Introduction to Trees

Objectives

Upon completion you will be able to:

- **Understand and use basic tree terminology and concepts**
- **Recognize and define the basic attributes of a binary tree**
- **Process trees using depth-first and breadth-first traversals**
- **Parse expressions using a binary tree**
- **Design and implement Huffman trees**
- **Understand the basic use and processing of general trees**

Basic Tree Concepts

- A tree consists of finite set of elements, called nodes, and a finite set of directed lines called branches, that connect the nodes.
- The number of branches associated with a node is the degree of the node.

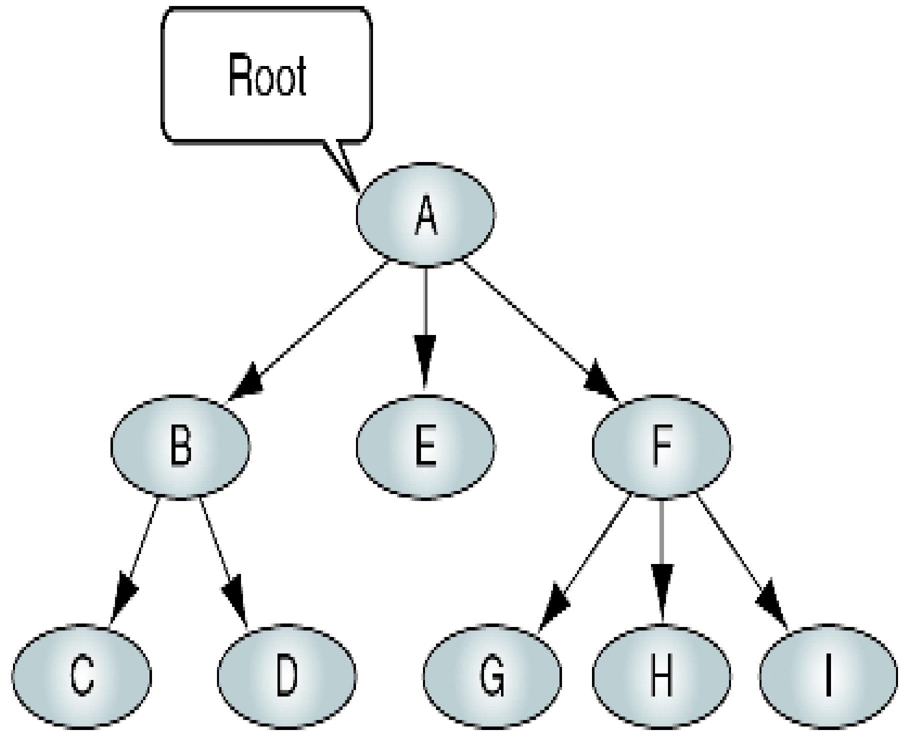


FIGURE 6-1 Tree

Basic Tree Concepts

- When the branch is directed toward the node, it is indegree branch.
- When the branch is directed away from the node, it is an outdegree branch.
- The sum of the indegree and outdegree branches is the degree of the node.
- If the tree is not empty, the first node is called the root.

Basic Tree Concepts

- The indegree of the root is, by definition, zero.
- With the exception of the root, all of the nodes in a tree must have an indegree of exactly one; that is, they may have only one predecessor.
- All nodes in the tree can have zero, one, or more branches leaving them; that is, they may have outdegree of zero, one, or more.

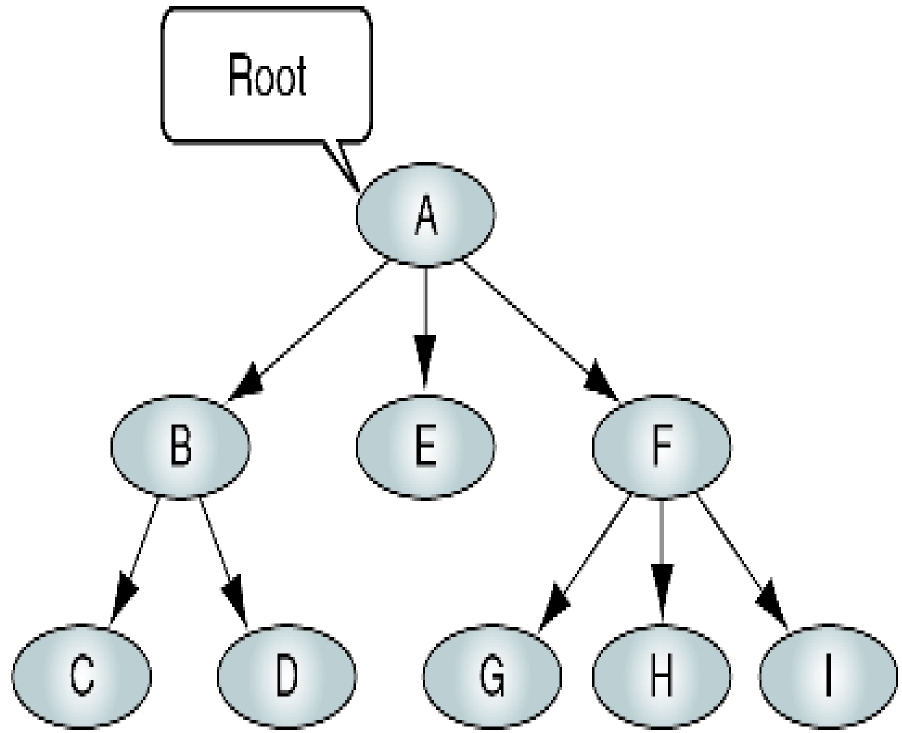


FIGURE 6-1 Tree

Basic Tree Concepts

- A leaf is any node with an outdegree of zero, that is, a node with no successors.
- A node that is not a root or a leaf is known as an internal node.
- A node is a parent if it has successor nodes; that is, if it has outdegree greater than zero.
- A node with a predecessor is called a child.

Basic Tree Concepts

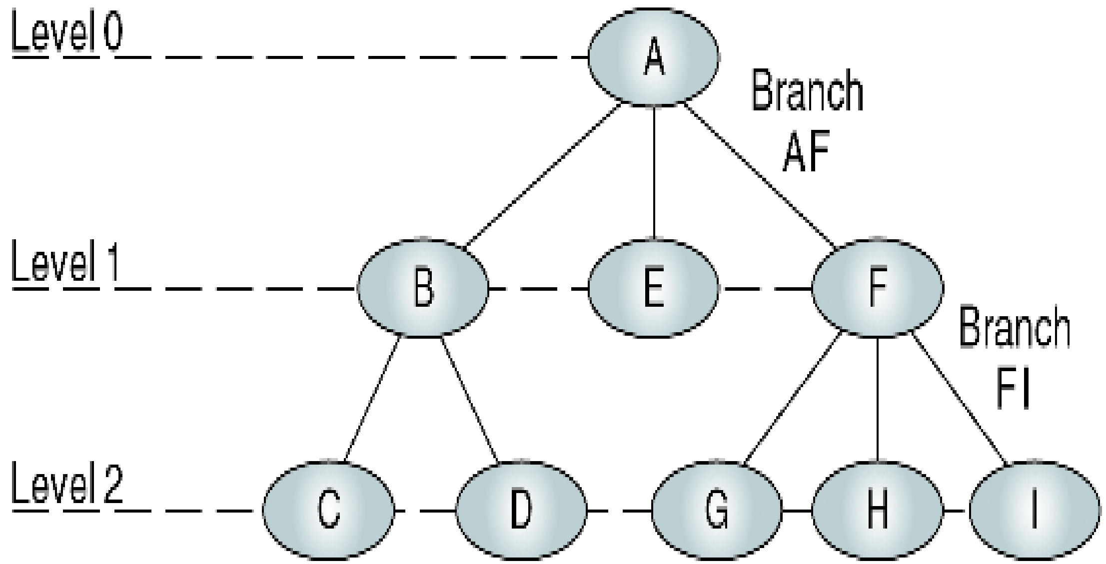
- Two or more nodes with the same parents are called siblings.
- An ancestor is any node in the path from the root to the node.
- A descendant is any node in the path below the parent node; that is, all nodes in the paths from a given node to a leaf are descendants of that node.

Basic Tree Concepts

- A path is a sequence of nodes in which each node is adjacent to the next node.
- The level of a node is its distance from the root. The root is at level 0, its children are at level 1, etc. …

Basic Tree Concepts

- The height of the tree is the level of the leaf in the longest path from the root plus 1. By definition the height of any empty tree is -1.
- A subtree is any connected structure below the root. The first node in the subtree is known is the root of the subtree.



Root: A	Siblings: {B,E,F}, {C,D}, {G,H,I}
Parents: A, B, F	Leaves: C,D,E,G,H,I
Children: B, E, F, C, D, G, H, I	Internal nodes: B,F

FIGURE 6-2 Tree Nomenclature

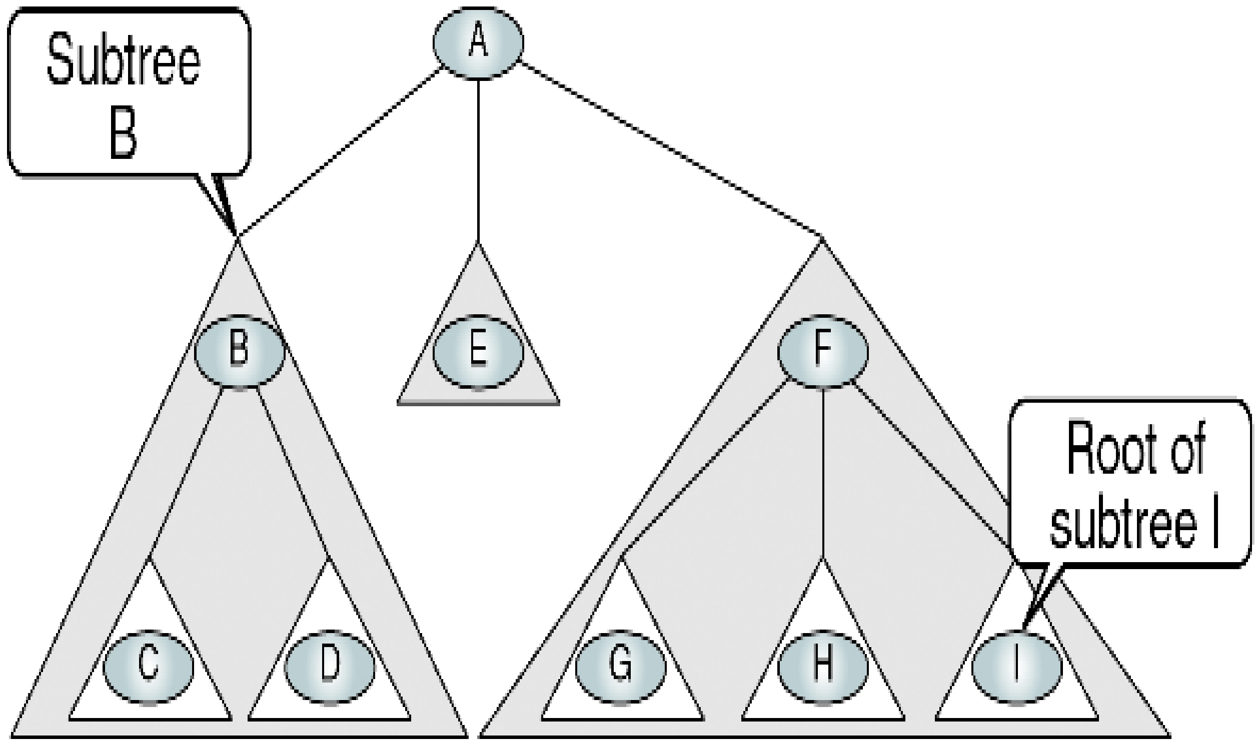


FIGURE 6-3 Subtrees

Recursive definition of a tree

- A tree is a set of nodes that either:
- is empty or
- has a designated node, called the root, from which hierarchically descend zero or more subtrees, which are also trees.

Tree Representation

- General Tree – organization chart format
- Indented list – bill-of-materials system in which a parts list represents the assembly structure of an item

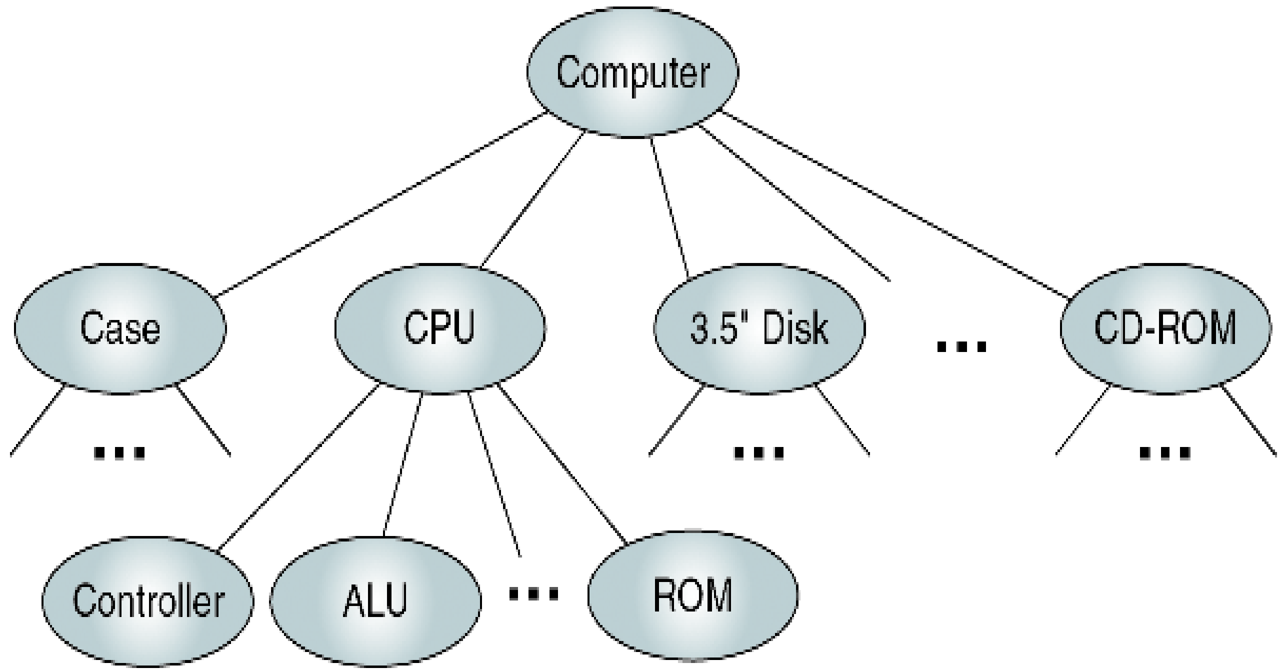


FIGURE 6-4 Computer Parts List as a General Tree

Part number	Description
301	Computer
301-1	Case
…	…
301-2	CPU
301-2-1	Controller
301-2-2	ALU
…	…
301-2-9	ROM
301-3	3.5" Disk
…	…
301-9	CD-ROM
…	…

TABLE 6-1 Computer Bill of Materials

Parenthetical Listing

- Parenthetical Listing – the algebraic expression, where each open parenthesis indicates the start of a new level and each closing parenthesis completes the current level and moves up one level in the tree.

Parenthetical Listing

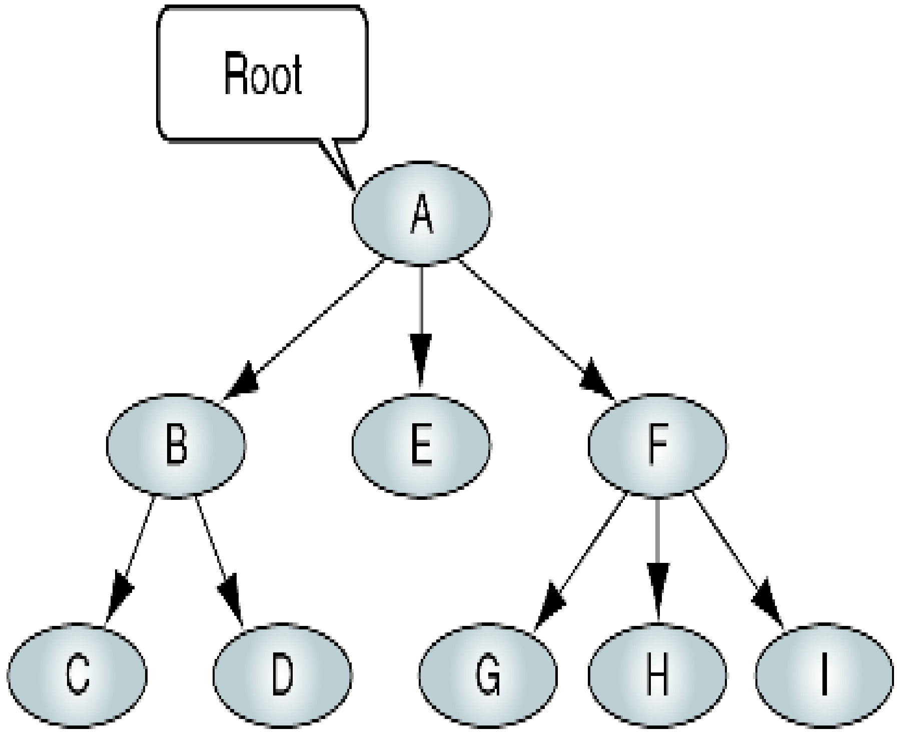


FIGURE 6-1 Tree

A (B (C D) E F (G H I))

ALGORITHM 6-1 Convert General Tree to Parenthetical Notation

```
Algorithm ConvertToParen (root, output)
Convert a general tree to parenthetical notation.
   Pre  root is a pointer to a tree node
   Post output contains parenthetical notation
1 Place root in output
2 if (root is a parent)
   1  Place an open parenthesis in the output
   2  ConvertToParen (root's first child)
   3  loop (more siblings)
      1  ConvertToParen (root's next child)
```

continued

ALGORITHM 6-1 Convert General Tree to Parenthetical Notation *(continued)*

```
   4  end loop
   5  Place close parenthesis in the output
3 end if
4 return
end ConvertToParen
```

6-2 Binary Trees

A binary tree can have no more than two descendents. In this section we discuss the properties of binary trees, four different binary tree traversals

- **Properties**
- **Binary Tree Traversals**
- **Expression Trees**
- **Huffman Code**

Binary Trees

- A **binary tree** is a tree in which no node can have more than two subtrees; the maximum outdegree for a node is two.
- In other words, a node can have zero, one, or two subtrees.
- These subtrees are designated as the left subtree and the right subtree.

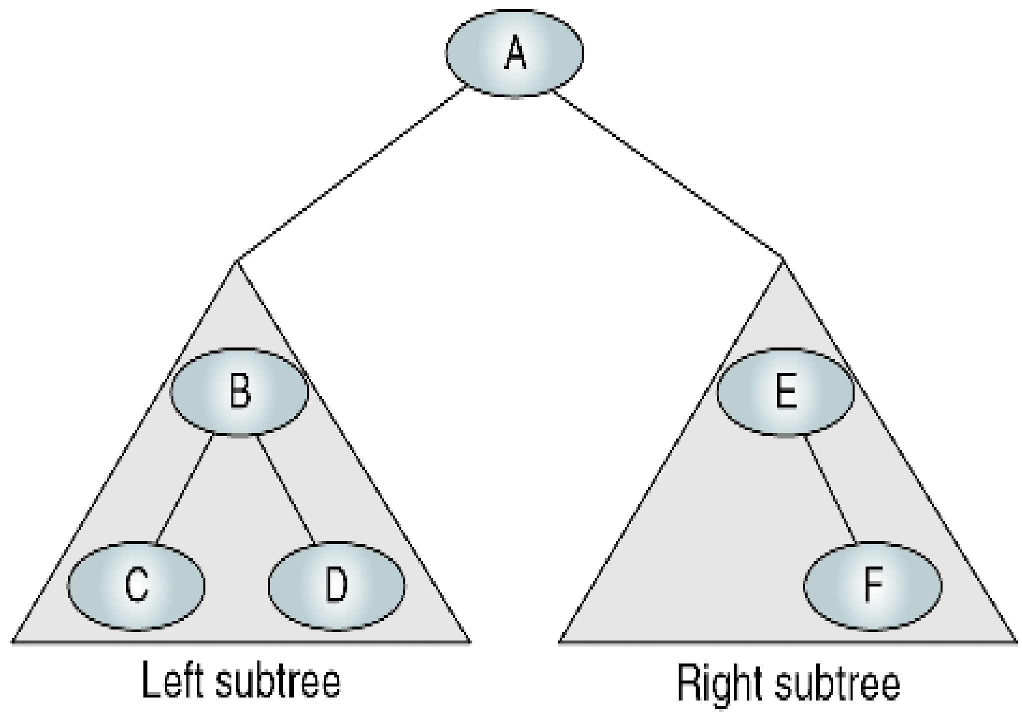
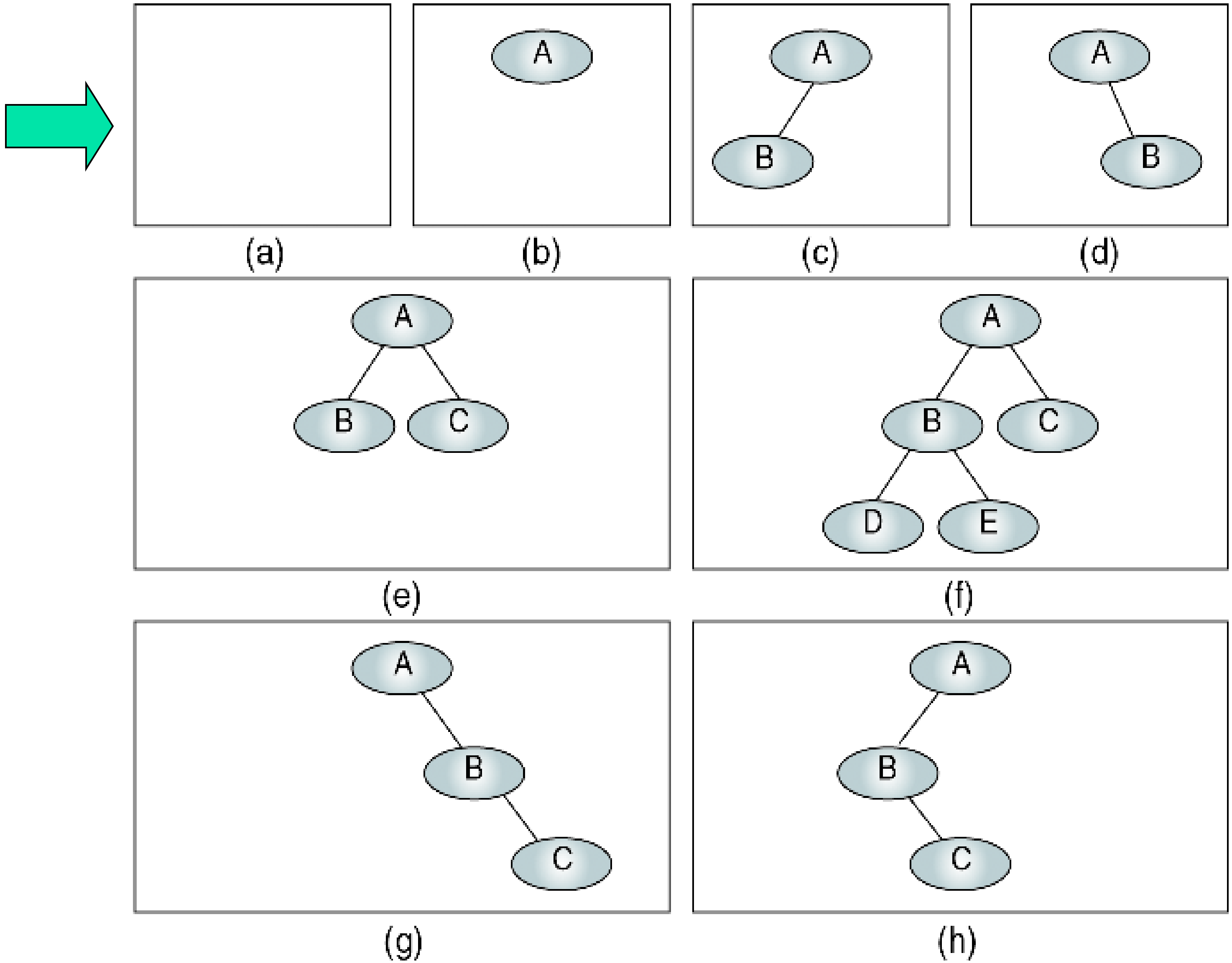


FIGURE 6-5 Binary Tree

A null tree is a tree with no nodes



(a) (b) (c) (d) (e) (f) (g) (h)

FIGURE 6-6 Collection of Binary Trees

Some Properties of Binary Trees

- The height of binary trees can be mathematically predicted
- Given that we need to store N nodes in a binary tree, the maximum height is

$$H_{\max} = N$$

A tree with a maximum height is rare. It occurs when all of the nodes in the entire tree have only one successor.

Some Properties of Binary Trees

- The minimum height of a binary tree is determined as follows:

$$H_{\min} = \left[\log_2 N\right] + 1$$

For instance, if there are three nodes to be stored in the binary tree (N=3) then $H_{\min}$=2.

Some Properties of Binary Trees

- Given a height of the binary tree, H, the minimum number of nodes in the tree is given as follows:

$$N_{\min} = H$$

Some Properties of Binary Trees

- The formula for the maximum number of nodes is derived from the fact that each node can have only two descendents. Given a height of the binary tree, $\boldsymbol{H}$, the maximum number of nodes in the tree is given as follows:

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

Some Properties of Binary Trees

- The children of any node in a tree can be accessed by following only one branch path, the one that leads to the desired node.
- The nodes at level 1, which are children of the root, can be accessed by following only one branch; the nodes of level 2 of a tree can be accessed by following only two branches from the root, etc.
- The balance factor of a binary tree is the difference in height between its left and right subtrees:

$$B = H_L - H_R$$

Balance of the tree

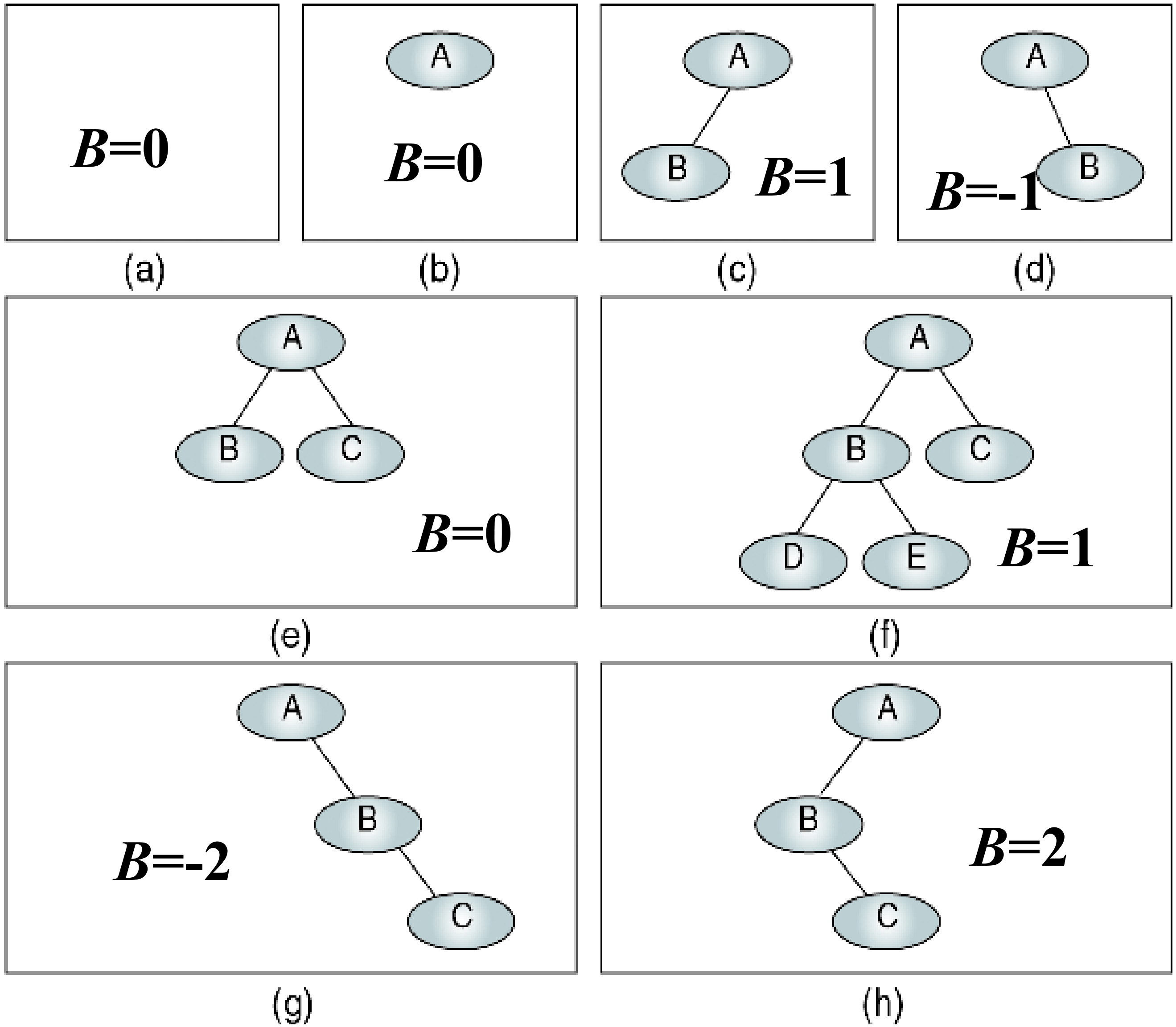


FIGURE 6-6 Collection of Binary Trees

Some Properties of Binary Trees

- In the balanced binary tree (definition of Russian mathematicians Adelson-Velskii and Landis) the height of its subtrees differs by no more than one (its balance factor is -1, 0, or 1), and its subtrees are also balanced.

Complete and nearly complete binary trees

- A complete tree has the maximum number of entries for its height. The maximum number is reached when the last level is full.
- A tree is considered nearly complete if it has the minimum height for its nodes and all nodes in the last level are found on the left

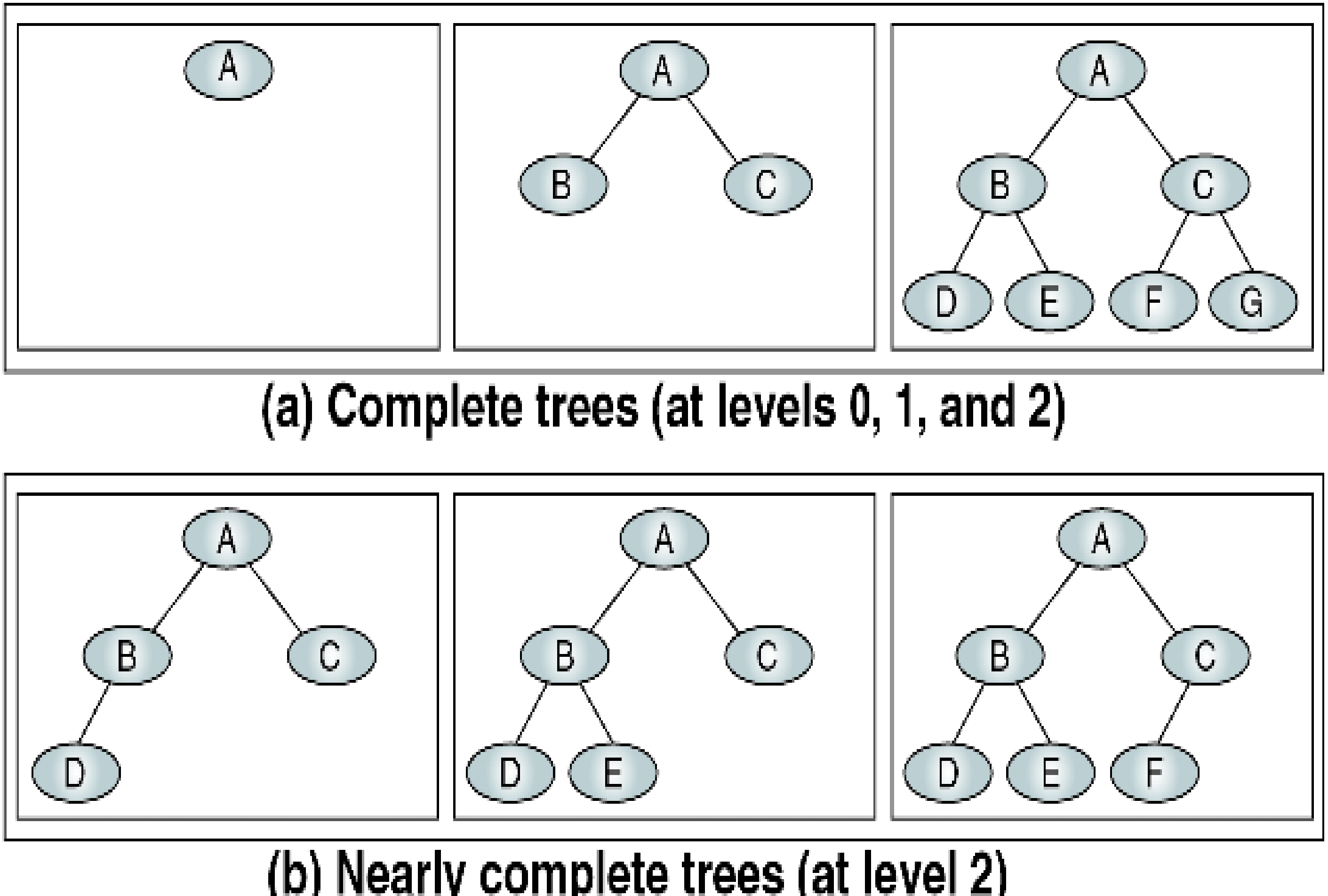


FIGURE 6-7 Complete and Nearly Complete Trees

Binary Tree Traversal

- A binary tree traversal requires that each node of the tree be processed once and only once in a predetermined sequence.
- In the depth-first traversal processing process along a path from the root through one child to the most distant descendant of that first child before processing a second child.

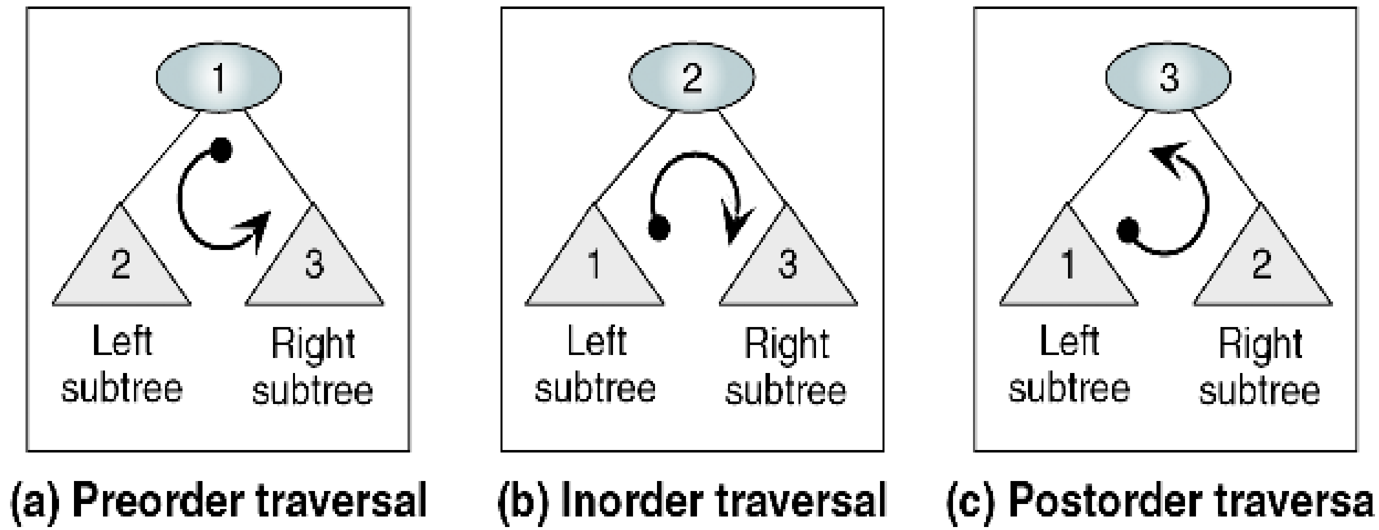


FIGURE 6-8 Binary Tree Traversals

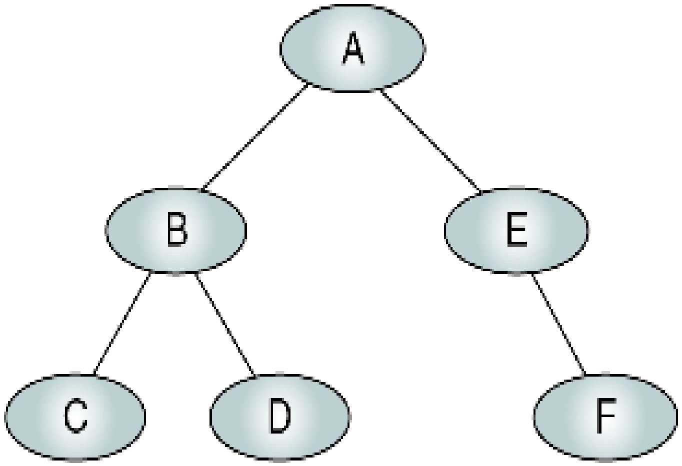
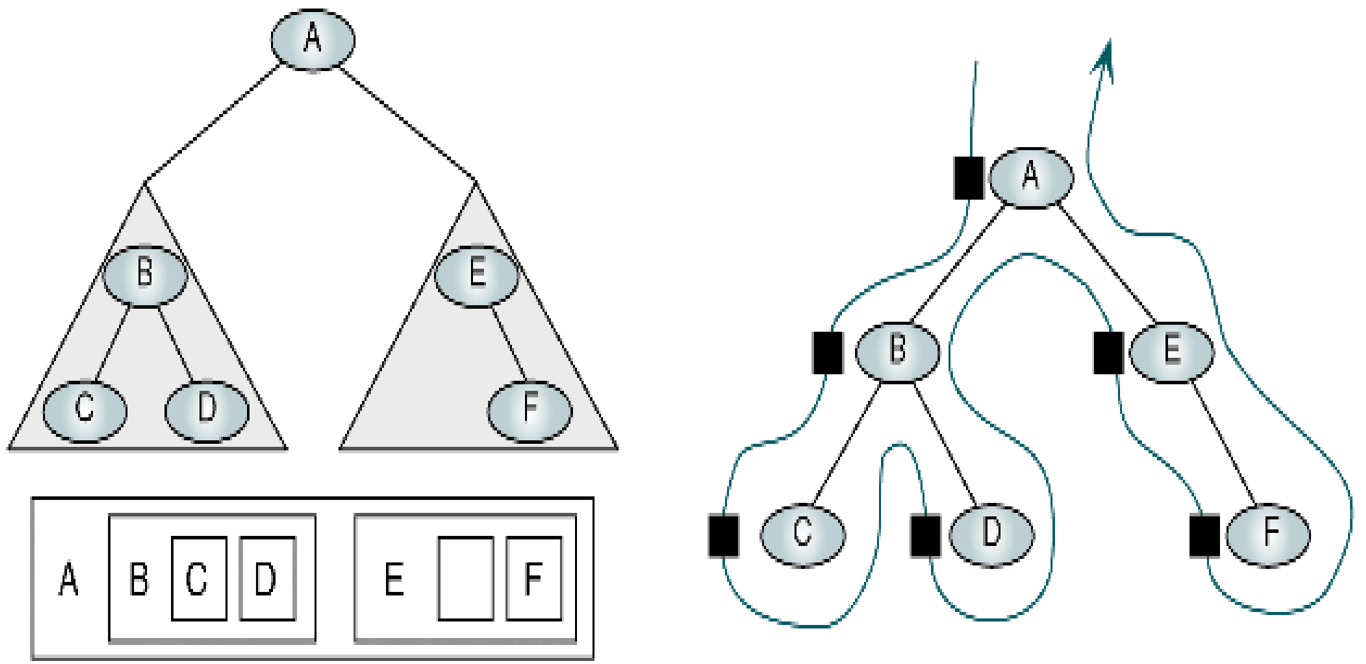


FIGURE 6-9 Binary Tree for Traversals

ALGORITHM 6-2 Preorder Traversal of a Binary Tree

```
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
```



(a) Processing order

(b) "Walking" order

FIGURE 6-10 Preorder Traversal—A B C D E F

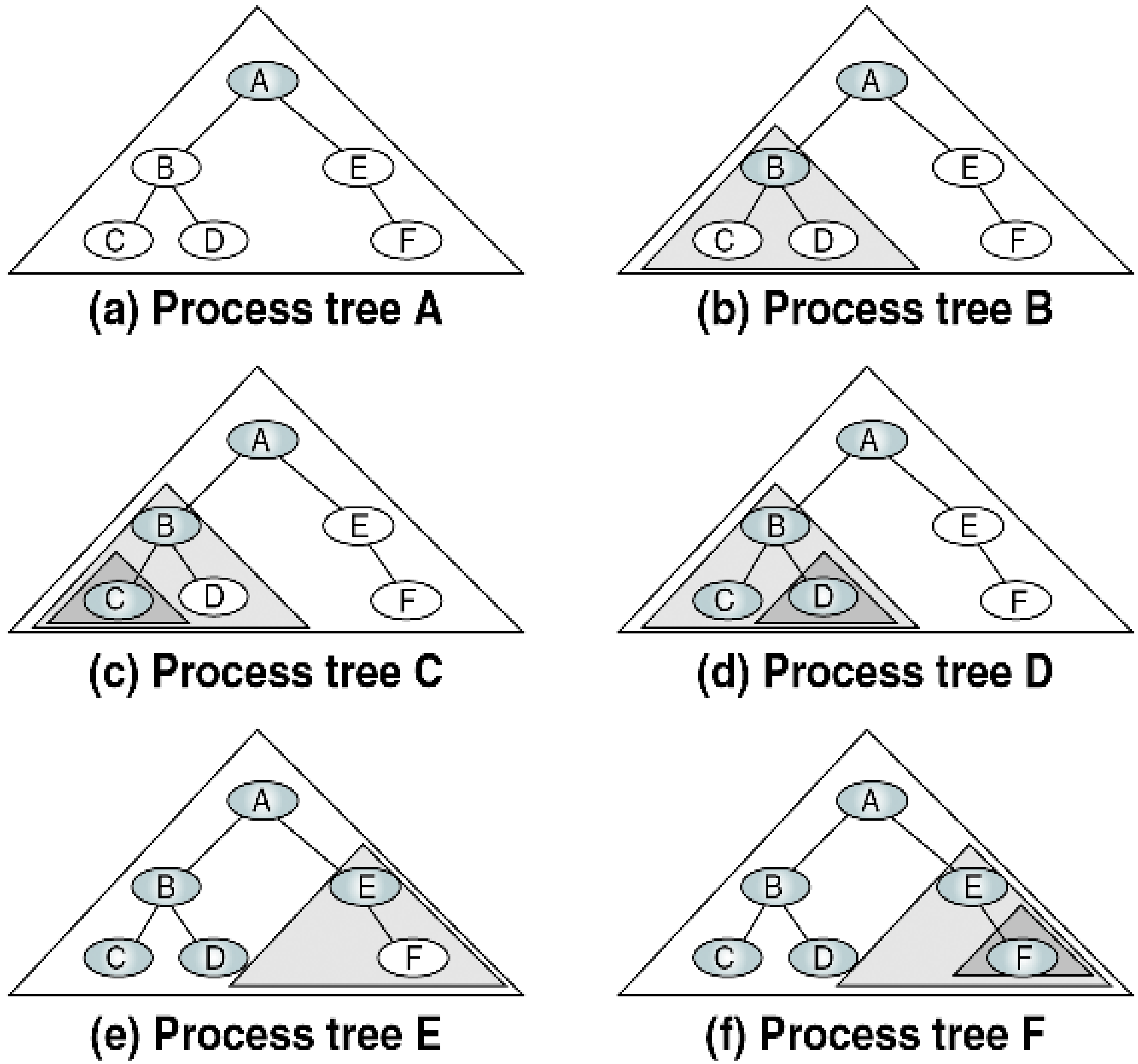


FIGURE 6-11 Algorithmic Traversal of Binary Tree

ALGORITHM 6-3 Inorder Traversal of a Binary Tree

```
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
```

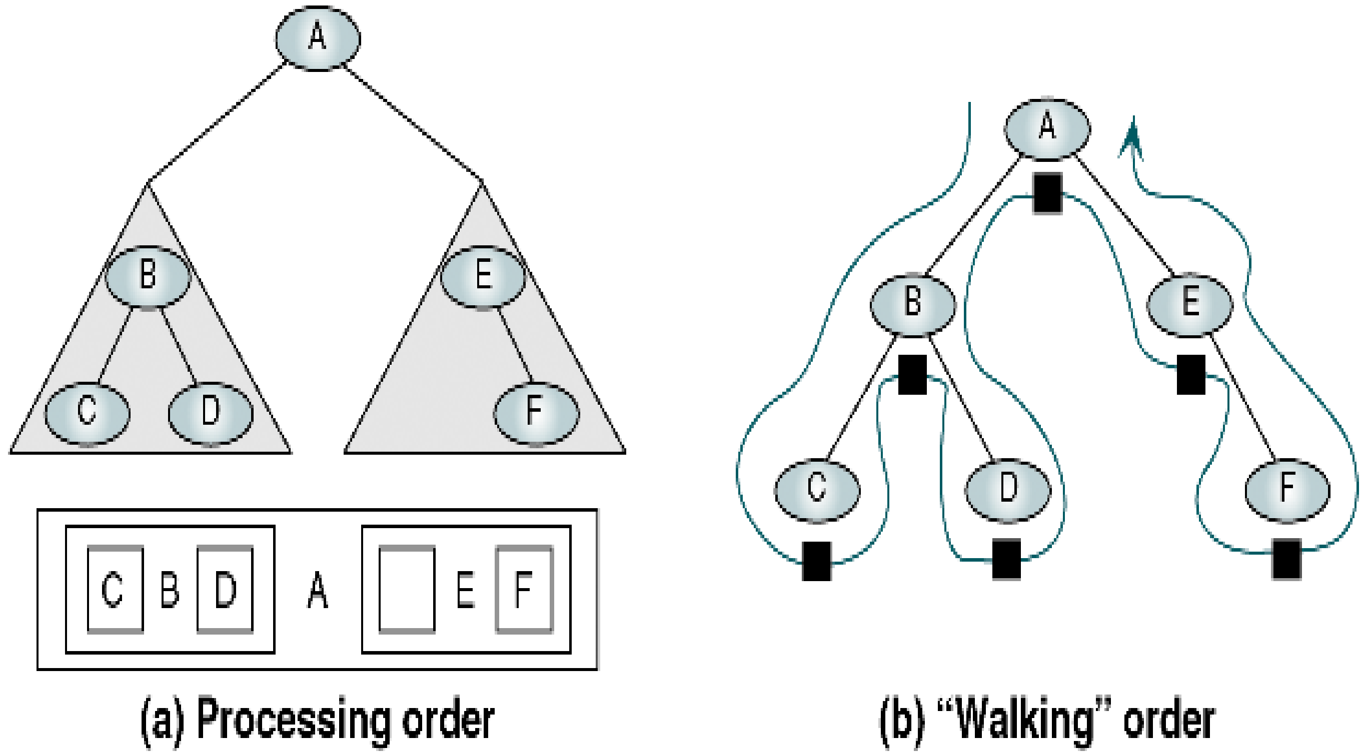


FIGURE 6-12 Inorder Traversal—C B D A E F

ALGORITHM 6-4 Postorder Traversal of a Binary Tree

```
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
```

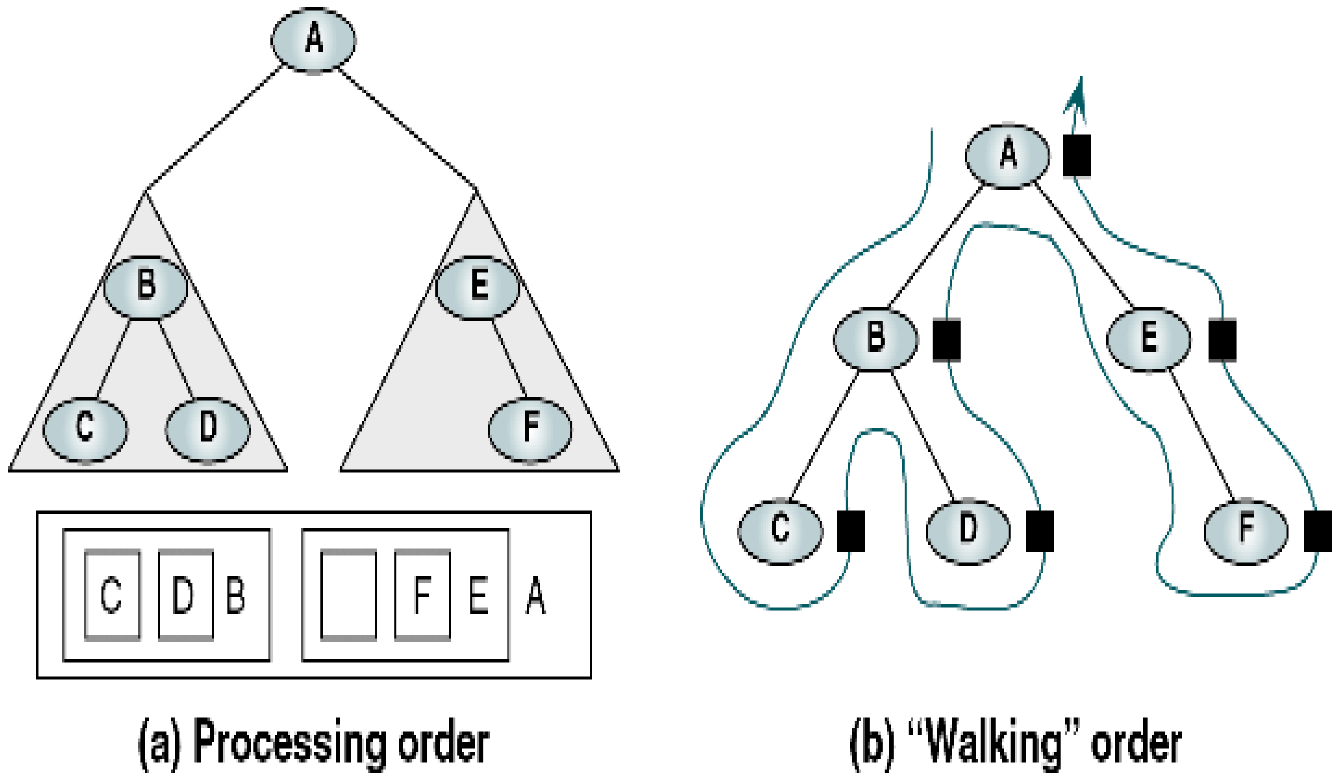


FIGURE 6-13 Postorder Traversal—C D B F E A

ALGORITHM 6-5 Breadth-first Tree Traversal

```
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
```

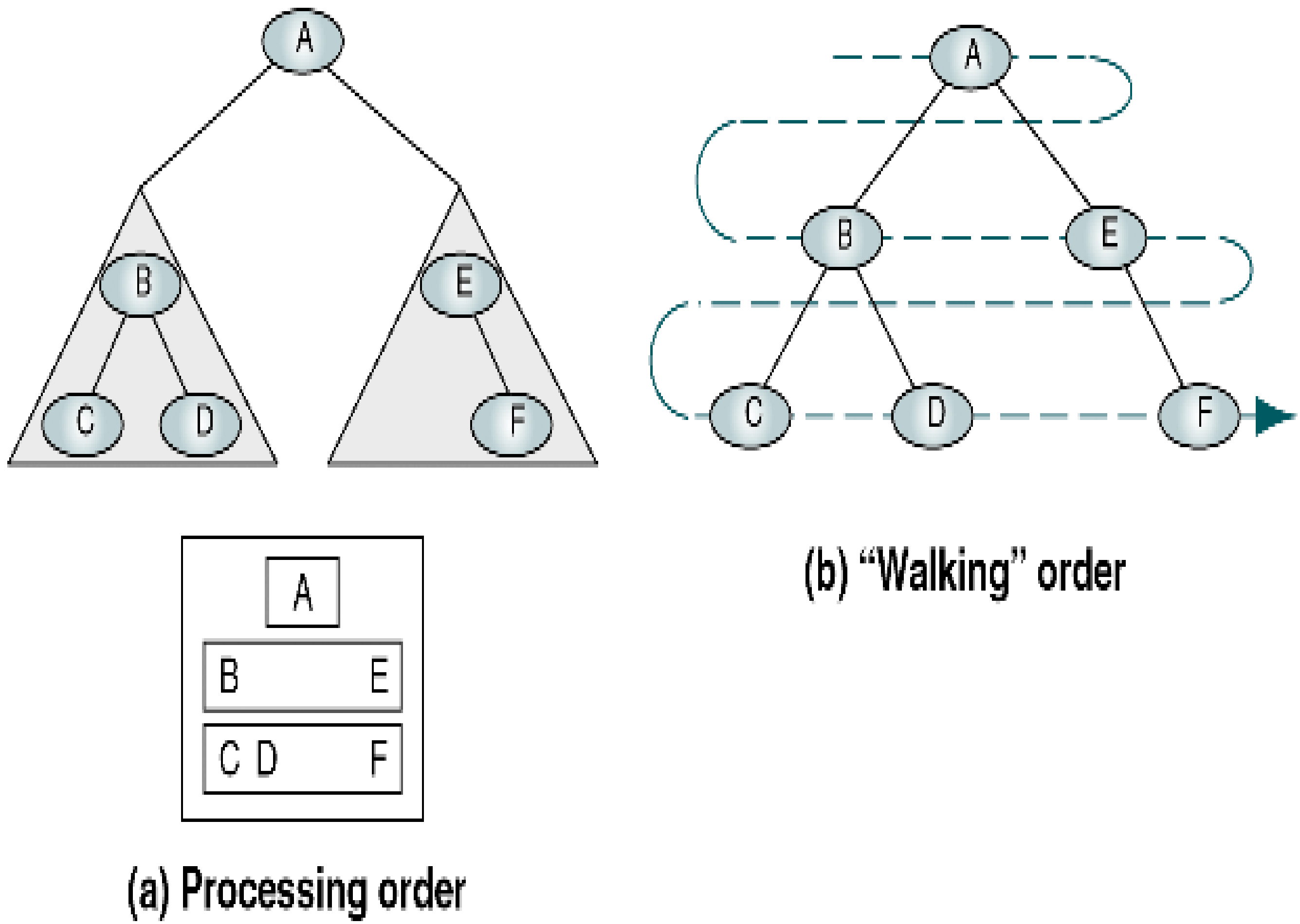


FIGURE 6-14 Breadth-first Traversal

$a \times (b + c) + d$

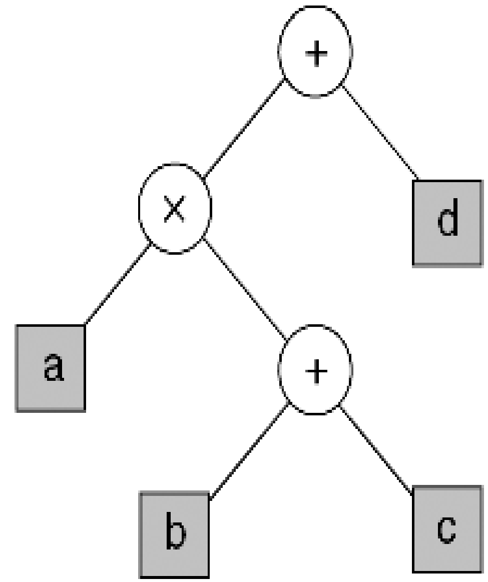


FIGURE 6-15 Infix Expression and Its Expression Tree

$((a \times (b + c)) + d)$

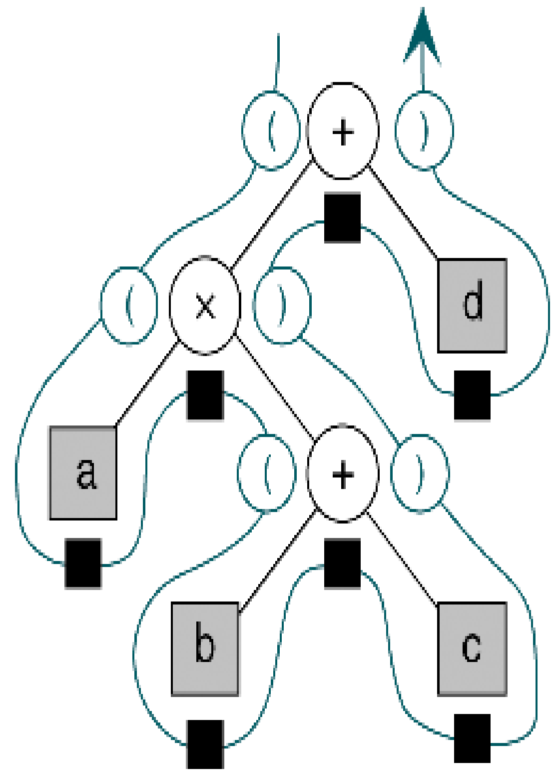


FIGURE 6-16 Infix Traversal of an Expression Tree

ALGORITHM 6-6 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
```

ALGORITHM 6-7 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)
```

continued

ALGORITHM 6-7 Postfix Traversal of an Expression Tree *(continued)*

```
    1  postfix (tree left subtree)
    2  postfix (tree right subtree)
    3  print   (tree token)
 2  end if
end postfix
```

ALGORITHM 6-8 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
```