SCC-223 Estruturas de Dados I

Árvores AVL – Parte 3

Profa. Elaine Parros Machado de Sousa

ICMC USP SÃO CARLOS

Aula baseada em material do professor Rudinei Goularte.

Grupo de Bases de Dados e Imagens
USP - São Carlos

Nas aulas anteriores...

- AVL
  - definição
  - fator de balanceamento
- Rotações
- Inserção
- Remoção
- Exercícios

TAD AVL - Definição de Tipos

```
// AVL.c

#define max(a, b) ((a > b) ? a : b)

typedef struct no NO;

struct no {
      ITEM item;
      struct NO *fesq;
      struct NO *fdir;
      int altura;  // altura do nó
  };

 struct avl {
     NO *raiz;
  };
```

TAD AVL – Funções Básicas

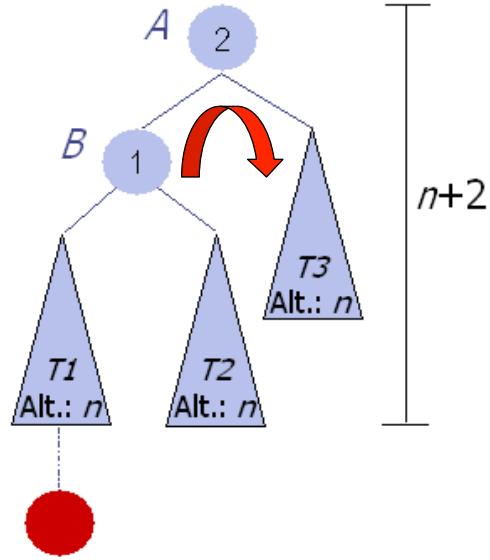
```
/* AVL.h – somente funções e tipos visíveis para a
   aplicação */

// #define  ...
typedef struct avl AVL;

typedef int ITEM;


AVL *avl_criar(void);

void avl_apagar(AVL **arvore);

Boolean avl_inserir(AVL *T, ITEM item);

Boolean avl_remover(AVL *T, int chave);

ITEM* avl_buscar(AVL *T, int chave);

int avl_altura(AVL *T);
```

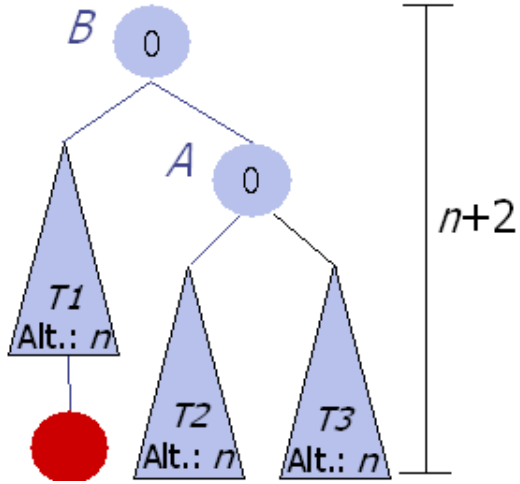
TAD AVL – Funções Auxiliares

```
/* AVL.c - implementação do tad e funções/tipos
   internos - não visíveis para a aplicação */


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

TAD AVL - Rotação Direita

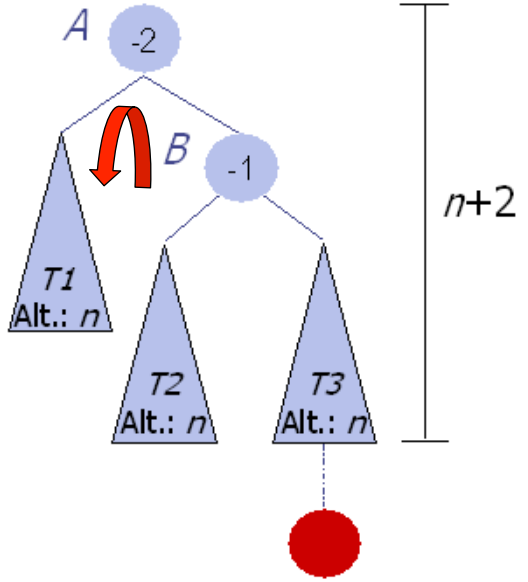
```
// AVL.c
NO *rodar_direita(NO *a) {
   NO *b = a->fesq;
   a->fesq = b->fdir;
   b->fdir = a;
   // atualizar alturas de A e B
   a->altura = max(avl_altura_no(a->fesq),
                   avl_altura_no(a->fdir)) + 1;

   b->altura = max(avl_altura_no(b->fesq),
                   a->altura) + 1;
    return b;
}
```

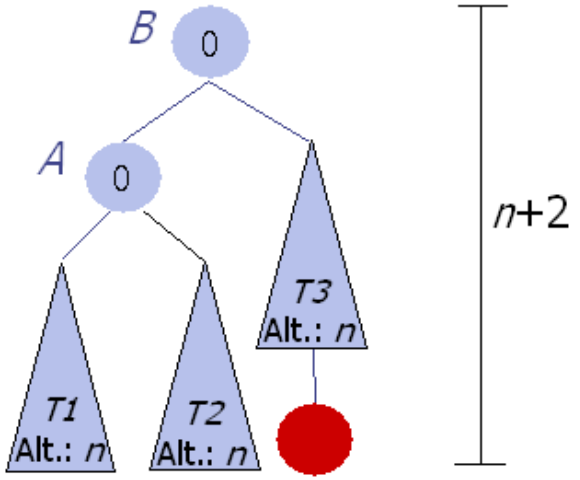




TAD AVL - Rotação Esquerda

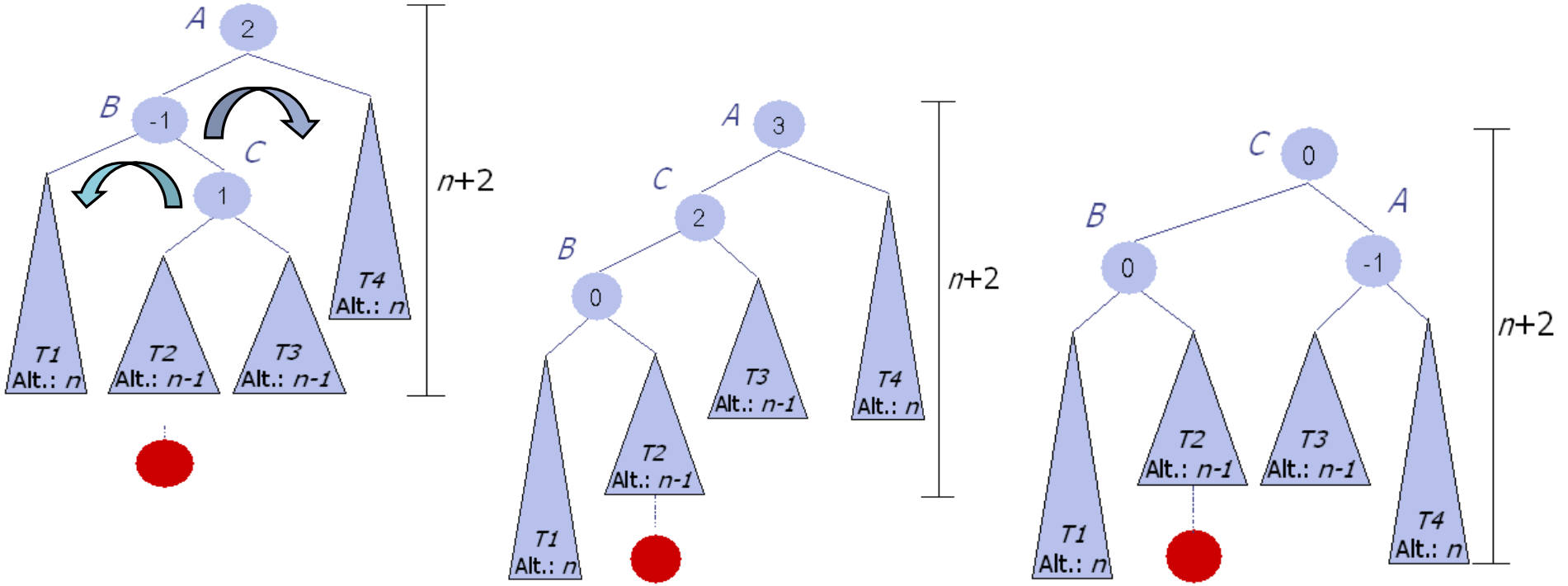
```
// AVL.c
NO *rodar_esquerda(NO *a) {
    NO *b = a->fdir;
    a->fdir = b->fesq;
    b->fesq = a;
    // atualizar alturas de A e B
    a->altura = max(avl_altura_no(a->fesq),
                    avl_altura_no(a->fdir)) + 1;

    b->altura = max(avl_altura_no(b->fdir)
                    a->altura) + 1;
    return b;
}
```





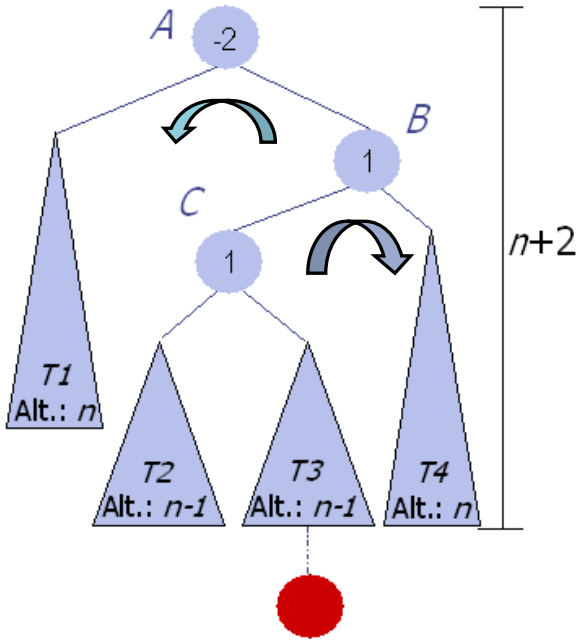
TAD AVL - Rotação Esquerda/Direita

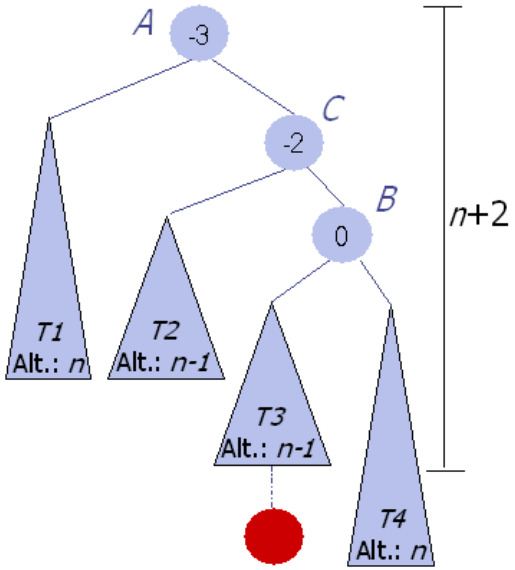
```
// AVL.c
NO *rodar_esquerda_direita(NO *a)   {
  a->fesq = rodar_esquerda(a->fesq);
  return rodar_direita(a);
}
```

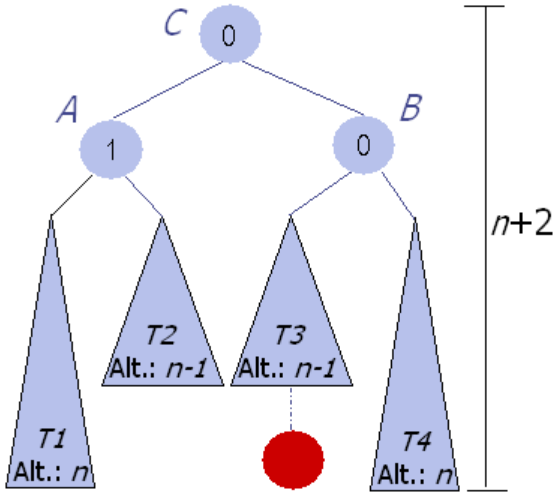


TAD AVL - Rotação Direita/Esquerda

```
// AVL.c
NO *rodar_direita_esquerda(NO *a)   {
   a->dir = rodar_direita(a->fdir);
   return rodar_esquerda(a);
 }
```







TAD AVL - Inserção

- Algoritmo de **inserção** em AVL utiliza as rotinas de rotação
- Implementação pode manter em cada nó:
  - a **altura** do nó
  - o fator de balanceamento do nó
- A inserção é feita em dois passos

  1) inserção em ABB e **atualização de alturas**;

  2) **rebalanceamento**, se necessário

Relembrando.... Inserção de nó na ABB

```
// função de apoio - interna no .c do TAD ABB
boolean abb_inserir_no(NO *raiz, ITEM item) {
  if (item < raiz->item) {    // no .h: #define ITEM int
     if(raiz->esq != NULL)
       return (abb_inserir_no(raiz->esq, item));
     else
        return (abb_inserir_filho(FILHO_ESQ, raiz, item)!=NULL);
   }
  else {
    if(item > raiz->item) {
      if(raiz->dir != NULL)
        return abb_inserir_no(raiz->dir, item);
      else
        return (abb_inserir_filho(FILHO_DIR, raiz, item)!=NULL);
     }
     else return (FALSE);
    }
}
```

TAD AVL - Rebalanceamento

- Na “volta” da inserção => **balanceamento** é verificado
  - se a árvore estiver desbalanceada, aplicar as rotações necessárias
  - desbalanceamento pode ser verificado com base na altura dos nós **=>** o fator de balanceamento não precisa ser, necessariamente, armazenado

TAD AVL - Rebalanceamento

- **Caso 1:** $altura\,(fesq) - altura\,(fdir) == -2$
  - **Caso 1.1:** $chave\,(novo) > chave\,(fdir)$ => rot. **Esquerda**
  - **Caso 1.2:** $chave\,(novo) < chave\,(fdir)$ => rot. **Direita/Esquerda**

- **Caso 2:** $altura\,(fesq) - altura\,(fdir) == 2$
  - **Caso 2.1:** $chave\,(novo) < chave\,(fesq)$ => rot. **Direita**
  - **Caso 2.2:** $chave\,(novo) > chave\,(fesq)$ => rot. **Esquerda/Direita**

Adaptando Inserção para AVL...

```
// função de apoio - interna no .c do TAD AVL
Boolean avl_inserir_no(NO *raiz, ITEM item) {
   Boolean res = TRUE; // retorno
   if (item < raiz->item) {
     if(raiz->esq != NULL)
       res = avl_inserir_no(raiz->esq, item);
     else res = avl_inserir_filho(FILHO_ESQ, raiz, item)!=NULL;
     if (avl_altura_no(raiz->fesq)
         - avl_altura_no(raiz->fdir) == 2) {
         if (item < (raiz->fesq->item))
             raiz = rodar_direita(raiz);  // Caso 2.1
         else
             raiz = rodar_esquerda_direita(raiz); // Caso 2.2
       }
   }
   ....
}
```

Caso 2

Adaptando Inserção para AVL...

```
// função de apoio - interna no .c do TAD AVL
Boolean avl_inserir_no(NO *raiz, ITEM item) {
   Boolean res = TRUE; // retorno
   if (item < raiz->item) {
      ... // insere esquerda
      ... // testa CASO 2
   }
   else {
         if (item > raiz->item) {
            ... // insere direita ...
             ...// testa CASO 1
         }
        else return (FALSE)
   }

   raiz->altura = max(avl_altura_no(raiz->fesq),
                      avl_altura_no(raiz->fdir)) + 1;
   return res;
}
```

Exercício

- Elabore e implemente o algoritmo para remoção em AVL