# Árvores AVL

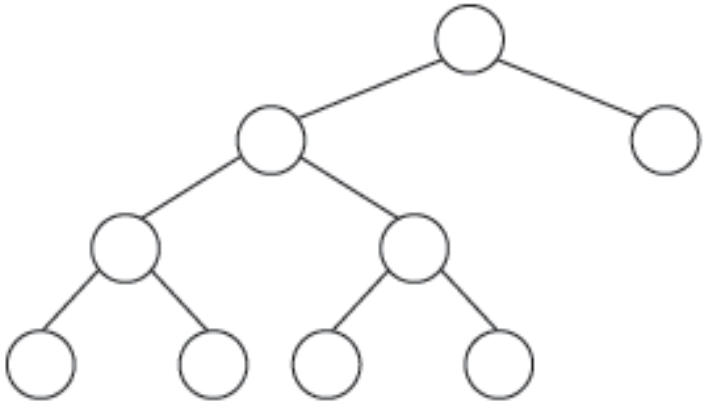
Estrutura de Dados II

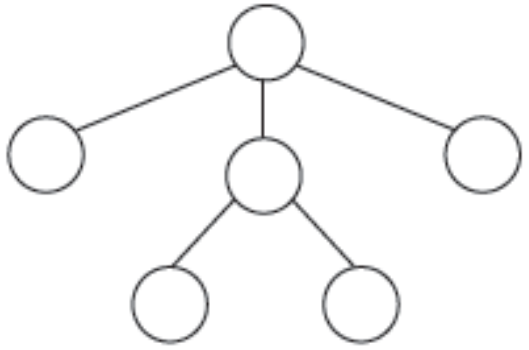
Prof. João Dallyson

Email: Joao.dallyson@ufma.br

# Árvore - Definição

- **Uma árvore é uma coleção de *n ≥ 0 nodos:***
  - Se *n =0, a árvore é dita nula*
  - Se *n > 0, a árvore tem as características:*
    - O nodo inicial é chamado de raiz (*root)*
    - Os demais nodos são particionados em T1, T2, ..., Tk estruturas disjuntas de árvores
    - As estruturas T1, T2, ..., Tk denominam-se subárvores



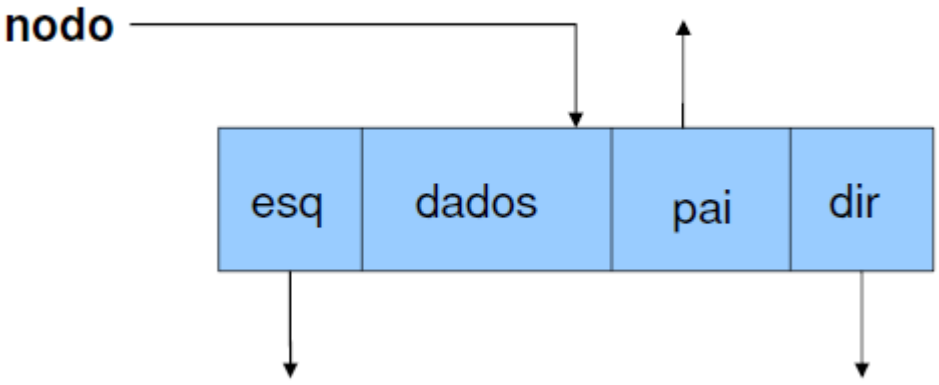


# Árvores

- **Aplicações:**
  - Representação de estruturas hierárquicas não lineares
    - Ex: relação de descendência (pai, filhos, etc...); diagrama hierárquico de uma organização; taxonomia.
  - Em computação:
    - Organização de arquivos (estrutura de diretórios)
    - Busca de dados
    - Representação de espaço de soluções (ex: jogo xadrez)

# Representação de um Nodo

- **Um nodo para uma árvore binária deve conter:**
  - Um campo DADOS
  - Um ponteiro para o Nodo Filho ESQUERDO
  - Um ponteiro para o Nodo Filho DIREITO
  - Um ponteiro para o Nodo PAI



# Representação de num Nodo

```
public class NoAVL <AnyType> {

    AnyType elemento;           // Dados do nó
    NoAVL<AnyType> esquerda;    // Filho à esquerda
    NoAVL<AnyType> direita;     // Filho à direita
    int   altura;               //Altura

    NoAVL( AnyType e )
    {
        this (e, null, null );
    }

    NoAVL(AnyType e, NoAVL esq, NoAVL  dir)
    {
        elemento  = e;
        esquerda     = esq;
        direita    = dir;
        altura   = 0;
    }

}
```

# Árvores Balanceadas

- Árvore binária de busca **balanceada**
  - é uma árvore binária onde as alturas das sub árvores esquerda e direita de cada nó da árvore diferem de no máximo uma unidade.
  - A eficiência da busca em uma árvore binária depende do seu balanceamento;
  - Algoritmos de inserção e remoção não garantem que a árvore gerada a cada passo seja balanceada.
- **Estas árvores balanceadas são chamadas de Árvores AVL (ou aproximadamente balanceadas Árvores Vermelho-Preto)**

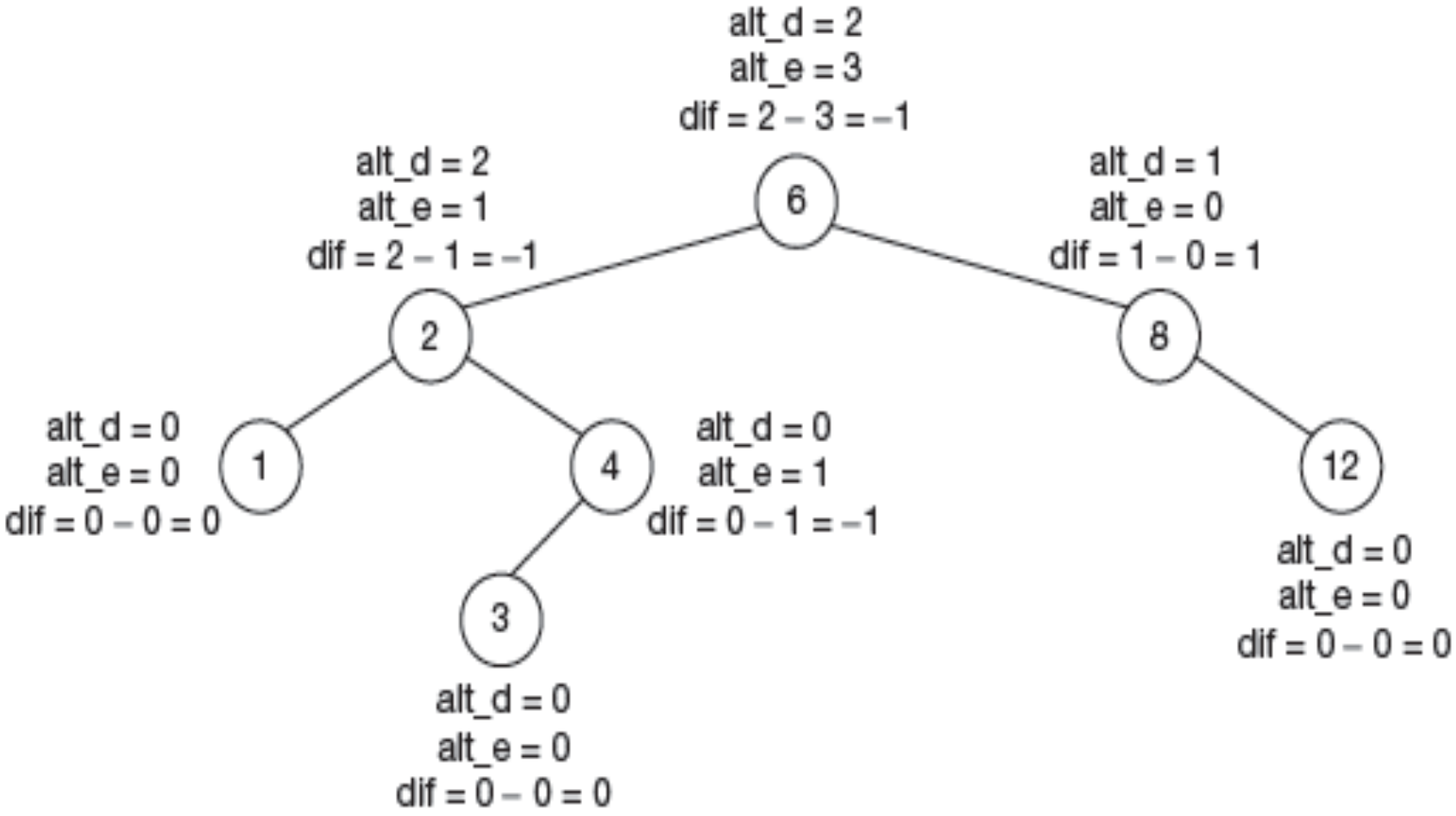
# Árvore AVL

- **Criada em 1962 por Adelson-Velsky e Landis, é uma árvore binária balanceada que obedece a todas as propriedades da árvore binária e em que cada nó apresenta diferença de altura entre as sub-árvores direita e esquerda de 1, 0 ou –1.**

# Por que usar uma árvore AVL?

- **Foi a primeira estrutura de dados a oferecer operações de inserção, remoção e busca em tempo logarítmico.**
- **Em uma árvore desbalanceada de 10.000 nós, são necessárias 5000 comparações para realizar uma busca, já uma árvore AVL, com o mesmo número de nós, essa média cai para 14 comparações.**
- **As árvores AVL tenta manter o custo de acesso o menor possível, deixando a árvore sempre com a menor altura possível.**

# Árvore AVL


alt_d = 2
alt_e = 3
dif = 2 – 3 = –1
6
alt_d = 2
alt_e = 1
dif = 2 – 1 = –1
2
alt_d = 1
alt_e = 0
dif = 1 – 0 = 1
8
alt_d = 0
alt_e = 0
dif = 0 – 0 = 0
1
4
alt_d = 0
alt_e = 1
dif = 0 – 1 = –1
12
alt_d = 0
alt_e = 0
dif = 0 – 0 = 0
3
alt_d = 0
alt_e = 0
dif = 0 – 0 = 0

# Altura de uma árvore Binária

```
Altura(x)
    SE x = None
        retorne -1
    h1 = Altura(x.esquerda)
    h2 = Altura(x.direita)
    retorne (1+Max(h1,h2))
```

# Balanceamento

- Uma árvore AVL é dita balanceada quando, para cada nodo da árvore, a diferença entre as alturas das suas subárvores (direita e esquerda) não é maior do que um.
- Caso a árvore não esteja balanceada é necessário balancea-la através da **rotação simples** ou **rotação dupla**. O balanceamento é requerido para as operações de inclusão e remoçao de elementos.

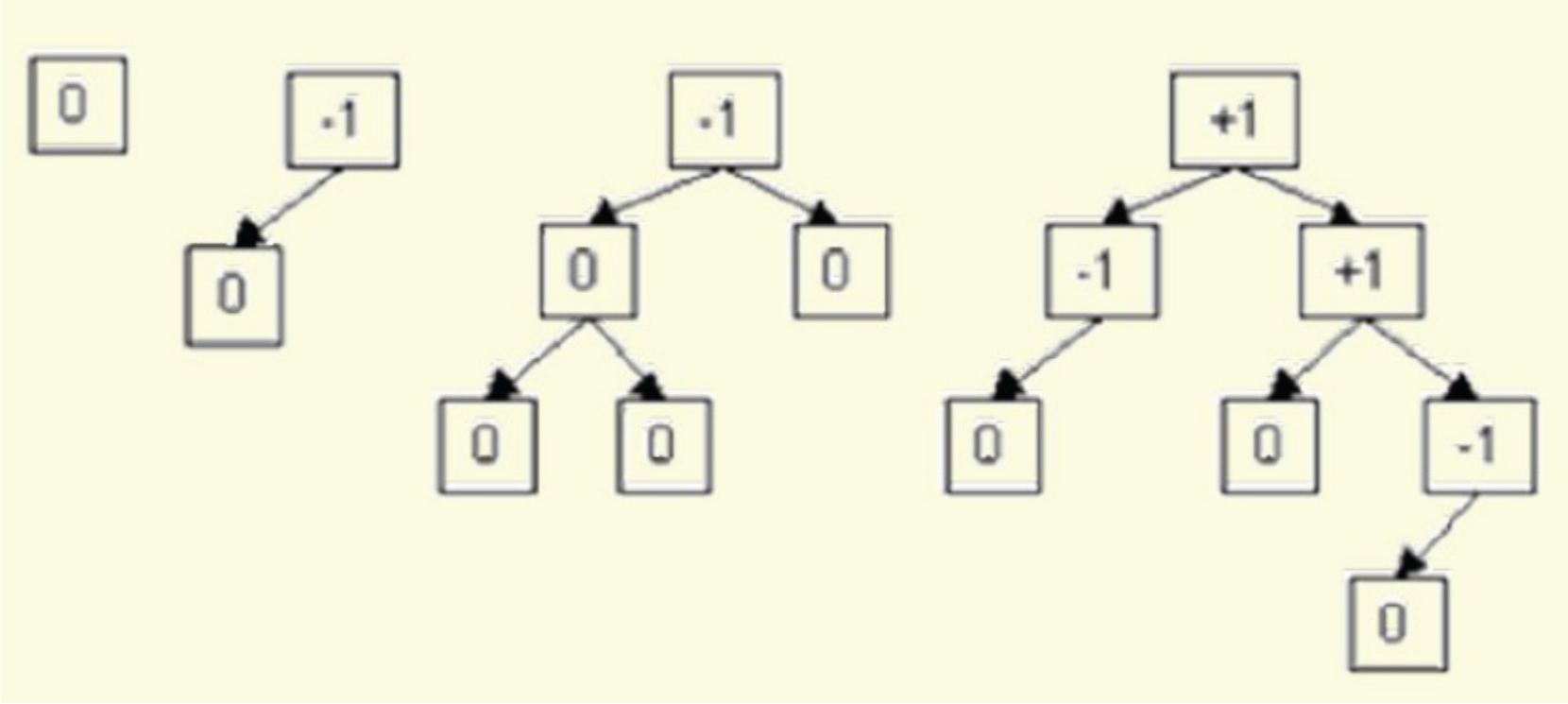
# Fator de balanceamento

- **É dado pelo seu peso em relação a sua subárvore.**
  - Um nodo pode ter um fator balanceado de 1, 0, ou -1.
  - Um nodo com fator de balanceamento -2 ou 2 é considerado um árvore não AVL e requer um balanceamento por rotação ou dupla rotação.

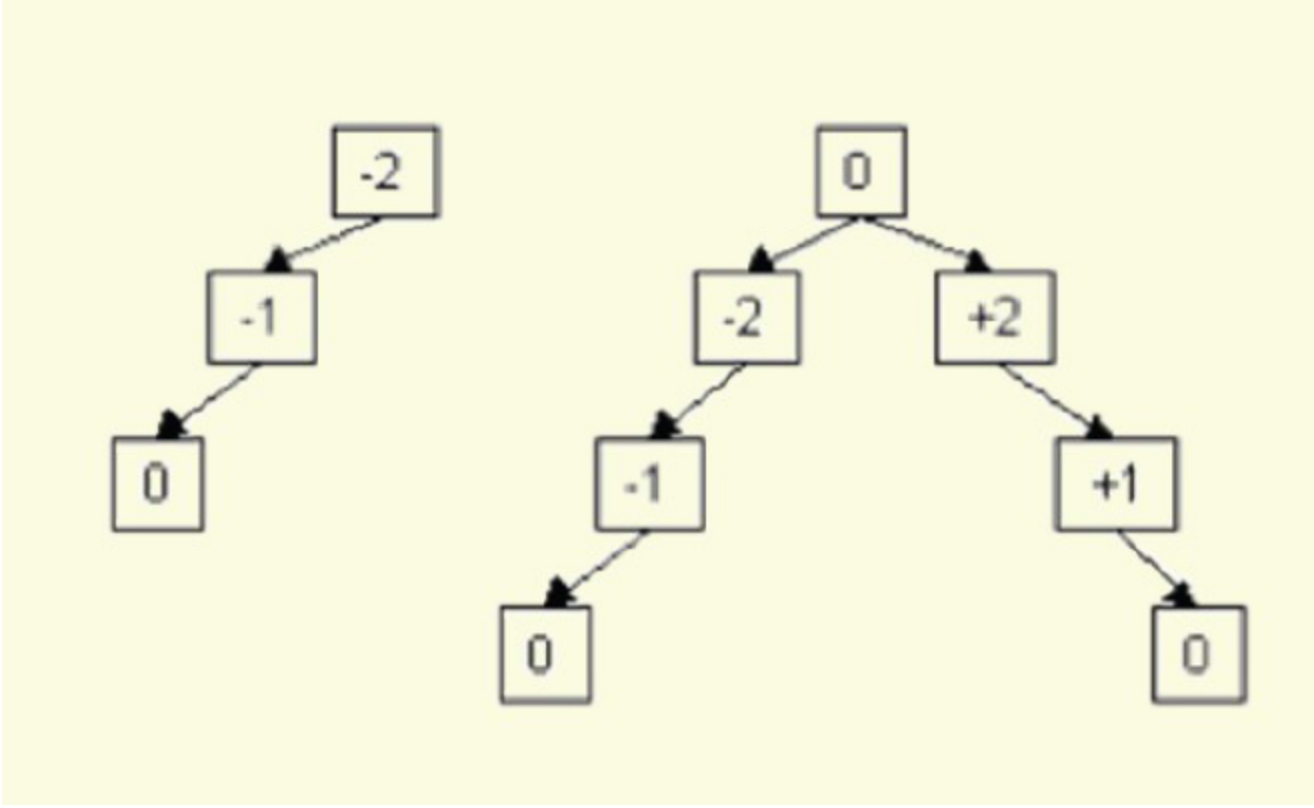
# Calculando o balanceamento

- **O fator de balanceamento (FB) pode ser dado como:**
  - h(subárvore esquerda) –h(subárvore direita)
  - Ou, h(subárvore direita) –h(subárvore esquerda)
    - Onde h(x) é a altura do nodo x.
    - Essa escolha irá influenciar no momento do balanceamento

# Exemplos de Árvores AVL

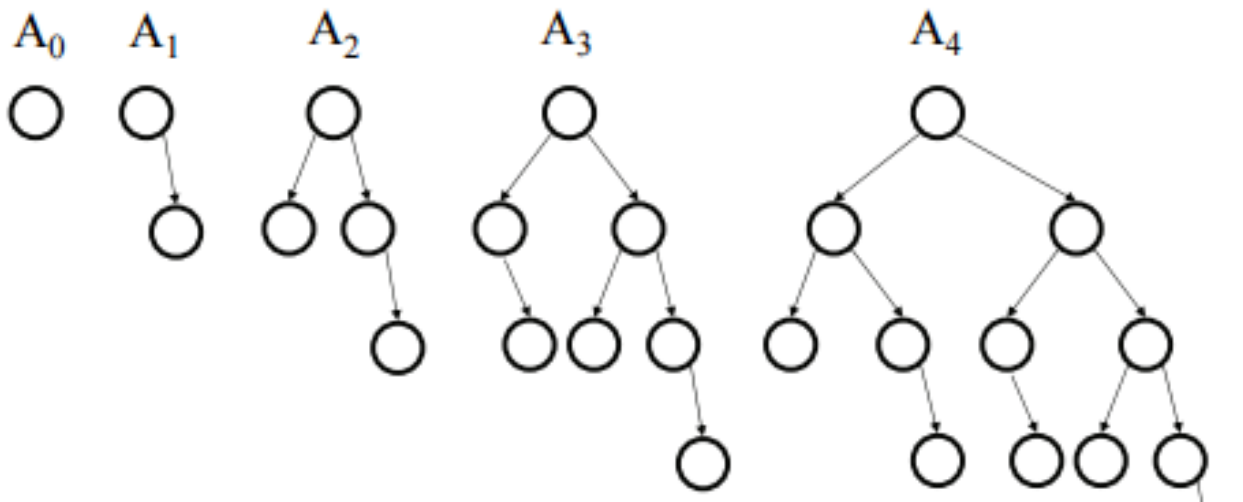


# Exemplo de Árvore Não AVL



# Tamanho mínimo de uma árvore AVL

- **Qtd. Mínima de nós em uma árvore AVL de altura H**

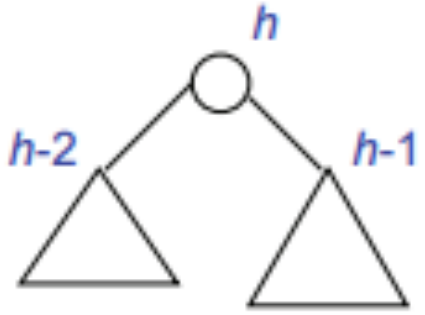



$A_h$ is smallest AVL tree of height h.
Built from $A_{h-1}$ and $A_{h-2}$ attached to a new root.
(Note: these $A_h$'s are not unique; e.g., could swap children.)

- **A3 = A2 + A1 + 1**
- **A4 = A3 + A2 + 1**

# Tamanho mínimo de uma árvore AVL

- **Qtd. Mínima de nós em uma árvore AVL de altura H**
  - $m(h)$ = minimum number of nodes in an AVL tree of height $h$.
  - Base cases
    - $m(0) = 1, m(1) = 2$
  - Induction
    - $m(\text{h}) = m(h\text{-}1) + m(h\text{-}2) + 1$



# Balanceando uma Árvore

- **Dada uma árvore não balanceada, como balanceá- la?**
  - Operações de rotações!!!!
- **Existem 4 operações de Rotações:**
  - Rotação à Esquerda
  - Rotação à Direita
  - Rotação Dupla à Esquerda
    - Rotação simples à direita
    - Rotação simples à esquerda
  - Rotação Dupla à Direita
    - Rotação simples à esquerda
    - Rotação simples à direita

# Quando usar as rotações

- **Na inserção de um elemento**
- **Na remoção de um elemento**
- **A ideia da operação de balanceamento é equilibrar o fator de balanceamento da árvore**

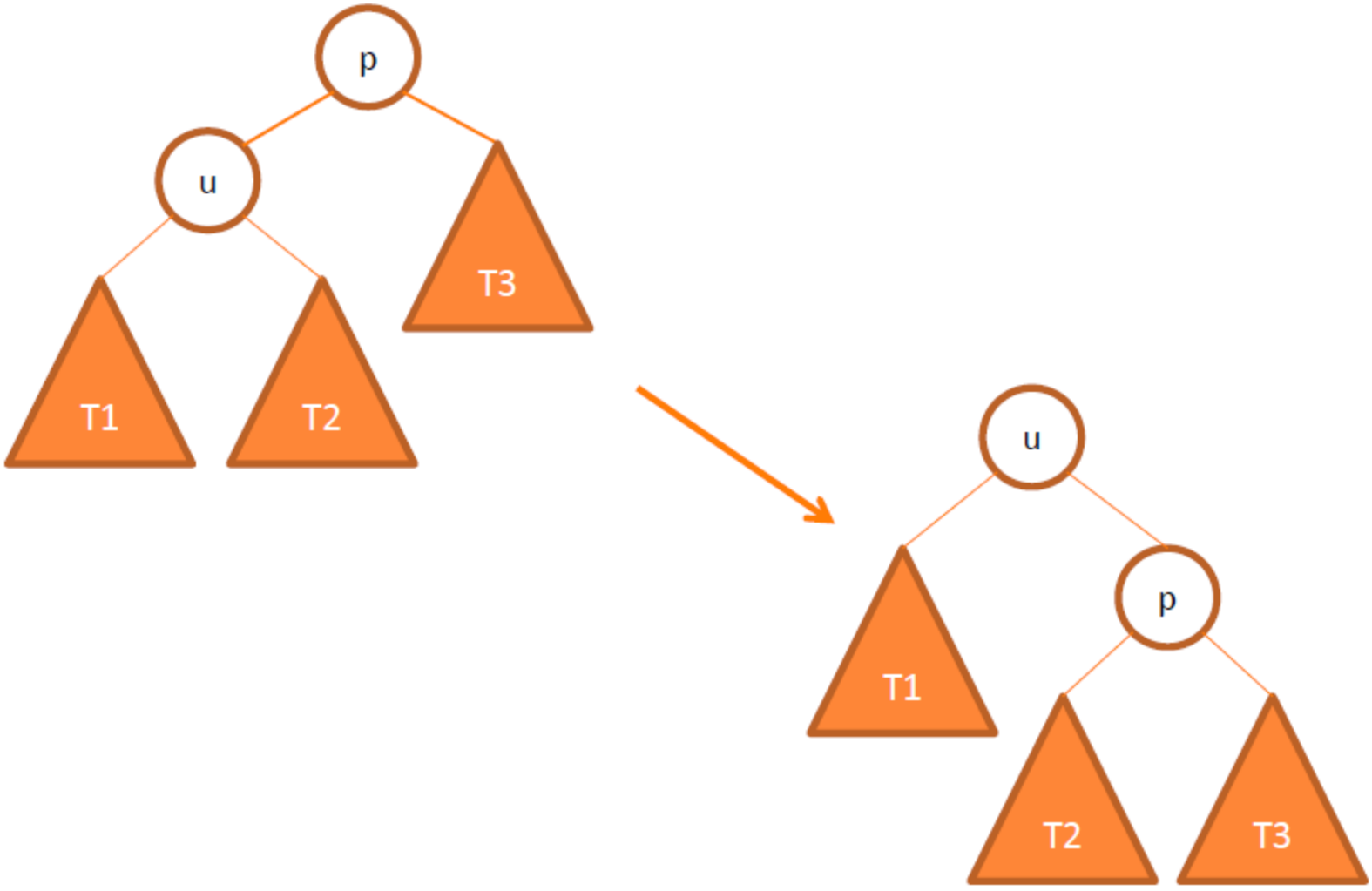
# Descrições de Rotações

| Diferença de altura de um nó | Diferença de altura do nó filho do nó desbalanceado | Tipo de rotação |
|---|---|---|
| 2 | 1 | Simples à esquerda |
| | 0 | Simples à esquerda |
| | –1 | Dupla com filho para a direita e pai para a esquerda |
| –2 | 1 | Dupla com filho para a esquerda e pai para a direita |
| | 0 | Simples à direita |
| | –1 | Simples à direita |

# Quando executar cada tipo de rotação

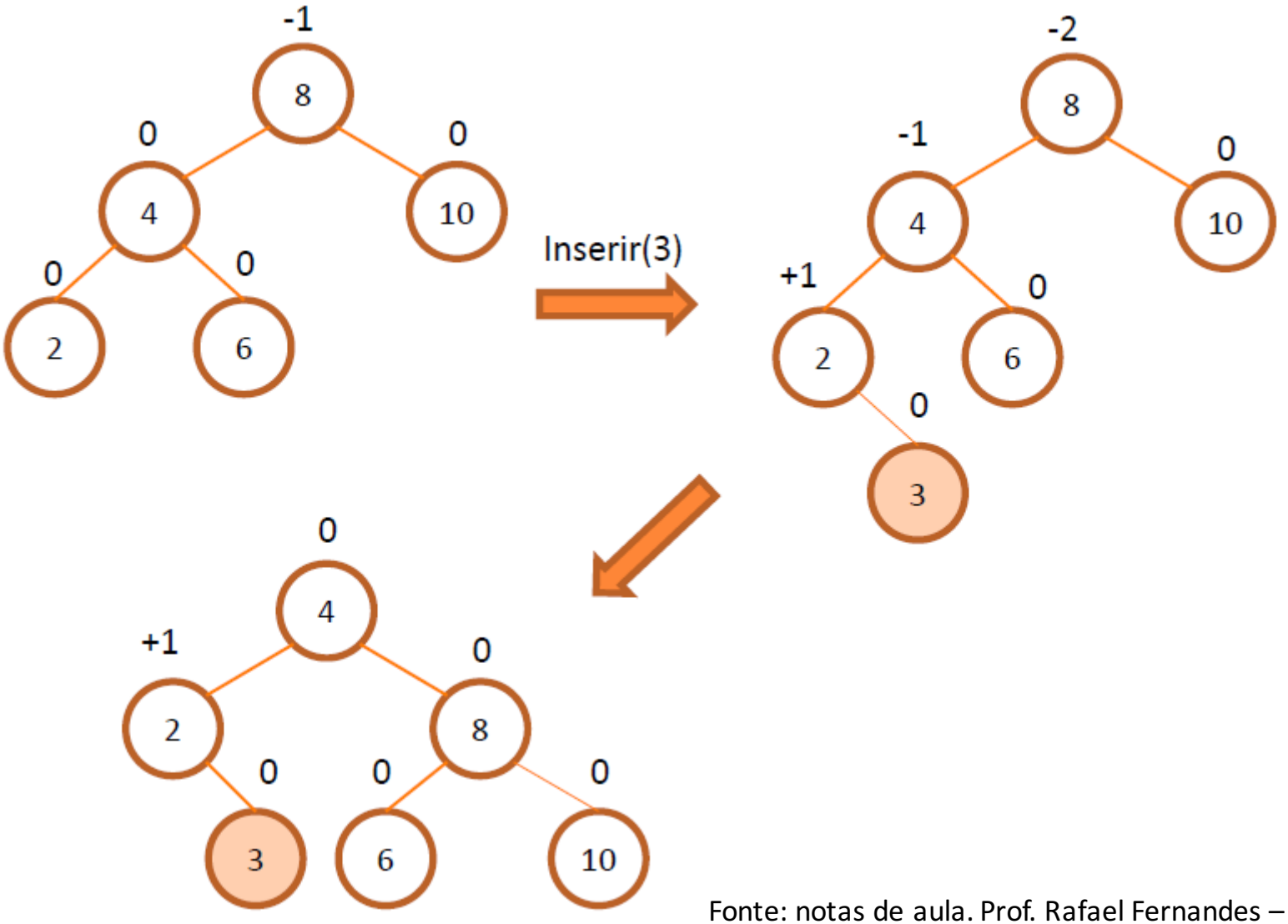
- Se FB é positivo Rotação para a esquerda
- Se FB é negativo Rotação para a direita
- Se FB do pai é negativo e o FB do filho é positivo Rotação dupla à direita
- Se FB do pai é positivo e o FB do filho é negativo Rotação dupla à esquerda

# Rotação para Direita




Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação para Direita

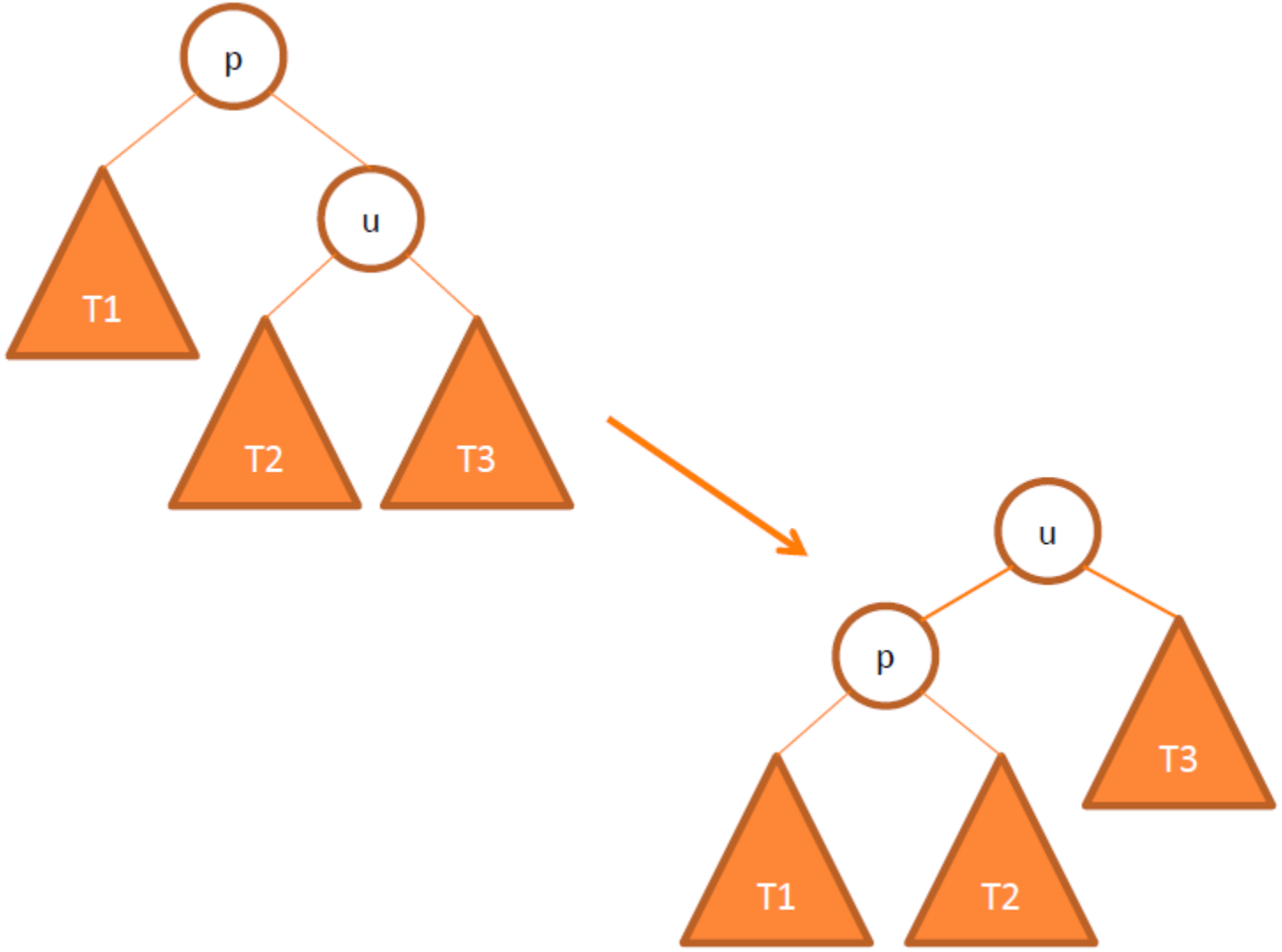



Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação para Direita

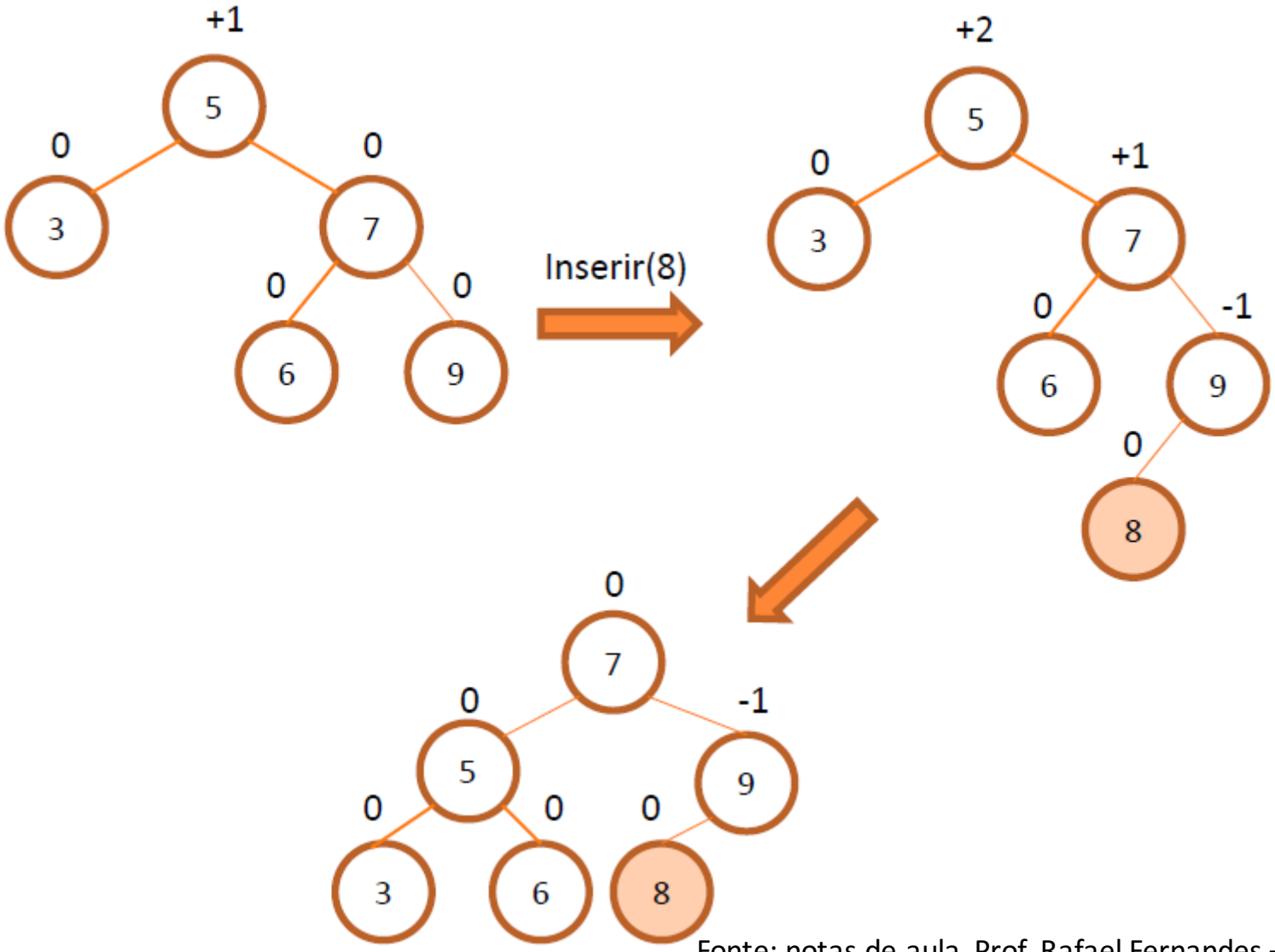
```
private NoAVL<AnyType> rotacaoDireita( NoAVL<AnyType> k2 )
{
    NoAVL<AnyType> k1 = k2.esquerda;
    k2.esquerda = k1.direita;
    k1.direita = k2;
    k2.altura = Math.max( getAltura( k2.esquerda ), getAltura( k2.direita ) ) + 1;
    k1.altura = Math.max( getAltura( k1.esquerda), k2.altura ) + 1;
    return k1;
}
```

# Rotação para Esquerda




Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação para Esquerda

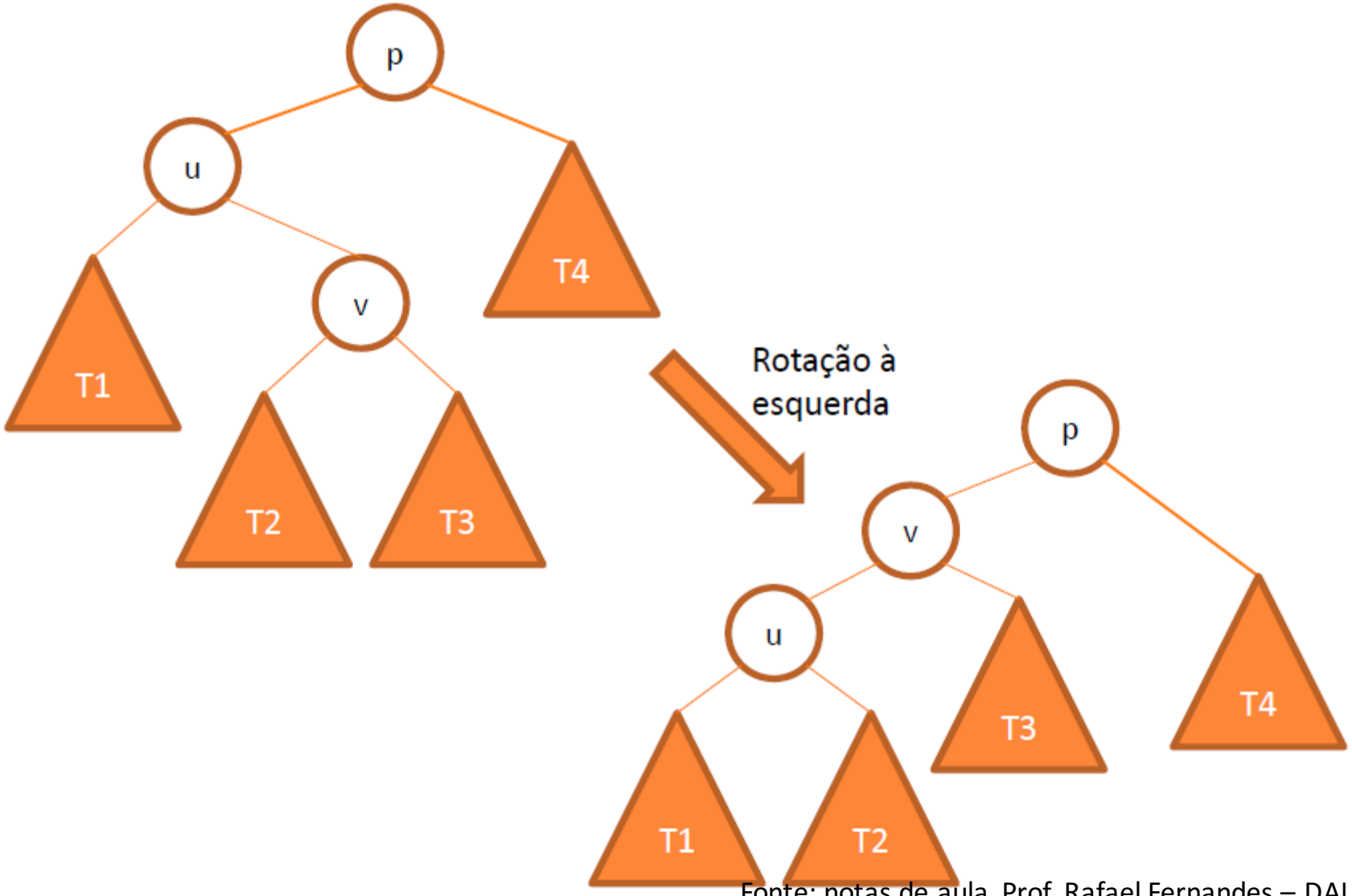



Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação para Esquerda

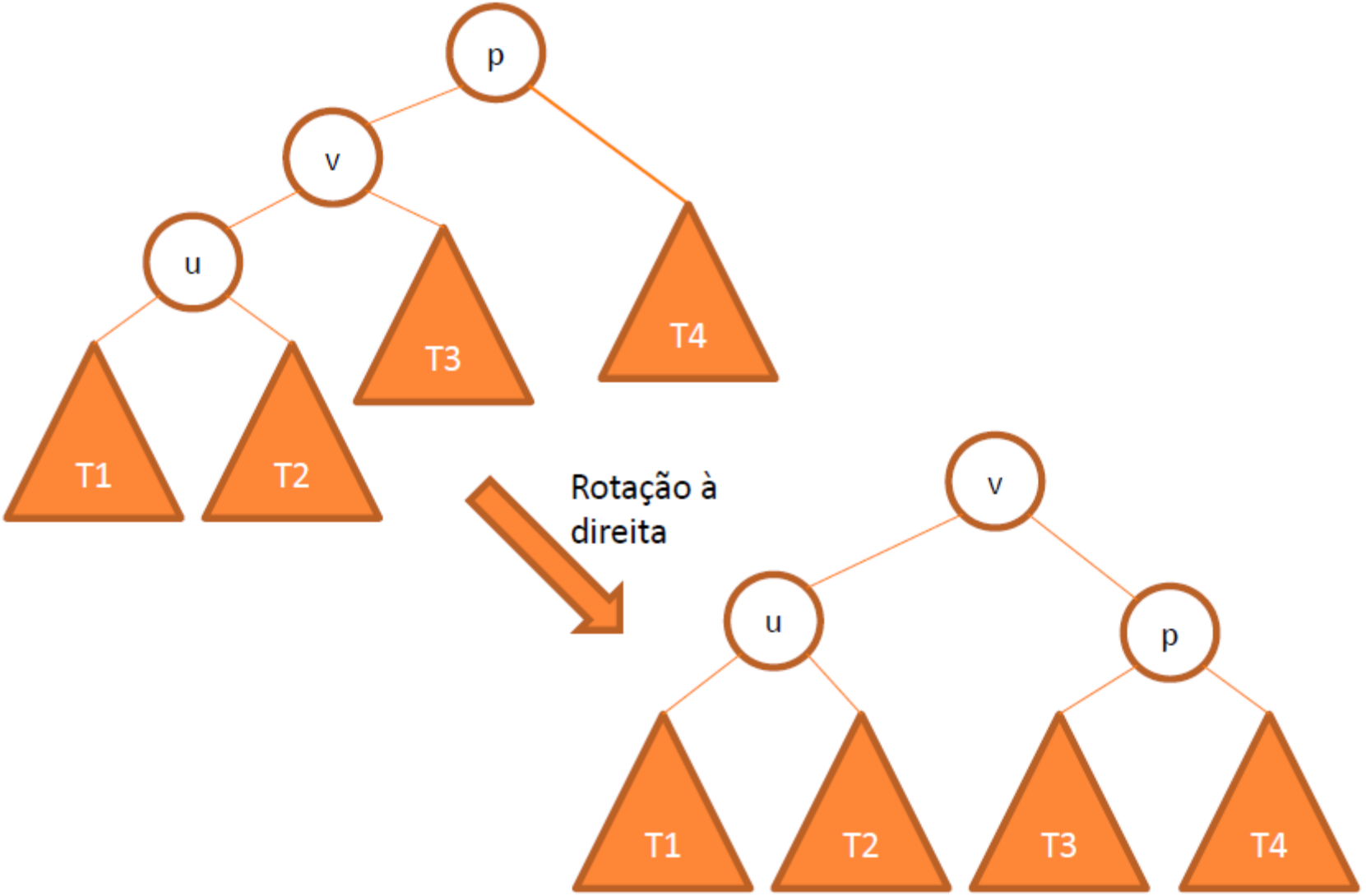
```
private NoAVL<AnyType> rotacaoEsquerda( NoAVL<AnyType> k1 )
{
    NoAVL<AnyType> k2 = k1.direita;
    k1.direita = k2.esquerda;
    k2.esquerda = k1;
    k1.altura = Math.max( getAltura( k1.esquerda ), getAltura( k1.direita ) ) + 1;
    k2.altura = Math.max( getAltura( k2.direita ), k1.altura ) + 1;
    return k2;
}
```

# Rotação Dupla à Direita (1)



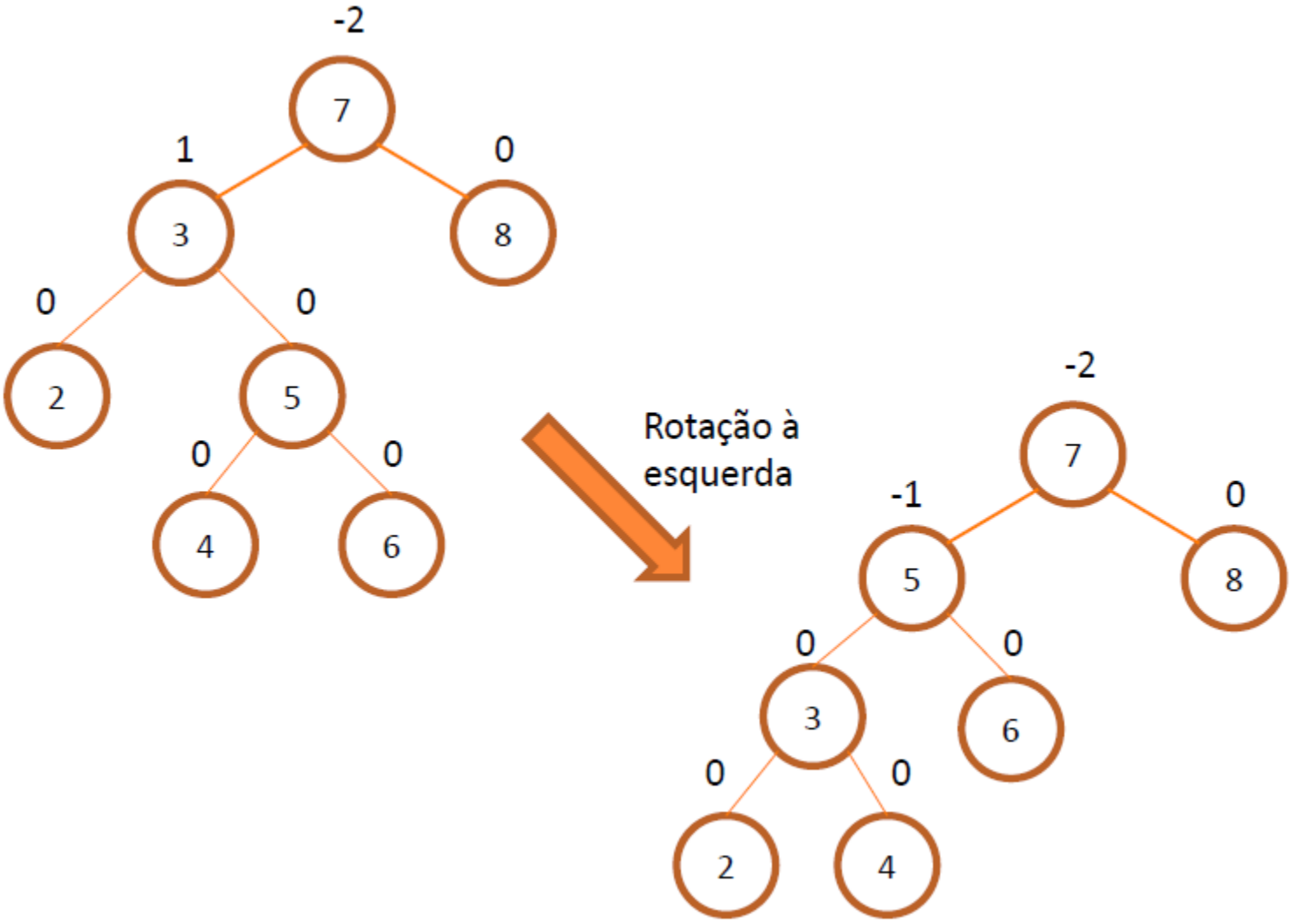

Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação Dupla à Direita (1)



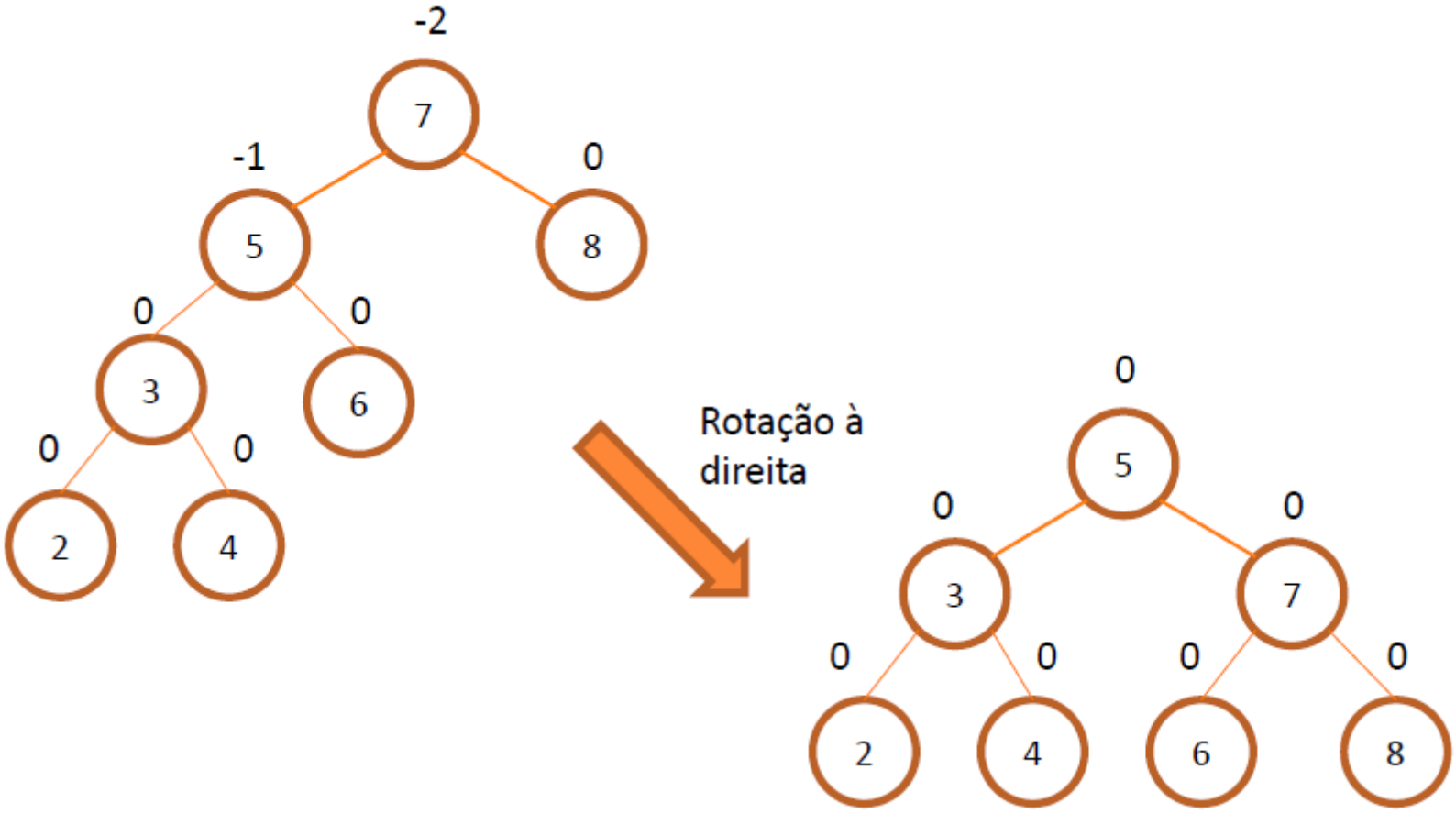

Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação Dupla à Direita




Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação Dupla à Direita




Fonte: notas de aula. Prof. Rafael Fernandes – DAI/IFMA

# Rotação Dupla Direita

```
// Rotacao Simples Esquerda + Rotacao Simples Direita
private NoAVL<AnyType> rotacaoDuplaDireita( NoAVL<AnyType> k3 )
{
    k3.esquerda = rotacaoEsquerda( k3.esquerda );
    return rotacaoDireita( k3 );
}
```

# Rotação Dupla Esquerda

```
// Rotacao Simples Direita + Rotacao Simples Esquerda
private NoAVL<AnyType> rotacaoDuplaEsquerda( NoAVL<AnyType> k1 )
{
    k1.direita = rotacaoDireita( k1.direita );
    return rotacaoEsquerda( k1 );
}
```

# Inserção

```
/* Function to insert data recursively */
private AVLNode insert(int x, AVLNode t)
{
    if (t == null)
        t = new AVLNode(x);
    else if (x < t.data)
    {
        t.left = insert( x, t.left );
        if( height( t.left ) - height( t.right ) == 2 )
            if( x < t.left.data )
                t = rotateWithLeftChild( t );
            else
                t = doubleWithLeftChild( t );
    }
    else if( x > t.data )
    {
        t.right = insert( x, t.right );
        if( height( t.right ) - height( t.left ) == 2 )
            if( x > t.right.data)
                t = rotateWithRightChild( t );
            else
                t = doubleWithRightChild( t );
    }
    else
      ;  // Duplicate; do nothing
    t.height = max( height( t.left ), height( t.right ) ) + 1;
    return t;
}
```

FONTE: https://www.sanfoundry.com/java-program-implement-avl-tree/

# Removendo elementos

- **Os exemplos vistos são para o balanceamento de uma árvore binária a partir da inserção de um novo nodo na árvore**
- **Mas e se for desejado remover um determinado nodo da árvore?**
  - Esta também irá, provavelmente, necessitar de um balanceamento!
  - Mas como proceder para a remoção de um nodo?

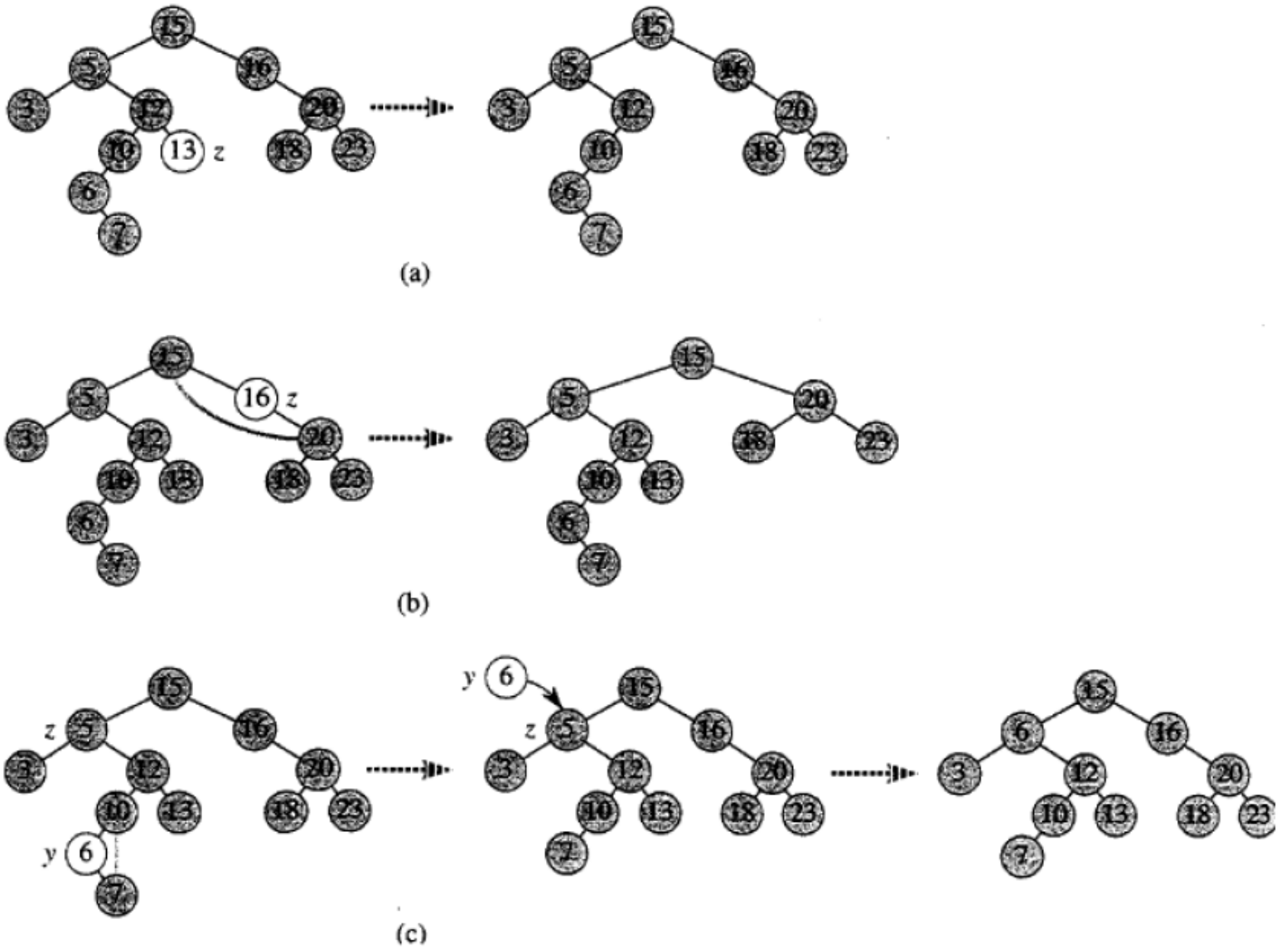



FIGURA 12.4 Eliminação de um nó $z$ de uma árvore de pesquisa binária. O nó realmente removido depende de quantos filhos $z$ tem; esse nó está levemente sombreado. (a) Se $z$ não tem nenhum filho, simplesmente o removemos. (b) Se $z$ tem apenas um filho, extraímos $z$. (c) Se $z$ tem dois filhos, extraímos seu sucessor $y$, que tem no máximo um filho, e depois substituímos a chave e os dados satélite de $z$ pela chave e os dados satélite de $y$

# Análise de complexidade

- O custo das operações é semelhante ao das árvores binárias.
- Ao se inserir um novo nó u, o pai desse nó (chamado v) terá a altura de uma de suas sub-árvores alterada. É necessário checar se a sub-árvore de raiz v está desbalanceada. Isso se faz subtraindo-se as alturas das duas sub-árvores de v, cujos valores estão armazenados no próprio nó v. Em caso de desbalanceamento, deve-se realizar uma rotação simples ou dupla.

# Análise de complexidade

- Outros nós (além do v) no caminho de v até a raiz podem também ficar desbalanceados e a verificação deverá ser feita. **O percurso do nó até a raiz é feito em O(log n) passos**.
- A **exclusão** de algum nó também pode ser feita em **O(log n)** passos. Depois, deve-se verificar se a árvore ficou desbalanceada e examinar os nós no caminho da raiz até alguma folha. O **número de rotações necessárias pode alcançar a ordem O(log n).**

# Exercício

**(Poscomp 2002)**

**27 -** Suponha que $T$ seja uma árvore AVL inicialmente vazia, e considere a inserção dos elementos $10, 20, 30, 5, 15, 2$ em $T$, nesta ordem. Qual das seqüências abaixo corresponde a um percurso de $T$ em pré-ordem:

(a) $10, 5, 2, 20, 15, 30$

(b) $20, 10, 5, 2, 15, 30$

(c) $2, 5, 10, 15, 20, 30$

(d) $30, 20, 15, 10, 5, 2$

(e) $15, 10, 5, 2, 20, 30$

# Exercício 2

**2) Dada a sequência abaixo, insira os elementos em uma árvore binária e em uma árvore AVL.**

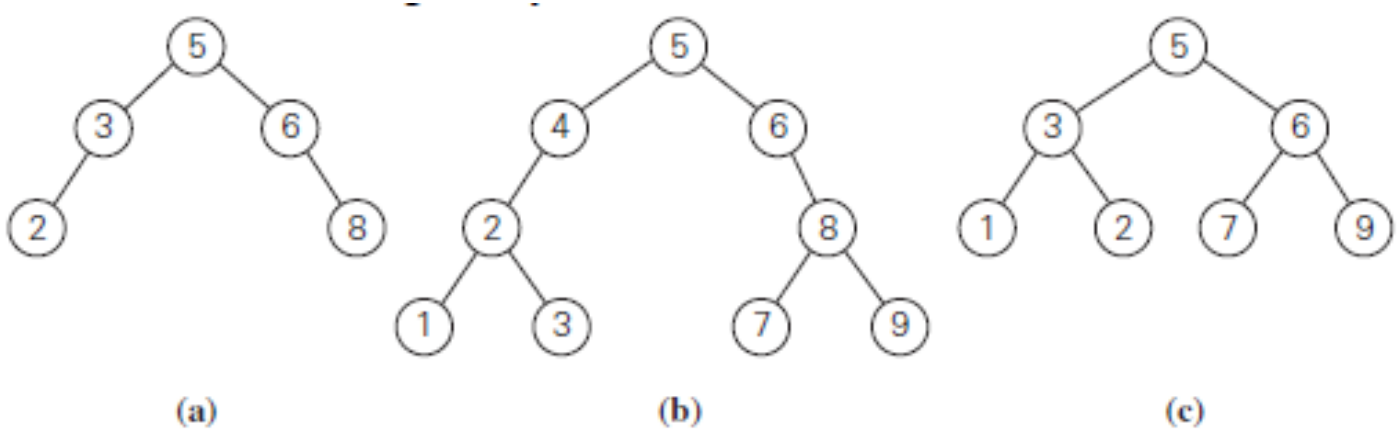
- **1 – 2 – 3 – 4 – 5 – 6 – 7 – 8**
- **2 – 5 – 8 – 3 – 10 – 7 – 1 – 4**
- **10 – 9 – 8 – 7 – 1 – 2 – 3 – 4**

**3) Implementar os métodos de inserção e de remoção de nós em Arvores AVL.**

**4) Redesenhando a árvore a cada remoção, desenhe uma árvore AVL com os números a seguir: 50, 30, 55, 10, 15, 20, 80, 90, 68. A seguir remova os números 50 e 90.**

# Exercicio

**5) Quais das seguintes árvores binárias são árvores AVL**



# Referências


**Básica**

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. Editora Campus, 2002
- Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C, Cengage Learning, 2004.
- Notas de aula. Prof. Rafael Fernandes DAI/IFMA
- Notas de aula. Prof. Tiago A. E. Ferreira. DEINFO/UFRPE

**Complementar**

- ASCENCIO, Ana Fernanda Gomes; ARAUJO, Graziela Santos. Estruturas de Dados: Algoritmos, análise da complexidade e implementações em Java e C/C++. Pearson Prentice Hall, 2010
- DROZDEK, Adam. Adam Drozdek. Data Structures and Algorithms in Java. 2. Cengage Learning. 2004. 2. Cengage Learning. 2004
- GOODRICH, Michael T. Estruturas de dados e algoritmos em java. 4 ED. Porto Alegre: Bookman, 2007. 600.
- SKIENA, Steven S.. **The Algorithm Design Manual**. 2. Springer-Verlag. 2008

# Perguntas....