Programação Funcional
Aula 16 — Árvores equilibradas

Pedro Vasconcelos
DCC/FCUP

2021

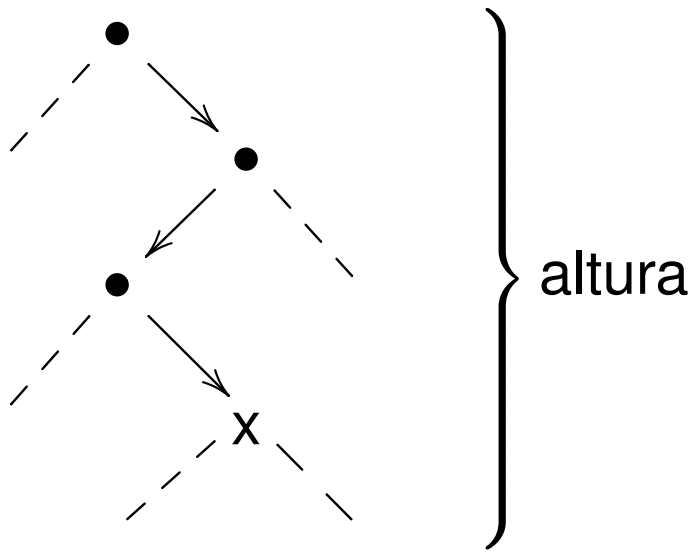
Aula anterior

- Noção de árvore binária de pesquisa
- Operações:
  1. pesquisa;
  2. inserção;
  3. remoção.

Complexidade

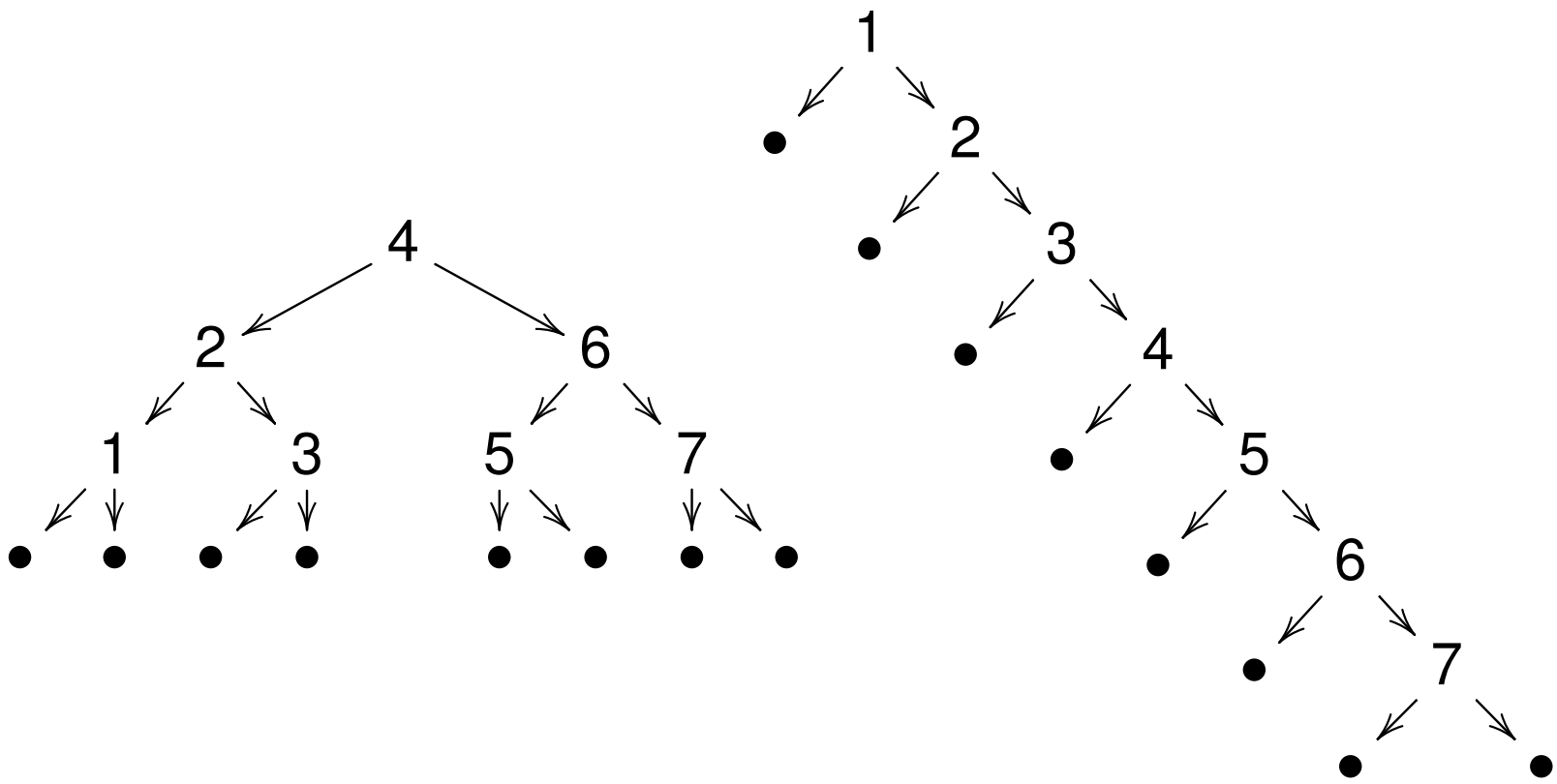
Para procurar um valor numa árvore de pesquisa:

- percorremos um *caminho* da raiz até um nó intermédio;
- o comprimento é limitado pela altura da árvore.



Complexidade (cont.)

Para um mesmo conjunto de valores, árvores com *menor altura* (ou seja, *mais equilibradas*) garantem pesquisas mais eficientes.



Árvores equilibradas

Uma árvore diz-se equilibrada se em cada nó a altura das sub-árvores difere no máximo de 1.

- Vamos escrever uma função para testar se uma árvore é equilibrada
- Começamos por definir a altura por recursão sobre a árvore

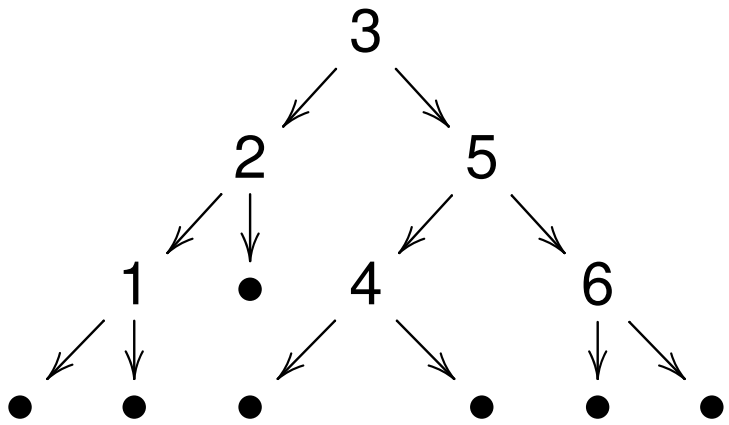
```
altura :: Arv a -> Int
altura Vazia = 0
altura (No _ esq dir) = 1 + max (altura esq) (altura dir)
```

Árvores equilibradas (cont.)

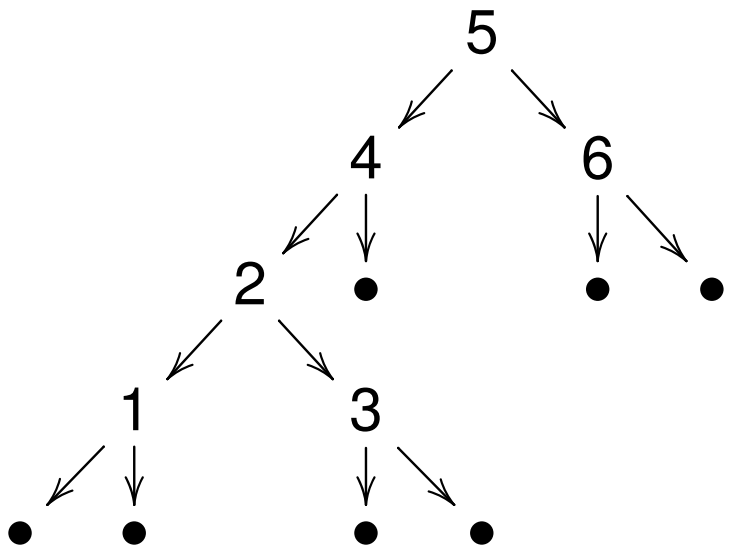
A condição de equilíbrio é também definida por recursão.

```
equilibrada :: Arv a -> Bool
equilibrada Vazia = True
equilibrada (No _ esq dir)
     = abs (altura esq - altura dir)<=1 &&
       equilibrada esq &&
       equilibrada dir
```

Exemplos



Árvore equilibrada



Árvore desequilibrada

Árvores AVL

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

Vamos seguir a apresentação no capítulo 9 do livro de Bird e Wadler (ver bibliografia).

Árvores AVL (cont.)

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

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

Árvores AVL (cont.)

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
```

Invariante

Propriedade AVL

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

Esta propriedade será invariante:

- assumimos que é válida *antes* de todas as operações;
- vamos garantir que é mantida *após* a operação.

Árvores AVL: pesquisa

A pesquisa é feita exactamente 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 trivialmente mantida.

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 efectuar *rotações* para corrigir o desvio.

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

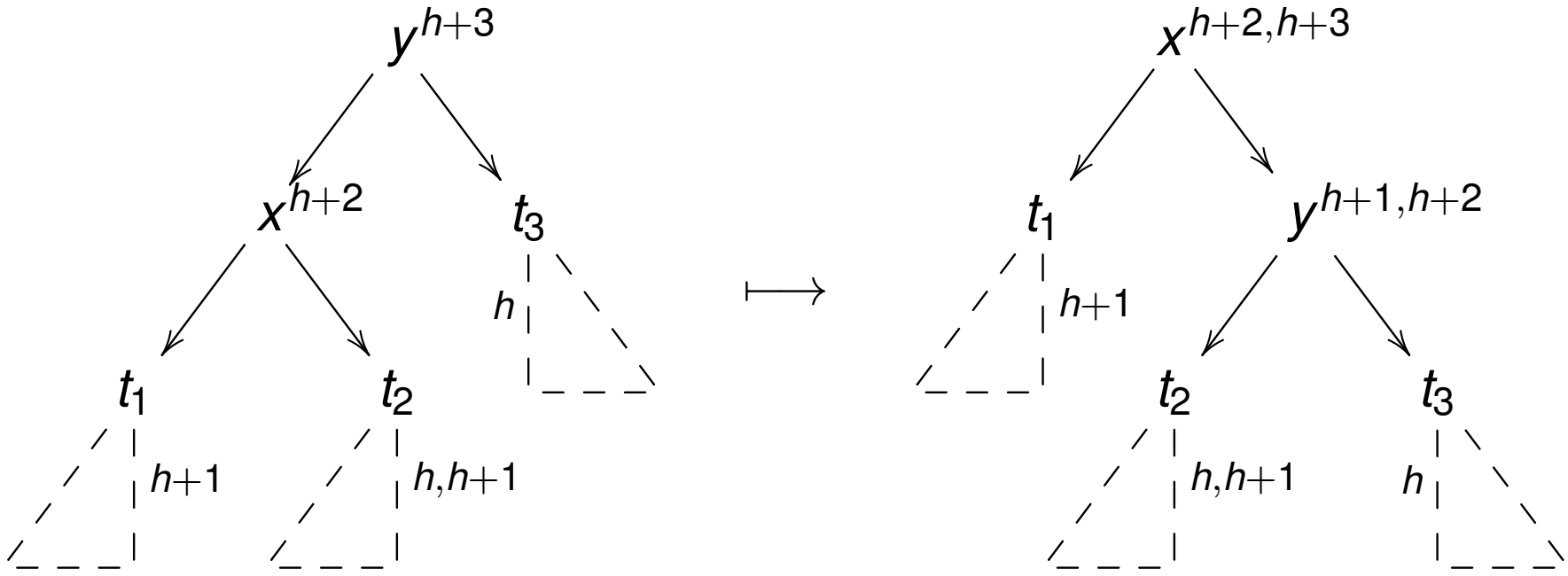
se *desvio* t' **é 1 ou 0:** efectuamos uma rotação simples de t para a direita;

se *desvio* $t' = -1$**:** efectuamos 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 (anotando cada nó com a sua altura):



Note que a raiz da árvore resultante tem desvio 0 ou -1 e a sub-árvore direita têm desvio 1 ou 0.

Rotações simples: implementação

```
rodarDir :: Arv a -> Arv a
rodarDir (No x (No y t1 t2) t3) =  No y t1 (No x t2 t3)
rodarDir t = t          -- nada a fazer nos outros casos

rodarEsq :: Arv a -> Arv a
rodarEsq (No x t1 (No y t2 t3)) = No y (No x t1 t2) t3
rodarEsq t = t          -- nada a fazer nos outros casos
```

Propriedades das rotações

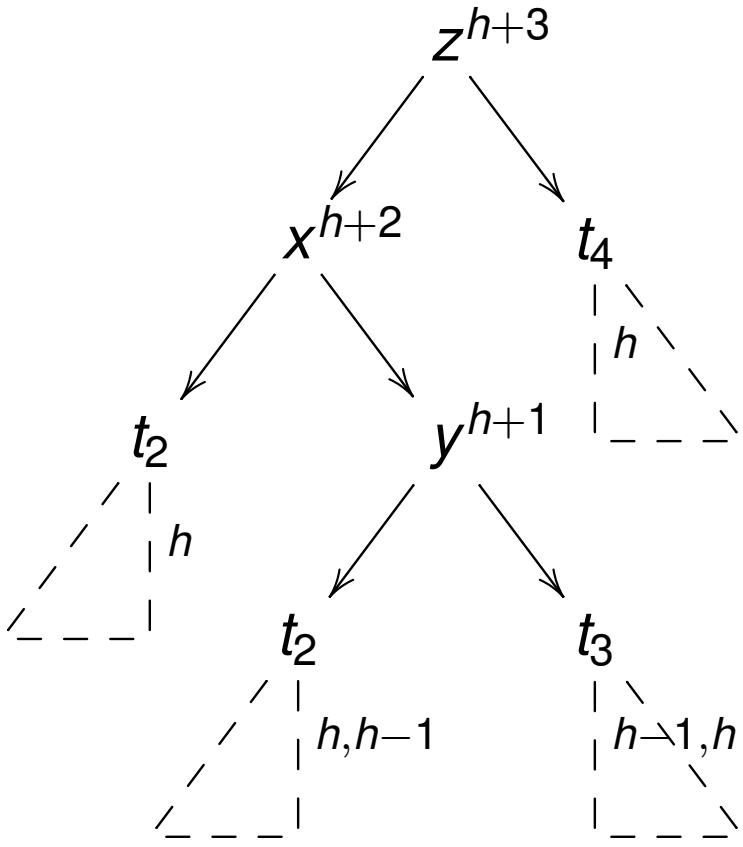
As rotações preservam os valores na árvore e a ordem entre eles.
Ou seja, para qualquer árvore t temos:

$$listar\ t = listar\ (rodarDir\ t)$$
$$listar\ t = listar\ (rodarEsq\ t)$$

Em particular: se t é uma árvore de pesquisa, então $rodarDir\ t$ e $rodarEsq\ t$ também são árvores de pesquisa.

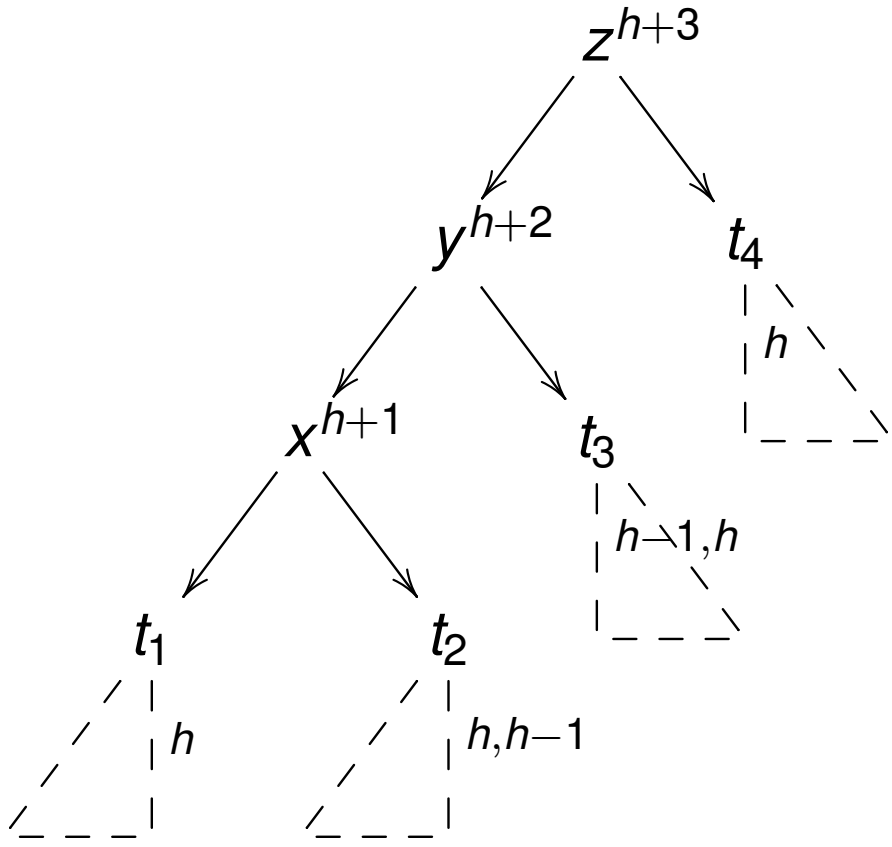
Rotação composta (esquerda-direita)

Configuração inicial:



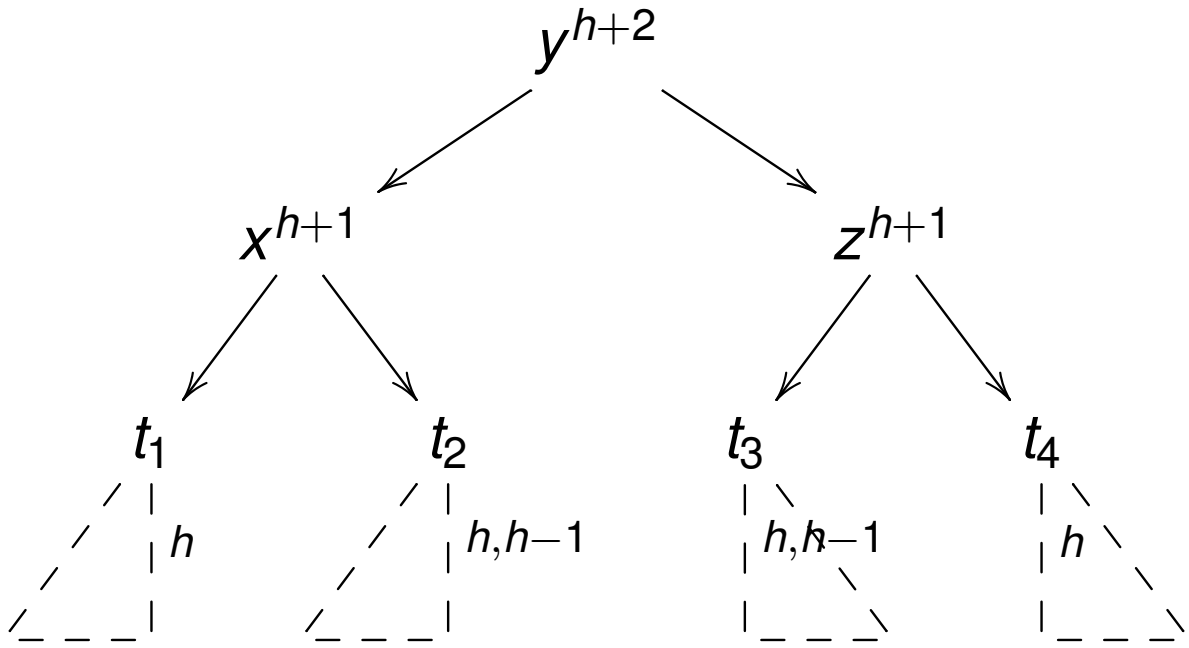
Rotação composta (esquerda-direita) (cont.)

Após a 1ª rotação para a esquerda:



Rotação composta (esquerda-direita) (cont.)

Após a rotação final 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 desequilíbrio

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

```
corrigeDir :: Arv a -> Arv a
```

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

```
corrigeEsq :: Arv a -> Arv a
```

Corrigir desequilíbrio (cont.)

```
corrigeDir :: Arv a -> Arv a
corrigeDir (No x t1 t2)
    | desvio t1 == -1 = rodarDir (No x (rodarEsq t1) t2)
    | otherwise       = rodarDir (No x t1 t2)
corrigeDir t = t          -- nada a fazer noutros casos

corrigeEsq :: Arv a -> Arv a
corrigeEsq (No x t1 t2)
    | desvio t2 == 1 = rodarEsq (No x t1 (rodarDir t2))
    | otherwise      = rodarEsq (No x t1 t2)
corrigeEsq t = t          -- nada a fazer noutros casos
```

Re-equilibrar a árvore

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

```
reequilibrar :: Arv a -> Arv a
reequilibrar t
    | d== 2  = corrigeDir t
    | d== -2 = corrigeEsq t
    | otherwise = t
    where d = desvio t
```

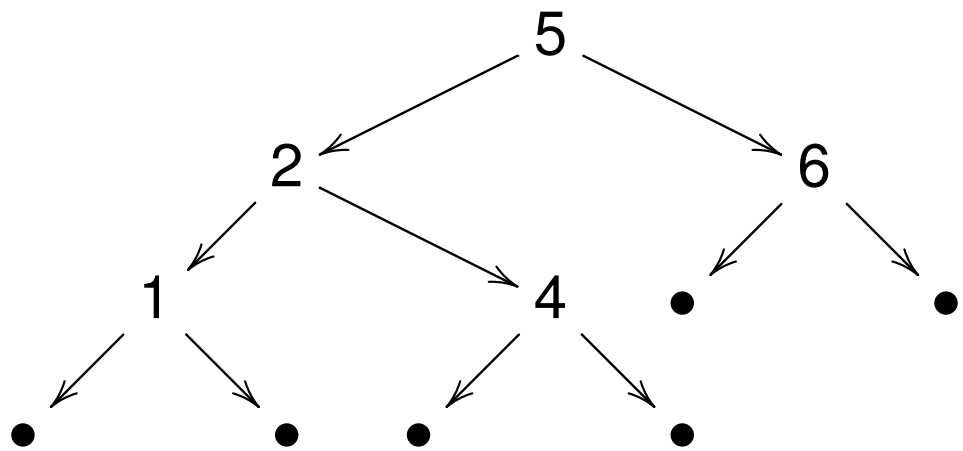
Inserir um valor

Modificamos agora a inserção em árvores simples para re-equilibrar 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
       = reequilibrar (No y (inserirAVL x esq) dir)
     | x>y   -- inserir à direita
       = reequilibrar (No y esq (inserirAVL x dir))
```

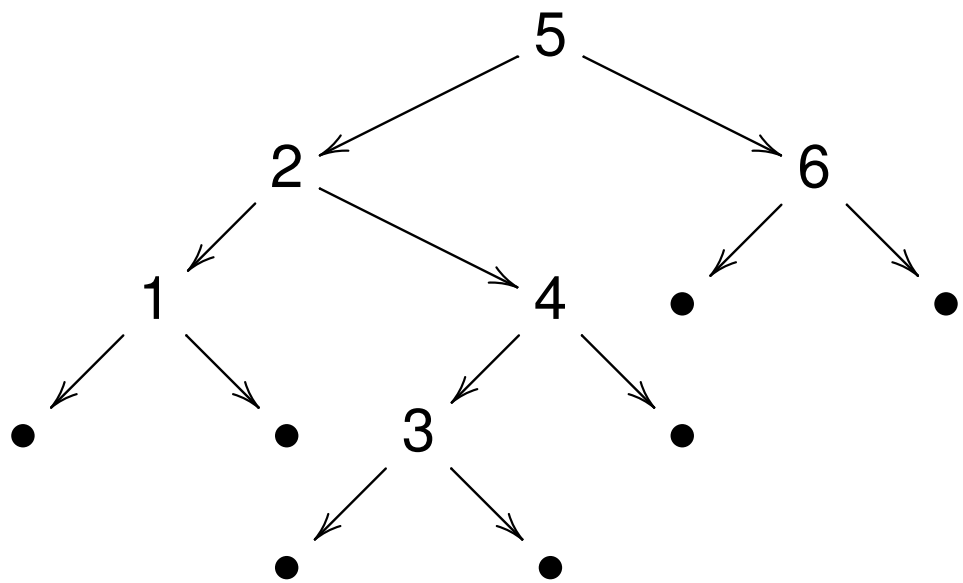
Exemplo

Inserir o valor 3 na seguinte árvore AVL.



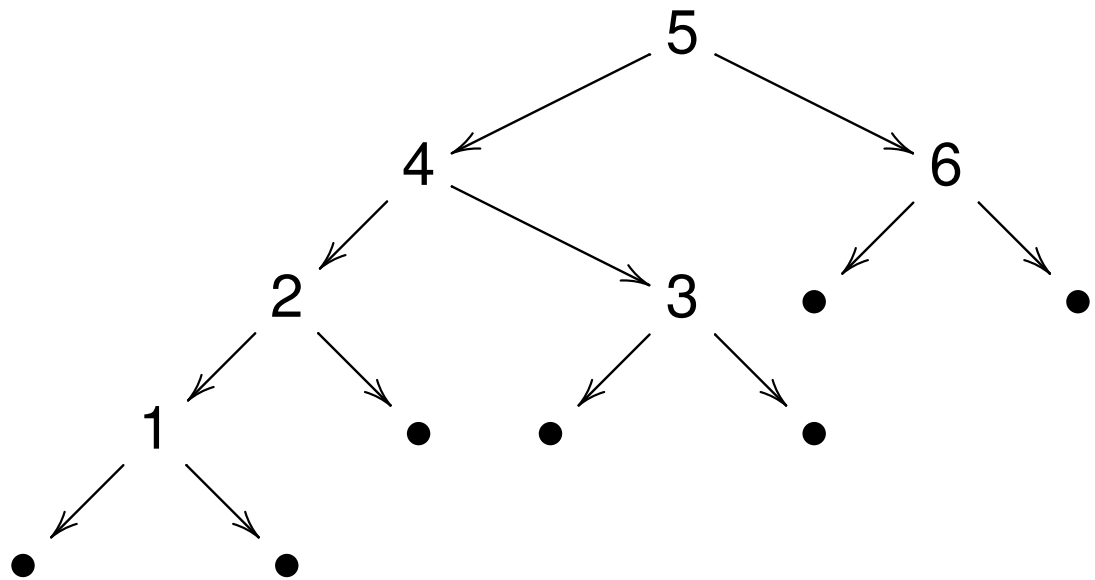
Exemplo (cont.)

Após a descida recursiva, a raiz tem desvio 2 e a sub-árvore esquerda tem desvio -1...



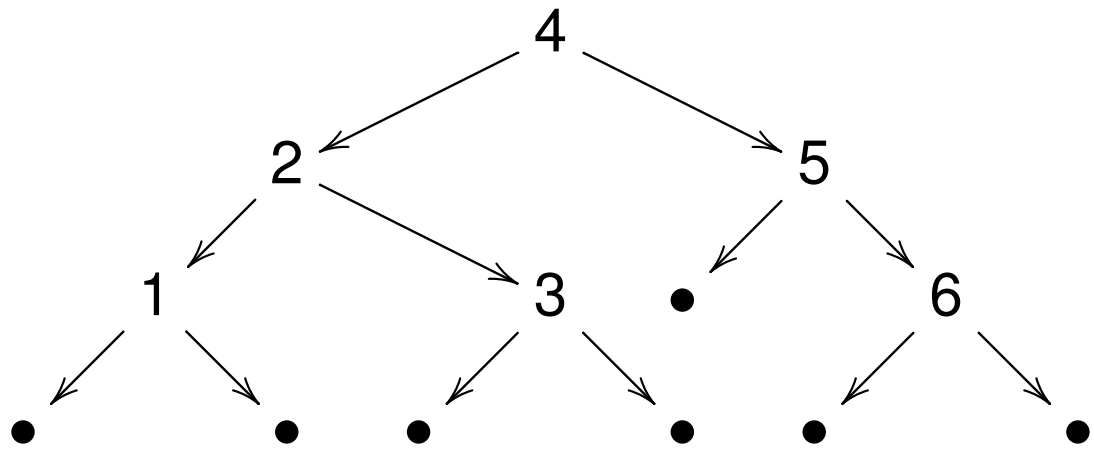
Exemplo (cont.)

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



Exemplo (cont.)

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



Remover um valor

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

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

Sugestão: efectuar a remoção como no caso simples e usar as funções de rotação para re-equilibrar.

Evitar re-calcular alturas

Exercício (folhas)

- O cálculo de desvios necessita da altura de cada nó
- Podemos evitar o cálculo recursivo guardando esta informação diretamente em cada nó
- Ao construir um nó atualizamos imediatamente a sua altura

```
data Arv a = No Int a (Arv a) (Arv a) -- altura, valor,...
           | Vazia

altura :: Arv a -> Int
altura (No h _ _ _) = h
altura Vazia        = 0

constroiNo :: a -> Arv a -> Arv a -> Arv a
constroiNo x esq dir = No h x esq dir
   where h = 1 + max (altura esq) (altura dir)
```