
Árvores AVL

Árvores Binárias de Busca

- Altura de uma árvore binária (AB): igual à profundidade, ou nível máximo, de suas folhas
- A eficiência da busca em árvore depende do seu balanceamento
- Algoritmos de inserção e remoção em ABB não garantem que a árvore gerada a cada passo seja balanceada
- Pergunta: Vale a pena balancear uma ABB de tempos em tempos?

Árvore Binária de Busca Aleatória

- Para uma ABB 'aleatória' (onde a probabilidade de inserção de um nó é igual para todos os lugares possíveis), foi mostrado que o número esperado de comparações para recuperar um registro qualquer é cerca de $1,39 \cdot \log_2(n)$.
 - ou seja, **39% pior** do que o custo do acesso em uma árvore perfeitamente balanceada
- Isto é: o balanceamento a cada operação aumenta o tempo e garante um desempenho melhor que numa árvore aleatória de, no máximo, 39% (o pior caso é muito raro!)
- Essa estratégia é aconselhável apenas se o número de buscas for muito maior do que o de inserções.
- A conservação do balanceamento pode ser mais simples se relaxarmos a condição de **perfeitamente balanceada** para **balanceada** apenas.

AB Balanceada *versus* AB Perfeitamente Balanceada

- Seja $h_b(n)$ a altura de uma AB balanceada e $h_{pb}(n)$ a altura de uma AB perfeitamente balanceada. Foi demonstrado que:

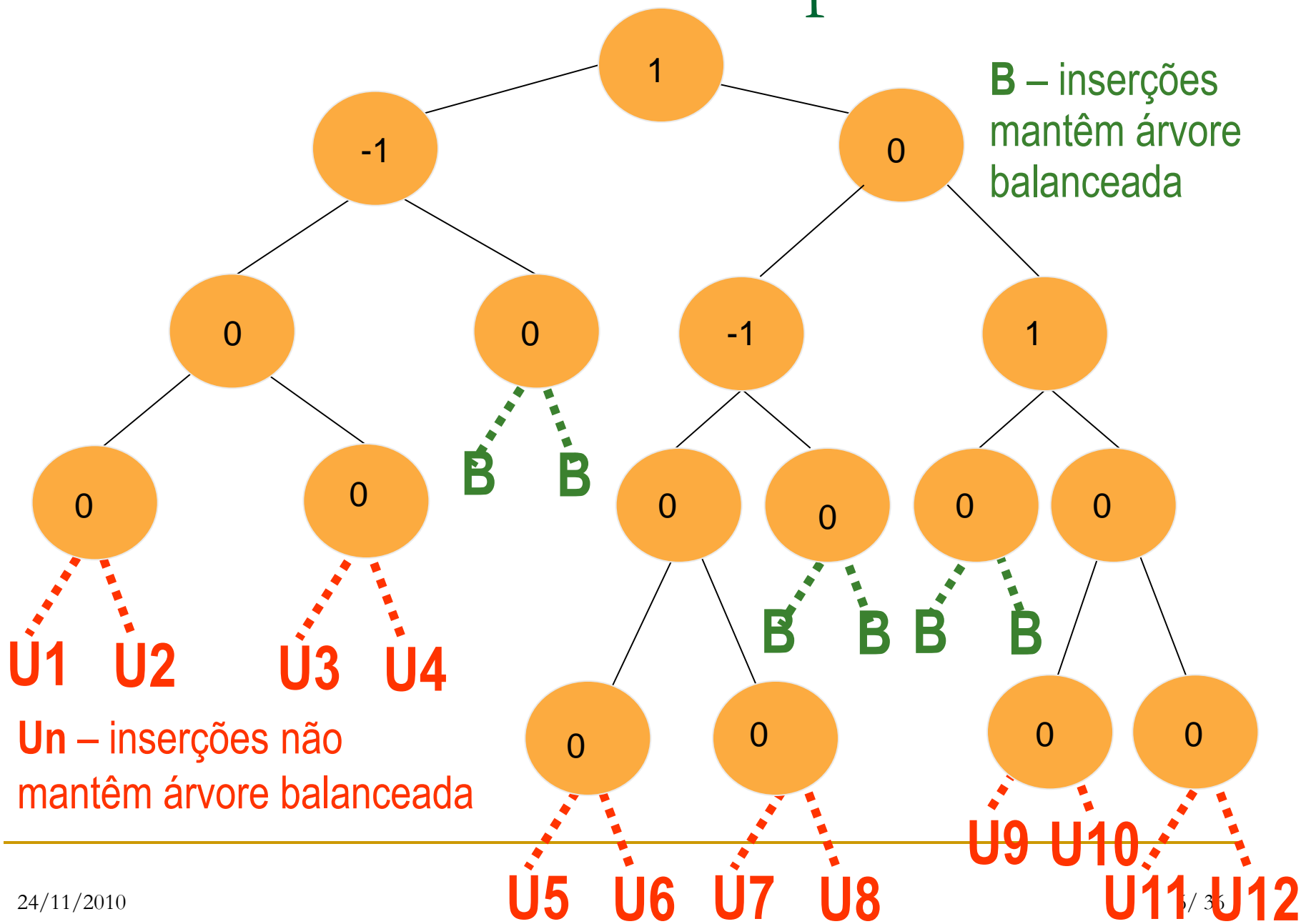
$$\begin{array}{c} \sim h_{pb}(n) \\ \swarrow \quad \searrow \\ \log_2(n+1) \leq h_b(n) \leq 1.4404 \log_2(n+2) - 0.328 \end{array}$$

- Ou seja: Uma AB Balanceada nunca terá altura superior a 45% da altura de sua correspondente Perfeitamente Balanceada.
- As operações numa AB Balanceada serão portanto da $O(\log_2 n)$.

Árvores AVL

- Árvore AVL: ABB na qual as alturas das duas sub-árvores de todo nó nunca diferem em mais de 1. Ou seja, é uma ABB Balanceada.
- Proposta em 1962 pelos matemáticos russos G.M. Adelson-Velski e E.M. Landis
- Seja o **Fator de Balanceamento de um Nó (FB)** a altura de sua sub-árvore direita menos a altura de sua sub-árvore esquerda
- Em uma árvore AVL todo nó tem FB igual a 1, -1 ou 0

Árvores AVL – Exemplo




Árvores AVL

 Desbalanceamento:

Se NÓ inserido é descendente esquerdo de um nó que tinha $FB = -1$ (U1 a U8)

OU

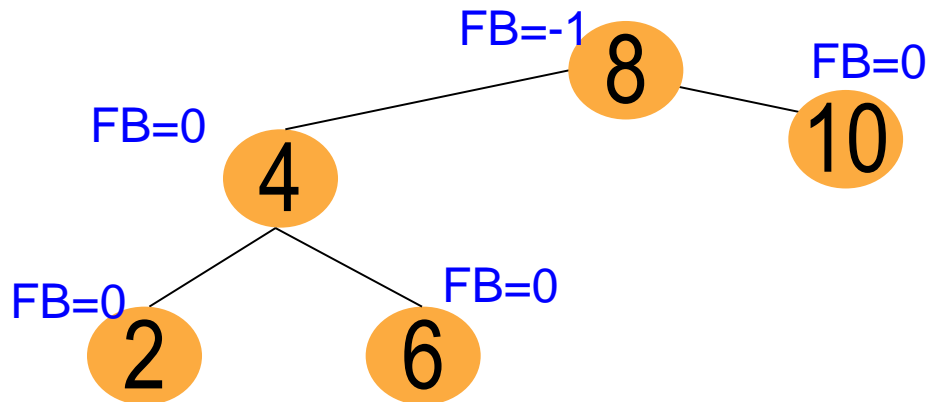
 Se NÓ inserido é descendente direito de um nó que tinha $FB = 1$ (U9 a U12)

Árvores AVL

- Para manter uma árvore balanceada é necessário aplicar uma transformação na árvore tal que:
 1. o percurso in-ordem na árvore transformada seja igual ao da árvore original (isto é, a árvore transformada continua sendo uma ABB);
 2. a árvore transformada fique balanceada (todos os nós com $FB = -1, 0$ ou 1).

Árvore AVL

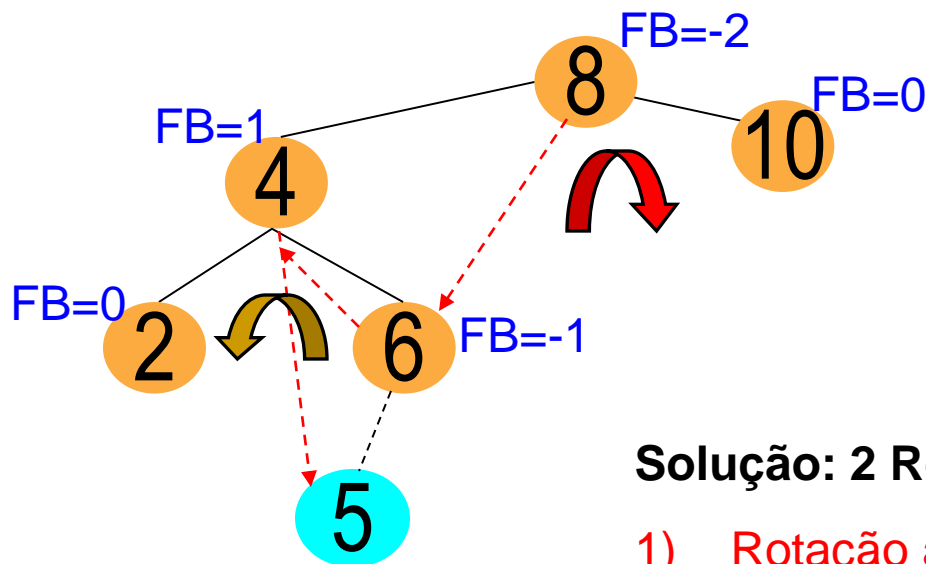
- Numa AVL, a inserção é feita na posição adequada, e depois verifica-se se houve desbalanceamento.
- Veja o que pode acontecer a cada inserção:



- As chaves 9 e 11 não violam o balanceamento (até melhoram!).
- As chaves 1, 3, 5 ou 7 violam.

AVL: 2 Casos de Rebalanceamento

- **TIPO 1:** a raiz de uma sub-árvore tem $FB=2$ (ou -2) e tem um filho com $FB=-1$ (ou 1), i.e. FB com sinal oposto ao do pai.



Inserção de 5

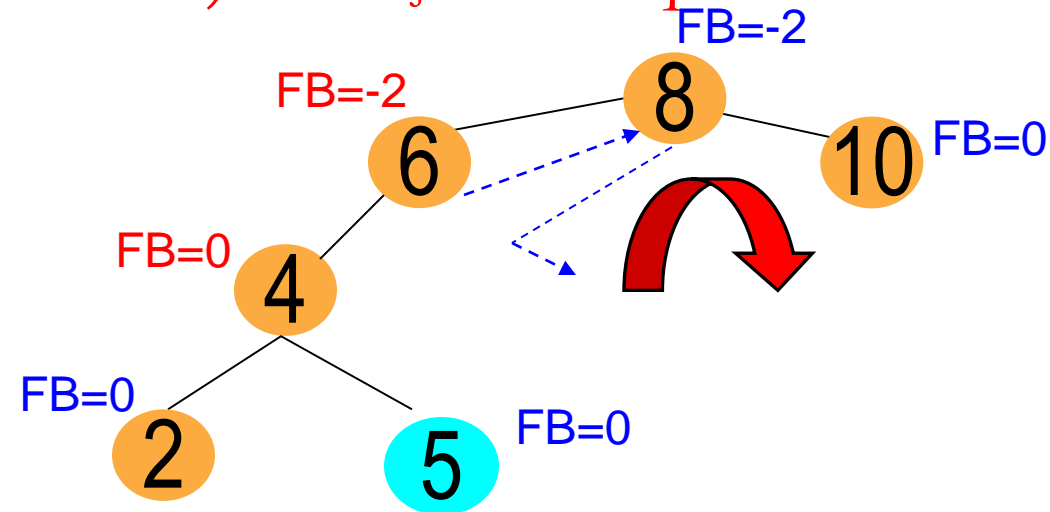
Como rebalancear?

Solução: 2 Rotações:

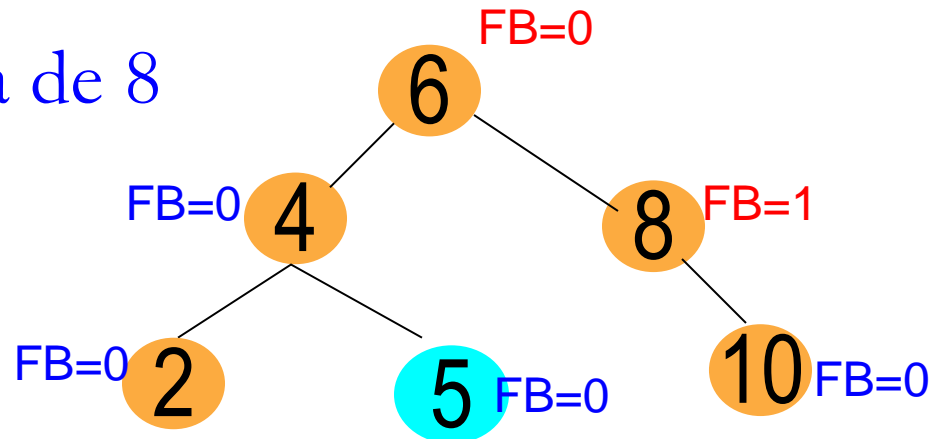
- 1) Rotação à esquerda de 4
- 2) Rotação à direita de 8

AVL: Exemplo Rotação Esquerda-Direita

- 1) Rotação à esquerda de 4



- 2) Rotação à direita de 8

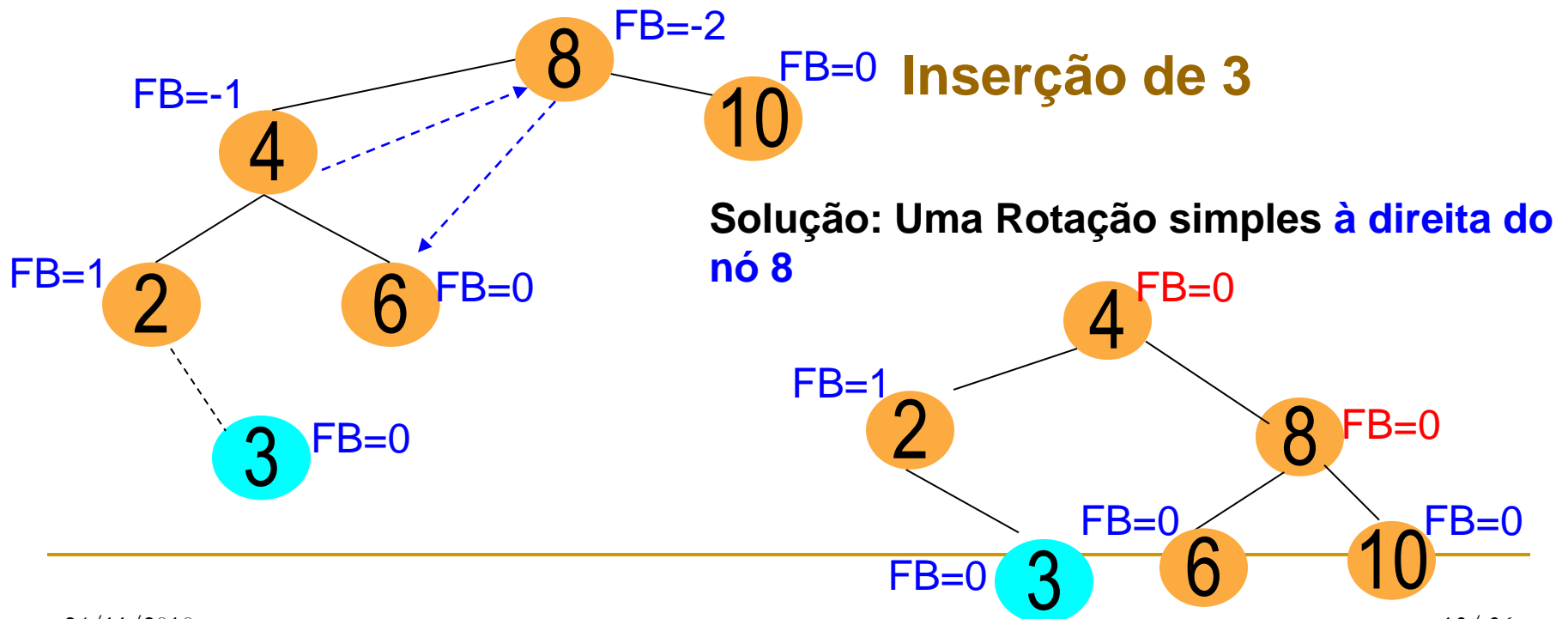


Tipo 1: Resumo

- Requer uma rotação dupla: ESQUERDA-DIREITA ou DIREITA-ESQUERDA:
 1. Rotacionar o nó com $FB = -1$ (ou 1) na direção apropriada, i.e., se FB negativo, para a direita; se positivo, para a esquerda.
 2. Rotacionar o nó com $FB = -2$ (ou 2) na direção oposta.

Casos de Rebalanceamento

- **Tipo 2:** A raiz de uma subárvore tem $FB = -2$ (ou 2) e tem filho com $FB = -1$ (ou 1), i.e., com mesmo sinal do pai.



Tipo 2: Resumo

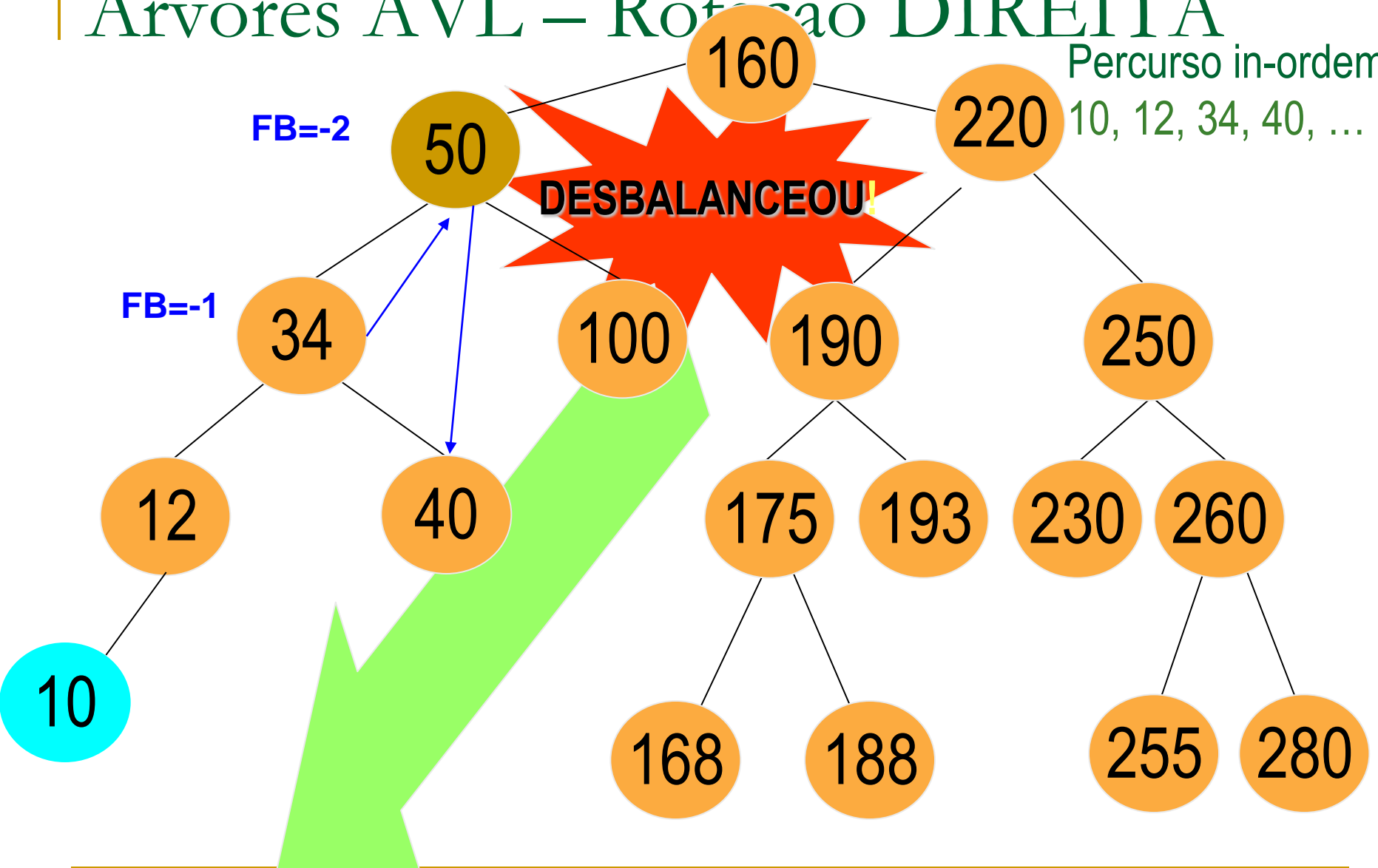
- Rotacionar uma única vez o nó de $FB = -2$ ou 2 :
 - se negativo: à direita;
 - se positivo: à esquerda.

Inserção em AVL: Resumo

- A cada inserção, verificar se balanceamento foi conservado.
- Em caso negativo (se algum nó ficou com FB igual a 2 ou -2), verificar qual caso se aplica (Tipo 1 ou Tipo 2).
- Efetuar as operações de rotação adequadas.

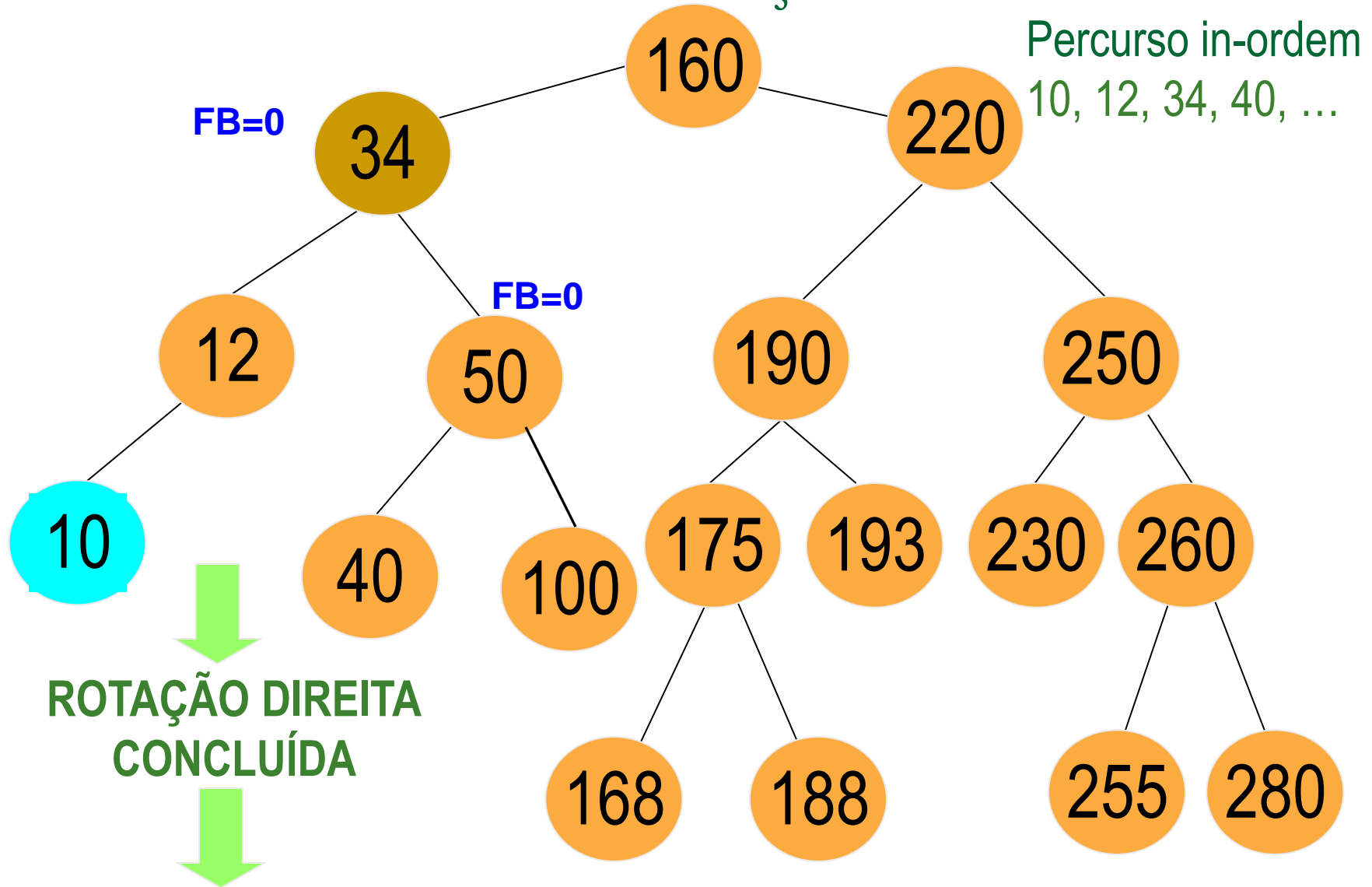
Árvores AVL – Rotação DIREITA

Percurso in-ordem
10, 12, 34, 40, ...



ROTAÇÃO DIREITA 50

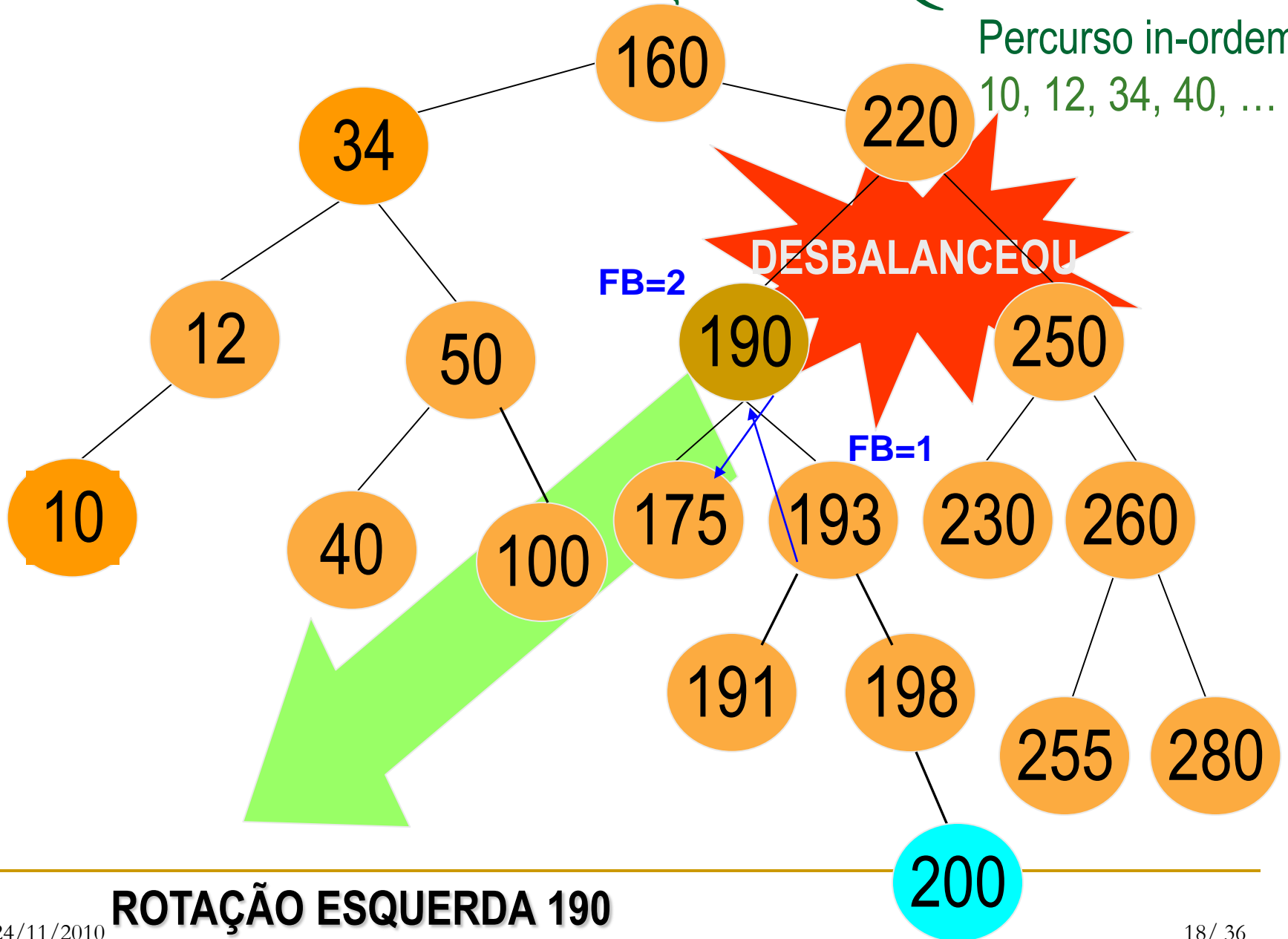
Árvores AVL – Rotação DIREITA



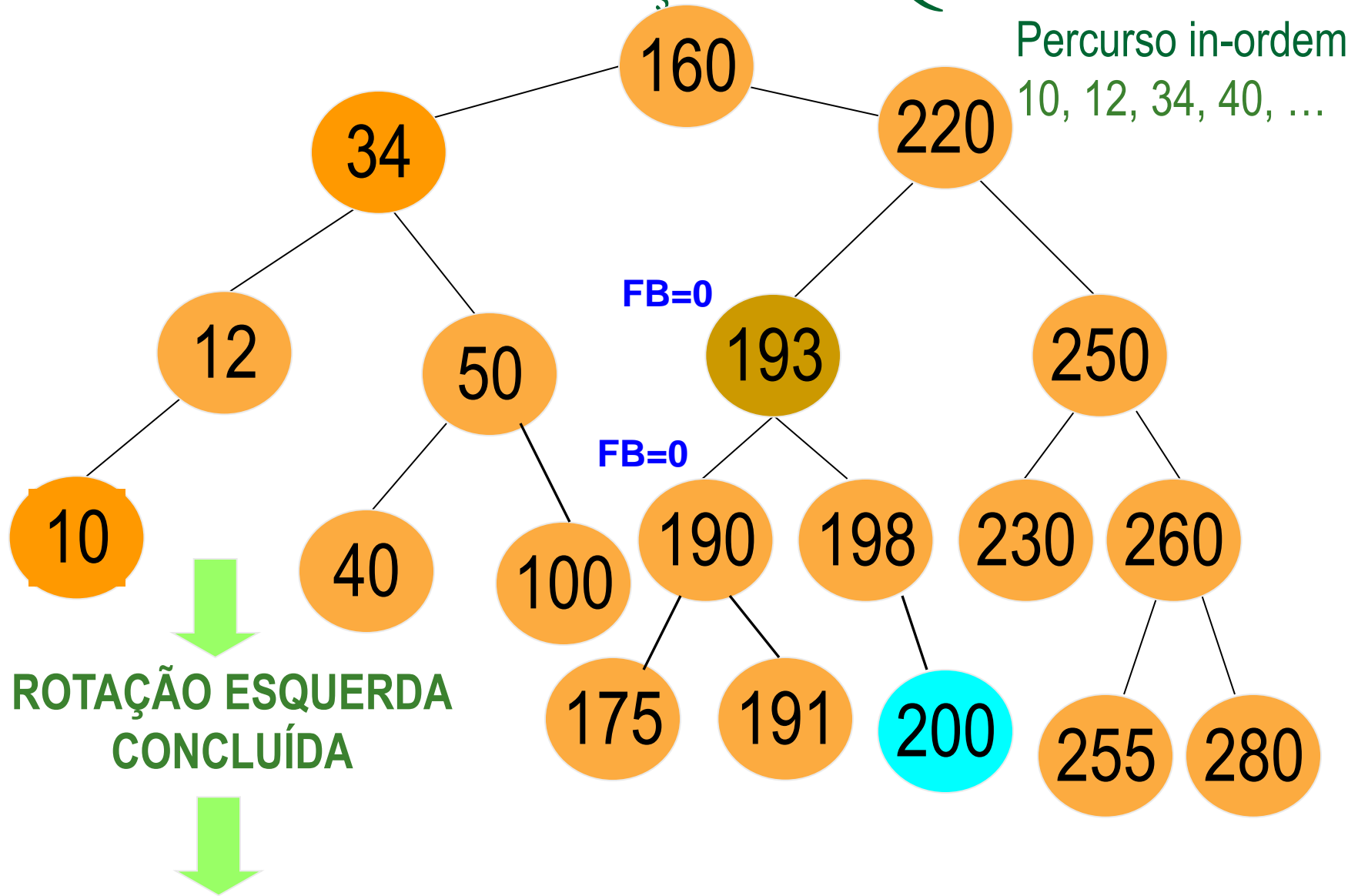
ÁRVORE NOVAMENTE BALANCEADA!!!

Árvores AVL – Rotação ESQUERDA

Percurso in-ordem
10, 12, 34, 40, ...



Árvores AVL – Rotação ESQUERDA



24/11/2010 **ÁRVORE NOVAMENTE BALANCEADA!!!**

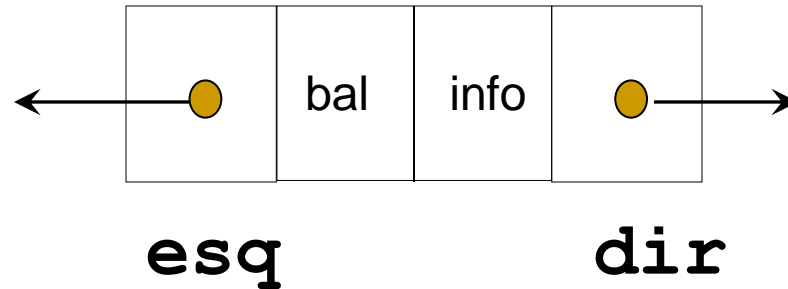
Árvores AVL

- Nos exemplos anteriores as regras foram mantidas:
 - o percurso in-ordem na árvore transformada coincide com o da árvore original (a menos do nó inserido), i.e., a árvore transformada é uma ABB;
 - a árvore transformada ficou balanceada

Árvores AVL

- Para verificar qual rotação deve ser efetuada para rebalancear a árvore, é necessário calcular o Fator de Balanceamento do nó (p):
- $FB(p) = h(\text{subarv-direita}) - h(\text{subarv-esquerda})$
 - Se FB positivo: rotações à esquerda
 - Se FB negativo: rotações à direita
- Repare que o efeito das transformações é diminuir em 1 a altura da sub-árvore cuja raiz (p) tem $|FB| = 2$ após a inserção.
- Isso assegura o rebalanceamento de todos os ancestrais de p, e portanto, o rebalanceamento de toda a árvore.
- Considere um novo campo em cada nó – **bal** – que armazena o FB do nó. Ao ser inserido como folha, o campo bal deve ser inicializado com zero.

Declaração da Estrutura



```
typedef struct no *pno;
```

```
typedef struct no{  
    int bal;  
    tipo_elem info;  
    pno dir, esq;  
}no;
```

```
typedef pno tree;
```

```
tree raiz;
```

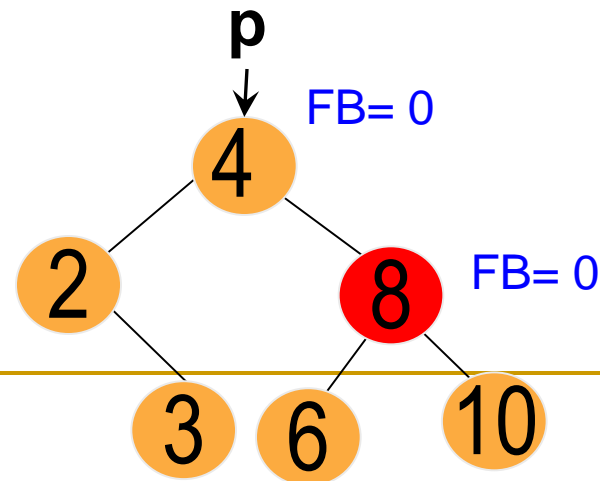
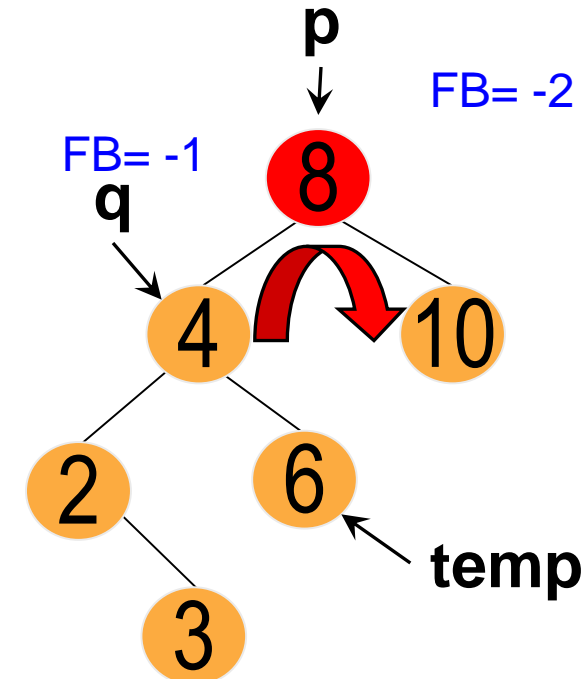
Função de Busca por uma Chave

```
pno buscar (tree p, int x) {  
  //retorna o endereço de x, se achar; Null, caso contrário//  
  
  if (p == Null) return Null  
    else if (p->info ==x) return p;  
      else if (p->info < x ) return (buscar(p->dir, x));  
        else return (buscar(p->esq, x));  
}
```

Árvores AVL

Algoritmo de Rotação à direita

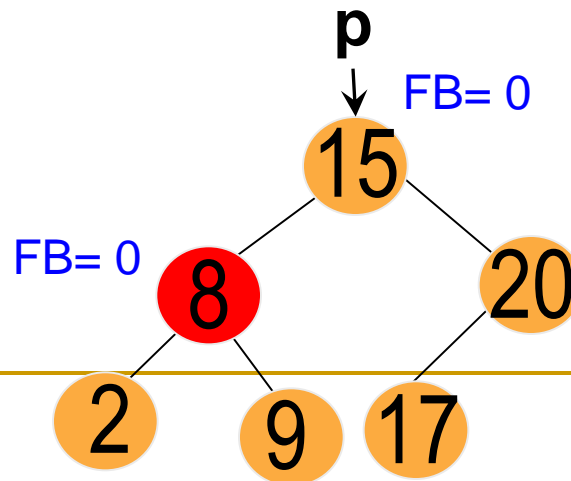
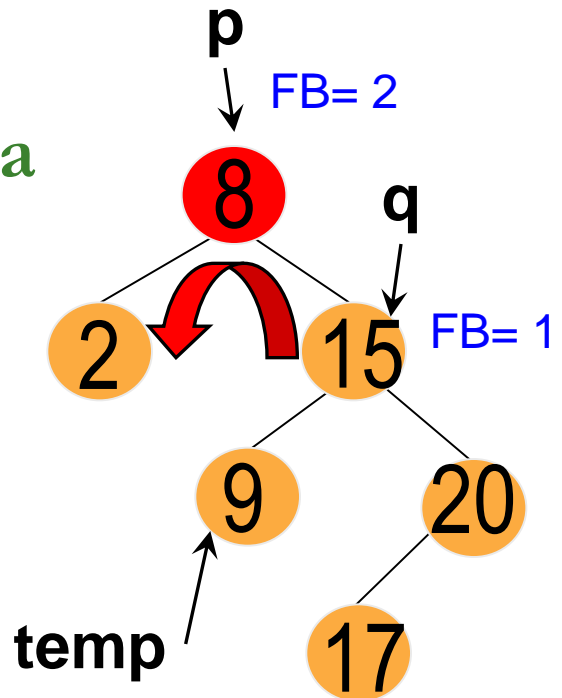
```
void rot_dir(pno *p) {  
    pno q, temp;  
  
    q = (*p)->esq;  
    temp = q->dir;  
    q->dir = (*p);  
    (*p)->esq = temp;  
    (*p) = q;  
}
```



Árvores AVL

Algoritmo de Rotação à esquerda

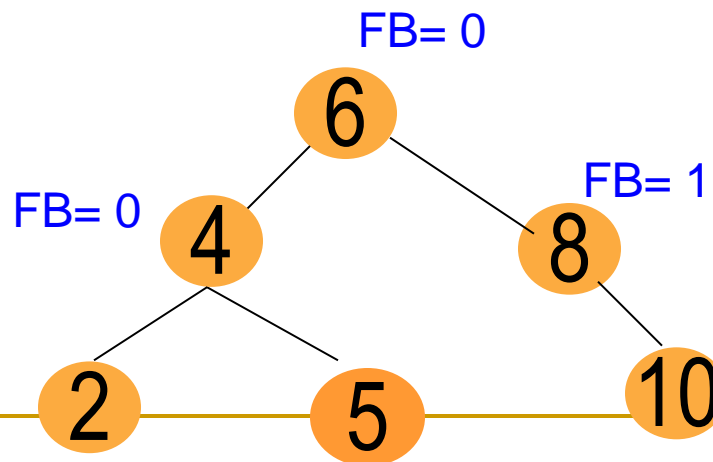
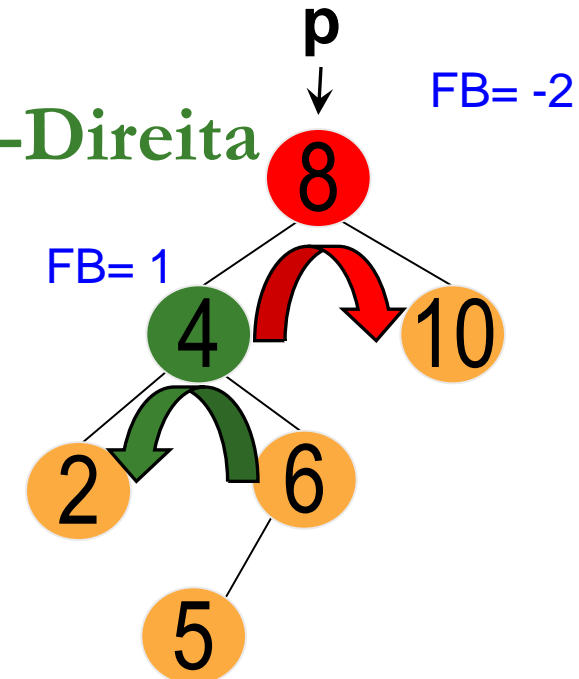
```
void rot_esq(pno *p) {  
    pno q, temp;  
  
    q = (*p)->dir;  
    temp = q->esq;  
    q->esq = (*p);  
    (*p)->dir = temp;  
    (*p) = q;  
}
```



Árvores AVL

Algoritmo de Rotação Esquerda-Direita

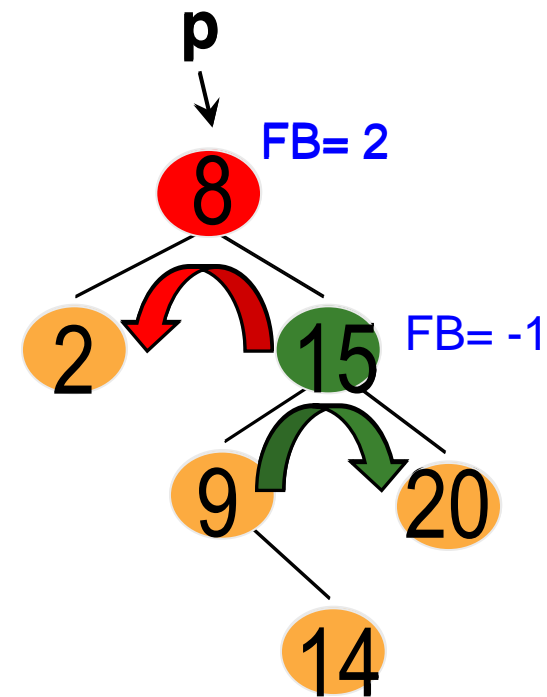
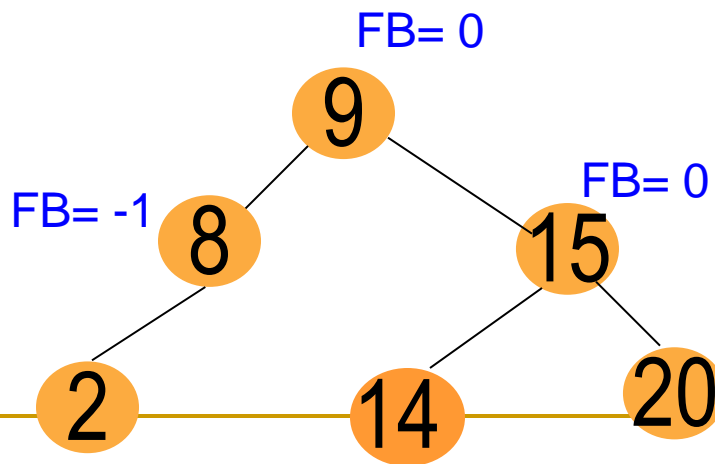
```
void rot_esq_dir(pno *p) {  
    rot_esq(&(*p)->esq);  
    rot_dir(&(*p));  
}
```



Árvores AVL

Algoritmo de Rotação Direita-Esquerda

```
void rot_dir_esq(pno *p) {  
    rot_dir(&(*p)->dir);  
    rot_esq(&(*p));  
}
```



Calculando o FB

- Numa próxima inserção, é preciso verificar o FB dos nós para saber se houve desbalanceamento ou não.
- Assim, a cada inserção e rebalanceamento, é necessário recalcular o valor de FB para os nós envolvidos nas rotações.
- Repare que não é preciso calcular a altura das subárvores

Algoritmo **Recursivo** de Busca e Inserção em **Árvore AVL**

ins_AVL (x, raiz, flag);

chave de
busca/inserção
(E)

raiz da árvore
(E/S)

auxiliar para propagar
verificação de FB (E/S)

```
void ins_AVL(tipo_elem x, pno *p, boolean *flag) {
```

```
    if (*p == NULL) {
```

```
        /*árvore vazia: insere e sinaliza alteração de FB*/
```

```
        /*insere nó p com conteúdo x, como nó folha*/
```

```
        *p = (pno) malloc(sizeof(pno));
```

```
        (*p)->esq = NULL;
```

```
        (*p)->dir = NULL;
```

```
        (*p)->info = x;
```

```
        (*p)->bal = 0;
```

```
        *flag = TRUE;
```

```
    return;
```

```
}
```

...

```

if (x < (*p)->info) { /*recursividade à esquerda*/
    ins_AVL(x, (*p)->esq, flag);
    if (*flag) /*inseriu: verificar balanceamento*/
        switch ((*p)->bal) {
            case 1: /*mais alto a direita*/
                (*p)->bal = 0; /*balanceou com ins. esq*/
                *flag = FALSE; /*interrompe propagação*/
                break;
            case 0:
                (*p)->bal = -1; /*ficou maior à esq.*/
                break;
                case -1: /*FB(p) = -2*/
                    CASO1(p); /*p retorna balanceado*/
                    *flag = FALSE;
                    break; /*não propaga mais*/
        }
    }
return;
}

```

```

if (x > (*p)->info) { /*recursiva a direita*/
    ins_AVL(x, &(*p)->dir, flag);
    if (*flag) /*inseriu: verificar balanceamento*/
        switch ((*p)->bal) {
            case -1: /*era mais alto à esq.: zera FB*/
                (*p)->bal = 0; *flag = FALSE; break;
            case 0: (*p)->bal = 1; break;
            /*direita fica maior: propaga verificação*/
            case 1: /*FB(p) = 2 e p retorna balanceado*/
                CASO2(p); *flag = FALSE; break;
        }
    return;
}
/* else if (x = p->info) – nada a fazer; pare!*/
}

```



```
void CASO1 (pno *p) {
```

```
    /*x foi inserido à esq. de p e causou FB= -2*/
```

```
    pno u;
```

```
    u = (*p)->esq;
```

```
    if (u->bal == -1) /*caso sinais iguais
```

```
        e negativos: rotação à direita*/
```

```
        rot_dir(&(*p));
```

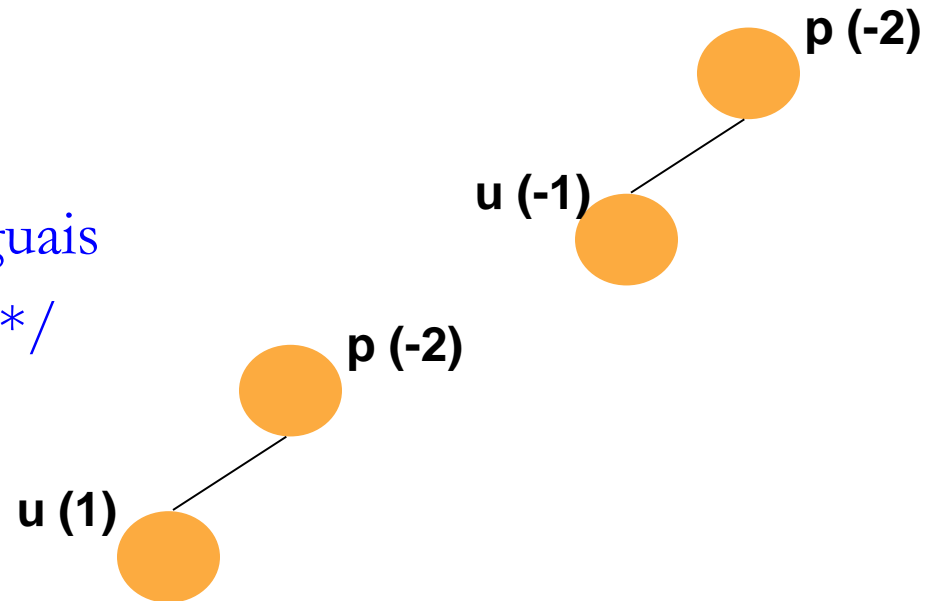
```
    else /*caso sinais trocados:
```

```
        rotação dupla u + p*/
```

```
        rot_esq_dir(&(*p));
```

```
    (*p)->bal = 0;
```

```
}
```



```
void CASO2 (pno *p) {
```

```
    /*x foi inserido à direita de p e causou FB=2*/
```

```
    pno u;
```

```
    u = (*p)->dir;
```

```
    if (u->bal == 1) /*caso sinais iguais  
        e positivos: rotação à esquerda*/
```

```
        rot_esq(&(*p));
```

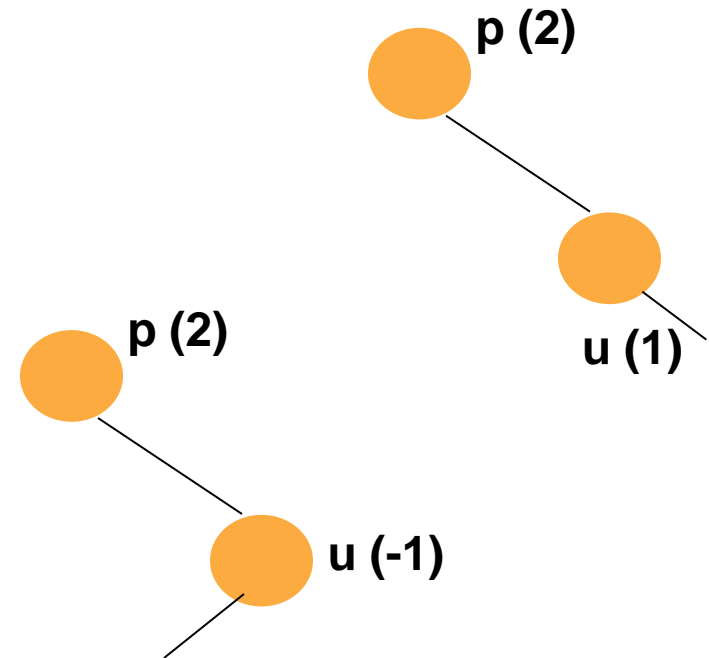
```
    else /*caso sinais trocados:
```

```
        rotação dupla u + p*/
```

```
        rot_dir_esq(&(*p));
```

```
    (*p)->bal = 0;
```

```
}
```



Refazendo as rotações duplas para ajustar FB

```
void rot_dir_esq(pno *p) {
```

```
    pno z, v;
```

```
    z = (*p)->dir; v = z->esq;
```

```
    z->esq = v->dir;
```

```
    v->dir = z;
```

```
    (*p)->dir = v->esq;
```

```
    v->esq = *p;
```

rot_dir(z)

rot_esq(p)

```
/*atualizar FB de z e p em função de FB de v - a nova raiz*/
```

```
if (v->bal == 1) {
```

```
    (*p)->bal = -1;
```

```
    z->bal = 0;
```

```
} else {
```

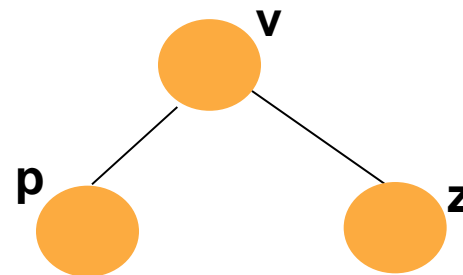
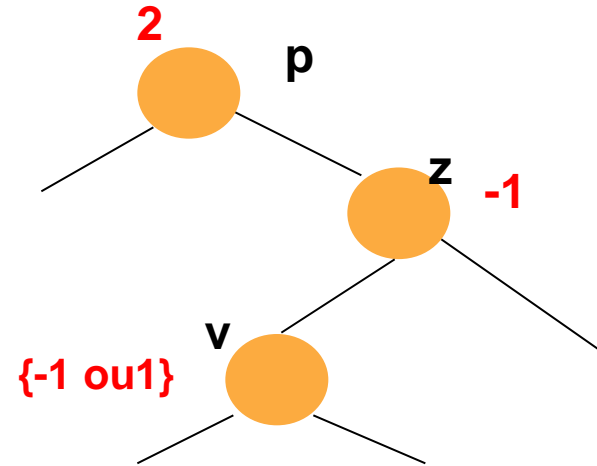
```
    (*p)->bal = 0;
```

```
    z->bal = 1;
```

```
}
```

```
*p = v;
```

```
}
```



Eliminação em AVL

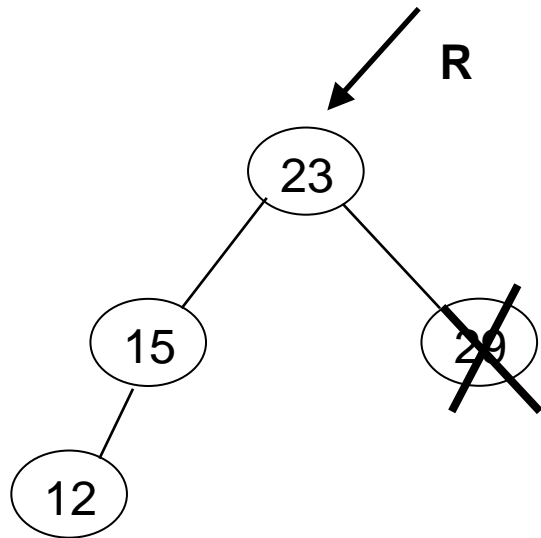
- Operações análogas à da ABB (eliminação simples ou substituição por menor chave da direita ou maior da esquerda), porém, como podem desbalancear a árvore, requerem a verificação dos valores de Fatores de Balanceamento e acionamento das rotações adequadas, quando for o caso.
 - Exemplo
-

AVL

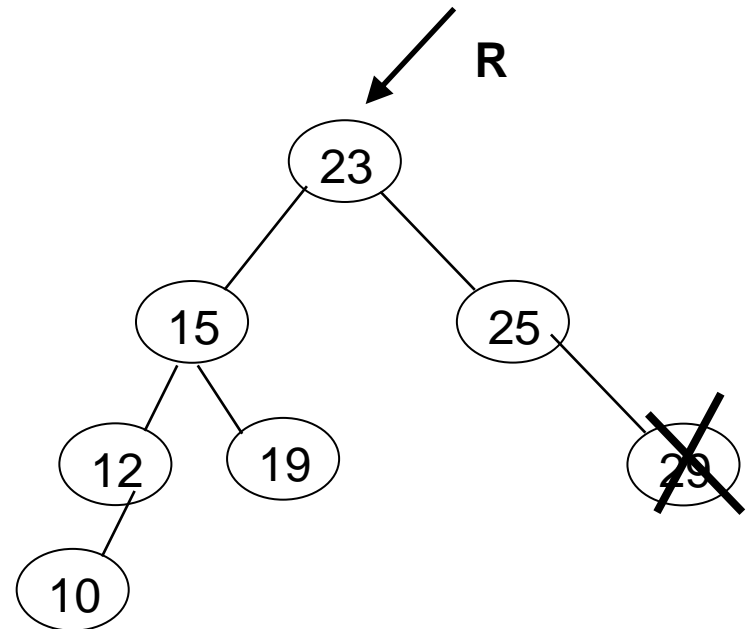
- Como fazer a remoção de elementos da AVL?
 - Análogo à ABB, recalculando o Fator de Balanceamento após remoção, e fazendo rotações, se necessário
-

AVL: remoção

■ Exemplos



remoção de 29 = inserção de 12

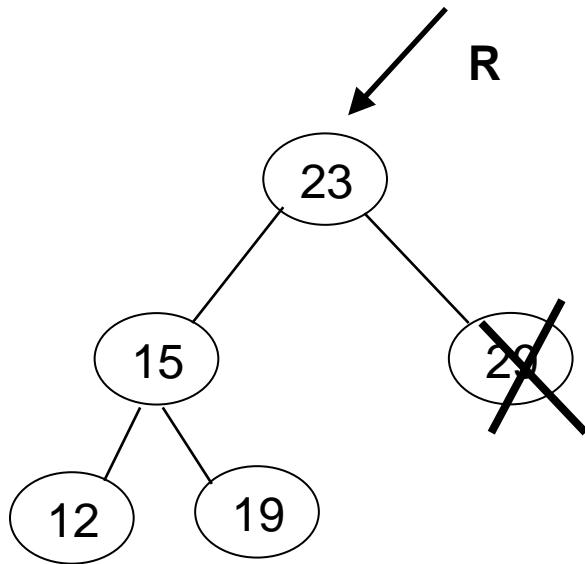


remoção de 29 = inserção de 10

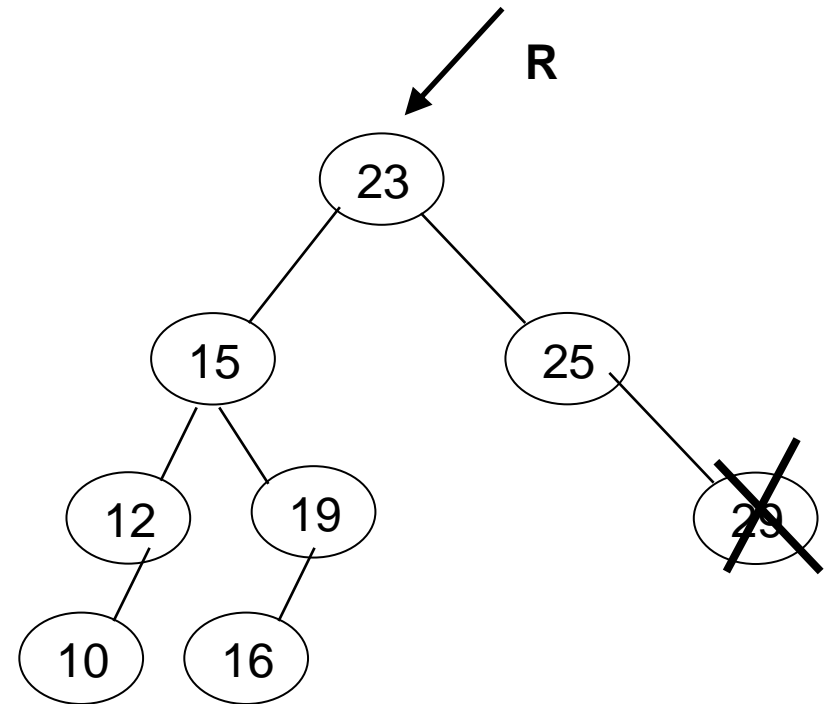
Como balancear?

AVL: remoção

■ Exemplos



remoção de 29 = inserção de 12 ou 19



remoção de 29 = inserção de 10 ou 16

Como balancear?

Rotação simples em R

AVL: remoção

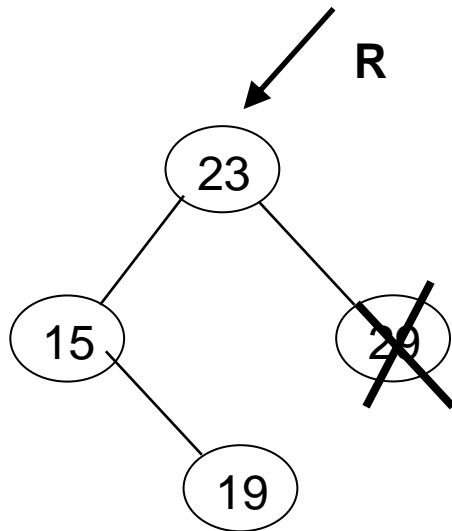
- Primeiro caso

- Rotação simples em R (FB=2 ou -2) com filho com fator de balanceamento de mesmo sinal (1 ou -1) ou zero

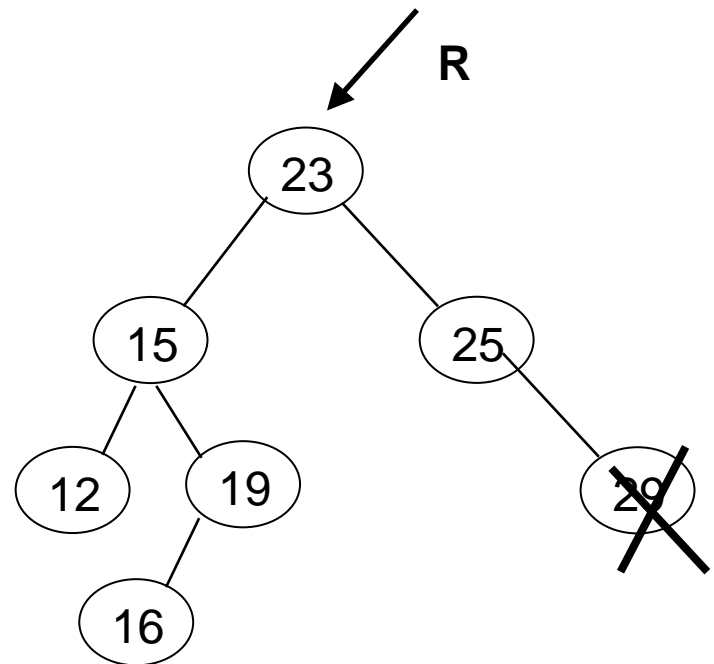
- Se R negativo, rotaciona-se para a direita; caso contrário, para a esquerda

AVL: remoção

■ Exemplos



remoção de 29 = inserção de 19

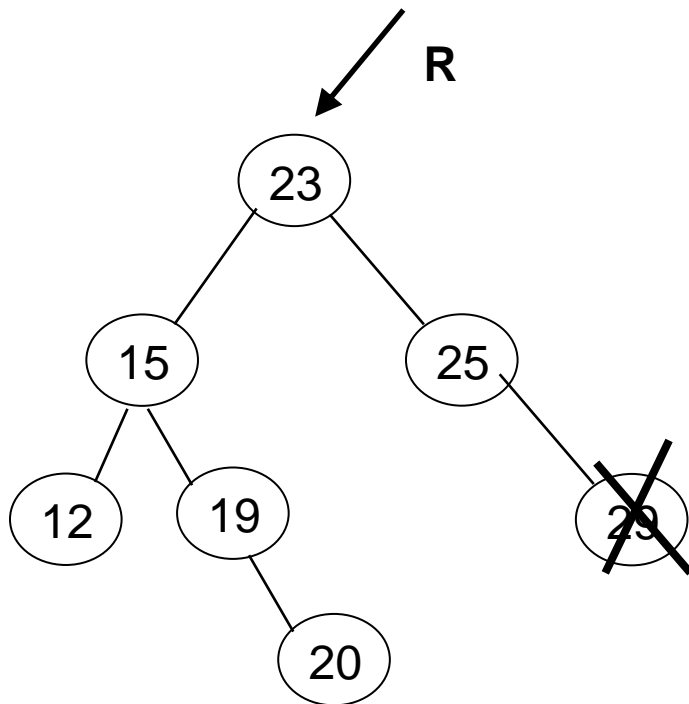


remoção de 29 = inserção de 16

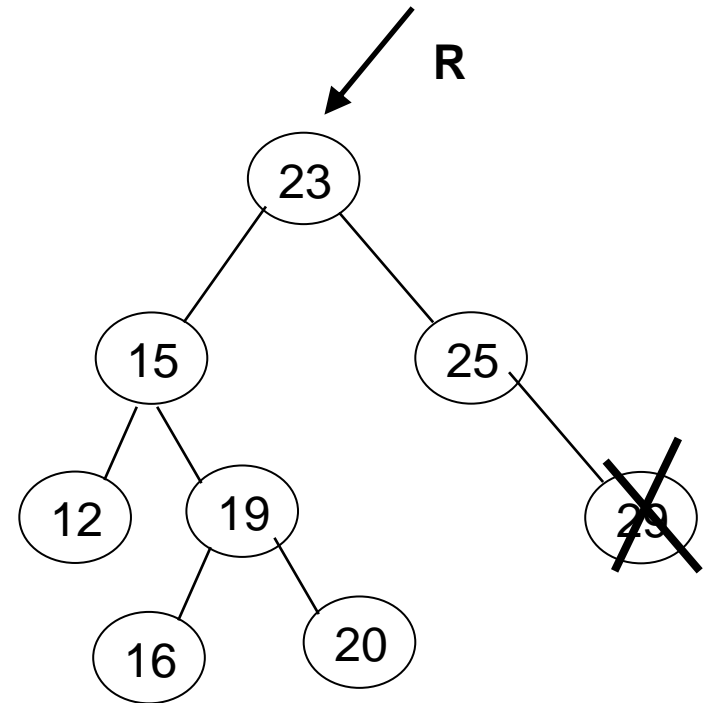
Como balancear?

AVL: remoção

■ Exemplos



remoção de 29 = inserção de 20



remoção de 29 = inserção de 16 ou 20

Como balancear?

Rotação dupla: filho de R e R

AVL: remoção

- Segundo caso

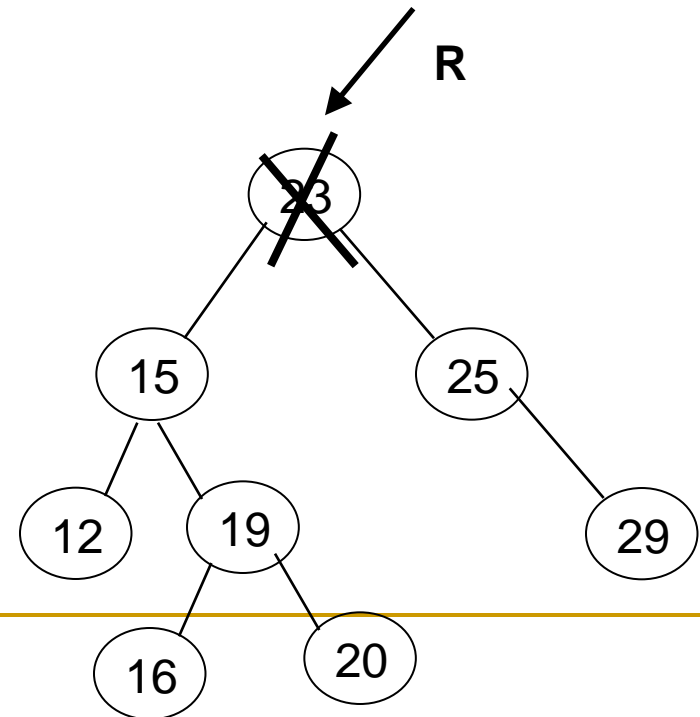
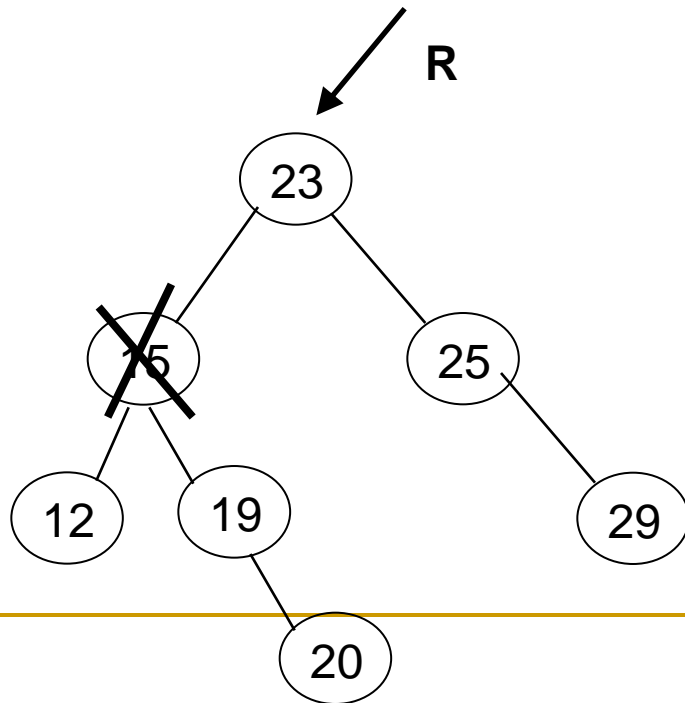
- Rotação dupla quando R (FB=2 ou -2) e seu filho (1 ou -1) tem fatores de balanceamento com sinais opostos

- Rotaciona-se o filho para o lado do desbalanceamento do pai

- Rotaciona-se R para o lado oposto do desbalanceamento

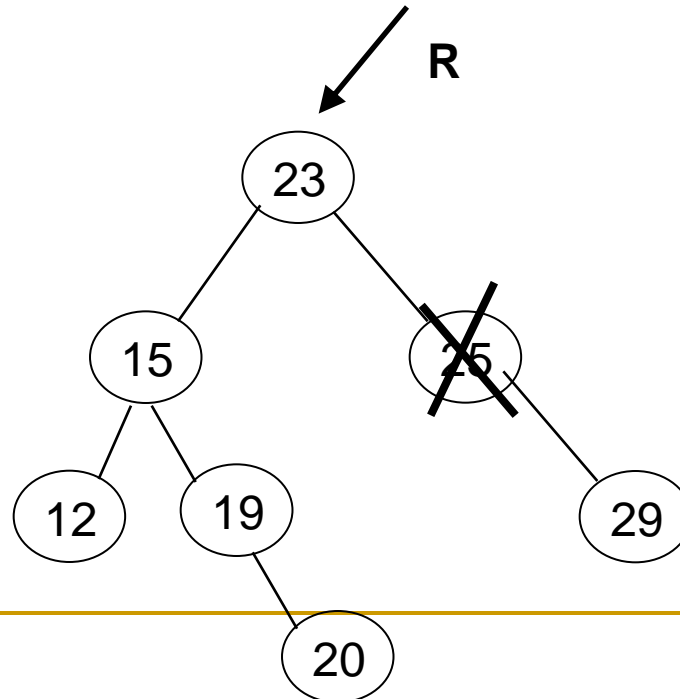
AVL: remoção

- Questão: como remover um nó intermediário em vez de um nó folha?
 - É necessário balancear?



AVL: remoção

- Questão: como remover um nó intermediário em vez de um nó folha?
 - É necessário balancear?



AVL

- Exercício para casa
 - Implementar sub-rotina de remoção de elemento de uma AVL