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

Because a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops. Therefore any tree must be a simple graph.

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

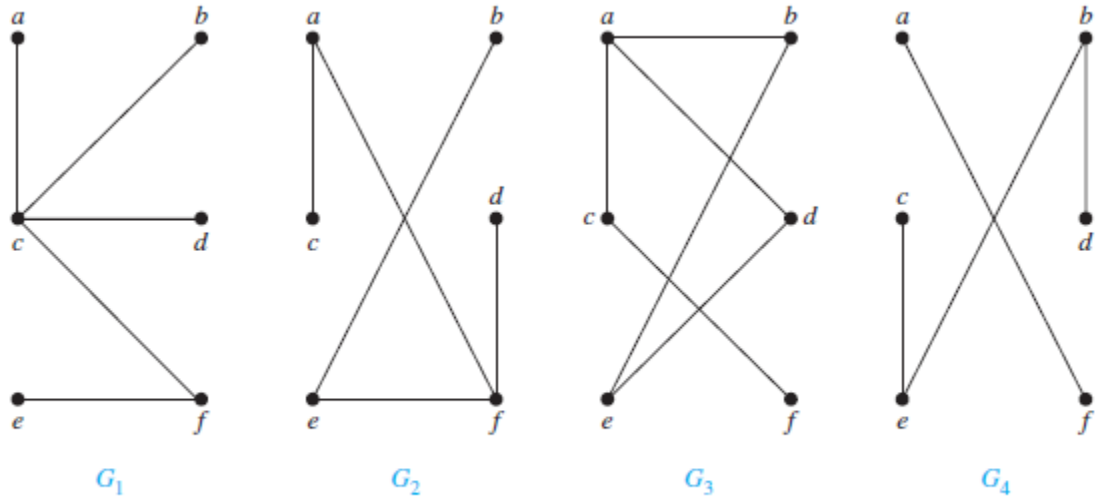


FIGURE 2 **Examples of Trees and Graphs That Are Not Trees.**

Which of the graphs shown in Figure 2 are trees?

Solution: 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. Finally, G_4 is not a tree because it is not connected. ◀

Any connected graph that contains no simple circuits is a tree. What about graphs containing no simple circuits that are not necessarily connected? These graphs are called **forests** and have the property that each of their connected components is a tree. Figure 3 displays a forest.

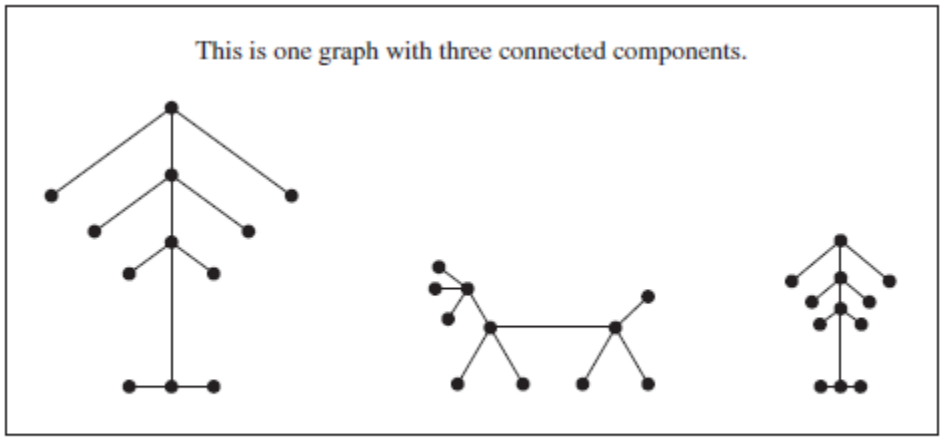


FIGURE 3 **Example of a Forest.**

A particular vertex of a tree is designated as the **root**.

There is a unique path from the root to each vertex of the graph, we direct each edge away from the root. Thus, a tree together with its root produces a directed graph called a **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.

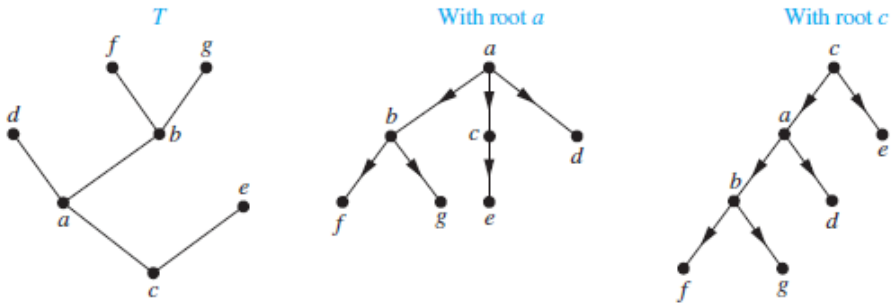


FIGURE 4 A Tree and Rooted Trees Formed by Designating Two Different Roots.

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 (the reader should show that such a vertex is unique). When u is the parent of v, v is called a **child** of u. Vertices with the same parent are called **siblings**. 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. 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 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.

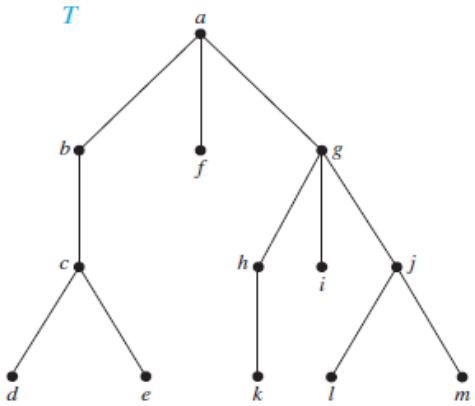


FIGURE 5 A Rooted Tree T.

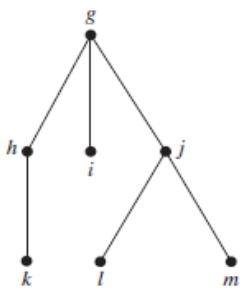
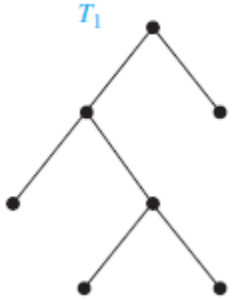


FIGURE 6 The Subtree Rooted at g.

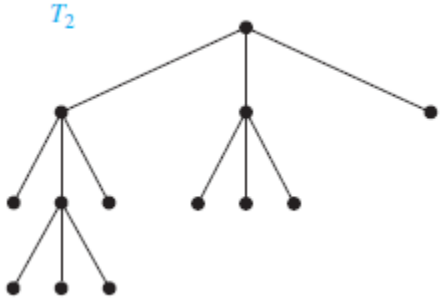
A rooted tree is called an *m-ary tree* if every internal vertex has no more than m children. The tree is called a *full m-ary tree* if every internal vertex has exactly m children. An m-ary tree with $m = 2$ is called a *binary tree*.

T_1



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

T_2

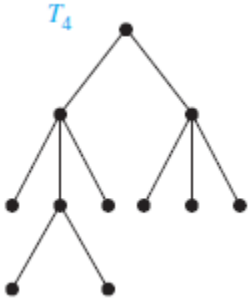


T_2 is a full 3-ary tree because each of its internal vertices has three children.

T_3

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

T_4

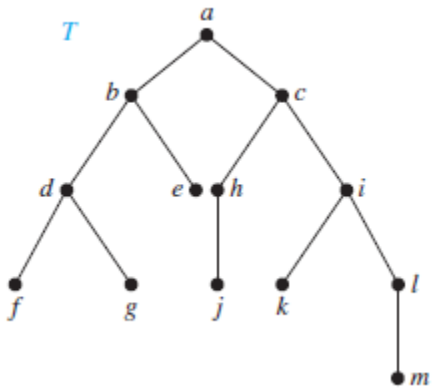


T_4 is not a full m-ary tree for any because some of its internal vertices have two children and others have three children.

ORDERED ROOTED TREES An **ordered rooted tree** is a rooted tree where the children of each internal vertex are ordered. Ordered rooted trees are drawn so that the children of each internal vertex are shown in order from left to right. Note that a representation of a rooted tree in the conventional way determines an ordering for its edges. We will use such orderings of edges in drawings without explicitly mentioning that we are considering a rooted tree to be ordered.

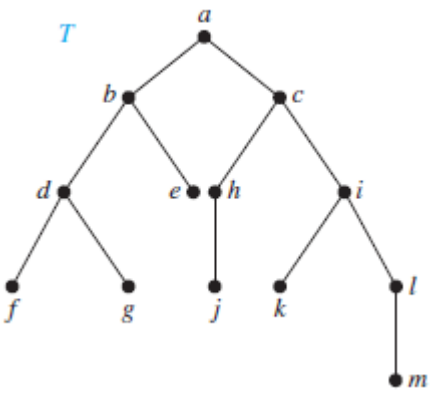
In an ordered binary tree (usually called just a **binary tree**), if an internal vertex has two children, the first child is called the **left child** and the second child is called the **right child**. The tree rooted at the left child of a vertex is called the **left subtree** of this vertex, and the tree rooted at the right child of a vertex is called the **right subtree** of the vertex.

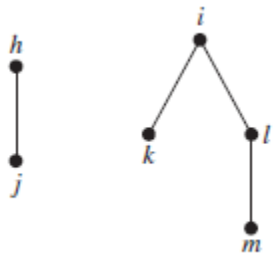
What are the left and right children of d in the binary tree T



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

What are the left and right subtrees of c?





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

Searching for items in a list is one of the most important tasks that arises in computer science. Our primary goal is to implement a searching algorithm that finds items efficiently when the items are totally ordered. This can be accomplished through the use of a **binary search tree**, which is a binary tree in which each child of a vertex is designated as a right or left child, no vertex has more than one right child or left child,

JUST READ:

What is meant by alphabetical order with example?

We often sort letters and words in alphabetical order. This means **to order them as they appear in the alphabet**. When sorting words in to alphabetical order, we look at the first letter of the word. For example: the word c at comes before the word d og because c comes before d in the alphabet.

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

SOLUTION:

The word *mathematics* is the key of the root. Because *physics* comes after *mathematics* (in alphabetical order), add a right child of the root with key *physics*. Because *geography* comes before *mathematics*, add a left child of the root with key *geography*. Next, add a right child of the vertex with key *physics*, and assign it the key *zoology*, because *zoology* comes after *mathematics* and after *physics*. Similarly, add a left child of the vertex with key *physics* and assign this new vertex the key *meteorology*. Add a right child of the vertex with key *geography* and assign this new vertex the key *geology*. Add a left child of the vertex with key *zoology* and assign it the key *psychology*. Add a left child of the vertex with key *geography* and assign it the key *chemistry*.

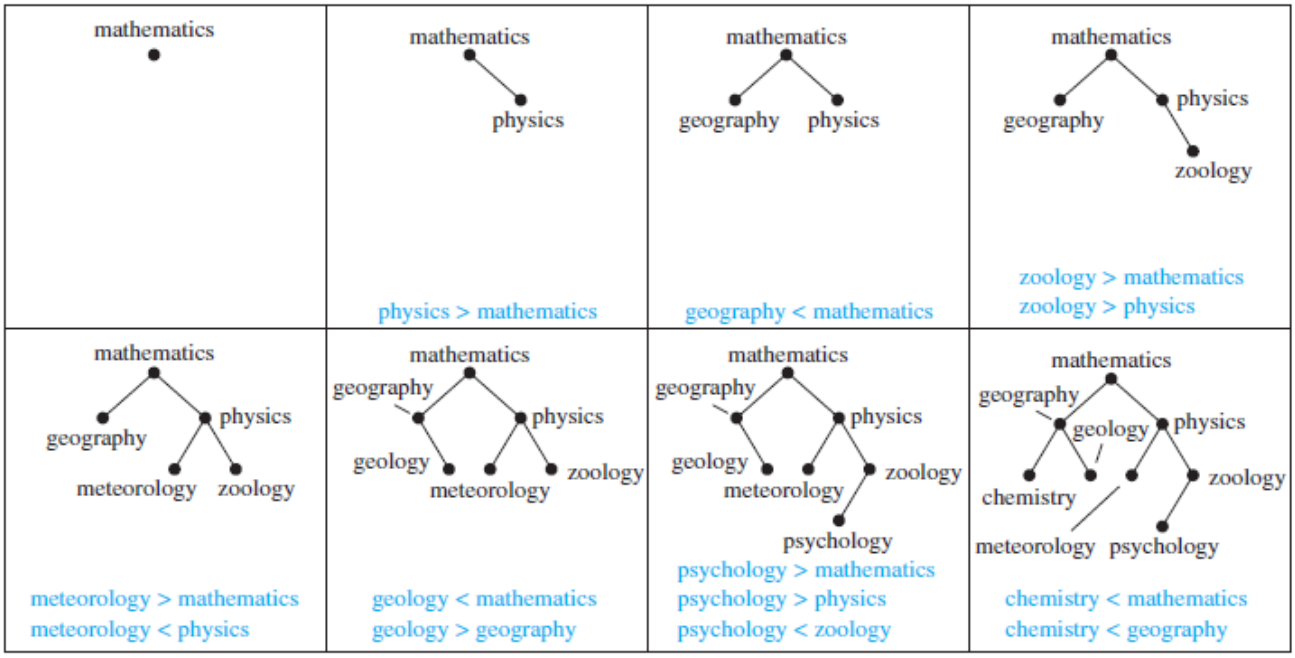


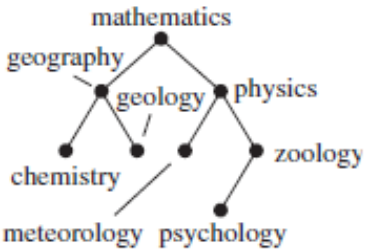
FIGURE 1 Constructing a Binary Search Tree.

1. Compare the ITEM with the root node.
2. If ITEM>ROOT NODE, proceed to the right child, and it becomes a root node for the right subtree.
3. If ITEM<ROOT NODE, proceed to the left child.
4. Repeat the above steps until we meet a node which has no left and right subtree.
5. Now if the ITEM is greater than the node, then the ITEM is inserted as the right child, and if the ITEM is less than the node, then the ITEM is inserted as the left child.

ALGORITHM 1 Locating an Item in or Adding an Item 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 $label(v) \neq x$
  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 $v := null$
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
return v {$v =$ location of x}

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



Solution: Algorithm 1 begins with v, the vertex under examination, equal to the root of T, so $label(v) = mathematics$. Because $v \neq null$ and $label(v) = mathematics < oceanography$, we next examine the right child of the root. This right child exists, so we set v, the vertex under examination, to be this right child. At this step we have $v \neq null$ and $label(v) = physics > oceanography$, so we examine the left child of v. This left child exists, so we set v, the vertex under examination, to this left child. At this step, we also have $v \neq null$ and $label(v) = metereology < oceanography$, so we try to examine the right child of v. However, this right child does not exist, so we add a new vertex as the right child of v (which at this point is the vertex with the key *metereology*) and we set $v := null$. We now exit the **while** loop because $v = null$. Because the root of T is not *null* and $v = null$, we use the **else if** statement at the end of the algorithm to label our new vertex with the key *oceanography*. ◀

Tree Traversal

Introduction

Ordered rooted trees are often used to store information. We need procedures for visiting each vertex of an ordered rooted tree to access data. We will describe several important algorithms for visiting all the vertices of an ordered rooted tree. Ordered rooted trees can also be used to represent various types of expressions, such as arithmetic expressions involving numbers, variables, and operations. The different listings of the vertices of ordered rooted trees used to represent expressions are useful in the evaluation of these expressions.

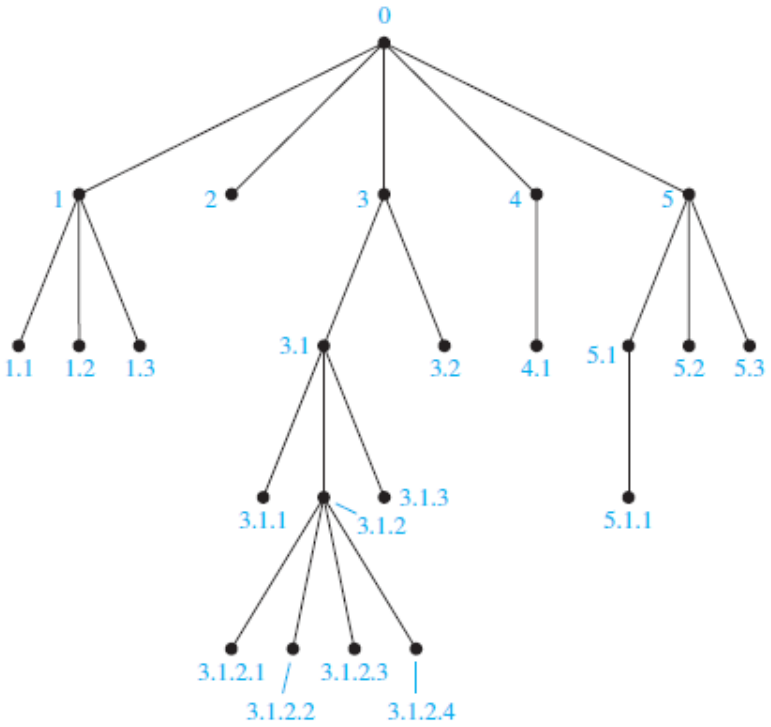
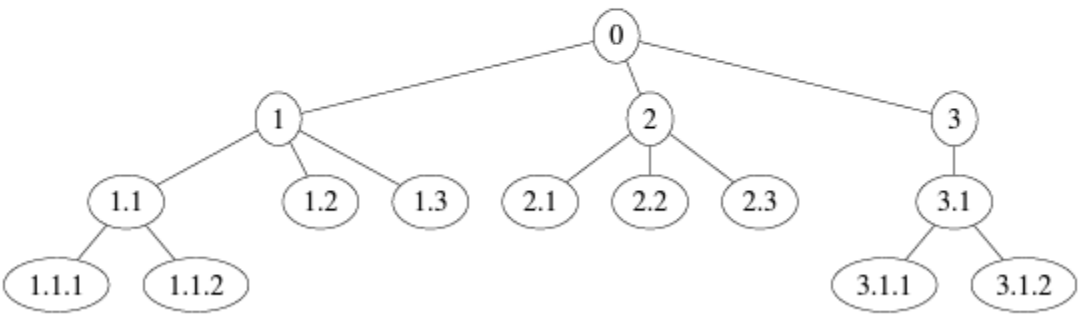


FIGURE 1 The Universal Address System of an Ordered Rooted Tree.

Universal Address Systems

A way to totally order the vertices of and ordered rooted tree. We do this recursively:

- Label the root with the integer 0. Then label its k children (at level 1) from left to right with $1, 2, 3, \ldots, k$.
- For each vertex v at level n with label A, label its k_v children, as they are drawn from left to right, with $A.1, A.2, \ldots, A.k_v$.



The lexicographic ordering for this tree is $0 < 1 < 1.1 < 1.1.1 < 1.1.2 < 1.2 < 1.3 < 2 < 2.1 < 2.2 < 2.3 < 3 < 3.1 < 3.1.1 < 3.1.2$

Traversal Algorithms

Procedures for systematically visiting every vertex of an ordered rooted tree are called **traversal algorithms**. We will describe three of the most commonly used such algorithms, **preorder traversal**, **inorder traversal**, and **postorder traversal**. Each of these algorithms can be defined recursively.

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.

The reader should verify that the preorder traversal of an ordered rooted tree gives the same ordering of the vertices as the ordering obtained using a universal address system.

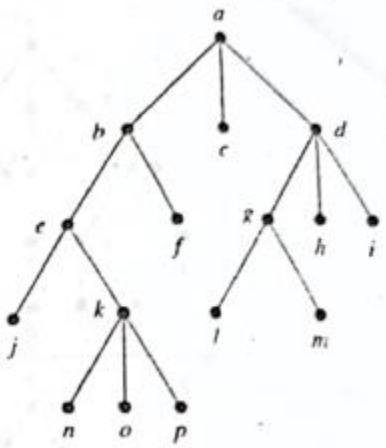


Figure 3 The Ordered Rooted Tree T.

Consequently, the preorder traversal of T is $a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i$.

ANOTHER DEFINITION OF PREORDER TRAVERSAL:

In preorder traversal, **first, root node is visited, then left sub-tree and after that right sub-tree is visited**. The process of preorder traversal can be represented as - root → left → right.

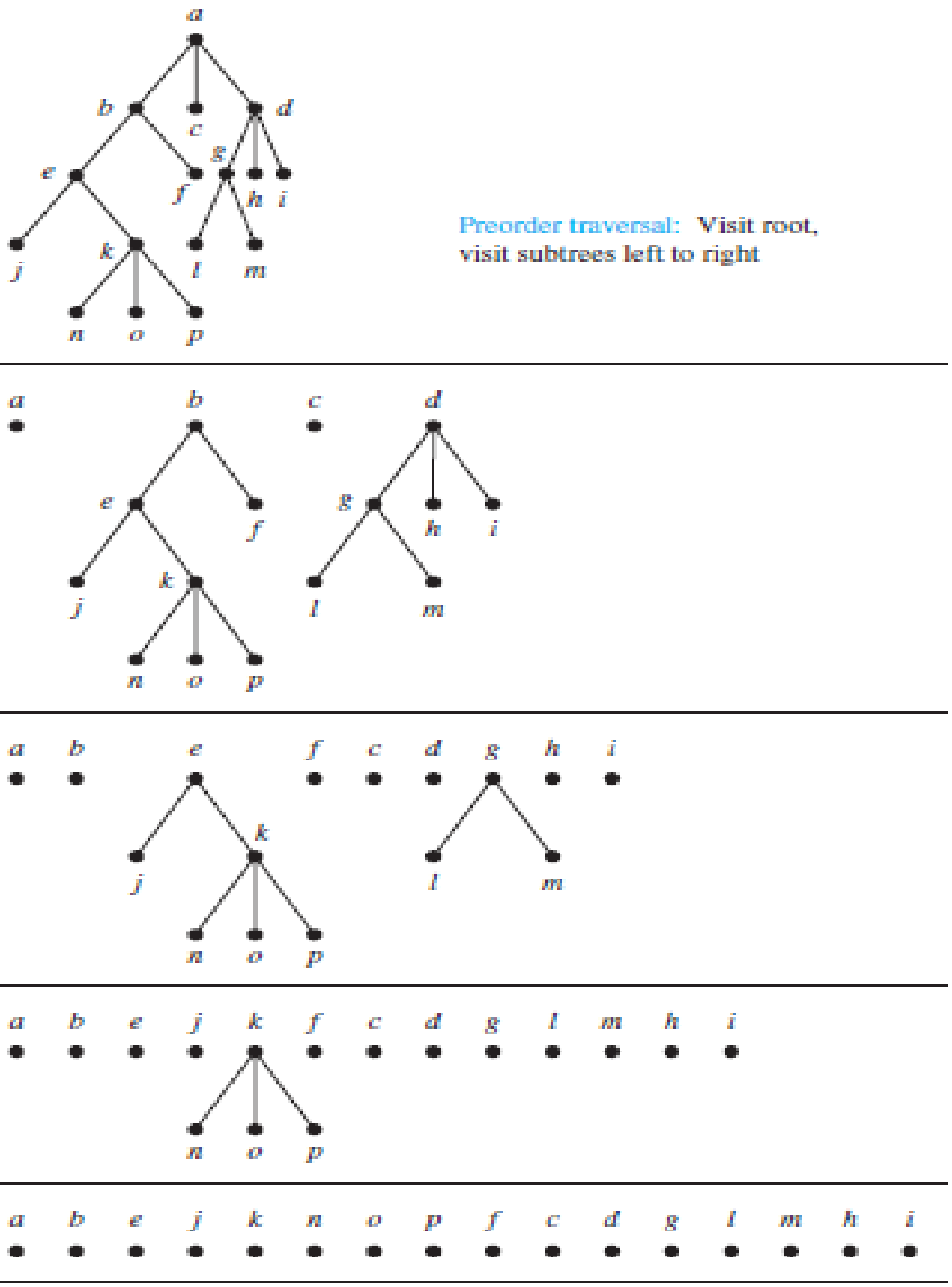


FIGURE 4 **The Preorder Traversal of T.**

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.

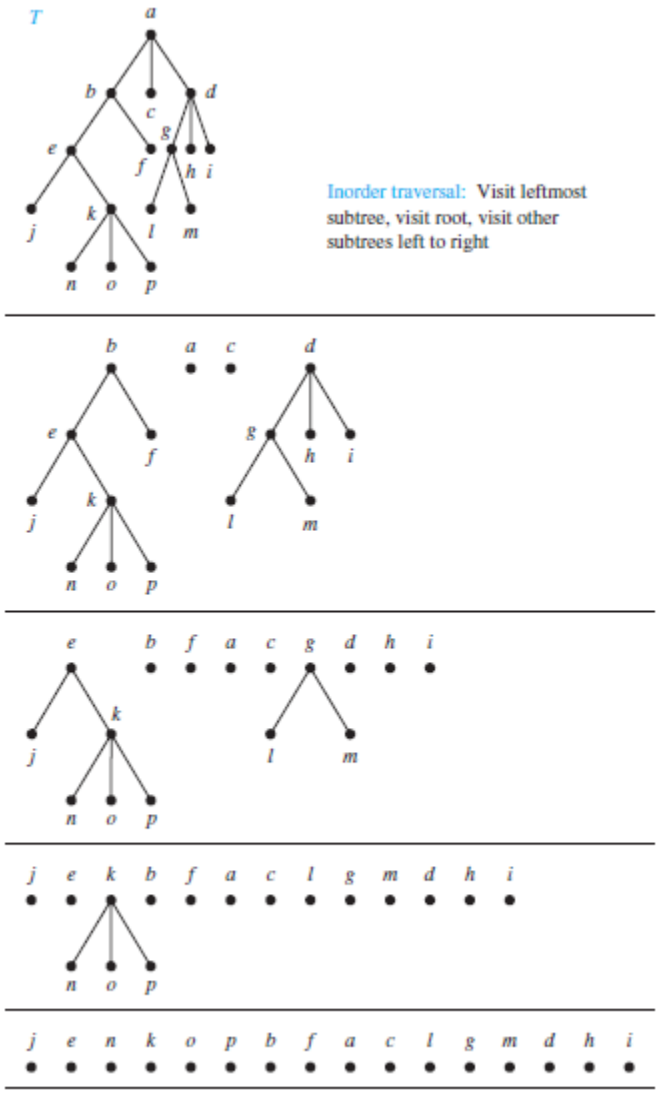


FIGURE 6 **The Inorder Traversal of T.**

Consequently, the inorder listing of the ordered rooted tree is *j*, *e*, *n*, *k*, *o*, *p*, *b*, *f*, *a*, *c*, *l*, *g*, *m*, *d*, *h*, *i*.

For Inorder, you traverse from the left subtree to the root then to the right subtree.

ANOTHER DEFINITION OF INORDER TRAVERSAL:

It means that **first left subtree is visited after that root node is traversed, and finally, the right subtree is traversed**. As the root node is traversed between the left and right subtree, it is named inorder traversal. So, in the inorder traversal, each node is visited in between of its subtrees.

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.

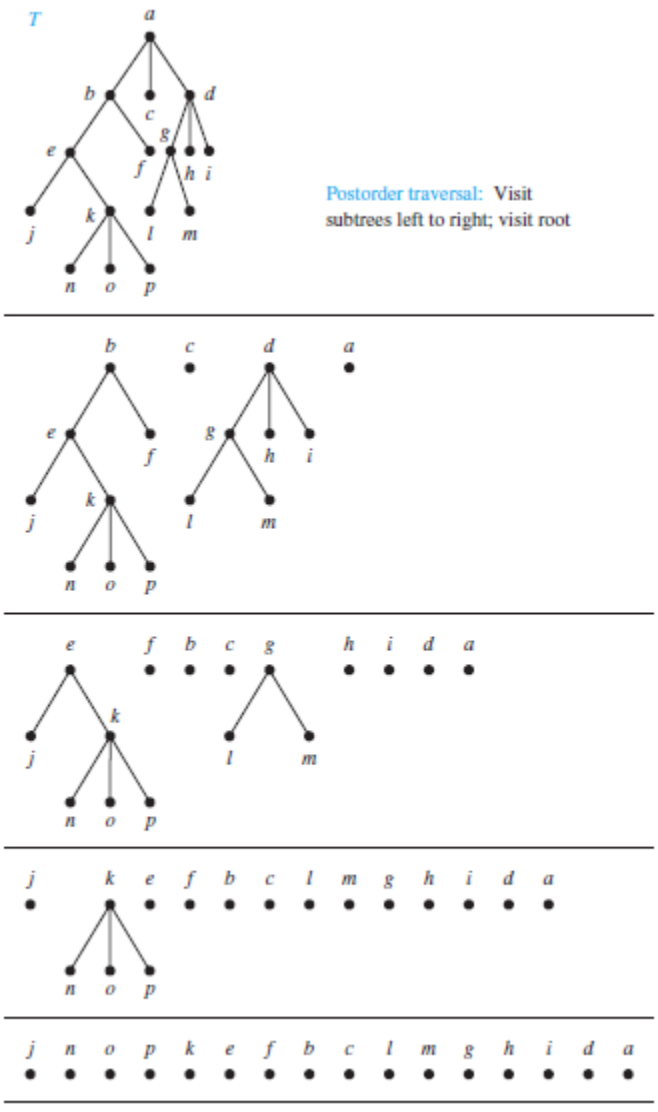
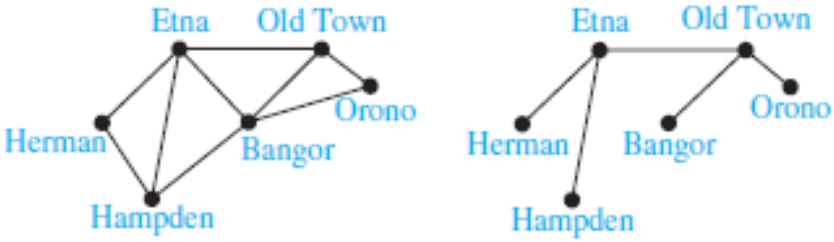


FIGURE 8 The Postorder Traversal of T.

ANOTHER DEFINITION OF POSTORDER TRAVERSAL:

The postorder traversal is **one of the traversing techniques used for visiting the node in the tree**. It follows the principle LRN (Left-right-node). For Post order, you traverse from the left subtree to the right subtree then to the root.



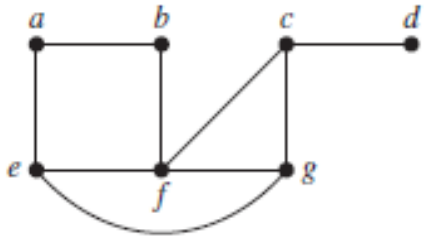
Consider the system of roads in Maine represented by the simple graph shown in Figure 1(a). The only way the roads can be kept open in the winter is by frequently plowing them. The highway department wants to plow the fewest roads so that there will always be cleared roads connecting any two towns. How can this be done?

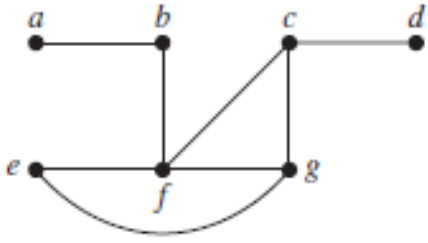
At least five roads must be plowed to ensure that there is a path between any two towns. Figure 1(b) shows one such set of roads. Note that the subgraph representing these roads is a tree, because it is connected and contains six vertices and five edges.

This problem was solved with a connected subgraph with the minimum number of edges containing all vertices of the original simple graph. Such a graph must be a tree.

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.

G:





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

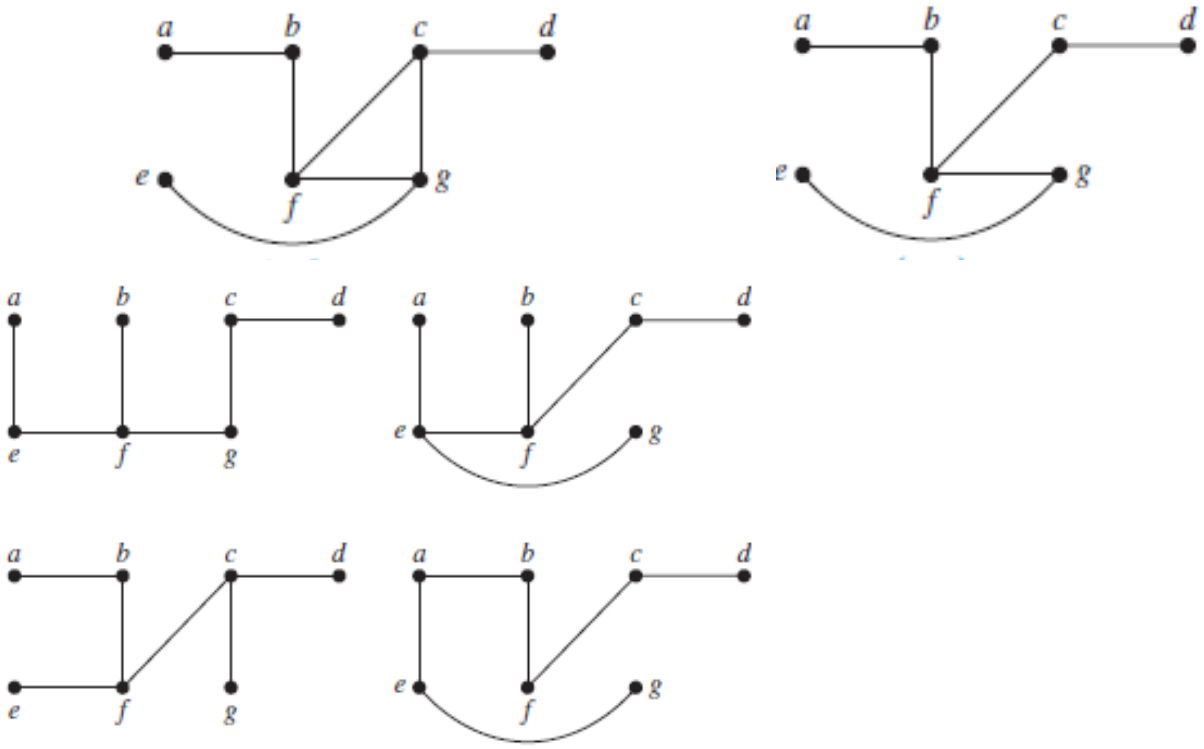


FIGURE 4 **Spanning Trees of G.**

The tree shown in Figure 3 is not the only spanning tree of G. For instance, each of the trees shown in Figure 4 is a spanning tree of G. ◀

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

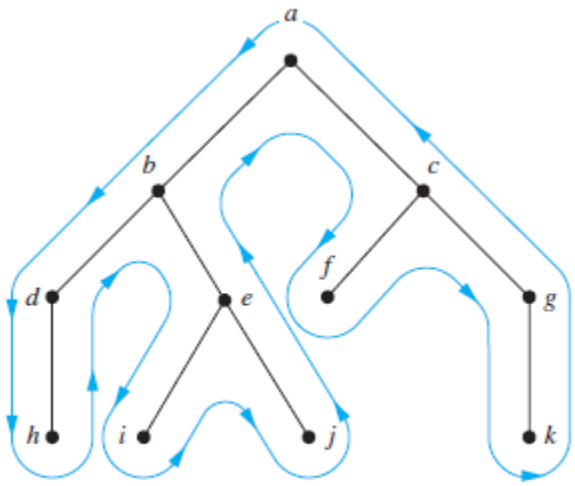


FIGURE 9 **A Shortcut for Traversing an Ordered Rooted Tree in Preorder, Inorder, and Postorder.**

TERMS

tree: a connected undirected graph with no simple circuits

forest: an undirected graph with no simple circuits

rooted tree: a directed graph with a specified vertex, called the root, such that there is a unique path to every other vertex from this root

subtree: a subgraph of a tree that is also a tree

parent of v in a rooted tree: the vertex u such that (u, v) is an edge of the rooted tree

child of a vertex v in a rooted tree: any vertex with v as its parent

sibling of a vertex v in a rooted tree: a vertex with the same parent as v

ancestor of a vertex v in a rooted tree: any vertex on the path from the root to v

balanced tree: a tree in which every leaf is at level h or $h - 1$, where h is the height of the tree

binary search tree: a binary tree in which the vertices are labeled with items so that a label of a vertex is greater than the labels of all vertices in the left subtree of this vertex and is less than the labels of all vertices in the right subtree of this vertex

decision tree: a rooted tree where each vertex represents a possible outcome of a decision and the leaves represent the possible solutions of a problem

game tree: a rooted tree where vertices represents the possible positions of a game as it progresses and edges represent legal moves between these positions

prefix code: a code that has the property that the code of a character is never a prefix of the code of another character

minmax strategy: the strategy where the first player and second player move to positions represented by a child with maximum and minimum value, respectively

value of a vertex in a game tree: for a leaf, the payoff to the first player when the game terminates in the position represented by this leaf; for an internal vertex, the maximum or minimum of the values of its children, for an internal vertex at an even or odd level, respectively

tree traversal: a listing of the vertices of a tree

preorder traversal: a listing of the vertices of an ordered rooted tree defined recursively—the root is listed, followed by the first subtree, followed by the other subtrees in the order they occur from left to right

inorder traversal: a listing of the vertices of an ordered rooted tree defined recursively—the first subtree is listed, followed by the root, followed by the other subtrees in the order they occur from left to right

RESULTS

A graph is a tree if and only if there is a unique simple path between every pair of its vertices.

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

A full m-ary tree with i internal vertices has $mi + 1$ vertices.

The relationships among the numbers of vertices, leaves, and internal vertices in a full m-ary tree (see Theorem 4 in Section 11.1)

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

If an m-ary tree has l leaves, its height h is at least $\lceil \log_m l \rceil$. If the tree is also full and balanced, then its height is $\lceil \log_m l \rceil$.

Huffman coding: a procedure for constructing an optimal binary code for a set of symbols, given the frequencies of these symbols

depth-first search, or backtracking: a procedure for constructing a spanning tree by adding edges that form a path until this is not possible, and then moving back up the path until a vertex is found where a new path can be formed

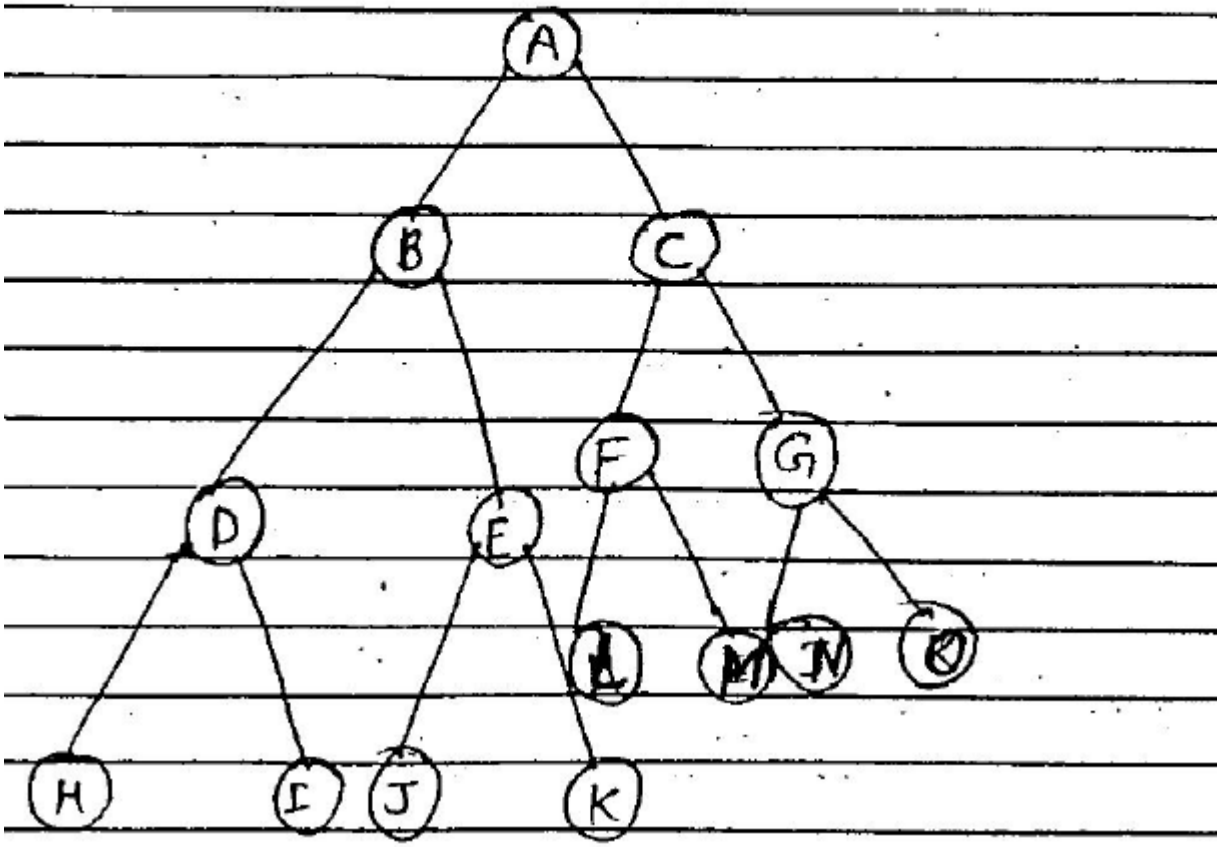
breadth-first search: a procedure for constructing a spanning tree that successively adds all edges incident to the last set of edges added, unless a simple circuit is formed

Prim's algorithm: a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges with minimal weight among all edges incident to a vertex already in the tree so that no edge produces a simple circuit when it is added

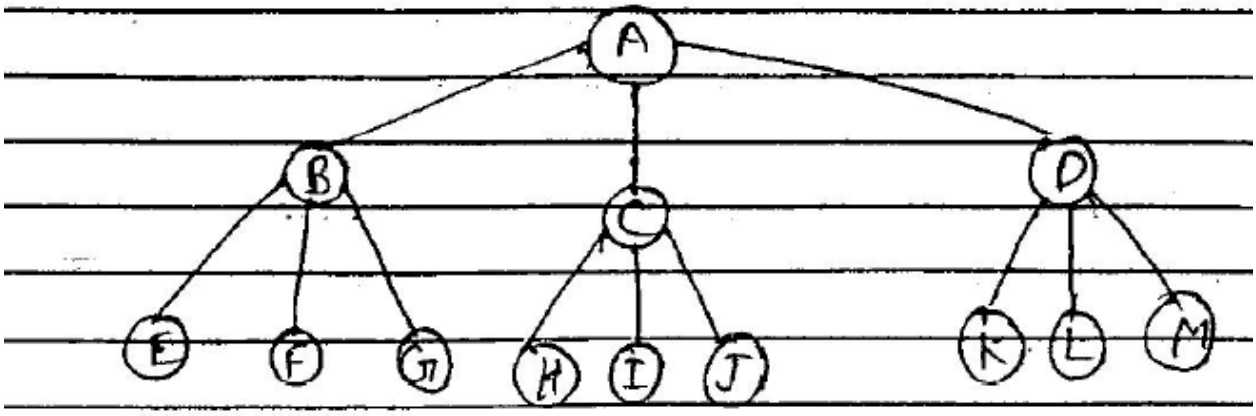
Kruskal's algorithm: a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges of least weight that are not already in the tree such that no edge produces a simple circuit when it is added

Construct a Complete binary tree of height three.

Ans



(b) 3-ary tree

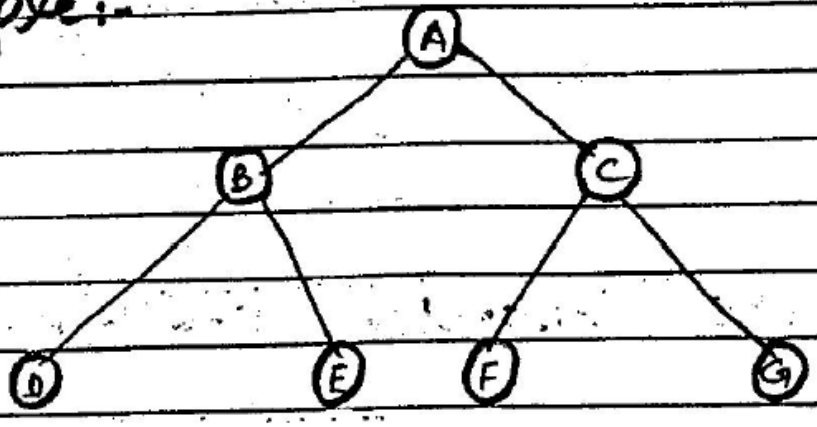


	Path	Circuit
or line	It is a sequence that connects other nodes in the graph.	A circuit is also similar to path with same starting and ending points.

Q.8 What is the height of a rooted tree?

Ans The number of edges from a particular node to a root node is called height of rooted tree.

Example:-



Height A to D = 2

Q.10 Define spanning tree of a graph?

Ans A spanning tree of a graph G (undirected), is a sub-graph that is a tree which includes all of the vertices of G, with minimum possible number of edges.

16. What is the height of rooted tree?

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.

2. Can there be two different simple paths between the vertices of a tree?

2. No, for each ordered pair of vertices u and v, there is a unique simple path from u to v.

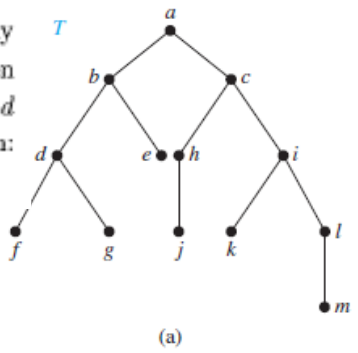
Give at least three examples of how trees are used in modeling.

Trees are used as models in such diverse areas as computer science, chemistry, geology, botany and psychology

1) Graphs can be used to represent molecules, where atoms are represented by vertices and bonds between them by edges.
2) The structure of a large organization can be modeled using a rooted tree. Each vertex in this tree represents a position in the organization.
3) A file system may be represented by a rooted tree, where the root represents the root directory, internal vertices represent subdirectories, and leaves represent ordinary files or empty directories.

Draw a rooted tree with at least 10 vertices, where the degree of each vertex does not exceed 3. Identify the root, the parent of each vertex, the children of each vertex, the internal vertices, and the leaves.

d) Figure 8a in Section 11.1 is such a tree. Its root is a. The parent of each vertex is the vertex immediately above it; thus the parent of b is a, the parent of c is also a, the parent of d is b, and so on. The children of a vertex are the vertices immediately below it; thus the children of a are b and c, the children of b are d and e, the only child of h is j, e has no children, and so on. The internal vertices are the ones with children: a, b, c, d, h, i, and l. The leaves are the vertices without children: e, f, g, j, k, and m.



(a)

FIGURE 8

Discrete structure

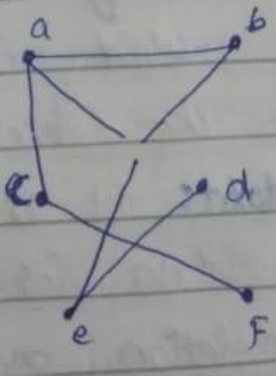
Trees

order rooted collection of data.

Defination of tree:- A tree is a connected undirected graph with no simple circuits.

Example:

Any connected graph that contains no simple circuit is tree. what about graph containing no simple circuits that are ~~only~~ not neccesarily connected? These graph are called Forests. and have the property that each of their connected components is a tree.



- **Trie:** It is a special kind of tree that is used to store the dictionary.

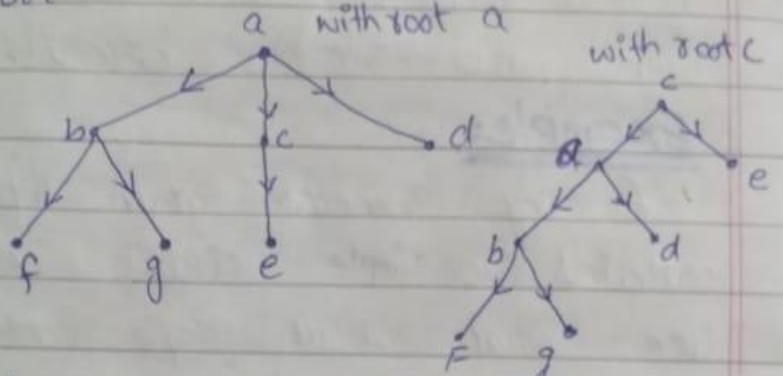
Tree Traversal :-

ordered rooted trees are often used to store information. we need procedures for visiting each vertex of an ordered rooted tree to access data. we will describe several important algorithms for visiting all the vertices of an ordered rooted tree. Ordered rooted tree can also be used to represent various types of expression such as arithmetic expression involving.

Traversal Algorithms: Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithms. we will describe

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.



A Tree and Rooted trees Formed by designating two Roots.

Application of tree:-

There are following application of trees:

- **Storing naturally hierarchical data:** Trees are used to store the data in the hierarchical structure.
- **Organize data:** It is used to organize data for efficient insertion, deletion and searching.

three of the most commonly used such algorithms,

(i) Preorder traversal
(ii) Post in order traversal
(iii) Postorder traversal

Each of these algorithms can be defined recursively.

I Preorder traversal:-

In Preorder traversal Root, Left and right

5. a) How many edges does a tree with n vertices have?
   b) What do you need to know to determine the number of edges in a forest with n vertices?

5. a) $n-1$ b) If c is the number of components, then $e = n - c$.

How many vertices does a full m-ary tree have if it has i internal vertices? How many leaves does the tree have?

i internal vertices has $n = mi + 1$ vertices and $l = (m-1)i + 1$ leaves,

What is the height of a rooted tree?

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.

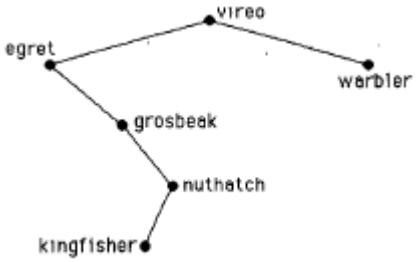
What is a balanced tree?

A balanced binary tree, also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right subtree of any node differ by not more than 1

How many leaves can an *m*-ary tree of height *h* have?

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

Form a binary search tree for the words *vireo, warbler, egret, grosbeak, nuthatch*, and *kingfisher*.



What is a prefix code?

One way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter. Codes with this property are called **prefix codes**.