

# *Discrete Mathematics (151)*

*Department of Mathematics  
College of Sciences  
King Saud University*

# Chapter 5: Trees

## 5.1 Introduction to Trees (11.1 in book).

# Introduction to Trees

- A connected graph that contains no simple circuits is called a tree.
- Trees have been employed to solve problems in a wide variety of disciplines, as the examples in this chapter will show.
- Trees are particularly useful in computer science, where they are employed in a wide range of algorithms. For instance, trees are used to construct efficient algorithms for locating items in a list.
- In this chapter we will focus on a particular type of graph called a tree, so named because such graphs resemble trees. For example, family trees are graphs that represent genealogical charts. Family trees use vertices to represent the members of a family and edges to represent parent. child relationships.

# Introduction to Trees

## DEFINITION 1:

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

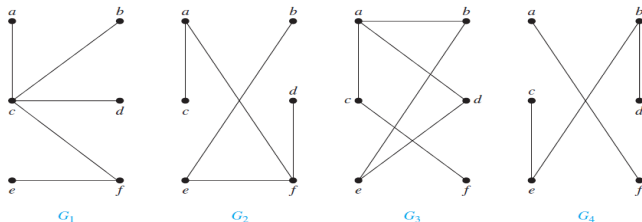


Figure 1: Examples of Trees and Graphs That Are Not Trees.

## Example 1:

Which of the graphs shown in Figure 1 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.

# Introduction to Trees

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 2 displays a forest.

This is one graph with three connected components.

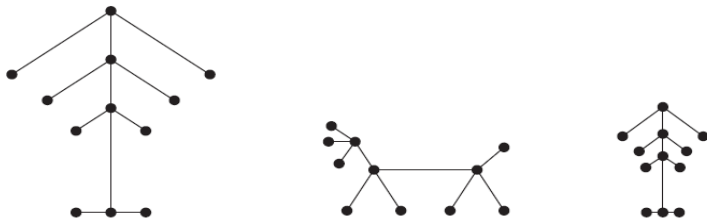


Figure 2: Example of a Forest.

Trees are often defined as undirected graphs with the property that there is a unique simple path between every pair of vertices. Theorem 1 shows that this alternative definition is equivalent to our definition.

## THEOREM 1:

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

**Rooted (Directed) Trees:** In many applications of trees, a particular vertex of a tree is designated as the **root**. Once we specify a root, we can assign a direction to each edge as follows. Because there is a unique path from the root to each vertex of the graph (by Theorem 1), we direct each edge away from the root. Thus, a tree together with its root produces a directed graph called a **rooted tree**.

# Introduction to Trees

## DEFINITION 2:

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.

We can change an unrooted tree into a rooted tree by choosing any vertex as the root. Different choices of the root produce different rooted trees. For instance, Figure 3 displays the rooted trees formed by designating  $a$  to be the root and  $c$  to be the root, respectively, in the tree  $T$ .

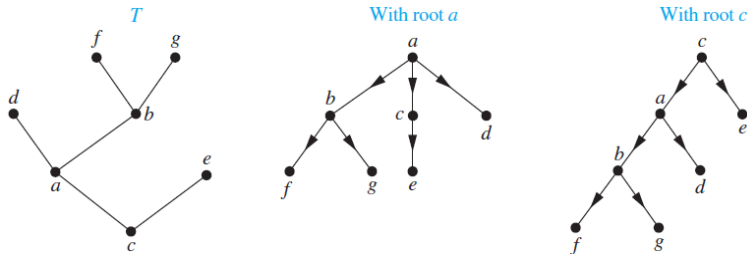


Figure 3: A Tree and Rooted Trees Formed by Designating Two Different Roots.



# Introduction to Trees

The terminology for trees has botanical and genealogical origins. 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**.
- 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.
- 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**.
- 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.

# Introduction to Trees

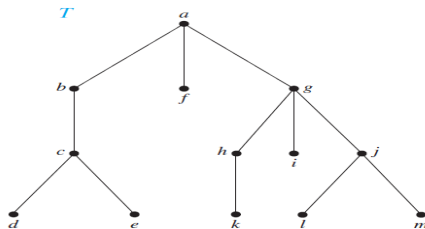


Figure 4: A Rooted Tree  $T$ .

## Example 2:

In the rooted tree  $T$  (with root  $a$ ) shown in Figure 4, 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. What is the subtree rooted at  $g$ ?

**Solution:** 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  $e$ . The internal vertices are  $a$ ,  $b$ ,  $c$ ,  $g$ ,  $h$ , and  $j$ . The leaves are  $d$ ,  $e$ ,  $f$ ,  $i$ ,  $k$ ,  $l$ , and  $m$ . The subtree rooted at  $g$  is shown in Figure 5.

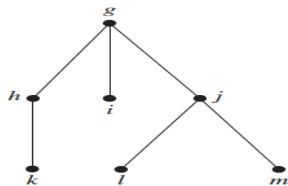


Figure 5: The Subtree Rooted at  $g$ .

## DEFINITION:

- **Level (of  $v$ )** is the length of the unique path from the root to  $v$ .
- **Height** is maximum of vertices levels.

## DEFINITION 3:

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 3:

Are the rooted trees in Figure 6 full  $m$ -ary trees for some positive integer  $m$ ?

# Introduction to Trees

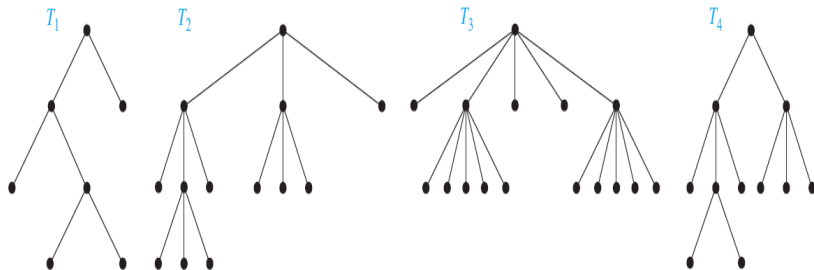


Figure 6: Rooted trees.

**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

- 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.
- 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.
- For  $m$ -ary trees with  $m > 2$ , we can use terms like **leftmost**, **rightmost**, etc.

# Introduction to Trees

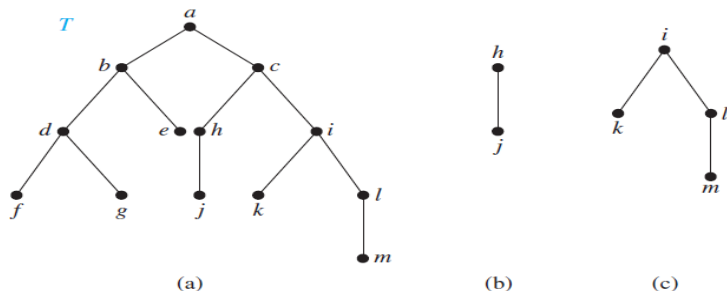


Figure 7: Rooted trees.

## Example 4:

What are the left and right children of  $d$  in the binary tree  $T$  shown in Figure 7(a) (where the order is that implied by the drawing)? What are the left and right subtrees of  $c$ ?

**Solution:** The left child of  $d$  is  $f$  and the right child is  $g$ . We show the left and right subtrees of  $c$  in Figures 7(b) and 7(c), respectively.

# Introduction to Trees

## Trees as Models

Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, and psychology. We will describe a variety of such models based on trees.

### Example 5: Saturated Hydrocarbons and Trees

Graphs can be used to represent molecules, where atoms are represented by vertices and bonds between them by edges.

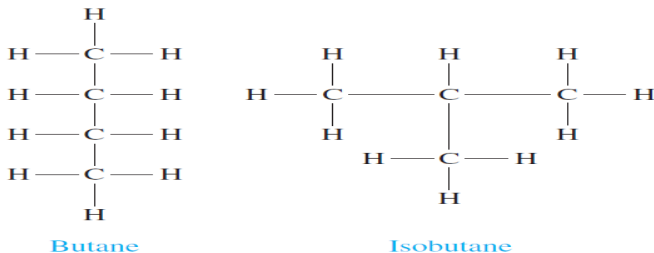


Figure 8: The Two Isomers of Butane.



## Example 6: Representing Organizations

The structure of a large organization can be modeled using a rooted tree. Each vertex in this tree represents a position in the organization. An edge from one vertex to another indicates that the person represented by the initial vertex is the (direct) boss of the person represented by the terminal vertex.

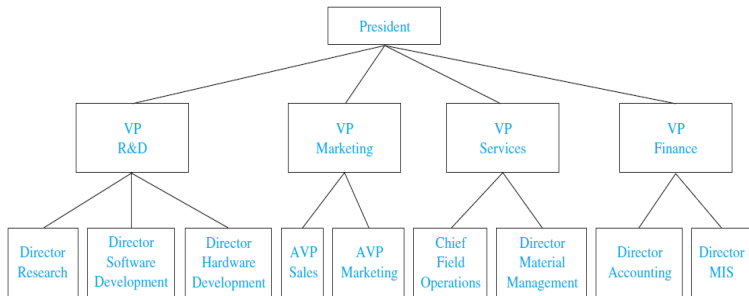


Figure 9: An Organizational Tree for a Computer Company.

## Example 7: Computer File Systems

Files in computer memory can be organized into directories. A directory can contain both files and subdirectories. The root directory contains the entire file system. Thus, 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

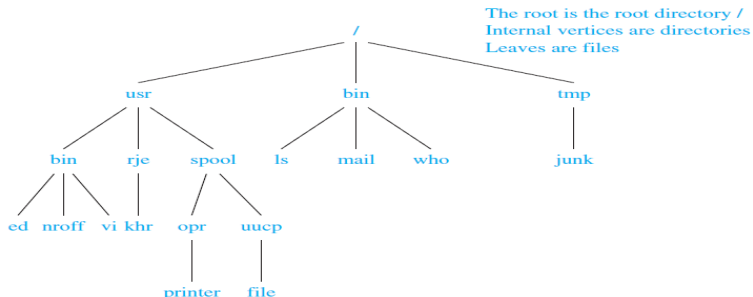


Figure 10: A Computer File System.

## Properties of Trees

We will often need results relating the numbers of edges and vertices of various types in trees.

### THEOREM 2:

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

### THEOREM 3:

A full  $m$ -ary tree with  $i$  internal vertices and  $l$  leaves contains  $n = mi + 1$  ( $n = i + l$ ) vertices.

## THEOREM 4:

A full  $m$ -ary tree with

- 1  $n$  vertices has  $i = (n - 1)/m$  internal vertices and  $l = [(m - 1)n + 1]/m$  leaves,
- 2  $i$  internal vertices has  $n = mi + 1$  vertices and  $l = (m - 1)i + 1$  leaves,
- 3  $l$  leaves has  $n = (ml - 1)/(m - 1)$  vertices and  $i = (l - 1)/(m - 1)$  internal vertices.

## Example 8 (9 in book):

Suppose that someone starts a chain letter. Each person who receives the letter is asked to send it on to four other people. Some people do this, but others do not send any letters. How many people have seen the letter, including the first person, if no one receives more than one letter and if the chain letter ends after there have been 100 people who read it but did not send it out? How many people sent out the letter?

**Solution:** The chain letter can be represented using a 4-ary tree. The internal vertices correspond to people who sent out the letter, and the leaves correspond to people who did not send it out. Because 100 people did not send out the letter, the number of leaves in this rooted tree is  $l = 100$ . Hence, part (3) of Theorem 4 shows that the number of people who have seen the letter is  $n = (4 \times 100 - 1)/(4 - 1) = 133$ . Also, the number of internal vertices is  $133 - 100 = 33$ , so 33 people sent out the letter.

- 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 9 (10 in book):

Find the level of each vertex in the rooted tree shown in Figure 11. What is the height of this tree?

# Introduction to Trees

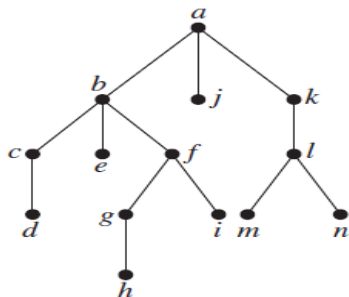


Figure 11: A Rooted Tree.

**Solution:** 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. Finally, vertex  $h$  is at level 4. Because the largest level of any vertex is 4, this tree has height 4.

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

# Introduction to Trees

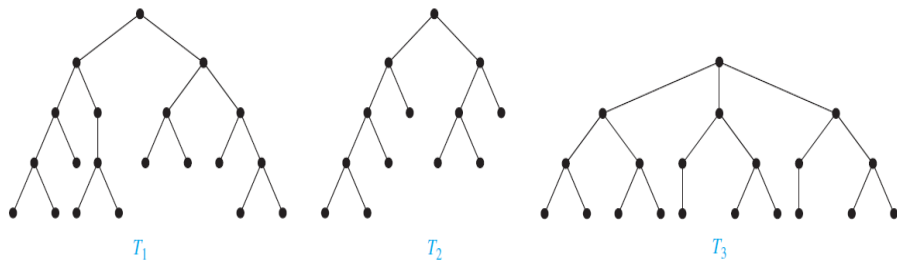


Figure 12: A Rooted Tree.

## Example 10 (11 in book):

Which of the rooted trees shown in Figure 12 are balanced?

**Solution:**  $T_1$  is balanced, because all its leaves are at levels 3 and 4. However,  $T_2$  is not balanced, because it has leaves at levels 2, 3, and 4. Finally,  $T_3$  is balanced, because all its leaves are at level 3.



## 5.2 Applications of Trees (11.2 in book).

# Applications of Trees

We will discuss three problems that can be studied using trees.

- The first problem is: How should items in a list be stored so that an item can be easily located?
- The second problem is: What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?
- The third problem is: How should a set of characters be efficiently coded by bit strings?

## Binary Search Trees:




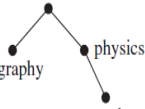
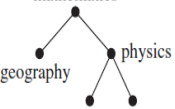
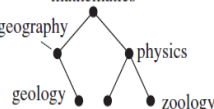
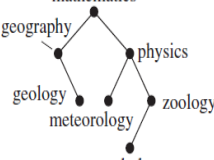
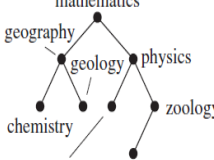
- Searching for items in a list is one of the most important tasks that arises in computer science.
- Our primary goal is to implement a searching algorithm that finds items efficiently when the items are totally ordered. This can be accomplished through the use of a **binary search tree**.
- A **binary search tree** is 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, and each vertex is labeled with a key, which is one of the items. Furthermore, vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

## Example 1:

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

**Solution:** Figure 13 displays the steps used to construct this binary search tree. 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**.

# Applications of Trees

<p>mathematics</p> 	<p>mathematics</p>  <p>physics</p> <p>physics &gt; mathematics</p>	<p>mathematics</p>  <p>geography physics</p> <p>geography &lt; mathematics</p>	<p>mathematics</p>  <p>geography physics</p> <p>zoology</p> <p>zoology &gt; mathematics zoology &gt; physics</p>
<p>mathematics</p>  <p>geography physics</p> <p>meteorology zoology</p> <p>meteorology &gt; mathematics meteorology &lt; physics</p>	<p>mathematics</p>  <p>geography physics</p> <p>geology meteorology zoology</p> <p>geology &lt; mathematics geology &gt; geography</p>	<p>mathematics</p>  <p>geography physics</p> <p>geology meteorology zoology</p> <p>psychology</p> <p>psychology &gt; mathematics psychology &gt; physics psychology &lt; zoology</p>	<p>mathematics</p>  <p>geography physics</p> <p>chemistry geology zoology</p> <p>meteorology psychology</p> <p>chemistry &lt; mathematics chemistry &lt; geography</p>

## Decision Trees:

- Rooted trees can be used to model problems in which a series of decisions leads to a solution.
- For instance, a binary search tree can be used to locate items based on a series of comparisons, where each comparison tells us whether we have located the item, or whether we should go right or left in a subtree.
- 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**.
- The possible solutions of the problem correspond to the paths to the leaves of this rooted tree.

# Applications of Trees

## Example 2:

We display in Figure 14 a decision tree that orders the elements of the list  $a, b, c$ .

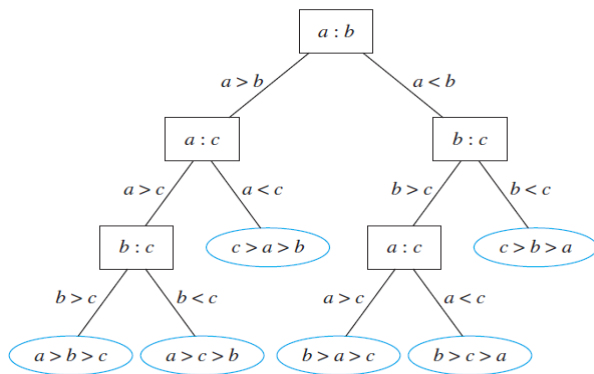


Figure 13: A decision Tree.

## Prefix Codes:

- Consider using bit strings of different lengths to encode letters.
- When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end.
- 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**.
- A prefix code can be represented using a binary tree, where the characters are the labels of the leaves in the tree. The edges of the tree are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned a 1.



# Applications of Trees

- 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.
- For instance, the tree in Figure 15 represents the encoding of e by 0, a by 10, t by 110, n by 1110, and s by 1111.

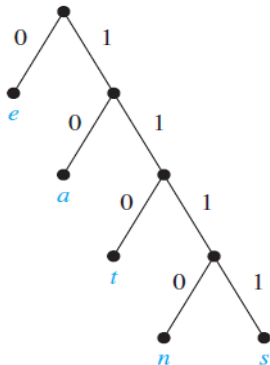


Figure 14: A Binary Tree with a Prefix Code.

# Applications of Trees

- The tree representing a code can be used to decode a bit string. For instance, consider the word encoded by 11111011100 using the code in Figure 15. This bit string can be decoded by starting at the root, using the sequence of bits to form a path that stops when a leaf is reached.

## 5.3 Tree Traversal (11.3 in book).

# 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 trees can also be used to represent various types of expressions, such as arithmetic expressions involving numbers, variables, and operations.

## Traversal Algorithms:

Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithms.

## 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,**
- **postorder traversal.**

## DEFINITION 1: Preorder

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  $T_1, T_2, \dots, T_n$  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.

## Example 1:

In which order does a preorder traversal visit the vertices in the ordered rooted tree  $T$  shown in Figure 15?

**Solution:** The steps of the preorder traversal of  $T$  are shown in Figure 17.

# Tree Traversal

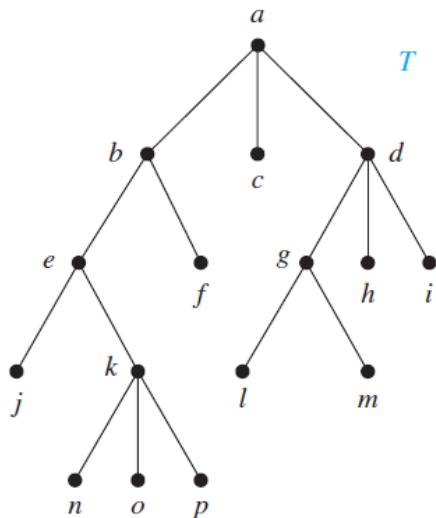
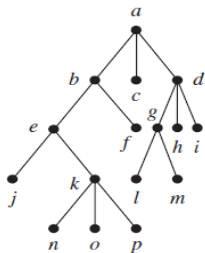
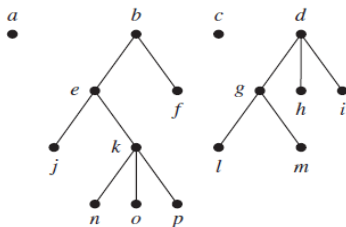


Figure 15: The Ordered Rooted Tree  $T$ .

# Tree Traversal



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





# Tree Traversal

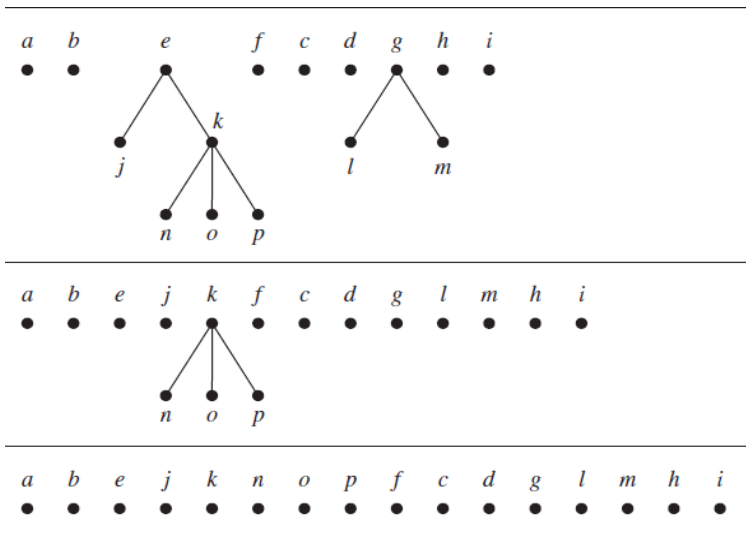


Figure 16: The Preorder Traversal of  $T$ .

## DEFINITION 2: Inorder

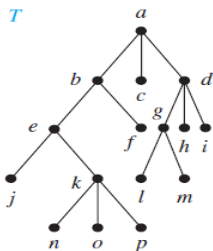
Let  $T$  be an ordered rooted tree with root  $r$ . If  $T$  consists only of  $r$ , then  $r$  is the inorder traversal of  $T$ . Otherwise, suppose that  $T_1, T_2, \dots, T_n$  are the subtrees at  $r$  from left to right. The inorder traversal begins by traversing  $T_1$  in inorder, then visiting  $r$ . It continues by traversing  $T_2$  in inorder, then  $T_3$  in inorder,  $\dots$ , and finally  $T_n$  in inorder.

## Example 2:

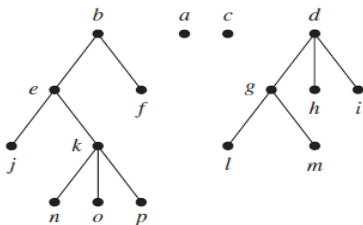
In which order does an inorder traversal visit the vertices of the ordered rooted tree  $T$  in Figure 15?

**Solution:** The steps of the inorder traversal of  $T$  are shown in Figure 17.

# Tree Traversal



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



# Tree Traversal

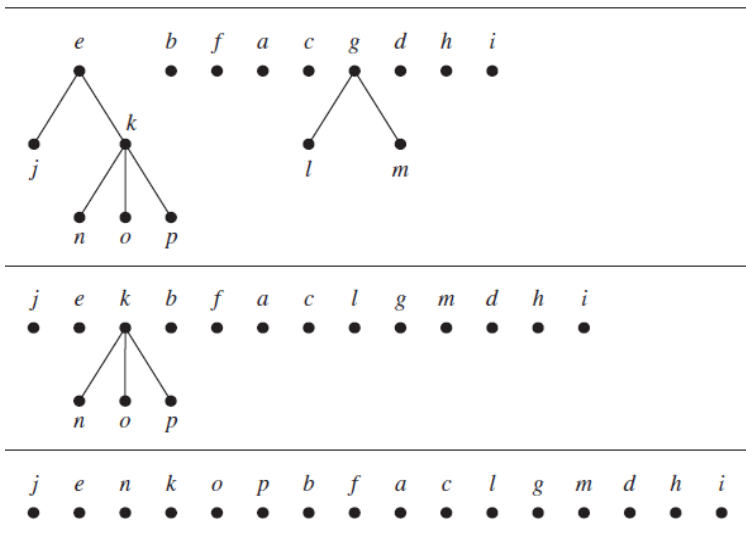


Figure 17: The Inorder Traversal of T.

## DEFINITION 3: Postorder

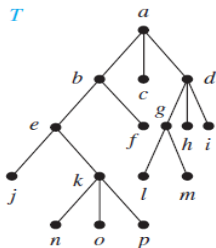
Let  $T$  be an ordered rooted tree with root  $r$ . If  $T$  consists only of  $r$ , then  $r$  is the postorder traversal of  $T$ . Otherwise, suppose that  $T_1, T_2, \dots, T_n$  are the subtrees at  $r$  from left to right. The postorder traversal begins by traversing  $T_1$  in postorder, then  $T_2$  in postorder,  $\dots$ , then  $T_n$  in postorder, and ends by visiting  $r$ .

## Example 3:

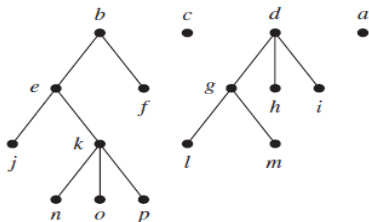
In which order does a postorder traversal visit the vertices of the ordered rooted tree  $T$  shown in Figure 15?

**Solution:** The steps of the postorder traversal of the ordered rooted tree  $T$  are shown in Figure 18.

# Tree Traversal



Postorder traversal: Visit subtrees left to right; visit root



# Tree Traversal

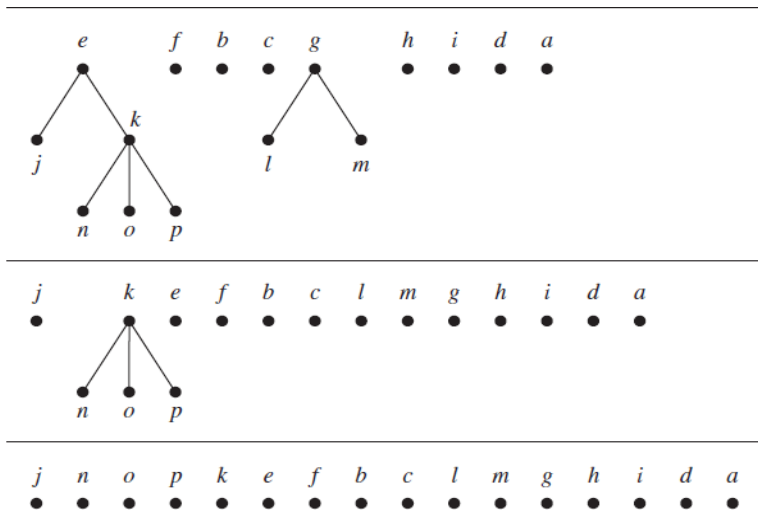


Figure 18: The Postorder Traversal of T.

## Traversal Algorithms

Is visiting every vertex of ordered rooted tree.

- **Preorder traversal:** **R**oot, **L**eft, **R**ight.
- **Inorder traversal:** **L**eft, **R**oot, **R**ight.
- **Postorder traversal:** **L**eft, **R**ight, **R**oot.



## Infix, Prefix, and Postfix Notation:

We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees. For instance, consider the representation of an arithmetic expression involving the operators  $+$  (addition),  $-$  (subtraction),  $*$  (multiplication),  $/$  (division), and  $\uparrow$  (exponentiation). We will use parentheses to indicate the order of the operations. 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. Each operation operates on its left and right subtrees (in that order).

## Example 4:

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

**Solution:** The binary tree for this expression can be built from the bottom up. First, a subtree for the expression  $x + y$  is constructed. Then this is incorporated as part of the larger subtree representing  $(x + y) \uparrow 2$ . Also, a subtree for  $x4$  is constructed, and then this is incorporated into a subtree representing  $(x4)/3$ . Finally the subtrees representing  $((x + y) \uparrow 2)$  and  $((x - 4)/3)$  are combined to form the ordered rooted tree representing  $((x + y) \uparrow 2) + ((x - 4)/3)$ . These steps are shown in Figure 19.

# Tree Traversal

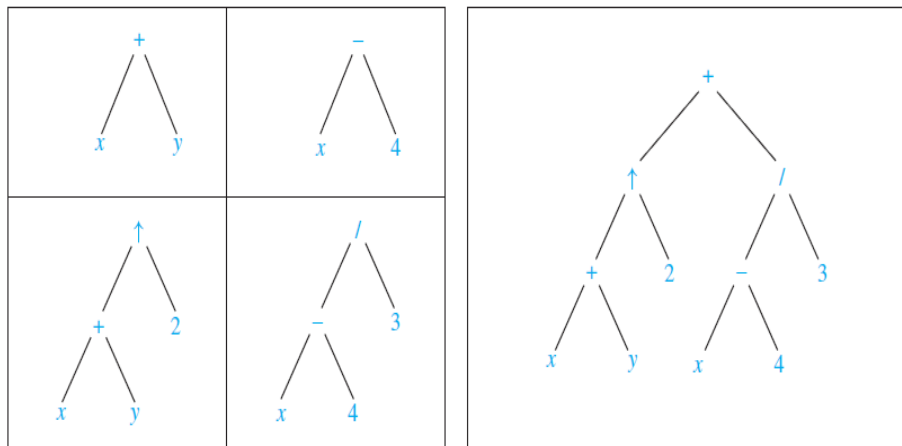


Figure 19: A Binary Tree Representing  $((x + y) \uparrow 2) + ((x - 4) / 3)$ .

# Tree Traversal

An inorder traversal of the binary tree representing an expression produces the original expression with the elements and operations in the same order as they originally occurred, except for unary operations, which instead immediately follow their operands. For instance, inorder traversals of the binary trees in Figure 20, which represent the expressions  $(x + y)/(x + 3)$ ,  $(x + (y/x)) + 3$ , and  $x + (y/(x + 3))$ , all lead to the infix expression  $x + y/x + 3$ .

To make such expressions unambiguous it is necessary to include parentheses in the inorder traversal whenever we encounter an operation.

# Tree Traversal

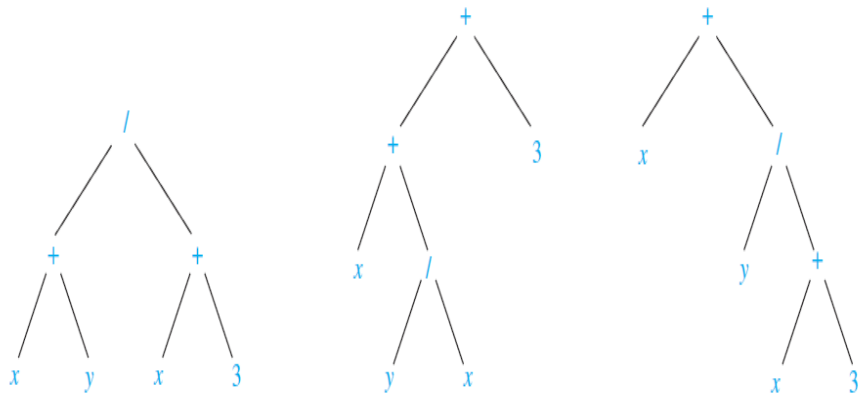


Figure 20: Rooted Trees Representing  $(x + y)/(x + 3)$ ,  $(x + (y/x)) + 3$ , and  $x + (y/(x + 3))$ .

- The fully parenthesized expression obtained in this way is said to be in **infix form**.
- We obtain the **prefix form** of an expression when we traverse its rooted tree in preorder.
- We obtain the **postfix form** of an expression by traversing its binary tree in postorder.

## example 5:

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

**Solution:** We obtain the prefix form for this expression by traversing the binary tree that represents it in preorder, shown in Figure 19. This produces  $+ \uparrow + x y 2 / - x 4 3$ .

## example 6:

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

**Solution:** The postfix form of the expression is obtained by carrying out a postorder traversal of the binary tree for this expression, shown in Figure 19. This produces the postfix expression:  $x y + 2 \uparrow x 4 - 3 / +$ .

# Tree Traversal

## example 7:

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

**Solution:** The steps used to evaluate this expression by working right to left, and performing operations using the operands on the right, are shown in Figure 21. The value of this expression is 3.

$$\begin{array}{cccccccc} + & - & * & 2 & 3 & 5 & / & \uparrow & 2 & 3 & 4 \\ & & & & & & & \underbrace{\phantom{\uparrow 2 3}} & & & \\ & & & & & & & 2 \uparrow 3 = 8 & & & \\ + & - & * & 2 & 3 & 5 & / & 8 & 4 & & \\ & & & & & & & \underbrace{\phantom{8 4}} & & & \\ & & & & & & & 8 / 4 = 2 & & & \\ + & - & * & 2 & 3 & 5 & 2 & & & & \\ & & & \underbrace{\phantom{2 3}} & & & & & & & \\ & & & 2 * 3 = 6 & & & & & & & \\ + & - & * & 2 & 3 & 5 & 2 & & & & \\ & & & \underbrace{\phantom{6 5}} & & & & & & & \\ & & & 6 - 5 = 1 & & & & & & & \\ + & - & * & 2 & 3 & 5 & 2 & & & & \\ & & & \underbrace{\phantom{1 2}} & & & & & & & \\ & & & 1 + 2 = 3 & & & & & & & \\ \text{Value of expression: } & & & 3 & & & & & & & \end{array}$$

Figure 21: Evaluating a Prefix Expression.



# Tree Traversal

## example 8:

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

**Solution:** The steps used to evaluate this expression by starting at the left and carrying out operations when two operands are followed by an operator are shown in Figure 22. The value of this expression is 4.

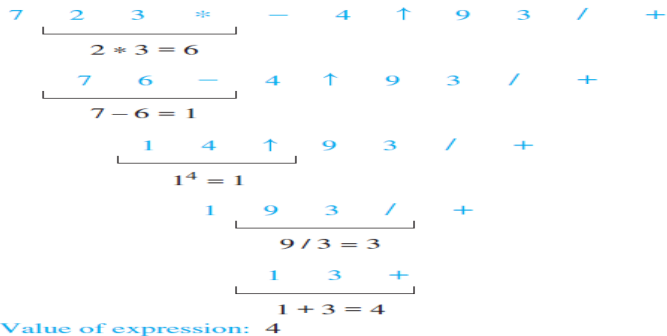


Figure 22: Evaluating a Postfix Expression.

## 5.4 Spanning Trees (11.4 in book).

# Spanning Trees

## Definition 1:

Let  $G$  be a simple graph. A **spanning tree** of  $G$  is a subgraph of  $G$  that is a tree containing every vertex of  $G$ .

## example 1:

Find a spanning tree of the simple graph  $G$  shown in Figure 23.

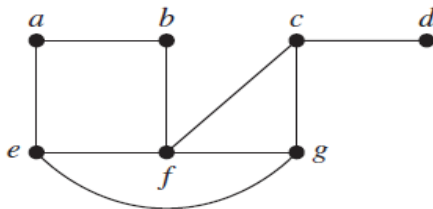
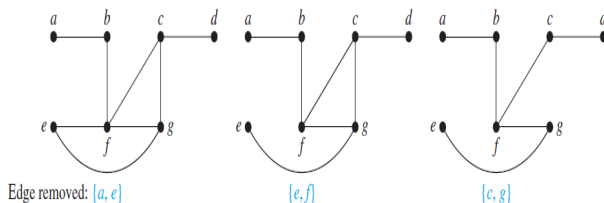


Figure 23: The Simple Graph  $G$ .

# Spanning Trees

**Solution:** The graph  $G$  is connected, but it is not a tree because it contains simple circuits. Remove the edge  $\{a, e\}$ . This eliminates one simple circuit, and the resulting subgraph is still connected and still contains every vertex of  $G$ . Next remove the edge  $\{e, f\}$  to eliminate a second simple circuit. Finally, remove edge  $\{c, g\}$  to produce a simple graph with no simple circuits. This subgraph is a spanning tree, because it is a tree that contains every vertex of  $G$ . The sequence of edge removals used to produce the spanning tree is illustrated in Figure 24.



**Figure 24:** Producing a Spanning Tree for  $G$  by Removing Edges That Form Simple Circuits.

# Spanning Trees

The tree shown in Figure 24 is not the only spanning tree of  $G$ . For instance, each of the trees shown in Figure 25 is a spanning tree of  $G$ .

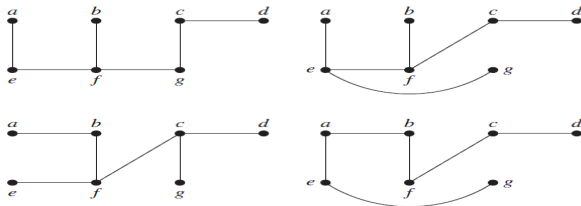
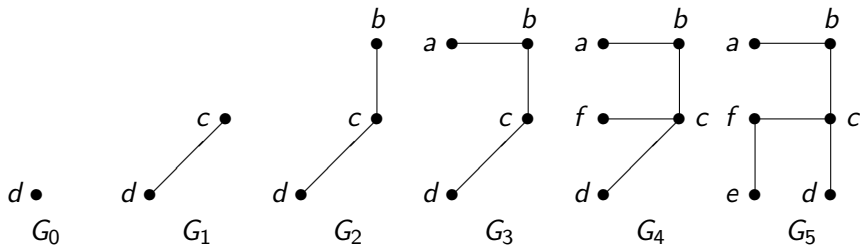
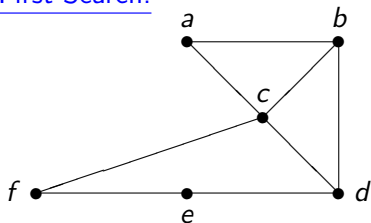


Figure 25: Spanning Trees of  $G$ .

# Spanning Trees

## Depth-First Search:



# Spanning Trees

## Breadth-First Search:

