Chapter 10
Trees

Objectives

- 10.1- Introduction to Trees
- 10.2- Applications of Trees
- 10.3- Tree Traversal

10.1- Introduction to Trees

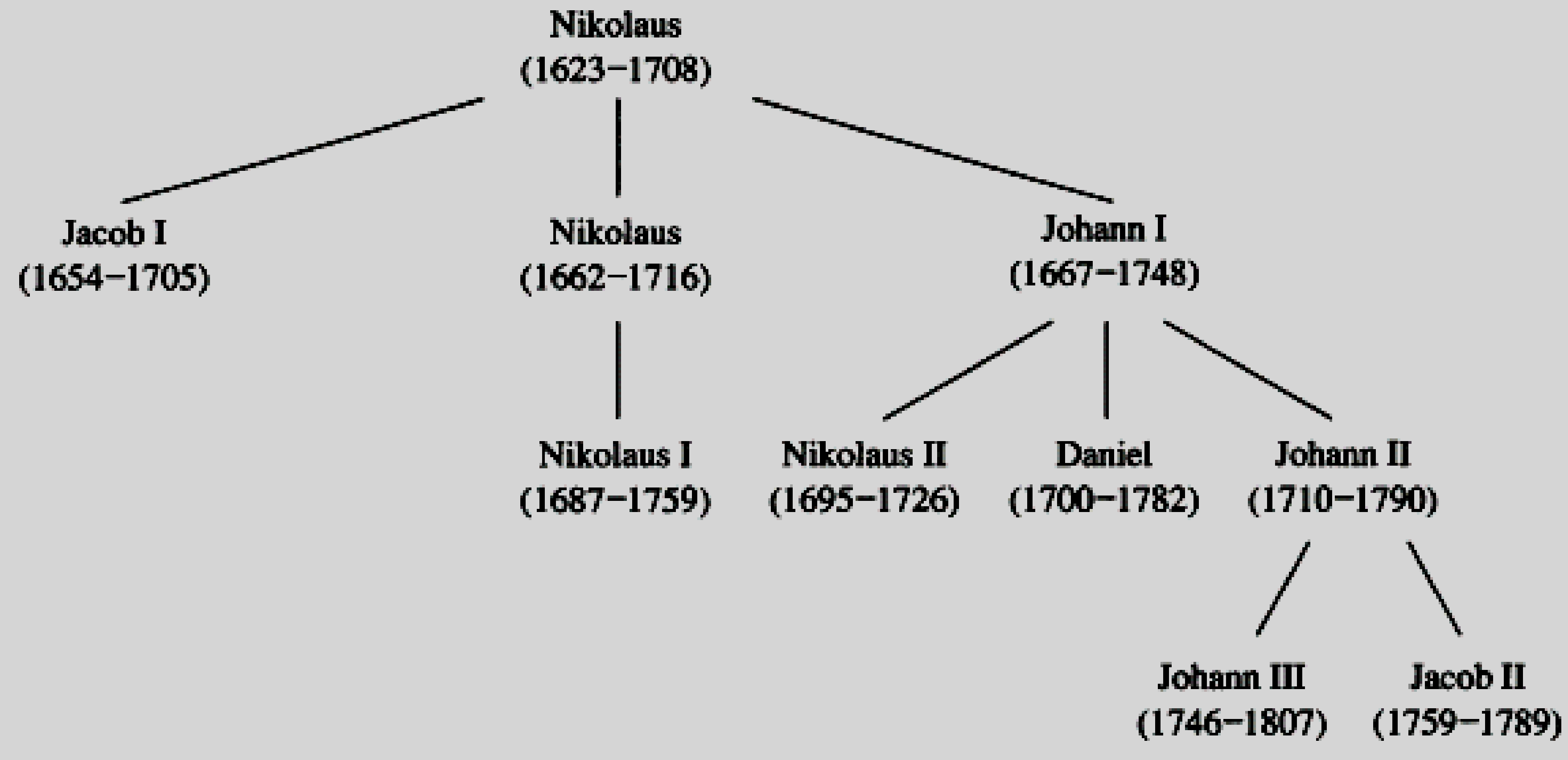


FIGURE 1 **The Bernoulli Family of Mathematicians.**

Introduction to Trees…

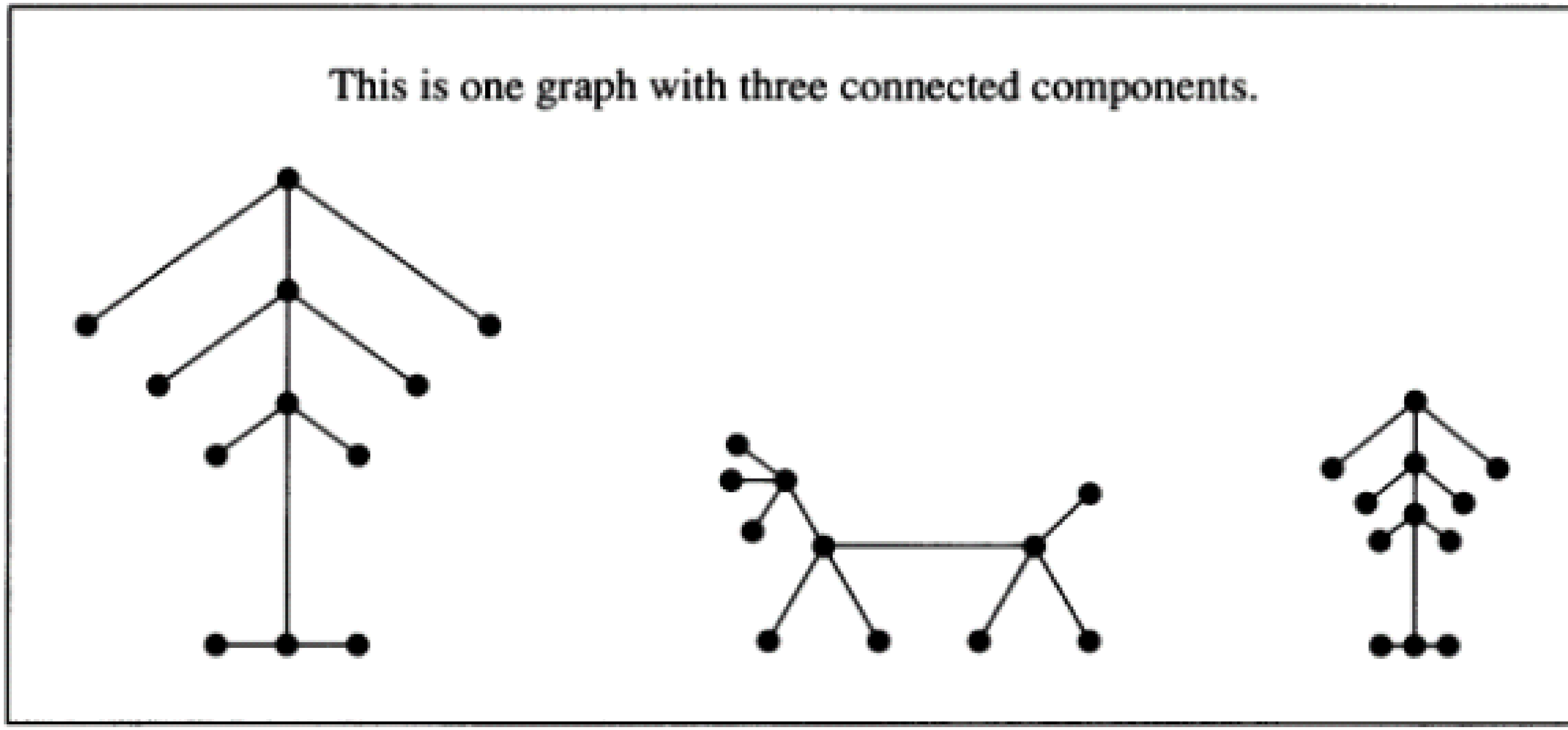


FIGURE 3 Example of a Forest.

Introduction to Trees

DEFINITION 1

A *tree* is a connected undirected graph with no simple circuits.

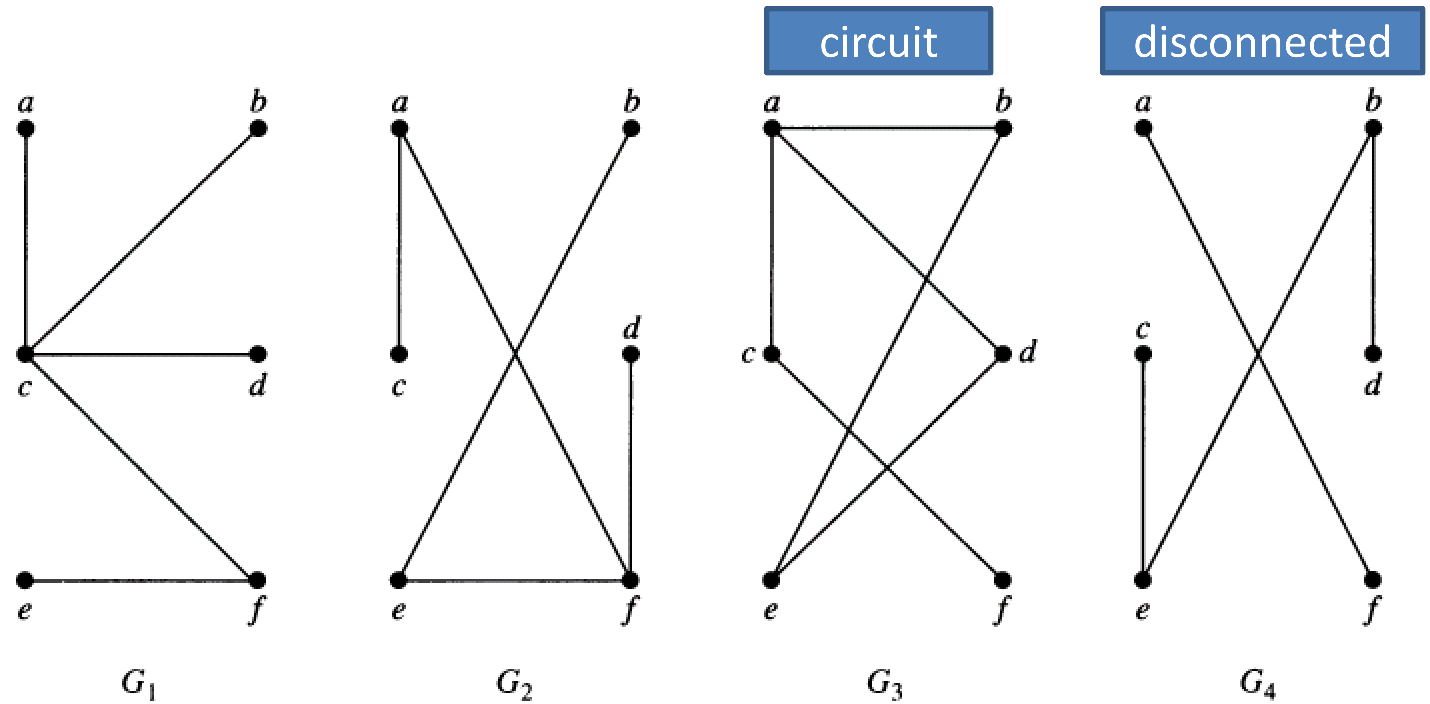


FIGURE 2 **Examples of Trees and Graphs That Are Not Trees.**

Introduction to Trees…

THEOREM 1 Proof: page 684

An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

DEFINITION 2

A *rooted tree* is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

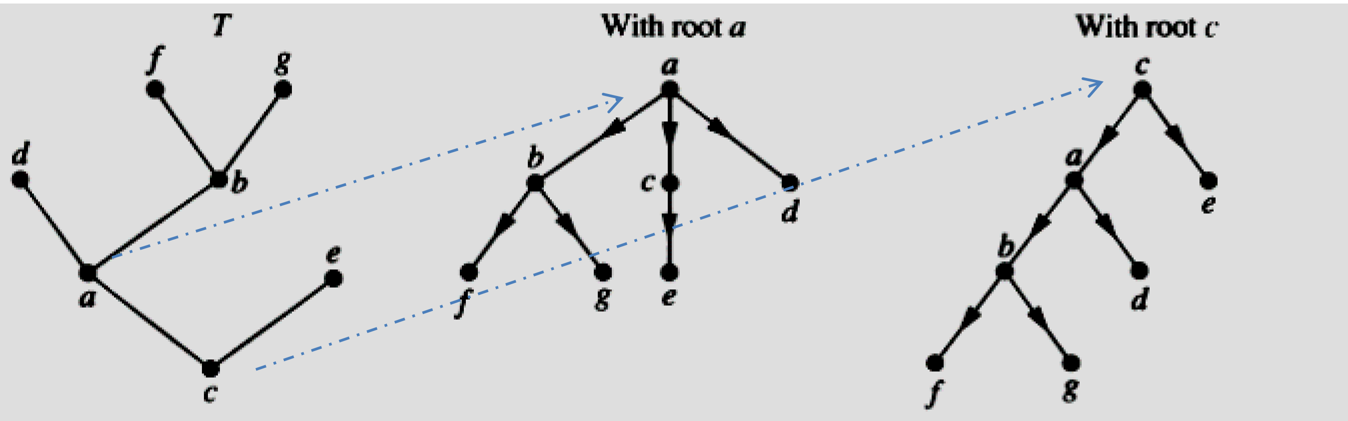


FIGURE 4 A Tree and Rooted Trees Formed by Designating Two Roots.

Introduction to Trees…

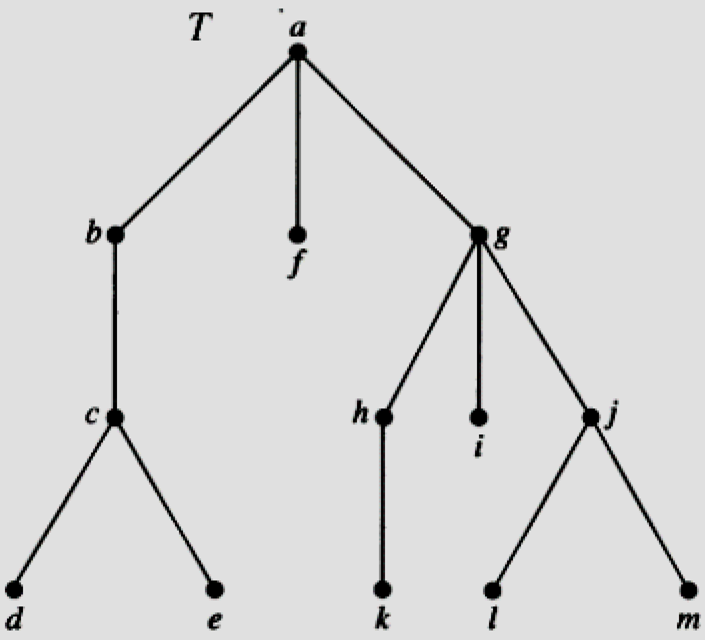


FIGURE 5 A Rooted Tree T.

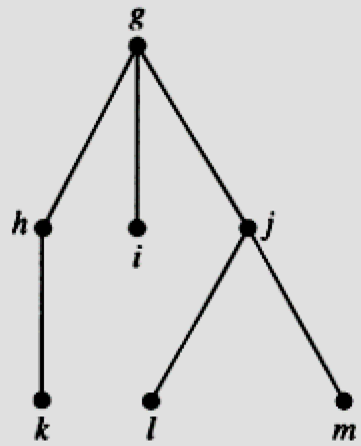


FIGURE 6 The Subtree Rooted at g.

Some terminologies:

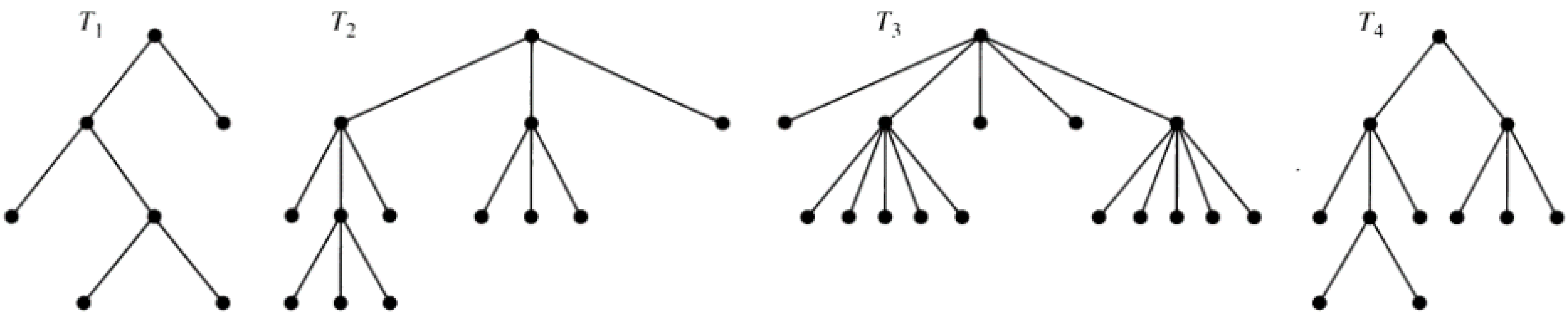
Page 686

- Subtree
- Root node (vertex)
- Internal node
- Leaf
- Parent
- Child
- Siblings
- Descendants
- Ancestors

Introduction to Trees…

DEFINITION 3

A rooted tree is called an *m-ary tree* if every internal vertex has no more than m children. The tree is called a *full m-ary tree* if every internal vertex has exactly m children. An m-ary tree with $m = 2$ is called a *binary tree*.



Full Binary

Full Ternary

Full 5-ary

3-ary

Some terminologies on binary tree:

Left child, right child, left subtree, right subtree

Introduction to Trees…

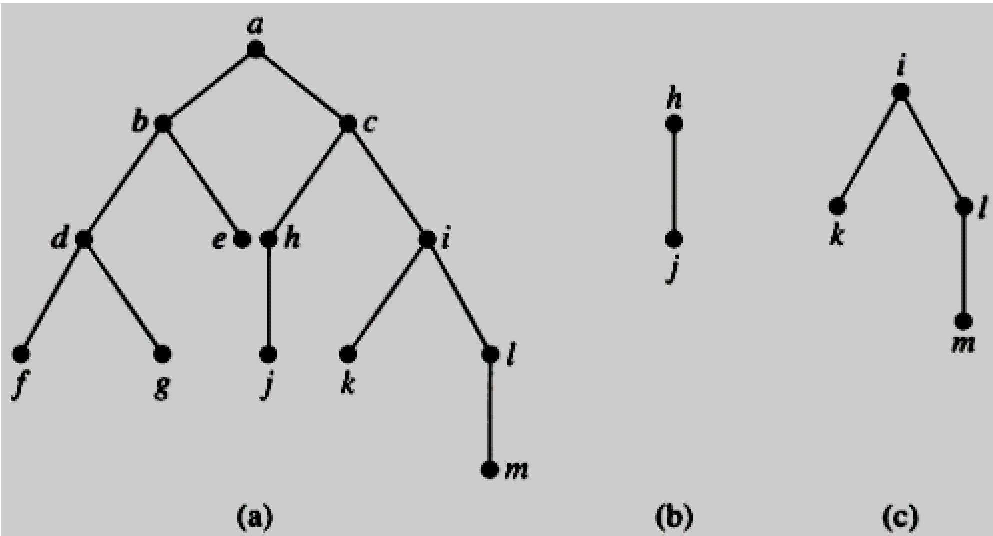
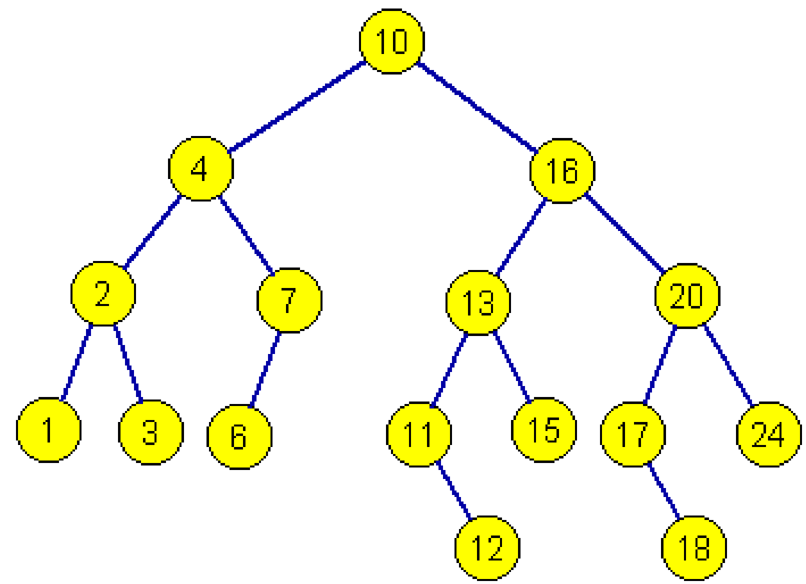
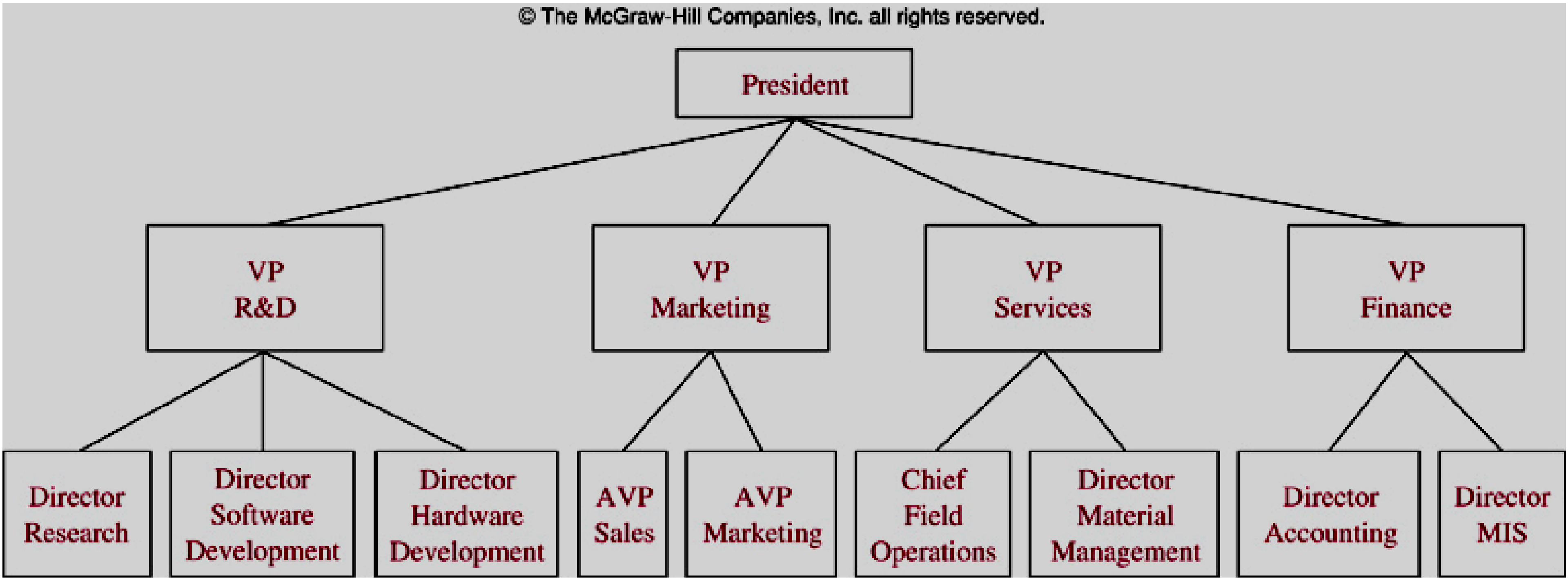


FIGURE 8 **A Binary Tree *T* and Left and Right Subtrees of the Vertex *c*.**



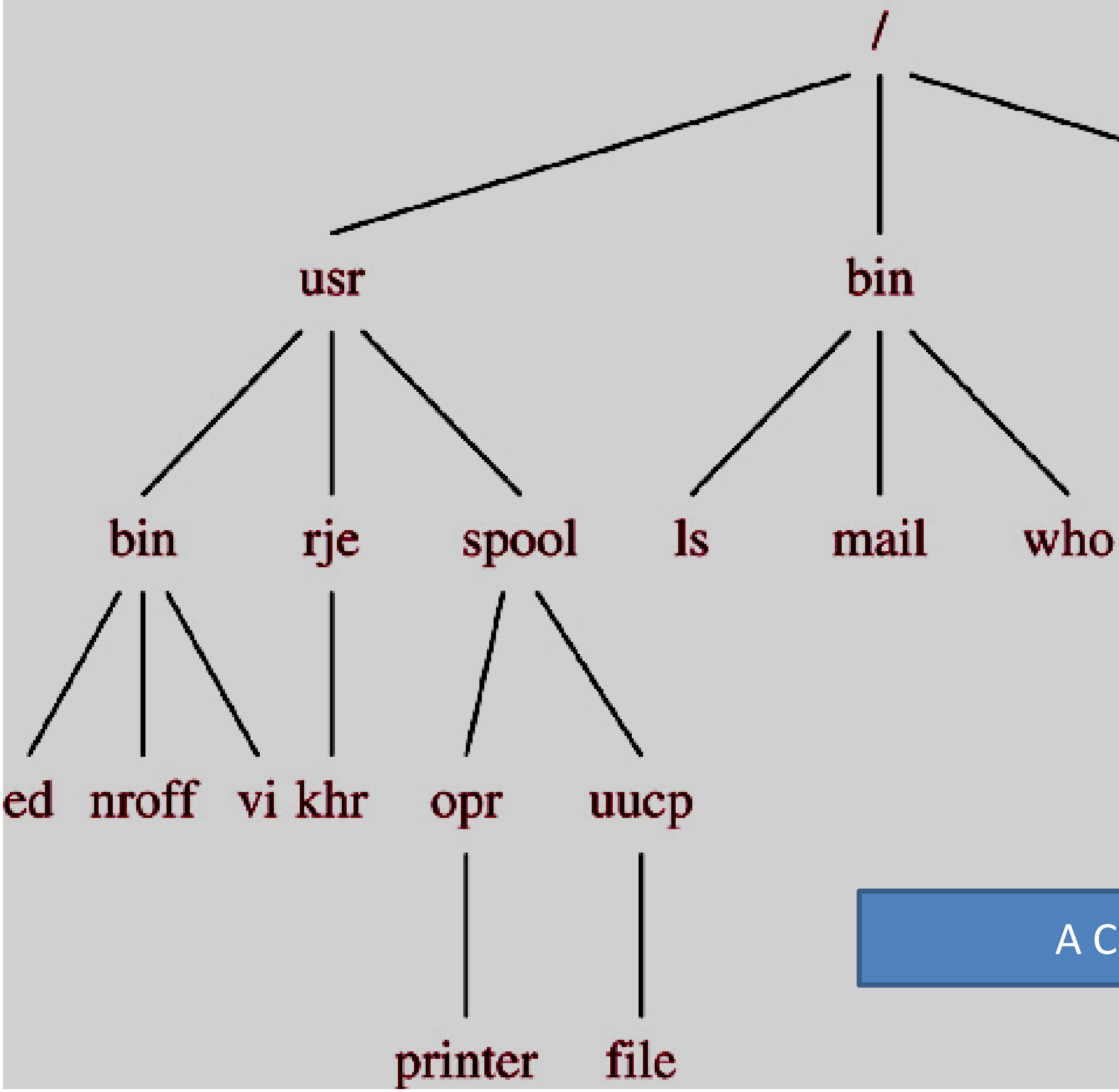
Ordered rooted tree

Some Tree Models



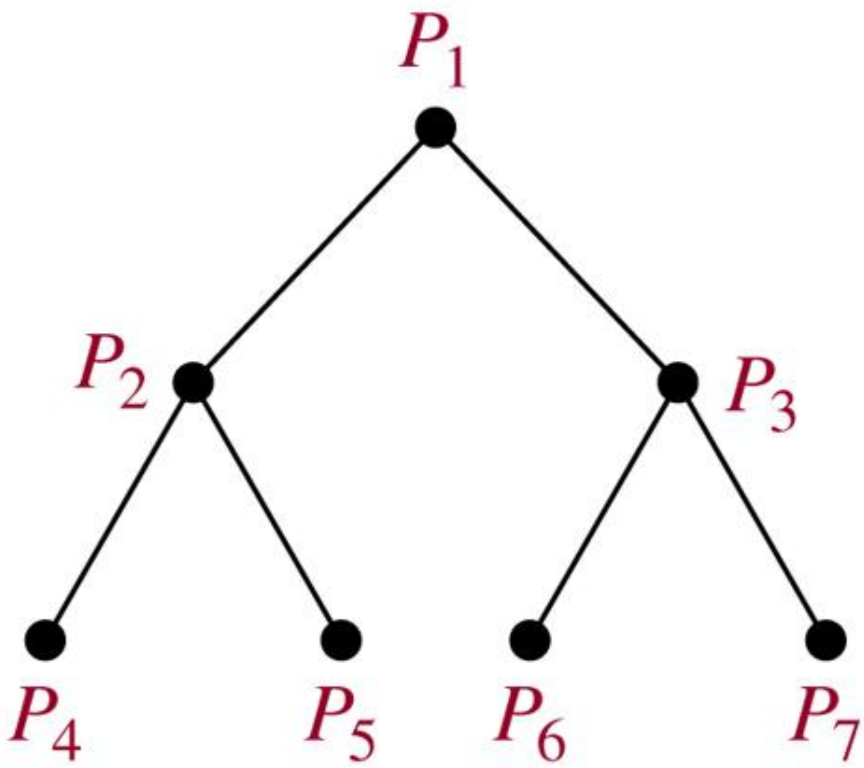
A Organizational Tree

The root is the root directory /
Internal vertices are directories
Leaves are files



A Computer File System

Some Tree Models



A Tree-connected Network of seven Processors

Properties of Trees

THEOREM 2

A tree with n vertices has $n-1$ edges.

Using Mathematic Induction.

Let nE be number of edges.

P(n): If the tree T having n vertices then nE=n-1

Basic step:

P(1): n=1, tree has the root node only → nE= 0 = n-1 → P(1) true

Induction step:

Suppose that P(k) is true for all k>=1, ie nE=k-1

Add a leaf vertex v to the tree T so that T having k+1 verteices still is a tree.

Let w be the parent of v.

Because T is connected and has no simple circuit → there is only one new edge between u and v → nE= (k-1)+1 = k .

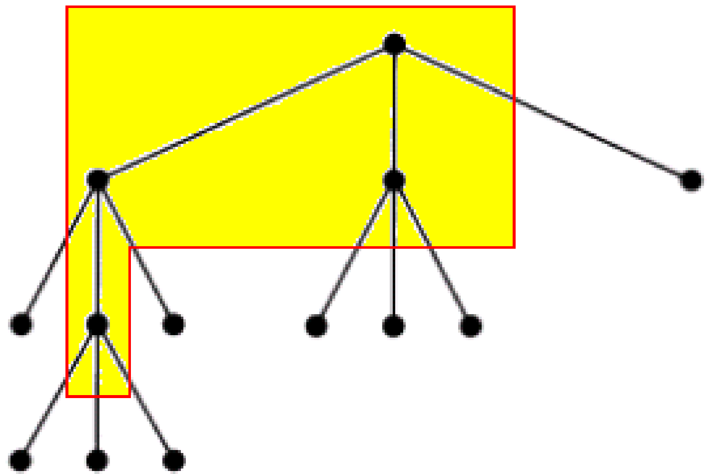
→ P(k+1) true

Proved.

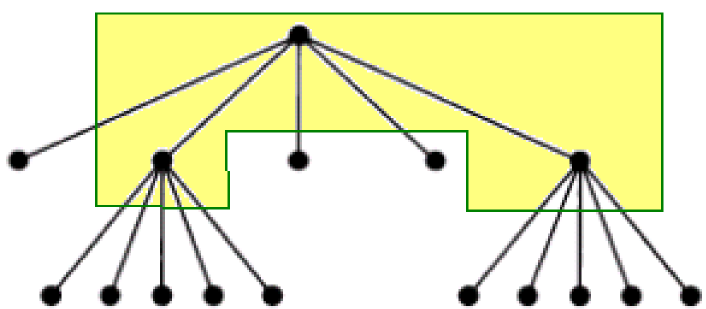
Introduction to Trees…

THEOREM 3

A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.



m=3
i=4
n= 3.4+1=13



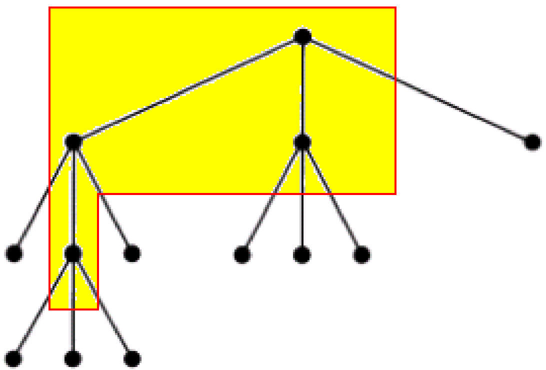
m=5
i=3
n= 3.5+1=16

Introduction to Trees…

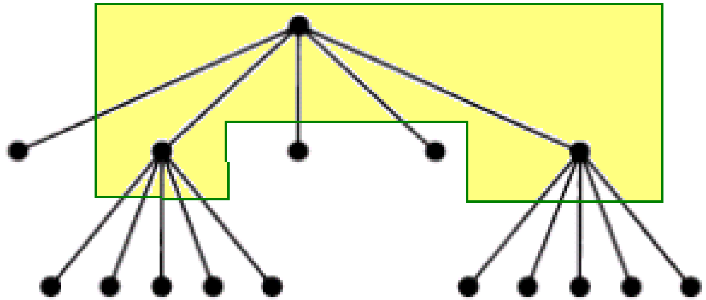
THEOREM 4

A full m-ary tree with

(*i*) n vertices has $i = (n-1)/m$ internal vertices and $l = [(m-1)n+1]/m$ leaves,
(*ii*) i internal vertices has $n = mi + 1$ vertices and $l = (m-1)i + 1$ leaves,
(*iii*) l leaves has $n = (ml-1)/(m-1)$ vertices and $i = (l-1)/(m-1)$ internal vertices.



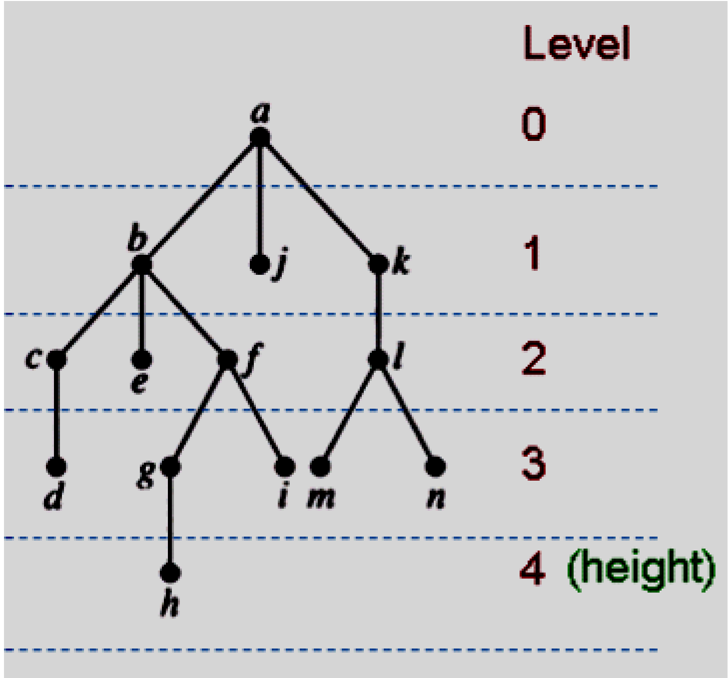
m=3
n=13
i=12/3= 4
l= [(3-1)13+1]/3=9



m=5
n=16
i= 15/5=3
l= (4.16+1)/5= 13

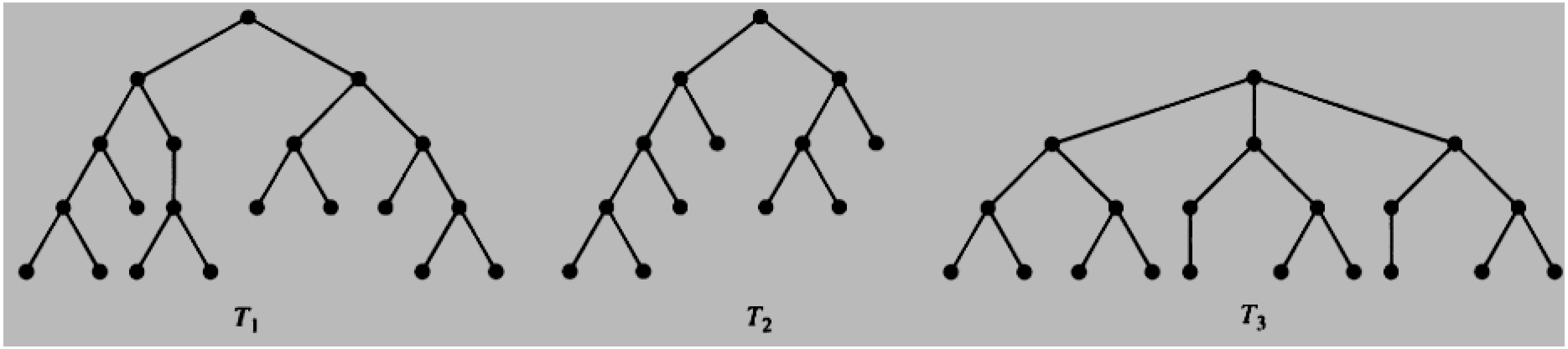
Introduction to Trees…

- **Level of a vertex: The length of the path from the root to this vertex.**
- **Height of Tree: The maximum of levels of vertices = The length of the longest path from the root to any vertex.**



Introduction to Trees…

A m-ary tree is called *balanced* if all leaves are at levels h or h-1.



h=4
All leafs are at levels 3, 4
→ Balanced

h=4
All leafs are at levels 2,3, 4
→ Not Balanced

h=3
All leafs are at levels 3
→ Balanced

Introduction to Trees…

THEOREM 5: (Proof: page 692)

There are at most m^h leaves in an m-ary tree of height **h**.

COROLLARY 1

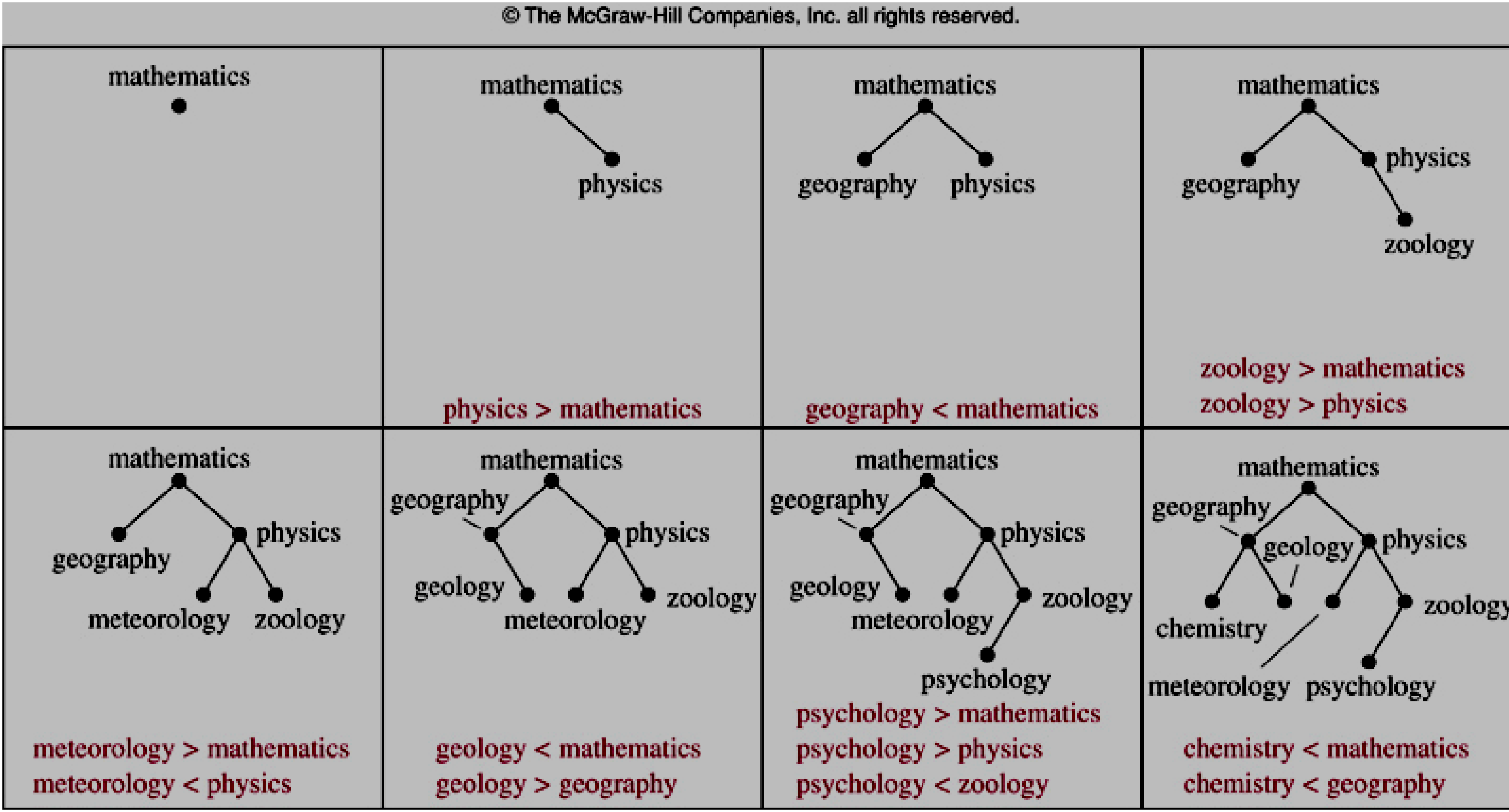
Proof: page 693

If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$. (We are using the ceiling function here. Recall that $\lceil x \rceil$ is the smallest integer greater than or equal to x.)

10.2- Applications of Trees

- Binary Search Trees
- Decision Trees
- Prefix Codes

Constructing a Binary Search Tree



Construct a binary search tree for numbers: 23, 16, 43, 5, 9, 1, 6, 2, 33, 27.

Algorithm for inserting an element to BST

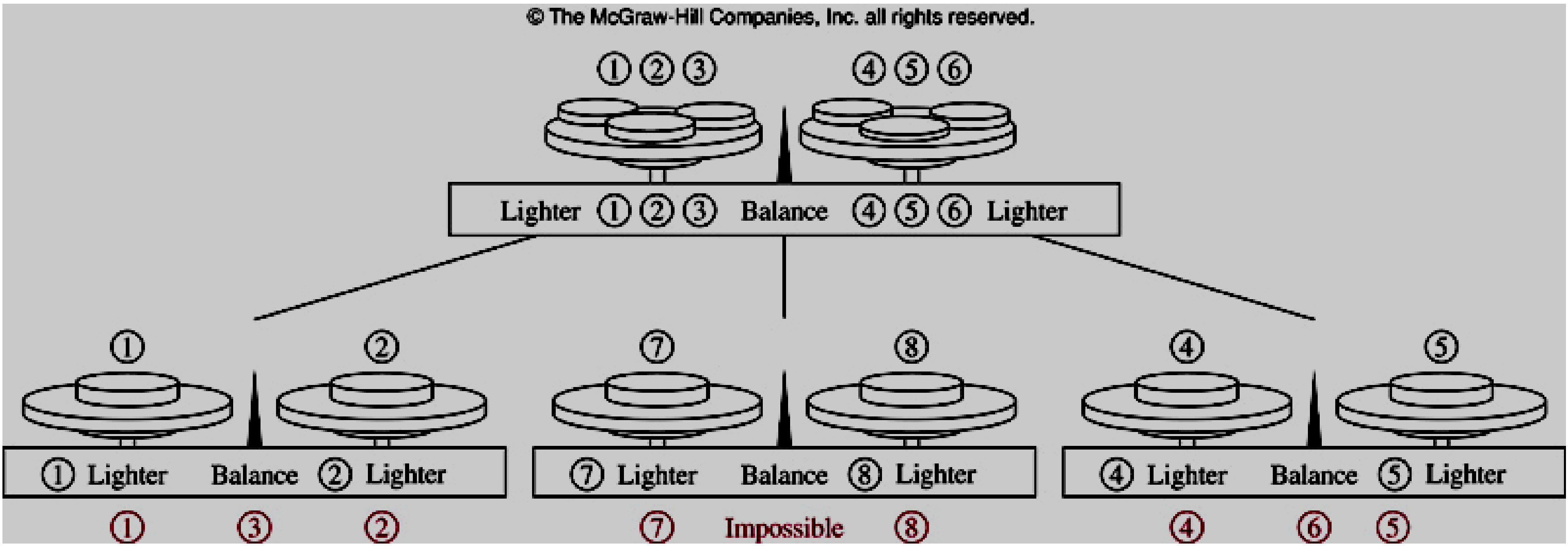
ALGORITHM 1 Locating and Adding Items to a Binary Search Tree.

procedure *insertion*(T: binary search tree, x: item)
v := root of T
{a vertex not present in T has the value *null*}
while $v \neq$ *null* and *label*(v) $\neq x$
begin
    if $x <$ *label*(v) **then**
        if left child of $v \neq$ *null* **then** v := left child of v
        else add *new vertex* as a left child of v and set v := *null*
    else
        if right child of $v \neq$ *null* **then** v := right child of v
        else add *new vertex* as a right child of v to T and set v := *null*
end
if root of T = *null* **then** add a vertex v to the tree and label it with x
else if v is null or *label*(v) $\neq x$ **then** label *new vertex* with x and let v be this new vertex
{v = location of x}

Complexity: O(logn)
Proof: page 698

Decision Trees

The Counterfeit Coin Problem



Decision Trees:
Sorting based on Binary Comparisons

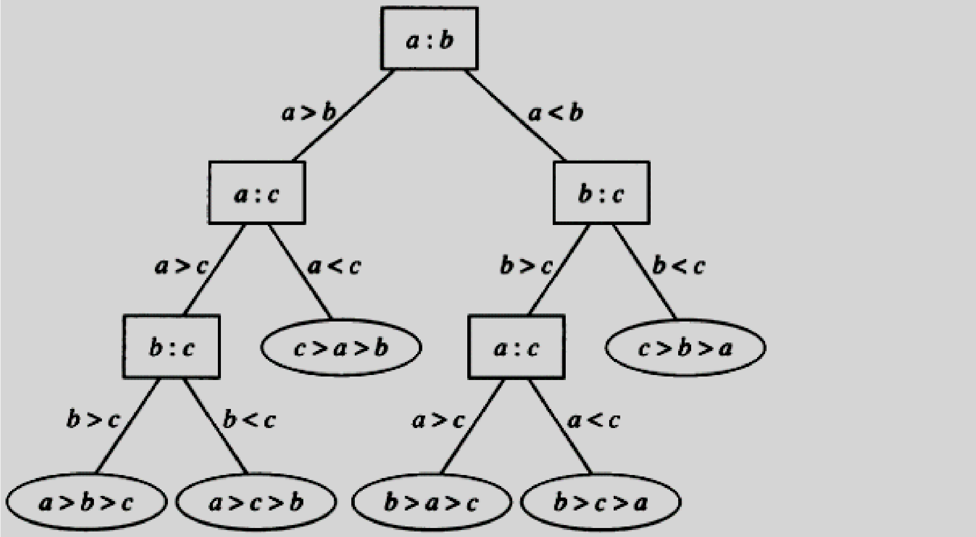


FIGURE 4 A Decision Tree for Sorting Three Distinct Elements.

THEOREM 1

A sorting algorithm based on binary comparisons requires at least $\lceil \log n! \rceil$ comparisons.

COROLLARY 1

The number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n \log n)$.

THEOREM 2

The average number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n \log n)$.

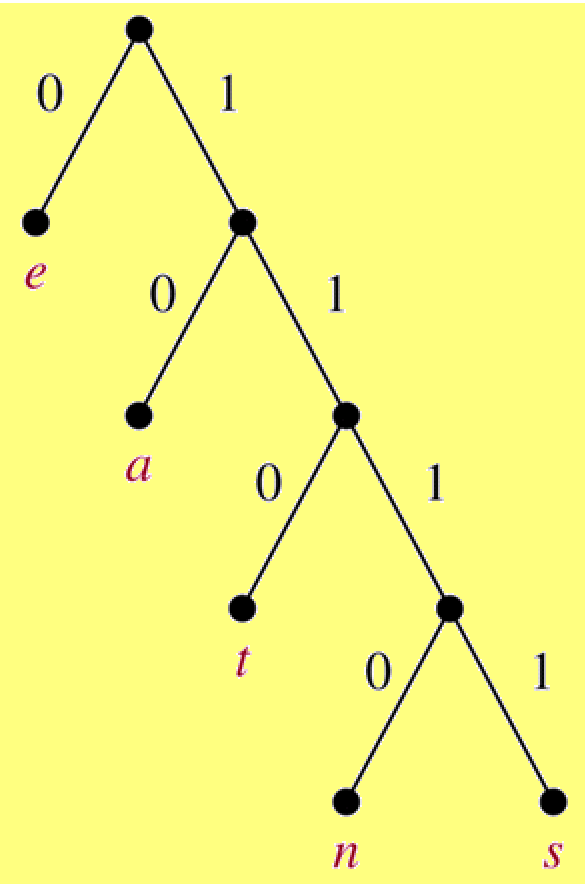
Prefix Codes

- Introduction to Prefix Codes
- Huffman Coding Algorithm

Prefix Codes: Introduction

- English word “sane” is stores 1 byte/character → 4-byte memory block is needed (32 bits).
- There are 26 characters → We can use 5 bit only to store a character ($2^5=32$)
- The word “sane” can be stored in 20 bits
- May we can store this word in fewer bit?

Prefix Codes: Introduction



- Construct a binary tree with a prefix code.
- “sane” will be store as

  11111011100 ➔ 11 bits

11111011100 : s

11111011100 : a

11111011100 : n

11111011100 : e

➔Compression factor: 32/11 ~ 3

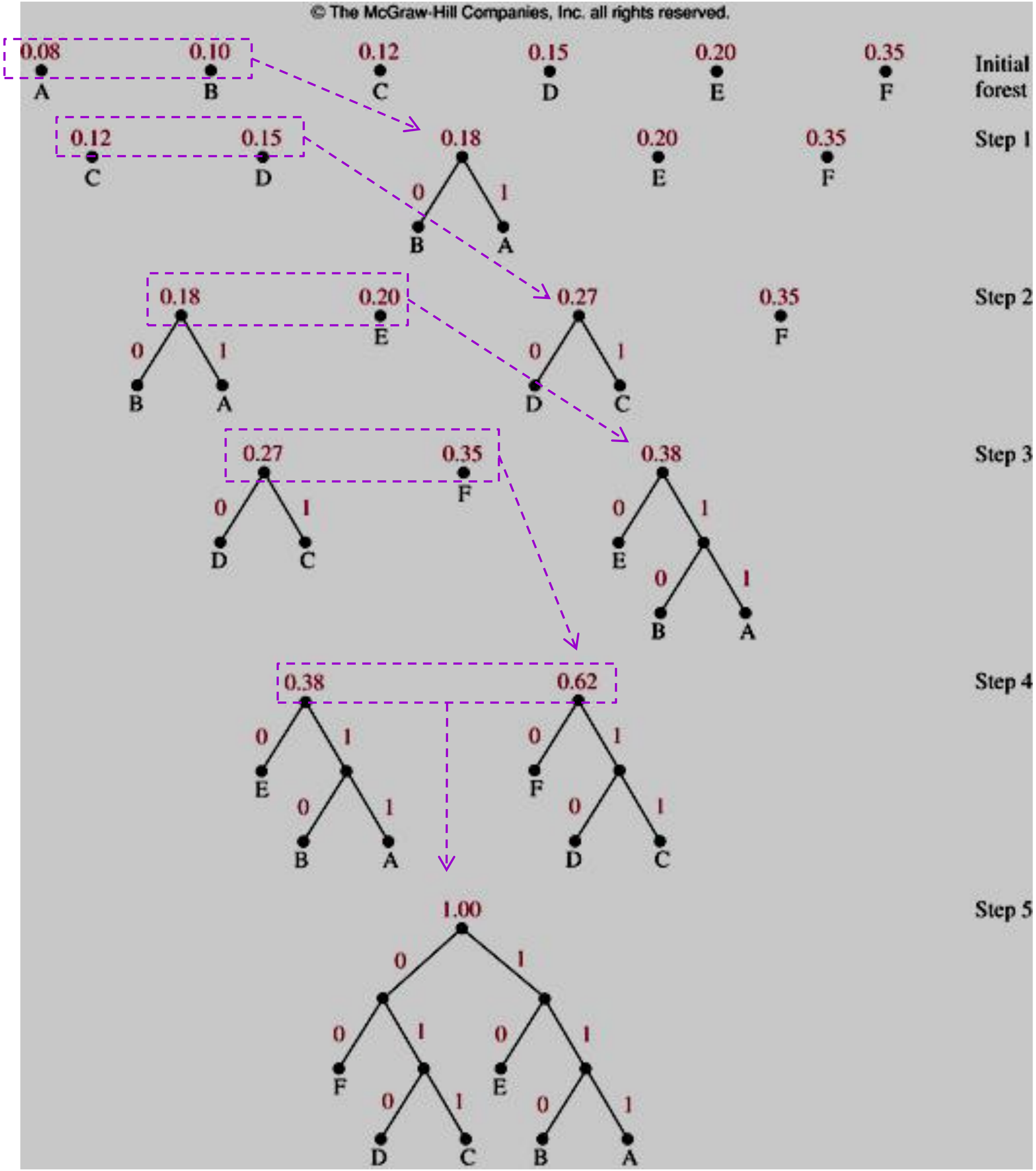
Prefix Codes: Huffman Coding Algorithm

- Counting occurrences of characters in a text ➔ frequencies (probabilities) of each character.
- Constructing a binary tree representing prefix codes of characters.

➔ The set of binary codes representing each character.

➔ Coding source text

Prefix Codes: Huffman Coding Algorithm



Prefix Codes: Huffman Coding Algorithm

ALGORITHM 2 Huffman Coding.

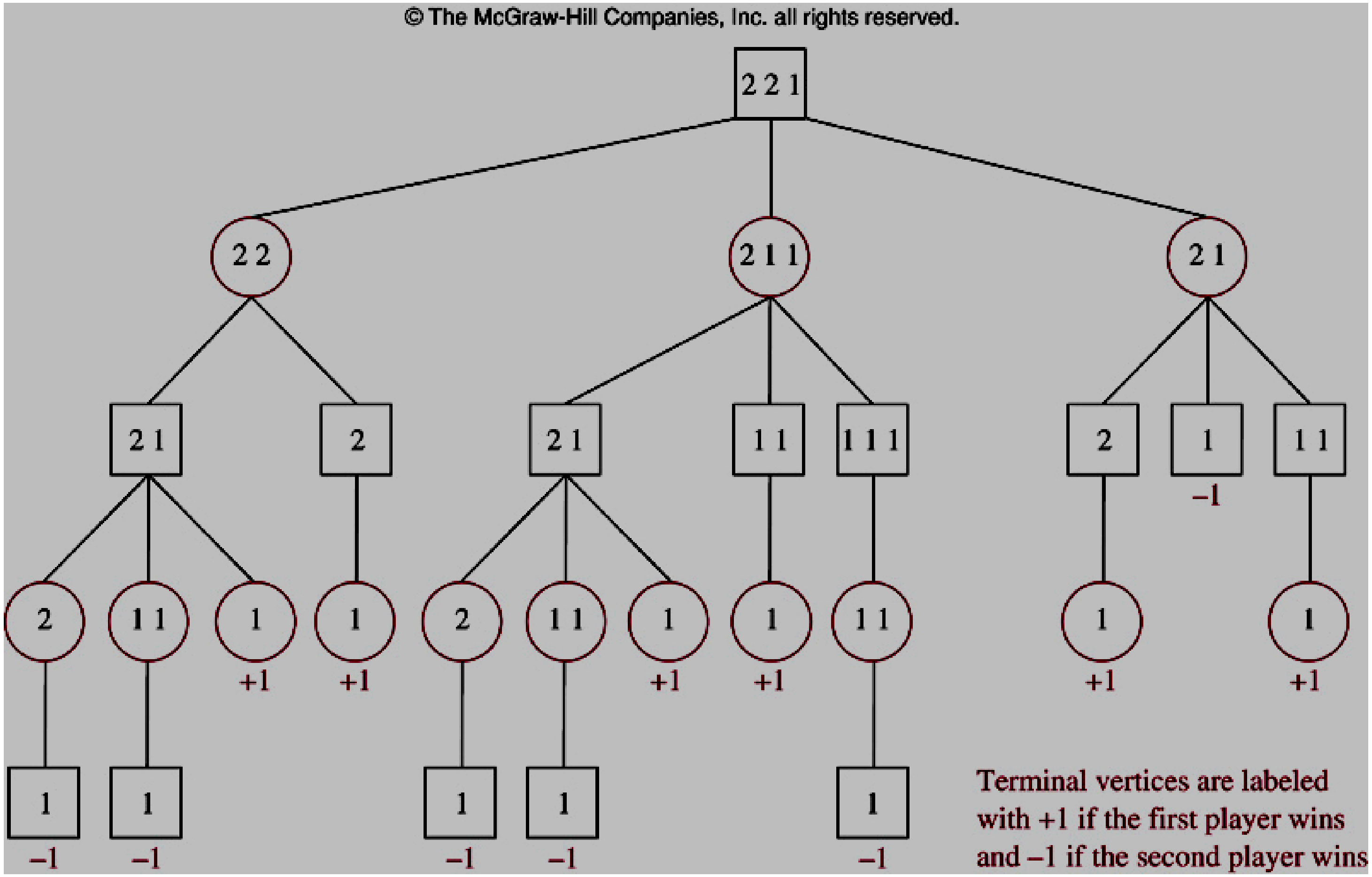
procedure *Huffman*(C: symbols a_i with frequencies w_i, $i = 1, \ldots, n$)
F := forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
while F is not a tree
begin
    Replace the rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree having a new root that has T as its left subtree and T' as its right subtree. Label the new edge to T with 0 and the new edge to T' with 1.
    Assign $w(T) + w(T')$ as the weight of the new tree.
end
{the Huffman coding for the symbol a_i is the concatenation of the labels of the edges in the unique path from the root to the vertex a_i}

Game Trees: The Game Nim

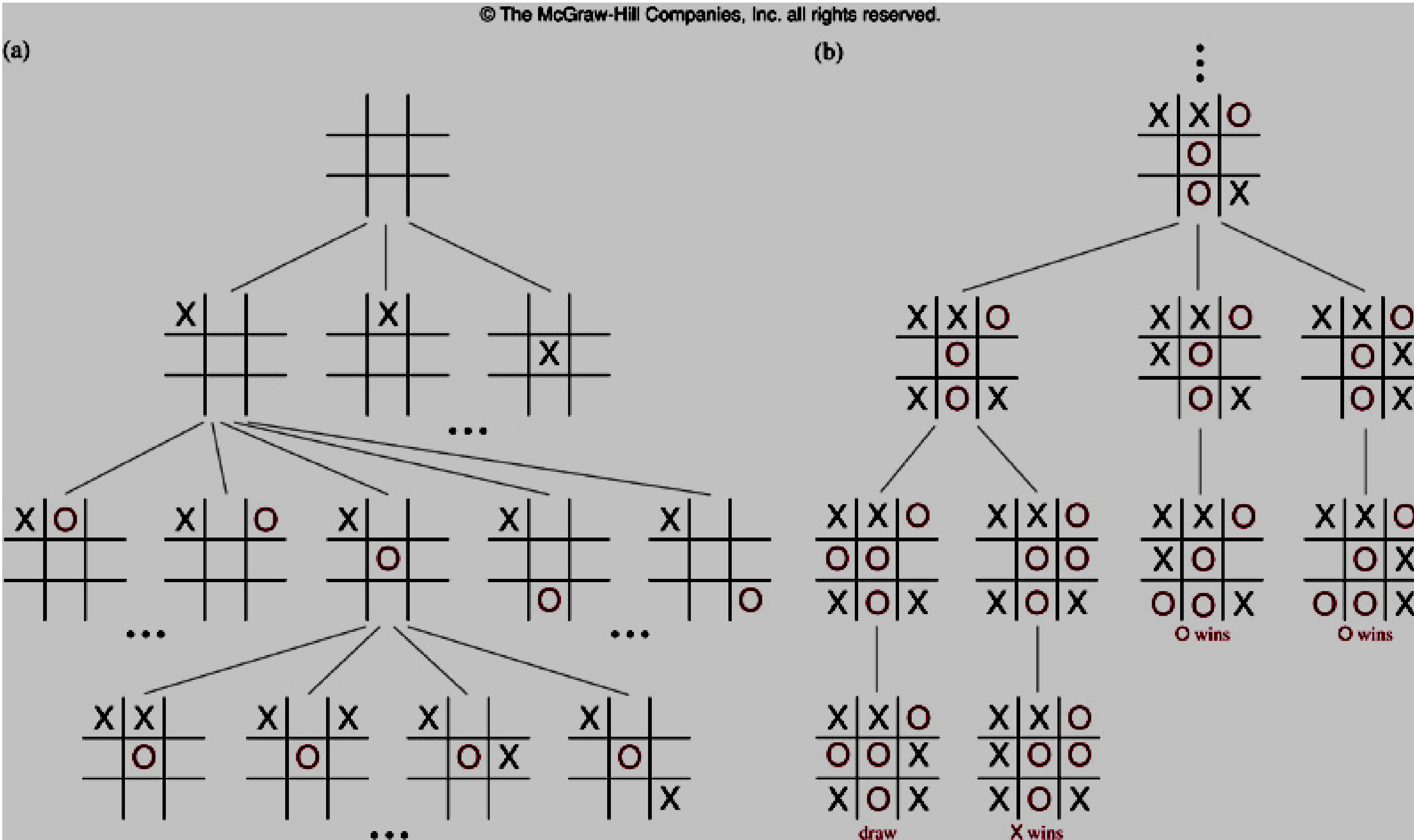
There are some piles of stones (ex: 2,2,1).

Two players will take turns picking stones from one pile.

The player picks the last stones is loser.



Game Trees : Tic-tac-toe



Game Trees…

DEFINITION 1

The *value of a vertex in a game tree* is defined recursively as:

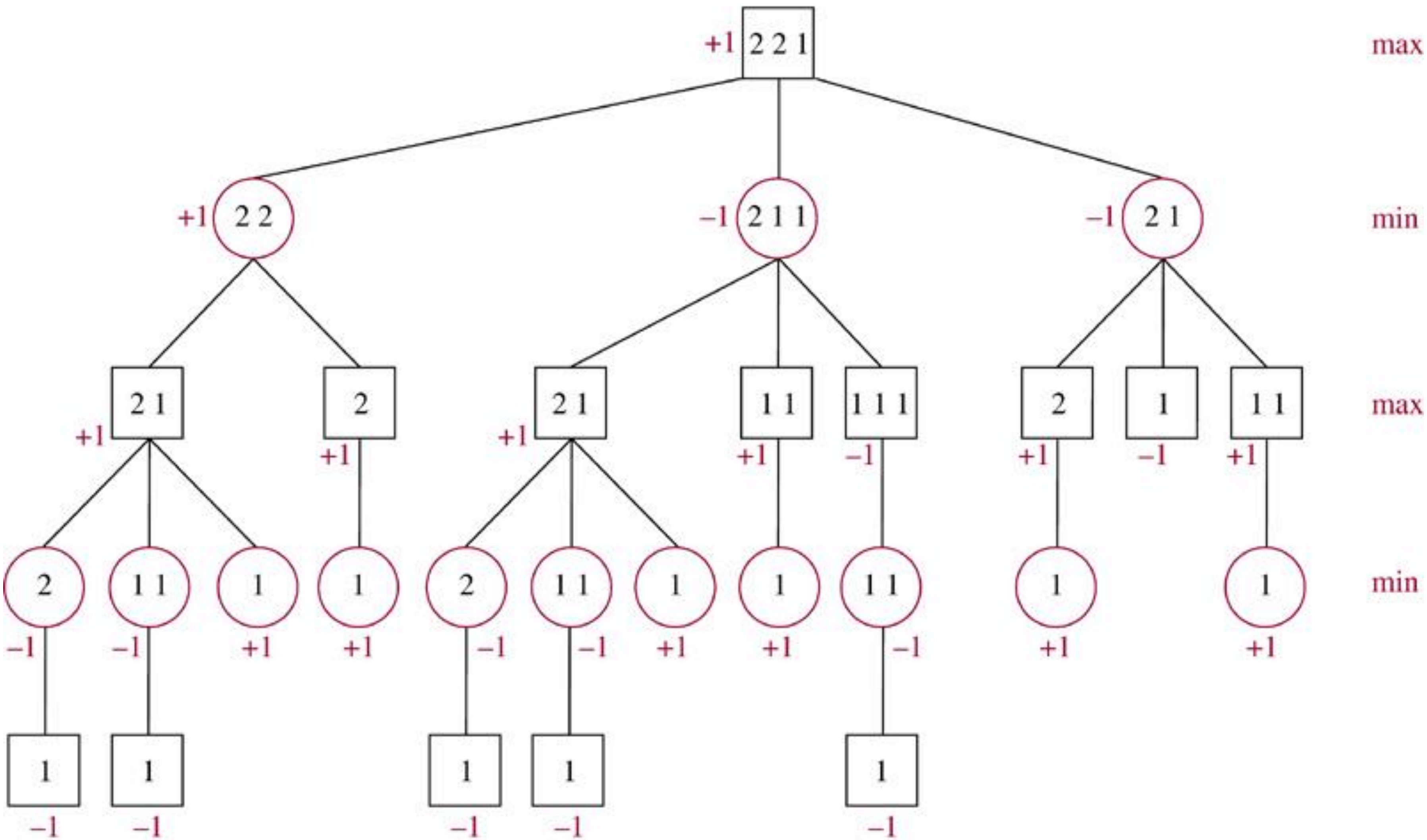
(*i*) the value of a leaf is the payoff to the first player when the game terminates in the position represented by this leaf.

(*ii*) the value of an internal vertex at an even level is the maximum of the values of its children, and the value of an internal vertex at an odd level is the minimum of the values of its children.

THEOREM 3

The value of a vertex of a game tree tells us the payoff to the first player if both players follow the minmax strategy and play starts from the position represented by this vertex.

Game Trees…


+1 2 2 1
max
+1 2 2
–1 2 1 1
–1 2 1
min
2 1 +1
2 +1
2 1 +1
1 1 +1
1 1 1 –1
2 +1
1 –1
1 1 +1
max
2 –1
1 1 –1
1 +1
1 +1
2 –1
1 1 –1
1 +1
1 +1
1 1 –1
1 +1
1 +1
min
1 –1
1 –1
1 –1
1 –1
1 –1

Traversal Algorithms

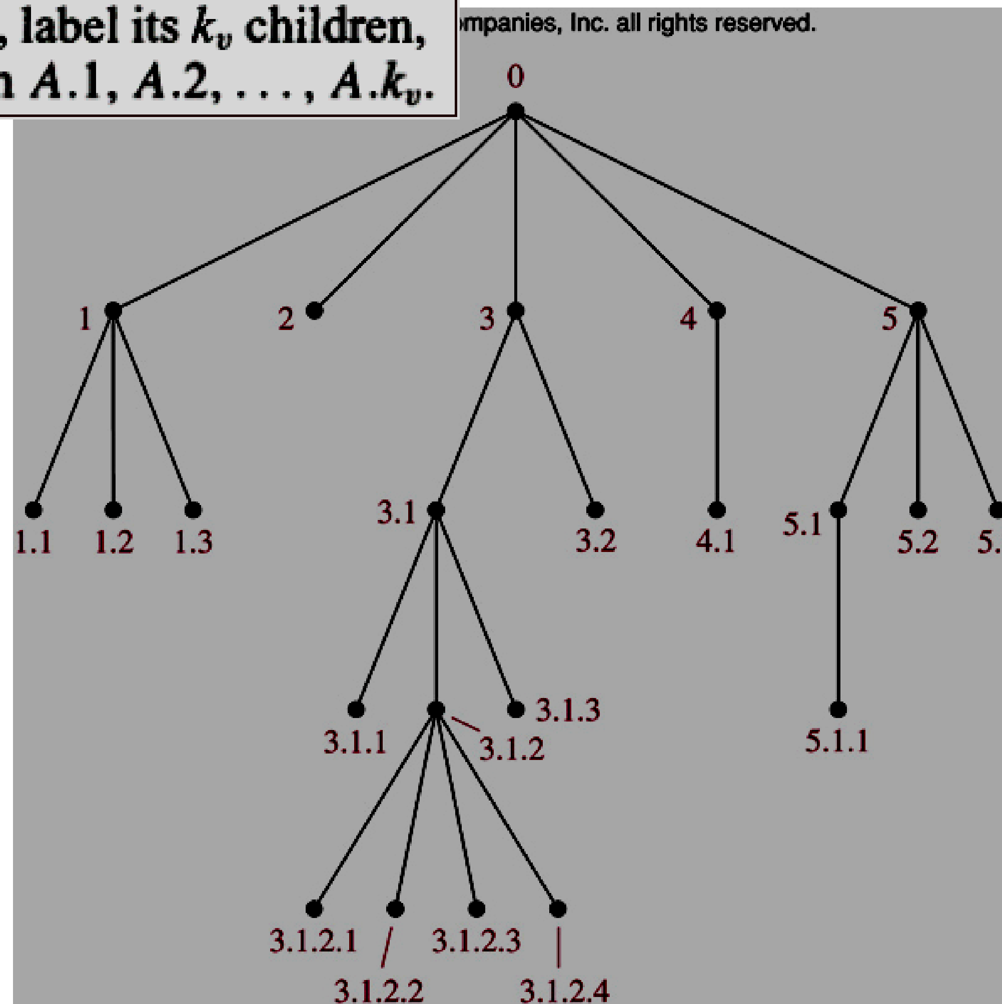
- At a time, a vertex is visited
- Operations are done:
  - Process the visited vertex, e.g. list it's information
  - Traversing recursively subtree.
- Bases on orders of tasks, traversals are classified into:
  - Preorder traversal. N L R
  - Inorder traversal. L N R
  - Postorder traversal. L R N

10.3- Tree Traversal

- Traversal a tree: A way to visit all vertices of the rooted tree.
  - Universal Address Systems
  - Traversal Algorithms
  - Infix, Prefix, and Postfix Notation

Universal Address Systems

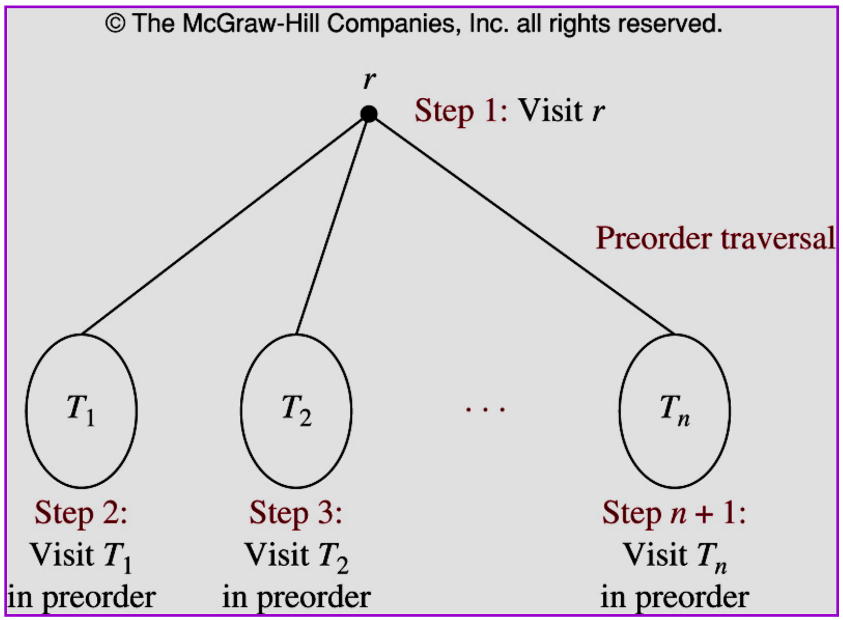
1. Label the root with the integer 0. Then label its k children (at level 1) from left to right with $1, 2, 3, \ldots, k$.
2. For each vertex v at level n with label A, label its k_v children, as they are drawn from left to right, with $A.1, A.2, \ldots, A.k_v$.

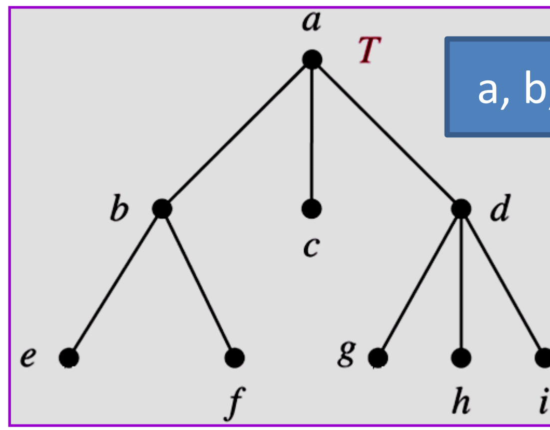


Preorder Traversal

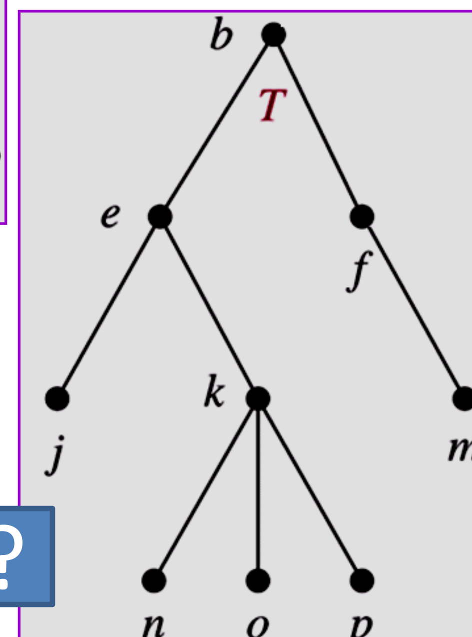
DEFINITION 1

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.





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

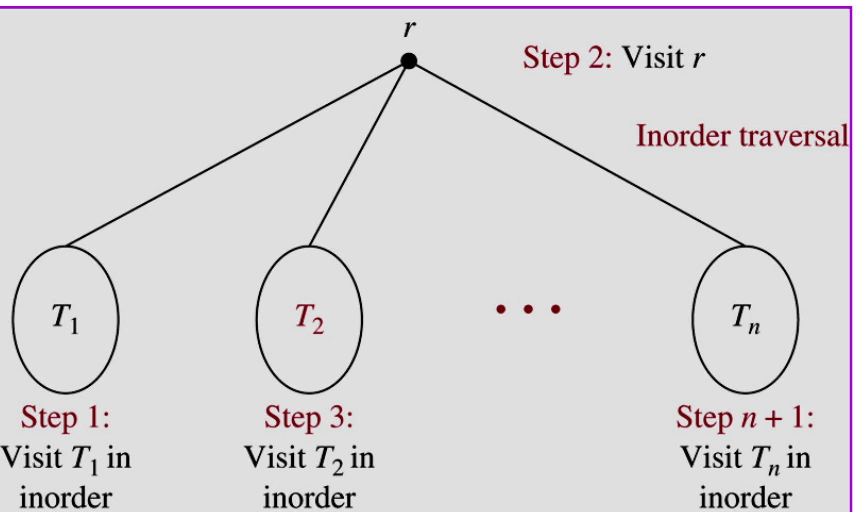


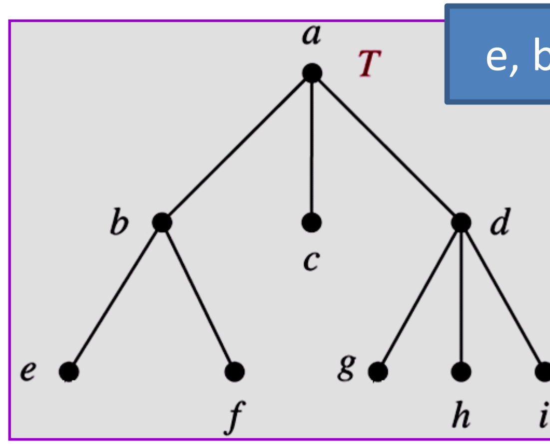
?

Inorder Traversal

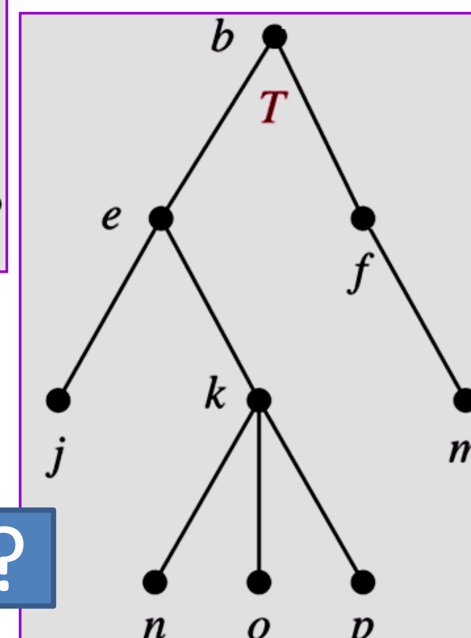
DEFINITION 2

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.





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

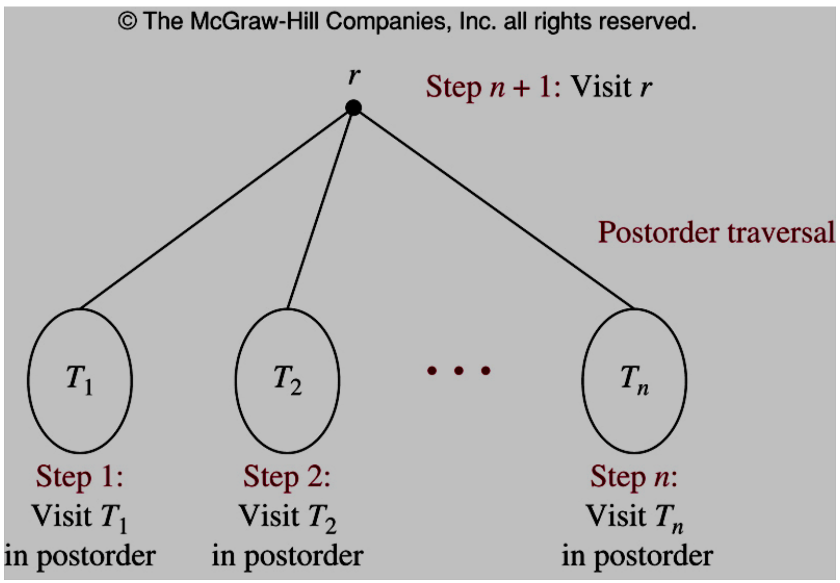


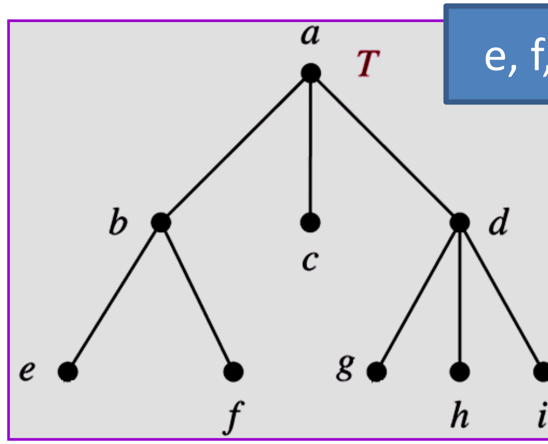
?

Postorder Traversal

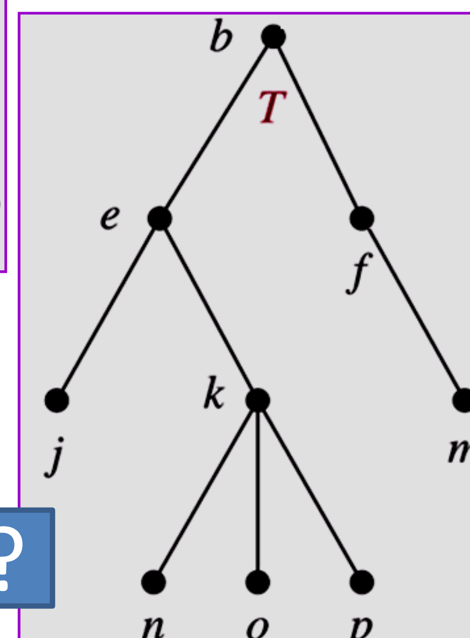
DEFINITION 3

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, $\ldots$, then T_n in postorder, and ends by visiting r.



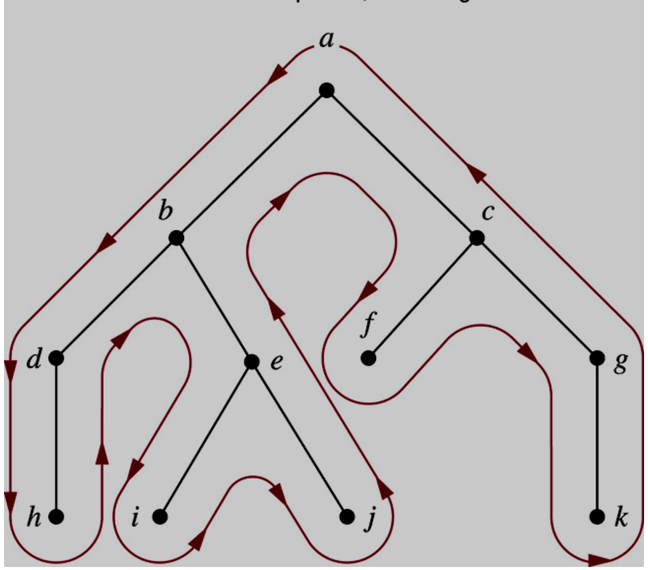


e, f, b, c, g, h, i, d, a



?

Traverse Algorithms



ALGORITHM 1 Preorder Traversal.

procedure *preorder*(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  $T(c)$:= subtree with c as its root
  preorder($T(c)$)
end

ALGORITHM 2 Inorder Traversal.

procedure *inorder*(T: ordered rooted tree)
r := root of T
if r is a leaf **then** list r
else
begin
  l := first child of r from left to right
  $T(l)$:= subtree with l as its root
  inorder($T(l)$)
  list r
  for each child c of r **except for** l from left to right
    $T(c)$:= subtree with c as its root
    inorder($T(c)$)
end

ALGORITHM 3 Postorder Traversal.

procedure *postorder*(T: ordered rooted tree)
r := root of T
for each child c of r from left to right
begin
  $T(c)$:= subtree with c as its root
  postorder($T(c)$)
end
list r

Infix, Prefix, and Postfix Notation

- Expression Trees

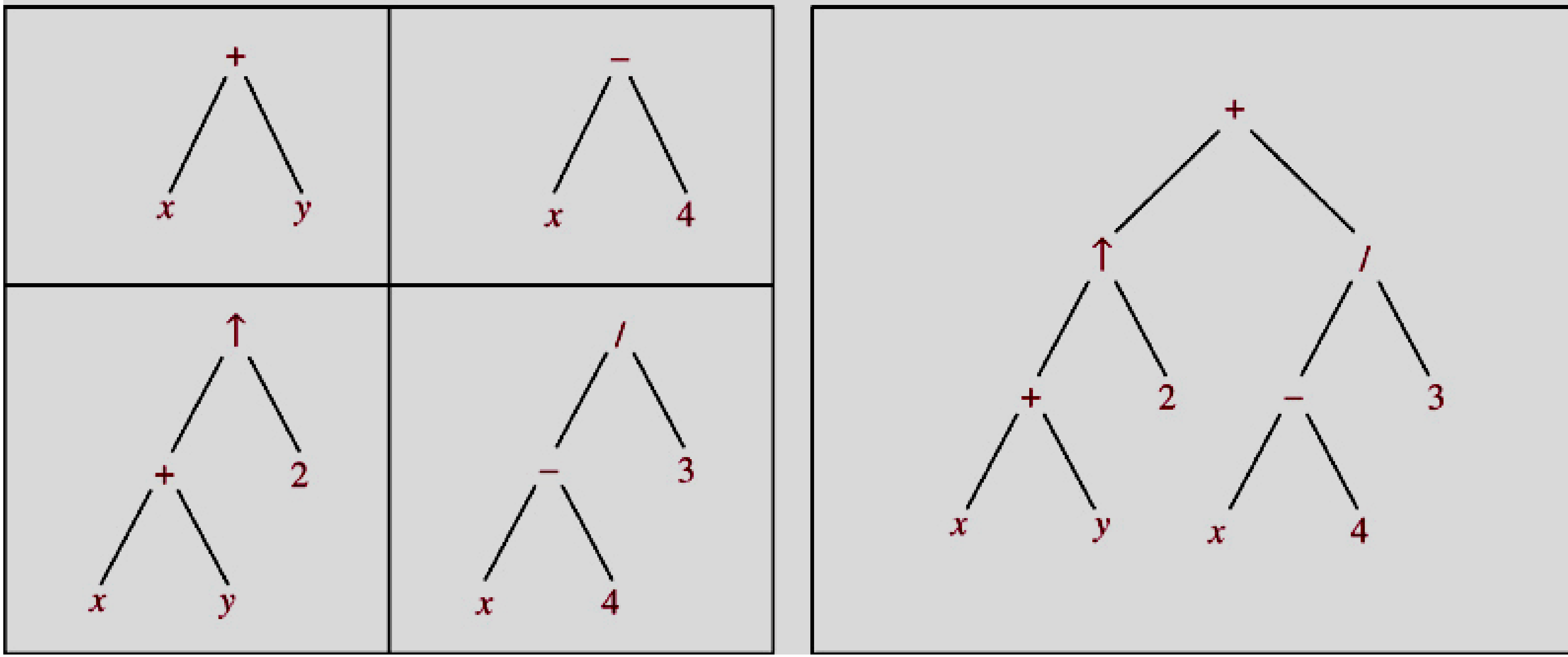


FIGURE 10 A Binary Tree Representing $((x + y) \uparrow 2) + ((x - 4)/3)$.

Infix, Prefix, and Postfix Notation

- Expression Trees

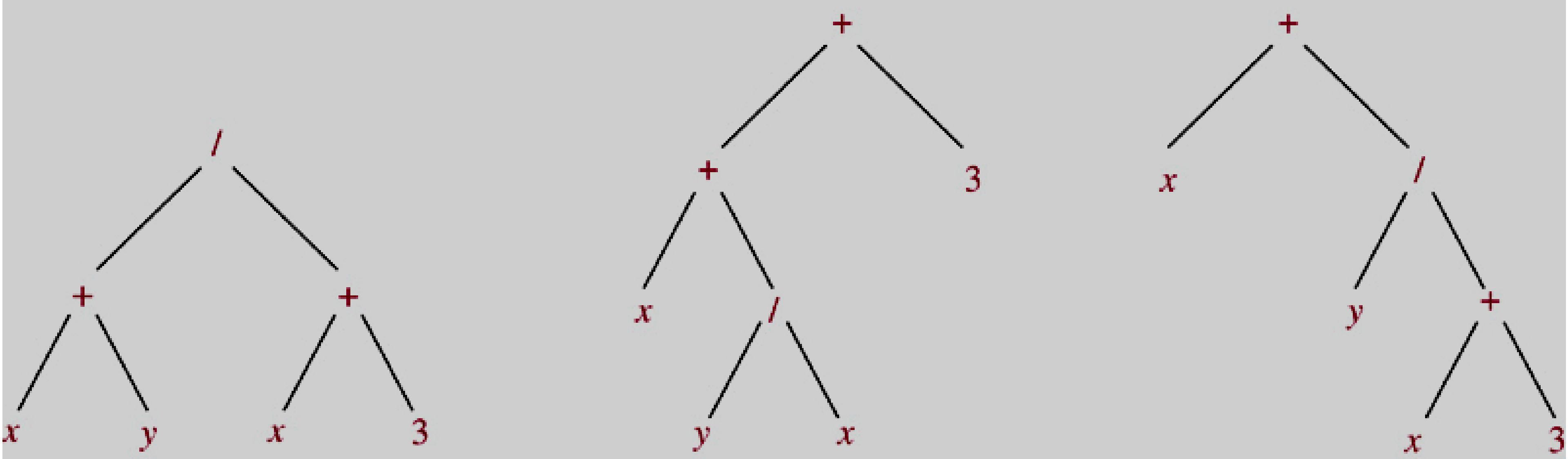


FIGURE 11 Rooted Trees Representing $(x + y)/(x + 3)$, $(x + (y/x)) + 3$, and $x + (y/(x + 3))$.

Infix, Prefix, and Postfix Notation

- **Infix form:**

  operand_1 operator operand_2 x + y

- **Prefix form:**

  operator(operand_1,operand_2) + x y

- **Postfix form:**

  (operand_1,operand_2)operator x y +

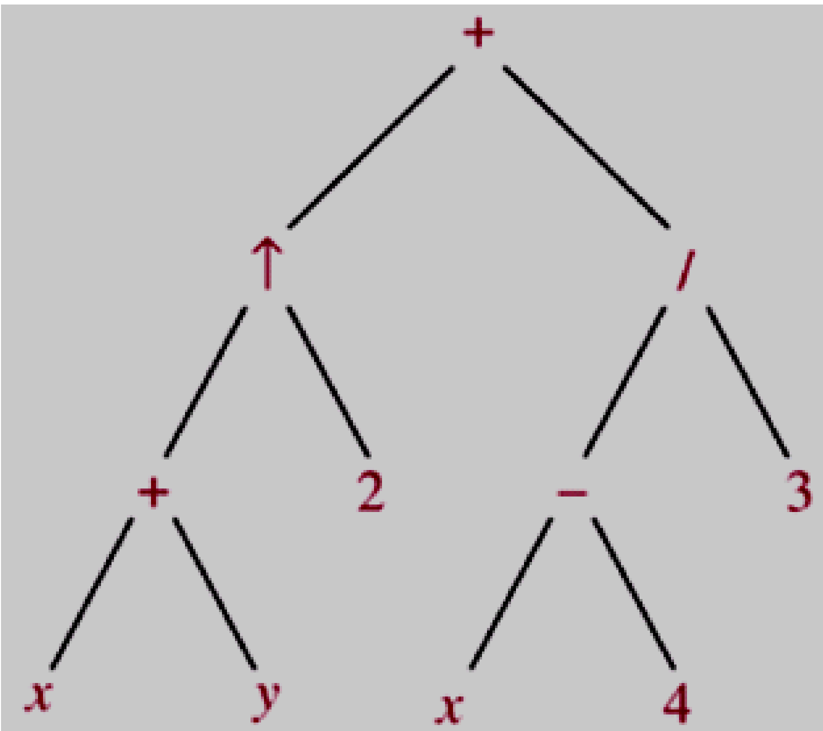
- How to find prefix and postfix form from infix form?

  (1) Draw expression tree.

  (2) Using Preorder traverse → Prefix form

  Using Postorder traverse → Postfix form

Infix, Prefix, and Postfix Notation



Infix form

$$((x + y) \uparrow 2) + ((x - 4)/3)$$

Prefix form

$$+ \uparrow + \, x \; y \; 2 \; / - x \; 4 \; 3$$

Postfix form

$$x \; y + 2 \uparrow x \; 4 - 3 \; / \; +$$

Infix, Prefix, and Postfix Notation

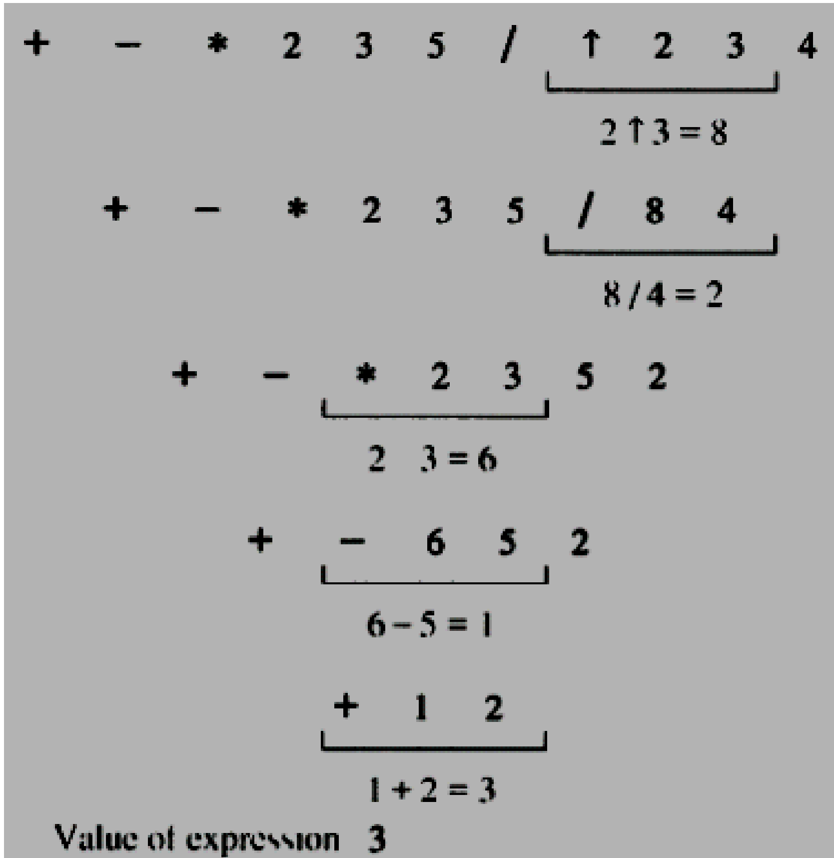


FIGURE 12 Evaluating a Prefix Expression.

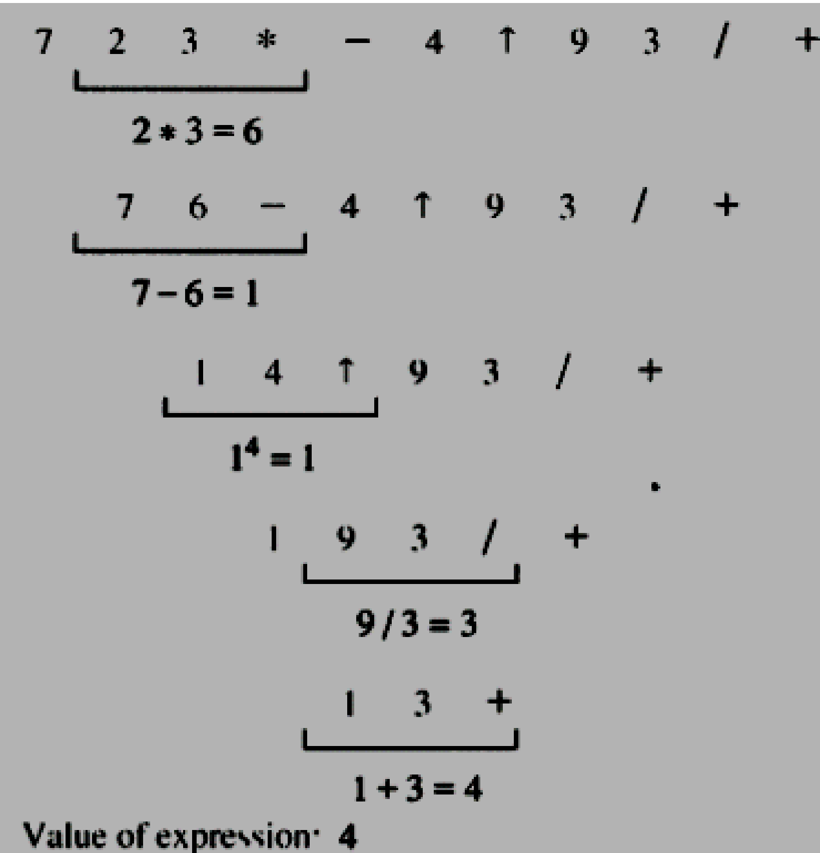


FIGURE 13 Evaluating a Postfix Expression.

Summary

- 10.1- Introduction to Trees
- 10.2- Applications of Trees
- 10.3- Tree Traversal

- **Thanks**