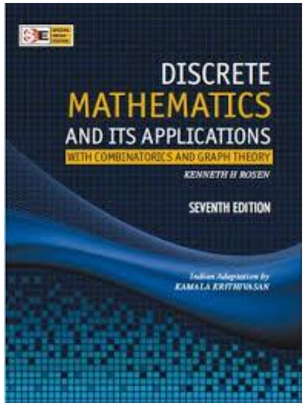
Vietnam National University of HCMC
International University
School of Computer Science and Engineering



Session 11 Tree

(Dec, 3th 2014)

Nguyen Van Sinh, Ph.D

nvsinh@hcmiu.edu.vn

Outline

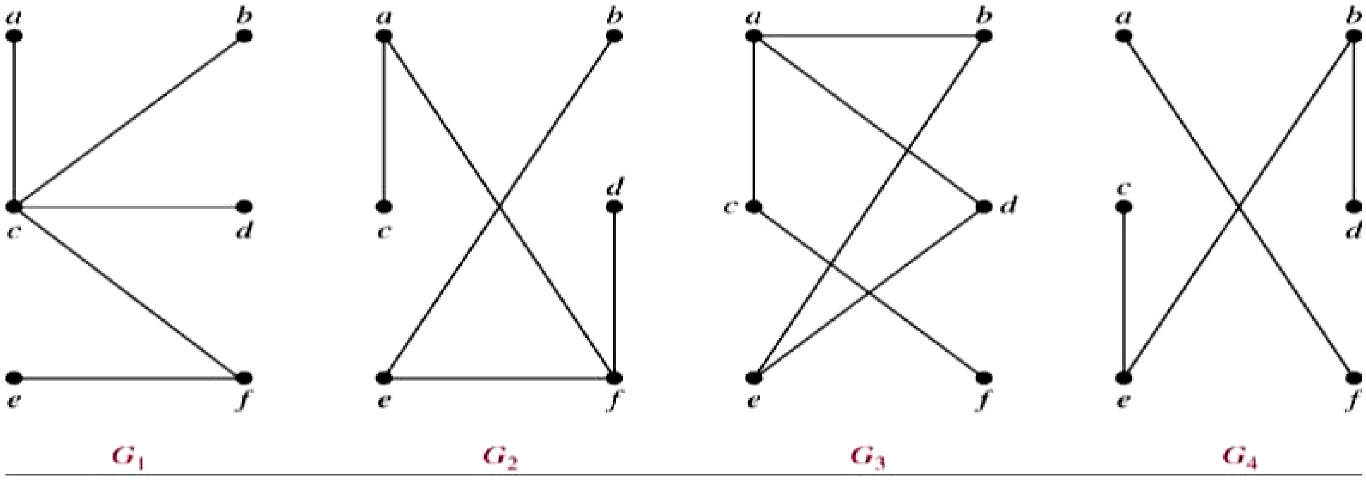
- Introduction to Trees
- Applications of Trees
- Tree Traversal
- Spanning Trees

Refer: chapter 10 in the textbook

Introduction to Trees

Def 1: A tree is a connected undirected graph with no simple circuits.

Example 1. Which of the graphs are trees?



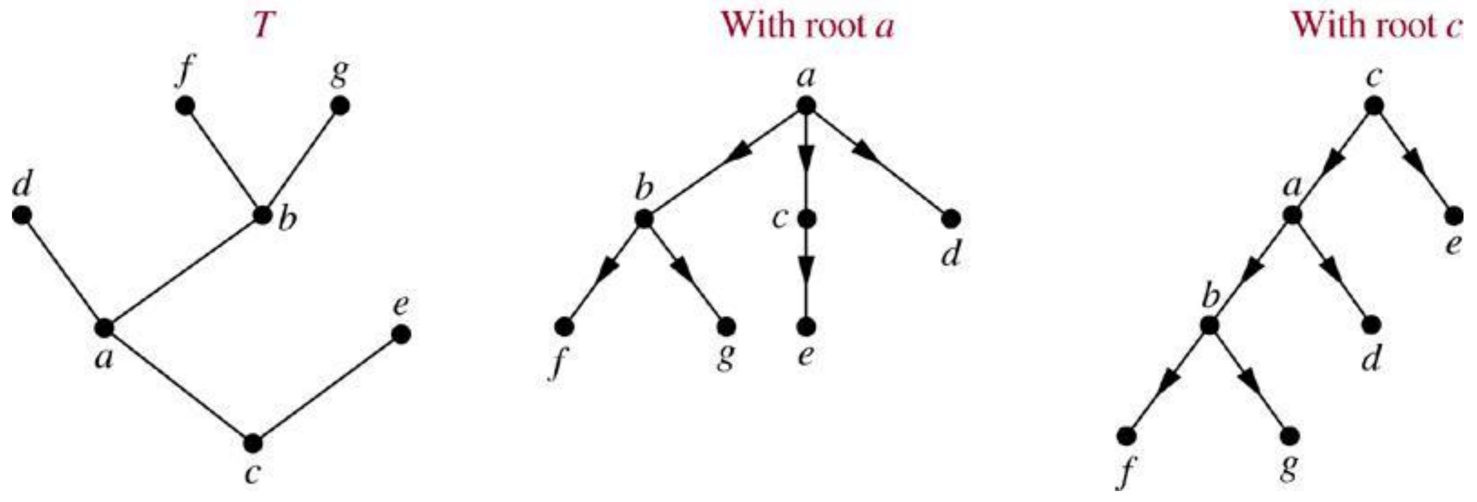
Sol: G_1, G_2

Introduction to Trees

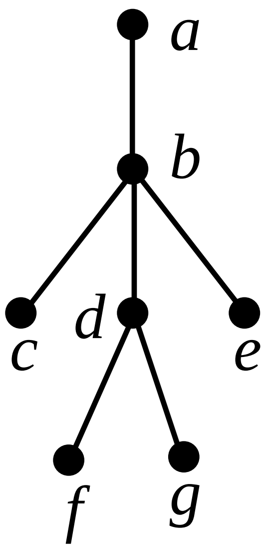
Thm 1: Any undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Def 2. A rooted tree is a tree in which one vertex has been designed as the root and every edge is directed away from the root.

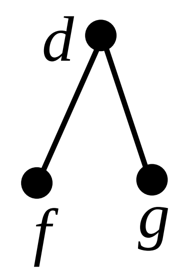
Example



Introduction to Trees



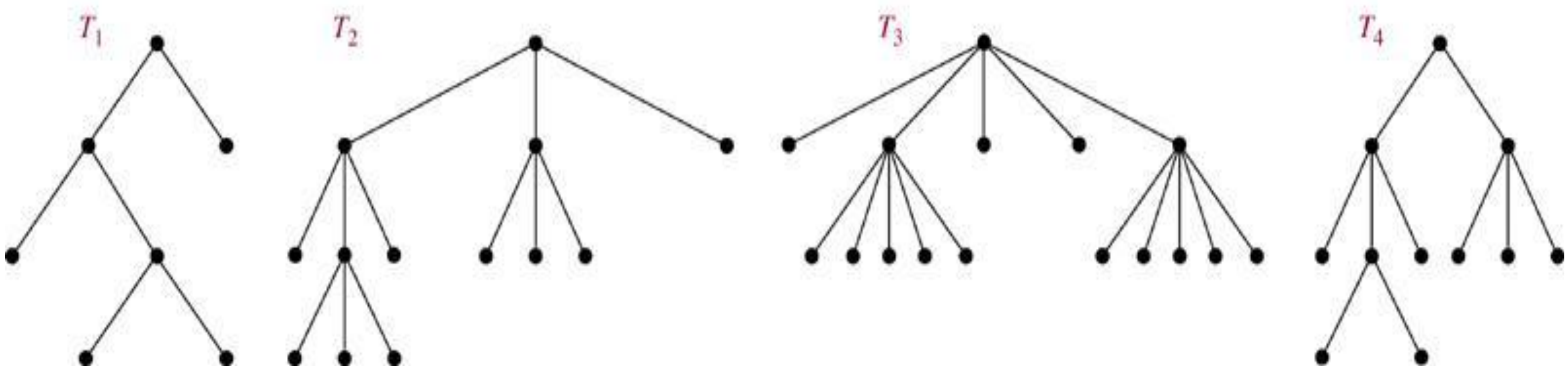
Def: *a* is the parent of *b*, *b* is the child of *a*;
c, *d*, *e* are siblings;
a, *b*, *d* are ancestors of *f*;
c, *d*, *e*, *f*, *g* are descendants of *b*;
c, *e*, *f*, *g* are leaves of the tree (deg=1)
a, *b*, *d* are internal vertices of the tree (at least one child)
sub-tree with *d* as its root:



Introduction to Trees

Def 3 A rooted tree is called an *m-ary tree* if every internal vetex 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*.

Example 3



full binary tree | full 3-ary tree | full 5-ary tree | not full 3-ary tree

Introduction to Trees

Binary tree

Each non-leaf node has *up to 2 children*. If every non-leaf node has exactly two nodes, then it becomes a **full binary tree**

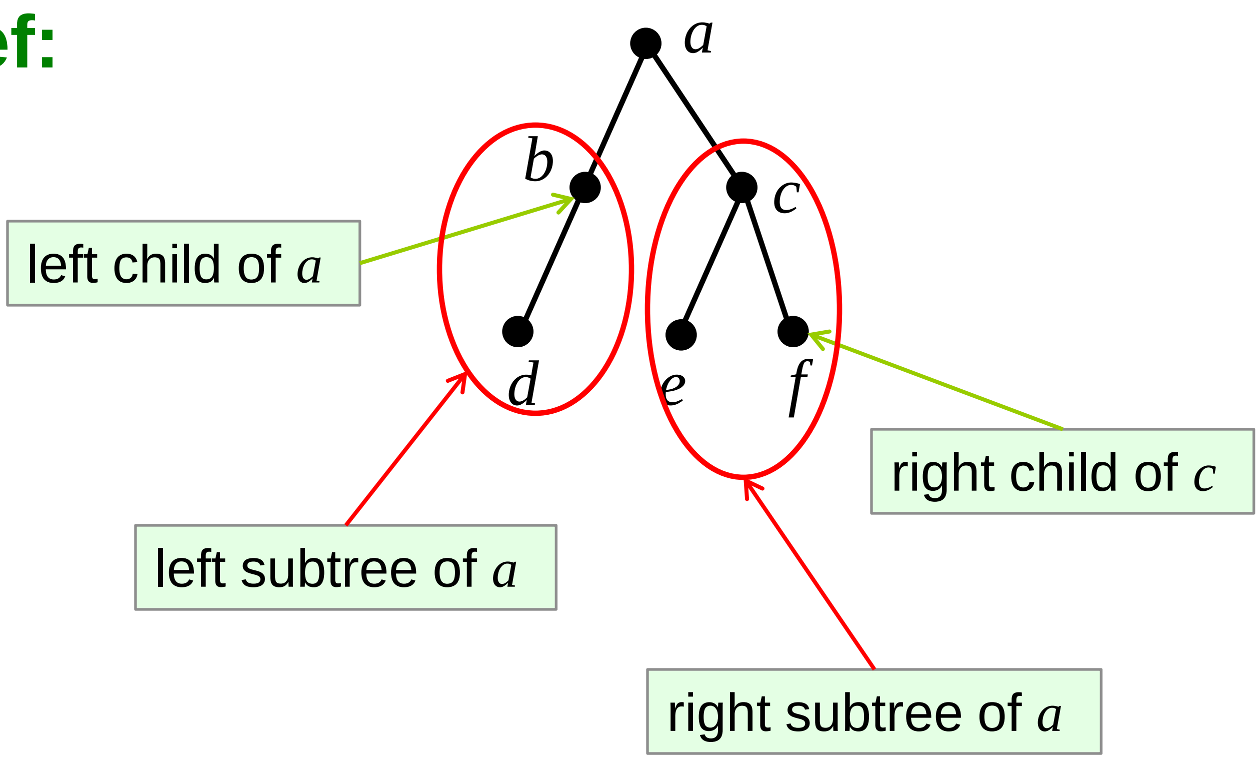
Question:

1. How many edges does a **full binary tree with n nodes** have?
2. How many edges does a **full m-ary tree with n nodes** have?

n-1?

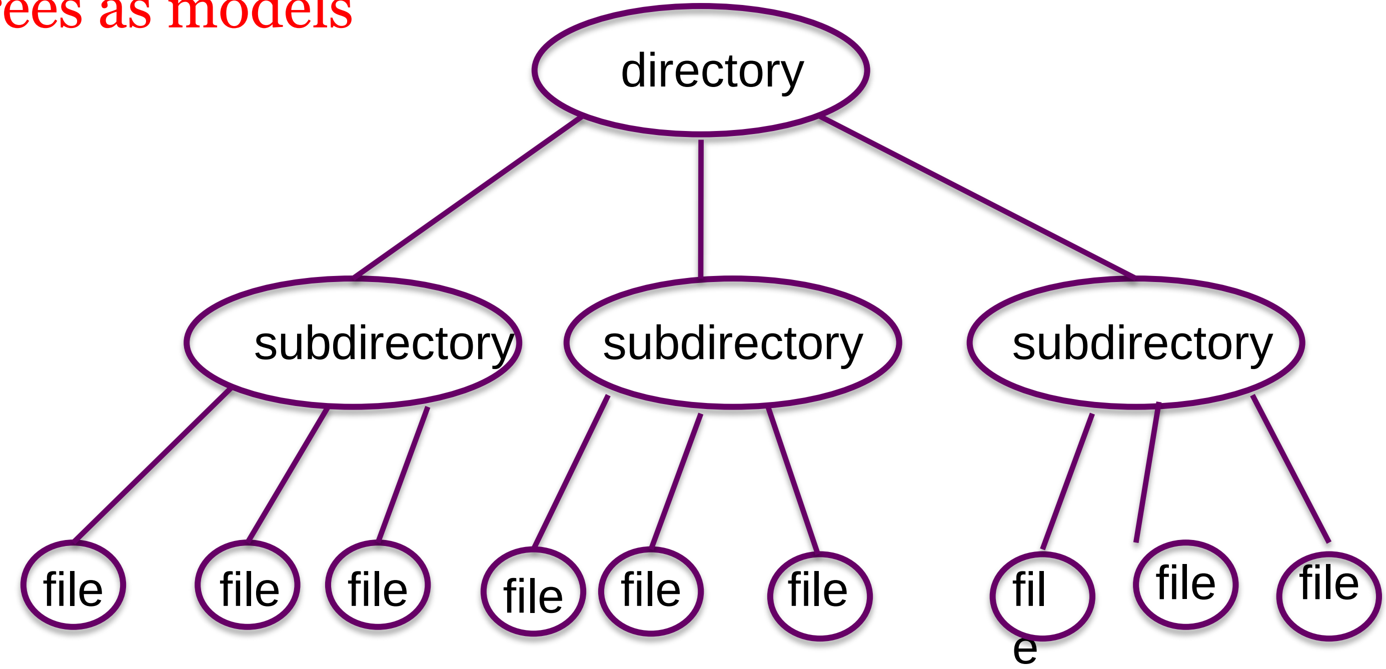
Introduction to Trees

Def:



Introduction to Trees

Trees as models

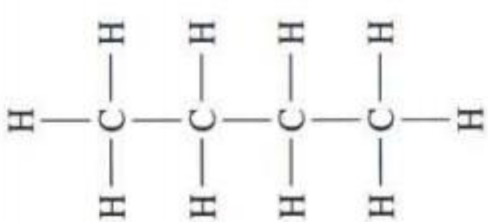


Computer File System

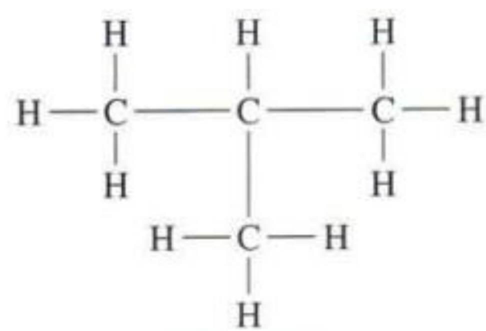
This tree is a ternary (3-ary) tree, since each non-leaf node has three children

Introduction to Trees

Trees as models



Butane



Isobutane

Introduction to Trees

Properties of Trees

Thm 2. A tree with n vertices has $n-1$ edges.

Pf. (by induction on n)

$n = 1$: K_1 is the only tree of order 1, $|E(K_1)| = 0$. ok!

Assume the result is true for every trees of order $n = k$.

Let T be a tree of order $n = k+1$, v be a leaf of T, and w be the parent of v.

Let T' be the tree $T - \{v\}$.

$\therefore |V(T')| = k$, and $|E(T')| = k-1$ by the induction hypothesis.

$\Rightarrow |E(T)| = k$

By induction, the result is true for all trees. (sol for slide 7)

Introduction to Trees

Thm 3: A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

Pf. Every vertex, except the root, is the child of an internal vertex.

Each internal vertex has m children.

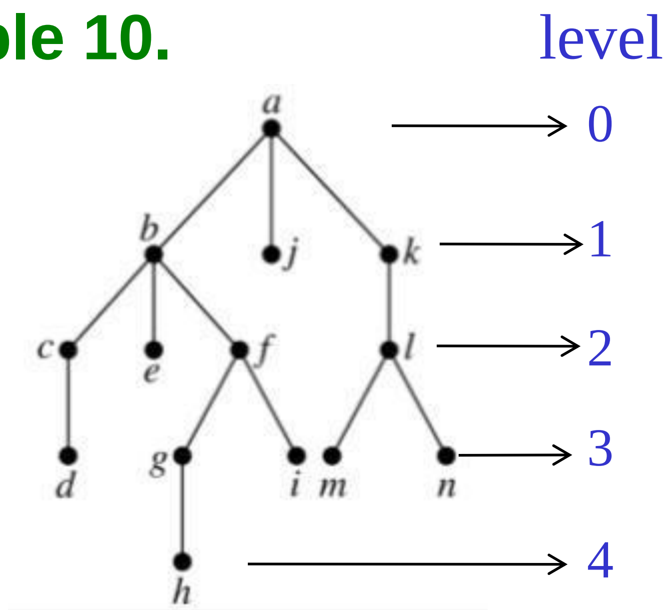
$\Rightarrow$ there are $mi + 1$ vertices in the tree

Cor. A full m-ary tree with n vertices contains $(n-1)/m$ internal vertices, and hence $n - (n-1)/m = ((m-1)n+1)/m$ leaves.

Introduction to Trees

Def: The level of a vertex v in a rooted tree is the length of the unique path from the root to this vertex. The level of the root is defined to be zero. The height of a rooted tree is the maximum of the levels of vertices.

Example 10.

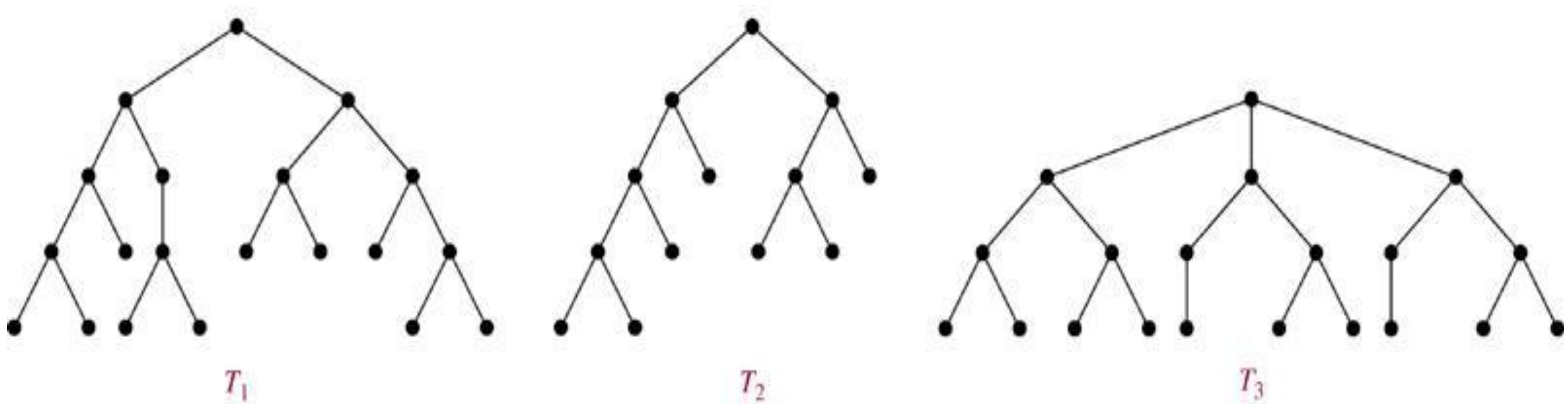


height = 4

Introduction to Trees

Def: A rooted m-ary tree of height h is balanced if all leaves are at levels h or $h-1$.

Example 11 Which of the rooted trees shown below are balanced?



Sol. T_1, T_3

Thm 5: There are at most m^h leaves in an m-ary tree of height h

Introduction to Trees

Def: A complete m-ary tree is a full m-ary tree, where every leaf is at the same level.

Question: How many vertices and how many leaves does a complete m-ary tree of height h have?

Sol.

- number of vertices = $1+m+m^2+\ldots+m^h = (m^{h+1}-1)/(m-1)$

- number of leaves = m^h

Applications of Trees

Binary Search Trees

Goal: Implement a searching algorithm that finds items efficiently when the items are totally ordered.

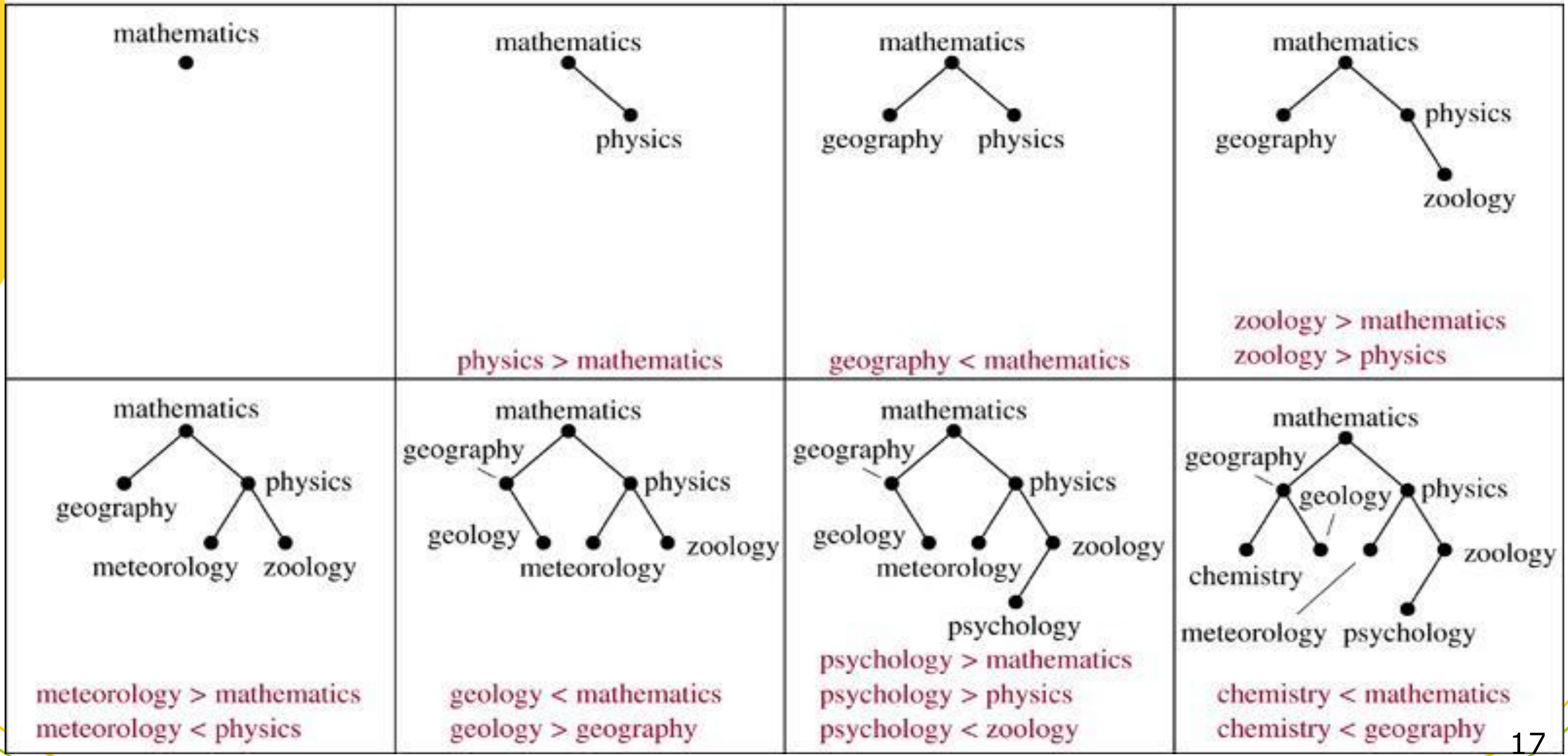
Binary Search Tree: Binary tree, each child of a vertex is designed as a right or left child, and each vertex v is labeled with a key $label(v)$, which is one of the items.

Note: $label(v) > label(w)$ if w is in the left subtree of v and $label(v) < label(w)$ if w is in the right subtree of v

Applications of Trees

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

Sol



Applications of Trees

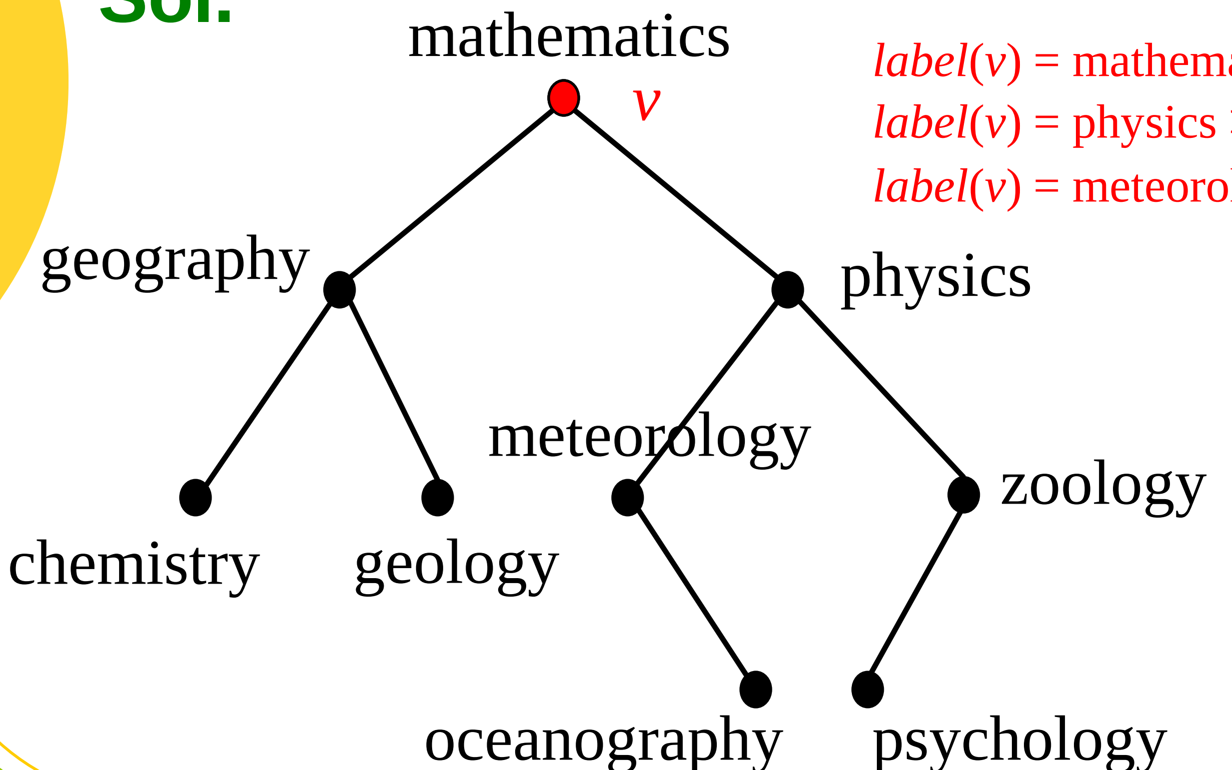
Algorithm 1: Locating and Adding Items to a Binary Search Tree

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 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}

Applications of Trees

Example 2 Use Algorithm 1 to insert the word *oceanography* into the binary search tree in Example 1.

Sol.



label(*v*) = mathematics < *oceanography*

label(*v*) = physics > *oceanography*

label(*v*) = meteorology < *oceanography*

Applications of Trees

Decision Trees

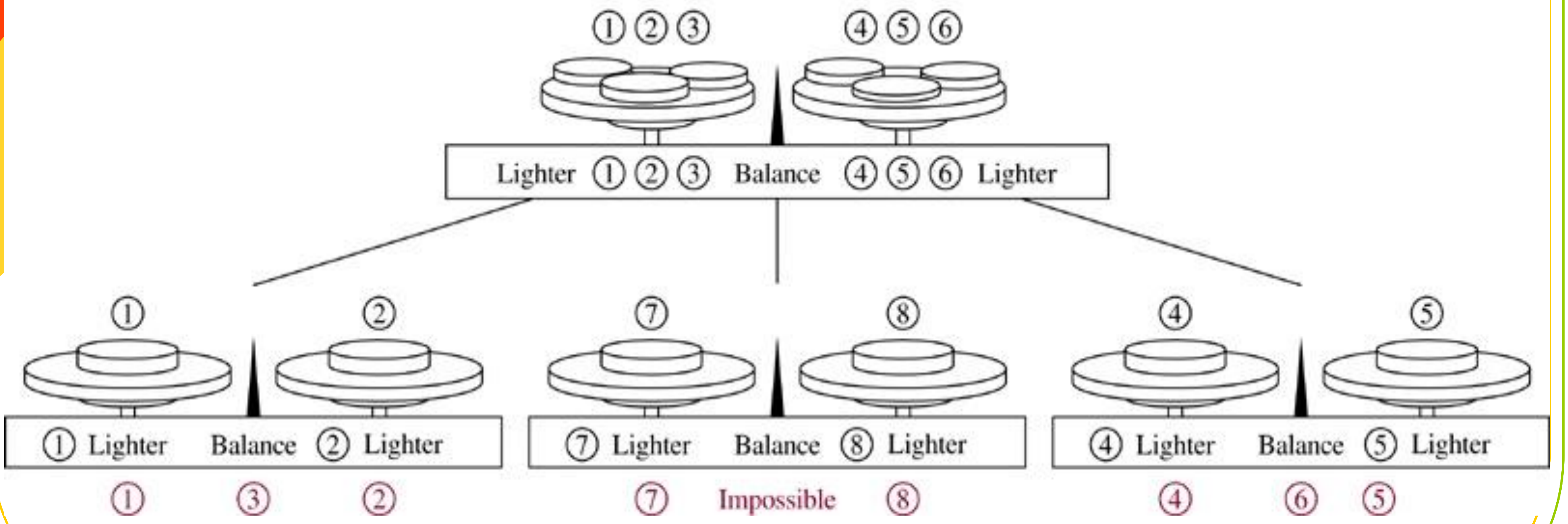
A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a decision tree.

Example 3 Suppose there are seven coins, all with the same weight, and a counterfeit coin that weights less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.

Applications of Trees

Sol. $\Rightarrow$ 3-ary tree

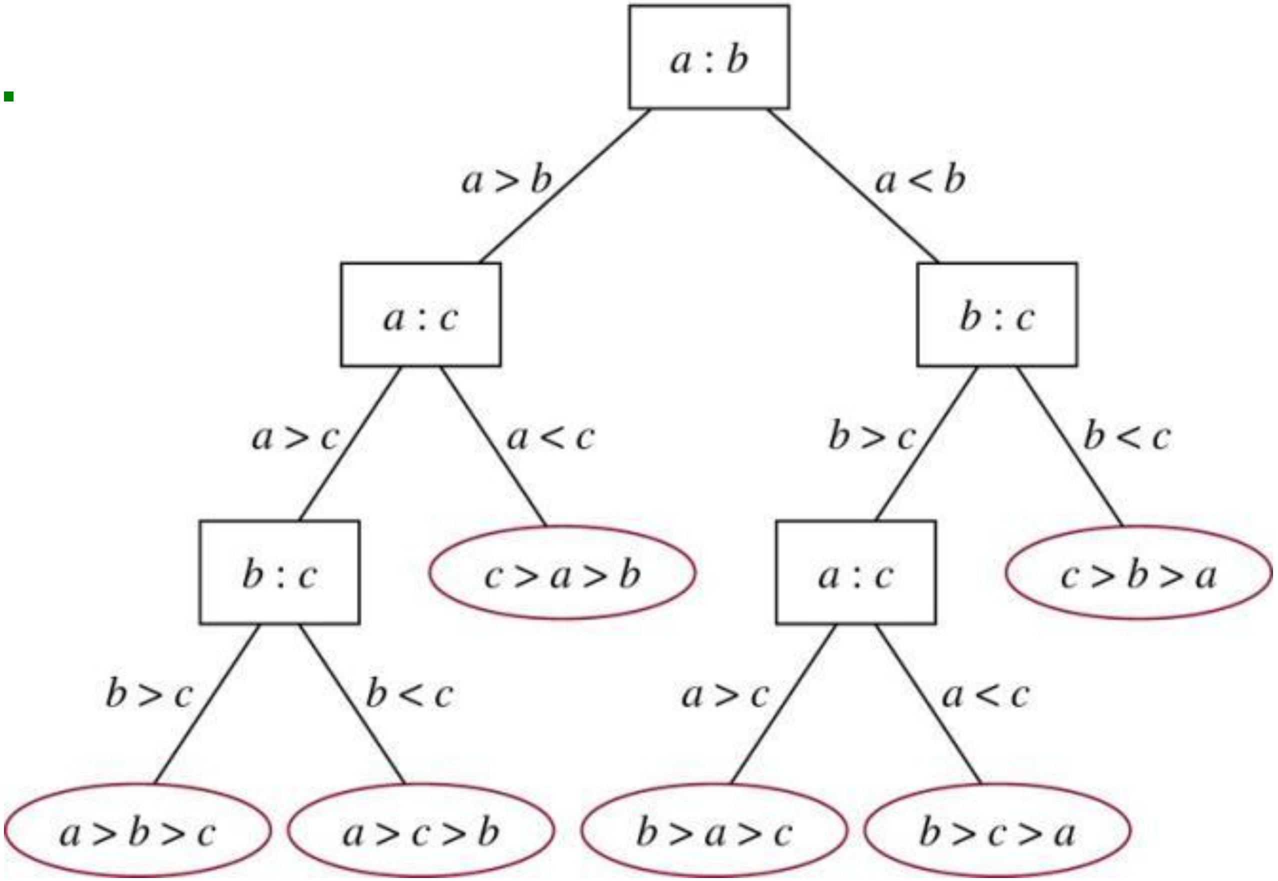
Need 8 leaves $\Rightarrow$ Cor 1: $\text{Log}_3 8 = 2$



Applications of Trees

Example 4 A decision tree that orders the elements of the list a, b, c.

Sol.



Applications of Trees

Prefix Codes

Problem: Using bit strings to encode the letter of the English alphabet

$\Rightarrow$ each letter needs a bit string of length 5 ($2^4 < 26 < 2^5$)

$\Rightarrow$ Is it possible to find a coding scheme of these letter such that when data are coded, fewer bits are used?

$\Rightarrow$ Encode letters using varying numbers of bits.

$\Rightarrow$ Some methods must be used to determine where the bits for each character start and end.

$\Rightarrow$ Prefix codes: Codes with the property that the bit string for a letter never occurs as the first part of the bit string for another letter.

Applications of Trees

Example: (not prefix code)

e : 0, a : 1, t : 01

The string 0101 could correspond to *eat?*, *tea?*, *eaea?*, or *tt?*.

Example: (prefix code)

e : 0, a : 10, t : 11

The string 10110 is the encoding of *ate*.

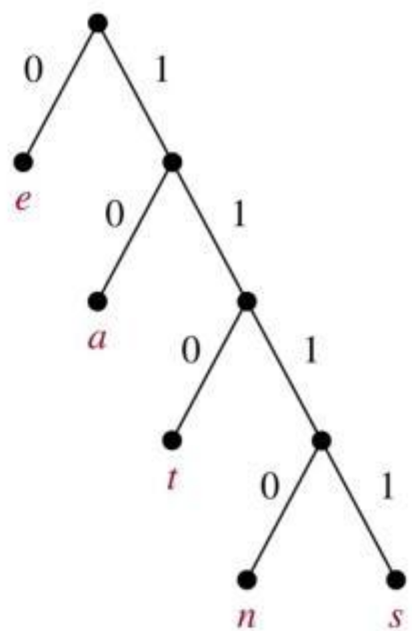
Applications of Trees

A prefix code can be represented using a binary tree.

character: the label of the leaf edge label: left child → 0, right child → 1

The bit string used to encode a character is the sequence of labels of the edges in the unique path from the root to the leaf that has this character as its label.

Example:



encode

$e : 0$

$a : 10$

$t : 110$

$n : 1110$

$s : 1111$

decode

$$\underbrace{1111}_{s}\underbrace{10}_{a}\underbrace{1110}_{n}\underbrace{0}_{e}$$

$\Rightarrow$ *sane*

Applications of Trees

Huffman Coding (data compression)

Main idea: Input the frequencies of symbols in a string and output a prefix code that encodes the string using the fewest possible bits, among all possible binary prefix codes for these symbols.

Applications of Trees

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 weighted 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

Applications of Trees

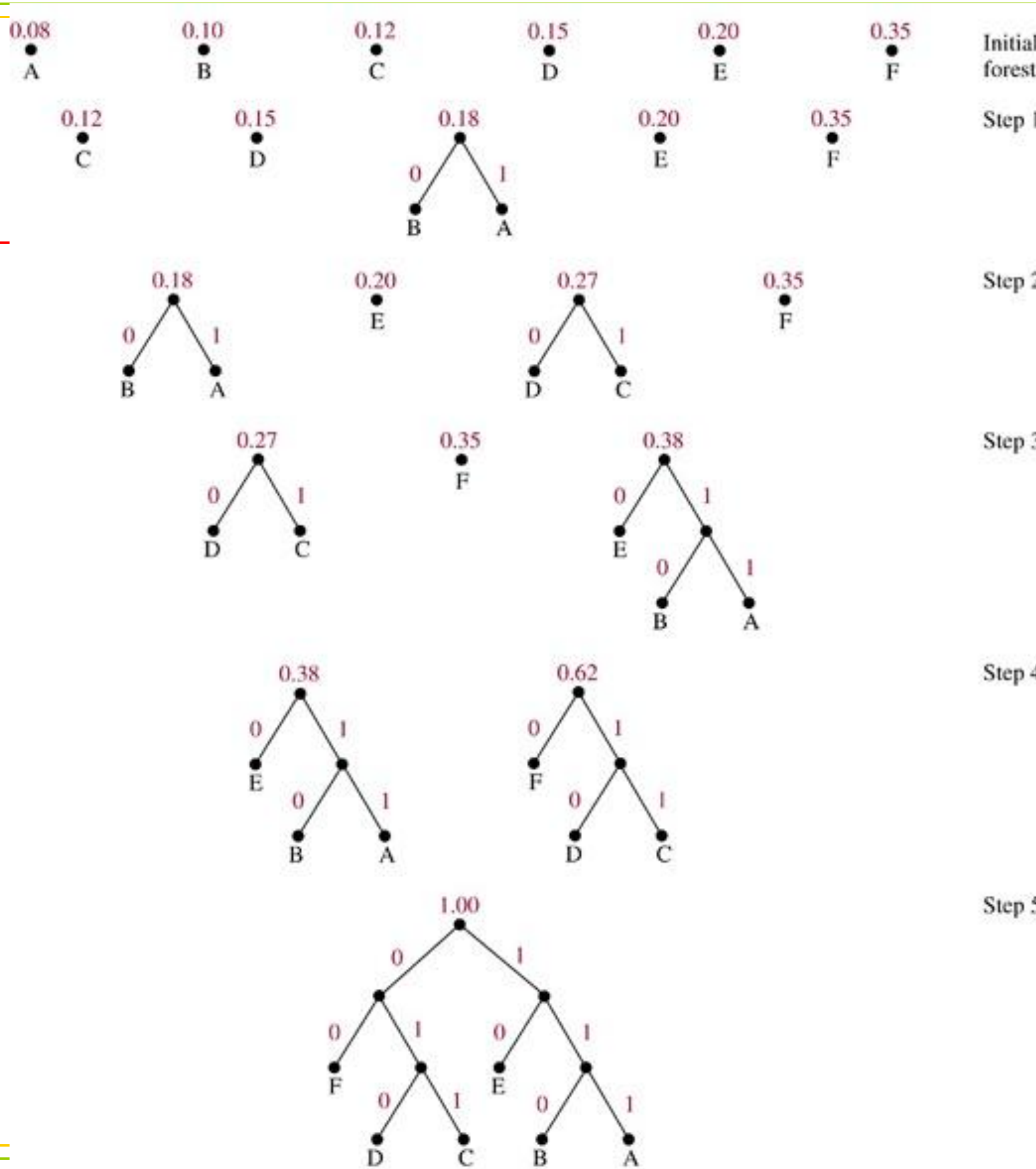
Example 5 Use Huffman coding to encode the following symbols with the frequencies listed:
A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35.
What is the average number of bits used to encode a character?

Sol:

The average number of bits is:

$$= 3\times0.08 + 3\times0.10 + 3\times0.12 + 3\times0.15 + 2\times0.20 + 2\times0.35$$

$$= 2.45$$


0.08 A
0.10 B
0.12 C
0.15 D
0.20 E
0.35 F
Initial forest
Step 1
0.12 C
0.15 D
0.18
0.20 E
0.35 F
Step 2
0.18
0.20 E
0.27
0.35 F
Step 3
0.27
0.35 F
0.38
Step 4
0.38
0.62
Step 5
1.00

Tree Traversal

We need procedures for visiting each vertex of an ordered rooted tree to access data.

Universal Address Systems

Label vertices:

1.root $\rightarrow$ 0, its k children $\rightarrow 1, 2, \ldots, k$ (from left to right)

2.For each vertex v at level n with label A, its r children $\rightarrow A.1, A.2, \ldots, A.r$ (from left to right).

We can totally order the vertices using the lexicographic ordering of their labels in the universal address system.

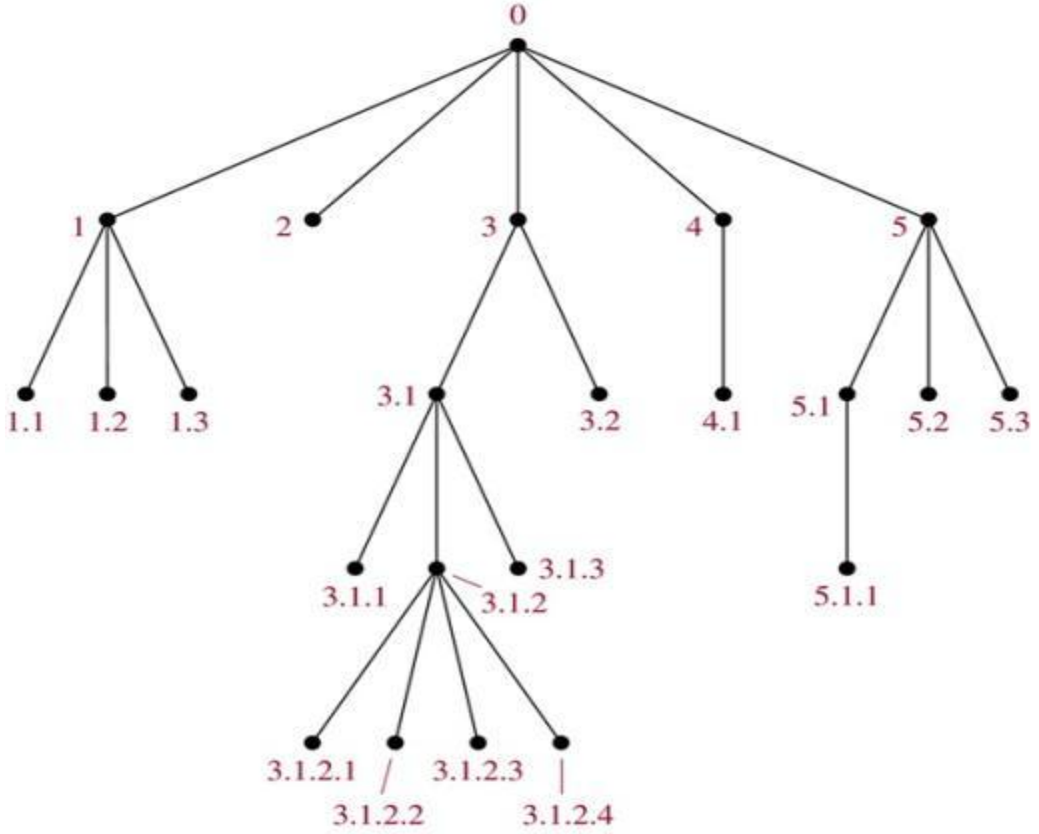
$x_1.x_2.\ldots.x_n < y_1.y_2.\ldots.y_m$

if there is an i, $0 \leq i \leq n$, with $x_1=y_1$, $x_2=y_2$, …, $x_{i-1}=y_{i-1}$, and $x_i<y_i$;

or if $n<m$ and $x_i=y_i$ for $i=1, 2, \ldots, n$.

Tree Traversal

Example 1



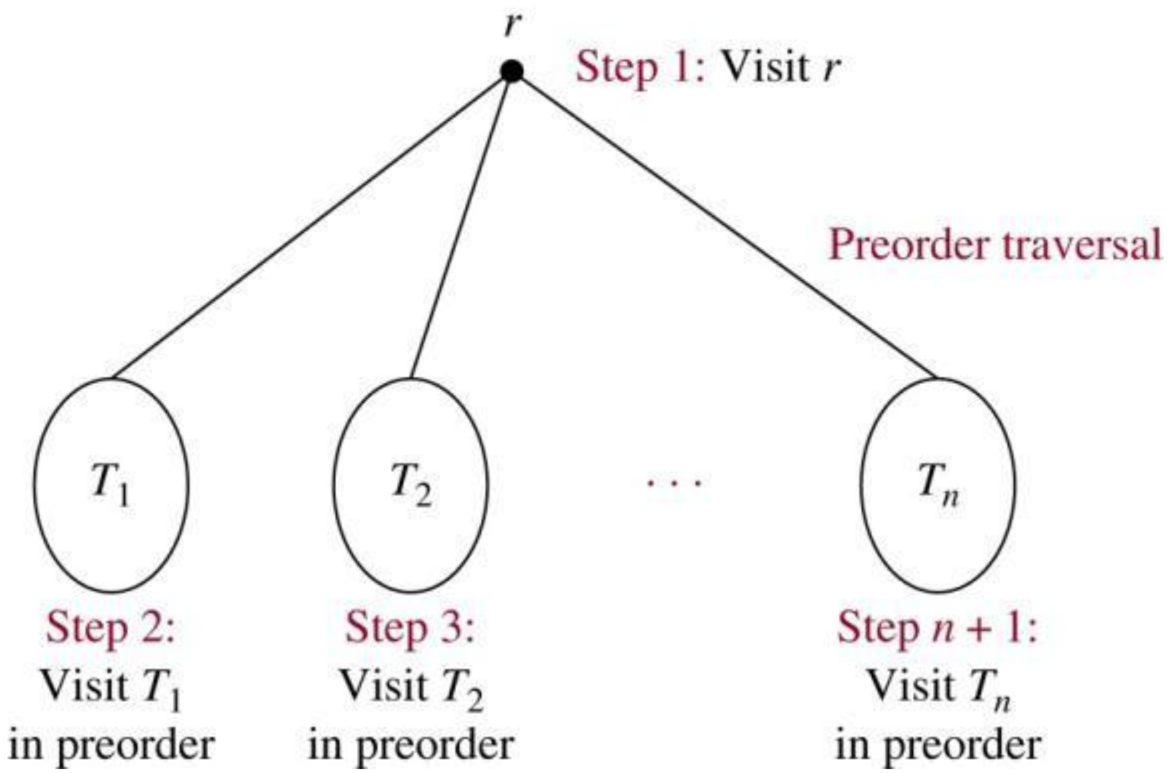
The lexicographic ordering is:

0 < 1<1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3

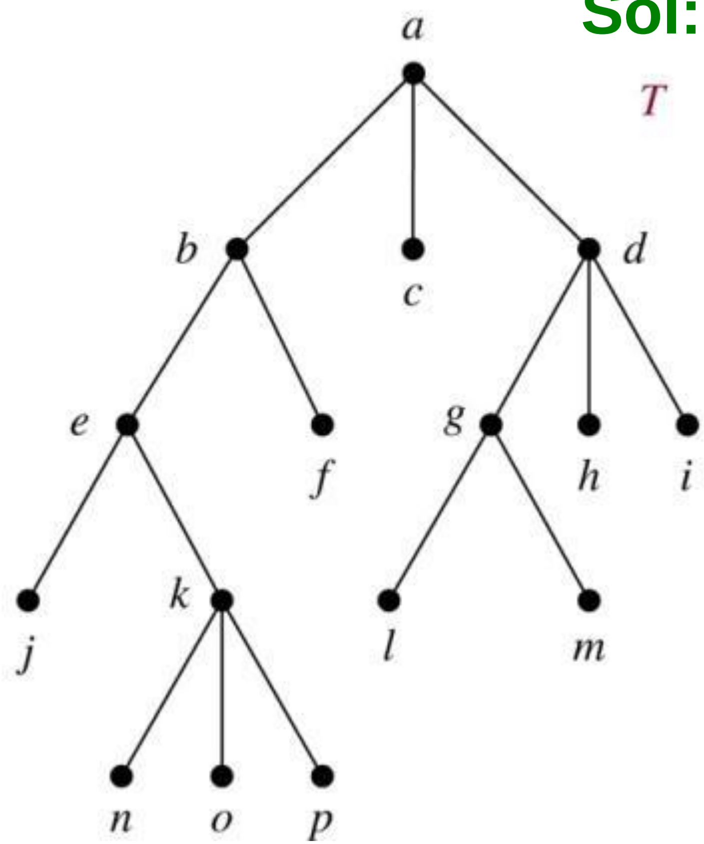
Tree Traversal

Traversal Algorithms

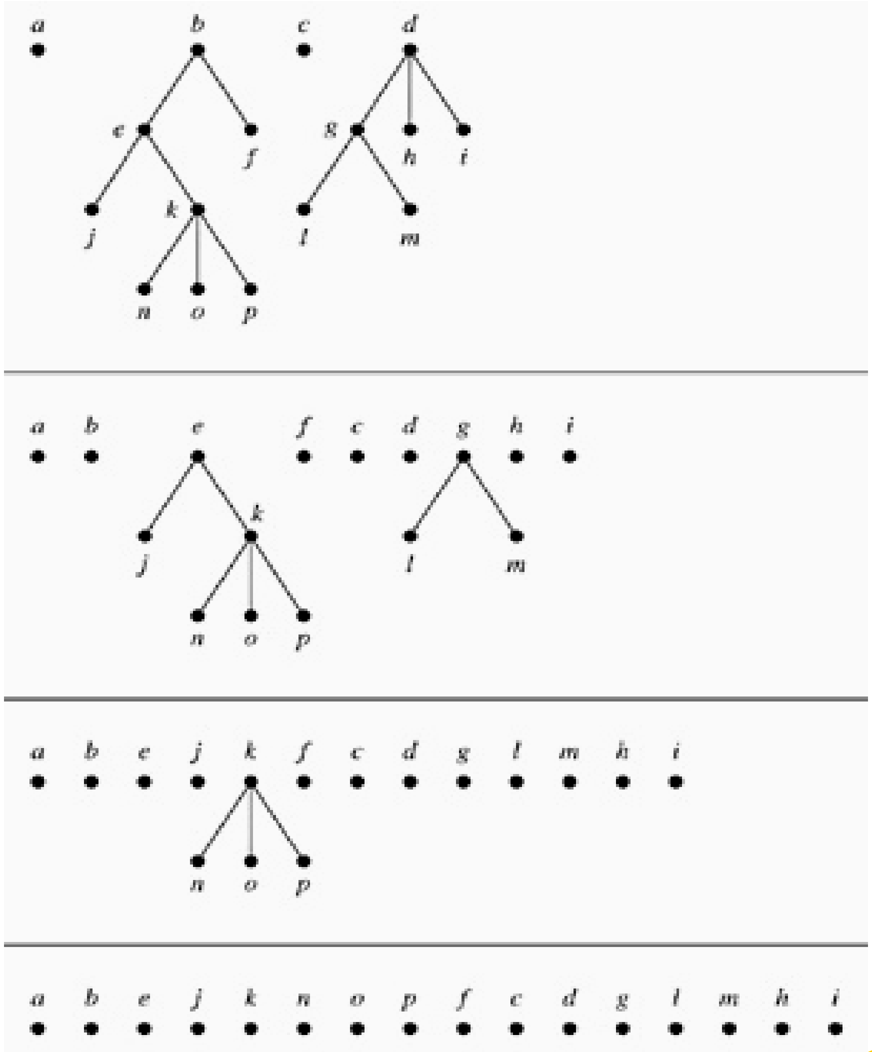
Preorder traversal: Root → Left → Right



Example 2. In which order does a preorder traversal visit the vertices in the ordered rooted tree T shown below?



Sol:



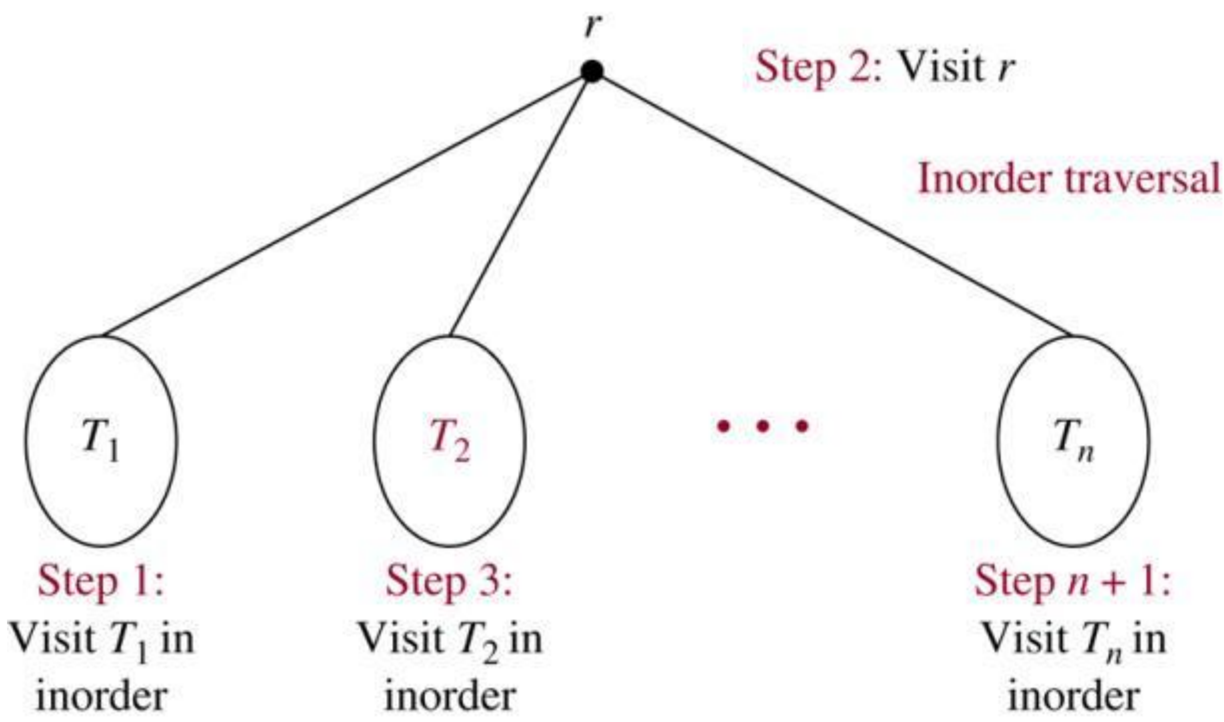
Tree Traversal

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

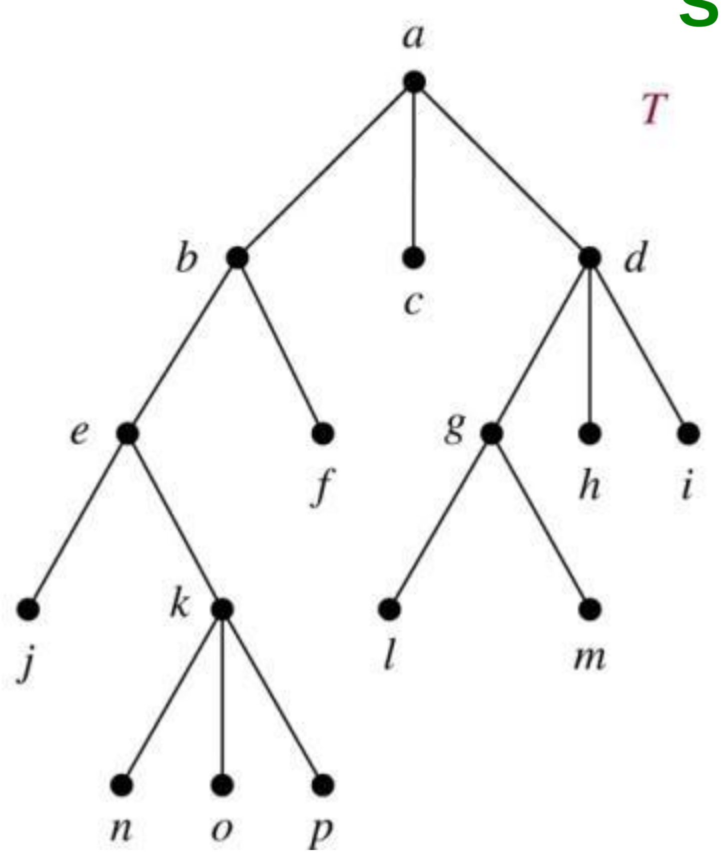
Tree Traversal

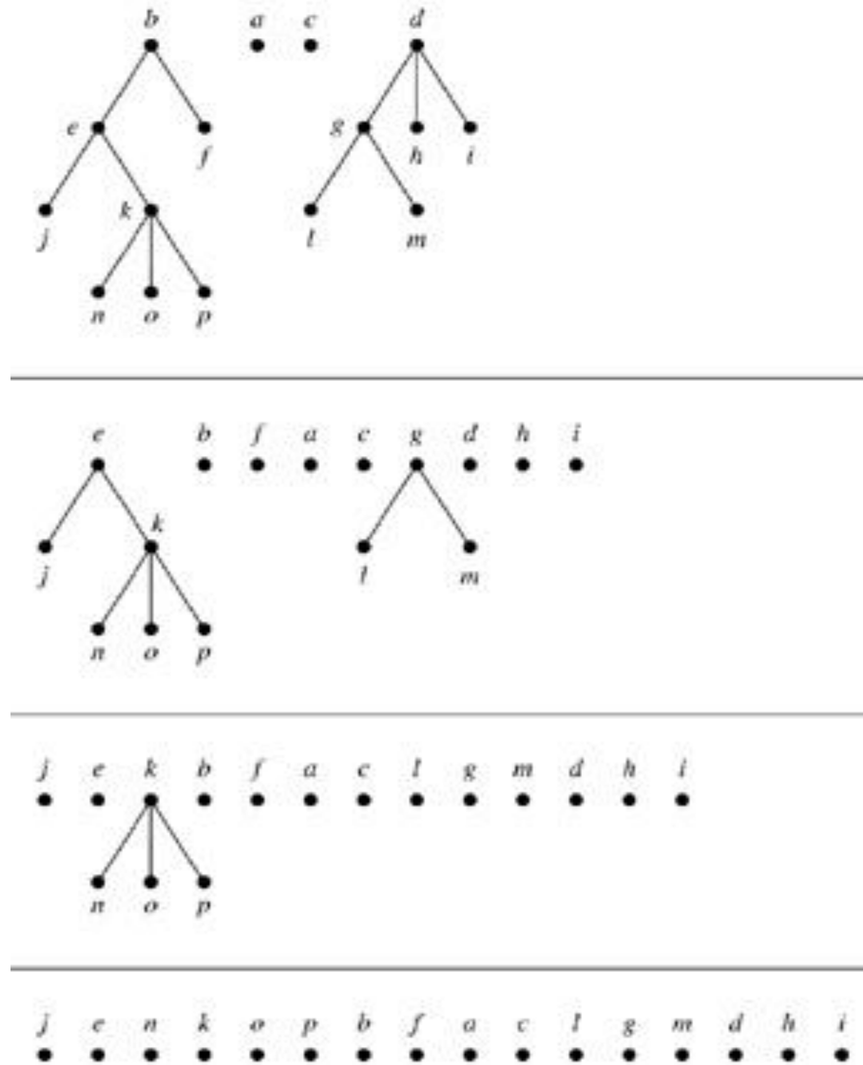
Inorder traversal: Left → Root → Right



Example 3. In which order does a preorder traversal visit the vertices in the ordered rooted tree T shown below?

Sol:





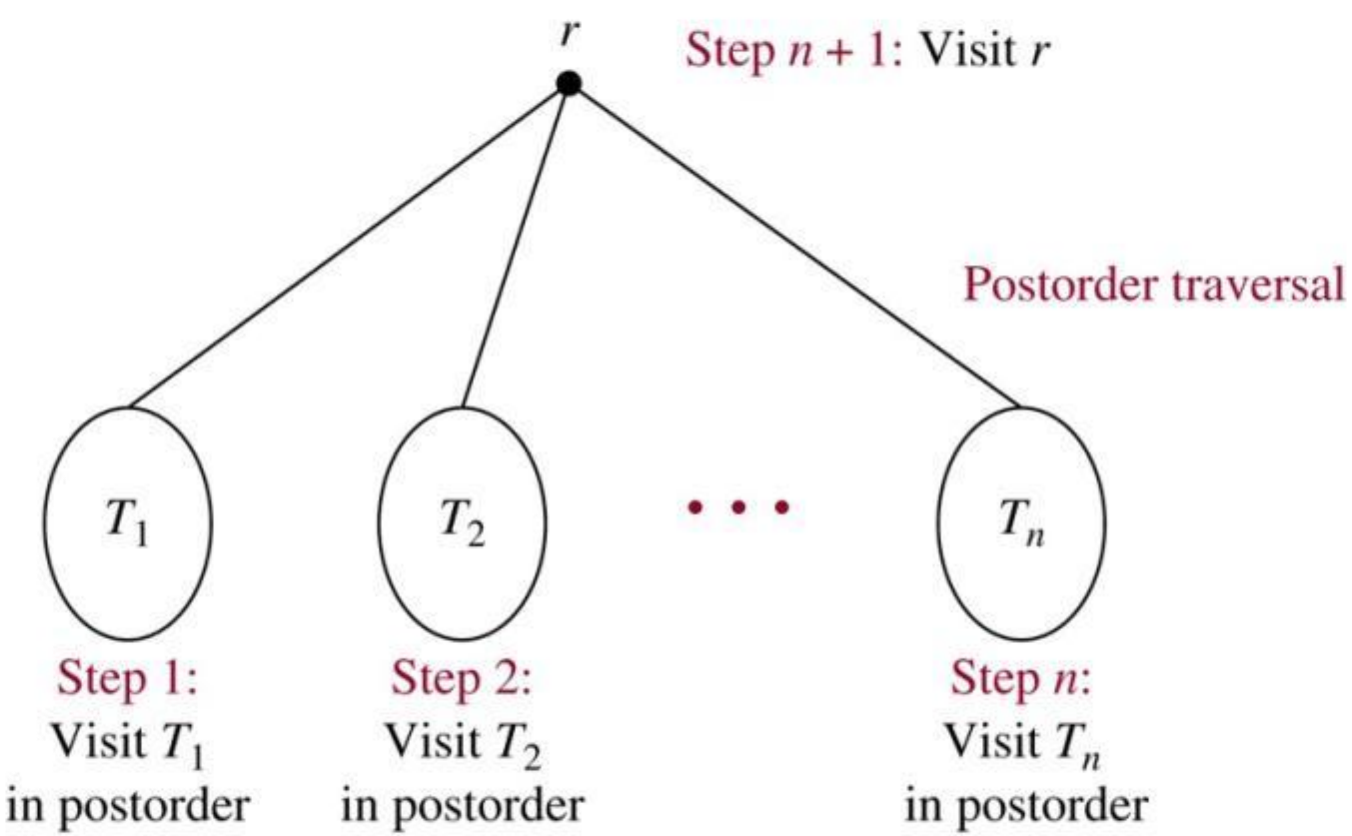
Tree Traversal

Algorithm 2 (Inorder Traversal)

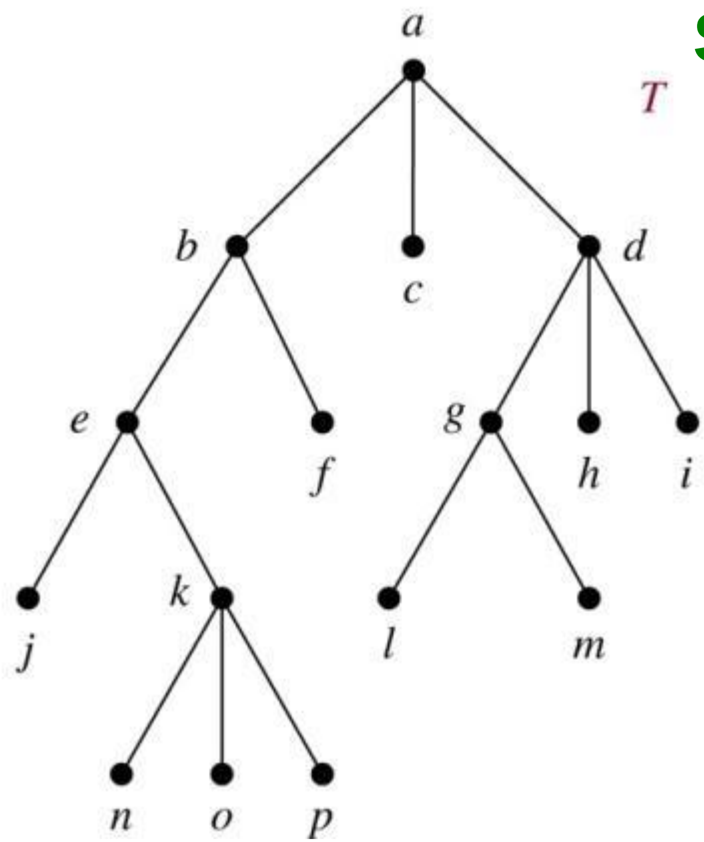
Procedure *inorder*(T: ordered rooted tree)
r := root of T
If r is a leaf **then** list r
else
begin
    l := first child of r from left to right
    $T(l)$:= subtree with l as its root
    inorder($T(l)$)
    list r
    for each child c of r except for l from left to right
        $T(c)$:= subtree with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

Tree Traversal

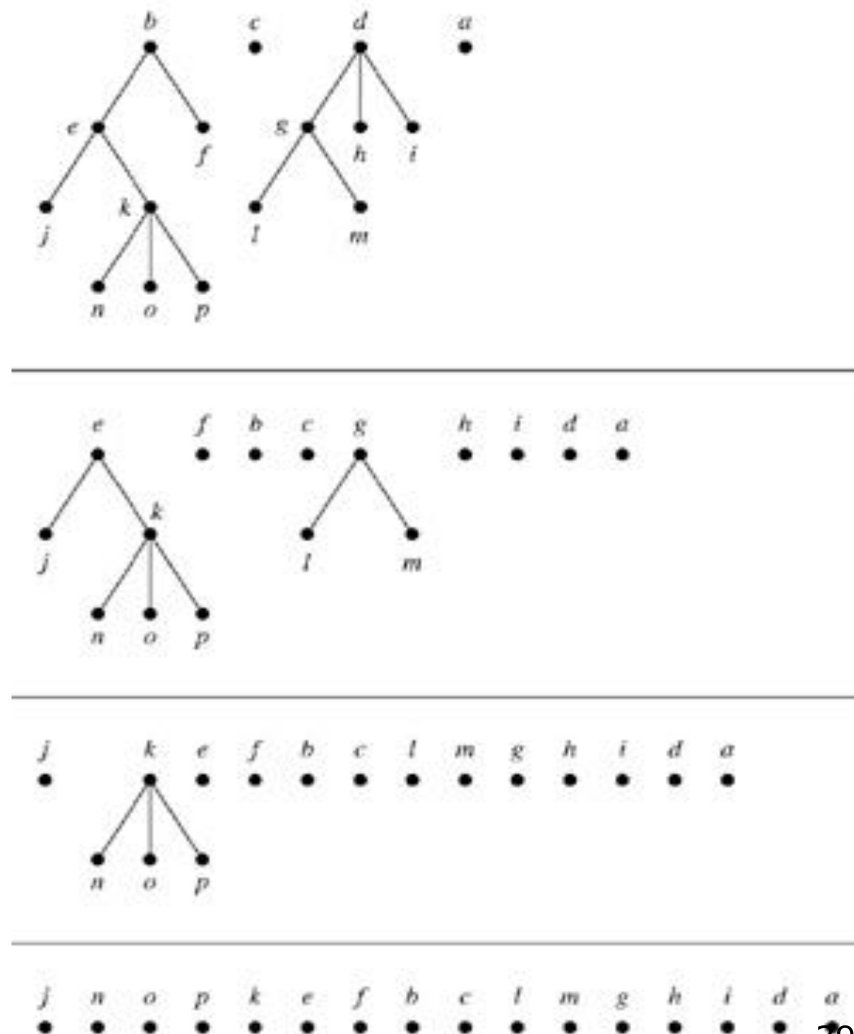
Postorder traversal: Left → Right → Root



Example 4. In which order does a preorder traversal visit the vertices in the ordered rooted tree T shown below?



Sol:



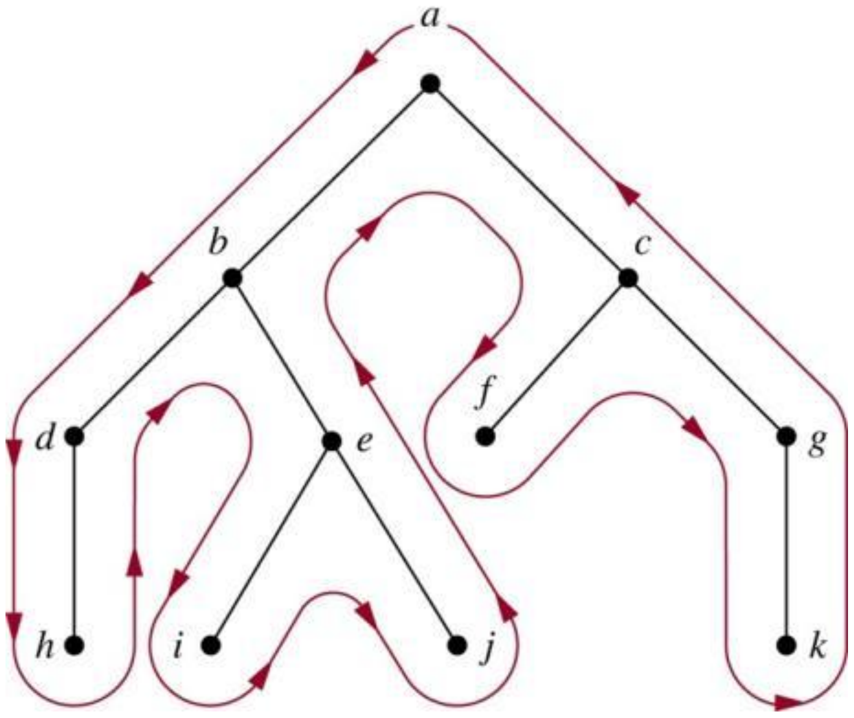
Tree Traversal

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 Traversal

Preorder: curve
Inorder: curve internal list
Postorder: curve



Preorder:
$a, b, d, h, e, i, j, c, f, g, k$

Inorder:
$h, d, b, i, e, j, a, f, c, k, g$

Postorder:
$h, d, i, j, e, b, f, k, g, c, a$

Tree Traversal

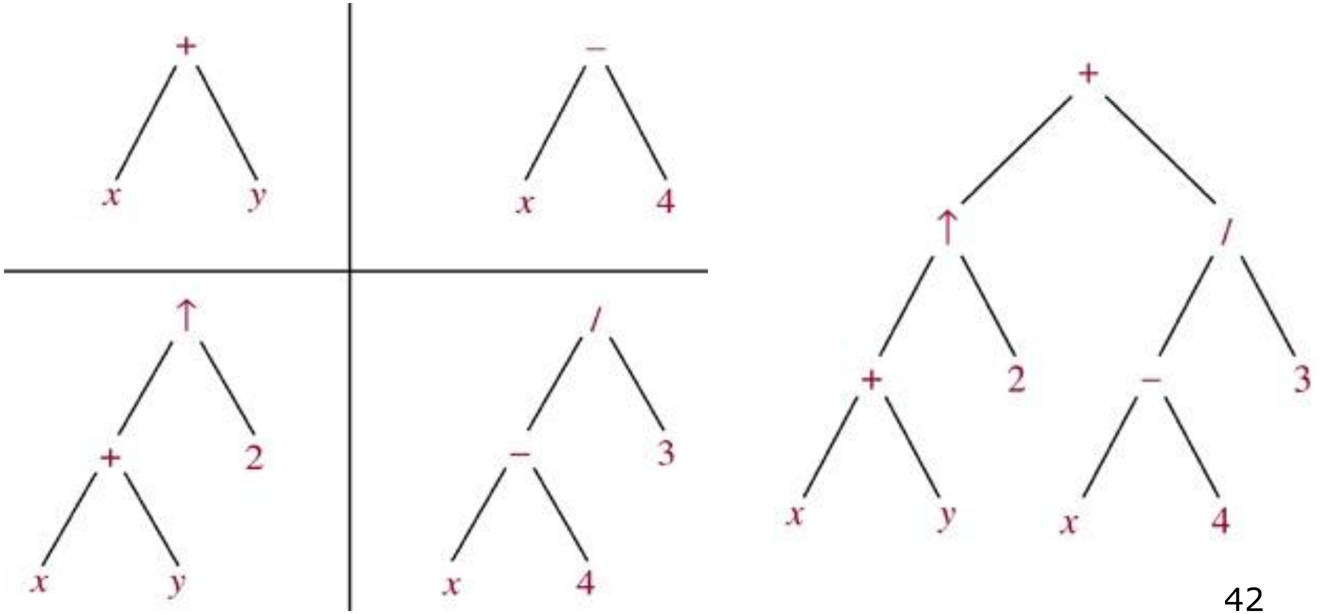
Infix, Prefix, and Postfix Notation

We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees.

Example 1: Find the ordered rooted tree for: $((x+y)\uparrow 2)+((x-4)/3)$.

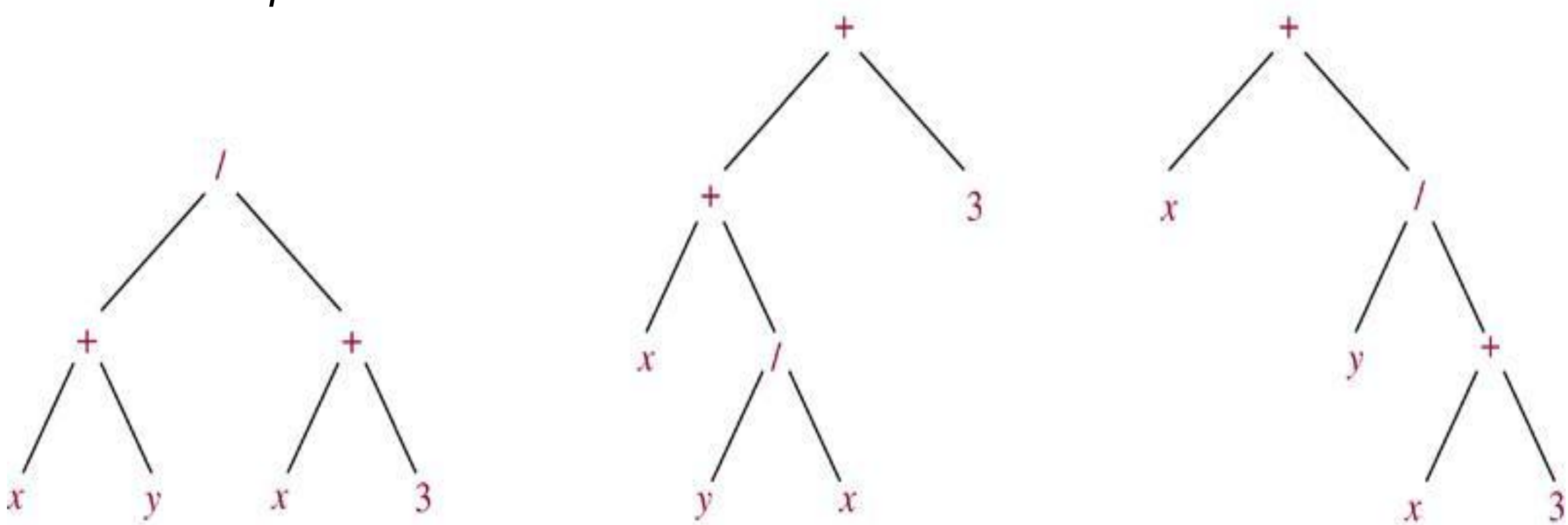
Sol.

leaf:
variable
internal vertex:
operation on its left and right subtrees



Tree Traversal

The following binary trees represent the expressions:
(x+y)/(x+3), (x+(y/x))+3, x+(y/(x+3)).
All their inorder traversals lead to x+y/x+3 ⇒ ambiguous
⇒ need parentheses



Infix form: An expression obtained when we traverse its rooted tree with <u>inorder</u>.

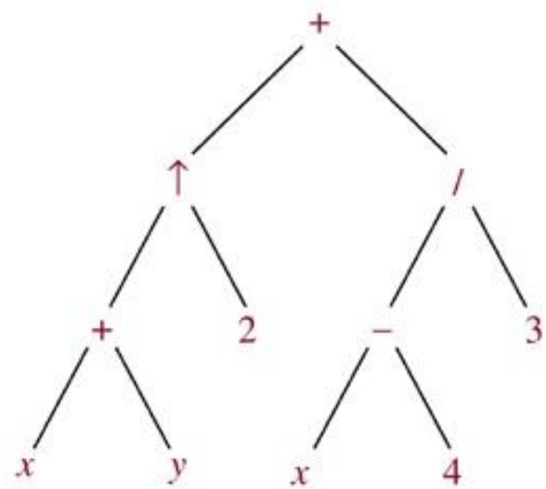
Prefix form: … … by <u>preorder</u>. (also named Polish notation)

Postfix form: … … by <u>postorder</u>. (reverse Polish notation)

Tree Traversal

Example 6 What is the prefix form for $((x+y)\uparrow 2)+((x-4)/3)$?

Sol.



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

Example 8 What is the postfix form of the expression $((x+y)\uparrow 2)+((x-4)/3)$?

Sol. $x\ \ y + 2 \uparrow x\ 4 - 3\ /\ +$

Note. An expression in prefix form or postfix form is unambiguous, so no parentheses are needed.

Tree Traversal

Example 7 What is the value of the prefix expression + – * 2 3 5 / ↑ 2 3 4?

Sol.

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

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

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

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

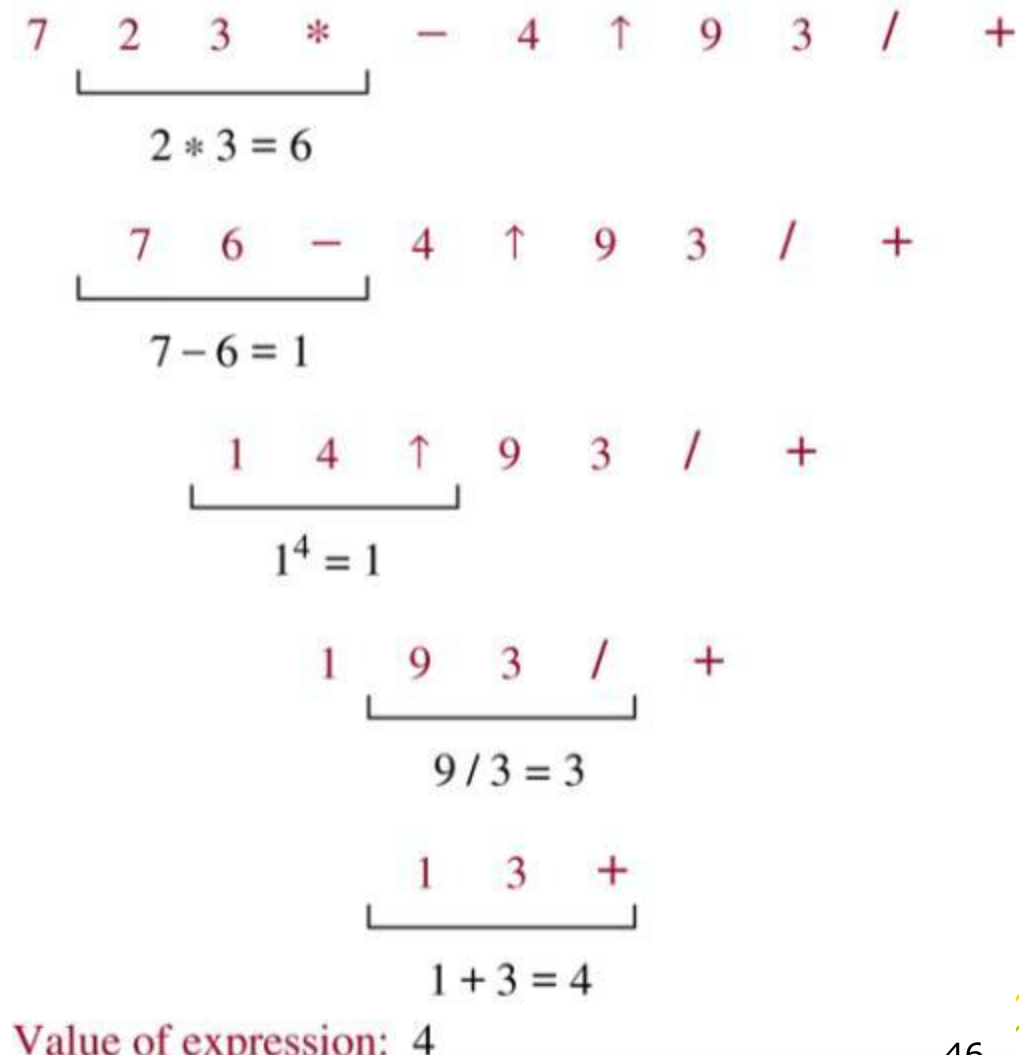
$$\underbrace{+\ 1\ 2}_{1 + 2 = 3}$$

Value of expression: 3

Tree Traversal

Example 9 What is the value of the postfix expression
7 2 3 * – 4 ↑ 9 3 / +?

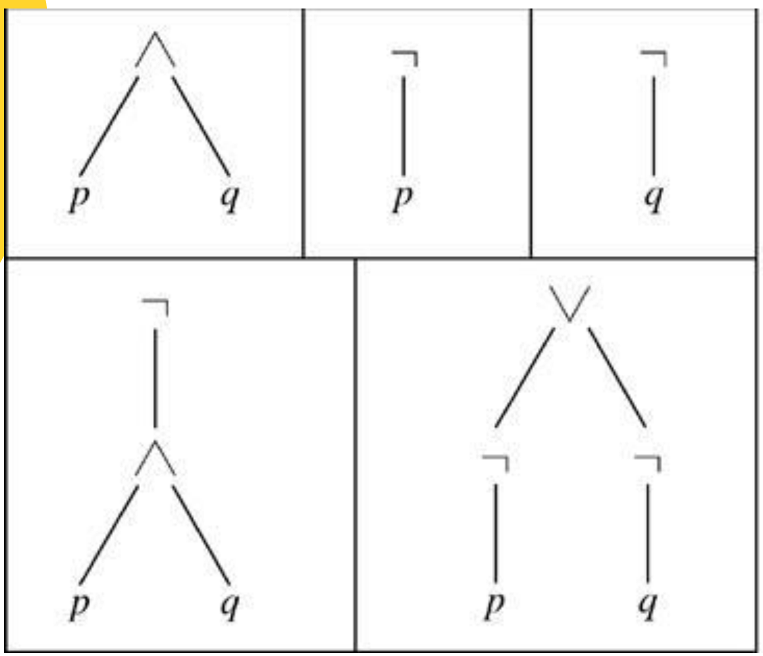
Sol.

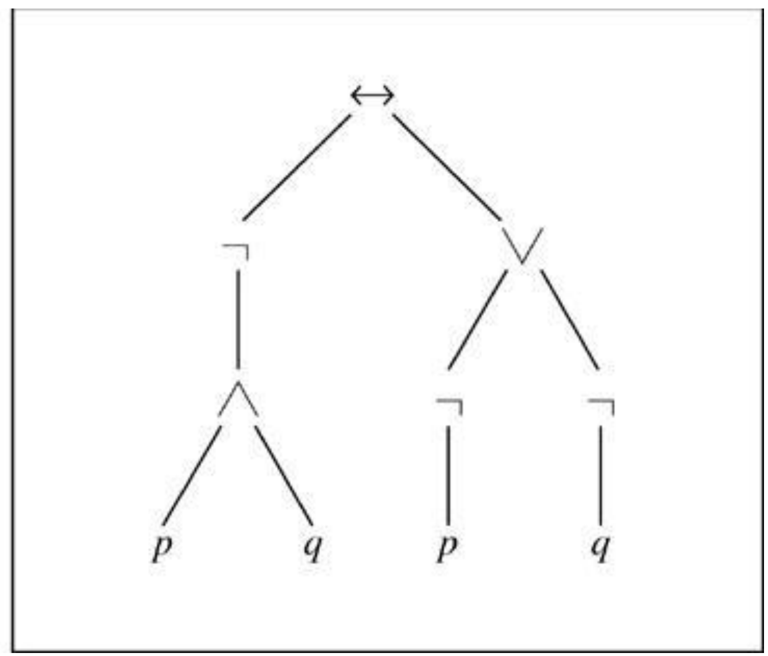


Tree Traversal

Example 10 Find the ordered rooted tree representing the compound proposition $(\neg(p \wedge q)) \leftrightarrow (\neg p \vee \neg q)$. Then use this rooted tree to find the prefix, postfix, and infix forms of this expression.

Sol.





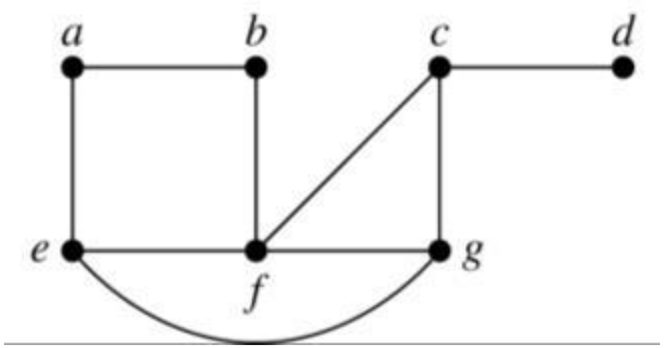
prefix: $\leftrightarrow\ \neg\ \wedge\ p\ q\ \vee\ \neg\ p\ \neg\ q$
postfix: $p\ q\ \wedge\ \neg\ p\ \neg\ q\ \neg\ \vee\ \leftrightarrow$

infix: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$

Spanning Trees

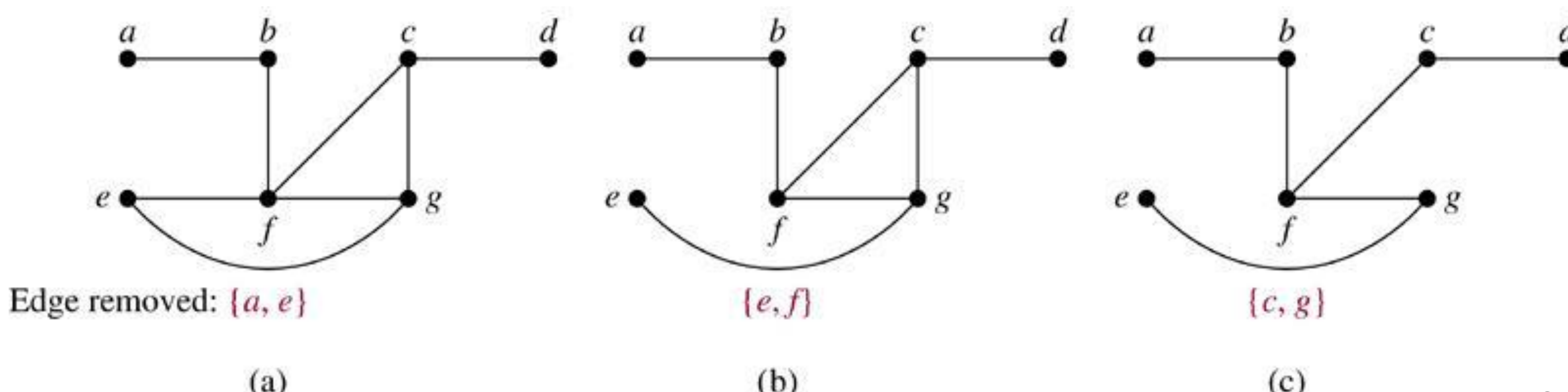
Recall (session 10): Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.

Example 1 Find a spanning tree of G.



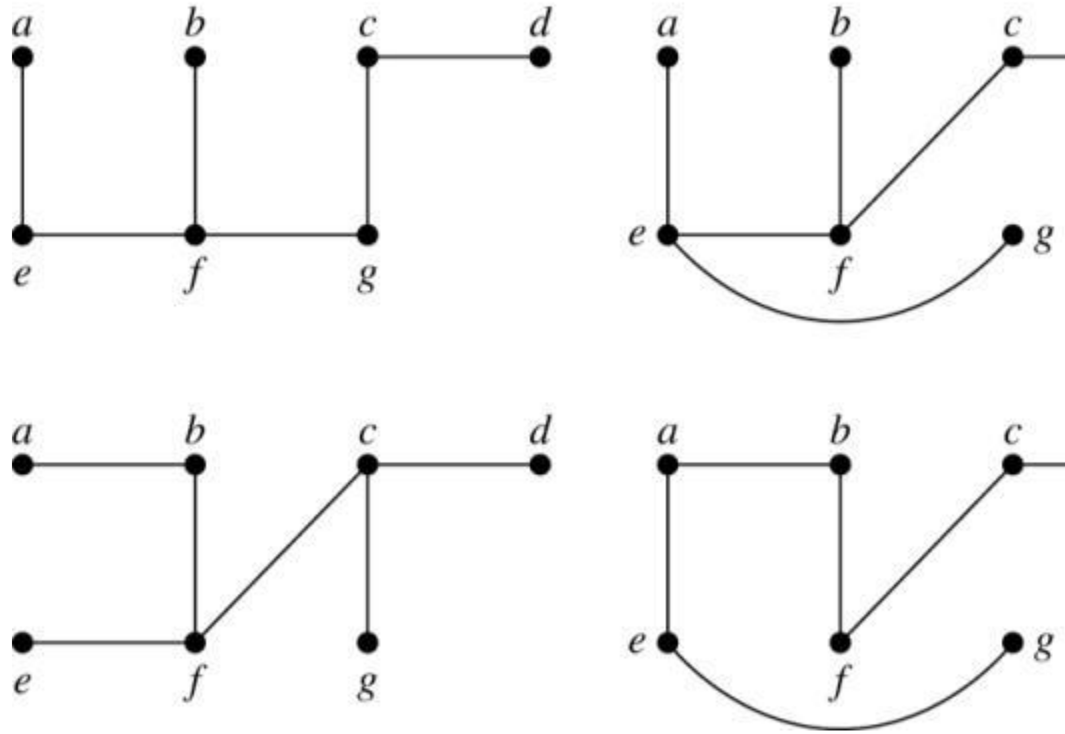
Sol.

Remove an edge from any circuit.
(repeat until no circuit exists)



Tree Traversal

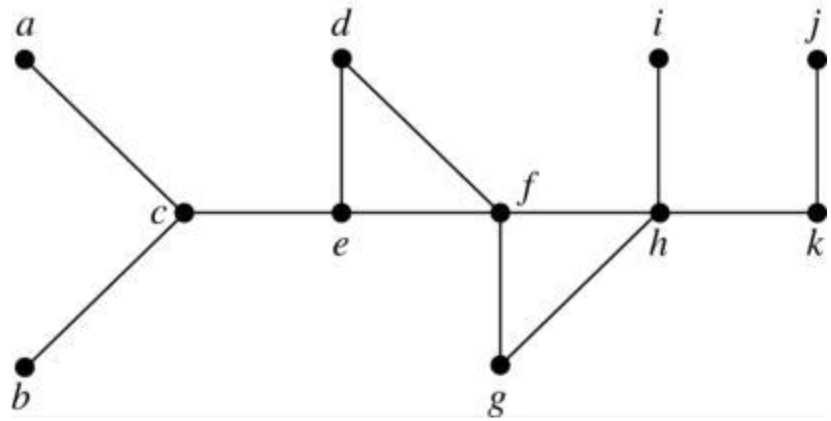
Four spanning trees of G:



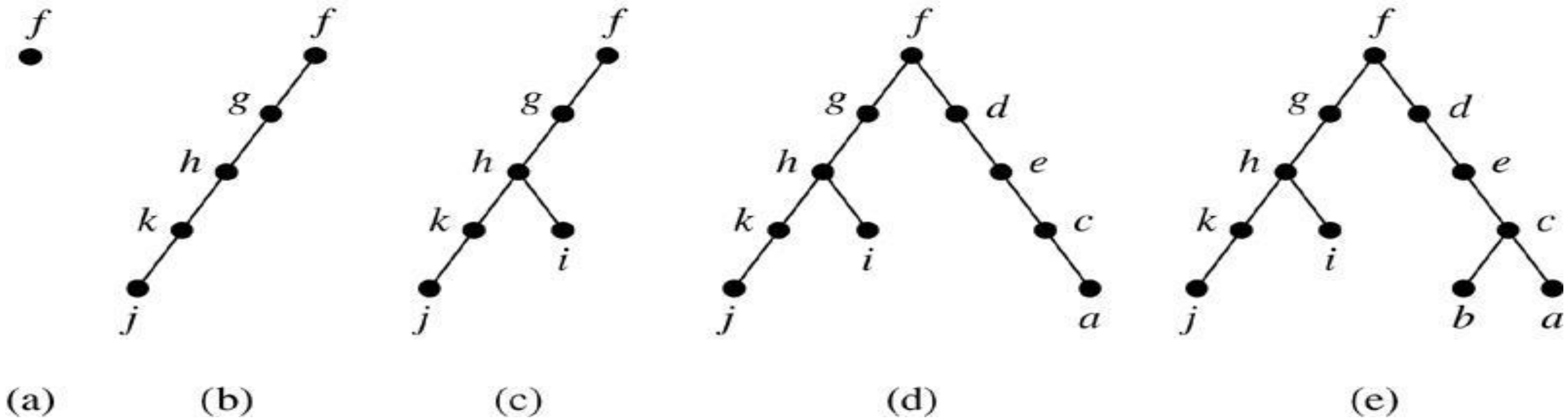
Thm 1: A simple graph is connected if and only if it has a spanning tree

Depth-First Search (DFS)

Example 3 Use depth-first search to find a spanning tree for the graph.



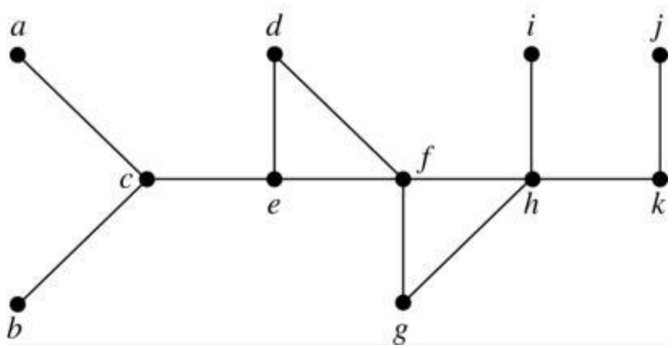
Sol. (arbitrarily start with the vertex *f*)

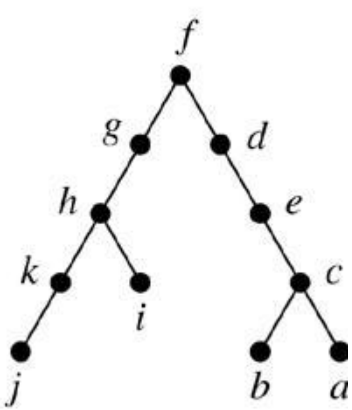


Tree Traversal

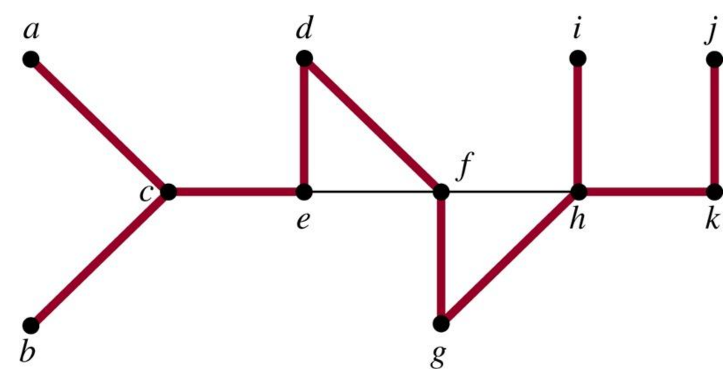
The edges selected by DFS of a graph are called tree edges. All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called back edges.

Example 4





⇒



The tree edges (red)
and back edges (black)

Tree Traversal

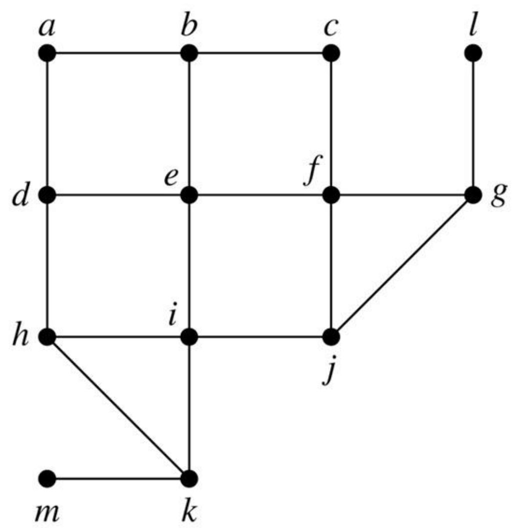
Algorithm 1 (Depth-First Search)

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
begin
    add vertex w and edge $\{v, w\}$ to T
    visit(w)
end

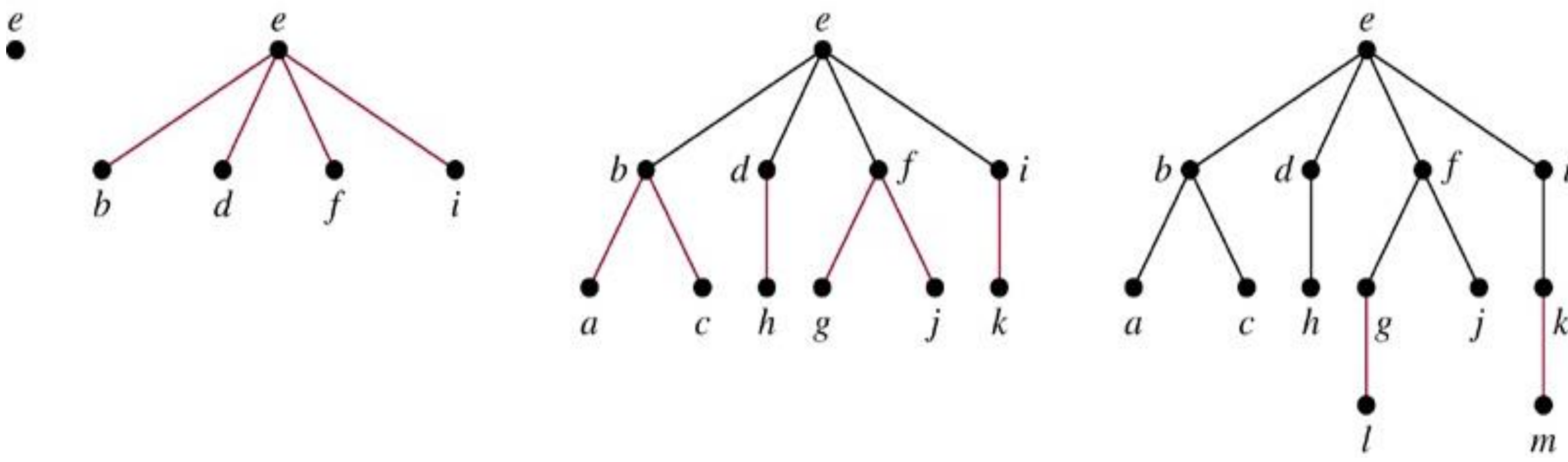
Tree Traversal

Breadth-First Search (BFS)

Example 5 Use breadth-first search to find a spanning tree for the graph.



Sol. (arbitrarily start with the vertex e)



Tree Traversal

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
begin
    remove the first vertex v from L
    for each neighbor w of v
        if w is not in L and not in T **then**
        begin
            add w to the end of the list L
            add w and edge $\{v, w\}$ to T
        end
end

Homework

- **10.1**: 2,4,6,8,10,22,28,32 in pages 694,694.
- **10.2**: 2,6 in page 708; 20,22in page 709.
- **10.3**: 2,4,6,8,12,14,18 in pages 722, 723.
- **10.4**: 2,4,6,8,10 in pages 734, 735.

Deadline: Dec, 10th 2014

- **REVIEW FOR FINAL EXAMINATION**

Review for final examination

- Date: see announcement from OAA
- Time: 120 mins or more
- Form: writing, analysis, answer the questions and program.
- Content: all sessions, BUT focus on sessions 2, 4, 7, 8, 9, 10, 11.
- Questions?