Árvores AVL

SCC0202 - Algoritmos e Estruturas de Dados I

Prof. Fernando V. Paulovich

**Baseado no material do Prof. Gustavo Batista. Figuras editadas por Isadora Maria Mendes*

http://www.icmc.usp.br/~paulovic

paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

13 de novembro de 2017

Sumário

Sumário

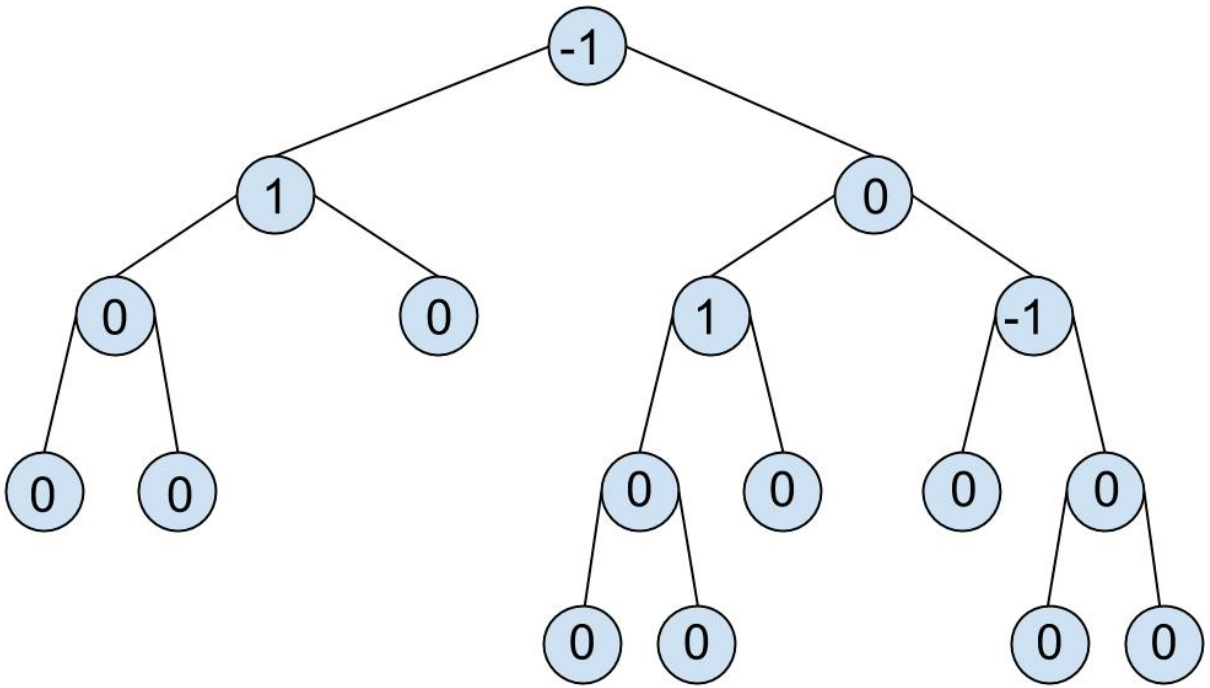
Árvores Binárias de Busca

- Altura de uma árvore binária (AB): igual à profundidade, ou nível máximo, de suas folhas
- A eficiência da busca em árvore depende do seu balanceamento
- Algoritmos de inserção e remoção em ABB não garantem que a árvore gerada a cada passo seja balanceada

Árvores AVL

- Árvore AVL: ABB na qual as alturas das duas sub-árvores de todo nó nunca diferem em mais de 1
  - Fator de balanceamento de nó: a altura de sua sub-árvore esquerda menos a altura de sua sub-árvore direita
    - $FB(p) = h(T_E(p)) - h(T_D(p))$
  - Em uma árvore AVL todo nó tem fator de balanceamento igual a 1, -1 ou 0

Árvores AVL

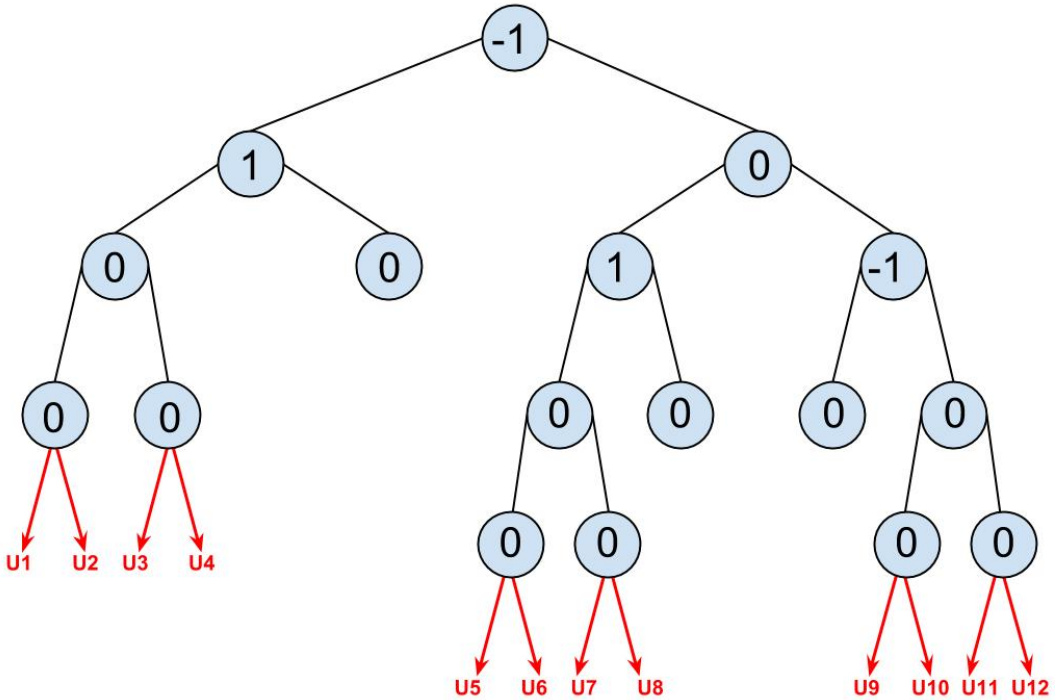


Árvores AVL

- O problema das árvores AVL e das árvores balanceadas de uma forma geral é como manter a estrutura balanceada após operações de inserção e remoção
- As operações de inserção e remoção sobre ABBs não garantem o balanceamento

Árvores AVL

- As seguintes inserções tornam a árvore desbalanceada



Árvores AVL

- As seguintes situações podem levar ao desbalaceamento de uma árvore AVL
  - O nó inserido é descendente esquerdo de um nó que tinha $FB = 1$ ($U1$ a $U8$)
  - O nó inserido é descendente direito de um nó que tinha $FB = -1$ ($U9$ a $U12$)

Árvores AVL

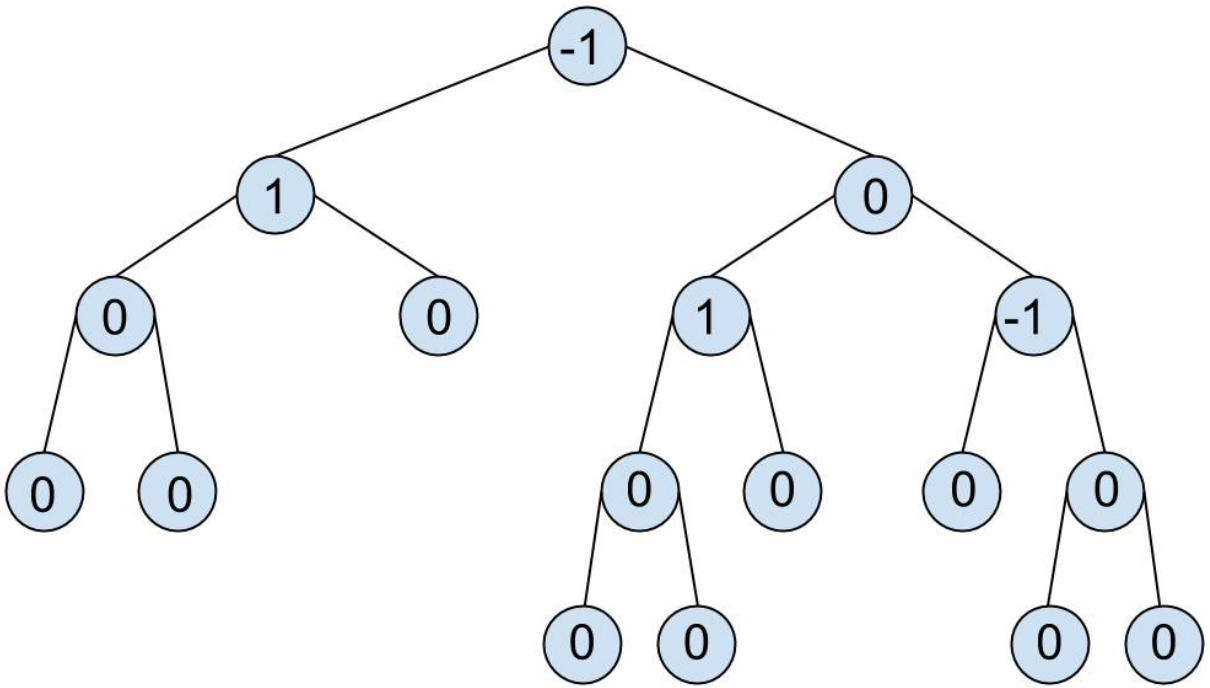
- Para manter uma árvore balanceada é necessário aplicar uma transformação na árvore tal que
  1. O percurso em-ordem na árvore transformada seja igual ao da árvore original (isto é, a árvore transformada continua sendo uma ABB)
  2. A árvore transformada fique balanceada

Árvores AVL

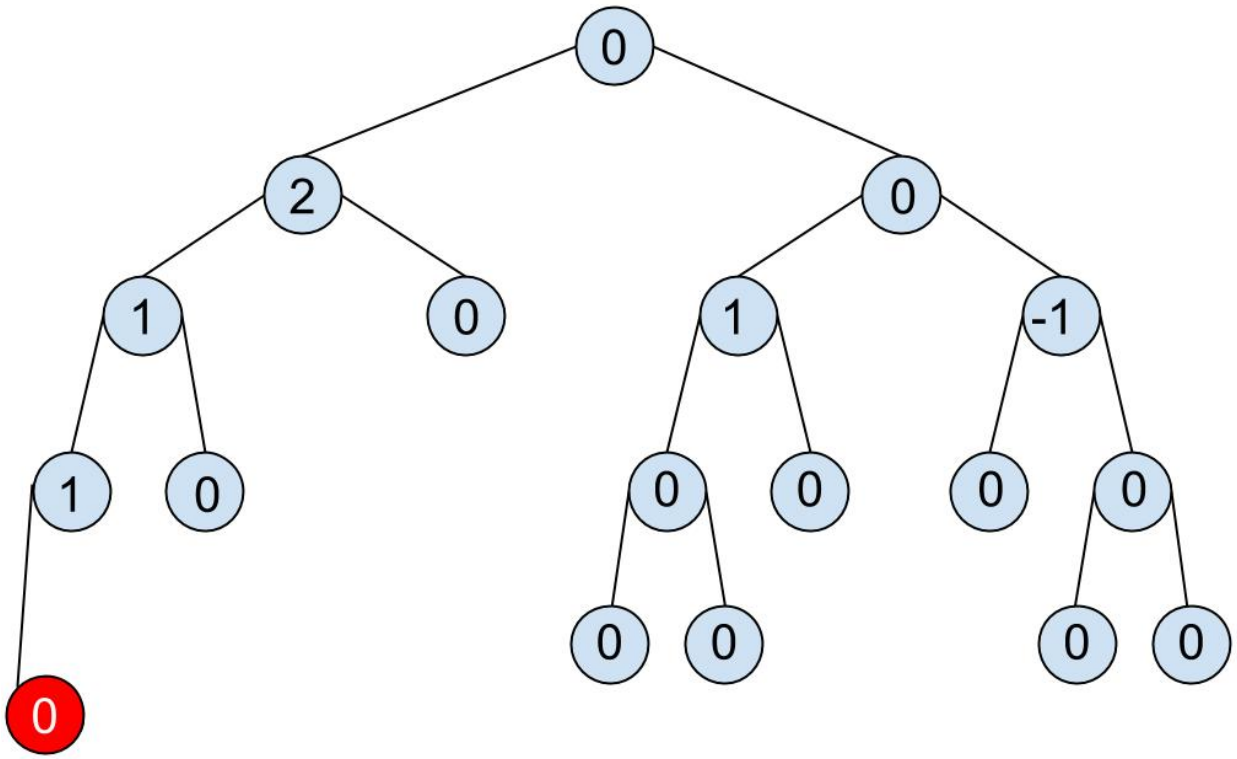
- A transformação que mantém a árvore balanceada é chamada de **rotação**
- A rotação pode ser feita à esquerda ou à direita, dependendo do desbalanceamento a ser tratado
- A rotação deve ser realizada de maneira a respeitar as regras 1 e 2 definidas no slide anterior
- Dependendo do desbalanceamento a ser tratado, uma única rotação pode não ser suficiente

Sumário

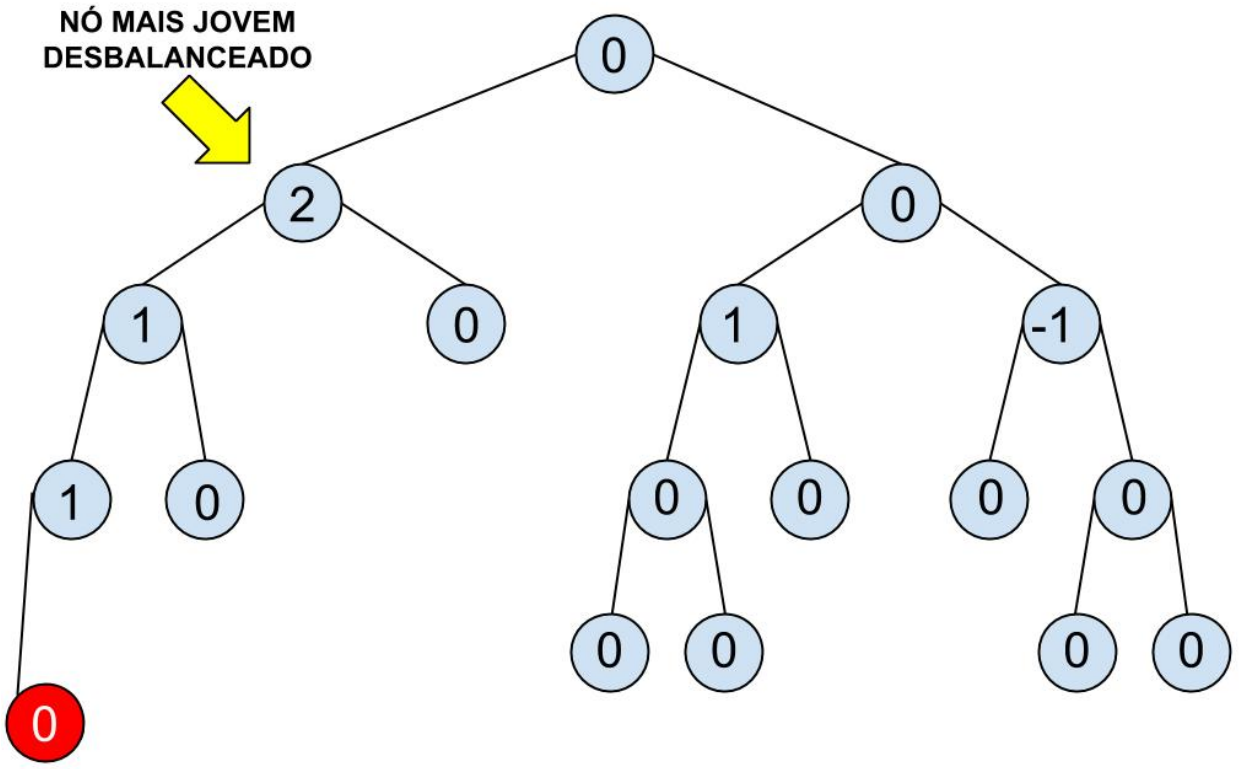
Árvores AVL - Rotação Direita



Árvores AVL - Rotação Direita

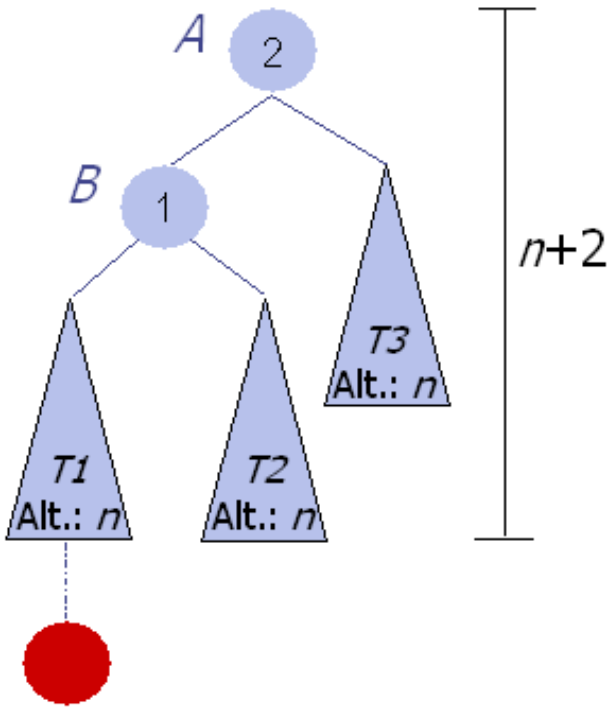


Árvores AVL - Rotação Direita



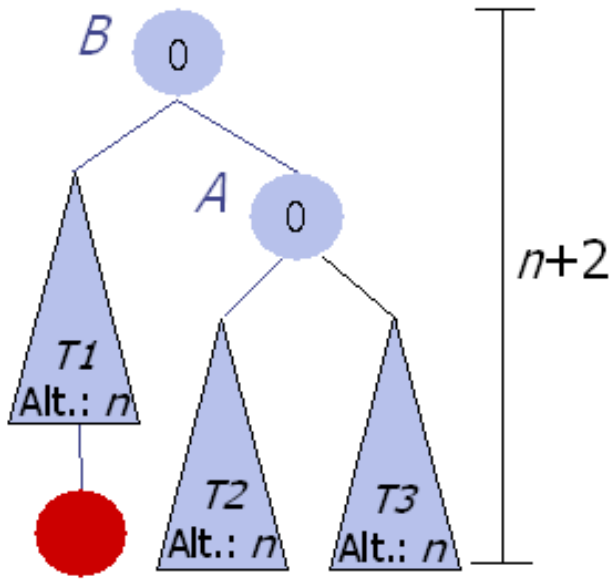
Árvores AVL - Rotação Direita

- A rotação direita tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado

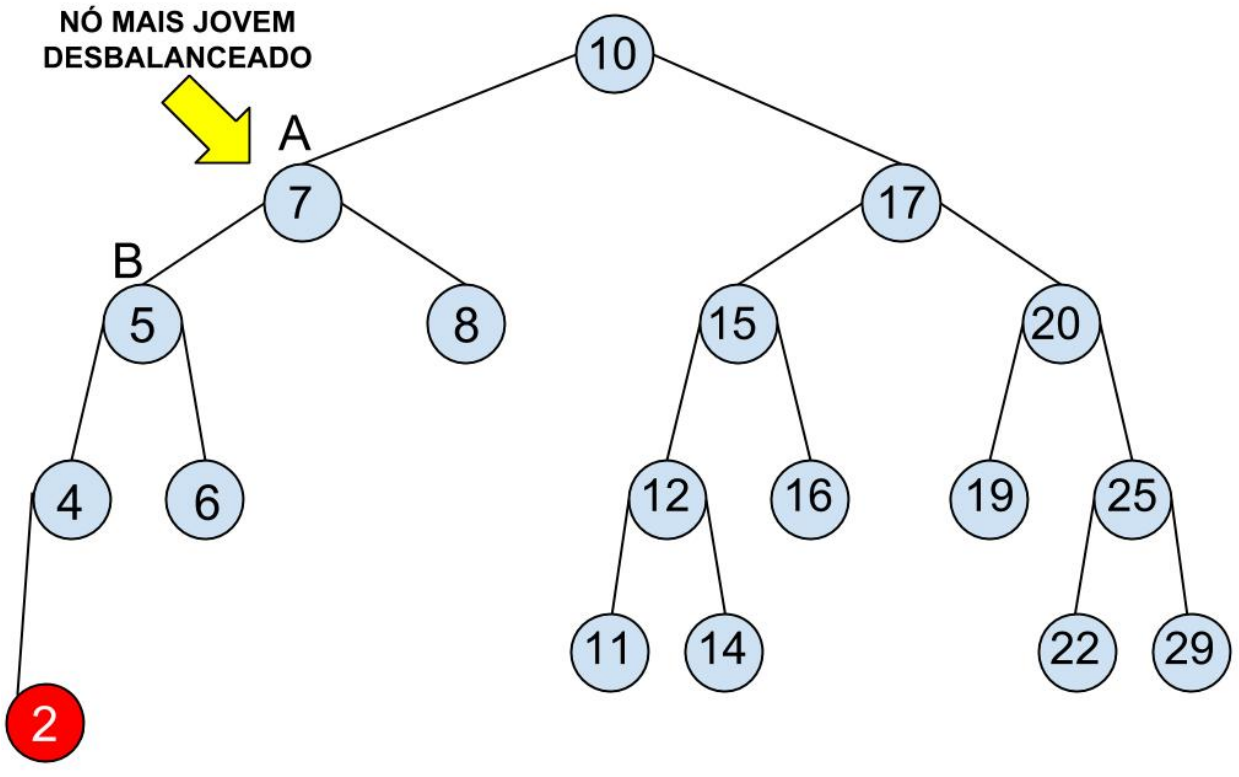


Árvores AVL - Rotação Direita

- A rotação direita tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado

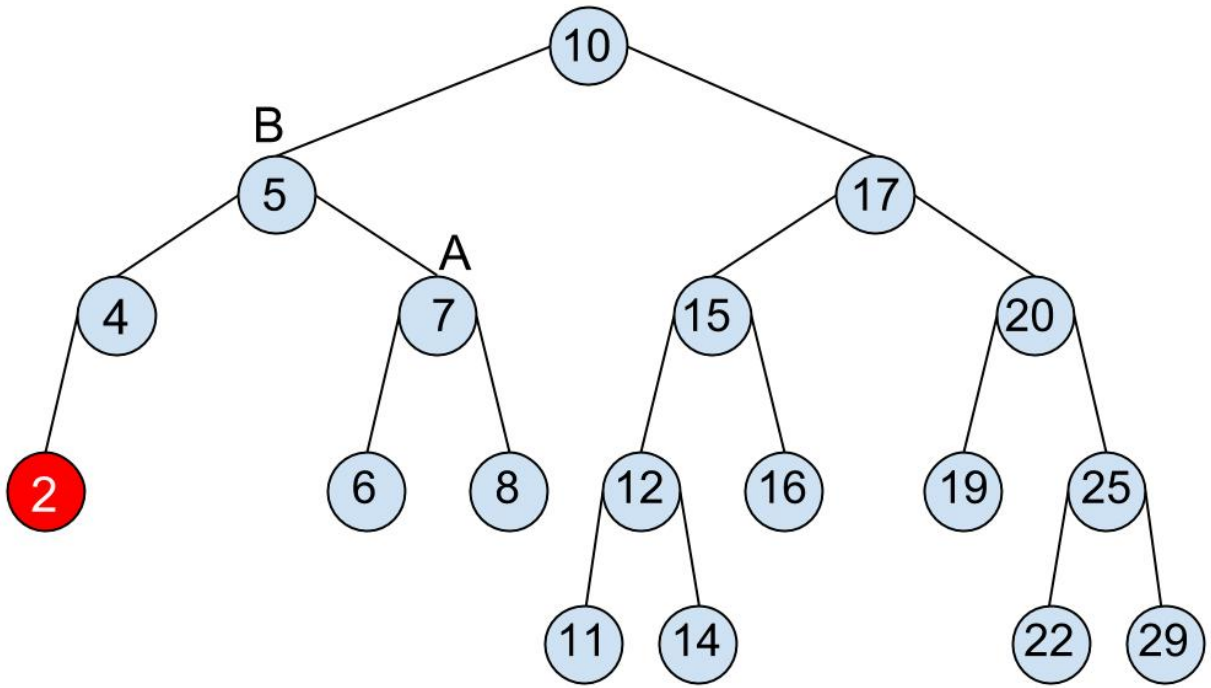


Árvores AVL - Rotação Direita



ANTES DA ROTAÇÃO DIREITA

Árvores AVL - Rotação Direita



APÓS A ROTAÇÃO DIREITA

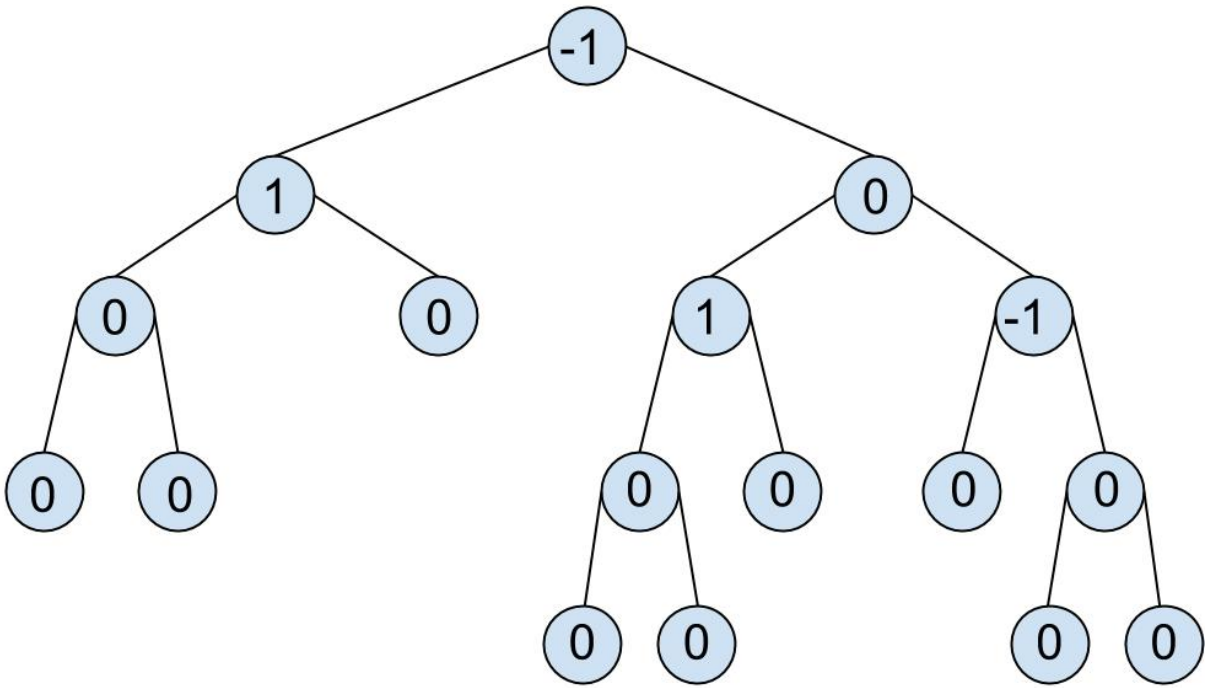
Árvores AVL - Rotação Direita

Exercício

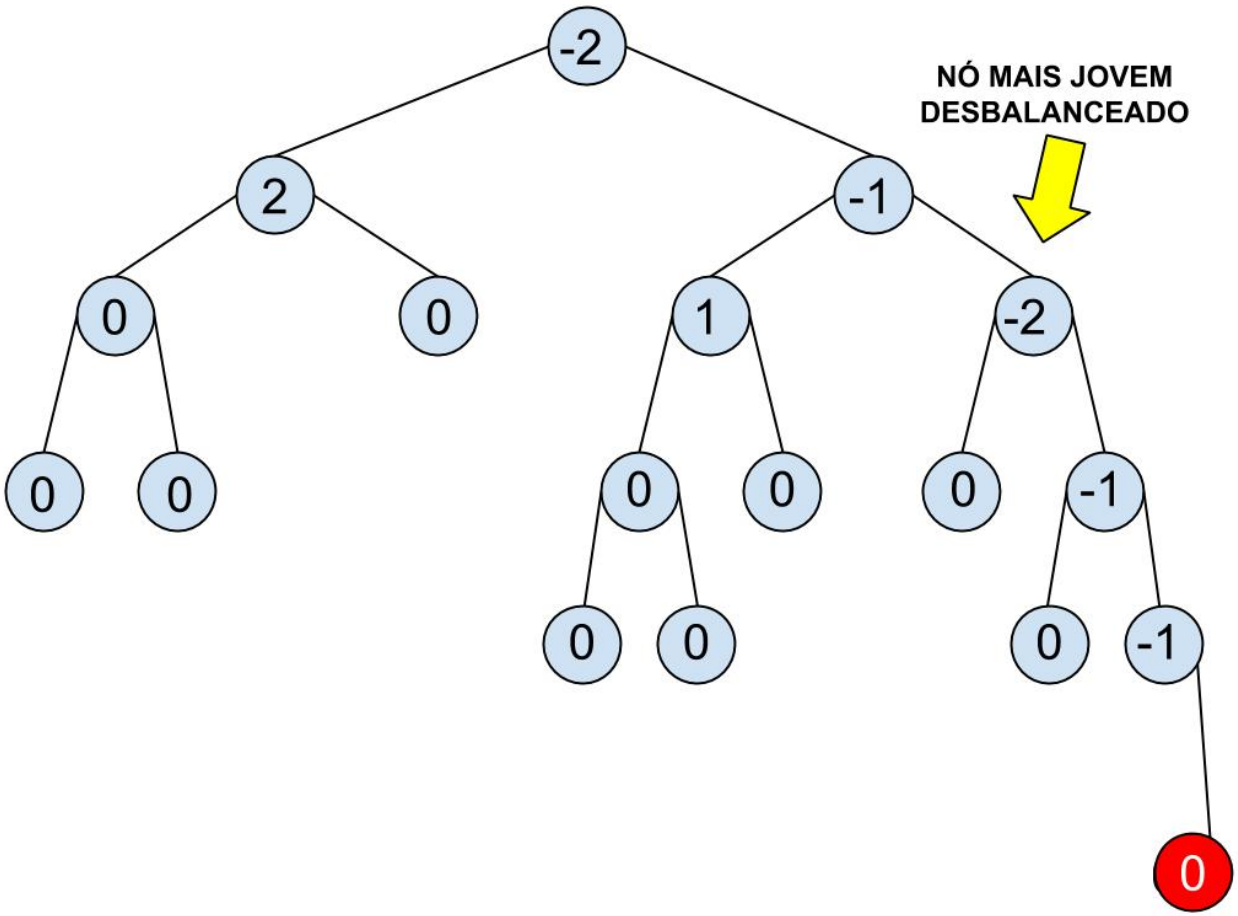
- Insira em uma árvore AVL a seqüência de valores: $5, 4, 3, 2, 1$. Na ordem que os valores foram listados

Sumário

Árvores AVL - Rotação Esquerda

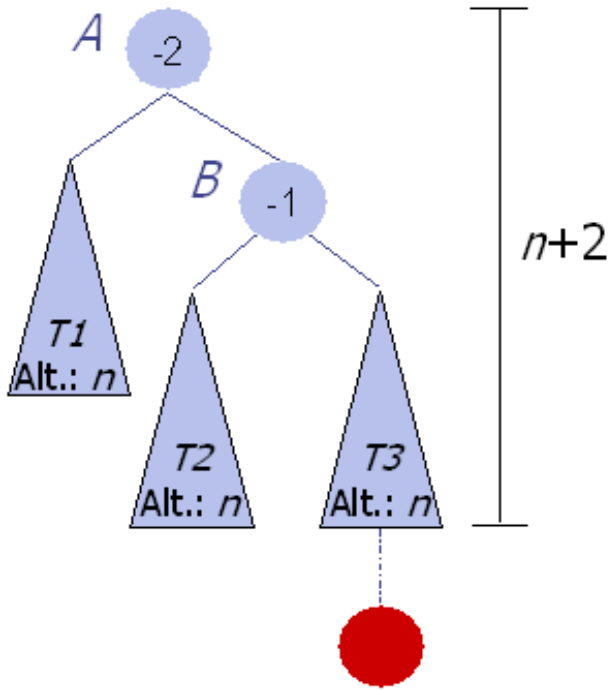


Árvores AVL - Rotação Esquerda



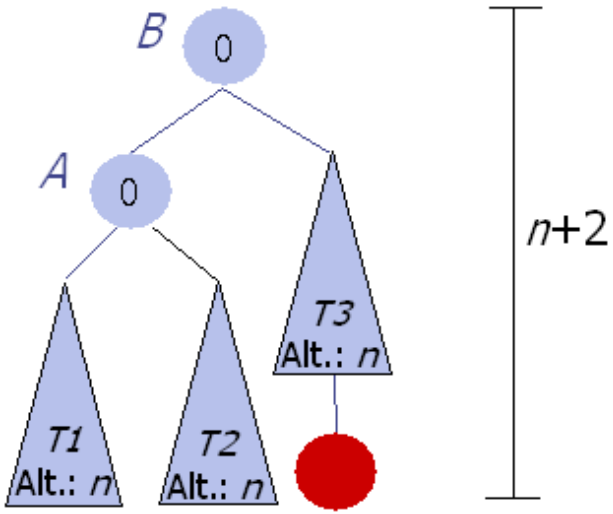
Árvores AVL - Rotação Esquerda

- A rotação esquerda tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado

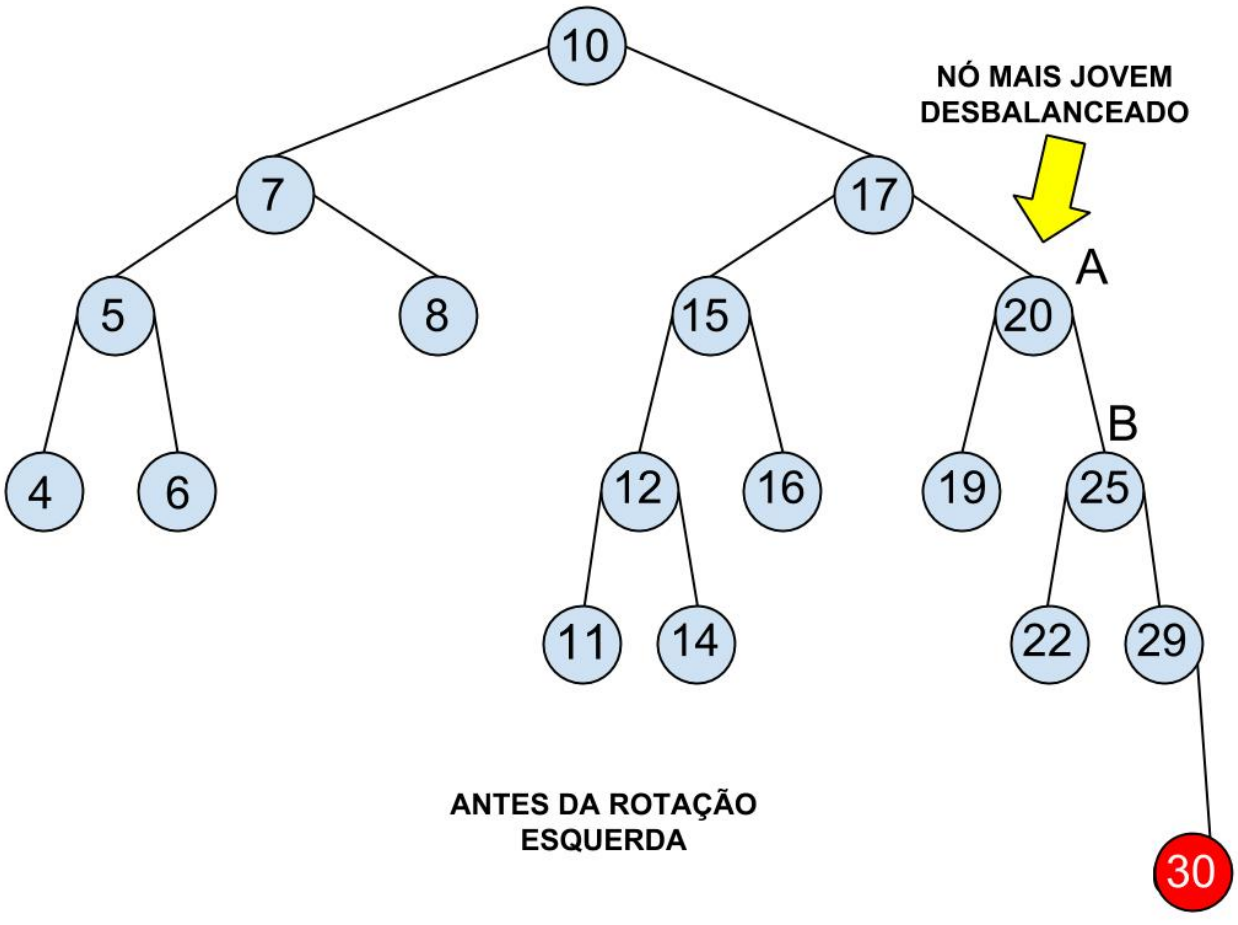


Árvores AVL - Rotação Esquerda

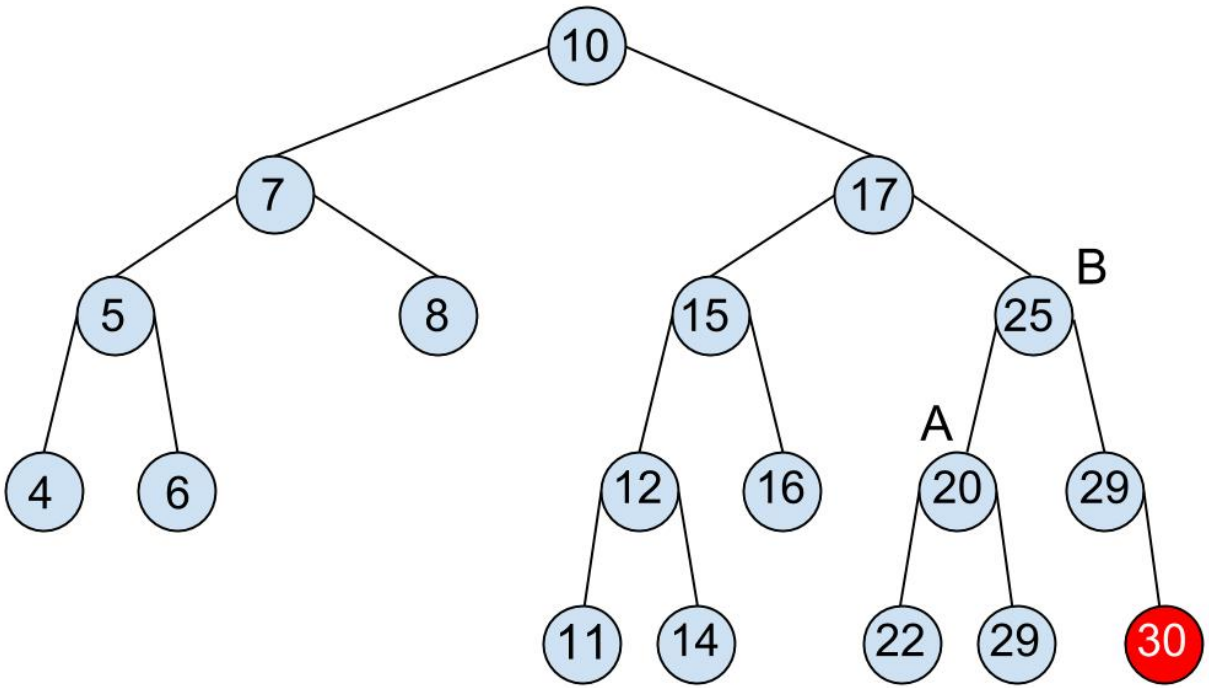
- A rotação esquerda tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado



Árvores AVL - Rotação Esquerda



Árvores AVL - Rotação Esquerda



APÓS A ROTAÇÃO ESQUERDA

Árvores AVL - Rotação Esquerda

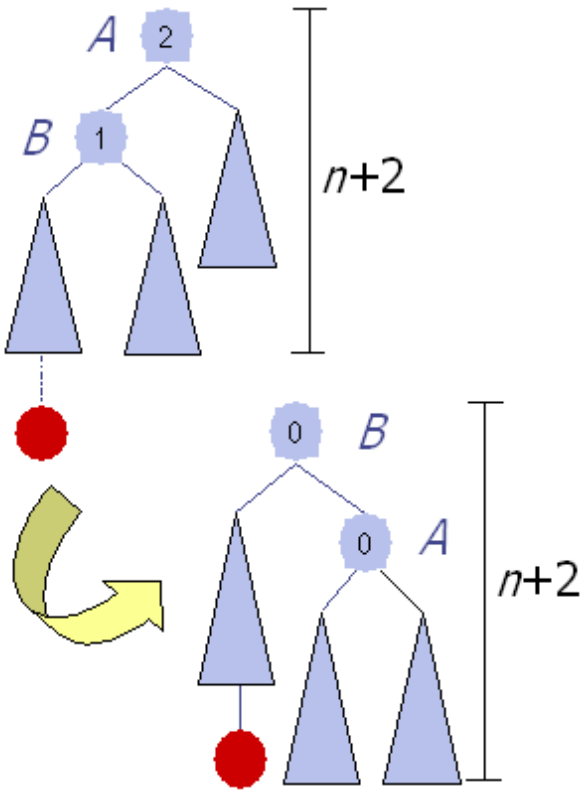
Exercício

- Insira em uma árvore AVL a seqüência de valores: $1, 2, 3, 4, 5$. Na ordem que os valores foram listados

Sumário

Rotações Simples

- Tanto para a rotação direita quanto para a rotação esquerda, a sub-árvore resultante tem como altura a mesma altura a sub-árvore original
- Isso significa que o fator de balanceamento de nenhum nó acima de A é afetado

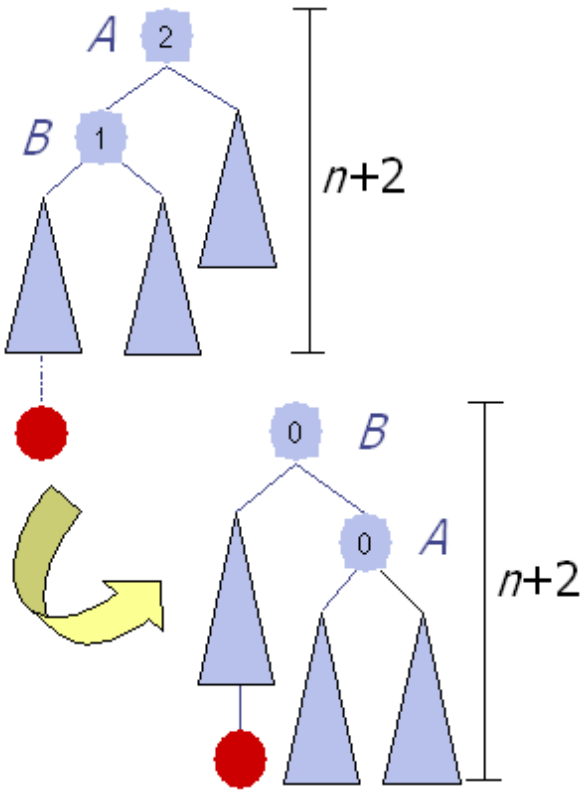


Rotações Simples

- Quando se deve utilizar a rotação direita ou esquerda?

Rotações Simples

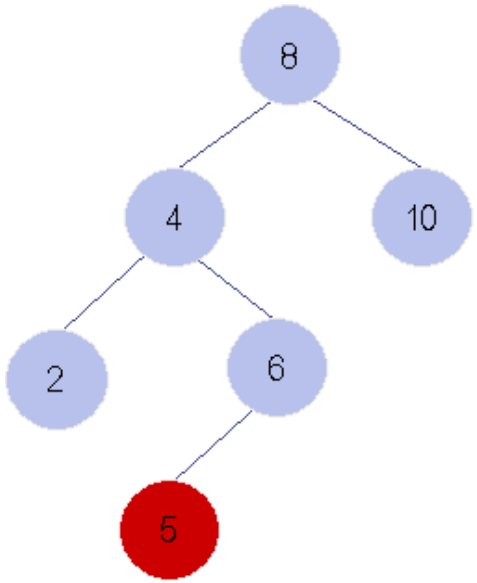
- Quando se deve utilizar a rotação direita ou esquerda?
  - Quando o fator de balanceamento do nó A é positivo, a rotação é direita. Se for negativo a rotação é esquerda



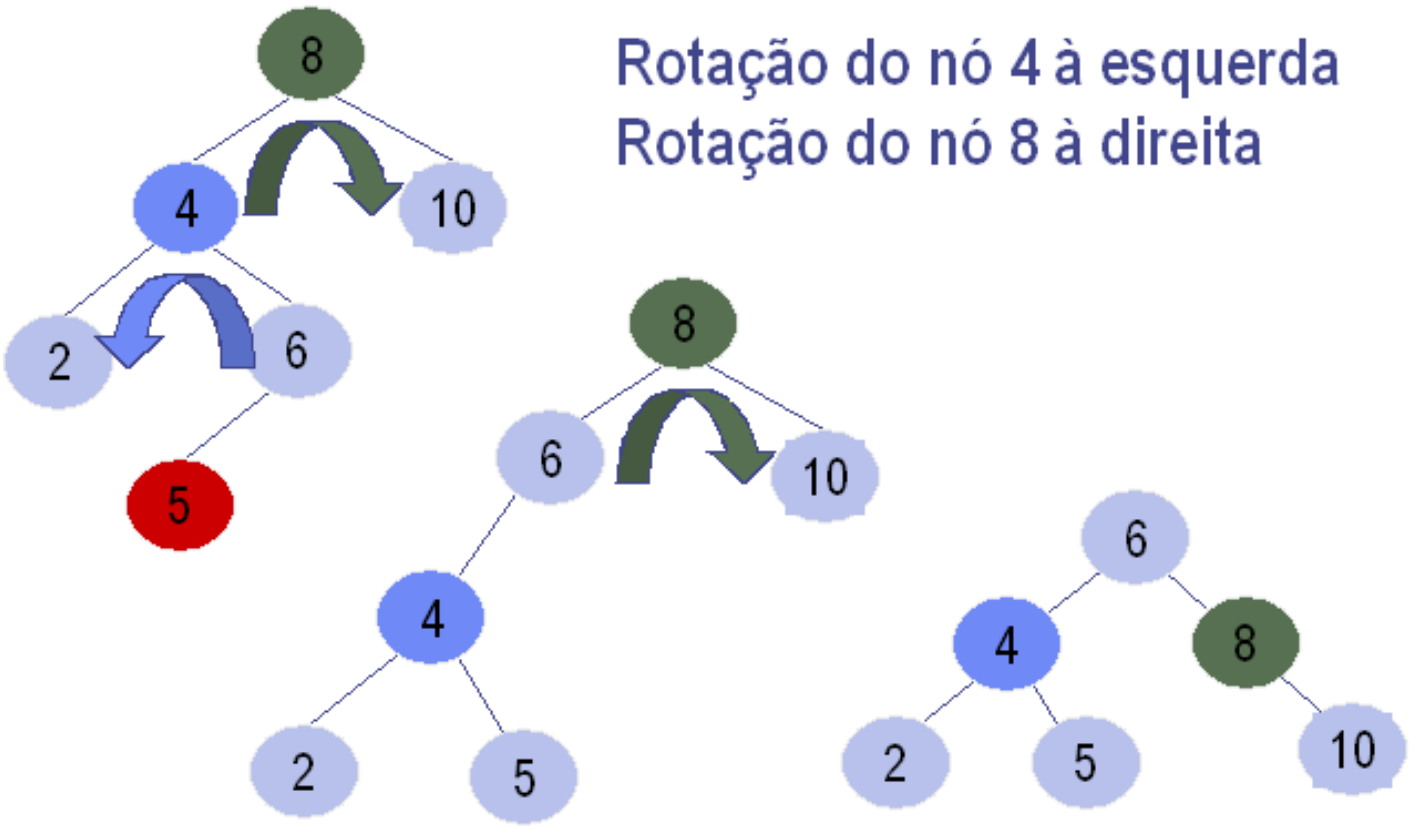
Sumário

Rotações Duplas

- Será que as rotações simples solucionam todos os tipos de desbalanceamento?
  - Infelizmente, não
- Existem situações nas quais é necessário uma rotação dupla

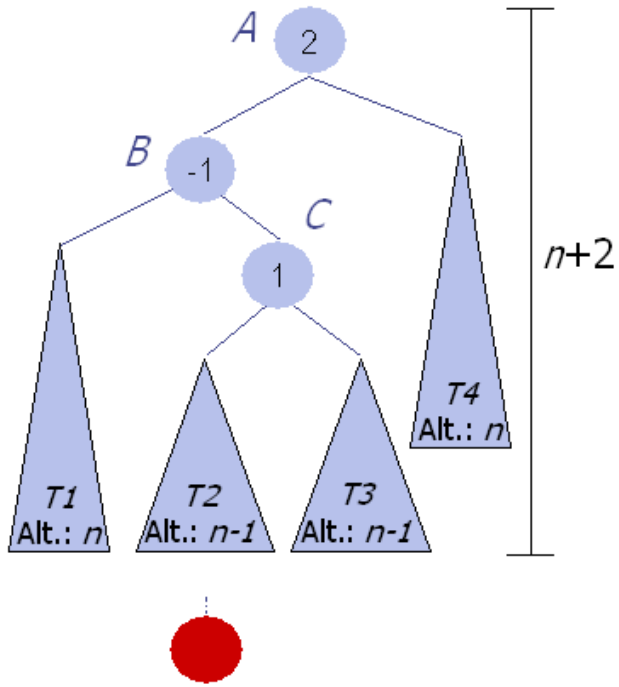


Rotações Duplas



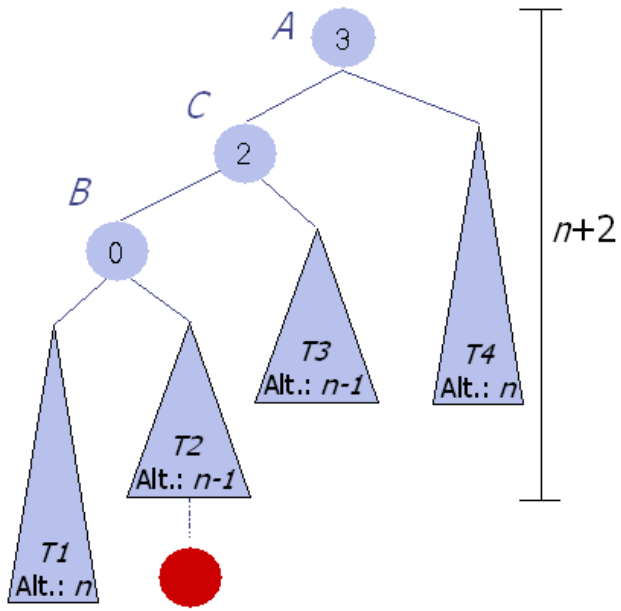
Árvores AVL - Rotação Esq./Dir.

- A rotação dupla esquerda/direita tem formato geral ilustrado à direita
- $T1$, $T2$, $T3$ e $T4$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado



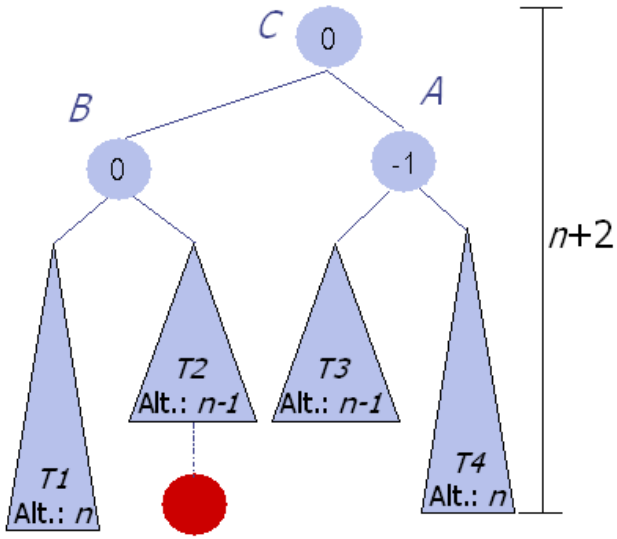
Árvores AVL - Rotação Esq./Dir.

- Passo 1: rotação esquerda em B
- A princípio a rotação esquerda parece deixar a árvore ainda mais desbalanceada
- Entretanto...



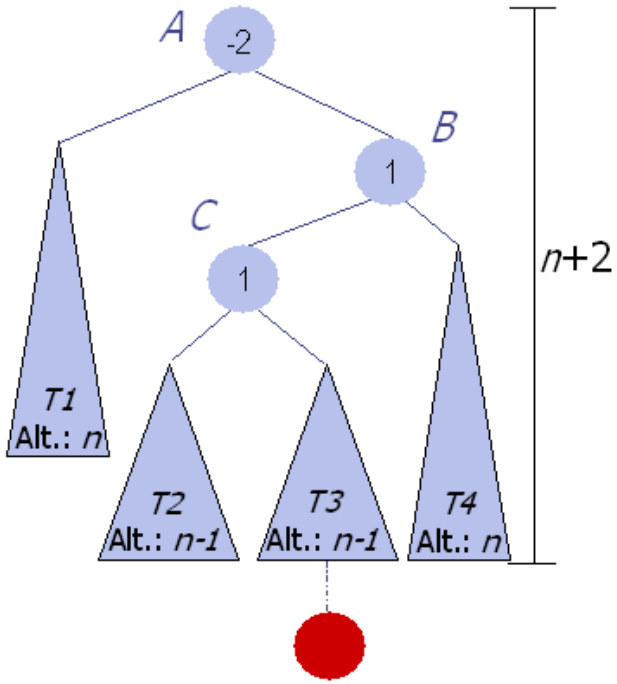
Árvores AVL - Rotação Esq./Dir.

- Passo 2: rotação direita em A
- Repare que a altura final da sub-árvore é $n + 2$
- Funciona também se o novo nó tivesse sido inserido em $T3$



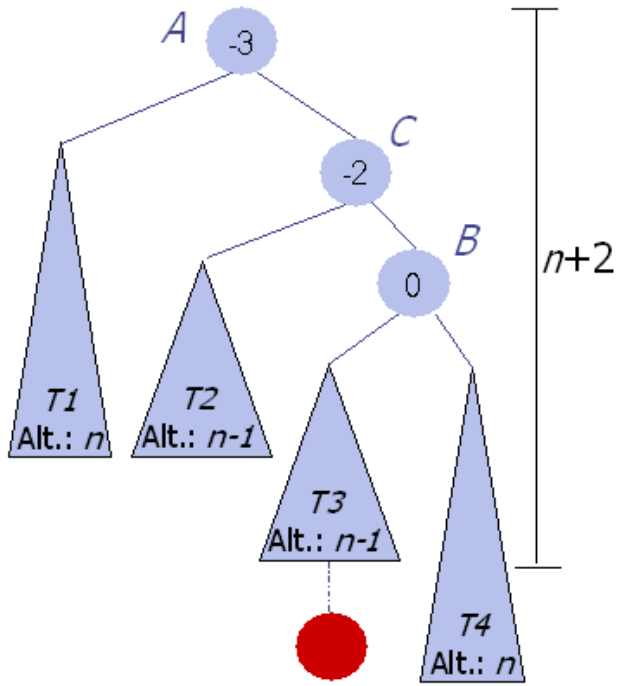
Árvores AVL - Rotação Esq./Dir.

- A rotação dupla direita/esquerda tem formato geral ilustrado à direita
- $T1$, $T2$, $T3$ e $T4$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado



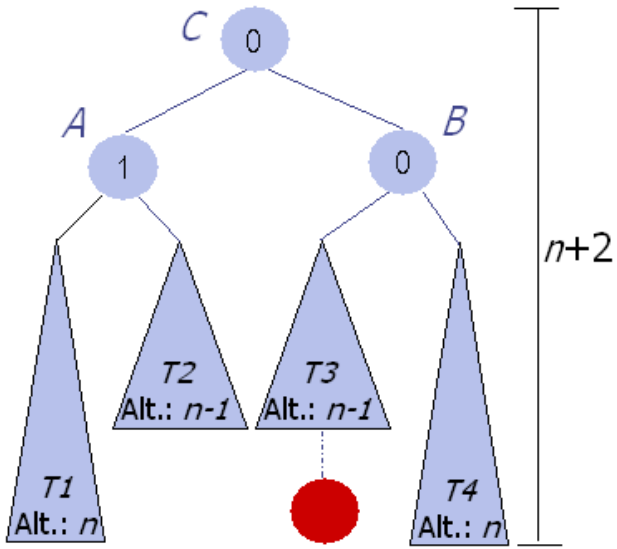
Árvores AVL - Rotação Dir./Esq.

- Passo 1: rotação direita em B
- A princípio a rotação direita parece deixar a árvore ainda mais desbalanceada
- Entretanto...



Árvores AVL - Rotação Dir./Esq.

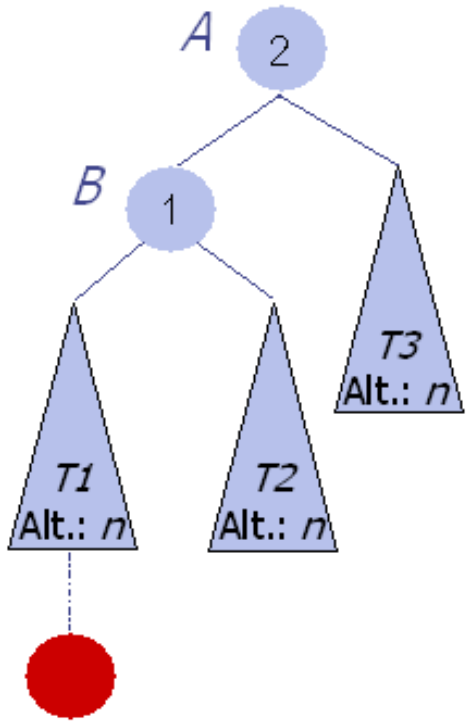
- Passo 2: rotação esquerda em A
- Repare que a altura final da sub-árvore é $n + 2$
- Funciona também se o novo nó tivesse sido inserido em $T2$



Sumário

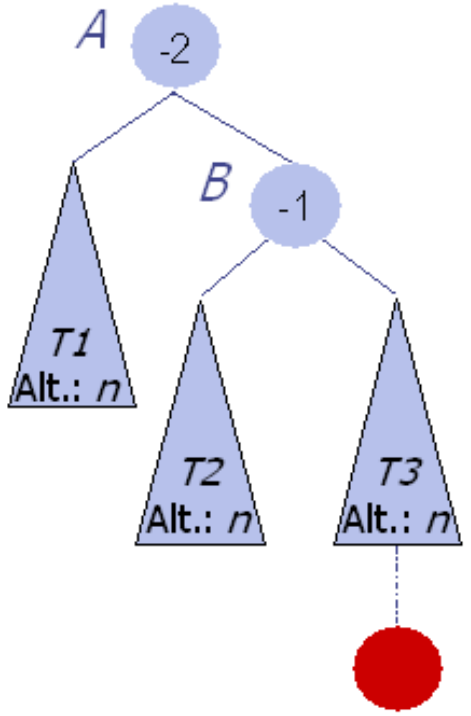
Como decidir qual rotação usar?

- Se o **sinal** do nó A e do nó B forem **iguais** então a rotação é **simples**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **positivo**, então a rotação é **direita**



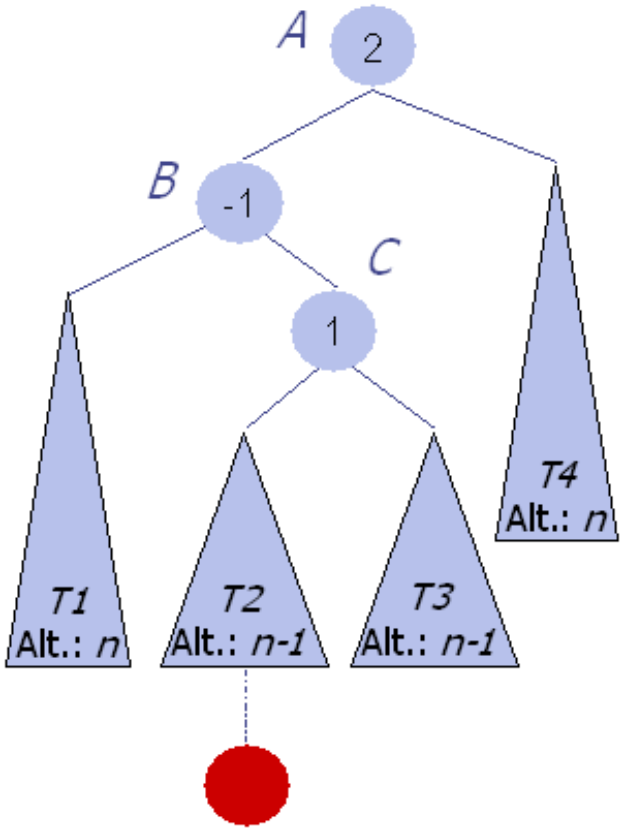
Como decidir qual rotação usar?

- Se o **sinal** do nó A e do nó B forem **iguais** então a rotação é **simples**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **negativo**, então a rotação é **esquerda**



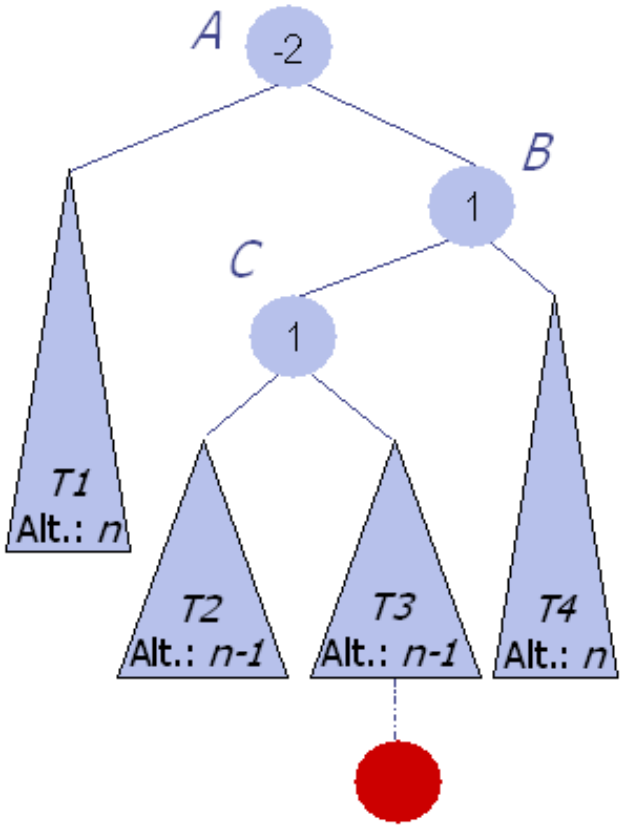
Como decidir qual rotação usar?

- Se o *sinal* do nó A e do nó B forem **diferentes** então a rotação é **dupla**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **positivo**, então a rotação é **esquerda/direita**



Como decidir qual rotação usar?

- Se o **sinal** do nó A e do nó B forem **diferentes** então a rotação é **dupla**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **negativo**, então a rotação é **direita/esquerda**



Sumário

Definição de Tipos

```
1  typedef struct arvore_avl ARVORE_AVL;
2
3  typedef struct NO {
4    ITEM *item;
5    struct NO *fesq;
6    struct NO *fdir;
7    int altura;
8  } NO;
9
10 struct arvore_avl {
11   NO *raiz;
12 };
```

Métodos Básicos

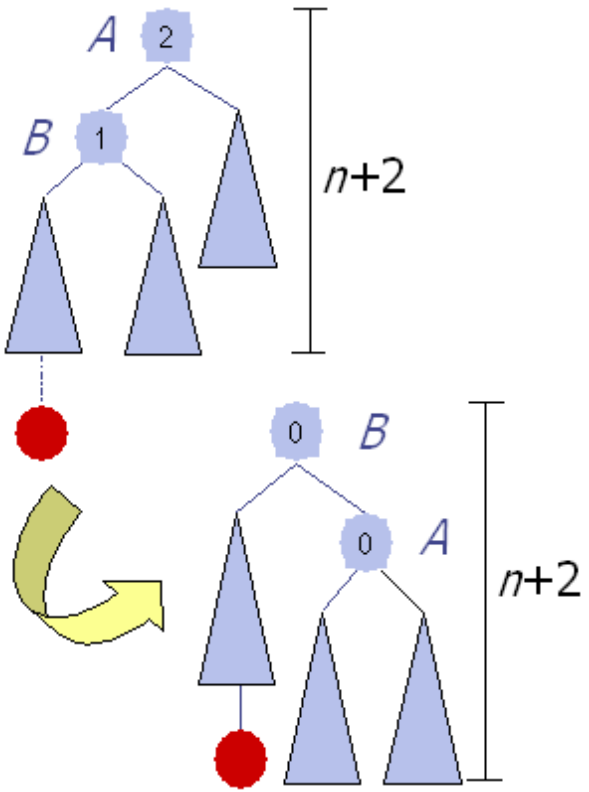
```
1  ARVORE_AVL *criar_avl() {
2    ARVORE_AVL *arvore = (ARVORE_AVL *) malloc(sizeof (ARVORE_AVL));
3    if (arvore != NULL) {
4      arvore->raiz = NULL;
5    }
6    return arvore;
7  }
8
9  void apagar_avl_aux(NO *raiz) {
10   if (raiz != NULL) {
11     apagar_avl_aux(raiz->fesq);
12     apagar_avl_aux(raiz->fdir);
13     apagar_item(&(raiz->item));
14     free(raiz);
15   }
16 }
17
18 void apagar_avl(ARVORE_AVL **arvore) {
19   apagar_avl_aux((*arvore)->raiz);
20   free(*arvore);
21   *arvore = NULL;
22 }
```

Métodos Auxiliares

```
1  #define max(a, b) ((a > b) ? a : b)
2
3  int altura(NO* raiz) {
4    if (raiz == NULL) {
5      return -1;
6    } else {
7      return raiz->altura;
8    }
9  }
10
11 NO *novo_no(ITEM *item) {
12   NO *no = (NO *) malloc(sizeof (NO));
13   if (no != NULL) {
14     no->altura = 0;
15     no->fdir = NULL;
16     no->fesq = NULL;
17     no->item = item;
18   }
19   return no;
20 }
```

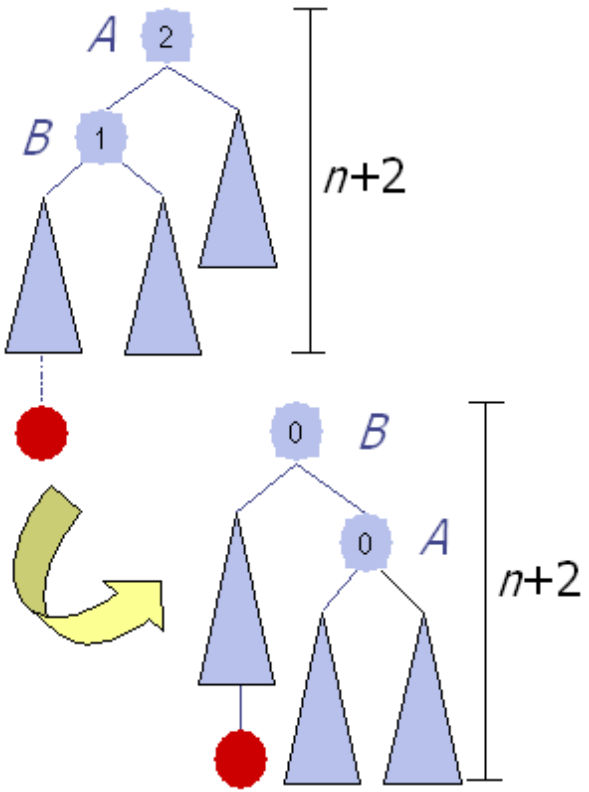
Algoritmo - Rotação Direita

```
NO *rodar_direita(NO *a) {
  NO *b = a->fesq;
  a->fesq = b->fdir;
  b->fdir = a;

  a->altura = max(altura(a->fesq),
                  altura(a->fdir)) + 1;
  b->altura = max(altura(b->fesq),
                  a->altura) + 1;

  return b;
}
```



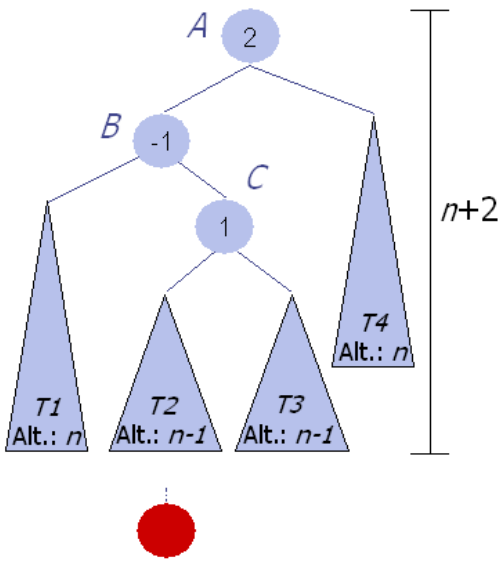
Algoritmo - Rotação Esquerda

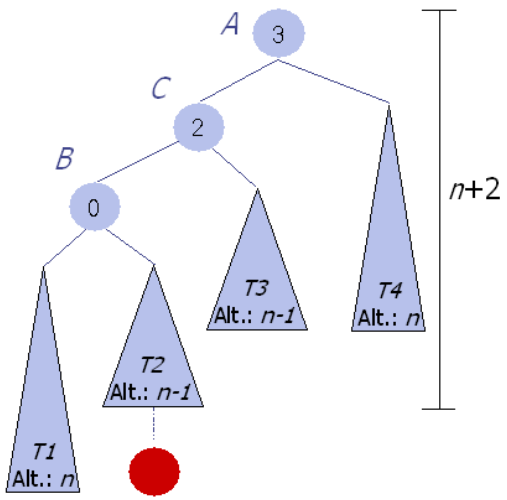
```
NO *rodar_esquerda(NO *a) {
  NO *b = a->fdir;
  a->fdir = b->fesq;
  b->fesq = a;

  a->altura = max(altura(a->fesq),
                  altura(a->fdir)) + 1;
  b->altura = max(altura(b->fdir),
                  a->altura) + 1;

  return b;
}
```

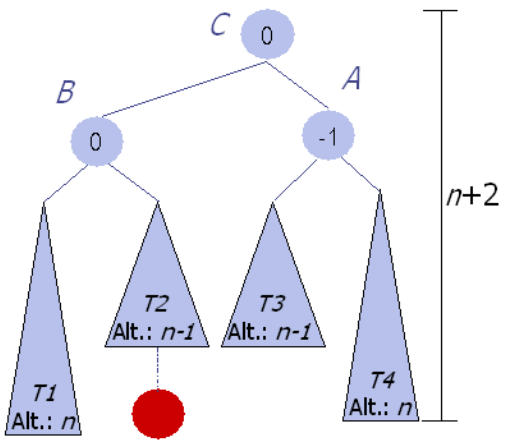


Algoritmo - Rotação Esq./Dir.

```
1  NO *rodar_esquerda_direita(NO *a) {
2    a->fesq = rodar_esquerda(a->fesq);
3    return rodar_direita(a);
4  }
```

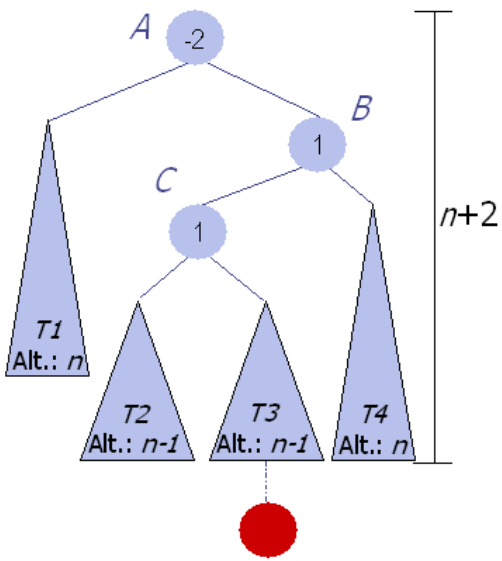


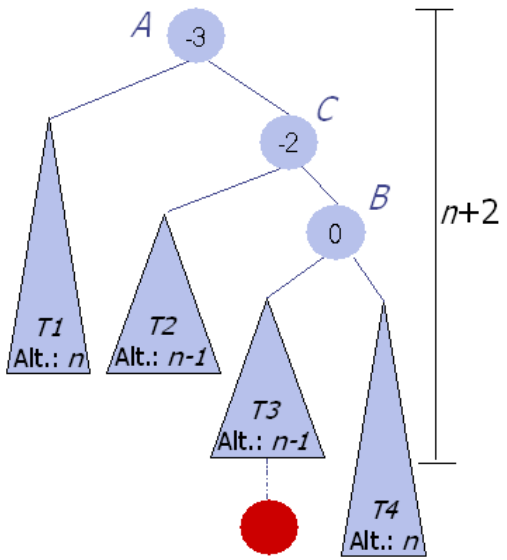


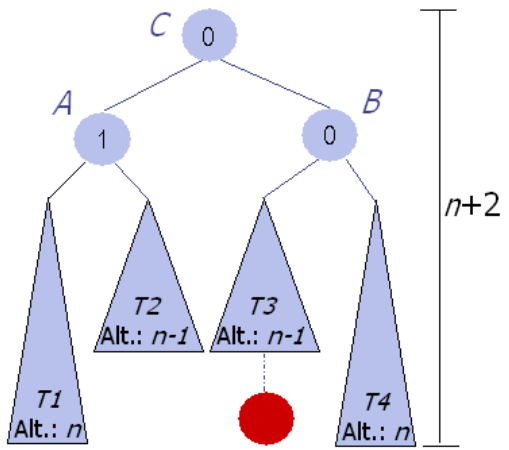


Algoritmo - Rotação Dir./Esq.

```
1  NO *rodar_direita_esquerda(NO *a) {
2    a->fdir = rodar_direita(a->fdir);
3    return rodar_esquerda(a);
4  }
```







Sumário

Algoritmo de Inserção

- Utilizando as rotinas de rotação pode-se definir um algoritmo de inserção em árvores AVL
- A maioria das implementações guardam o fator de balanceamento, porém guardar a altura dos nós facilita
- A inserção é feita em dois passos
  - o primeiro é uma inserção em ABBs; e
  - o segundo é o rebalanceamento, se necessário

Algoritmo de Inserção

- A primeira etapa é definir uma inserção em ABB e atualizar as alturas dos nós

Algoritmo de Inserção

```
1  NO *inserir_avl_aux(NO *raiz, ITEM *item) {
2    if (raiz == NULL) {
3      raiz = novo_no(item);
4    } else if (item->chave > raiz->item->chave) {
5      raiz->fdir = inserir_avl_aux(raiz->fdir, item);
6    } else if (item->chave < raiz->item->chave) {
7      raiz->fesq = inserir_avl_aux(raiz->fesq, item);
8    }
9
10   raiz->altura = max(altura(raiz->fesq), altura(raiz->fdir)) + 1;
11
12   return raiz;
13 }
14
15 int inserir_avl(ARVORE_AVL *arvore, ITEM *item) {
16   return (arvore->raiz = inserir_avl_aux(arvore->raiz, item)) != NULL;
17 }
```

Algoritmo de Inserção

- Na volta da inserção o balanceamento é verificado, se a árvore estiver desbalanceada, aplicar as rotações necessárias

- O desbalaceamento é verificado com base na altura dos nós, o fator de desbalanceamento não precisa ser armazenado

Algoritmo de Inserção

- Se $altura(fesq) - altura(fdir) == -2$ as rotações podem ser
  - Esquerda
  - Direita/Esquerda
- Se $chave(novo) > chave(fdir)$ rotação Esquerda, caso contrário rotação Direita/Esquerda

- Se $altura(fesq) - altura(fdir) == 2$ as rotações podem ser
  - Direita
  - Esquerda/Direita
- Se $chave(novo) < chave(fesq)$ rotação Direita, caso contrário rotação Esquerda/Direita

Algoritmo de Inserção

```
1  NO *inserir_avl_aux(NO *raiz, ITEM *item) {
2    if (raiz == NULL) {
3      raiz = novo_no(item);
4    } else if (item->chave > raiz->item->chave) {
5      raiz->fdir = inserir_avl_aux(raiz->fdir, item);
6      if (altura(raiz->fesq) - altura(raiz->fdir) == -2) {
7        if (item->chave > raiz->fdir->item->chave) {
8          raiz = rodar_esquerda(raiz);
9        } else {
10         raiz = rodar_direita_esquerda(raiz);
11       }
12     }
13   } else if (item->chave < raiz->item->chave) {
14     raiz->fesq = inserir_avl_aux(raiz->fesq, item);
15     if (altura(raiz->fesq) - altura(raiz->fdir) == 2) {
16       if (item->chave < raiz->fesq->item->chave) {
17         raiz = rodar_direita(raiz);
18       } else {
19         raiz = rodar_esquerda_direita(raiz);
20       }
21     }
22   }
23
24   raiz->altura = max(altura(raiz->fesq), altura(raiz->fdir)) + 1;
25
26   return raiz;
27 }
```

Remoção em AVLs

- Para eliminar um nó de uma árvore AVL, o algoritmo é um pouco mais complicado
- Enquanto que a inserção pode requerer no máximo uma rotação (simples ou dupla), a remoção pode requerer mais de uma rotação
  - No pior caso, pode-se fazer uma rotação a cada nível da árvore
  - Ou seja, no pior caso O($\log\ n$) rotações
  - Na prática, são necessárias apenas $0,214$ rotação por eliminação, em média

Complexidade das AVLs

- A altura máxima de uma ABB AVL é $1,44 \log_2 n$
  - Dessa forma, uma pesquisa nunca exige mais do que 44% mais comparações que uma ABB totalmente balanceada.
- Na prática, para n grande, os tempos de busca são por volta de $\log_2 n + 0,25$
- Na média, é necessária uma rotação em $46,5\%$ das inserções

Exercícios

- Simule a inserção da seguinte seqüência de valores em uma árvore AVL: $10, 7, 20, 15, 17, 25, 30, 5, 1$
- Em cada opção abaixo, insira as chaves na ordem mostrada de forma a construir uma arvore AVL. Se houver rebalanceamento de nós, mostre qual o procedimento a fazer
  1. a, z, b, y, c, x
  2. $a, z, b, y, c, x, d, w, e, v, f$
  3. $a, v, l, t, r, e, i, o, k$
  4. m, t, e, a, z, g, p

Exercícios

- Escreva uma função que retorna a altura da árvore AVL. Qual é a complexidade da operação implementada? Ela é mais eficiente que a implementação para ABBs?
- Implemente o TAD AVL com as operações de inserção e busca e demais operações auxiliares

Exercícios

- Mostre a árvores AVL gerada passo-a-passo pelas inserções das seguintes chaves na ordem fornecida
  - $10, 5, 20, 1, 3, 4, 8, 30, 40, 35, 50, 45, 55, 51, 100$