

# Aula 17: Árvores AVL

David Déharbe

Programa de Pós-graduação em Sistemas e Computação

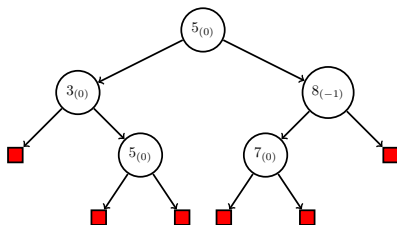
Universidade Federal do Rio Grande do Norte

Centro de Ciências Exatas e da Terra

Departamento de Informática e Matemática Aplicada

Download me from <http://DavidDeharbe.github.io>.





Introdução

Propriedades

Operações

# Árvores AVL

## Introdução

- ▶ Em 1962, Adelson-Velskii e Landis inventaram essa estrutura de dados:
  - ▶ busca em  $O(\lg n)$ ,
  - ▶ remoção em  $\Theta(\lg n)$  e
  - ▶ inserção em  $\Theta(\lg n)$
- ▶ uma **árvore AVL** é uma árvore binária de busca +restrição estrutural: altura das sub-árvores
- ▶ após inserção e remoção, a árvore pode se auto-balancear caso médio:  $\Theta(1)$ , pior caso:  $\Theta(\lg n)$
- ▶ a altura da árvore é  $\Theta(\lg \langle \text{número de nós} \rangle)$



# Árvores AVL

## Especificação

$$\begin{aligned} avl(x) \equiv & \quad bst(x) \wedge \\ & ( x = \text{NIL} \vee \\ & \quad ( |\alpha(x.left) - \alpha(x.right)| \leq 1 \wedge \\ & \quad \quad avl(x.left) \wedge avl(x.right) ) ) \end{aligned}$$

- ▶ árvore binária de busca, e
- ▶ árvore vazia, ou
- ▶ para qualquer sub-árvore, a diferença de altura entre as duas sub-árvores não ultrapassa um (1).



# Implementação

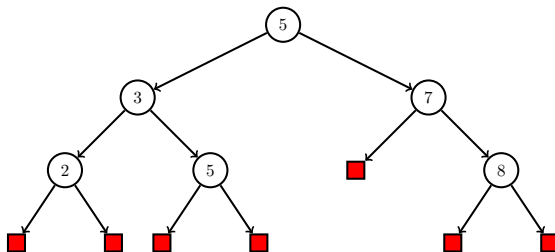
- ▶  $x.bal \in \{-1, 0, 1\}$ : balanço da árvore enraizada em  $x$
- ▶  $x.bal = \alpha(x.right) - \alpha(x.left)$



# Ilustração

Coleção: 2, 3, 5, 5, 7, 8

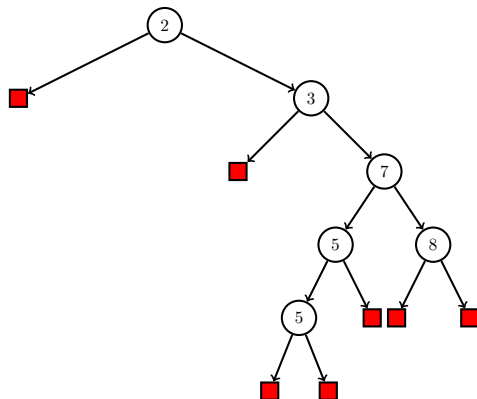
- ▶ Uma árvore AVL



# Ilustração

Coleção: 2, 3, 5, 5, 7, 8

- ▶ Uma árvore binária de busca que não é AVL



# Exercício

1. Desenhar a estrutura de árvores AVL com o menor número de nós possível com alturas de 0 até 5.
2. Como definir a relação entre a altura e o menor número de nós possível? e o maior?





# Propriedade de altura

$$\phi = \frac{1 + \sqrt{5}}{2}$$

(número de ouro)

## Teorema

*Uma árvore AVL, enraizada em  $x$ , e com  $n$  nós tem a sua altura tal que*

$$\alpha(x) \leq \log_{\phi}(\sqrt{5} \cdot (n + 2)) - 2 \approx 1,44 \ln(n + 1) - 0,328$$



# Propriedade de altura

## Demonstração.

(Esbouço) Seja  $\min$  a função que a uma dada altura  $a$  associa o número mínimo de nós em uma árvore AVL de altura  $a$ :

$$\min 0 = 0, \min 1 = 1, \min n = 1 + \min(n - 1) + \min(n - 2)$$

As propriedades da função  $\min$  tem como consequência o teorema enunciado. Não entraremos mais em detalhes sobre este assunto. □



# Operação de busca

- ▶ A busca não modifica a árvore.
- ▶ O algoritmo de busca em árvores AVL é o mesmo da busca em árvores binárias de busca qualquer.



# Operação de inserção

1. inserção em árvore binária de busca
2. *se necessário*, re-balanceamento



# Re-balanceamento

## Inserção

aparece desequilíbrio quando:

- ▶ há pelo menos um nó entre a raiz e o nó criado tal que  $|n.bal| > 1$  após a inserção
- ▶ seja  $n$  o nó de maior nível (mais próximo do nó inserido) com desequilíbrio
- ▶ antes da inserção:  $n.bal = 1$  ou  $n.bal = -1$ .
- ▶  $n$  possui pelo uma sub-árvore
  - ▶ com altura maior que a outra sub-árvore
  - ▶ logo não é vazia
- ▶ análise por caso:
  1. sub-árvore esquerda  $E$ , ou
  2. sub-árvore direita  $D$



# Inserção em $E$

## Re-balanceamento

$$\alpha(E) = \alpha(D) + 1, E \neq \text{NIL}$$

$$a : \alpha(D) \text{ (e } \alpha(E) = a + 1),$$

$n_e$  : a raiz de  $E$ ,

$EE$  : a sub-árvore esquerda de  $n_e$ ,

$ED$  : a sub-árvore direita de  $n_e$ .

Logo,  $a$  é a altura da sub-árvore mais alta entre  $EE$  e  $ED$ .

1. inserção: em  $EE$ , ou
2. inserção: em  $ED$ .



# Inserção em *EE*

## Re-balanceamento

$$\alpha(E) = \alpha(D) + 1, E \neq \text{NIL}$$

$$a : \alpha(D) \text{ (e } \alpha(E) = a + 1),$$

$n_e$  : a raiz de  $E$ ,

$EE$  : a sub-árvore esquerda de  $n_e$ ,

$ED$  : a sub-árvore direita de  $n_e$ .

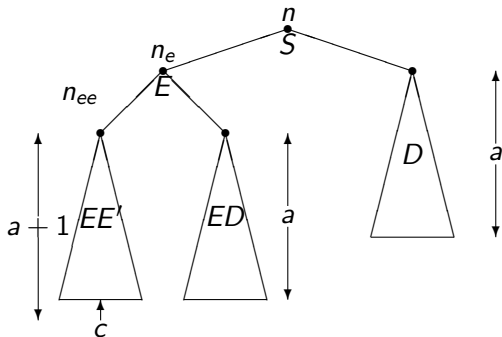
inserção em  $EE$

- ▶ altura de  $EE$  antes:  $a$
- ▶ altura de  $EE$  depois:  $a + 1$
- ▶ altura de  $ED$  antes:  $a$  ou  $a - 1$ 
  - ▶ se fosse  $a - 1$ ,  $n_e$  estaria desbalanceado
  - ▶ contradiz hipótese que  $n$  é o nó desbalanceado de maior nível
  - ▶ altura de  $ED$  é  $a$



# Inserção em $EE$ : ilustração

Re-balanceamento

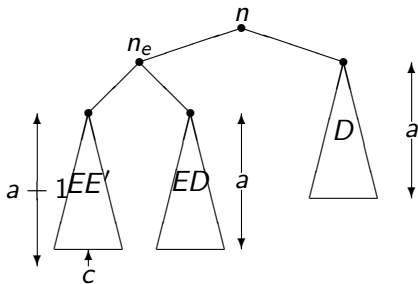


rotação simples a direita

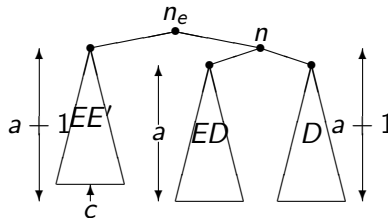


# Rotação simples a direita

## Remanejamento



Antes



Depois

# Rotação simples a direita: algoritmo

## Remanejamento

ROTATE-SIMPLE-RIGHT( $n$ )

```
1  $ne = n.left$ 
2  $n.left = ne.right$ 
3  $ne.right = n$ 
4  $ne.bal = 0$ 
5  $n.bal = 0$ 
6 return  $ne$ 
```

- ▶ o parâmetro é a raiz da sub-árvore a remanejar
- ▶ o resultado é a raiz da sub-árvore após o remanejamento

$\Theta(1)$  operações.



# Rotação simples a direita: algoritmo

## Remanejamento

ROTATE-SIMPLE-RIGHT( $n$ )

```
1   $ne = n.left$ 
2   $n.left = ne.right$ 
3   $ne.right = n$ 
4   $ne.bal = 0$ 
5   $n.bal = 0$ 
6  return  $ne$ 
```

Exercício:

1. termine o algoritmo com cálculo de  $.up$ .
2. argumente que a árvore resultante é uma árvore binária de busca



# Inserção em *ED*

## Re-balanceamento

$$\alpha(E) = \alpha(D) + 1, E \neq \text{NIL}$$

$a$  :

$n_e$  : a raiz de  $E$ ,

$EE$  a sub-árvore esquerda de  $n_e$ ,

$ED$  a sub-árvore direita de  $n_e$ .

inserção em *ED*

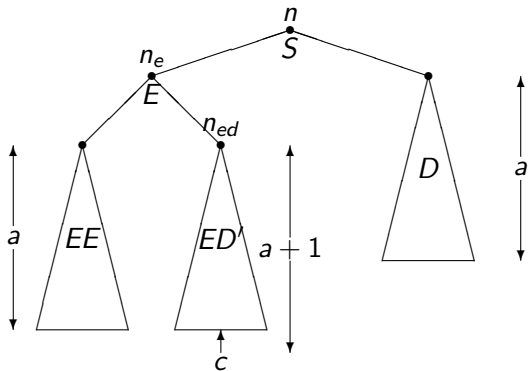
$$\alpha(D) \text{ (e } \alpha(E) = a + 1),$$

- ▶ altura de *ED* antes:  $a$
- ▶ altura de *ED* depois:  $a + 1$
- ▶ altura de *EE* antes:  $a$  ou  $a - 1$ 
  - ▶ se fosse  $a - 1$ ,  $n_e$  estaria desbalanceado
  - ▶ contradiz hipótese que  $n$  é o nó desbalanceado de maior nível
  - ▶ altura de *EE* é  $a$



# Inserção em $ED$

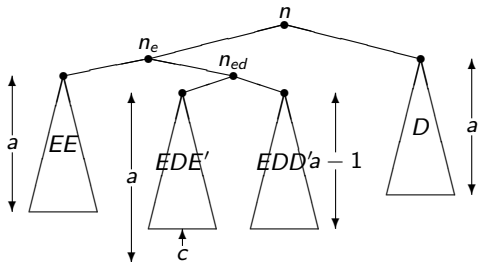
Re-balanceamento



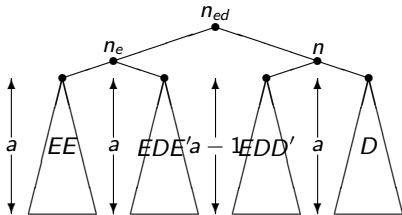
rotação dupla a direita

# Rotação dupla a direita

## Remanejamento



antes



depois

# Rotação dupla a direita: algoritmo

## Remanejamento

ROTATE-DOUBLE-RIGHT( $n$ )

$ne = n.left$

$ned = ne.right$

$n.left = ned.right$

$ne.right = ned.left$

$ned.left = ne$

$ned.right = n$

**if**  $ned.bal == 1$

$n.bal = 0$

$ne.bal = -1$

**else if**  $ned.bal == 0$

$n.bal = 0$

$ne.bal = 0$

**else**

$n.bal = 1$

$ne.bal = 0$

$ned.bal = 0$

**return**  $ned$

- ▶ o parâmetro é a raiz da sub-árvore a remanejar
- ▶ o resultado é a raiz da sub-árvore após o remanejamento

$\Theta(1)$  operações.



# Rotação dupla a direita: algoritmo

## Remanejamento

ROTATE-DOUBLE-RIGHT( $n$ )

$ne = n.left$

$ned = ne.right$

$n.left = ned.right$

$ne.right = ned.left$

$ned.left = ne$

$ned.right = n$

**if**  $ned.bal == 1$

$n.bal = 0$

$ne.bal = -1$

**else if**  $ned.bal == 0$

$n.bal = 0$

$ne.bal = 0$

**else**

$n.bal = 1$

$ne.bal = 0$

$ned.bal = 0$

**return**  $ned$

Exercício:

1. termine o algoritmo com cálculo de  $.up$ .
2. argumente que a árvore resultante é uma árvore binária de busca





# Inserção em $D$

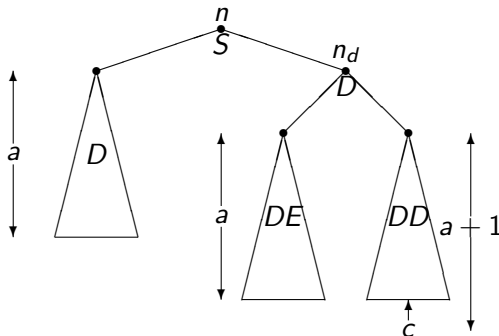
## Re-balanceamento

- ▶ A situação é simétrica à inserção em  $E$ .
- ▶ A análise é similar.
- ▶ O tratamento é simétrico, com 2 sub-casos ( $DD$  e  $DE$ ).



# Inserção em $DD$ : ilustração

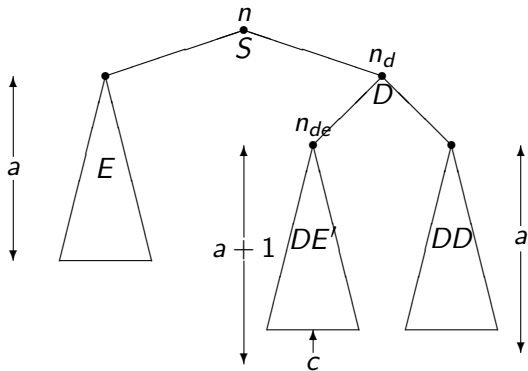
Re-balanceamento



rotação simples a esquerda

# Inserção em $DE$

Re-balanceamento



rotação dupla a esquerda

## Inserção: algoritmo

- $n$  : raiz da sub-árvore onde a inserção ocorrerá (ref)
- $v$  : valor a inserir
- Booleano indicando se a altura aumentou

INSERT( $n, v$ )

```
1  if  $n == \text{NIL}$ 
2       $n = \text{MAKE-AVL-NODE}(v, \text{NIL}, \text{NIL}, 0)$ 
3      return TRUE
4  if  $n.\text{val} == v$ 
5      return FALSE
6  if  $v < n.\text{val}$ 
7       $\text{inc} = \text{INSERT}(n.\text{left}, v)$ 
8      if  $\text{inc}$ 
9           $\langle \text{tratamento de incremento de altura na sub-árvore} \rangle$ 
10 else ...
```



## Inserção: algoritmo

$n$  : raiz da sub-árvore onde a inserção ocorrerá (ref)  
 $v$  : valor a inserir  
→ Booleano indicando se a altura aumentou

```
1  if inc
2      if  $n.bal == 0$ 
3           $n.bal = -1$ 
4          return TRUE
5      elseif  $n.bal == +1$ 
6           $n.bal = 0$ 
7          return FALSE
8      else
9          if  $n.left.bal == -1$ 
10              $n = \text{ROTATE-SIMPLE-RIGHT}(n)$ 
11             else  $n = \text{ROTATE-DOUBLE-RIGHT}(n)$ 
12             return FALSE
13 else ⟨Inserção na sub-árvore direita⟩
```

# Inserção: algoritmo

- $n$  : raiz da sub-árvore onde a inserção ocorrerá (ref)
- $v$  : valor a inserir
- Booleano indicando se a altura aumentou

## Exercícios:

1. escrever o tratamento correspondente a  $v > n.val$ .
2. verifique que, quando há remanejamento, a altura da árvore após a inserção é a mesma da altura antes da inserção.



# Remoção

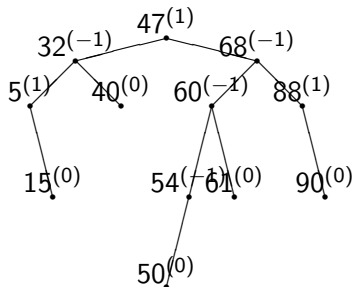
## Introdução

- ▶ remove valor como em árvore binária de busca
- ▶ reequilibrar a árvore para reestabelecer a propriedade AVL
- ▶ mais de uma rotação pode ser necessária  
 $O(\lg n)$  no pior caso

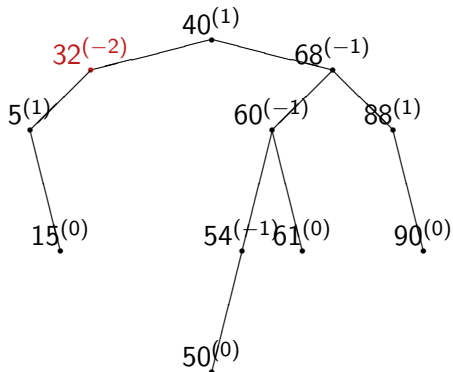


# Remoção

## Exemplo



(1) árvore inicial

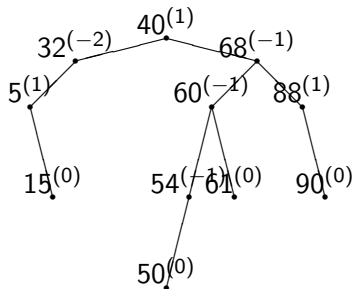


(2) após remoção de 47

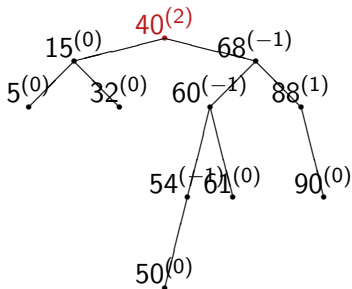


# Remoção

## Exemplo



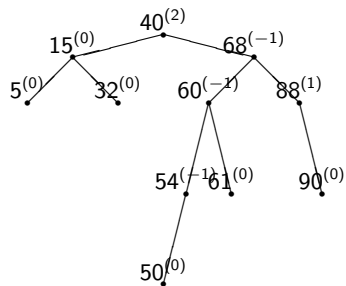
após remoção de 47



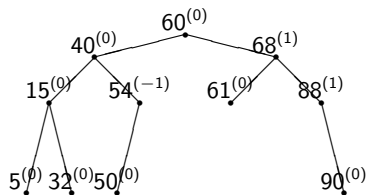
após rotação dupla a direita de 32

# Remoção

## Exemplo



rotação dupla a direita de 32



rotação dupla a esquerda de 40

# Algoritmo de remoção

## Principais etapas

Entradas: a raiz  $n$  de uma árvore AVL, um valor  $v$

1. encontrar o valor  $v$  em  $n$
2. se a busca for mal sucedida: término
3. remover o nó  $m$  com o valor  $v$ 
  - ▶ nó folha: remover nó  $m$
  - ▶ nó interno com uma sub-árvore vazia: remover nó  $m$
  - ▶ nó interno sem sub-árvore vazia: remover nó com valor mínimo  $v'$  da sub-árvore direita, substituir  $v$  por  $v'$  em  $m$ .
4. possivelmente reequilibrar a árvore



# Algoritmo de remoção

## Principais etapas

Entradas: a raiz  $n$  de uma árvore AVL, um valor  $v$

1. encontrar o valor  $v$  em  $n$
2. se a busca for mal sucedida: término
3. remover o nó  $m$  com o valor  $v$ 
  - ▶ nó folha: remover nó  $m$
  - ▶ nó interno com uma sub-árvore vazia: remover nó  $m$
  - ▶ nó interno sem sub-árvore vazia: remover nó com valor mínimo  $v'$  da sub-árvore direita, substituir  $v$  por  $v'$  em  $m$ .
4. possivelmente reequilibrar a árvore

algoritmos auxiliares:

- ▶ REMOVE-MIN
- ▶ BALANCE-RIGHT, BALANCE-LEFT: rebalanceamento da sub-árvore direita, esquerda



# Remoção

- $n$  : raiz da sub-árvore cujo valor mínimo deve ser removido (**ref**)
- $v$  : valor a remover
- Booleano indicando se a altura diminuiu



# Remoção

```
REMOVE(n, v)
1  if n == NIL
2      return FALSE
3  if v < n.val
4      return BALANCE-LEFT(n, REMOVE(n.left))
5  if v > n.val
6      return BALANCE-RIGHT(n, REMOVE(n.right))
7  if n.left == NIL
8      tmp = n
9      n = n.right
10     FREE-AVL-NODE(tmp)
11     return TRUE
12 if n.right == NIL
13     tmp = n
14     n = n.left
15     FREE-AVL-NODE(tmp)
16     return TRUE
17 return BALANCE-RIGHT(n, REMOVE-MIN(n.right, n.val))
```



## Remoção do nó com valor mínimo

- $n$  : raiz da sub-árvore cujo valor mínimo deve ser removido (ref)
- $v$  : variável onde o valor mínimo será armazenado (ref)
- Booleano indicando se a altura diminuiu

REMOVE-MIN( $n, v$ )

```
1  if  $n.left == \text{NIL}$ 
2       $v = n.val$ 
3       $tmp = n$ 
4       $n = n.right$ 
5      FREE-AVL-NODE( $tmp$ )
6      return TRUE
7  else
8      return BALANCE-LEFT( $n, \text{REMOVE-MIN}(n.left, v)$ )
```



## Rebalanceamento da sub-árvore direita

- $n$  : a sub-árvore direita de  $n$  sofreu remoção (ref)  
 $dec$  : indica se a altura diminuiu na remoção na sub-árvore  
→ indica se a altura diminuiu na remoção em  $n$

BALANCE-RIGHT( $n, dec$ )

```
1  if not  $dec$ 
2      return FALSE
3  if  $n.bal == -1$            // desequilíbrio em  $n$ 
4      if  $n.left.bal == -1$  or  $n.left.bal == 0$ 
5           $n = ROTATE-SIMPLE-RIGHT(n)$ 
6      else  $n = ROTATE-DOUBLE-RIGHT(n)$ 
7      return TRUE
8  elseif  $n.bal == 0$ 
9       $n.bal = -1$ 
10     return FALSE
11 else  $n.bal = 0$ 
12     return TRUE
```





1. Escrever o algoritmo `BALANCE-LEFT`
2. Aplicar repetidamente o algoritmo de remoção à árvore inicial do exemplo até ter removido todos os valores da árvore:
  - ▶ em ordem decrescente de valor
  - ▶ em ordem crescente de valor
  - ▶ escolhendo sempre o valor mediano da coleção representada