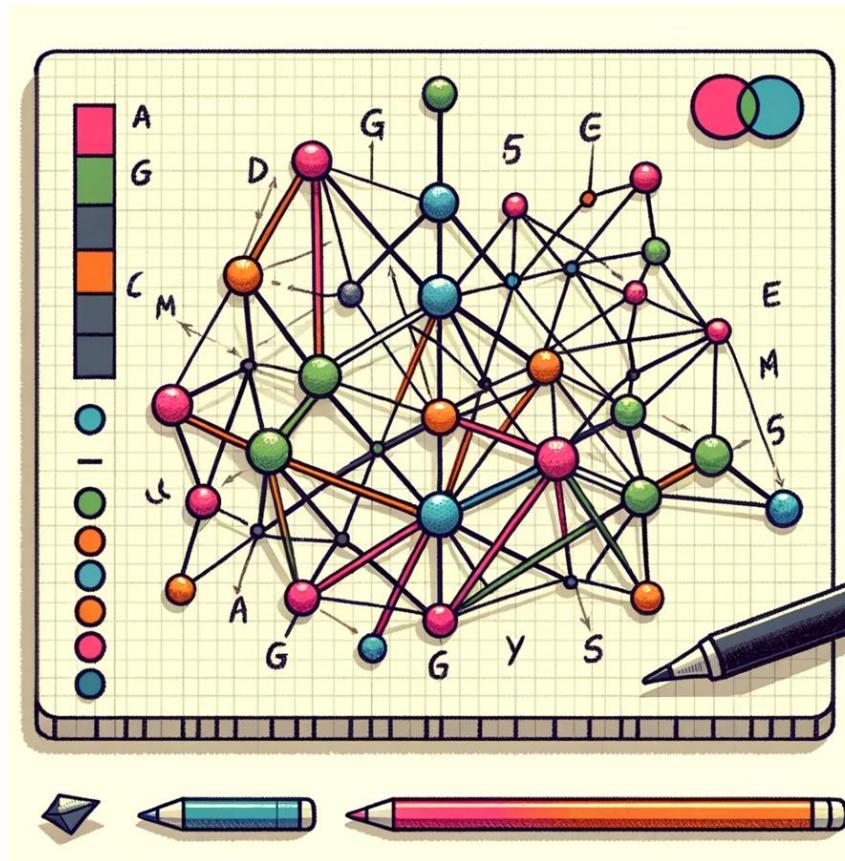


# Teoria dos Grafos

Jairo Lucas



# Teoria dos Grafos - Propriedades grafos e árvores

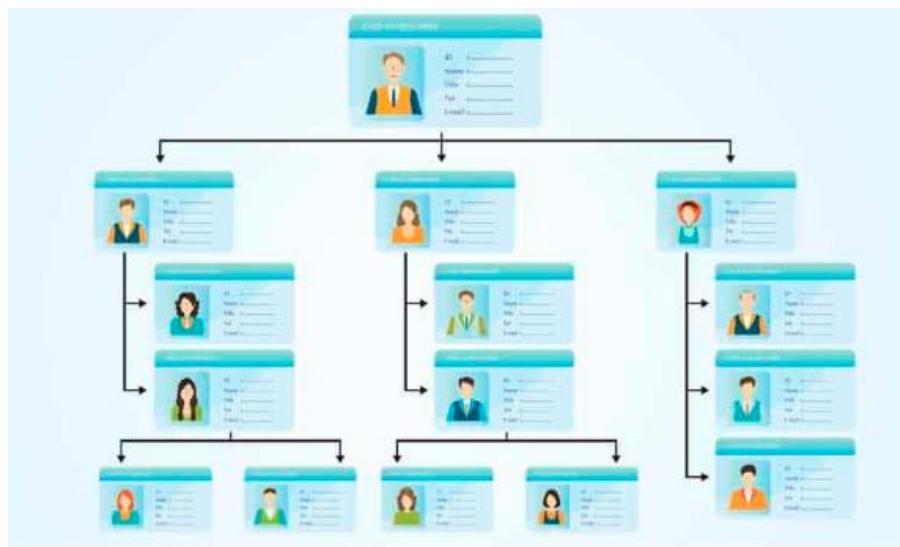
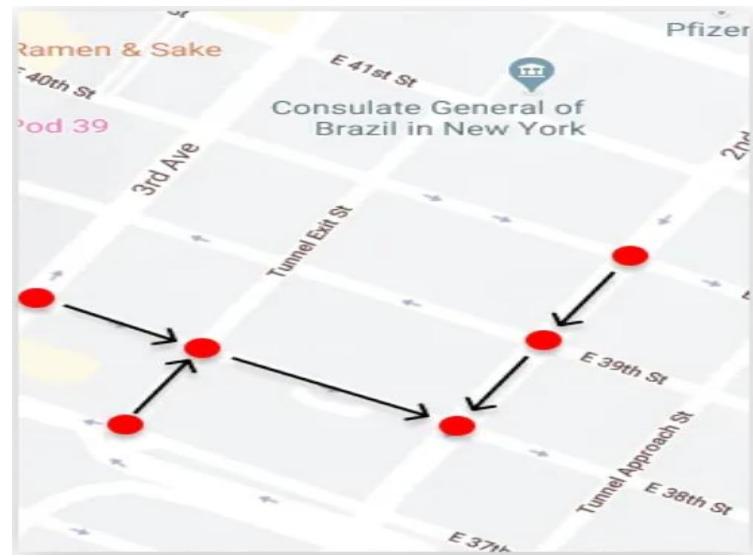
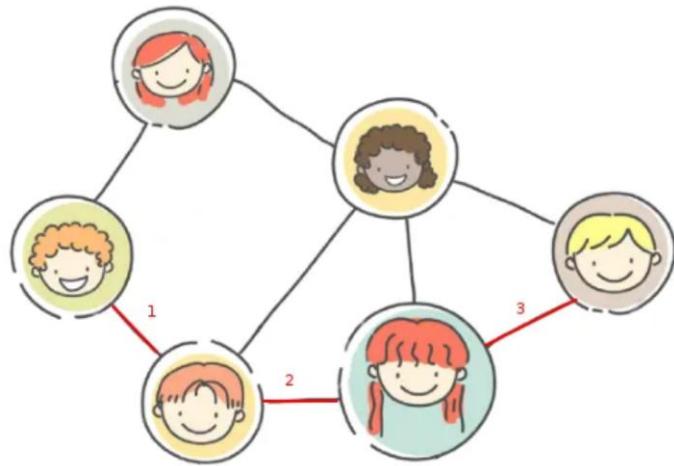
## Árvores

Uma árvore é um grafo conexo ( $G$ ), **sem arestas paralelas**, dotado da seguinte propriedade: para cada aresta  $a$ , o grafo  $G - a$  não é conexo.

Ou de forma mais simples:

**Uma árvore é um grafo conexo que não possui circuitos.**

# Teoria dos Grafos - Propriedades grafos e árvores



# Teoria dos Grafos - Propriedades grafos e árvores

- Principais conceitos:

- Um grafo  $G$  é uma árvore se, e somente se, existir um e **apenas um caminho** entre cada par de vértices.
  - Se existem dois caminhos distintos entre um par de vértices então a união destes caminhos contém **um circuito**. Logo, o grafo não seria uma árvore

# Teoria dos Grafos - Propriedades grafos e árvores

## ■ Principais conceitos:

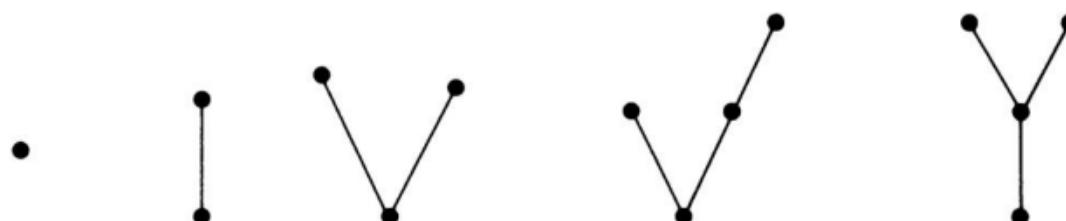
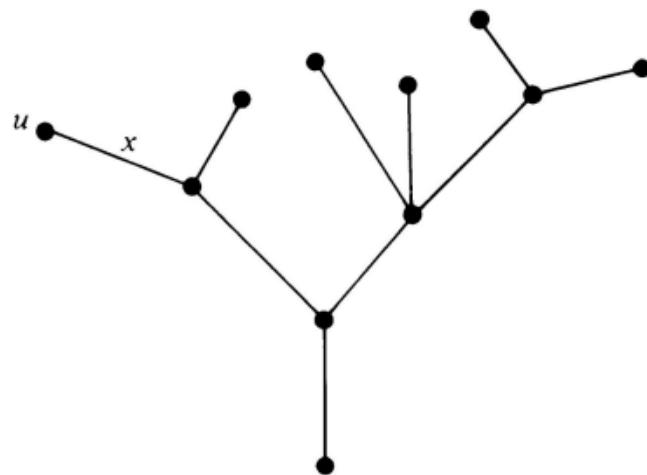
Seja  $G$  uma árvore com  $n$  vértices. As seguintes afirmações são equivalentes:

- **$G$  é conexo e possui  $n - 1$  arestas.**
- $G$  possui  $n - 1$  arestas e não possui circuitos.
- Existe exatamente **um único caminho** entre cada par de vértices.

- Se tivéssemos menos de  $n-1$  arestas, o grafo não poderia ser conexo, pois não haveria arestas suficientes para conectar todos os vértices, deixando pelo menos dois vértices em componentes separados.
- Se tivéssemos mais de  $n-1$  arestas: A adição de uma aresta além do mínimo necessário para manter a conexidade (ou seja, mais de  $n-1$  arestas) cria um caminho alternativo entre dois vértices, formando assim um ciclo

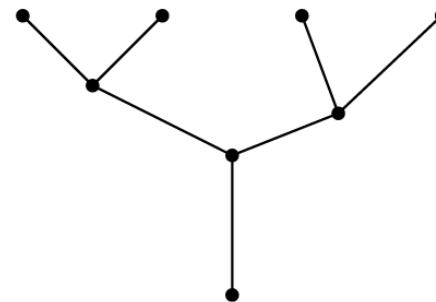
# Teoria dos Grafos - Propriedades grafos e árvores

Toda árvore é um grafo, mas nem todo grafo é uma árvore!!!

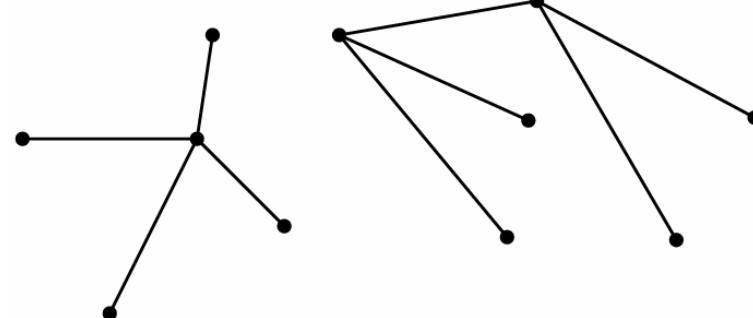


# Teoria dos Grafos - Propriedades grafos e árvores

- Uma **FLORESTA** é um grafo **desconexo** formado por **várias árvores**.
  - ATENÇÃO:** Não confundir os conceitos de árvores e florestas.
    - Árvore sempre vai ser um grafo **CONEXO**
    - Floresta, são dois ou mais grafos do tipo árvores disjuntos, ou seja, floresta sempre será um grafo .



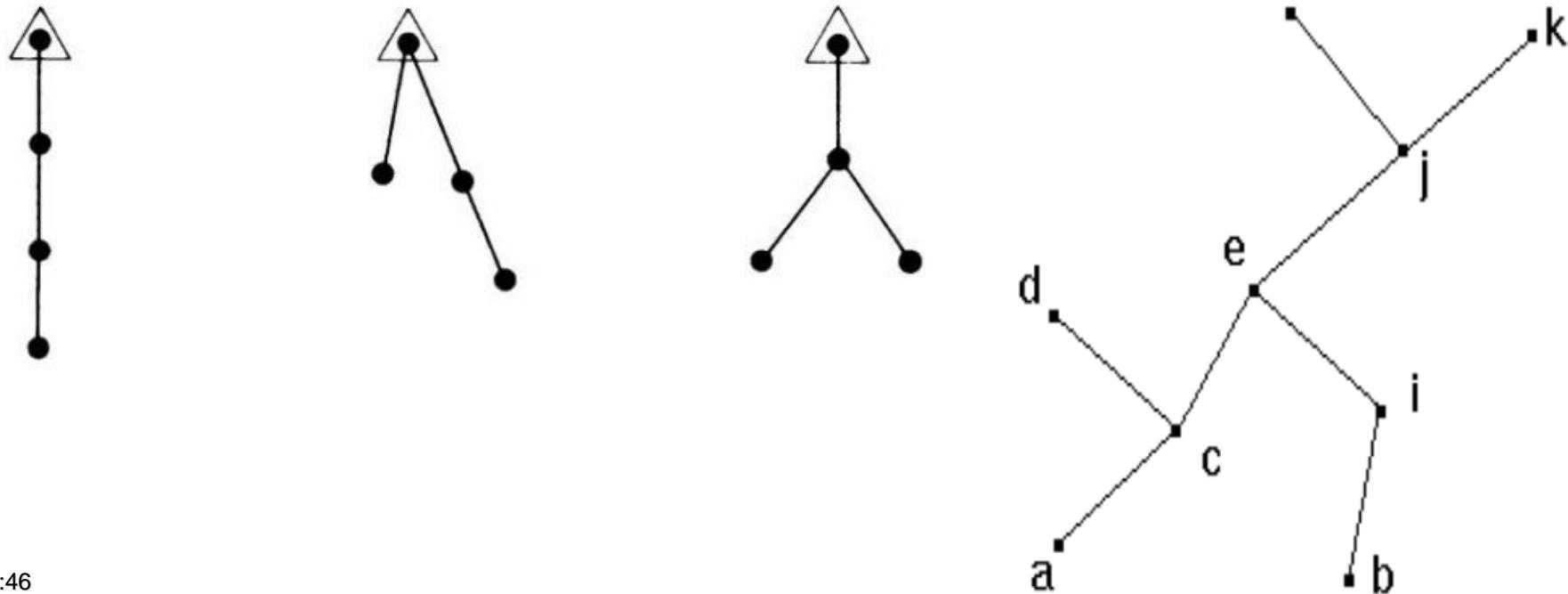
Uma **árvore**.



Uma **floresta**.

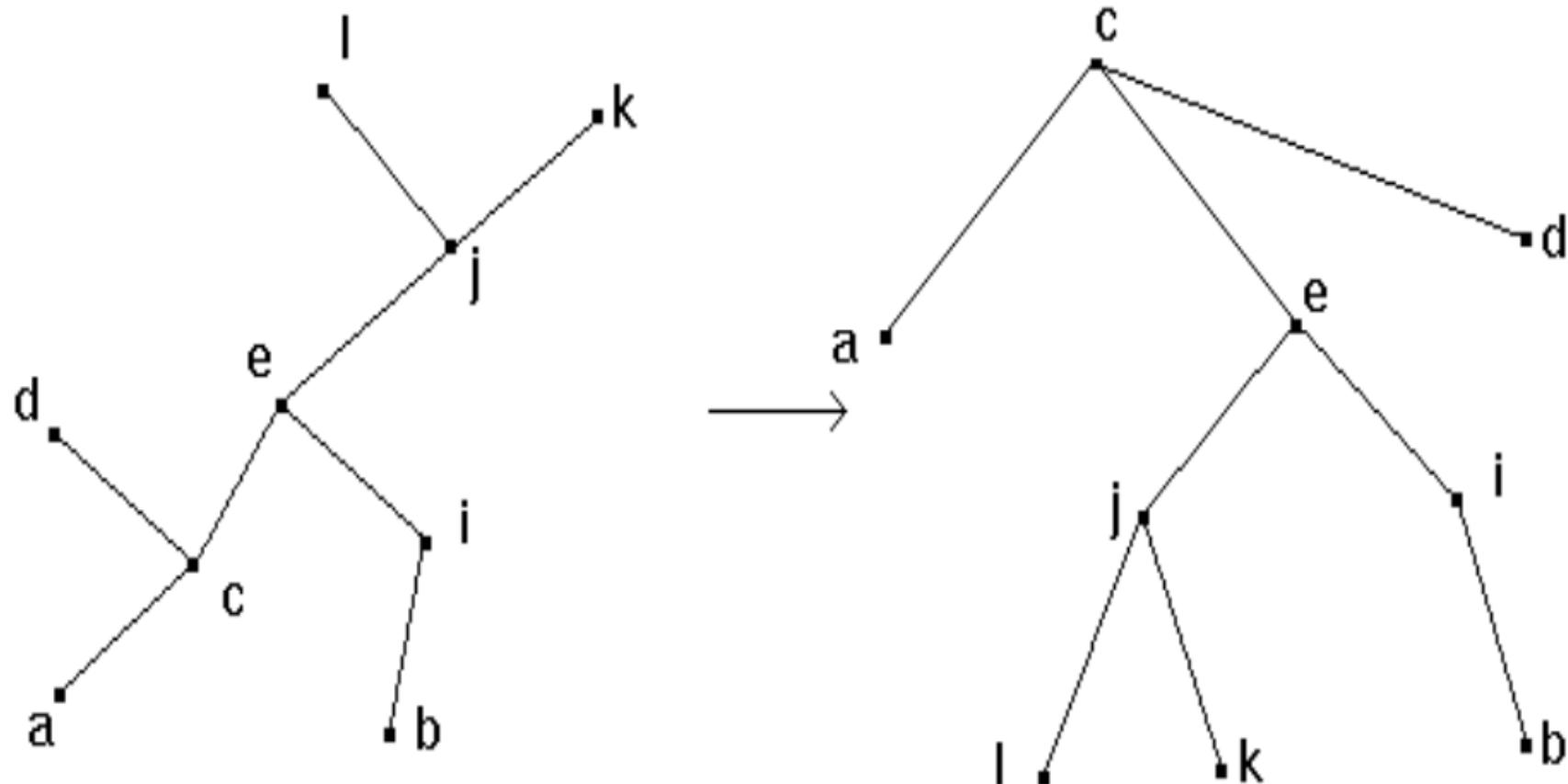
# Teoria dos Grafos - Propriedades grafos e árvores

- Uma árvore na qual definimos um determinado vértice, como **vértice raiz**, é chamada de **árvore enraizada**. Uma árvore **não enraizada** é chamada de **árvore livre**.



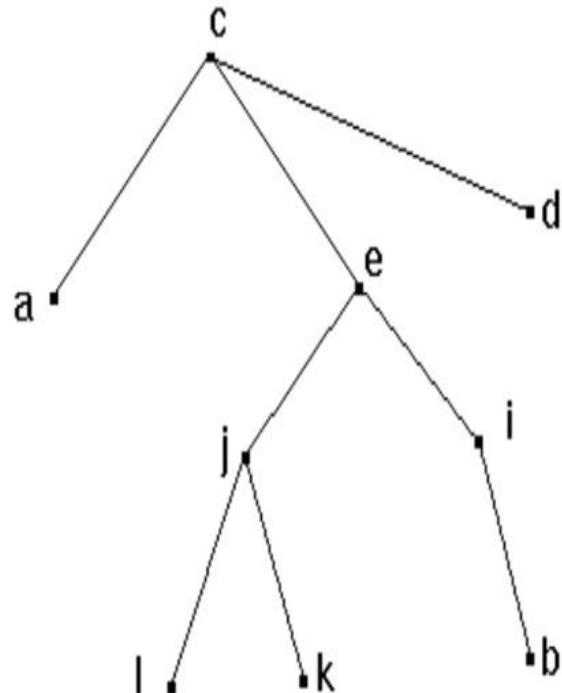
# Teoria dos Grafos - Propriedades grafos e árvores

- Podemos transformar uma árvore **sem raiz** em uma **árvore enraizada** simplesmente escolhendo um vértice como raiz e redesenhando a árvore.



# Teoria dos Grafos - Propriedades grafos e árvores

- Em uma árvore enraizada, com o vértice raiz posicionado na parte superior da figura, podemos definir níveis na árvore.



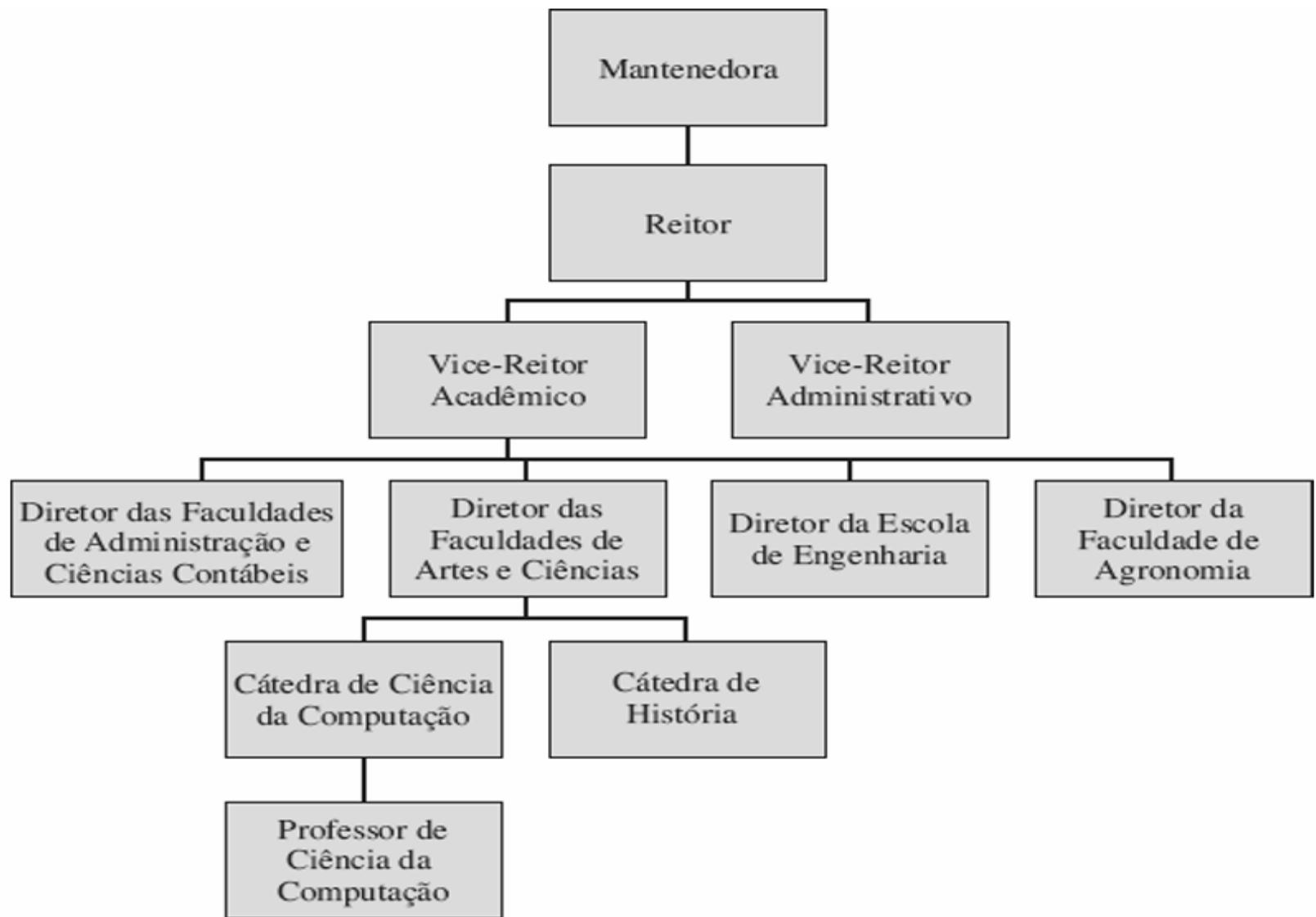
- A **profundidade** de um nó é o comprimento do caminho da raiz até este nó.
- A Raiz tem profundidade **ZERO**.
- A **profundidade (ou altura)** de um árvore é a profundidade até o nó mais distante.
- Um nó sem filhos é denominado de **FOLHA**.
- Todos os nós que NÃO são folhas, são denominados **NÓS INTERNOS**.

# Teoria dos Grafos - Propriedades grafos e árvores

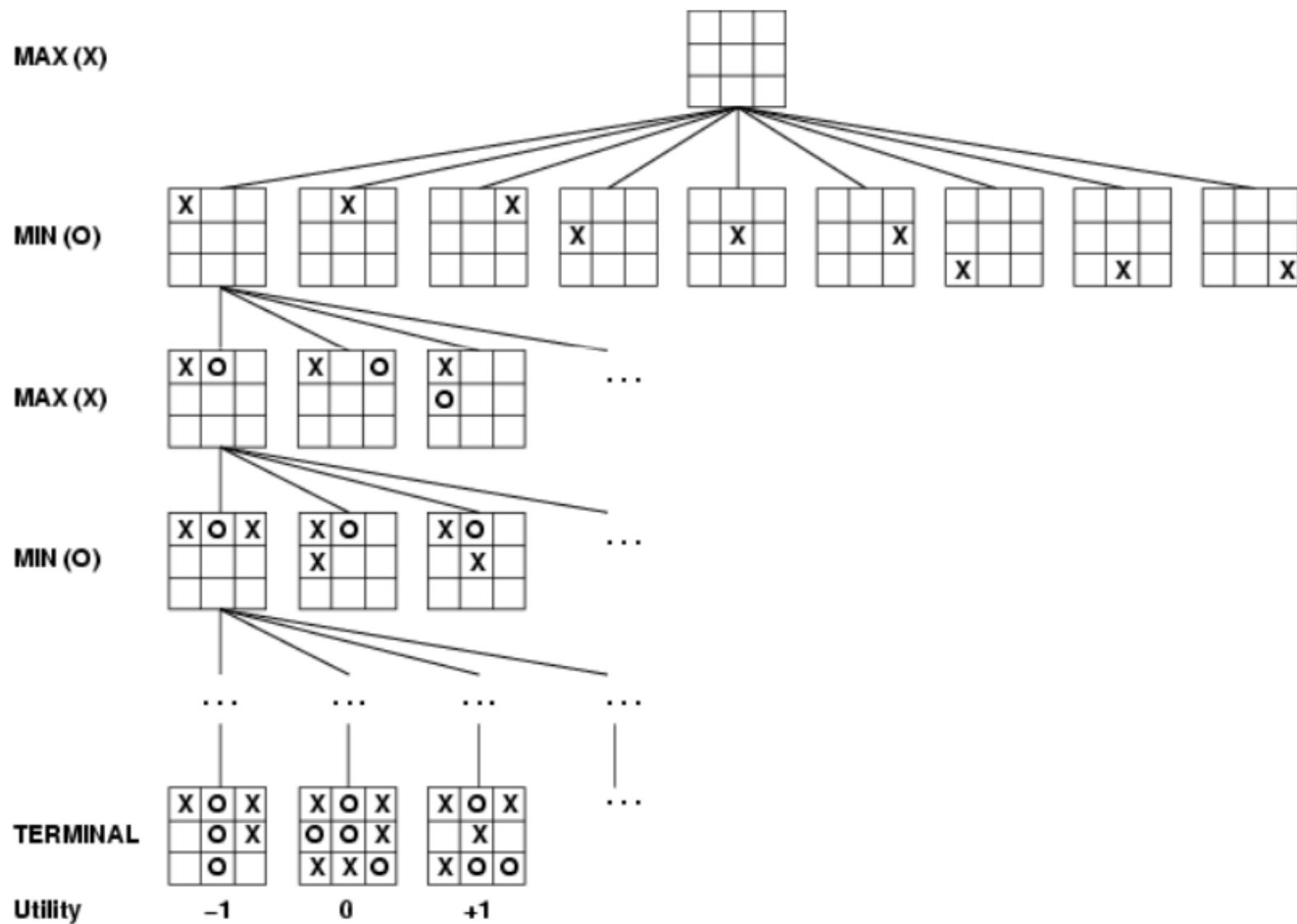
- Aplicações:



# Teoria dos Grafos - Propriedades grafos e árvores



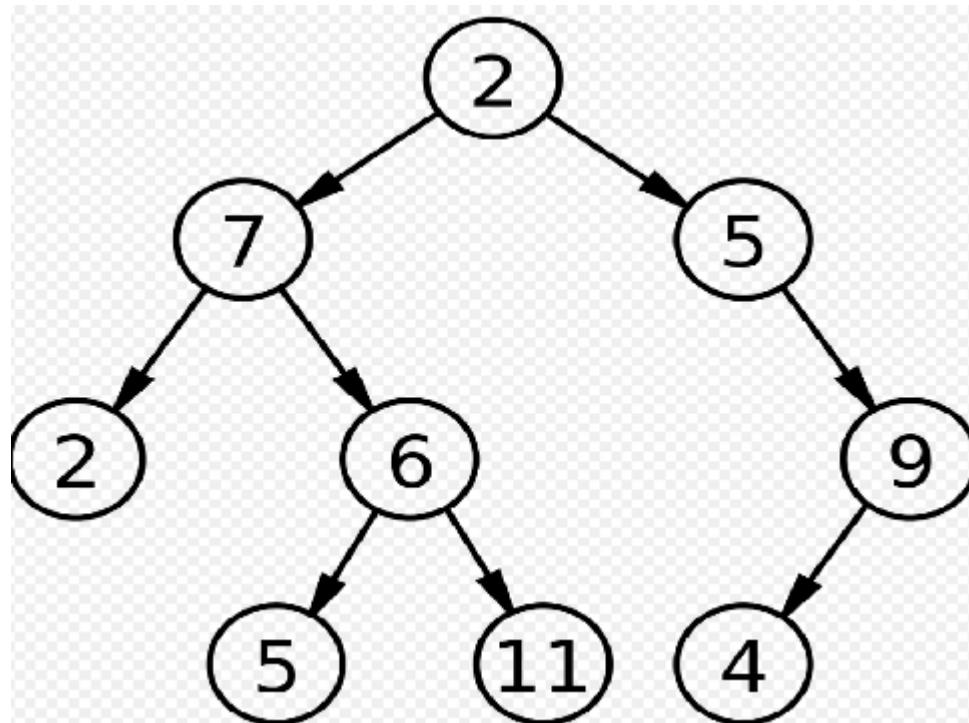
# Teoria dos Grafos - Propriedades grafos e árvores



# Teoria dos Grafos - Propriedades grafos e árvores

## Árvores Binárias

- São um tipo especial de árvore onde cada nó possui no máximo 2 (dois) filhos, chamados filho da esquerda e filho da direita.

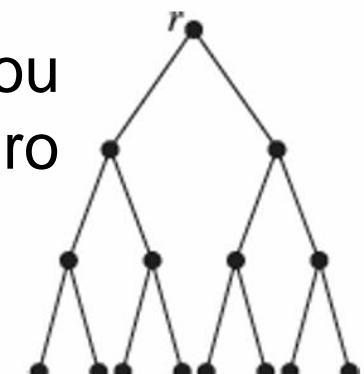


# Teoria dos Grafos - Propriedades grafos e árvores

## Árvores Binárias

- Árvore binária **CHEIA** possui TODOS os graus internos com **dois filhos** e todas as folhas estão na mesma **profundidade**.
  - Uma árvore binária **CHEIA** (ou estritamente binária) possui **SEMPRE** um número IMPAR de vértices.
  - O número **mínimo de vértices** necessários para criar uma árvore binária cheia pela formula abaixo:
    - O número **máximo de vértices é igual**, ou seja, uma árvore cheia tem sempre o número de vértices dados pela formula acima.

$$N = 2^{h+1} - 1$$



Árvore binária cheia

# Teoria dos Grafos - Propriedades grafos e árvores

## Árvores Binárias

- Uma **árvore binária completa** é uma árvore binária em que todos os níveis, exceto possivelmente o último, estão completamente preenchidos, e todos os nós estão o mais à esquerda possível.
  - O número **mínimo de vértices** necessários para criar uma árvore binária completa com altura  $h$ , pode ser calculado da seguinte forma:

Todos os níveis, exceto o último, estão completamente preenchidos. Ou seja, até a penúltimo nível teríamos uma árvore cheia, o que nos daria:  $N = 2^{h+1} - 1$  vértices (**considerar a altura até o penúltimo nível**)

A árvore completa teria **no mínimo** mais um vértice no último nível, portanto, a quantidade mínima de vértices seria:

$$N = 2^h$$



Árvore binária completa

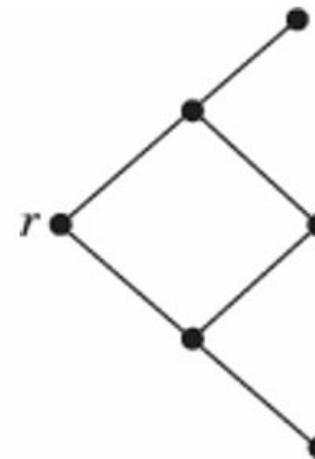
# Teoria dos Grafos - Propriedades grafos e árvores

Quais dos grafos a seguir são árvores com raiz  $r$ ? Se o grafo for uma árvore, desenhe-o da maneira mais convencional. Se não for, diga qual a propriedade que falha.

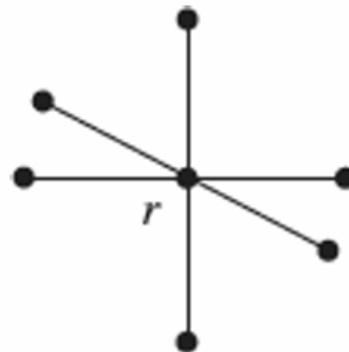
a.



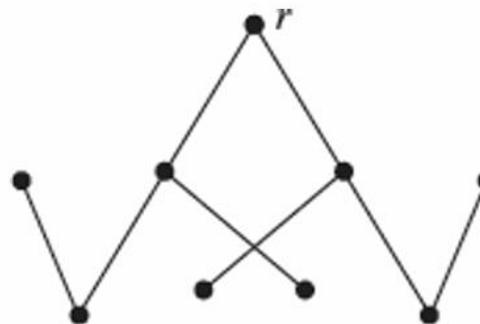
b.



C.



d.



# Teoria dos Grafos - Propriedades grafos e árvores

Quais dos grafos a seguir são árvores com raiz  $r$ ? Se o grafo for uma árvore, desenhe-o da maneira mais convencional. Se não for, diga qual a propriedade que falha.

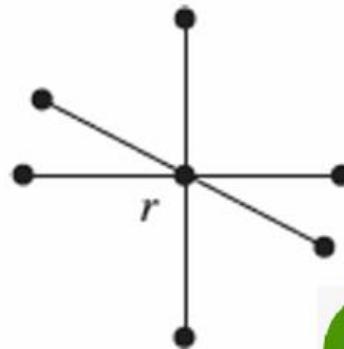
a.



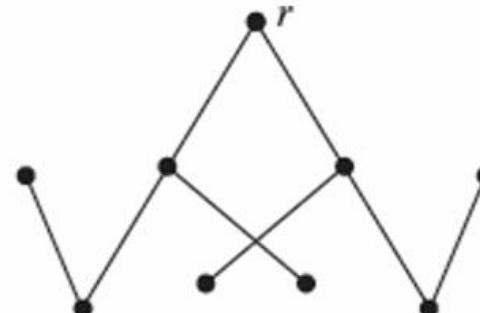
b.



c.



d.



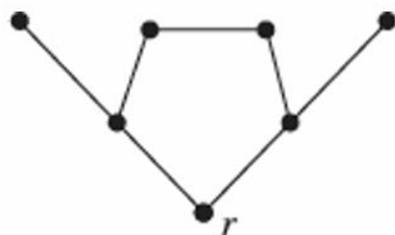
# Teoria dos Grafos - Propriedades grafos e árvores

Quais dos grafos a seguir são árvores binárias com raiz  $r$ ? Se o grafo não for uma árvore binária, diga qual a propriedade que falha.

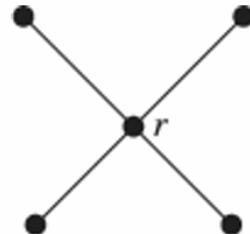
a.



b.



c.



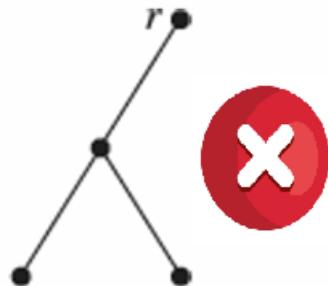
d.



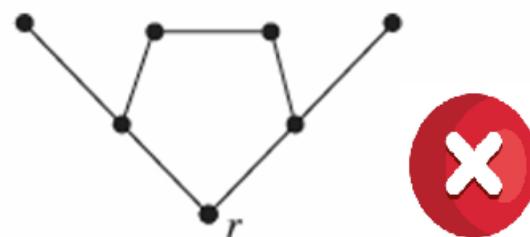
# Teoria dos Grafos - Propriedades grafos e árvores

Quais dos grafos a seguir são árvores binárias com raiz  $r$ ? Se o grafo não for uma árvore binária, diga qual a propriedade que falha.

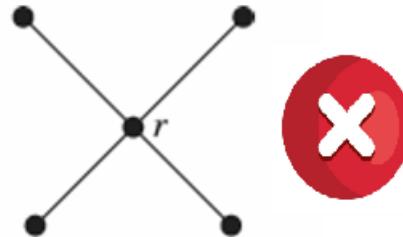
a.



b.



c.



d.



# Teoria dos Grafos - Propriedades grafos e árvores

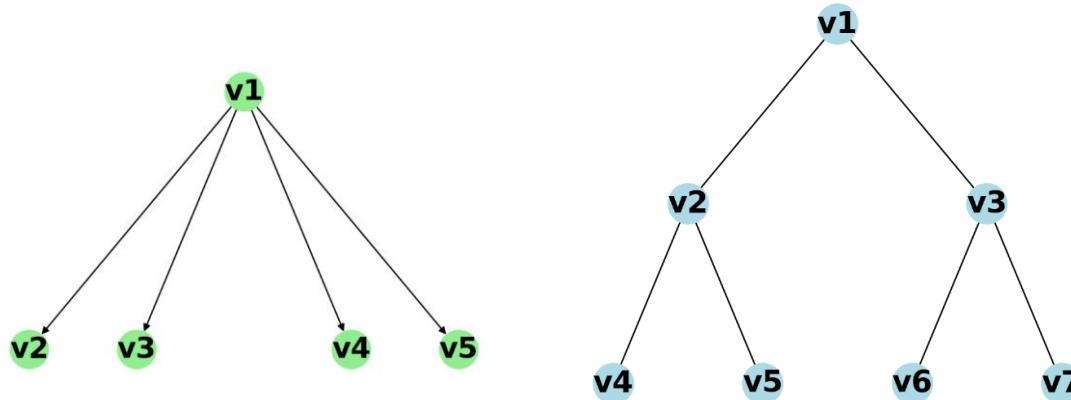
Esboce uma figura para cada uma das seguintes árvores:

- a. Árvore com cinco nós e altura 1.
- b. Árvore binária cheia de altura 2.
- c. Árvore de altura 3 em que cada nó de profundidade  $i$  tem  $i + 1$  filhos.

# Teoria dos Grafos - Propriedades grafos e árvores

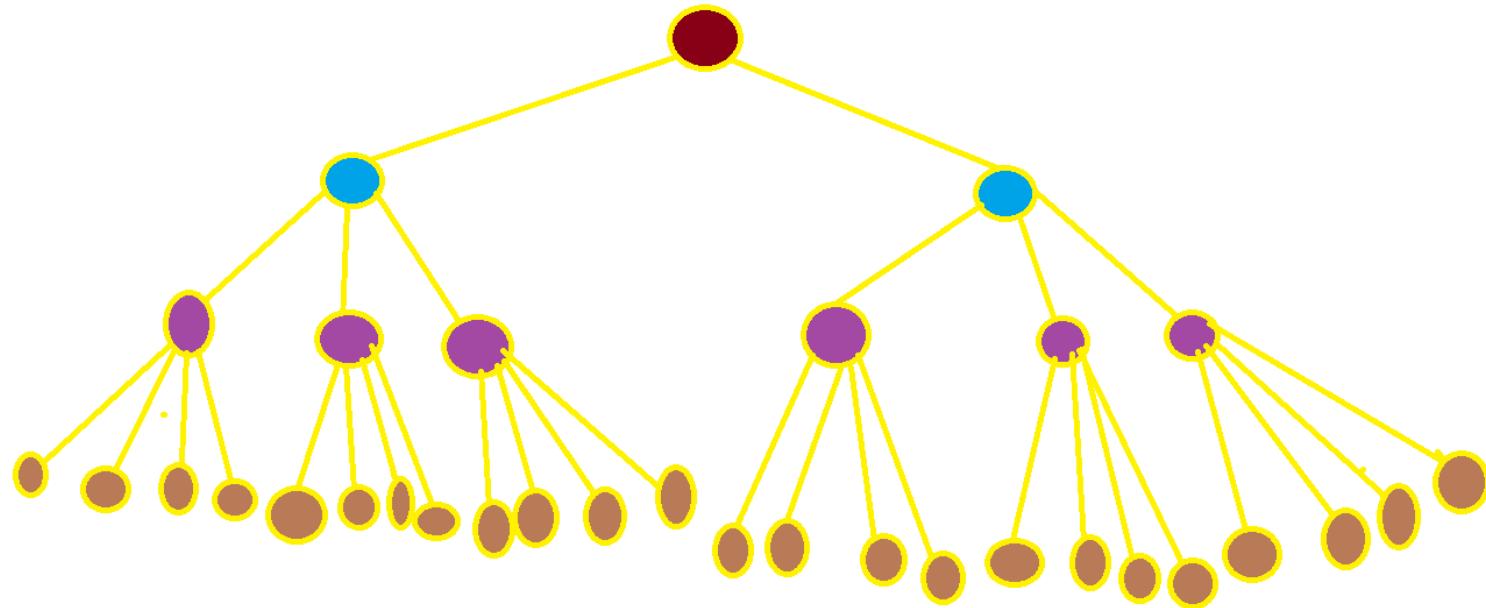
Esboce uma figura para cada uma das seguintes árvores:

- Árvore com cinco nós e altura 1.
- Árvore binária cheia de altura 2.
- Árvore de altura 3 em que cada nó de profundidade  $i$  tem  $i + 1$  filhos.

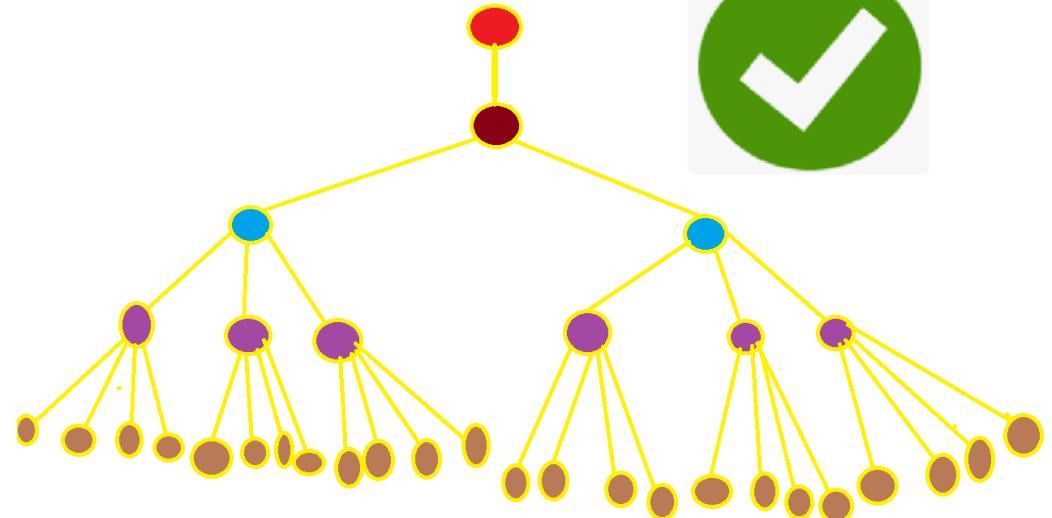
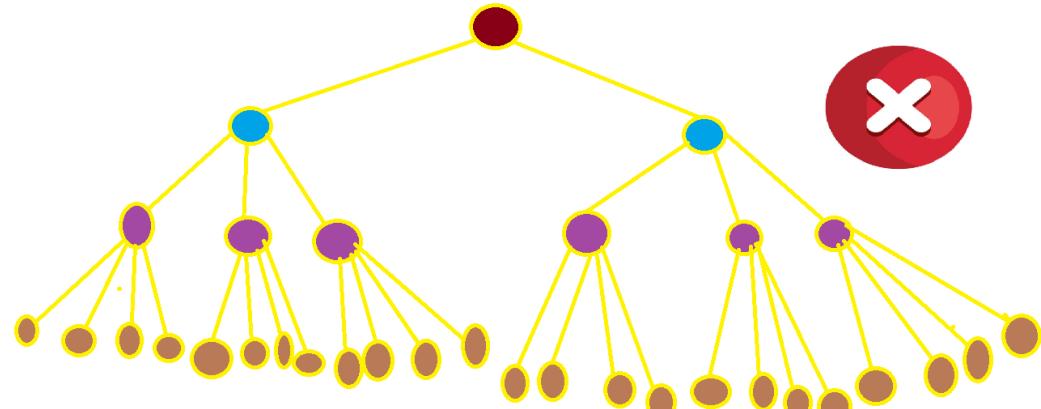


$$N = 2^{h+1} - 1$$

# Teoria dos Grafos - Propriedades grafos e árvores



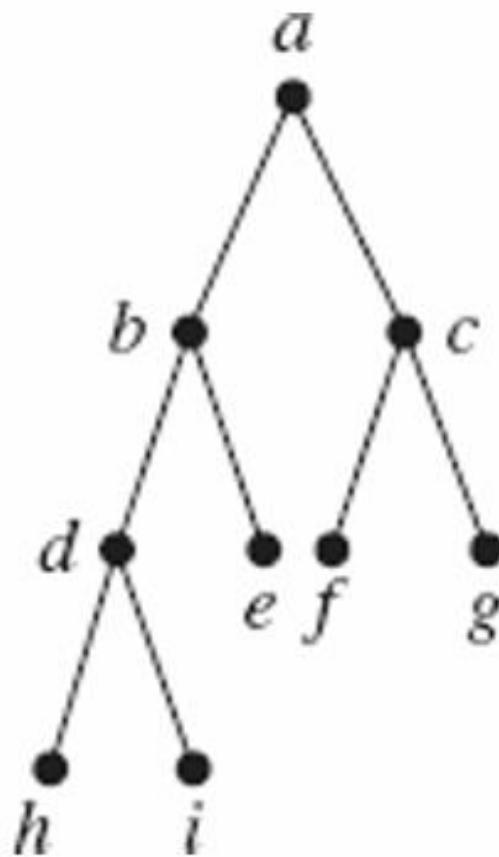
# Teoria dos Grafos - Propriedades grafos e árvores



# Teoria dos Grafos - Propriedades grafos e árvores

Responda as perguntas a seguir sobre o grafo na figura correspondente com raiz  $a$ .

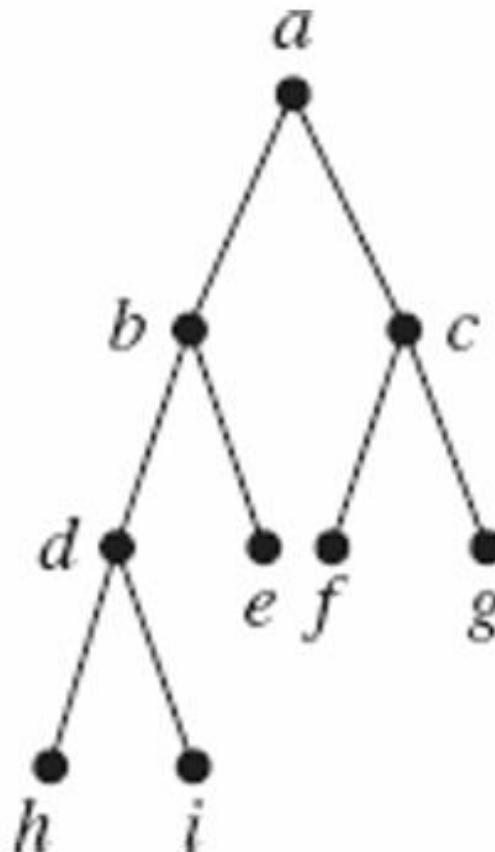
- a. Essa árvore é binária?
- b. É uma árvore binária cheia?
- c. É uma árvore binária completa?
- d. Qual nó é pai de  $e$ ?
- e. Qual nó é o filho direito de  $e$ ?
- f. Qual é a profundidade do nó  $g$ ?
- g. Qual é a altura da árvore?



# Teoria dos Grafos - Propriedades grafos e árvores

Responda as perguntas a seguir sobre o grafo na figura correspondente com raiz *a*.

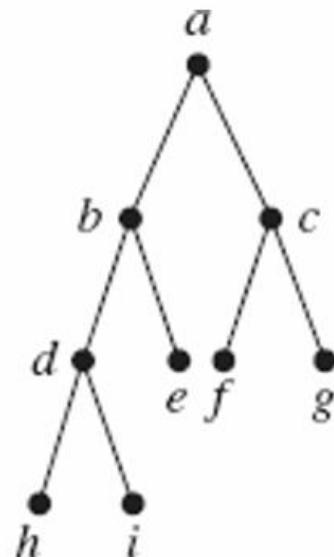
- a. Essa árvore é binária? 
- b. É uma árvore binária cheia? 
- c. É uma árvore binária completa? 
- d. Qual nó é pai de *e*? **b**
- e. Qual nó é o filho direito de *e*? **Não tem**
- f. Qual é a profundidade do nó *g*? **2**
- g. Qual é a altura da árvore? **3**



# Teoria dos Grafos – Percurso em árvores binárias

## ■ **Percursos (caminhamento em árvores):**

- Um caminhamento ou percurso em uma árvore binária é uma forma de visitar todos os nós da árvore seguindo uma ordem específica.
- Existem várias estratégias para percorrer uma árvore binária, cada uma com suas próprias características e aplicações.

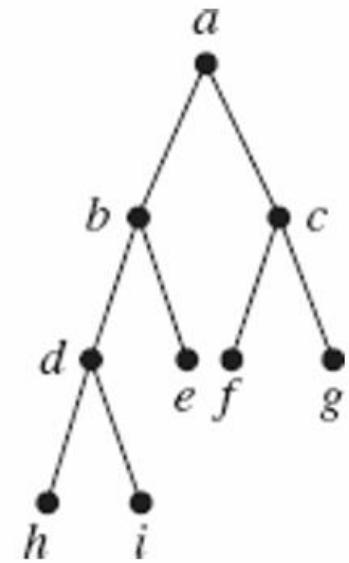


# Teoria dos Grafos – Percurso em árvores binárias

- **Percursos (caminhamento em árvores):**
- **Pré-ordem:** O caminhamento pré-ordem, também conhecido como travessia pré-fixada. (**RED**)

- A característica principal dessa travessia é que ela visita a raiz de cada **subárvore** antes de visitar seus filhos, seguindo a regra "**Raiz-Esquerda-Direita**".

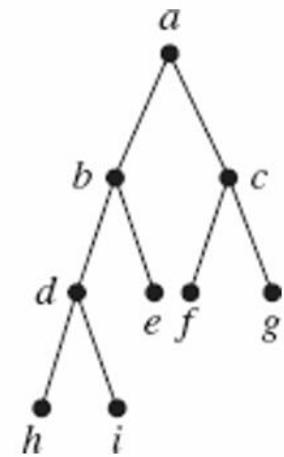
- **Visite a raiz (a)**
- **Visite o filho da esquerda de (a), que é (b):**
- **Visite o filho da esquerda de (b), que é (d):**
- **Visite o filho da esquerda de (d), que é (h):** Se (h) tivesse um filho à esquerda, você o visitaria. Como (h) for uma folha (ou seja, não tem filhos), então você completou a visita dos descendentes de (d) pela esquerda.
- **Volte para (d) e visite o filho da direita de (d), que é (i):** Como (i) não tem filhos, você "retorna" para (d), então completou a visita nos descendentes de d.



# Teoria dos Grafos – Percurso em árvores binárias

■ **Pré-ordem:** O caminhamento pré-ordem, também conhecido como travessia pré-fixada . **RED**.

- Já que ambos os filhos de (d) foram visitados, retornamos para (b), e visitamos o seu filho direito, que é o (e)
- Se (e) tivesse filhos, eles seriam visitados seguindo a mesma lógica. Como (e) é uma folha, e já visitamos todos os filhos esquerdos de (a), vamos visitar os filhos direitos de (a), começando com (c)
- Após visitar o (c), vamos para seu filho da esquerda, o (f).
- Como o (f) não tem mais filhos, voltamos para o (c) e visitamos seu filho da direita (g) , terminando assim o percurso.



Como (g) é uma folha, após visita-lo o percurso estará completo, pois todos os nós da árvore foram visitados seguindo a ordem pré-ordem. O percurso gerado foi:

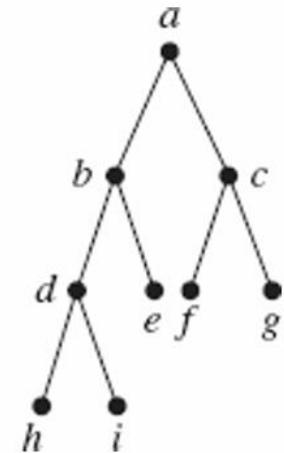
**A, B, D, H, I, E, C, F, G.**

# Teoria dos Grafos – Percurso em árvores binárias

## Pré-ordem – RED

### ■ Considerações:

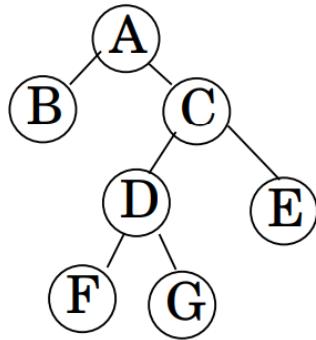
- **Características:** No percurso pré-ordem, a raiz da árvore ou subárvore é visitada primeiro, seguida pela subárvore esquerda e, por fim, pela subárvore direita.
- **Distribuição Final:** O elemento **raiz** será sempre o **primeiro elemento** da sequencia. Os nós **folhas** da subárvore direita serão **sempre os últimos elementos**.
- Esse percurso é útil para criar uma cópia da árvore ou expressar a árvore em notação prefixa



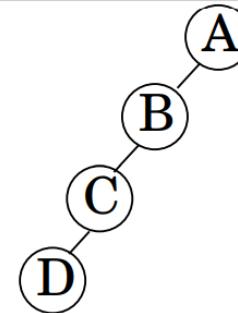
**A, B, D, H, I, E, C, F, G.**

# Teoria dos Grafos – Percurso em árvores binárias

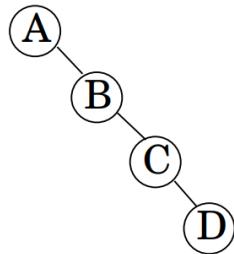
1.



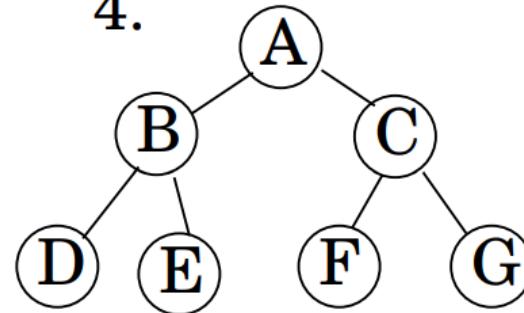
2.



3.

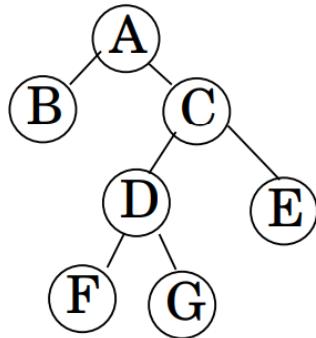


4.



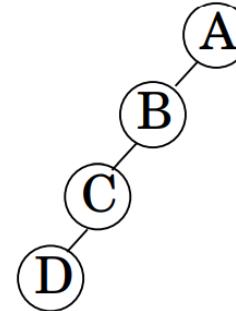
# Teoria dos Grafos – Percurso em árvores binárias

1.



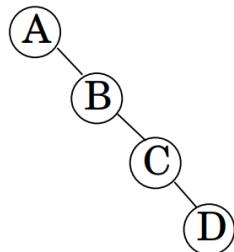
A B C D F G E

2.



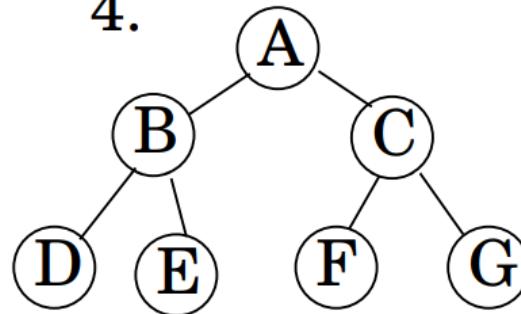
A B C D

3.



A B C D

4.

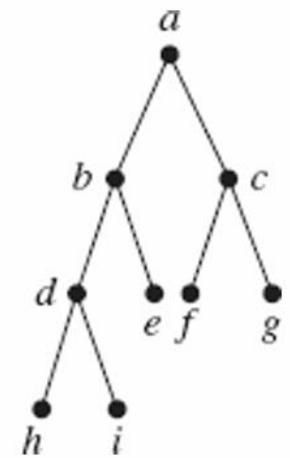


A B D E C F G

# Teoria dos Grafos – Percurso em árvores binárias

■ **Ordem Simétrica (em ordem):** O percurso simétrico visita **primeiro todos os nós na subárvore esquerda** depois o próprio nó, e então todos os nós na subárvore direita (que são maiores). (**ERD**)

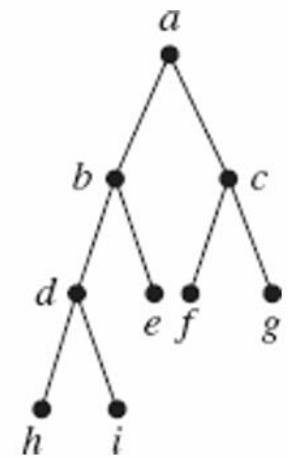
- Início na raiz (a): Começamos na raiz, mas **não a processamos** ainda porque precisamos começar com a subárvore esquerda.
- Movemos para o filho esquerdo da raiz (b). **Ainda não processamos** b porque temos que lidar com sua subárvore esquerda primeiro.
- Vamos para o filho esquerdo de b (d). Mais uma vez, **não processamos** d ainda porque temos que visitar sua subárvore esquerda.
- Visitar filho esquerdo (d): o h . h é um nó folha, então o **processamos** (visitamos) primeiro.



# Teoria dos Grafos – Percurso em árvores binárias

■ **Ordem Simétrica (em ordem):** O percurso simétrico visita **primeiro todos os nós na subárvore esquerda** depois o próprio nó, e então todos os nós na subárvore direita (que são maiores). (**ERD**)

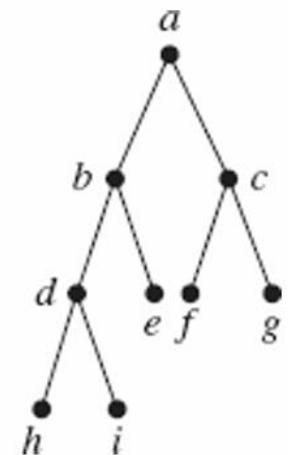
- Depois de visitar ( e processar) h, não há mais nada à esquerda para visitar, então voltamos e **processamos d**.
- Agora visitamos o filho direito de d (i). i é um nó folha, **então o processamos**.
- Depois de visitar i, retornamos para d, como não há mais filhos, então voltamos para **b e o processamos**.
- Agora visitamos o filho direito de b (e). Como e é um nó folha, **o processamos**.
- Depois de processar (e) , com toda subárvore esquerda processada, voltamos para a raiz e **processamos a**.



# Teoria dos Grafos – Percurso em árvores binárias

- **Ordem Simétrica (em ordem):** O percurso simétrico visita **primeiro todos os nós na subárvore esquerda** depois o próprio nó, e então todos os nós na subárvore direita (que são maiores). (**ERD**)

- Agora movemos para o filho direito da raiz (c). **Não processamos c** ainda, vamos primeiro para sua subárvore esquerda (f).
- (f) é um nó folha, **então o processamos**.
- Depois de processar (f), voltamos e **processamos c**.
- Por fim, visitamos o filho direito de c (g). **g é um nó folha, então o processamos.**



**h, d, i, b, e, a, f, c, g.**

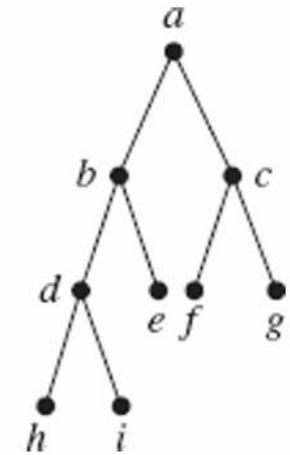
# Teoria dos Grafos – Percurso em árvores binárias

## Ordem Simétrica (em ordem) – ERD

### ■ Considerações:

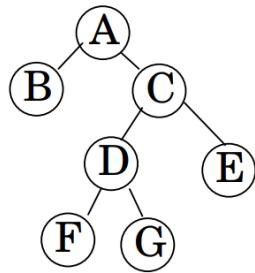
- **Características:** Nesta ordem, primeiro a subárvore esquerda é explorada (até chegar aos nós folha), depois o nó atual (raiz da subárvore) é processado, e por fim, a subárvore direita é explorada.
- **Distribuição Final:** Como o percurso vai "descendo" à esquerda até encontrar o primeiro nó folha disponível, os **primeiros elementos** da sequência gerada será sempre **a subárvore mais a esquerda, no sentido ERD**. Os **últimos elementos da lista** serão sempre a **última subárvore da direita** (sempre percorrida no sentido ERD).
- Esse percurso é particularmente útil em arvores binárias de busca, para obter uma sequência ordenada dos elementos da árvore.

**h, d, i, b, e, a, f, c, g**

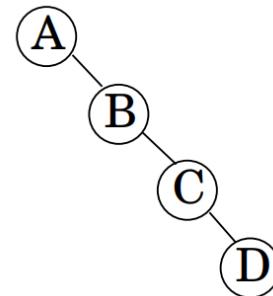


# Teoria dos Grafos – Percurso em árvores binárias

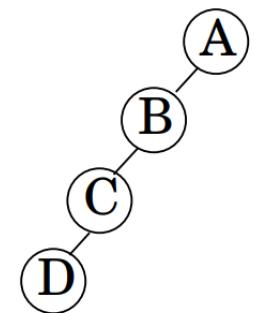
1.



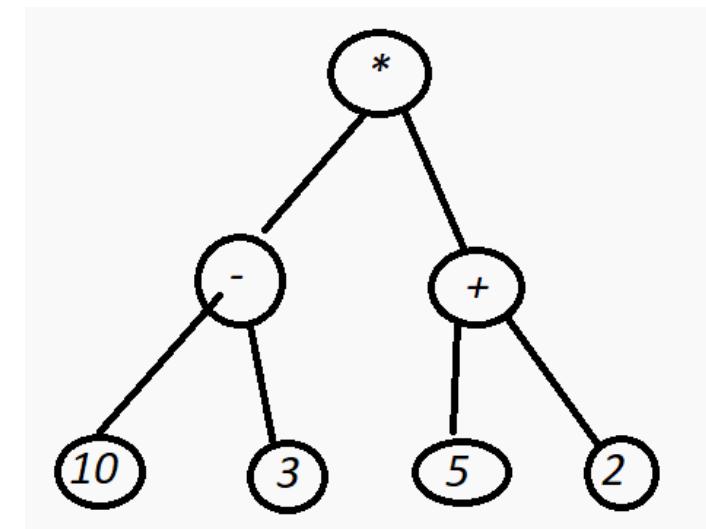
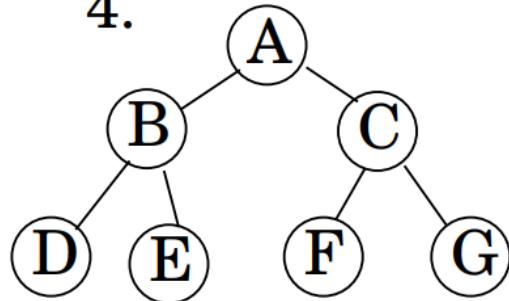
2.



3.

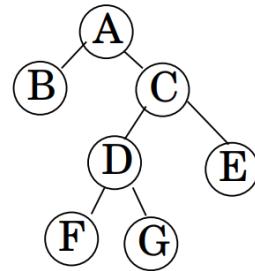


4.



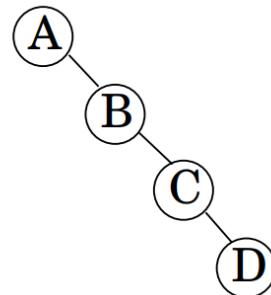
# Teoria dos Grafos – Percurso em árvores binárias

1.



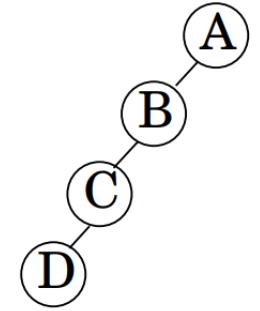
B A F D G C E

2.



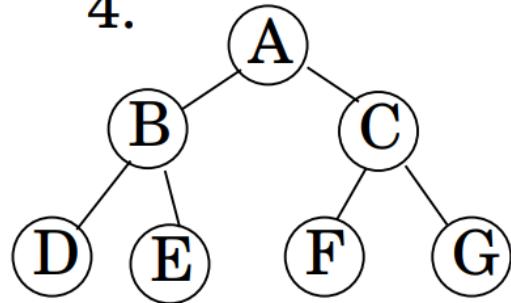
A B C D

3.



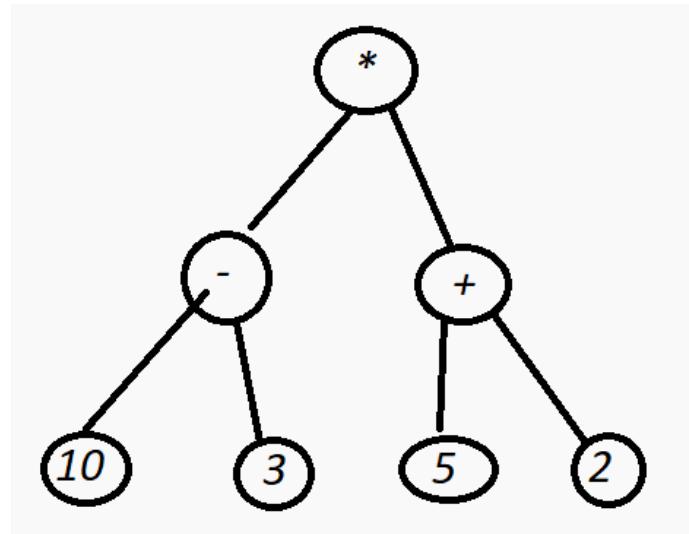
D C B A

4.



D B E A F C G

14:26:46



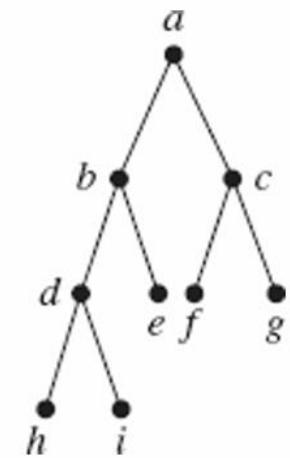
$$10 - 3 * 5 + 2 = -3 \text{ ou } 49$$

# Teoria dos Grafos – Percurso em árvores binárias

## ■ **Pós-ordem:** (Esquerda-Direita-Raiz - **EDR**)

percorre uma árvore binária na sequencia: subárvore esquerda, depois a subárvore direita, e por último o próprio nó (a raiz da subárvore atual).

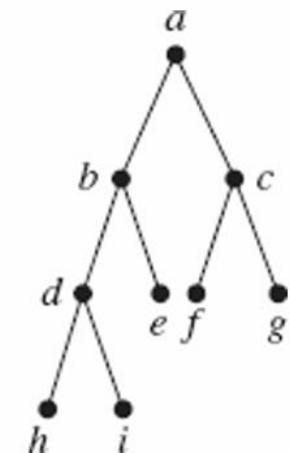
- Começamos na raiz (a), mas **não a processamos ainda**, pois o percurso em pós-ordem exige que processemos as subárvores primeiro.
- Movemos para a subárvore esquerda de a, que é b. Novamente, **não processamos b ainda**, pois devemos começar com sua subárvore esquerda.
- Dentro da subárvore b, vamos para a esquerda, chegando a d. **Ainda não processamos d** porque temos que verificar se há subárvores à esquerda ou à direita para processar primeiro.



# Teoria dos Grafos – Percurso em árvores binárias

## ■ **Pós-ordem:** (Esquerda-Direita-Raiz - **EDR**)

percorre uma árvore binária na ordem: subárvore esquerda, depois a subárvore direita, e por último o próprio nó (a raiz da subárvore atual).



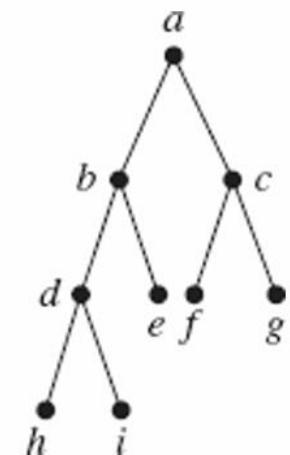
- Chegamos a h, que é o filho esquerdo de d e uma folha, então **processamos o (h) primeiro**.
- Então voltamos para d. **mas não o processmos**.
- Movemos para a direita de d e encontramos i, **processamos i**, pois é uma folha.
- Retornamos para (d) , como todos seus filhos foram processados, **agora processamos d**.
- voltamos para b e agora processamos a subárvore direita de b, que é (e). Como e não tem filhos, **então é processado diretamente (e)**.

# Teoria dos Grafos – Percurso em árvores binárias

■ **Pós-ordem:** (Esquerda-Direita-Raiz- **EDR**)

percorre uma árvore binária na ordem: subárvore esquerda, depois a subárvore direita, e por último o próprio nó (a raiz da subárvore atual).

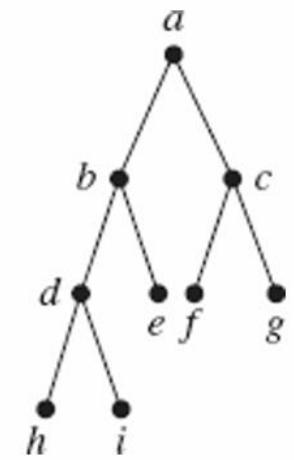
- Agora, tendo processado ambas subárvore de b, processamos a raiz **(b)**.
- Voltamos para (a). E agora vamos para a subárvore direita de a, que é c. **Não processamos c ainda**, pois devemos verificar suas subárvore primeiro.
- Vamos para a esquerda de c e encontramos f, **processamos f, pois é uma folha**.
- Vamos para a direita de c e encontramos g, **processamos g**, pois é uma folha.



# Teoria dos Grafos – Percurso em árvores binárias

## ■ **Pós-ordem:** (Esquerda-Direita-Raiz - **EDR**)

percorre uma árvore binária na ordem: subárvore esquerda, depois a subárvore direita, e por último o próprio nó (a raiz da subárvore atual).



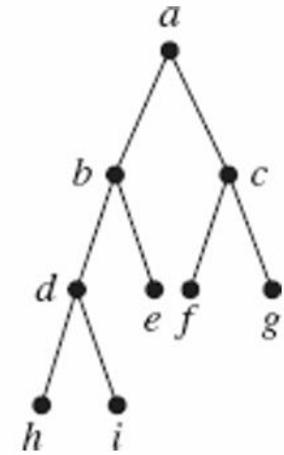
- Tendo processado ambos os filhos de c, agora **podemos processar c.**
- Finalmente, após todas as subárvore terem sido processadas, voltamos para a raiz e **processamos a.**

**h, i, d, e, b, f, g, c, a.**

# Teoria dos Grafos – Percurso em árvores binárias

## Pós-ordem: (EDR) Considerações:

- **Características:** Cada nó raiz é processado somente após o processamento completo de suas subárvores esquerda e direita.
- **Distribuição Final:** Como o percurso vai "descendo" à esquerda até encontrar o primeiro nó folha disponível, os **primeiros elementos** da sequência gerada serão sempre os **nós da subarvore mais a esquerda**. O **último elemento da lista** será sempre a **o elemento que está na RAIZ da arvore**.
- Esse percurso é particularmente útil em árvores binárias de busca, para obter uma sequência ordenada dos elementos da árvore.

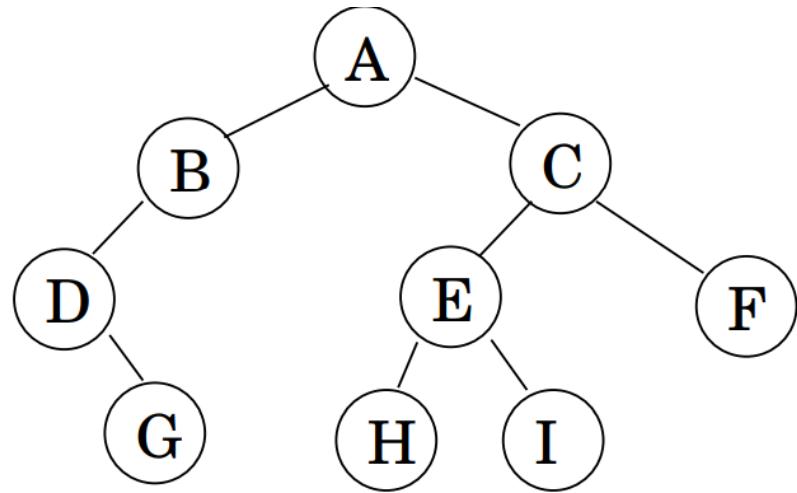
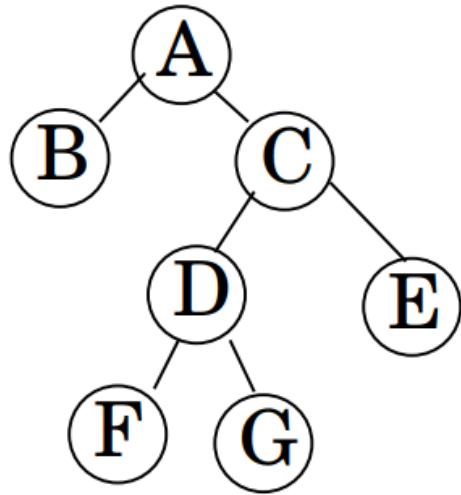


**h, i, d, e, b, f, g, c, a**

# Teoria dos Grafos – Percurso em árvores binárias

Percorso pós ordem

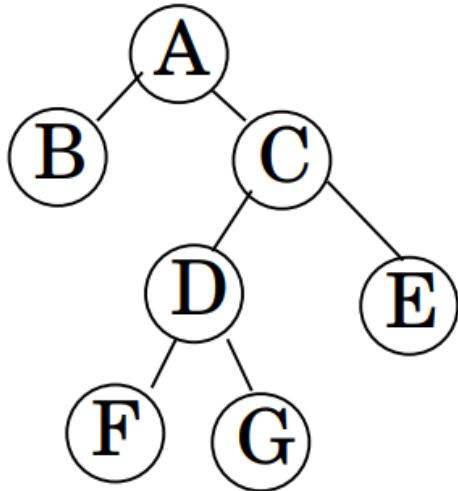
1.



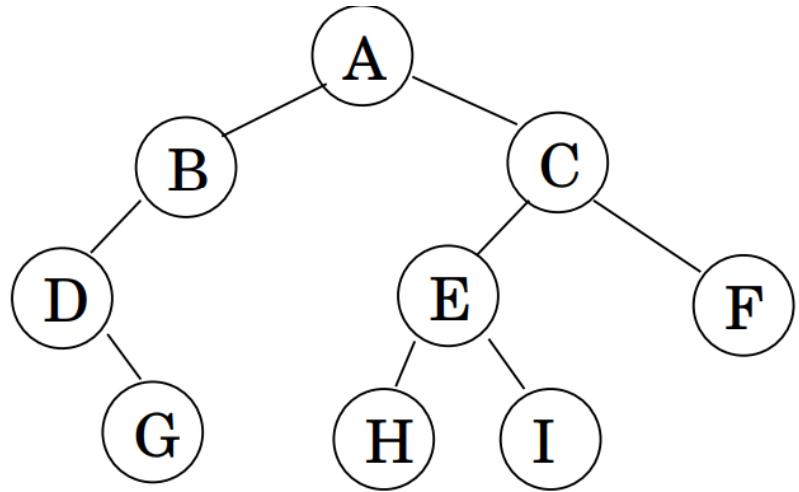
# Teoria dos Grafos – Percurso em árvores binárias

Percorso pós ordem

1.



B, F, G, D, E, C, A.



G D B H I E F C A

# Teoria dos Grafos - Árvores de Pesquisa

## Árvores de Pesquisa

# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores de Pesquisa:** Árvores de pesquisa são uma categoria especial de árvores usadas em estruturas de dados e algoritmos de grafos, principalmente para organizar e facilitar a busca de informações.
  - **Propósito:** O principal objetivo das Árvores de Pesquisa é manter os dados de forma ordenada, permitindo buscas eficientes, além de inserções e remoções com manutenção da ordem.
  - **Aplicações:** São amplamente utilizadas em bancos de dados para indexação, em sistemas de arquivos para organização de dados e em qualquer cenário onde a eficiência das operações de busca, inserção e exclusão é crítica.
    - Incluem Árvores binárias, Árvores AVL, Árvores Vermelho-Preto..

# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores Binárias de Pesquisa (Binary Search Trees - BST)** : Uma Árvore Binária de Pesquisa é uma estrutura de dados que organiza os elementos de forma hierárquica, de forma que para cada nó:
  - Os filhos da **esquerda** são **menores** que o nó
  - Os filhos da **direita** são **maiores** que o nó
- **Busca Eficiente**: Se a árvore estiver balanceada, operações de busca, inserção e remoção podem ser realizadas em tempo logarítmico  $O(\log n)$ .
- **Organização Dinâmica**: Diferentemente de arrays, onde a inserção e remoção de elementos podem ser custosas, BSTs permitem essas operações de forma mais eficiente.
- **Flexibilidade**: A estrutura de dados não tem um tamanho fixo e pode crescer ou diminuir conforme necessário.  
14:26 26

# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores Binárias de Pesquisa (Binary Search Trees - BST)** : Uma Árvore Binária de Pesquisa é uma estrutura de dados que organiza os elementos de forma hierárquica, de forma que para cada nó:
  - Os filhos da **esquerda** são **menores** que o nó
  - Os filhos da **direita** são **maiores** que o nó
- **Desbalanceamento:** Sem mecanismos de auto-balanceamento, operações consecutivas de inserção ou remoção podem levar a uma árvore desbalanceada, degradando performance.
- **Complexidade de Implementação:** A implementação de operações de balanceamento pode ser complexa.
- **Uso de Memória:** Cada nó na BST requer espaço adicional para os ponteiros dos filhos, além do próprio dado.  
14:26:46

# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores Binárias de Pesquisa (BST) :**

Considere a seguinte série de inserções em uma BST vazia:

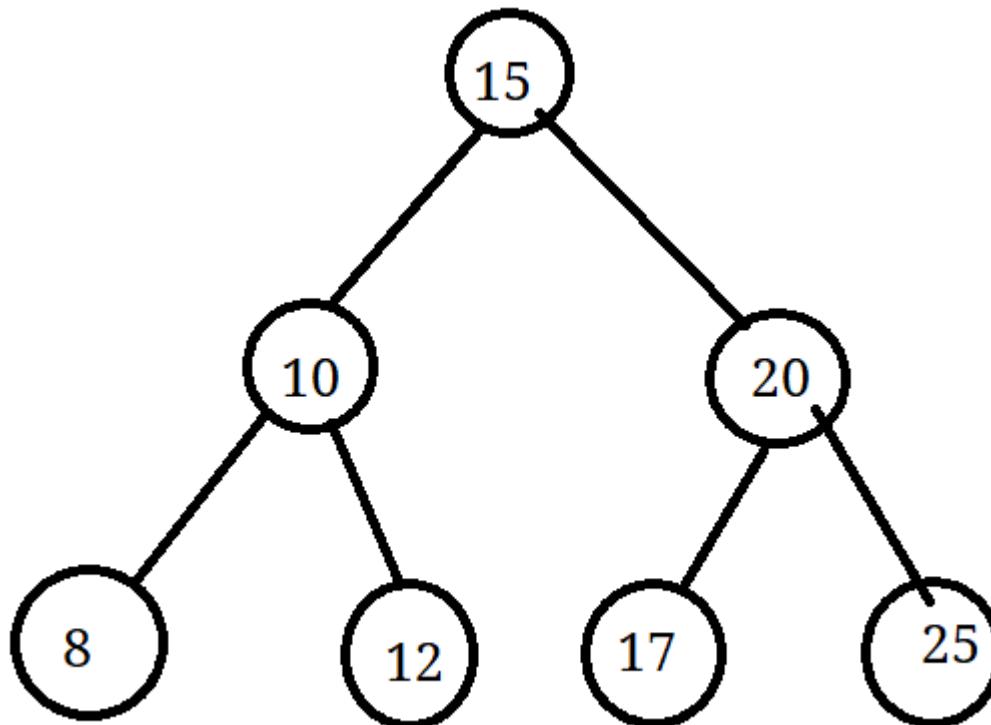
15, 10, 20, 8, 12, 17, 25.

# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores Binárias de Pesquisa (BST) :**

Considere a seguinte série de inserções em uma BST vazia:

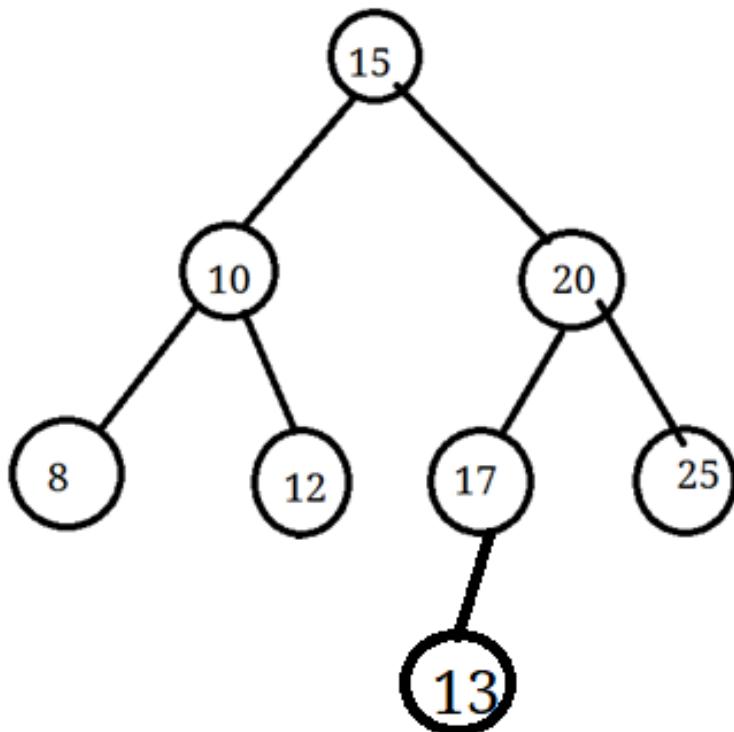
15, 10, 20, 8, 12, 17, 25.



# Teoria dos Grafos - Árvores de Pesquisa

- Árvores Binárias de Pesquisa (BST) :

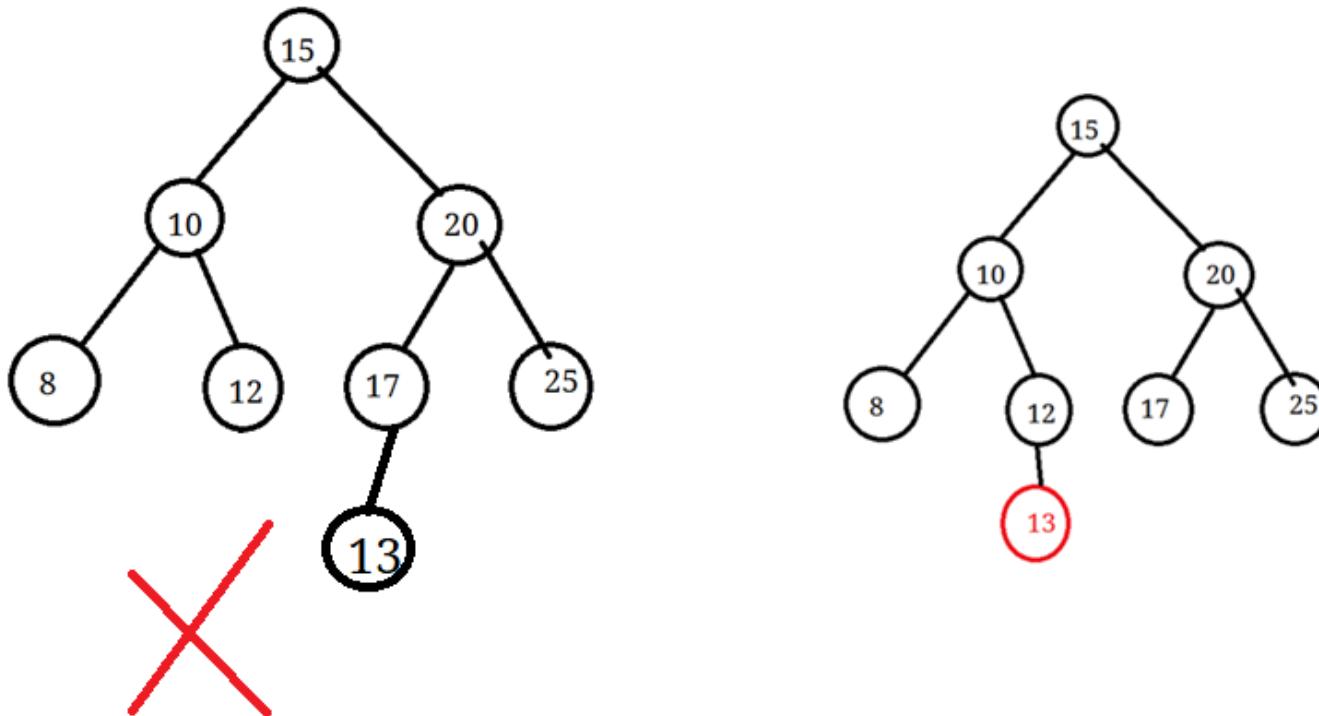
Agora insira o valor **13** na mesma árvore.



# Teoria dos Grafos - Árvores de Pesquisa

- Árvores Binárias de Pesquisa (BST) :

Agora insira o valor **13** na mesma árvore.



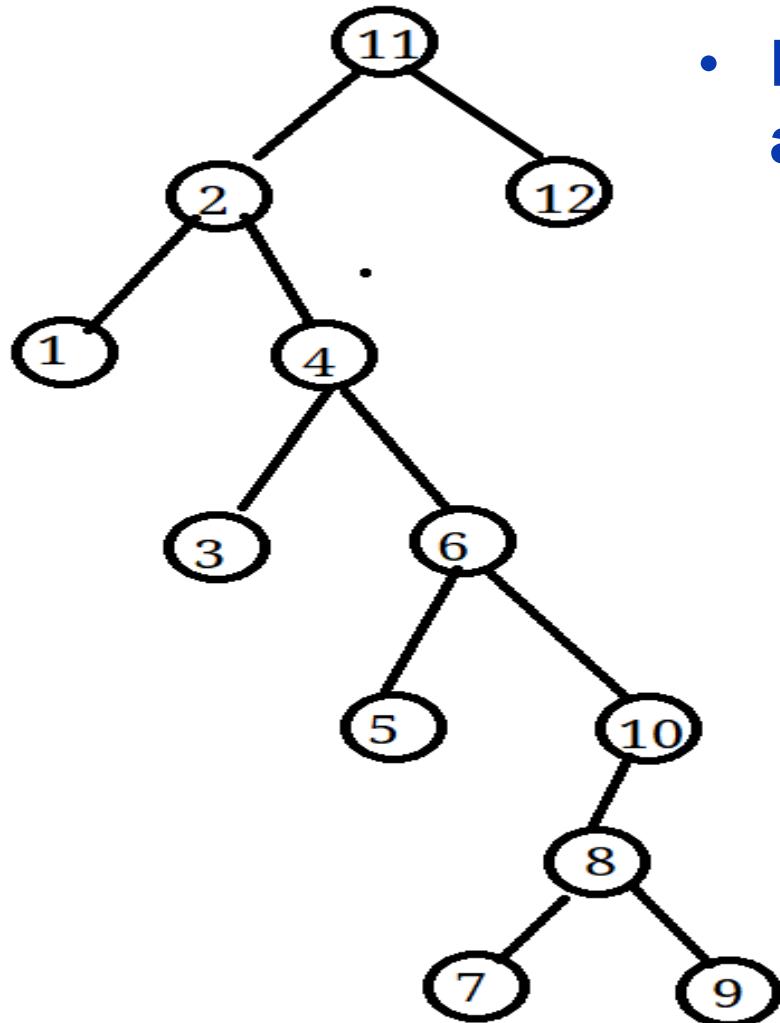
# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores Binárias de Pesquisa (BST) :**

Crie uma árvore com a sequencia:

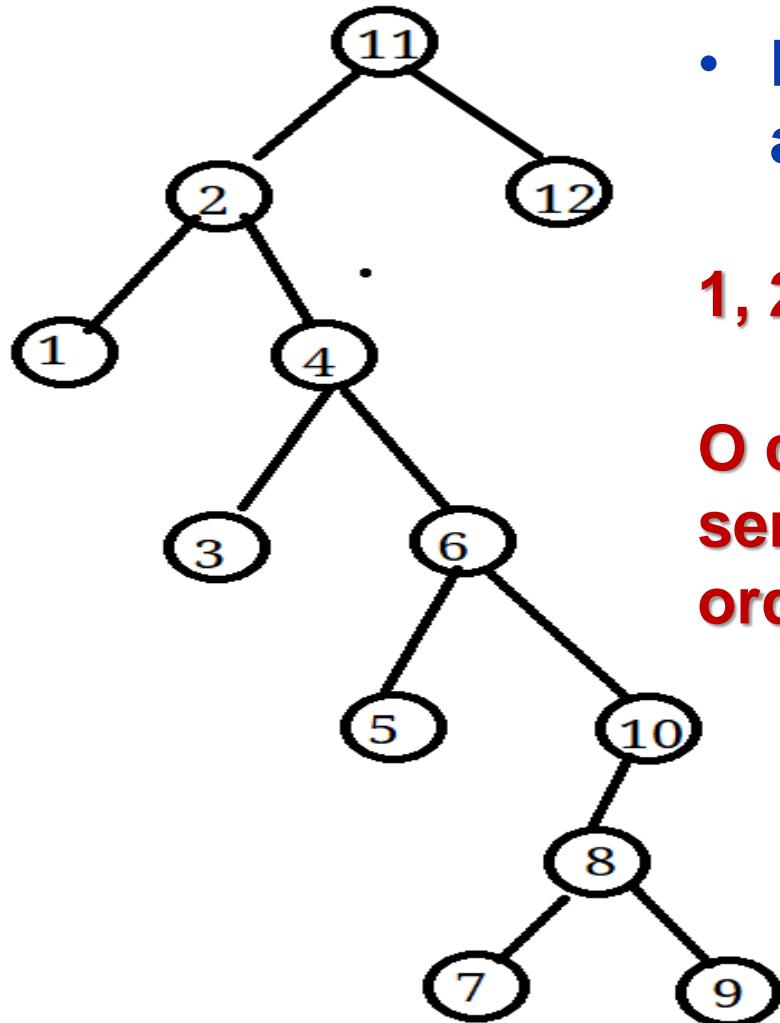
11, 2, 12, 1, 4, 3, 6, 5, 10, 8, 7, 9

# Teoria dos Grafos - Árvores de Pesquisa



- Faça o caminho in-ordem na árvore:

# Teoria dos Grafos - Árvores de Pesquisa

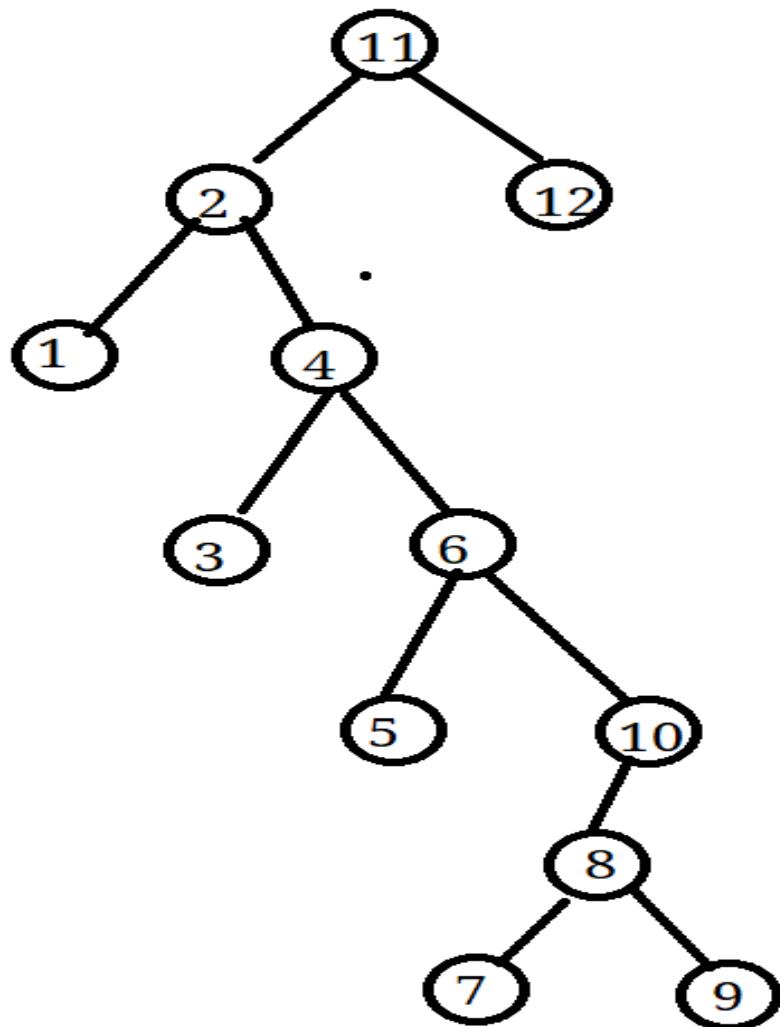


- Faça o caminho in-ordem na árvore:

**1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12**

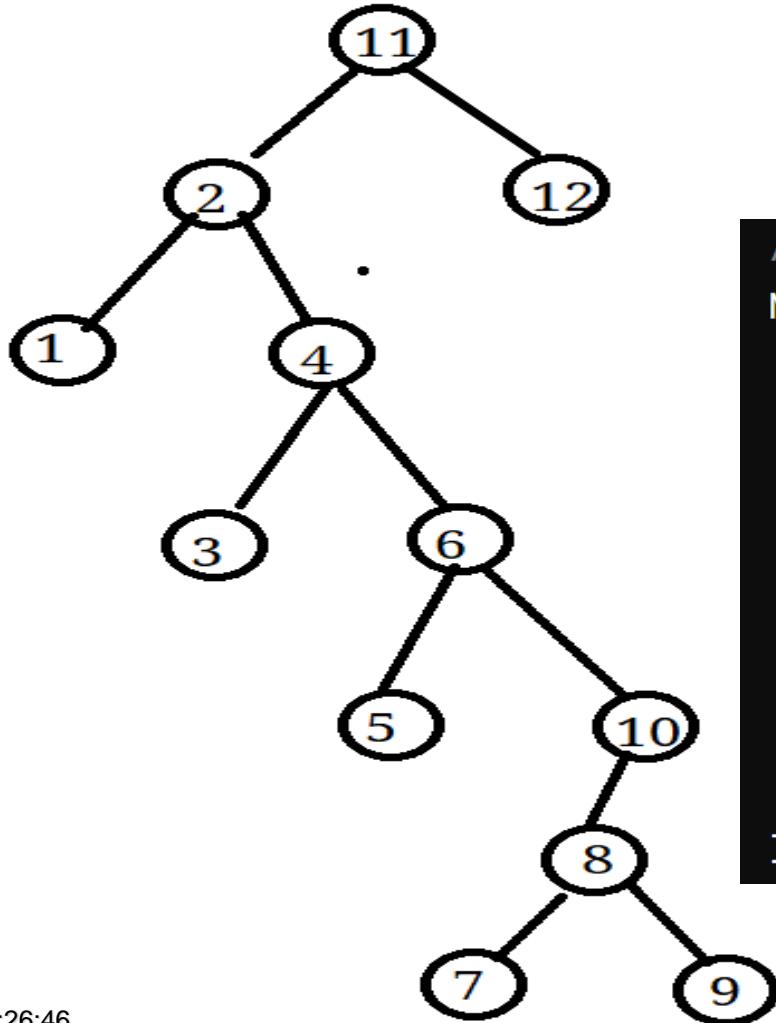
**O caminho em ordem ERD sempre irá retornar os dados ordenados.**

# Teoria dos Grafos - Árvores de Pesquisa



- Localizar o valor 8:

# Teoria dos Grafos - Árvores de Pesquisa



- Localizar o valor 8:

```
// Função para buscar um valor na BST
Node* search(Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return search(root->left, data);
    } else {
        return search(root->right, data);
    }
}
```

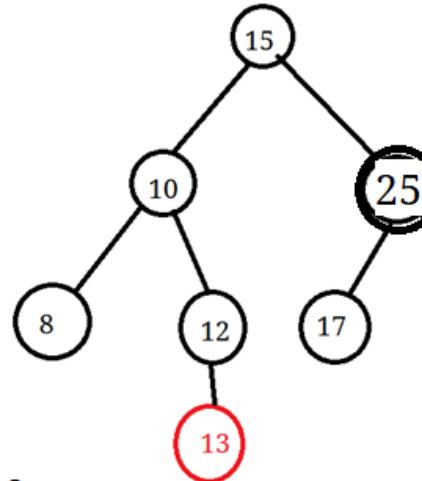
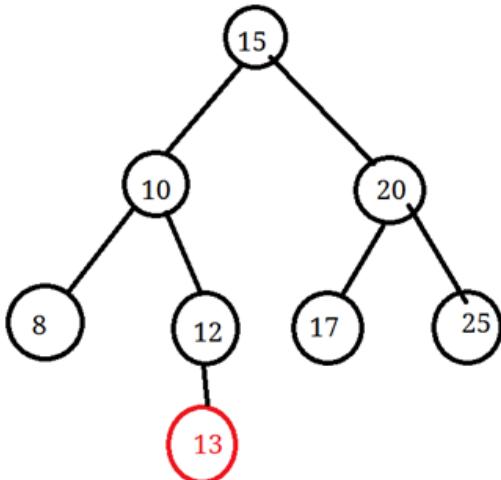
# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores Binárias de Pesquisa (BST) :**
- **Remover um valor:** Quando um nó é removido de uma BST, a estrutura da árvore precisa ser ajustada para manter as propriedades da BST.
- O processo varia dependendo se o nó a ser removido é um nó folha, tem um filho ou tem dois filhos:
  - **Nó Folha:** Simplesmente removido da árvore.
  - **Nó com Um Filho:** O nó é removido e substituído diretamente pelo seu filho.
  - **Nó com Dois Filhos:** substituir o valor do nó a ser removido pelo **menor valor na subárvore direita** ou pelo o **maior valor na subárvore esquerda**.

**Importante:** Uma vez definida a regra de escolha do nó substituto, ela deve ser a mesma para todas as operações de remoção!

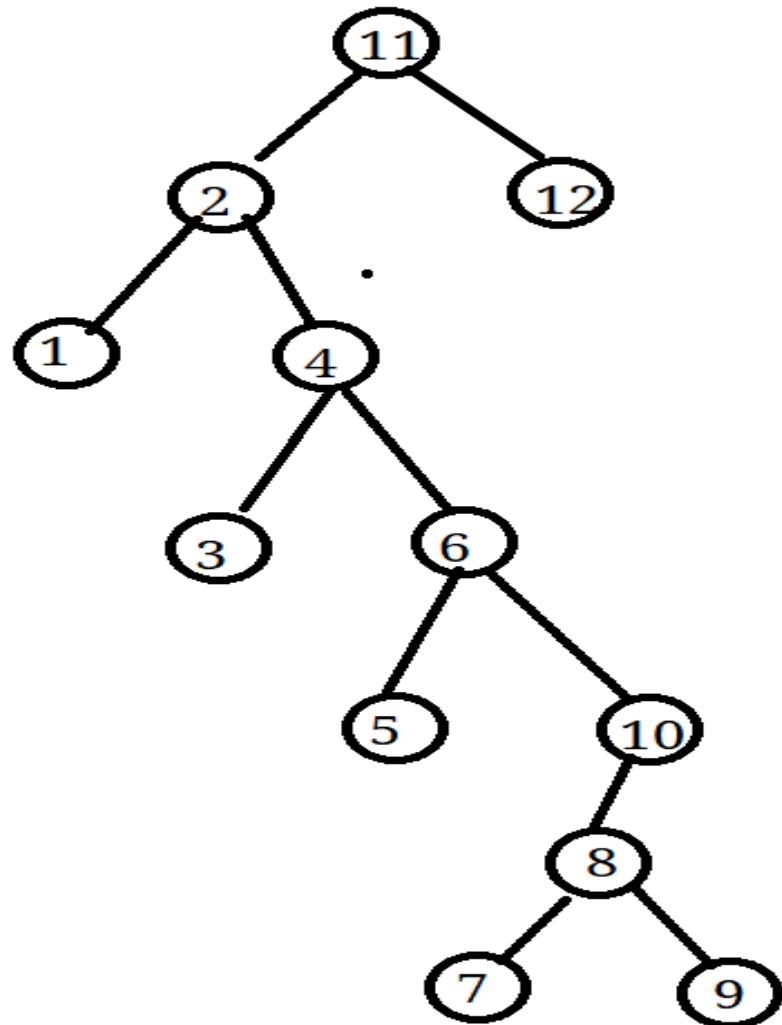
# Teoria dos Grafos - Árvores de Pesquisa

- **Árvores Binárias de Pesquisa (BST) :**
- **Remover um valor:** Quando um nó é removido de uma BST, a estrutura da árvore precisa ser ajustada para manter as propriedades da BST.



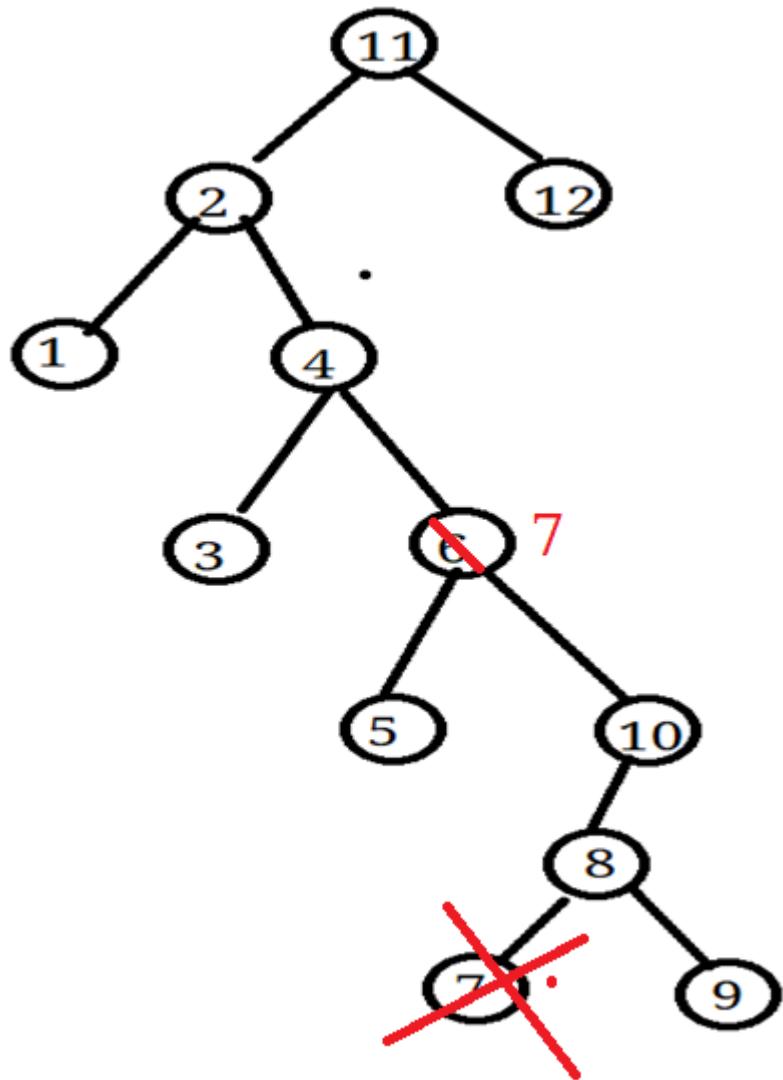
Removendo o nó 20

# Teoria dos Grafos - Árvores de Pesquisa



- Removendo o elemento 6 .

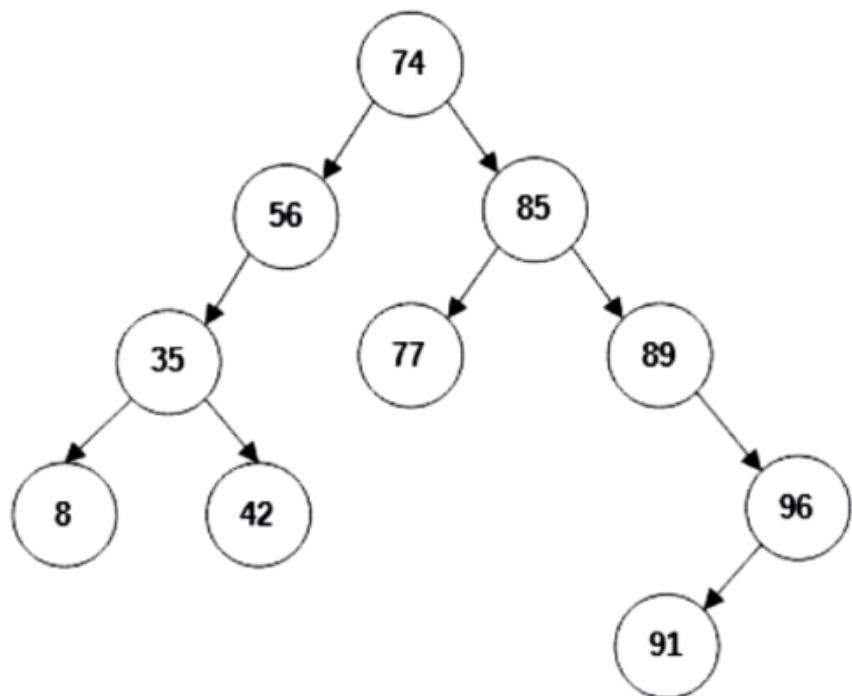
# Teoria dos Grafos - Árvores de Pesquisa



- Removendo o elemento 6.
- Substituindo o valor do nó a ser removido pelo valor do **menor valor na subárvore direita = (7)**
- Caminhamento in ordem: (ERD)  
1,2,3,4,5,7,8,9,10,11,12

# Teoria dos Grafos - Árvores de Pesquisa

Qual a saída do percurso **pós-ordem?**



- A 8, 12, 23, 35, 42, 56, 64, 71, 74, 77, 85, 86, 89, 91, 93, 96.
- B 74, 56, 35, 8, 23, 12, 42, 64, 71, 85, 77, 89, 86, 96, 91, 93.
- C 12, 23, 8, 42, 35, 71, 64, 56, 77, 86, 93, 91, 96, 89, 85, 74.
- D 12, 23, 8, 42, 35, 64, 56, 71, 77, 86, 93, 91, 96, 89, 85, 74.
- E Nenhuma das alternativas.

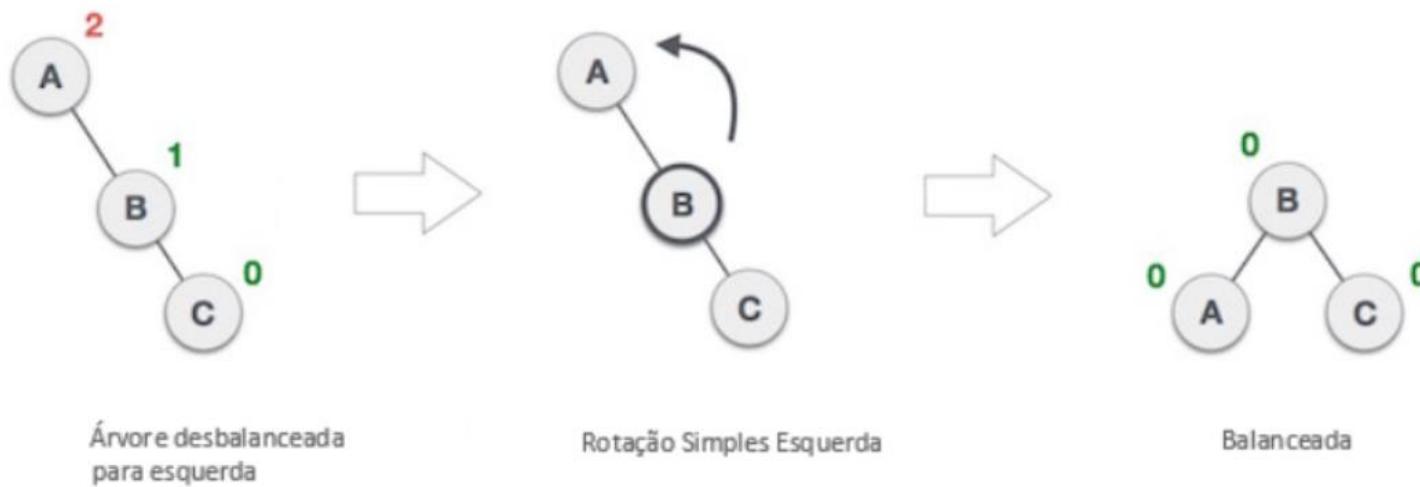
Inserir os elementos:  
23, 12, 64, 93, 71, 86

# Teoria dos Grafos - Árvores de Pesquisa

- **Árvore AVL (ou Estritamente Balanceada):** é uma árvore de busca binária de altura balanceada:
  - para cada nó  $x$ , a diferença entre as alturas das subárvore à esquerda e à direita de  $x$  é no máximo 1.
- **Balanceamento:** O balanceamento de um NÓ é definido como a altura de sua subárvore esquerda menos a altura de sua subárvore direita. Logo, será -1 , 0, ou 1.
- **Após inserções / remoções:** Pode ser necessária fazer transformações na árvore para que a mesma continue balanceada.
  - Estas transformações são feitas através de rotações.
    - A rotação poderá ser feita à esquerda ou à direita dependendo do desbalanceamento que tiver que ser solucionado

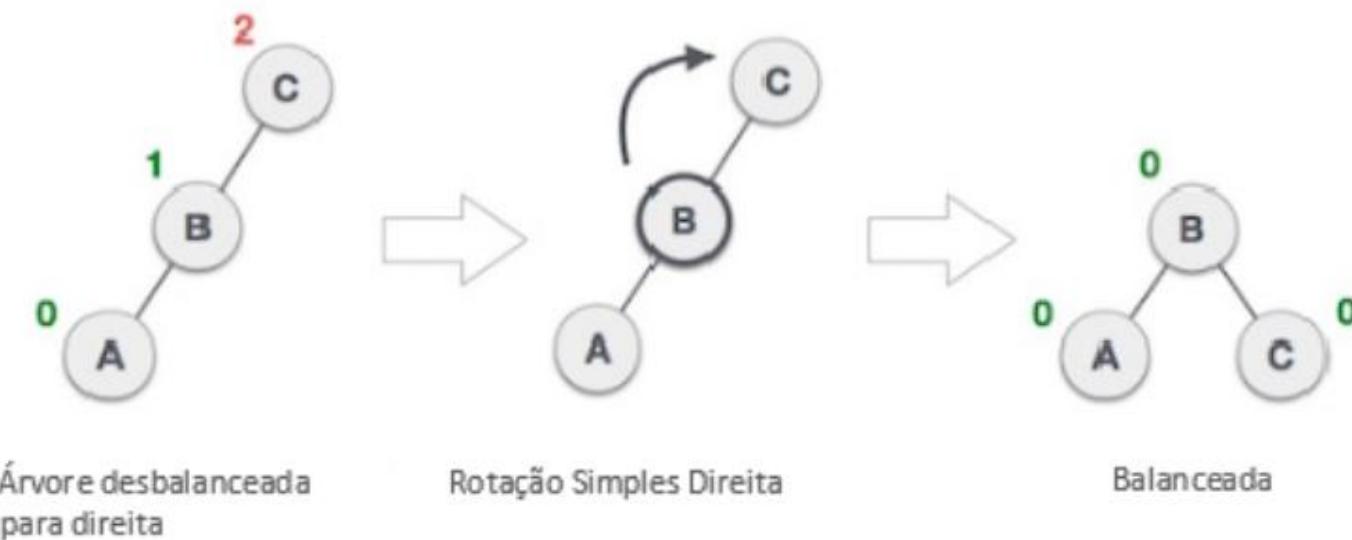
# Teoria dos Grafos - Árvores de Pesquisa

- **Árvore AVL (ou Estritamente Balanceada):** é uma árvore de busca binária de altura balanceada:
  - para cada nó  $x$ , a diferença entre as alturas das subárvore à esquerda e à direita de  $x$  é no máximo 1.



# Teoria dos Grafos - Árvores de Pesquisa

- **Árvore AVL (ou Estritamente Balanceada):** é uma árvore de busca binária de altura balanceada:
  - para cada nó  $x$ , a diferença entre as alturas das subárvore à esquerda e à direita de  $x$  é no máximo 1.



# Teoria dos Grafos - Representação de Grafos

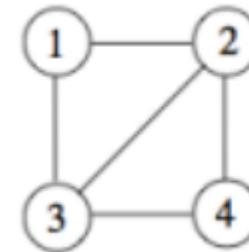
## Representação de Grafos

# Teoria dos Grafos – Representação computacional

A representação **gráfica** de um grafo através de vértices e arestas permite rápida compreensão e utilização dos conceitos da teoria de grafos, porém, **não é adequado para armazenar o grafo no computador.**

$$V = \{1,2,3,4\}$$

$$E = \{(1,2), (1,3), (2,3), (2,4), (3,4)\}$$



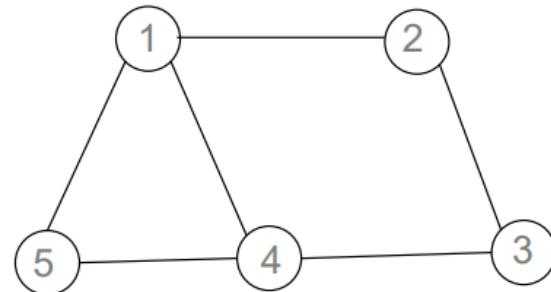
**Representação Computacional de Grafos:** É uma forma de representar grafos em um formato que pode ser facilmente manipulado por algoritmos em um computador.

# Teoria dos Grafos – Representação computacional

- Existem diversas maneiras de representar grafos computacionalmente, sendo as mais comuns:
  - Listas de adjacência,
  - Matrizes de adjacência,

# Teoria dos Grafos – Representação computacional

- **Lista de adjacência:** É uma coleção de listas, **uma para cada vértice do grafo**, onde cada lista contém os vértices diretamente conectados a ele.
  - Essa representação é mais eficiente em termos de espaço para grafos esparsos, uma vez que armazena apenas as conexões existentes.
  - Permite uma iteração fácil sobre os vizinhos de um vértice, o que é útil para vários algoritmos de grafos



vértice	vértices adjacentes
1	2 → 4 → 5
2	1 → 3
3	2 → 4
4	1 → 3 → 5
5	1 → 4

# Teoria dos Grafos – Representação computacional

## **Lista de adjacência: (Vantagens)**

- **Eficiência de Espaço:** Listas de adjacência são eficientes em termos de espaço para grafos esparsos, pois armazenam apenas as arestas existentes.
- **Facilidade para Adicionar/Remover Arestras:** Adicionar ou remover arestas é relativamente simples e rápido em listas de adjacência, exigindo apenas a adição ou remoção de um item na lista correspondente.
- **Iteração Eficiente Sobre Arestras:** Para algoritmos que precisam iterar sobre as arestas de vértices específicos, as listas de adjacência são ideais, pois fornecem acesso direto às arestas conectadas a um vértice.

# Teoria dos Grafos – Representação computacional

## **Lista de adjacência: (Desvantagens)**

- **Verificação de Aresta Lenta:** Verificar a existência de uma aresta específica entre dois vértices pode ser lento em listas de adjacência, pois pode exigir uma busca linear na lista de um dos vértices.
- **Espaço para Grafos Densos:** Para grafos muito densos, as listas de adjacência podem acabar consumindo mais espaço do que uma matriz de adjacência, especialmente se as listas armazenarem informações adicionais como peso das arestas.
- **Variação de Desempenho com a Implementação:** O desempenho das operações pode variar significativamente com a implementação específica das listas (por exemplo, listas encadeadas versus arrays dinâmicos), influenciando a eficiência de diferentes operações.
- **Exclusão de Vértices:** A exclusão de vértices em uma implementação de lista de adjacências pode ser mais complexa e menos eficiente do que em outras representações.

```
class Grafo:  
  
    def __init__(self, vertices):  
        self.vertices = vertices  
        self.lista_adjacencia = []  
        for i in range(vertices):  
            self.lista_adjacencia.append([])  
  
    def adiciona_aresta(self, u, v):  
        self.lista_adjacencia[u].append(v) # Adiciona v à lista de u  
        self.lista_adjacencia[v].append(u) # Adiciona u à lista de v,
```

```
def remove_aresta(self, u, v):  
    if v in self.lista_adjacencia[u]:  
        self.lista_adjacencia[u].remove(v)  
    if u in self.lista_adjacencia[v]:  
        self.lista_adjacencia[v].remove(u)
```

```
if __name__ == "__main__":  
  
    grafo = GrafoNaoDirecionado()  
    grafo.adicionar_aresta(0, 1)  
    grafo.adicionar_aresta(0, 2)  
    grafo.adicionar_aresta(1, 2)  
    grafo.adicionar_aresta(1, 3)  
    grafo.adicionar_aresta(1, 4)  
    grafo.adicionar_aresta(2, 3)
```

```
def listar_vertices(self):
    for i in range(self.vertices):
        print(f"Vértice {i} com arestas: {self.lista_adjacencia[i]}")

def listar_arestas(self):
    arestas_vistas = set()
    for i, adjacencias in enumerate(self.lista_adjacencia):
        for adjacente in adjacencias:
            if (adjacente, i) not in arestas_vistas:
                print(f"Aresta entre {i} e {adjacente}")
                arestas_vistas.add((i, adjacente))
```

```
Vértice 0 com arestas: [1, 4]
Vértice 1 com arestas: [0, 2, 3]
Vértice 2 com arestas: [1, 3]
Vértice 3 com arestas: [1, 2, 4]
Vértice 4 com arestas: [0, 3]
```

```
Aresta entre 0 e 1
Aresta entre 0 e 4
Aresta entre 1 e 2
Aresta entre 1 e 3
Aresta entre 2 e 3
Aresta entre 3 e 4
```

# Teoria dos Grafos – Representação computacional

```
def remove_aresta(self, u, v):
    if v in self.lista_adjacencia[u]:
        self.lista_adjacencia[u].remove(v)
    if u in self.lista_adjacencia[v]:
        self.lista_adjacencia[v].remove(u)
```

Note que a remoção de um **vértice** em uma estrutura de dados baseada em lista para grafos não é tão trivial quanto as operações de adição ou remoção de arestas.

Isso se deve ao fato de que a remoção de um vértice pode afetar os índices de todos os vértices subsequentes na lista de adjacência.

Uma abordagem simplificada é marcar o vértice como removido, definindo sua lista de adjacência como **None** ou uma lista vazia;

# Teoria dos Grafos – Representação computacional

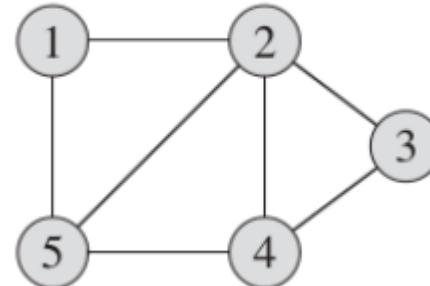
- **Matrizes de Adjacência:** Nesta representação, um grafo é expresso como uma matriz quadrada de tamanho  $V \times V$ , onde  $V$  é o número de vértices no grafo.
  - Cada célula  $M[i][j]$  na matriz indica a presença (em alguns casos, o peso) de [uma aresta entre o vértice  $i$  e o vértice  $j$ .
    - $M[i][j] = 1$  – Indica que existe uma aresta entre  $i$  e  $j$
    - $M[i][j] = 0$  – Indica que NÃO existe uma restas entre  $i$  e  $j$

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

# Teoria dos Grafos – Representação computacional

- **Matrizes de Adjacência:** Nesta representação, um grafo é expresso como uma matriz quadrada de tamanho  $V \times V$ , onde  $V$  é o número de vértices no grafo.
  - Cada célula  $M[i][j]$  na matriz indica a presença (em alguns casos, o peso) de [uma aresta entre o vértice  $i$  e o vértice  $j$ .
    - $M[i][j] = 1$  – Indica que existe uma aresta entre  $i$  e  $j$
    - $M[i][j] = 0$  – Indica que NÃO existe uma restas entre  $i$  e  $j$

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



# Teoria dos Grafos – Representação computacional

## Matriz de adjacência: (Vantagens)

- **Verificação Rápida de Arestas:** Verificar se existe uma aresta entre dois vértices é feito em tempo constante  $O(1)$ , simplesmente verificando o valor da célula correspondente na matriz.
- **Representação Simples:** A representação é intuitivamente simples e facilita a implementação de algoritmos que manipulam grafos, especialmente para operações que necessitam verificar rapidamente a existência de conexões entre vértices.
- **Adequada para Grafos Densos:** Em grafos onde o número de arestas é próximo do número máximo possível, a matriz de adjacência pode ser mais eficiente em termos de espaço do que listas de adjacência.

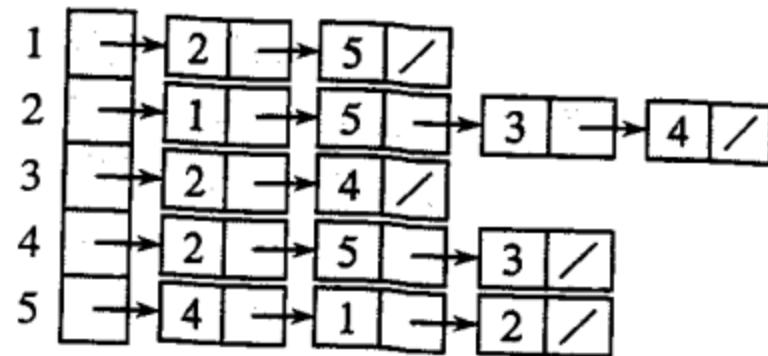
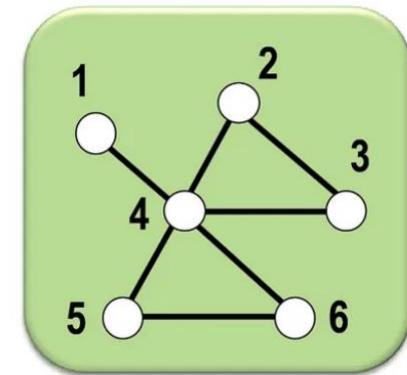
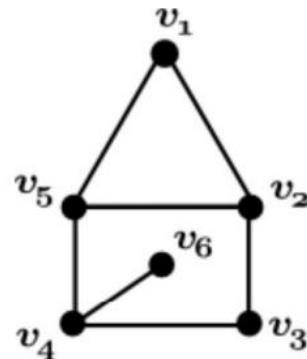
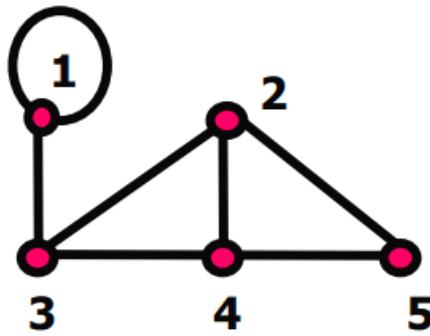
# Teoria dos Grafos – Representação computacional

## Matriz de adjacência: (Desvantagens)

- **Uso Ineficiente de memória:** Matrizes irão utilizar sempre um espaço de memória equivalente a  $V^2$  (onde  $v$  é a qtd de vértices) , mesmo que existam poucas arestas ligando estes vértices .
- **Adição/Remoção de Vértices:** Alterar o número de vértices (por exemplo, adicionar ou remover um vértice) requer a reconstrução da matriz, o que pode ser custoso em termos de tempo de execução, especialmente para grafos grandes.
- **Iterar sobre Vértices Adjacentes:** Para um dado vértice, encontrar todos os seus vértices adjacentes requer a iteração sobre uma linha ou coluna inteira da matriz, o que pode ser ineficiente comparado às listas de adjacência, especialmente se o grafo for esparsa.
- .

# Teoria dos Grafos – Representação computacional

Represente os grafos abaixo através de lista e matriz de Adjacência .



Forneça uma representação de lista e matriz de adjacência para uma arvore binaria completa com 7 vértices.  
14:26:46

# Teoria dos Grafos – **Grafos Direcionados (Digrafos)**

## **Grafos Direcionados (Digrafos)**

# Teoria dos Grafos – Grafos Direcionados (Digrafos)

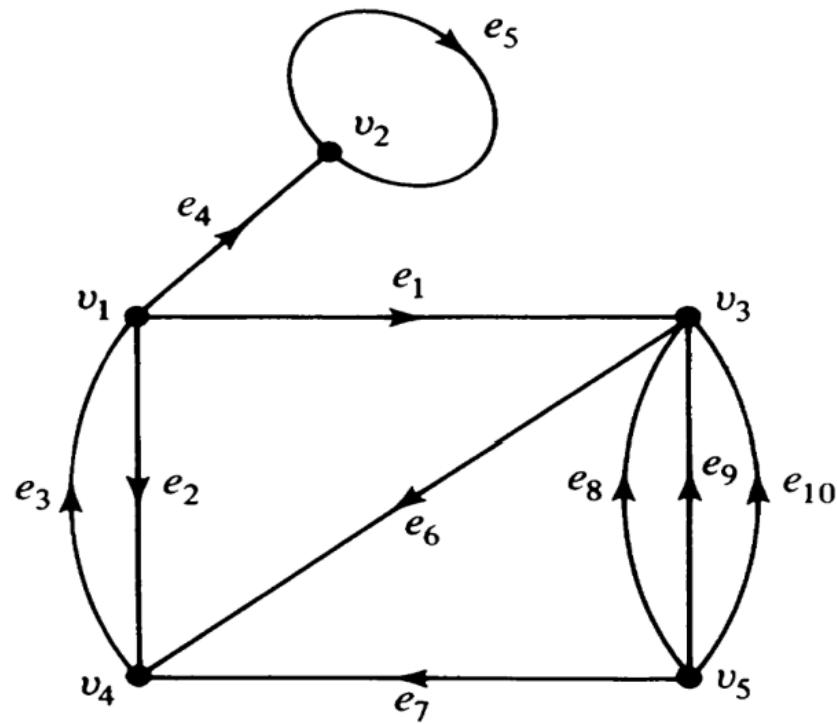
- **Grafos Direcionados (Digrafos)** : Grafos direcionados, também conhecidos como digrafos, são tipos de grafos onde as **arestas têm uma direção associada**, indicando uma relação unidirecional entre dois vértices.
- **Grau de Entrada e Saída**: Cada vértice em um grafo direcionado tem dois tipos de graus –
  - o grau de entrada (o número de arestas que entram no vértice)
  - e o grau de saída (o número de arestas que saem do vértice).
  - **Uma fonte**: quando o grau de entrada é nulo
  - **Um sumidouro**: Quando o grau de saída é nulo.
- Duas arestas são **paralelas** se elas incidem nos mesmos vértices e possuem a **mesma orientação**.

# Teoria dos Grafos – Grafos Direcionados (Digrafos)

- **Grafos Direcionados (Digrafos)** : Grafos direcionados, também conhecidos como digrafos, são tipos de grafos onde as **arestas têm uma direção associada**, indicando uma relação unidirecional entre dois vértices.
- Aplicações:
  - **Sistemas de Navegação e Mapas**: Para representar ruas de mão única.
  - **Redes Sociais**: Para indicar relações assimétricas, como seguidores e seguidos.
  - **Análise de Redes**: Em telecomunicações, redes de computadores e outros sistemas de fluxo de informação.

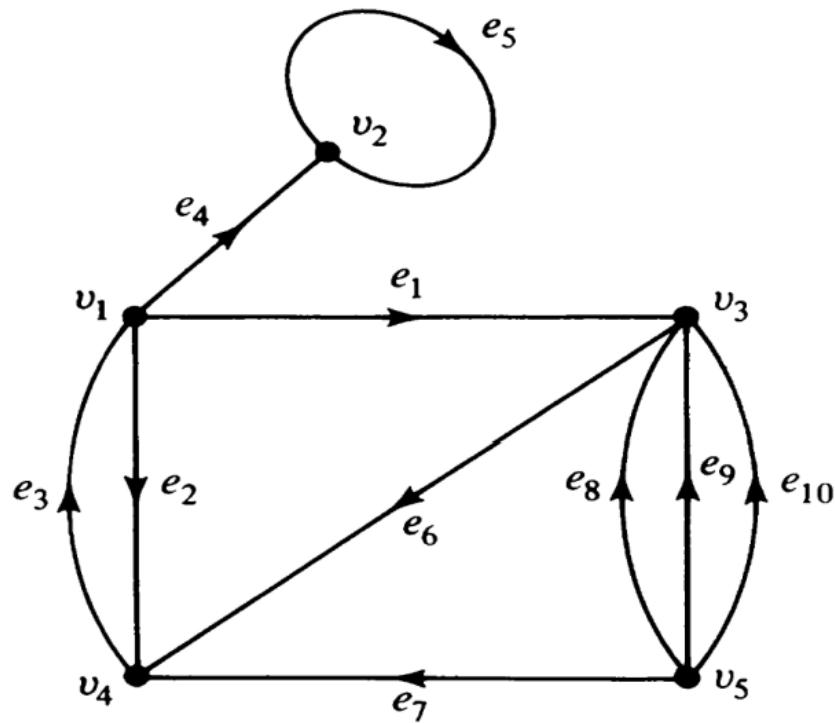
# Teoria dos Grafos – Grafos Direcionados (Digrafos)

- **Grafos Direcionados (Digrafos)** : Grafos direcionados, também conhecidos como digrafos, são tipos de grafos onde as **arestas têm uma direção associada**, indicando uma relação unidirecional entre dois vértices.



# Teoria dos Grafos – Grafos Direcionados (Digrafos)

- **Grafos Direcionados (Digrafos)** : Grafos direcionados, também conhecidos como digrafos, são tipos de grafos onde as **arestas têm uma direção associada**, indicando uma relação unidirecional entre dois vértices.



$$\begin{aligned}d_s(v_1) &= 3, & d_e(v_1) &= 1; \\d_s(v_2) &= 1, & d_e(v_2) &= 2; \\d_s(v_5) &= 4, & d_e(v_5) &= 0.\end{aligned}$$

# Teoria dos Grafos – Grafos Direcionados (Digrafos)

- **Tipos de Digrafos:**

- **Simples**: se não possui loops ou arestas paralelas;
- **Assimétrico** se possui no máximo uma aresta orientada entre cada par de vértices;
- **Simétrico** se para cada aresta  $(a, b)$  existe também uma aresta  $(b, a)$ ;
- **Completo Simétrico**; se  $G$  é simples e existe exatamente uma aresta direcionada de todo vértice para todos os outros vértices;
- **Completo Assimétrico** se  $G$  é assimétrico e existe exatamente uma aresta entre cada par de vértices;
- **Balanceado** se os graus de entrada e saídas são iguais para todos os vértices

# Teoria dos Grafos – Grafos Direcionados (Digrafos)

- **Tipos de Digrafos:**

- **Simples**: se não possui loops ou arestas paralelas;
- **Assimétrico** se possui no máximo uma aresta orientada entre cada par de vértices;
- **Simétrico** se para cada aresta  $(a, b)$  existe também uma aresta  $(b, a)$ ;
- **Completo Simétrico**; se  $G$  é simples e existe exatamente uma aresta direcionada de todo vértice para todos os outros vértices;
- **Completo Assimétrico** se  $G$  é assimétrico e existe exatamente uma aresta entre cada par de vértices;
- **Balanceado** se os graus de entrada e saídas são iguais para todos os vértices