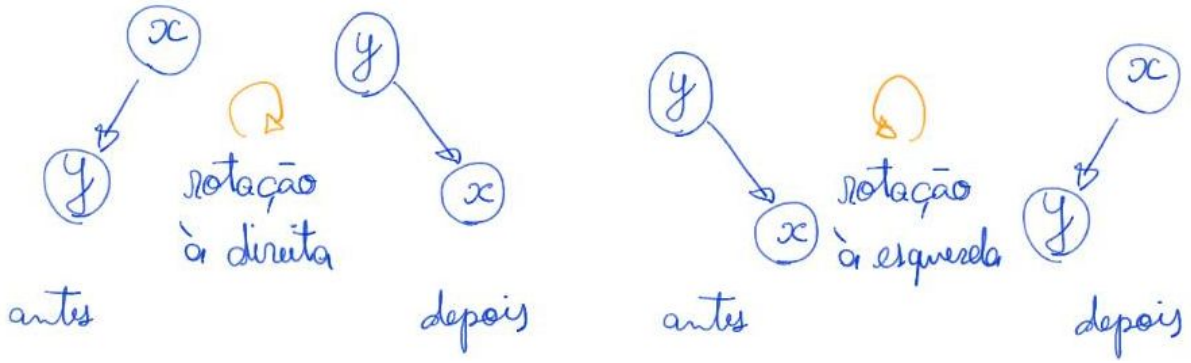
AED2 - Aula 03
Rotações e árvores AVL: definição e inserção

Rotações

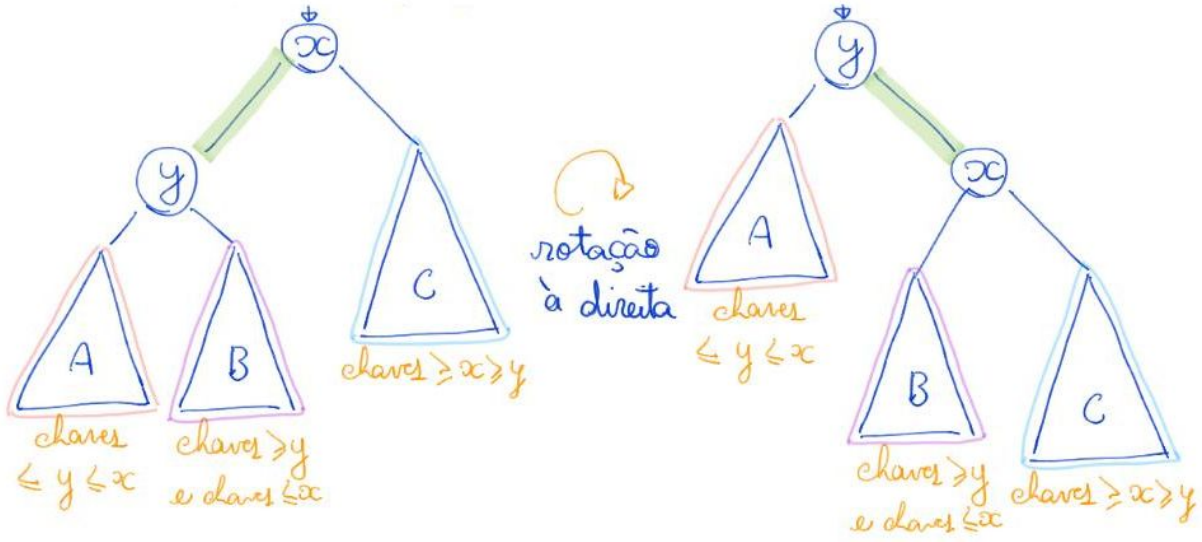
Uma rotação pega um par pai-filho e inverte sua relação.

- temos rotações à esquerda e à direita.

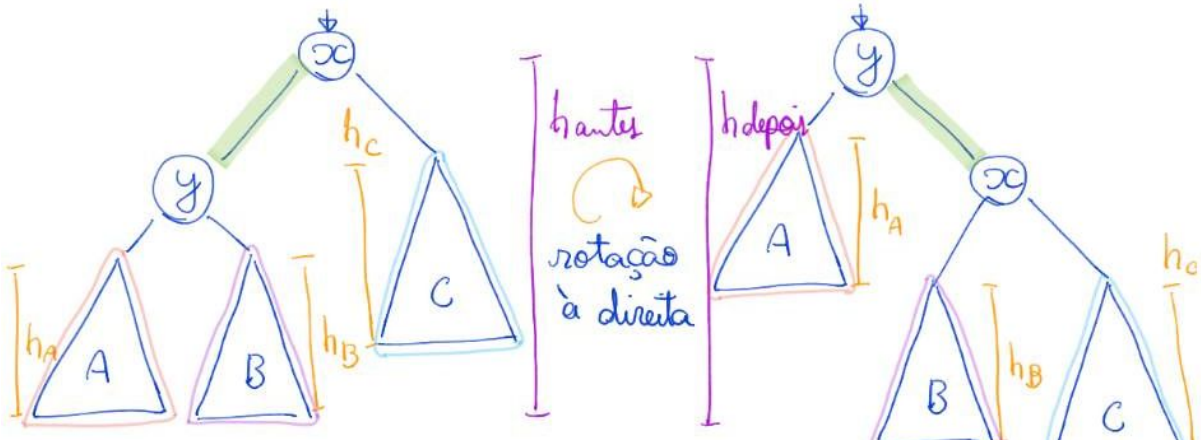



```
Arvore rotacaoDir(Arvore r) {
   Noh *aux;
   aux = r->esq;
   r->esq = aux->dir;
   if (aux->dir != NULL) aux->dir->pai = r;
   aux->dir = r;
   r->pai = aux;
   return aux; }
```

- vamos analisar como uma rotação pode ser realizada
    - utilizando um número pequeno de operações
    - e preservando a propriedade de busca.



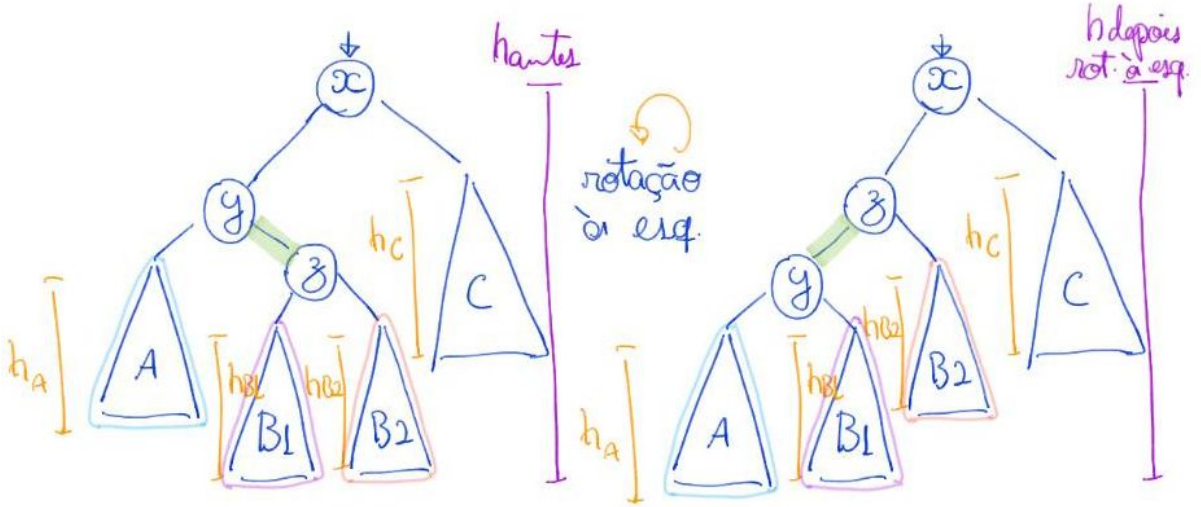
- então vamos analisar o impacto de uma rotação na altura das subárvores envolvidas.



$$h_{antes} = \max\{2 + h_A, 2 + h_B, 1 + h_C\}$$

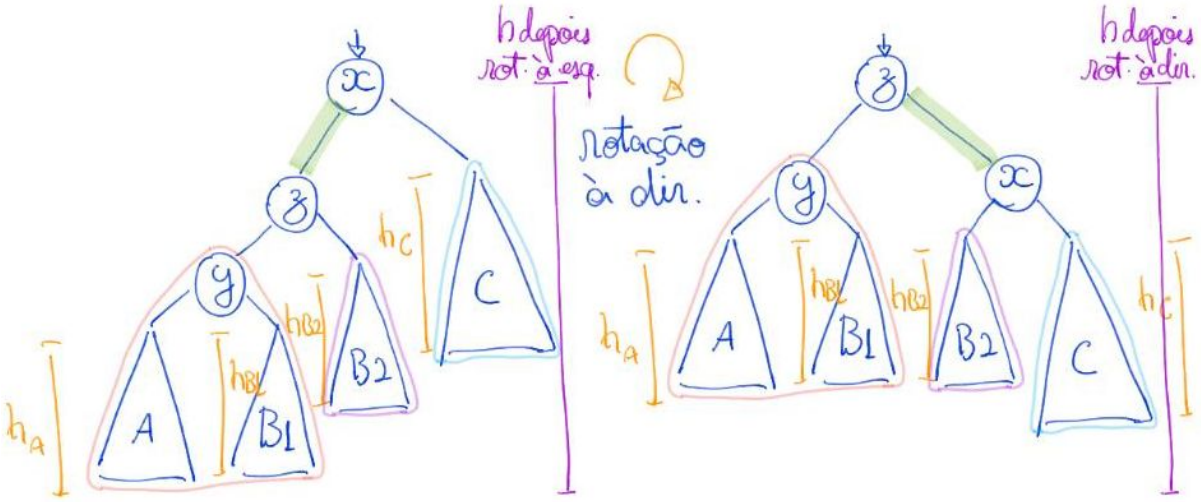
$$h_{depois} = \max\{1 + h_A, 2 + h_B, 2 + h_C\}$$

  - note que a rotação parece interessante se h_a > h_c,
    - pois diminui o impacto de h_a na altura final,
    - mas aumenta o impacto de h_c.
  - observe que o impacto de h_b na altura não é alterado pela rotação.
    - para tanto precisaremos fazer uma rotação dupla, i.e.,
      - uma rotação simples à esquerda
        - que inverte a relação entre y e z,
      - seguida de uma rotação simples à direita entre x e z,
      - e que muda a raiz da subárvore.



$$h_{antes} = \max\{2+h_A, 3+h_{B1}, 3+h_{B2}, 1+h_C\}$$

$$h_{depois\ rot.\ à\ esq.} = \max\{3+h_A, 3+h_{B1}, 2+h_{B2}, 1+h_C\}$$



$$h_{depois\ rot.\ à\ dir.} = \max\{2+h_A, 2+h_{B1}, 2+h_{B2}, 2+h_C\}$$

- Verifique que a propriedade de busca é preservada
  - e que o impacto de h_b1 e h_b2 na altura final é reduzido
    - enquanto o impacto de h_c aumenta.

Árvores AVL

AVL vem dos sobrenomes dos seus inventores: Adelson-Velsky and Landis.

Definições:

- O fator de balanceamento de um nó é a diferença entre
  - a altura de sua subárvore direita e a altura de sua subárvore esquerda.
- Uma árvore é dita AVL se todos os seus nós tem
  - fator de balanceamento entre -1 e +1.
- Intuitivamente, essa propriedade garante que
  - uma árvore AVL é pouco desbalanceada.
- Veremos que, de fato, tal propriedade

    - limita o pior caso do desbalanceamento dessas árvores.

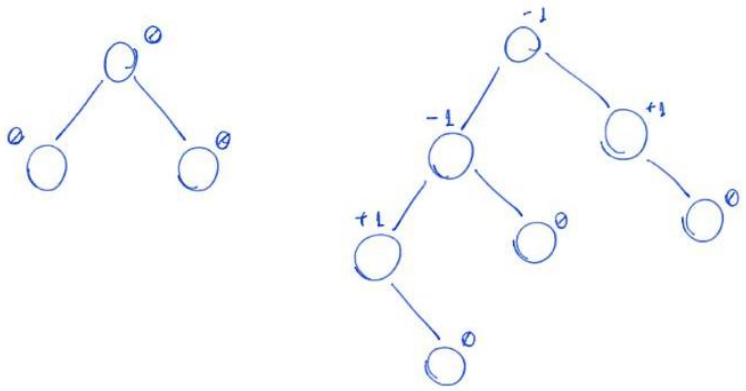
A seguir o código para a estrutura de um nó de árvore AVL:

```
typedef int Cont;
typedef int Chave;

typedef struct noh
{
   int bal;
   Chave chave;
   Cont conteudo;
   struct noh *pai;
   struct noh *esq;
   struct noh *dir;
} Noh;

typedef Noh *Arvore;
```

Exemplos de árvores AVL:



Lembramos que a organização de uma árvore só muda quando

- ocorrem operações de inserção e remoção.

A seguir vemos como tratar as mudanças decorrentes de uma inserção.

Inserção em árvores AVL

Supomos que o algoritmo recursivo de inserção começa

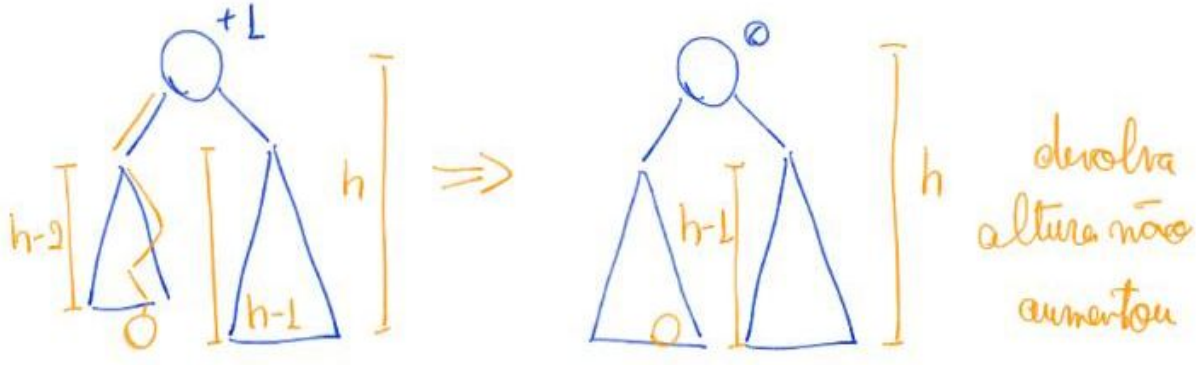
- inserindo o novo nó como uma folha,
    - como ocorre nas árvores binárias de busca comuns,
- e então analisamos o que precisa ser feito na volta da recursão

- quando a altura de uma das subárvores aumentou após a inserção.

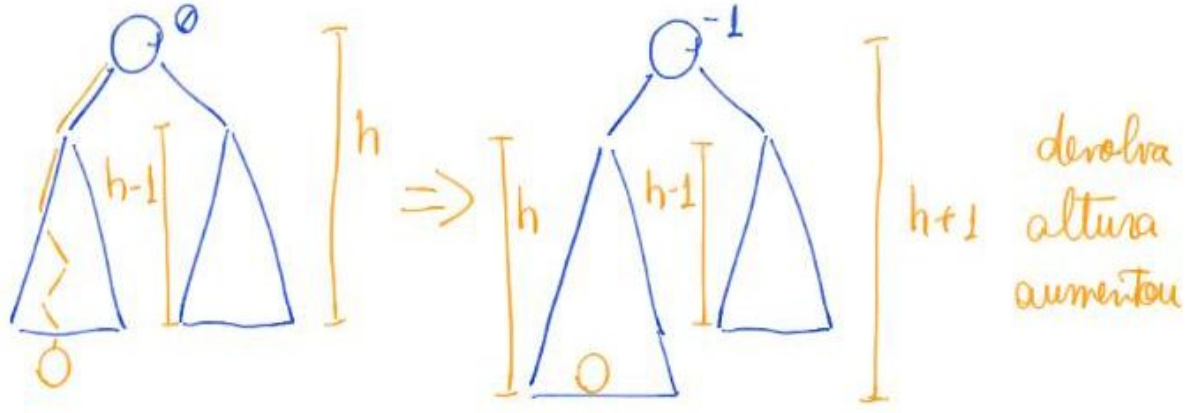
Caso 0: se a altura da subárvore em que ocorreu a inserção não aumentou,
- o algoritmo não precisa realizar correções
- e devolve que a altura da sua árvore não aumentou.

Caso 1: se a sua árvore era vazia,
- crie um nó com dois filhos NULL e balanceamento 0,
- e devolva que a altura da sua árvore aumentou.

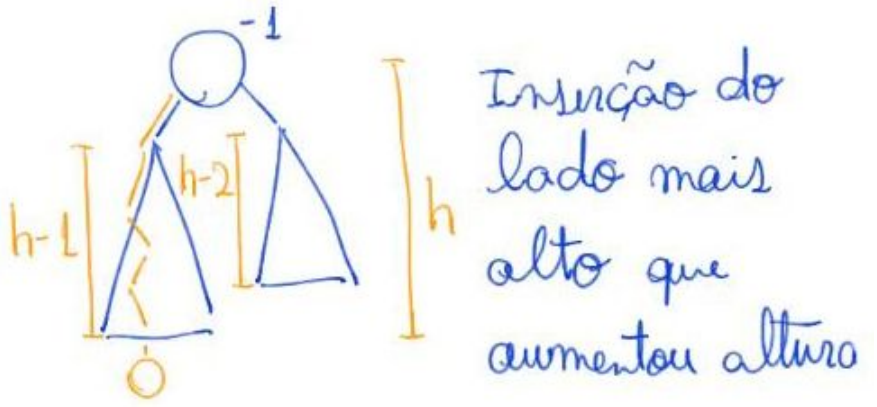
Caso 2: se inseriu na subárvore mais baixa e a altura desta aumentou
- mude o balanceamento da raiz para zero
- e devolva que a altura da sua árvore não aumentou.



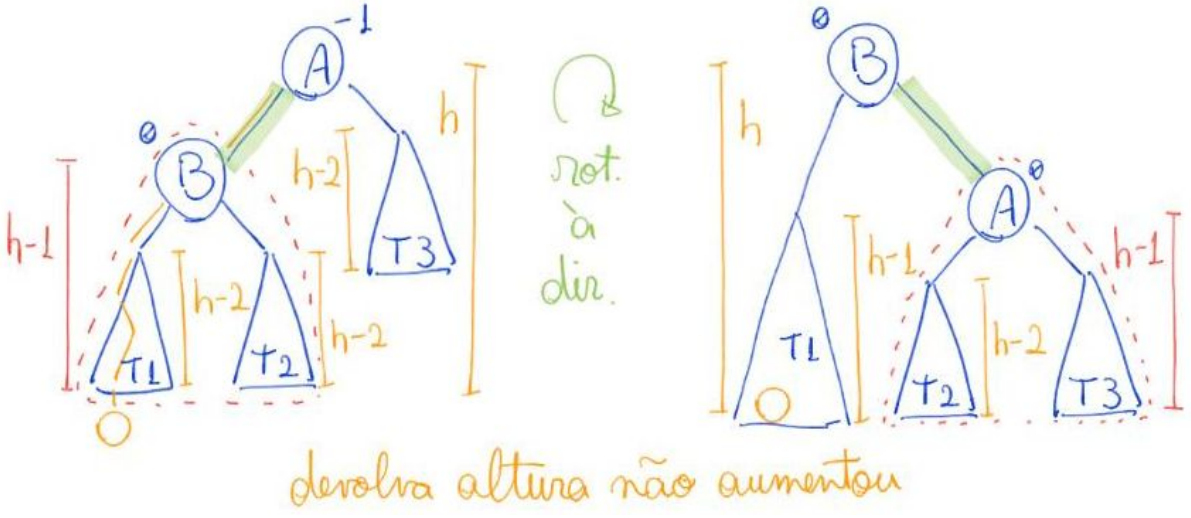
Caso 3: se inseriu em qualquer das subárvores quando as alturas eram iguais (i.e., balanceamento da raiz era 0) e a altura da subárvore aumentou,
- atualize o balanceamento para -1 ou +1 (dependendo do lado da inserção)
- e devolva que a altura da sua árvore aumentou.



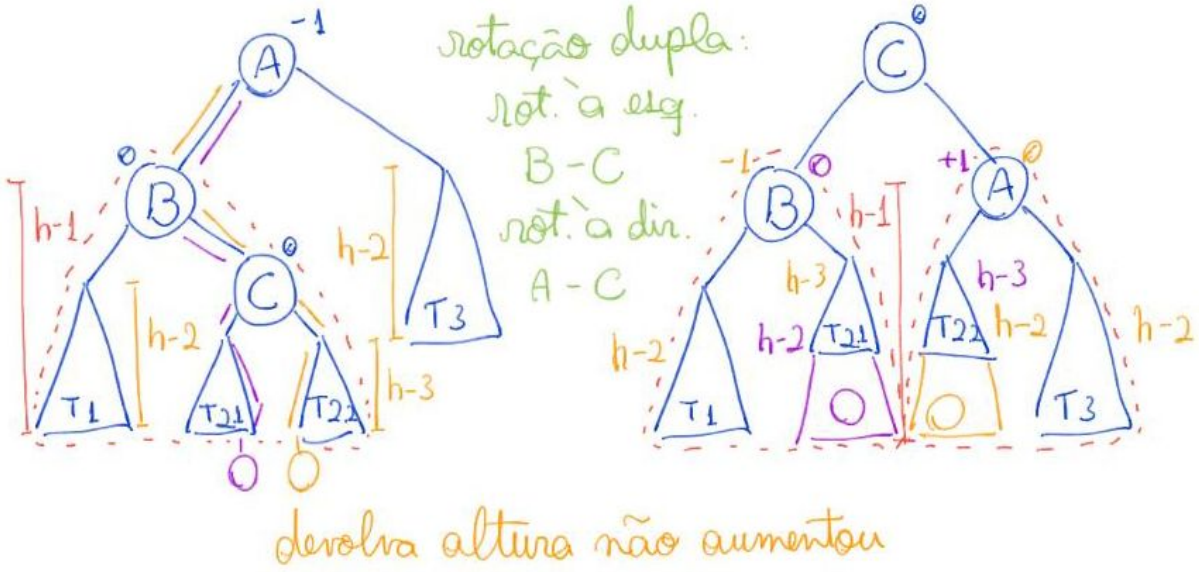
Caso 4: se inseriu na subárvore mais alta e a altura desta aumentou
- é preciso realizar uma ou mais rotações para restaurar a propriedade AVL.



- Caso 4.1: após inserção na subárvore esquerda
    - o fator de balanceamento da raiz desta subárvore é -1.



- Caso 4.2: após inserção na subárvore esquerda
    - o fator de balanceamento da raiz desta subárvore é +1



A seguir o código para inserção em uma árvore AVL:

```
Noh *novoNoh(Chave chave, Cont conteudo)
{
   Noh *novo;
   novo = (Noh *)malloc(sizeof(Noh));
   novo->bal = 0;
   novo->chave = chave;
   novo->conteudo = conteudo;
   novo->esq = NULL;
   novo->dir = NULL;
   //    novo->pai = ??
   return novo;
}

Arvore insereAVL(Noh *r, Noh *novo, int *aumentou_altura)
{
   if (r == NULL) // subárvore era vazia
   {
       novo->pai = NULL;
       *aumentou_altura = 1;
       return novo;
   }
   if (novo->chave <= r->chave) // desce à esquerda
   {
       r->esq = insereAVL(r->esq, novo, aumentou_altura);
       r->esq->pai = r;
       if (*aumentou_altura == 1) // altura da subárvore esquerda
aumentou após inserção
       {
           if (r->bal == +1) // inseriu do lado mais baixo
           {
               r->bal = 0;
               *aumentou_altura = 0;
           }
           else if (r->bal == 0) // dois lados tinham a mesma altura
           {
```

```
                r->bal = -1;
                *aumentou_altura = 1;
            }
            else if (r->bal == -1) // inseriu do lado mais alto
            {
                if (r->esq->bal == -1) // inseriu à esquerda do filho
esquerdo
                {
                    // rotação simples a direita
                    r = rotacaoDir(r);
                    r->dir->bal = 0;
                }
                else // r->esq->bal == +1 - inseriu à direita do
filho esquerdo
                {
                    // rotação dupla
                    r->esq = rotacaoEsq(r->esq);
                    r = rotacaoDir(r);
                    if (r->bal == 0)
                    {
                        r->esq->bal = 0;
                        r->dir->bal = 0;
                    }
                    else if (r->bal == -1)
                    {
                        r->esq->bal = 0;
                        r->dir->bal = +1;
                    }
                    else // r->bal == +1
                    {
                        r->esq->bal = -1;
                        r->dir->bal = 0;
                    }
                }
                r->bal = 0;
```

```
                *aumentou_altura = 0;
            }
        }
    }
    else // desce à direita
    {
       // complementar à inserção à esquerda (preencher essa parte é
um bom exercício)
    }
    return r;
}

Arvore inserir(Arvore r, Chave chave, Cont conteudo)
{
    int aumentou_altura;
    Noh *novo = novoNoh(chave, conteudo);
    return insereAVL(r, novo, &aumentou_altura);
}
```

- Eficiência da inserção continua proporcional à altura da árvore, i.e., O(altura).