Estrutura de Dados

Árvores Binárias

Árvores Binárias

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

i. Se T = ϕ, a árvore é dita vazia, ou

ii. T contém um nó especial, chamado raiz de T, e os demais nós podem ser subdivididos em

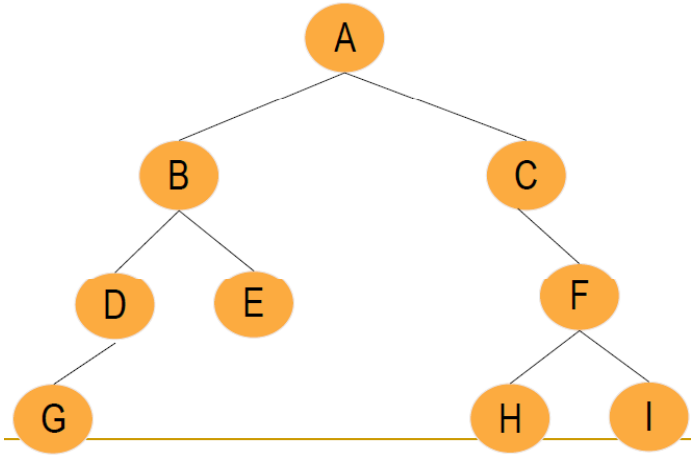
- dois sub-conjuntos distintos TEe TD, os quais também são árvores binárias.
- T_E e T_D são denominados sub-árvore esquerda e sub-árvore direita de T, respectivamente

Árvores Binárias

- 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 $\mathbf{T_{Er}}$ **e** $\mathbf{T_{Dr}}$ são as sub-árvores esquerda e direita de T, respectivamente

Árvore Binária é uma árvore ordenada de grau 2.

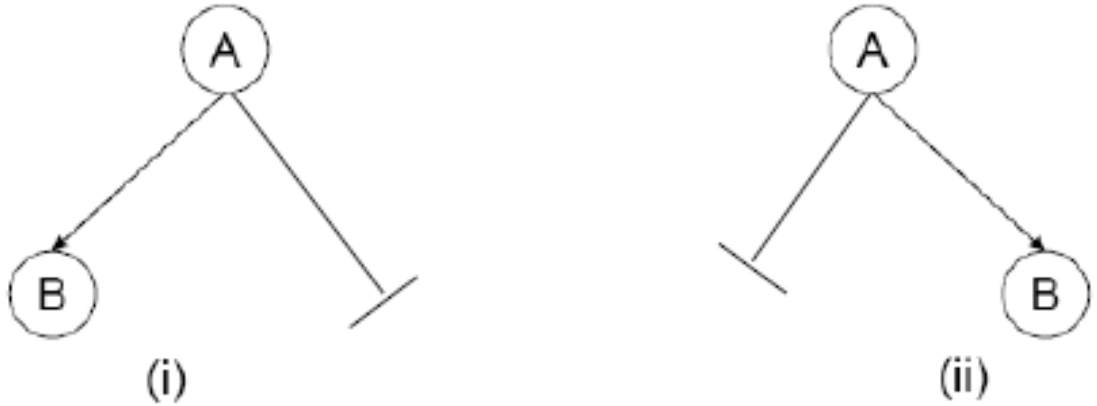


Exemplo – Árvore Binária

- As duas AB seguintes são distintas

i. A primeira tem subárvore direita vazia

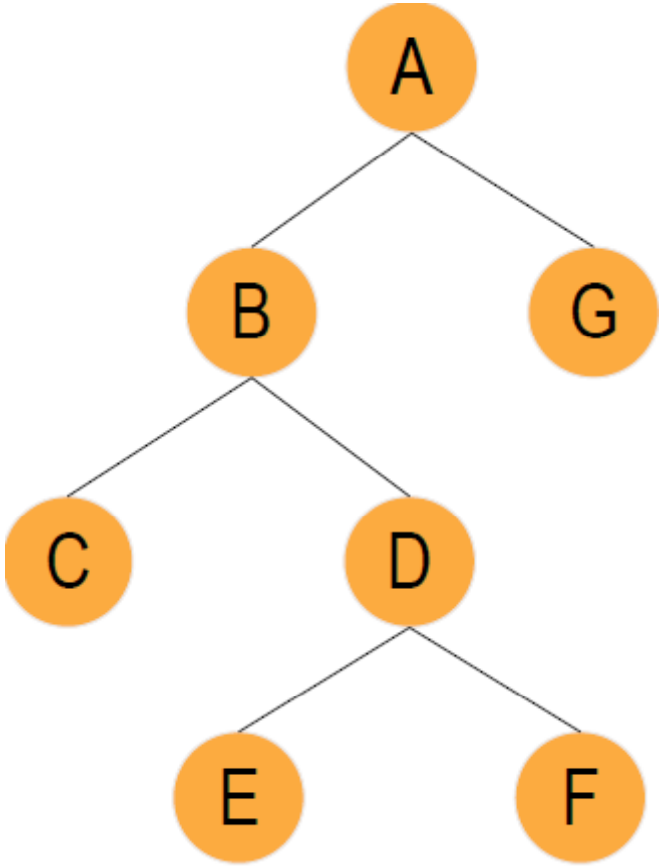
ii. A segunda tem subárvore esquerda vazia



Árvore Estritamente Binária

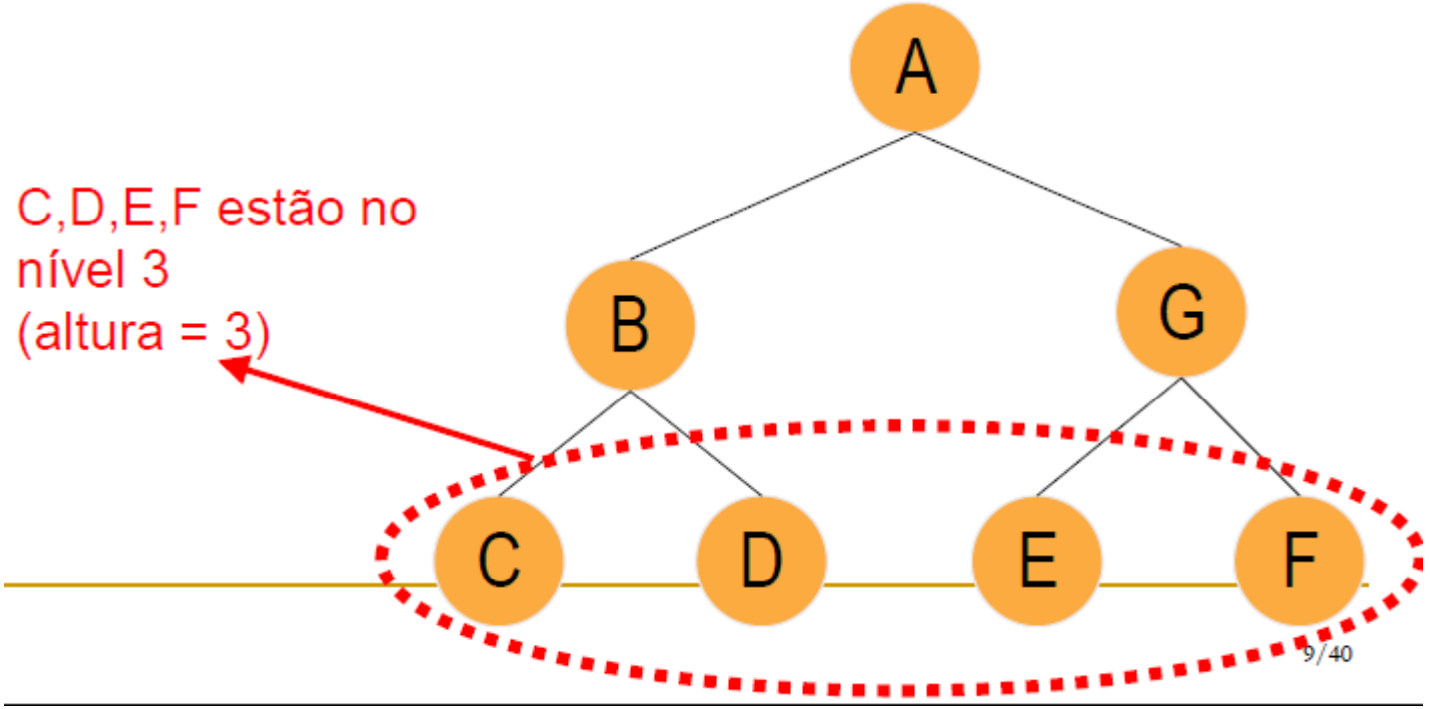
- Uma **Árvore Estritamente Binária** tem nós que têm ou 0 (nenhum) ou dois filhos

- Nós internos (não folhas) sempre têm 2 filhos



Árvore Binária Completa (ABC)

- É estritamente binária, de nível d; e
- Todos os seus nós-folha estão no mesmo nível (d)



- Dada uma ABC e sua altura, pode-se calcular o número total de nós na árvore

- Por exemplo, uma ABC com altura 3 tem 7 nós
  - Nível 1: => 1 nó
  - Nível 2: => 2 nós
  - Nível 3: => 4 nós
  - No. Total de nós = 1 + 2 + 4 = 7

- Verifique que: se uma ABC tem altura h, então o número de nós da árvore é dado por:

$$\mathbf{N = 2^h - 1}$$

Inversamente:

$N = 2^h - 1$

$2^h = N + 1$

$\log_2(2^h) = \log_2(N+1)$

$h = \log_2(N+1)$

Árvore Binária Quase Completa
Árvore Binária Balanceada

ÁRVORE BINÁRIA QUASE COMPLETA

- Se a altura da árvore é **d,** cada nó folha está no nível **d** ou no nível **d-1.**
- Em outras palavras, a diferença de altura entre as sub-árvores de qualquer nó é no máximo 1.

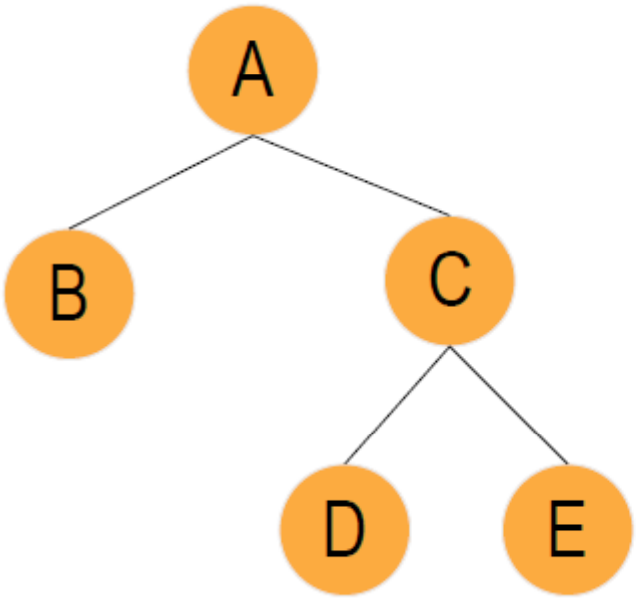
ÁRVORE BINÁRIA BALANCEADA

- Para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo 1

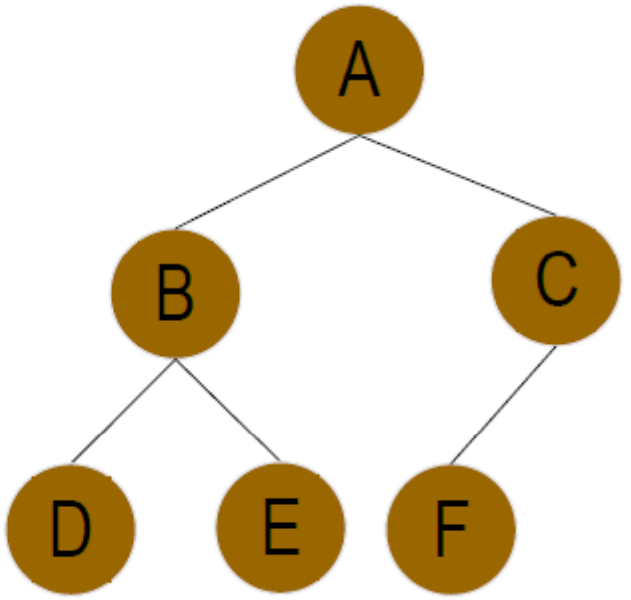
Árvore Binária Perfeitamente Balanceada

- Para cada nó, o número de nós de suas sub-árvores esquerda e direita difere em, no máximo 1
- Toda AB Perfeitamente Balanceada é Balanceada, mas o inverso não é necessariamente verdade.
- Uma AB com N nós tem altura mínima se e só se for Balanceada.
- Se uma AB for Perfeitamente Balanceada então ela tem altura mínima.

Exemplo



Árvore Balanceada

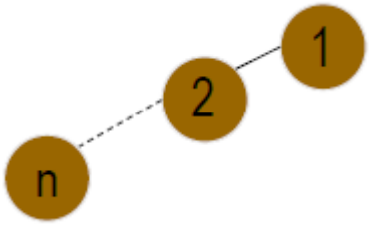


Árvore Perfeitamente Balanceada

6 nós: h_{min}= 3

Questões

- Qual a altura máximade uma AB com **n** nós?
  - Resposta: n
  - Árvore degenerada ≈ Lista



- Qual a altura mínima de uma AB c/ **n** nós?
  - Resposta: a mesma de uma AB Perfeitamente Balanceada com **N** nós

- Uma árvore binária completa é uma árvore estritamente binária?

- Uma árvore estritamente binária é uma árvore binária completa?

Implicações dos conceitos de balanceada e perfeitamente balanceada

- *As árvores balanceadas e perfeitamente balanceadas permitem que a recuperação de informação possa ser realizada de maneira a garantir a altura da ordem de $O(\log_2 n)$ e assim a eficiência em termos de tempo.*

Árvores Binárias

REPRESENTAÇÃO/IMPLEMENTAÇÃO

Representação/Implementação

Estática Sequencial

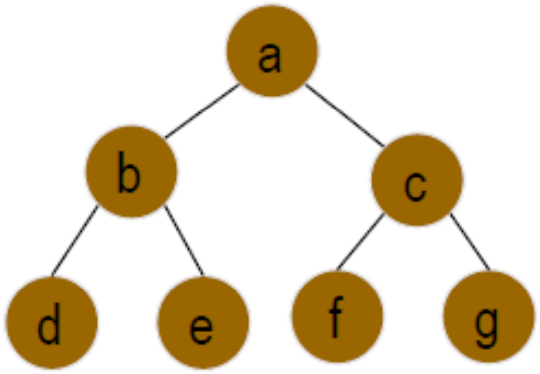
- Simples,
- Melhor solução quando é completa (cheia). Se não for completa (cheia), um campo “usado” é necessário para indicar qual posição tem ou não elemento.

Encadeada Dinâmica

- O limite para o número de nós é a memória
- Preferida no caso geral

Implementação Estática

- Armazenar os nós, por nível, em um *vetor*



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

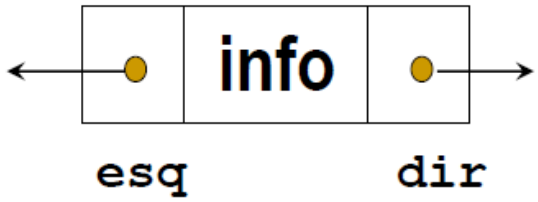
- Se um nó está na posição i, seus filhos diretos estão nas posições 2i e 2i+1

Vantagem: espaço só p/ armazenar conteúdo; ligações implícitas

Desvantagem: espaços vagos se árvore não é completa por níveis, ou se sofrer eliminação

Implementação de AB (dinâmica)

- Para qualquer árvore, cada nó é do tipo




```
typedef int tipo_elem;

typedef struct No{
        tipo_elem info;
        struct No* esq;   // filho esquerdo
        struct No* dir;   // filho direito
}No;

typedef struct{
        No *raiz;
}tree;
```

TAD Árvore Binária

Operações do TAD

1. Criação de uma árvore
    Dois casos: seguindo a definição recursiva
        1. Criar uma árvore vazia
        2. Criar uma árvore com 2 subárvores(e,d): **faz o papel de inserir nó**
2. Verificar se árvore está vazia, mostrar o conteúdo do nó
3. Buscar o pai de um nó, buscar um nó
4. Retornar: nro de nós, nro de folhas, altura
5. Destruir árvore
6. Remover um nó
    Dois casos:
        1. O nó não tem filhos
        2. O nó tem um filho a esq, ou a dir ou os dois
7. Verificar nível de um nó
8. Verificar se é balanceada, se é perfeitamente balanceada

Outras?

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, atualizar um campo de cada nó, buscar um item, destruir uma árvore, etc.

AB –Percursos em Árvores

3 percursos básicos para AB's:

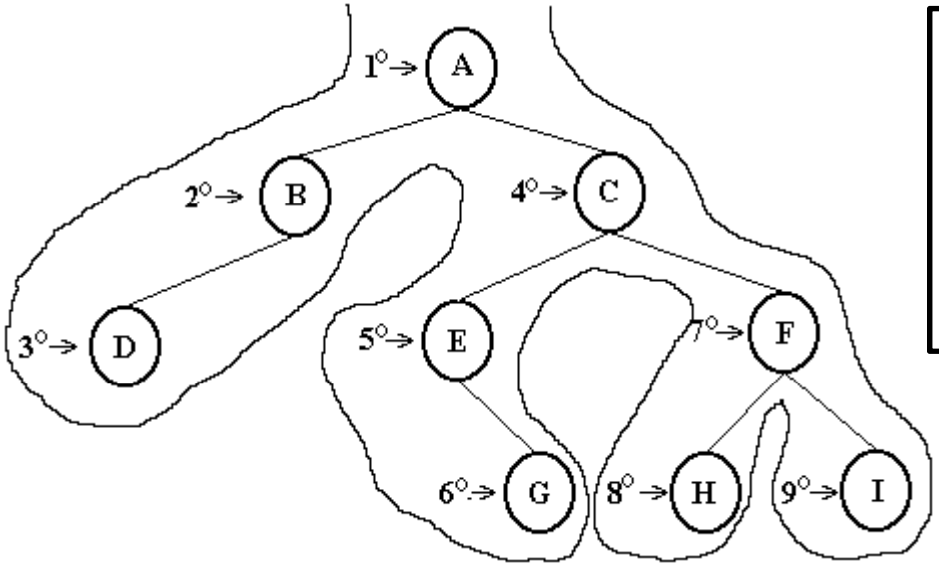
- pré-ordem (*Pre-order*)
- in-ordem (*In-order*)
- pós-ordem (*Post-order*)

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

- "Visitar" um nó pode ser:
  - Mostrar (imprimir) o seu valor;
  - Modificar o valor do nó;
  - Verificar se o valor pertence a árvore (membro)

Pré-ordem

Se árvore vazia; fim
1. visitar o nó raiz
2. percorrer em pré-ordem a subárvore esquerda
3. percorrer em pré-ordem a subárvore direita

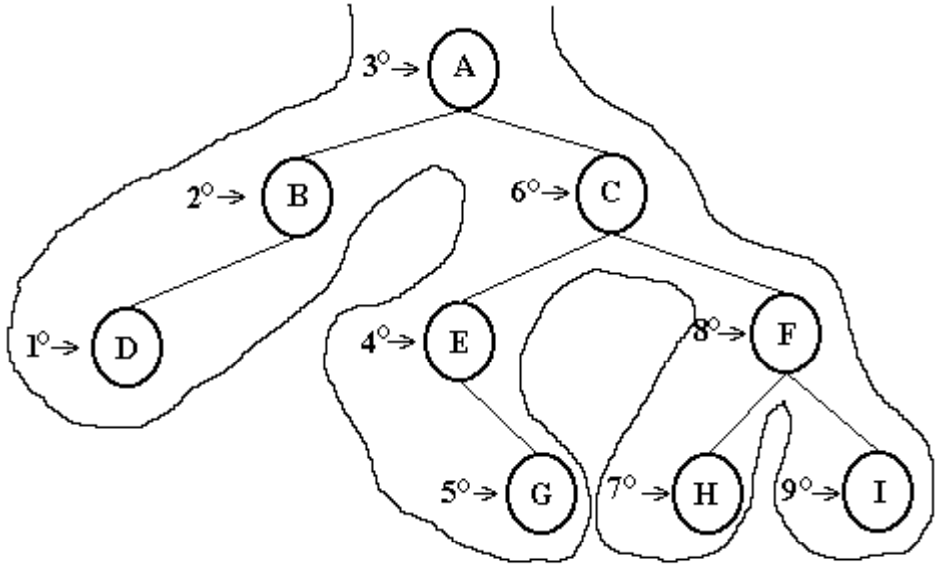



```
void pre_ordem(Arvore *raiz){
   visita(raiz);
   pre_ordem(raiz->esq);
   pre_ordem(raiz->dir);
}
```

Resultado: A B D C E G F H I

Em-Ordem

Se árvore vazia, fim
1. percorrer em in-ordem a subárvore esquerda
2. visitar o nó raiz
3. percorrer em in-ordem a subárvore direita

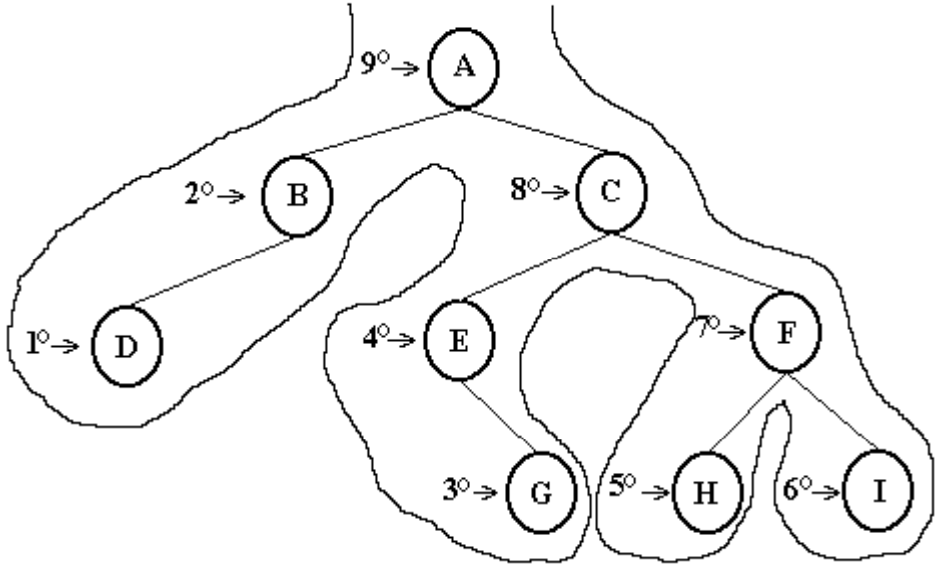



```
void in_ordem(Arvore *raiz){
   if(raiz != NULL){
      in_ordem(raiz->esq);
      visita(raiz);
      in_ordem(raiz->dir);
   }
}
```

Resultado: D B A E G C H F I

Pós-Ordem

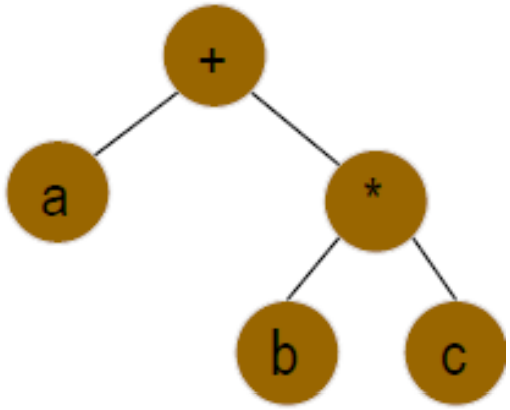
Se árvore vazia, fim
1. percorrer em Pós-Ordem a subárvore esquerda
2. percorrer em Pós-Ordem a subárvore direita
3. visitar o nó raiz




```
void pos_ordem(Arvore *raiz){
   if(raiz != NULL){
      pos_ordem(raiz->esq);
      pos_ordem(raiz->dir);
      visita(raiz);
   }
}
```

Resultado: D B G E H I F C A

Percurso para expressões aritméticas



Pré-ordem: +a*bc

In-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)

Exercícios

1. Fazer função para procurar um nó cujo conteúdo seja “item” e retornar seu endereço. Se não for encontrado, retornar null. Usar o percurso **pré-ordem** para a busca.

2. Fazer unção para calcular o nível de um nó. Dado o valor de um elemento, se ele está na árvore, retorna seu nível, retorna null c.c. OBS.: Nivel da raiz = 1

3. Uma árvore binária completa é uma árvore estritamente binária?

4. Uma árvore estritamente binária é uma árvore binária completa?

5. Escreva um procedimento recursivo que calcula a altura de uma AB.

6. Procedimento recursivo p/ destruir árvore, liberando o espaço alocado (percurso em pós-ordem)