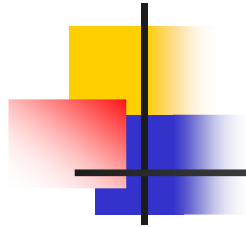


# Algoritmos e Estruturas de Dados



---

Prof. Marcelo Zorzan  
Profa. Melissa Zanatta



# Situação-Problema

---

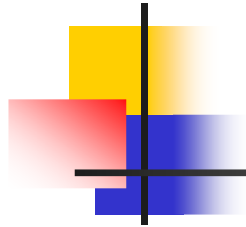
- Maria desde criança gostou de ouvir as histórias contadas por sua mãe sobre o modo de vida de seus avós, tataravós. Depois de adulta ela resolveu resgatar os integrantes de sua família montando uma árvore genealógica. Com o objetivo de agilizar essa tarefa ela procurou por profissionais de TI para implementar uma solução computacional em que as atividades de inserção, remoção e busca tivessem o melhor desempenho possível.
  - Considerando as estruturas de dados vistas até agora, que solução você propõe?



# Aula de Hoje

---

- ED Árvore

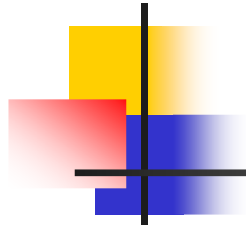


# Problema

---

- Representações/Implementações do TAD Lista Linear:
  - Lista **encadeada dinâmica**: eficiente para inserção e remoção dinâmica de elementos (início ou fim), mas ineficiente para busca.
  - Lista **seqüencial (ordenada) estática**: eficiente para busca (busca binária), mas ineficiente para inserção e remoção de elementos (requer abrir espaços)

“Haveria uma estrutura de dados que tivesse o melhor desempenho nas 3 operações?”



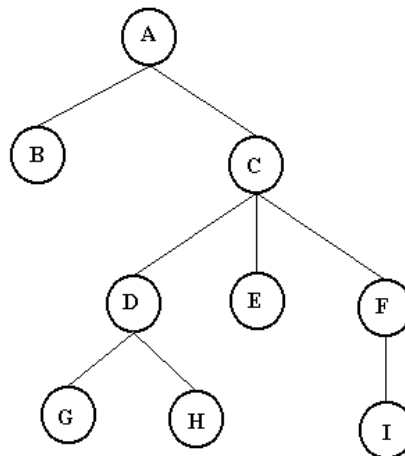
# Solução

---

- Árvores: solução eficiente para inserção, remoção e busca
- Representação não-linear mais importante em algoritmos computacionais.
- Conceitualmente diferente das listas encadeadas, em que os dados se encontram numa sequência.

# Árvore - definição

- Uma árvore  $A$  é uma estrutura hierárquica composta por  $n \geq 0$  nós. Se  $n = 0$  dizemos que a árvore é vazia; caso contrário:
  - \* Existe um nó especial em  $A$  denominado raiz;
  - \* Os demais nós de  $A$  são organizados em  $A_1, A_2, \dots, A_k$ , estruturas de árvores disjuntas, denominadas subárvores de  $A$ .



**Representação gráfica  
de uma árvore**



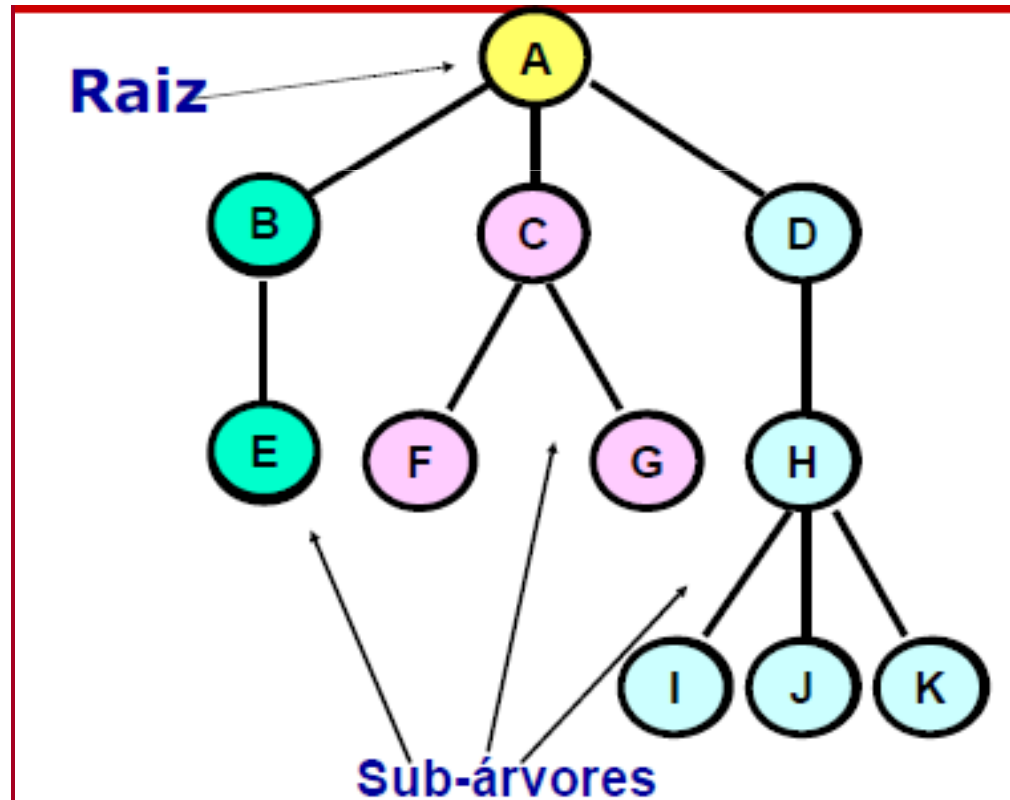
# Árvore – aplicação

---

- As árvores, diferente das listas, não são estruturas de dados lineares. São estruturas úteis para representar informações que precisam ser organizadas de forma hierárquica
- Exemplos:
  - estruturas de diretórios, subdiretórios e arquivos,
  - interfaces gráficas com o usuário (organização dos menus, por exemplo),
  - organização das páginas de um site,
  - organização hierárquica de cargos ou setores de uma empresa, etc

# Árvore – Terminologias

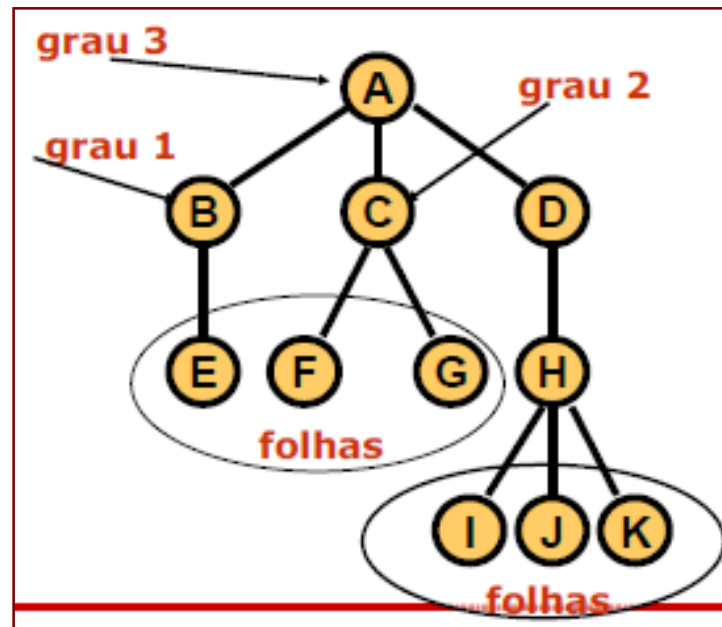
- **Raiz:** primeiro nó de uma árvore.





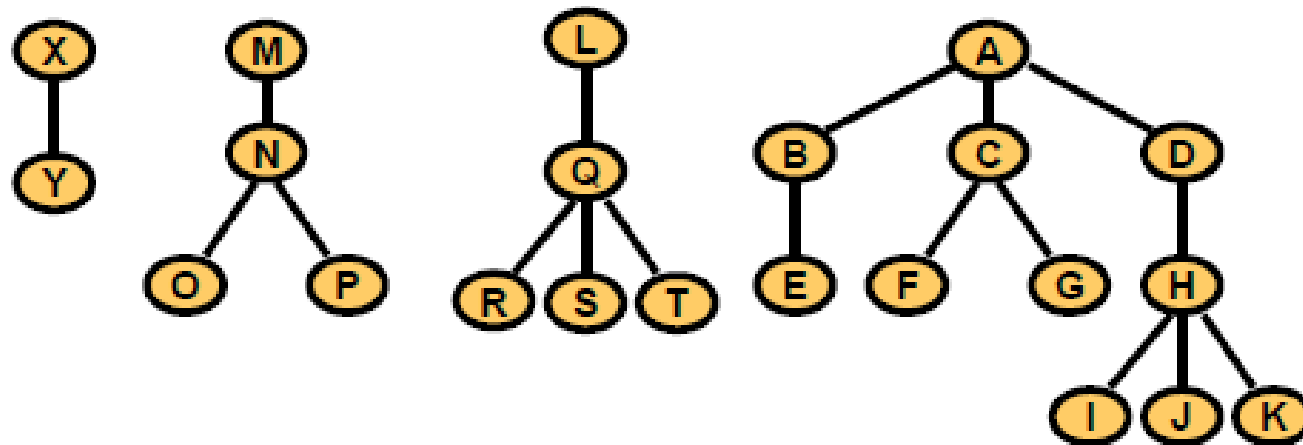
# Árvore – Terminologias

- **Grau** de um nó é igual ao número de sub-árvores (ou número de filhos) do mesmo
- **Folha ou Terminal**: nó com grau igual a zero
- **Grau de uma árvore**: maior grau entre seus nós



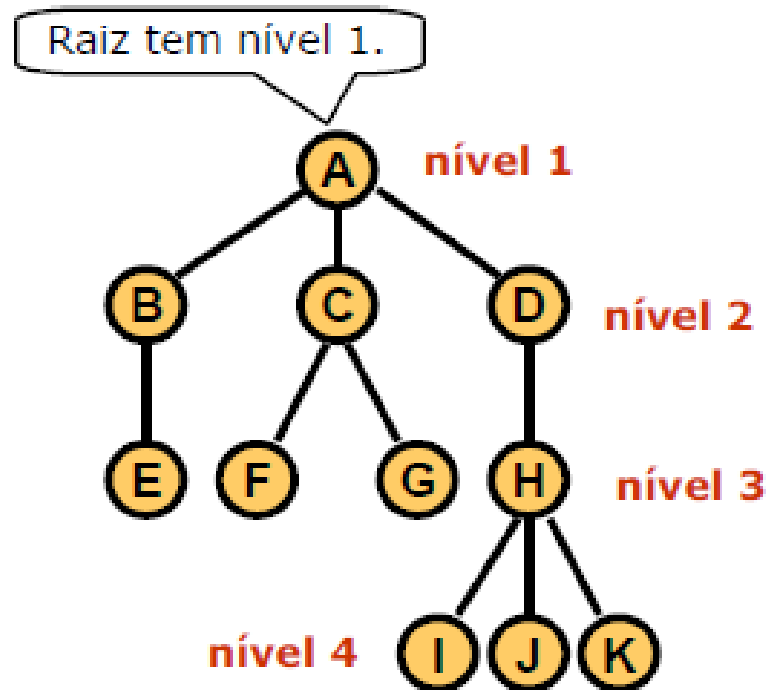
# Árvore – Terminologias

- **Floresta** é conjunto de zero ou mais árvores disjuntas.
- Há pouca distinção entre árvores e florestas:
  - se excluirmos a raiz de uma árvore, teremos uma floresta,
  - se adicionarmos um nó a uma floresta, teremos uma árvore.



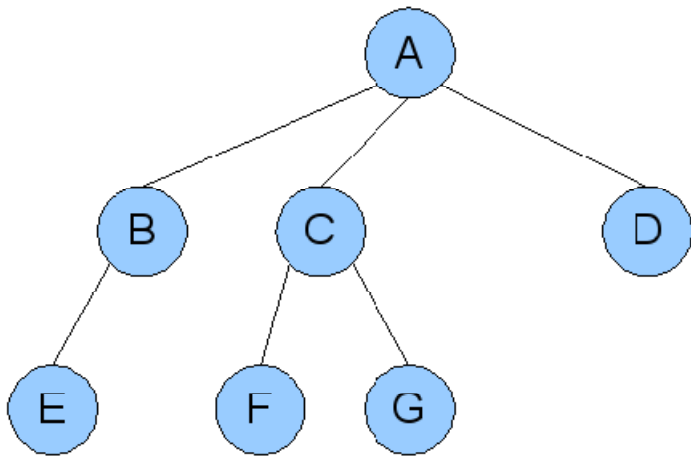
# Árvore – Terminologias

- **Nível** de um nó é o número de nós entre ele e a raiz.
- **Altura ou profundidade** de uma árvore é igual ao seu maior nível

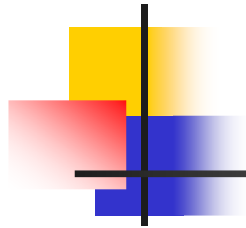


# Árvore – Terminologias

- A raiz A possui três subárvores: {B; E}, {C; F; G} e {D}
- A árvore {C; F; G} tem o nó C como raiz. O nó C está no nível 2 quando comparada à árvore toda.



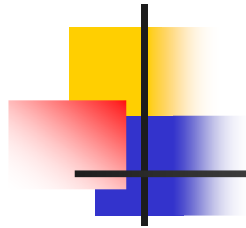
Nó	Grau	Nível	Tipo
A	3	1	Raiz
B	1	2	
C	2	2	
D	0	2	Folha
E	0	3	Folha
F	0	3	Folha
G	0	3	Folha



# Árvore – Representação

---

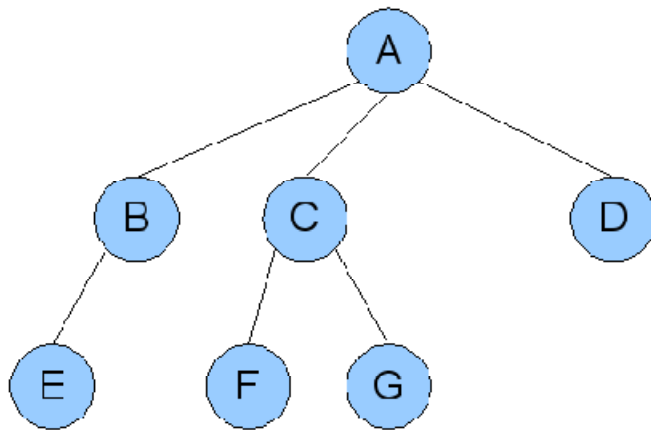
- Há vários exemplos de representação gráfica de árvores:
  - ✓ grafo,
  - ✓ conjuntos aninhados,
  - ✓ parênteses aninhados,
  - ✓ endentação



# Árvore – Representação

- Representações em grafo e em parênteses aninhados

Grafo

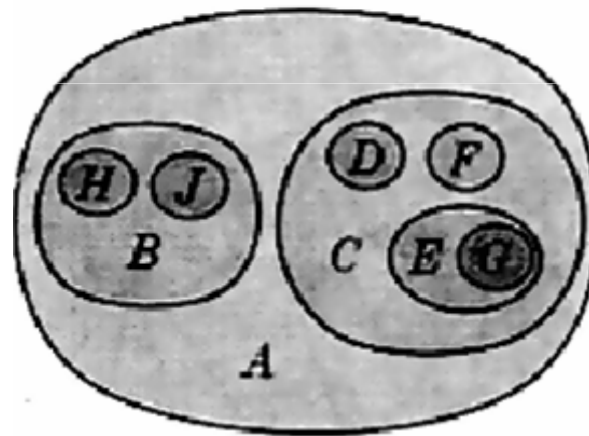


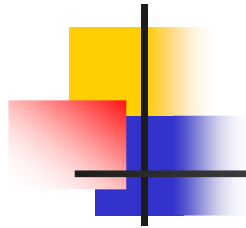
Parênteses aninhados

$(A(B(D)(E))(C(F)))$

# Árvore – Representação

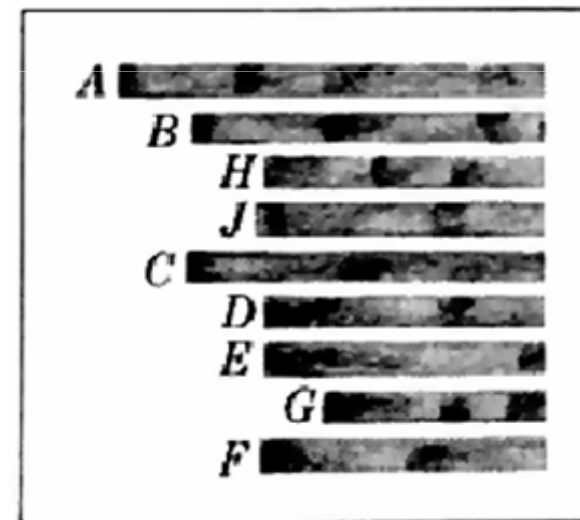
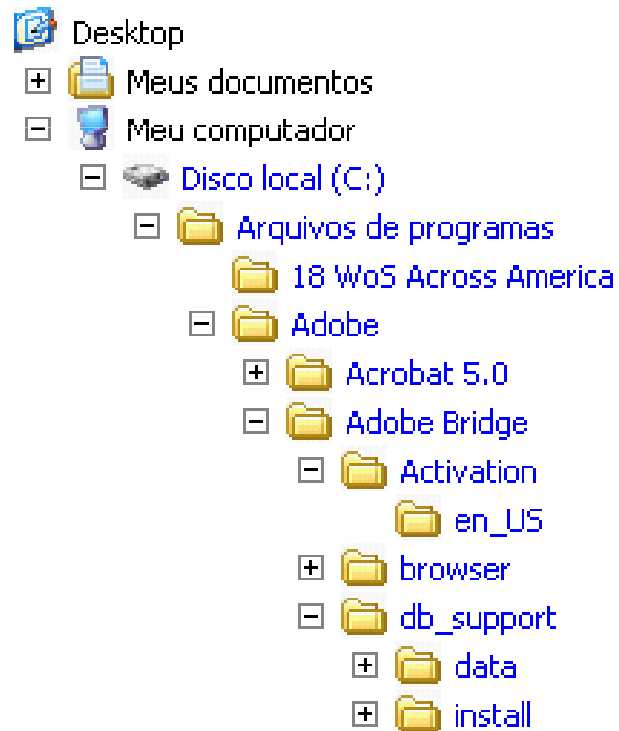
Representação em conjuntos aninhados





# Árvore – Representação

## Representação por endentação







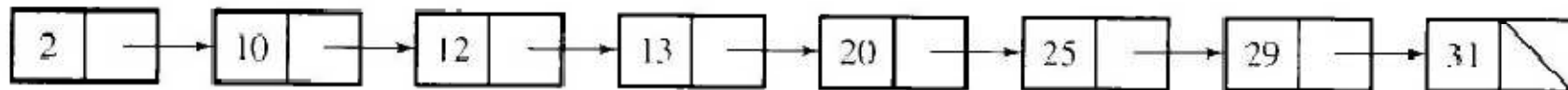
# Árvore: acelerando a pesquisa

---

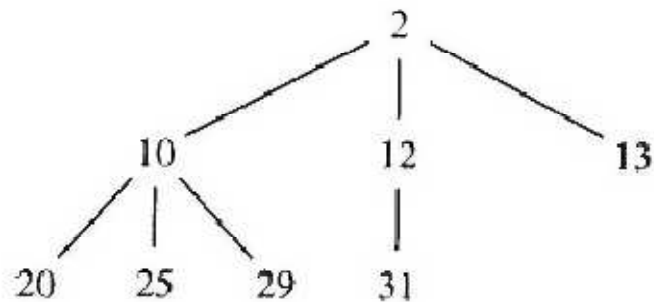
- A representação hierárquica possibilita acelerar o processo de pesquisa e esse é, geralmente, o principal enfoque quando trabalhamos com árvores como estruturas de dados.
- Considerando uma lista ordenada, a pesquisa tem sempre que iniciar no primeiro elemento e percorrer a lista completamente.
- Se os elementos são armazenados numa árvore ordenada, seguindo algum critério, o número de comparações pode ser consideravelmente reduzido mesmo para o pior caso.

# Construindo árvores

- É possível construir uma árvore a partir de uma lista.



(a)



(b)

- Quantas comparações são necessárias para encontrar o elemento 31?



# Árvore Binária – definição

---

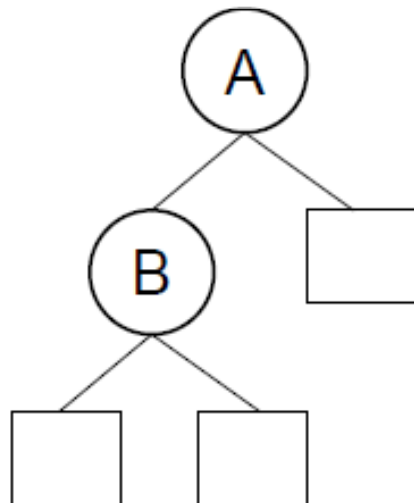
- Uma árvore binária  $T$  é um conjunto finito de nós com as seguintes propriedades:
  - ✓ O conjunto é vazio,  $T = \emptyset$ ; ou
  - ✓ O conjunto consiste em uma raiz,  $r$  e em exatamente duas subárvores binárias distintas  $T_L$  e  $T_R$

$$T = \{r ; T_L ; T_R\}$$

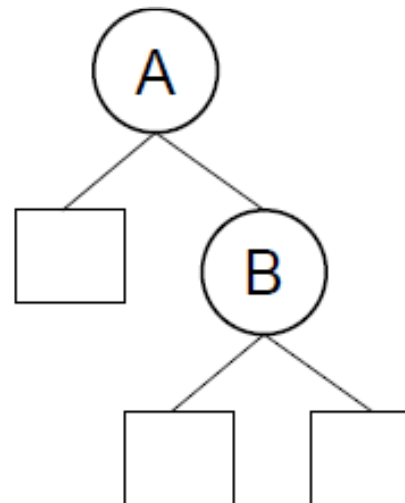


# Árvore Binária – definição

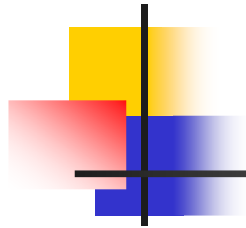
- $T_L$  é dita subárvore esquerda de  $T$  e
- $T_R$  é dita subárvore direita de  $T$ .



(a)



(b)



# Árvore Binária – definição

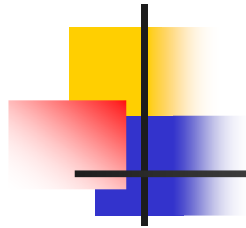
---

- **Número máximo de nós** de uma árvore binária de altura  $h$  maior ou igual a 0:

$$2^h - 1$$

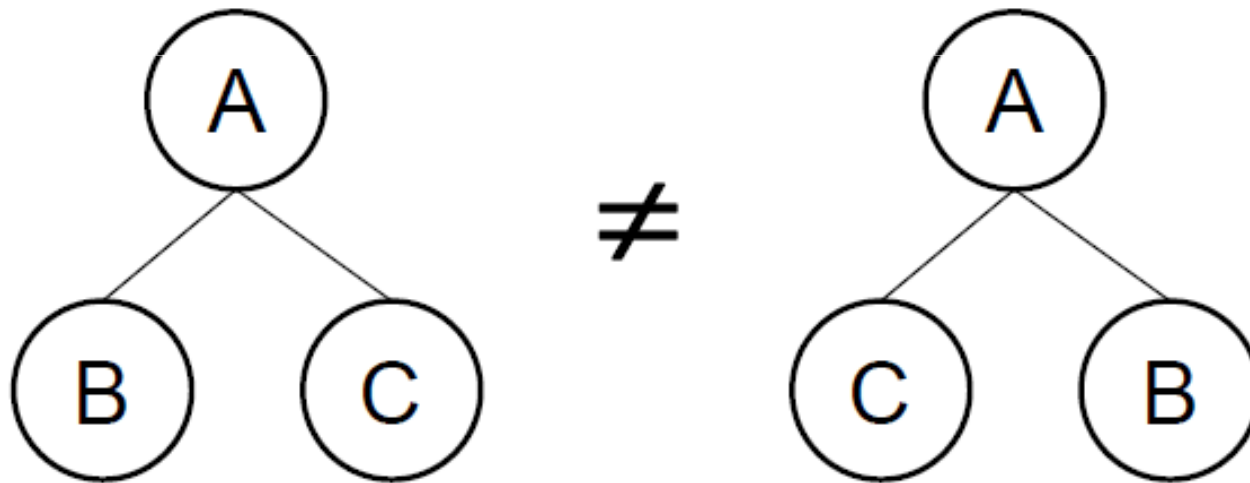
- **Número máximo de folhas:**

$$2^{h-1}$$



# Árvore Binária – definição

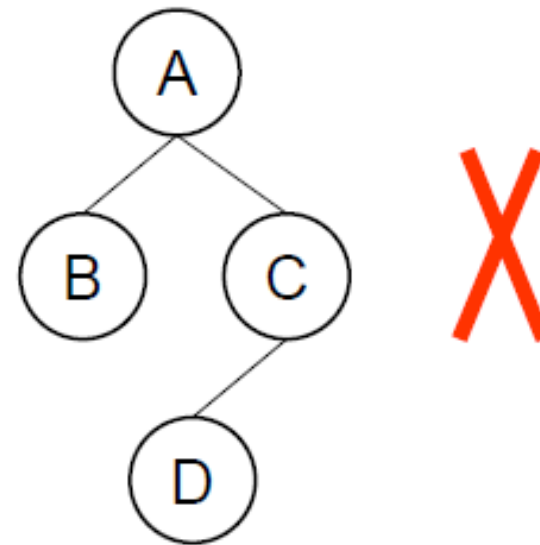
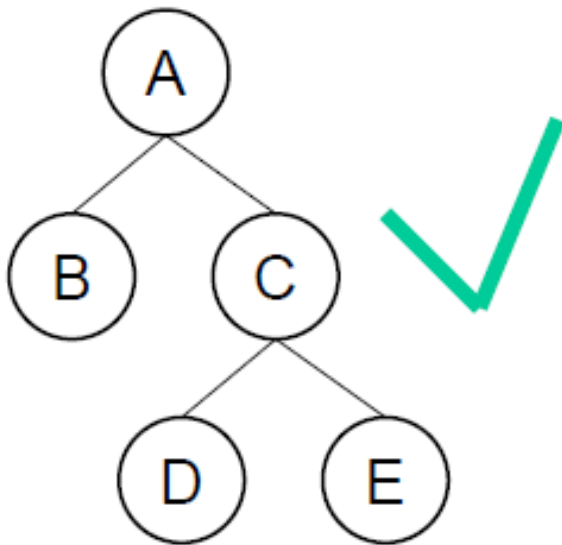
- **Árvore ordenada:** as subárvores estão ordenadas





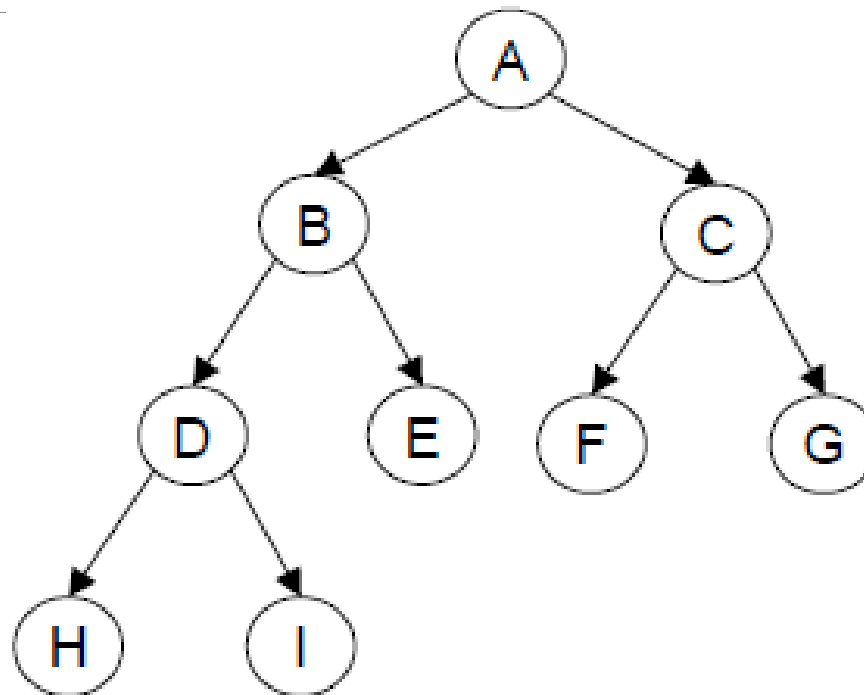
# Árvore Binária – definição

- **Árvore estritamente binária:** cada nó não terminal ou não tem filhos ou tem tanto a subárvore esquerda quanto a direita.



# Árvore Binária – definição

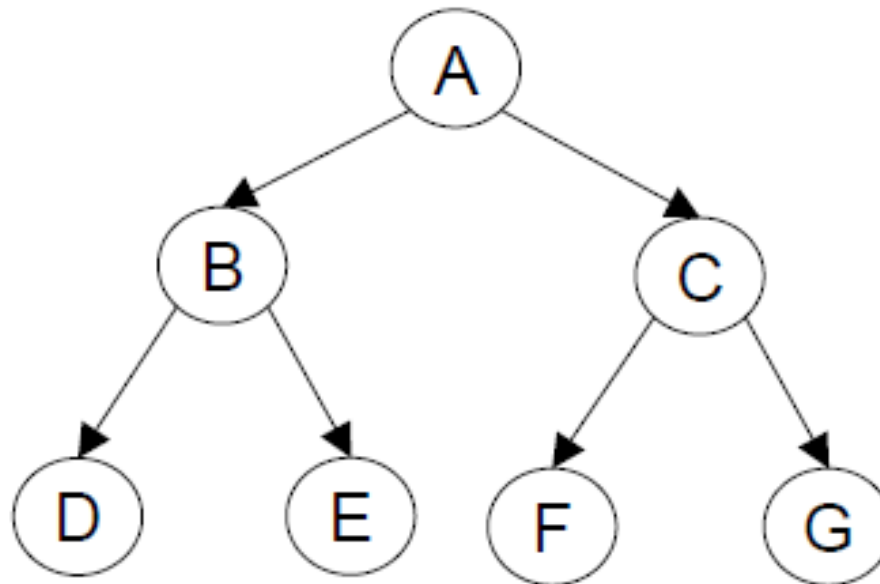
- **Árvore binária completa:** é uma árvore estritamente binária onde todos os nós folhas se encontram ou no **último** ou no **penúltimo** nível da árvore.





# Árvore Binária – definição

- **Árvore binária cheia:** todas as subárvores vazias são filhas de nós do último nível, isto é, uma árvore binária cheia é aquela cujos nós, exceto os do último nível, tem exatamente duas sub-árvores.

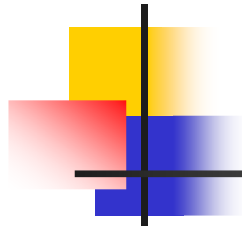




# Representação - Árvores Binárias

---

- Assim como as listas lineares, estruturas de dados do tipo árvore podem ser implementadas:
  - Através de vetores (*array*): representação estática
  - Através de ponteiros: representação dinâmica



# Representação - Árvores Binárias

---

- Uma árvore binária pode ser representada por uma estrutura de 3 campos elementares:
  - ✓ informação
  - ✓ subárvore esquerda
  - ✓ subárvore direita.



# Representação - Árvores Binárias

---

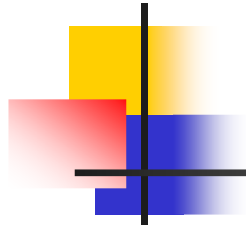
- Operações básicas sobre uma árvore binária:
  - ✓ Inicializar
  - ✓ Verificar se a árvore está vazia
  - ✓ Inserir nó a direita
  - ✓ Inserir nó a esquerda
  - ✓ Remover nó a direita
  - ✓ Remover nó a esquerda



# Representação - Árvores Binárias

---

- Outras operações
  - ✓ Função para calcular a altura da árvore
  - ✓ Função para calcular o número de nós da árvore
  - ✓ Função para calcular o nível de um nó
  - ✓ etc.



# Árvores Binárias - estática

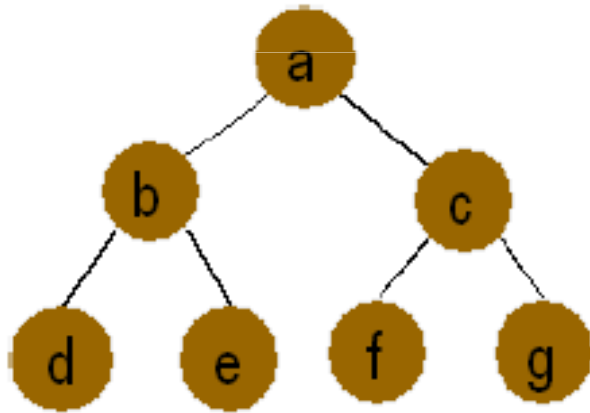
```
#define MAX 50
typedef struct sNo
{
    int info;
}NO;
```

```
typedef struct sArvore
{
    NO arv[MAX];
}Arvore;
```

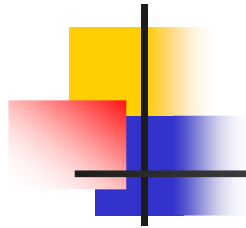


# Árvores Binárias - estática

- Armazenar os nós, por nível, em um array



0	1	2	3	4	5	6	...
<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>g</b>	...

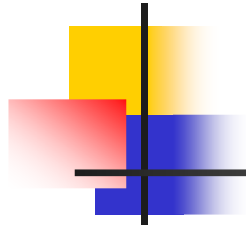


# Árvores Binárias - estática

---

- Para um vetor indexado a partir da posição 0, se um nó está na posição  $i$ , seus filhos diretos estão nas posições
  - ✓  $2i + 1$  : filho da esquerda
  - ✓  $2i + 2$  : filho da direita
- **Desvantagem:** espaços vagos se a árvore não é completa por níveis, ou se sofrer eliminação





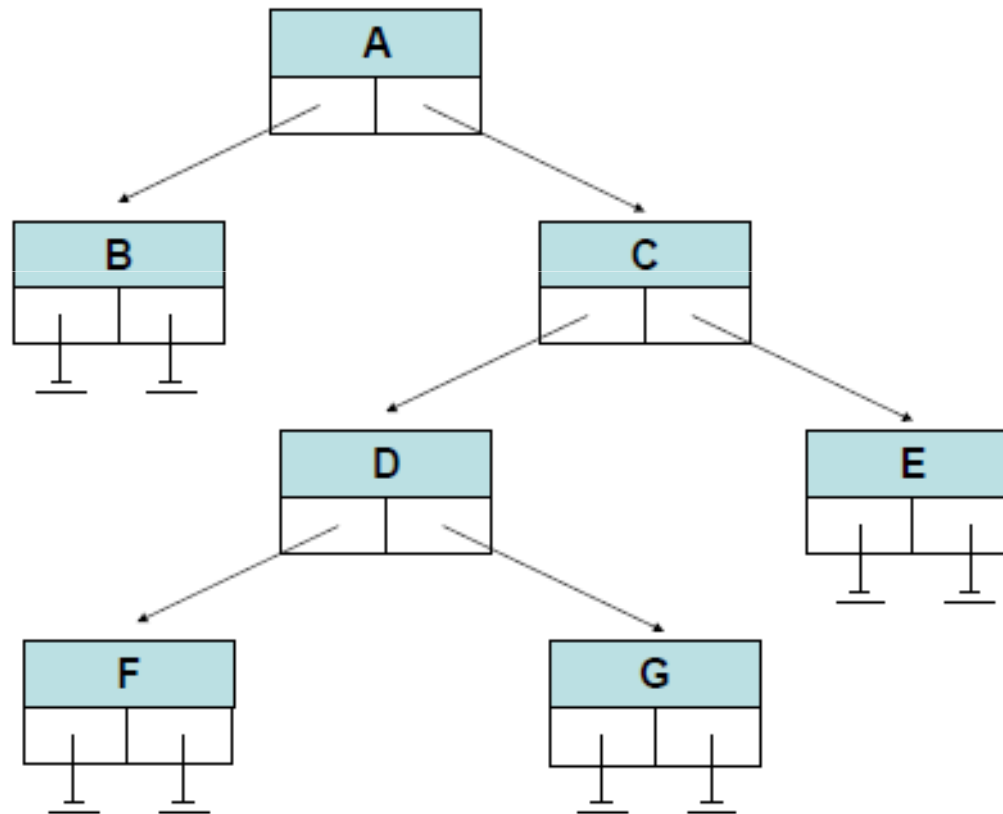
# Árvores Binárias - dinâmica

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

```
typedef struct sArvore
{
    NO* ptRaiz;
}Arvore;
```



# Árvores Binárias - dinâmica





# Árvores Binárias

## percurso/travessia

---

- Objetivo: Percorrer uma árvore 'visitando' cada nó uma única vez. Um percurso gera uma seqüência linear de nós. Com isso podemos então falar em 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, procurar um item, etc.



# Árvores Binárias

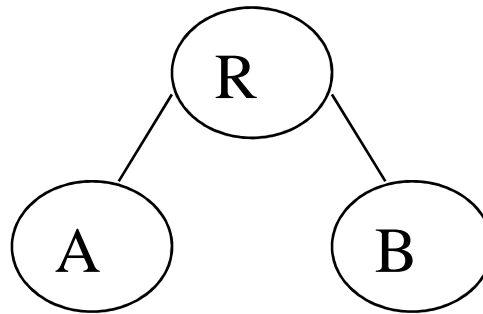
## percurso/travessia

---

- Três percursos são básicos:
  - \* pré-ordem (*pre-order*)
  - \* em-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;
  - ...

# Árvores Binárias percurso/travessia

- Exemplo:

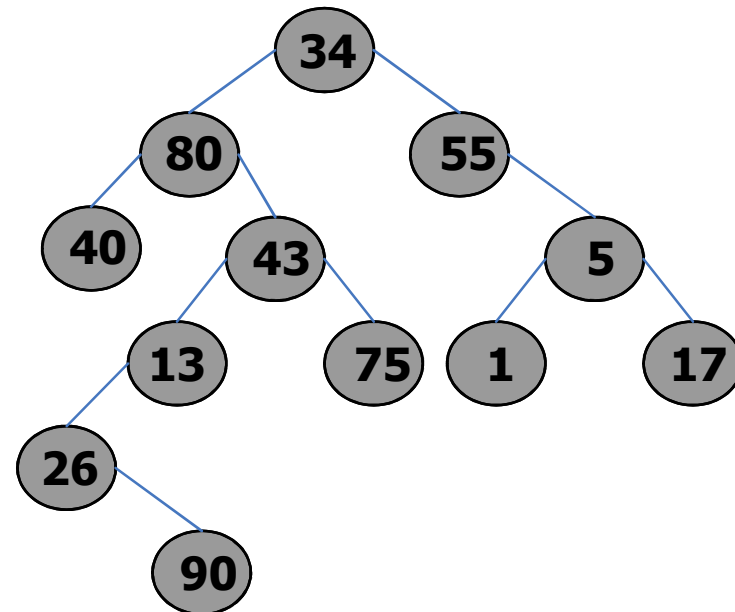


- 1) Pré-Ordem (Raiz – Esquerda – Direita): R – A – B
- 2) Em-Ordem (Esquerda – Raiz – Direita): A – R – B
- 3) Pós-Ordem (Esquerda – Direita – Raiz): A – B – R

# Percurso em Árvore Pré-Ordem

- 1- Visitar a raiz da árvore
- 2- Visitar recursivamente sua subárvore esquerda
- 3- Visitar recursivamente sua subárvore direita

→ Exiba o percurso em pré-ordem para a árvore ao lado



**Resultado:** 34, 80, 40, 43, 13, 26, 90, 75, 55, 5, 1, 17.



# Percurso em Árvore Pré-Ordem

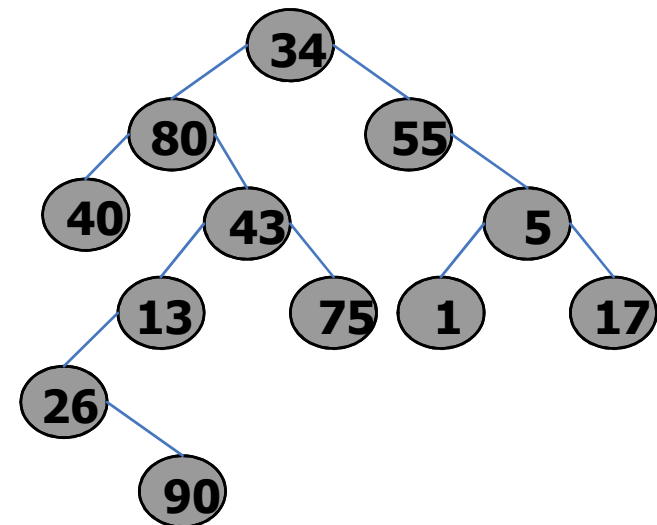
→ Implementação

```
void pre_ordem(No* raiz)
{
    if (raiz != NULL)
    {
        visita(raiz);
        pre_ordem(raiz->esq);
        pre_ordem(raiz->dir);
    }
}
```

# Percurso em Árvore Em-Ordem

- 1- Visitar recursivamente a subárvore esquerda
- 2- Visitar a raiz da árvore
- 3- Visitar recursivamente a subárvore direita

→ Exiba o percurso em em-ordem para a árvore ao lado



**Resultado:** 40, 80, 26, 90, 13, 43, 75, 34, 55, 1, 5, 17.





# Percurso em Árvore Em-Ordem

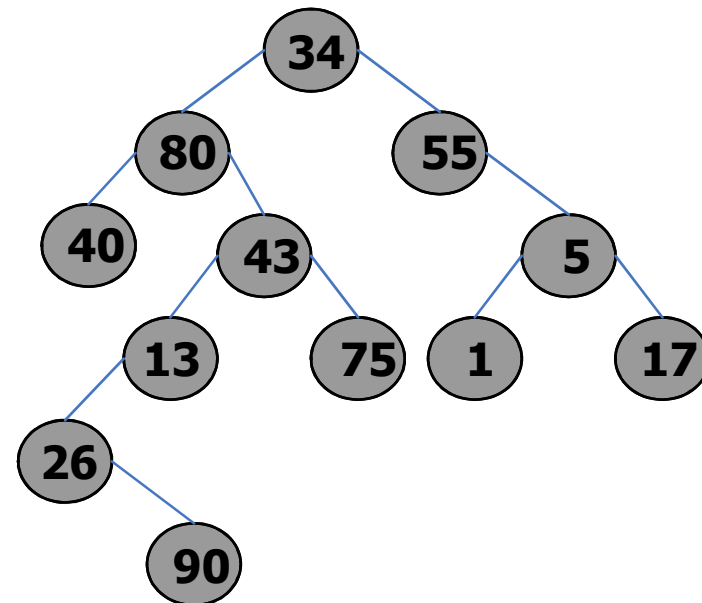
→ Implementação

```
void em_ordem(No* raiz)
{
    if (raiz != NULL)
    {
        em_ordem(raiz->esq) ;
        visita(raiz) ;
        em_ordem(raiz->dir) ;
    }
}
```

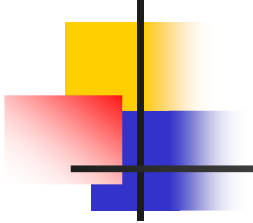
# Percurso em Árvore Pós-Ordem

- 1- Visitar recursivamente a subárvore esquerda
- 2- Visitar recursivamente a subárvore direita
- 3- Visitar a raiz da árvore

→ Exiba o percurso em pós-ordem para a árvore ao lado



**Resultado:** 40, 90, 26, 13, 75, 43, 80, 1, 17, 5, 55, 34.



# Percurso em Árvore Pós-Ordem

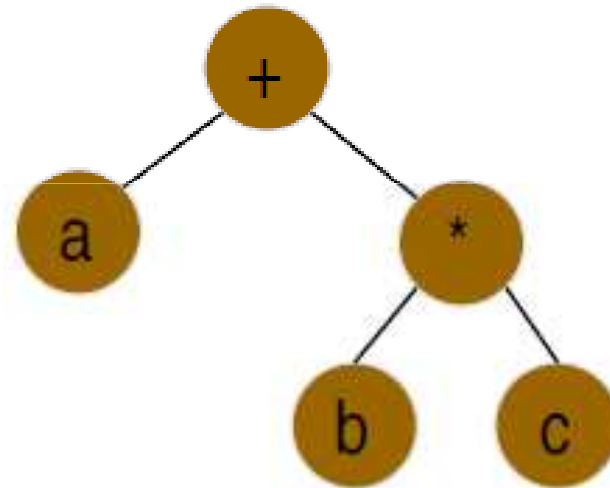
---

→ Implementação

```
void pos_ordem(No* raiz)
{
    if (raiz != NULL)
    {
        pos_ordem(raiz->esq) ;
        pos_ordem(raiz->dir) ;
        visita(raiz) ;
    }
}
```

# Percurso em Árvore

→ Percurso para expressões aritméticas



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



# Árvores Binárias de Pesquisa

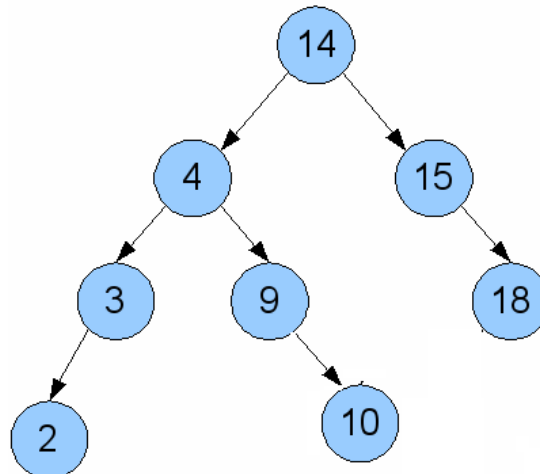
---

- A operação de busca em uma estrutura visa a descobrir se o elemento pertence à estrutura ou não.
- Numa árvore, é preciso percorrê-la até encontrar o elemento, se existir.
- Organizar os elementos numa árvore binária permite facilitar a busca.
  - Exemplo: dados um nó  $t$ , seu filho à esquerda  $x$  e à direita  $y$ , elementos estão arranjados tal que:  $x \leq t \leq y$

# Árvores Binárias de Pesquisa

→ Propriedades das Árvores Binárias de Pesquisa

- i) todos os itens da subárvore esquerda são menores do que a raiz.
- ii) todos os itens da subárvore direita são maiores ou iguais à raiz.
- iii) cada subárvore é também uma árvore binária de busca.





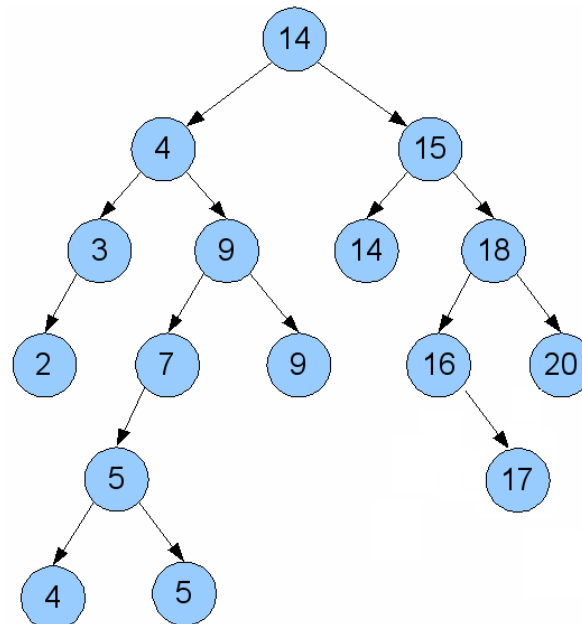
# Árvores Binárias de Pesquisa

---

- O operador  $\leq$  ou  $<$  deve ser sobrecarregado caso seja necessário processar alguma chave não numérica ou de conteúdo diferente.
- A ordem alfabética é geralmente usada quando em chaves do tipo "string" (conjunto de caracteres na linguagem C).

# Árvores Binárias de Pesquisa

- Construindo uma Árvore Binária de Pesquisa  
→ Como ficaria uma árvore binária de pesquisa para a seguinte sequência de elementos?  
{14; 15; 4; 9; 7; 18; 3; 5; 16; 14; 20; 17; 9; 2; 5; 4}







# Exercícios

---

- 1) Desenvolva usando a linguagem C funções recursivas para:
  - a) Calcular o número de nós da árvore binária de pesquisa
  - b) Calcular o número de folhas de uma árvore binária
  - c) Calcular a altura de uma árvore binária de pesquisa