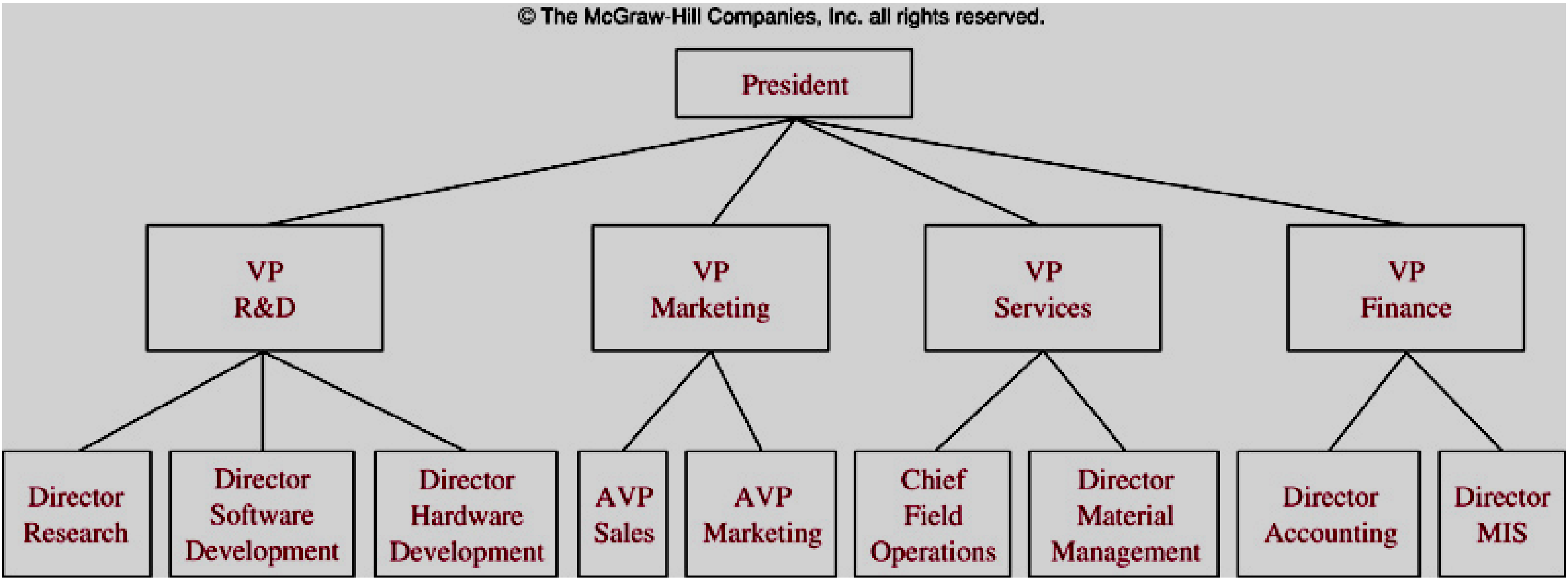
CHAPTER 10 - **TREES**

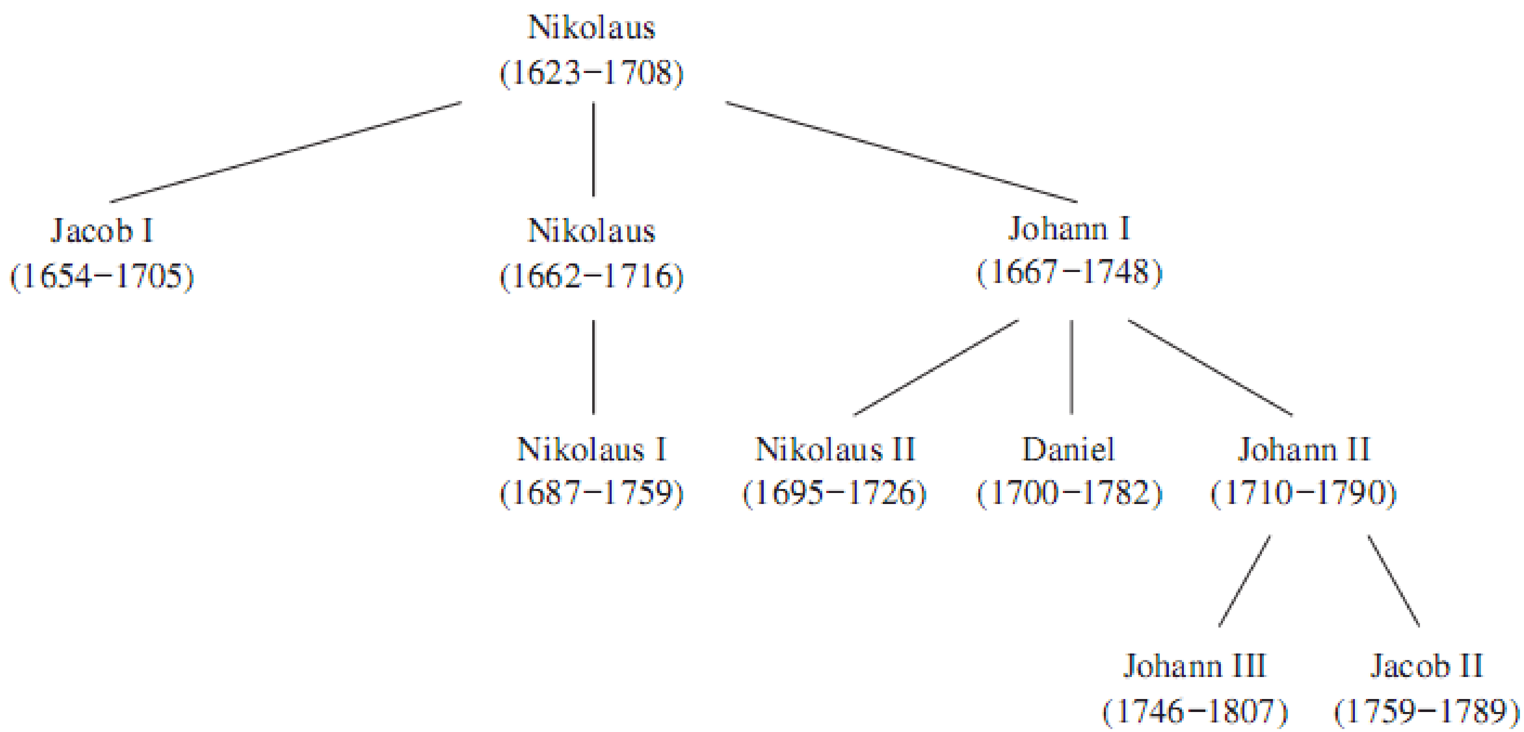
Our goal

Some Tree Models



A Organizational Tree

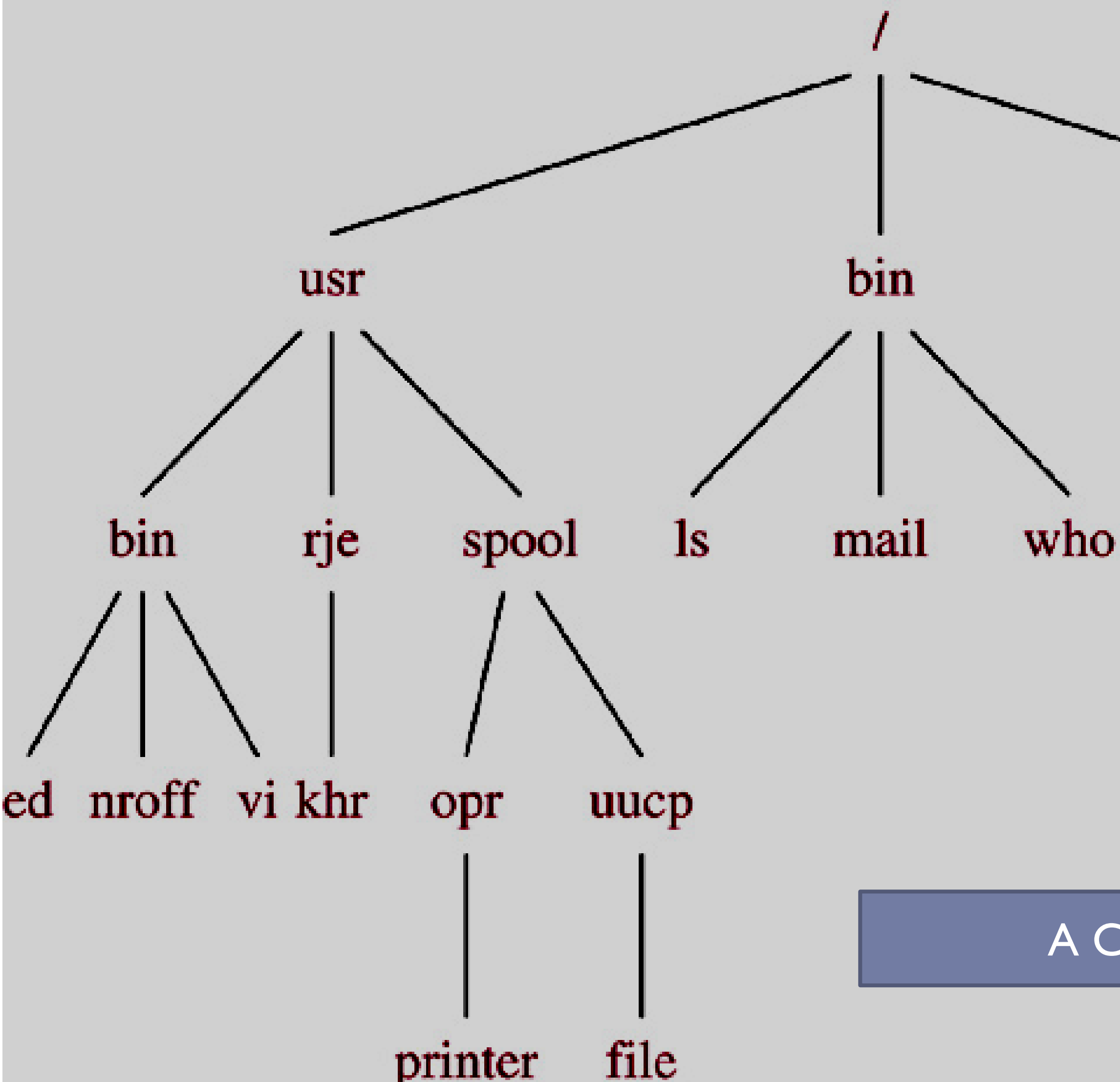
Family tree


Nikolaus
(1623–1708)
Jacob I
(1654–1705)
Nikolaus
(1662–1716)
Johann I
(1667–1748)
Nikolaus I
(1687–1759)
Nikolaus II
(1695–1726)
Daniel
(1700–1782)
Johann II
(1710–1790)
Johann III
(1746–1807)
Jacob II
(1759–1789)

The root is the root directory /

Internal vertices are directories

Leaves are files



A Computer File System

Introduction to Trees

DEFINITION 1

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

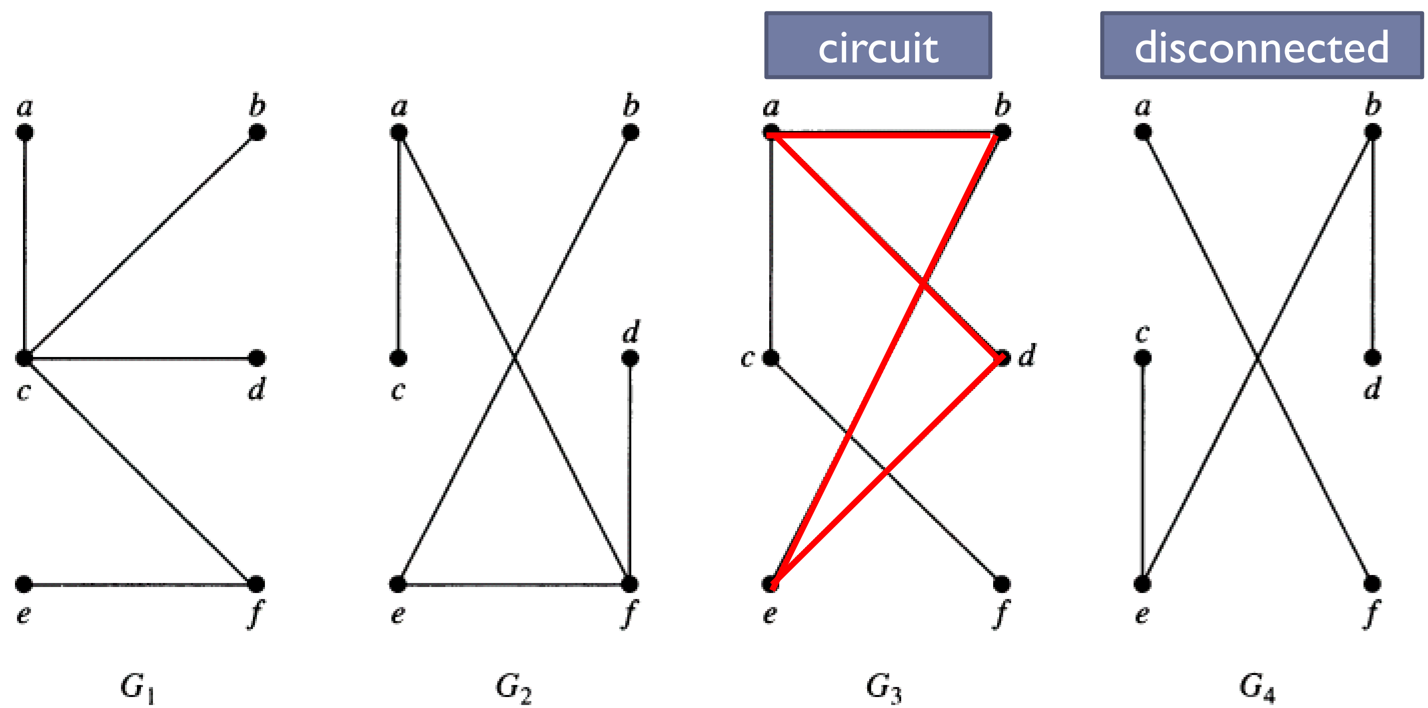
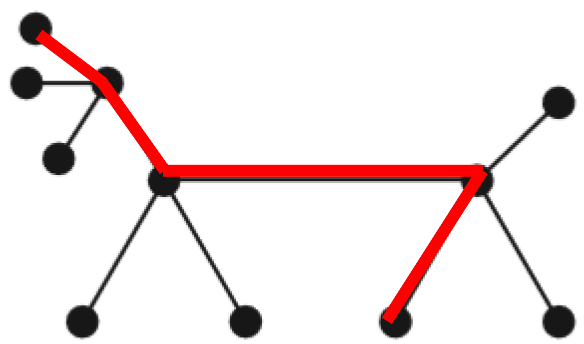


FIGURE 2 **Examples of Trees and Graphs That Are Not Trees.**

Trees

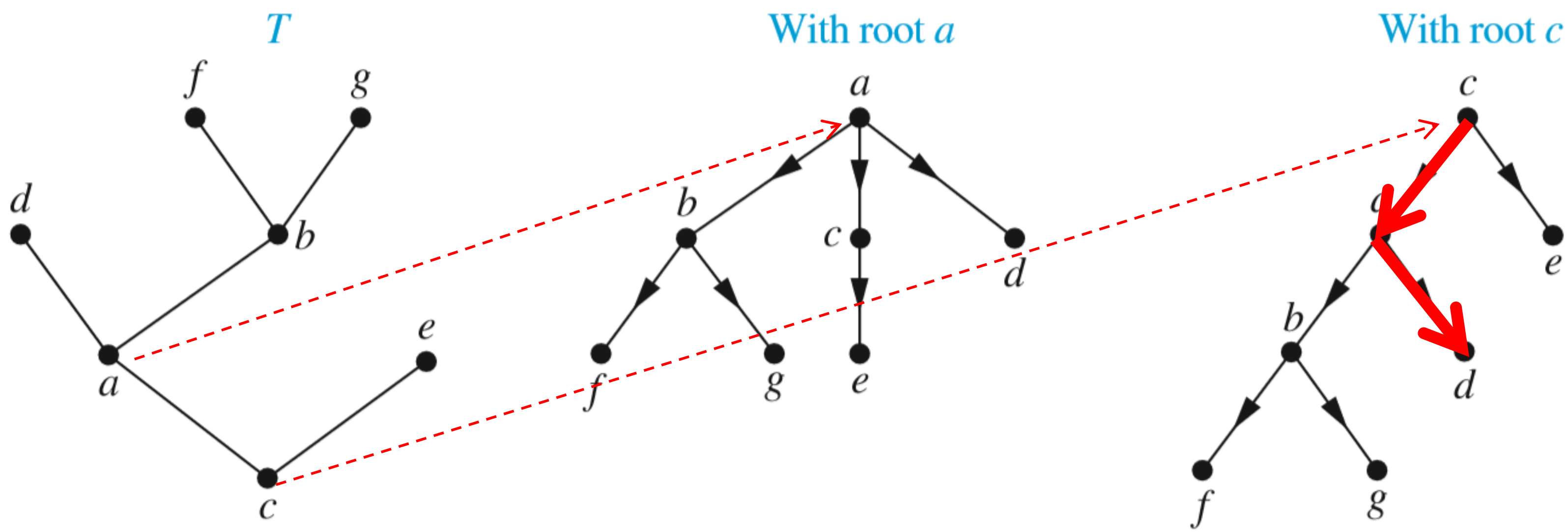
- **Theorem.**

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

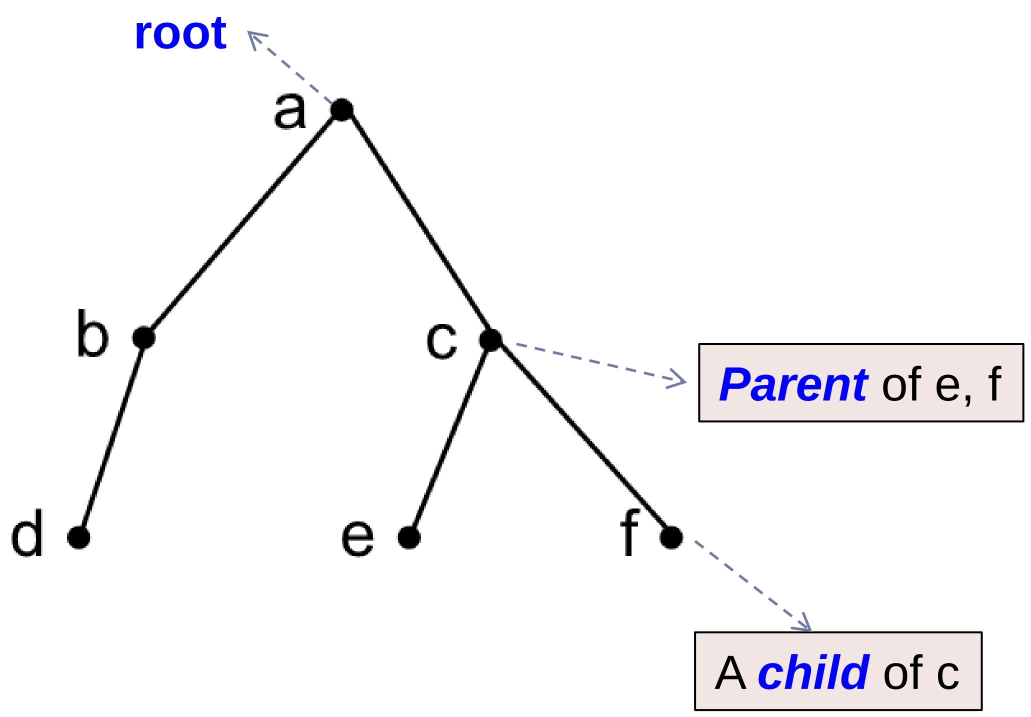


Rooted trees

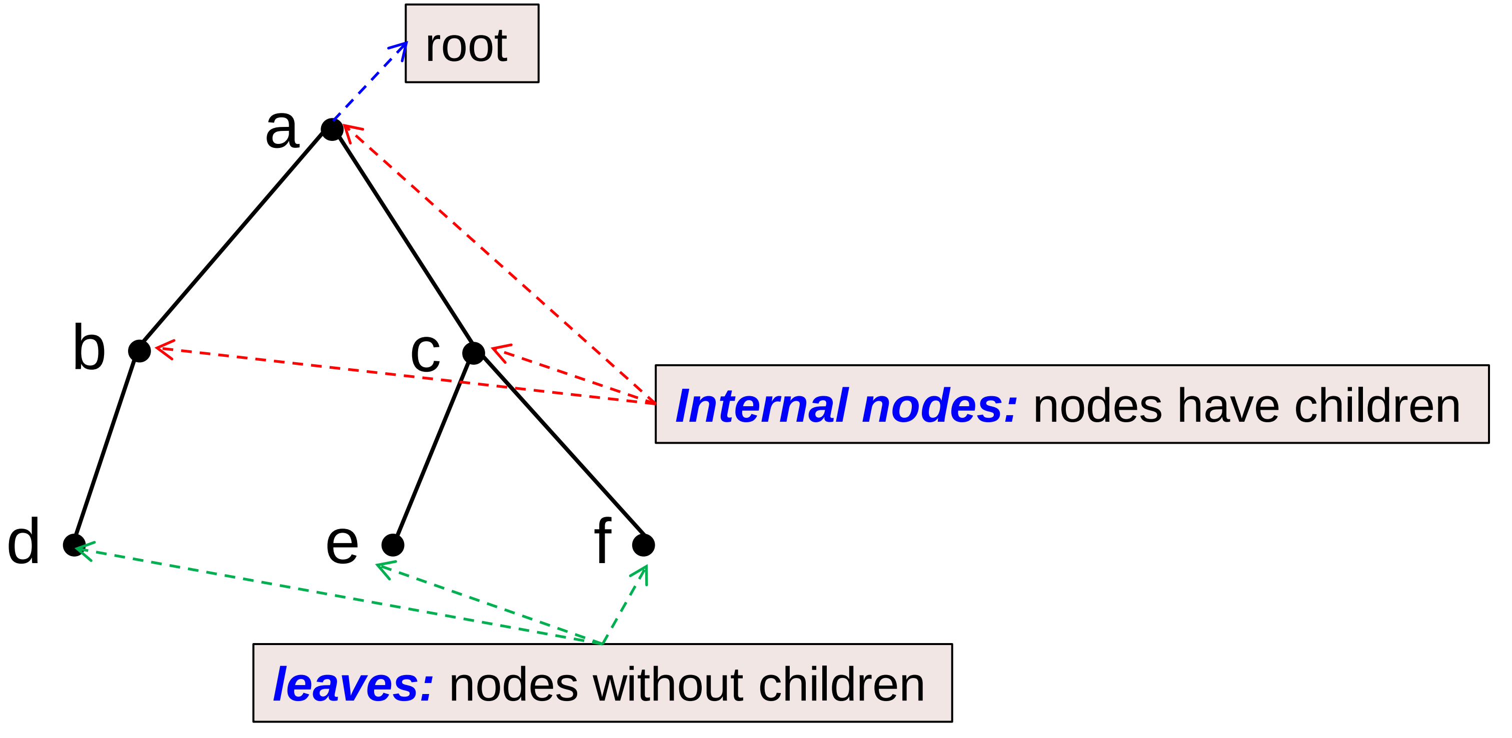
A ***rooted tree*** is a tree in which one vertex has been designated as the root



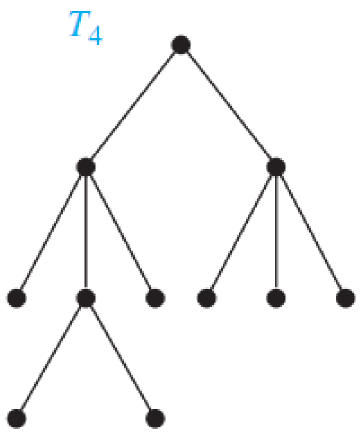
Terminologies



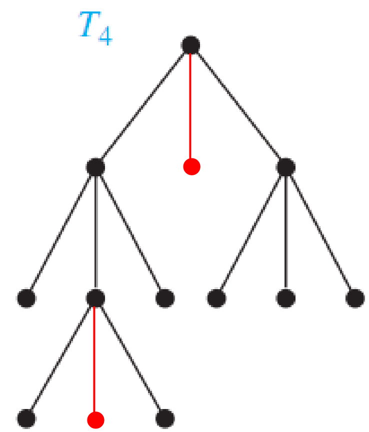
Terminologies



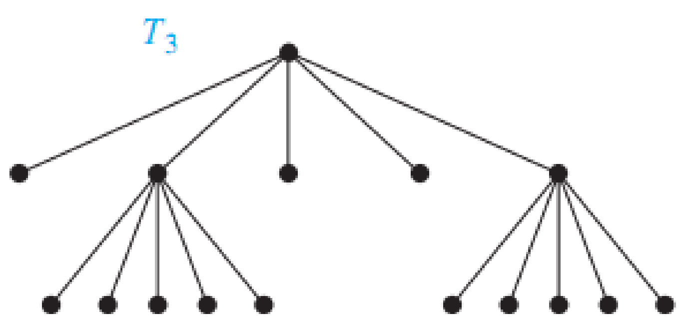
Full m-ary trees



m = 3: Ternary tree

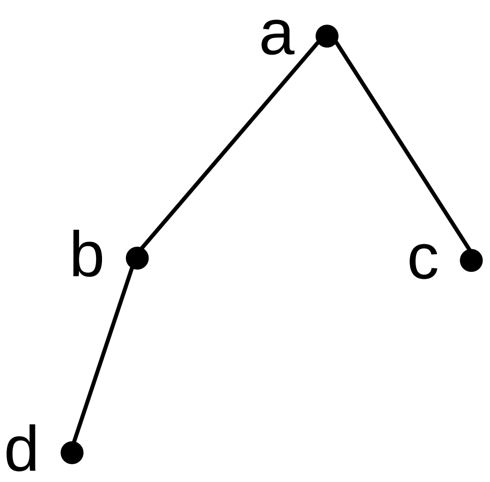


Full Ternary tree

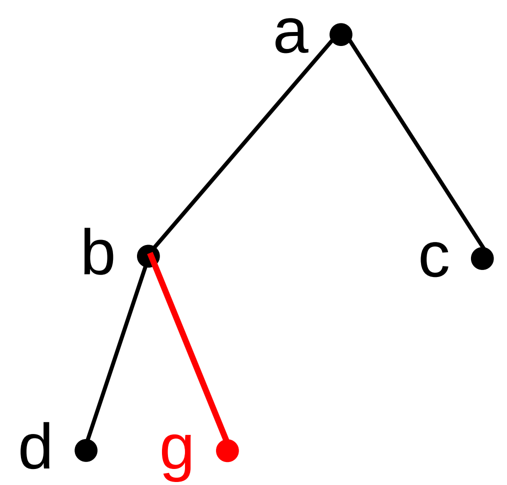


Full
5-ary tree

Full m-ary trees

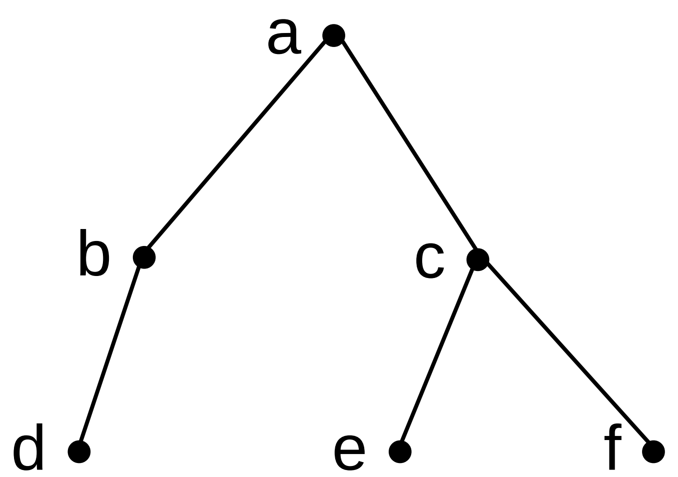


m=2: binary tree T

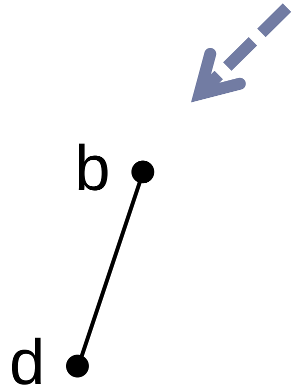


A **full** binary tree T

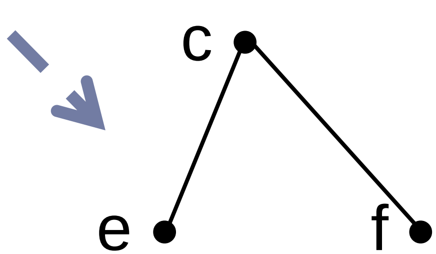
Binary tree



A *binary tree* T



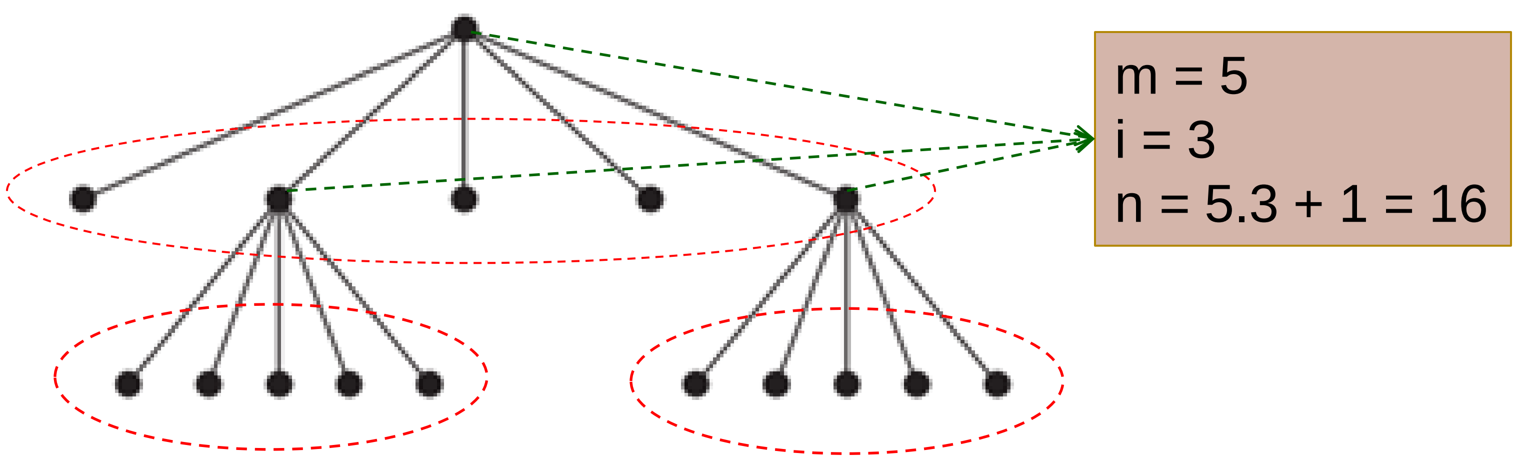
Left subtree of T



Right subtree of T

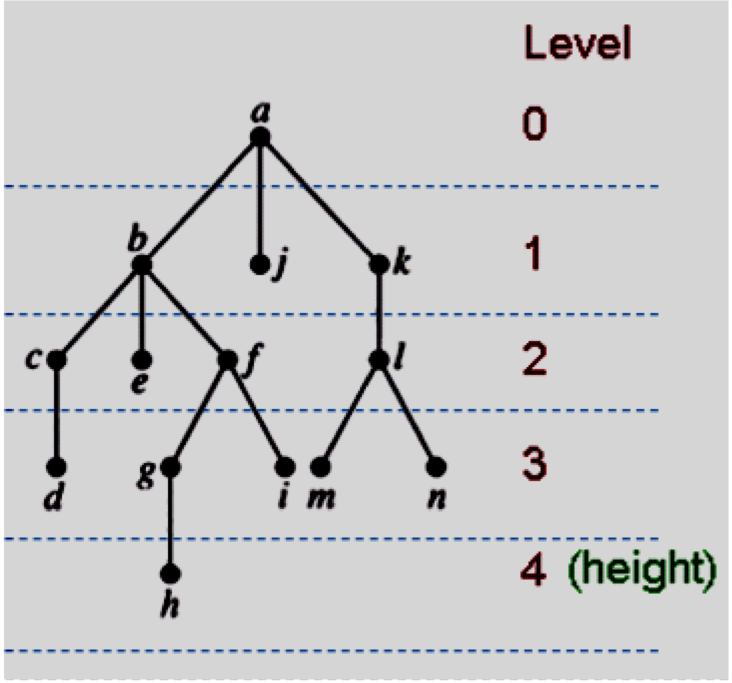
Properties of Trees

- A tree with ***n*** vertices has *n-1 edges* #n: Number of *nodes*
- For a **full m-ary** tree:
  - $n = mi + 1$ // i : Number of *internal nodes*
  - $n = i + \ell$ // ℓ : Number of *leaves*



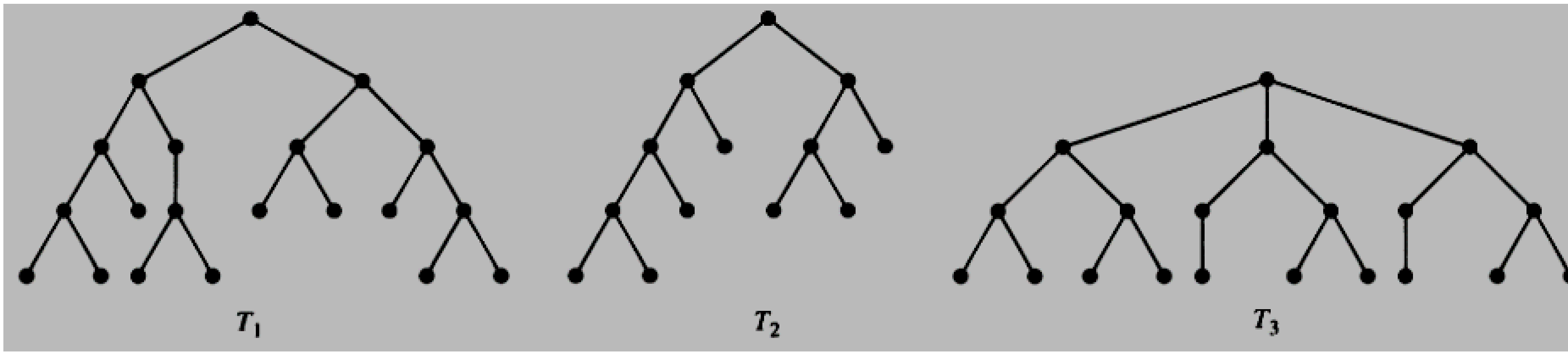
Introduction to Trees…

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



Introduction to Trees…

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



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

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

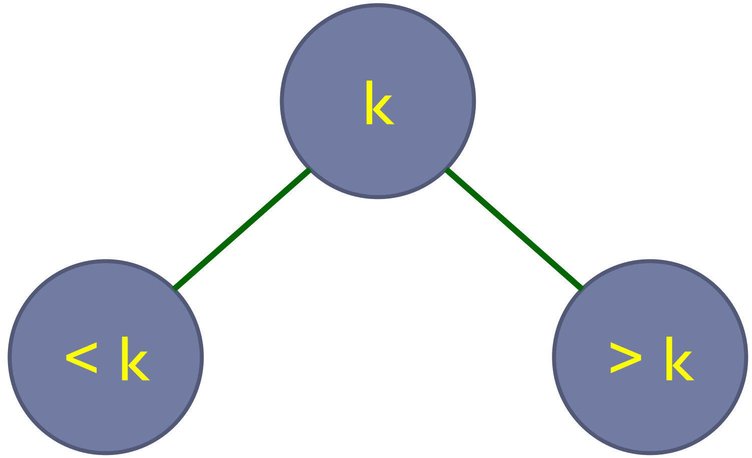
h=3
All leaves are at levels 3
→ Balanced

10.2- Applications of Trees

- Binary Search Trees
- Decision Trees
- Prefix Codes

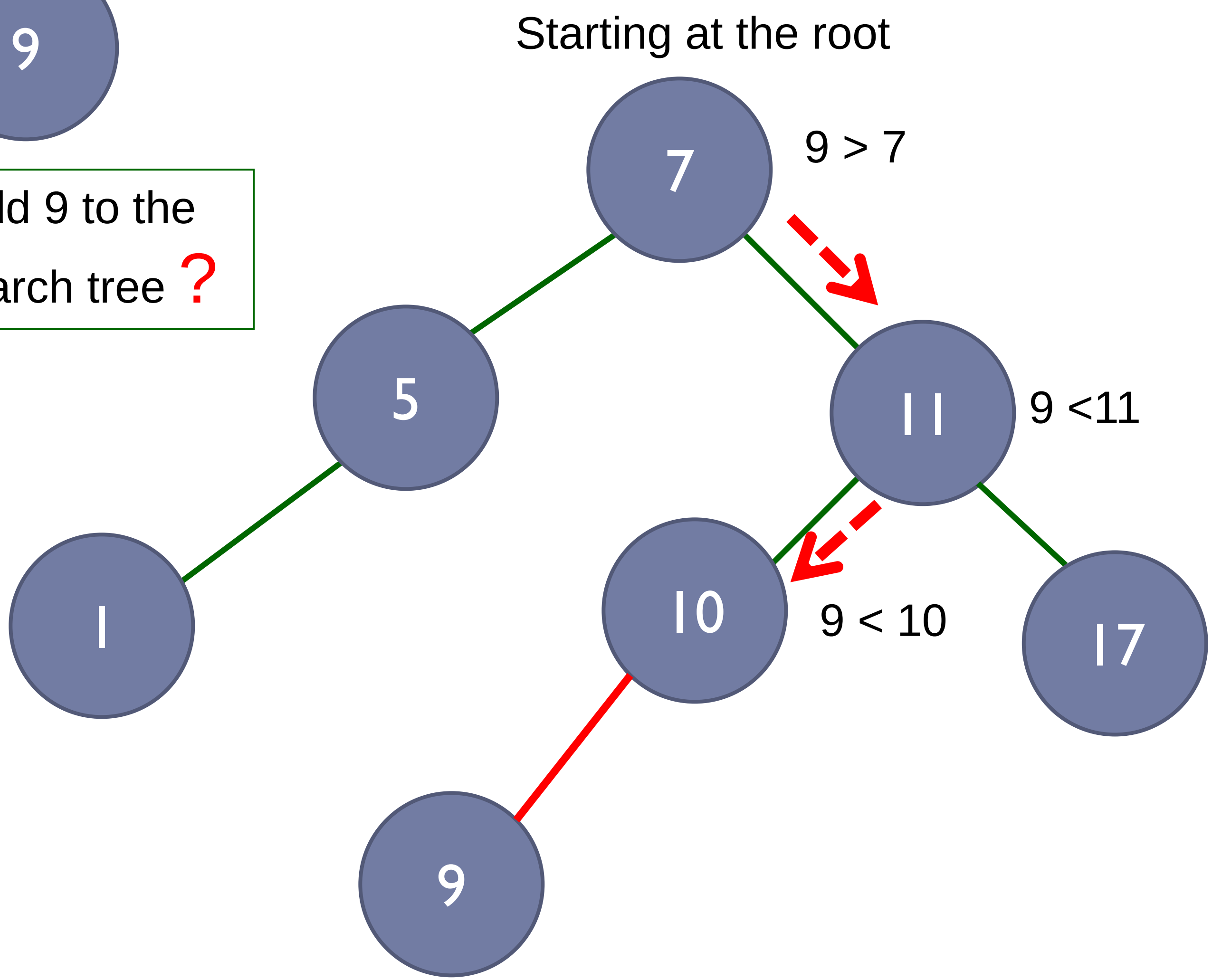
Binary search tree

- A binary tree where each vertex is labeled with a key
- the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.



Binary search tree - Add a new vertex

How to add 9 to the binary search tree ?

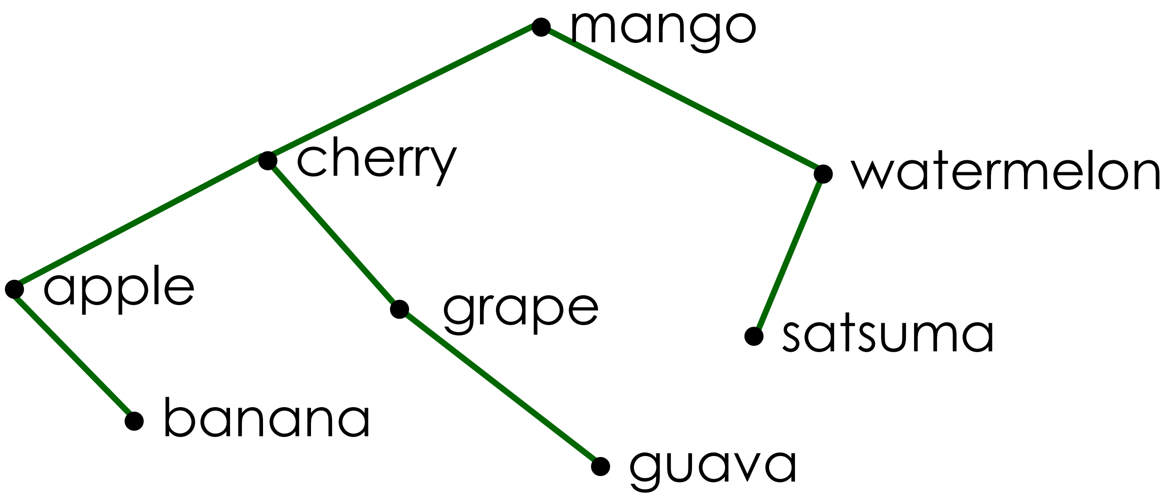


add 9 as a left child

Constructing a binary search tree

- Binary search tree for words: mango, cherry, grape, apple, guava, watermelon, satsuma, banana

smaller key → go/add to the left
greater key → go/add to the right



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}

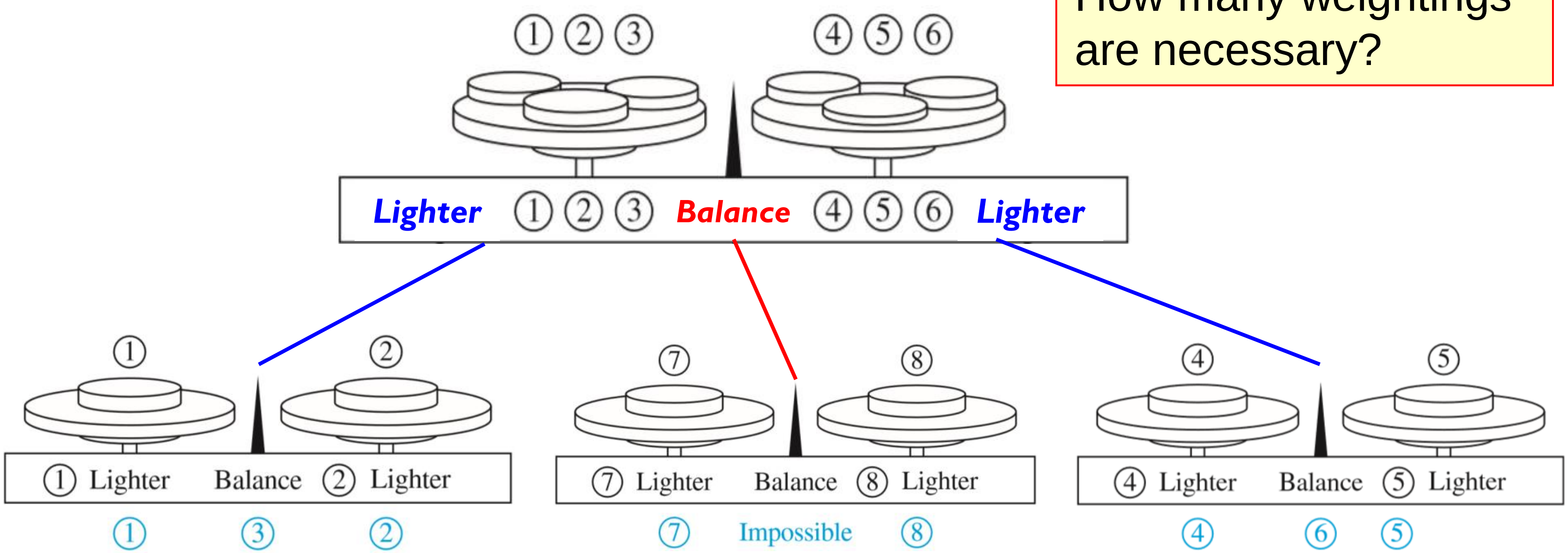
Complexity: O(logn)
Proof: page 698

Decision Trees

The Counterfeit Coin Problem.

- 8 coins, *1 fake coin* (counterfeit, lighter) ①②③④⑤⑥⑦⑧
- Use a two-pan ***balance scale*** to determine the ***lighter one***.

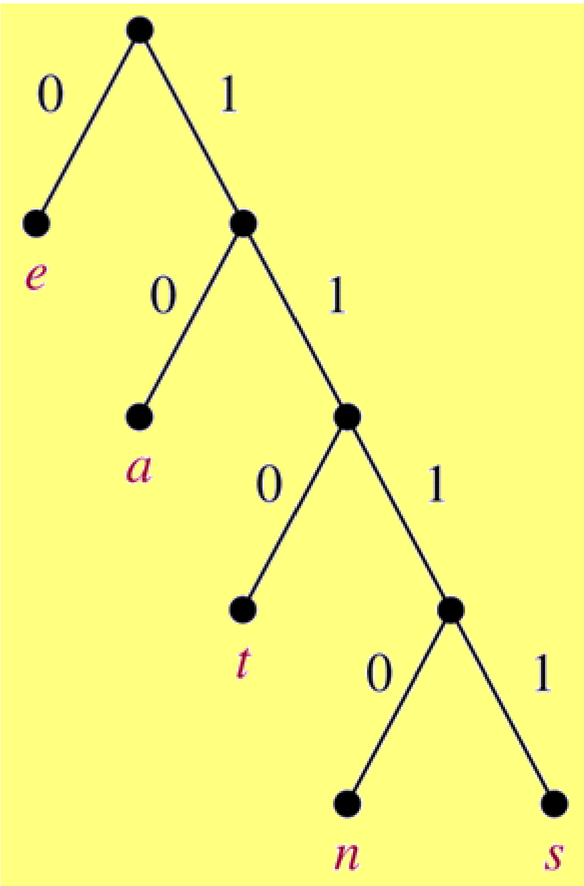
How many weightings are necessary?



Prefix Codes

- Introduction to Prefix Codes
- Huffman Coding Algorithm

Prefix Codes



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

  11111011100 ➔ 11 bits

11111011100 : s

11111011100 : a

11111011100 : n

11111011100 : e

ASCII uses 32 bits to store sane

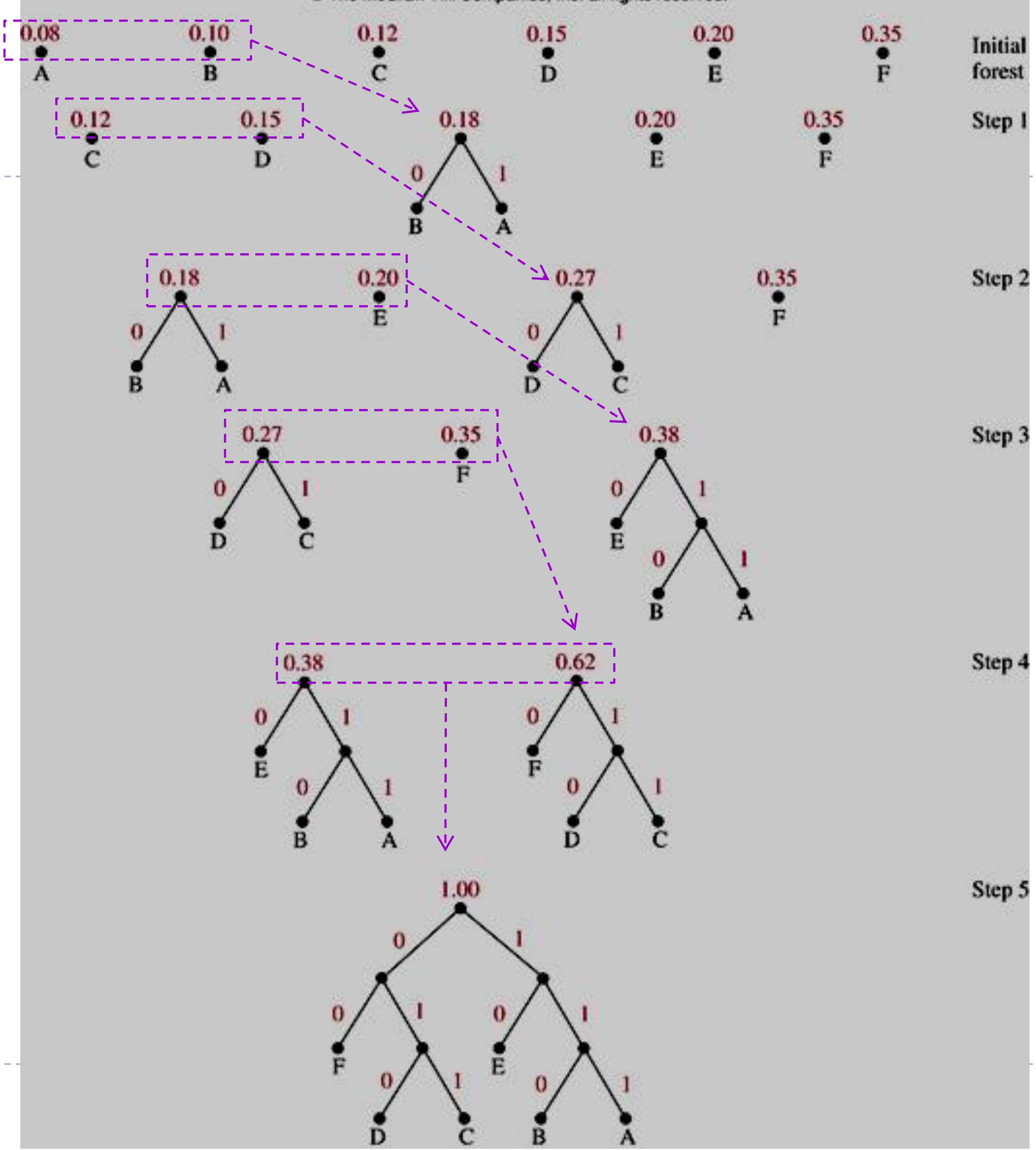
➔ Compression factor: 32/11 ~ 3

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.

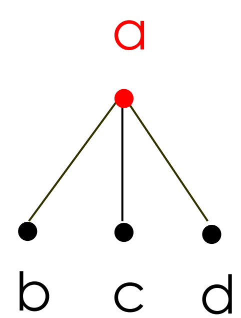
procedure *Huffman*(C: symbols a_i with frequencies w_i, $i = 1, \ldots, n$)
F := forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
while F is not a tree
begin
  Replace the rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree having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

Traversal Algorithms

- At a time, a vertex is **visited**
- Recursive algorithm
- Traversals are classified into:
  - Pre-order traversal. N L R
  - In-order traversal. L N R
  - Post-order traversal. L R N

// N: root node
// L: left subtree
// R: right subtree

Tree traversals



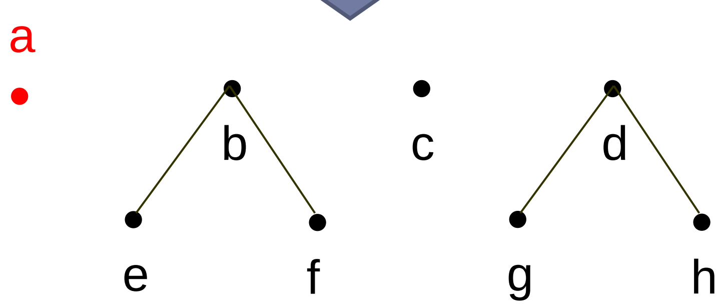
Pre-order traversal:
N L R
a b c d

in-order traversal:
L N R
b a c d

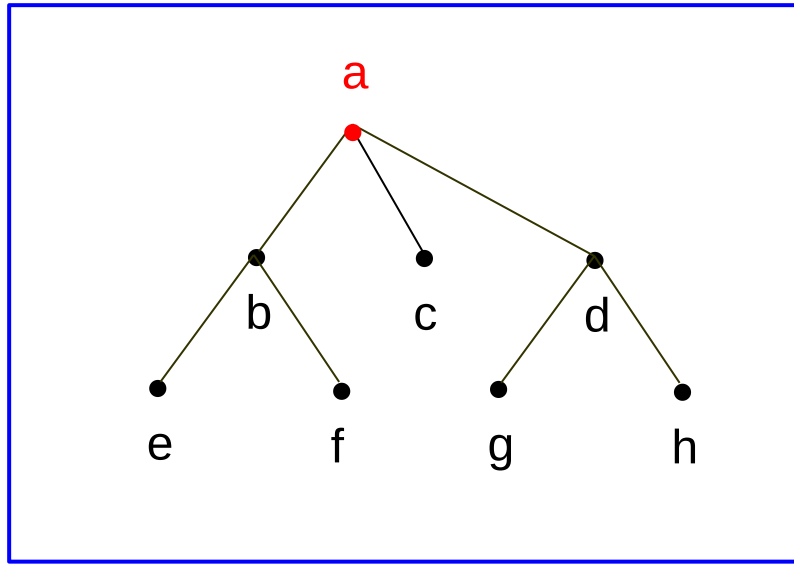
post-order traversal:
L R N
b c d a

Preorder traversal - example

Pre-order traversal: N L R



a b e f c d g h

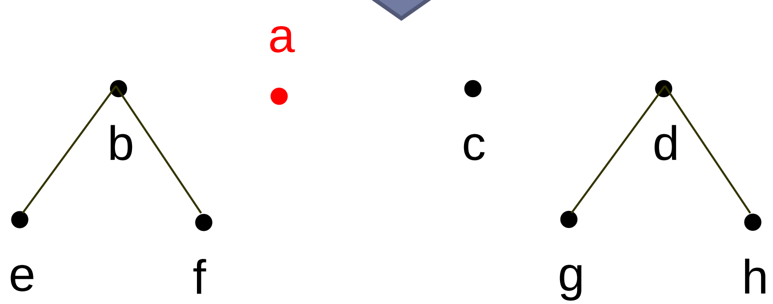


ALGORITHM 1 Preorder Traversal.

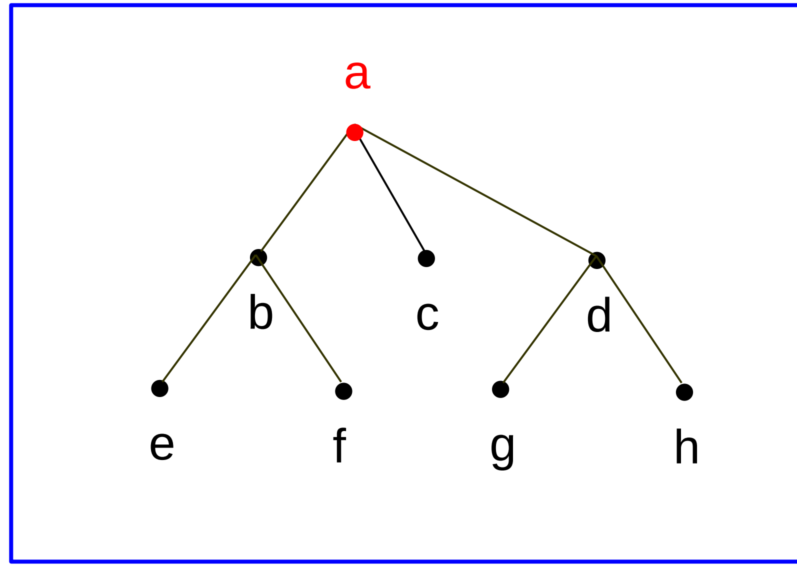
procedure *preorder*(T: ordered rooted tree)
r := root of T
list r
for each child c of r from left to right
begin
  $T(c)$:= subtree with c as its root
  preorder($T(c)$)
end

Inorder traversal - example

in-order traversal: L N R



e b f a c g d h

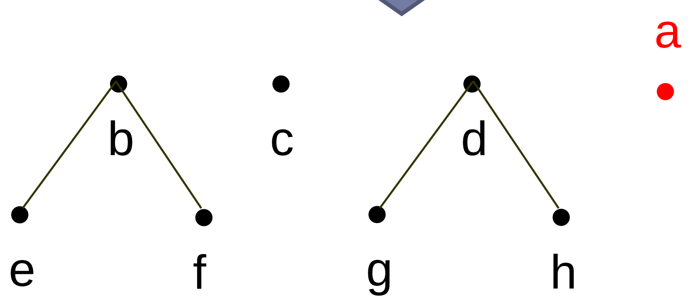


ALGORITHM 2 Inorder Traversal.

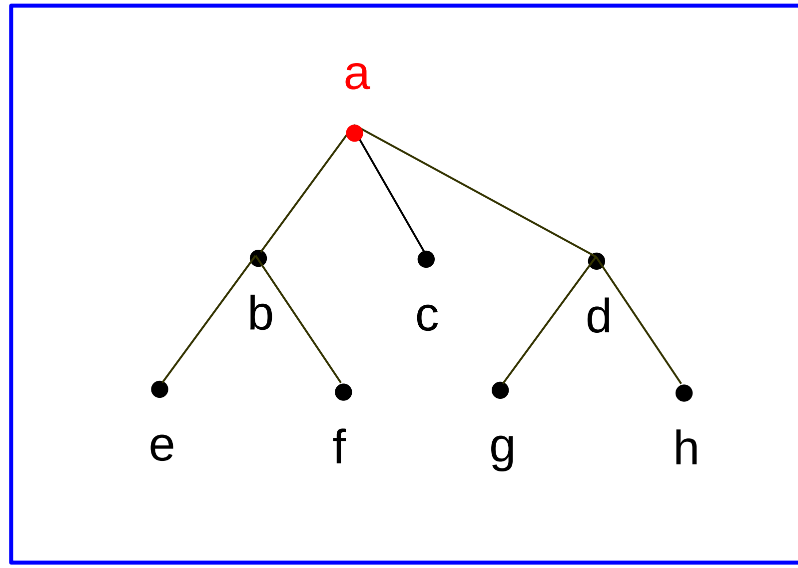
procedure *inorder*(T: ordered rooted tree)
r := root of T
if r is a leaf **then** list r
else
begin
  l := first child of r from left to right
  $T(l)$:= subtree with l as its root
  inorder($T(l)$)
  list r
  for each child c of r except for l from left to right
    $T(c)$:= subtree with c as its root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

Postorder traversals - examples

post-order traversal: L R N



e f b c g h d a

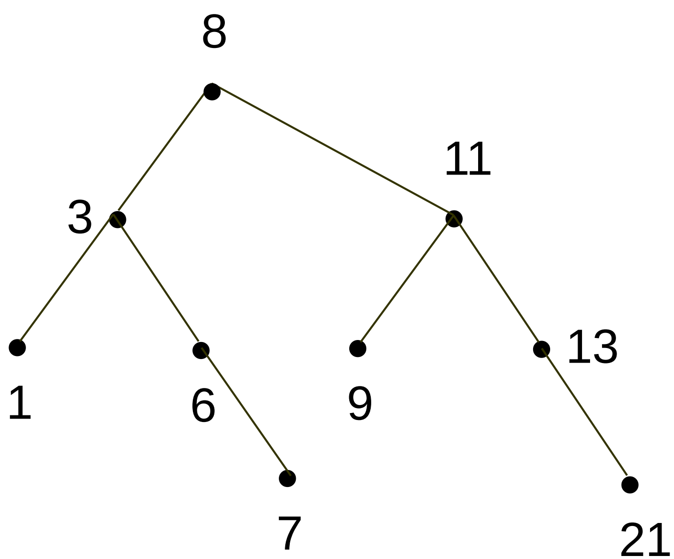


ALGORITHM 3 Postorder Traversal.

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

Tree traversals - examples

Construct a **binary search tree** for the numbers:
8, 11, 3, 6, 9, 1, 13, 7, 21
What are the order of numbers after applying:
- preorder traversal
- inorder traversal
- postorder traversal?



Binary search tree

Preorder traversal:
8 3 1 6 7 11 9 13 21

Inorder traversal:
1 3 6 7 8 9 11 13 21

Sorted

Postorder traversal:
1 7 6 3 9 21 13 11 8

Infix, Prefix, and Postfix Notation

Expression Trees

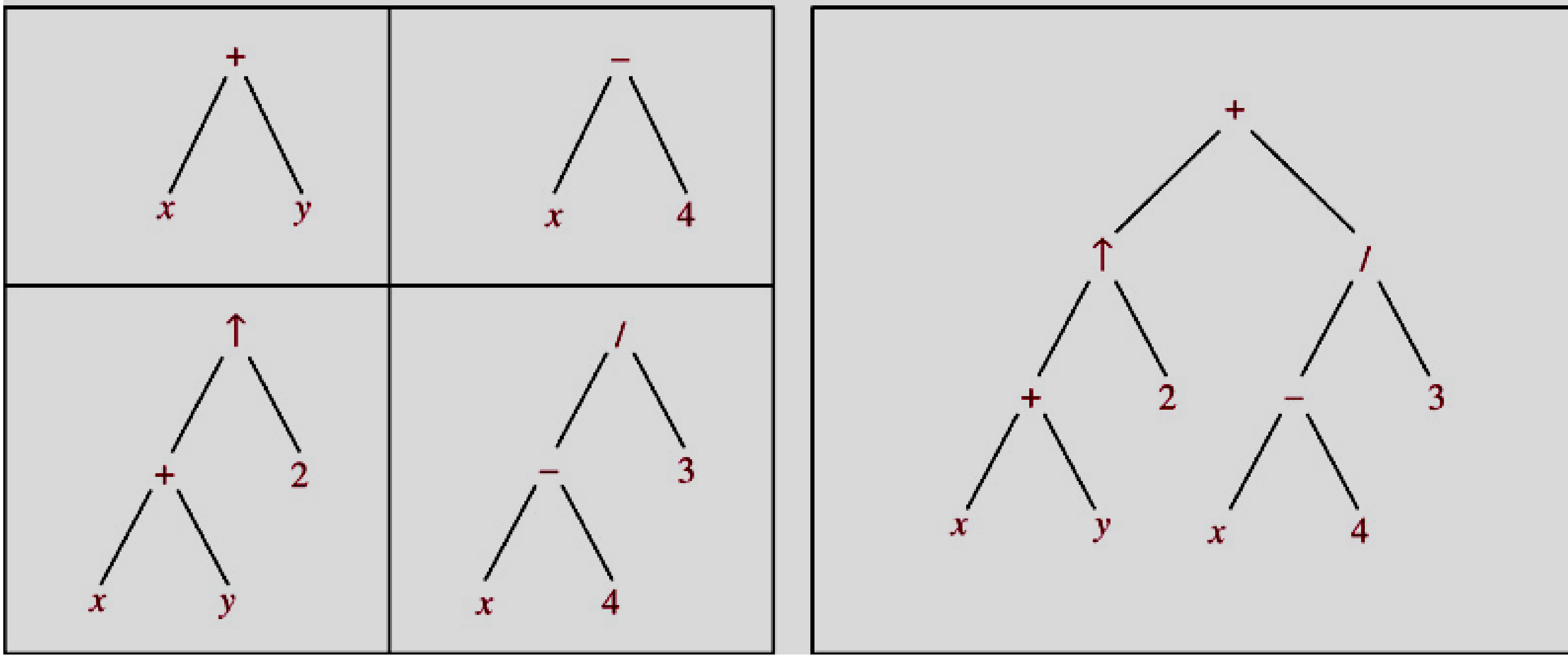


FIGURE 10 A Binary Tree Representing $((x + y) \uparrow 2) + ((x - 4)/3)$.

Infix, Prefix, and Postfix Notation

- Expression Trees

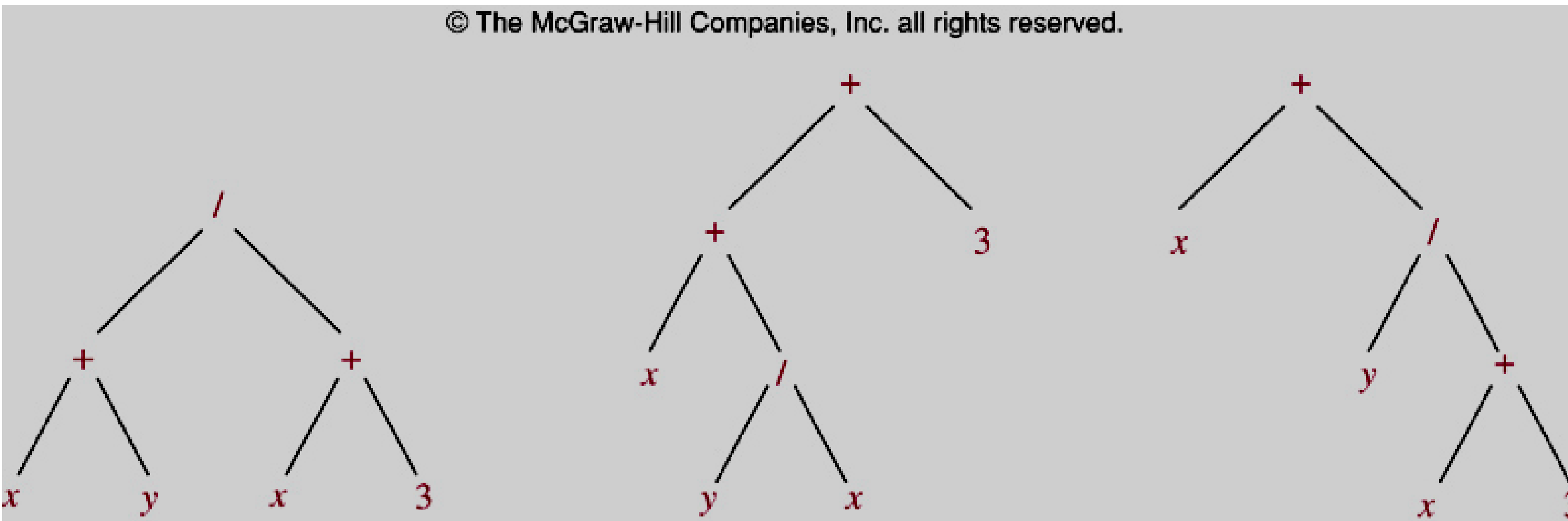


FIGURE 11 **Rooted Trees Representing $(x + y)/(x + 3)$, $(x + (y/x)) + 3$, and $x + (y/(x + 3))$.**

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:**

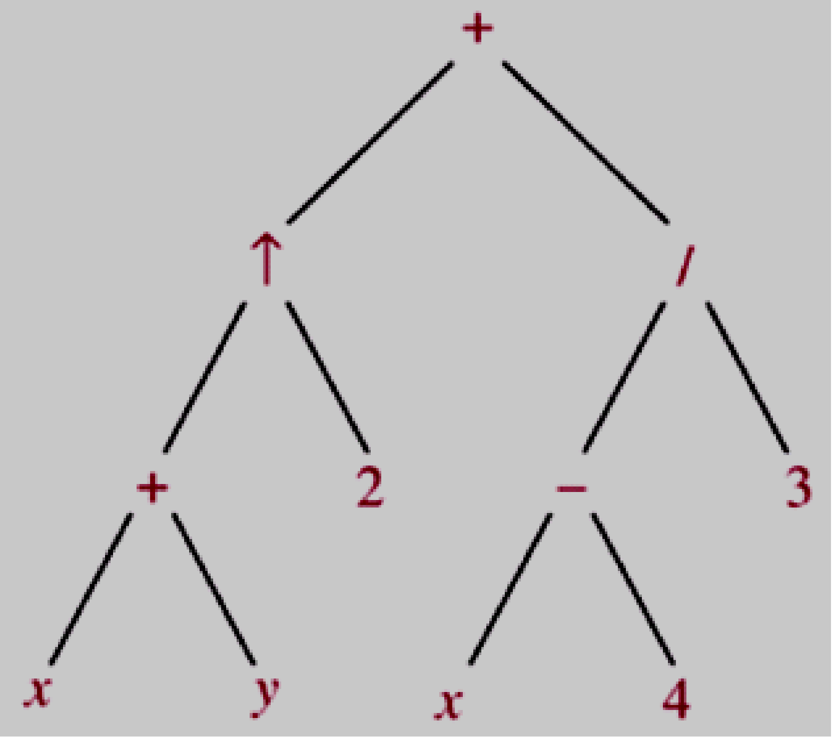
  (operand_1,operand_2)operator x y +
- How to find prefix and postfix form from infix form?

  (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

$$+\ -\ *\ 2\ 3\ 5\ /\ \underbrace{\uparrow\ 2\ 3}_{2 \uparrow 3 = 8}\ 4$$

$$+\ -\ *\ 2\ 3\ 5\ \underbrace{/\ 8\ 4}_{8 / 4 = 2}$$

$$+\ -\ \underbrace{*\ 2\ 3}_{2\quad 3 = 6}\ 5\ 2$$

$$+\ \underbrace{-\ 6\ 5}_{6 - 5 = 1}\ 2$$

$$\underbrace{+\ 1\ 2}_{1 + 2 = 3}$$

Value of expression 3

FIGURE 12 Evaluating a Prefix Expression.

$$7\ \underbrace{2\ 3\ *}_{2 * 3 = 6}\ -\ 4\ \uparrow\ 9\ 3\ /\ +$$

$$\underbrace{7\ 6\ -}_{7 - 6 = 1}\ 4\ \uparrow\ 9\ 3\ /\ +$$

$$\underbrace{1\ 4\ \uparrow}_{1^4 = 1}\ 9\ 3\ /\ +$$

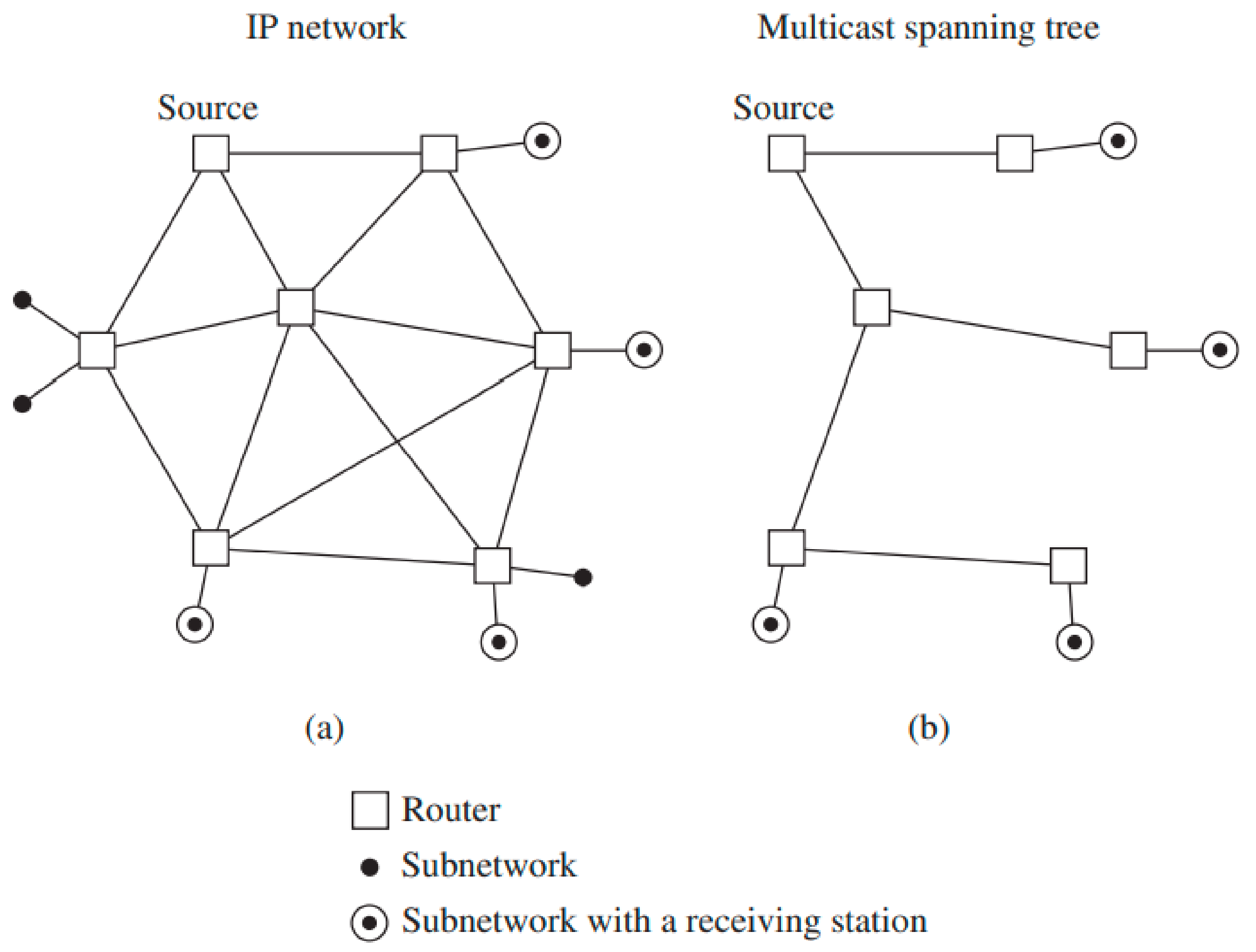
$$1\ \underbrace{9\ 3\ /}_{9 / 3 = 3}\ +$$

$$\underbrace{1\ 3\ +}_{1 + 3 = 4}$$

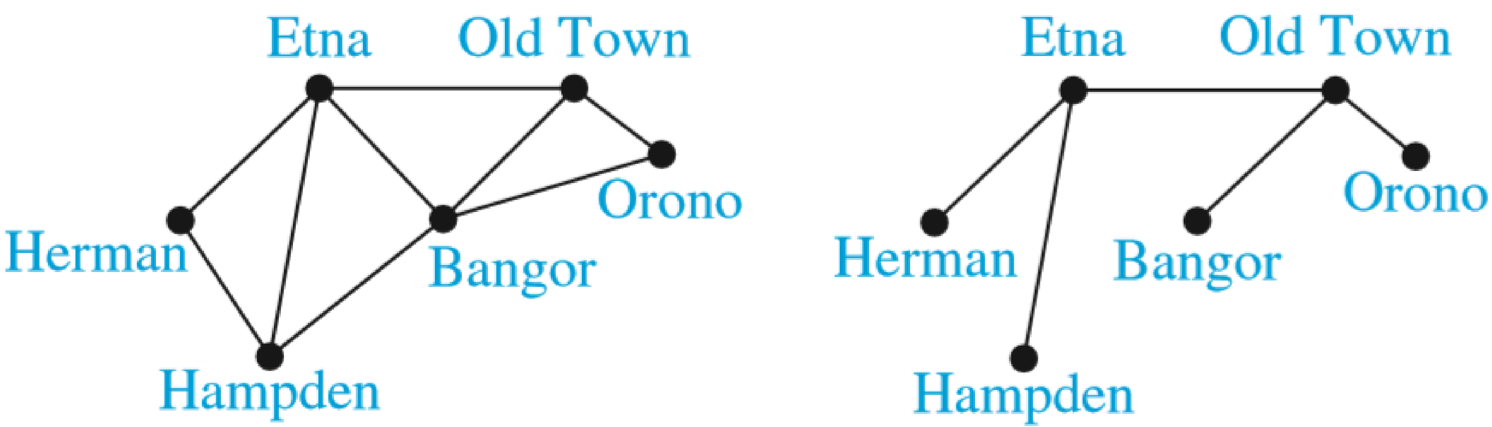
Value of expression: 4

FIGURE 13 Evaluating a Postfix Expression.

Spanning trees



Spanning trees



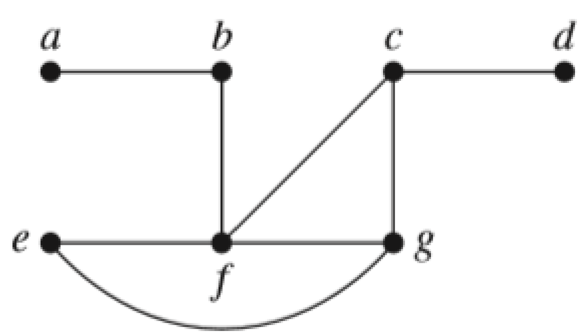
(a) (b)

Spanning trees

Definition.

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.

EXAMPLE 1 Find a spanning tree of the simple graph G shown in Figure 2.



Edge removed: $\{a, e\}$

(a)

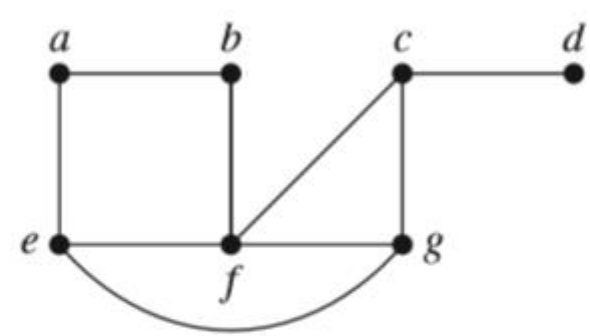
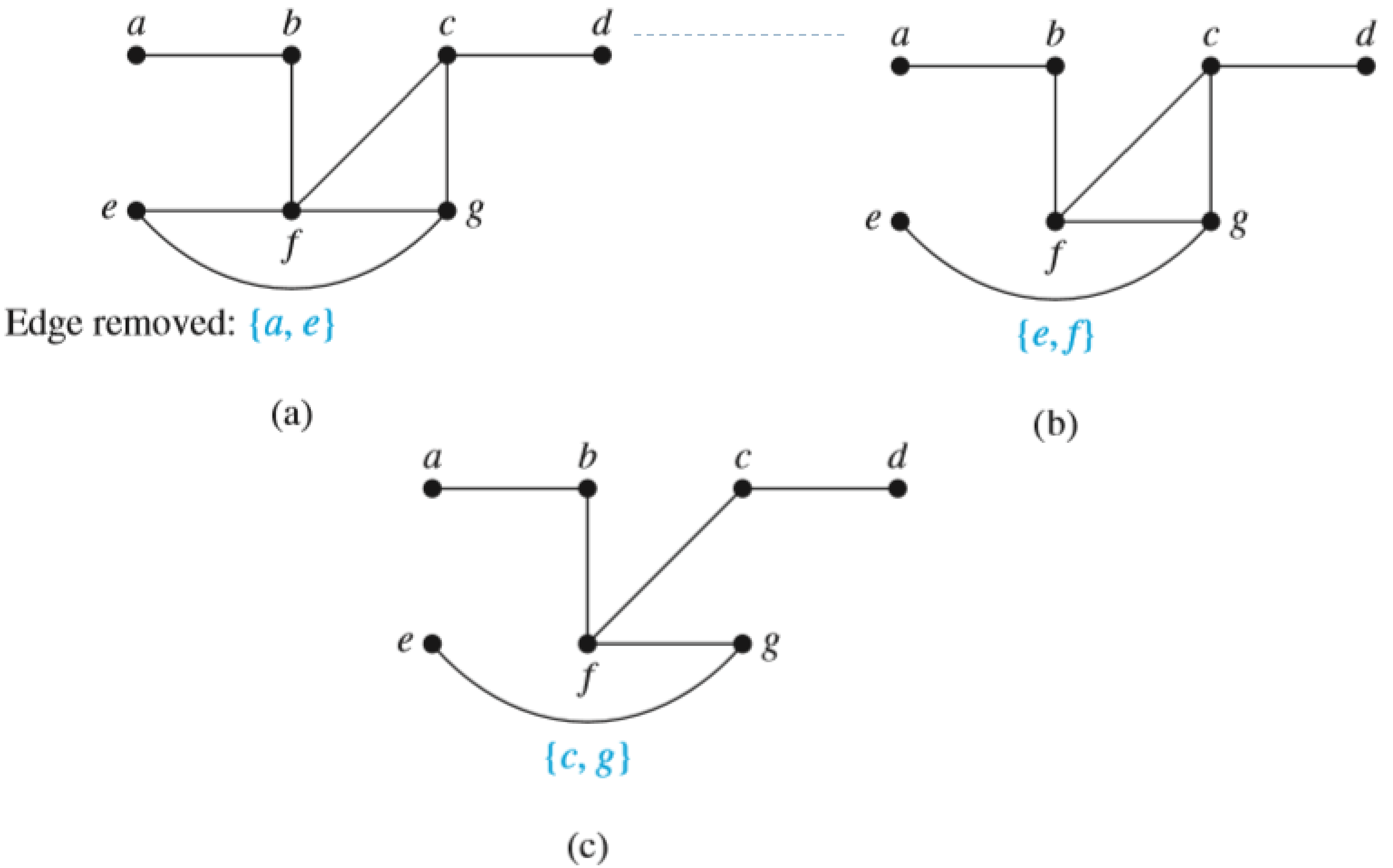


FIGURE 2 The Simple Graph G.



Remov EdgesThat Form Simple Circuits.

THEOREM 1

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

Depth-First Search

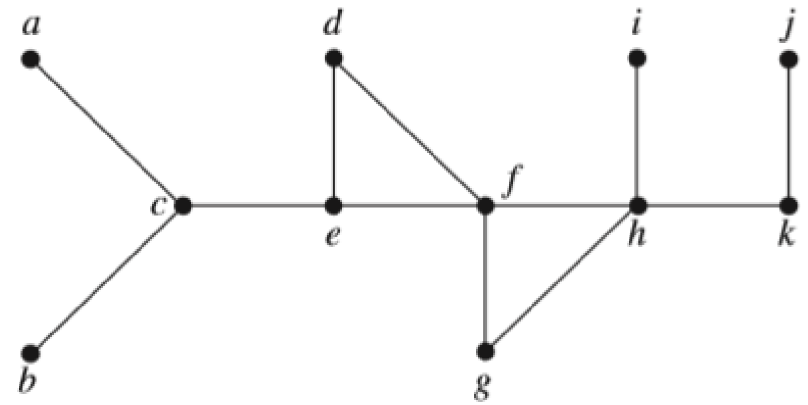


FIGURE 6 The Graph G.

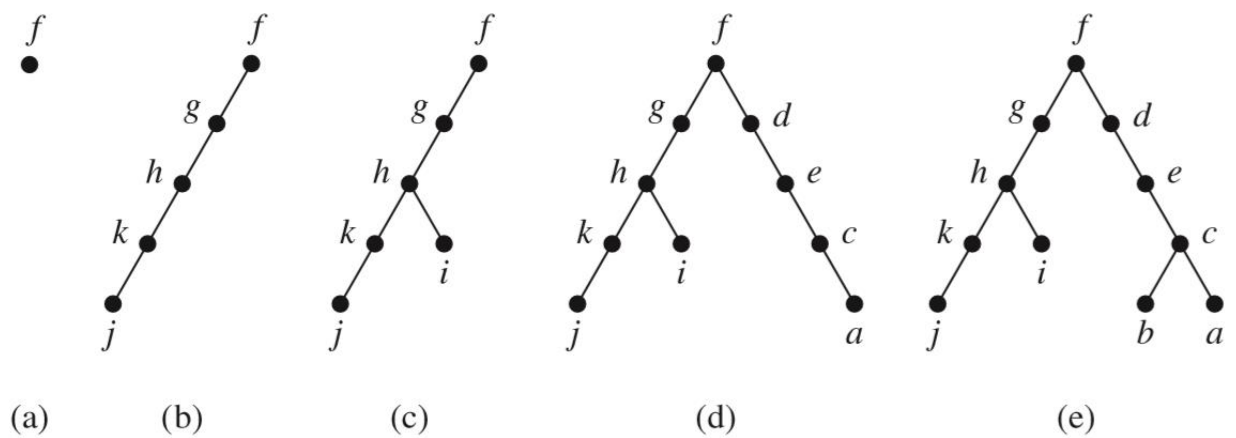
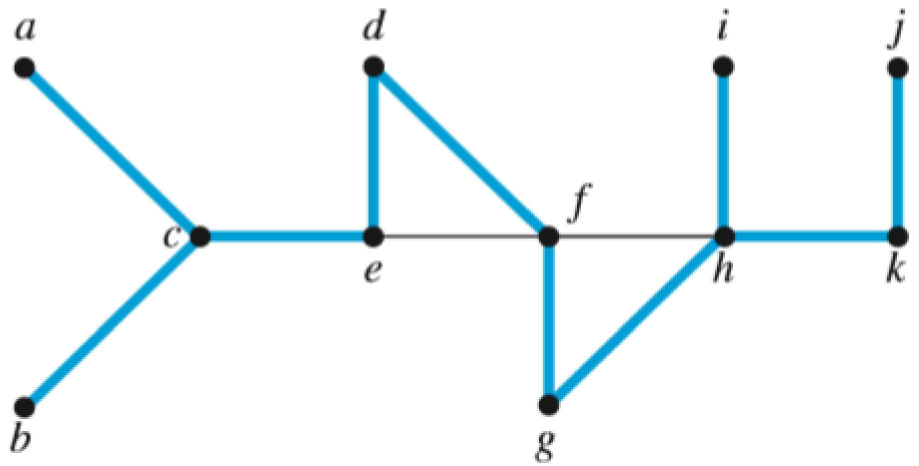


FIGURE 7 Depth-First Search of G.



The edges selected by depth-first search of a graph are called tree edges.
All other edges are called back edges.

FIGURE 8 **The Tree Edges and Back Edges of the Depth-First Search in Example 4.**

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
    $visit(w)$

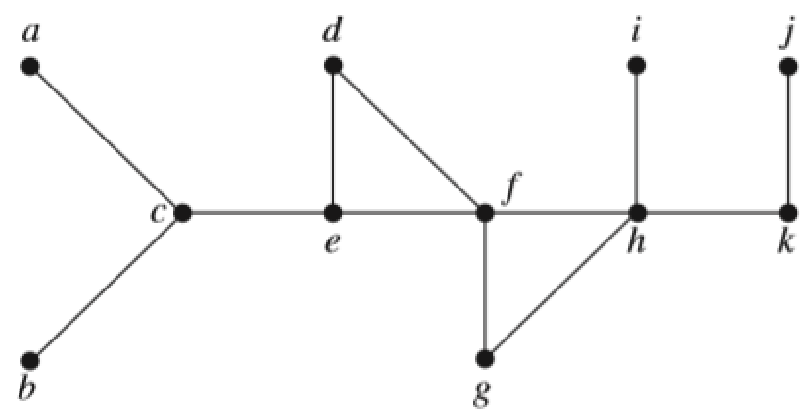


FIGURE 6 **The Graph G.**

Backtracking Applications

- Graph Colorings
- Then-Queens Problem
- Sums of Subsets

Graph Colorings - example

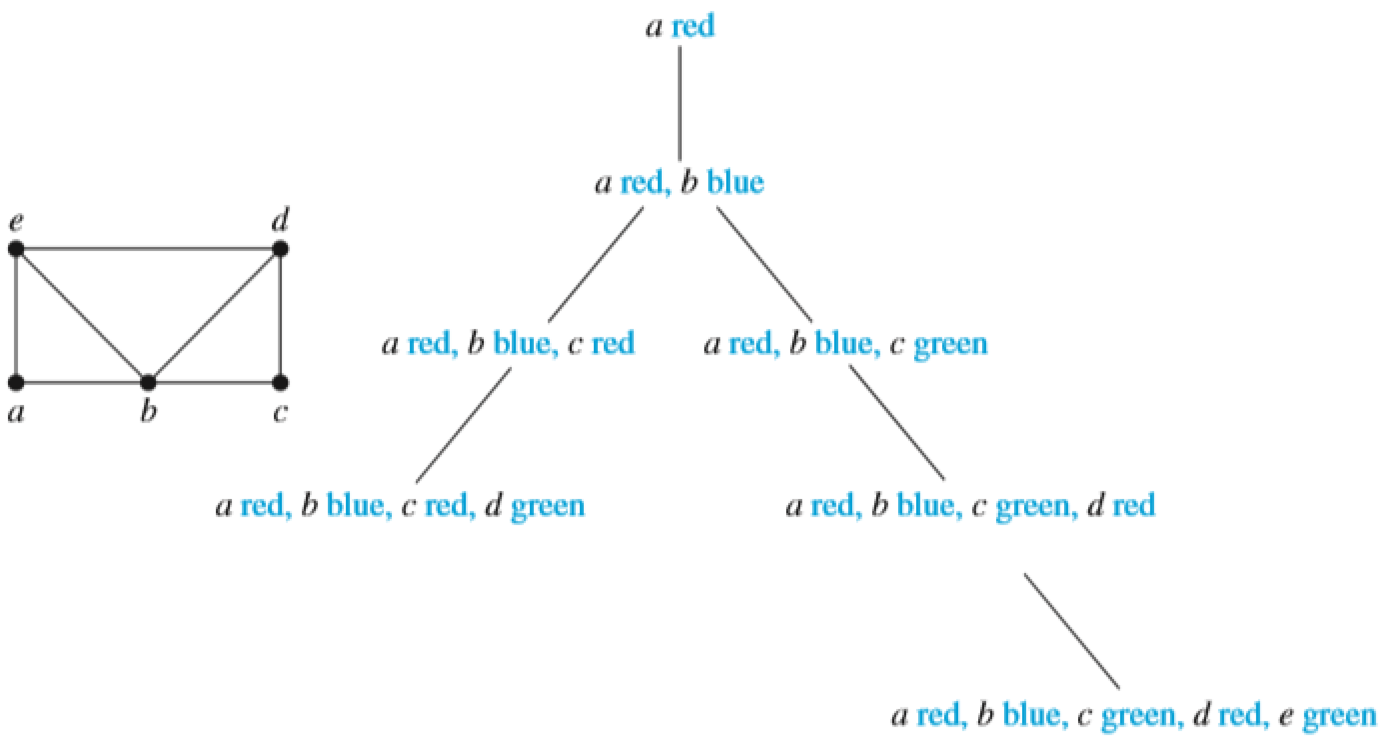


FIGURE 11 **Coloring a Graph Using Backtracking.**

The n-Queens Problem

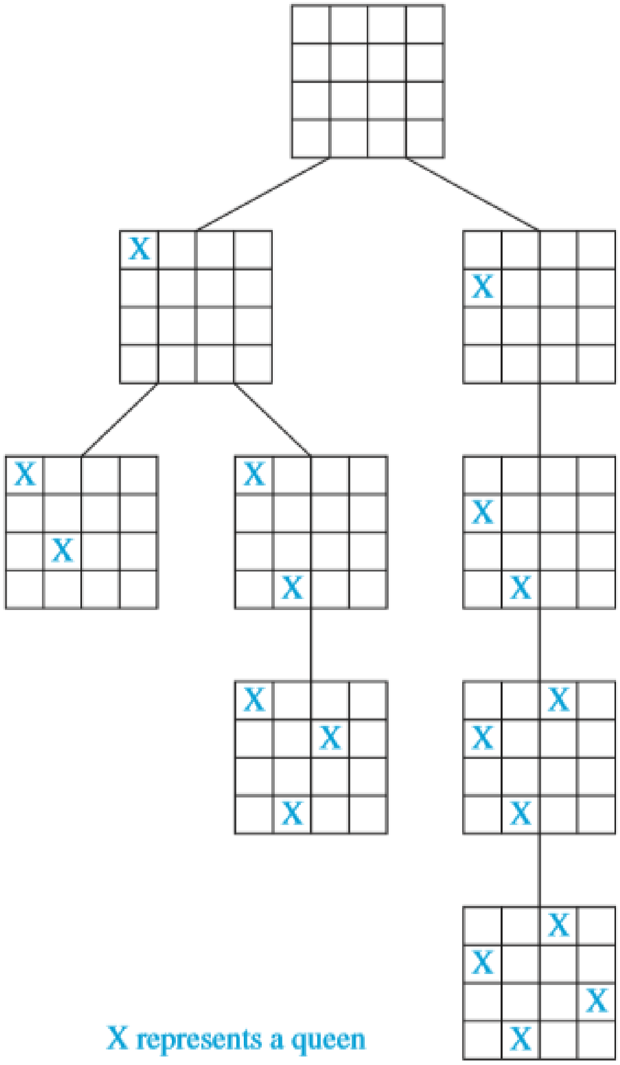


FIGURE 12 A Backtracking Solution of the Four-Queens Problem.

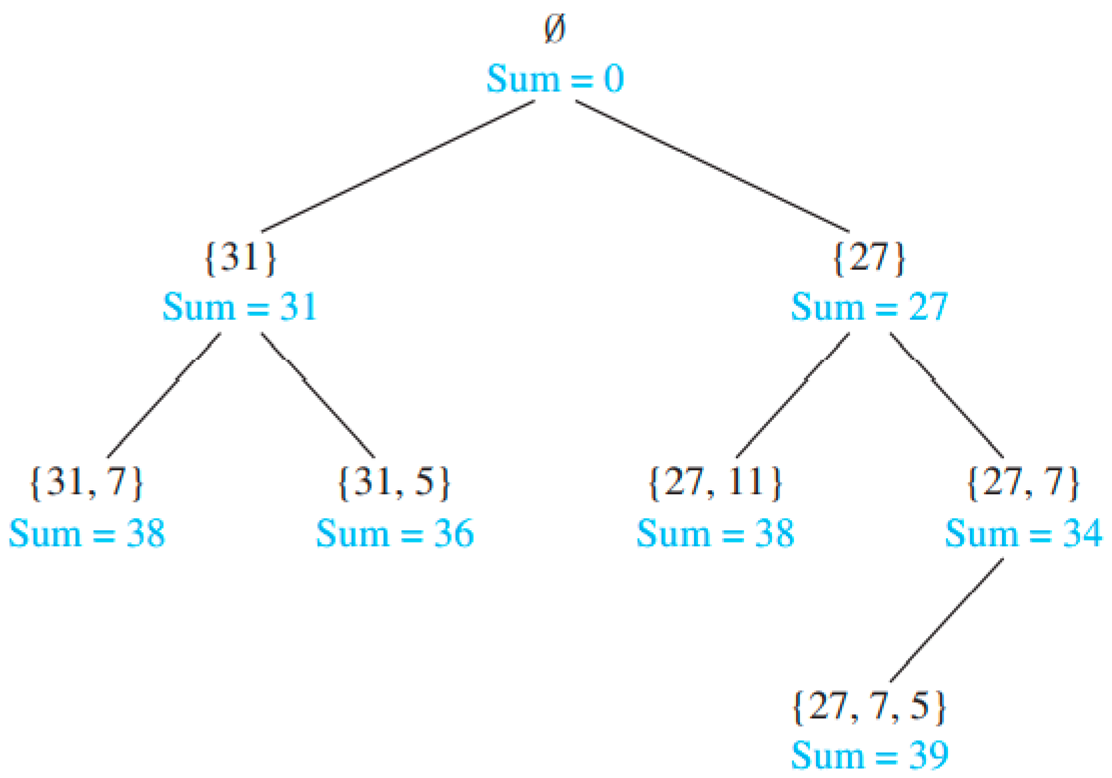


FIGURE 13 **Find a Sum Equal to 39 Using Backtracking.**

Breadth-First Search

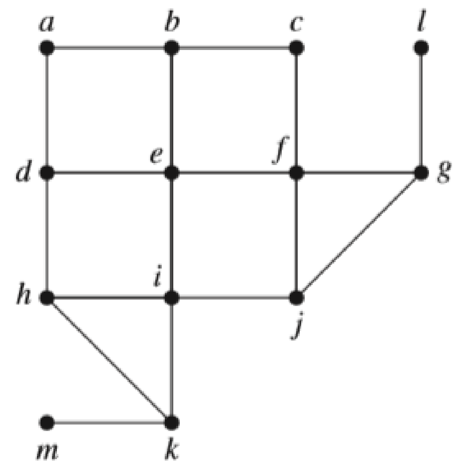


FIGURE 9 A Graph G.

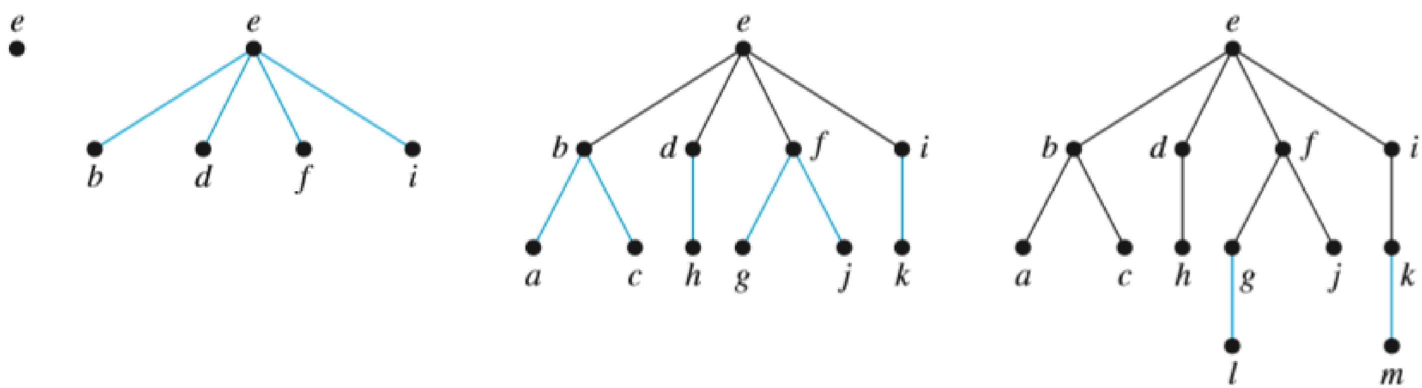


FIGURE 10 Breadth-First Search of G.

ALGORITHM 2 Breadth-First Search.

```
procedure BFS (G: connected graph with vertices v1, v2, ..., vn)
T := tree consisting only of vertex v1
L := empty list
put v1 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
```

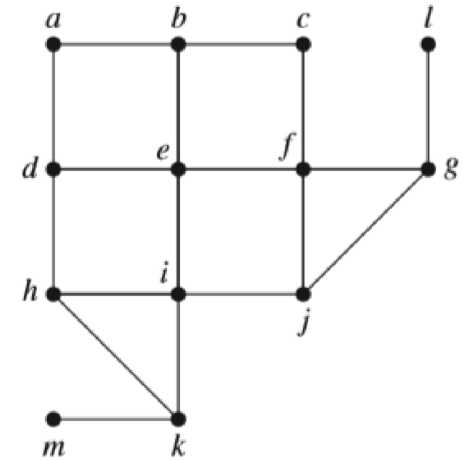


FIGURE 9 A Graph G.

Minimum Spanning Trees

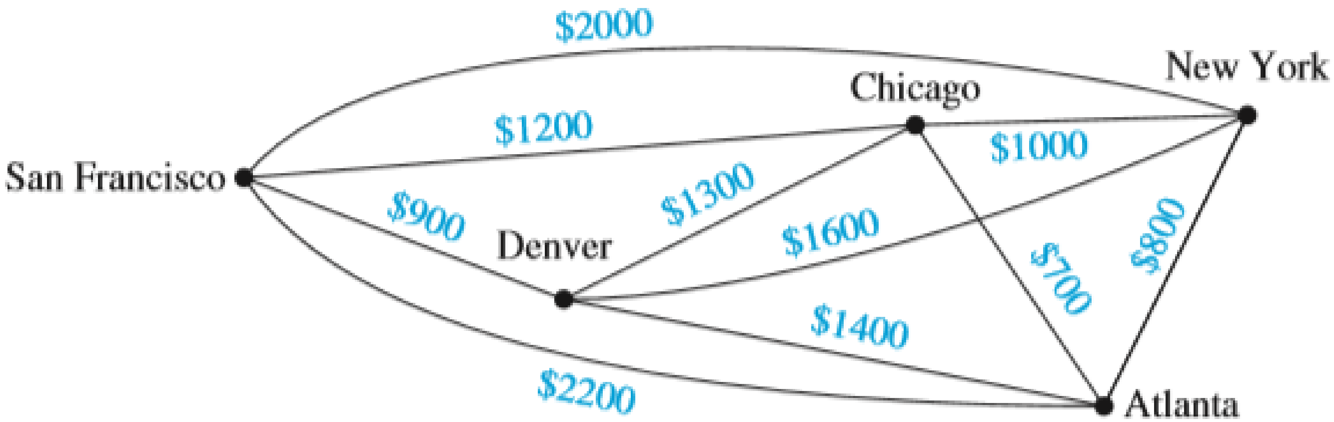


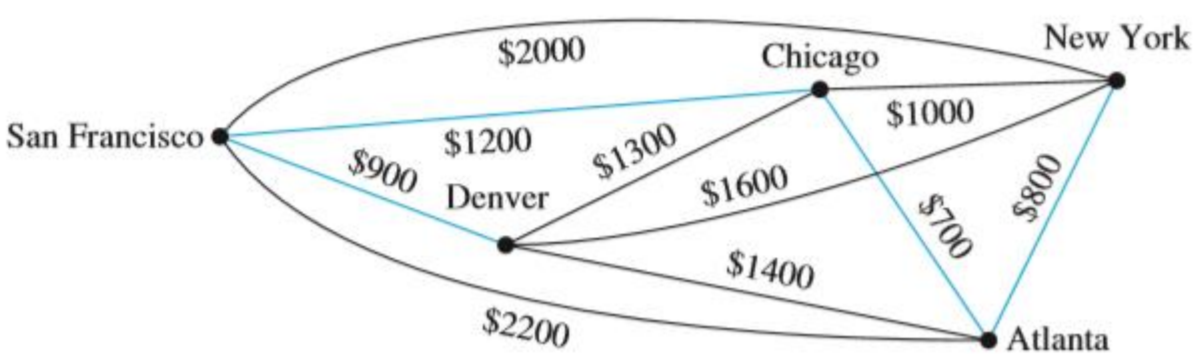
FIGURE 1 **A Weighted Graph Showing Monthly Lease Costs for Lines in a Computer Network.**

Which links should be made to ensure that there is a path between any two computer centers so that the total cost of the network is minimized?

➔ minimum spanning tree: a spanning tree that has the smallest possible sum of weights of its edges.

Algorithms for Minimum Spanning Trees

- Prim's algorithm (1957 by Robert Prim) (originally discovered by Vojtech Jarník in1930).
- Choosing any edge with smallest weight, putting it into the spanning tree.
- Add to the tree edges of minimum weight that are incident to a vertex already in the tree, never forming a simple circuit with those edges already in the tree.
- Stop when n−1 edges have been added.



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

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}

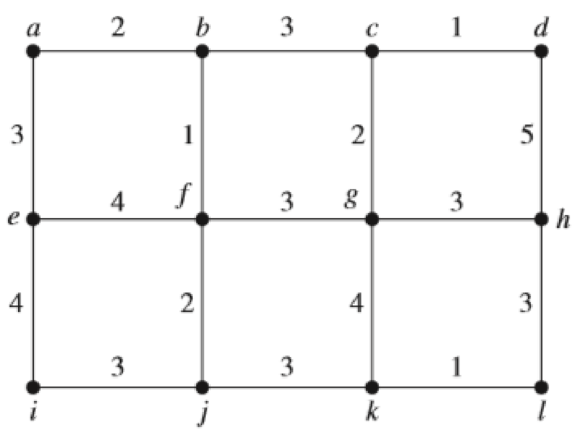


FIGURE 3 A Weighted Graph.

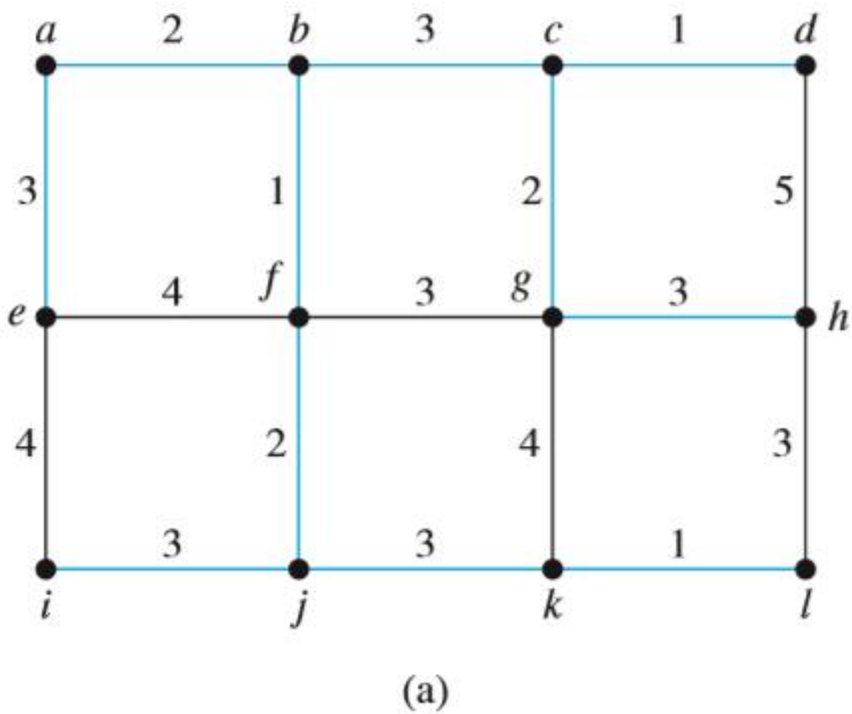
Choice	Edge	Weight
1	$\{b, f\}$	1
2	$\{a, b\}$	2
3	$\{f, j\}$	2
4	$\{a, e\}$	3
5	$\{i, j\}$	3
6	$\{f, g\}$	3
7	$\{c, g\}$	2
8	$\{c, d\}$	1
9	$\{g, h\}$	3
10	$\{h, l\}$	3
11	$\{k, l\}$	1
	Total:	24

Kruskal's algorithm (by Joseph Kruskal in 1956)

- Choose an edge in the graph with minimum weight.
- Add edges with minimum weight that do not form a simple circuit with those edges already chosen.
- Stop after n−1 edges have been selected.

ALGORITHM 2 Kruskal's Algorithm.

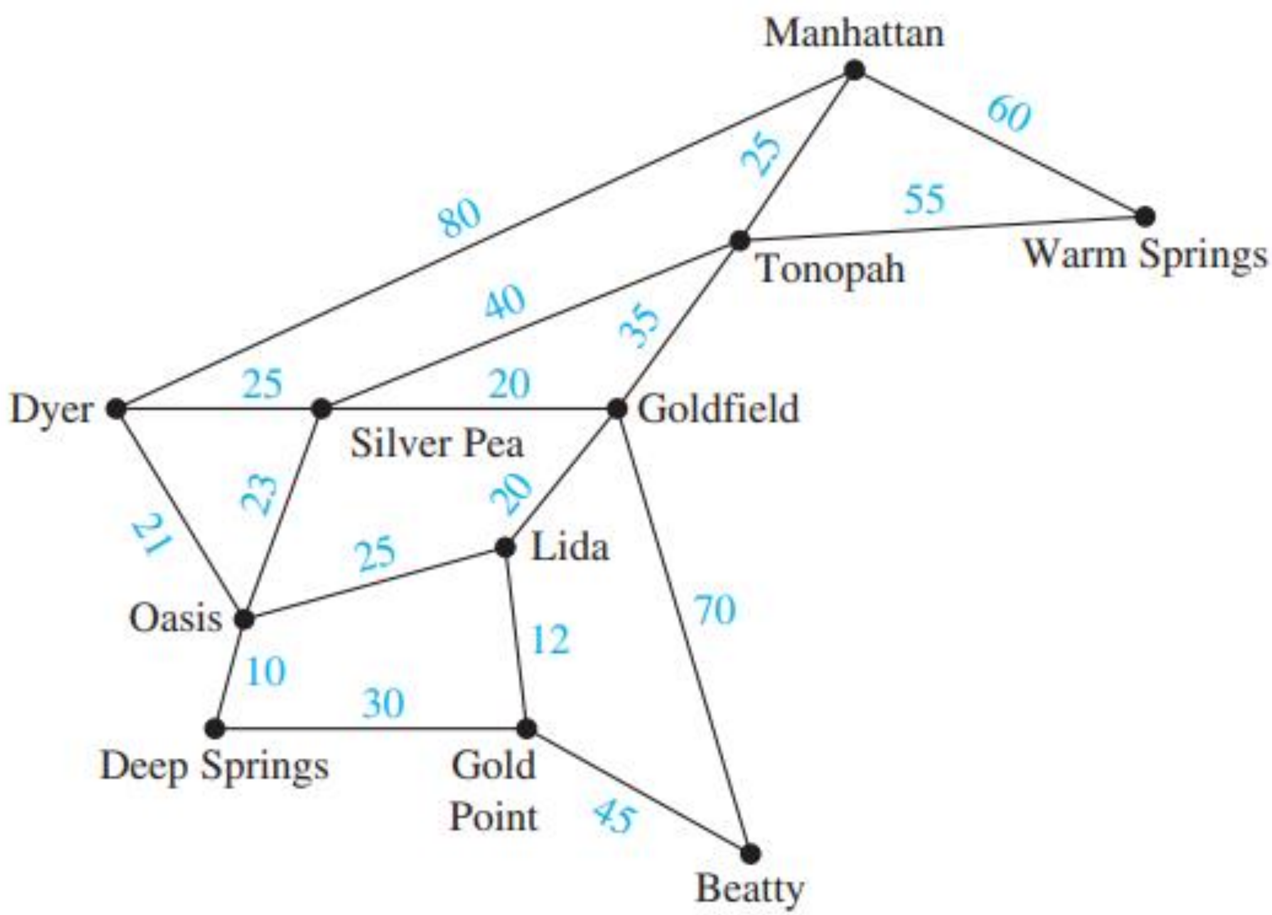
procedure *Kruskal*(G: weighted connected undirected graph with n vertices)
T := empty graph
for $i := 1$ **to** $n - 1$
    e := any edge in G with smallest weight that does not form a simple circuit
        when added to T
    $T := T$ with e added
return T {T is a minimum spanning tree of G}

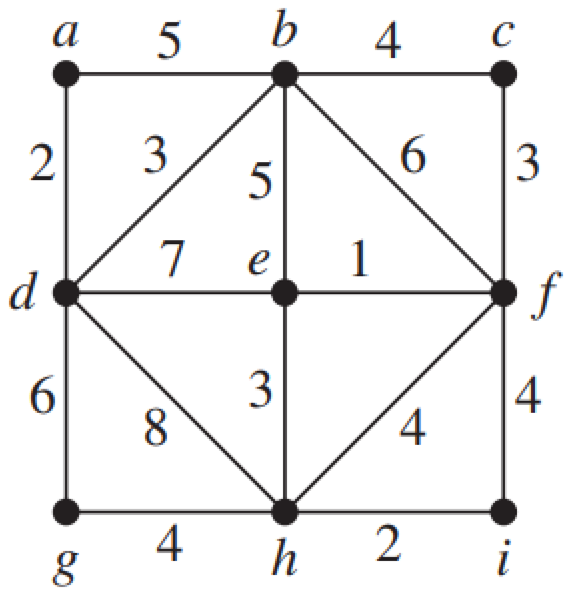


Choice	Edge	Weight
1	$\{c, d\}$	1
2	$\{k, l\}$	1
3	$\{b, f\}$	1
4	$\{c, g\}$	2
5	$\{a, b\}$	2
6	$\{f, j\}$	2
7	$\{b, c\}$	3
8	$\{j, k\}$	3
9	$\{g, h\}$	3
10	$\{i, j\}$	3
11	$\{a, e\}$	3
	Total:	24

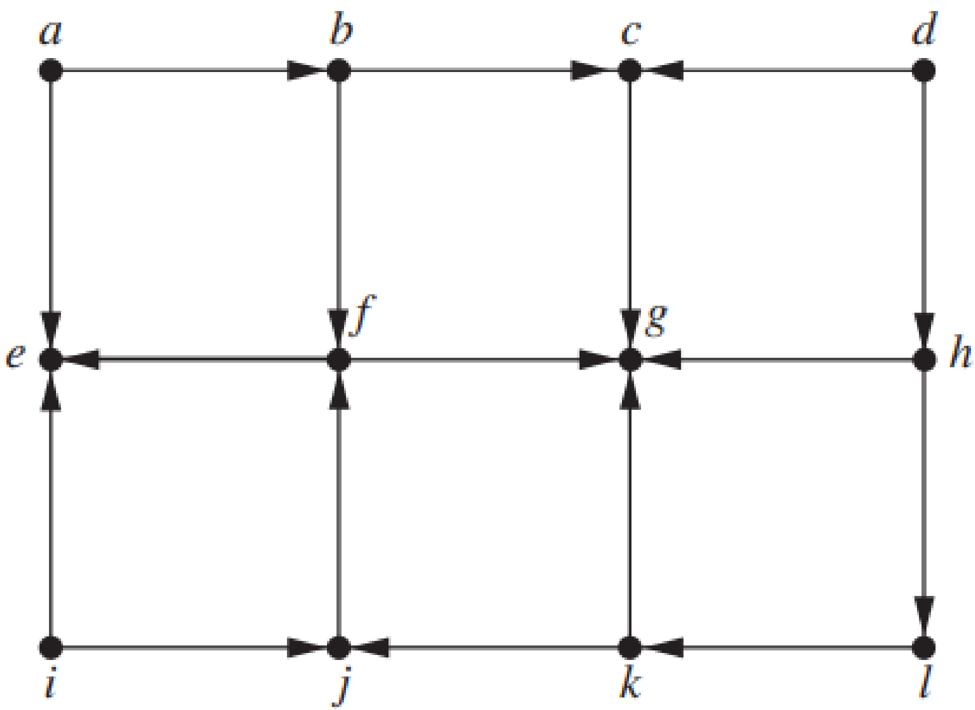
(b)

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


Manhattan
60
25
80
55
Tonopah
Warm Springs
40
35
Dyer
25
20
Goldfield
Silver Pea
21
23
20
Lida
25
Oasis
70
12
10
30
Deep Springs
Gold
Point
45
Beatty


a
5
b
4
c
2
3
5
6
3
d
7
e
1
f
6
8
3
4
4
g
4
h
2
i

DFS



Summary

11 Trees

- 11.1 Introduction to Trees
- 11.2 Applications of Trees
- 11.3 Tree Traversal
- 11.4 Spanning Trees
- 11.5 Minimum Spanning Trees

- **Thanks**