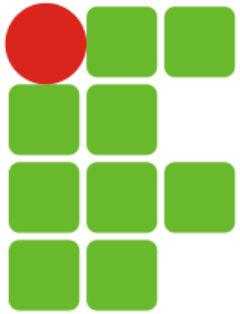


Estrutura de Dados 2

Aula 05 parte 2– Árvores AVL: Inserção e Remoção.

Antonio Angelo de Souza Tartaglia

angelot@ifsp.edu.br

Estrutura de Dados 2

Árvore AVL – Inserção em Árvore AVL

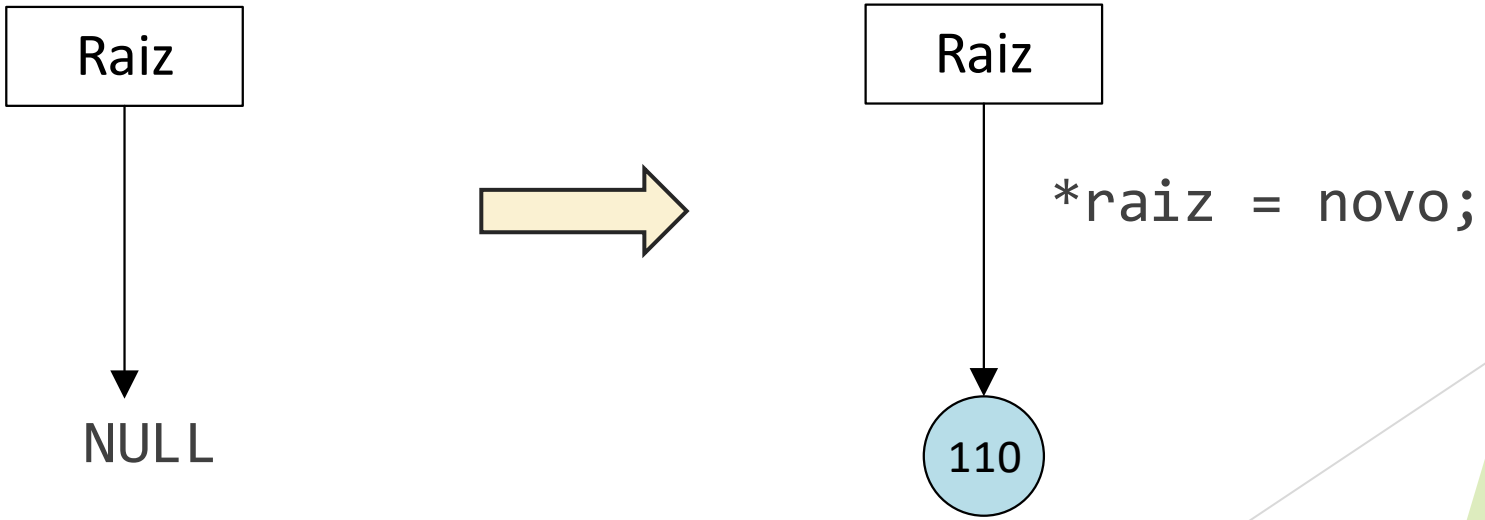
- Para inserir um novo nó em uma Árvore AVL, basicamente o que tem que ser feito é alocar espaço para um novo nó, e procurar sua posição na Árvore usando os passos à seguir:
  - A Raiz é NULL : insira o nó na Raiz;
  - Se a chave do nó é menor do que a da Raiz: vá para sua Sub-Árvore esquerda;
  - Se a chave do nó é maior do que a da Raiz: vá para sua Sub-Árvore direita;
  - Aplique o método recursivamente.

  - **Ao voltar da recursão, recalcule as alturas de cada Sub-Árvore**;
  - Aplique a rotação necessária se o fato de Balanceamento passa a ser +2 ou -2.

Estrutura de Dados 2

Árvore AVL – Inserção em Árvore AVL

- Também existe o caso onde a inserção é feita em uma Árvore AVL que está vazia.



Estrutura de Dados 2

Árvore AVL – Inserção em Árvore AVL

```
//Arquivo arvoreAVL.h
int insere_arvAVL(arvAVL *raiz, int valor);

/*********************************
*      programa principal       *
* para cada inserção, deve-se   *
*   aplicar o tratamento para   *
*    código de erro em "x"      *
*********************************/
x = insere_arvAVL(raiz, 100);
x = insere_arvAVL(raiz, 140);
x = insere_arvAVL(raiz, 160);
x = insere_arvAVL(raiz, 130);
x = insere_arvAVL(raiz, 150);
x = insere_arvAVL(raiz, 110);
x = insere_arvAVL(raiz, 120);
```

A inserção de um novo nó na Árvore AVL é exatamente igual à inserção na Árvore Binária. Porém uma vez inserido, começam a surgir as diferenças entre uma simples Árvore Binária de Busca e uma Árvore AVL.

```
if(x){
    printf("Elemento inserido com sucesso!.");
}else{
    printf("Erro, não foi possível inserir o elemento.");
}
```

Estrutura de Dados 2

Árvore AVL – Inserção em Árvore AVL

```c
//Arquivo arvoreAVL.c
int insere_arvAVL(arvAVL *raiz, int valor){
    int res; // pega resposta das chamadas de função
    if(*raiz == NULL){ //arvore vazia ou nó folha
        struct NO *novo;
        novo = (struct NO*) malloc(sizeof(struct NO));
        if(novo == NULL){
            return 0;
        }
        novo->info = valor;
        novo->alt = 0;
        novo->esq = NULL;
        novo->dir = NULL;
        *raiz = novo;
        return 1;
    }
```

Desci na recursão até chegar ao Nó folha. Este primeiro “if”, só será executado quando a chamada recursiva encontrar a posição onde inserir o novo nó.

Preenche novo Nó e sua altura como 0, este Nó será uma folha.

Estrutura de Dados 2

Árvore AVL – Inserção em Árvore AVL

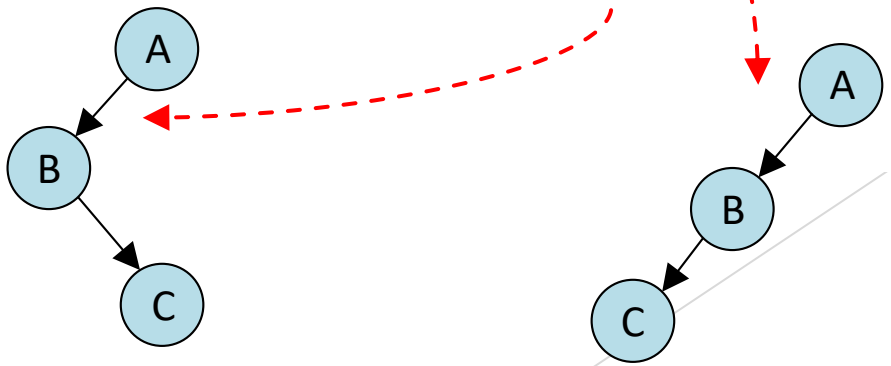
```
}
struct NO *atual = *raiz;
if(valor < atual->info){
    if((res = insere_arvAVL(&(atual->esq), valor)) == 1){
        if(fatorBalanceamento_NO(atual) >= 2){
            if(valor < (*raiz)->esq->info){
                rotacaoLL(raiz);
            }else{
                rotacaoLR(raiz);
            }
        }
    }
}else{
```

Se valor a ser inserido for menor do que campo “info” do Nó atual, a inserção tem que ser feita na esquerda

Aqui é feita a chamada recursiva.

Já pega a resposta da chamada de função (res), se deu certo tem que ser feito o Balanceamento da Árvore, se necessário usa a função fatorBalanceamento_NO() para calcular Balanceamento do Nó atual.

Se o valor é menor do que o conteúdo do filho da esquerda da raiz, que é o atual, então é uma inserção deste tipo.



Estrutura de Dados 2

Árvore AVL – Inserção em Árvore AVL

Se o valor a inserir não era menor do que o atual...

Aqui é feita a chamada recursiva.

Se deu certo, chama o Balanceamento e calcula para o Nó atual.

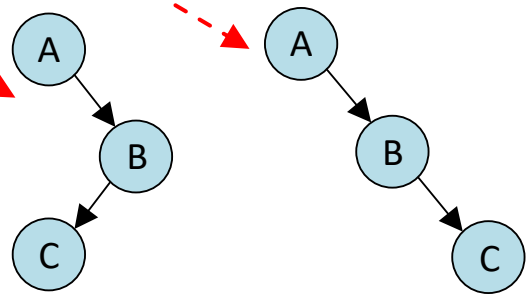
Igualmente ao trecho de código anterior, este if detecta em que direção a inserção ocorreu.

Se o valor a inserir não for menor nem maior, ele é igual. Não temos valores repetidos...

Recalcula a altura do Nó atual: maior valor entre a altura do Nó esquerdo e direito com soma de + 1.

Retorna resultado da Inserção.

```
}else{
    if(valor > atual->info){
        if((res = insere_arvAVL(&(atual->dir), valor)) == 1){
            if(fatorBalanceamento_NO(atual) >= 2){
                if((*raiz)->dir->info < valor){
                    rotacaoRR(raiz);
                }else{
                    rotacaoRL(raiz);
                }
            }
        }
    }else{
        printf("Valor duplicado!\n");
        return 0;
    }
}
atual->alt = maior(alt_no(atual->esq), alt_no(atual->dir)) + 1;
return res;
}
```

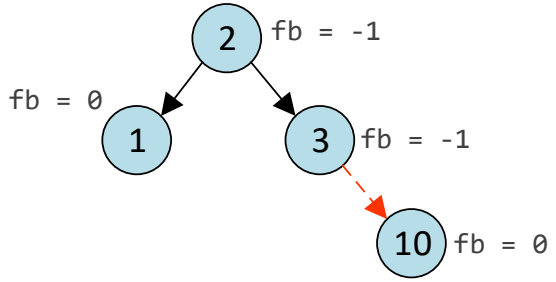


Estrutura de Dados 2

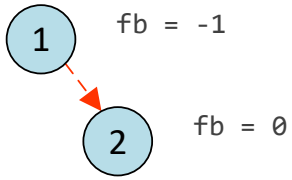
INSTITUTO FEDERAL
SÃO PAULO
Campus Guarulhos

Insere valor 1

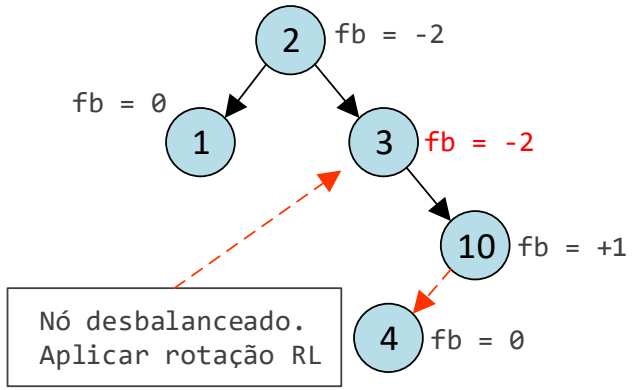
Insere valor 10



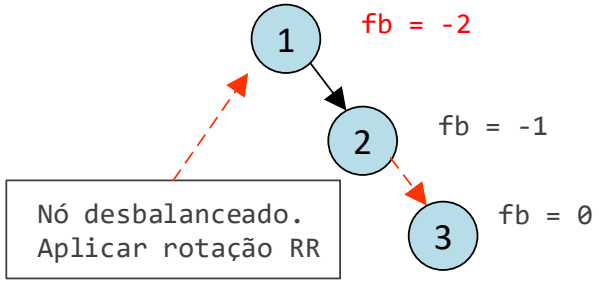
Insere valor 2



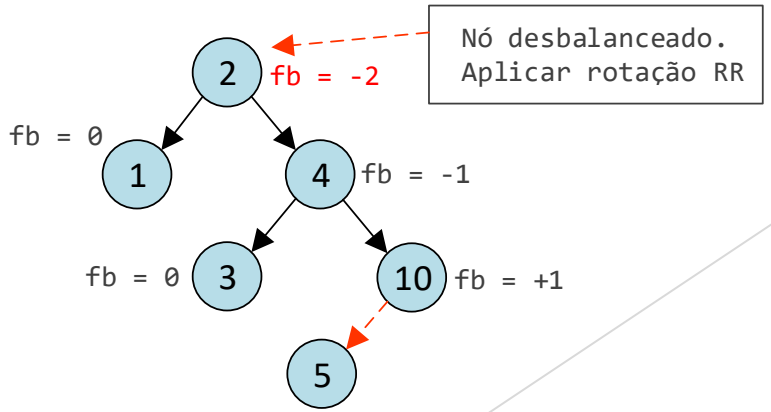
Insere valor 4



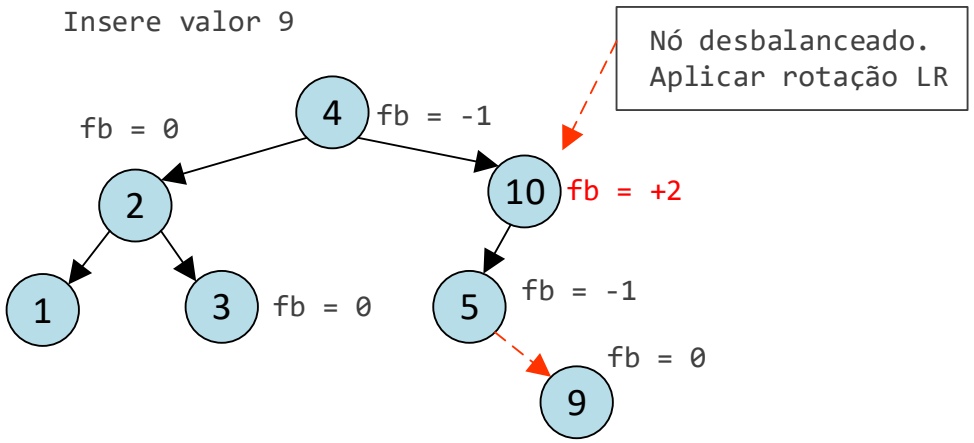
Insere valor 3

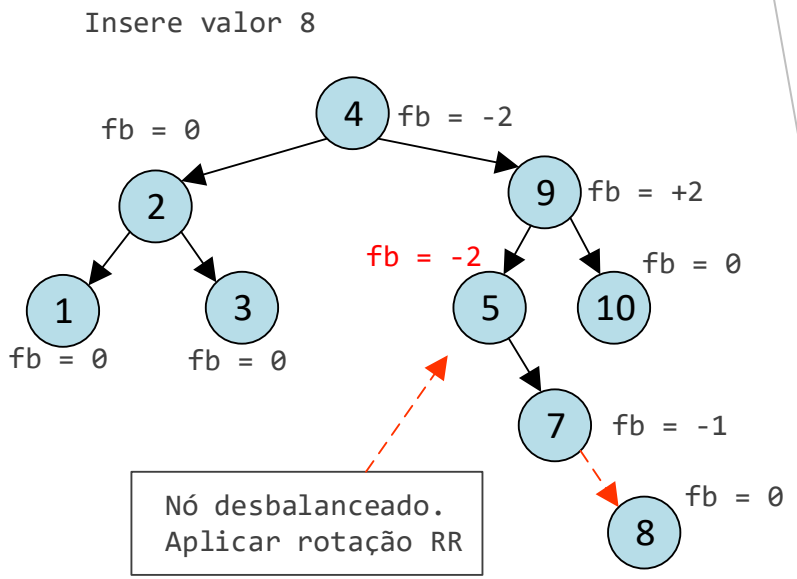


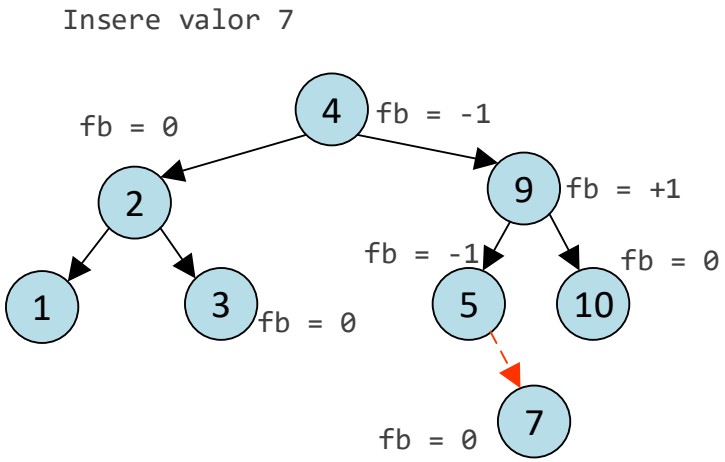
Insere valor 5

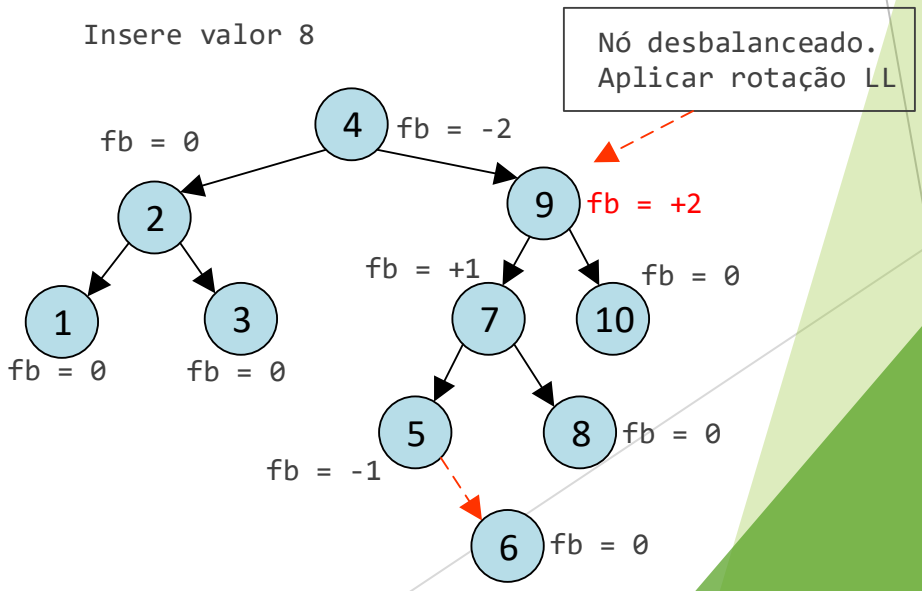


Estrutura de Dados 2



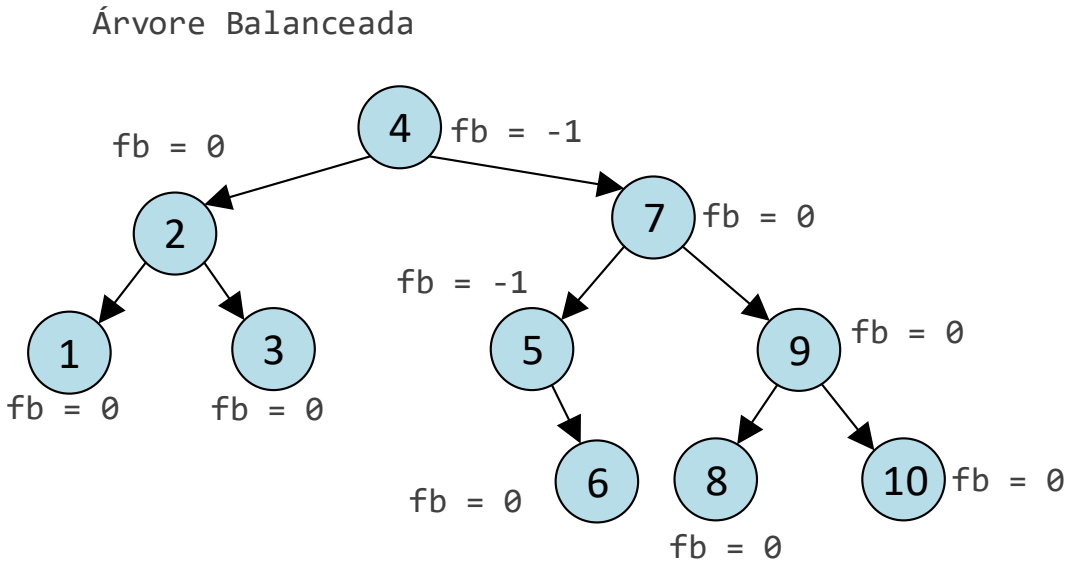






Estrutura de Dados 2

Árvore AVL – Inserção em Árvore AVL



Estrutura de Dados 2

Árvore AVL – Remoção em Árvore AVL

- Existem 3 tipos de remoção:
  - Nó folha, sem filhos;
  - Nó com 1 filho;
  - Nó com 2 filhos.

- Os 3 tipos de remoção trabalham juntos. A remoção sempre remove um elemento específico da Árvore, o qual pode ser Nó folha, ter um ou dois filhos. Somente é possível ter essa informação no momento da remoção.

Estrutura de Dados 2

Árvore AVL – Remoção em Árvore AVL

- Cuidado:
  - Não se pode remover de uma Árvore vazia;
  - Removendo o último Nó, a Árvore fica vazia.

- Balanceamento:
  - Valem as mesmas regras da inserção;
  - Remover um Nó da Sub-Árvore da direita equivale a inserir um Nó na Sub-Árvore da esquerda.

Removido um nó da árvore da direita, balanceia-se a árvore da esquerda, e vice e versa.

Estrutura de Dados 2

Árvore AVL – Remoção em Árvore AVL

```
//Arquivo arvoreAVL.h
int remove_arvAVL(arvAVL *raiz, int valor);

//programa principal
x = remove_arvAVL(raiz, valor);
if(x){
    printf("Elemento removido com sucesso!.");
}else{
    printf("Erro, não foi possível remover o elemento.");
}


//Arquivo arvAVL.c
int remove_arvAVL(arvAVL *raiz, int valor){
    //função responsável pela busca do Nó a ser removido

struct NO *procuramenor(struct NO *atual){
    //função responsável por tratar a remoção de um Nó com 2 filhos
```

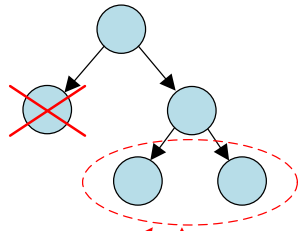
Não é possível simplesmente remover o nó, é necessário substituí-lo por outro.

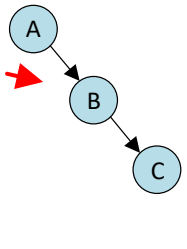
Estrutura de Dados 2

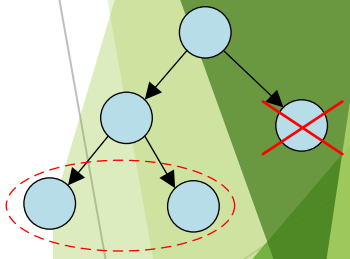
```
//Arquivo arvoreAVL.c
int remove_arvAVL(arvAVL *raiz, int valor){
    if(*raiz == NULL){
        return 0;
    }
    int res;
    if(valor < (*raiz)->info){
        if((res = remove_arvAVL(&(*raiz)->esq, valor)) == 1){
            if(fatorBalanceamento_NO(*raiz) >= 2){
                if(alt_no((*raiz)->dir->esq) <= alt_no((*raiz)->dir->dir)){
                    rotacaoRR(raiz);
                }else{
                    rotacaoRL(raiz);
                }
            }
        }
    }
    if((*raiz)->info < valor){
        if((res = remove_arvAVL(&(*raiz)->dir, valor)) == 1){
            if(fatorBalanceamento_NO(*raiz) >= 2){
                if(alt_no((*raiz)->esq->dir) <= alt_no((*raiz)->esq->esq)){
                    rotacaoLL(raiz);
                }else{
                    rotacaoLR(raiz);
                }
            }
        }
    }
```

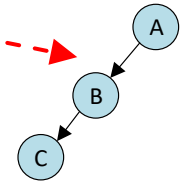
Se a resposta for positiva, removeu um nó, então verifica balanceamento

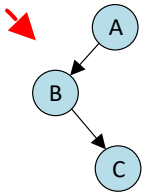
Como foi removido da esquerda, é necessário verificar a Árvore da direita











Estrutura de Dados 2

```
if((*raiz)->info == valor){
    if(((*raiz)->esq == NULL) || (*raiz)->dir == NULL){
        struct NO *no_velho = (*raiz);
        if((*raiz)->esq != NULL){
            *raiz = (*raiz)->esq;
        }else{
            *raiz = (*raiz)->dir;
        }
        free(no_velho);
    }else{
        struct NO *temp = procuramenor((*raiz)->dir);
        (*raiz)->info = temp->info;
        remove_arvAVL( (*raiz)->dir, (*raiz)->info);
        if(fatorBalanceamento_NO(*raiz) >= 2){
            if(alt_no((*raiz)->esq->dir) <= alt_no((*raiz)->esq->esq)){
                rotacaoLL(raiz);
            }else{
                rotacaoLR(raiz);
            }
        }
    }
    if(*raiz != NULL){
        (*raiz)->alt = maior(alt_no((*raiz)->esq), alt_no((*raiz)->dir)) + 1;
    }
    return 1;
}
if(*raiz != NULL){
    (*raiz)->alt = maior(alt_no((*raiz)->esq), alt_no((*raiz)->dir)) + 1;
}
return res;
}
```

Trata realmente a remoção

Determina quantos filhos tem

Pai tem um filho ou nenhum

Verifica qual é o filho que existe

Pai tem 2 filhos, substituir pelo Nó mais a esquerda (menor) da Sub-Árvore da direita

Remove o nó que foi retornado pela função procuramenor(), e balanceia na esquerda

Remove da Sub-Árvore da direita o valor recuperado no passo anterior que está armazenado em “temp”

Terminada a remoção, atualiza a altura do nó e retorna 1 indicando o sucesso na remoção

INSTITUTO FEDERAL
SÃO PAULO
Campus Guarulhos

Estrutura de Dados 2

Árvore AVL – Remoção em Árvore AVL

```
//função auxiliar - procura nó mais a esquerda
struct NO *procuramenor(struct NO *atual){
    struct NO *no1 = atual;
    struct NO *no2 = atual->esq;
    while(no2 != NULL){
        no1 = no2;
        no2 = no2->esq;
    }
    return no1;
}
```

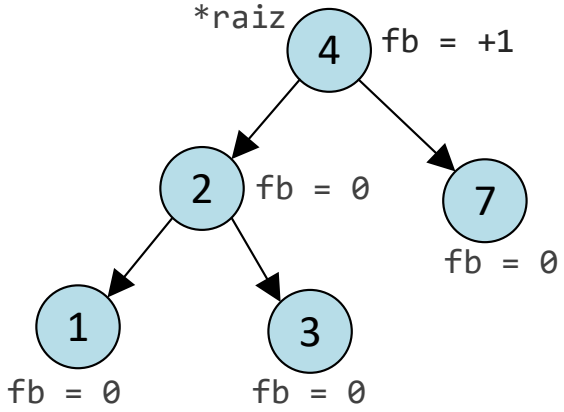
Procura Nó mais a esquerda

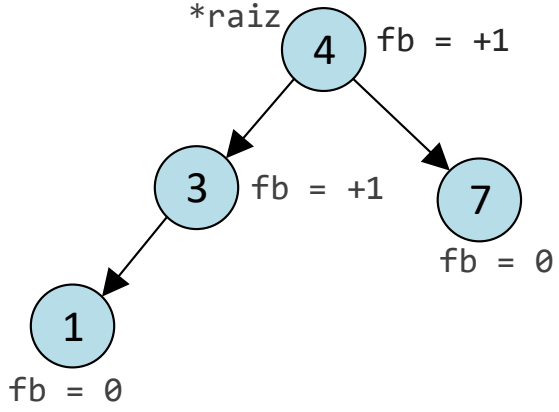
Enquanto for diferente de NULL, “anda” cada vez mais a esquerda. Sempre guardando o último Nó

Estrutura de Dados 2

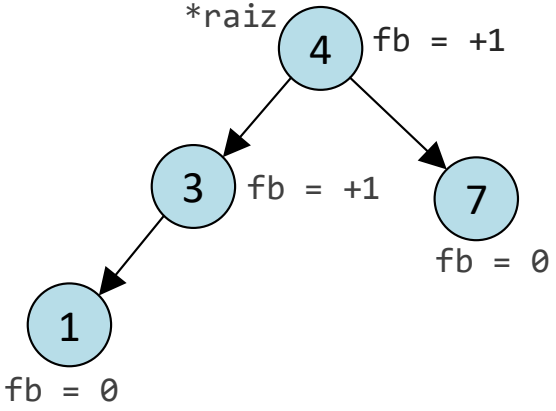
Árvore AVL – Remoção em Árvore AVL

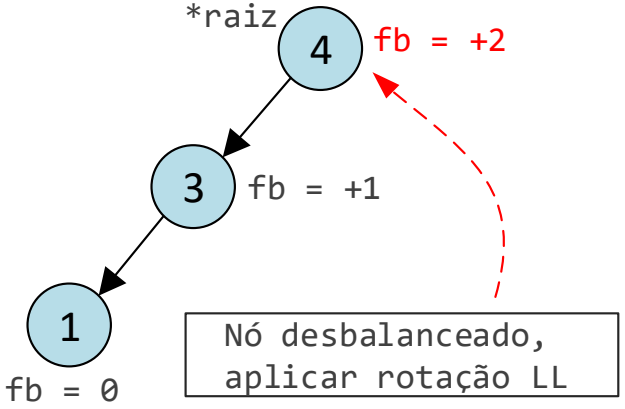
Remove valor 2

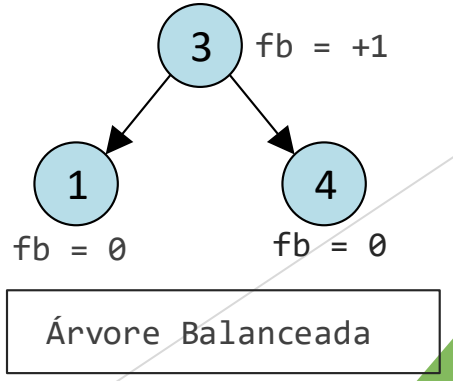




Remove valor 7







Estrutura de Dados 2

Atividade Árvore AVL

- Entregue no Moodle o projeto Árvore AVL final.