

# Estrutura de Dados II

## Árvores Binárias

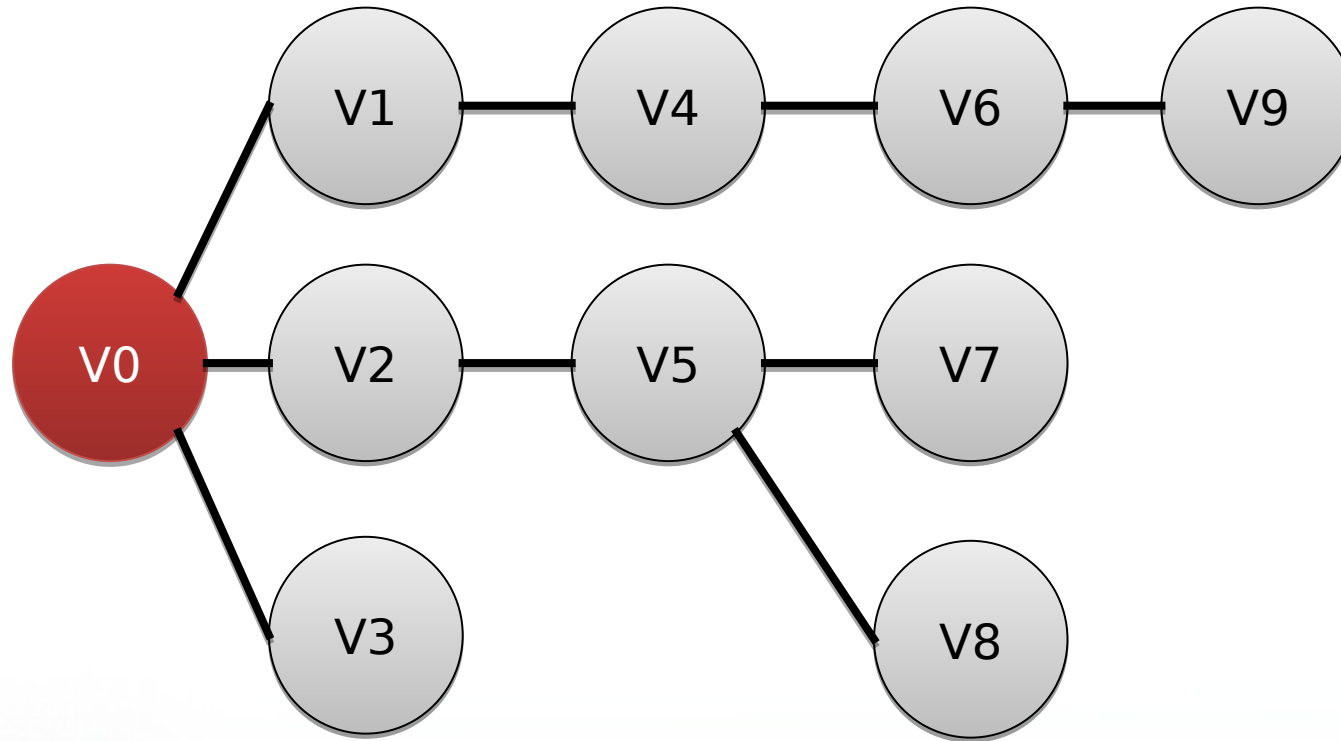
Prof. MSc. Rafael Staiger Bressan



# Árvore

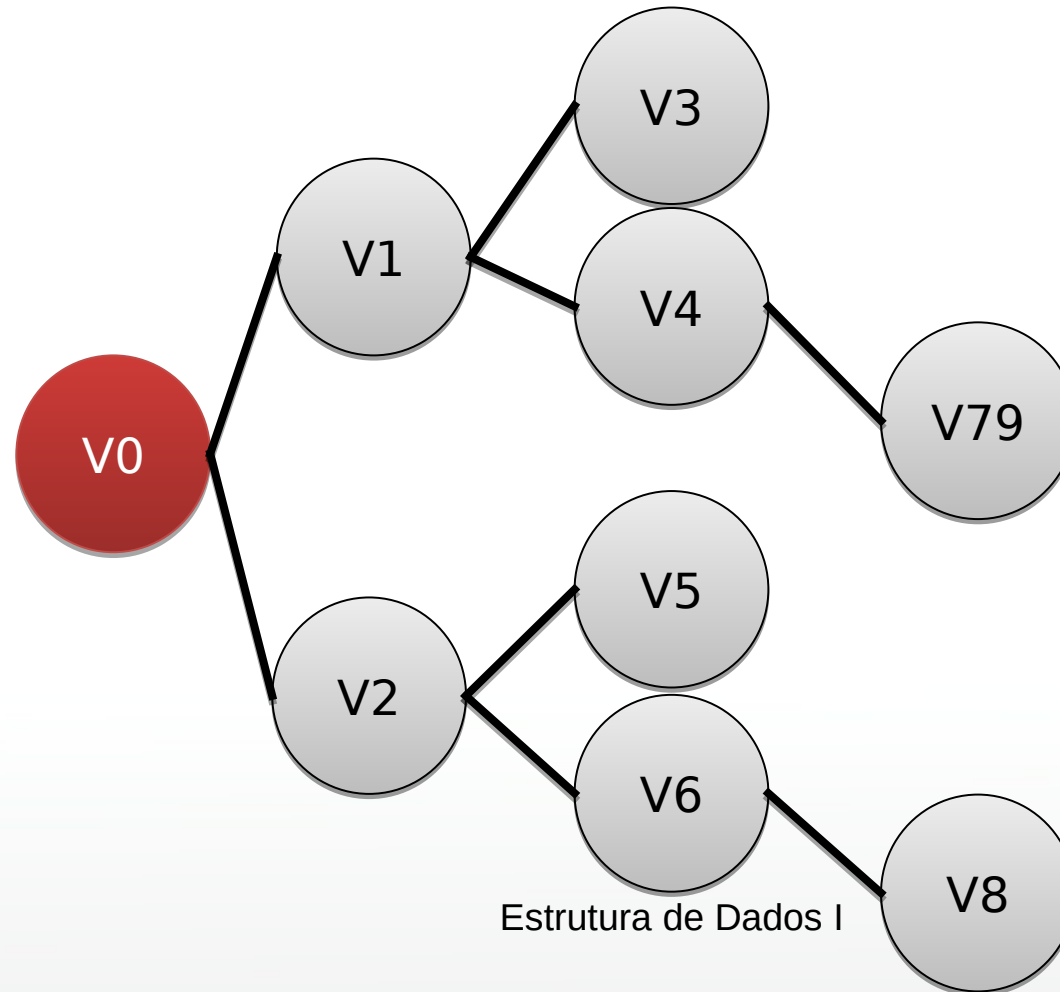
- Um grafo conexo sem ciclos é chamado de árvore.
  - Determina-se um nó para ser a raiz da árvore.
  - A distância entre cada nó e a raiz é denominado de (Nível);
  - Os vértices de grau 1 em uma árvore são chamados de folhas;

# Árvore



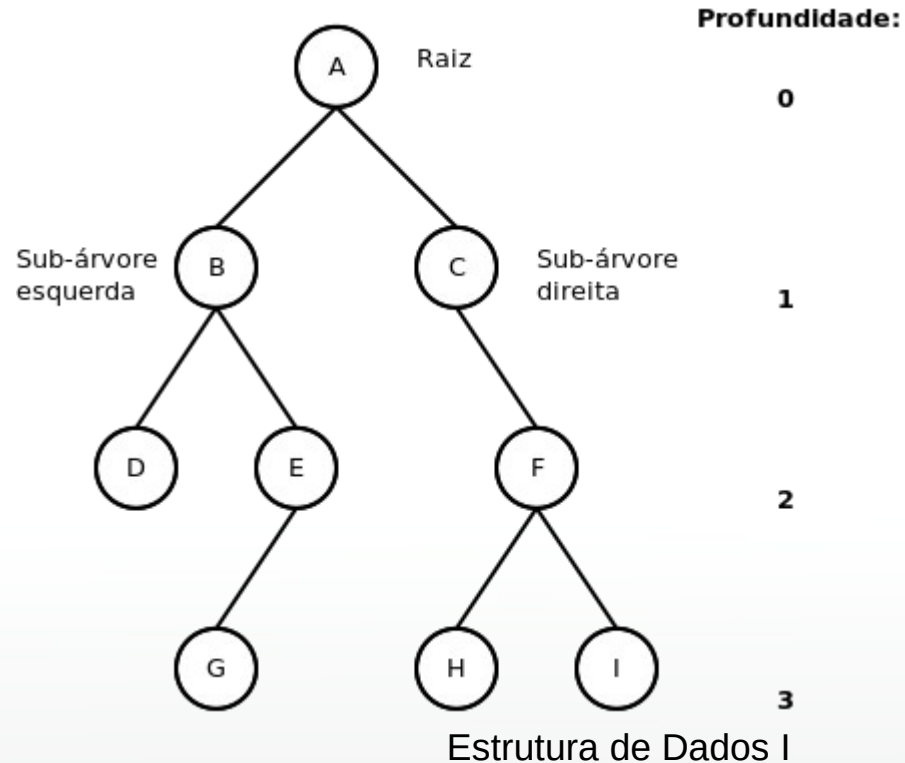
Qual é o nível de V6? Quais são os vértices de grau 1?

# Árvore Binária



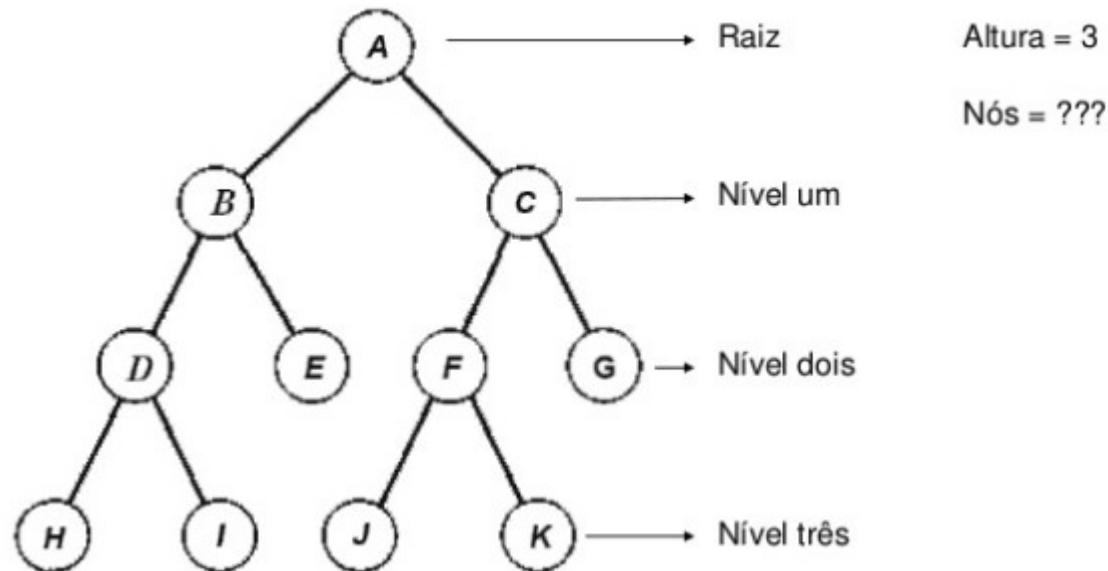
# Introdução

- Quando utilizamos estruturas de dados do tipo árvore?

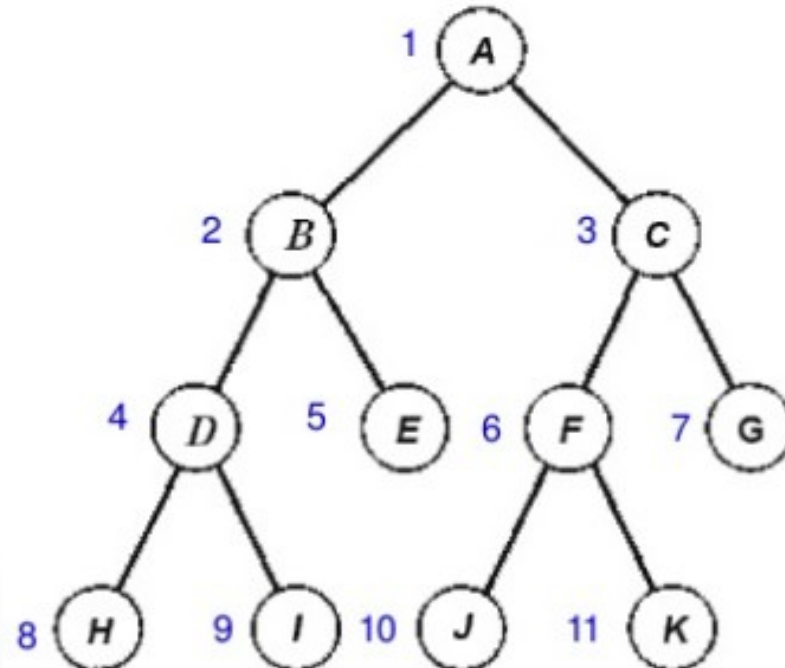


# Árvore Binária

- Tipo particular de árvore.
- Cada nó pode ter no máximo 2 sucessores (filhos), uma à esquerda e outro à direita do nó (pai).



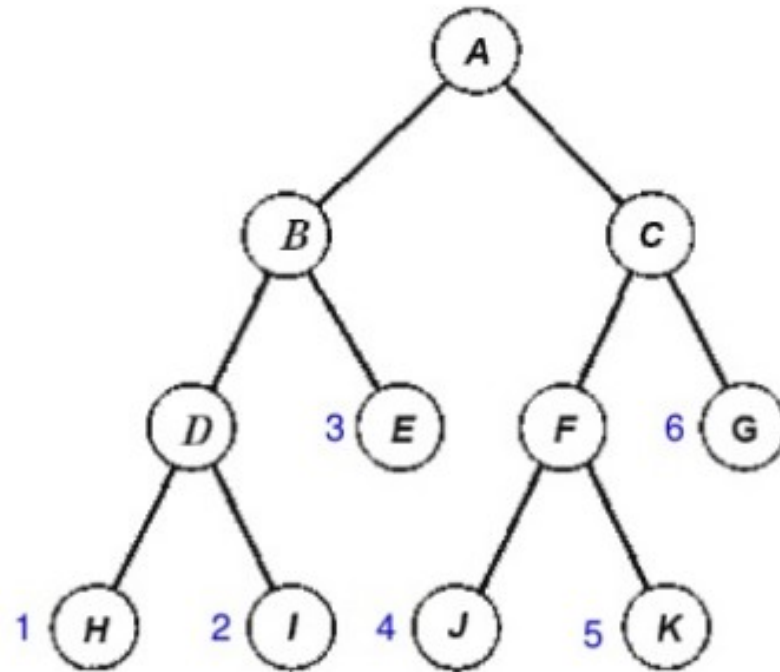
# Árvore Binária



Altura = 3

Nós = ???

# Árvore Binária



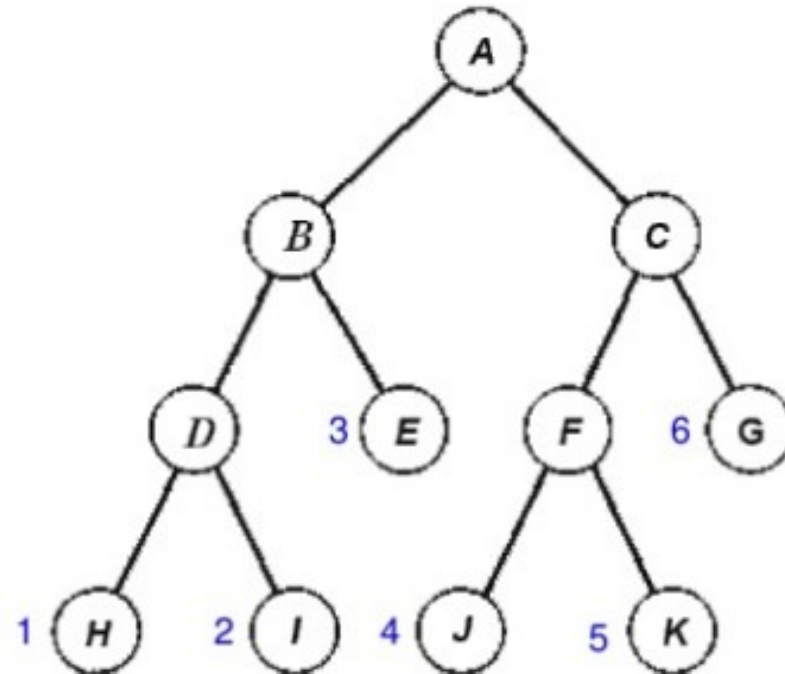
Altura = 3

Nós = 11

Folhas = ?



# Árvore Binária



Altura = 3

Nós = 11

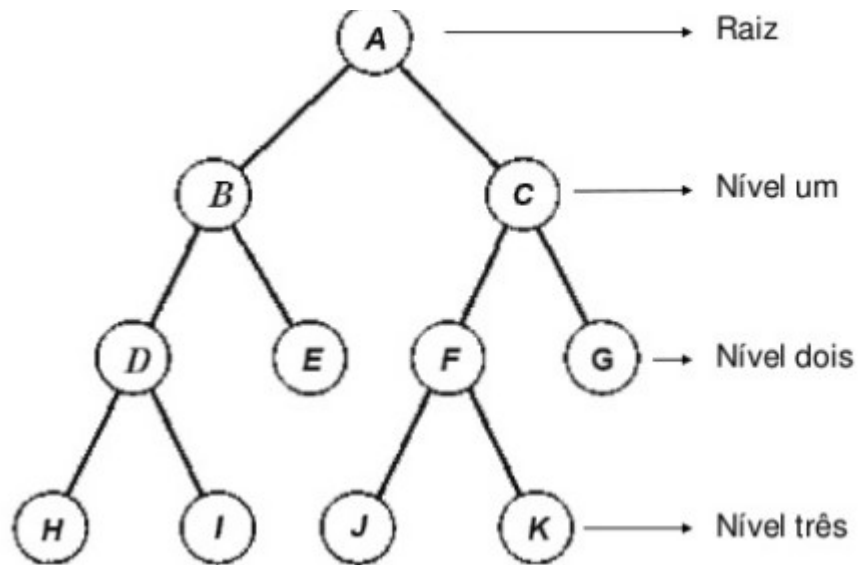
Folhas = 6



# Árvore Estritamente Binária

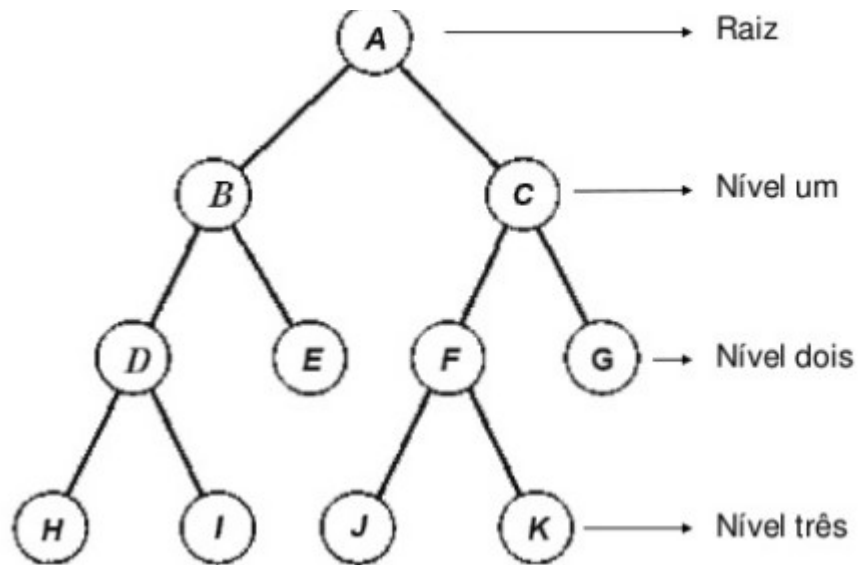
- É uma árvore binária em que cada nó tem **0 ou 2 filhos**.
- A quantidade de nós presentes numa árvore estritamente binária se dá pela formula:  **$n = (2 * f) - 1$** , onde,  **$f$**  = número de folhas na árvore e  **$n$**  é a quantidade de nós.

# Prática

$$n = (2 * f) - 1$$


quantidade de nós?

# Prática

$$n = (2 * f) - 1$$


quantidade de nós?

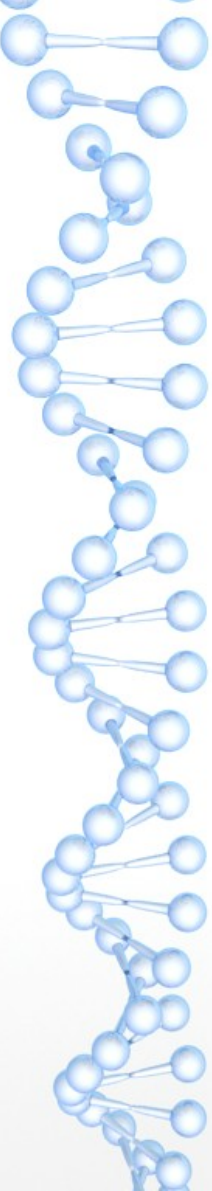
$$N = (2 * 6) - 1 = (12) - 1 = 11$$



## Prática

$$n = (2 * f) - 1$$

- 1 - Uma árvore estritamente binária possui exatamente **4 folhas**, qual a quantidade de nós presentes na árvore?
- 2 - Uma árvore estritamente binária possui exatamente 7 nós, A | B | C | D | E | F | G.
  - Qual a quantidade de folhas?



# Prática

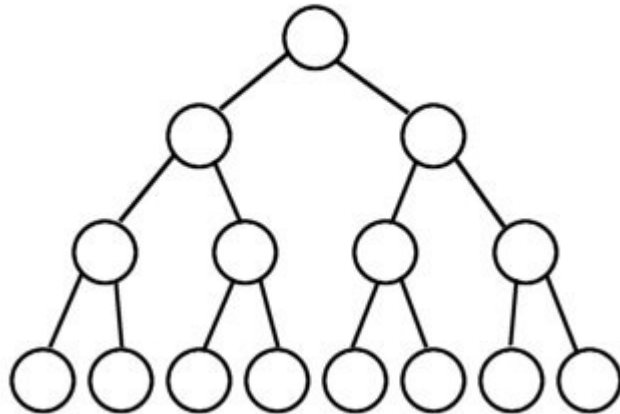
$$n = (2 * f) - 1$$

- 1 - Uma árvore estritamente binária possui exatamente **4 folhas**, qual a quantidade de nós presentes na árvore?
  - $n = (2 * 4) - 1 = 7$
- 2 - Uma árvore estritamente binária possui exatamente 7 nós, A | B | C | D | E | F | G. Qual a quantidade de folhas?
  - $n = (2*f)-1$
  - $n+1 = (2*f)-1+1$
  - $n+1 = (2*f)$
  - $(n+1)/2 = (2*f)/2$
  - $(n+1)/2 = f$
  - $f = (n+1)/2$
  - $f = (7+1)/2 = (8)/2 = 4$

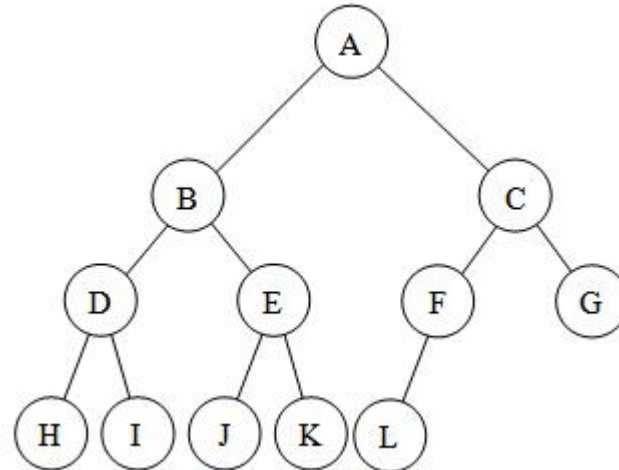
# Árvore Binária Completa

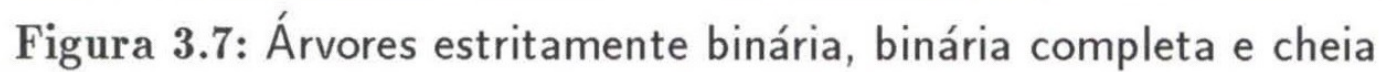
- Se  $n$  é um nó com alguma sub-árvore vazias, então  $n$  se localiza no penúltimo ou no último nível. Portanto, toda árvore cheia é completa e estritamente binária.

Árvore binária cheia



Árvore Binária Completa







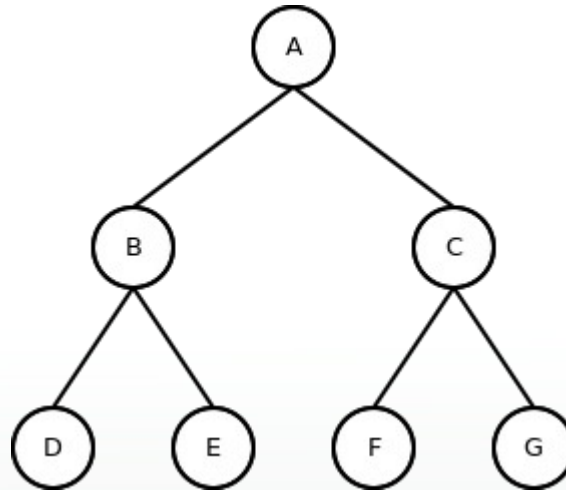


# Implementação Árvore Binária

- Implementação
  - `arvore-binaria.c` (página 20 – livro base)

# Uma Árvore Binária Diferente

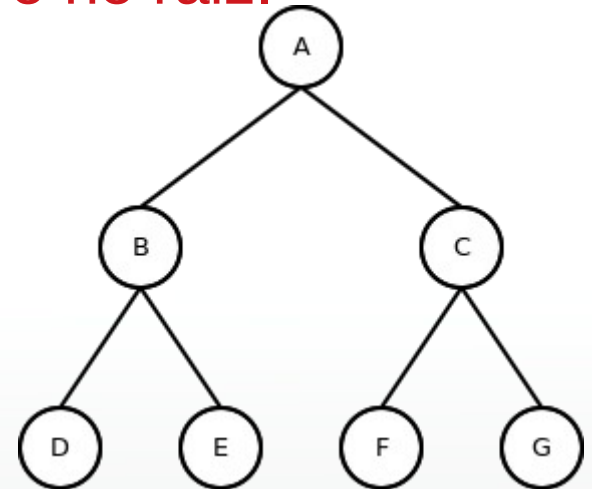
- Outra forma de armazenar uma árvore binária em um vetor é reservar as posições de acordo com o nível e descendência de cada nó.



# Uma Árvore Binária Diferente

- Outra forma de armazenar uma árvore binária em um vetor é reservar as posições de acordo com o nível e descendência de cada nó.
- O primeiro nó a ser armazenado é o nó raiz.

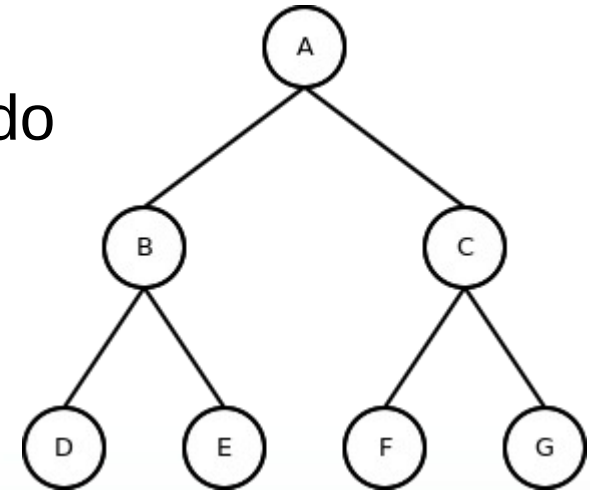
0	1	2	3	4	5	6	7	8	9	10
A										



# Uma Árvore Binária Diferente

- Nosso objetivo é indexar a árvore dentro do vetor.
- Dado o nó inserido, no caso o nó (A), inserimos os nós finos de (A) esquerdo depois o direito.

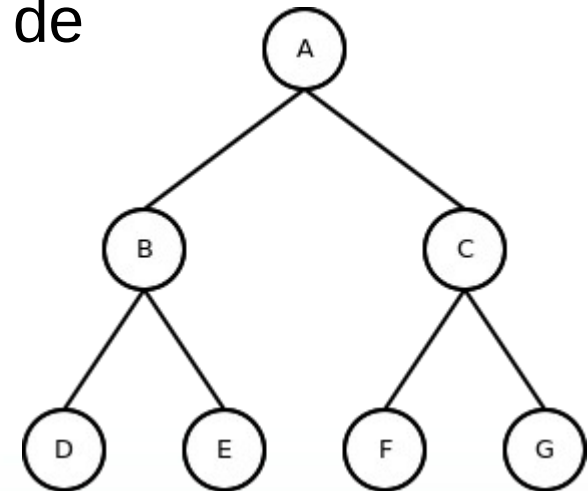
0	1	2	3	4	5	6	7	8	9	10
A										



# Uma Árvore Binária Diferente

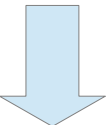
- Nosso objetivo é indexar a árvore dentro do vetor. Dado o nó inserido, no caso o nó (A), inserimos os nós finos de (A) esquerdo depois o direito.
- Nó filho (Esquerdo =  $p + 1$ )
- Nó filho (Direito =  $p + 2$ )

0	1	2	3	4	5	6	7	8	9	10
A	B	C								

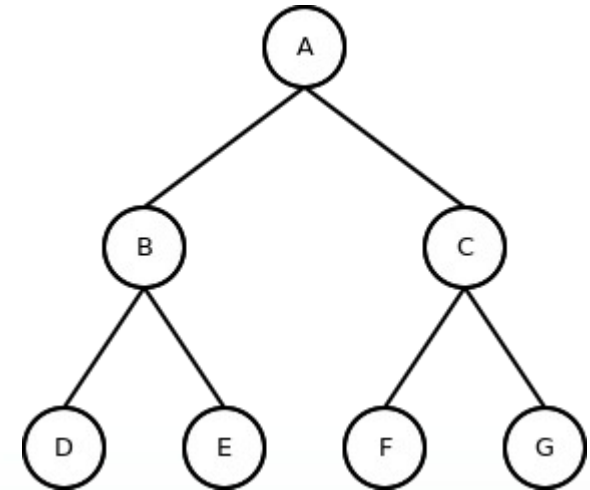


# Uma Árvore Binária Diferente

- Se aplicarmos a regra  $(p + 1)$  e  $(p + 2)$  para inserção dos nós filhos de (B) teremos um erros.
- Filhos da esquerda  $(1 + 1) = 2?$



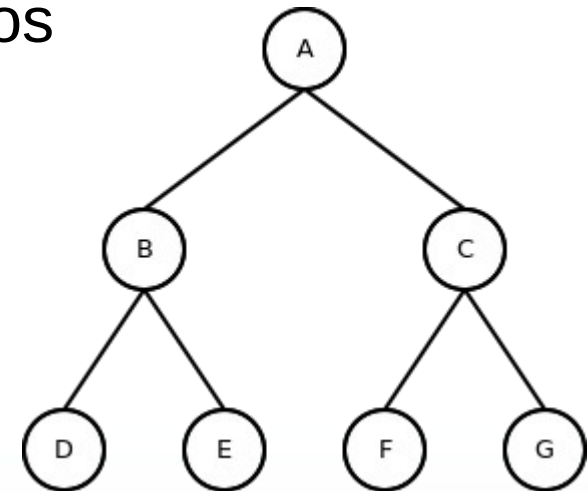
0	1	2	3	4	5	6	7	8	9	10
A	B	C								



# Uma Árvore Binária Diferente

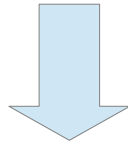
- Para solucionar o problema, alteramos a formula para  $(2 * p + 1)$  para os filhos da esquerda e  $(2 * p + 2)$  para os filhos da direita.

0	1	2	3	4	5	6	7	8	9	10
A	B	C								

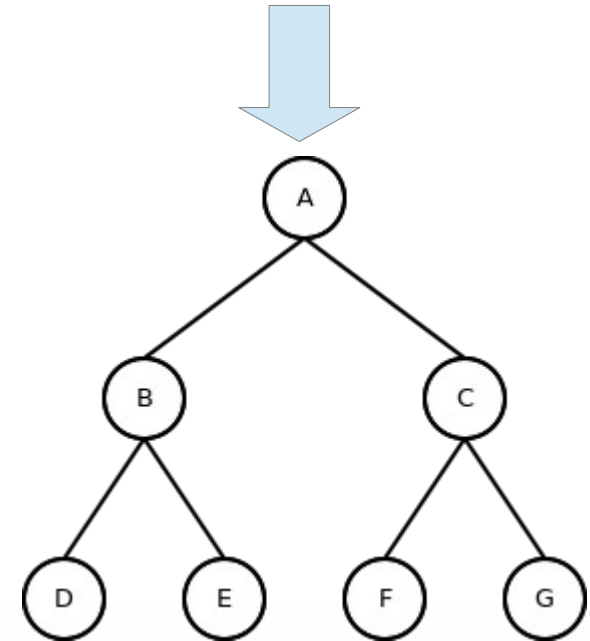


# Uma Árvore Binária Diferente

- Filhos de A
  - Esquerda  $(0 * 2 + 1) = 1$
  - Direita  $(0 * 2 + 2) = 2$



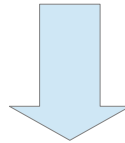
0	1	2	3	4	5	6	7	8	9	10
A	B	C								



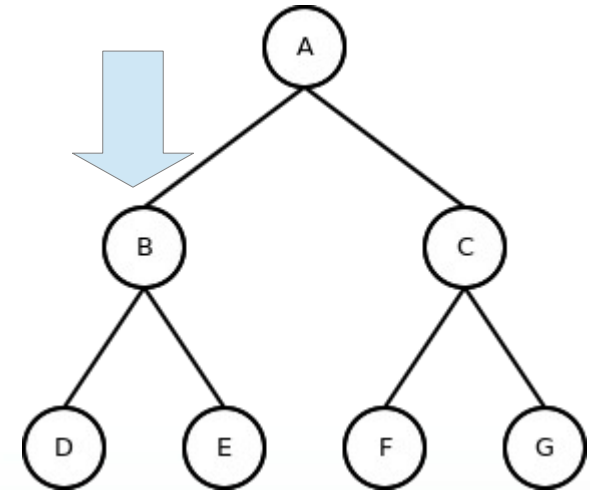


# Uma Árvore Binária Diferente

- Filhos de B
  - Esquerda  $(1 * 2 + 1) = 3$
  - Direita  $(1 * 2 + 2) = 4$



0	1	2	3	4	5	6	7	8	9	10
A	B	C	D	E						

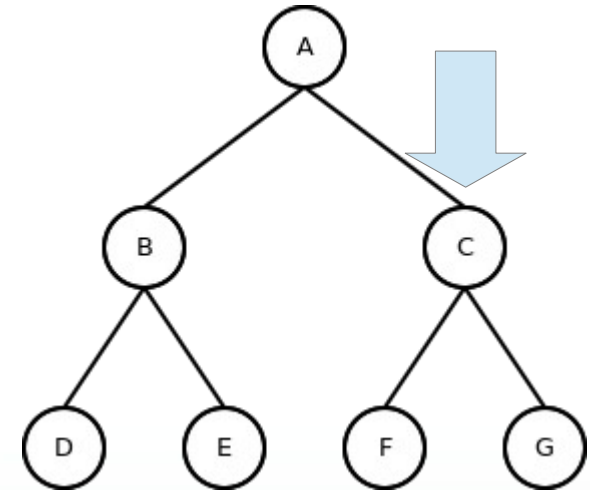


# Uma Árvore Binária Diferente

- Filhos de B
  - Esquerda  $(2 * 2 + 1) = 5$
  - Direita  $(2 * 2 + 2) = 6$

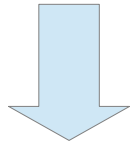


0	1	2	3	4	5	6	7	8	9	10
A	B	C	D	E	F	G				

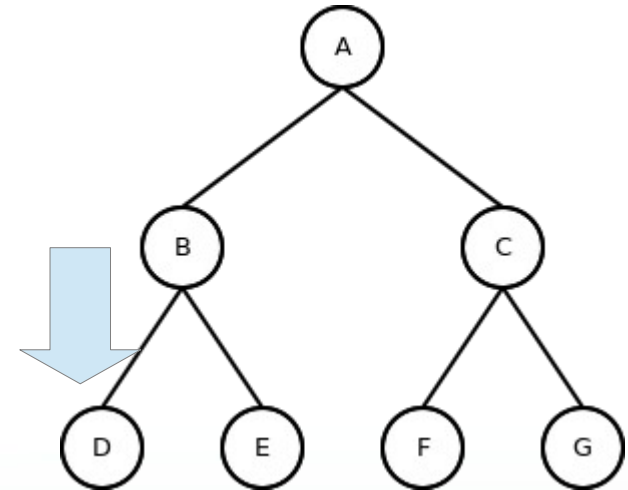


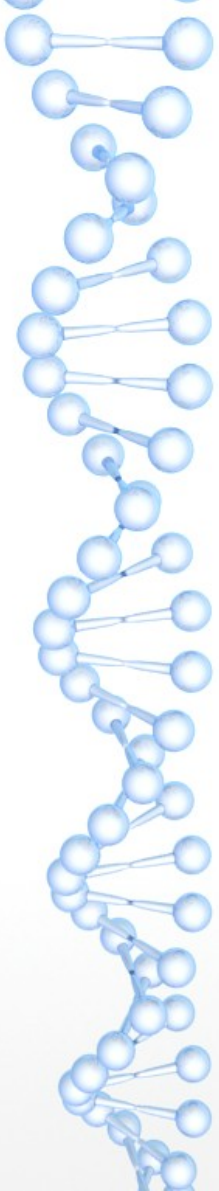
# Uma Árvore Binária Diferente

- Filhos de B
  - Esquerda  $(3 * 2 + 1) = 7$
  - Direita  $(3 * 2 + 2) = 8$



0	1	2	3	4	5	6	7	8	...	14
A	B	C	D	E	F	G	-	-	-	-



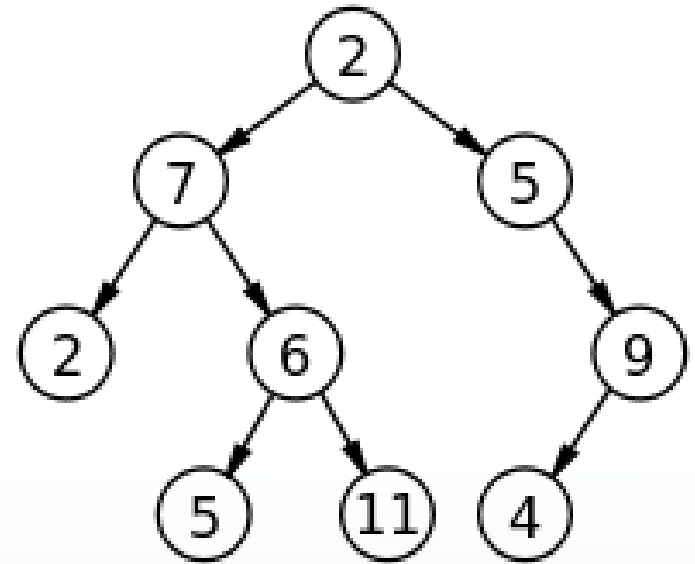


*“Só existem dois dias no ano que nada pode ser feito.  
Um se chama ontem e o outro se chama amanhã”*

**Dalai Lama**

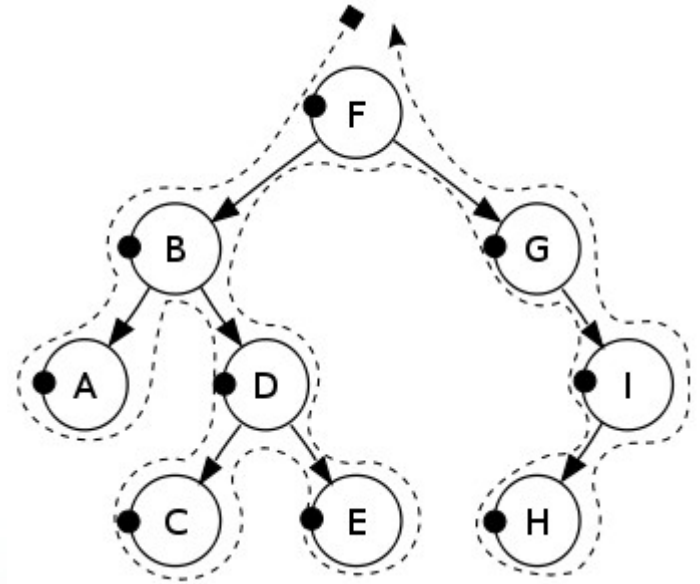
# Introdução

- Operações em árvores binárias
  - Busca
    - Ordem de visitas;
      - Pré-ordem
      - Em-ordem
      - Pós-ordem



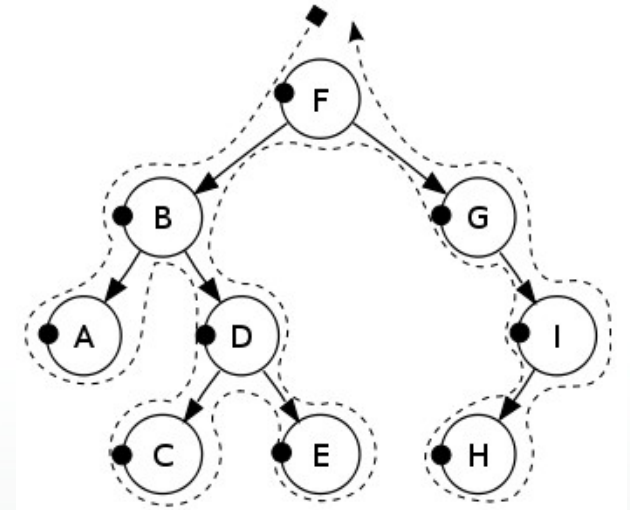
# Caminhando em Árvore Binária

- Dado o nó raiz, percorrer todos os nós de uma árvore binária.
  - Algoritmos clássicos.



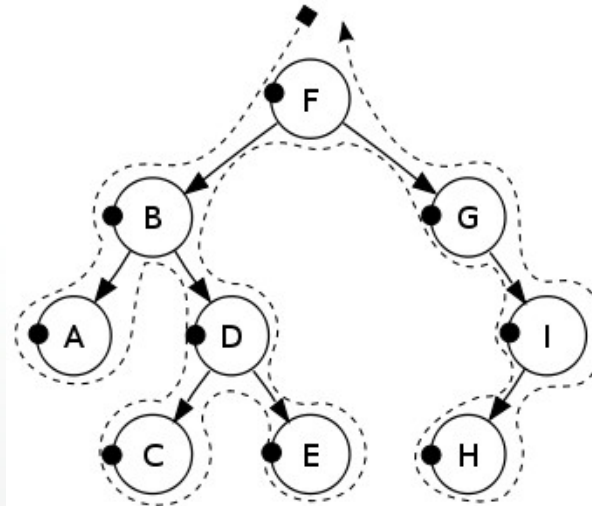
# Percurso Pré-Ordem

- Também conhecido como caminho pré fixado;
  - Marca a raiz como visitada, e depois
  - Visitamos as subárvores a esquerda e direita, respectivamente.



# Percurso Pré-Ordem

```
void exibirPreOrdem(noh **n){  
    if(*n != NULL){  
        printf("\n%d", (*n)->valor);  
        exibirPreOrdem(&((*n)->filhoEsquerda));  
        exibirPreOrdem(&((*n)->filhoDireita));  
    }  
}
```





# Percurso Em-Ordem

- Também conhecido como inter-fixado

```
void exibirEmOrdem(noh **n){  
    if(*n != NULL){  
        exibirEmOrdem(&((*n)->filhoEsquerda));  
        printf("\n%d", (*n)->valor);  
        exibirEmOrdem(&((*n)->filhoDireita));  
    }  
}
```



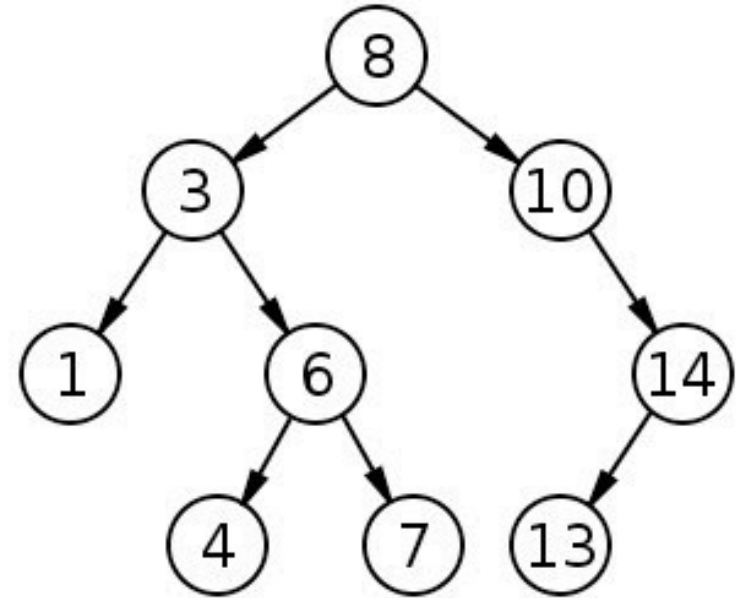
# Percurso Pós-Ordem

- Também conhecido como pós-fixado

```
void exibirPosOrdem(noh **n){  
    if(*n != NULL){  
        exibirPosOrdem(&((*n)->filhoEsquerda));  
        exibirPosOrdem(&((*n)->filhoDireita));  
        printf("\n%d", (*n)->valor);  
    }  
}
```

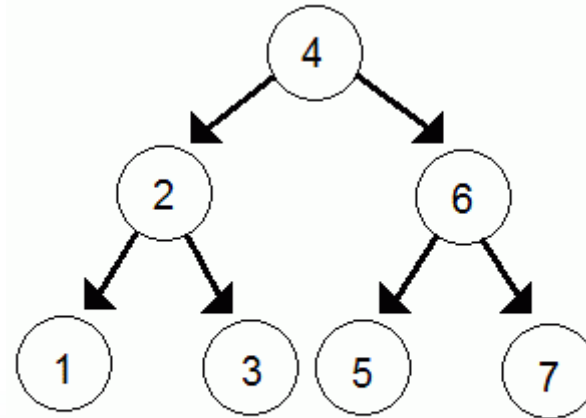
# Exercícios

- Crie uma árvore binária
  - Cadastre os números.
  - Exiba os dados
    - Em-Ordem
    - Pré-Ordem
    - Pós-Ordem



# Busca em Árvores Binárias

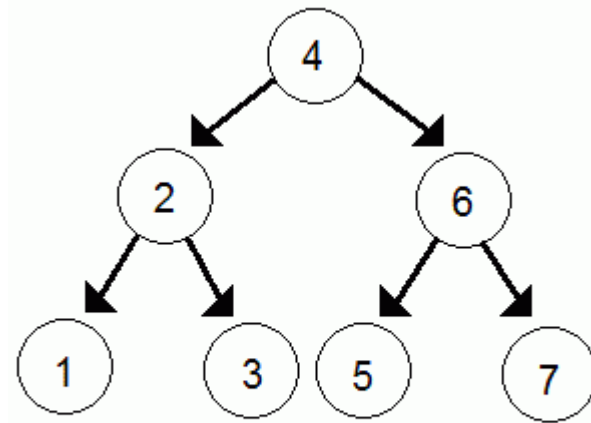
- Árvores binárias
  - Busca com tempo relativamente curto.



# Busca em Árvores Binarias

## Estrutura dinâmica

```
void busca(noh **n, int valor){
    if(*n != NULL){
        if ((*n)->valor == valor){
            printf("\nAchou --> %d \n", (*n)->valor);
        }else{
            if (valor < (*n)->valor){
                busca(&((*n)->filhoEsquerda), valor);
            }else{
                busca(&((*n)->filhoDireita), valor);
            }
        }
    }else{
        printf("\n Valor não encontrado! \n");
    }
}
```

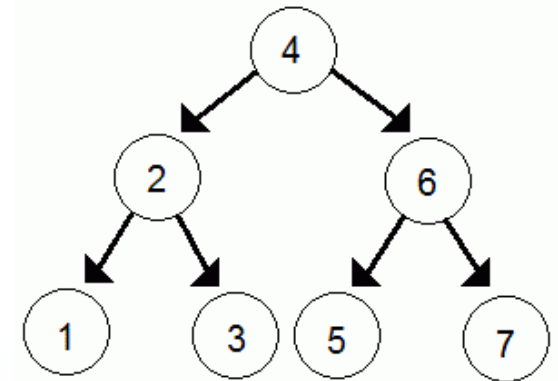


# Busca em Árvores Binárias

## Estrutura estática (Vetor)

- Utilizando a regra apresentada na aula anterior, utilizando um vetor para armazenar os dados da árvore, como ficaria a busca para o exemplo abaixo:

4	2	6	1	3	5	7	-	-	-
---	---	---	---	---	---	---	---	---	---



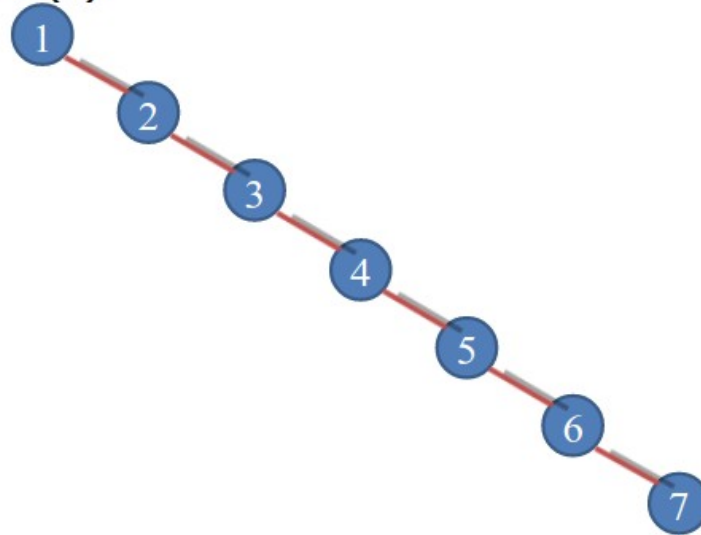


# Atividade

- Crie um programa utilizando a estrutura de uma árvore binária que contenha:
  - Inserção
  - Busca
  - Exibição (Em-ordem | Pré-ordem | Pós-ordem)
  - \*\*\* Remoção (Pensa em uma estratégia para remover um elemento da árvore).

# Árvore AVL

- Próxima aula.
  - A: Inserção de 1, 2, 3, 4, 5, 6 e 7
  - Pior caso:  $O(n)$







# Árvore AVL

- Árvore balanceada (árvore completa) são as árvores que minimizam o número de comparações efetuadas no pior caso para uma busca com chaves de probabilidades de ocorrências idênticas.
- O nome AVL vem de seus criadores soviéticos Adelson Velsky e Landis, e sua primeira referência encontra-se no documento "Algoritmos para organização da informação" de 1962.



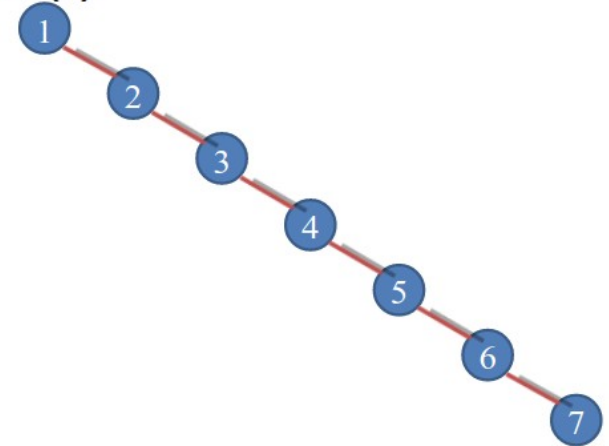
# Complexidade

	<b>Média</b>	<b>Pior Caso</b>
<b>Espaço</b>	$O(n)$	$O(n)$
<b>Busca</b>	$O(\log n)$	$O(\log n)$
<b>Inserção</b>	$O(\log n)$	$O(\log n)$
<b>Deleção</b>	$O(\log n)$	$O(\log n)$

# Árvore AVL

- Árvore desbalanceada.
  - Busque o número 7.
    - Percorre todos os elementos.
- Solução
  - Balanceamento.
    - Reorganizar os elementos de maneira a minimizar sua altura.

- A: Inserção de 1, 2, 3, 4, 5, 6 e 7
- Pior caso:  $O(n)$





# Árvore AVL

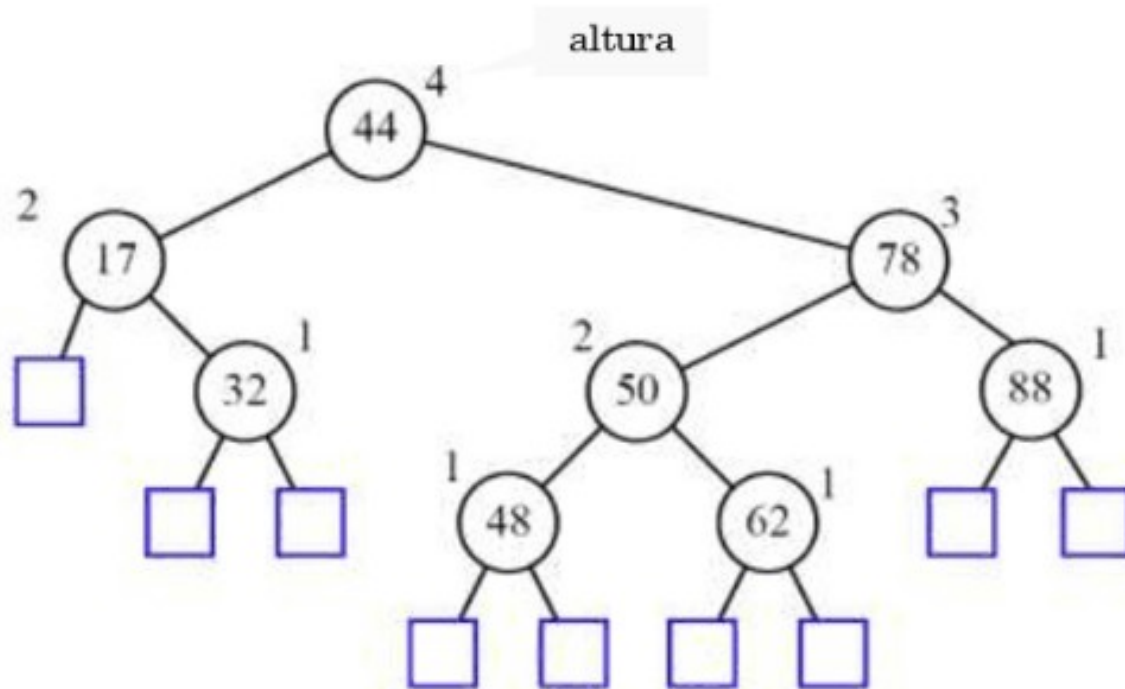
- O conceito de balanceamento está diretamente ligado à altura das subárvores.
  - Recordando, altura ou profundidade de uma árvore é igual ao número de nós visitados desde a raiz até o nó folha mais distante.



# Árvore AVL

- Objetivos:
  - Remover o problema de pior caso.
- Uso de invariantes
  - Balanceamento
    - A altura das subárvores diferem no máximo em 1.

# Árvore AVL





# Árvore AVL

- Cada nó em uma árvore binária balanceada (AVL) tem balanceamento de -1, 0 ou +1.
- Se o valor for diferente, a árvore não é balanceada.



# Árvore AVL

## Fator de Balanceamento (FB)

- Para cada nó, define-se um fator de balanceamento (FatBal) -1, 0 ou +1.
  - FatBal = -1, quando a subárvore da direita é um nível mais alto que a esquerda.
  - FatBal = 0, quando as duas subárvores tem a mesma altura.
  - FatBal = 1. quando a subárvore da esquerda é um nível mais alto que a direita.





# Árvore AVL

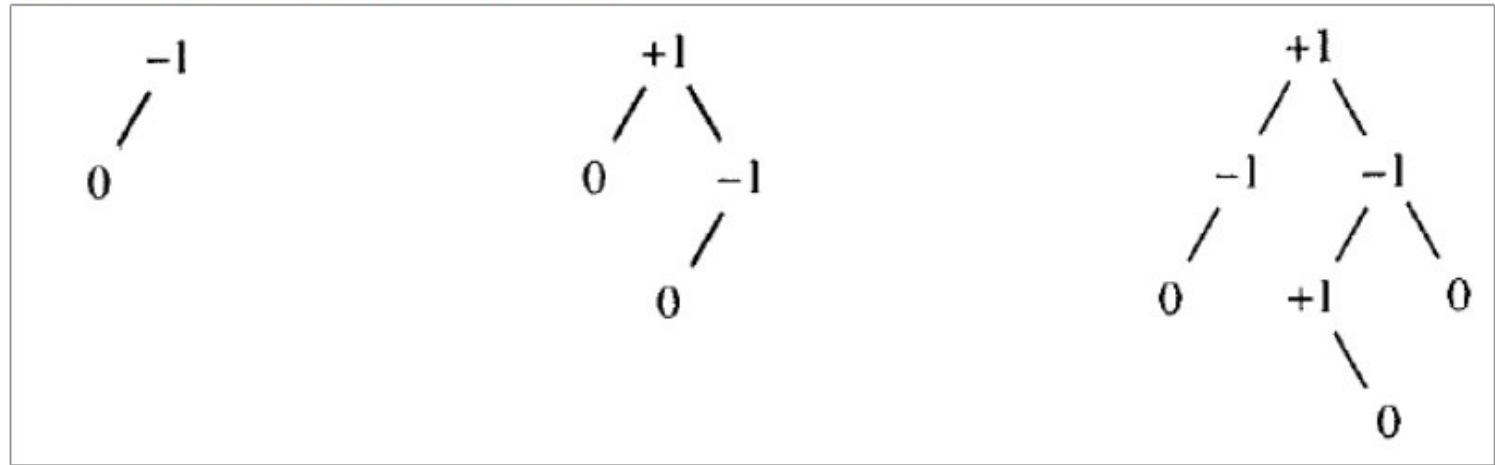
## Fator de Balanceamento (FB)

- Quando é usado o FatBal.
  - Inserção ou Remoção de um novo nó.

# Árvore AVL

- Em uma árvore AVL, para todo nó, seja  $hd$  a altura de uma subárvore direita e  $he$  a altura de uma subárvore esquerda de um nó:

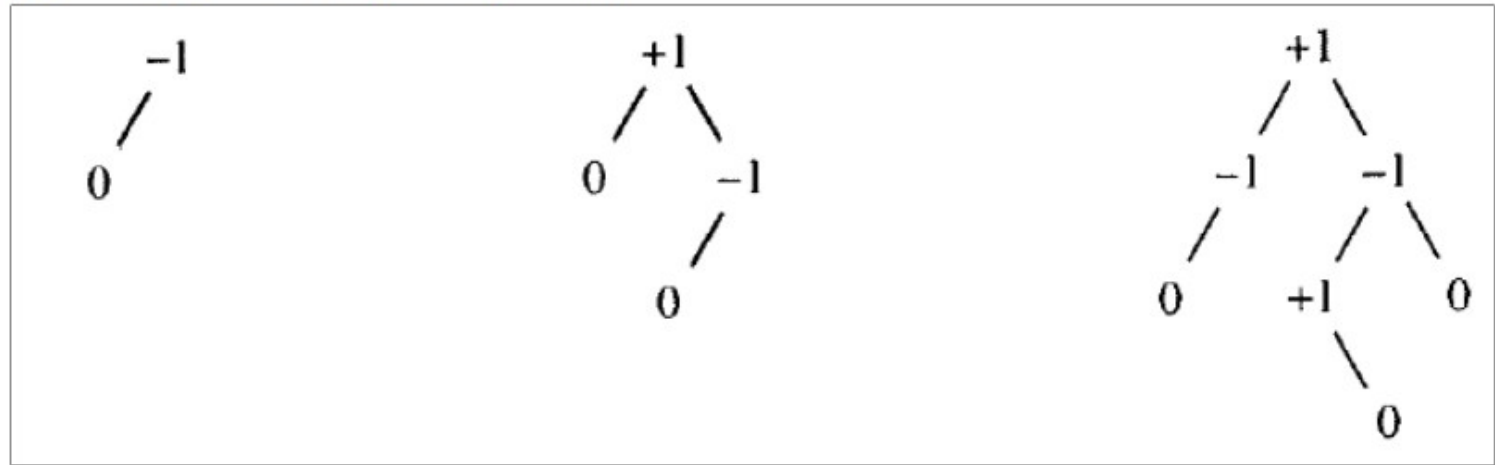
$hd - he \in \{0, 1, -1\}$



# Árvore AVL

- Se o fator de balanceamento de qualquer nó ficar menor do que -1 ou maior do que 1 então a árvore tem que ser balanceada

$hd - he \in \{0, 1, -1\}$



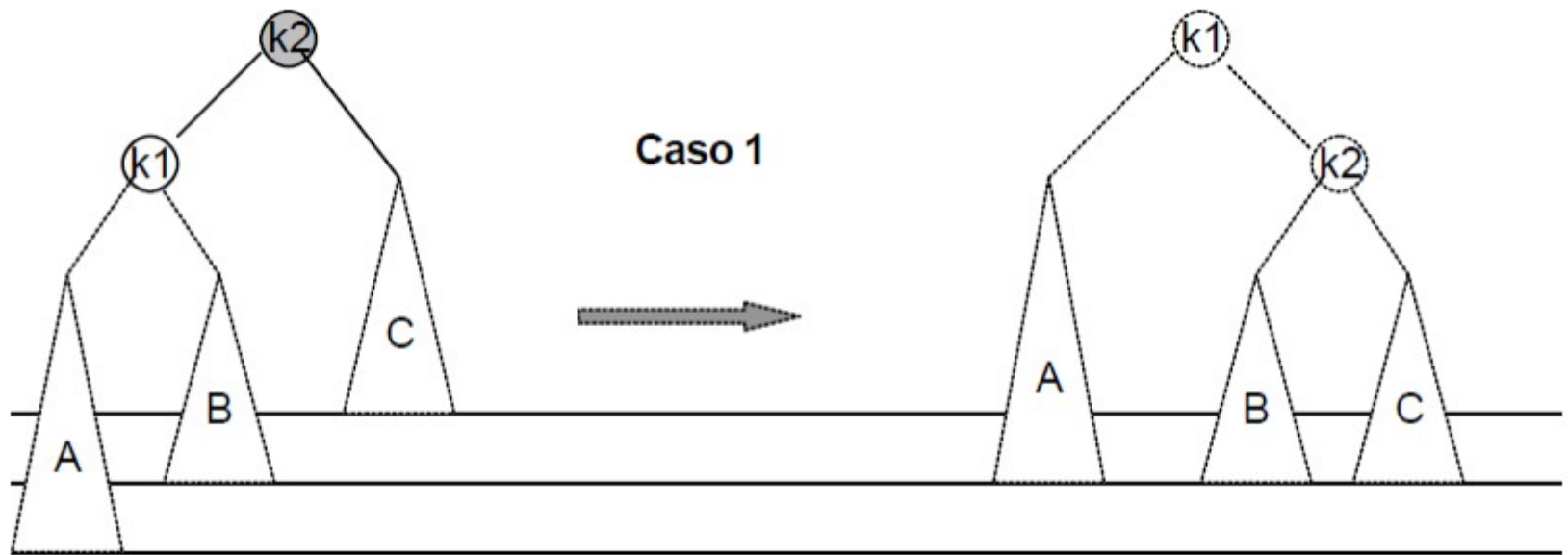


# Rotações em AVL

- Na inserção utiliza-se um processo de balanceamento que pode ser de 2 tipos gerais:
  - Rotação simples
  - Rotação dupla

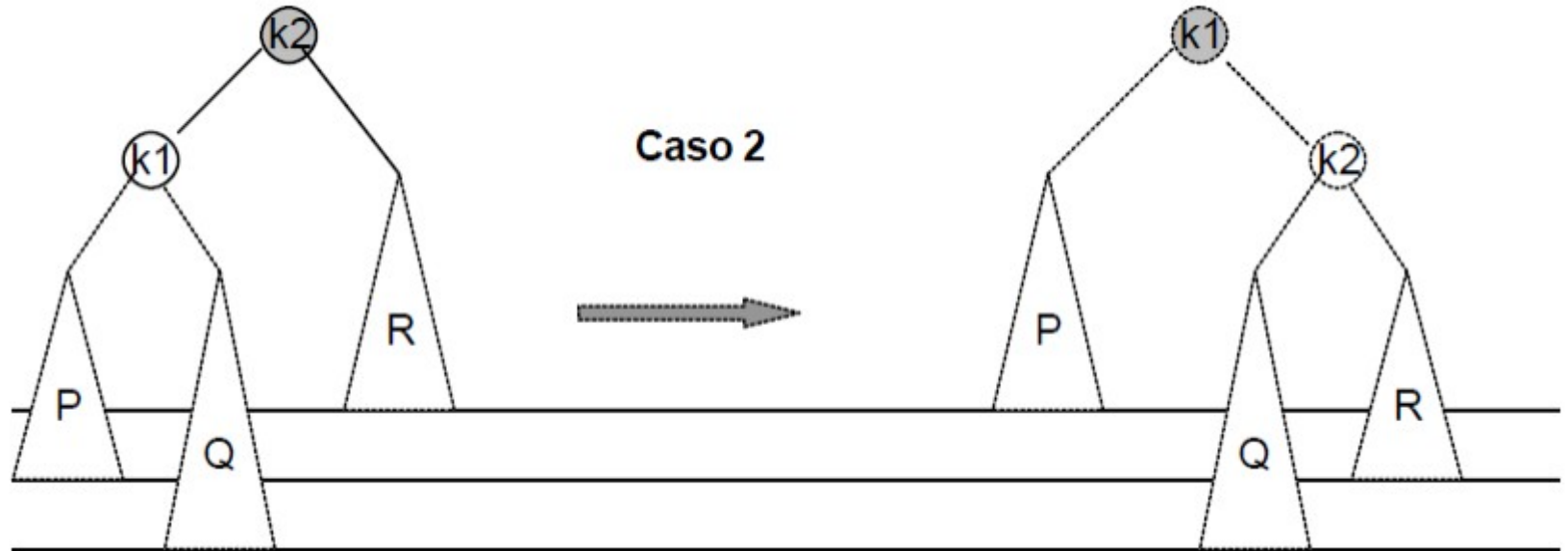
# Rotações simples

- k2 é nó mais profundo onde falha o equilíbrio sub-árvore esquerda está 2 níveis abaixo da direita



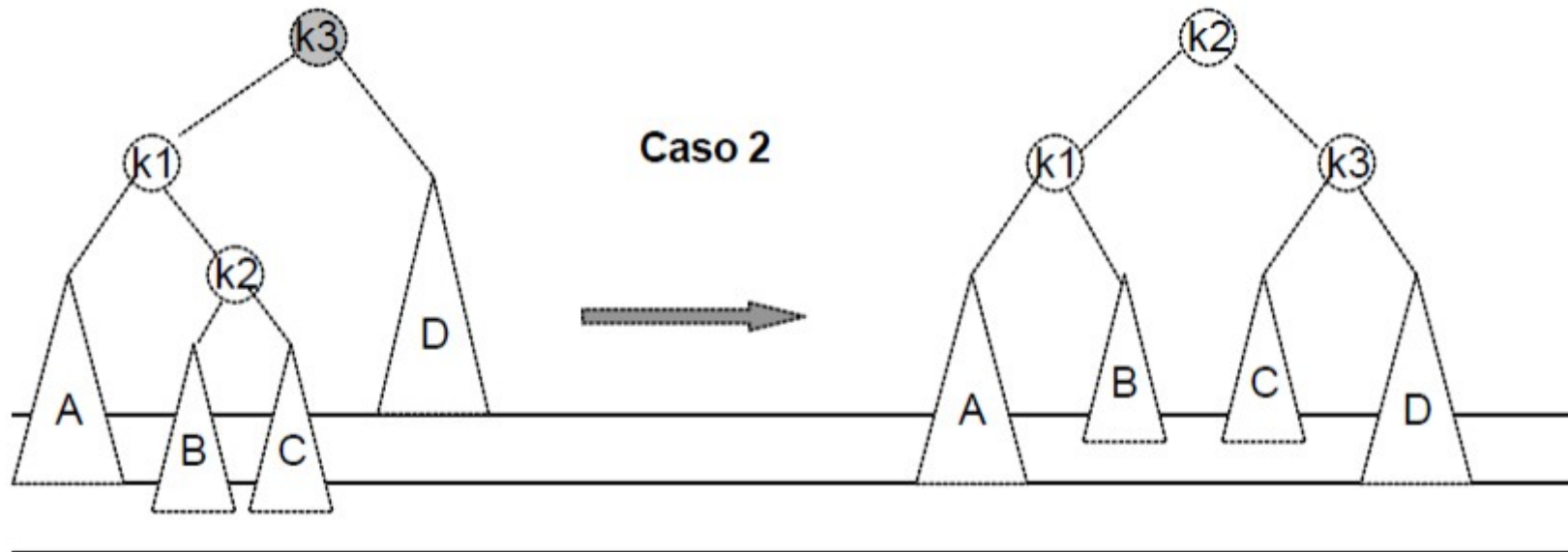
# Rotações dupla

- Rotação simples não resolve o desequilíbrio!
  - sub-árvore Q está a 2 níveis de diferença de R
  - sub-árvore Q passa a estar a 2 níveis de diferença de P



# Rotações dupla

- Uma das subárvores B ou C está 2 níveis abaixo de D
- $k_2$ , a chave intermédia, fica na raiz
- posições de  $k_1$ ,  $k_3$  e subárvores completamente determinadas pela ordenação



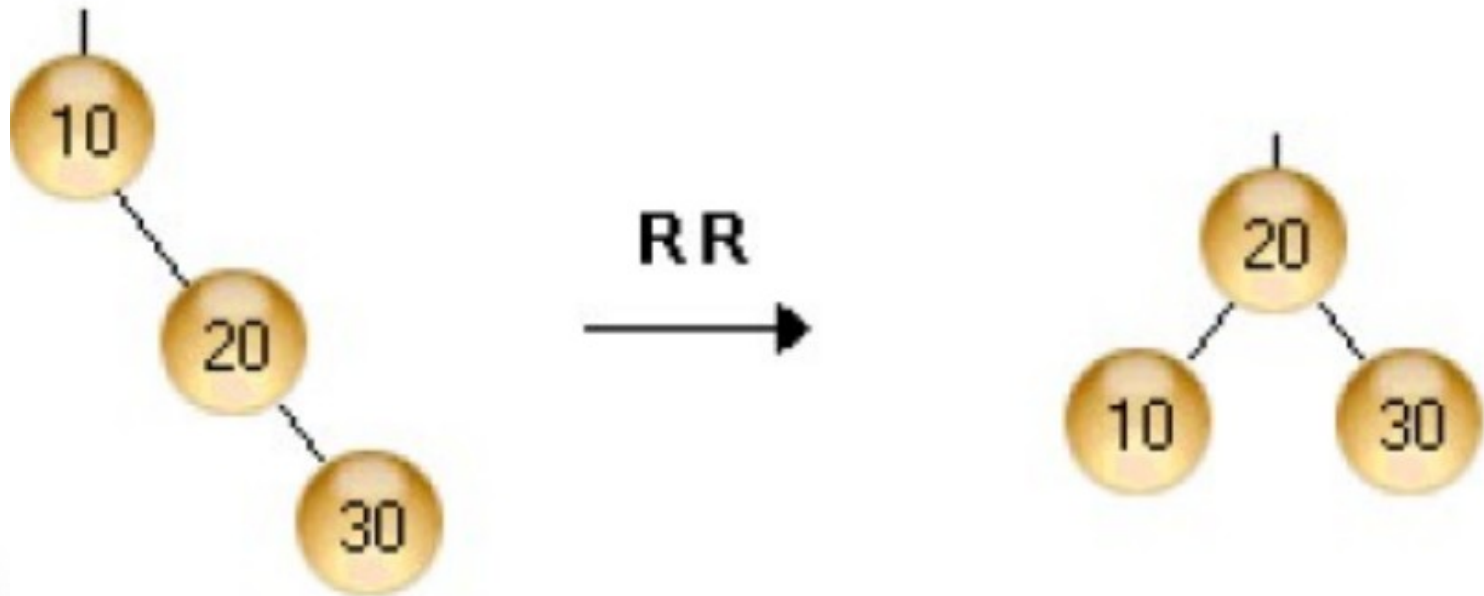


# Rotações em AVL

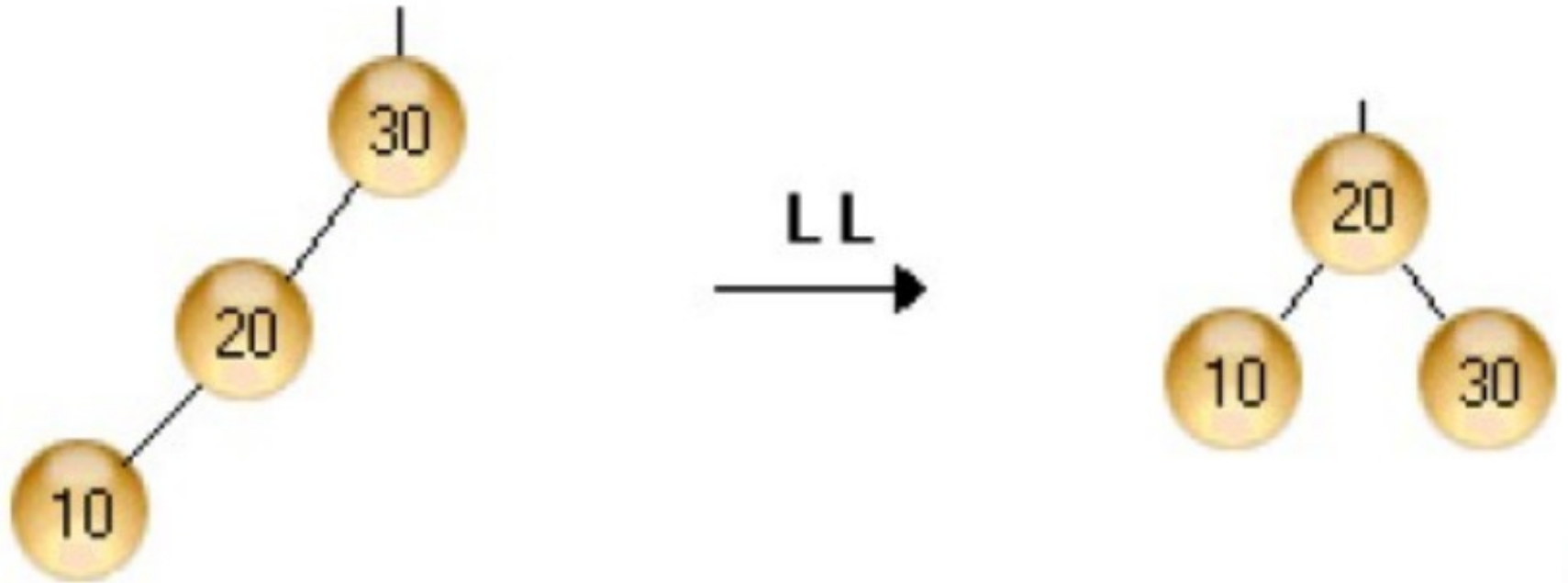
- Na inserção utiliza-se um processo de balanceamento que pode ser de 4 tipos específicos:
  - RR → caso Right-Right (rotação a esquerda)
  - LL → caso Left-Left (rotação a direita)
  - LR → caso Left-Right (rotação esquerda-direita)
  - RL → caso Right-Left (rotação direita-esquerda)



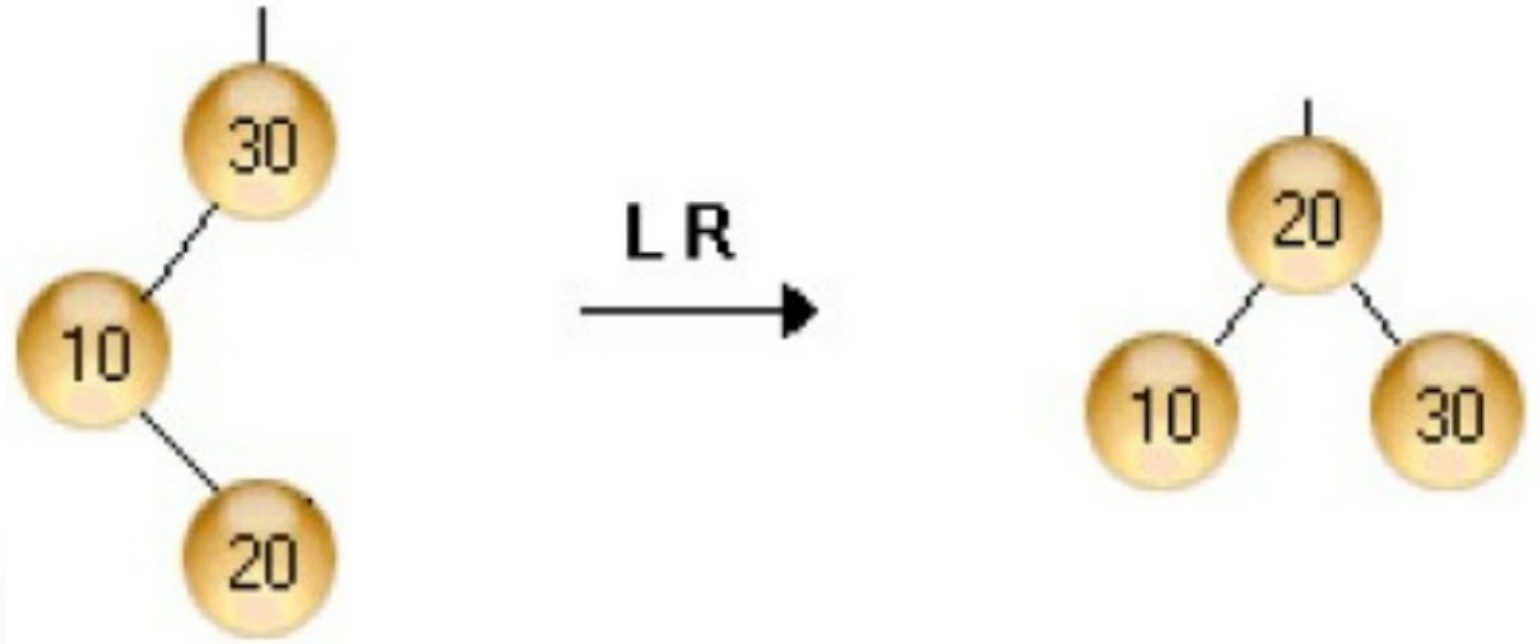
## Caso Right-Right (rotação a esquerda)



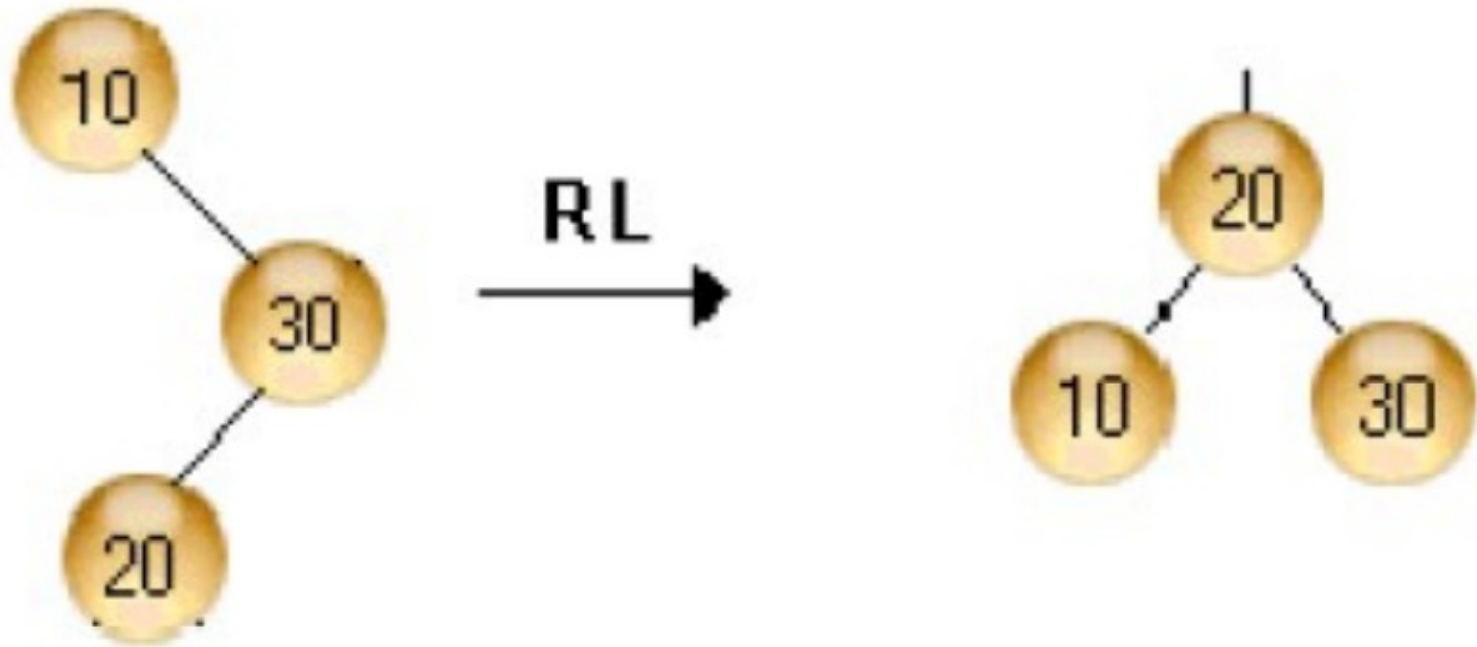
## Caso Left-Left (rotação a direita)



# Left-Right (rotação esquerda-direita)



## Caso Right-Left (rotação direita-esquerda)

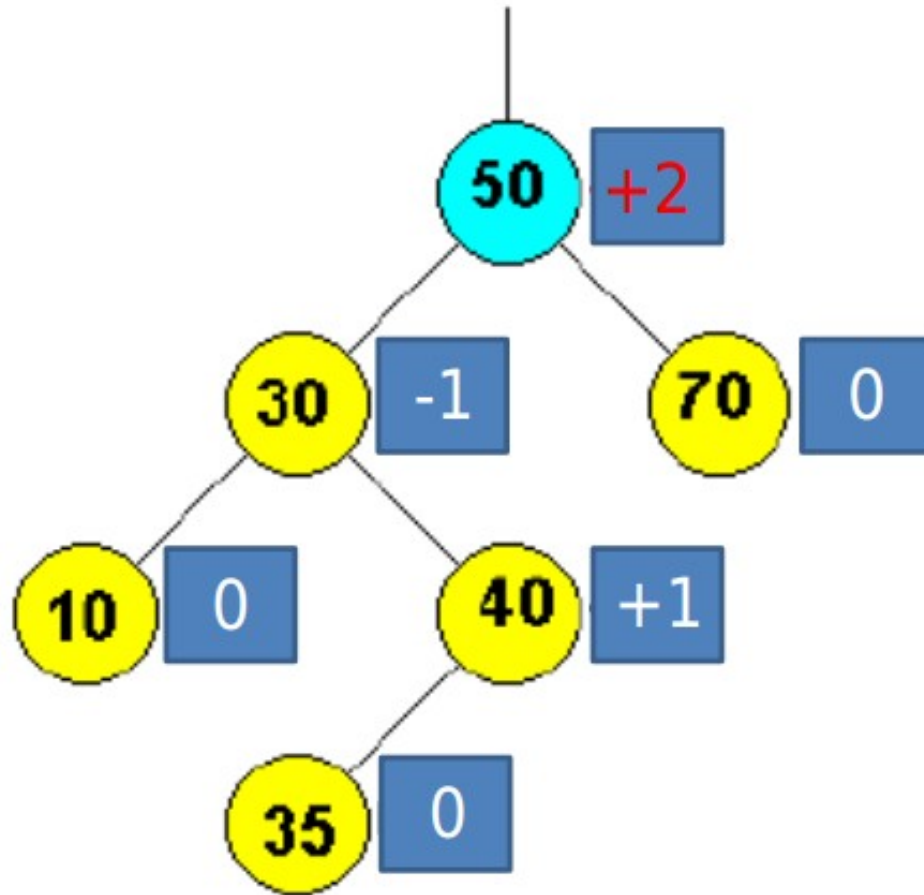




# Fator de Balanceamento

- Coeficiente que serve como referência para verificar se uma árvore AVL está ou não balanceada
- O fator é calculado nó a nó e leva em consideração a diferença das alturas das sub-árvores da direita e da esquerda
- Genericamente
  - $FB = h_e - h_d$

## Exemplo – FB de cada nó





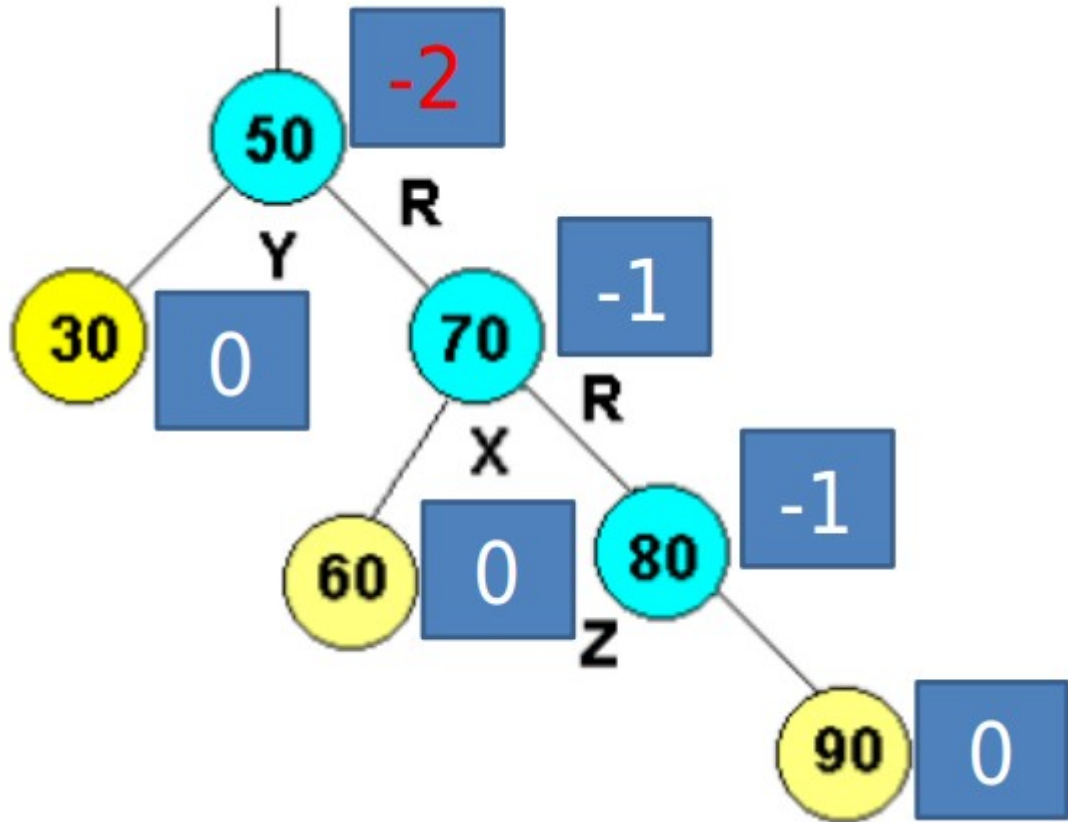
# Quando balancear?

- Sempre que existir um fator de balanceamento superior a +1 ou inferior a -1
- Caso exista mais de um nó que se encaixe neste perfil deve-se sempre balancear o nó com o nível mais alto
- Como balancear? Utilizando os processos:
  - Right-Right
  - Left-Left
  - Left-Right
  - Right-Left



# Tipos de Balanceamento - RR

- Suponha na figura que a última célula a ser inserida foi a célula de chave 90



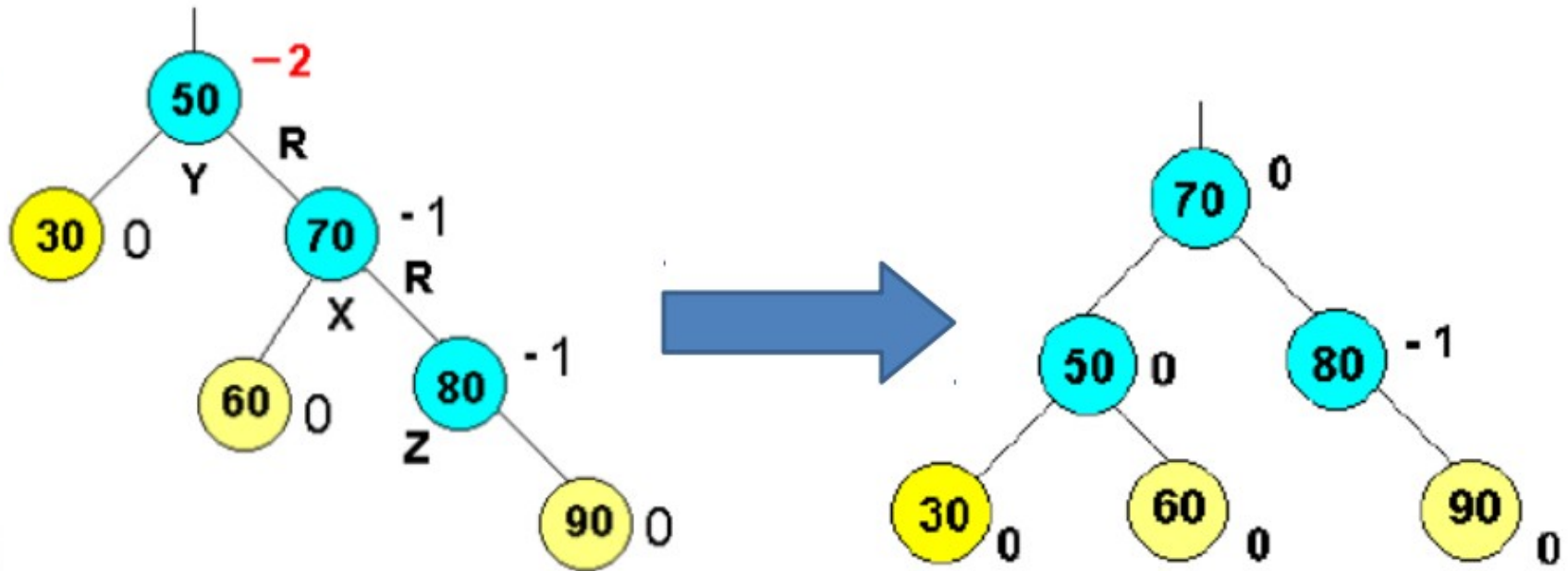




## Tipos de Balanceamento - RR

- O nó X que está no nível do meio dos três envolvidos toma o lugar do nó com  $FB=-2$
- A sub-árvore direita do nó X permanece
- A sub-árvore esquerda do nó X será colocada como sub-árvore direita do nó Y
- O filho esquerdo do nó X aponta para o nó Y

# Tipos de Balanceamento - RR



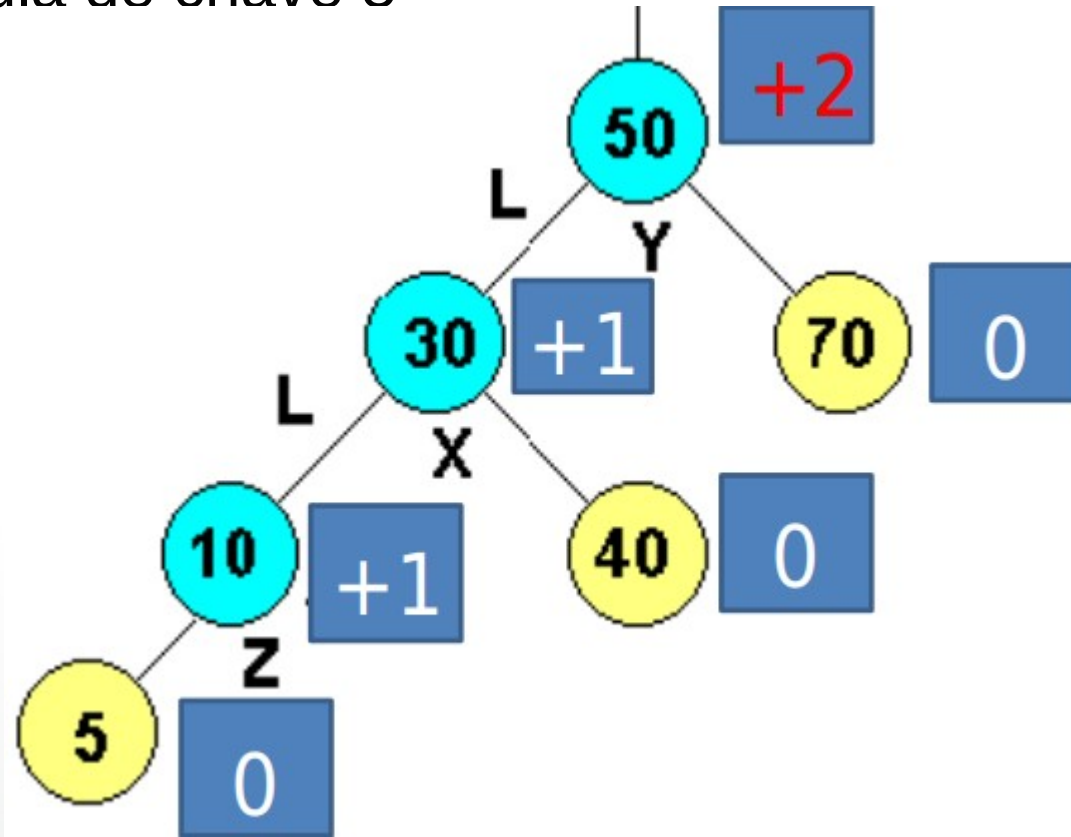


# Tipos de Balanceamento - RR

```
static Node rotacaoRR(Node y) {  
    Node x = y.right;  
    y.right = x.left;  
    x.left = y;  
    return x;  
}
```

# Tipos de Balanceamento - LL

- Suponha na figura que a última célula a ser inserida foi a célula de chave 5

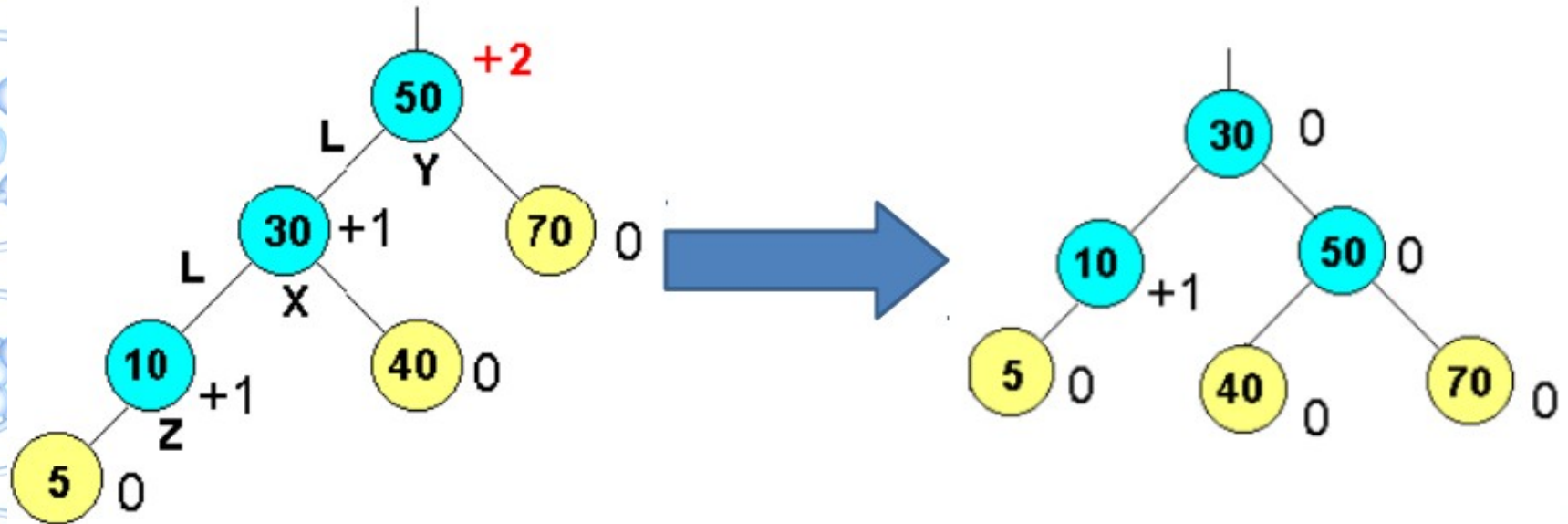




## Tipos de Balanceamento - LL

- O nó X que está no nível do meio dos três envolvidos toma o lugar do nó com  $FB=-2$
- A sub-árvore esquerda do nó X permanece
- A sub-árvore direita do nó X será colocada como sub-árvore esquerda do nó Y
- O filho direito do nó X aponta para o nó Y

# Tipos de Balanceamento - LL



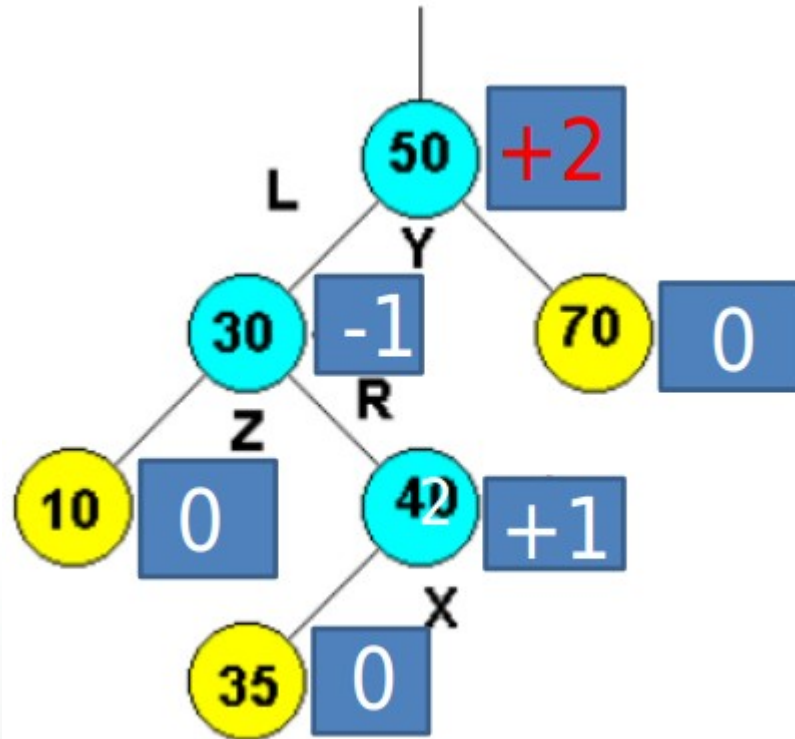


# Tipos de Balanceamento - LL

```
static Node rotacaoLL(Node y) {  
    Node x = y.left;  
    y.left = x.right;  
    x.right = y;  
    return x;  
}
```

# Tipos de Balanceamento - LR

- Suponha na figura que a última célula a ser inserida foi a célula de chave 35



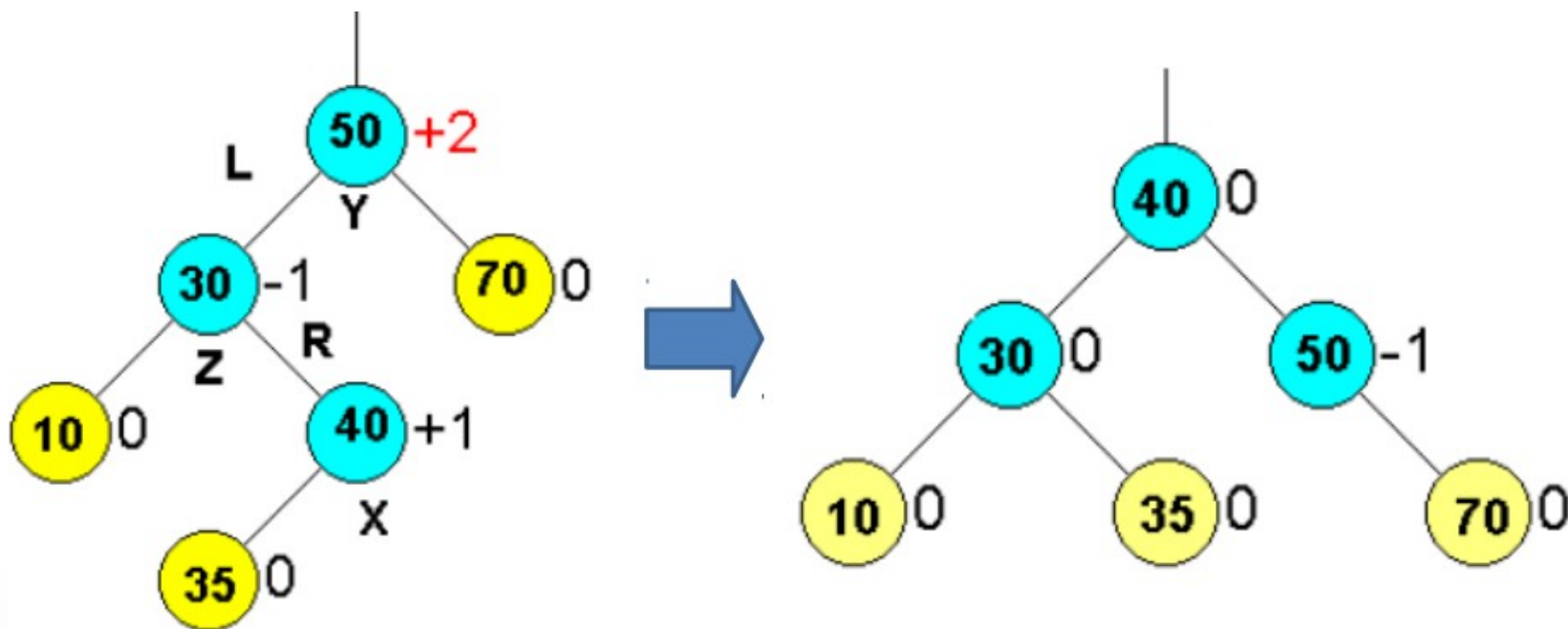




# Tipos de Balanceamento - LR

- O nó que está no nível mais alto das três envolvidas (nó X) toma o lugar da célula cujo fator de balanceamento é +2 (nó Y)
- A sub-árvore direita do nó X será colocada como sub-árvore esquerda do nó Y
- A sub-árvore esquerda do nó X será colocada como sub-árvore direita do nó Z
- O filho direito do nó X aponta para o nó Y
- O filho esquerdo do nó X aponta para o nó Z

# Tipos de Balanceamento - LR



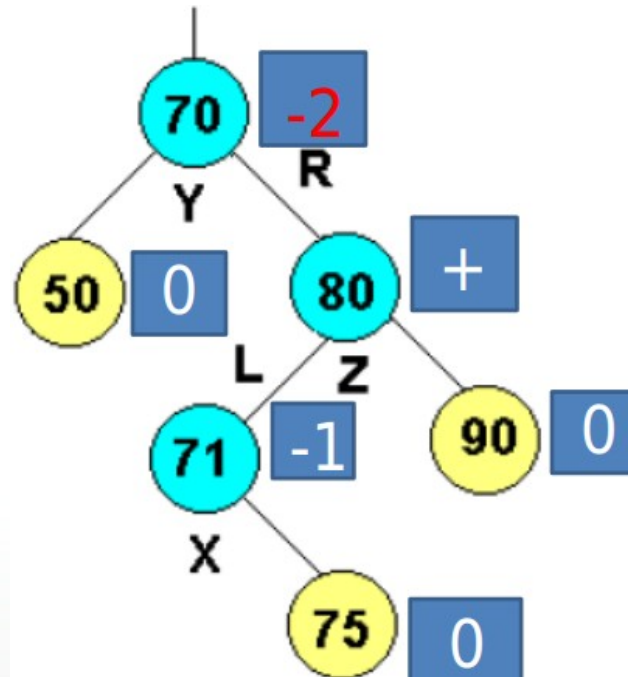


# Tipos de Balanceamento - LR

```
static Node rotacaoLR(Node y) {  
    y.left = rotacaoRR(y.left);  
    return rotacaoLL(y);  
}
```

# Tipos de Balanceamento - RL

- Suponha na figura que a última célula a ser inserida foi a célula de chave 75

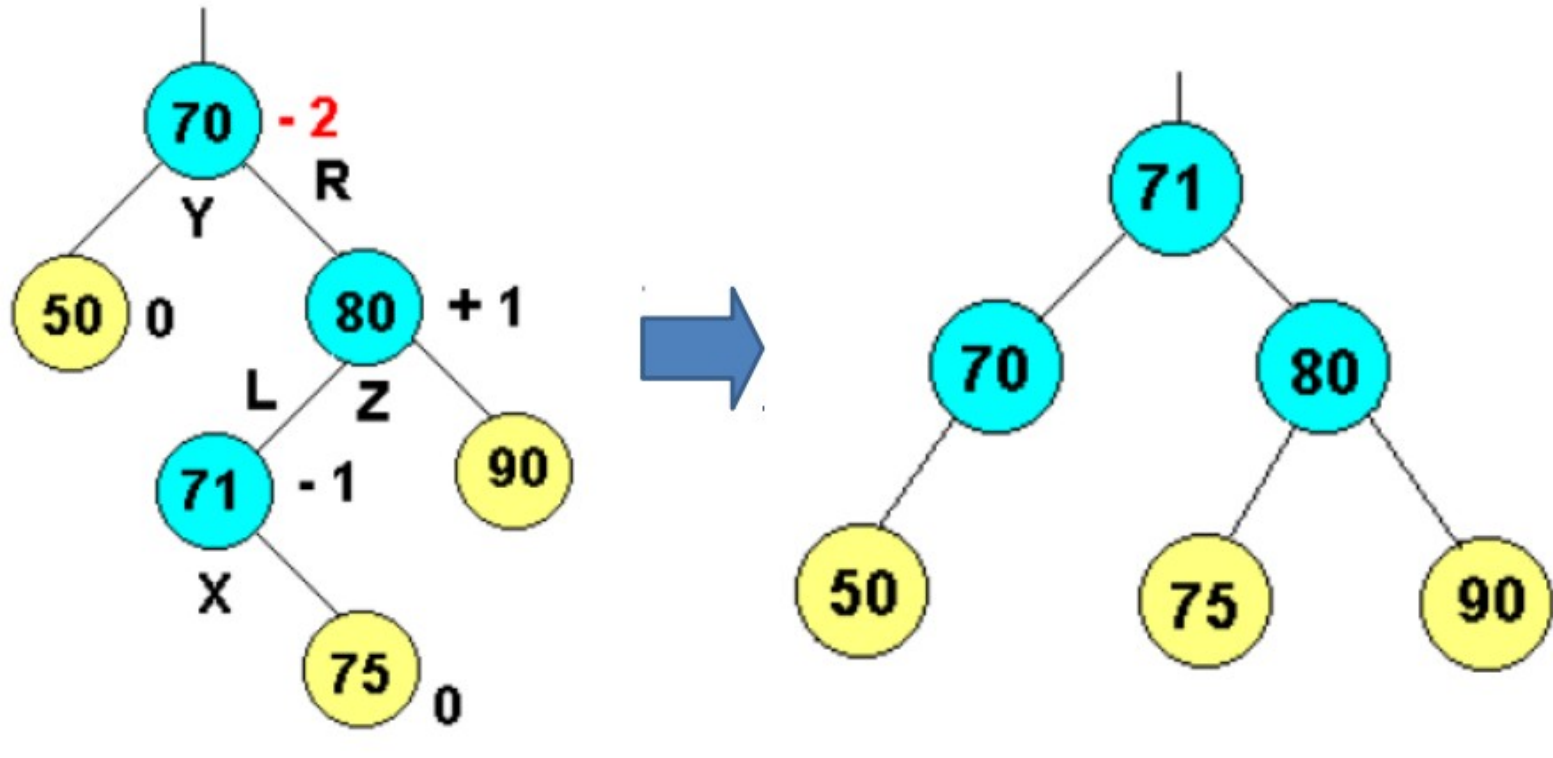




# Tipos de Balanceamento - RL

- O nó que está no nível mais alto das três envolvidas (nó X) toma o lugar da célula cujo fator de balanceamento é -2 (nó Y)
- A sub-árvore direita do nó X será colocada como sub-árvore esquerda do nó Z
- A sub-árvore esquerda do nó X será colocada como sub-árvore direita do nó Y
- O filho direito do nó X aponta para o nó Y
- O filho esquerdo do nó X aponta para o nó Z

# Tipos de Balanceamento - RL



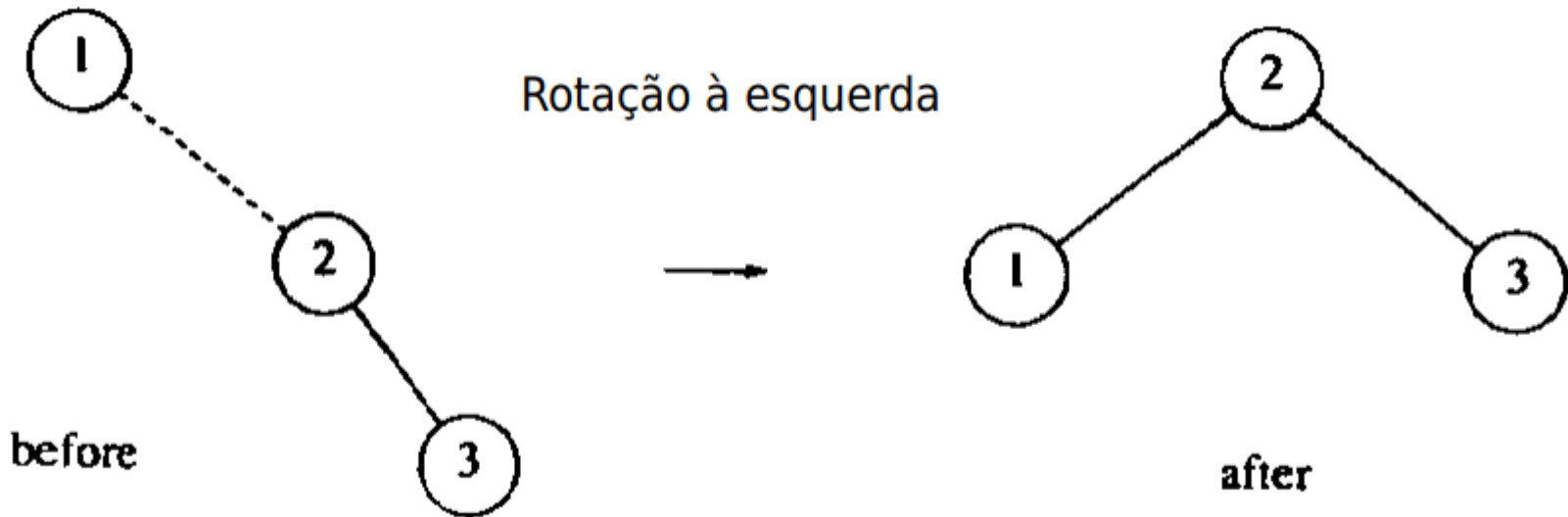


# Tipos de Balanceamento - RL

```
static Node rotacaoRL(Node y){  
    y.right = rotacaoLL(y.right);  
    return rotacaoRR(y);  
}
```

# Árvore AVL

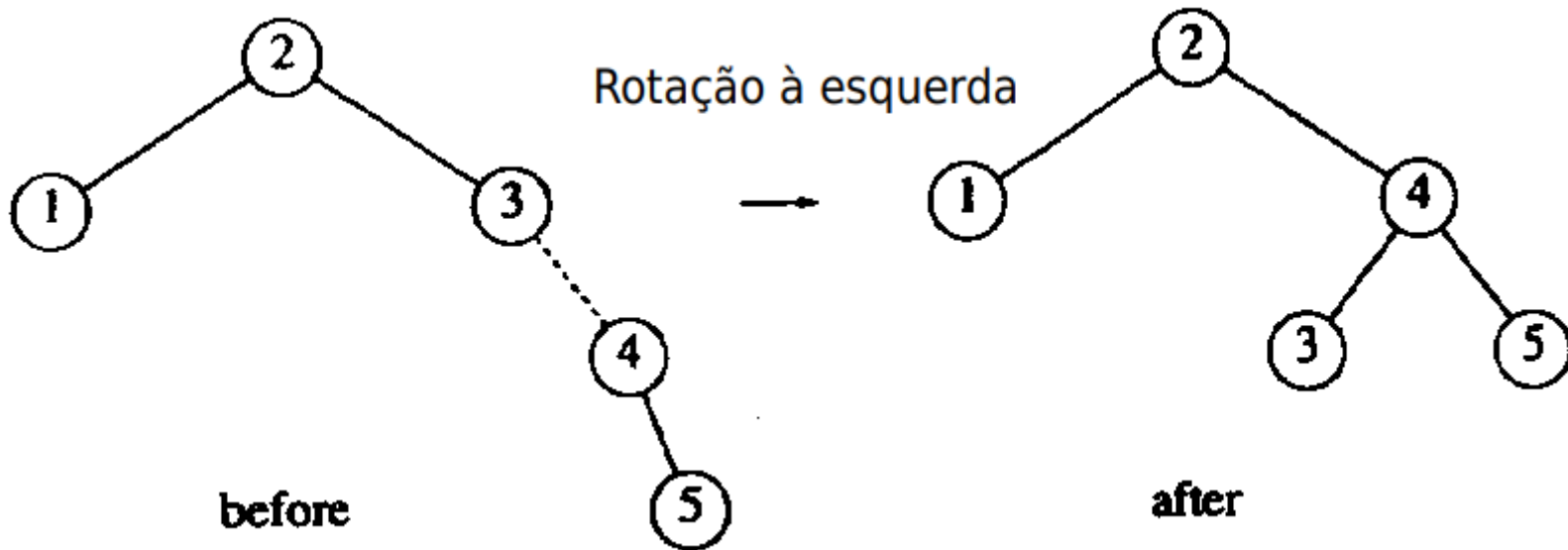
- Inserção de 1, 2 e 3.
- Ao inserir 3, o nó raiz fica desbalanceado (+2)





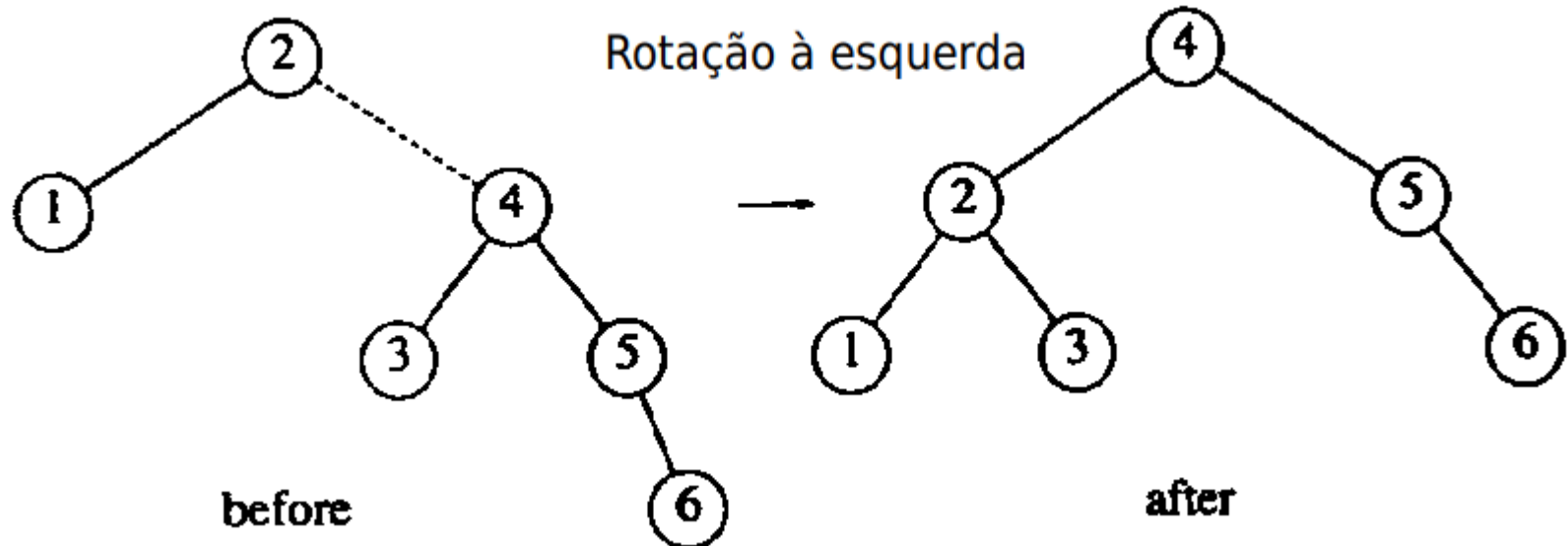
# Árvore AVL

- Inserção do 4 e 5.
- 4: sem problemas
- 5: desbalanceamento do nó 3 (+2)



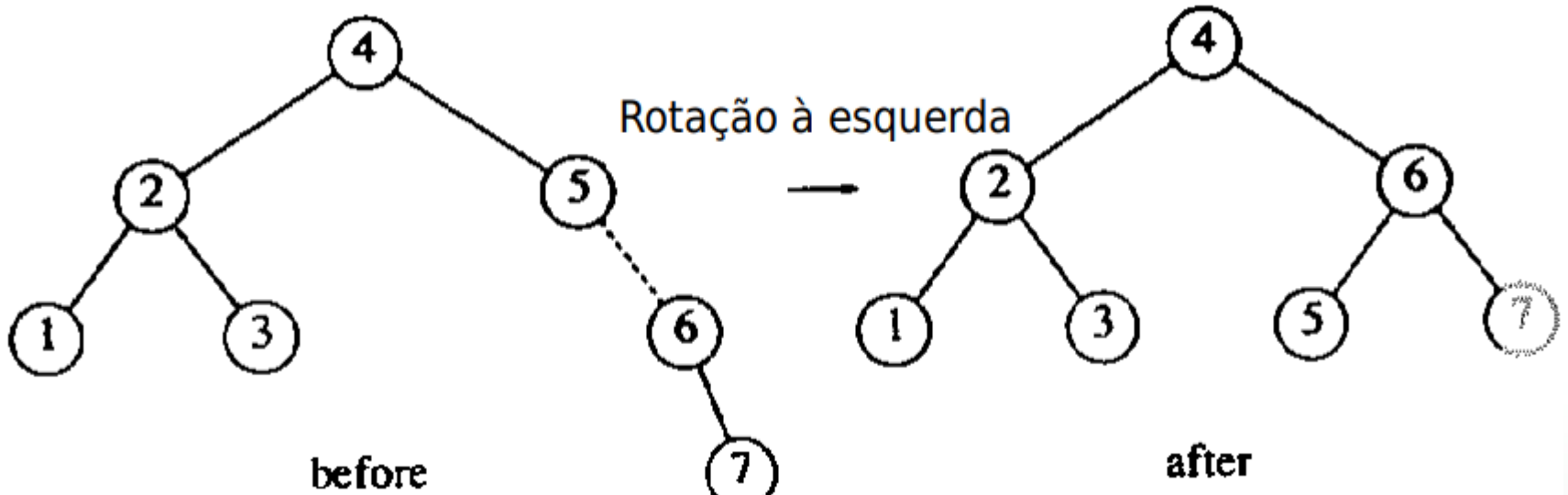
# Árvore AVL

- Inserção do 6. Nó 2 fica desbalanceado (+2)



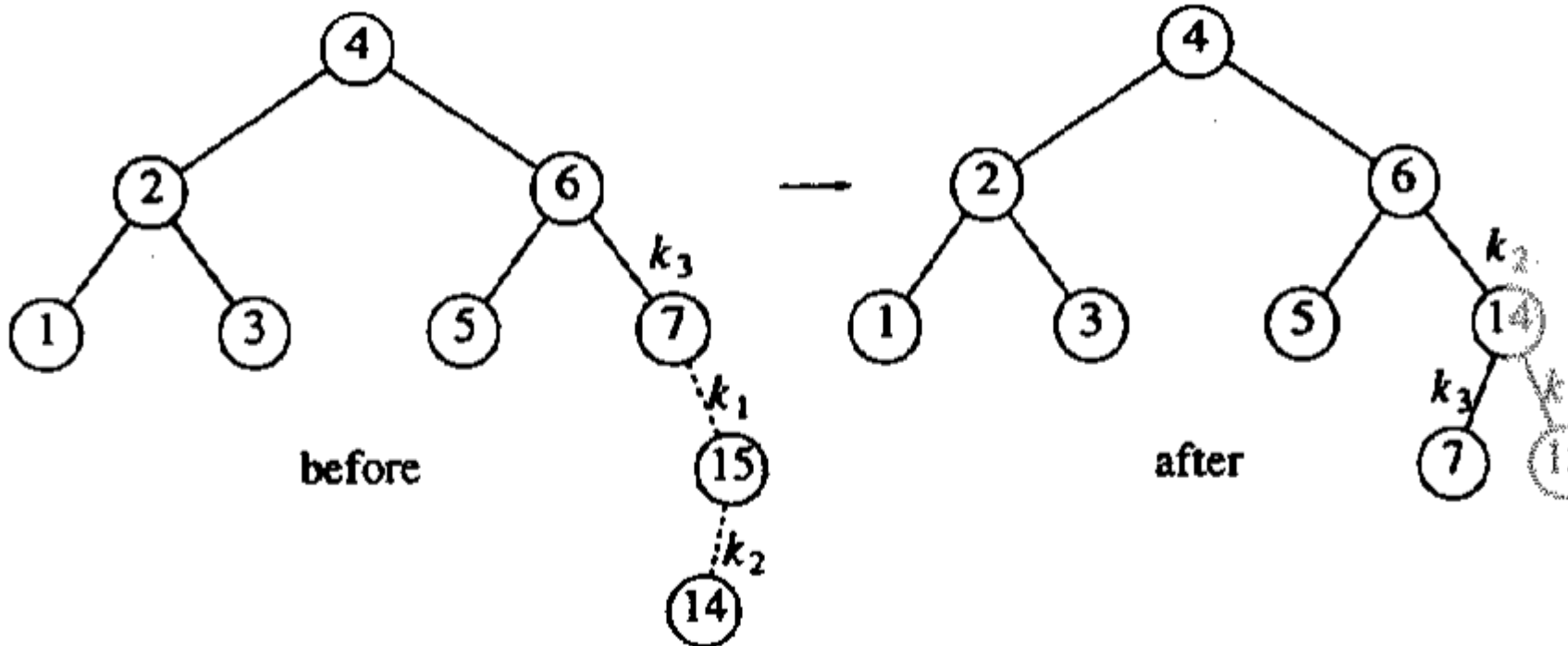
# Árvore AVL

- Inserção do nó 7. Nó 5 fica desbalanceado (+2)



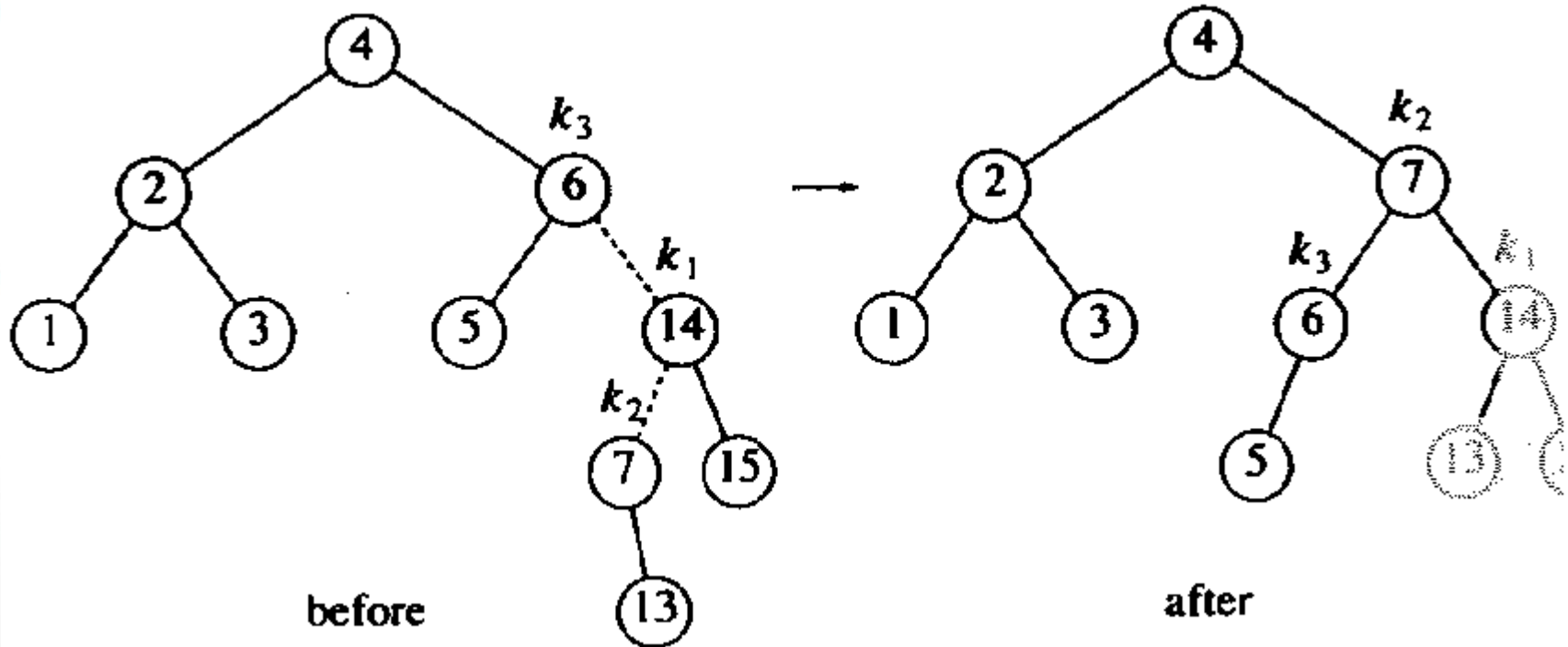
# Árvore AVL

- Inserção de 15 e 14. Rotação dupla: 14 e 15 à direita e depois 7 e 14 à esquerda.



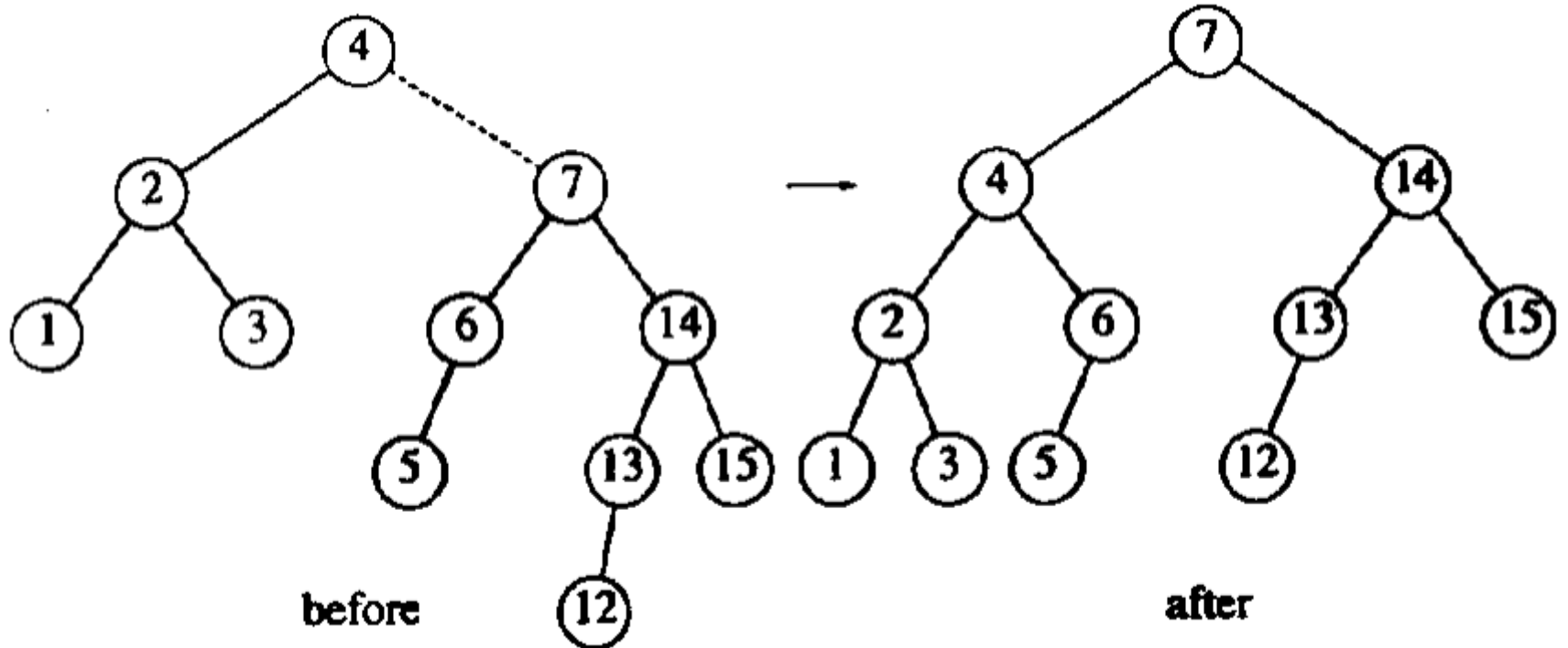
# Árvore AVL

- Inserção do 13. Rotação do 7 e 14 à direita. Rotação de 6 e 7 à esquerda



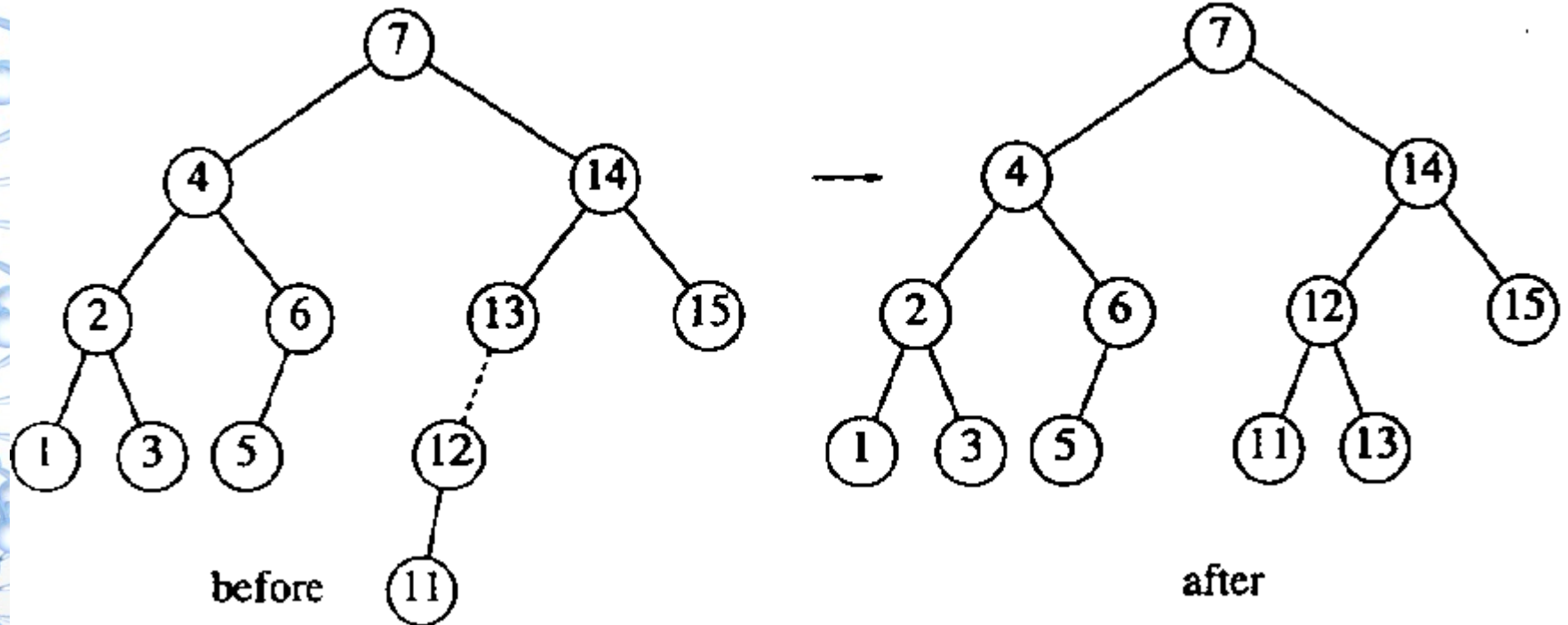
# Árvore AVL

- Inserção do 12. Rotação da raiz à esquerda



# Árvore AVL

- Inserção do 11. Rotação de 12 e 13 à direita





# Atividades

- Construir uma AVL com as chaves:
  - (10, 20, 30, 5, 3, 50, 40, 70, 60, 90)
- Construir uma AVL com as chaves:
  - (PSC, INF, ENG, QUI, MAT, LET, MED, ECO, ADM)
- Página 63 do livro base.





# Vídeos

- <https://www.youtube.com/watch?v=3zmjQlJhBLM>
- <https://www.youtube.com/watch?v=1HkWqH7L2rU>
- <https://www.youtube.com/watch?v=Au-6c55J90c>