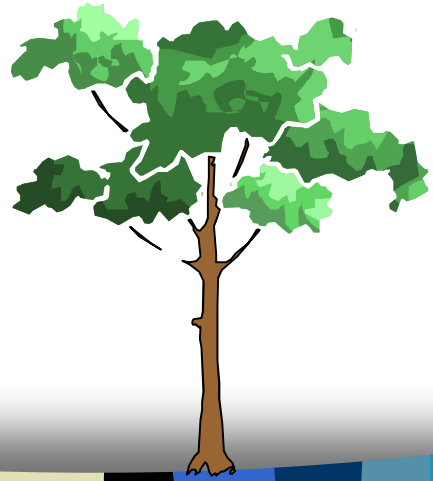
SCC122 – Estruturas de Dados



Árvores &
Árvores Binárias

Prof. Roseli A. F. Romero

Material Original:

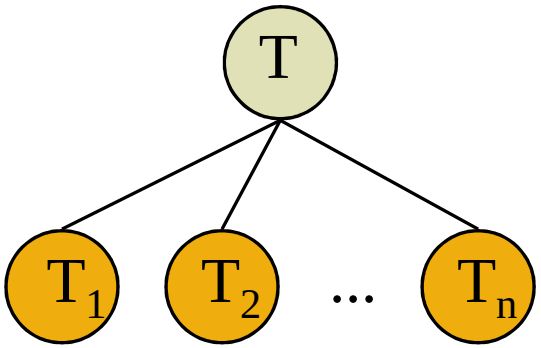
Walter Aoiama Nagai;

Maria das Graças Volpe Nunes;

Definições

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

- T = ∅, 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 >= 1 conjuntos **disjuntos** não vazios (T_1,T_2,…,T_n), cada qual, por sua vez é uma árvore;
- T_1,T_2,…,T_n são chamadas sub-árvores de T;
- Um nó sem subárvores é denominado **nófolha** ou simplesmente **folha**.

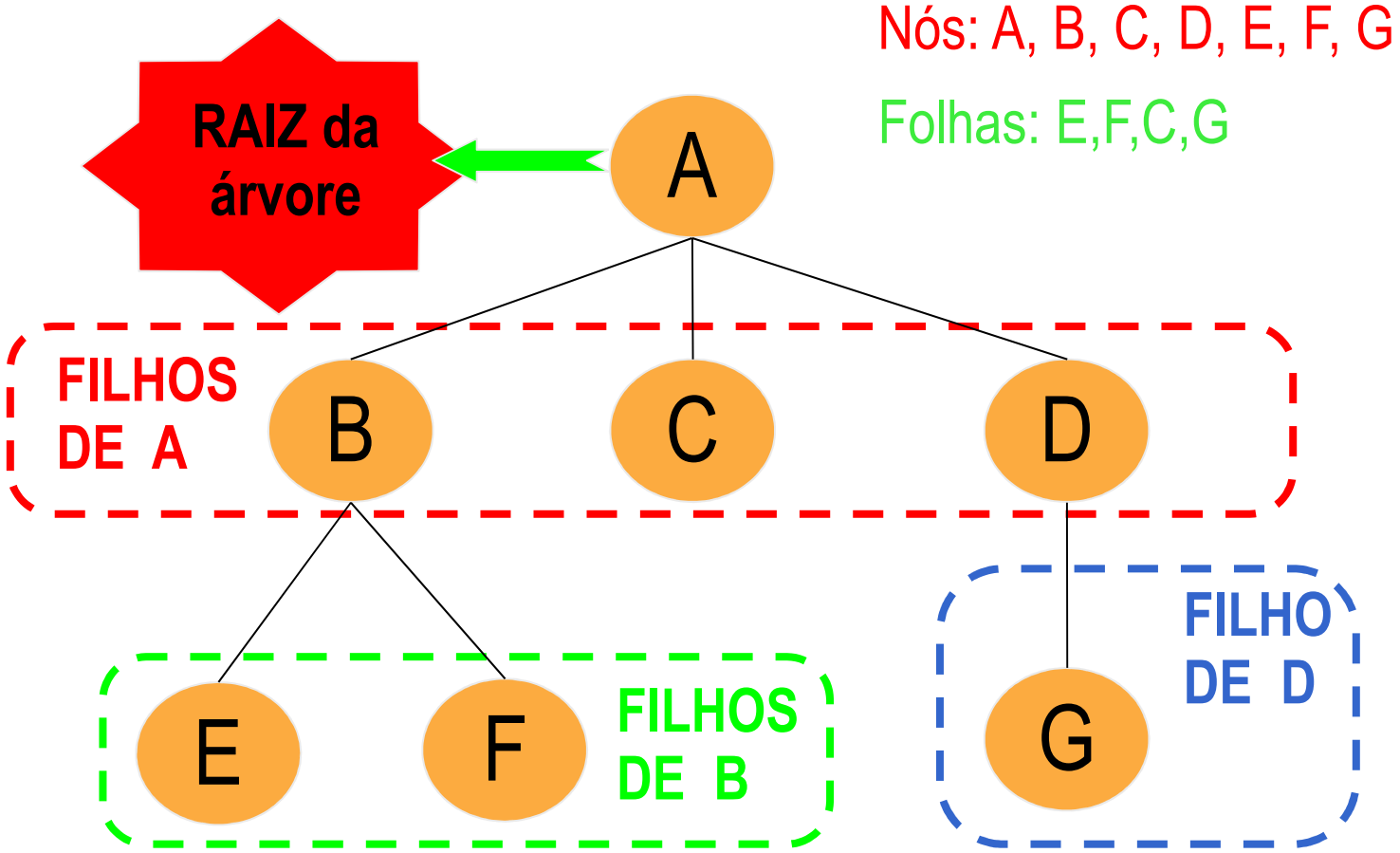


Definições (cont.)

- 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.)



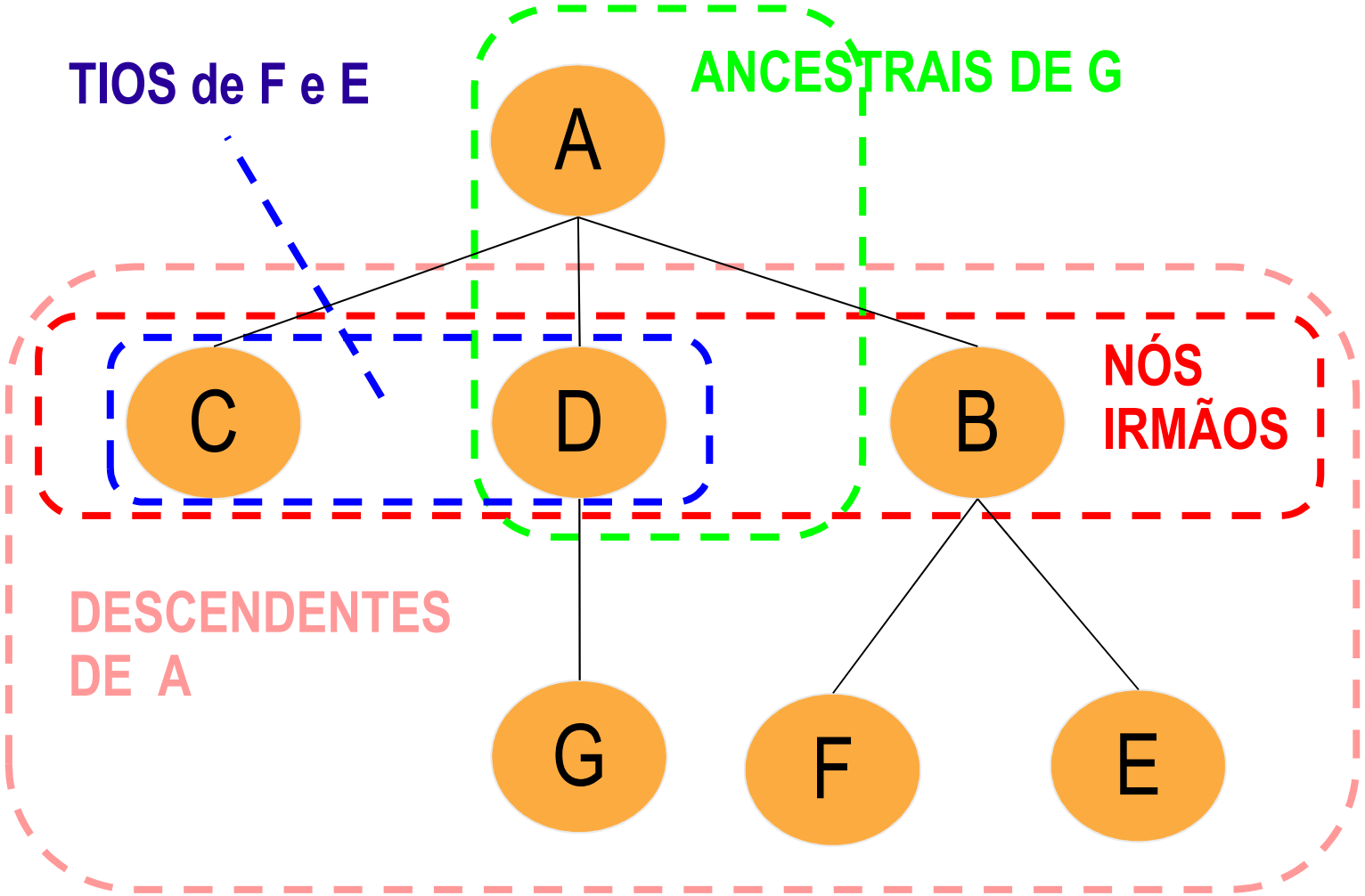
Definições (cont.)

- 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 Y1, Y2, ...Yj são irmãos e se o nó Z é filho de Y1 então Y2,...Yj são TIOs de Z.

Definições (cont.)



Conceitos

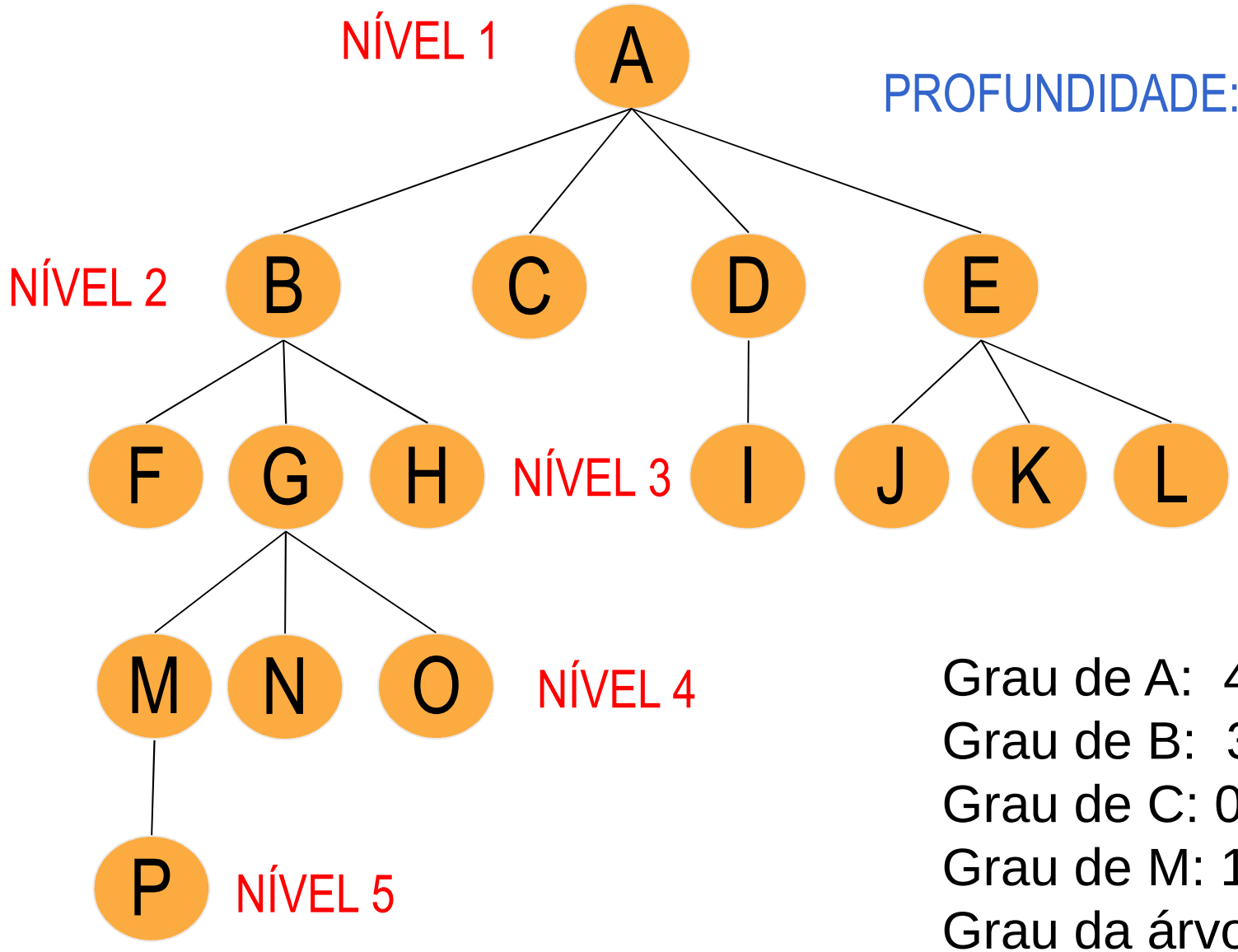
- O NÍVEL de um nó X é definido como:
  - O nó raiz está no nível 1;
  - Os outros nós possuem um nível a mais que o nível de seu nó PAI;

- 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.)

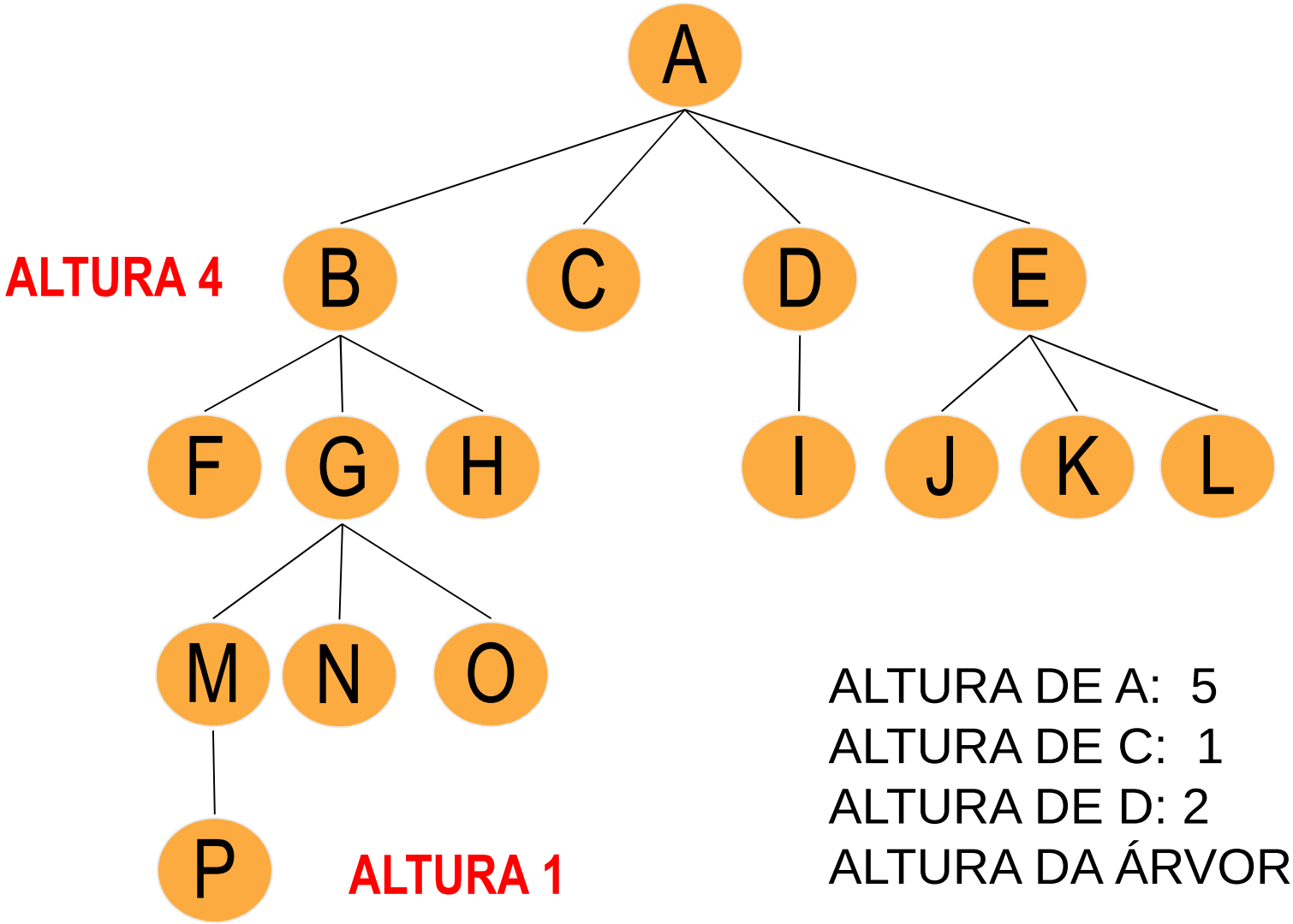


Grau de A: 4
Grau de B: 3
Grau de C: 0
Grau de M: 1
Grau da árvore:4

Conceitos (cont.)

- A ALTURA de um nó é o número de nós no maior caminho do nó 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).

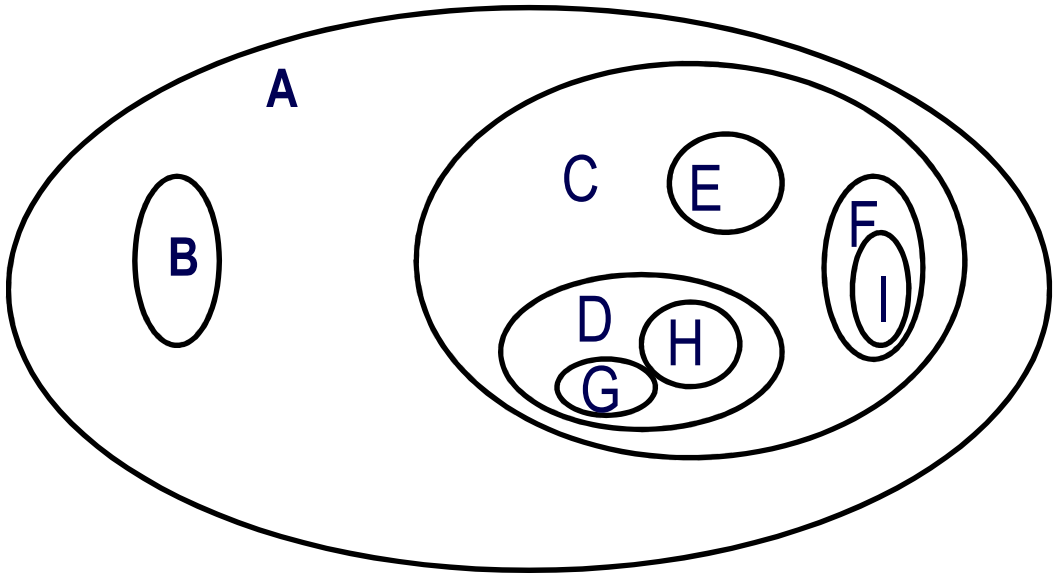
Conceitos (cont.)



ALTURA DE A: 5
ALTURA DE C: 1
ALTURA DE D: 2
ALTURA DA ÁRVORE: 5

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:

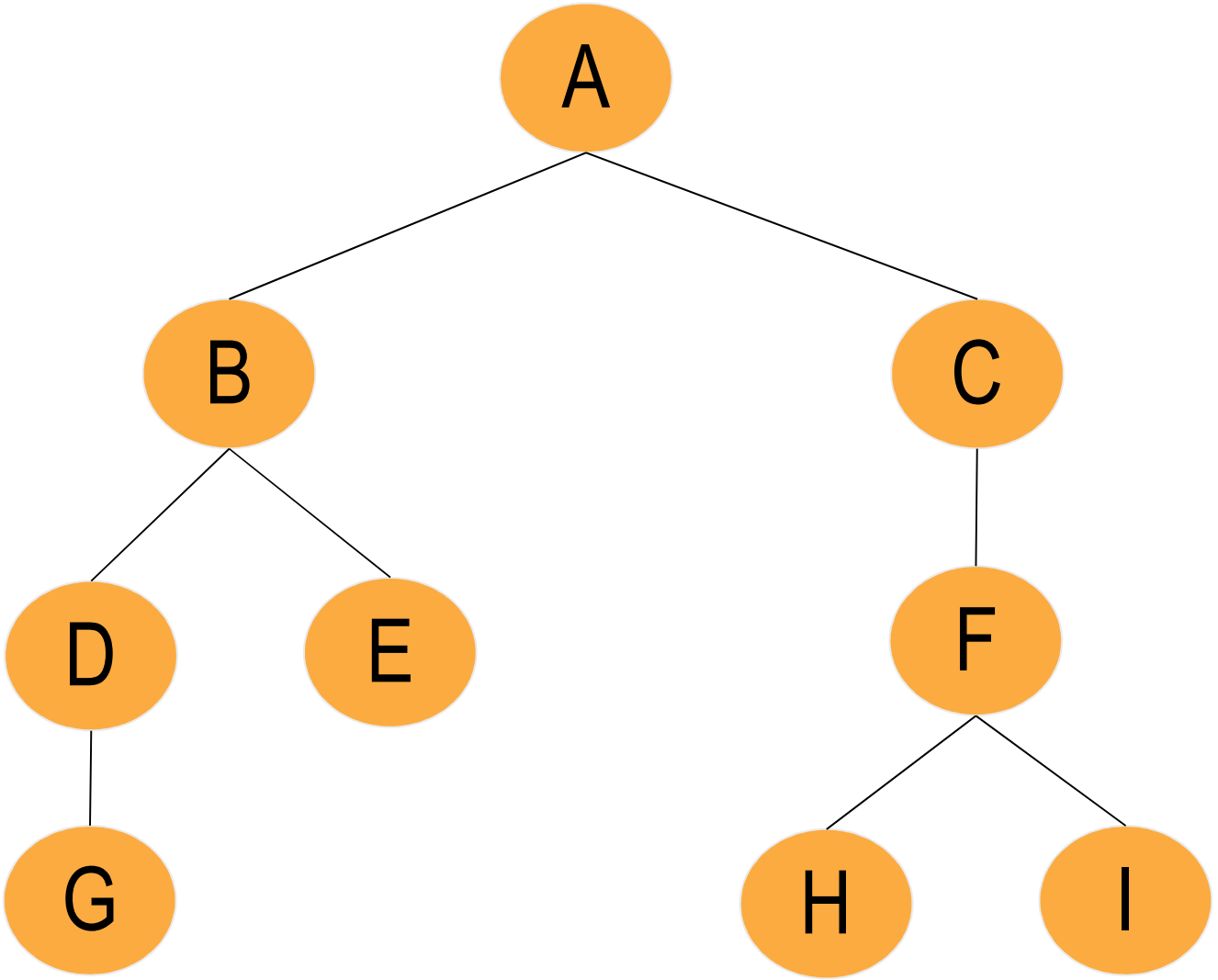
- T = ∅, a árvore é dita vazia, ou
- Existe um nó especial (inicial) chamado raiz de T, e os nós restantes em dois subconjuntos distintos TE e TD, a subárvore esquerda e a subárvore direita de T, respectivamente, as quais também são árvores binárias.

Árvore Binárias (AB) (cont.)

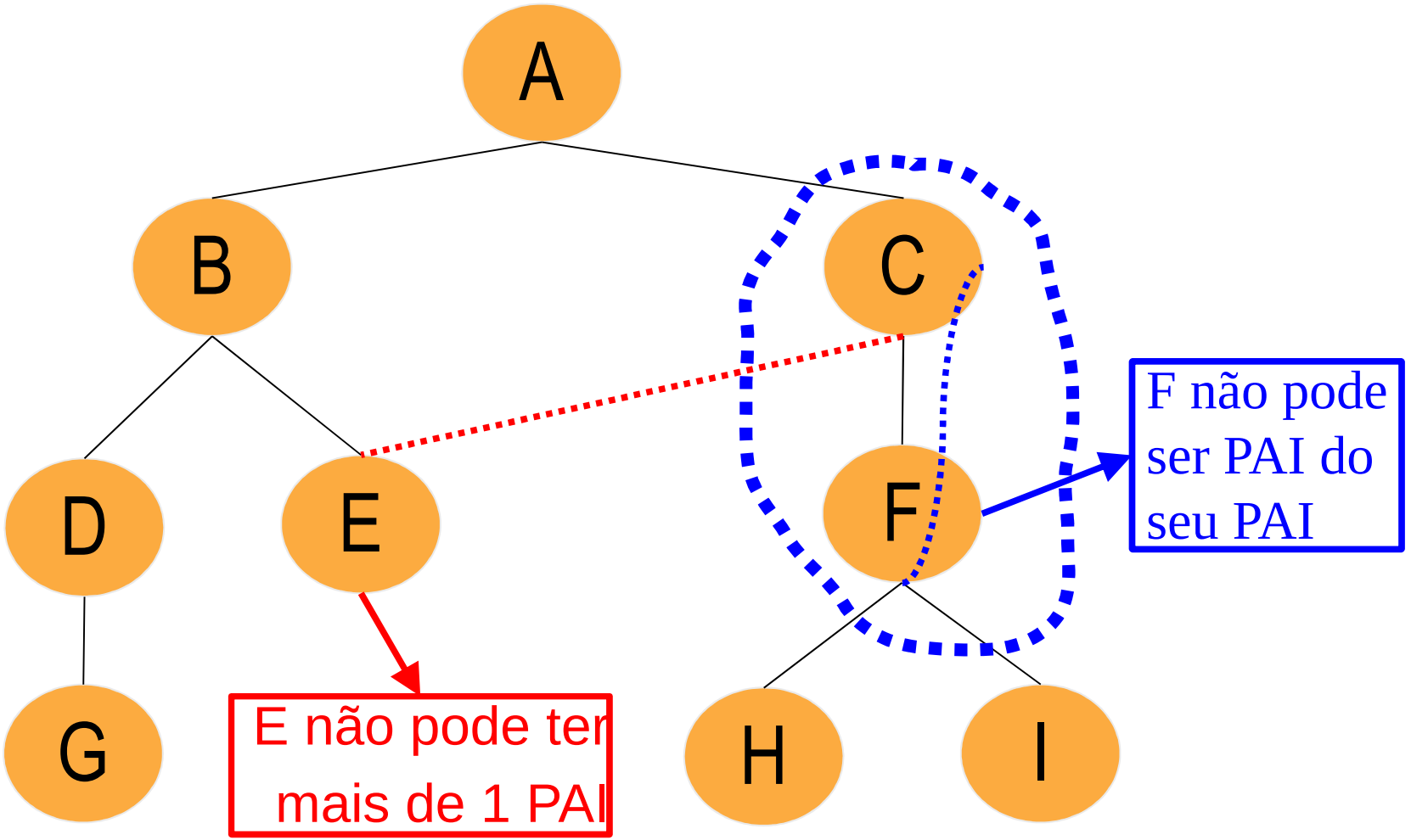
- A **raiz** da **sub-árvore esquerda** (direita) de um nó v, se existir, é denominada **filho esquerdo** (direito) de v. Pela natureza da árvore binária, o filho **esquerdo** pode existir sem o direito e vice-versa.

- Se r é a raiz de T, diz-se que T^E_r e T^D_r são as subárvores esquerda e direita de T, respectivamente.

Árvore Binárias (AB) (exemplo)

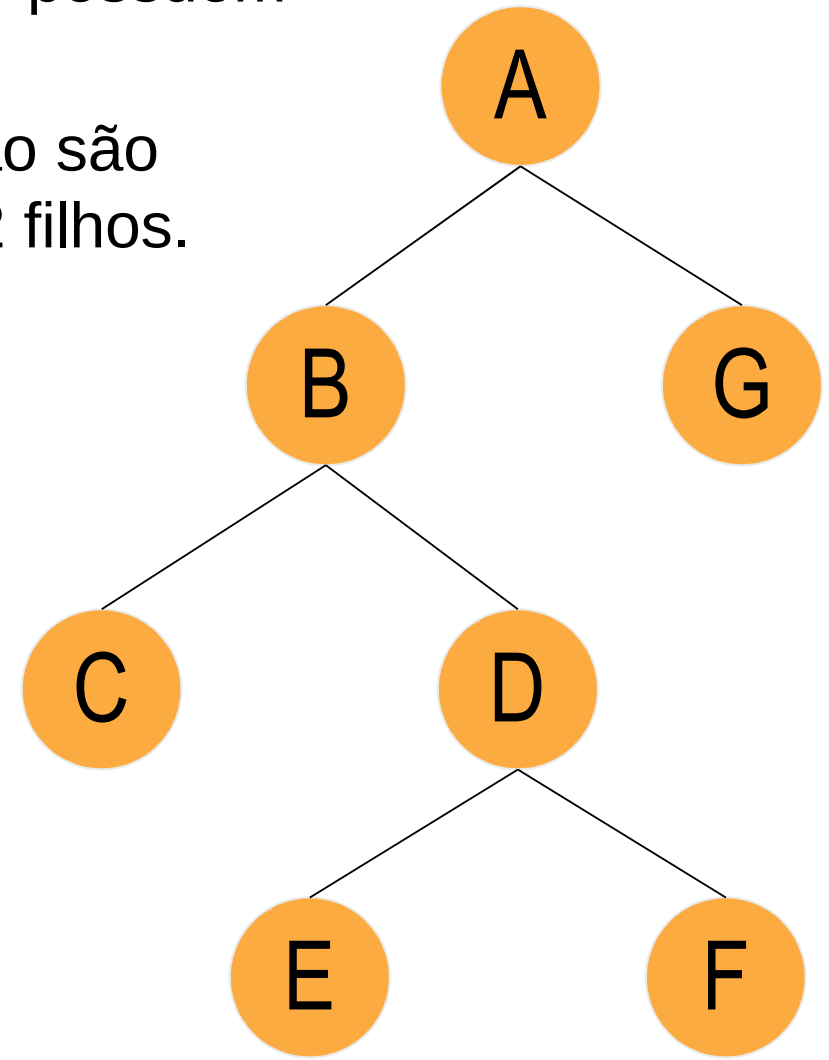


Árvore Binárias (AB) (cont.)



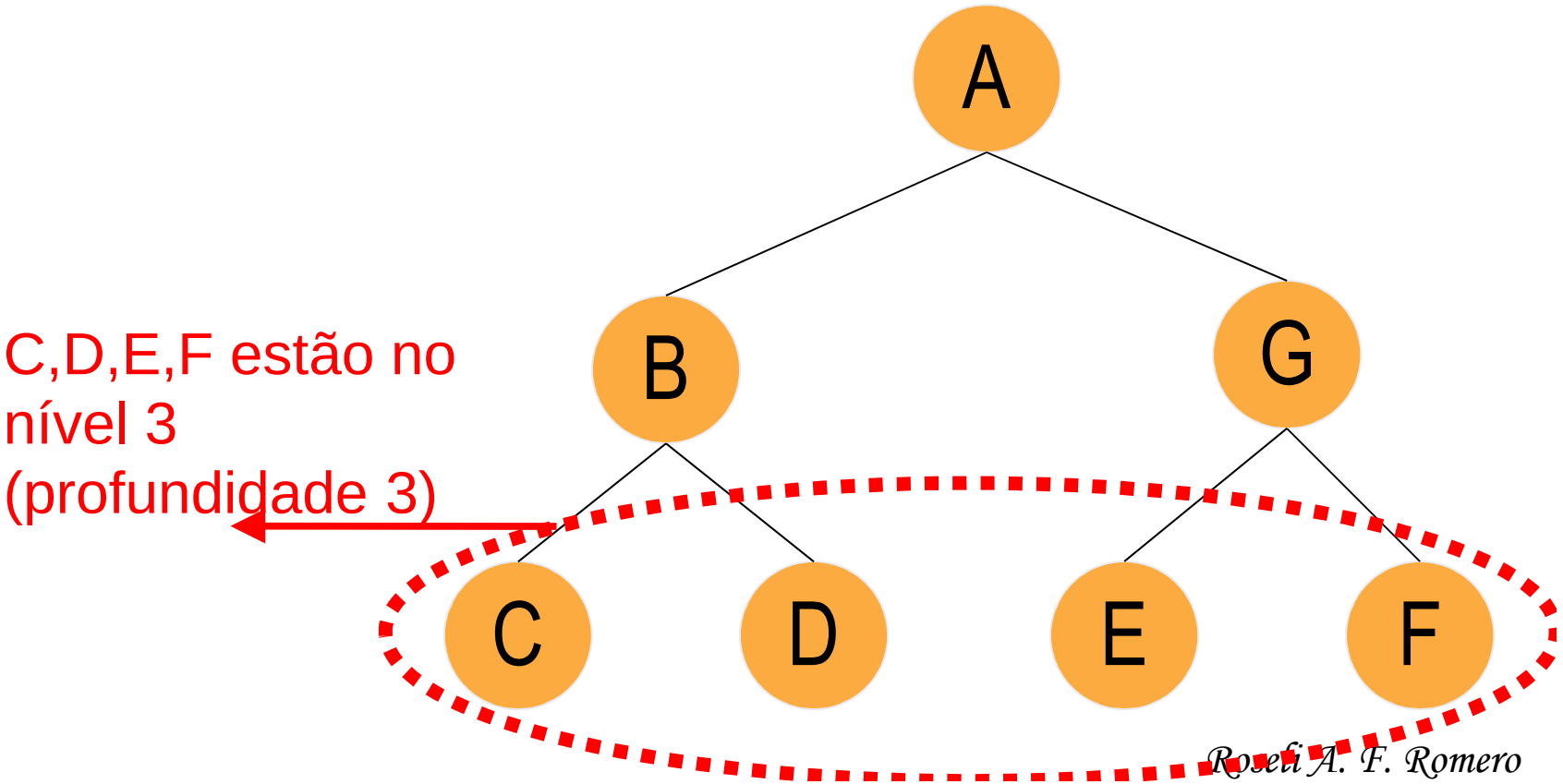
Árvore Estritamente Binária

- 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.)

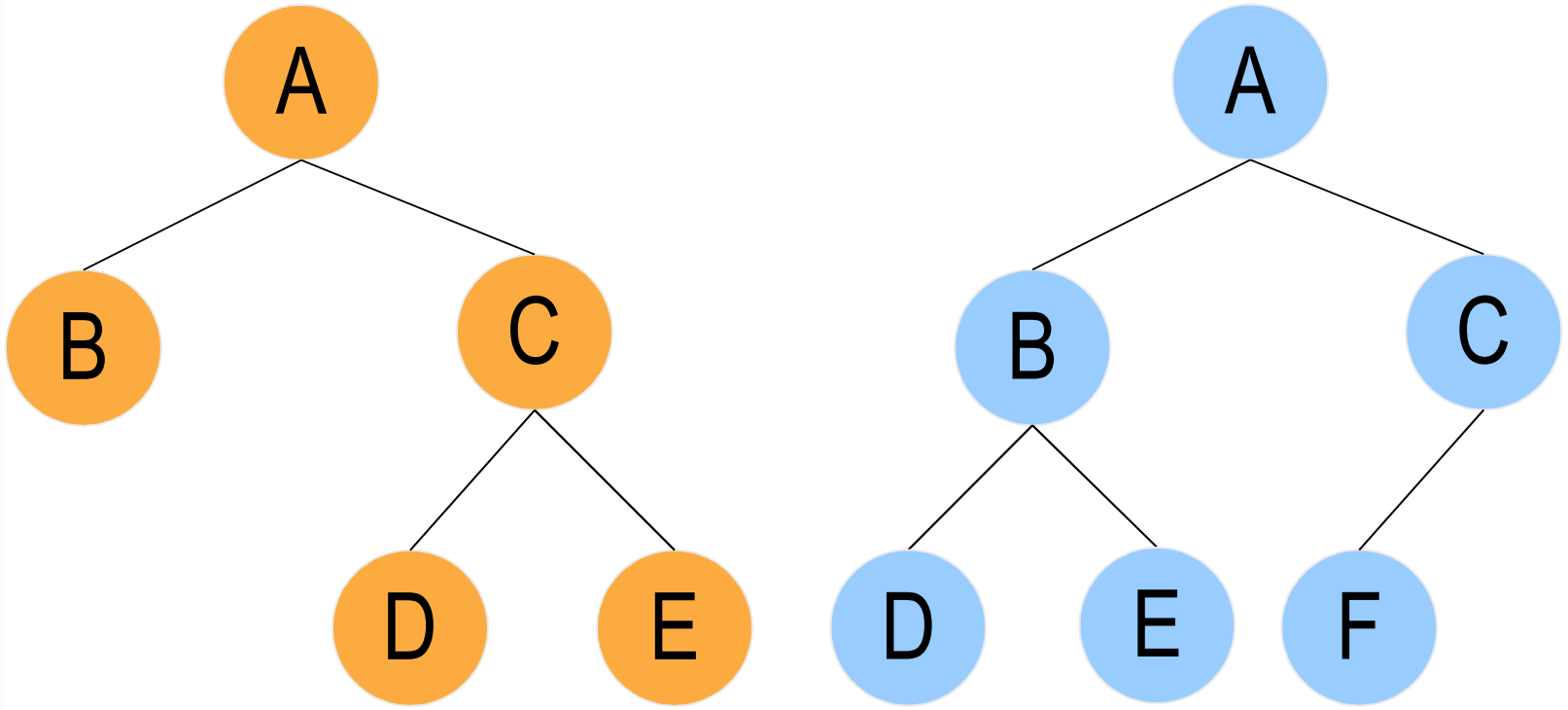
- Pode-se calcular o número total de nós (NT) de uma árvore binária completa, somando-se o número de nós em cada nível da árvore.
- Por exemplo, uma AB completa com profundidade 3 possui 7 nós, considerando a variação de níveis entre 0-2:
  - Nível 1: => 1
  - Nível 2: => 2
  - Nível 3: => 4
  - NT =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

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.



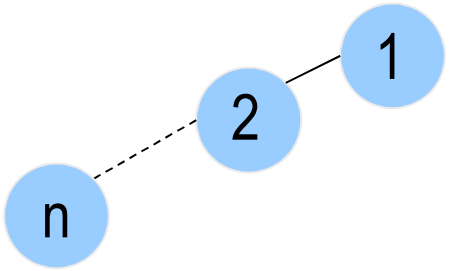
Árvore Binária Perfeitamente Balanceada

- 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.

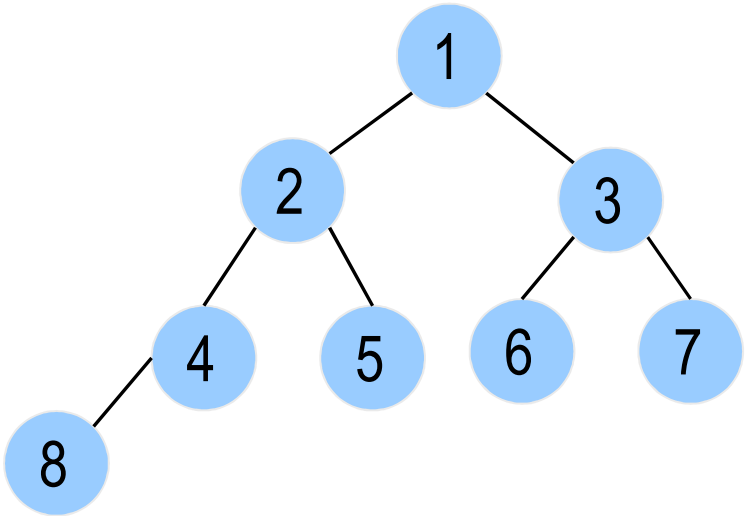
- Toda AB Perfeitamente Balanceada é Balanceada, mas não vale o inverso.

Questões

- Qual é altura máxima de uma AB com **n** nós?
- Resposta: n
  Árvore degenerada ≡ 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; h=1

n=2,3; h=2

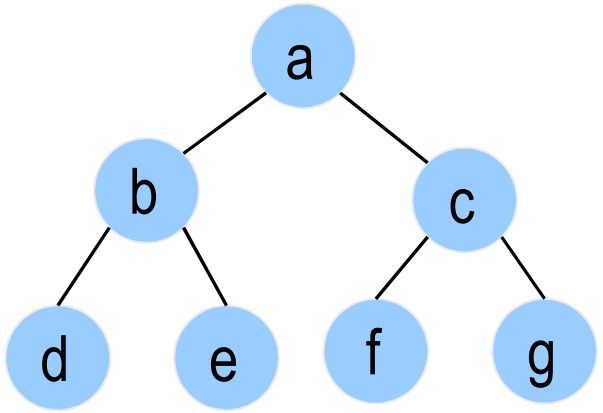
n=4..7; h=3

n=8..15; h=4

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

Implementação de AB (seqüencial)

- Armazenar os nós, por nível, num array:

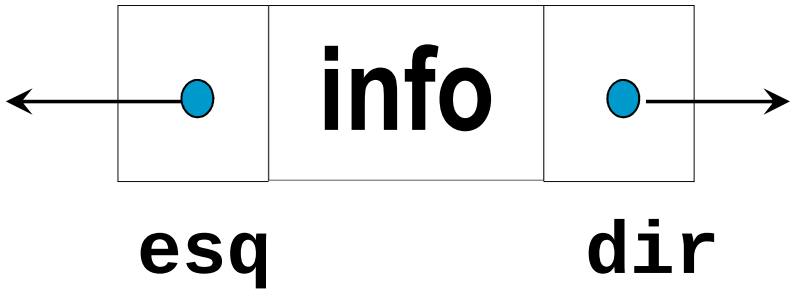


1	2	3	4	5	6	7		...
a	**b**	**c**	**d**	**e**	**f**	**g**		**...**

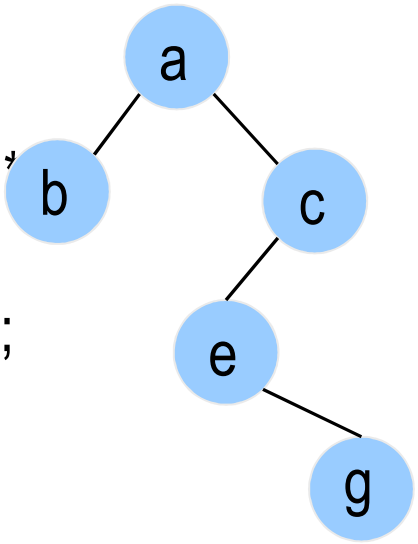
- Assim: se um nó ocupa a posição i, então seus filhos diretos estão nas posições 2i e 2i+1.
- Vantagem: espaço só para armazenar conteúdo; ligações implícitas.
- Desvantagens: espaços vazios se a árvore não for completa por níveis, ou se sofrer eliminação.

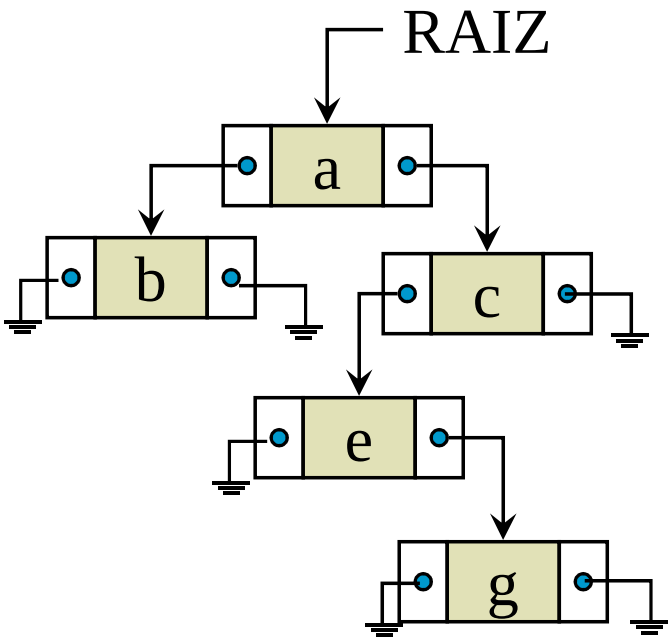
Implementação de AB (dinâmica)

Para qualquer árvore, cada nó é do tipo:




```
typedef struct NO{
        tipo_elem info;
         struct NO  *esq, *
      }
Typedef struct No *Tree;
VAR raiz: Tree;
```





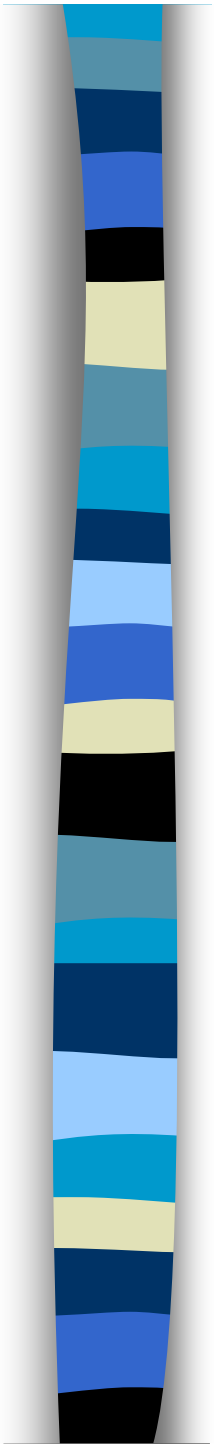
Roseli A. F. Romero

Operações do TAD AB

```
procedure define(var t: tree);
 begin t:= nil end; {Cria AB vazia}

procedure cria_raiz(var t: tree; item: tipo_elem);
 var no: tree;
 begin
  new(no);
  no^.esq := nil;
  no^.dir := nil;
  no^.info := item;
  t := no;
 end;

function vazia(t: tree): boolean;
 begin vazia := (t=nil) end;
```



```
{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".}
procedure adiciona_dir(VAR t:tree;item_pai,item:
     tipo_elem);
   var pai, no: tree;
   begin
     pai := localiza(t,item_pai);
     if pai <> nil then
      if (pai^.dir <> nil) then
        erro("já tem filho à direita");
      else
        begin
          new(no);
          no^.esq := nil;
          no^.dir := nil;
          no^.info := item;
          pai^.dir := no;
        end;
   end;
```

AB - Percursos

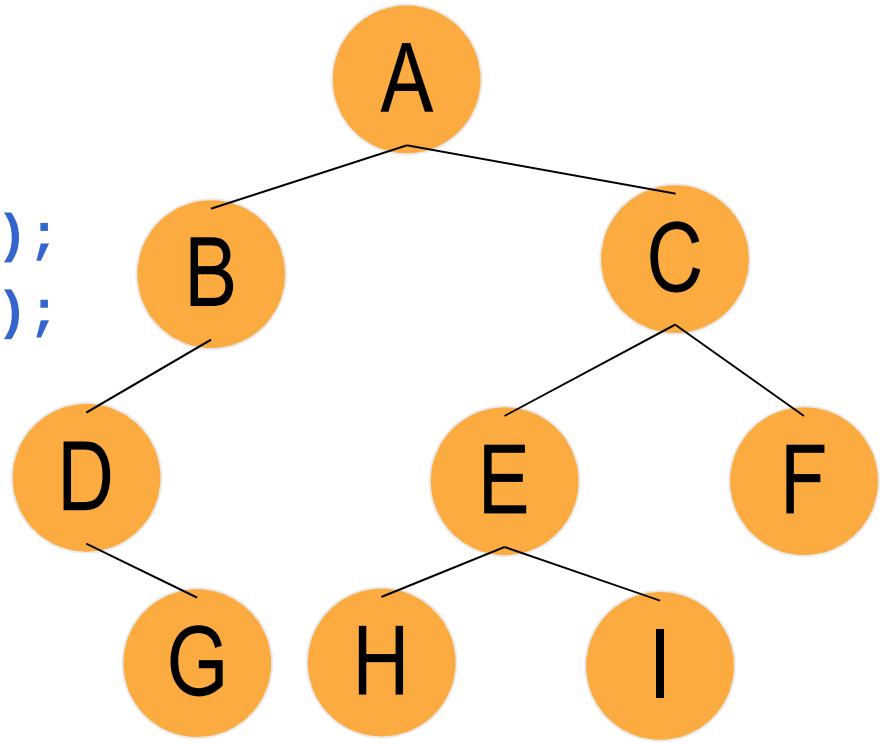
- Objetivo: Percorrer uma AB visitando cada nó uma única vez. Um percurso gera uma sequência linear de nós e podemos então falar de nó predecessor ou sucessor de um nó, segundo um dado percurso.
- Não existe um percurso único para árvores (binárias ou não); Diferentes percursos podem ser realizados, 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

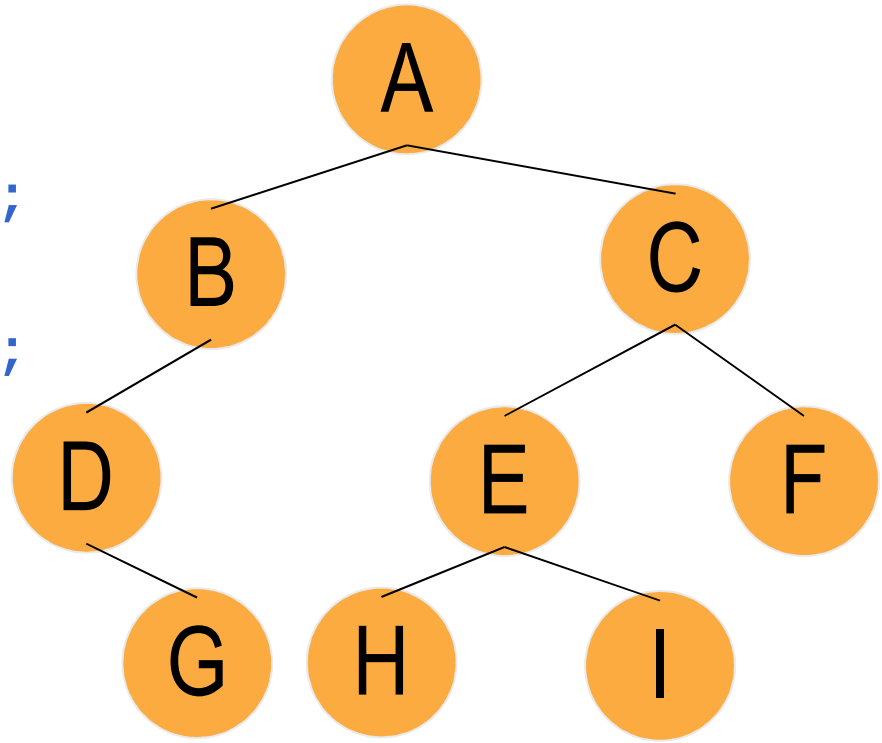
```
procedure pre_ordem(raiz: tree)
Begin
   if raiz <> nil then
   begin
      visita(raiz);
      pre_ordem(raiz^.esq);
      pre_ordem(raiz^.dir);
   end;
End;
```



- Resultado: ABDGCEHIF

AB - Percurso Em-Ordem

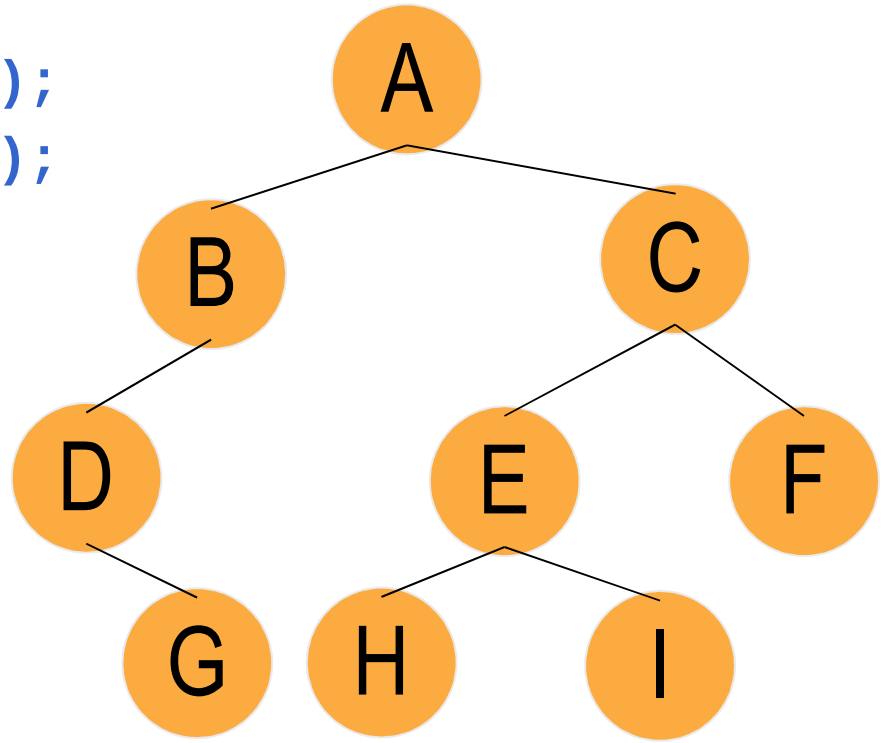
```
procedure em_ordem(raiz: tree)
Begin
   if raiz <> nil then
    begin
       em_ordem(raiz^.esq);
       visita(raiz);
       em_ordem(raiz^.dir);
    end;
End;
```



- Resultado: DGBAHEICF

AB - Percurso Pós-Ordem

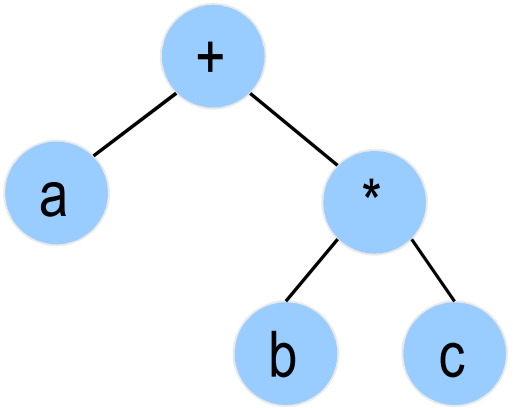
```
procedure pos_ordem(raiz: tree)
Begin
   if raiz <> nil then
    begin
       pos_ordem(raiz^.esq);
       pos_ordem(raiz^.dir);
       visita(raiz);
    end;
End;
```



- Resultado: GDBHIEFCA

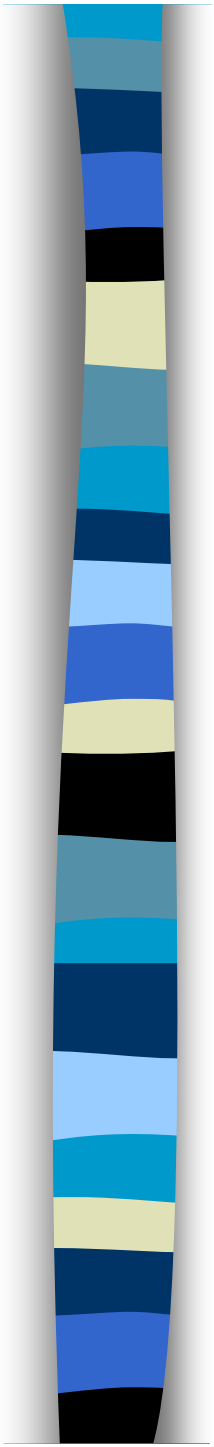
AB – Percursos

- 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 nil caso contrário. OBS.:
Nivel da raiz=1}
function nivel(t:tree;item: tipo_elem): integer;
  var n: integer;
      achou: boolean;
  begin
    achou := false;
    n := 0;
    travessia(t,n,item,achou);
    nivel := n;
  end;
```

```
{percorre a árvore ptr em Pre-ordem procurando por item e
calculando e retornando seu nível}
procedure travessia(ptr:tree; var niv: integer; item:
tipo_elem; var achou: boolean);
 begin
   if ptr <> nil then
   begin
     niv := niv+1;
     if ptr^.info = item then
        achou := true;
     else begin
        travessia(ptr^.esq,niv,item,achou);
        if not achou then
           travessia(ptr^.dir,niv,item,achou);
        if not achou then
           niv := niv - 1;
     end;
   end;
 end;
```

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 formula 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

```
procedure enigma(raiz: tree);
Var S: pilha;
    x,pont: tree;
Begin
   criar(S); {S é uma pilha}
   pont := raiz; acabou := (raiz=nil);
   while not acabou do
   begin
      while pont <> nil do
      begin
         visita(pont);
         push(pont,S); {insere pont na pilha S}
         pont := pont^.esq;
      end;
      if not vazia(S) then
      begin
        x := topo(S); {lê o conteúdo do topo da pilha S}
        pont := x^.dir;
        pop(S); {retira um elemento da pilha}
      end
      else acabou := true;
   end;
End;
```