# Árvores Balanceadas

Daniel Ventura
INF/UFG

2024/02

## Aula anterior

- Operações sobre árvores binárias ordenadas:
  1. pesquisa;
  2. inserção;
  3. remoção.
- Estas operações são mais eficientes sobre árvores balanceadas (menor altura).
- A inserção e remoção preservam a ordenação mas não preservam o balanceamento.

## Nesta aula

Vamos ver árvores binárias que preservam as duas propriedades:

ordenação: o valor em cada nó é maior que os valores à esquerda e menor que os valores à direita;

balanceamento: em cada nó a altura das sub-árvores esquerda e direita difere 1 no máximo.

Essa estrutura de dados é chamada de árvores de pesquisa auto-balanceada.

## Árvores AVL I

- Primeiras árvores de pesquisa auto-balanceadas (Adelson-Velskii e Landis, 1962).
- Mantêm automaticamente as propriedades de ordenação e balanceamento.
- A pesquisa é efetuada como anteriormente.
- Após cada inserção ou remoção efetuamos rotações da árvore para re-establecer o balanceamento (se necessário).

Vamos seguir a apresentação no capítulo 9 em:

Richard Bird, Philip Wadler, *"Introduction to Functional Programming"*, Prentice-Hall, 1988

## Árvores AVL II

A declaração de tipo é idêntica às árvores de pesquisa simples.

```
data Arv a = No a (Arv a) (Arv a)
           | Vazia
```

## Árvores AVL III

Necessitamos de funções auxiliares para calcular a altura e o desvio de uma árvore (a diferença entre a altura esquerda e direita).

```
altura :: Arv a -> Int
altura Vazia = 0
altura (No _ esq dir) = 1 + max (altura esq) (altura dir)

desvio :: Arv a -> Int
desvio Vazia = 0
desvio (No _ esq dir) = altura esq - altura dir
```

## Propriedade AVL

### Propriedade AVL

Para cada sub-árvore duma árvore AVL, o desvio só pode ser 1, 0 ou $-1$.

Esta propriedade é invariante:

- assumimos que é válida *antes* de qualquer operação;
- vamos garantir que é preservada *após* a operação.

## Árvores AVL: pesquisa

A pesquisa é feita exatamente como no caso de árvores simples.

```
pesquisaAVL :: Ord a => a -> Arv a -> Bool
pesquisaAVL x Vazia = False
pesquisaAVL x (No y esq dir)
    | x==y = True
    | x<y  = pesquisaAVL x esq
    | x>y  = pesquisaAVL x dir
```

Como a árvore não é modificada, a propriedade AVL é mantida trivialmente.

# Árvores AVL: inserção

A inserção dum valor numa árvore binária pode modificar o desvio de alguma sub-árvore para 2 ou $-2$; nesses casos vamos efetuar rotações para corrigir o desvio.

Seja $t = ($`No _ t' _`$)$ a sub-árvore tal que `desvio t` $= 2$:

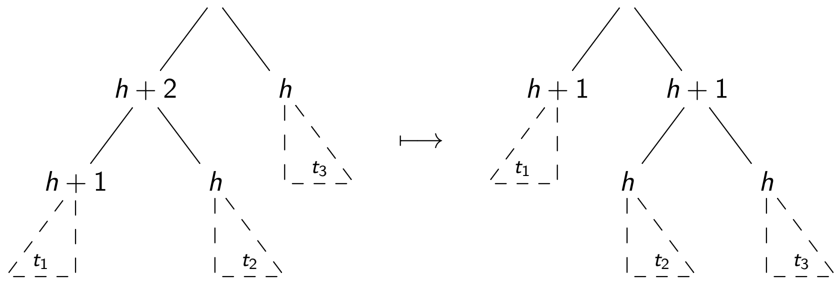
se `desvio t'` é 1 ou 0: efetuamos uma rotação simples de $t$ para a direita;

se `desvio t'` $= -1$: efetuamos duas rotações; primeiro rodamos $t'$ para a esquerda e depois rodamos $t$ para a direita.

O caso em que `desvio t` $= -2$ é simétrico.

# Rotação simples à direita

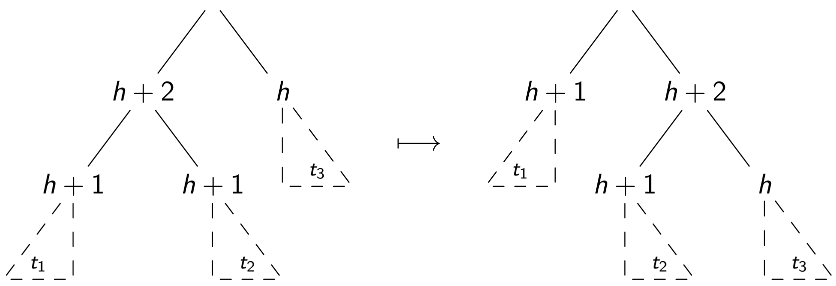
Diagrama para sub-árvore da esquerda com desvio 1 (anotando cada nó com a sua altura):




Note que, neste caso, a raiz da árvore resultante tem desvio 0 e a sub-árvore direita tem desvio 0.

# Rotação simples à direita

Diagrama para sub-árvore da esquerda com desvio 0 (anotando cada nó com a sua altura):




Note que, neste caso, a raiz da árvore resultante tem desvio -1 e a sub-árvore direita tem desvio 1.

# Rotações simples: implementação

```
rodar_dir :: Arv a -> Arv a
rodar_dir (No x (No y t1 t2) t3) =  No y t1 (No x t2 t3)
rodar_dir t = t                   -- identidade nos outros casos

rodar_esq :: Arv a -> Arv a
rodar_esq (No x t1 (No y t2 t3)) = No y (No x t1 t2) t3
rodar_esq t = t                   -- identidade nos outros casos
```

## Propriedades das rotações

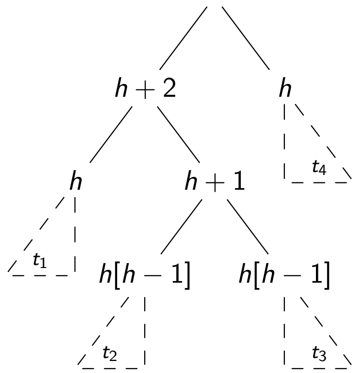
Notar que as rotações preservam a ordem entre valores, i.e. para qualquer árvore $t$ temos:

$$\texttt{listar t} = \texttt{listar (rodar\_dir t)}$$
$$\texttt{listar t} = \texttt{listar (rodar\_esq t)}$$

Em particular: se $t$ é uma árvore ordenada, então `rodar_dir t` e `rodar_esq t` também são ordenadas.

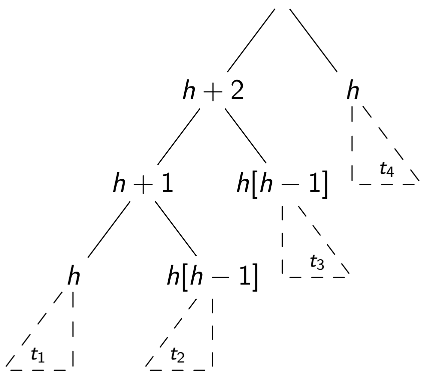
## Rotação composta (esquerda-direita) I

Configuração inicial:



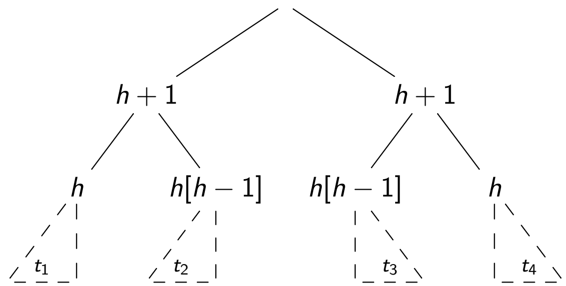

## Rotação composta (esquerda-direita) II

Após a 1ª rotação para a esquerda:




## Rotação composta (esquerda-direita) III

Após a 2ª rotação para a direita:




Note que a raiz tem desvio 0, a sub-árvore esquerda tem desvio 0 ou 1 e a direita 0 ou -1.

## Corrigir desbalanceamento I

Vamos definir uma função para rebalancear uma árvore com desvio 2 usando uma ou duas rotações.

```
corrige_dir :: Arv a -> Arv a
```

Analogamente, definimos outra função para a situação simétrica em que o desvio é -2.

```
corrige_esq :: Arv a -> Arv a
```

## Corrigir desbalanceamento II

```
corrige_dir :: Arv a -> Arv a
corrige_dir (No x t1 t2)
    | desvio t1 == -1 = rodar_dir (No x (rodar_esq t1) t2)
    | otherwise       = rodar_dir (No x t1 t2)
corrige_dir t = t                -- identidade noutros casos

corrige_esq :: Arv a -> Arv a
corrige_esq (No x t1 t2)
    | desvio t2 == 1 = rodar_esq (No x t1 (rodar_dir t2))
    | otherwise      = rodar_esq (No x t1 t2)
corrige_esq t = t                -- identidade noutros casos
```

## Rebalancear a árvore

A função seguinte verifica o desvio da árvore e, se necessário, aplica uma das funções de correção.

```
rebalancear :: Arv a -> Arv a
rebalancear t
    | d== 2  = corrige_dir t
    | d== -2 = corrige_esq t
    | otherwise = t
    where d = desvio t
```

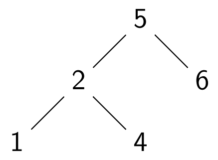
## Inserir um valor

Modificamos agora a inserção em árvores simples para rebalancear a árvore após cada chamada recursiva.

```
inserirAVL :: Ord a => a -> Arv a -> Arv a
inserirAVL x Vazia = No x Vazia Vazia
inserirAVL x (No y esq dir)
    | x==y  -- já ocorre
      = No y esq dir
    | x<y   -- inserir à esquerda
      = rebalancear (No y (inserirAVL x esq) dir)
    | x>y   -- inserir à direita
      = rebalancear (No y esq (inserirAVL x dir))
```

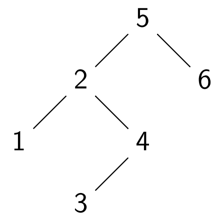
## Exemplo I

Inserir o valor 3 na seguinte árvore AVL.



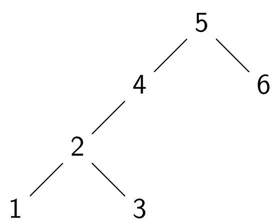

## Exemplo II

Após a inserção simples, a raiz tem desvio 2 e a sub-árvore esquerda tem desvio -1...



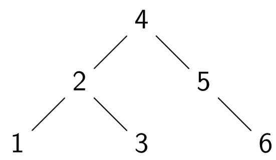

## Exemplo III

Após a 1ª rotação à esquerda, a sub-árvore esquerda fica com desvio 1...




## Exemplo IV

Após a 2ª rotação à direita, a árvore fica balanceada.



## Remover um valor

**Exercício**: escrever a função para remover um valor duma árvore AVL mantendo-a balanceada.

```
removerAVL :: Ord a => a -> Arv a -> Arv a
```

*Sugestão*: efetuar a remoção como no caso simples e usar as funções de rotação para rebalancear.

## Evitar re-calcular alturas

- o cálculo dos desvios necessita da altura de cada nó;
- podemos evitar re-calcular guardando esta informação nos nós da árvore.

**Exercício**: Defina uma implementação de AVL com nós que permitam anotação de altura do nó em uma árvore.