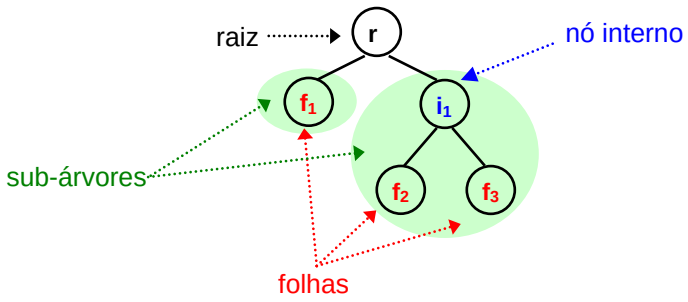
# Introdução

Os Tipos Abstratos de Dados estudados foram Listas Simplesmente e Duplamente Encadeadas, Fila e Pilha, sendo que o que muda nesses Tipos as operações, pois a Estrutura de Dados base para tais tipos são as listas lineares, sejam elas estáticas ou dinâmicas. Embora tais listas apresentem vantagens quanto ao uso, à manipulação e à alocação, ainda possuem problemas:

- **Lista encadeada**
  - Eficiente para inserção e remoção dinâmica de elementos, mas ineficiente para busca;
- **Lista seqüencial** (ordenada)
  - Eficiente para busca, mas ineficiente para inserção e remoção de elementos.

Em busca de contornar essas desvantagens, foi proposto o conceito de **Árvores**, que apresenta solução eficiente para inserção, remoção e busca. Vale destacar que as árvores possuem uma representação não linear.
As árvores são estruturas de dados adequadas para a representação de hierarquias. A forma mais natural para definirmos uma estrutura de árvore é usando recursividade. Uma árvore é composta por um conjunto de nós. Existe um nó **r**, denominado **raiz**, que contém **zero ou mais sub-árvores**, cujas raízes são ligadas diretamente a **r**.
Esses nós **raízes** das sub-árvores são ditos **filhos** do nó pai, **r**. Nós com filhos são comumente chamados de **nós internos** e nós que não têm filhos são chamados de **folhas**, ou **nós externos**. É tradicional desenhar as árvores com a raiz para cima e folhas para baixo, ao contrário do que seria de se esperar. A figura a seguir exemplifica a estrutura de uma árvore.

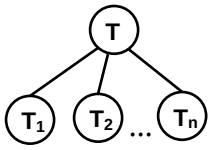



O número de filhos permitido por nó e as informações armazenadas em cada nó diferenciam os diversos tipos de árvores existentes. Serão estudados dois tipos de árvores. Primeiro, examinaremos as árvores binárias, onde cada nó tem, no máximo, dois filhos. Depois examinaremos as chamadas árvores genéricas, onde o número de filhos é indefinido. Estruturas recursivas serão usadas como base para o estudo e a implementação das operações com árvores.

# Árvores

Definição: Árvore T – conjunto finito de elementos, denominados **nós** ou **vértices**, tais que:

- Se **T** = Ø, a árvore é dita vazia;
- Caso contrário:
    i. **T** contém um nó especial, denominado **raiz**;
    ii. os demais nós ou constituem um único conjunto vazio, ou são divididos em ***m≥1*** conjuntos disjuntos não vazios ($\mathbf{T_1}$,$\mathbf{T_2}$,…,$\mathbf{T_n}$), que são, por sua vez, cada qual uma árvore;
    - $\mathbf{T_1}$,$\mathbf{T_2}$,…,$\mathbf{T_n}$ são chamadas sub-árvores de **T**;
    - Um nó sem sub-árvores é denominado **nó-folha**, ou simplesmente, **folha**.




Se um nó **T** é a raiz de uma árvore e um nó $\mathbf{T_1}$ é raiz de uma sub-árvore de **T**, então **T** é o **PAI** de $\mathbf{T_1}$ e $\mathbf{T_1}$ é o **FILHO** de **T**.

## *Conceitos*

**NÍVEL:** o nível de um nó **T** é definido como:

- O nível de um nó raiz é **0;**
- O nível de um nó não raiz é dado por (nível de seu nó **PAI** + 1).

**GRAU:** o grau de um nó **T** de uma árvore é igual ao número de filhos do nó **T**;

**GRAU DA ÁRVORE:** o grau de uma árvore **T** é o grau máximo entre os graus de todos os seus nós;

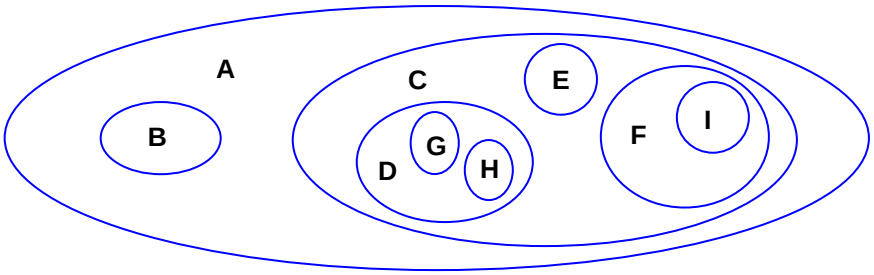
**ALTURA :** A altura de um nó **v** em uma árvore binária é a distância entre **v** e o seu descendente mais afastado. Mas precisamente, a altura de **v** é o número de passos do mais longo caminho que leva de **v** até uma folha. Os nós **folha** sempre têm altura igual a **0**;

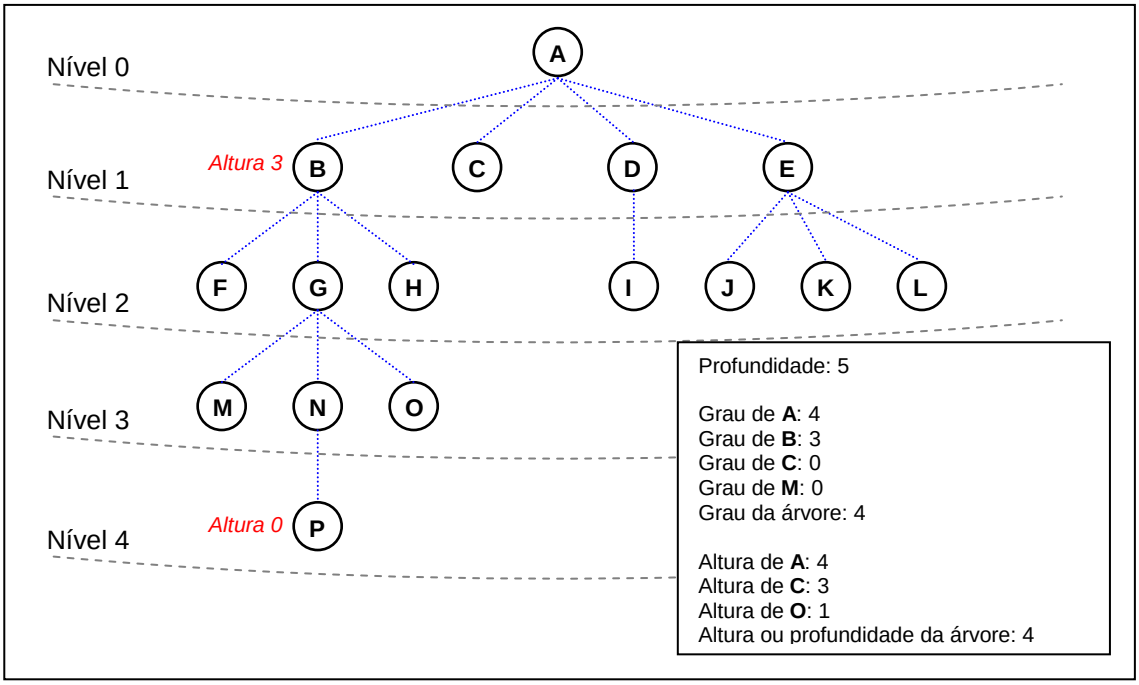
**ALTURA ou PROFUNDIDADE DE UMA ÁRVORE:** A altura de uma árvore **T** é dada pela altura da raiz da árvore.

- Denota-se a altura de uma árvore com raiz dada pelo nó **T** por **h(T)**, e a altura de uma sub-árvore com raiz $\mathbf{T_1}$ por $\mathbf{h(T_1)}$.

- Outras formas de representação:

- Representação por parênteses aninhados
    ( A (B) ( C (D (G) (H)) (E) (F (I)) ) )  *ou seja, uma lista generalizada!!*
- Representação por Diagramas de Venn






Além disso, mediante a representação hierárquica, pode-se definir que:
O nó **A** é **ANCESTRAL** dos nós **B, C, D, E ... P**.

- Um 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;**

O nó **B** é irmão de **C**, **D** e **E**.

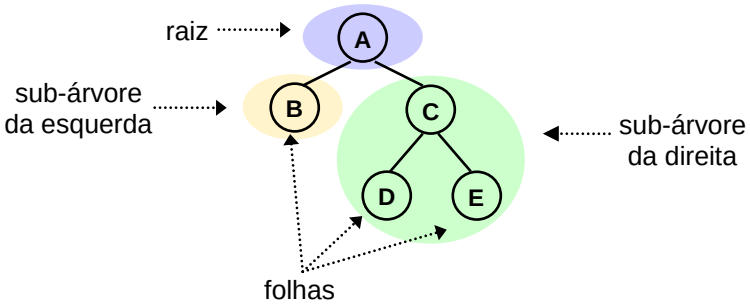
- Dois nós são **IRMÃOS** se forem filhos do mesmo pai.

Por fim, uma **FLORESTA** é o conjunto de **Árvores disjuntas.**

## *Árvores Binárias*

Uma **árvore binária** é um conjunto finito de elementos que ou é vazio ou é dividido em três subconjuntos disjuntos:

- A **raiz** da árvore;
- Uma **árvore binária** chamada de **sub-árvore da esquerda**;
- Uma **árvore binária** chamada de **sub-árvore da direita**.



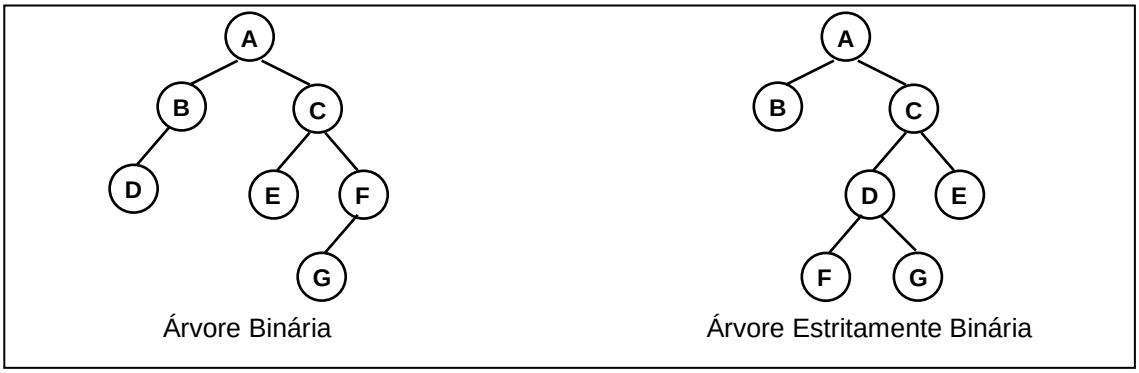

☞ Note que uma árvore binária não é um caso particular de árvore, é um conceito diferente. Pode-se dizer que toda árvore binária é uma árvore, mas nem toda árvore é uma árvore binária.

Uma árvore binária possui algumas propriedades:
- O número máximo de nós no nível ***i*** de uma árvore binária é $2^i$ (Como uma árvore binária pode conter no máximo um nó no nível 0 (raiz), ela poderá conter no máximo $2^i$ nós no nível ***i***).

- O número máximo de nós em uma árvore binária de altura ***k*** é $2^{k+1}-1$ (Como uma árvore binária

## Árvores Estritamente Binárias

Se todo nó que não é folha numa árvore binária tiver sub-árvores esquerda e direita não vazias. Uma árvore estritamente binária com ***n*** folhas tem ***2n-1*** nós. Nós interiores (não folhas) possuem sempre 2 filhos.




## Árvore Binária Completa

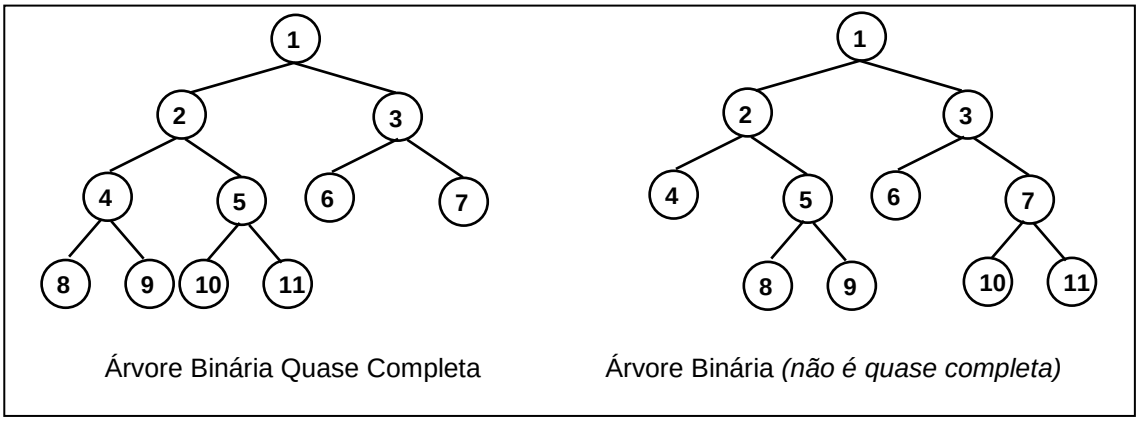
Uma Árvore Binária Completa de Profundidade ***k*** é uma ***árvore estritamente binária*** nas quais todas as folhas estão no nível ***k***.
Se ***k*** é a profundidade da árvore, o número total de nós é $2^{k+1}-1$. A árvore contém $2^{k}$ folhas e, portanto, $2^{k}-1$ nós não folhas.


☞ Embora uma árvore binária completa possua vários nós (o máximo para cada profundidade), a distância da raiz a uma folha qualquer é relativamente pequena.

## Árvore Binária Quase Completa

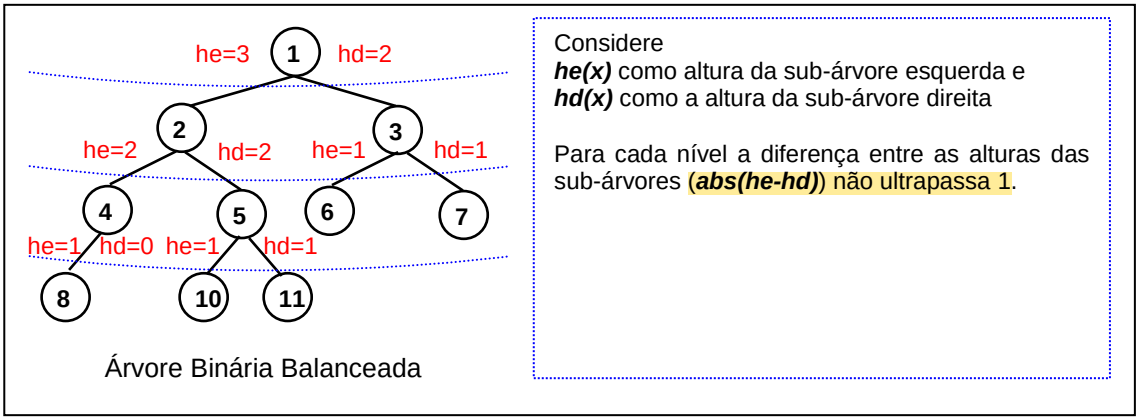
É uma Árvore Binária com profundidade ***k*** onde:

- Cada nó folha na árvore está no nível ***k*** ou no nível ***k-1*** (até o nível ***k-1*** ela é completa);
- Para qualquer nó ***n*** da árvore com um descendente direito no nível ***k***, também deve existir o descendente esquerdo correspondente no nível ***k.***
- Uma árvore binária quase completa pode ter seus nós numerados começando da raiz, de cima para baixo e da esquerda para a direita sem que haja a ausência de nós.




## Árvores Balanceadas

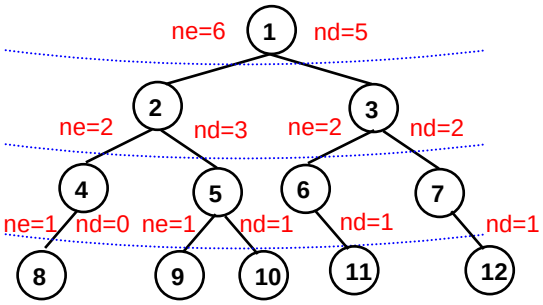
Uma **árvore balanceada** é aquela onde para cada nó, as alturas de suas duas sub-árvores diferem de, no máximo, 1.




## Árvore Binária Perfeitamente Balanceada

O número de nós de suas sub-árvores esquerda e direita difere em, no máximo, 1.

☞ *Toda Árvore Binária Perfeitamente Balanceada é Balanceada, mas o inverso não é necessariamente verdade.*




Árvore Binária Perfeitamente Balanceada

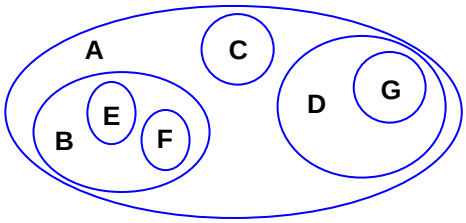
Considere
***ne(x)*** como o número de nós da sub-árvore esquerda, e
***nd(x)*** como o número de nós da sub-árvore direita.

Para cada nível a diferença entre as quantidades de nós das sub-árvores (***abs(ne-nd)***) não ultrapassa 1.

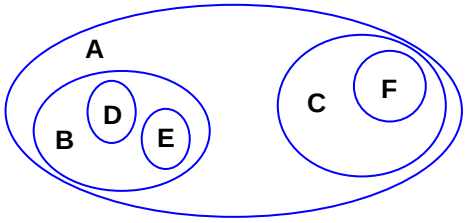
## *Exercícios*

1- Dado o seguinte Diagrama de Venn, construa a respectiva representação em forma de lista (com parênteses) e árvore.

a)




b)




Para cada árvore responda:

a. Descreva para cada árvore:
   i. Grau dos nós;
   ii. Grau da árvore;
   iii. Folhas da árvore;
   iv. Raiz da árvore;
   v. Nós em cada nível;
   vi. Altura da árvore;
b. É uma árvore binária?
c. Se for binária:
   i. ela é estritamente binária?
   ii. ela é completa ou quase completa?
   iii. ela está balanceada? Ela é perfeitamente balanceada?

2- Qual a altura máxima de uma Árvore Binária com ***n*** nós?

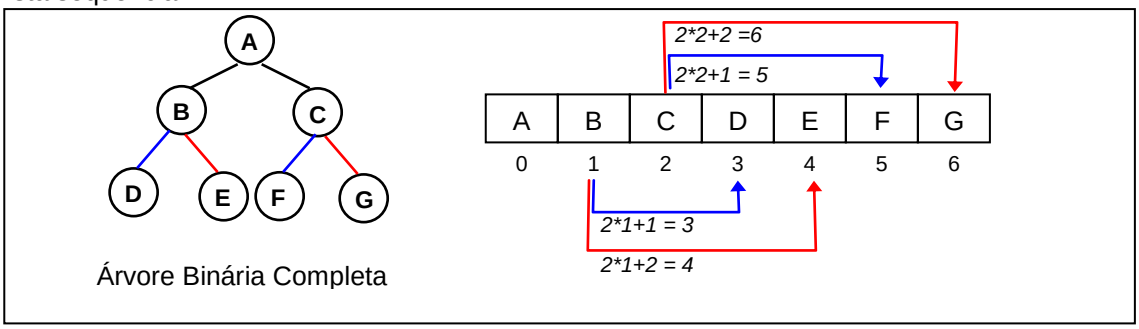
3- Qual a altura mínima de uma Árvore Binária com ***n*** nós?

*Use exemplos para facilitar a resolução desses exercícios?*

# *Implementação*

## Usando Lista Seqüencial

Possível implementação de uma Árvore Binária Completa usando alocação estática em uma lista seqüencial.

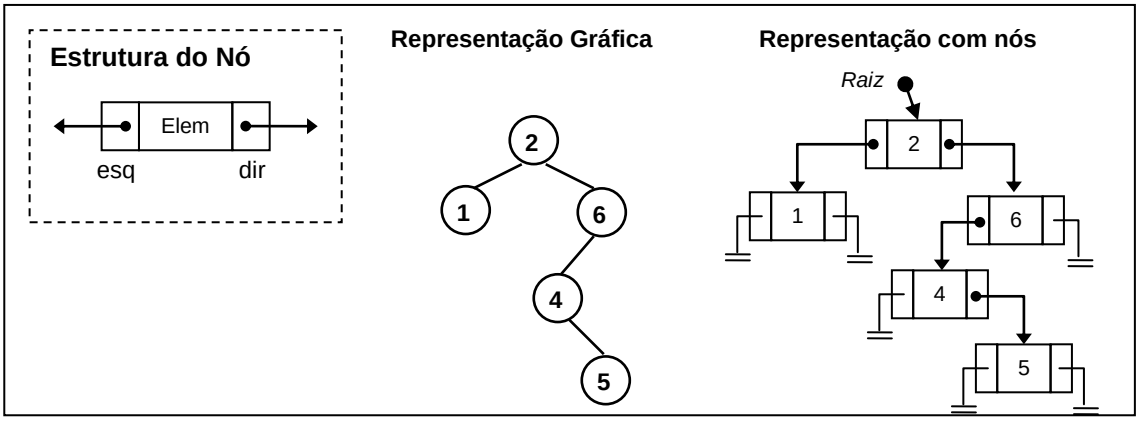



- Armazenar os nós, por nível, num vetor;
- Assim: se um nó ocupa a posição ***i***, então seus filhos diretos estão nas posições ***2i+1*** e ***2i+2***

***Vantagem***: espaço só para armazenar conteúdo; ligações implícitas
***Desvantagens***: espaços vazios se a árvore não for completa por níveis, ou se sofrer eliminação.

## Usando Alocação Dinâmica

Para qualquer árvore binária os nós serão do tipo:




### *Definição do Nó*

```
typedef struct No pno;
struct No {
   // conteúdo a ser armazenado no nó
   int elem;
   // auto-referências para os nós da esquerda e direita da árvore
   pno *esq;
   pno *dir;
}
```

## Operações Associadas ao Tipo Abstrato – Árvore Binária

- Definir uma árvore vazia
- Criar um nó raiz
- Verificar se árvore vazia ou não
- Criar um filho à direita de um dado nó
- Criar um filho à esquerda de um dado nó
- Verificar qual o nível de um dado nó
- Retornar o pai de um dado nó

### *Exercícios*

1- Implementar as operações associadas ao TAD árvore binária dinâmica.

## *Percursos*

Uma vez que determinadas informações são armazenadas em uma estrutura como uma árvore, é preciso de alguma maneira acessar as informações lá contidas. Se fosse uma lista linear, bastaria percorrer essa lista do início ao fim um após o outro de forma seqüencial. Entretanto, uma árvore é uma estrutura não linear e dessa maneira, por onde iniciar o percurso? Onde encerrar? *É necessário ressaltar que em uma árvore binária não há nenhuma relação de ordem dentre os valores!*

Pode-se dizer que o objetivo, então, é percorrer uma Árvore Binária 'visitando' cada nó uma única vez. Um percurso gera uma seqüência linear de nós, e dessa forma é possível falar de nó predecessor ou sucessor de um nó, segundo um dado percurso. Em outras palavras, obter uma seqüência a partir de uma estrutura não linear!

Não existe um percurso único para árvores (binárias ou não): diferentes percursos podem ser realizados, dependendo da aplicação. A utilização é imprimir uma árvore, remover um item, buscar por um item, entre outras.

Há **três** percursos básicos para Árvores Binárias:

- pré-ordem (*Pre-order*) ou profundidade;
- em-ordem (*In-order*) ou ordem simétrica;
- pós-ordem (*Post-order*).

A diferença entre esses percursos é, basicamente, a ordem em que os nós são "**visitados**". Visitar um nó pode ser:

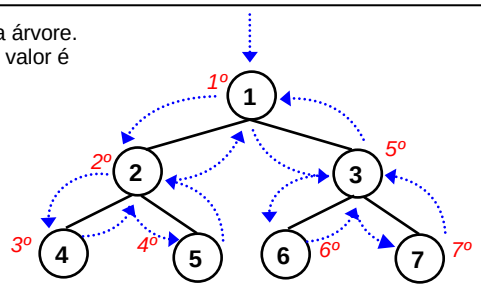
- Mostrar o seu valor;
- Modificar o valor do nó;
- ...

## Percurso Pré-Ordem ou Profundidade

O percurso pré-ordem consiste nos seguintes passos:

1. Mostra o valor do nó;
2. Visita o nó esquerdo;
3. Visita o nó direito;

Inicia o percurso pela raiz da árvore. Assim que o nó é visitado, o valor é mostrado (1ª passagem).




Resultado do Percurso: 1,2,4,5,3,6,7

### *Exercícios*

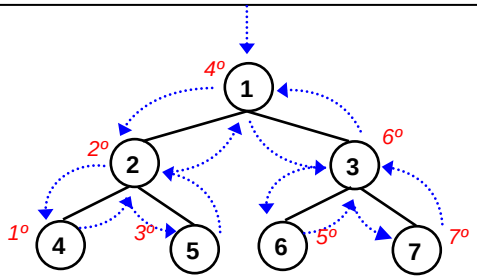
1- Implementar o percurso em Pré-ordem.

## Percurso Em-Ordem ou Ordem Simétrica

O percurso em-ordem consiste nos seguintes passos:

1. Visita o nó esquerdo;
2. Mostra o valor do nó;
3. Visita o nó direito;

Inicia o percurso pela raiz da árvore.
Caminha inicialmente pelos nós da esquerda, só exibindo os valores quando todos à esquerda já tiverem sido visitados (2ª passagem).




Resultado do Percurso: 4,2,5,1,6,3,7

### *Exercícios*

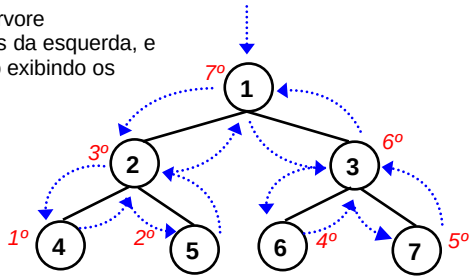
1- Implementar o percurso em Em-ordem.

## Percurso Pós-Ordem

O percurso pós-ordem consiste nos seguintes passos:

1. Visita o nó esquerdo;
2. Visita o nó direito;
3. Mostra o valor do nó;

Inicia o percurso pela raiz da árvore
Caminha inicialmente pelos nós da esquerda, e em seguida pelos da direita, só exibindo os valores quando todos os nós descendentes já tiverem sido visitados (3ª passagem).



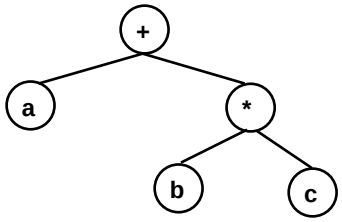

Resultado do Percurso: 4,5,2,6,7,3,1

### *Exercícios*

1- Implementar o percurso em Pós-ordem.

Esses percursos são implementados recursivamente, pois permite um algoritmo simples, versátil e que não compromete o desempenho da máquina, pois em geral tais estruturas, quando balanceadas, se utilizam de uma pilha de recursão correspondente à altura da árvore. Em algoritmos iterativos é preciso o uso de uma pilha, ou então uma referência em cada nó ao nó Pai respectivo.

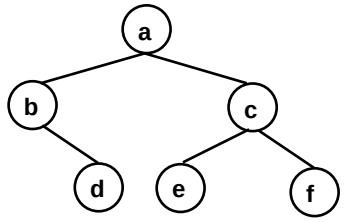
## Percurso para Expressões Aritméticas




***Pré-ordem*** **=** +a*bc
***Em-ordem*** **=** a+(b*c)
***Pós-ordem*** **=** abc*+

## *Exercícios*

1- Implemente um método que mostre a Árvore Binária na representação de parênteses aninhados.




Representação da árvore binária ao lado na forma de parênteses aninhados:

**( a ( b ( ) ( d ( ) ( ) ) ) ( c ( e ( ) ( ) ) ( f ( ) ( ) ) ) )**

Dica: Veja os métodos para percurso!!!

2- Escrever o algoritmo de percurso Pré-Ordem utilizando alocação dinâmica, mas sem utilizar o método recursivo. Utilizar uma pilha para saber o endereço da sub-árvore que resta à direita.
   - processar raiz A
   - guardar A para poder acessar C depois
   - passa à B e processa essa sub-árvore
   - idem para D
   - retorna B (topo da pilha) para acessar D que é a sub-árvore esquerda

3- Escrever um método recursivo que calcule a **altura** de uma dada árvore binária. A altura de uma árvore é igual ao nível máximo de seus nós.

4- Escreva um método recursivo que mostre o nível em que se encontra um determinado nó da árvore.

5- Escreva um método que verifique se a árvore está balanceada.

6- Adapte o método anterior para que o mesmo verifique se a árvore está perfeitamente balanceada.

7- Escreva um método que verifique se a árvore é estritamente binária.