

Estruturas de Dados Básicas II

Prof. Dr. Matheus da Silva Menezes



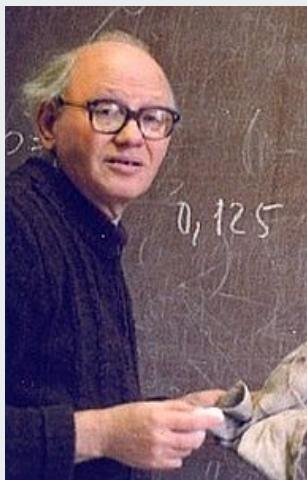
Unidade 02.

Estruturas de Dados Básicas II

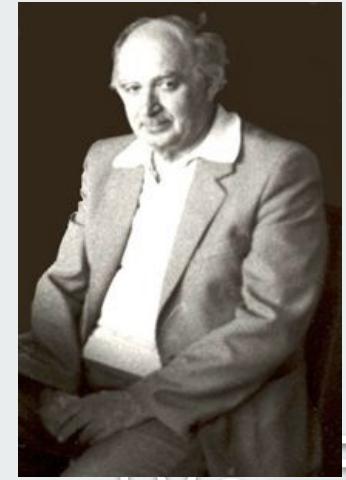
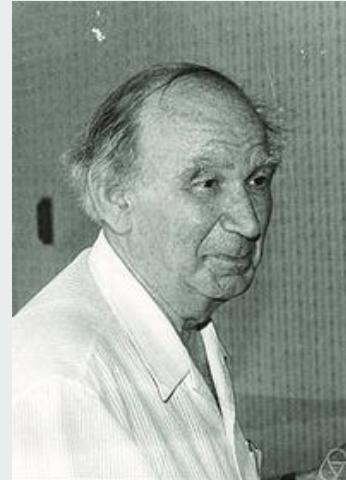
ÁRVORE AVL
Busca, Inclusão e Remoção

Origem

AVL = Georgy Maximovich **Adelson-Velsky** e Evgenii Mikhailovich **Landis**



.....

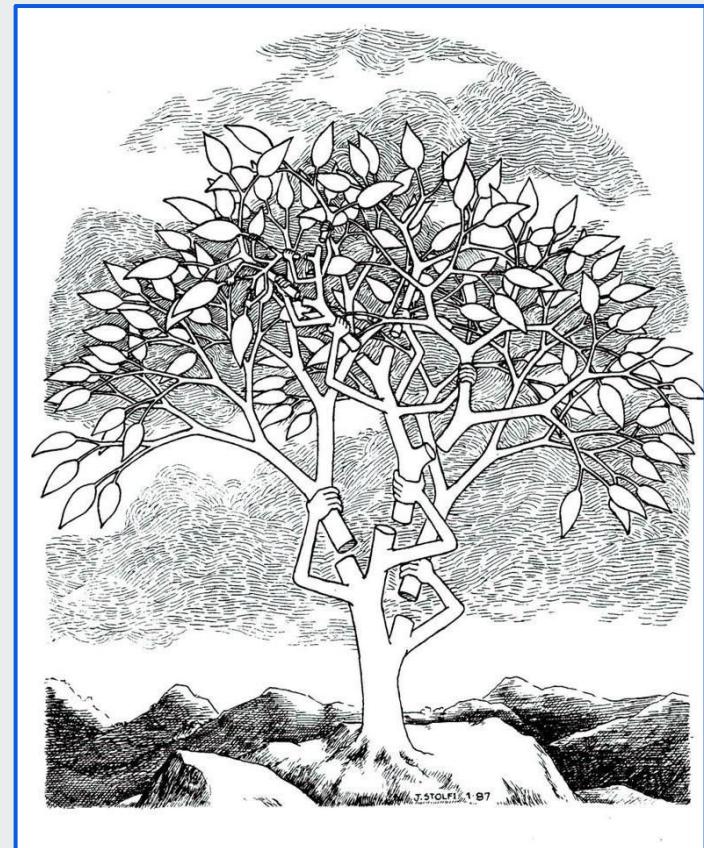


G.M. Adel'son-Vel'skii, E. M. Landis (1962). An algorithm for the organization of information, Soviet Mathematics Doklady 3, 1259-1263.



Árvore AVL

- Árvore Binária de Busca
- Árvore Balanceada
- Árvore Autoajustável



Árvore AVL

• Árvore Binária de Busca

Definição

Uma árvore binária de busca é uma árvore binária tal que:

- A raiz possui uma chave
- As chaves dos nós da subárvore **esquerda** da raiz são **menores** que a chave da raiz.
- As chaves dos nós da subárvore **direita** da raiz são **maiores** que a chave da raiz.
- As subárvores esquerda e direita são árvores binárias de busca

Árvore AVL

• Árvore Balanceada

Definição

Uma árvore binária de busca é dita balanceada se ela possui altura:

$$h = O(\log n)$$

- onde n é o número de nós da árvore

Árvore AVL

- Árvore Autoajustável

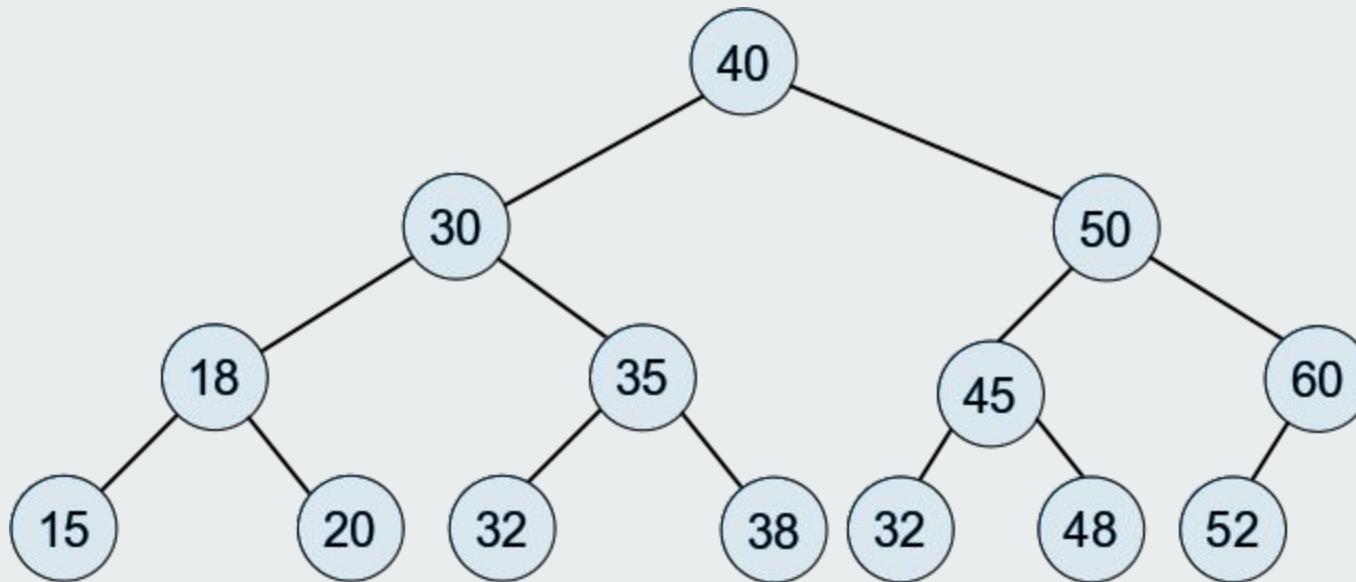
Idéia

Manter o custo de acesso na mesma ordem de grandeza de uma árvore binária de busca ótima: $O(\log n)$

Característica: A estrutura é alterada periodicamente.

Árvore de Busca Binária

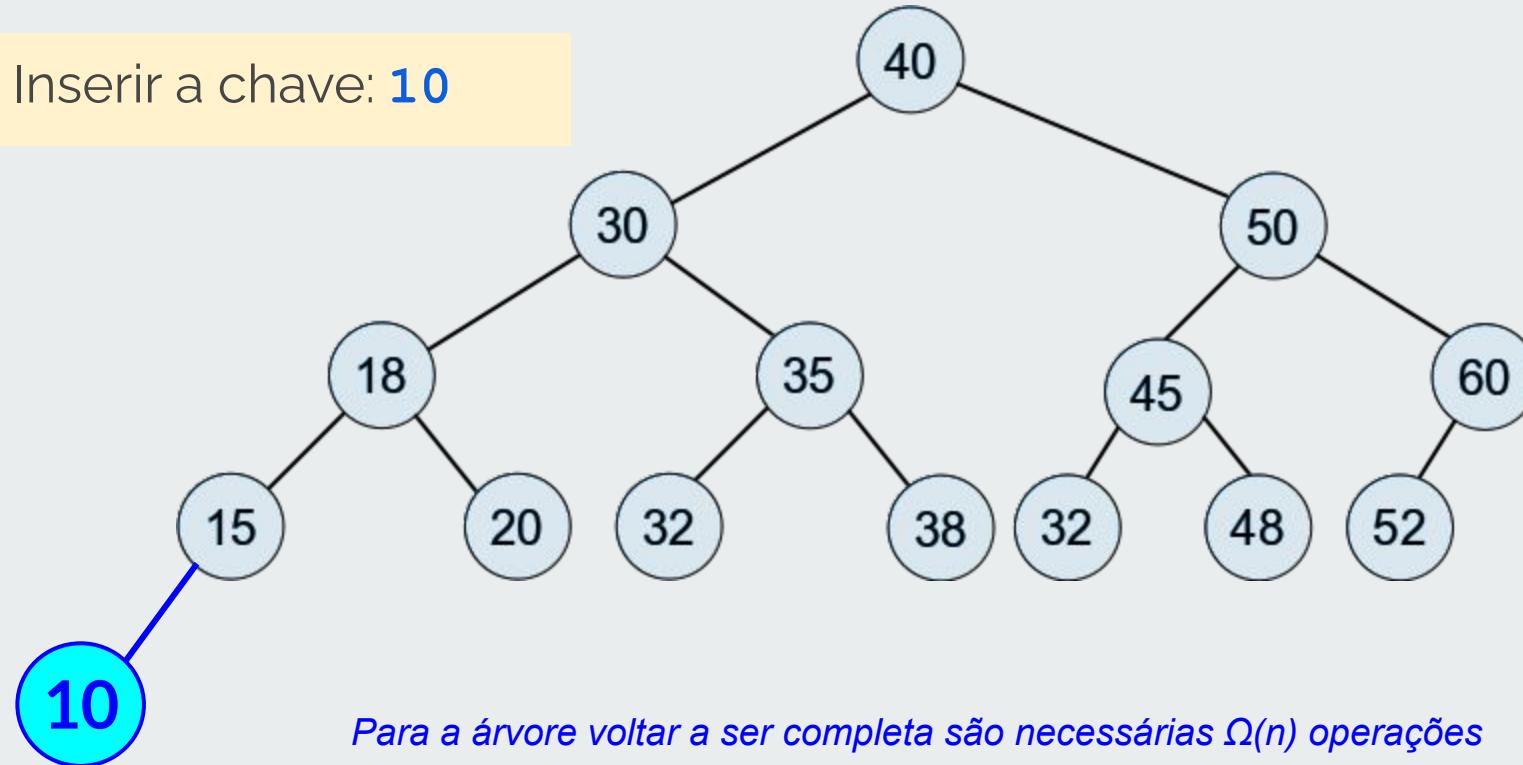
Esta árvore é completa, portanto tem altura mínima.



Árvore de Busca Binária

Esta árvore é completa, portanto tem altura mínima.

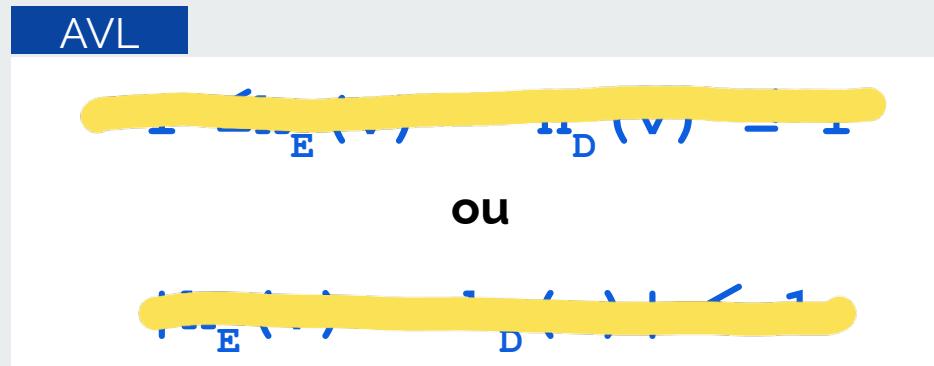
Inserir a chave: **10**



Para a árvore voltar a ser completa são necessárias $\Omega(n)$ operações

Definição

Uma árvore AVL é uma árvore binária de busca onde para cada nó v , tem-se:



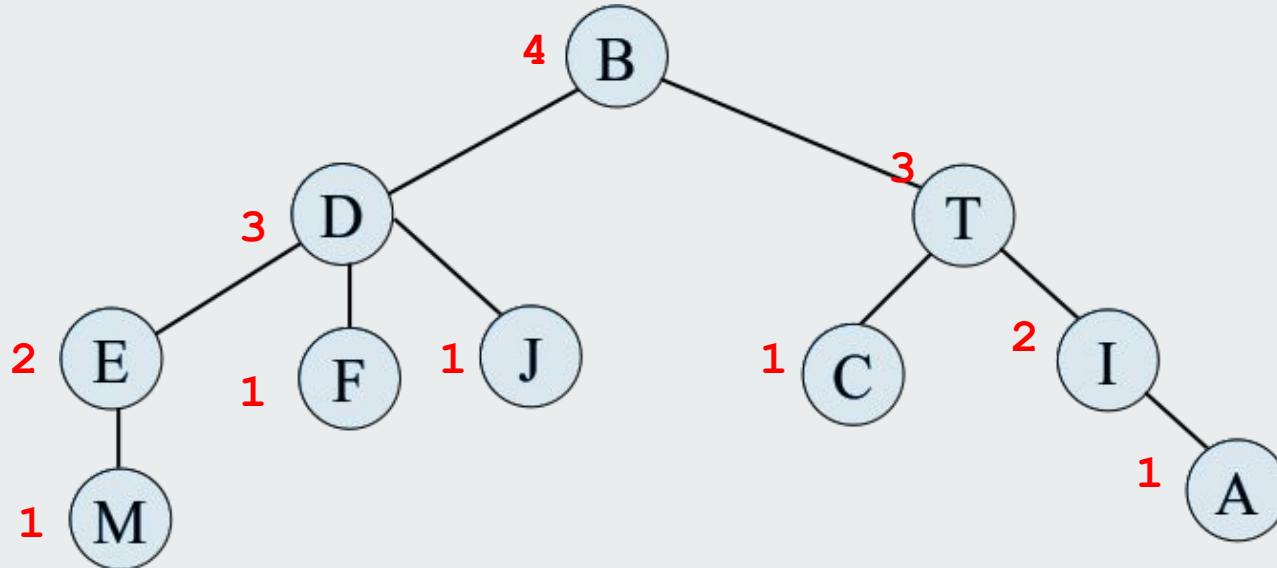
Onde:

$h_E(v)$ - altura da subárvore *esquerda* de v

$h_D(v)$ - altura da subárvore *direita* de v



Altura do nó v é o número de nós no caminho entre v e seu **descendente mais distante**.

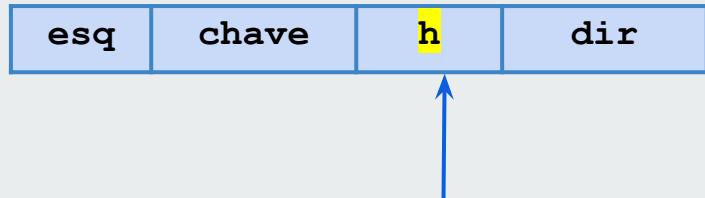


A **altura da árvore**, $h(T)$, é dada pela **altura da raiz** (ou pelo nível máximo de seus nós)

Uma representação para nó de uma árvore binária / AVL

Exemplo C

```
struct nodo{  
    int chave;  
    int h; ←  
    struct nodo *esq;  
    struct nodo *dir; };  
typedef struct nodo no;  
}
```



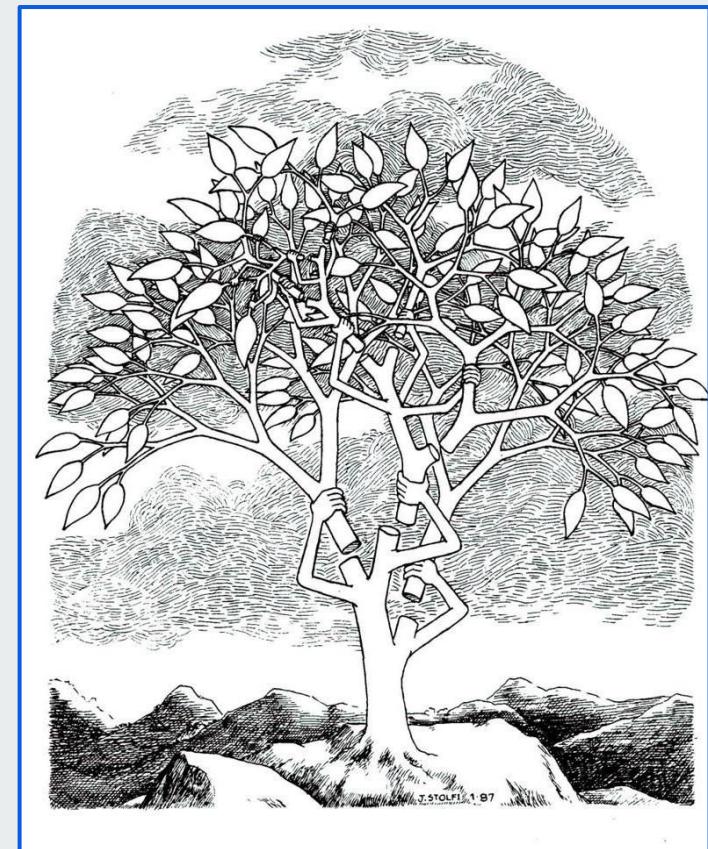
campo para altura do nó

```
typedef no *pont_no;
```

Observação: O campo chave será usado como chave e conteúdo/prioridade

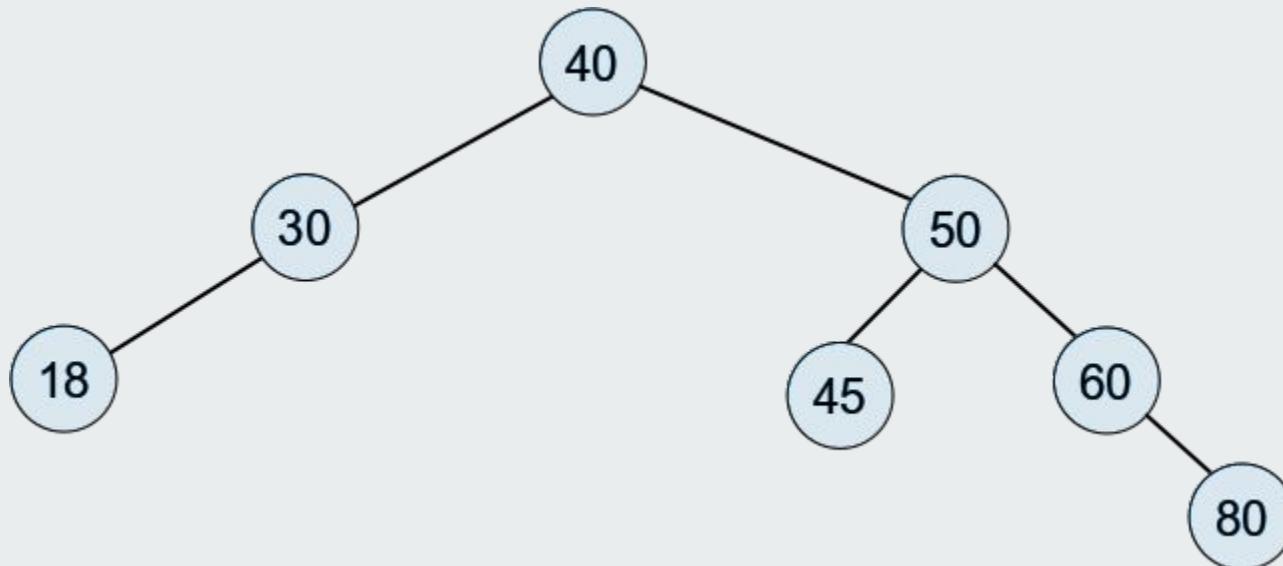
Árvores Balanceadas

- Árvores AVL
- Árvores Graduadas e Rubro Negras
- Árvores B



Árvore AVL

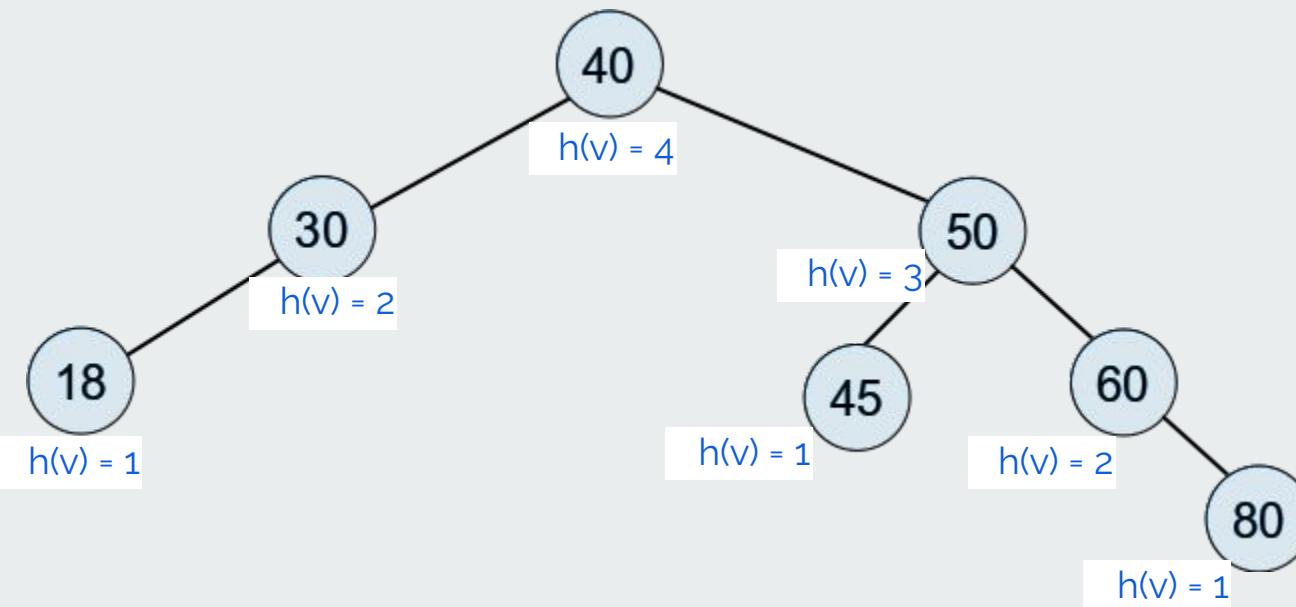
Essa árvore é AVL?



Árvore AVL

Essa árvore é AVL?

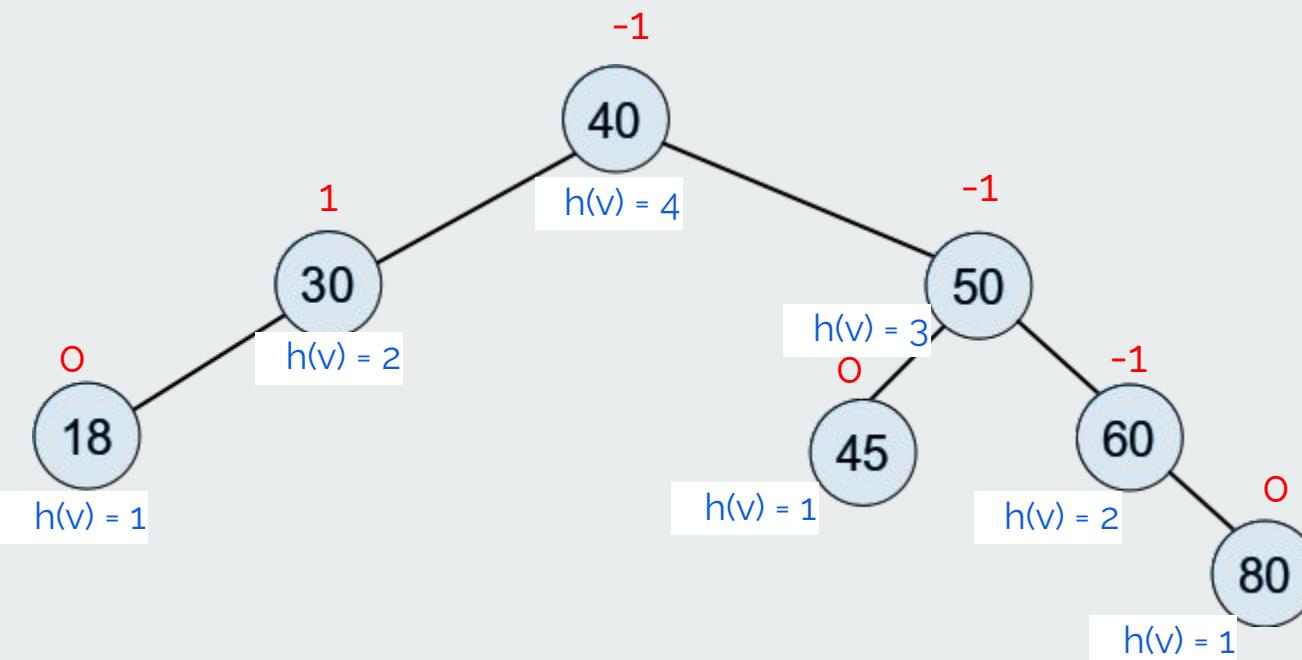
$$-1 \leq h_{E(v)} - h_{D(v)} \leq 1$$



Árvore AVL

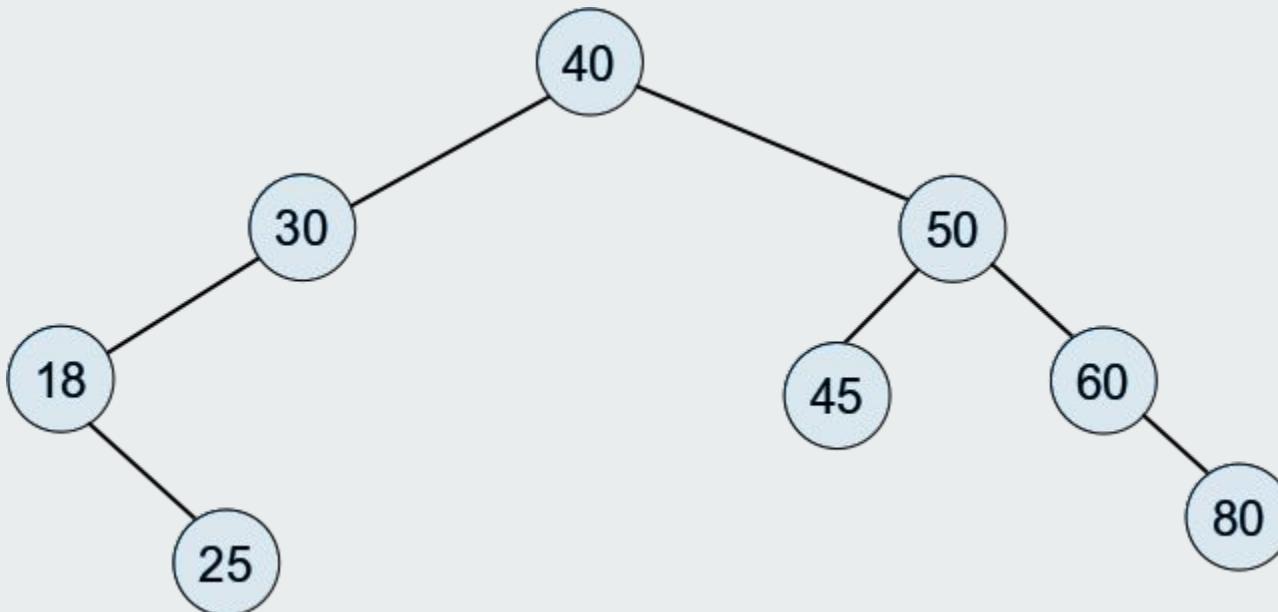
Essa árvore é AVL?

$$-1 \leq h_{E(v)} - h_{D(v)} \leq 1$$



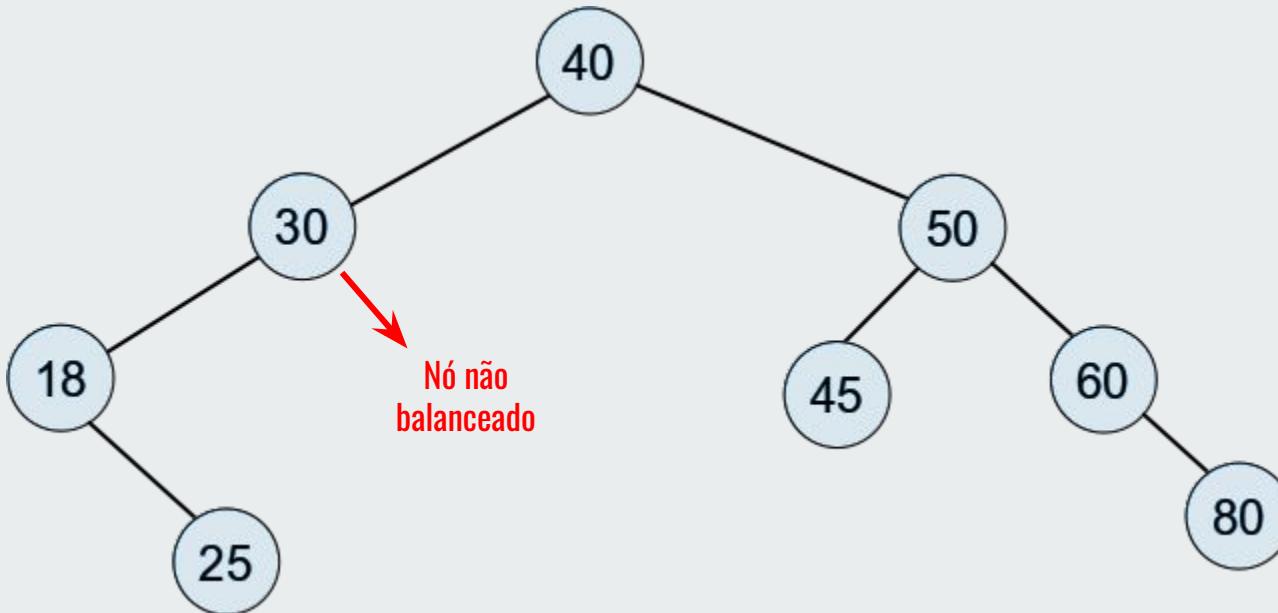
Árvore AVL

Essa árvore é AVL?



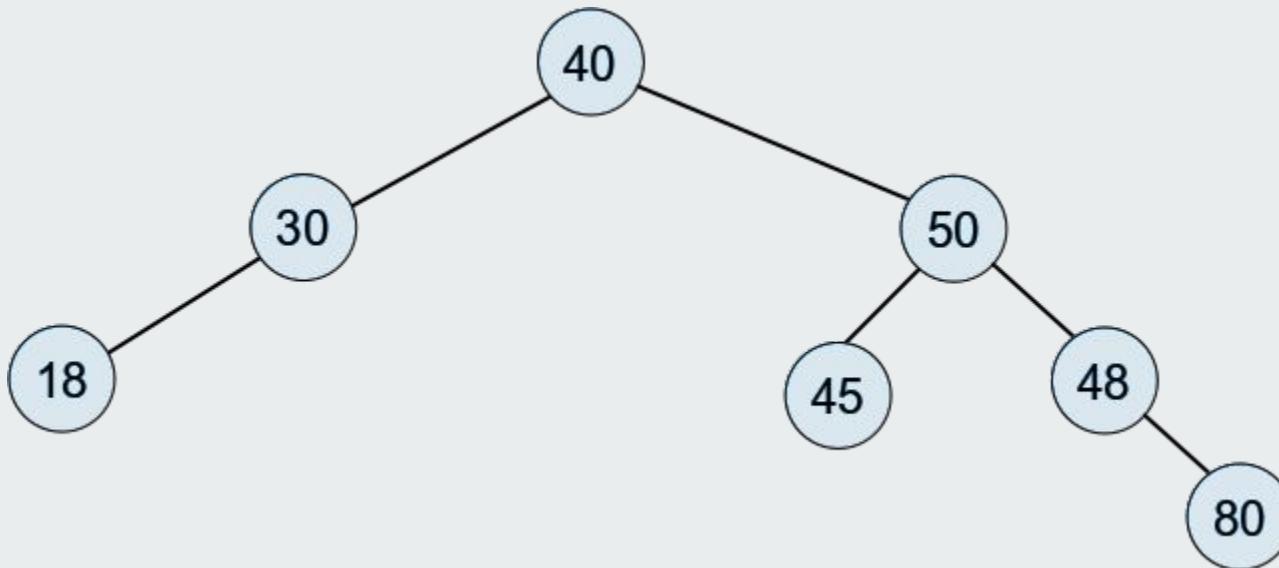
Árvore AVL

Essa árvore é AVL?



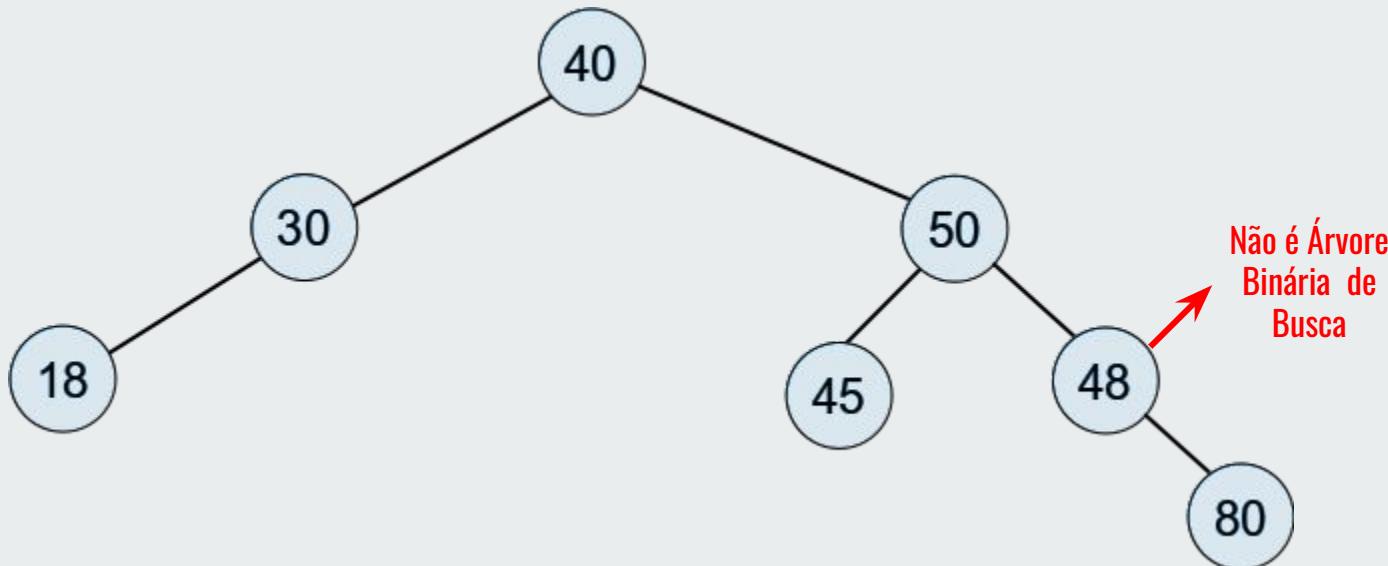
Árvore AVL

Essa árvore é AVL?



Árvore AVL

Essa árvore é AVL?



Operações Básicas

0 ROTAÇÕES

1 BUSCA

2 INSERÇÃO

3 REMOÇÃO



Operações Básicas

0 ROTAÇÕES

1 BUSCA

2 INSERÇÃO

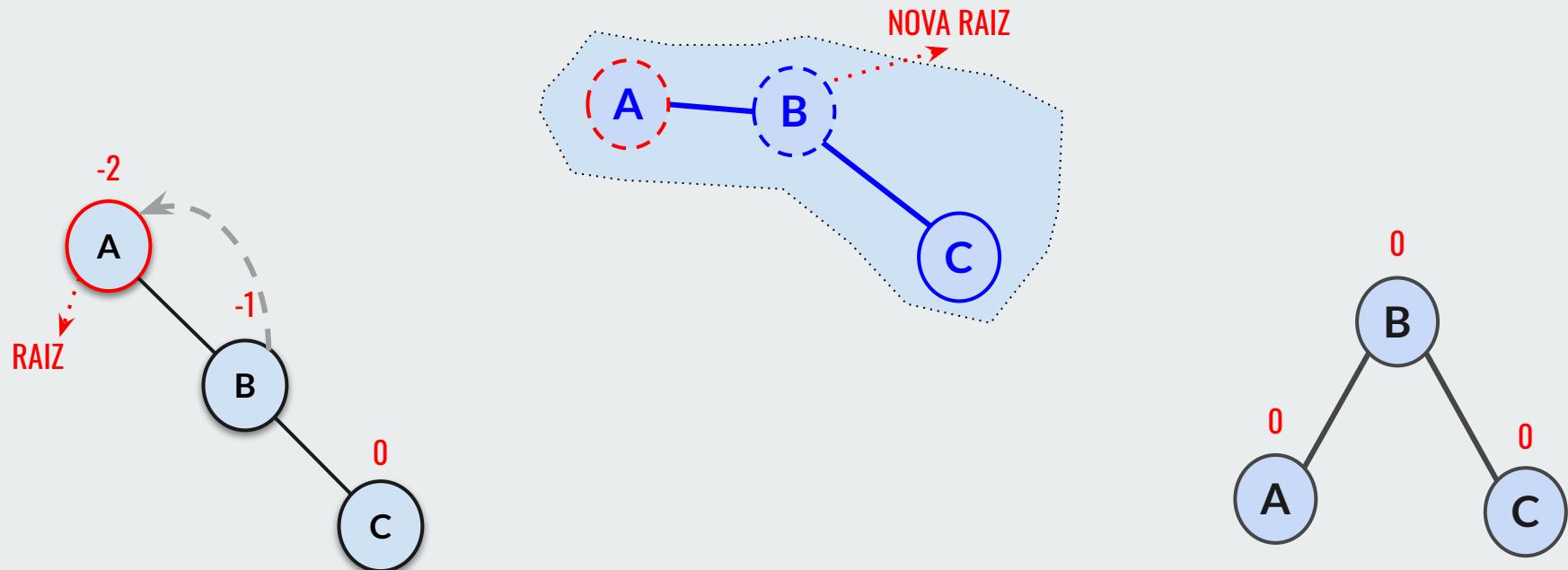
3 REMOÇÃO

- 0.1. Simples Esquerda
- 0.2. Simples Direita
- 0.3. Dupla Esquerda
- 0.4. Dupla Direita



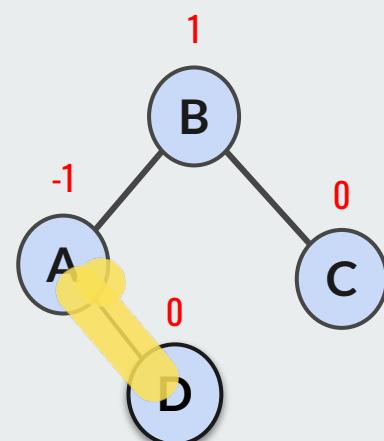
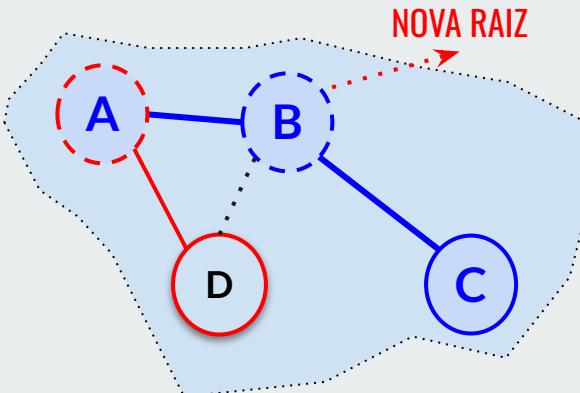
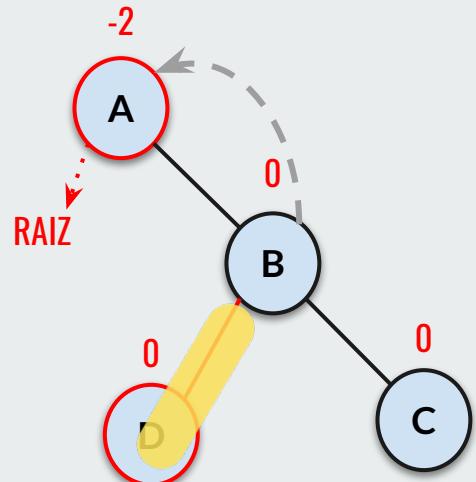
Rotação Esquerda

Caso 1: Filho direito da raiz não possui filho esquerdo ($A < B < C$)



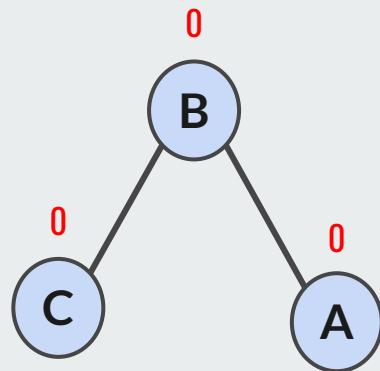
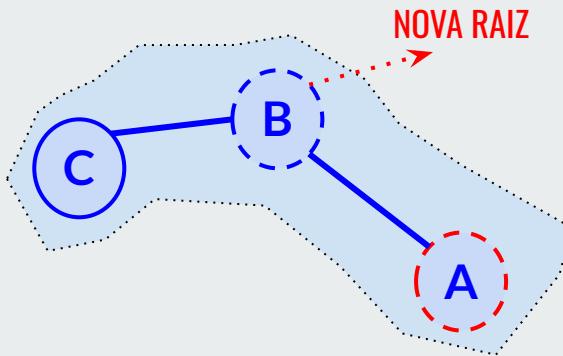
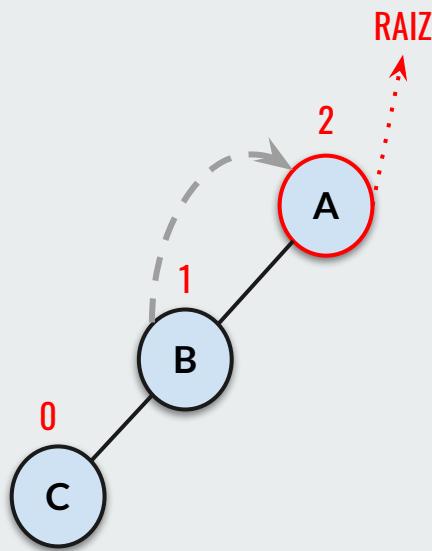
Rotação Esquerda

Caso 2: Filho direito da raiz possui filho esquerdo (A < D < B < C)



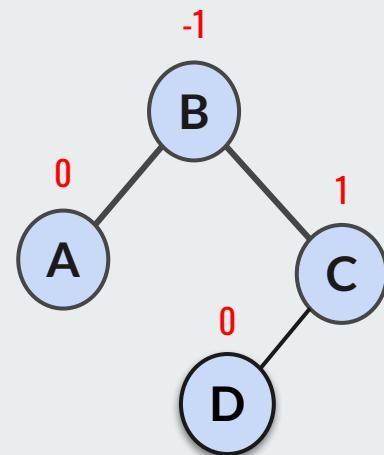
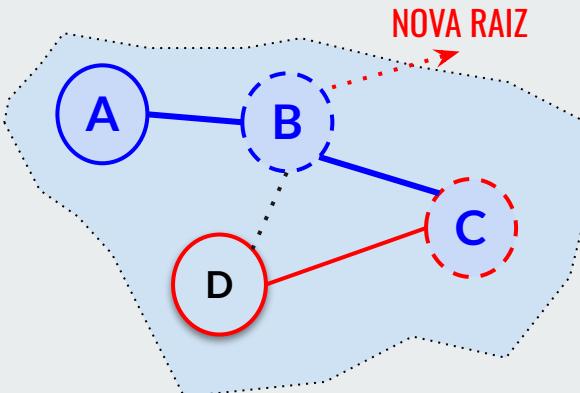
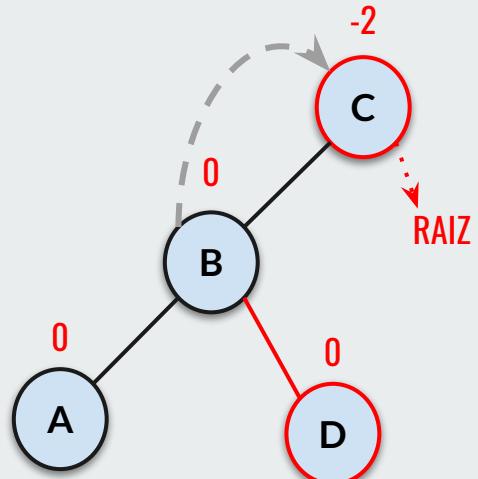
Rotação Direita

Caso 1: Filho esquerdo da raiz não possui filho direito ($C < B < A$)



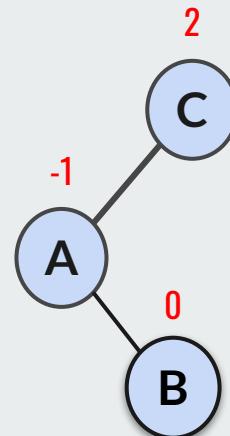
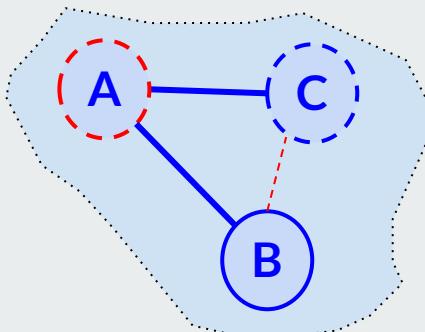
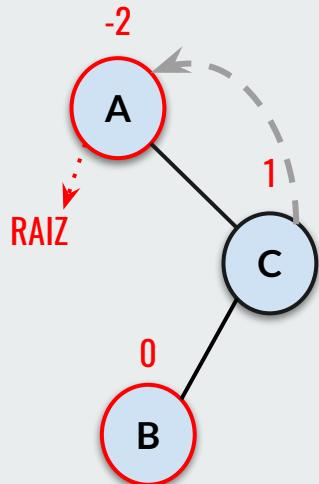
Rotação Direita

Caso 2: Filho esquerdo da raiz possui filho direito ($C < D < B < A$)



Rotação Dupla Esquerda

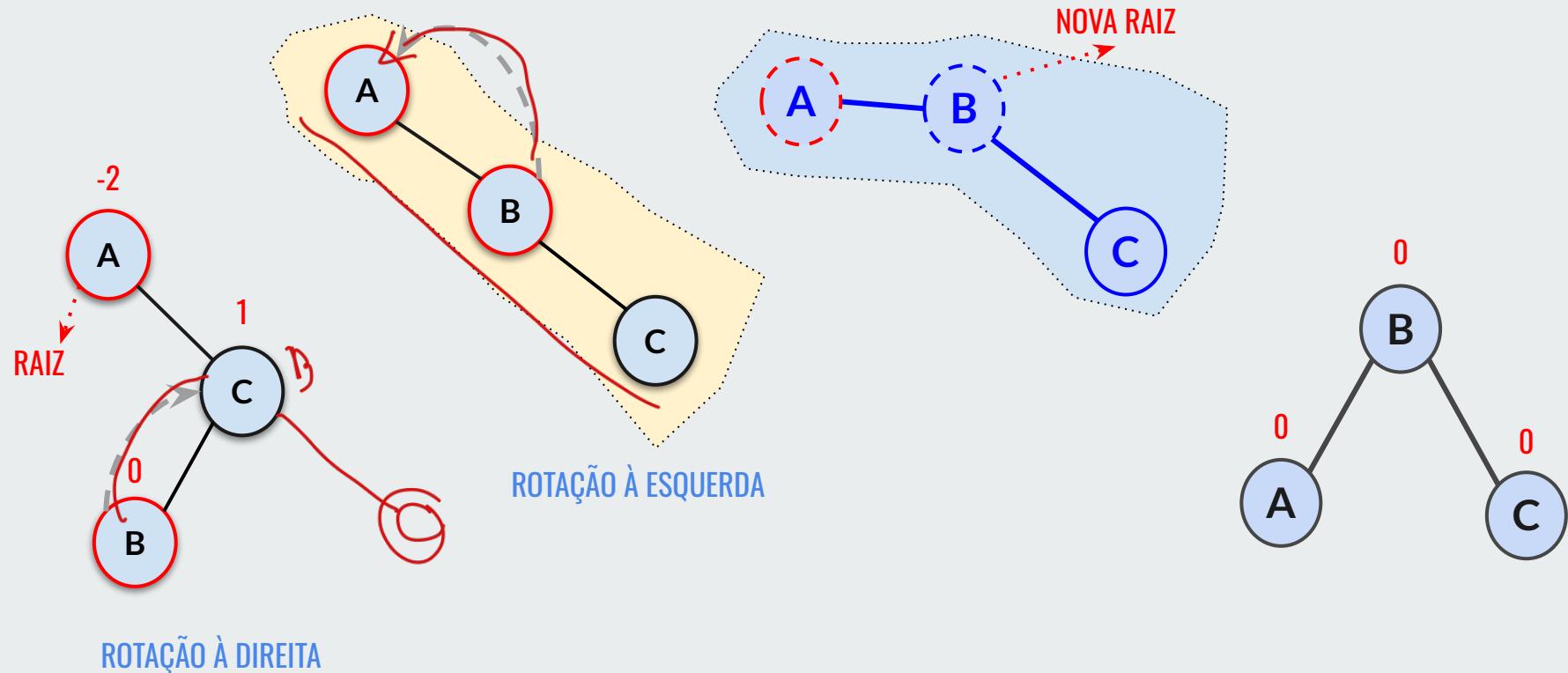
Considere o seguinte cenário:



E AGORA??

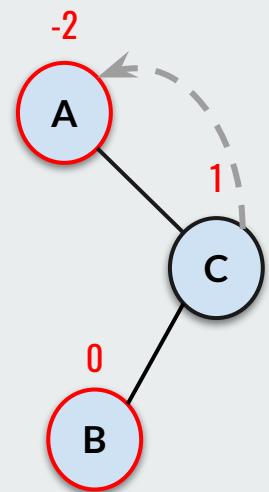
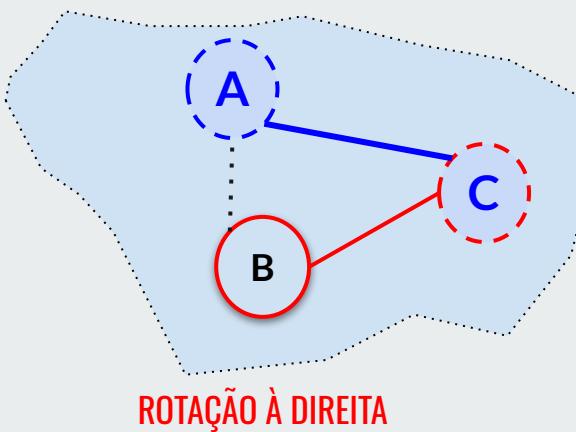
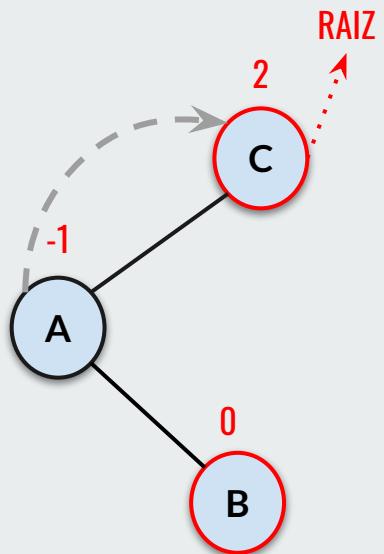
Rotação Dupla Esquerda

Considere o seguinte cenário:



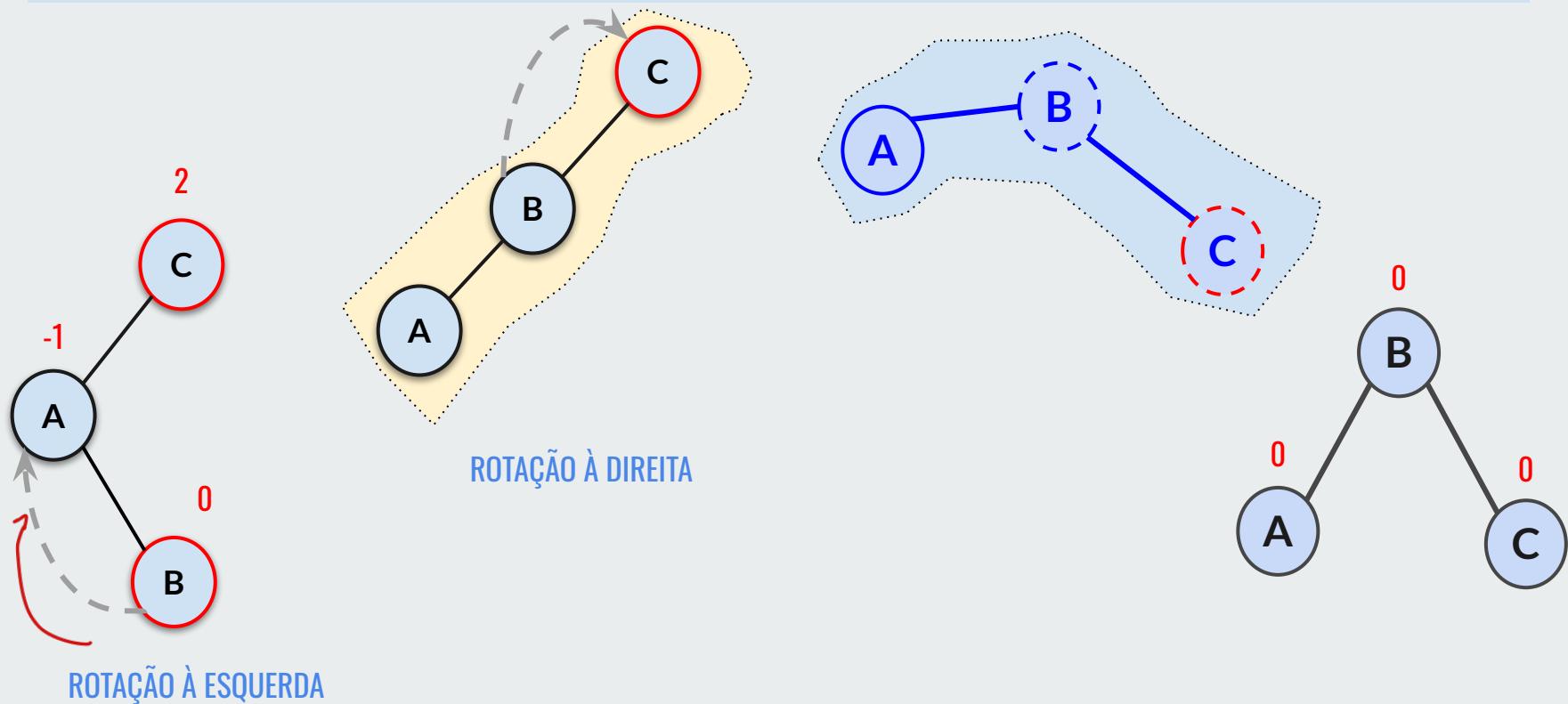
Rotação Dupla Direita

Considere o seguinte cenário:

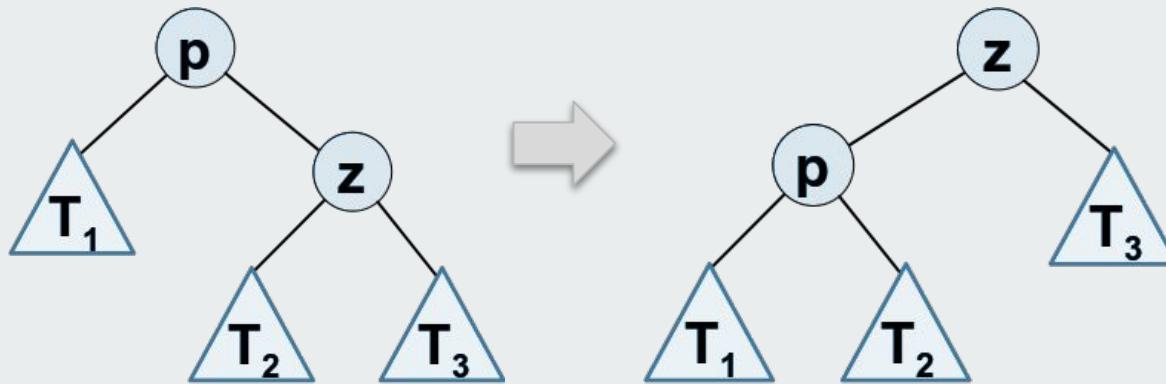


Rotação Dupla Esquerda

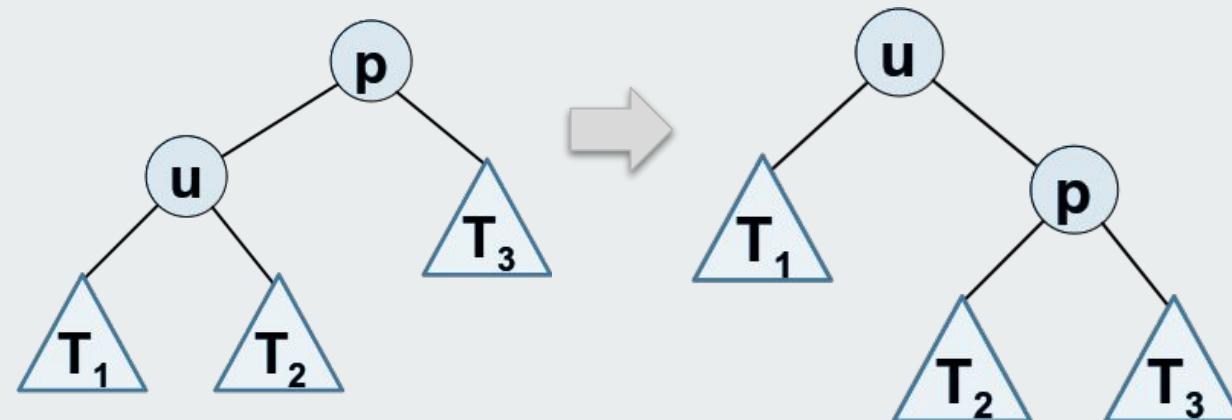
Considere o seguinte cenário:



EM RESUMO...

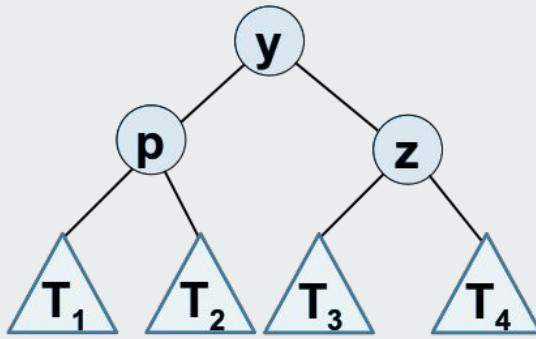
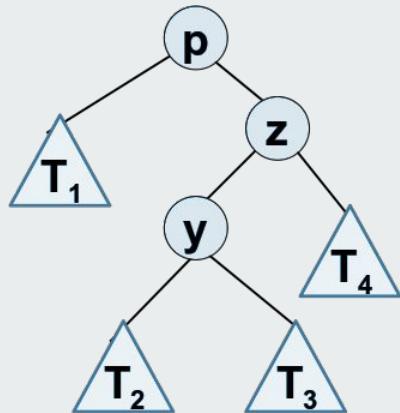


ROTAÇÃO ESQUERDA

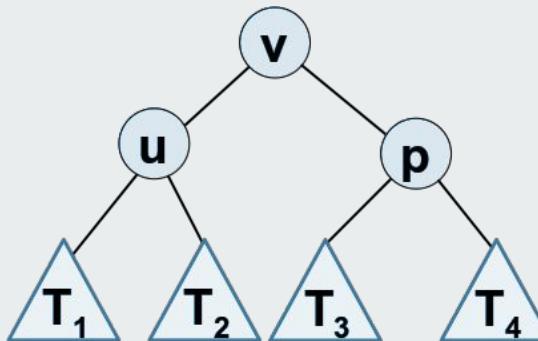
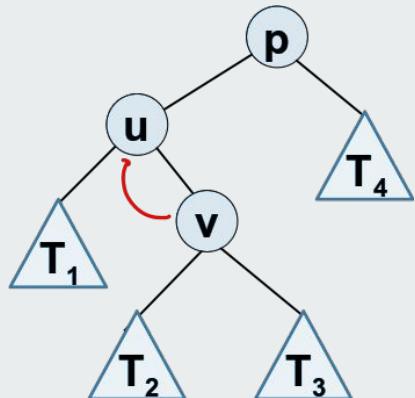
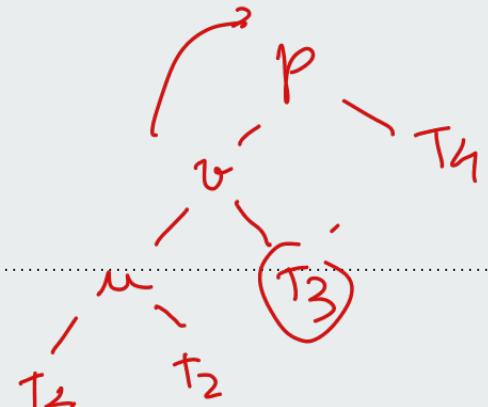


ROTAÇÃO DIREITA

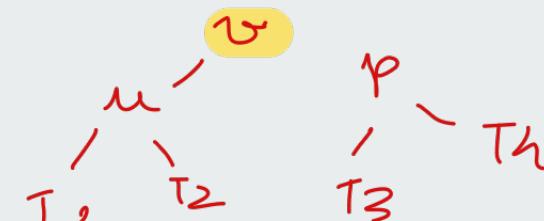
EM RESUMO...



ROTAÇÃO DUPLA ESQUERDA



ROTAÇÃO DUPLA DIREITA



Operações Básicas

0 ROTAÇÕES

1 BUSCA

2 INSERÇÃO

3 REMOÇÃO



Operações Básicas

0 ROTAÇÕES



*Idêntico ao visto na
árvore de busca
binária*

1 BUSCA



2 INSERÇÃO



3 REMOÇÃO



Operações Básicas

0 ROTAÇÕES

1 BUSCA

2 INSERÇÃO

3 REMOÇÃO



Operações Básicas

Pseudocódigo

```
void insere(pont_no raiz, int chave){  
    if (raiz== NULL)  
        pont_no novo = (pont_no)malloc(sizeof(no));  
        novo->chave = chave;  
        novo->esq = novo->dir = NULL;  
    if (chave < raiz->chave) // Busca na subárvore esquerda  
        return insere (raiz->esq, chave);  
    if (chave > raiz->chave) // Busca na subárvore direita  
        return insere (raiz->dir, chave);  
(...) CONTINUA
```



Inserção

Pseudocódigo

(...)

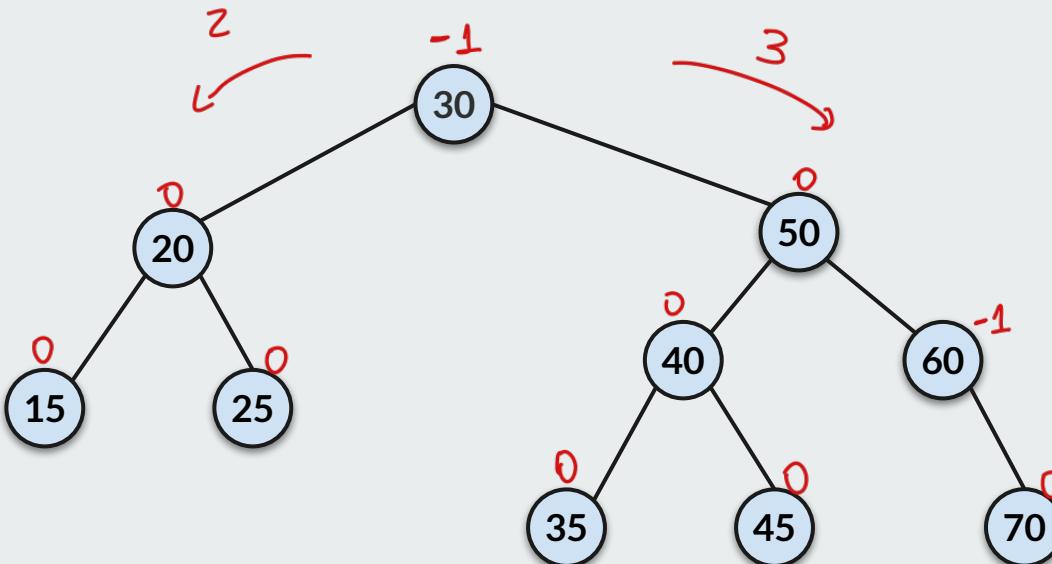
```
    raiz.h = 1+max(raiz->esq.h, raiz->dir.h)  
    bl= raiz.h
```

```
if (bl > 1 e chave < raiz->esq.chave)  
    return rotDireita(raiz); //caso 1  
if (bl > 1 e chave > raiz->esq.chave)  
    raiz->esq = rotEsquerda(raiz->esq);  
    return rotDireita(raiz); //caso 2  
  
if (bl < -1 e chave > raiz->dir.chave)  
    return rotEsquerda(raiz); //caso 3  
if (bl < -1 e chave < raiz->dir.chave)  
    raiz->dir = rotDireita(raiz->dir);  
    return rotEsquerda(raiz); //caso 4  
}
```



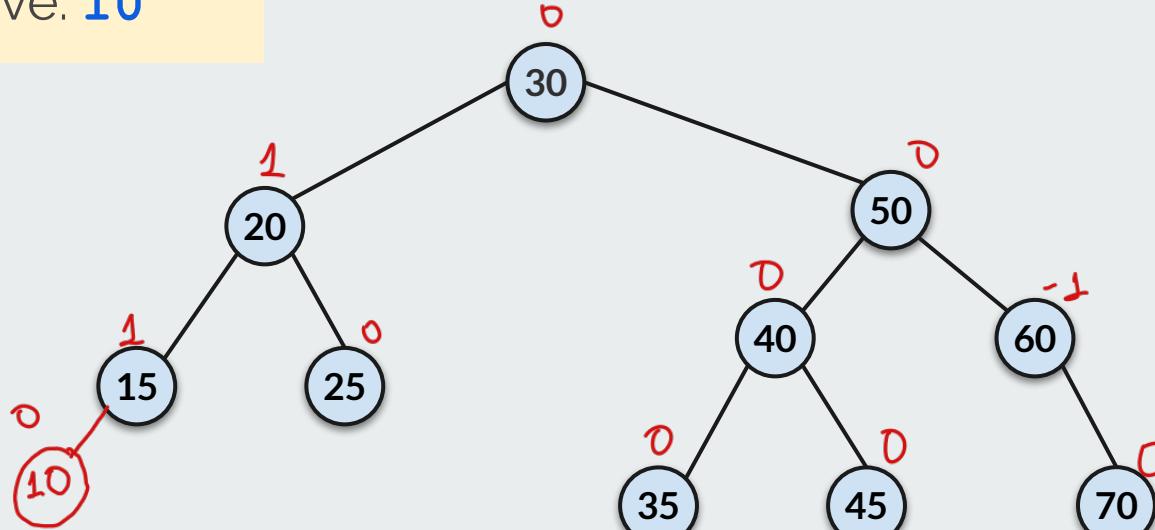
Árvore AVL

Essa árvore é AVL?



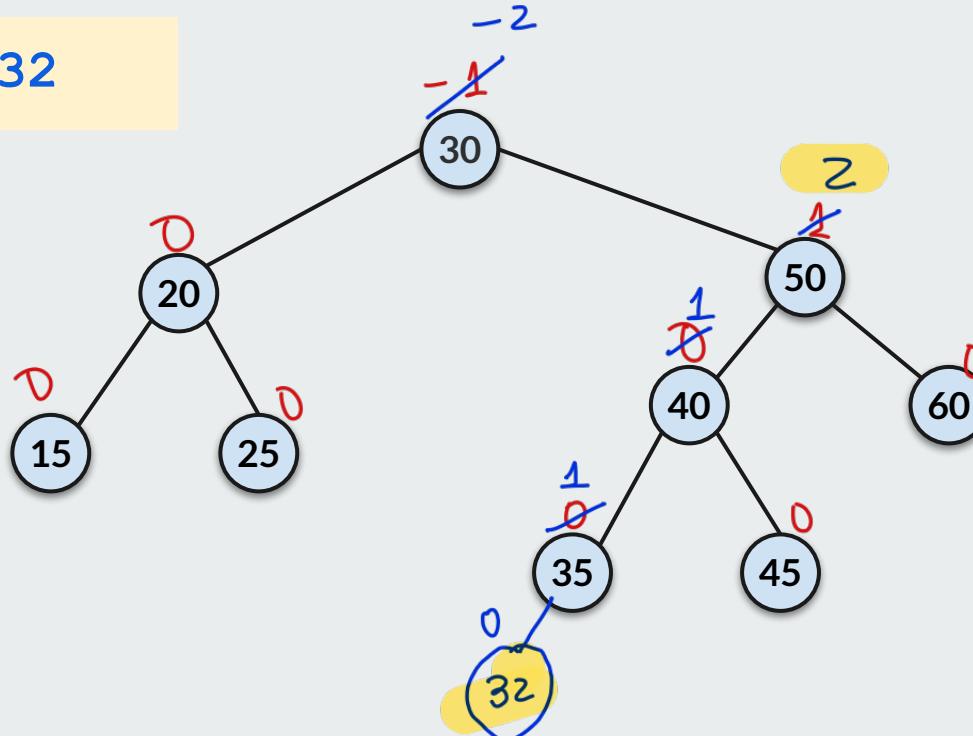
Inclusão em Árvore AVL

Inserir a chave: **10**



Inclusão em Árvore AVL

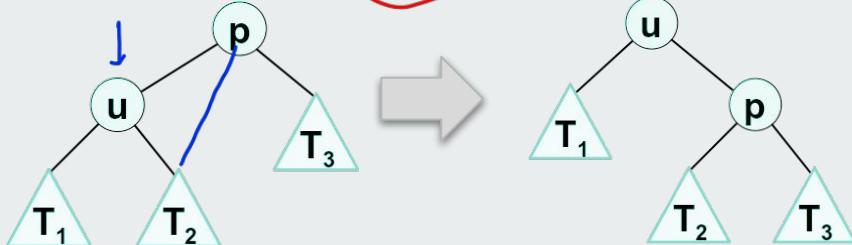
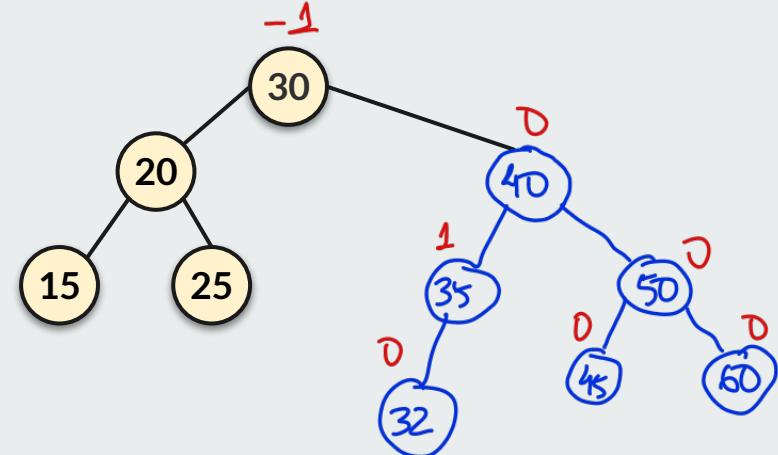
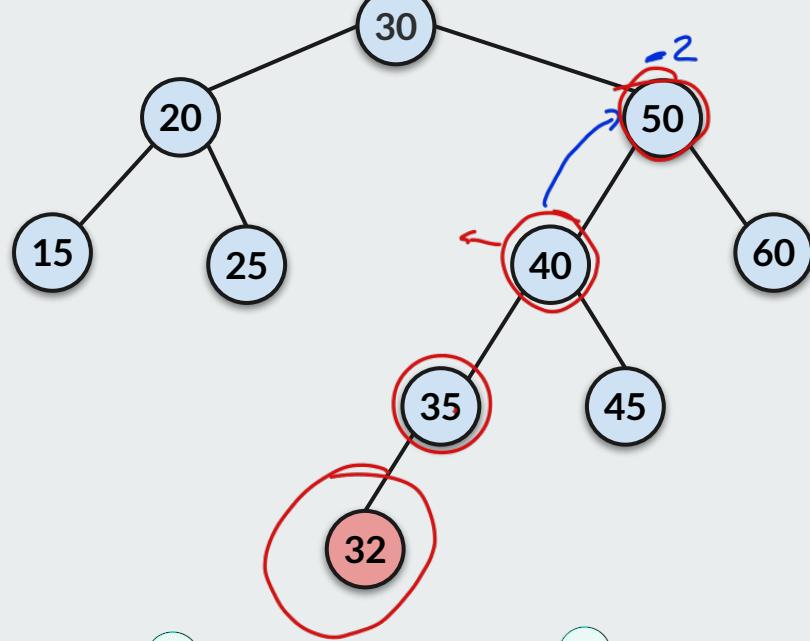
Inserir a chave: **32**



Inclusão em Árvore AVL

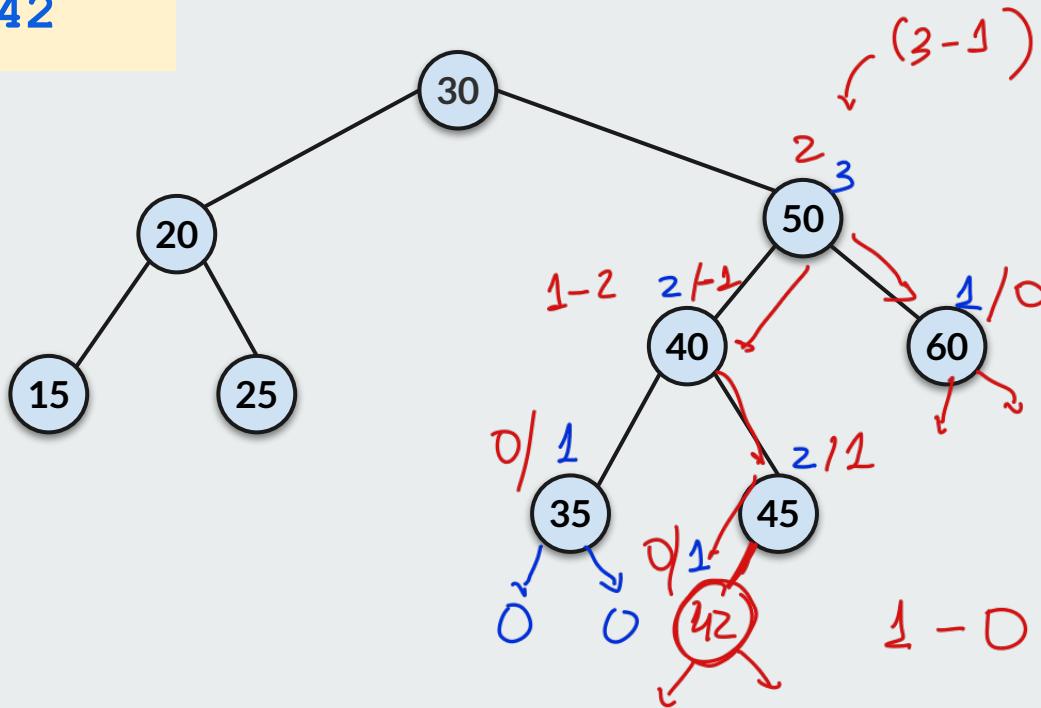
ESQ

```
if (bl > 1 e chave < raiz->esq.chave)
    return rotDireita(raiz); //caso 1
```

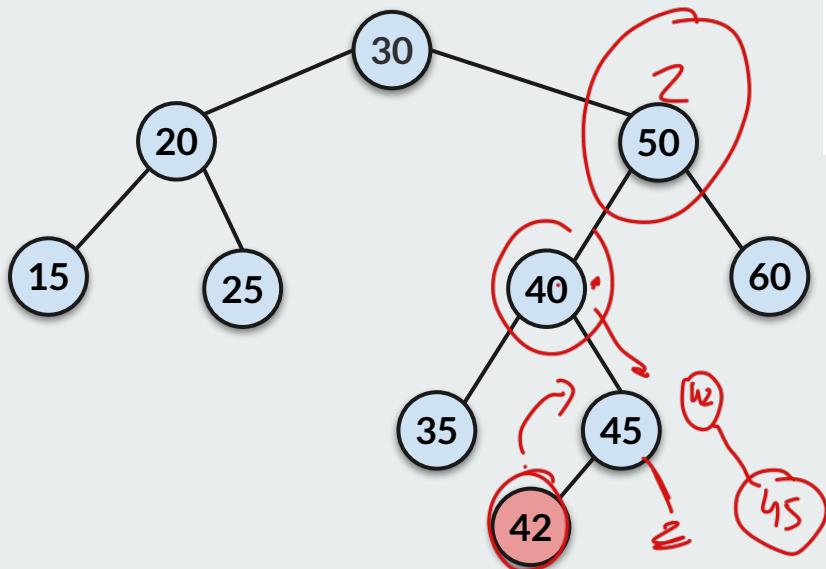


Inclusão em Árvore AVL

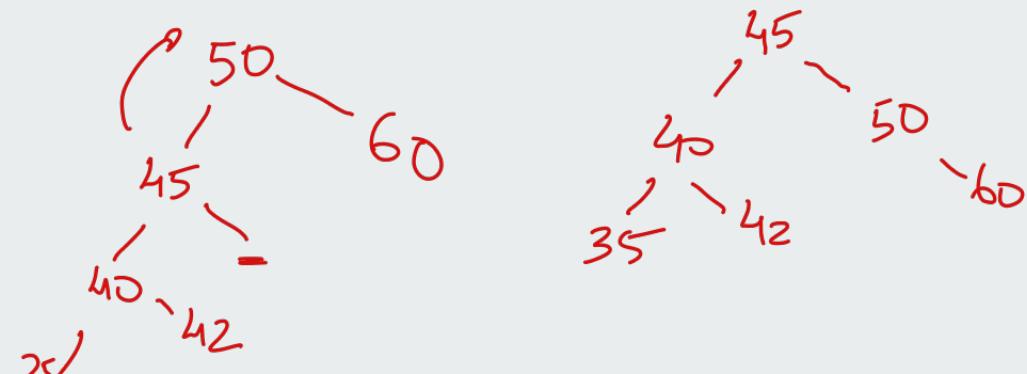
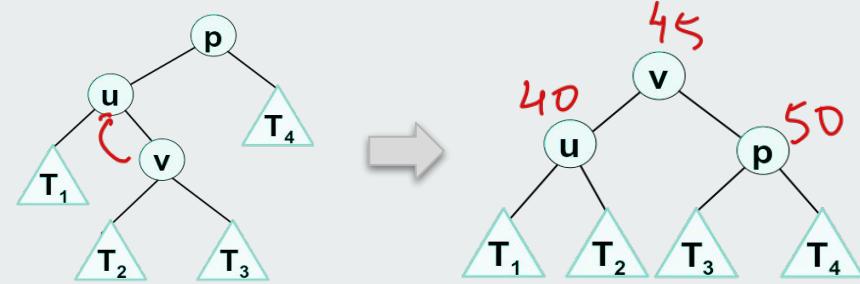
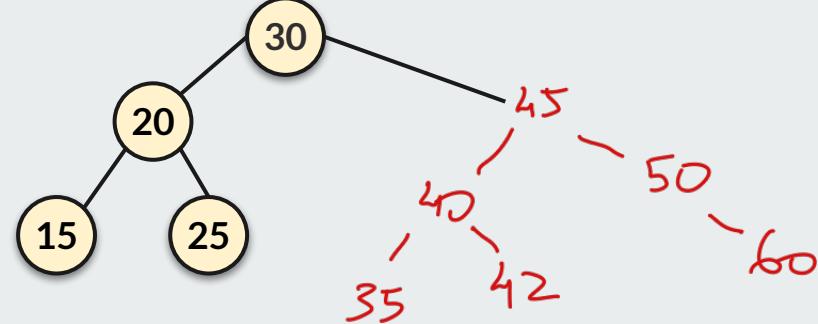
Inserir a chave: **42**



Inserção em Árvore AVL

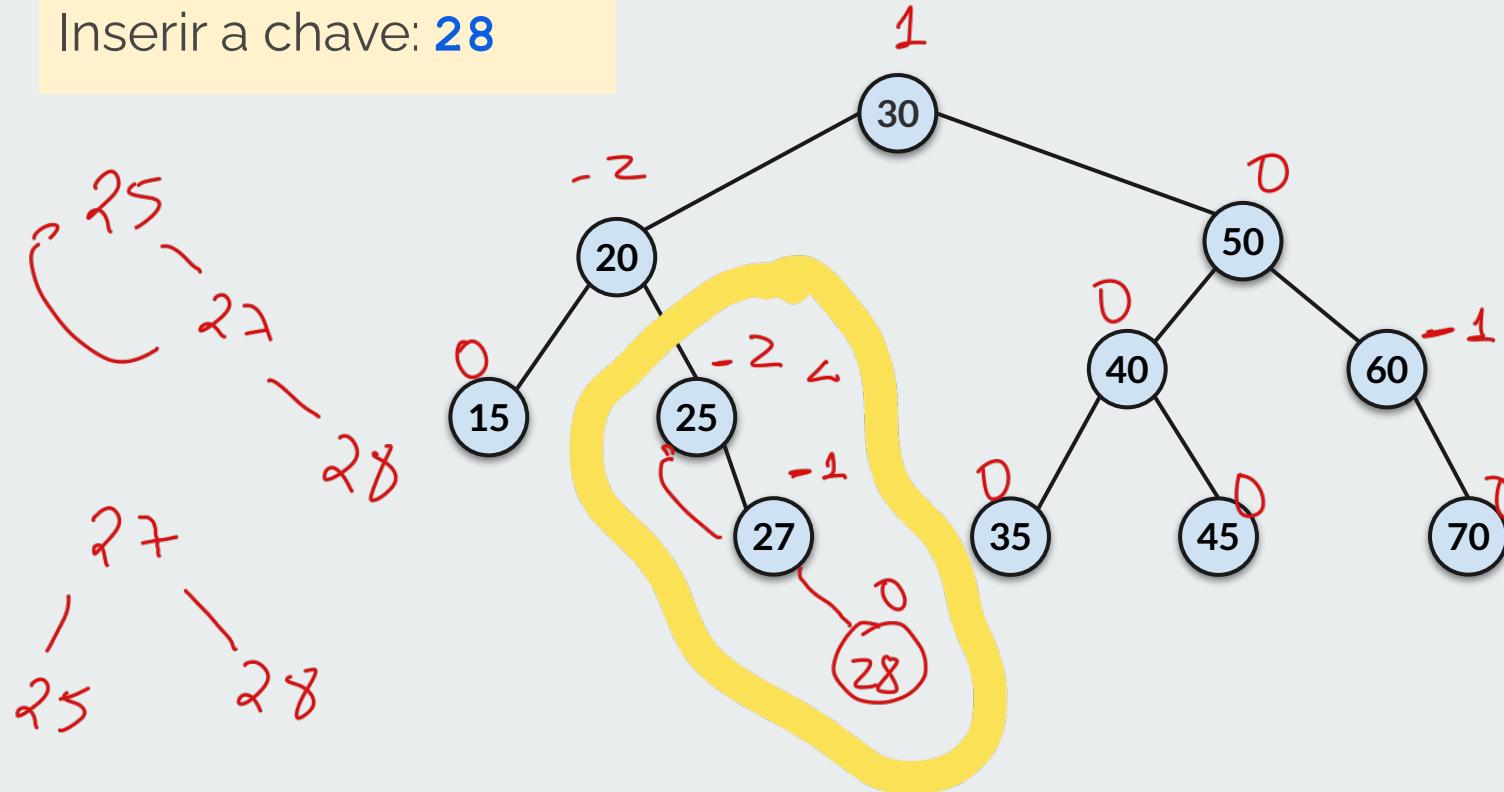


```
if (bl > 1 e chave > raiz->esq.chave)
    raiz->esq = rotEsquerda(raiz->esq);
    return rotDireita(raiz); //caso 2
```

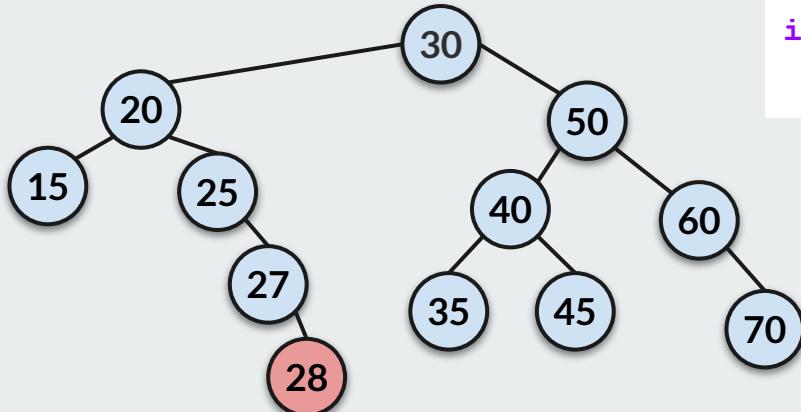


Inclusão em Árvore AVL

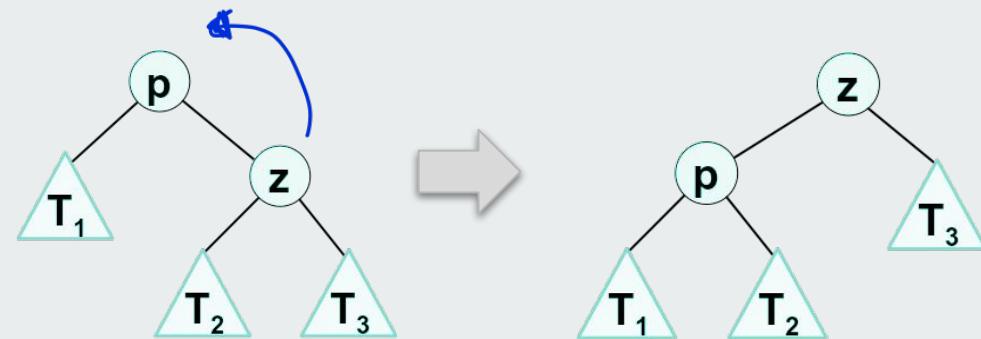
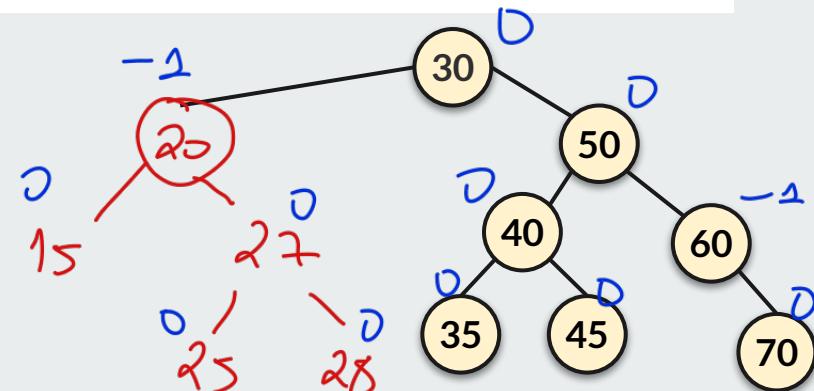
Inserir a chave: **28**



Inclusão em Árvore AVL

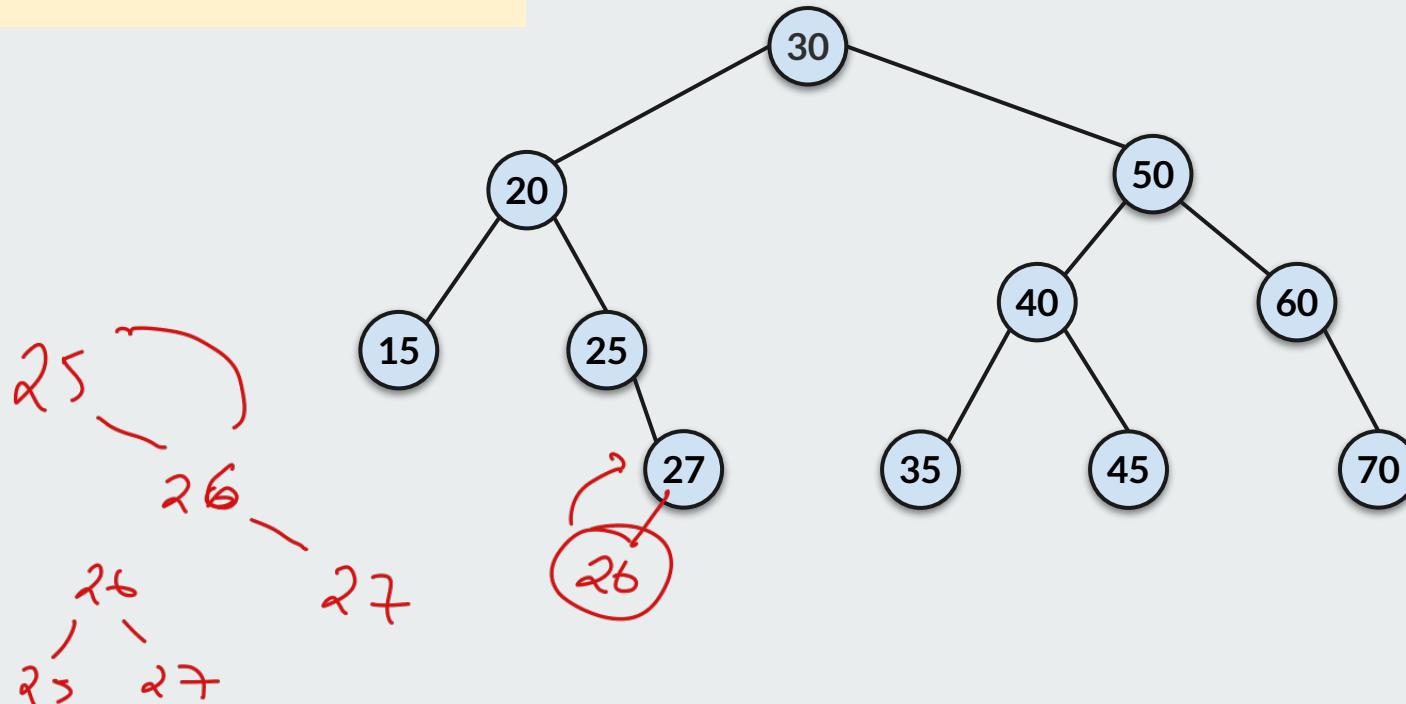


```
if (bl < -1 e chave > raiz->dir.chave)
    return rotEsquerda(raiz); //caso 3
```

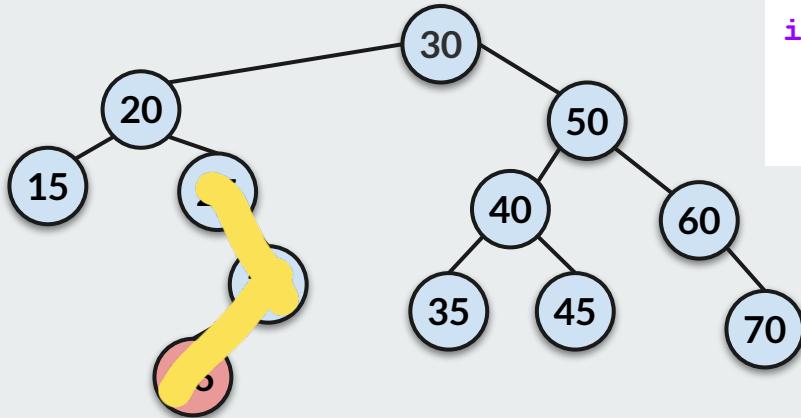


Inclusão em Árvore AVL

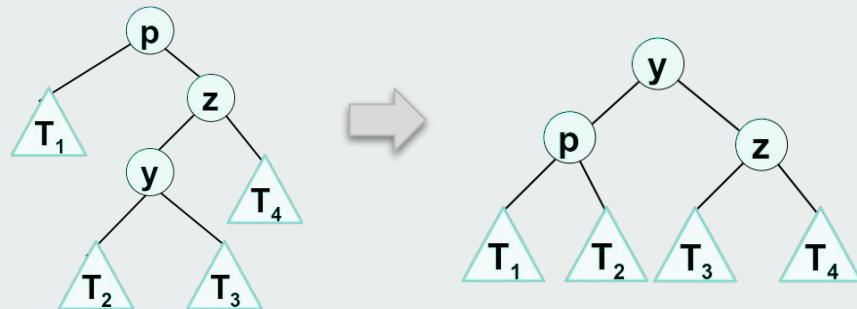
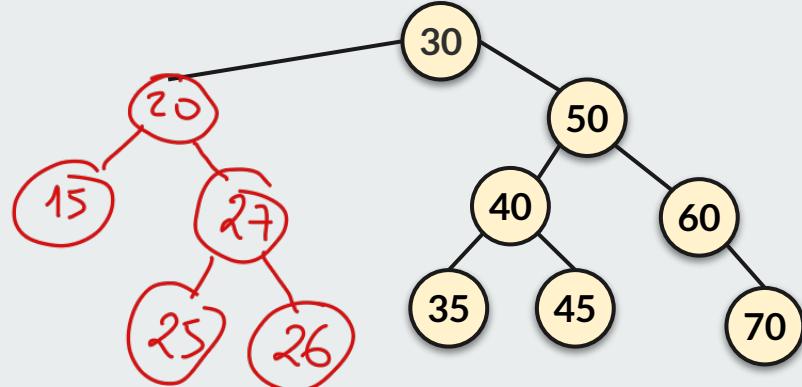
Inserir a chave: **26**



Inclusão em Árvore AVL



```
if (bl < -1 e chave < raiz->dir = rotDireita;
    return rotEsquerda(); //caso 4
```



Operações Básicas

0 ROTAÇÕES

1 BUSCA

2 INSERÇÃO

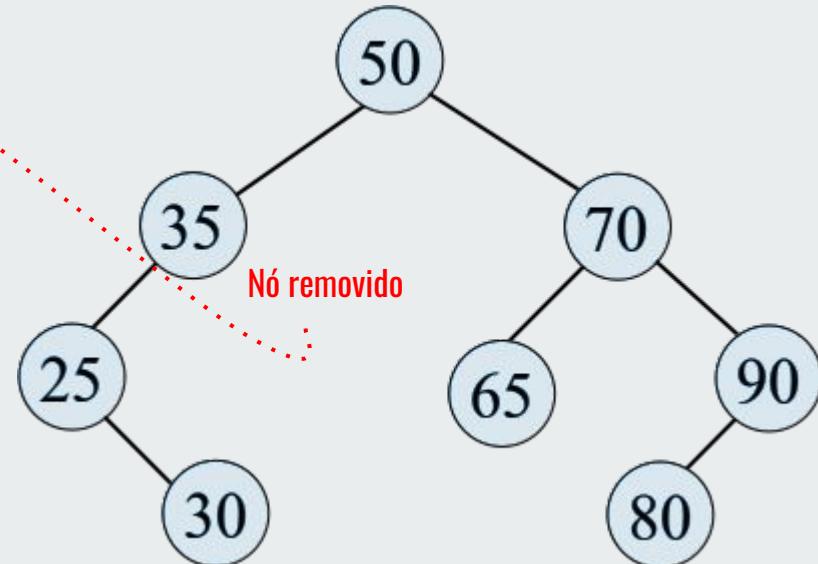
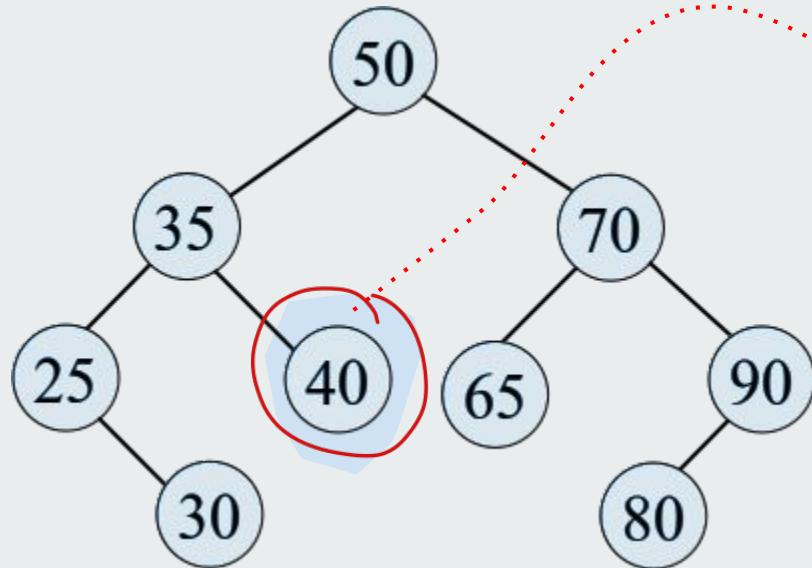
3 REMOÇÃO



CASO 1. Deseja-se remover uma chave que está em uma folha.

Remover a chave: **40**

(relembrando) remoção em árvore binária...

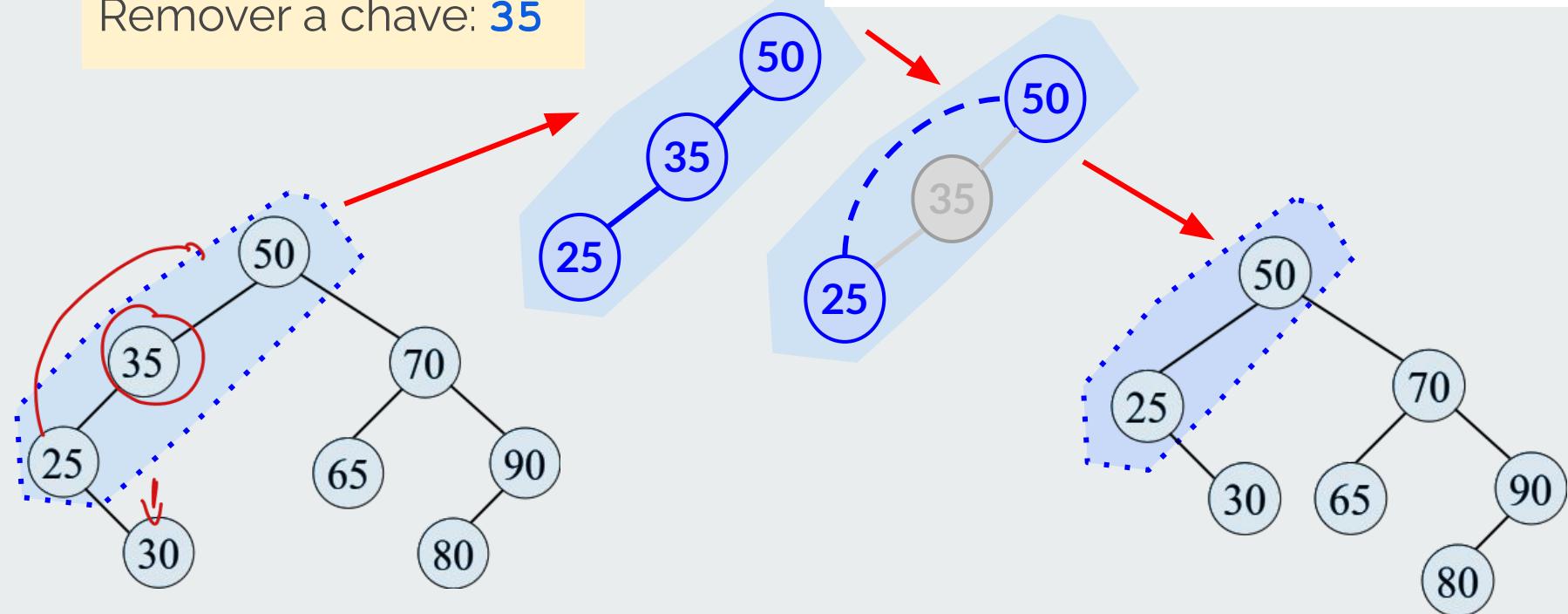


Removemos o nó e ajustamos o ponteiro do pai para **NULL**.

CASO 2. A chave removida possui uma subárvore vazia.

Remover a chave: 35

(relembrando) remoção em árvore binária...

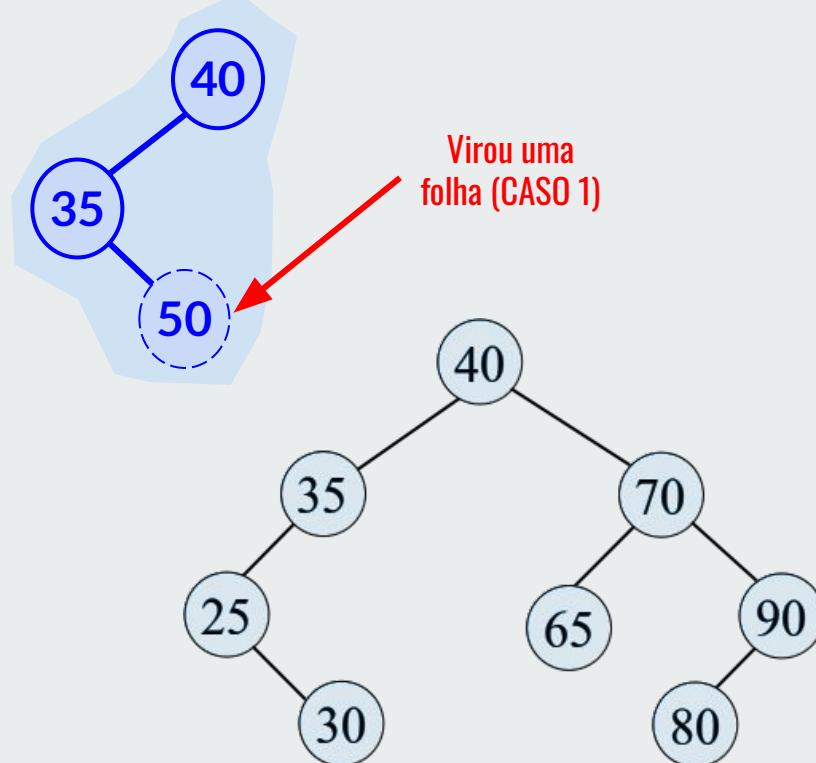
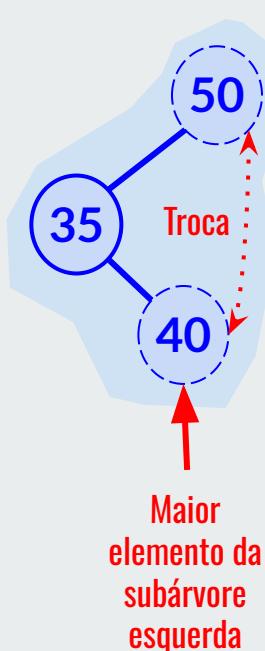
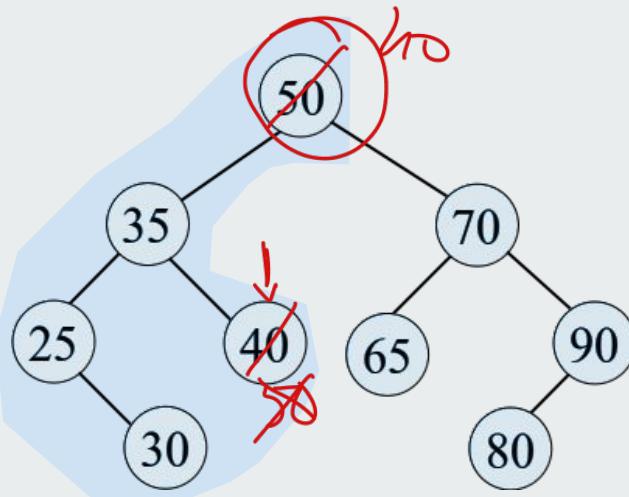


O filho do nó a ser removido substitui o nó.
Ligamos o pai do nó diretamente ao seu único filho.

CASO 3. A chave removida possui duas subárvores não vazias.

Remover a chave: 50

(relembrando) remoção em árvore binária...



Substitui o elemento a ser removido pelo **maior elemento da subárvore esquerda** ou o menor da subárvore direita. Daí, **reduzimos ao caso 1** ou caso 2.

Operações Básicas

Pseudocódigo

(...)

```
raiz.h = 1+max(raiz->esq.h, raiz->dir.h)
```

```
bl= raiz.h
```

```
if (bl > 1 e chave < raiz->esq.chave bl>=0)
```

```
    return rotDireita(raiz); //caso 1
```

```
if (bl > 1 e chave > raiz->esq.chave bl<0) }
```

```
    raiz->esq = rotEsquerda(raiz->esq); }
```

```
    return rotDireita(raiz); //caso 2
```

```
if (bl < -1 e chave > raiz->dir.chave bl<=0) }
```

```
    return rotEsquerda(raiz); //caso 3
```

```
if (bl < -1 e chave < raiz->dir.chave bl>0) }
```

```
    raiz->dir = rotDireita(raiz->dir); }
```

```
    return rotEsquerda(raiz); //caso 4
```

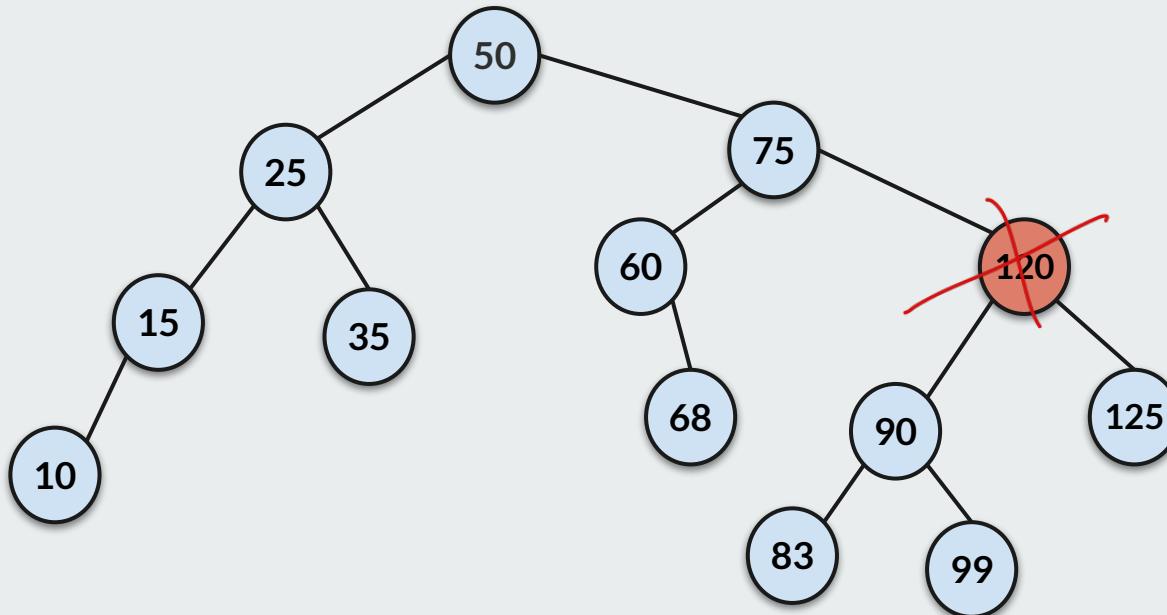
```
}
```

} } } }



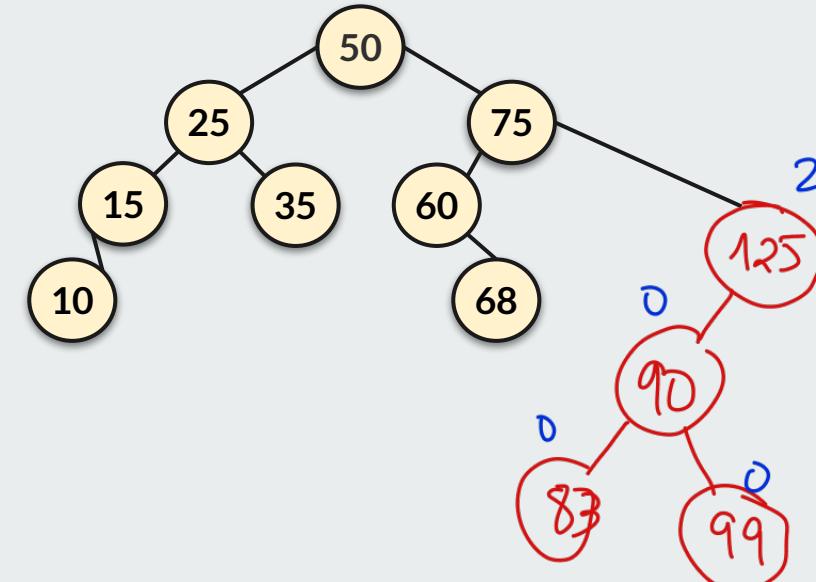
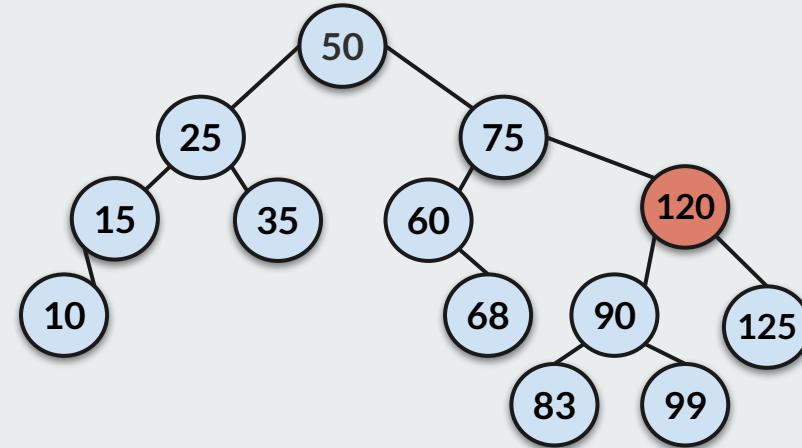
Inclusão em Árvore AVL

Remover a chave: **120**



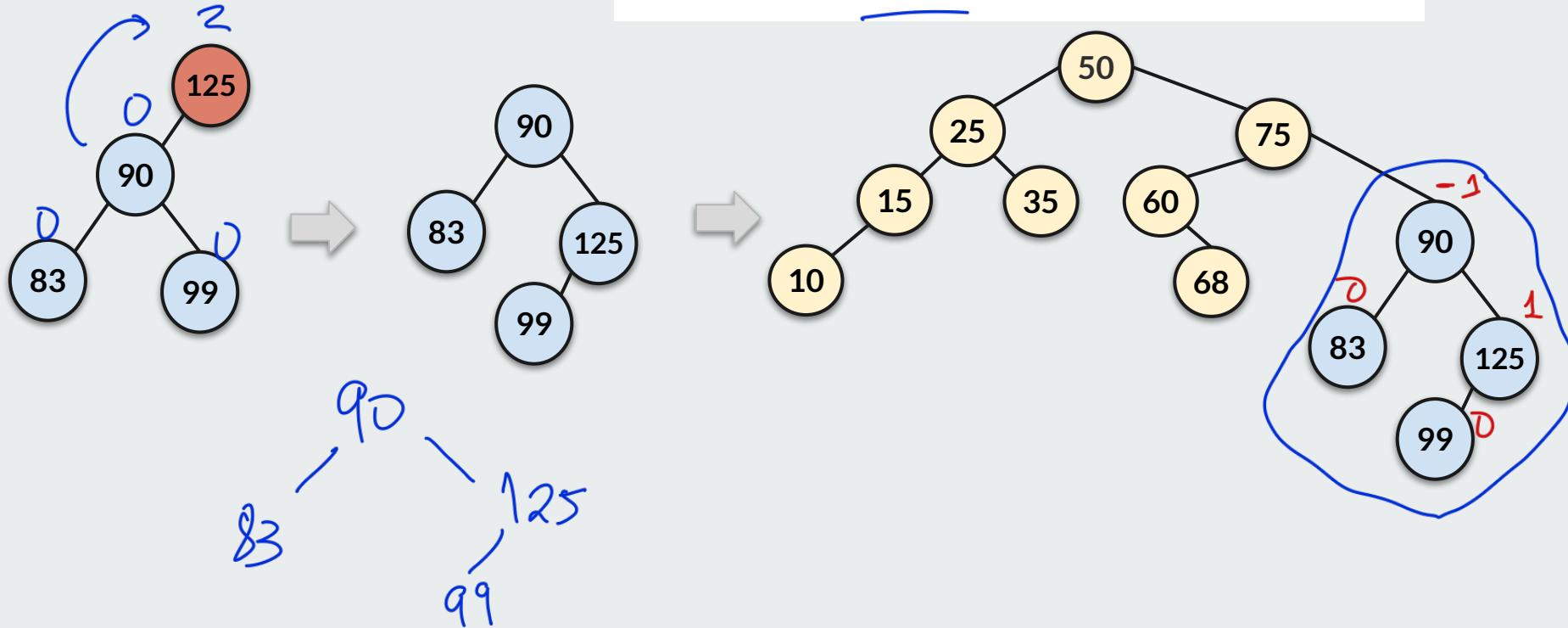
Inclusão em Árvore AVL

Remover a chave: **120**



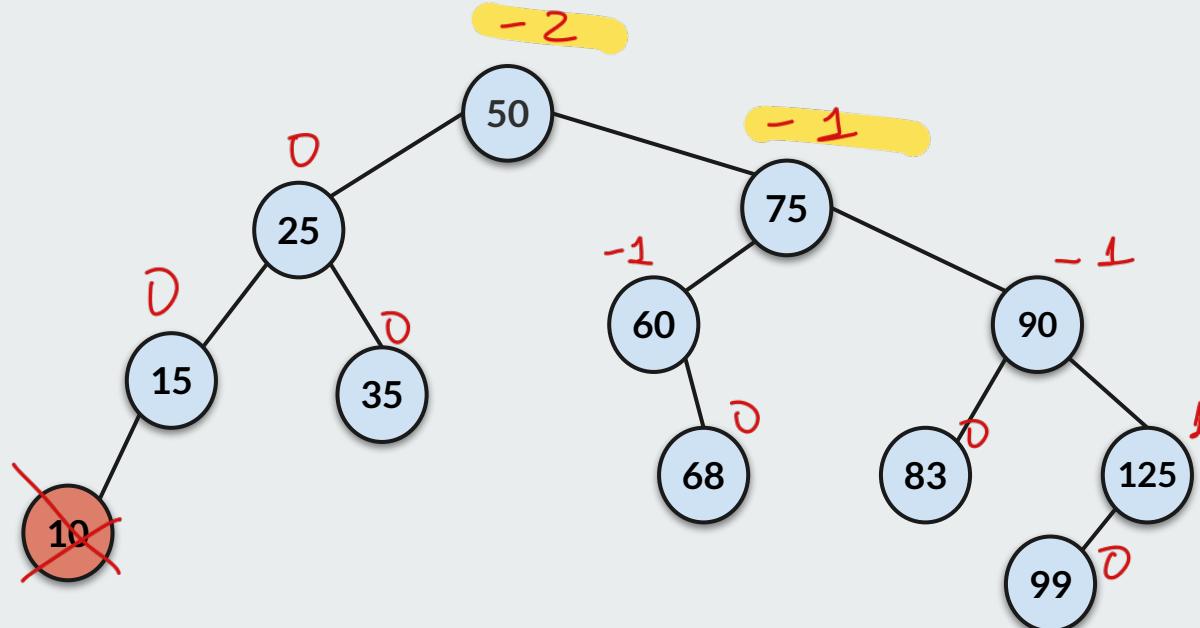
Remoção em Árvore AVL

```
if (bl > 1 e chave < raiz->esq.chave bl>=0)
    return rotDireita(raiz); //caso 1
```



Remoção em Árvore AVL

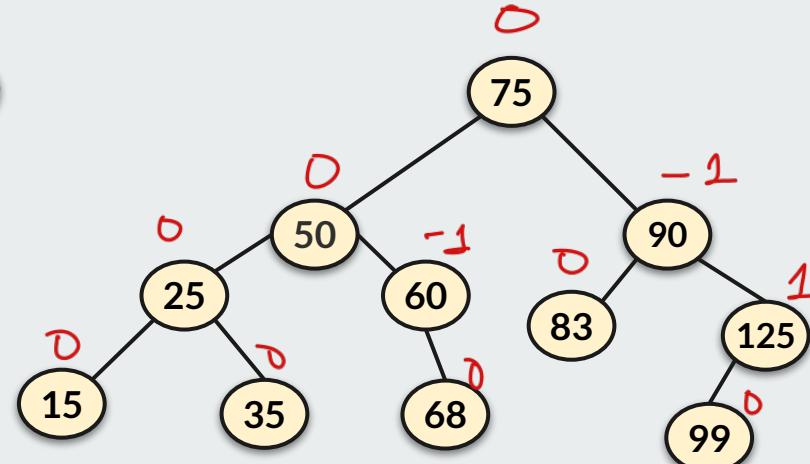
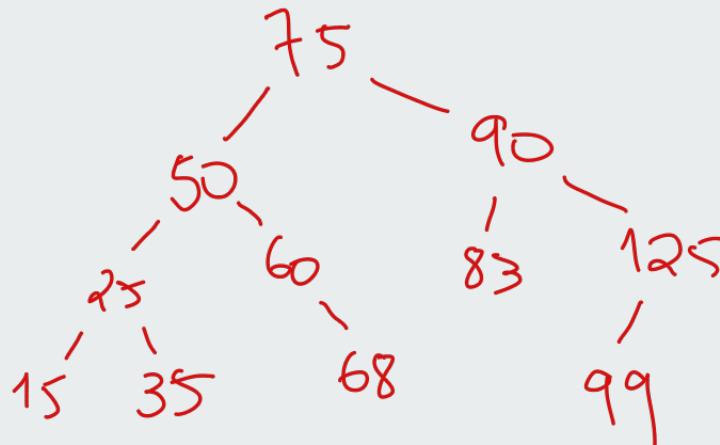
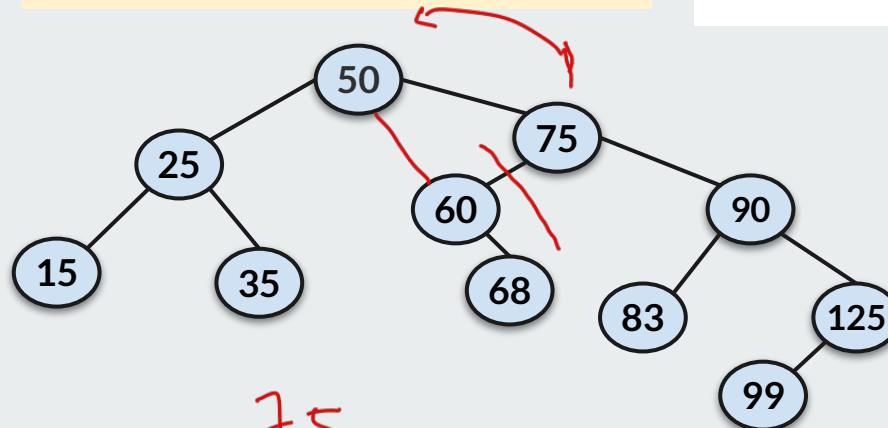
Remover a chave: 10



Remoção em Árvore AVL

Remover a chave: 10

```
if (bl < -1 e chave > raiz->dir.chave bl<=0)
    return rotEsquerda(raiz); //caso 3
```



Vamos programar?



1. Implementação da árvore AVL com as funções de:
 - a. *Buscar*
 - b. *Inserir*
 - c. *Remover*

