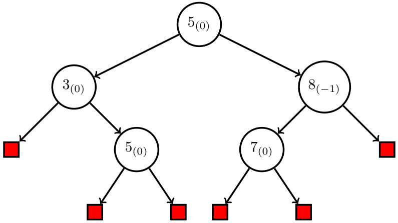
Aula 17: Árvores AVL

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`.

Plano da aula



Introdução

Propriedades

Operações

- 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 $\Theta(\lg\langle\text{número de nós}\rangle)$

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

- 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).

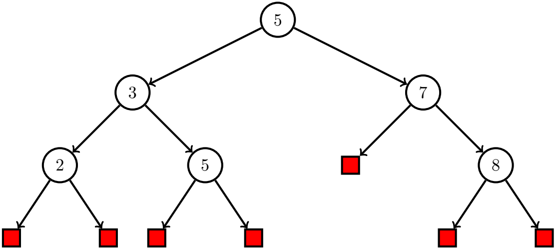
Implementação

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

Ilustração

Coleção: $2, 3, 5, 5, 7, 8$

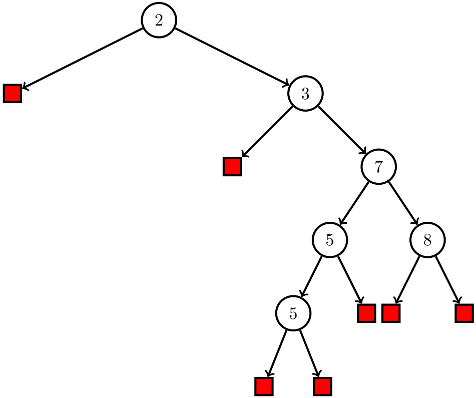
- Uma árvore AVL



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?

PPgSC

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

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

PPgSC

Demonstração.
(Esboço) 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

aparece desequilibro quando:

- há pelo menos um nó entre a raiz e o nó criado tal $|n.bal| > 1$ após a inserção
- seja n o nó de maior nível (mais próximo do nó inserido) com desequilibro
- 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

$$
\begin{aligned}
&\alpha(E) = \alpha(D) + 1,\ E \neq \textsc{Nil} \\
&\quad a \quad : \quad \alpha(D)\ (\text{e } \alpha(E) = a + 1), \\
&\quad n_e \quad : \quad \text{a raiz de } E, \\
&\quad EE \quad : \quad \text{a sub-árvore esquerda de } n_e, \\
&\quad ED \quad : \quad \text{a sub-árvore direita de } n_e.
\end{aligned}
$$

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

$$\alpha(E) = \alpha(D) + 1,\ E \neq \textsc{Nil}$$

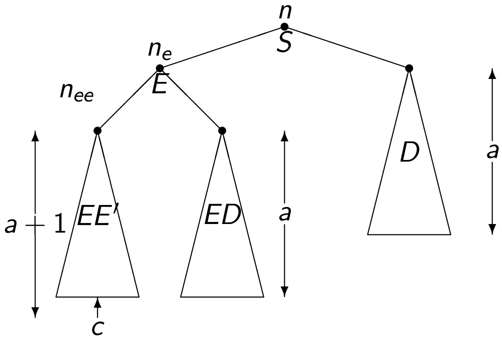
$$\begin{array}{rcl} a & : & \alpha(D) \text{ (e } \alpha(E) = a + 1), \\ n_e & : & \text{a raiz de } E, \\ EE & : & \text{a sub-árvore esquerda de } n_e, \\ ED & : & \text{a sub-árvore direita de } n_e. \end{array}$$

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

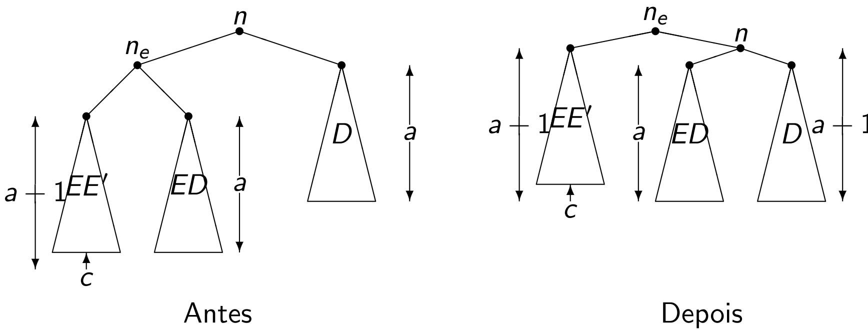
Re-balanceamento



rotação simples a direita

Rotação simples a direita

Remanejamento



ROTATE-SIMPLE-RIGHT(n)

```
ne = n.left
n.left = ne.right
ne.right = n
ne.bal = 0
n.bal = 0
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)

$ne = n.left$
$n.left = ne.right$
$ne.right = n$
$ne.bal = 0$
$n.bal = 0$
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

$\alpha(E) = \alpha(D) + 1,\ E \neq \textsc{Nil}$

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

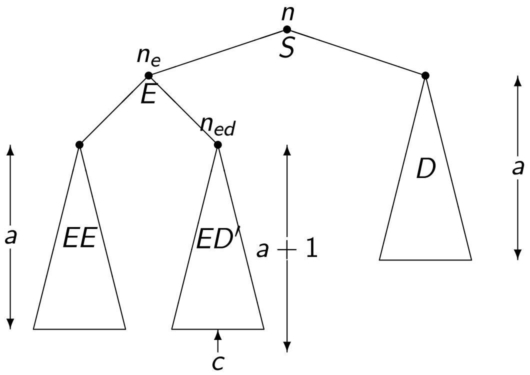
n_e : a raiz de E,

EE a sub-árvore esquerda de n_e,

ED a sub-árvore direita de n_e.

inserção em ED

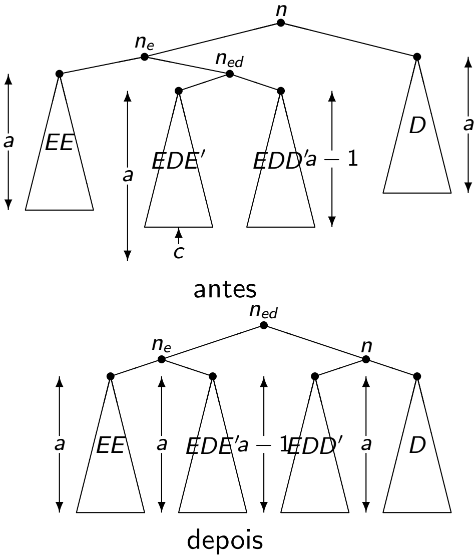
- altura de ED antes: a
- altura de ED depois: $a + 1$
- altura de EE antes: a ou $a - 1$
  - se fosse $a - 1$, n_e estaria desbalanceado
  - contradiz hipótese que n é o nó desbalanceado de maior nível
  - altura de EE é a



rotação dupla a direita

Rotação dupla a direita

Remanejamento



$\text{ROTATE-DOUBLE-RIGHT}(n)$

$ne = n.left$
$ned = ne.right$
$n.left = ned.right$
$ne.right = ned.left$
$ned.left = ne$
$ned.right = n$

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

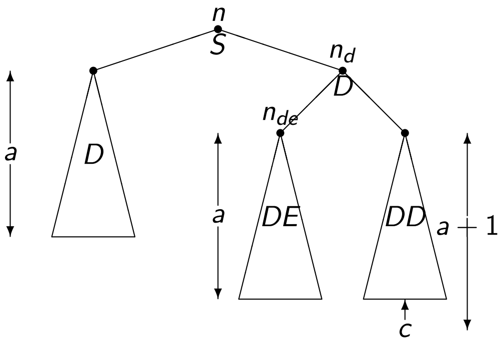
Re-balanceamento

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

PPgSC

Inserção em DD: ilustração

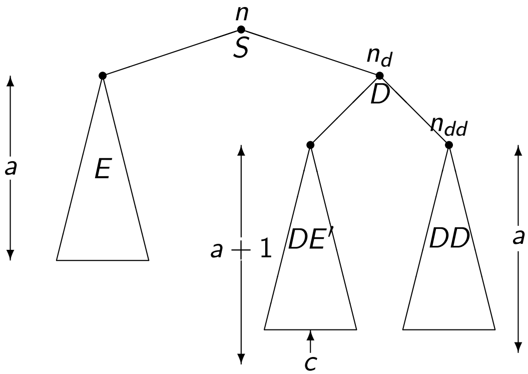
Re-balanceamento



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
	$\rightarrow$	Booleano indicando se a altura aumentou

```
INSERT(n, v)
if n == NIL
    n = MAKE-AVL-NODE(v, NIL, NIL, 0)
    return TRUE
if n.val == v
    return FALSE
if v < n.val
    inc = INSERT(n.left, v)
    if inc
    ⟨tratamento de incremento de altura na sub-árvore ⟩
else ···
```

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

```
if inc
    if n.bal == 0
        n.bal = −1
        return TRUE
    elseif n.bal == 11
        n.bal = 0
        return FALSE
    else
        if n.left.bal == −1
            n = ROTATE-SIMPLE-RIGHT(n)
        else n = ROTATE-DOUBLE-RIGHT(n)
        return FALSE
else ⟨Inserção na sub-árvore direita⟩
```

Inserção: algoritmo

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

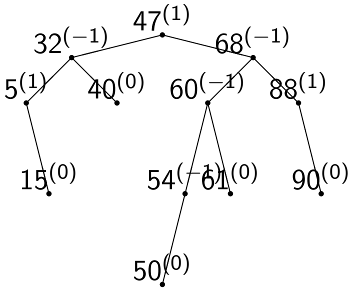
Exercícios:

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.

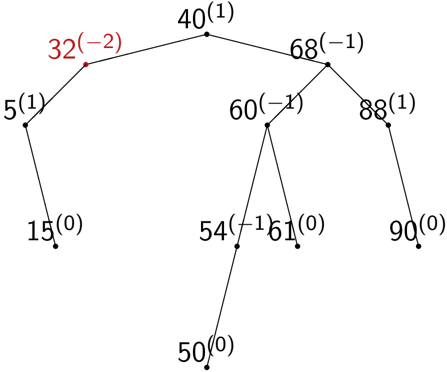
- remove valor como em árvore binária de busca
- reequilibrar a árvore para reestabelecer a propriedade AVL
- mais de uma rotação pode ser necessária
  $O(\lg n)$ no pior caso

Remoção

Exemplo



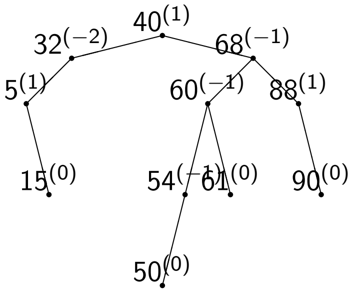
(1) árvore inicial



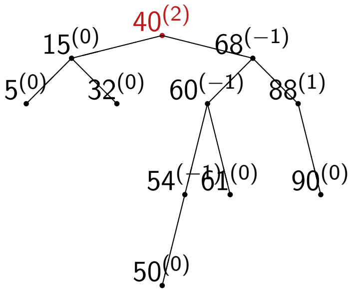
(2) após remoção de 47

Remoção

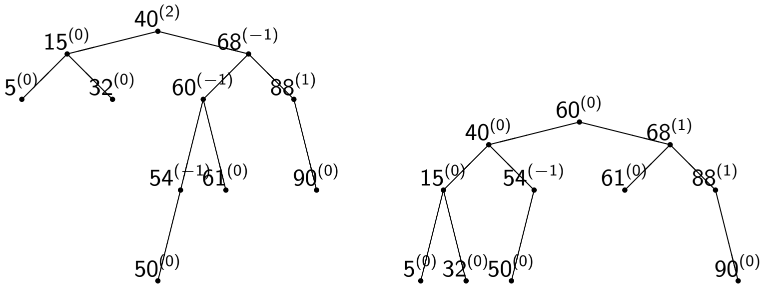
Exemplo



após remoção de 47



após rotação dupla a direita de 32



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

PPgSC

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
	$\rightarrow$	Booleano indicando se a altura diminuiu

PPgSC

Remoção

```
REMOVE(n, v)
if n == NIL
    return FALSE
if v < n.val
    return BALANCE-LEFT(n, REMOVE(n.left))
if v > n.val
    return BALANCE-RIGHT(n, REMOVE(n.right))
if n.left == NIL
    tmp = n
    n = n.right
    FREE-AVL-NODE(tmp)
    return TRUE
if n.right == NIL
    tmp = n
    n = n.left
    FREE-AVL-NODE(tmp)
    return TRUE
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)
	$\rightarrow$	Booleano indicando se a altura diminuiu

```
REMOVE-MIN(n, v)
if n.left == NIL
    v = n.val
    tmp = n
    n = n.right
    FREE-AVL-NODE(tmp)
    return TRUE
else
    return BALANCE-LEFT(n, REMOVE-MIN(n.left, v), n)
```

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
	$\rightarrow$	indica se a altura diminuiu na remoção em n

```
BALANCE-RIGHT(n, dec)
if not dec
    return FALSE
if n.bal == −1                  // desequilibro em n
    if n.left.bal == −1 or n.left.bal == 0
        n = ROTATE-SIMPLE-RIGHT(n)
    else n = ROTATE-DOUBLE-RIGHT(n)
    return TRUE
elseif n.bal == 0
    n.bal = −1
    return FALSE
else n.bal = 0
    return TRUE
```

Exercícios

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

PPgSC