厦門大學信息学院
School of Informatics Xiamen University
（特色化示范性软件学院）
(National Characteristic Demonstration Software School)

《离散数学》

Chapter 11：Trees（I）

王晓黎

2025年12月17日

Outline

- Introduction to Trees
- Tree Traversal
- Spanning Trees
- Minimum Spanning Trees

Introduction to Trees

厦門大學信息学院
School of Informatics Xiamen University
(特色化示范性软件学院)
(National Characteristic Demonstration Software School)

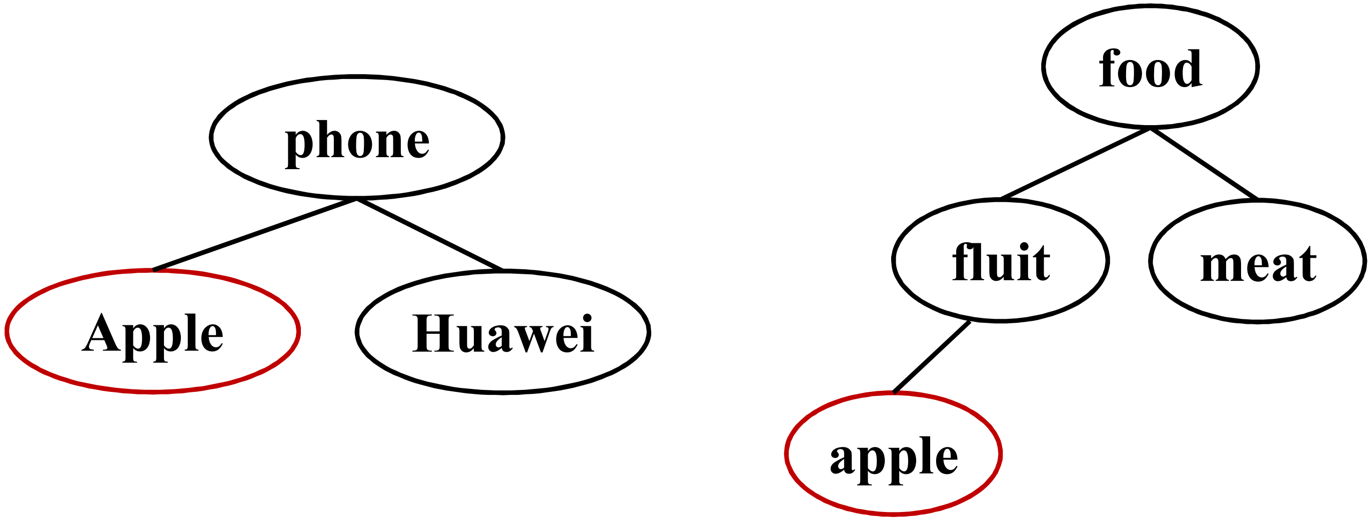
- Introduction to Trees
- Rooted Trees
- Trees as Models
- Properties of Trees

A Motivated Example

- Type in “buy apple to eat” using the search engine

Apples are so expensive, so why do so many people still buy them?

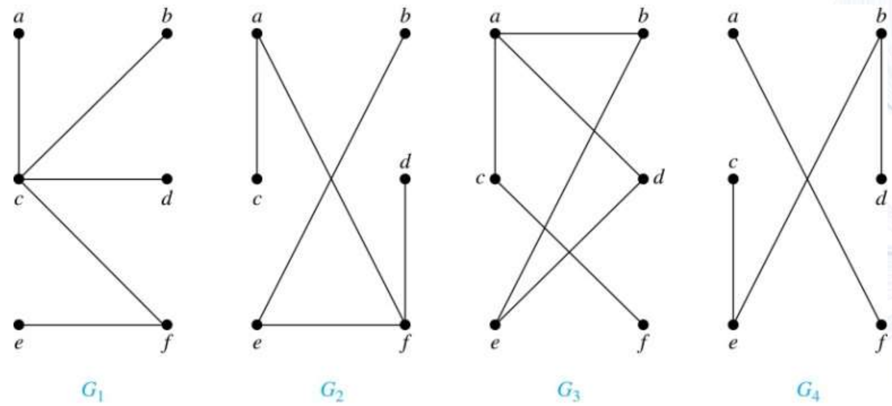
Apples are so expensive this year, they're even pricier than meat.



Trees

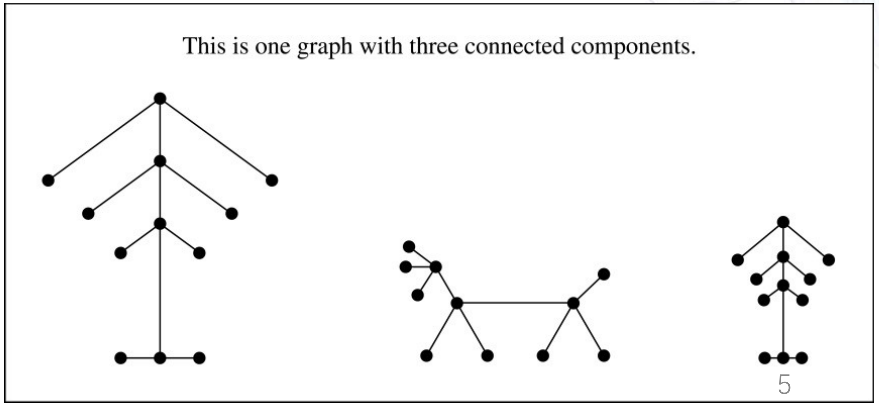
Definition: A *tree* (树) is a connected undirected graph with no simple circuits.

Example: Which of these graphs are trees?



Solution: G_1 and G_2 are trees - both are connected and have no simple circuits. Because *e*, *b*, *a*, *d*, *e* is a simple circuit, G_3 is not a tree. G_4 is not a tree because it is not connected.

Definition: A *forest* (森林) is a graph that has no simple circuit, but is not connected. Each of the connected components in a forest is a tree.



Trees (continued)

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

Proof: Assume that T is a tree. Then T is connected with no simple circuits. Hence, if x and y are distinct vertices of T, there is a simple path between them (by Theorem 1 of Section 10.4). This path must be unique - for if there were a second path, there would be a simple circuit in T. Hence, there is a unique simple path between any two vertices of a tree.

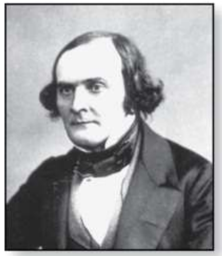
Now assume that there is a unique simple path between any two vertices of a graph T. Then T is connected because there is a path between any two of its vertices. Furthermore, T can have no simple circuits since if there were a simple circuit, there would be two paths between some two vertices.

Hence, a graph with a unique simple path between any two vertices is a tree. ◀

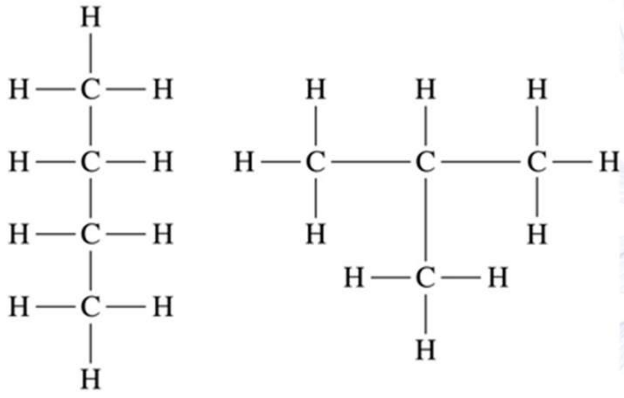
THEOREM 1 There is a simple path between every pair of distinct vertices of a connected undirected graph.

Trees as Models

- Trees are used as models in computer science, chemistry, geology, botany, psychology, and many other areas
- Trees were introduced by the mathematician Cayley in 1857 in his work counting the number of isomers of saturated hydrocarbons. The two isomers of butane are shown at the right
- The organization of a computer file system into directories, subdirectories, and files is naturally represented as a tree
- Trees are used to represent the structure of organizations

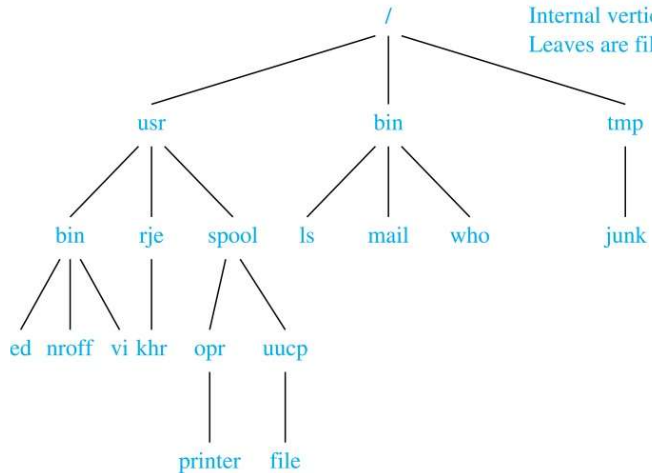


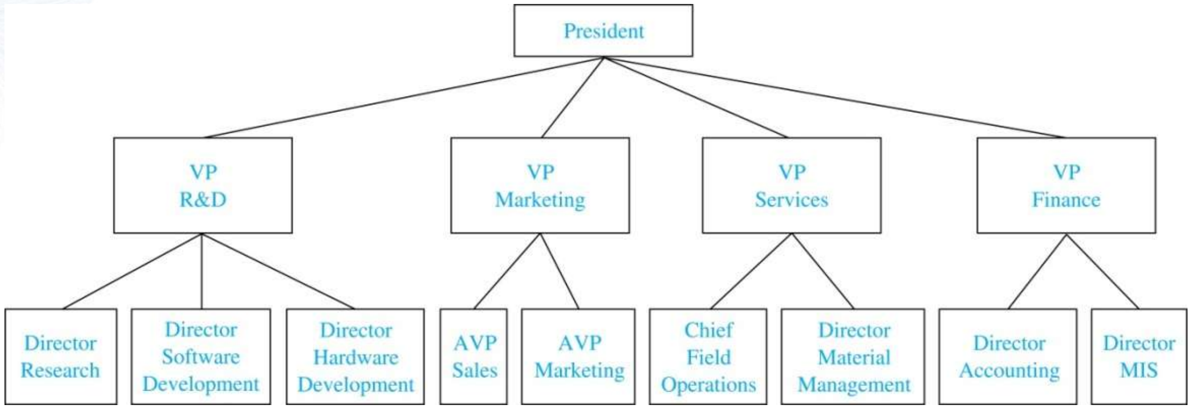
Arthur Cayley
(1821-1895)



Butane   Isobutane

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

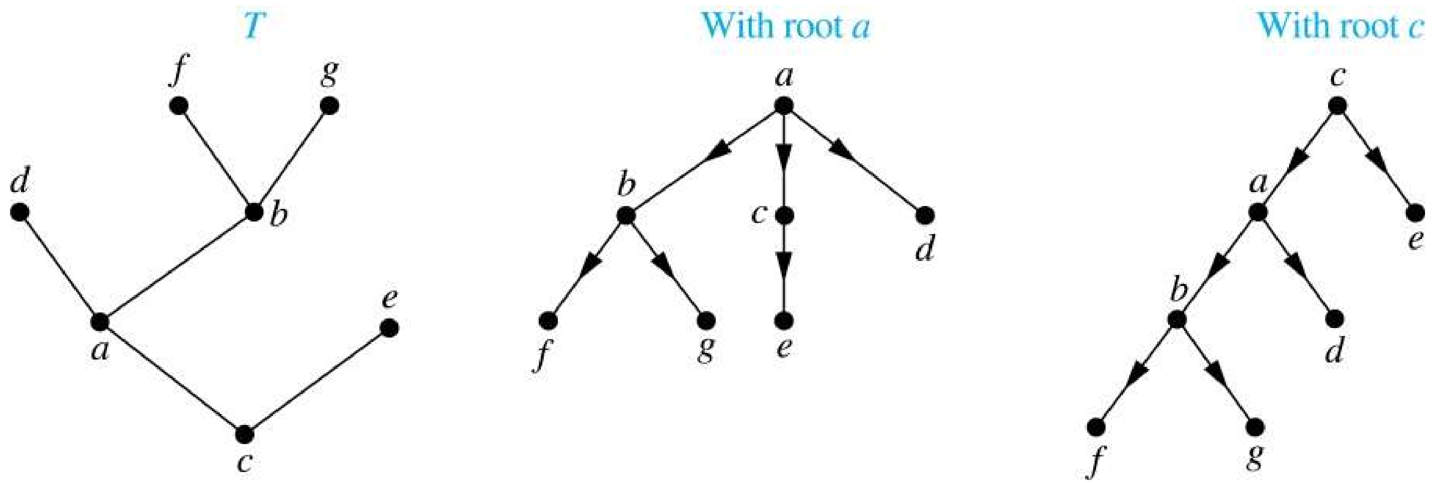




Rooted Trees

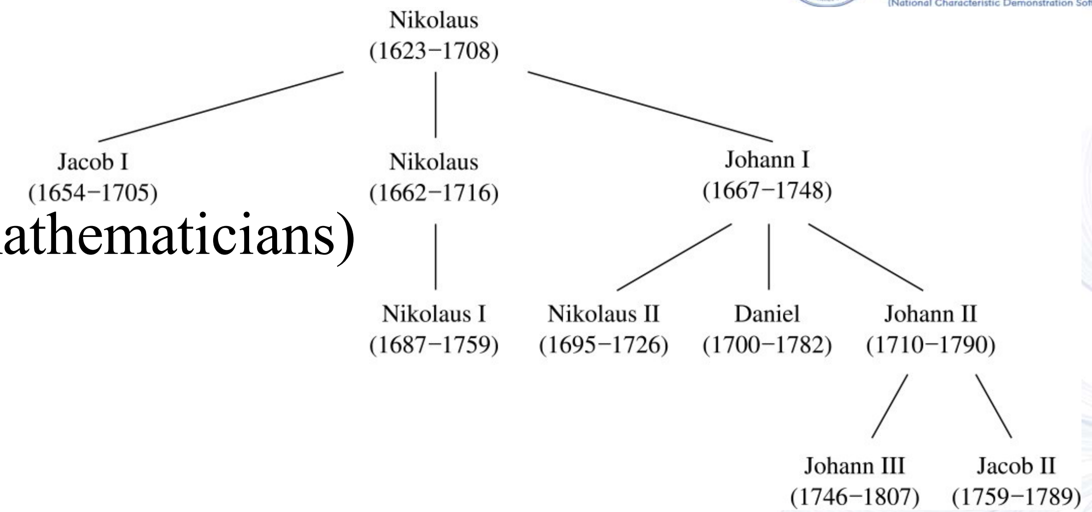
Definition: A *rooted tree* (有根树) is a tree in which one vertex has been designated as the *root* and every edge is directed away from the root.

An unrooted tree is converted into different rooted trees when different vertices are chosen as the root.



Rooted Tree Terminology

- Terminology for rooted trees is a mix from botany and genealogy (such as the Bernoulli family of mathematicians)



- If *v* is a vertex of a rooted tree other than the root, the *parent* of *v* is the unique vertex *u* such that there is a directed edge from *u* to *v*. When *u* is a parent of *v*, *v* is called a *child* of *u*. Vertices with the same parent are called *siblings*.
- The *ancestors* of a vertex are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root. The *descendants* of a vertex *v* are those vertices that have *v* as an ancestor.
- A vertex of a rooted tree with no children is called a *leaf*. Vertices that have children are called *internal vertices*.
- If *a* is a vertex in a tree, the *subtree* with *a* as its root is the subgraph of the tree consisting of *a* and its descendants and all edges incident to these descendants.

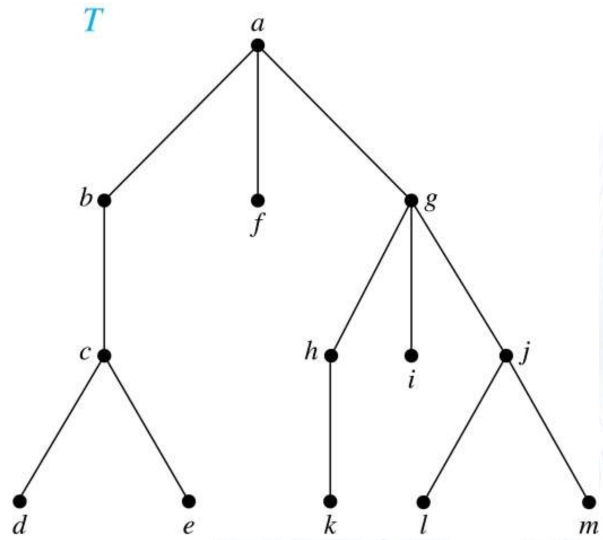
Terminology for Rooted Trees

Example: In the rooted tree T (with root a):

(i) Find the parent of c, the children of g, the siblings of h, the ancestors of e, and the descendants of b.

(ii) Find all internal vertices and all leaves.

(iii) What is the subtree rooted at g?

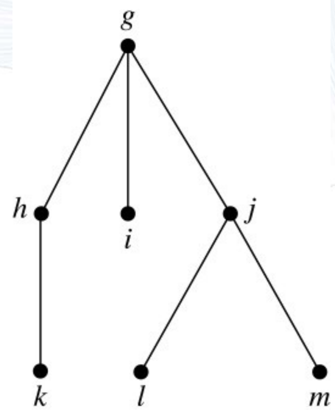


Solution:

(i) The parent of c is b. The children of g are h, i, and j. The siblings of h are i and j. The ancestors of e are c, b, and a. The descendants of b are c, d, and e.

(ii) The internal vertices are a, b, c, g, h, and j. The leaves are d, e, f, i, k, l, and m.

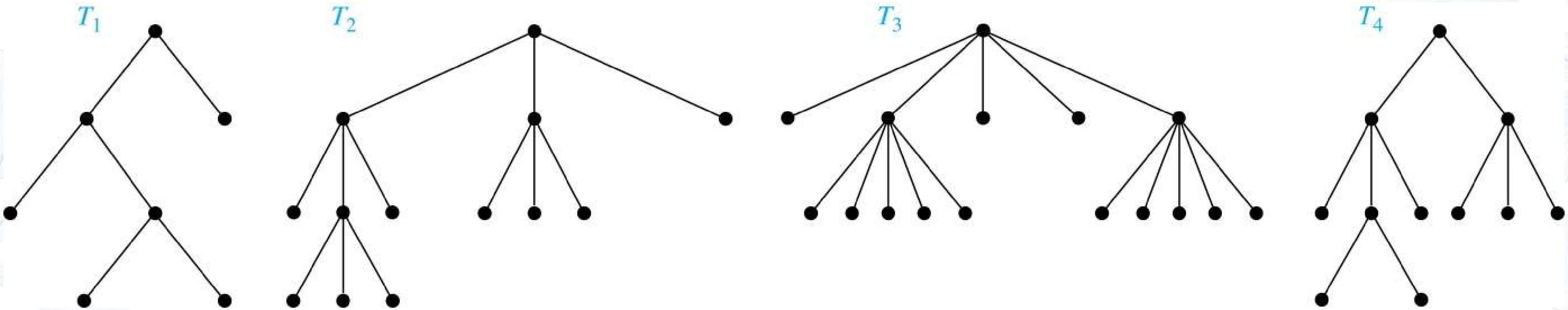
(iii) We display the subtree rooted at g.



m-ary Rooted Trees

Definition: 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*.

Example: Are the following rooted trees full m-ary trees for some positive integer m?



Solution:

T_1 is a full binary tree because each of its internal vertices has two children.

T_2 is a full 3-ary tree because each of its internal vertices has three children.

In T_3 each internal vertex has five children, so T_3 is a full 5-ary tree.

T_4 is not a full m-ary tree for any m because some of its internal vertices have two children and others have three children.

Ordered Rooted Trees

Definition: An *ordered rooted tree* (有序根树) is a rooted tree where the children of each internal vertex are ordered.

- We draw ordered rooted trees so that the children of each internal vertex are shown in order from left to right.

Definition: A *binary tree* (二叉树) is an ordered rooted tree where each internal vertex has at most two children. If an internal vertex of a binary tree has two children, the first is called the *left child* and the second the *right child*. The tree rooted at the left child of a vertex is called the *left subtree* of this vertex, and that rooted at the right child of a vertex is called the *right subtree* of this vertex.

Example: Consider the binary tree T.

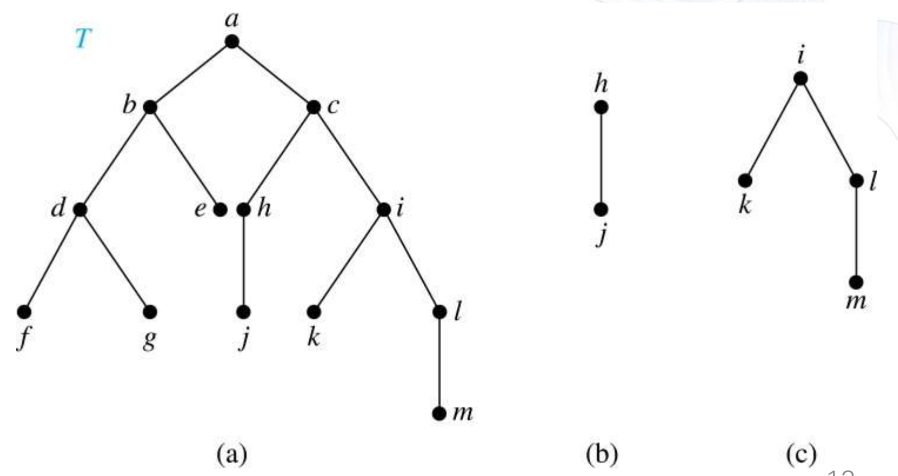
(i) What are the left and right children of d?

(ii) What are the left and right subtrees of c?

Solution:

(i) The left child of d is f and the right child is g.

(ii) The left and right subtrees of c are displayed in (b) and (c).



Properties of Trees

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

Proof **(*by mathematical induction*):**

BASIS STEP: When $n=1$, a tree with one vertex has no edges. Hence, the theorem holds when $n=1$.

INDUCTIVE STEP: Assume that every tree with k vertices has $k-1$ edges.

Suppose that a tree T has $k+1$ vertices and that v is a leaf of T. Let w be the parent of v. Removing the vertex v and the edge connecting w to v produces a tree T' with k vertices. By the inductive hypothesis, T' has $k-1$ edges. Because T has one more edge than T', we see that T has k edges. This completes the inductive step. ◀

Counting Vertices in Full m-Ary Trees

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

Proof : Every vertex, except the root, is the child of an internal vertex. Because each of the i internal vertices has m children, there are mi vertices in the tree other than the root. Hence, the tree contains $n = mi + 1$ vertices. ◄

Counting Vertices in Full m-Ary Trees (continued)

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.

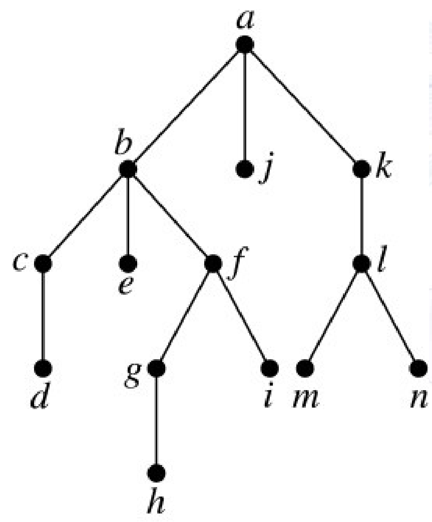
proofs of parts (ii) and (iii) are left as take-home assignments

***Proof* (*of part i*):** Solving for i in $n = mi + 1$ (from Theorem 3) gives $i = (n - 1)/m$. Since each vertex is either a leaf or an internal vertex, $n = l + i$. By solving for l and using the formula for i, we see that

$$l = n - i = n - (n - 1)/m = [(m - 1)n + 1]/m .$$ ◄

Level of vertices and height of trees

- When working with trees, we often want to have rooted trees where the subtrees at each vertex contain paths of approximately the same length
- To make this idea precise we need some definitions:
  - The *level* of a vertex in a rooted tree is the length of the unique path from the root to this vertex
  - The *height* of a rooted tree is the maximum of the levels of the vertices



Example:

(i) Find the level of each vertex in the tree to the right.

(ii) What is the height of the tree?

Solution:

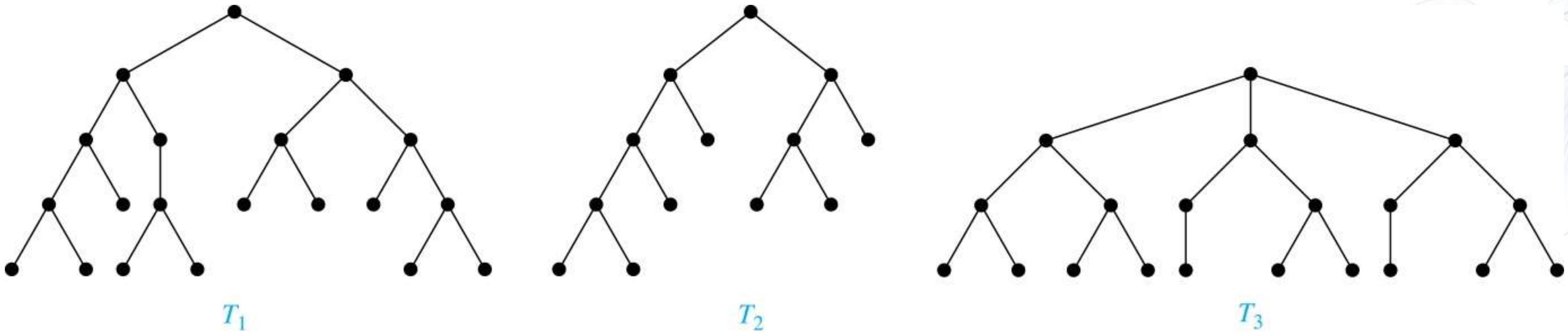
(i) The root a is at level 0. Vertices b, j, and k are at level 1.
Vertices c, e, f, and l are at level 2. Vertices d, g, i, m, and n are at level 3.
Vertex h is at level 4.

(ii) The height is 4, since 4 is the largest level of any vertex.

Balanced m-Ary Trees

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

Example: Which of the rooted trees shown below is balanced?



Solution: T_1 and T_3 are balanced, but T_2 is not because it has leaves at levels 2, 3, and 4.

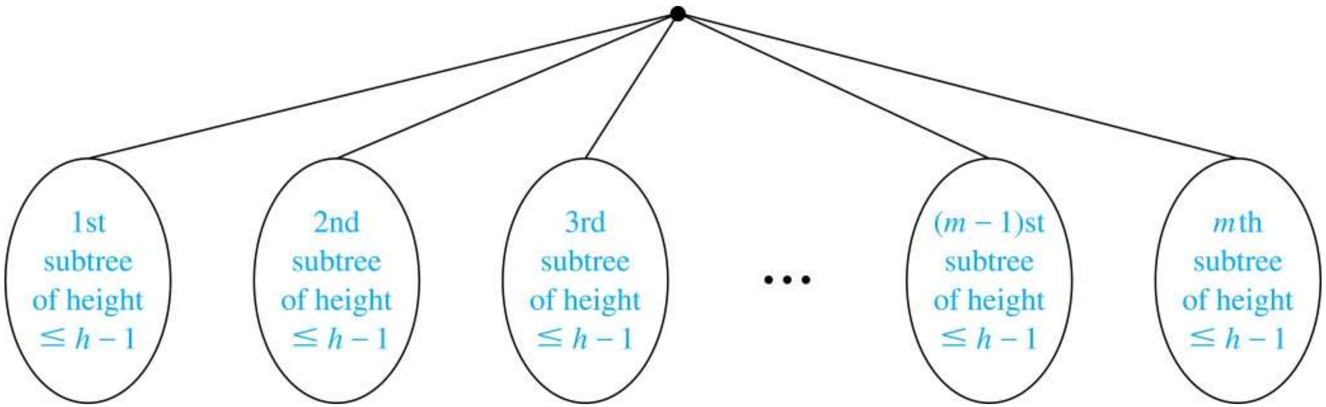
The Bound for the Number of Leaves in an m-Ary Tree

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

***Proof* (by mathematical induction on height):**

BASIS STEP: Consider an m-ary trees of height 1. The tree consists of a root and no more than m children, all leaves. Hence, there are no more than $m^1 = m$ leaves in an m-ary tree of height 1.

INDUCTIVE STEP: Assume the result is true for all m-ary trees of height $< h$. Let T be an m-ary tree of height h. The leaves of T are the leaves of the subtrees of T we get when we delete the edges from the root to each of the vertices of level 1.



Each of these subtrees has height $\leq h-1$. By the inductive hypothesis, each of these subtrees has at most m^{h-1} leaves. Since there are at most m such subtees, there are at most $m \cdot m^{h-1} = m^h$ leaves in the tree. ◀

Tree Traversal

厦門大學信息学院
School of Informatics Xiamen University
(特色化示范性软件学院)
(National Characteristic Demonstration Software School)

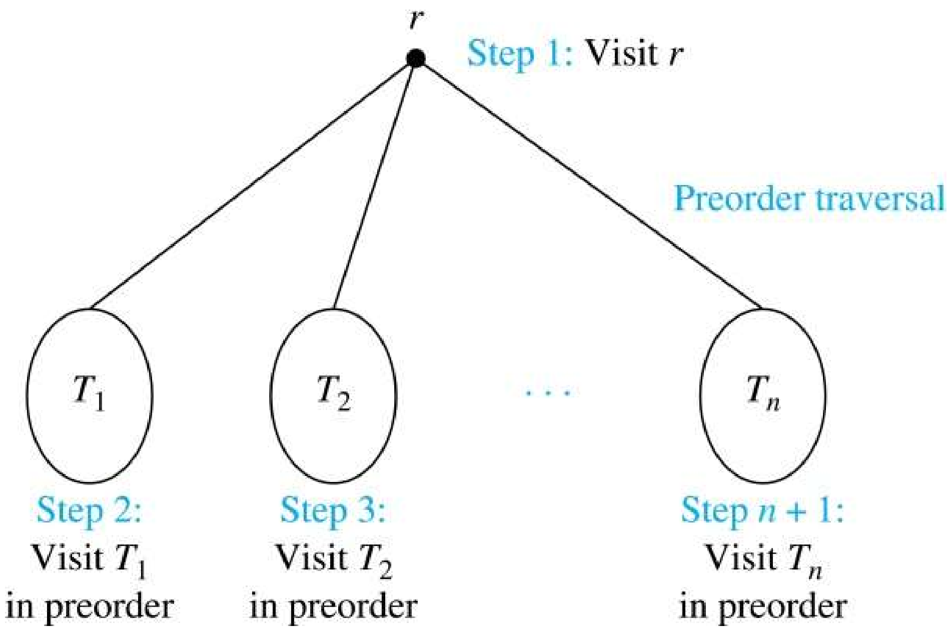
- Traversal Algorithms
- Infix, Prefix, and Postfix Notation

Tree Traversal

- Procedures for systematically visiting every vertex of an ordered tree are called *traversals* (遍历)
- The three most commonly used *traversals* are *preorder traversal* (前序遍历), *inorder traversal* (中序遍历), and *postorder traversal* (后序遍历)

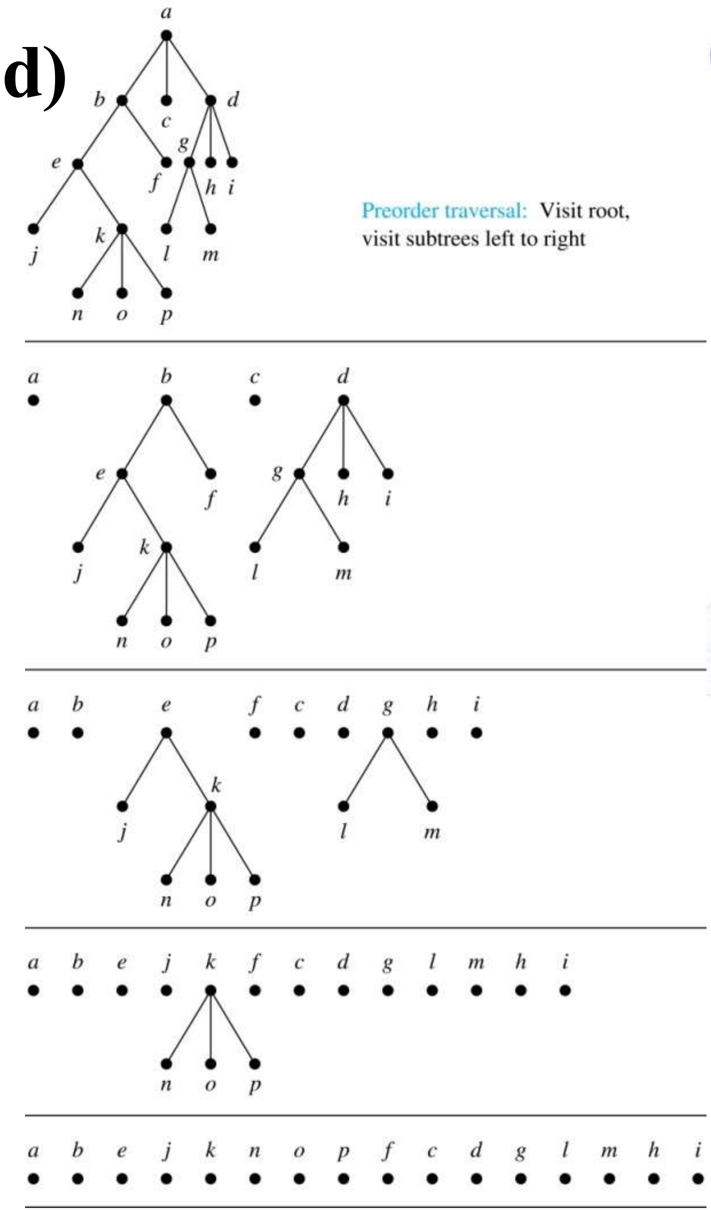
Preorder Traversal

Definition: 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 of r from left to right in T. The preorder traversal begins by visiting r, and continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.



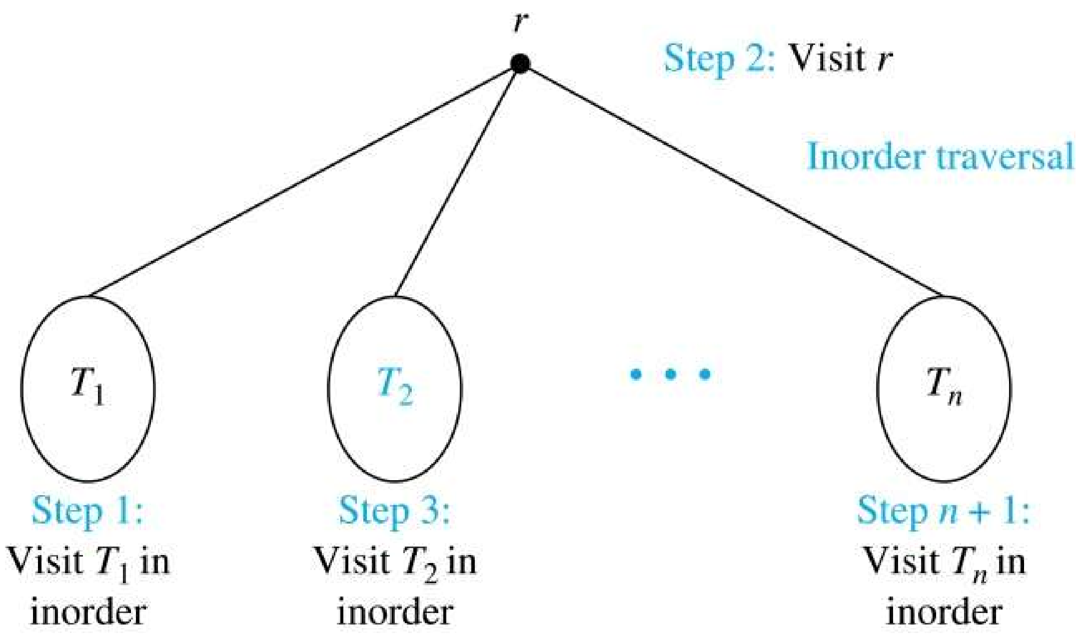
Preorder Traversal (continued)

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 root
    preorder(*T*(*c*))



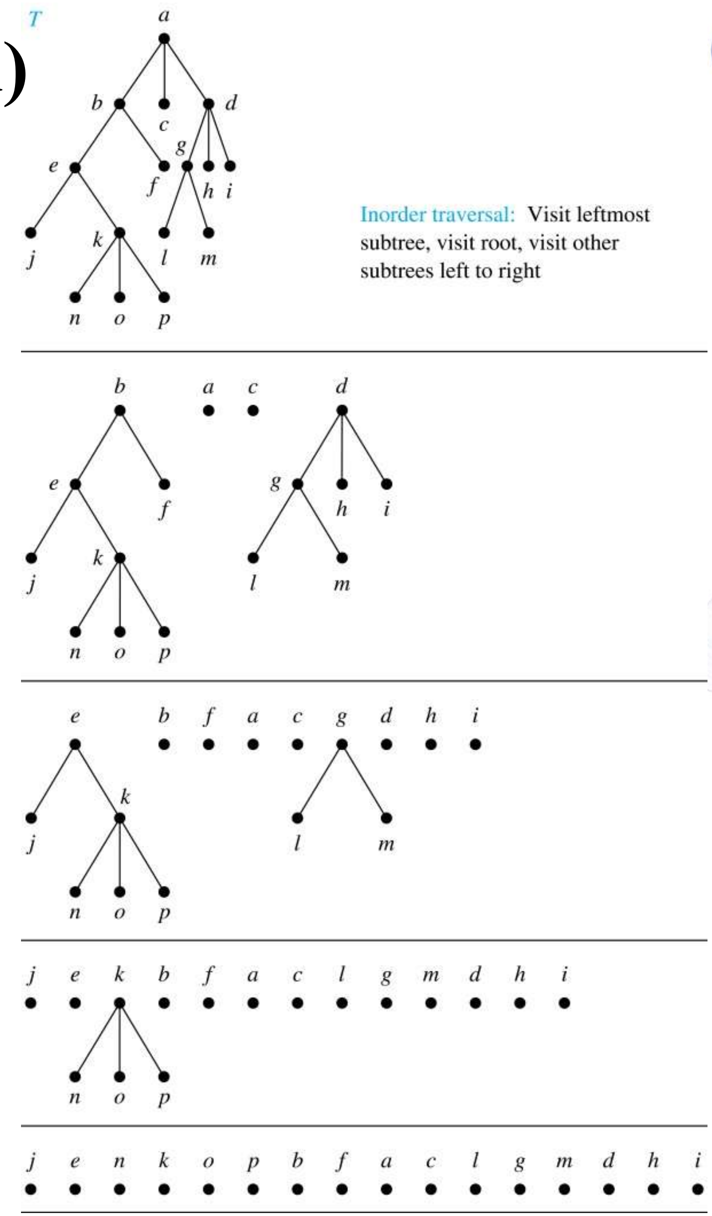
Inorder Traversal

Definition: 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 of r from left to right in T. The inorder traversal begins by traversing T_1 in inorder, then visiting r, and continues by traversing T_2 in inorder, and so on, until T_n is traversed in inorder.



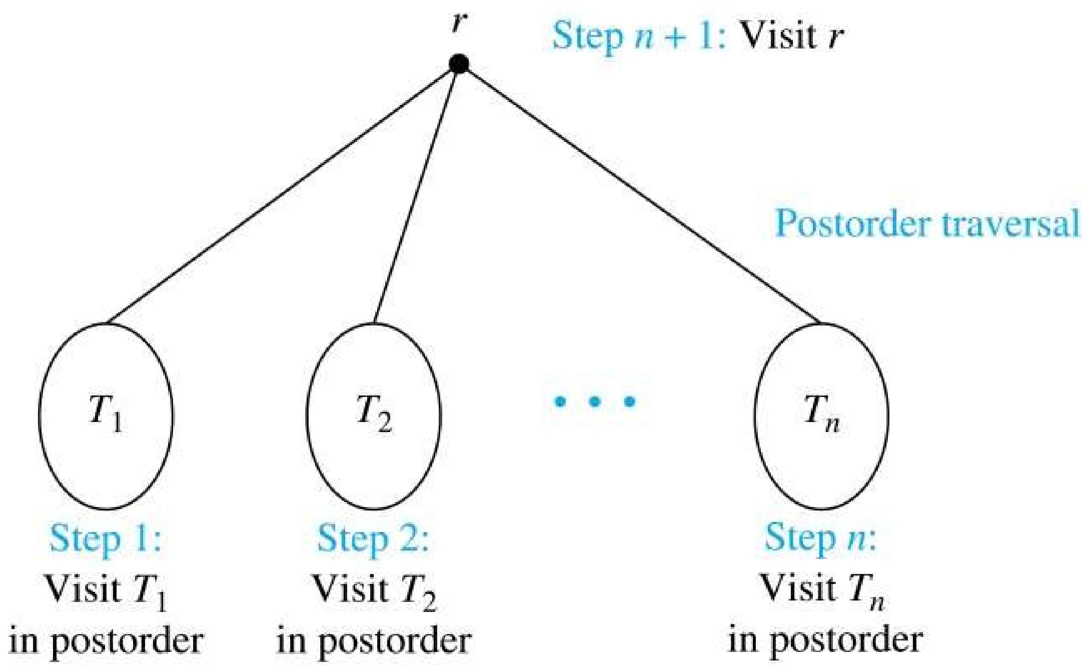
Inorder Traversal (continued)

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* from left to right
    T(*c*) := subtree with *c* as root
    inorder(*T*(*c*))



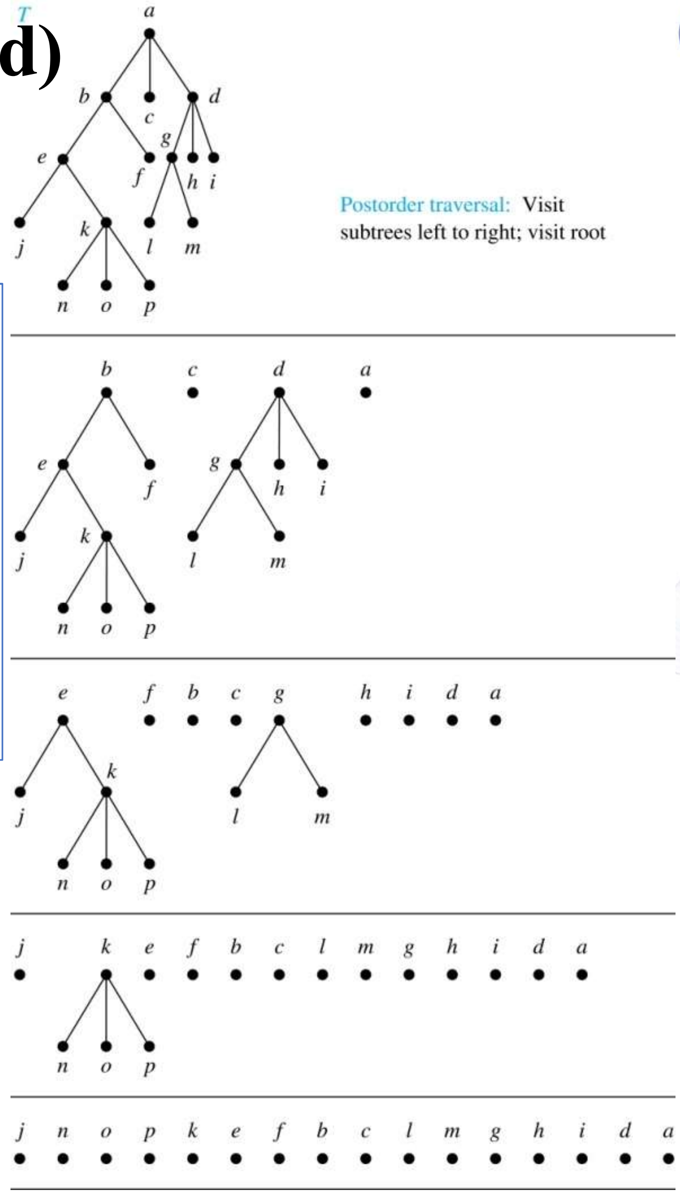
Postorder Traversal

Definition: 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 of r from left to right in T. The postorder traversal begins by traversing T_1 in postorder, then T_2 in postorder, and so on, after T_n is traversed in postorder, r is visited.



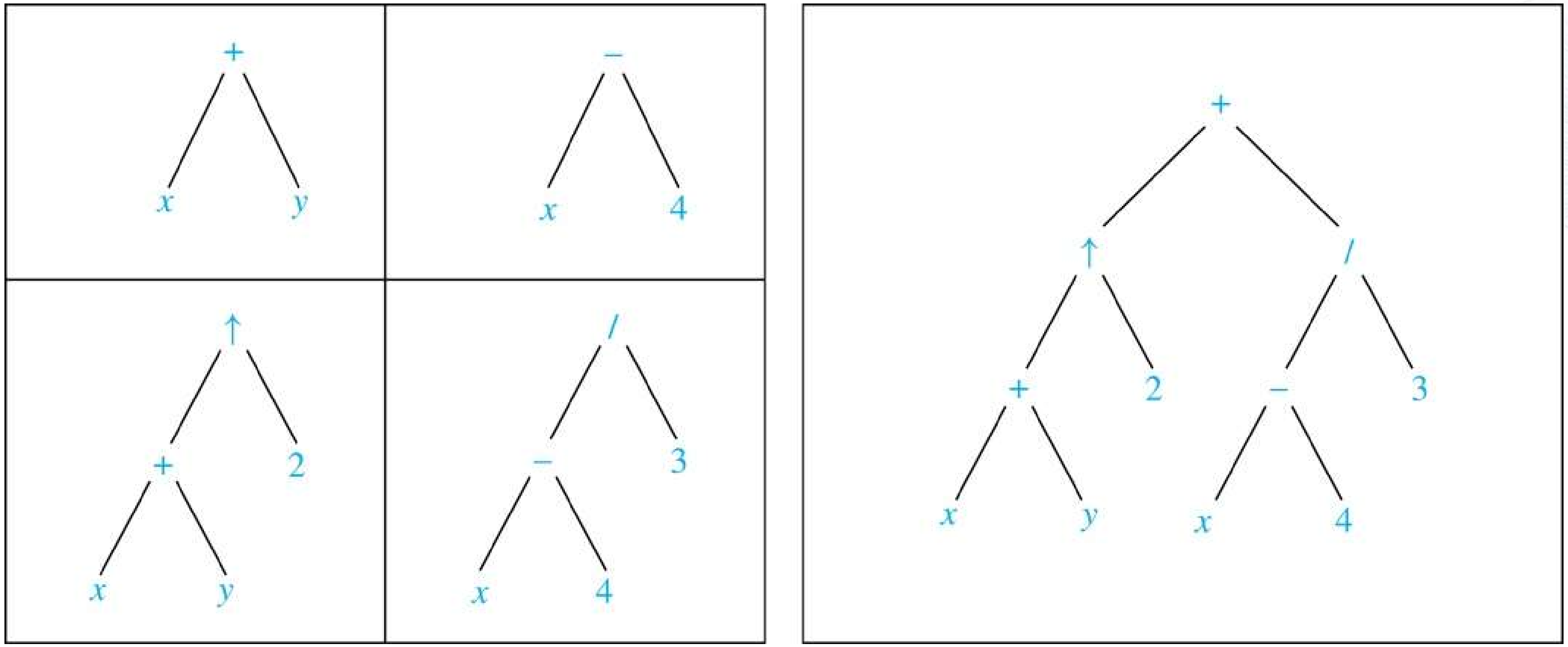
Postorder Traversal (continued)

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



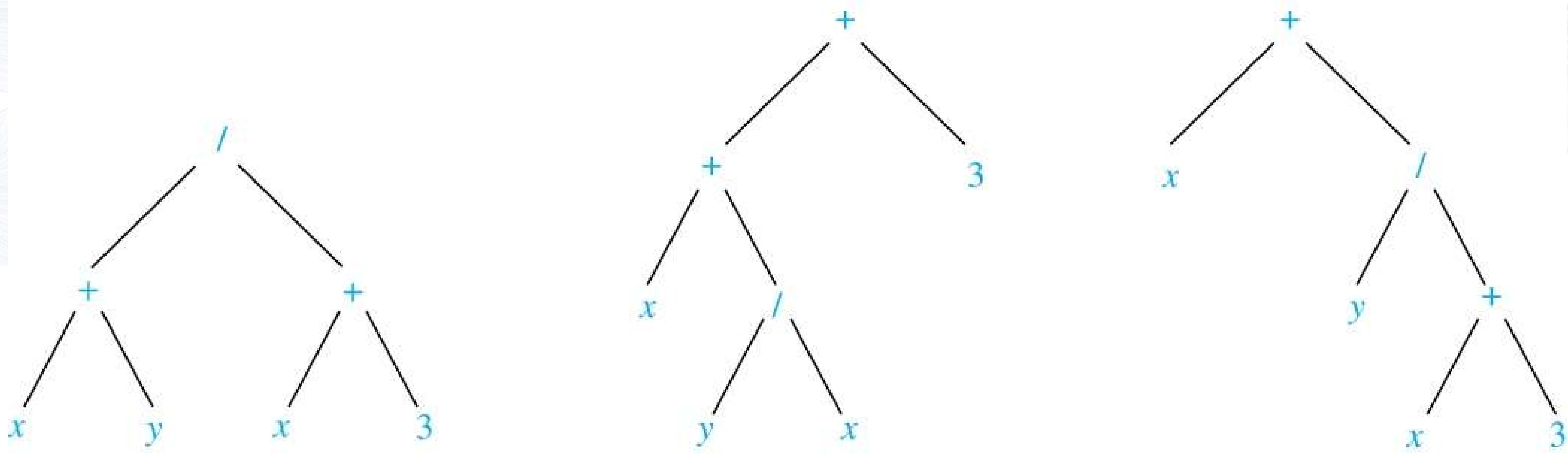
Expression Trees

- Complex expressions can be represented using ordered rooted trees
- Consider the expression $((x + y) \uparrow 2) + ((x - 4)/3)$
- A binary tree for the expression can be built from the bottom up, as is illustrated here



Infix Notation

- An inorder traversal of the tree representing an expression produces the original expression when parentheses are included except for unary operations, which now immediately follow their operands
- We illustrate why parentheses are needed with an example that displays three trees all yield the same *infix representation* (**中缀表示**)



Prefix Notation

- When we traverse the rooted tree representation of an expression in preorder, we obtain the *prefix form* (**前缀形式**) of the expression. Expressions in prefix form are said to be in *Polish notation* (**波兰记法**), named after the Polish logician Jan Łukasiewicz
- Operators precede their operands in the prefix form of an expression. Parentheses are not needed as the representation is unambiguous
- The prefix form of $((x + y) \uparrow 2\ ) + ((x - 4)/3)$ is $+ \uparrow + x\ y\ 2\ /\ - x\ 4\ 3$
- Prefix expressions are evaluated by working from right to left. When we encounter an operator, we perform the corresponding operation with the two operations to the right

Jan Łukasiewicz (1878-1956)

Example: We show the steps used to evaluate a particular prefix expression:

$+ \ - \ * \ 2 \ 3 \ 5 \ / \ \uparrow \ 2 \ 3 \ 4$

$2 \uparrow 3 = 8$

$+ \ - \ * \ 2 \ 3 \ 5 \ / \ 8 \ 4$

$8/4 = 2$

$+ \ - \ * \ 2 \ 3 \ 5 \ 2$

$2 * 3 = 6$

$+ \ - \ 6 \ 5 \ 2$

$6 - 5 = 1$

$+ \ 1 \ 2$

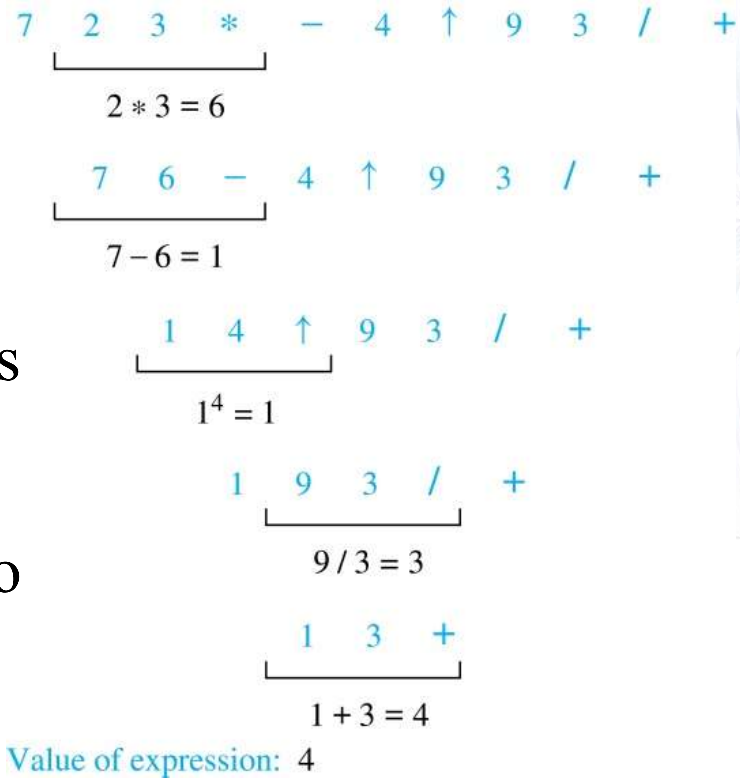
$1 + 2 = 3$

Value of expression: 3

Postfix Notation

- We obtain the *postfix form* (后缀形式) of an expression by traversing its binary trees in postorder. Expressions written in postfix form are said to be in *reverse Polish notation* (逆波兰记法)
- Parentheses are not needed as the postfix form is unambiguous
- The postfix form of $((x + y) \uparrow 2\ ) + ((x - 4)/3)$ is $x\ y + 2 \uparrow x\ 4 - 3\ / +$
- A binary operator follows its two operands. So, to evaluate an expression one works from left to right, carrying out an operation represented by an operator on its preceding operands

Example: We show the steps used to evaluate a particular postfix expression.



厦門大學信息学院
School of Informatics Xiamen University
(特色化示范性软件学院)
(National Characteristic Demonstration Software School)

Q&A