Chapter 10
Trees

MSc.Nguyen Quoc Thanh

Objectives

10.1- Introduction to Trees

10.2- Applications of Trees

10.3- Tree Traversal

10.4- Spanning Trees

10.5-Minimum Spanning Trees

10.1- Introduction to Trees

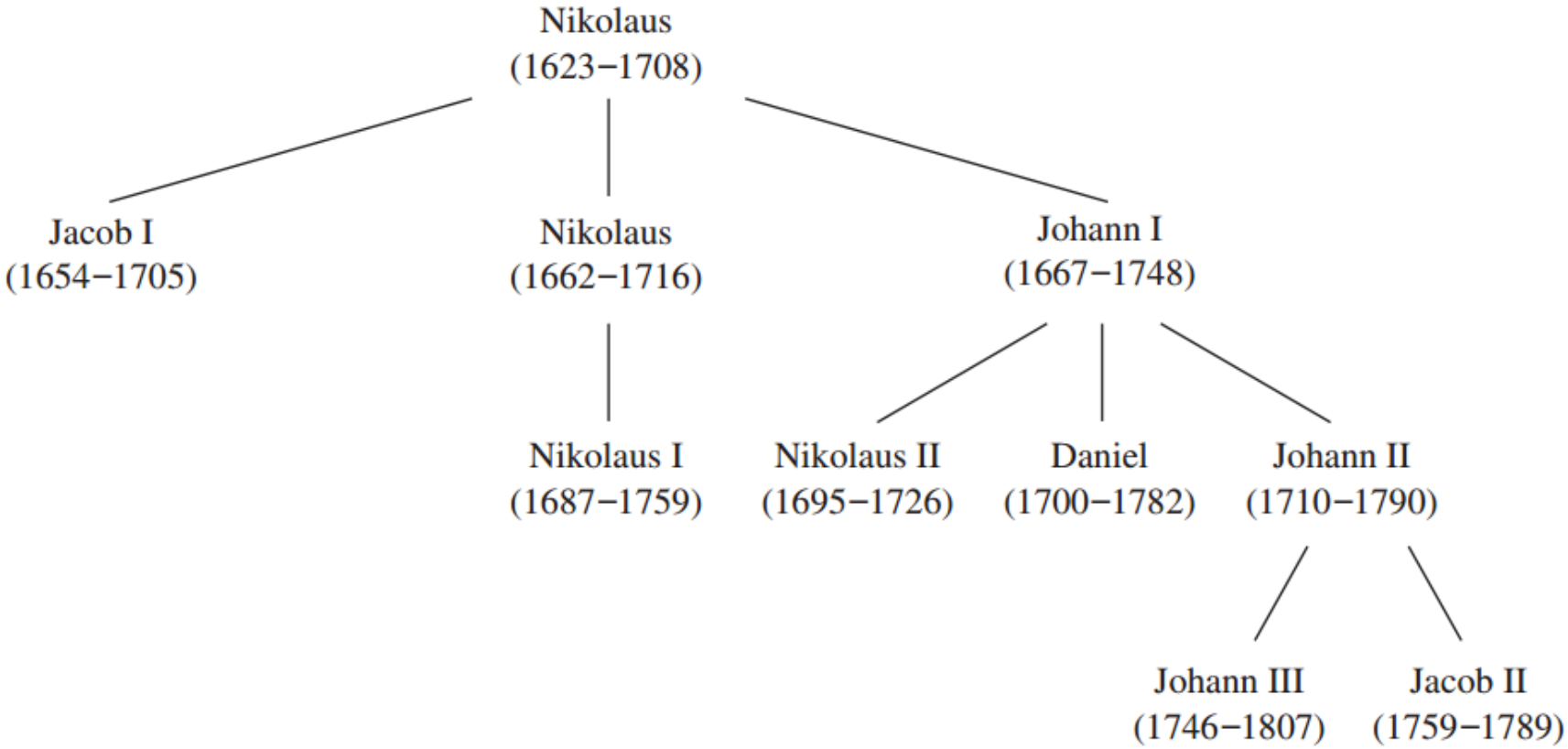


FIGURE 1 **The Bernoulli Family of Mathematicians.**

Introduction to Trees…

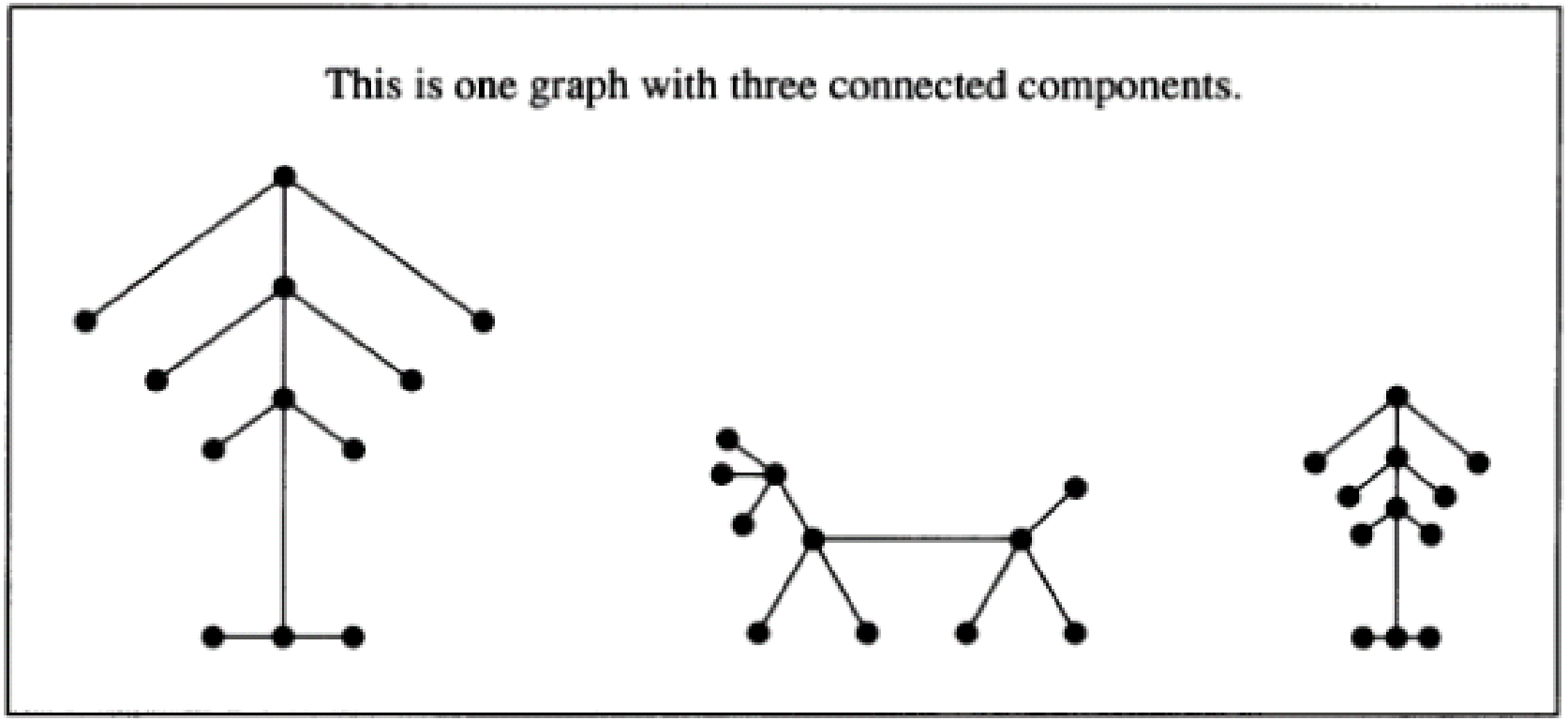


FIGURE 3 Example of a Forest.

Introduction to Trees

DEFINITION 1

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

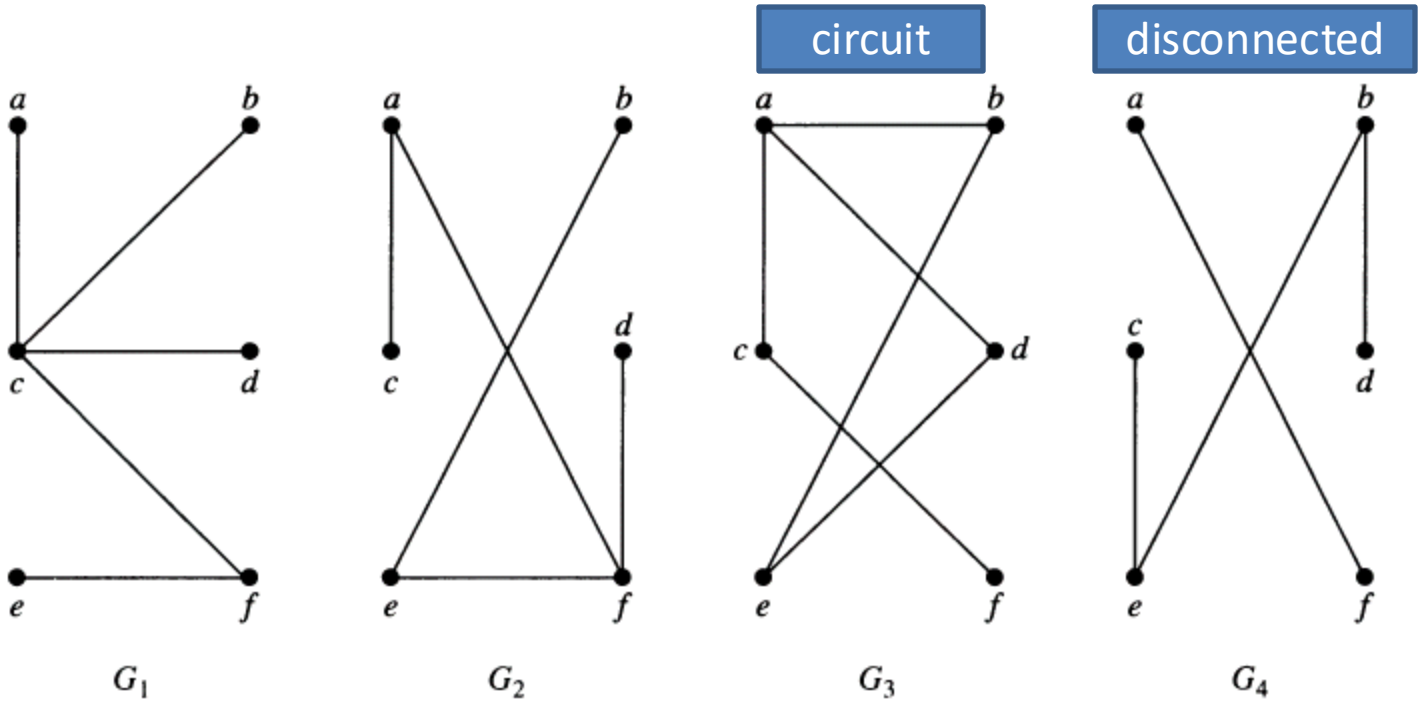


FIGURE 2 Examples of Trees and Graphs That Are Not Trees.

Introduction to Trees…

THEOREM 1

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.

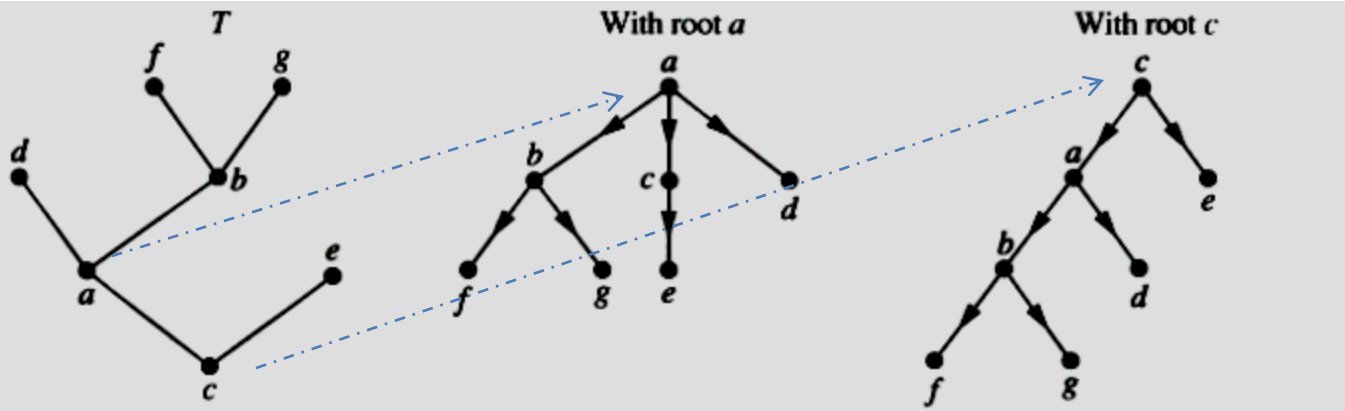


FIGURE 4 A Tree and Rooted Trees Formed by Designating Two Roots.

Introduction to Trees…

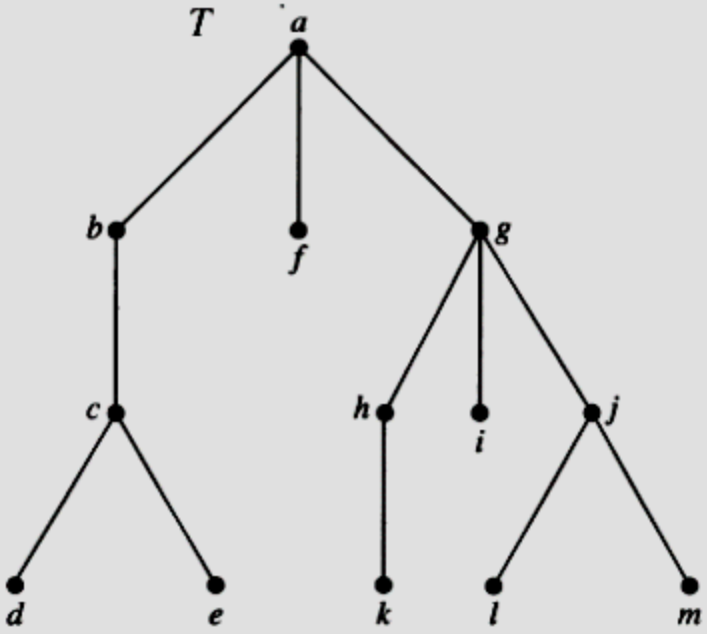


FIGURE 5 A Rooted Tree T.

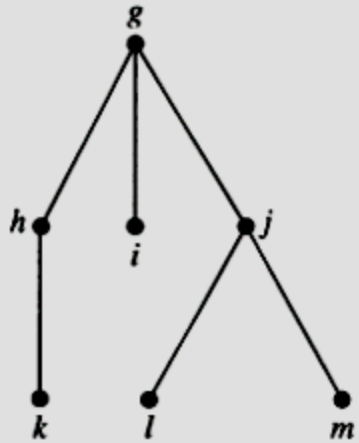


FIGURE 6 The Subtree Rooted at g.

Some terminologies:

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

Introduction to Trees…

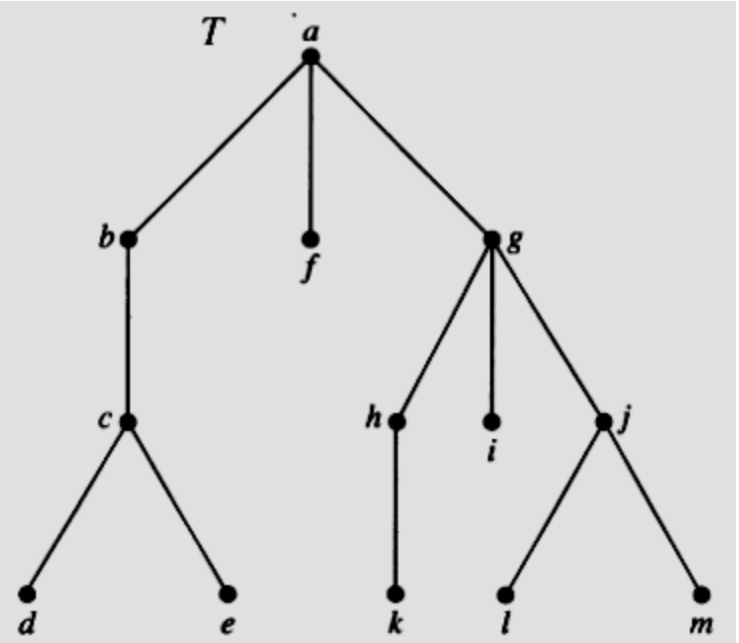


FIGURE 5 A Rooted Tree *T*.

EXAMPLE 2: In the rooted tree T (with root a) shown in Figure 5,

find the parent of c,

the children of g,

the siblings of h,

all ancestors of e,

all descendants of b,

all internal vertices,

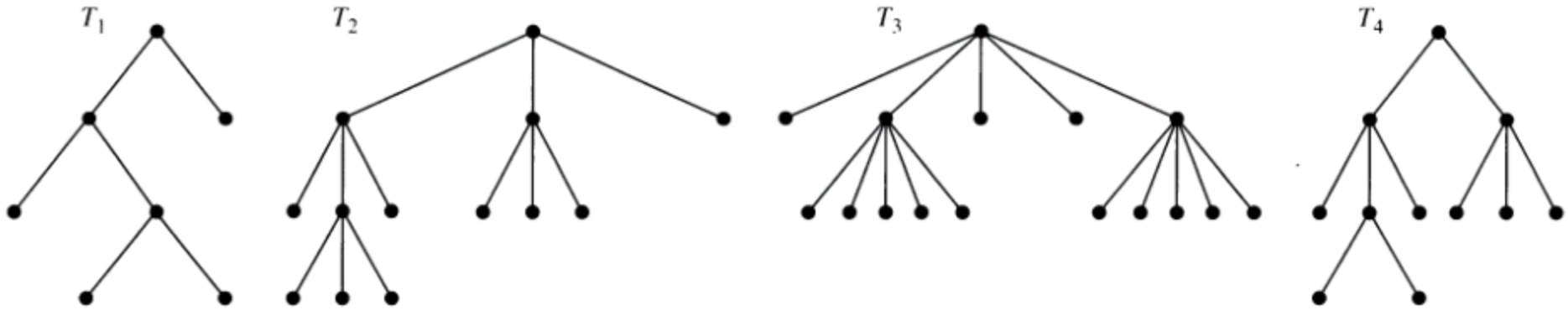
and all leaves.

What is the subtree rooted at g?

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…

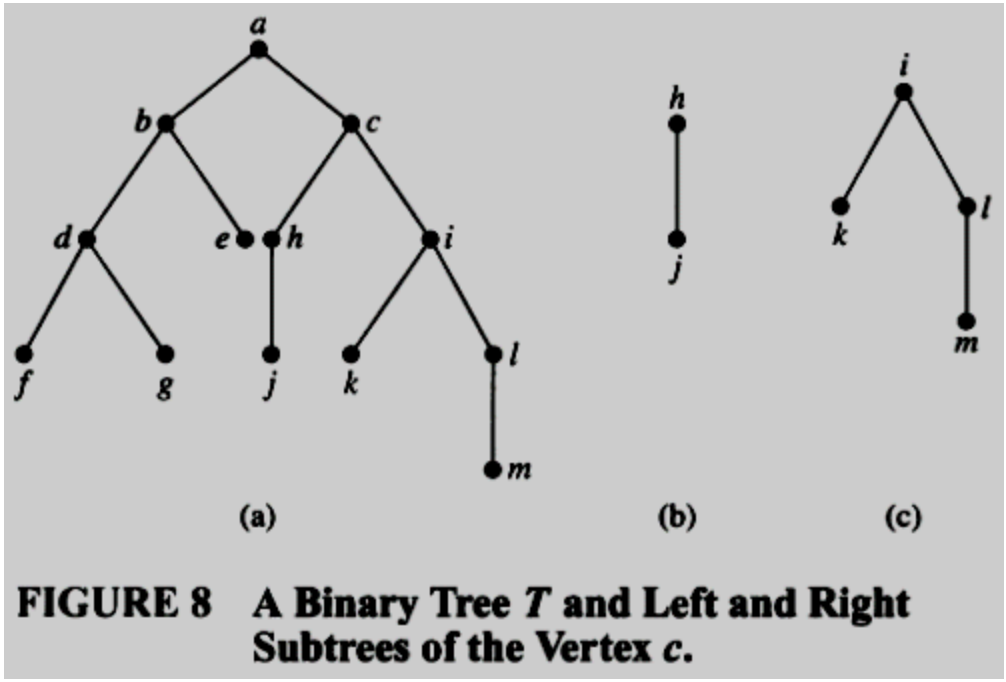
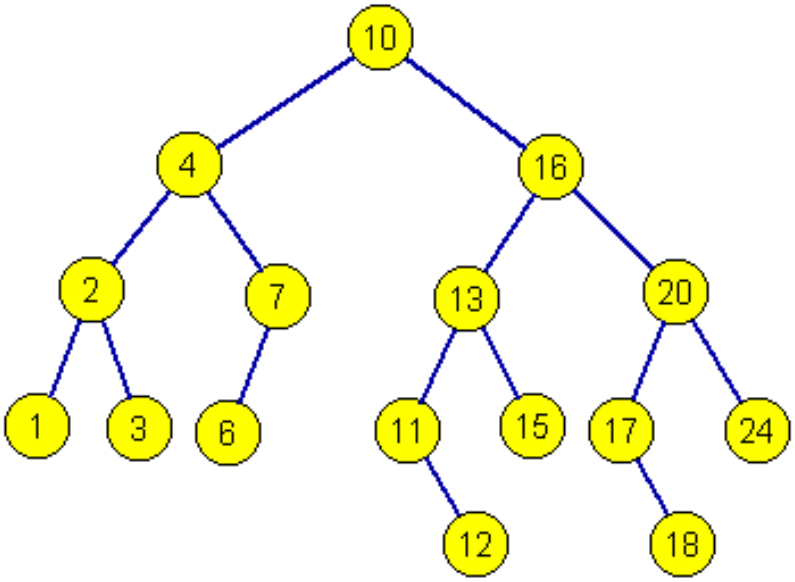
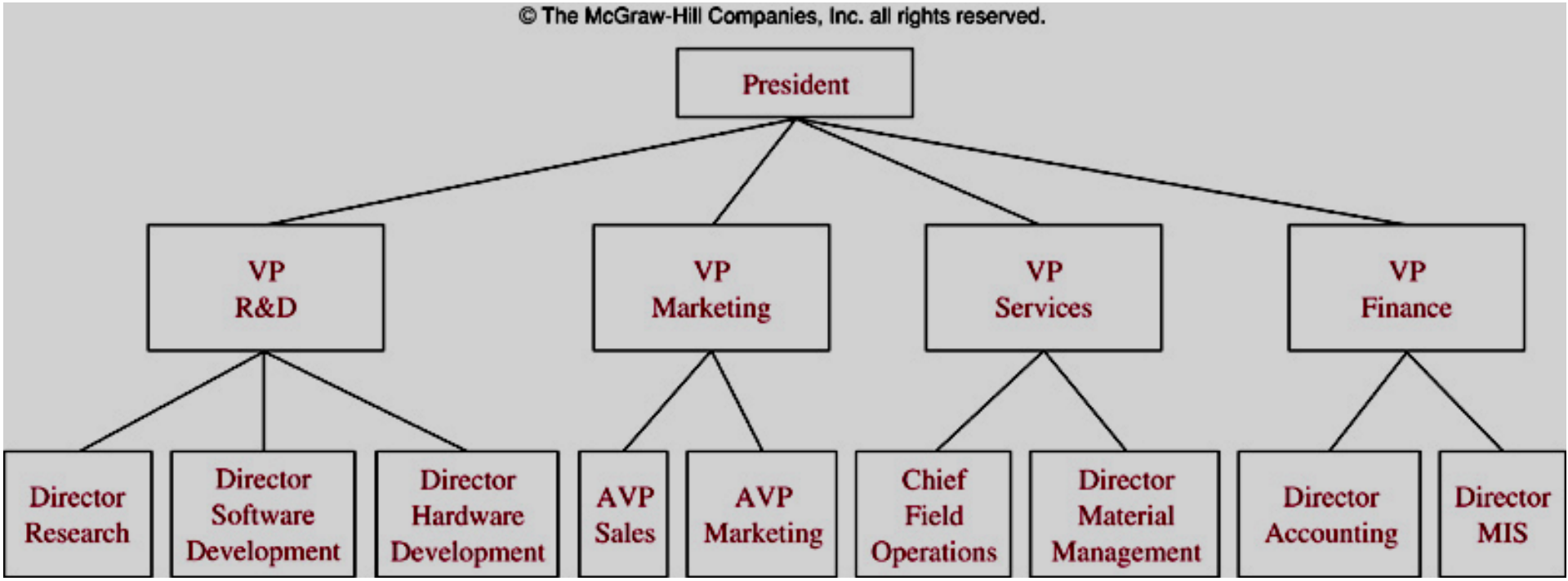


FIGURE 8 A Binary Tree T and Left and Right Subtrees of the Vertex c.



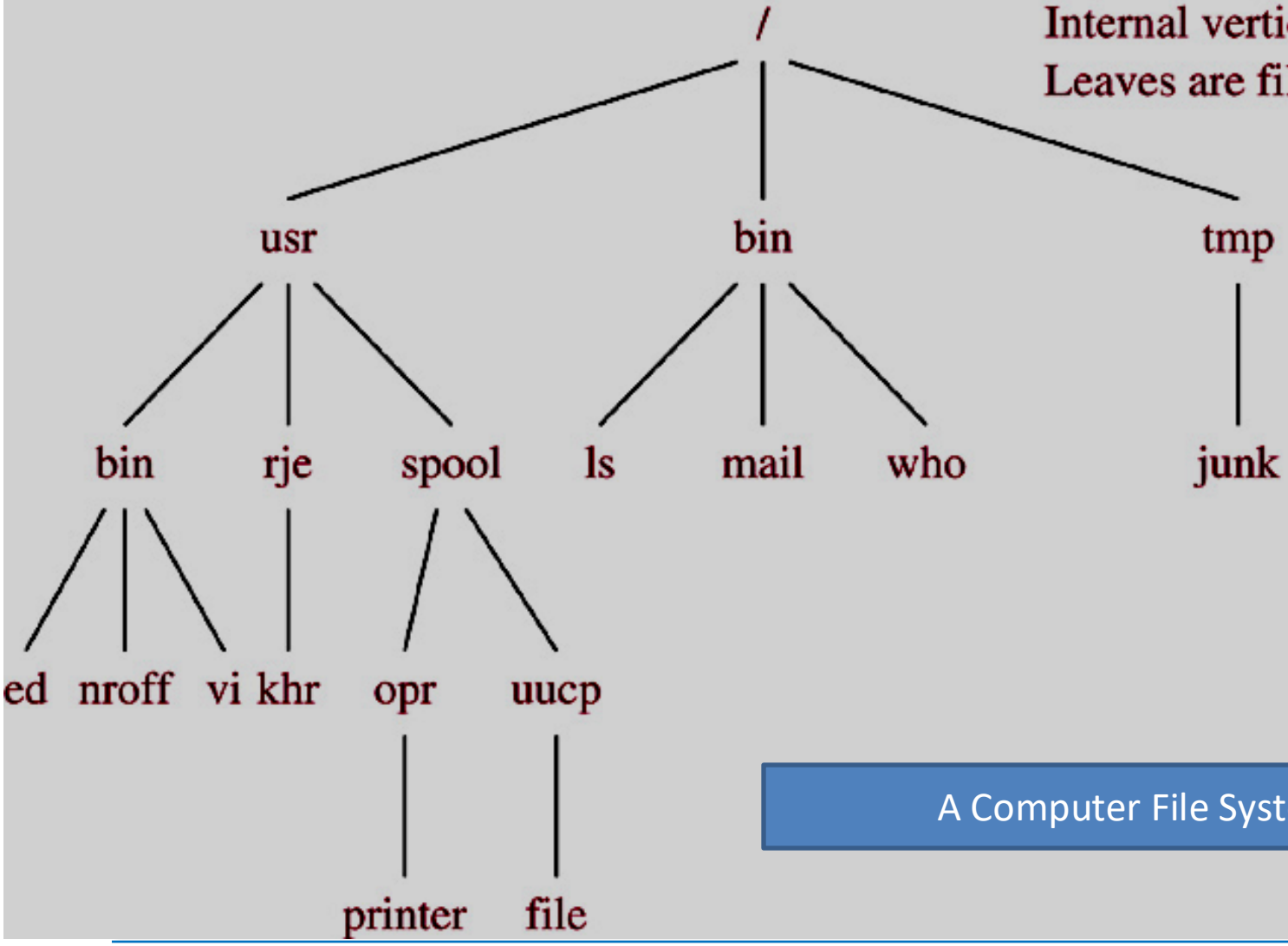
Ordered rooted tree

Some Tree Models



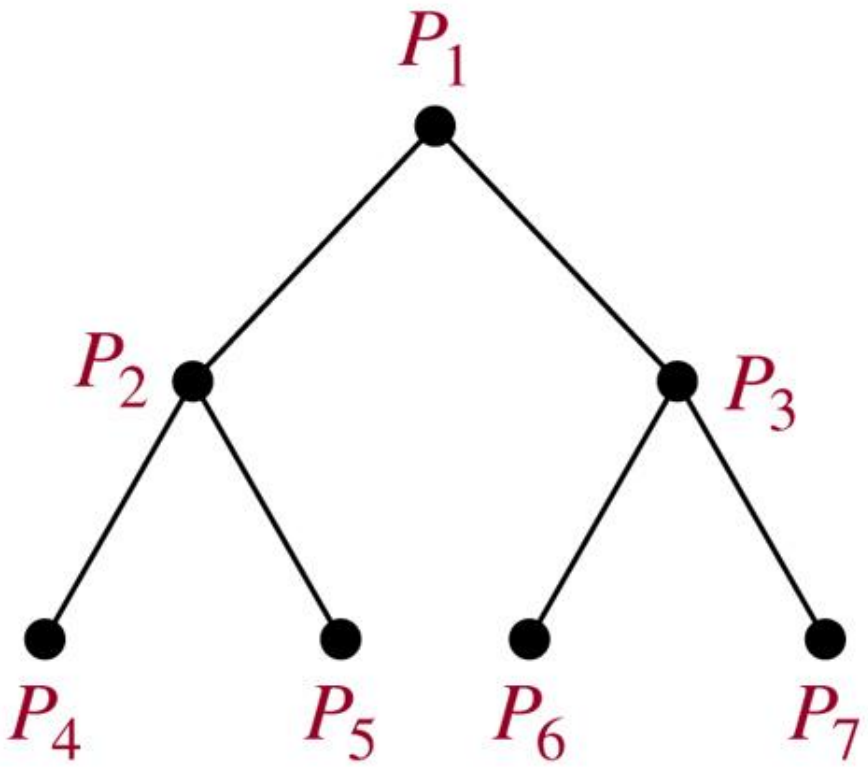
A Organizational Tree

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 .

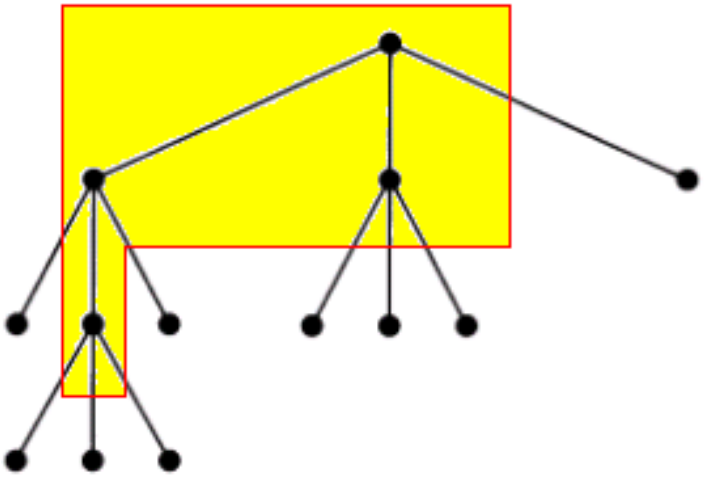
→ P(k+1) true

Proved.

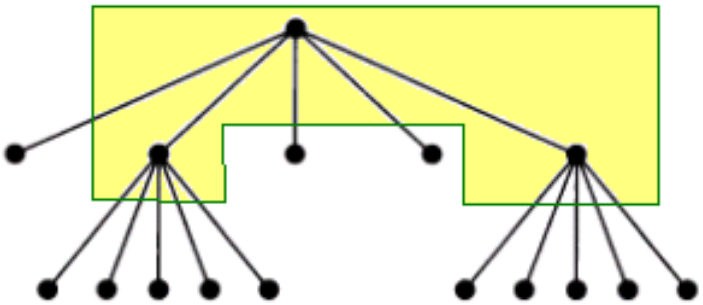
Introduction to Trees…

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…

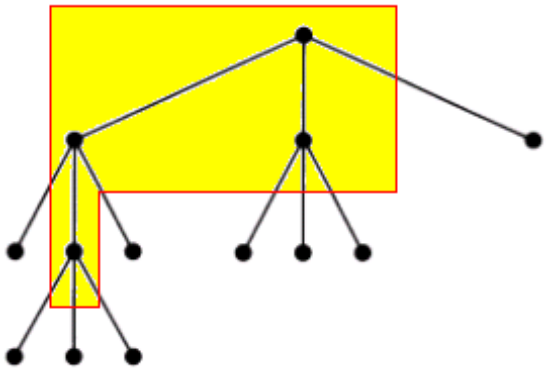
THEOREM 4

A full m-ary tree with

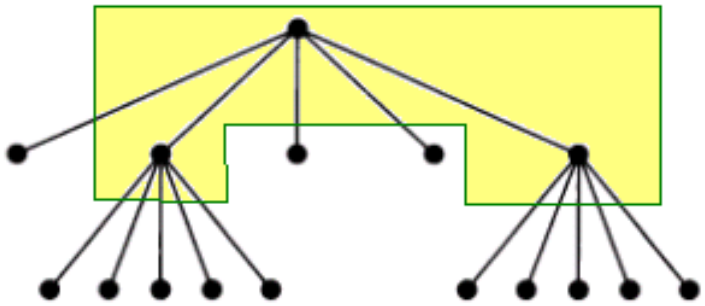
(*i*) n vertices has $i = (n-1)/m$ internal vertices and $l = [(m-1)n+1]/m$ leaves,

(*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.



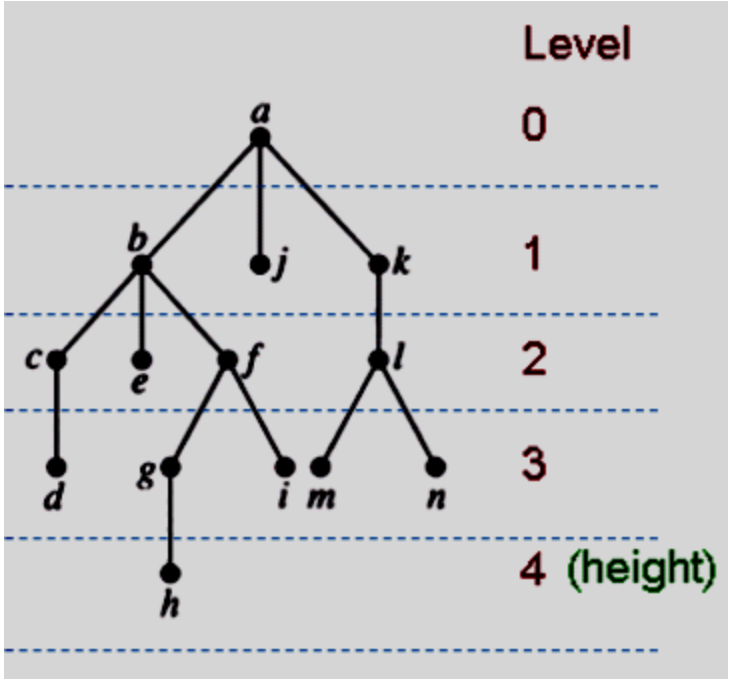
m=3
n=13
i=12/3= 4
l= [(3-1)13+1]/3=9



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

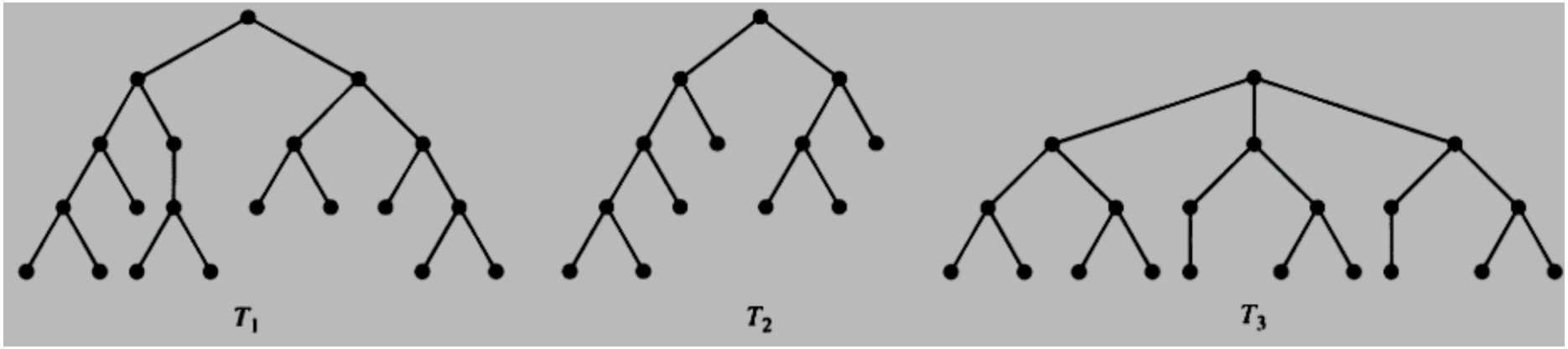
Introduction to Trees…

- 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)

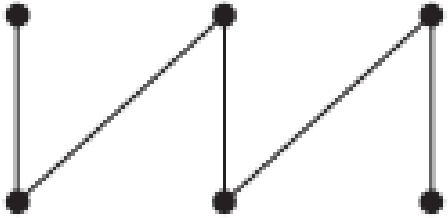
There are at most m^h leaves in an m-ary tree of height **h**.

COROLLARY 1 Proof: page 693

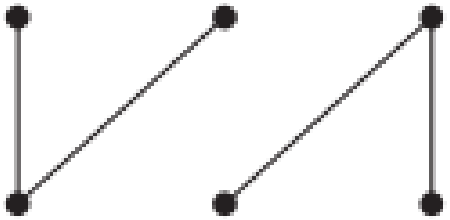
If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$. (We are using the ceiling function here. Recall that $\lceil x \rceil$ is the smallest integer greater than or equal to x.)

1. Which of these graphs are trees?

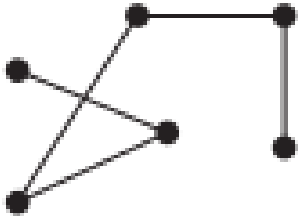
a)



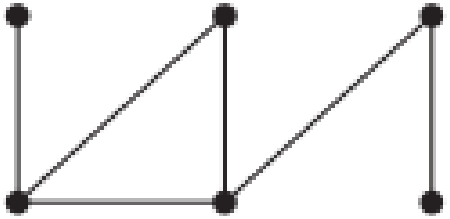
b)



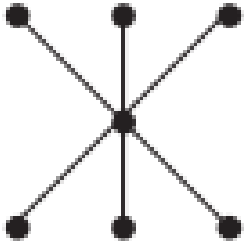
c)



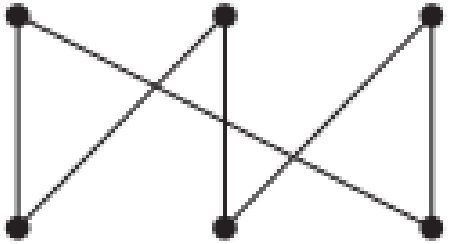
d)



e)

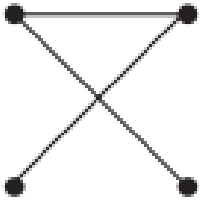


f)

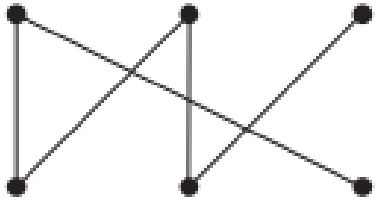


2. Which of these graphs are trees?

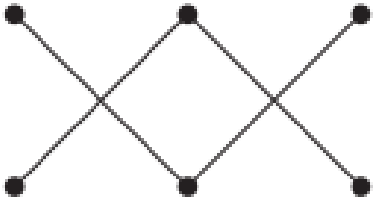
a)



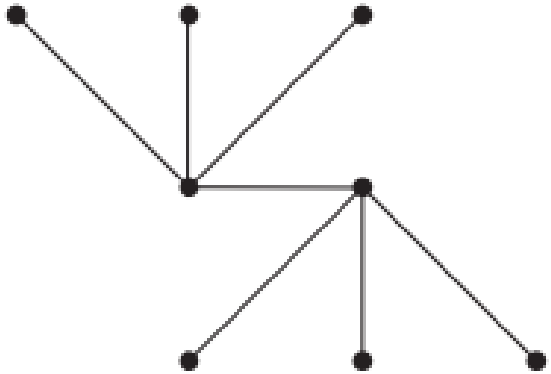
b)



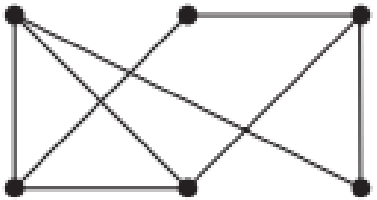
c)



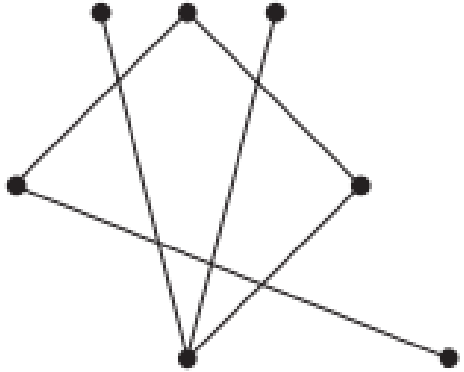
d)



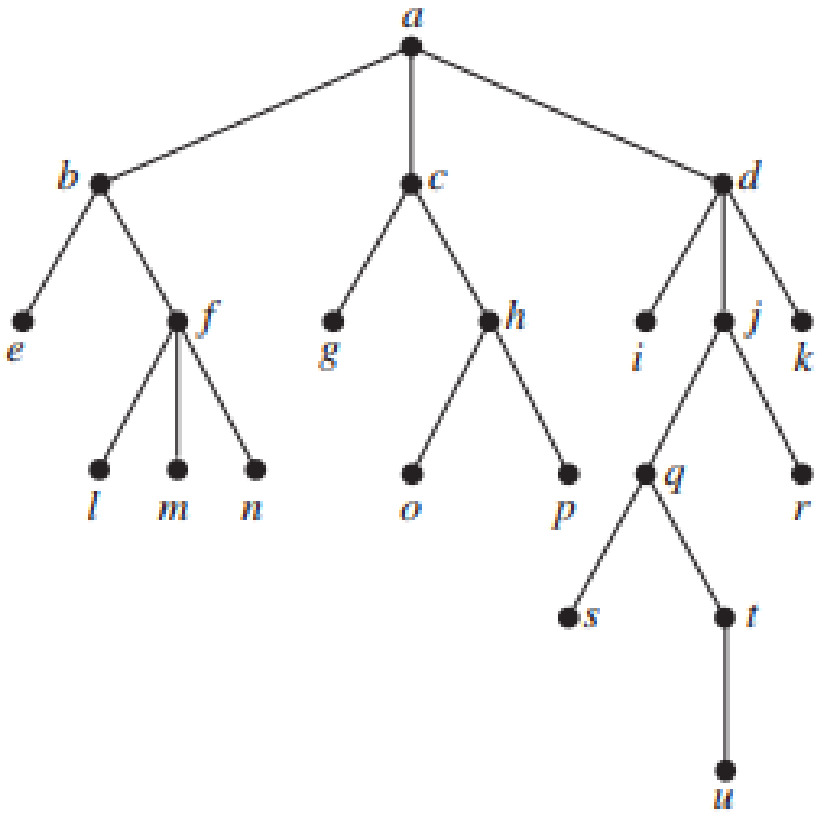
e)



f)



3. Answer these questions about the rooted tree illustrated.



a) Which vertex is the root?
b) Which vertices are internal?
c) Which vertices are leaves?
d) Which vertices are children of *j*?
e) Which vertex is the parent of *h*?
f) Which vertices are siblings of *o*?
g) Which vertices are ancestors of *m*?
h) Which vertices are descendants of *b*?

16. Which complete bipartite graphs $K_{m,n}$, where m and n are positive integers, are trees?

17. How many edges does a tree with 10,000 vertices have?

18. How many vertices does a full 5-ary tree with 100 internal vertices have?

19. How many edges does a full binary tree with 1000 internal vertices have?

20. How many leaves does a full 3-ary tree with 100 vertices have?

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.

10.2- Applications of Trees

- Binary Search Trees
- Decision Trees
- Prefix Codes

10.2- Applications of Trees

- Binary Search Trees
- Decision Trees
- Prefix Codes

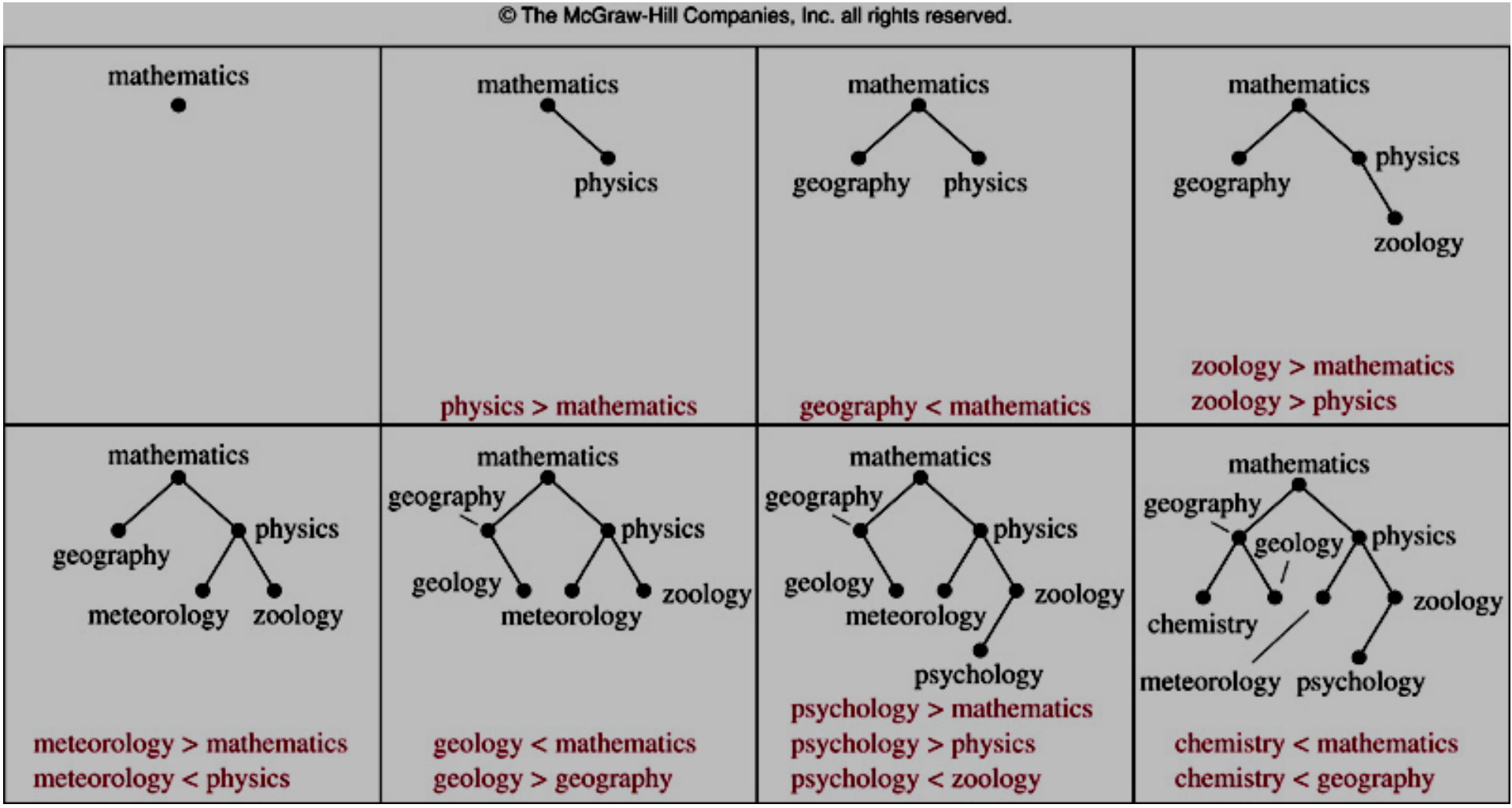
Constructing a Binary Search Tree

a b c d e f g h i j k l m n o p q r s t u v w x y z

EXAMPLE 1: Form a binary search tree for the words *mathematics, physics, geography, zoology, meteorology, geology, psychology*, and *chemistry* (using alphabetical order)

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

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
Bài toán đồng tiền giả

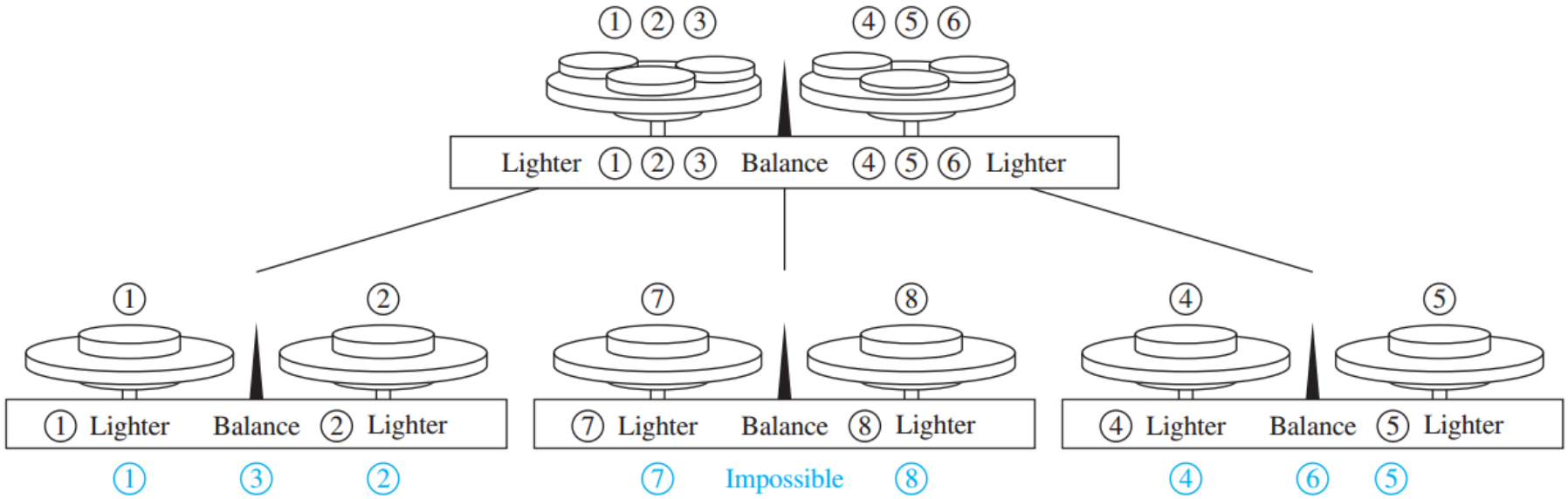


FIGURE 3 **A Decision Tree for Locating a Counterfeit Coin. The counterfeit coin is shown in color below each final weighing.**

Decision Trees:

Sorting based on Binary Comparisons

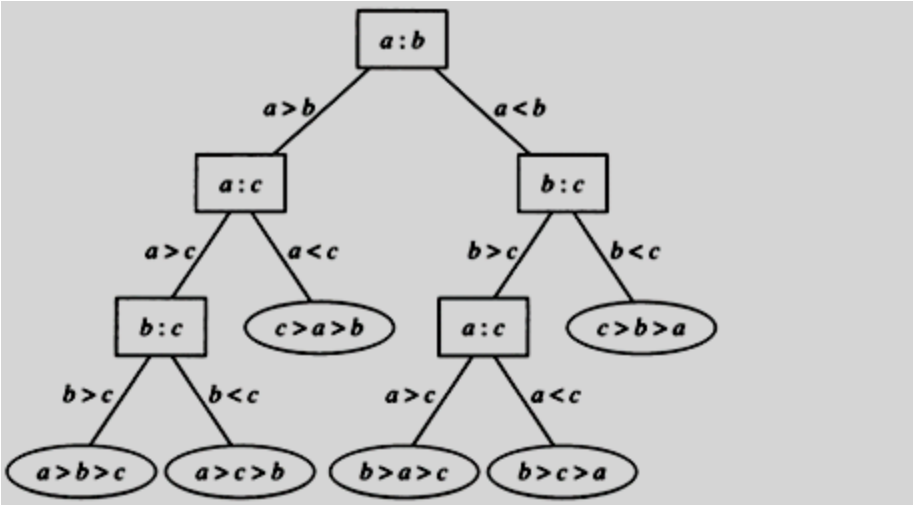


FIGURE 4 A Decision Tree for Sorting Three Distinct Elements.

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 $\Omega(n \log n)$.

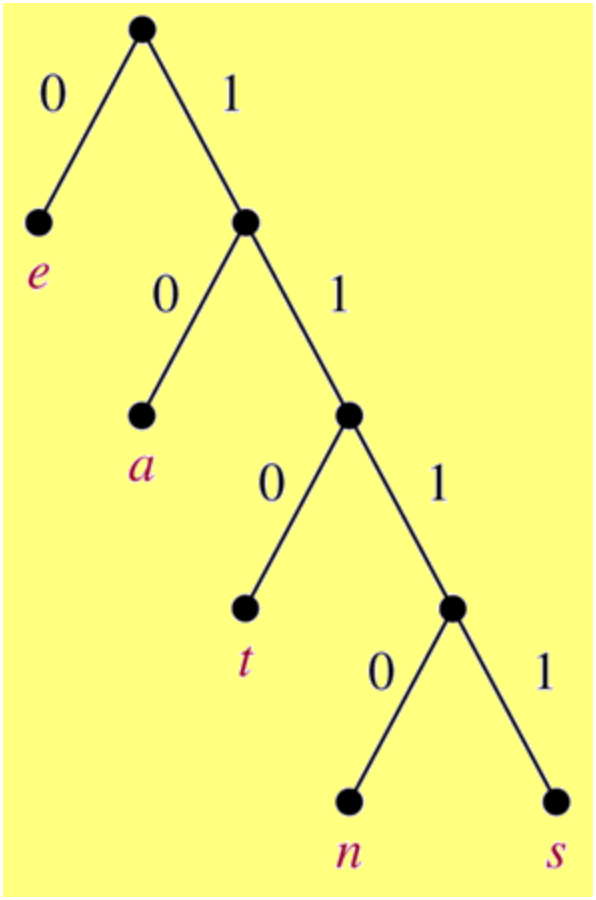
Prefix Codes

- 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

- Consider the codes:

(i) a: 10 e: 01 t: 0110 r: 1

➔ Not a prefix code

Decode the message 01101:

- 01 10 1 may be ear
- 0110 1 may be tr

(ii) a: 10 e: 01 t: 001 r: 11

➔ Prefix code

Decode the message 0010110

001 01 10 means tea only

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.

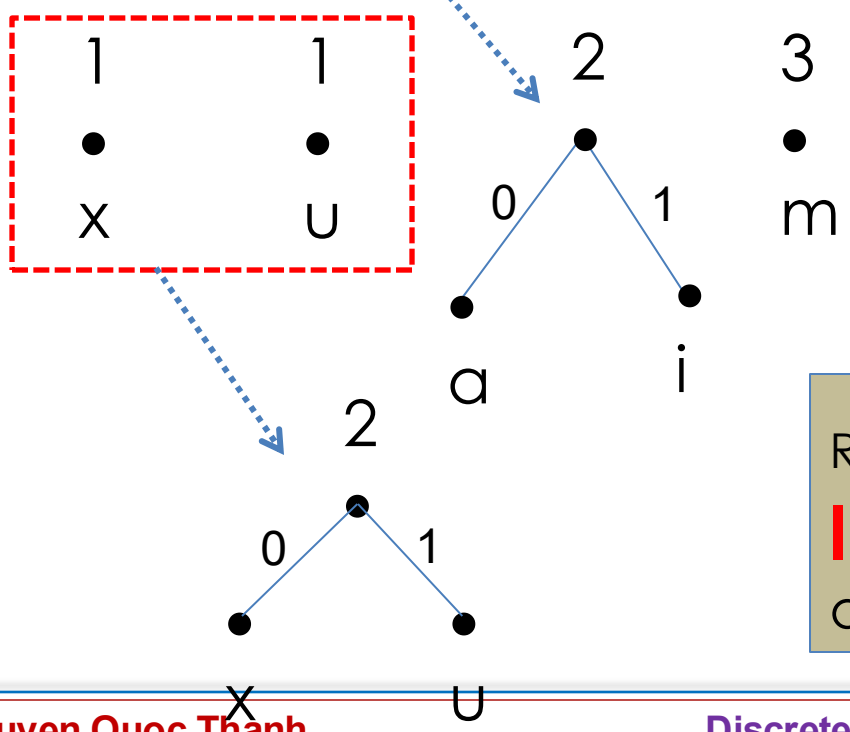
➔ Coding source text (Mã hóa văn bản nguồn)

Ex1: Use Huffman coding algorithm for encoding the word *"maximum"*

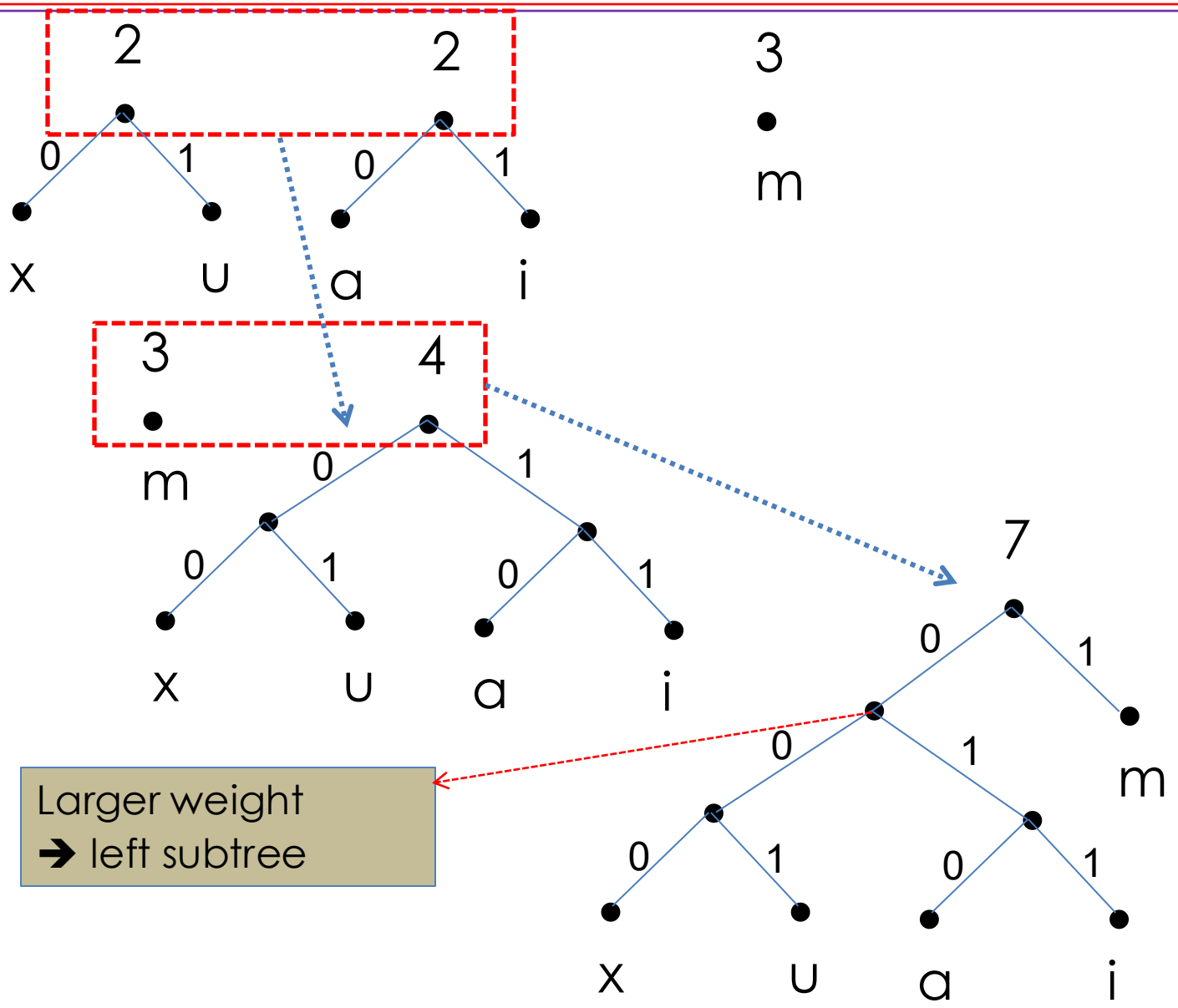
- Counting:

1	1	1	1	3
a	i	x	u	m

- Construct a binary tree



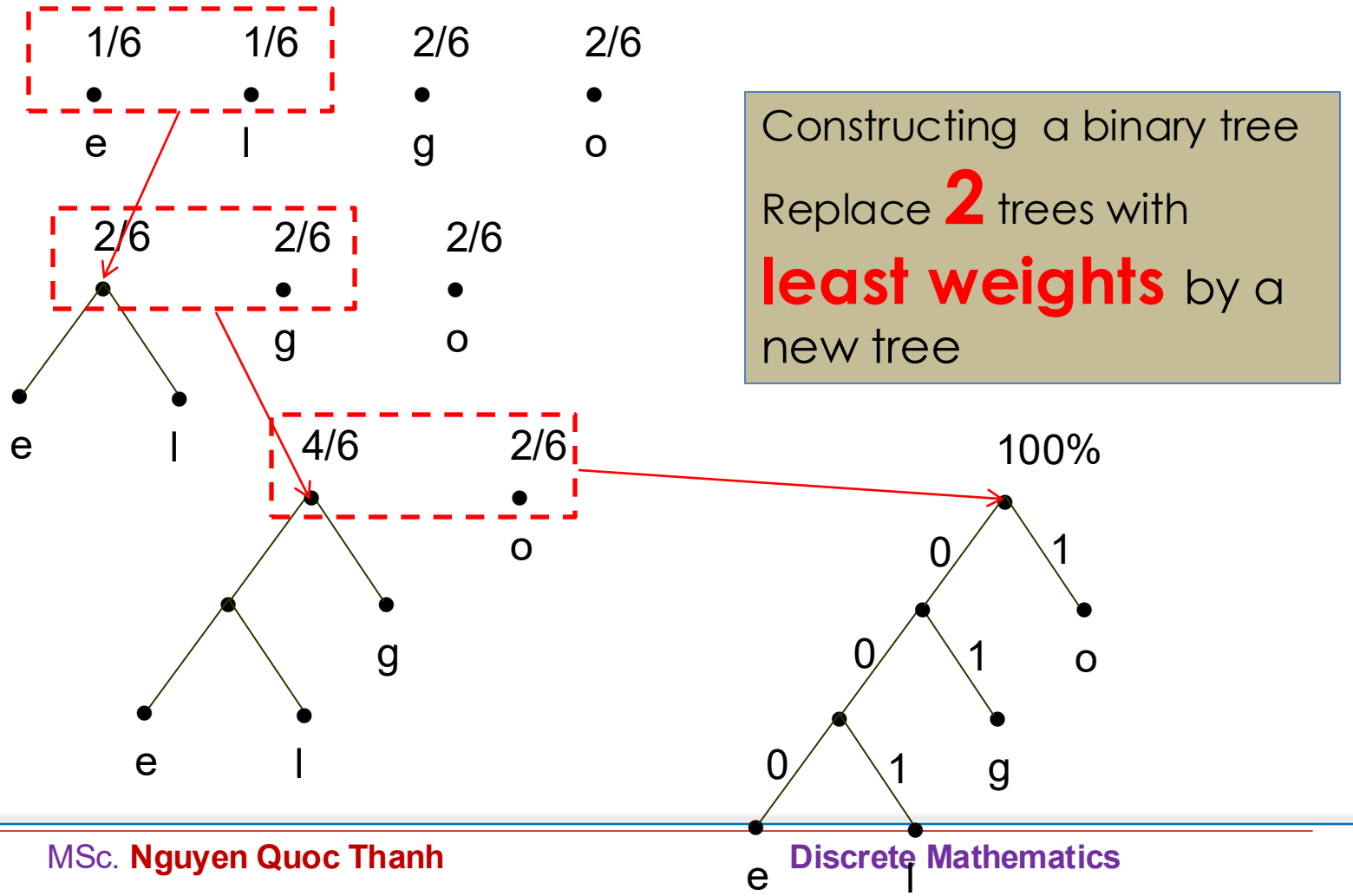
Replace **2** trees with **least weights** by a new tree



Larger weight
➔ left subtree

Ex2: Use Huffman coding algorithm for encoding the word "google"

- Counting frequencies of letters:



Constructing a binary tree

Replace **2** trees with **least weights** by a new tree

Prefix Codes: Huffman Coding Algorithm

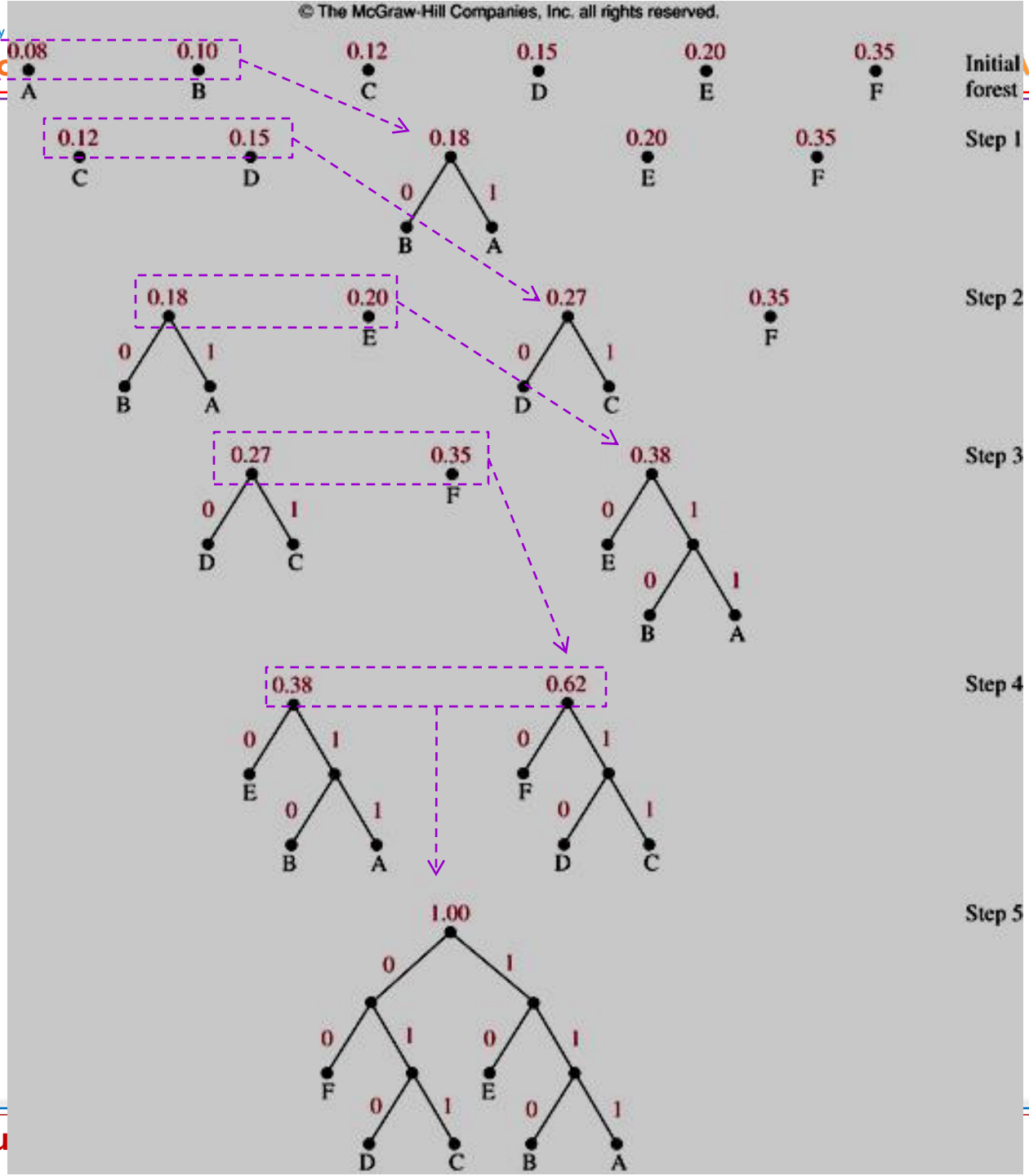
Ex3:

- Use Huffman coding to encode the following symbols with the frequencies listed:

A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35.

- What is the average number of bits used to encode a character?

Prefix Codes: Huffman Coding Algorithm



Prefix Codes: Huffman Coding Algorithm

EX:

- Use Huffman coding to encode the following symbols with the frequencies listed:

A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35.

- What is the average number of bits used to encode a character?

- *The encoding produced encodes A by 111, B by 110, C by 011, D by 010, E by 10, and F by 00.*

- *The average number of bits used to encode a symbol using this encoding is*

$$3 \cdot 0.08 + 3 \cdot 0.10 + 3 \cdot 0.12 + 3 \cdot 0.15 + 2 \cdot 0.20 + 2 \cdot 0.35 = 2.45$$

Prefix Codes: Huffman Coding Algorithm

ALGORITHM 2 Huffman Coding.

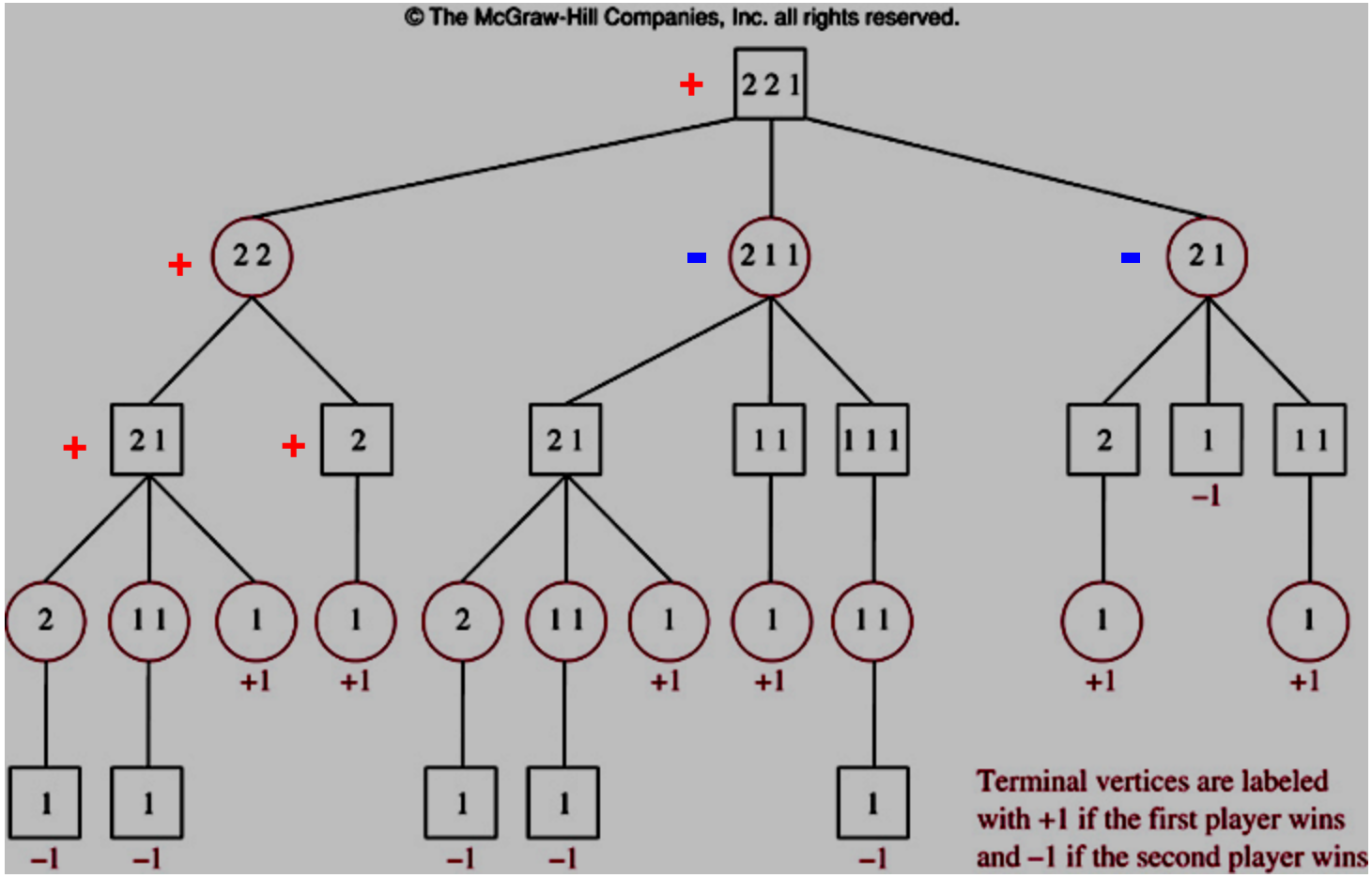
procedure *Huffman*(C: symbols a_i with frequencies $w_i, i = 1, \ldots, n$)
F := forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
while F is not a tree
begin
  Replace the rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree having 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…

DEFINITION 1

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

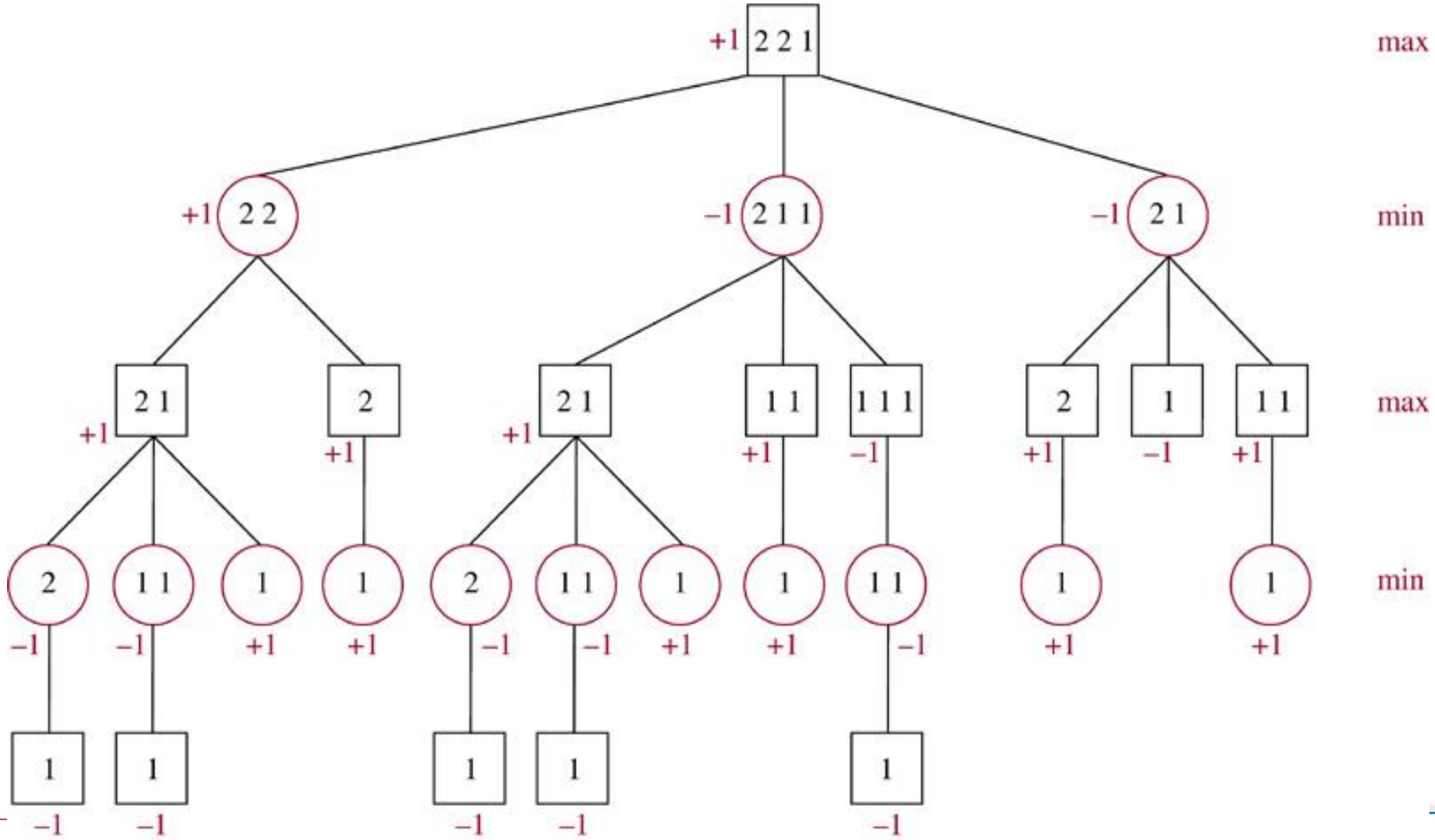
(*i*) the value of a leaf is the payoff to the first player when the game terminates in the position represented by this leaf.

(*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.

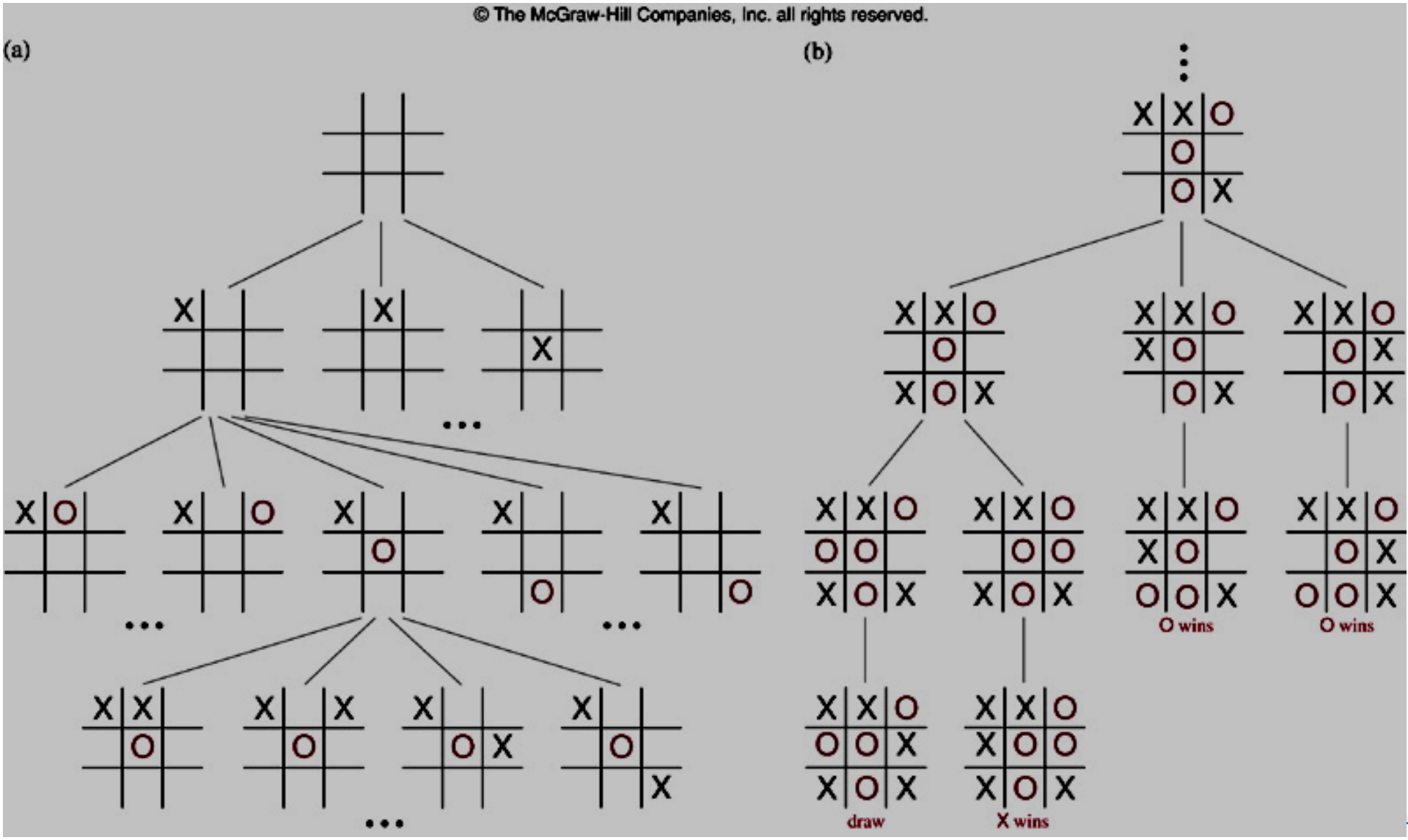
THEOREM 3

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…



Game Trees : Tic-tac-toe – carô 3



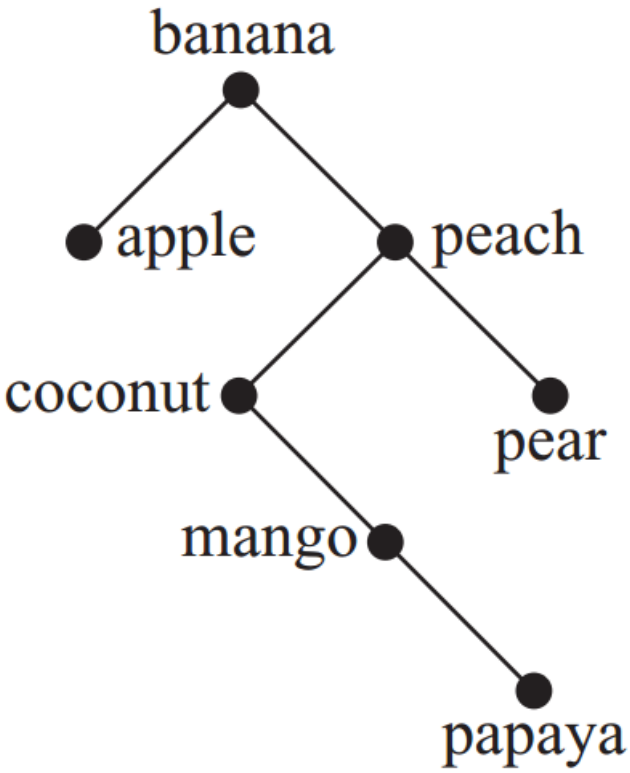
1. Build a binary search tree for the words *banana, peach, apple, pear, coconut, mango*, and *papaya* using alphabetical order.

3. How many comparisons are needed to locate or to add each of these words in the search tree for Exercise 1, starting fresh each time?

a) *pear*
b) *banana*
c) *kumquat*
d) *orange*

a b c d e f g h i j k l m n o p q r s t u v w x y z

1.



3. How many comparisons are needed to locate or to add each of these words in the search tree for Exercise 1, starting fresh each time?

a) *pear* **b)** *banana*

c) *kumquat* **d)** *orange*

3. **a)** 3 **b)** 1 **c)** 4 **d)** 5

19. Which of these codes are prefix codes?

a) a: 11, e: 00, t: 10, s: 01

b) a: 0, e: 1, t: 01, s: 001

c) a: 101, e: 11, t: 001, s: 011, n: 010

d) a: 010, e: 11, t: 011, s: 1011, n: 1001, i: 10101

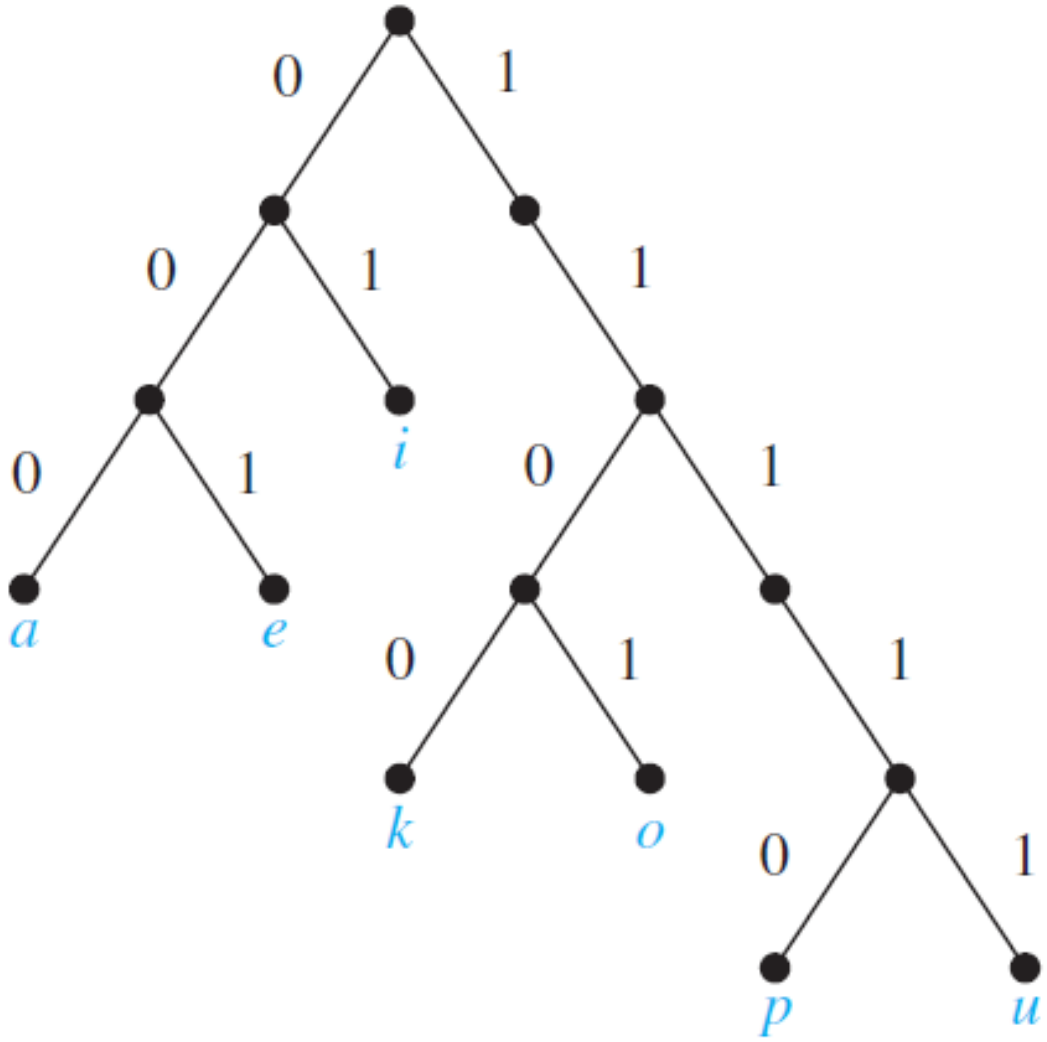
20. Construct the binary tree with prefix codes representing these coding schemes.

a) a: 11, e: 0, t : 101, s: 100

b) a: 1, e: 01, t : 001, s: 0001, n: 00001

c) a: 1010, e: 0, t : 11, s: 1011, n: 1001, i: 10001

21. What are the codes for *a*, *e*, *i*, *k*, *o*, *p*, and *u* if the coding scheme is represented by this tree?



22. Given the coding scheme *a*: 001, *b*: 0001, *e*: 1, *r*: 0000, *s*: 0100, *t* : 011, *x*: 01010, find the word represented by

a) 01110100011; **b)** 0001110000.

c) 0100101010 ;**d)** 01100101010

24. Use Huffman coding to encode these symbols with given frequencies: A: 0.10, B: 0.25, C: 0.05, D: 0.15, E: 0.30, F: 0.07, G: 0.08. What is the average number of bits required to encode a symbol?

Traversal Algorithms

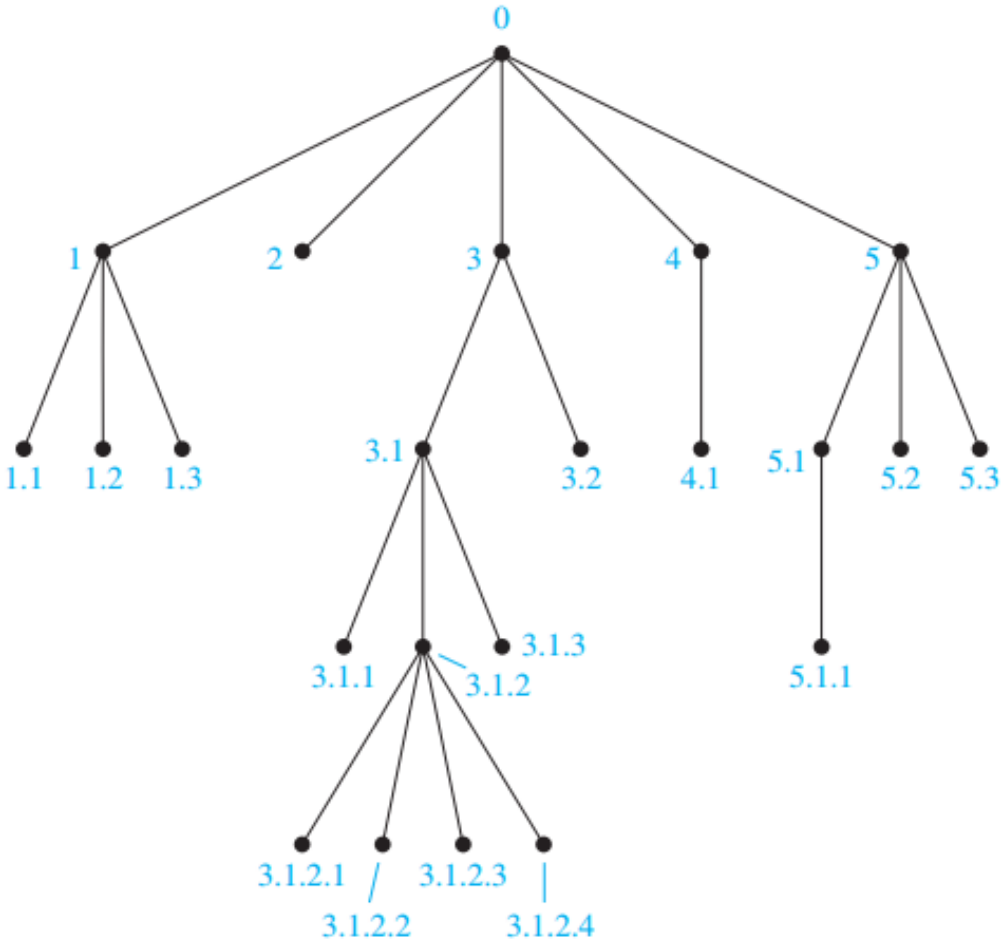
- At a time, a vertex is visited
- Operations are done:
  - Process the visited vertex, e.g. list it's information
  - Traversing recursively subtree.
- Bases on orders of tasks, traversals are classified into:
  - Preorder traversal. N L R
  - Inorder traversal. L N R
  - Postorder traversal. L R N

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, . . . , 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 $A.1$, $A.2$, . . . , $A.k_v$.

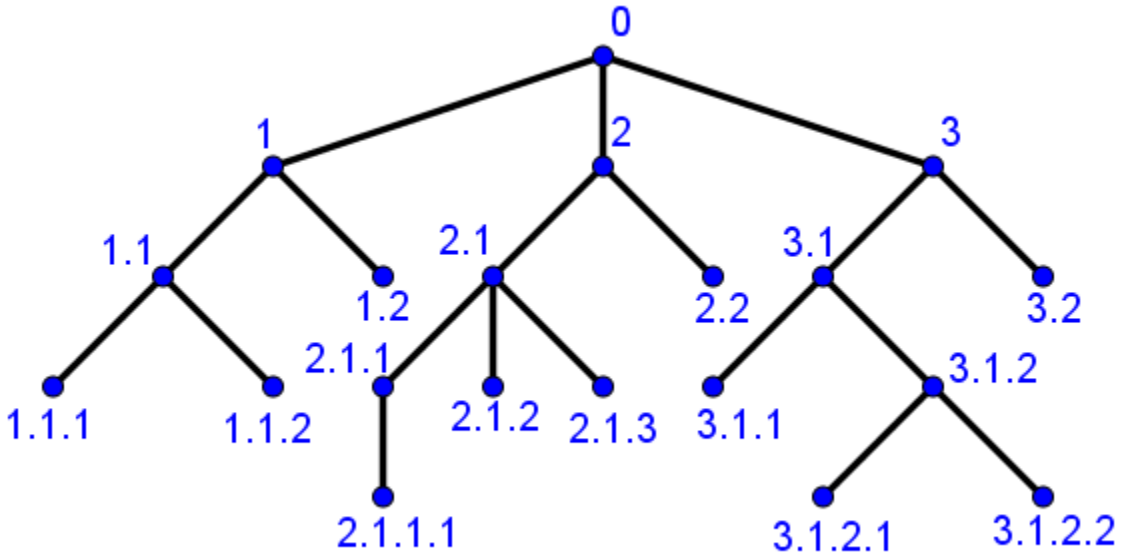


Can the leaves of an ordered rooted tree have the following list of universal addresses? If so, construct such an ordered rooted tree.

a) 1.1.1, 1.1.2, 1.2, 2.1.1.1, 2.1.2, 2.1.3, 2.2, 3.1.1, 3.1.2.1, 3.1.2.2, 3.2

b) 1.1, 1.2.1, 1.2.2, 1.2.3, 2.1, 2.2.1, 2.3.1, 2.3.2, 2.4.2.1, 2.4.2.2, 3.1, 3.2.1, 3.2.2

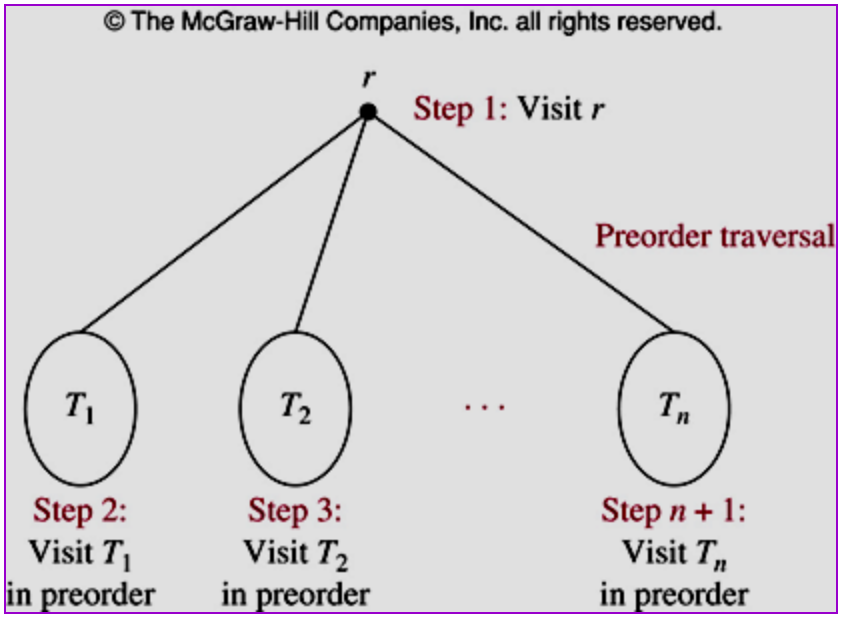
c) 1.1, 1.2.1, 1.2.2, 1.2.2.1, 1.3, 1.4, 2, 3.1, 3.2, 4.1.1.1

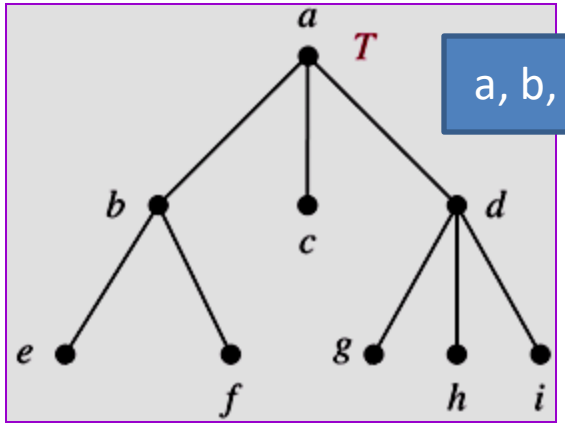


Preorder Traversal

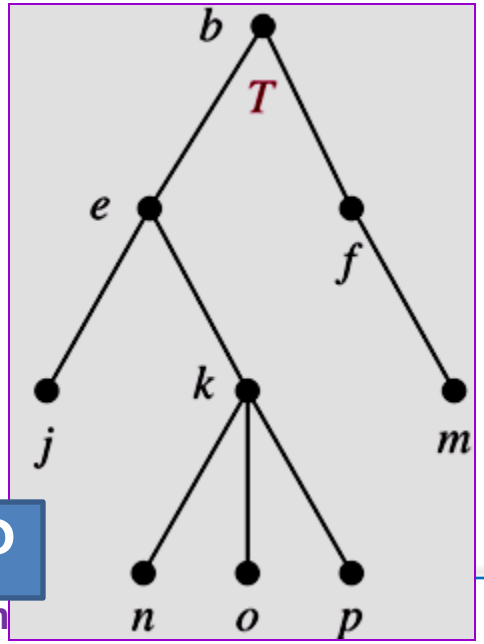
DEFINITION 1

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

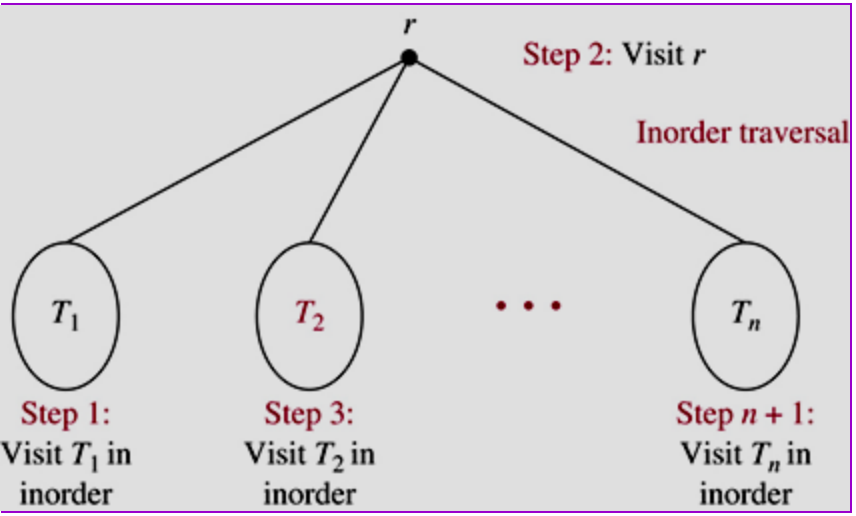


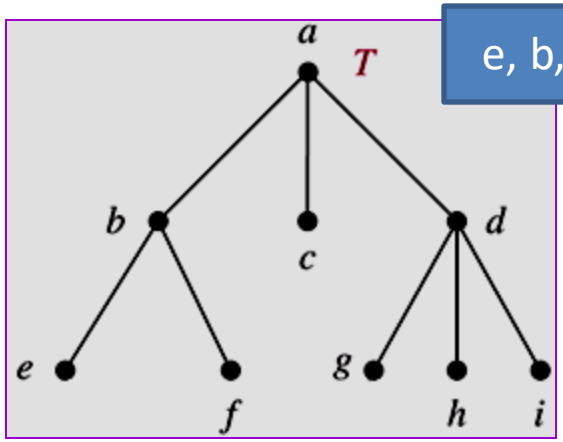
?

Inorder Traversal

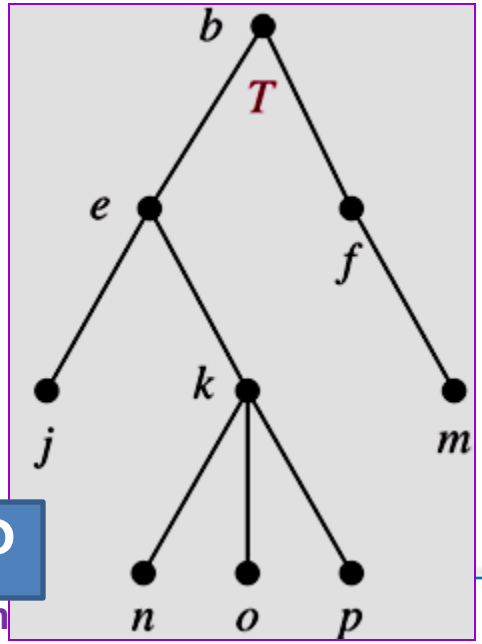
DEFINITION 2

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

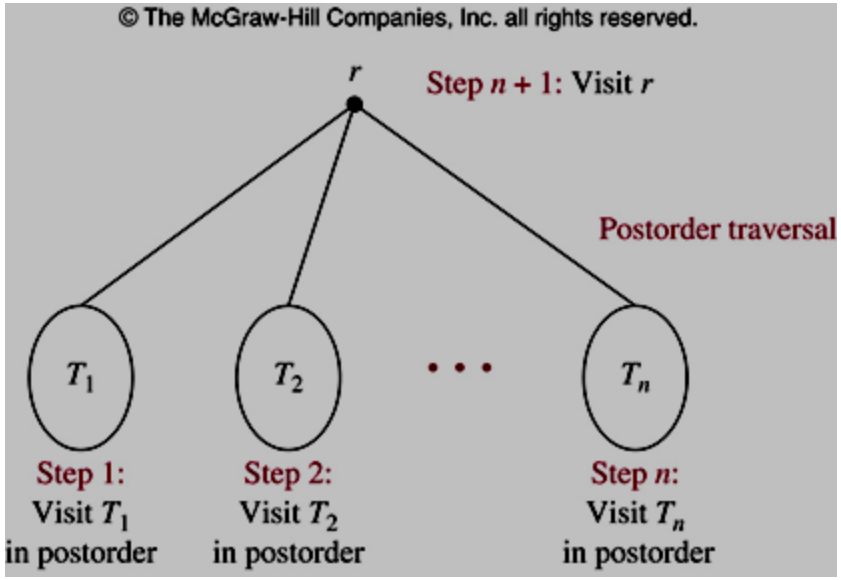


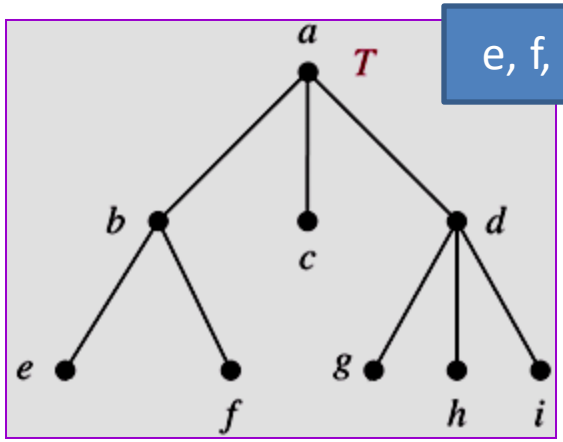
?

Postorder Traversal

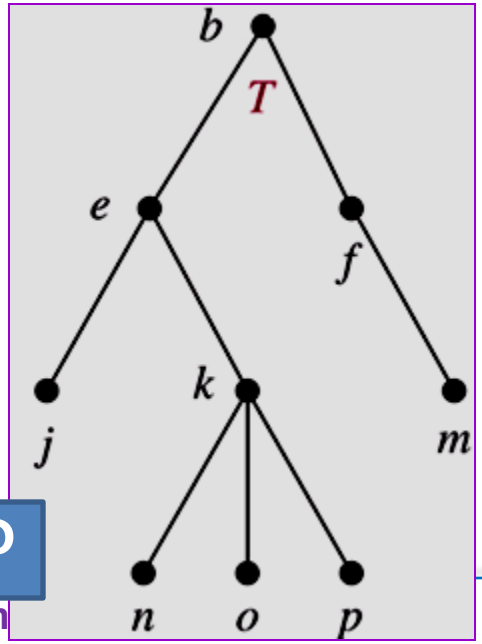
DEFINITION 3

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, ..., then T_n in postorder, and ends by visiting r.





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



?

Traverse Algorithms

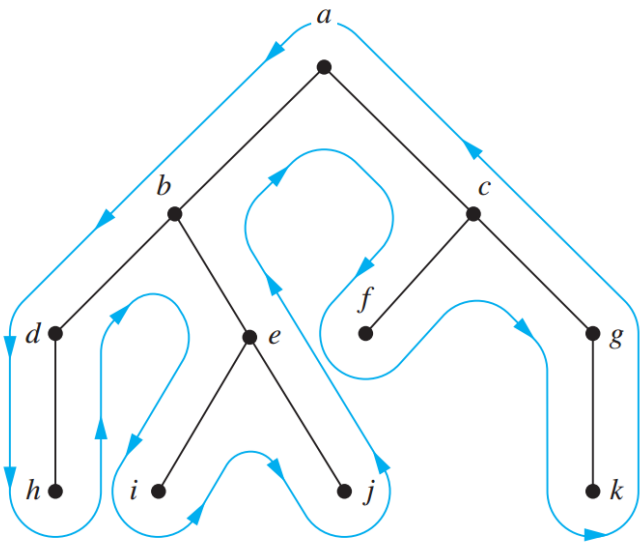


FIGURE 9 **A Shortcut for Traversing an Ordered Rooted Tree in Preorder, Inorder, and Postorder.**

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
  $T(c)$:= subtree with c as its root
  preorder($T(c)$)

ALGORITHM 2 Inorder Traversal.

procedure *inorder*(T: ordered rooted tree)
r := root of T
if r is a leaf **then** list r
else
  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($T(c)$)

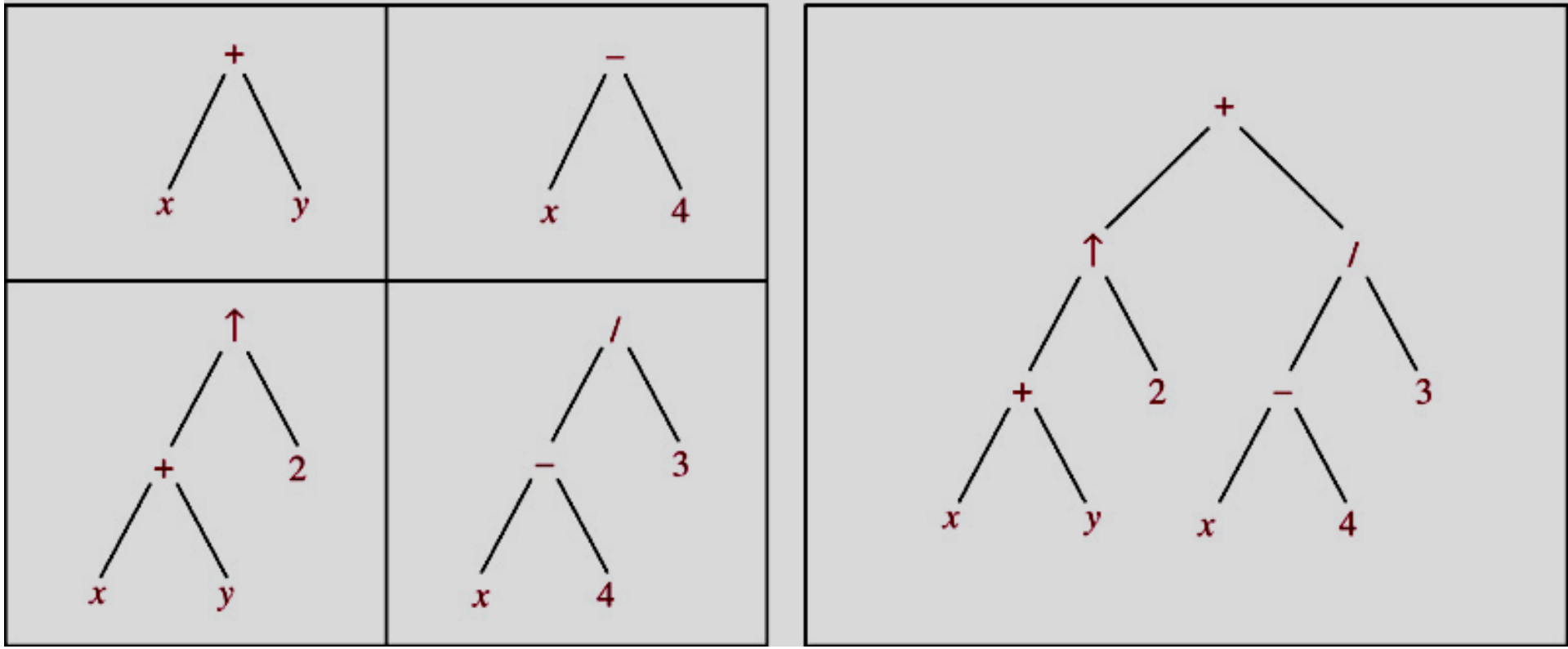
ALGORITHM 3 Postorder Traversal.

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

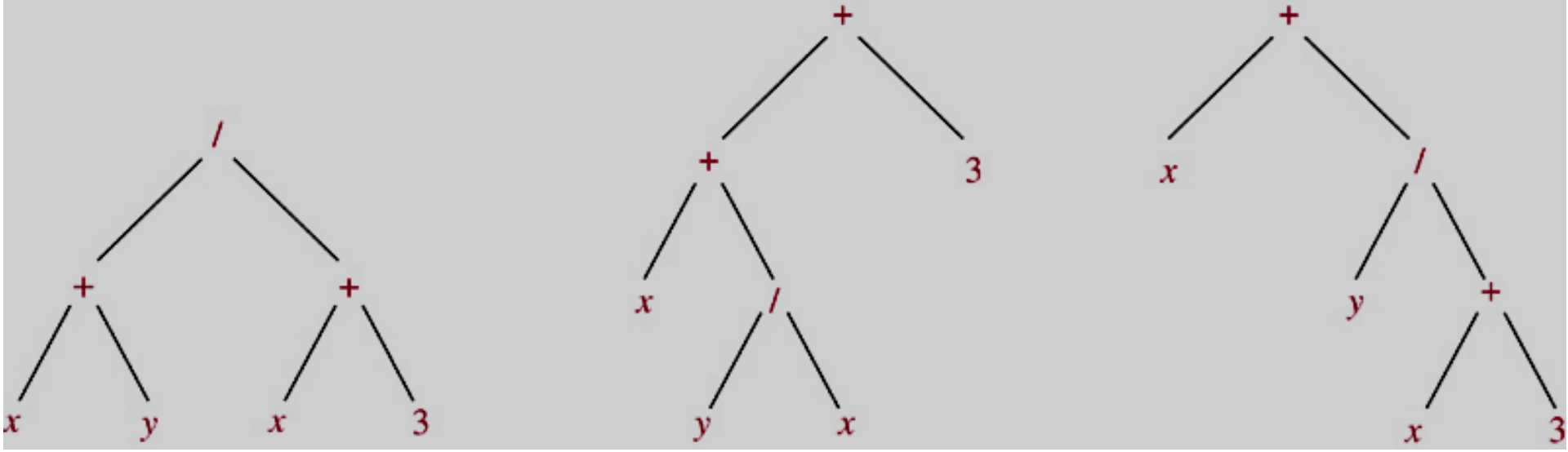


FIGURE 11 Rooted Trees Representing $(x + y)/(x + 3)$, $(x + (y/x)) + 3$, and $x + (y/(x + 3))$.

Infix, Prefix, and Postfix Notation

- **Infix form (dạng trung tố)**:

  operand_1 operator operand_2 x + y

- **Prefix form (tiền tố-Polish notation):**

  operator(operand_1,operand_2) + x y

- **Postfix form (hậu tố, reverse Polish form):**

  (operand_1,operand_2)operator x y +

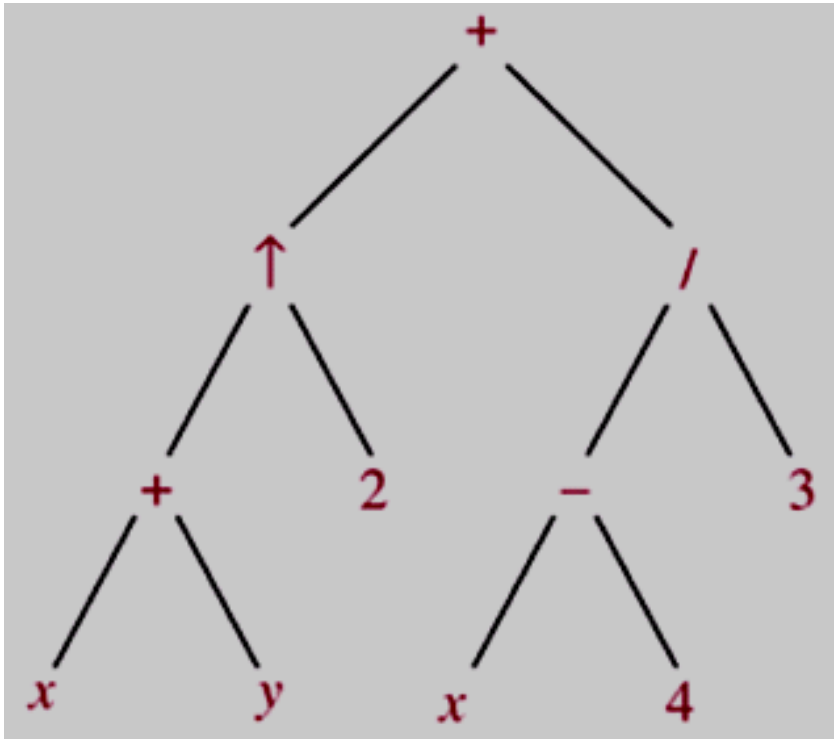
- How to find prefix and postfix form from infix form?

  (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

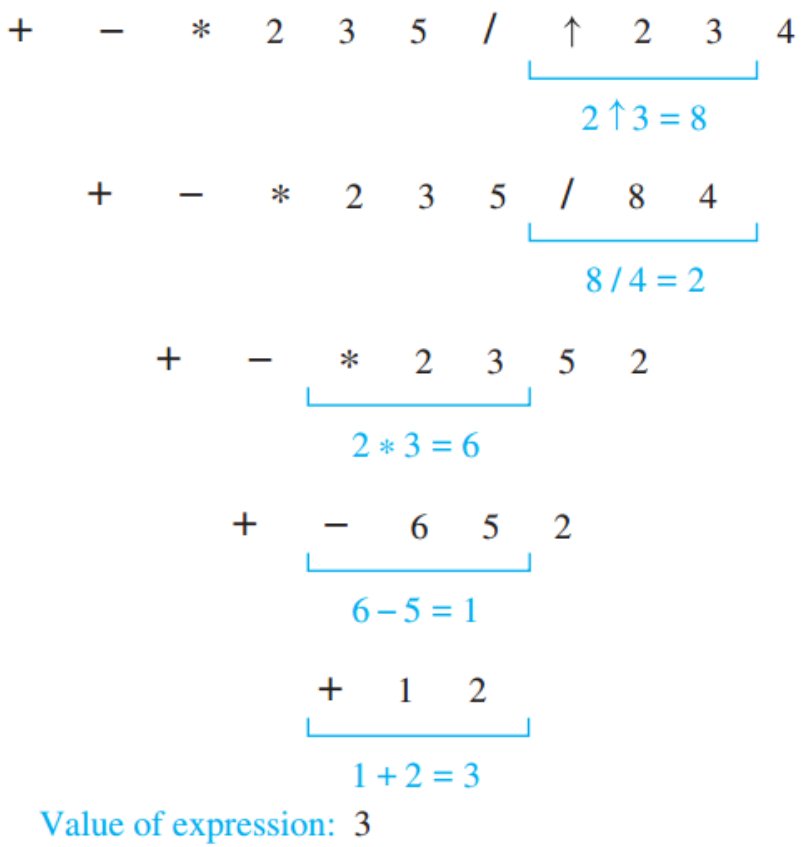


FIGURE 12 **Evaluating a Prefix Expression.**

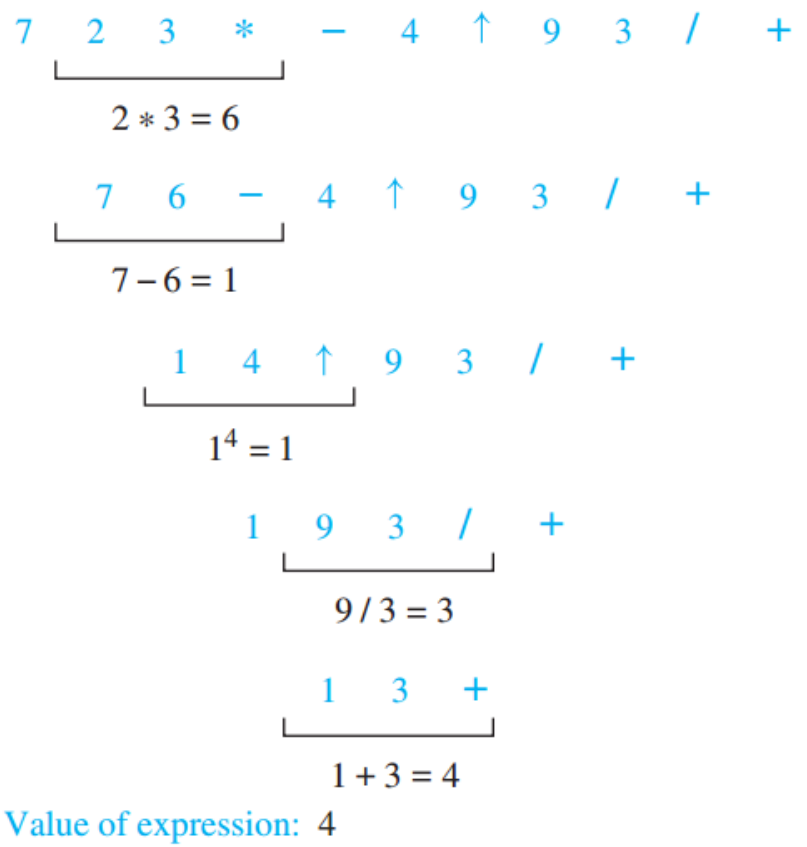


FIGURE 13 **Evaluating a Postfix Expression.**

22. Draw the ordered rooted tree corresponding to each of these arithmetic expressions written in prefix notation. Then write each expression using infix notation.

a) + ∗ + - 5 3 2 1 4

b) ↑ + 2 3 - 5 1

c) ∗ / 9 3 + ∗ 2 4 - 7 6

23. What is the value of each of these prefix expressions?

a) - ∗ 2 / 8 4 3

b) ↑ - ∗ 3 3 ∗ 4 2 5

c) + - ↑ 3 2 ↑ 2 3 / 6 - 4 2

d) ∗ + 3 + 3 ↑ 3 + 3 3 3

24. What is the value of each of these postfix expressions?

a) 5 2 1 - - 3 1 4 ++ ∗

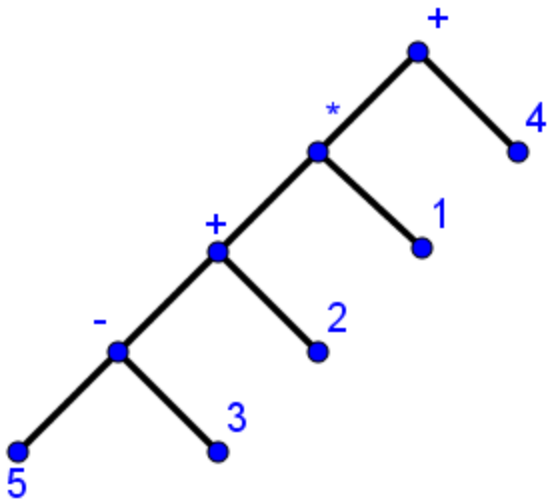
b) 9 3 / 5 + 7 2 - ∗

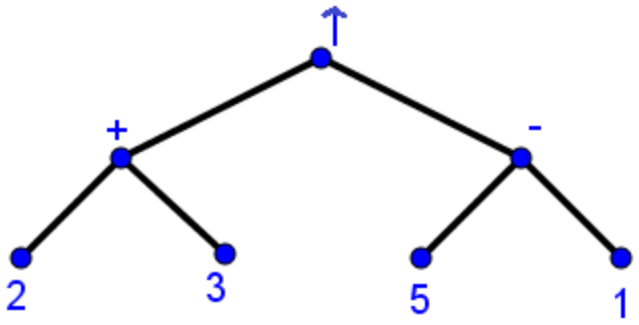
c) 3 2 ∗ 2 ↑ 5 3 - 8 4 / ∗ -

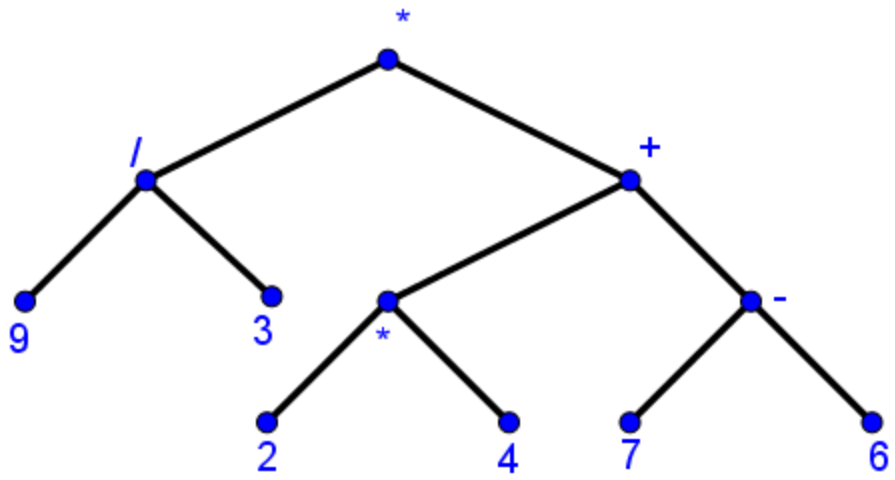
22. **a)** + ∗ + - 5 3 2 1 4

b) ↑ + 2 3 - 5 1

c) ∗ / 9 3 + ∗ 2 4 - 7 6







(a) ((((5 − 3) + 2) ∗ 1) + 4)

(b) ((2 + 3) ↑ (5 − 1))

(c) (((9/3) ∗ ((2 ∗ 4) + (7 − 6)))

23. What is the value of each of these prefix expressions?

a) - * 2 / 8 4 3

b) ↑ - * 3 3 * 4 2 5

c) + - ↑ 3 2 ↑ 2 3 / 6 - 4 2

d) * + 3 + 3 ↑ 3 + 3 3 3

24. What is the value of each of these postfix expressions?

a) 5 2 1 - - 3 1 4 ++ *

b) 9 3 / 5 + 7 2 - *

c) 3 2 * 2 ↑ 5 3 - 8 4 / * -

23. What is the value of each of these prefix expressions?

a) - * 2 / 8 4 3

b) ↑ - * 3 3 * 4 2 5

c) + - ↑ 3 2 ↑ 2 3 / 6 - 4 2

d) * + 3 + 3 ↑ 3 + 3 3 3

(a) 1
(b) 1
(c) 4
(d) 2205

24. What is the value of each of these postfix expressions?

a) 5 2 1 - - 3 1 4 ++ *

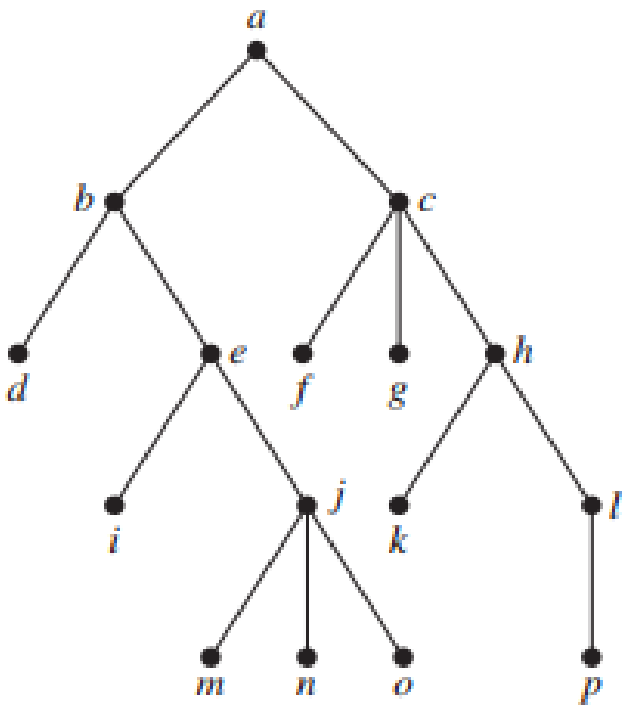
b) 9 3 / 5 + 7 2 - *

c) 3 2 * 2 ↑ 5 3 - 8 4 / * -

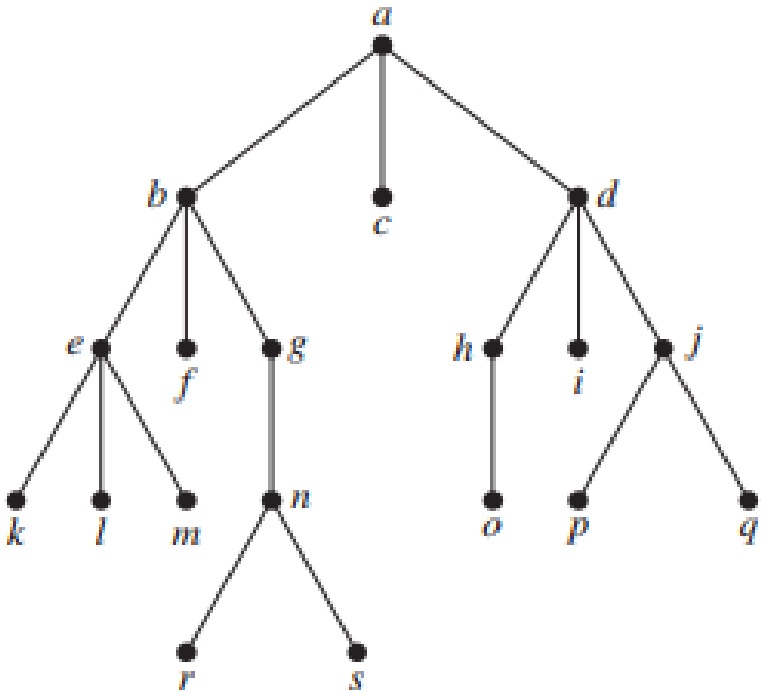
(a) 32
(b) 40
(c) 32

Determine the order in which a preorder traversal visits the vertices of the given ordered rooted tree.

8.



9.



10.4- Spanning Trees

10.4- Spanning Trees

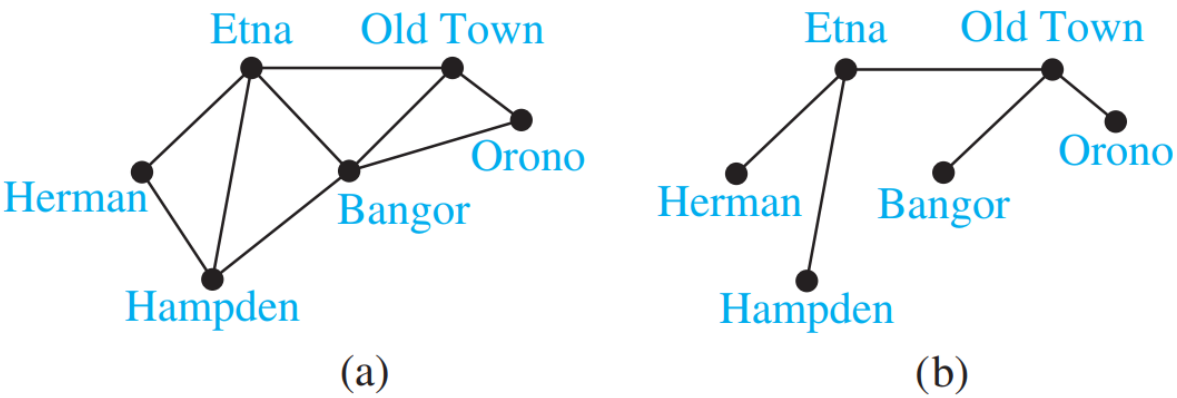


FIGURE 1 **(a) A Road System and (b) a Set of Roads to Plow.**

DEFINITION 1

Let G be a simple graph. A *spanning tree* of G is a subgraph of G that is a tree containing every vertex of G.

10.4- Spanning Trees

Find a spanning tree of the simple graph G shown in Figure 2.

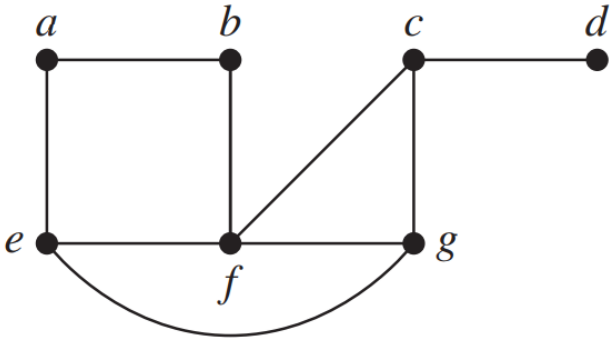


FIGURE 2 The Simple Graph G.

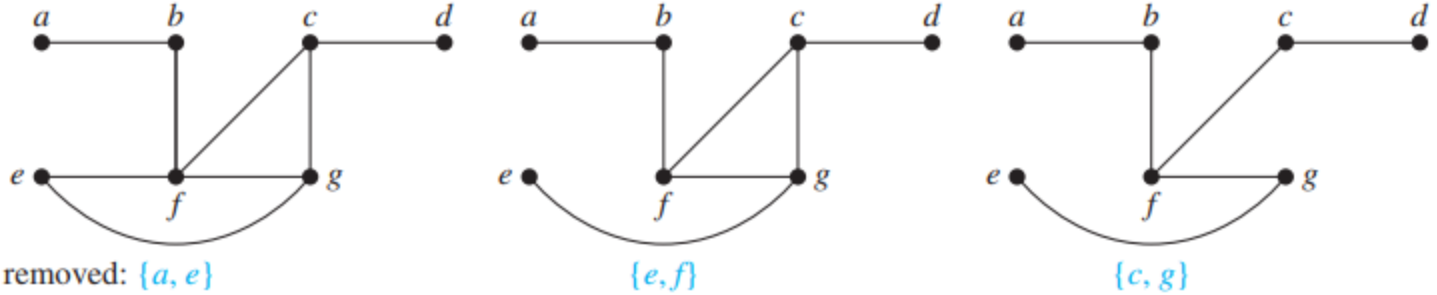


FIGURE 3 Producing a Spanning Tree for G by Removing Edges That Form Simple Circuits.

10.4- Spanning Trees

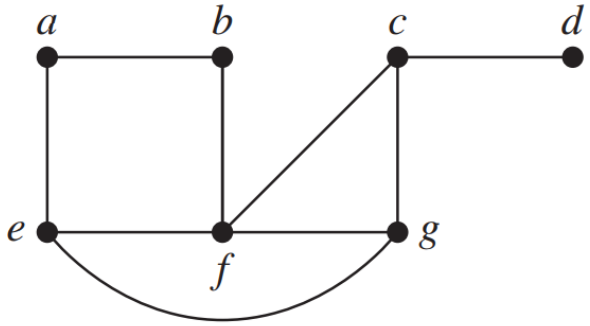


FIGURE 2 The Simple Graph G.

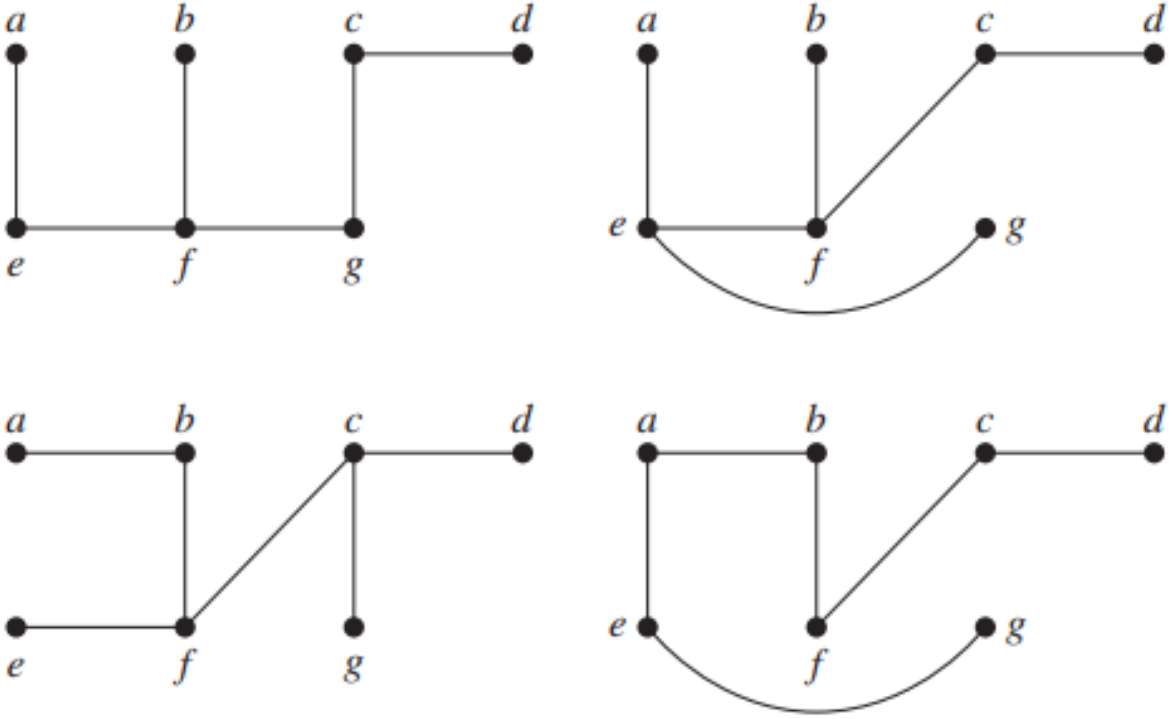


FIGURE 4 Spanning Trees of G.

10.4- Spanning Trees

THEOREM 1

A simple graph is connected if and only if it has a spanning tree.

Depth-First Search

EXAMPLE 3

Use depth-first search to find a spanning tree for the graph G shown in Figure 6.

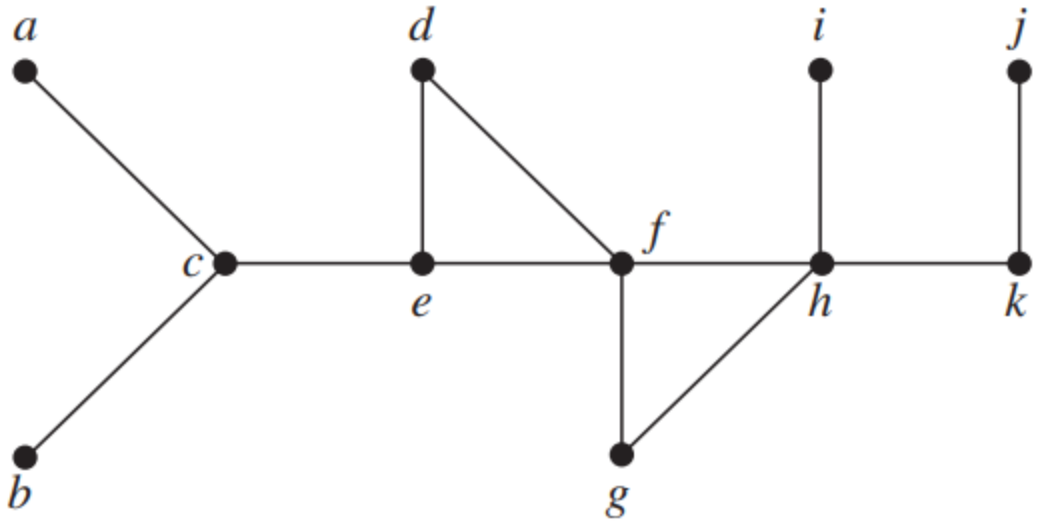
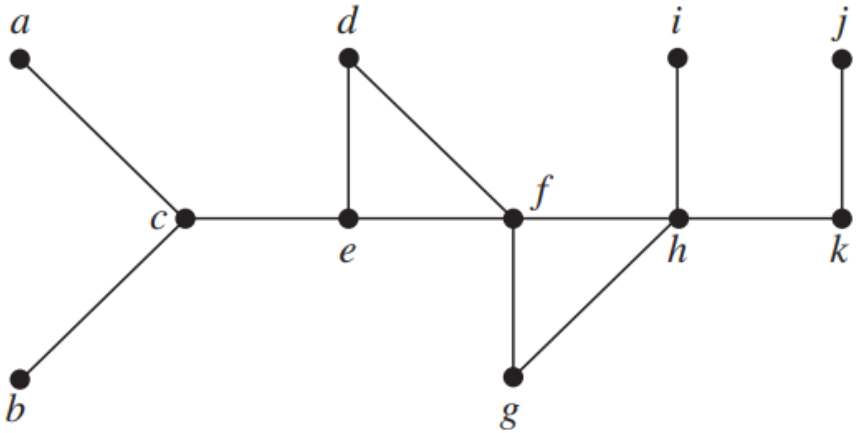


FIGURE 6 The Graph G.

Depth-First Search



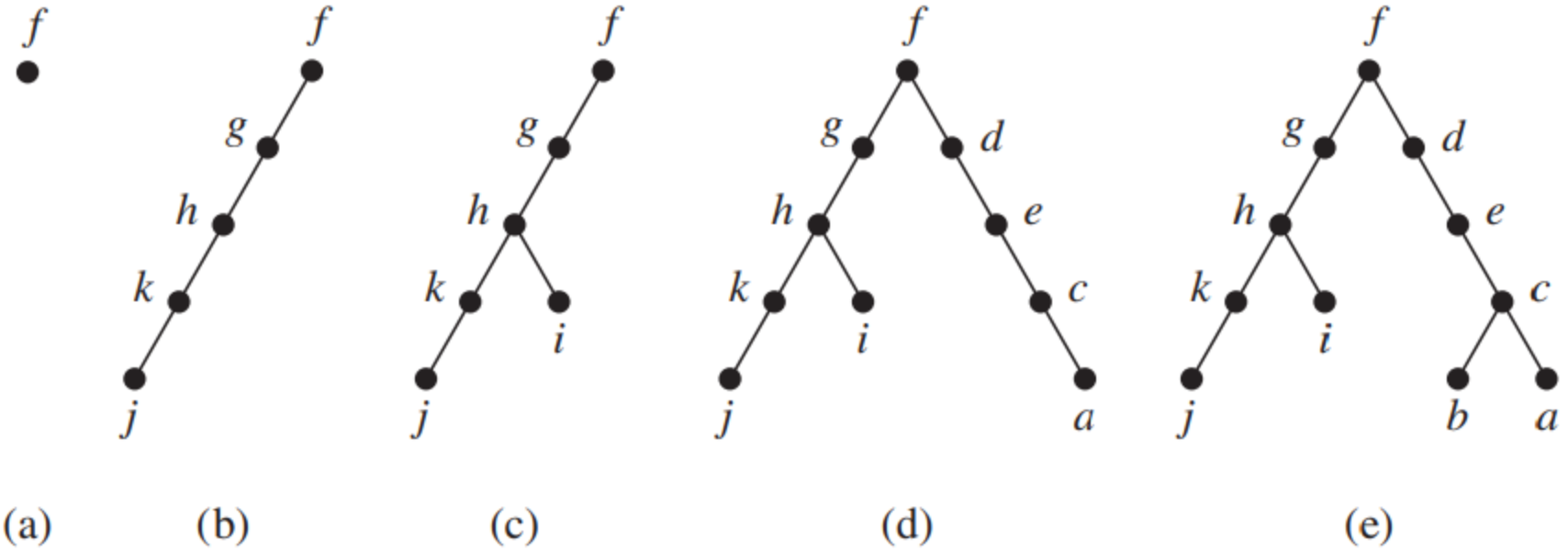


FIGURE 7 **Depth-First Search of *G*.**

Depth-First Search

ALGORITHM 1 Depth-First Search.

procedure *DFS*(*G*: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of the vertex v_1
visit(v_1)
procedure *visit*(*v*: vertex of *G*)
for each vertex *w* adjacent to *v* and not yet in *T*
    add vertex *w* and edge {*v*, *w*} to *T*
    visit(*w*)

Breadth-First Search

EXAMPLE 5

Use breadth-first search to find a spanning tree for the graph shown in Figure 9.

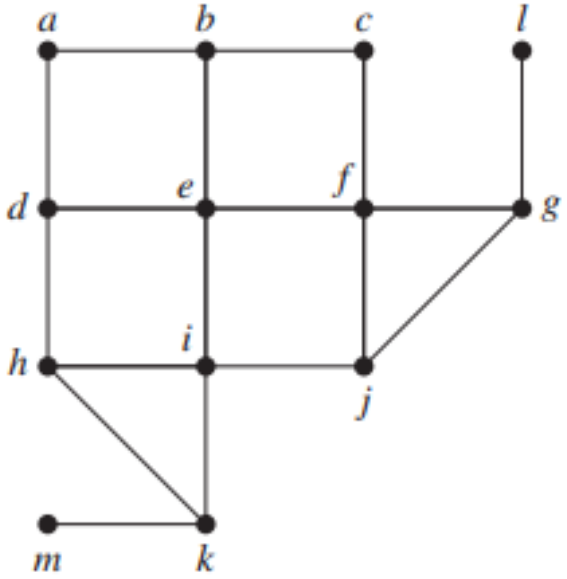


FIGURE 9 **A Graph *G*.**

Breadth-First Search

Solution:

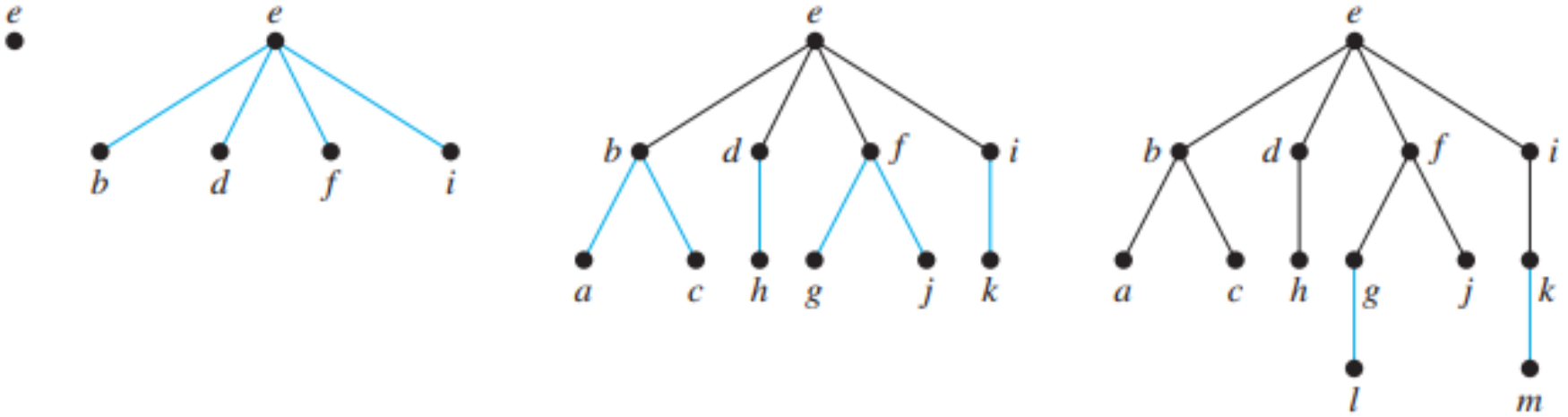
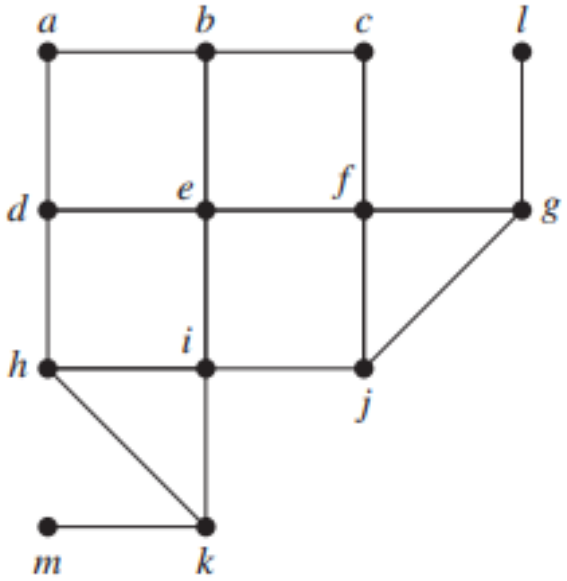


FIGURE 10 **Breadth-First Search of (**



Breadth-First Search

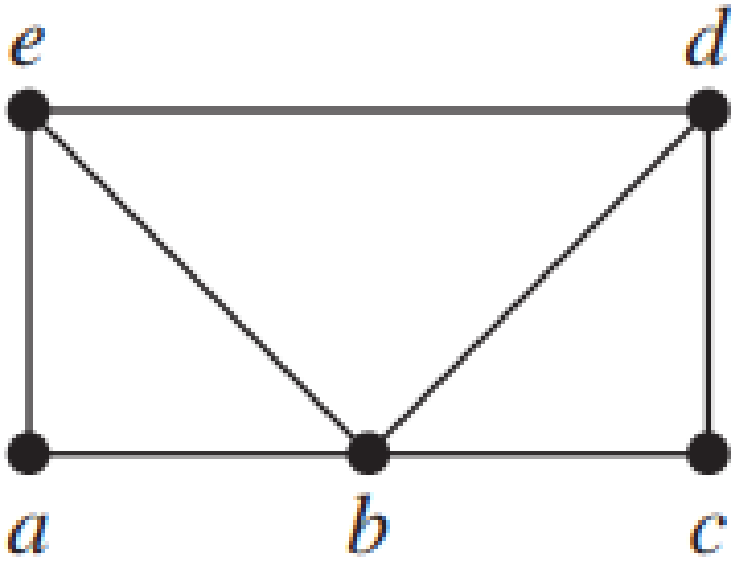
ALGORITHM 2 Breadth-First Search.

procedure *BFS* (*G*: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of vertex v_1
L := empty list
put v_1 in the list L of unprocessed vertices
while L is not empty
    remove the first vertex, v, from L
     for each neighbor w of v
          if w is not in L and not in T **then**
               add w to the end of the list L
               add w and edge $\{v, w\}$ to T

Backtracking Applications

EXAMPLE 6

Graph Colorings How can backtracking be used to decide whether a graph can be colored using n colors?



Backtracking Applications

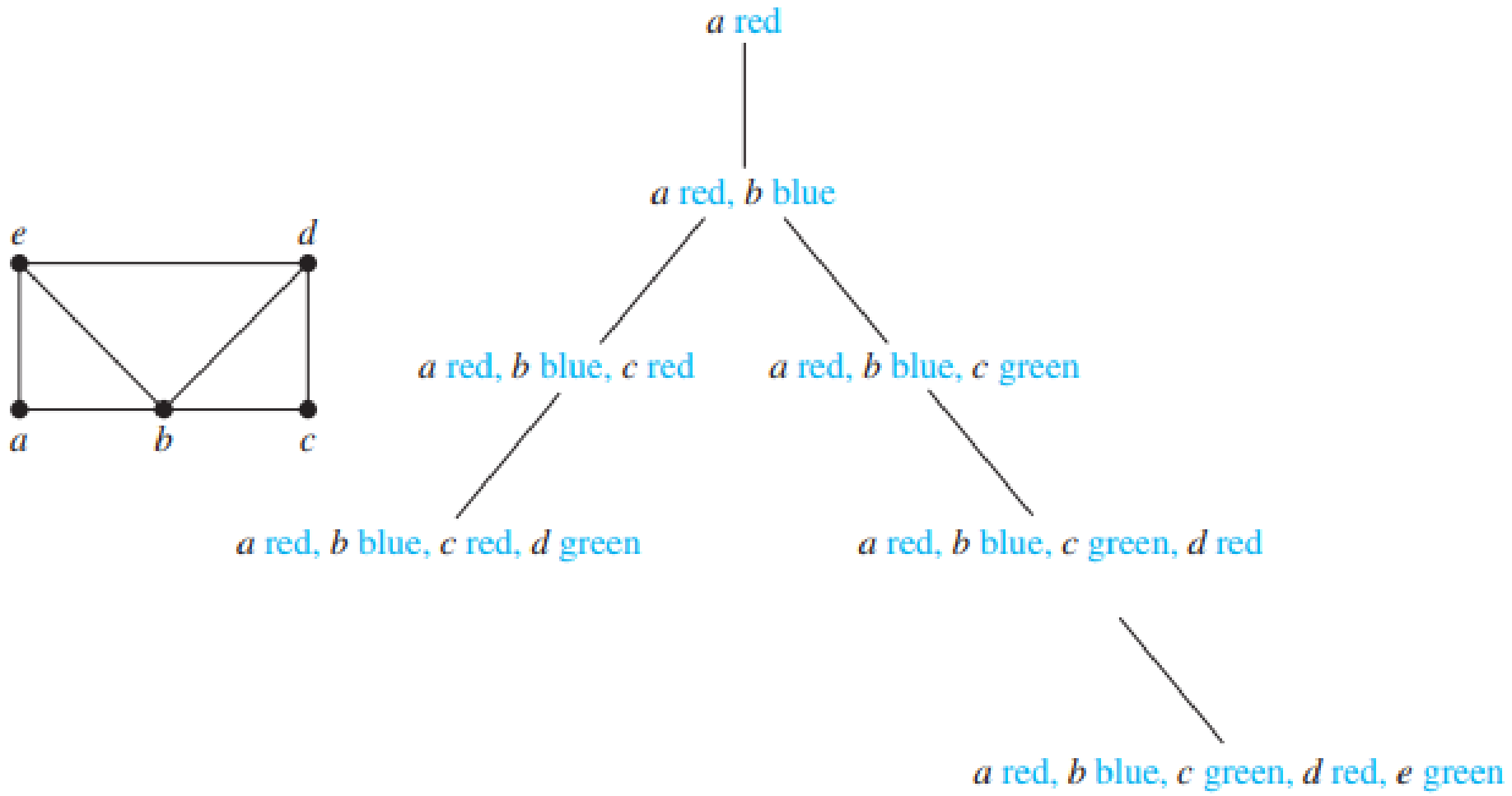


FIGURE 11 **Coloring a Graph Using Backtracking.**

Backtracking Applications

EXAMPLE 7

The *n*-Queens Problem The *n*-queens problem asks how *n* queens can be placed on an $n \times n$ chessboard so that no two queens can attack one another. How can backtracking be used to solve the *n*-queens problem?

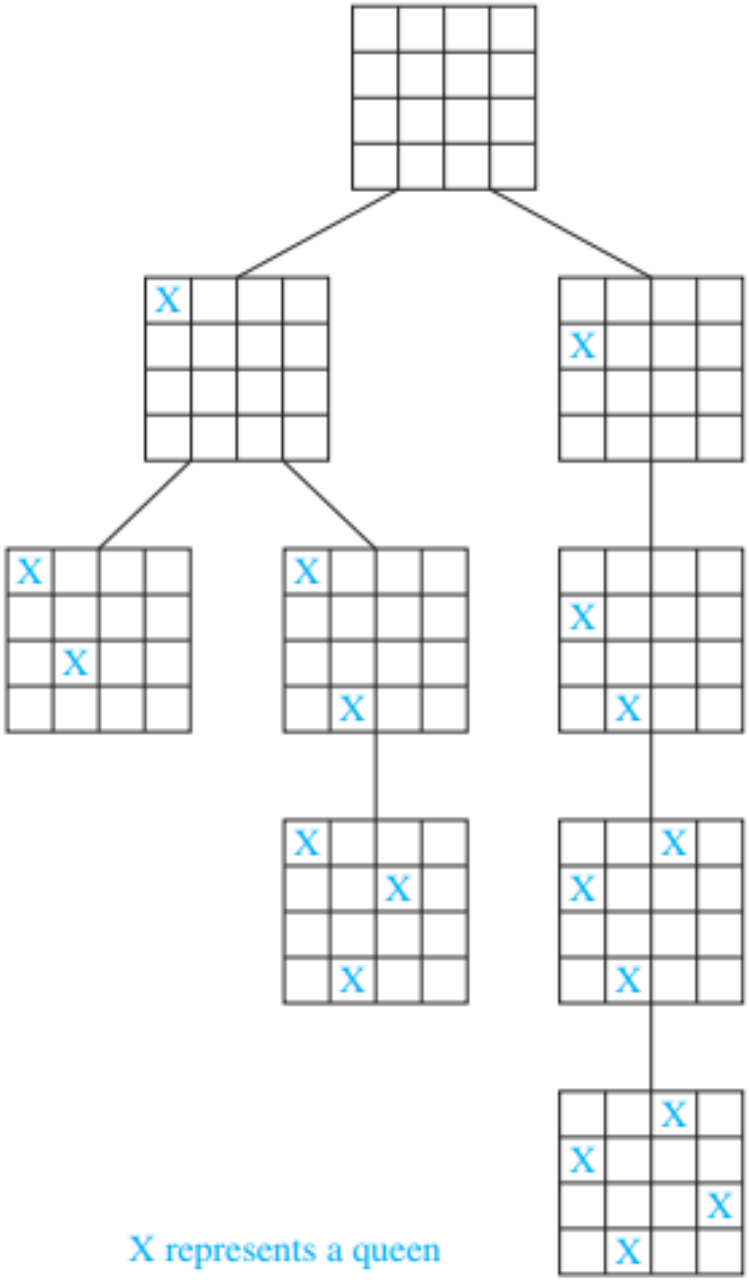


FIGURE 12 **A Backtracking Solution of the Four-Queens Problem.**

Backtracking Applications

EXAMPLE 7

Sums of Subsets Consider this problem. Given a set of positive integers $x_1, x_2, \ldots, x_n$, find a subset of this set of integers that has M as its sum. How can backtracking be used to solve this problem?

Finding a subset of {31, 27, 15, 11, 7, 5} with the sum equal to 39?

Finding a subset of {31, 27, 15, 11, 7, 5} with the sum equal to 39?

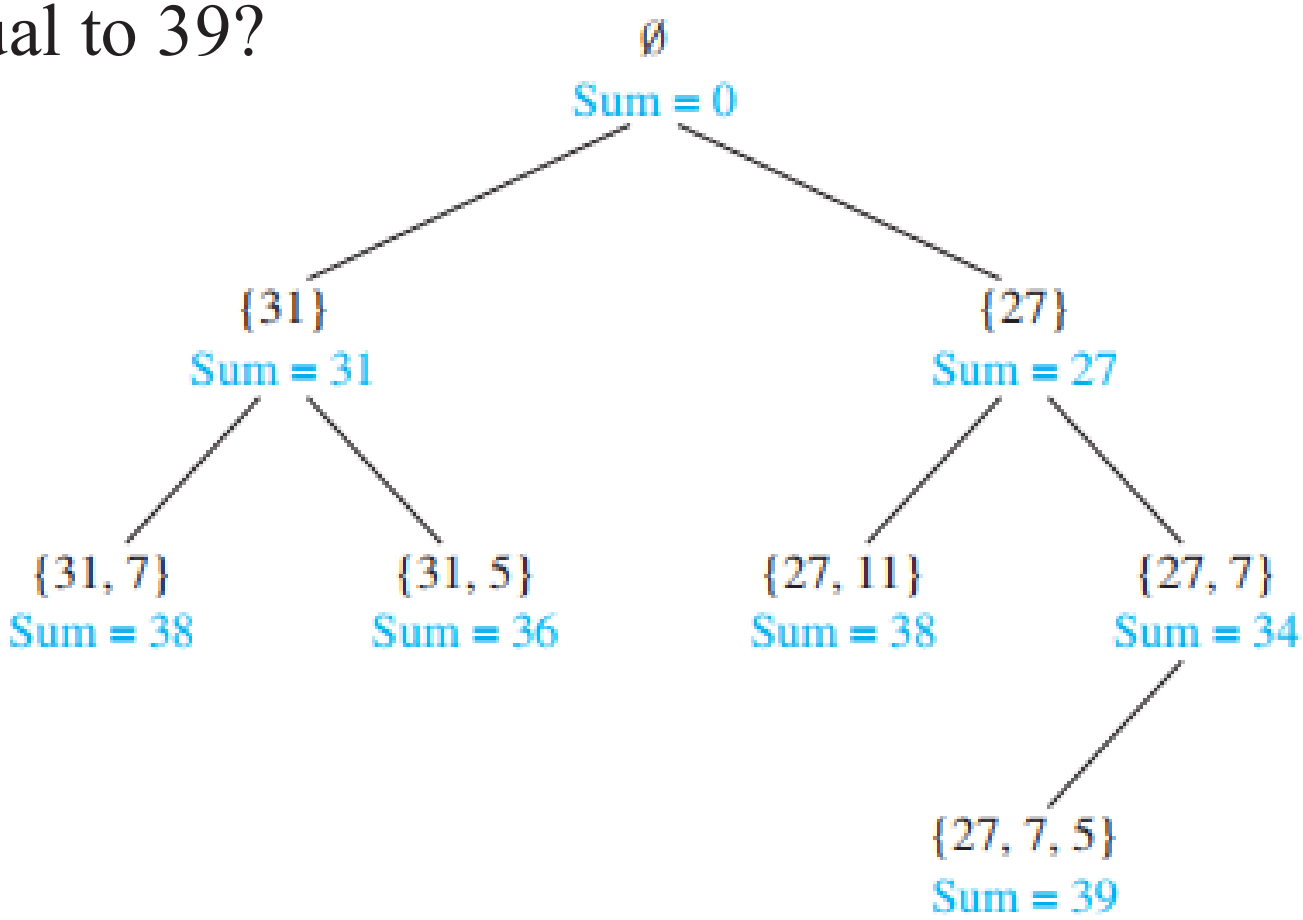
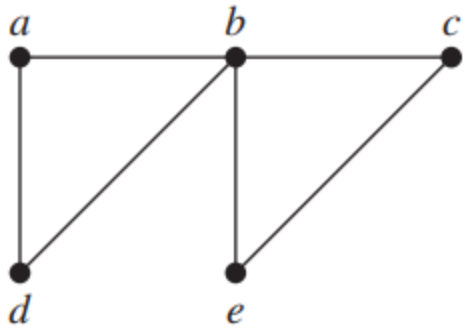


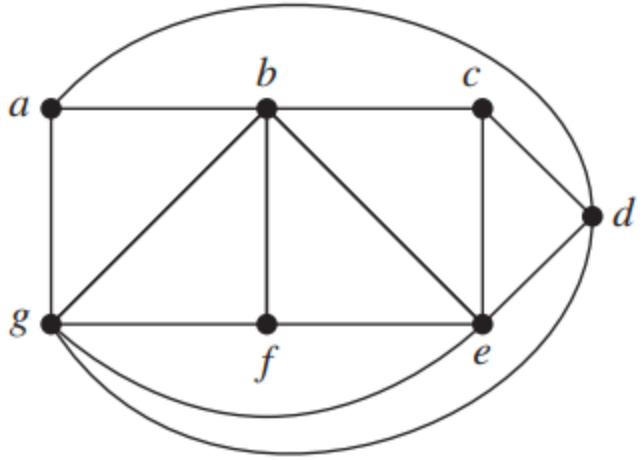
FIGURE 13 Find a Sum Equal to 39 Using Backtracking.

Find a spanning tree for the graph shown by removing edges in simple circuits.

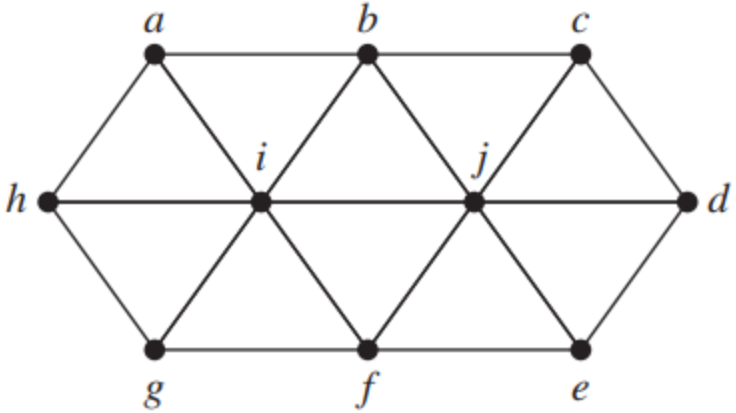
2.



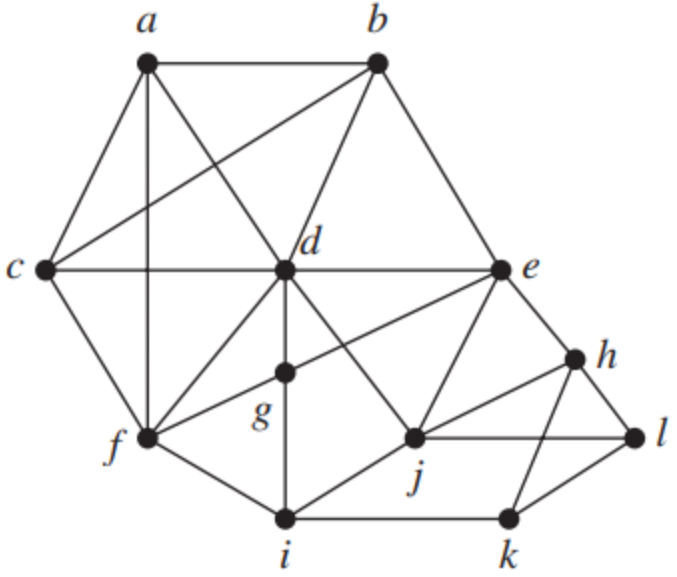
3.



4.



5.



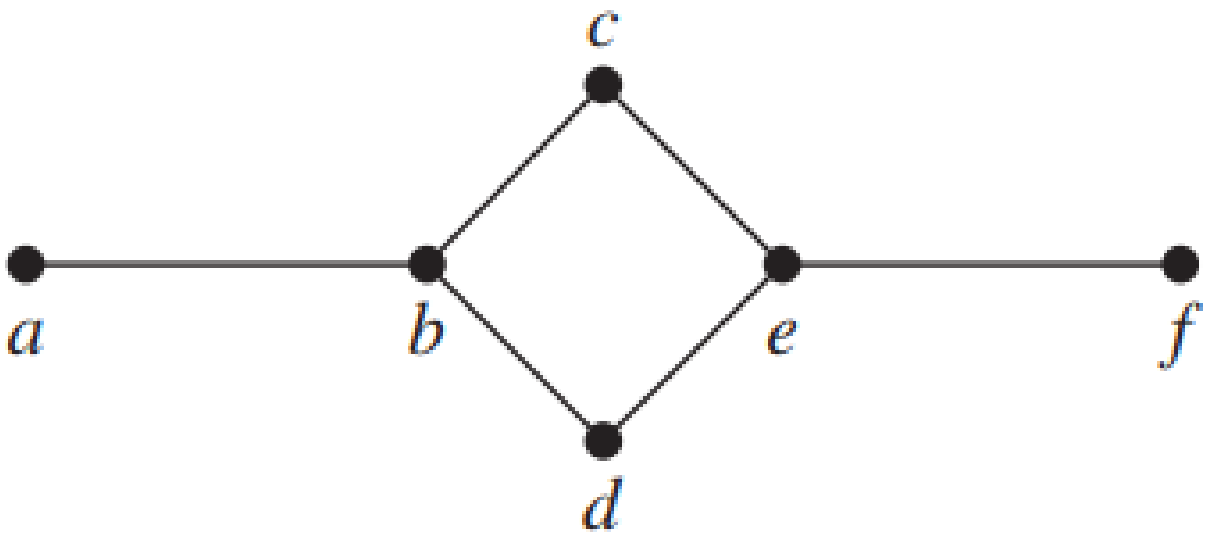
1. How many edges must be removed from a connected graph with n vertices and m edges to produce a spanning tree?

7. Find a spanning tree for each of these graphs.

a) K_5 **b)** $K_{4,4}$ **c)** $K_{1,6}$

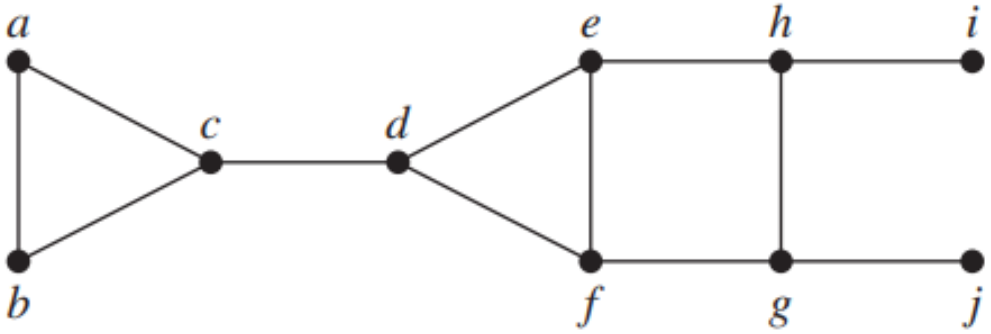
d) Q_3 **e)** C_5 **f)** W_5

8. Draw all the spanning trees of the given simple graphs.

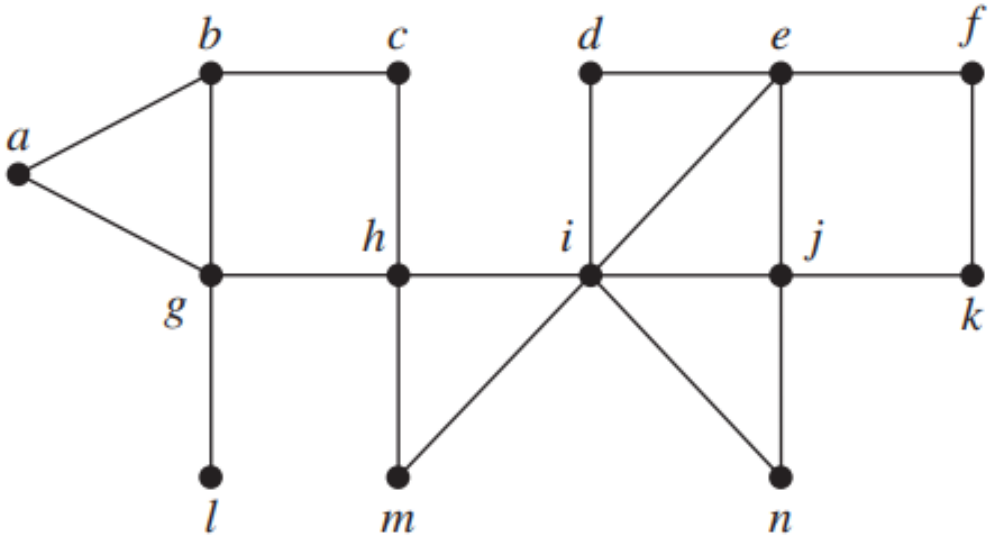


Use depth-first search to produce a spanning tree for the given simple graph. Choose a as the root of this spanning tree and assume that the vertices are ordered alphabetically.

13.

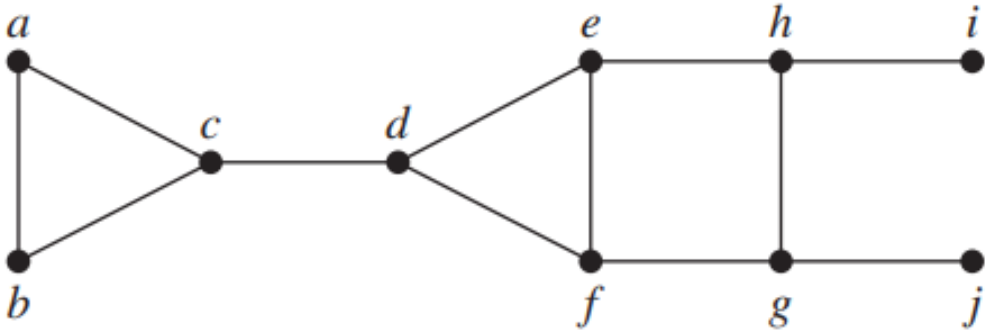


14.

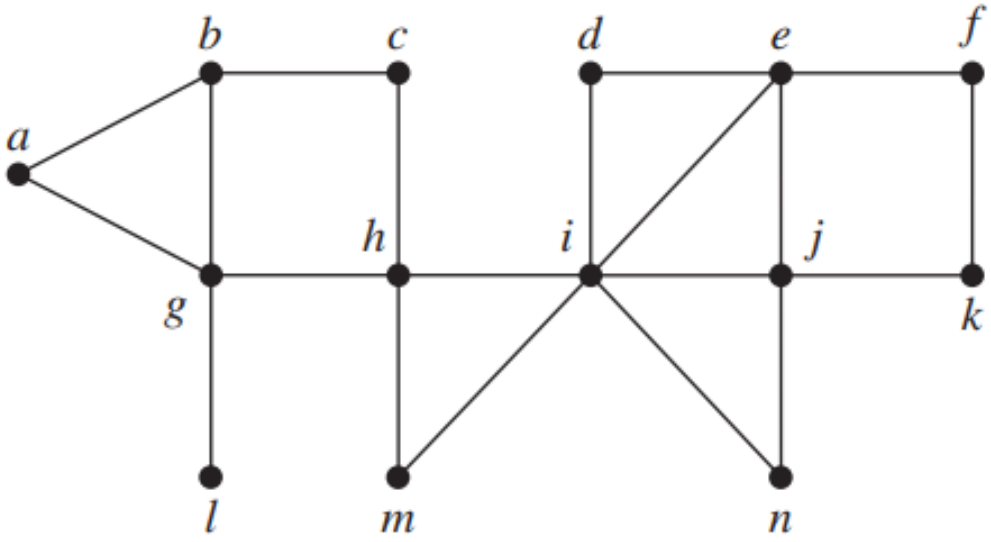


Use breadth-first search to produce a spanning tree for each of the simple graphs in Exercises 13–14. Choose *a* as the root of each spanning tree.

13.



14.



27. Use backtracking to solve the n-queens problem for these values of n.

a) $n = 3$ **b)** $n = 5$ **c)** $n = 6$

28. Use backtracking to find a subset, if it exists, of the set $\{27, 24, 19, 14, 11, 8\}$ with sum

a) 20. **b)** 41. **c)** 60.

10.5- Minimum Spanning Trees

10.5- Minimum Spanning Trees

Algorithms for Minimum Spanning Trees

DEFINITION 1

A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

Algorithms for Minimum Spanning Trees

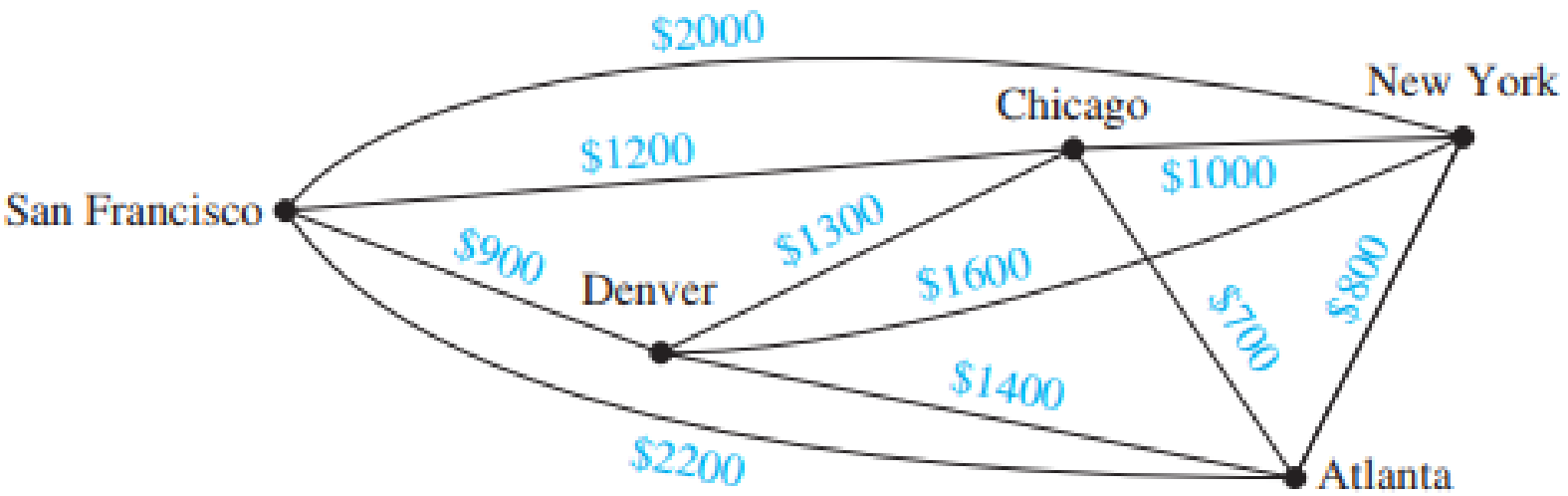
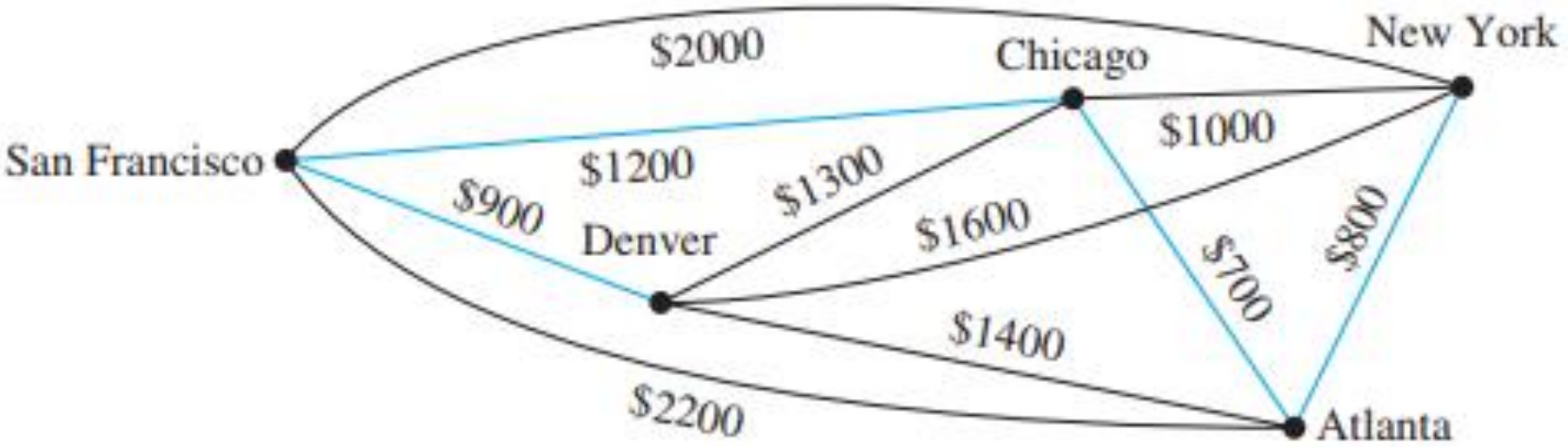


FIGURE 1 **A Weighted Graph Showing Monthly Lease Costs for Lines in a Computer Network.**

Algorithms for Minimum Spanning Trees



Choice	Edge	Cost
1	{Chicago, Atlanta}	$ 700
2	{Atlanta, New York}	$ 800
3	{Chicago, San Francisco}	$1200
4	{San Francisco, Denver}	$ 900
	Total:	$3600

FIGURE 2 **A Minimum Spanning Tree for the Weighted Graph in Figure 1.**

10.5- Minimum Spanning Trees

ALGORITHM 1 Prim's Algorithm.

procedure *Prim*(*G*: weighted connected undirected graph with *n* vertices)
T := a minimum-weight edge
for *i* := 1 **to** *n* - 2
    e := an edge of minimum weight incident to a vertex in *T* and not forming a simple circuit in *T* if added to *T*
    T := *T* with *e* added
return *T* {*T* is a minimum spanning tree of *G*}

10.5- Minimum Spanning Trees

EXAMPLE 2

Use Prim's algorithm to find a minimum spanning tree in the graph shown in Figure 3.

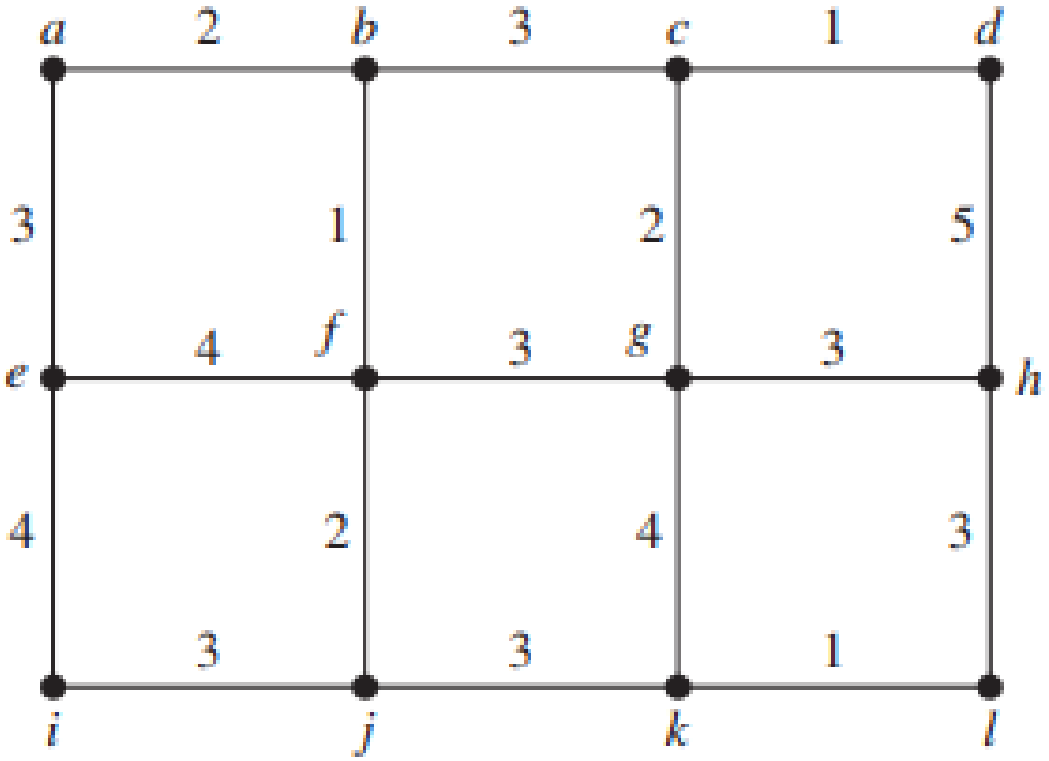
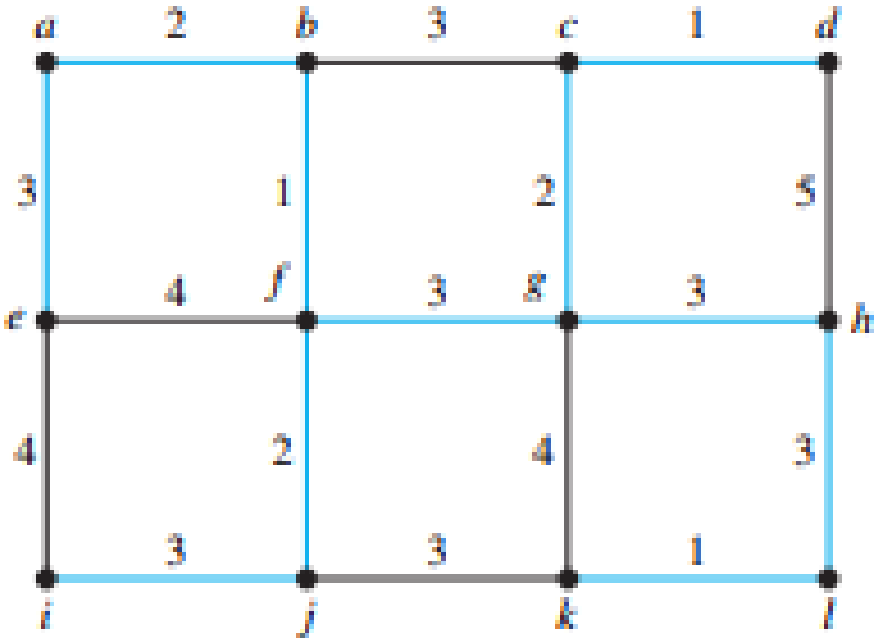


FIGURE 3 A Weighted Graph.



(a)

Choice	Edge	Weight
1	$\{b, f\}$	1
2	$\{a, b\}$	2
3	$\{f, j\}$	2
4	$\{a, e\}$	3
5	$\{i, j\}$	3
6	$\{f, g\}$	3
7	$\{c, g\}$	2
8	$\{c, d\}$	1
9	$\{g, h\}$	3
10	$\{h, l\}$	3
11	$\{k, l\}$	1
	Total:	24

(b)

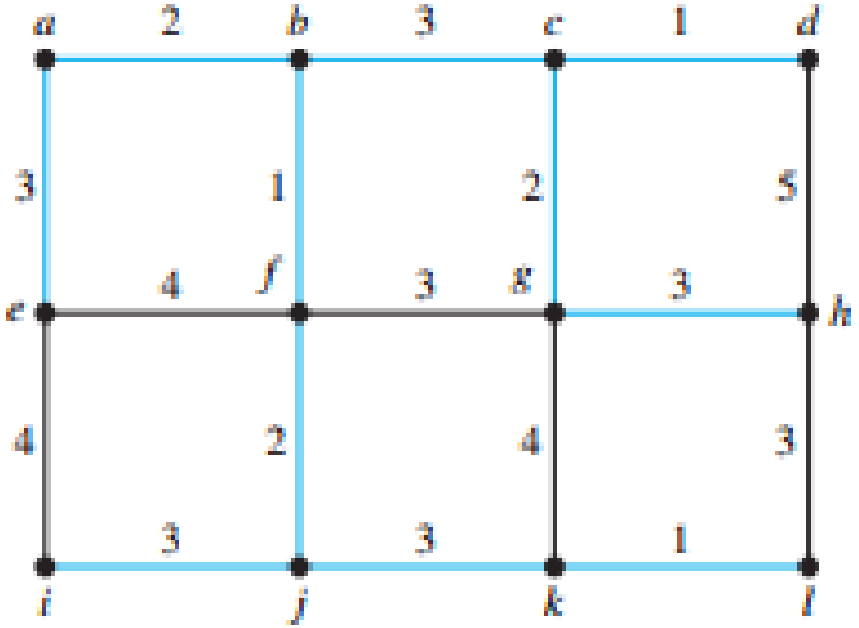
FIGURE 4 **A Minimum Spanning Tree Produced Using Prim's Algorithm.**

10.5- Minimum Spanning Trees

ALGORITHM 2 Kruskal's Algorithm.

procedure *Kruskal*(G: weighted connected undirected graph with n vertices)
T := empty graph
for $i := 1$ **to** $n - 1$
    e := any edge in G with smallest weight that does not form a simple circuit when added to T
    $T := T$ with e added
return T {T is a minimum spanning tree of G}

10.5- Minimum Spanning Trees



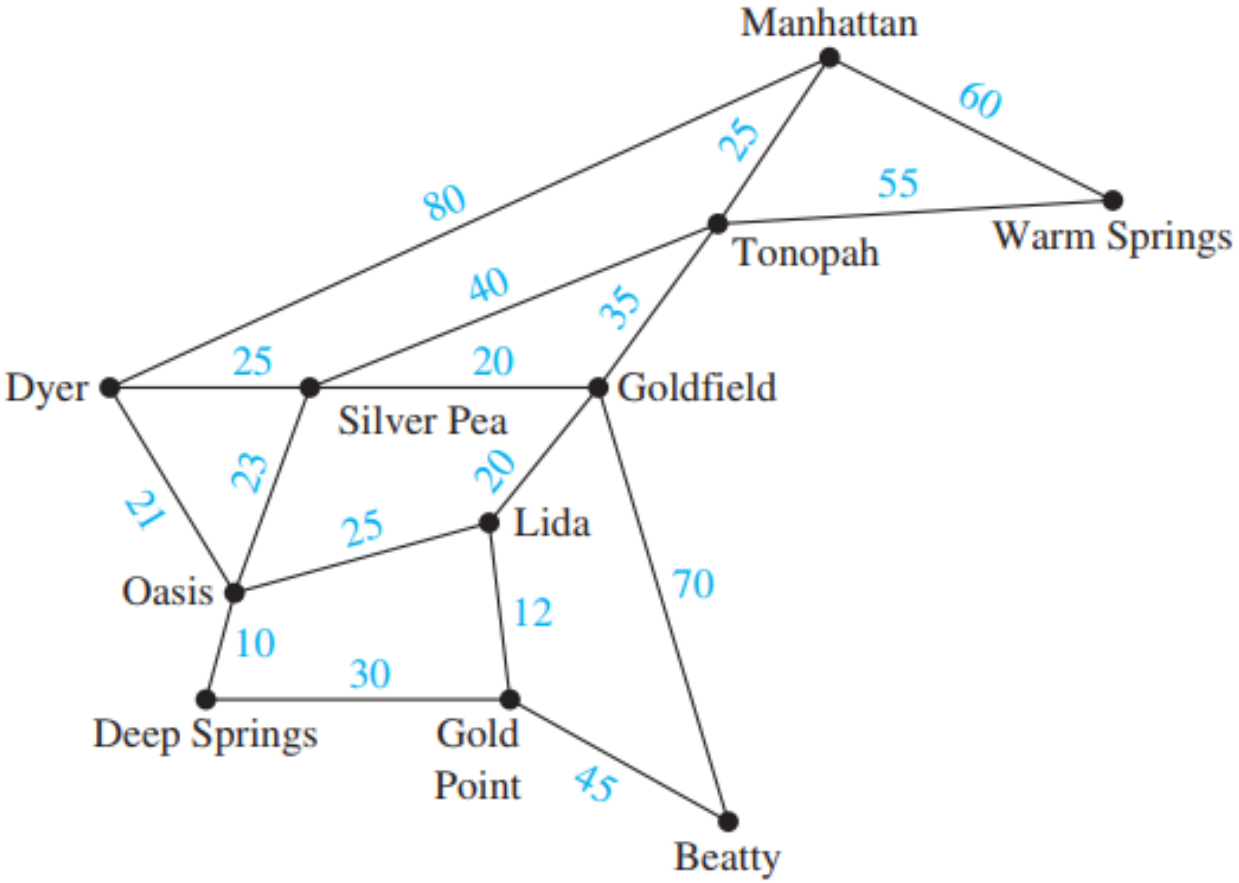
(a)

Choice	Edge	Weight
1	{c, d}	1
2	{k, l}	1
3	{b, f}	1
4	{c, g}	2
5	{a, b}	2
6	{f, j}	2
7	{b, c}	3
8	{j, k}	3
9	{g, h}	3
10	{i, j}	3
11	{a, e}	3
	Total:	24

(b)

FIGURE 5 A Minimum Spanning Tree Produced by Kruskal's Algorithm.

The roads represented by this graph are all unpaved. The lengths of the roads between pairs of towns are represented by edge weights. Which roads should be paved so that there is a path of paved roads between each pair of towns so that a minimum road length is paved?



Use Prim's algorithm, Kruskal's algorithm to find a minimum spanning tree for the given weighted graph.