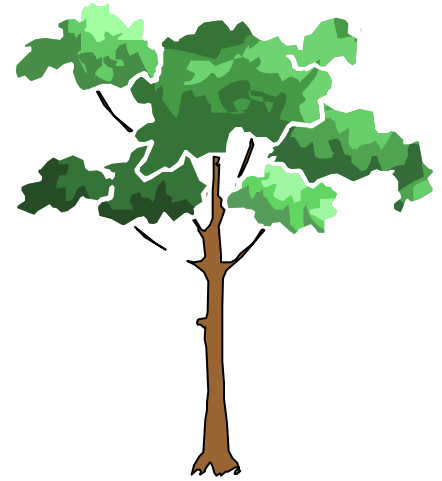


---

# Árvores & Árvores Binárias



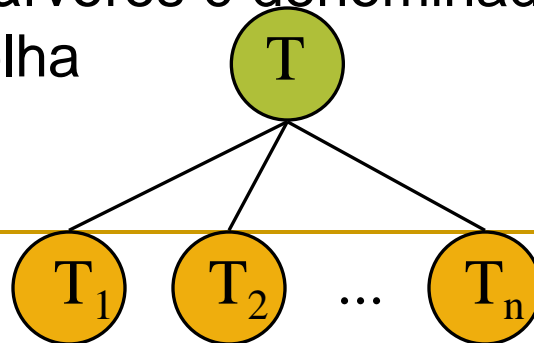
---

# Problema

- Implementações do TAD Lista Linear
  - Lista encadeada
    - eficiente para inserção e remoção dinâmica de elementos, mas ineficiente para busca
  - Lista seqüencial (ordenada)
    - Eficiente para busca, mas ineficiente para inserção e remoção de elementos
- Árvores: solução eficiente para inserção, remoção e busca
  - Representação não linear...

# Definições

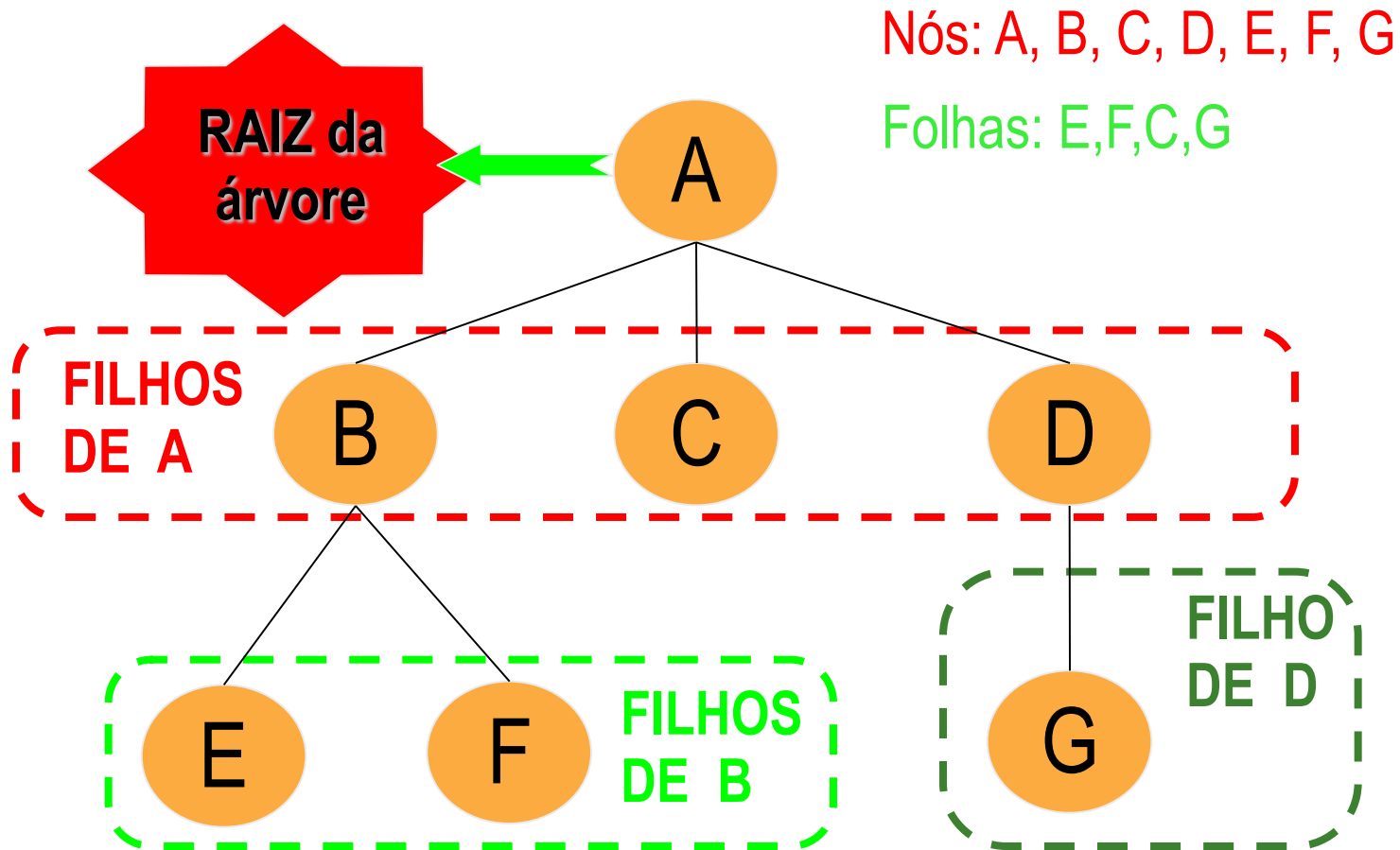
- Árvore  $T$ : conjunto finito de elementos, denominados **nós** ou vértices, tais que:
  - Se  $T = \emptyset$ , a árvore é dita vazia; c.c.
    - (i)  $T$  contém um nó especial, denominado raiz;
    - (ii) os demais nós, ou constituem um único conjunto vazio, ou são divididos em  $n \geq 1$  conjuntos disjuntos não vazios  $(T_1, T_2, \dots, T_n)$ , que são, por sua vez, cada qual uma árvore;
  - $T_1, T_2, \dots, T_n$  são chamadas sub-árvores de  $T$ ;
  - Um nó sem sub-árvores é denominado nó-folha, ou simplesmente, folha



# Definições (cont.)

- **Árvore:** adequada para representar estruturas hierárquicas não lineares, como relações de descendência (pai, filho, irmãos, etc.)
- Se um nó  $X$  é raiz de uma árvore, e um nó  $Y$  é raiz de uma sub-árvore de  $X$ , então  $X$  é **PAI** de  $Y$  e  $Y$  é **FILHO** de  $X$

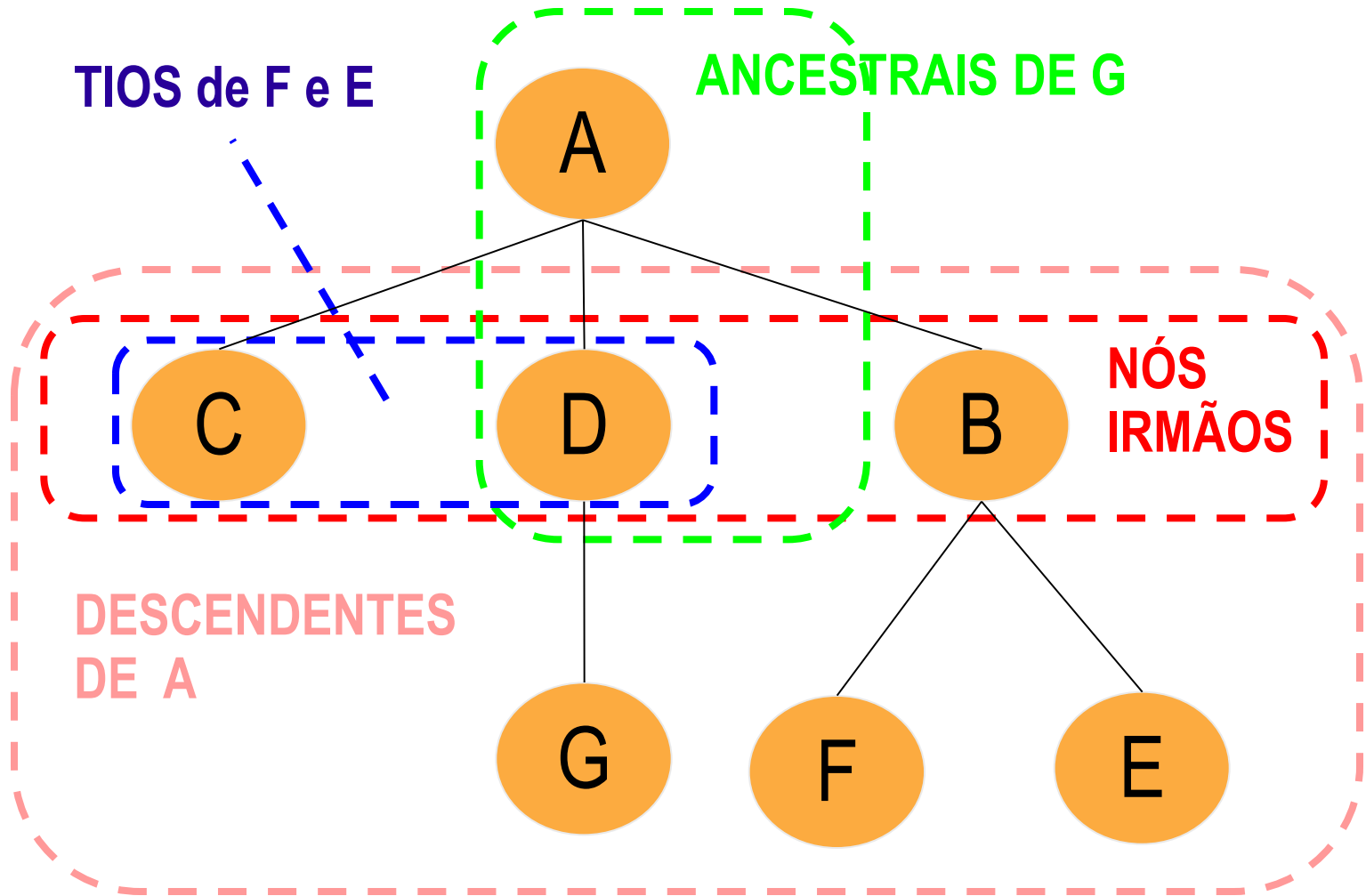
# Definições (cont.)



# Definições (cont.)

- O nó  $X$  é um **ANCESTRAL** do nó  $Y$  (e  $Y$  é **DESCENDENTE** de  $X$ ) se  $X$  é o **PAI** de  $Y$ , ou se  $X$  é **PAI** de algum **ANCESTRAL** de  $Y$
- Dois nós são **IRMÃOS** se são filhos do mesmo pai
- Se os nós  $Y_1, Y_2, \dots, Y_j$  são irmãos, e o nó  $Z$  é filho de  $Y_1$ , então  $Y_2, \dots, Y_j$  são **TIOs** de  $Z$

# Definições (cont.)



---

# Conceitos

- O NÍVEL de um nó **X** é definido como:
  - O nível do nó raiz é 1
  - O nível de um nó não-raiz é dado por (nível de seu nó PAI + 1)
  
- Os nós de maior nível são também nós-folha.

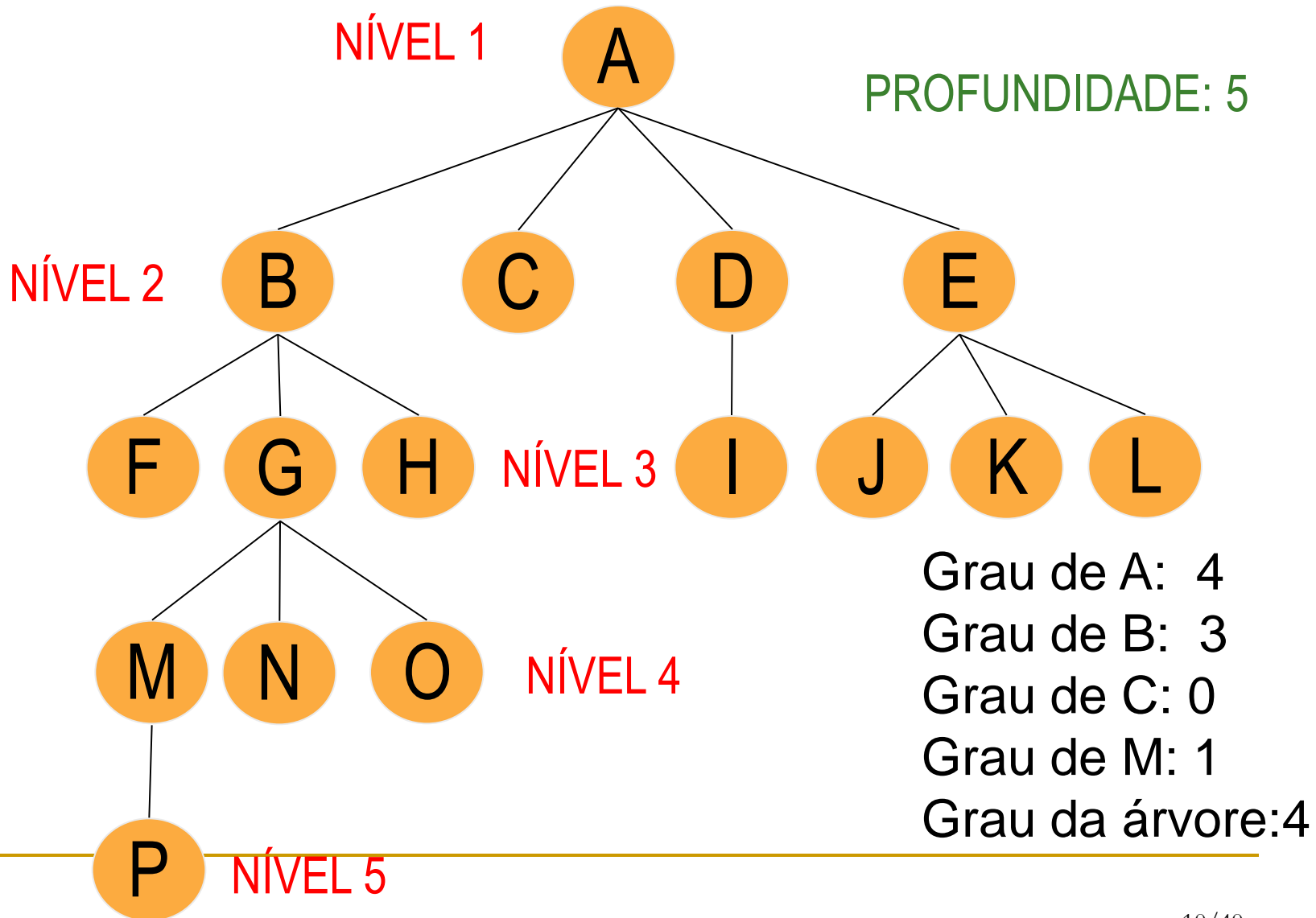


---

# Conceitos (cont.)

- O GRAU de um nó **X** pertencente a uma árvore é igual ao número de filhos do nó **X**
- O GRAU de uma árvore **T** é o maior entre os graus de todos os seus nós

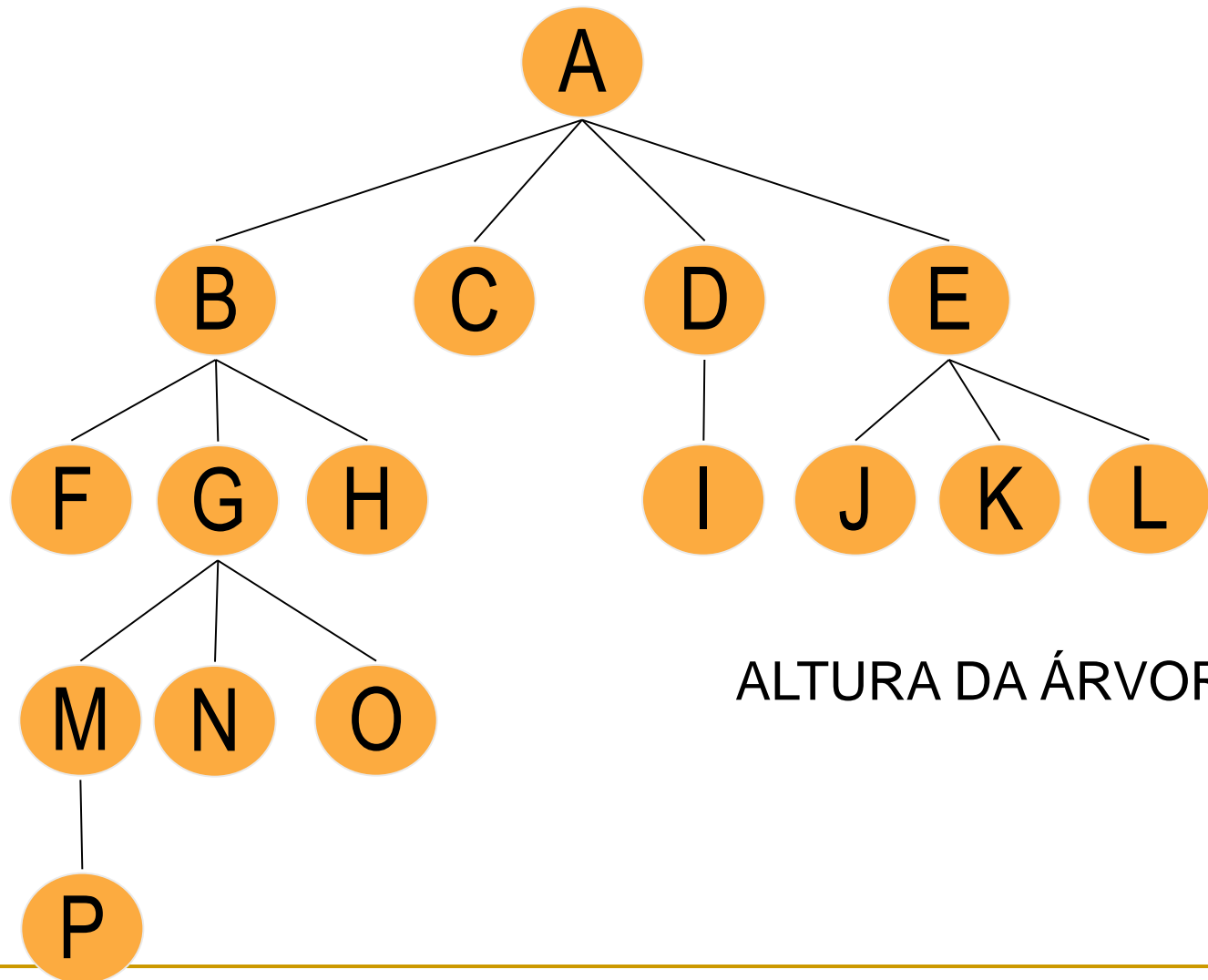
# Conceitos (cont.)



# Conceitos (cont.)

- Uma sequência de nós distintos  $v_1, \dots, v_k$  tal que cada nó  $v_{i+1}$  é filho de  $v_i$  é denominada um **CAMINHO** na árvore (diz-se que  $v_i$  alcança  $v_k$ ).
- O número de arestas de um caminho define o **COMPRIMENTO DO CAMINHO**.
- A **ALTURA** ou **PROFUNDIDADE** de uma árvore **X** é dada pelo **MAIOR NÍVEL** de seus nós. Alternativamente, corresponde ao número de nós do maior caminho entre a raiz e os nós folhas.
- Denota-se a altura de uma árvore com raiz **X** por  **$h(X)$** , e a altura de uma sub-árvore com raiz **y** por  **$h(y)$**

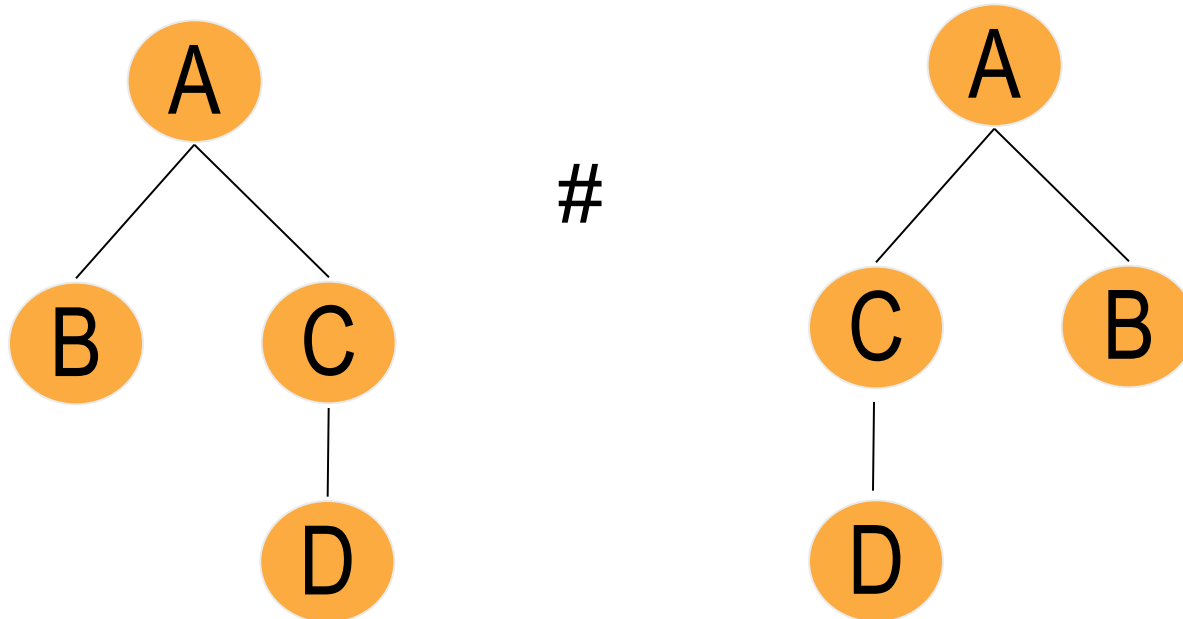
# Conceitos (cont.)



ALTURA DA ÁRVORE: 5

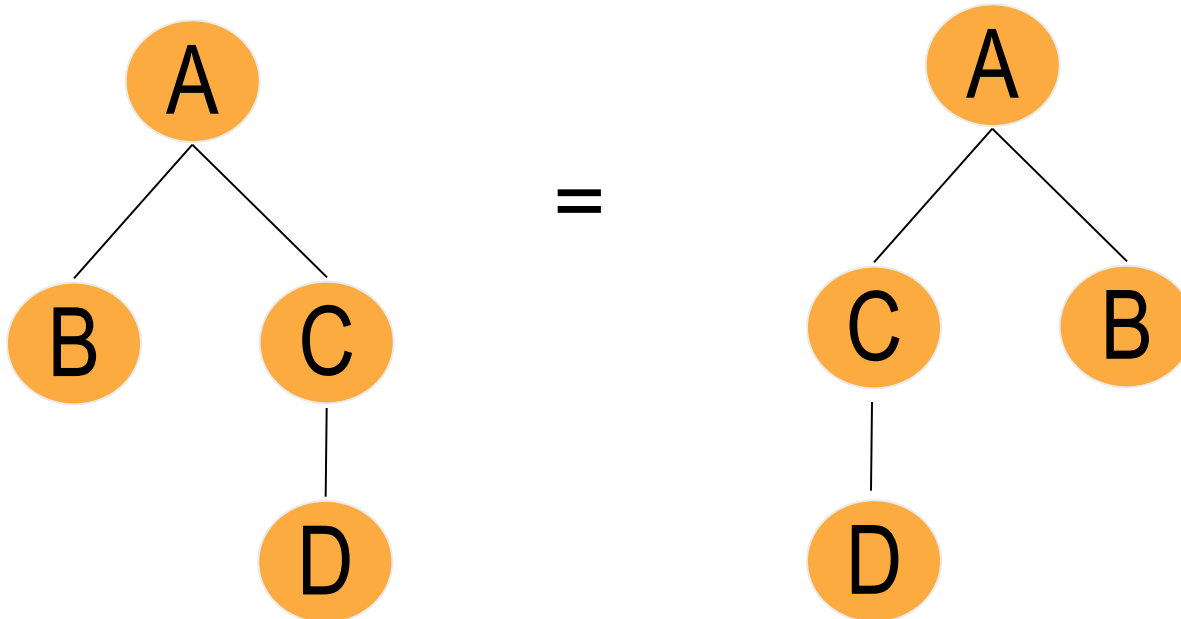
# Conceitos (cont.)

- Uma árvore é **ORDENADA** se considerarmos o conjunto de sub-árvores  $T_1, T_2, \dots, T_n$  como um conjunto ordenado.



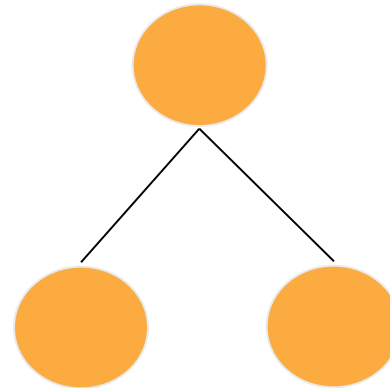
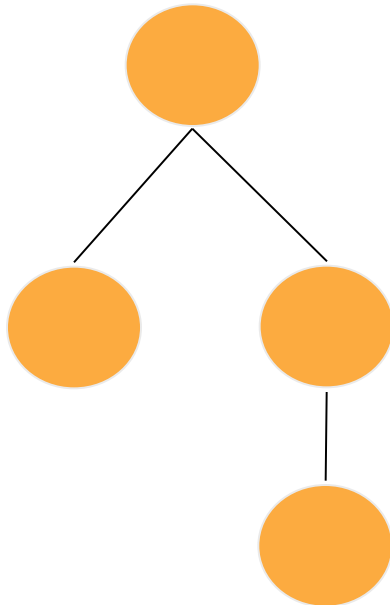
# Conceitos (cont.)

- Uma árvore é **ORIENTADA** se apenas a orientação relativa dos nós – e não sua ordem – está sendo considerada.



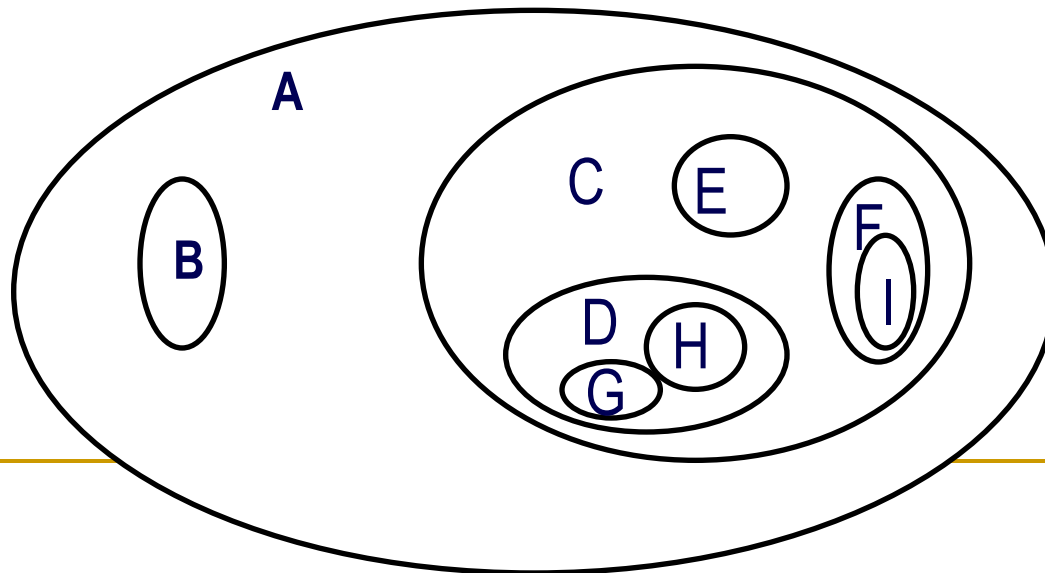
# Conceitos (cont.)

- Uma **FLORESTA** é um conjunto de 0 ou mais árvores distintas



# Outras Representações Gráficas

- Representação por parênteses aninhados
  - ( A ( B ) ( C ( D ( G ) ( H ) ) ( E ) ( F ( I ) ) ) )
  - ou seja, uma lista generalizada!!
- Representação por Diagramas de Venn





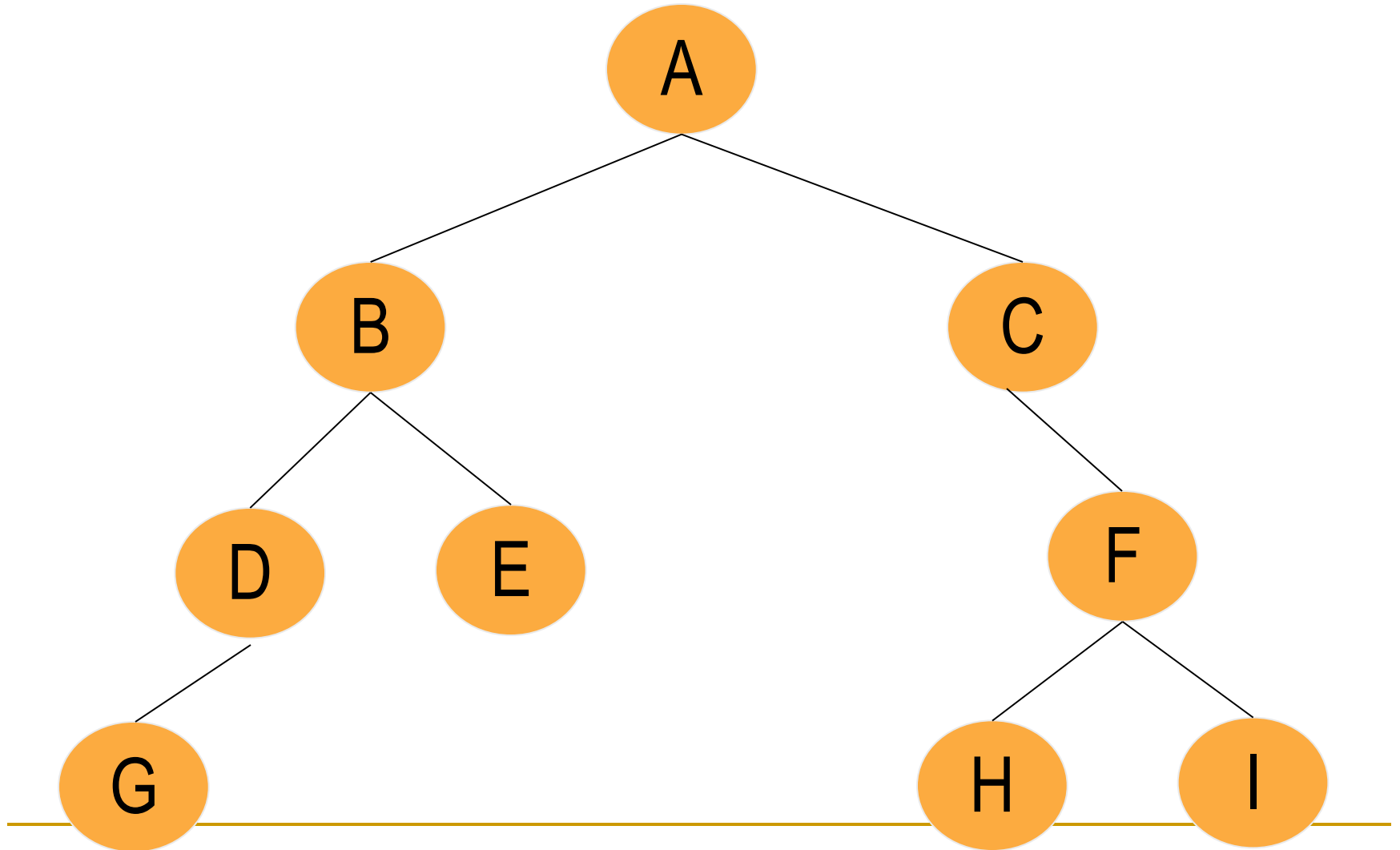
# Árvore Binárias (AB)

- Uma Árvore Binária (AB)  $T$  é um conjunto finito de elementos, denominados nós ou vértices, tal que:
  - (i) Se  $T = \emptyset$ , a árvore é dita vazia, ou
  - (ii)  $T$  contém um nó especial, chamado raiz de  $T$ , e os demais nós podem ser subdivididos em dois sub-conjuntos distintos  $T_E$  e  $T_D$ , os quais também são árvores binárias.  $T_E$  e  $T_D$  são denominados sub-árvore esquerda e sub-árvore direita de  $T$ , respectivamente

# Árvore Binárias (AB) (cont.)

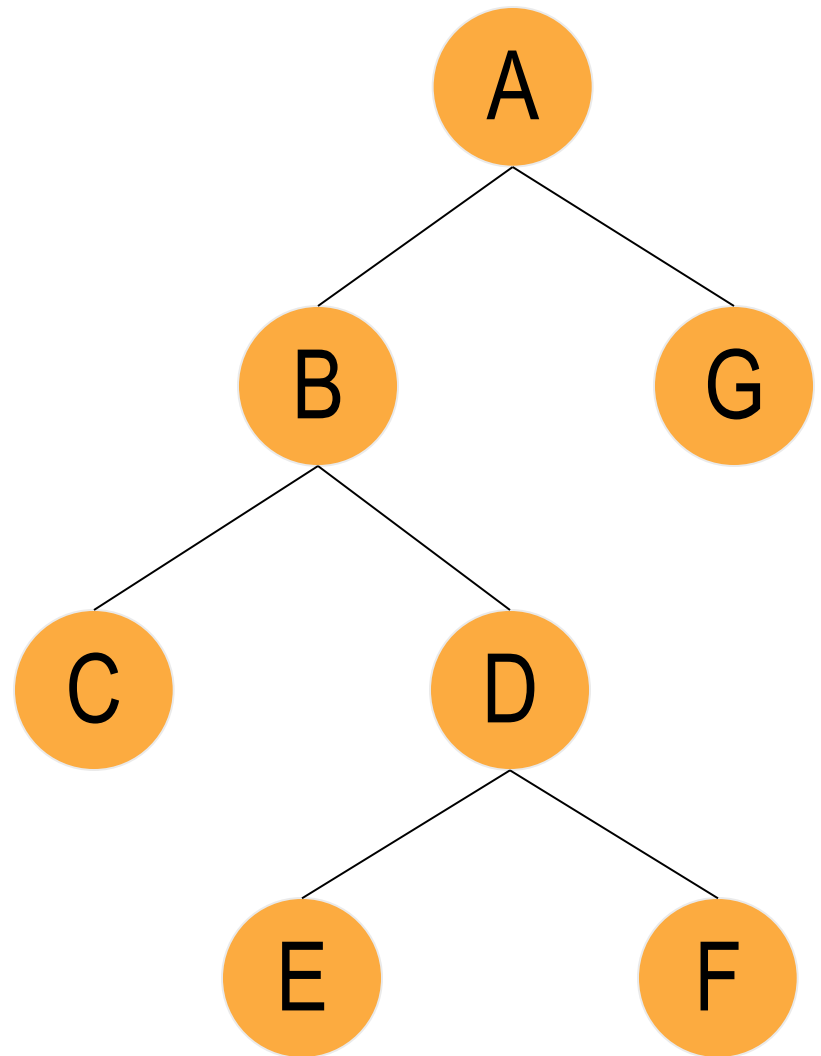
- A raiz da sub-árvore esquerda (direita) de um nó  $v$ , se existir, é denominada filho esquerdo (direito) de  $v$ . Pela natureza da árvore binária, o filho esquerdo pode existir sem o direito, e vice-versa
- Se  $r$  é a raiz de  $T$ , diz-se que  $T_{Er}$  e  $T_{Dr}$  são as sub-árvores esquerda e direita de  $T$ , respectivamente

# Árvore Binárias (AB) (exemplo)



# Árvore Estritamente Binária

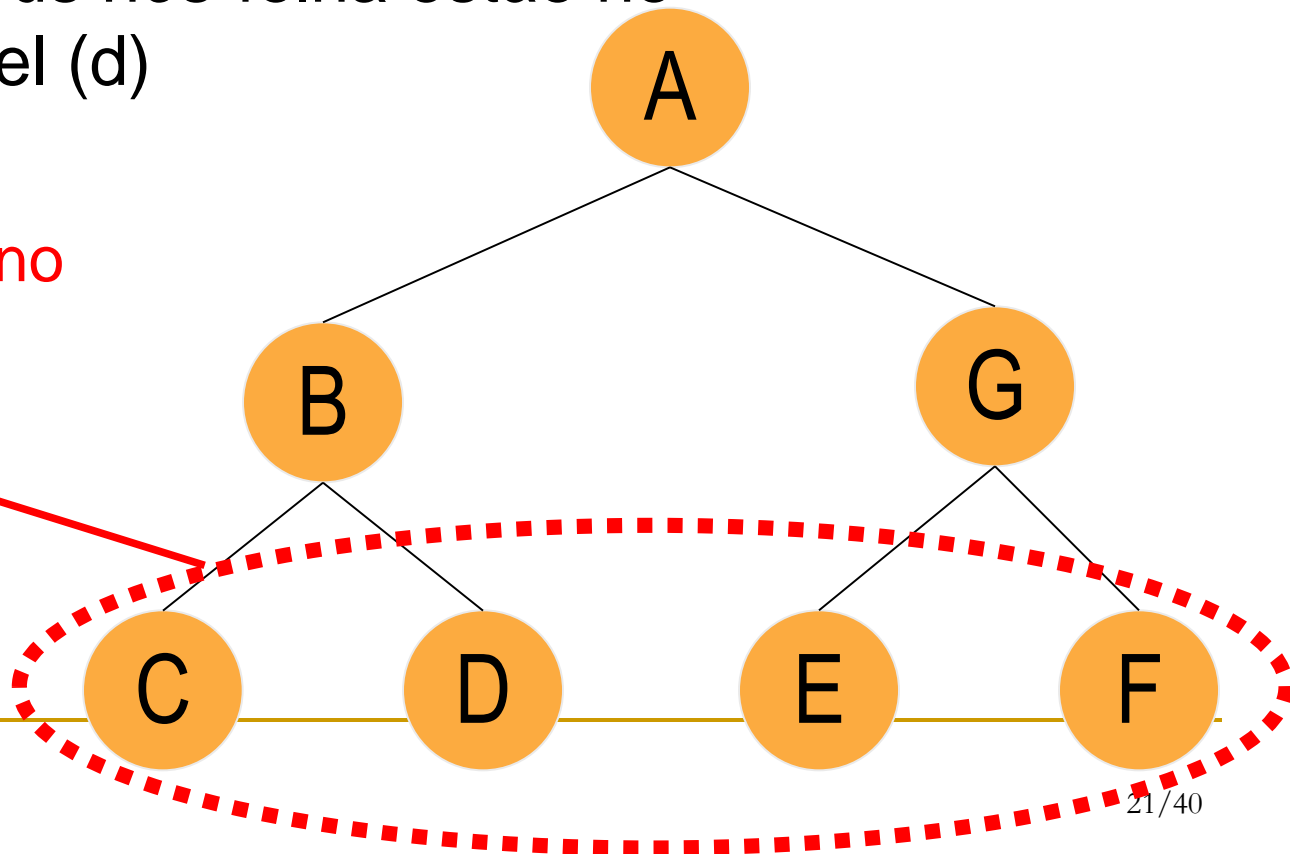
- Uma **Árvore Estritamente Binária** tem nós que têm ou 0 (nenhum) ou dois filhos
- Nós internos (não folhas) sempre têm 2 filhos



# Árvore Binária Completa

- **Árvore Binária Completa (ABC)**
  - é estritamente binária, de nível  $d$ ; e
  - todos os seus nós-folha estão no mesmo nível ( $d$ )

C,D,E,F estão no nível 3  
(altura = 3)



# Árvore Binária Completa (cont.)

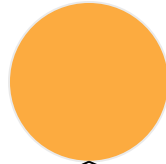
- Dada uma ABC e sua altura, pode-se calcular o número total de nós na árvore
  - p.ex., uma ABC com altura 3 tem 7 nós
    - Nível 1: => 1 nó
    - Nível 2: => 2 nós
    - Nível 3: => 4 nós
    - No. Total de nós =  $1 + 2 + 4 = 7$
  - Verifique que: se uma ABC tem altura  $h$ , então o número de nós da árvore é dado por:

$$N = 2^h - 1$$

Nível

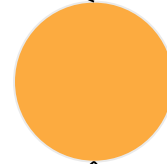
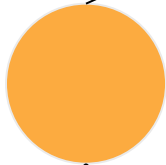
Número de nós por nível

1



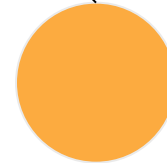
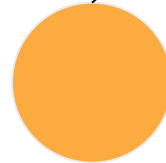
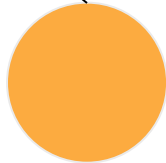
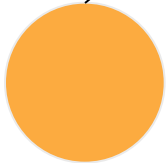
$$1 = 2^0$$

2



$$2 = 2^1$$

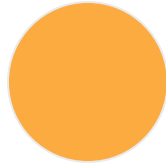
3



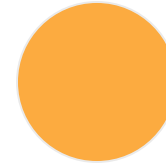
$$4 = 2^2$$

.....

h



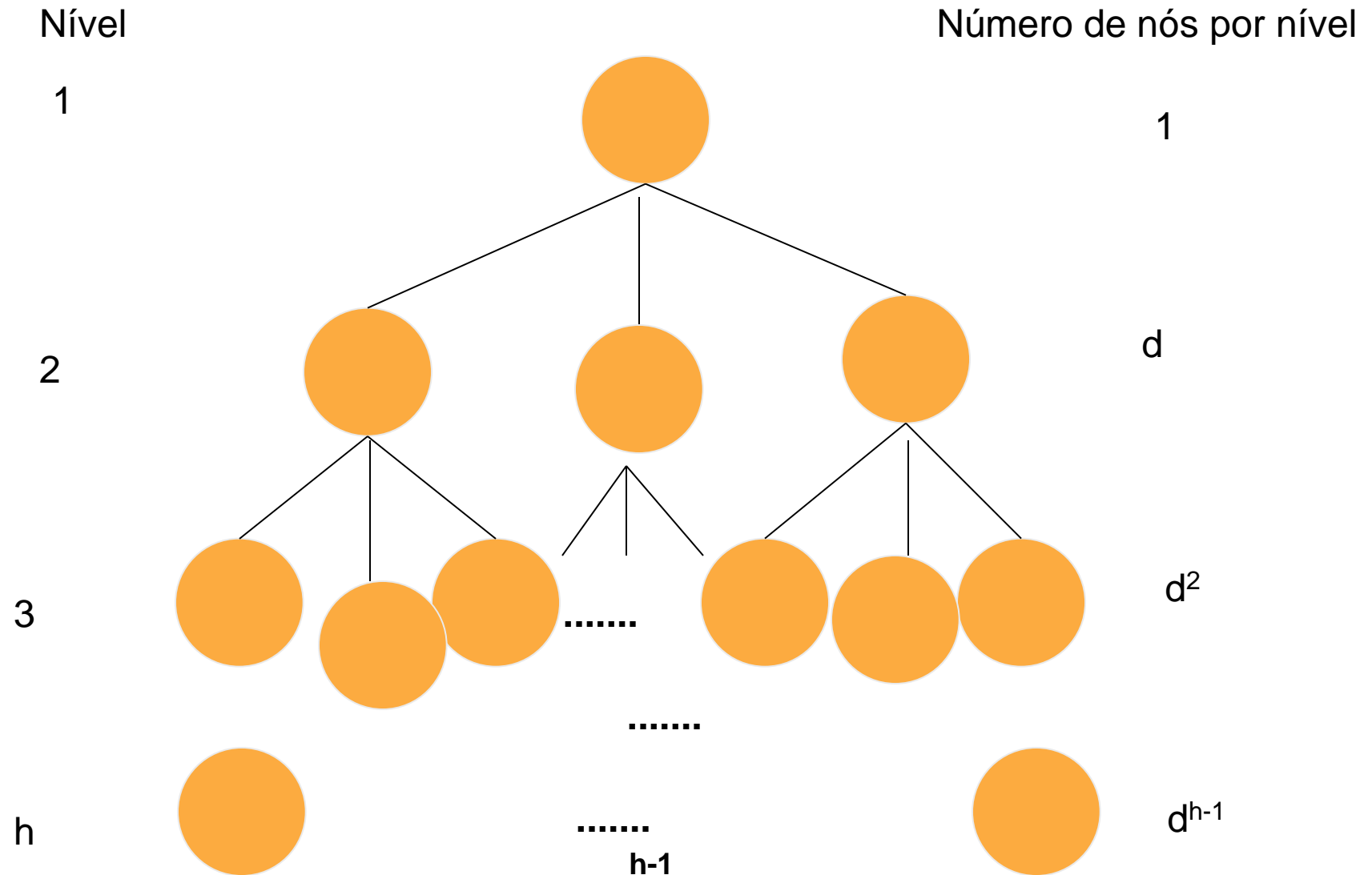
.....



$$2^{h-1}$$

$$\therefore N = \sum_{i=0}^{h-1} 2^i = 2^h - 1$$

# Generalize para *Árvore Completa* de grau *d* e altura *h*



$$\therefore N(h) = \sum_{i=0}^{h-1} d^i = \frac{d^h - 1}{d - 1}$$



# Inversamente:

- Se **N** é o número de nós de uma Árvore Completa, de grau **d**, qual é a altura **h** da árvore?

$$N = \frac{d^h - 1}{d - 1}$$

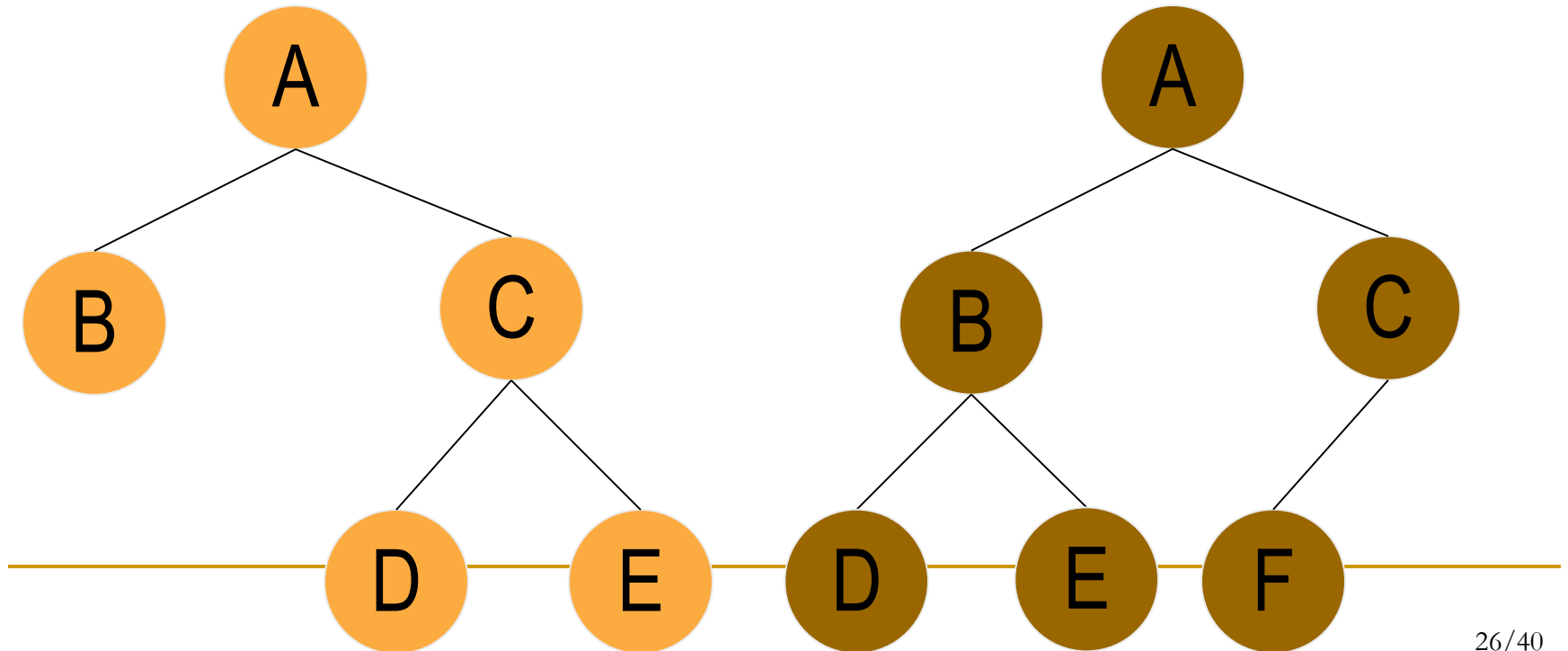
$$h = \log_d(N \cdot d - N + 1)$$

$$\text{para } d=2: h = \log_2(N + 1)$$

# Árvore Binária Quase Completa

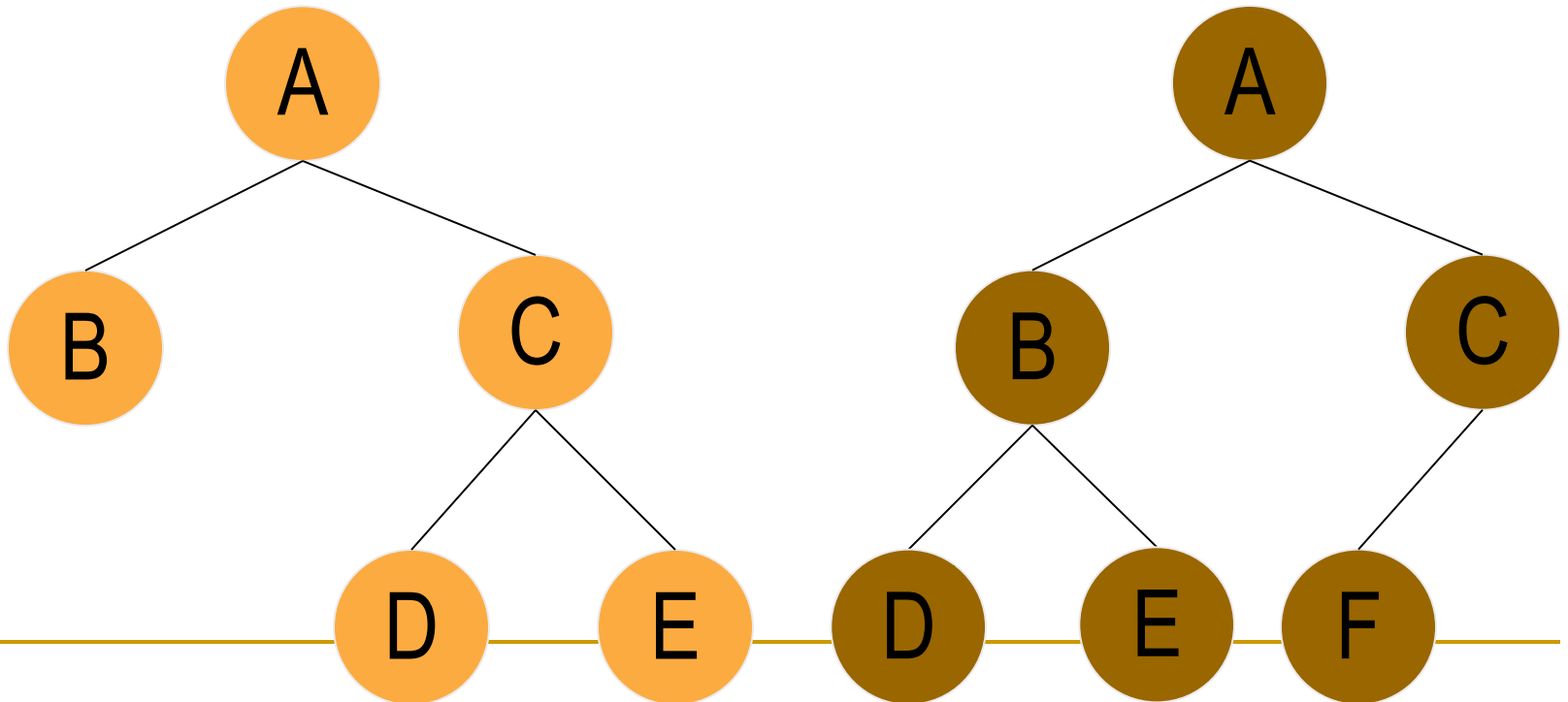
## ■ Árvore Binária Quase Completa

- Se a diferença de altura entre as sub-árvores de qualquer nó é no máximo 1.
- Como consequência, se a altura da árvore é **d**, cada nó folha está no nível **d** ou no nível **d-1**.



# Árvore Binária Balanceada

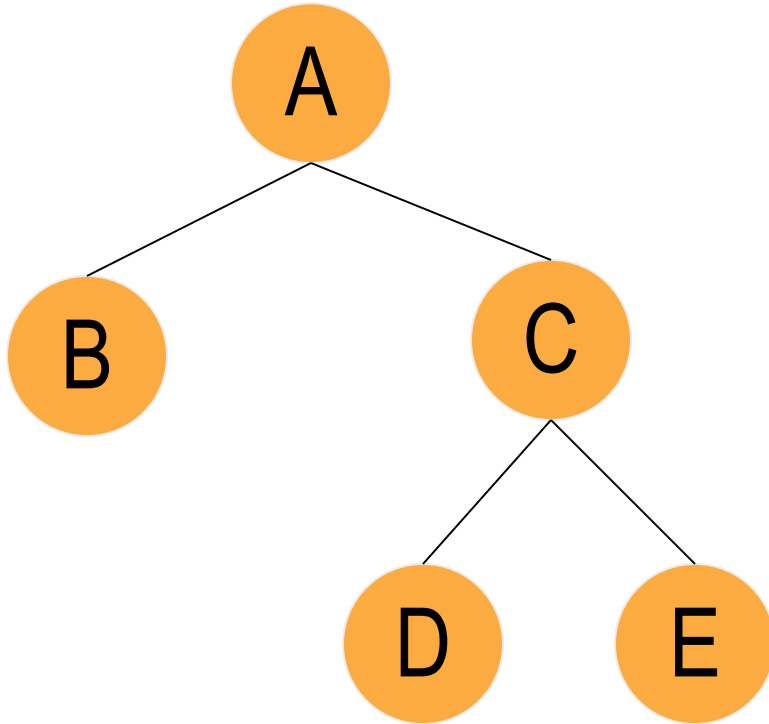
- **Árvore Binária Balanceada**
  - para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1



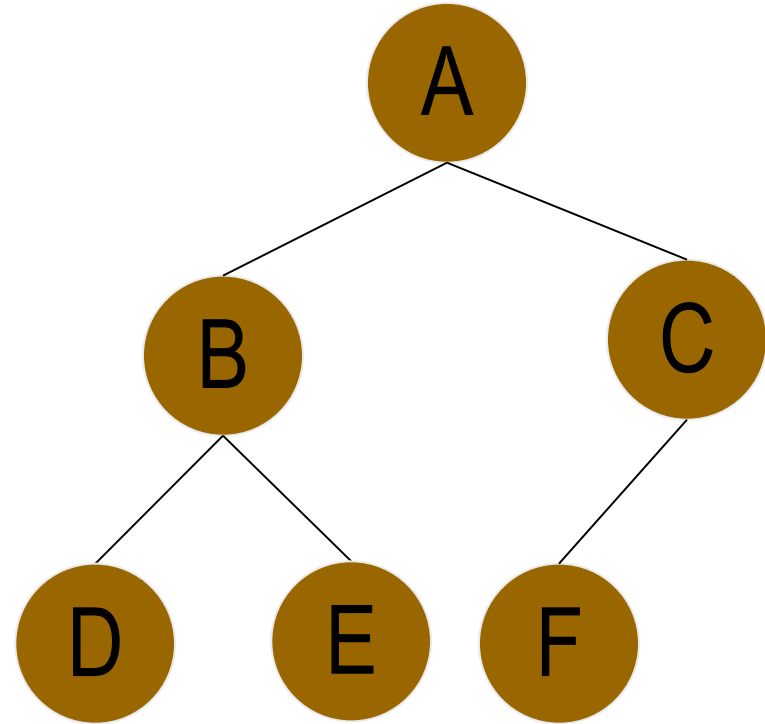
# Árvore Binária Perfeitamente Balanceada

- **Árvore Binária Perfeitamente Balanceada:** para cada nó, o número de nós de suas sub-árvores esquerda e direita difere em, no máximo, 1
- Toda AB Perfeitamente Balanceada é Balanceada, mas o inverso não é necessariamente verdade.
- Uma AB com N nós tem altura mínima se e só se for Balanceada.
- Se uma AB for Perfeitamente Balanceada então ela tem altura mínima.
  - Demonstre!!!!

# Exemplo



Árvore Balanceada



Árvore Perfeitamente Balanceada

---

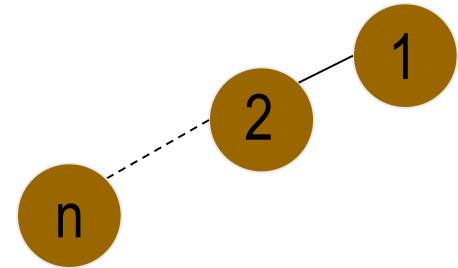
6 nós:  $h_{\min} = 3$

# Questões

- Qual a altura máxima de uma AB com  $n$  nós?

- Resposta:  $n$

- Árvore degenerada  $\equiv$  Lista



- Qual a altura mínima de uma AB c/  $n$  nós?

- Resposta: a mesma de uma AB Perfeitamente Balanceada com  $N$  nós

$N=1$ ;  $h=1$

$N=2,3$ ;  $h=2$

$N=4..7$ ;  $h=3$

$N=8..15$ ;  $h=4$

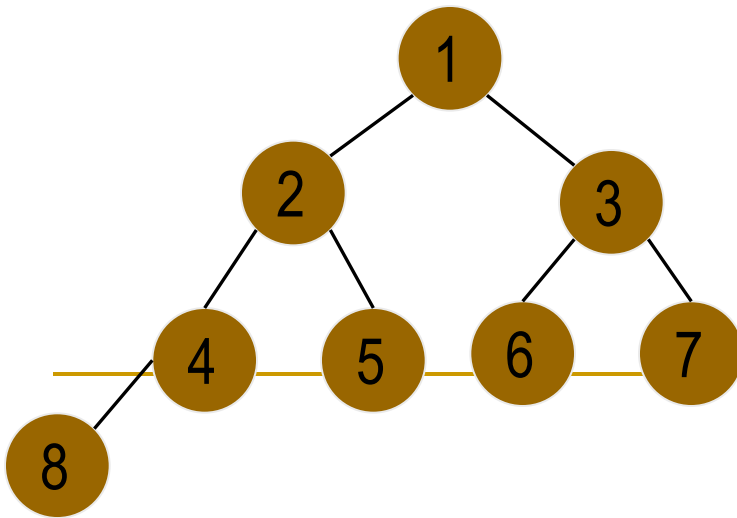
$$h_{\min} = \lfloor \log_2 N \rfloor + 1$$

(maior inteiro  $\leq \log_2 N$ ) + 1

ou

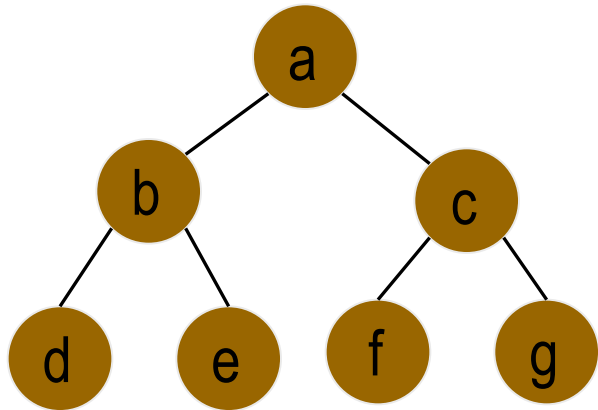
$$h_{\min} = \lceil \log_2 (N+1) \rceil$$

menor inteiro  $\geq \log_2 (N+1)$



# Implementação de AB Completa (alocação estática, seqüencial)

- Armazenar os nós, por nível, em um *array*



1	2	3	4	5	6	7	...
a	b	c	d	e	f	g	...

- Se um nó está na posição  $i$ , seus filhos diretos estão nas posições  $2i$  e  $2i+1$ 
  - Vantagem: espaço só p/ armazenar conteúdo; ligações implícitas
  - Desvantagem: espaços vagos se árvore não é completa por níveis, ou se sofrer eliminação

# Implementação de AB (dinâmica)

Para qualquer árvore, cada nó é do tipo

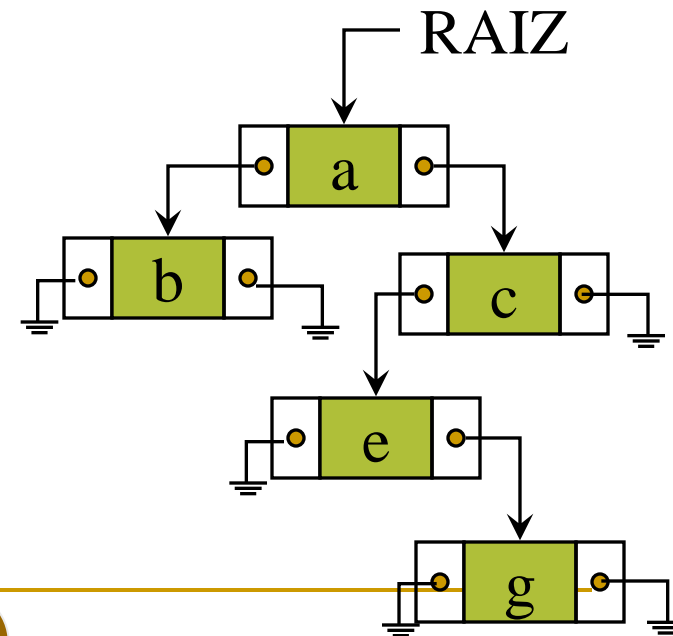
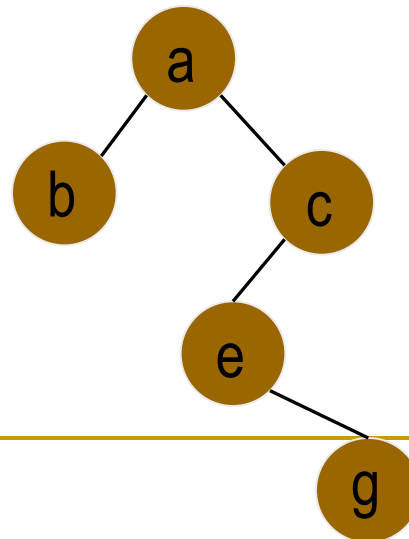


```
typedef struct no *pno;
```

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

```
typedef pno tree;
```

```
tree raiz;
```





# Operações do TAD AB

```
void define (tree t){  
    t = NULL;    /*Cria AB vazia*/  
}
```

```
void cria_raiz(tree t, tipo_elem item){  
    pno no = malloc(sizeof(no));  
    no->esq = NULL;  
    no->dir = NULL;  
    no->info = item;  
    t = no;  
}
```

```
boolean vazia(tree t){  
    return (t == NULL);  
}
```

# Função recursiva para calcular altura de uma árvore

```
int altura(tree r){  
  
    if (r == NULL)  
        return 0;  
  
    int altE = altura(r->esq);  
    int altD = altura(r->dir);  
  
    if (altE > altD)  
        return (altE + 1);  
  
    return (altD + 1); /*altura = max(altE, altD) + 1*/  
  
}
```

# Função recursiva para verificar se uma AB é balanceada

```
boolean balanceada(tree r){

    if (r == NULL)
        return true;
    else
        if (r->esq == NULL && r->dir==NULL) /* r não tem filhos */
            return true;
        else
            if (r->esq!=NULL && r->dir!=NULL) /* r tem ambas subárvores não-nulas */
                {int dif = altura(r->esq) - altura(r->dir);
                 return (balanceada(r->esq) && balanceada(r->dir) && ((dif
==0) || (dif ==1) || (dif == -1)));} /* recursão */
            else
                if (r->esq != NULL) /* tem um único filho - `a esquerda */
                    return (altura(r->esq) == 1);
                else /* tem um único filho - `a direita */
                    return (altura(r->esq) == 1);
    }
}
```

# Função recursiva para calcular o número de nós de uma AB

```
int numeronos(tree r) {  
  
    if (r == NULL)  
        return 0;  
  
    int nE = numeronos(r->esq);  
    int nD = numeronos(r->dir);  
  
    return (nE + nD + 1);  
}
```

# Função recursiva para verificar se uma AB é perfeitamente balanceada

```
boolean perfbalanceada(tree r){

    if (r == NULL)
        return true;
    else
        if (r->esq == NULL && r->dir==NULL) /* r não tem filhos */
            return true;
        else
            if (r->esq!=NULL && r->dir!=NULL) /* r tem ambas subárvores não-nulas */
                {int dif = numeronos(r->esq) - numeronos(r->dir);
                 return (balanceada(r->esq) && balanceada(r->dir) && ((dif ==0)
|| (dif ==1) || (dif == -1)));} /* recursão */
            return(perfbalanceada(r->esq) && perfbalanceada(r->dir);/*recursão*/
        else
            if (r->esq != NULL) /* tem um único filho - `a esquerda */
                return (numeronos(r->esq) == 1);
            else /* tem um único filho - `a direita */
                return (numeronos(r->esq) == 1);
    }
```

```
/* Função p/ adicionar um filho à direita de um nó, cujo  
   ponteiro é dado (pai). Se o nó não possui filho à  
   direita, então cria esse filho com conteúdo "item" */
```

```
boolean insere_dir(tree pai, tipo_elem item){
```

```
    if (pai == NULL)  
        return FALSE;
```

```
    if (pai->dir != NULL) {  
        printf("já tem filho à direita");  
        return FALSE;  
    }
```

```
    tree no = malloc(sizeof(no));  
    no->esq = NULL;  
    no->dir = NULL;  
    no->info = item;  
    pai->dir = no;  
    return TRUE;
```

**OU**

```
    cria_raiz(pai->dir,  
             item);  
    return TRUE;
```

```
}
```

# AB - Percursos

- **Objetivo:** Percorrer uma AB ‘visitando’ cada nó uma única vez. Um percurso gera uma seqüência linear de nós, e podemos então falar de nó **predecessor** ou **sucessor** de um nó, segundo um dado percurso.
- Não existe um percurso único para árvores (binárias ou não): diferentes percursos podem ser realizados, dependendo da aplicação.
- **Utilização:** imprimir uma árvore, atualizar um campo de cada nó, procurar um item, etc.

---

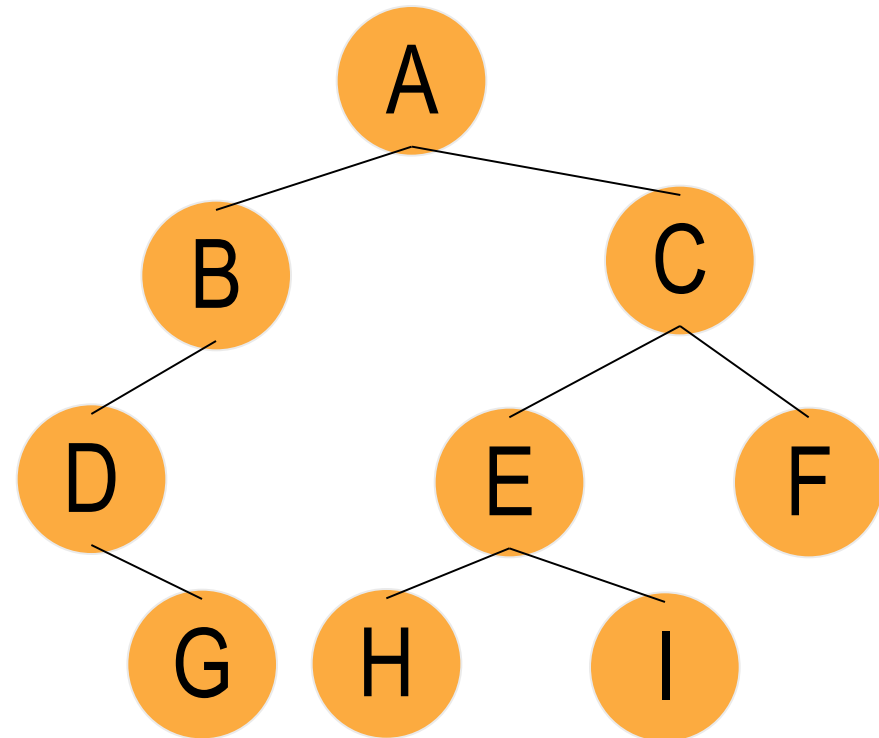
# AB – Percursos em Árvores

- 3 percursos básicos para AB's:
  - pré-ordem (Pre-order)
  - in-ordem (In-order)
  - pós-ordem (Post-order)
- A diferença entre eles está, basicamente, na ordem em que cada nó é alcançado pelo percurso
  - “Visitar” um nó pode ser:
    - Mostrar (imprimir) o seu valor;
    - Modificar o valor do nó;
    - ...



# AB - Percurso Pré-Ordem

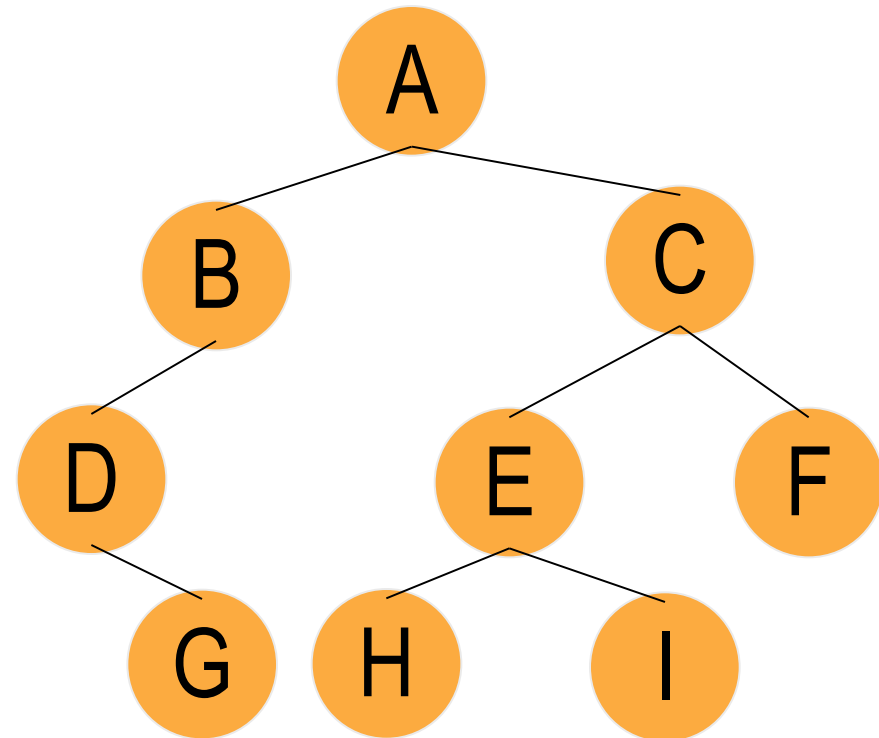
```
void pre_ordem(tree raiz){  
    if (raiz != NULL){  
        visita(raiz);  
        pre_ordem(raiz->esq);  
        pre_ordem(raiz->dir);  
    }  
}
```



Resultado: ABDGCEHIF

# AB - Percurso Em-Ordem

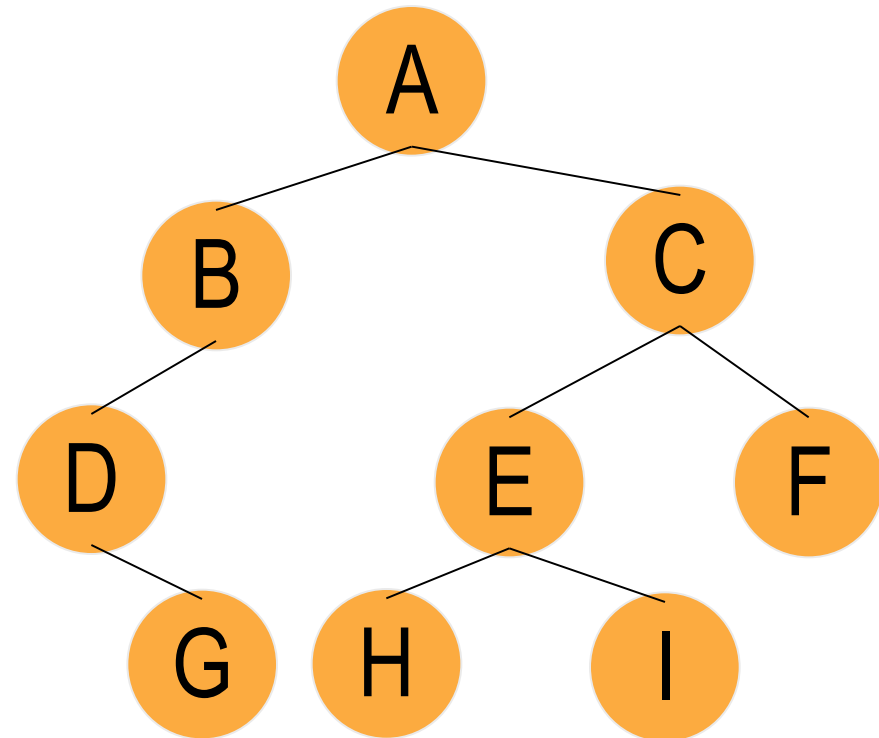
```
void in_ordem(tree raiz){  
    if (raiz != NULL){  
        in_ordem(raiz->esq);  
        visita(raiz);  
        in_ordem(raiz->dir);  
    }  
}
```



Resultado: DGBAHEICF

# AB - Percurso Pós-Ordem

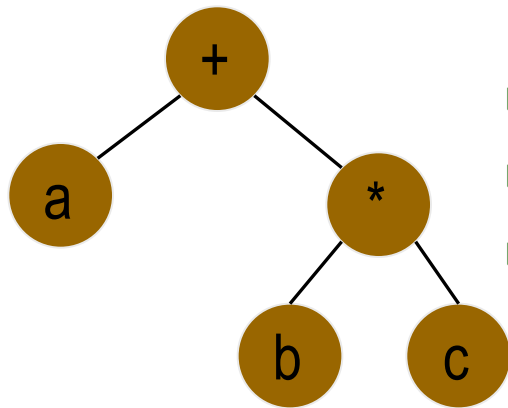
```
void pos_ordem(tree raiz){  
    if (raiz != NULL){  
        pos_ordem(raiz->esq);  
        pos_ordem(raiz->dir);  
        visita(raiz);  
    }  
}
```



Resultado: GDBHIEFCA

# AB – Percursos

- Percurso para expressões aritméticas



- Pré-ordem:  $+a*bc$
- In-ordem:  $a+(b*c)$
- Pós-ordem:  $abc*+$

- Em algoritmos iterativos utiliza-se uma pilha ou um campo a mais em cada nó para guardar o nó anterior (pai)

# Exercícios

/\* Função para calcular o nível de um nó. Dado o valor de um elemento, se ele está na árvore, retorna seu nível, retorna null c.c. OBS.: Nivel da raiz = 1\*/

```
int nivel(tree t, tipo_elem item){  
    int n;  
    boolean achou = FALSE;  
    n = 0;  
  
    travessia(t, &n, item, &achou);  
    return n;  
}
```

---

```
/*percorre a árvore com raiz em ptr em Pré-ordem,  
procurando pelo item dado e calculando e retornando seu  
nível na variável n*/
```

```
void travessia(tree ptr, int *niv, tipo_elem item,  
                boolean *achou){  
  
    if (ptr != NULL) {  
        niv ++;  
        if (ptr->info == item)  
            *achou == TRUE;  
  
        travessia(ptr->esq, niv, item, achou);  
        if (!*achou) {  
            travessia(ptr->dir, niv, item, achou);  
            if (!*achou)  
                niv --;  
        }  
    }  
}
```

---

# Exercícios

- Uma árvore binária completa é uma árvore estritamente binária?
- Uma árvore estritamente binária é uma árvore binária completa?
- Escreva um procedimento recursivo que calcula a altura de uma AB.
- Verifique o que faz o procedimento enigma (a seguir)

```
void enigma(tree raiz){
    pilha *P;
    tree x, pont;

    define_pilha(P); /*P é uma pilha*/
    pont = raiz;
    boolean acabou = (raiz == NULL);

    while (!acabou){
        while (pont != NULL){
            visita(pont);
            push(pont, P); /*insere pont na pilha P*/
            pont = pont->esq;
        }
        if (!pilha_vazia(P)){
            x = topo(P); /*recupera o conteúdo do topo de P*/
            pont = x->dir;
            pop(P); /*retira o elemento no topo da pilha*/
        } else
            acabou = TRUE;
    }
}
```



# Procedimento recursivo p/ destruir árvore, liberando o espaço alocado (percurso em pós-ordem)

```
void destruir(tree r){
    if (!vazia(r)) {
        destruir(r->esq);
        destruir(r->dir);
        free(r);
    }

    r = NULL;
}
```