CENTRO DE CIÊNCIA E TECNOLOGIA
LABORATÓRIO DE CIÊNCIAS MATEMÁTICAS
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

Árvores

Disciplina: Estrutura de Dados I

Prof. Fermín Alfredo Tang Montané

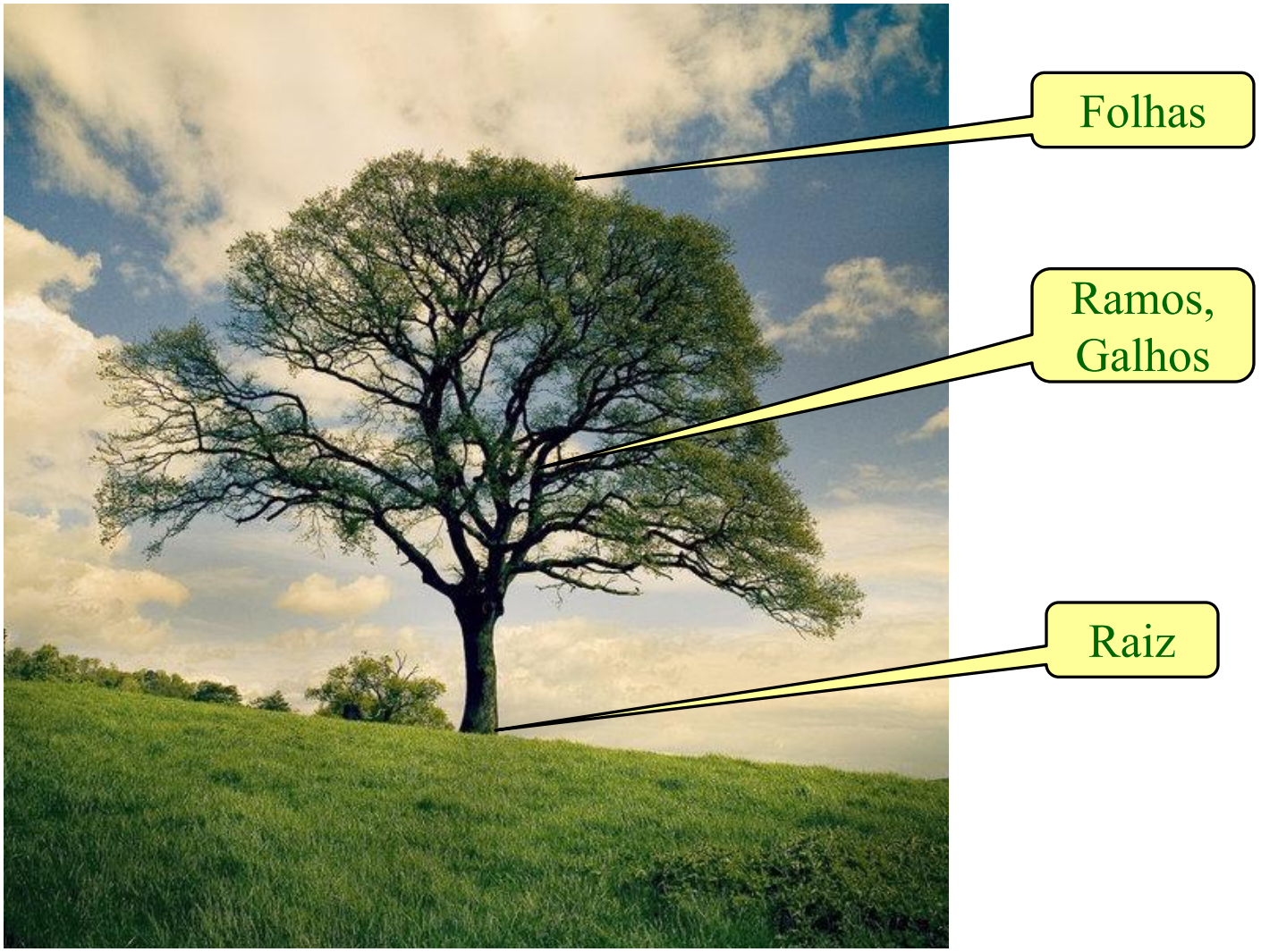
Curso: Ciência da Computação

Árvores

- As árvores são usadas amplamente na Ciência da Computação para representar fórmulas algébricas; como um método eficiente para buscar em grandes listas dinâmicas; e em aplicações diversas como sistemas de inteligência artificial e algoritmos de codificação.
- Apresenta-se conceitos básicos sobre as árvores.

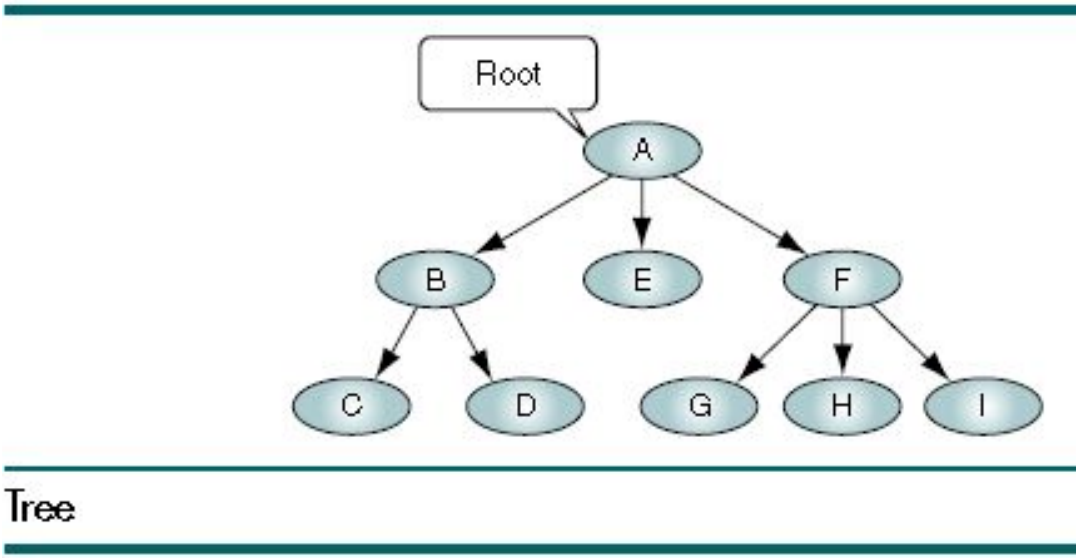
Árvores

Contexto



Árvores
Conceitos Básicos

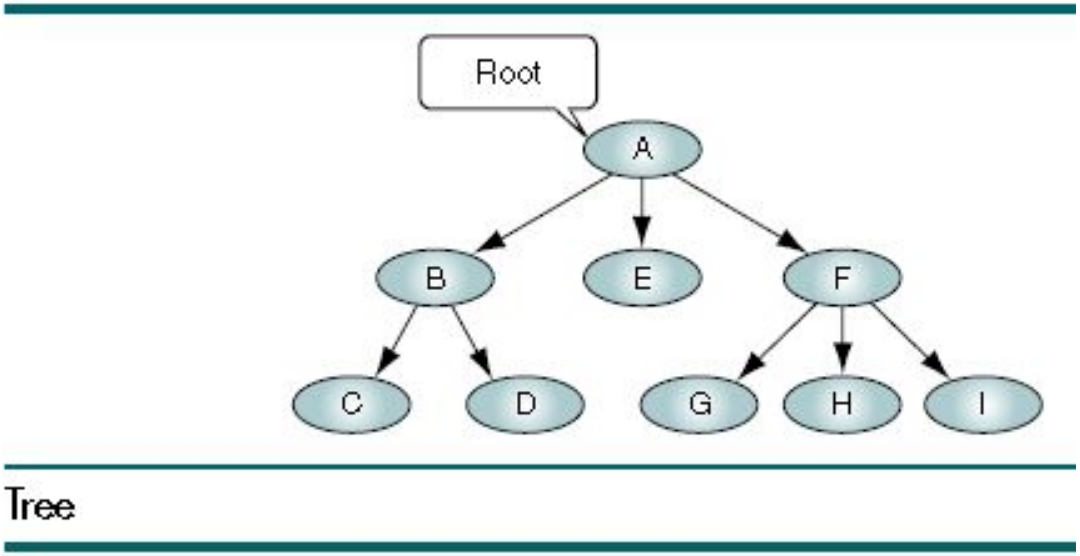
- Uma árvore consiste de um conjunto finito de elementos, chamados **nós**, e um conjunto finito de conexões direcionadas, chamadas de **ramos**, que conectam os nós.



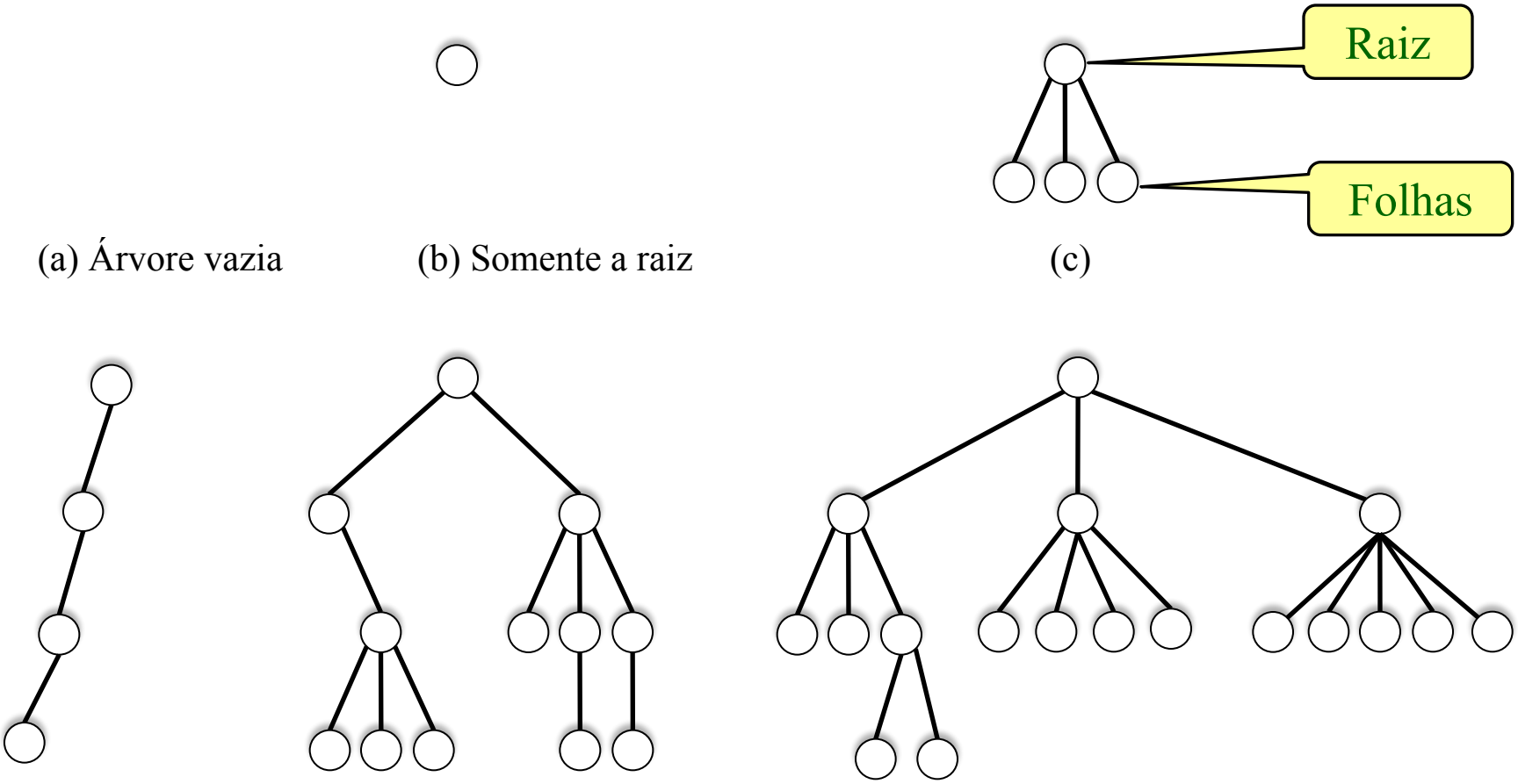
- O número de ramos associados a um nó é chamado de **grau do nó**. O grau de um nó pode ser decomposto em grau de entrada e grau de saída.
- O grau de entrada de um nó, corresponde ao número de ramos que entram nesse nó (ou apontados para esse nó)
- O grau de saída de um nó, corresponde ao número de ramos que saem desse nó.

Árvores
Conceitos Básicos

- Se a árvore não está vazia, o primeiro nó é chamado de raiz.
- Por definição, o **grau de entrada** da raiz é zero. Já, todos os outros nós da árvore devem ter grau de entrada exatamente igual a um. Ou seja, devem ter exatamente um predecessor.
- Por outro lado, todos os nós na árvore podem ter zero, um ou mais ramos saindo deles. Ou seja, **grau de saída** igual a zero, um ou mais. Assim, todo nó pode ter zero, um ou mais sucessores.



Árvores
Exemplos



(a) Árvore vazia

(b) Somente a raiz

(c)

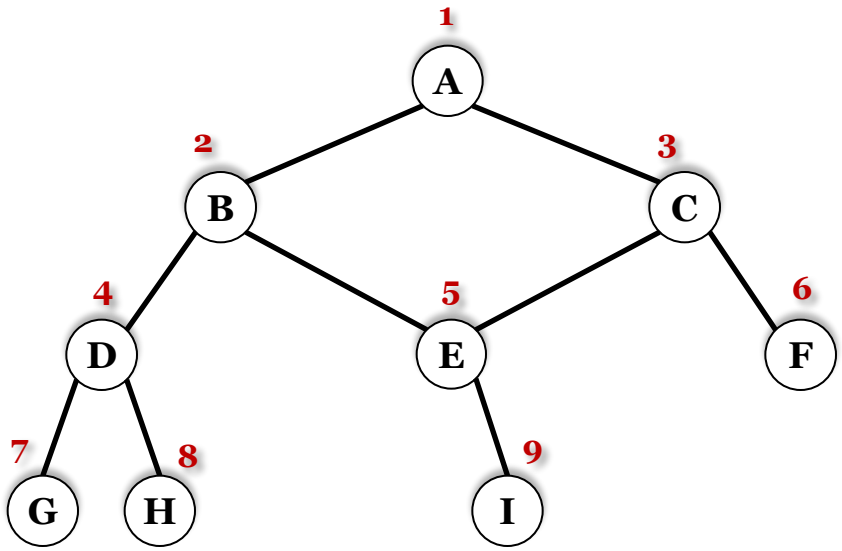
(d) Lista Encadeada

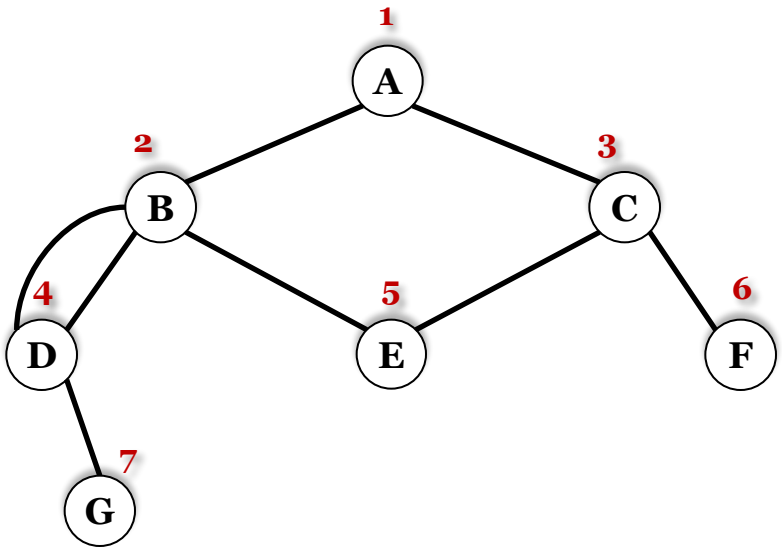
(e)

(f)

Árvores
Contra-exemplos

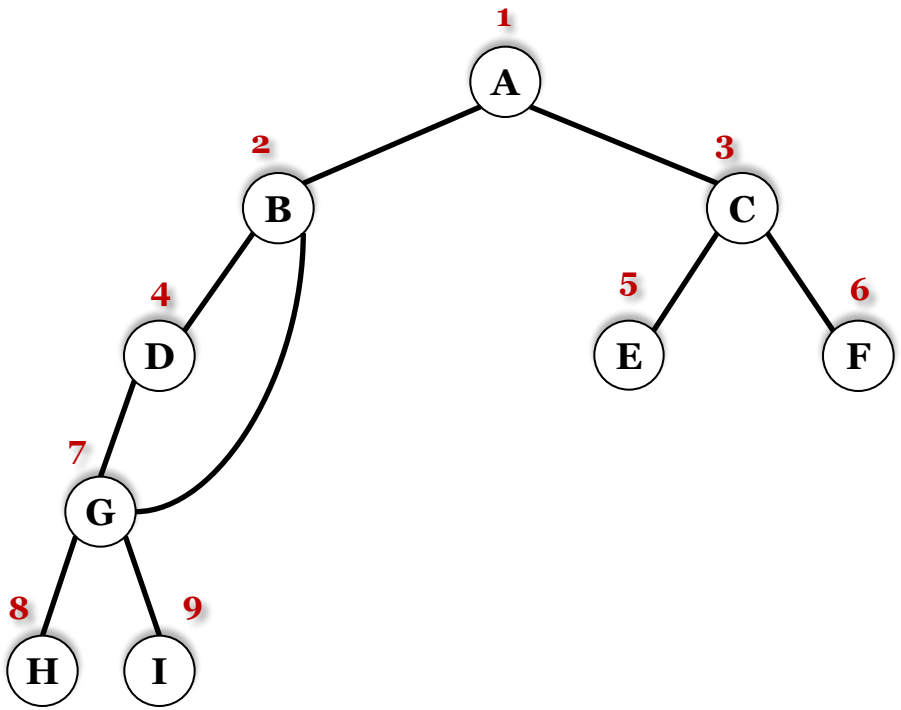
- Procure entender porque as seguintes estruturas não são árvores binárias.





Árvores
Contra-exemplos

- Procure entender porque as seguintes estruturas não são árvores binárias.



Árvores
Conceitos Básicos

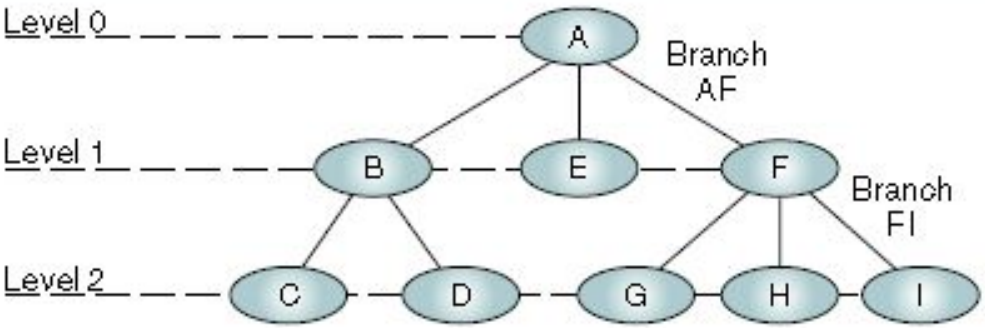
- Um nó é **pai** (*parent*) se possui nós sucessores, ou seja, se tem grau de saída maior que zero.
- Por outro lado, um nó com um predecessor é um **filho** *(child)*. Ou seja, com grau de entrada igual a um.
- Dois ou mais nós com o mesmo pai são chamados de irmaõs (sibblings).

- Em alguns textos, utiliza-se o termo nó avó (*grandparent*). No entanto, é mais comum utilizar o termo mais geral de **ascendente** (*ancestor*). Um nó ascendente de um nó é qualquer nó no caminho da raiz até esse nó. Já um nó **descendente** (*descendent*) é qualquer nó em um caminho partindo desde esse nó até um nó folha.

Árvores

Conceitos Básicos - Terminologia

- Um **caminho** é uma sequência de nós em que cada nó é adjacente ao próximo.
- Cada nó na árvore pode ser alcançado\acessado seguindo um único caminho a partir do nó raiz.
- Na figura, o nó folha I, pode ser alcançado a partir da raiz seguindo o caminho: A-F-I. Que compreende 2 ramos, AF e FI.

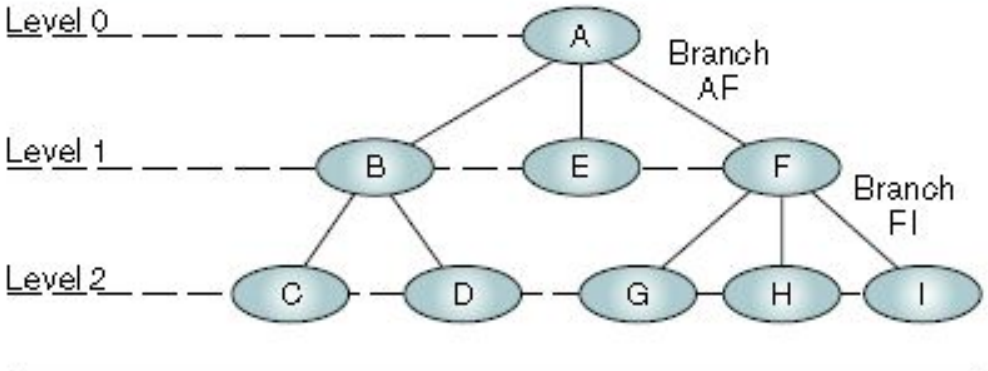


Tree Nomenclature

Árvores

Conceitos Básicos - Terminologia

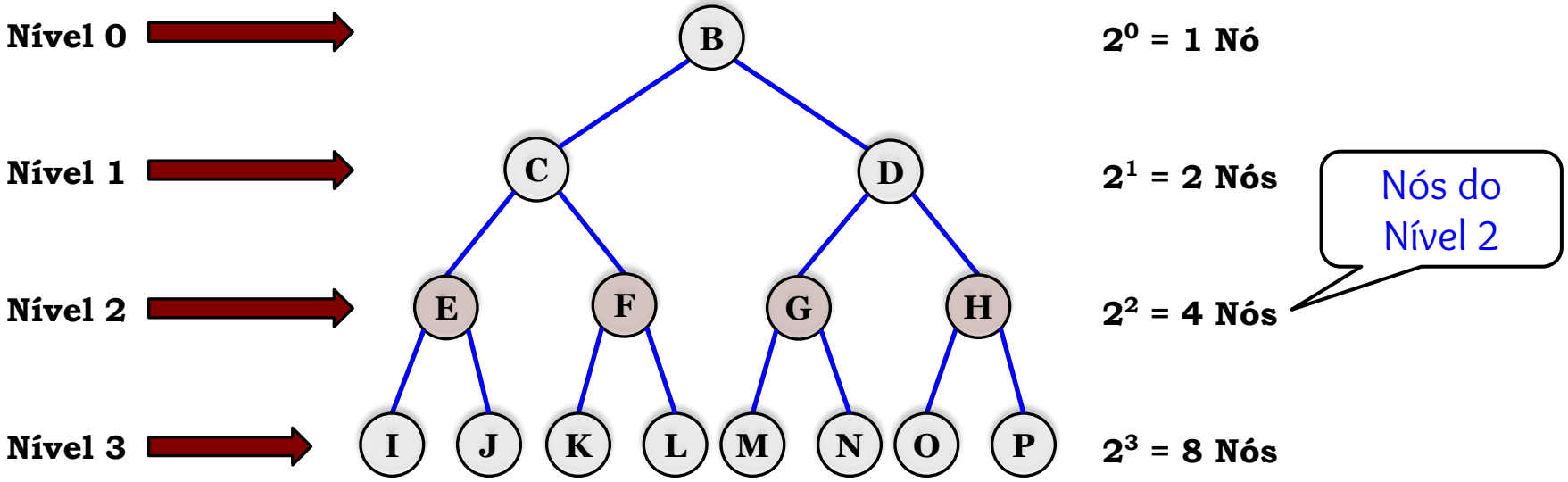
- O **nível** de um nó é a distância desde a raiz. Como a raiz tem distância zero desde se própria, a raiz se encontra no nível 0.
- Os filhos da raiz se encontram no nível 1. Pelo sua vez seus filhos se encontram no nível 2, e assim sucessivamente.
- Note a relação entre os nós irmaõs (*sibblings*). O nós irmãos sempre se encontram no mesmo nível. No entanto, nem todos os nós que pertencem ao mesmo nível serão necessariamente irmaõs.
- Por exemplo, no nível 2, os nós C e D são irmaõs, assim como G, H e I. No entanto, os nós D e G não são irmãos devido a que tem pais diferentes.



Árvores

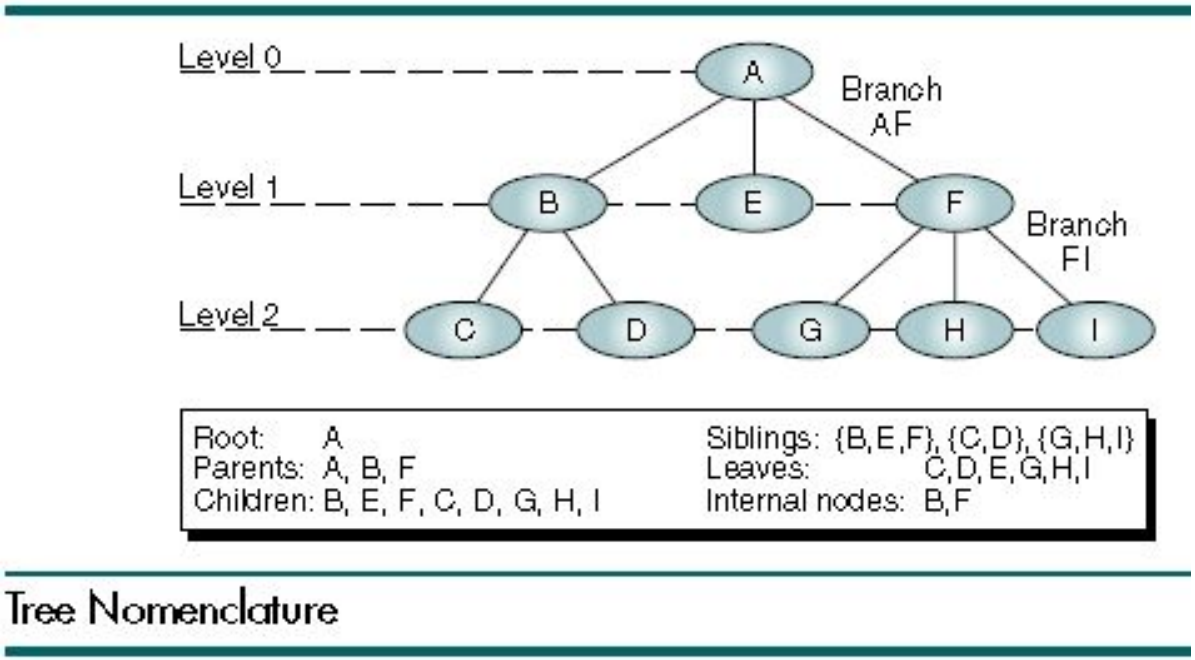
Conceitos Básicos - Terminologia

- **Nível de um nó.-** É um número que indica a hierarquia do nó. O seu valor esta diretamente relacionado à posição que o nó ocupa na árvore. Quanto menor o nível de um nó, maior é a sua hierarquia.
- **Nível k em uma árvore**.- O número máximo de nós no nível k de uma árvore binária é igual a 2^k.



Árvores
Conceitos Básicos - Terminologia

- A **altura *H*,** de uma árvore é igual ao nível da folha mais distante desde a raiz mais um. Por definição a altura de uma árvore vazia é -1.
- No exemplo, as folhas mais distantes se encontram no nível 2. Com isso, a altura de árvore é 3.
- Em alguns textos, considera-se que como a árvore é representada invertida então trata-se do conceito de **profundidade** (*depth*) em vez de altura.

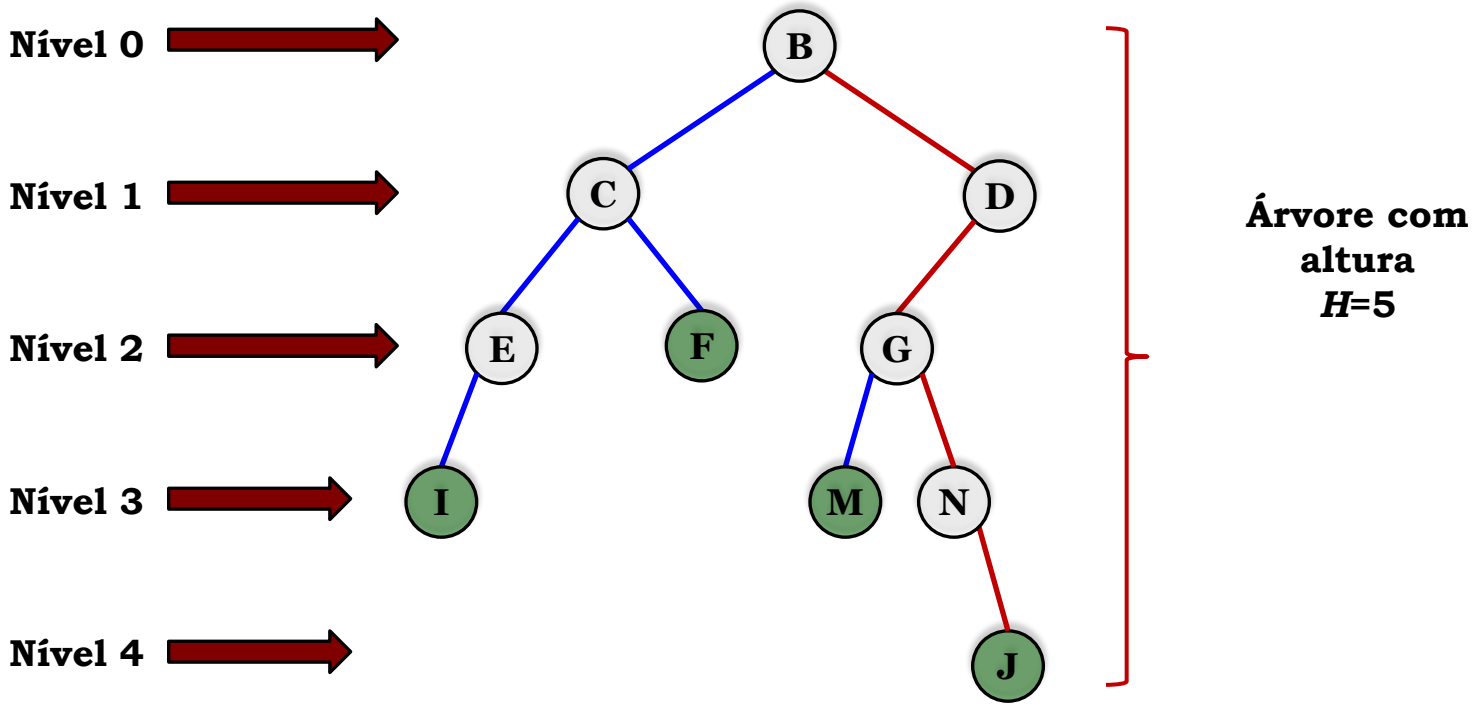


Tree Nomenclature

Árvores

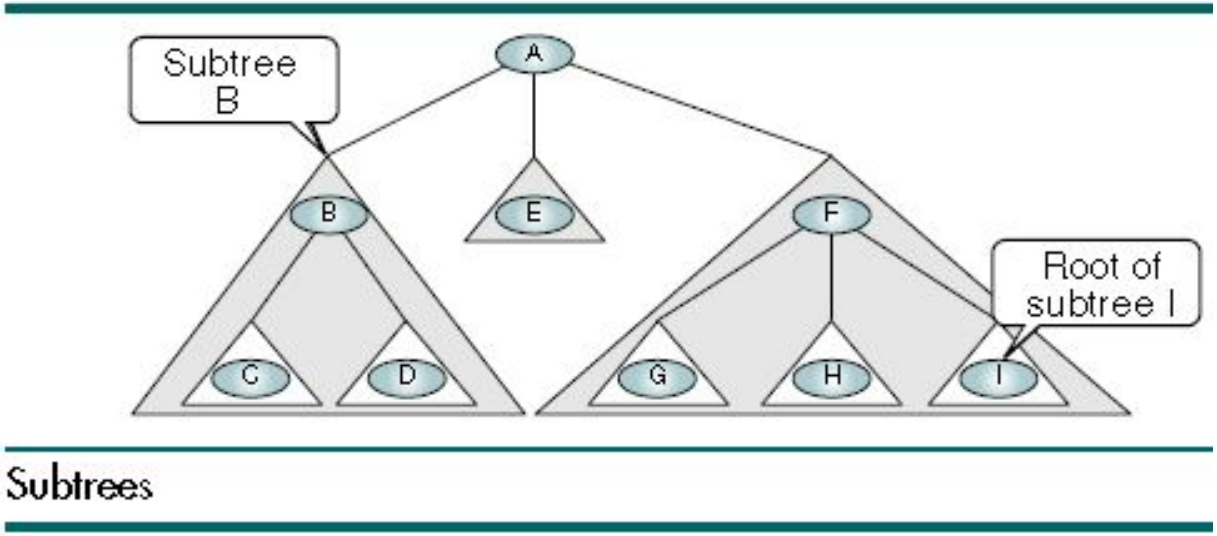
Conceitos Básicos - Terminologia

- A **altura *H*,** de uma árvore é igual nível da folha mais distante desde a raiz mais um. Por definição a altura de uma árvore vazia é -1.



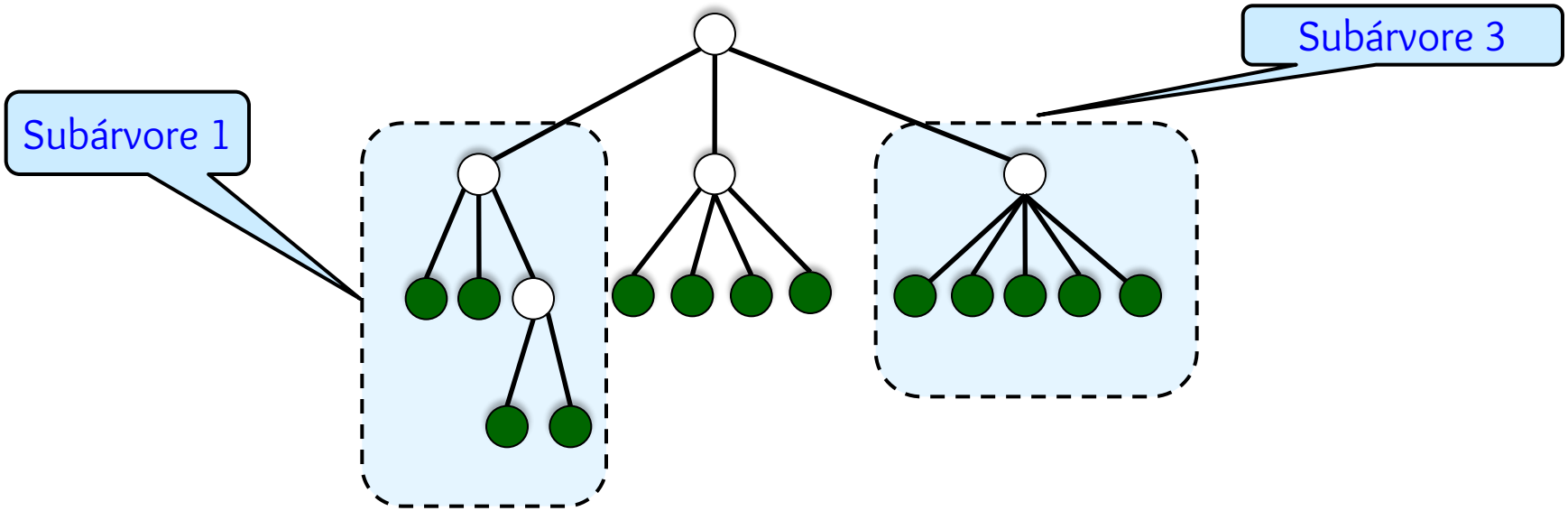
Árvores
Definição Recursiva

- Uma árvore pode ser dividida em **sub-árvores**. Uma sub-árvore é qualquer estrutura conectada abaixo da raiz.
- O primeiro nó de uma sub-árvore é a raiz da sub-árvore. Ela costuma servir para denotar a subárvore.
- No exemplo, BCD é uma subárvore, assim como E, FGHI. Um único nó pode ser uma sub-árvore. A subárvore B pode ser dividida em dois subárvores, C e D, enquanto a sub-árvore D contém as sub-árvores G, H e I.



Árvores
Definição Recursiva

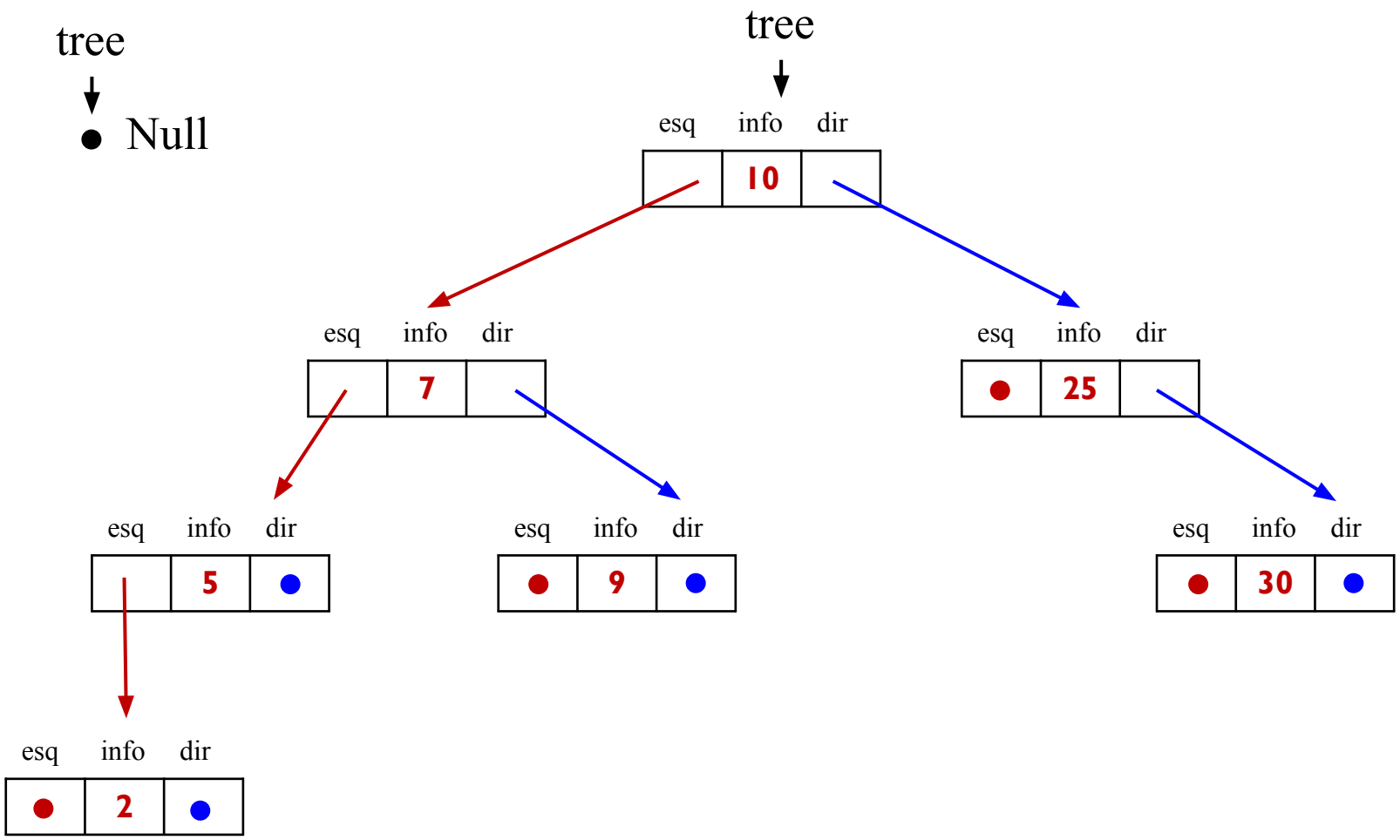
- Com conceito de sub-árvore nos leva a uma **definição recursiva** de árvore.
- Uma árvore é um conjunto de nós que pode:
- i) Estar vazio;
- ii) Ter um nó específico designado como nó raiz, do qual descendem de maneira hierárquica, zero, um ou mais sub-árvores que também são árvores.



Árvores

Representação de Arvores

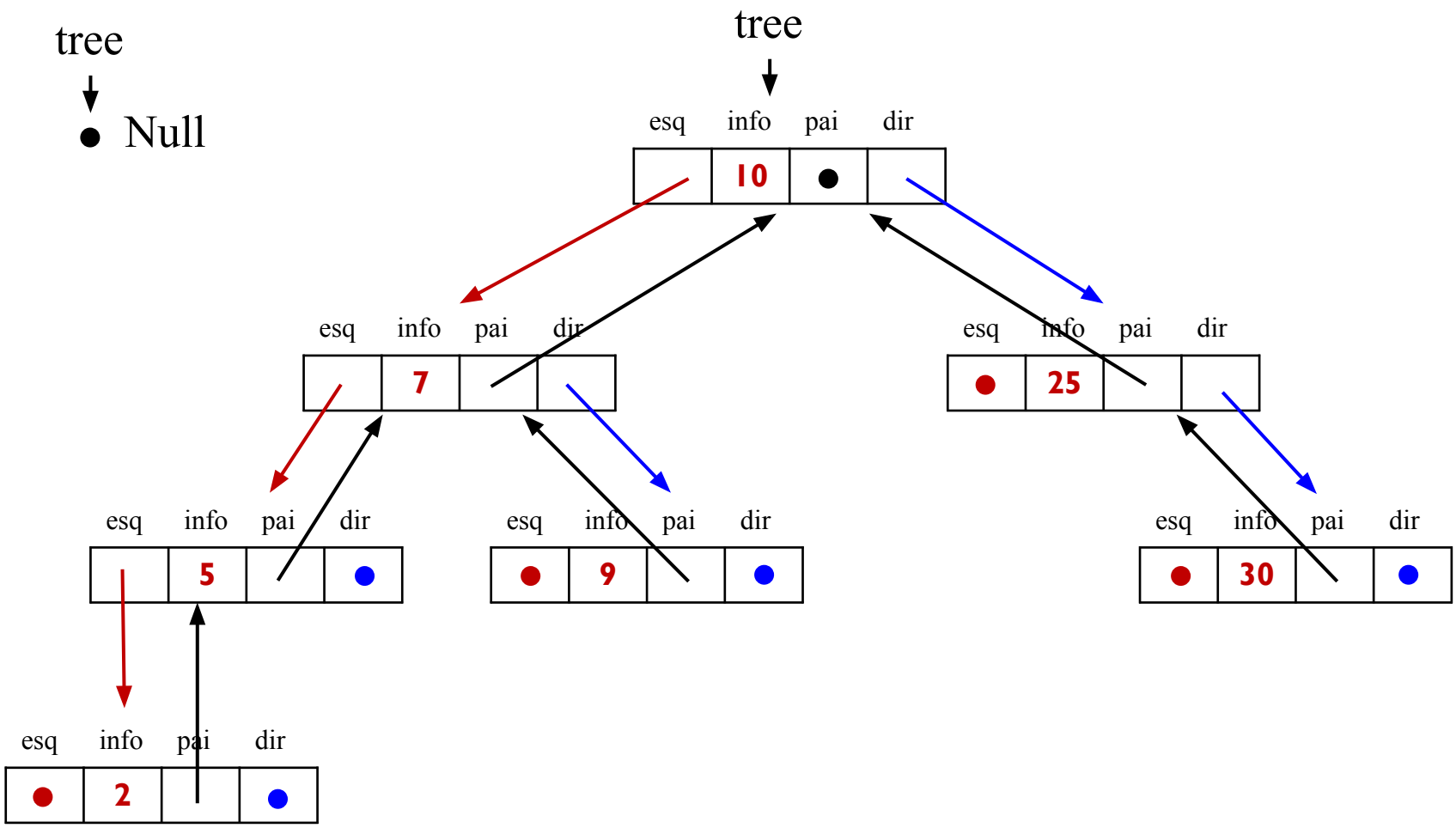
- Encadeamento simples descendente.



Árvores

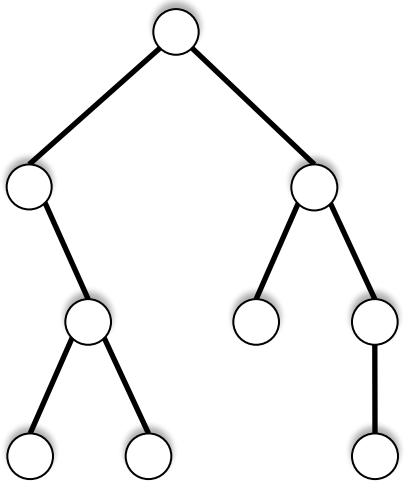
Representação de Arvores

- Encadeamento "duplo" descendente e ascendente.

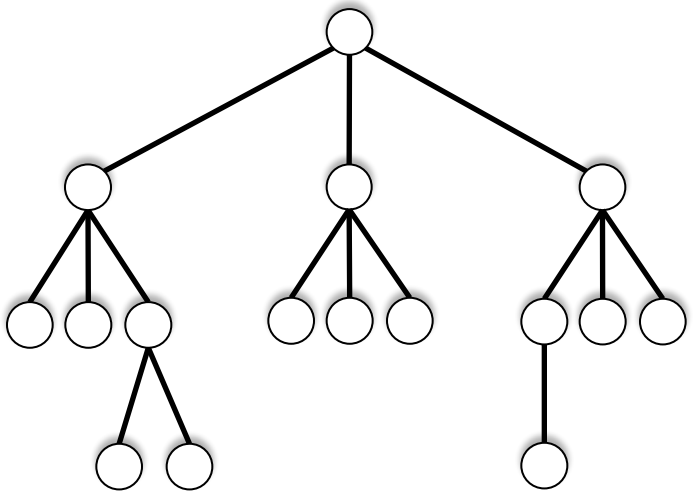


Árvores
Classificação

- As árvores podem ser classificadas pelo número máximo de filhos que um nó pode ter. Por exemplo, árvores binárias e ternárias, etc.



(a) Árvore Binária

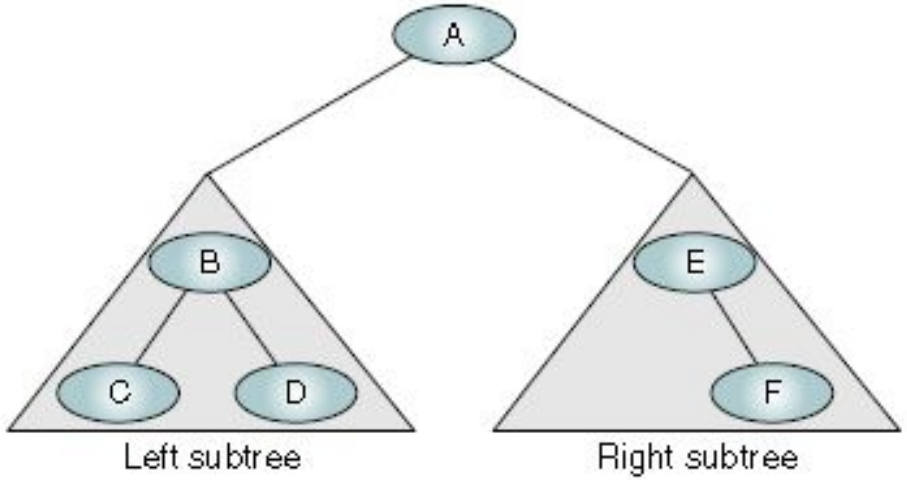


(b) Árvore Ternária

Árvores Binárias

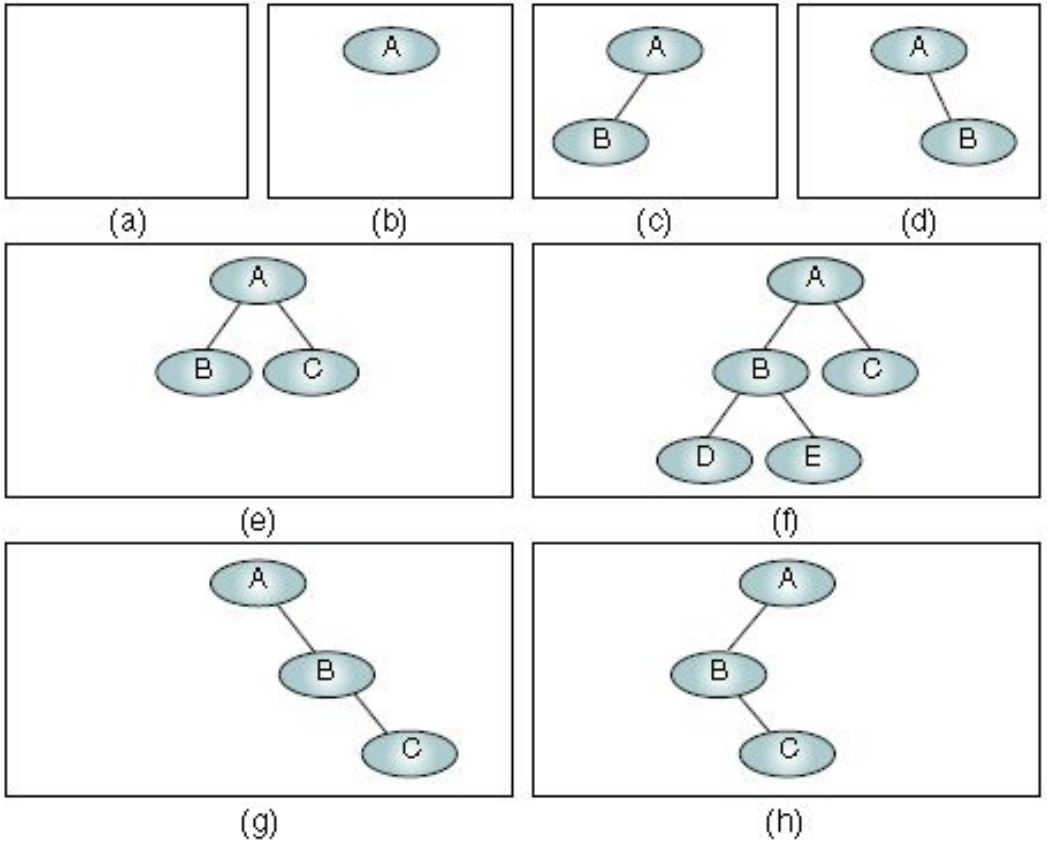
Definição

- Uma **árvore binária** é uma árvore na qual cada nó pode ter até duas subárvores. O grau de saída de cada nó pode ser no máximo dois.
- Assim, cada nó pode ter zero, um ou duas subárvores.
- As subárvores são designadas como sub-arvore esquerda e direita.
- Na figura, mostra-se uma árvore binária com duas subárvores. Note que cada sub-árvore é pela sua vez uma árvore binária.



Árvores Binárias
Exemplos

- A figura mostra exemplos de árvores binárias. A primeiro exemplo (a) mostra uma árvore nula (ou vazia), que não possui nós.
- Observe que a simetria não é um requisito em árvores.



Árvores Binárias
Propriedades

- A Altura de uma árvore binária pode ser determinada matematicamente.

- **Altura máxima.** Dada uma árvore binária com N nós armazenados, a altura máxima H_{max} é:

$$H_{\max} = N$$

- **Altura mínima.** Dada uma árvore binária com N nós armazenados, a altura mínima H_{min} é:

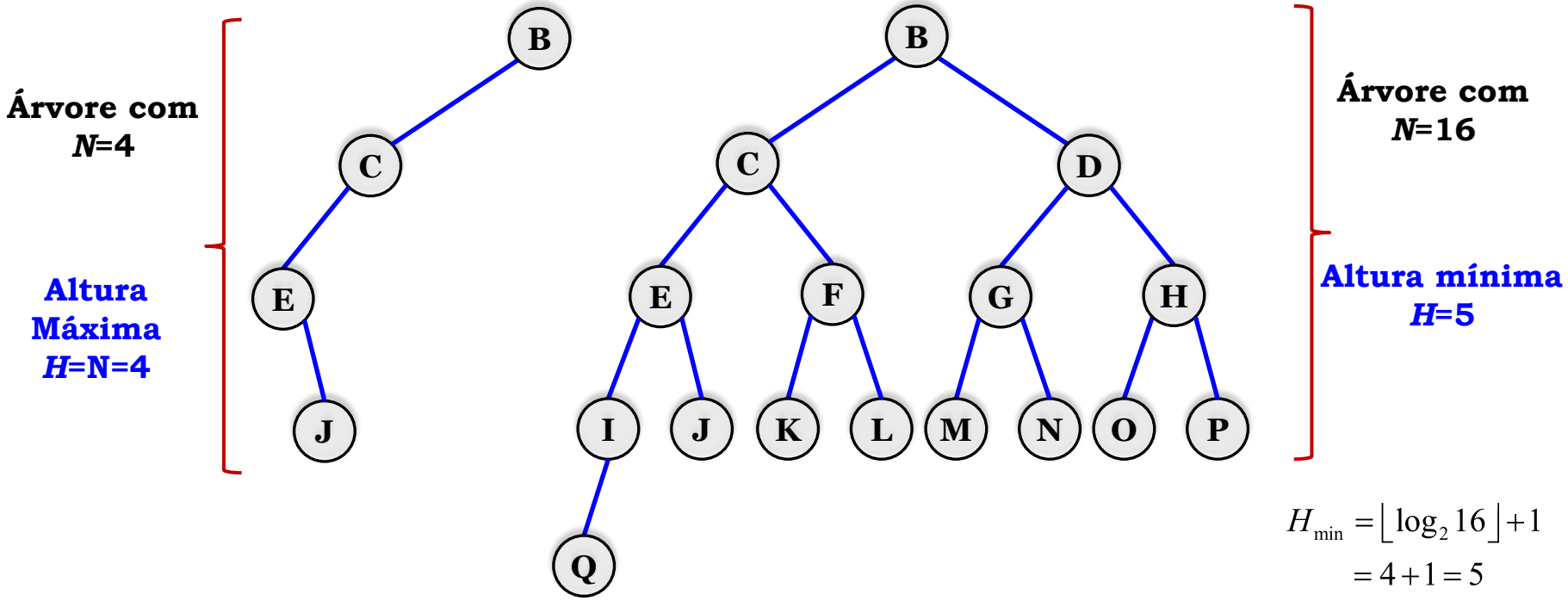
$$H_{\min} = \lfloor \log_2 N \rfloor + 1$$

Árvores Binárias
Propriedades

- A altura da árvore. Mínima. Máxima.

$$H_{max} = N$$

$$H_{min} = \lfloor \log_2 N \rfloor + 1$$

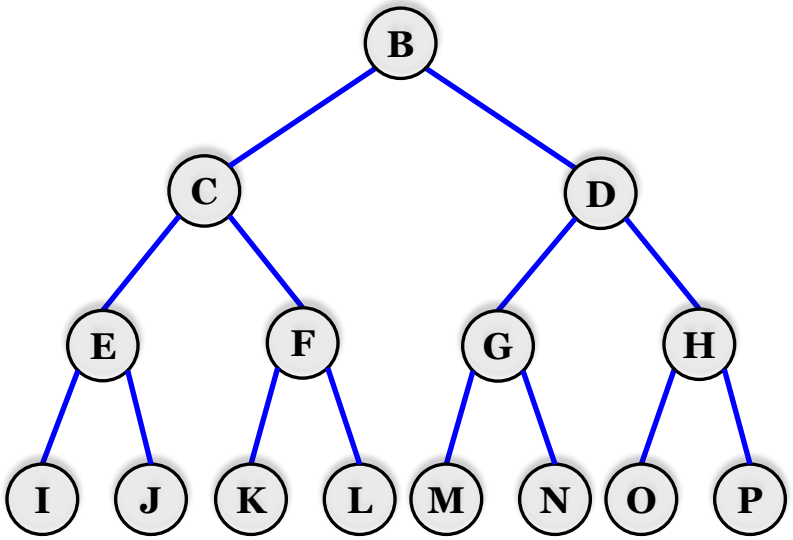


$$H_{min} = \lfloor \log_2 16 \rfloor + 1$$
$$= 4 + 1 = 5$$

Árvores Binárias
Propriedades

- A altura da árvore. Mínima.

$$H_{\min} = \lfloor \log_2 N \rfloor + 1$$



Árvore com *N*=15

Altura mínima *H*=4

$$H_{\min} = \lfloor \log_2 15 \rfloor + 1$$
$$= \lfloor 3,9068 \rfloor + 1 = 4$$

$$\log_2 15 = \frac{\log_{10} 15}{\log_{10} 2} = 3,90689$$
$$= \lfloor 3,9068 \rfloor + 1 = 4$$

Árvores Binárias
Propriedades

- O número de nós de uma árvore binária também pode ser determinada matematicamente.
- **Número mínimo de nós.** Dada uma árvore binária com altura H, o número mínimo de nós na árvore é dada por N_{min} é:

$$N_{\min} = H$$

- **Número máximo de nós.** Dada uma árvore binária com altura H, o número máximo de nós na árvore é dada por N_{max} é:

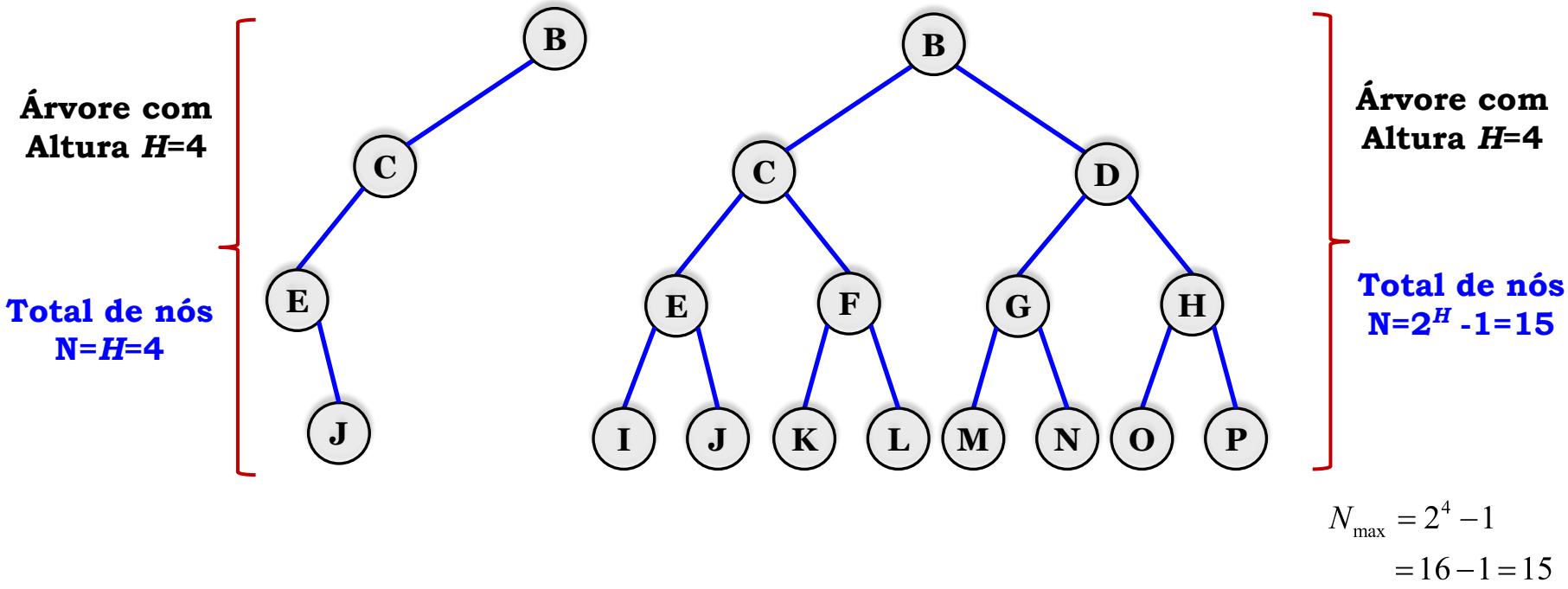
$$N_{\max} = 2^{H} - 1$$

Árvores Binárias
Propriedades

- O número de nós. Mínimo. Máximo.

$$N_{min} = H$$

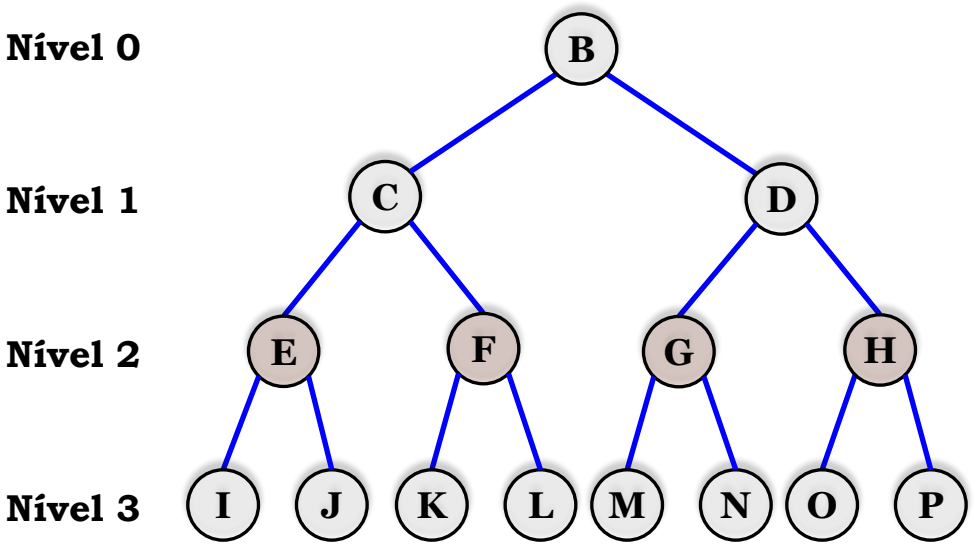
$$N_{max} = 2^H - 1$$



$$N_{max} = 2^4 - 1$$
$$= 16 - 1 = 15$$

Árvores Binárias
Propriedades

- Observe que:



$$N_{\max} = 2^H - 1$$

$$\sum_{k=0}^{d} 2^k = 2^0 + 2^1 + \ldots + 2^d = 2^{d+1} - 1$$

$$\sum_{k=0}^{d} a^k = \frac{a^{d+1} - 1}{a - 1}$$

Árvores Binárias
Propriedades

- Relação entre o número de nós e a altura *H* (ou número de níveis *L*).

$$N = 2^H - 1$$

$$H = \log_2(N+1)$$

Número de Nós (N)	Número de Níveis (L)
1	1
3	2
7	3
15	4
31	5
1.023	10
32.767	15
1.048.575	20
33.554.431	25
1.073.741.823	30

Árvores Binárias
Propriedades

- Para determinar se uma árvore esta balanceada, calcula-se seu fator de balanceamento. O **fator de balanceamento** em uma árvore binária é a diferença de alturas entre as subárvores esquerda e direita.
- Se definimos a altura da sub-árvore esquerda como H_E e a altura da sub-árvore direita como H_D o fator de balanceamento B é determinado pela fórmula:

$$B = H_E - H_D$$

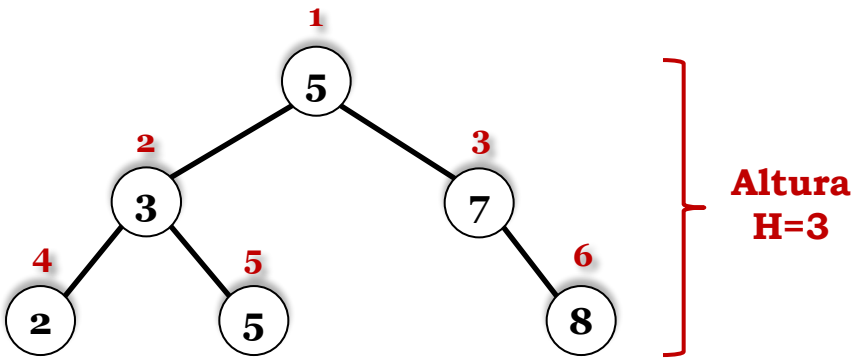
- Em uma árvore binária balanceada, as alturas das subárvores diferem por no mais do que um. Ou seja, o fator de balanceamento é 0, +1 ou -1.

Árvores Binárias
Propriedades

- Em uma árvore binária balanceada, as alturas das subárvores diferem por no mais do que um. Ou seja, o fator de balanceamento é 0, +1 ou -1.

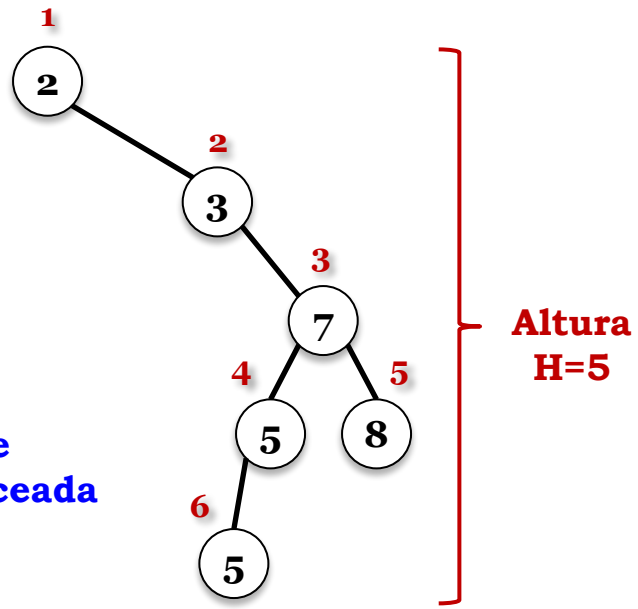
$$B = H_E - H_D$$

$B = 2 - 2 = 0$



Árvore Balanceada

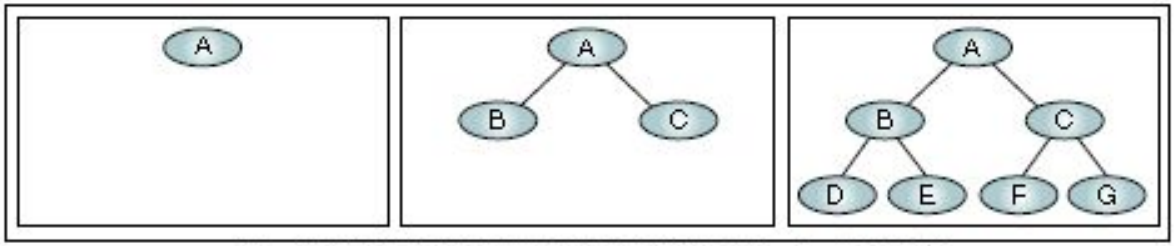
$B = 0 - 4 = -4$



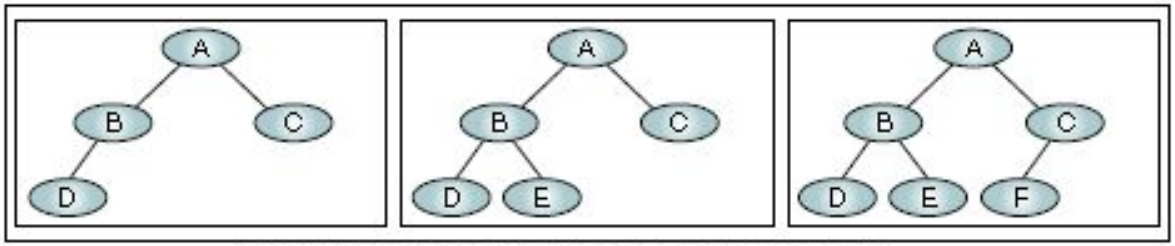
Árvore Não Balanceada

Árvores Binárias
Propriedades

- **Árvores Binárias Completas e Quase Completas.-**
- Uma **árvore binária completa** tem o número máximo de nós para a sua altura. Esse número de nós é atingido quando o último nível está cheio.
- Uma árvore binária é considerada quase completa se possui a altura mínima dados os seus nós e todos os nós do último nível se encontram à esquerda.



(a) Complete trees (at levels 0, 1, and 2)



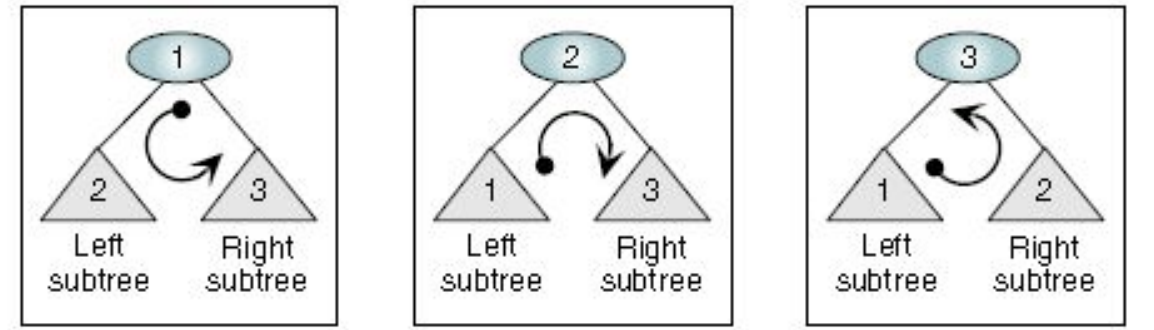
(b) Nearly complete trees (at level 2)

Percurso de Árvores Binárias
(Binary Tree Transversals)

- O percurso de uma árvore binária requer que cada nó da árvore seja processado exatamente uma vez em uma sequência predeterminada.
- Existem duas abordagens gerais para definir a sequência de percurso:
  - Profundidade Primeiro (*depth first*);
  - Largura Primeiro (*Breadth first*).
- No percurso em profundidade primeiro, o processamento dos nós segue uma estratégia vertical desde a raiz. Escolhe-se um caminho em profundidade desde a raiz até um descendente mais distante nos últimos níveis. Para definir o caminho escolhe-se um primeiro filho da raiz. Todos os descendentes mais distantes desse filho serão visitados primeiro, depois os menos distantes. Quando todos os descendentes de um filho forem visitados inicia-se um novo caminho a partir de outro filho.
- No percurso em largura primeiro, o processamento dos nós segue uma estratégia horizontal (ou por níveis) desde a raiz. Primeiro, todos os filhos da raiz são visitados. Depois todos os filhos de cada filho serão visitados e assim sucessivamente, até que todos os nós sejam processados. Neste tipo de percurso, cada nível é processado completamente antes de se iniciar o processamento do seguinte nível.

Percursos em Profundidade Primeiro
(Depth-First Transversals)

- Dado que uma árvore binária consiste de uma raiz, uma sub-árvore esquerda e uma sub-árvore direita, é possível definir seis diferentes sequências de **percursos em profundidade**.
- A Ciência de Computação usa com mais frequência três destas sequências às quais designa nomes padrão. As outras sequências são facilmente derivadas.



(a) Preorder traversal (b) Inorder traversal (c) Postorder traversal

- Para designar estes percursos usa-se a seguinte notação: (N) para o nó processado, (L) visita à sub-árvore esquerda e (R) visitar à sub-árvore direita.
  - O percurso pré-ordem é denotado como NLR.
  - O percurso in-ordem é denotado como LNR.
  - O percurso pós-ordem é denotado como LRN.

Percurso Pré-Ordem NLR
(Preorder Transversal NLR)

- No **percurso pré-ordem**, a raiz é processada primeiro, seguida da subárvore esquerda e depois pela sub-árvore direita.
- O prefixo "pré", que deve ser entendido como "antes de" refere-se ao processamento do nó raiz. A raiz é processada antes das sub-árvores.

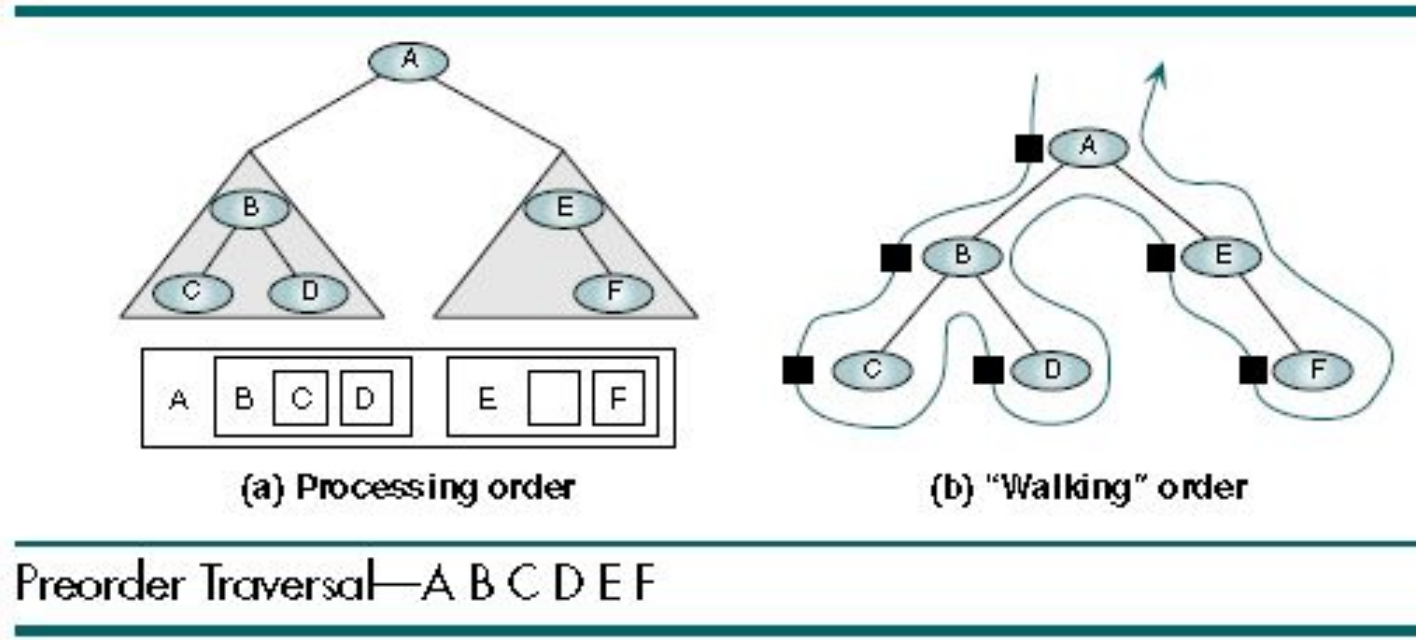
- Dadas as caraterísticas recursivas das árvores, é possível implementar os percursos de maneira recursiva.

```
Algorithm preOrder (root)
Traverse a binary tree in node-left-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   process (root)
   2   preOrder (leftSubtree)
   3   preOrder (rightSubtree)
2  end if
end preOrder
```

- No caso do pré-ordem, primeiro processamos a raiz, depois a sub-árvore esquerda e depois a sub-árvore direita. A sub-árvore esquerda é processada recursivamente, assim como a direita.

Percurso Pré-Ordem NLR
(Preorder Transversal NLR)

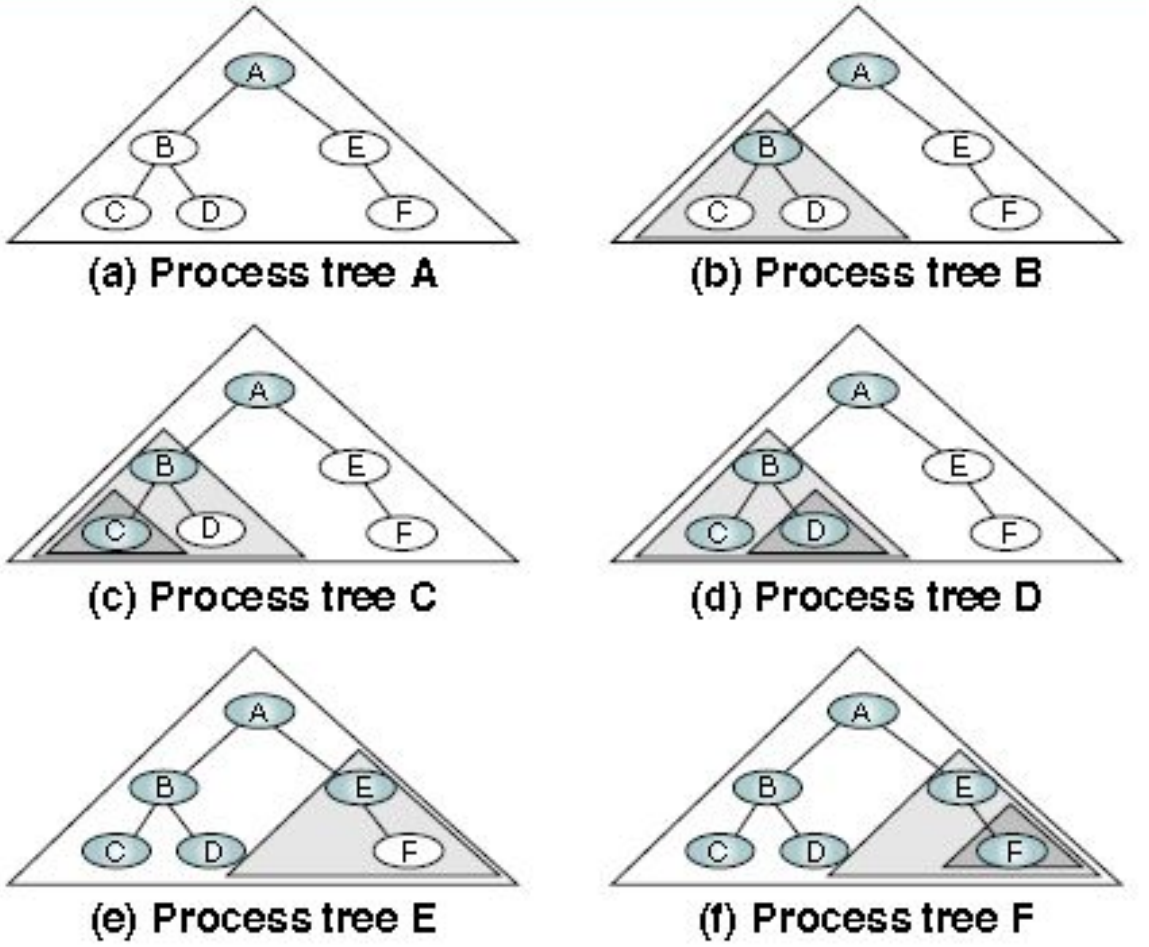
- O exemplo ilustra o percurso pré-ordem.



- Observe o percurso seguido. Processamos um nó sempre na primeira vez que visitamos ele. Observe a posição do quadrado preto.

Percurso Pré-Ordem NLR
(Preorder Transversal NLR)

- A figura ilustra as chamadas recursivas realizadas pelo percurso pre-ordem na forma de um triangulo.
- Além disso, ilustra-se o processamento de cada nó (nó sombreado).



Percurso Em-Ordem LNR
(Inorder Transversal LNR)

- O **percurso Em-ordem** processa primeiro a sub-árvore esquerda, depois a raiz, e finalmente a sub-árvore direita.
- O prefixo "Em", que deve ser entendido como "no meio de" refere-se ao processamento do nó raiz. A raiz é processada entre as duas sub-árvores.

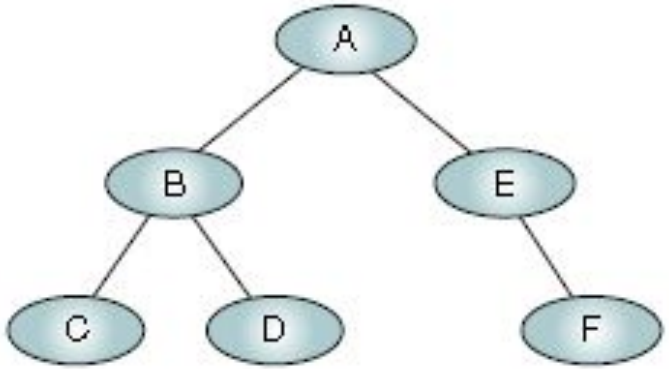
- O algoritmo pode ser implementado de maneira recursiva.

```
Algorithm inOrder (root)
Traverse a binary tree in left-node-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   inOrder (leftSubTree)
   2   process (root)
   3   inOrder (rightSubTree)
2  end if
end inOrder
```

- No caso do em-ordem, primeiro processamos sub-árvore esquerda, depois a raiz e finalmente a sub-árvore direita.

Percurso Em-Ordem LNR
(Inorder Transversal LNR)

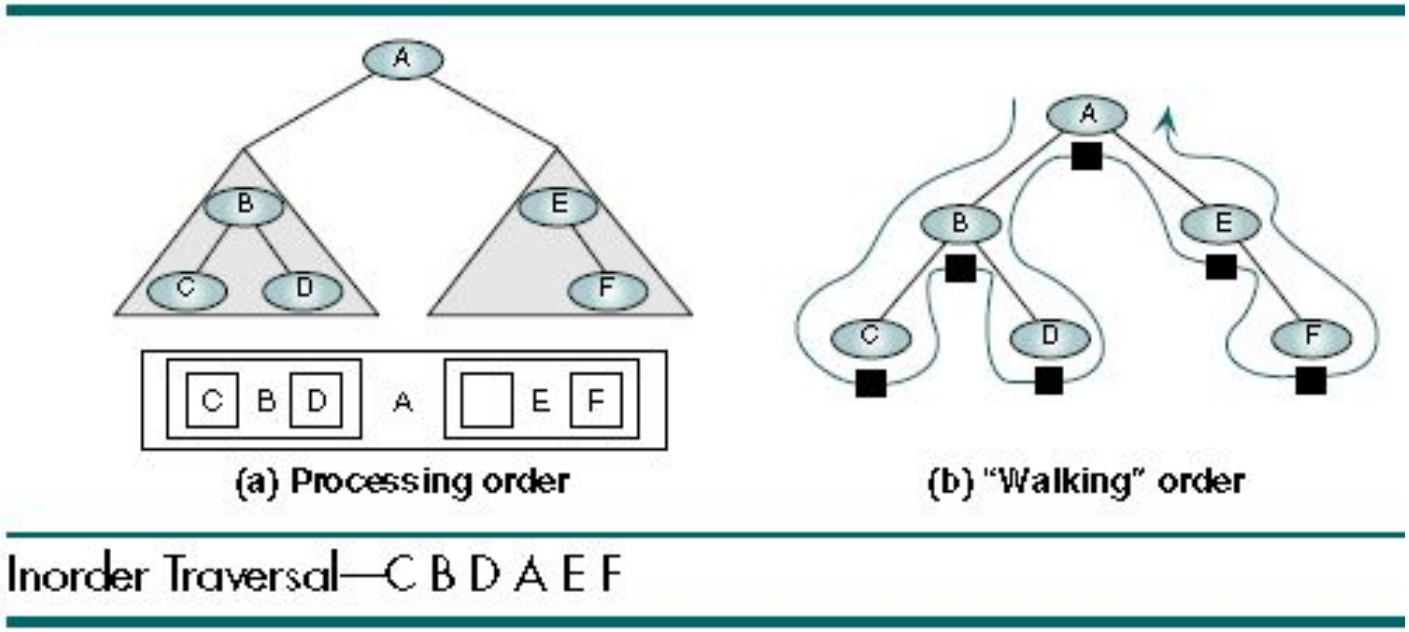
- O exemplo ilustra o percurso Em-ordem.



- Como a subárvore esquerda deve ser processada primeiro, fazemos um percurso desde a raiz até a folha esquerda mais distante sem processar nenhum nó. Após processar o nó C, processamos seu pai, o nó B e depois o nó D.

Percurso Em-Ordem LNR
(Inorder Transversal LNR)

- O exemplo ilustra o percurso Em-ordem.



- Observe que seguimos o mesmo percurso, no entanto somente processamos um nó a segunda vez que visitamos ele. Observe a mudança na posição do quadrado preto.

Percurso Pós-Ordem LRN
(Posorder Transversal LRN)

- O **percurso Pós-ordem** processa o nó raiz depois de processar as sub-árvores esquerda e direita.
- O prefixo "Pós", que deve ser entendido como "depois de" refere-se ao processamento do nó raiz. A raiz é processada depois das duas sub-árvores.

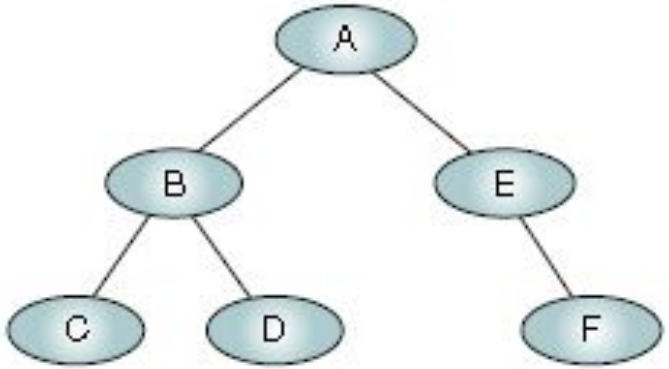
- O algoritmo pode ser implementado de maneira recursiva.

```
Algorithm postOrder (root)
Traverse a binary tree in left-right-node sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1  if (root is not null)
   1   postOrder (left subtree)
   2   postOrder (right subtree)
   3   process (root)
2  end if
end postOrder
```

- No caso do pós-ordem, primeiro processamos sub-árvore esquerda, depois sub-árvore direita e finalmente a raiz.

Percurso Pós-Ordem LRN
(Posorder Transversal LRN)

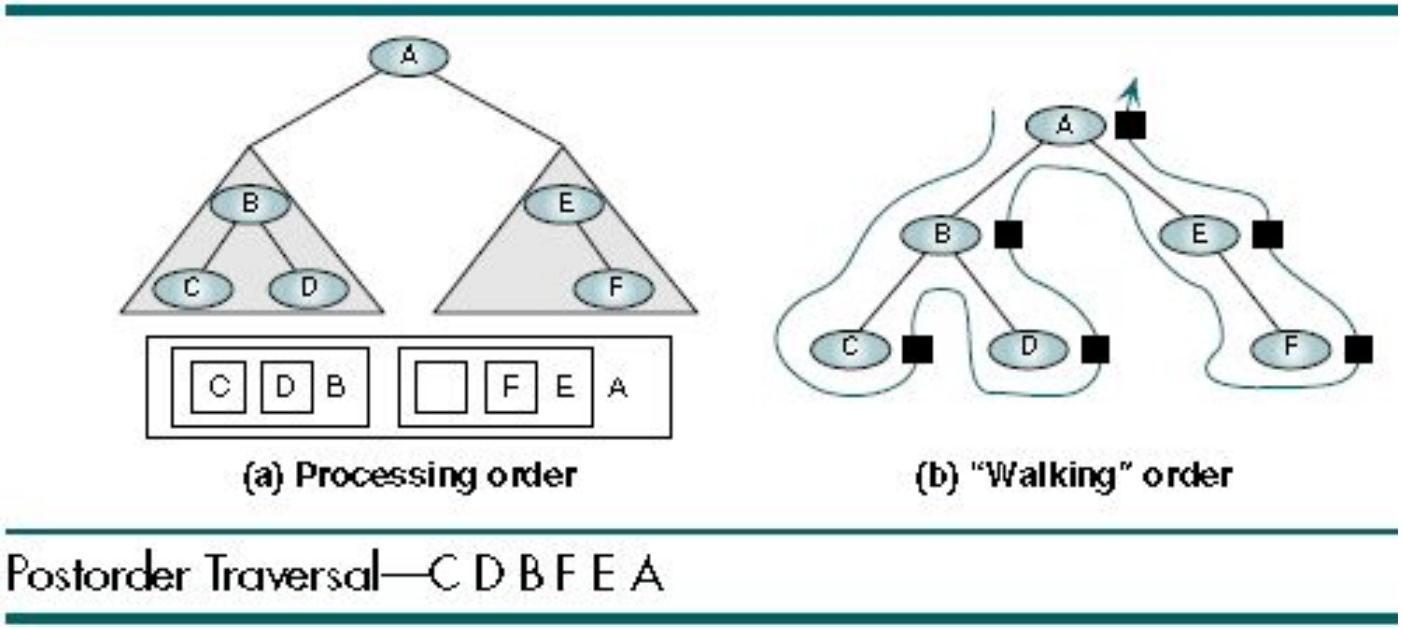
- O exemplo ilustra o percurso Pós-ordem.



- Como a subárvore esquerda deve ser processada primeiro, fazemos um percurso desde a raiz até a folha esquerda mais distante sem processar nenhum nó. Após processar o nó C, processa seu irmão direito D e depois o nó pai B.

Percurso Pós-Ordem LRN
(Posorder Transversal LRN)

- O exemplo ilustra o percurso Pos-ordem.



- Observe que seguimos o mesmo percurso em todos os casos, apenas muda o tempo em que o nó é processado.
- No caso pos-ordem, somente processamos um nó a terceira vez que visitamos ele. Observe a mudança na posição do quadrado preto.

Percursos em Largura Primeiro
(Breadth-First Transversals)

- No **Percurso em Largura** de uma árvore binária, processamos primeiro todos os filhos de um nó (todos os nós de um nível), antes de processar os nós do seguinte nível (níveis abaixo do atual). Assim, dado um nó no nível n, devemos processar todos os nós no nível n antes de processar o nós no nível n+1.
- Para percorrer uma árvore em profundidade utilizamos recursividade, lembre que recursividade **utiliza uma pilha** para lembrar das tarefas pendentes, na ordem inversa. Já no percurso de uma árvore em largura **utiliza-se uma fila** para manter as tarefas pendentes na ordem em que apareceram.

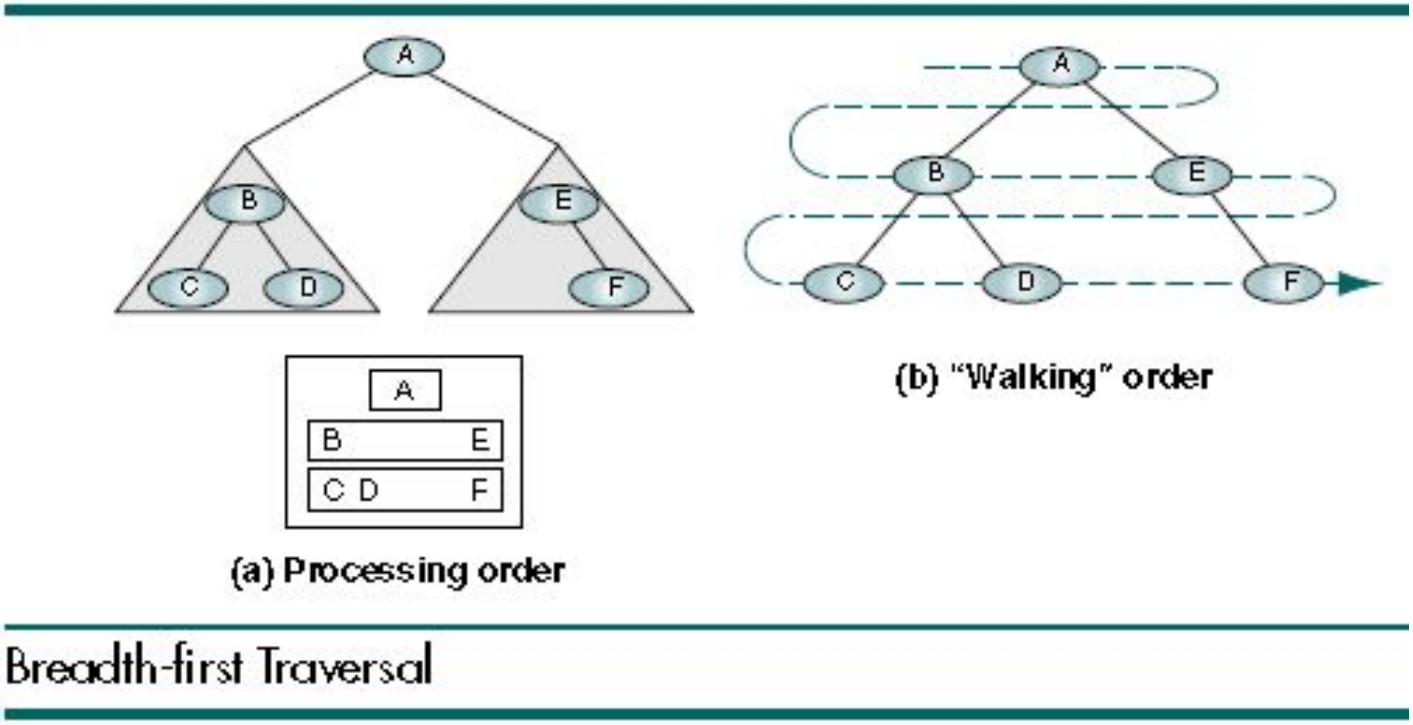
Percursos em Largura Primeiro
(Breadth-First Transversals)

- O algoritmo para o percurso em largura de uma árvore binária é mostrado ao lado.

```
Algorithm breadthFirst (root)
Process tree using breadth-first traversal.
   Pre    root is node to be processed
   Post   tree has been processed
 1 set currentNode to root
 2 createQueue (bfQueue)
 3 loop (currentNode not null)
   1   process (currentNode)
   2   if (left subtree not null)
       1   enqueue (bfQueue, left subtree)
   3   end if
   4   if (right subtree not null)
       1   enqueue (bfQueue, right subtree)
   5   end if
   6   if (not emptyQueue(bfQueue))
       1   set currentNode to dequeue (bfQueue)
   7   else
       1   set currentNode to null
   8   end if
 4 end loop
 5 destroyQueue (bfQueue)
end breadthFirst
```

Percursos em Largura Primeiro
(Breadth-First Transversals)

- A figura ilustra o caminho realizado pelo percurso em largura.
- O caminho faz uma varredura horizontal da árvore. Primeiro no primeiro nível, depois no nível 1, depois no nível 2 e assim sucessivamente até que a árvore inteira é percorrida.



Árvores de Expressões

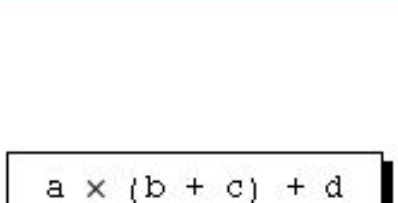
Descrição

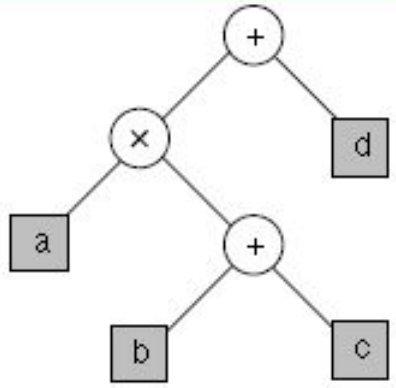
- Uma aplicação interessante de árvores binárias são as árvores utilizadas para representar expressões.
- Uma expressão é uma sequência de *tokens* que seguem uma serie de regras. Um *token* pode ser um operador ou um operando. No exemplo, considera-se somente operadores aritméticos binários, no formato (operando operador operando)
- Os operadores aritméticos padrão são: +, - , * , e /.

Árvores de Expressões
Descrição

- A árvore que representar uma expressão tem as seguintes propriedades:
  - Cada folha é um operando;
  - A raiz e os nós internos são operadores;
  - As sub-árvores representam sub-expressões, onde a raiz é um operador.

- A figura ilustra expressão infixa e a sua árvore de expressão correspondente.





Infix Expression and Its Expression Tree

Árvores de Expressões
Descrição

- Para um árvore de expressão, os três tipos de percurso em profundidade padrão, representam os três diferentes formatos de expressão:
  - infixa,
  - posfixa e
  - prefixa.
- O percurso em-ordem produz uma expressão in-fixa.
- O percurso pos-ordem produz uma expressão pós-fixa.
- O percurso pre-ordem produz uma expressão pré-fixa.

Árvores de Expressões
Percurso Infixo

- Mostramos o percurso infixo de uma árvore de expressão. O algoritmo percorre a árvore e imprime a expressão infixa.
- Neste caso é necessário imprimir parênteses de abertura e fechamento, no inicio e fim de cada expressão.
- Como a raiz de cada árvore e sub-árvore, representa uma expressão ou sub-expressão imprime-se um parêntese de abertura sempre que identificamos o processamento de uma raiz (nó interno) antes das chamadas recursivas esquerda e direita, e um parêntese de fechamento após essas chamadas.

Árvores de Expressões
Percurso Infixo

- O algoritmo para o percurso infixo é o seguinte:

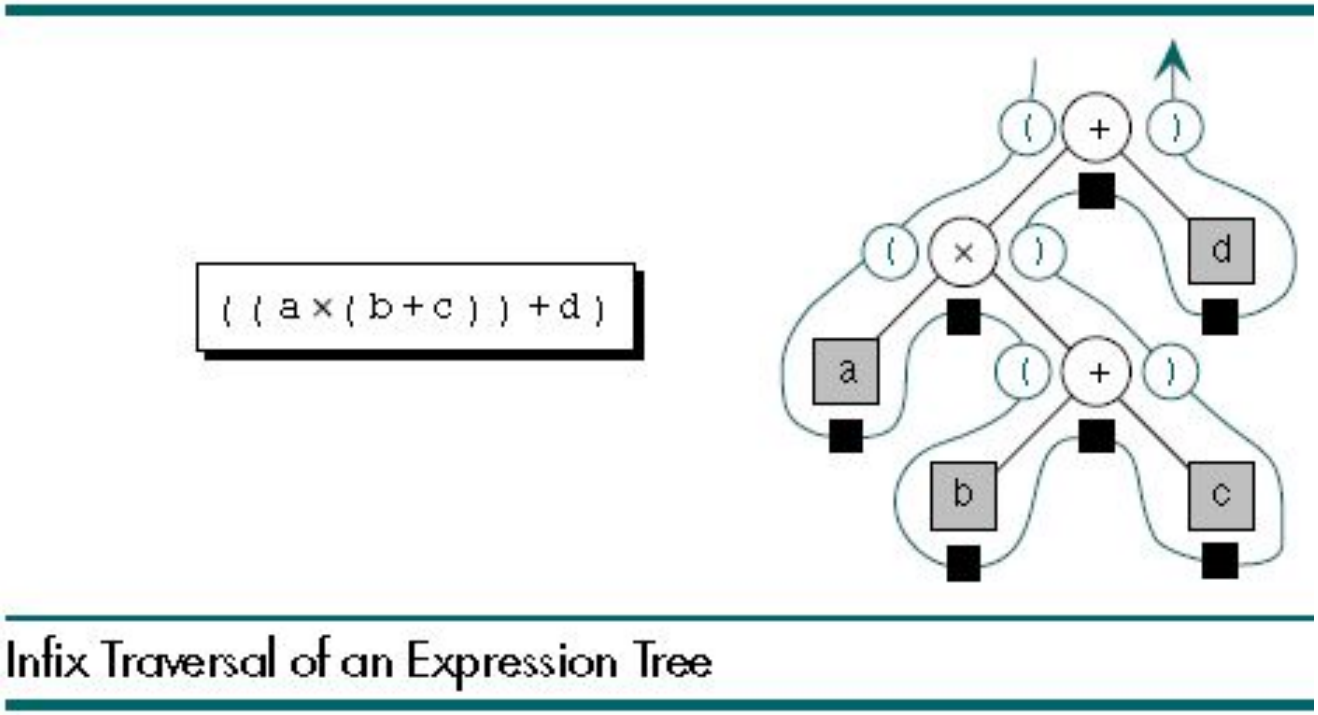
Infix Expression Tree Traversal

```
Algorithm infix (tree)
Print the infix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the infix expression has been printed
1  if (tree not empty)
   1   if (tree token is an operand)
       1  print (tree-token)
   2   else
       1  print (open parenthesis)
       2  infix (tree left subtree)
       3  print (tree token)
       4  infix (tree right subtree)
       5  print (close parenthesis)
   3   end if
2  end if
end infix
```

- Observe a colocação dos parênteses de abertura e fechamento antes da primeiro chamada recursiva e depois da segunda chamada recursiva respectivamente.

Árvores de Expressões
Percurso Infixo

- A figura ilustra o caminho seguido pelo percurso em-ordem da árvore de expressão indicando a colocação do parênteses de abertura e fechamento.



Infix Traversal of an Expression Tree

Árvores de Expressões
Percurso Pósfixo

- A expressão obtida pelo percurso pósfixo da árvore é uma expressão pósfixa. Utiliza-se o percurso pós-ordem da árvore. Neste caso não é necessária a impressão de parênteses.
- O algoritmo correspondente é o seguinte:

Postfix Traversal of an Expression Tree

```
Algorithm postfix (tree)
Print the postfix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the postfix expression has been printed
 1 if (tree not empty)
   1   postfix (tree left subtree)
   2   postfix (tree right subtree)
   3   print   (tree token)
 2 end if
end postfix
```

Árvores de Expressões
Percurso Prefixo

- A expressão obtida pelo percurso prefixo da árvore é uma expressão prefixa. Utiliza-se o percurso pré-ordem da árvore. Neste caso não é necessária a impressão de parênteses.
- O algoritmo correspondente é o seguinte:

Prefix Traversal of an Expression Tree

```
Algorithm prefix (tree)
Print the prefix expression for an expression tree.
   Pre  tree is a pointer to an expression tree
   Post the prefix expression has been printed
 1 if (tree not empty)
   1  print  (tree token)
   2  prefix (tree left subtree)
   3  prefix (tree right subtree)
 2 end if
end prefix
```

Referências

- Gilberg, R.F. e Forouzan, B.A. Data Structures_A Pseudocode Approach with C. Capítulo 6. Introdution to Trees. Segunda Edição. Editora Cengage, Thomson Learning, 2005.