

**Instituto Luterano de Ensino Superior de Ji-Paraná**  
**Curso Bacharelado em Informática**  
**Estrutura de Dados I**  
**Prof.: José Luiz A. Duizith**

**ÁRVORE (TREE)**

Estrutura hierárquica (+ complexa)

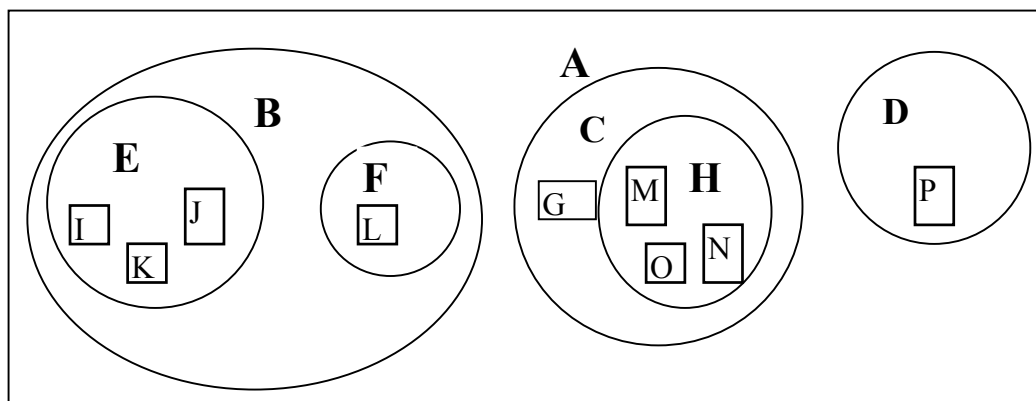
Conceito:

Uma árvore “A” é um conjunto finito de de “n” nodos talque se  $N > 0$  então:

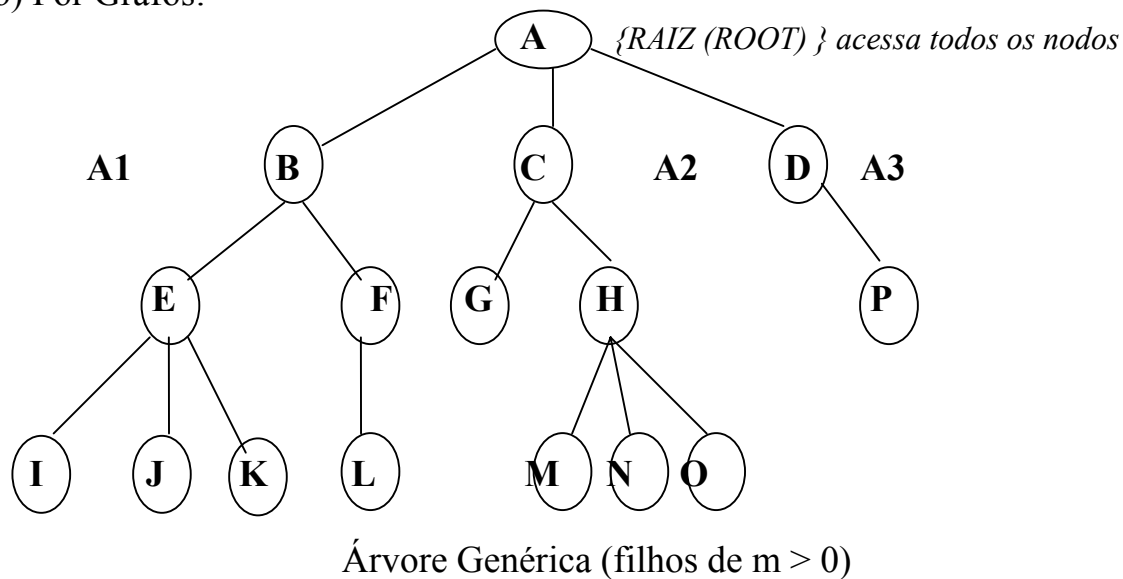
- 1) Existe um nodo especial chamado **raiz árvore**.
- 2) Os restantes  $N-1$  nodos estÃo particionados em “m” conjuntos disjuntos  $A_1, A_2, \dots, A_n$  cada um dos quais é por sua vez uma árvore. Estas árvores  $A_1, A_2, \dots, A_n$  sÃo chamadas de **sub-árvores da raiz**.

Representação:

a) Diagrama de Venn

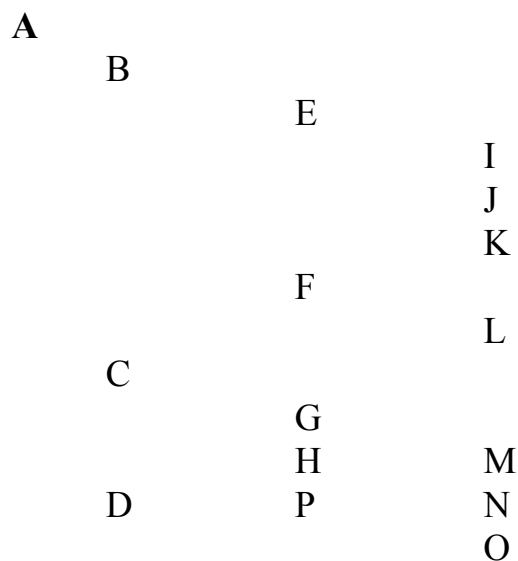


b) Por Grafos:



**Obs.:** Se cortar a Raiz A, ficaria 3 subconjuntos: **A1,A2,A3**.

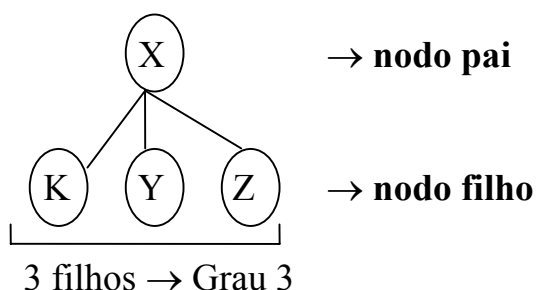
c) Paragrafação:



- Mais a esquerda → representam a raiz
- Mais a direita → subconjunto

**Conceitos:**

- Um nodo raiz é dito ser um nodo **pai** das suas sub-árvores, cujos nodos por sua vez são os nodos **filhos** do nodo raiz.



- **Nodo Irmão:** possuem o mesmo pai

- **Grau** de 1 nodo: é o nº de sub-árvores que o nodo possui:

- se Grau = 0 → então o nodo é chamado extremo terminal ou folha (*nodos que não tem filho*)
- se Grau > 0 → então o nodo é chamado interno

- **Nível** de 1 nodo : é o nº de linhas (arestas) que ligam o nodo ao nodo raiz.

- Nível do nodo raiz = 0

→ no desenho acima, qual o nível de k → nível = 1

↓  
quantas linhas  
percorre até chegar  
a raiz.

- **Altura** de 1 árvore : é definida pelo nível mais alto da árvore.

↘ N° de linhas

*Ex: altura da árvore representada por grafos : 3*

- **Floresta** : conjunto (finito) de árvores disjuntas.

*Se eu tirasse a raiz A então eu ficaria c/ 3 sub-conjuntos, ou seja a floresta seria completa por 3 árvores.*

- **Árvore Ordenada** : aquela na qual a ordem das sub-árvores  $A_1, A_2, \dots, A_n$  é importante na definição.

- **Árvore Orientada** : Aquela na qual a ordem é irrelevante

EX:



- Se A1 e A2 são árvores ordenadas então  $A1 \triangleleft A2$
- Se A1 e A2 são árvores orientadas então  $A1 = A2$  (não importa a ordem)

■ **Caminho (path)** : dado um conjunto de nodos  $n_1, n_2, \dots, n_k$ , será caminho de  $n_1$  até  $n_k$ , se  $n_i$  é pai de  $n_{i+1}$  p/  $i=1, 2, \dots, k-1$ . Dado um nó "S" qualquer de uma árvore existe um único caminho da raiz até o nodo "S".

Ex: na representação por grafo:

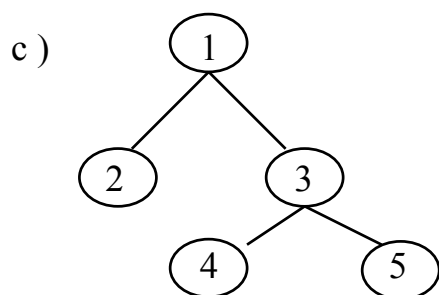
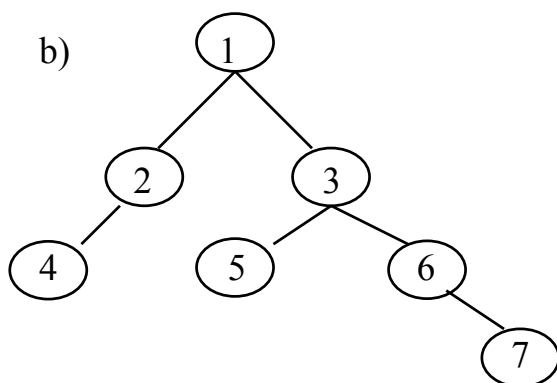
→ caminho até F → A B C

→ caminho até M → A C H M

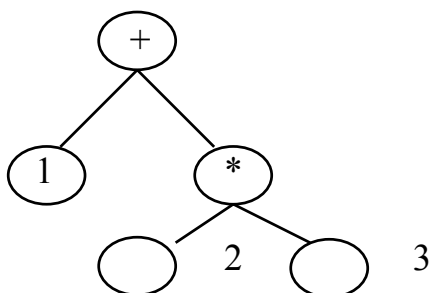
[ *incluo a raiz e  
o próprio elemento*

**Tipos:**

- a) **Árvore genérica** → qualquer nodo pode ter  **$m \geq 0$  filhos**;  
 b) **Árvore binária de Knuth** → qualquer nodo pode ter **no máximo 2 filhos** (0,1,2)  
 c) **Árvore estritamente binária** → qualquer nodo deve ter **0 ou 2 filhos** (0 ou 2)

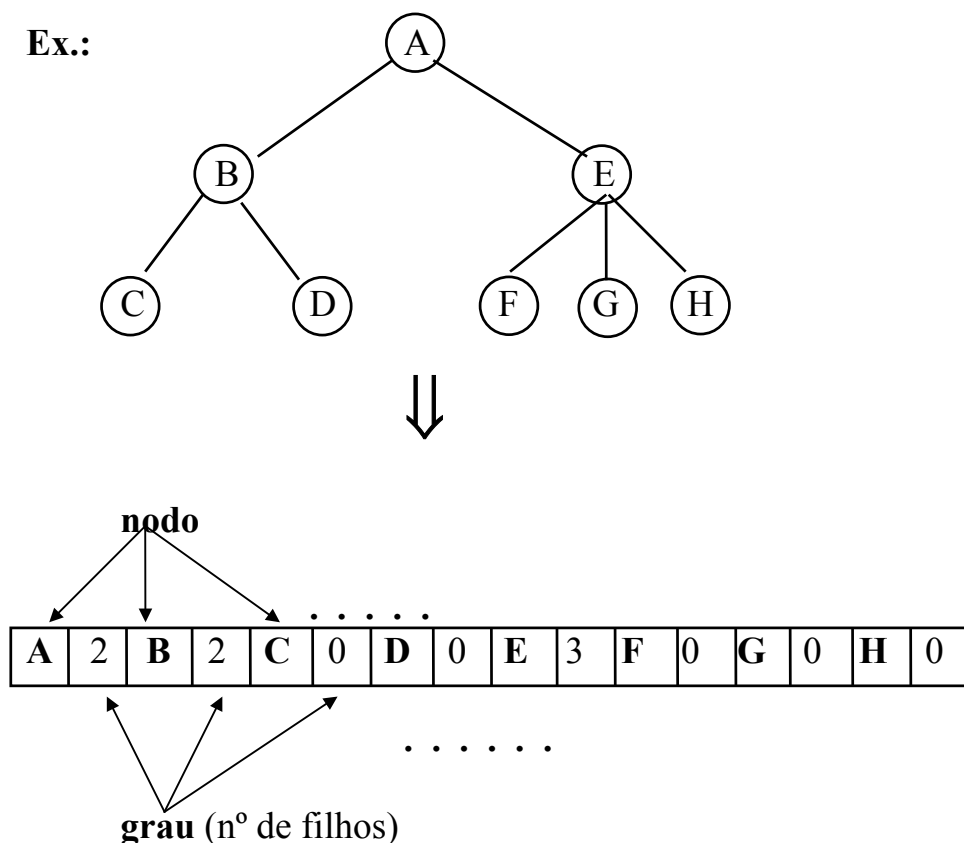


**Ex.:** Rep. Expressão aritmética



## REPRESENTAÇÃO (alocação) DE ÁRVORES EM MEMÓRIA

Ex.:



- Como colocar? **A** tem 2 filhos, então os próximos nodos são filhos de **A**. Um deles é **B**, mas **B** tem 2 filhos, então os próximos nodos são filhos de **B**, que são **C** e **D**, como eles não tem filhos, então o próximo nodo é o outro filho de **A**, que é **E**.

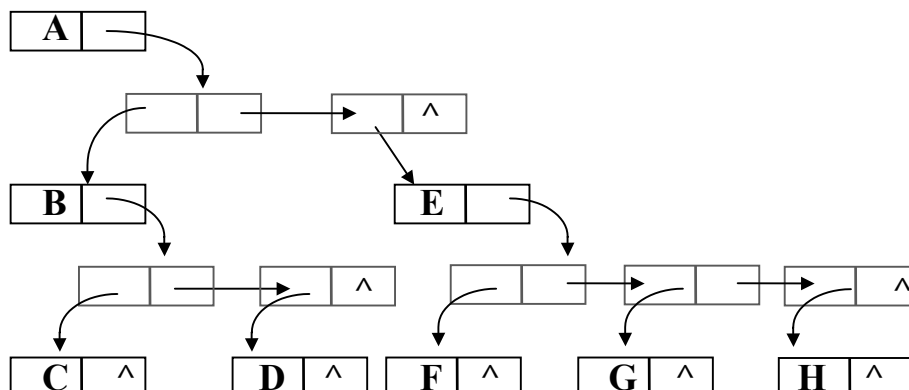
A) Por Contiguidade:

→ dificuldade para operações de manipulação:

- inserção do nodo
- remoção do nodo
- localização do nodo

→ adequada para armazenamento físico permanente (em disco/fita magnéticas) de uma árvore representada por encadeamento.

B) Por Encadeamento (lista de lista)



### Pascal:

```

Type
  Aponta_nodo = ^Nodo;
  Aponta_Aponta = ^Aponta
  Nodo = RECORD
    Dado : Informação
    Filhos : Aponta_Aponta
  End;

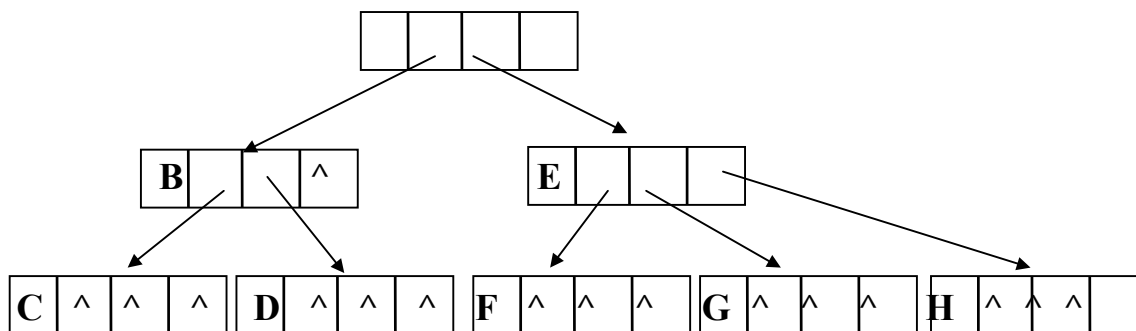
```

```

Aponta = RECORD
  Filho : Aponta_Nodo
  Prox : Aponta_Aponta
End;

```

A      ^



**8 nodos x 4 campos = 32 campos** (cada campo +/- 1 byte)

*17 campos não utilizados (+ de 50% da estrutura está sendo desperdiçado)*

*Não vale a pena trabalhar c/ esta forma em termos de estrutura*

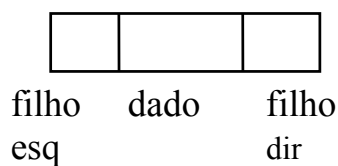
*Solução: - Transformar a árvore genérica em uma árvore binária representativa.*

Todo nodo deve possuir um campo para a informação e tantos campos para referenciar os filhos, normalmente o grau máximo da árvore.

*Quanto maior o grau, maior o desperdício.*

### Árvore Binária (nodo);

max. 2 filhos



### **Pascal:**

```

Aponta_Nodo = ^Nodo
  Nodo = RECORD
    Esq : Aponta_Nodo;
    Dado : Informação;
    Dir : Aponta_Nodo;
  End;

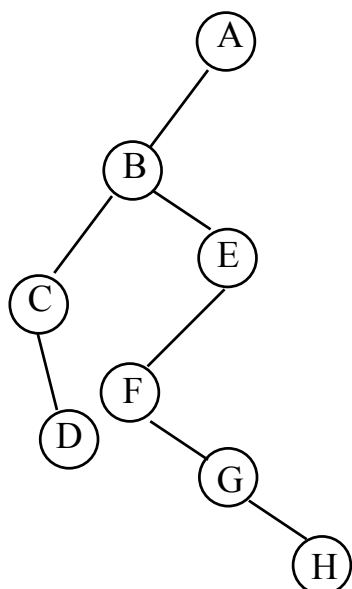
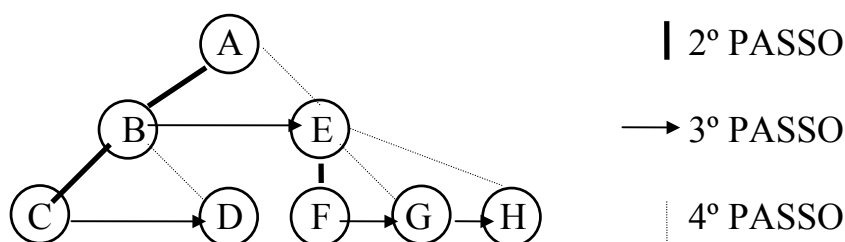
```

## Transformação de Árvore Genérica em Árvore Binária Representativa

1) A raiz da árvore genérica será também a raiz da árvore binária

- 2) Manter a ligação de cada nodo com seu filho mais à esquerda, se o nodo possuir apenas um nodo este é o filho a esquerda (esta ligação mostrará o filho a esquerda na árvore binária).
- 3) Formar uma nova ligação de cada nodo com seu irmão à direita (esta ligação mostrará o filho a direita na árvore binária).
- 4) As ligações restantes são desconsideradas.

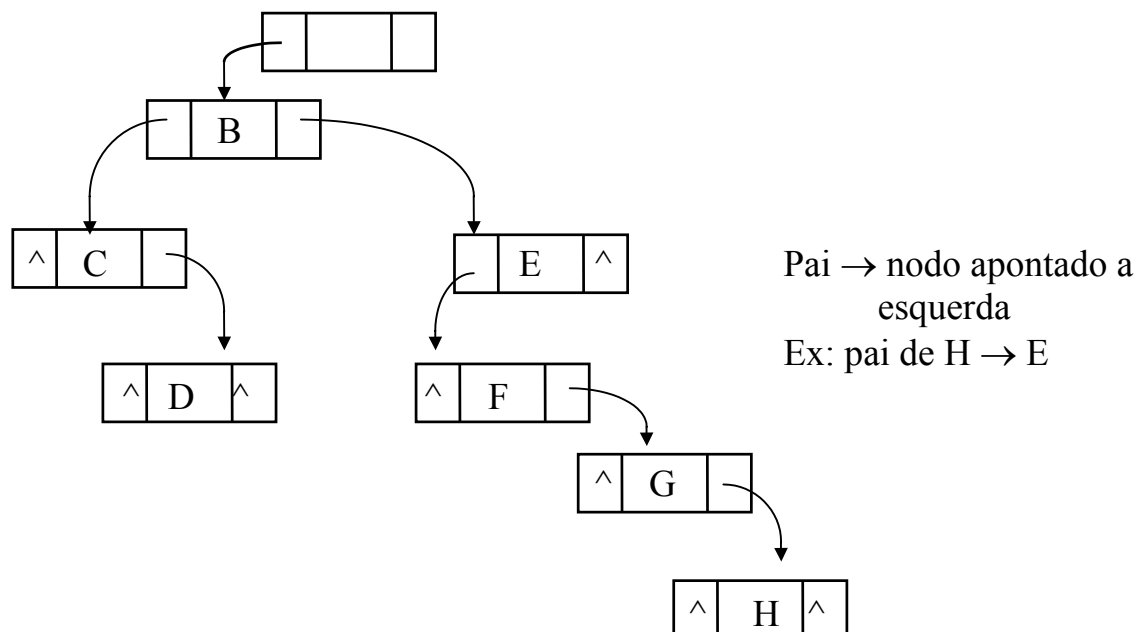
**Ex.:**



- Filhos: 1° a esquerda e depois a direita  
 Ex.: filhos de A: → 1° esquerda (B)  
                           → 2° direita (E)  
 → irmão : a direita

## Árvore Binária em Memória

A ^



8 nodos x 3 campos = 24 campos  
 ↓  
 9 campos desperdiçados

### Caminhamento em Árvores Binárias (percorre todos os elementos)

■ **Caminhamento** → ordem de processamento dos nodos de forma que não

seja necessário passar 2 ou mais vezes pelo mesmo nodo.

→ percorrer todos os nodos da árvore numa ordem pré-determinada processando cada nodo apenas uma vez.

#### Tipos :

##### A) Pré-ordem (pré-fixada)

→ Visita (processa informações) do nodo  $\{Visita : imprime\}$

→ Percorre subárvore à esquerda no nodo ou pré-ordem

→ Percorre subárvore à direita do nodo ou pré-ordem

##### B) In-Ordem (Infixada)

→ Percorre subárvore à esquerda do nodo em In-Ordem

→ Visita (processa informações) do nodo

→ Percorre subárvore à direita do nodo em In-Ordem

##### C) Pós-Ordem (Pós-Fixada)

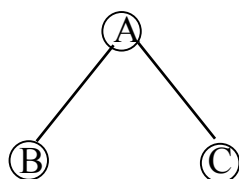
→ Percorre subárvore à esquerda do nodo em Pós-Ordem

→ Percorre subárvore à direita do nodo em Pós-Ordem

→ Visita (processa informação) do nodo

Ex.:

1)

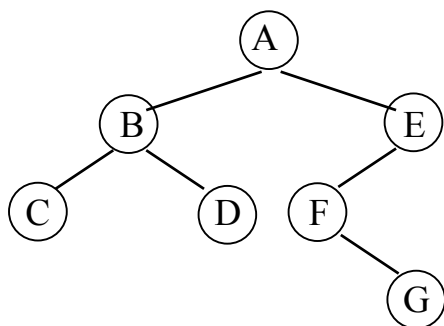


Pré-Ordem :    A    B    C  
 In-Ordem  :    B    A    C  
 Pos-Ordem :    B    C    A

←  
 A raiz fica intercalada se do lado esquerdo  
 existir tantos elementos quanto a direita.

**Começando pela raiz** : Pré-Ordem → Visita (imprime o conteúdo), percorre a esquerda, então encontra o nodo B, ele é diferente de A, então tenho que começar novamente, ou seja, visita o nodo B, percorre a esquerda (não possui nada) então percorre a direita (não possui nada também), assim termina no nodo B. Então volta ao nodo A e percorre a direita. Encontro o nodo C. Devo visitá-lo, percorrer a esquerda (nada) e a direita (nada). Assim termina a Pré-Ordem.

2)



Pré : A B C D E F G

In : C B D A F G E

Pos : C D B G F E A