# MC3305
# Algoritmos e Estruturas de Dados II

## Aula 10 –
## Árvores Adelson-Velskii e Landis

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2014

# Árvores balanceadas

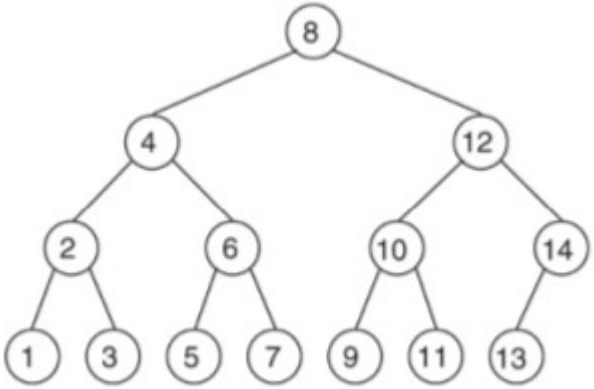
# Árvores balanceadas

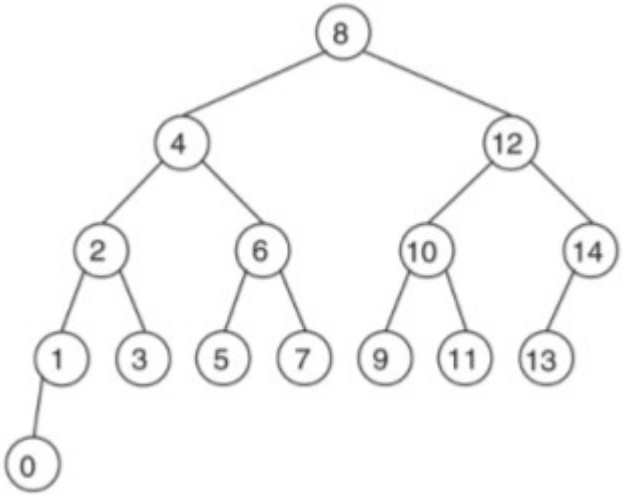
- As ABB permitem buscar de forma eficiente um elemento dada uma chave.
- Deseja-se que o custo de acesso tenha a ordem de grandeza de uma árvore ótima → O(lg(n))
  - Este custo deve-se manter ao longo da utilização da estrutura (inclusive após inserções/remoções).
  - O custo de ter a estrutura balanceada deve estar na, idealmente, mesma ordem de grandeza.

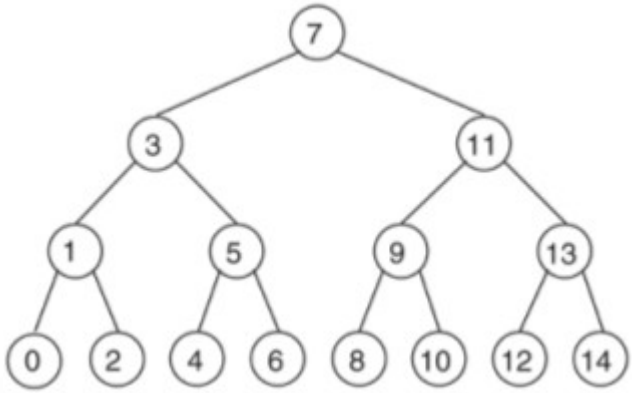
Uma árvore binária é balanceada (ou equilibrada) se, em cada um de seus nós, as subárvores esquerda e direita tiverem aproximadamente a mesma altura.

# Árvores balanceadas

Exemplo ruim para o restabablecimento de árvores completas




Para obter esta árvore é necessário percorrer toda a árvore, O(n)

*Fonte: SZWARCFITER, J. L.; MARKEZON, L. Estruturas de Dados e seus Algoritmos, 3a edição, LTC, 2010.*

# Árvores balanceadas: completas e AVLs

**AVL's**

$$h \geq 1 + \lfloor \log_2 n \rfloor$$

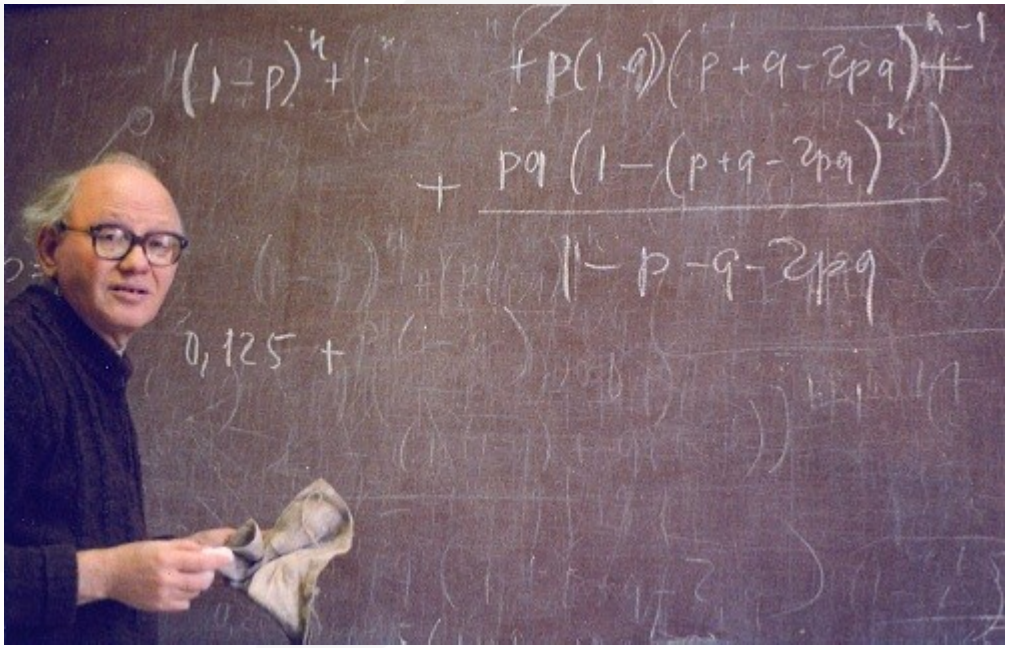
$$h \leq \frac{1}{\log_2 a} \cdot \log_2 (n+1) + \log_a \sqrt{5}$$

**Completas**

$h = 1 + \lfloor \log_2 n \rfloor$

where $a = (\frac{1+\sqrt{5}}{2})$

# AVL



Georgy M. **Adelson-Velsky**
Russia

(1922-2014/abril/26)

Evgenii Mikhailovich **Landis**
Ucrania

(1921-1997)

# AVL

G.M. Adelson-Velskii y E.M. Landis
"**An algorithm for the organization of information**".
Proceedings of the USSR Academy of Sciences, vol. 146, pp. 263–266, **1962**

AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION

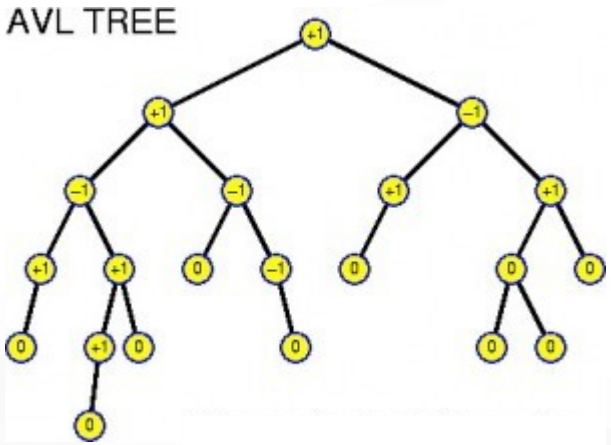
G. M. ADEL'SON-VEL'SKIĬ AND E. M. LANDIS

In the present article we discuss the organization of information contained in the cells of an automatic calculating machine. A three-address machine will be used for this study.

Statement of the problem. The information enters a machine in sequence from a certain reserve. The information element is contained in a group of cells which are arranged one after the other. A certain number (the information estimate), which is different for different elements, is contained in the information element. The information must be organized in the memory of the machine in such a way that at any moment a very large number of operations is not required to scan the information with the given evaluation and to record the new information element.

An algorithm is proposed in which both the search and the recording are carried out in $C \lg N$ operations, where $N$ is the number of information elements which have entered at a given moment.

A part of the memory of the machine is set aside to store the incoming information. The information elements are arranged there in their order of entry. Moreover, in another part of the memory a "reference board" [1] is formed, each cell of which corresponds to one of the information elements.




*AVL foi a primeira estrutura (conhecida) de árvore de altura balanceada/equilibrada.*

# Árvores AVL

- Devido ao balanceamento da altura da árvore, as operações de:
  - Busca
  - Inserção
  - Remoção

  em uma árvore com **n** elementos podem ser efetuadas em $O(\lg n)$ mesmo no pior caso.

# Árvores AVL

- Devido ao balanceamento da altura da árvore, as operações de:
  - Busca
  - Inserção
  - Remoção

  em uma árvore com **n** elementos podem ser efetuadas em $O(\lg n)$ mesmo no pior caso.

- Um teorema provado por Adelson-Velskii e Landis **garante** que **a árvore balanceada nunca será 45% mais alta que a correspondente árvore perfeitamente balanceada**, independentemente do número de nós existentes.

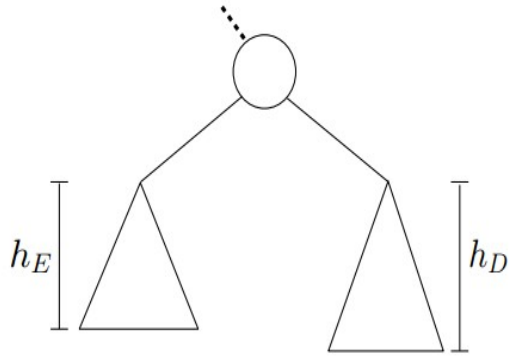
# Árvores AVL

- Uma árvore AVL é definida como:
  - Uma árvore vazia é uma árvore AVL.

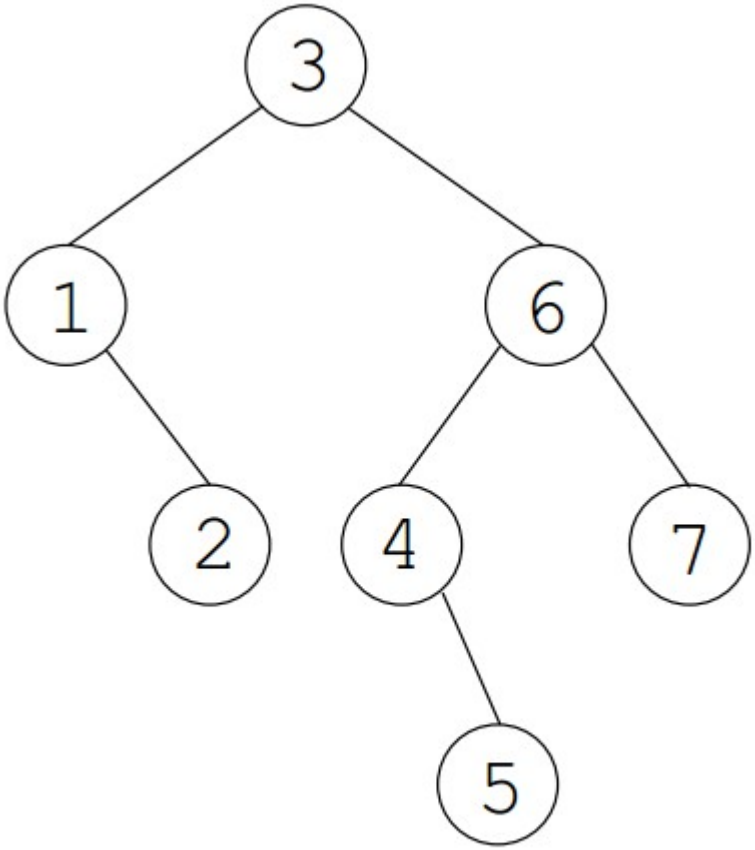
# Árvores AVL

- Uma árvore AVL é definida como:
  - Uma árvore vazia é uma árvore AVL.
  - Sendo **T** uma ABB, com subárvores esquerda (L) e direita (R) , **T** será uma árvore AVL contanto que:
    - L e R são árvores AVL
    - $|h_L - h_R| \leqslant 1$

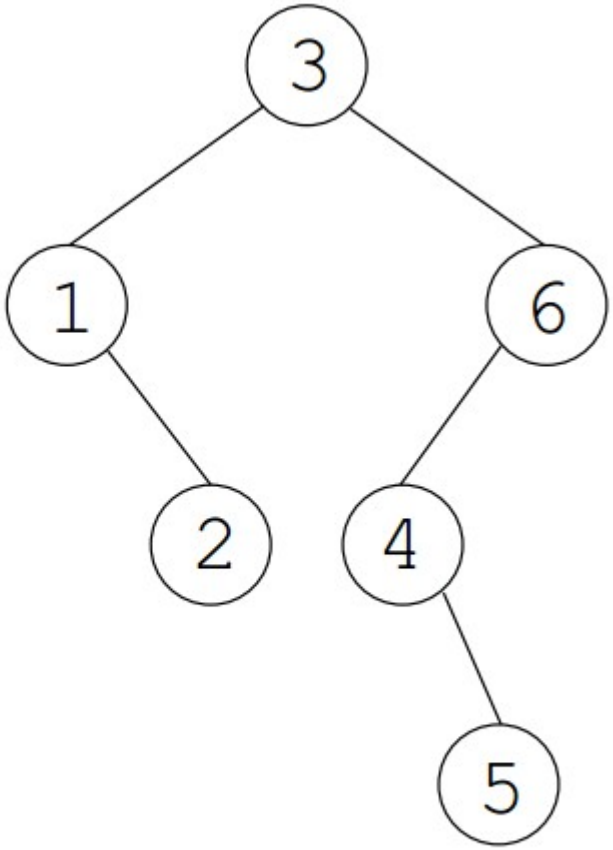
- **A definição de uma ABB de altura equilibrada (AVL) requer que cada subárvore seja também de altura equilibrada.**



# Árvores AVL




AVL




NÃO AVL

# Fator de balanceamento

# Fator de balanceamento

- O **fator de balanceamento/equilibrio** de um nó **T** em uma ABB é definido como: $h_L - h_R$

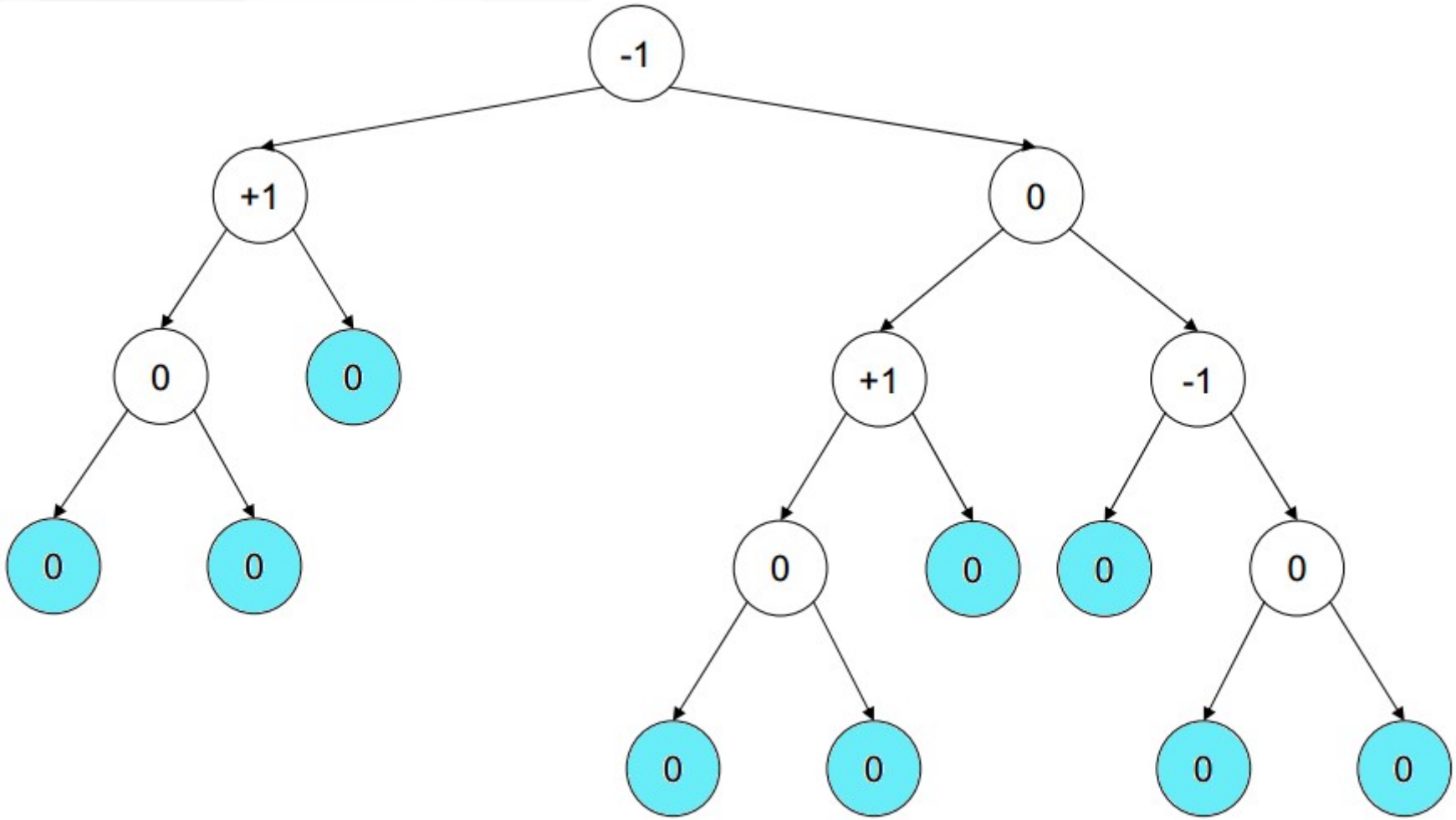
# Fator de balanceamento

- O **fator de balanceamento/equilibrio** de um nó **T** em uma ABB é definido como: $h_L - h_R$

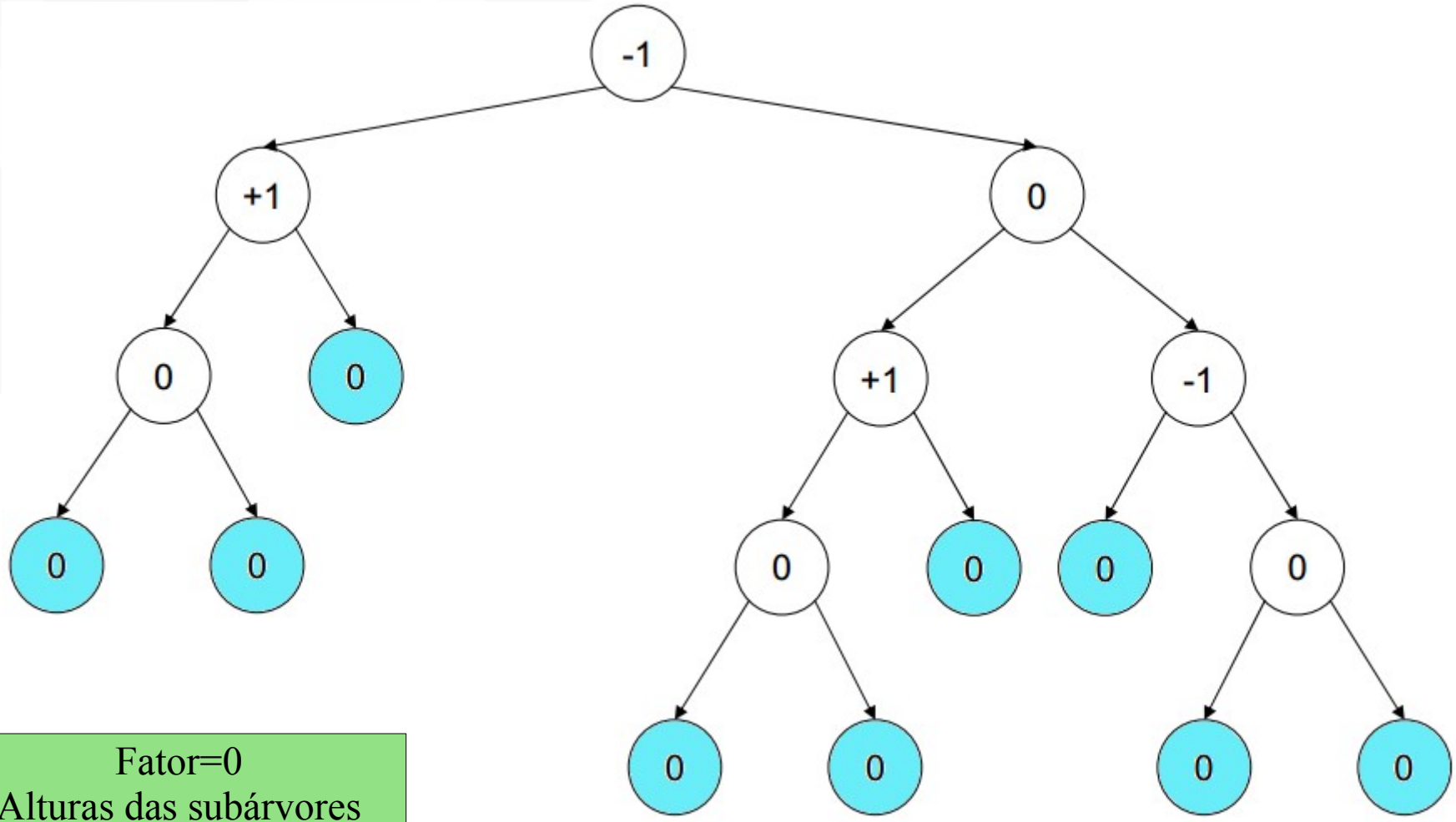
- Para qualquer nó **T** em uma árvore AVL, o fator de balanceamento assume o valor: **+1, 0, -1**.

  - **O fator de balanceamento de uma folha?**

# Fator de balanceamento

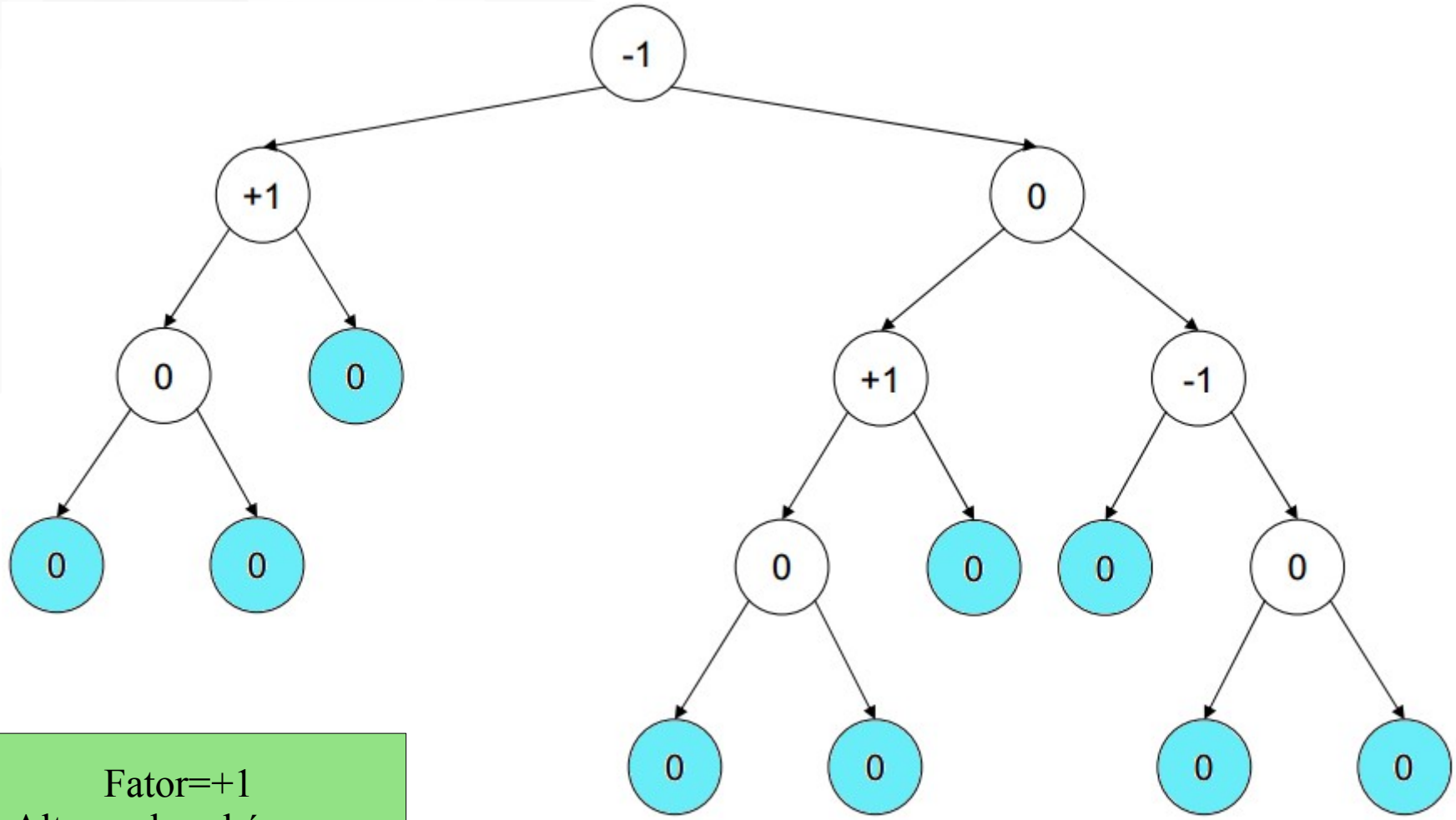


# Fator de balanceamento



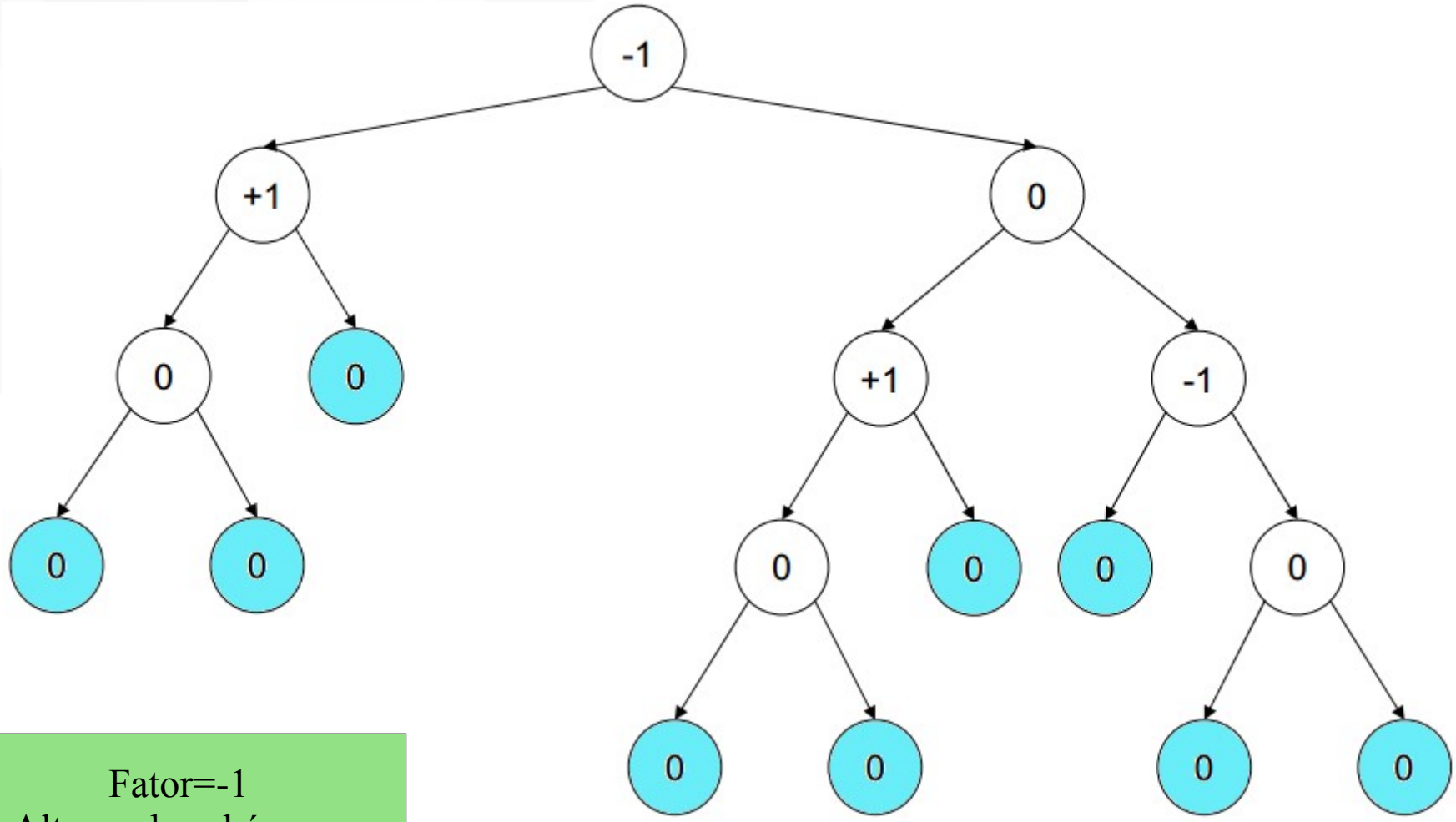

Fator=0
Alturas das subárvores
esquerda e direita
são iguais

# Fator de balanceamento




Fator=+1
Alturas da subárvore esquerda é maior

# Fator de balanceamento




Fator=-1
Alturas da subárvore esquerda é menor

# Exemplo: Inserção de 'Maio'

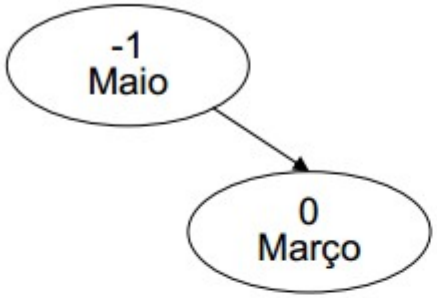
*Depois da inserção*


*Depois do rebalanceamento*

*Sem necessidade de rebalanceamento*

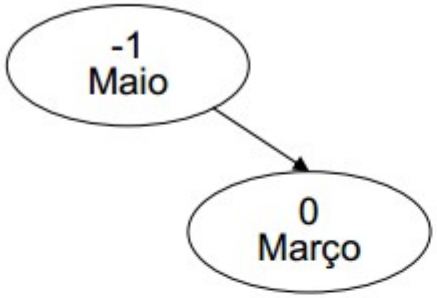
# Exemplo: Inserção de 'Março'

*Depois da inserção*



# Exemplo: Inserção de 'Março'

*Depois da inserção*

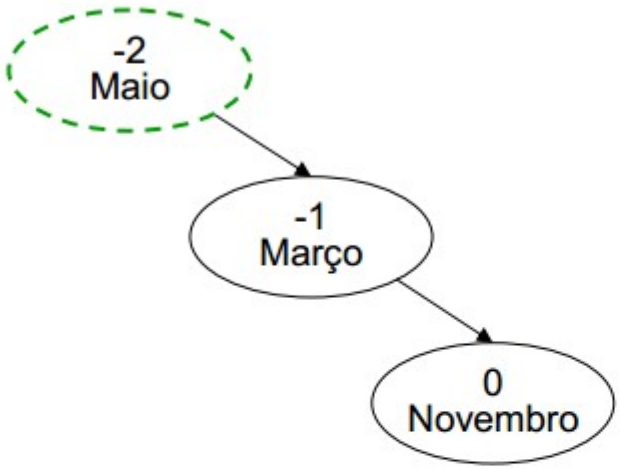



*Depois do rebalanceamento*

*Sem necessidade de rebalanceamento*

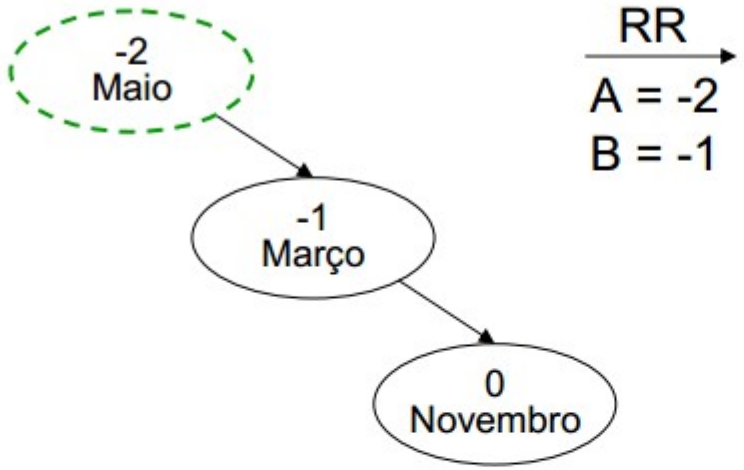
# Exemplo: Inserção de 'Novembro'

*Depois da inserção*

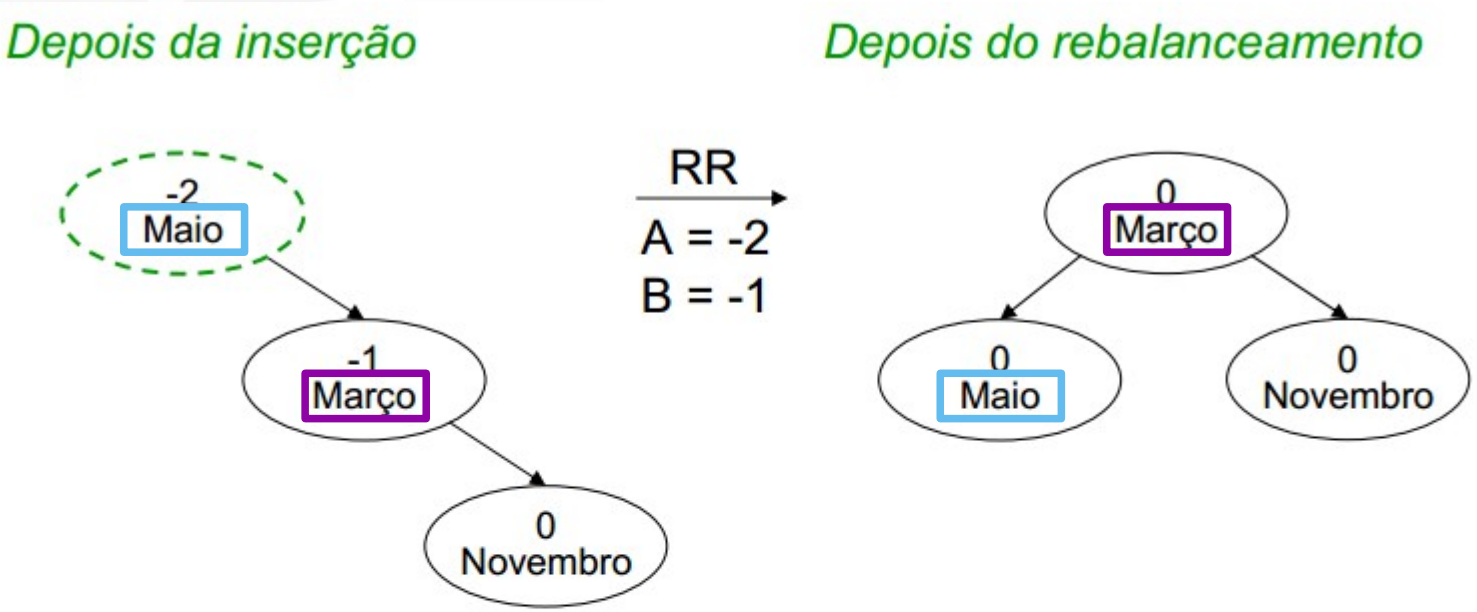


# Exemplo: Inserção de 'Novembro'

*Depois da inserção*

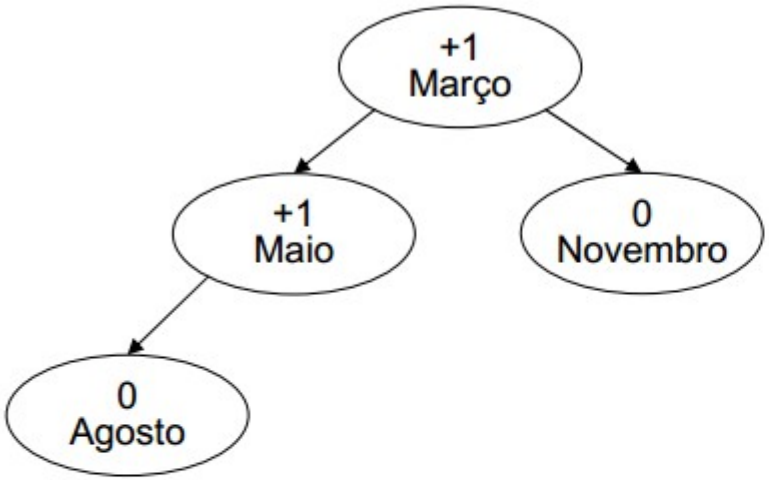


# Exemplo: Inserção de 'Novembro'



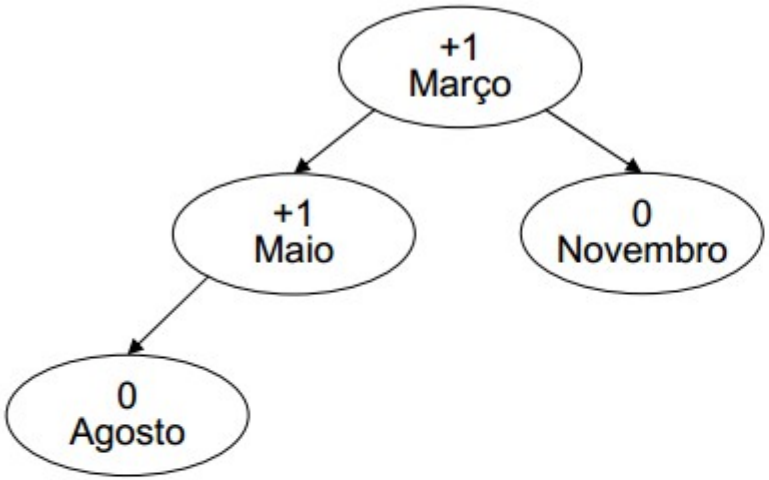
# Exemplo: Inserção de 'Agosto'

*Depois da inserção*



# Exemplo: Inserção de 'Agosto'

*Depois da inserção*

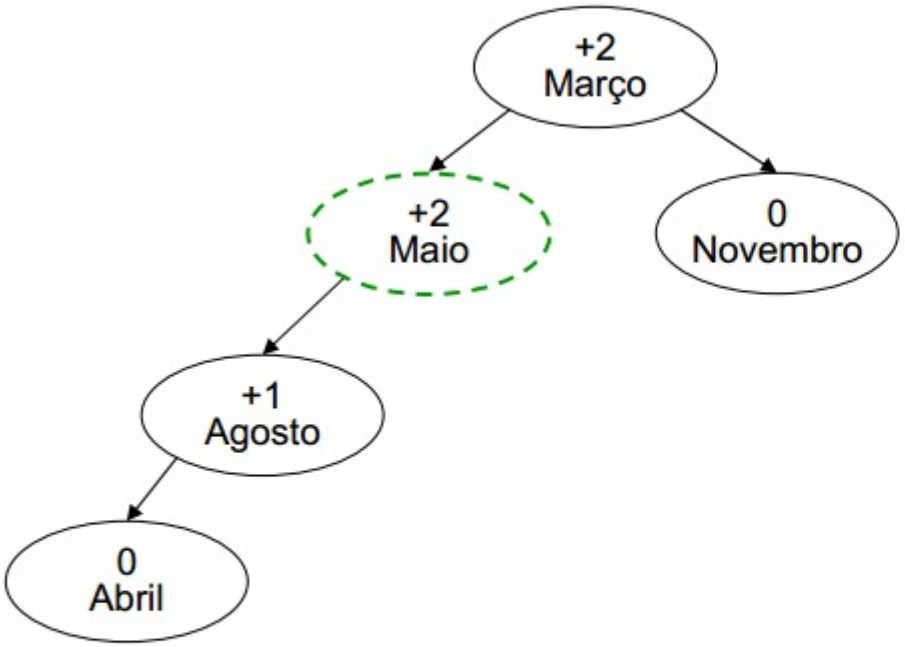



*Depois do rebalanceamento*

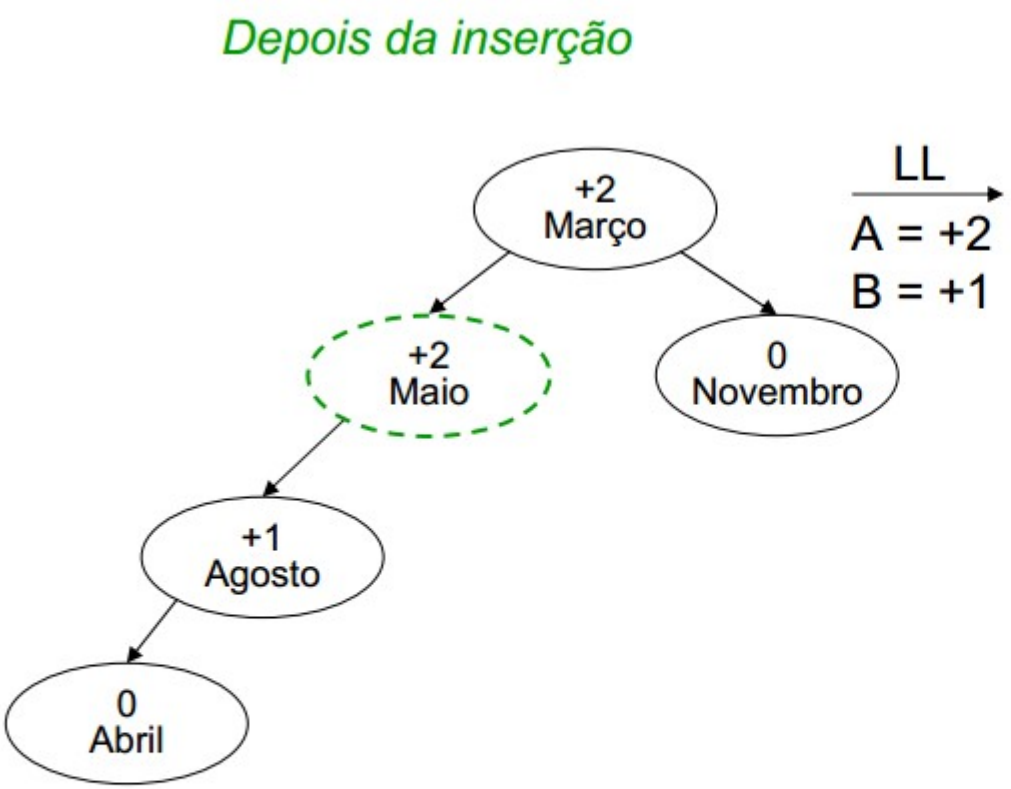
*Sem necessidade de rebalanceamento*

# Exemplo: Inserção de 'Abril'

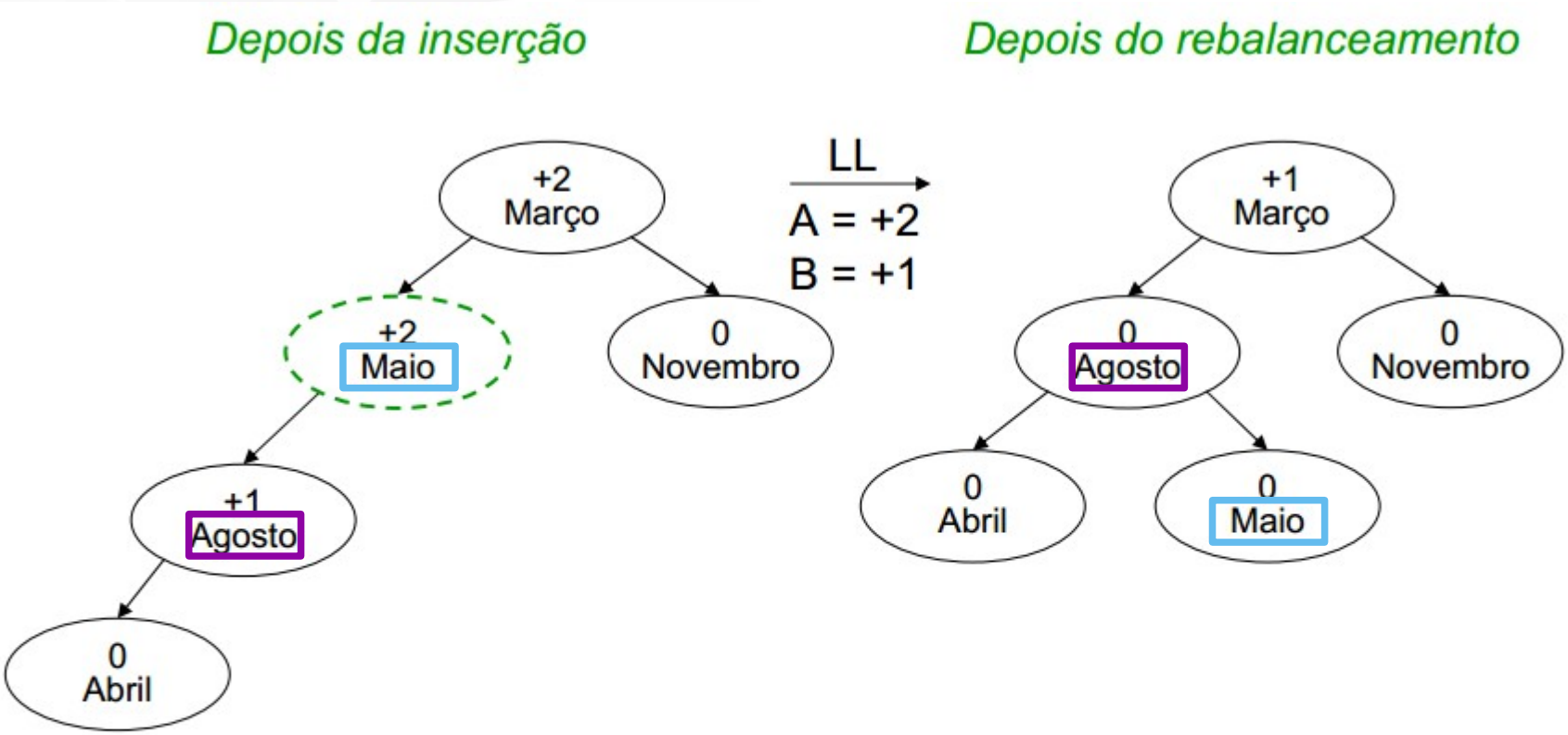
*Depois da inserção*



# Exemplo: Inserção de 'Abril'

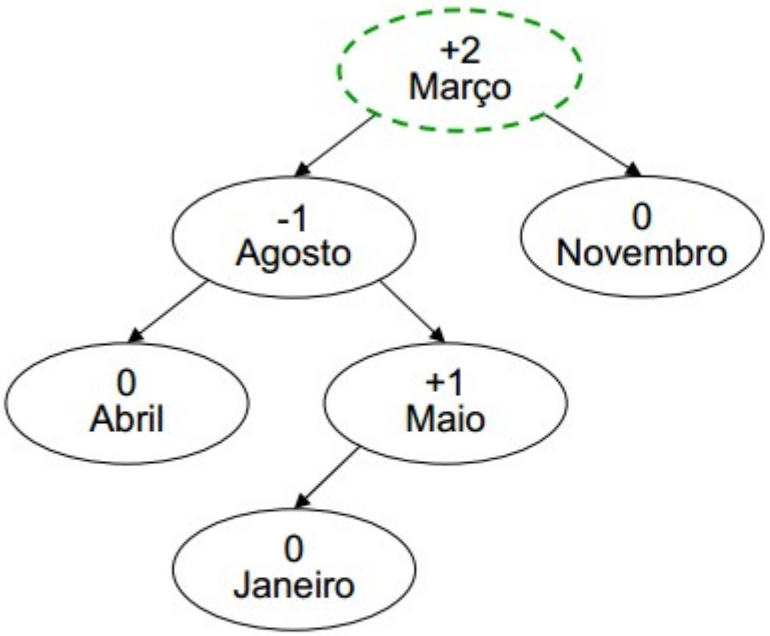


# Exemplo: Inserção de 'Abril'



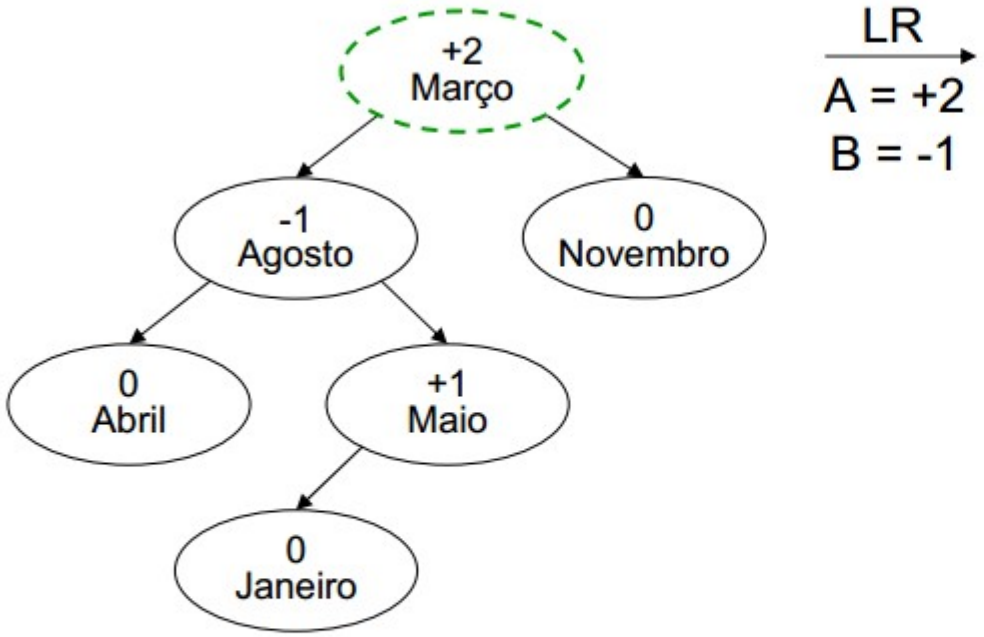
# Exemplo: Inserção de 'Janeiro'

*Depois da inserção*

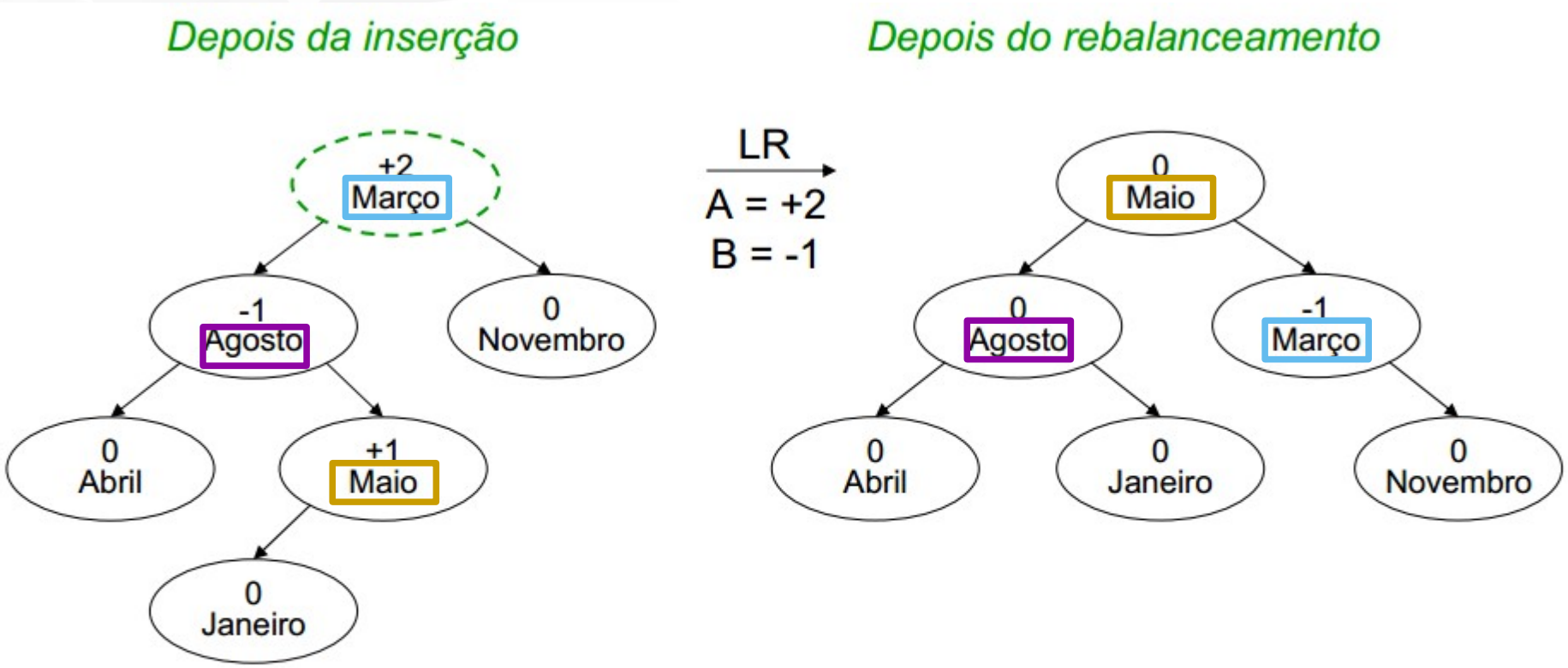


# Exemplo: Inserção de 'Janeiro'

*Depois da inserção*

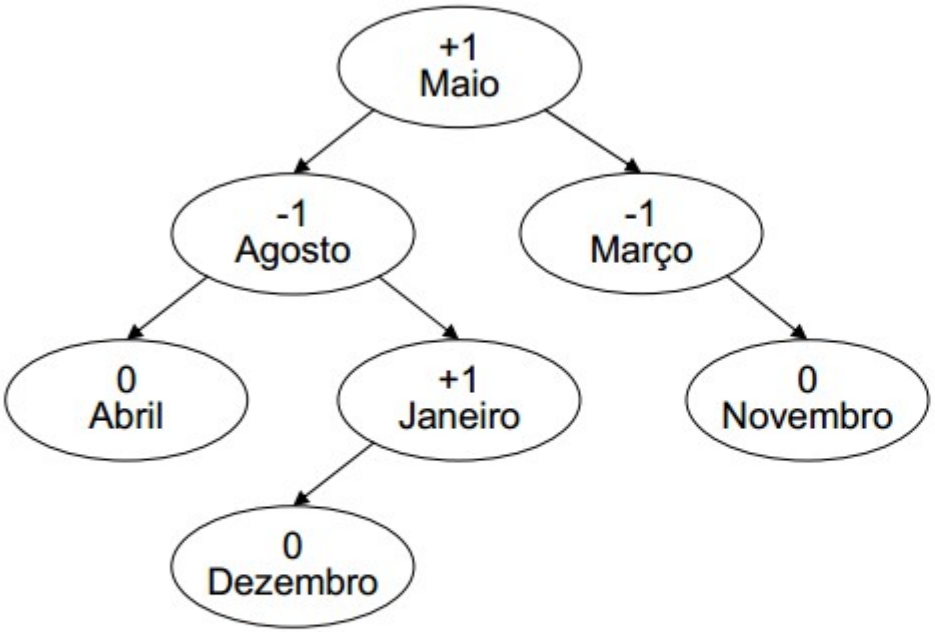


# Exemplo: Inserção de 'Janeiro'



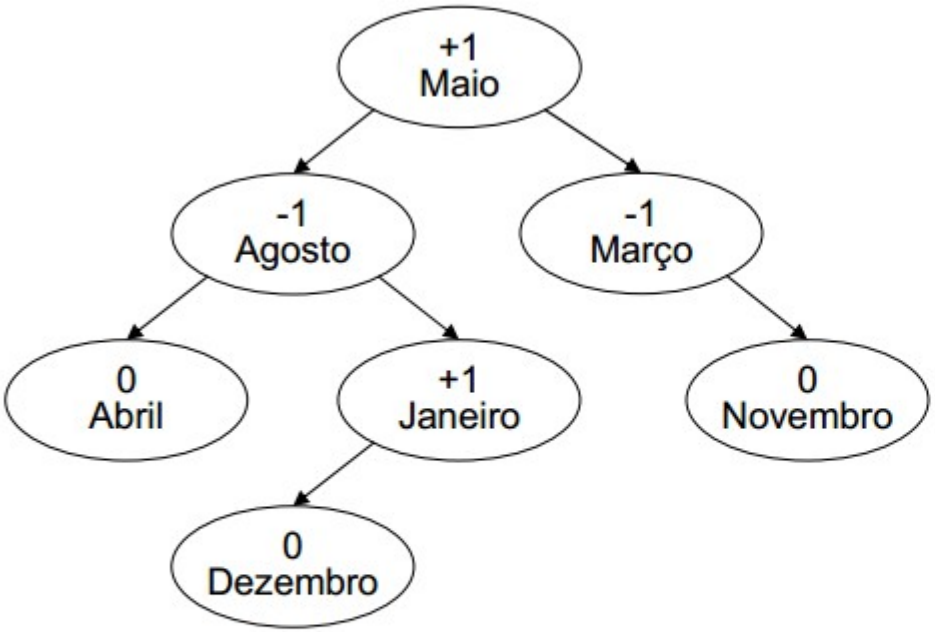
# Exemplo: Inserção de 'Dezembro'

*Depois da inserção*



# Exemplo: Inserção de 'Dezembro'

*Depois da inserção*

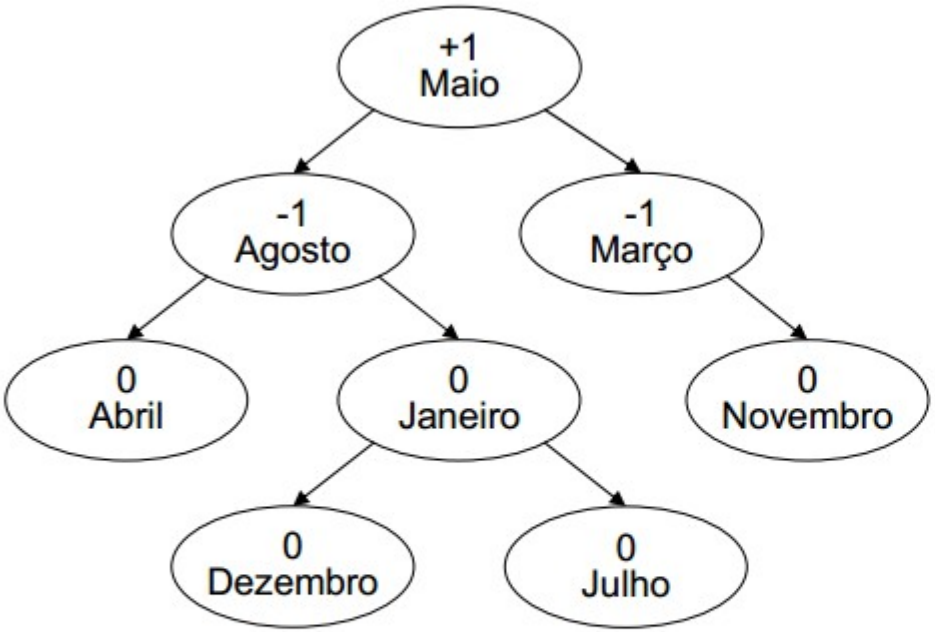



*Depois do rebalanceamento*

*Sem necessidade de rebalanceamento*

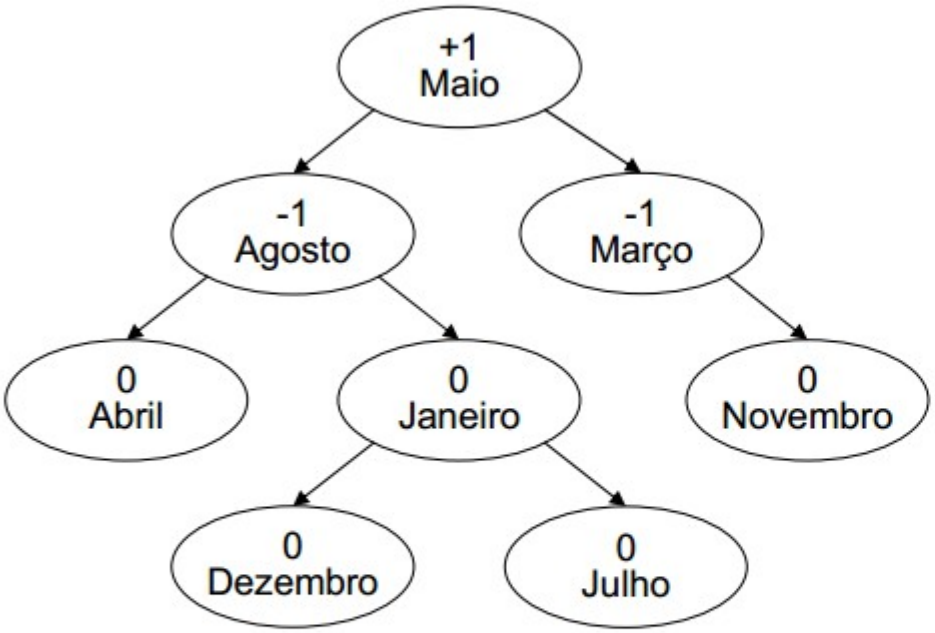
# Exemplo: Inserção de 'Julho'

*Depois da inserção*



# Exemplo: Inserção de 'Julho'

*Depois da inserção*

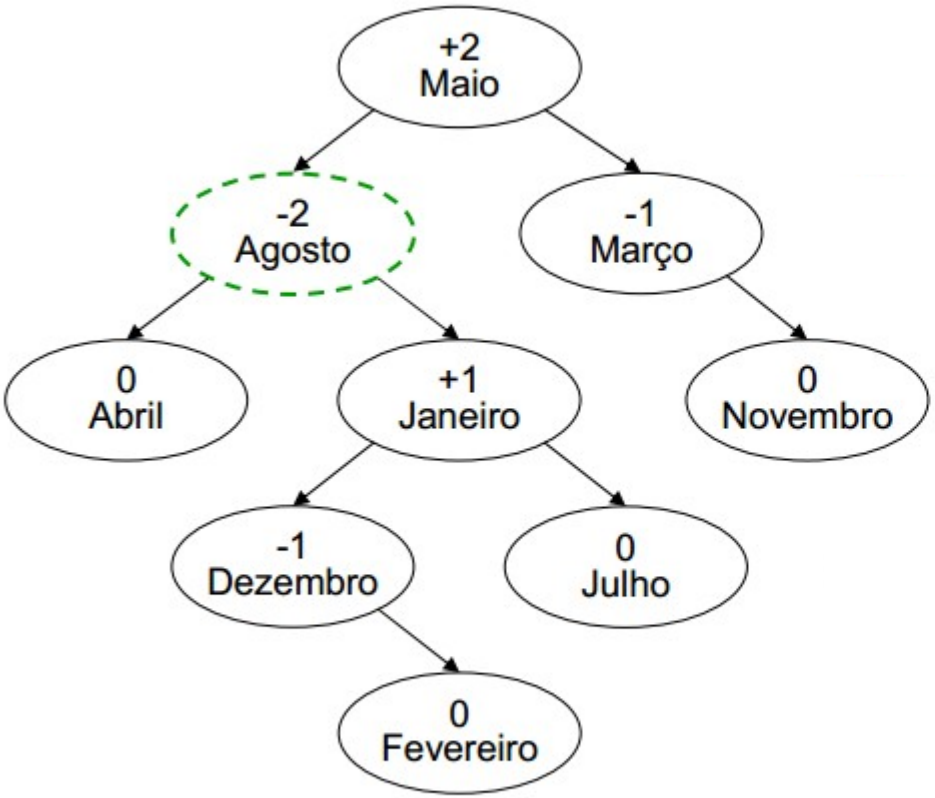



*Depois do rebalanceamento*

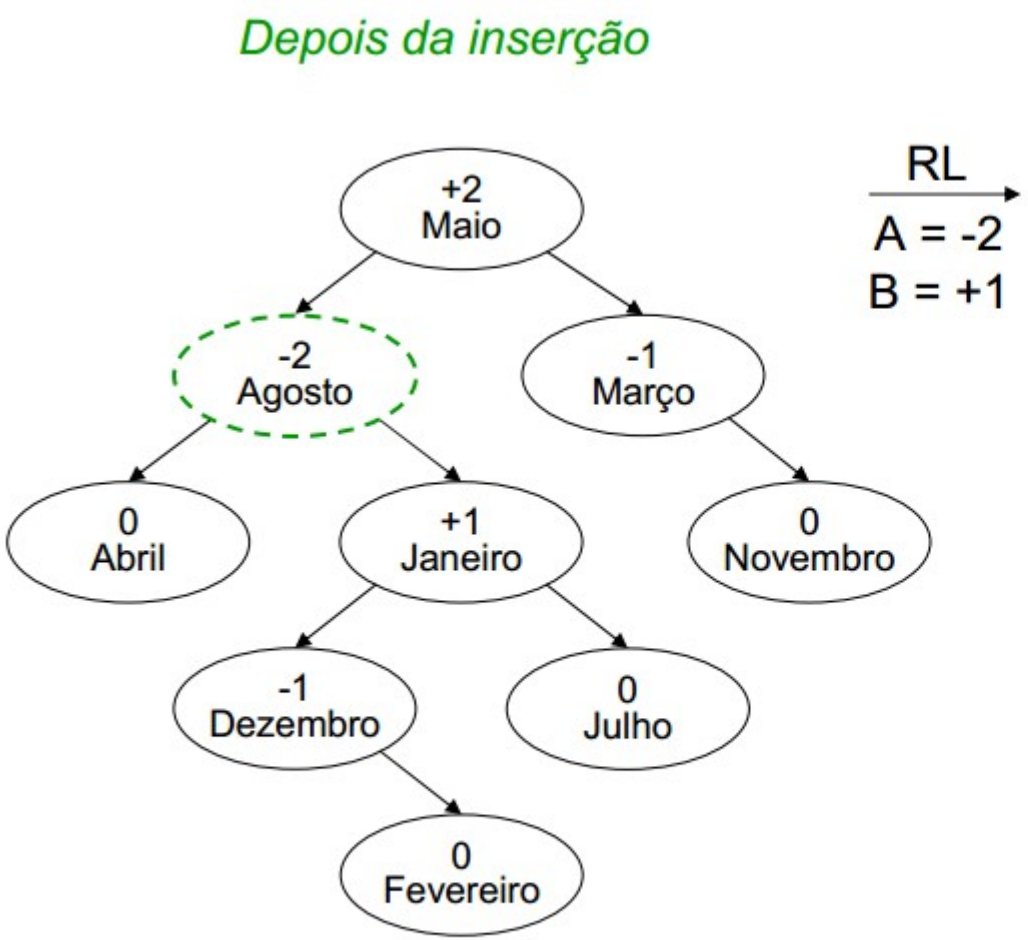
*Sem necessidade de rebalanceamento*

# Exemplo: Inserção de 'Fevereiro'

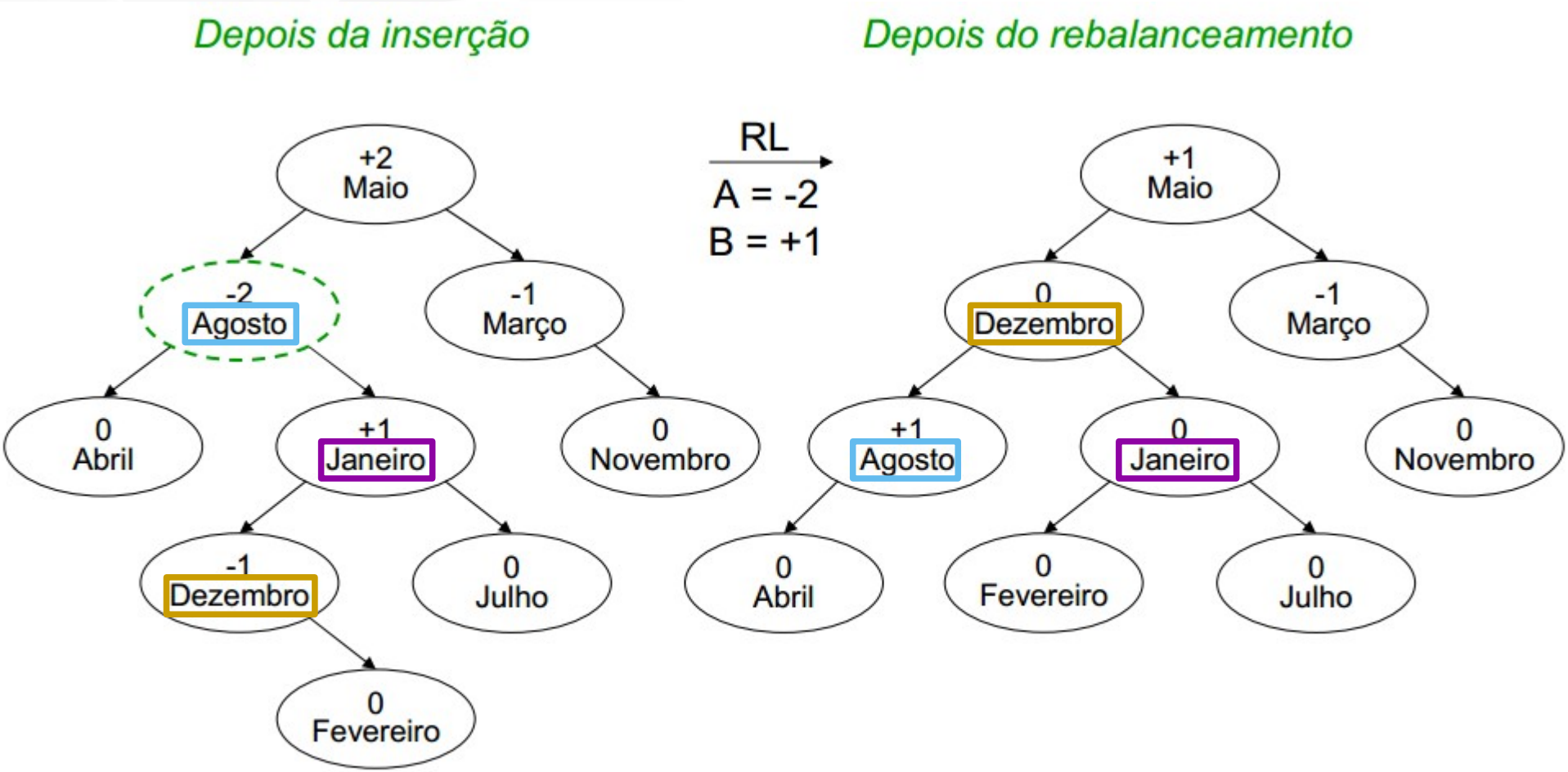
*Depois da inserção*



# Exemplo: Inserção de 'Fevereiro'

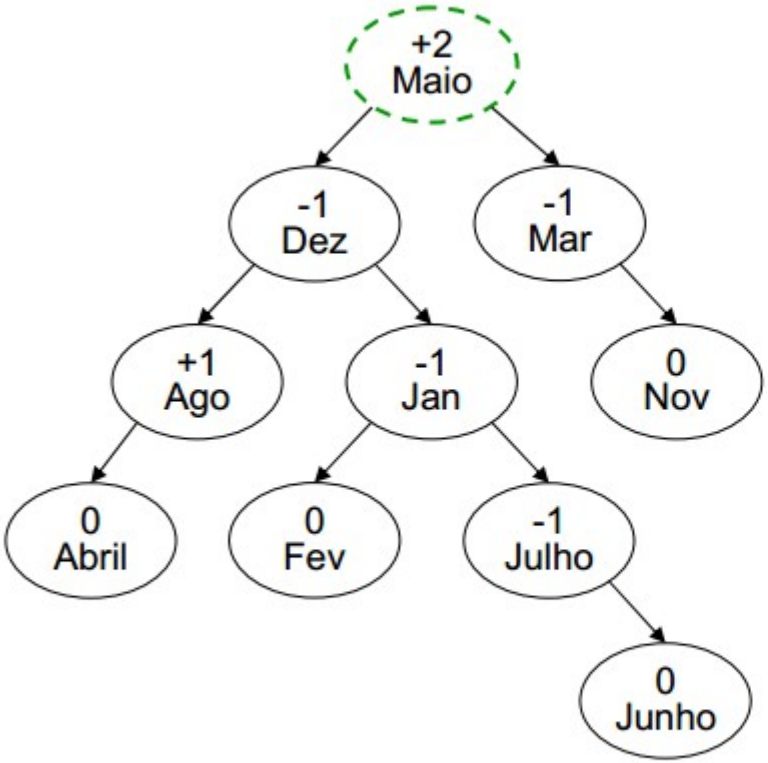


# Exemplo: Inserção de 'Fevereiro'



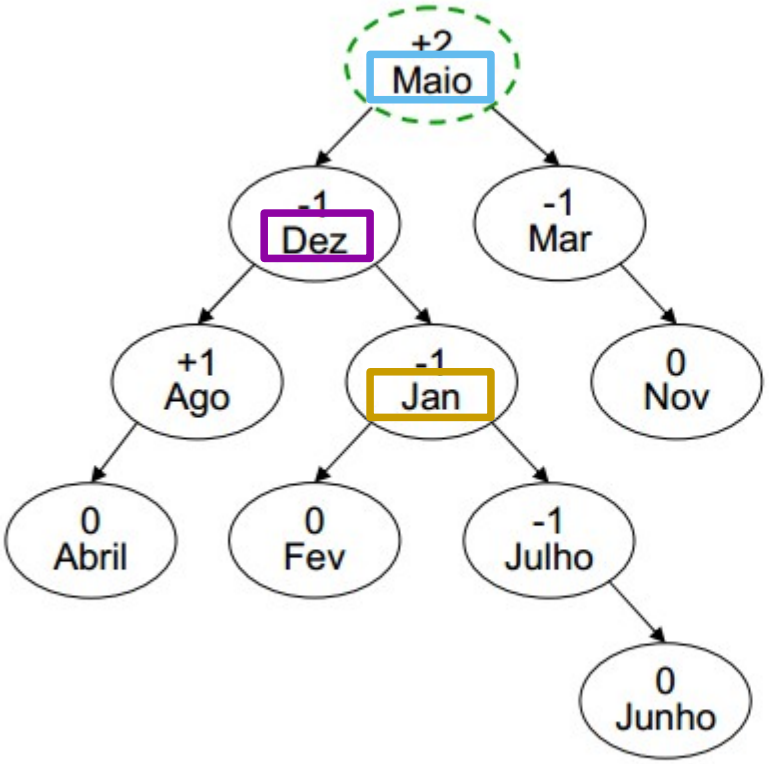
# Exemplo: Inserção de 'Junho'

*Depois da inserção*



# Exemplo: Inserção de 'Junho'

*Depois da inserção*

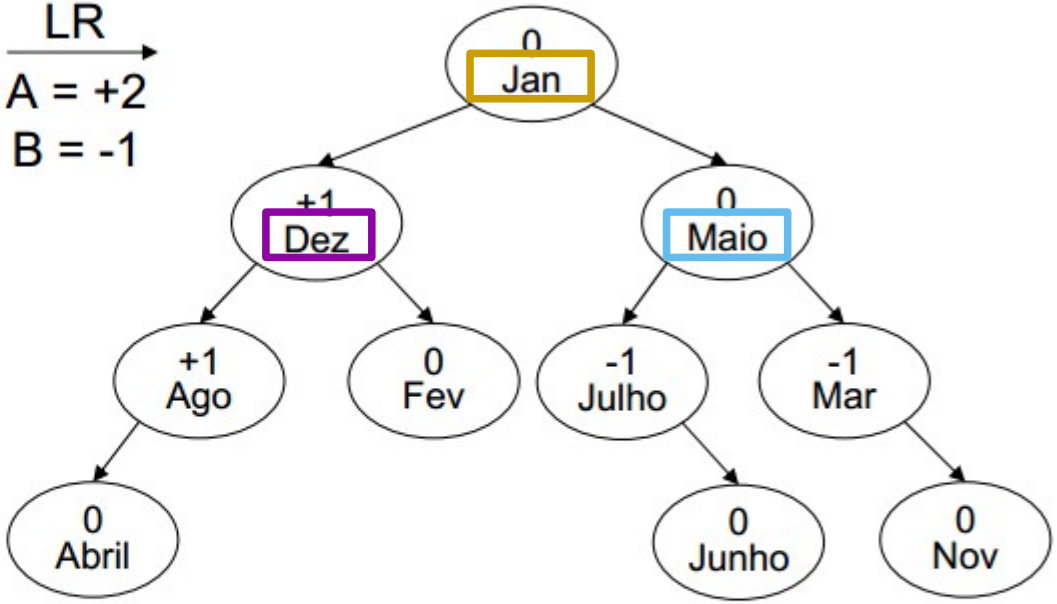



LR

A = +2

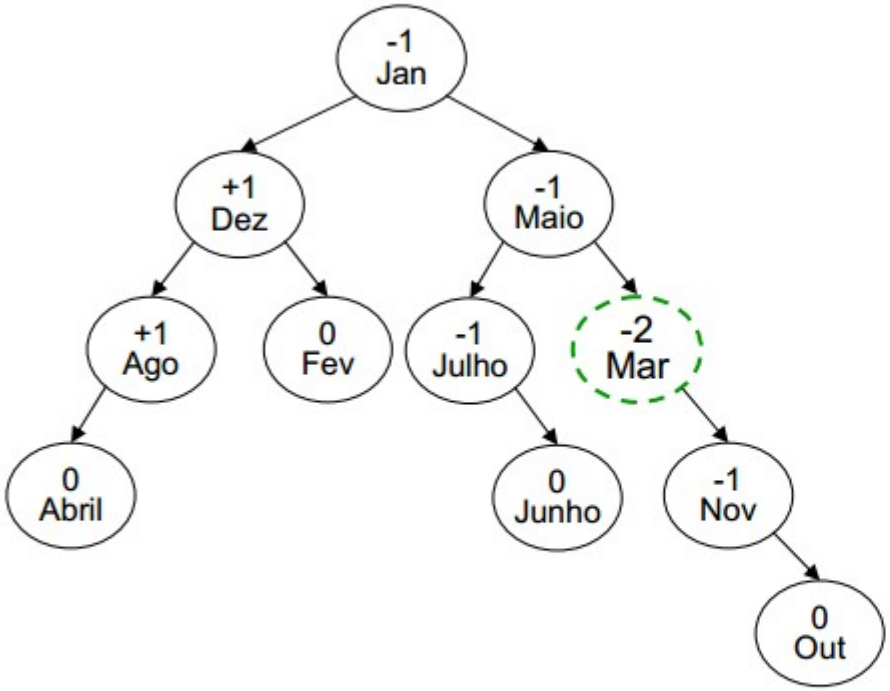
B = -1

*Depois do rebalanceamento*

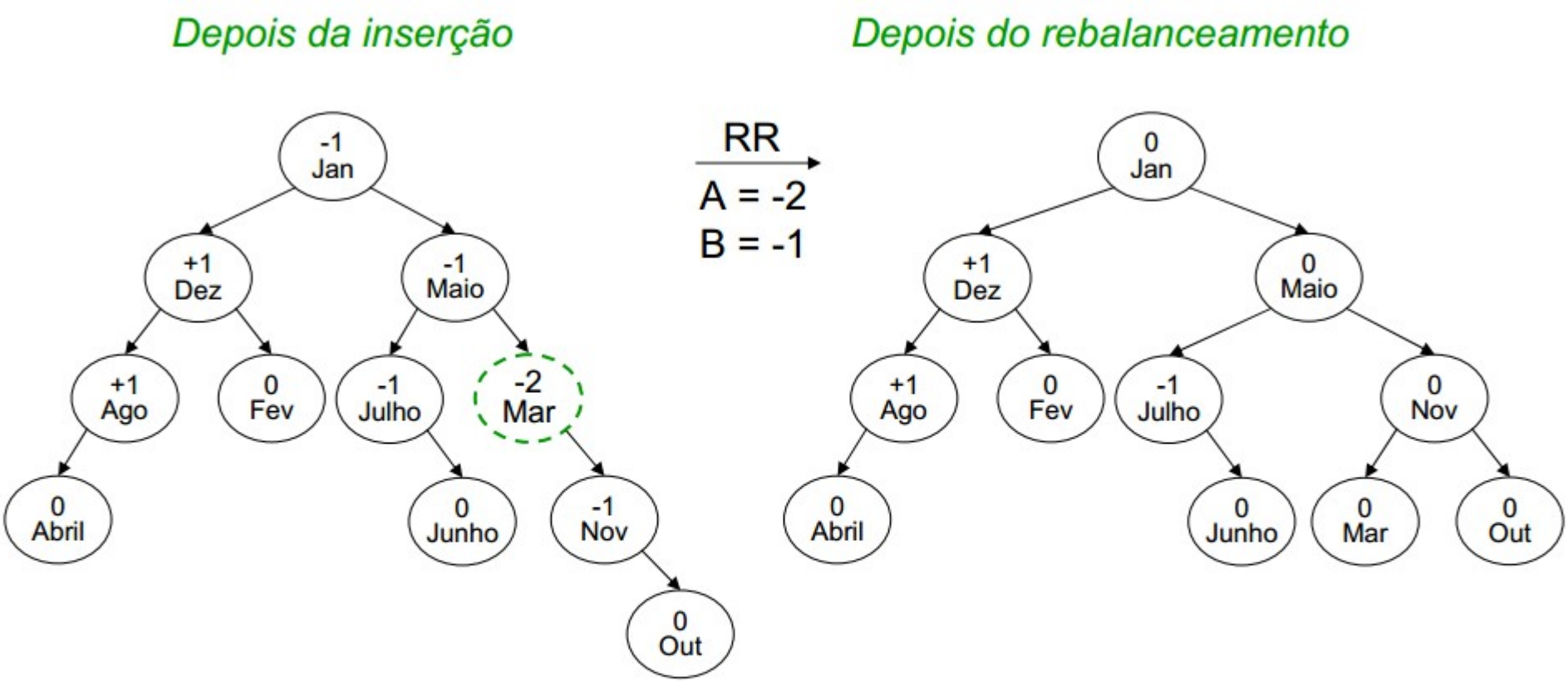


# Exemplo: Inserção de 'Outubro'

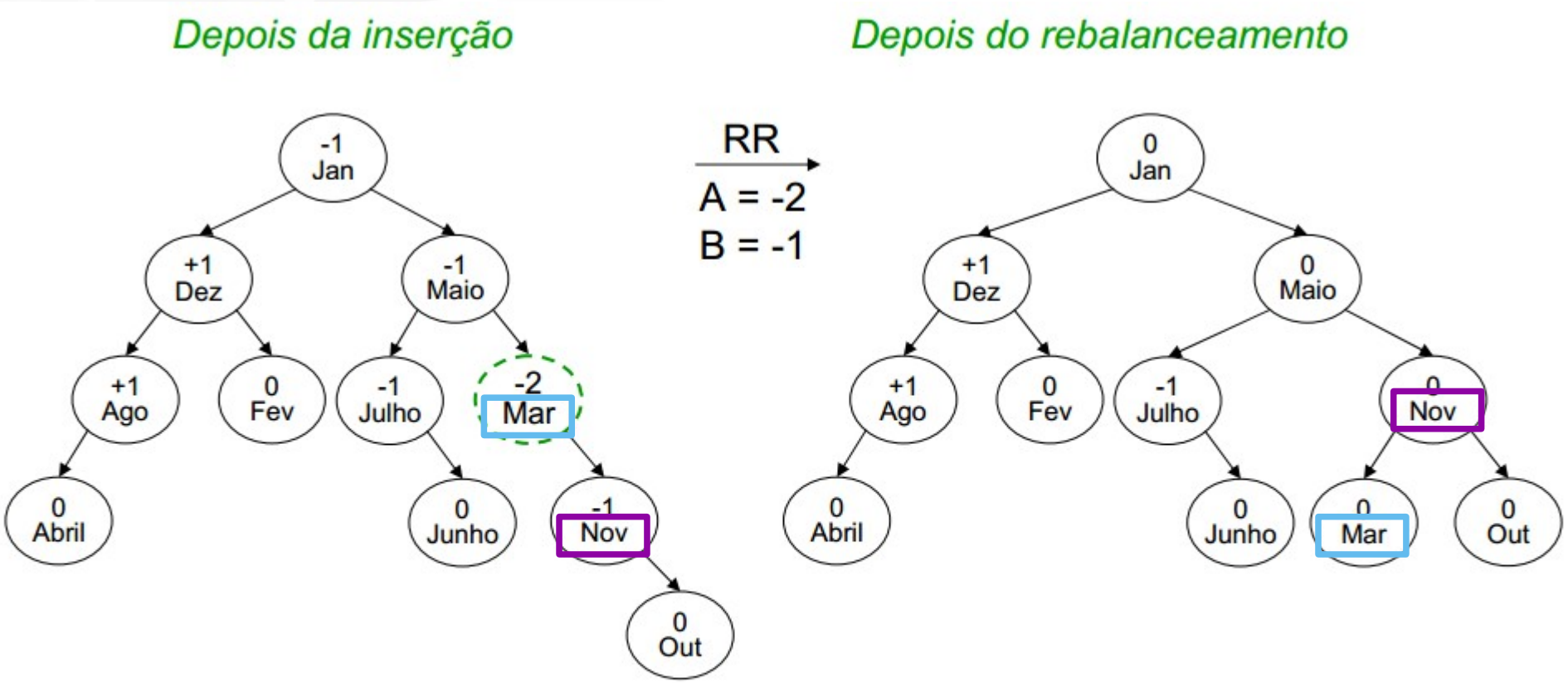
*Depois da inserção*



# Exemplo: Inserção de 'Outubro'

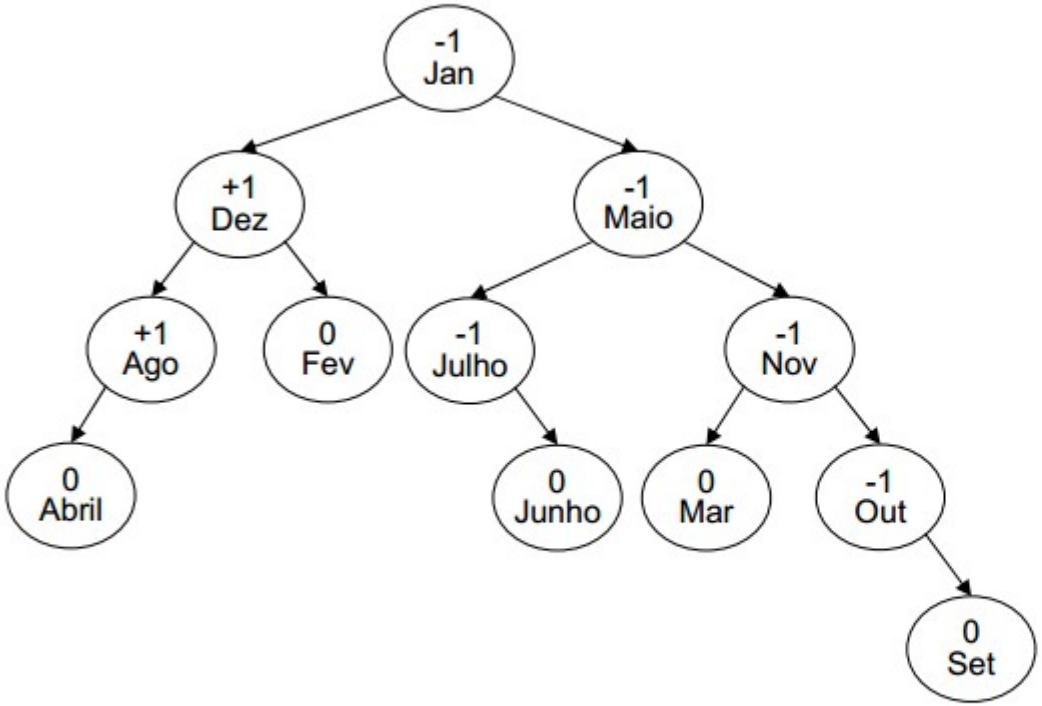


# Exemplo: Inserção de 'Outubro'



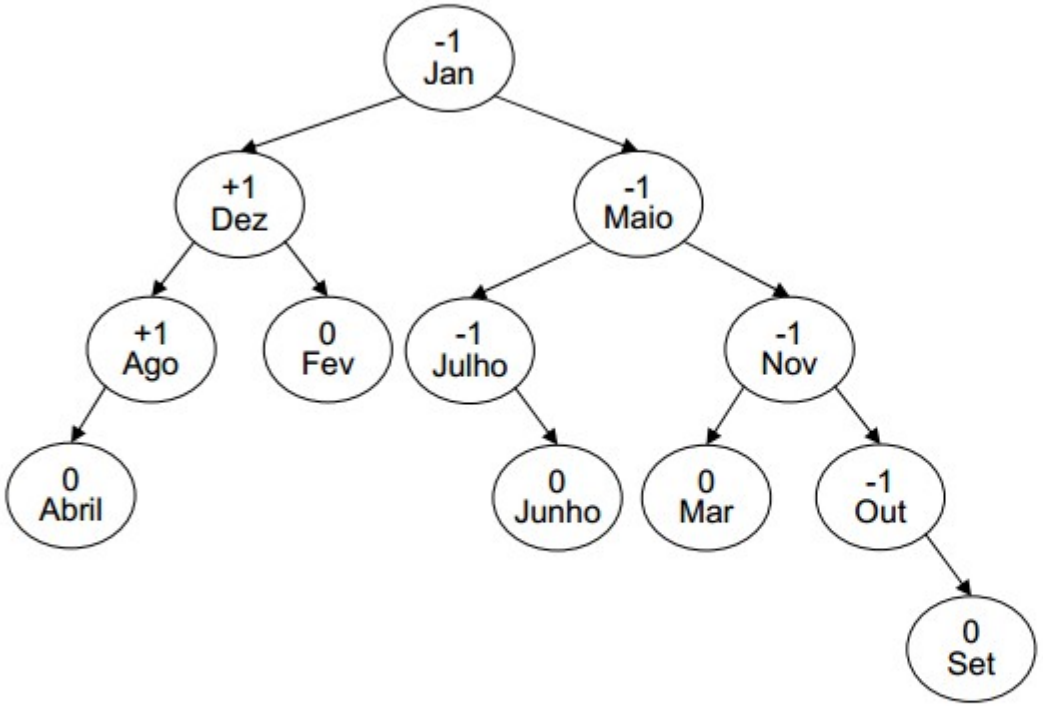
# Exemplo: Inserção de 'Setembro'

*Depois da inserção*



# Exemplo: Inserção de 'Setembro'

*Depois da inserção*




*Depois do rebalanceamento*

*Sem necessidade de rebalanceamento*

# Rotações

# Rotações

- O processo de rebalanceamento é conduzido utilizando 4 tipos de rotações
  - LL
  - RR
  - LR
  - RL

- Suponha que o novo nó inserido é **Y**:

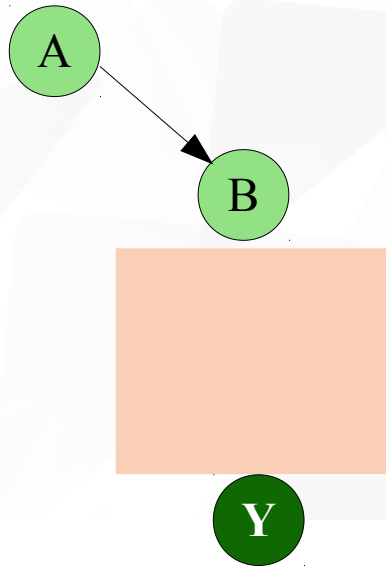
  As rotações são caracterizadas pelo ancestral **A (com fator de balanceamento +2 ou -2)** mais próximo do nó **Y**.

# Rotações

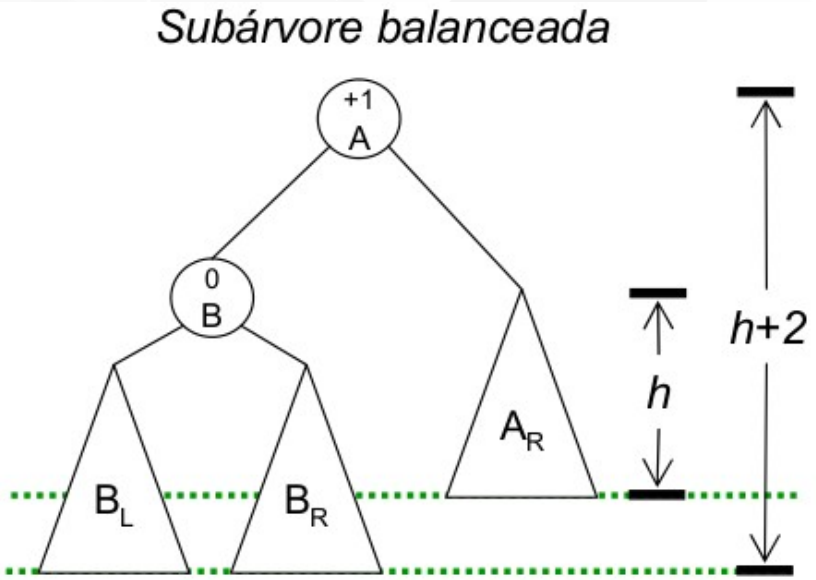
- **LL:** Y inserido na subárvore esquerda da subárvore esquerda de A
- **LR:** Y inserido na subárvore direita da subárvore esquerda de A
- **RR:** Y inserido na subárvore direita da subárvore direita de A
- **RL:** Y inserido na subárvore esquerda da subárvore direita de A

Seja **B** o filho de **A** no qual ocorreu a inserção de **Y**

- **LL (A = +2; B = +1)**
- **LR (A = +2; B = -1)**
- **RR (A = -2; B = -1)**
- **RL (A = -2; B = +1)**

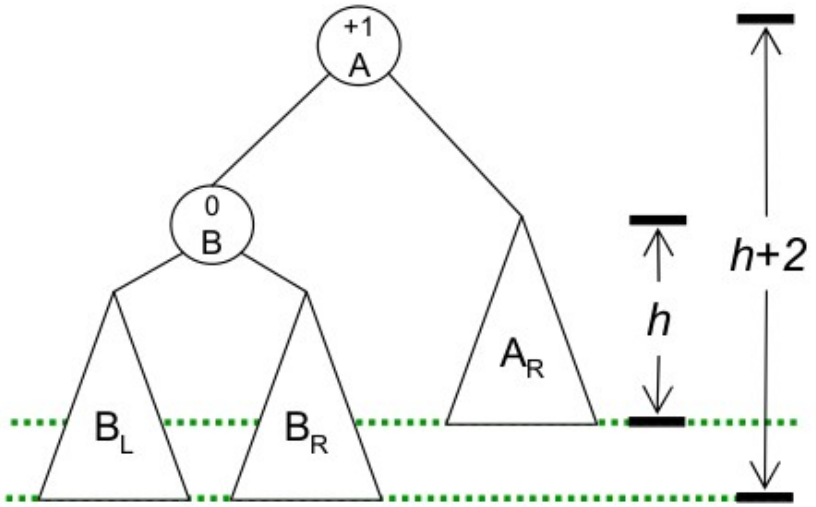


# Rotação LL

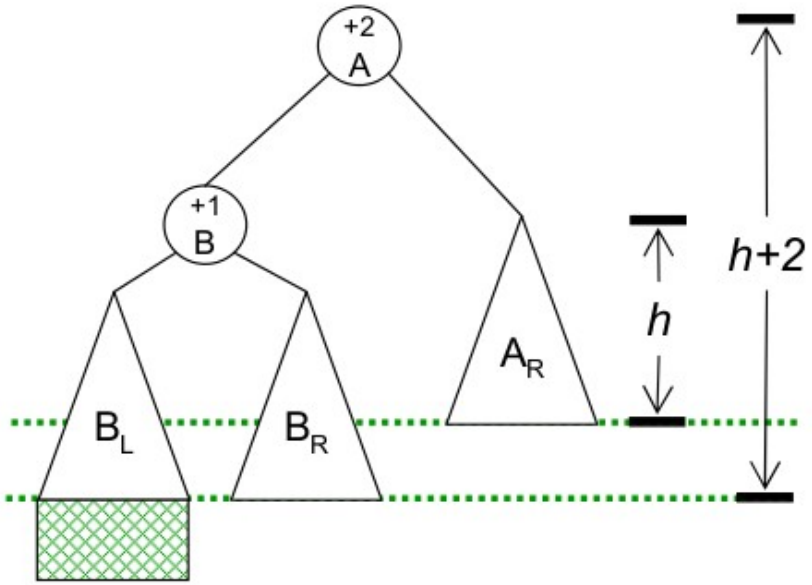


# Rotação LL

*Subárvore balanceada*



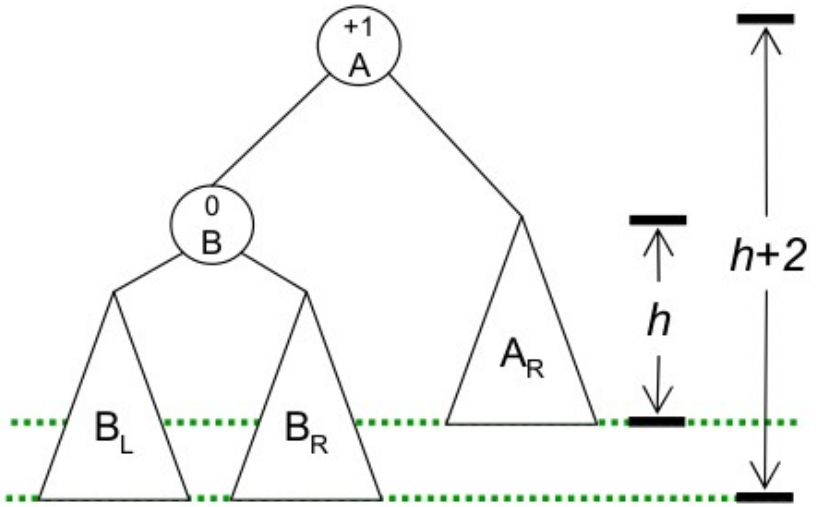

*Subárvore desbalanceada após inserção*



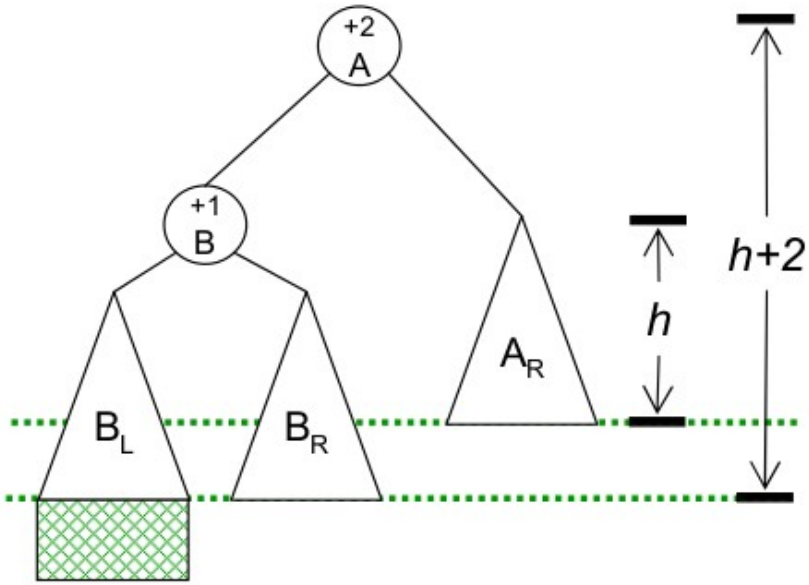

Altura de $B_L$ aumenta para $h$+1

# Rotação LL

*Subárvore balanceada*

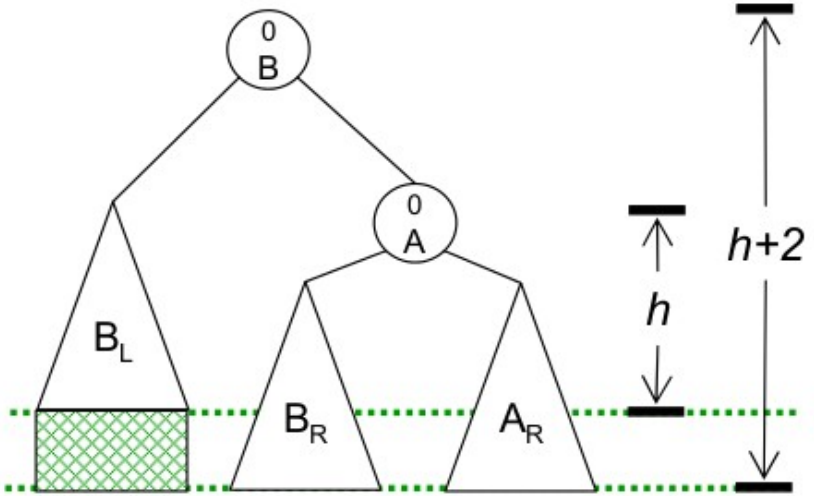



*Subárvore desbalanceada após inserção*

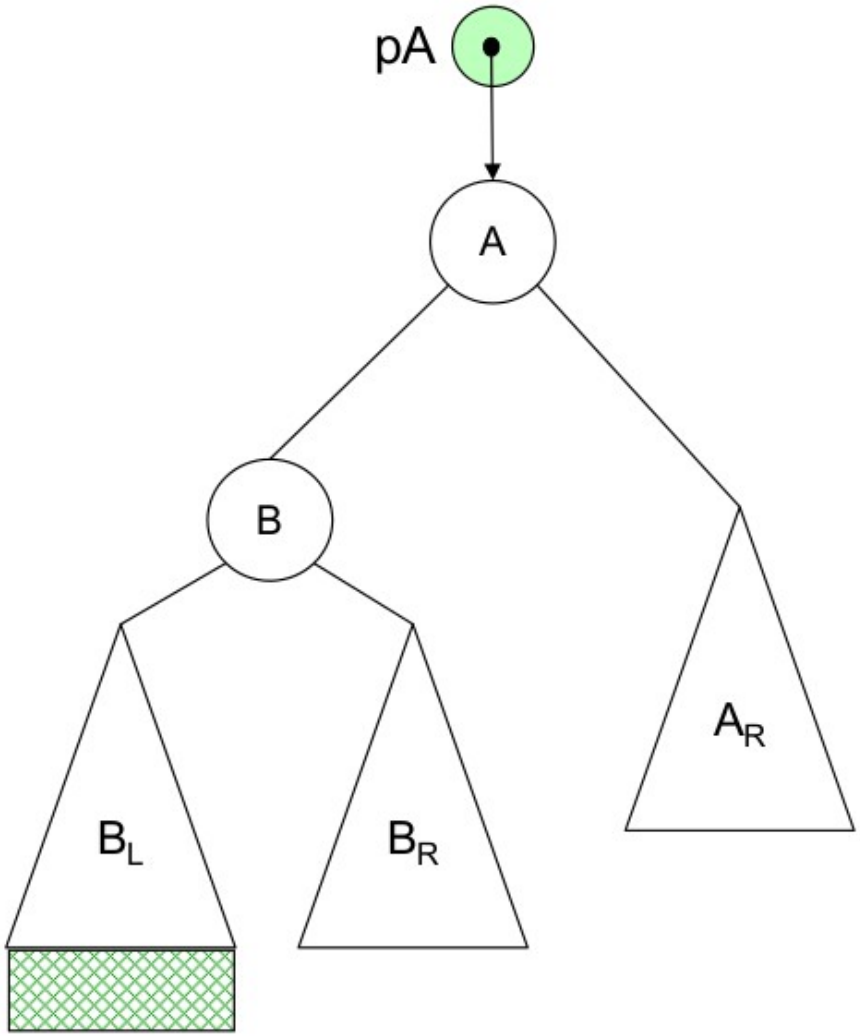



Altura de $B_L$ aumenta para $h$+1

*Subárvore rebalanceada*

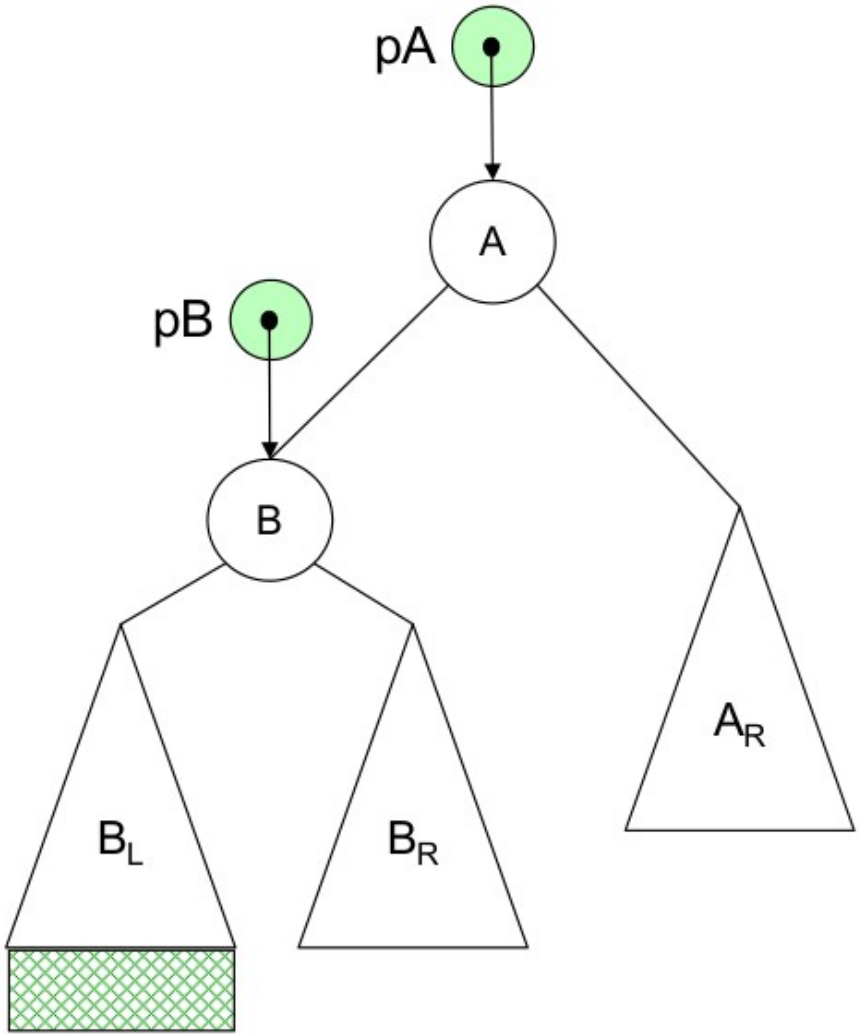


# Rotação LL



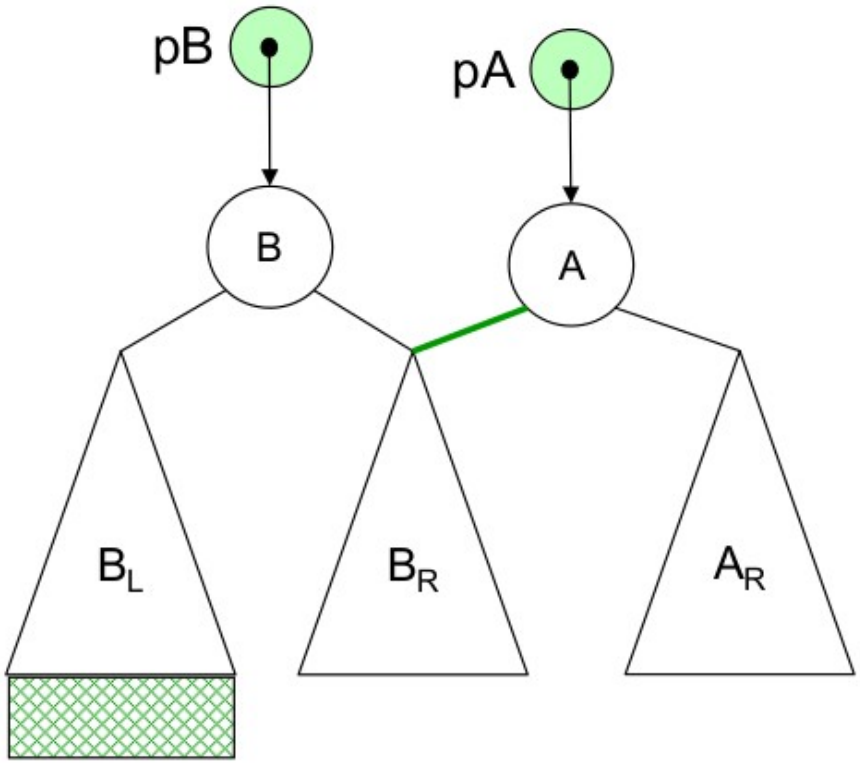

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->LeftNode;
  - pA->LeftNode = pB->RightNode;
  - pB->RightNode = pA;
  - pA = pB;

# Rotação LL



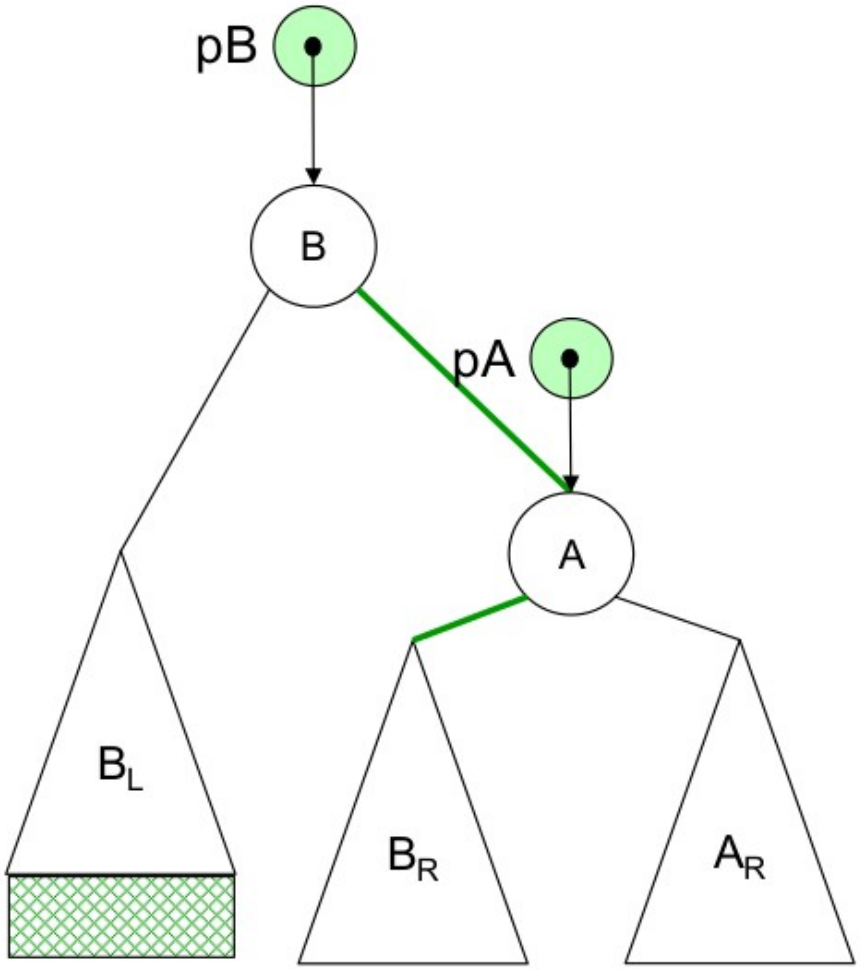

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - **pB = pA->LeftNode;**
  - pA->LeftNode = pB->RightNode;
  - pB->RightNode = pA;
  - pA = pB;

# Rotação LL



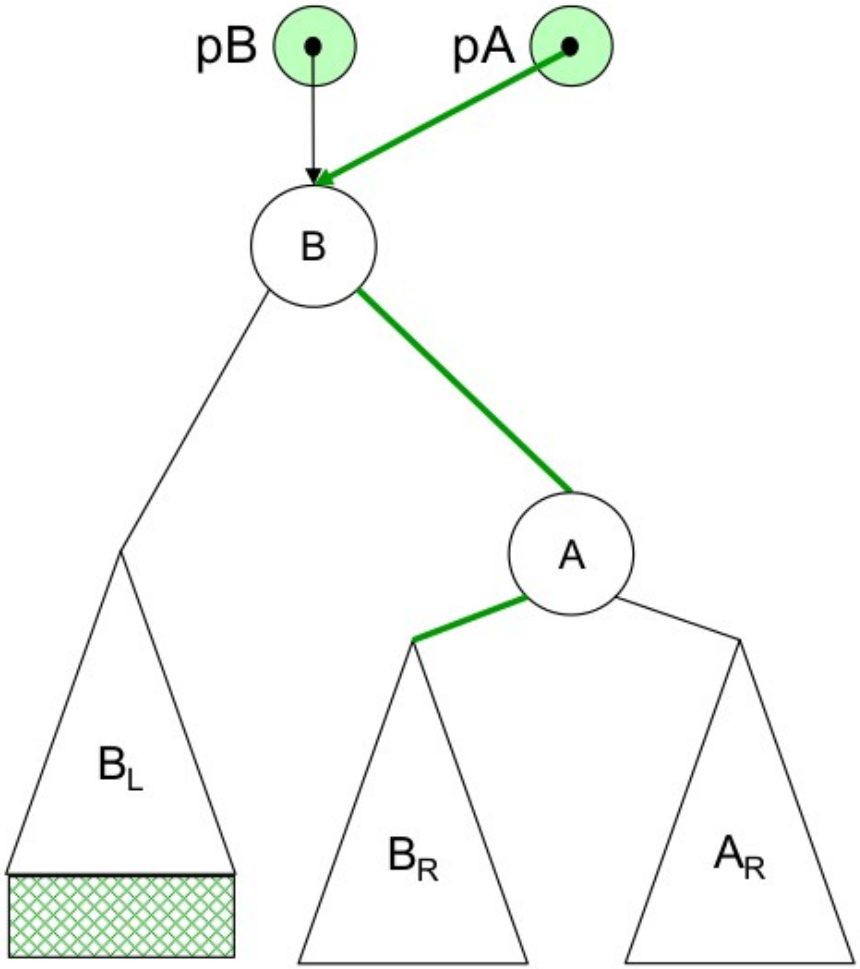

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->LeftNode;
  - **pA->LeftNode = pB->RightNode;**
  - pB->RightNode = pA;
  - pA = pB;

# Rotação LL



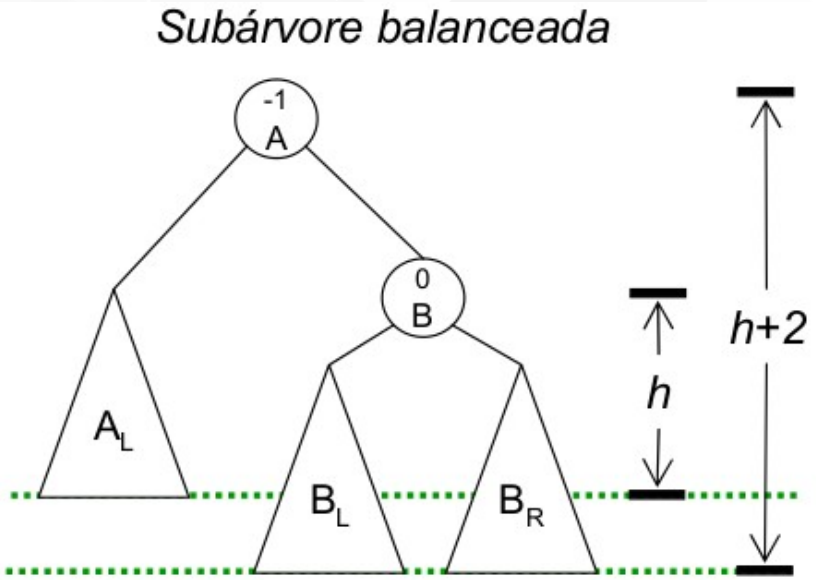

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->LeftNode;
  - pA->LeftNode = pB->RightNode;
  - **pB->RightNode = pA;**
  - pA = pB;

# Rotação LL



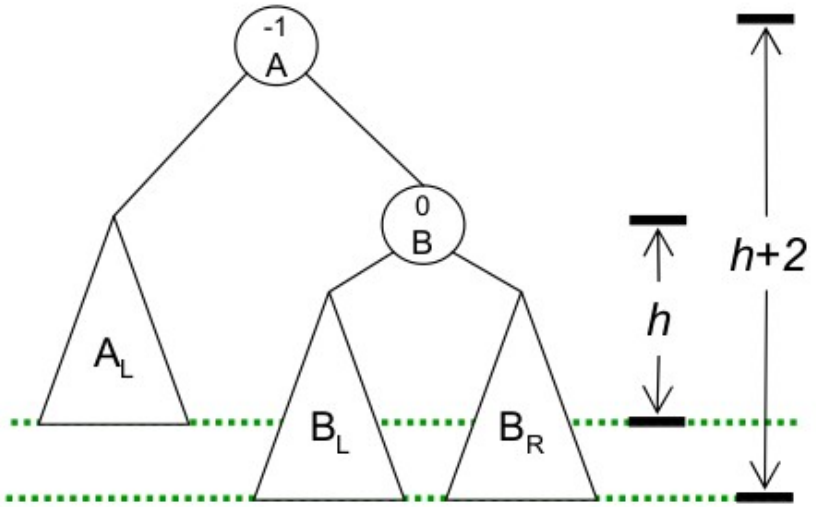

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->LeftNode;
  - pA->LeftNode = pB->RightNode;
  - pB->RightNode = pA;
  - **pA = pB;**

# Rotação RR

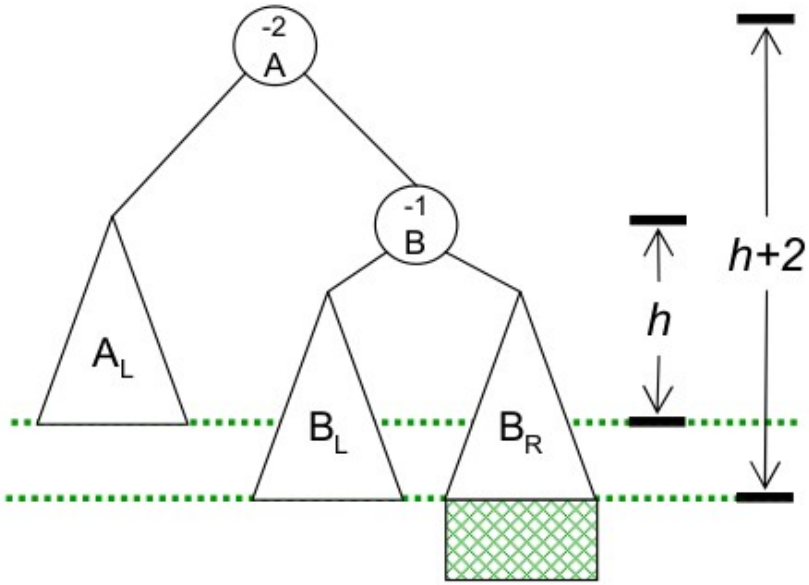


# Rotação RR

*Subárvore balanceada*



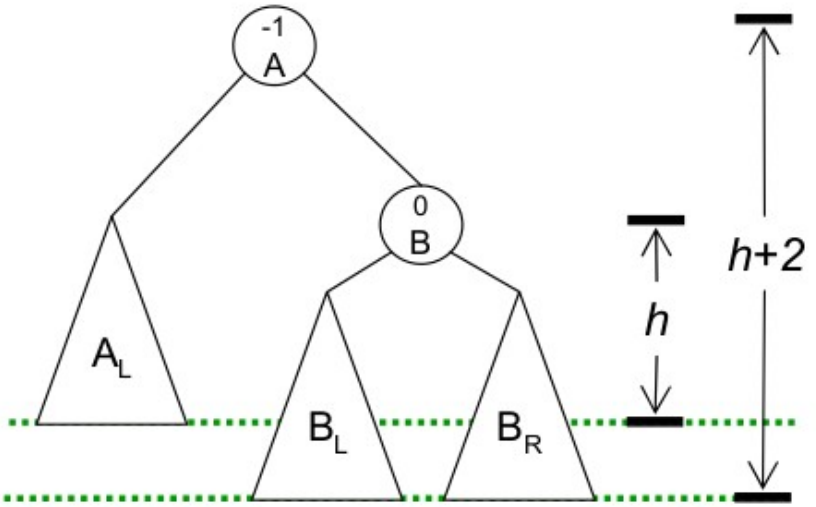

*Subárvore desbalanceada após inserção*



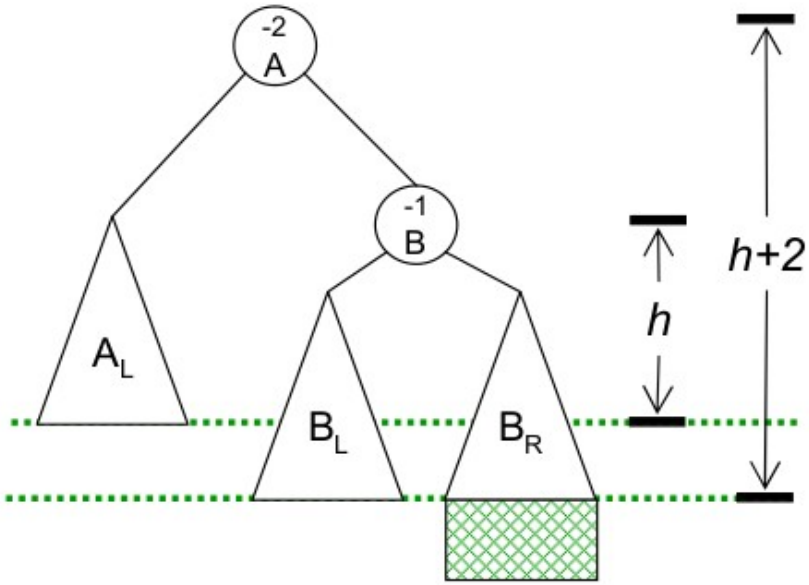

Altura de $B_R$ aumenta para *h*+1

# Rotação RR

*Subárvore balanceada*

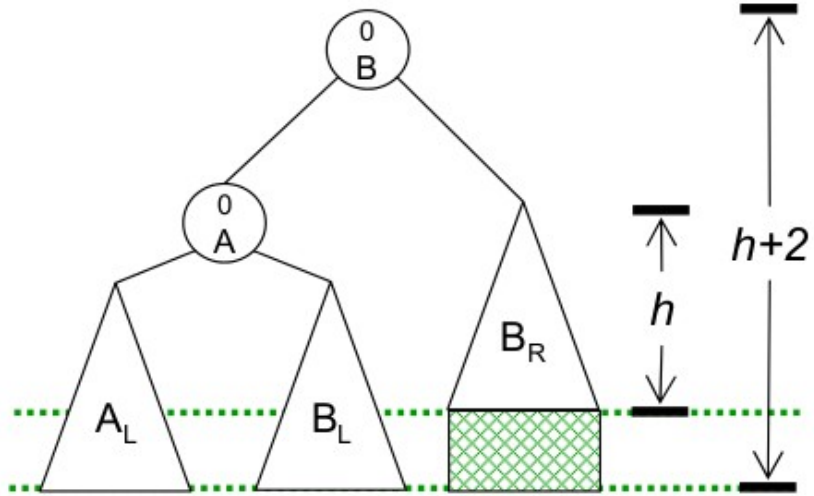



*Subárvore desbalanceada após inserção*

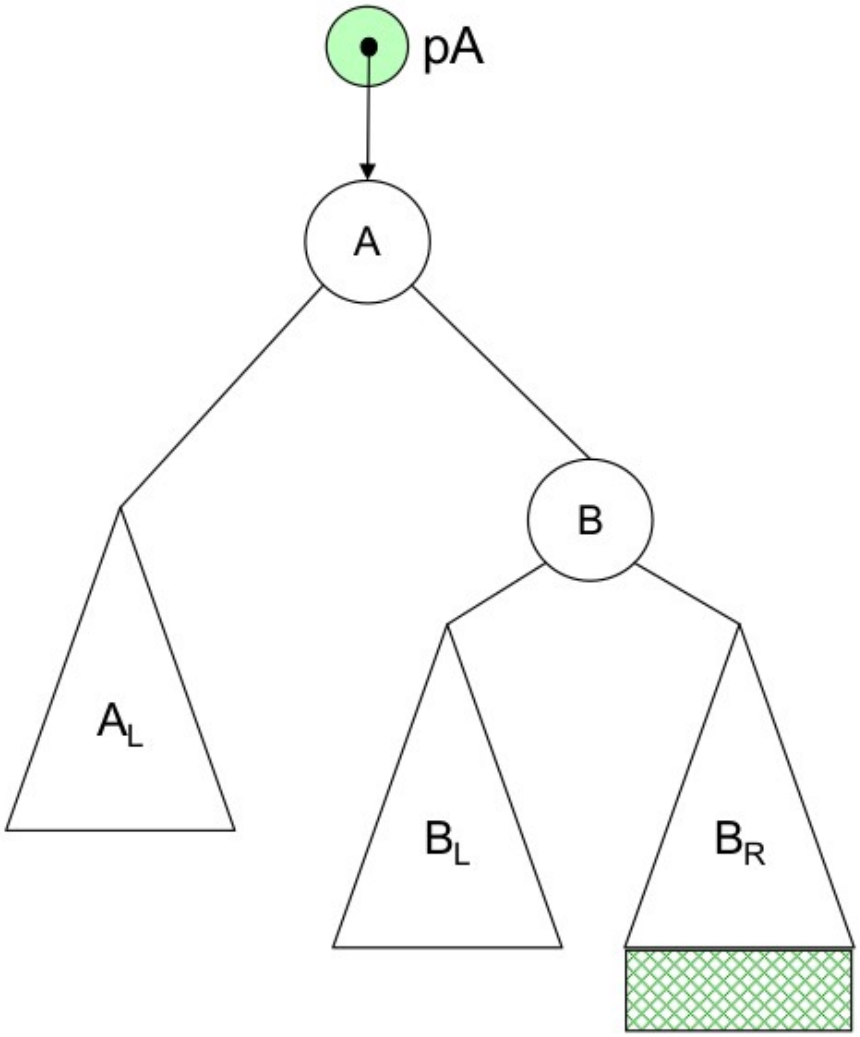



Altura de $B_R$ aumenta para $h$+1

*Subárvore rebalanceada*

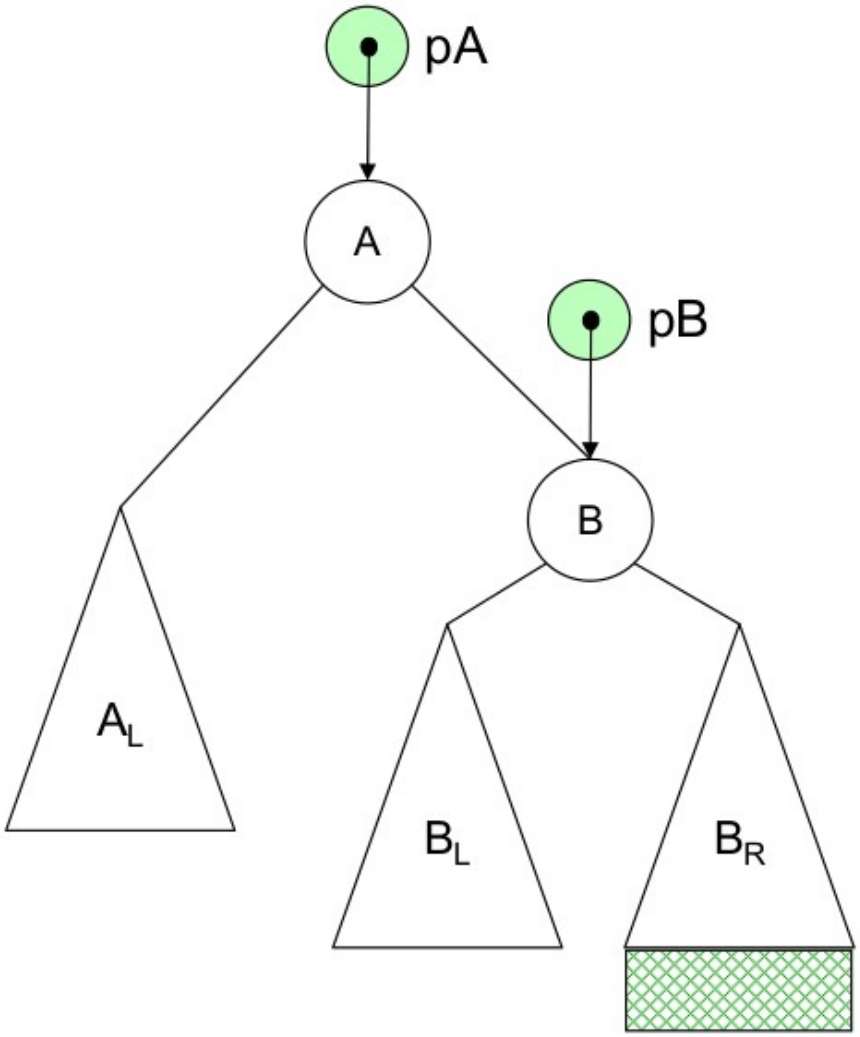


# Rotação RR



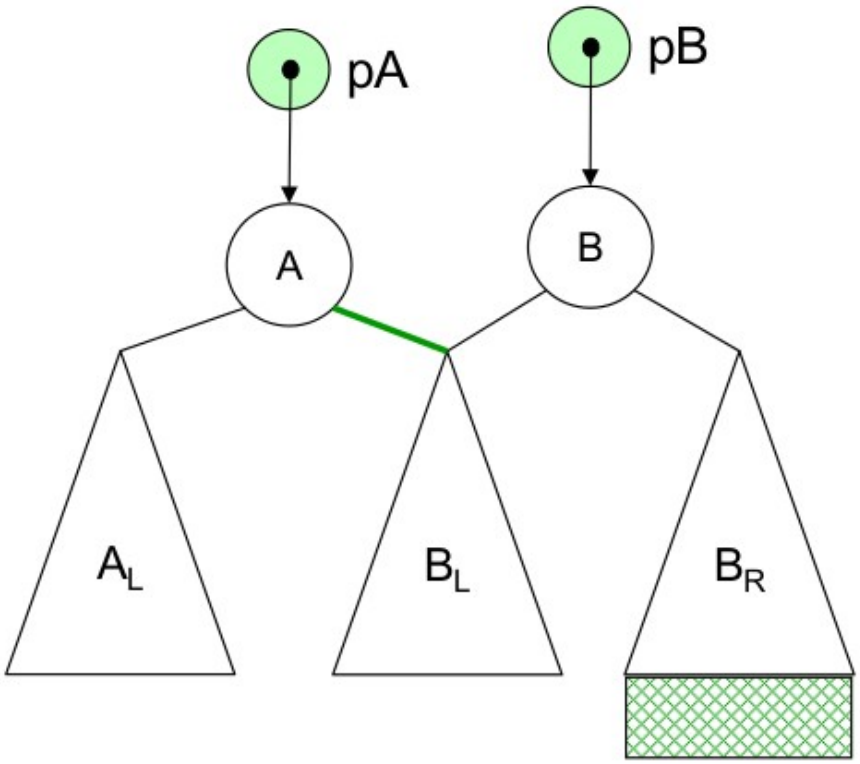

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->RightNode;
  - pA->RightNode = pB->LeftNode;
  - pB->LeftNode = pA;
  - pA = pB;

# Rotação RR



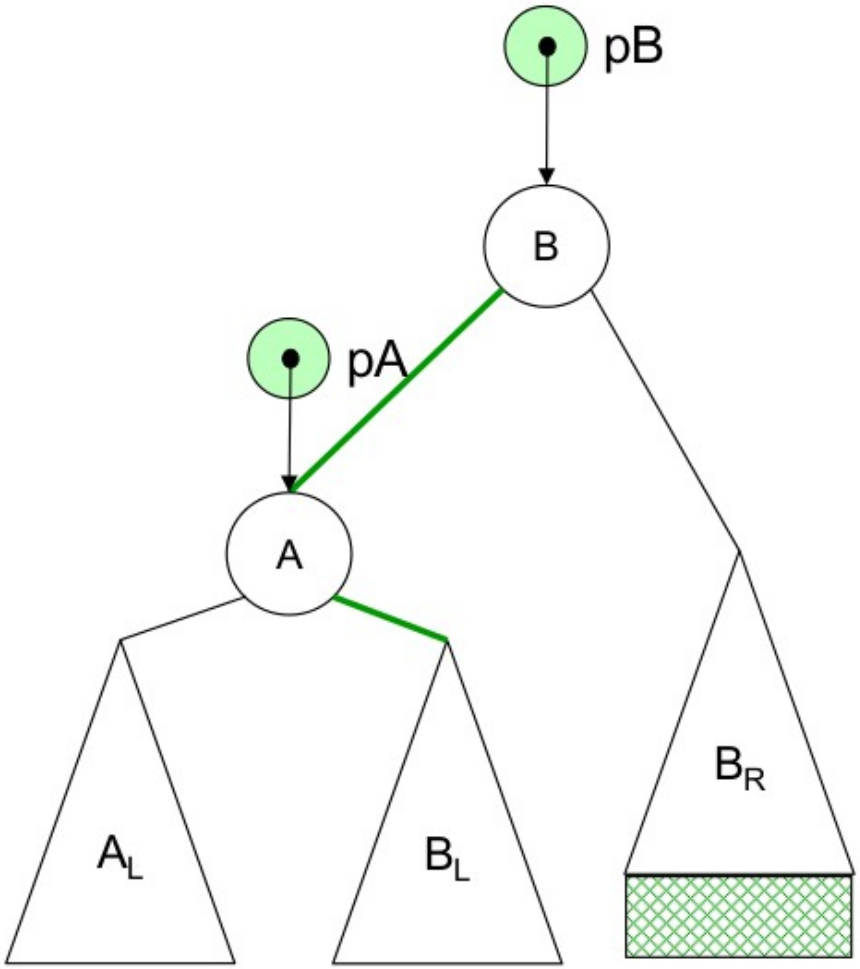

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - **pB = pA->RightNode;**
  - pA->RightNode = pB->LeftNode;
  - pB->LeftNode = pA;
  - pA = pB;

# Rotação RR



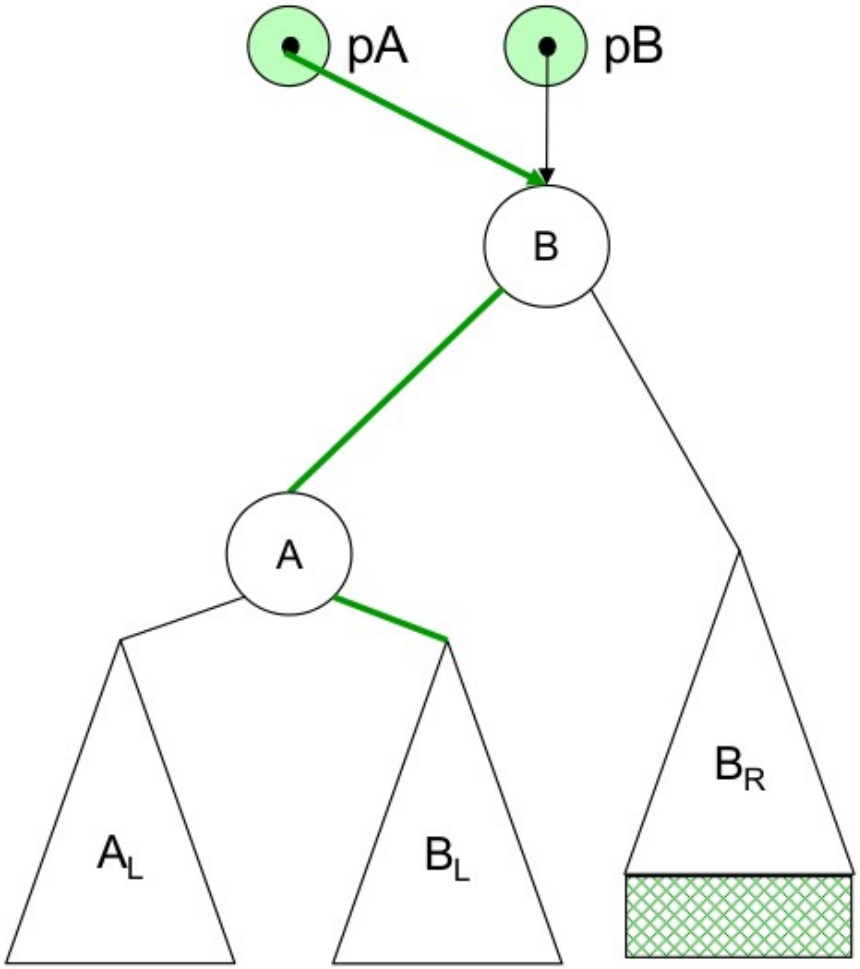

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->RightNode;
  - **pA->RightNode = pB->LeftNode;**
  - pB->LeftNode = pA;
  - pA = pB;

# Rotação RR




- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->RightNode;
  - pA->RightNode = pB->LeftNode;
  - **pB->LeftNode = pA;**
  - pA = pB;

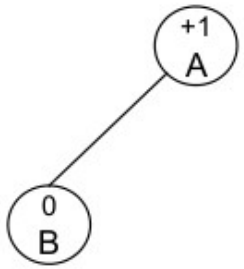
# Rotação RR



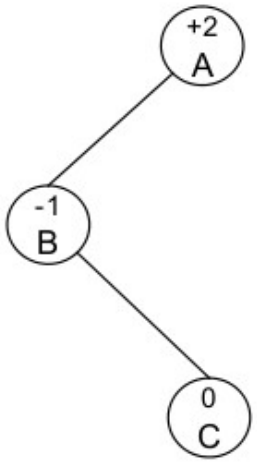

- Assumindo pA e pB ponteiros para as subárvores com raízes A e B:
  - pB = pA->RightNode;
  - pA->RightNode = pB->LeftNode;
  - pB->LeftNode = pA;
  - **pA = pB;**

# Rotação LR (a)

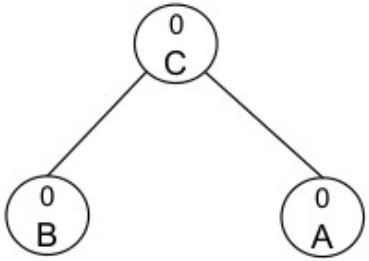
*Subárvore balanceada*



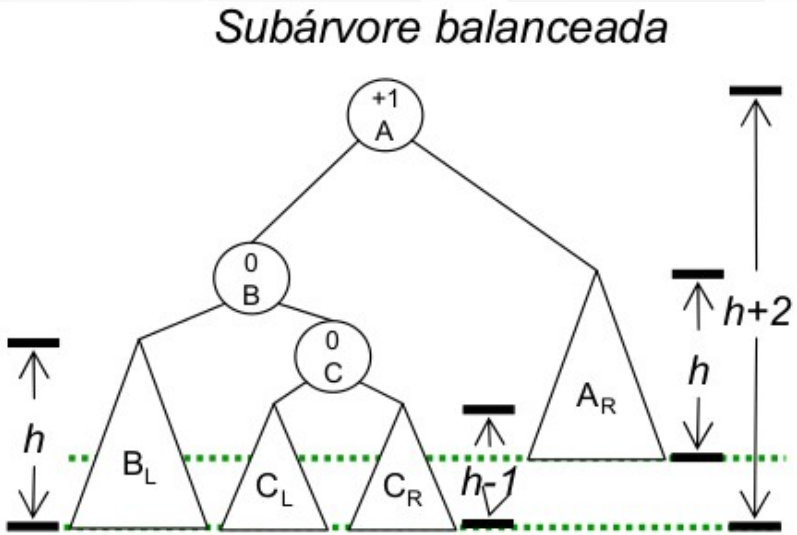

*Subárvore desbalanceada após inserção*




*Subárvore rebalanceada*

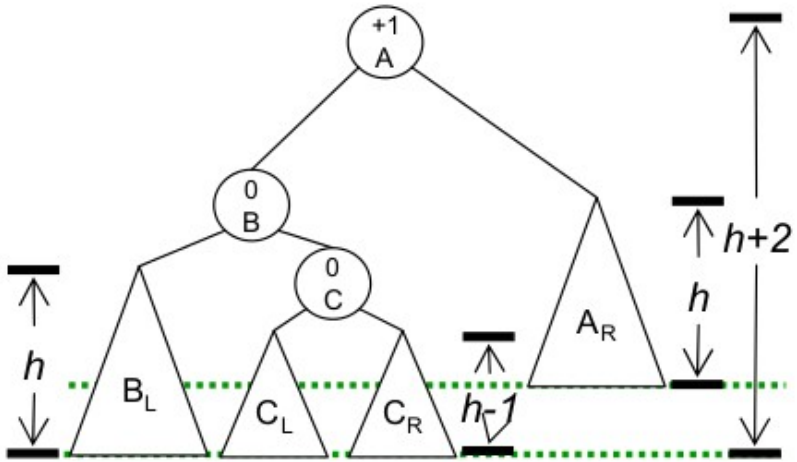


# Rotação LR (b)

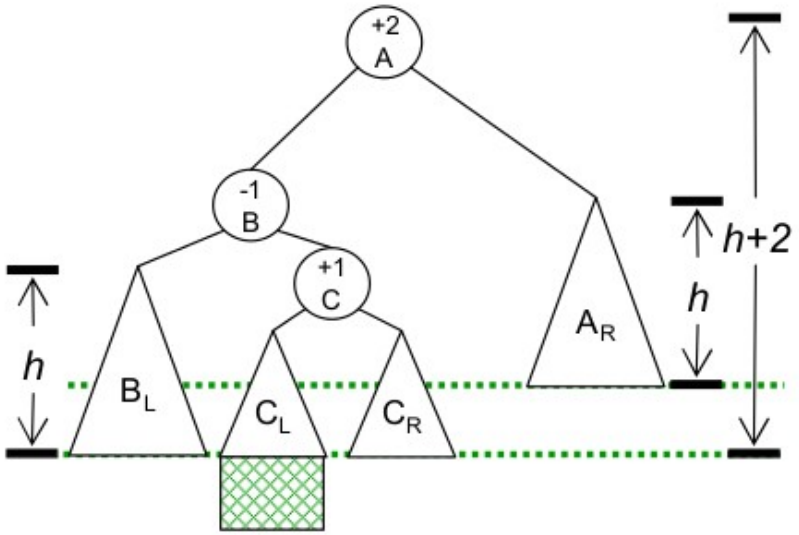


# Rotação LR (b)

*Subárvore balanceada*

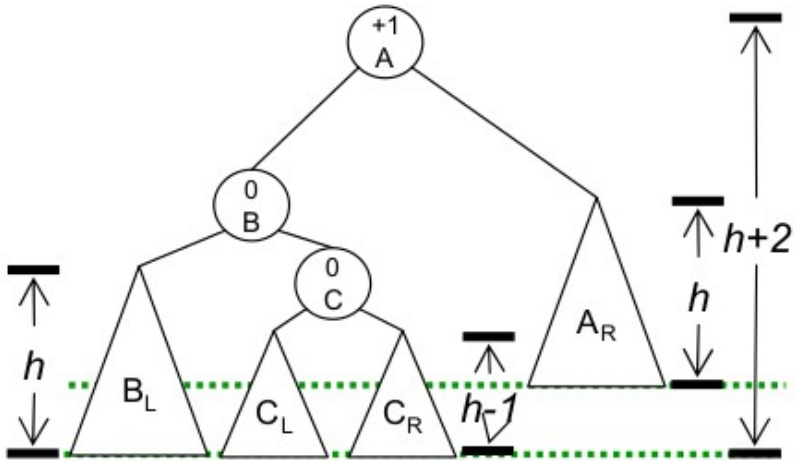



*Subárvore desbalanceada após inserção*

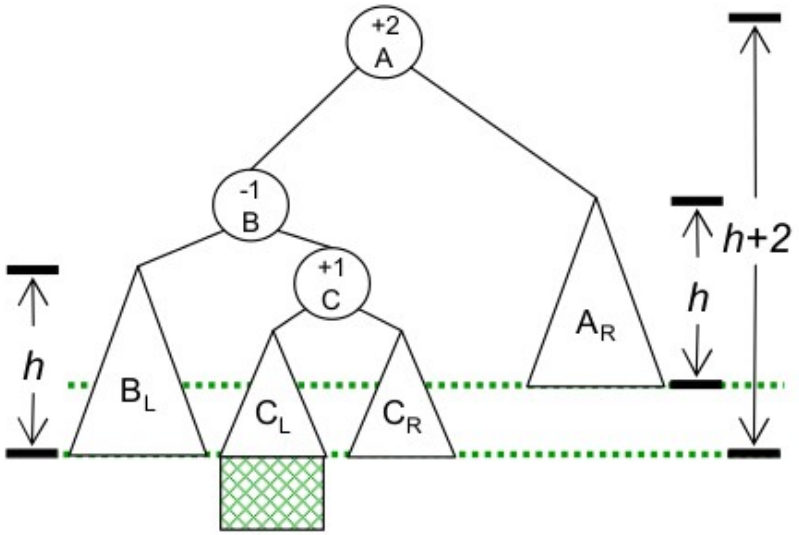


# Rotação LR (b)

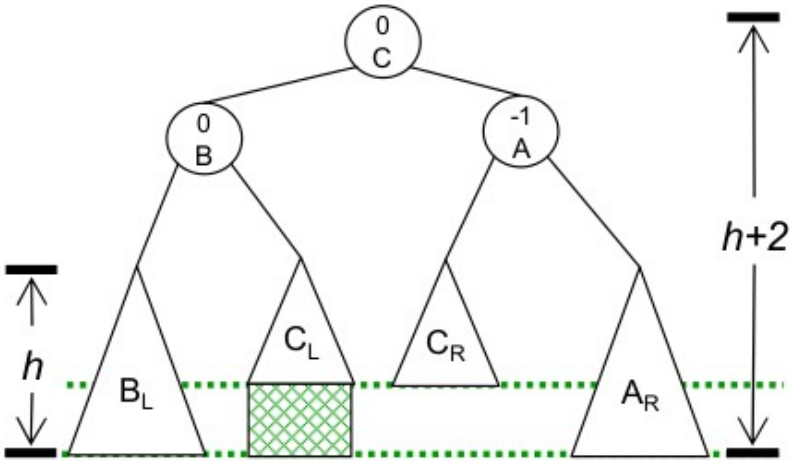
*Subárvore balanceada*



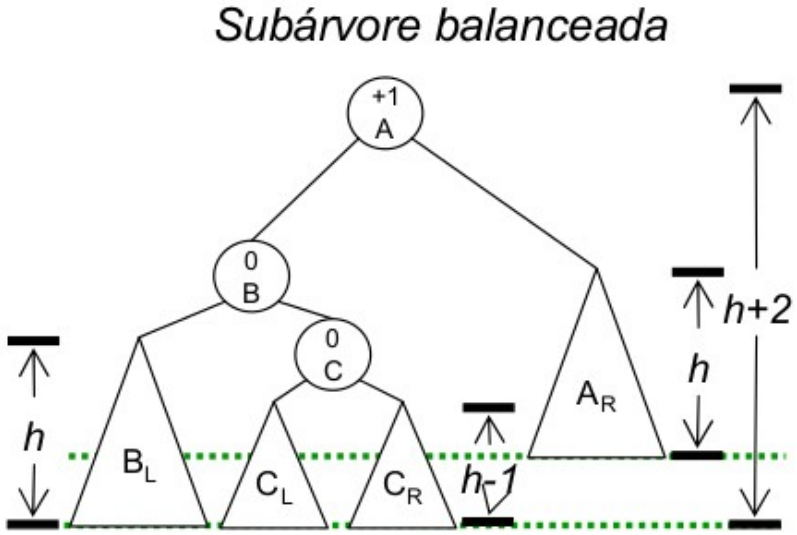

*Subárvore desbalanceada após inserção*




*Subárvore rebalanceada*

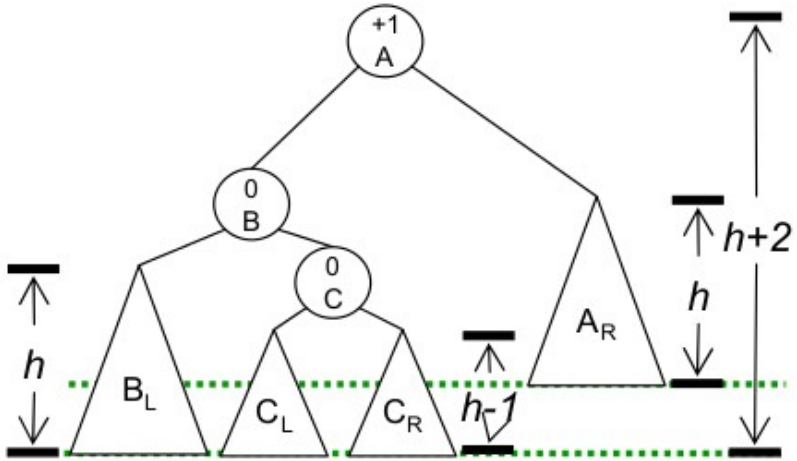


# Rotação LR (c)

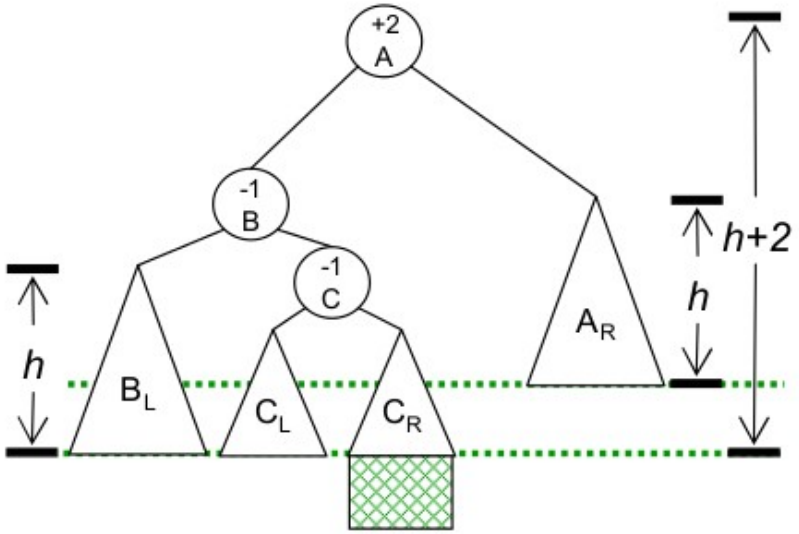


# Rotação LR (c)

*Subárvore balanceada*

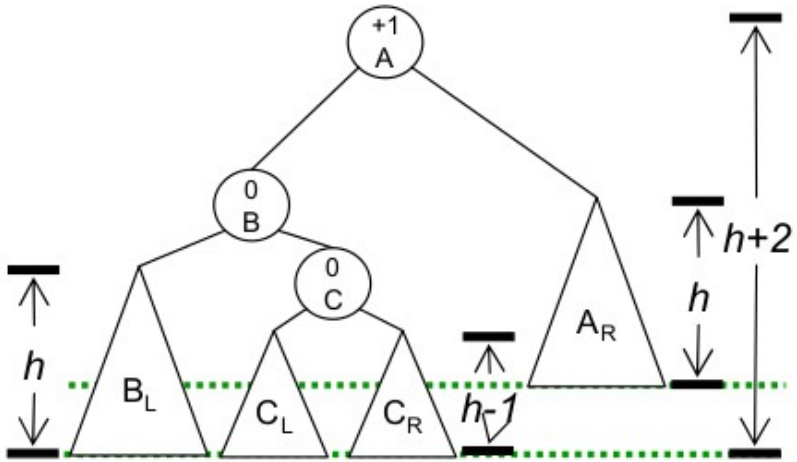



*Subárvore desbalanceada após inserção*

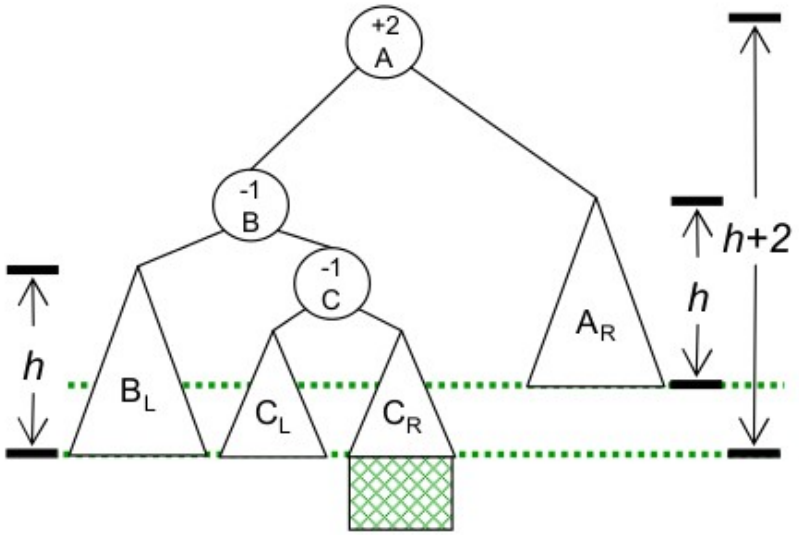


# Rotação LR (c)

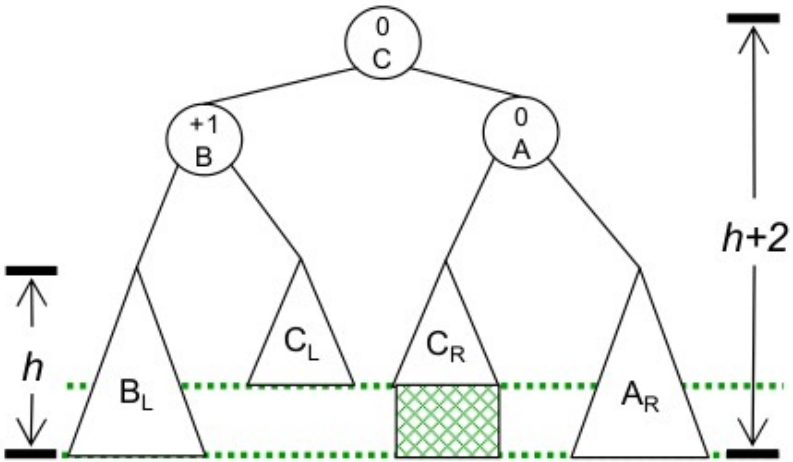
*Subárvore balanceada*



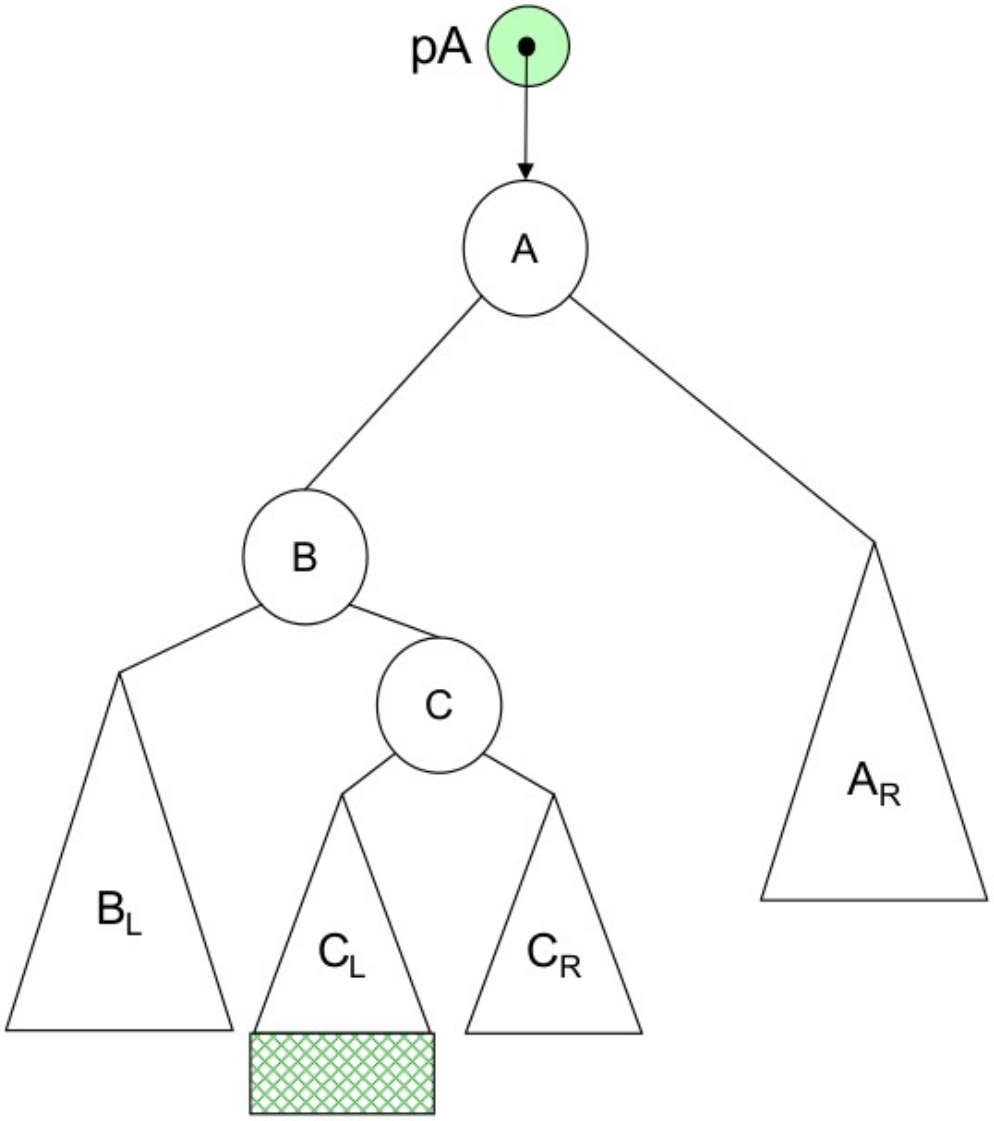

*Subárvore desbalanceada após inserção*




*Subárvore rebalanceada*

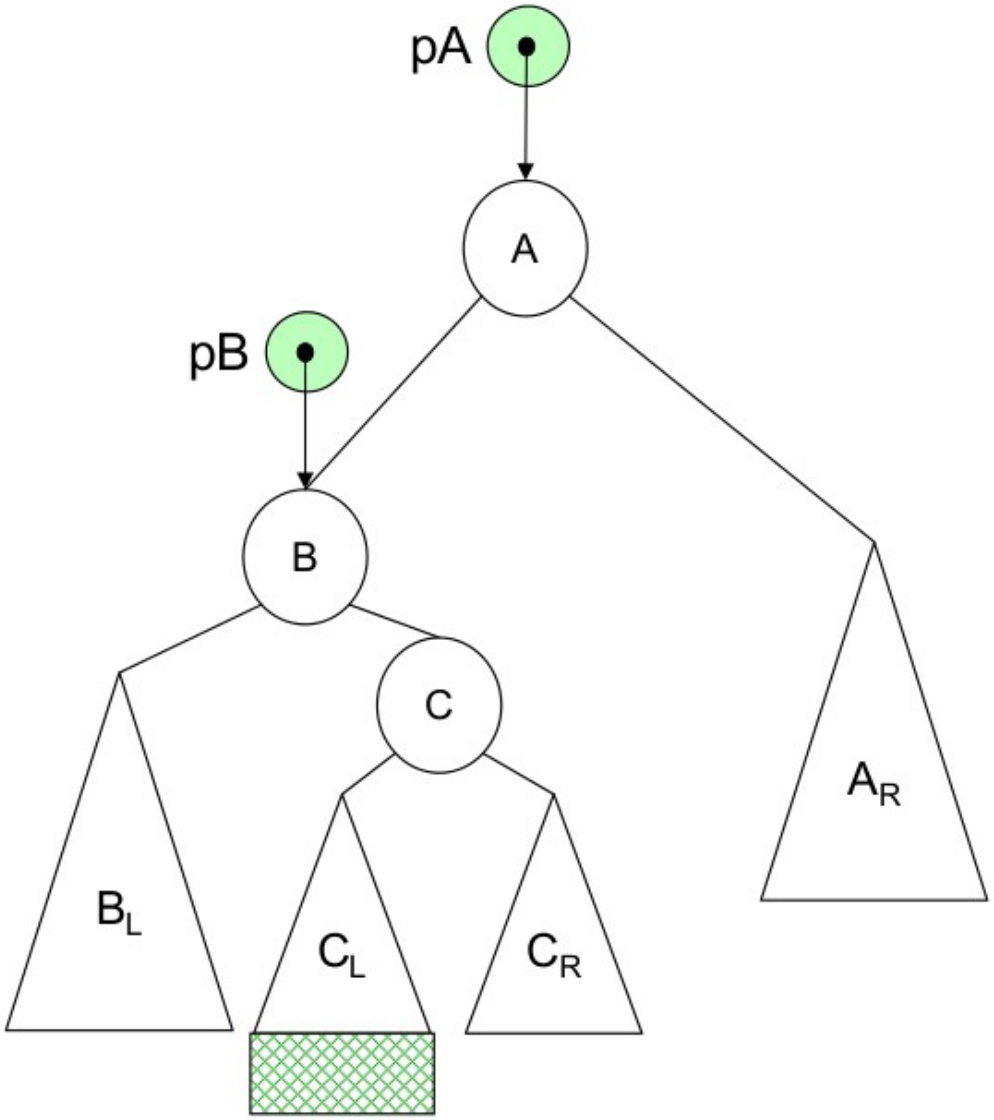


# Rotação LR



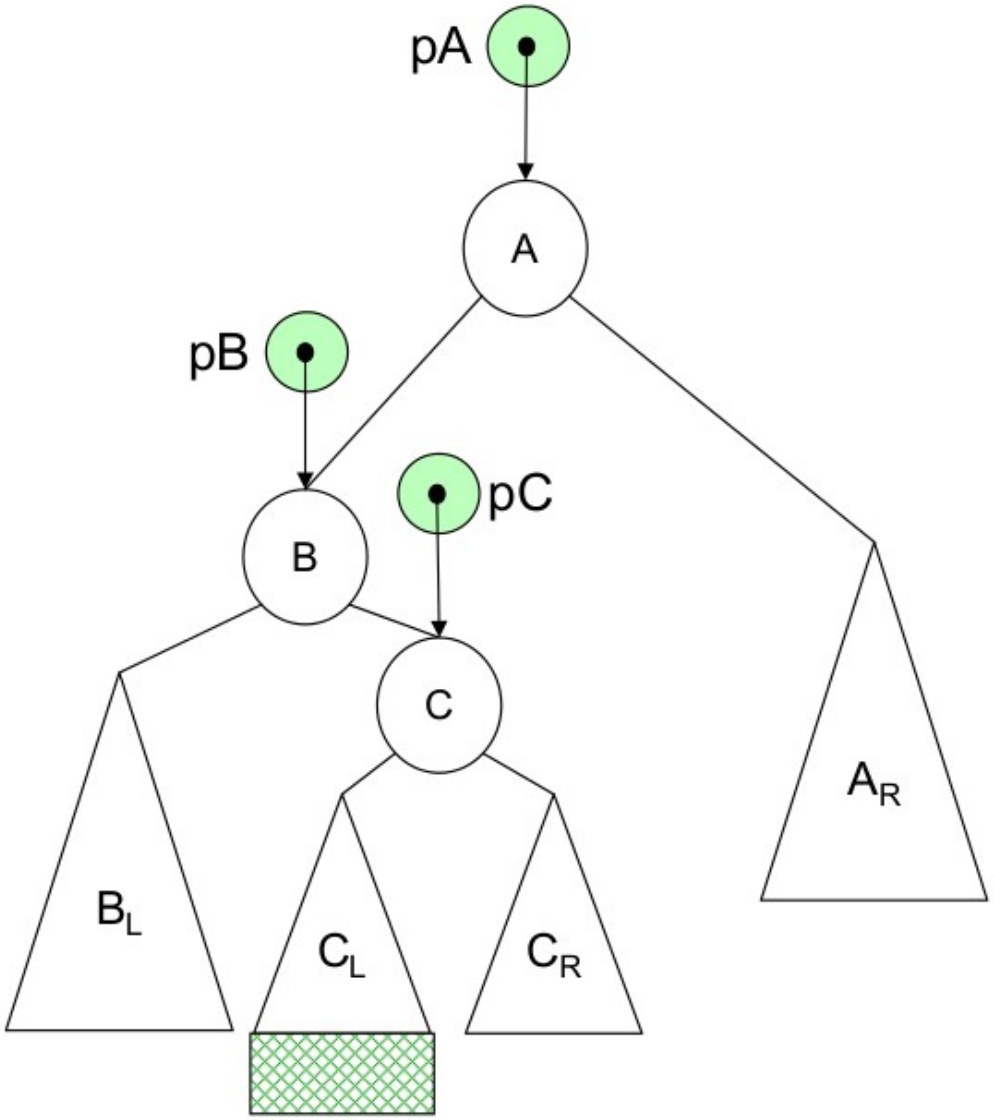

- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - pB = pA->LeftNode;
  - pC = pB->RightNode;
  - pB->RightNode = pC->LeftNode;
  - pC->LeftNode = pB;
  - pA->LeftNode = pC->RightNode;
  - pC->RightNode = pA;
  - pA = pC;

# Rotação LR



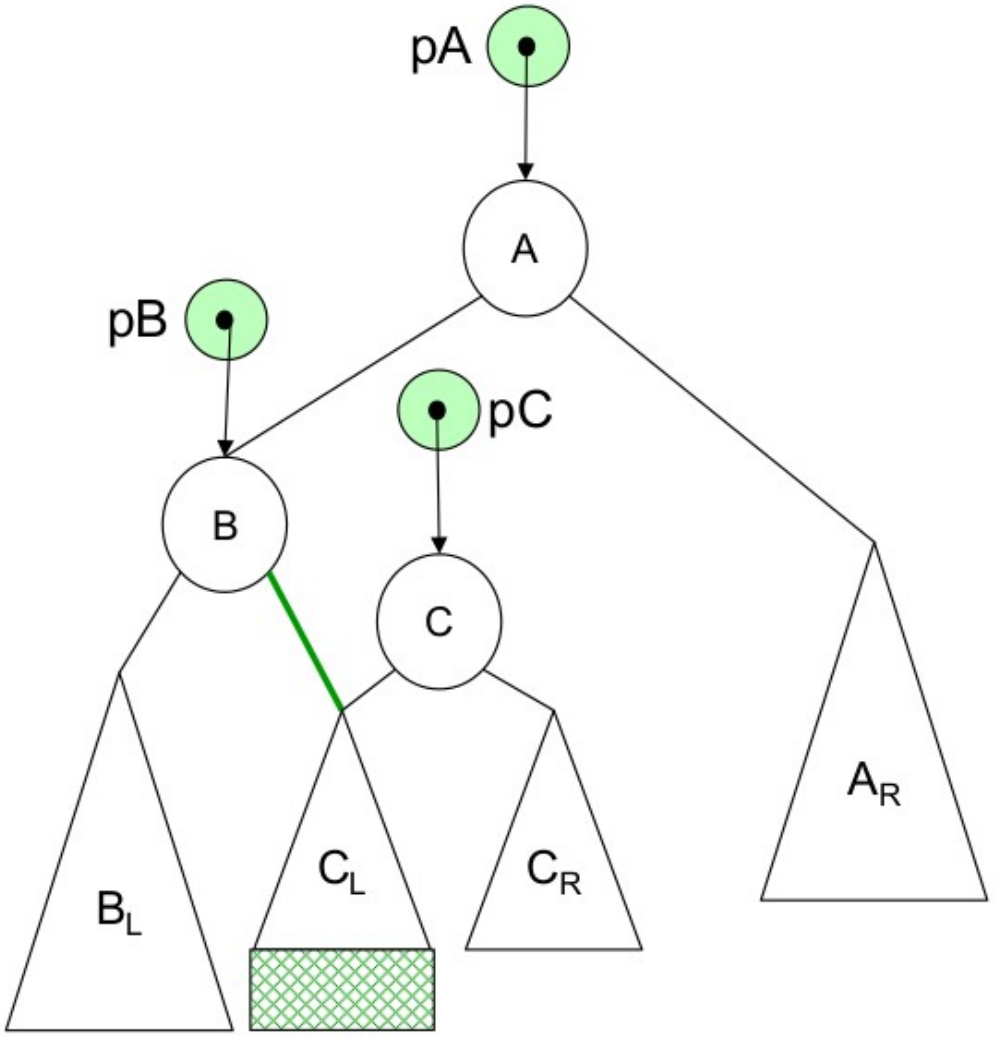

- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - **pB = pA->LeftNode;**
  - pC = pB->RightNode;
  - pB->RightNode = pC->LeftNode;
  - pC->LeftNode = pB;
  - pA->LeftNode = pC->RightNode;
  - pC->RightNode = pA;
  - pA = pC;

# Rotação LR



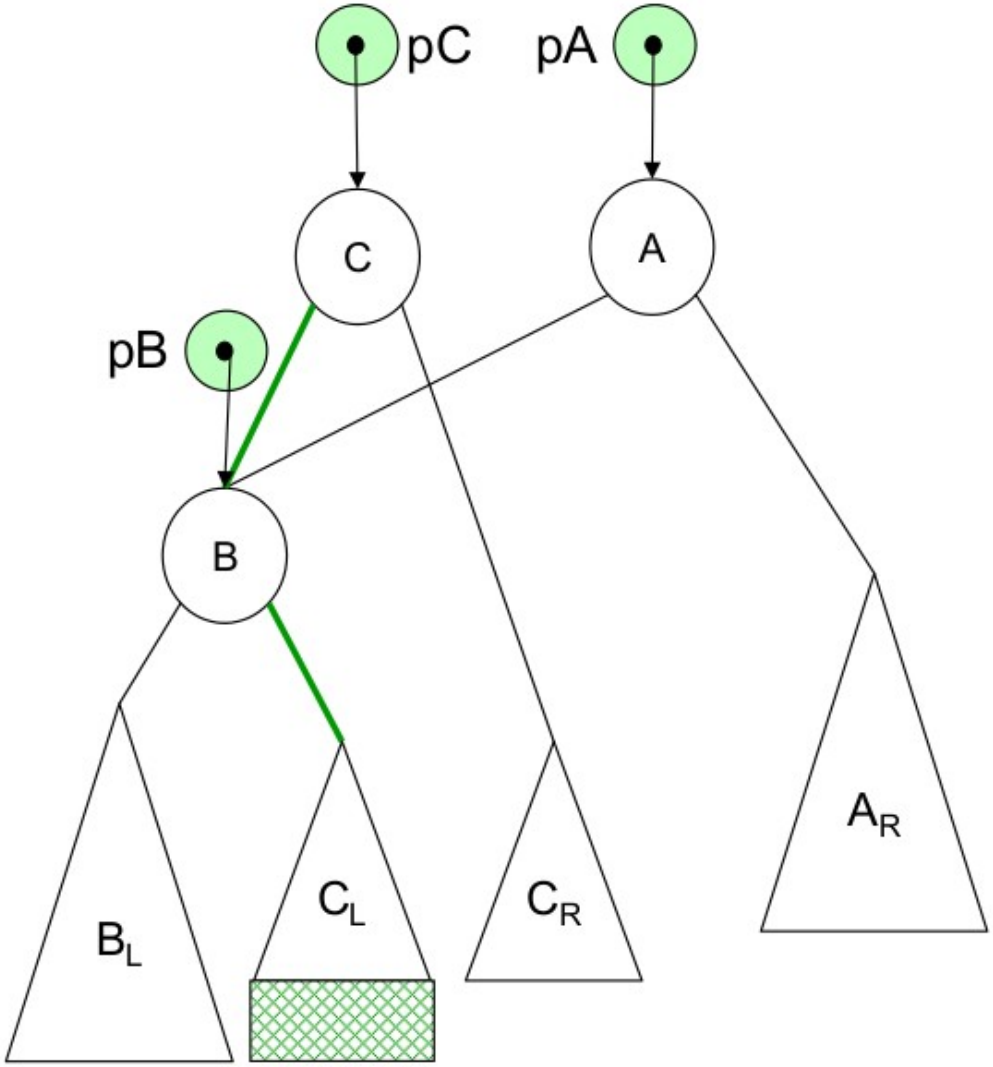

- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - pB = pA->LeftNode;
  - **pC = pB->RightNode;**
  - pB->RightNode = pC->LeftNode;
  - pC->LeftNode = pB;
  - pA->LeftNode = pC->RightNode;
  - pC->RightNode = pA;
  - pA = pC;

# Rotação LR



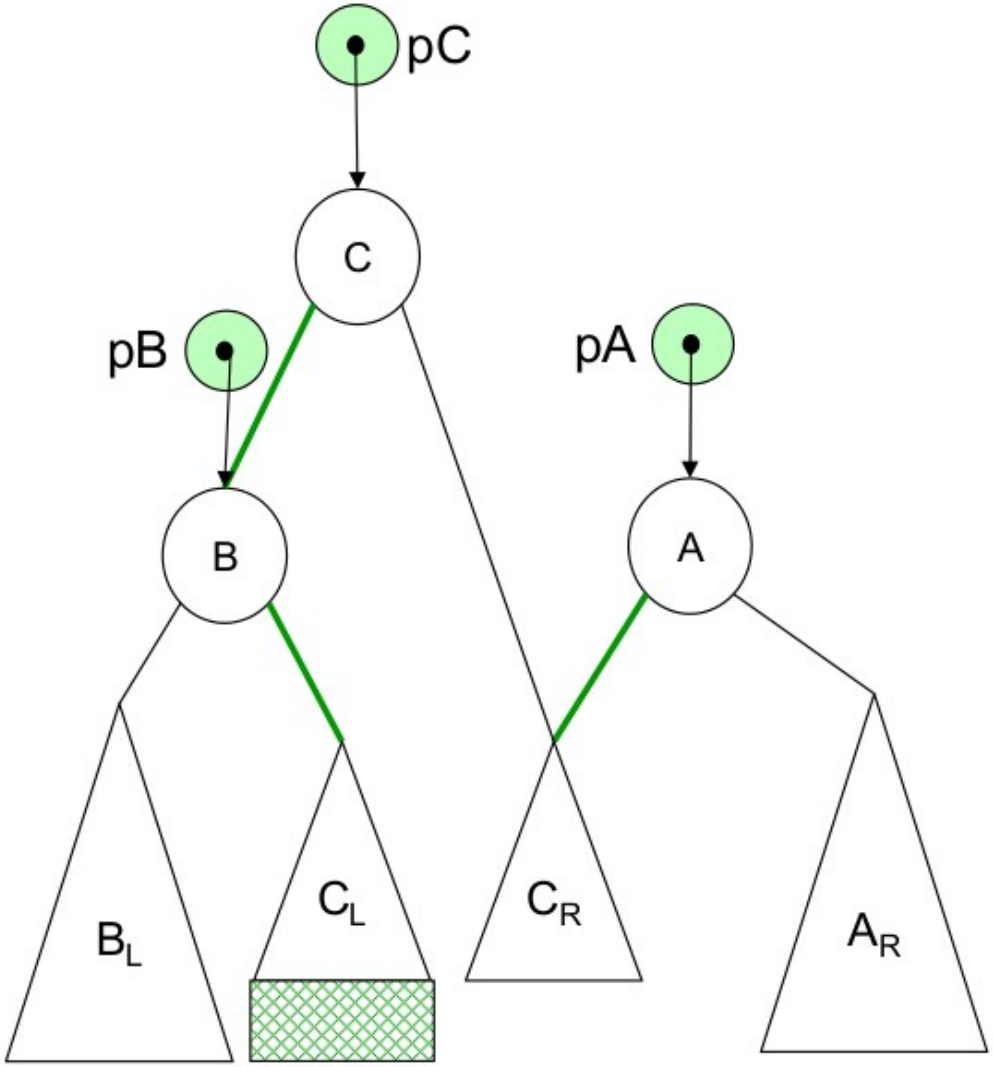

- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - pB = pA->LeftNode;
  - pC = pB->RightNode;
  - **pB->RightNode = pC->LeftNode;**
  - pC->LeftNode = pB;
  - pA->LeftNode = pC->RightNode;
  - pC->RightNode = pA;
  - pA = pC;

# Rotação LR



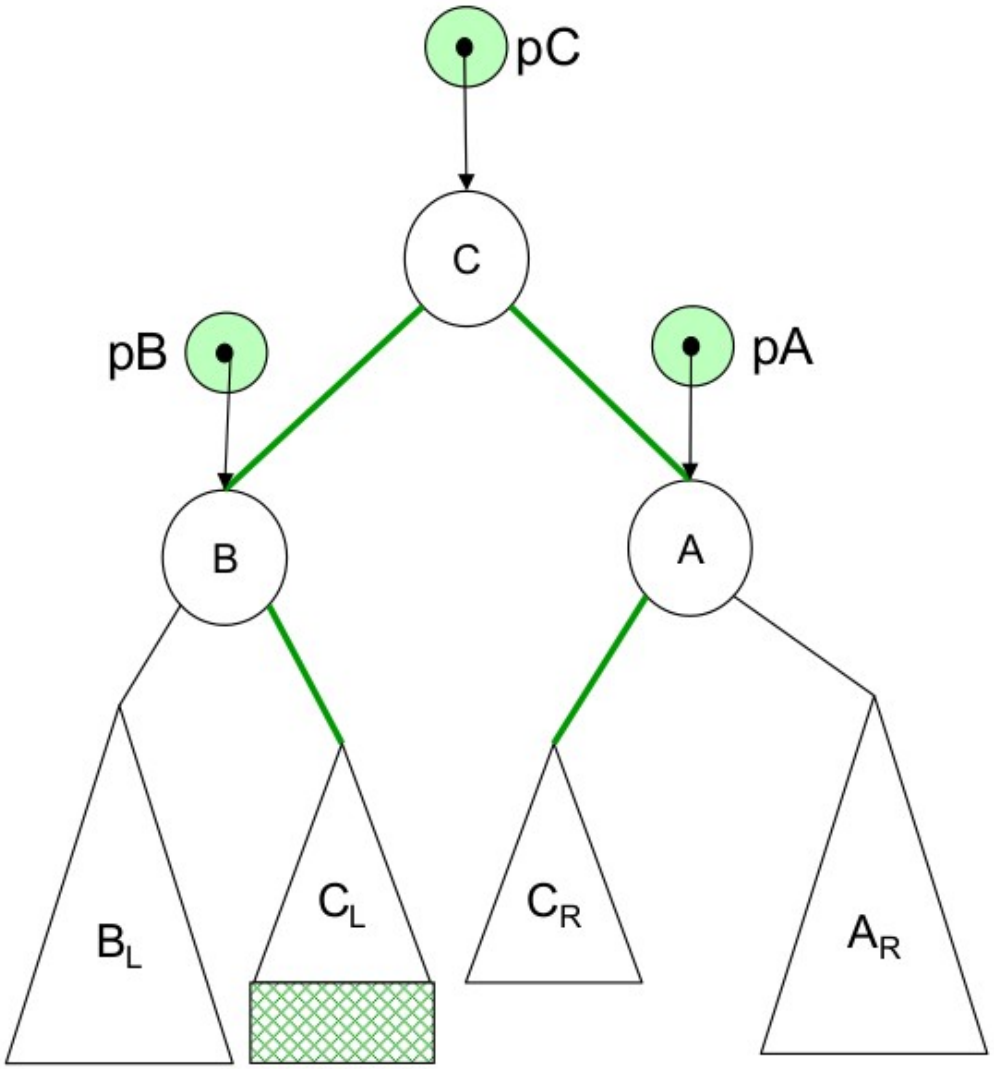

- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - pB = pA->LeftNode;
  - pC = pB->RightNode;
  - pB->RightNode = pC->LeftNode;
  - **pC->LeftNode = pB;**
  - pA->LeftNode = pC->RightNode;
  - pC->RightNode = pA;
  - pA = pC;

# Rotação LR



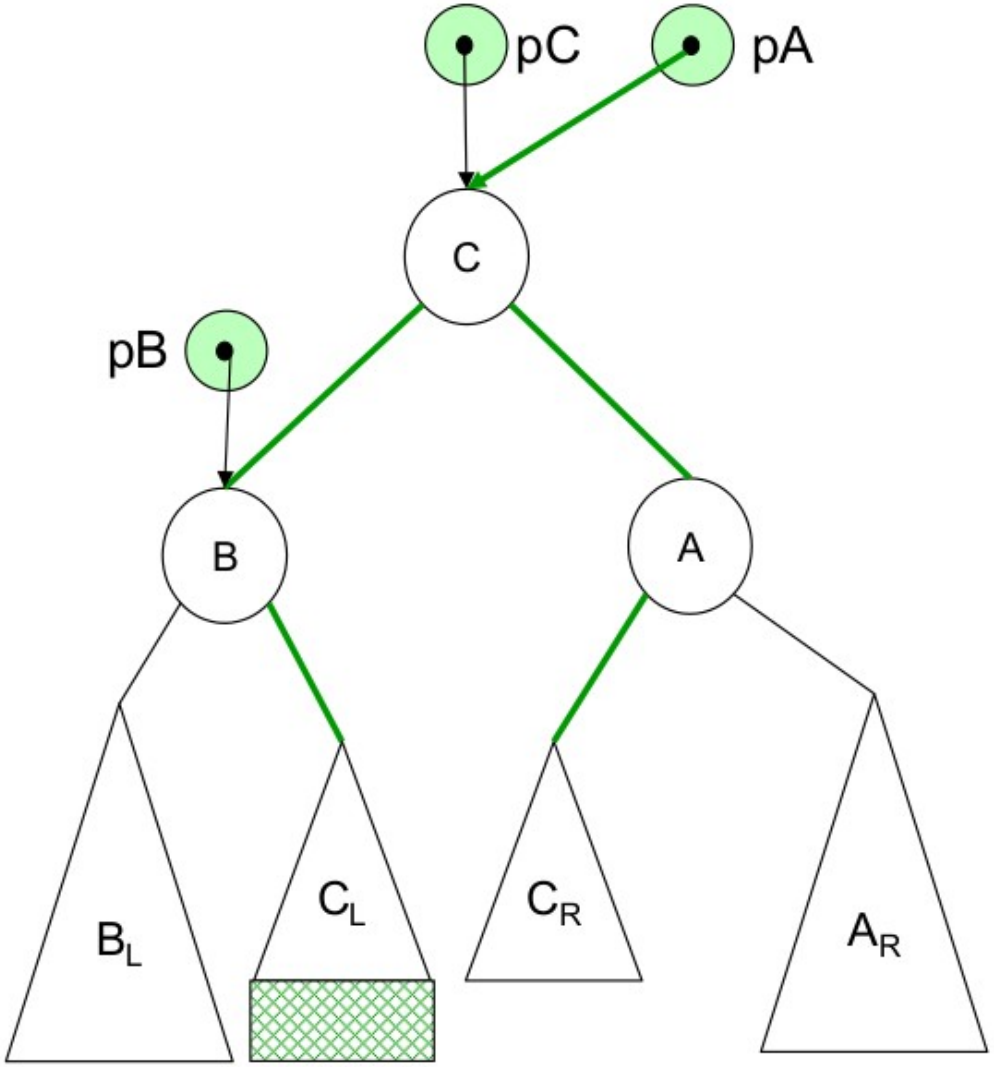

- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - pB = pA->LeftNode;
  - pC = pB->RightNode;
  - pB->RightNode = pC->LeftNode;
  - pC->LeftNode = pB;
  - **pA->LeftNode = pC->RightNode;**
  - pC->RightNode = pA;
  - pA = pC;

# Rotação LR




- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - pB = pA->LeftNode;
  - pC = pB->RightNode;
  - pB->RightNode = pC->LeftNode;
  - pC->LeftNode = pB;
  - pA->LeftNode = pC->RightNode;
  - **pC->RightNode = pA;**
  - pA = pC;

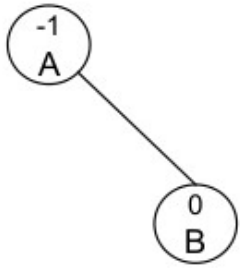
# Rotação LR



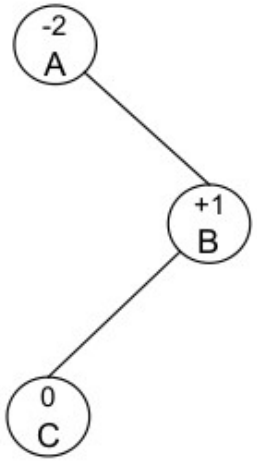

- Assumindo pA, pB e pC ponteiros para as subárvores com raízes A, B e C:
  - pB = pA->LeftNode;
  - pC = pB->RightNode;
  - pB->RightNode = pC->LeftNode;
  - pC->LeftNode = pB;
  - pA->LeftNode = pC->RightNode;
  - pC->RightNode = pA;
  - **pA = pC;**

# Rotação RL (a)

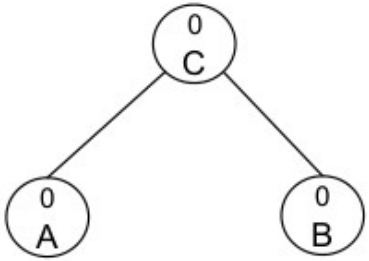
*Subárvore balanceada*



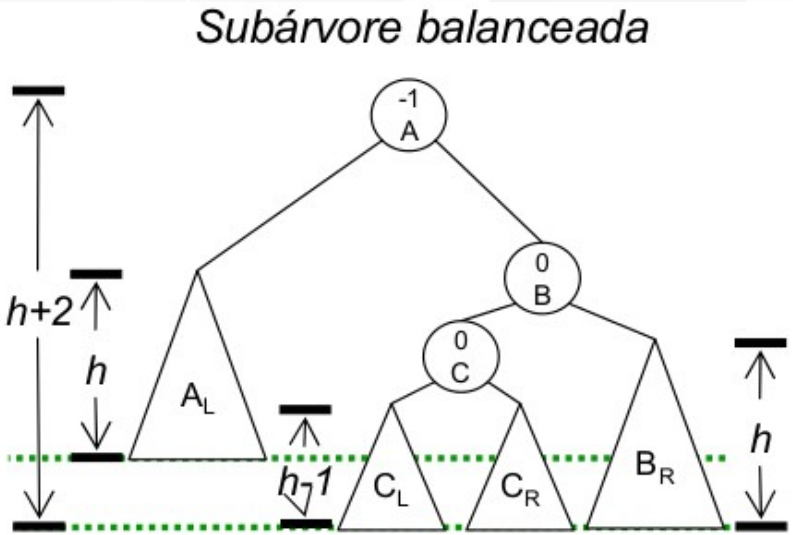

*Subárvore desbalanceada após inserção*




*Subárvore rebalanceada*

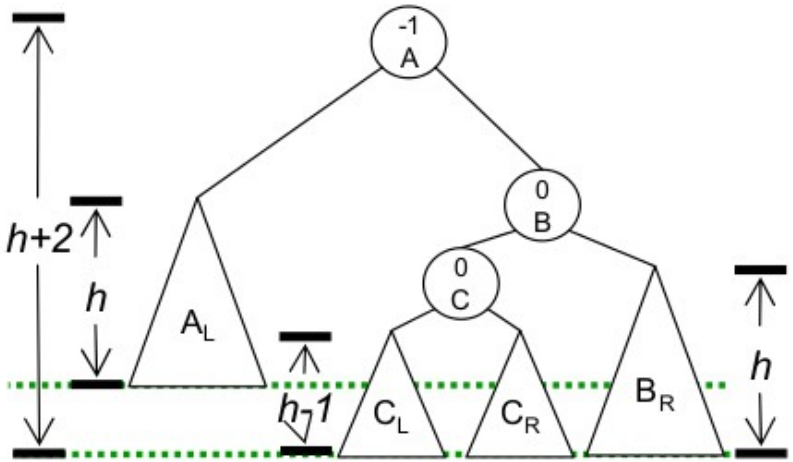


# Rotação RL (b)

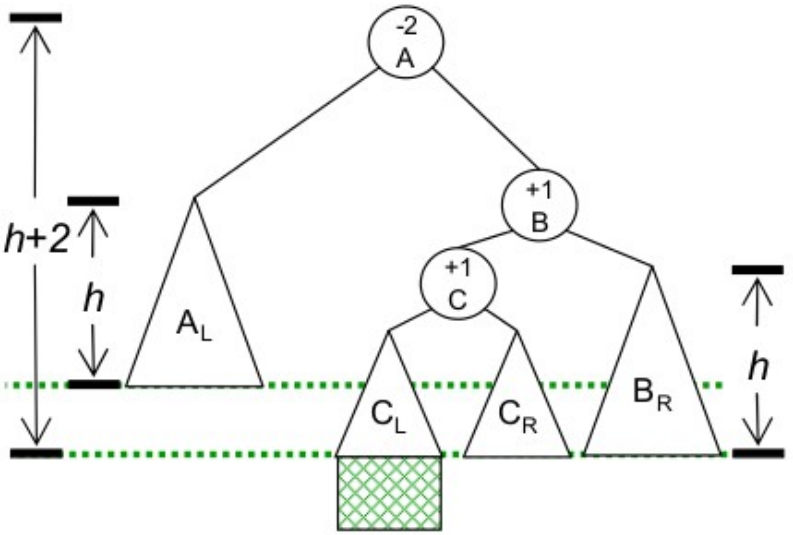


# Rotação RL (b)

*Subárvore balanceada*

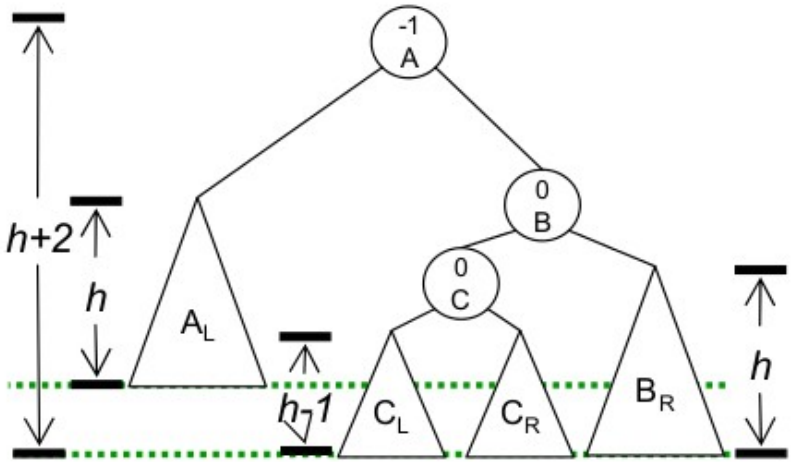



*Subárvore desbalanceada após inserção*

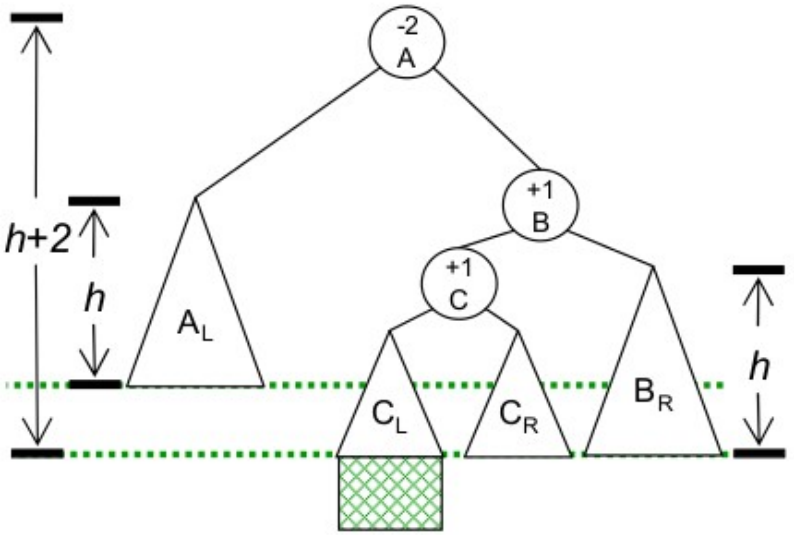


# Rotação RL (b)

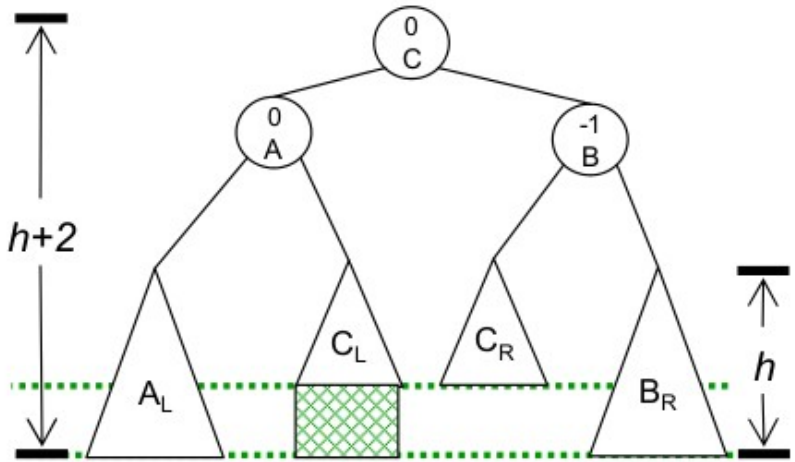
*Subárvore balanceada*




*Subárvore desbalanceada após inserção*

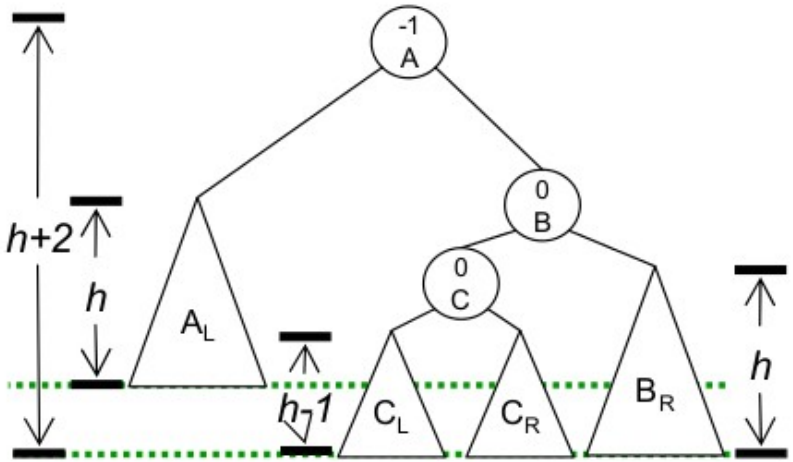



*Subárvore rebalanceada*

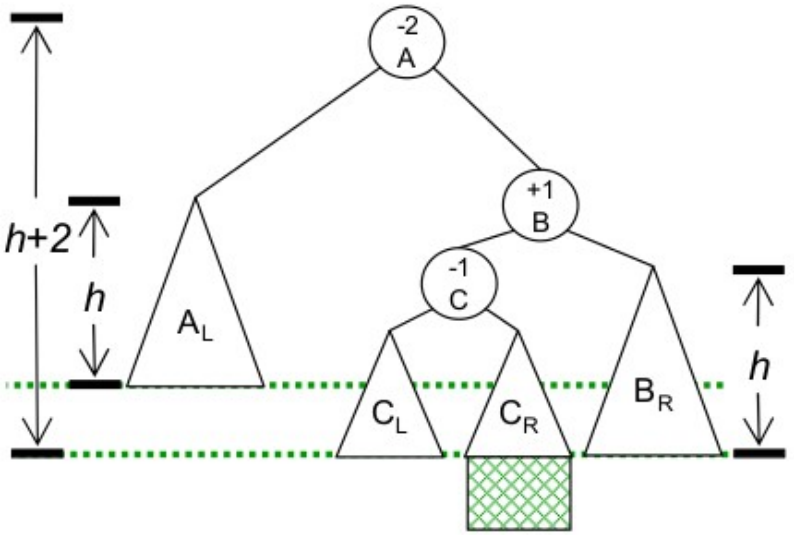


# Rotação RL (c)

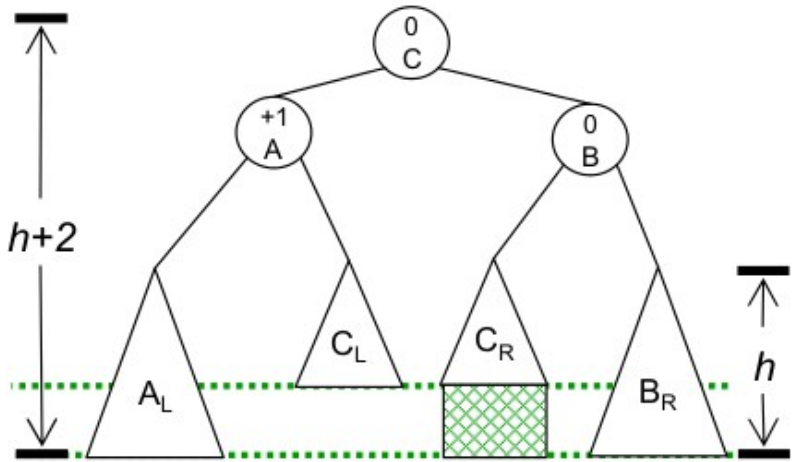
*Subárvore balanceada*




*Subárvore desbalanceada após inserção*

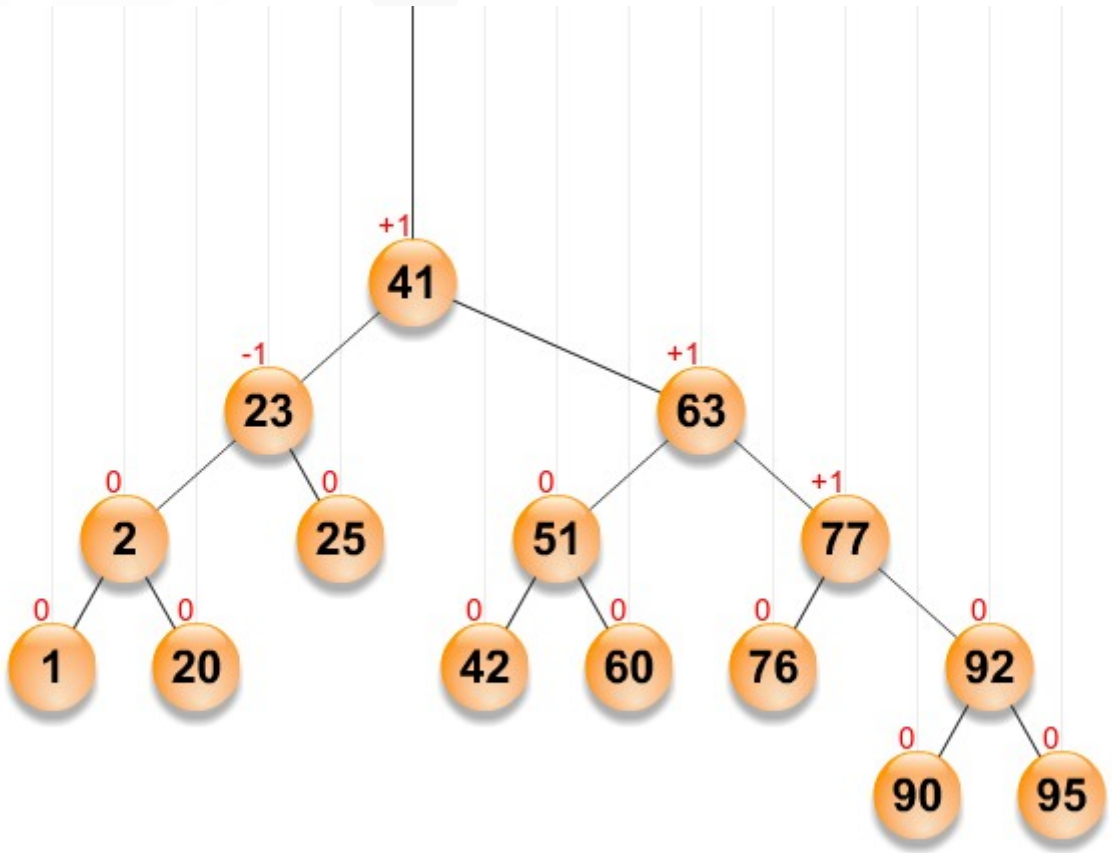



*Subárvore rebalanceada*



# Ferramenta de visualização: AVL

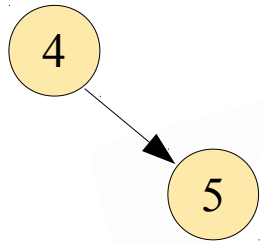
http://www.qmatica.com/DataStructures/Trees/AVL/AVLTree.html




-1*( $h_L - h_R$ )

# Atividade em aula

- Suponha que serão realizadas as seguintes inserções de chaves na árvore AVL ao lado:
  - 7
  - 2
  - 1
  - 3
  - 6



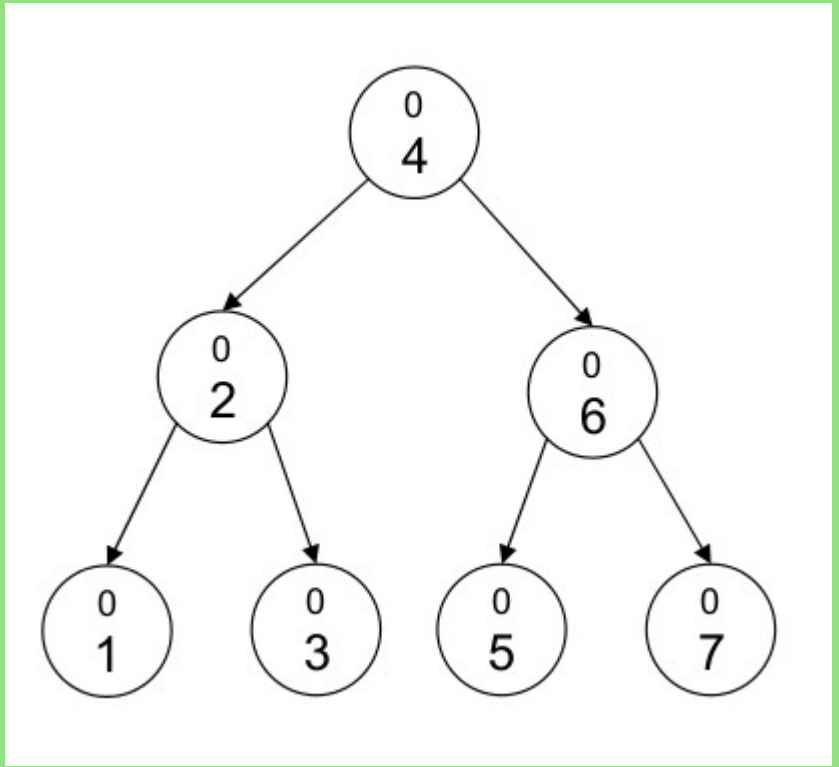

(a) Apresente a árvore AVL resultante (1 árvore)

(b) Indique o tipo de rotações consideradas em cada inserção

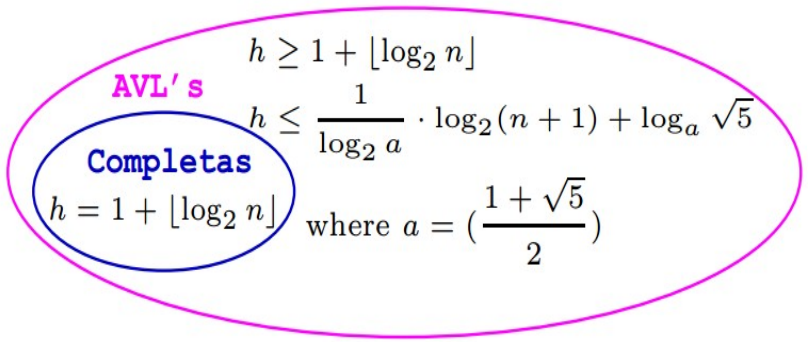
# Atividade em aula

**Operações**

- 7 : RR
- 2 : --
- 1 : LL
- 3 : LR
- 6 : RL



# AVL é uma árvores balanceada?

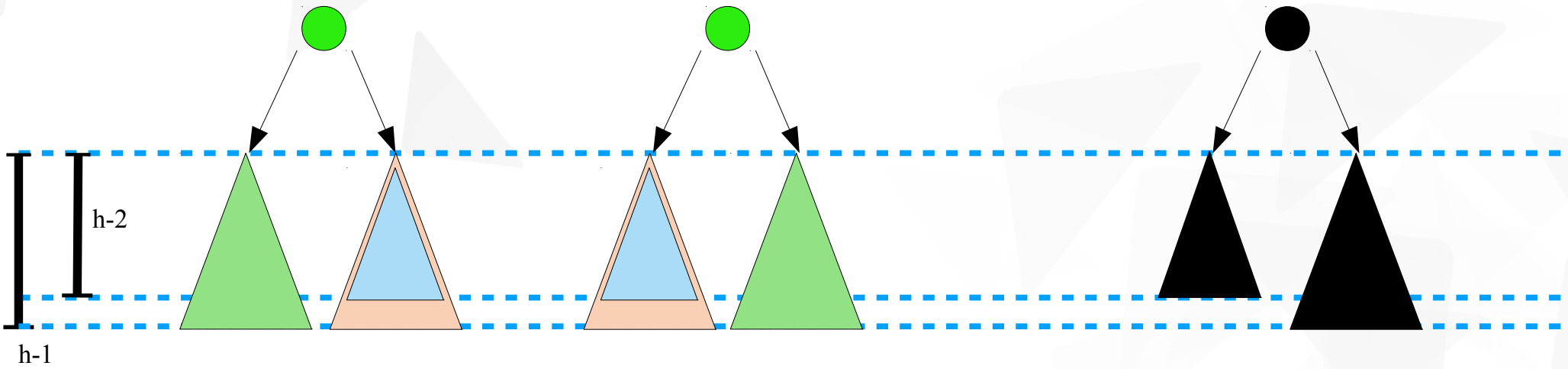


# Balanceamento de árvores AVL

- Uma árvore AVL de altura h é balanceada se h = O(log(n))
- Outra forma de pensar: **Dada uma árvore AVL de altura h, qual seria o valor mínimo possivel para n?**

# Balanceamento de árvores AVL

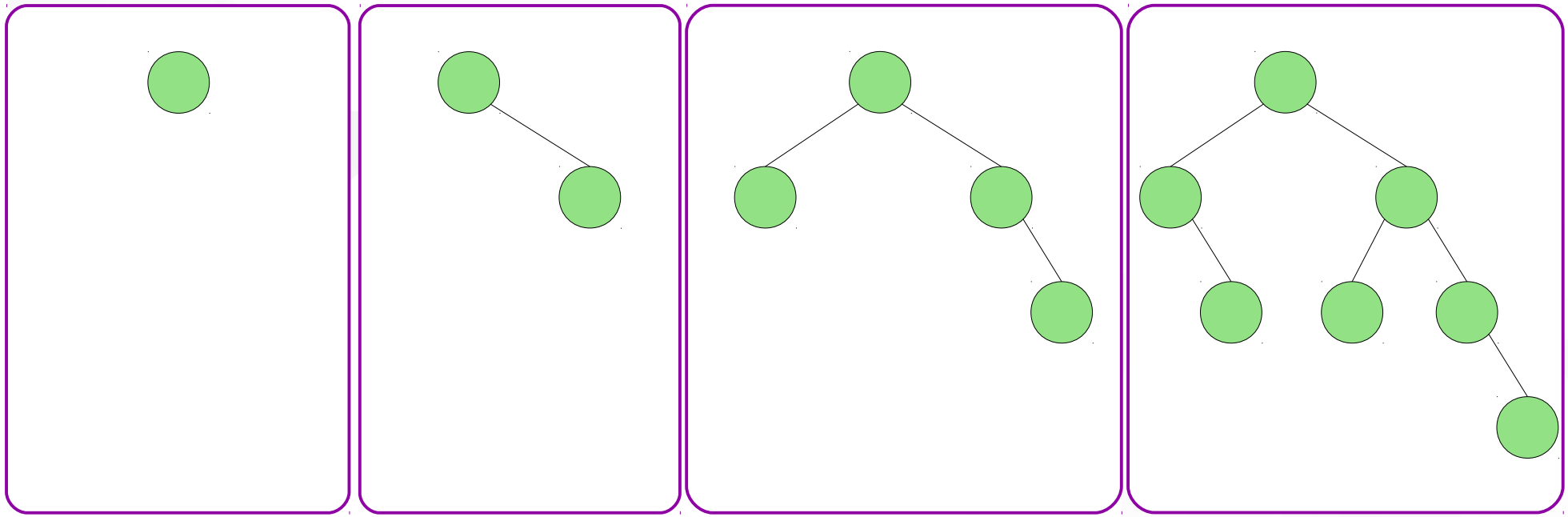
- Uma árvore AVL de altura h é balanceada se h = O(log(n))
- Outra forma de pensar: **Dada uma árvore AVL de altura h, qual seria o valor mínimo possivel para n?**



# Balanceamento de árvores AVL

- Seja $T_h$ uma árvore AVL com altura h e número mínimo de nós.

*Nesta definição → h=4*



$T_1$ $T_2$ $T_3$ $T_4$

# Balanceamento de árvores AVL

- Basta calcular um limite inferior do número de nós de Th. Seja |Th| o número de nós de Th.

$$\begin{cases} |T_h| = 0 & , \text{para } h = 0 \\ |T_h| = 1 & , \text{para } h = 1 \\ |T_h| = 1 + |T_{h-1}| + |T_{h-2}| & , \text{para } h > 1 \end{cases}$$

| $h$ | $\lvert T_h \rvert$ |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 4 |
| 4 | 7 |
| 5 | 12 |
| 6 | 20 |
| 7 | 33 |
| 8 | 54 |
| 9 | 88 |
| 10 | 143 |

# Analogia com a sequência de Fibonacci

$$\begin{cases} f_h = 0 & , \text{para } h = 0 \\ f_h = 1 & , \text{para } h = 1 \\ f_h = f_{h-1} + f_{h-2} & , \text{para } h > 1 \end{cases}$$

$$f_h = \frac{1}{\sqrt{5}} \left[ \left( \frac{1+\sqrt{5}}{2} \right)^h - \left( \frac{1-\sqrt{5}}{2} \right)^h \right]$$

$$|T_h| \geq f_h$$

# AVL

$|T_h| \geq f_h$

$$f_h = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^h - \left(\frac{1-\sqrt{5}}{2}\right)^h\right]$$

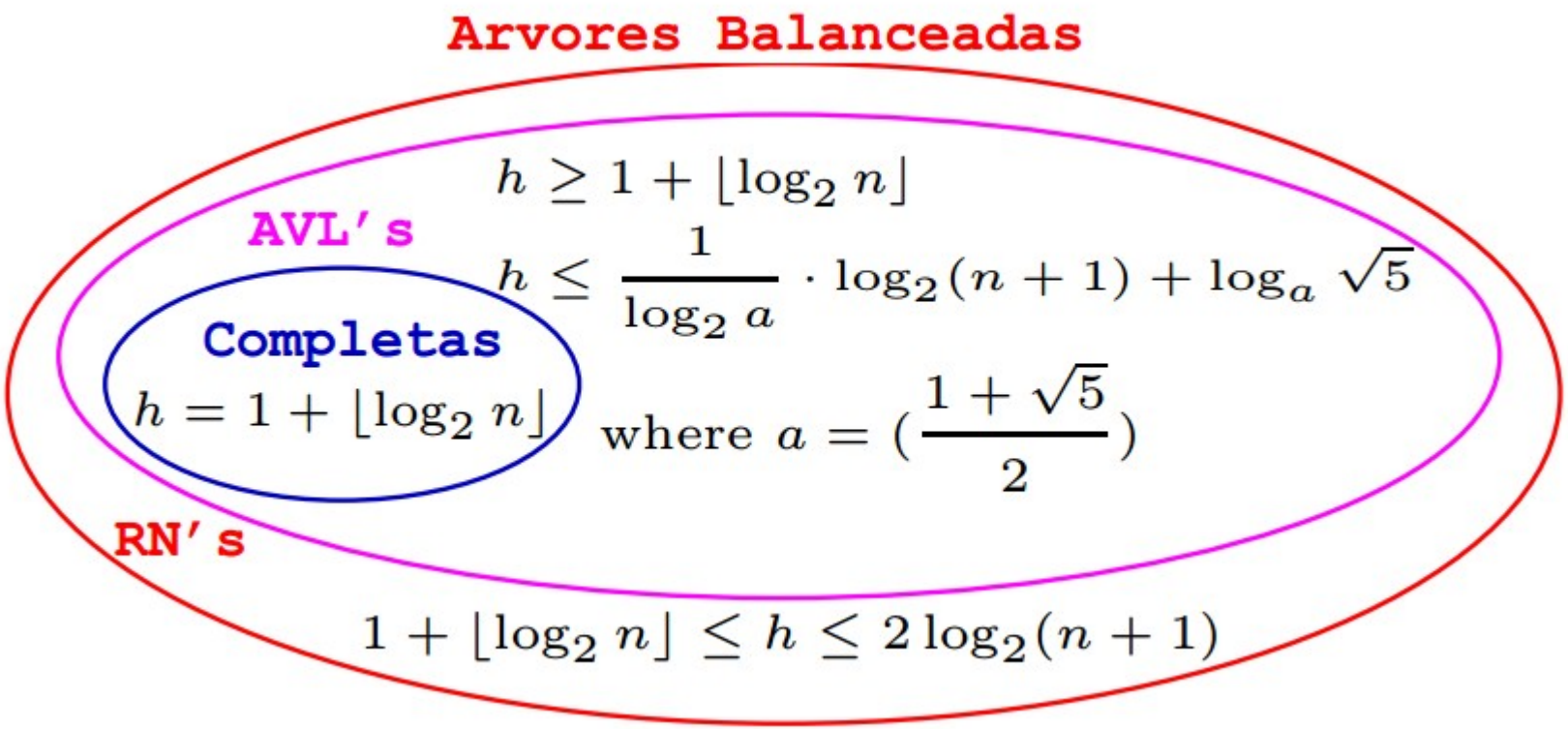
Como $\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^h < 1$ Temos $|T_h| > \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^h - 1$

$$h < \frac{1}{\log_2 a}\log_2(|T_h|+1) + \log_a(\sqrt{5}) \qquad a = \frac{1+\sqrt{5}}{2}$$

$$h = O(\log n)$$

AVL é uma árvore balanceada!

# AVL




•Um teorema provado por Adelson-Velskii e Landis **garante** que **a árvore balanceada nunca será 45% mais alta que a correspondente árvore perfeitamente balanceada**, independentemente do número de nós existentes.

# AVL

| n | Completa = C | AVL = A | A/C |
|---|---|---|---|
| 10 | 4.322 | 6.655 | 1.540 |
| 200 | 8.644 | 12.693 | 1.468 |
| 3000 | 12.551 | 18.311 | 1.459 |
| 40000 | 16.288 | 23.693 | 1.455 |
| 500000 | 19.932 | 28.942 | 1.452 |
| 6000000 | 23.517 | 34.106 | 1.450 |
| 70000000 | 27.061 | 39.211 | 1.449 |
| 800000000 | 30.575 | 44.273 | 1.448 |
| 9000000000 | 34.067 | 49.303 | 1.447 |
| 100000000000 | 37.541 | 54.307 | 1.447 |
| 1100000000000 | 41.001 | 59.290 | 1.446 |
| 12000000000000 | 44.448 | 64.256 | 1.446 |
| 130000000000000 | 47.886 | 69.207 | 1.445 |
| 1400000000000000 | 51.314 | 74.146 | 1.445 |
| 15000000000000000 | 54.736 | 79.074 | 1.445 |
| 160000000000000000 | 58.151 | 83.994 | 1.444 |
| 1700000000000000000 | 61.560 | 88.904 | 1.444 |
| 18000000000000000000 | 64.965 | 93.808 | 1.444 |
| 190000000000000000000 | 68.365 | 98.706 | 1.444 |
| 2000000000000000000000 | 71.760 | 103.597 | 1.444 |

# Para finalizar

- Árvores balanceadas são muito utilizadas em problemas reais:
  - JAVA: TreeMap, TreeSet.
  - C++: Map, Set do STL.
- Custo de busca, inserção, remoção da árvore AVL:  O(log n)

# Sobre os slides

Slides baseados em:

- Szwarcfiter, J.L. & Markezon, L. **Estruturas de Dados e seus Algoritmos**, 3a edição, LTC, 2010.
- Horowitz, E. & Sahni, S.; **Fundamentos de Estruturas de Dados**, Editora Campus, 1984.
- Wirth, N.; **Algoritmos e Estruturas de Dados**, Prentice/Hall do Brasil, 1989.
- Material de aula do **Prof. José Augusto Baranauskas** (USP/Riberão Preto)