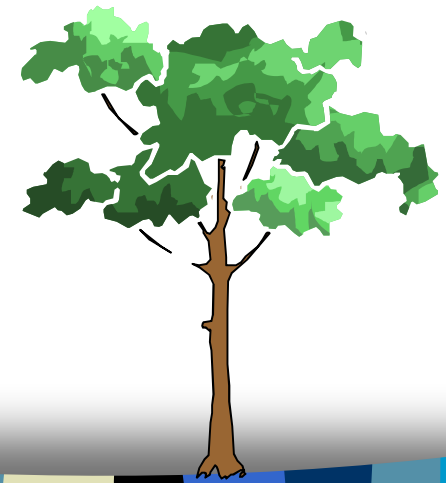


SCC122 – Estruturas de Dados



# Árvores & Árvores Binárias

Prof. Roseli A. F. Romero

**Material Original:**

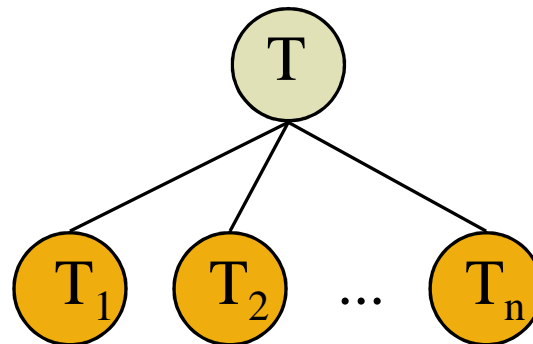
Walter Aoiama Nagai;

Maria das Graças Volpe Nunes;

# Definições

Árvore  $T$  é um conjunto finito de elementos denominados nós ou vértices, tais que:

- $T = \emptyset$ , a árvore é dita vazia;
- Existe um nó especial, denominado **raiz** de  $T$ ; os nós restantes constituem um único conjunto vazio ou são divididos em  $m \geq 1$  conjuntos **disjuntos** não vazios  $(T_1, T_2, \dots, T_n)$ , cada qual, por sua vez é uma árvore;
- $T_1, T_2, \dots, T_n$  são chamadas sub-árvores de  $T$ ;
- Um nó sem subárvores é denominado **nófolha** ou simplesmente **folha**.

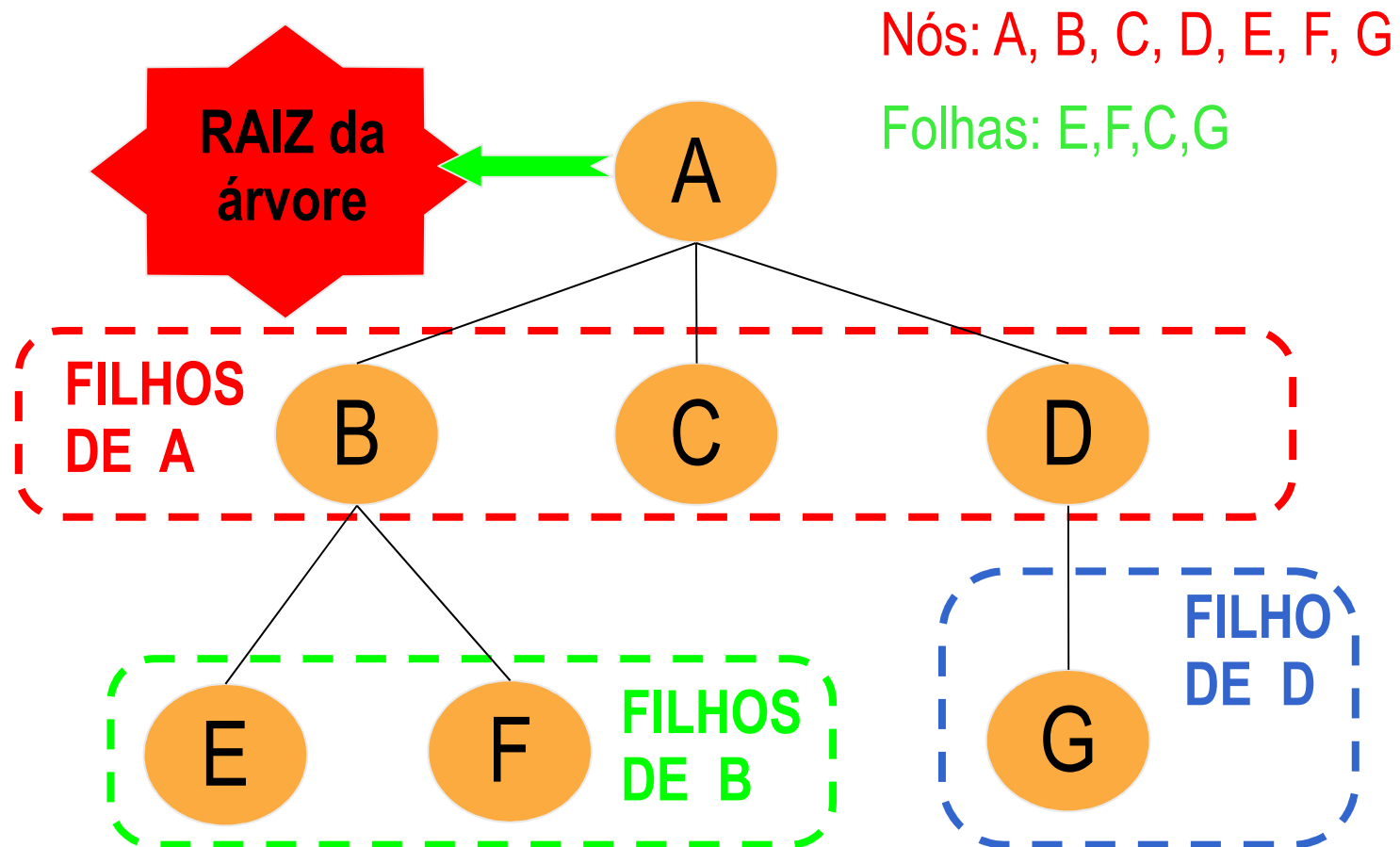




## Definições (cont.)

- Uma Árvore é capaz de representar estruturas hierárquicas, como relações de pai, filho, irmãos, etc.
- Se um nó  $X$  é a raiz de uma árvore e um nó  $Y$  é a raiz de sua subárvore direita ou subárvore esquerda, então  $X$  é o **PAI** de  $Y$  e  $Y$  é o **FILHO** de  $X$ .

# Definições (cont.)

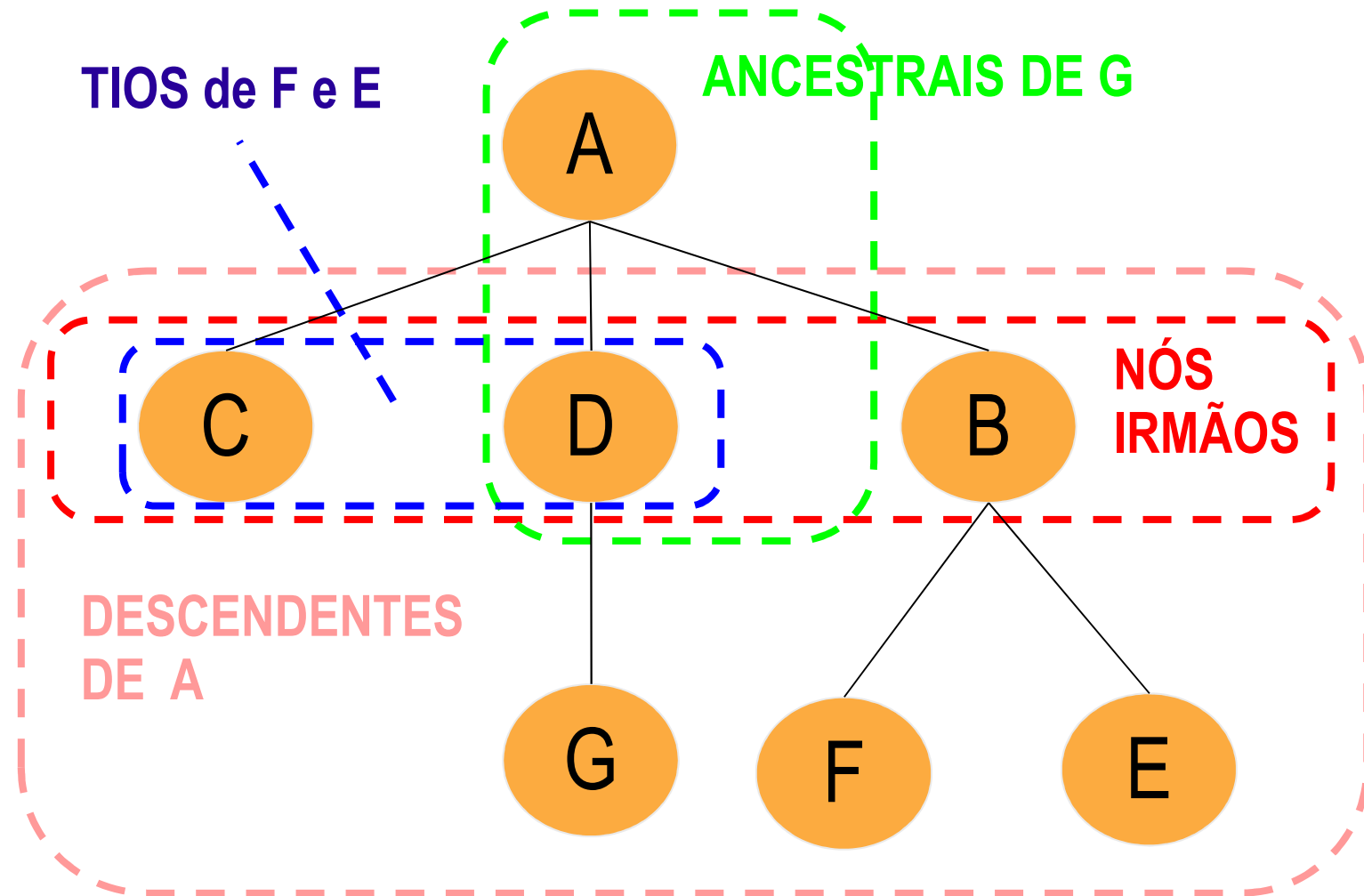




## Definições (cont.)

- O nó  $X$  é um **ANCESTRAL** do nó  $Y$  (e  $Y$  é **DESCENDENTE** de  $X$ ) se  $X$  for o **PAI** de  $Y$  ou então se  $X$  for o **PAI** de algum **ANCESTRAL** de  $Y$ .
- Dois nós são **IRMÃOS** se forem filhos do mesmo pai.
- Os nós  $Y_1, Y_2, \dots, Y_j$  são irmãos e se 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 nó raiz está no nível 1;
  - Os outros nós possuem um nível a mais que o nível de seu nó PAI;
- A **PROFUNDIDADE** de uma árvore significa o nível máximo de qualquer FOLHA na árvore, ou o tamanho do percurso mais distante da RAIZ até qualquer FOLHA.

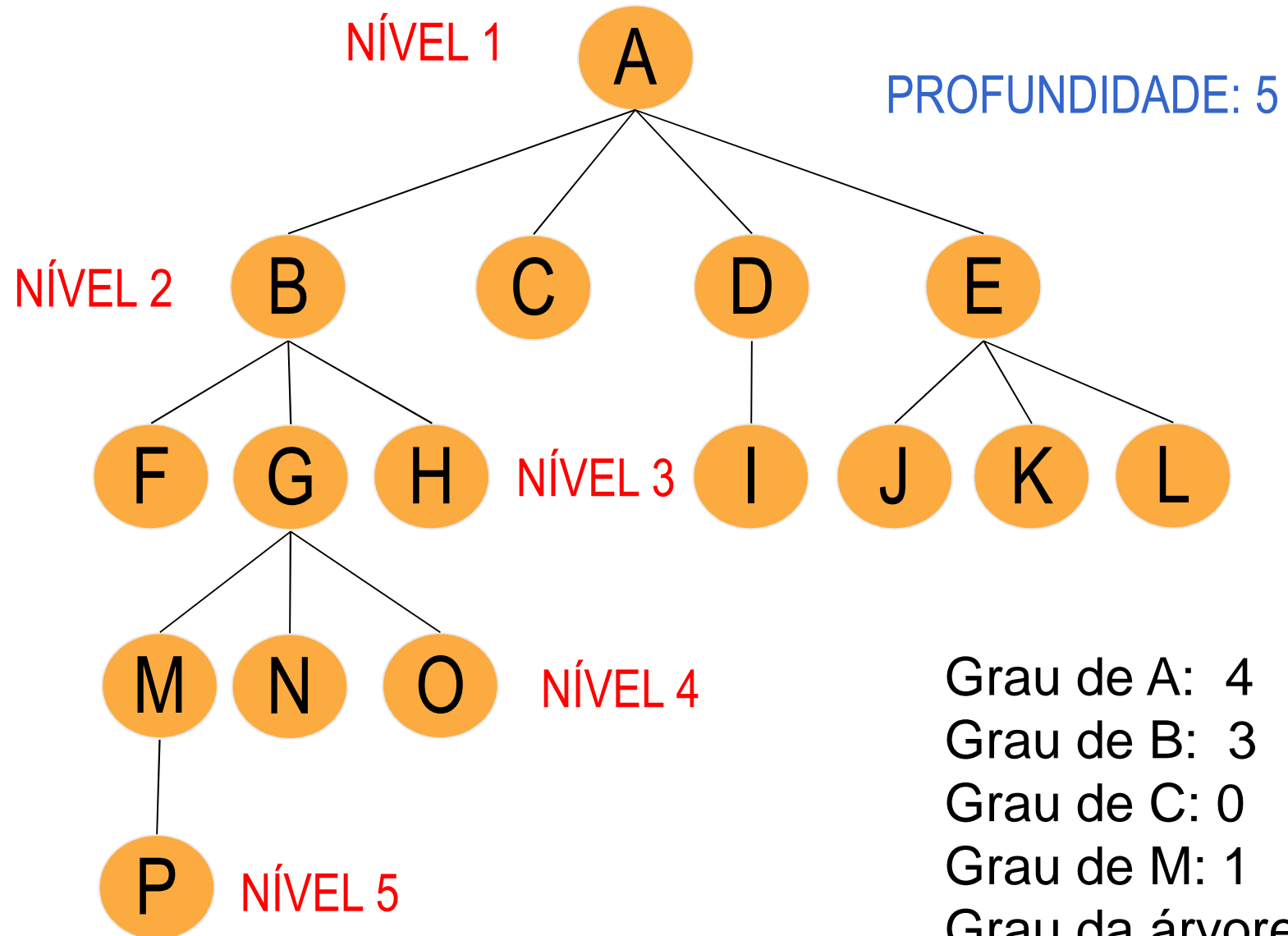


## Conceitos (cont.)

- O **GRAU** de um nó  $X$  de uma árvore é determinado pelo número de filhos do nó  $X$ ;
- O **GRAU de uma árvore  $T$**  é o grau máximo entre todos os seus nós;



# Conceitos (cont.)



Grau de A: 4

Grau de B: 3

Grau de C: 0

Grau de M: 1

Grau da árvore: 4

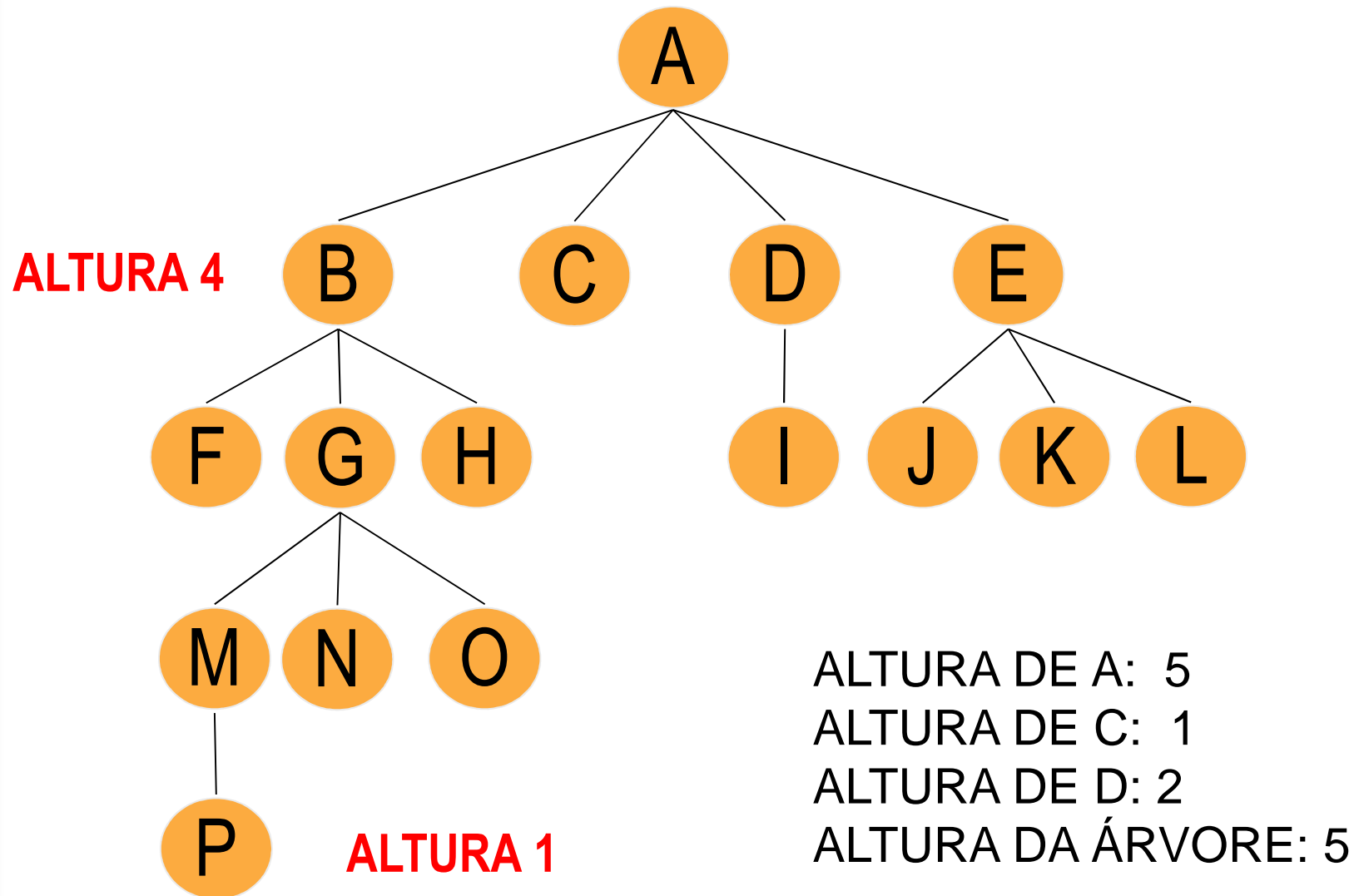
*Roseli A. F. Romero*



## Conceitos (cont.)

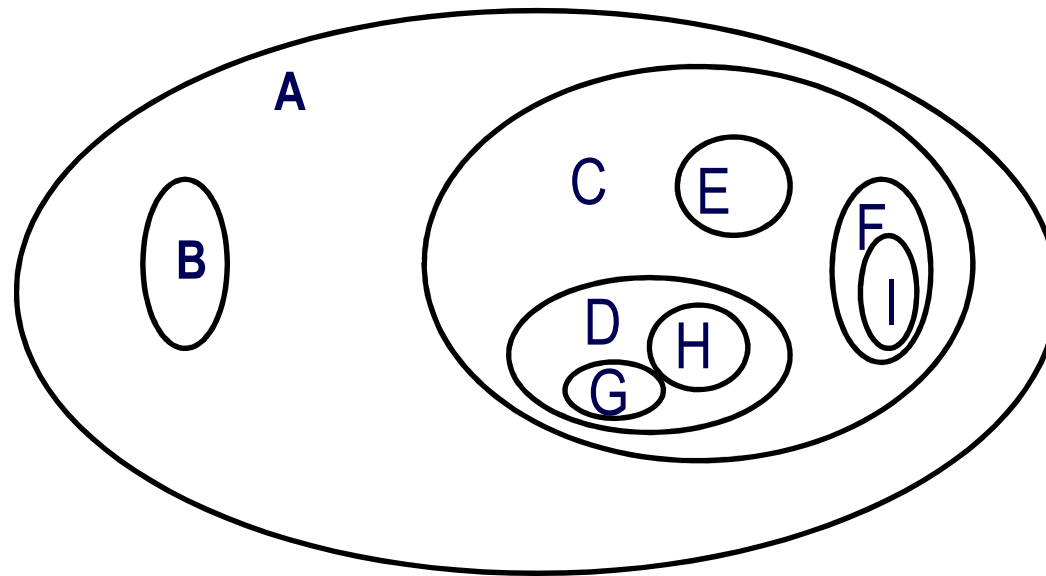
- A **ALTURA** de um nó é o número de nós no maior caminho do nó (inclusive) até um de seus descendentes.
- Os nós **FOLHA** têm sempre altura igual a 1;
- A **ALTURA** de uma árvore  $X$  é igual a altura máxima de seus nós.
- Representa-se a altura de  $X$  por  $h(X)$  e a altura de uma subárvore com raiz  $y$  por  $h(y)$ .

# Conceitos (cont.)



## Conceitos (cont.)

- Outras formas de representação:
  - Representação por parênteses aninhados
    - ( A (B) ( C (D (G) (H)) (E) (F (I)) ) ) )
  - Representação com diagramas





## Árvore Binárias (AB)

Uma Árvore Binária (AB)  $T$  é um conjunto finito de elementos denominados nós ou vértices, tal que:

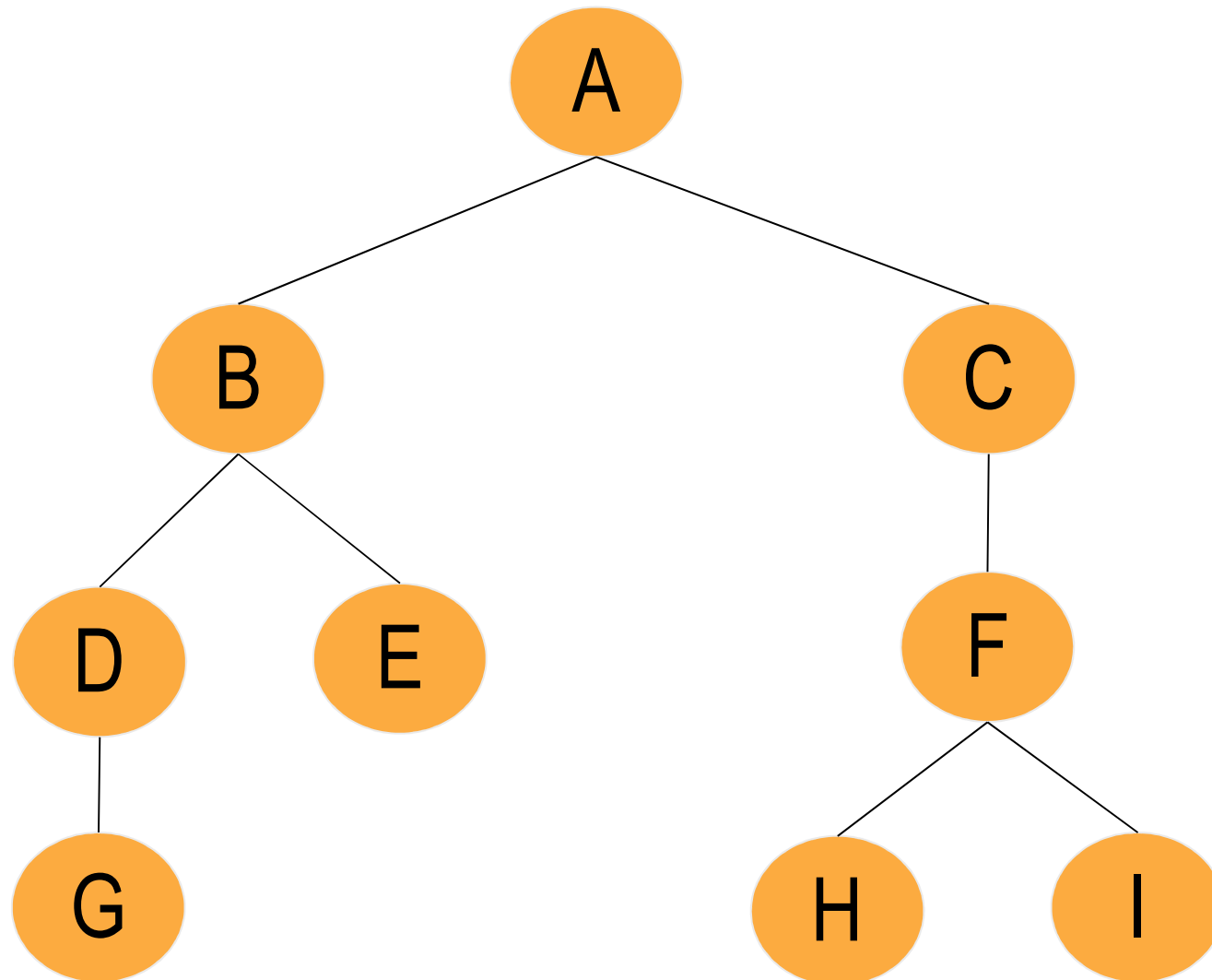
- $T = \emptyset$ , a árvore é dita vazia, ou
- Existe um nó especial (inicial) chamado raiz de  $T$ , e os nós restantes em dois subconjuntos distintos  $T_E$  e  $T_D$ , a subárvore esquerda e a subárvore direita de  $T$ , respectivamente, as quais também são árvores binárias.



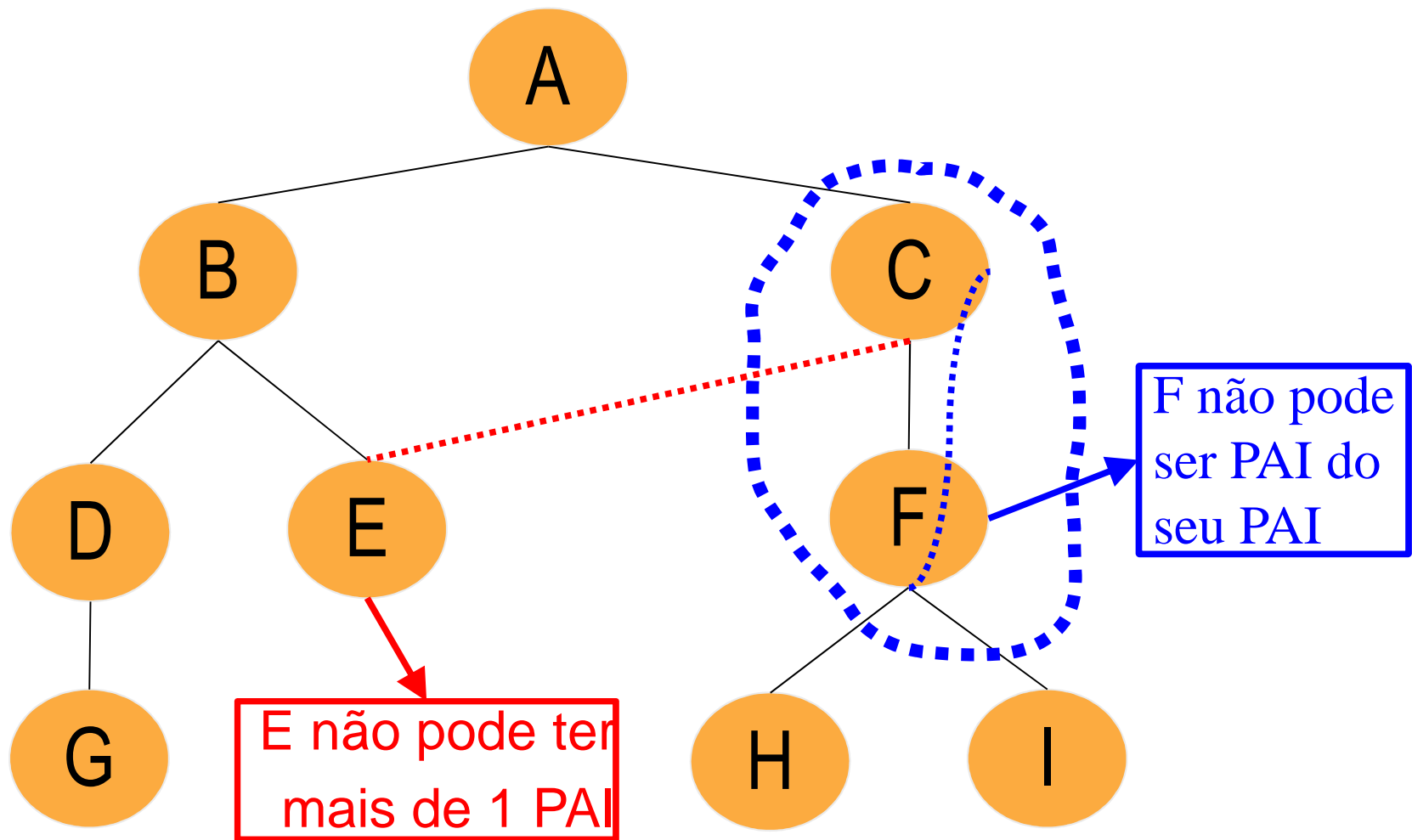
## Árvore Binárias (AB) (cont.)

- 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  $T_r^E$  e  $T_r^D$  são as subárvores esquerda e direita de  $T$ , respectivamente.

# Árvore Binárias (AB) (exemplo)



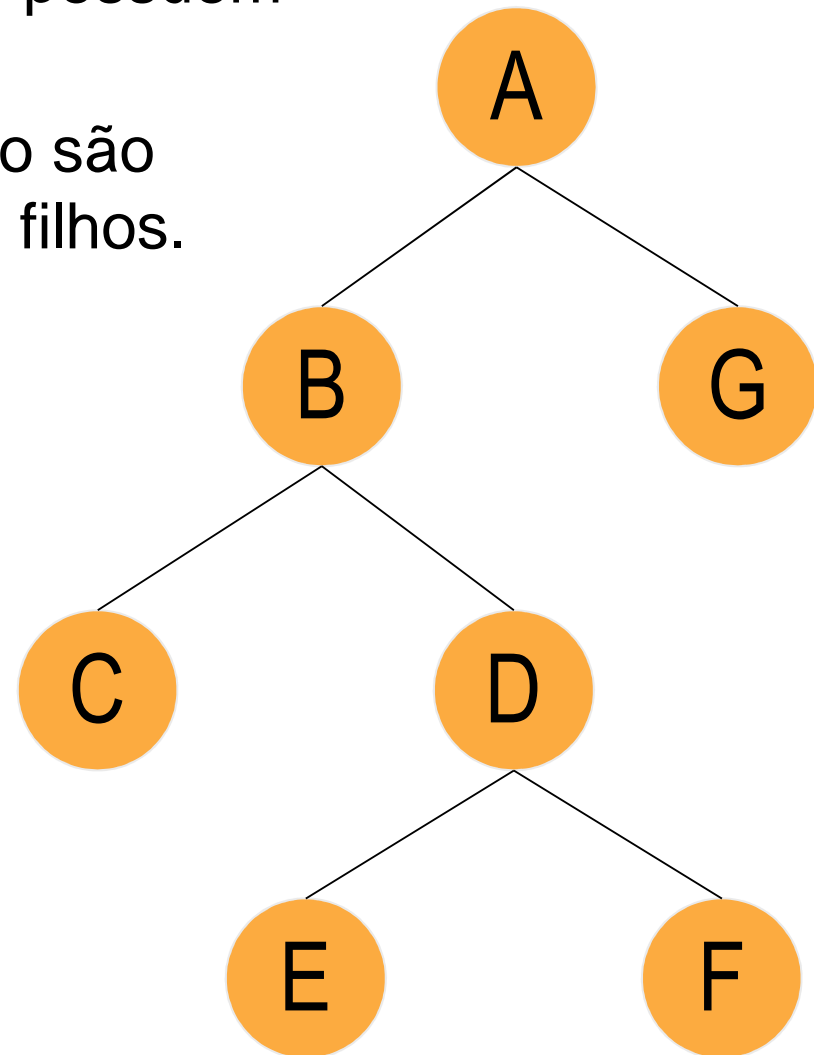
# Árvore Binárias (AB) (cont.)





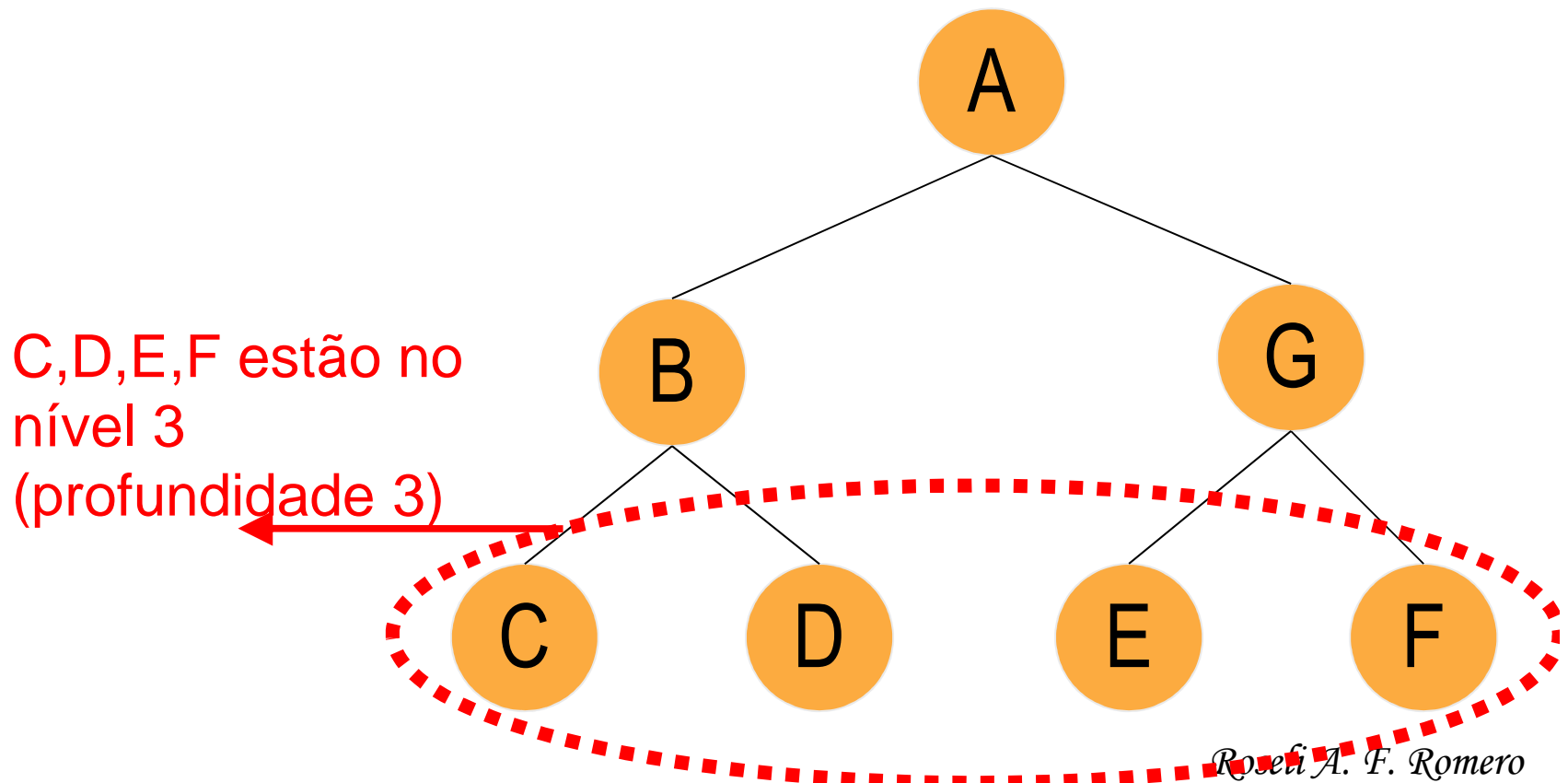
# Árvore Estritamente Binária

- Uma **Árvore Estritamente Binária** é aquela que **tem nós** que possuem **0** ou **dois** filhos.
- Nós interiores (nós que não são folhas) possuem sempre 2 filhos.



# Árvore Binária Completa

- Uma **Árvore Binária Completa** é aquela que é estritamente binária de nível  $d$  e com todos os **nós-folhas** no **mesmo nível  $d$** .





## Árvore Binária Completa (cont.)

- Pode-se calcular o número total de nós (NT) de uma árvore binária completa, somando-se o número de nós em cada nível da árvore.
- Por exemplo, uma AB completa com profundidade 3 possui 7 nós, considerando a variação de níveis entre 0-2:
  - Nível 1:  $\Rightarrow 1$
  - Nível 2:  $\Rightarrow 2$
  - Nível 3:  $\Rightarrow 4$
  - $NT = 1 + 2 + 4 = 7$

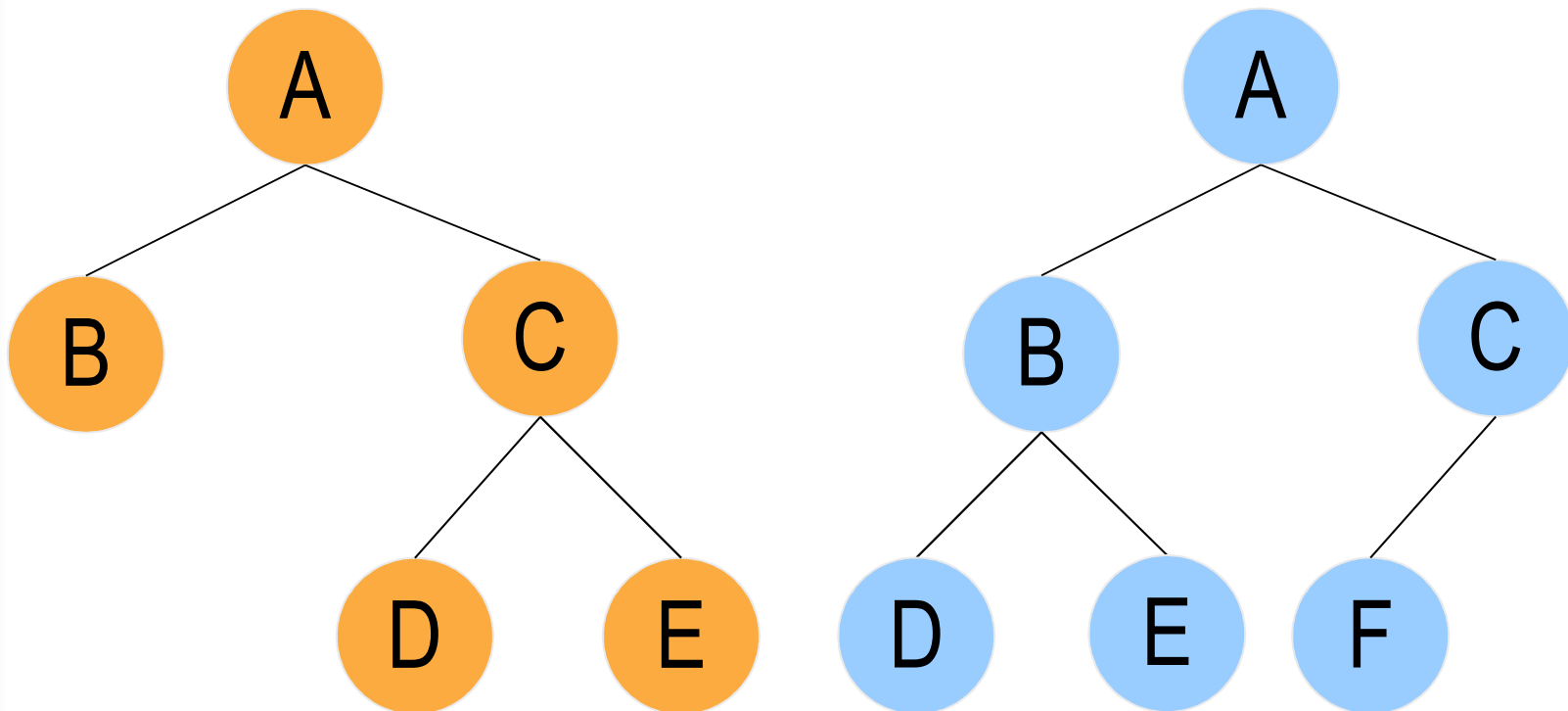


## Árvore Binária Quase Completa

- Uma Árvore Binária Quase Completa é aquela na qual cada nó pode estar no nível  $d$  (profundidade) ou no nível  $d-1$ , ou seja, a diferença de nível entre as subárvores deve ser em 1 nível.

# Árvore Binária Balanceada

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





## Árvore Binária Perfeitamente Balanceada

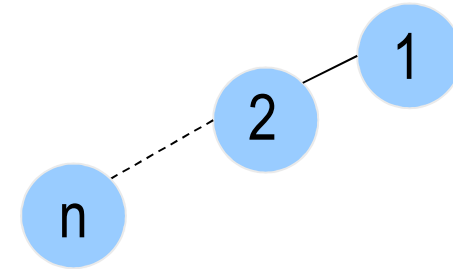
- Uma **Árvore Binária Perfeitamente Balanceada** (ou de altura mínima) é aquela cujo número de nós de suas sub-árvores esquerda e direita diferem, no máximo em 1.
- Toda AB Perfeitamente Balanceada é Balanceada, mas não vale o inverso.

# Questões

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

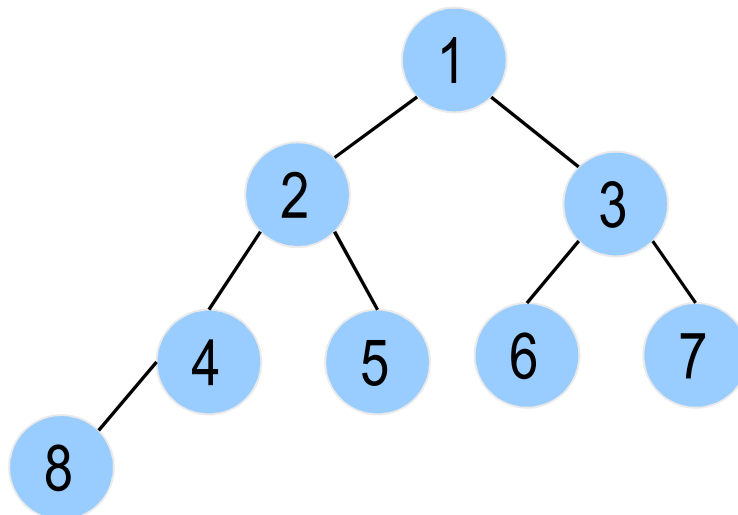
- Resposta:  $n$

Árvore degenerada  $\equiv$  Lista



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

- Resposta: É a mesma de uma AB perfeitamente Balanceada com  $n$  nós.



$$n=1; \quad h=1$$

$$n=2,3; \quad h=2$$

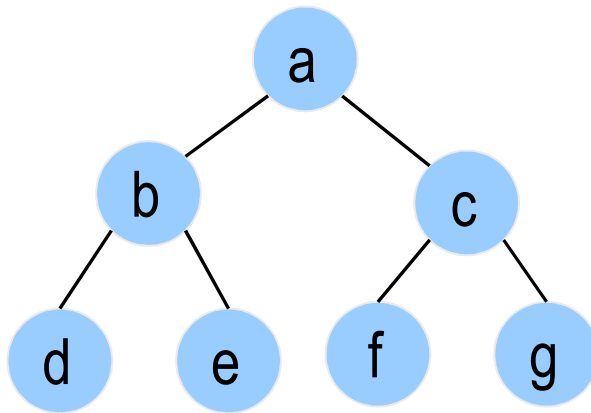
$$n=4..7; \quad h=3$$

$$n=8..15; \quad h=4$$

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

# Implementação de AB (seqüencial)

- Armazenar os nós, por nível, num array:



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

- Assim: se um nó ocupa a posição  $i$ , então seus filhos diretos estão nas posições  $2i$  e  $2i+1$ .
- Vantagem: espaço só para armazenar conteúdo; ligações implícitas.
- Desvantagens: espaços vazios se a árvore não for completa por níveis, ou se sofrer eliminação.

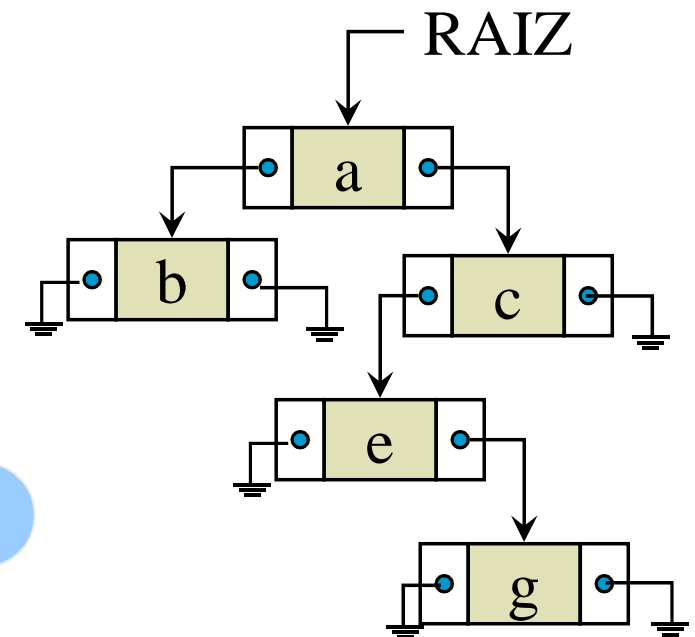
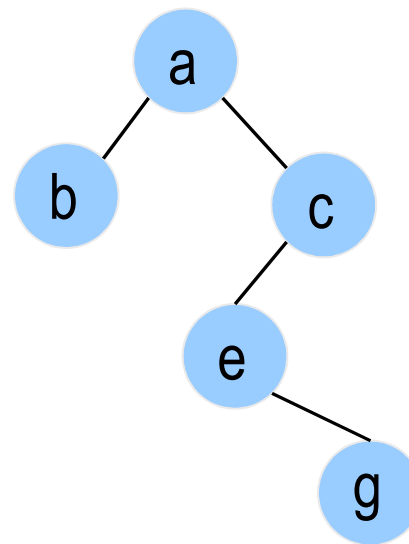


# Implementação de AB (dinâmica)

Para qualquer árvore, cada nó é do tipo:



```
typedef struct _no bintree;  
Struct _no {  
    Tipo_elem data;  
    bintree *fesq, *fdir;  
    bintree *Tree;
```



# Operações do TAD AB

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

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

```
int vazia(bintree *t){  
    return (t==NULL); }
```



```
{adicionar um filho à direita de um dado nó (item_pai):
```

1. Busca nó contendo "item\_pai"
2. Se ele não possui filho à direita, então cria esse filho com conteúdo "item".}

```
void adiciona_dir(bintree *t, tipo_elem item_pai,item){  
    bintree *pai, *no;  
    pai = Search_tree(t,item_pai);  
    if (pai != NULL)  
        if (pai->fdir != NULL)  
            printf("erro já tem filho à direita");  
        else {  
            no = (bintree *) malloc(sizeof(bintree));  
            no->fesq = NULL;  
            no->fdir = NULL;  
            no->data = item;  
            pai->fdir = no;  
        }  
};
```



## AB - Percursos

- **Objetivo:** Percorrer uma AB visitando cada nó uma única vez. Um percurso gera uma sequência 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, remover um item, buscar por um item, ...
- Existem 3 percursos básicos para AB's:
  - Percurso Pré-Ordem (Pre-Order);
  - Percurso Em-Ordem (In-Order);
  - Percurso Pós-Ordem (Post-Order);

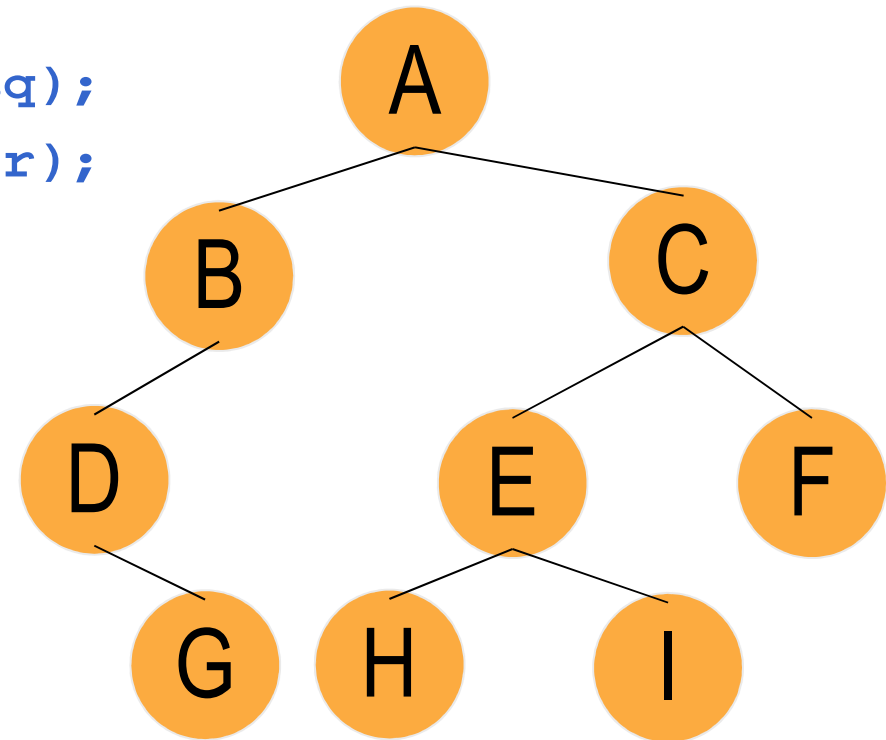


## AB - Percursos

- A diferença entre os 3 percursos é, basicamente:
  - A ordem em que os nós são “visitados”.
  - “Visitar” um nó pode ser:
    - Mostrar algum valor;
    - Modificar o valor do nó;
    - Remover o nó;
    - entre outras operações.

# AB - Percurso Pré-Ordem

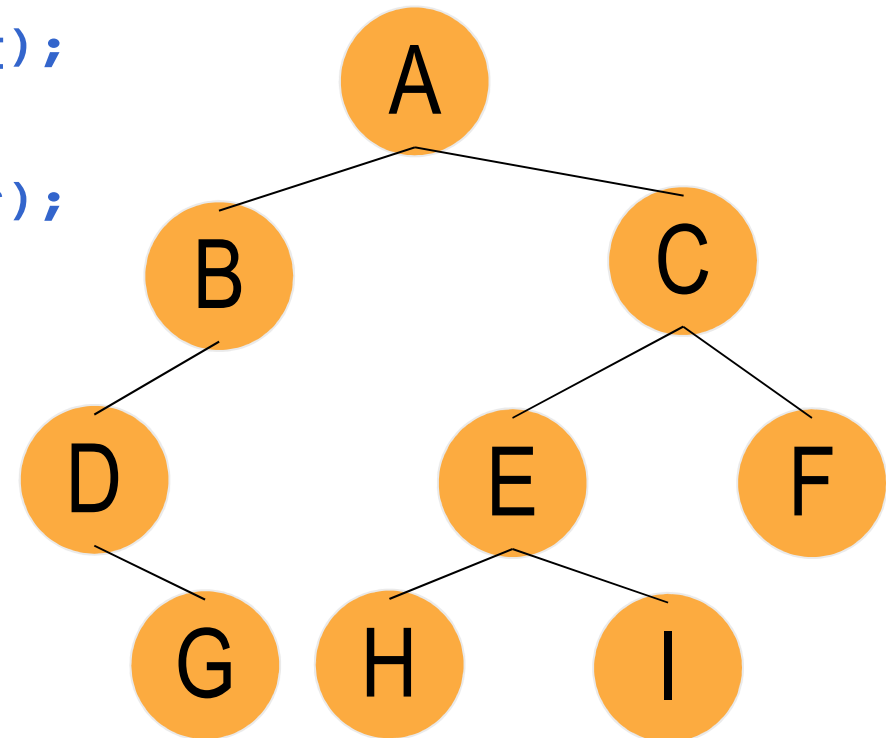
```
void pre_ordem(bintree *raiz){  
    if (raiz != NULL) {  
        visita(raiz);  
        pre_ordem(raiz->fesq);  
        pre_ordem(raiz->fdire);  
    }  
}
```



- Resultado: ABDGCEHIF

# AB - Percurso Em-Ordem

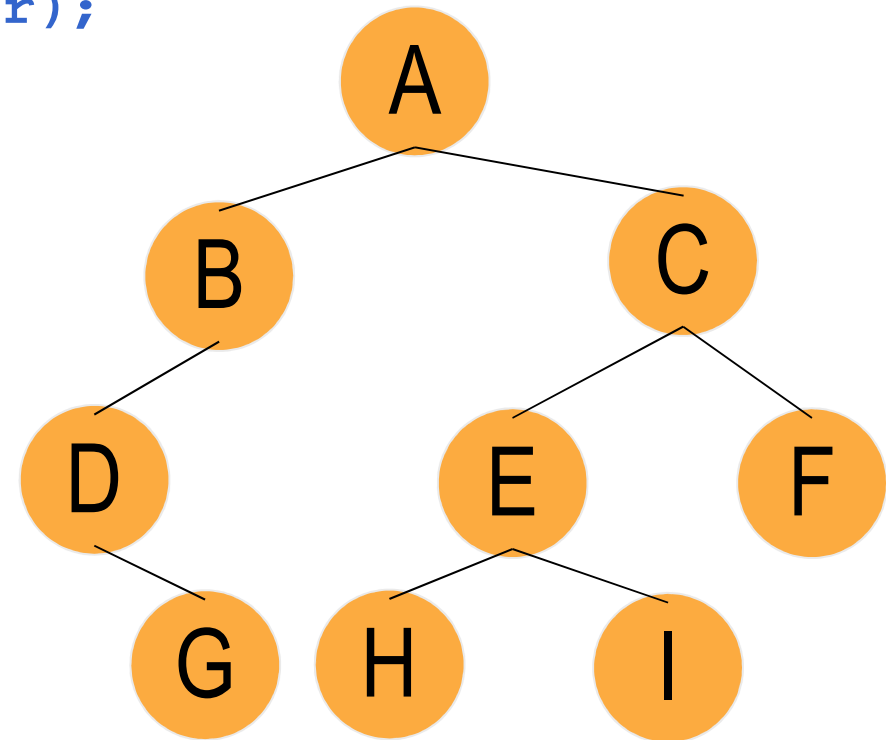
```
void em_ordem(bintree *raiz){  
    if (raiz != NULL){  
        em_ordem(raiz->fesq);  
        visita(raiz);  
        em_ordem(raiz->fdire);  
    }  
}
```



- Resultado: DGBAHEICF

# AB - Percurso Pós-Ordem

```
void pos_ordem(bintree *raiz){  
    if (raiz != NULL)  
        pos_ordem(raiz->fesq);  
        pos_ordem(raiz->fdire);  
        visita(raiz);  
    }  
}
```

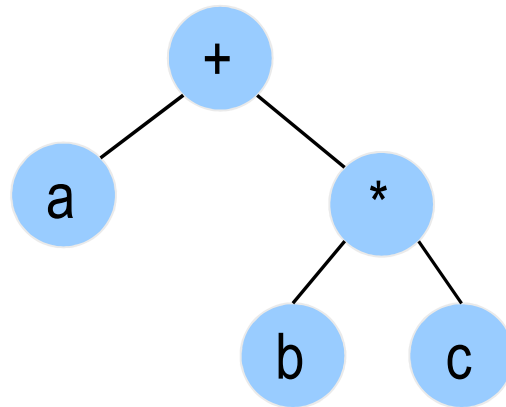


- Resultado: GDBHIEFCA



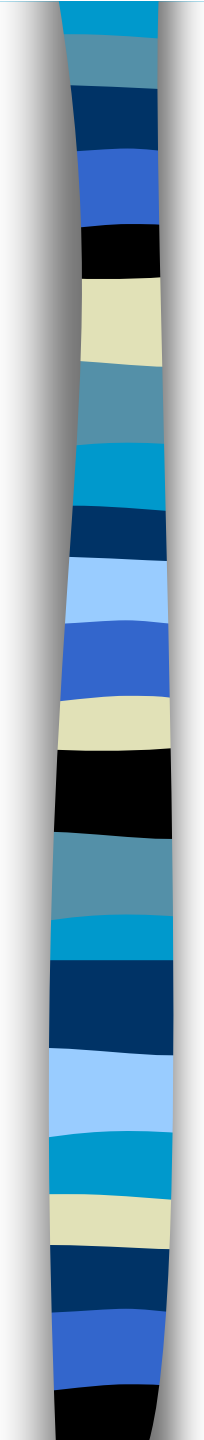
## AB – Percursos

- Em expressões utiliza-se o percurso em pós-ordem .



- Pré-ordem:  $+a^*bc$
- Em-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).



{calcular o nível de um nó.  
Dado o valor de um elemento, se ele se encontra na  
árvore, retorna seu nível e NULL caso contrário. OBS.:  
Nível da raiz=1}

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



{percorre a árvore ptr em Pre-ordem procurando por item e calculando e retornando seu nível}

```
void travessia(bintree *ptr, int niv, tipo_elem  
item, int achou ) {
```

```
    if (ptr != NULL)
```

```
    {
```

```
        niv++;
```

```
        if (ptr->data == item)
```

```
            achou = 1;
```

```
        else {
```

```
            travessia(ptr->fesq,niv,item,achou);
```

```
            if (!achou)
```

```
                travessia(ptr->fdire,niv,item,achou);
```

```
            if (!achou)
```

```
                niv --;
```

```
        }
```

```
    }
```

```
}
```



## Exercícios

- Crie uma árvore, com as relações existentes sobre sua família. Relações como: pai, irmãos, tios, avôs.
- Uma árvore binária completa é uma árvore estritamente binária?
- Uma árvore estritamente binária é uma árvore binária completa?
- Escreva a fórmula para calcular o número total de nós de uma AB completa dada sua profundidade.
- Escreva o procedimento recursivo que calcula a altura de uma AB.
- Verifique o que faz o procedimento **enigma** e faça mudanças caso seja necessário.

# Exercícios

```
void enigma(bintree *raiz){
    pilha S;
    bintree *x,*pont;

    criar(S); {S é uma pilha}
    pont = raiz; acabou = (raiz==NULL);
    while (!acabou){
        while (pont != NULL){
            visita(pont);
            push(pont,S); {insere pont na pilha S}
            pont = pont->esq;
        }
        if (!vazia(S)){
            x = topo(S); {lê o conteúdo do topo da pilha S}
            pont = x->dir;
            pop(S); {retira um elemento da pilha}
        } else acabou = 1;
    }
}
```