Estruturas de Dados 2

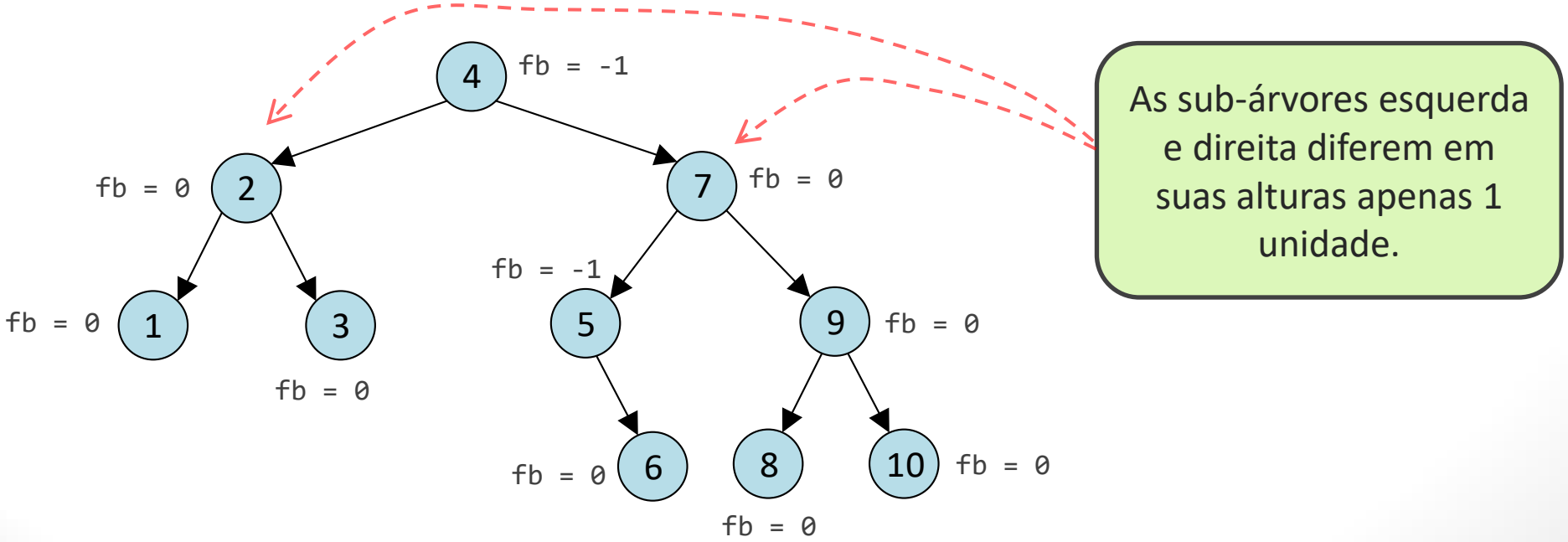
aula 05 – árvore AVL – parte 1

Antonio Angelo de Souza Tartaglia
angelot@ifsp.edu.br

Estrutura de Dados 2

árvores Balanceadas

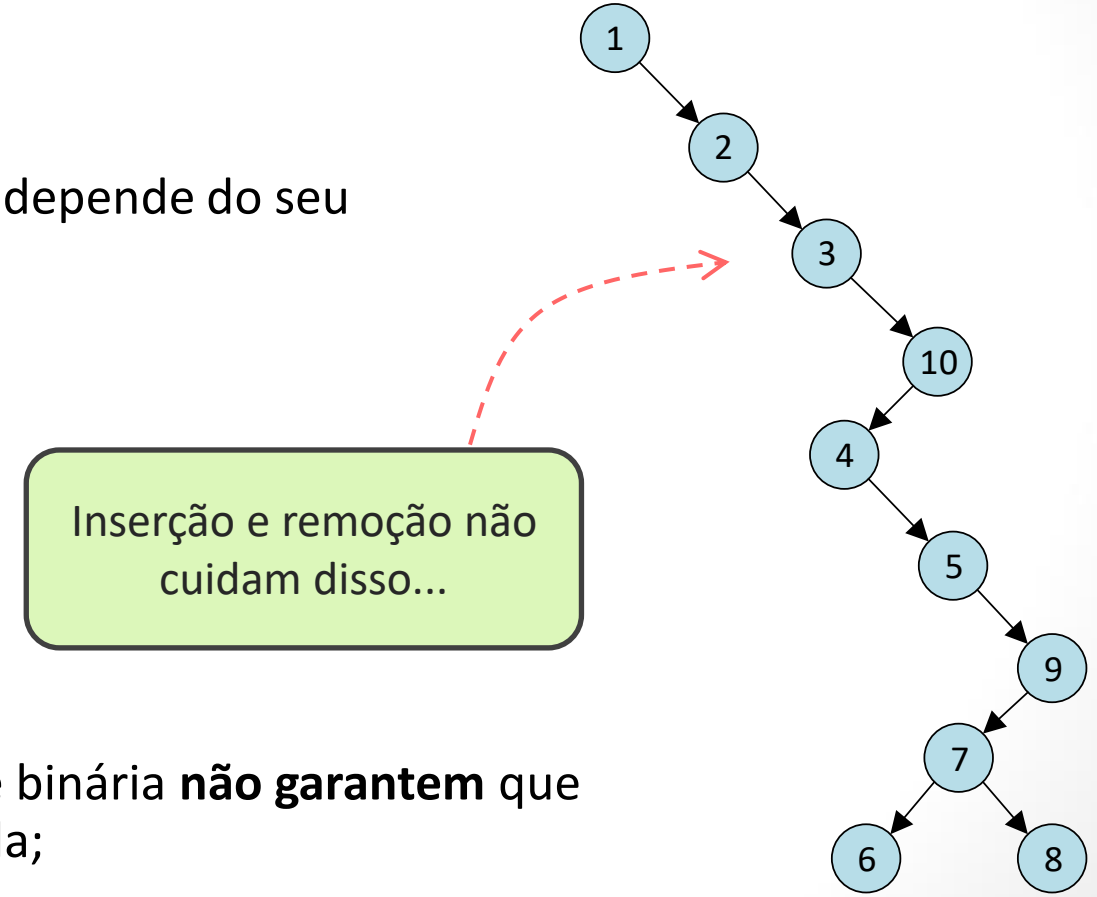
- árvore Balanceada:
  - É uma árvore Binária onde as alturas das sub-árvores esquerda e direita de cada nó, diferem no máximo em 1 unidade;
  - Essa diferença é chamada de Fator de Balanceamento do nó.



Estrutura de Dados 2

árvores Balanceadas

- árvore Balanceada:
  - A eficiência da busca em uma árvore Binária depende do seu **balanceamento**.



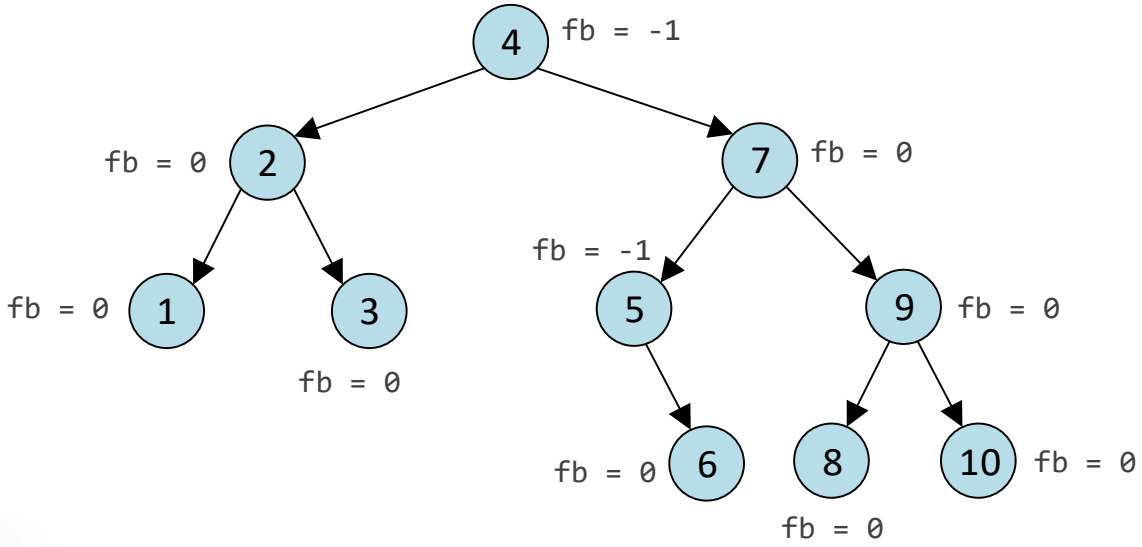
- Problema:
  - Algoritmos de inserção e remoção da árvore binária **não garantem** que a árvore gerada a cada passo seja balanceada;
  - Sequência de inserção em ordem de **escada**.

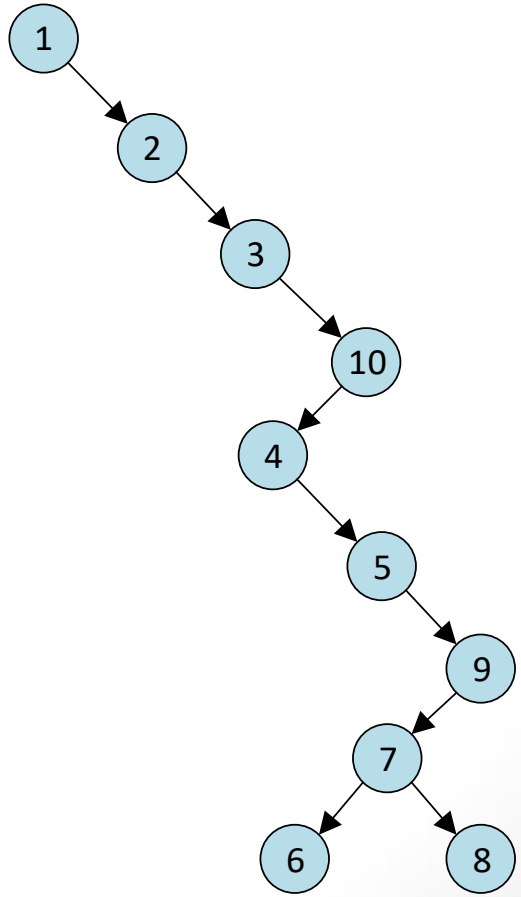
Estrutura de Dados 2

árvores Balanceadas

- Custo da inserção, busca e remoção em árvores Binárias:
  - Balanceada: $O(log\ n)$;
  - Não Balanceada: $O(n)$.

- Em que n, corresponde ao número de nós da árvore.





árvores Balanceadas

- Solução para o problema de balanceamento?
  - Modificar as operações de **inserção** e **remoção** da árvore.

- Exemplos de árvores Balanceadas:
  - árvore AVL;
  - árvore 2-3-4;
  - árvore **Red-Black**, também conhecida como **Rubro-Negra** ou **Vermelha e Preta**.

Estas são árvores que já consideram a necessidade do Balanceamento durante as operações de inserção e remoção.

Estrutura de Dados 2

árvore AVL

- É um tipo de árvore Binária Balanceada;
- Criada por **A**delson-**V**elskii e **L**andis, em 1962.

- Permite o rebalanceamento local:
  - Apenas a parte afetada pela inserção ou pela remoção é rebalanceada;
  - Utiliza a técnica de rotações, que podem ser **simples** ou **duplas** na etapa de rebalanceamento.

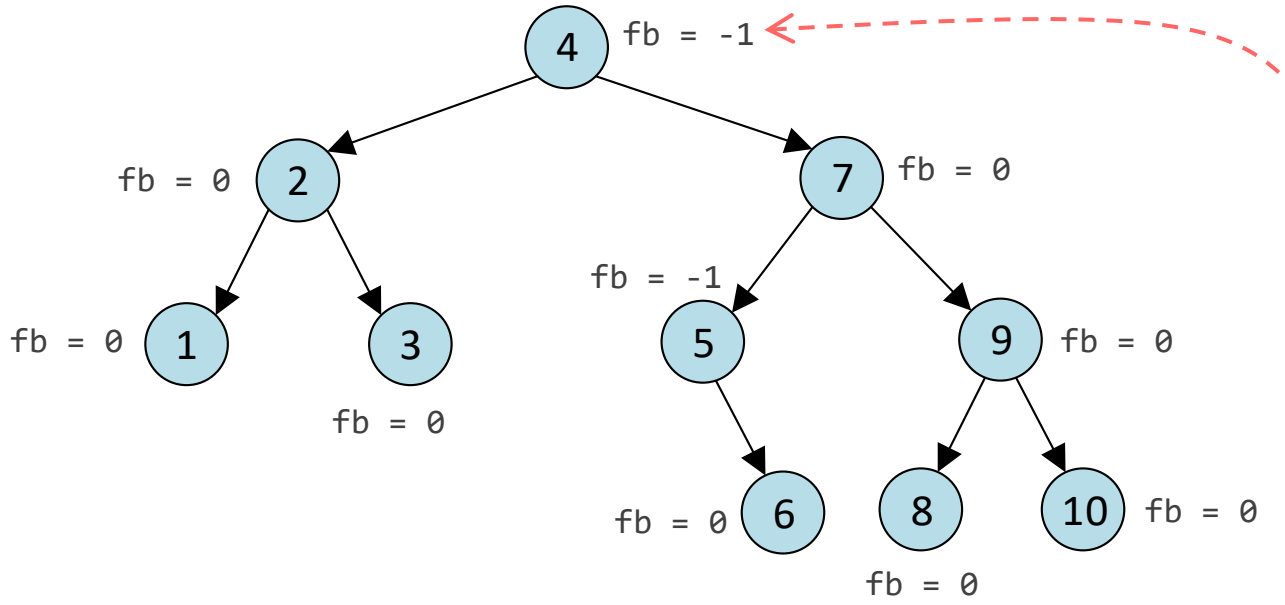
Dependendo de como ficou a árvore após a inserção ou remoção.

árvore AVL

- A árvore AVL busca se manter como uma árvore Binária quase completa;
- Custo de qualquer algoritmo é no máximo $\boldsymbol{O}(\log \boldsymbol{n})$.

Fator de Balanceamento

FB = AE - AD



fb = -1: árvore da direita é 1 elemento maior;

fb = 0: árvores direita e esquerda são iguais;

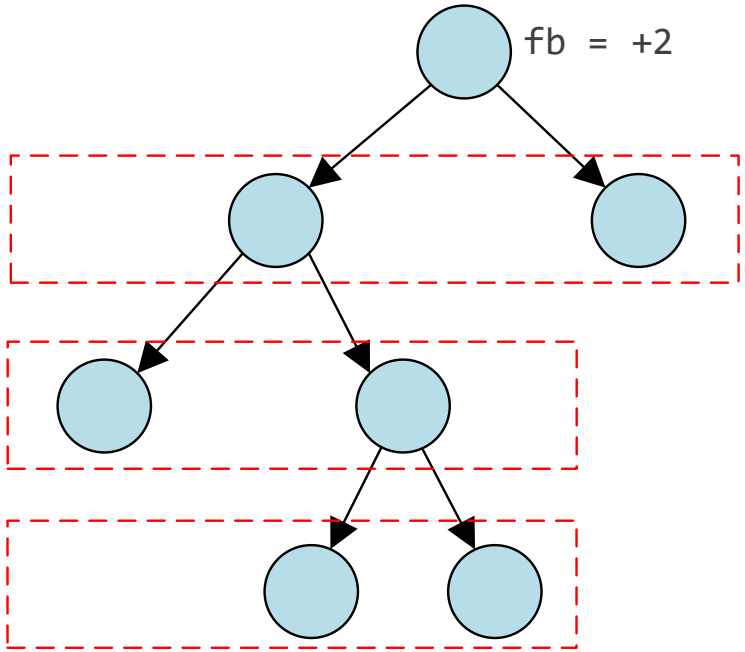
fb = +1: árvore da esquerda é maior 1 elemento.

árvore AVL

- Fator de Balanceamento ou fb:
  - Diferença nas alturas das sub-árvores esquerda e direita:
    - Se uma das sub-árvores não existir, sua altura será -1.

Fator de Balanceamento

FB = AE - AD



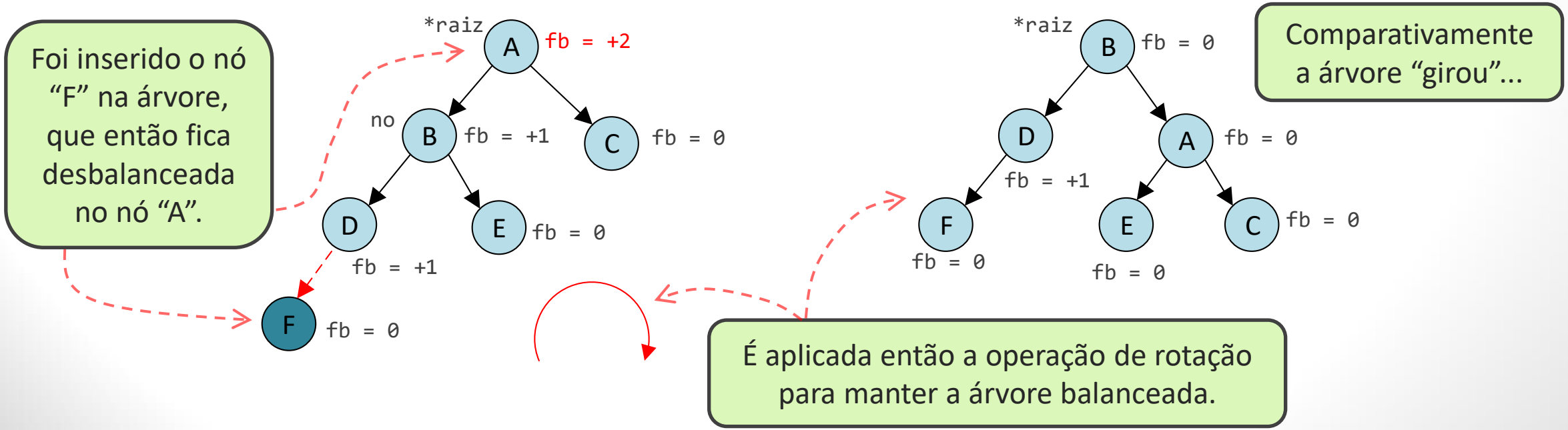
Altura da sub-árvore da direita.

Níveis a mais na sub-árvore da esquerda.

Estrutura de Dados 2

árvore AVL

- Fator de Balanceamento ou `fb`:
  - É o parâmetro usado para balancear a árvore AVL;
  - Assim, em uma árvore AVL, o fb deve ser `+1`, `0` ou `-1`;
  - Se `fb < -1` ou `fb > +1`: a árvore estará desbalanceada e isso deve ser corrigido.



árvore AVL - Implementação

- A implementação é idêntica a da árvore Binária:
  - Para guardar o primeiro nó da árvore utilizamos um ponteiro para ponteiro;
  - Um ponteiro para ponteiro pode guardar o endereço de um ponteiro;
  - Assim fica fácil mudar a raiz da árvore, caso seja necessário.

Normalmente em uma árvore AVL, mudar a raiz é necessário.

árvore AVL - Implementação

- Implementando uma árvore AVL:
  - Arquivo `arvoreAVL.h`, serão definidos:
    - Os protótipos das funções;
    - O tipo de dado armazenado na árvore;
    - O ponteiro árvore.
  - Arquivo `arvoreAVL.c`, serão definidos:
    - O tipo de dado árvore;
    - Implementação de suas funções

Reaproveitando o código desenvolvido para a árvore de Busca Binária, exclua (ou comente) as funções de inserção e remoção, e modifique em todos os pontos onde foi usado "`arvBin`", para "`arvAVL`".
O `CodeBlocks` possui uma função própria para esta operação (ALT+SHIFT+R com todos os arquivos do projeto abertos no editor).

- Com exceção da **inserção** e da **remoção**, as demais funções da árvore AVL são **idênticas** as da árvore Binária.

árvore AVL - Implementação

```c
//arquivo arvoreAVL.h
typedef struct NO *arvAVL;
```

```c
//arquivo main()
#include <stdio.h>
#include <stdlib.h>
#include "arvoreAVL.h"

int main(){
    int x;
    arvAVL *raiz;
```

```c
//arquivo arvoreAVL.c
#include <stdio.h>
#include <stdlib.h>
#include "arvoreAVL.h" //inclui protótipos

struct NO{
    int info;
    int alt; //FB - altura da sub-arvore
    struct NO *esq;
    struct NO *dir;
};
```

Estrutura de Dados 2

```
/*
* Funções auxiliares para tratamento de inserções e remoções em Árvores AVL.
* As três funções são internas ao aquivo arvoreAVL.C, e não têm seu protótipo
* relacionado no arquivo arvoreAVL.h, pois não são exportadas.
*/
```

```
//Calcula a altura de um nó
int alt_no(struct NO *no){
    if(no == NULL){
        return -1;
    }else{
        return no->alt;
    }
}
```

```
//devolve o maior entre dois valores
int maior(int x, int y){
    if(x > y){
        return(x);
    }else{
        return(y);
    }
}
```

```
//Calcula o Fator de Balnceamento de um nó
int fatorBalanceamento_NO(struct NO *no){
    return abs(alt_no(no->esq) - alt_no(no->dir));
}
```

A função abs( ), converte o valor passado em seu parâmetro, para o valor absoluto, ou seja, sem sinal.

árvore AVL - Implementação

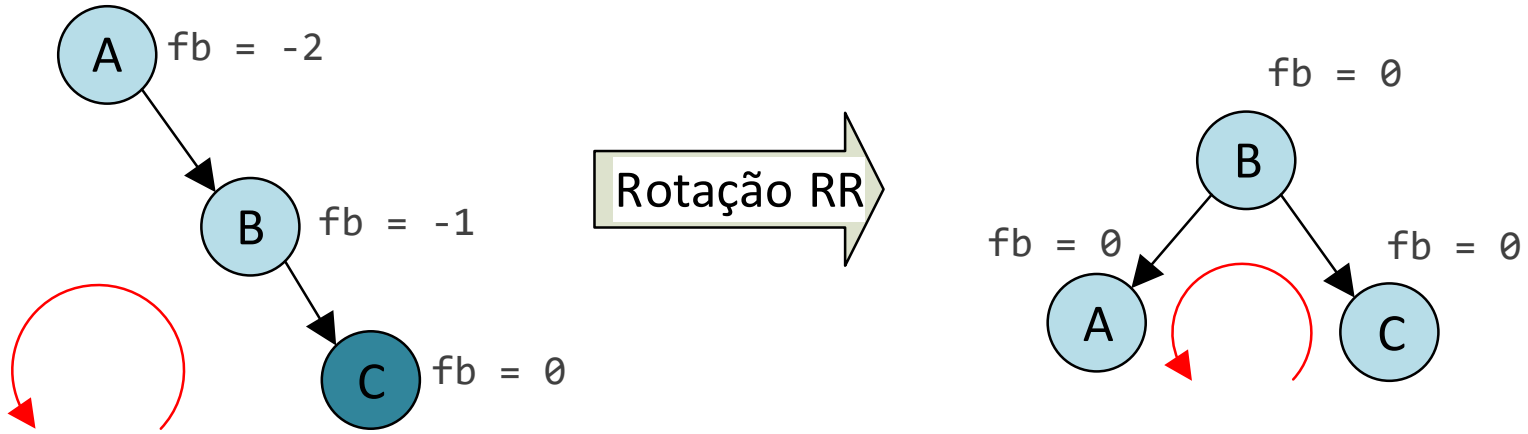
- Rotação:
  - Operação básica para Balanceamento da árvore AVL;
  - Existem dois tipos de rotação: **Simples** e **Dupla**.

- **Simples**:
  - O nó desbalanceado e seu filho estão no mesmo sentido da inclinação.

- **Dupla**:
  - O nó está desbalanceado e seu filho está inclinado no sentido inverso ao Pai;
  - Equivale a duas rotações simples.
- Existem 2 rotações simples e 2 rotações duplas;
  - Rotação simples à direita e rotação simples à esquerda.
  - Rotação dupla à direita e rotação dupla a esquerda

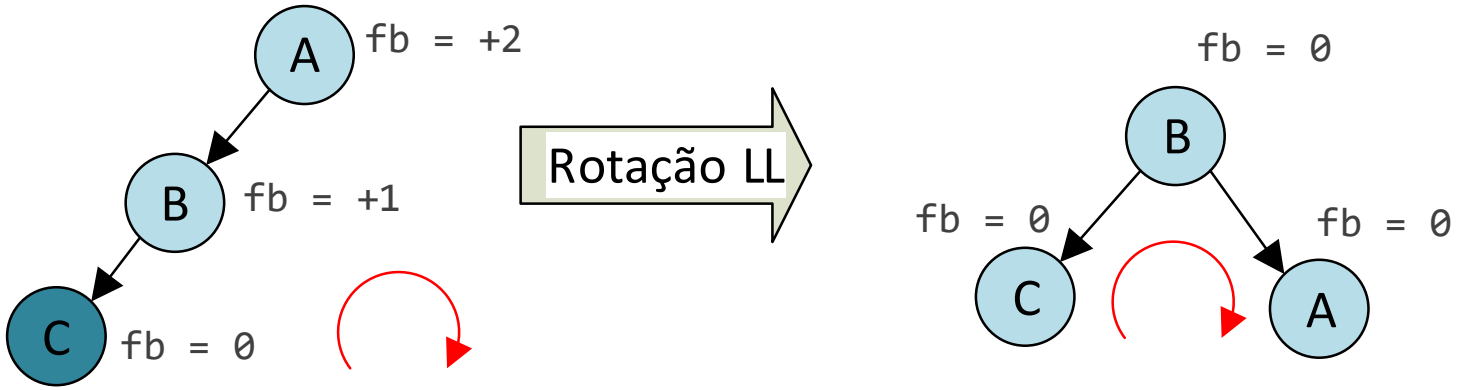
árvore AVL - Implementação

- Rotação RR ou rotação simples a esquerda:
  - O nó “C” é inserido na sub-árvore direita da sub-árvore direita de “A”.
  - O nó intermediário “B”, deve ser escolhido para ser a raiz da árvore resultante.



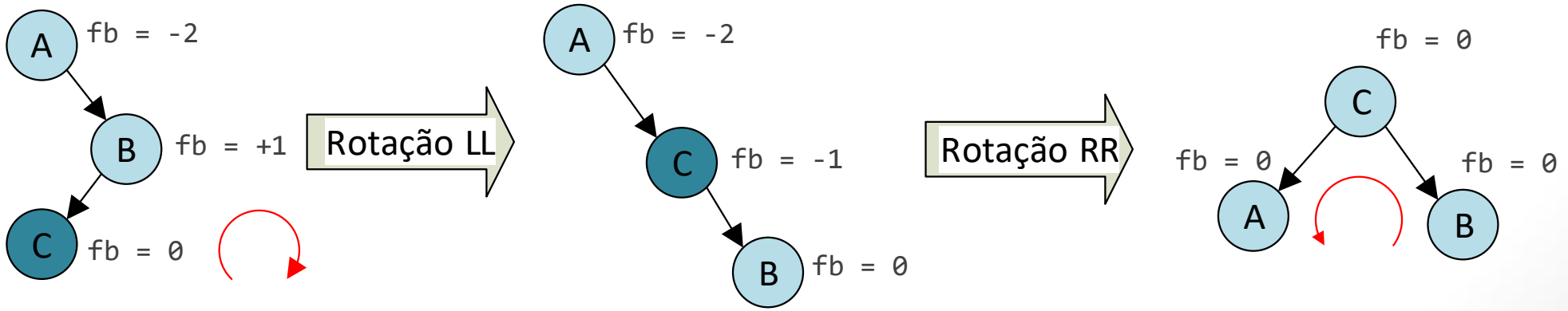
árvore AVL - Implementação

- Rotação LL ou rotação simples á direita:
  - O nó “C” é inserido na sub-árvore da esquerda da sub-árvore da esquerda de “A”;
  - O nó intermediário “B” deve ser escolhido para ser a raiz da árvore resultante.



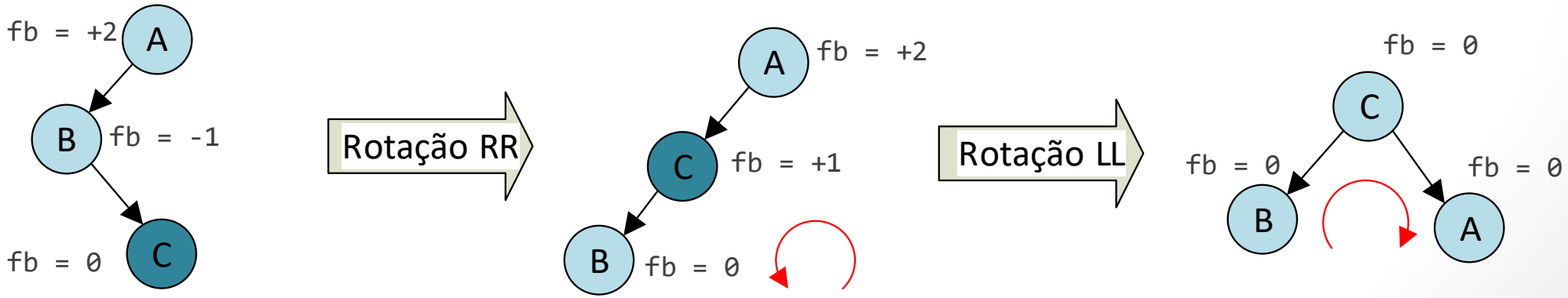
árvore AVL - Implementação

- Rotação RL ou Rotação Dupla a Esquerda:
  - O nó “C” é inserido na sub-árvore da esquerda da sub-árvore da direita de “A”;
  - O nó “C” deve ser escolhido para ser a raiz da árvore resultante.



árvore AVL - Implementação

- Rotação LR ou Rotação Dupla a Direita:
  - O nó “C” é inserido na sub-árvore da direita da sub-árvore da esquerda de “A”;
  - O nó “C” deve ser escolhido para ser a raiz da árvore resultante.



árvore AVL - Implementação

- Quando usar cada rotação?
  - Considerando que o nó “C” foi inserido como filho do nó “B”, e que o nó “B” é filho do nó “A”, se o fator de balanceamento for:

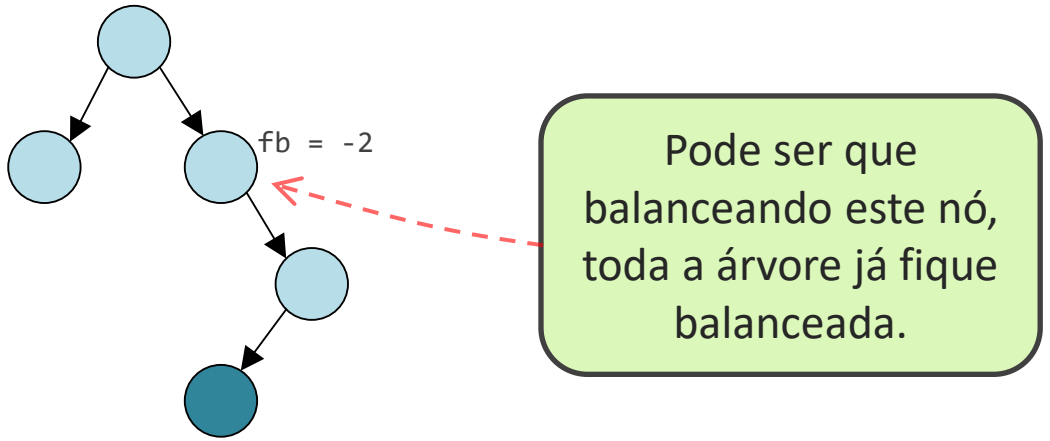
```
A = +2 e B = + 1: Rotação LL;
A = -2 e B = - 1: Rotação RR;
A = +2 e B = - 1: Rotação LR;
A = -2 e B = + 1: Rotação RL;
```

Note que para as rotações LR e RL os sinais de “A” e “B” são invertidos.

- As rotações LL e RR são simétricas entre si, assim como LR e RL também o são.

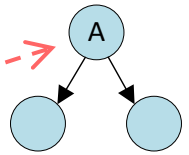
árvore AVL - Implementação

- Implementando as Rotações:
  - As rotações são aplicadas no nó ***ancestral*** mais próximo do nó inserido cujo fator de Balanceamento passa a ser +2 ou -2;
  - Esse é o parâmetro das funções implementadas;
  - As rotações simples, LL e RR, atualizam as novas alturas das sub-árvores;
  - As rotações duplas, LR e RL, podem ser implementadas com duas rotações simples.



árvore AVL - Implementação

- Rotação simples à direita:




```
void rotacaoLL(arvAVL *A){
     struct NO *B;
     B = (*A)->esq;
     (*A)->esq = B->dir;
     B->dir = *A;
     (*A)->alt = maior(alt_no((*A)->esq), alt_no((*A)->dir)) + 1;
     B->alt = maior(alt_no(B->esq), (*A)->alt) + 1;
     *A = B;
}
```

A função `rotaçãoLL()`, recebe o nó "A" que foi desbalanceado, juntamente com seus filhos `esq` e `dir`.

Calcula a nova altura de "A" em relação aos seus atuais filhos. A nova altura passa a ser a maior altura entre os seus dois filhos + 1.

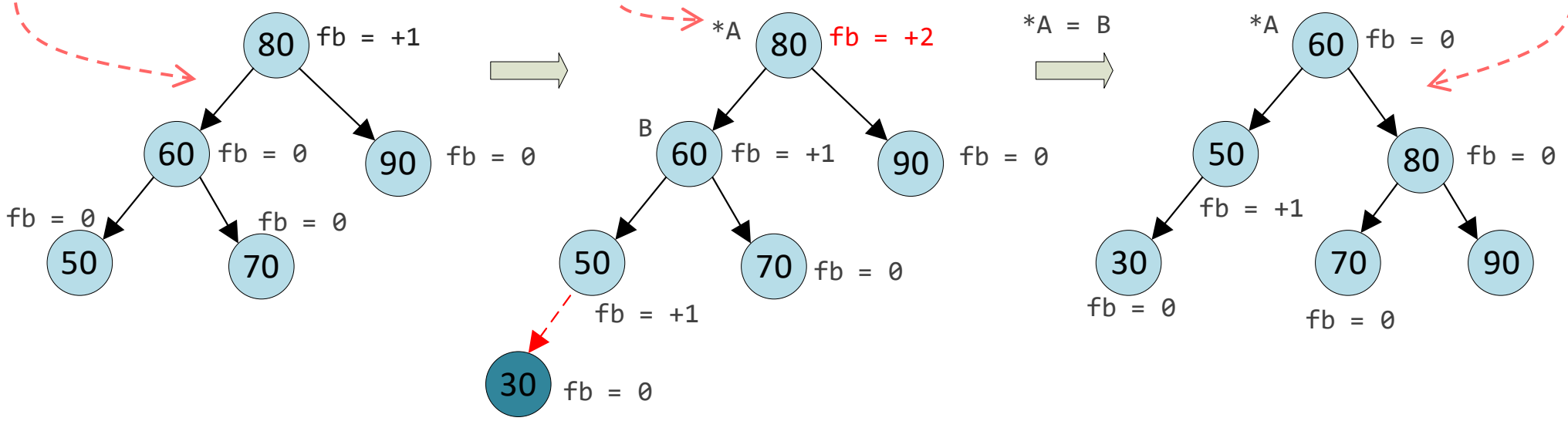
A altura do nó "B", passa a ser o maior valor entre o filho da esquerda e a altura do nó "A", que agora é seu filho da direita, + 1

árvore AVL - Implementação

Árvore AVL e fator de balanceamento de cada nó, antes da inserção que a desbalanceará.

Inserção no nó “30” na árvore. Árvore fica desbalanceada. Aplicar rotação LL no nó “A”

Árvore balanceada, após a aplicação da rotação LL a direita para sua correção.

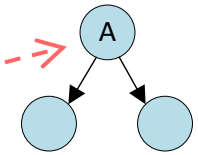


Neste trecho código não estamos levando em conta o cálculo da altura para focarmos somente na rotação.

```
B = (*A)->esq;
(*A)->esq = B>dir;
B>dir = *A;
*A = B;
```

árvore AVL - Implementação

- Rotação simples a esquerda




```
void rotacaoRR(arvAVL *A){
    struct NO *B;
    B = (*A)->dir;
    (*A)->dir = B->esq;
    B->esq = (*A);
    (*A)->alt = maior(alt_no((*A)->esq), alt_no((*A)->dir)) + 1;
    B->alt = maior(alt_no(B->dir), (*A)->alt) + 1;
    (*A) = B;
}
```

A função `rotaçãoLL()`, recebe o nó "A" que foi desbalanceado, juntamente com seus filhos `esq` e `dir`.

Calcula a nova altura de "A" em relação aos seus atuais filhos. A nova altura passa a ser a maior altura entre os seus dois filhos + 1.

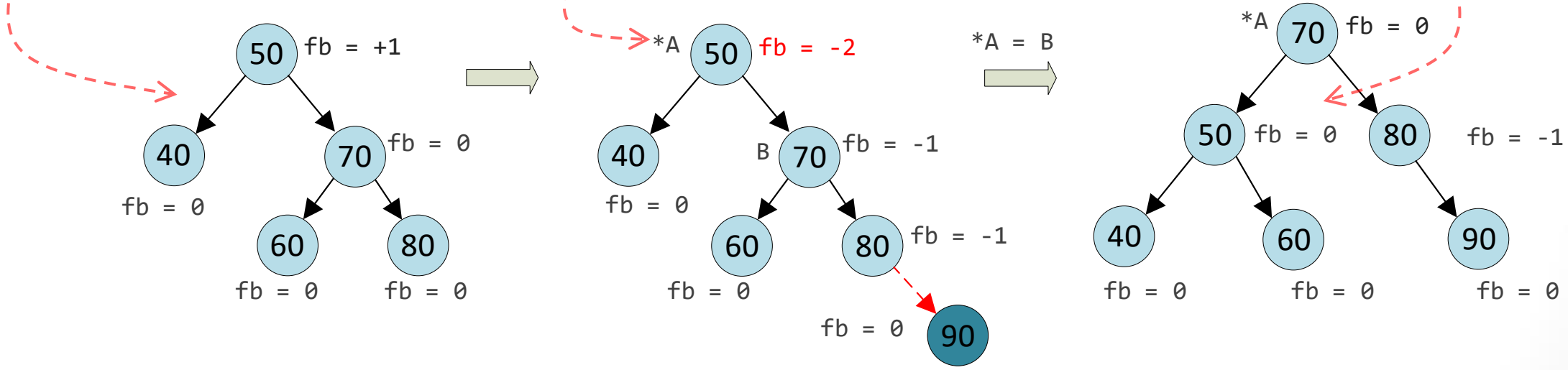
A altura do nó "B", passa a ser o maior valor entre o filho da esquerda e a altura do nó "A", que agora é seu filho da direita, + 1

árvore AVL - Implementação

Árvore AVL e fator de balanceamento de cada nó, antes da inserção que a desbalanceará.

Inserção no nó “90” na árvore. Árvore fica desbalanceada. Aplicar rotação RR no nó “A”

Árvore balanceada, após a aplicação da rotação RR a direita para sua correção.

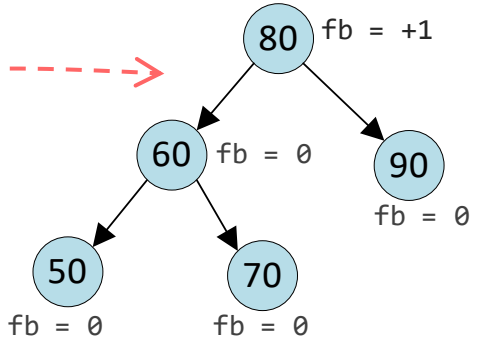


Neste trecho código não estamos levando em conta o cálculo da altura para focarmos somente na rotação.

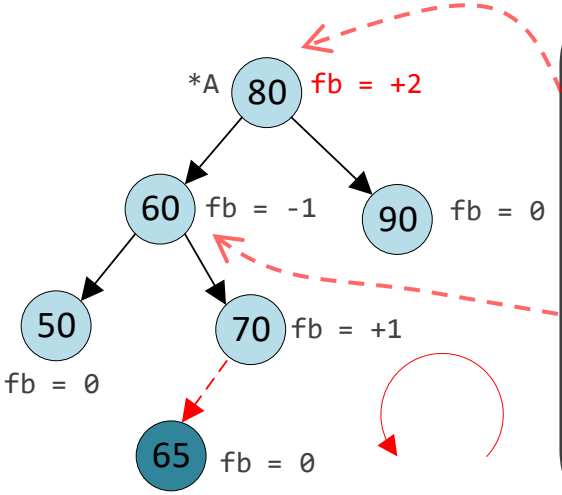
```
B = (*A)->dir;
(*A)->dir = B->esq;
B->esq = *A;
*A = B;
```

Estrutura de Dados 2

```
void rotacaoLR(arvAVL *A){
    rotacaoRR(&(*A)->esq);
    rotacaoLL(A);
}
```

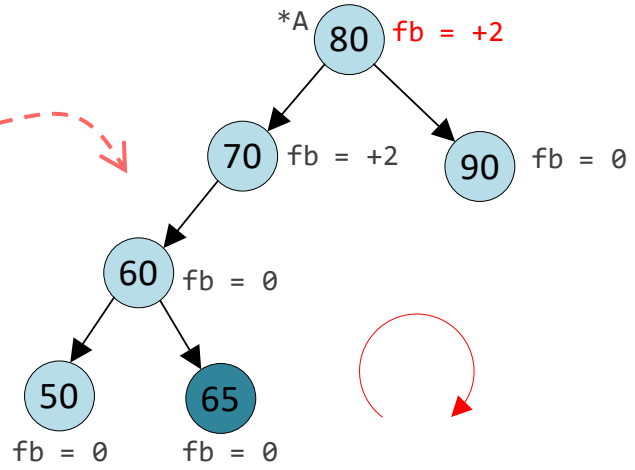


Árvore AVL e fator de balanceamento de cada nó, antes da inserção

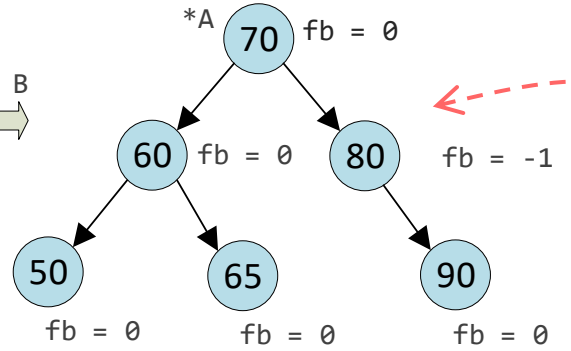


Inserção no nó “65” na árvore. Árvore fica desbalanceada no nó “A”. Aplicar rotação LR no nó “A”. Isso equivale à: Aplicar rotação RR em seu filho esquerdo (“B”), e aplicar rotação LL no nó “A”

Árvore após aplicar rotação RR no filho esquerdo de “A”.



*A = B

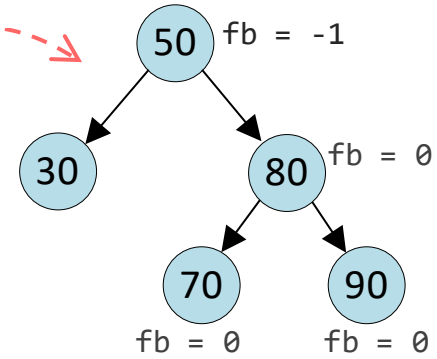


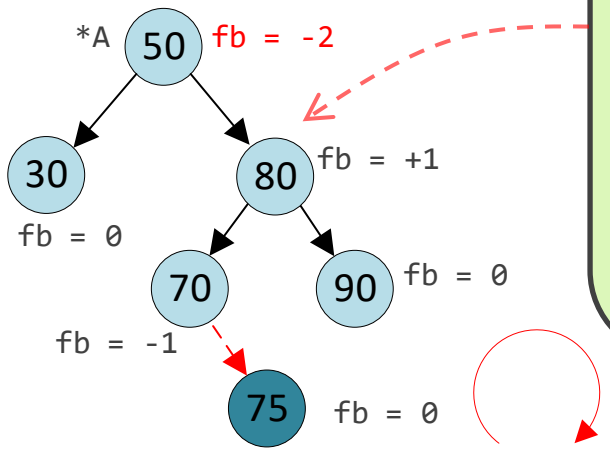
Árvore após aplicar rotação LL no nó “A”: árvore balanceada.

Estrutura de Dados 2

```
void rotacaoRL(arvAVL *A){
    rotacaoLL(&(*A)->dir);
    rotacaoRR(A);
}
```

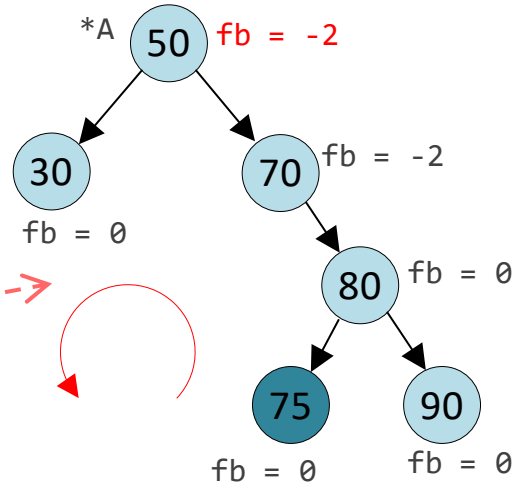
Árvore AVL e fator de balanceamento de cada nó, antes da inserção.



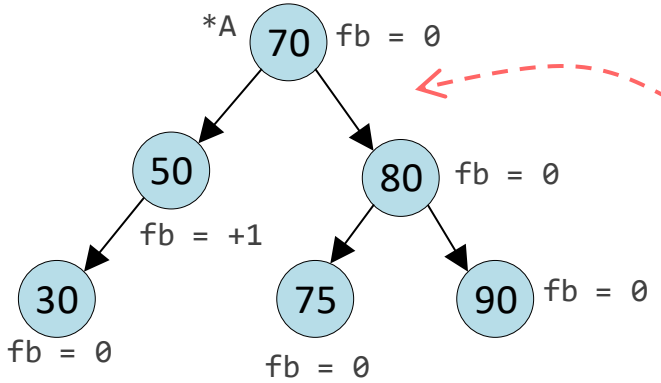


Inserção do nó 75 na árvore. árvore fica desbalanceada no nó "A", aplicar rotação RL no nó "A". Isso equivale à: aplicar rotação LL em 80, e aplicar rotação RR no nó "A".

Árvore após aplicar rotação LL em 80.



*A = B



Árvore após aplicar rotação RR no nó "A": árvore balanceada.

Atividade árvore AVL

- Entregue no Moodle o projeto árvore AVL parcial.