Department Of Computer Science, CUI Lahore Campus

CSC102 - Discrete Structures

By

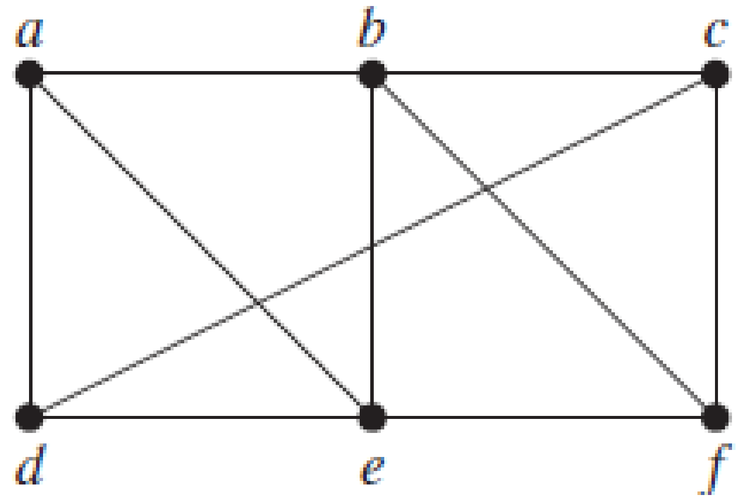
Mahwish Waqas

Lecture Outline

- Trees
  - Rooted Tree
  - Tree Terminologies
  - Ordered Rooted Tree
  - Tree Traversal
  - Infix, Prefix and Postfix Notation

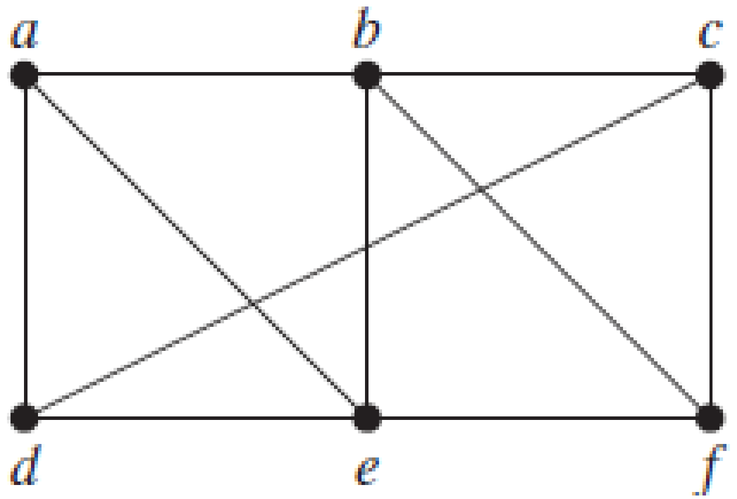
Path

- A **path** is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph.



Circuit

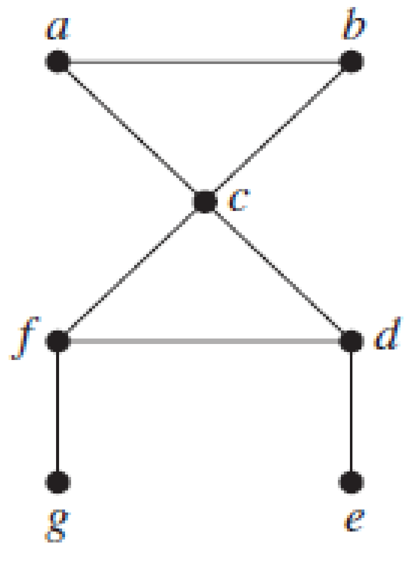
- The path is a ***circuit*** if it begins and ends at the same vertex.



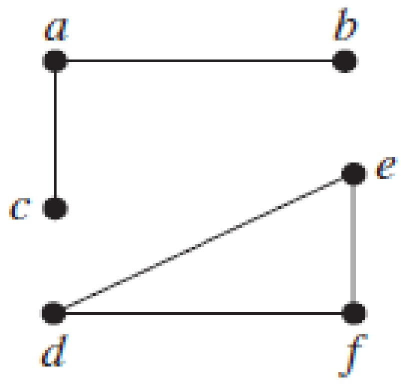
- Path ***a, b, c, d, a*** is a circuit.

Connected Graph

- An undirected graph is called ***connected*** if there is a path between every pair of distinct vertices of the graph.



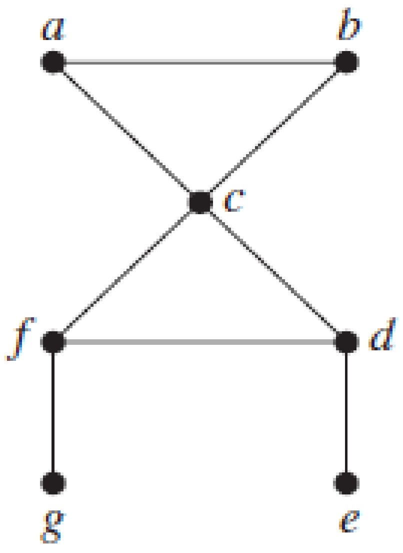
G_1



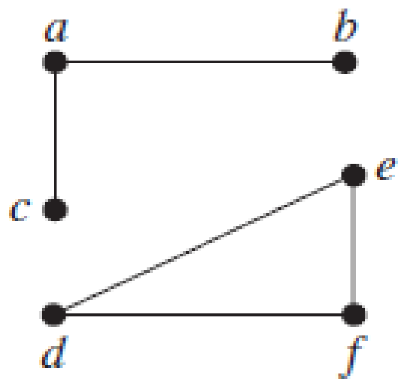
G_2

Connected Graph

- The graph ***G1*** is connected.
- The graph ***G2*** is not connected. (For instance, there is no path in *G*2 between vertices *a* and *d*.)



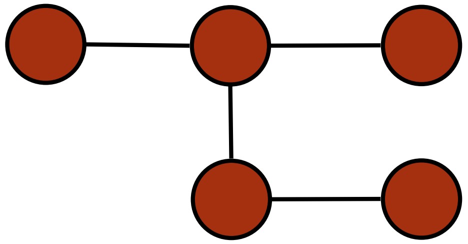
G_1



G_2

Tree

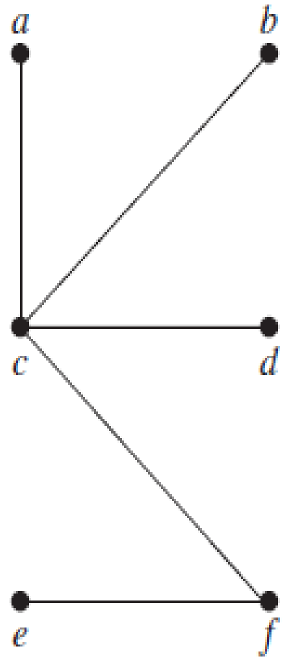
- Trees are particularly useful in computer science, where they are employed in a wide range of algorithms.
- A tree is an undirected graph T such that
  - T is connected
  - T has no cycles (circuits)


Tree

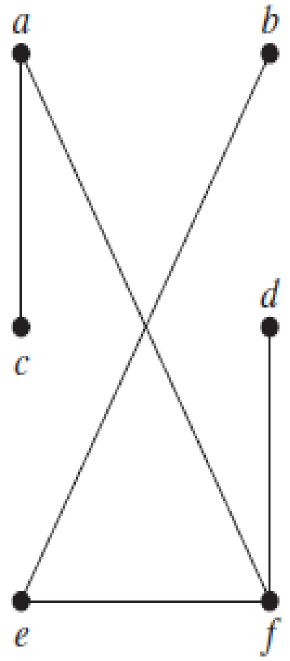
- A tree cannot contain multiple edges or loops. Therefore any tree must be a simple graph.

Example

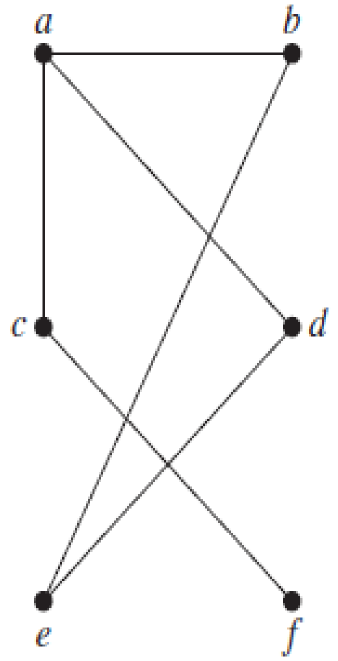
Which of the graphs are trees?



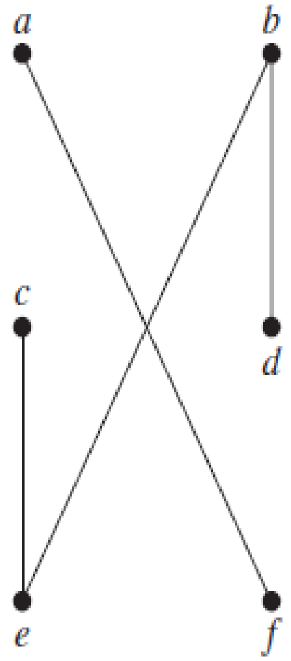
G_1



G_2



G_3



G_4

Example

- *G*1 and *G*2 are trees, because both are connected graphs with no simple circuits.
- *G*3 is not a tree because *e*, *b*, *a*, *d*, *e* is a simple circuit in this graph.
- *G*4 is not a tree because it is not connected.

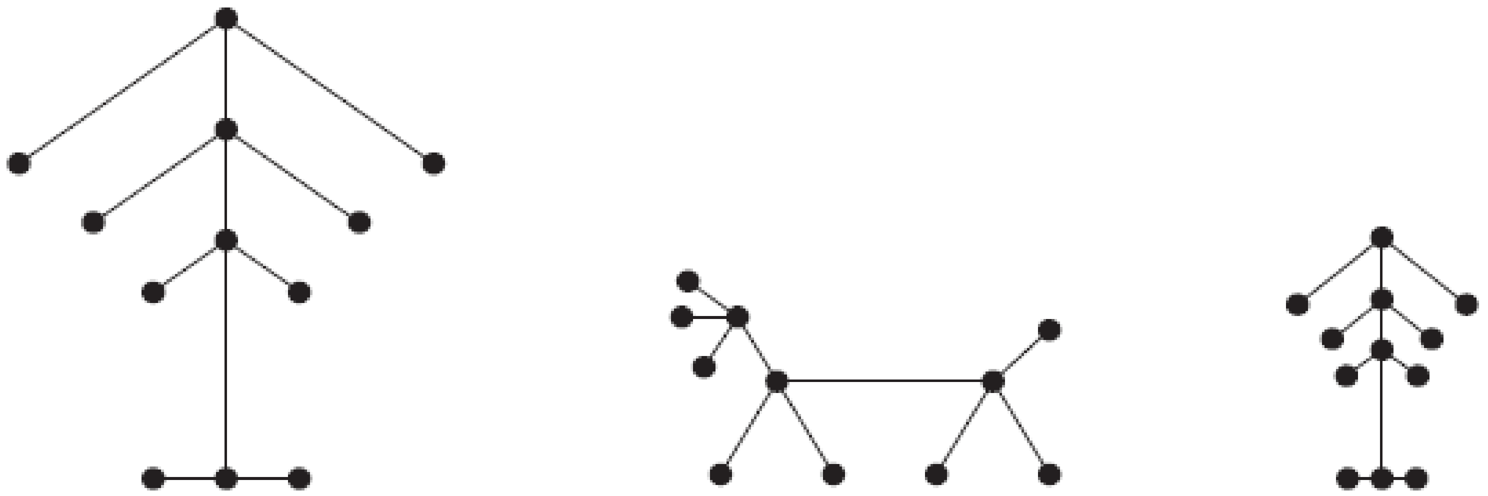
Forest

A **forest** is an undirected graph such that

- It has no simple circuits
- **It is not necessarily connected**

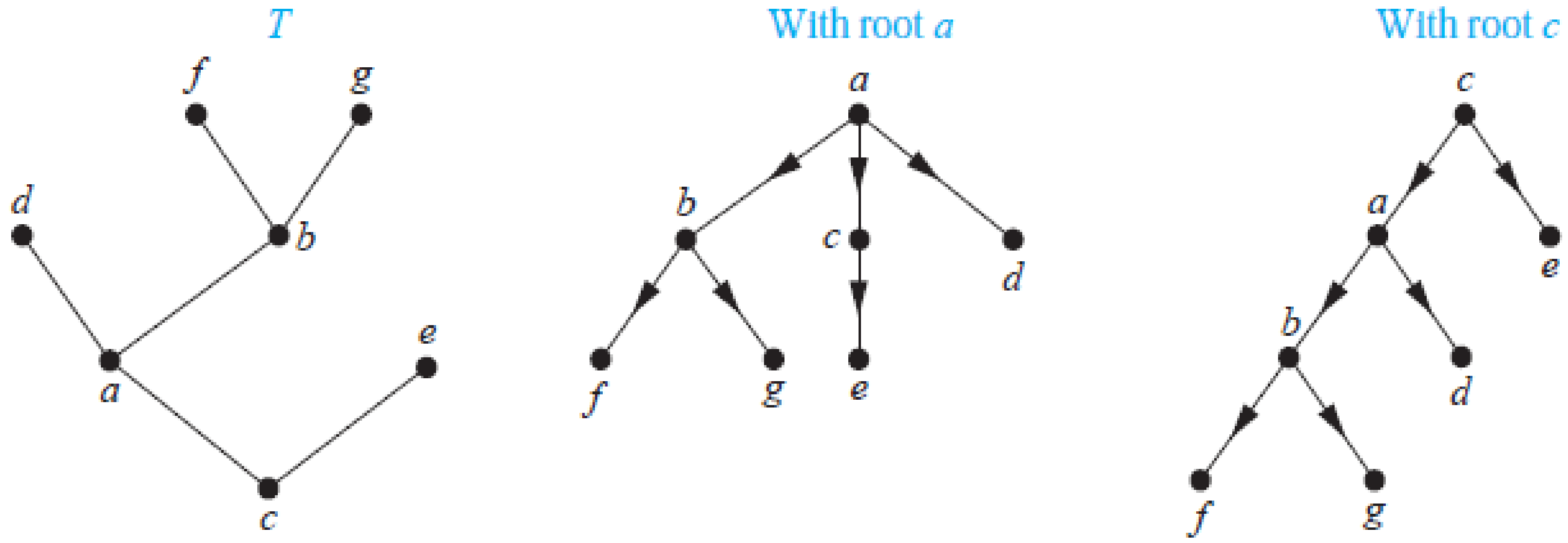
The connected components of a forest are trees

This is one graph with three connected components.



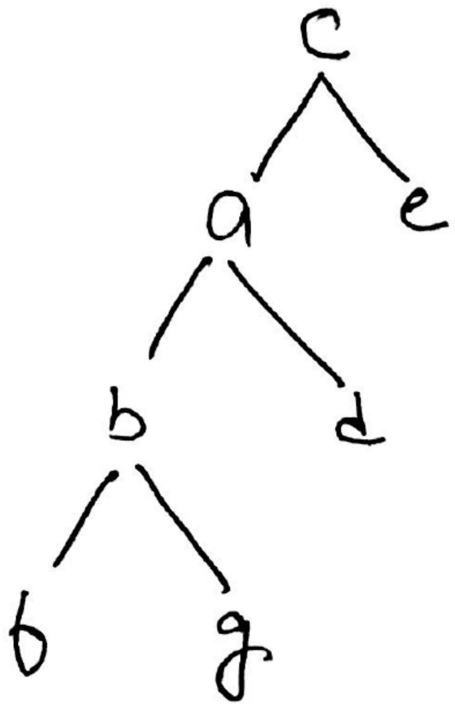
Rooted Tree

- 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.
- Root, parent, child, siblings relations between vertices



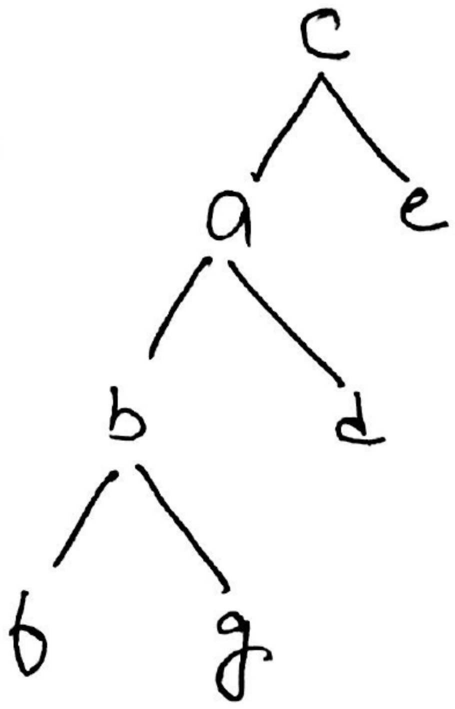
Terminologies

- Suppose that *T* is a rooted tree.
- If ***v*** is a vertex in *T* 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 the parent of ***v***, ***v*** is called a **child** of ***u***.
- Vertices with the same parent are called **siblings**.



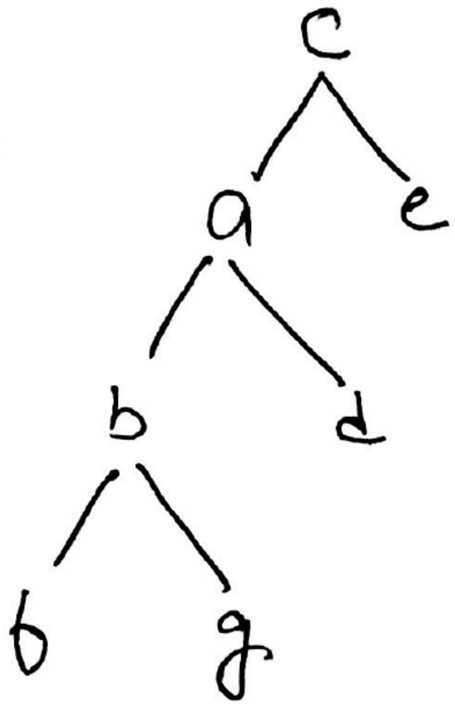
Terminologies

- The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root (that is, its parent, its parent's parent, and so on, until the root is reached).
- The **descendants** of a vertex ***v*** are those vertices that have ***v*** as an ancestor.



Terminologies

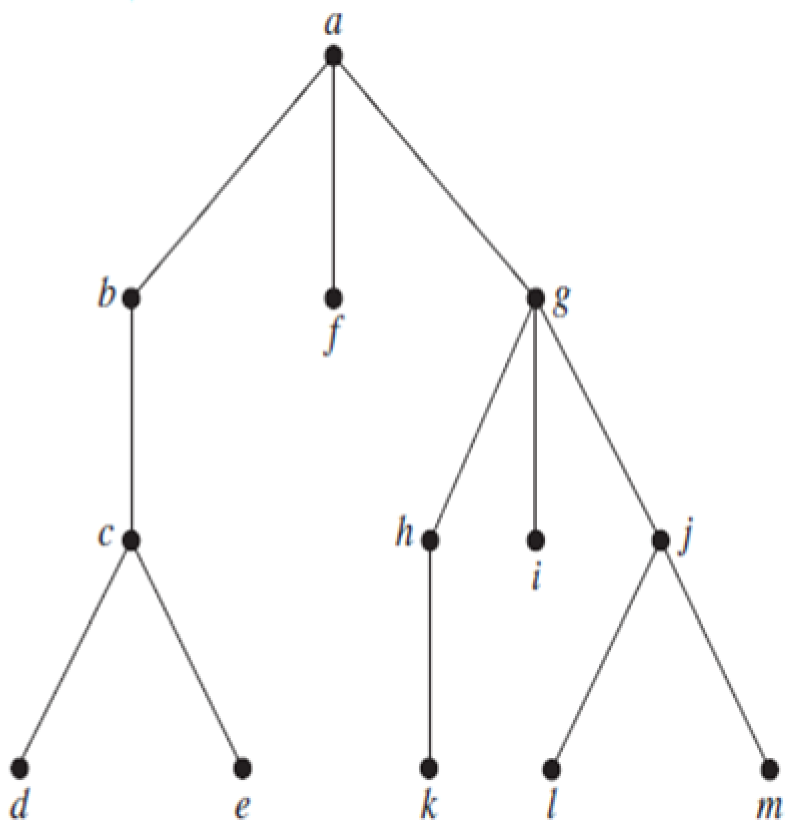
- A vertex of a rooted tree is called a **leaf** if it has no children.
- Vertices that have children are called **internal vertices**. The root is an internal vertex unless it is the only vertex in the graph, in which case it is a leaf.
- If ***v*** is a vertex in a tree, the **subtree** with ***v*** as its root is the subgraph of the tree consisting of ***v*** and its descendants and all edges incident to these descendants.



Example

- In the rooted tree ***T*** (with root ***a***), 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.

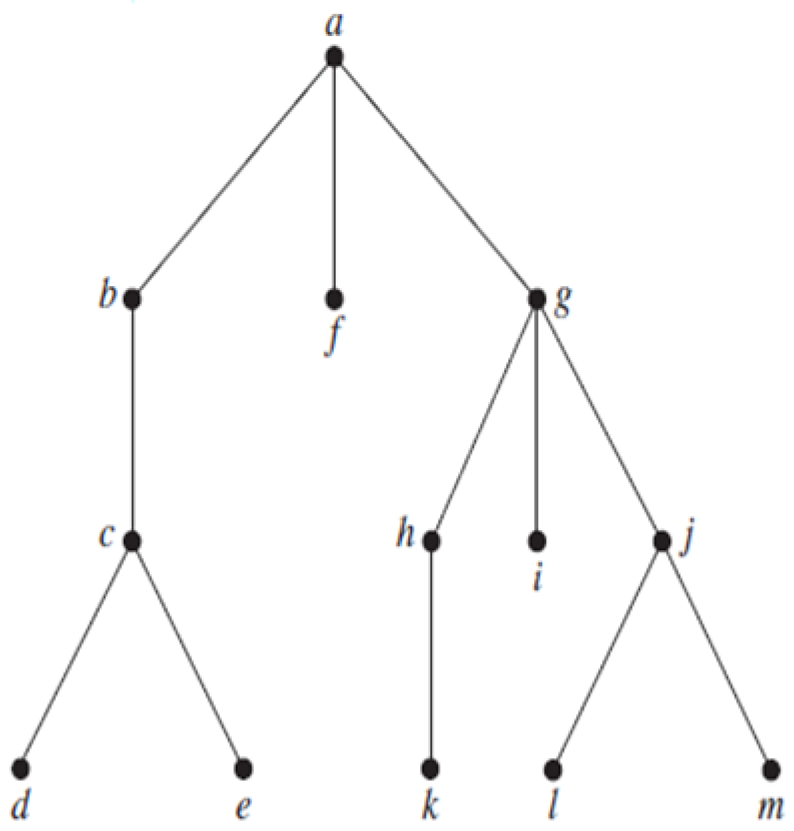
- Parent of **c**: b
- Children of **g**: h, i, j
- Siblings of **h**: i, j
- All ancestors of **e**: c, b, a
- All descendants of **b**: c, d, e
- All internal vertices: a, b, g, c, h, j
- All leaves: f, d, e, i, k, l, m



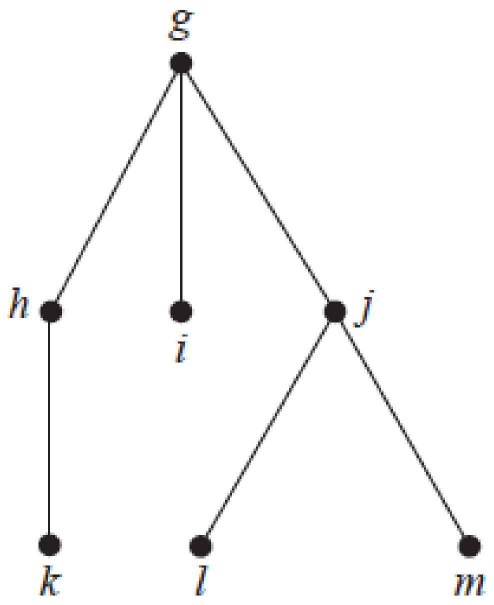
A Rooted Tree *T*

Example

- In the rooted tree ***T*** (with root ***a***), What is the subtree rooted at ***g***?



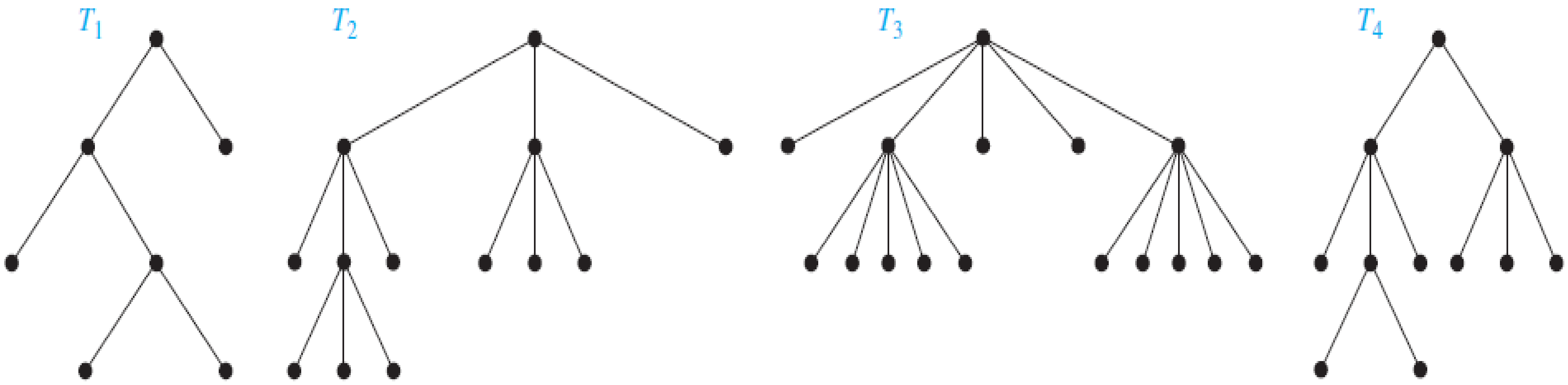
A Rooted Tree *T*



Subtree Rooted at *g*

Rooted Tree

- 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***.



Definitions

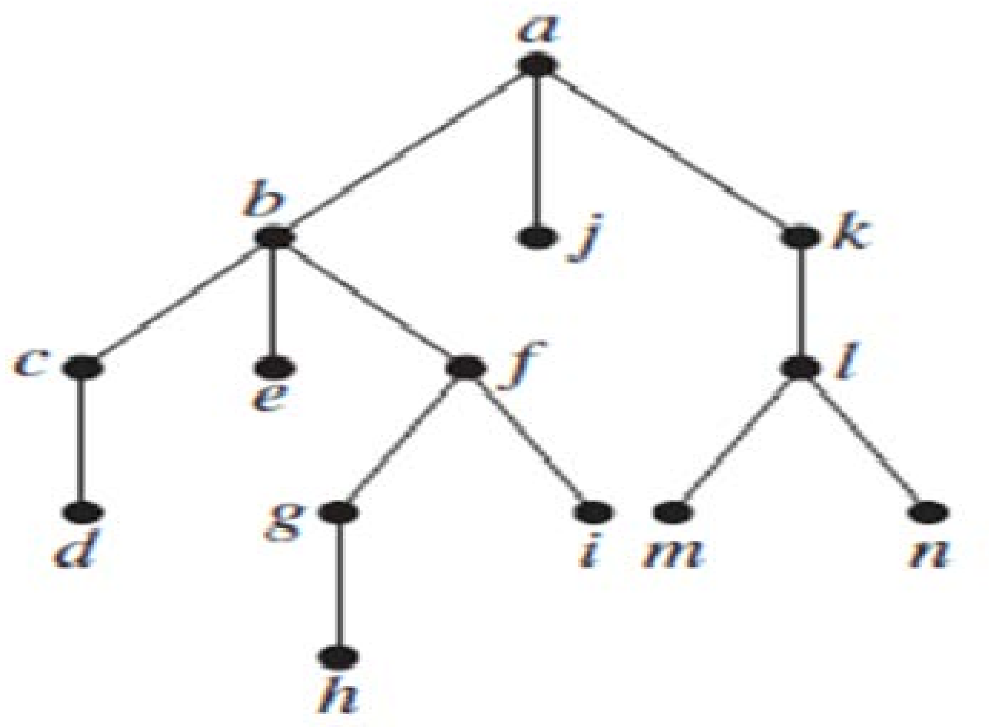
- The **level** of a vertex ***v*** in a rooted tree is the length of the unique path from the root to this vertex.

- The level of the root is defined to be zero.

- The **height** of a rooted tree is the maximum of the levels of vertices. In other words, the height of a rooted tree is the length of the longest path from the root to any vertex.

Example

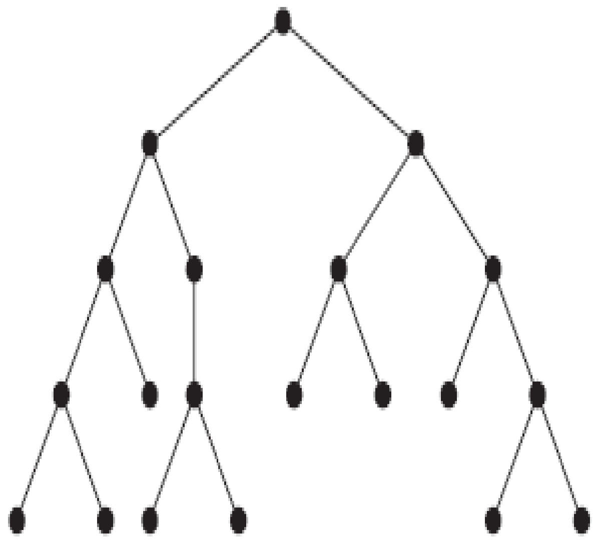
- Find the level of each vertex in the rooted tree **T**. What is the height of this tree?

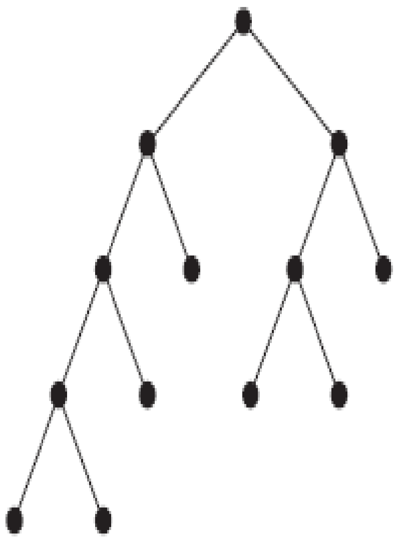


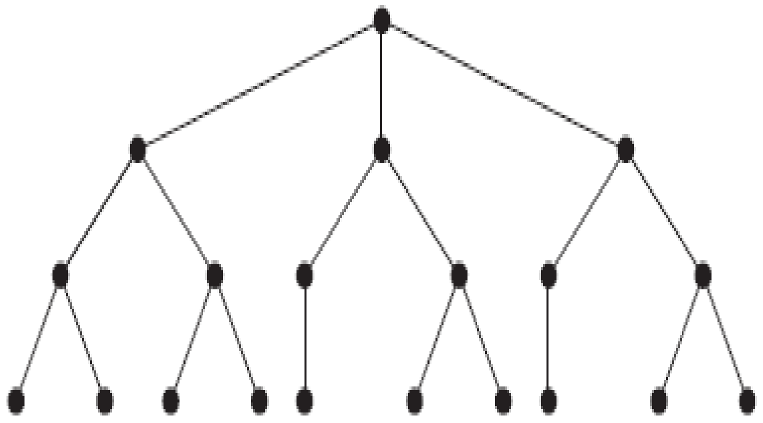
T

Balanced Trees

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


T_1


T_2


T_3

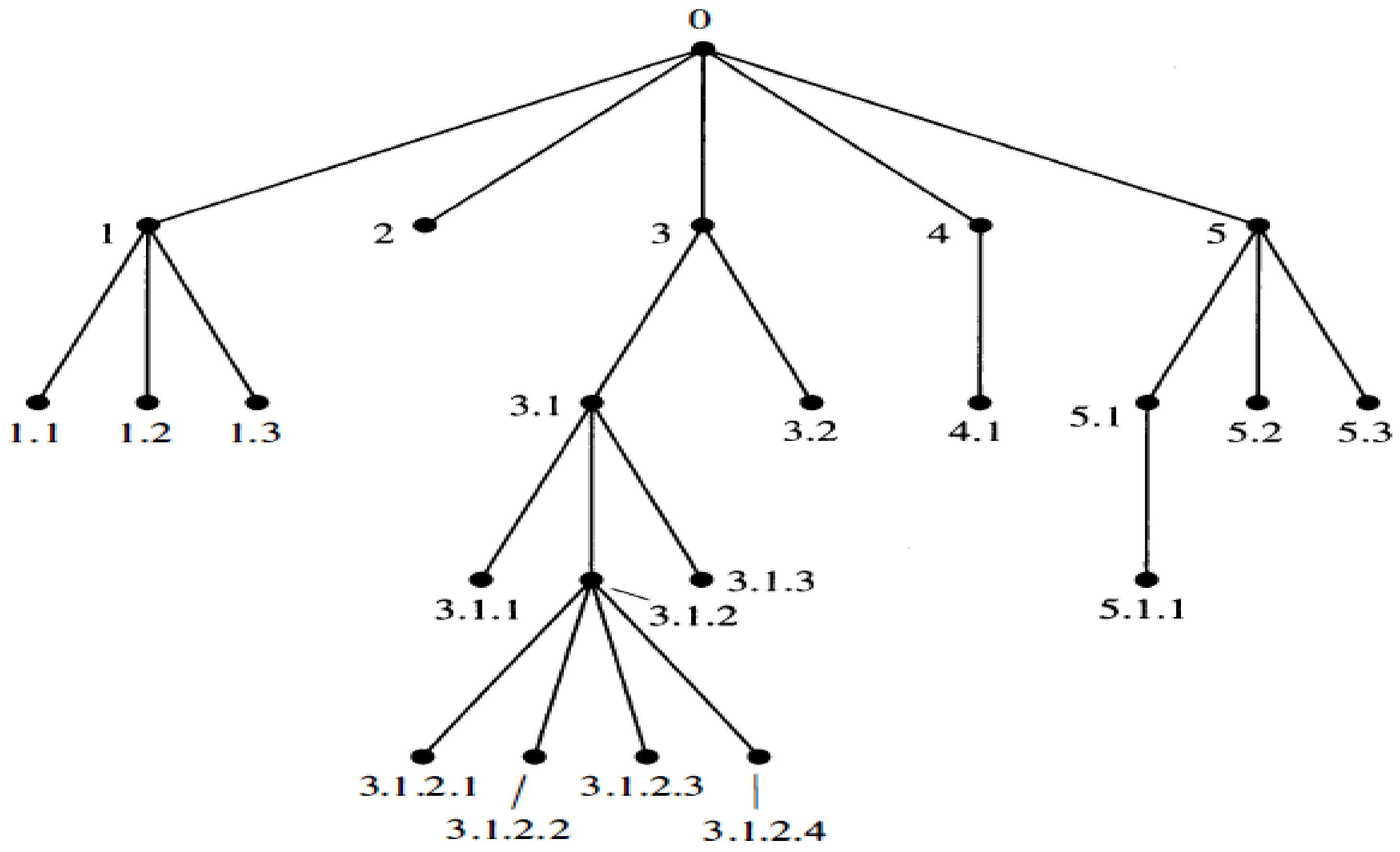
Ordered rooted Tree

- An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.

- In a drawing of an ordered rooted tree, with the root shown at the top, the children of a vertex are shown from left to right.

Ordered rooted Tree

- An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.



Tree Traversal and Traversal Algorithms

- Ordered rooted trees are often used to store information. Tree traversal is the procedure of visiting different vertices of the tree to read information stored in that vertex. There are three different orders of the tree traversal.
- Most commonly used algorithms:
  - Preorder traversal
  - Inorder traversal
  - Postorder traversal
- All recursively defined.

Preorder Traversal

- Let T be an ordered rooted tree with root ***r*** and $T_1, T_2, \ldots T_n$ are the subtrees at ***r*** from left to right. The preorder traversal begins by visiting ***r* .** It continues by traversing T_1 in preorder, then T_2 in preorder, then T_3 in preorder, . . . , and finally T_n in preorder.

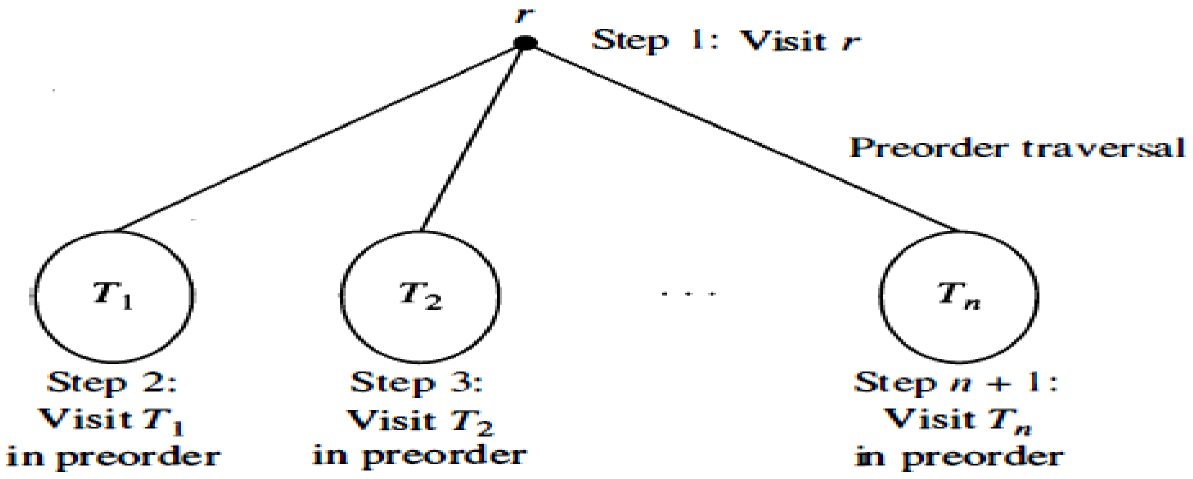
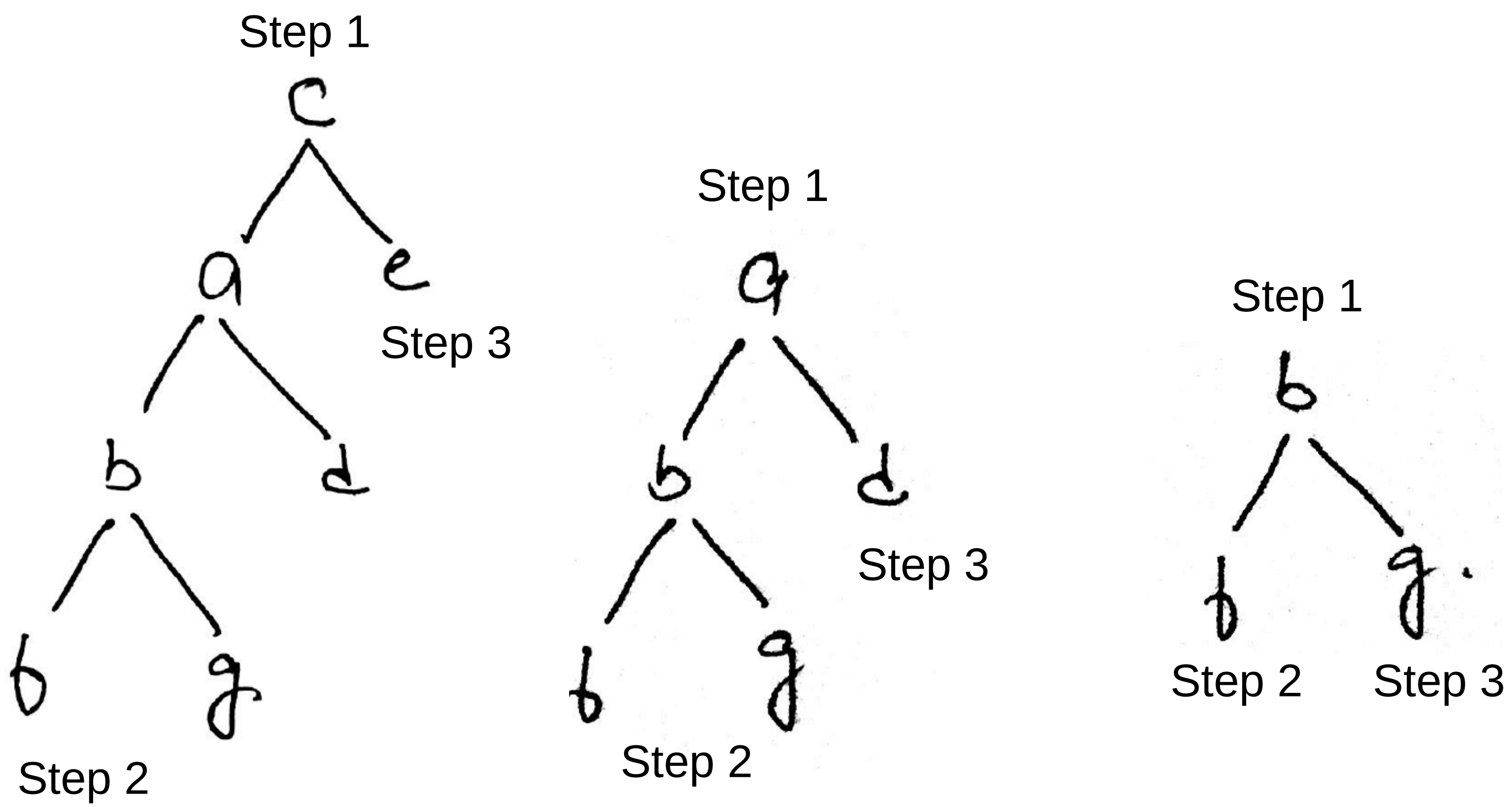

r
Step 1: Visit r
Preorder traversal
T_1
T_2
. . .
T_n
Step 2: Visit T_1 in preorder
Step 3: Visit T_2 in preorder
Step $n+1$: Visit T_n in preorder

FIGURE 2 Preorder Traversal.

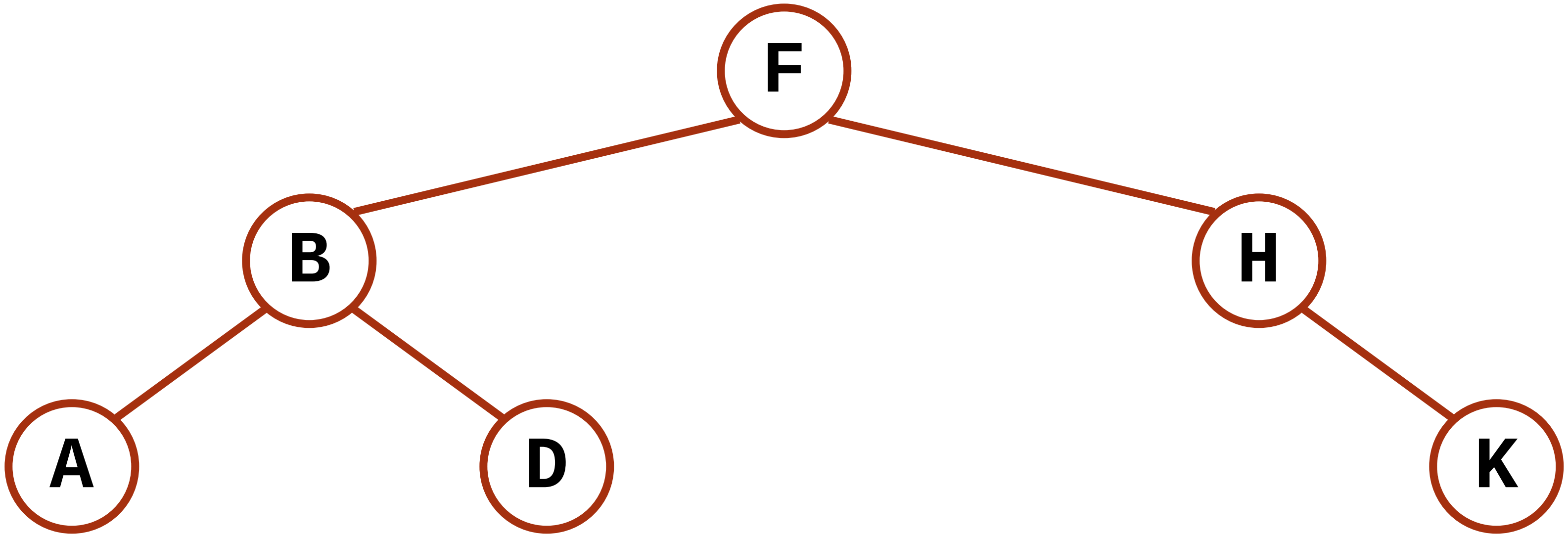
Preorder Traversal



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

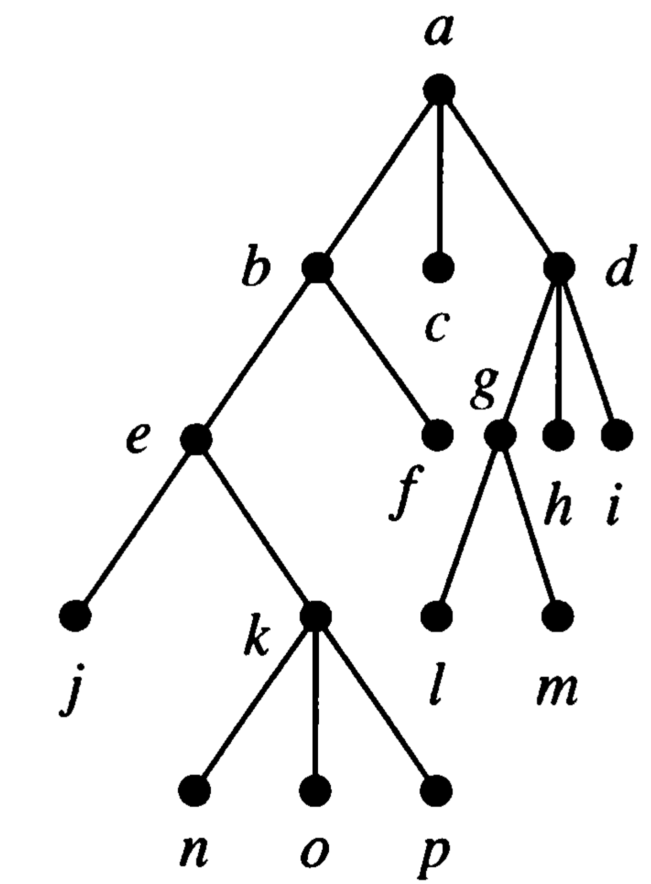
Preorder Traversal

- Root, left, right
- Example:

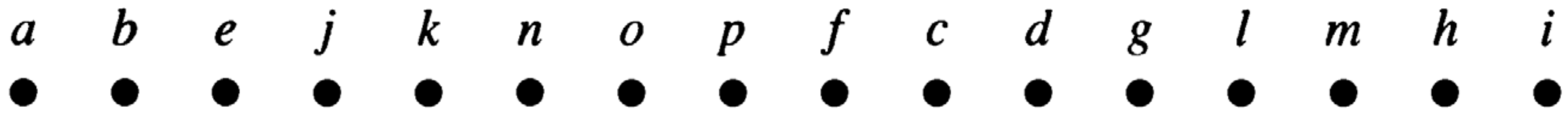


- F, B, A, D, H, K

Preorder Traversal



Preorder traversal: **Visit root, visit subtrees left to right**



Postorder Traversal

- Let T be an ordered rooted tree with root ***r*** and $T_1, T_2, \ldots T_n$ 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_3 in postorder, . . . , and finally T_n in postorder and ends by visiting ***r***.

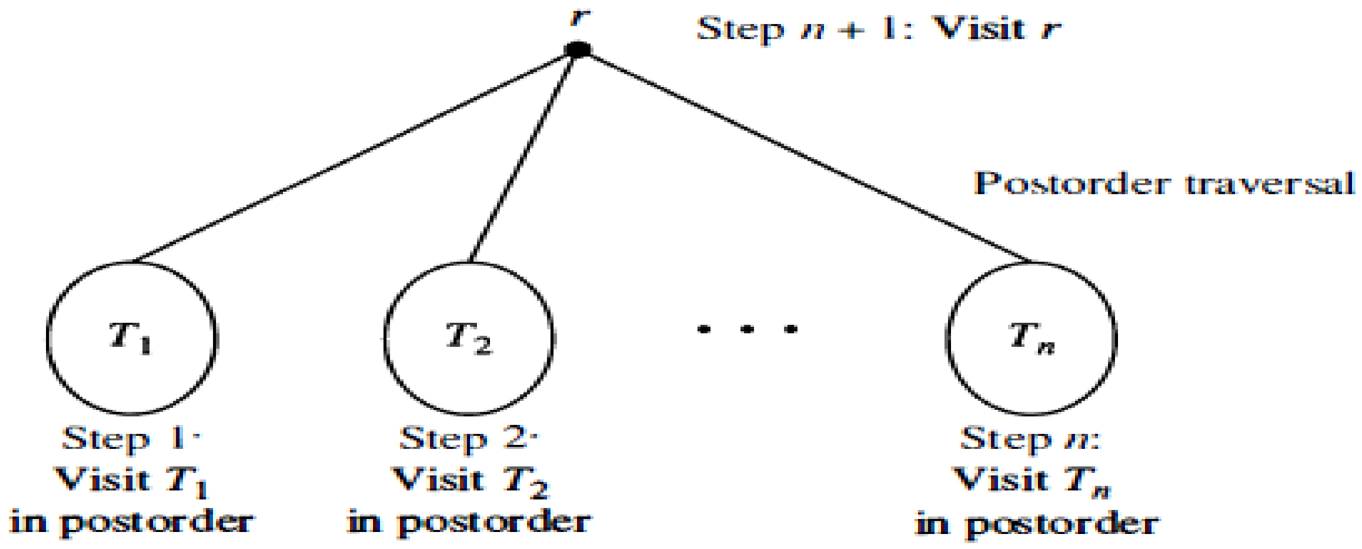
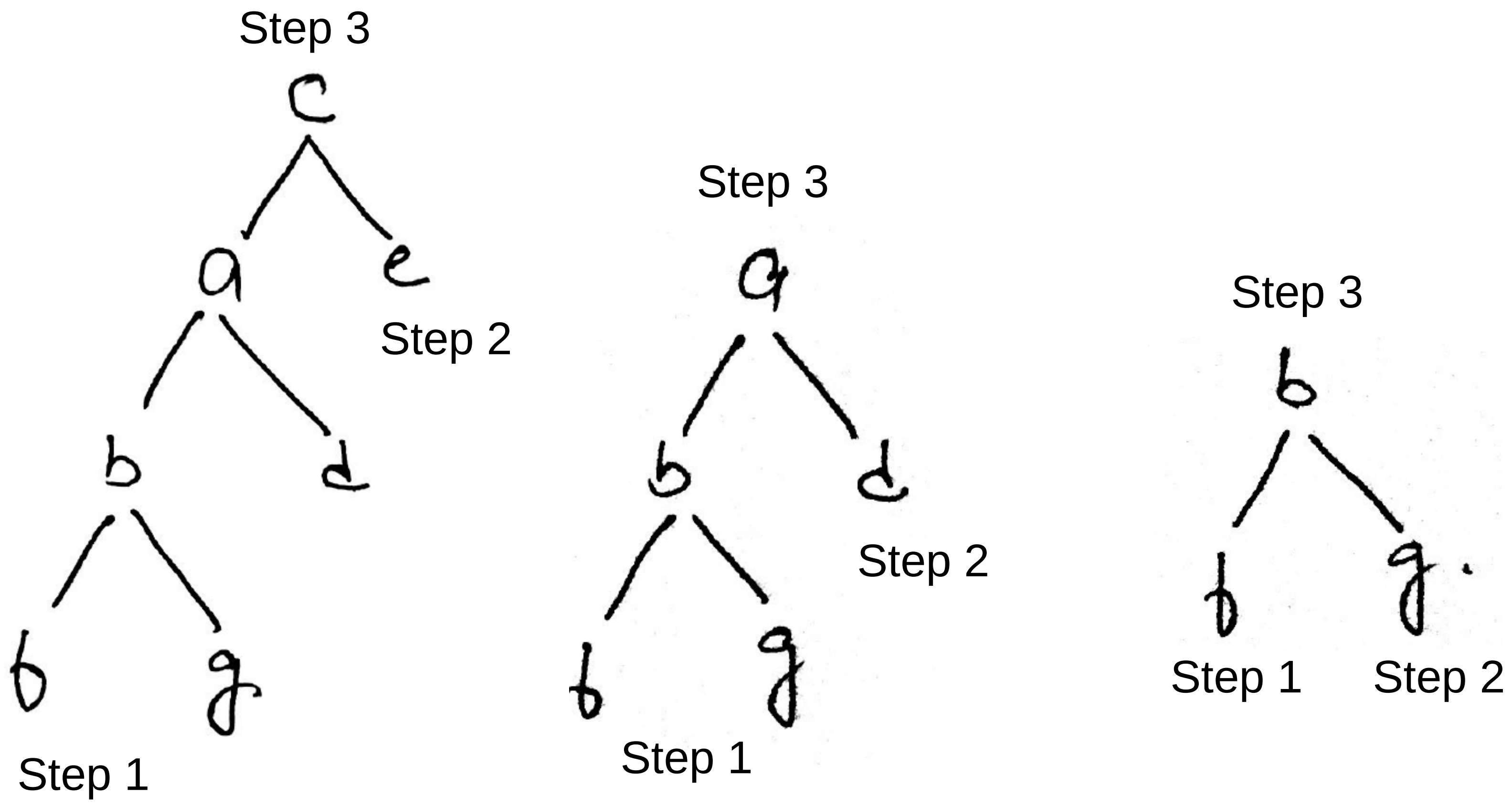

r
Step $n + 1$: Visit r
Postorder traversal
T_1
T_2
T_n
Step 1: Visit T_1 in postorder
Step 2: Visit T_2 in postorder
Step n: Visit T_n in postorder

FIGURE 7 Postorder Traversal.

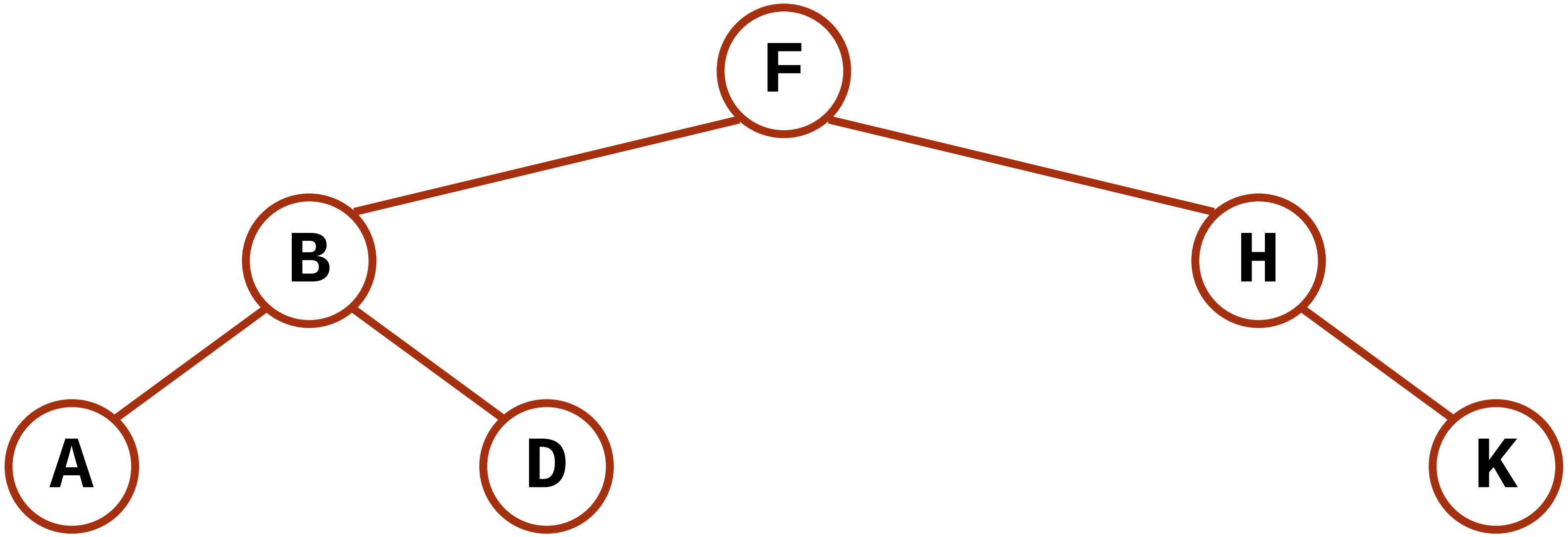
Postorder Traversal



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

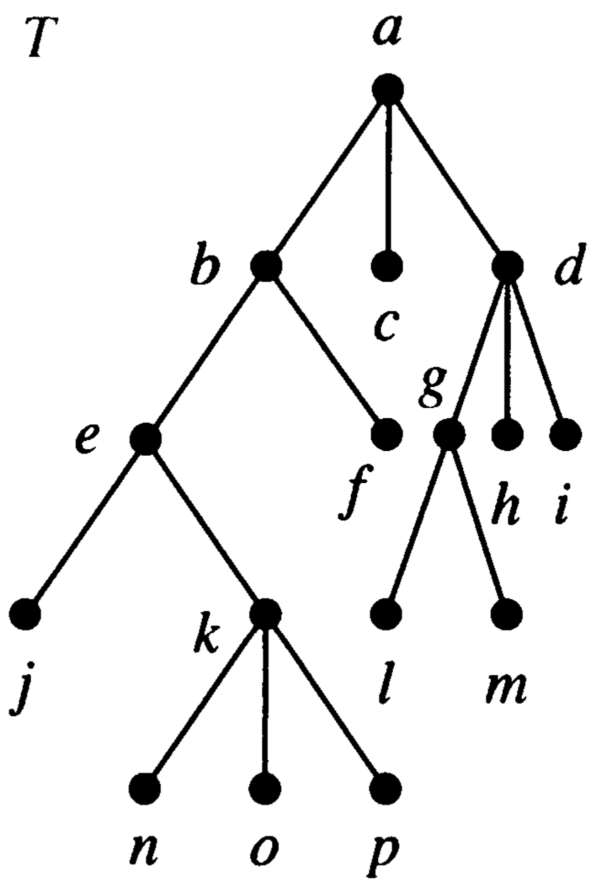
Postorder Traversal

- Left, right, root
- Example:

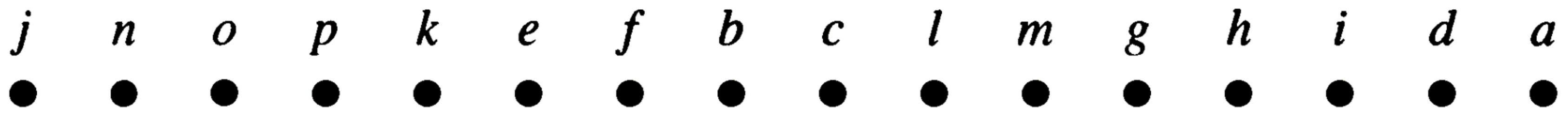


- A, D, B, K, H, F

Postorder Traversal



Postorder traversal: Visit subtrees left to right; visit root



Inorder Traversal

- Let T be an ordered rooted tree with root ***r*** and $T_1, T_2, \ldots T_n$ are the subtrees at ***r*** from left to right. The in order 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.

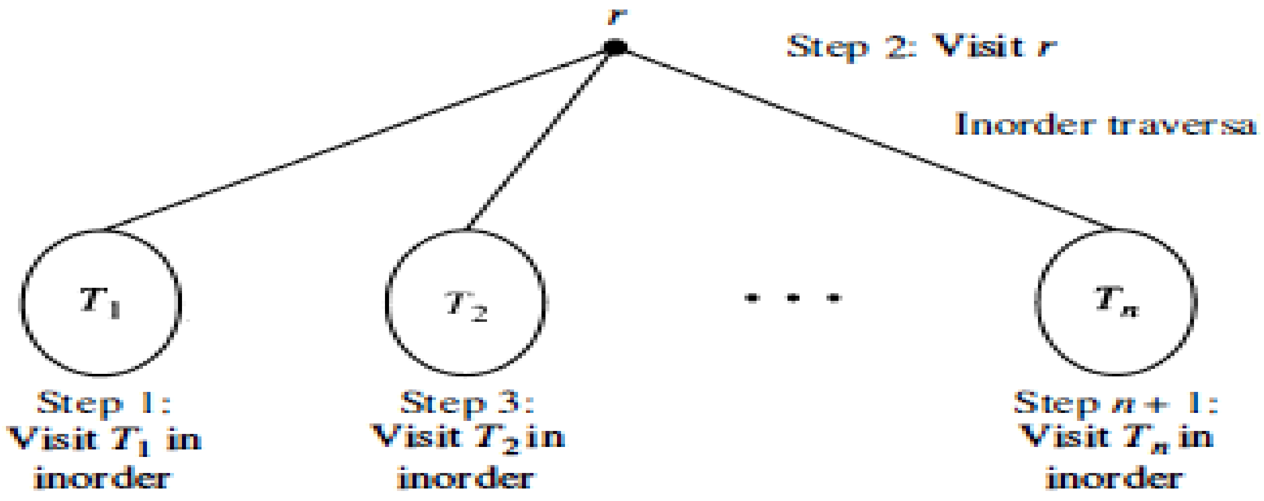
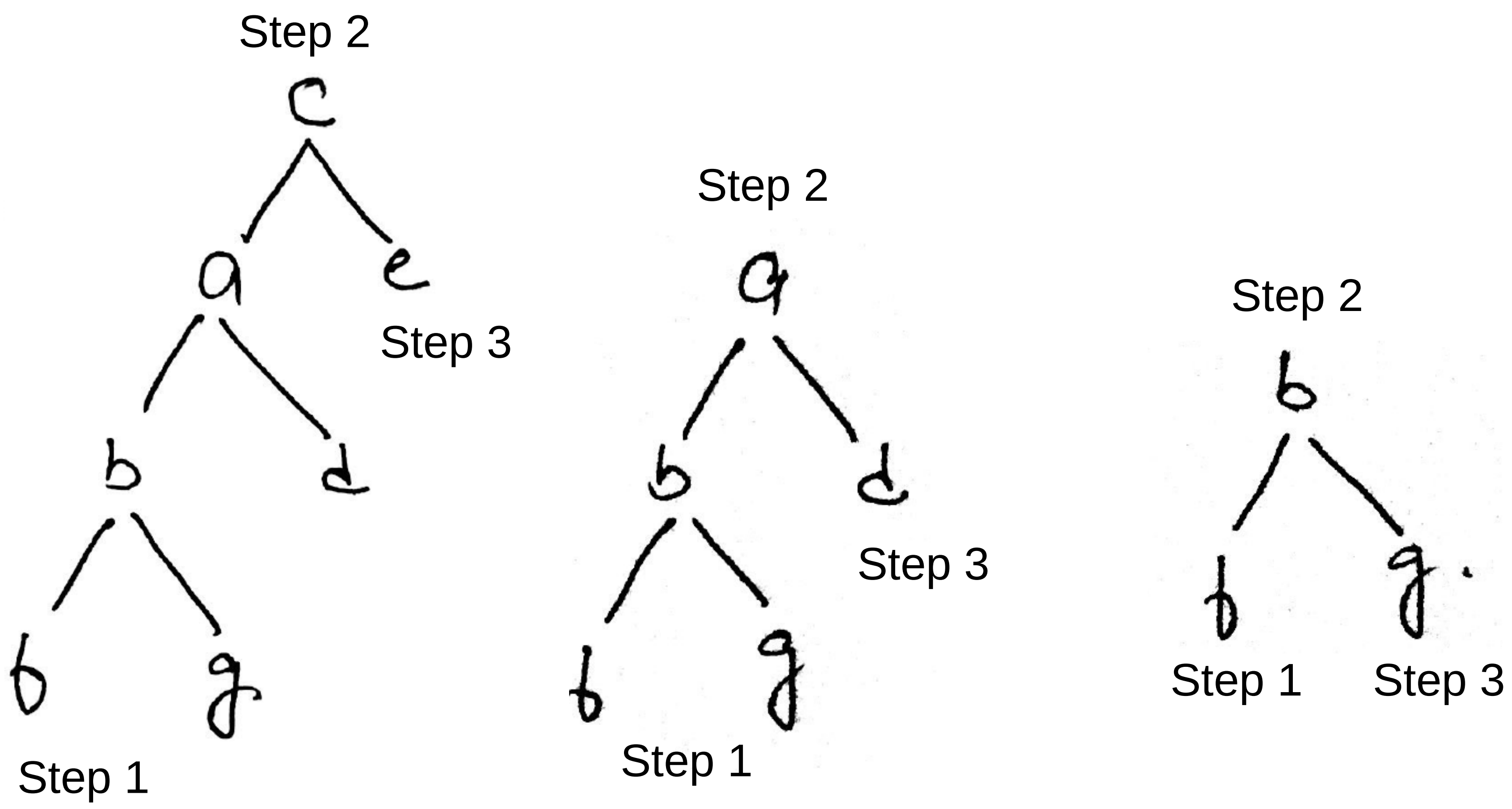

r
Step 2: Visit r
Inorder traversa
T_1
T_2
. . .
T_n
Step 1: Visit T_1 in inorder
Step 3: Visit T_2 in inorder
Step $n+1$: Visit T_n in inorder

FIGURE 5 Inorder Traversal.

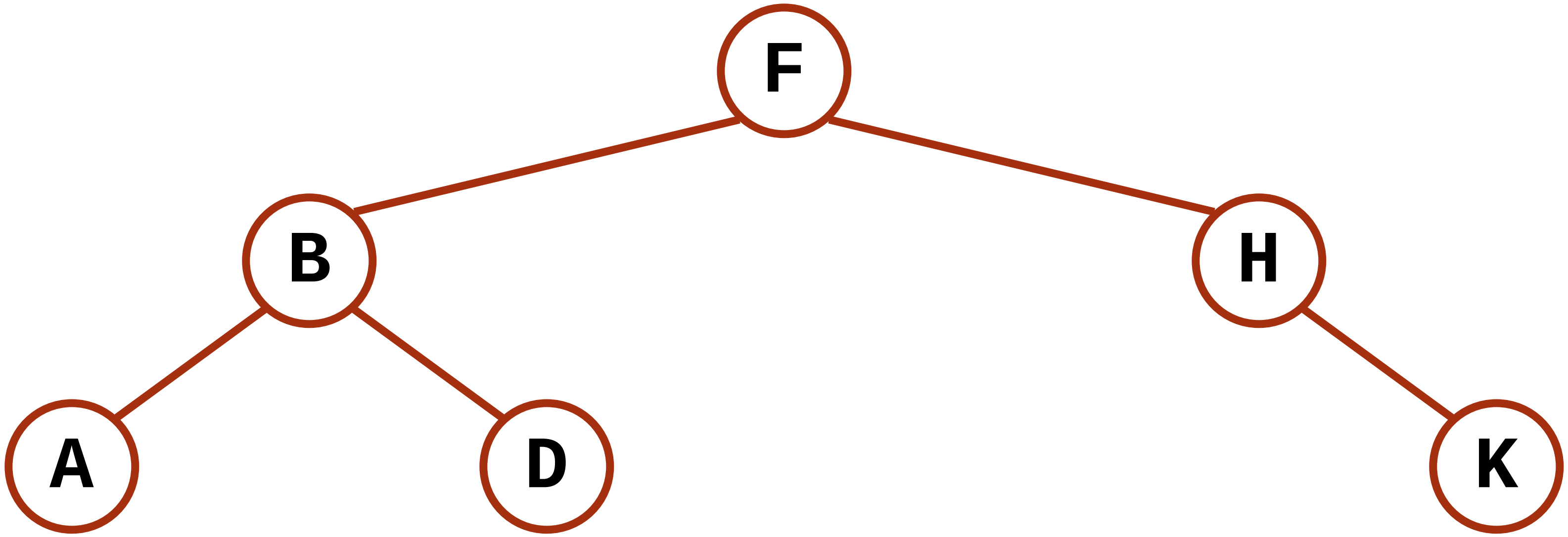
In-order Traversal



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

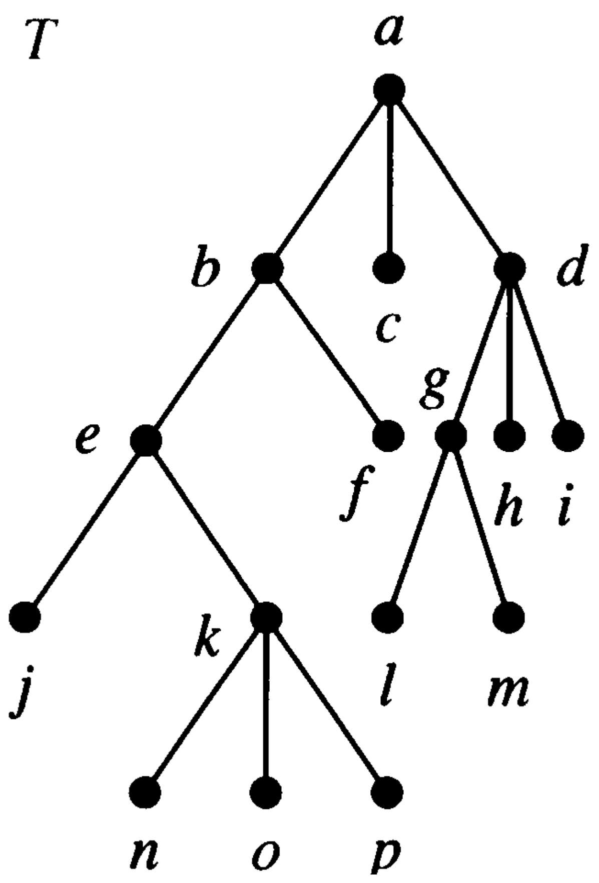
In-order Tree Walk

- Left, root, right
- Example:

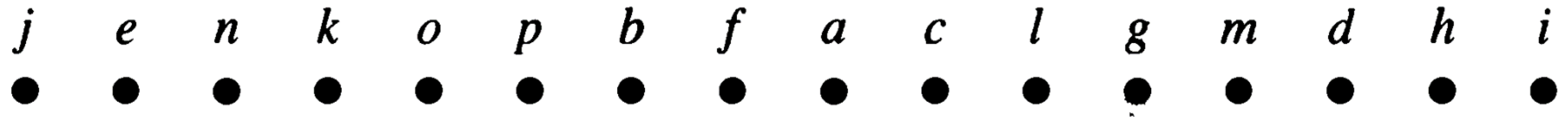


- A, B, D, F, H, K

Example

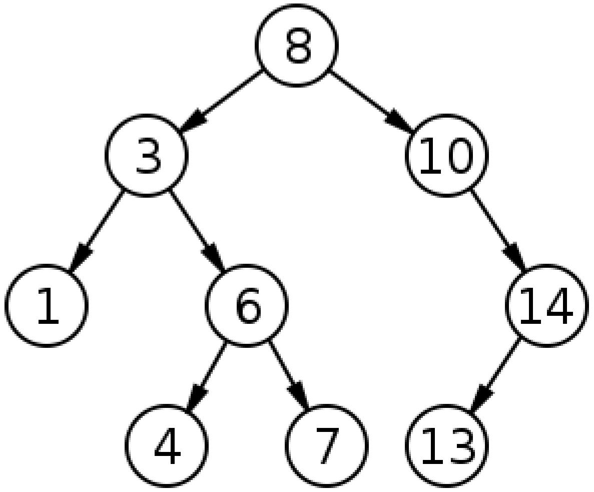


Inorder traversal: **Visit leftmost subtree, visit root, visit other subtrees left to right**



Binary Search Trees

- A Binary search tree is a binary tree in symmetric order.
- Symmetric order means that:
    - every node has a key
    - every node's key is
        - larger than all keys in its left subtree
        - smaller than all keys in its right subtree

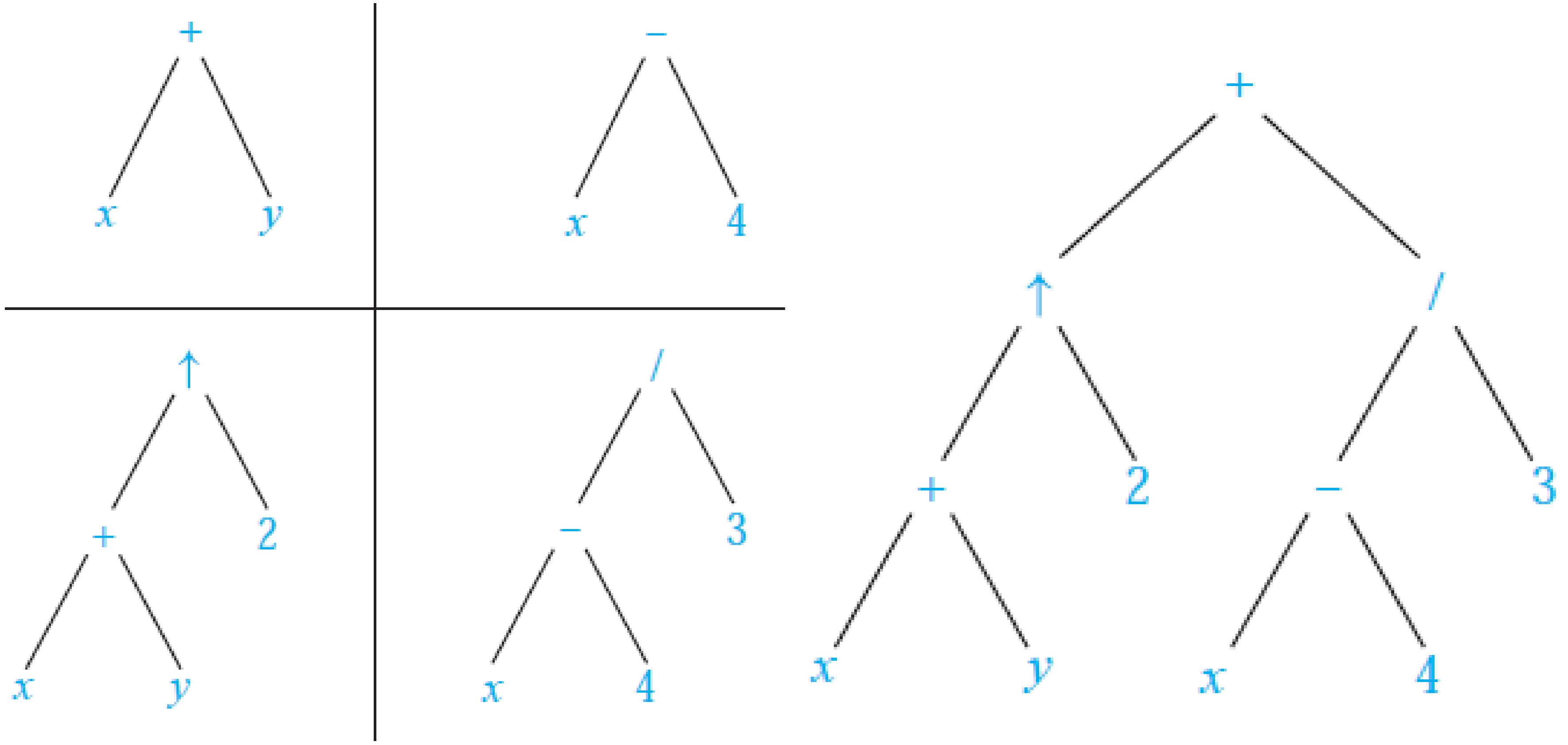


Infix, Prefix and Postfix Notation

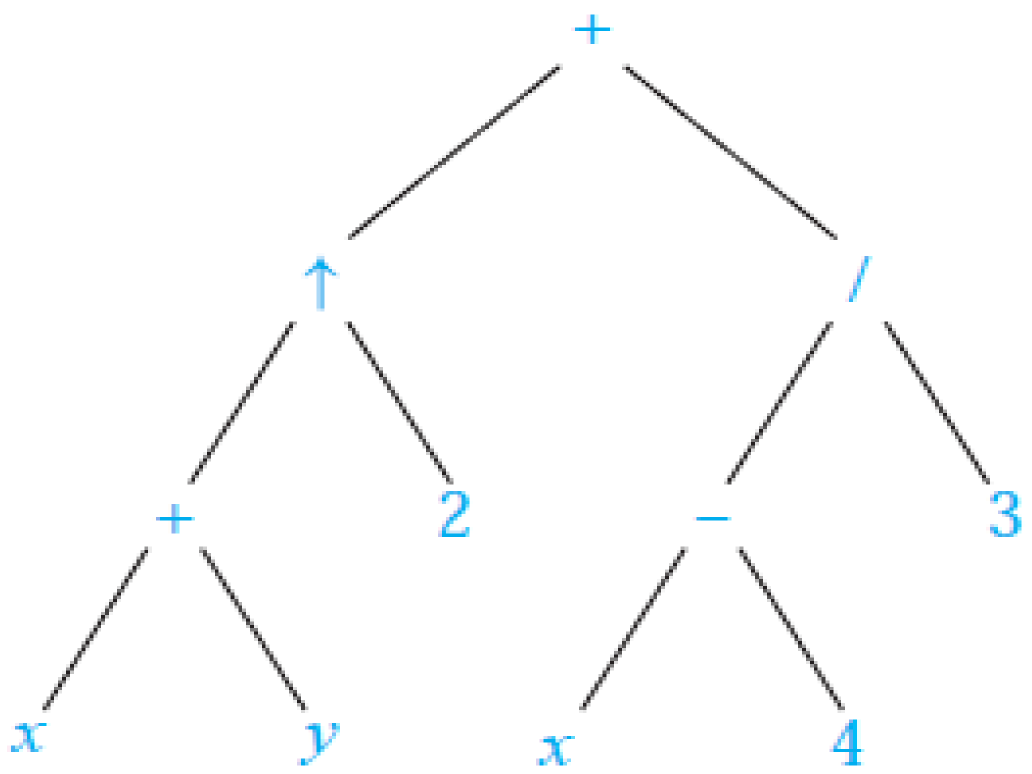
- We can represent complicated expressions using ordered rooted tree.
- The representation of an arithmetic expression involving operators + (addition), − (subtraction), $*$ (multiplication), / (division), and ↑ (exponentiation).
- An ordered rooted tree can be used to represent such expressions, where the internal vertices represent operations, and the leaves represent the variables or numbers.
- Use parentheses to indicate the order of the operations.
- Each operation operates on its left and right sub trees (in that order).

Example

- What is the ordered rooted tree that represents the expression $((x + y) \uparrow 2) + ((x - 4)/3)$.



Example



$$preorder : + \uparrow + xy2 / - x43$$

$$postorder : xy + 2 \uparrow x4 - 3 / +$$

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

Infix, Prefix and Postfix form

- Infix Form
  - It is the common arithmetic and logical formula notation, in which operators are written between the operands
- Prefix Form
  - Also known as **Polish notation.**
  - A form of notation for logic, arithmetic, and algebra.
  - It places operators to the left of their operands.
- Postfix Form
  - Also known as **Reverse Polish Notation**.
  - We obtain the postfix form of an expression by traversing its tree in **postorder**.

Example

- What is the value of prefix expression $+ - \times\ 2\ 3\ 5 / \uparrow 2\ 3\ 4$.

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

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

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

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

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

Value of expression: 3

Example

- What is the value of postfix expression $7\ 2\ 3\ \times -\ 4 \uparrow 9\ 3 / +$.

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

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

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

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

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

Value of expression: 4

Chapter Exercise

- Chapter # 11
- Topic # 11.1
- Question # 1, 2, 3, 4, 5, 6, 7, 8, 9,10
- Topic # 11.3
- Question # 7 – 18, 23 ,24