Discrete Mathematics (MATH 151)

Dr. Borhen Halouani

King Saud University

February 9, 2020

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

Introduction to Trees

In mathematics, a **tree** is a connected graph that does not contain any circuits.

Definition 2.1

A graph is said to be circuit-free if, and only if, it has no circuits.
A graph is called a tree if, and only if, it is circuit-free and connected.
A trivial tree is a graph that consists of a single vertex.
A graph is called a forest if, and only if it is circuit-free and disconnected.

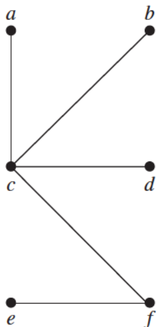
Remark

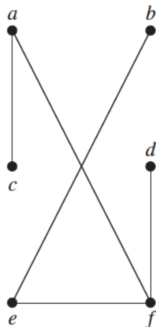
Because a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops. Therefore any tree must be a simple graph.

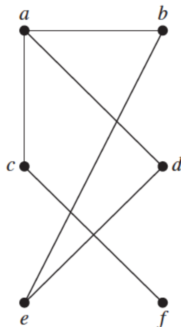
Introduction to Trees

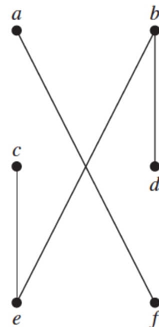
Example:

Which of the graphs shown in Figure are trees and which forest?







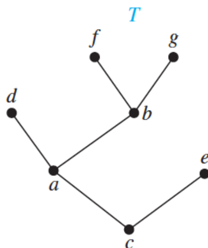


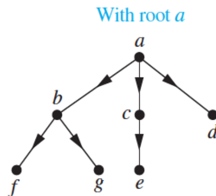
Theorem 2.1

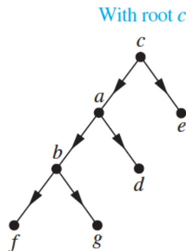
An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Definition 2.2

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.





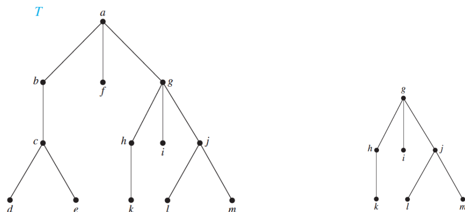


Terminology

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.

Example:



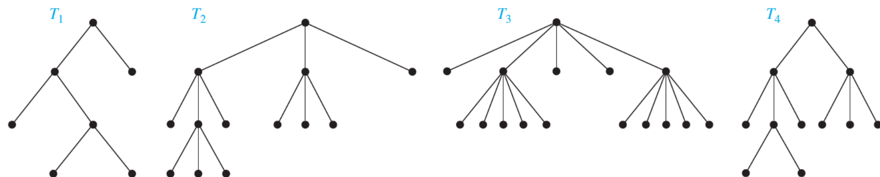
- The parent of c is b.
- The children of g are h, i, and j .
- The siblings of h are i and j .
- The ancestors of e are c, b, and a.
- The descendants of b are c, d, and e.
- The internal vertices are a, b, c, g, h, and j .
- The leaves are d, e, f , i, k, l, and m.
- The subtree rooted at g is shown in Figure.

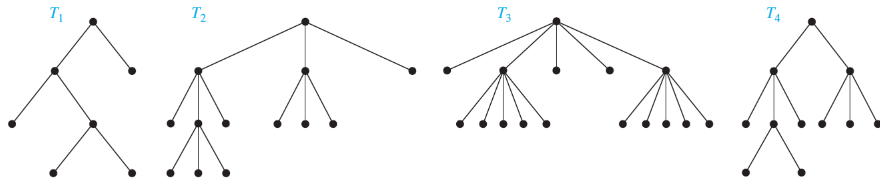
Definition 2.3

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 2.1

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





1. T_1 is a full binary tree because each of its internal vertices has two children.
2. T_2 is a full 3-ary tree because each of its internal vertices has three children.
3. T_3 each internal vertex has five children, so T3 is a full 5-ary tree.
4. 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.
- 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.

Properties of Trees

Theorem 2.2

A tree with n vertices has n -1 edges.

Theorem 2.3

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

Theorem 2.4

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*

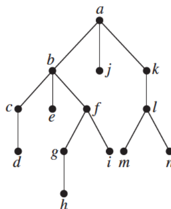
Tree Terminology

Definition 2.4

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

Example 2.2

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



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,

Theorem 2.5

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

Definition 2.5

It is often desirable to use rooted trees that are "balanced" so that the subtrees at each vertex contain paths of approximately the same length.
A rooted m-ary tree of height h is balanced if all leaves are at levels h or h - 1.

Corollary 2.1

If any m-ary tree of hight h has L leaves, then $h \geq \lceil log_m L \rceil$. If the m-ary tree is full and balanced, then $h = \lceil log_m L \rceil$.
(We are using the ceiling function here. Recall that $\lceil x \rceil$ is the smallest integer greater than or equal to x.)

Binary Search Trees

Binary Search Trees

We consider a particular kind of a binary tree called a Binary Search Tree (BST). The basic idea behind this data structure is to have such a storing repository that provides the efficient way of data sorting, searching and retriving.

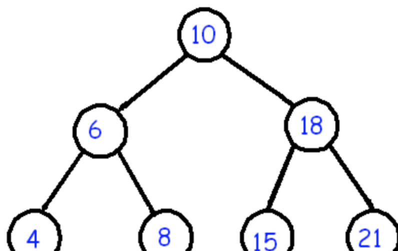
A BST is a binary tree where vertices are ordered in the following way:

1. each vertixe contains one key (also known as data)
2. the keys in the left subtree are less then the key in its parent vertixe, in short $L < P$;
3. the keys in the right subtree are greater the key in its parent vertixe, in short $P < R$;
4. duplicate keys are not allowed.

Binary Search Trees

Example 2.3

In the following tree all vertices in the left subtree of 10 have keys < 10 while all vertices in the right subtree > 10.
Because both the left and right subtrees of a BST are again search trees; the above definition is recursively applied to all internal vertices:



Binary Search Trees

Example 2.4

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

mathematics	physics > mathematics	geography < mathematics	zoology > mathematics zoology > physics
meteorology > mathematics	geology < mathematics	psychology > mathematics psychology > physics	chemistry < mathematics

Decision Trees

Definition 2.6

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

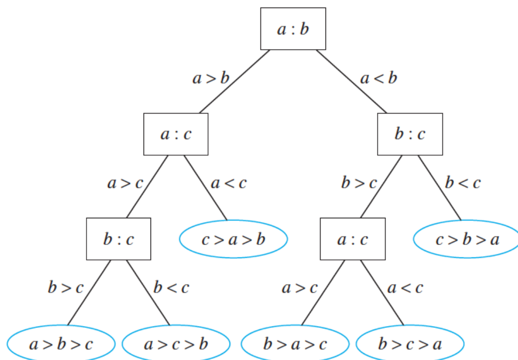
Example 2.5

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

Decision Trees

Example 2.6

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



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.
- 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.

Traversal Algorithms

- Traversal is a process to visit all the vertices of a tree .
- Because, all vertices are connected via edges (links) we always start from the root (head) vertex.
- That is, we cannot randomly access a vertex in a tree.

There are three ways which we use to traverse a tree :

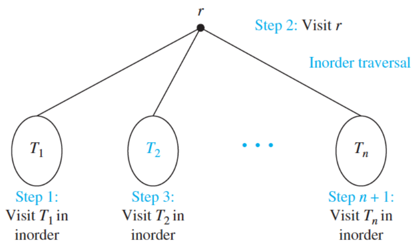
1. In-order Traversal
2. Pre-order Traversal
3. Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree.

In-order Traversal

In-order Traversal

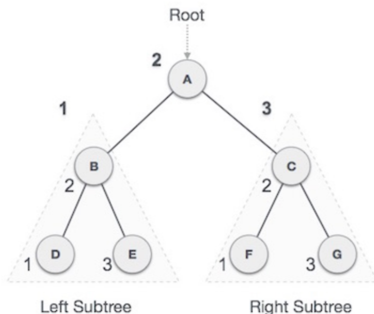
- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every vertex may represent a subtree itself.
- If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.



In-order Traversal(Example)

Example 2.7

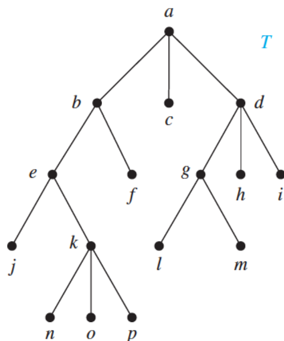
We start from A, and following in-order traversal, we move to its left subtree B. B is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be: $D \to B \to E \to A \to F \to C \to G$.



In-order Traversal(Example)

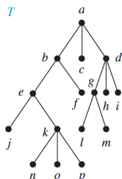
Example 2.8

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

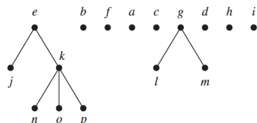


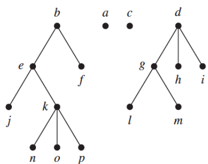
In-order Traversal(Solution)

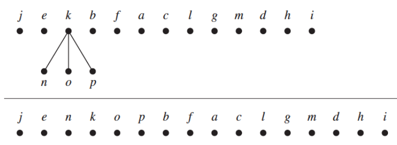
The Inorder Traversal of T .



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



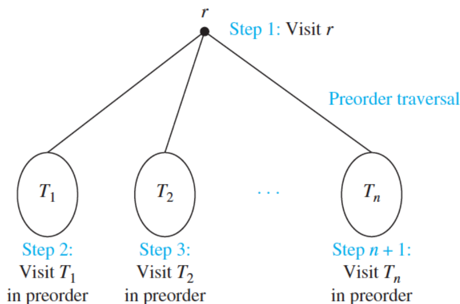




Pre-order Traversal

Pre-order Traversal

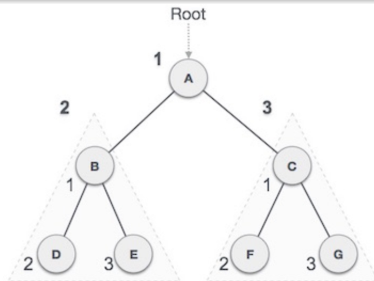
In this traversal method, the root vertex is visited first, then the left subtree and finally the right subtree.



Pre-order Traversal(Example)

Example 2.9

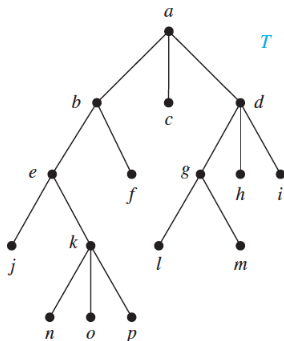
We start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B. B is also traversed pre-order. The process goes on until all the vertices are visited. The output of pre-order traversal of this tree will be:
$A \to B \to D \to E \to C \to F \to G.$



Pre-order Traversal(Example)

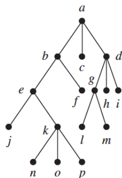
Example 2.10

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

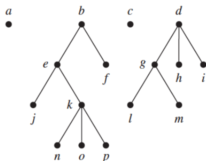


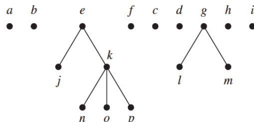
Pre-order Traversal(Solution)

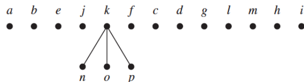
The Preorder Traversal of T .

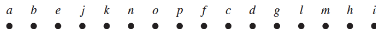


Preorder traversal: Visit root, visit subtrees left to right





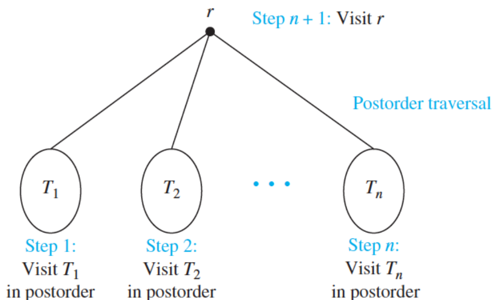




Post-order Traversal

Post-order Traversal

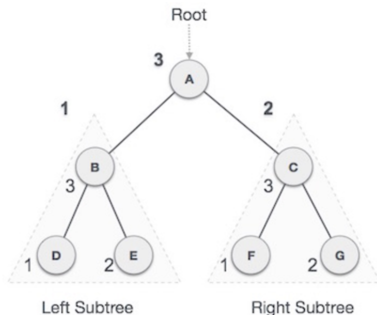
In this traversal method, First we traverse the left subtree, then the right subtree and finally the root vertex.



Post-order Traversal(Example)

Example 2.11

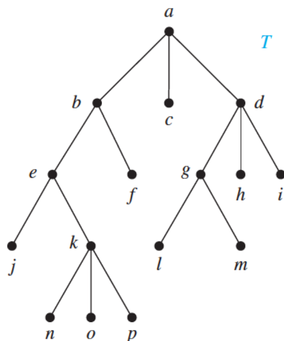
We start from A, and following Post-order traversal, we first visit the left subtree B. B is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be: $D \to E \to B \to F \to G \to C \to A$.



Post-order Traversal(Example)

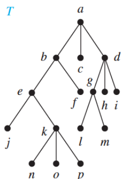
Example 2.12

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

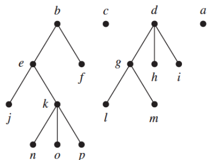


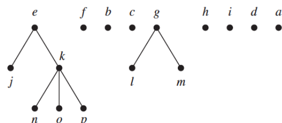
Post-order Traversal(Solution)

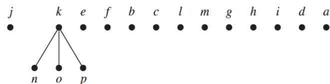
The Postorder Traversal of T .

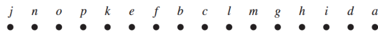


Postorder traversal: Visit subtrees left to right; visit root









Infix, Prefix, and Postfix Notation

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

Infix, Prefix, and Postfix Notation

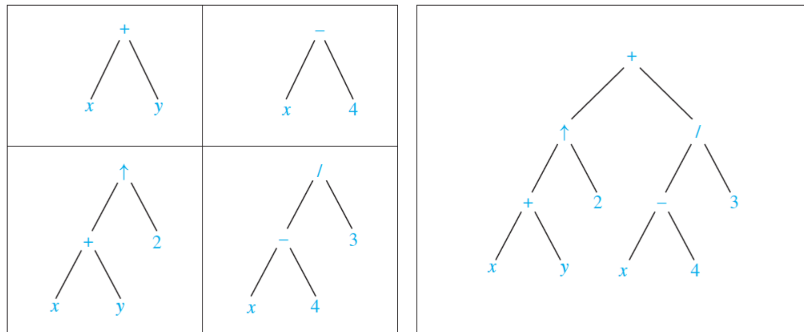
Example 2.13

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 $x-4$ is constructed, and then this is incorporated into a subtree representing $(x-4)/3$. 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

Infix, Prefix, and Postfix Notation



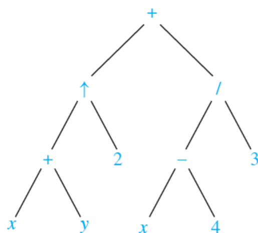
A Binary Tree Representing $((x + y) \uparrow 2) + ((x - 4)/3)$

Infix, Prefix, and Postfix Notation

- 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 2.14

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



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

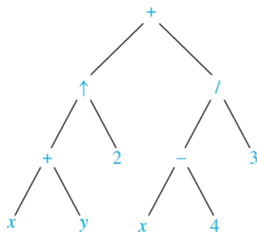
Solution

We obtain the prefix form for this expression by traversing the binary tree that represents it in preorder. This produces

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

Example 2.15

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



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

Solution

The postfix form of the expression is obtained by carrying out a postorder traversal of the binary tree for this expression. This produces the postfix expression:

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

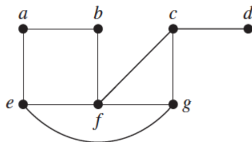
Introduction

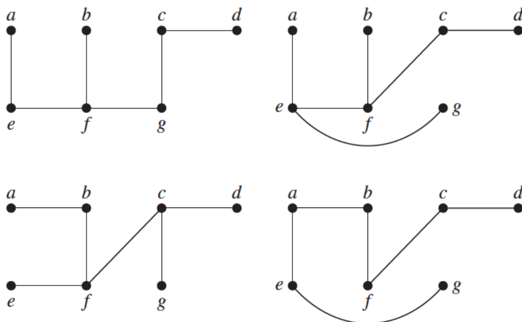
Definition 2.7

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 2.16

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





Spanning Trees of G.

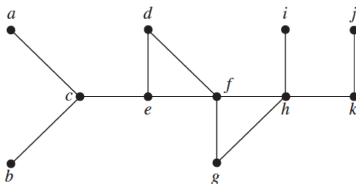
Theorem 2.6

A simple graph is connected if and only if it has a spanning tree.

There are two important techniques when it comes to visiting each vertex in a tree: a **depth first search** and **breadth first search**. We can build a spanning tree for a connected simple graph using depth-first search.

Example 2.17

Use depth-first search to find a spanning tree for the graph G shown in Figure.



The Graph *G*.

Depth-First Search

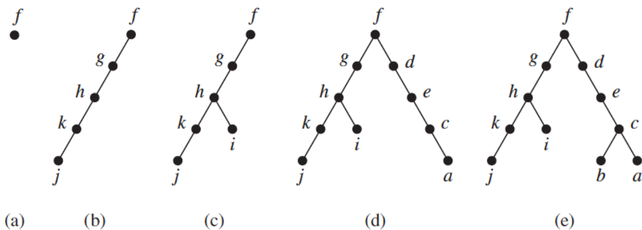
a- We arbitrarily start with the vertex f.

b- A path is built by successively adding edges incident with vertices not already in the path, as long as this is possible. This produces a path f , g, h, k, j.

c- Next, backtrack to k. There is no path beginning at k containing vertices not already visited.

d- So we backtrack to h. Form the path h, i. Then backtrack to h, and then to f .

e- From f build the path f , d, e, c, a. Then backtrack to c and form the path c, b. This produces the spanning tree.



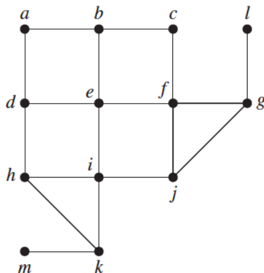
Depth-First Search of G.

Breadth-First Search

We can also produce a spanning tree of a simple graph by the use of breadth-first search.

Example 2.18

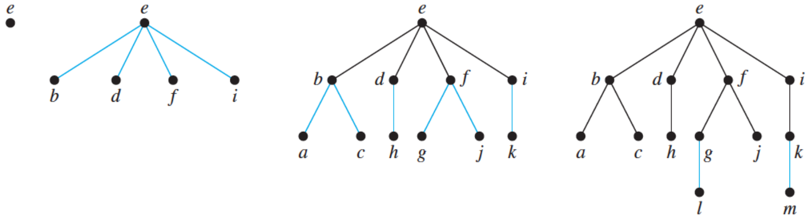
Use breadth-first search to find a spanning tree for the graph shown in Figure.



Breadth-First Search

1. We choose the vertex e to be the root.
2. Then we add edges incident with all vertices adjacent to e, so edges from e to b, d, f , and i are added. These four vertices are at level 1 in the tree.
3. Then add the edges from these vertices at level 1 to adjacent vertices not already in the tree. The new vertices a, c, h, j , g, and k are at level 2.
4. Next, add edges from these vertices to adjacent vertices not already in the graph.

Breadth-First Search



Breadth-First Search of G.