MC-202 — Aula 13
Árvores Balanceadas

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

Roteiro

Introdução

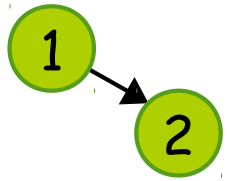
Vamos inserir alguns número em uma Árvore de Busca:

Introdução

Vamos inserir alguns número em uma Árvore de Busca:

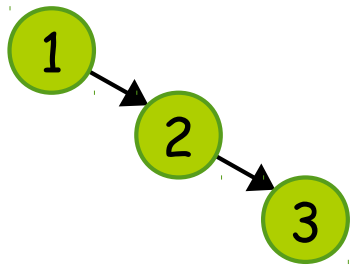
Introdução

Vamos inserir alguns número em uma Árvore de Busca:



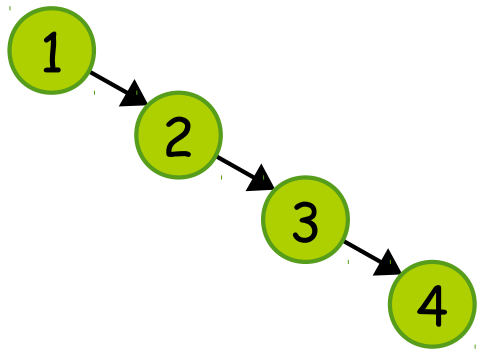
Introdução

Vamos inserir alguns número em uma Árvore de Busca:



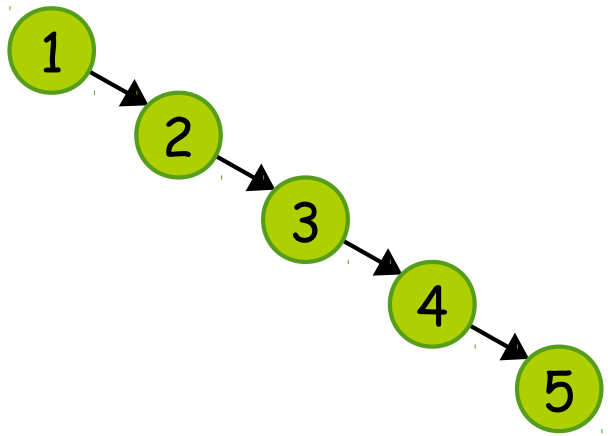
Introdução

Vamos inserir alguns número em uma Árvore de Busca:



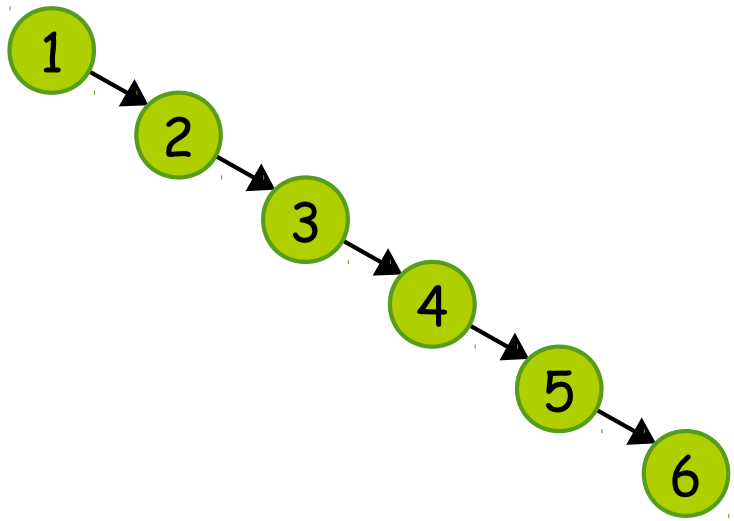
Introdução

Vamos inserir alguns número em uma Árvore de Busca:



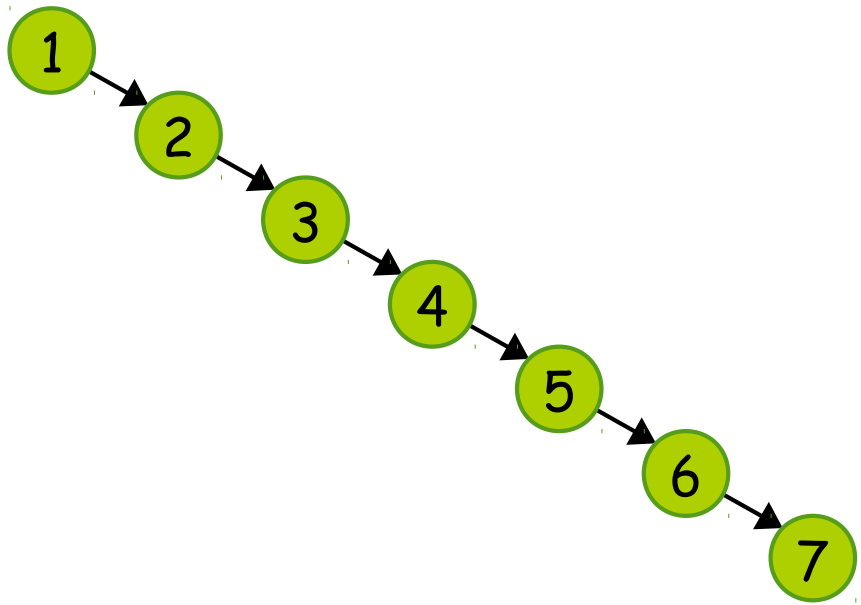
Introdução

Vamos inserir alguns número em uma Árvore de Busca:



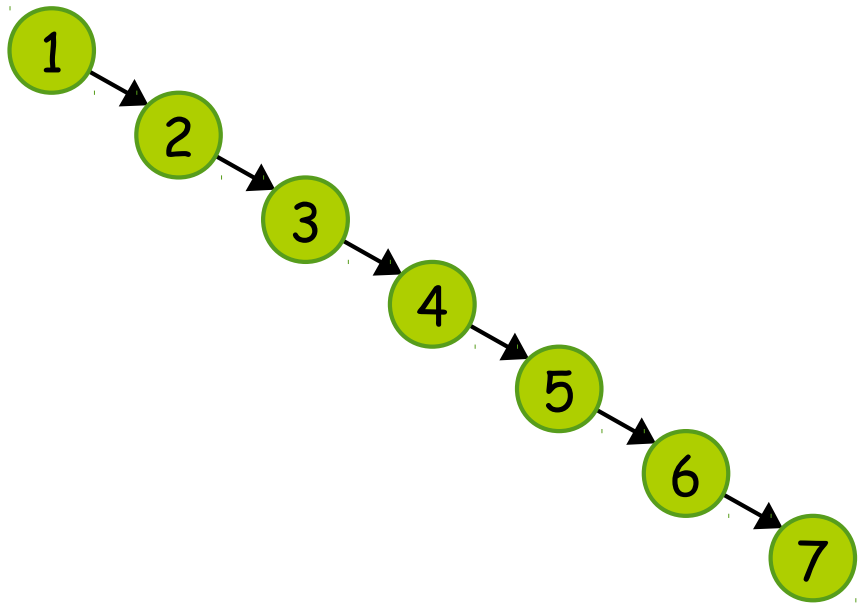
Introdução

Vamos inserir alguns número em uma Árvore de Busca:



Introdução

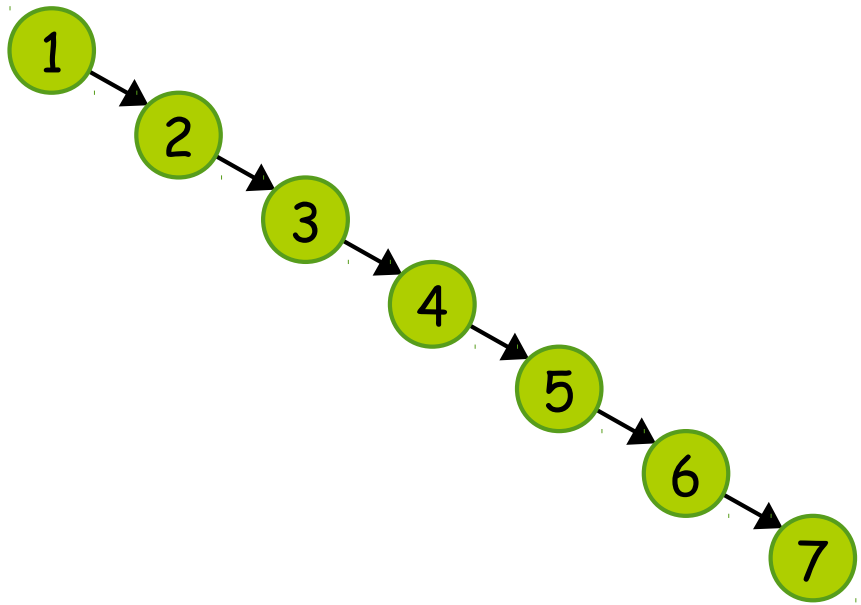
Vamos inserir alguns número em uma Árvore de Busca:



A árvore virou uma lista!

Introdução

Vamos inserir alguns número em uma Árvore de Busca:



A árvore virou uma lista!
Problema: A busca pelo 7 tem que percorrer todos elementos!

Solução

Como diminuir o tempo de busca?

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução 1:

Ao inserir um nó, calcular a árvore de altura mínima!

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução 1:

Ao inserir um nó, calcular a árvore de altura mínima!

- **Busca:** no máximo a altura da árvore mínima: $\lceil \log_2(n+1) \rceil$

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução 1:

Ao inserir um nó, calcular a árvore de altura mínima!

- **Busca:** no máximo a altura da árvore mínima: $\lceil \log_2(n+1) \rceil$
- **Inserção:** criar uma árvore de altura mínima: $\Theta(n)$

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução 1:

Ao inserir um nó, calcular a árvore de altura mínima!

- **Busca:** no máximo a altura da árvore mínima: $\lceil \log_2(n+1) \rceil$
- **Inserção:** criar uma árvore de altura mínima: $\Theta(n)$

Solução 2:

Árvore de altura quase mínima!

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução 1:

Ao inserir um nó, calcular a árvore de altura mínima!

- **Busca:** no máximo a altura da árvore mínima: $\lceil \log_2(n+1) \rceil$
- **Inserção:** criar uma árvore de altura mínima: $\Theta(n)$

Solução 2:

Árvore de altura quase mínima!

- **Busca:** no máximo a altura da árvore: $O(\log_2 n)$

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução 1:

Ao inserir um nó, calcular a árvore de altura mínima!

- **Busca:** no máximo a altura da árvore mínima: $\lceil \log_2(n+1) \rceil$
- **Inserção:** criar uma árvore de altura mínima: $\Theta(n)$

Solução 2:

Árvore de altura quase mínima!

- **Busca:** no máximo a altura da árvore: $O(\log_2 n)$
- **Inserção:** "consertar" o balanceamento: $O(\log_2 n)$

Solução

Como diminuir o tempo de busca?

- Cada subárvore divida o número de elementos tanto quanto possível!
- Modificamos a árvore sempre que um novo elemento é inserido

Solução 1:

Ao inserir um nó, calcular a árvore de altura mínima!

- **Busca:** no máximo a altura da árvore mínima: $\lceil \log_2(n+1) \rceil$
- **Inserção:** criar uma árvore de altura mínima: $\Theta(n)$

Solução 2:

Árvore de altura quase mínima!

- **Busca:** no máximo a altura da árvore: $O(\log_2 n)$
- **Inserção:** "consertar" o balanceamento: $O(\log_2 n)$

Exemplos de árvores aproximadas: Árvore AVL, árvore rubro-negra

Árvore AVL

- introduzida por Adelson-Velsky and Evgenii Landis (1962)
- tenta diminuir a diferença entre as alturas das subárvores

Árvore AVL

- introduzida por Adelson-Velsky and Evgenii Landis (1962)
- tenta diminuir a diferença entre as alturas das subárvores

Dado um nó, o **fator de balanceamento** é definido como:

$$\text{fb} = \text{altura da subárvore direita} - \text{altura da subárvore esquerda}$$

Árvore AVL

- introduzida por Adelson-Velsky and Evgenii Landis (1962)
- tenta diminuir a diferença entre as alturas das subárvores

Dado um nó, o **fator de balanceamento** é definido como:

$$\text{fb} = \text{altura da subárvore direita} - \text{altura da subárvore esquerda}$$

Definição

Uma árvore de busca é chamada Árvore AVL se:

- ela é vazia; ou
- o fator de balanceamento é -1, 0 ou 1.

Árvore AVL

- introduzida por Adelson-Velsky and Evgenii Landis (1962)
- tenta diminuir a diferença entre as alturas das subárvores

Dado um nó, o **fator de balanceamento** é definido como:

$$\text{fb} = \text{altura da subárvore direita} - \text{altura da subárvore esquerda}$$

Definição

Uma árvore de busca é chamada Árvore AVL se:

- ela é vazia; ou
- o fator de balanceamento é -1, 0 ou 1.

Note que fb é na verdade uma diferença!

Especificação do nó

Nó de árvore AVL

```
typedef struct NoArv{
    int chave;
    struct NoArv *esq, *dir;
```

Especificação do nó

Nó de árvore AVL

```
typedef struct NoArv{
    int chave;
    struct NoArv *esq, *dir;

    enum { MENOS, ZERO, MAIS } fb;
```

Especificação do nó

Nó de árvore AVL

```
typedef struct NoArv{
    int chave;
    struct NoArv *esq, *dir;

    enum { MENOS, ZERO, MAIS } fb;
} *NoArv;
```

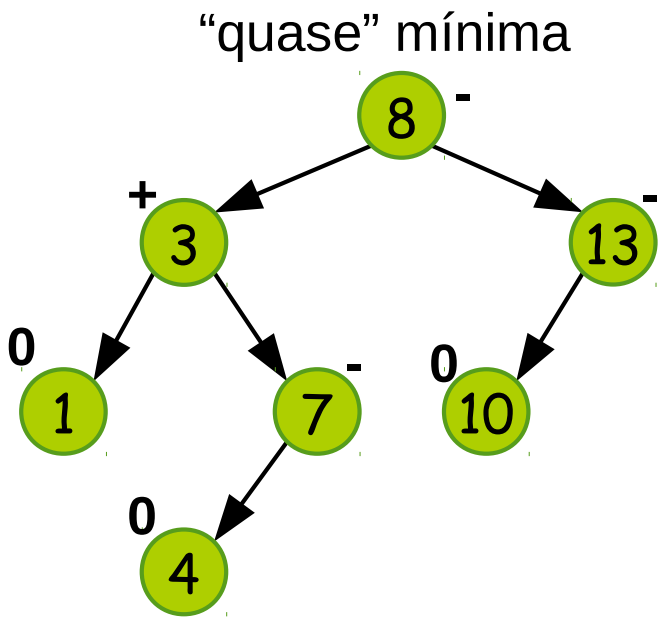
Especificação do nó

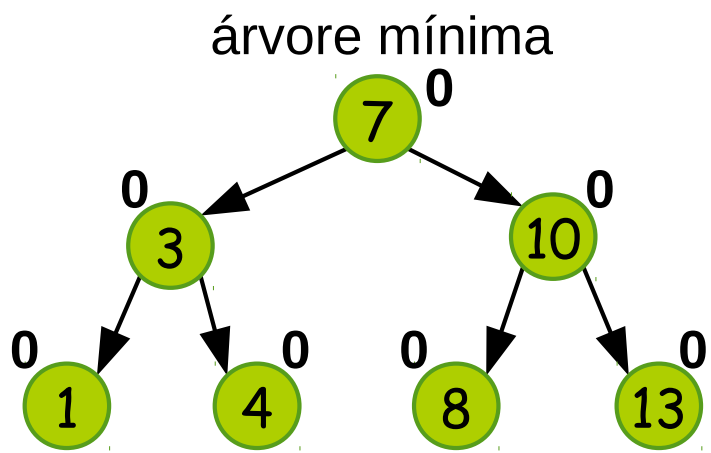
Nó de árvore AVL

```
typedef struct NoArv{
    int chave;
    struct NoArv *esq, *dir;

    enum { MENOS, ZERO, MAIS } fb;
} *NoArv;
```

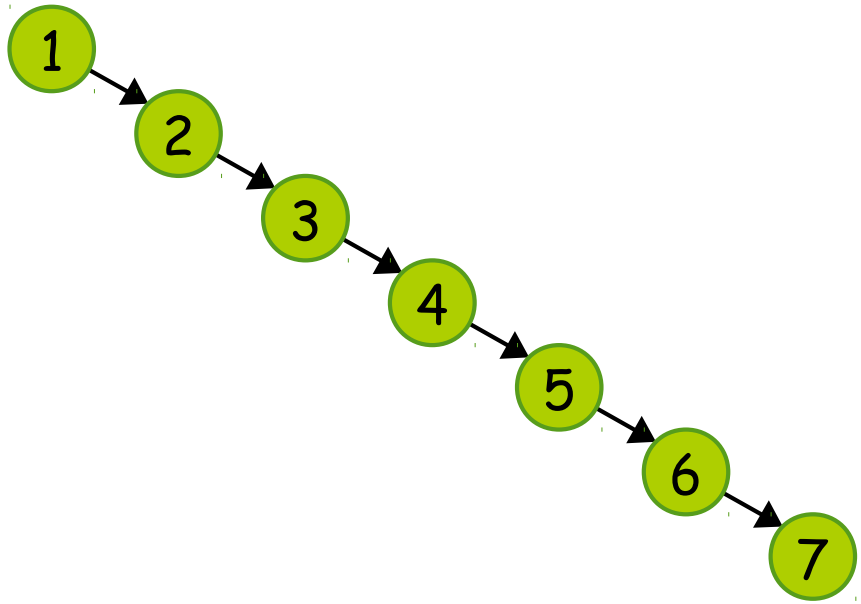
Em uma árvore AVL, fb só pode assumir 3 valores distintos!

Exmplos de árvores AVL



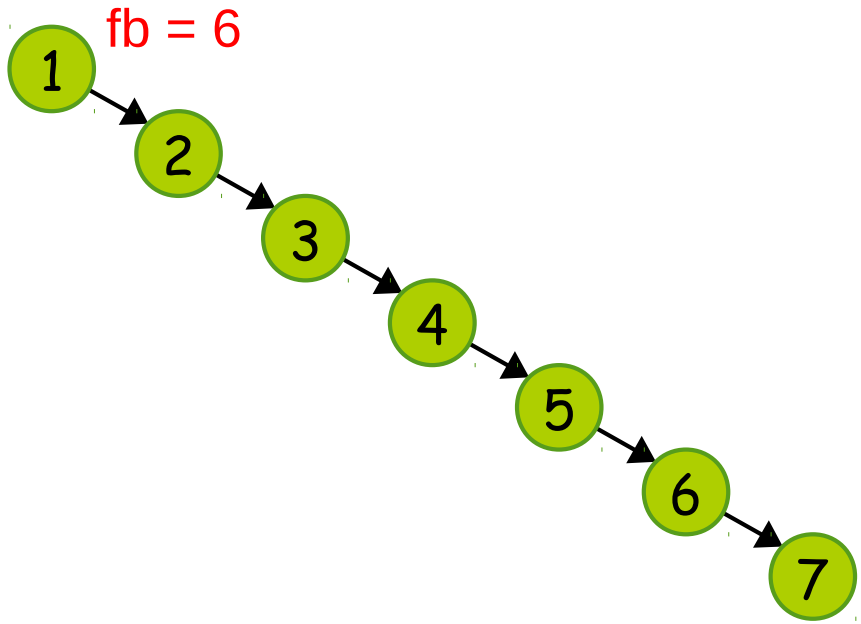


Preliminares



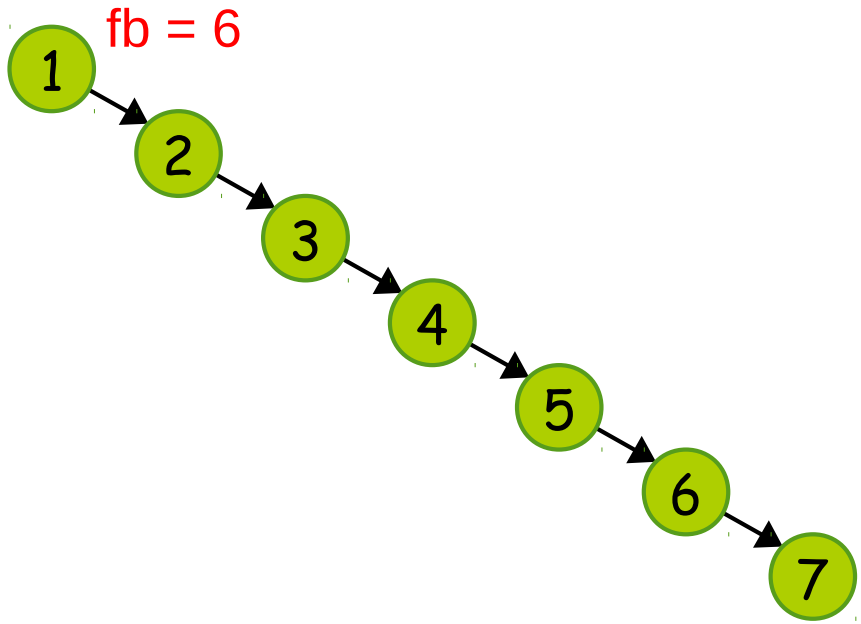
Qual o FB do nó raiz?

Preliminares



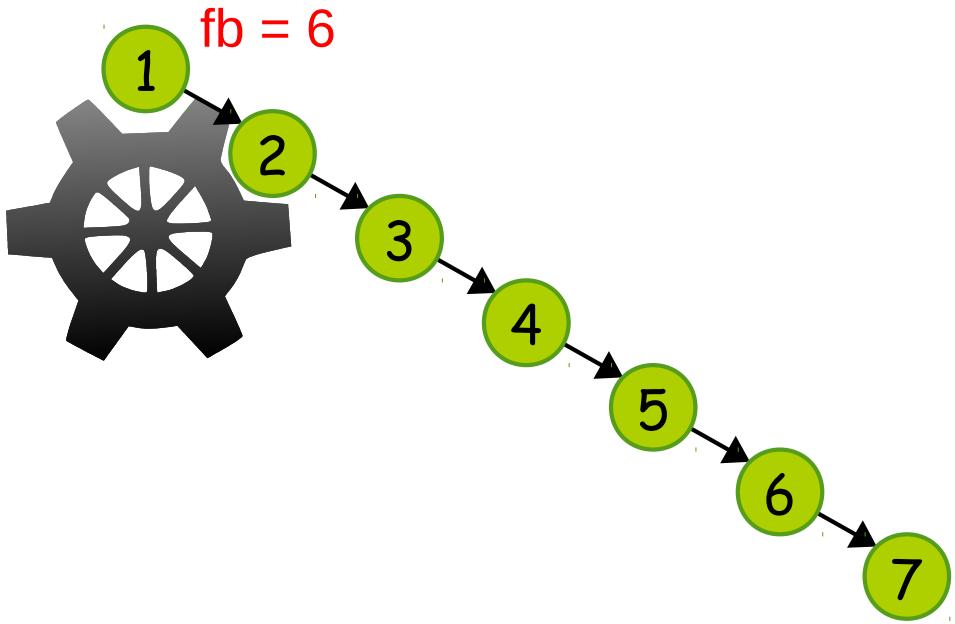
Qual o FB do nó raiz?

Preliminares



Qual o FB do nó raiz? Como diminuí-lo?

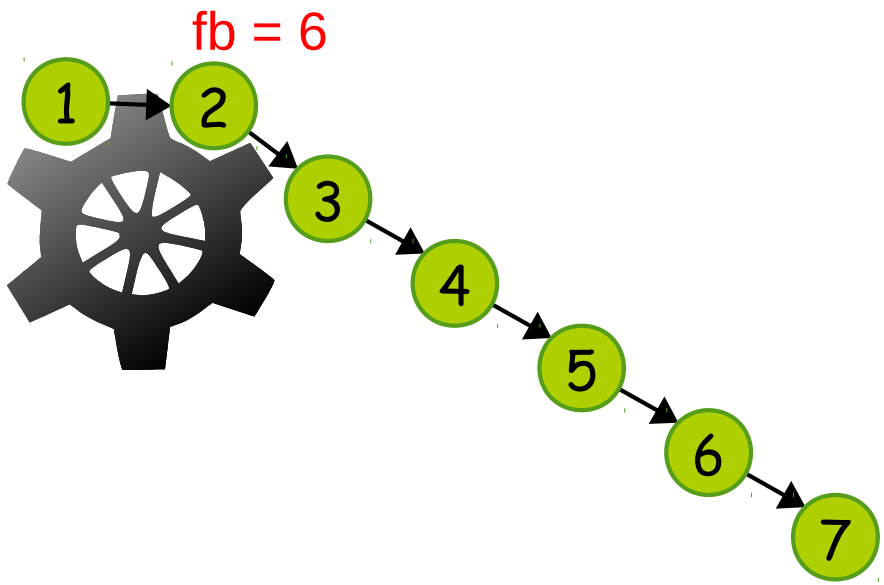
Preliminares



Qual o FB do nó raiz? Como diminuí-lo?

- Imaginamos um eixo

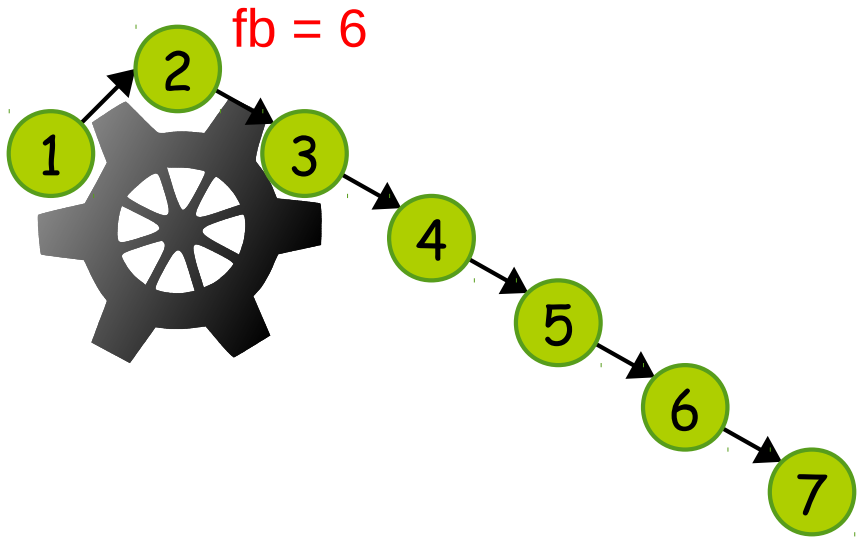
Preliminares



Qual o FB do nó raiz? Como diminuí-lo?

- Imaginamos um eixo
- "Rodamos" a árvore para a esquerda

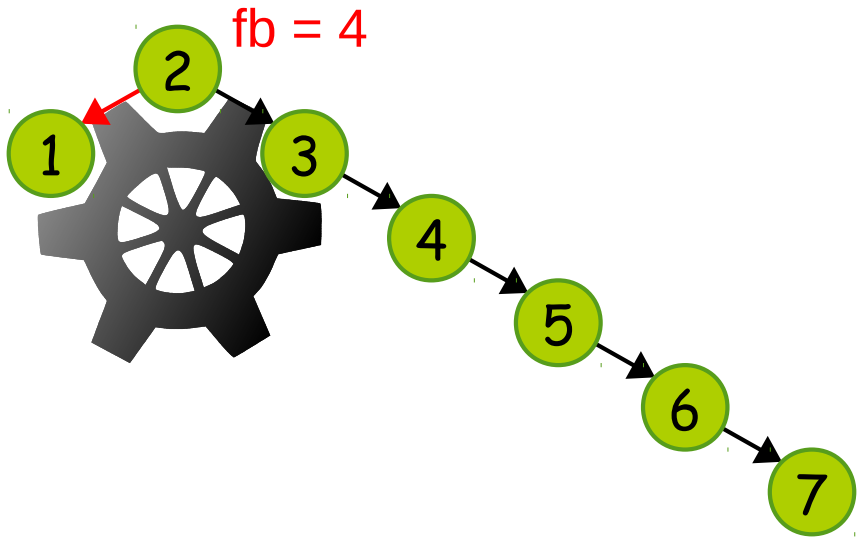
Preliminares



Qual o FB do nó raiz? Como diminuí-lo?

- Imaginamos um eixo
- "Rodamos" a árvore para a esquerda
- E alteramos o fb!

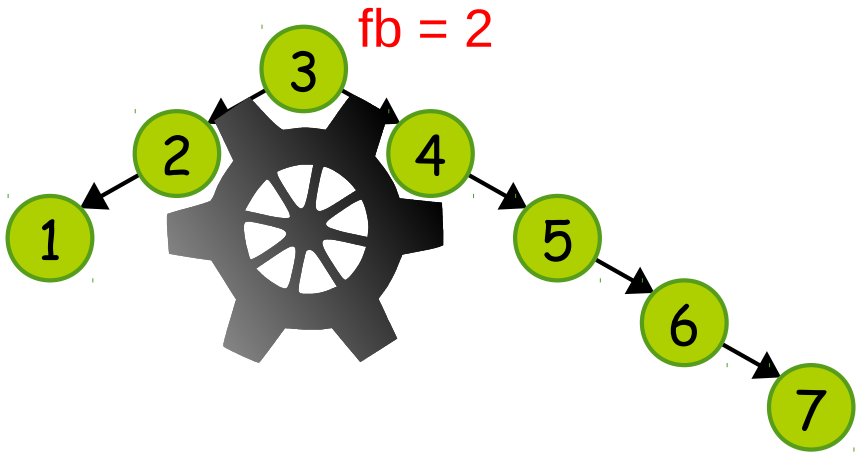
Preliminares



Qual o FB do nó raiz? Como diminuí-lo?

- Imaginamos um eixo
- "Rodamos" a árvore para a esquerda
- E alteramos o fb!

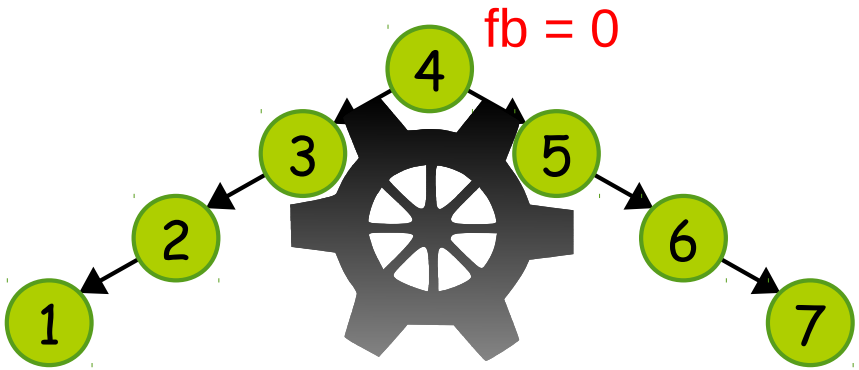
Preliminares



Qual o FB do nó raiz? Como diminuí-lo?

- Imaginamos um eixo
- "Rodamos" a árvore para a esquerda
- E alteramos o fb!

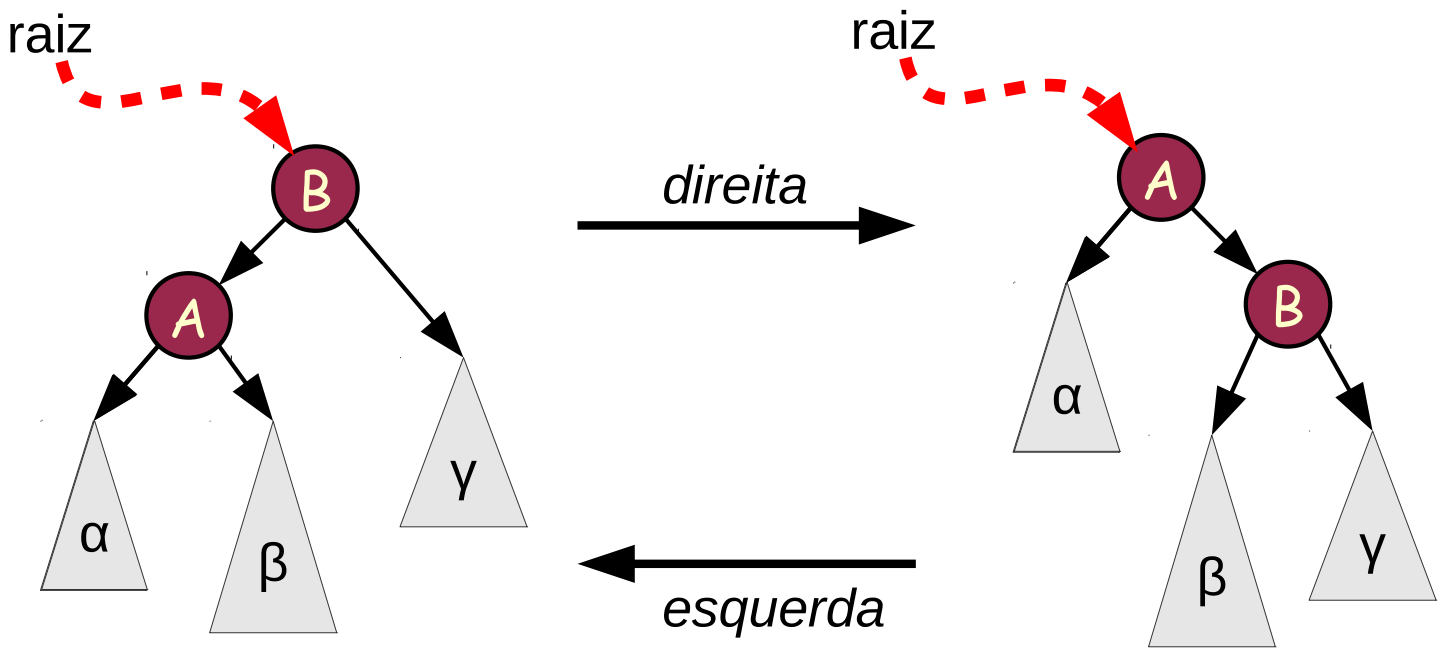
Preliminares



Qual o FB do nó raiz? Como diminuí-lo?

- Imaginamos um eixo
- "Rodamos" a árvore para a esquerda
- E alteramos o fb!

Rotações



Implementando rotação

Rotação para a direita

```
void rodar_direita(NoArv **raiz) {

    NoArv *a, *b;
```

Implementando rotação

Rotação para a direita

```
void rodar_direita(NoArv **raiz) {

    NoArv *a, *b;

    // obtém partes
    b = *raiz;
    a = b->esq;
```

Implementando rotação

Rotação para a direita

```
void rodar_direita(NoArv **raiz) {

    NoArv *a, *b;

    // obtém partes
    b = *raiz;
    a = b->esq;

    // troca ponteiros
    *raiz = a;
    b->esq = a->dir;
    a->dir = b;
}
```

Implementando rotação

Rotação para a direita

```
void rodar_direita(NoArv **raiz) {

    NoArv *a, *b;

    // obtém partes
    b = *raiz;
    a = b->esq;

    // troca ponteiros
    *raiz = a;
    b->esq = a->dir;
    a->dir = b;
}
```

Exercício: Implementar rotação para a esquerda.

Revendo inserção

Inserção recursiva

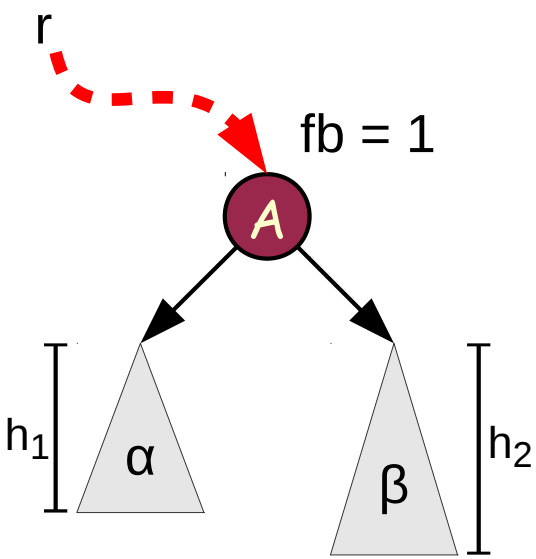
```
void inserir(NoArv **r, int x) {

    // caso base
    if (*r == NULL) {
        *r = malloc(sizeof(NoArv));
        (*r)->chave = x; (*r)->esq = (*r)->dir = NULL;
        return ;
    }

    // caso geral
    if ((*r)->chave < x)
        inserir(&((*r)->dir), x);
    else
        inserir(&((*r)->esq), x);
}
```

Corrigindo o balanceamento

Quando uma inserção altera fb?

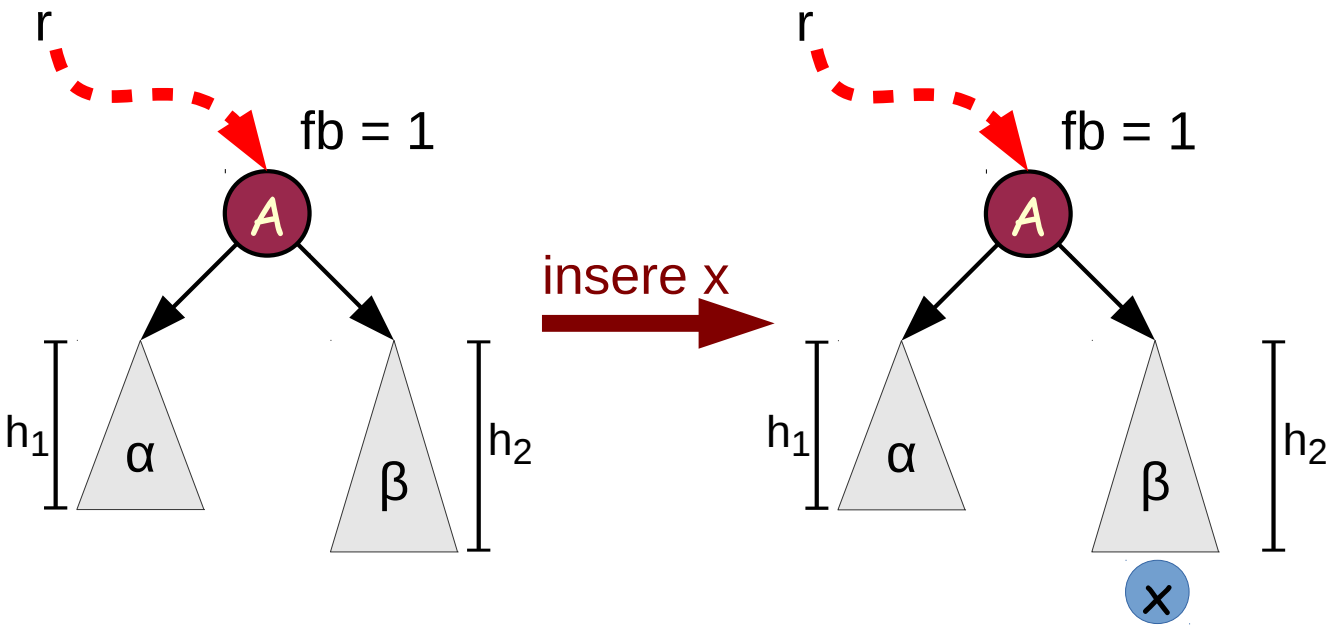
Corrigindo o balanceamento

Quando uma inserção altera fb?



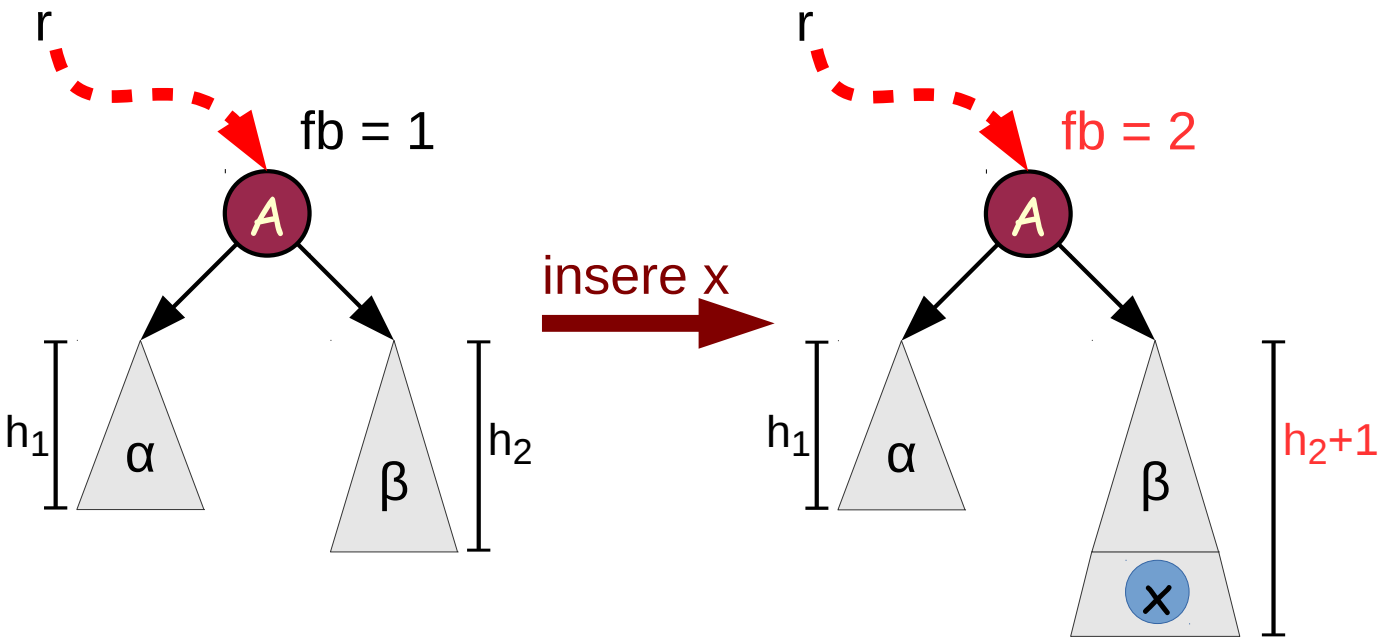
Corrigindo o balanceamento

Quando uma inserção altera fb?



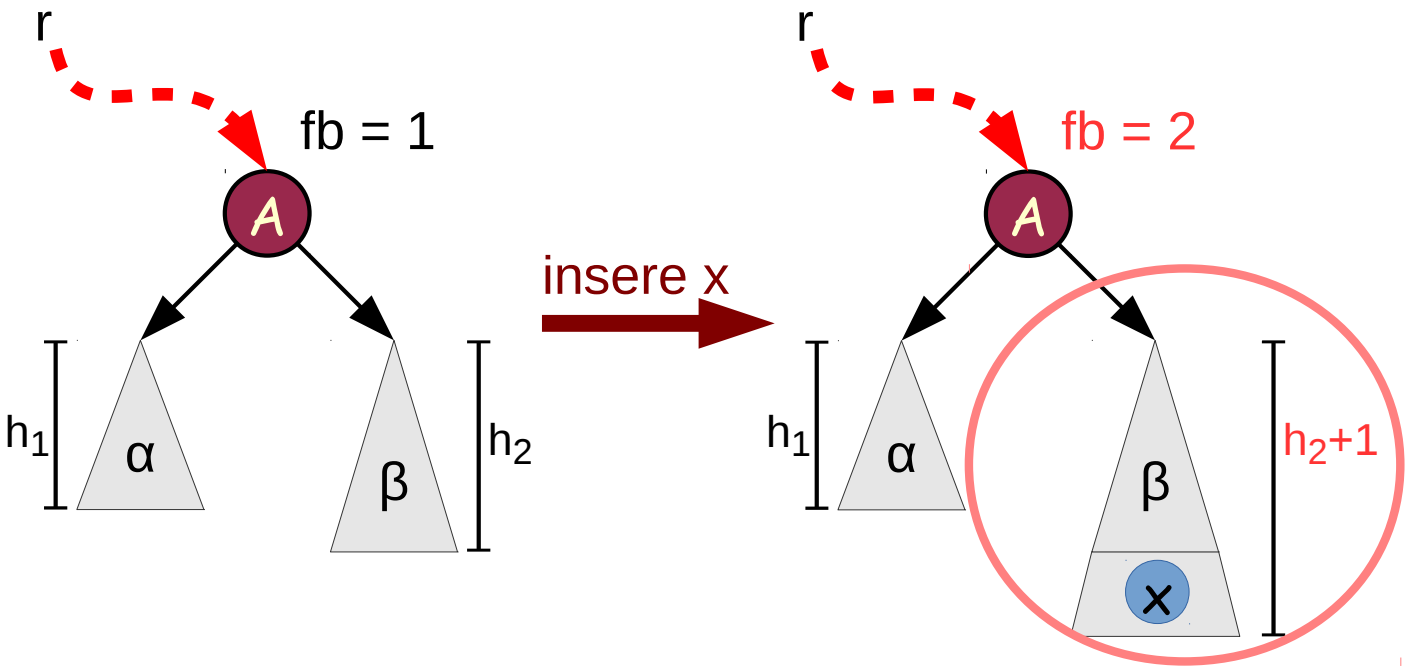
Corrigindo o balanceamento

Quando uma inserção altera fb?



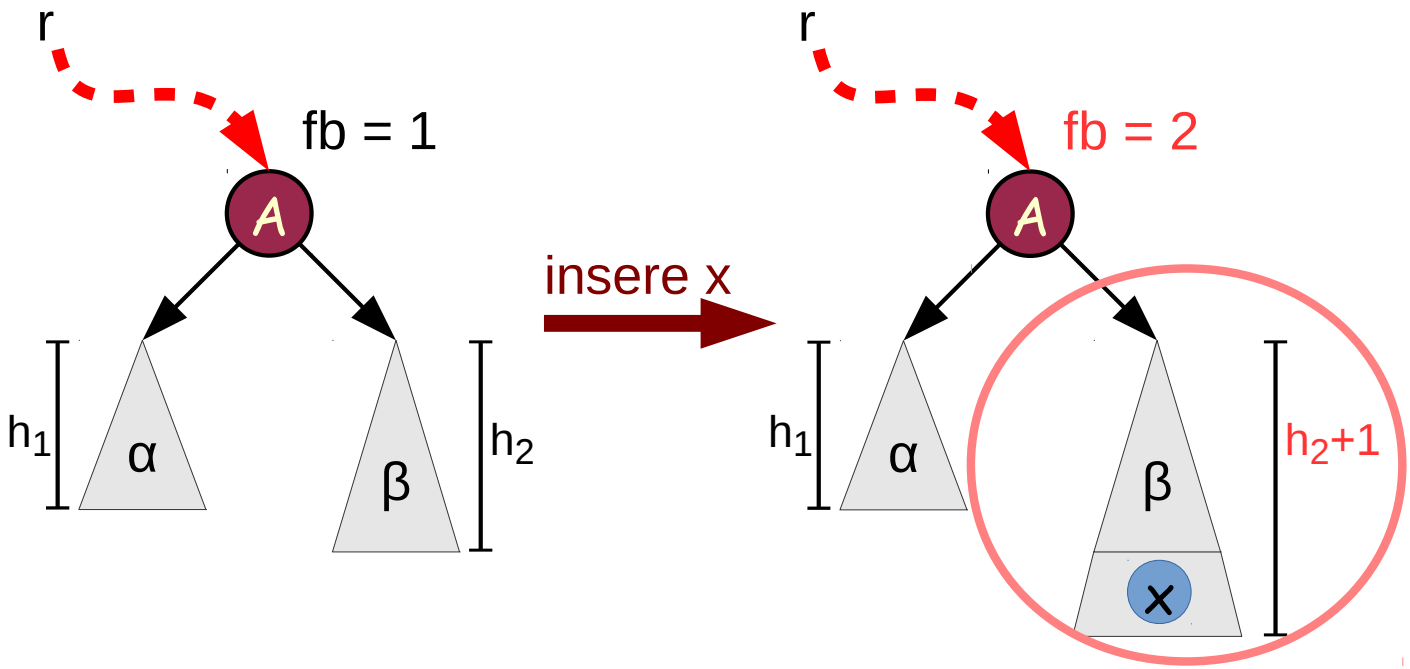
Corrigindo o balanceamento

Quando uma inserção altera fb? Quando a subárvore aumenta de tamanho!



Corrigindo o balanceamento

Quando uma inserção altera fb? Quando a subárvore aumenta de tamanho!

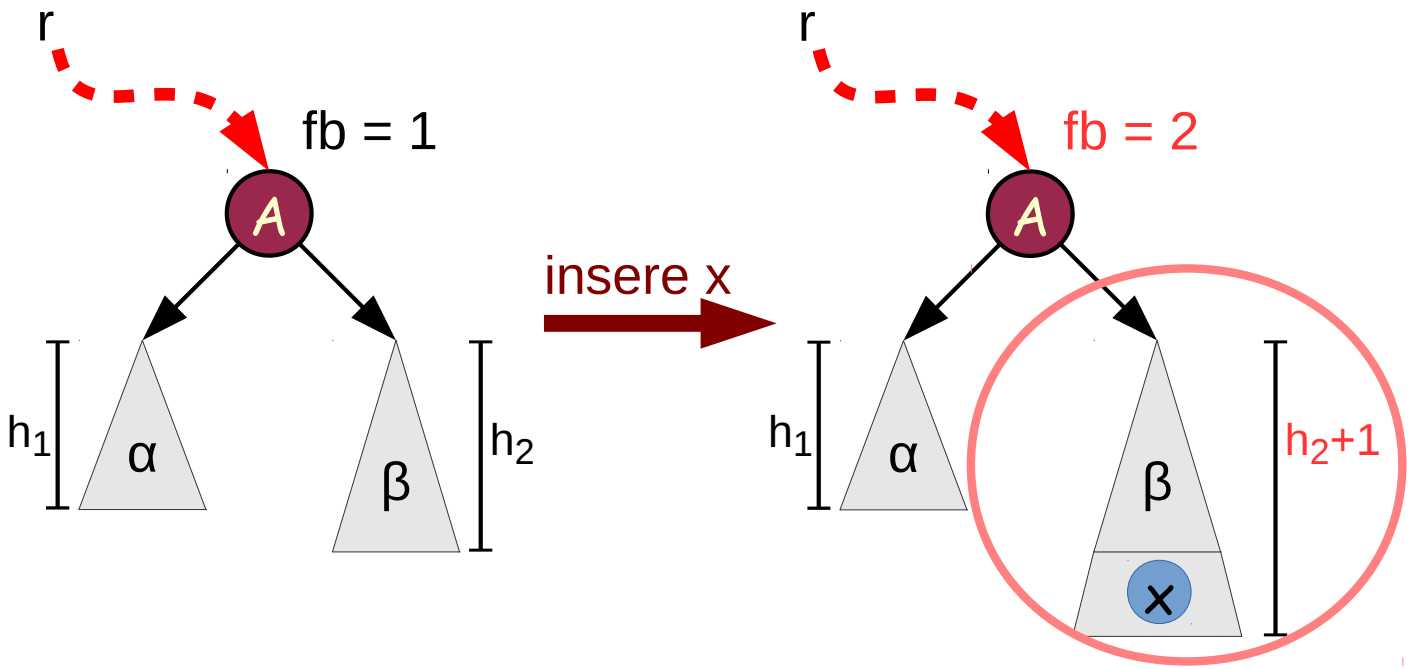


Estratégia

- redefinimos: `int inserir(NoArv **r, int x)`: devolve 1 se aumenta

Corrigindo o balanceamento

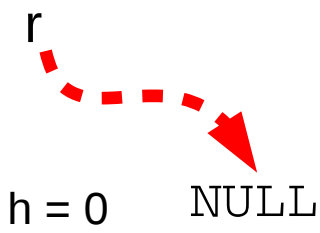
Quando uma inserção altera fb? Quando a subárvore aumenta de tamanho!



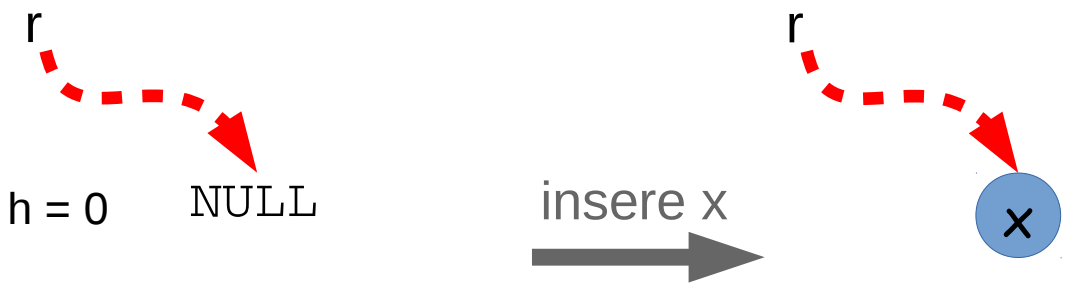
Estratégia

- redefinimos: `int inserir(NoArv **r, int x)`: devolve 1 se aumenta
- nesse caso, ajustamos a árvore usando rotações

Caso 1: árvore é vazia



Caso 1: árvore é vazia



- **Procedimento:** Inserimos o item

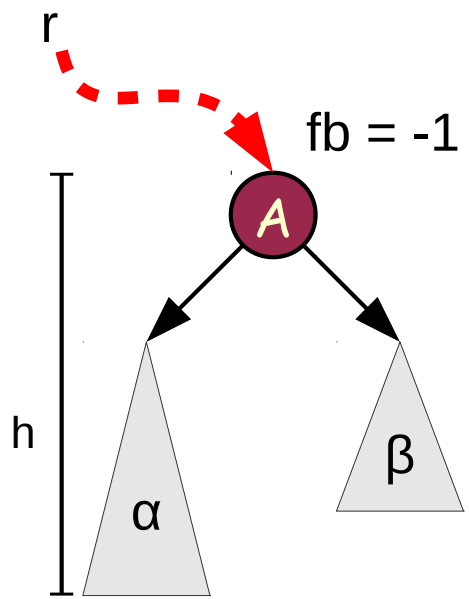
Caso 1: árvore é vazia

- **Procedimento:** Inserimos o item
- **Fator de balanceamento:** fb = 0

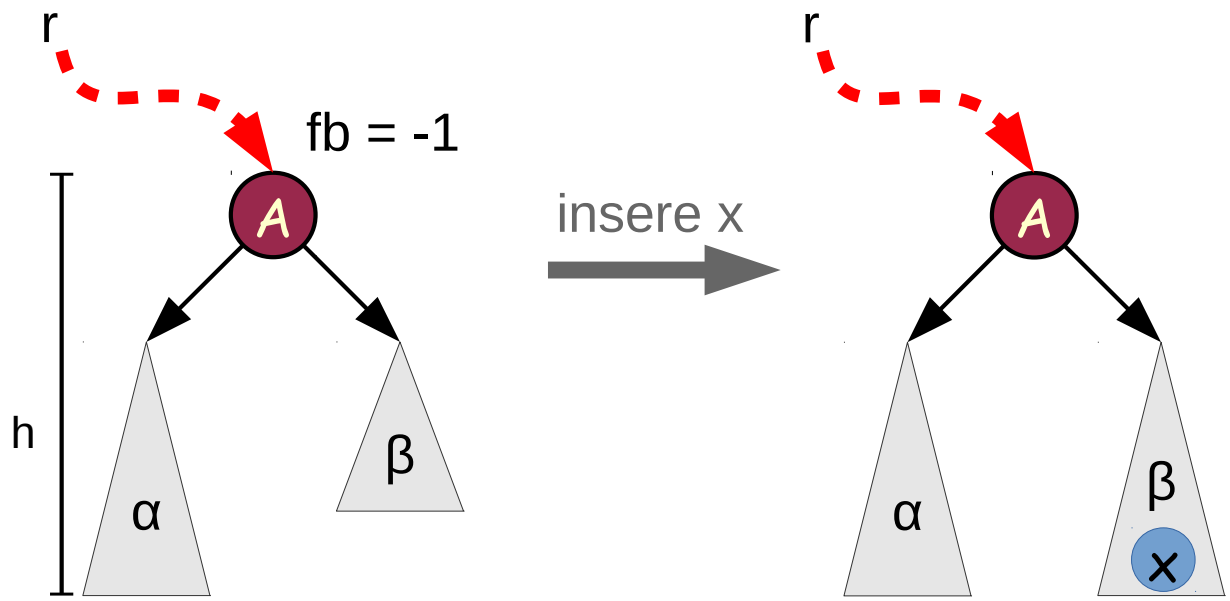
Caso 1: árvore é vazia

- **Procedimento:** Inserimos o item
- **Fator de balanceamento:** fb = 0
- **Altura:** aumentou

Caso 2: insere em subárvore menor

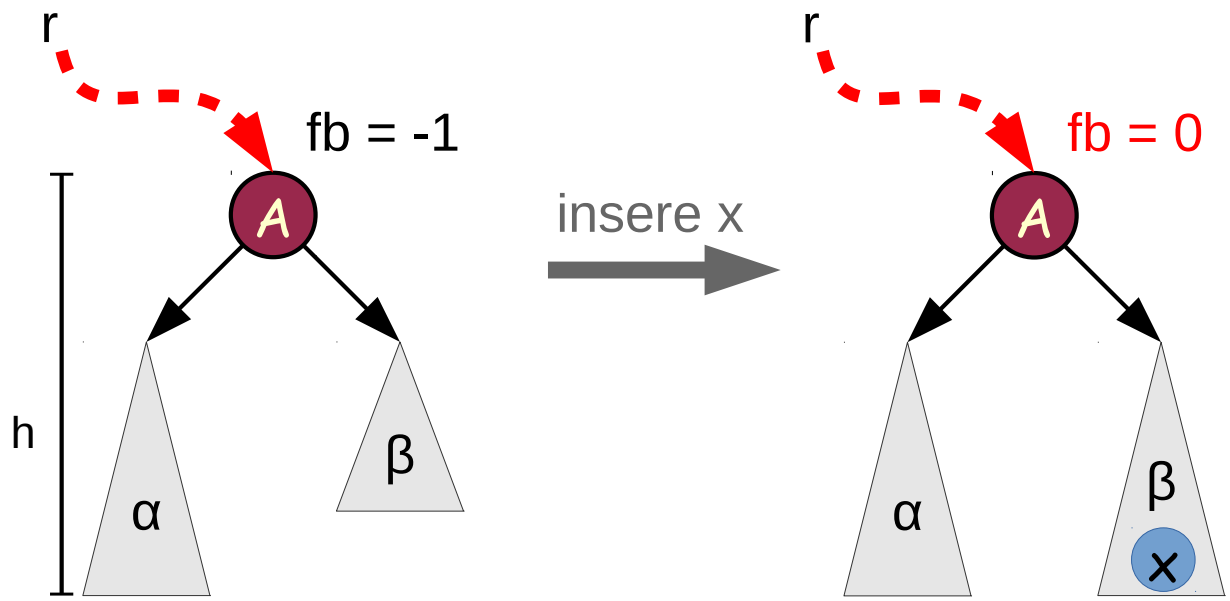


Caso 2: insere em subárvore menor



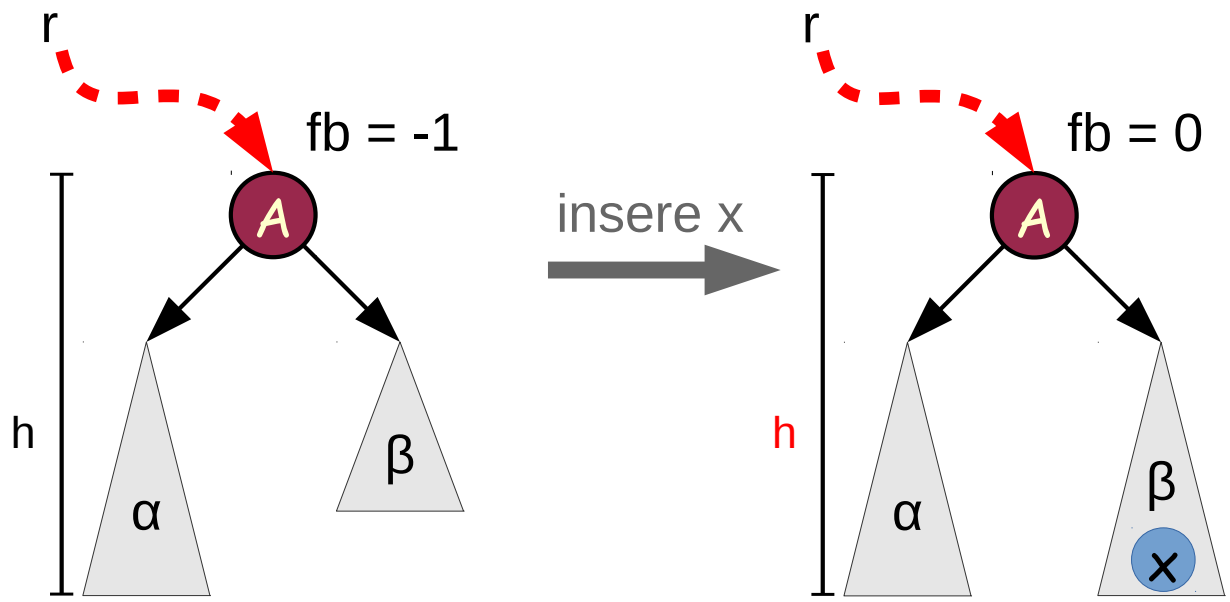
- **Procedimento:** Inserimos o item

Caso 2: insere em subárvore menor



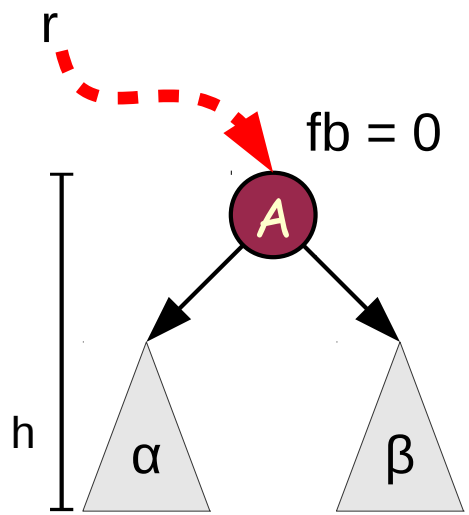
- **Procedimento:** Inserimos o item
- **Fator de balanceamento:** fb = 0

Caso 2: insere em subárvore menor

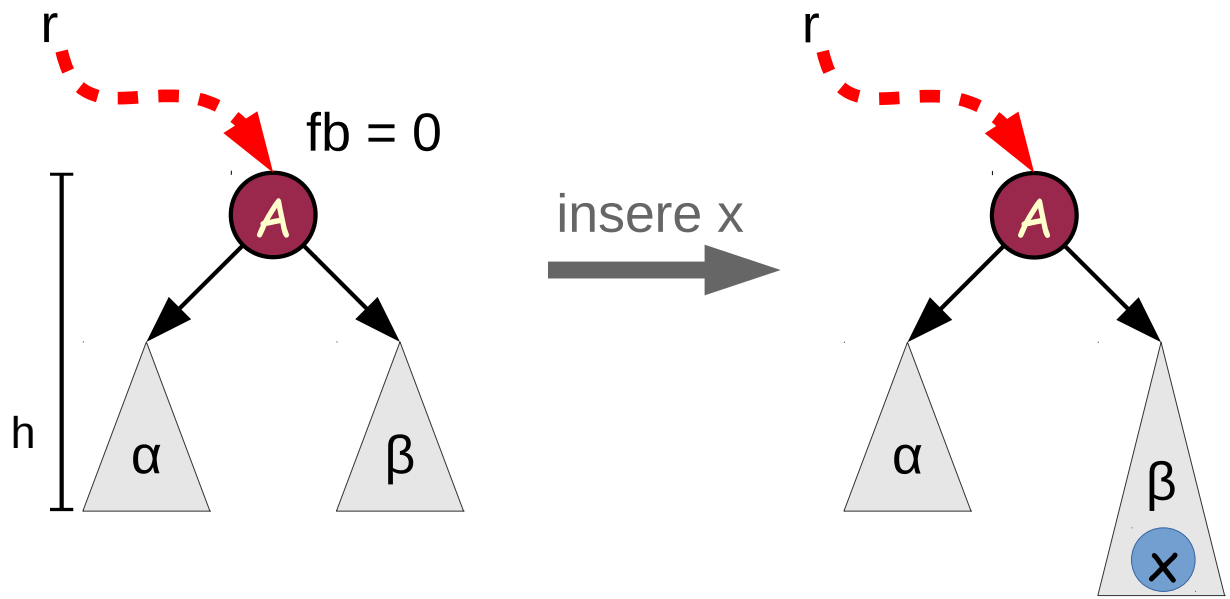


- **Procedimento:** Inserimos o item
- **Fator de balanceamento:** fb = 0
- **Altura:** não aumentou

Caso 3: subárvore da mesma altura

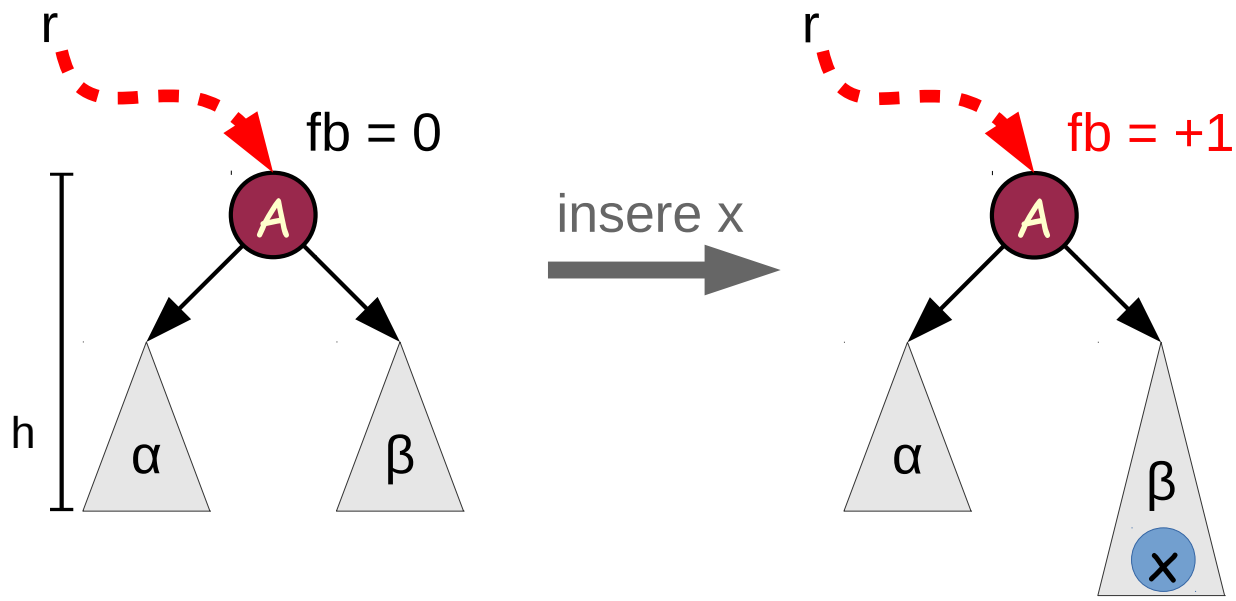


Caso 3: subárvore da mesma altura



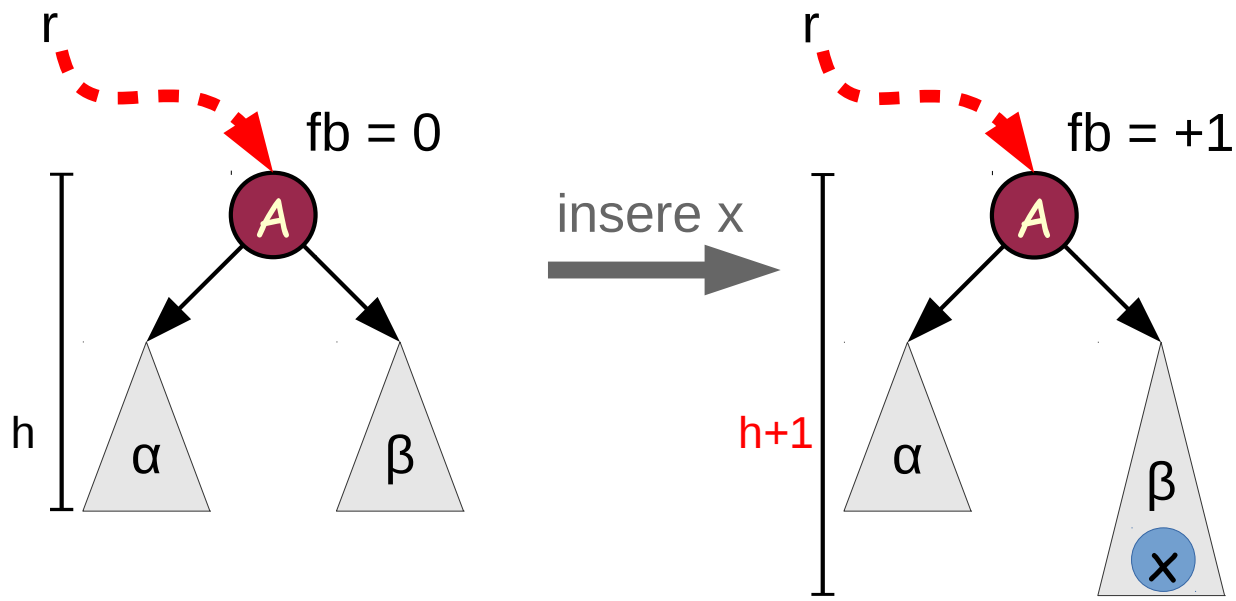
- **Procedimento:** Inserimos o item

Caso 3: subárvore da mesma altura



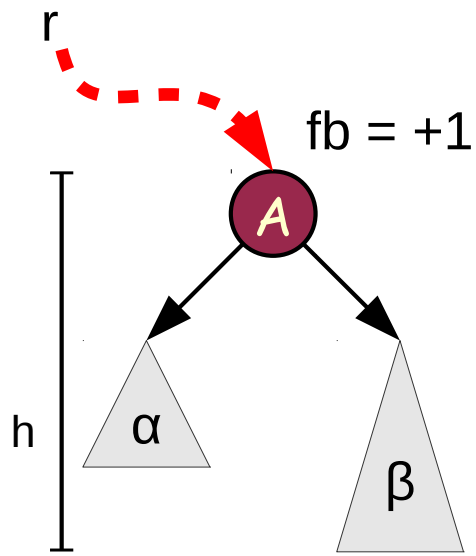
- **Procedimento:** Inserimos o item
- **Fator de balanceamento:** fb $= +1$

Caso 3: subárvore da mesma altura

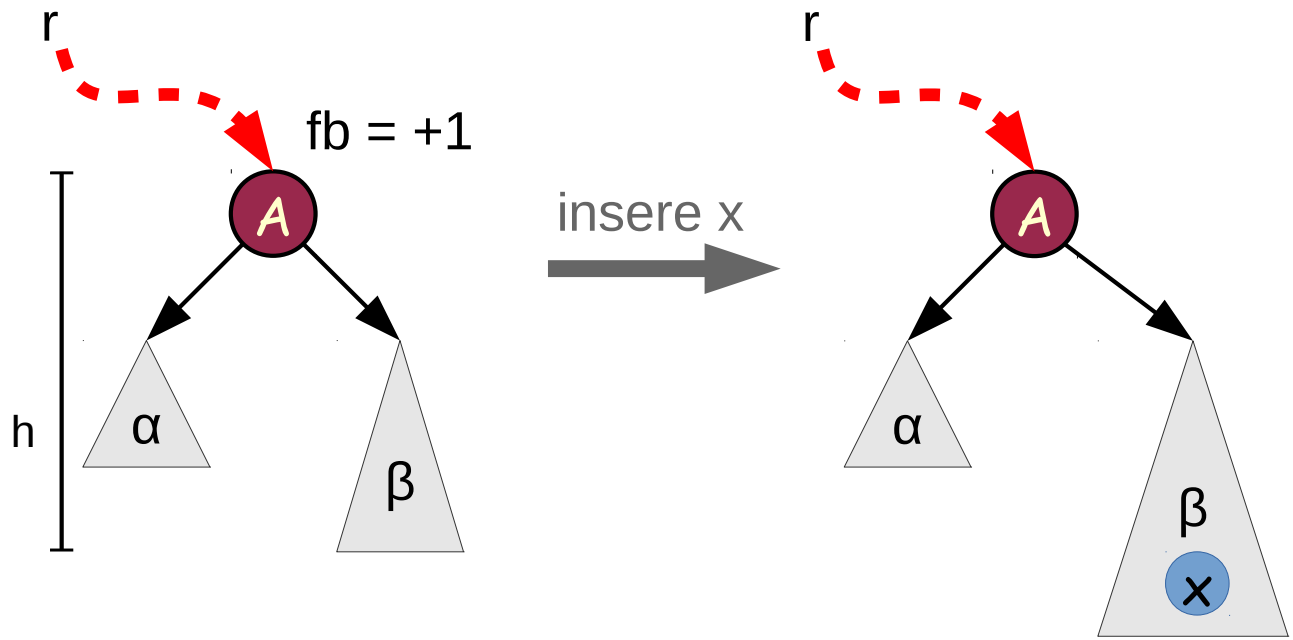


- **Procedimento:** Inserimos o item
- **Fator de balanceamento:** fb $= +1$
- **Altura:** aumentou

Caso 4: insere em subárvore maior

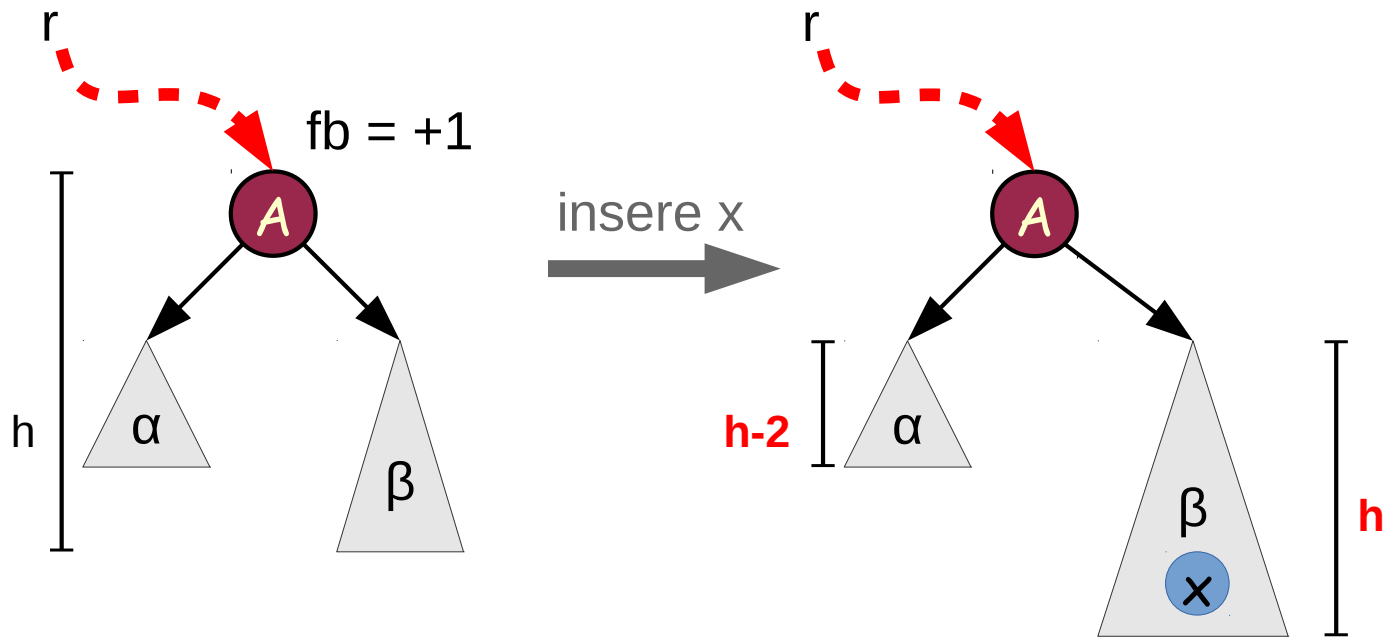


Caso 4: insere em subárvore maior



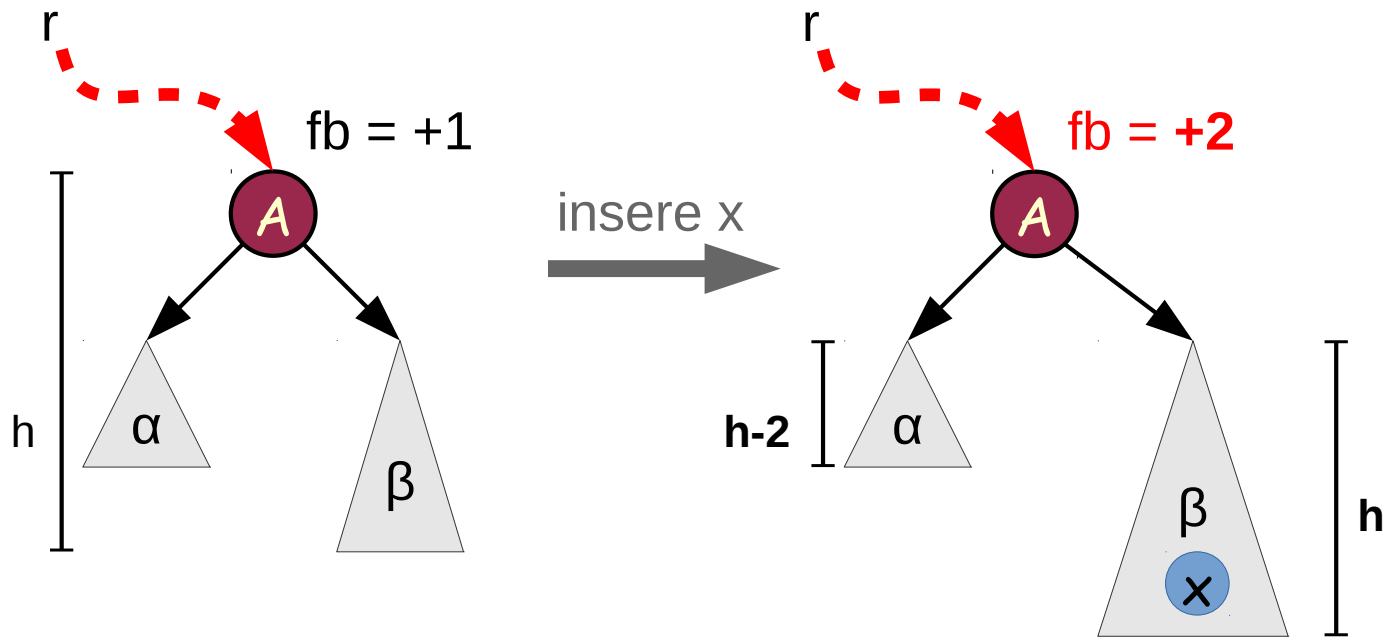
- **Procedimento:** Inserimos o item

Caso 4: insere em subárvore maior



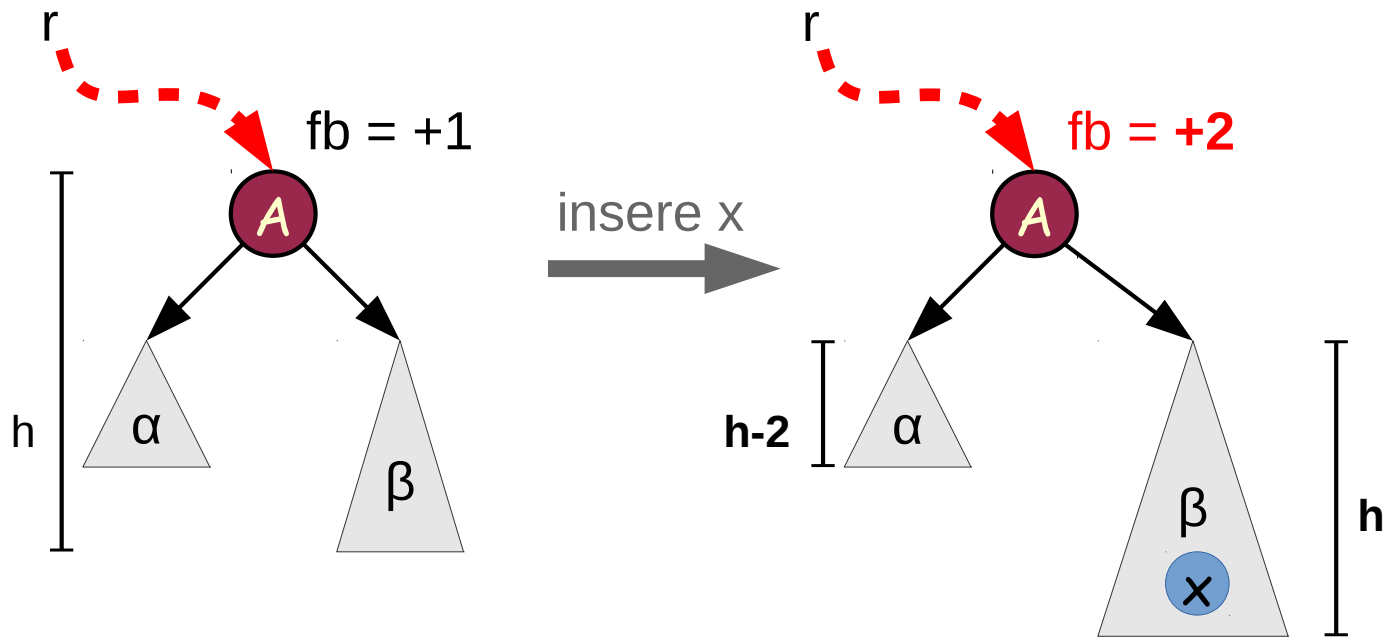
- **Procedimento:** Inserimos o item

Caso 4: insere em subárvore maior



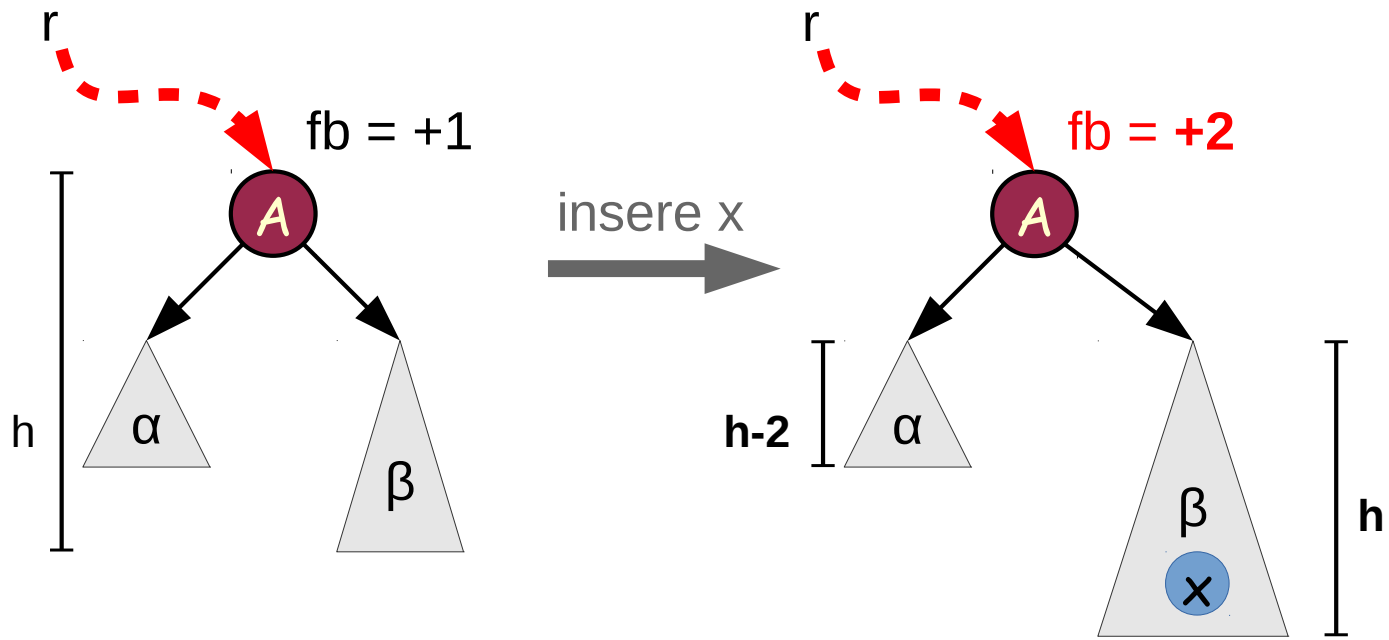
- **Procedimento:** Inserimos o item

Caso 4: insere em subárvore maior



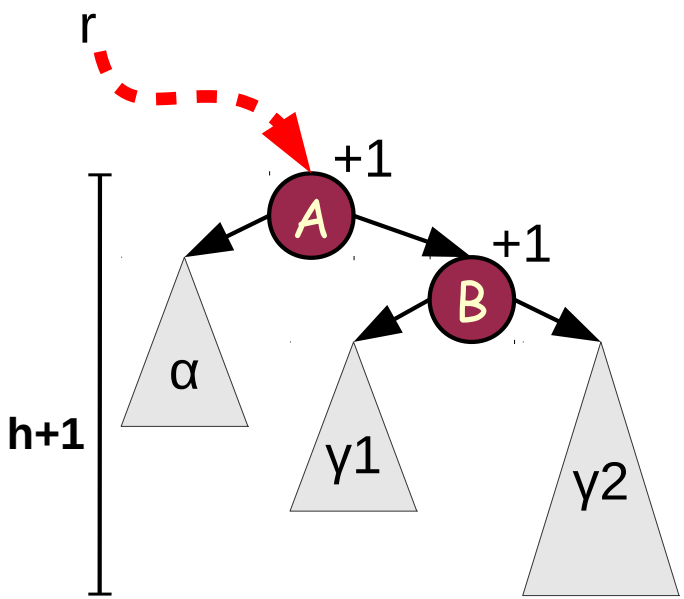
- **Procedimento:** Inserimos o item
- Árvore não é mais AVL:

Caso 4: insere em subárvore maior

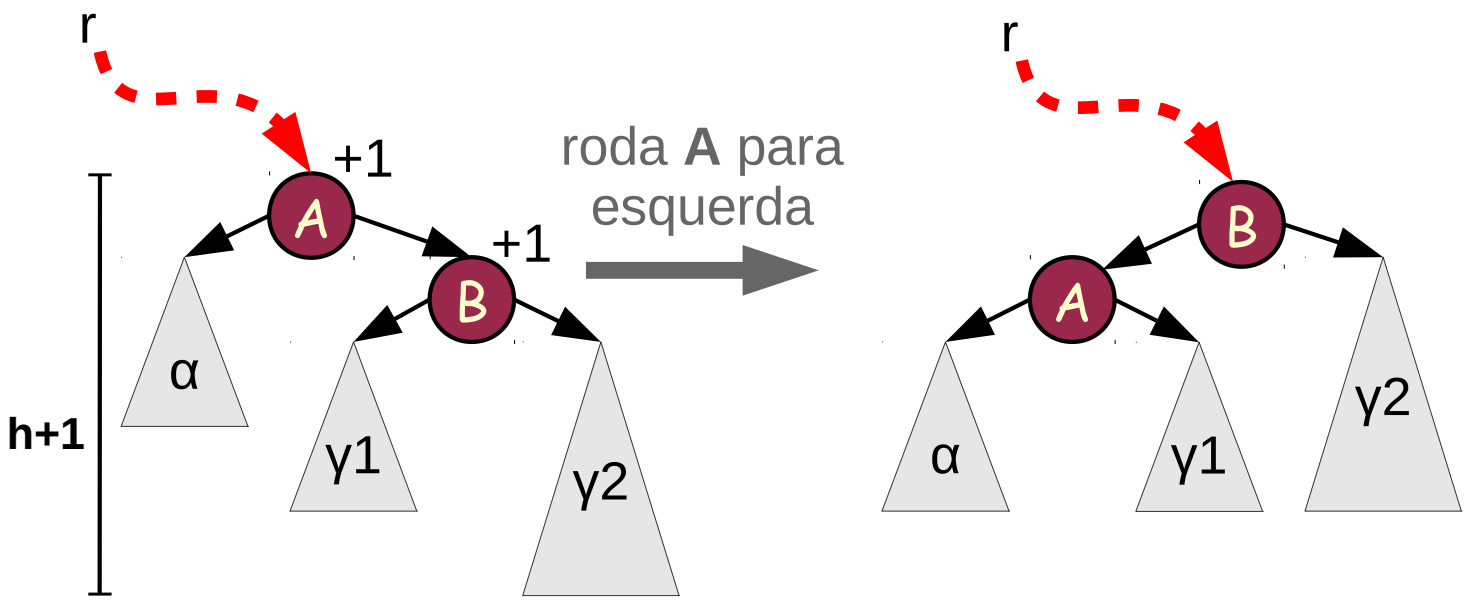


- **Procedimento:** Inserimos o item
- Árvore não é mais AVL:
  $\Rightarrow$ temos que ajustar a árvore!

Caso 4: i) subárvore desbalanceada positivamente

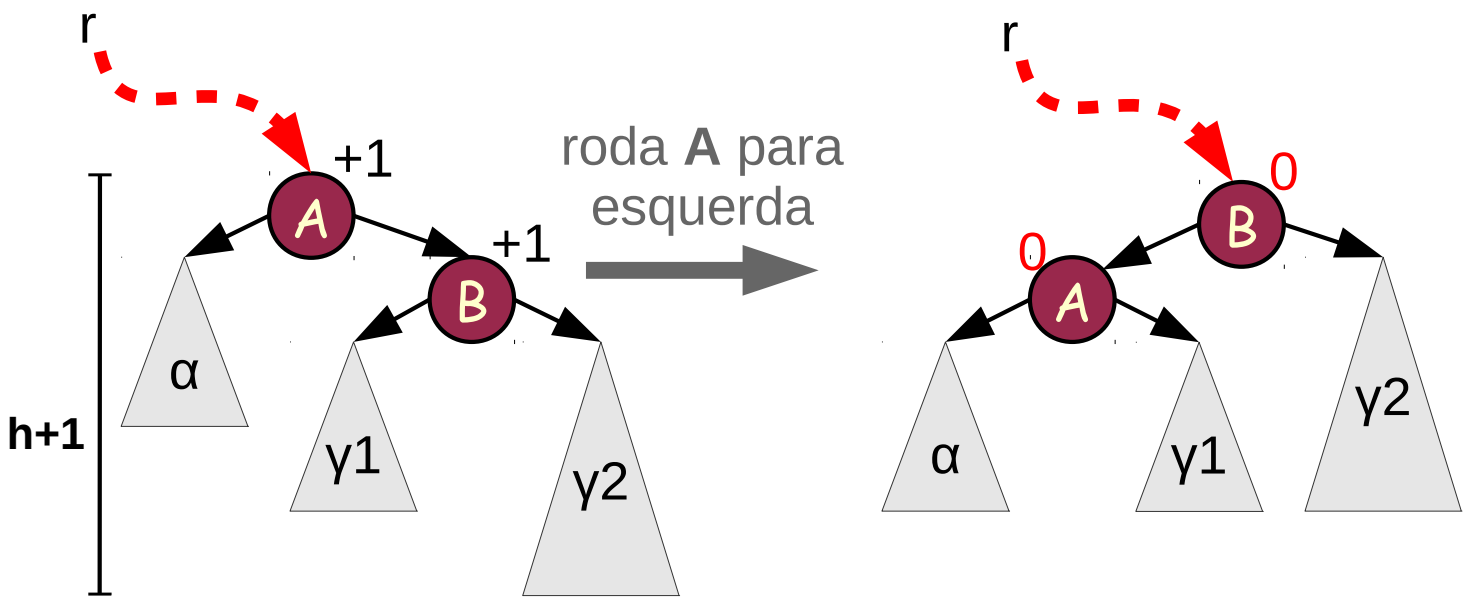


Caso 4: i) subárvore desbalanceada positivamente



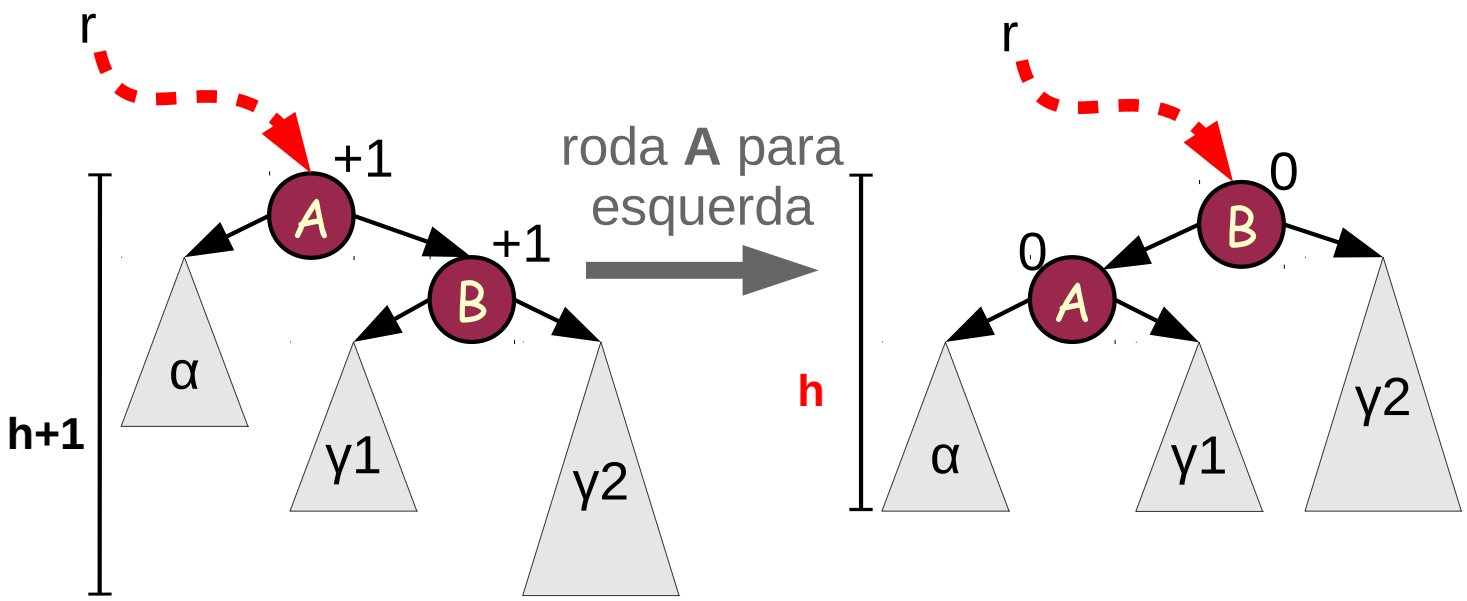
- **Procedimento:** rotação para esquerda

Caso 4: i) subárvore desbalanceada positivamente



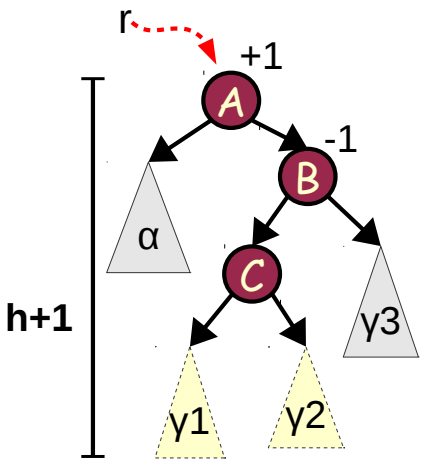
- **Procedimento:** rotação para esquerda
- **Fator de balanceamento:** fb = 0 (ajusta filho)

Caso 4: i) subárvore desbalanceada positivamente

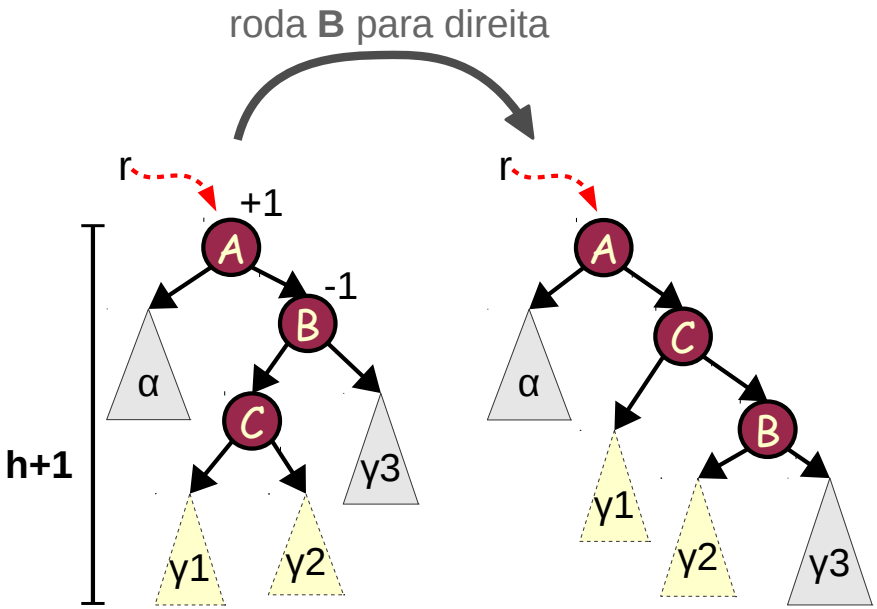


- **Procedimento:** rotação para esquerda
- **Fator de balanceamento:** fb = 0 (ajusta filho)
- **Altura:** não aumentou

Caso 4: ii) subárvore desbalanceada negativamente

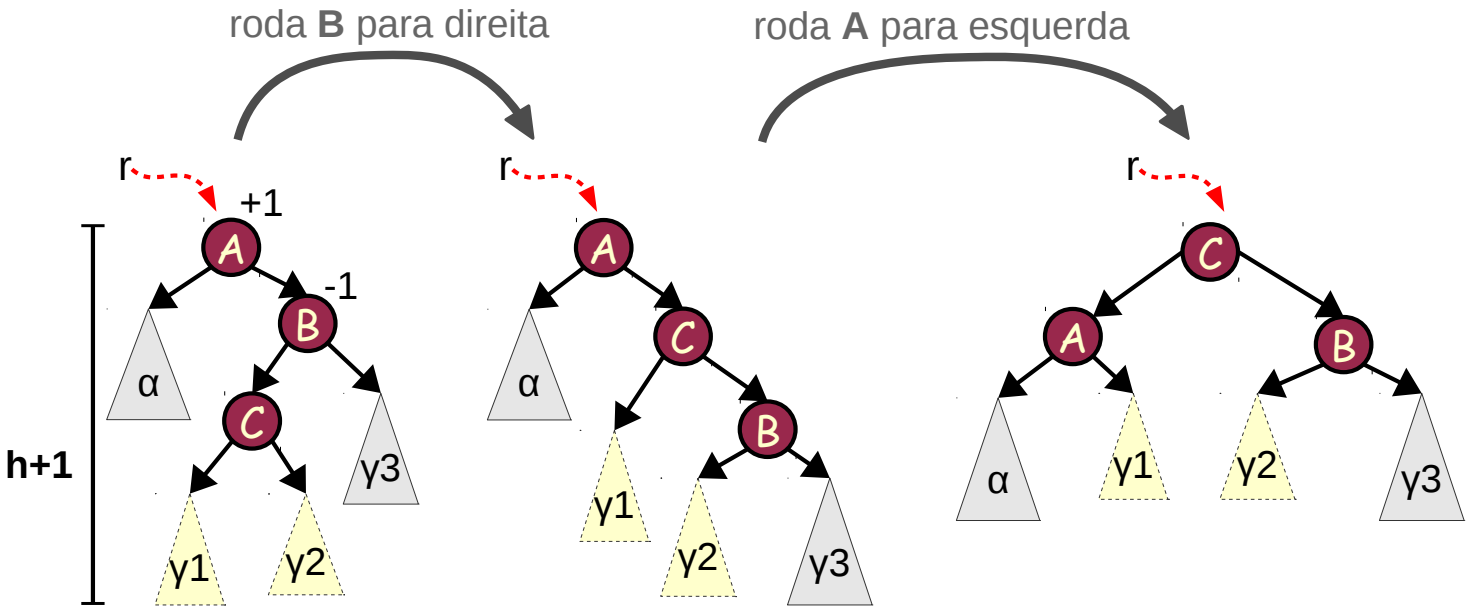


Caso 4: ii) subárvore desbalanceada negativamente



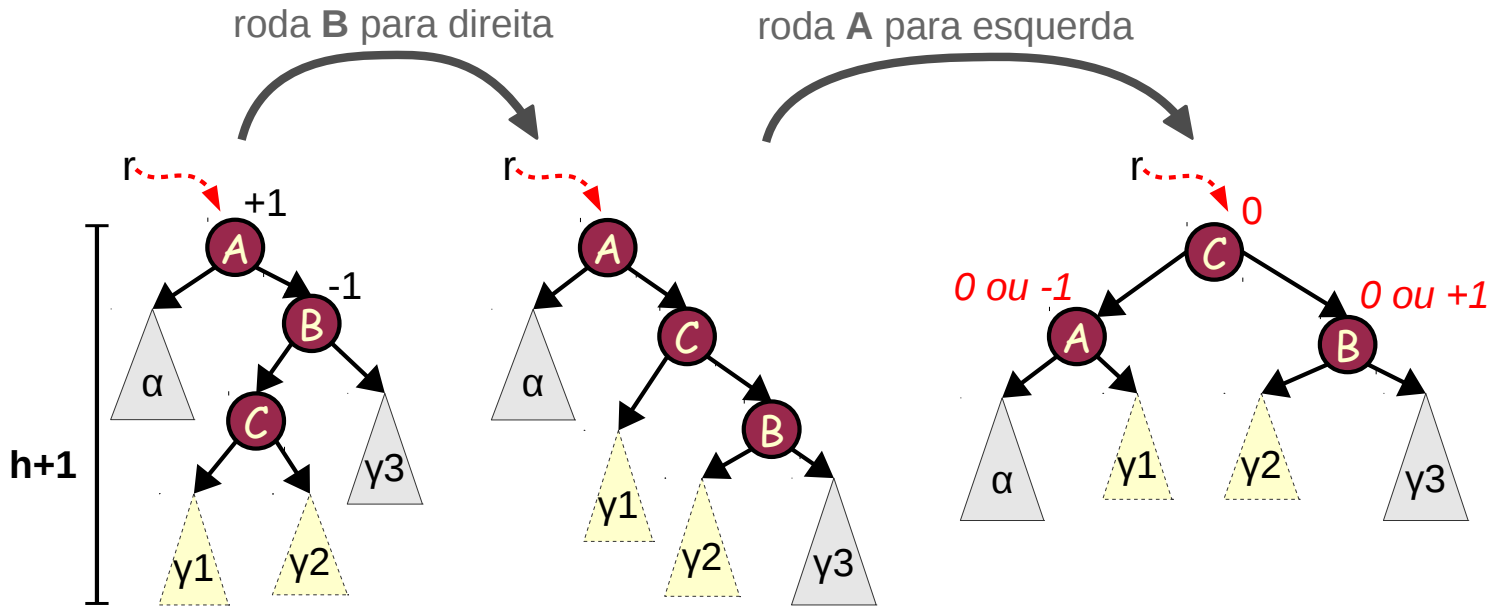
- **Procedimento:** rotação para direita

Caso 4: ii) subárvore desbalanceada negativamente



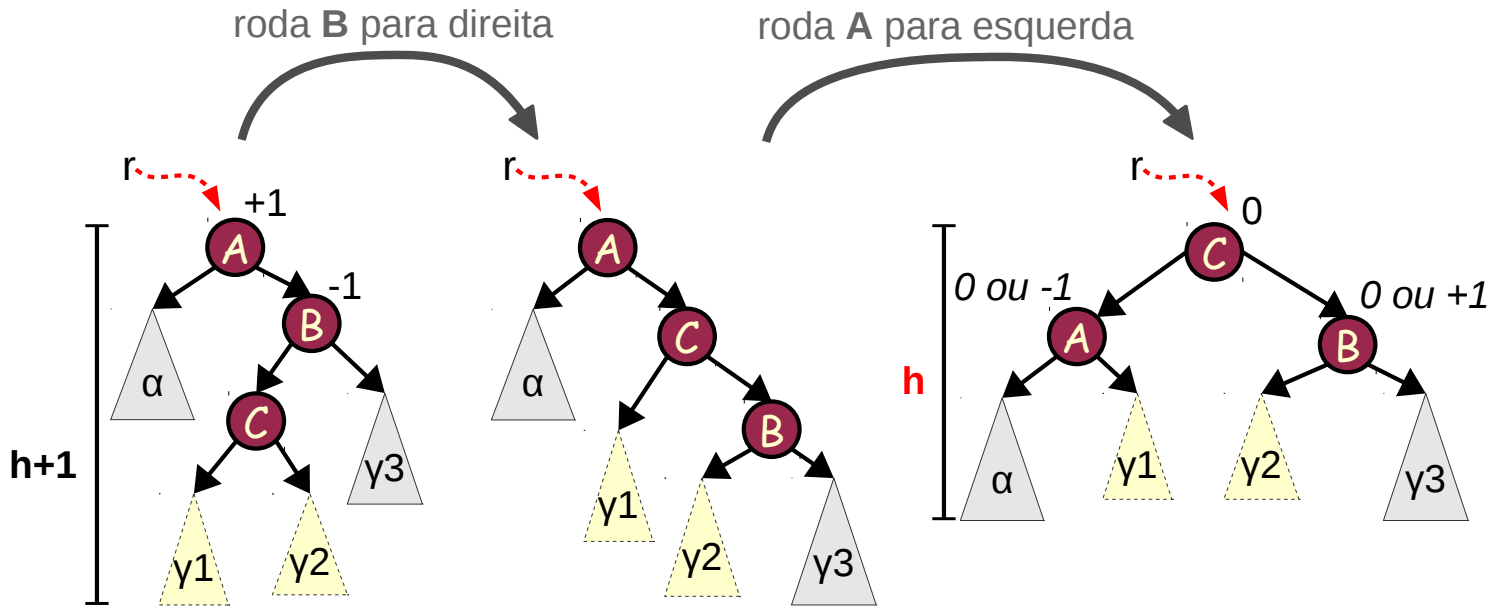
- **Procedimento:** rotação para direita e para esquerda

Caso 4: ii) subárvore desbalanceada negativamente



- **Procedimento:** rotação para direita e para esquerda
- **Fator de balanceamento:** fb = 0 (também ajusta filhos)

Caso 4: ii) subárvore desbalanceada negativamente



- **Procedimento:** rotação para direita e para esquerda
- **Fator de balanceamento:** fb = 0 (também ajusta filhos)
- **Altura:** não aumentou

Inserção em AVL

```
void inserir(NoArv **r, int x) {
    int aumentou;
    // árvore vazia
    if (*r == NULL) {
        *r = malloc(sizeof(NoArv));
        (*r)->chave = x;
        return 1; // aumentou
    }
```

Inserção em AVL

```
void inserir(NoArv **r, int x) {
    int aumentou;
    // árvore vazia
    if (*r == NULL) {
        *r = malloc(sizeof(NoArv));
        (*r)->chave = x;
        return 1; // aumentou
    }
    // insere na direita
    if ((*r)->chave < x) {
        aumentou = inserir(&((*r)->dir), x);
```

Inserção em AVL

```
void inserir(NoArv **r, int x) {
    int aumentou;
    // árvore vazia
    if (*r == NULL) {
        *r = malloc(sizeof(NoArv));
        (*r)->chave = x;
        return 1; // aumentou
    }
    // insere na direita
    if ((*r)->chave < x) {
        aumentou = inserir(&((*r)->dir), x);
        if (aumentou) {
            // analisa casos
            ...
        }
```

Inserção em AVL

```
void inserir(NoArv **r, int x) {
    int aumentou;
    // árvore vazia
    if (*r == NULL) {
        *r = malloc(sizeof(NoArv));
        (*r)->chave = x;
        return 1; // aumentou
    }
    // insere na direita
    if ((*r)->chave < x) {
        aumentou = inserir(&((*r)->dir), x);
        if (aumentou) {
            // analisa casos
            ...
        } else
            return 0;
```

Inserção em AVL

```
void inserir(NoArv **r, int x) {
    int aumentou;
    // árvore vazia
    if (*r == NULL) {
        *r = malloc(sizeof(NoArv));
        (*r)->chave = x;
        return 1; // aumentou
    }
    // insere na direita
    if ((*r)->chave < x) {
        aumentou = inserir(&((*r)->dir), x);
        if (aumentou) {
            // analisa casos
            ...
        } else
            return 0;
    // inserção na esquerda é simétrica
    } else {
        ...
    }
}
```

Exercício

1. Desenhe uma árvore AVL após a inserção de 9,1,5,6,7.
2. Pesquise sobre as árvores de Fibonacci e descubra qual é a maior altura de uma árvore AVL com n nós.
3. Remover um item de uma árvore AVL é tão (ou mais) problemático quanto inserir. Esboce um algoritmo para remoção de um nó de uma árvore AVL. Quais são todos os casos que você deve considerar? Tente fazer um desenho no papel de cada caso
4. Crie uma função que verifique se uma dada árvore binária arbitrária é uma árvore AVL.
5. (extra) Crie uma função que receba um vetor ordenado e cria uma árvore de altura mínima.