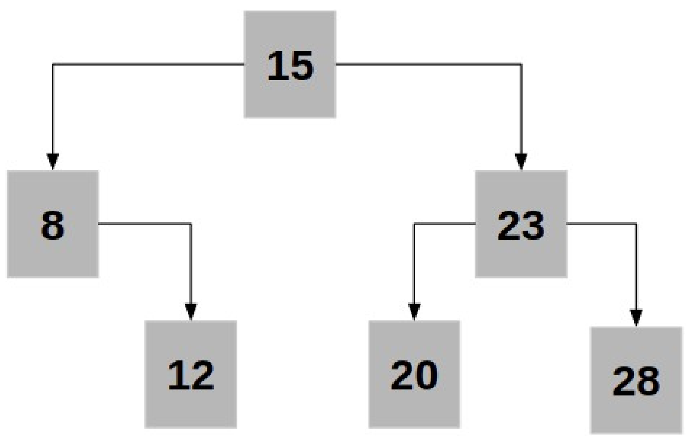
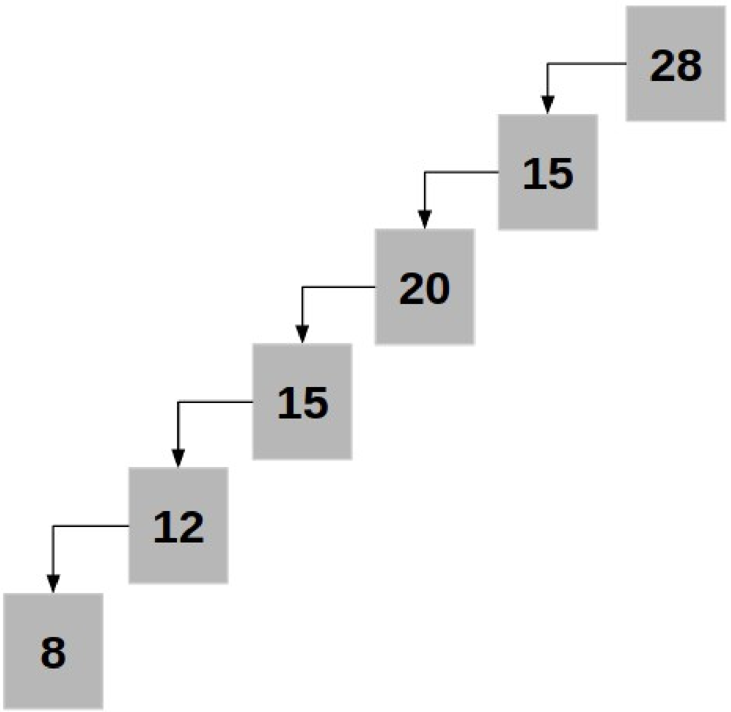
DCC405 – ESTRUTURA DE DADOS II

Aula 08 – Árvores AVL

Árvores AVL

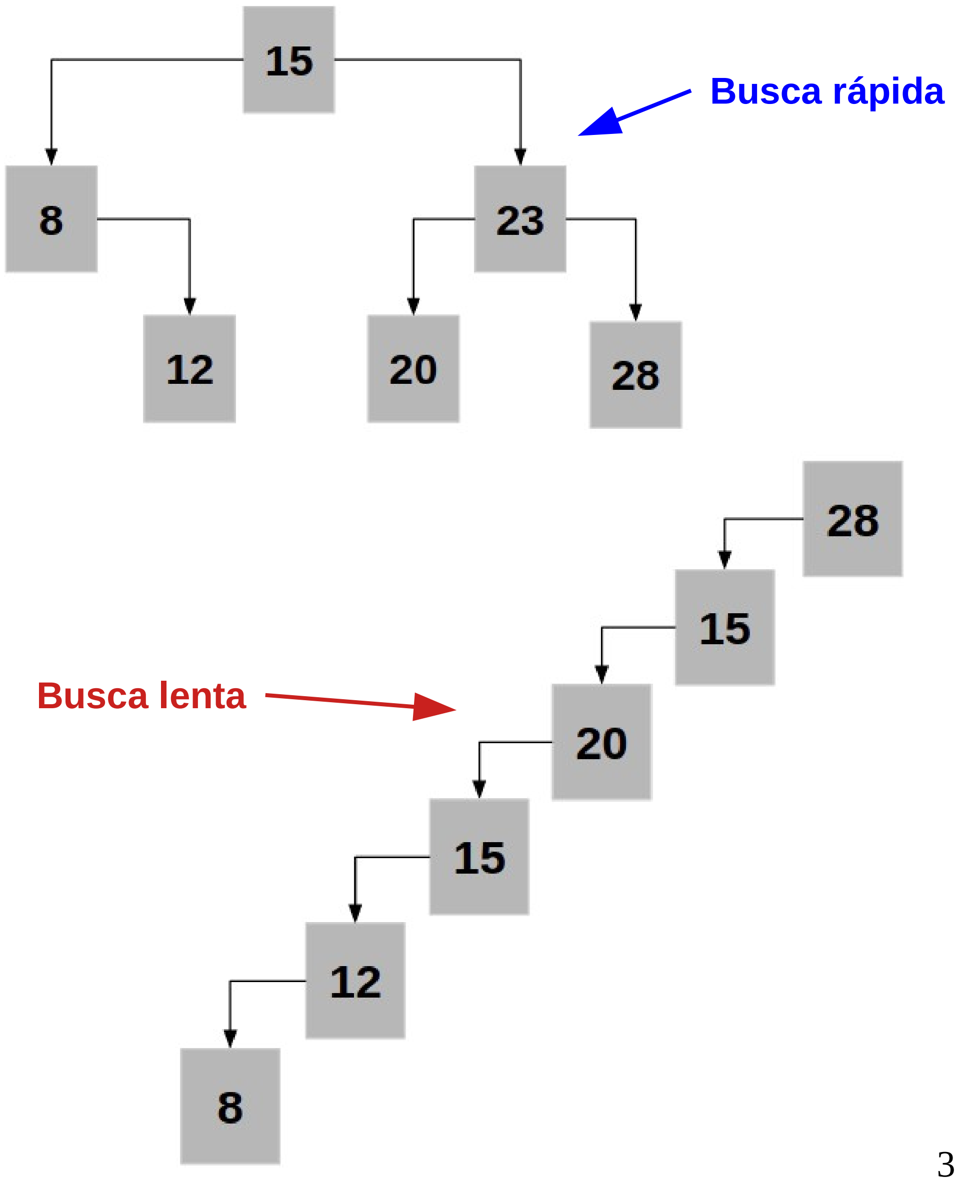
Revisando - Árvore Binária de Busca: Vimos que a ordem de inserção determina o formato de uma Árvore de Busca Binária





Árvores AVL

Revisando - Árvore Binária de Busca:
Vimos também que isso era a diferença entre ela se comportar como uma busca binária ou uma busca sequencial.

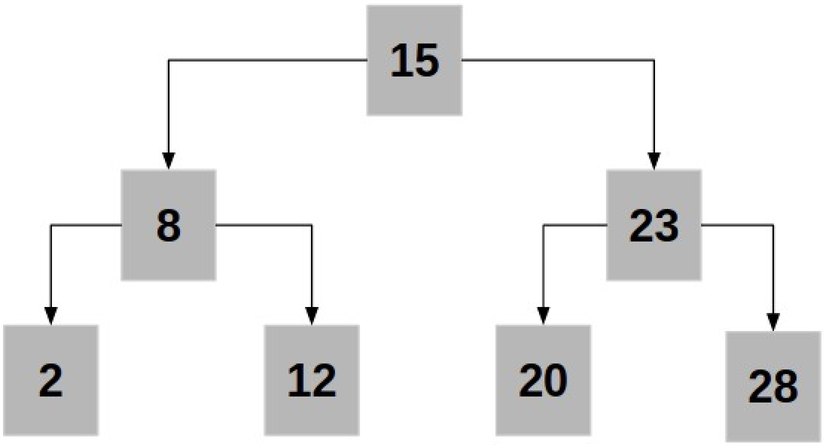


Árvores AVL

Por isso:

Balanceamento é muito importante.

Contudo o balanceamento perfeito é computacionalmente caro.



Árvores AVL

→ Para mitigar um pouco o custo computacional, a ideia é fazer um balanceamento **“bom”**, onde a árvore pode estar “um pouco” desbalanceada, e isso é aceitável.

→ **Para isso podemos aplicar o algoritmo de Adelson-Velskii e Landis (as Árvores AVL)**

Árvores AVL

- Nome com origem em seus inventores:
  - Georgii Adelson-Velsky e Yevgeniy Landis;
  - Publicaram um documento chamado: "*Algoritmos para organização da informação*“, em 1962;

- Uma árvore binária de pesquisa **T** é denominada **AVL** se:
  - Para todos nós de **T**, as alturas de suas duas subárvores diferem **no máximo de uma unidade**.

Para cada inserção ou exclusão no pior caso é de **O(log *n*)**.

Árvores AVL

→ AVL é uma **árvore de busca binária balanceada** com relação à **altura de suas subárvores**.

→ Uma árvore AVL verifica a altura das subárvores da esquerda e da direita, garantido que essa diferença não seja maior que **+1 ou -1.**

13-3 Árvores AVL

Uma ***árvore AVL*** é uma árvore de busca binária de ***altura balanceada***: para cada nó x, a diferença entre as alturas das subárvores à esquerda e à direita de x é no máximo 1. Para implementar uma árvore AVL, mantemos um atributo extra em cada nó: $x.h$ é a altura do nó x. Como em qualquer outra árvore de busca binária T, supomos que $T.raiz$ aponta para o nó raiz.

Cormen et al. – Algoritmos: Teoria e Prática – Cap 13.

Árvores AVL

- **Como saber se a árvore está desbalanceada?**
  - Verificando se existe algum nó “desregulado”.

- **Como saber se um nó está desregulado?**
  - Subtraindo as alturas das suas subárvores
  - Fator de Balanceamento

- Por questões de **eficiência**, estas diferenças são pré calculadas e armazenadas nos nós correspondentes, sendo atualizadas durante as operações.

Árvores AVL – Fator de Balanceamento

→ Uma árvore AVL verifica a altura das subárvores da esquerda e da direita, garantido que essa diferença não seja maior que **+1 ou -1.**

Esta diferença é seu **Fator de Balanceamento**

$$f_b = h_{esq} - h_{dir}$$

OBS: **A altura de uma <u>árvore vazia</u> é -1**

→ **O fator de balanceamento, ou alternativamente a altura do nó, é armazenado no próprio nó**

Árvores AVL – Propriedades

- Procurar manter todas as folhas mais ou menos na mesma altura de forma a respeitar o **Fb < 2**

- Ou seja,

  Para todo nó

  | altura(esq) - altura(dir) | < 2

Árvores AVL

Relembrando as definições:

- **Altura** de uma árvore (também denominada profundidade) é a distância entre x e o seu descendente mais afastado. Mais precisamente, a altura de x é o número de passos do mais longo caminho que leva de x até uma folha somando um.
  - Por definição a **altura de uma árvore vazia é -1**

```
            E                 Altura dessa árvore é 3
         /     \
      D           I           Altura de I é 2
    /           /    \
   B        G          K      Altura de K é 1
  /  \     /  \       /
 A    C   F    H     J        Altura de J é 0
```

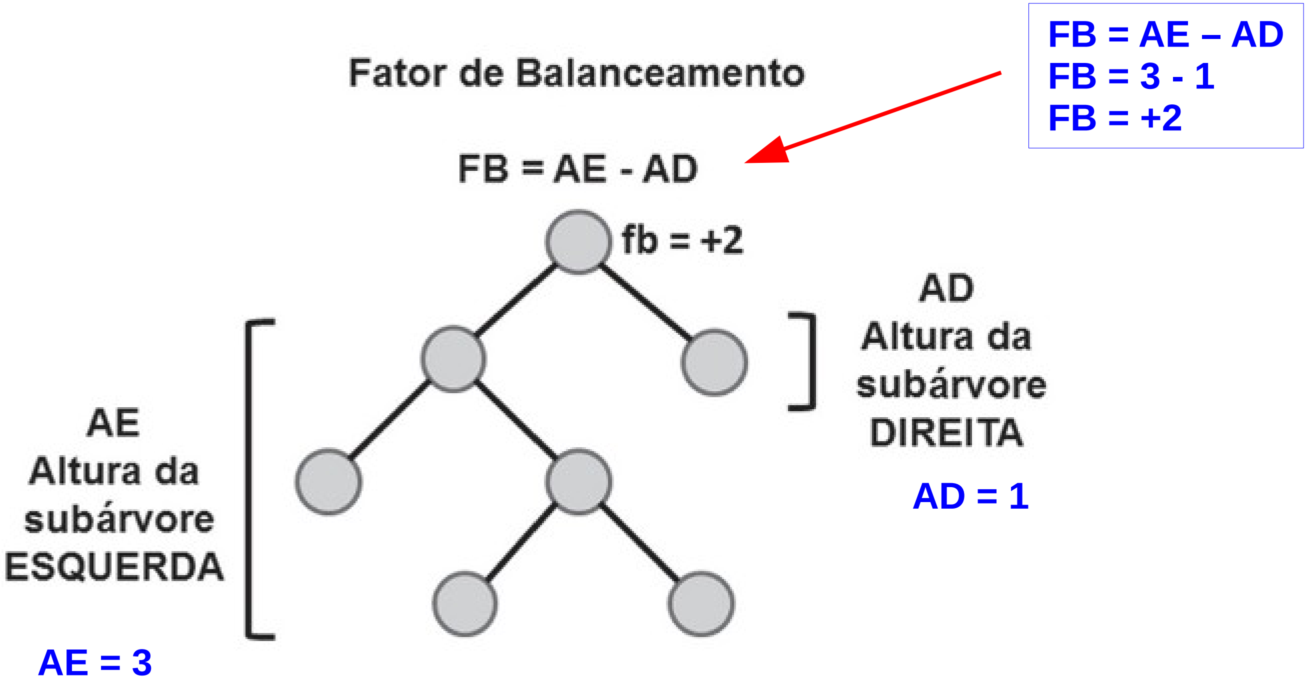
Árvores AVL - Rotações

→ Trata-se de um tipo de árvore que permite o rebalanceamento local da árvore, ou seja, apenas a parte afetada pela **inserção** ou **remoção** é rebalanceada.

Para manter o balanceamento, a **árvore AVL** faz uso de **rotações** **simples** e **duplas** na etapa de rebalanceamento, o qual ocorre a cada **inserção** ou **remoção**.

Árvores AVL - Rotações

→ Por meio destas rotações, a árvore AVL busca manter-se como uma árvore binária **quase completa.** Assim, o custo de qualquer algoritmo é *O(log n)*.



Árvores AVL - Rotações

Se o **fator de balanceamento** for maior do que +1 ou menor do que -1 em um nó da árvore AVL, então a árvore deve ser balanceada naquele nó.

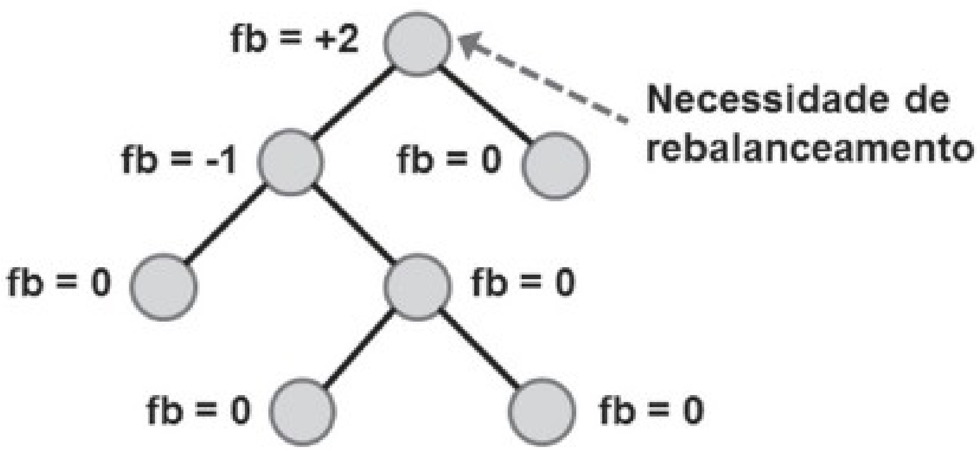


FIGURA 11.39

Árvores AVL - Rotações

→ A figura abaixo mostra uma árvore AVL e o fator de balanceamento de cada nó. Esta árvore é obtida ao se inserir os valores (1, 2, 3, 10, 4, 5, 9, 7, 8, e 6), nessa ordem, na árvore:

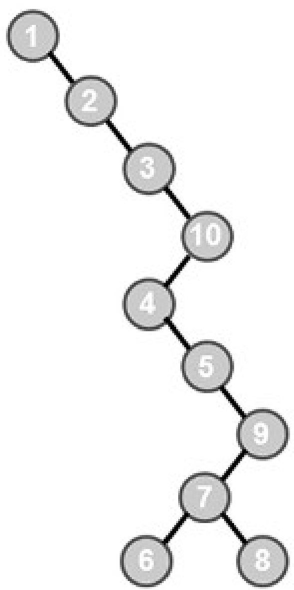


FIGURA 11.37

Árvore Binária de Busca

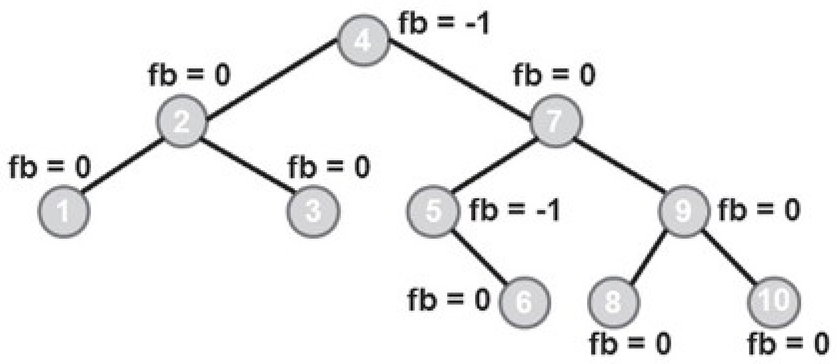


Figura 11.40

Árvore AVL

Árvores AVL - Implementação

Colocar imagem do código e explicar

Árvores AVL - Rotações

→ A operação básica usada para balancear uma árvore AVL é a **rotação**. É por meio dela que a árvore AVL busca manter-se como uma árvore binária quase completa. Ao todo, existem dois tipos de rotação: **simples** e **dupla**.

→ Os dois tipos diferem entre si pelo **sentido da inclinação** entre o nó **pai** e **filho**.

Árvores AVL - Rotações

→ **Rotação Simples:** o nó desbalanceado (pai), seu filho e o seu neto estão todos no mesmo sentido de inclinação.

→ **Rotação dupla:** o nó desbalanceado (pai) e seu filho estão inclinados no sentido inverso ao neto. Equivale a duas rotações **simples**.

Existem duas rotações **simples** e duas **duplas**:

→ **Rotação simples** à **direita** ou **rotação RSD** (LL)**.**

→ **Rotação simples** à **esquerda** ou **rotação RSE** (RR)**.**

→ **Rotação dupla** à **direita** ou **rotação RDD** (LR)**.**

→ **Rotação dupla** à **esquerda ou rotação RDE** (RL)**.**

Rotação RSD – Rotação Simples à Direita

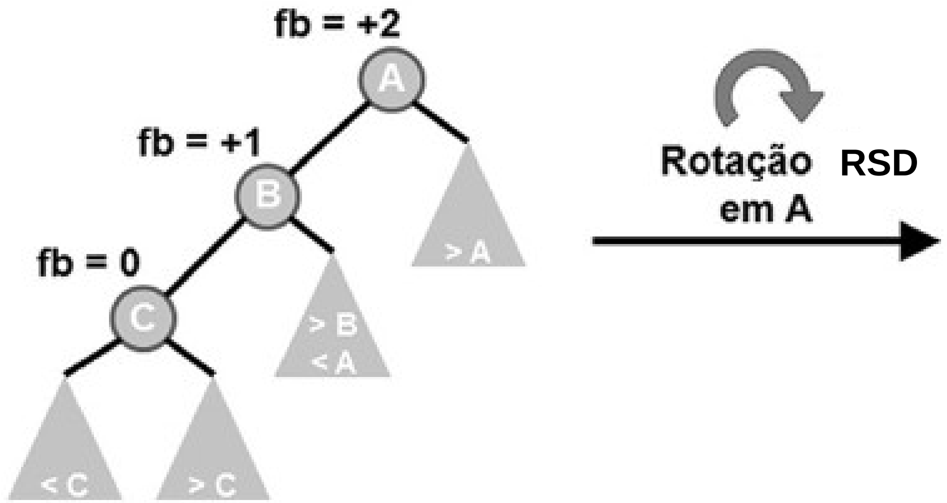


FIGURA 11.43

Árvores AVL - Rotações

Rotação RSD – Rotação Simples à Direita

A Figura 11.45 mostra um exemplo passo a passo desse tipo de rotação.

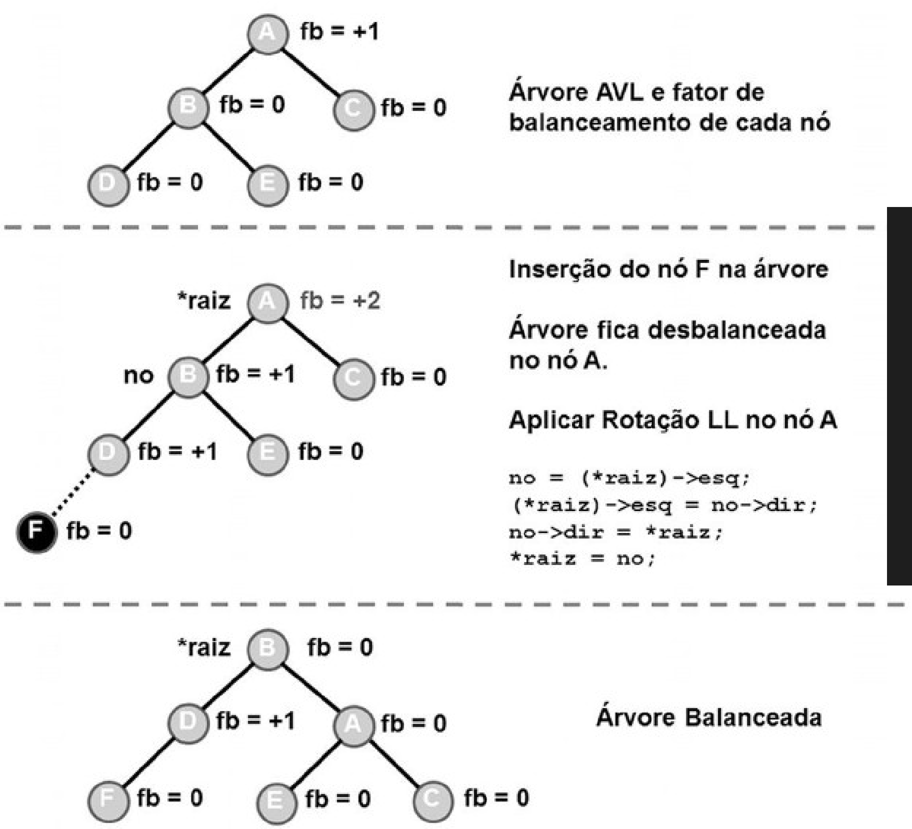


FIGURA 11.45

```
//Rotação Simples à Direita
void rotacaoRSD(NO *root) {
    NO *b;
    b = root->left;
    root->left = b->right;
    b->right = root;
    root->alt = maior(alt_NO(root->left), alt_NO(root->right)) + 1;
    b->alt = maior(alt_NO(b->left), root->alt) + 1;
    root = b;
}
```

Rotação RSE – Rotação Simples à Esquerda

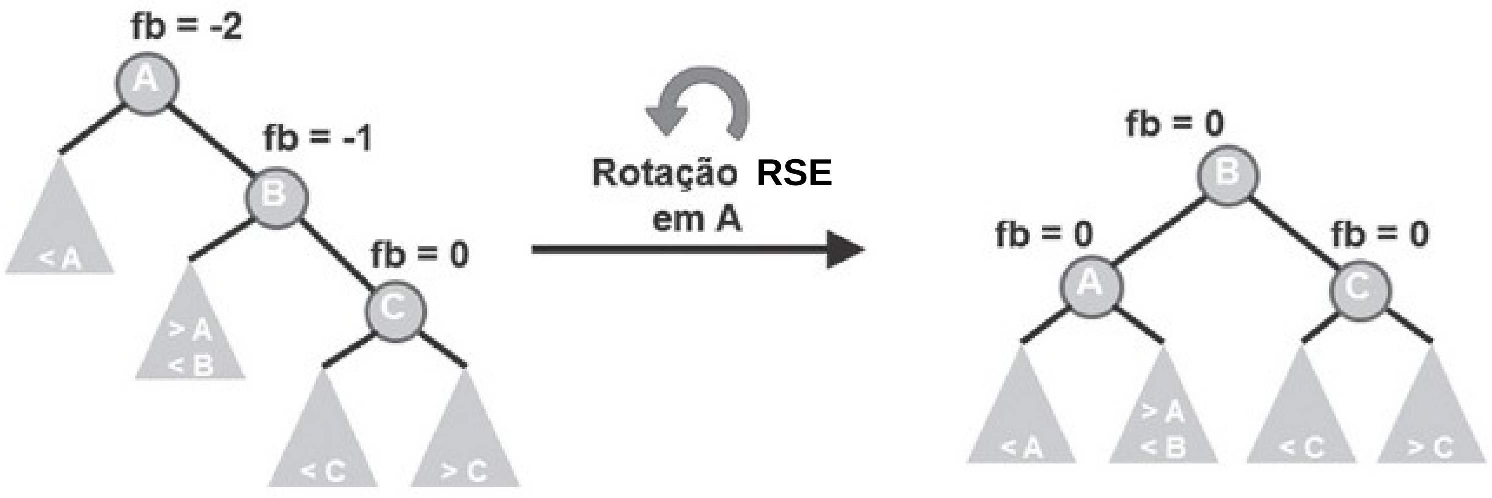


FIGURA 11.46

Árvores AVL

Rotação RSE – Rotação Simples à Esquerda

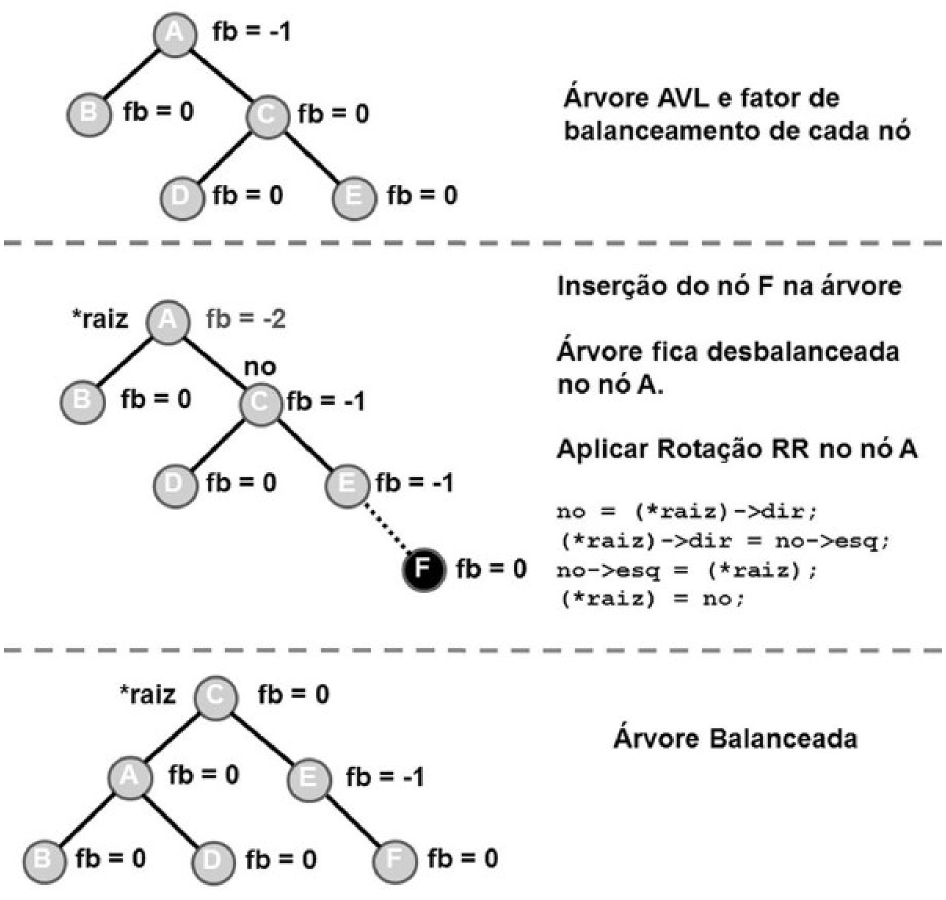


FIGURA 11.48

```
//Rotação Simples à Esquerda
void rotacaoRSE(NO *root) {
    NO *b;
    b = root->right;
    root->right = b->left;
    b->left = root;
    root->alt = maior(alt_NO(root->left), alt_NO(root->right)) + 1;
    b->alt = maior(alt_NO(b->right), root->alt) + 1;
    root = b;
}
```

Árvores AVL - Rotações

Rotação RDD – Rotação Dupla à Direita

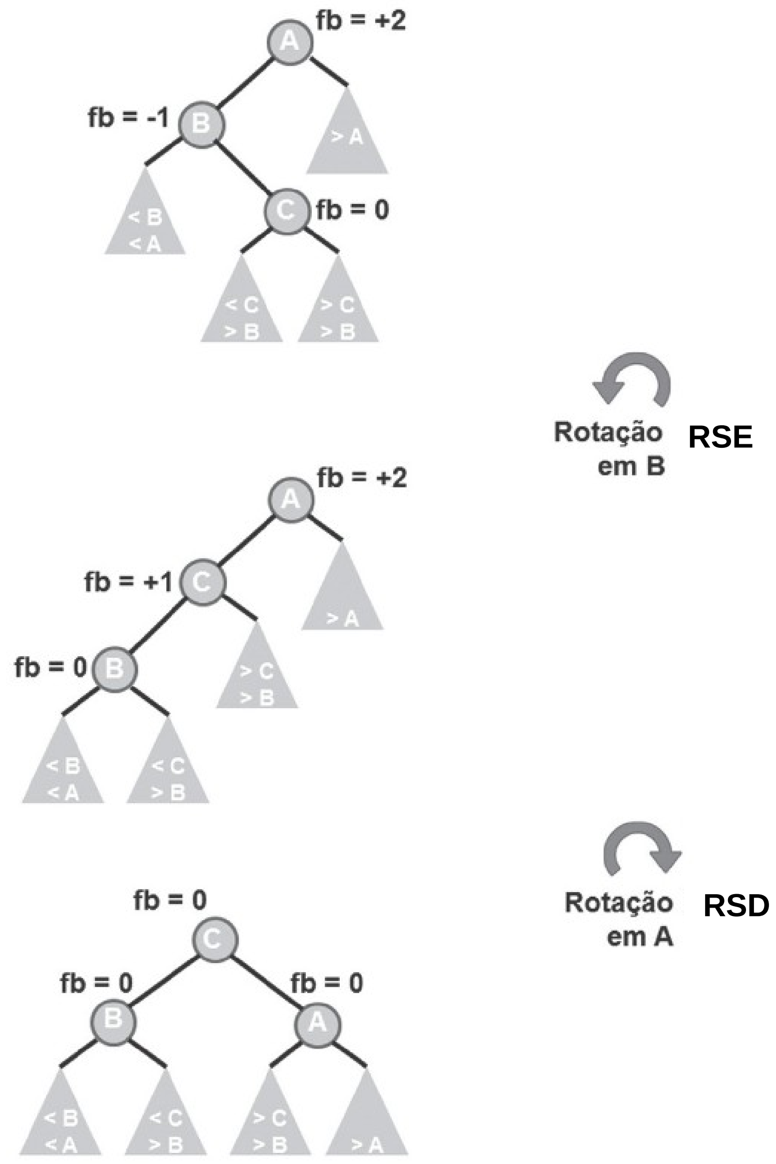


FIGURA 11.49

Árvores AVL - Rotações

Rotação RDD – Rotação Dupla à Direita

```
//Rotação Dupla à Direita
void rotacaoRDD(NO *root) {
    rotacaoRSE(root->left);
    rotacaoRSD(root);
}
```

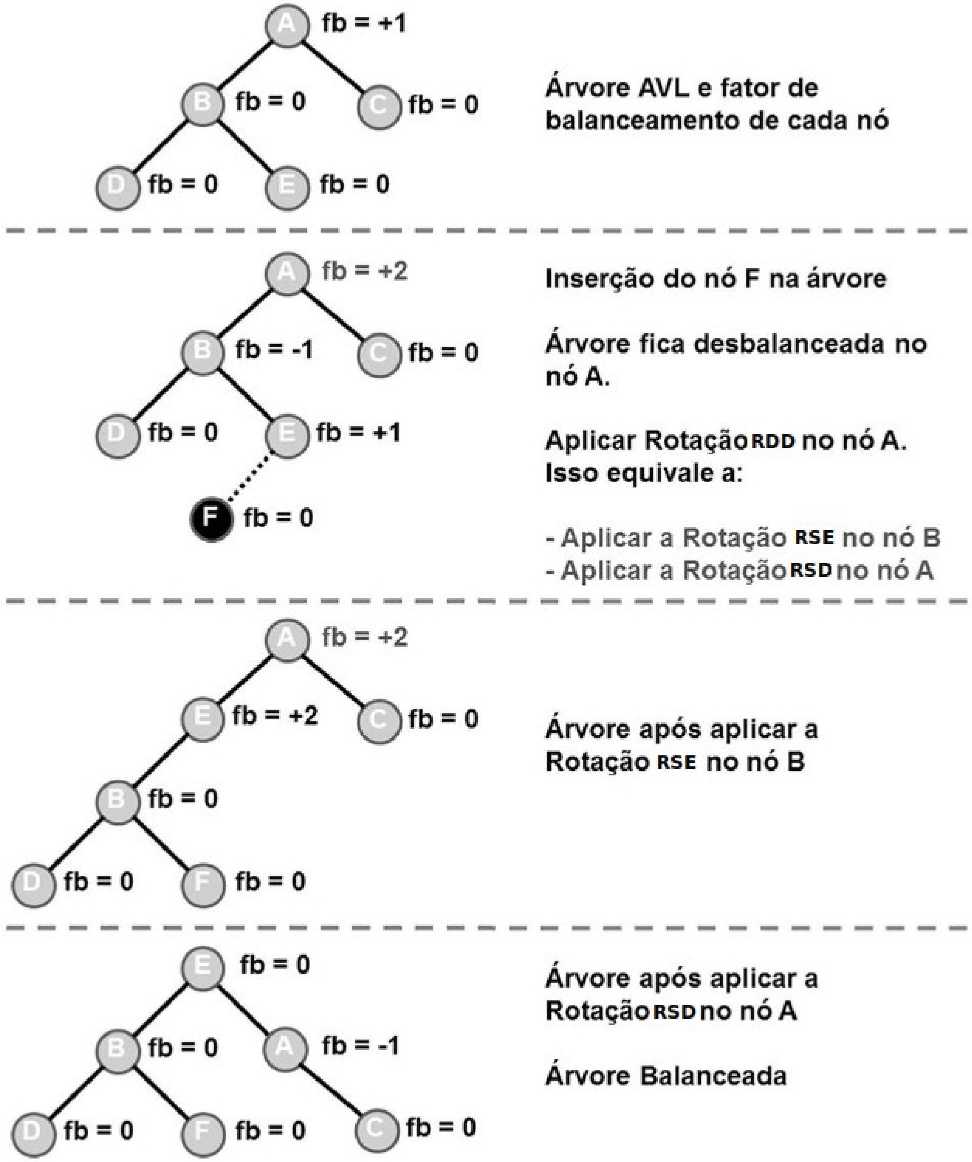


FIGURA 11.51

Árvores AVL - Rotações

Rotação RDD – Rotação Dupla à Esquerda

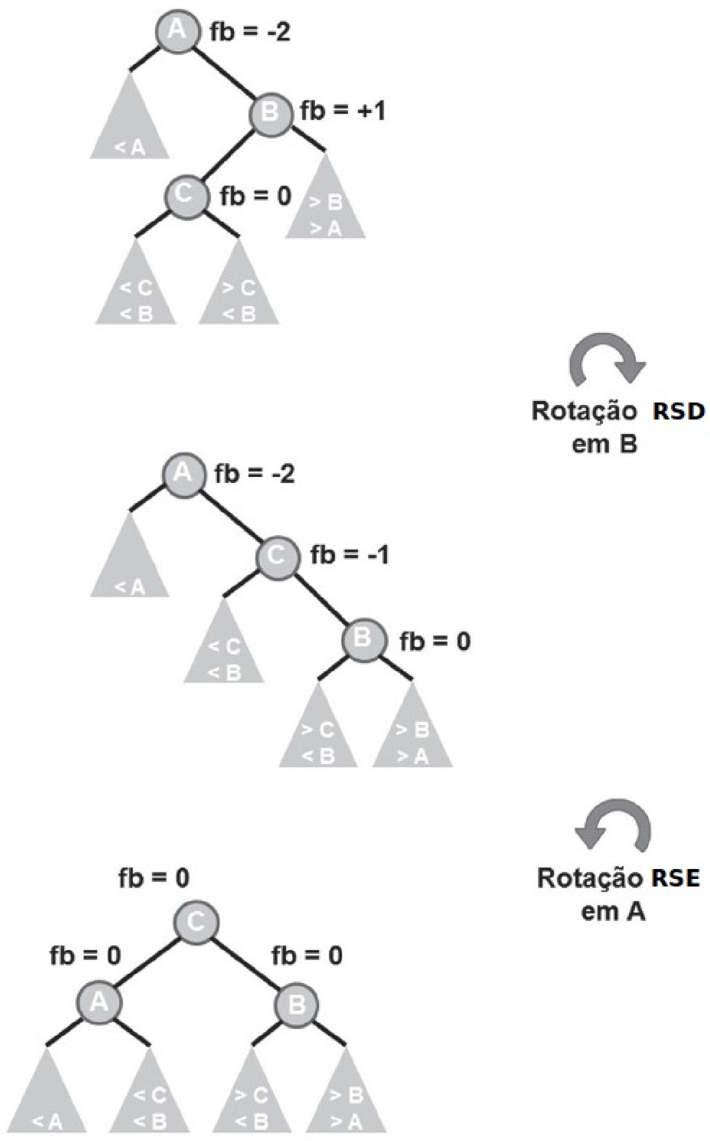


FIGURA 11.52

Árvores AVL - Rotações

Rotação RDD – Rotação Dupla à Esquerda

```
//Rotação Dupla à Esquerda
void rotacaoRDE(NO *root) {
    rotacaoRSD(root->right);
    rotacaoRSE(root);
}
```

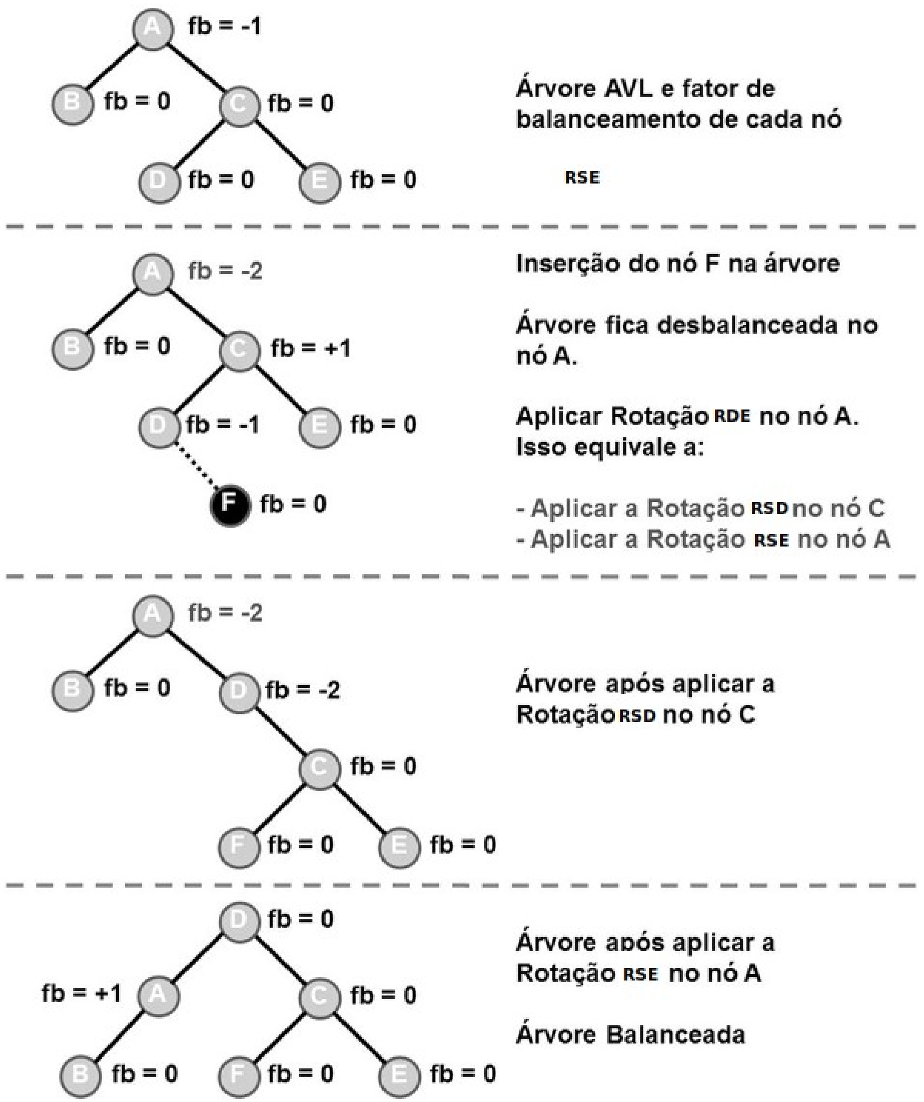


FIGURA 11.54

Árvores AVL - Rotações

Consolidando as rotações - *O(1)*

Fator de Balanceamento de A	Fator de Balanceamento de B	Posições dos nós B e C em relação ao nó A	Rotação
+2	+1	B é filho à esquerda de A C é filho à esquerda de B	RSD
-2	-1	B é filho à direita de A C é filho à direita de B	RSE
+2	-1	B é filho à esquerda de A C é filho à direita de B	RDD
-2	+1	B é filho à direita de A C é filho à esquerda de B	RDE

Árvores AVL – Inserindo um nó

A inserção de um novo nó na árvore AVL é exatamente igual à inserção na árvore binária de busca. No entanto, uma vez inserido o novo nó na árvore, começam a surgir as diferenças entre uma simples árvore binária de busca e uma árvore AVL.

→ Para inserir um nó, temos que percorrer um conjunto de nós da árvore até chegar ao nó folha que irá se tornar o pai do novo nó. Uma vez inserido este nó, devemos voltar pelo caminho percorrido e calcular o fator de balanceamento de cada um dos nós e, se necessário, aplicar uma das quatro rotações para reestabelecer o balanceamento da árvore.