SCC0202 – Algoritmos e Estruturas de Dados I

Árvores AVL

Prof.: Dr. Rudinei Goularte
(rudinei@icmc.usp.br)
Instituto de Ciências Matemáticas e de Computação - ICMC
Sala 4-229

Conteúdo

- Conceitos Introdutórios
- Rotação Direita
- Rotação Esquerda
- Rotações Simples
- Rotações Duplas
- Qual Rotação Usar
- Implementação
- Inserção em Árvores AVL
- Remoção em Árvores AVL

Árvores Binárias de Busca

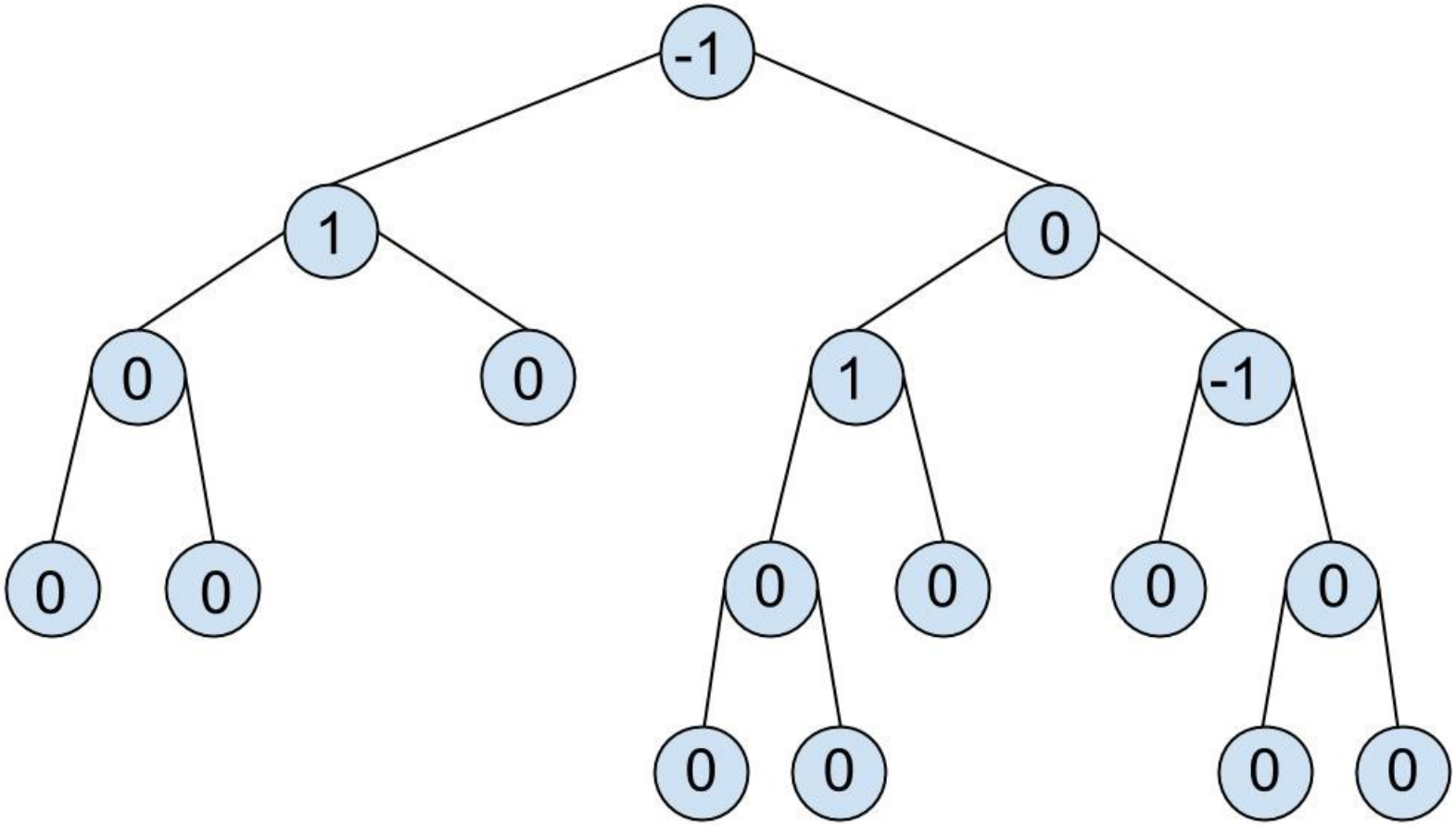
- Altura de uma árvore binária (AB): igual à profundidade, ou nível máximo, de suas folhas
- A eficiência da busca em árvore depende do seu balanceamento
- Algoritmos de inserção e remoção em ABB não garantem que a árvore gerada a cada passo seja balanceada

- Árvore balanceada é aquela que ...

Árvores AVL

- Proposta em 1962 pelos matemáticos russos G.M. Adelson-Velskki e E.M. Landis
- 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
  - Em uma árvore AVL todo nó tem fator de balanceamento igual a 1, -1 ou 0

Árvores AVL

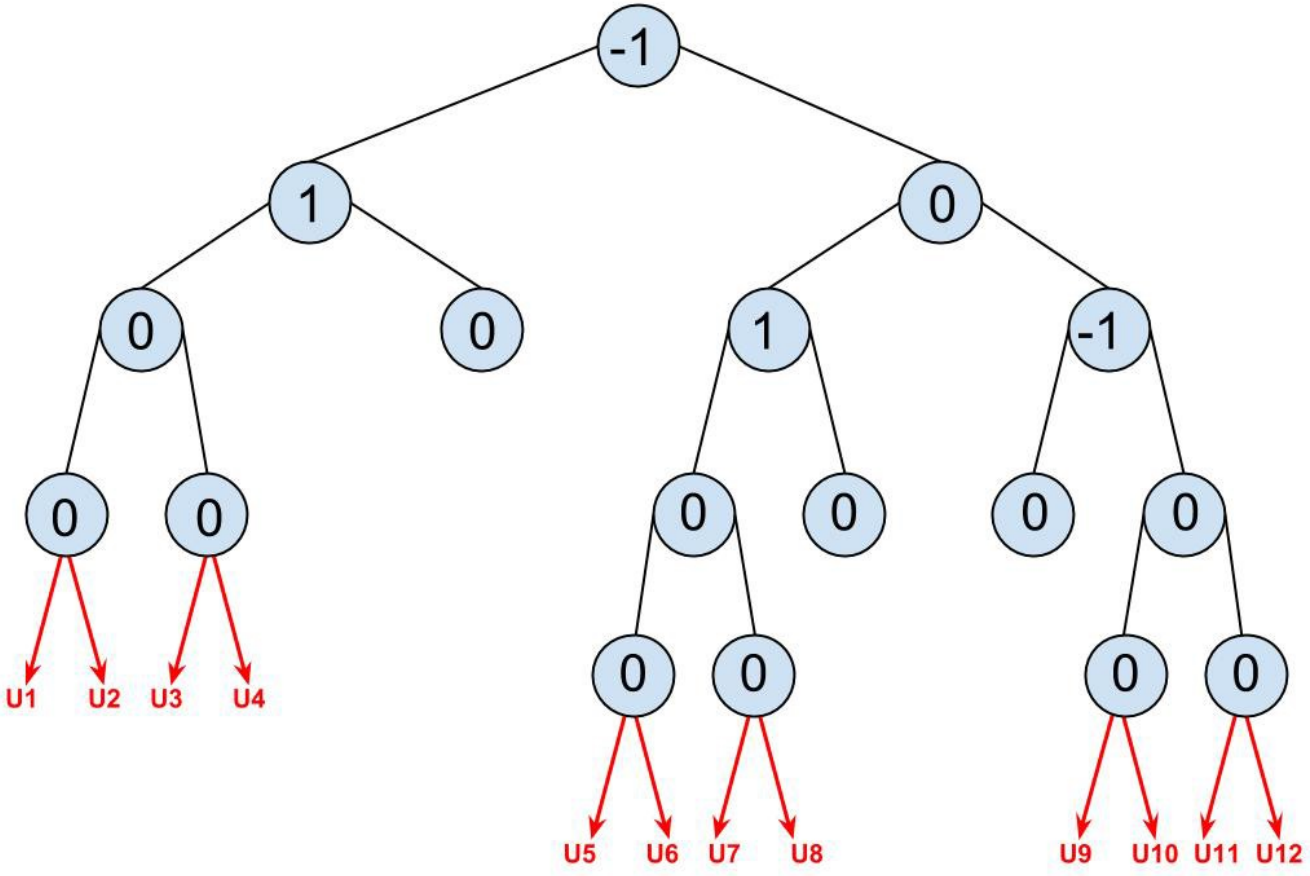


Árvores AVL

- O problema 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)

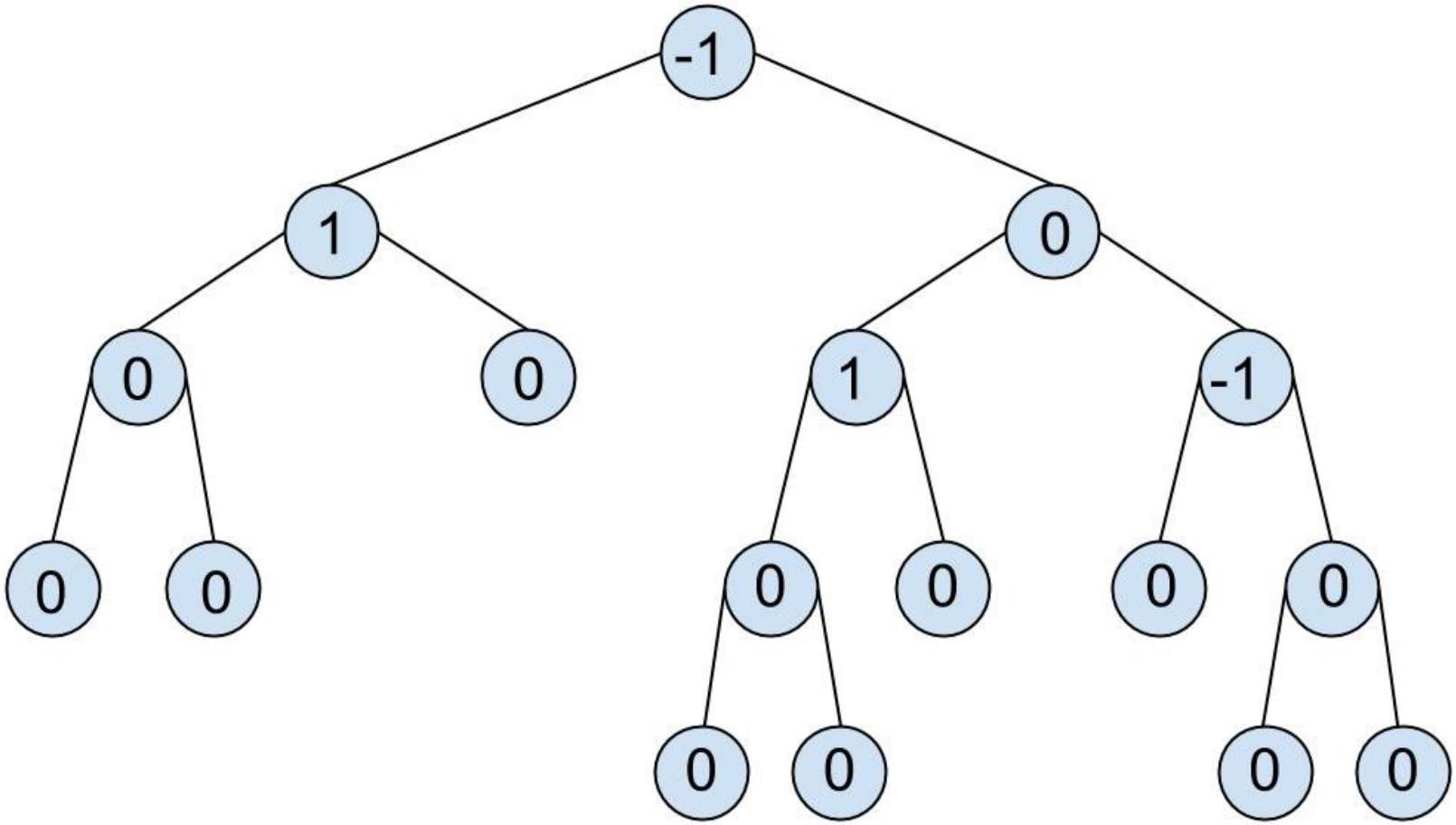
Árvores AVL

- 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

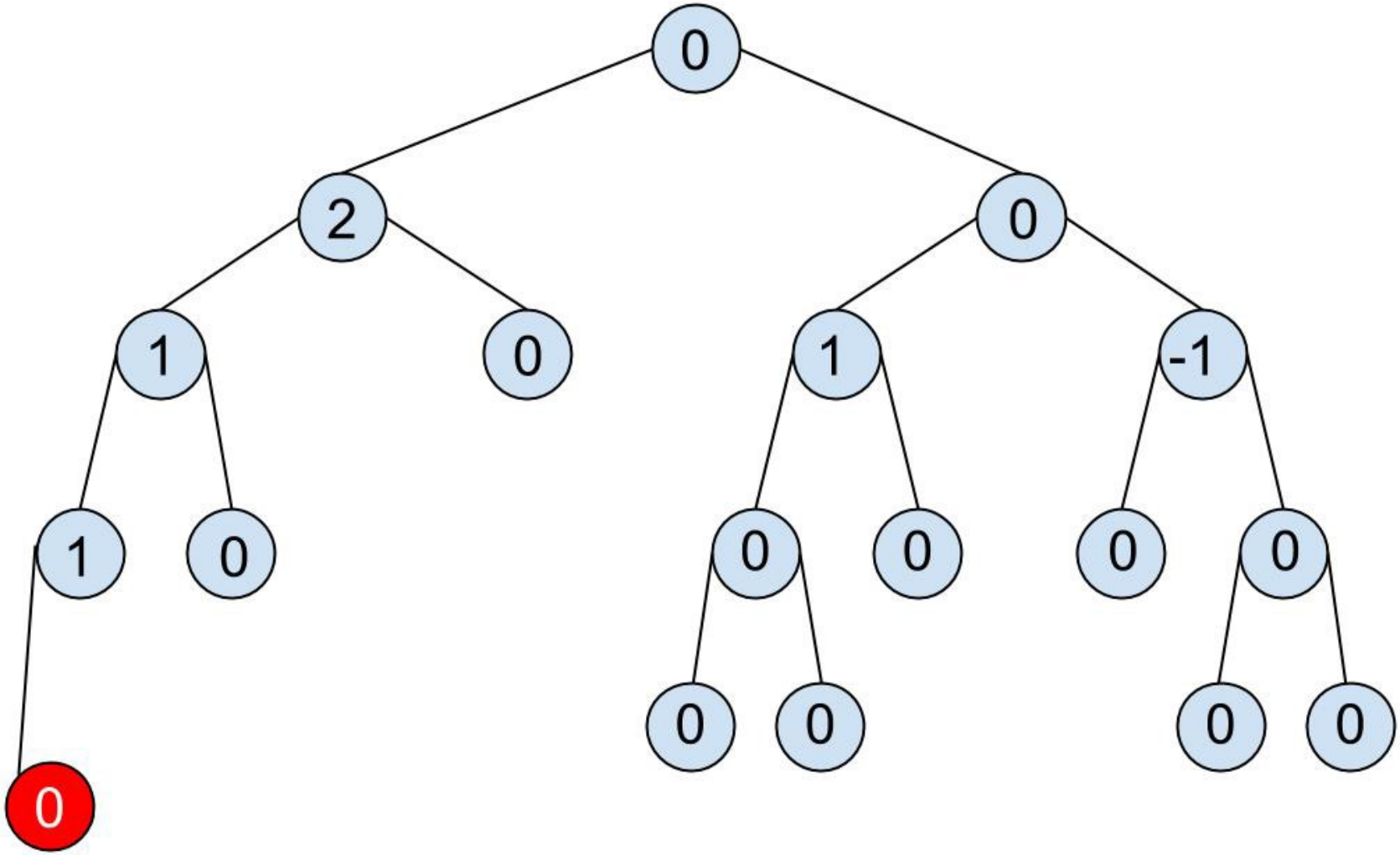
Árvores AVL

- 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

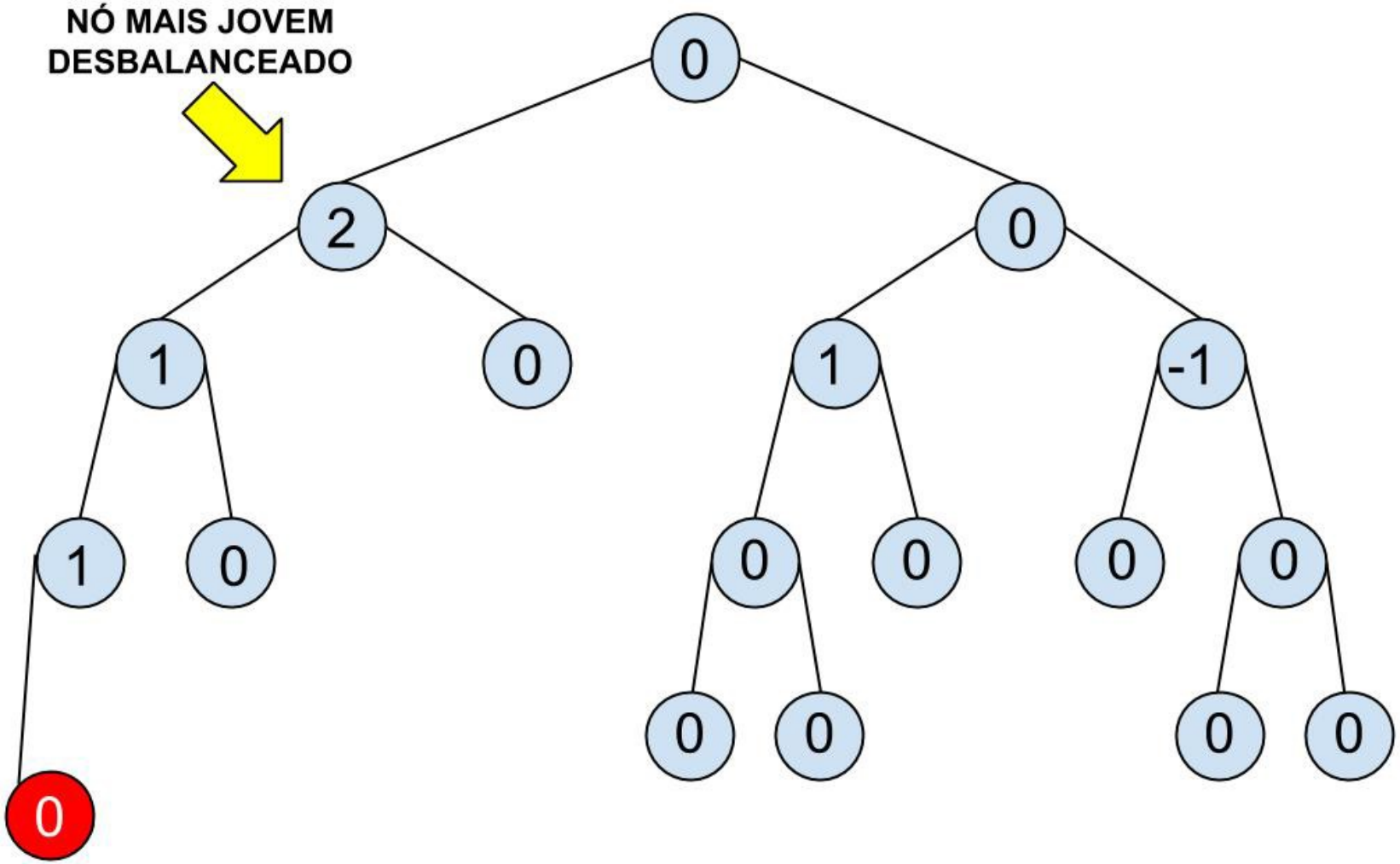
Árvores AVL - Rotação Direita



Árvores AVL - Rotação Direita

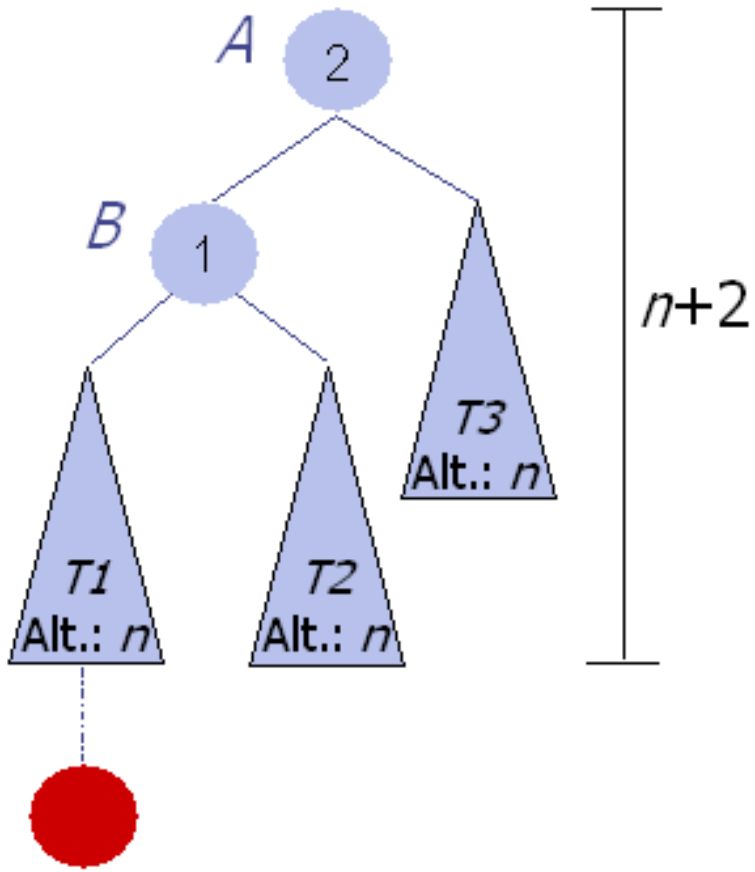


Árvores AVL - Rotação Direita



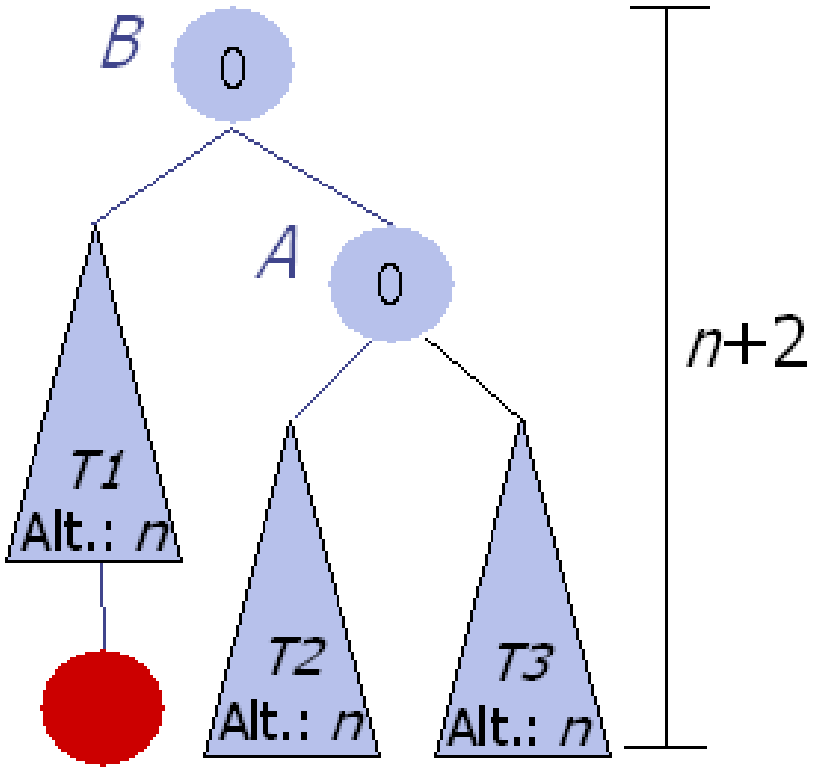
Árvores AVL - Rotação Direita

- 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

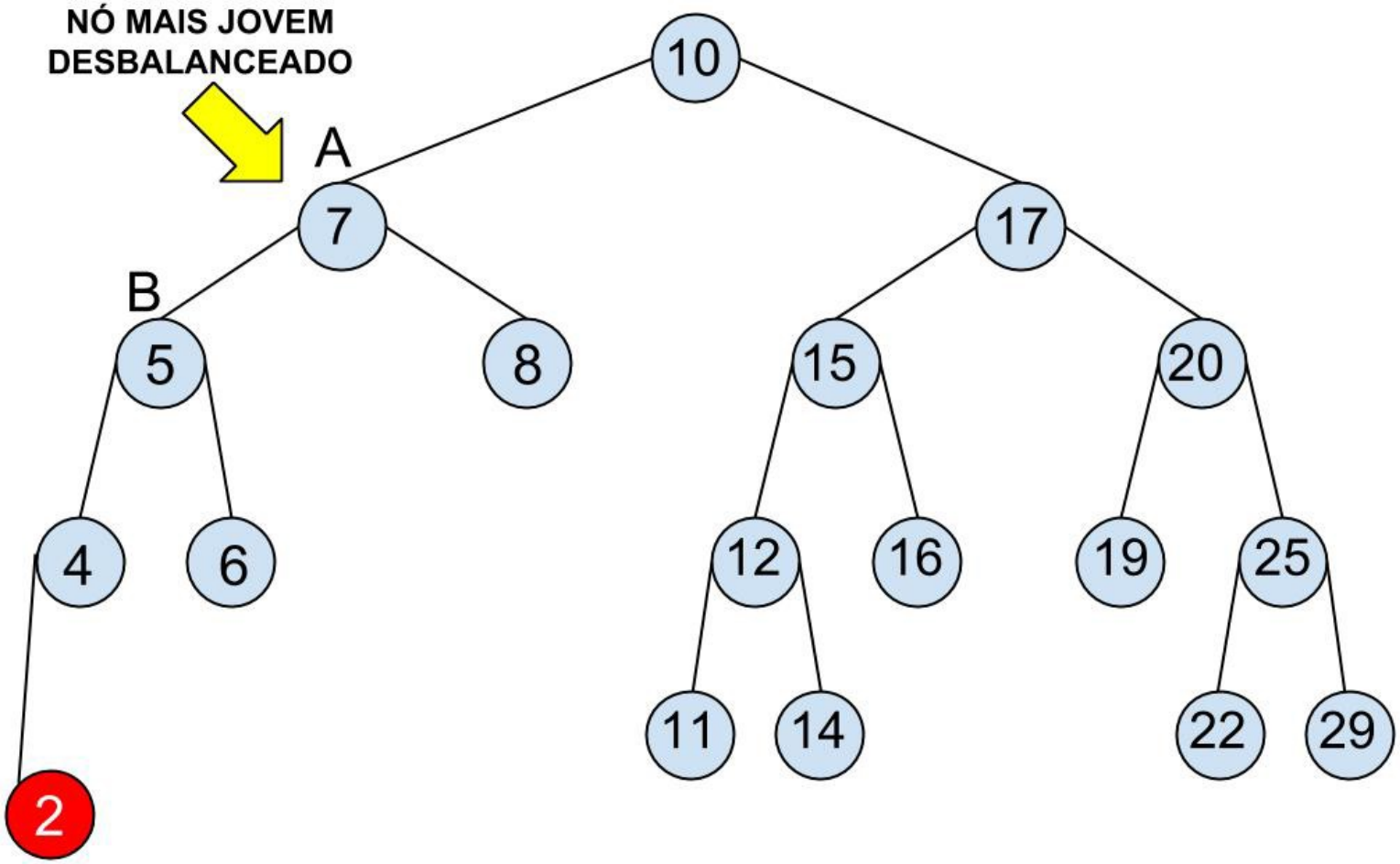


Árvores AVL - Rotação Direita

- 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

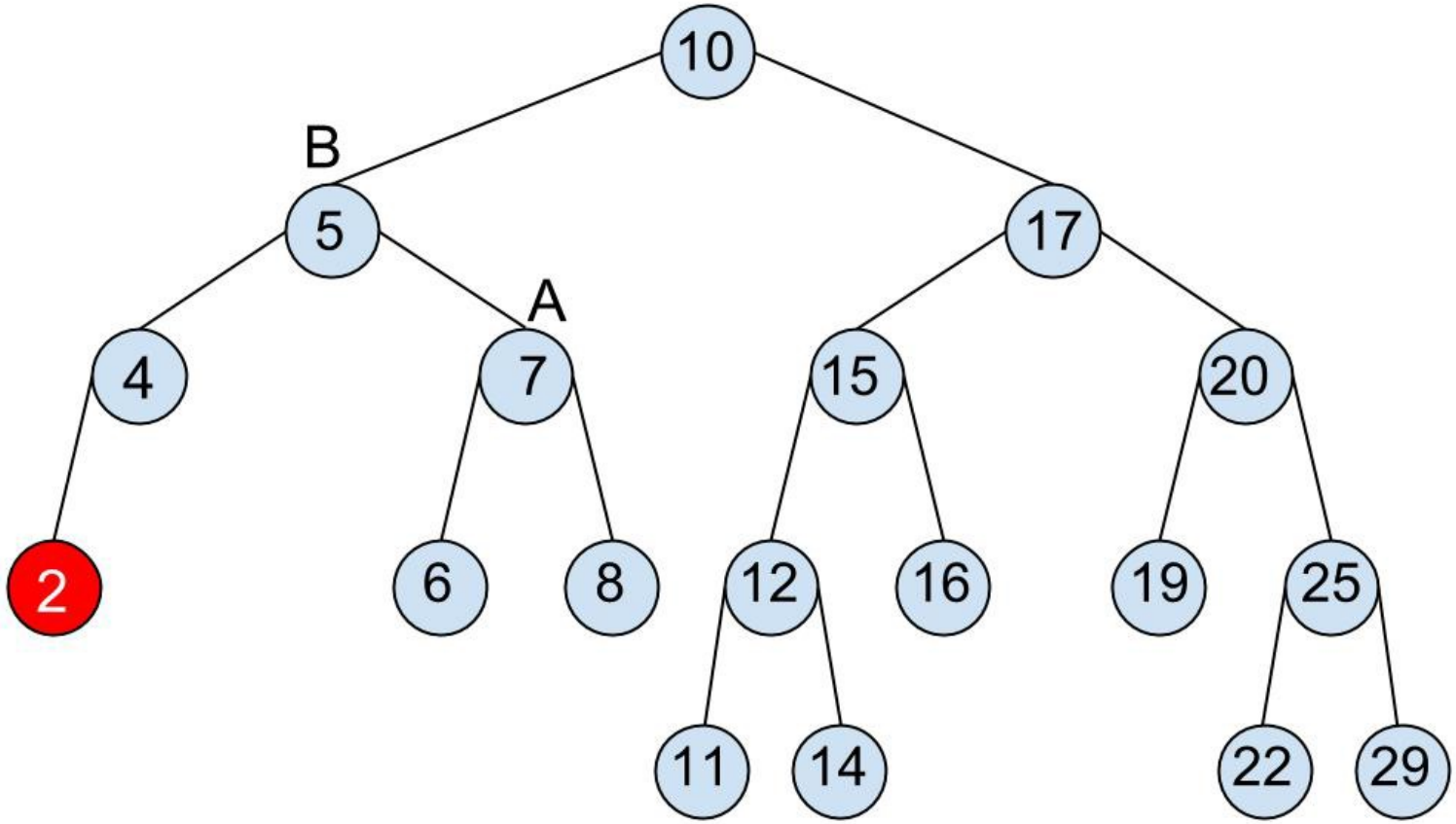


Árvores AVL - Rotação Direita



ANTES DA ROTAÇÃO DIREITA

Árvores AVL - Rotação Direita

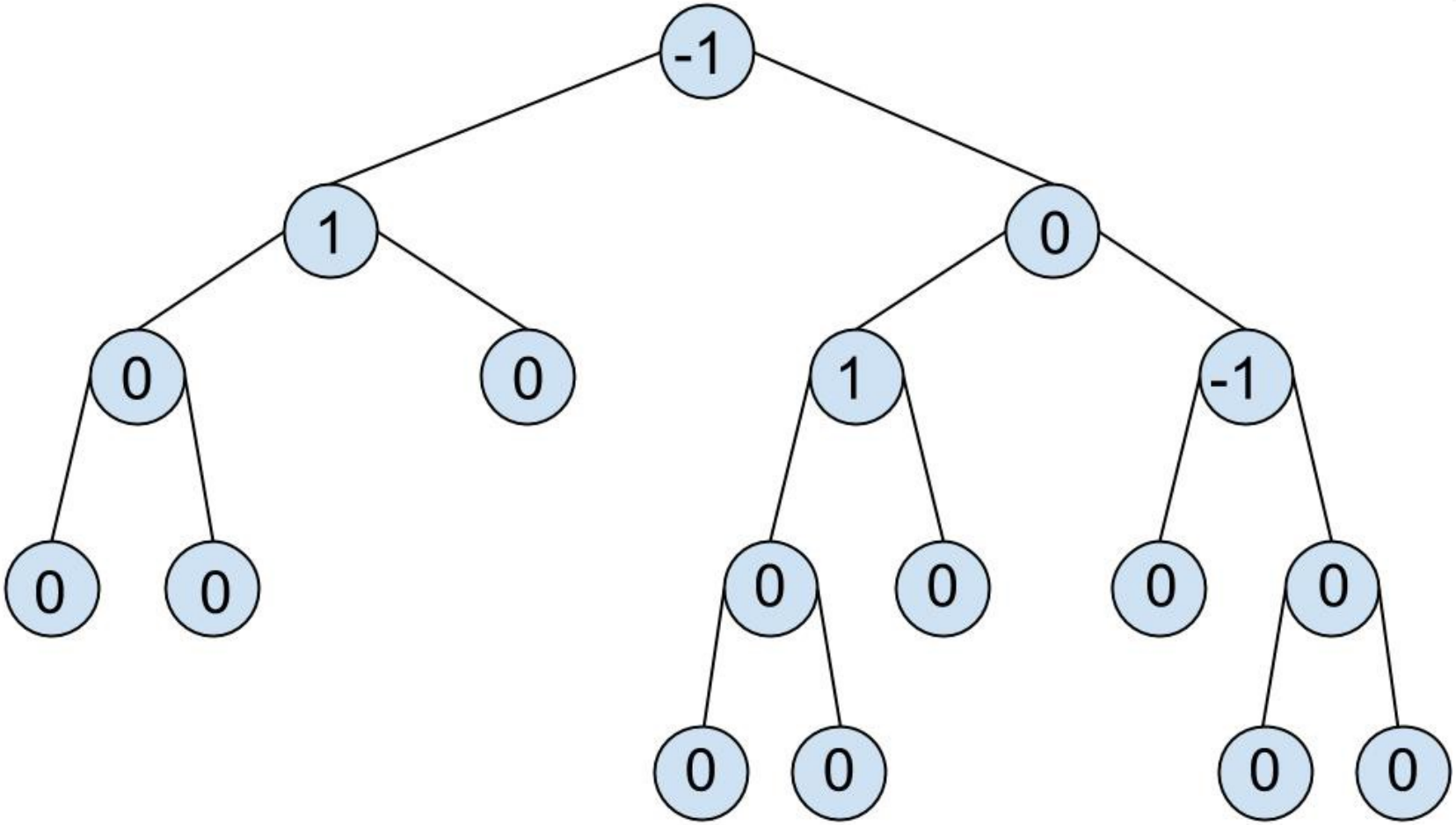


APÓS A ROTAÇÃO DIREITA

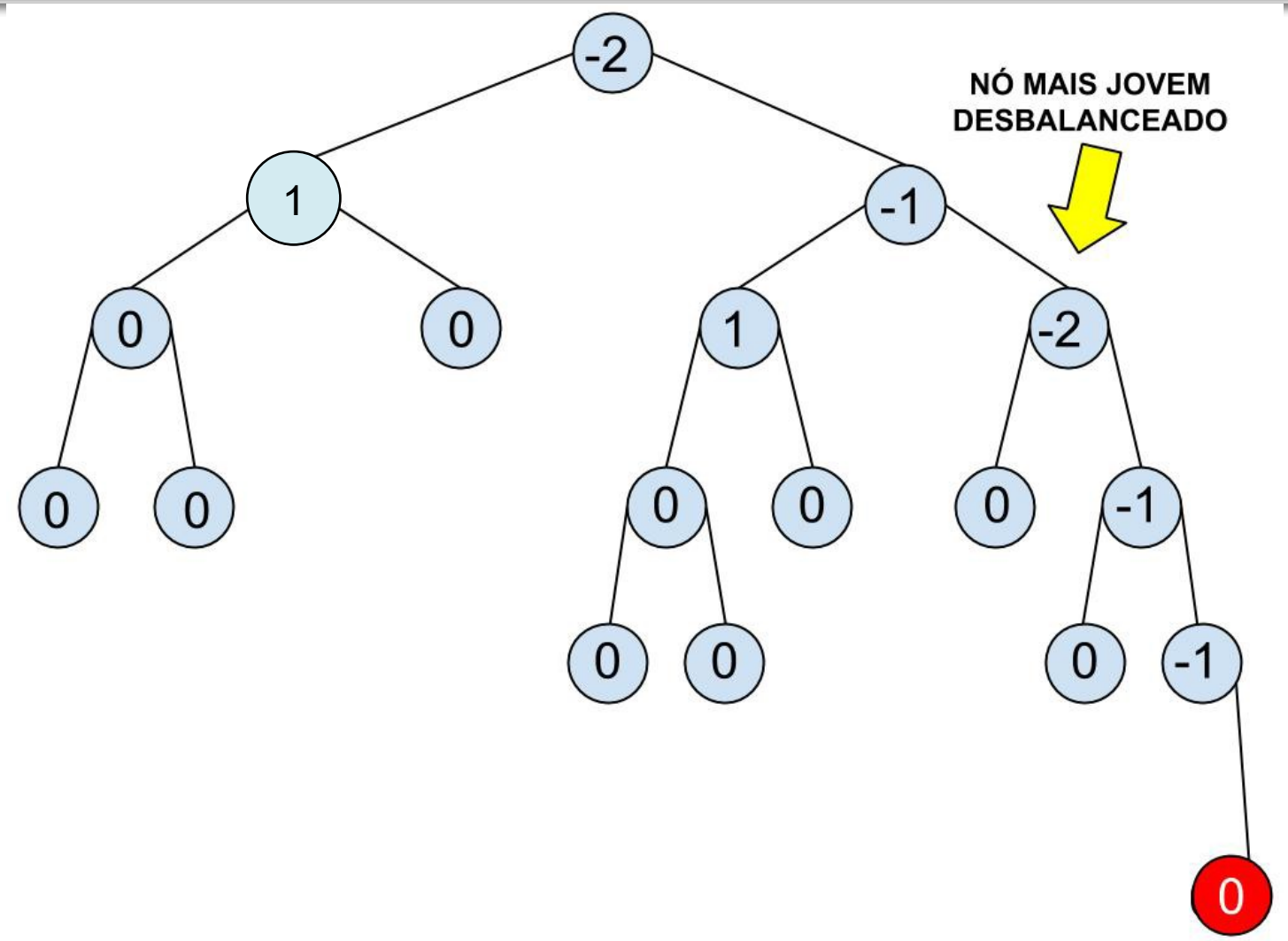
Árvores AVL - Rotação Direita

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

Árvores AVL - Rotação Esquerda

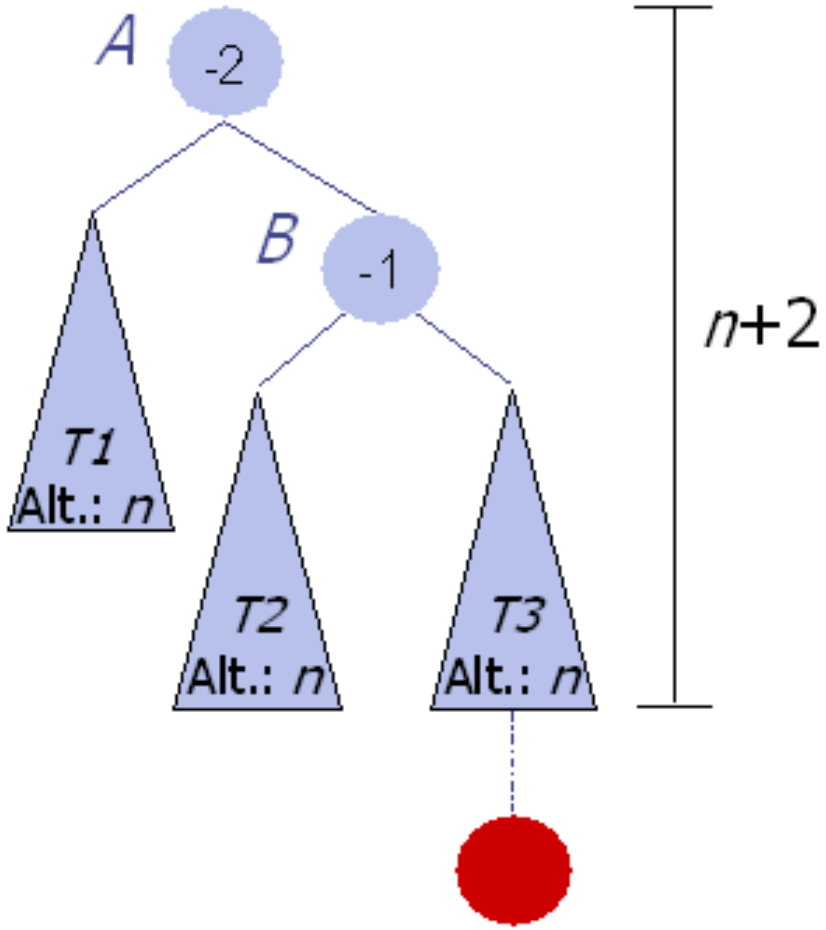


Árvores AVL - Rotação Esquerda



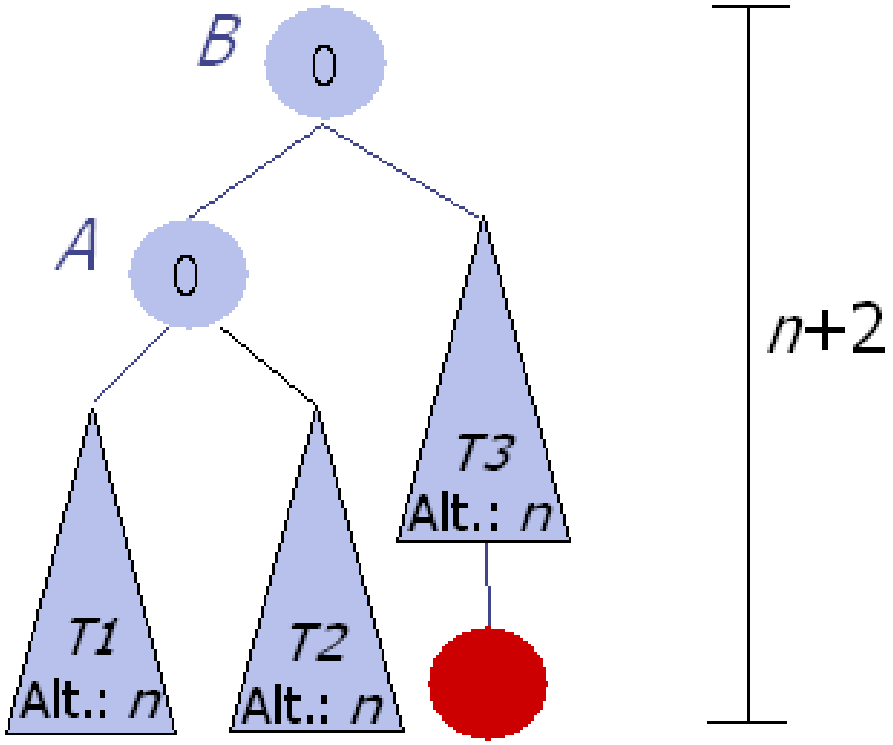
Árvores AVL - Rotação Esquerda

- 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

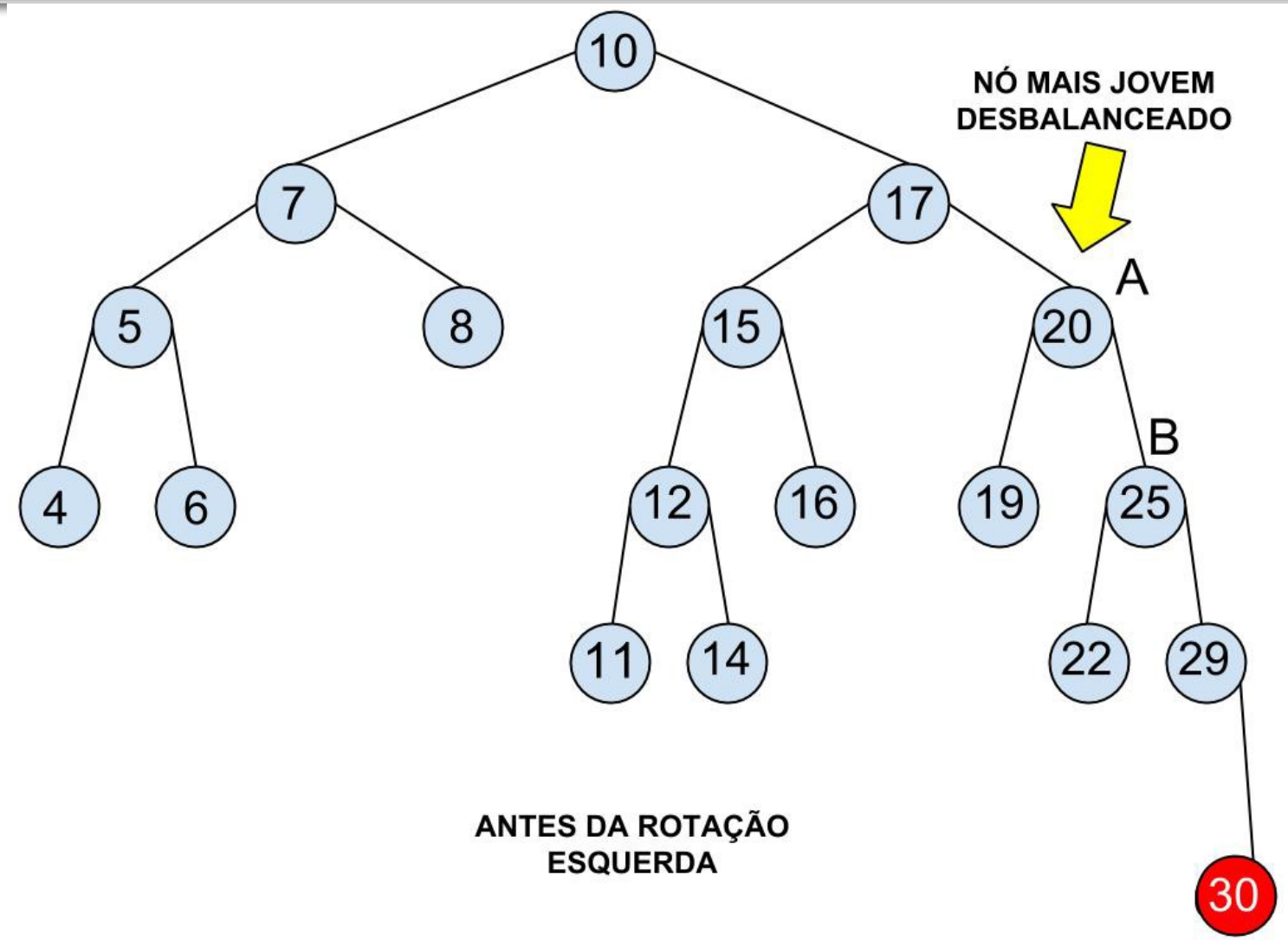


Árvores AVL - Rotação Esquerda

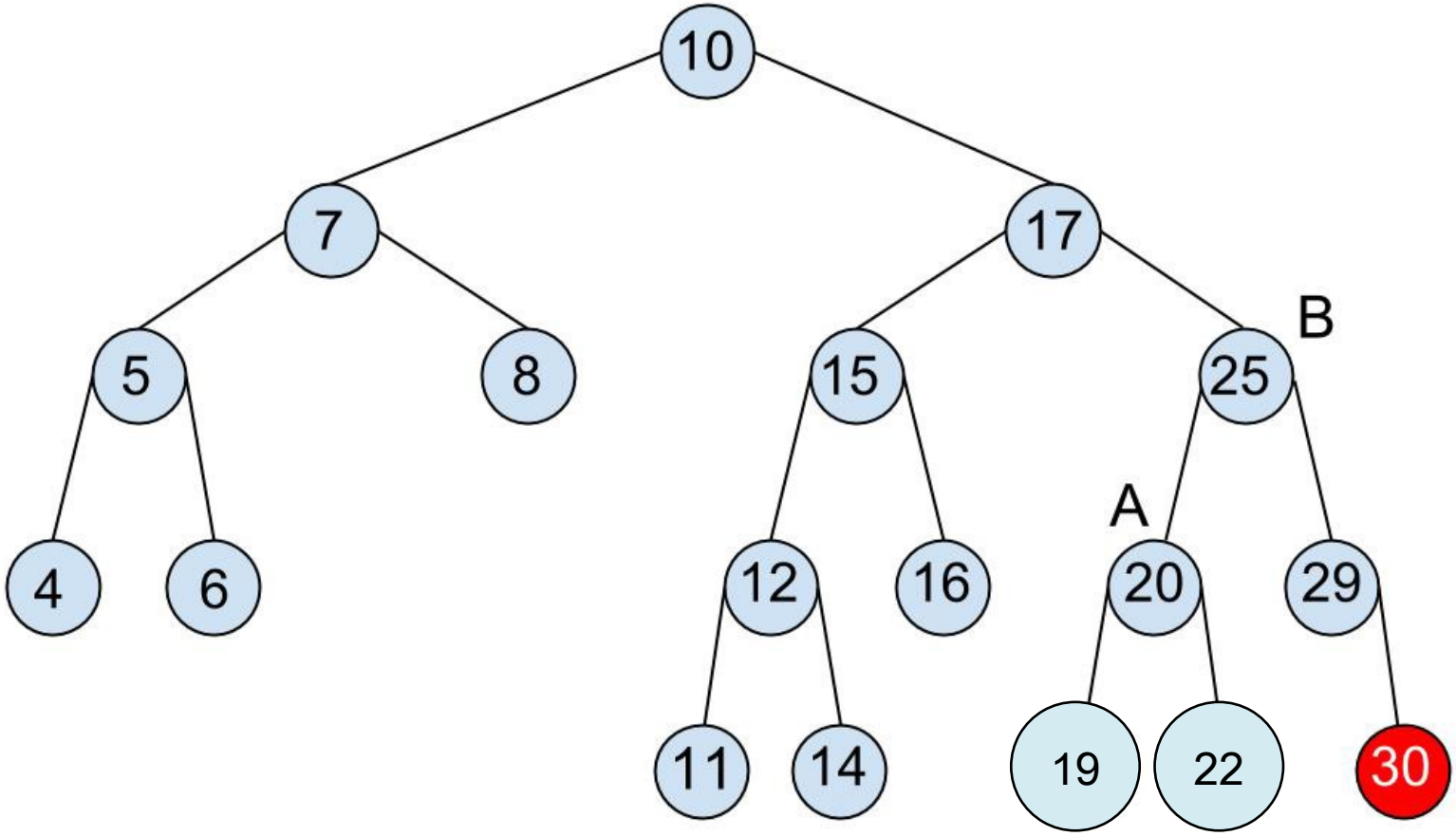
- A rotação esquerda tem formato geral ilustrado à direita
- T1, T2 e T3 podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado



Árvores AVL - Rotação Esquerda



Árvores AVL - Rotação Esquerda



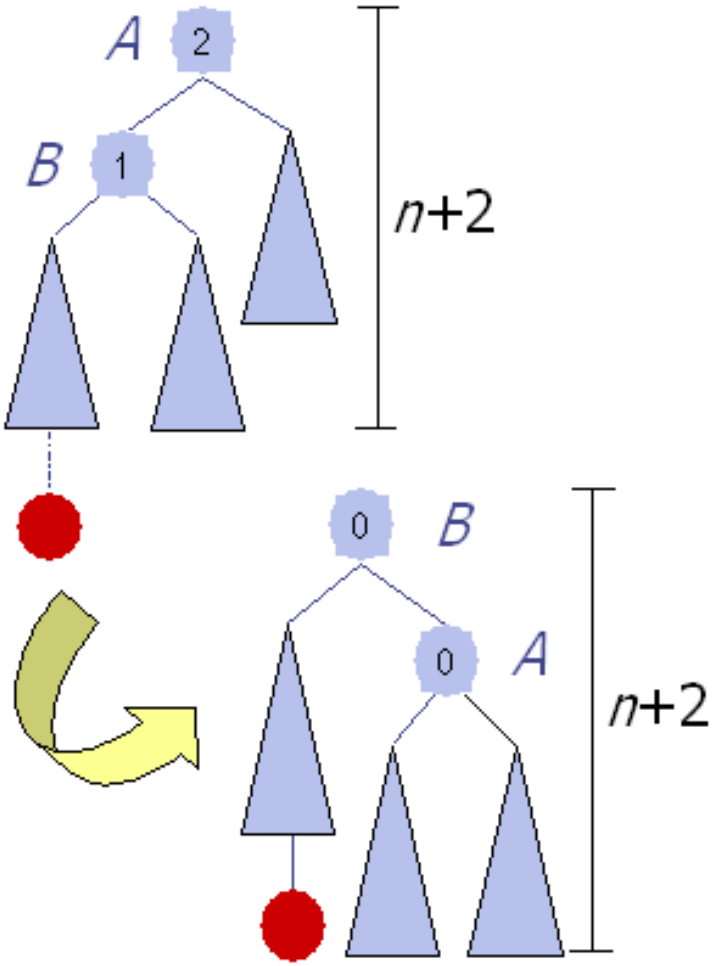
APÓS A ROTAÇÃO ESQUERDA

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

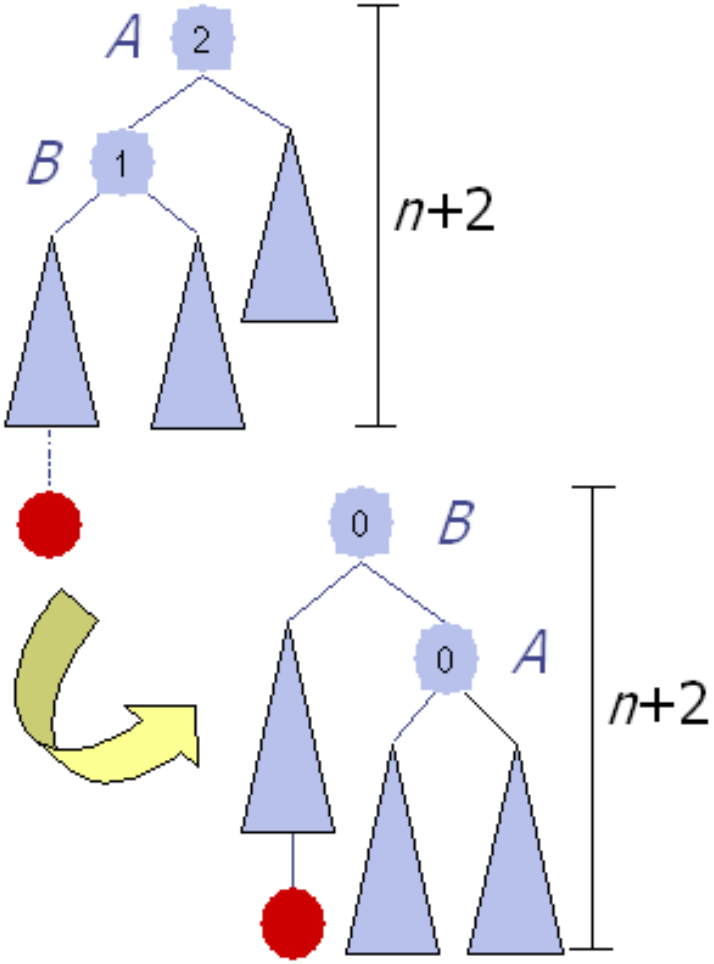
Rotações Simples

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



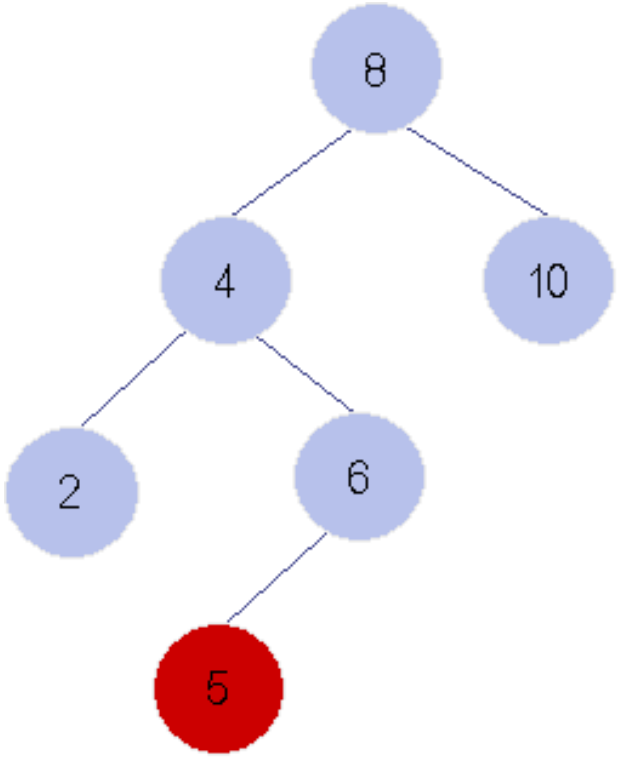
Rotações Simples

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

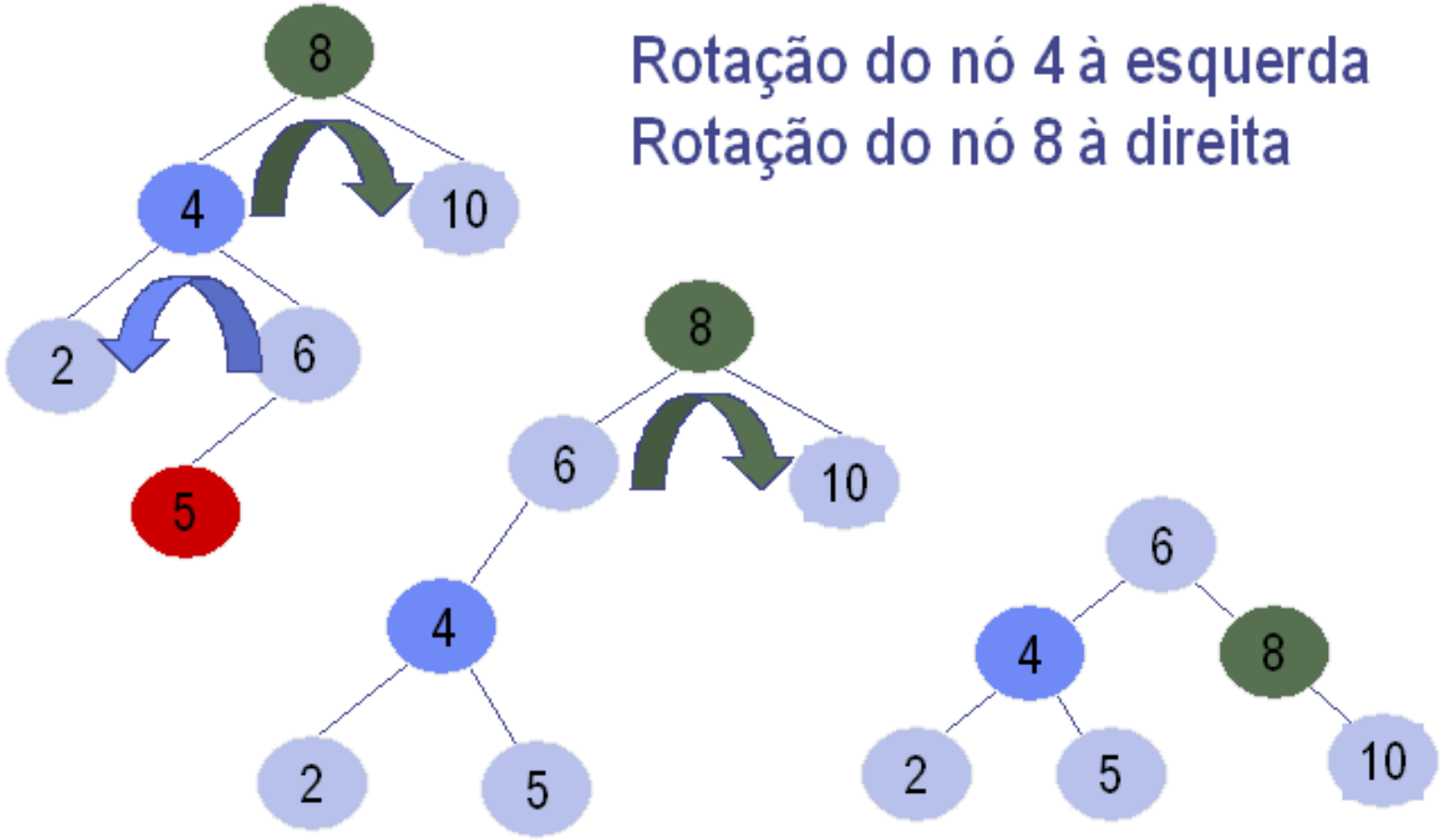


Rotações Duplas

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

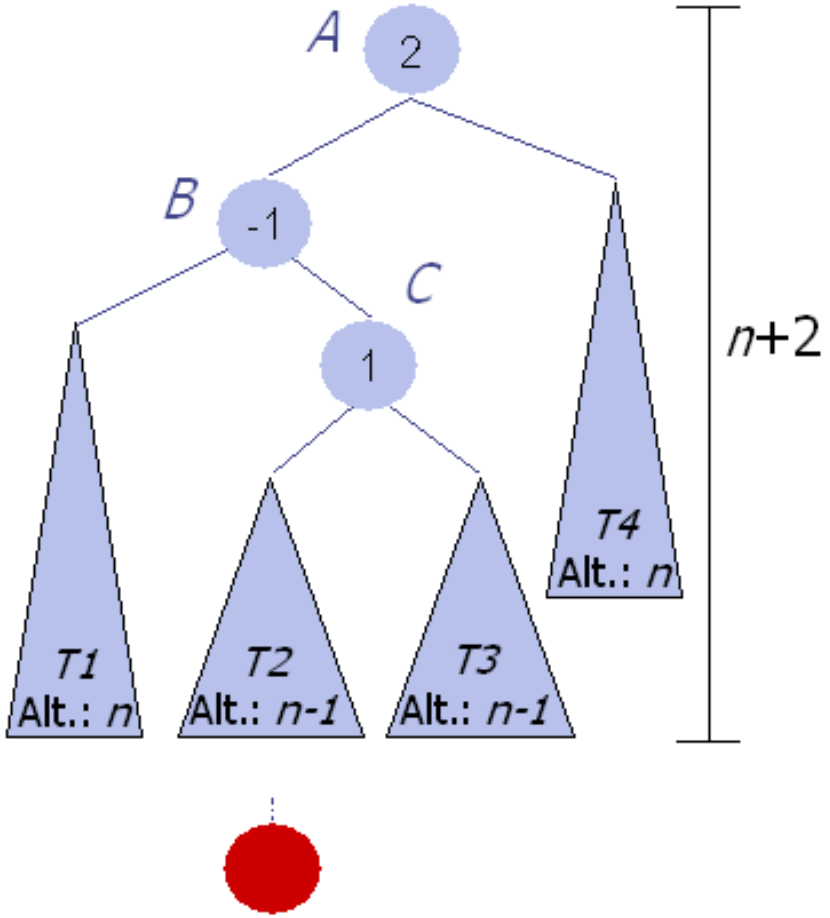


Rotações Duplas



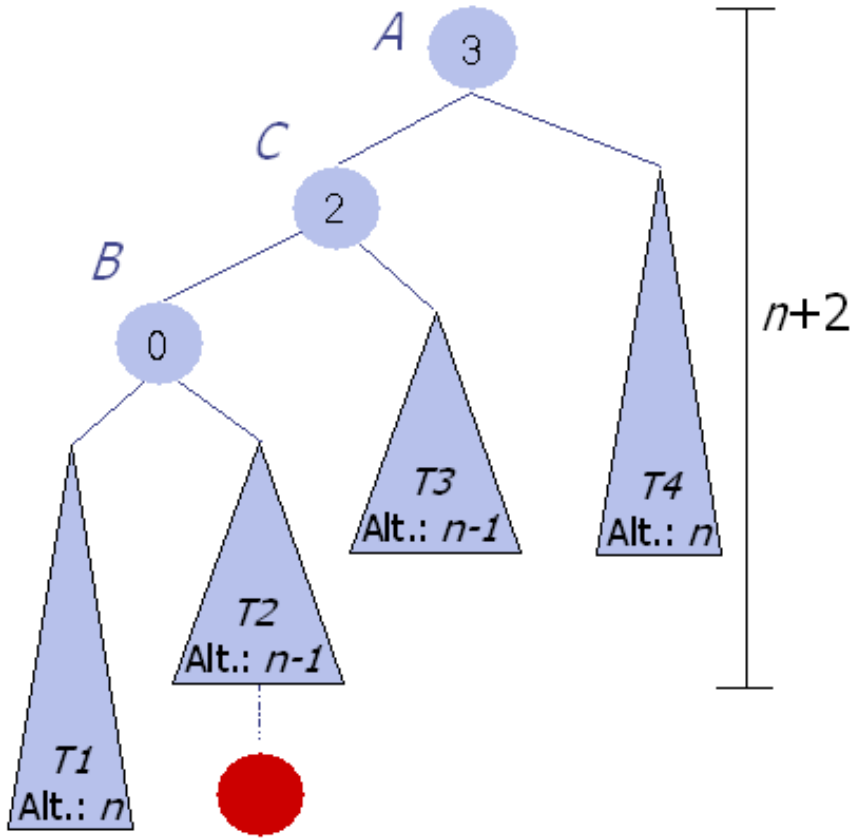
Árvores AVL - Rotação Esq./Dir.

- 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



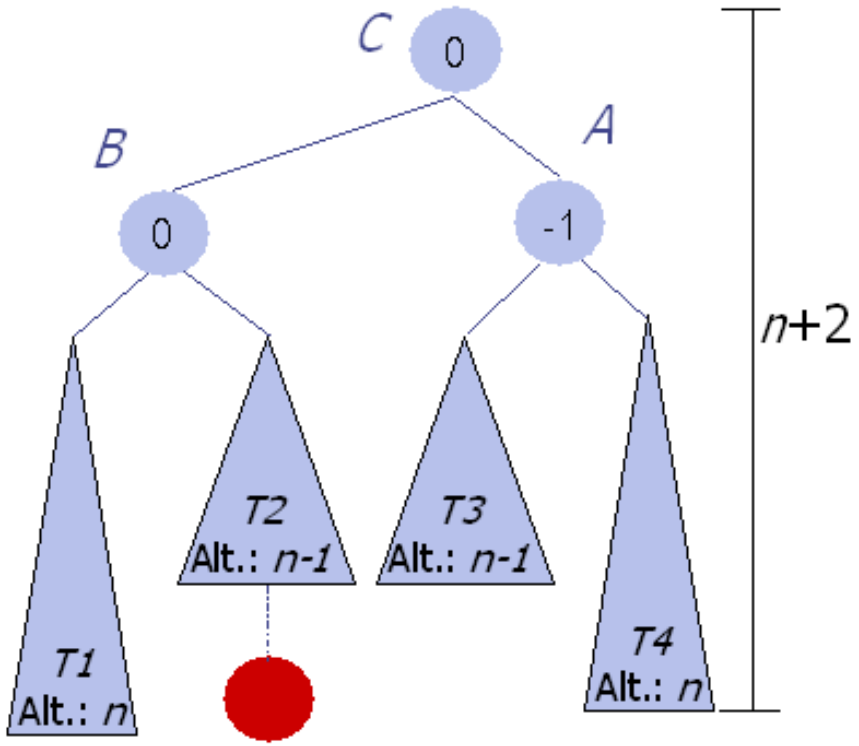
Árvores AVL - Rotação Esq./Dir.

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



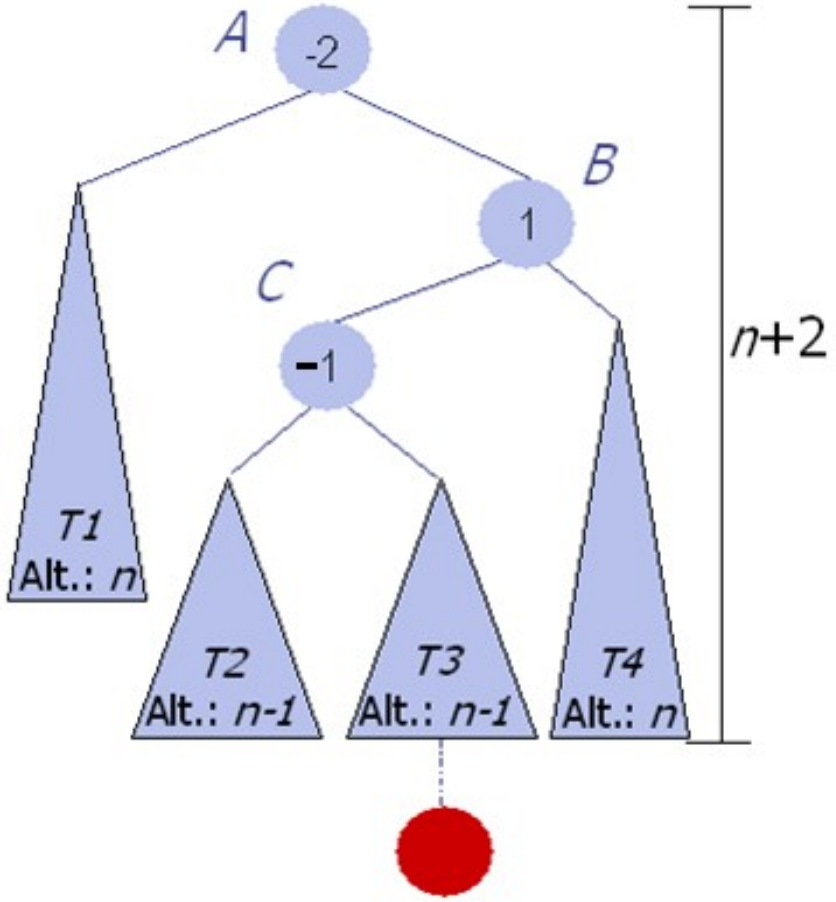
Árvores AVL - Rotação Esq./Dir.

- 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



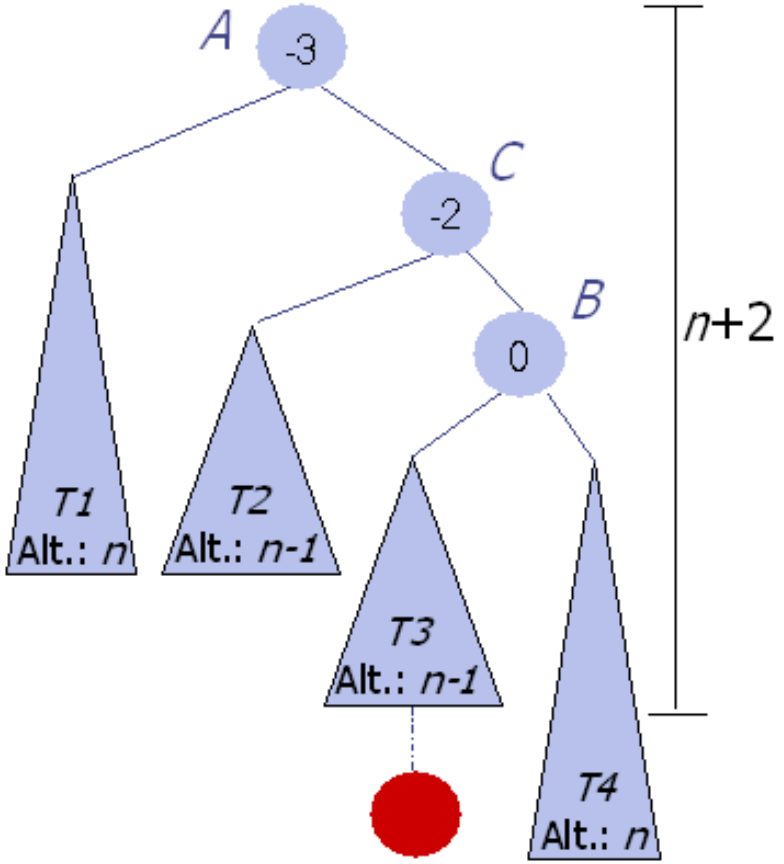
Árvores AVL - Rotação Dir./Esq.

- 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



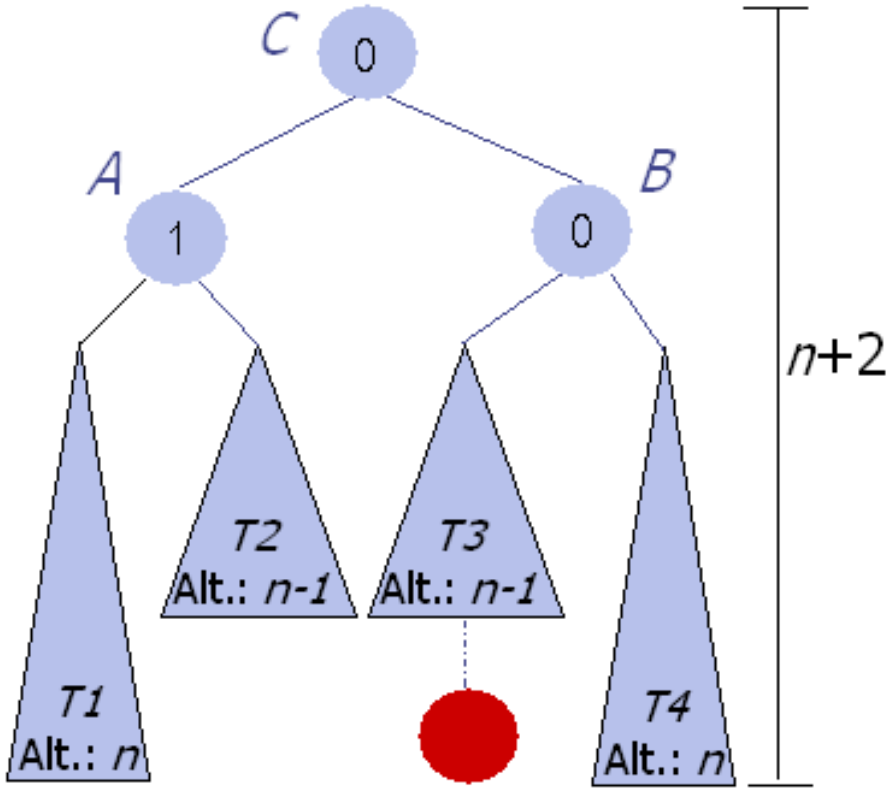
Árvores AVL - Rotação Dir./Esq.

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



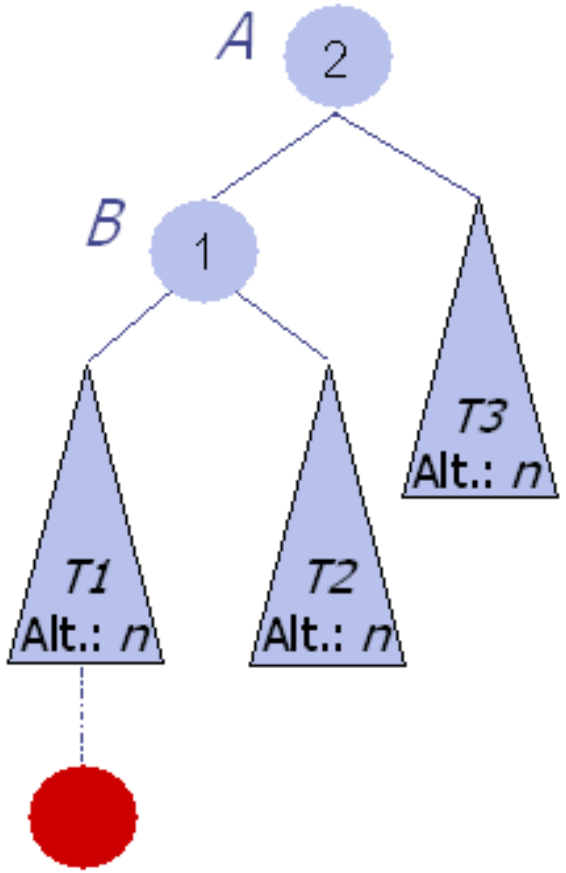
Árvores AVL - Rotação Dir./Esq.

- 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



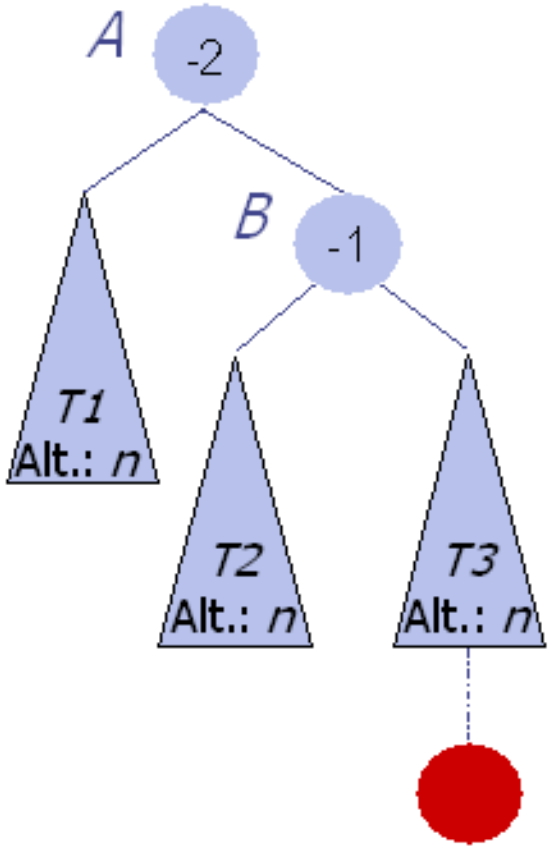
Como decidir qual rotação usar?

- 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



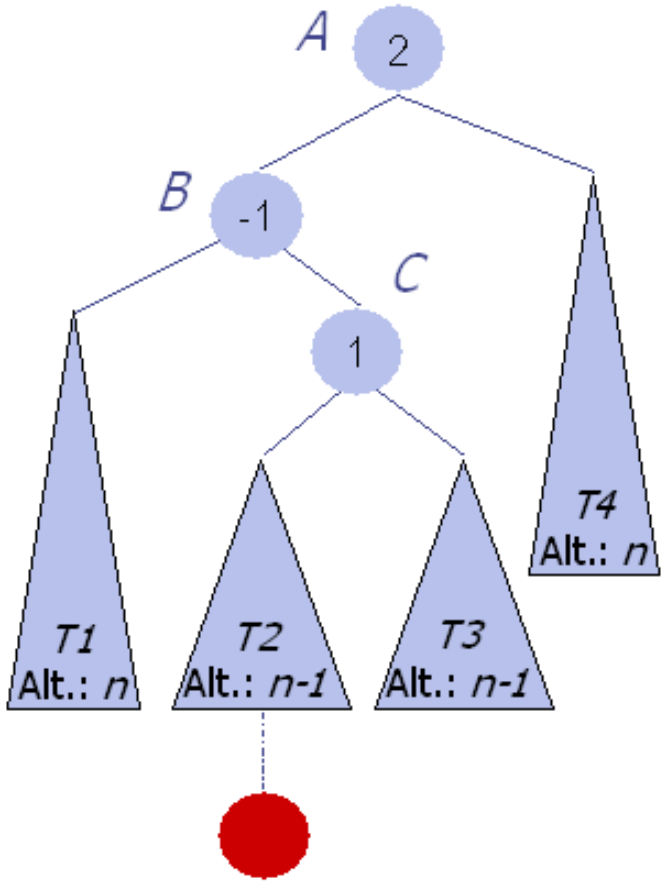
Como decidir qual rotação usar?

- 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



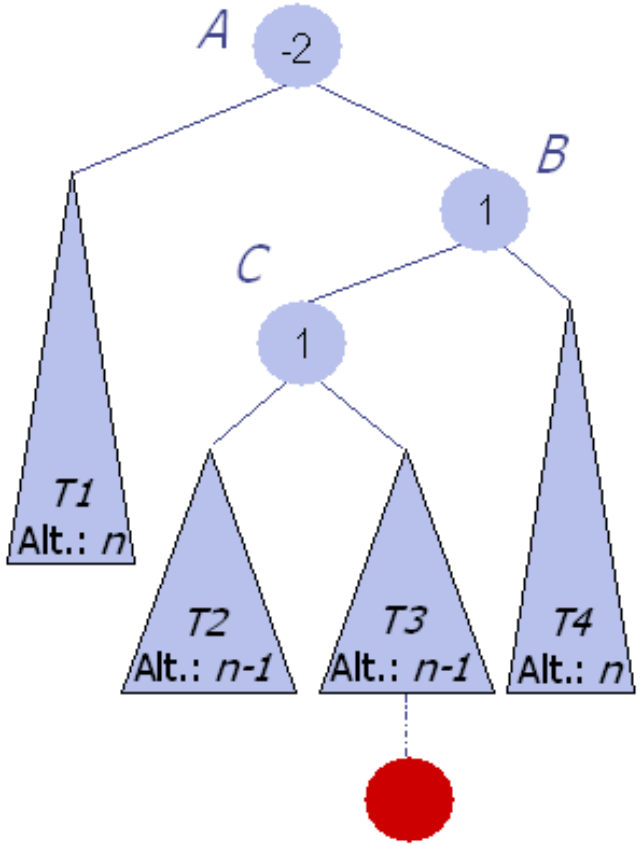
Como decidir qual rotação usar?

- 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



Definição de Tipos

```
(.h)
 #define max(a, b) ((a > b) ? a : b)
 typedef struct avl AVL;
(.c)
1  #include “avl.h”
2  typedef struct no NO;
3  struct no {
4      ITEM *item;
5      NO *fesq;
6      NO *fdir;
7      int FB;
8  };
9
10 struct avl {
11     NO *raiz;
12     int profundidade; ...
13};
```

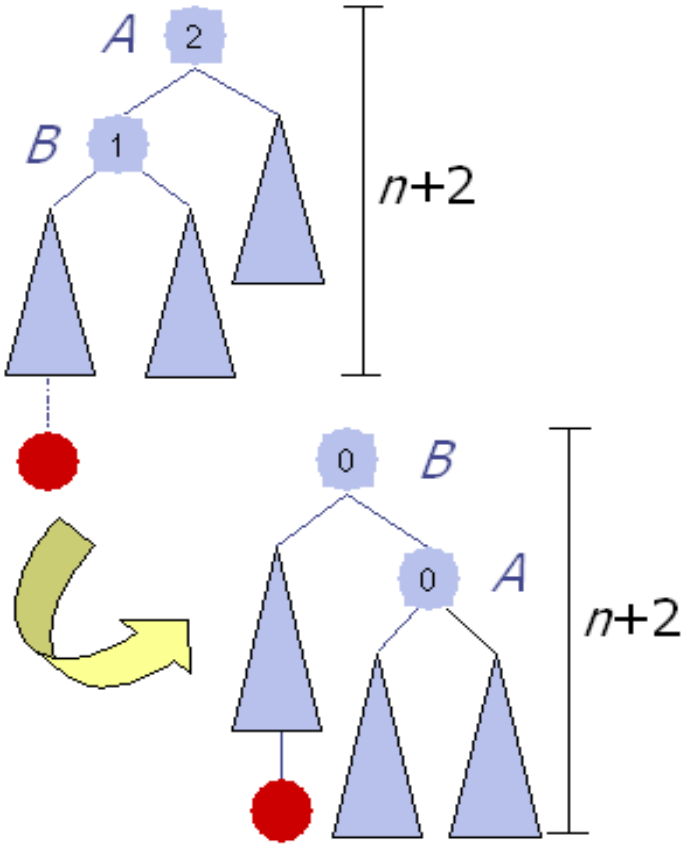
Métodos Básicos

```
1  AVL *avl_criar(void) {
2    AVL *arvore = (AVL *) malloc(sizeof (AVL));
3    if (arvore != NULL) {
4      arvore->raiz = NULL; arvore->profundidade = -1;
5    }
6    return arvore;
7  }
8
9  void avl_apagar_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 avl_apagar(AVL **arvore) {
19   avl_apagar_aux((*arvore)->raiz);
20   free(*arvore);
21   *arvore = NULL;
22 }
```

Métodos Auxiliares

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

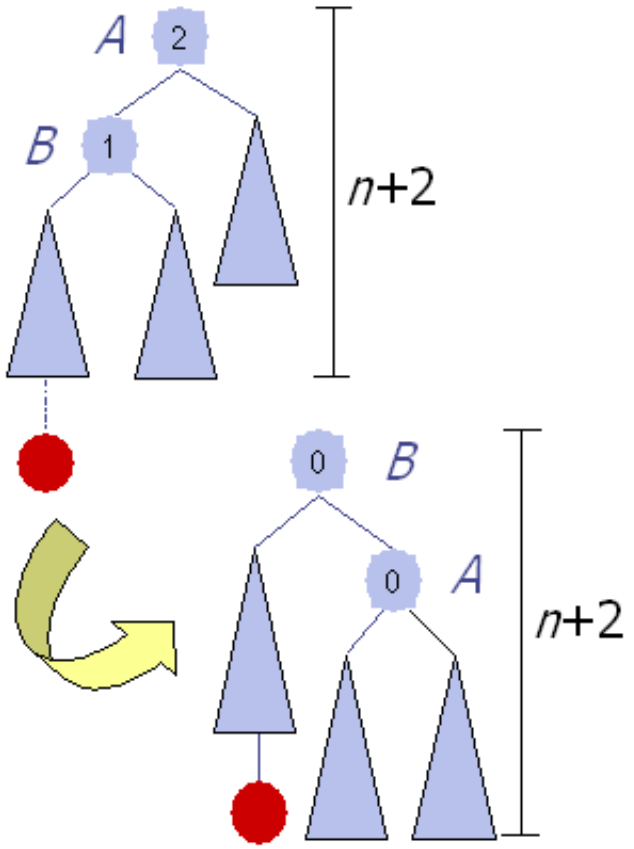
Algoritmo - Rotação Direita



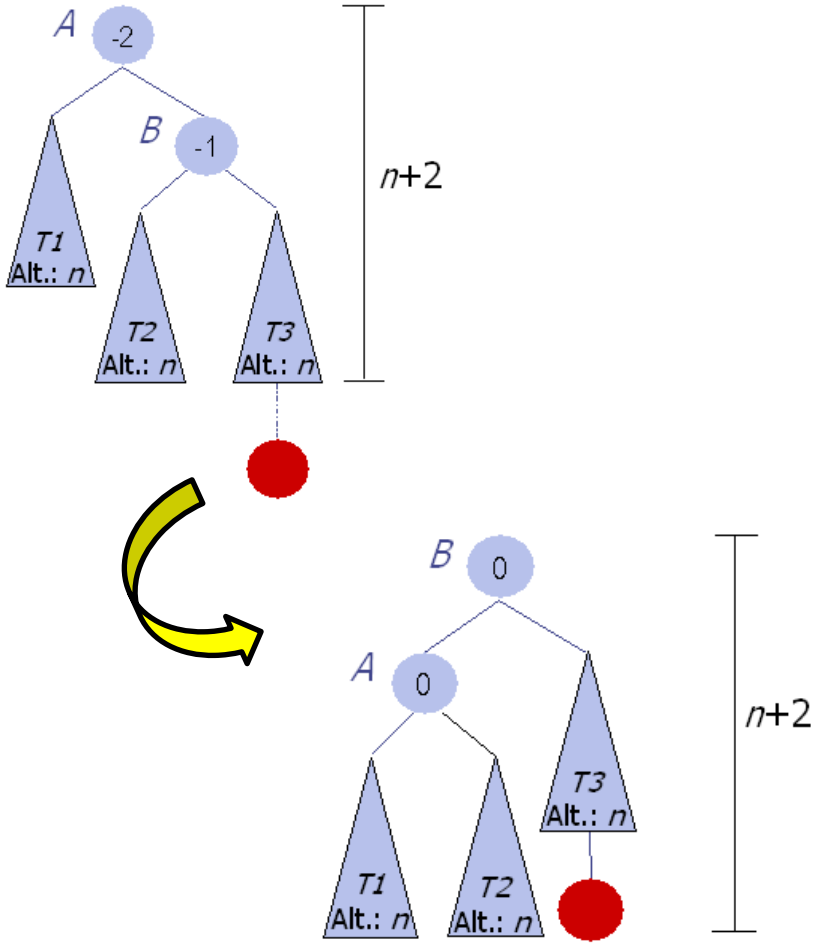
Algoritmo - Rotação Direita

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

  a->FB = b->FB = 0;
  /*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;
}
```



Algoritmo - Rotação Esquerda

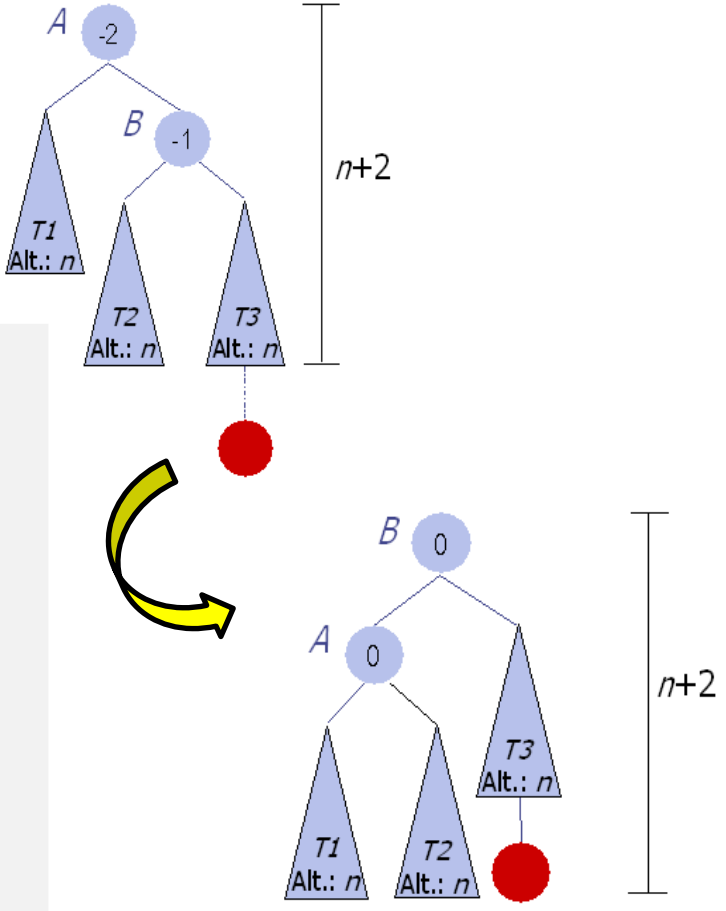


Algoritmo - Rotação Esquerda

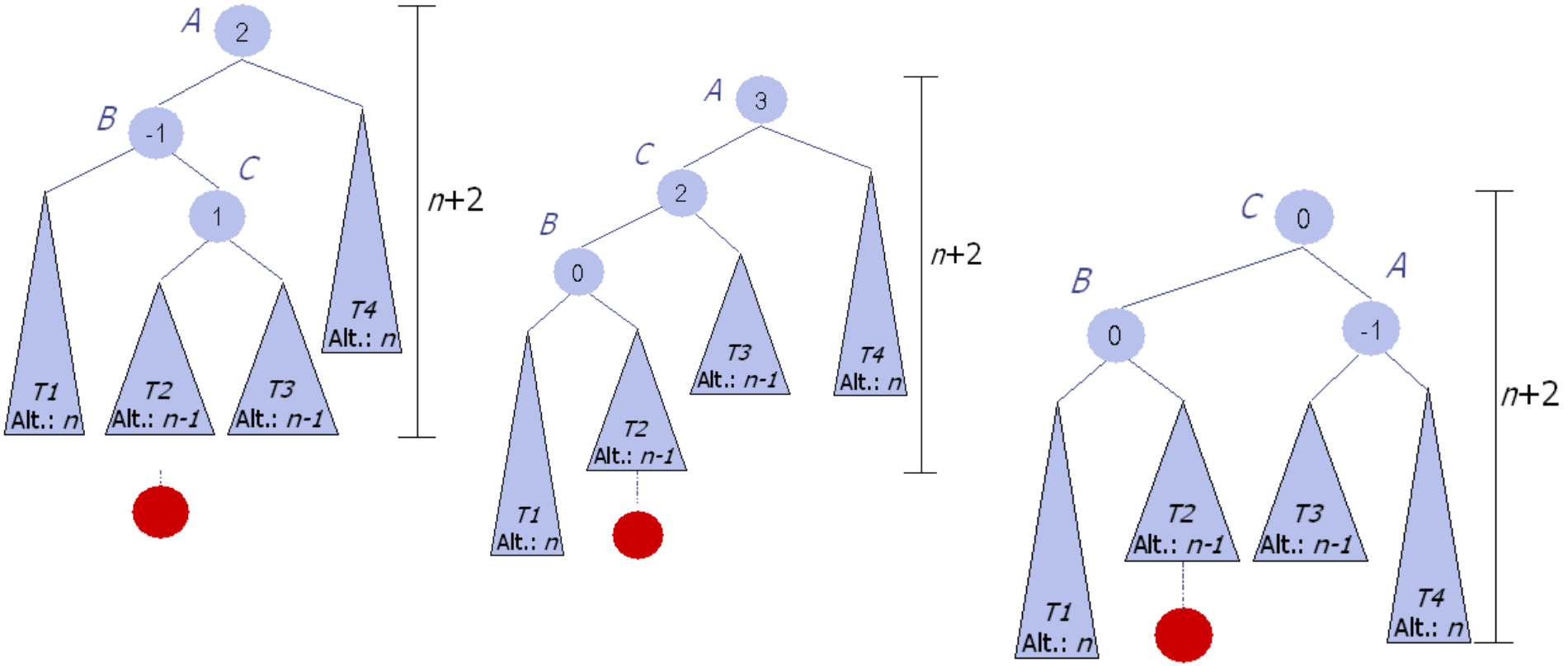
```
NO *rodar_esquerda(NO *a) {
  NO *b = a->fdir;
  a->fdir = b->fesq;
  b->fesq = a;

  a->FB = b->FB = 0;

  /*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;
}
```

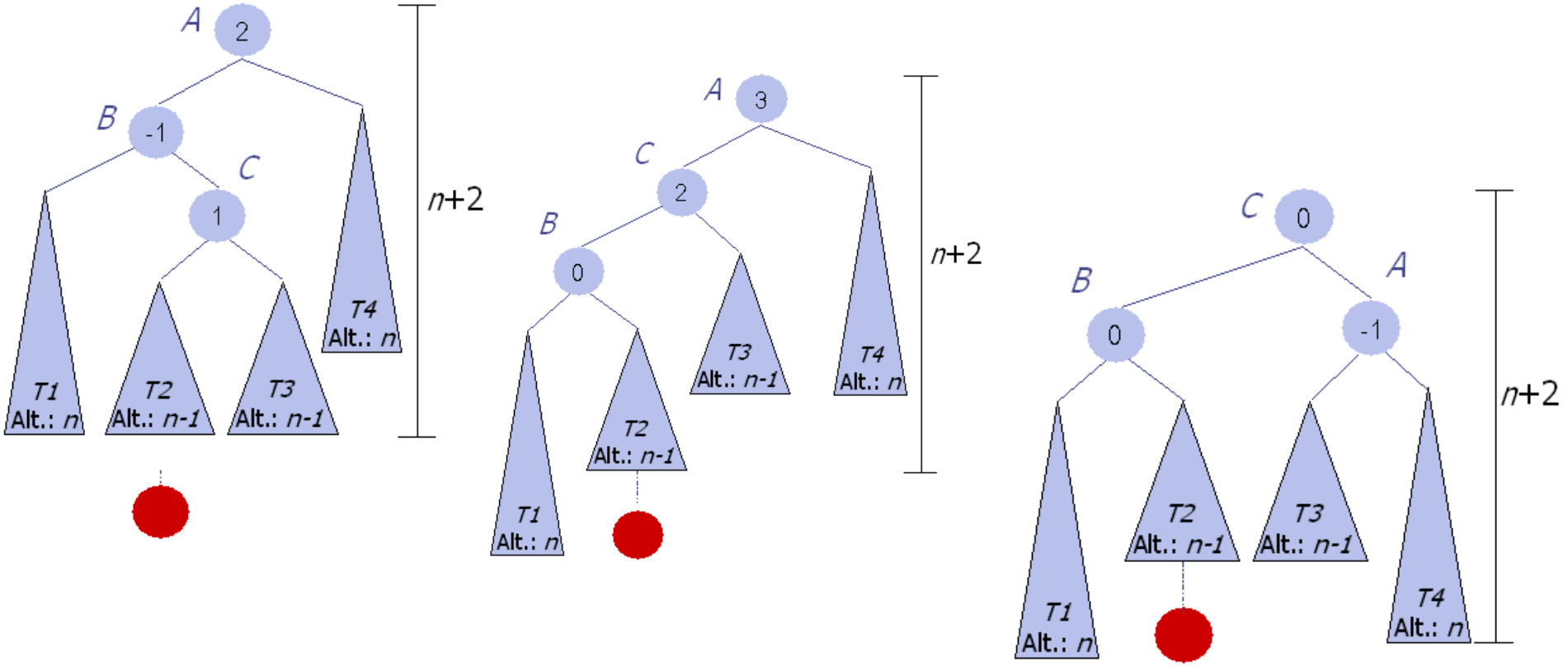


Algoritmo - Rotação Esq./Dir.



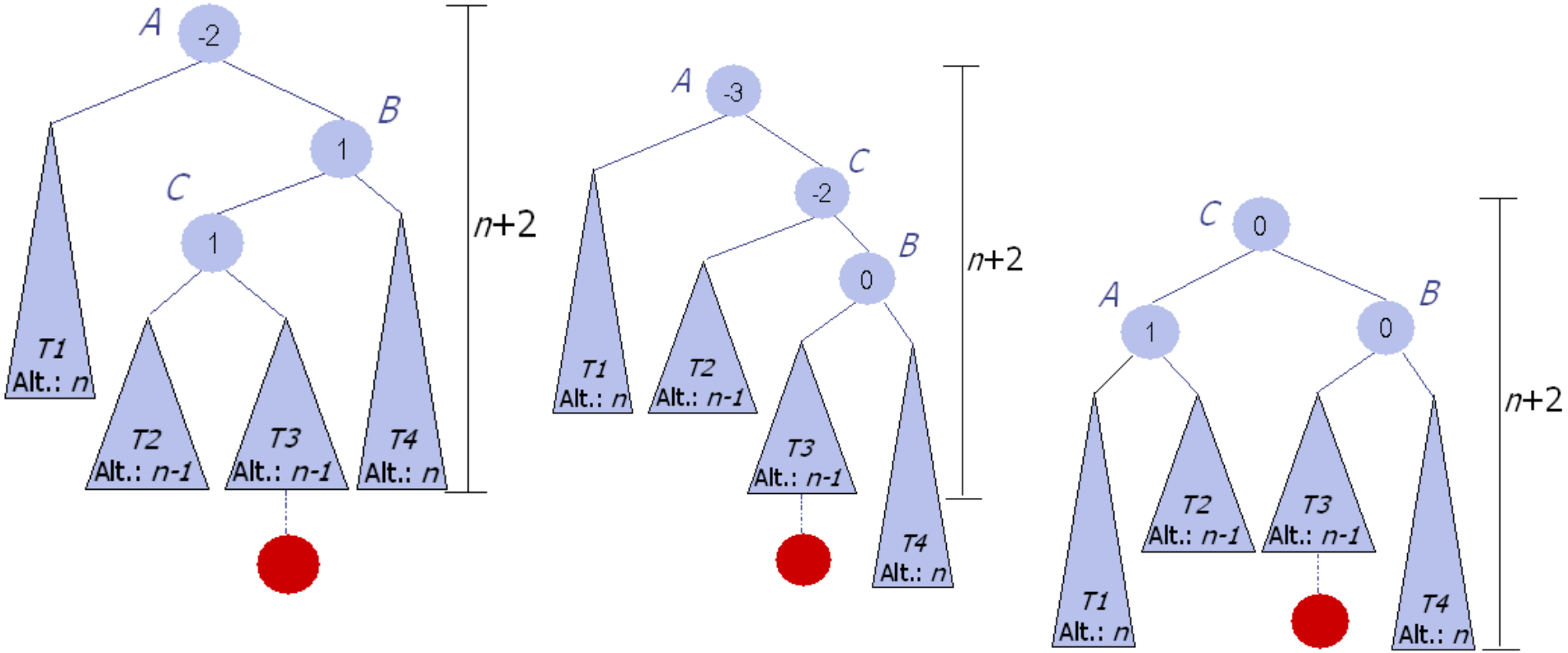
Algoritmo - Rotação Esq./Dir.

```
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->dir = rodar_direita(a->dir);
3   return rodar_esquerda(a);
4 }
```



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 *avl_inserir_no(NO *raiz, ITEM *item) {
2    if (raiz == NULL)
3       raiz = avl_cria_no(item);
4    else if (item_chave(item) < item_chave(raiz->item))
5       raiz->fesq = avl_inserir_no(raiz->fesq, item);
6    else if (item_chave(item) > item_chave(raiz->item))
7       raiz->fdir = avl_inserir_no(raiz->fdir, item);
8
9
10   raiz->FB = (avl_altura2(raiz->esq)) - (avl_altura2(raiz->dir));
11
12   return raiz;
13 }
14
15 bool avl_inserir(AVL *arvore, ITEM *item) {
16   return ((arvore->raiz = avl_inserir_no(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 pode ser verificado:
  - com base na altura das sub-árvores - cada nó armazena sua altura e daí calcula-se o FB; ou
  - com base no fator de balanceamento – cada nó armazena seu FB.

Algoritmo de Inserção

- Se FB = -2 as rotações podem ser
  - Esquerda
  - Direita/Esquerda
- Se FB do filho direito (B) é negativo, rotação Esquerda, caso contrário rotação Direita/Esquerda
- Se FB = 2 as rotações podem ser
  - Direita
  - Esquerda/Direita
- Se FB do filho esquerdo (B) é positivo, rotação Direita, caso contrário rotação Esquerda/Direita

```
FB: avl_altura_no(no->fesq) – avl_altura_no(no->fdir)
```

Algoritmo de Inserção

```
1  NO *avl_inserir_no(NO *raiz, ITEM *item) {
2    if (raiz == NULL)
3       raiz = avl_cria_no(item);
4    else if (item_chave(item) < item_chave(raiz->item))
5       raiz->fesq = avl_inserir_no(raiz->fesq, item);
6    else if (item_chave(item) > item_chave(raiz->item))
7       raiz->fdir = avl_inserir_no(raiz->fdir, item);
8
9   raiz->FB = (avl_altura2(raiz->esq)) - (avl_altura2(raiz->dir));
10
11   if (raiz->FB == -2)
12      if (raiz->dir->FB <= 0)
13         raiz = rodar_esquerda(raiz);
14      else
15         raiz = rodar_direita_esquerda(raiz);
16
17   if (raiz->FB == 2)
18       if (raiz->esq->FB >=0)
19         raiz = rodar_direita(raiz);
20       else
21         raiz = rodar_esquerda_direita(raiz);
22
23   return raiz;
24 }
```

Remoção em AVLs

- Utilizando as rotinas de rotação pode-se definir um algoritmo de remoção em árvores AVL
- A remoção é feita em dois passos
  - o primeiro é uma remoção em ABBs
    - Existem 3 casos possíveis: o nó a ser removido possui grau 0, 1 ou 2.
  - o segundo é o rebalanceamento, se necessário
- O processo é semelhante à inserção

Remoção em AVLs

- Exemplos de remoção

Remoção em AVLs

```
18  bool avl_remover(AVL *T, int chave){
19    return((T->raiz = avl_remover_aux(&T->raiz, chave)) != NULL);
20  }
21
22  NO *avl_remover_aux (NO **raiz, int chave){
23    NO *p;
24
25      if(*raiz == NULL)
26        return (NULL);
27      else if(chave == item_get_chave((*raiz)->item)) {
28        if((*raiz)->esq == NULL || (*raiz)->dir == NULL)
29        {//*Caso 1 se resume ao caso 2: há um filho ou nenhum
30          p = *raiz;
31          if((*raiz)->esq == NULL)
32            *raiz = (*raiz)->dir;
33          else
34            *raiz = (*raiz)->esq;
35
36          item_apagar(&p->item);
37          free(p);
38          p = NULL;
39        }
40        //Caso 3: há ambos os filhos
41        else
42          troca_max_esq((*raiz)->esq, (*raiz), (*raiz));
43      }
44      else if (chave < item_get_chave((*raiz)->item))
45        (*raiz)->esq = avl_remover_aux(&(*raiz)->esq, chave);
46      else if (chave > item_get_chave((*raiz)->item))
47        (*raiz)->dir = avl_remover_aux(&(*raiz)->dir, chave);
```

Remoção em AVLs

```
48
49      if (*raiz != NULL){
50        (*raiz)->FB = avl_altura2((*raiz)->esq) - avl_altura2((*raiz)->dir);
51        if ((*raiz)->FB == 2) {
52              if ((*raiz)->esq->FB >= 0) //FB Filho esq positivo = rot simples
53                  *raiz = rodar_direita(*raiz);
54              else
55                  *raiz = rodar_esquerda_direita(*raiz);
56        }
57        if ((*raiz)->FB == -2) {
58              if ((*raiz)->dir->FB <= 0) //FB Filho dir negativo = rot simples
59                *raiz = rodar_esquerda(*raiz);
60              else
61                *raiz = rodar_direita_esquerda(*raiz);
62        }
63      }
64      return *raiz;
65  }
```

Remoção em AVLs

```
1  void troca_max_esq(NO *troca, NO *raiz, NO *ant)
2  {
3    if(troca->dir != NULL){
4      troca_max_esq(troca->dir, raiz, troca);
5      return;
6    }
7    if(raiz == ant)
8      ant->esq = troca->esq;
9    else
10     ant->dir = troca->esq;
11
12   ITEM* it = rem->item;
13   raiz->item = troca->item;
14   item_apagar(&it);
15   free(troca);
16   troca = NULL;
17 }
```

- Exatamente igual ao algoritmo da ABB!
  - Quase… (aqui o item está sendo removido)

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 completa cheia.
- 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
  - a, z, b, y, c, x
  - a, z, b, y, c, x, d, w, e, v, f
  - a, v, l, t, r, e, i, o, k
  - 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 passo-a-passo a árvore AVL gerada pelas inserções das seguintes chaves na ordem fornecida
  - 10, 5, 20, 1, 3, 4, 8, 30, 40, 35, 50, 45, 55, 51, 100
- Para a AVL gerada, mostre passo-a-passo a remoção das chaves 51, 3, e 40