quanto ao caso 1 — Rotação Simples EE.

Exemplo com mais de três valores

Temos um segundo exemplo da Rotação simples Esquerda-Esquerda na [Figura 9.10](#). O valor 9 acabou de ser inserido, causando desbalanceamento. Na [Figura 9.10a](#), a altura da Subárvore Esquerda de R é 3 (três níveis) e a altura da Subárvore Direita de R é 1 (um nível). A diferença entre a altura das Subárvores é 2, o que viola o critério de balanceamento.

Para rebalancear a Árvore, o primeiro passo é identificar os três Nós principais. O primeiro desses Nós é aquele em que foi detectado o desbalanceamento, ou seja, o Nó apontado por R. Nesse Nó apontado por R, a diferença entre as alturas de suas subárvores é 2, o que viola o critério de balanceamento.

Para identificar os outros dois Nós principais para o rebalanceamento, caminhamos em direção ao Nó que causou o desbalanceamento (ou seja, o Nó que acabou de ser inserido) e encontramos os Nós com valores 15 e 12 ([Figura 9.10b](#)). Como caminhamos para a esquerda e outra vez para a esquerda, fica caracterizado o caso 1 — Rotação Simples EE.

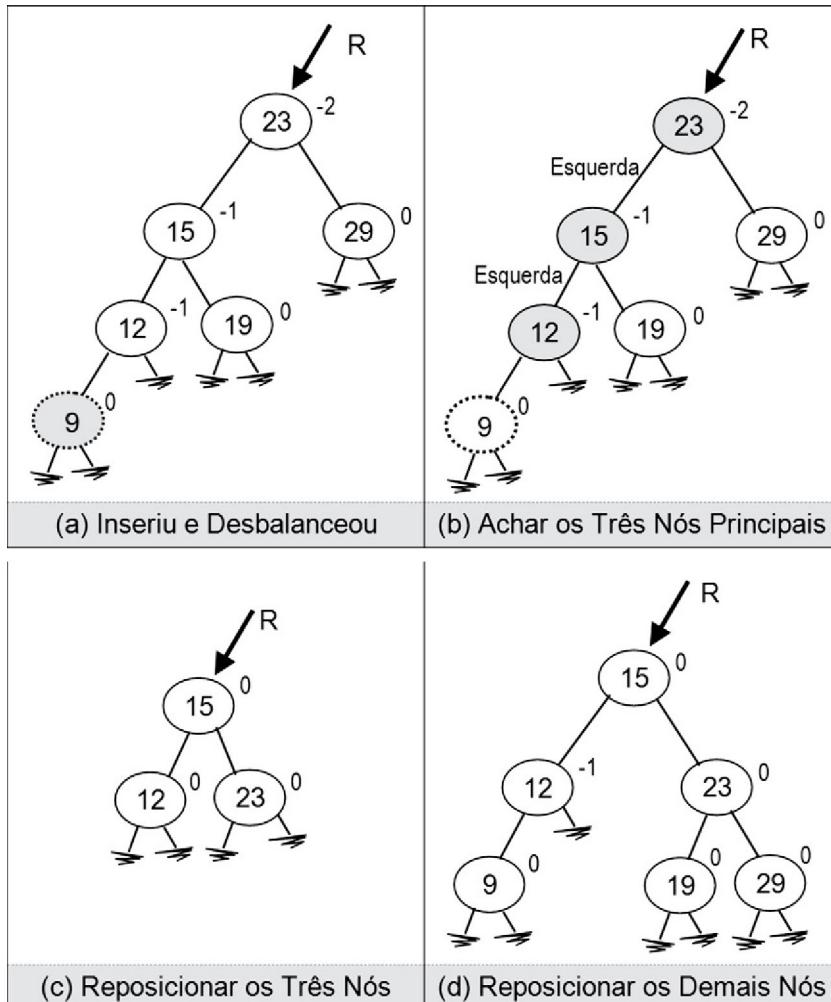


Figura 9.10 Insere em ABBB: caso 1 — Rotação Simples EE.

Note que os valores dos três Nós principais são exatamente os mesmos valores e nas mesmas posições do Exercício 9.5 que apresentava uma Árvore com apenas esses três valores. A lógica do rebalanceamento desses três Nós principais será a mesma lógica aplicada no Exercício 9.5, resultando na situação da [Figura 9.10c](#). Após identificar os três Nós principais, pense em uma Árvore com apenas esses três valores. A solução será bastante intuitiva.

Para posicionar os demais valores da Árvore — 9, 19 e 29 — é preciso seguir o critério que define uma Arvore Binária de Busca: valores menores vão para a Subárvore Esquerda, valores maiores vão para a Subárvore Direita. Considerando que os valores 12, 15 e 23 já estão posicionados segundo mostra a [Figura 9.10c](#), em qual lugar deve ser posicionada a chave 9, de modo a respeitar o critério que define uma Árvore Binária de Busca? A única posição possível para a chave 9 é à esquerda da chave 12, pois 9 é menor do que 12. Se o 9 ficasse à direita do 12, quebraria o critério. Se o 9 ficasse abaixo do 23 (seja à esquerda ou à direita), estaria à direita do 15, o que também quebraria o critério.

Seguindo o mesmo raciocínio que utilizamos para posicionar a chave 9, qual o único lugar no qual podemos posicionar os valores 19 e 29, sem quebrar a definição de Árvore Binária de Busca? A única solução possível é posicionar o 19 à esquerda do 23, e o 29 à direita do 23, como mostra a [Figura 9.10d](#). Não há outra solução que não quebre o critério que define uma ABB.

Para rebalancear uma Árvore manualmente

- Passo 1: identificar os três Nós principais.** O primeiro desses três é o Nó em que foi detectado o desbalanceamento (a diferença de altura das subárvore é maior que 1). Selecione os outros dois Nós caminhando em direção ao Nó que causou o desbalanceamento (o Nó que foi inserido).
- Passo 2: reposicionar os três Nós principais.** Considere uma Árvore com apenas esses três nós e os reposicione na única configuração possível que respeite ambos os critérios: ABB e Balanceamento.
- Passo 3: reposicione os demais valores respeitando o critério que define uma ABB.** Posicione os demais valores abaixo dos três Nós principais. Só há um local possível para cada valor ou Subárvore a ser reposicionada.
- Passo 4: atualize o Fator de Balanceamento de cada Nó.** Conforme definido na Figura 9.5, $\text{Bal} = \text{Hd} - \text{He}$, ou seja, o Fator de Balanceamento de um Nó R é a altura da Subárvore Direita de R menos a altura da Subárvore Esquerda de R.

Figura 9.11 Passos para rebalancear uma Árvore.

Exercício 9.6 Rebalanceamento manual — EE

Nas situações da [Figura 9.12](#), um novo valor acabou de ser inserido e causou desbalanceamento. O valor inserido foi posicionado em um Nó com fundo cinza. Em cada exemplo, faça o rebalanceamento pelo caso 1: Rotação Simples EE. Desenhe a Árvore resultante respeitando o critério que define uma ABB e o critério de Balanceamento. Siga o roteiro da [Figura 9.11](#).

Generalização do caso 1: Rotação Simples EE — Insere

A [Figura 9.13](#) apresenta um diagrama genérico do rebalanceamento pelo caso 1: Rotação Simples EE do algoritmo que insere um novo valor na Árvore. Esse estilo de diagrama é uma adaptação do estilado adotado por [Knuth \(1998, p. 461\)](#) e também por Camargo (p. 3-4).

Note na [Figura 9.13a](#) que um novo valor X acabou de ser inserido, causando desbalanceamento. O Fator de Balanceamento do Nó que contém o valor 'A' antes da inserção de X era zero, pois a altura das Subárvore Esquerda e Direita era igual. Após a inserção de X, o Fator de Balanceamento do Nó que contém 'A' passou a ser -1, pois He passou a ser maior que Hd em 1. Mesmo com a inserção de X, o Nó que contém 'A' continua balanceado.

O Fator de Balanceamento do Nó que contém o valor 'B' era -1 antes da inserção de X. Com a inserção de X, passou a ser -2. Ou seja, a Subárvore Esquerda do Nó que

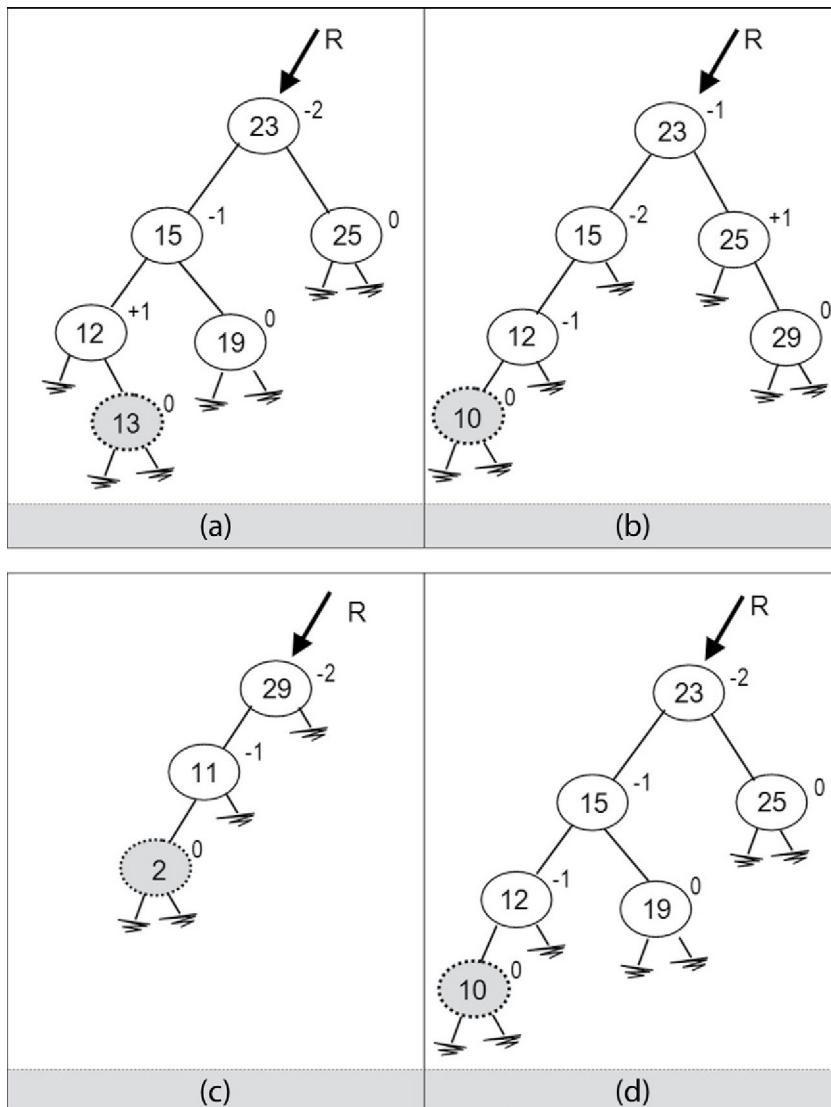


Figura 9.12 Exemplos do caso 1 — Rotação Simples EE.

contém 'B' já era maior que a Subárvore Direita; com a inserção de X em S1, passou a ser ainda maior, violando o critério de平衡amento. Será preciso rebalancear a Árvore.

O rebalanceamento no diagrama da [Figura 9.13](#) generaliza o rebalanceamento dos exemplos numéricos das [Figuras 9.9 e 9.10](#), e também dos quatro casos do Exercício 9.6. Procure identificar a equivalência entre os exemplos numéricos e o diagrama genérico da [Figura 9.13](#).

A [Figura 9.14](#) mostra um trecho de algoritmo que implementa o rebalanceamento pelo caso 1: Rotação Simples EE. É um trecho do algoritmo que insere um valor X em uma Árvore Binária de Busca Balanceada (ABBB).

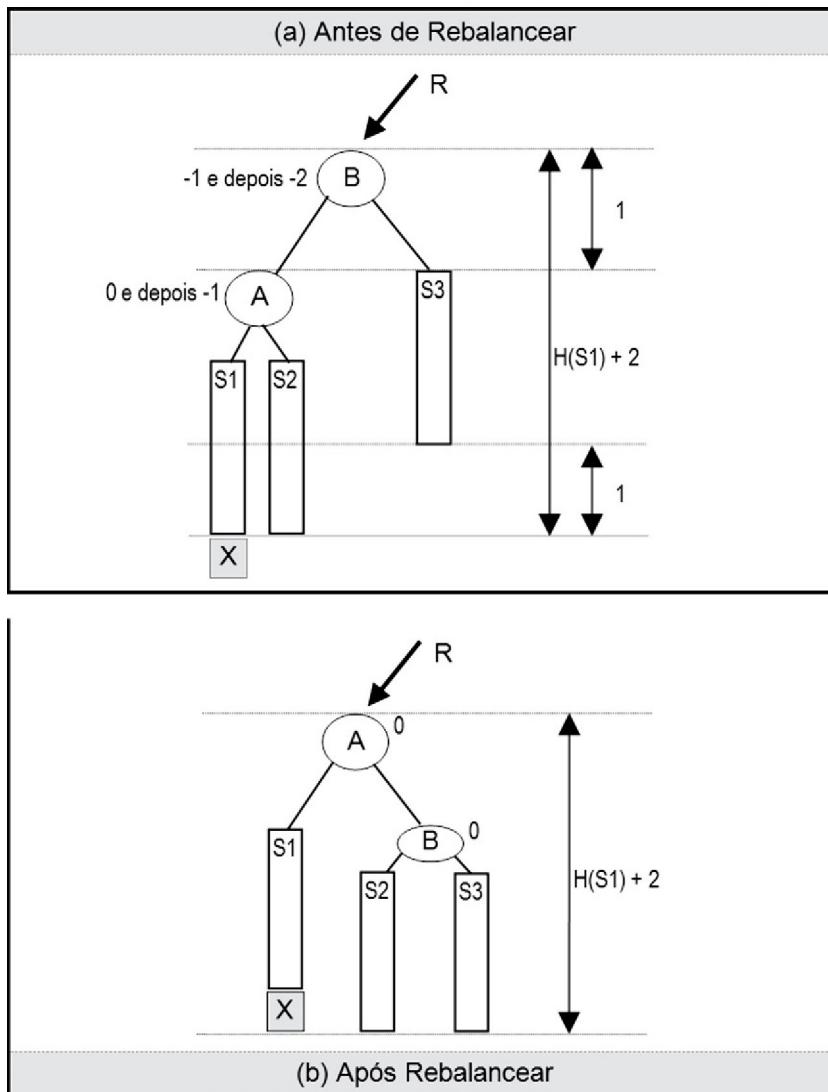


Figura 9.13 Generalização do caso 1 — Rotação Simples EE — estilo de diagrama adaptado de Knuth (1998) e Camargo.

O algoritmo da [Figura 9.14](#) providencia a movimentação das Subárvores e dos ponteiros, partindo da situação da [Figura 9.13a](#) e levando à situação da [Figura 9.13b](#). Em seguida, o algoritmo ajusta os balanceamentos, muda a raiz da Árvore e atualiza uma variável chamada MudouAltura.

Note que a altura da Subárvore S1 é igual à altura da Subárvore S2 e também igual à altura da Subárvore S3. Ou seja, $H(S1) = H(S2) = H(S3)$. Antes da inserção de X, a altura total da Árvore era $H(S1) + 2$. Após a inserção de X e o rebalanceamento, a altura da Árvore continua sendo $H(S1) + 2$ (veja na [Figura 9.13b](#)). Por isso, no algoritmo da [Figura 9.14](#) a variável MudouAltura recebeu o valor Falso.

```

Variável Filho do tipo NodePtr; // Filho é ponteiro auxiliar, que apontará R→Esq
/* movimentando as Subárvores e os ponteiros */
Filho = R→Esq; // Filho aponta para o Nó que contém o valor 'A' - Figura 9.13
R→Esq = Filho→Dir; // R→Esq passa a apontar para S2 - Figura 9.13
Filho→Dir = R; // Filho→Dir passa a apontar o Nó que contém 'B' - Figura 9.13
/* ajustando os balanceamentos */
R→Bal = 0; // atualizando o Fator de Balanceamento de R
Filho→Bal = 0; // atualizando o Fator de Balanceamento de Filho
/* mudando a Raiz da árvore*/
R = Filho; // o Nó que contém 'A' passará a ser a Raiz - Figura 9.13
/* atualizando a variável MudouAltura */
MudouAltura = Falso; // após inserir X e rebalancear, a altura da Árvore continua sendo
                     // a mesma: H(S1) + 2.
    
```

Figura 9.14 Insere em ABBB — caso 1: Rotação Simples EE.

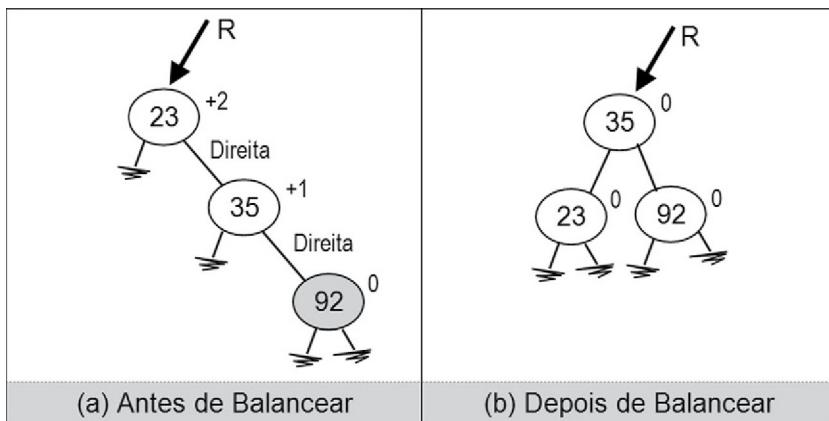


Figura 9.15 Insere em ABBB: caso 2 — Rotação Simples DD. Exemplo com apenas três valores.

Casos de rebalanceamento: caso 2 — Rotação Simples DD

O caso 2 — Rotação Simples Direita-Direita (DD) é um caso de rebalanceamento absolutamente simétrico à Rotação Simples Esquerda-Esquerda (EE). A [Figura 9.15](#) apresenta um exemplo do caso de rebalanceamento Rotação Simples DD. A [Figura 9.15](#) mostra a situação inicial ([Figura 9.15a](#)): o Nó que contém o valor 92 acabou de ser inserido e causou o desbalanceamento. A [Figura 9.15b](#) mostra a situação final da Árvore após ter sido rebalanceada. Não há outra configuração possível que atenda os critérios que definem uma ABBB.

Exercício 9.7 Rebalanceamento — caso 2: Rotação Simples DD

Nas situações da [Figura 9.16](#), um novo valor acabou de ser inserido e causou desbalanceamento. O valor inserido está posicionado em um Nó destacado com fundo cinza. Em cada caso, faça o rebalanceamento pelo caso 2 — Rotação Simples DD. Desenhe

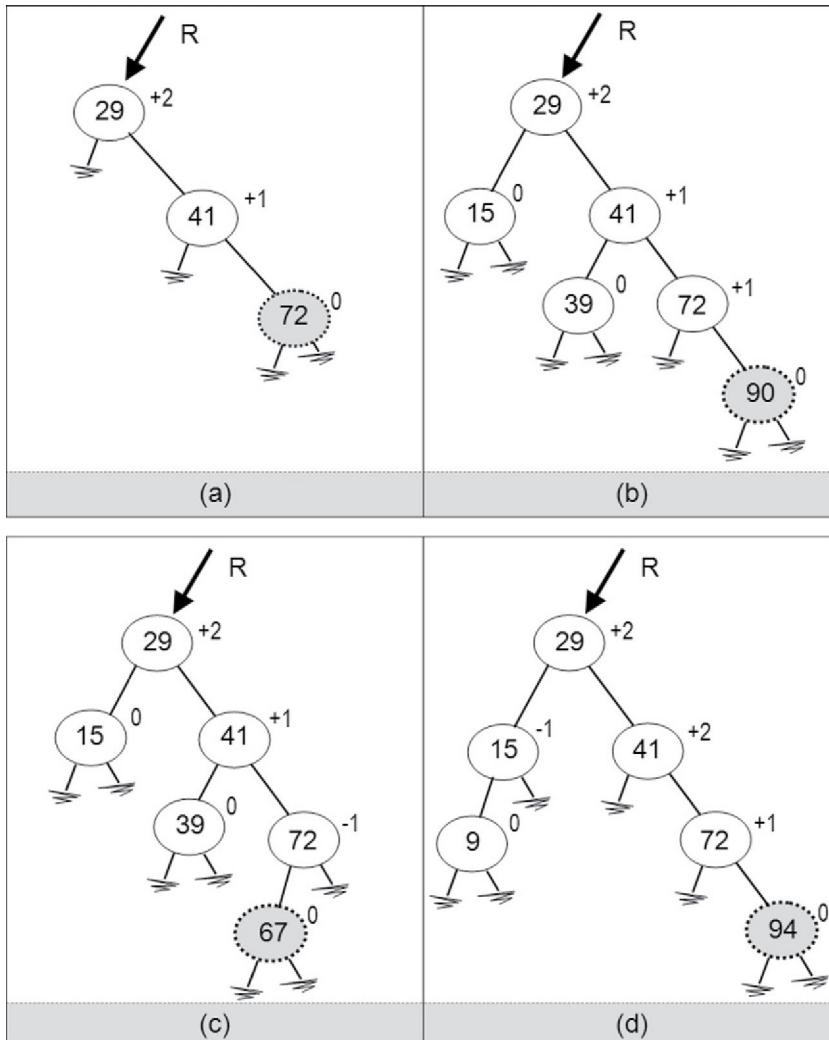


Figura 9.16 Exemplos do caso 2 — Rotação Simples DD.

a Árvore resultante respeitando o critério que define uma ABB e também o critério de Balanceamento. Siga o roteiro da [Figura 9.11](#).

Exercício 9.8 Diagrama — caso 2: Rotação Simples DD

Generalizamos a Rotação Simples EE através do diagrama da [Figura 9.13](#). Desenvolva um diagrama genérico (simétrico ao do [Figura 9.13](#)) para o caso 2: Rotação Simples DD.

Exercício 9.9 Algoritmo — caso 2: Rotação Simples DD

Generalizamos a Rotação Simples EE através do diagrama da [Figura 9.13](#) e então desenvolvemos o trecho de algoritmo da [Figura 9.14](#). Desenvolva um trecho de algoritmo que implemente o caso 2: Rotação Simples DD. O trecho de algoritmo resultante deve ser análogo ao trecho de algoritmo da [Figura 9.14](#).

Exercício 9.10 Como rebalancear?

Na [Figura 9.17](#), o valor 19 acabou de ser inserido e causou desbalanceamento. Como essa Árvore pode ser rebalanceada? Desenhe a nova Árvore respeitando o critério de balanceamento ([Figura 9.4](#)) e também o critério que define uma Árvore Binária de Busca ([Figura 8.4](#)).

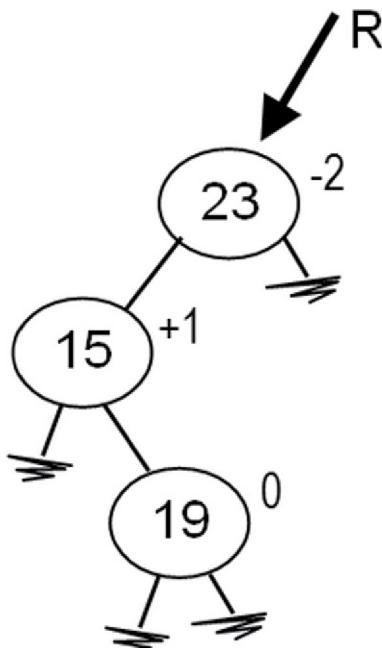


Figura 9.17 Inserção do valor 19 causou desbalanceamento.

Casos de rebalanceamento: caso 3 — Rotação Dupla ED

O Exercício 9.10 é um exemplo do caso de rebalanceamento Rotação Dupla ED ou Rotação Dupla do tipo Esquerda-Direita (nomenclatura de Camargo). A [Figura 9.18](#) mostra a situação inicial ([Figura 9.18a](#)): o Nó que contém o valor 19 acabou de ser inserido e causou o desbalanceamento da Árvore. O nome Rotação Dupla Esquerda-Direita reflete a situação em que, a partir do Nó em que foi detectado o desbalanceamento (Nó apontado por R), em direção ao Nó que causou o desbalanceamento, temos que seguir para a Subárvore Esquerda e depois para a Subárvore Direita.

A [Figura 9.18b](#) mostra a situação final da Árvore após ter sido rebalanceada. Você consegue achar outra configuração para uma Árvore com esses mesmos três valores que respeite o critério de uma ABB e também o critério de balanceamento? Tente achar outra configuração que não a da [Figura 9.18b](#).

Não há outra configuração que atenda a ambos os critérios. A situação da [Figura 9.18b](#) é a única solução. A lógica de平衡amento nesse exemplo com apenas três valores na Árvore é bastante intuitiva. Lembre-se desse exemplo com apenas três valores na Árvore se estiver em dúvida quanto ao caso 3 — Rotação Dupla ED.

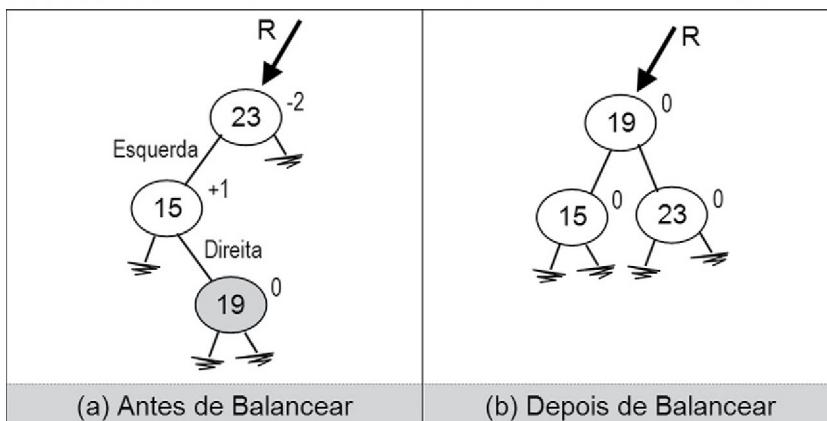


Figura 9.18 Insere em ABBB: caso 3 — Rotação Dupla ED. Exemplo com apenas três valores.

Exemplo com mais de três valores — caso ED

Temos um segundo exemplo da Rotação Dupla Esquerda-Direita na [Figura 9.19](#). Note, na [Figura 9.19a](#), que o valor 19 acabou de ser inserido, causando desbalanceamento, pois a altura da Subárvore Esquerda de R é 3 (três níveis) e a altura da Subárvore Direita de R é 1 (um nível). A diferença entre a altura das Subárvores é 2, o que viola o critério de balanceamento.

Para rebalancear a Árvore, o primeiro passo é identificar os três Nós principais. O primeiro desses Nós é aquele em que foi detectado o desbalanceamento, ou seja, o Nós apontado por R. Para identificar os outros dois Nós principais para o rebalanceamento, caminhamos em direção ao Nós que causou o desbalanceamento (ou seja, o Nós que acabou de ser inserido) e encontramos os Nós com valores 15 e 17 ([Figura 9.19b](#)). Como caminhamos para a esquerda e depois para a direita, fica caracterizado o caso 3 — Rotação Dupla ED.

Note que os valores dos três Nós principais são exatamente os mesmos e nas mesmas posições do Exercício 9.10, que apresentava uma Árvore com apenas três valores. A lógica do rebalanceamento desses três Nós principais será a mesma lógica aplicada no Exercício 9.10, resultando na situação da [Figura 9.19c](#). Após identificar os três Nós principais, pense em uma Árvore com apenas esses três valores. A solução será bastante intuitiva.

Para posicionar os demais valores da Árvore (12, 19 e 29), é preciso seguir o critério que define uma Árvore Binária de Busca: valores menores vão para a Subárvore Esquerda, valores maiores vão para a Subárvore Direita. Considerando que os valores 15, 17 e 23 já estão posicionados ([Figura 9.19c](#)), em qual lugar deve ser posicionada a chave 12 de modo a respeitar o critério que define uma Árvore Binária de Busca? A única posição possível para a chave 12 (sem deslocamento dos valores já posicionados) é à esquerda da chave 15, pois 12 é menor do que 15. Seguindo o mesmo raciocínio, posicionamos o 19 à direita do 15, e o 29 à direita do 23, como mostra a [Figura 9.19d](#). Não há outra posição possível, sem deslocamento dos três valores principais já posicionados, que não quebre o critério que define uma ABB.

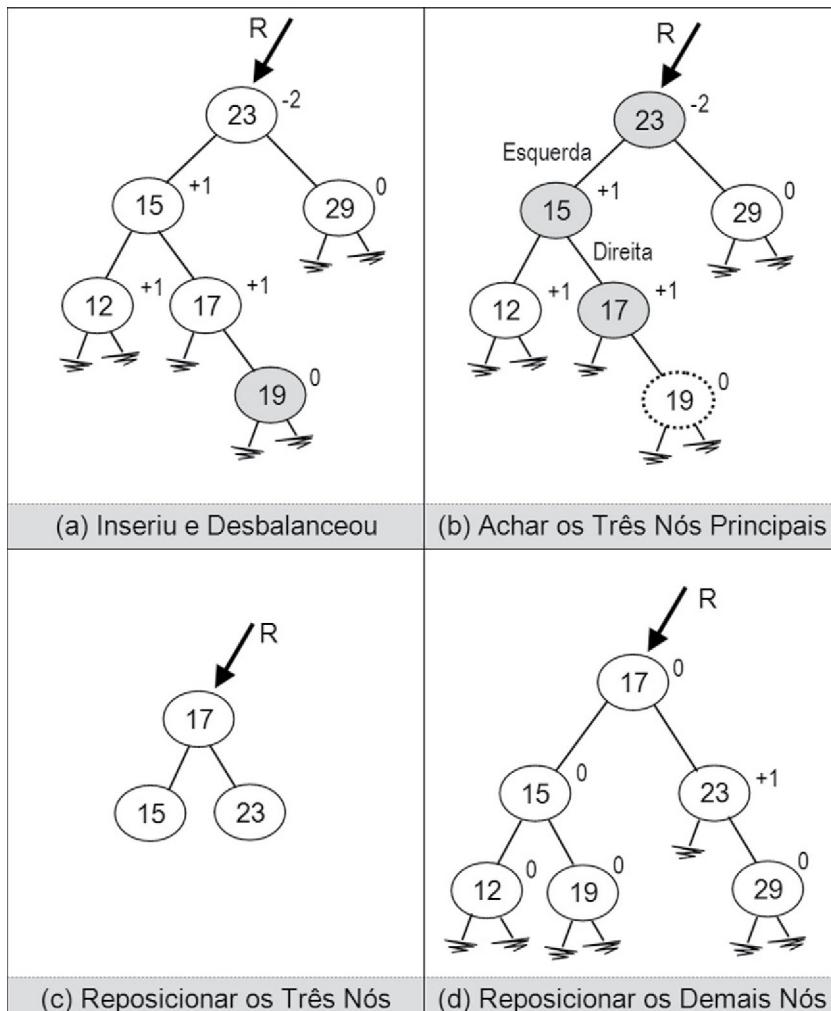


Figura 9.19 Insere em ABBB: caso 3 — Rotação Dupla ED.

Exercício 9.11 Rebalanceamento manual — ED

Na [Figura 9.20](#), um novo valor (destaque em cinza) foi inserido e causou desbalanceamento. Em cada caso, faça o rebalanceamento pelo caso 3: Rotação Dupla ED. Desenhe a Árvore resultante respeitando o critério que define uma ABB e também o critério de Balanceamento. Siga o roteiro da [Figura 9.11](#).

Generalização do caso 3: Rotação Dupla ED — Insere

A [Figura 9.21](#) apresenta um diagrama genérico do rebalanceamento pelo caso 3: Rotação Dupla ED do algoritmo que insere um novo valor na Árvore. Um novo valor X1 ou um novo valor X2 (ou um ou outro; nunca ambos) acabou de ser inserido, causando desbalanceamento.

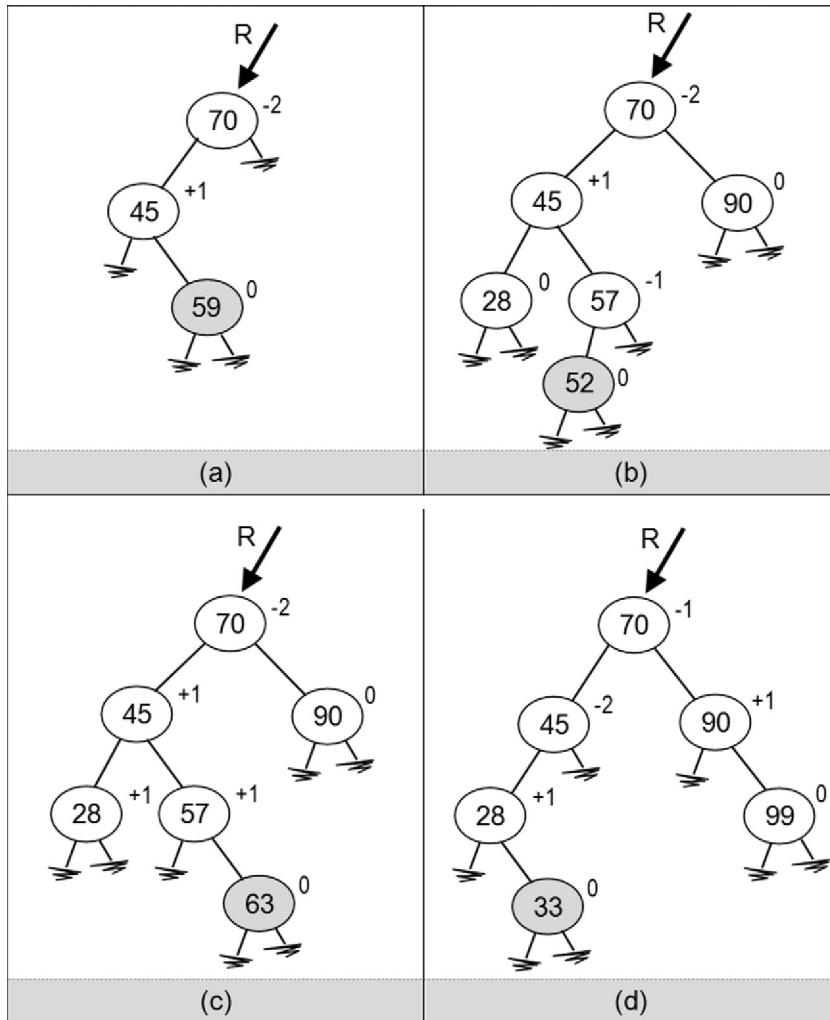


Figura 9.20 Exemplos do caso 3 — Rotação Dupla ED.

O Fator de Balanceamento do Nó que contém o valor 'B' antes da inserção de X era zero, pois a altura das Subárvores Esquerda e Direita era igual. Após a inserção de X1 o Fator de Balanceamento do Nó que contém 'B' passou a ser -1 . Se, em vez de X1, houve a inserção de X2, o Fator de Balanceamento do Nó que contém 'B' passou a ser $+1$.

O Fator de Balanceamento do Nó que contém o valor 'A' antes da inserção de X1 ou X2 era zero e passou a ser $+1$ após a inserção de X1 ou X2. Considerando apenas o Nó que contém o valor 'A' e o Nó que contém o valor 'B', a Árvore continua balanceada.

O Fator de Balanceamento do Nó que contém o valor 'C' antes da inserção de X1 ou X2 era -1 . Após a inserção de X1 ou X2, o Fator de Balanceamento passou a ser -2 , quebrando o critério de平衡amento. Ou seja, a Subárvore Esquerda do Nó que contém 'C' já era maior que a Subárvore Direita; com a inserção de X1 ou

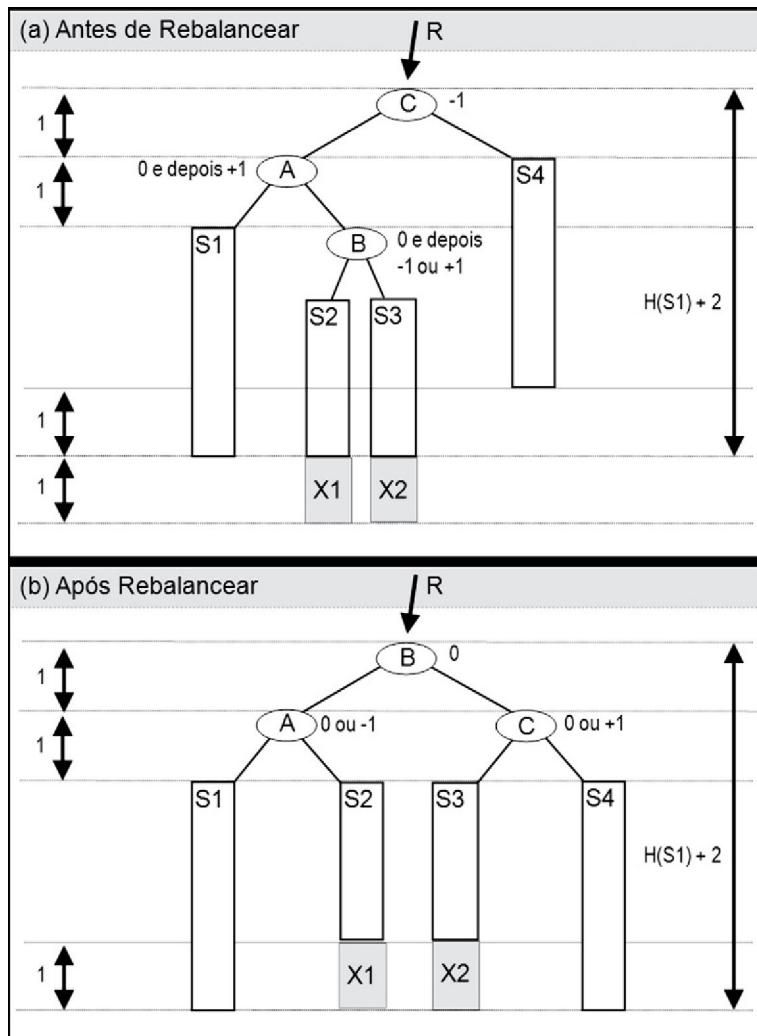


Figura 9.21 Generalização do caso 3 — Rotação Dupla ED — estilo de diagrama adaptado de Knuth (1998) e Camargo.

X₂, a Subárvore Esquerda passou a ser ainda maior, violando o critério. Será preciso rebalancear a Árvore.

O rebalanceamento ilustrado no diagrama da [Figura 9.21](#) generaliza os exemplos numéricos das [Figuras 9.18 e 9.19](#), e também os quatro casos do Exercício 9.11. Procure identificar a equivalência entre os exemplos numéricos e o diagrama genérico da [Figura 9.21](#).

A [Figura 9.22](#) mostra o trecho de algoritmo que implementa o rebalanceamento pelo caso 3: Rotação Dupla ED. É um trecho do algoritmo que insere um valor X em uma Árvore Binária de Busca Balanceada. O algoritmo providencia a movimentação das Subárvore e dos ponteiros, partindo da situação da [Figura 9.21a](#) e levando à situação

```

variável Filho do tipo NodePtr; // Filho é ponteiro auxiliar, que apontará R→Esq
variável Neto do tipo NodePtr; // Neto é ponteiro auxiliar, que apontará Filho→Dir

/* posicionando os ponteiros Filho e Neto */
Filho = R→Esq; // Filho aponta para o Nó que contém o valor 'A' - Figura 2.21
Neto = Filho→Dir; // Filho aponta para o Nó que contém o valor 'B' - Figura 2.21

/* movimentando a Subárvore S2 */
Filho→Dir = Neto→Esq; // Filho→Dir passa a apontar para S2 - Figura 2.21
Neto→Esq = Filho; // Neto→Esq passa a apontar Nó que contém 'A', Figura 2.21

/* movimentando a Subárvore S3 */
R→Esq = Neto→Dir; // R→Esq passa a apontar para S3 - Figura 2.21
Neto→Dir = R; // Neto→Dir passa a apontar o Nó que contém 'C'

/* ajustando os balanceamentos */
Caso Neto→Bal for
    -1 : { R→Bal = 1; // inseriu X1. Exemplo numérico na Figura 9.20b.
            Filho→Bal = 0;
            Neto→Bal = 0; }
    +1 : { R→Bal = 0; // inseriu X2. Exemplo numérico na Figura 9.19 e...
            Filho→Bal = -1;
            Neto→Bal = 0; }
    0 : { R→Bal = 0; // inseriu o Nó apontado por Neto, que contém...
            Filho→Bal = 0; // ... o valor 'B', na Figura 9.21. Exemplo numérico...
            Neto→Bal = 0; } // ... na Figura 9.18

/* mudando a Raiz da árvore*/
R = Neto; // o Nó que contém 'B' passará a ser a Raiz - Figura 2.21

/* atualizando a variável MudouAltura */
MudouAltura = Falso; // após inserir X1 ou X2 e rebalancear...
// ... a altura da Árvore continua a mesma: H(S1) + 2.

```

Figura 9.22 Insere em ABBB — algoritmo do caso 3: Rotação Dupla ED.

da [Figura 9.21b](#). Em seguida, o algoritmo ajusta os balanceamentos, muda a raiz da Árvore e atualiza uma variável chamada MudouAltura.

Note que a altura da Subárvore S1 é igual à altura da Subárvore S4; a altura da Subárvore S2 é igual à altura da Subárvore S3; e S1 e S4 são maiores que S2 e S3 em 1. Ou seja, $H(S1) = H(S4)$; $H(S2) = H(S3)$; $H(S1) = H(S2) + 1$. Antes da inserção de X1 ou X2, a altura total da Árvore era $H(S1) + 2$. Após a inserção de X1 ou X2 e o rebalanceamento, a altura da Árvore continua sendo $H(S1) + 2$ (veja na [Figura 9.21b](#)). Por isso, no algoritmo da [Figura 9.22](#), a variável MudouAltura recebeu o valor Falso.

Casos de rebalanceamento: caso 4 — Rotação Dupla DE

O caso 4 — Rotação Dupla Direita-Esquerda (DE) é um caso de rebalanceamento absolutamente simétrico à Rotação Dupla Esquerda-Direita (ED). A [Figura 9.23](#) apresenta um exemplo do caso de rebalanceamento Rotação Dupla DE. A [Figura 9.23](#) mostra a situação inicial ([Figura 9.23a](#)): o Nó que contém o valor 40 acabou de ser inserido e causou o desbalanceamento. A [Figura 9.23b](#) mostra a situação final da Árvore após

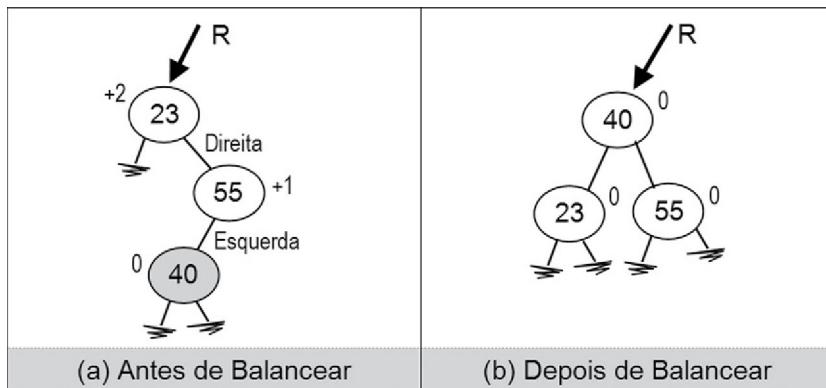


Figura 9.23 Insere em ABBB: caso 4 — Rotação Dupla DE. Exemplo com apenas três valores.

ter sido rebalanceada. Não há outra configuração possível que atenda os critérios que definem uma ABBB.

Exercício 9.12 Rebalanceamento — caso 4: Rotação Dupla DE

Nas situações da [Figura 9.24](#), um novo valor acabou de ser inserido e causou desbalanceamento. O valor inserido está posicionado em um Nó destacado com fundo cinza. Em cada caso, faça o rebalanceamento pelo caso 4 — Rotação Dupla DE. Desenhe a Árvore resultante respeitando o critério que define uma ABB e também o critério de Balanceamento. Siga o roteiro da [Figura 9.11](#).

Exercício 9.13 Diagrama — caso 4: Rotação Dupla DE — diagrama

Generalizamos a Rotação Dupla ED através do diagrama da [Figura 9.21](#). Desenvolva um diagrama genérico (simétrico ao da [Figura 9.21](#)) para o caso 4: Rotação Dupla DE.

Exercício 9.14 Algoritmo — caso 4: Rotação Dupla DE

Generalizamos a Rotação Dupla ED através do diagrama da [Figura 9.21](#) e então desenvolvemos o trecho de algoritmo da [Figura 9.22](#). Desenvolva um trecho de algoritmo que implemente o caso 4: Rotação Dupla DE. O trecho de algoritmo resultante deve ser análogo ao trecho de algoritmo da [Figura 9.22](#).

Exercício 9.15 Algoritmo Insere — ABB Balanceada (ABBB)

No Capítulo 8, elaboramos um algoritmo que insere um novo valor em uma Árvore Binária de Busca — ABB ([Figura 8.20](#)). Adapte esse algoritmo fazendo-o monitorar o balanceamento da Árvore e desencadear ações de rebalanceamento, sempre que necessárias, de modo a manter a Árvore sempre balanceada. Para monitorar o balanceamento, tome como ponto de partida o diagrama da [Figura 9.6](#) e faça um diagrama análogo para a inserção de um valor X na Subárvore Direita de R. Como ações de rebalanceamento, considere os casos 1, 2, 3 e 4 estudados anteriormente.

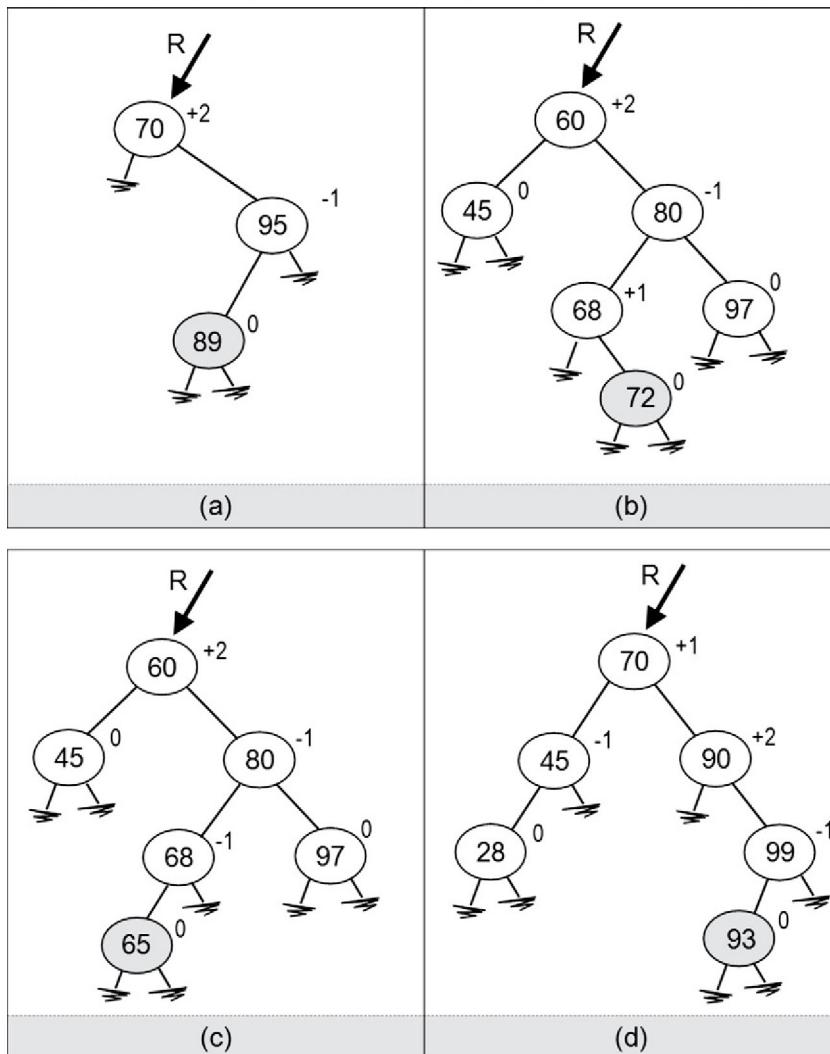


Figura 9.24 Exemplos do caso 4 — Rotação Dupla DE.

A Figura 9.25 apresenta o algoritmo para a operação que insere um valor X em uma Árvore Binária de Busca Balanceada (ABBB). O algoritmo da Figura 9.25 foi elaborado tendo como base, em alguns aspectos, o algoritmo de Camarqo (p. 6-8).

Para ajustar o algoritmo que insere um valor X em uma Árvore Binária de Busca não balanceada (Figura 8.20), cada Nó da Árvore precisará armazenar o seu Fator de Balanceamento. Assim, além dos campos Info, Dir e Esq, cada Nó precisará ter também o campo Bal — Fator de Balanceamento.

O balanceamento de um Nó apontado por um ponteiro R precisa ser ajustado logo após a inserção de um novo valor em uma de suas Subárvores. No algoritmo da [Figura 9.25](#), esse monitoramento é implementado logo após as chamadas recursivas que inserem um valor X nas Subárvores Esquerda ou Direita de R.

```

Insere (parâmetro por referência R tipo ABBB, parâmetro X tipo Inteiro, parâmetro por referência Ok tipo Boolean, parâmetro por referência MudouAltura tipo Boolean) {

    /* Insere o valor X na ABBB de Raiz R como um Nó terminal (sem Filhos). Monitora o balanceamento da Árvore e desencadeia ações de rebalanceamento, caso necessário. Ok retorna Verdadeiro se X foi inserido e Falso caso contrário. MudouAltura retorna Verdadeiro se a inserção de X aumentou a altura da Árvore e Falso caso contrário */

    Variáveis P, Filho, Neto do tipo NodePtr; // NodePtr = Ponteiro para Nó;

    Se (R == Null)
        Então /* Caso 1 da Figura 8.19 - Achou o lugar; insere! */
            P = NewNode; P→Info = X; P→Bal = 0; P→Dir = Null; P→Esq = Null;
            R = P; P = Null; Ok = Verdadeiro; MudouAltura = Verdadeiro;
        } // fim do Caso 1 da Figura 8.19
    Senão Se (X == R→Info)
        Então /* Caso 2 da Figura 8.19: X já está na árvore; não insere! */
            { Ok = Falso; MudouAltura = Falso; } // fim do Caso 2 da Figura 8.19
        Senão {
            Se (R→Info > X)
                Então /* Caso 3 da Figura 8.19: tenta inserir X na Sub Esq de R */
                    { Insere (R→Esq, X, Ok, MudouAltura);
                        // monitora balanceamento voltando de inserir na Sub Esq de R
                        Se MudouAltura // Se a Subárvore esquerda cresceu.
                            Então Caso R→Bal for:
                                +1: { R→Bal = 0; MudouAltura = Falso; } // Figura 9.6a
                                0: R→Bal= -1; // Figura 9.6b.
                                    // MudouAltura continua Verdadeiro
                                -1: /* Figura 9.6c. É preciso rebalancear! */
                                    { Filho = R→Esq;
                                        Se (Filho→Bal == +1)
                                            Então RotDuplaEDInsere; // Figura 9.22
                                        Senão RotSimplesEEInsere; } // Figura 9.14
                            } // fim do Caso 3 da Figura 8.19
                        Senão /* Caso 4 da Figura 8.19: tenta inserir X na Sub Dir de R */
                            { Insere(R→Dir, X, Ok, MudouAltura);
                                // monitora balanceamento voltando de inserir na Sub Dir de R
                                Se MudouAltura // se a Subárvore Direita cresceu...
                                    então Caso R→Bal for:
                                        -1: { R→Bal = 0; MudouAltura = Falso; };
                                        0: R→Bal=1; // MudouAltura continua Verdadeiro
                                        +1: { Filho = R→Dir; // É preciso rebalancear!!
                                            Se (Filho→Bal = -1)
                                                Então RotDuplaDeInsere; // Exercício 9.15
                                            Senão RotSimplesDDInsere; } // Exercício 9.9
                            }; // fim do Caso 4 da Figura 8.19;
                    } // fim do algoritmo Insere em ABBB */
    }
}

```

Figura 9.25 Algoritmo conceitual — Insere em ABBB — algoritmo adaptado de Camargo (p. 6-8).



A [Figura 9.6](#) ilustra o monitoramento e o ajuste do平衡amento quando inserimos um novo valor na Árvore, aumentando a altura da Subárvore Esquerda de R. Assim, logo após retornar da chamada recursiva que insere o novo valor X na Subárvore Esquerda de R — destacada em negrito na [Figura 9.25](#) — tratamos os casos *a*, *b* e *c* da [Figura 9.6](#), trecho de algoritmo também destacado em negrito na [Figura 9.25](#). Compare esse trecho do algoritmo com os diagramas da [Figura 9.6](#).

Nos casos *a* e *b*, o Fator de Balanceamento de R é ajustado, mas a Árvore continua balanceada. No caso *c*, a Árvore precisa ser rebalanceada. Considerando que estamos retornando de uma inserção na Subárvore Esquerda, podem ser desencadeados os casos de rebalanceamento EE (Esquerda-Esquerda) ou ED (Esquerda-Direita), dependendo do Fator de Balanceamento do Filho Esquerdo de R. Se o Fator de Balanceamento do Filho Esquerdo de R for +1 (veja exemplo na [Figura 9.18](#)), desencadeamos o caso ED; caso contrário, desencadeamos o Caso EE (veja exemplo na [Figura 9.15](#)).

Analogamente, ao retornarmos de uma chamada recursiva para inserir um valor X na Subárvore Direita de R, podemos ajustar o Balanceamento de R ou desencadear o rebalanceamento pelos casos DE ou DD. Os algoritmos dos casos EE e ED foram apresentados nas [Figuras 9.14 e 9.22](#); os algoritmos dos casos DD e DE são objeto dos Exercícios 9.9 e 9.15.

A variável MudouAltura recebe o valor Verdadeiro quando um novo Nô é inserido e o valor Falso quando detectamos que, dada a disposição dos demais Nôs, a inserção do novo valor não alterou a altura da Árvore (veja a [Figura 9.6a](#)).

MudouAltura também recebe o valor Falso nos processos de rebalanceamento. Relembre nas [Figuras 9.13 e 9.14](#) que, antes da inserção do novo valor X, a altura da Árvore era $H(S1) + 2$ e após a inserção e o rebalanceamento pelo caso EE a altura da Árvore continuava sendo $H(S1) + 2$. Situação semelhante pode ser observada nas [Figuras 9.21 e 9.22](#), referentes ao caso ED.

Uma vez que a variável MudouAltura receber o valor Falso, não será mais necessário ajustar o balanceamento ou desencadear processos de rebalanceamento. Considere, por exemplo, a situação da [Figura 9.26](#). Antes da inserção do novo valor 22, a altura de R2 — ou seja, a altura da Árvore R na segunda chamada recursiva — era 2 ([Figura 9.26a](#)).

Em R4, ou seja, na quarta chamada recursiva, inserimos efetivamente o novo valor, 22, e com isso a variável MudouAltura recebe o valor Verdadeiro. Voltamos para R3 (terceira chamada recursiva) e ajustamos o Fator de Balanceamento de zero para -1. A variável MudouAltura continuou com o valor Verdadeiro, pois a altura de R3 passou de 1 para 2, mas não detectamos a necessidade de rebalancear em R3. Voltamos então para R2 (segunda chamada recursiva). Após a inserção, a altura de R2 passou a ser 3 e detectamos a necessidade de rebalancear ([Figura 2.26b](#)).

Em R2 executamos o rebalanceamento, caso EE (Rotação Simples Esquerda-Esquerda). Com a execução do rebalanceamento, a altura de R2 volta a ser 2. A altura de R2 antes da inserção era 2; após a inserção e o rebalanceamento, a altura de R2 voltou a ser 2; por isso, a variável MudouAltura recebe o valor Falso ([Figura 2.26c](#)).

Voltamos a R1 (primeira chamada recursiva) com a variável MudouAltura com o valor Falso. Isso significa que solicitamos a inserção de um novo valor na Subárvore Esquerda de R1 através de uma chamada recursiva. Após a execução dessa chamada recursiva, o novo valor foi inserido mas a altura da Subárvore Esquerda continua a mesma. Logo, o balanceamento de R1 não precisa ser ajustado. Como já mencionamos anteriormente, uma vez que a variável MudouAltura recebe o valor Falso, não é mais necessário ajustar o balanceamento ou desencadear processos de rebalanceamento.

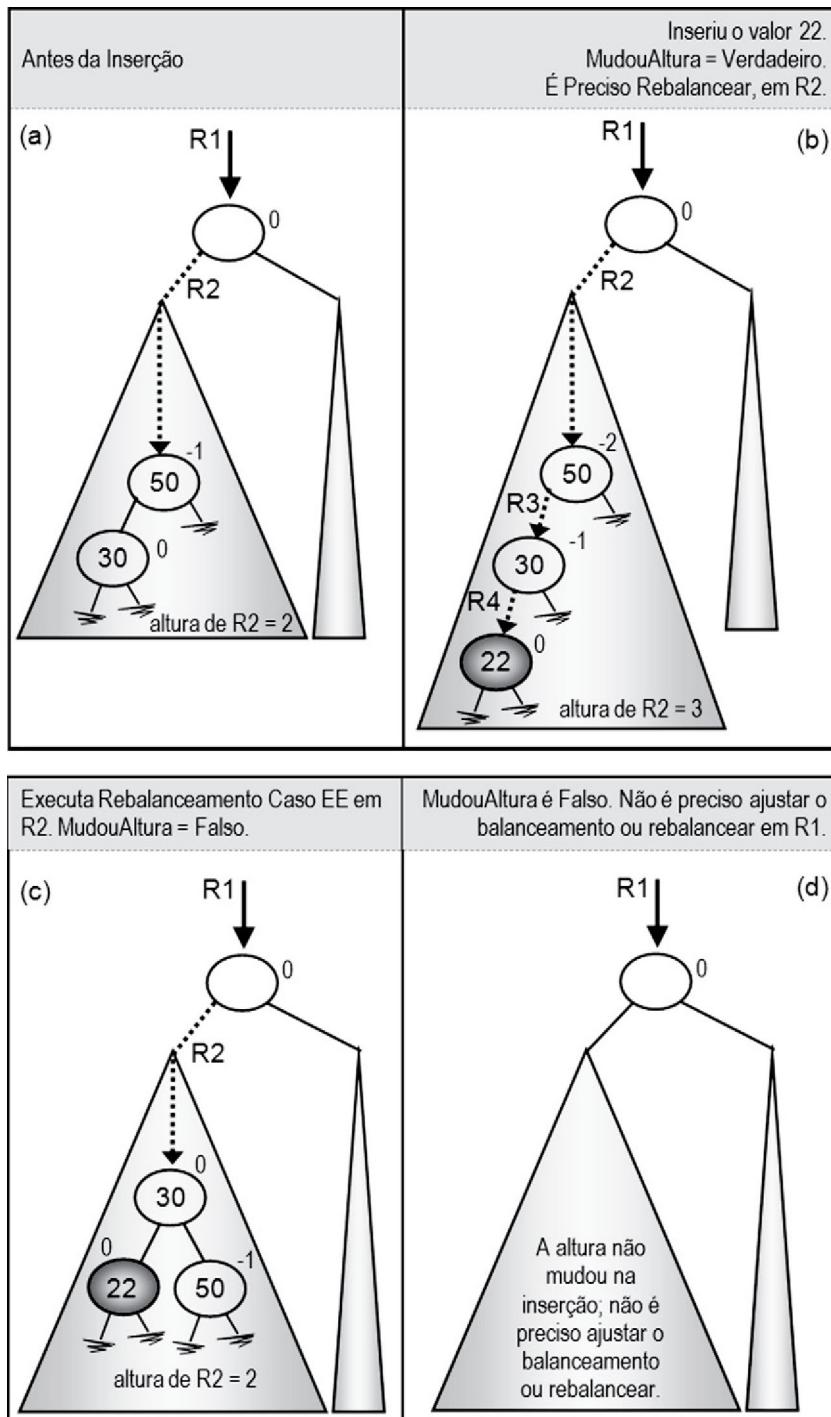


Figura 9.26 Insere em ABBB: exemplo de execução.

9.3 Remover elementos de uma ABB Balanceada

Os ajustes necessários no algoritmo que remove um valor X de uma Árvore Binária de Busca Balanceada (ABB) são análogos aos ajustes que realizamos no algoritmo Insere. Assim como fizemos na [Figura 9.6](#) para o algoritmo Insere, a [Figura 9.27](#) ilustra o ajuste do balanceamento de um Nó apontado por R após a remoção de um valor X, com diminuição da altura da Subárvore Direita.

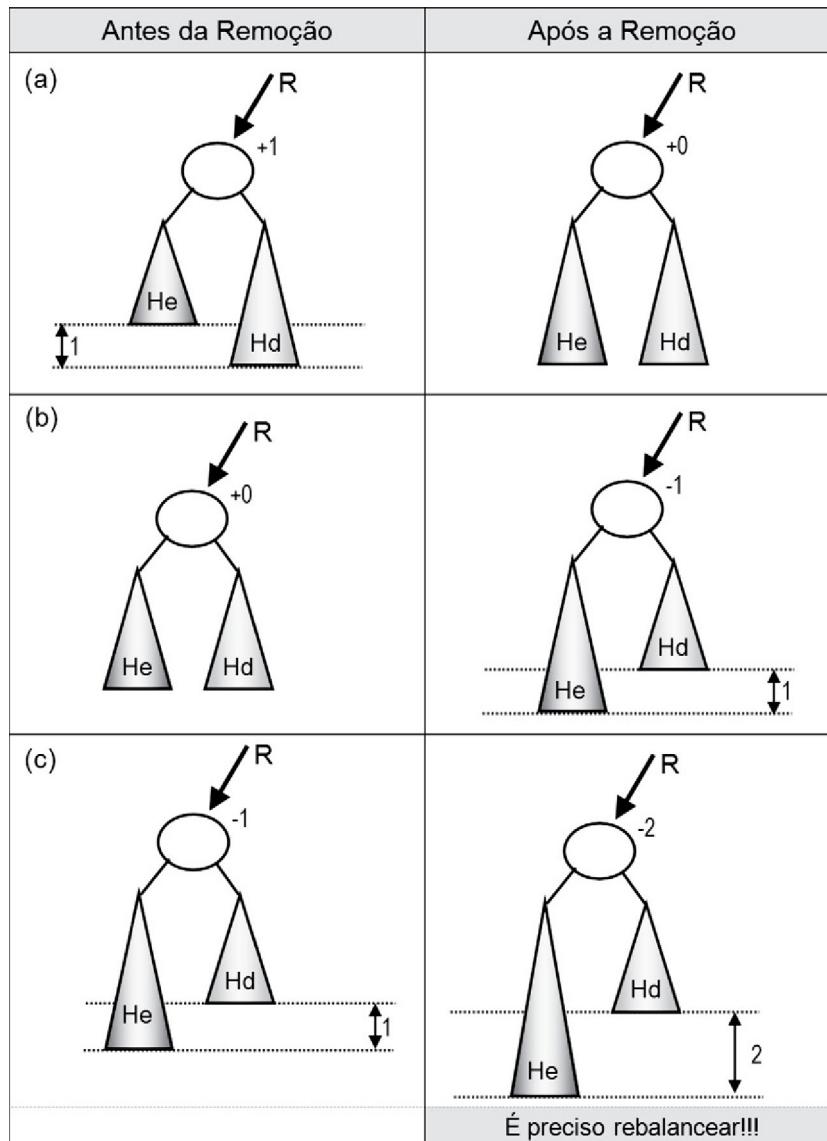


Figura 9.27 Monitorando o Balanceamento na remoção, com redução da Altura da Subárvore Direita.

Compare a [Figura 9.27](#) com a [Figura 9.6](#): os diagramas são idênticos, o que indica que a operação que remove um valor da Subárvore Direita tem implicações muito parecidas com as implicações da operação que insere um novo valor na Subárvore Esquerda.

Na [Figura 9.27a](#), antes da remoção (lado esquerdo do diagrama), o Balanceamento de R tem valor +1. Isso significa que a Subárvore Direita de R era maior que a Subárvore Esquerda, em 1. A remoção de um valor da Subárvore Direita de R causou redução da altura H_d , que passou a ser igual a H_e . Após a remoção (lado direito do diagrama), o Balanceamento de R passou a ser zero.

Na [Figura 9.26b](#), antes da remoção, as alturas das Subárvore H_d e H_e eram iguais e o Balanceamento de R era zero. Após a remoção de um valor da Subárvore Direita de R, H_d passou a ser menor que H_e em 1 e o Balanceamento de R passou a ser -1.

A situação inicial da [Figura 9.26c](#) mostra H_e maior que H_d (em 1) e, consequentemente, o Balanceamento de R tendo valor -1. Com a remoção de um valor da Subárvore Direita de R, H_d diminuiu e passou a ser menor que H_e em 2, o que quebra o critério de balanceamento. Assim, na situação da [Figura 9.26c](#), a operação de remoção implica a necessidade de rebalanceamento da Árvore.

Casos de rebalanceamento

A analogia entre inserir um valor na Subárvore Esquerda e eliminar um valor da Subárvore Direita é válida também nos casos de rebalanceamento. Note, na [Figura 9.28a](#), que a remoção do valor 29 gera uma situação idêntica à da [Figura 9.9](#), que exemplifica o caso de rebalanceamento Rotação Simples do tipo EE do algoritmo Insere. Similarmente, na [Figura 9.28c](#), a remoção do valor 29 gera uma situação idêntica à da [Figura 9.18](#), que exemplifica o caso de rebalanceamento Rotação Dupla do tipo ED do algoritmo Insere.

Contudo, a [Figura 9.28](#) mostra também que, embora os casos de rebalanceamento sejam, em essência, os mesmos que estudamos no algoritmo Insere EE, ED, DD e DE, alguns ajustes precisam ser realizados para sua aplicação na operação Remove. A movimentação das Subárvore e ponteiros é a mesma nos casos do Insere e do Remove, mas os ajustes nos valores do Fator de Balanceamento e no valor da variável MudouAltura precisam ser adaptados para a operação Remove. Por exemplo, na [Figura 9.28a](#), ao contrário do que ocorre no caso EE do Insere, a variável MudouAltura precisa continuar com valor Verdadeiro, pois a altura mudou de 3 para 2 com a remoção e rebalanceamento. Essa diferença com relação à variável MudouAltura também ocorre no caso ED ([Figura 9.28c](#)).

Na [Figura 9.28b](#), após o rebalanceamento, o Fator de Balanceamento do Nó que contém o valor 15 é +1, e o do Nó que contém o valor 23 é -1. No caso EE do Insere, o Fator de Balanceamento de todos os Nós movimentados passa a ser zero. Note que o Balanceamento do Filho Esquerdo de R na [Figura 9.28a](#) é -1 e na [Figura 9.28b](#) é zero. No Insere, no caso EE, o rebalanceamento do Filho Esquerdo de R sempre será -1.

Exercício 9.16 Remove ABBA: rebalanceamento — caso EE

Aplice manualmente o caso de rebalanceamento EE do Remove nos exemplos numéricos da [Figura 9.29](#). A movimentação das Subárvore e ponteiros deve ser idêntica ao que ocorre no caso EE do Insere, mas os valores do Fator de Balanceamento e da variável MudouAltura devem ser ajustados. Aplice o caso EE do Remove quando o Balanceamento do Filho Esquerdo de R for -1 ou zero.

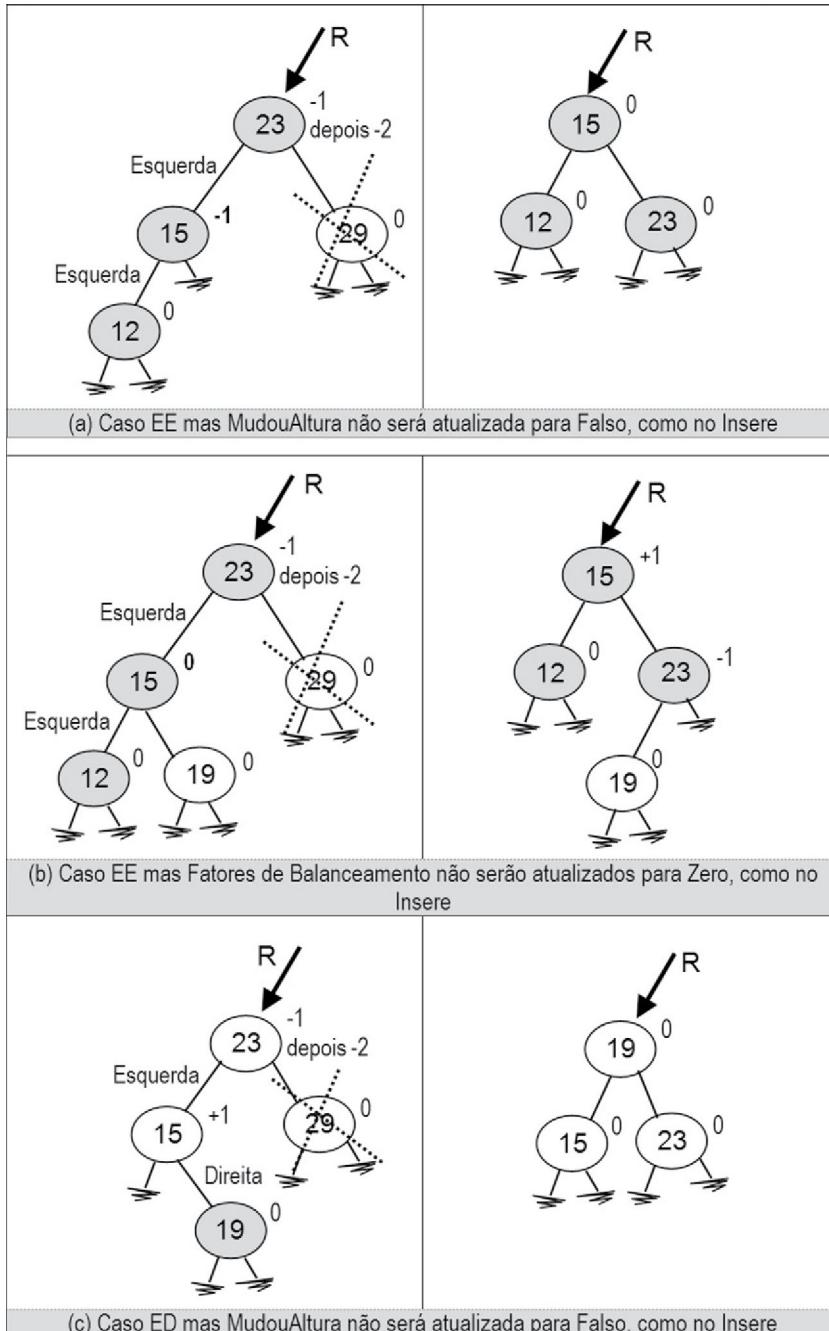


Figura 9.28 Remove de ABBB: casos de rebalanceamento análogos aos do Insere, mas com ajustes.



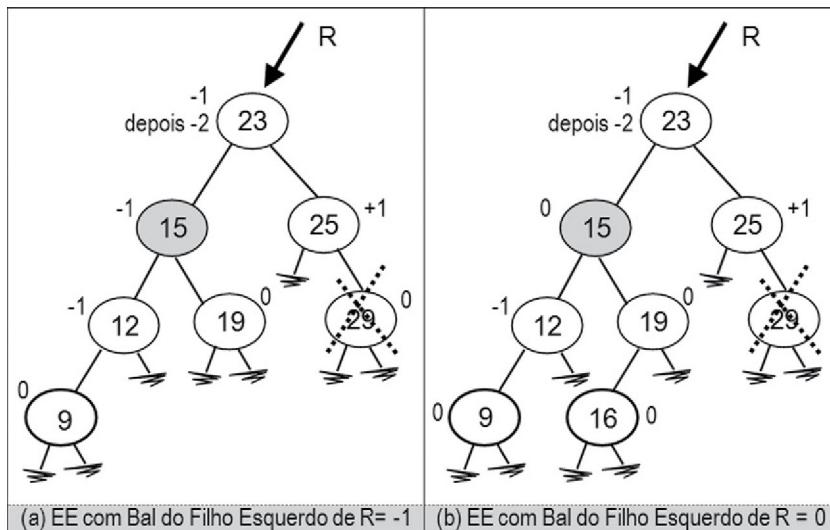


Figura 9.29 Remove de ABBB: exemplos do caso 5 — rotação simples EE do Remove.

Exercício 9.17 Remove de ABBB: rebalanceamento manual ED

Aplique manualmente o caso de rebalanceamento ED do Remove nos exemplos numéricos da [Figura 9.30](#). A movimentação das Subárvores e ponteiros deve ser idêntica ao que ocorre no caso ED do Insere, mas os valores do Fator de Balanceamento e da variável MudouAltura devem ser ajustados. Aplique o caso ED do Remove somente quando o Balanceamento do Filho Esquerdo de R for +1.

Exercício 9.18 Generalização do caso 5: Rotação Simples EE do Remove — diagrama e algoritmo

Analogamente ao que fizemos para o algoritmo Insere nas [Figuras 9.13 e 9.14](#), faça um diagrama generalizando a Rotação Simples EE do Remove e desenvolva um algoritmo que implemente esse caso de rebalanceamento.

Exercício 9.19 Generalização do caso 6: Rotação Simples DD do Remove — diagrama e algoritmo

O caso DD do Remove é simétrico ao caso EE do Remove. Faça um diagrama generalizando a Rotação Simples DD do Remove e desenvolva um algoritmo que implemente esse caso de rebalanceamento.

Exercício 9.20 Generalização do caso 7: Rotação Dupla ED do Remove — diagrama e algoritmo

Analogamente ao que fizemos para o algoritmo Insere nas [Figuras 9.21 e 9.22](#), faça um diagrama generalizando a Rotação Dupla ED do Remove e desenvolva um algoritmo que implemente esse caso de rebalanceamento.

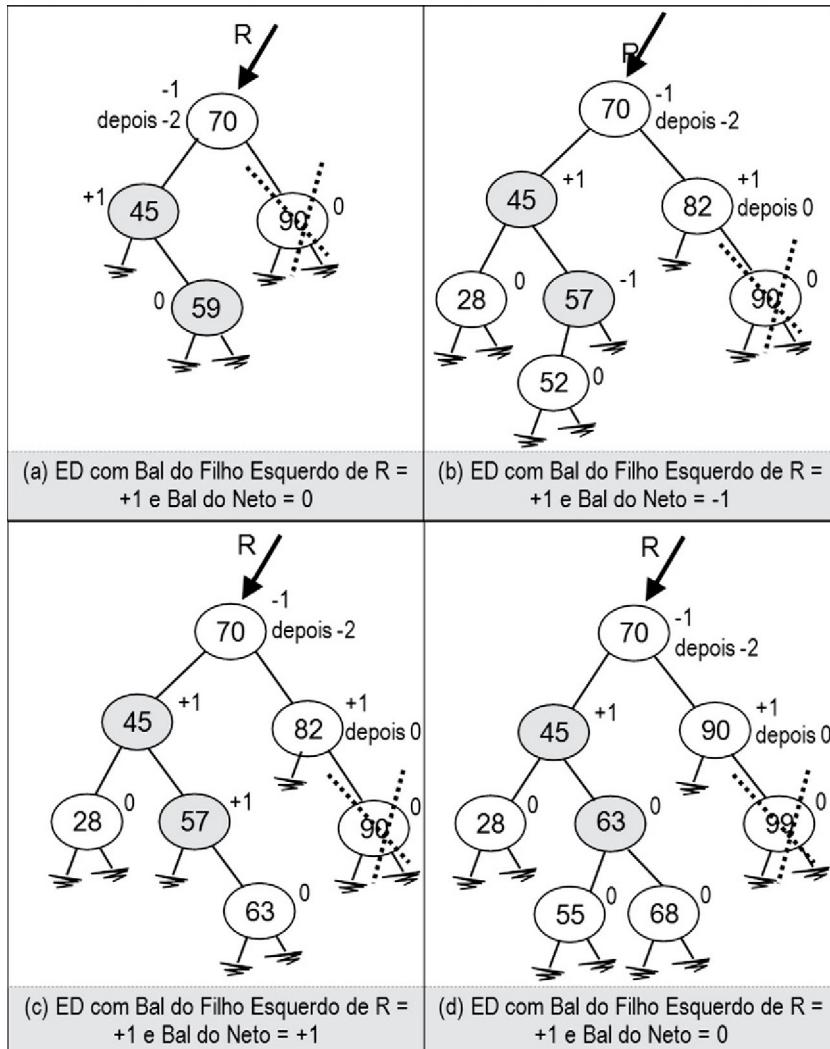


Figura 9.30 Remove de ABBB: exemplos do caso 7 — Rotação Dupla ED do Remove.

Exercício 9.21 Generalização do caso 8: Rotação Dupla DE do Remove — algoritmo

O caso DE do Remove é simétrico ao caso ED do Remove. Desenvolva um algoritmo generalizando a Rotação Dupla DE do Remove.

Exercício 9.22 Algoritmo Remove para uma Árvore Binária de Busca Balanceada (ABBB)

No Capítulo 8 elaboramos um algoritmo para remover um valor X de uma Árvore Binária de Busca — ABB (Exercício 8.11). Adapte esse algoritmo fazendo-o monitorar o balanceamento da Árvore e desencadear ações de rebalanceamento, sempre que necessário, de modo a manter a Árvore sempre balanceada. Para realizar o monitoramento do balanceamento,

tome como ponto de partida o diagrama da [Figura 9.27](#). Como ações de rebalanceamento, considere os casos EE, DD, ED e DE do Remove, objeto dos Exercícios 9.18 a 9.21.

Remove (parâmetro por referência R tipo ABBB, parâmetro X tipo Inteiro, parâmetro por referência Ok tipo Boolean, parâmetro por referência MudouAltura tipo Boolean) {

/* Remove o valor X da ABBB de Raiz R. Monitora o balanceamento e dispara processos de rebalanceamento, sempre que necessário, para manter a Árvore sempre balanceada. Ok retorna Verdadeiro se X for encontrado e removido, e Falso caso contrário. MudouAltura retorna Verdadeiro se na remoção de X a altura da Árvore diminuiu e Falso caso contrário. */

Exercício 9.23 Implemente uma Árvore Binária de Busca Balanceada (ABBB) em uma linguagem de programação

Implemente uma Árvore Binária de Busca Balanceada (ABBB) como um Tipo Abstrato de Dados, com operações Cria, Vazia, Insere, Remove e Destroi. Implemente em uma linguagem de programação, como C++.

Desempenho depende de平衡amento!

Para que uma Árvore Binária de Busca mantenha seu excelente desempenho durante todo o tempo, ela precisa estar balanceada durante todo o tempo. Para isso, os algoritmos para inserir e eliminar elementos precisam monitorar o balanceamento da Árvore e desencadear operações de rebalanceamento sempre que necessário.

Consulte nos Materiais Complementares

Vídeos sobre Árvores Balanceadas

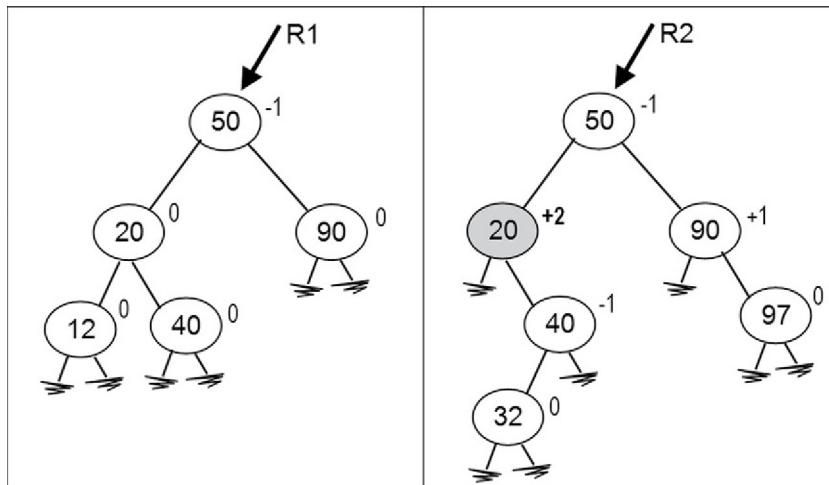


<http://www.elsevier.com.br/edcomjogos>

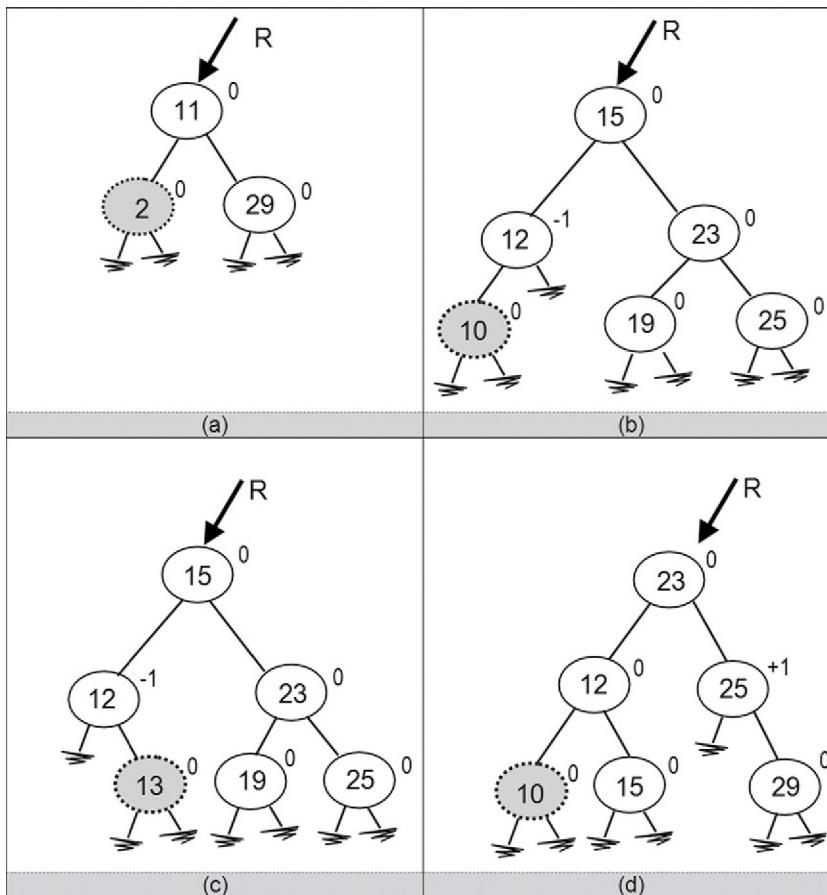


Exercícios de fixação

Exercício 9.24 Execute algum simulador de operações em uma Árvore Binária de Busca Balanceada, como, por exemplo, o Tree Explorer ([link 4](#)). Execute operações para inserir e eliminar elementos.

Soluções para alguns dos exercícios*Exercício 9.3 Fator de Balanceamento**Exercício 9.4 Causaria desbalanceamento?*

A inserção de 25 e de 40 não causaria desbalanceamento. A inserção dos demais valores causaria.

Exercício 9.6 Rebalanceamento manual


No caso *d*, note que o Nó em que o desbalanceamento foi detectado é o Nó que contém o valor 15, e não o Nó apontado por *R*.

Exercício 9.9 Generalização do caso 2 — Rotação Simples DD do Insere — algoritmo

```

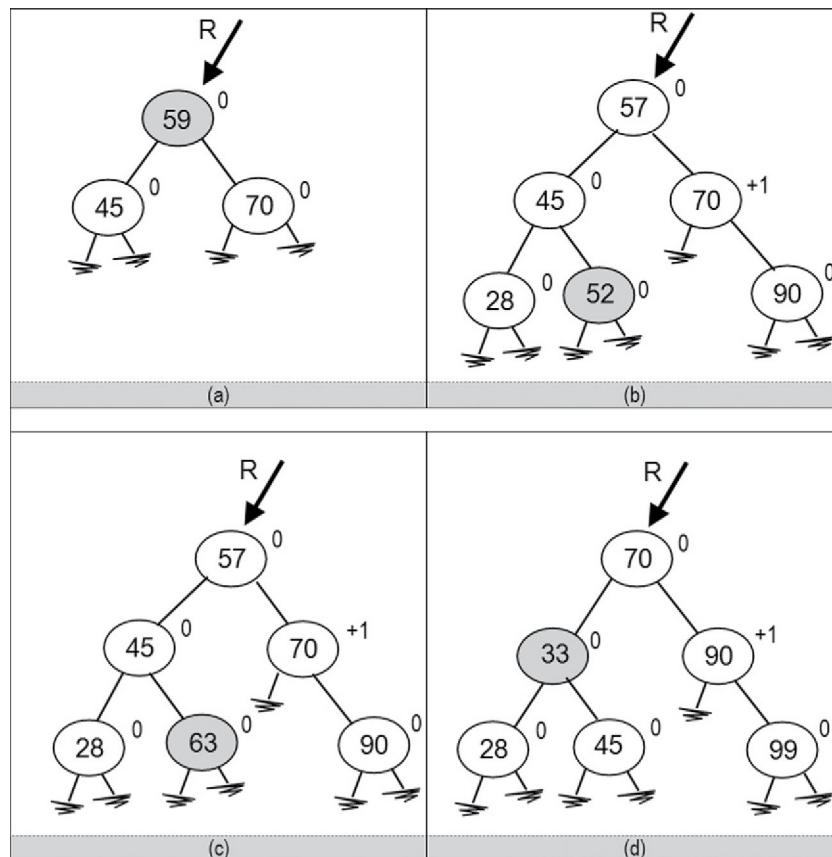
/* caso absolutamente simétrico ao caso EE; o que era Esq passa a ser Dir e vice-versa */
variável Filho do tipo NodePtr;           // Filho é ponteiro auxiliar, que apontará R→Dir
/* movimentando as Subárvores e os ponteiros */
Filho = R→Dir;
R→Dir = Filho→Esq;
Filho→Esq = R;

/* ajustando os balanceamentos */
R→BAL = 0;                                // atualizando o Fator de Balanceamento de R
Filho→Bal = 0;                             // atualizando o Fator de Balanceamento de Filho

/* mudando a Raiz da árvore */
R = Filho;
/* atualizando a variável MudouAltura */
MudouAltura = Falso;                      // após inserir X e rebalancear, a altura da Árvore...
                                            // ... continua a mesma: H(S1) + 2.

```

Exercício 9.11 Rebalanceamento manual — ED



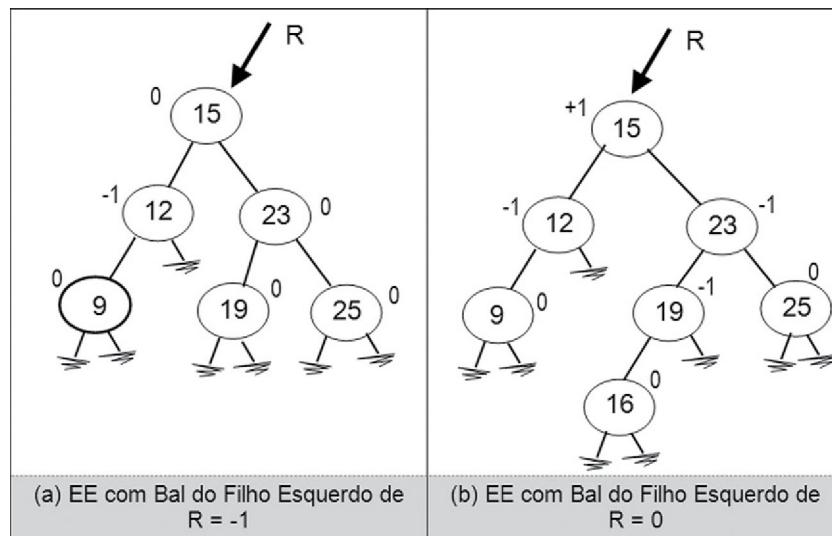
Exercício 9.14 Generalização do caso 4: Rotação Dupla DE — Insere — algoritmo

```

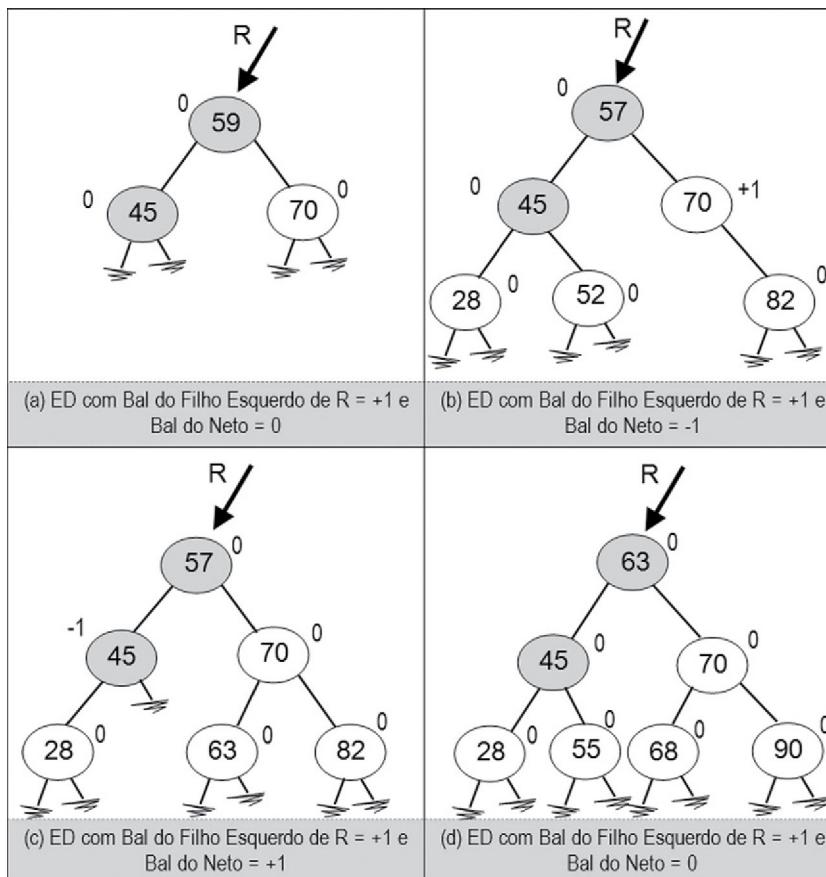
/* caso simétrico ao caso ED; o que era Esq passa a ser Dir e vice-versa */
variável Filho do tipo NodePtr;
variável Neto do tipo NodePtr;
/* posicionando os ponteiros Filho e Neto */
Filho = R→Dir;
Neto = Filho→Esq;
/* movimentando a Subárvore S2 */
Filho→Esq = Neto→Dir;
Neto→Dir = Filho;
/* movimentando a Subárvore S3 */
R→Dir = Neto→Esq;
Neto→Esq = R;
/* ajustando os balanceamentos */
Caso Neto→Bal for
    -1 : { R→Bal = 0;           // inseriu X2.
            Filho→Bal = 1;
            Neto→Bal = 0; };
    +1 : { R→Bal = -1;         // inseriu X1.
            Filho→Bal = 0;
            Neto→Bal = 0; };
    0 : { R→Bal = 0;           // inseriu o Nó apontado por Neto; caso com apenas três valores
            Filho→Bal = 0;
            Neto→Bal = 0; };
/* mudando a Raiz da árvore*/
R = Neto;           // o Nó que contém 'B' passará a ser a Raiz - Figura 2.21
/* atualizando a variável MudouAltura */
MudouAltura = Falso; // após inserir X1 ou X2 e rebalancear a altura da Árvore continua a
mesma: H(S1) + 2.

```

Exercício 9.16 Remove de uma ABBB: rebalanceamento manual — caso EE



Exercício 9.17 Remove de ABBB: rebalanceamento manual — caso ED



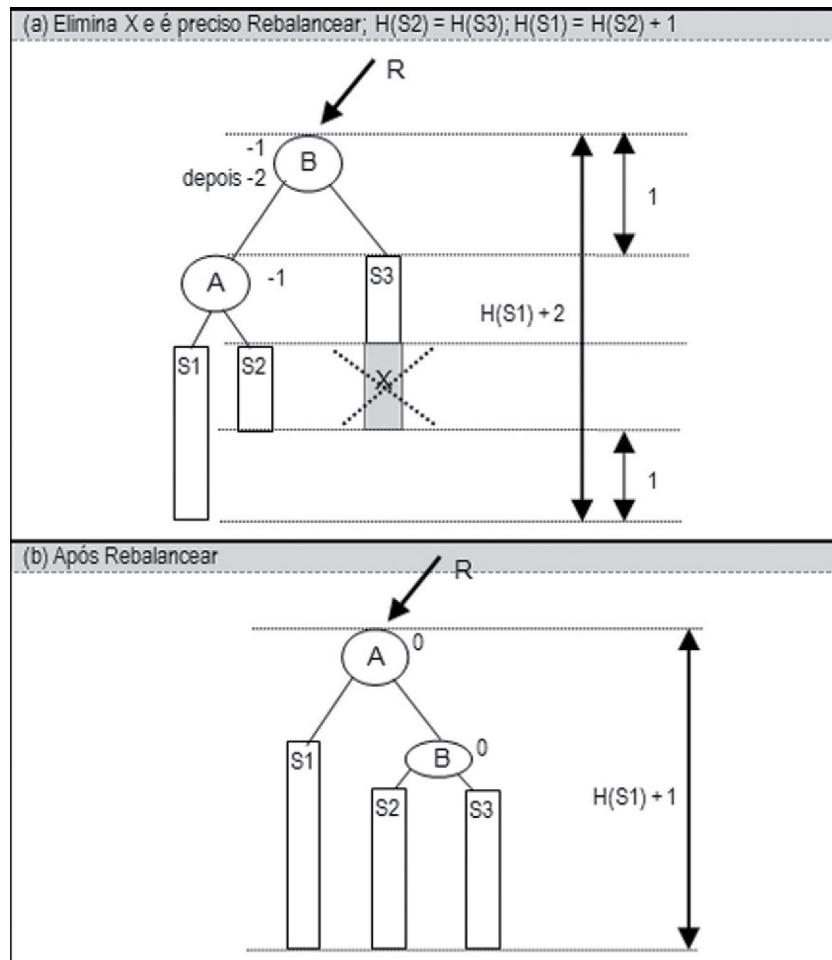
*Exercício 9.18 Generalização do caso 5: Rotação Simples EE do Remove — diagrama e algoritmo*

Dividimos o caso EE do Remove em dois subcasos, para mostrar algumas diferenças: caso EE(a): Quando $\text{Filho} \rightarrow \text{Bal} = -1$; caso EE(b): Quando $\text{Filho} \rightarrow \text{Bal} = 0$. Para exemplificar as diferenças, note que, quando Bal do Filho é -1 , a altura da Árvore muda; quando o Bal do Filho é zero, a altura da Árvore não muda. Observe também os Balanceamentos.

Caso EE (a) do Remove: quando $\text{Filho} \rightarrow \text{Bal} = -1$

```
variável Filho do tipo NodePtr;           // Filho é ponteiro auxiliar, que apontará R→Esq
/* movimentando as Subárvore e os ponteiros */
Filho = R→Esq;
Se (Filho→Bal == -1)
Então {
    R→Esq = Filho→Dir;
    Filho→Dir = R;
    /* ajustando os balanceamentos */
    R→BAL = 0;    // atualizando o Fator de Balanceamento de R
    Filho→Bal = 0; // atualizando o Fator de Balanceamento de Filho
    /* mudando a Raiz da árvore*/
    R = Filho;
    /* a variável MudouAltura continua Verdadeiro */
    MudouAltura = Verdadeiro; }
```

Caso EE (a) do Remove: quando Filho→Bal = -1



* Estilo de diagramas adaptado de Knuth (1998) e Camargo.

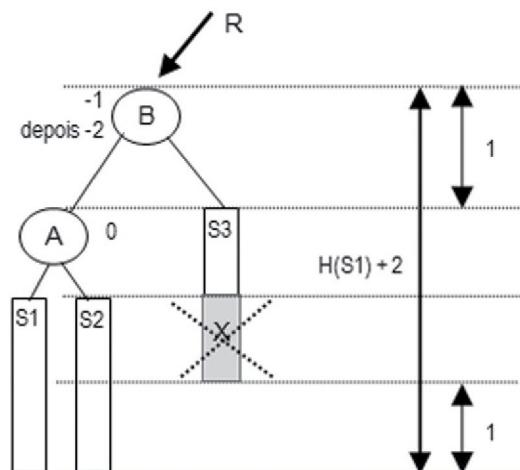
Caso EE (b) do Remove: quando Filho→Bal = 0

```

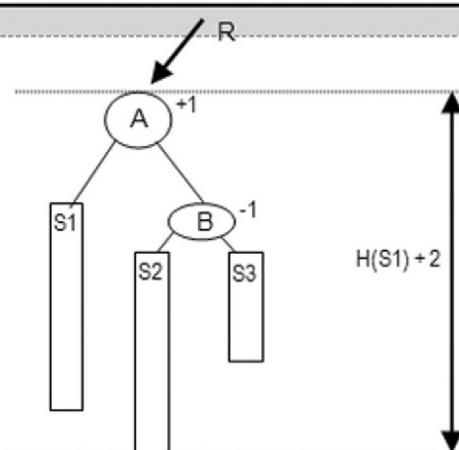
variável Filho do tipo NodePtr;           // Filho é ponteiro auxiliar, que apontará R→Esq
/* movimentando as Subárvores e os ponteiros */
Filho = R→Esq;
Se (Filho→Bal == 0)
Então {
    R→Esq = Filho→Dir;
    Filho→Dir = R;
    /* ajustando os balanceamentos */
    R→BAL = -1;      // atualizando o Fator de Balanceamento de R
    Filho→Bal = +1; // atualizando o Fator de Balanceamento de Filho
    /* mudando a Raiz da árvore*/
    R = Filho;
    /* atualizando a variável MudouAltura */
    MudouAltura = Falso;
}
  
```

Caso EE (b) do Remove: quando Filho→Bal = 0

(a) Elimina X e é preciso Rebalancear; $H(S1) = H(S2); H(S1) = H(S3) + 1$



(b) Após Rebalancear



* Estilo de diagramas adaptado de [Knuth \(1998\)](#) e Camargo.

Exercício 9.20 Generalização do caso 7: Rotação Dupla ED do Remove — algoritmo

```
variável Filho do tipo NodePtr;      // Filho é ponteiro auxiliar, que apontará R→Esq
variável Neto do tipo NodePtr;      // Neto é ponteiro auxiliar, que apontará Filho→Dir

Filho = R→Esq;          /* posicionando os ponteiros Filho e Neto */
Neto = Filho→Dir;

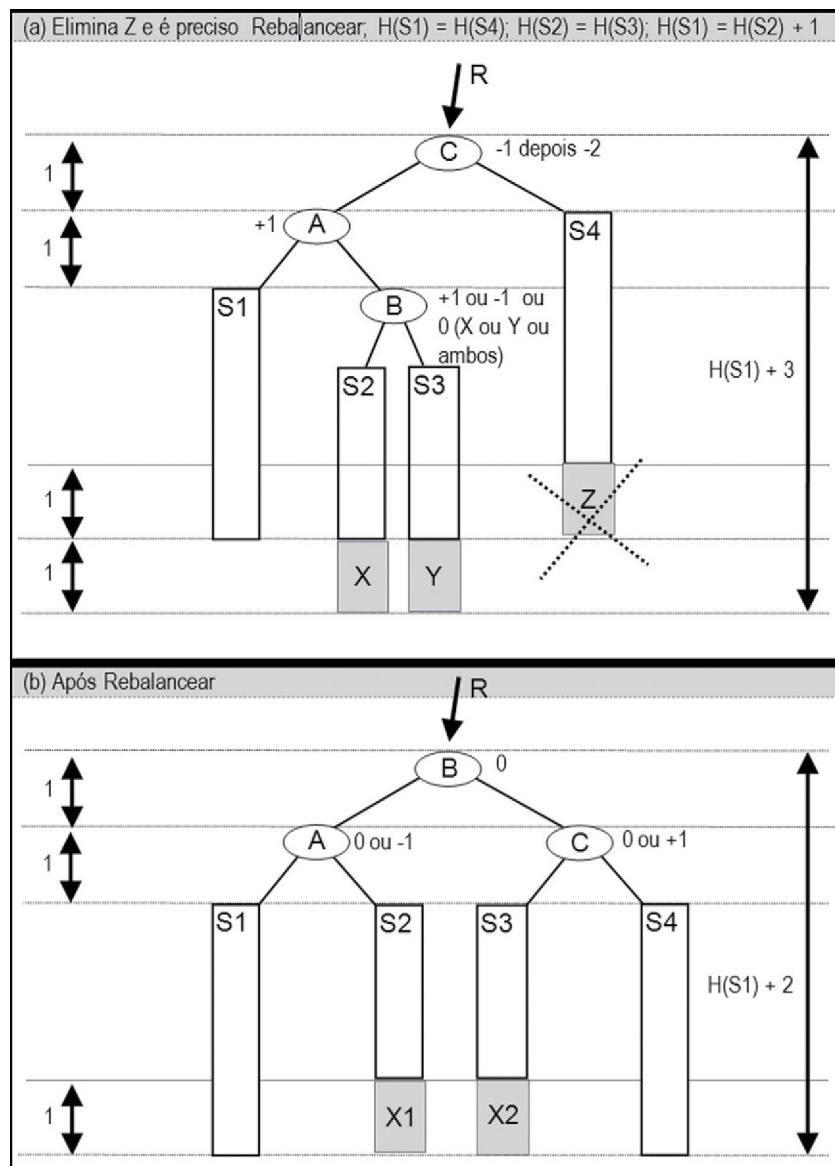
Filho→Dir = Neto→Esq;      /* movimentando a Subárvore S2 */
Neto→Esq = Filho;

R→Esq = Neto→Dir;      /* movimentando a Subárvore S3 */
Neto→Dir = R;

Caso Neto→Bal for      /* ajustando os balanceamentos */
-1 : { R→Bal = 1;
        Filho→Bal = 0;
        Neto→Bal = 0; };
+1 : { R→Bal = 0;
        Filho→Bal = -1;
        Neto→Bal = 0; };
0 : { R→Bal = 0;
        Bal (Filho) = 0;
        Neto→Bal = 0; };

R = Neto;    /* mudando a Raiz da árvore*/
MudouAltura = Verdadeiro; /* variável MudouAltura - continua Verdadeiro */
```

Exercício 9.20 Generalização do caso 7: Rotação Dupla ED do Remove — diagrama



* Estilo de diagramas adaptado de [Knuth \(1998\)](#) e Camargo.

Exercício 9.22 Algoritmo Remove para uma Árvore Binária de Busca Balanceada (ABBB)

```

/* Esse algoritmo foi elaborado tendo como base, em alguns aspectos, o algoritmo de Camargo, p. 10-13 */
Remove (parâmetro por referência R do tipo ABBB, parâmetro X do tipo Inteiro, parâmetro por
referência Ok do tipo Boolean, parâmetro por referência MudouAltura tipo Boolean) {
    /* Remove o valor X da ABB de Raiz R. Monitora o balanceamento e dispara processos de
    rebalanceamento, sempre que necessário, para manter a Árvore sempre balanceada. Ok retorna
    Verdadeiro se X for encontrado e removido, e Falso caso contrário. MudouAltura retorna
    Verdadeiro se na remoção de X a altura da Árvore diminuiu e Falso caso contrário */

    variável Aux do tipo NodePtr;           // NodePtr = Ponteiro para Nó;

    Se (R == Null)
        Então      // Caso 1: Árvore Vazia: não remove.
            { Ok = Falso; MudouAltura = Falso; }
        Senão
            Se (R→Info > X)
                Então      // Caso 3: remove X da Subárvore Esq de R
                    { Remove (R→Esq, X, Ok, MudouAltura);

                        /* Monitora balanceamento após eliminar da Subárvore Esquerda */
                        Se MudouAltura
                            Então Caso R→Bal for:
                                -1: R→Bal = 0; // MudouAltura continua Verdadeiro
                                0: { R→Bal = 1; MudouAltura = Falso;};
                                1: { Filho = R→Dir;   // vai precisar balancear
                                      Se (Filho→Bal == -1)
                                          Então RotDuplaDE-Remove
                                          Senão RotSimplesDD-Remove; // casos Filho→Bal = 0 e +1
                                      }; // fim do Caso
                            } // fim do Caso 3
                        Senão
                            Se (R→Info < X)
                                Então      // Caso 4: remove X da Subárvore Dir de R
                                    { Remove(R→Dir, X, Ok, MudouAltura);

                                        /* Monitora balanceamento após eliminar da Subárvore Direita */
                                        Se MudouAltura
                                            Então Caso R→Bal for:
                                                1: R→Bal = 0; // MudouAltura continua Verdadeiro
                                                0: { R→Bal = -1; MudouAltura = Falso;};
                                                -1: { Filho= R→Esq; // vai precisar balancear
                                                      Se (Filho→Bal == 1)
                                                          Então RotDuplaED-Remove
                                                          Senão RotSimplesEE-Remove; // Filho→Bal=0 e -1
                                                      }; // fim do Caso
                                            } // fim do Caso 4
                                        Senão
                                    /* Caso 2: Encontrou X - Remove e Ajusta a Árvore. 3 casos: 0, 1 ou 2 Filhos */
                                    { Aux = R;

```

```

Se ((R→Esq == Null) E (R→Dir == Null ))
Então { DeleteNode(Aux); R=NULL; Ok=Verdadeiro; MudouAltura=Verdadeiro; } // Zero Filhos
Senão Se ((R→Dir != Null) E (R→Esq != Null))
    Então { /* 2 Filhos. Acha o Nó que contém o Maior Elemento da Subárvore Esquerda de R. O maior é o elemento
        mais à direita da Subárvore. Ele nunca terá o Filho Direito */
        Aux = R→Esq;
        Enquanto (Aux→Dir != Null) Faça
            Aux = Aux→Dir;

        /* Substitui o valor de R→Info - que é o elemento que estamos querendo eliminar - pelo valor do
        Maior da Subárvore Esquerda de R. A Árvore ficará com 2 elementos com o mesmo valor. */
        R→Info = Aux→Info; // Aux aponta para o Nó que contém o Maior

        /* Remove o valor repetido da Subárvore Esquerda de R através de uma chamada recursiva.
        Atenção aos parâmetros. Não estamos mais removendo X, mas R→Info, que está repetido.
        Não estamos mais removendo de R, mas de R→Esq. */
        Remove(R→Esq, R→Info, Ok, MudouAltura);

        /* Monitora balanceamento após eliminar da Subárvore Esquerda */
        Se MudouAltura
            Então Caso R→Bal for:
                -1: R→Bal = 0; // MudouAltura continua Verdadeiro
                0: { R→Bal = 1; MudouAltura = Falso;};
                1: { Filho = R→Dir; // vai precisar balancear
                    Se (Filho→Bal == -1)
                        Então RotDuplaDE-Remove
                    Senão RotSimplesDD-Remove; // casos Filho→Bal = 0 e +1
                }; // fim do Caso
            // Ok e MudouAltura só são ajustadas nos casos com 0 e 1 Filhos
        } // fim - 2 Filhos

        Senão
            { // 1 Único Filho
                Se (R→Esq == Null)
                    Então R = R→Dir; // "puxa" o Filho Direito; Filho esquerdo é nulo
                Senão R = R→Esq; // "puxa" o Filho Esquerdo; Filho direito é nulo
                    DeleteNode (Aux); // desaloca o Nó
                    Ok=Verdadeiro;
                    MudouAltura=Verdadeiro;
            } // fim 1 único filho
    } // fim Remove
}
    
```

Links

1. Buricea, M. *Height Balanced Trees*. Lecture Notes. Universitatea Din Craeova. Disponível em: <http://software.ucv.ro/~mburicea/lab6ASD.pdf> (acesso em: novembro de 2013).
2. Devadas S.; Daskalakis, K.; Dzunic, Z.; Onak, K.; Schwendner, A. Singh, R. *Introduction to Algorithms*. Llecture Notes. Massachusetts Institute of Technology, 2009. Disponível em: http://courses.csail.mit.edu/6.006/fall09/lecture_notes/lecture04.pdf (acesso em: novembro de 2013).
3. Leser, U. *Algorithms and Data Structures*. AVL: Balanced Search Trees. Lecture Notes. Humboldt Universitat, Zu Berlin, 2011. Disponível em http://www.informatik.hu-berlin.de/forschung/gebiete/wbi/teaching/archive/ss11/vl_algorithmen/15_avl_trees.pdf (acesso em: novembro de 2013).

4. Rocha. V.; Vervloet, M.; Franco, A.; Andrade, C. A. *Tree Explorer*. EDGames, 2013. Disponível em: <http://edgames.dc.ufscar.br> (acesso em: setembro de 2013).

Referências e leitura adicional

- Adelson-Velskii, G. M.; Landis, E. M. *An Algorithm for the Organization of Information*. Soviet Mathematics 3 (1962), 1259-1263 - conforme citado por Drozdek (2002). p. 232 e 268.
- Camargo, H. A. *Apostila da disciplina Estruturas de Dados*. Departamento de Computação e Estatística. Universidade Federal de São Carlos, s/d.
- Drozdek, A. *Estruturas de dados e algoritmos em C++*. São Paulo: Thomson, 2002.
- Knuth, D. *The Art of Computer Programming*. V. 3 Sorting and Searching. 2. ed. Reading, MA: Addison-Wesley; 1998. p. 458-81.
- Langsam , Y; Augenstein, M. J.; Tenenbaum, A. M. *Data Structures Using C and C++*. 2nd ed. Upper Saddle River. New Jersey: Prentice Hall, 1996. p. 413-423.