TAD: Árvores e árvore binária

Estrutura de Dados – ED I

Motivação: estrutura (morfologia) "natural" de uma árvore

Árvore

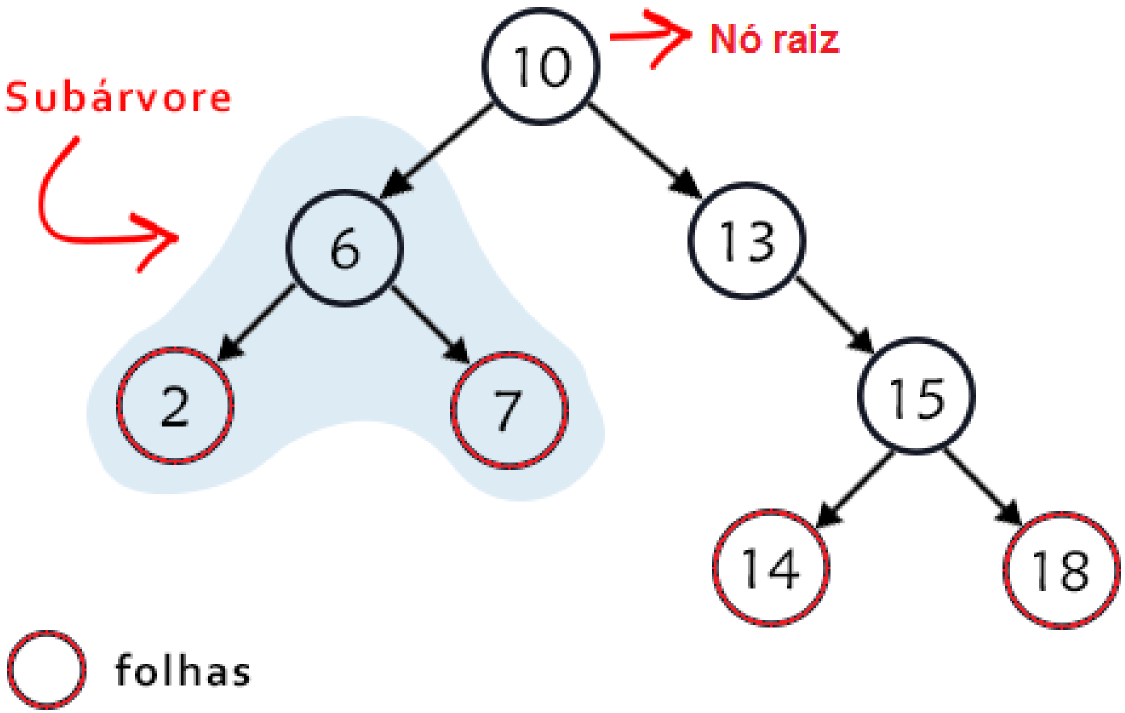
- **Definição:** Uma árvore T é um conjunto finito de elementos, denominados nós ou vértices, que satisfazem as seguintes condições:

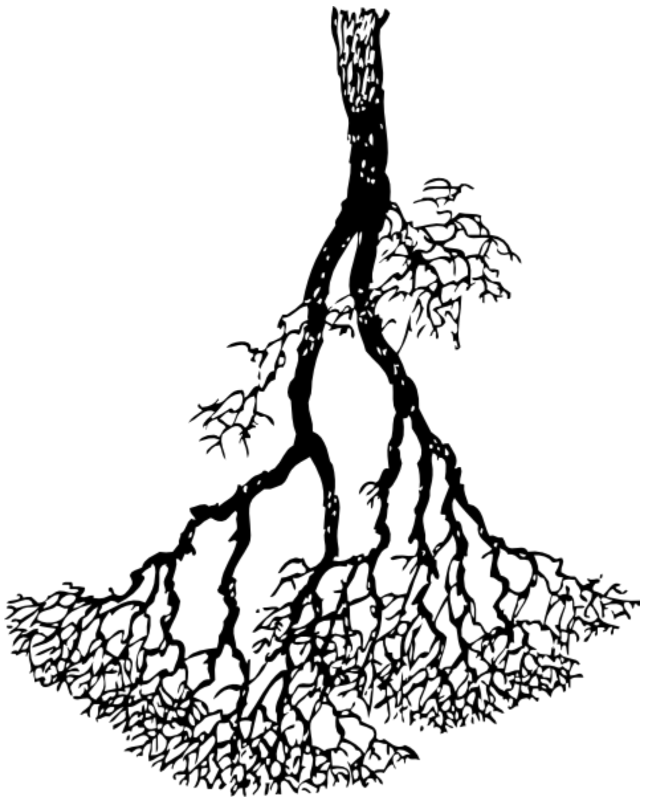
  1. T contém um nó especial chamado **nó raiz (nó inicial).**
  2. Os demais nós, ou **(i)** constituem um conjunto vazio, ou então **(ii)** são divididos em $n \geq 1$ conjuntos **disjuntos não-vazios**, $(T_1, T_2, \dots, T_n)$, cada qual sendo também uma árvore.

- **Nomenclaturas:**
  - $(T_1, T_2, \dots, T_n)$ são chamados de sub-árvores de T.
  - Um nó **sem sub-árvores** é denominado **nó-folha**, ou simplesmente, **folha**.

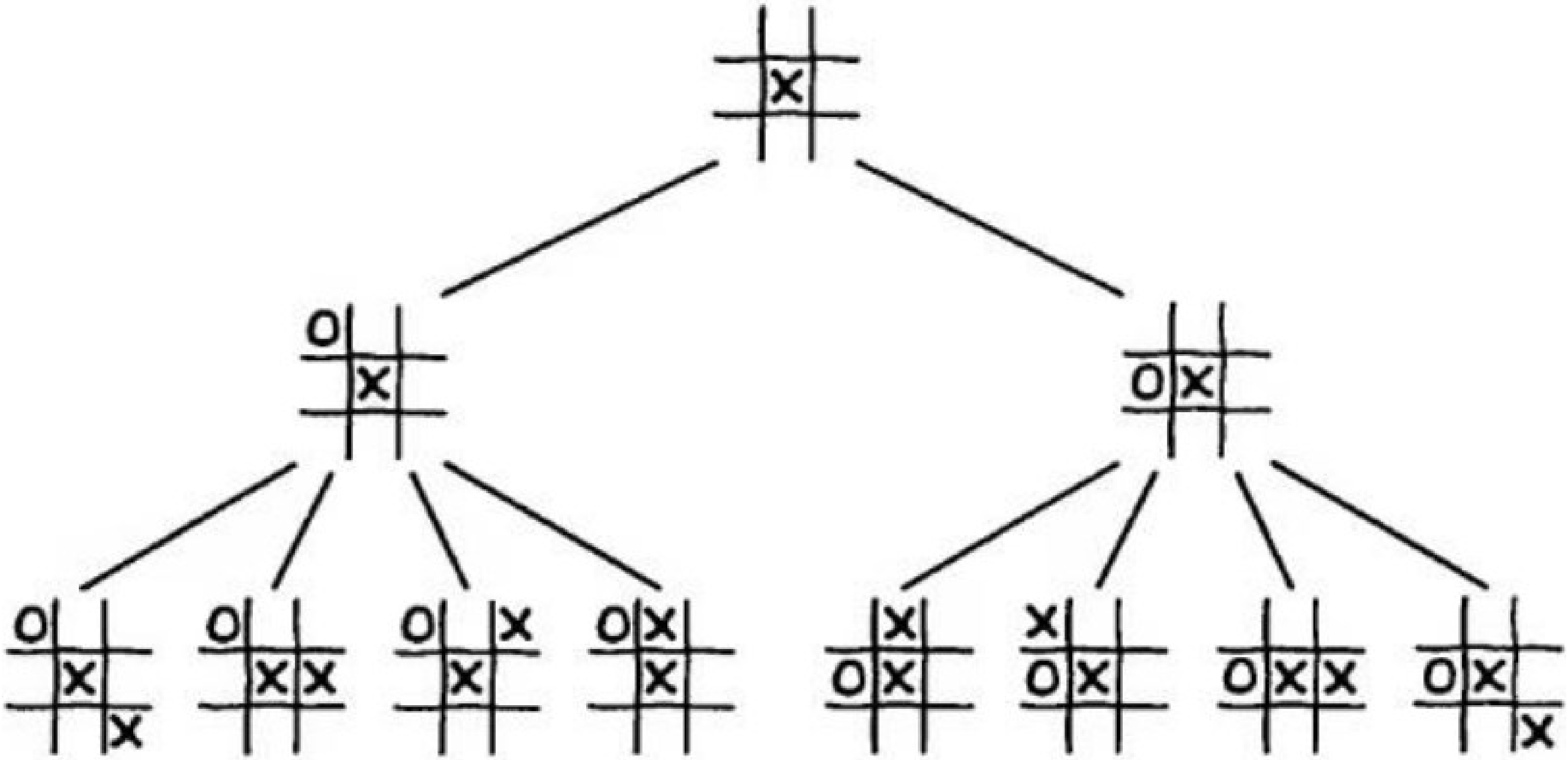
- **Observação:** Se T $= \emptyset$, então T é uma árvore vazia.

Árvore – abstração visual

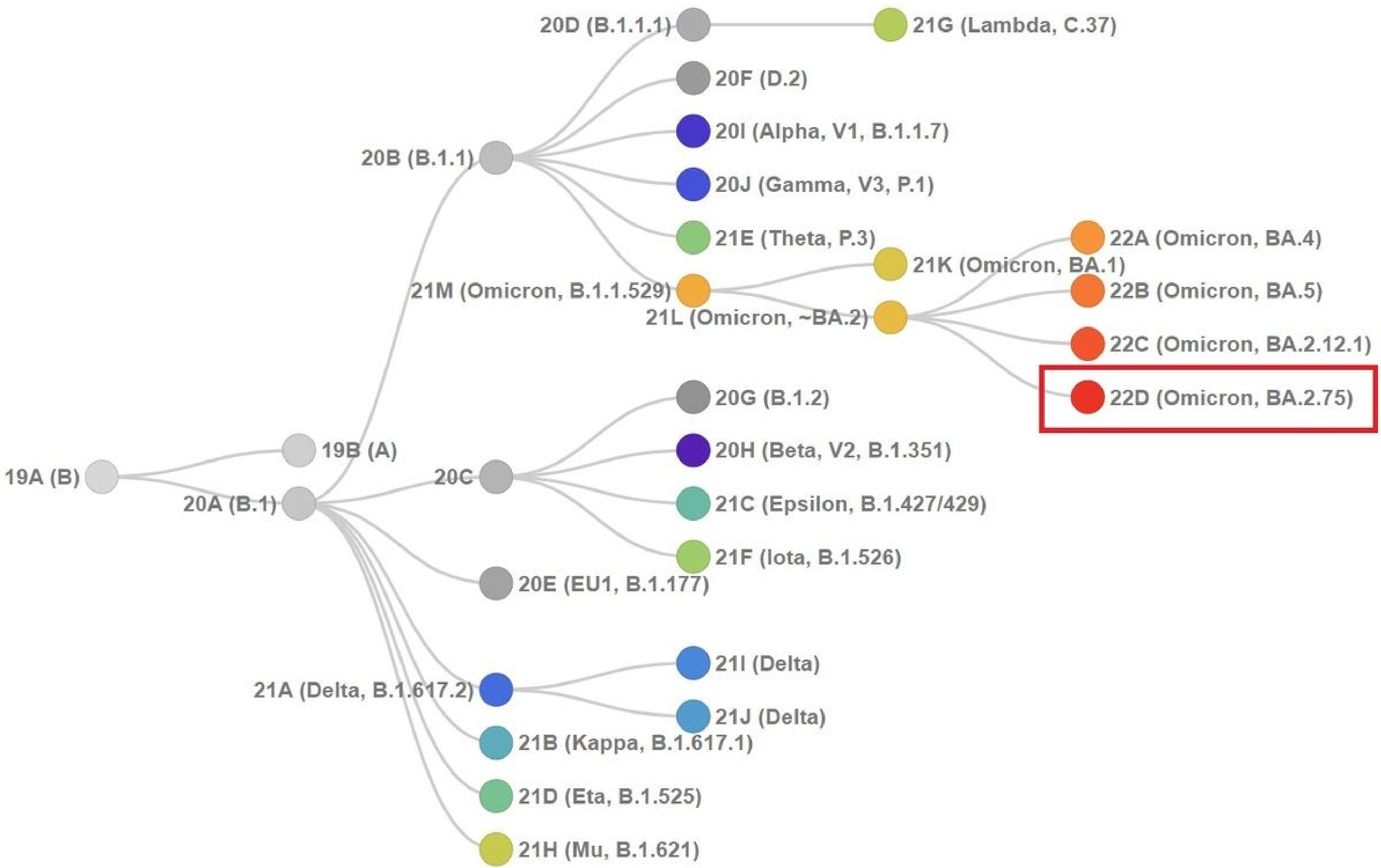




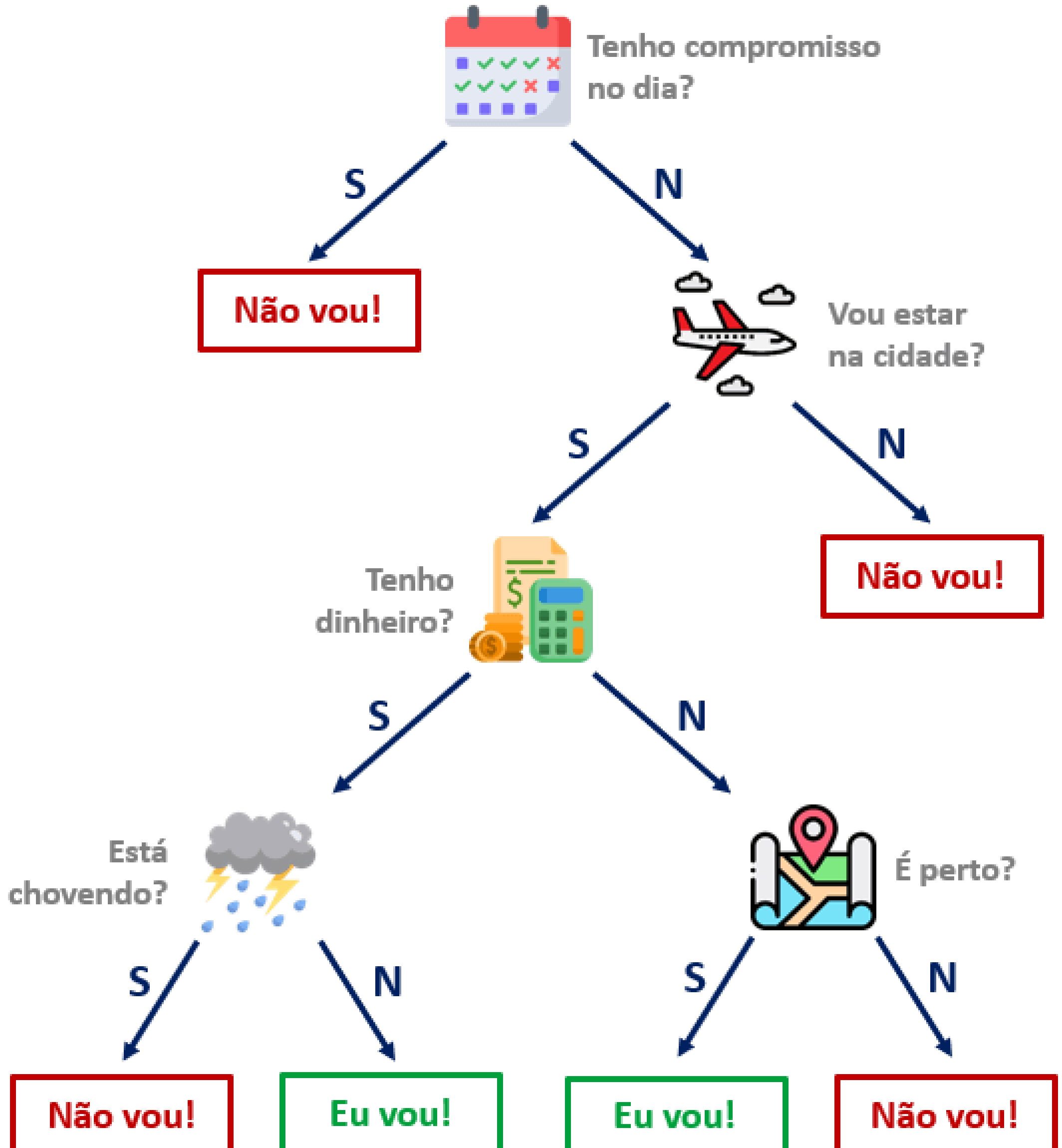
Árvore – motivação



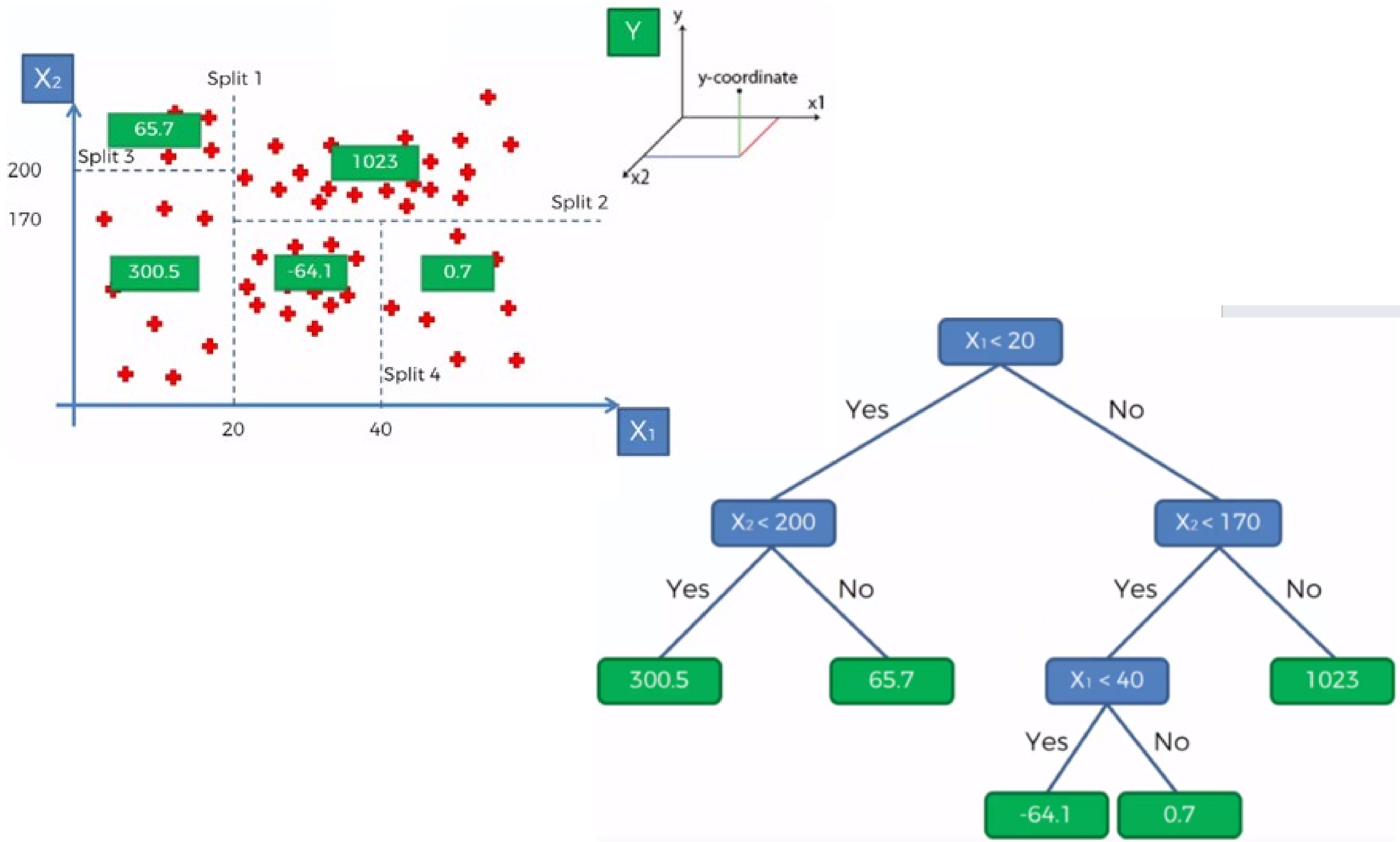
Árvore – motivação



Árvore – motivação

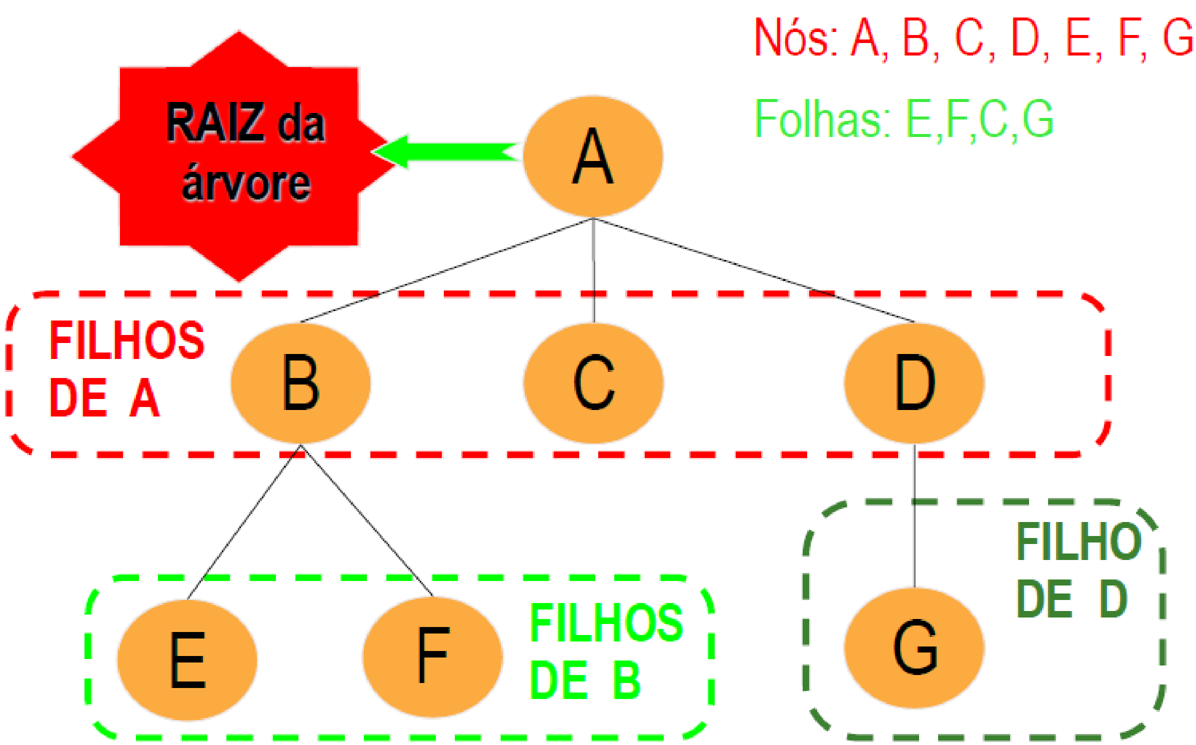


Árvore – motivação



Árvore – Propriedades e características

- As árvores são adequadas para representar **estruturas hierárquicas não-lineares**, como relações de descendência (pai, filho, irmãos, etc).
- Se um nó x é raiz de uma árvore T, e um nó y é raiz de uma sub-árvore de x, então x é **PAI** de y, e y é **FILHO** de x.



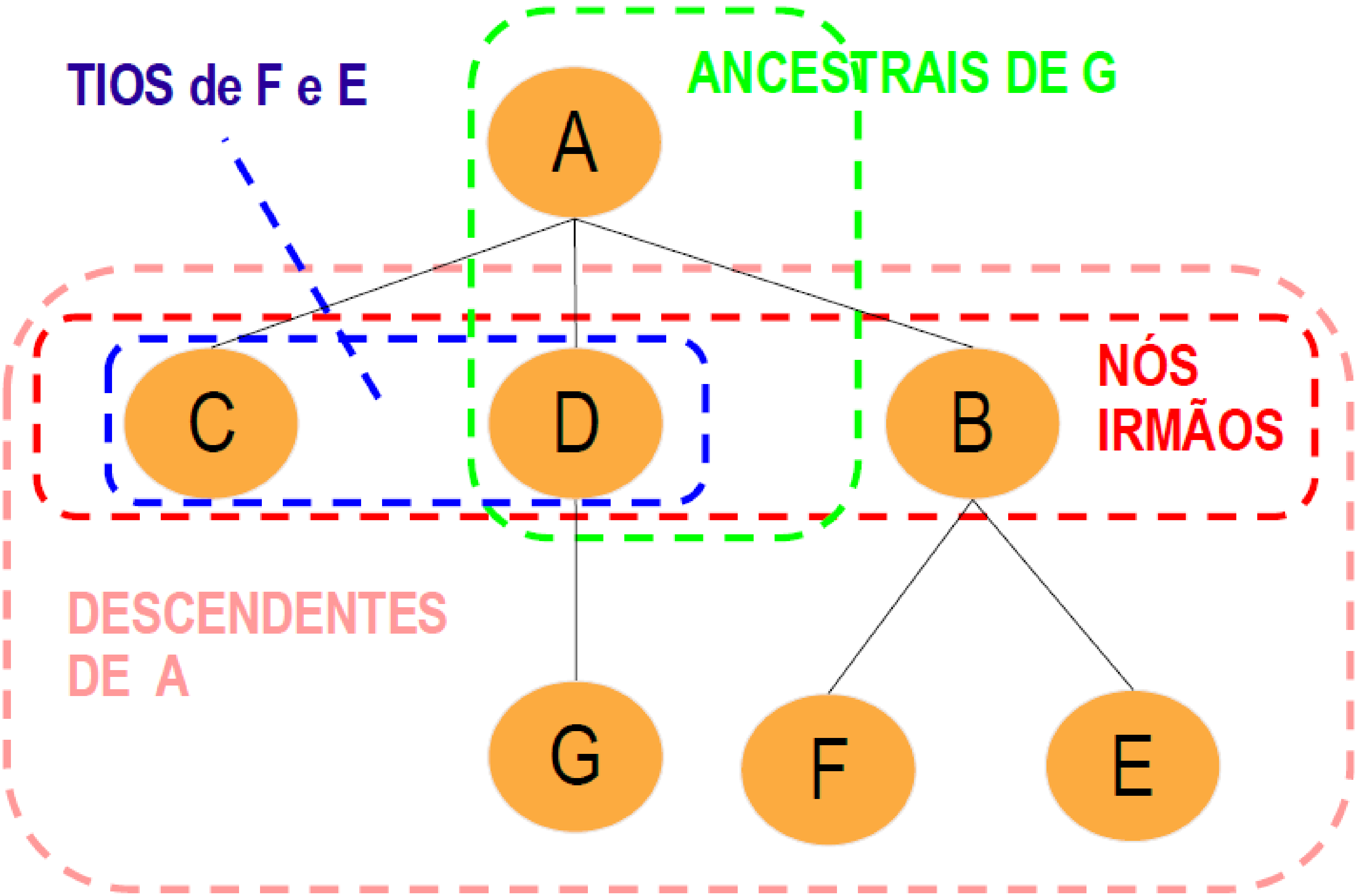
Árvore – Propriedades e características

- O nó x é chamado de **ANCESTRAL** do nó y (sendo y **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 $y_1, y_2, \ldots, y_j$ são IRMÃOS, e o nó z é FILHO de y_1, então $y_2, \ldots, y_j$ são **TIOS** de z.

Árvore – Propriedades e características



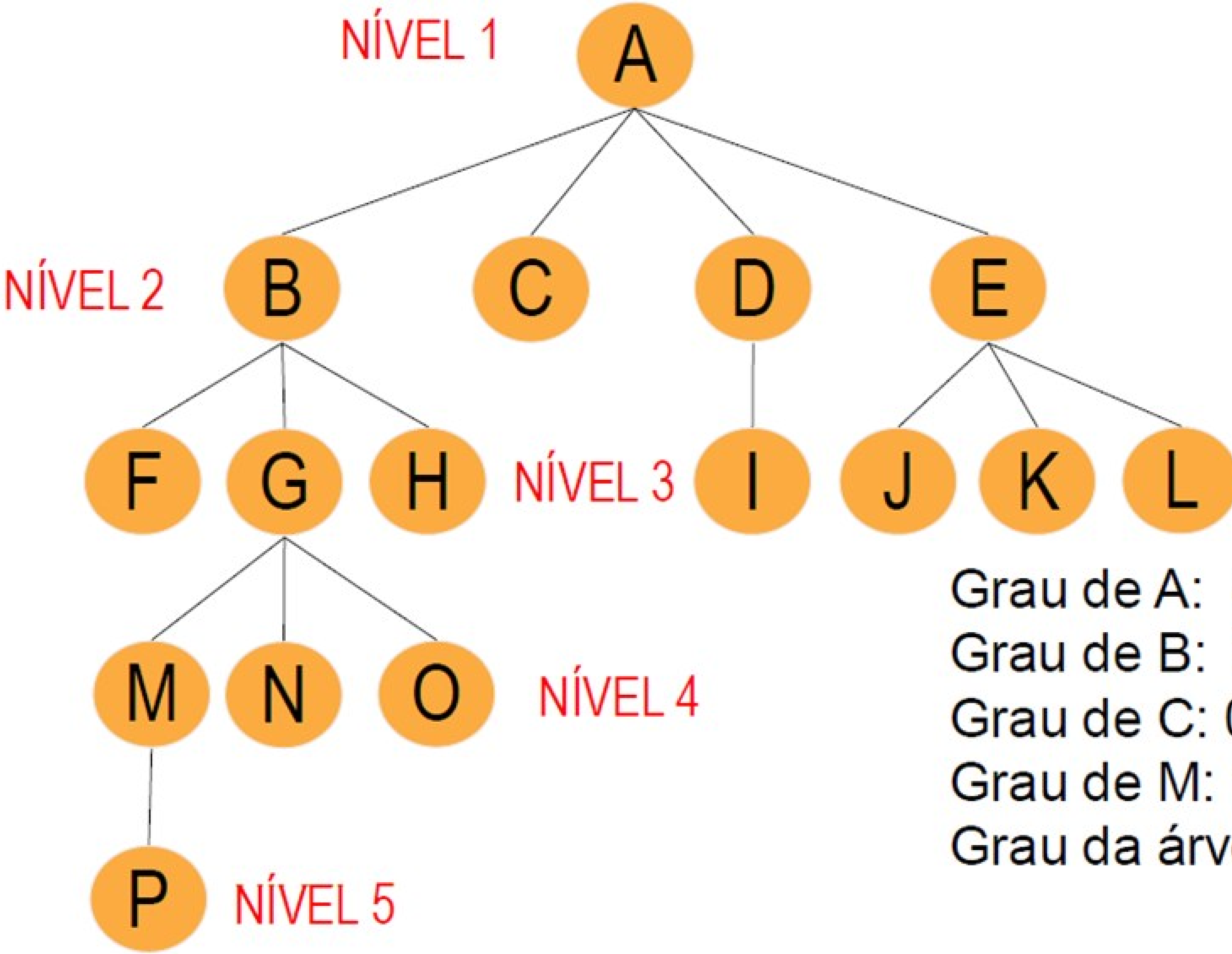
Árvore – Propriedades e características

- **Definição:** O **NÍVEL** de um nó x é definido como:
  - Se x é o nó raiz, então seu nível é 1.
  - O nível de um nó **não-raiz** é dado por (nível de seu PAI + 1).
- **Observação:** Os nós de maior nível serão, desta forma, os nós-folhas.

- **Definição:** O **GRAU de um nó** x é igual ao seu número de FILHOS.

- **Definição:** O **GRAU de uma árvore** T é o maior entre os graus de todos os seus nós.

Árvore – Propriedades e características

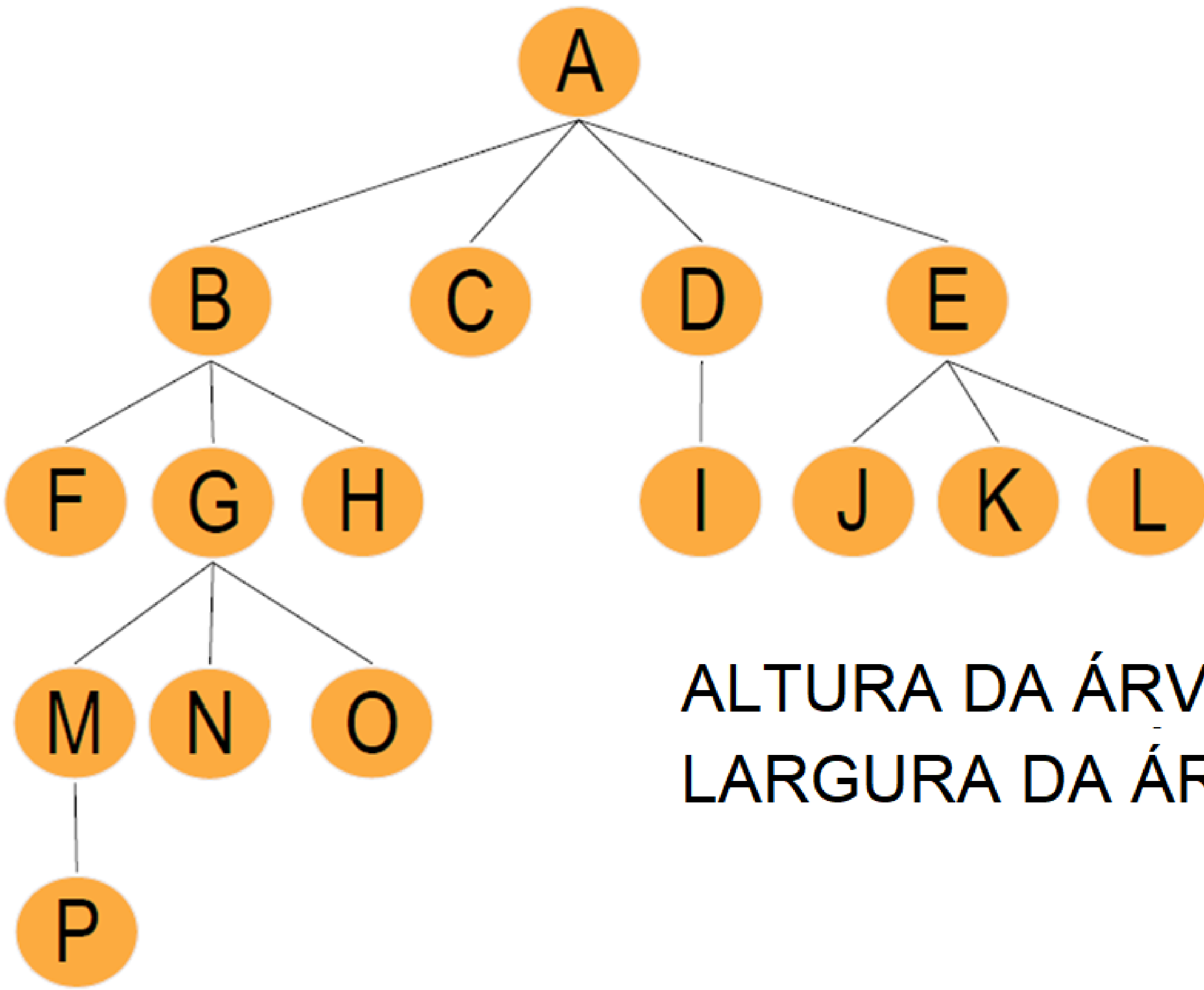


Grau de A: 4
Grau de B: 3
Grau de C: 0
Grau de M: 1
Grau da árvore:4

Árvore – Propriedades e características

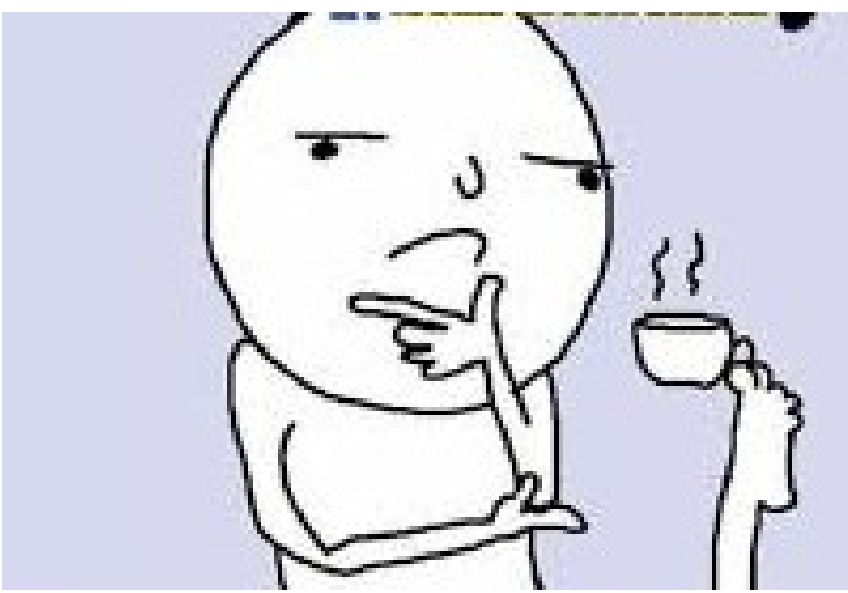
- **Definição:** Uma sequência de nós distintos $v_1, v_2, \dots, v_k$ tal que cada nó v_{i+1} é FILHO de v_i é dito **CAMINHO** na árvore T (diz-se que v_i alcança v_k).
  - O número de arestas (ligações entre nós) de um caminho define o **COMPRIMENTO** do CAMINHO.
- **Definição:** A **ALTURA** (ou **PROFUNDIDADE**) de uma árvore T é dada pelo **maior NÍVEL** de seus nós. Alternativamente, corresponde ao número de nós do maior caminho entre a raiz e os nós-folhas.
  - **Notação:**
    - Altura de uma árvore com raiz x: $h(x)$.
    - Altura de uma sub-árvore com raiz y: $h(y)$
- **Definição:** A **LARGURA** de uma árvore T é definida pelo maior número de nós dentre todos os níveis da árvore.

Árvore – Propriedades e características



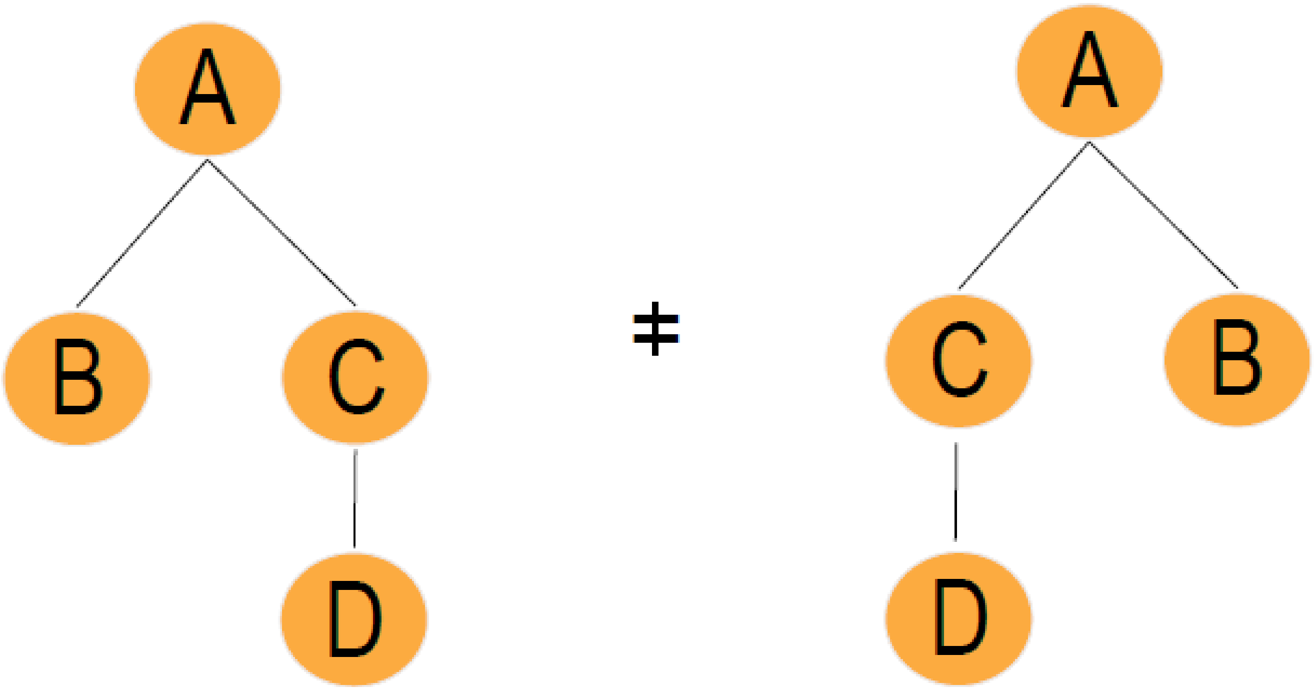
ALTURA DA ÁRVORE: 5
LARGURA DA ÁRVORE: 7

Dúvidas?



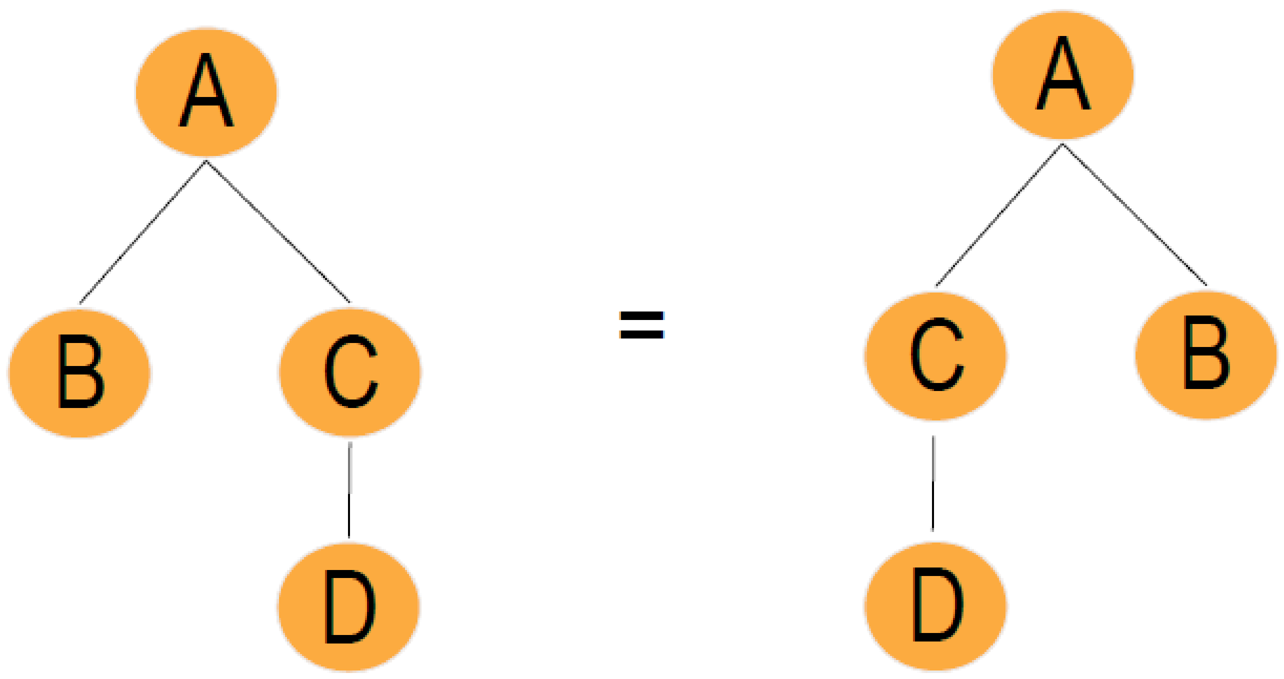
Árvore – Propriedades e características

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



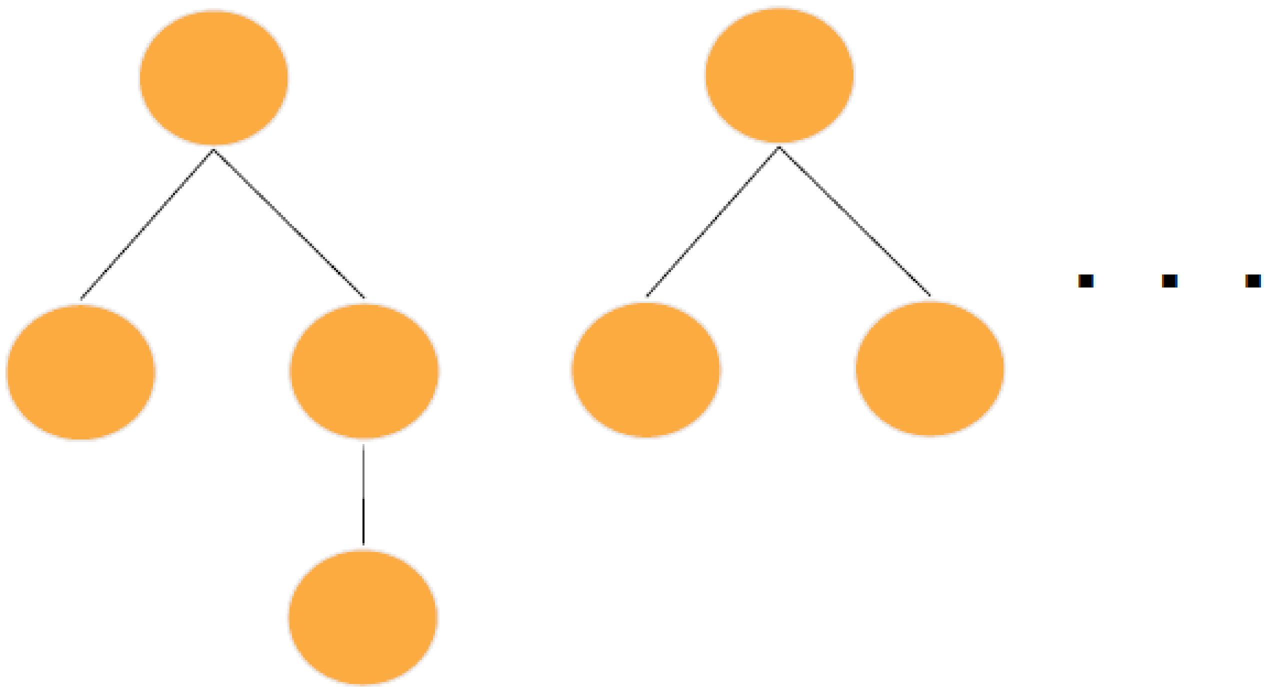
Árvore – Propriedades e características

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



Árvore – Propriedades e características

- **Definição:** Uma **FLORESTA** é um conjunto de 0 (árvore vazia), ou um conjunto $N \geq 1$ finito árvores distintas.




* Me after learning web development
* Job
* Data Structures And Algorithms

Árvore Binária (AB)

- **Definição:** Uma **Árvore Binária** T é um conjunto finito de elementos, denominados nós ou vértices, que satisfazem o seguinte:
  1. T contém um nó especial, denominado **raiz** de T.
  2. Os demais nós podem ter zero ou até dois filhos, isto é, podem ser **subagrupados em até DOIS** sub-conjuntos distintos T_E e T_D, que também são árvores binárias.

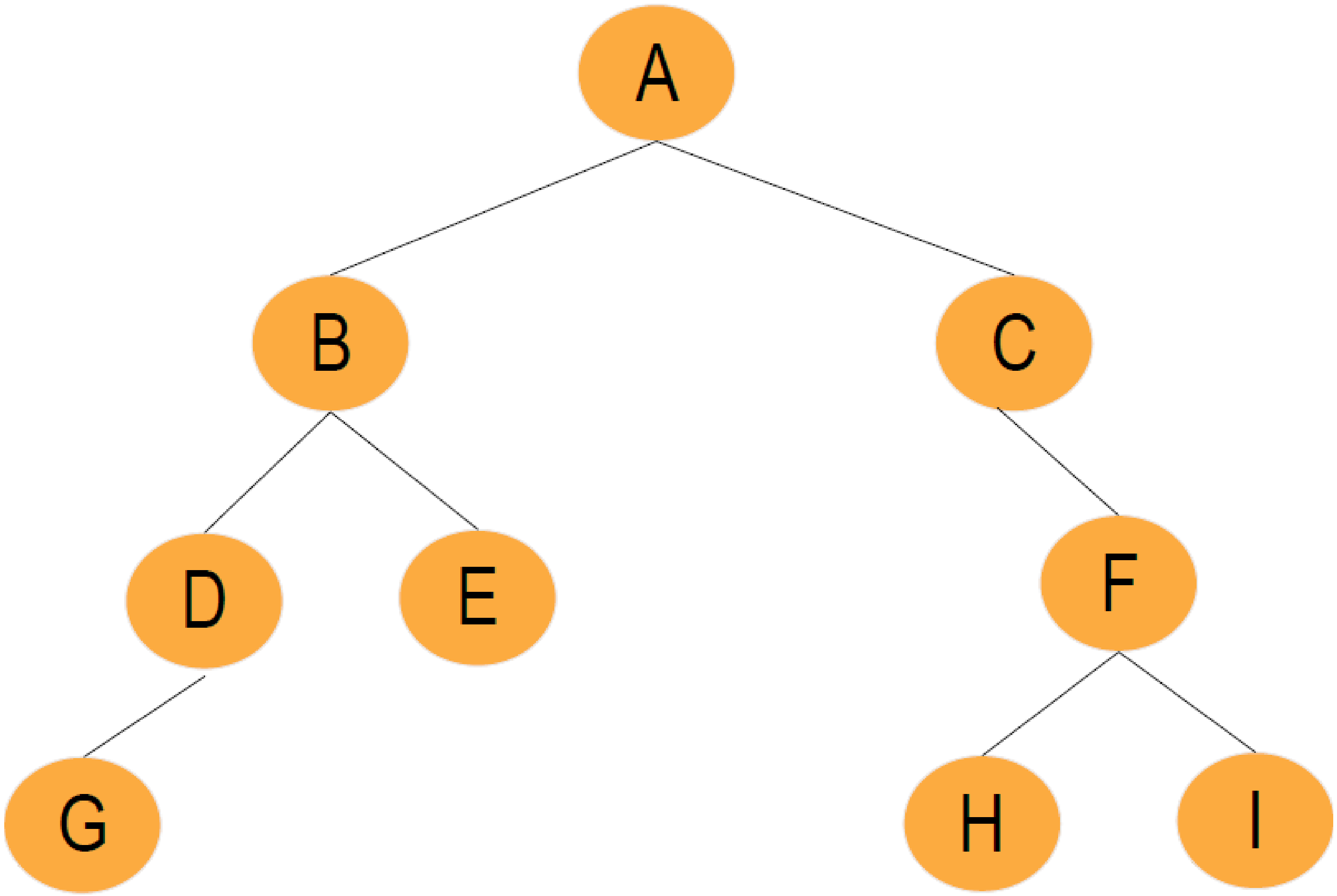
- **Nomenclatura:** T_E e T_D são denominados sub-árvore esquerda e sub-árvore direita de T, respectivamente.

Árvore Binária (AB)

- **Definição:** Seja v um nó de uma árvore binária. O nó u da sub-árvore esquerda (direita) de v, se existir, é denominada **FILHO ESQUERDO** (**DIREITO**) de v.
  - **Observação:** o FILHO ESQUERDO pode existir sem o DIREITO, e vice-versa.

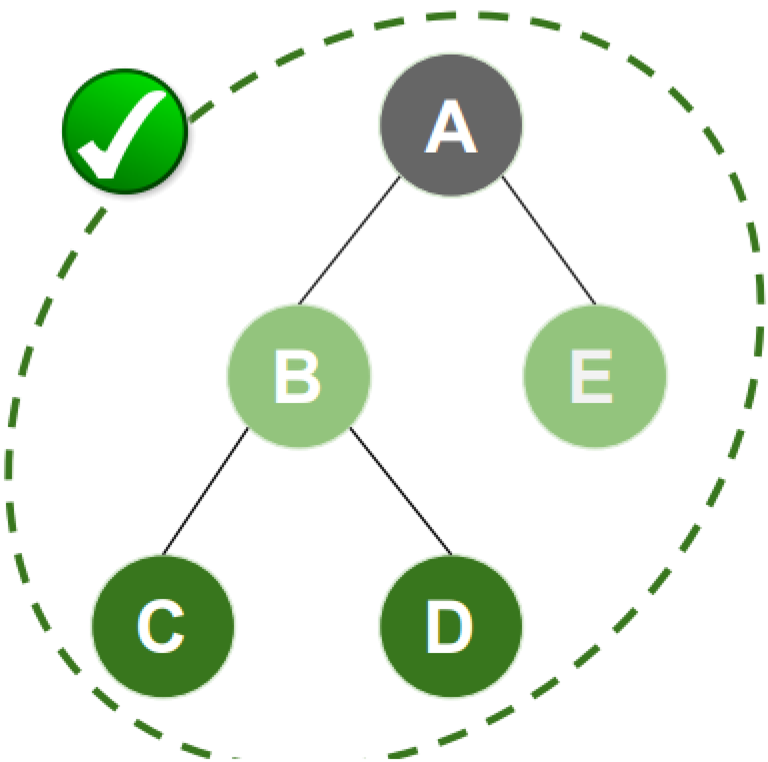
- **Definição:** Se r é a raiz de T, diz-se que T_{Er} e T_{Dr} são as **SUB-ÁRVORES ESQUERDA** e **DIREITA** de T, respectivamente.

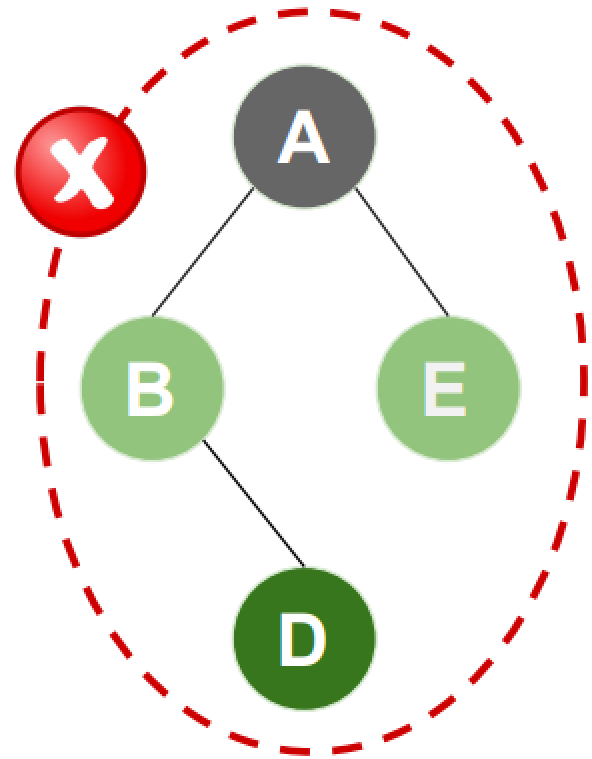
Árvore Binária (AB)



Árvore estritamente binária

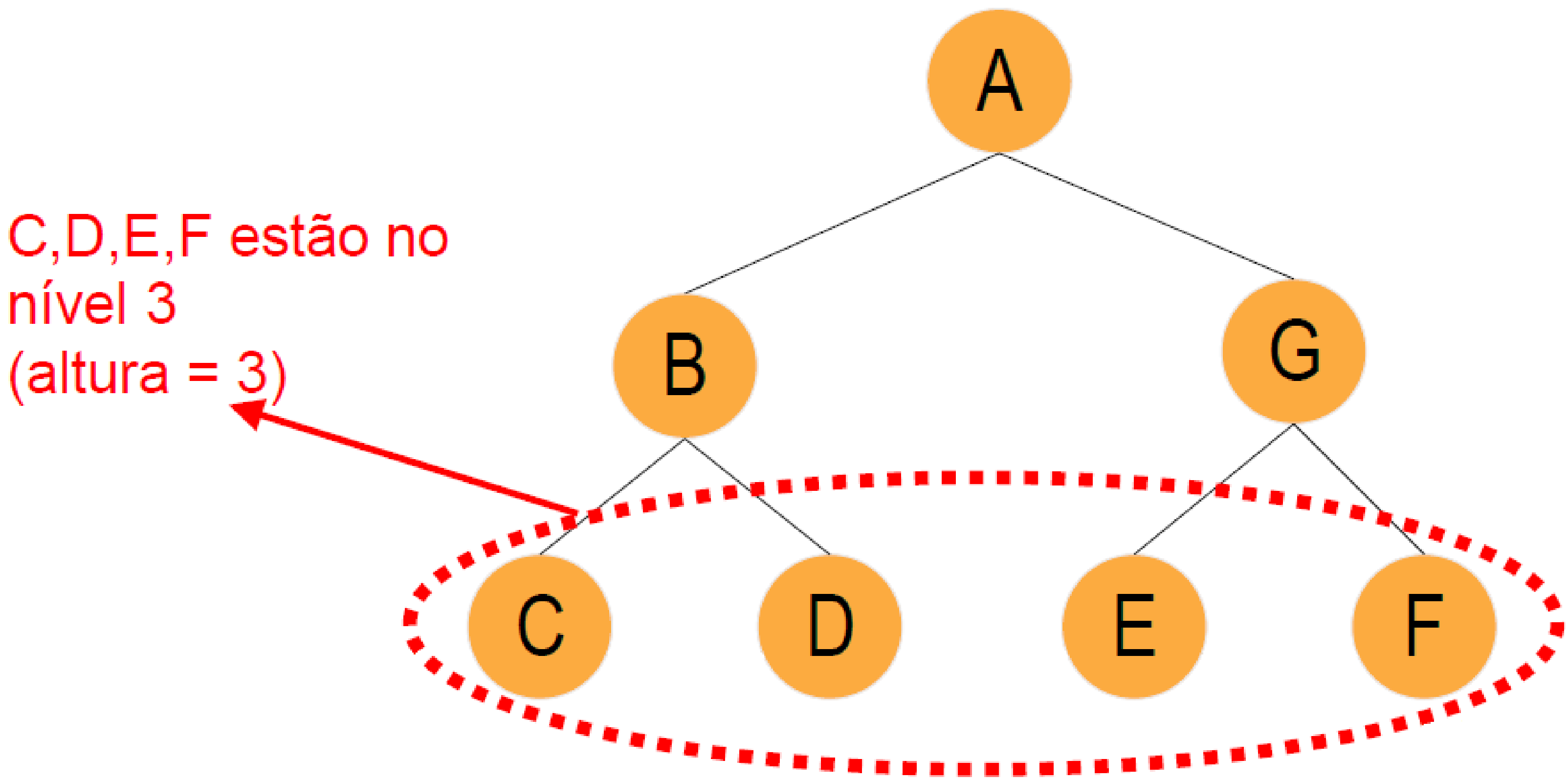
- **Definição:** Uma **Árvore Estritamente Binária** tem nós que têm ou 0 (nenhum) ou dois filhos.
  - Neste caso, os nós internos (não folhas) sempre têm 2 filhos.





Árvore binária completa

- **Definição:** Uma **Árvore Binária Completa** (ABC) é:
  1. Estritamente binária, de nível máximo d;
  2. Todos os seus nós-folhas estão no mesmo nível, d.



Árvore binária completa - propriedades

- Dada uma ABC e sua altura h, é possível calcular o número total de nós na árvore.
- **Exemplo:** uma ABC, com altura 3, terá 7 nós.
  - Nível 1: 1 nó.
  - Nível 2: 2 nós.
  - Nível 3: 4 nós.
  - Número Total de nós: 7.

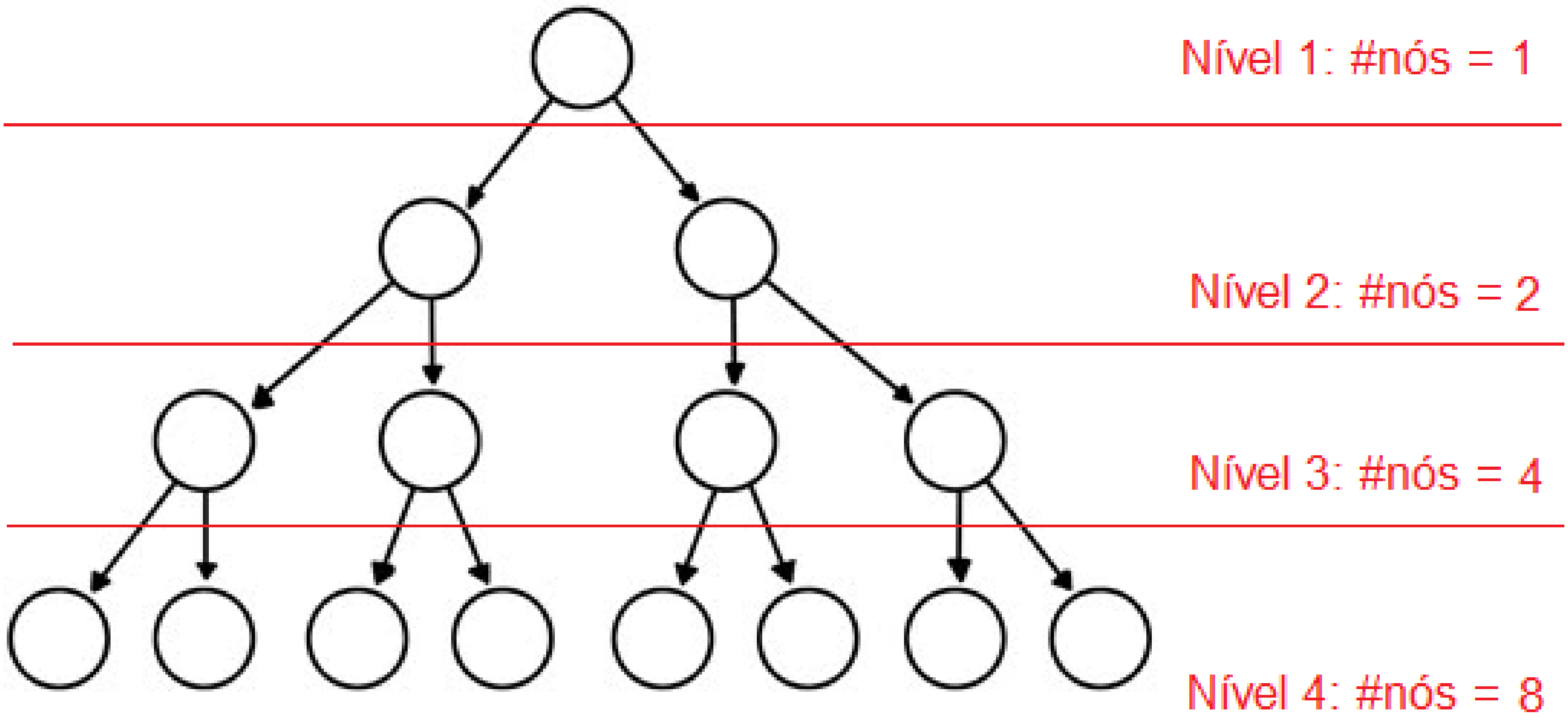
- **Teorema:** Se uma ABC tem altura h, então o número de nós N da árvore é dado pela fórmula:

$$N = N(h) = 2^h - 1$$

Árvore binária completa - propriedades

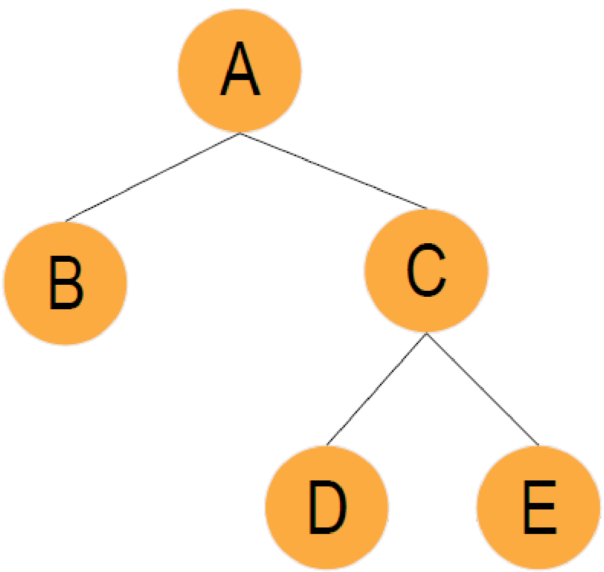
- **Teorema:** Se uma ABC tem altura h, então o número de nós N da árvore é dado pela fórmula:

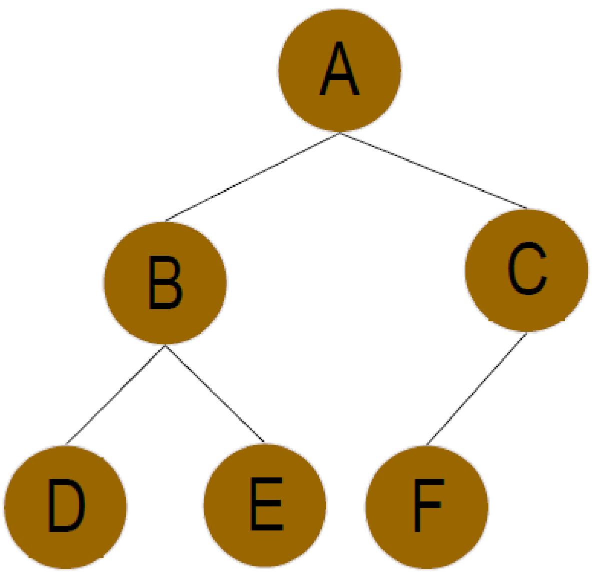
$$N = N(h) = 2^h - 1$$

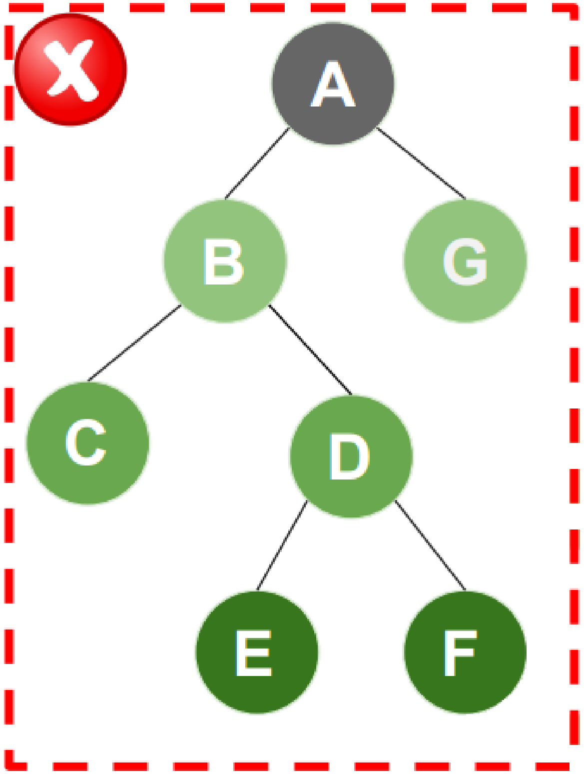


Árvore binária balanceada

- **Definição:** Uma árvore binária T é dita **Balanceada** se, para cada nó de T, as alturas de suas sub-árvores diferem, no máximo, 1.







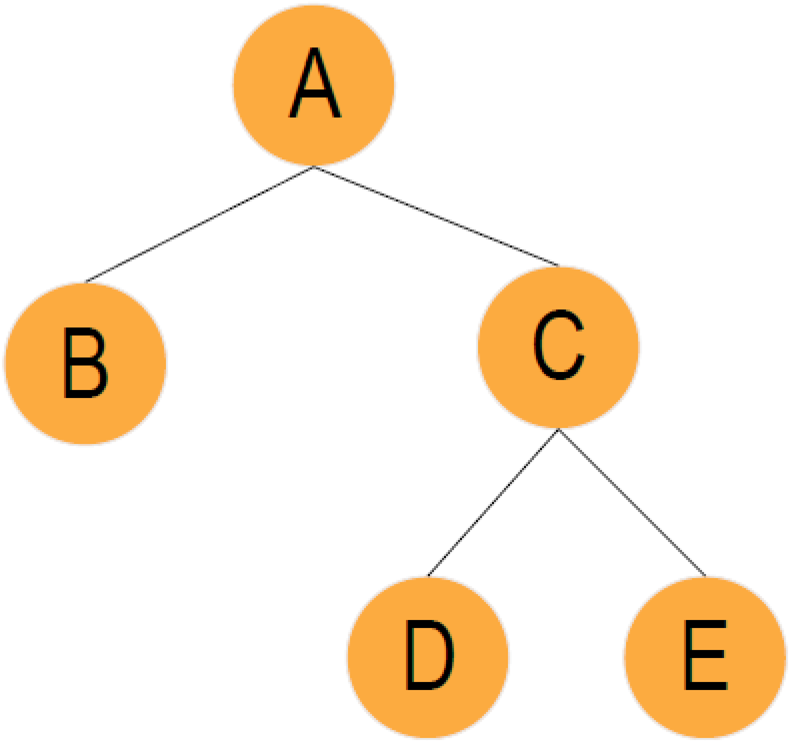
Árvore binária perfeitamente balanceada

- **Definição:** Uma árvore binária T é dita **Perfeitamente Balanceada** se, para cada nó de T, os números de nós de suas sub-árvores esquerda e direita diferem em, no máximo, 1.

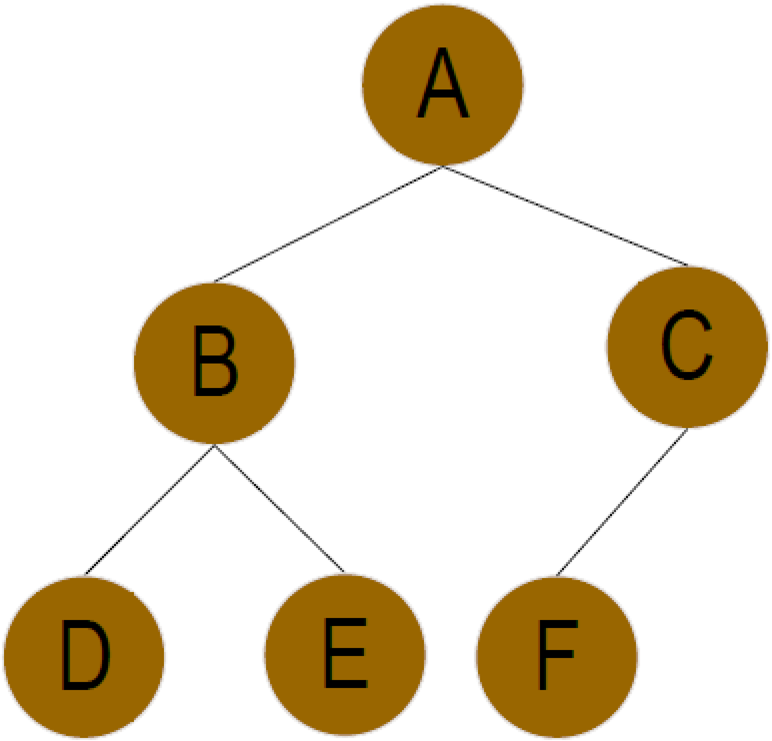
- **Resultados da Análise Cominatória e Teoria dos Grafos:**
  1. Toda AB perfeitamente balanceada é também balanceada, mas o inverso não é necessariamente verdade!
  2. Uma AB com N nós tem altura mínima se e somente se for balanceada.
  3. Se uma AB for perfeitamente balanceada, então ela tem altura mínima.

Árvore binária perfeitamente balanceada

- Árvore balanceada

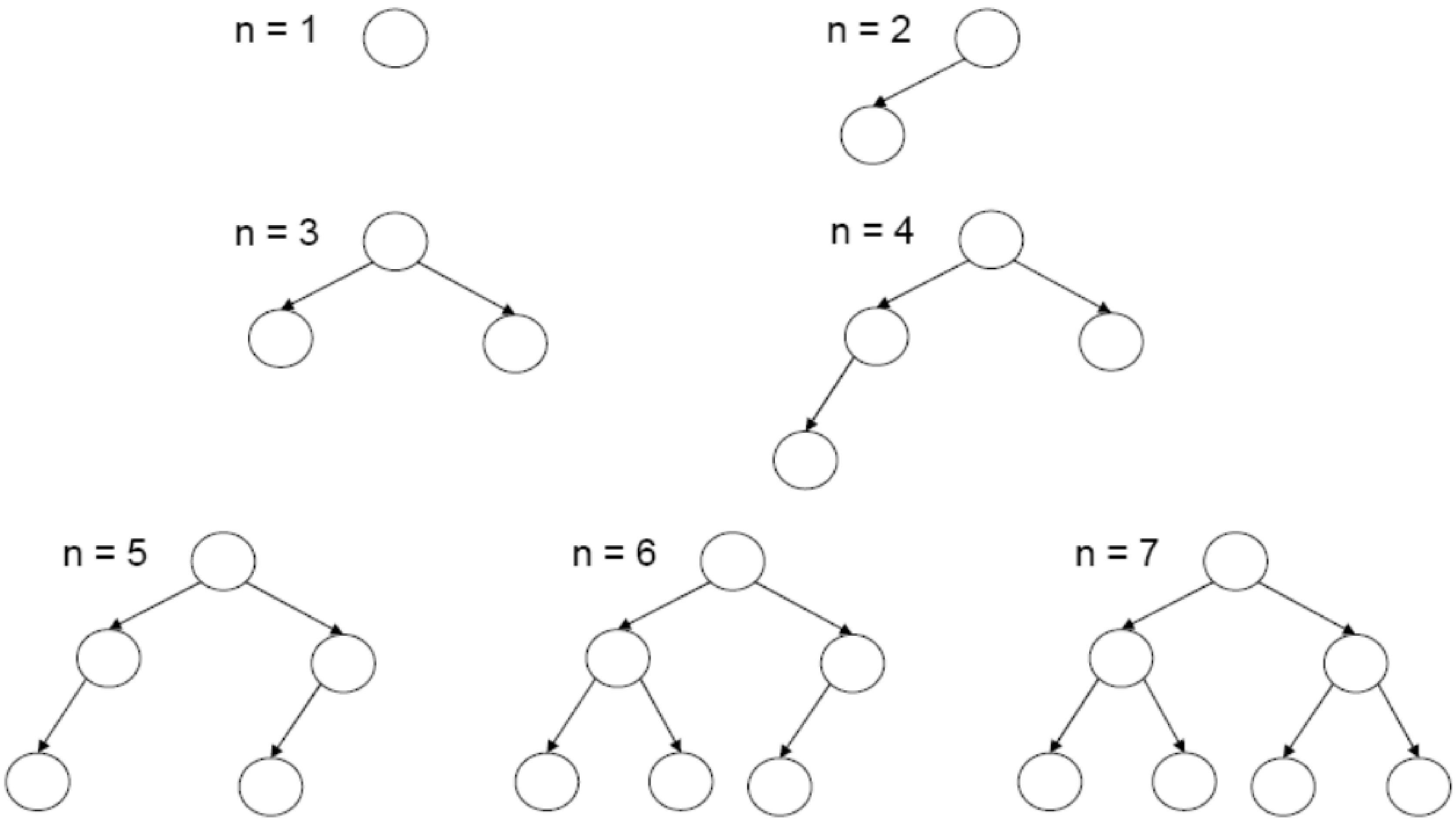


- Árvore perfeitamente balanceada (6 nós e $h_{min} = 3$)

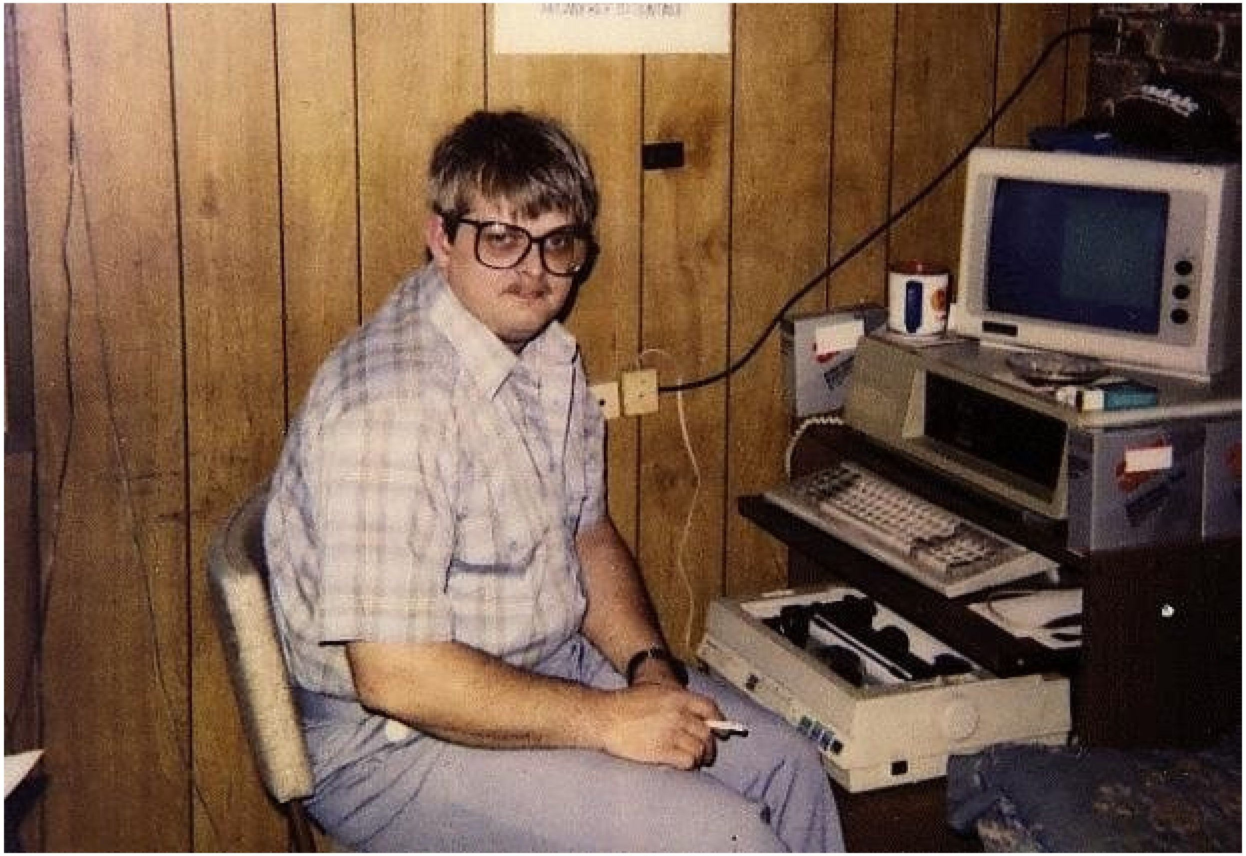


Árvore binária perfeitamente balanceada

- Exemplos: árvores perfeitamente balanceadas (de grau 2)



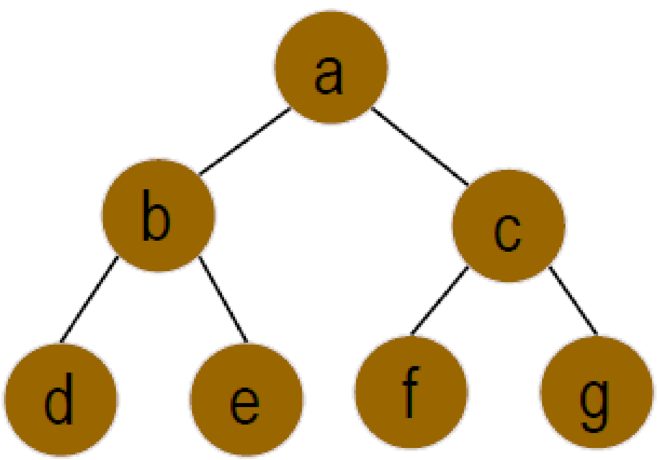
E a implementação dessas TADs, como fica?



TAD 1: Árvore binária completa

- **Objetivo:** Implementação de uma **Árvore Binária Completa** (ABC), com **alocação estática e sequencial**.

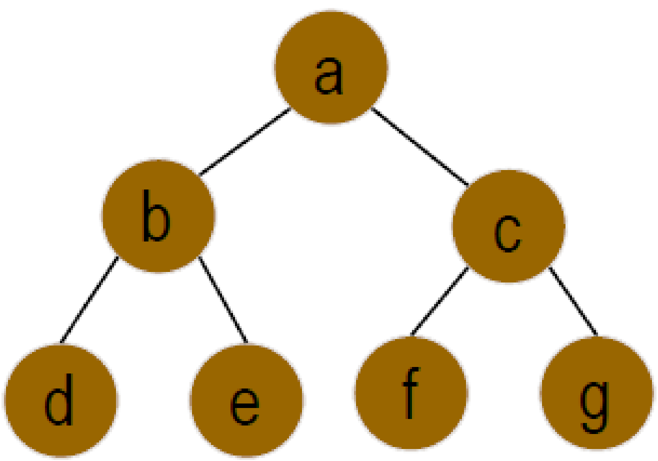
- **Ideia básica:** Armazenar os nós, por nível, em um array.



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

TAD 1: Árvore binária completa

- **Critério lógico:** Se um nó está na posição i, seus filhos estão nas posições $2i$ e $(2i + 1)$.



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

TAD 1: Árvore binária completa

- **Vantagens:**
  - Utiliza um critério matemático simples para estabelecer a conectividade dos nós e seus filhos.
  - Alocação de fácil gerenciamento, pois usa array.

- **Desvantagem:**
  - Espaços vagos se a árvore não for completa por níveis, ou se sofrer algum tipo de eliminação de nós.

Para treinar ...

- Tentar implementar o TAD de árvore estática.
- Faça um exemplo simples de aplicação para visualizar melhor a propagação dos nós da árvore.

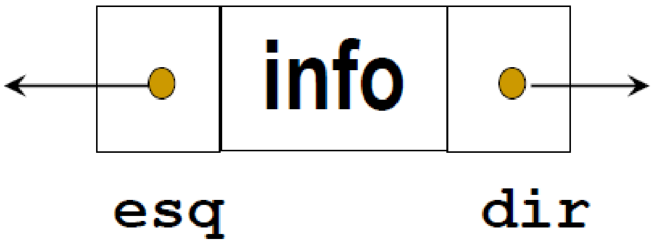
Para treinar ...

- Tentar implementar o TAD de árvore estática.
- Faça um exemplo simples de aplicação para visualizar melhor a propagação dos nós da árvore.

TAD 2: Árvore binária

- **Objetivo:** Implementação de uma **Árvore Binária** (AB), com **alocação dinâmica.**

- **Ideia básica:** Cada nó da AB será um elemento do tipo

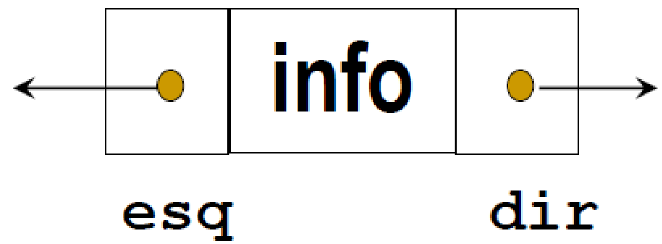


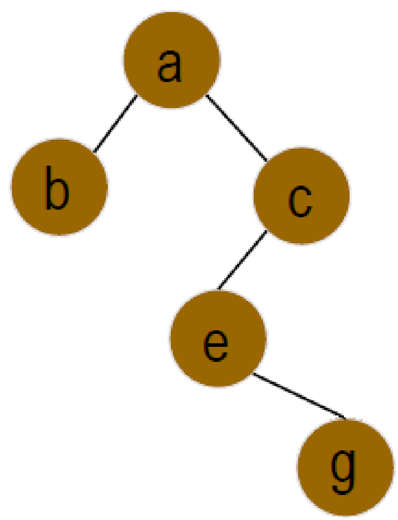
Árvore binária (aloc. dinâmica)

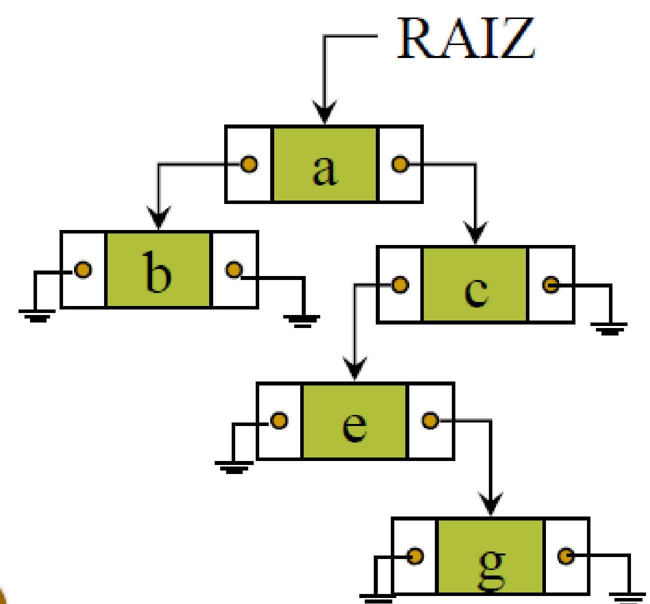
```c
//Tipo registro
typedef struct
{
   int valor;
   //char nome[30];
   //... (caso tenha outros campos)
} tipo_dado;

//Tipo nó
typedef struct no
{
   tipo_dado info;
   struct no *esq;
   struct no *dir;
} no;

//Tipo árvore binária (AB)
typedef no *tree;
```







Implementação das operações

```c
//Cria uma árvore binária vazia
void Definir(tree t)
{
   t = NULL;
}

//Verifica se a AB é uma árvore vazia
int Vazia(tree t)
{
   return (t == NULL);
}
```

```c
//Define nó raiz
void Criar_raiz(tree t, tipo_dado elem)
{
   //o tipo 'tree' é um ponteiro de nó!
   tree raiz = malloc(sizeof(no));
   if (!raiz)
   {
      printf( "Memoria insuficiente!\n" );
      return;
   }

   raiz->esq = NULL;
   raiz->dir = NULL;
   raiz->info = elem;
   t = raiz;
}
```

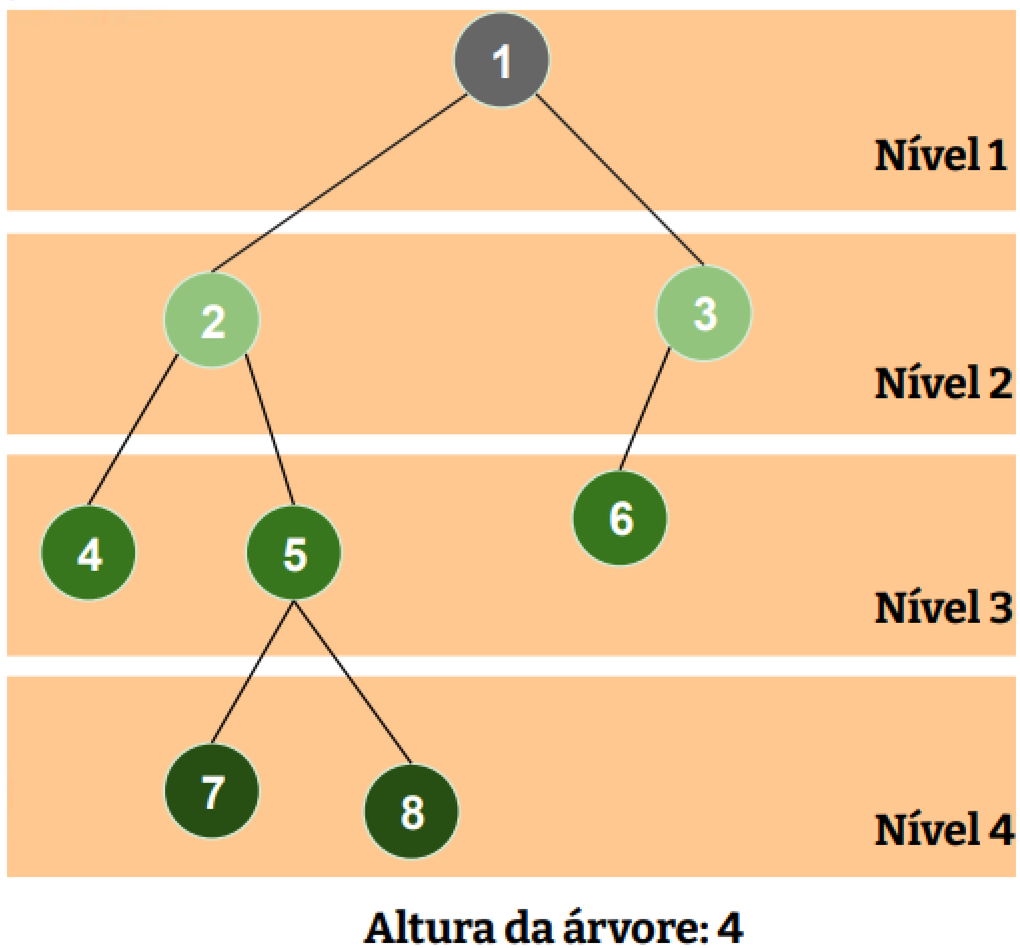
Implementação das operações

```c
//Retorna a altura (profundidade) da AB
int Altura(tree t)
{
   if (t == NULL)
      return 0;

   int altE = Altura(t->esq);
   int altD = Altura(t->dir);

   if (altE > altD)
      return (altE + 1);

   return(altD + 1);
   //altura = max(altE,altD) + 1
}
```

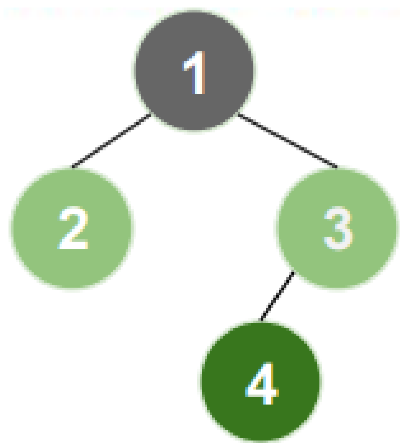


```
//Função recursiva para verificar se uma AB é balanceada
boolean Balanceada(tree t)
{
   if (t == NULL)
      return TRUE;
   else if (t->esq == NULL && t->dir == NULL) //t não tem filhos
      return TRUE;
   else if (t->esq != NULL && t->dir != NULL) //t tem ambas subárvores não-nulas
      return (Balanceada(t->esq) && Balanceada(t->dir)); //recursão
   else if (t->esq != NULL) //t tem um único filho - na esquerda
      return (Altura(t->esq) == 1);
   else                        //t tem um único filho - na direita
      return (Altura(t->esq) == 1);
}

//Função (recursida) para calcular o número de nós da AB
int Numero_nos(tree t)
{
   if (t == NULL)
      return 0;

   int nE = Numero_nos(t->esq);
   int nD = Numero_nos(t->dir);

   return(nE + nD + 1);
}
```



```
//Função (recursiva) para verificar se
//uma AB é perfeitamente balanceada
boolean Perf_balanceada(tree t)
{
   if (t == NULL)
      return TRUE;
   else if (t->esq == NULL && t->dir == NULL)  //t não tem filhos
      return TRUE;
   else if(t->esq != NULL && t->dir != NULL)  //t tem ambas subárvores não-nulas
      return (Perf_balanceada(t->esq) && Perf_balanceada(t->dir)); //recursão
   else if(t->esq != NULL)                    //t tem um único filho - na  esquerda
      return (Numero_nos(t->esq) == 1);
   else                                        //t tem um único filho - na direita
      return (Numero_nos(t->esq) == 1);
}
```

```c
//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 "elem"
boolean Insere_dir(tree pai, tipo_dado elem)
{
    if (pai == NULL)
        return FALSE;

    if (pai->dir != NULL)
    {
        printf("Ja tem filho a direita\n");
        return FALSE;
    }

    //Jeito #1
    Criar_raiz(pai->dir, elem);

    //Jeito #2 ('na mão')
    //tree no = malloc(sizeof(no));
    //no->esq = NULL;
    //no->dir = NULL;
    //no->info = item;
    //pai->dir = no;

    return TRUE;
}
```

```c
//Função p/ adicionar um filho à esquerda de um nó, cujo ponteiro é dado (pai).
//Se o nó não possui filho à esquerda, então cria esse filho com conteúdo "elem"
boolean Insere_esq(tree pai, tipo_dado elem)
{
   if (pai == NULL)
      return FALSE;

   if (pai->esq != NULL)
   {
      printf("Ja tem filho a esquerda\n");
      return FALSE;
   }

   Criar_raiz(pai->esq, elem);

   return TRUE;
}
```

Árvore binária: percursos

- **Objetivo:** Percorrer uma AB de forma a visitar cada nó uma única vez. Isso é importante, pois permite realizar tarefas como imprimir uma árvore, atualizar um campo de cada nó, procurar um elemento, etc.
  - Lembre-se: a árvore não é uma estrutura linear. No entanto…

- **Concepção de percurso e ordem:** Um percurso gera uma **sequência linear de nós**, e assim faz sentido falar de nó predecessor ou sucessor de um nó, segundo um dado percurso.

- **Observação:** Não existe um percurso único para árvores (binárias ou não). Diferentes percursos podem ser realizados, dependendo do tipo de aplicação.

Árvore binária: percursos

- Há 3 tipos de percursos básicos para ABs:
  1. Pré-ordem (Pre-order).
  2. Em-ordem (In-order).
  3. Pós-ordem (Post-order).

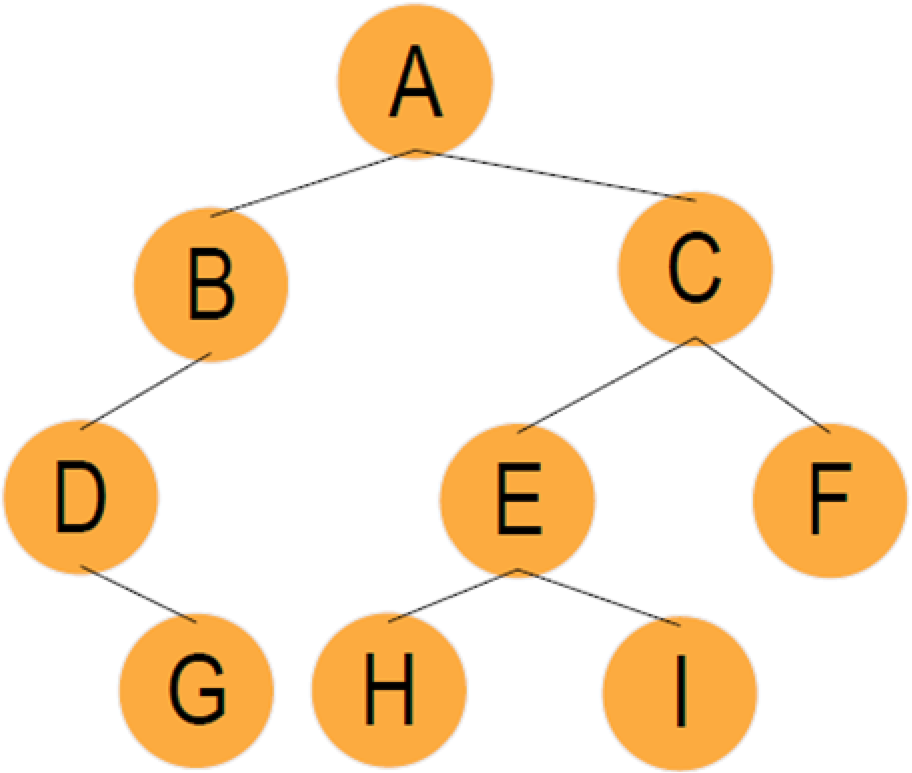
- A diferença entre eles está, basicamente, na **ordem** em que cada nó é alcançado pelo percurso.

- Neste caso, a operação (função) de "Visitar" um nó pode ser:
  - Imprimir o seu valor;
  - Modificar o valor do nó;
  - Outras operações.

```c
//Função "Visita" na forma de impressão de dado
void Visita(tree t)
{
   printf("Valor: %d\n", t->info.valor);
}
```

Percurso em pré-ordem

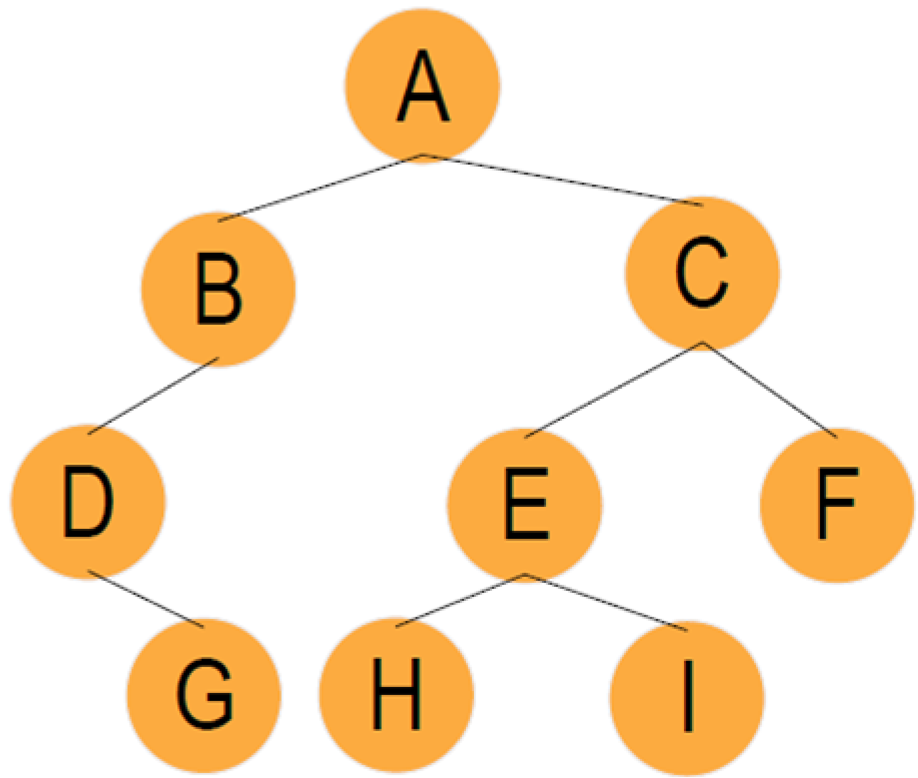
```
//Percorre a árvore em pré-ordem
void Pre_ordem(tree t)
{
    if(t != NULL)
    {
        Visita(t);
        Pre_ordem(t->esq);
        Pre_ordem(t->dir);
    }
}
```



Resultado: ABDGCEHIF

Percurso em in-ordem

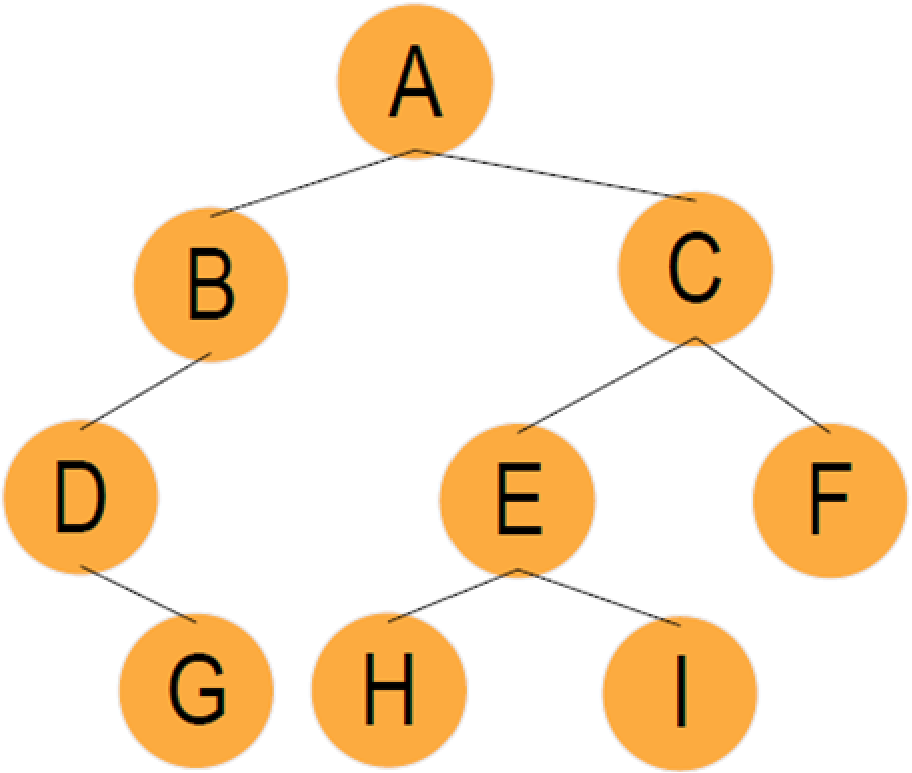
```
//Percorre no esquema in-ordem
void In_ordem(tree t)
{
    if(t != NULL)
    {
        In_ordem(t->esq);
        Visita(t);
        In_ordem(t->dir);
    }
}
```



Resultado: DGBAHEICF

Percurso em pós-ordem

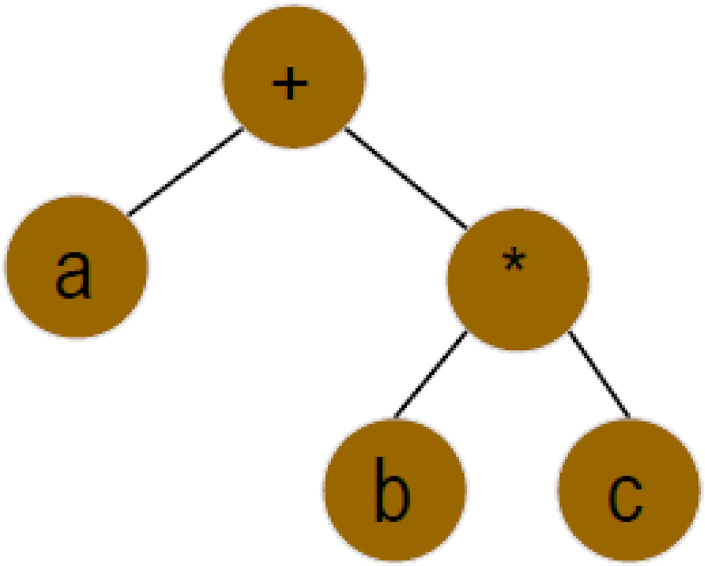
```
//Percorre a árvore em pós-ordem
void Pos_ordem(tree t)
{
    if(t != NULL)
    {
        Pos_ordem(t->esq);
        Pos_ordem(t->dir);
        Visita(t);
    }
}
```



Resultado: GDBHIEFCA

Percursos em AB: exemplo

- Percursos para expressões aritméticas
  - Pré-ordem: +a*bc
  - In-ordem: a+(b*c)
  - Pós-ordem: abc*+



- **Aplicações**
  - Ordenação de arquivos
  - Avaliação de expressões
  - Índices remissivos

Perguntas?

Atividade extra-classe - dupla

- Gerar a árvore abaixo na main.
- Em seguida, imprimir (com qualquer percurso) cada um dos nós dessa árvore.