Estrutura de Dados

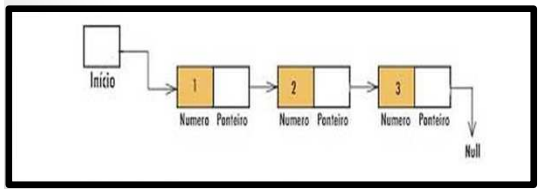
Árvores - Introdução

Profa. Dra. Lúcia Guimarães

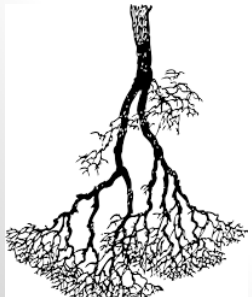
- **Esta aula foi baseada na Apostila de Estrutura de Dados – PUC-RIO**

  Profs. Waldemar Celes e José Lucas Rangel

- **Problema:**
  - **Listas Lineares**
    - **Lista Encadeada**
      - Eficiente para inserção e remoção dinâmica de elementos, mas
      - ineficiente para busca
    - **Lista Sequencial (ordenada)**
      - Eficiente para busca,
      - mas ineficiente para inserção e remoção de elementos

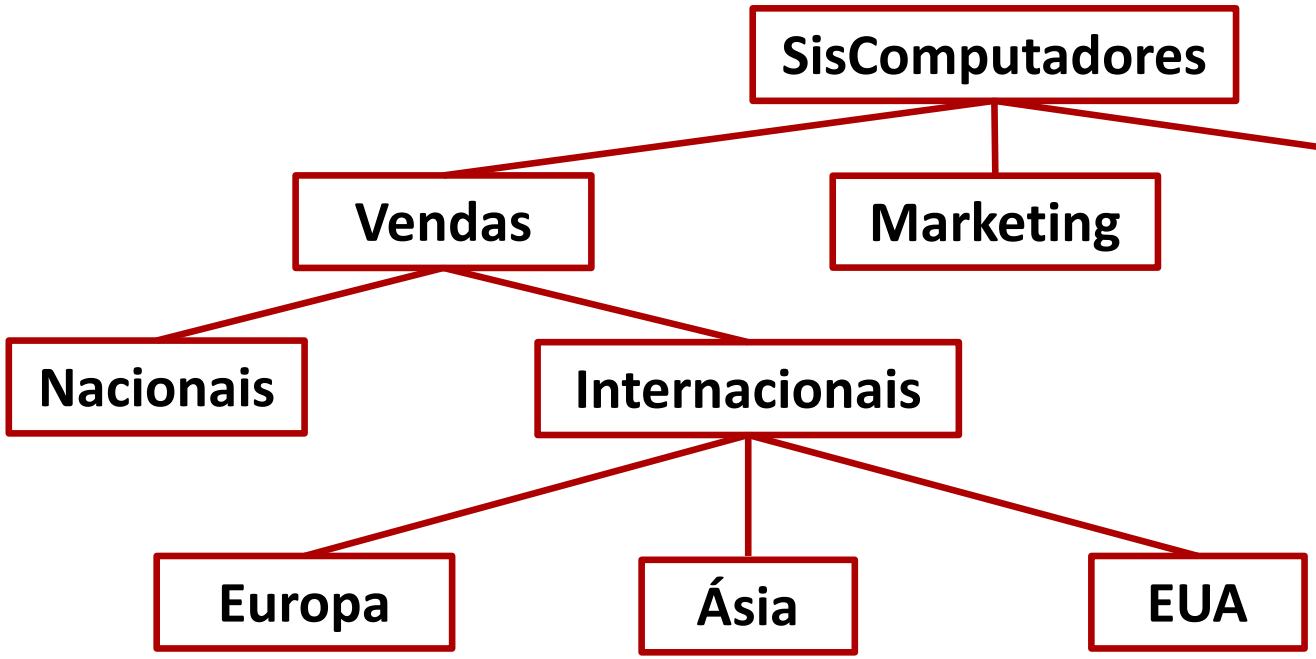


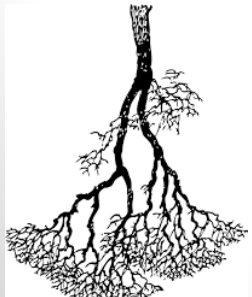
- **Possível Solução:**
  - **Árvores**
    - Eficientes para inserção, remoção e busca
    - Representação não linear...
    - **Definição:**
      - Computacionalmente, **árvore é um modelo abstrato de uma estrutura hierárquica**
      - É uma estrutura não-linear constituída de nós com relações de parentesco (pai-filho)




ESTRUTURA
DE
DADOS

- **Exemplo:**

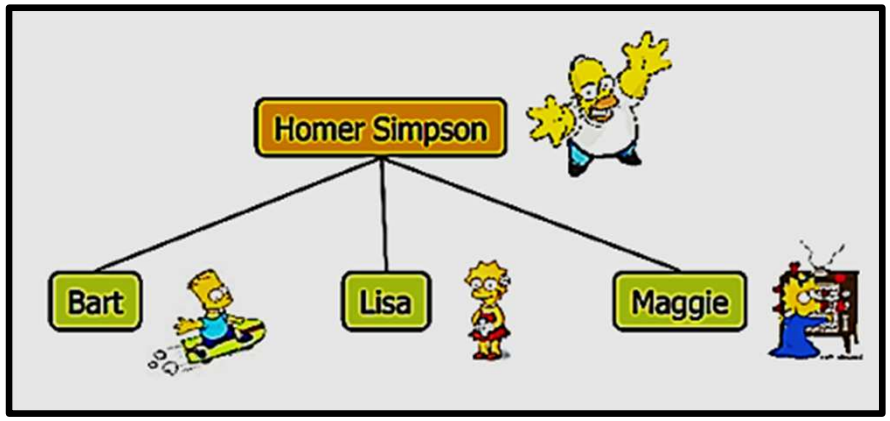

SisComputadores
Vendas
Marketing
Produção
Nacionais
Internacionais
Europa
Ásia
EUA

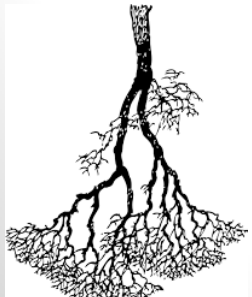



Árvores

- **Introdução**

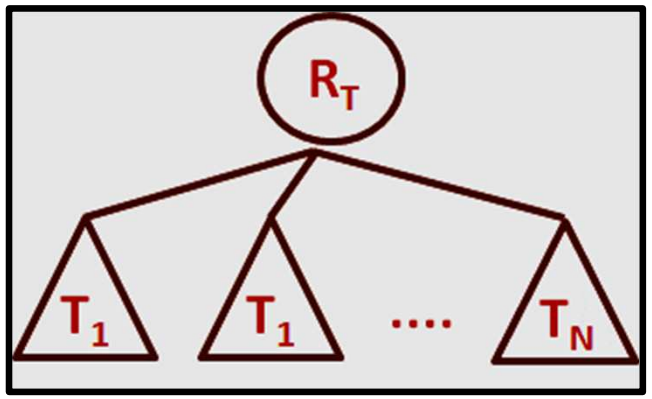
  - **Aplicações:**
    - Sistemas Operacionais – Arquivos
    - Linguagem Orientada Objeto
    - Etc....
  - São adequadas para **representar estruturas hierárquicas não lineares**, **como relações de descendência - pai, filhos, irmãos, etc.**



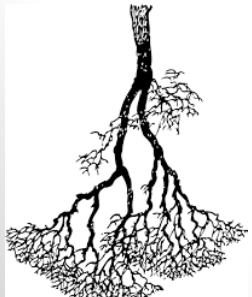


Arvores - Introdução

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

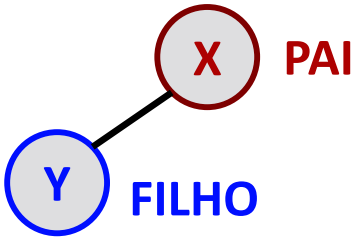


    - **Se T = ∅**
      - A árvore está vazia
    - **Caso Contrário:**
      - **T** contém um nó especial, denominado **raiz (R_T)**
    - Os demais nós, ou
      - constituem um único conjunto vazio, ou
      - são divididos em n conjuntos disjuntos não vazios (T_1, T_2, ..., T_N), que são, por sua vez, cada qual uma árvore
  - T_1, T_2, ..., T_N são chamadas de **sub-árvores** de R_T



- **Terminologia**

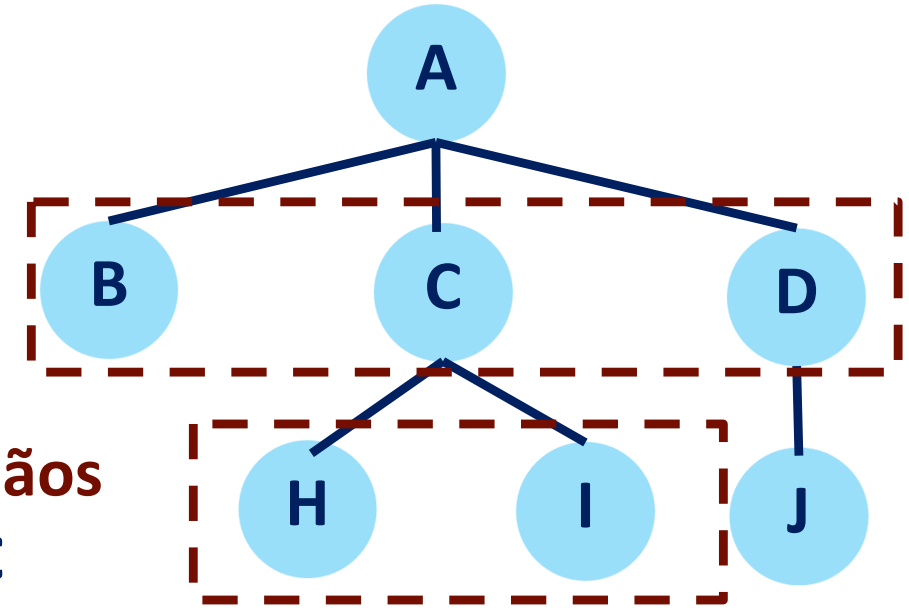
  - Se um **nó Y é raiz** de uma sub-árvore de um nó **X**,
    - **X** é **PAI** de **Y** e
    - **Y** é **FILHO** de **X**

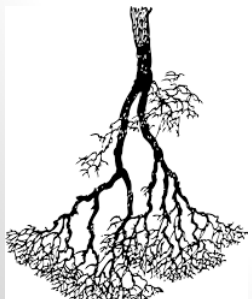


  - Dois nós são **IRMÃOS** se são **filhos do mesmo pai**

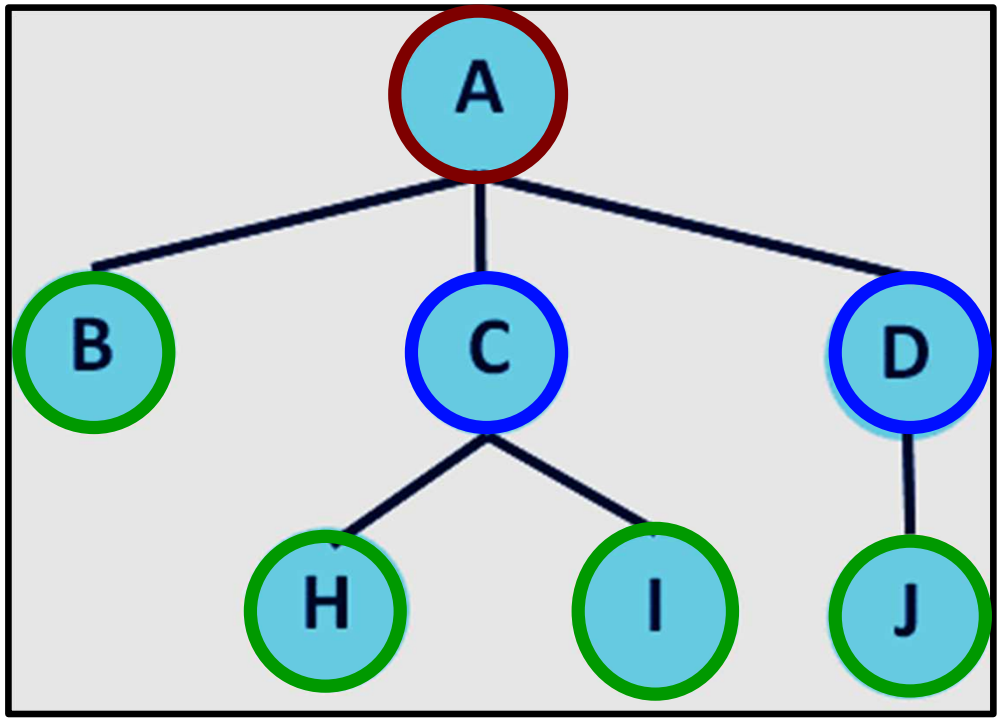
Filhos de A

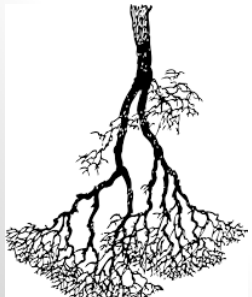
H e I são irmãos
e Filhos de C



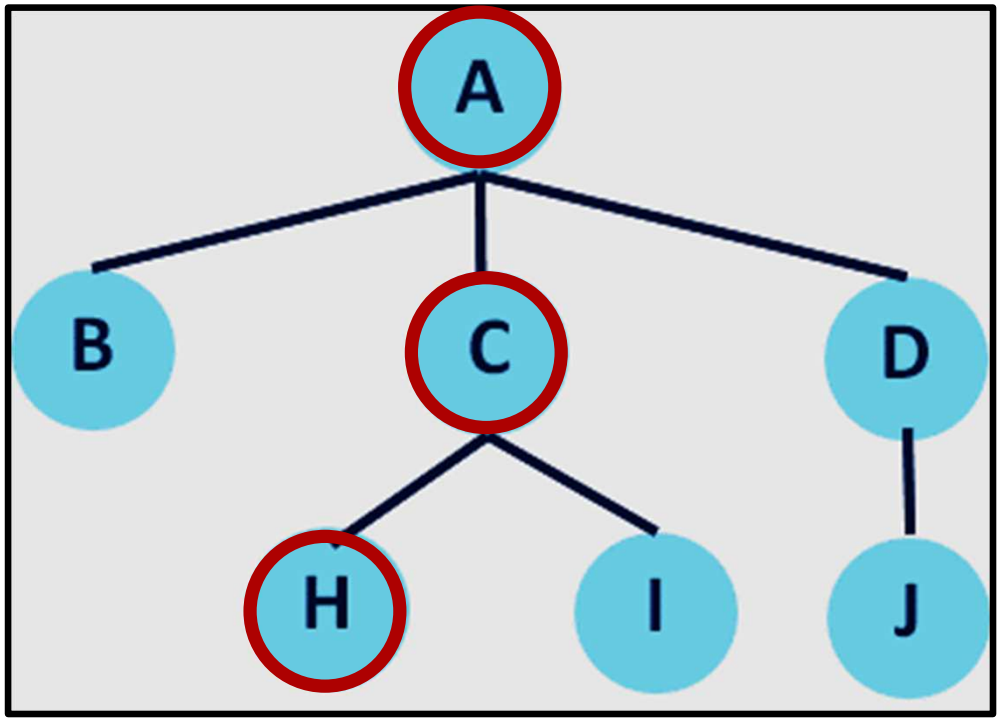


- **Terminologia**
  - **Raiz** (root): nó sem pai **(A).**
  - **Nó Interno:** nó com pelo menos um filho **(C, D).**
  - **Nó Externo ou Folha:** nó sem filhos (B,H, I, J)





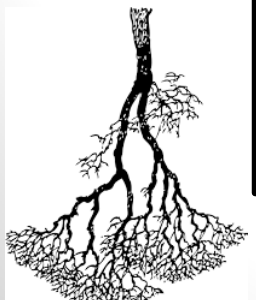
- **Terminologia**
  - **Ancestrais** (de um nó): pai ou ancestrais do pai do nó



Ancestral do nó H:

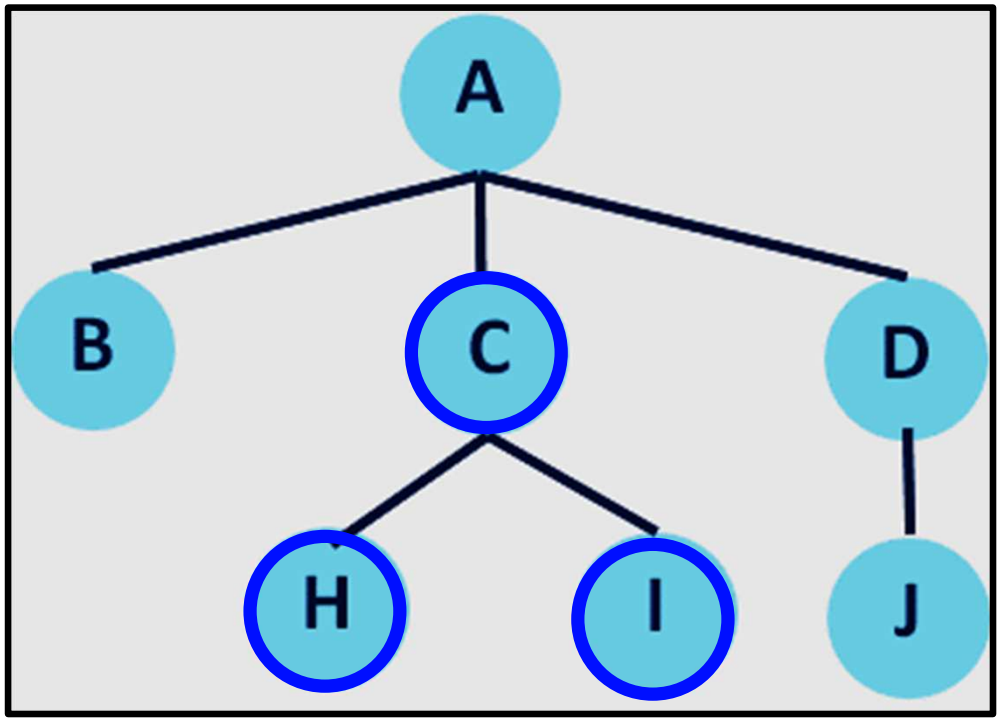
Nó C

Nó A



- **Terminologia**

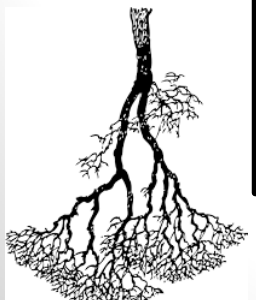
  - **Ancestrais** (de um nó): pai ou ancestrais do pai do nó
  - **Descendentes** (de um nó): nós que o possuem como ancestral.



Ancestral do nó C:

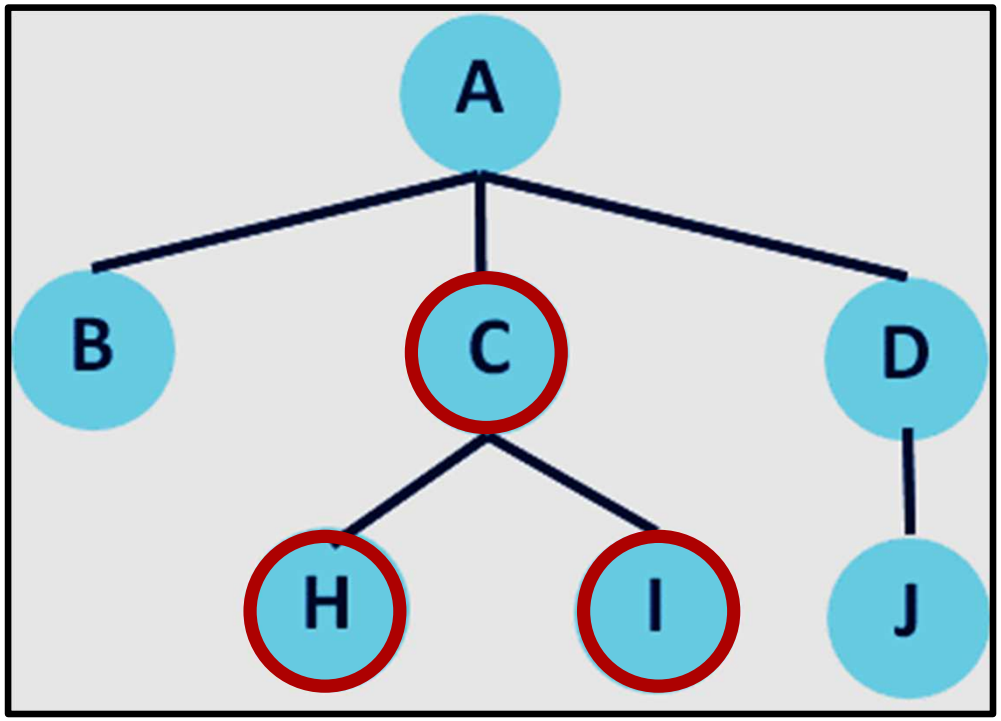
Nó H

Nó I



- **Terminologia**

  - **Ancestrais** (de um nó): pai ou ancestrais do pai do nó
  - **Descendentes** (de um nó): nós que o possuem como ancestral.
  - **Sub-Árvore**: árvore consistindo de um nó e dos seus descendentes.

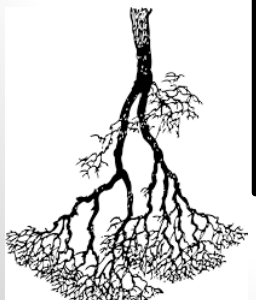


Sub-Árvore C:

Nó C

Nó H

Nó I

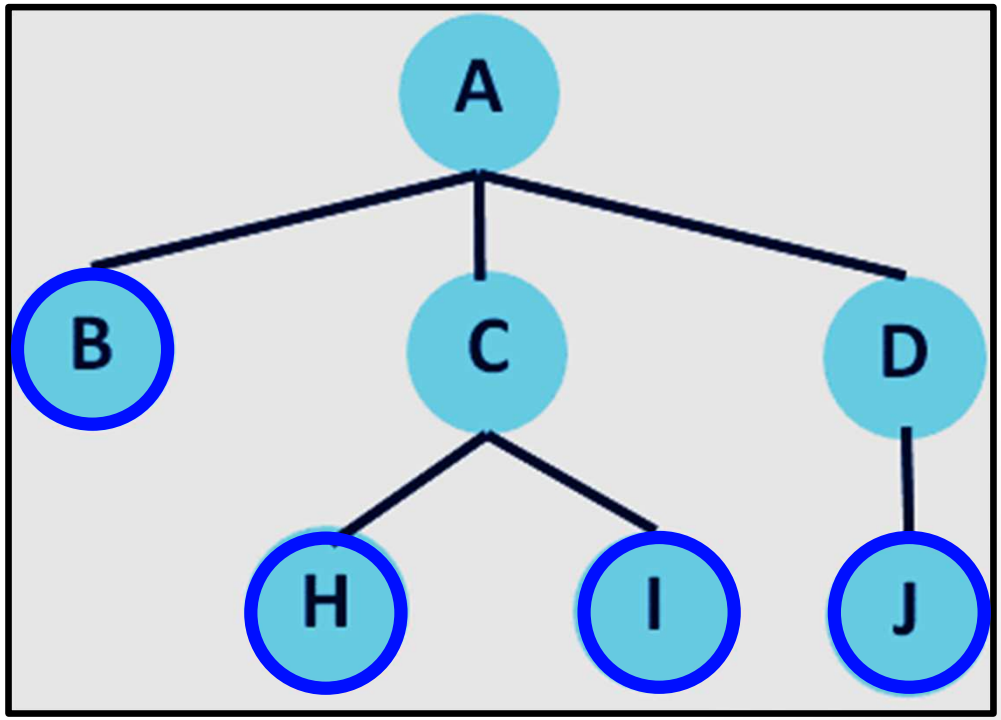


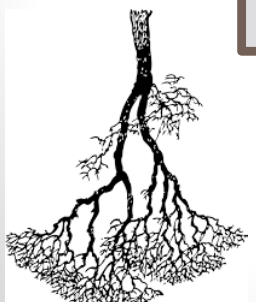
- **Terminologia**
  - **Grau.**
    - de um nó X pertencente a uma árvore é igual ao **número de filhos de X**
    - Se X é **folha**, então Grau(X) = 0

Nós Folhas:

Nó B **Nó H**

Nó I **Nó J**



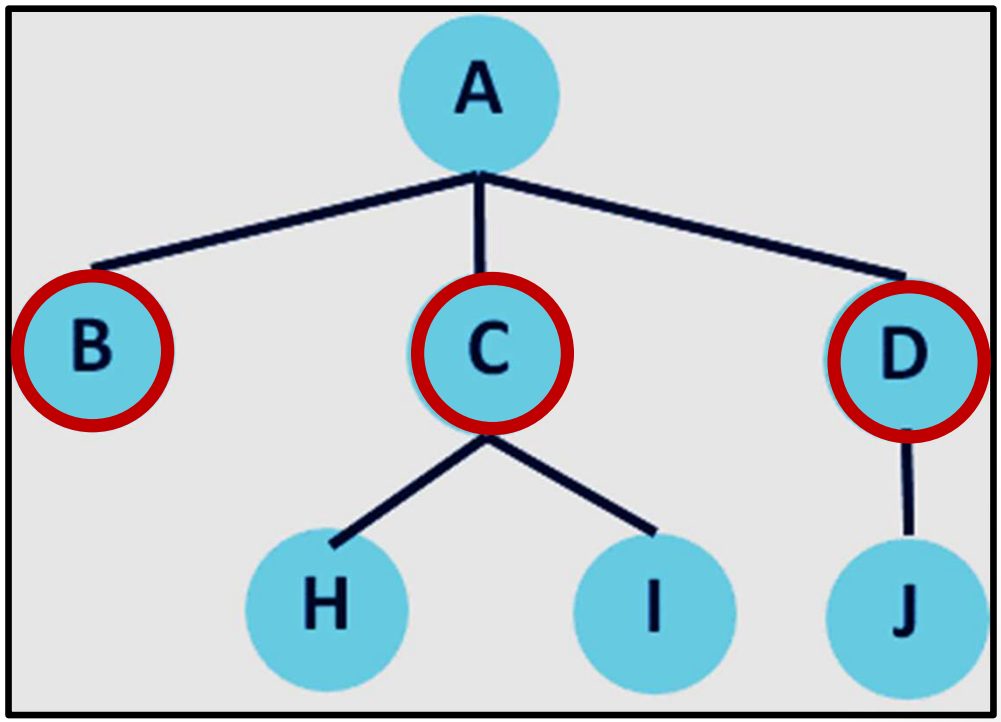


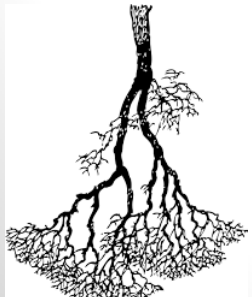
- **Terminologia**
  - **Grau.**
    - **O GRAU de uma árvore T é o maior entre os graus de todos os seus nós**

Nó	Grau
A	3
B	0
C	2

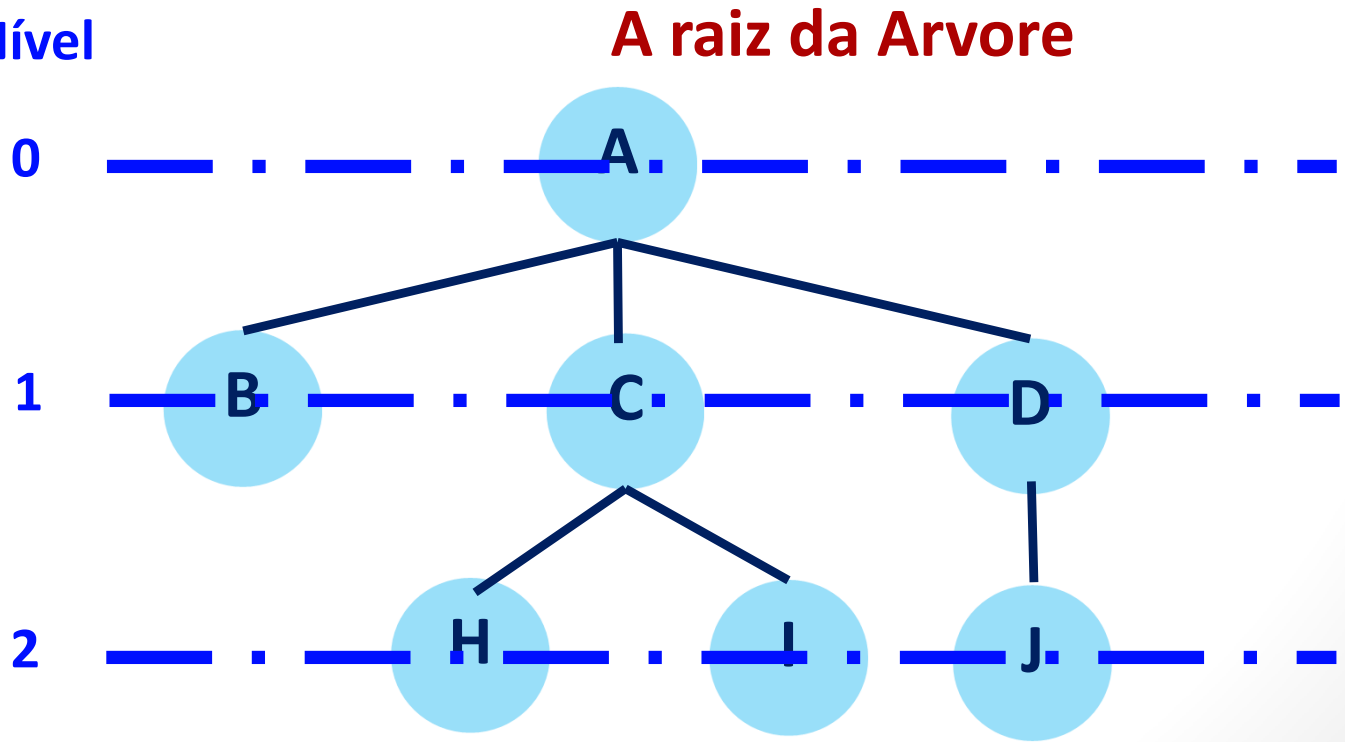
Grau dessa Árvore;

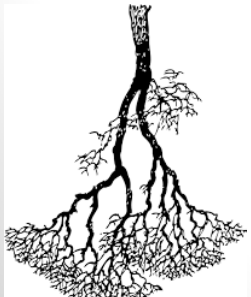
É 3



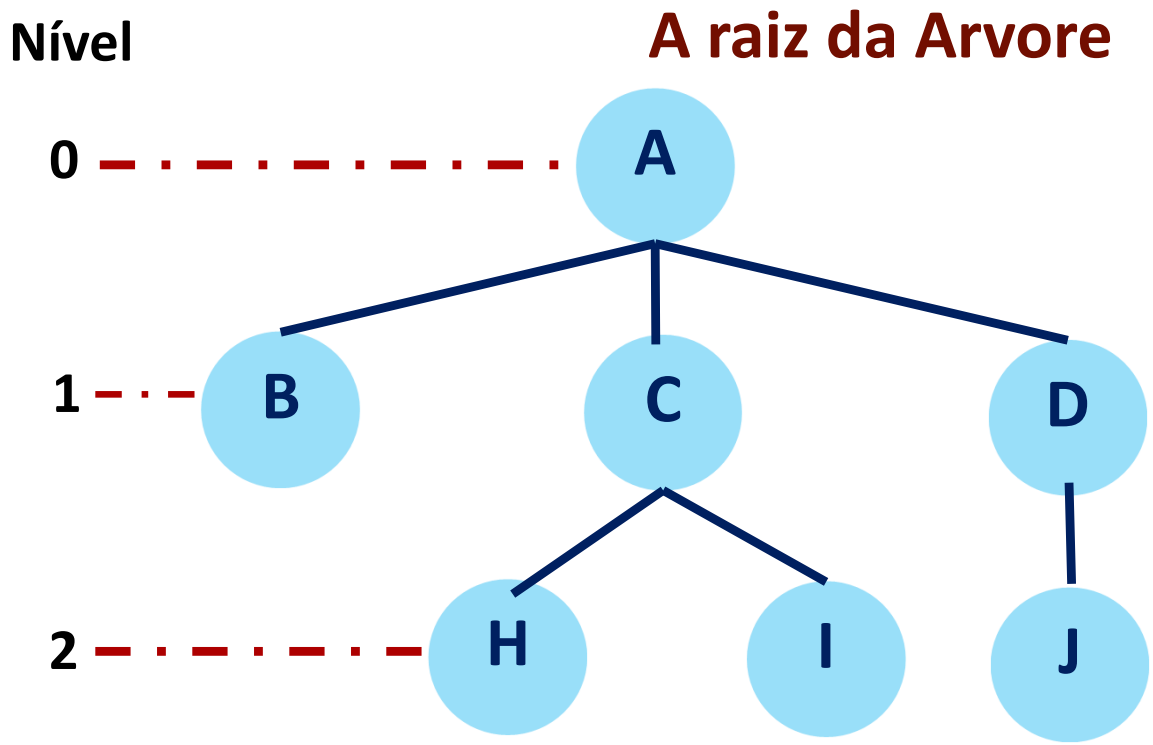


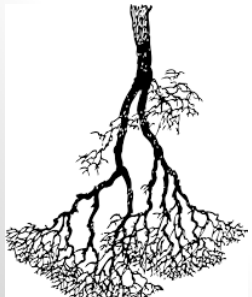
- **Terminologia**
  - **Nível**
    - O nível de um nó raiz é 0.
    - O nível de um nó não raiz é dado por Nível de seu nó PAI + 1





- **Introdução**
  - **Profundidade(Altura)**
    - De um **nó** e o seu **nível**
    - Da **árvore T** é o **maior nível** da árvore





- **Introdução**
  - **Representação de uma Árvore**

Hierárquica - Grafo

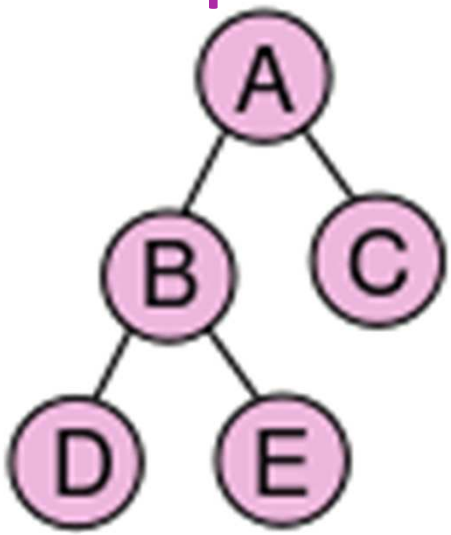
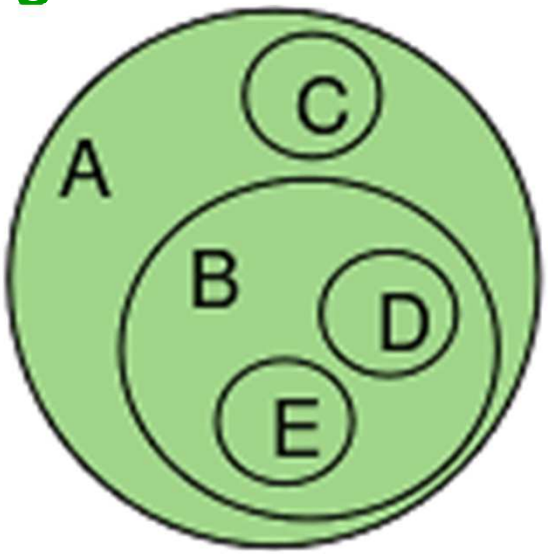
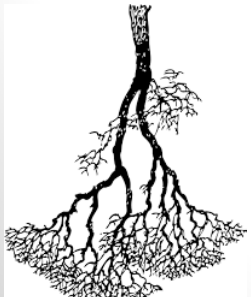


Diagrama de Inclusão

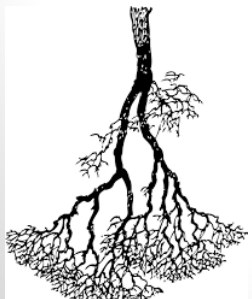


Representação por Parênteses Aninhados

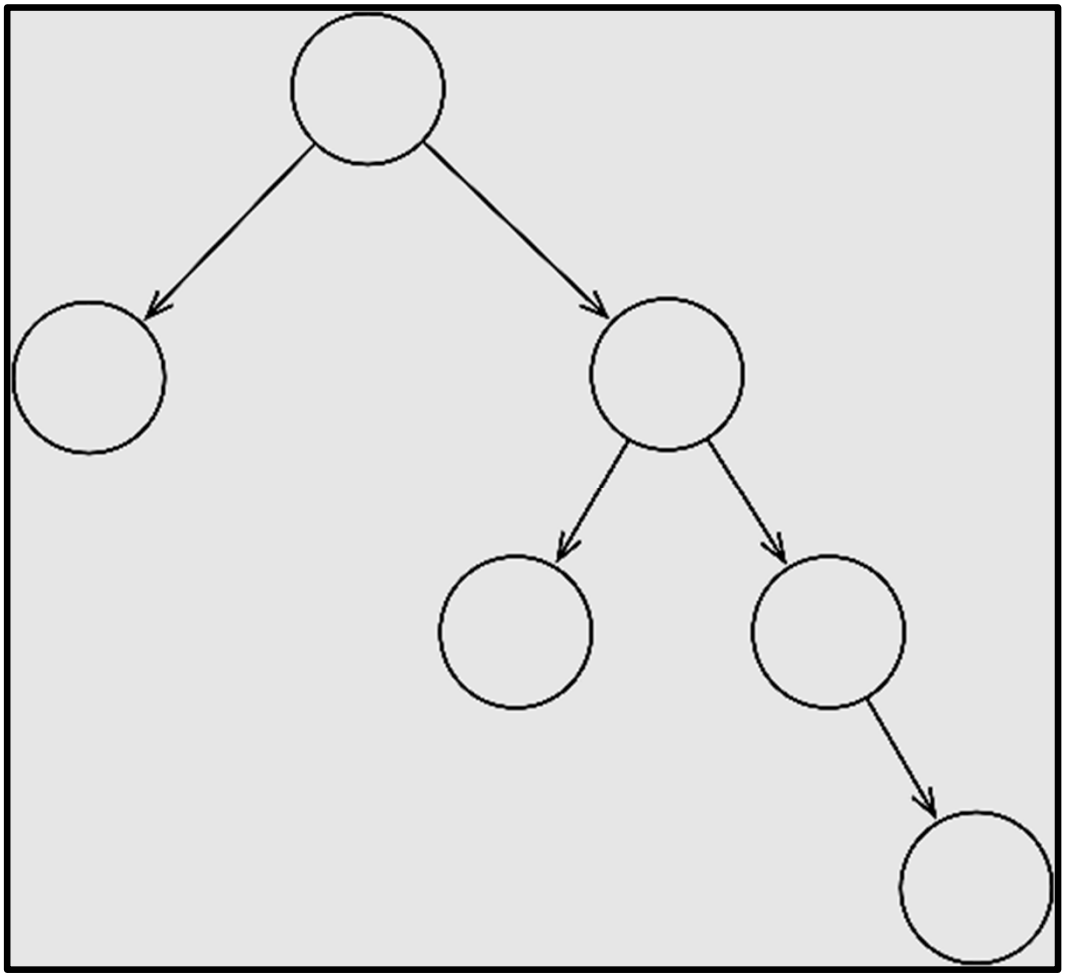
(A(B(D)(E))(C))

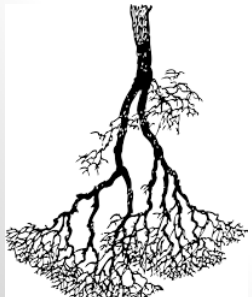


- **Árvore Binária**
  - Uma **Árvore Binária T** é um conjunto finito de elementos, denominados nós, ou vértices, tal que:
    - **Se T = Ø**, a árvore é dita vazia, ou
    - T **contém um nó** especial, chamado **raiz de T (RT),**
    - E os demais nós podem ser **subdivididos em dois subconjuntos distintos, TE e TD**, os quais também são árvores binárias (possivelmente vazias).
    - **TE e TD são denominados sub-árvore esquerda e sub-árvore direita de t respectivamente**



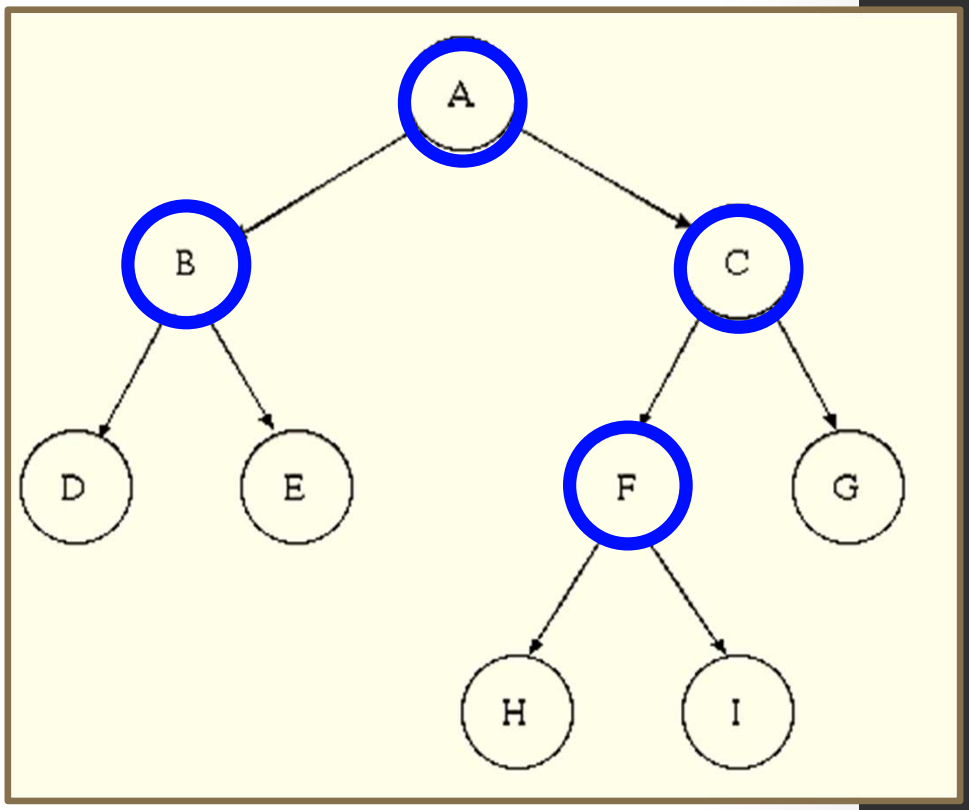
- **Árvore Binária - Exemplo**

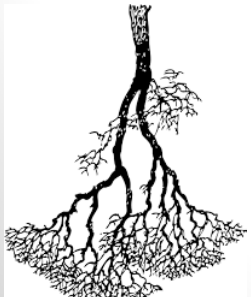




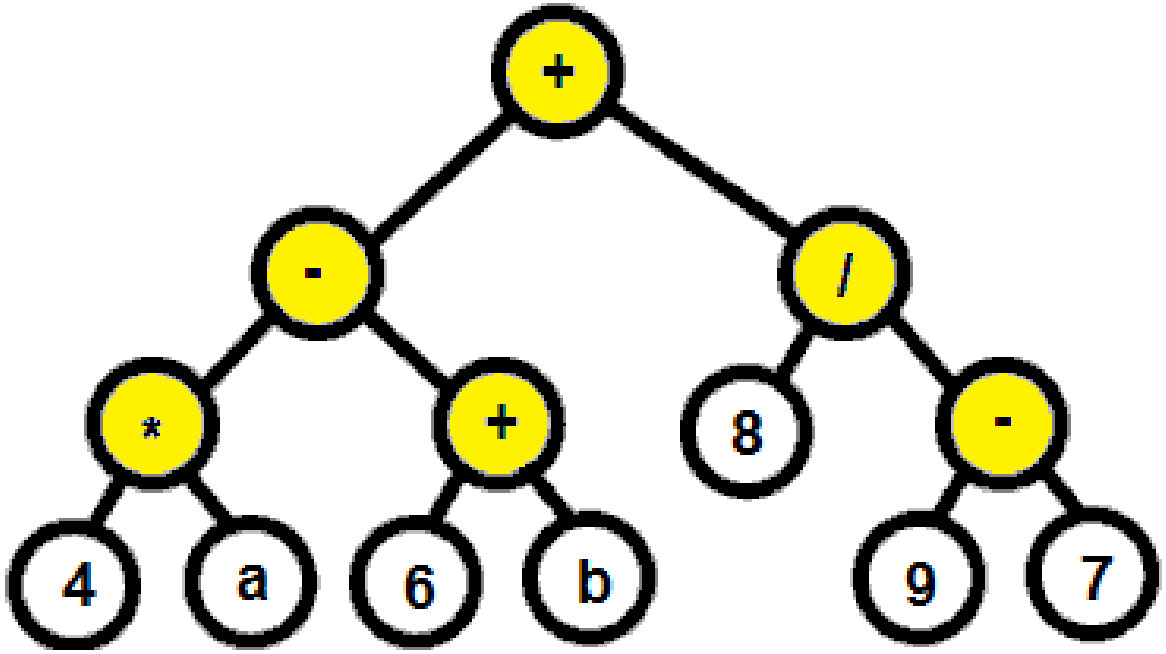
- **Árvore ESTRITAMENTE Binária**

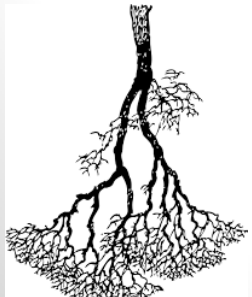
  - Todo nó que **NÃO** é folha possui exatamente 2 filhos (não vazios).
    - filho esquerdo, e
    - filho direito





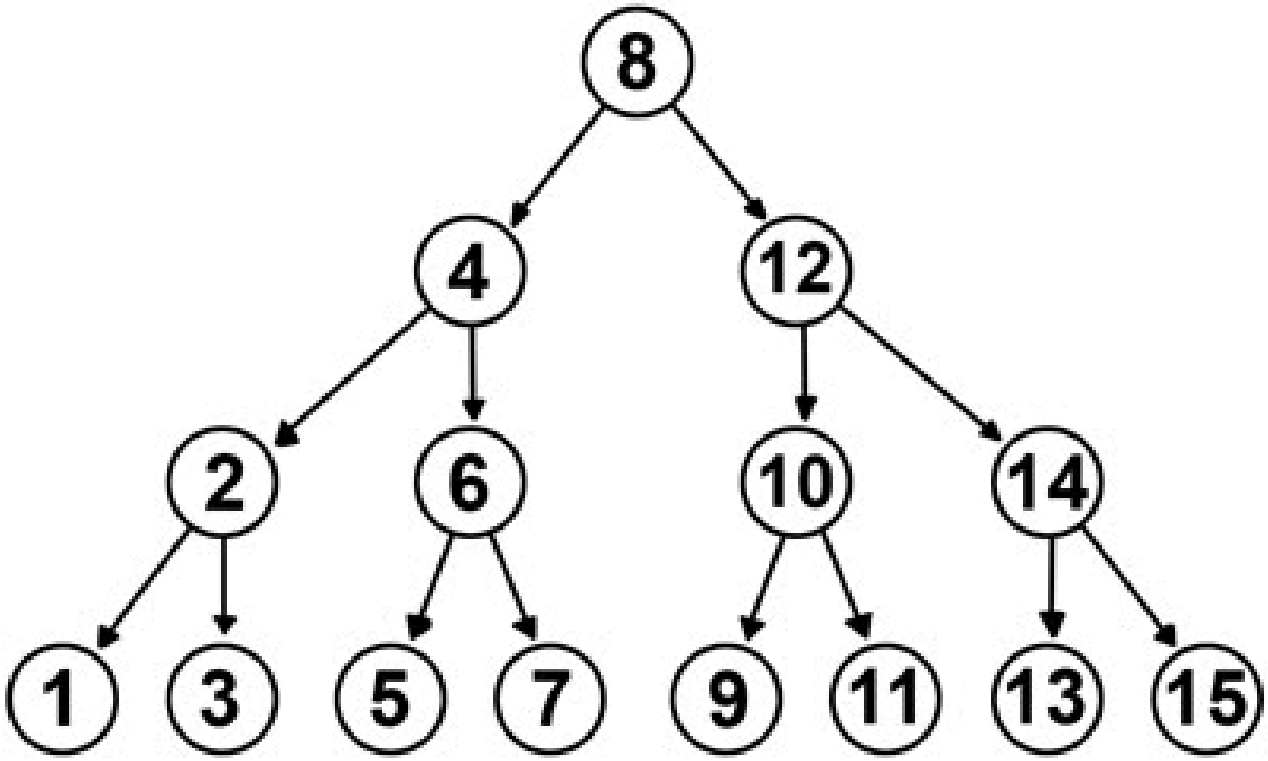
- **Árvore ESTRITAMENTE Binária**
  - **Aplicação:**
    - Árvore Binária de Expressão Aritmética
      - Nós Internos: **Operadores**
      - Nós Externos: **Operandos**
      - Exemplo: **4*a – (6 + b) + 8 / (9 – 7)**

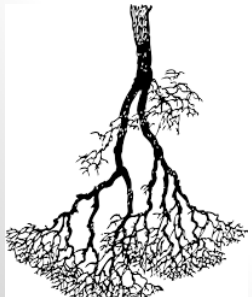




- **Árvore Binária Completa (Cheia)**

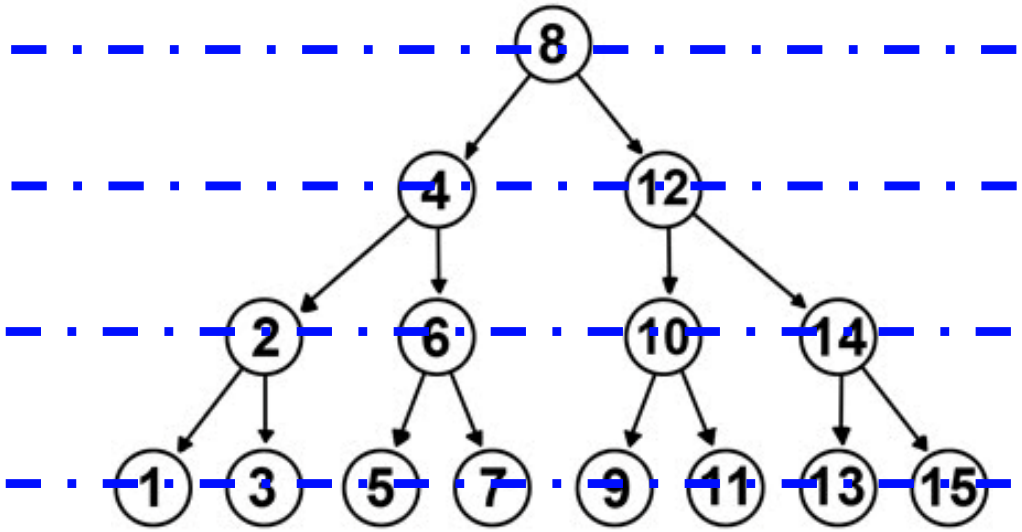
  - Arvore Estritamente Binária, onde **todas as folha estão no mesmo nível**, **ou seja,**
  - **Árvore Binária onde todo nó que não é folha tem dois filhos e todas as folhas estão no mesmo nível**





- **Árvore Binária Completa (Cheia)**

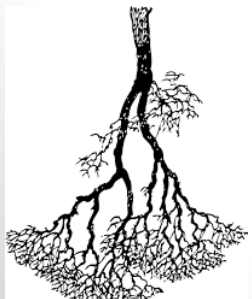
  - **Características**



Nível	Número de nós		
0	1	➡	2^0
1	2	➡	2^1
2	4	➡	2^2
3	8	➡	2^3

Logo **o número de nós desta árvore** é dado por:

$$\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$, onde ***h*** é a **profundidade da árvore**



- **Exercícios**

1. Represente a árvore binária para as seguintes expressões aritméticas:
   - (3+4)*(6-1)+5
   - 8*3 + 2*9-4/2
   - (5 + ((2 / (a + 1)) − (3 × b))) × (10 / c)

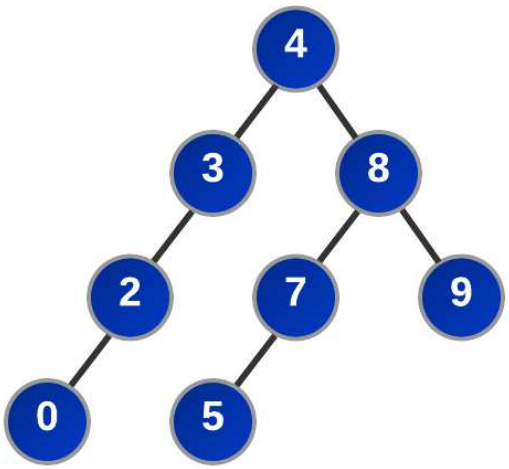
Não se esqueça:
Nós Internos: Operadores
Nós Externos: Operandos

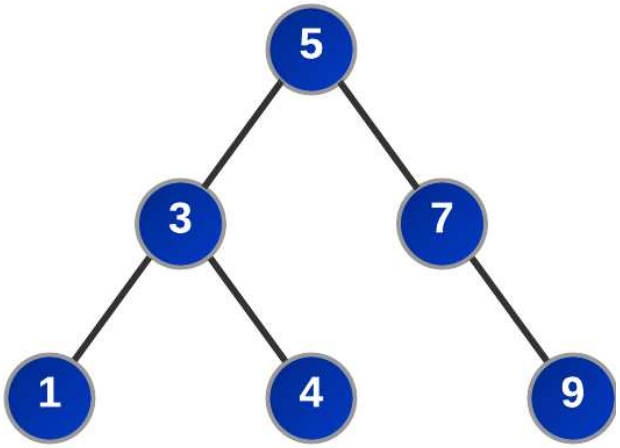
2. Quantos nós possui o nível 4 de uma árvore Binária Completa?
3. Quantos nós possui uma árvore binária completa com profundidade 5?

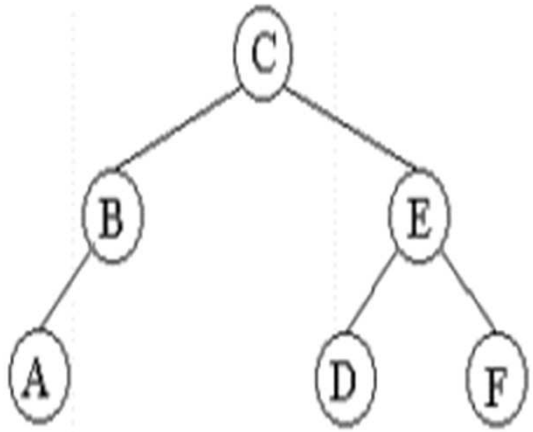
- **Exercícios**

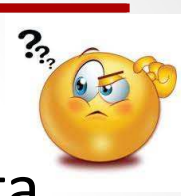
4. Dada as árvores abaixo represente pela forma Hierárquica de Inclusão
   - (50(30(25(10)(27))(45))(80(60(55))))
   - (A(B(C(D(E)(F(G)(H(I))))(K(L)))(M(N))))

5. Dada as árvores abaixo represente pela forma de Inclusão e Parênteses Aninhados







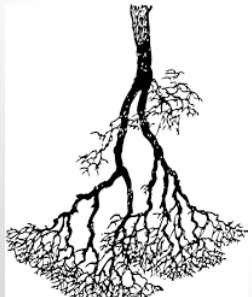


- **Arvores Binárias - Implementação**
  - Toda árvore Binária para ser implementada necessita de uma lei de formação
  - **Vamos pensar um pouco???**
    - Como poderíamos inserir um elemento numa árvore, considerando a lei de formação:
      - **Se Informação > Pai.Info**

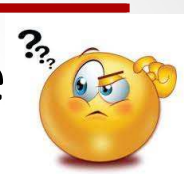
        ENTÃO Filho_Direita

        SENÃO Filho_Esquerda

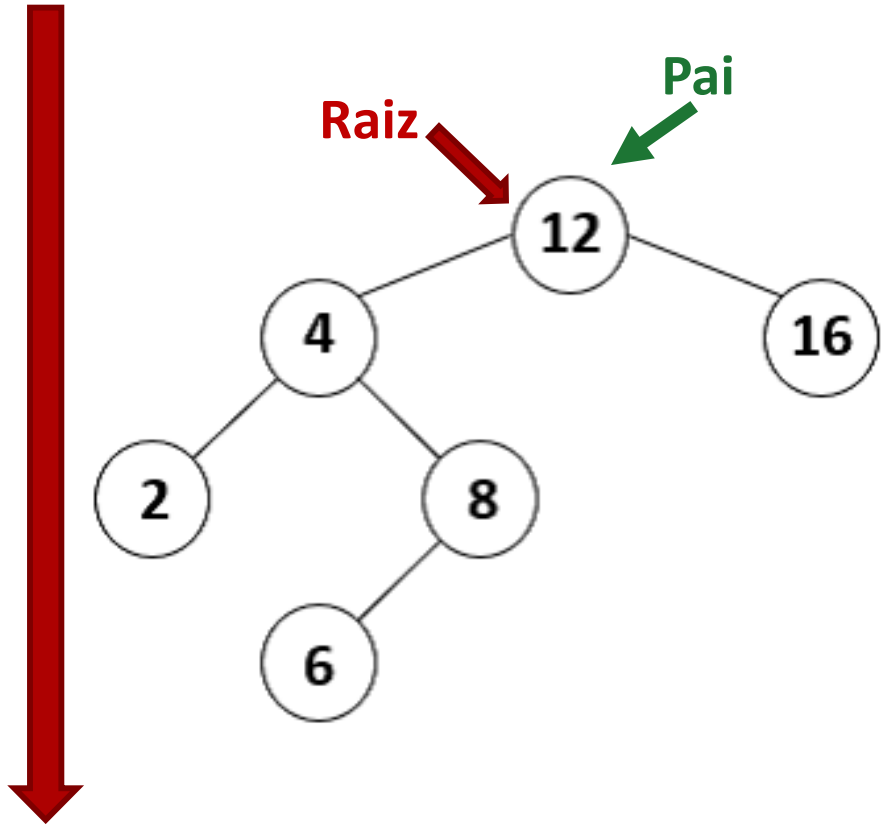
?????...



Arvores Binárias - Implementação

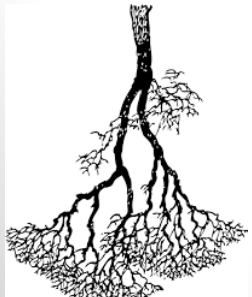


- Vamos pensar numa árvore para entender a lei de formação

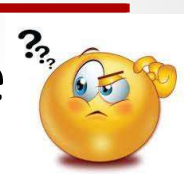


- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**

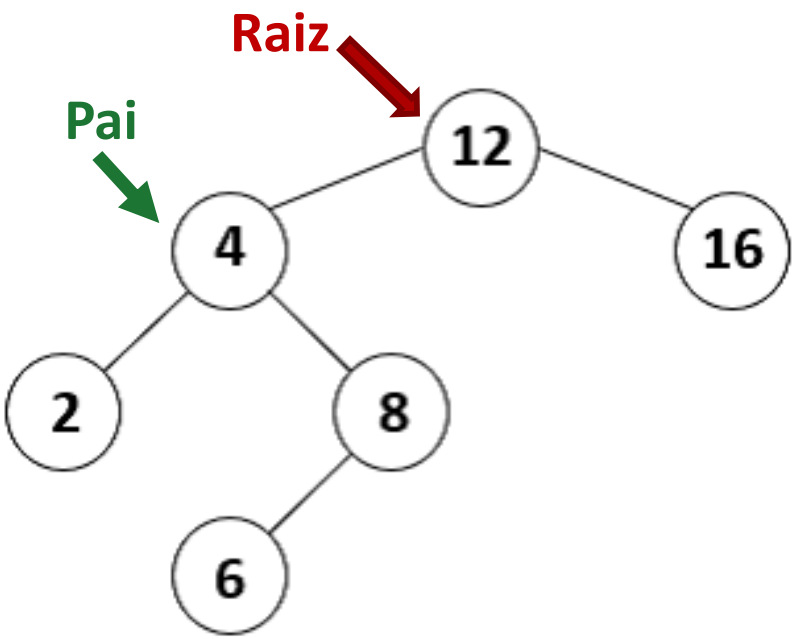
9 é maior que o pai = 12 ??



Arvores Binárias - Implementação

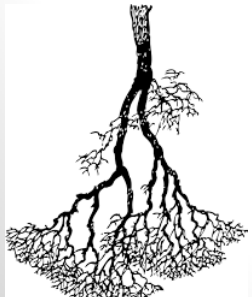


- Vamos pensar numa árvore para entender a lei de formação

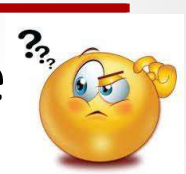


- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**

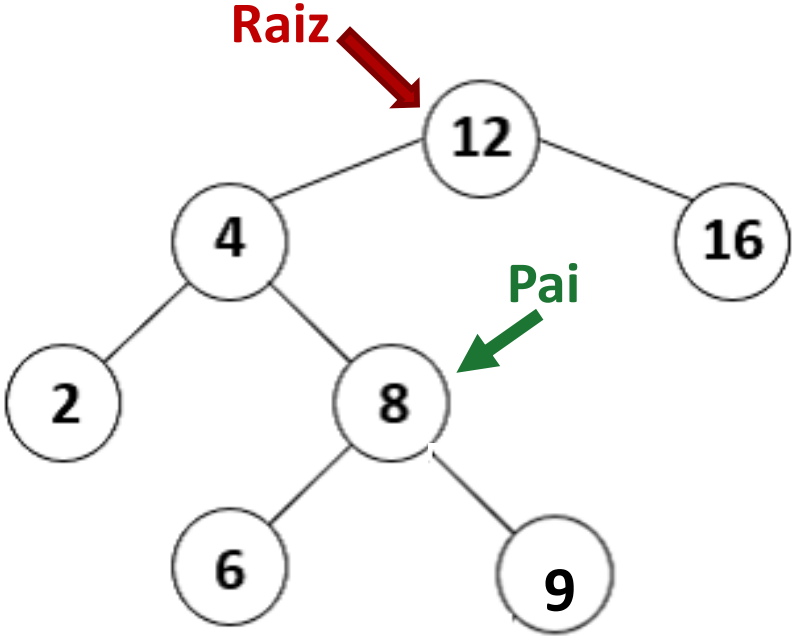
9 é maior que o pai = 4 ??



Arvores Binárias - Implementação

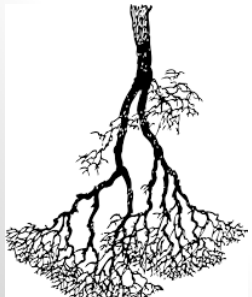


- Vamos pensar numa árvore para entender a lei de formação



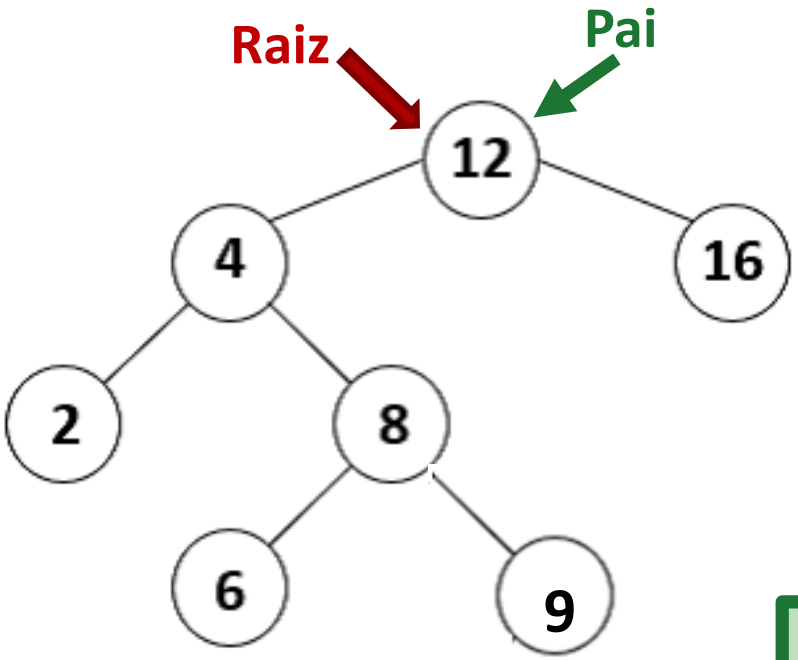
- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**

9 é maior que o pai = 8 ??



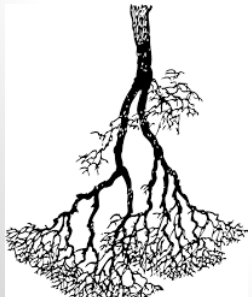
Arvores Binárias - Implementação

- Vamos pensar numa árvore para entender a lei de formação

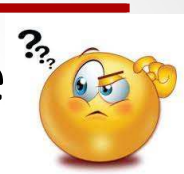


- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**
  - **15**

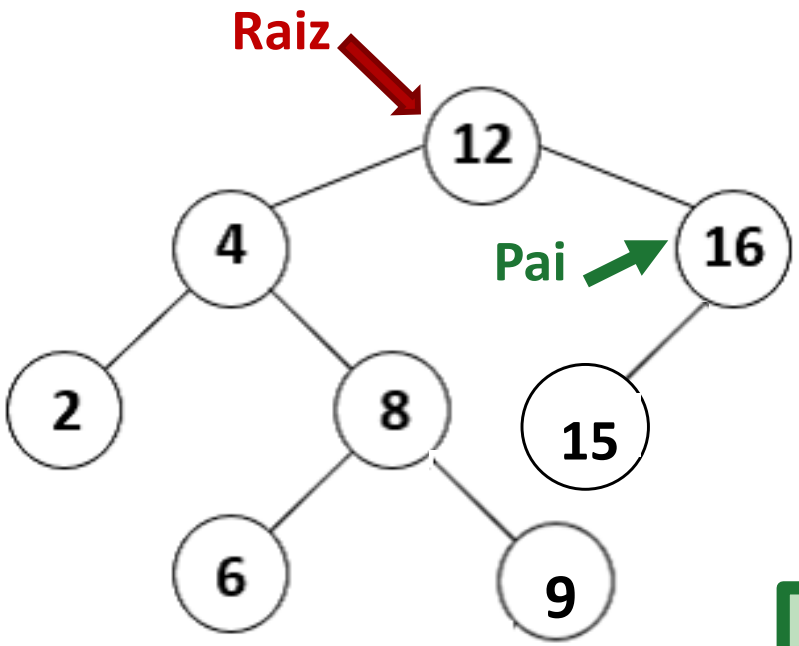
15 é maior que o pai = 12 ??



Arvores Binárias - Implementação

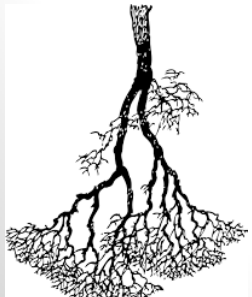


- Vamos pensar numa árvore para entender a lei de formação



- **Raiz: 12**
- **Se fossemos inserir os números:**
  - **9**
  - **15**

15 é maior que o pai = 16 ??

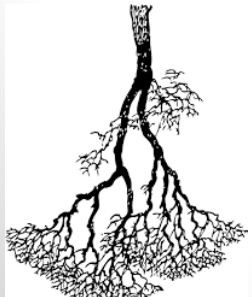


Arvores Binárias - Implementação

- **PseudoCódigo –**

ITERATIVO

```
leia (num)
Aloca espaço para novo
novo.info =num
novo Direita = ∅;
novo.Esquerda = ∅;
if(Raiz == ∅) então Raiz = novo
senão
    Pai = Raiz;
    flag =0;
    Enquanto(flag <>1) faça
        Se (Pai.info < num) então
                Se (Pai.Direita = ∅) então
                        Pai.Direita =novo;
                        flag= 1;
                            senão
                        Pai=Pai.Direita;
            senão
                Se (Pai.Esquerda = ∅ ) então
                        Pai.Esqueda = novo;
                         flag= 1;
                                    senão
                        Pai=Pai.Esquerda;
    Fim Enquanto
```

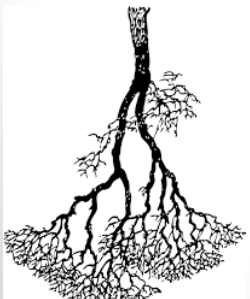


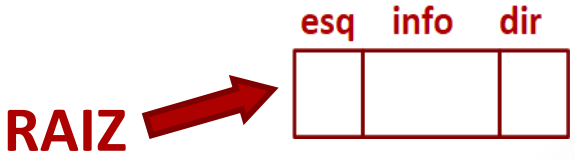
Arvores Binária - Implementação

- **Representação de um NÓ da árvore**:
  - Estrutura em C contendo
    - A **informação propriamente dita** (exemplo: um inteiro)
    - **Dois ponteiros** para as sub-árvores:
      - da esquerda e
      - da direita

- **Representação de uma ÁRVORE**:

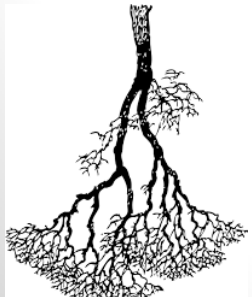
  Através de um **ponteiro para o nó raiz**





- **Estrutura em C contendo**
  - A **informação propriamente dita** (exemplo: um inteiro)
  - **Dois ponteiros** para as sub-árvores, da esquerda e da direita


```
typedef struct NoArvore
{
    int info;
    struct NoArvore *esq;
    struct NoArvore *dir;
}NoArv;
```

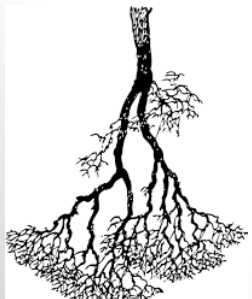


Arvores Binária - Implementação


```
typedef struct NoArvore
{
    int info;
    struct NoArvore *esq;
    struct NoArvore *dir;
}NoArv;
```

- **Criar uma estrutura para armazenar a raiz dessa árvore**

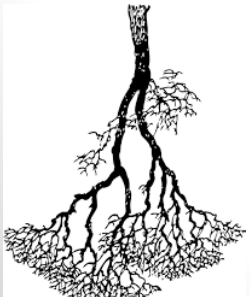
```
typedef struct Arvore
{
    NoArv *raiz;
}Arv;
```



Arvores Binária - Implementação

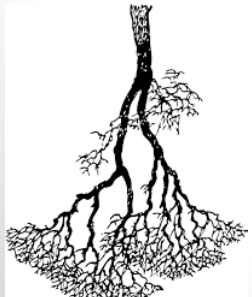
- **Funções Básicas para Manipulação de uma Árvore:**

  - **Inserir**
  - Verificar se o elemento existe na árvore (Busca)
  - Criar
  - Verificar se está vazia
  - Imprimir
  - Liberar
  - Remover



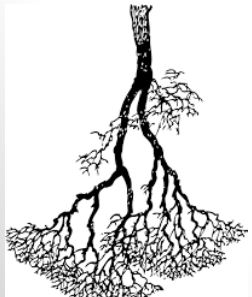
Funções Básicas para Manipulação de uma Árvore:

```
void insere(Arv *Arvore, int num)
{
    Arvore->raiz=aux_insere(Arvore->raiz,num);
}
```



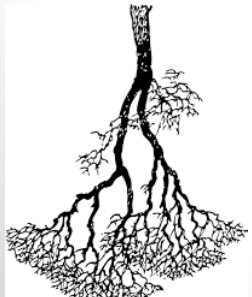
Funções Básicas para Manipulação de uma Árvore:

```
NoArv* aux insere(NoArv *no, int num)
{
    int flag;
    NoArv *Pai;
    NoArv *novo = (NoArv*)malloc(sizeof(NoArv));
    novo->info = num;
    novo->esq=NULL;
    novo->dir=NULL;
    if(no==NULL)
    {
        return novo;
    }
```



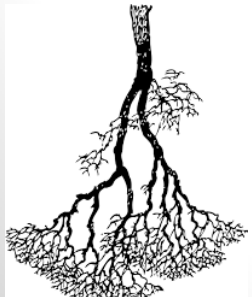
a Árvore:

```
    else
    {
        Pai = no;
        flag=0;
        while (flag==0)
        {
            if(Pai->info<num)
            {
                if(Pai->dir==NULL)
                {
                    Pai->dir = novo;
                    flag=1;
                }
                else
                {
                    Pai=Pai->dir;
                }
            }
            else
            {
                if(Pai->info>num)
                {
                    if(Pai->esq==NULL)
                    {
                        Pai->esq = novo;
                        flag=1;
                    }
                    else
                    {
                        Pai=Pai->esq;
                    }
                }
            }
        }
    }
    return no;
}
```



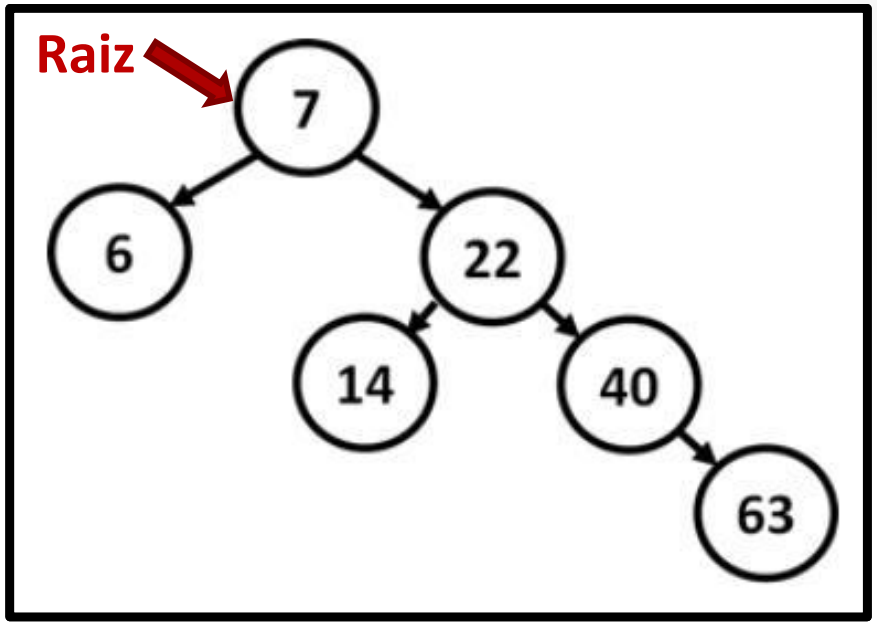
Arvores Binária - Implementação

- **Procedimentos de Percorrimento**
  - Tem por **finalidade percorrer a árvore como um todo.**
  - **Podem ser utilizados para imprimir a árvore**
  - **São três:**
    - **Pré-Order**
    - **In-Order**
    - **Pos-Order**

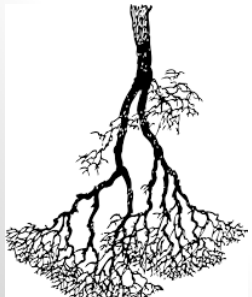


- **Procedimento Pré-Order - *Impressão***

```
PreOrder( Pai)
inicio
    imprime(Pai-info)
    Se (F_Direita≠∅) então
          PreOrder(F_Direita)
    Se (F_Esquerda≠∅) então
          PreOrder(F_Esquerda)
fim
```



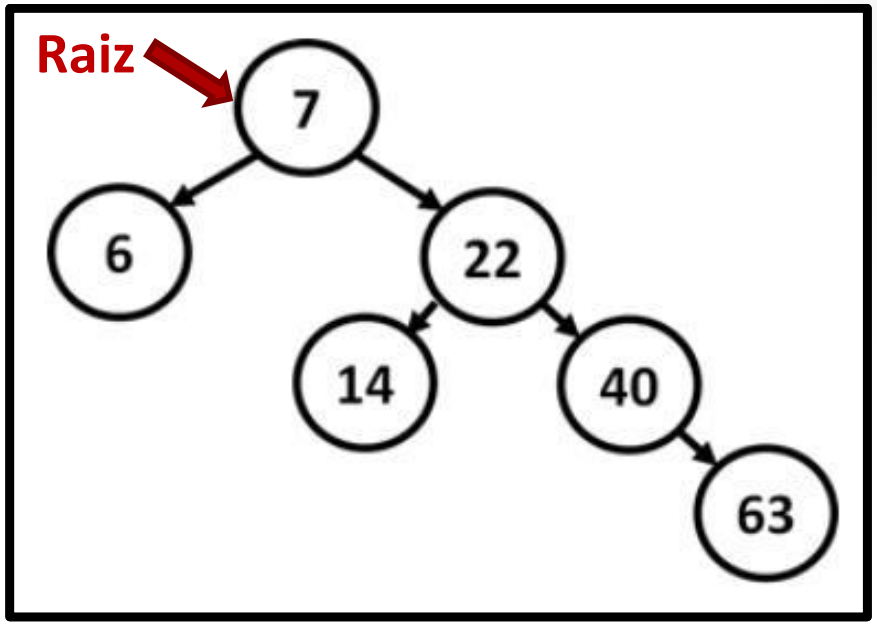
7 - 22 - 40 - 63 - 14 - 6

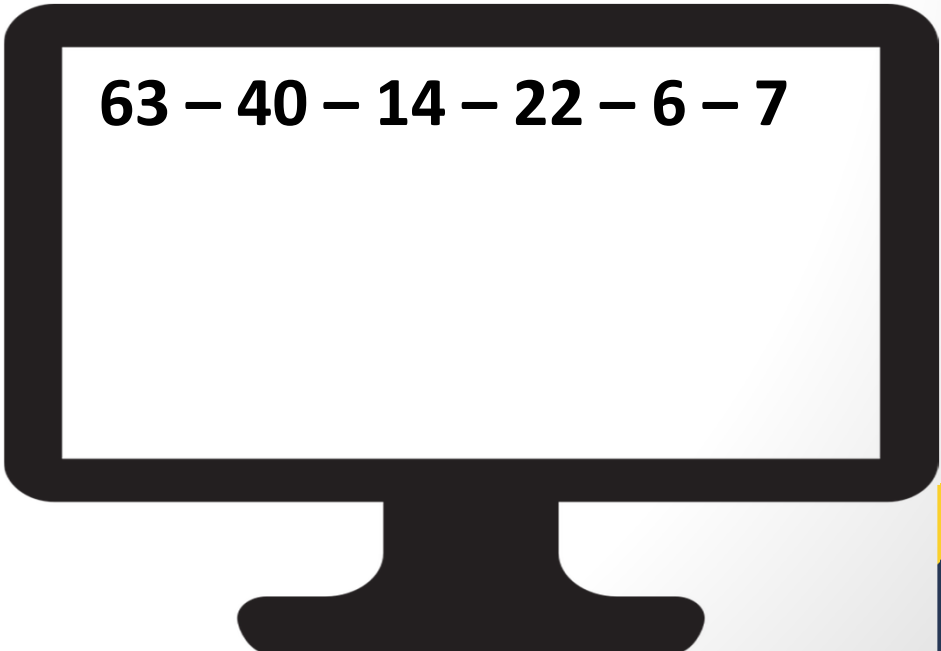


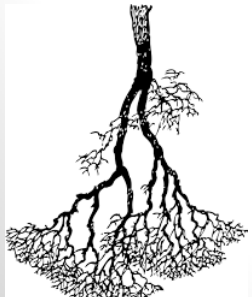
Arvores Binária - Implementação

- **Procedimento Pos-Order - *Impressão***

```
PosOrder( Pai)
inicio
   Se (F_Direita≠∅) então
         PosOrder(F_Direita)
   Se (F_Esquerda≠∅) então
         PosOrder(F_Esquerda)
    imprime(Pai-info)
fim
```



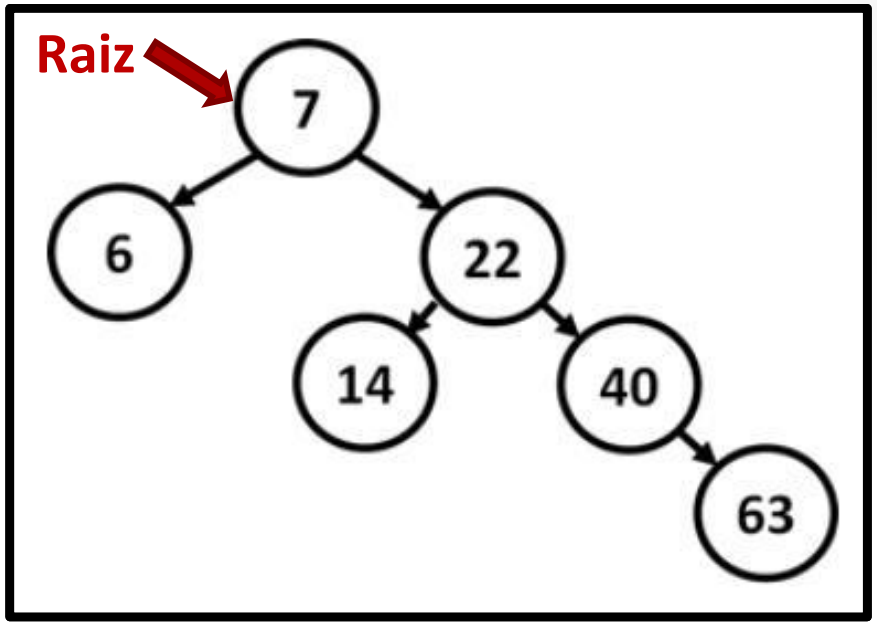


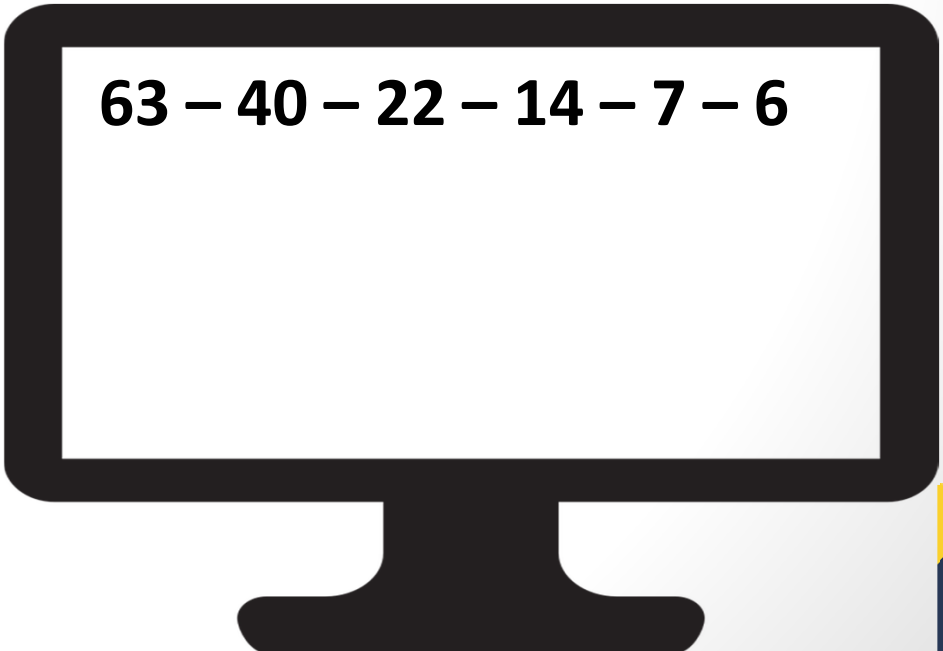


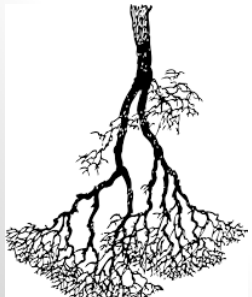
Arvores Binária - Implementação

- **Procedimento In-Order - *Impressão***

```
InOrder( Pai)
inicio
   Se (F_Direita≠∅) então
         InOrder(F_Direita)
    imprime(Pai-info)
   Se (F_Esquerda≠∅) então
         InOrder(F_Esquerda)
fim
```



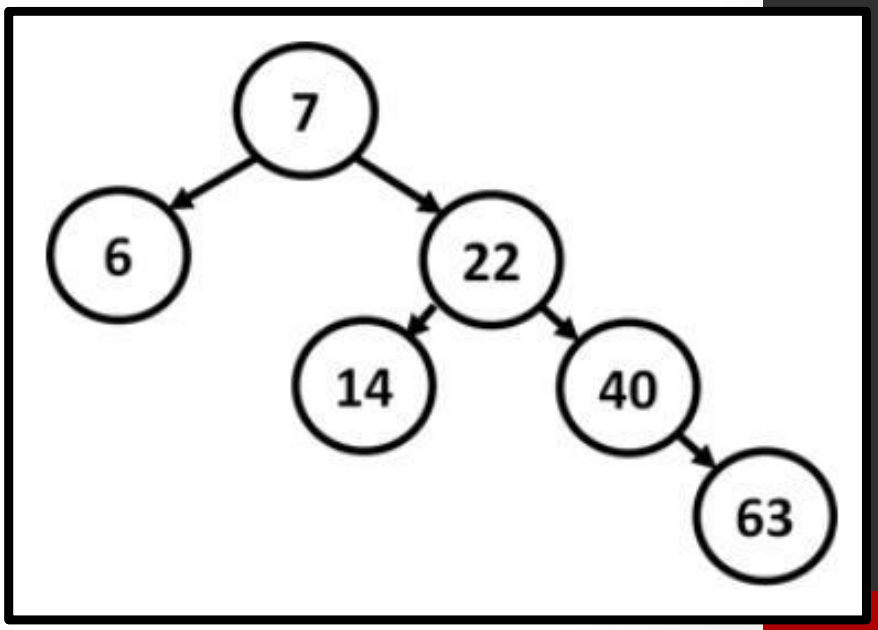


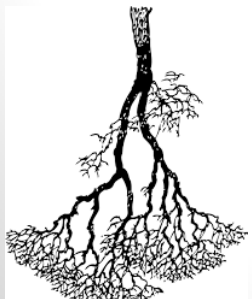


Arvores Binária - Implementação

- **Comentários para Elaboração de Procedimentos**

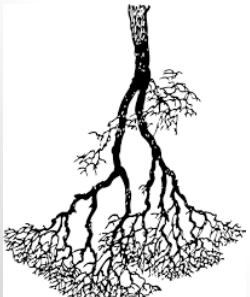
  - Basicamente todo procedimento de Árvore Binária pode ser desenvolvido pela adaptação de dois procedimentos básicos:
    - **Algoritmo de Busca ou Inserção**, onde a **lei de formação é conhecida** e usada para otimizar o procedimento
    - **Algoritmo de Percorrimento,** quando a árvore tem que ser **percorrida integralmente**





- **Exercícios**

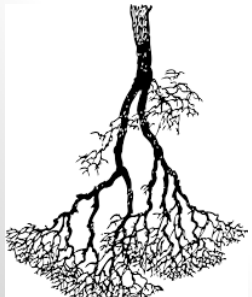
1. Construa um procedimento de busca de um certo elemento em uma árvore
2. Construa os procedimentos para verificar se uma árvore está vazia e o procedimento de criar uma árvore



- **Procedimento de Busca**
  - **Vamos pensar um pouco???**
    - Como poderíamos **buscar** um elemento numa árvore, considerando a lei de formação:
      - **Se Informação > Pai.Info**

        ENTÃO Filho_Direita

        SENÃO Filho_Esquerda



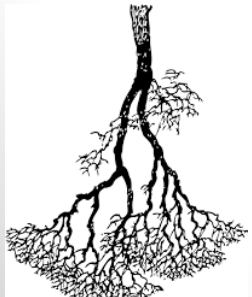
?????...

Arvores Binárias - Implementação

- **PseudoCódigo –**

ITERATIVO

```
leia (num)
Aloca espaço para novo
novo.info =num
novo Direita = ∅;
novo.Esquerda = ∅;
if(Raiz == ∅) então Raiz = novo
senão
    Pai = Raiz;
    flag =0;
    Enquanto(flag <>1) faça
        Se (Pai.info < num) então
                Se (Pai.Direita = ∅) então
                        Pai.Direita =novo;
                        flag= 1;
                            senão
                        Pai=Pai.Direita;
            senão
                Se (Pai.Esquerda = ∅ ) então
                        Pai.Esqueda = novo;
                         flag= 1;
                                    senão
                        Pai=Pai.Esquerda;
    Fim Enquanto
```



Arvores Binárias - Implementação

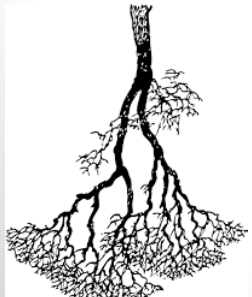
- **PseudoCódigo –**

  ITERATIVO

flag:

- **0 – Está procurando**
- **1 – Valor EXISTE**
- **2 – Valor NÃO Existe**

if(Raiz == ∅) então *"Arvore Vazia"*



senão

Pai = Raiz;

flag =0;

Enquanto flag==0 faça

Se (num>Pai.info) então

Se (Pai.Direita = ∅) então

flag= 2;

senão

Pai=Pai.Direita;

senão

Se (num<Pai.info) então

Se (Pai.Esquerda = ∅)

então

flag= 2;

senão

Pai=Pai.Esquerda;

senão

flag= 1;

Fim Enquanto

Funções Básicas para Manipulação de uma Árvore:

- Criar uma árvore
- Verificar se a árvore está vazia

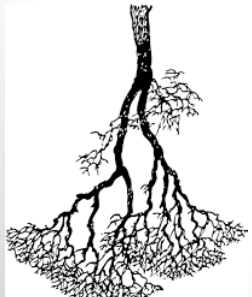
```
typedef struct NoArvore
{
      int info;
      struct NoArvore *esq;
      struct NoArvore *dir;
}NoArv;
```

```
typedef struct BaseArv
{
      NoArv *raiz;
}Arv;
```

ESTRUTURAS USADAS

```
Arv* Criar_Arvore()
{
    Arv *aux;
    aux=(Arv*)malloc(sizeof(Arv));
    aux->raiz = NULL;
    return aux;
}
```

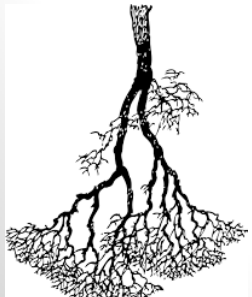
```
int ArvVazia(Arv *base)
{
    if(base->raiz==NULL)
    {
        return 1;
    }
    return 0;
}
```



Funções B

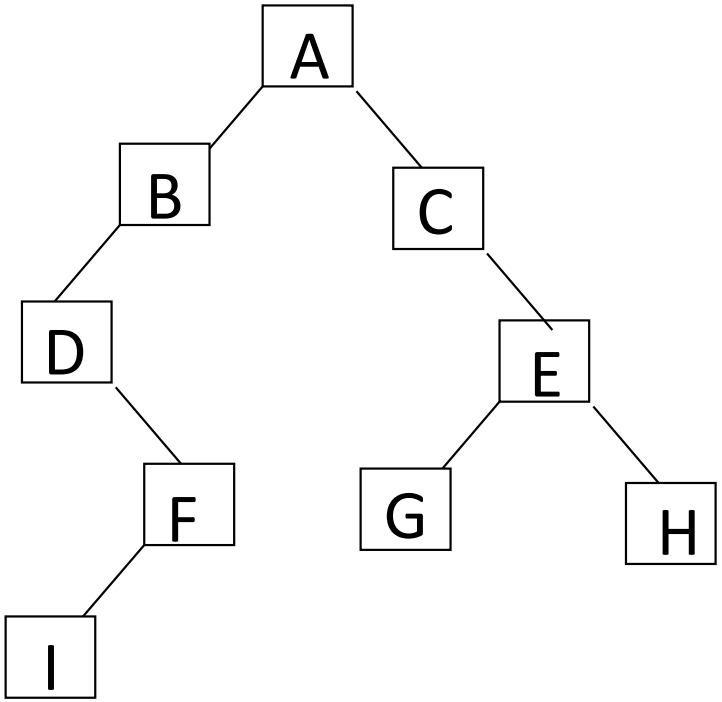
FALTA CONSTRUIR.....

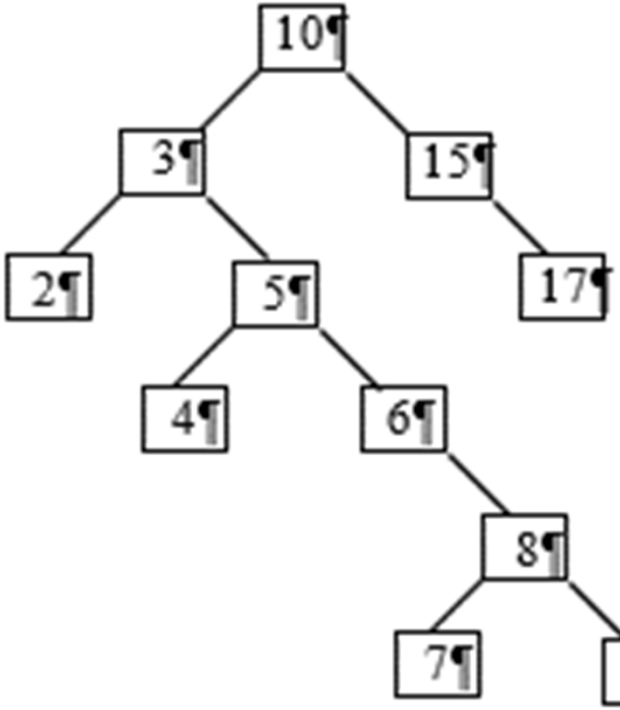
```c
int main()
{
    setlocale(LC_ALL,"portuguese");
    int i, num;
    Arv *RAIZ=NULL;
    RAIZ = Criar_Arvore();

    if(ArvVazia(RAIZ))
    {
        printf("\n\nÁRVORE VAZIA\n\n");
    }

    for(i=0; i<max; i++)
    {
        printf("\tDigite um número: ");
        scanf("%d",&num);
        insere(RAIZ,num);
    }
    if(!ArvVazia(RAIZ))
    {
        printf("\n\n\t\t ==> IMPRESSÃO\n\t");
        imprime_preOrder(RAIZ->raiz);
    }
    else
    {
        printf("\n\nÁRVORE VAZIA\n\n");
    }
    liberaArvore(RAIZ->raiz);
    free(RAIZ);
    RAIZ = NULL;
}
```

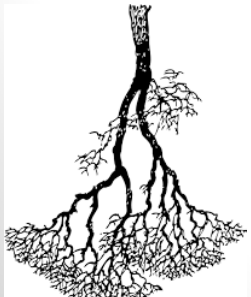


- **Exercícios**

1. Faça um teste de mesa para as árvores abaixo, usando os algoritmos pré, in e pós order







- **Exercícios**

2. Elabore um procedimento que

   a) Determine a soma de elementos de uma árvore

   b) Determine o número de ancestrais de uma determinada informação, se ela existir na árvore.

   c) Número de descendentes de um nó se ele existir na árvore

   d) Imprime o pai de um certo nó, se ele existir na árvore