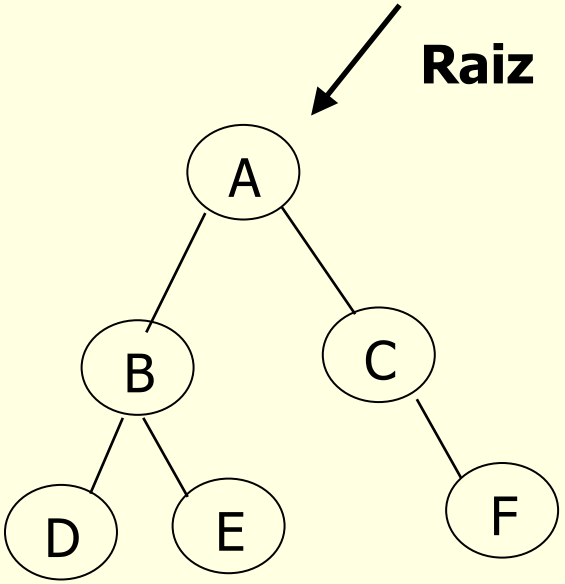
Introdução a AVL: teoria e prática

22/11, 25/11 e 30/11

Árvores binárias de busca (ABB)

- Árvores de grau 2, isto é, cada nó tem dois filhos, no máximo



Terminologia:
- filho esquerdo
- filho direito
- informação

ABB

- Também chamadas “árvores de pesquisa” ou “árvores ordenadas”

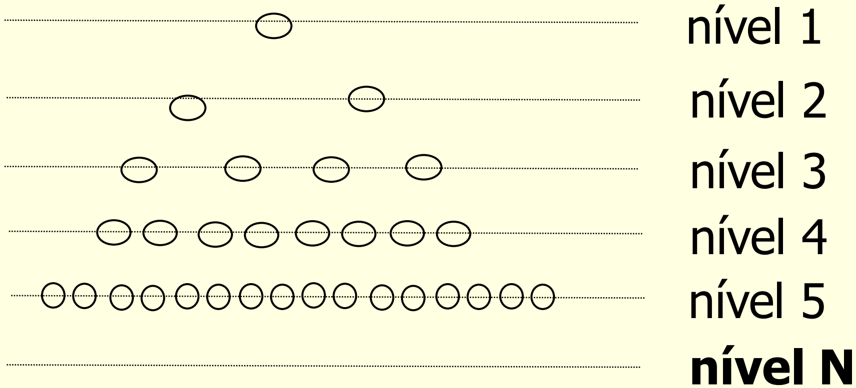
- Definição
  - Uma árvore binária com raiz R é uma ABB se:
    - a chave (informação) de cada nó da subárvore esquerda de R é menor do que a chave do nó R (em ordem alfabética, por exemplo)
    - a chave de cada nó da subárvore direita de R é maior do que a chave do nó R
    - as subárvores esquerda e direita também são ABBs

ABB

- Muito boa para busca
  - Em uma árvore de altura A, visitam-se, no máximo, A nós
  - Grande quantidade de informação em relativamente poucos níveis

ABB

- Quantidade de informação



Nível	**Quantos cabem**
1	1
2	3
3	7
4	15
...	...
N	$\mathbf{2^N - 1}$
10	1.024
13	8.192
16	65.536
18	262.144
20	1 milhão
30	1 bilhão
	...

ABB

- Vantagens
  - Se nós espalhados uniformemente, consulta rápida para grande quantidade de dados
    - Divide-se o espaço de busca restante em dois em cada passo da busca
    - O(Log N)

ABB

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

ABB

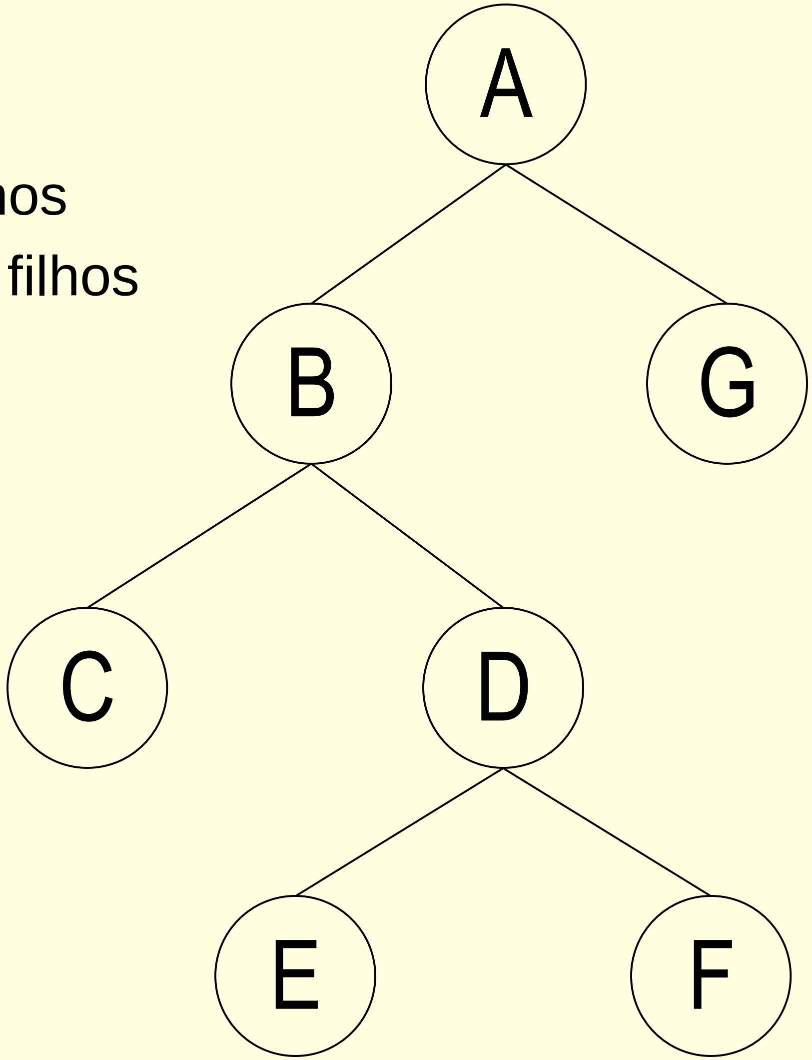
- O desbalanceamento da árvore pode tornar a busca tão ineficiente quanto a busca seqüencial (no pior caso)
  - O(N)

- Solução?

Balanceamento da árvore quando necessário!

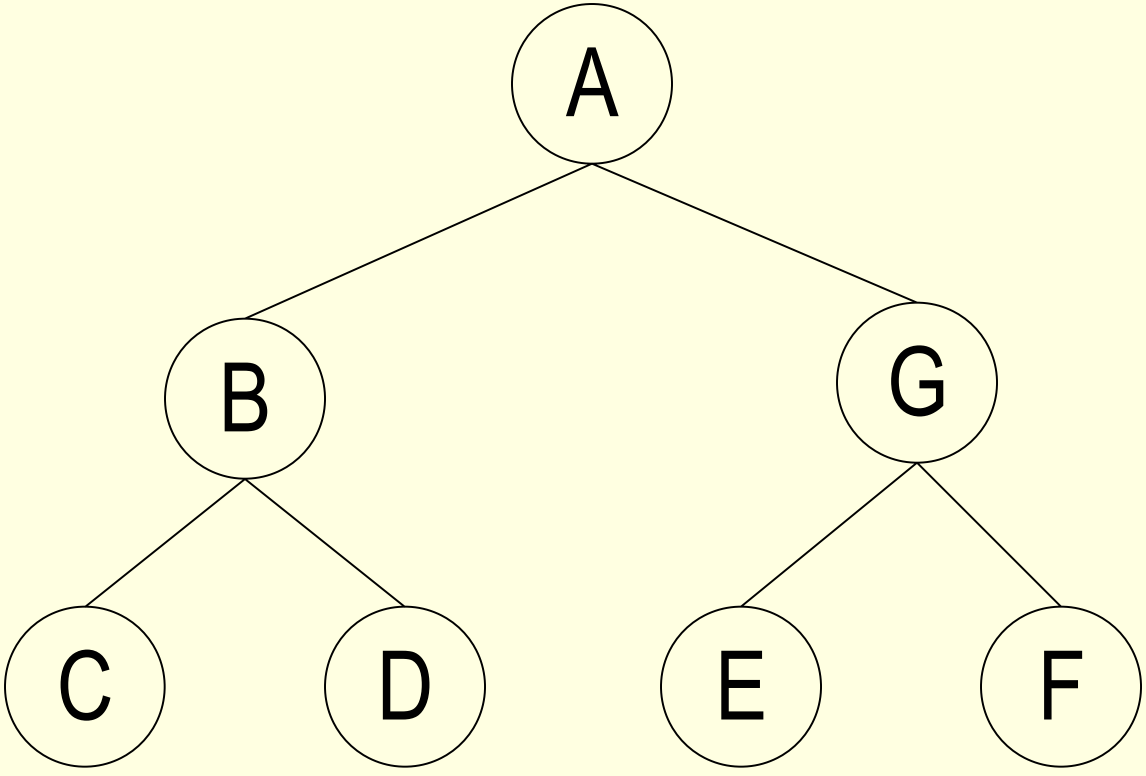
Conceitos

- Árvore estritamente binária
  - Os nós tem 0 ou 2 filhos
    - Todo nó interno tem 2 filhos
    - Somente as folhas têm 0 filhos



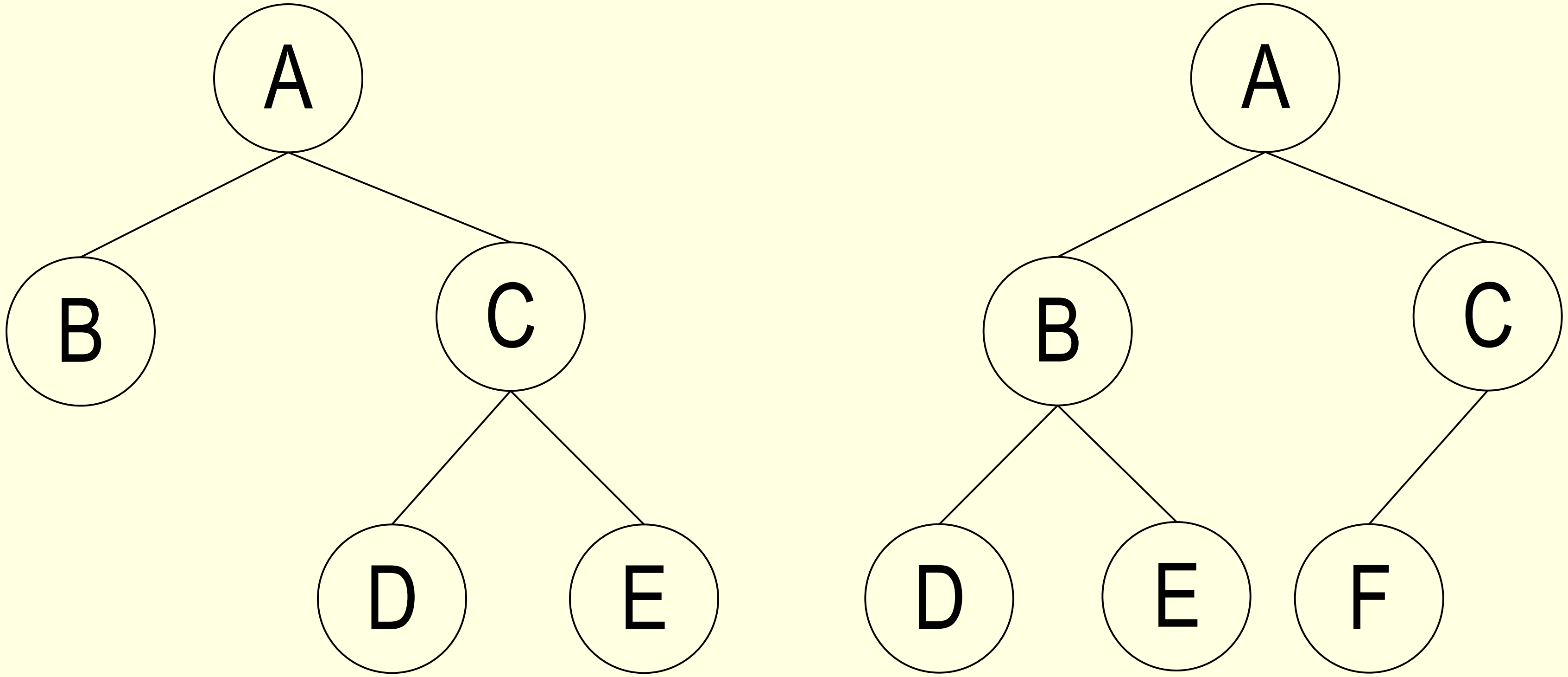
Conceitos

- Árvore binária <u>completa</u> (ou cheia)
  - Árvore estritamente binária
  - Todos os nós folha no mesmo nível



Conceitos

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



Conceitos

- Uma árvore binária <u>perfeitamente balanceada</u> é aquela cujo número de nós de suas subárvores esquerda e direita diferem em 1, no máximo
  - Toda árvore binária perfeitamente balanceada é balanceada
    - Vale o inverso?

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-Velskki e E.M. Landis
    - Métodos de inserção e remoção de elementos da árvore de forma que ela fique balanceada

Por que não se exige que seja perfeitamente balanceada?
Custo alto...certos casos exigem movimentar a árvore inteira

Métodos de Balanceamento

- Há duas categorias: dinâmico e global (ou estático).

- O rebalanceamento **dinâmico** mantém a árvore balanceada toda vez que é um nó é inserido ou removido.
  - AVL é o melhor exemplo

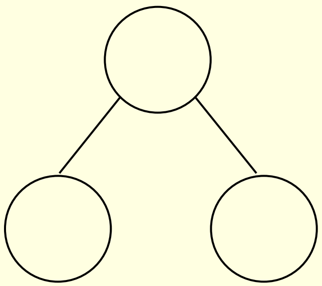
- O **global** permite a árvore crescer sem limites e somente faz o balanceamento quando tal necessidade é acionada, externamente.
  - Há vários métodos. O indicado para o 3o trabalho é o de Chang & Iyengar, de 1984.

  - Códigos destes rebalanceamentos são mostrados em:

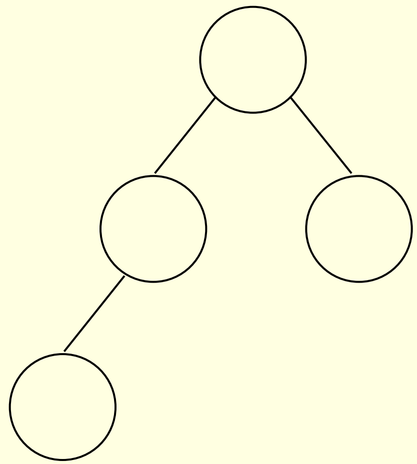
Binary Search Tree Balancing Methods: A Critical Study (http://paper.ijcsns.org/07_book/200708/20070834.pdf)

AVL: quem é e quem não é?

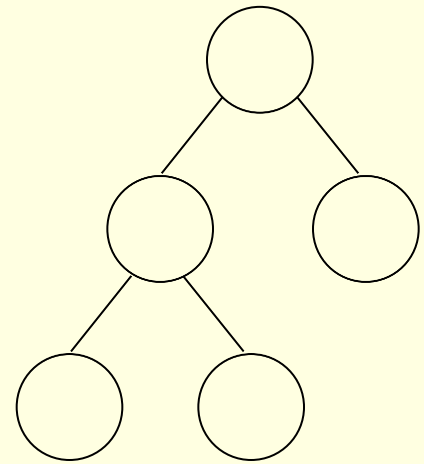
(a)



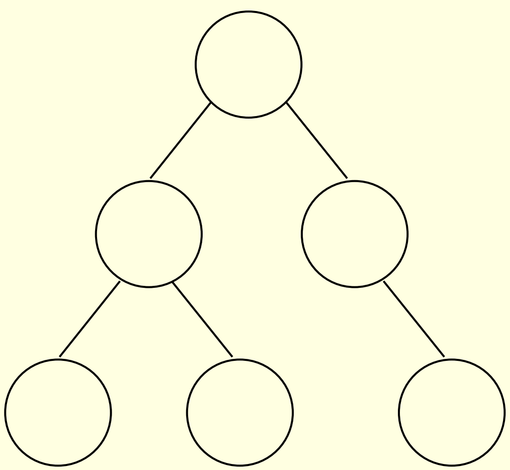
(b)



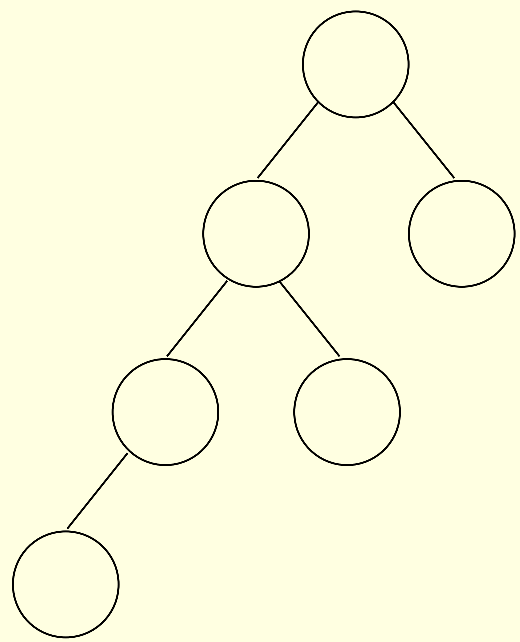
(c)



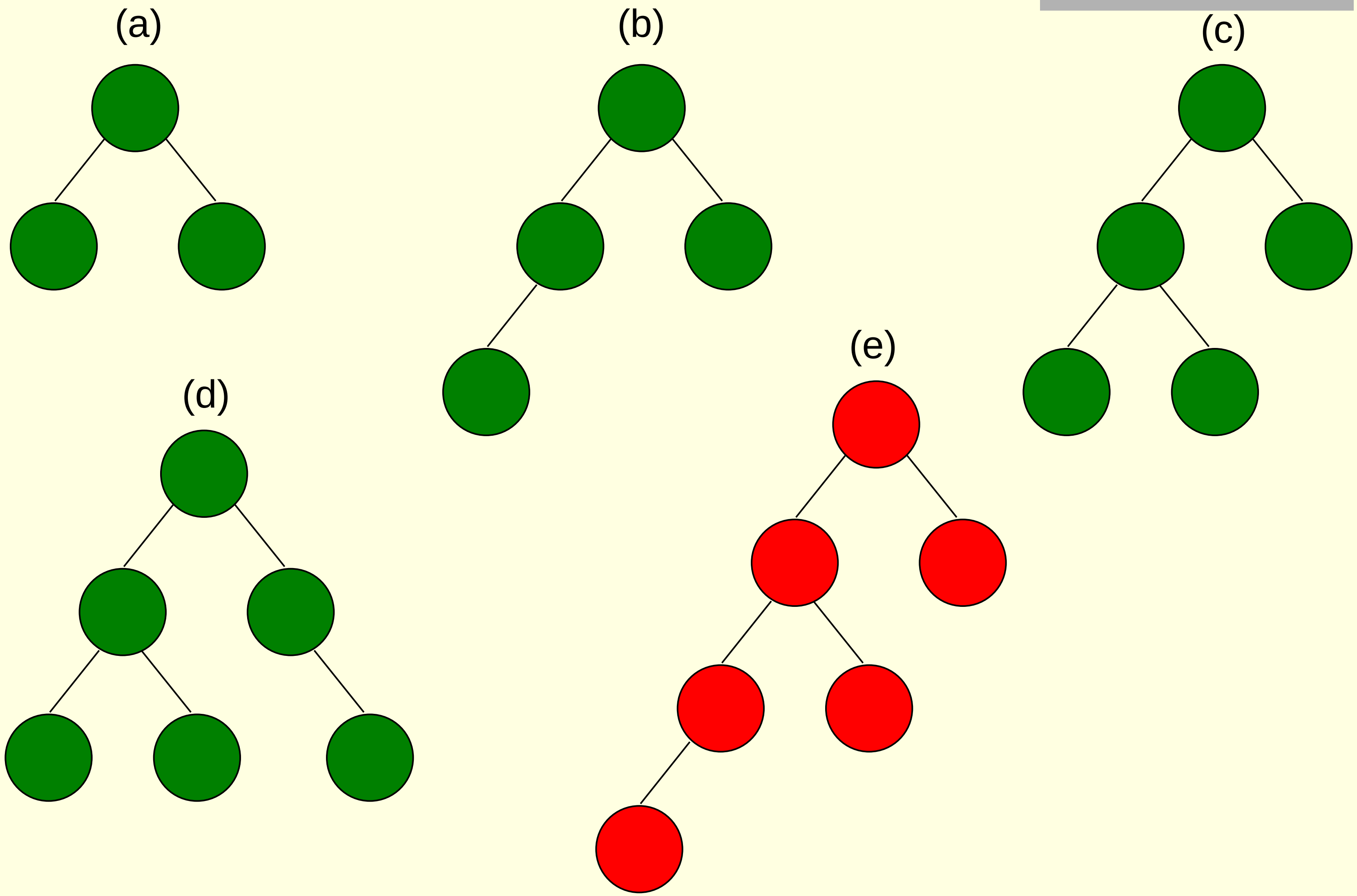
(d)



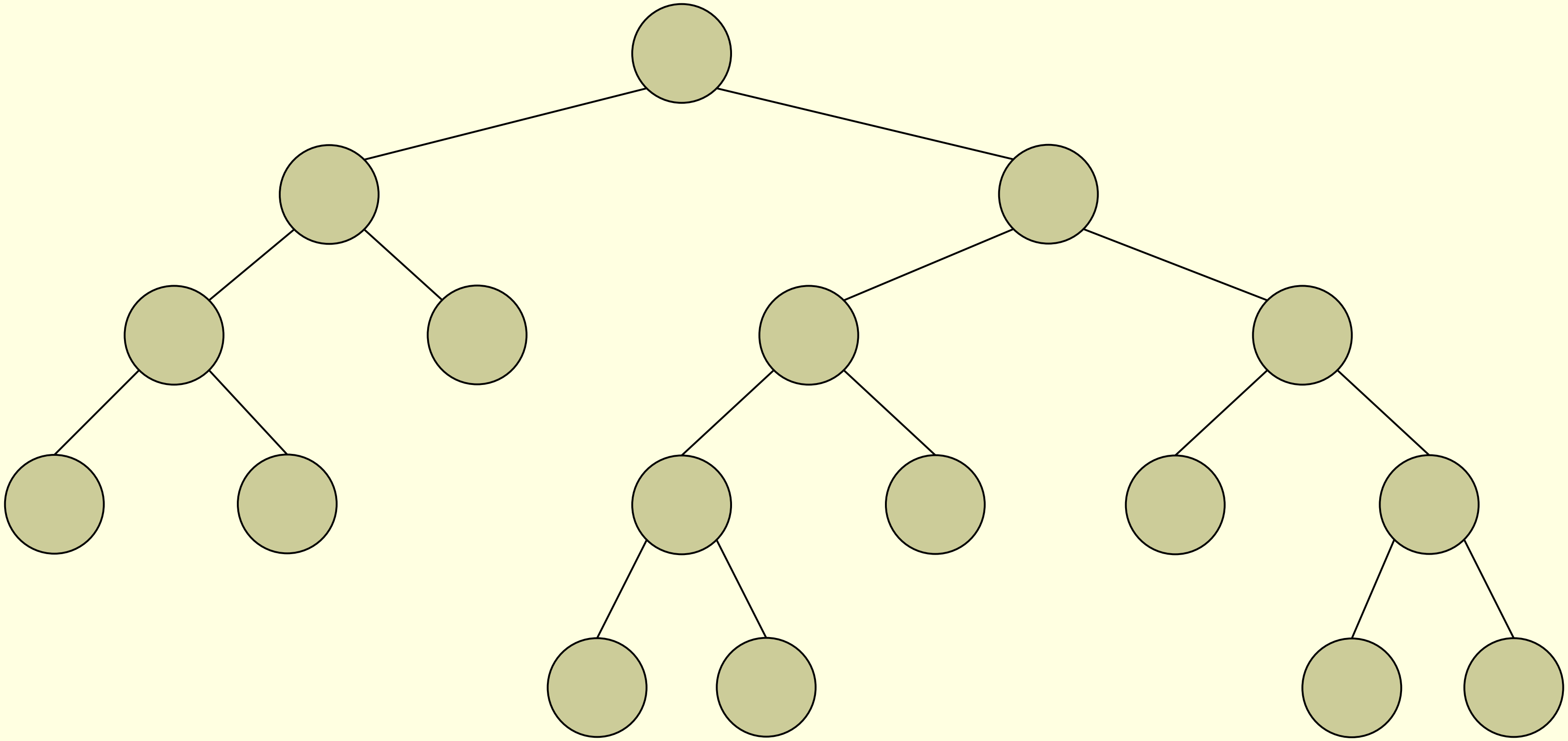
(e)



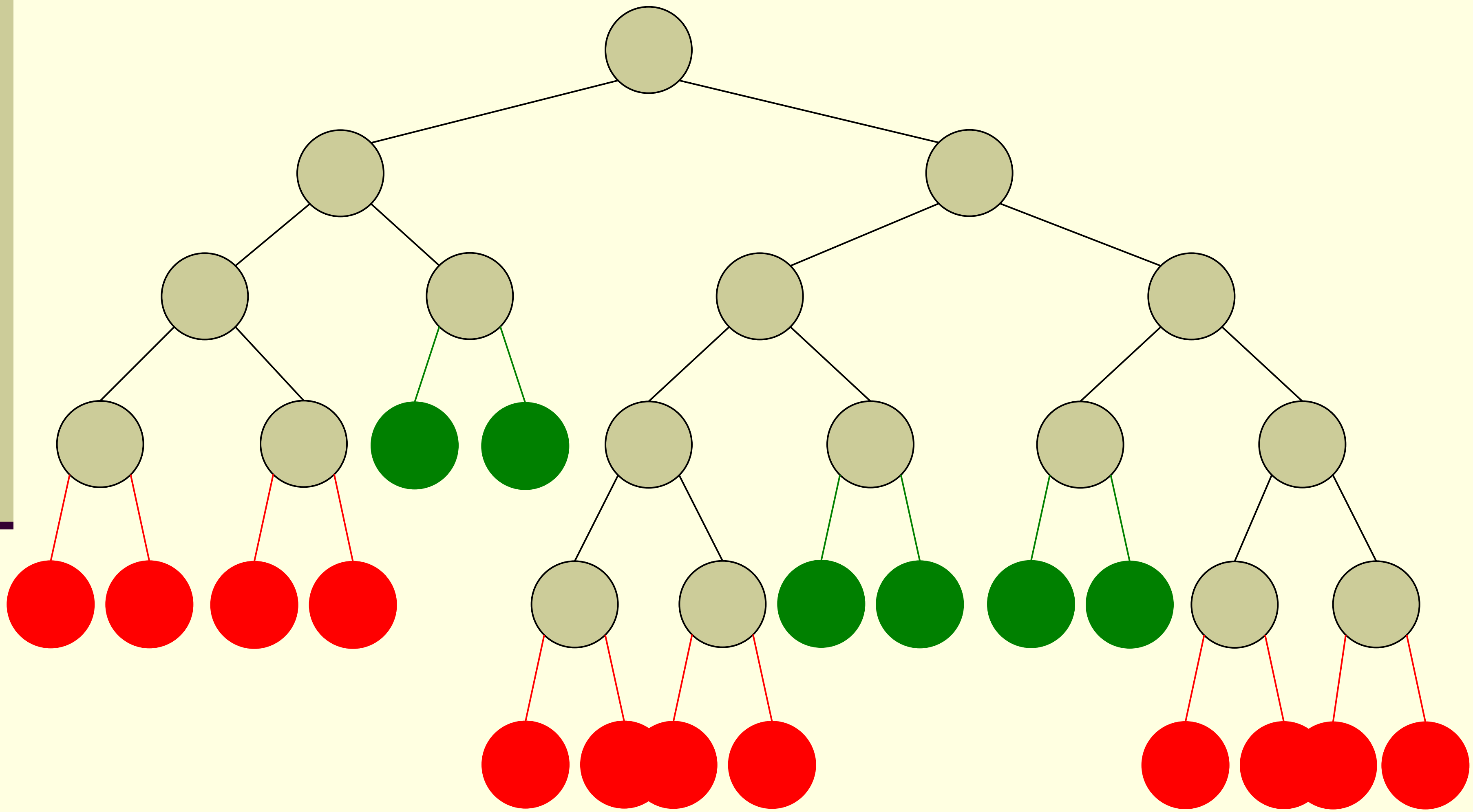
AVL: quem é e quem não é?



Pergunta: a árvore abaixo é AVL?



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



AVL: prática (valendo 0,5 na última prova)

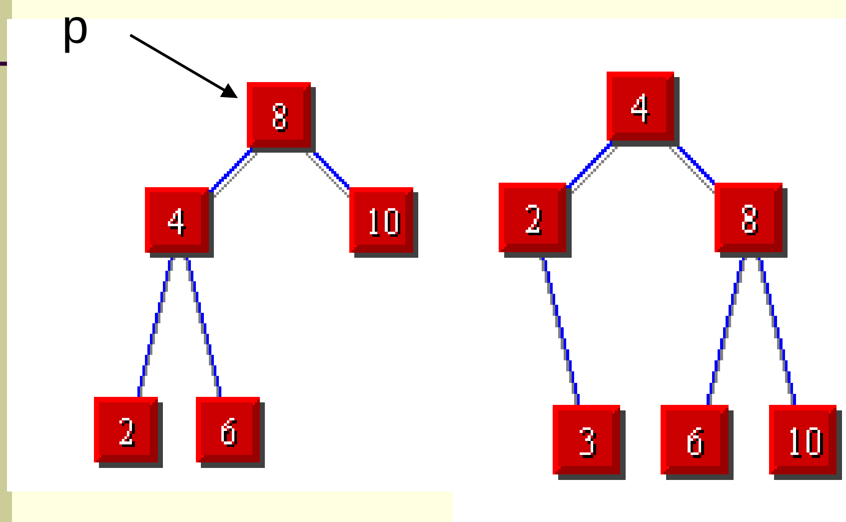
- Formar grupos:

  1. Simular exemplos de inserção em uma AVL e identificar os casos em que ocorre desbalanceamento

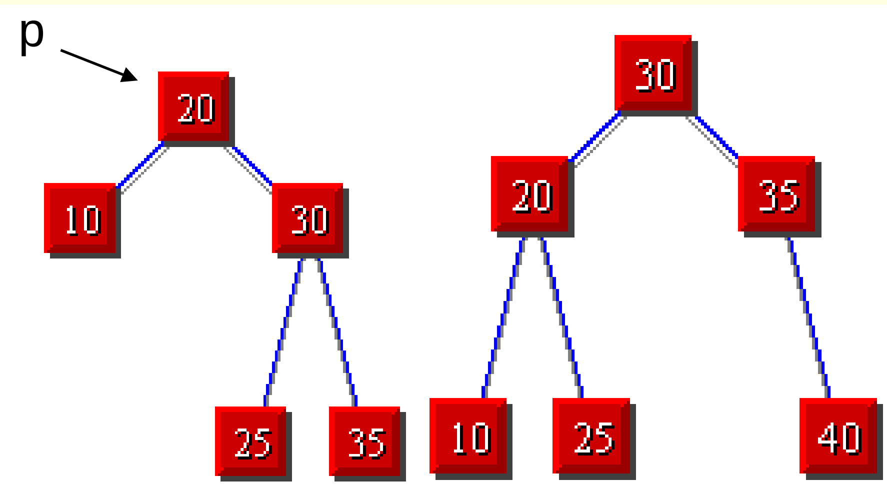
  2. Com base nos casos anteriores, esboçar um método de inserção de elementos na AVL que, quando necessário, rearranje os elementos da árvore para que ela continue balanceada
     - Explore formas de rearranjar a árvore: qualquer uma é bem-vinda

- Dica: pense em como representar a necessidade de rebalanceamento!!!

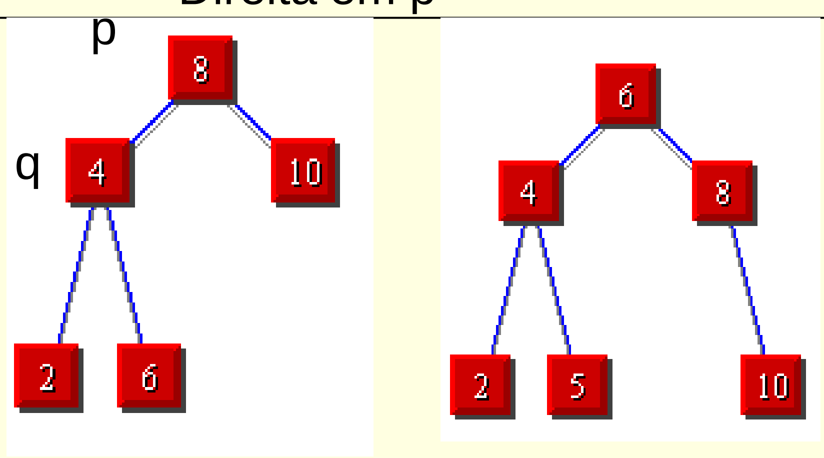
4 casos



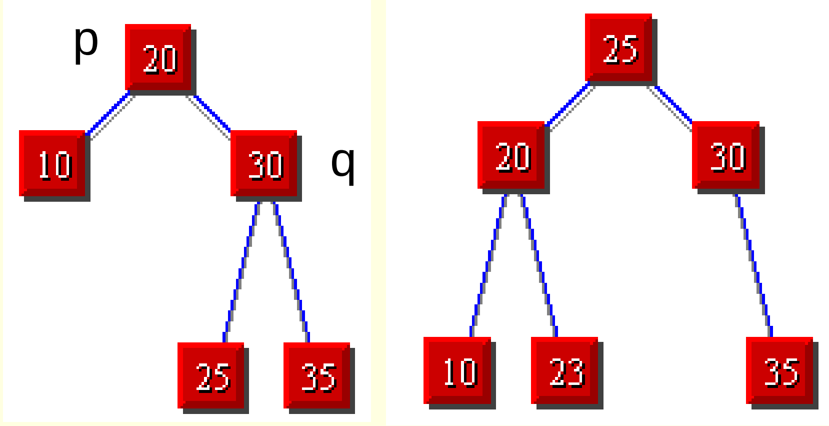
Inserção de 3 - Rotação
Direita em p



Inserção de 40 - Rotação
Esquerda em p



Inserção de 5 – Rotação
Esq em q e Dir em p



Inserção de 23 – Rotação
Dir em q e Esq em p

AVL

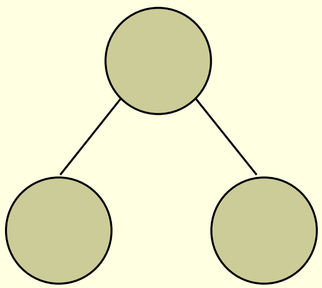
- Como é que se sabe quando é necessário balancear a árvore?
  - Se a diferença de altura das subárvores deve ser 1, no máximo, então temos que procurar diferenças de altura maior do que isso
  - Possível solução: cada nó pode manter a diferença de altura de suas subárvores
    - Convencionalmente chamada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

AVL

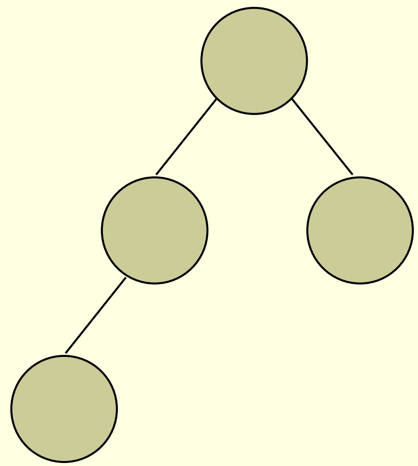
- Fatores de balanceamento dos nós
  - Altura da subárvore direita menos altura da subárvore esquerda
    - Hd - He
  - Atualizados sempre que a árvore é alterada (elemento é inserido ou removido)
  - Quando um fator é 0, 1 ou -1, a árvore está balanceada
  - Quando um fator se torna 2 ou -2, a árvore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

AVL: quem é e quem não é

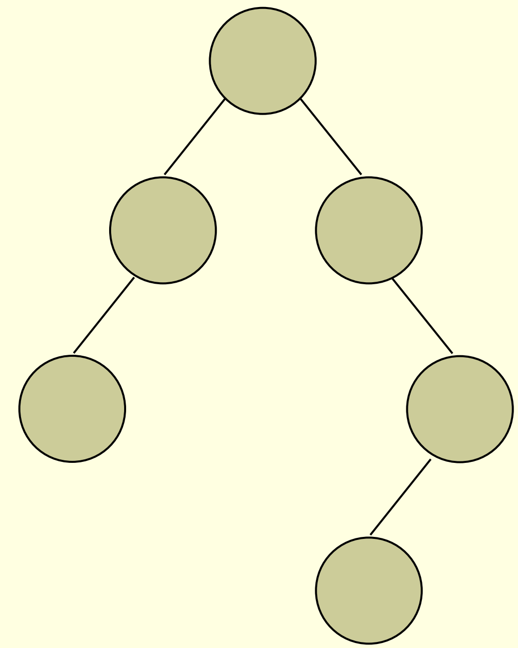
(a)



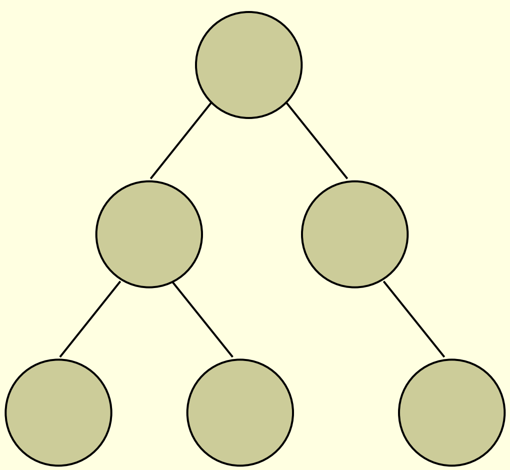
(b)



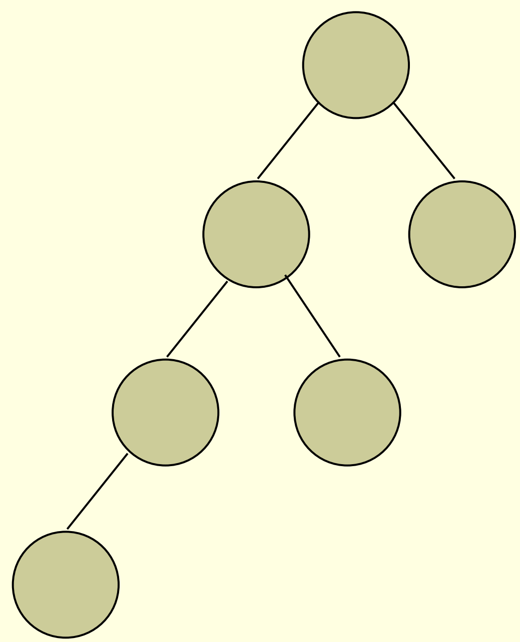
(c)



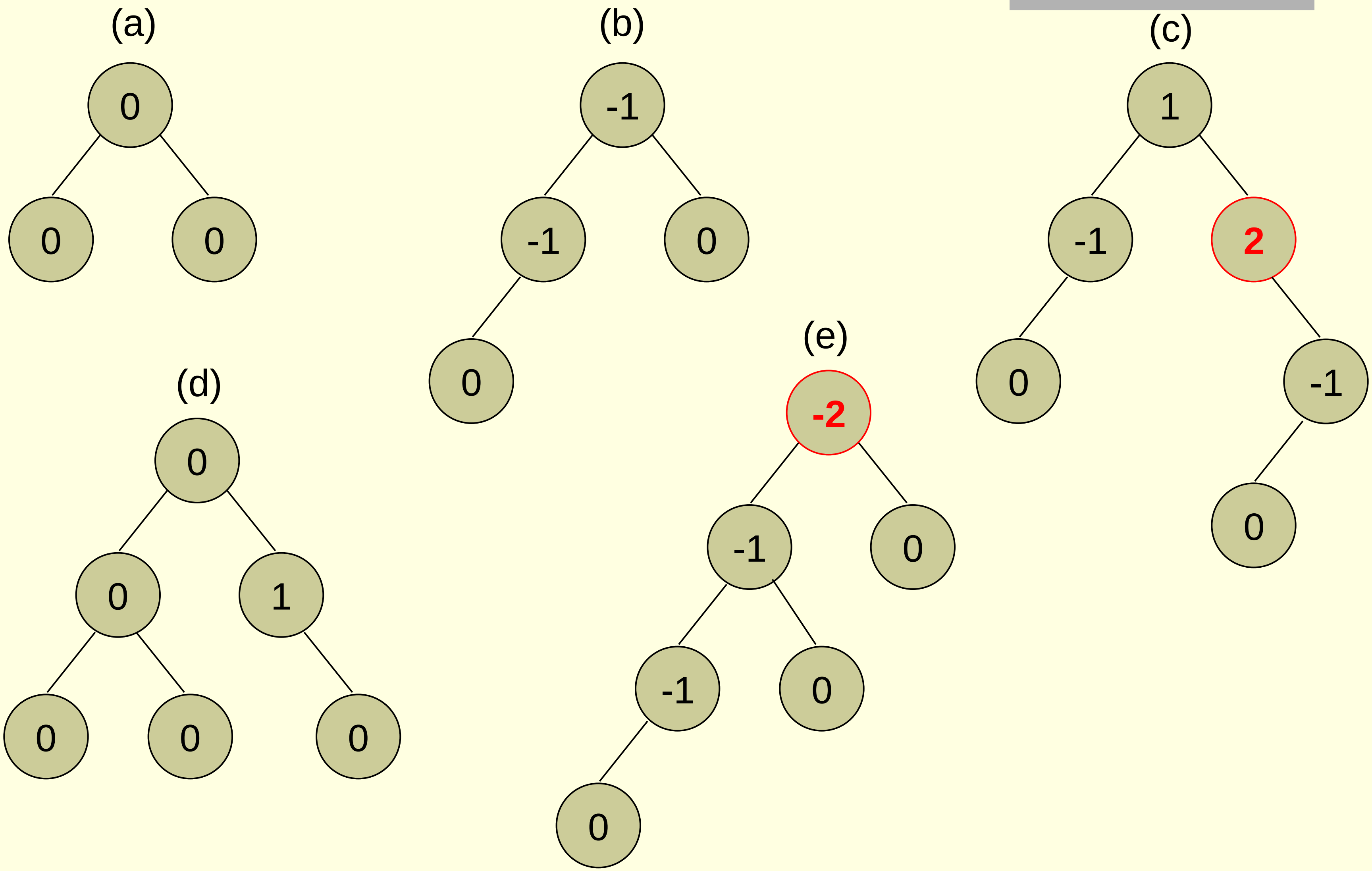
(d)



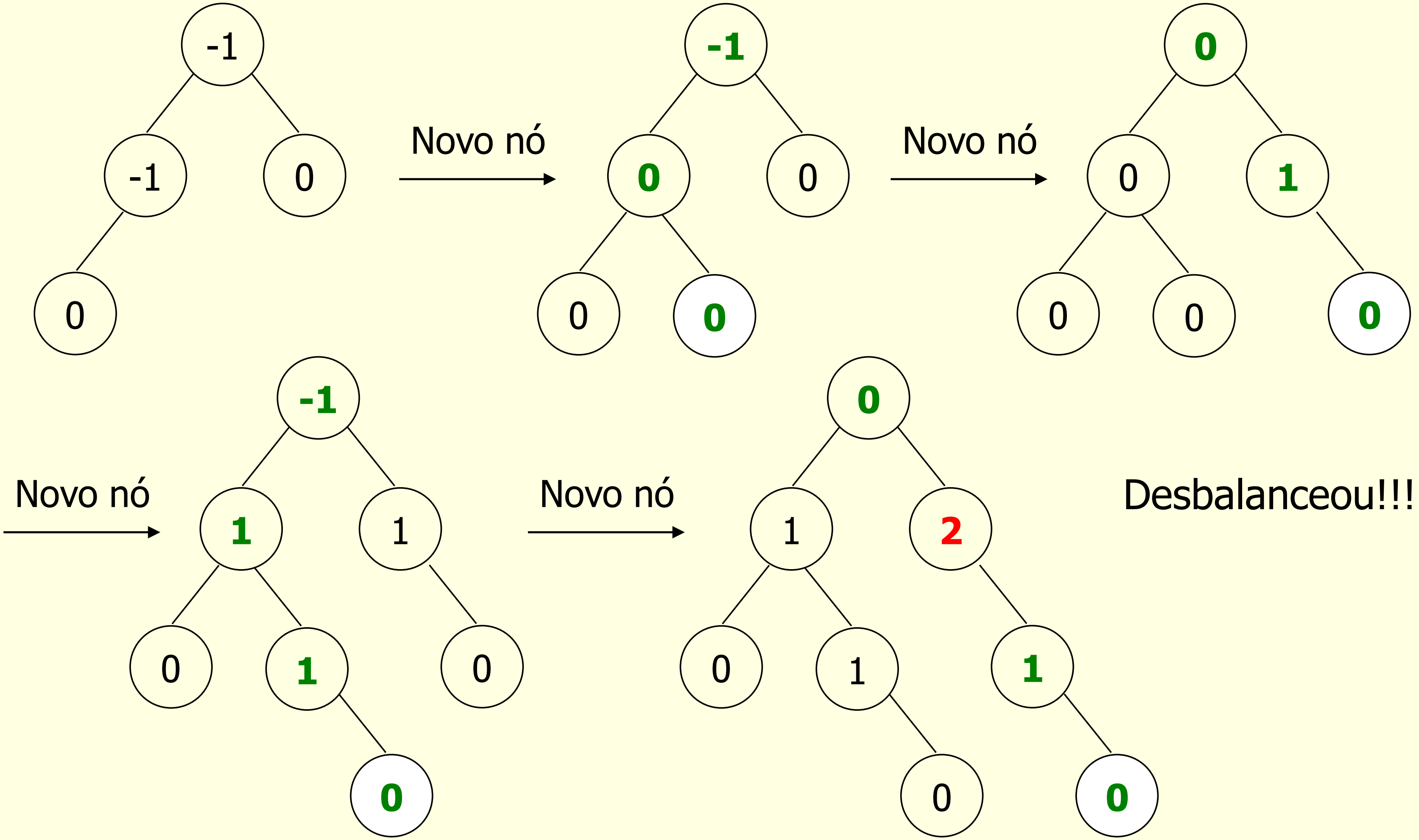
(e)



AVL: quem é e quem não é



AVL: exemplo de desbalanceamento

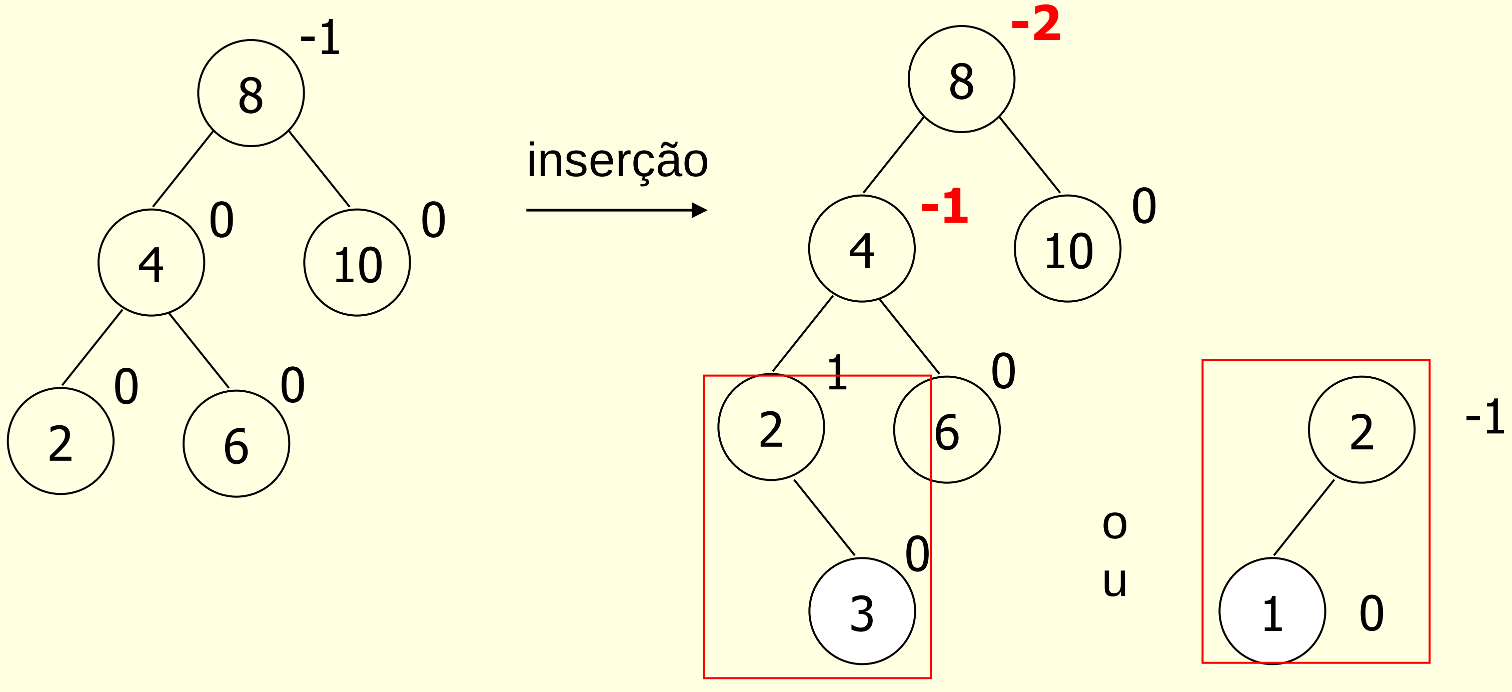


AVL

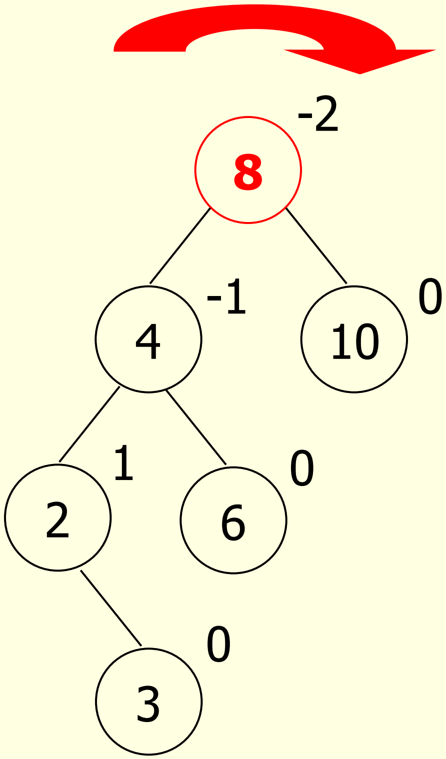
- Controle do balanceamento
  - Altera-se o algoritmo de inserção para balancear a árvore quando ela se tornar desbalanceada após uma inserção (nó com FB 2 ou -2)
    - Rotações
      - Se árvore pende para esquerda (FB negativo), rotaciona-se para a direita
      - Se árvore pende para direita (FB positivo),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

AVL: primeiro caso

- Raiz de uma subárvore com FB -2 (ou 2) e um nó filho com FB -1 (ou 1)
  - Os fatores de balanceamento têm sinais iguais: subárvores de nó raiz e filho pendem para o mesmo lado

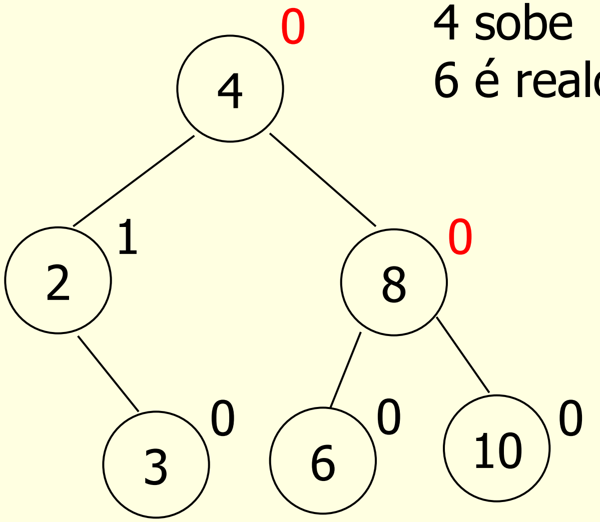


AVL: primeiro caso



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

8 desce
4 sobe
6 é realocado



Pendendo para
a esquerda

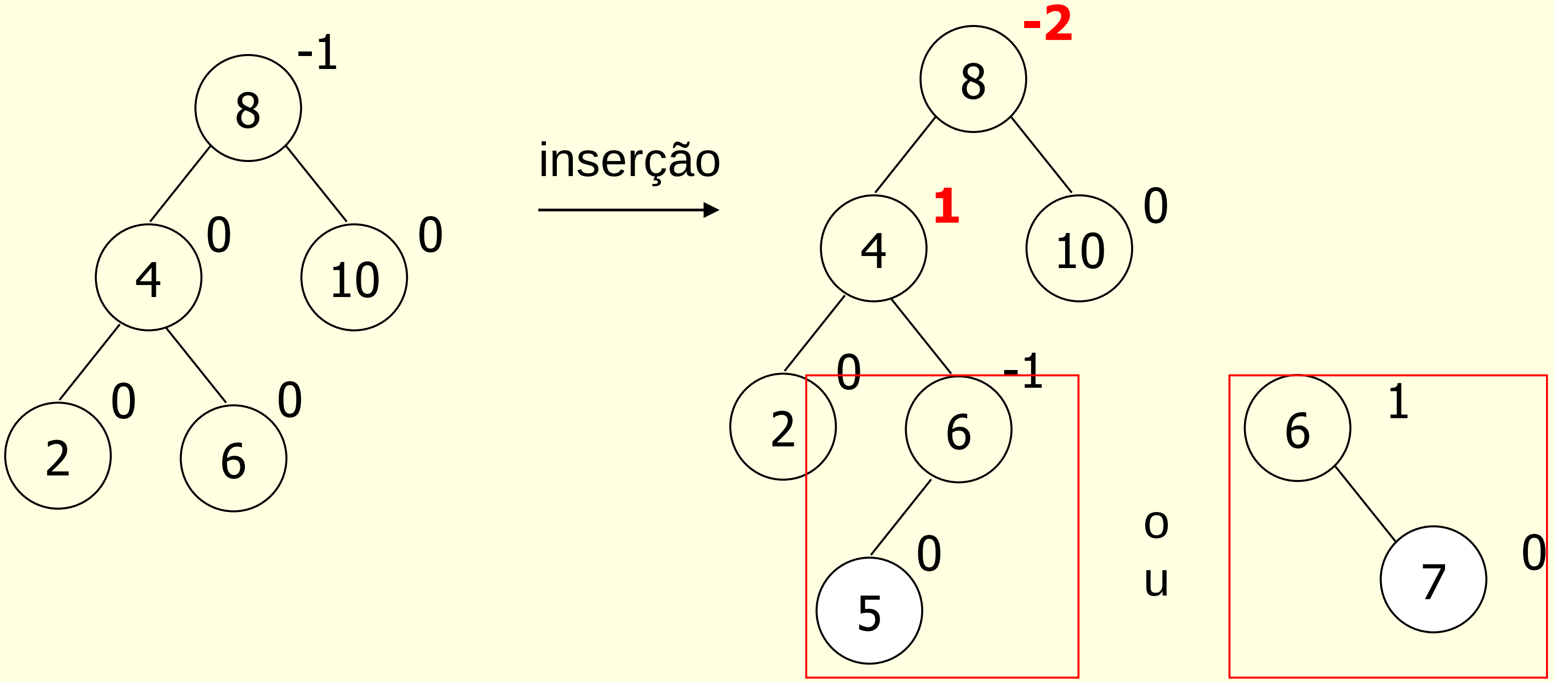
Árvore balanceada!!!

AVL: primeiro caso

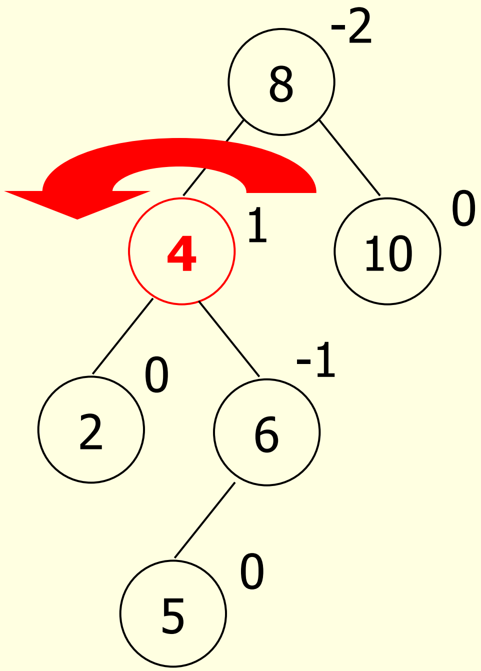
- Quando subárvores do pai e filho pendem para um mesmo lado
  - Rotação simples para o lado oposto
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

AVL: segundo caso

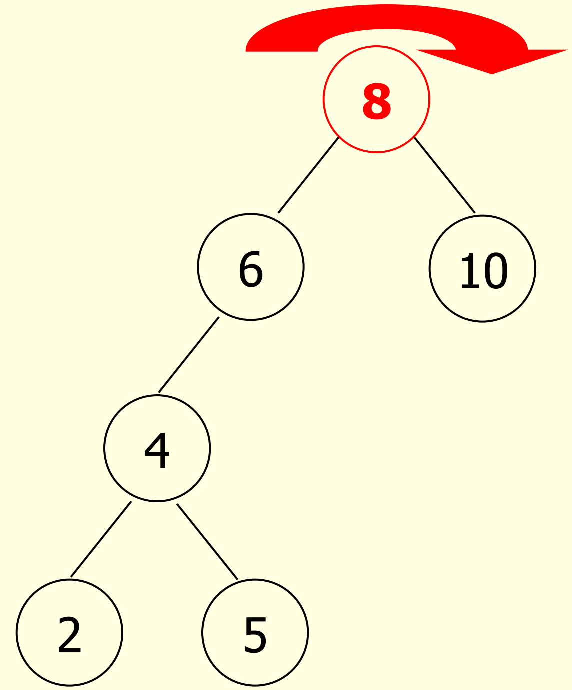
- Raiz de uma subárvore com FB -2 (ou 2) e um nó filho com FB 1 (ou -1)
  - Os fatores de balanceamento têm sinais opostos: subárvore de nó raiz pende para um lado e subárvore de nó filho pende para o outro (ou o contrário)



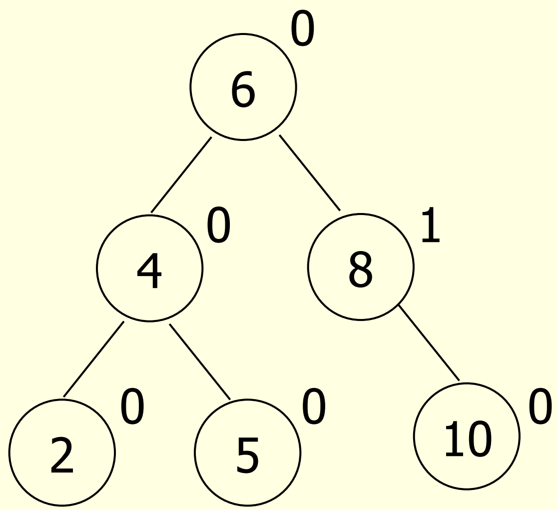
AVL: segundo caso



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



Rotação do nó pai
para a direita
(nó com FB=-2 ou 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
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

AVL

- As transformações dos casos anteriores diminuem em 1 a altura da subárvore com raiz desbalanceada p

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

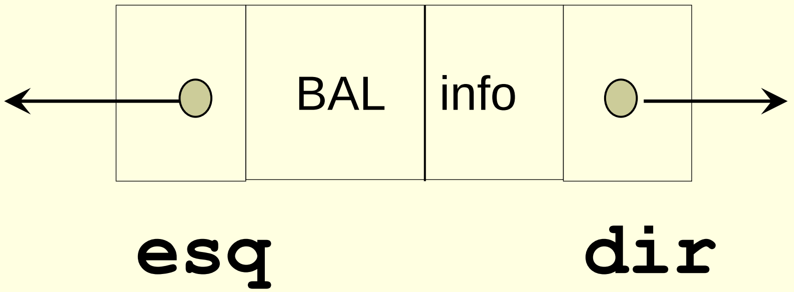
AVL

- 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)

  - Em geral, armazena-se uma variável de balanceamento em cada nó para indicar o FB

AVL – Estrutura de Dados

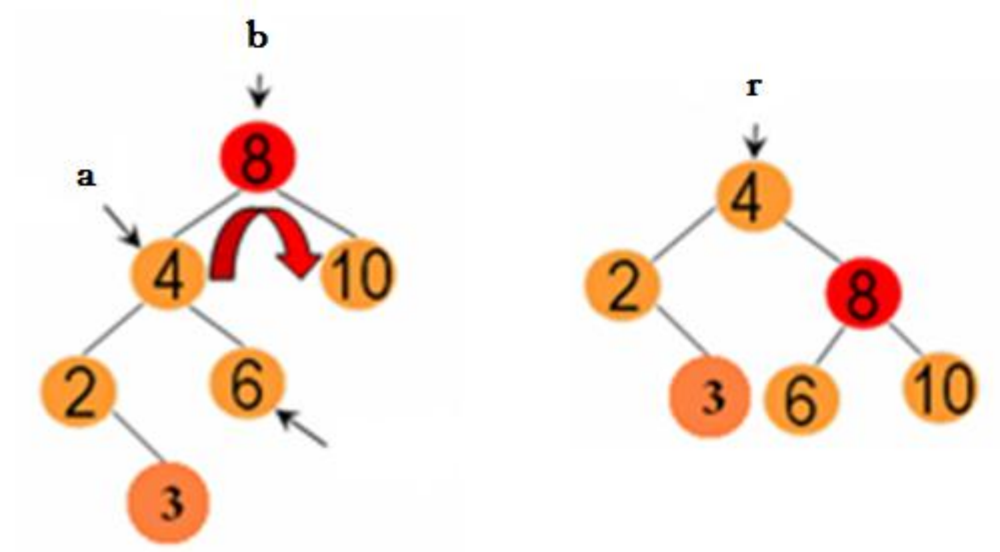



```
typedef  int tipo_elem;

typedef struct arv *Arv;

struct arv {

        tipo_elem info;
        struct arv *esq;
        struct arv *dir;
        int BAL;
}
```

AVL: algoritmo de inserção

- Rotação simples à direita

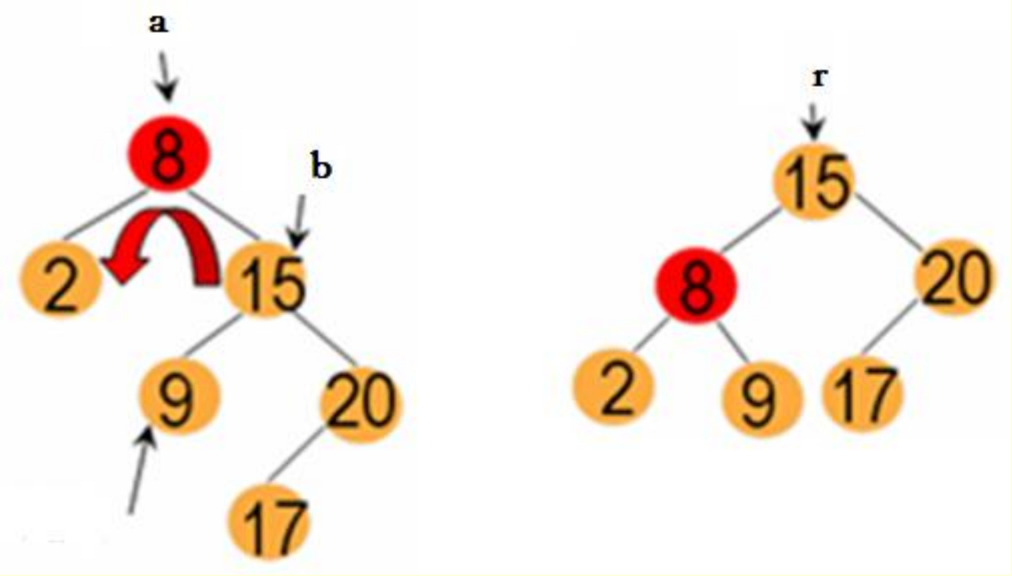
```
void rot_dir(Arv *r) {
    no *b=*r;
    no *a=b->esq;
    b->esq=a->dir;
    a->dir=b;
    a->BAL=0;
    b->BAL=0;
    *r=a;
}
```



AVL: algoritmo de inserção

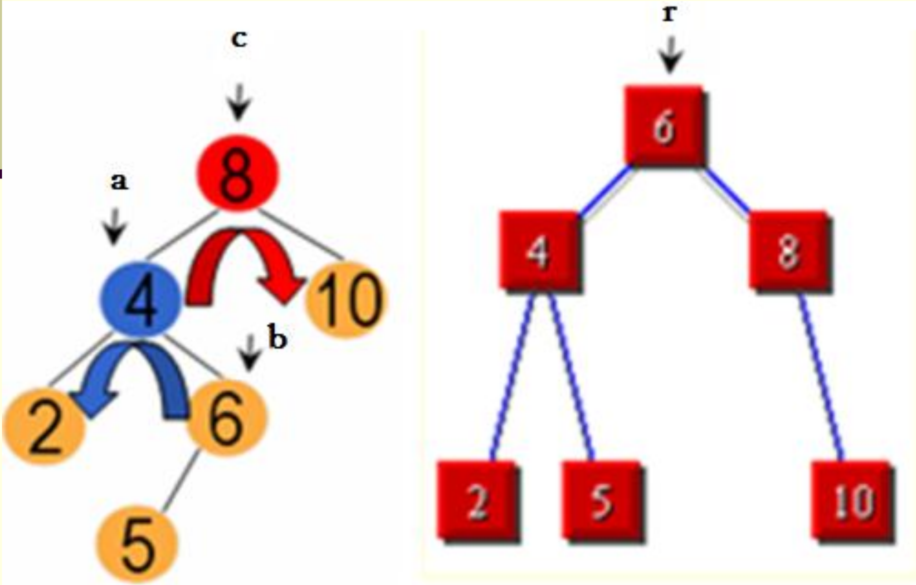
- Rotação simples à esquerda

```
void rot_esq (Arv *r) {
    no *a=*r;
    no *b=a->dir;
    a->dir=b->esq;
    b->esq=a;
    a->BAL=0;
    b->BAL=0;
    *r=b;
}
```



Rotação dupla esquerda e direita

```
void  rot_esq_dir (Arv *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;
```

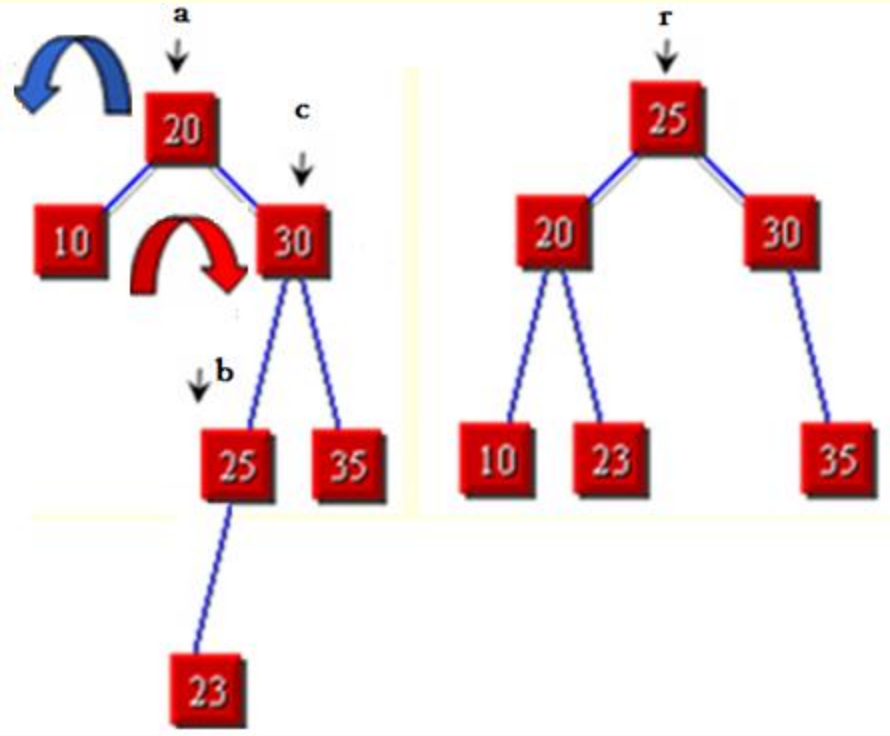


Atualizar BAL de a e c em função de b – nova raiz

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

Rotação dupla direita e esquerda

```
void rot_dir_esq(Arv *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;
```



Atualizar BAL de a e c em função de b – nova raiz

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

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

```
int insere_AVL(Arv * p, int x, int *cresceu) {
    if (*p==NULL) {
 { insere nó p com conteúdo x, como nó folha}
      *p = (Arv) malloc(sizeof(struct arv));
       if (*p == NULL)
        return 0; // falha na inserção: não há espaço
     else {
        (*p)->info = x;
        (*p)->BAL =0;
        (*p)->esq = NULL; (*p)->dir = NULL;
        *cresceu=1;
        return 1; // inserção com sucesso
     }
}
```

```
else if (x==(*p)->info) return 0;
        else if (x<(*p)->info) {
            if (insere_AVL(&(*p)->esq,x,cresceu)) {
            if (*cresceu) {              // inseriu esq: verificar balanceamento
                    switch ((*p)->BAL) {
                        case -1: // BAL(p) = -2
                            if ((*p)->esq->BAL== -1) // sinais iguais – pendem para mesmo lado
                              rot_dir(p); // p retorna balanceado
                            // sinais trocados: rotação dupla
                            else rot_dir_esq(p); // p retorna balanceado
                            *cresceu=0; //interrompe propagação
                            break;
                        case 0:
                            (*p)->BAL= -1; // ficou maior à esq.
                            *cresceu=1; // propaga verificação
                            break;
                        case 1: // era maior à direita
                            (*p)->BAL=0; // balanceou com ins. esq
                            *cresceu=0; //interrompe propagação
                            break;
              }
            }
            return 1;
          }
          else return 0;  }
```

```
else {
        if (insere_AVL(&(*p)->dir,x,cresceu)) {
          if (*cresceu) {    // inseriu à dir: verificar balanceamento}
                    switch ((*p)->BAL) {
                          case -1: // era mais alto à esq.
                              (*p)->BAL=0; ; // balanceou com ins. dir
                              *cresceu=0; //interrompe propagação
                              break;
                          case 0: ; // direita fica maior
                              (*p)->BAL=1; *cresceu=1; // propaga verificação
                              break;
                          case 1: // BAL(p) = 2
                              if ((*p)->dir->BAL==1) // sinais iguais – pendem para mesmo lado
                                 rot_esq(p); // p retorna balanceado
                              // sinais trocados: rotação dupla
                              else rot_dir_esq(p); // p retorna balanceado
                              *cresceu=0; //interrompe propagação
                              break;
              }
          }
         return 1;
        }
        else return 0;
    }
}
```

AVL

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

AVL

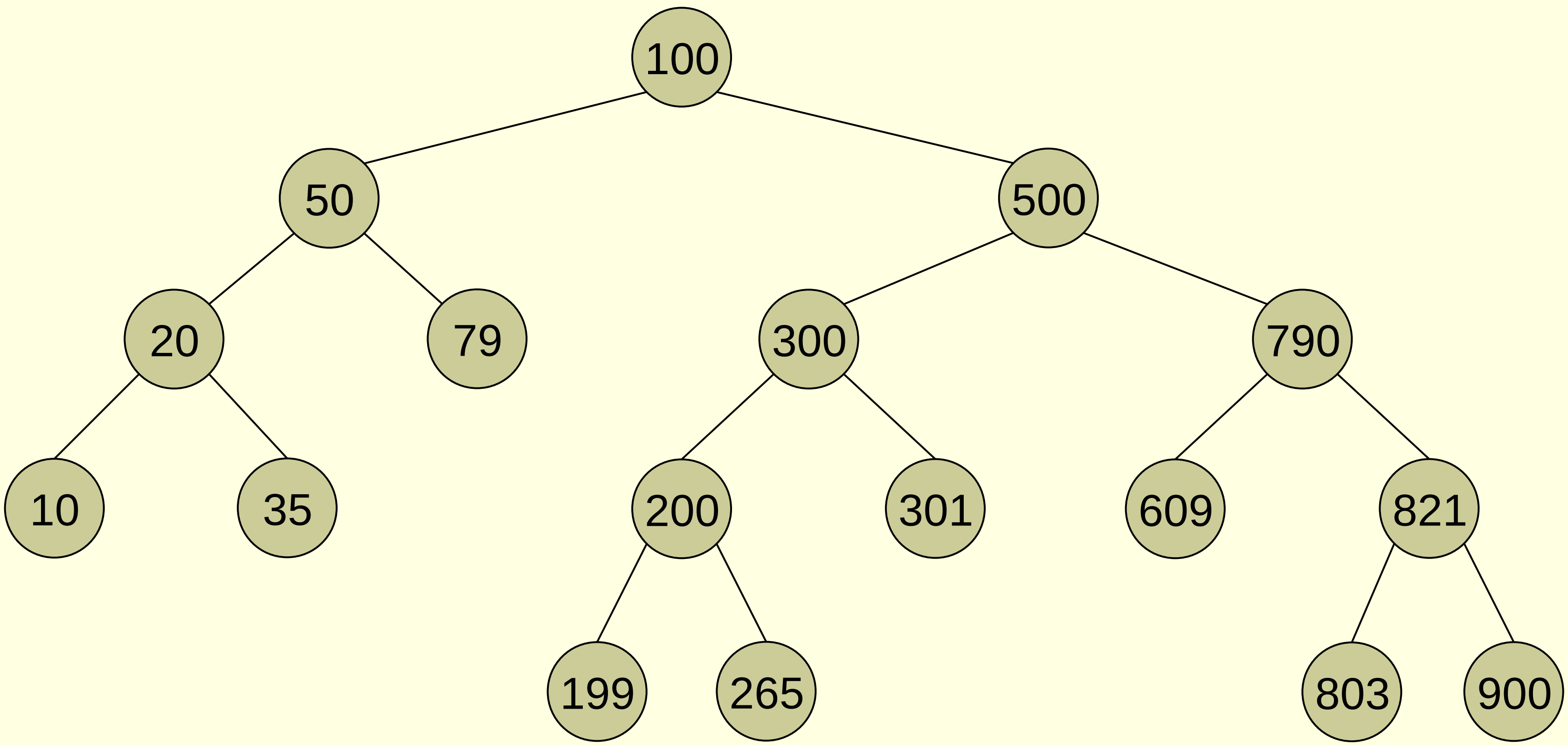
- Exercício Casa
  - Inserir os elementos A, B, C, ..., J em uma árvore e balancear quando necessário

AVL

- Os percursos in-ordem da árvore original e da balanceada permanecem iguais
  - Exercício: prove para um dos exemplos anteriores!

AVL

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



Remoção em AVL

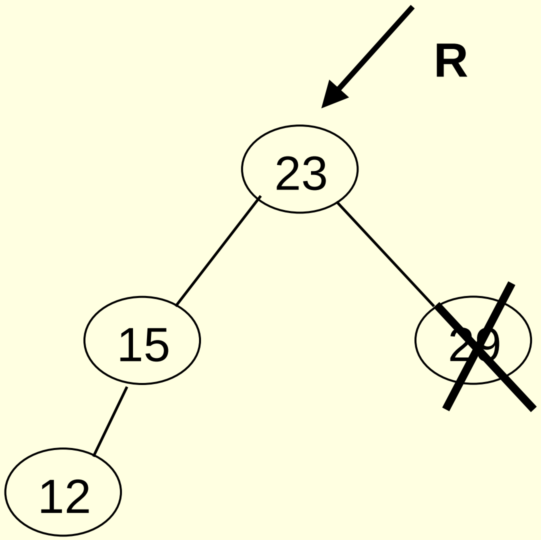
- Remoção é um pouco mais complexo e se divide em duas partes:
- Em primeiro lugar, a remoção de um nó qualquer é substituída pela remoção de uma folha. Para tanto, existem 3 casos possíveis:

1. o nó tem grau zero e, portanto, já é uma folha;
2. o nó tem grau um – pela propriedade AVL, a sua única subárvore é necessariamente constituída por uma folha, cujo valor é copiado para o nó pai; o nó a ser eliminado é a folha da subárvore;
3. o nó tem grau dois – o seu valor é substitído pelo maior valor contido na sua subárvore esquerda (ou o menor valor contido na sua subárvore direita); o nó que continha o menor (ou maior) valor copiado tem necessariamente grau zero ou um, recaindo num dos casos anteriores.

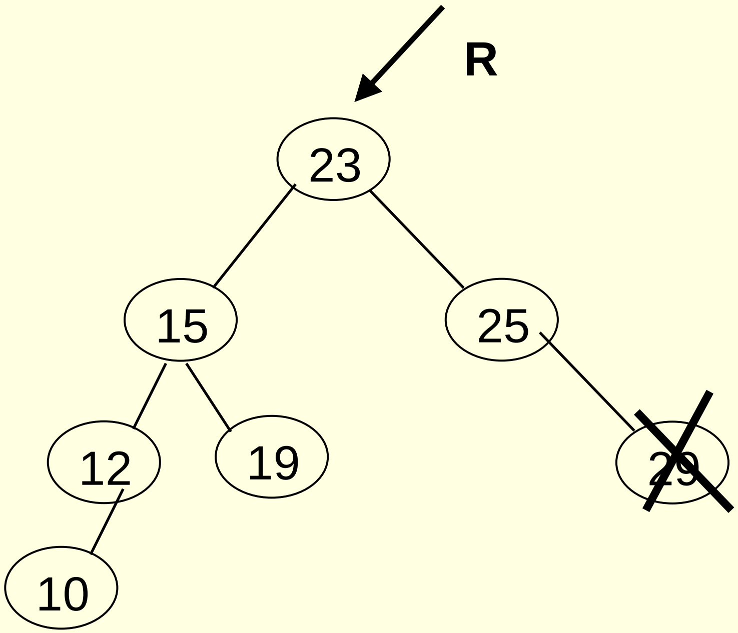
- A segunda parte do algoritmo consiste, portanto, na remoção de uma folha. O processo é semelhante à inserção.

AVL: remoção

- Exemplos



remoção de 29 = inserção de 12

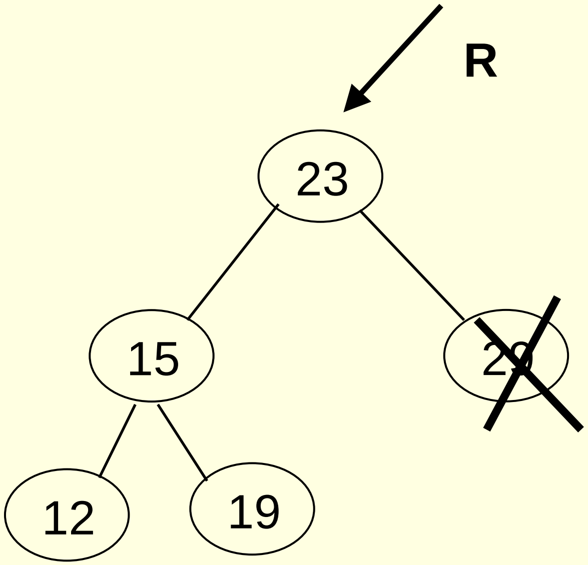


remoção de 29 = inserção de 10

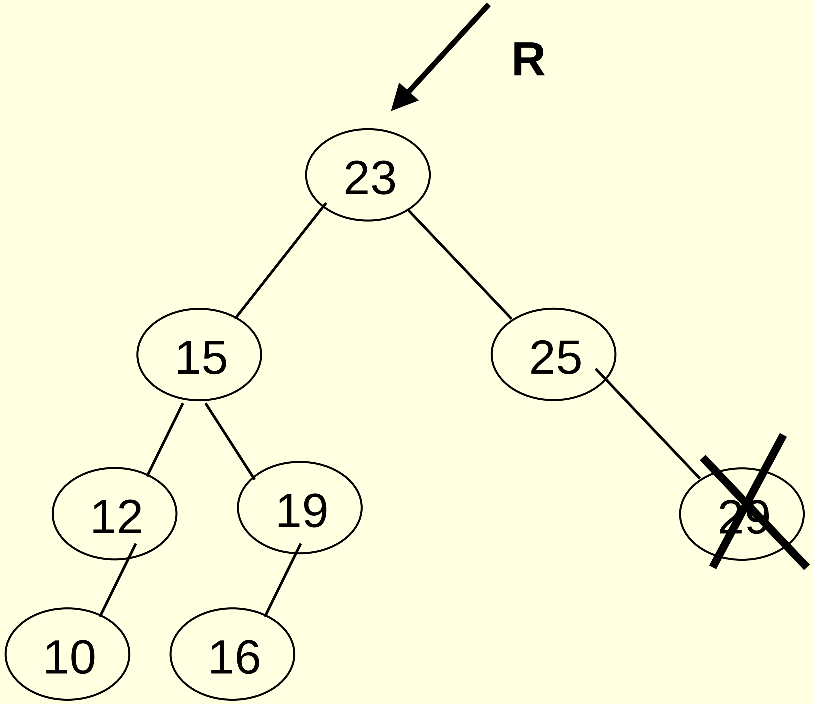
Como balancear?

AVL: remoção

- Exemplos



remoção de 29 = inserção de 12 OU 19



remoção de 29 = inserção de 10 OU 16

Como balancear?

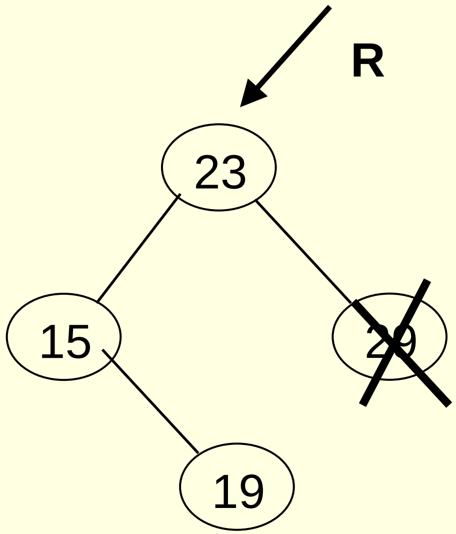
Rotação simples em R

AVL: remoção

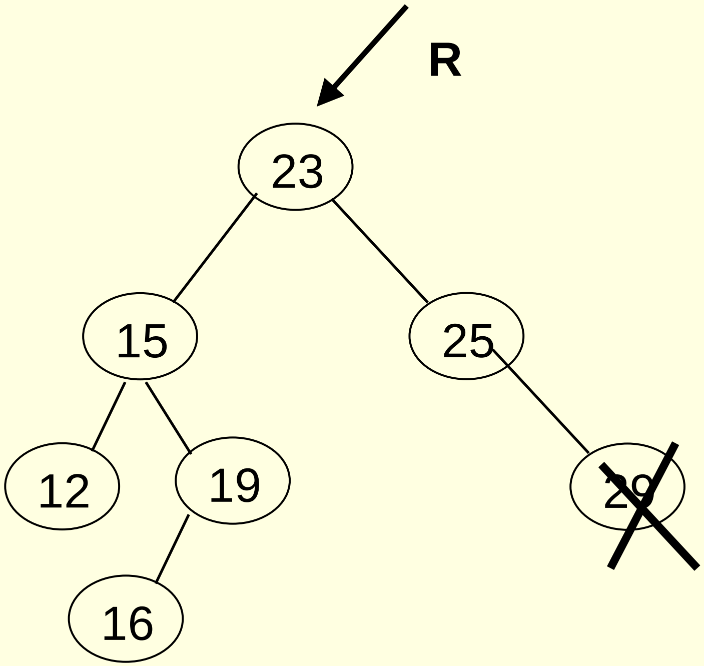
- Primeiro caso
  - Rotação simples em R (FB=2 ou -2) com filho com fator de balanceamento de mesmo sinal (1 ou -1) ou zero
    - Se R negativo, rotaciona-se para a direita; caso contrário, para a esquerda

AVL: remoção

- Exemplos



remoção de 29 = inserção de 19

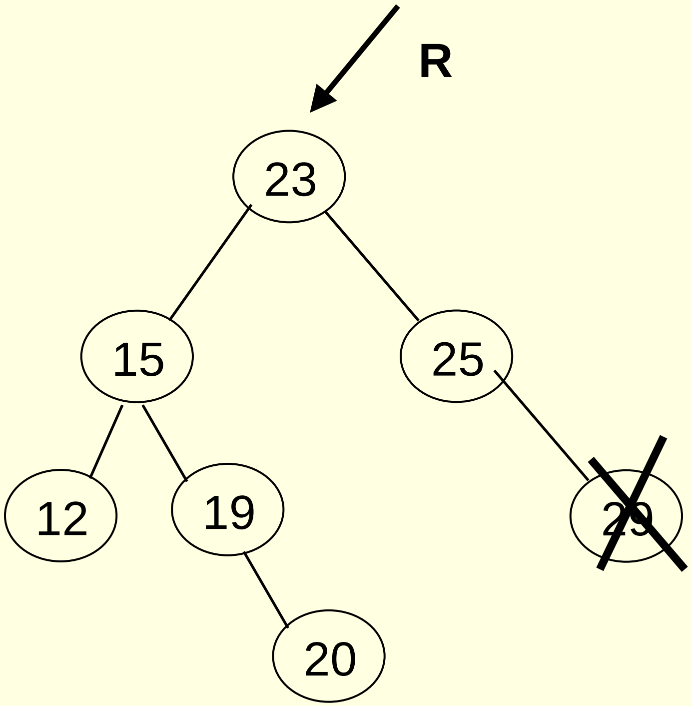


remoção de 29 = inserção de 16

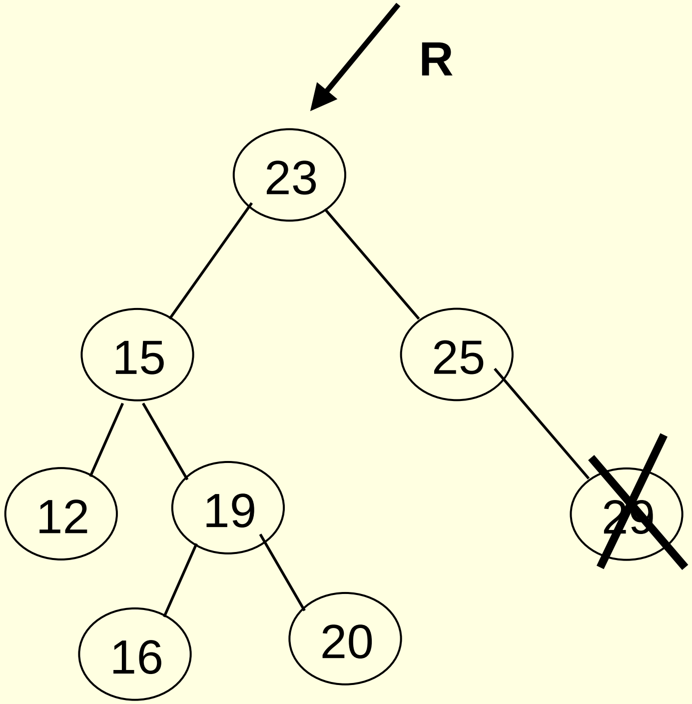
Como balancear?

AVL: remoção

- Exemplos



remoção de 29 = inserção de 20



remoção de 29 = inserção de 16 ou 20

Como balancear?

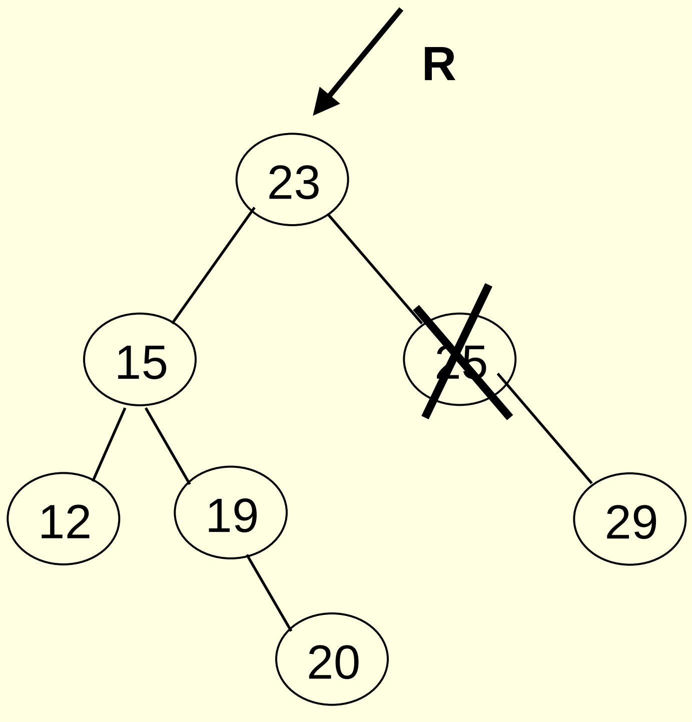
Rotação dupla: filho de R e R

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

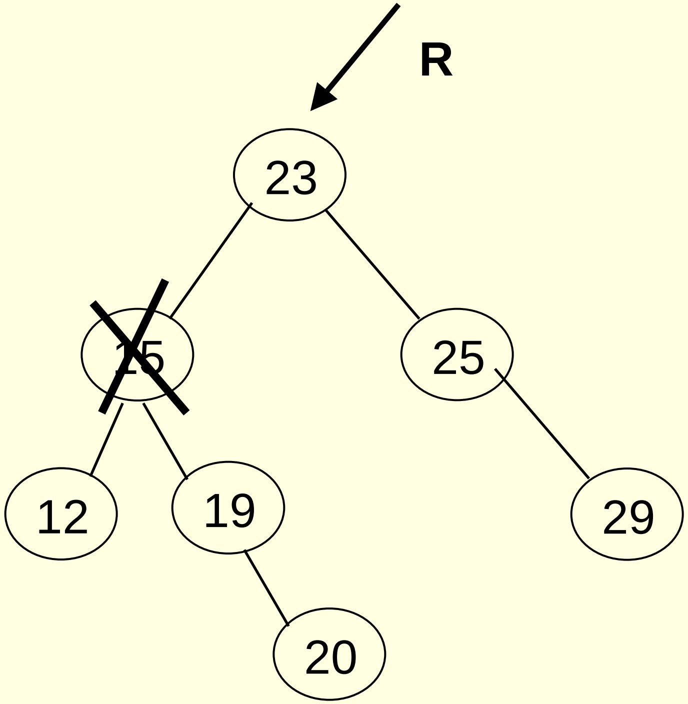
- Questão: como remover um nó intermediário em vez de um nó folha?

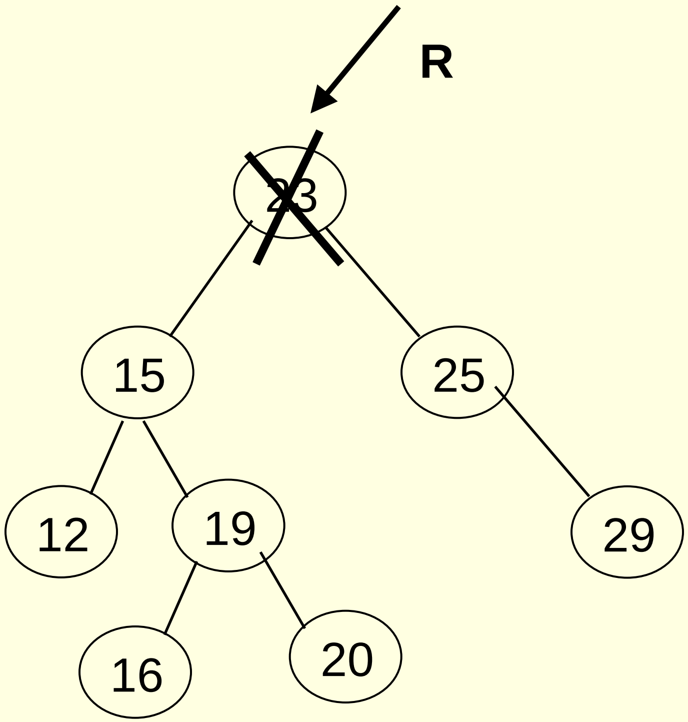


Se nó com grau 1 troca-se pela folha e remove-se a folha.

AVL: remoção

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





Se nó com grau 2, troca-se pelo maior da sub-árvore esquerda ou menor da sub-árvore direita e depois remove-se a folha trocada.

AVL

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

Resumo ABB

- Boa opção como ED para buscas de chaves, SE a árvore é balanceada => tempo proporcional a $\log_2 n$.
- Inserções (como Folhas) e Eliminações (mais complexas) causam desbalanceamento.
- Inserções: melhor se aleatórias (não ordenadas) para evitar linearizações.
- Para manter o balanceamento:
  - Balanceamento global
  - Árvores AVL