SBVESDD: Estruturas de Dados

Aula 08: Estruturas de Dados Não-Lineares - Árvores AVL

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

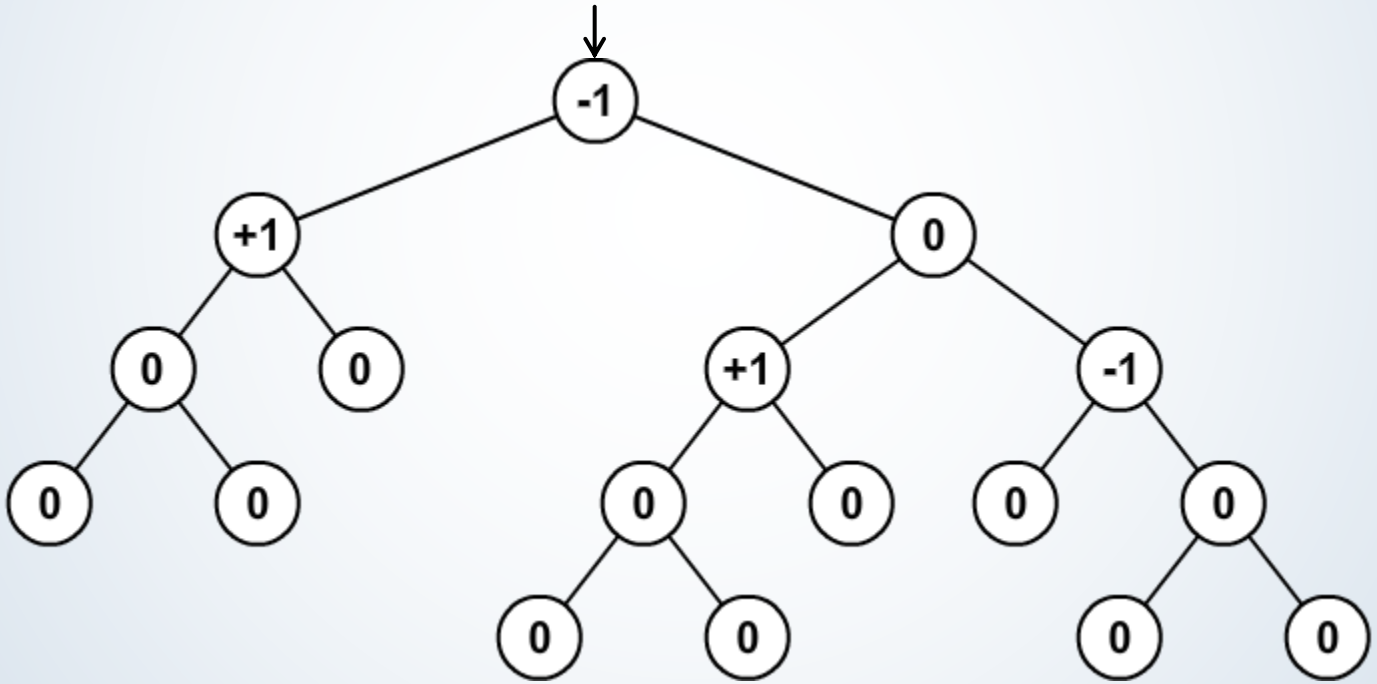
Árvores AVL

- São árvores binárias de busca balanceadas propostas em 1962 pelos matemáticos soviéticos Georgy Maximovich **A**delson-**V**elsky e Evgenii Mikhailovich **L**andis;
- A altura balanceada implica em operações com ordem de crescimento proporcionais a $O(\lg n)$.

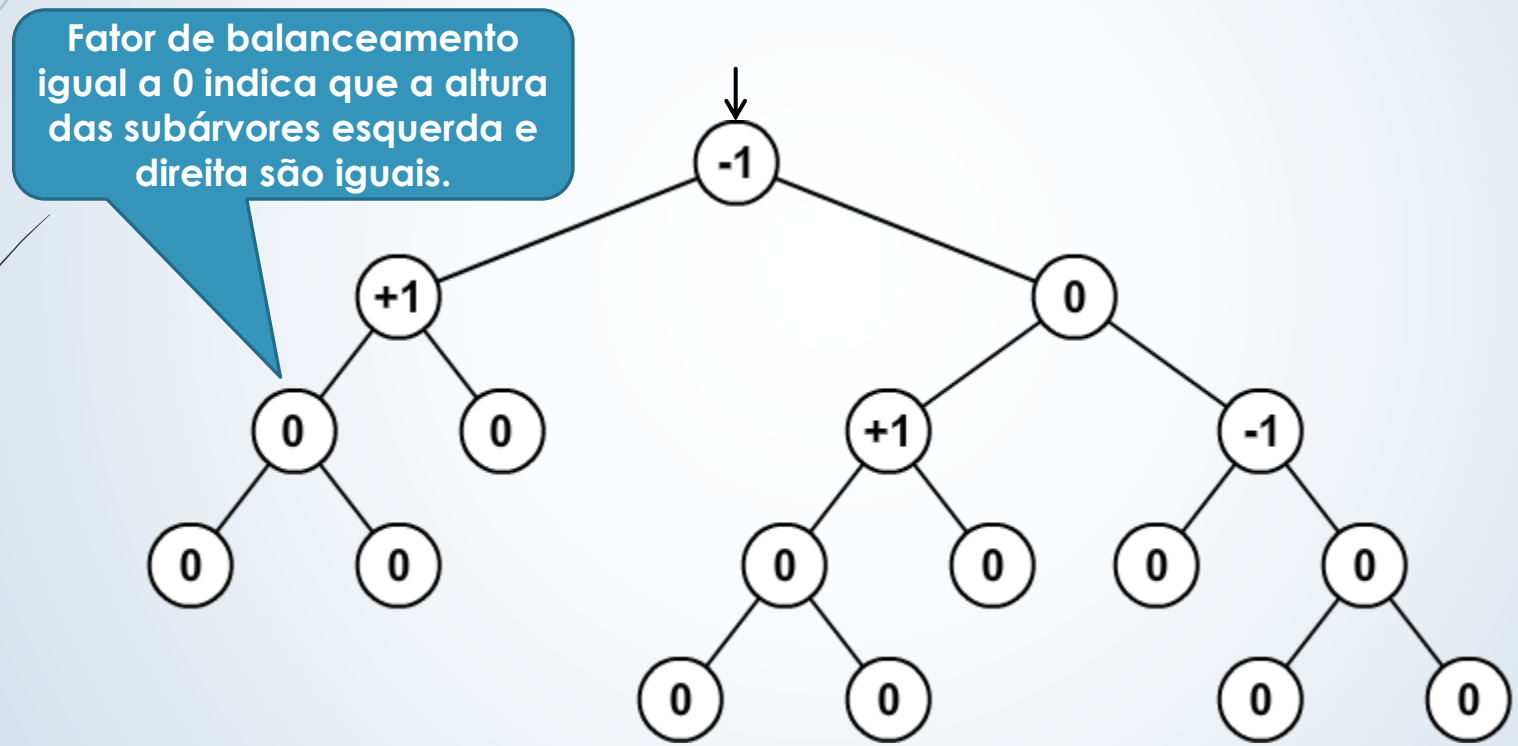
Árvores AVL
Fator de Balanceamento

- Dado um nó **T**, seu fator de balanceamento é dado por $h_e - h_d$;
- Os fatores de balanceamento aceitáveis para que **T** seja um nó de uma árvore AVL são -1, 0 ou +1;
- Se **T** for um nó folha, seu fator de balanceamento é 0.

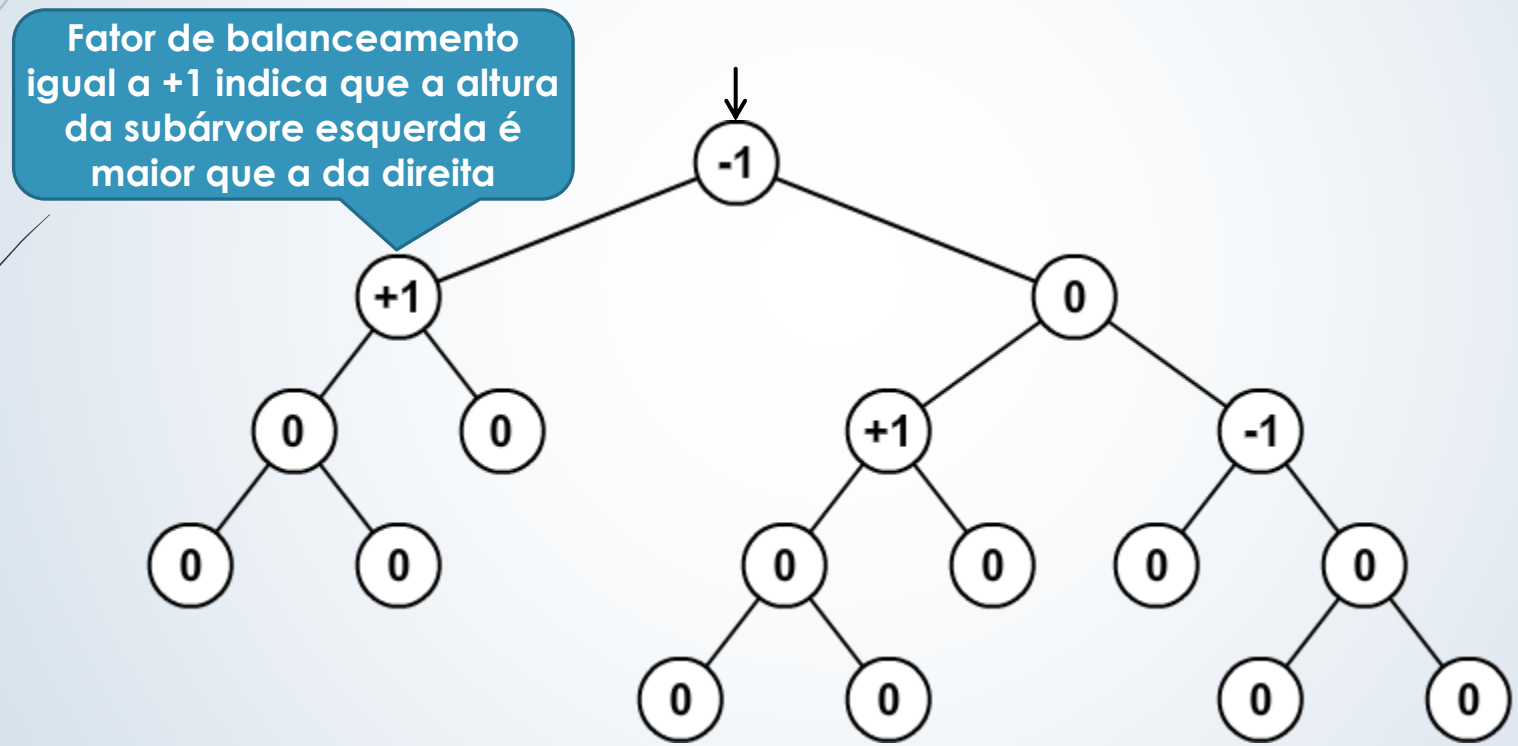
Árvores AVL
Fator de Balanceamento



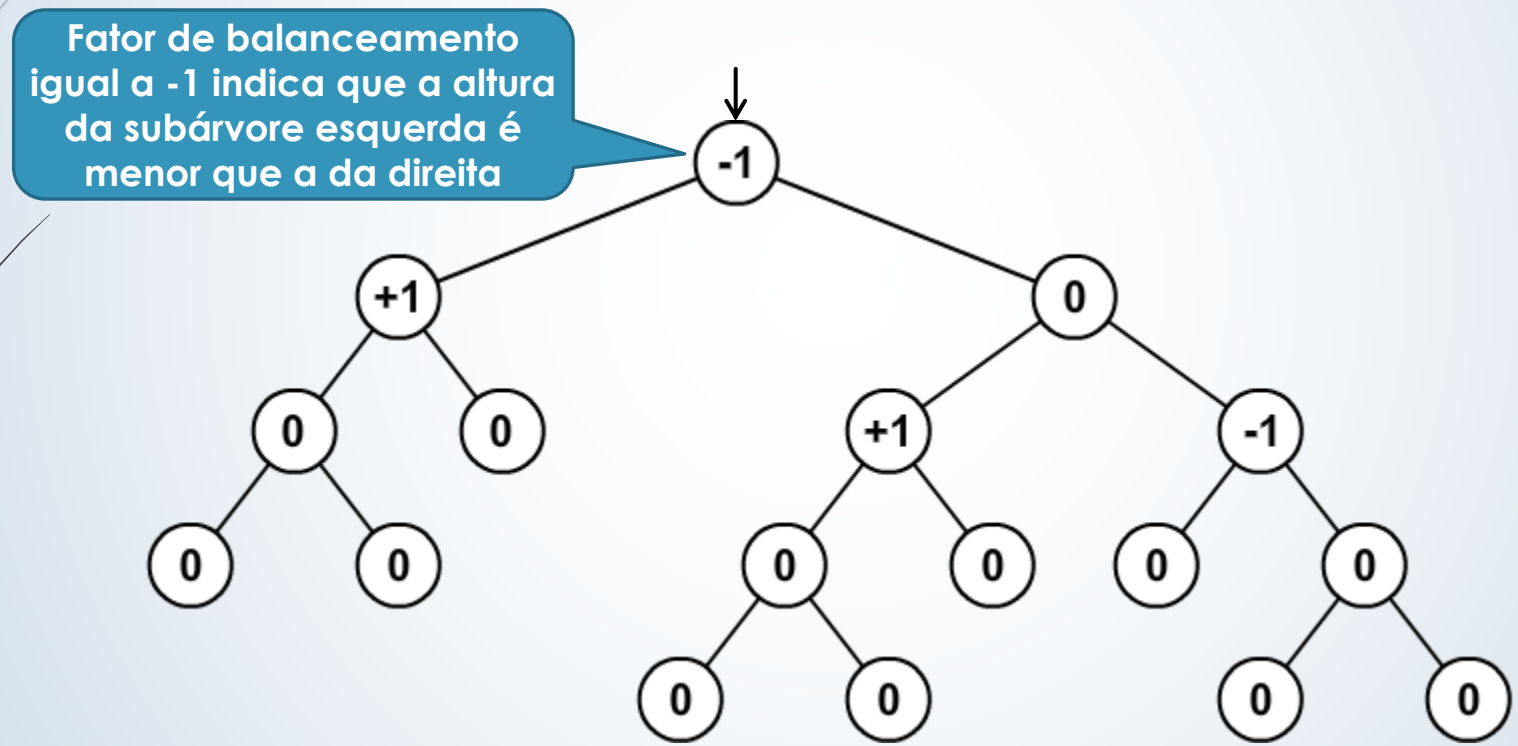
Árvores AVL
Fator de Balanceamento



Árvores AVL
Fator de Balanceamento



Árvores AVL
Fator de Balanceamento



Árvores AVL
Rotações

- O rebalanceamento utiliza quatro tipos de rotações: EE, DD, ED e DE
  - EE e DD são simétricas entre si;
  - ED e DE são simétricas entre si;
- As rotações tomam como base o fator de balanceamento do nó **A**, o ancestral mais próximo de **N**, o novo nó inserido. Para haver uma rotação o fator de balanceamento de **A** deve ser +2 ou -2.

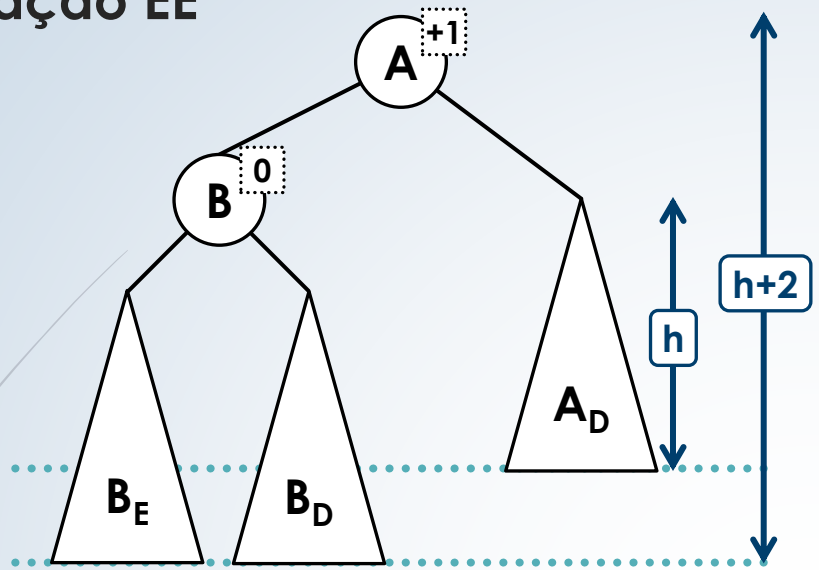
Árvores AVL
Rotações

- **EE: N** foi inserido na subárvore **esquerda** da subárvore **esquerda** de **A**;
- **ED: N** foi inserido na subárvore **direita** da subárvore **esquerda** de **A**;
- **DD: N** foi inserido na subárvore **direita** da subárvore **direita** de **A**;
- **DE: N** foi inserido na subárvore **esquerda** da subárvore **direita** de **A**;

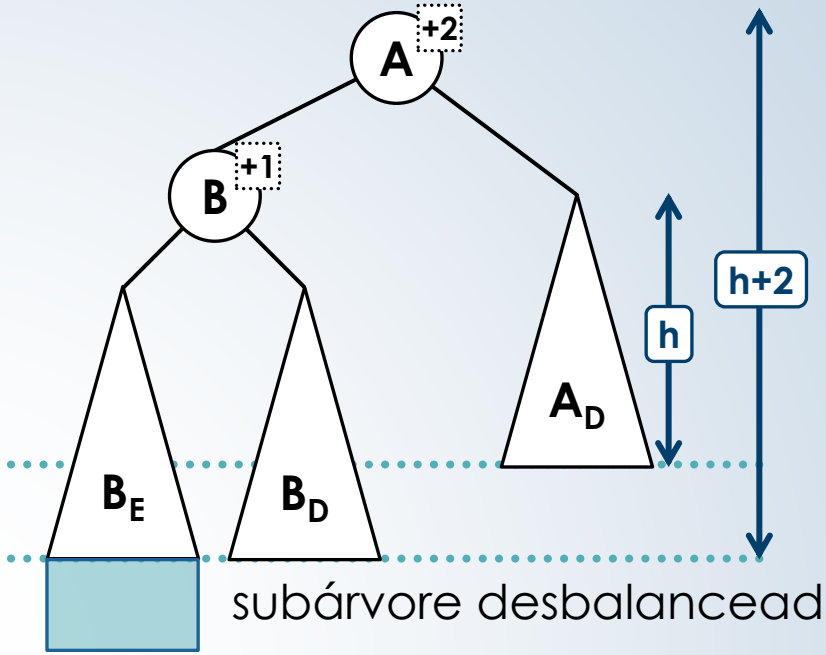
Árvores AVL
Rotações

- Se **B** é o filho de **A** aonde ocorreu a inserção de **N**:
  - **EE: A = +2; B = +1;**
  - **ED: A = +2; B = -1;**
  - **DD: A = -2; B = -1;**
  - **DE: A = -2; B = +1;**
- **C** é o filho de **B** aonde ocorreu a inserção de **N**.

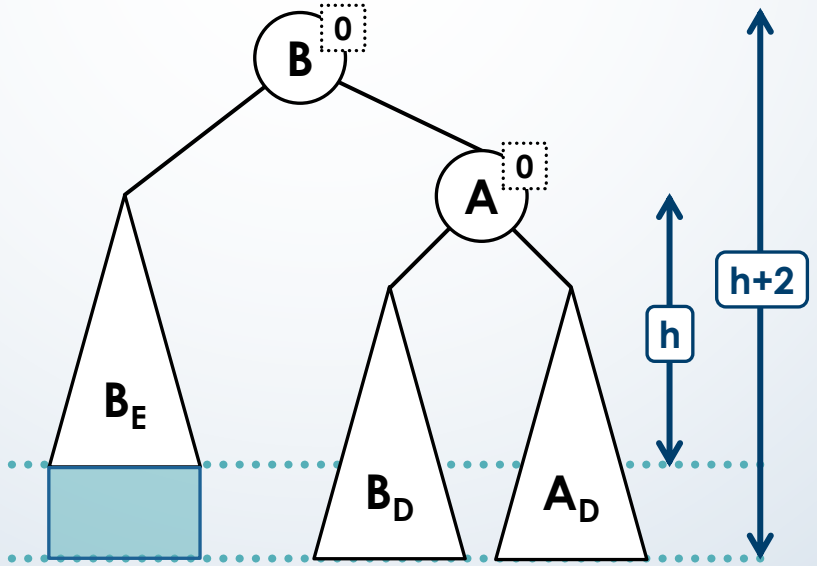
Rotação EE



subárvore balanceada

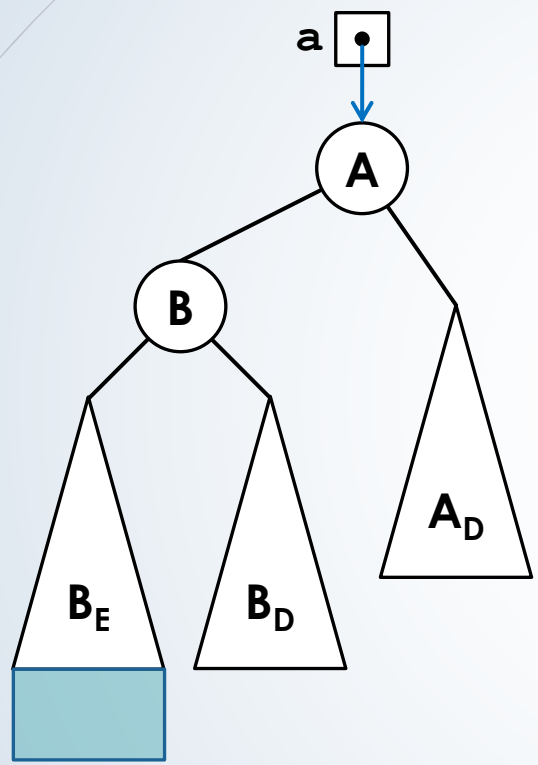


subárvore desbalanceada



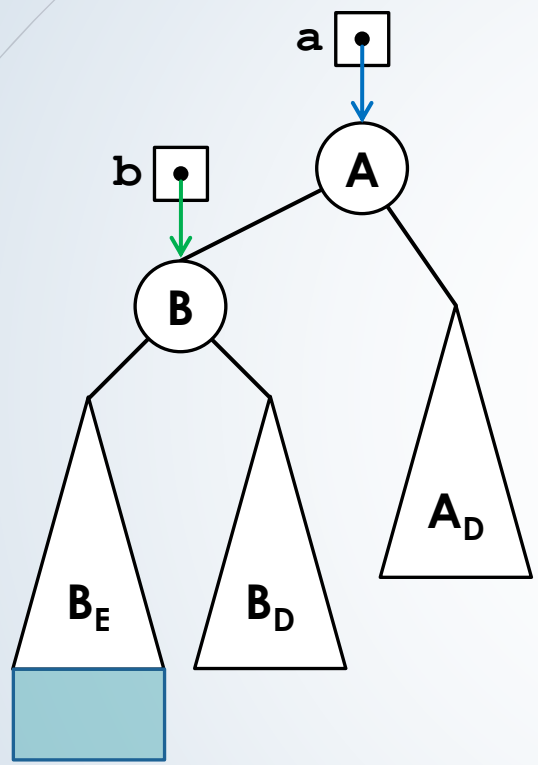
subárvore rebalanceada

Rotação EE



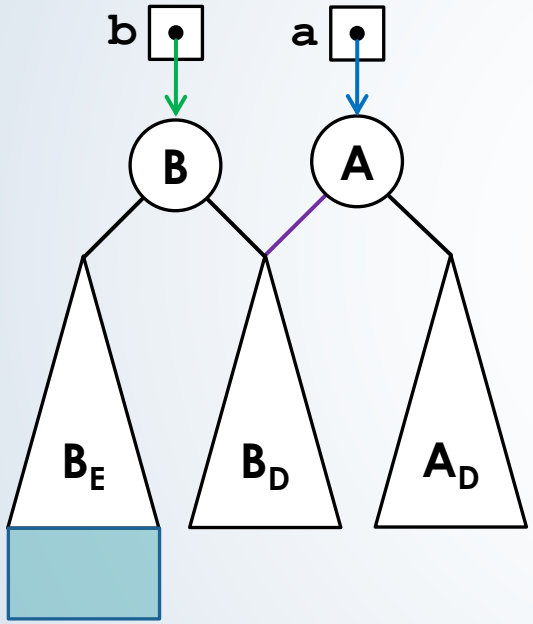

```
private No<Tipo> ee( No<Tipo> a ) {
    No<Tipo> b = a.esquerda;
    a.esquerda = b.direita;
    b.direita = a;
    return b;
}
```

Rotação EE



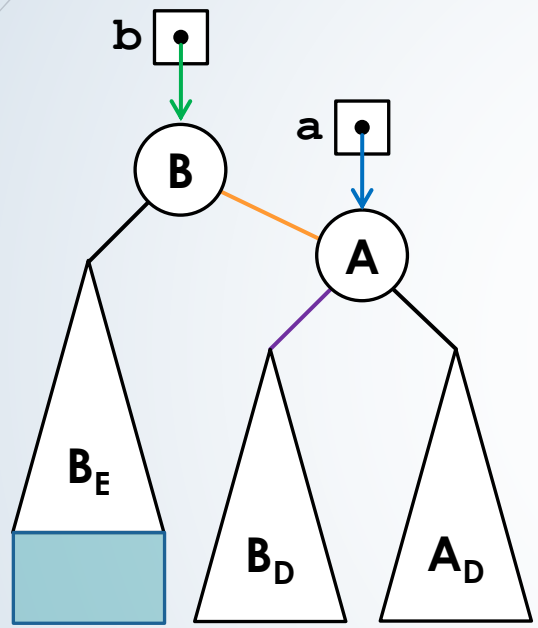

```
private No<Tipo> ee( No<Tipo> a ) {
    No<Tipo> b = a.esquerda;
    a.esquerda = b.direita;
    b.direita = a;
    return b;
}
```

Rotação EE



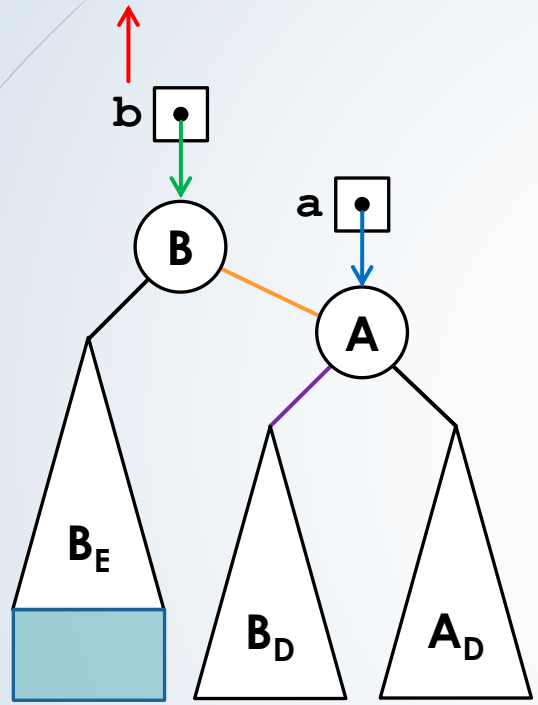

```
private No<Tipo> ee( No<Tipo> a ) {
    No<Tipo> b = a.esquerda;
    a.esquerda = b.direita;
    b.direita = a;
    return b;
}
```

Rotação EE



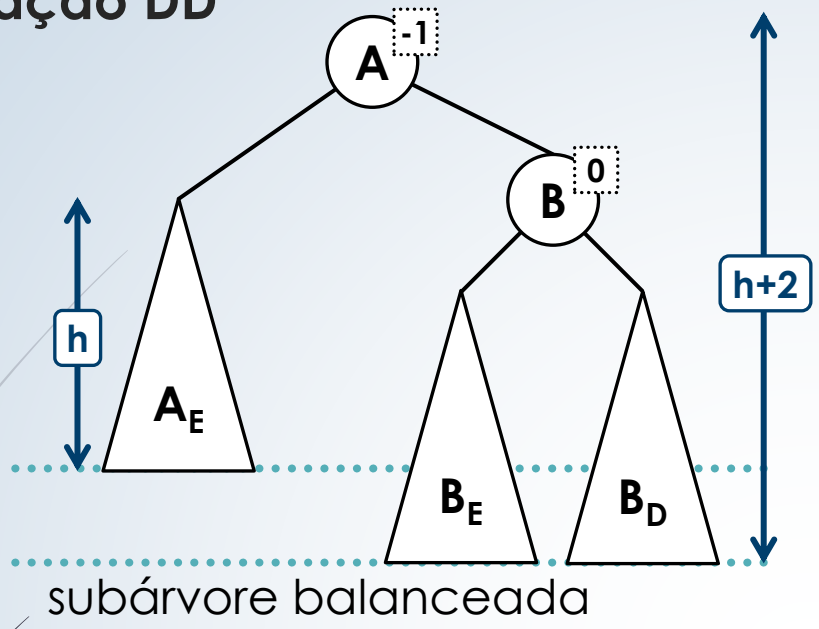

```
private No<Tipo> ee( No<Tipo> a ) {
    No<Tipo> b = a.esquerda;
    a.esquerda = b.direita;
    b.direita = a;
    return b;
}
```

Rotação EE

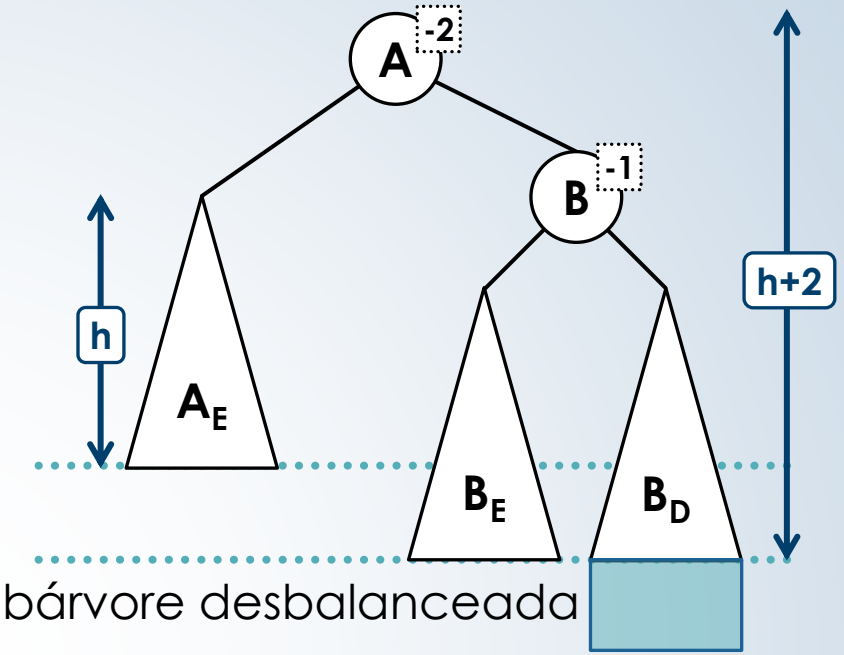



```
private No<Tipo> ee( No<Tipo> a ) {
    No<Tipo> b = a.esquerda;
    a.esquerda = b.direita;
    b.direita = a;
    return b;
}
```

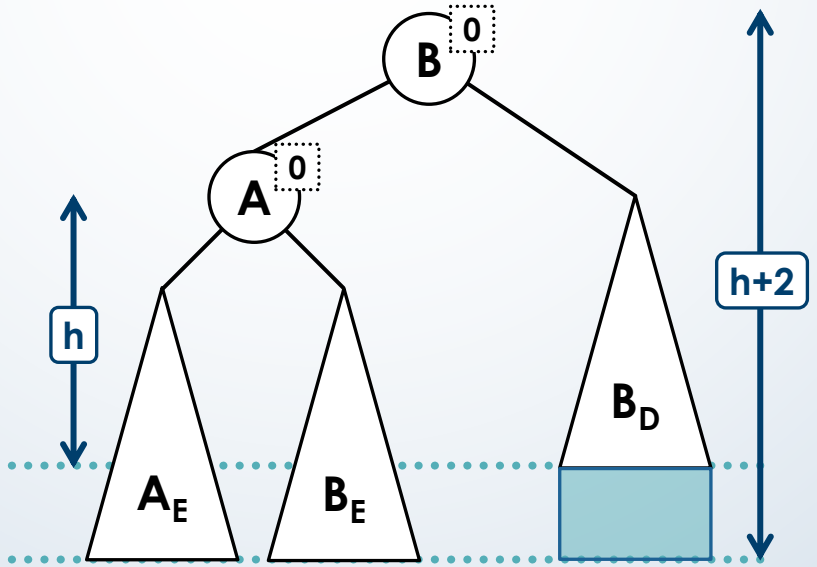
Rotação DD



subárvore balanceada

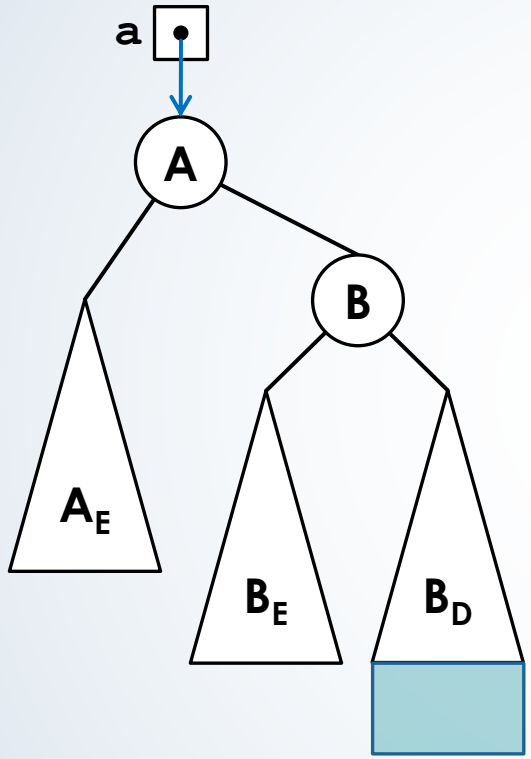


subárvore desbalanceada



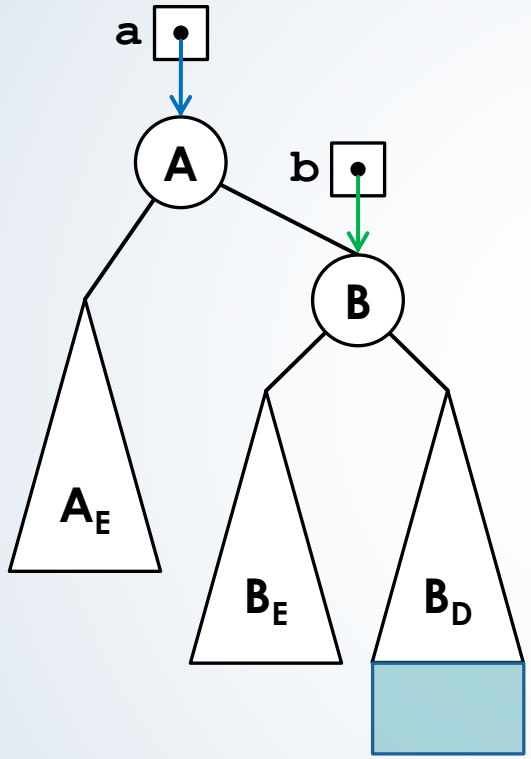
subárvore rebalanceada

Rotação DD



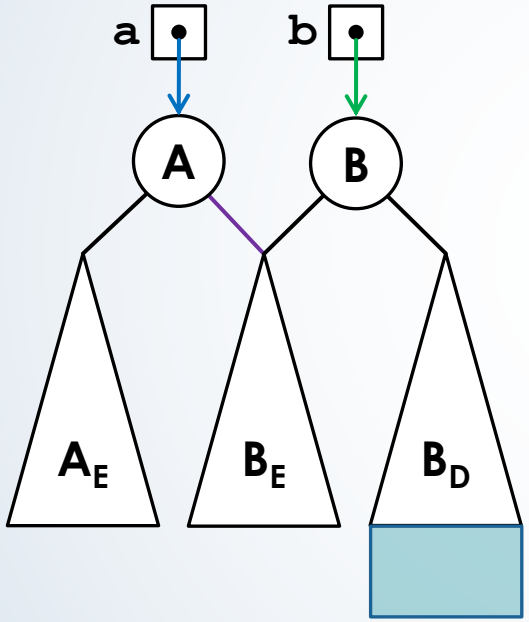

```
private No<Tipo> dd( No<Tipo> a ) {
    No<Tipo> b = a.direita;
    a.direita = b.esquerda;
    b.esquerda = a;
    return b;
}
```

Rotação DD



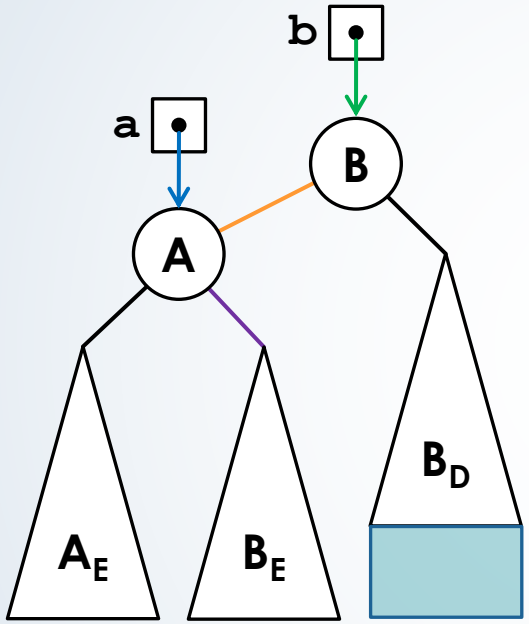

```
private No<Tipo> dd( No<Tipo> a ) {
    No<Tipo> b = a.direita;
    a.direita = b.esquerda;
    b.esquerda = a;
    return b;
}
```

Rotação DD



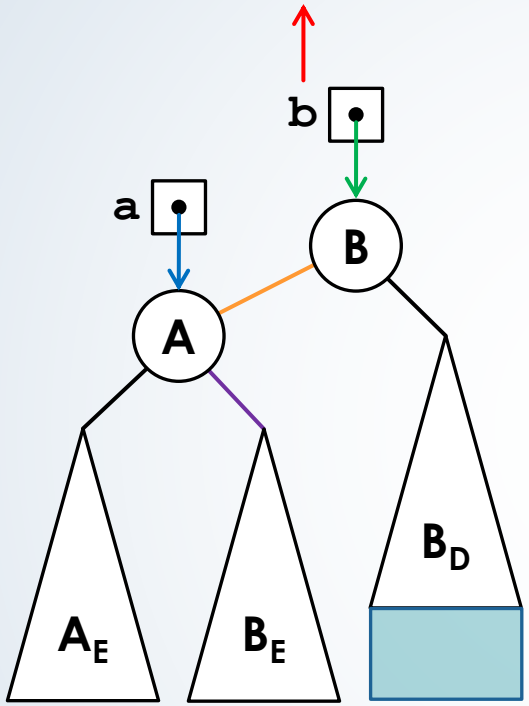

```
private No<Tipo> dd( No<Tipo> a ) {
    No<Tipo> b = a.direita;
    a.direita = b.esquerda;
    b.esquerda = a;
    return b;
}
```

Rotação DD



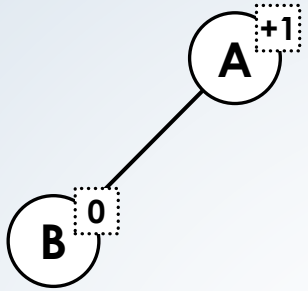

```
private No<Tipo> dd( No<Tipo> a ) {
    No<Tipo> b = a.direita;
    a.direita = b.esquerda;
    b.esquerda = a;
    return b;
}
```

Rotação DD

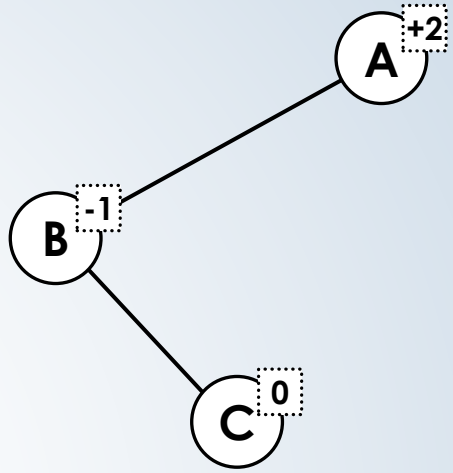



```
private No<Tipo> dd( No<Tipo> a ) {
    No<Tipo> b = a.direita;
    a.direita = b.esquerda;
    b.esquerda = a;
    return b;
}
```

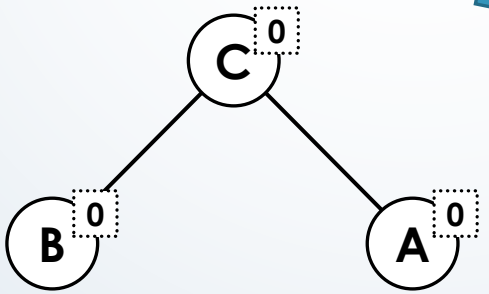
Rotação ED(a)



subárvore balanceada

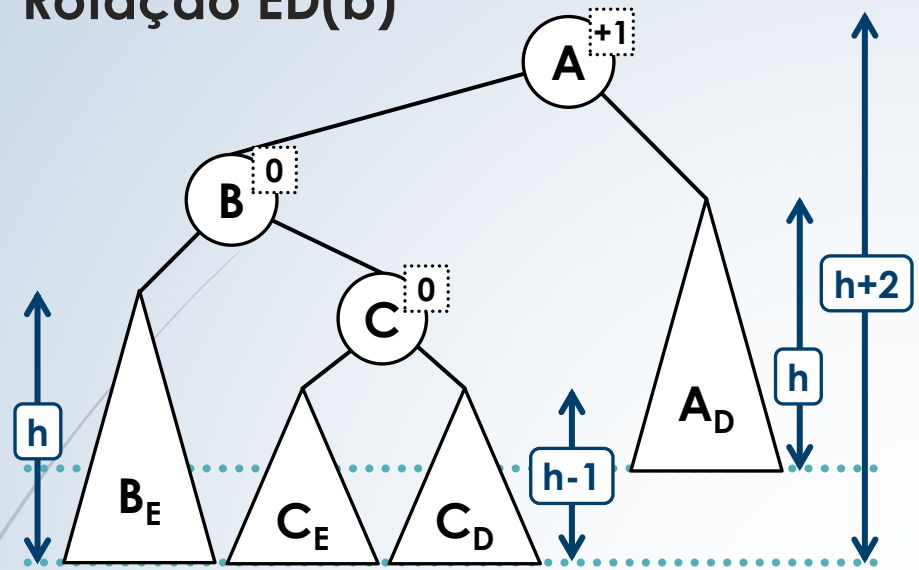


subárvore desbalanceada

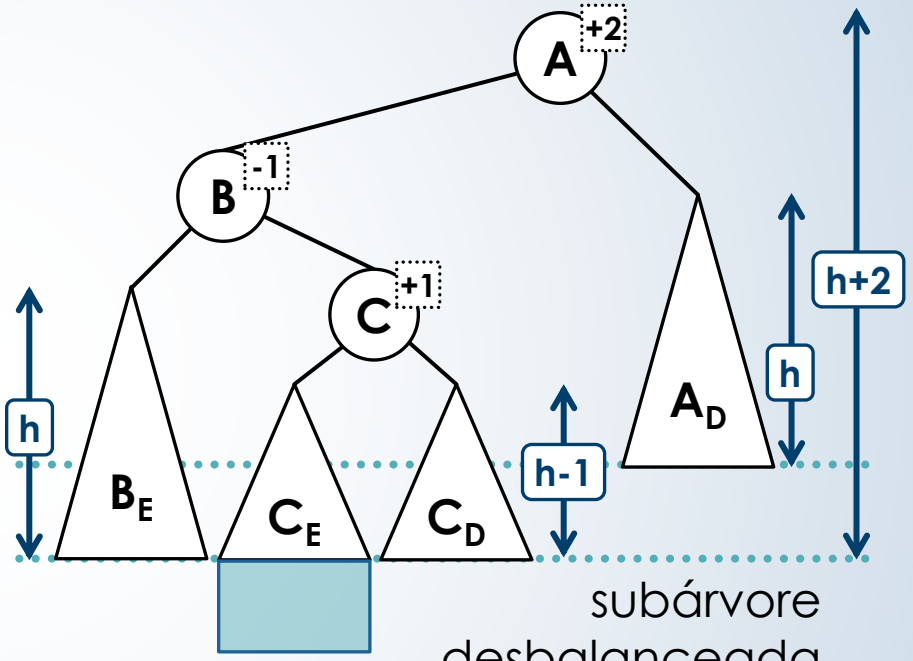


subárvore rebalanceada

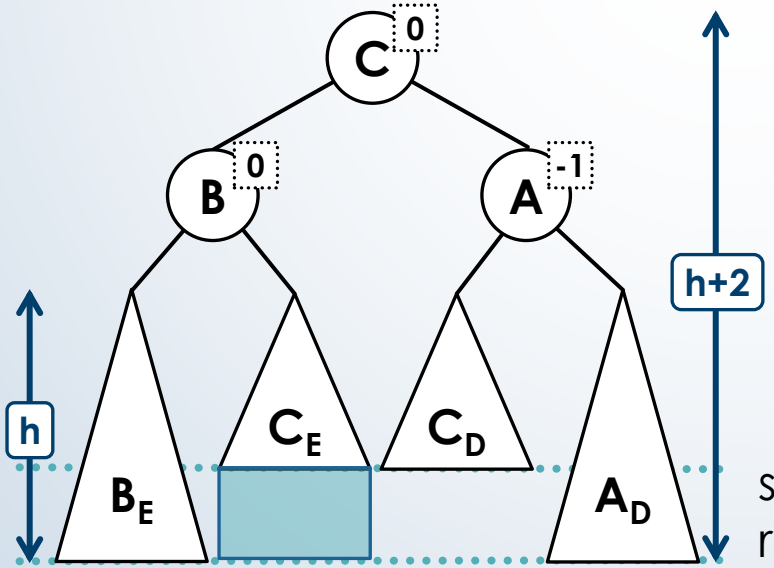
Rotação ED(b)



subárvore balanceada

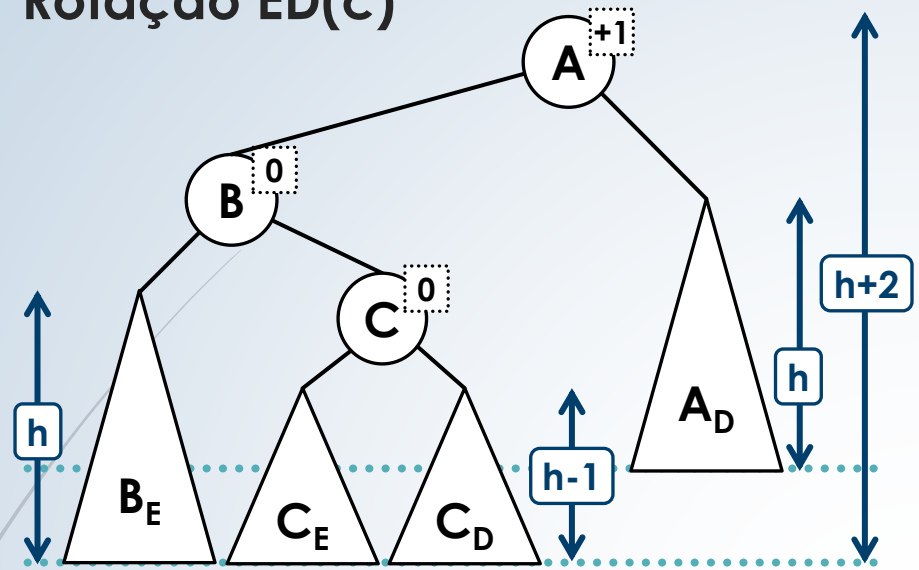


subárvore desbalanceada

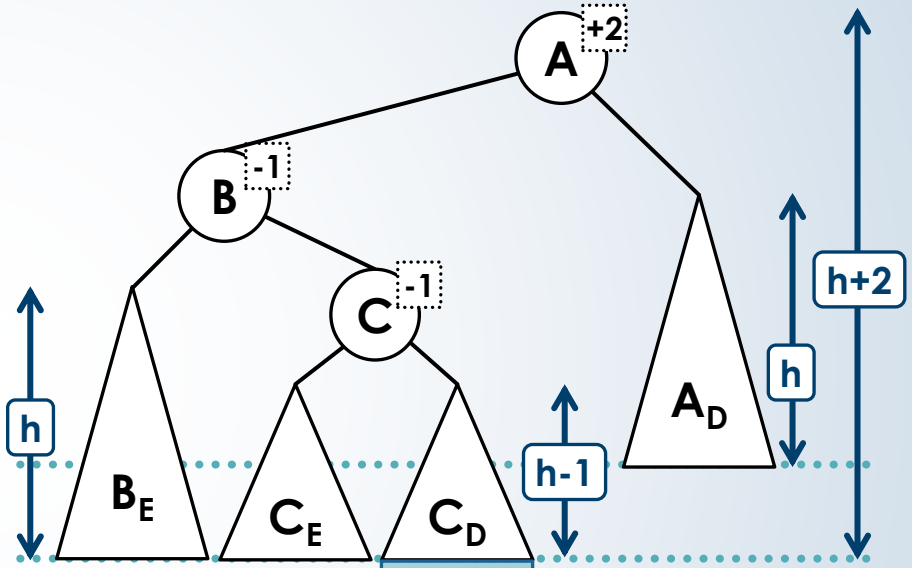


subárvore rebalanceada

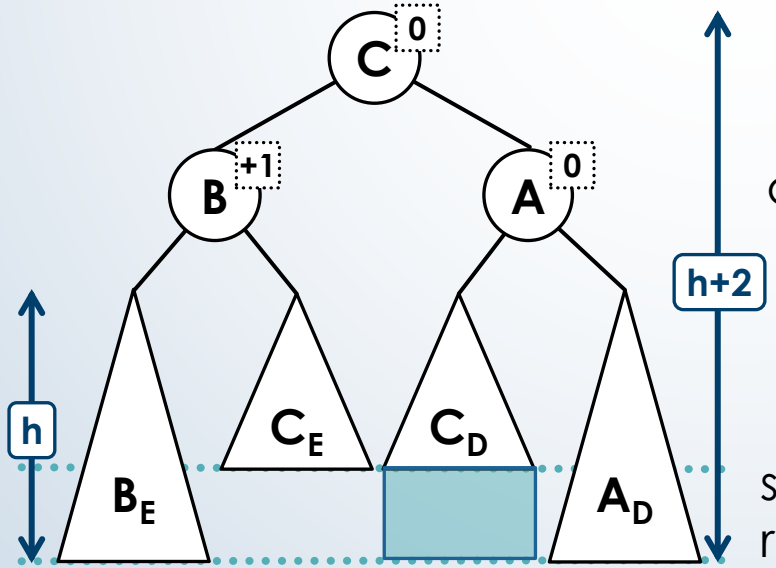
Rotação ED(c)



subárvore balanceada

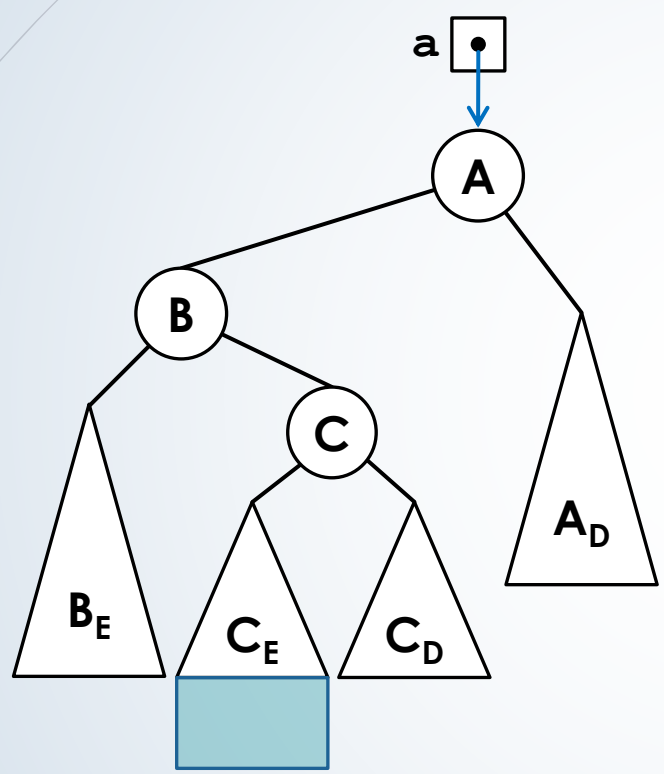


subárvore desbalanceada



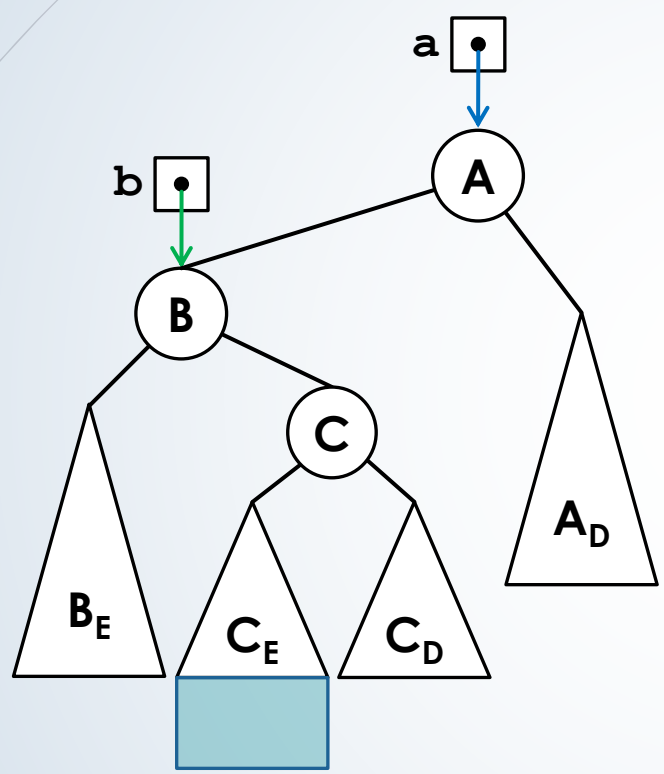
subárvore rebalanceada

Rotação ED



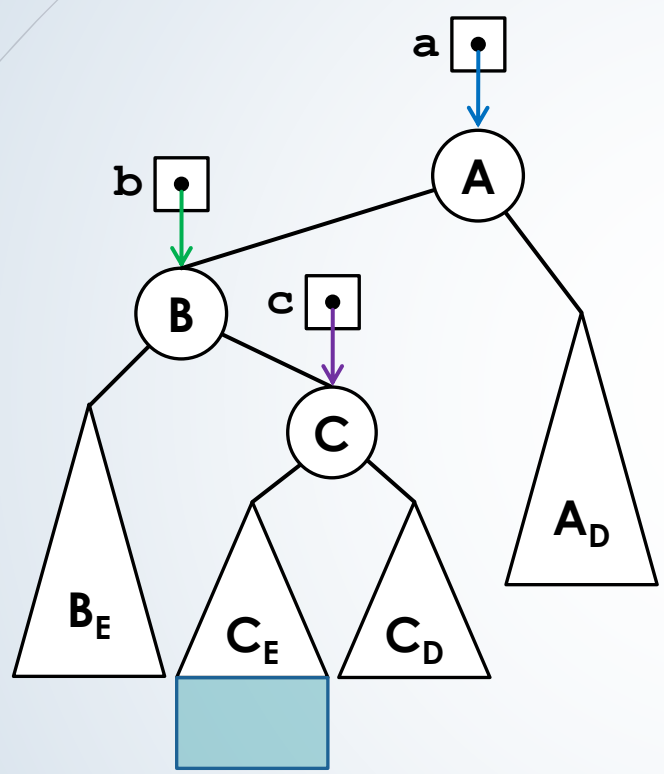

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



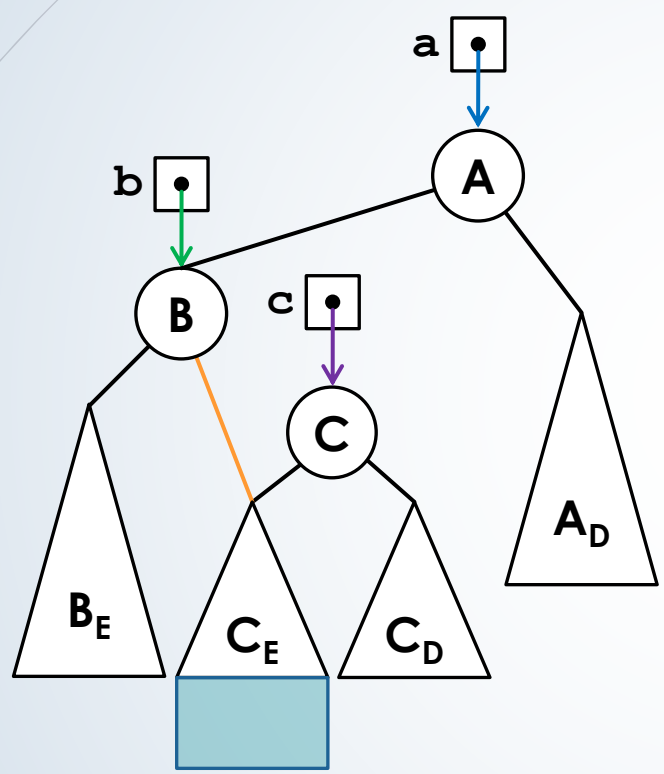

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



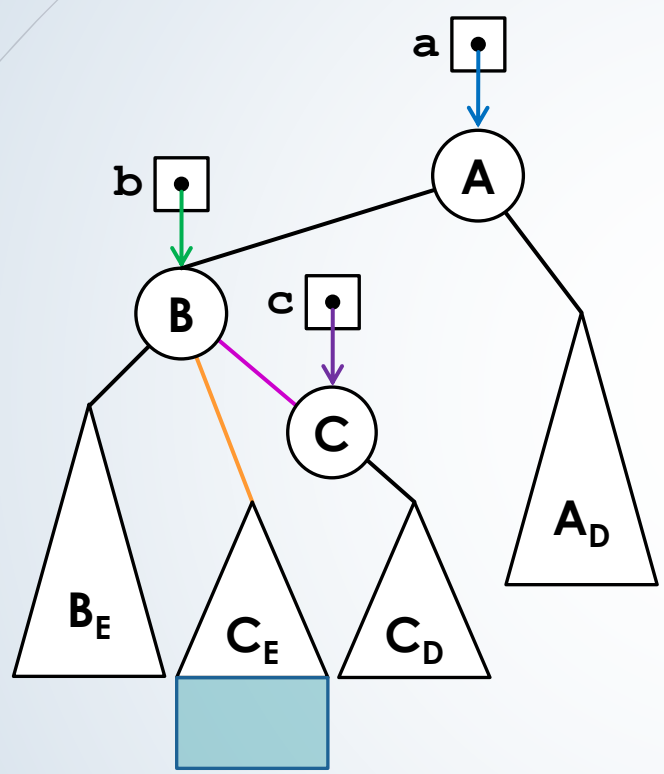

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



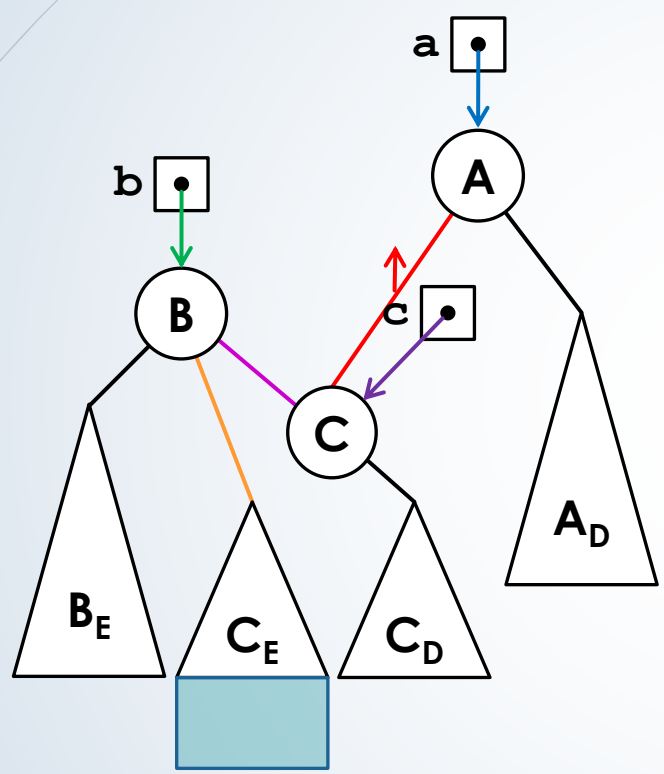

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



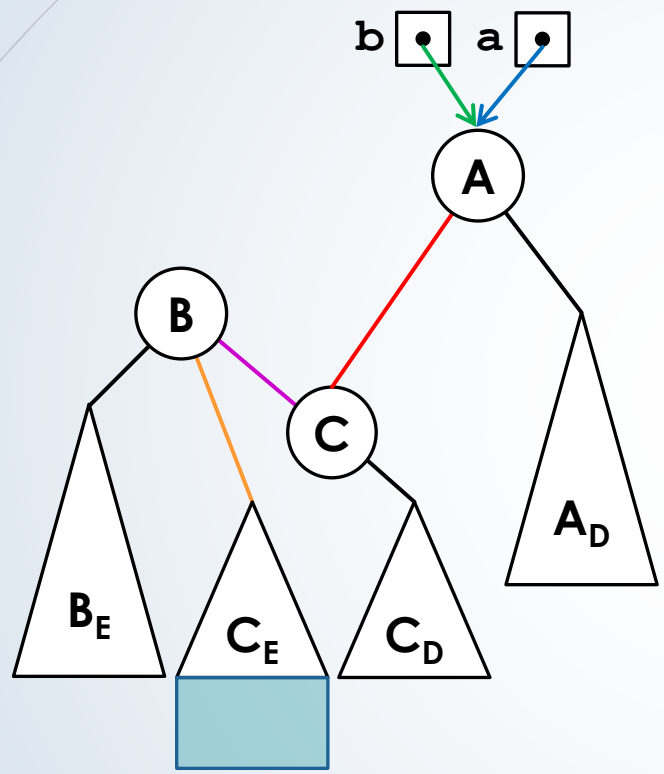

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



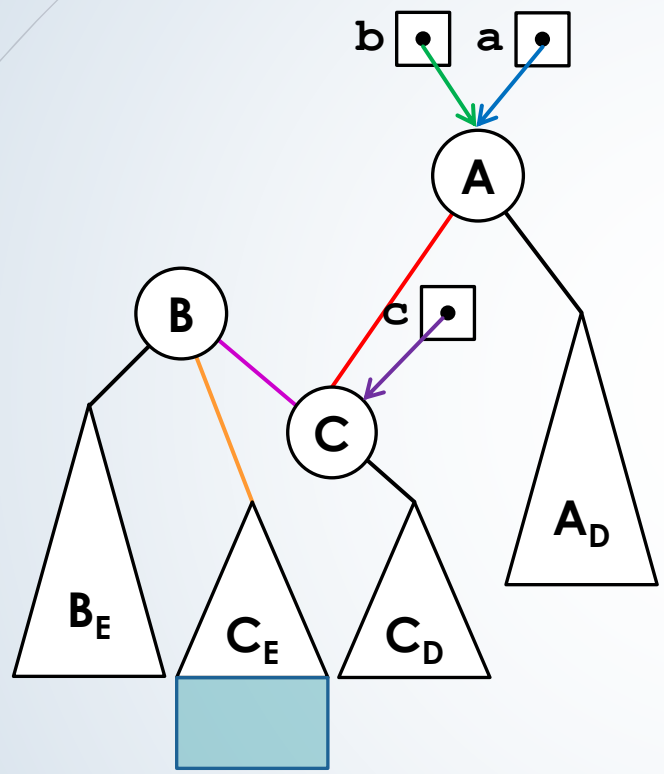

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



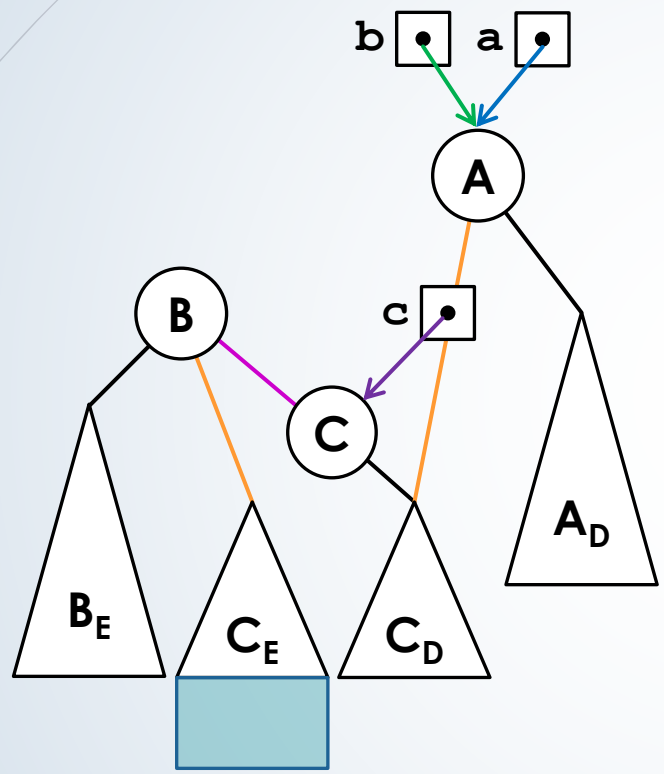

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



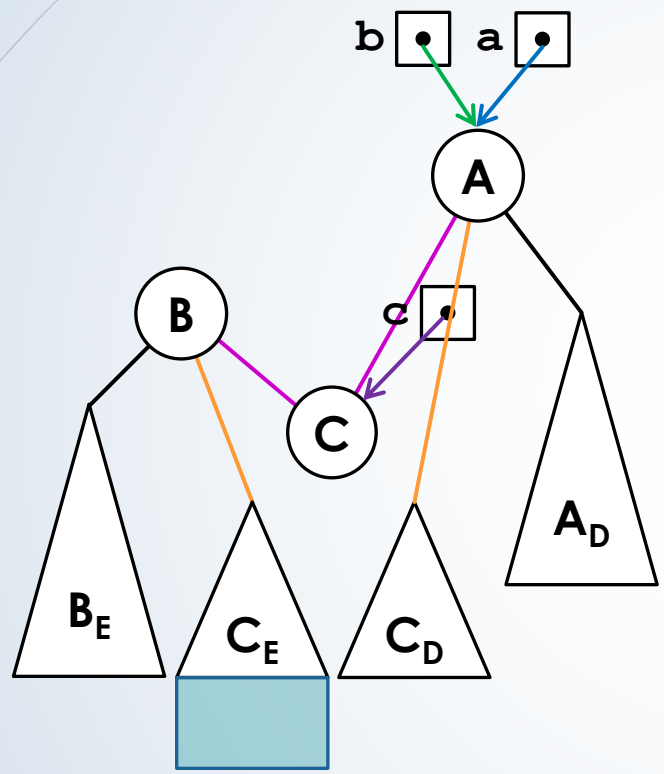

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



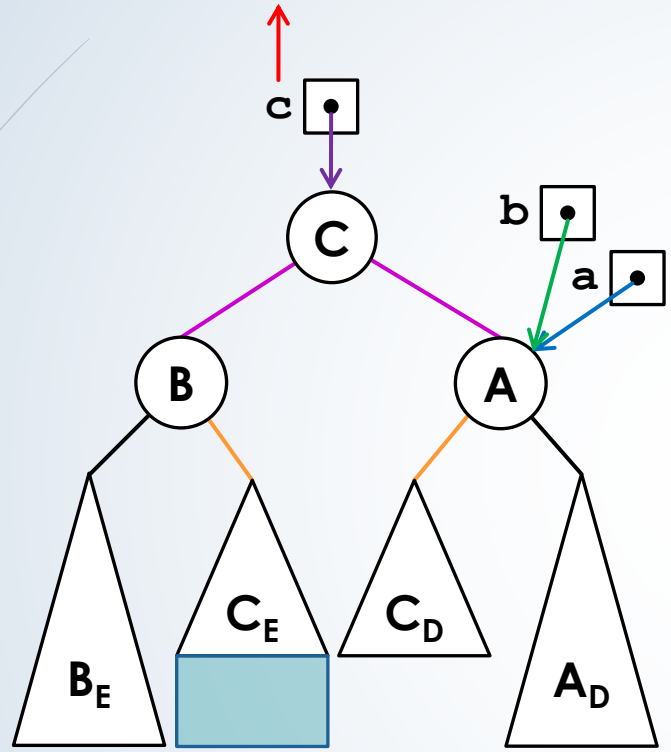

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED



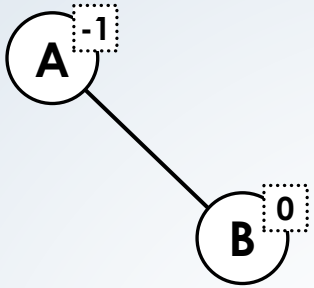

```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

Rotação ED

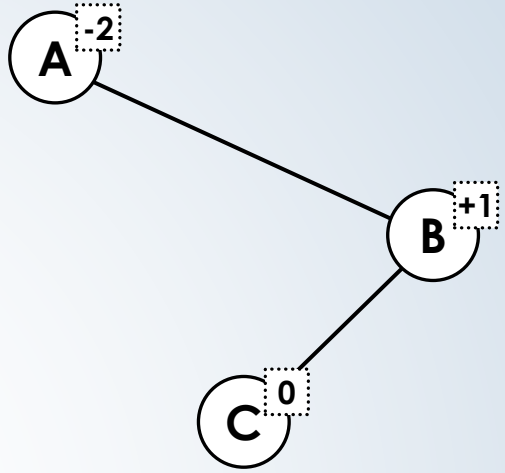



```
private No<Tipo> ed( No<Tipo> a ) {
    a.esquerda = dd( a.esquerda );
    return ee( a );
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}
```

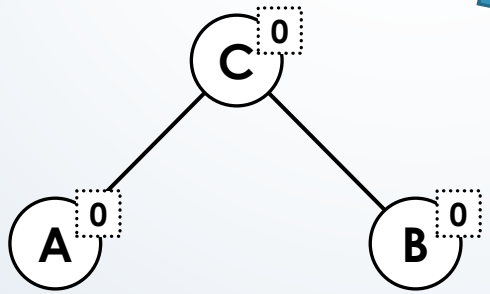
Rotação DE(a)



subárvore balanceada

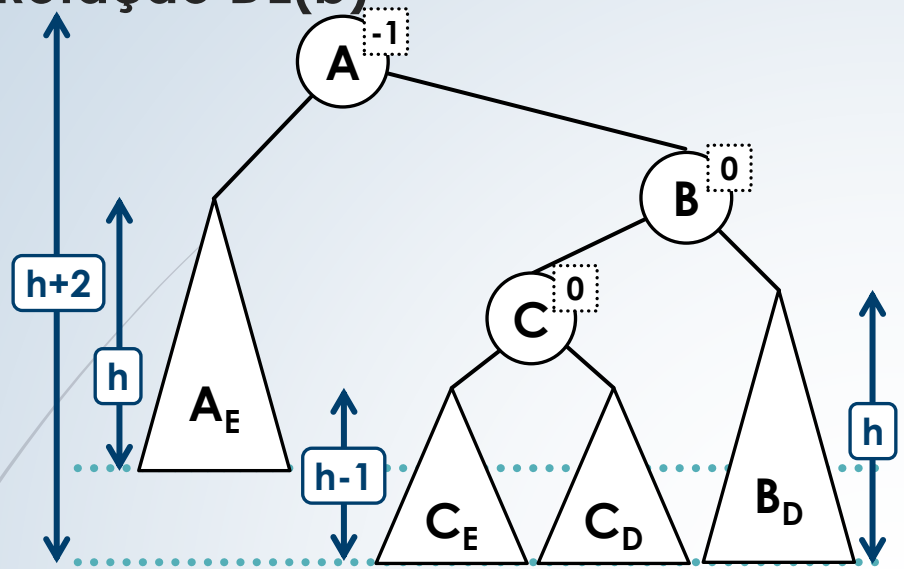


subárvore desbalanceada

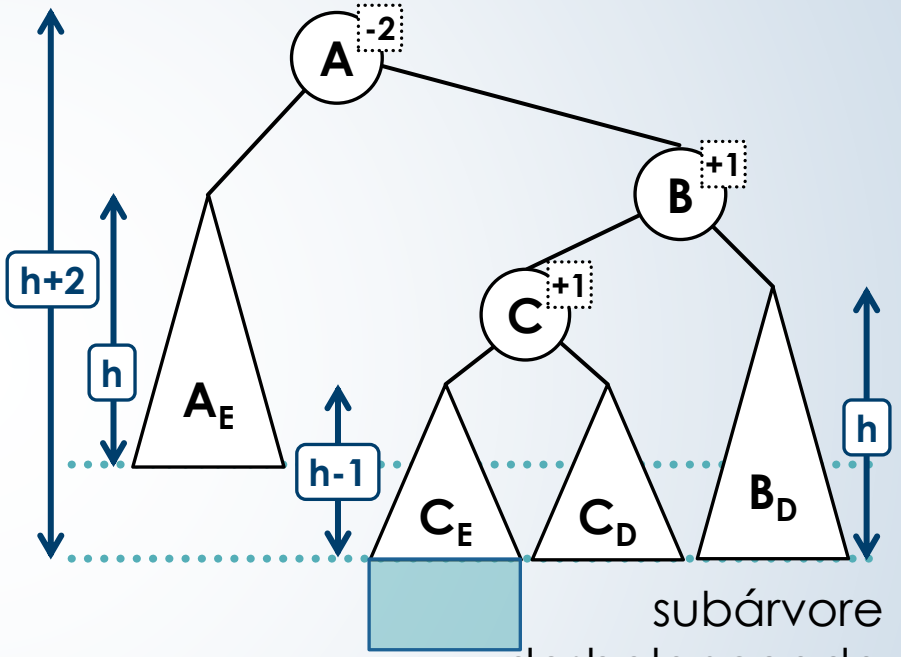


subárvore rebalanceada

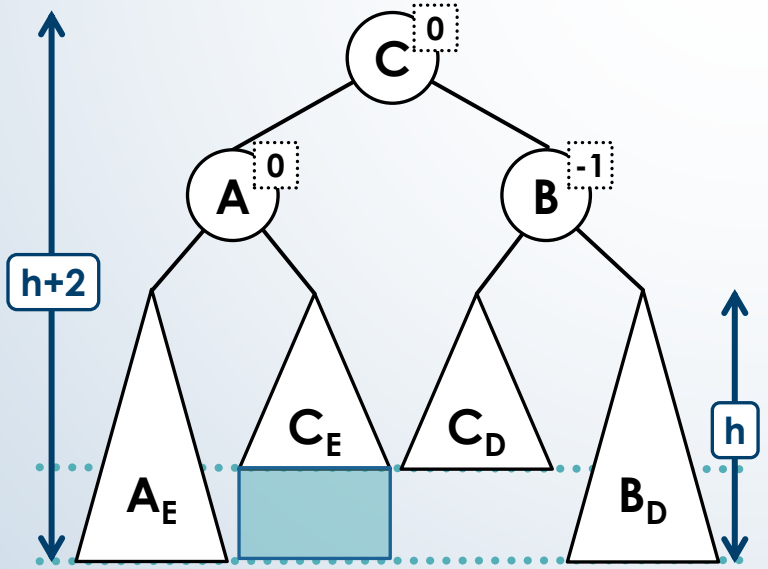
Rotação DE(b)



subárvore balanceada

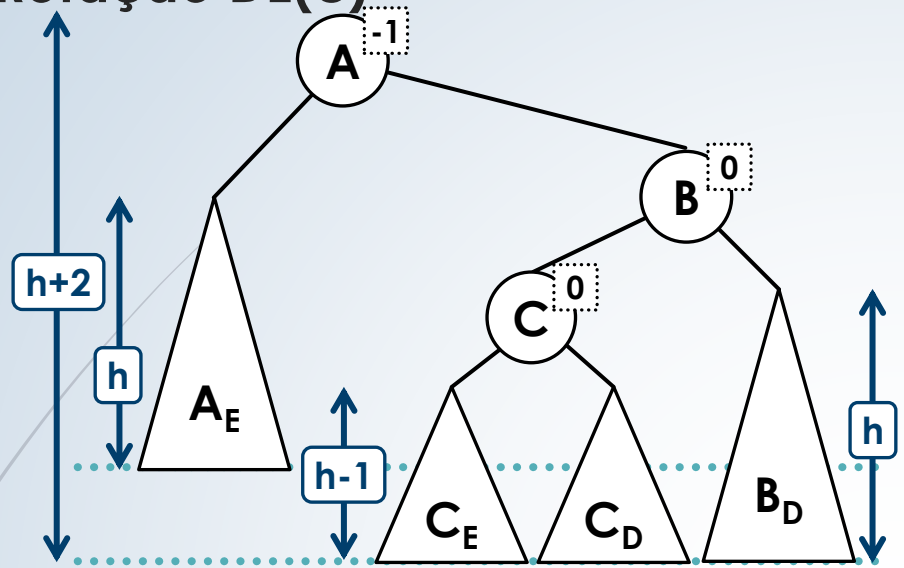


subárvore desbalanceada

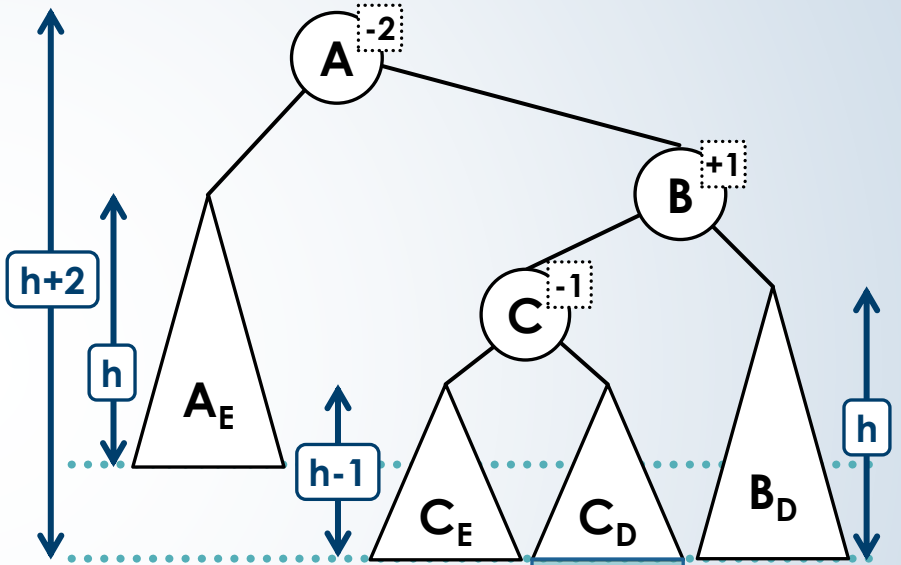


subárvore rebalanceada

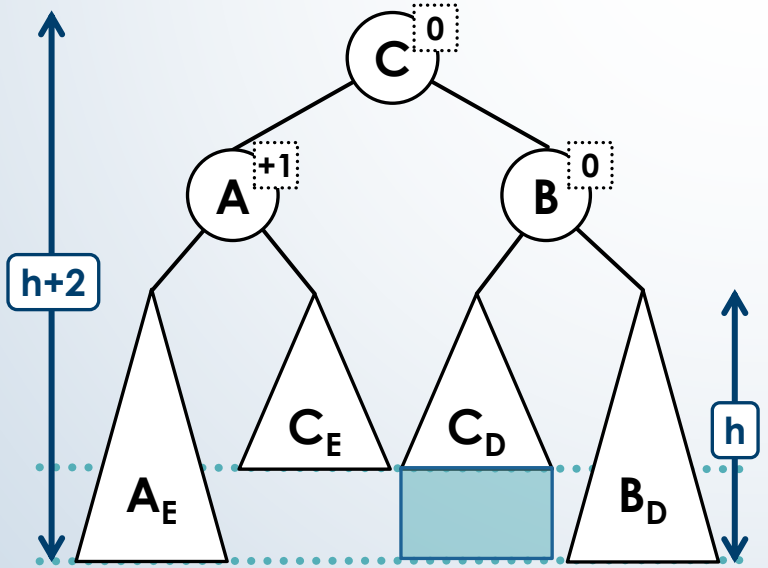
Rotação DE(c)



subárvore balanceada

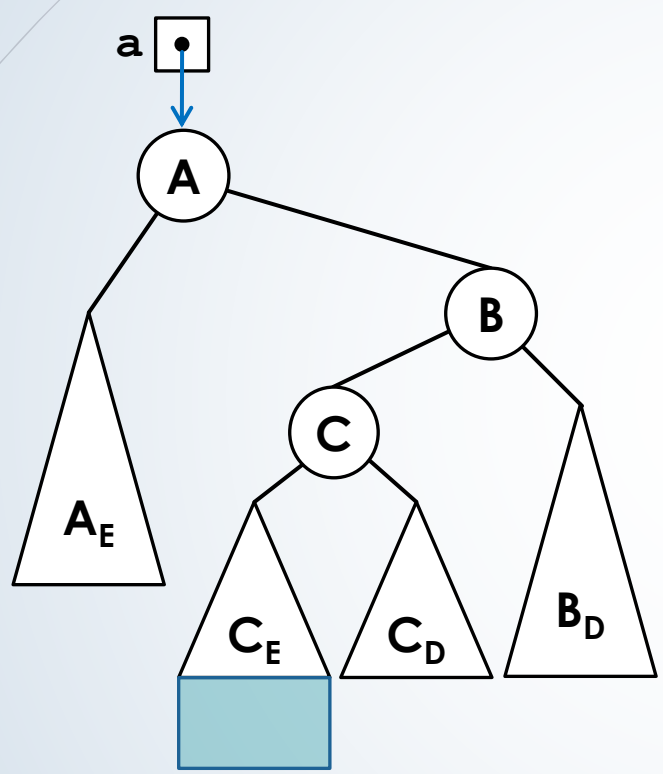


subárvore desbalanceada



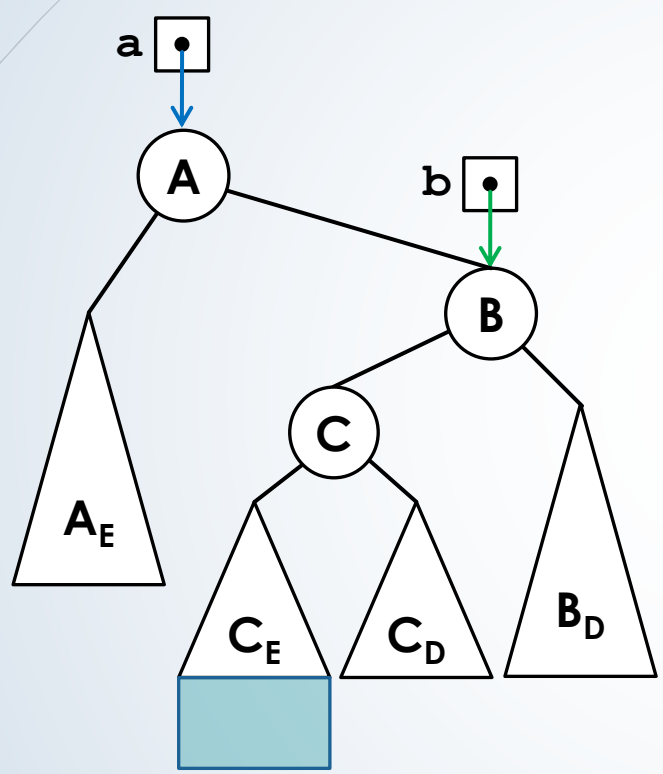
subárvore rebalanceada

Rotação DE



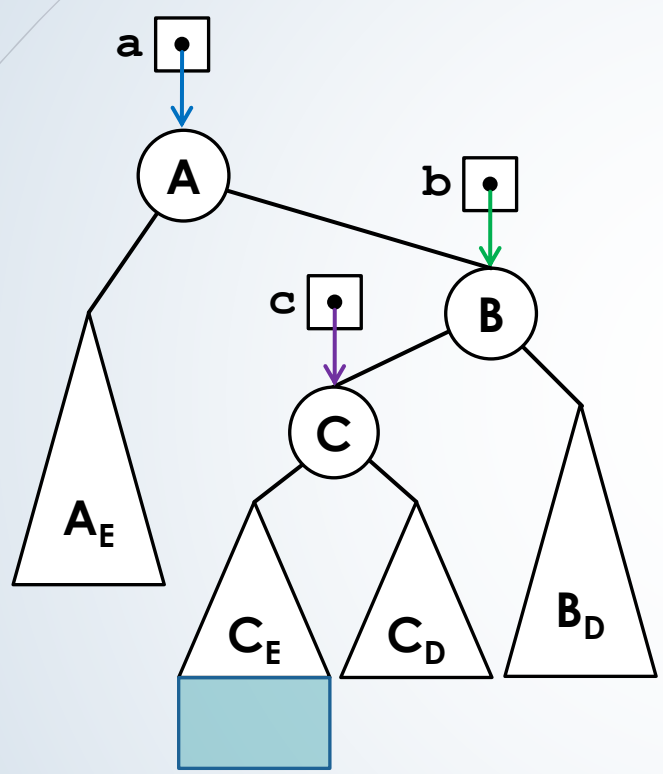

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



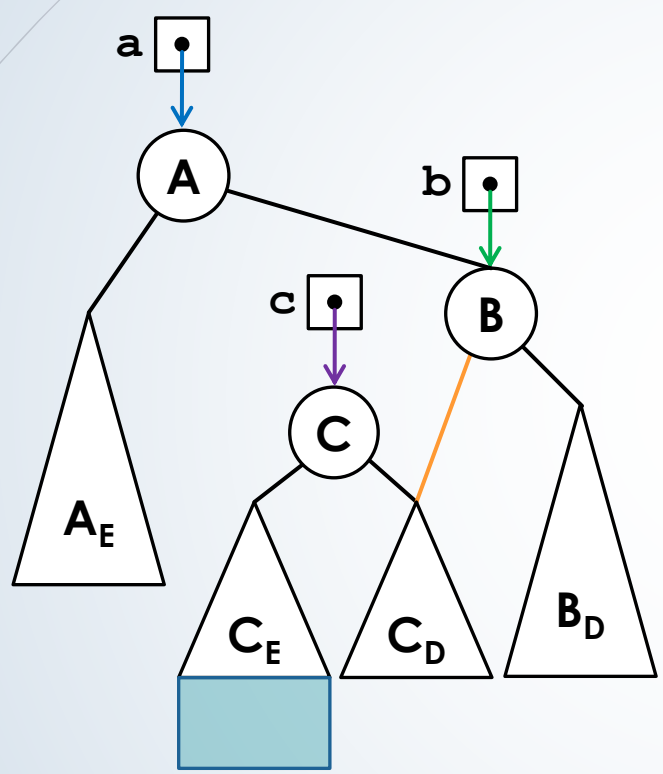

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



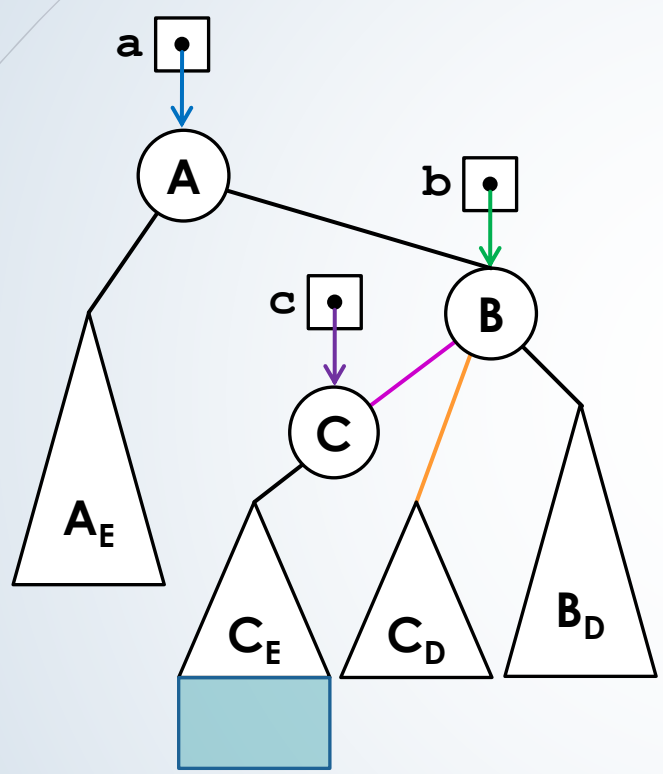

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



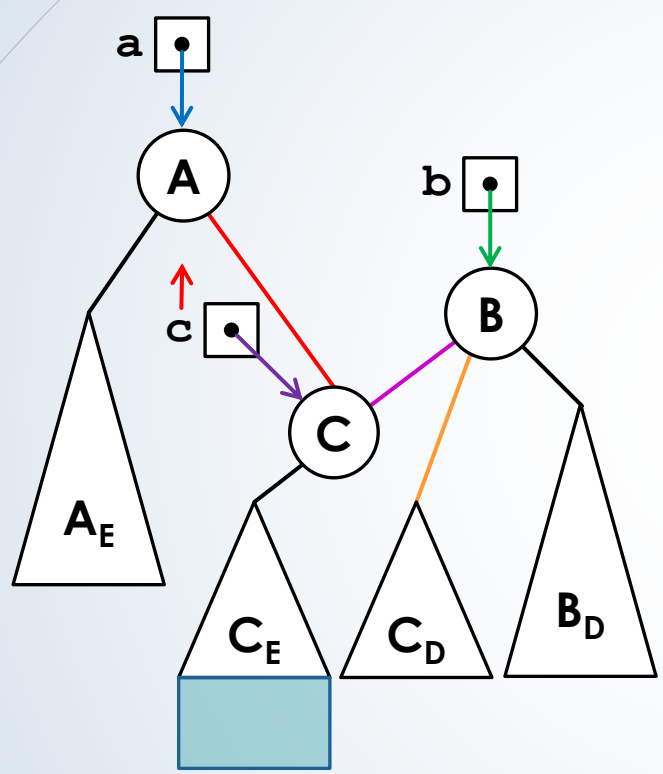

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



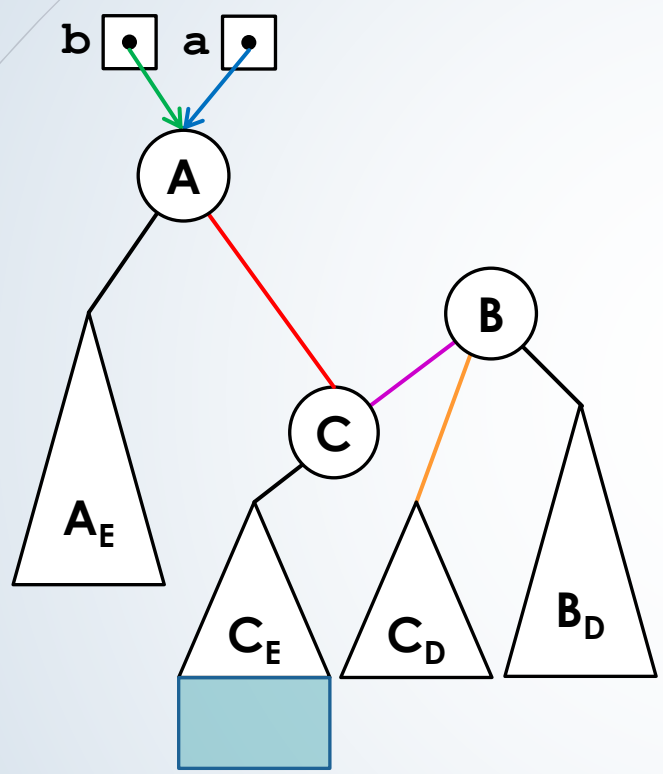

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



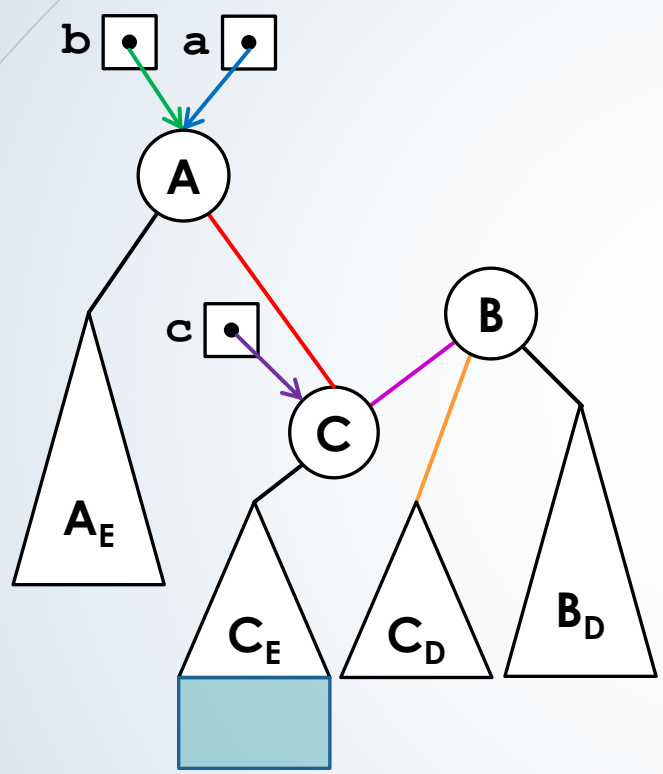

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



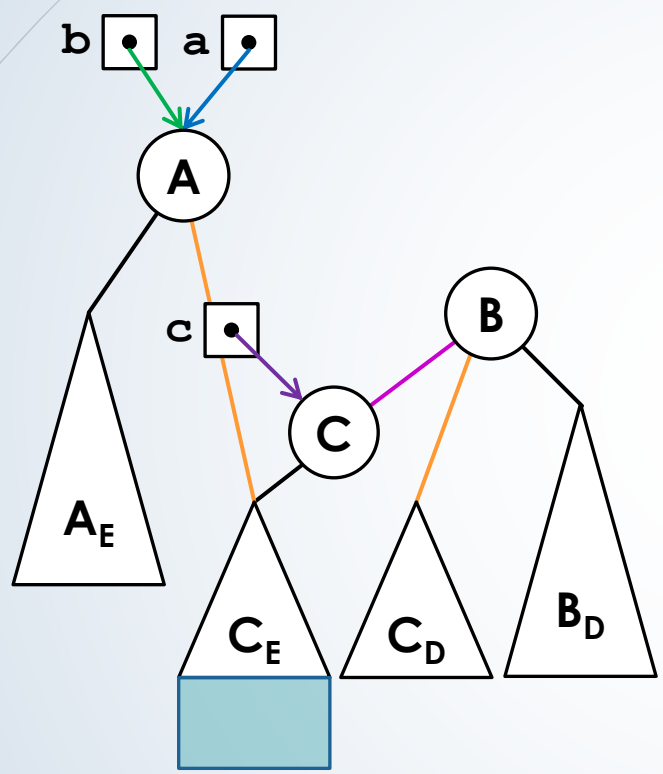

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



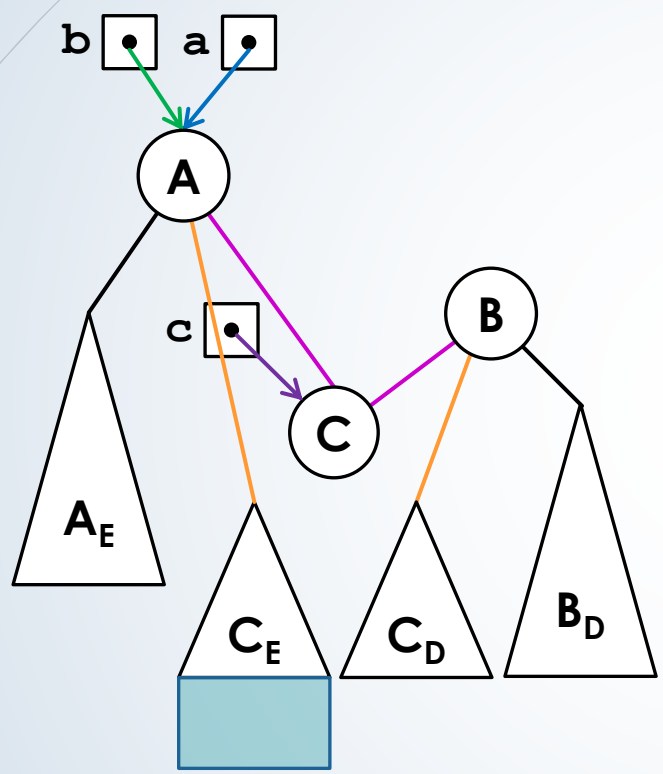

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



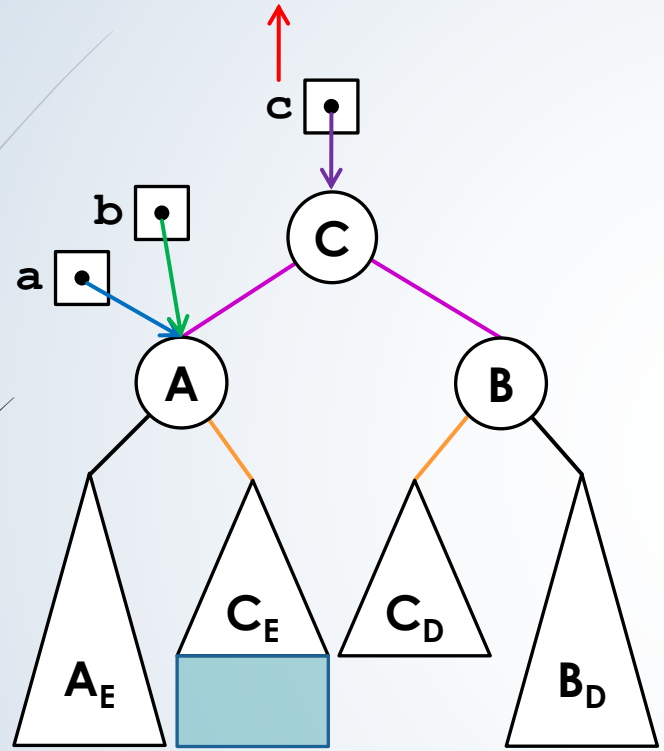

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE




```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Rotação DE



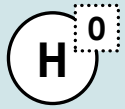

```
private No<Tipo> de( No<Tipo> a ) {
    a.direita = ee( a.direita );
    return dd( a );
}

private No<Tipo> ee( No<Tipo> b ) {
    No<Tipo> c = b.esquerda;
    b.esquerda = c.direita;
    c.direita = b;
    return c;
}

private No<Tipo> dd( No<Tipo> b ) {
    No<Tipo> c = b.direita;
    b.direita = c.esquerda;
    c.esquerda = b;
    return c;
}
```

Árvores AVL
Construção

- Inserir na ordem:
  - **H**, **I**, **J**, **B**, **A**, **E**, **C**, **F**, **D**, **G**, **K** e **L**

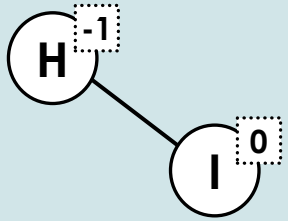
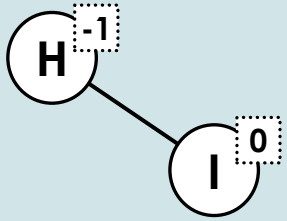
Árvores AVL
Construção

```
avl.put( "H" );
```

Inserção	Rebalanceamento
	 **Não há necessidade de rebalanceamento!**

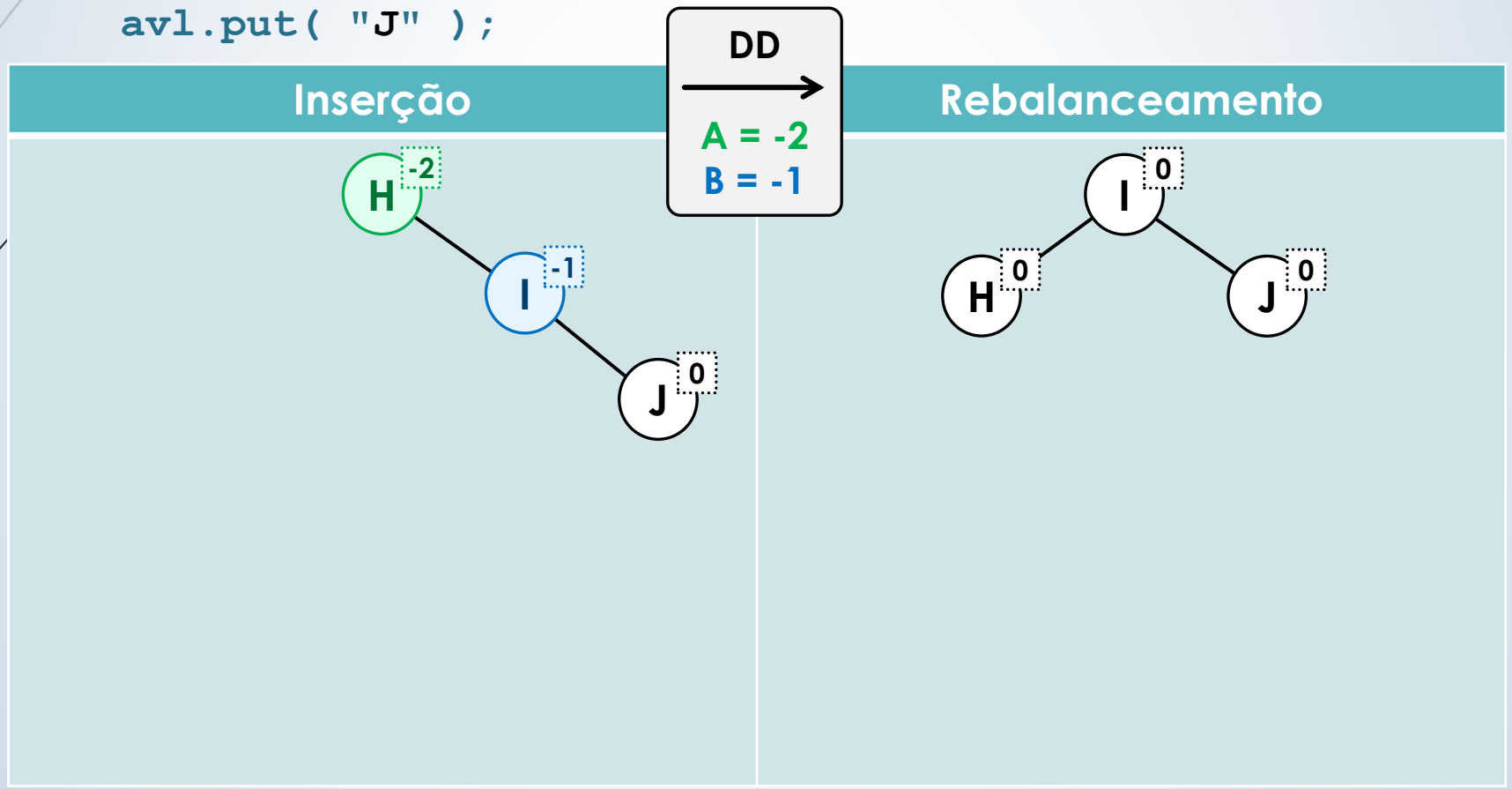
Árvores AVL
Construção

```
avl.put( "I" );
```

Inserção	Rebalanceamento
	
	Não há necessidade de rebalanceamento!

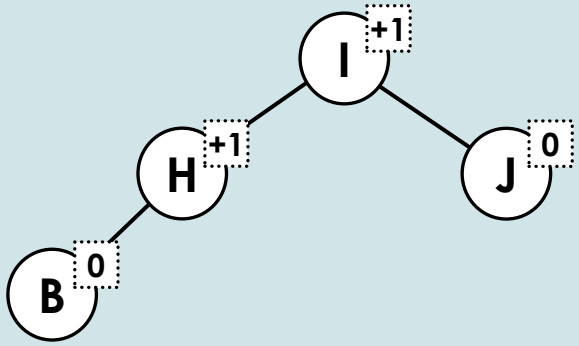
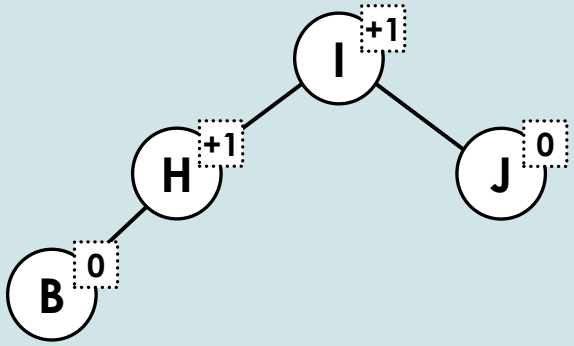
Árvores AVL
Construção

```
avl.put( "J" );
```



Árvores AVL
Construção

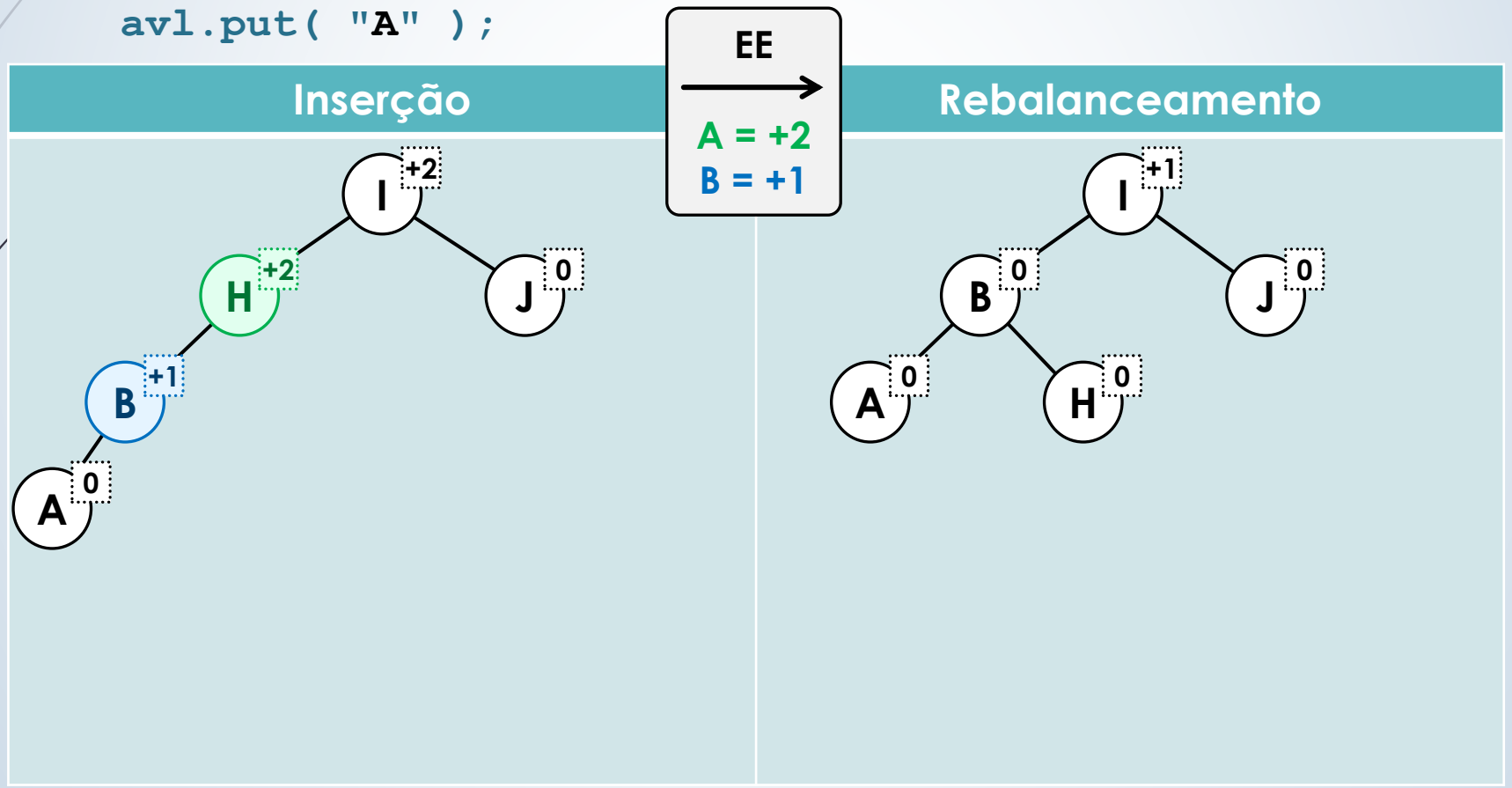
```
avl.put( "B" );
```

Inserção	Rebalanceamento
	

Não há necessidade de rebalanceamento!

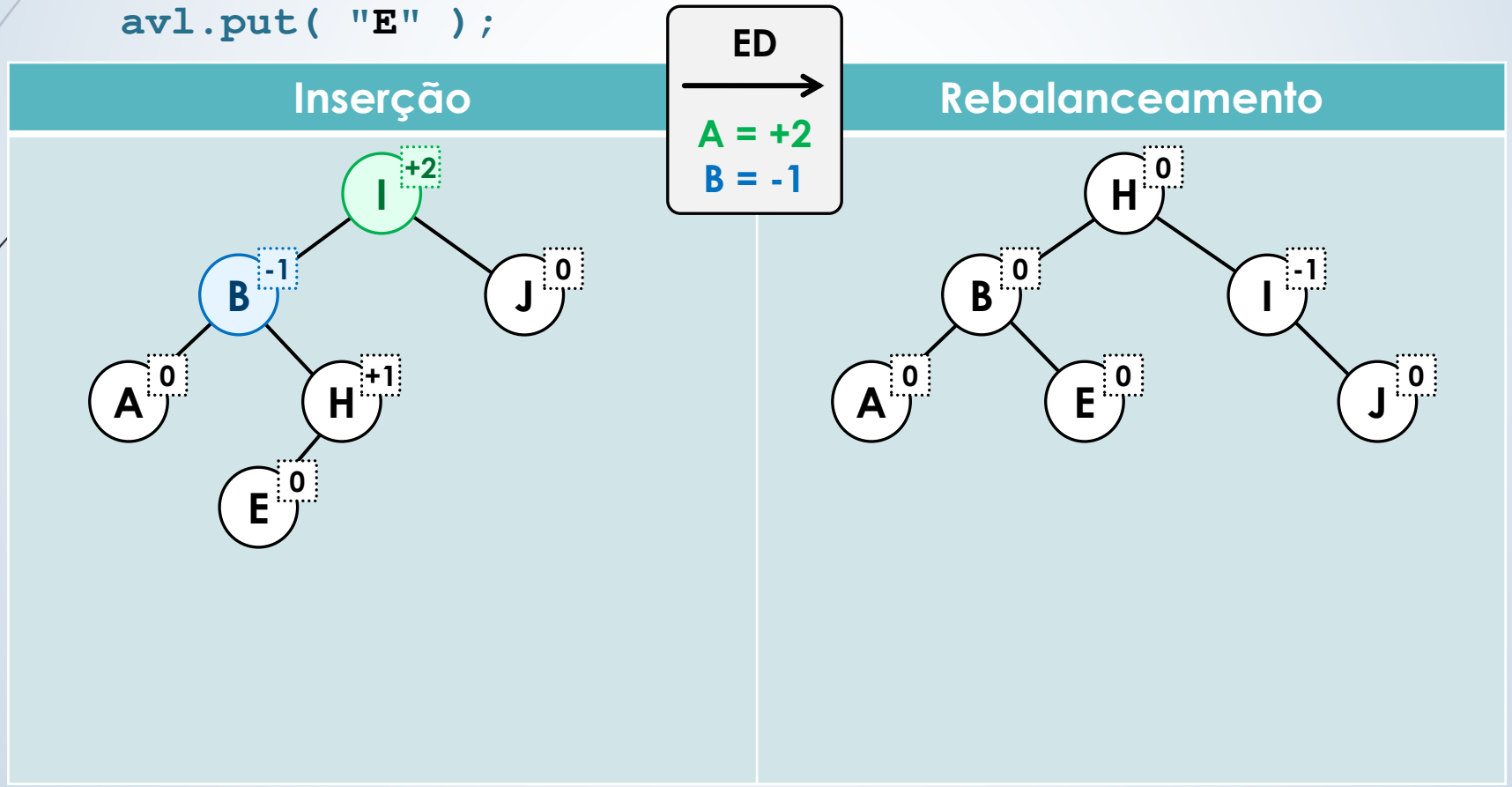
Árvores AVL
Construção

```
avl.put( "A" );
```



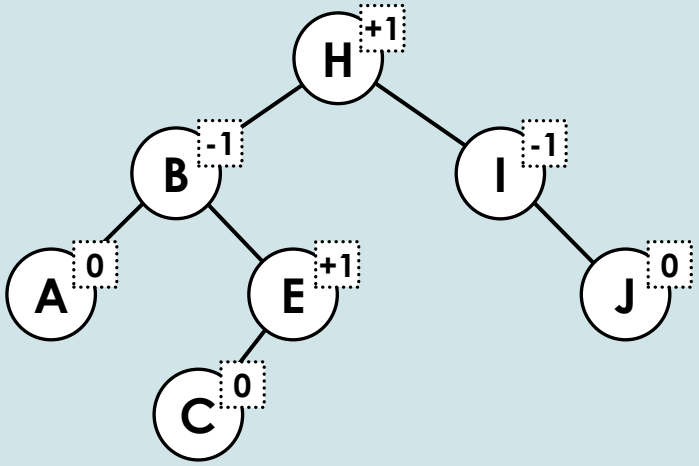
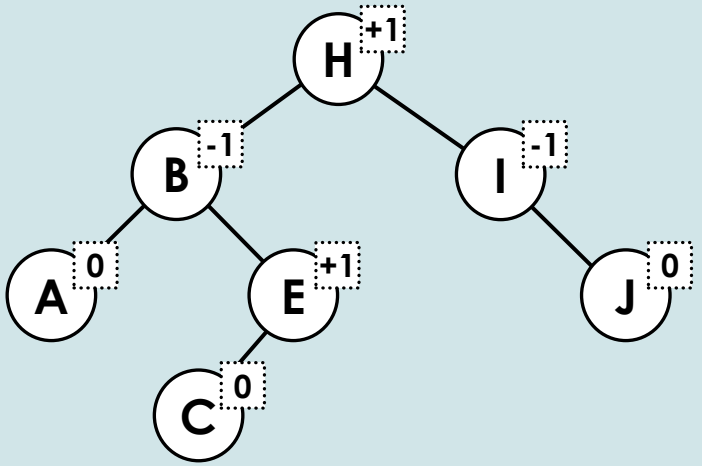
Árvores AVL
Construção

```
avl.put( "E" );
```



Árvores AVL
Construção

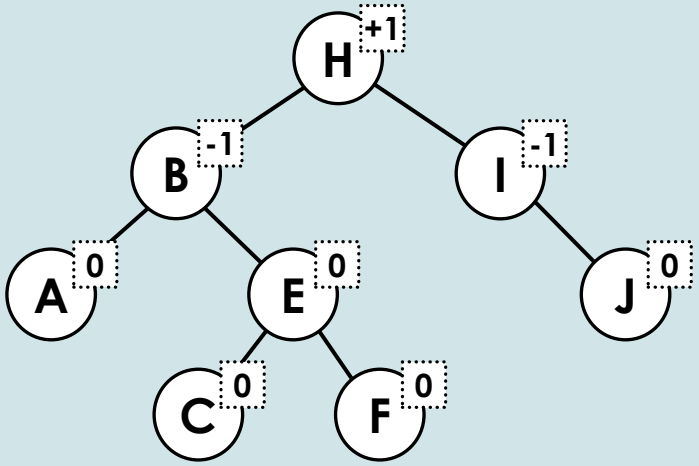
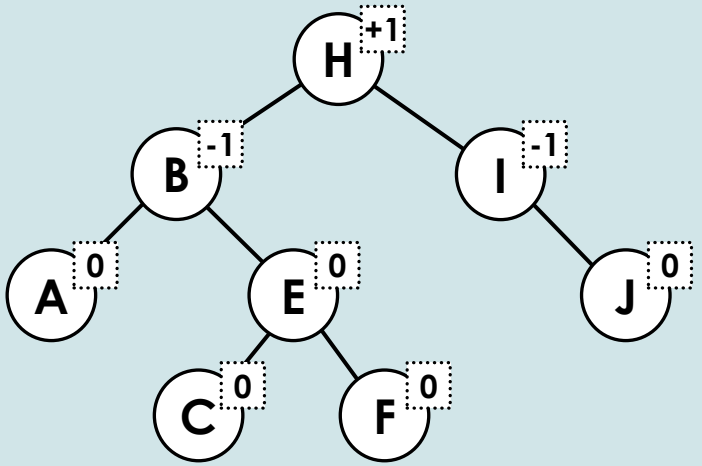
```
avl.put( "C" );
```

Inserção	Rebalanceamento
	

Não há necessidade de rebalanceamento!

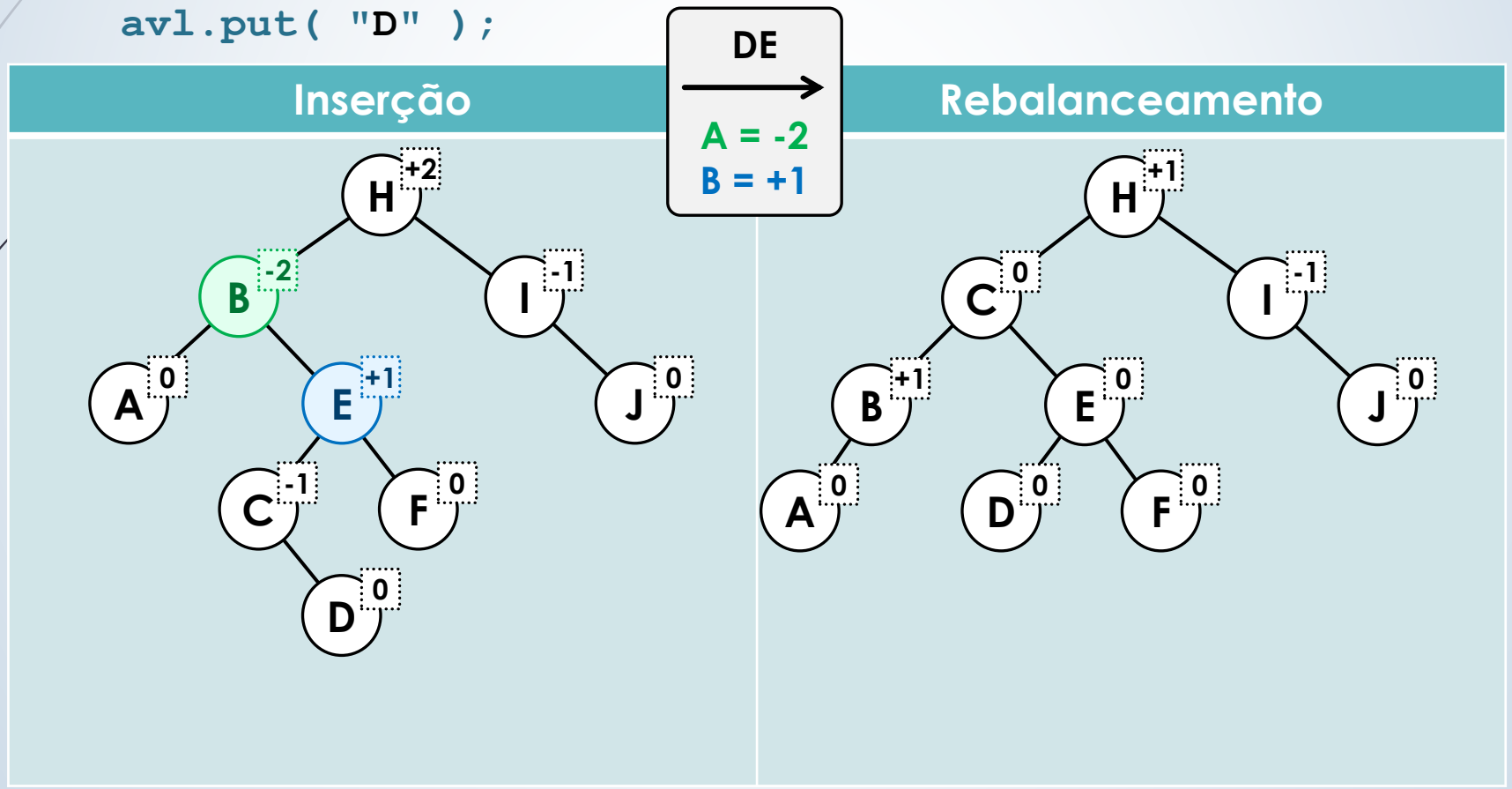
Árvores AVL
Construção

```
avl.put( "F" );
```

Inserção	Rebalanceamento
	
	Não há necessidade de rebalanceamento!

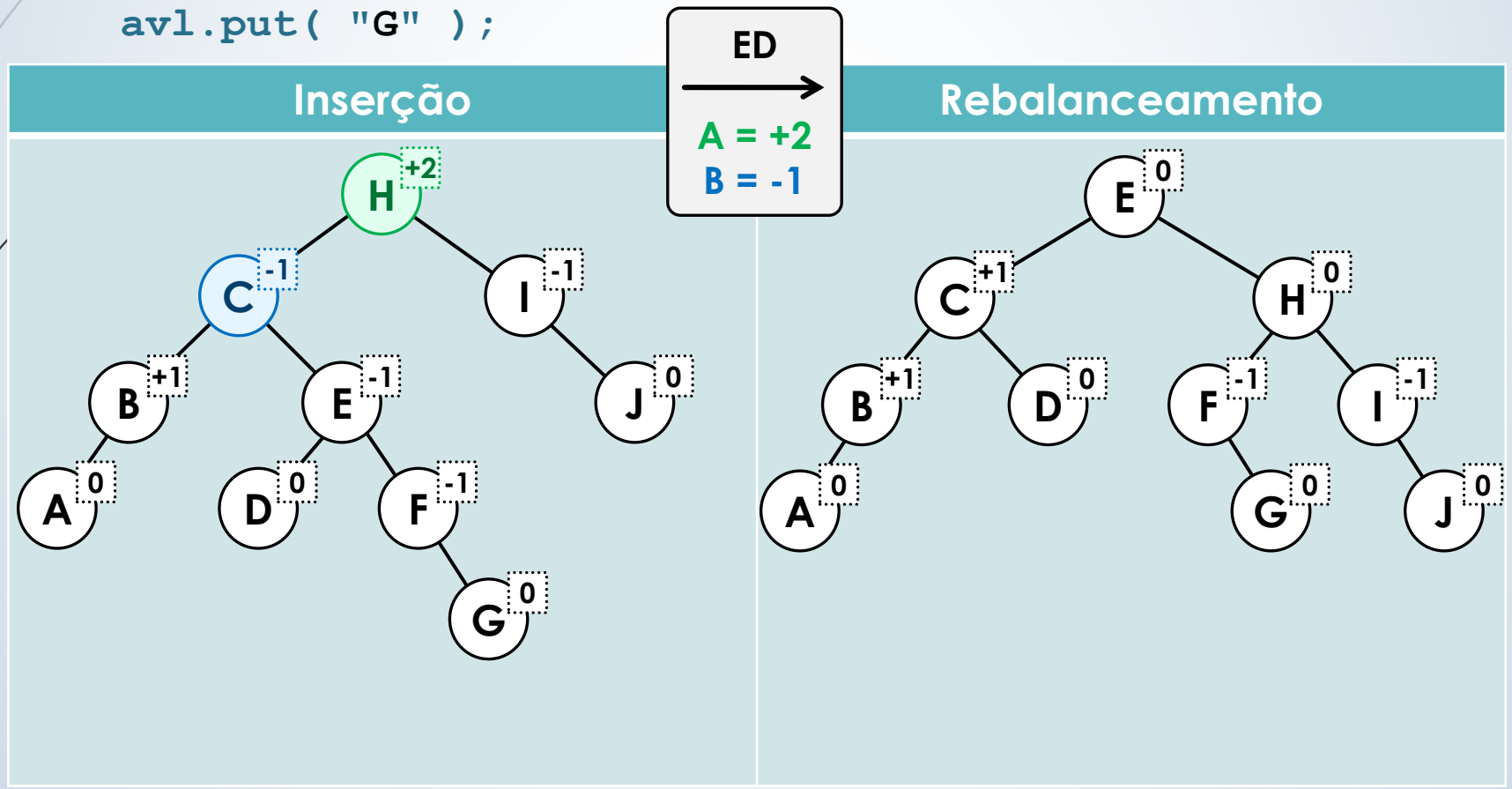
Árvores AVL
Construção

```
avl.put( "D" );
```



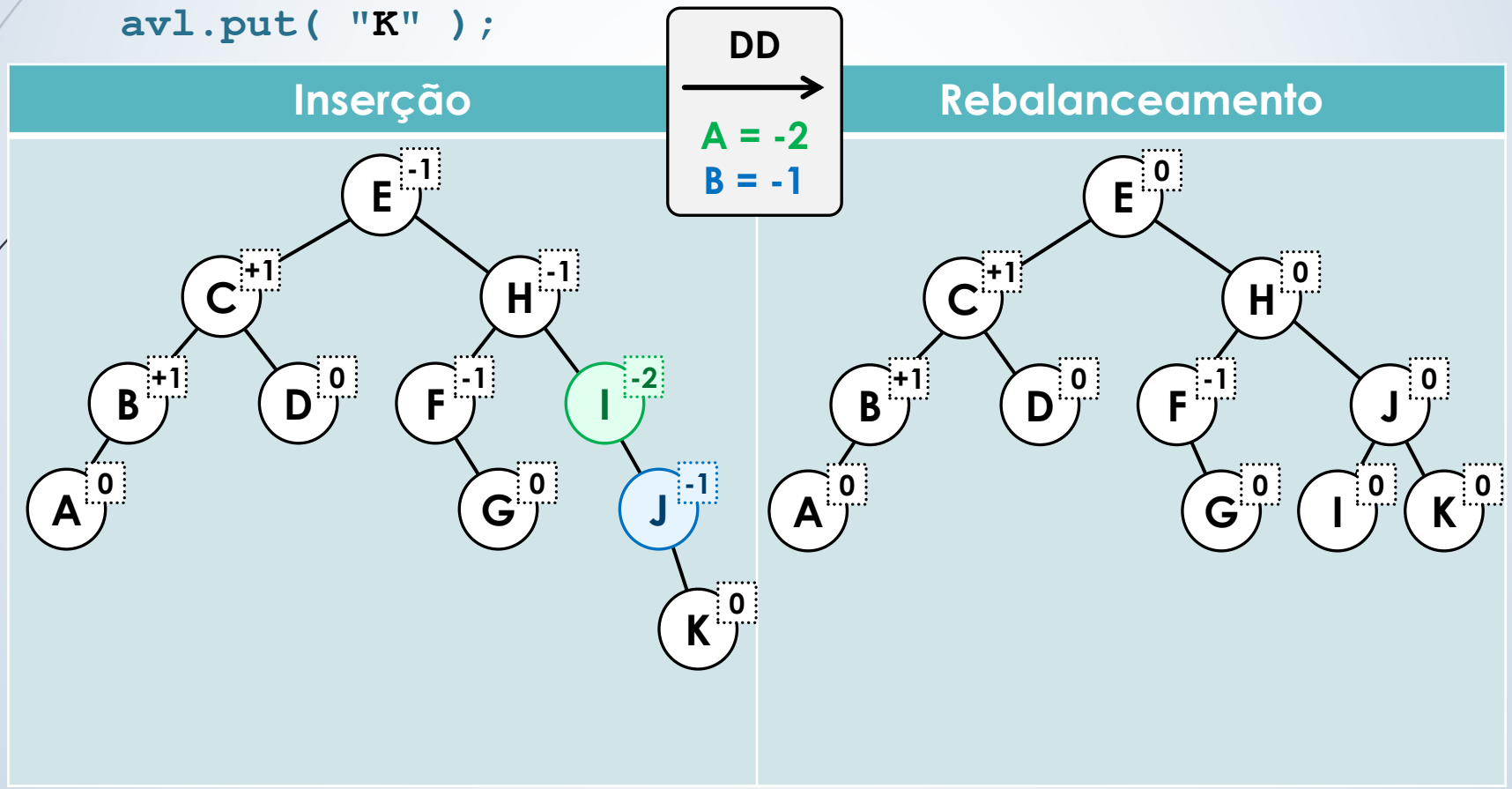
Árvores AVL
Construção

```
avl.put( "G" );
```



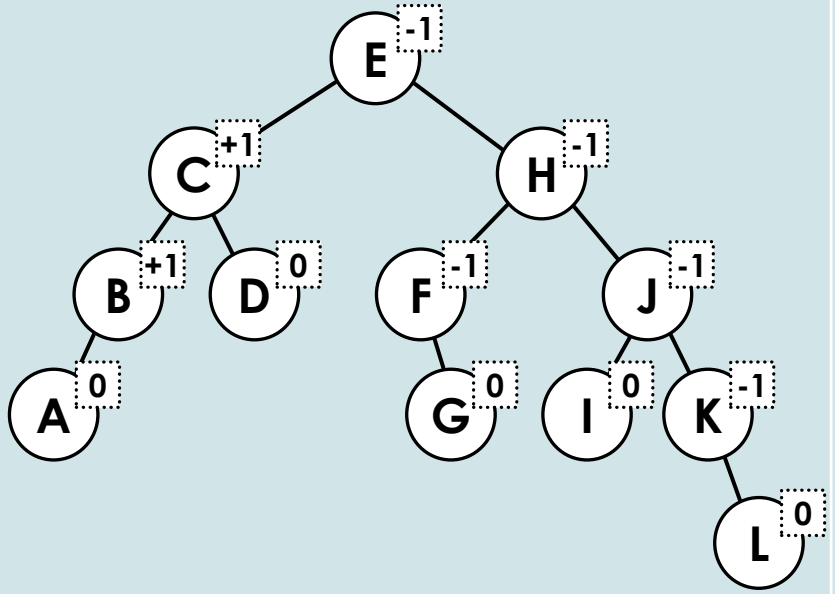
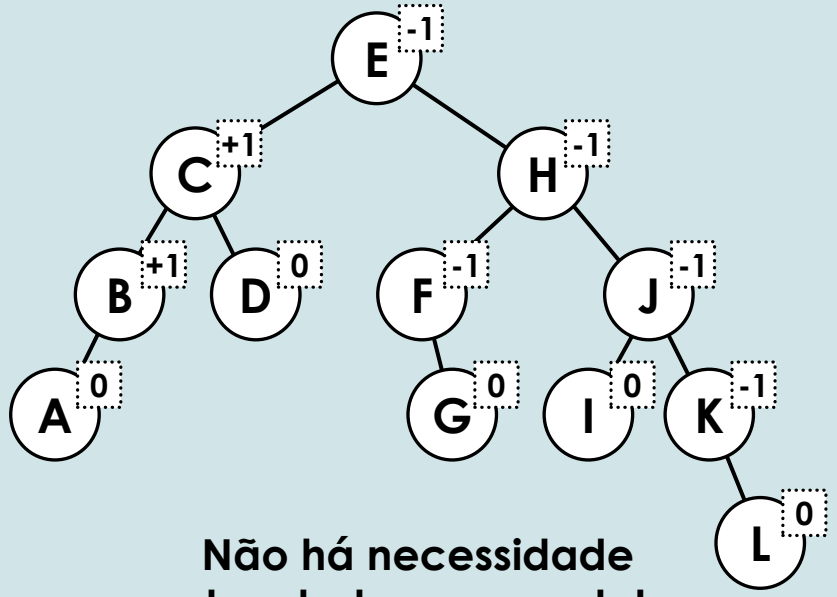
Árvores AVL
Construção

```
avl.put( "K" );
```



Árvores AVL
Construção

```
avl.put( "L" );
```

Inserção	Rebalanceamento
	

Não há necessidade de rebalanceamento!

Bibliografia

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

WEISS, M. A. Data Structures and **Algorithm Analysis in Java**. 3. ed. Pearson Education: New Jersey, 2012. 614 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.