

Algoritmos e Estruturas de Dados 2 (AED2)

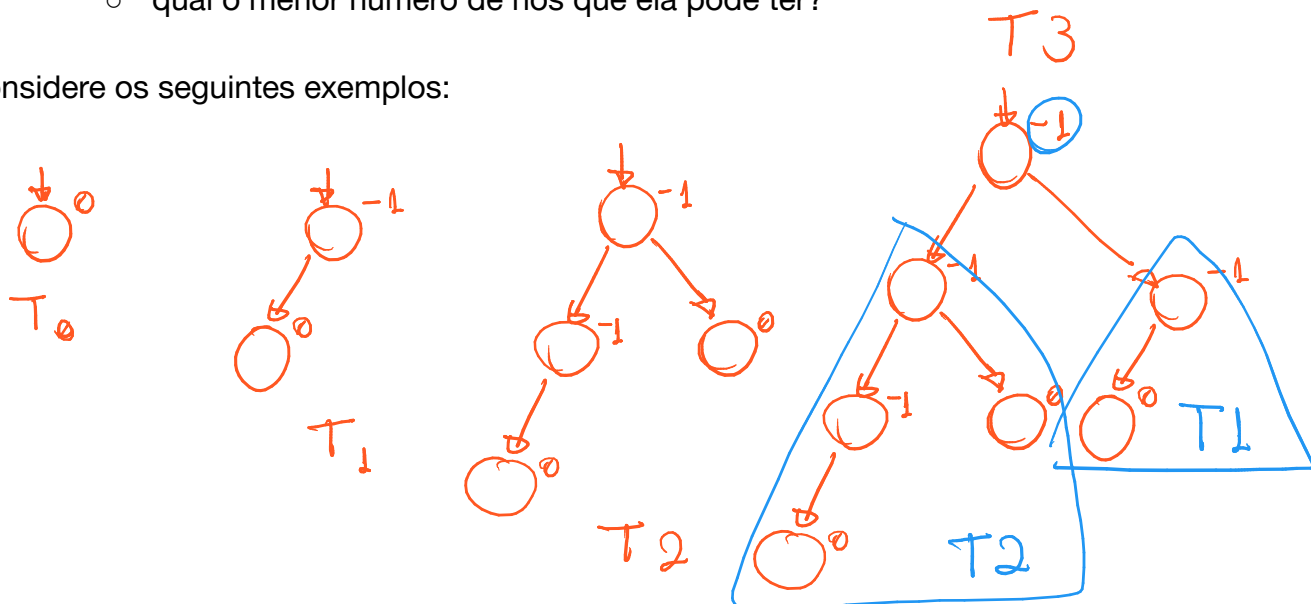
Árvores AVL: altura máxima e remoção

Altura máxima de árvores AVL

Quão esparsa pode ser uma árvore AVL?

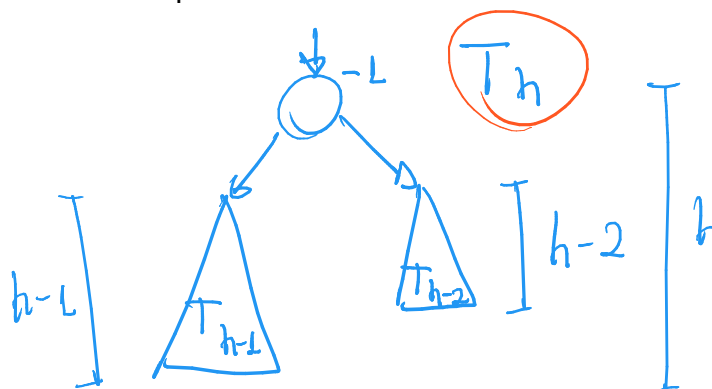
- Isto é, dada uma árvore AVL de altura h
 - qual o menor número de nós que ela pode ter?

Considere os seguintes exemplos:



Observe que, a regra recursiva de formação

- dessas árvores AVL esparsas/desbalanceadas é



- ou seja, T_h é composta por um nó raiz cujo
 - filho esquerdo é T_{h-1}
 - e o filho direito é T_{h-2}
- Ou seja, os filhos são árvores AVL com o menor número de nós possível.

Seja $N(h)$ o número de nós da árvore T_h . Temos que

- $N(0) = 1$
- $N(1) = 2$
- Para $h \geq 2$, $N(h) = 1 + N(h-1) + N(h-2)$

Note que, $N(h)$ é o menor número de nós que uma árvore AVL de altura h pode ter.

Sendo $N(h) = N(h - 1) + N(h - 2) + 1$ para $h \geq 2$, vamos expandir essa recorrência

- $N(0), N(1), N(2), N(3), N(4), N(5), N(6), \dots$
- 1, 2, 4, 7, 12, 20, 33, ...
 - Enxergam um padrão?

Vamos comparar com a expansão da sequência de Fibonacci

- $Fib(0), Fib(1), Fib(2), Fib(3), Fib(4), Fib(5), Fib(6), Fib(7), Fib(8), \dots$
- 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Comparando as sequências podemos perceber que

- $N(h) = Fib(h+2) - 1$
 - o que pode ser provado usando indução matemática.

Mas, $Fib(h+2) \geq 2^{h/2} = (2^{1/2})^h \approx 1,41^h$

- Para verificar isso, perceba que
 - $Fib(h+2) = Fib(h+1) + Fib(h) \geq 2Fib(h)$
- Ou seja, a cada dois incrementos no índice
 - o valor na sequência de Fibonacci pelo menos dobra.

Assim, seja n o número de nós de uma árvore AVL de altura h . Temos que

$$n \geq N(h) = Fib(h+2) - 1 \geq 2^{h/2} - 1$$

- Logo $2^{h/2} \leq n+1$
- Aplicando \lg dos dois lados $h/2 \leq \lg(n+1)$
- Portanto $h \leq 2 \lg(n+1)$
- Ou seja, a altura de uma árvore AVL é no máximo
 - i.e., $O(\lg n)$

Bônus:

- É possível fazer uma análise mais precisa
 - em que mostramos que $Fib(h) \geq 1,618^h$.
 - valor que deriva da razão áurea.
- Usando esse limitante inferior mais preciso para $Fib(h)$ temos

$$n \geq N(h) = Fib(h+2) - 1 \geq 1,618^{h+2} - 1$$

$$1,618^{h+2} \leq n+1 \Rightarrow h+2 \leq \log_{1,618}(n+1)$$

$$h+2 \leq \lg(n+1) / \lg 1,618 \approx 1,44 \lg(n+1)$$

- Portanto,

$$h \leq 1,44 \lg(n+1) = O(\lg n)$$

Algoritmos e Estruturas de Dados 2 (AED2)

Árvores AVL: remoção com códigos

Similar aos casos da inserção, mas um tanto mais complexo.

Supomos que o algoritmo recursivo de remoção começa

- transformando, se necessário, a remoção do nó alvo
 - na remoção de um nó com no máximo um filho,
 - como ocorre nas árvores binárias de busca comuns.
- Então, analisamos o que precisa ser feito na volta da recursão,
 - quando a altura de uma das subárvores diminui após uma remoção.

```
Arvore remover(Arvore r, Chave chave)
```

```
{  
  - Noh *alvo = NULL;  
  - int diminuiu_altura;  
  return removeAVL(r, chave, &alvo, &diminuiu_altura);  
  // return removeR(r, chave, &alvo);  
}
```

```
// recebe uma árvore (r) e uma chave "chave"  
// remove (nó) com chave "chave" da árvore (r)  
// devolve nova raiz da árvore
```

```
Arvore removeAVL(Arvore r, Chave chave, (Noh *)*palvo, int  
*pdiminuiu_altura)
```

```
{  
  Noh *aux;  
  int desceu_esq;
```

```
if (*palvo == NULL)  
  { // ainda não encontrei o nó que desejo remover
```

```
    if (r == NULL)  
      return NULL; // a chave alvo não está na árvore
```

Caso 1 do rebalanceamento: se o nó atual é o alvo,

- lembrando que ele tem no máximo um filho,
 - remova o nó,
 - corrija a subárvore resultante,
 - e devolva que a altura

```

if (chave == r->chave)
{ // se encontrou a chave alvo nesse nível
  *palvo = r;

```

```

if (r->esq == NULL && r->dir == NULL)
{ // se nó alvo é folha
  // printf("Caso 1 da remoção\n");
  free(r);
  *pdiminuiu_altura = 1;
  return NULL;
}

```

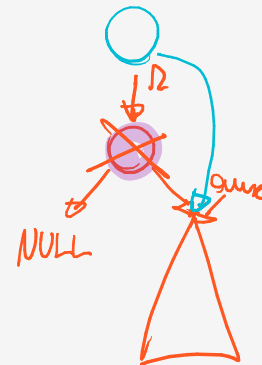


```

if (r->esq == NULL || r->dir == NULL)
{ // se nó alvo tem um único filho
  // printf("Caso 2 da remoção\n");
  if (r->esq == NULL)
    aux = r->dir;
  else // r->dir == NULL
    aux = r->esq;
  free(r);
  *pdiminuiu_altura = 1;

  return aux;
}

```



```

// se nó alvo tem dois filhos
// printf("Caso 3 da remoção\n");
// remove nó alvo trocando ele com maior nó da subárvore

```

esquerda

```

// (*palvo)->esq = removeR(r->esq, -1, palvo); // ERRO

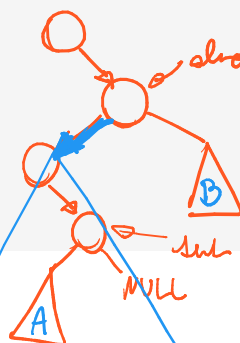
```

PERIGOSO

```

aux = removeAVL(r->esq, -1, palvo, pdiminuiu_altura);
r = *palvo;
desceu_esq = 1;
r->esq = aux;
// return r;
}

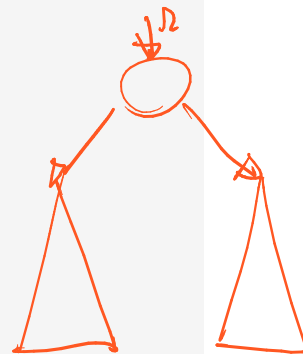
```



```

else // se não encontrou a chave alvo nesse nível
{
    if (chave < r->chave)
    { // desce na árvore à esquerda
        r->esq = removeAVL(r->esq, chave, palvo,
pdiminuiu_altura);
        desceu_esq = 1;
    }
    else
    { // r->chave > chave - ou à direita
        r->dir = removeAVL(r->dir, chave, palvo,
pdiminuiu_altura);
        desceu_esq = 0;
    }
    // return r;
}
}

```



```

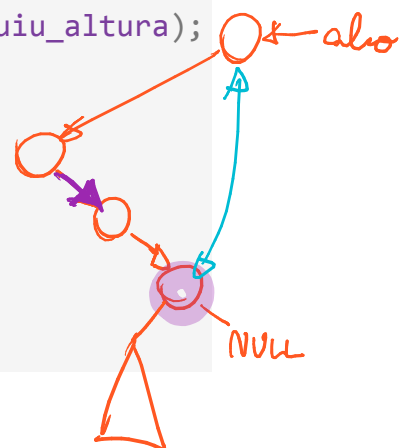
else // *palvo != NULL
{ // encontrei o nó que desejo remover e estou procurando com
quem trocá-lo

```

```

if (r->dir != NULL) // descendo à direita em busca do maior
{
    // Quiz1: por que passei -1 como chave?
    r->dir = removeAVL(r->dir, -1, palvo, pdiminuiu_altura);
    desceu_esq = 0;
    // return r;
}
else // r->dir == NULL
{

```



```
// encontrei o maior elemento da subárvore esquerda pra trocar com o alvo
```

```
printf("chave do predecessor do alvo = %d\n", r->chave);
```

```
if (r != (*palvo)->esq) // Quiz2: por que esse teste?
```

```
{
```

```
    // trocando nó alvo com r e removendo alvo
```

```
    ↗ r->dir = (*palvo)->dir;
```

```
    aux = r->esq;
```

```
    ↗ r->esq = (*palvo)->esq;
```

```
    r->bal = (*palvo)->bal;
```

```
    ↗ free(*palvo);
```

```
    *pdiminuiu_altura = 1;
```

```
    *palvo = r; // necessário na volta da recursão do
```

caso 3 da remoção

```
    return aux;
```

```
}
```

```
else // r == (*palvo)->esq - Quiz2: qual é esse caso?
```

```
{
```

```
    // trocando nó alvo com r e removendo alvo
```

```
    r->dir = (*palvo)->dir;
```

```
    aux = r->esq;
```

```
    r->bal = (*palvo)->bal;
```

```
    free(*palvo);
```

```
    *pdiminuiu_altura = 1;
```

```
    *palvo = r; // necessário na volta da recursão do
```

caso 3 da remoção

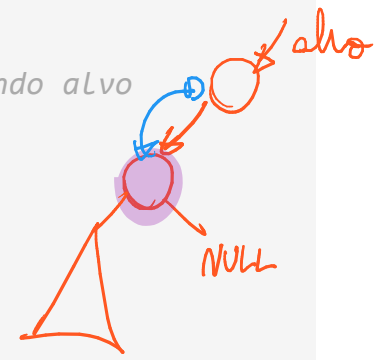
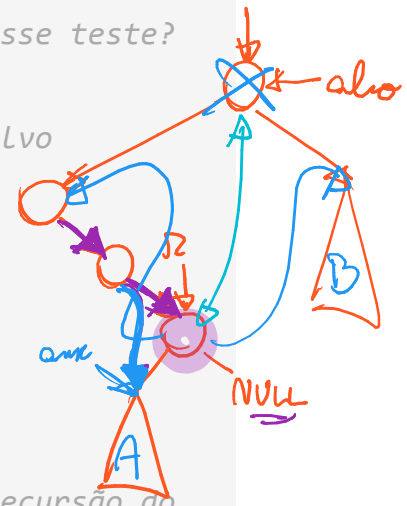
```
    return aux;
```

```
}
```

```
    // return aux e outras linhas comuns podiam vir pra cá
```

```
}
```

```
}
```



Agora vamos tratar da volta da recursão.

Caso 0 do rebalanceamento: se a altura da subárvore em que ocorreu a remoção não diminuiu,

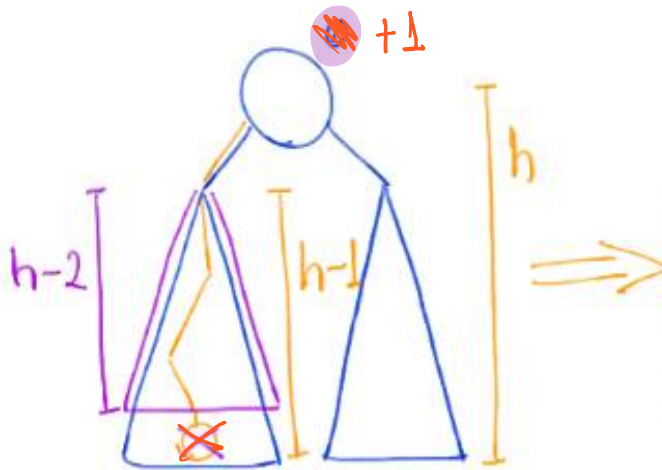
- o algoritmo não precisa realizar correções
- e devolve que a altura da sua árvore não diminuiu.

Já quando a altura de uma das subárvores diminui após uma remoção.

```
if (*pdiminuiu_altura == 1)
{ // corrigir balanceamento em cada caso

    if (desceu esq == 1)
    { // casos em que a remoção ocorreu à esquerda de r
```

Caso 2 do rebalanceamento: se a altura das duas subárvores era igual (i.e., balanceamento da raiz era 0) e a altura da subárvore em que ocorreu a remoção diminuiu,

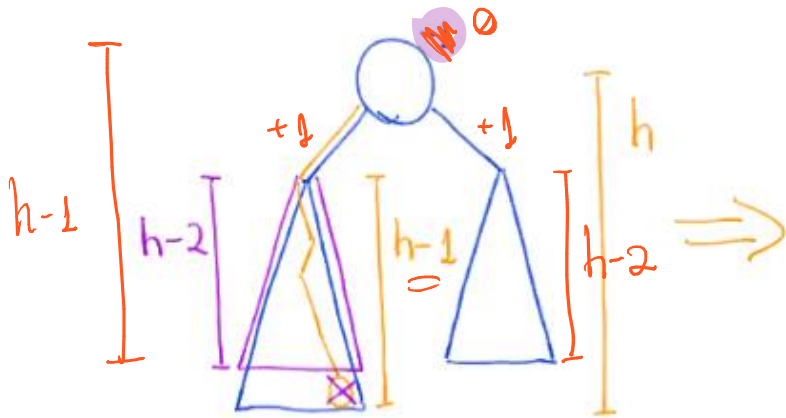


- corrige o fator de balanceamento
- e devolve que a altura da sua árvore *Não diminuiu*

```
if (r->bal == 0)
{ // caso 2 do rebalanceamento - subárvore r estava
balanceada

    r->bal = +1;
    *pdiminuiu_altura = 0;
    // raiz da subárvore não mudou
}
```

Caso 3 do rebalanceamento: se removeu da subárvore mais alta e a altura desta diminuiu



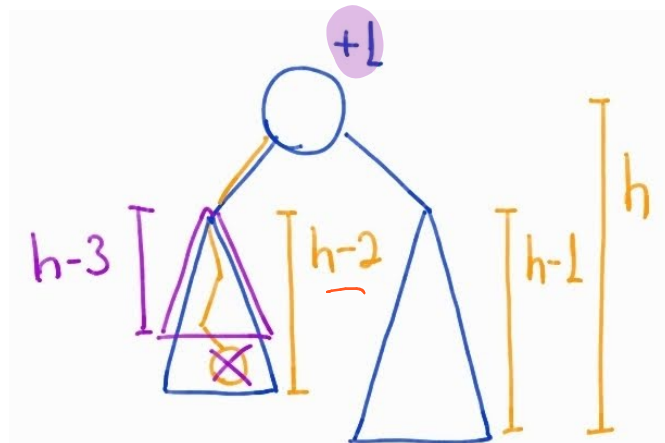
- corrige o fator de balanceamento para 0
- e devolve que a altura da sua árvore *diminuiu*

```

else if (r->bal == -1)
{ // caso 3 do rebalanceamento
  r->bal = 0;
  *pdiminuiu_altura = 1;
  // raiz da subárvore não mudou
}

```

Caso 4 do rebalanceamento: se removeu da subárvore mais baixa e a altura diminuiu



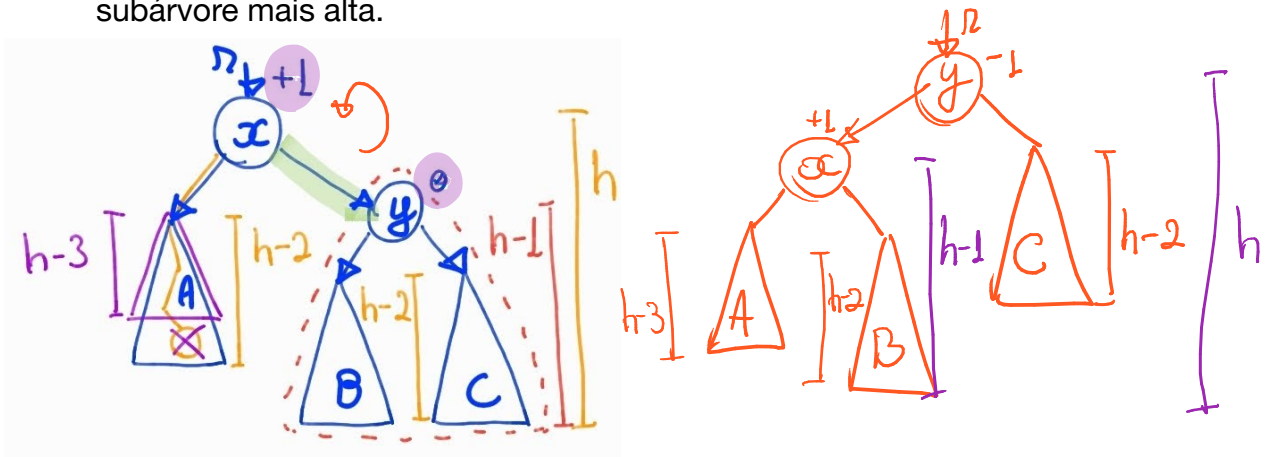
- é preciso realizar uma ou mais rotações para restaurar a propriedade AVL.

```

else // r->bal == +1
{ // caso 4 do rebalanceamento

```


- Caso 4.1 do rebalanceamento: fator de balanceamento 0 na raiz da subárvore mais alta.

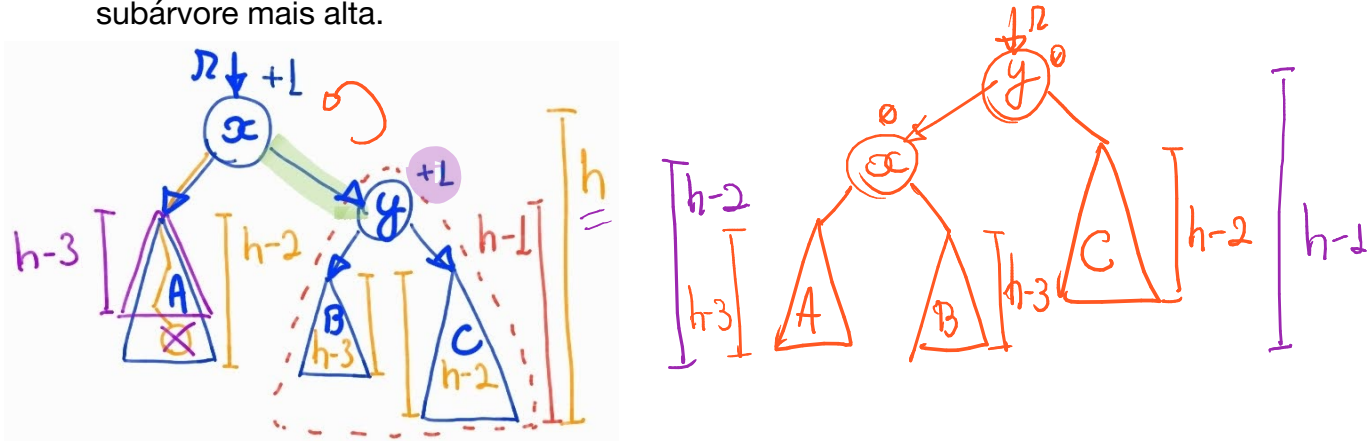


```

if (r->dir->bal == 0)
{ // caso 4.1 do rebalanceamento
  r = rotacaoEsq(r);
  // raiz da subárvore mudou
  r->esq->bal = +1; // suplérfu
  r->bal = -1;
  *pdiminuiu_altura = 0;
}

```

- Caso 4.2 do rebalanceamento - fator de balanceamento +1 na raiz da subárvore mais alta.

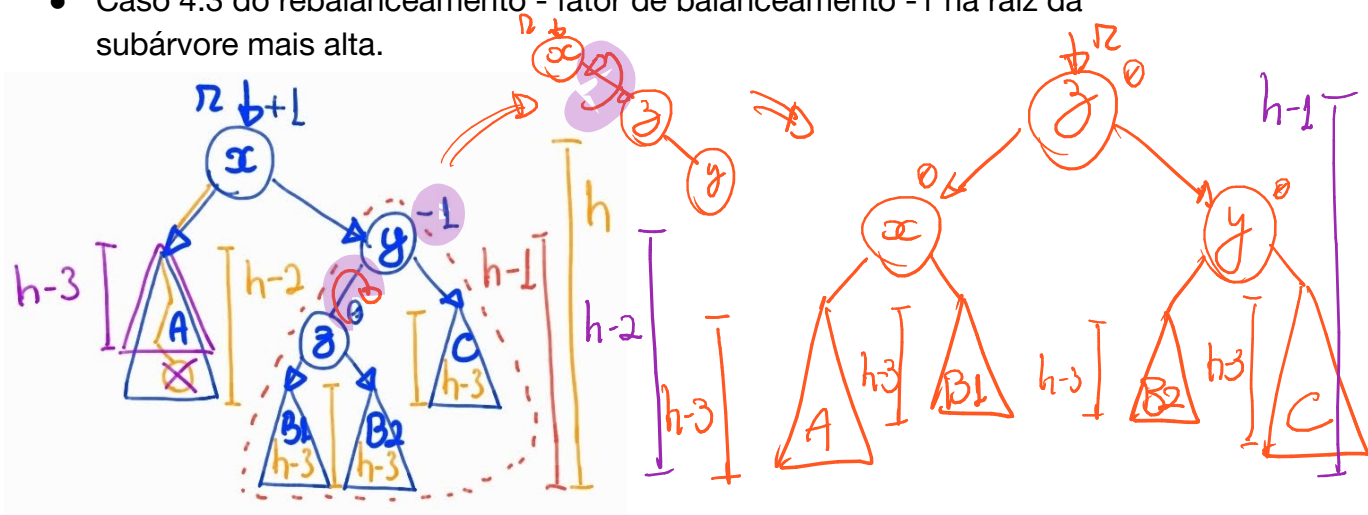


```

else if (r->dir->bal == +1)
{ // caso 4.2 do rebalanceamento
  r = rotacaoEsq(r);
  // raiz da subárvore mudou
  r->esq->bal = 0;
  r->bal = 0;
  *pdiminuiu_altura = 1;
}

```

- Caso 4.3 do rebalanceamento - fator de balanceamento -1 na raiz da subárvore mais alta.



- Quiz3: se o fator de balanceamento de z for $-1/+1$,
 - como ficam os balanceamentos de x e y após as rotações?

```

else // r->dir->bal == -1
{
    // caso 4.3 do rebalanceamento
    - r->dir = rotacaoDir(r->dir);
    - r = rotacaoEsq(r); // raiz da subárvore mudou
    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;
    }
    *pdiminuiu_altura = 1;
}
}
}
else // desceu_esq == 0
{
    // casos em que a remoção ocorreu à direita de r
    // preencher essa parte é um bom exercício
}
}
return r;
}

```

Note que, realizamos um número de operações constante por nível da árvore.

- Assim, a eficiência da remoção é proporcional à altura, i.e., $O(\text{altura})$.