Árvores AVL – balanceamento

Prof. Renato Pimentel

2025/2

Sumário

1. Balanceamento de ABB – Árvores AVL
   - Introdução
   - Árvore AVL
   - TAD árvore AVL
   - Rotações
     - Rotação LL (simples à direita)
     - Rotação RR (simples à esquerda)
     - Rotação LR (dupla à direita)
     - Rotação RL (dupla à esquerda)
     - Uso
   - Inserção de nó na AVL
   - Remoção de nó na AVL
   - Exercícios

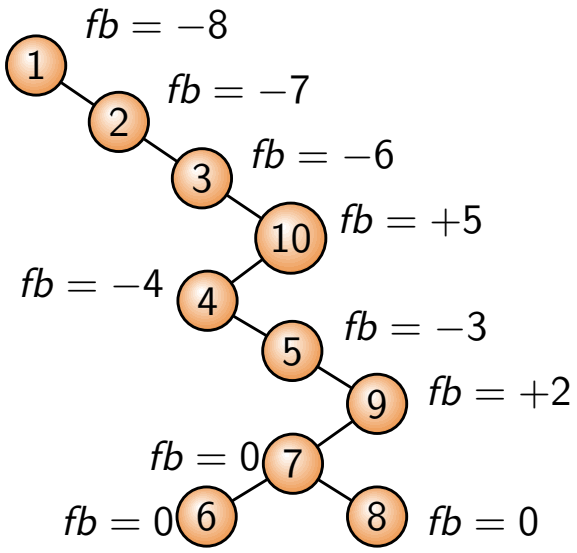
Introdução

- A eficiência da busca em uma árvore binária é proporcional a sua **altura** e depende do seu **balanceamento**.
- Numa árvore com n nós:
  - **Melhor caso**: $O(\log_2 n)$, se a árvore está idealmente balanceada;
  - **Pior caso**: $O(n)$, se a árvore está totalmente desbalanceada.

- Os algoritmos de inserção e remoção em ABB vistos não garantem que a árvore gerada a cada passo esteja balanceada;
- Dependendo da ordem em que os dados são inseridos na árvore, podemos criar uma ABB bastante ou completamente desbalanceada – situação em que os nós internos possuem uma única subárvore.

Exemplo

Inserção dos valores:
$\{1,\ 2,\ 3,\ 10,\ 4,\ 5,\ 9,\ 7,\ 8,\ 6\}$



Solução para o problema do balanceamento

- Modificar os algoritmos de inserção e remoção de modo a balancear a árvore a cada operação.
- Garantir que a **diferença de altura** das subárvores da esquerda e da direita de cada nó seja de no máximo uma unidade – **árvores quase completas**.
- Exemplos de implementação de árvores balanceadas:
  - Árvore AVL;
  - Árvore 2-3-4;
  - Árvore Rubro-Negra.

Sumário

1. Balanceamento de ABB – Árvores AVL
   - Introdução
   - Árvore AVL
   - TAD árvore AVL
   - Rotações
     - Rotação LL (simples à direita)
     - Rotação RR (simples à esquerda)
     - Rotação LR (dupla à direita)
     - Rotação RL (dupla à esquerda)
     - Uso
   - Inserção de nó na AVL
   - Remoção de nó na AVL
   - Exercícios

Árvore AVL

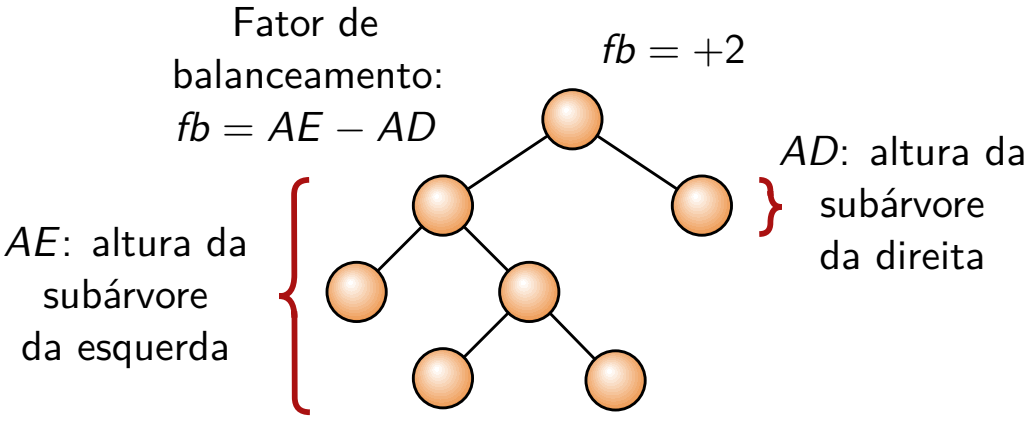
- Tipo de ABB balanceada com relação à altura das suas subárvores;
- Criada pelos matemáticos soviéticos Adelson-Velsky (1922-2014) e Evgenii Landis (1921-1997), de onde recebeu a sua nomenclatura (**A**delson-**V**elsky **L**andis), em 1962.
  - https://zhjwpku.com/assets/pdf/AED2-10-avl-paper.pdf

Aspectos

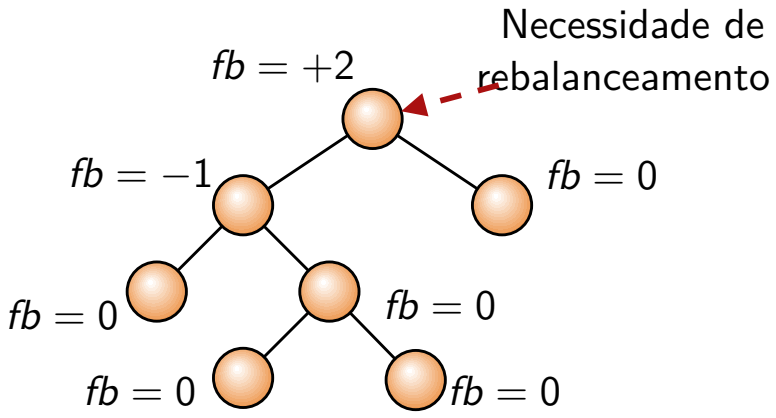
- Permite o rebalanceamento **local** da árvore – apenas a parte afetada pela inserção ou remoção é rebalanceada;
- Usa **rotações simples** ou **duplas** na etapa de rebalanceamento, executadas a cada inserção ou remoção;
- As rotações buscam manter a árvore como uma ABB quase completa;
  -
  - Custo máximo de qualquer algoritmo é $O(log_2 n)$

Fator de balanceamento

- Objetivo das rotações: corrigir o **fator de balanceamento** (fb).
  - fb = Diferença entre as alturas das subárvores de um nó.
- Caso uma das subárvores não exista, então a altura dessa será igual a -1.

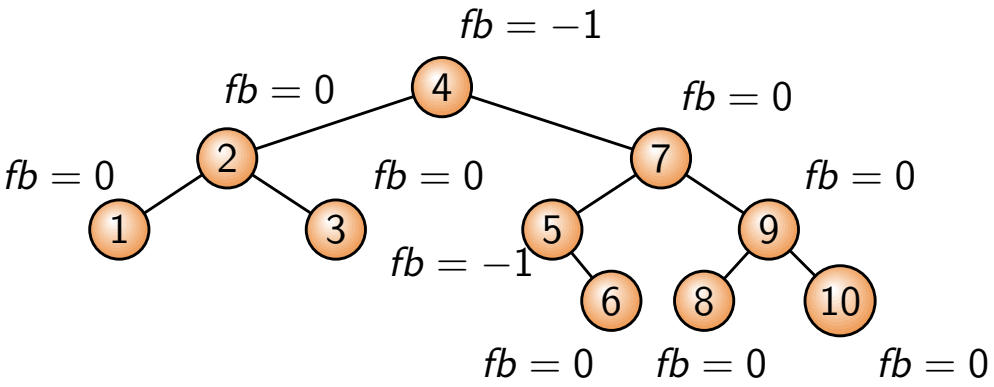


- Espera-se que as alturas das subárvores de cada nó difiram de no máximo uma unidade:
  - O fator de balanceamento deve ser $+1$, 0 ou -1.
- Se $fb > 1$ ou $fb < -1$: a árvore deve ser balanceada naquele nó.



Exemplo anterior – após balanceamento

Inserção dos valores:
$\{1,\ 2,\ 3,\ 10,\ 4,\ 5,\ 9,\ 7,\ 8,\ 6\}$



Sumário

1. Balanceamento de ABB – Árvores AVL
   - Introdução
   - Árvore AVL
   - TAD árvore AVL
   - Rotações
     - Rotação LL (simples à direita)
     - Rotação RR (simples à esquerda)
     - Rotação LR (dupla à direita)
     - Rotação RL (dupla à esquerda)
     - Uso
   - Inserção de nó na AVL
   - Remoção de nó na AVL
   - Exercícios

TAD árvore AVL

- Consideramos um tipo abstrato de dados **opaco** em Linguagem C para representação a árvore AVL, baseado na implementação vista para ABB.
- As considerações feitas com relação a tipo opaco, alocação dinâmica e acesso à estrutura a partir de sua raiz se mantém.

- Modularização do TAD:
  - **ArvoreAVL.h**: a **interface** em si: definição do tipo opaco; funções disponíveis para manipular a árvore AVL (protótipos).
  - **ArvoreAVL.c**: o que deve ficar oculto do usuário, além da implementação das funções definidas pela interface. Inclui a definição do **tipo nó**.
- **Tipo nó**: estrutura que contém:
  - `info`: Representa a **informação**, ou seja, os dados correspondentes ao elemento (nó/vértice) em questão;
  - `altura`: altura da subárvore cujo nó é sua raiz.
  - `*esq` e `*dir`: ponteiros para os nós filhos à esquerda e à direita do nó.

Interface: ArvoreAVL.h

ArvoreAVL.h

```
typedef struct NO* ArvAVL;

ArvAVL* cria_ArvAVL();
void libera_ArvAVL(ArvAVL *raiz);
int insere_ArvAVL(ArvAVL *raiz, int data);
int remove_ArvAVL(ArvAVL *raiz, int valor);
int estaVazia_ArvAVL(ArvAVL *raiz);
int altura_ArvAVL(ArvAVL *raiz);
int totalNO_ArvAVL(ArvAVL *raiz);
int consulta_ArvAVL(ArvAVL *raiz, int valor);
void preOrdem_ArvAVL(ArvAVL *raiz);
void emOrdem_ArvAVL(ArvAVL *raiz);
void posOrdem_ArvAVL(ArvAVL *raiz);
```

Definição do nó: ArvoreAVL.c

ArvoreAVL.c: bibliotecas usadas e definição do tipo

```c
#include <stdio.h>
#include <stdlib.h>
#include "ArvoreAVL.h" //inclui os Protótipos

struct NO
{
  int info;
  int altura;
  struct NO *esq;
  struct NO *dir;
};
```


Cálculo da altura e *fb* I

- As funções abaixo são auxiliares, e não estão visíveis ao usuário do TAD – portanto, não são listadas em ArvoreAVL.h;
- A função que calcula o *fb* retorna seu módulo, mas isto não interfere nas operações de inserção e remoção a serem mostradas adiante.
- A função `maior()` será usada para o comparativo das alturas das subárvores da esquerda e da direita de um nó.

ArvoreAVL.c

```c
int altura_NO(struct NO* no)
{
  if(no == NULL)
    return -1;
  else
    return no->altura;
}

```

Cálculo da altura e *fb* II

```
int fatorBalanceamento_NO(struct NO* no)
{
  return labs(altura_NO(no->esq) - altura_NO(no->dir));
}

int maior(int x, int y)
{
  if(x > y)
    return x;
  else
    return y;
}
```


Sumário

1. Balanceamento de ABB – Árvores AVL
   - Introdução
   - Árvore AVL
   - TAD árvore AVL
   - Rotações
     - Rotação LL (simples à direita)
     - Rotação RR (simples à esquerda)
     - Rotação LR (dupla à direita)
     - Rotação RL (dupla à esquerda)
     - Uso
   - Inserção de nó na AVL
   - Remoção de nó na AVL
   - Exercícios

Objetivo

Corrigir o fator de balanceamento (*fb*) de cada nó:
Operação básica para balancear uma árvore AVL.

- Existem dois tipos de rotação:
  - Rotação simples;
  - Rotação dupla.

- As rotações diferem entre si pelo sentido da inclinação entre o nó pai e filho.
- **Rotação simples**: o nó desbalanceado (pai), seu filho e seu neto estão todos no mesmo sentido de inclinação.
- **Rotação dupla**: o nó desbalanceado (pai) e seu filho estão inclinados em sentido inverso ao de seu neto
  - Equivale a duas rotações simples.

Tipos de rotações

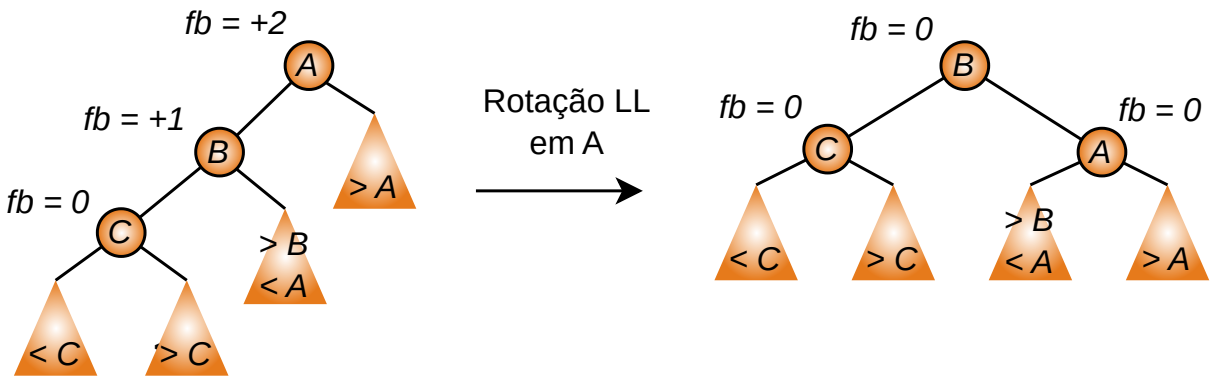
- Rotação **simples à direita** ou **rotação LL**;
- Rotação **simples à esquerda** ou **rotação RR**;
- Rotação **dupla à direita** ou **rotação LR**;
- Rotação **dupla à esquerda** ou **Rotação RL**.

- Rotações são aplicadas no ancestral mais próximo do nó inserido cujo fator de balanceamento passa a ser $+2$ ou -2;
  - Após uma inserção ou remoção, deve-se voltar pelo caminho feito e recalcular o *fb* de cada nó;
  - Para o nó onde $|fb| = 2$, uma rotação deverá ser aplicada.

Rotação LL (simples à direita)

- Quando um novo nó é inserido na subárvore da esquerda (*left*) do filho da esquerda (*left*), B, de A – daí o LL.
  - A é o nó desbalanceado.
- É necessário fazer uma **rotação à direita**, de modo que o nó intermediário B ocupe o lugar de A, e A se torne filho da direita de B.

Exemplo

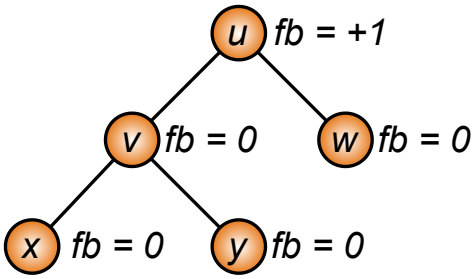


No TAD árvore AVL

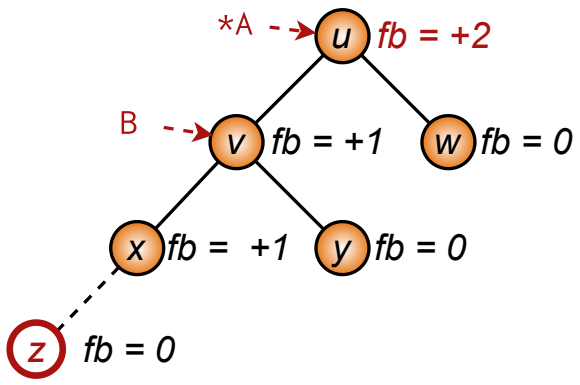
ArvoreAVL.c

```
153 void RotacaoLL(ArvAVL *A)
154 {
155   struct NO *B;
156   B = (*A)->esq;
157   (*A)->esq = B->dir;
158   B->dir = *A;
159   (*A)->altura = maior(altura_NO((*A)->esq),altura_NO
      ((*A)->dir)) + 1;
160   B->altura = maior(altura_NO(B->esq),(*A)->altura) +
      1;
161   *A = B;
162 }
```


Passo a passo

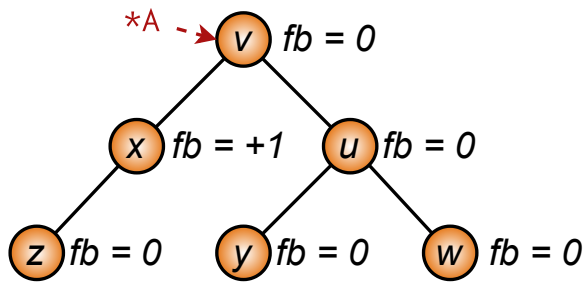


Árvore AVL e fator de balanceamento de cada nó.



Inserção de nó na árvore:
Árvore fica desbalanceada em A.
Aplica-se Rotação LL em A.

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

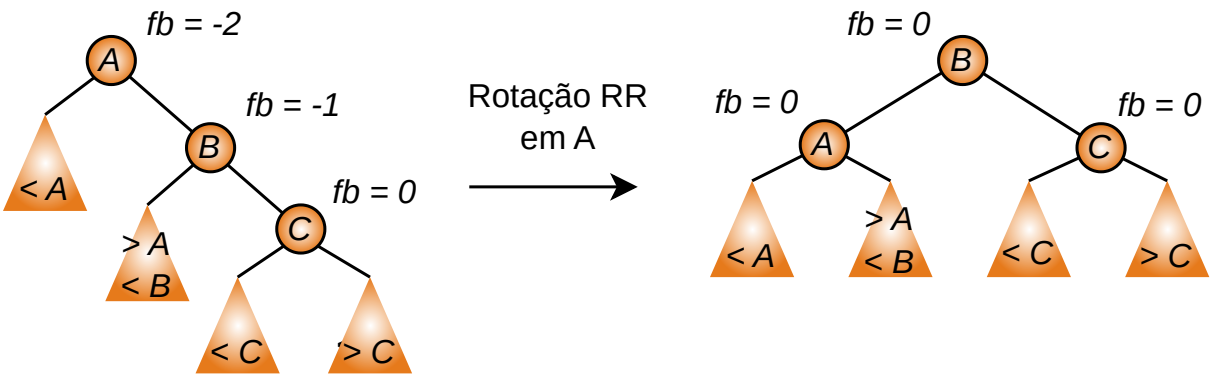



Árvore balanceada.

Rotação RR (simples à esquerda)

- Quando um novo nó é inserido na subárvore da direita (*right*) do filho da direita (*right*), B, de A – daí o RR.
  - A é o nó desbalanceado.
- É necessário fazer uma **rotação à esquerda**, de modo que o nó intermediário B ocupe o lugar de A, e A se torne filho da esquerda de B.

Exemplo

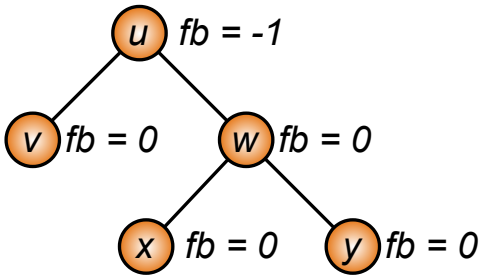


No TAD árvore AVL

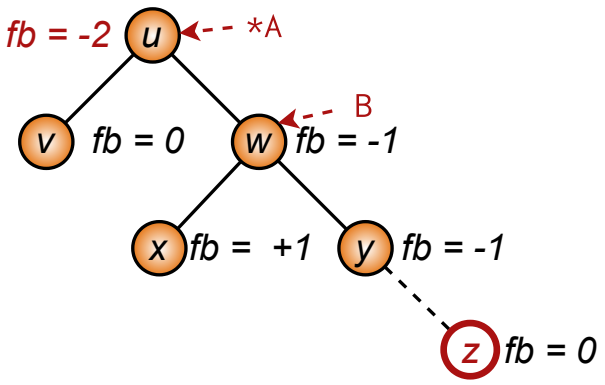
ArvoreAVL.c

```
164 void RotacaoRR(ArvAVL *A)
165 {
166   struct NO *B;
167   B = (*A)->dir;
168   (*A)->dir = B->esq;
169   B->esq = (*A);
170   (*A)->altura = maior(altura_NO((*A)->esq),altura_NO
      ((*A)->dir)) + 1;
171   B->altura = maior(altura_NO(B->dir),(*A)->altura) +
      1;
172   (*A) = B;
173 }
```


Passo a passo

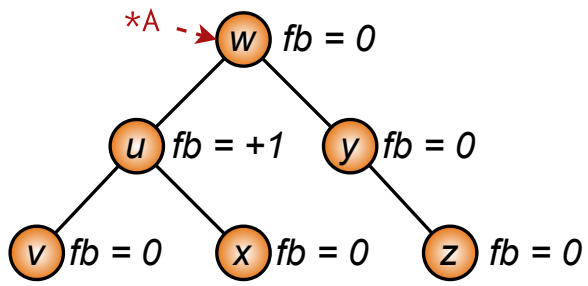


Árvore AVL e fator de balanceamento de cada nó.



Inserção de nó na árvore:
Árvore fica desbalanceada em A.
Aplica-se rotação RR em A.

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

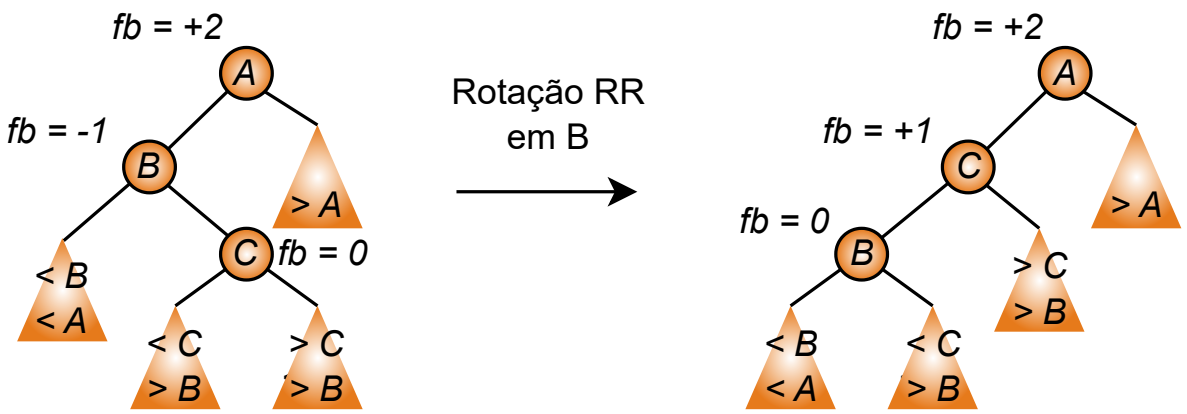



Árvore balanceada.

Rotação LR (dupla à direita)

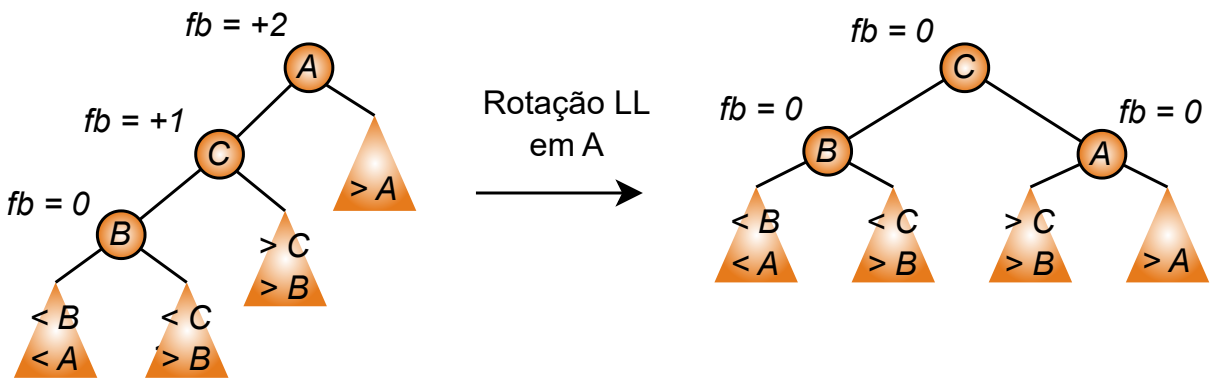
- Quando um novo nó é inserido na subárvore da direita (*right*) do filho da esquerda (*left*), B, de A – daí o LR.
  - A é o nó desbalanceado.
- É necessário fazer uma rotação dupla, de modo que o nó C – raiz da subárvore da direita de B – se torne o pai dos nós A (à sua direita) e B (à sua esquerda).
- Equivalente a duas rotações simples:
  1. Rotação RR em B;
  2. Rotação LL em A.

Exemplo: primeira rotação




```
RotacaoRR(&(*A)->esq);
RotacaoLL(A);
```

Exemplo: segunda rotação



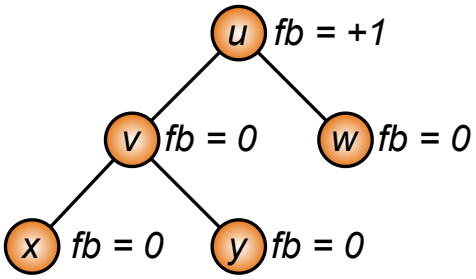

```
RotacaoRR(&(*A)->esq);
RotacaoLL(A);
```


No TAD árvore AVL

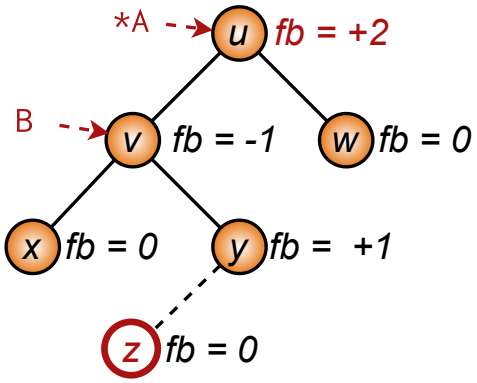
ArvoreAVL.c

```
175 void RotacaoLR(ArvAVL *A)
176 {
177   RotacaoRR(&(*A)->esq);
178   RotacaoLL(A);
179 }
```

Passo a passo

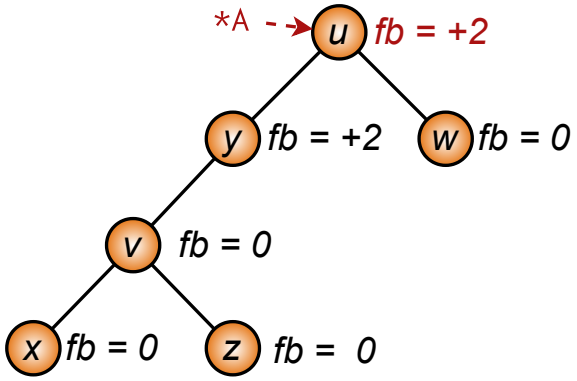


Árvore AVL e fator de balanceamento de cada nó.

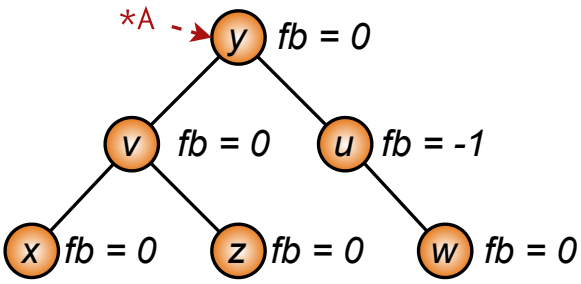


Inserção de nó na árvore:
Árvore fica desbalanceada em A.
Aplica-se Rotação LR em A. Isto equivale a:

- Aplicar Rotação RR em B;
- Aplicar Rotação LL em A.



Árvore após Rotação RR em B.

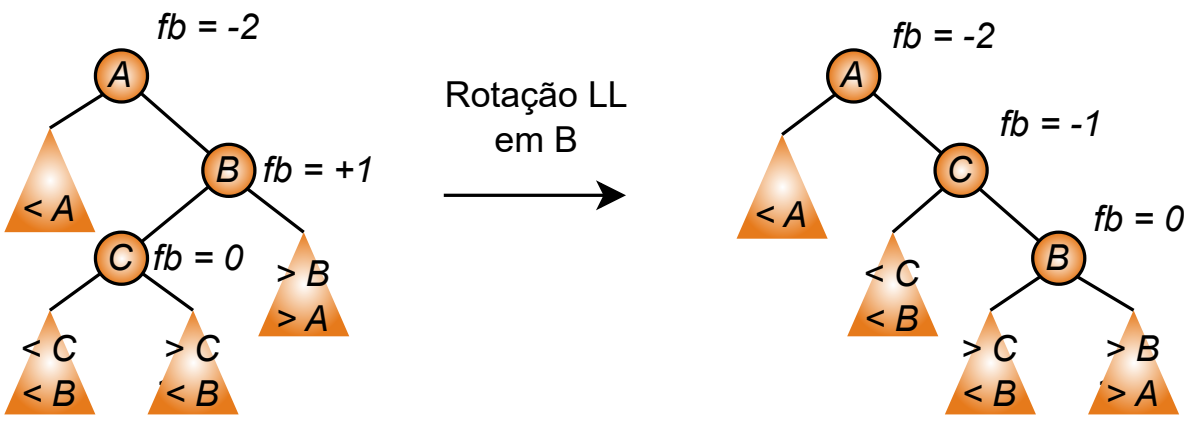


Árvore balanceada.

Rotação RL (dupla à esquerda)

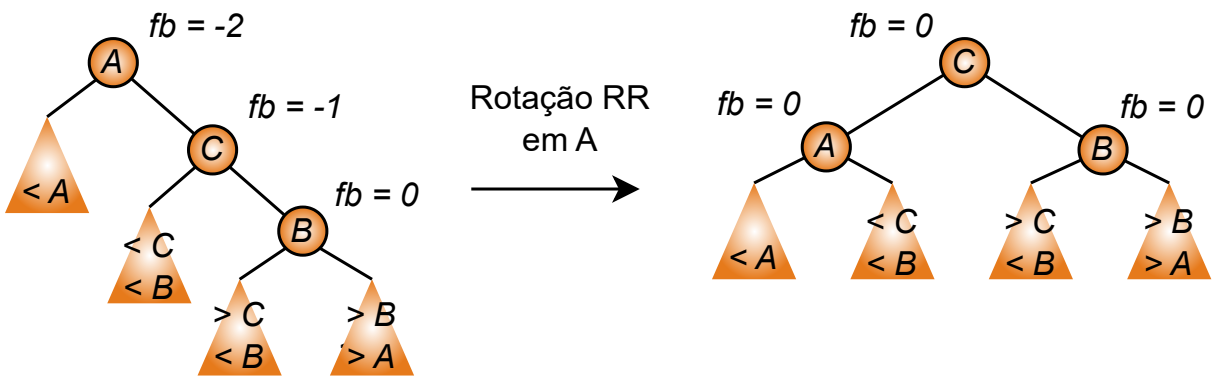
- Quando um novo nó é inserido na subárvore da esquerda (*left*) do filho da direita (*right*), B, de A – daí o RL.
  - A é o nó desbalanceado.
- É necessário fazer uma rotação dupla, de modo que o nó C – raiz da subárvore da esquerda de B – se torne o pai dos nós A (à sua esquerda) e B (à sua direita).
- Equivalente a duas rotações simples:
  1. Rotação LL em B;
  2. Rotação RR em A.

Exemplo: primeira rotação




```
RotacaoLL(&(*A)->dir);
RotacaoRR(A);
```

Exemplo: segunda rotação



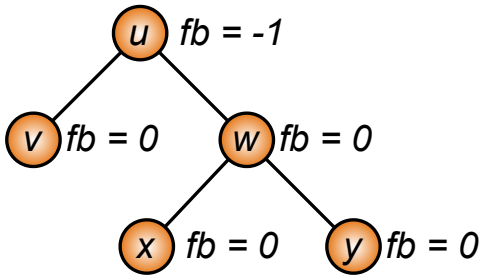

```
RotacaoLL(&(*A)->dir);
RotacaoRR(A);
```


No TAD árvore AVL

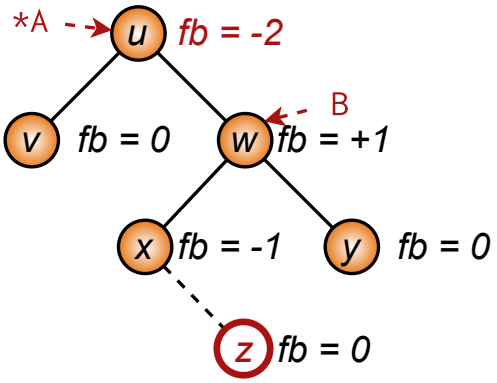
ArvoreAVL.c

```
181 void RotacaoRL(ArvAVL *A)
182 {
183   RotacaoLL(&(*A)->dir);
184   RotacaoRR(A);
185 }
```

Passo a passo

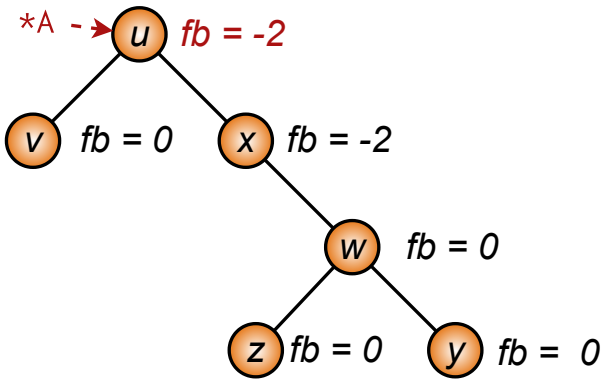


Árvore AVL e fator de balanceamento de cada nó.

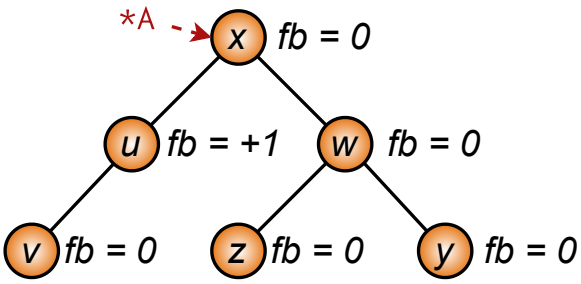


Inserção de nó na árvore:
Árvore fica desbalanceada em A.
Aplica-se Rotação RL em A. Isto equivale a:

- Aplicar Rotação LL em B;
- Aplicar Rotação RR em A.



Árvore após Rotação LL em B.



Árvore balanceada.

Quando usar cada tipo de rotação?

fb de *A*	*fb* de *B*	Posições de *B* e *C* em relação a *A*	Rotação
2	1	*B* é filho à esquerda de *A*; *C* é filho à esquerda de *B*.	LL
-2	-1	*B* é filho à direita de *A*; *C* é filho à direita de *B*.	RR
2	-1	*B* é filho à esquerda de *A*; *C* é filho à direita de *B*.	LR
-2	1	*B* é filho à de direita *A*; *C* é filho à esquerda de *B*.	RL

- **Sinais iguais**: rotações simples – $+$: à direita; $-$: à esquerda;
- **Sinais diferentes**: rotações duplas – $fb(A) > 0$: à direita (LR); $fb(A) < 0$: à esquerda (RL).

Sumário

1 Balanceamento de ABB – Árvores AVL
- Introdução
- Árvore AVL
- TAD árvore AVL
- Rotações
  - Rotação LL (simples à direita)
  - Rotação RR (simples à esquerda)
  - Rotação LR (dupla à direita)
  - Rotação RL (dupla à esquerda)
  - Uso
- Inserção de nó na AVL
- Remoção de nó na AVL
- Exercícios

Inserção de nó na AVL

- Para inserir um valor V na AVL:
  1. Se a raiz é igual a `NULL`, insira o nó;
  2. Se V é **menor que** a raiz: siga para a **subárvore da esquerda**;
  3. Se V é **maior que** a raiz: siga para a **subárvore da direita**;
  4. Aplique o método recursivamente – também é possível sem recursão.
- Percorre-se os nós da AVL até o **nó folha** que se tornará **pai** do novo nó.

- Após cada inserção de um nó na AVL:
  - Volta-se pelo caminho percorrido, recalculando o fator de balanceamento de cada um dos nós visitados;
  - Aplica-se a rotação necessária em um nó onde $|fb| > 1$, restabelecendo o balanceamento da árvore.

main.c: Exemplo de uso

```c
int res,i;
int N = 10, dados[10] = {1,2,3,10,4,5,9,7,8,6};
ArvAVL* avl = cria_ArvAVL();
for(i=0; i<N; i++)
  res = insere_ArvAVL(avl,dados[i]);
```

ArvoreAVL.c: inserção de nó

```c
187 int insere_ArvAVL(ArvAVL *raiz, int valor) {
188   if (raiz == NULL)
189     return 0;
190
191   int res;
192   if (*raiz == NULL) { // árvore vazia ou nó folha
193     struct NO *novo;
194     novo = (struct NO*)malloc(sizeof(struct NO));
195     if (novo == NULL)
196       return 0;
```



```c
197
198     novo->info = valor;
199     novo->altura = 0;
200     novo->esq = NULL;
201     novo->dir = NULL;
202     *raiz = novo;
203     return 1;
204   }
205
206   struct NO *atual = *raiz;
207   if (valor < atual->info){
208     if ((res = insere_ArvAVL(&(atual->esq), valor)) ==
    1) {
209       if (fatorBalanceamento_NO(atual) >= 2) {
210         if (valor < (*raiz)->esq->info)
211           RotacaoLL(raiz); // inserção na subárvore da
    esquerda do filho da esquerda de atual
212         else
213           RotacaoLR(raiz); // inserção na subárvore da
    direita do filho da esquerda de atual
```

```
214        }
215      }
216    } else {
217      if (valor > atual->info) {
218        if ((res = insere_ArvAVL(&(atual->dir), valor))
     == 1){
219          if (fatorBalanceamento_NO(atual) >= 2){
220            if ((*raiz)->dir->info < valor)
221              RotacaoRR(raiz); // inserção na subárvore
     da direita do filho da direita de atual
222            else
223              RotacaoRL(raiz); // inserção na subárvore
     da esquerda do filho da direita de atual
224          }
225        }
226      } else {
227        printf("Valor duplicado!!\n");
228        return 0;
229      }
230    }
```

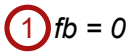


```
231    atual->altura = maior(altura_NO(atual->esq),altura_NO
     (atual->dir)) + 1;
232    return res;
233  }
```

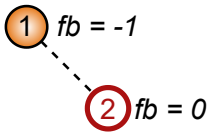
Passo a passo

Inserção dos valores:
$\{1,\ 2,\ 3,\ 10,\ 4,\ 5,\ 9,\ 7,\ 8,\ 6\}$

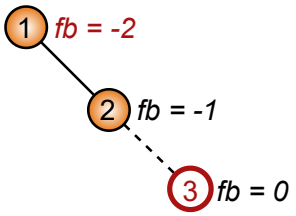
Insere valor: 1



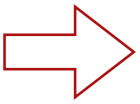
Insere valor: 2

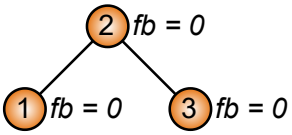


Insere valor: 3.

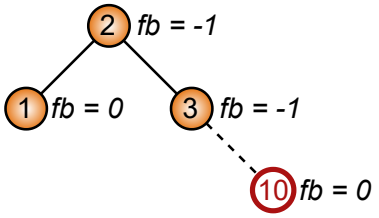


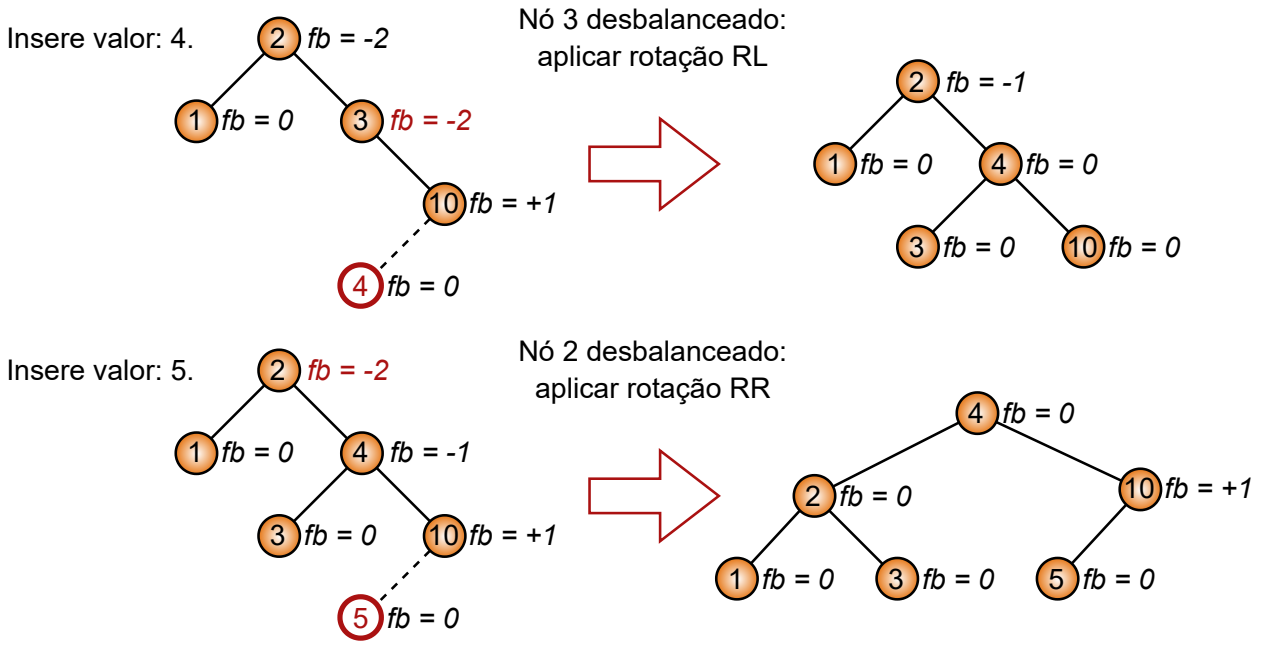
Nó 1 desbalanceado:
aplicar rotação RR

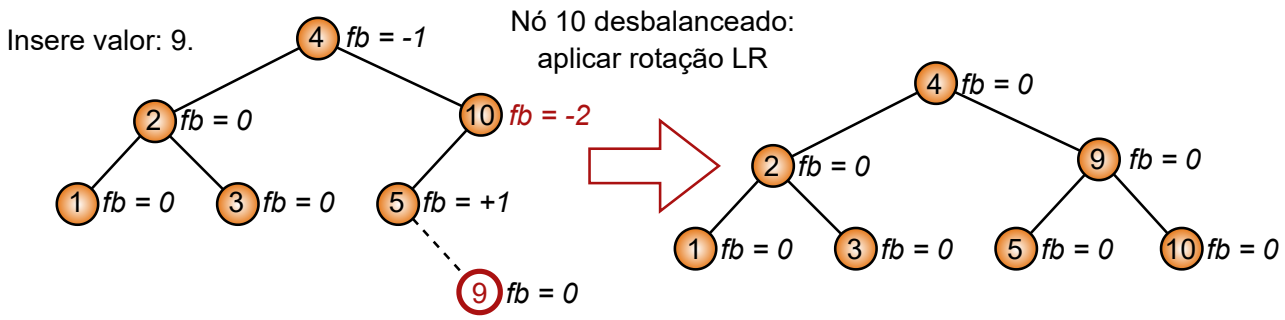




Insere valor: 10.







Exercício: inserir na AVL, mostrando o passo a passo, os valores faltantes: 7, 8 e 6.

Sumário

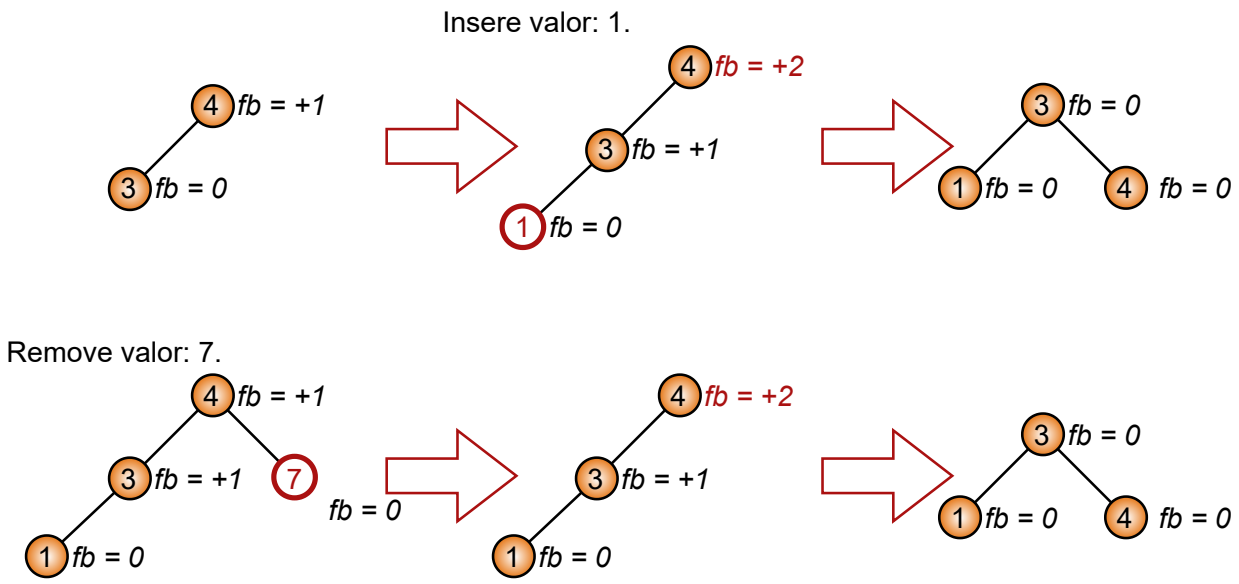
1. Balanceamento de ABB – Árvores AVL
   - Introdução
   - Árvore AVL
   - TAD árvore AVL
   - Rotações
     - Rotação LL (simples à direita)
     - Rotação RR (simples à esquerda)
     - Rotação LR (dupla à direita)
     - Rotação RL (dupla à esquerda)
     - Uso
   - Inserção de nó na AVL
   - Remoção de nó na AVL
   - Exercícios

Remoção de nó na AVL

- Percorre-se um conjunto de nós da árvore até chegar ao nó que será removido.
- Existem 3 tipos de remoção:
  1. Nó folha (sem filhos);
  2. Nó com um filho;
  3. Nó com dois filhos.

- Após cada remoção de um nó na AVL:
  - Volta-se pelo caminho percorrido, recalculando o fator de balanceamento de cada um dos nós visitados;
  - Aplica-se a rotação necessária em um nó onde $|fb| > 1$, restabelecendo o balanceamento da árvore.

Remover um nó da subárvore da **direita** equivale a **inserir** um nó na subárvore da **esquerda** (e vice-versa).



Uso de duas funções:

- `remove_ArvAVL()` – faz a busca pelo nó e sua remoção, além do rebalanceamento, quando necessário – disponível na interface com o usuário;
- `procuraMenor()`, auxiliar, procura o nó mais à esquerda da subárvore da esquerda do nó atual.

main.c: Exemplo de uso

```
int res,i;
int N = 10, dados[10] = {1,2,3,10,4,5,9,7,8,6};
ArvAVL* avl = cria_ArvAVL();
for(i=0; i<N; i++)
  res = insere_ArvAVL(avl,dados[i]);
.
.
if (remove_ArvAVL(raiz, 20))
  printf("20 foi excluido\n");
```


ArvoreAVL.c: remoção de nó

```
235 struct NO* procuraMenor(struct NO* atual) {
236   struct NO *no1 = atual;
237   struct NO *no2 = atual->esq;
238   while (no2 != NULL) {
239     no1 = no2;
240     no2 = no2->esq;
241   }
242   return no1;
243 }
244
245 int remove_ArvAVL(ArvAVL *raiz, int valor) {
246   if (raiz == NULL)
247     return 0;
248
249   int res;
250   if (*raiz == NULL) { // valor não existe
251     printf("valor não existe!!\n");
252     return 0;
253   }
```

```
254
255   if (valor < (*raiz)->info) {
256     if ((res = remove_ArvAVL(&(*raiz)->esq,valor)) ==
      1) { // removeu nó da subárvore da esquerda
257       if (fatorBalanceamento_NO(*raiz) >= 2) { // subá
      rvore da direita muito mais alta
258         if (altura_NO((*raiz)->dir->esq) <= altura_NO
      ((*raiz)->dir->dir))
259           RotacaoRR(raiz); // remoção correspondeu a
      inserção na subárvore da direita do filho da
      direita de atual
260         else
261           RotacaoRL(raiz); // remoção correspondeu a
      inserção na subárvore da esquerda do filho da
      direita de atual
262       }
263     }
264   }
265   if ((*raiz)->info < valor) {
```



```
266     if ((res = remove_ArvAVL(&(*raiz)->dir, valor)) ==
      1) { // removeu nó da subárvore da direita
267       if(fatorBalanceamento_NO(*raiz) >= 2) { // subá
      rvore da esquerda muito mais alta
268         if(altura_NO((*raiz)->esq->dir) <= altura_NO((*
      raiz)->esq->esq) )
269           RotacaoLL(raiz); // remoção correspondeu a
      inserção na subárvore da esquerda do filho da
      esquerda de atual
270         else
271           RotacaoLR(raiz); // remoção correspondeu a
      inserção na subárvore da direita do filho da
      esquerda de atual
272       }
273     }
274   }
275   if ((*raiz)->info == valor) {
276     if (((*raiz)->esq == NULL || (*raiz)->dir == NULL))
      {// nó tem somente um ou nenhum filho
277       struct NO *oldNode = (*raiz);
```

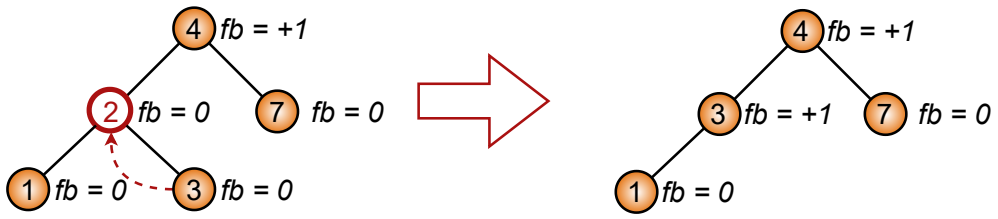
```
      if ((*raiz)->esq != NULL)
        *raiz = (*raiz)->esq;
      else
        *raiz = (*raiz)->dir;
      free(oldNode);
    } else {  // nó tem dois filhos: substitui-o pelo nó mais à esquerda da subárvore da direita
      struct NO* temp = procuraMenor((*raiz)->dir);
      (*raiz)->info = temp->info;
      remove_ArvAVL(&(*raiz)->dir, (*raiz)->info);
      if (fatorBalanceamento_NO(*raiz) >= 2) {
        if(altura_NO((*raiz)->esq->dir) <= altura_NO((*raiz)->esq->esq))
          RotacaoLL(raiz);
        else
          RotacaoLR(raiz);
      }
    }
    if (*raiz != NULL)
```



```
      (*raiz)->altura = maior(altura_NO((*raiz)->esq), altura_NO((*raiz)->dir)) + 1; // atualiza a altura da raiz
    return 1;
  }
  (*raiz)->altura = maior(altura_NO((*raiz)->esq), altura_NO((*raiz)->dir)) + 1; // atualiza a altura da raiz
  return res;
}
```

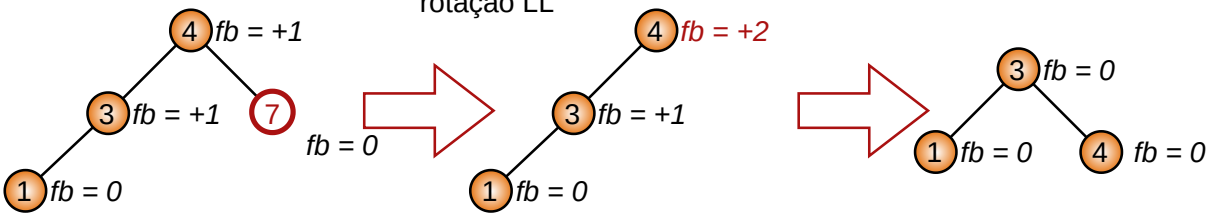
Passo a passo

Remove valor: 2.



Remove valor: 7.

Nó 4 desbalanceado: aplicar rotação LL



Sumário

1. Balanceamento de ABB – Árvores AVL
   - Introdução
   - Árvore AVL
   - TAD árvore AVL
   - Rotações
     - Rotação LL (simples à direita)
     - Rotação RR (simples à esquerda)
     - Rotação LR (dupla à direita)
     - Rotação RL (dupla à esquerda)
     - Uso
   - Inserção de nó na AVL
   - Remoção de nó na AVL
   - Exercícios

Exercícios I

1. Crie uma árvore AVL a partir da inserção dos seguintes valores, nesta ordem, mostrando o passo a passo a cada inserção:
   55, 42, 28, 74, 63, 12, 48, 54, 17, 9, 2, 99.
2. Agora, remova os nós da árvore, um a um, na ordem em que foram inseridos.

Referências

1. BACKES, A. R. **Algoritmos e Estruturas de Dados em Linguagem C**. 1. ed. Rio de Janeiro: LTC, 2023. Disponível em: https://www.sistemas.ufu.br/biblioteca-gateway/minhabiblioteca/9788521638315.
2. CORMEN, T. H.; et al. **Algoritmos**. 4. ed. Rio de Janeiro: GEN LTC, 2024. Disponível em: https://www.sistemas.ufu.br/biblioteca-gateway/minhabiblioteca/9788595159914.

Os materiais de parte desta seção foram gentilmente cedidos por André R. Backes (UFSCar); Bruno Travençolo e Christiane Brasil (FACOM/UFU).
TAD árvore AVL: disponível em:
https://www.facom.ufu.br/~backes/wordpress/ProjArvoreAVL.zip.
Acesso em: 18/mar/2025.
Adaptações: Renato Pimentel, FACOM/UFU