

Algoritmos e Estruturas de Dados 2

Árvores Binárias Balanceadas

Prof. Fábio M. Costa

Instituto de Informática
Universidade Federal de Goiás

1o. Semestre / 2019

Introdução

Árvores de busca balanceadas

Objetivo: Garantir que as operações de busca, inserção, remoção etc. tenham complexidade $\mathcal{O}(\log_2 n)$
Ou seja, impõe-se um limite para a altura da árvore

$$h = \mathcal{O}(\log_2 n)$$

Pois, em árvores binárias em geral, a complexidade das operações é determinada pela altura h e, em árvores não-balanceadas, $h = \mathcal{O}(n)$

Estratégia simples, mas ineficiente

Reconstruir a árvore periodicamente, re-inserindo os elementos (em uma nova árvore) obedecendo uma ordem de inserção que favoreça o balanceamento. **Custo?**

Estratégia eficiente

Manter a árvore balanceada por meio de regras aplicadas durante a inserção e remoção de elementos – Árvores Red-Black, Árvores AVL



Árvores Red-Black – Introdução

Ideia geral

Além das propriedades básicas de árvores binárias de busca, a topologia da árvore está sujeita a regras baseadas na coloração dos nós, as quais definem restrições sobre as relações entre um nó e seus filhos

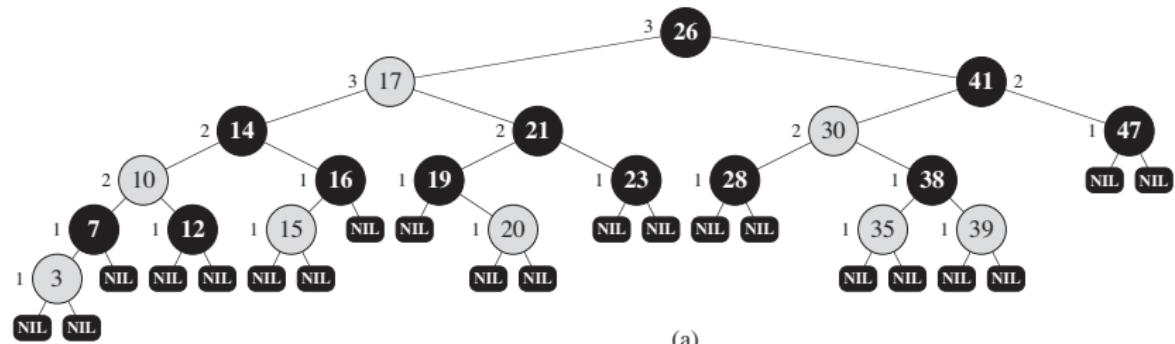
Objetivo: Limitar a altura da árvore

Propriedades de Árvores R-B (regras de平衡amento)

- ① Cada nó é preto ou vermelho
- ② A raiz é da cor preta
- ③ Cada folha (nó T.Nil) é da cor preta
- ④ Se um nó é de cor vermelha, então seus dois filhos são da cor preta
- ⑤ Para cada nó, todos os caminhos simples do nó até suas folhas descendentes contêm o mesmo número de nós de cor preta

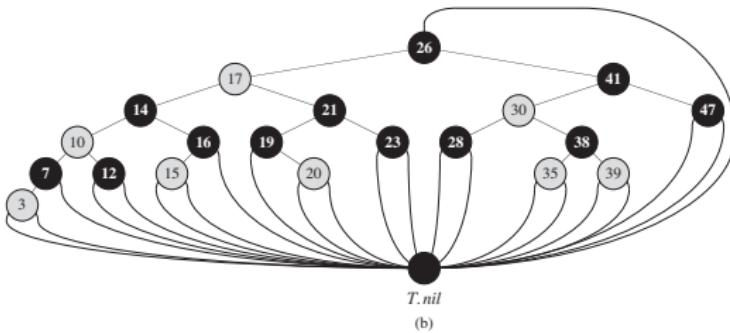
Referência: Cormen, 3a. ed., Capítulo 13

Exemplo de Árvore R-B



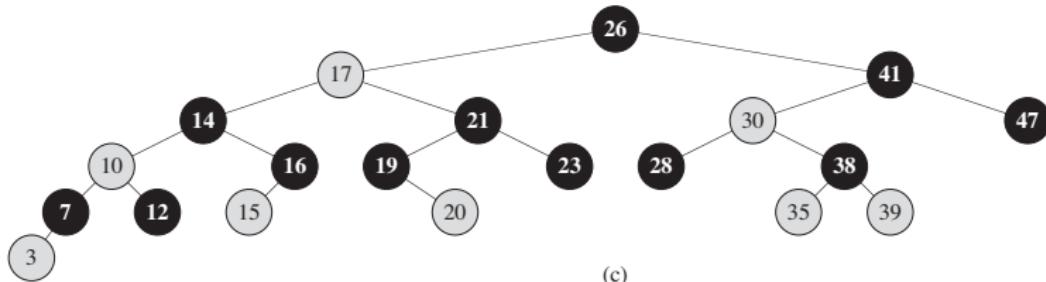
Árvores R-B: Convenções úteis para implementação

- Ponteiros para NIL são implementados como ponteiros para um **nó vazio**
- Esse nó vazio (referenciado como T.Nil) é usado como **sentinela** nos algoritmos de percurso da árvore
- Isto vale tanto para os nós folha, que passam a ter filhos NIL
- ... quanto para a raiz, que passa a ter um pai fictício (T.Nil)
- Ou seja, para todos os efeitos, as folhas são sempre nós T.Nil, e todo nó que contém dados é **nó interno**



Árvores R-B – Representação simplificada

Ao desenhar uma Árvore Red Black, o nó T.Nil (assim como os ponteiros para ele) ficam implícitos.



(c)

Efeito das Propriedades R-B

Limitar a altura h da árvore

Altura de uma árvore R-B com n chaves:

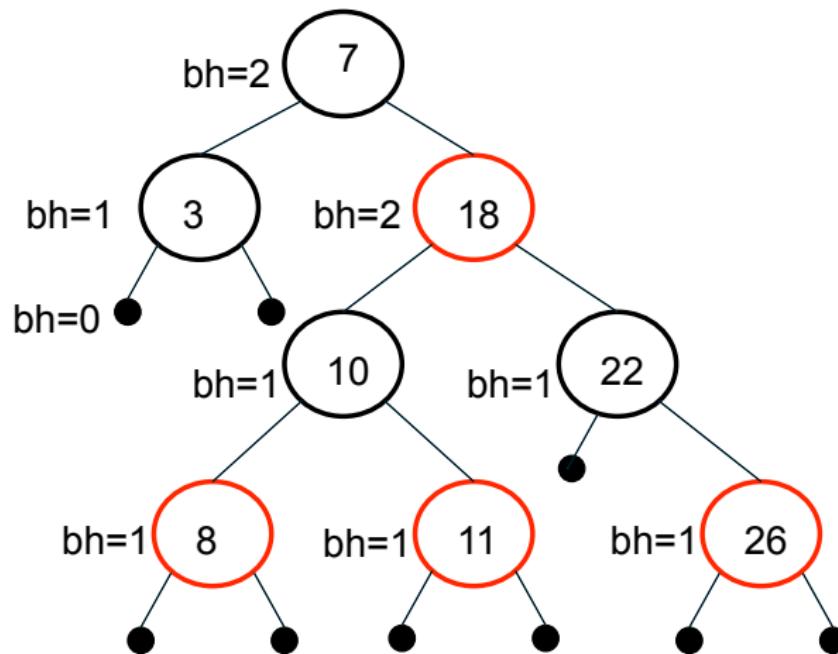
$$h \leq 2\log_2(n + 1)$$

Pela **Propriedade 5**, todos os nós da árvore têm a mesma altura se considerarmos apenas os nós da cor preta

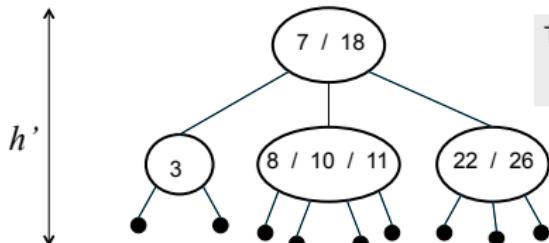
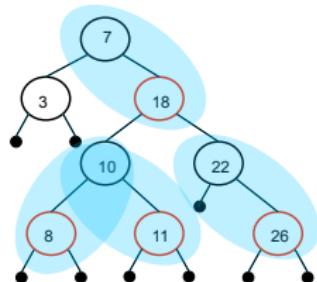
- `black_height(x)` = número de nós da cor preta em qualquer caminho simples de x até uma folha (excluindo o próprio x)
- Vamos mostrar que `black_height(raiz)` é $\log_2(n + 1)$

Pela **Propriedade 4**, em um caminho simples da raiz até uma folha, há, no máximo, um nó da cor vermelha para cada nó da cor preta. Ou seja, pelo menos a metade dos nós nesse caminho é da cor preta.

Black Height – Exemplo



Árvores R-B e Árvores 2-3-4



Árvore 2-3-4: todo nó tem entre 2 e 4 filhos

Toda árvore R-B pode ser facilmente convertida em uma árvore 2-3-4

Análise da altura de uma árvore R-B com base na altura da árvore 2-3-4 associada

- Número de folhas em uma árvore R-B com n chaves: $n + 1$
 - Isto vale também para a árvore 2-3-4 derivada da árvore R-B
- Número de folhas vs. altura em uma árvore 2-3-4 de altura h' :

$$2^{h'} \leq n_{folhas\,2-3-4} \leq 4^{h'}$$

- Ou seja, para uma certa altura h' , quanto maior o fator de ramificação (grau dos nós), maior o número de folhas
- Visto de outra forma, quanto menor o fator de ramificação de uma árvore com n chaves e $n + 1$ folhas, maior a sua altura
⇒ este é o pior caso para a altura
- Como o número de folhas de uma árvore 2-3-4 é $n + 1$:

$$2^{h'} \leq n + 1$$

$$h' \leq \log_2(n + 1)$$

Análise da altura de uma árvore R-B com base na altura da árvore 2-3-4 associada (cont.)

Pela Propriedade 4, não pode haver 2 (ou mais) nós de cor vermelha adjacentes (em um mesmo caminho)

Consequentemente: O número de nós de cor vermelha em um caminho é, no máximo, igual a metade da altura da árvore

Ou seja:

$$h \leq 2h'$$

$$h \leq 2\log_2(n + 1)$$

Consequência do limite para altura da árvore

Em uma árvore R-B, toda operação leva tempo $\mathcal{O}(\log_2 n)$

- busca, inserção, remoção, mín, máx, sucessor, predecessor etc.

Já sabíamos que era $\mathcal{O}(h)$, mas agora temos um limite para h

Para isto, basta manter válidas as propriedades R-B da árvore binária de busca

Operações em Árvores R-B

Busca, mín, máx, predecessor, sucessor:

- implementadas como em árvores binárias de busca convencionais

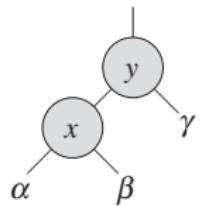
Inserção e remoção: funcionam, como em ABB, mas destroem as propriedades R-B

Ou seja, após atualizar uma árvore R-B, as propriedades R-B precisam ser restauradas!

Lembre-se: árvores de busca em geral são usadas para armazenar coleções dinâmicas de elementos – atualizações (inserções e remoções) são operações frequentes

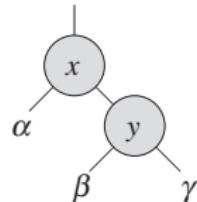
Rotação

Operação auxiliar utilizada pelo algoritmo que corrige as propriedades de uma Árvore R-B após uma inserção ou remoção.



LEFT-ROTATE(T, x)

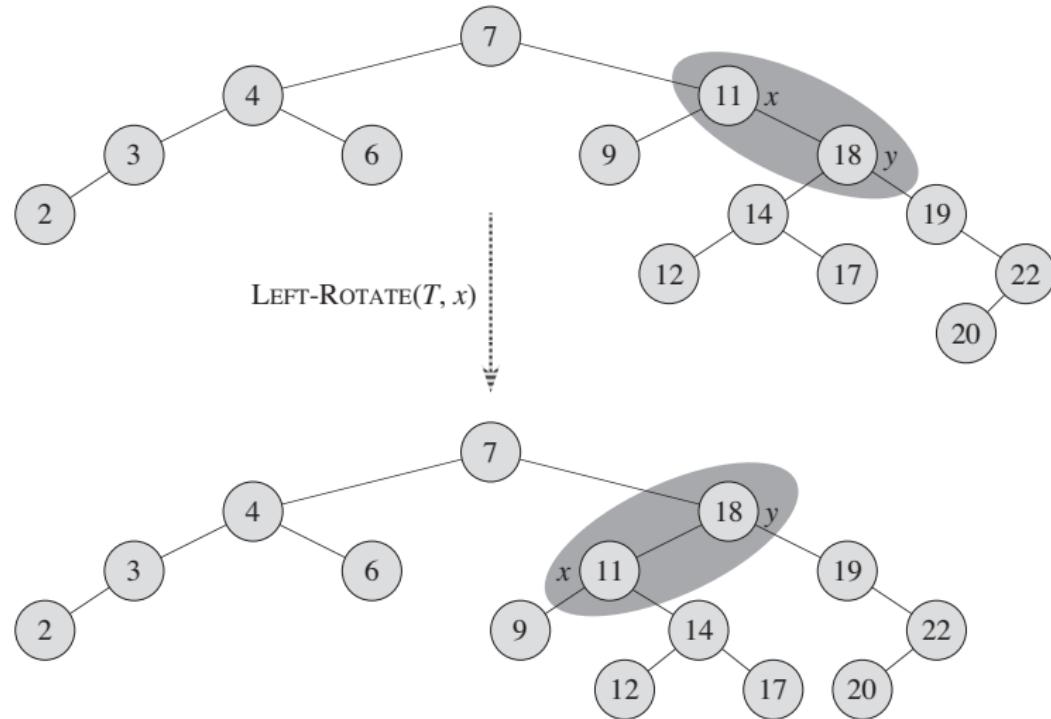
 RIGHT-ROTATE(T, y)



LEFT-ROTATE(T, x)

- 1 $y = x.right$
- 2 $x.right = y.left$
- 3 **if** $y.left \neq T.nil$
- 4 $y.left.p = x$
- 5 $y.p = x.p$
- 6 **if** $x.p == T.nil$
- 7 $T.root = y$
- 8 **elseif** $x == x.p.left$
- 9 $x.p.left = y$
- 10 **else** $x.p.right = y$
- 11 $y.left = x$
- 12 $x.p = y$

Rotação: Exemplo



Efeito da Inserção sobre as Propriedades R-B

Obs.: Nós são sempre inseridos com a cor vermelha

Que propriedades podem ser violadas?

- ① Cada nó é preto ou vermelho
 - preservada (obviamente)
- ② A raiz é da cor preta
 - pode ser violada se o novo nó for a raiz (i.e., se a árvore era inicialmente vazia)
- ③ Cada folha (nó T.Nil) é da cor preta
 - preservada: ambos os filhos do novo no? sa?o o no? sentinelas T.Nil, que é da cor preta
- ④ Se um nó é da cor vermelha, então seus dois filhos são da cor preta
 - pode ser violada se o pai do novo nó for vermelho
- ⑤ Para cada nó, todos os caminhos simples do nó até suas folhas descendentes contêm o mesmo número de nós de cor preta
 - preservada: o novo nó, que é vermelho, não altera a contagem de nós da cor preta em qualquer dos caminhos na árvore

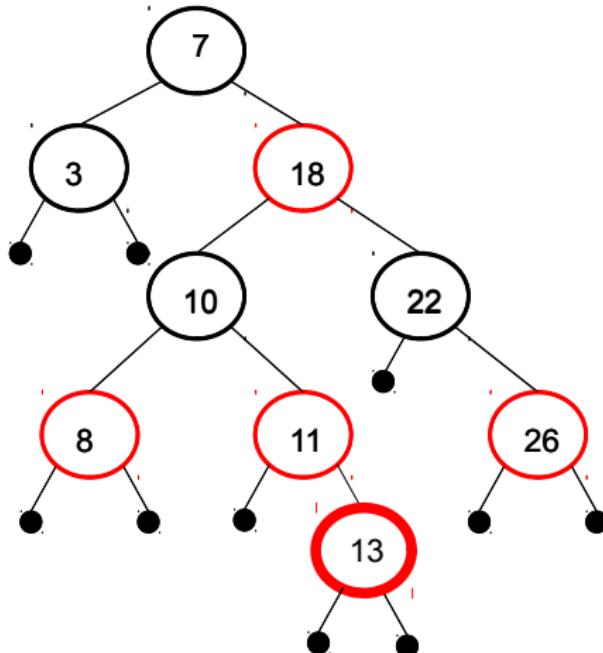
Árvores R-B: Algoritmo de Inserção

RB-INSERT(T, z)

```

1   $y = T.nil$ 
2   $x = T.root$ 
3  while  $x \neq T.nil$ 
4       $y = x$ 
5      if  $z.key < x.key$ 
6           $x = x.left$ 
7      else  $x = x.right$ 
8   $z.p = y$ 
9  if  $y == T.nil$ 
10      $T.root = z$ 
11  elseif  $z.key < y.key$ 
12       $y.left = z$ 
13  else  $y.right = z$ 
14   $z.left = T.nil$ 
15   $z.right = T.nil$ 
16   $z.color = RED$ 
17  RB-INSERT-FIXUP( $T, z$ )

```



Algoritmo para restaurar as propriedades de uma Árvore R-B após a Inserção

RB-INSERT-FIXUP(T, z)

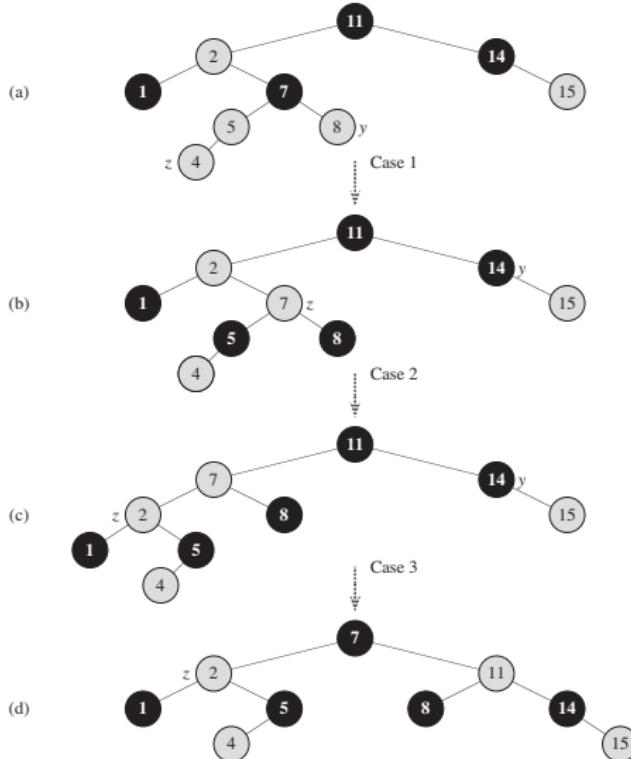
```

1  while  $z.p.color == \text{RED}$ 
2    if  $z.p == z.p.left$ 
3       $y = z.p.p.right$ 
4      if  $y.color == \text{RED}$ 
5         $z.p.color = \text{BLACK}$                                 // case 1
6         $y.color = \text{BLACK}$                                 // case 1
7         $z.p.p.color = \text{RED}$                             // case 1
8         $z = z.p.p$                                     // case 1
9      else if  $z == z.p.right$ 
10         $z = z.p$                                       // case 2
11        LEFT-ROTATE( $T, z$ )                           // case 2
12         $z.p.color = \text{BLACK}$                             // case 3
13         $z.p.p.color = \text{RED}$                             // case 3
14        RIGHT-ROTATE( $T, z.p.p$ )                      // case 3
15      else (same as then clause
           with “right” and “left” exchanged)
16     $T.root.color = \text{BLACK}$ 

```

Árvores R-B

Inserção: Exemplo

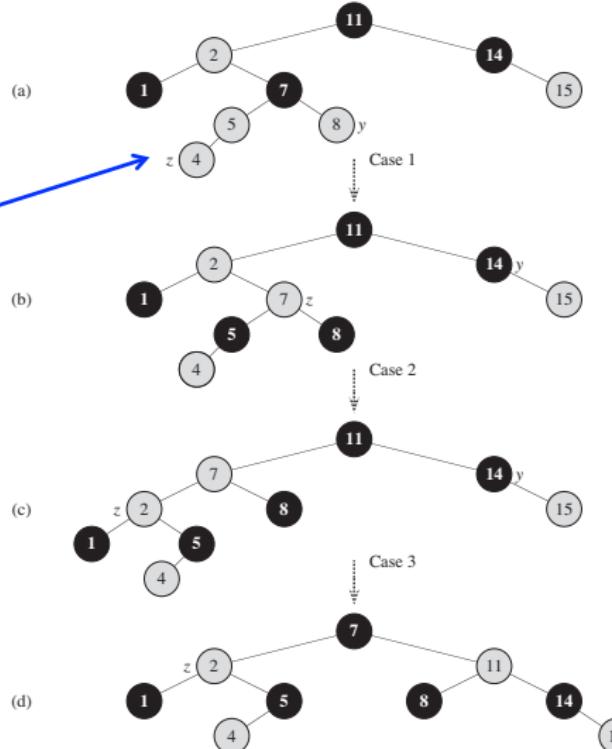


Árvores R-B

Inserção: Exemplo

A propriedade 4 é violada após a inserção do nó z

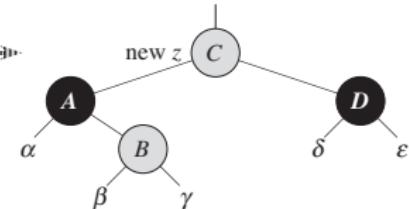
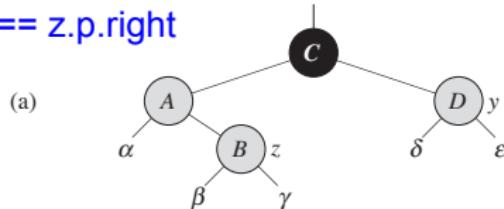
- tio de z (y) é da cor vermelha (caso 1)
- troca a cor do pai, do tio e do avô de z
- move o ponteiro z dois níveis acima (p/ o avô)



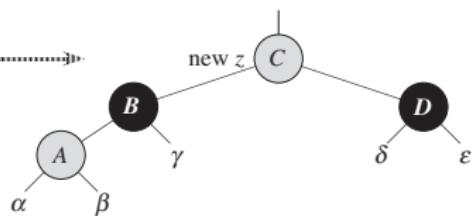
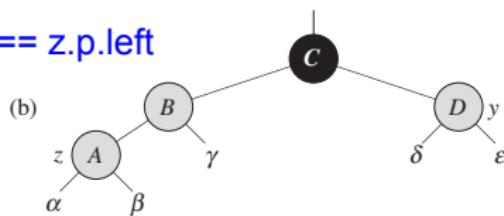
Restauração das Propriedades R-B (após inserção)

Caso 1: quando o tio de z (nó inserido) é da cor vermelha

$z == z.p.right$



$z == z.p.left$



Árvores R-B

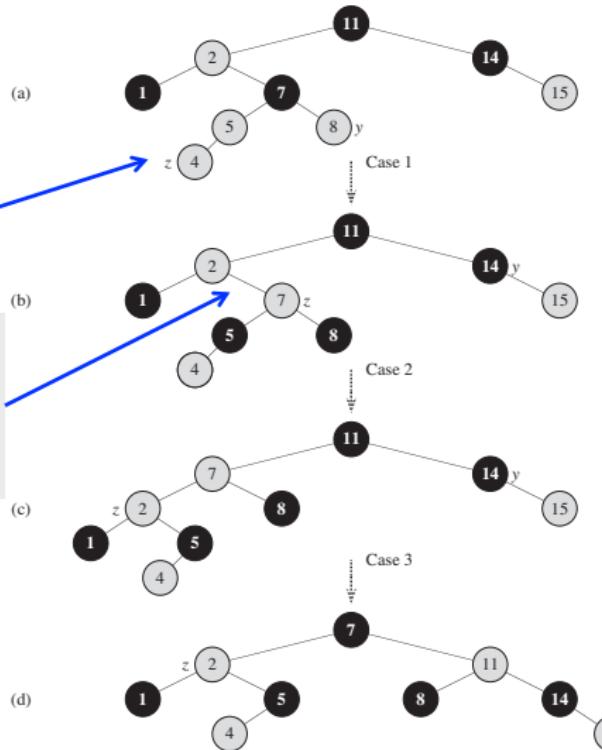
Inserção: Exemplo

A propriedade 4 é violada após a inserção do nó z

- tio de z (y) é da cor vermelha (caso 1)
- troca a cor do pai, do tio e do avô de z
- move o ponteiro z dois níveis acima (p/ o avô)

A propriedade 4 é novamente violada

- mas agora o tio de z (nó y) é da cor preta
- z é o filho da direita de z.p (caso 2)
- faz z apontar para o pai (z.p)
- faz uma rotação para a esquerda centrada no nó z

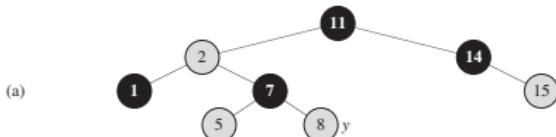


Árvores R-B

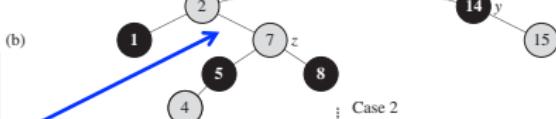
Inserção: Exemplo

A propriedade 4 é violada após a inserção do nó z

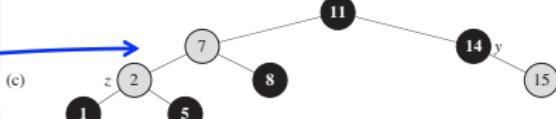
- tio de z (y) é da cor vermelha (caso 1)
- troca a cor do pai, do tio e do avô de z
- move o ponteiro z dois níveis acima (p/ o avô)



Case 1



Case 2



Case 3

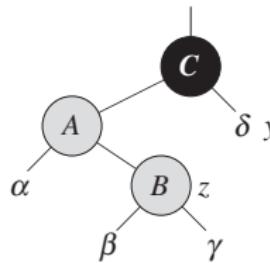


A propriedade 4 continua sendo violada

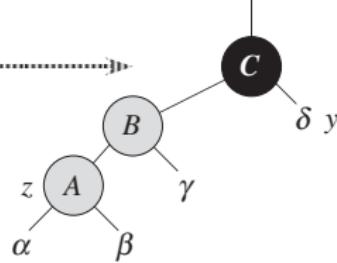
- mas agora z é o filho da esquerda (caso 3)
- troca a cor do pai de z (p/ a cor preta)
- troca a cor do do avô de z (p/ a cor vermelha)
- faz uma rotação para a direita centrada no avô de z (z.p.p)

Restauração das Propriedades R-B (após inserção)

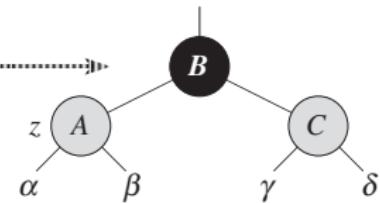
Caso 2 (não é o Caso 1 e z é filho da direita);
Caso 3 (não é o Caso 1 e z é filho da esquerda)



Case 2



Case 3

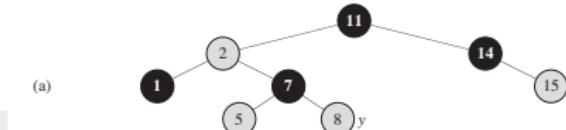


Árvores R-B

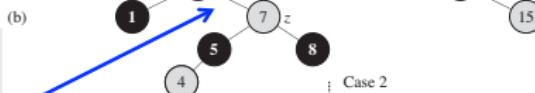
Inserção: Exemplo

A propriedade 4 é violada após a inserção do nó z

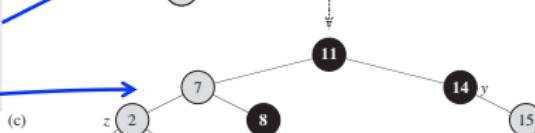
- tio de z (y) é da cor vermelha (caso 1)
- troca a cor do pai, do tio e do avô de z
- move o ponteiro z dois níveis acima (p/ o avô)



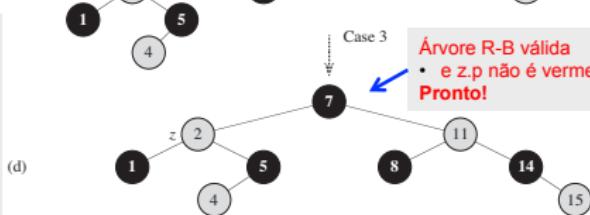
Case 1



Case 2



Case 3



Árvore R-B válida
• z.p não é vermelho
Pronto!

A propriedade 4 continua sendo violada

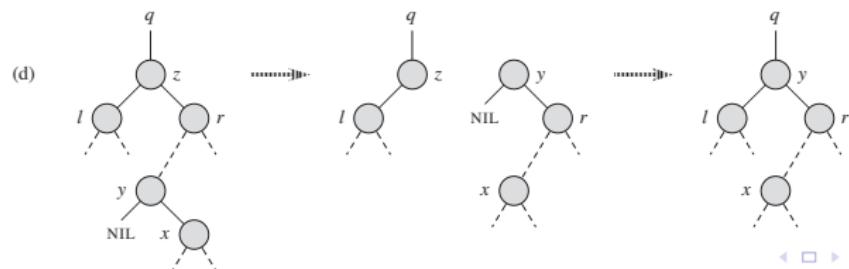
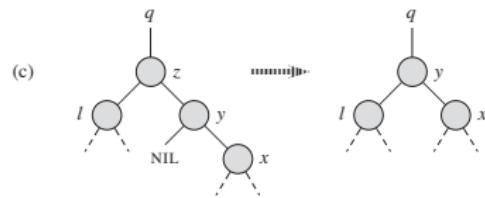
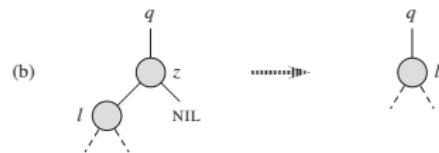
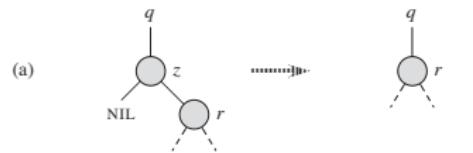
- mas agora z é o filho da esquerda (caso 3)
- troca a cor do pai de z (p/ a cor preta)
- troca a cor do do avô de z (p/ a cor vermelha)
- faz uma rotação para a direita centrada no avô de z (z.p.p)

Árvores R-B: Remoção

Ideia geral

- Primeiro remove-se o nó como em árvores binárias de busca
- Se o nó removido for vermelho, nenhuma propriedade é violada
- Se o nó removido for preto, deve-se restaurar a árvore R-B
 - trocando a cor de alguns nós
 - fazendo algumas rotações
 - **artifício:** o algoritmo considera uma cor preta extra nos caminhos afetados pela remoção – um nó ou é vermelho-preto ou tem dupla cor preta → para manter válida a propriedade 5.

Primeiro: Remoção em ABBs



Algoritmo de Remoção em ABBs

TREE-DELETE(T, z)

```

1  if  $z.left == \text{NIL}$ 
2    TRANSPLANT( $T, z, z.right$ )
3  elseif  $z.right == \text{NIL}$ 
4    TRANSPLANT( $T, z, z.left$ )
5  else  $y = \text{TREE-MINIMUM}(z.right)$ 
6    if  $y.p \neq z$ 
7      TRANSPLANT( $T, y, y.right$ )
8       $y.right = z.right$ 
9       $y.right.p = y$ 
10   TRANSPLANT( $T, z, y$ )
11    $y.left = z.left$ 
12    $y.left.p = y$ 
```

TRANSPLANT(T, u, v)

```

1  if  $u.p == \text{NIL}$ 
2     $T.root = v$ 
3  elseif  $u == u.p.left$ 
4     $u.p.left = v$ 
5  else  $u.p.right = v$ 
6  if  $v \neq \text{NIL}$ 
7     $v.p = u.p$ 
```

Algoritmo de Remoção em Árvores R-B

RB-DELETE(T, z)

```

1   $y = z$ 
2   $y\text{-original-color} = y.\text{color}$ 
3  if  $z.\text{left} == T.\text{nil}$ 
4     $x = z.\text{right}$ 
5    RB-TRANSPLANT( $T, z, z.\text{right}$ )
6  elseif  $z.\text{right} == T.\text{nil}$ 
7     $x = z.\text{left}$ 
8    RB-TRANSPLANT( $T, z, z.\text{left}$ )
9  else  $y = \text{TREE-MINIMUM}(z.\text{right})$ 
10    $y\text{-original-color} = y.\text{color}$ 
11    $x = y.\text{right}$ 
12   if  $y.p == z$ 
13      $x.p = y$ 
14   else RB-TRANSPLANT( $T, y, y.\text{right}$ )
15      $y.\text{right} = z.\text{right}$ 
16      $y.\text{right}.p = y$ 
17   RB-TRANSPLANT( $T, z, y$ )
18    $y.\text{left} = z.\text{left}$ 
19    $y.\text{left}.p = y$ 
20    $y.\text{color} = z.\text{color}$ 
21 if  $y\text{-original-color} == \text{BLACK}$ 
22   RB-DELETE-FIXUP( $T, x$ )

```

RB-TRANSPLANT(T, u, v)

```

1  if  $u.p == T.\text{nil}$ 
2     $T.\text{root} = v$ 
3  elseif  $u == u.p.\text{left}$ 
4     $u.p.\text{left} = v$ 
5  else  $u.p.\text{right} = v$ 
6   $v.p = u.p$ 

```

Algoritmo de Remoção em Árvores R-B: Restauração da Árvore

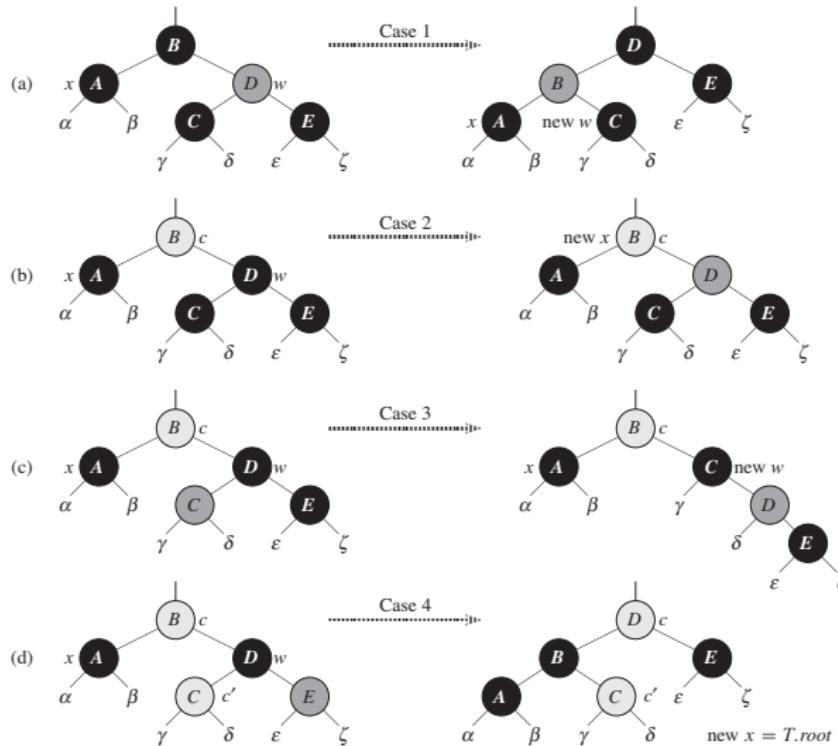
RB-DELETE-FIXUP(T, x)

```

1  while  $x \neq T.root$  and  $x.color == \text{BLACK}$ 
2      if  $x == x.p.left$ 
3           $w = x.p.right$ 
4          if  $w.color == \text{RED}$ 
5               $w.color = \text{BLACK}$                                 // case 1
6               $x.p.color = \text{RED}$                             // case 1
7              LEFT-ROTATE( $T, x.p$ )                      // case 1
8               $w = x.p.right$                             // case 1
9          if  $w.left.color == \text{BLACK}$  and  $w.right.color == \text{BLACK}$ 
10          $w.color = \text{RED}$                                 // case 2
11          $x = x.p$                                     // case 2
12     else if  $w.right.color == \text{BLACK}$ 
13          $w.left.color = \text{BLACK}$                           // case 3
14          $w.color = \text{RED}$                                 // case 3
15         RIGHT-ROTATE( $T, w$ )                         // case 3
16          $w = x.p.right$                             // case 3
17          $w.color = x.p.color$                           // case 4
18          $x.p.color = \text{BLACK}$                         // case 4
19          $w.right.color = \text{BLACK}$                      // case 4
20         LEFT-ROTATE( $T, x.p$ )                         // case 4
21          $x = T.root$                                 // case 4
22     else (same as then clause with "right" and "left" exchanged)
23      $x.color = \text{BLACK}$ 

```

Remoção em Árvores R-B: 4 Casos



Remoção em Árvores R-B: Interpretação útil e exemplos

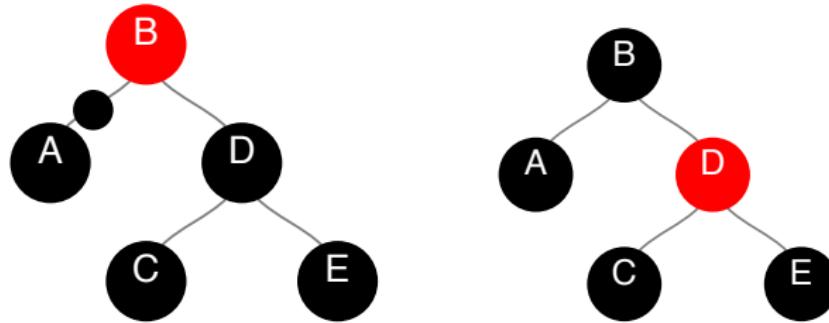
A remoção de um nó preto é tratada como o esvaziamento do nó "removido"

O nó preto vazio é (conceitualmente) mantido na árvore apenas por conta de sua cor (para evitar o desequilíbrio da *black height*) – Na prática, ele fica associado ao nó que foi transplantado em seu lugar após a remoção

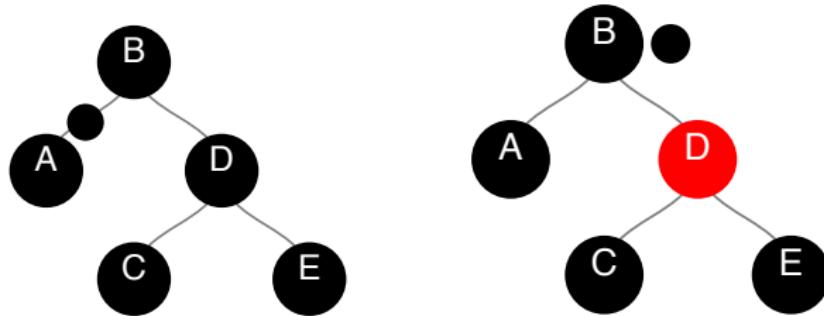
Esse nó preto vazio precisa então ser descartado:

- Se o nó preto vazio estiver associado à raiz, sua remoção não irá desequilibrar a *black height* da árvore (pois o decremento da contagem de nós pretos vai afetar todos os caminhos na árvore)
- Se o nó preto vazio estiver associado a um nó vermelho, seu descarte não irá desequilibrar a *black height*, pois o nó vermelho pode absorver a cor preta.

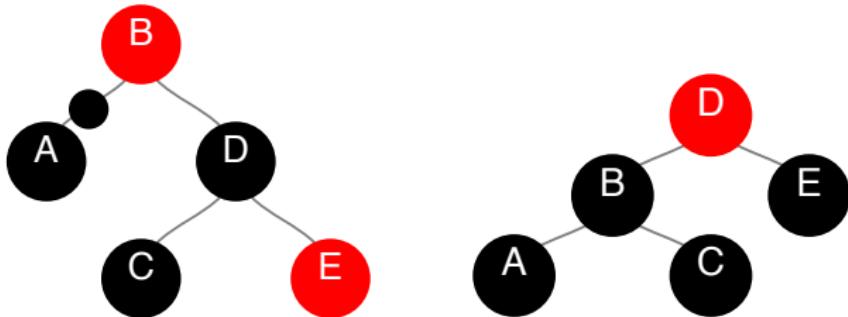
Remoção em Árvores R-B: Exemplo 1



Remoção em Árvores R-B: Exemplo 2



Remoção em Árvores R-B: Exemplo 3



As propriedades R-B foram preservadas?

- As sub-árvores de A, C e E não tiveram sua black-height alterada (pois a transformação realizada não as altera).
- Os descendentes de A, C e E continuam tendo o mesmo número de ancestrais (pais, avós etc.) da cor preta.
- Os ancestrais da raiz da sub-árvore (do pai original de B para cima) não foram alterados e, portanto, continuam tendo sua black height consistente.

Árvores AVL

Árvores AVL – Introdução

- Técnica de balanceamento de árvores binárias de busca
- A regra de balanceamento de árvores AVL opera diretamente sobre a altura da árvore
- (Diferente de árvores Red-Black, cuja regra de balanceamento trata a altura apenas indiretamente – manter as propriedades R-B, definidas com base na coloração dos nós, tem como efeito a limitação da altura da árvore.)
- O rebalanceamento da árvore (após inserções ou remoções) é feito por meio de operações locais na árvore (como em árvores R-B)
- Adelson-Velskii & Landis, 1962

Definição informal

Uma árvore AVL é uma ABB na qual as alturas das sub-árvores da esquerda e da direita da raiz diferem por no máximo 1

E as sub-árvores da esquerda e da direita são também árvores AVL

Árvores AVL – Definição Formal

- Uma árvore vazia é balanceada
- Uma árvore binária T não vazia com sub-árvores T_L e T_R é balanceada se:
 - T_L e T_R são平衡adas
 - $-1 \leq (h_R - h_L) \leq 1$, onde h_L e h_R são as alturas de T_L e T_R , respectivamente.

Fator de balanceamento

Diferença entre a altura da sub-árvore da direita e a altura da sub-árvore da esquerda, i.e.,

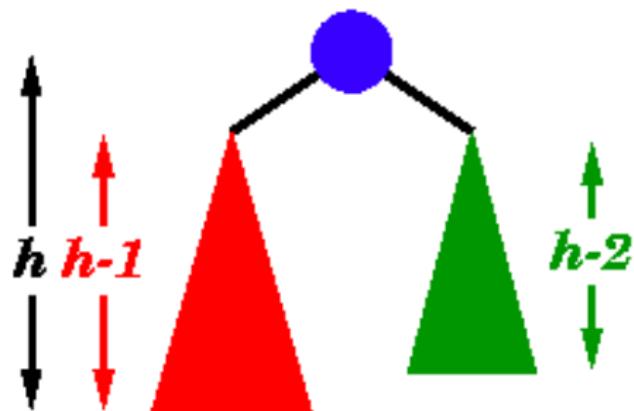
$$-1 \leq (h_R - h_L) \leq 1$$

Para uma árvore AVL, os fatores de balanceamento devem ser +1, 0 ou -1

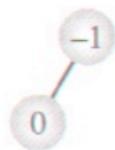
Importante

A definição de árvore AVL inclui também as técnicas para balanceamento da árvore.

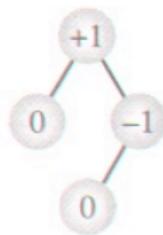
Árvores AVL – Esquema genérico



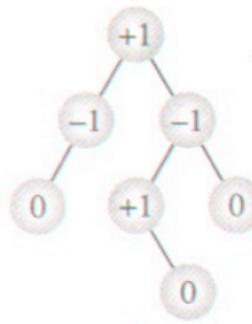
Árvores AVL: Exemplos (incluindo os fatores de balanceamento dos nós)



(a)

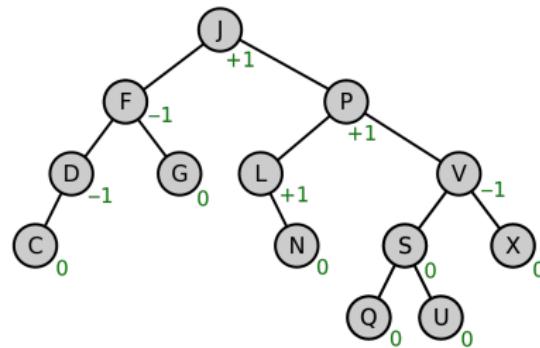


(b)



(c)

Árvores AVL: Mais um exemplo



Árvores AVL: Limite para a Altura h

Adelson-Velskii & Landis demonstraram que:

$$\log_2(n + 1) \leq h \leq 1,44\log_2(n + 2) - 0,328$$

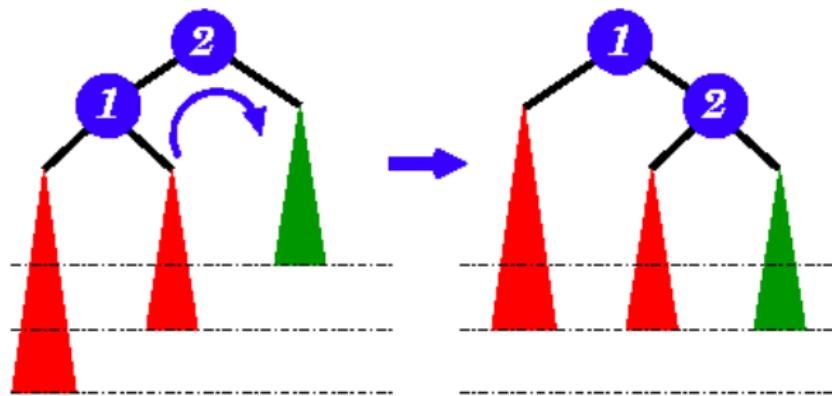
Consequentemente, o tempo de busca no pior caso é 44% pior que em árvores perfeitamente balanceadas (onde a busca é $\mathcal{O}(\log_2 n)$)

Knuth (1998) demonstrou empiricamente que, na média, o tempo de busca está muito mais próximo do melhor caso do que do pior caso:

$$h = \log_2 n + 0,25$$

(para n grande)

Árvores AVL – Inserção: Ideia geral (caso simples)



Estratégia geral

- Para cada nó: manter o fator de balanceamento
- Ao inserir um nó em uma das sub-árvores, causando um aumento do fator de balanceamento para ± 2 , fazer uma rotação pra re-equilibrar a árvore

Árvores AVL – Inserção

Algoritmo – Ideia geral

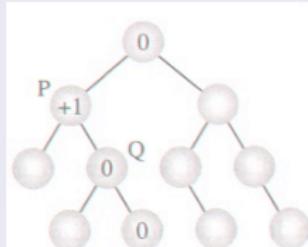
Inicialmente, insere-se o novo nó como em árvores binárias de busca.
Mas isso pode desbalancear a árvore.

Rebalanceamento: Ao inserir um nó, deve-se alterar apropriadamente os fatores de balanceamento de seus nós ascendentes

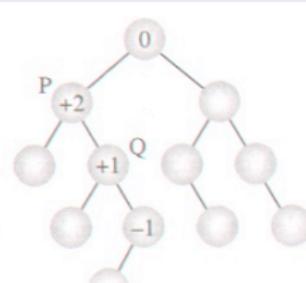
- se encontrar um nó cujo fator de平衡amento era ± 1 e muda para ± 2 , faz-se a correção do balanceamento em torno desse nó (veja abaixo) e o algoritmo termina.
- se os fatores de balanceamento de todos os ascendentes (inclusive a raiz) eram 0 inicialmente, eles são corrigidos para -1 ou +1, não gerando violações e, portanto, terminando o algoritmo.

Considerações sobre a Inserção

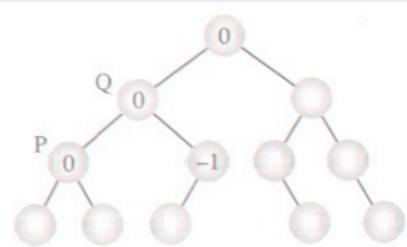
Determinação do ancestral P do nó inserido



(a)



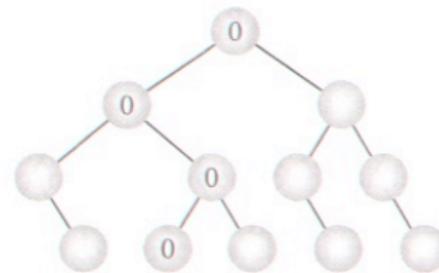
(b)



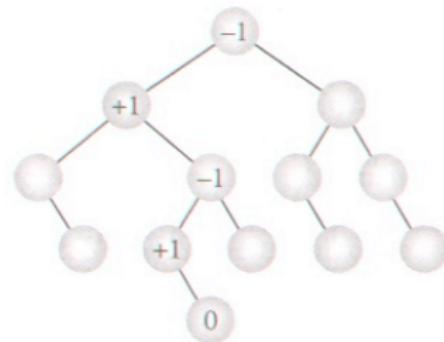
(c)

Considerações sobre a Inserção

Quando a inserção não gera violação do balanceamento



(a)



(b)

Árvores AVL – Inserção

Quatro Casos

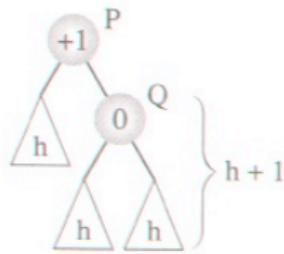
- (1) Quando um nó é inserido na sub-árvore direita do filho à direita
- (2) Quando um nó é inserido na sub-árvore esquerda do filho à direita
- (3) Quando um nó é inserido na sub-árvore direita do filho à esquerda
- (4) Quando um nó é inserido na sub-árvore esquerda do filho à esquerda

Obs.: Os casos 3 e 4 são simétricos

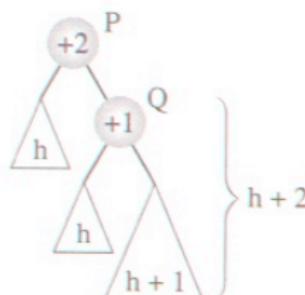
Inserção: Caso 1

Passo-a-passo

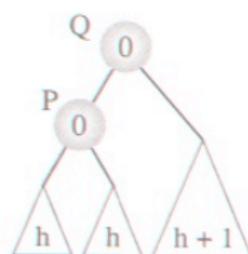
- (a) Árvore inicial
- (b) Um nó é inserido na **sub-árvore direita** de Q, que é o filho da direita do nó P, que, por sua vez é o primeiro ascendente com fator de balanceamento ± 2
- (c) Árvore resultante de uma **rotação para a esquerda** em torno de P



(a)



(b)

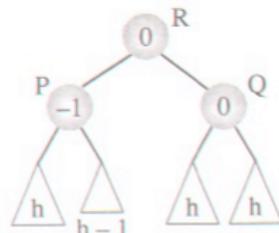
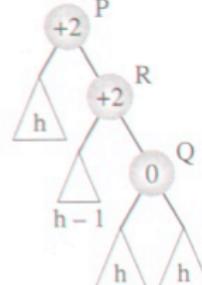
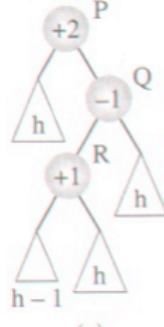
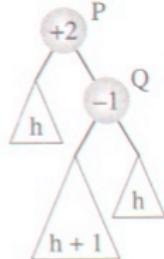
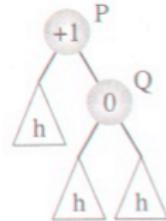


(c)

Inserção: Caso 2

Passo-a-passo

- (a) Árvore inicial
- (b) Um nó é inserido na **sub-árvore esquerda** de Q, que é o filho da direita de P, que, por sua vez é o primeiro ascendente com fator de balanceamento ± 2
- (c) Detalhamento da árvore resultante da inserção
- (d) Rotação para a **direita** de R em torno de Q
- (e) Rotação para a **esquerda** de R em torno de P



(a)

(b)

(c)

(d)

(e)

Exercícios

- 1) Descrever o passo-a-passo para inserção de um nó considerando os casos 3 e 4.
- 2) Considerando que o nó P utilizado nos exemplos anteriores pode ser parte de uma árvore AVL maior (ou seja, P pode ser filho de algum outro nó na árvore), seria necessário rebalancear os nós ascendentes de P após uma inserção?

Árvores AVL – Remoção

Algoritmo – Visão geral

Primeiro, remove-se o nó como em árvores binárias de busca.

P: Quando a remoção pode tornar a árvore desbalanceada?

Os fatores de平衡amento devem ser atualizados, **desde o nó ascendente do nó removido até a raiz**.

Para cada nó nesse caminho cujo fator de balanceamento se tornar ± 2 , realizar uma **rotação simples ou dupla** (dependendo do caso).

Diferente da inserção: o rebalanceamento não para no primeiro nó com fator de balanceamento ± 2 , ou seja, precisa subir **até a raiz**.

Ou seja: $\mathcal{O}(\log_2 n)$ rotações

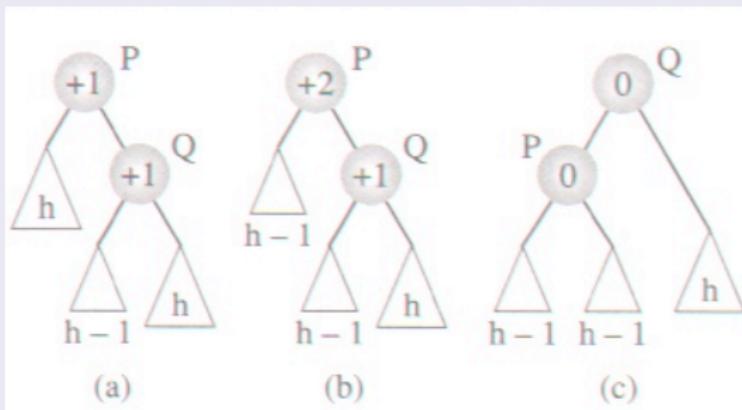
O algoritmo é organizado em torno de 8 casos (sendo 4 deles simétricos)

Árvores AVL – Remoção: Caso 1

Seja P o próximo nó ascendente cujo fator de平衡amento supera ± 1
Caracterização do Caso 1:

- O nó foi removido da sub-árvore esquerda de P (P tem fator de balanceamento +1)
- A raiz da sub-árvore direita de P (ou seja, Q) tem fator de balanceamento +1

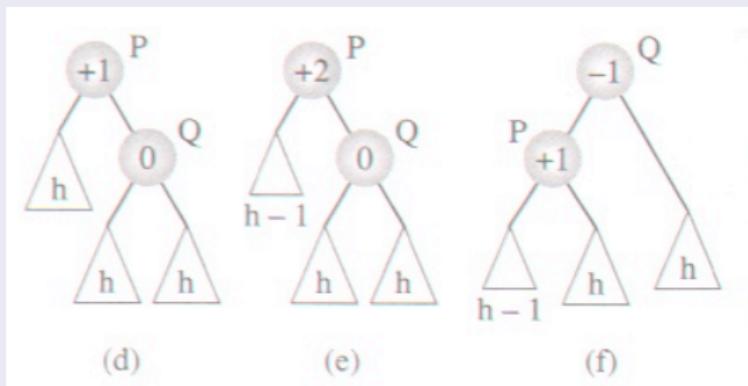
O que fazer: Rotação (para a esquerda) de Q em torno de P



Árvores AVL – Remoção: Caso 2

Caracterização: similar ao Caso 1, mas Q tem fator de balanceamento inicial igual a 0.

O que fazer: idem – rotação (para a esquerda) de Q em torno de P



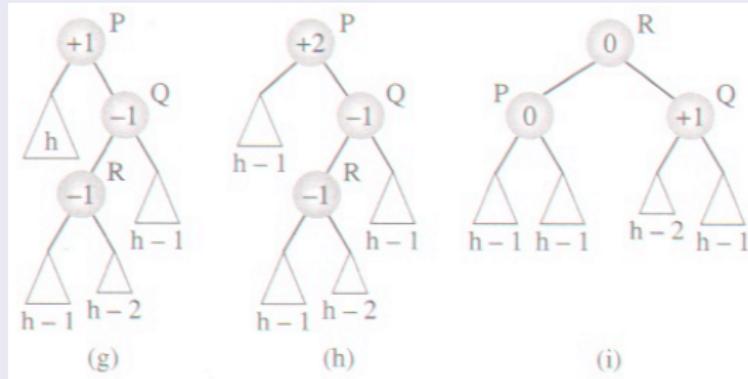
Os casos 1 e 2 podem ser implementados conjuntamente, bastando distinguir se o fator de平衡amento de Q é $+1$ ou 0 .

Árvores AVL – Remoção: Caso 3

Caracterização: a sub-árvore de Q com raiz em R tem fator de balanceamento igual a -1 .

O que fazer: rotação dupla

- Rotação (à direita) de R em torno de Q
- Rotação (à esquerda) de R em torno de P

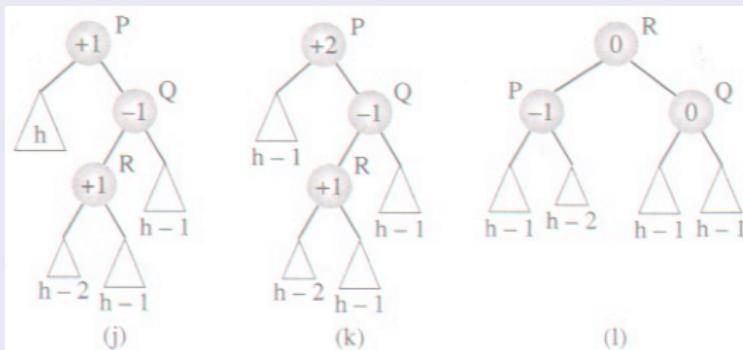


Árvores AVL – Remoção: Caso 4

Caracterização: como no Caso 3, mas o fator de balanceamento de R é igual a +1

O que fazer: idem, ou seja, rotação dupla

- Rotação (à direita) de R em torno de Q
- Rotação (à esquerda) de R em torno de P



Árvores AVL – Análise da Inserção e Remoção

Quantidade de nós visitados

- **Inserção e Remoção:** busca pelo nó a ser removido (ou pelo local de inserção): $\mathcal{O}(\log_2 n)$ nós visitados
- **Inserção:** percurso ascendente até encontrar o nó P: $\mathcal{O}(\log_2 n)$ nós visitados (se não houver desbalanceamento, precisa ir até a raiz)
- **Remoção:** sempre sobe até a raiz: $\Theta(\log_2 n)$

Quantidade de rotações realizadas

- **Inserção:** zero rotações, uma rotação simples, ou uma rotação dupla
- **Remoção:** $1,44\log_2(n + 2)$ rotações no pior caso. No caso médio: $\log_2(n) + 0,25$ (ou seja, $\mathcal{O}(\log_2 n)$ de qualquer forma)

Observação: Resultados empíricos indicam que

- 78% das remoções não resultam em desbalanceamento
- 53% das inserções não resultam em desbalanceamento

Consequentemente, remoções mais demoradas são pouco frequentes, não comprometendo a eficiência de rebalanceamento de árvores AVL.

Exercícios

- 1)** Árvores AVL podem ser estendidas permitindo diferença na altura $\Delta > 1$. Como a altura (e, portanto, o desempenho) da árvore se comporta com o aumento de Δ ? E como se comporta o número de rotações necessárias para o rebalanceamento?

- 2)** Compare árvores AVL e árvores Red-Black com relação a:
 - a)** Fator de balanceamento (pior caso e médio).
 - b)** Diferença entre a altura mínima e máxima da árvore.
 - c)** Desempenho da operação de rebalanceamento após a inserção e remoção.