Ceng 124
Discrete Structures

2018-2019 Spring Semester

by Sibel T. Özyer, Spring 2019

Topics

- 11.3 Tree Traversal
  - Introduction
    - Universal Address Systems
  - Traversal Algorithms
    - Preorder
    - Indorder
    - Postorder
  - Infix, Prefix, Postfix Notations

11.3 Introduction

- Ordered rooted trees are often used to store information.
- We need procedures for visiting each vertex of an ordered rooted tree to access data.
- We will describe several important algorithms for visiting all the vertices of an ordered rooted tree.
- Ordered rooted trees can also be used to represent various types of expressions, such as arithmetic expressions involving numbers, variables, and operations.

Universal Address Systems

- Procedures for traversing all vertices of an ordered rooted tree rely on the orderings of children.
- In ordered rooted trees, the children of an internal vertex are shown from left to right in the drawings representing these directed graphs.
- We will describe one way we can totally order the vertices of an ordered rooted tree.
- To produce this ordering, we must first label all the vertices. We do this recursively:
- 1. Label the root with the integer 0. Then label its k children (at level 1) from left to right with 1, 2, 3, . . . , k.
- 2. For each vertex v at level n with label A, label its kv children, as they are drawn from left to right, with $A.1, A.2, \ldots, A.kv$.

Universal Address Systems (cont.)

- Following this procedure, a vertex v at level n, for $n \geq 1$, is labeled $x1.x2. \ldots .xn$, where the unique path from the root to v goes through the $x1$st vertex at level 1, the $x2$nd vertex at level 2, and so on.
- This labeling is called the **universal address system** of the ordered rooted tree.
- We can totally order the vertices using the lexicographic ordering of their labels in the universal address system.
- The vertex labeled $x1.x2. \ldots .xn$ is less than the vertex labeled $y1.y2. \ldots .ym$ if there is an i, $0 \leq i \leq n$, with $x1 = y1$, $x2 = y2$, . . . , $xi-1 = yi-1$, and $xi < yi$; or if $n < m$ and $xi = yi$ for $i = 1, 2, \ldots, n$.

Example 1

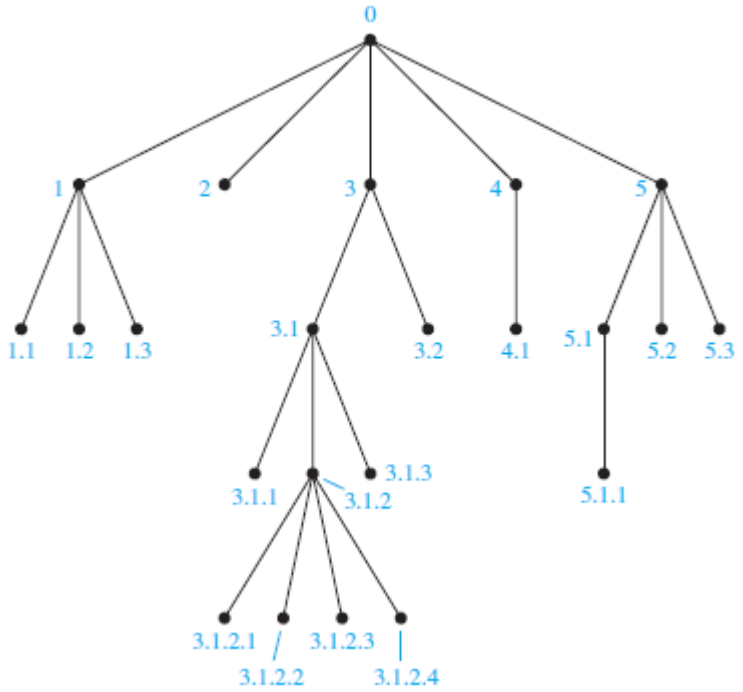

Figure 1: **The Universal Address System of an Ordered Rooted Tree**

Lexicographic Ordering of the Labeling

- We display the labelings of the universal address system next to the vertices in the ordered rooted tree shown in Figure 1. The lexicographic ordering of the labelings is
- 0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2
- < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3

Traversal Algorithms

- Procedures for systematically visiting every vertex of an ordered rooted tree are called **traversal algorithms.**
- We will describe three of the most commonly used such algorithms:
- **preorder traversal, inorder traversal,** and **postorder traversal.**
- Each of these algorithms can be defined recursively.

Definition 1

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *preorder traversal* of T. Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right in T. The *preorder traversal* begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.

Preorder Traversal

- Preorder traversal of an ordered rooted tree gives the same ordering of the vertices as the ordering obtained using a universal address system.
- Figure 2 indicates how a preorder traversal is carried out.
- Example 2 illustrates preorder traversal.

Illustration of Preorder Traversal

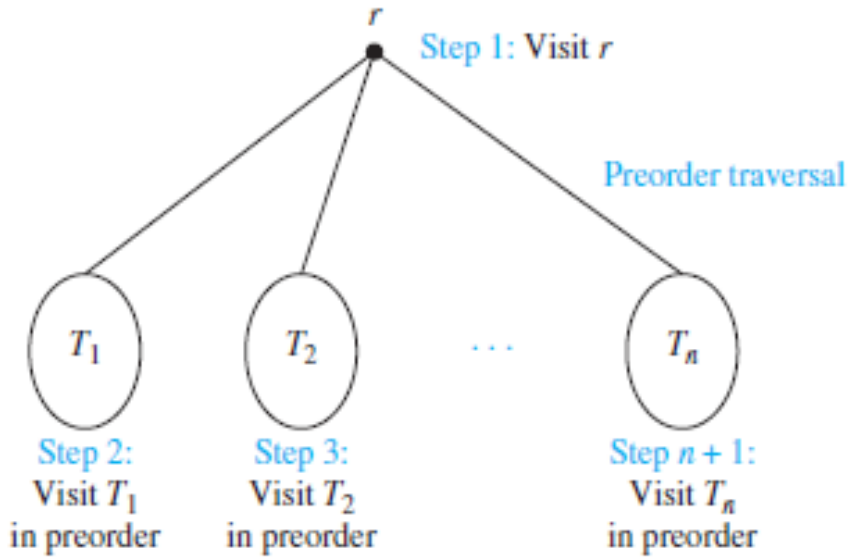

Figure 2: Preorder traversal

Example 2

- In which order does a preorder traversal visit the vertices in the ordered rooted tree *T* shown in Figure 3?

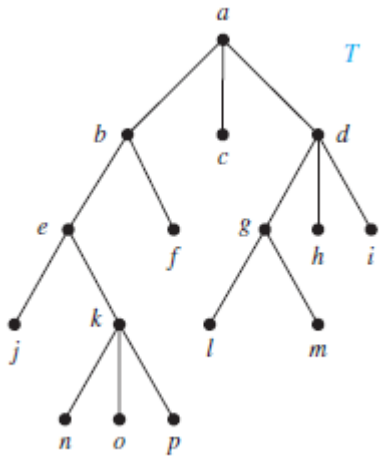

Figure 3: The Ordered Rooted Tree T

Solution

- The steps of the preorder traversal of *T* are shown in Figure 4.
- We traverse *T* in preorder by first listing the root *a*, followed by the preorder list of the subtree with root *b*, the preorder list of the subtree with root *c* (which is just *c*) and the preorder list of the subtree with root *d*.
- Consequently, the preorder traversal of *T* is
- *a*, *b*, *e*, *j* , *k*, *n*, *o*, *p*, *f* , *c*, *d*, *g*, *l*, *m*, *h*, *i*.

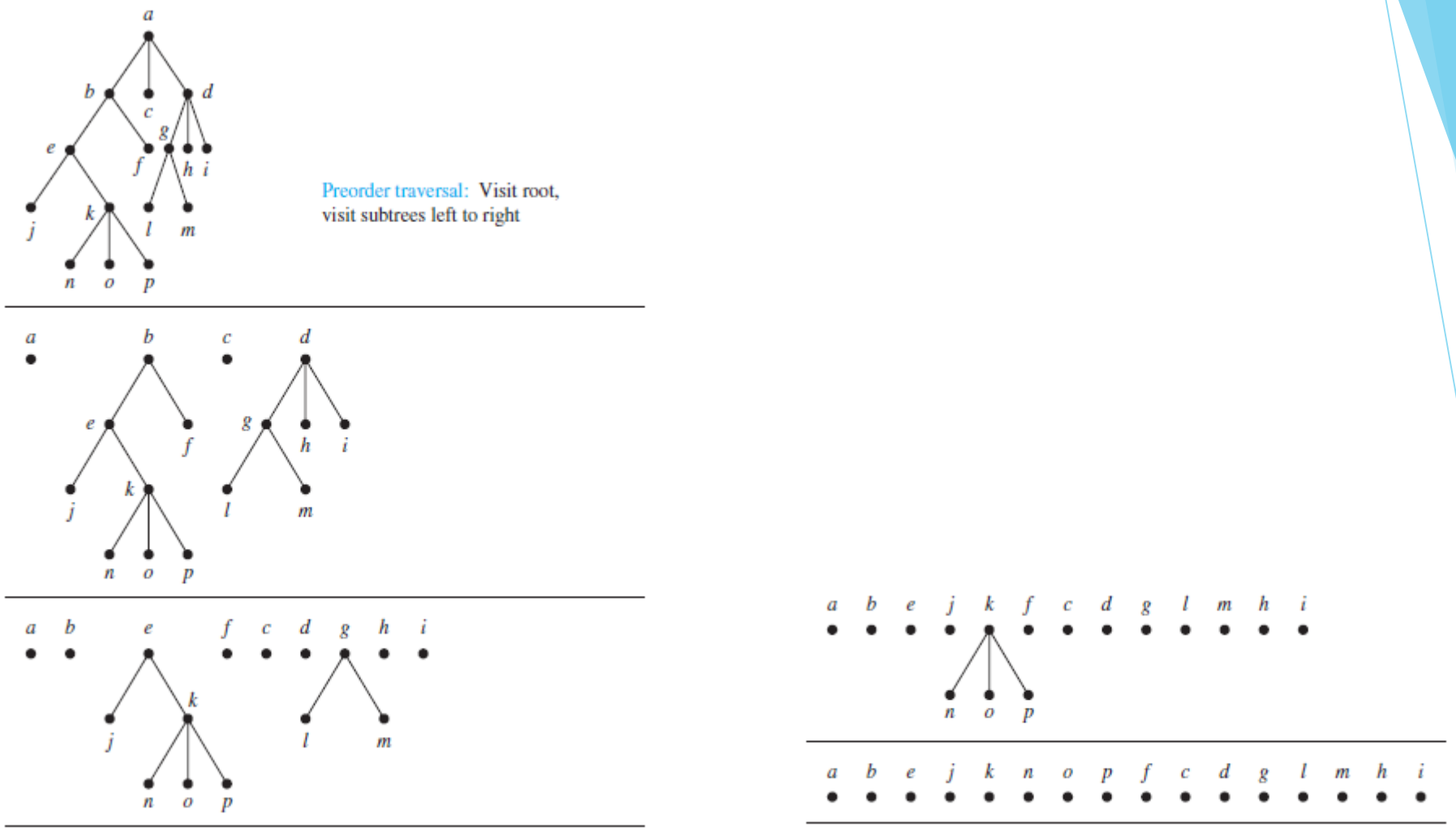

Figure 4 The Preorder Traversal of *T* .

Definition 2

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *inorder traversal* of T. Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right. The *inorder traversal* begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, then T_3 in inorder, ..., and finally T_n in inorder.

Inorder Traversal

- Figure 5 indicates how inorder traversal is carried out.
- Example 3 illustrates how inorder traversal is carried out for a particular tree.

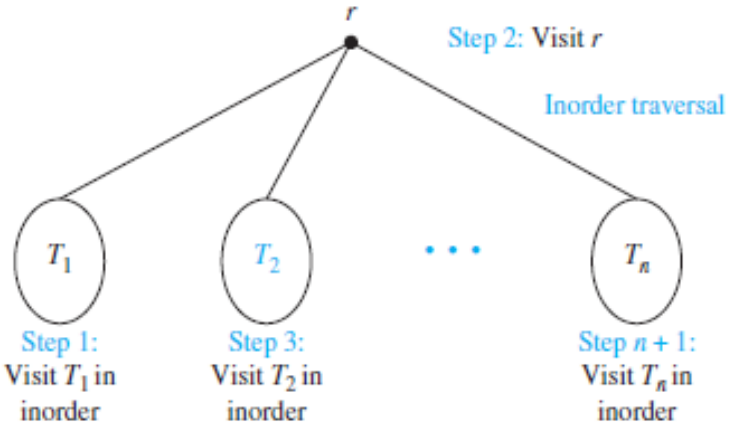

Figure 5 Inorder Traversal

Example 3

- In which order does an inorder traversal visit the vertices of the ordered rooted tree *T* in Figure 3?

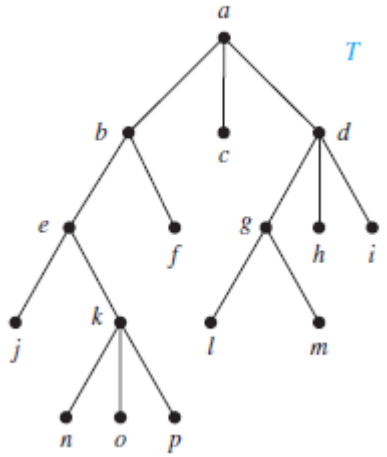

Figure 3: The Ordered Rooted Tree T

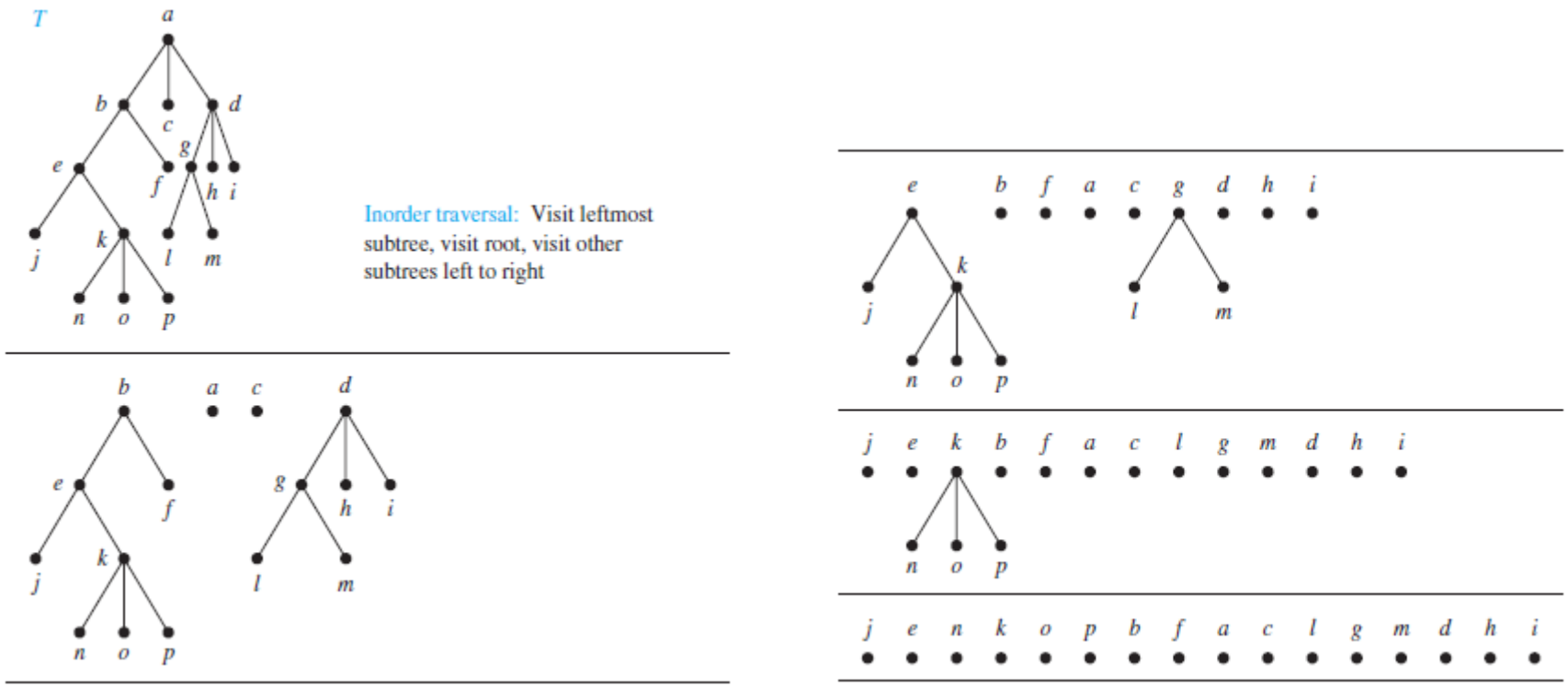

Figure 6 The Inorder Traversal of *T* .

Solution

- The steps of the inorder traversal of the ordered rooted tree *T* are shown in Figure 6.
- The inorder traversal begins with an inorder traversal of the subtree with root *b*, the root *a*, the inorder listing of the subtree with root *c*, which is just *c*, and the inorder listing of the subtree with root *d*.
- Consequently, the inorder listing of the ordered rooted tree is
- *j* , *e*, *n*, *k*, *o*, *p*, *b*, *f* , *a*, *c*, *l*, *g*, *m*, *d*, *h*, *i*.

Definition 3

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *postorder traversal* of T. Otherwise, suppose that $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right. The *postorder traversal* begins by traversing T_1 in postorder, then T_2 in postorder, ..., then T_n in postorder, and ends by visiting r.

Postorder Traversal

- Figure 7 illustrates how postorder traversal is done. Example 4 illustrates how postorder traversal works.

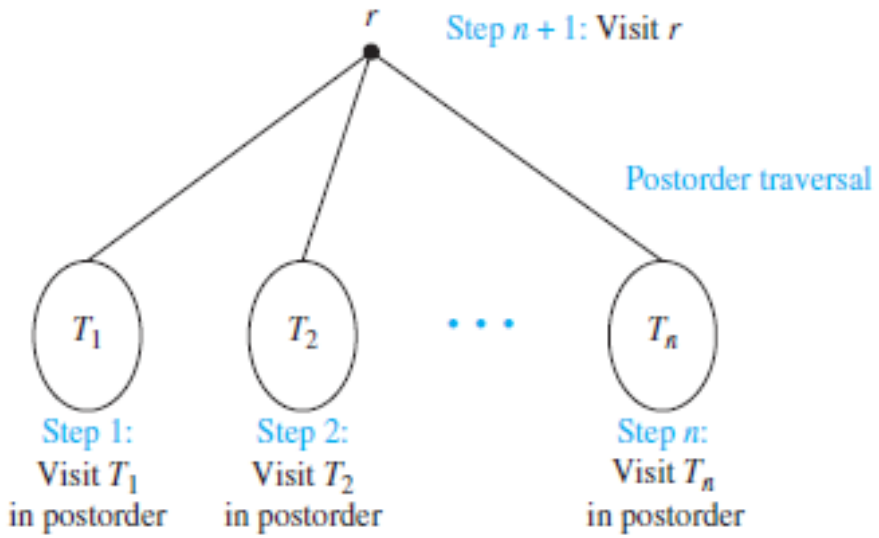

Figure 7 Postorder Traversal.

Example 4

- In which order does a postorder traversal visit the vertices of the ordered rooted tree *T* shown in Figure 3?

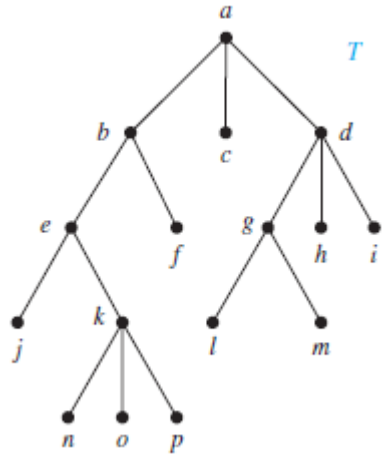

Figure 3: The Ordered Rooted Tree T

Solution

- The steps of the postorder traversal of the ordered rooted tree *T* are shown in Figure 8.
- The postorder traversal begins with the postorder traversal of the subtree with root *b*, the postorder traversal of the subtree with root *c*, which is just *c*, the postorder traversal of the subtree with root *d*, followed by the root *a*.
- Therefore, the postorder traversal of *T* is
- *j* , *n*, *o*, *p*, *k*, *e*, *f* , *b*, *c*, *l*, *m*, *g*, *h*, *i*, *d*, *a*.

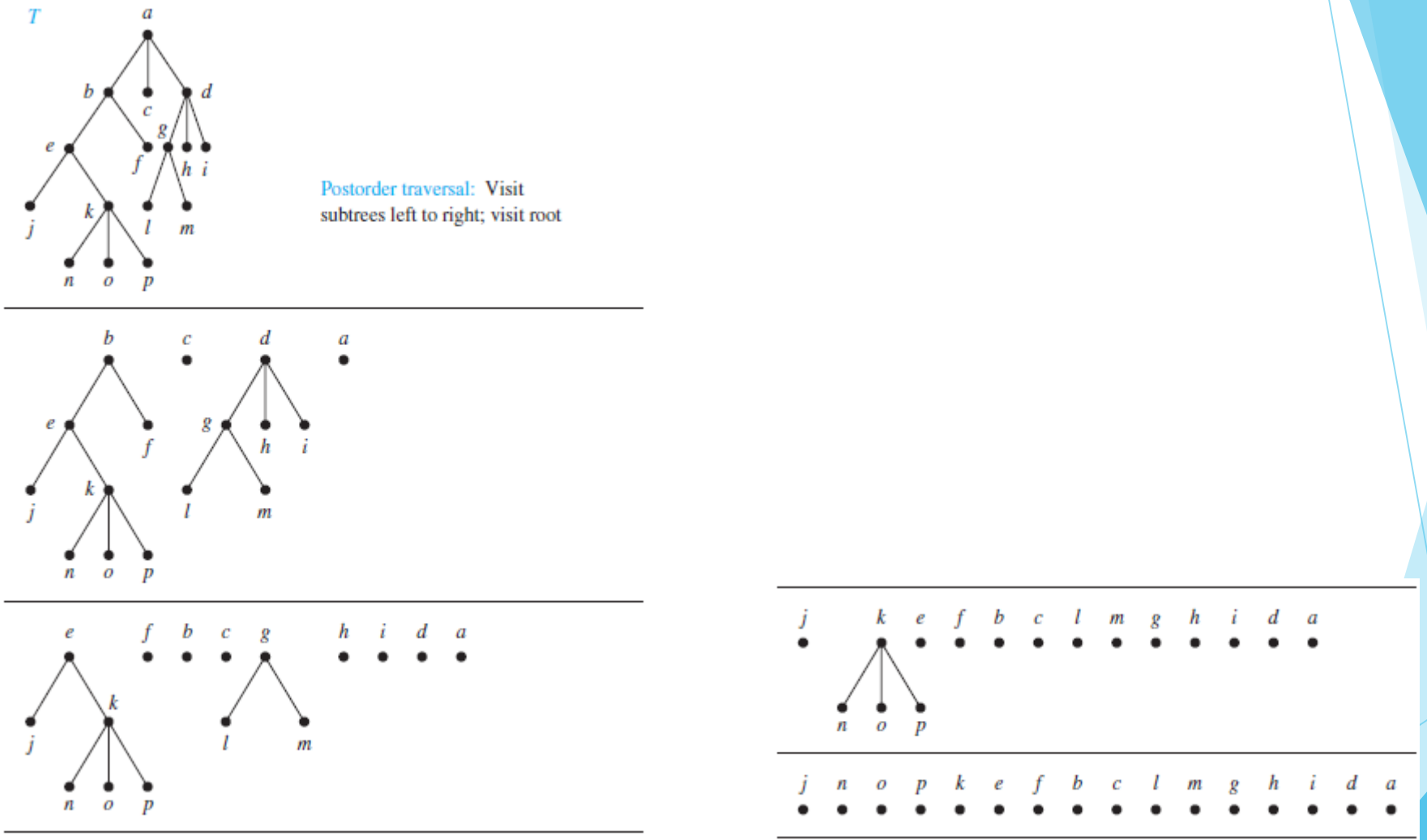

Figure 8 The Inorder Traversal of *T* .

Infix, Prefix, and Postfix Notations

- We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees.
- For instance, an arithmetic expression involving the operators + (addition), − (subtraction), ∗ (multiplication), / (division), and ↑ (exponentiation).
- We will use parentheses to indicate the order of the operations.
- An ordered rooted tree can be used to represent such expressions, where the internal vertices represent operations, and the leaves represent the variables or numbers.
- Each operation operates on its left and right subtrees.

Example 5

- What is the ordered rooted tree that represents the expression $((x + y)\uparrow 2) + ((x - 4)/3)$?

Solution

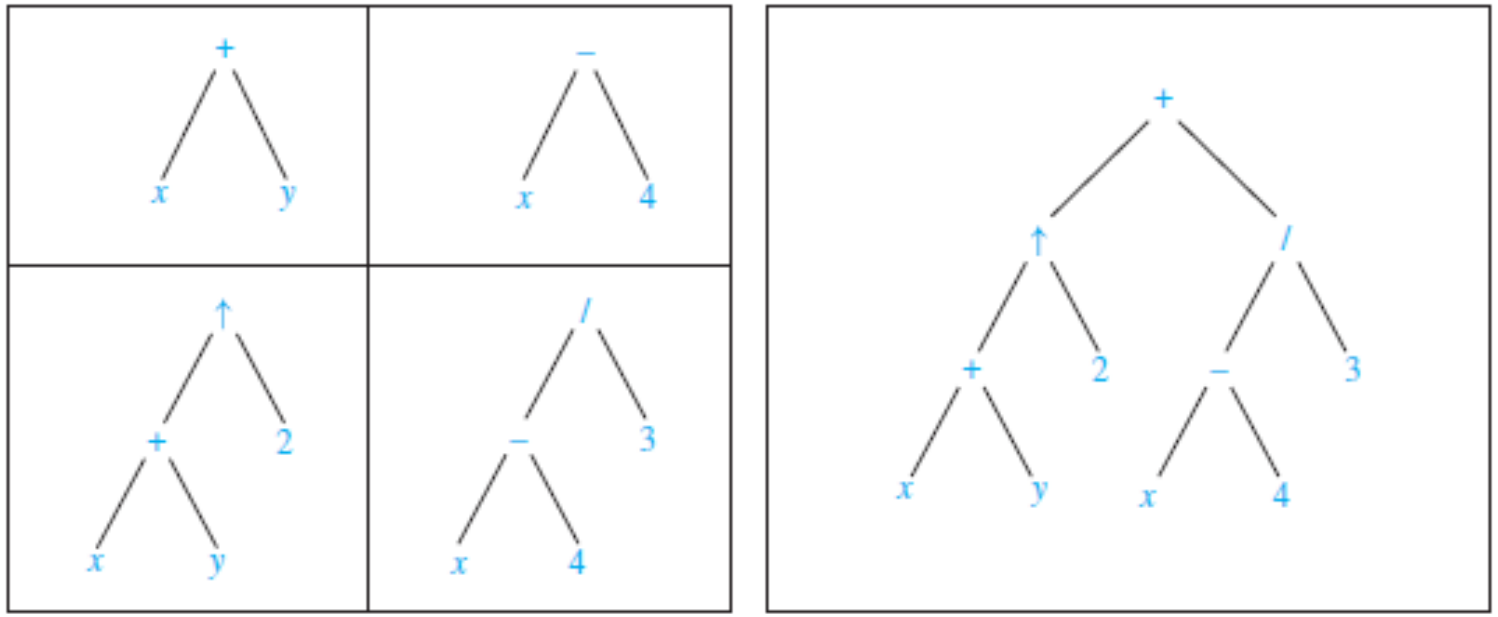

Figure 9 A Binary Tree Representing *((x + y) ↑ 2) + ((x − 4)/3)*.

Inorder traversals of the binary trees

- An inorder traversal of the binary tree representing an expression produces the original expression with the elements and operations in the same order as they originally occurred, except for unary operations, which instead immediately follow their operands.
- For instance, inorder traversals of the binary trees in Figure 10, which represent the expressions $(x + y)/(x + 3)$, $(x + (y/x)) + 3$, and $x + (y/(x + 3))$, all lead to the infix expression $x + y/x + 3$.
- To make such expressions unambiguous it is necessary to include parentheses in the inorder traversal whenever we encounter an operation.
- The fully parenthesized expression obtained in this way is said to be in **infix form.**
- We obtain the **prefix form** of an expression when we traverse its rooted tree in preorder.

Infix and Prefix

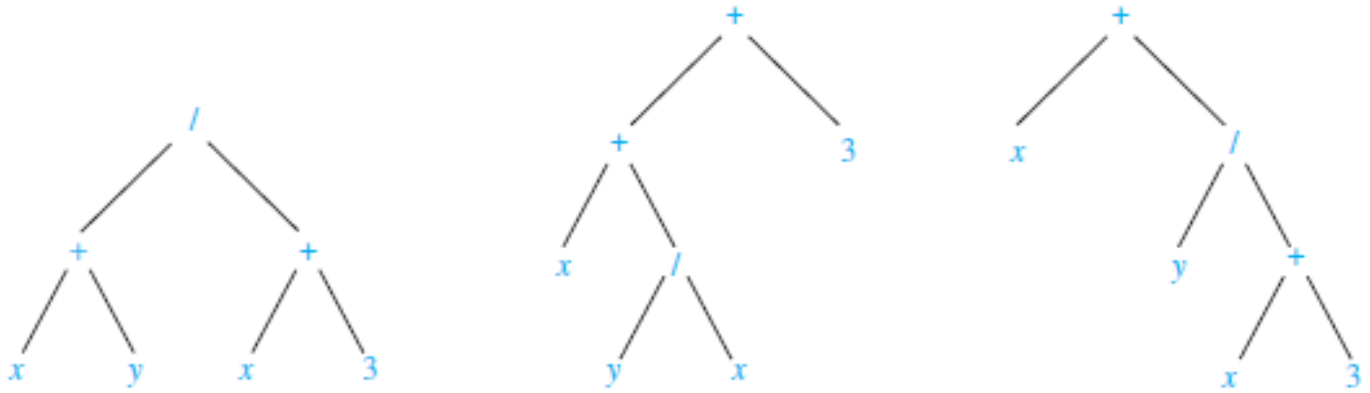

Figure 10 Rooted Trees Representing *(x + y)/(x + 3)*, *(x + (y/x))* + 3, and *x + (y/(x + 3))*.

Example 6

- What is the prefix form for $((x + y) \uparrow 2) + ((x - 4)/3)$?

Solution

- We obtain the prefix form for this expression by traversing the binary tree that represents it in preorder, shown in Figure 9. This produces
- + ↑ + *x y* 2 / − *x* 4 3.
- In the prefix form of an expression, a binary operator, such as +, precedes its two operands. Hence, we can evaluate an expression in prefix form by working from right to left. When we encounter an operator, we perform the corresponding operation with the two operands immediately to the right of this operand. Also, whenever an operation is performed, we consider the result a new operand.

Solution (cont.)

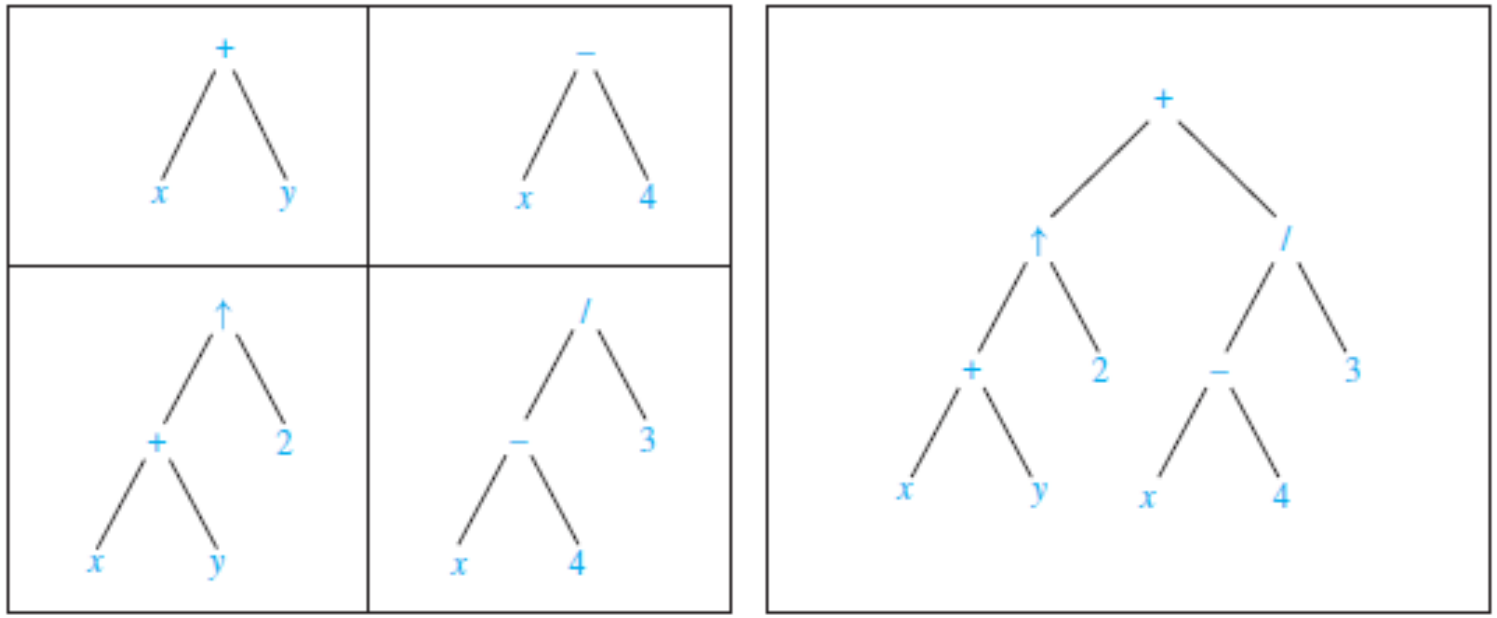

Figure 9 A Binary Tree Representing *((x + y) ↑ 2) + ((x − 4)/3)*.

Example 7

- What is the value of the prefix expression + − ∗ 2 3 5/↑ 2 3 4?

Solution

- The steps used to evaluate this expression by working right to left, and performing operations using the operands on the right, are shown in Figure 11. The value of this expression is 3.
- We obtain the **postfix form** of an expression by traversing its binary tree in postorder.

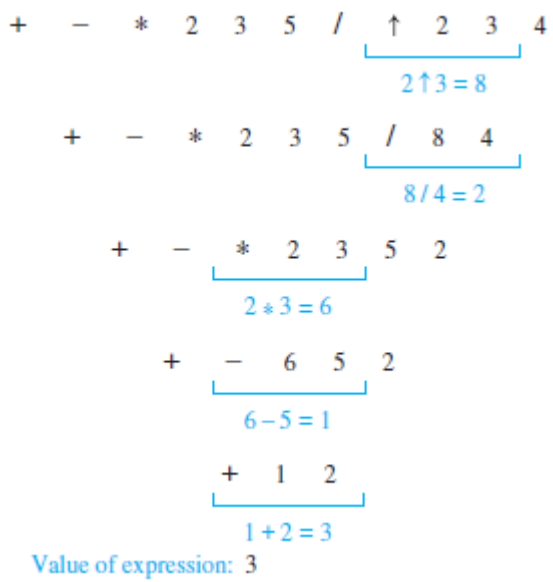

Figure 11 Evaluating a Prefix Expression.

Example 8

- What is the postfix form of the expression $((x + y) \uparrow 2) + ((x - 4)/3)$?

Solution

- The postfix form of the expression is obtained by carrying out a postorder traversal of the binary tree for this expression, shown in Figure 9.
- This produces the postfix expression:
- $x\ y + 2 \uparrow x\ 4 - 3\ /\ +$.
- In the postfix form of an expression, a binary operator follows its two operands. So, to evaluate an expression from its postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After an operation is carried out, the result of this operation becomes a new operand.

Example 9

- What is the value of the postfix expression 7 2 3 $*$ − 4 $\uparrow$ 9 3/+?

Solution

- The steps used to evaluate this expression by starting at the left and carrying out operations when two operands are followed by an operator are shown in Figure 12. The value of this expression is 4.

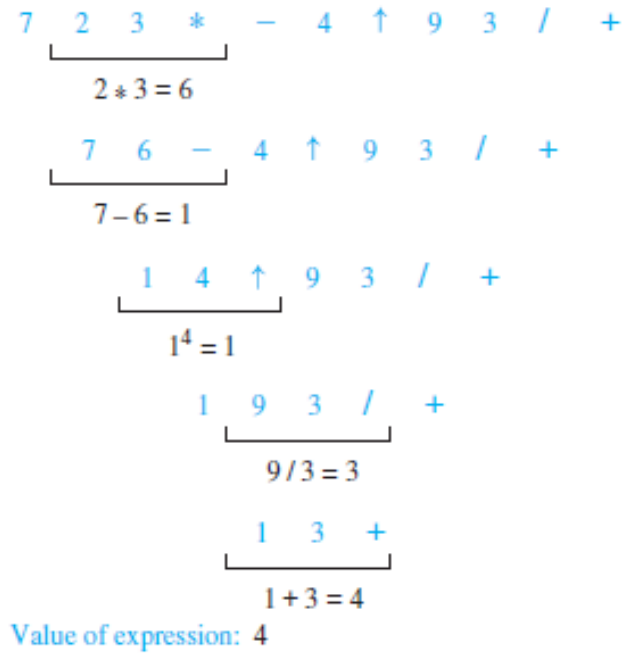

Figure 12 Evaluating a Postfix Expression.

Representing Compound Propositions

- Rooted trees can be used to represent other types of expressions, such as those representing compound propositions and combinations of sets.
- In these examples unary operators, such as the negation of a proposition, occur.
- To represent such operators and their operands, a vertex representing the operator and a child of this vertex representing the operand are used.

Example 10

- Find the ordered rooted tree representing the compound proposition

  $(\neg(p \wedge q)) \leftrightarrow (\neg p \vee \neg q)$.
- Then use this rooted tree to find the prefix, postfix, and infix forms of this expression.

Solution

- The rooted tree for this compound proposition is constructed from the bottom up. First, subtrees for $\neg p$ and $\neg q$ are formed (where $\neg$ is considered a unary operator).
- Also, a subtree for $p \wedge q$ is formed. Then subtrees for $\neg(p \wedge q)$ and $(\neg p) \vee (\neg q)$ are constructed.
- Finally, these two subtrees are used to form the final rooted tree. The steps of this procedure are shown in Figure 13.
- The prefix, postfix, and infix forms of this expression are found by traversing this rooted tree in preorder, postorder, and inorder (including parentheses), respectively. These traversals give
- $\leftrightarrow\neg\wedge pq \vee\neg p\neg q, pq \wedge\neg p\neg q\neg \vee\leftrightarrow$, and $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$, respectively.

Solutions (cont.)

- Because prefix and postfix expressions are unambiguous and because they can be evaluated easily without scanning back and forth, they are used extensively in computer science.
- Such expressions are especially useful in the construction of compilers.

Solution (cont.)

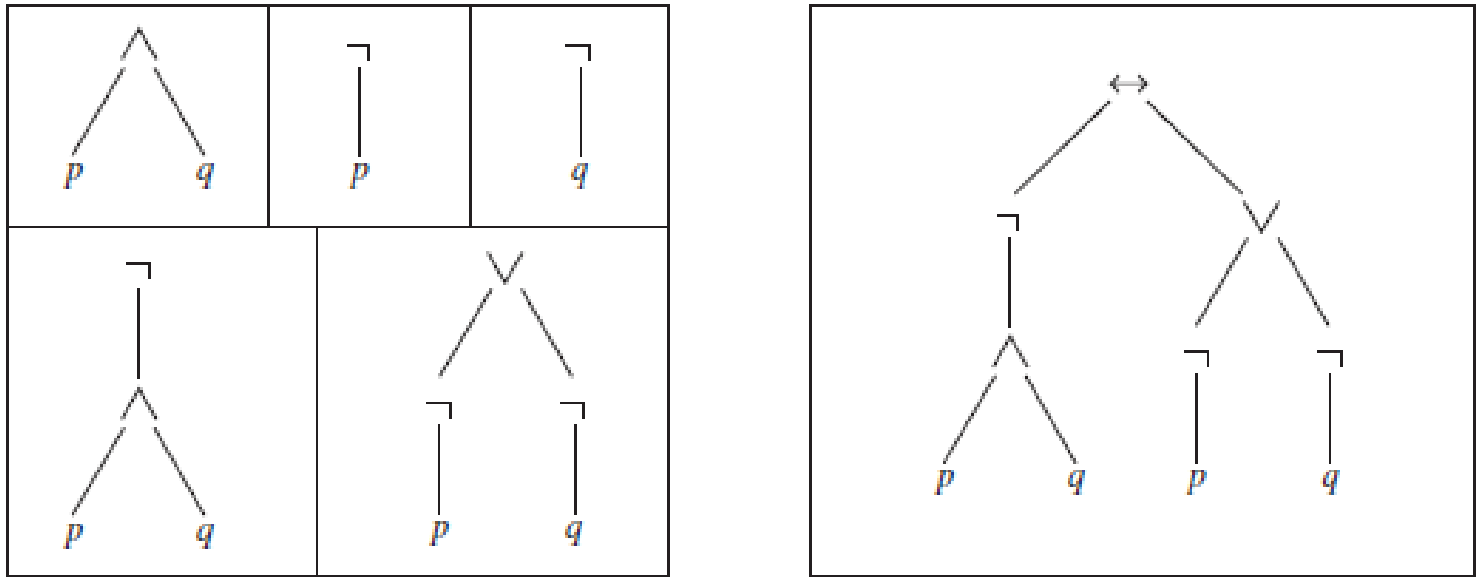

Figure 13 Constructing the Rooted Tree for a Compound Proposition.