

# Árvores binárias AVL

SCC-202 – Algoritmos e Estruturas de  
Dados I

# Árvores binárias de busca (ABB)

---

- Excelentes para busca
  - $O(\log_2 n)$
- Sabe-se que
  - Lista encadeada é
    - Eficiente para inserção e remoção dinâmica de elementos, mas ineficiente para busca
  - Lista sequencial (ordenada) é
    - Eficiente para busca, mas ineficiente para inserção e remoção de elementos

mas... ABBs: solução eficiente para inserção, remoção e busca

# ABB

---

- **Contra-exemplo**

- Inserção dos elementos na ordem em que aparecem
  - A, B, C, D, E, ..., Z
  - 1000, 999, 998, ..., 1

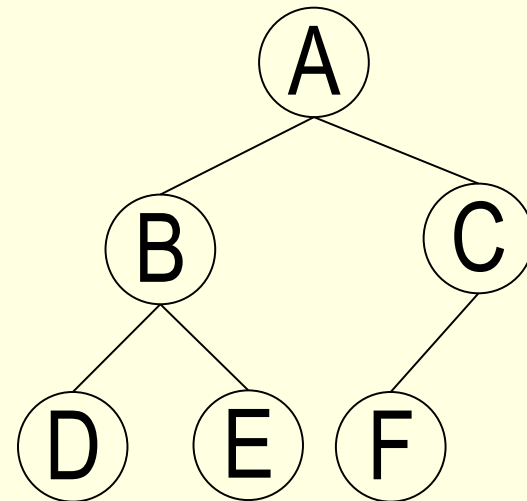
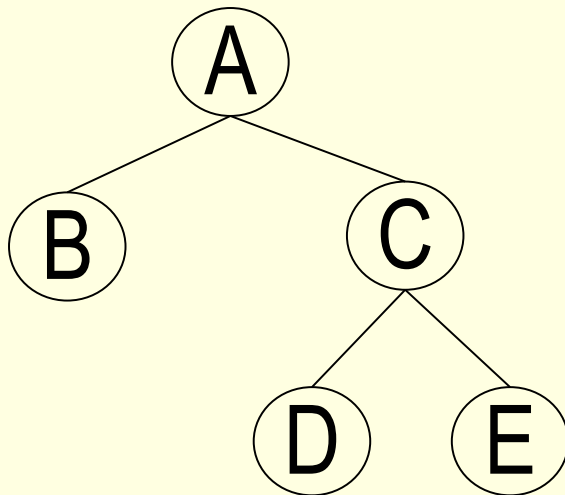
# ABB

---

- O desbalanceamento da árvore pode tornar a busca tão ineficiente quanto a busca seqüencial (no pior caso)
  - $O(N)$
- Solução:
  - Balanceamento da árvore

# Árvores balanceadas

- Uma árvore binária é dita balanceada se, para cada nó, as alturas de suas duas subárvores diferem de, no máximo, 1

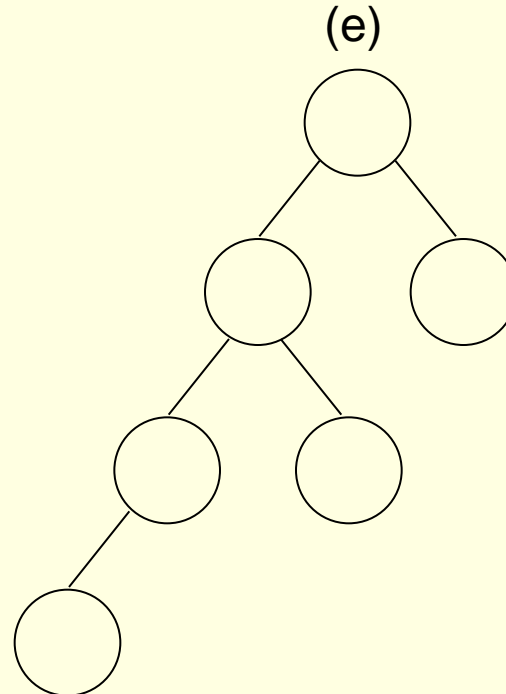
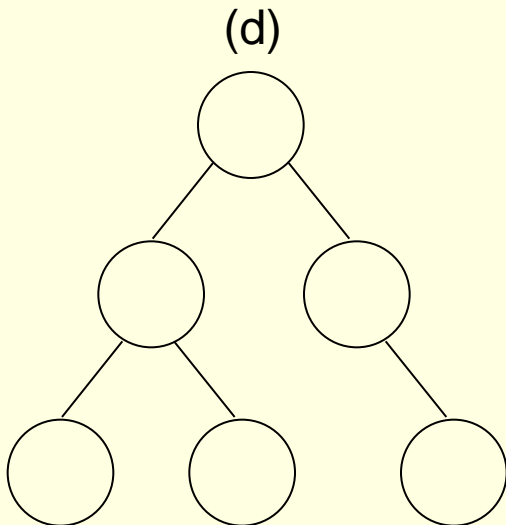
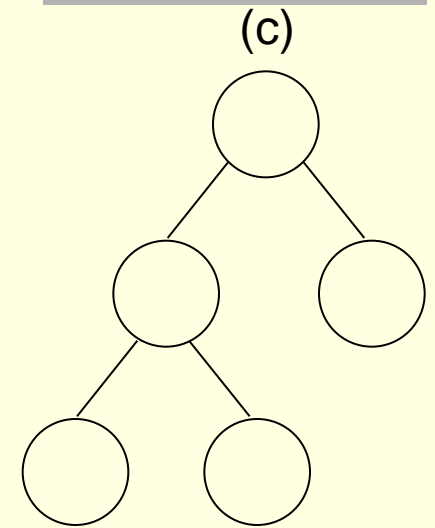
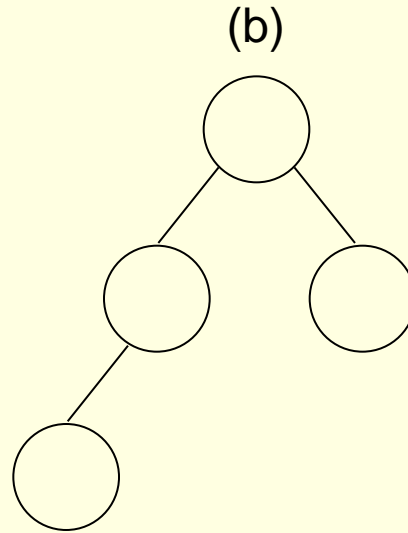
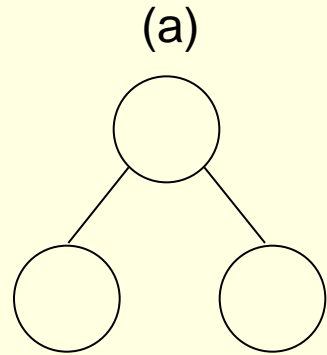


# AVL

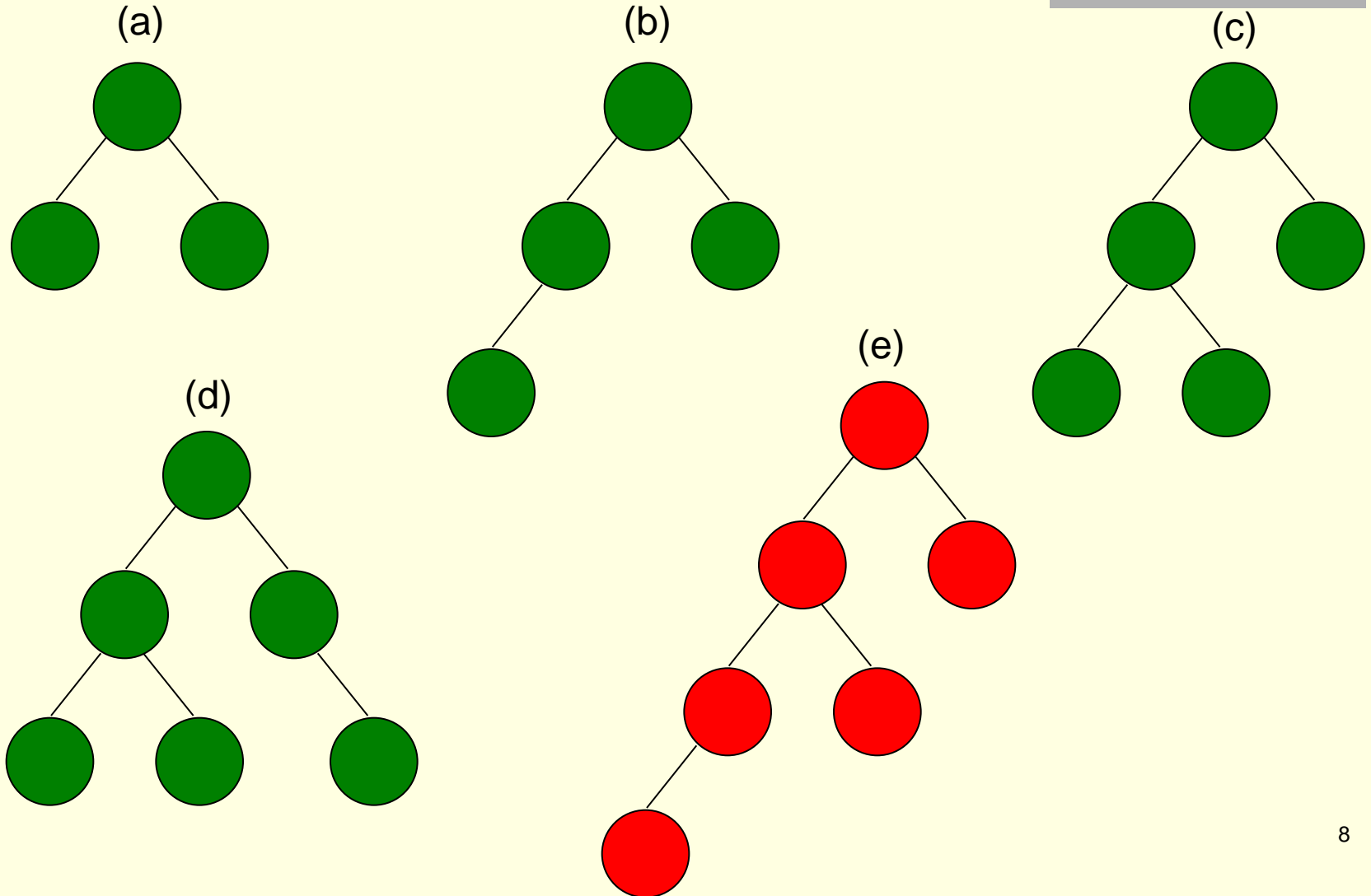
---

- Árvore binária de busca balanceada
  - Para cada nó, as alturas das subárvores diferem em 1, no máximo
  - Proposta em 1962 pelos matemáticos russos G.M. Adelson-Velski e E.M. Landis
    - Métodos de inserção e remoção de elementos desenvolvidos de forma que a árvore não fique desbalanceada

# AVL: quem é e quem não é?



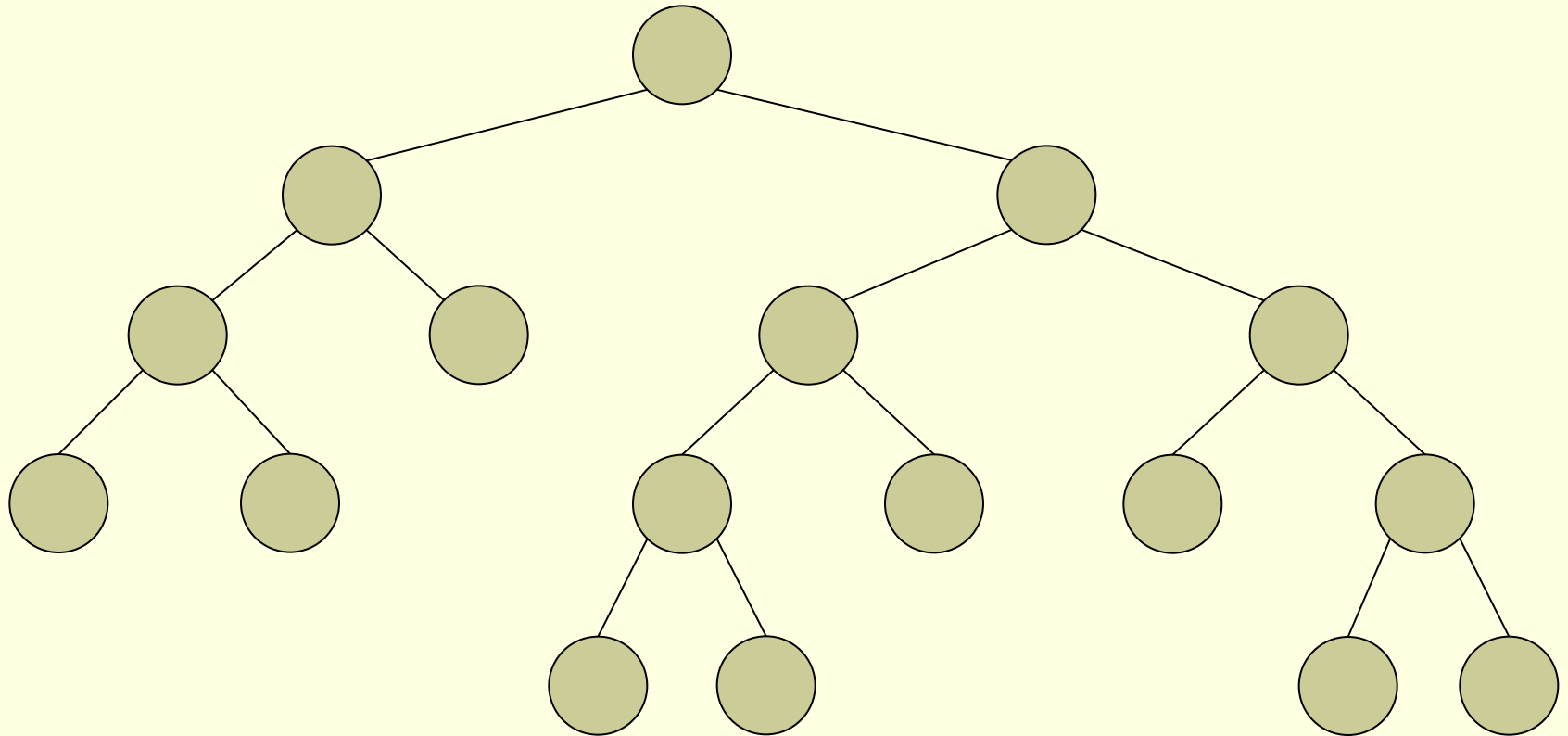
# AVL: quem é e quem não é?





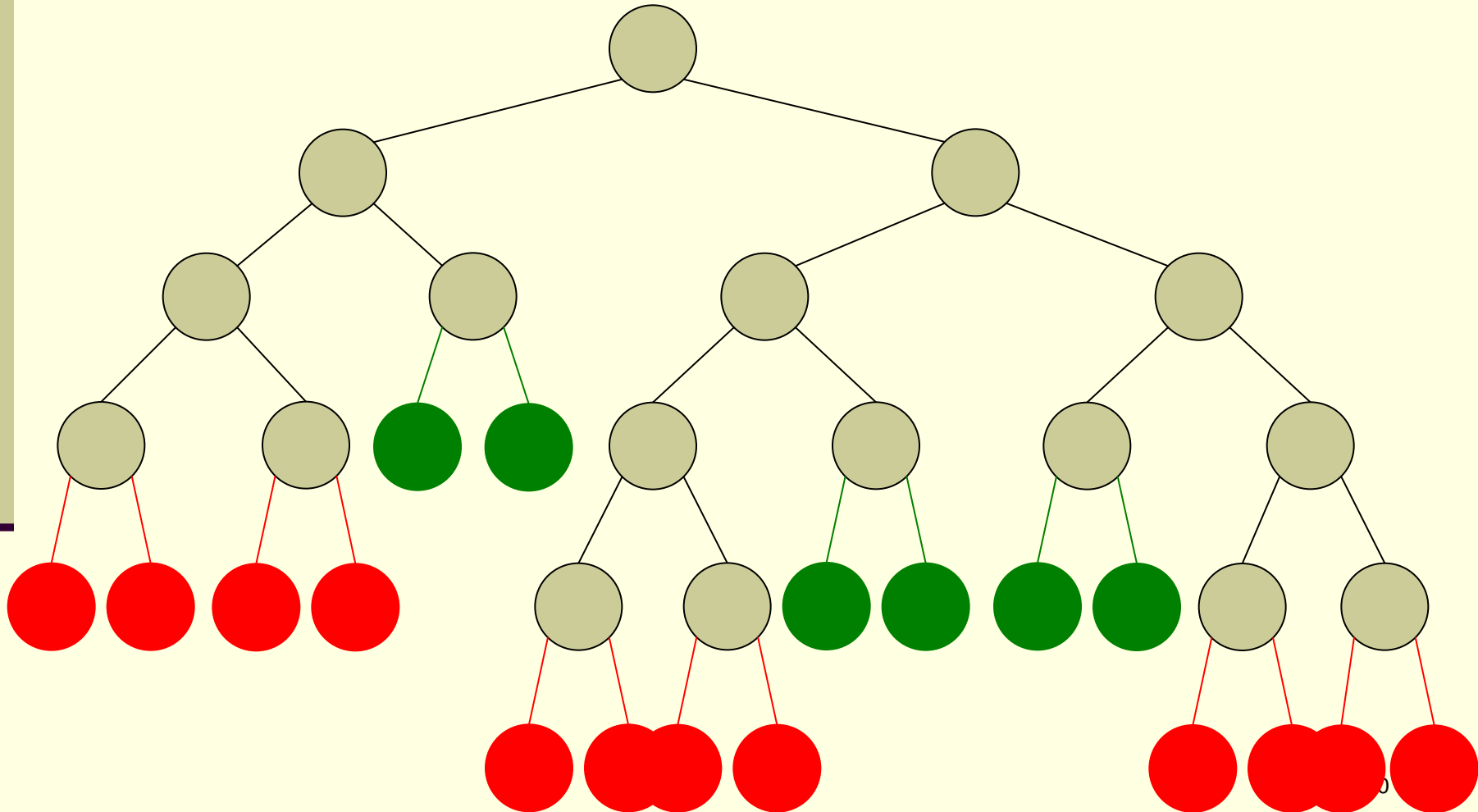
# Pergunta: a árvore abaixo é AVL?

---



# Exercício: onde se pode incluir um nó para a AVL continuar sendo AVL?

---



# AVL

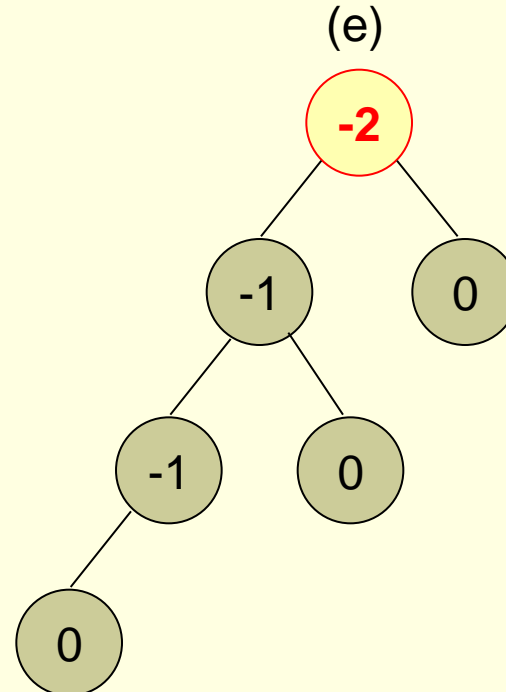
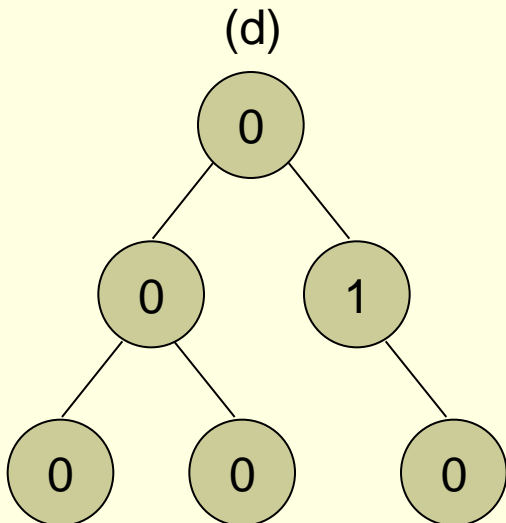
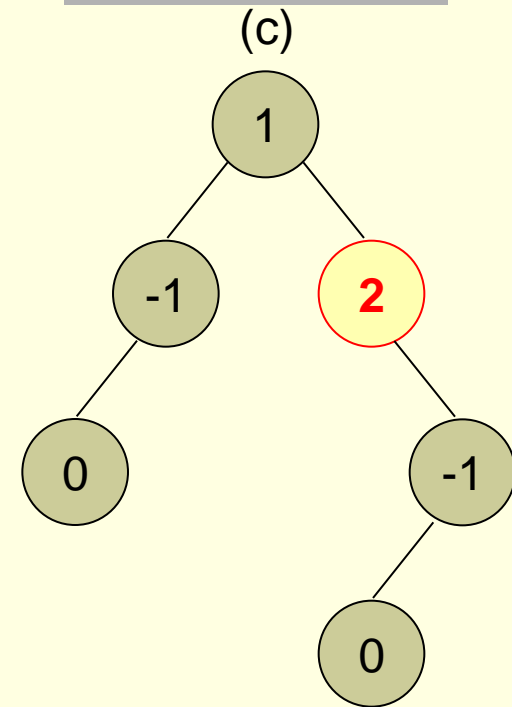
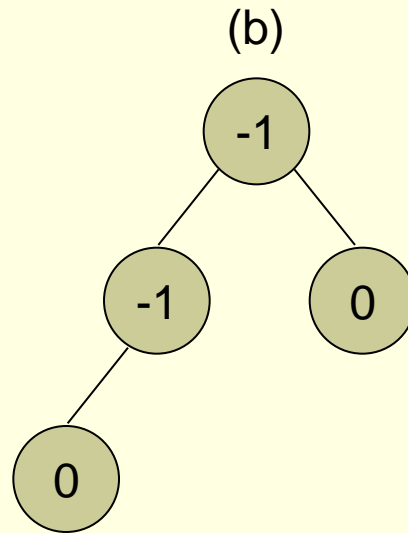
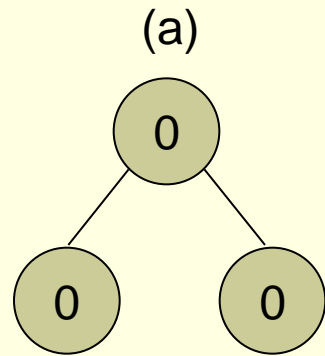
---

- Como é que se sabe **quando é necessário balancear** a árvore?
  - Se a diferença de altura das subárvores deve ser 1, no máximo, então temos que procurar diferenças de alturas maiores do que isso
  - *Possível solução*: cada nó pode manter a diferença de altura de suas subárvores
    - Convencionalmente chamada de fator de balanceamento (FB) do nó

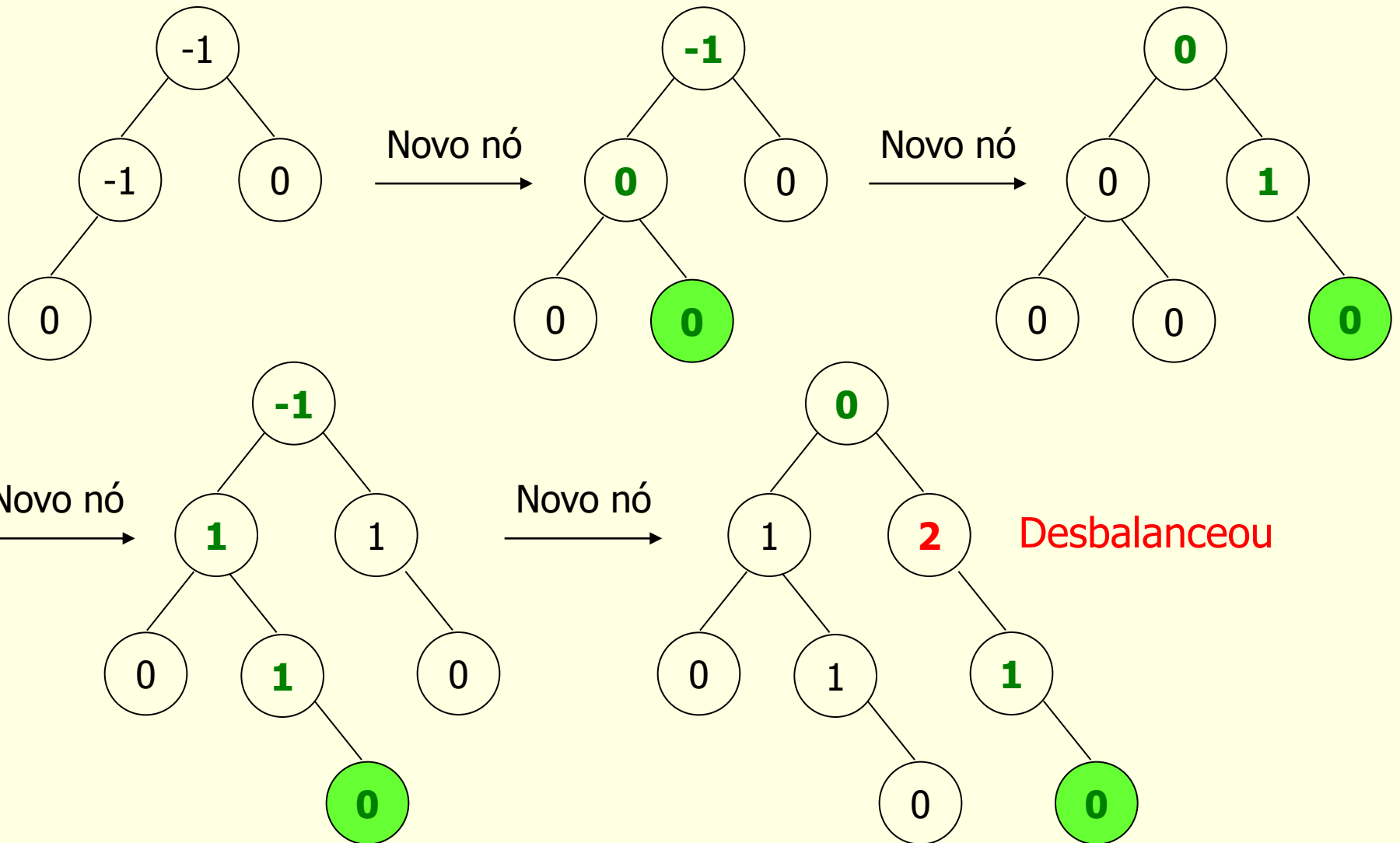
# AVL

- **Fatores de balanceamento** (FB's) dos nós
  - Altura da subárvore direita menos altura da subárvore esquerda
    - $H_d - H_e$
  - Atualizados sempre que a árvore é alterada (elemento é inserido ou removido)
  - Quando um fator é 0, 1 ou -1, a árvore está balanceada
  - Quando um fator se torna 2 ou -2, a árvore está desbalanceada
    - Operações de balanceamento!

# AVL: quem é e quem não é



# AVL: exemplo de desbalanceamento



# AVL

---

## ■ Controle do balanceamento

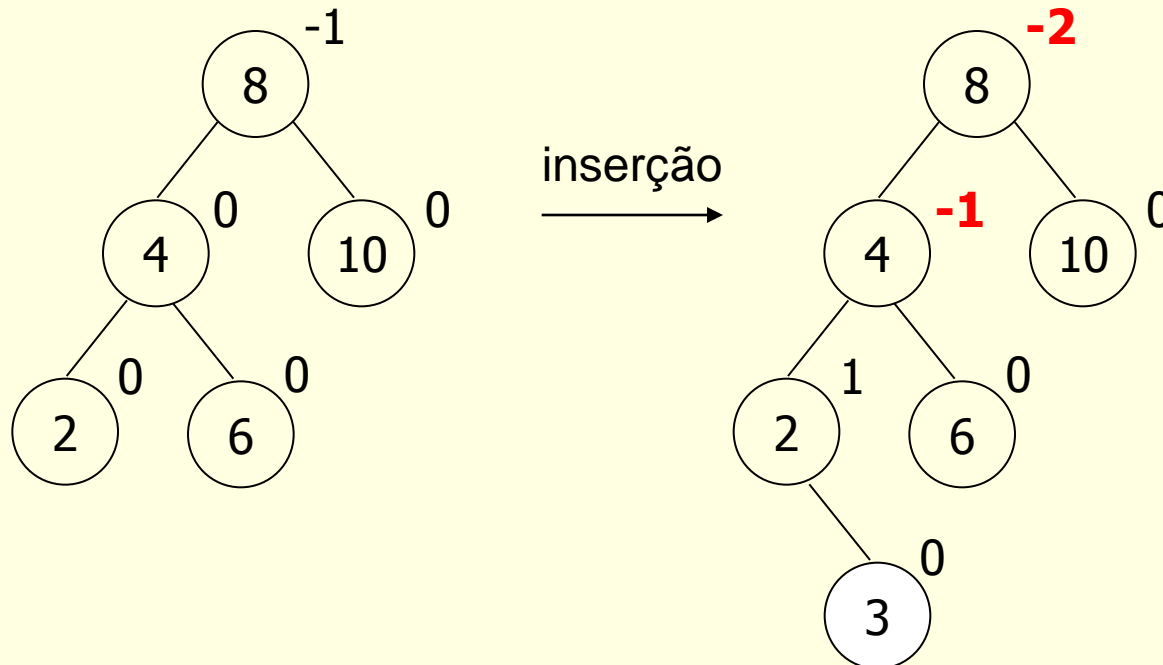
- Altera-se o algoritmo de inserção para balancear a árvore quando ela se tornar desbalanceada após uma inserção (nó com FB 2 ou -2)
  - Rotações
    - Se árvore pende para esquerda (FB negativo), rotaciona-se para a direita
    - Se árvore pende para direita (FB positivo), rotaciona-se para a esquerda
  - 2 casos podem acontecer

# AVL: primeiro caso

- Raiz de uma subárvore com FB -2 e um nó filho com FB -1  
ou

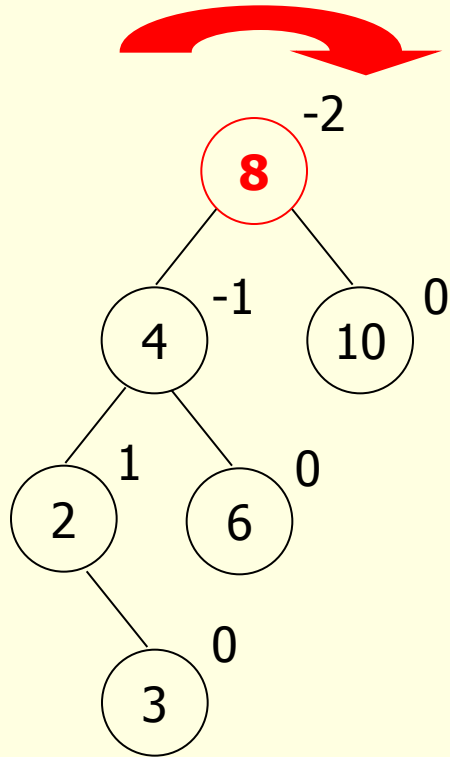
- Raiz de uma subárvore com FB 2 e um nó filho com FB 1

*Os fatores de balanceamento têm **sinais iguais**: subárvores do nó raiz e do filho pendem para o mesmo lado*



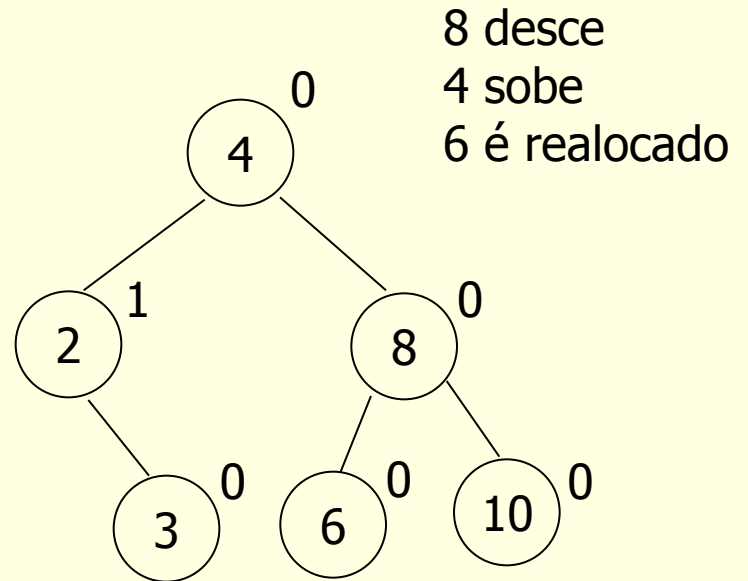


# AVL: primeiro caso



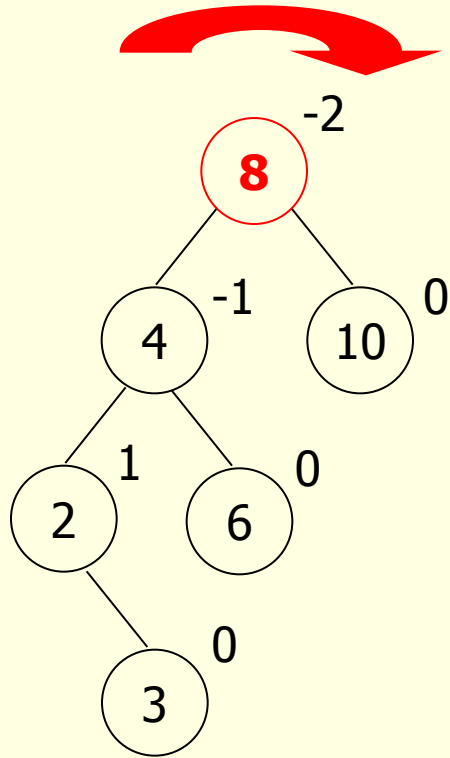
Pendendo para a esquerda

Rotação do nó pai para a direita (nó com FB=2)



Árvore balanceada

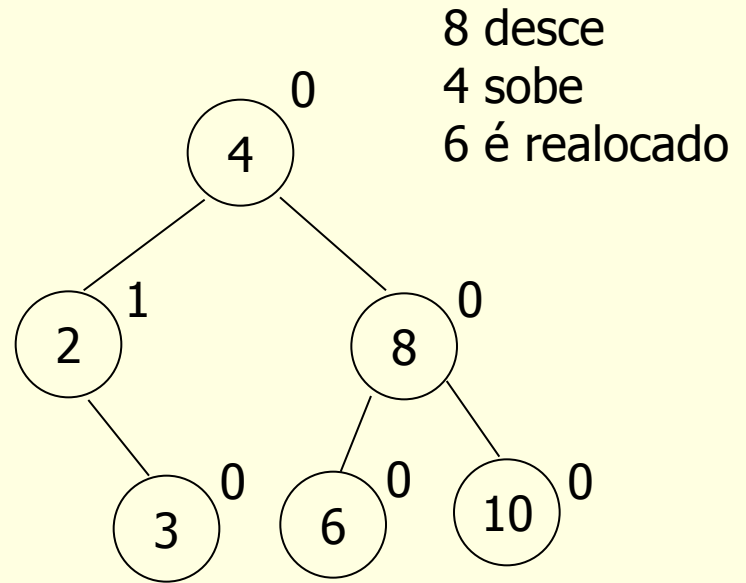
# AVL: primeiro caso



Pendendo para a esquerda

Rotação do nó pai para a direita (nó com  $FB=2$ )

Rotação simples DD (Direita-Direita)



Árvore balanceada

# AVL: primeiro caso

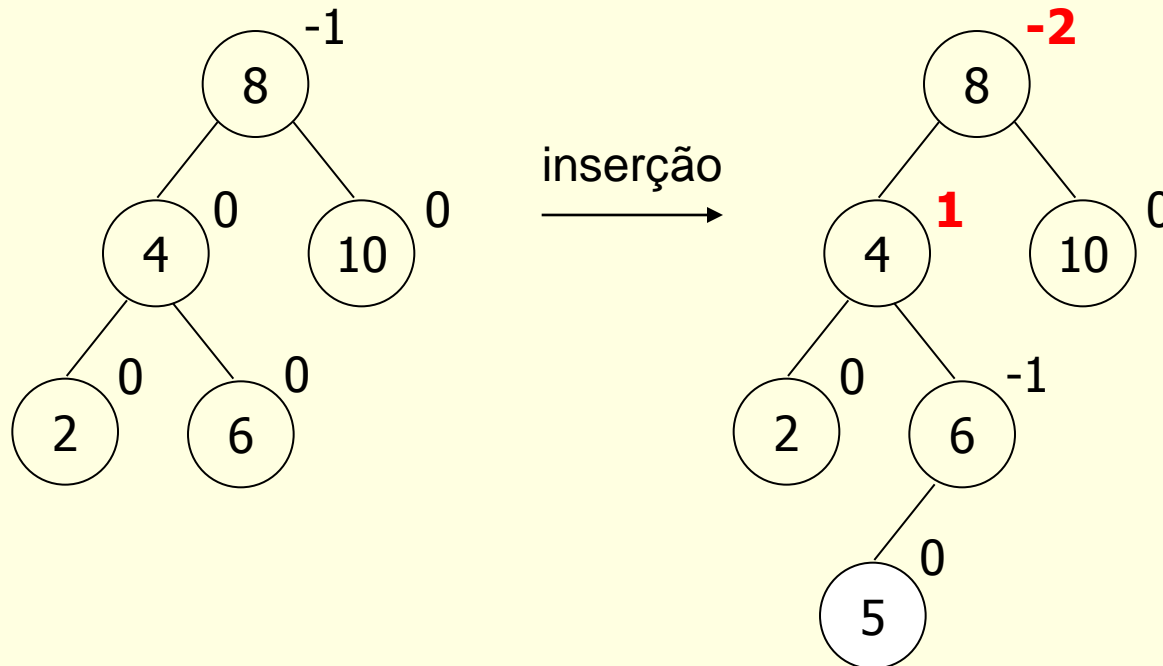
---

- Quando subárvores do pai e filho pendem para um mesmo lado
  - Rotação simples para o lado oposto
    - EE (Esquerda-Esquerda) ou DD (Direita-Direita)
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

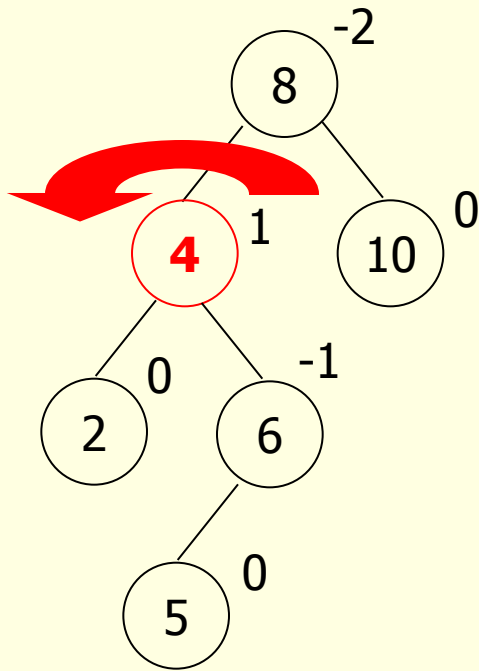
# AVL: segundo caso

- Raiz de uma subárvore com FB -2 e um nó filho com FB 1
- ou
- Raiz de uma subárvore com FB 2 e um nó filho com FB -1

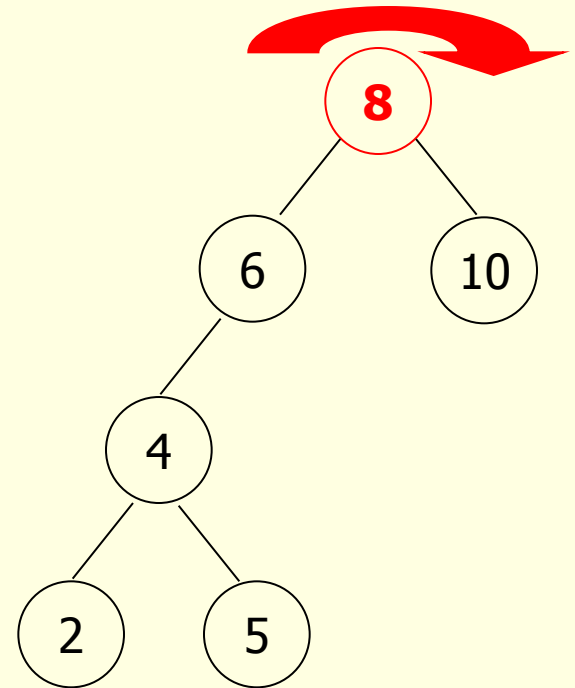
Os fatores de balanceamento têm *siniais opostos*: subárvore do nó raiz pende para um lado e subárvore do nó filho pende para o outro



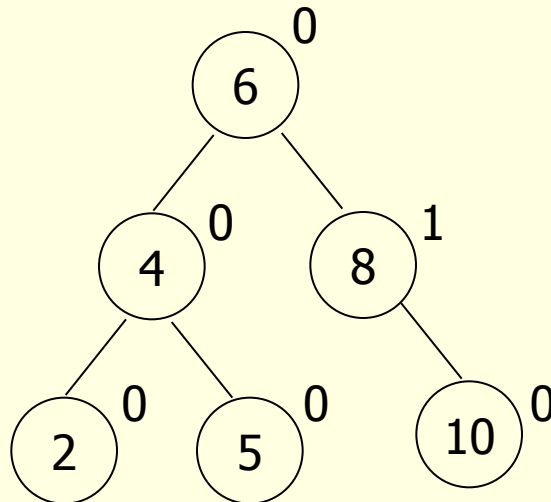
# AVL: segundo caso



Rotação do nó filho para a esquerda (nó com FB=1)

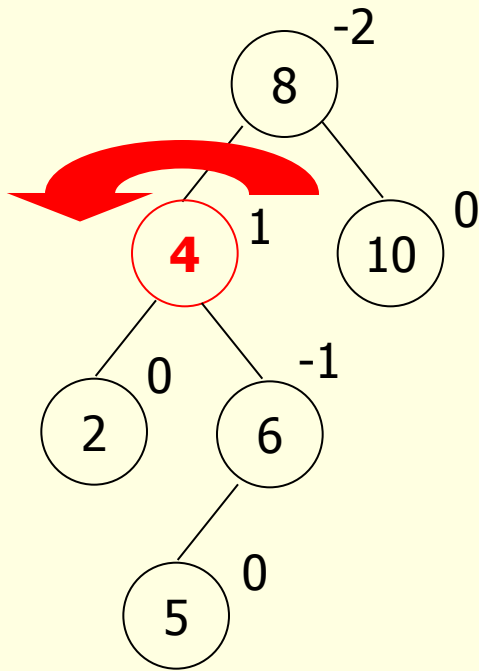


Rotação do nó pai para a direita (nó com FB=-2)



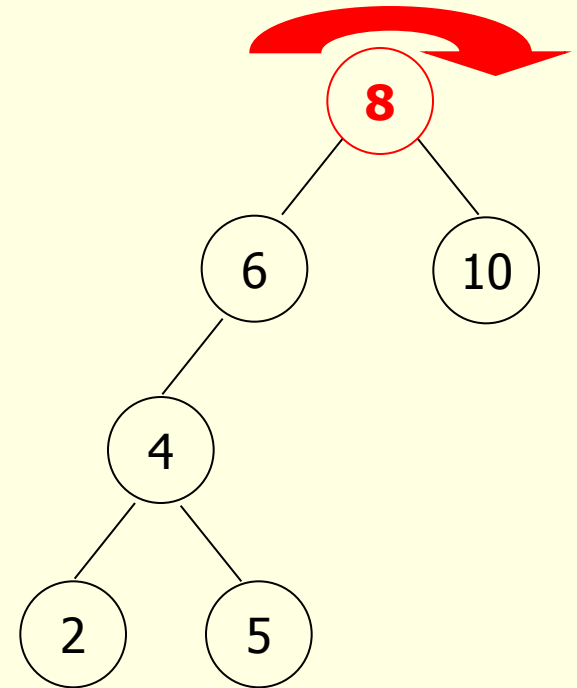
Árvore balanceada

# AVL: segundo caso

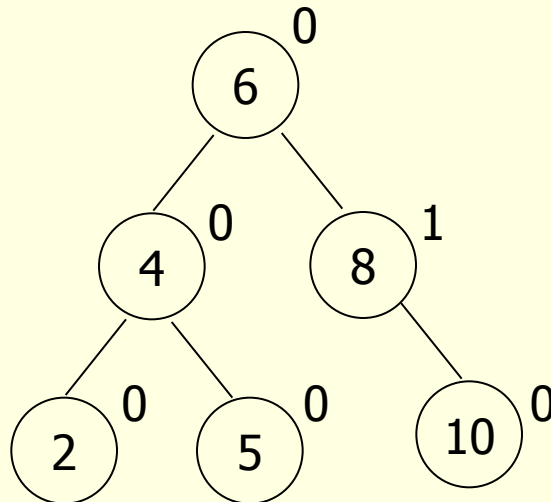


Rotação do nó filho para a esquerda (nó com FB=1)

Rotação dupla ED (Esquerda-Direita)



Rotação do nó pai para a direita (nó com FB=-2)



Árvore balanceada

# AVL: segundo caso

---

- Quando subárvores do pai e filho pendem para lados opostos
  - Rotação dupla
    - Primeiro, rotaciona-se o filho para o lado do desbalanceamento do pai
    - Em seguida, rotaciona-se o pai para o lado oposto do desbalanceamento
      - ED ou DE (com raciocínio inverso)
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

# AVL

---

- Assegura-se o rebalanceamento de todos os ancestrais de  $p$  e, portanto, o rebalanceamento da árvore toda



# AVL

---

- **Novo algoritmo de inserção**
  - A cada inserção, verifica-se o balanceamento da árvore
    - Se necessário, fazem-se as rotações de acordo com o caso (sinais iguais ou não)
  - Armazena-se uma variável de balanceamento em cada nó para indicar o FB

# AVL

---

## ■ Declaração

```
typedef int elem;
```

```
typedef struct no {  
    elem info;  
    struct no *esq, *dir;  
    int FB;  
} no;
```

```
typedef struct {  
    no *raiz;  
} AVL;
```

# AVL

---

- **Exercício**

- Inserir os elementos 10, 3, 2, 5, 7 e 6 em uma árvore AVL e balancear quando necessário

# AVL

---

- **Exercício**

- Inserir os elementos 1, 2, 3, ..., 10 em uma árvore AVL e balancear quando necessário

# AVL

---

- Os **percursos em-ordem** da árvore original e da balanceada **permanecem iguais**
- **Exercício**: mostre para os exemplos anteriores

# AVL

## ■ Rotação simples à esquerda

```
void EE(no **r) {  
    no *a = *r;  
    no *b = a->dir;  
    a->dir = b->esq;  
    b->esq = a;  
    a->fb = 0;  
    b->fb = 0;  
    *r = b;  
}
```

# AVL

## ■ Rotação simples à direita

```
void DD(no **r) {  
    no *b = *r;  
    no *a = b->esq;  
    b->esq = a->dir;  
    a->dir = b;  
    a->fb = 0;  
    b->fb = 0;  
    *r = a;  
}
```

# AVL

## ■ Rotação dupla esquerda-direita

```
void ED(no **r) {
    no *c = *r;
    no *a = c->esq;
    no *b = a->dir;
    c->esq = b->dir;
    a->dir = b->esq;
    b->esq = a;
    b->dir = c;
    switch (b->fb) {
        case -1:
            a->fb = 0;
            c->fb = 1;
            break;
    }
}
```

Continua →



# AVL

## ■ Rotação dupla esquerda-direita

```
    case 0:
        a->fb = 0;
        c->fb = 0;
        break;
    case 1:
        a->fb = -1;
        c->fb = 0;
        break;
}
b->fb = 0;
*r = b;
}
```

# AVL

## ■ Rotação dupla direita-esquerda

```
void DE(no **r) {
    no *a = *r;
    no *c = a->dir;
    no *b = c->esq;
    c->esq = b->dir;
    a->dir = b->esq;
    b->esq = a;
    b->dir = c;
    switch (b->fb) {
        case -1:
            a->fb = 0;
            c->fb = 1;
            break;
    }
}
```

Continua →

# AVL

## ■ Rotação dupla direita-esquerda

```
    case 0:
        a->fb = 0;
        c->fb = 0;
        break;
    case 1:
        a->fb = -1;
        c->fb = 0;
        break;
}
b->fb = 0;
*r = b;
}
```

# AVL

## ■ Inserção

```
int aux_inserere(no** p, int x, int *cresceu) {
    if (*p == NULL) {
        *p = (no*) malloc(sizeof(no));
        (*p)->info = x;
        (*p)->fb = 0;
        (*p)->esq = NULL;
        (*p)->dir = NULL;
        *cresceu = 1;
        return 1;
    } else if (x == (*p)->info)
        return 0;
```

Continua →

# AVL

## ■ Inserção

```
else if (x < (*p)->info) {
    if (aux_insere(&(*p)->esq, x, cresceu)) {
        if (*cresceu) {
            switch ((*p)->fb) {
                case -1:
                    if ((*p)->esq->fb == -1)
                        DD(p);
                    else
                        ED(p);
                    *cresceu = 0;
                    break;
            }
        }
    }
}
```

Continua →

# AVL

## ■ Inserção

```
        case 0:
            (*p)->fb = -1;
            *cresceu = 1;
            break;
        case 1:
            (*p)->fb = 0;
            *cresceu = 0;
            break;
    }
}
return 1;
} else
return 0;
```

Continua →

# AVL

## ■ Inserção

```
} else {
    if (aux_insere(&(*p)->dir, x, cresceu)) {
        if (*cresceu) {
            switch ((*p)->fb) {
                case -1:
                    (*p)->fb = 0;
                    *cresceu = 0;
                    break;
                case 0:
                    (*p)->fb = 1;
                    *cresceu = 1;
                    break;
```

Continua →

# AVL

## ■ Inserção

```
        case 1:
            if ((*p)->dir->fb == 1)
                EE(p);
            else
                DE(p);
            *cresceu = 0;
            break;
        }
    }
    return 1;
} else
    return 0;
}
}
```



# AVL

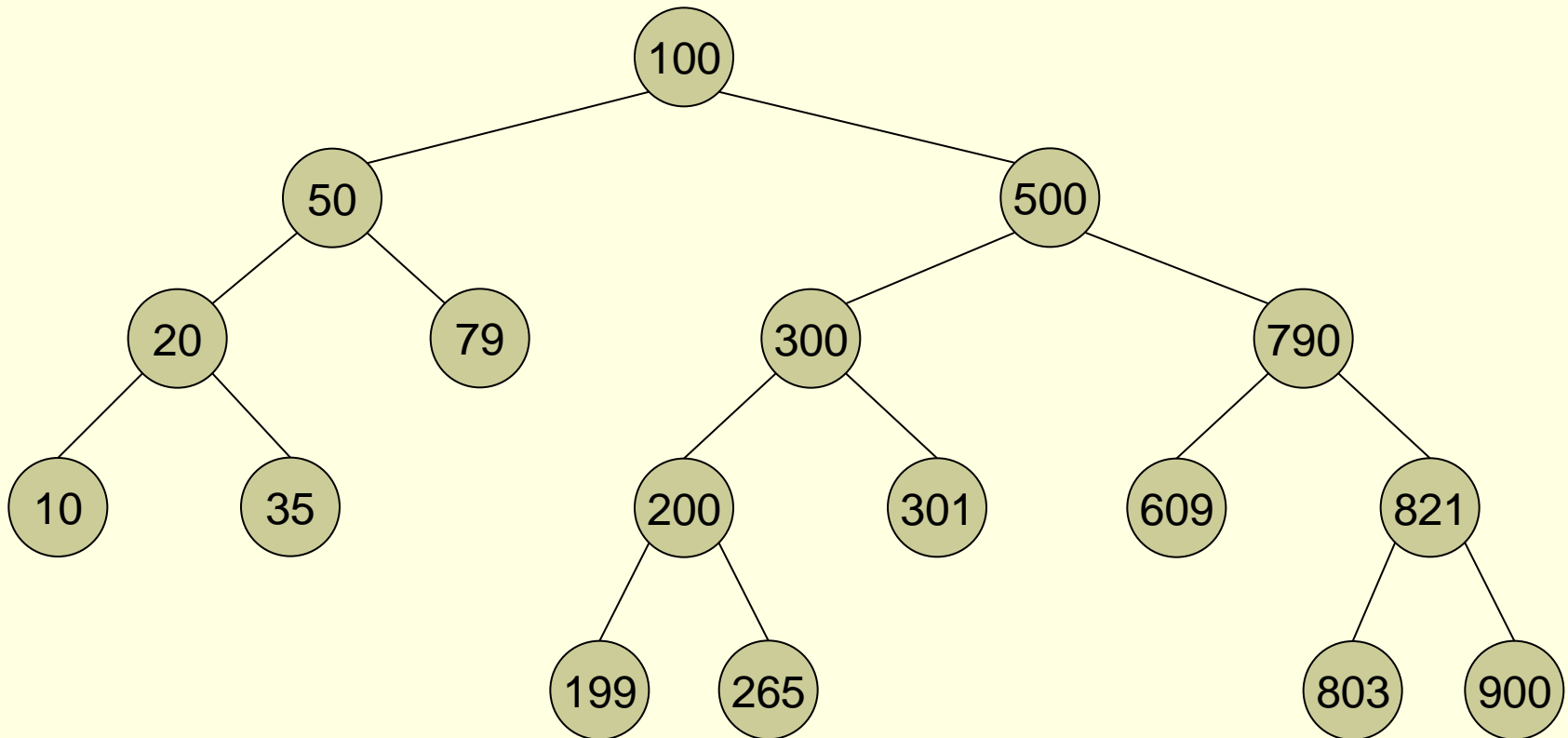
---

## ■ Inserção

```
int inserir(no **p, int x) {  
    int cresceu;  
    return aux_insere(p, x, &cresceu);  
}
```

# AVL

- **Exercício:** teste a rotina de inserção inserindo alguns elementos na árvore abaixo



# AVL

---

## ■ Exercício

- Pense no procedimento de remoção de um elemento da AVL e responda
  - Quais são os casos de desbalanceamento?
  - Esboce um método de remoção que balanceie a árvore se necessário

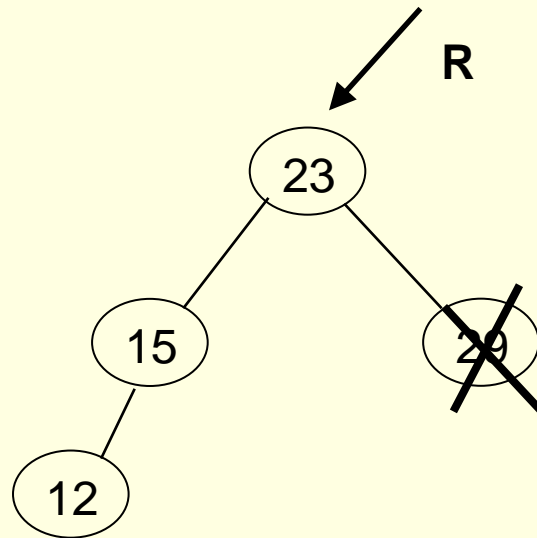
# AVL

---

- Como fazer a remoção de elementos da AVL?

# AVL: remoção

## ■ Exemplos

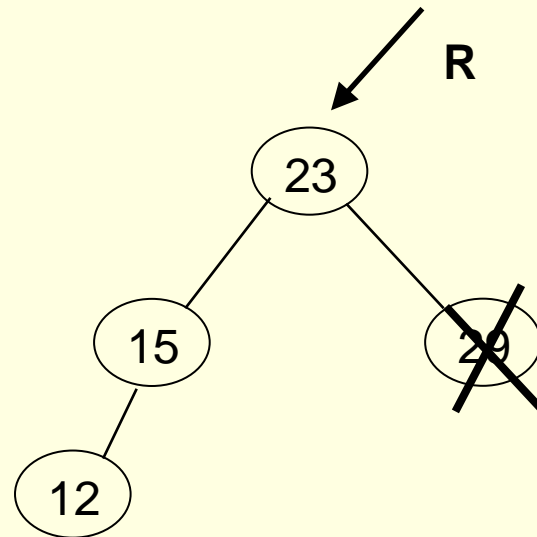


remoção de 29

Como balancear?

# AVL: remoção

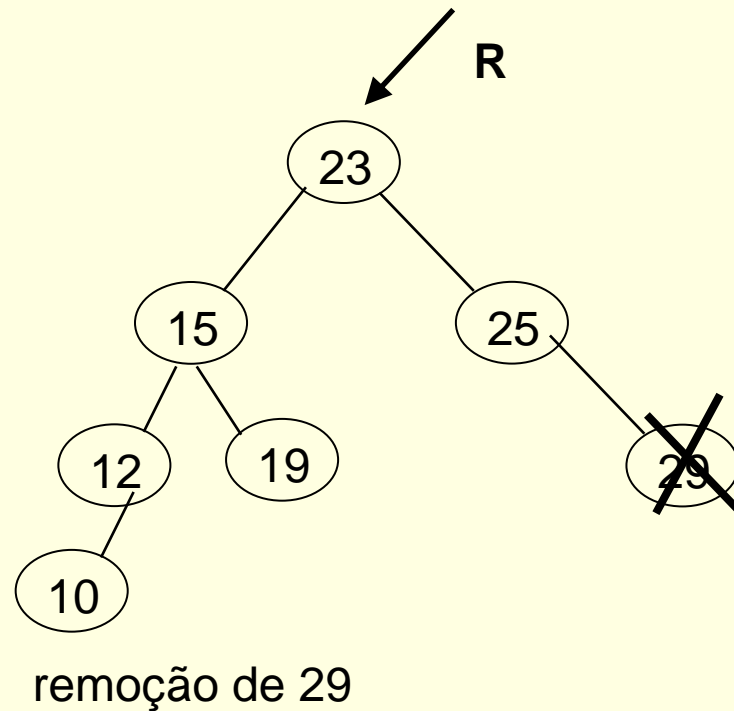
## ■ Exemplos



remoção de 29 = inserção de 12

# AVL: remoção

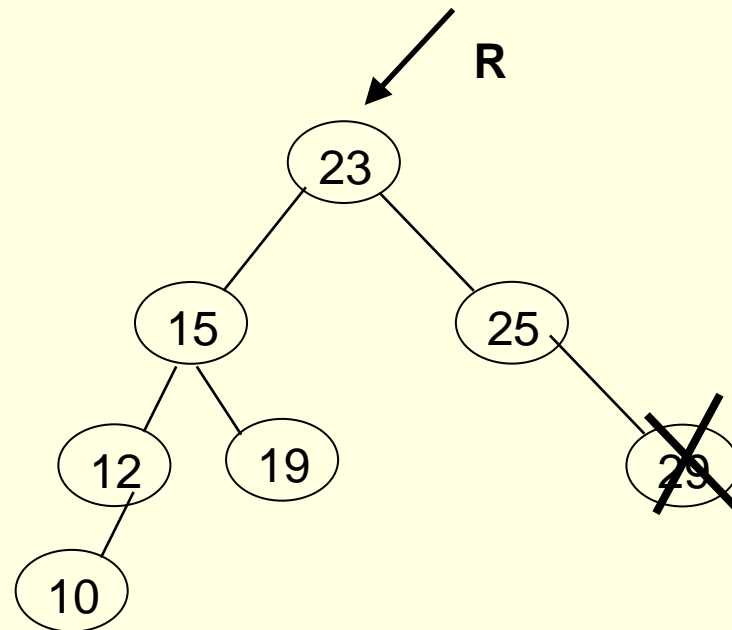
## ■ Exemplos



Como balancear?

# AVL: remoção

## ■ Exemplos

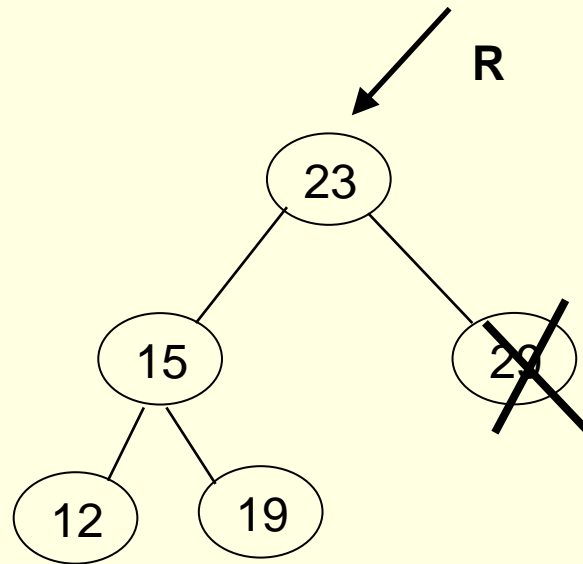


remoção de 29 = inserção de 10



# AVL: remoção

## ■ Exemplos

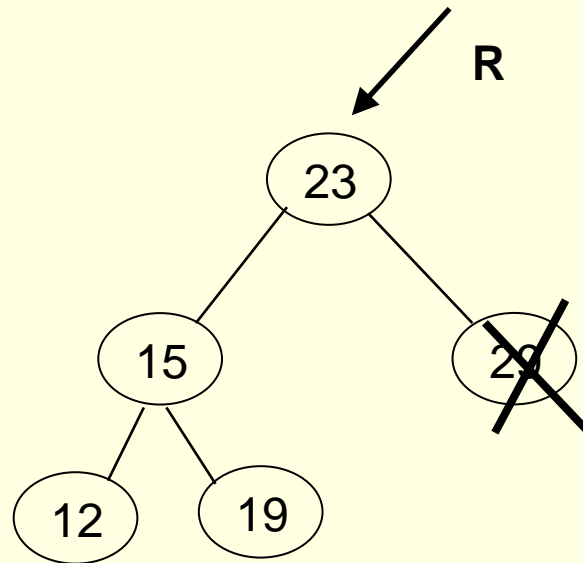


remoção de 29

Como balancear?

# AVL: remoção

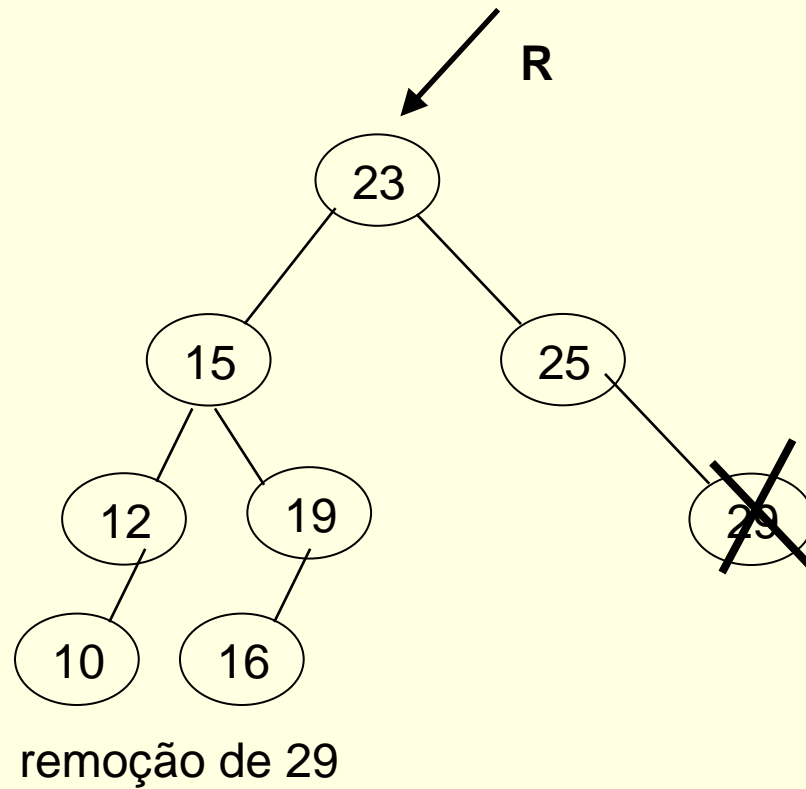
## ■ Exemplos



remoção de 29 = inserção de 12 ou 19  
(desbalanceamento que não aconteceria na inserção)

# AVL: remoção

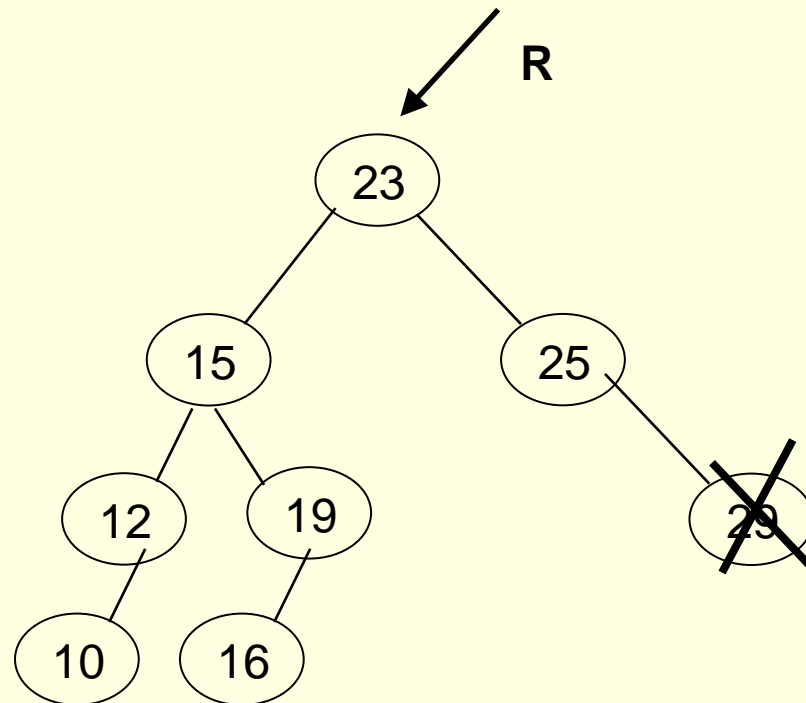
## ■ Exemplos



Como balancear?

# AVL: remoção

## ■ Exemplos



remoção de 29 = inserção de 10 ou 16  
(não aconteceria na inserção)

# AVL: remoção

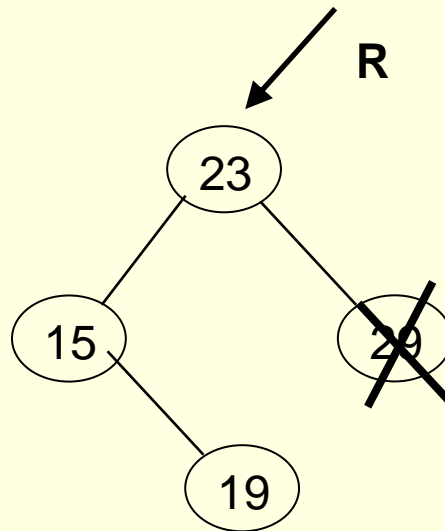
---

## ■ Primeiro caso

- Rotação simples em R com filho com fator de balanceamento de mesmo sinal ou zero
  - Se FB negativo, rotaciona-se para a direita; caso contrário, para a esquerda

# AVL: remoção

## ■ Exemplos

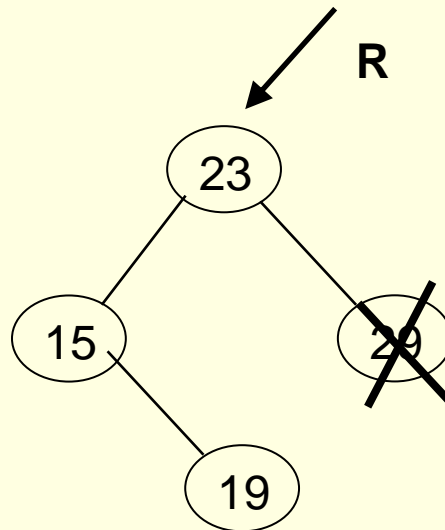


remoção de 29

Como balancear?

# AVL: remoção

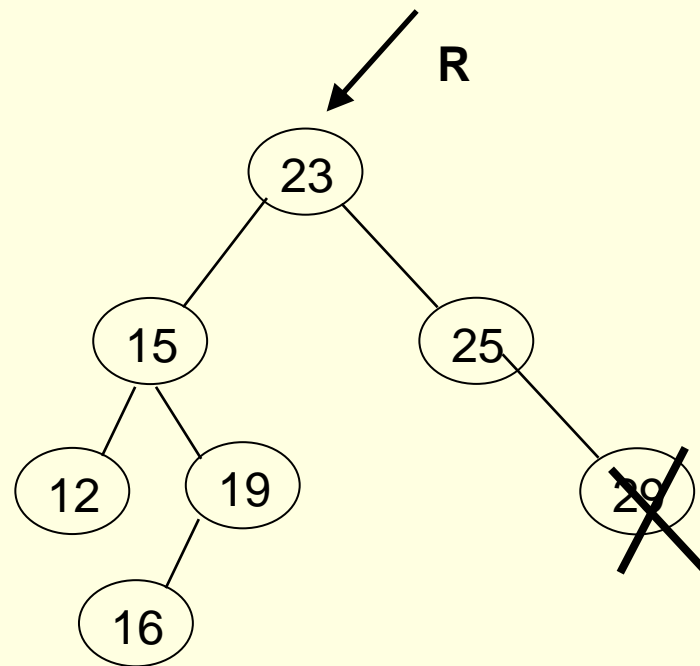
## ■ Exemplos



remoção de 29 = inserção de 19

# AVL: remoção

## ■ Exemplos



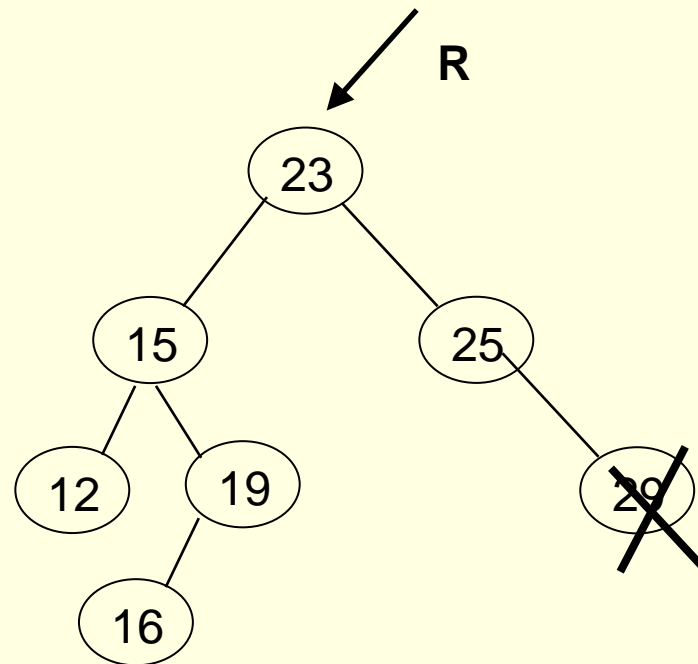
remoção de 29

Como balancear?



# AVL: remoção

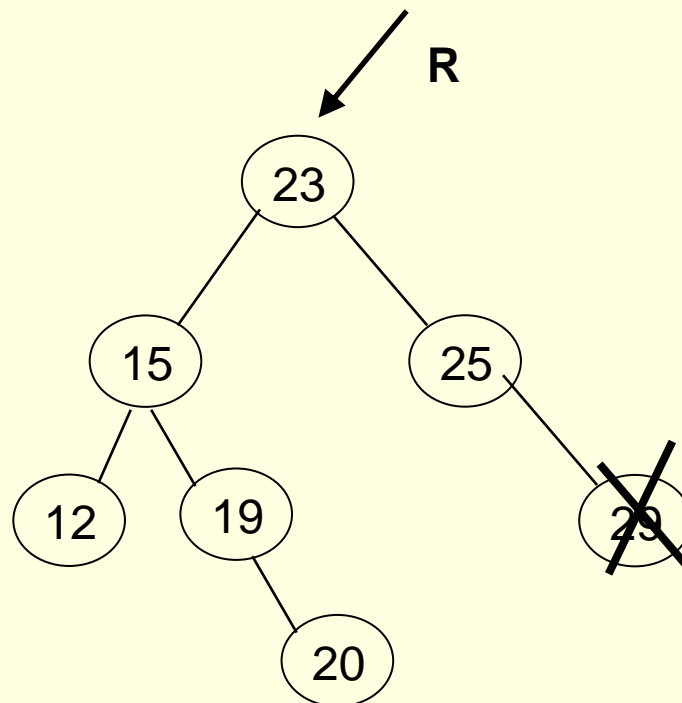
## ■ Exemplos



remoção de 29 = inserção de 16

# AVL: remoção

## ■ Exemplos

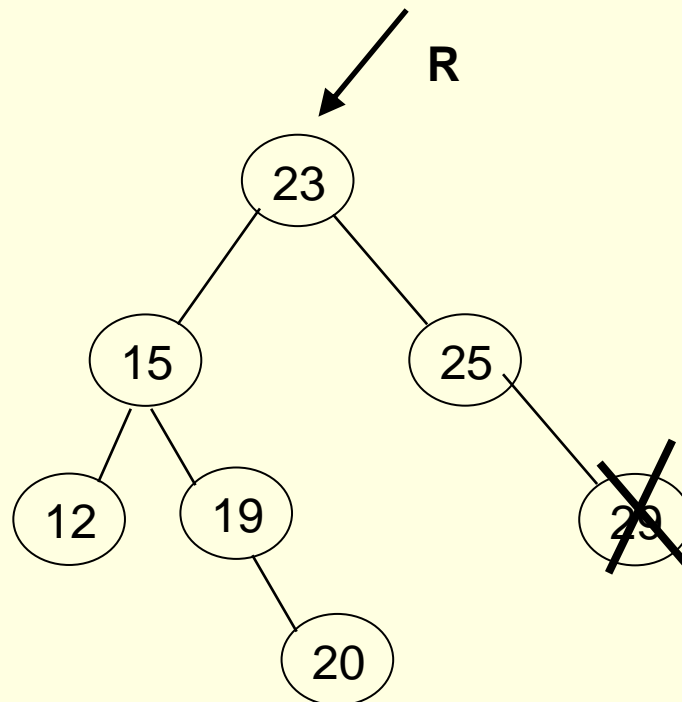


remoção de 29

Como balancear?

# AVL: remoção

## ■ Exemplos

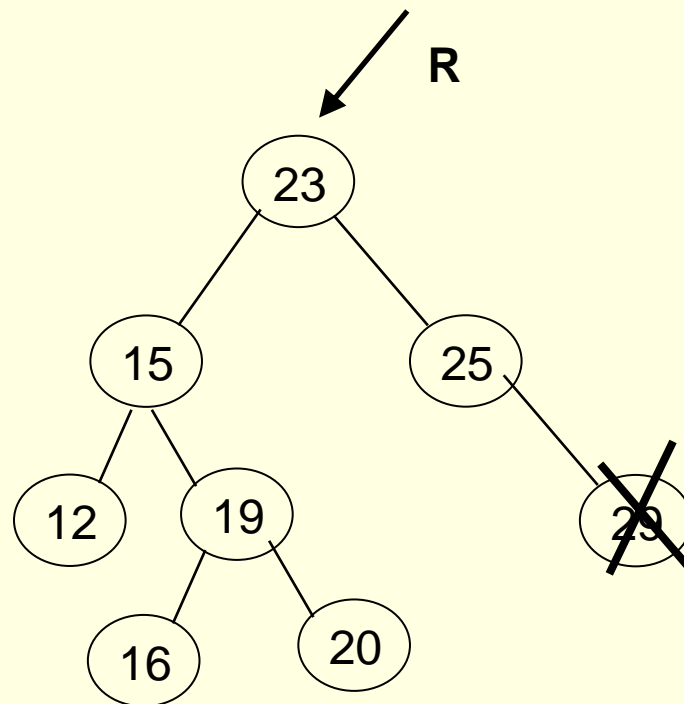


remoção de 29 = inserção de 20



# AVL: remoção

## ■ Exemplos



remoção de 29 = inserção de 16 ou 20  
(não aconteceria na inserção)

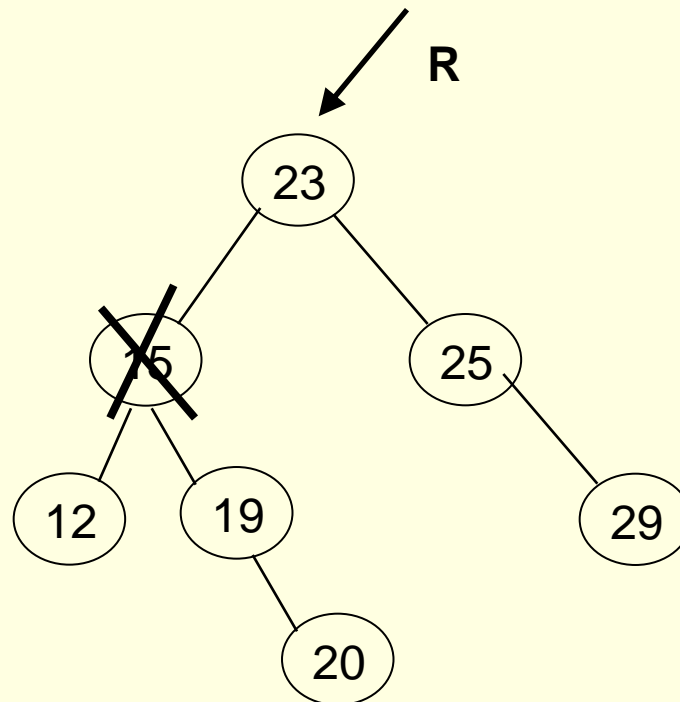
# AVL: remoção

---

- Segundo caso
  - Rotação dupla quando R e seu filho têm 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

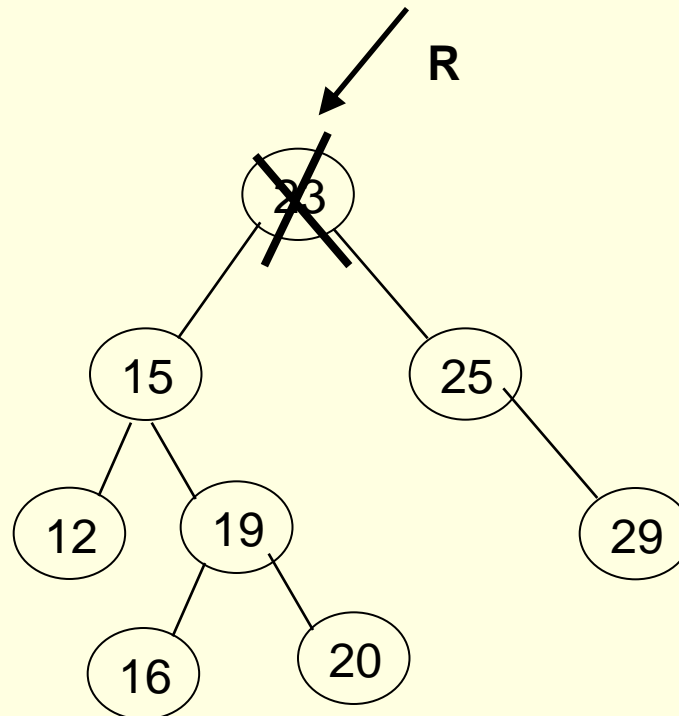
# AVL: remoção

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



# AVL: remoção

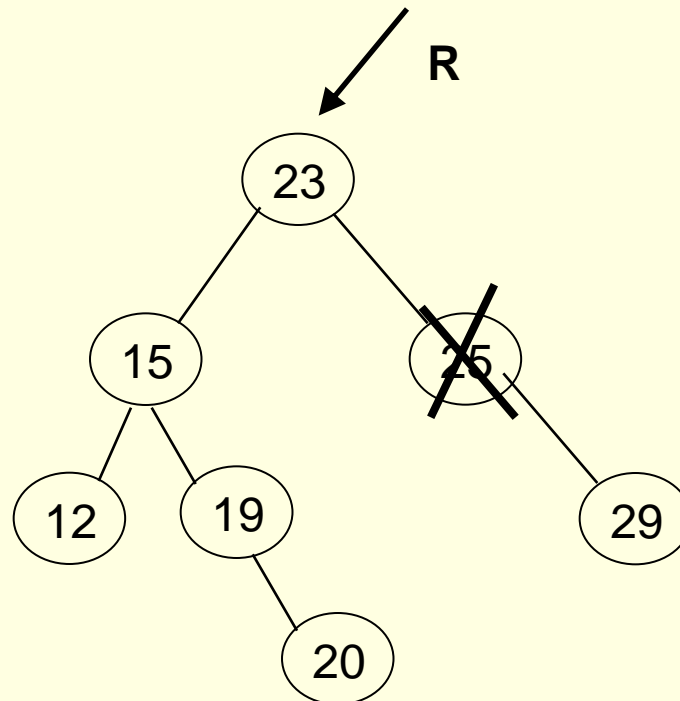
- Questão: como remover um nó intermediário em vez de um nó folha?
  - É necessário balancear?





# AVL: remoção

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



# AVL: remoção

---

## ■ Exercício

- Esquematizar cada caso
  - Que ponteiros alterar, como atualizar os fatores de balanceamento?
- Esboçar sub-rotina de remoção de elemento de uma AVL

# Créditos

---

- *Material gentilmente cedido pelo Prof. Thiago A. S. Pardo*