UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

## Algoritmos e Estrutura de Dados III

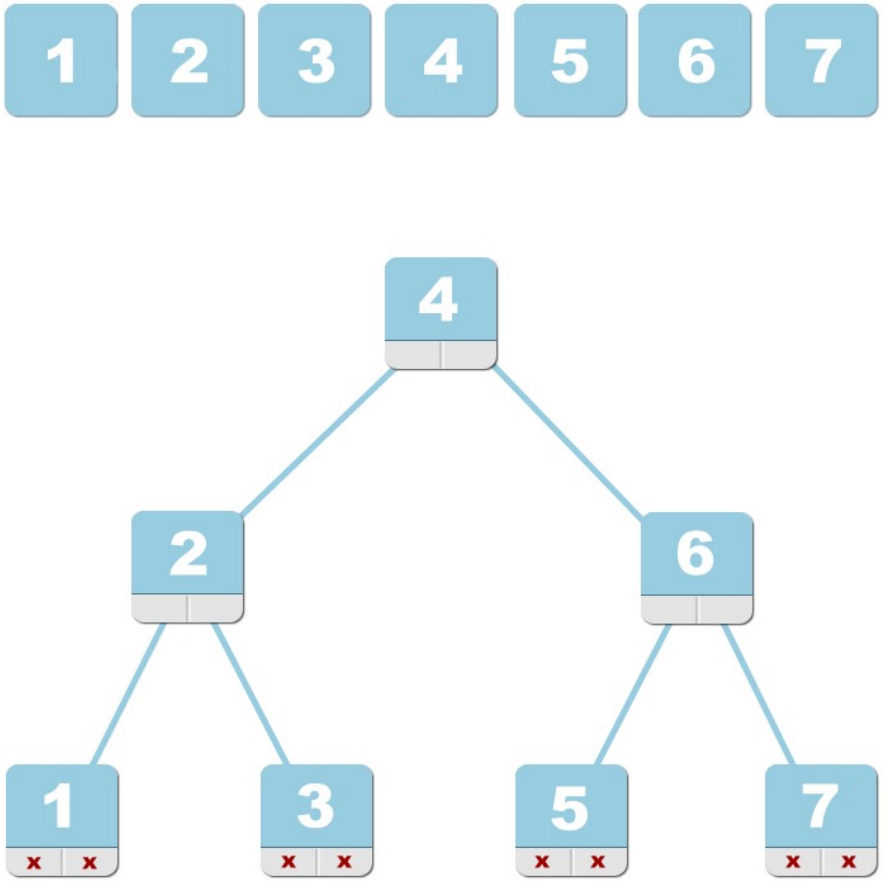
# Árvores

**Aula – 08**

**Thiago Naves**
**Universidade Tecnológica Federal do Paraná**
**Bacharelado em Ciência da Computação**

# Árvores

*Árvores*

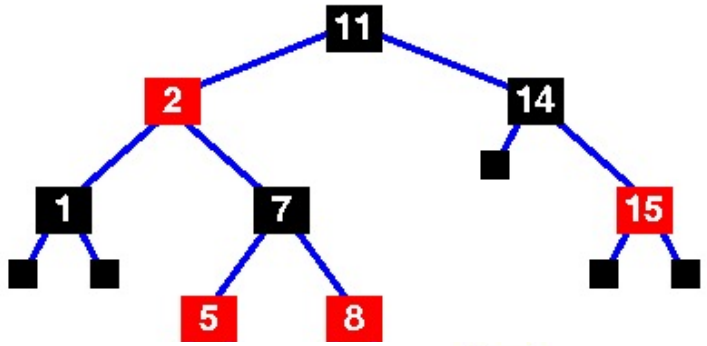


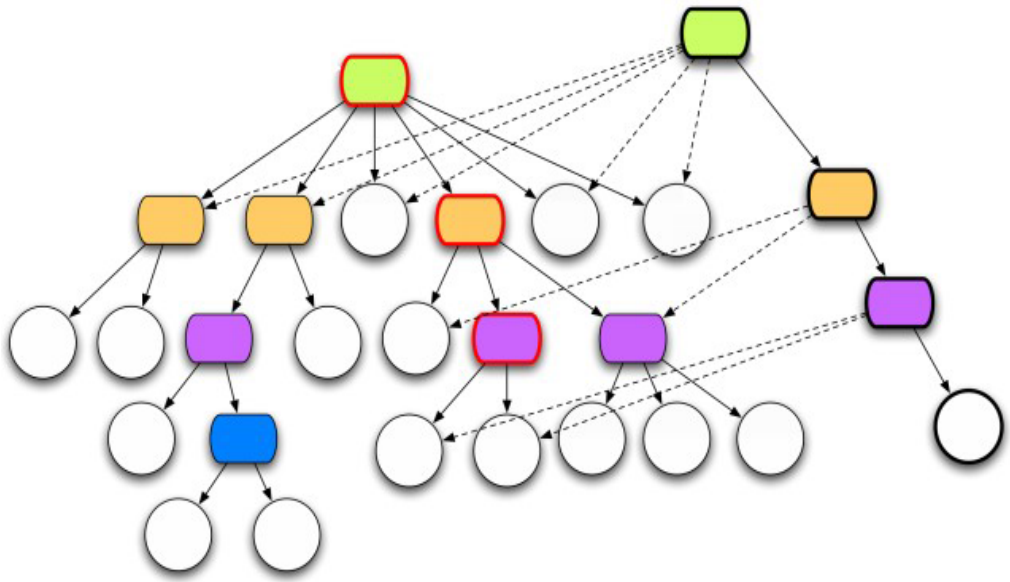
# Introdução

- Uma árvore é uma estrutura de dados mais geral que uma lista ligada.
- Árvores são estruturas de dados que se caracterizam por uma organização hierárquica.
- Essa organização permite a definição de algoritmos relativamente simples, recursivos e de eficiência bastante razoável.

# Introdução

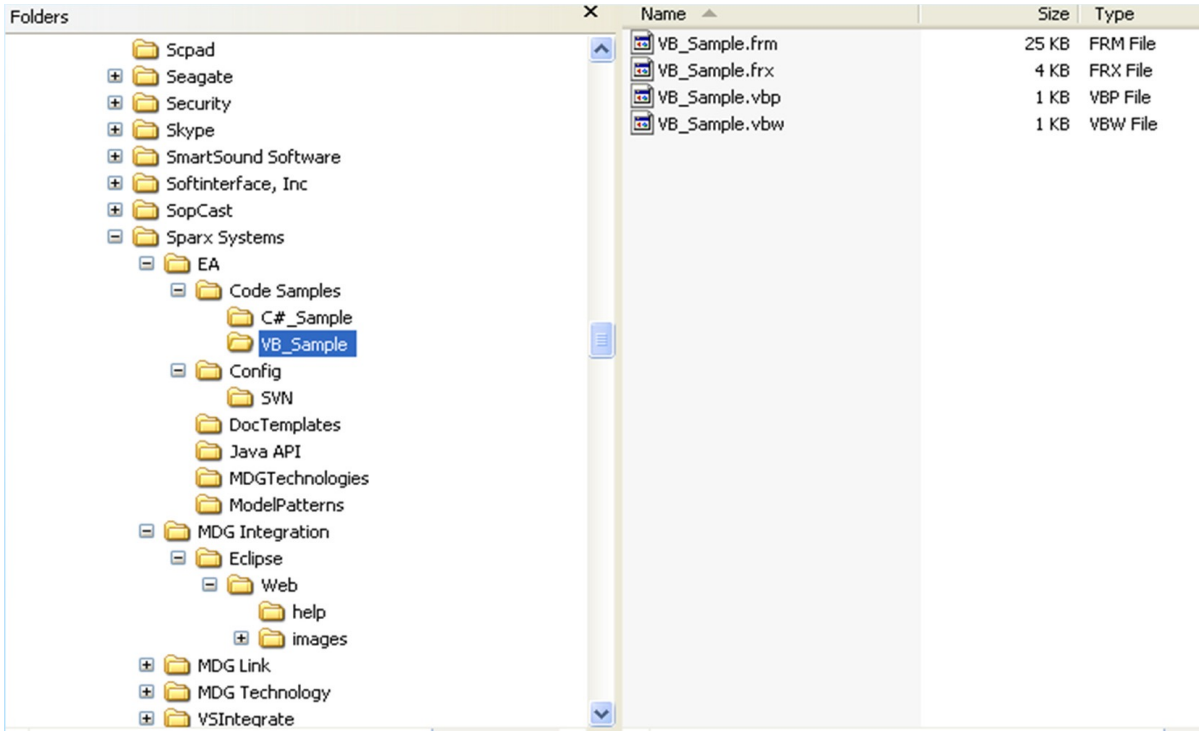
- São estruturas não lineares.
- Representação natural para dados aninhados.
- Muito úteis para resolver uma enorme variedade do problema envolvendo algoritmos.



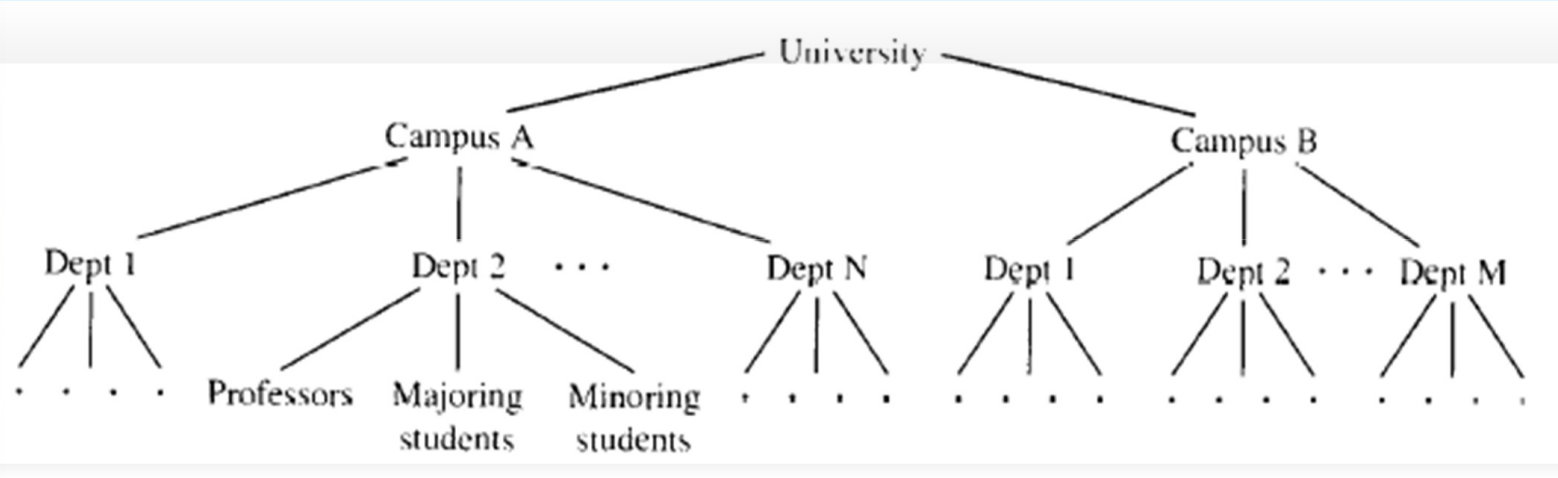
# Motivação

- Qual estrutura de dados o Windows Explorer deve utilizar para gerenciar os arquivos?



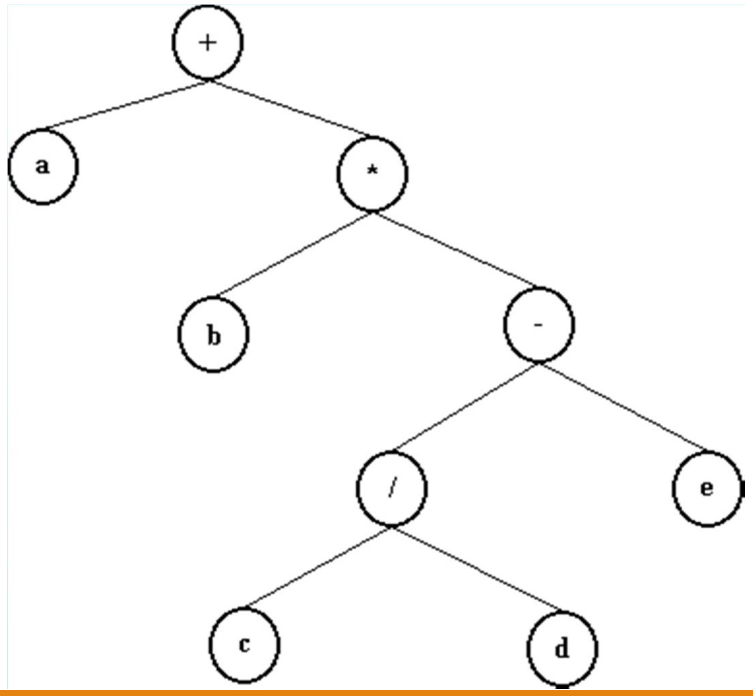
# Motivação

- Hierarquia universitária



# Motivação

- Representação da expressão aritmética:
  (a + (b *( (c / d) - e)))

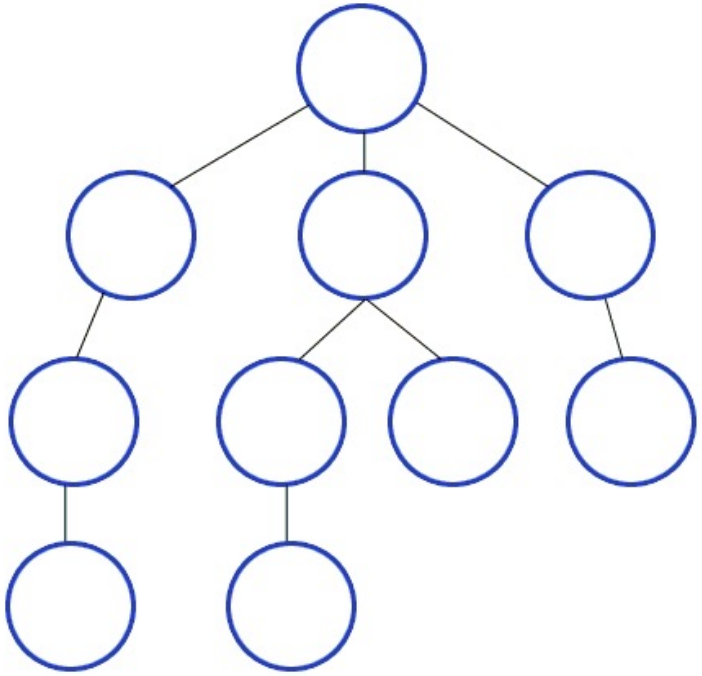


# Motivação

- Listas ligadas
  - São mais flexíveis do que matrizes;
  - Mas são estruturas lineares sendo difícil utilizá-las para organizar representação hierárquica de objetos.
- Pilhas e filas
  - Refletem alguma hierarquia;
  - Mas são limitadas a somente uma dimensão.
- Árvore
  - Estrutura criada para superar limitações de listas ligadas, pilhas e filas;
  - Consiste de nós e de arcos;
  - São representadas com a raiz no topo e as folhas na base (diferente de árvore natural).

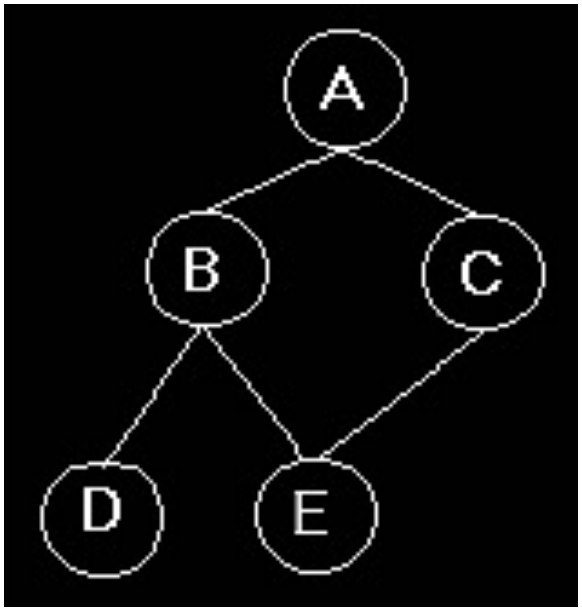
# Representações

## **Representação hierárquica**



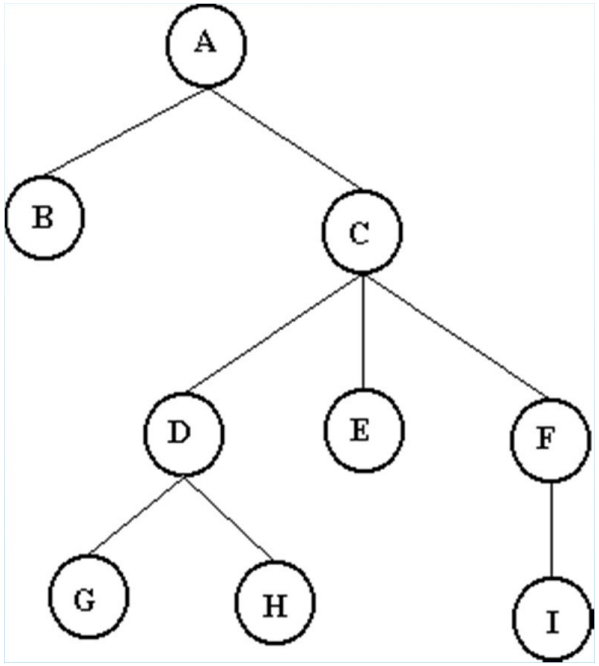
# Representações

- Como, por definição, os subconjuntos $c_1$, $c_2$, ..., $c_n$ são disjuntos, cada nodo pode ter apenas um pai. A representação a seguir, por exemplo, não corresponde a uma árvore.



# Representações

- Hierárquica




- Alinhamento dos nós

**A**

**B**

**C**

**D**

**G**

**H**

**E**

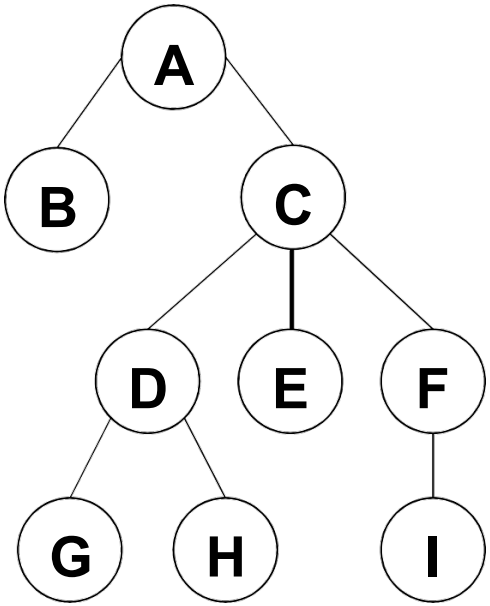
**F**

**I**

# Definições

- Nó: Elemento que contém a informação
- Arco: Liga dois nós
- Pai: nó superior de um arco
- Filho: nó inferior de um arco
- Raiz: nó topo – não tem um nó pai
- Folhas: nós das extremidades inferiores – não têm nós filhos.
- Grau: Representa o número de subávores de um nó.Ver exemplo no próximo slide.
- Grau de uma árvore (aridade): é definido como sendo igual ao máximo dos graus de todos os seus nós. A árvore do próximo slide tem grau 3.

# Definições




- **Graus dos nós**
  - **G(A)=2**
  - **G(B)=0**
  - **G(C)=3**
  - **G(D)=2**
  - **G(E)=0**
  - **G(F)=1**
  - **G(G)=0**
  - **G(H)=0**
  - **G(I)=0**

**Nós Internos**

**Folhas**

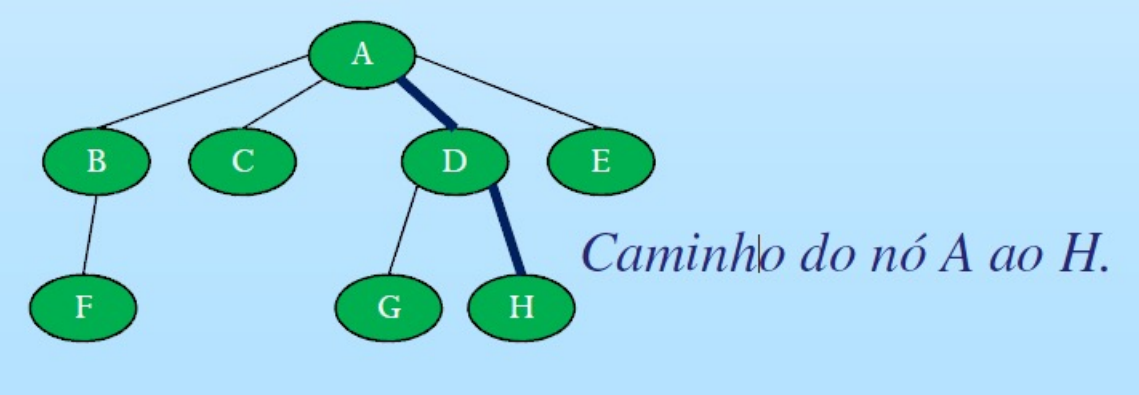
**Grau(T) = 3**

# Definições

- A linha que liga dois nodos da árvore denomina-se aresta;
- Existe um caminho entre dois nodos A e B da árvore, se a partir do nodo A é possível chegar ao nodo B percorrendo as arestas que ligam os nodos entre A e B;
  - Existe sempre um caminho entre a raiz e qualquer nodo da árvore.

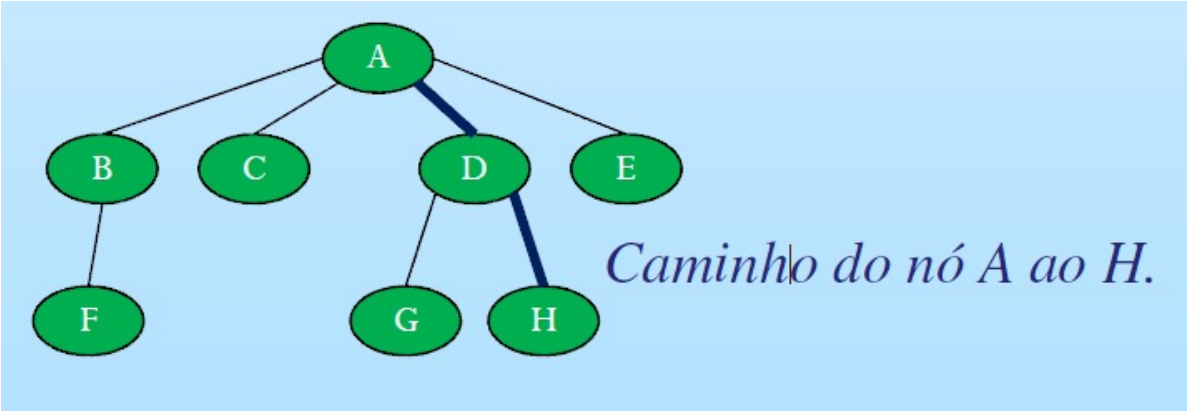
# Árvore

- Cada nó tem que ser atingível a partir da raiz através de uma sequência única de arcos, chamados de caminho.
- Comprimento do caminho: o número de arcos do caminho
  - O caminho de A até H tem comprimento 2
- Nível de um nó: é a sua distância da raiz da árvore. A raiz tem nível 0. Na fig. abaixo, o nó A tem nível 0; os nós B, C, D e E têm nível 1 e os nós F, G e H têm nível 2.



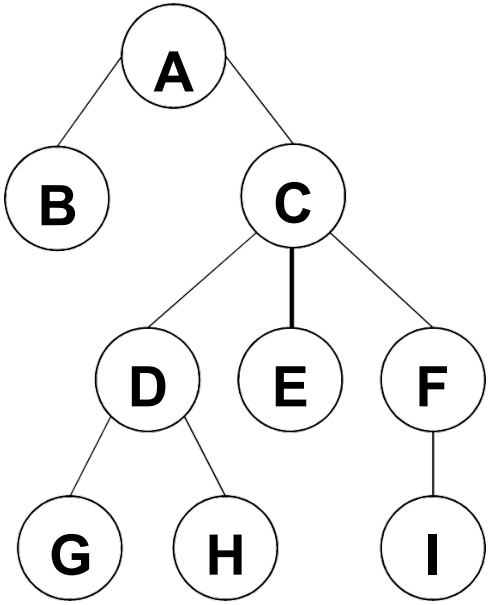
# Árvore

- Altura (ou profundidade) é o nível do nó folha que tem o mais longo caminho até a raiz.
  - A altura da árvore abaixo é igual a 3.
  - A árvore vazia é uma árvore de altura -1, por definição.
  - Uma árvore com um único nó tem altura 1.
- Toda árvore com n>1 nós possui no mínimo 1 e no máximo n-1 folhas.



# Árvore

Exemplo de níveis e altura da árvore)



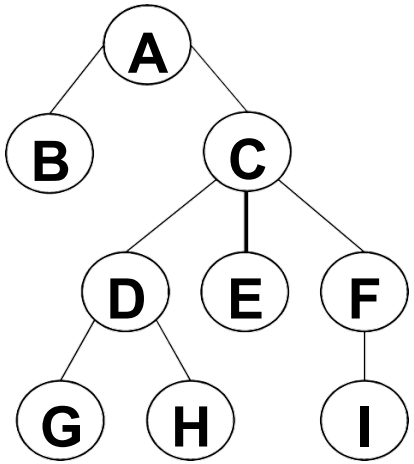

| NÍVEIS | |
|---|---|
| A | 0 |
| B, C | 1 |
| D, E, F | 2 |
| G, H, I | 3 |

**h(T) = 4**

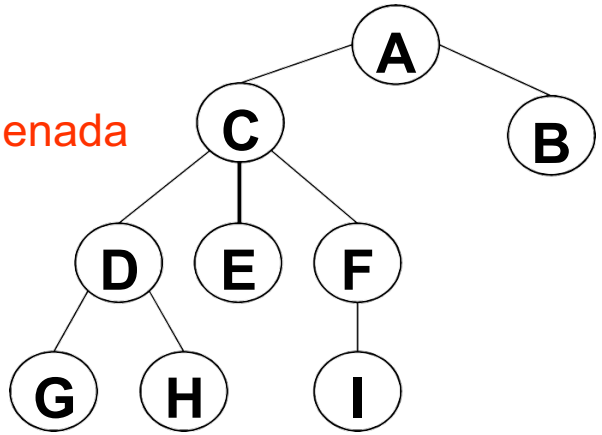
# Árvore

- Árvore Ordenada
  - Os filhos de cada nó estão ordenados (assume-se ordenação da esquerda para a direita)

ordenada

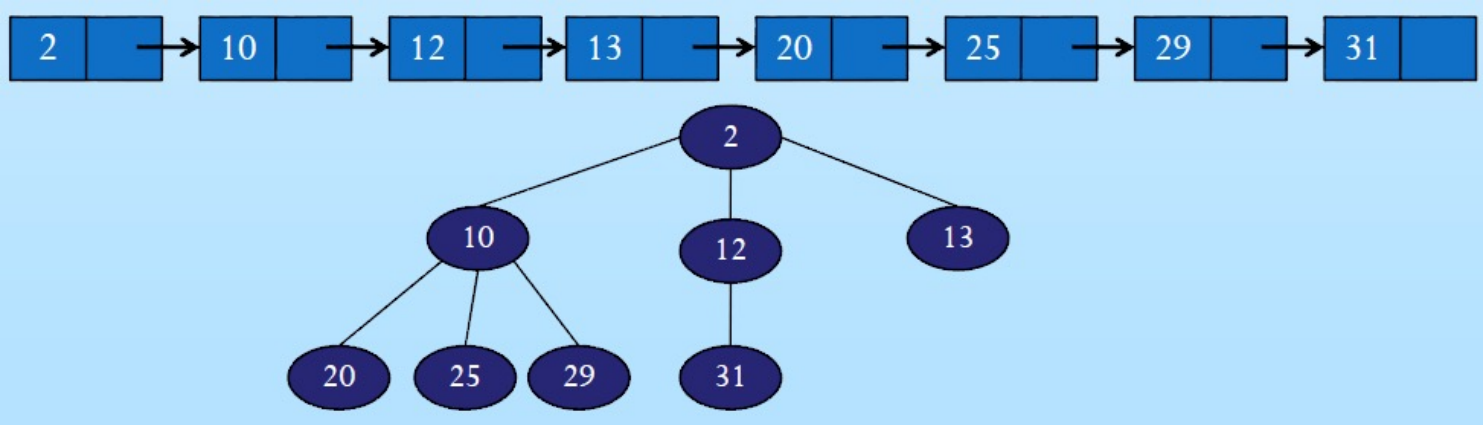



desordenada



# Árvore

- A definição de árvore não impõe qualquer condição sobre o número de filhos de um nó:
  - Pode variar de 0 a qualquer inteiro
- Árvores são muito utilizadas em sistemas gerenciadores de banco de dados.
- Considerando a lista encadeada e a árvore abaixo, qual pesquisa é mais rápida para achar um valor (chave)?

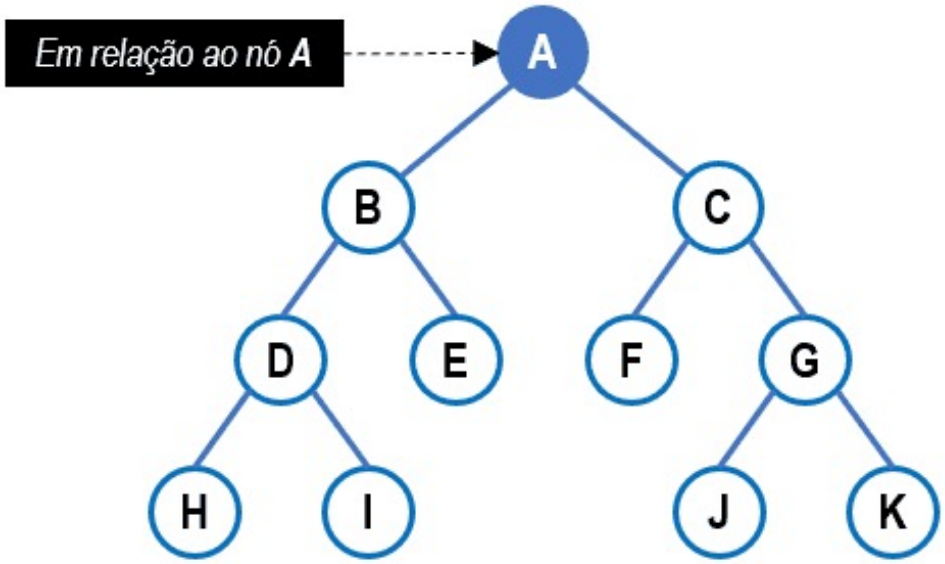


# Árvores Binárias

- A inclusão de limitações estruturais define tipos específicos de árvores.
- Até agora, as árvores vistas não possuíam nenhuma limitação quanto ao grau máximo de cada nodo.
- Uma ***árvore binária*** é uma árvore cujo grau máximo de cada nodo é 2. Essa limitação define uma nomenclatura específica:
  - Os filhos de um nodo são classificados de acordo com sua posição relativa à raiz.
  - Assim, distinguem-se o filho da esquerda e o filho da direita e, consequentemente, a subárvore da esquerda e a subárvore da direita.

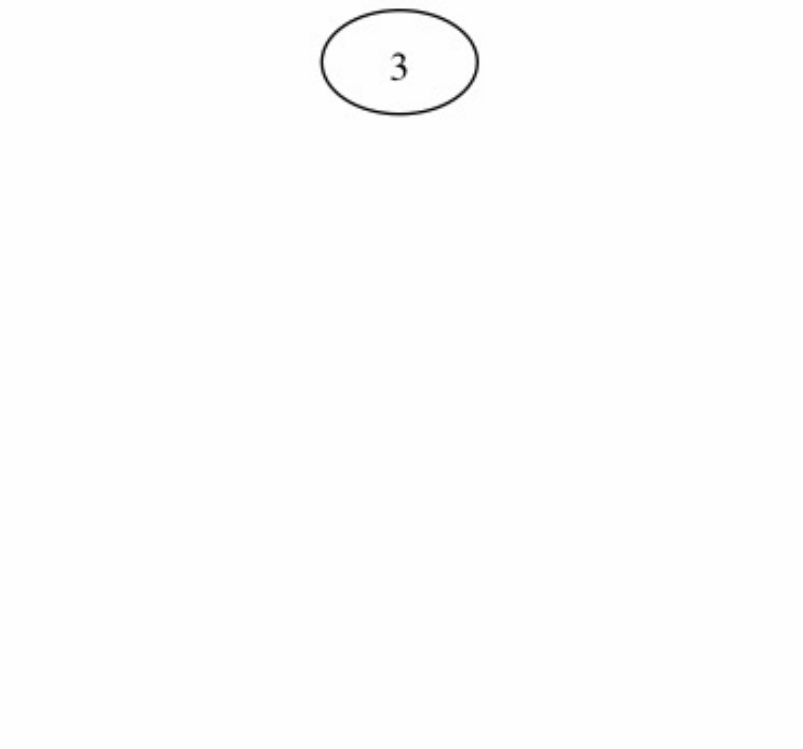
# Árvores Binárias

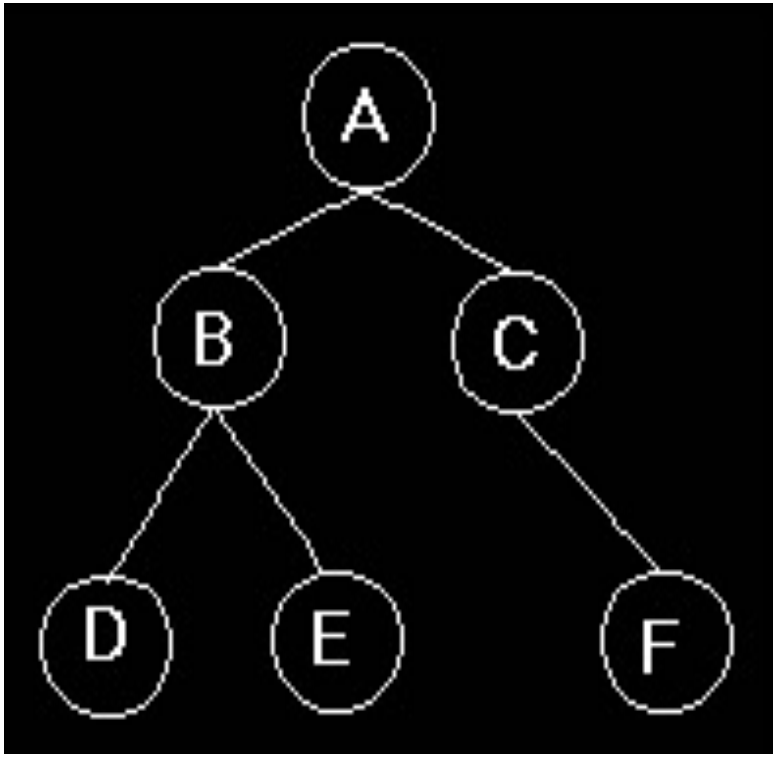
- Assim, distinguem-se o filho da **esquerda** e o filho da **direita** e, consequentemente, a **subárvore** da esquerda e a **subárvore** da direita.



# Árvores Binárias

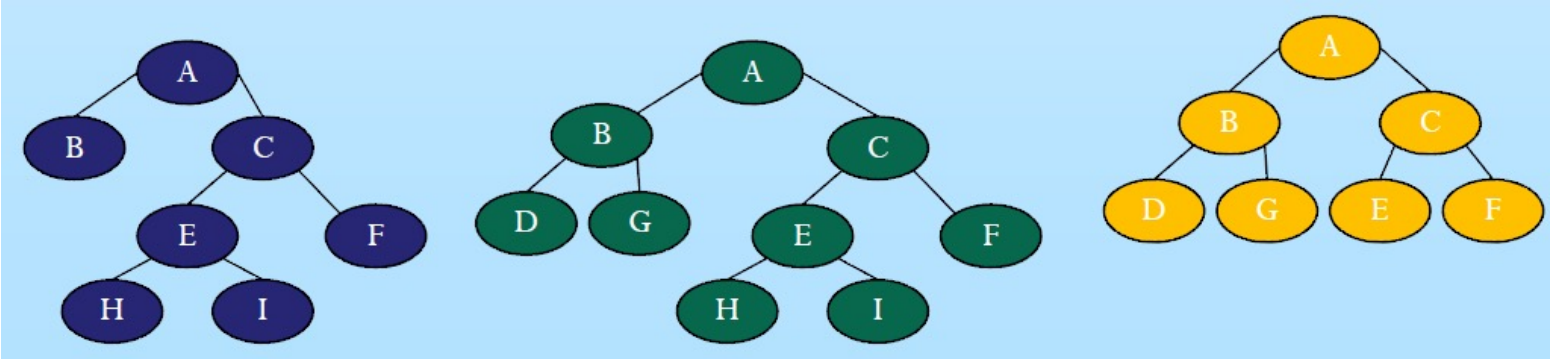
## Exemplo de árvore binária



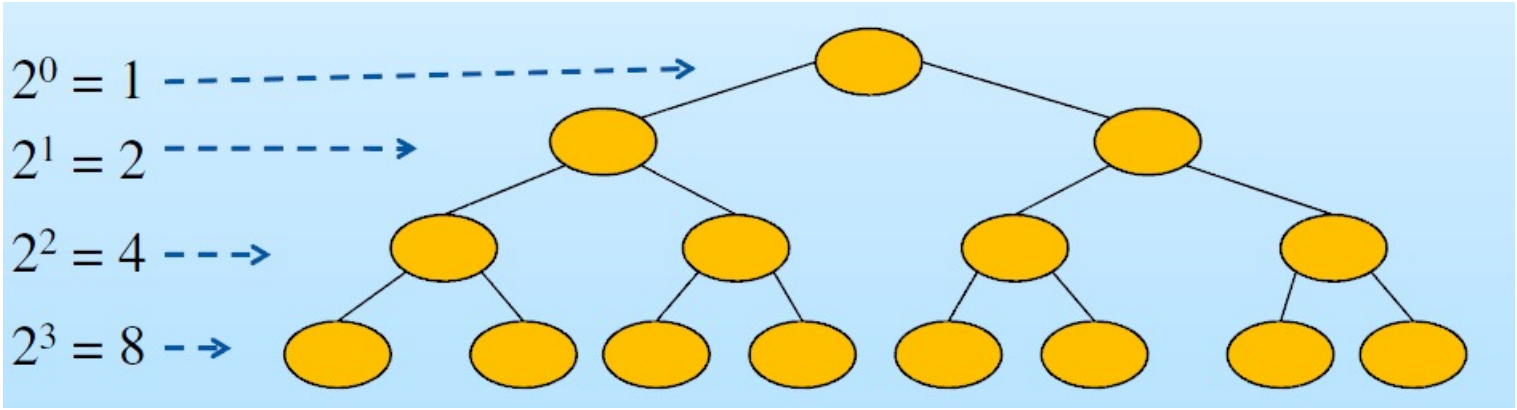
# Árvores Binárias

- Árvore estritamente binária
  - Cada nó possui 0 ou 2 filhos.
- Árvore binária completa apresenta a seguinte propriedade
  - Se v é um nó tal que alguma subárvore de v é vazia, então v se localiza ou no último (maior) ou no penúltimo nível da árvore.
- Árvore binária cheia apresenta a seguinte propriedade:
  - Se v é um nó tal que alguma subárvore de v é vazia, então v se localiza no último (maior) nível da árvore. v é um nó folha.



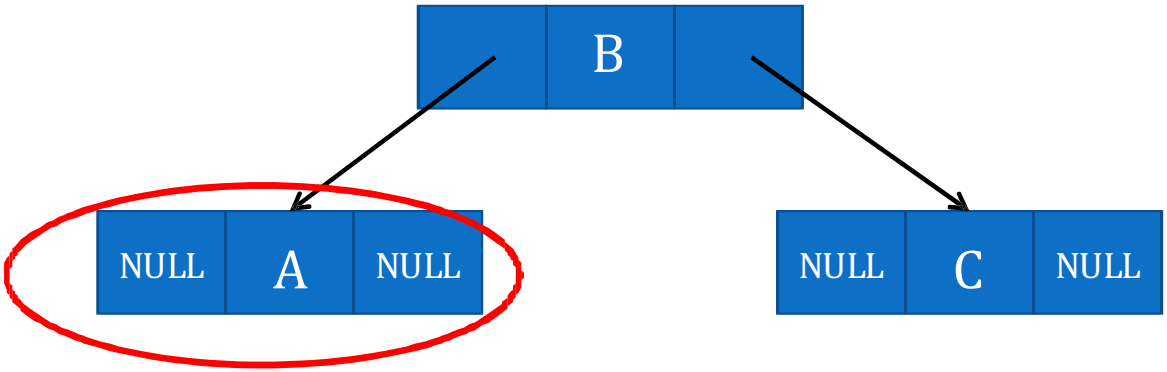
# Árvore binária

- Em árvore binária cheia o número de nós do nível i é igual a $2^i$.
- Consequentemente, em qualquer árvore binária existe no máximo $2^i$ nós no nível i.



# Árvore binária

- Representação encadeada de árvores binárias
- Um nó será formado por um registro composto de:
  - Campo de informação
  - Ponteiro para nó esquerdo
  - Ponteiro para nó direito




Nó da árvore

# Modelagem: nodo de uma árvore binária

Necessitamos:

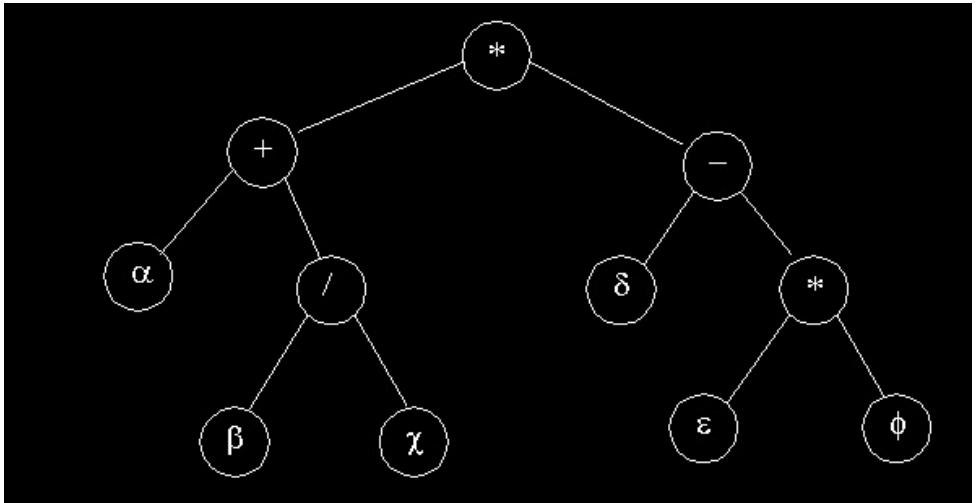
- um ponteiro para o filho localizado à esquerda;
- um ponteiro para o filho localizado à direita;
- um ponteiro para a informação que vamos armazenar.

Pseudo-código:

```
struct NO{
     int info;
     struct NO *esq;
     struct NO *dir;
};
```

# Percurso em árvores binárias

- O percurso em árvores binárias é equivalente ao caminhamento executado em listas:
  - partimos de um nodo inicial (raiz) e visitamos todos os demais nodos em uma ordem previamente especificada.



Representação da expressão (α+β/χ)*(δ−ε*ϕ) como árvore

# Percurso em árvores binárias

Existem três ordens para se percorrer uma árvore binária que são consequência natural da estrutura da árvore:

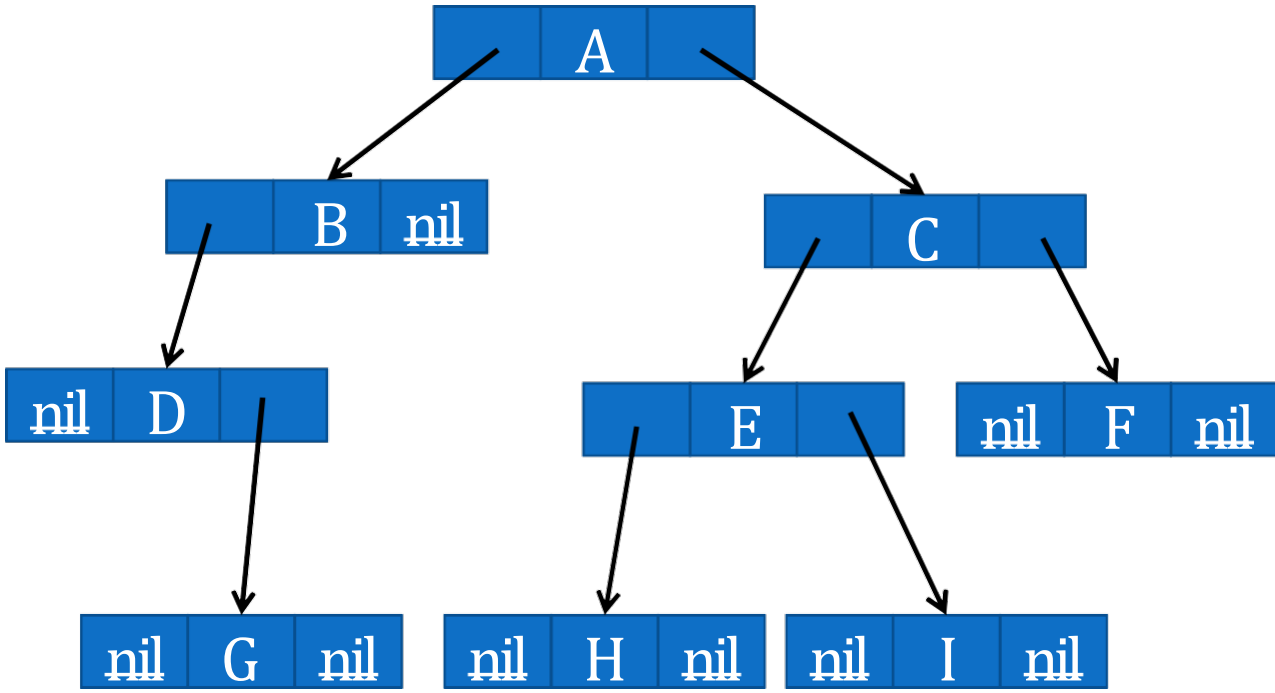
- Preordem(r,e,d) – *Preorder*
- Emordem(e,r,d) – *Inorder*
- Pósordem(e,d,r) – *Postorder*

# Percurso em árvores binárias

- Essas ordens são definidas recursivamente (definição natural para uma árvore) e em função da raiz(r), da subárvore esquerda(e) e da subárvore direita(d):
  - **Preordem(r,e,d)**: visite a raiz ANTES das subárvores
  - **Emordem(e,r,d)**: visite primeiro a subárvore ESQUERDA, depois a RAIZ e depois a subárvore DIREITA
  - **Pósordem(e,d,r)**: visite a raiz DEPOIS das subárvores
  - As subárvores são SEMPRE visitadas da esquerda para a direita

# Percurso em árvores binárias

- Percurso em em pré-ordem (pré-fixado)
  - Visitar a raiz;
  - Percorrer sua subárvore esquerda, em pré-ordem (VLR);
  - Percorrer sua subárvore direita, em pré-ordem (VLR).

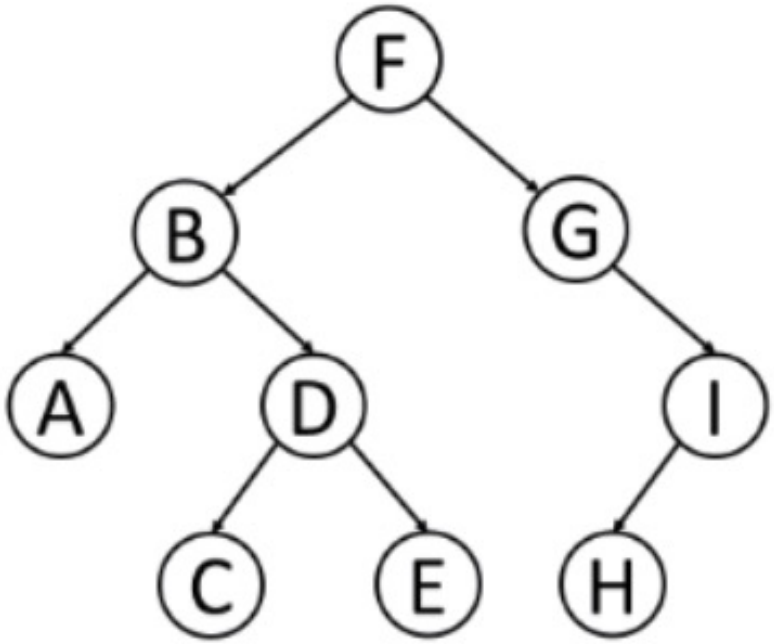



- Exemplo: ABDGCEHIF:

# Percurso em Preordem

```
FUNÇÃO Preordem(tNodo *raiz)
        início
                se raiz != NULO então
                imprime(raiz->info);
                Preordem(raiz->filhoEsquerda);
                Preordem(raiz->filhoDireita);
                fim se
        fim
```

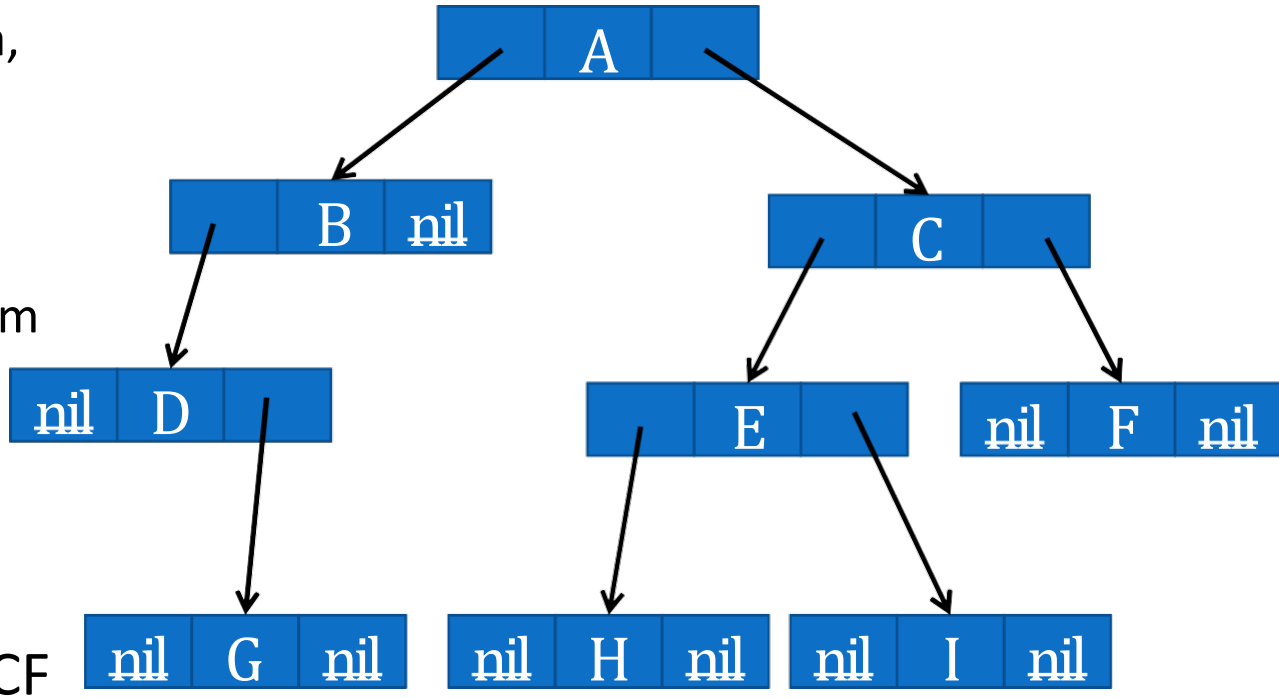
# Percurso em Preordem




Preorder:

# Percurso em árvores binárias

- Percurso em em-ordem ou central
  - Percorrer sua subárvore esquerda, em in-ordem;
  - Visitar a raiz;
  - Percorrer sua subárvore direita, em in-ordem.

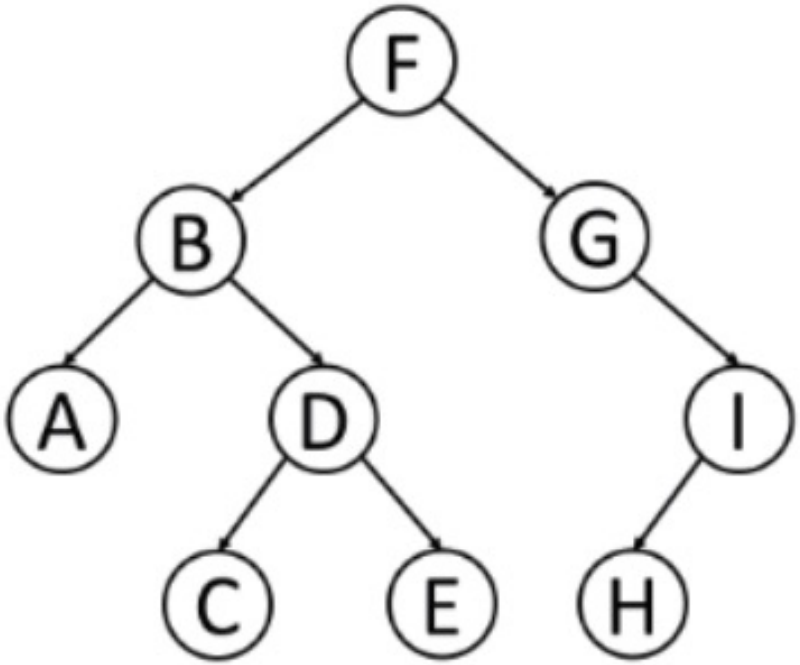



- Exemplo: DGBAHEICF

# Percurso Emordem

```
FUNÇÃO Emordem(tNodo *raiz)
        início
                se raiz != NULO então
                        Emordem(raiz->filhoEsquerda);
                        imprime(raiz->info);
                        Emordem(raiz->filhoDireita);
                fim se
        fim
```

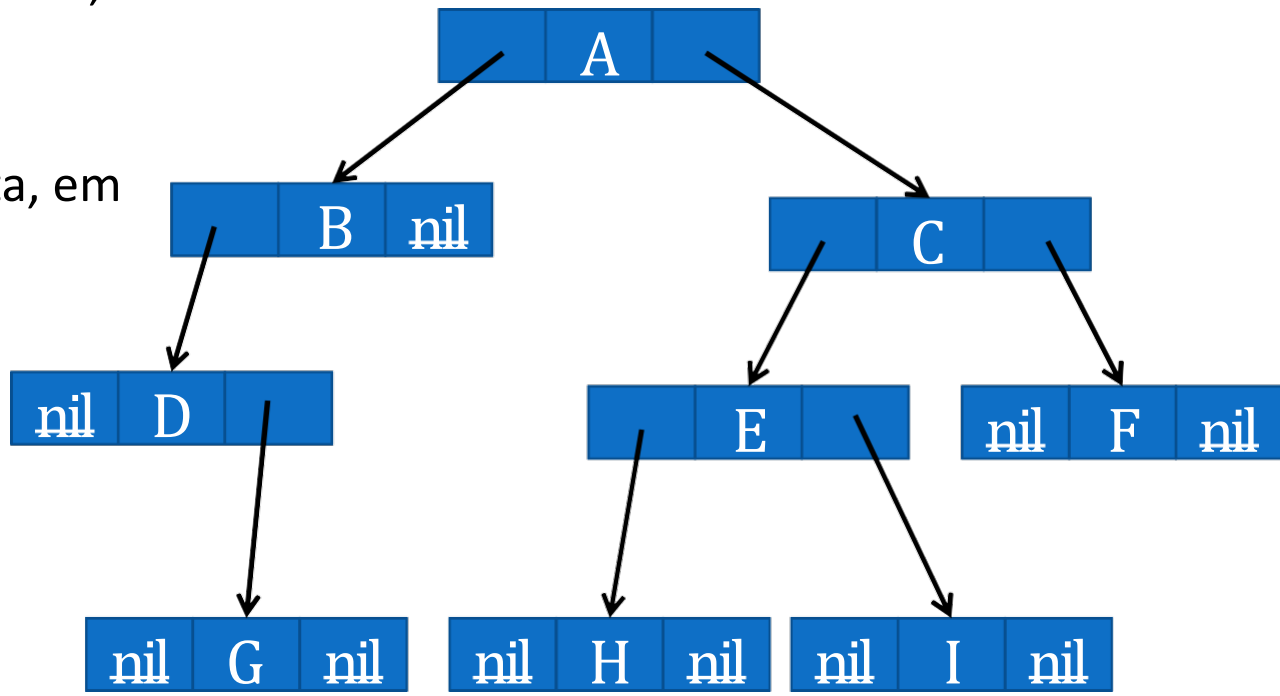
# Percurso Emordem




Inorder:

# Percurso em árvores binárias

- Percurso em pós-ordem
  - Percorrer sua subárvore esquerda, em pós-ordem;
  - Percorrer sua subárvore direita, em pós-ordem;
  - Visitar a raiz.

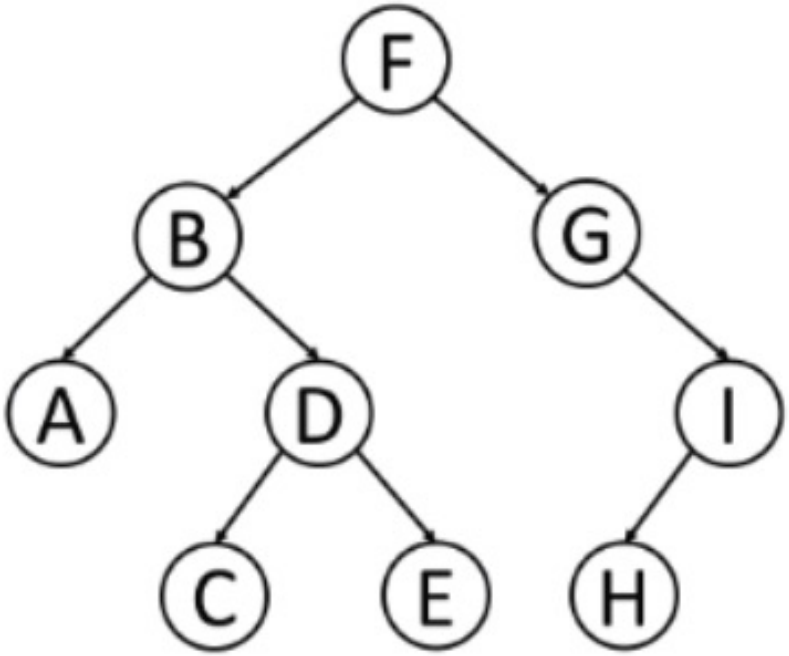



- Exemplo: GDBHIEFCA

# Percurso em Pósordem

```
FUNÇÃO Pósordem(tNodo *raiz)
        início
                se raiz != NULO então
                        Pósordem(raiz->filhoEsquerda);
                        Pósordem(raiz->filhoDireita);
                        imprime(raiz->info);
                fim se
        fim
```

# Percurso em Pósordem




Postorder:

# Referências Oficiais


- BOAVENTURA NETTO, Paulo Oswaldo. **Grafos: teoria, modelos, algoritmos**. 2011.

- CORMEM, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. **Algoritmos: teoria e prática**. 3. ed. São Paulo, SP: Campus, 2012.

- PEREIRA, Silvio L. **Estrutura de Dados Fundamentais: Conceitos e Aplicações**. 12. ed. São Paulo, SP: Érica, 2010.

# Referências Oficiais


- BOAVENTURA NETTO, Paulo O. JURKIEWICS, Samuel. **Grafos: Introdução e prática**. 5. ed. São Paulo, SP: Blucher, 2012.
- FORBELLONE, André L. V.; EBERSPASHER, Henri F. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3. ed. São Paulo, SP: Prentice Hall, 2005.
- GOLDBARG, Marco; GOLDBARG, Elizabeth. **Grafos: Conceitos, Algoritmos e Aplicações**. 1. ed. São Paulo, SP: Campus, 2012.
- SZWARCFITER, Jayme L.; MARKENSON, Lilian. **Estruturas de dados e seus algoritmos**. 3. ed. Rio de Janeiro, RJ: LTC, 2010.
- TOSCANI, Laira V.; VELOSO, Paulo A. S. **Complexidade de algoritmos :análise, projeto e métodos**. 3. ed. Porto Alegre, RS: Bookman, 2012.