TREES

DISCRETE STRUCTURE AND APPLICATIONS

CONTENT

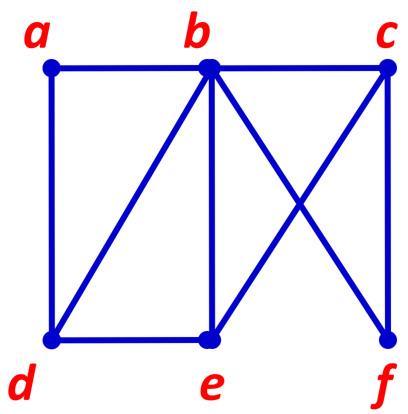
4.1 INTRODUCTION TO TREES

- Define and recognize a tree
- Define and recognize a rooted tree
- Define and recognize a *m*-ary tree

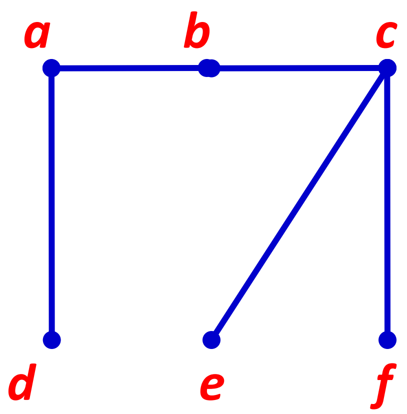
A Tree

- A **tree** is a connected undirected graph with no simple circuit.
- An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.
- THEOREM: A tree with n vertices has *n* – 1 edges.

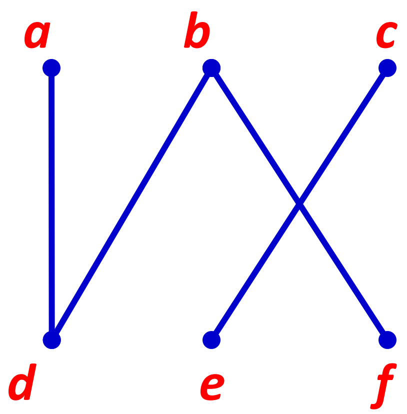
EXAMPLES



NOT A TREE



A TREE



NOT A TREE

EXERCISE 4.1

1. Which of the following graphs are trees?

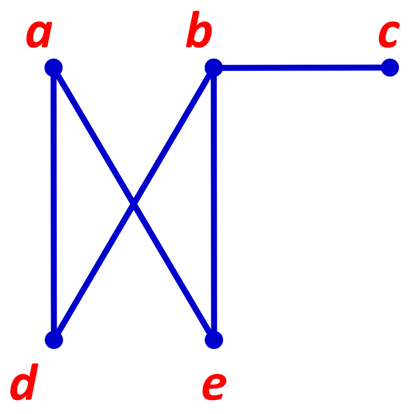


FIGURE 1

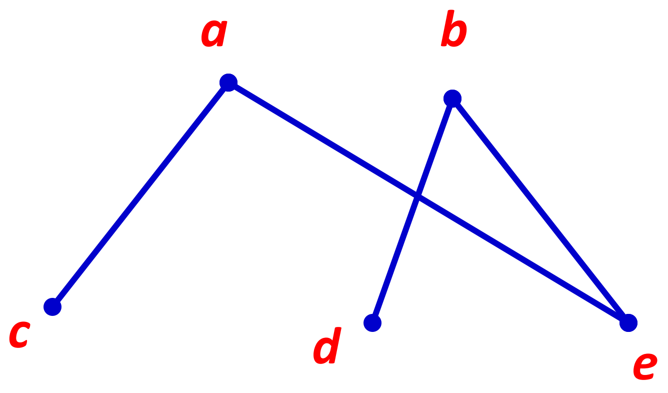


FIGURE 2

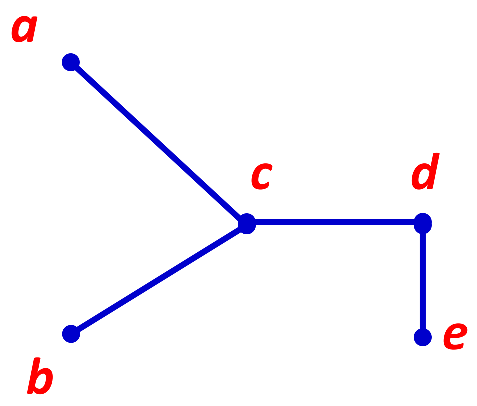


FIGURE 3

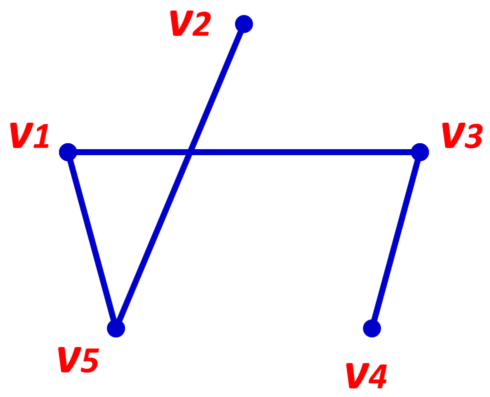
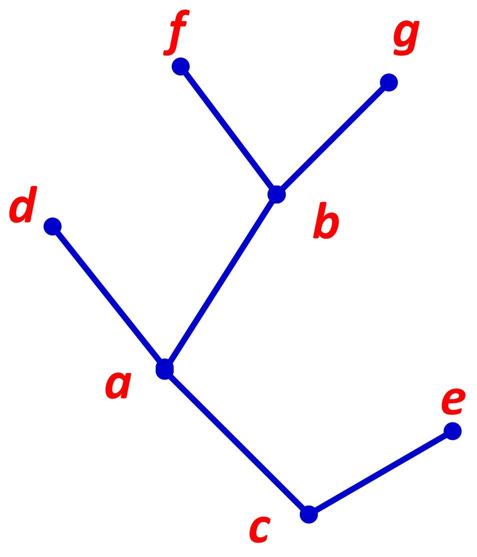


FIGURE 4

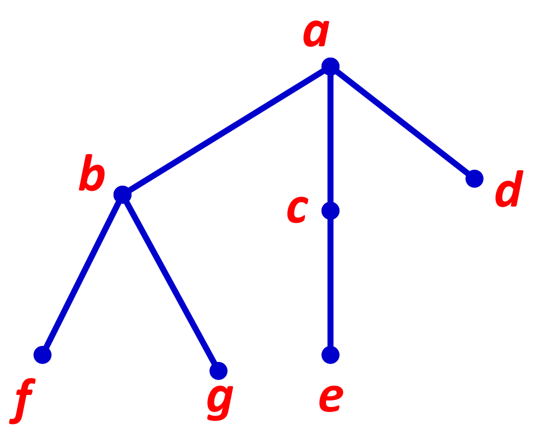
A Rooted Tree

- 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.
- Different choice of root produce different rooted tree

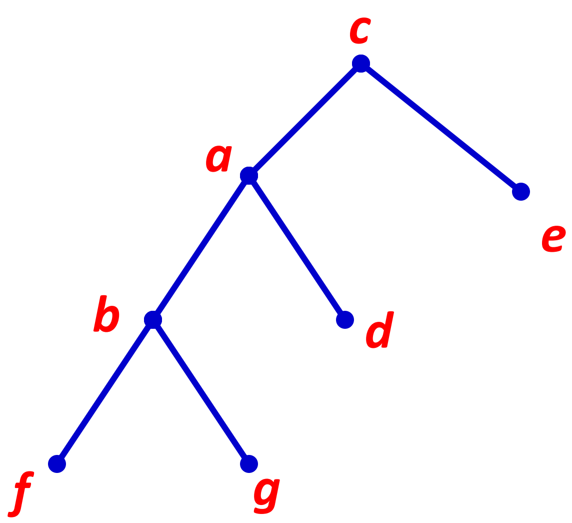
EXAMPLES



A TREE



A TREE with root *a*

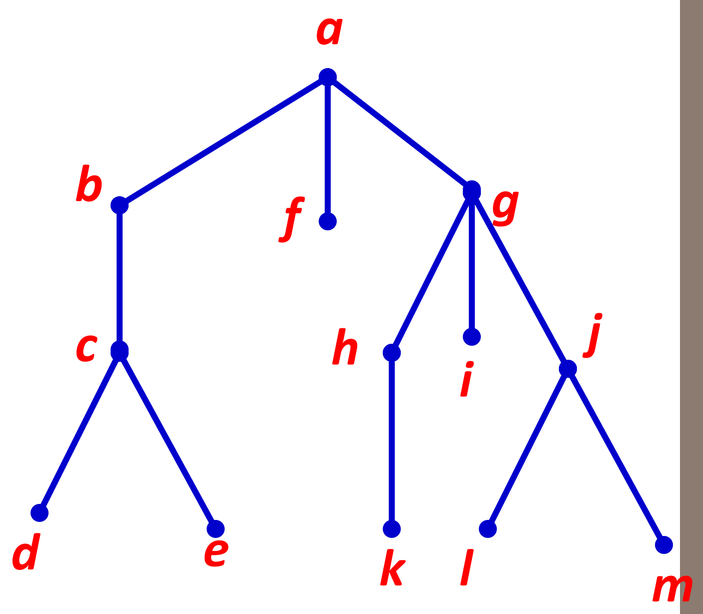


A TREE with root *c*

Properties of Rooted Trees

- Parent – A vertex is a parent if it has one or more children
    - ➢ The parent of *c* is *b*
- Children – If *A* is a vertex with successors *B* and *C*, then *B* and *C* are the children of *A*.
    - ➢ The children of *a* is *b*, *f* and *g*
- Siblings – Children with the same parent vertex.
    - ➢ *h*, *i* and *j* are siblings
- Level – the length of the unique path from the root to a vertex
    - ➢ Vertex *a* is at level 0
    - ➢ Vertices *d* and *e* is at level 3

EXAMPLES

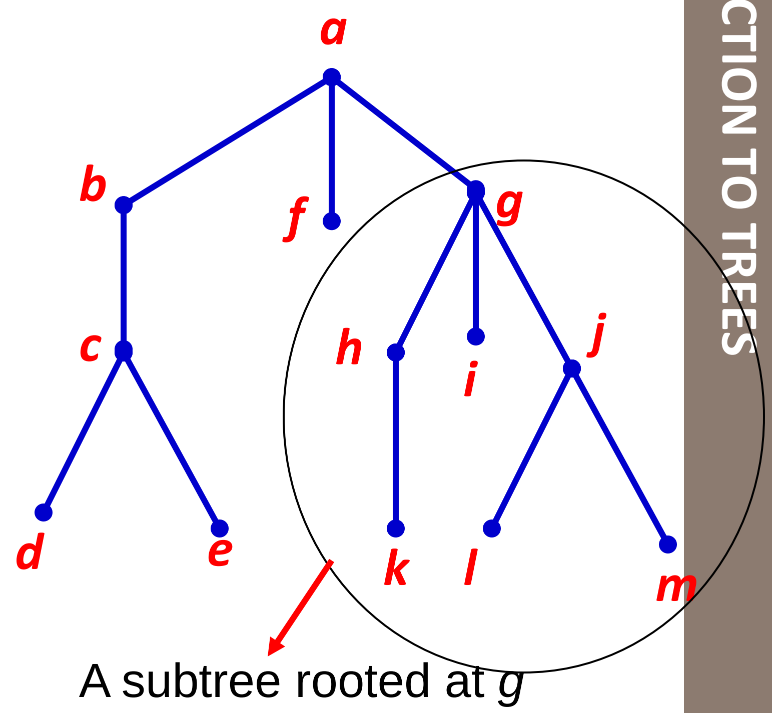


- Height – The maximum level of all the vertices
    - ➢ The height of this tree is 3.

Properties of Rooted Trees

- Ancestor of a vertex (v) – the vertices in the path from the root to this vertex excluding this vertex.
  - The ancestors of *e* are *c*, *b* and *a*
- Descendent of a vertex (v) – vertices that have *v* as ancestor.
  - The descendants of *b* are *c*, *d* and *e*
- Leaf – A vertex with no children
  - The leaves are *d*, *e*, *f*, *i*, *k*, *l* and *m*
- Internal Vertices – vertices that have children
  - The internal vertices are *a*, *b*, *c*, *g*, *h* and *j*

EXAMPLES



- Subtree – A subgraph of the tree consisting of a root and its descendent and all edges incident to these descendent.

EXERCISE 4.1

2. Answer these questions about the rooted tree illustrated.

a) Which vertex is the root?

b) Which vertices are internal?

c) Which vertices are leaves?

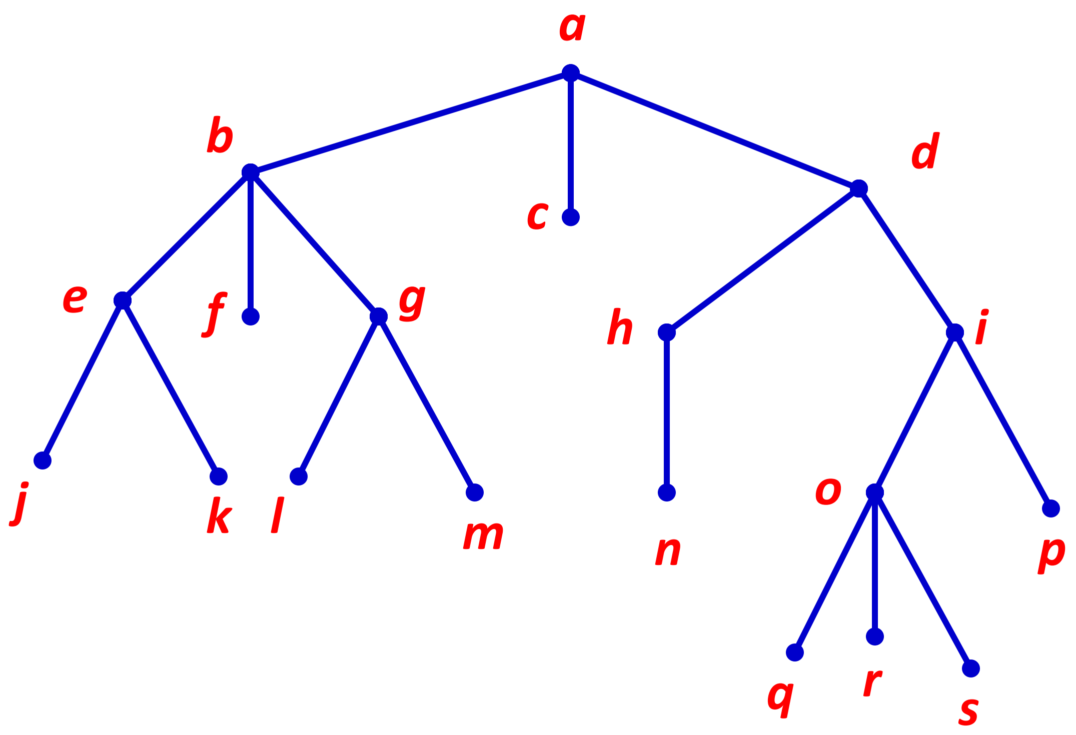
d) Which vertices are children of *g*?

e) Which vertex is the parent of *o*?

f) Which vertices are siblings of *e*?

g) Which vertices are ancestors of *m*?

h) Which vertices are descendants of *d*?



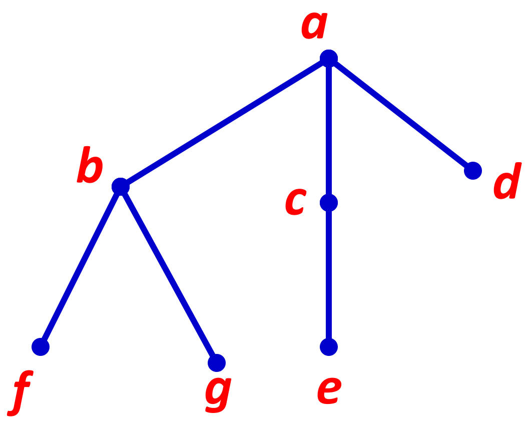
e) Find the level of each vertex.

f) What is the height of this tree?

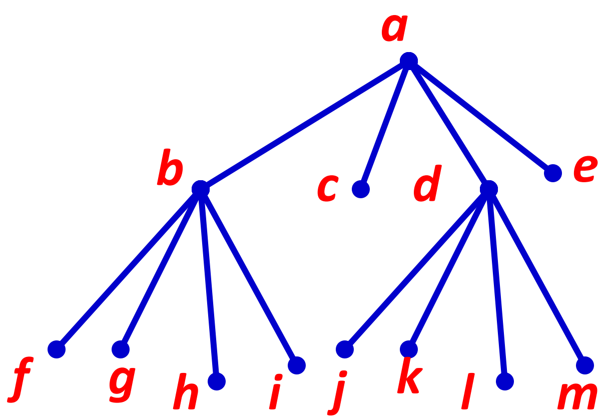
m-ary Tree

- A rooted tree is called an *m-ary tree* if every vertex has no more than *m* children.
- The tree is called a full *m*-ary tree if every internal vertex has exactly *m* children.
- A rooted *m*-ary tree is balanced if all leaves are at levels *h* or *h*-1.

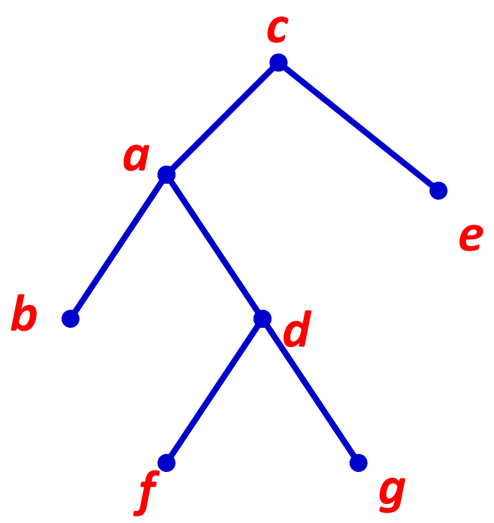
EXAMPLES



A 3-ary TREE



A full 4-ary TREE



A full binary TREE

4.2 APPLICATION OF TREES

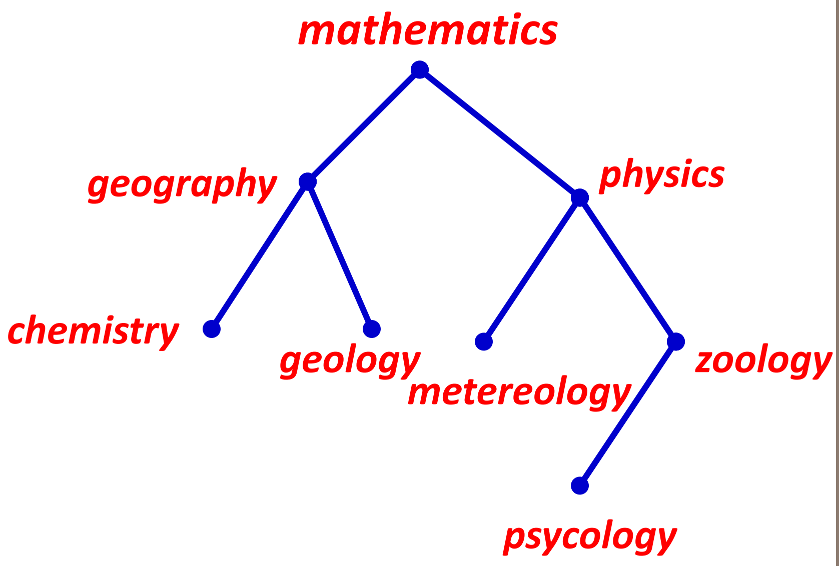
- Introduce Binary Search Trees
- Introduce Decision Trees
- Introduce Game Trees

Binary Search Trees

- 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
- Each vertex is labeled with a key
- 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

Binary search tree for the words *mathematics, physics, geography, zoology, meteorology, geology, psychology,* and *chemistry* using **alphabetical order**



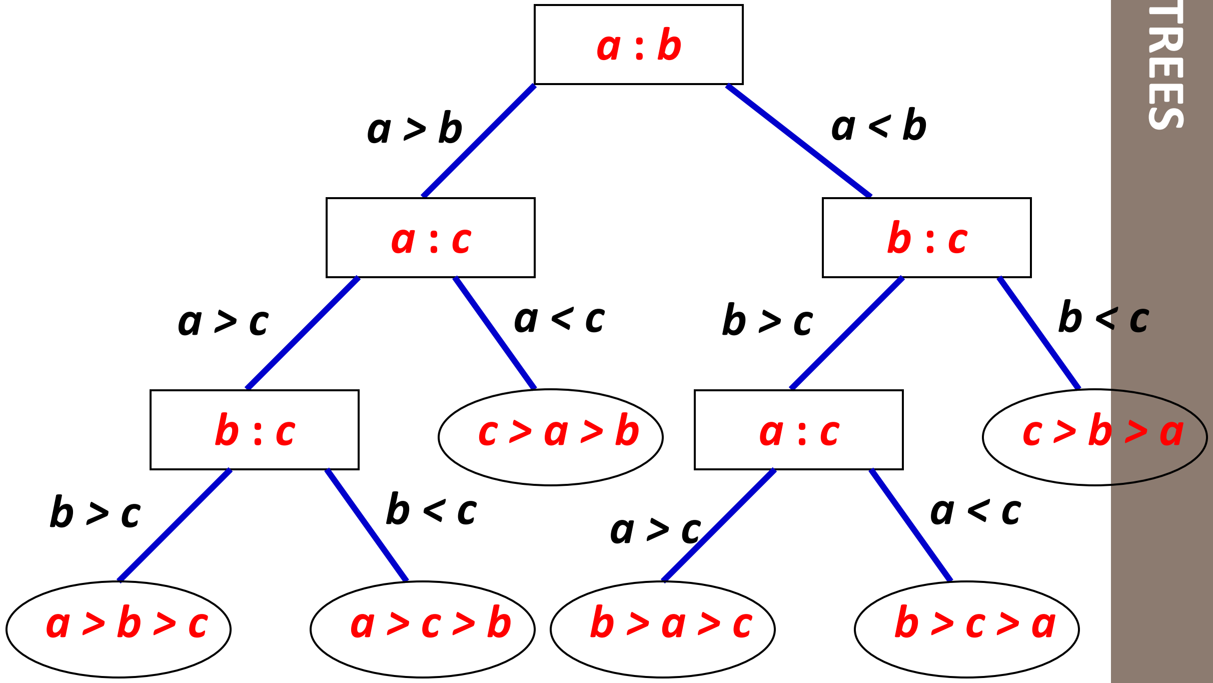
Decision Trees

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

- The possible solutions of the problem correspond to the paths to the leaves of this rooted tree.

EXAMPLE

A Decision tree that orders the elements of the list *a*, *b*, *c*

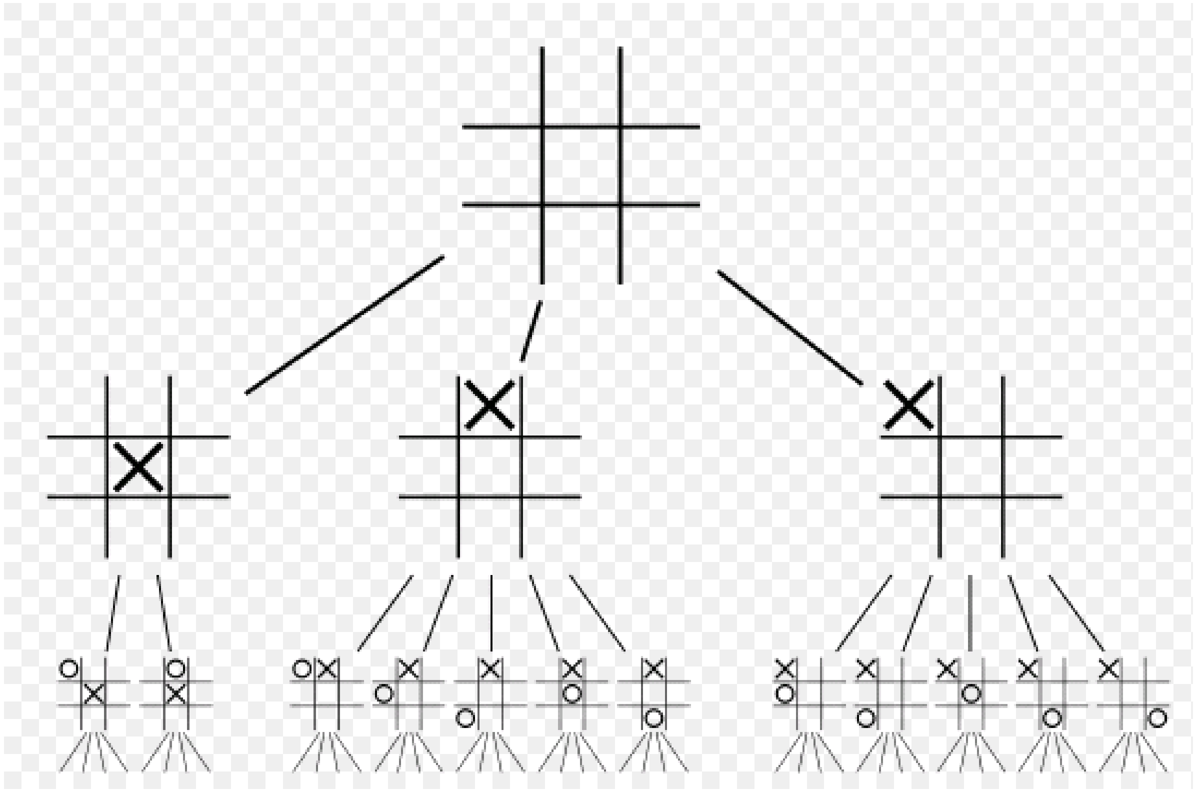


Game Trees

- Use trees to analyze certain types of games
- Vertices represent the positions can be in as it progresses
- Edges represent legal moves between this position
- Games tree is infinite if the games they represent never end

EXAMPLE

A Game tree that represents the first two level of the tic-tac-toe game.



4.3 TREE TRAVERSAL

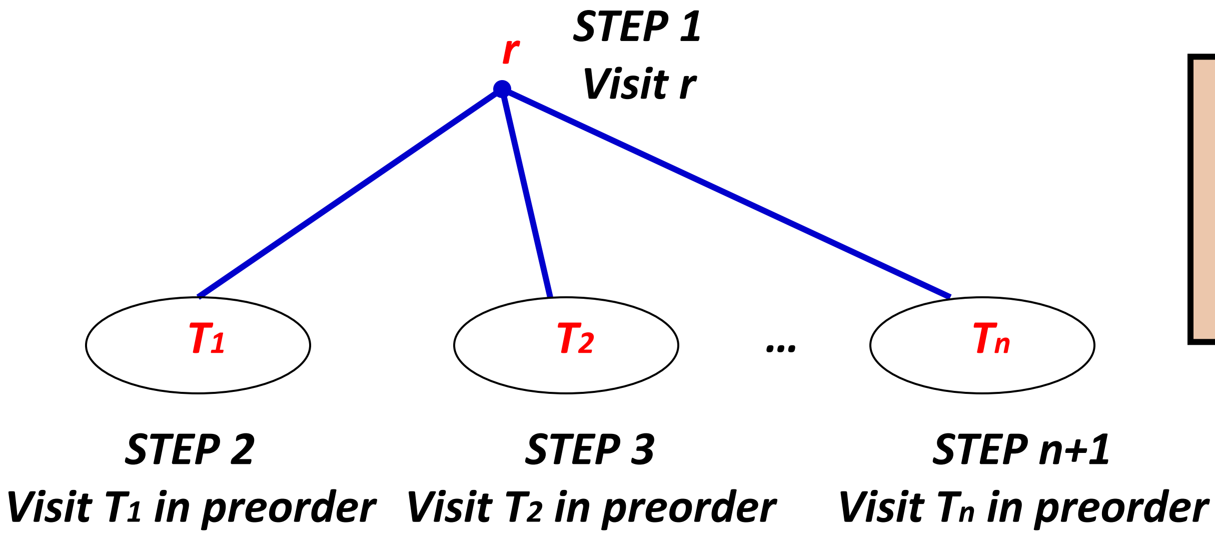
- Determine preorder traversal, inorder traversal and postorder traversal of an ordered rooted tree.
- Determine the infix, prefix, and postfix form of an expression.

Tree Traversal

- Ordered trees are often used to restore data/info.
- Tree traversal is a procedure for systematically visiting each vertex of an ordered rooted tree to access data.
- If the tree is label by Universal Address System we can totally order the vertices using lexicographic ordering
  - Example: 0 < 1 < 1.1 < 1.2 < 1.2.1 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 4 < 4.1
- Tree traversal algorithm
  - Preorder, inorder and postorder traversal

Preorder Traversal

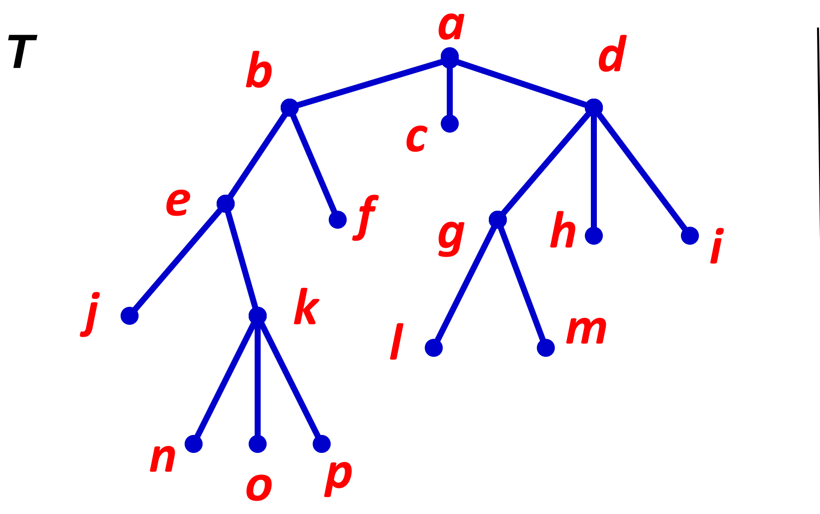
- Let T be an ordered rooted tree with root r.
  - If T consists only of r, then r is the *preorder* traversal of T.
  - If $T_1, T_2, \ldots, T_n$ are subtrees at r from left to right in T, then the preorder traversal begins by visiting r, continues by traversing T_1 in preorder, then T_2 in preorder, and so on until T_n is traversed in preorder.

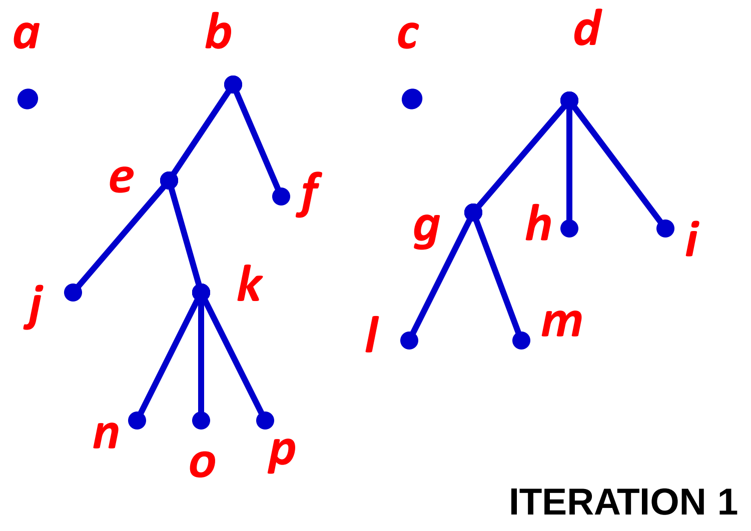


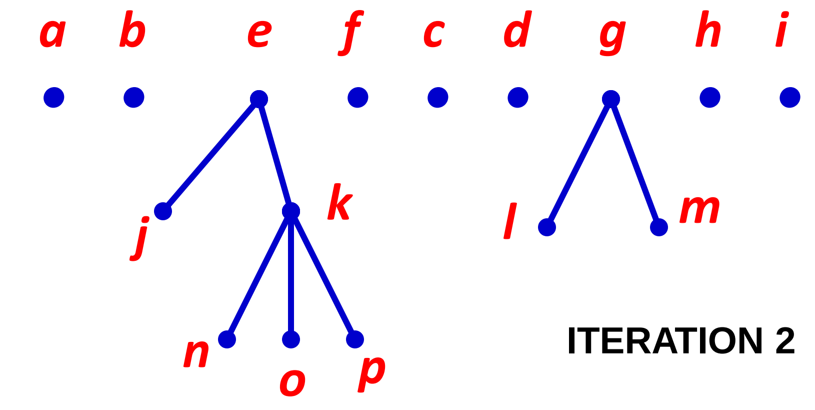
TIPS

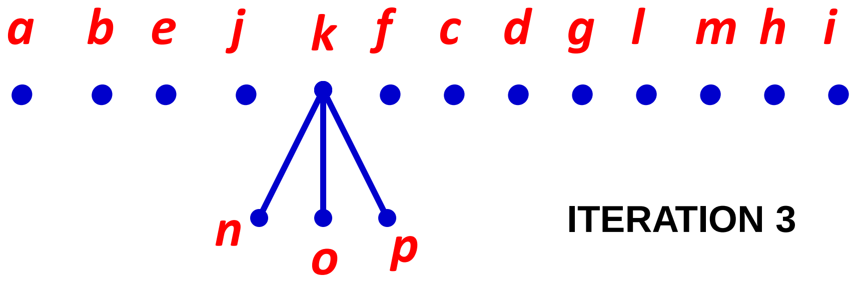
Preorder Traversal: Visit root, visit subtrees left to right

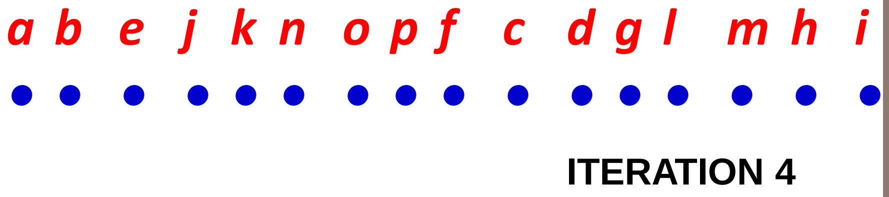
EXAMPLE: Preorder Traversal







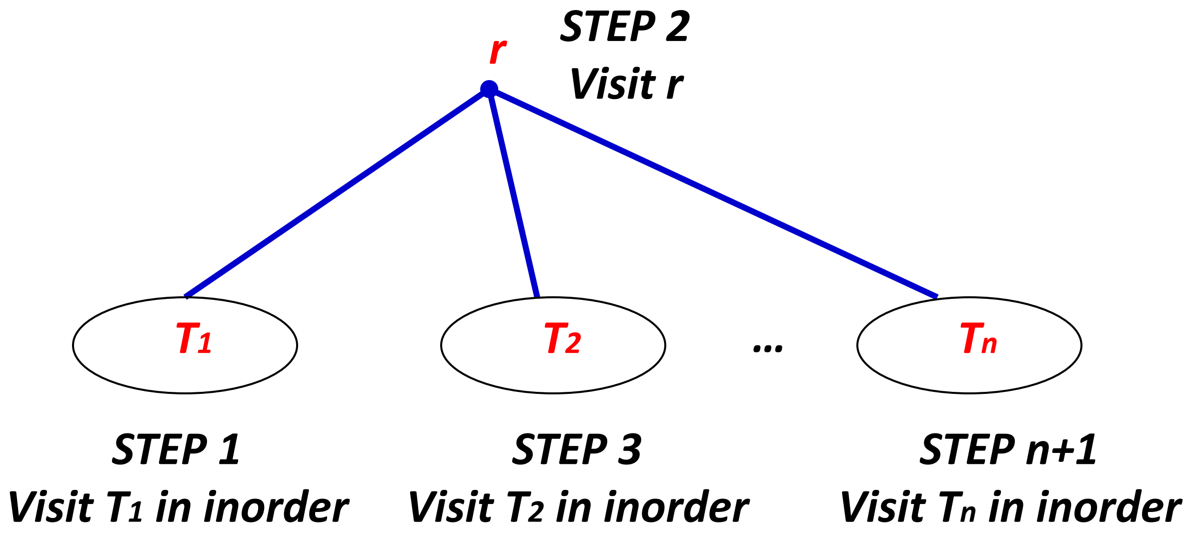




The preorder traversal of *T*

Inorder Traversal

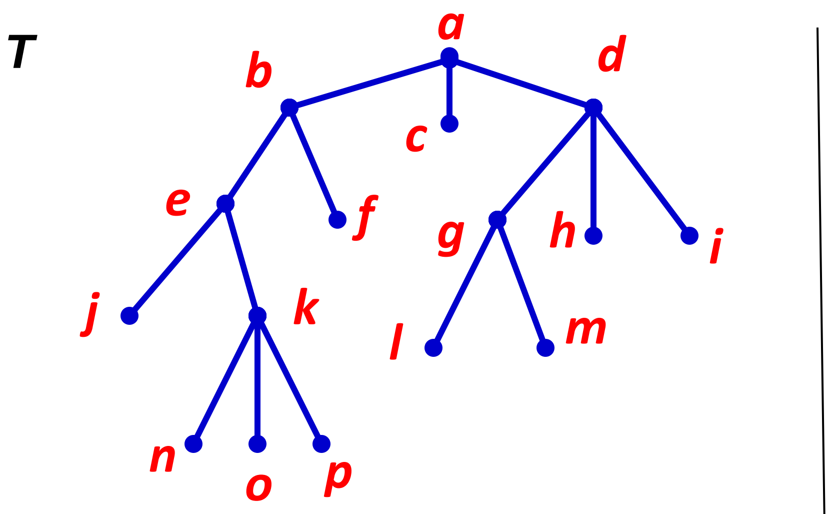
- Let *T* be an ordered rooted tree with root *r*.
  - If *T* consists only of *r*, then *r* is the *inorder* traversal of *T*.
  - If $T_1, T_2, \ldots, T_n$ are subtrees at *r* from left to right in *T*, then the inorder traversal begins by traversing T_1 in inorder, then visiting *r*, continues by traversing T_2 in inorder, and so on until T_n is traversed in inorder.

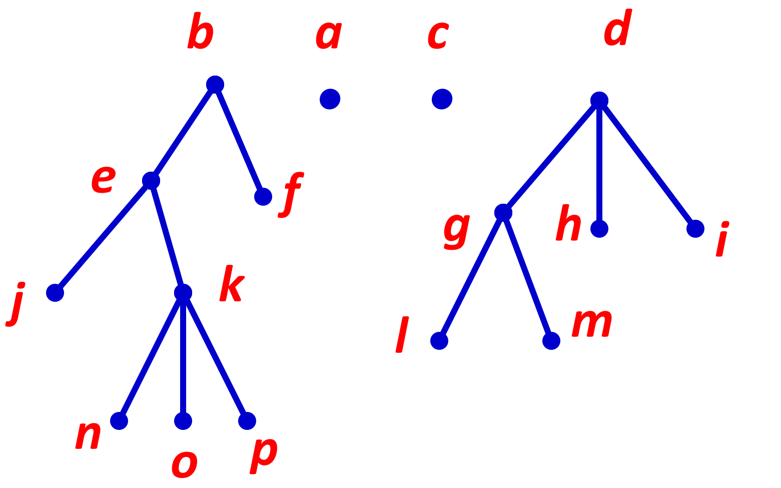


TIPS

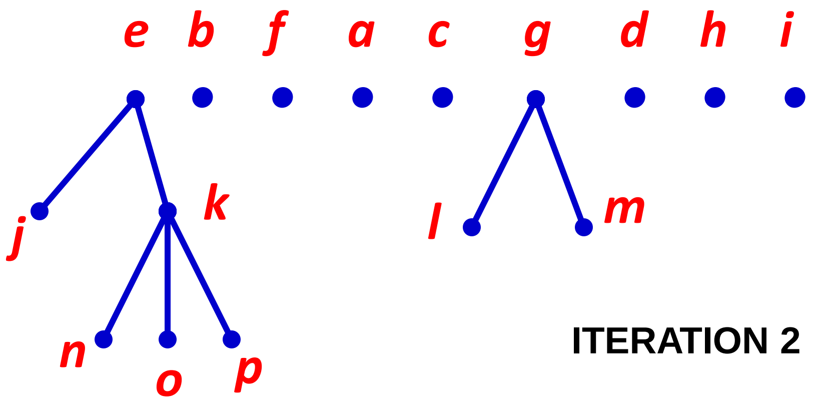
Inorder Traversal: Visit leftmost subtree, Visit root, Visit other subtrees left to right.

EXAMPLE: Inorder Traversal

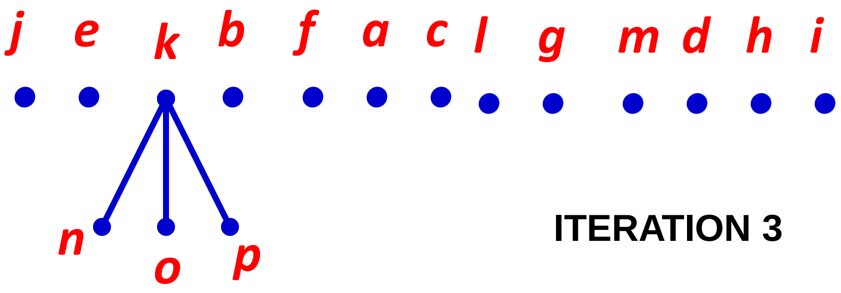




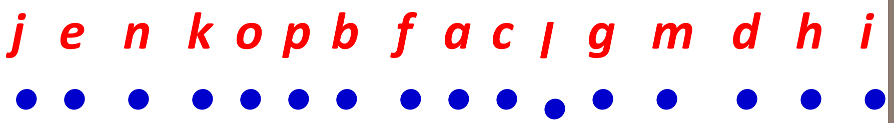
ITERATION 1



ITERATION 2



ITERATION 3

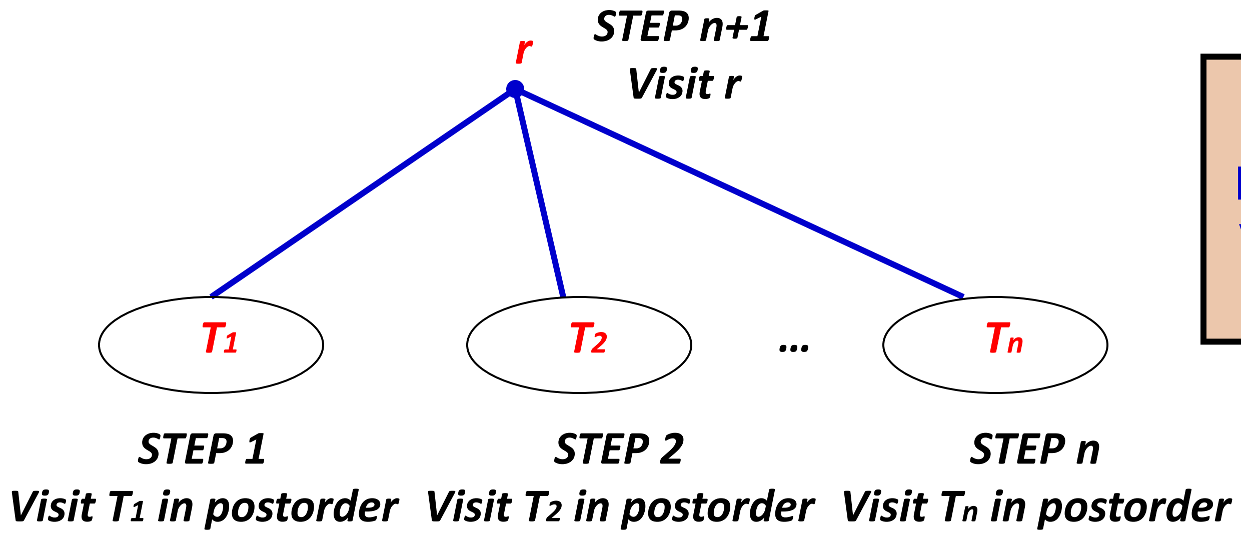


ITERATION 4

The inorder traversal of *T*

Postorder Traversal

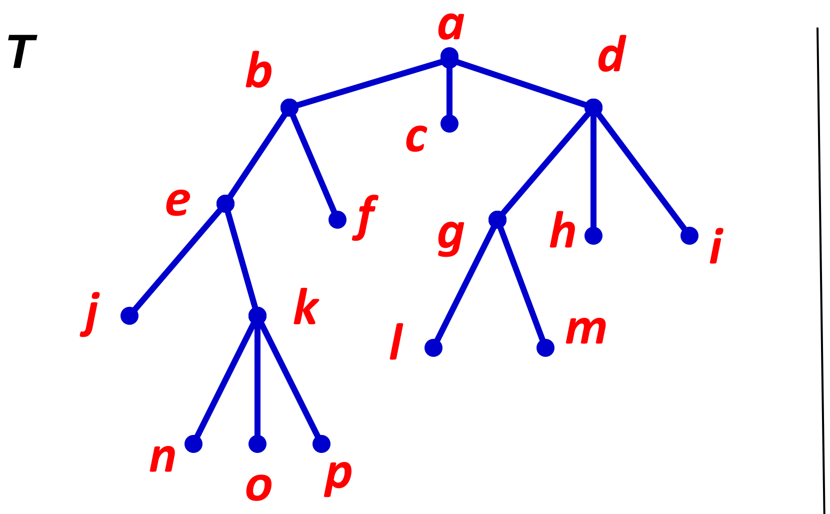
- Let *T* be an ordered rooted tree with root *r*.
  - If *T* consists only of *r*, then *r* is the *postorder* traversal of *T*.
  - If T_1, T_2, …, T_n are subtrees at *r* from left to right in *T*, then the preorder traversal begins by traversing T_1 in postorder, then T_2 in postorder, and so on until T_n is traversed in postorder and ends by visiting *r*.

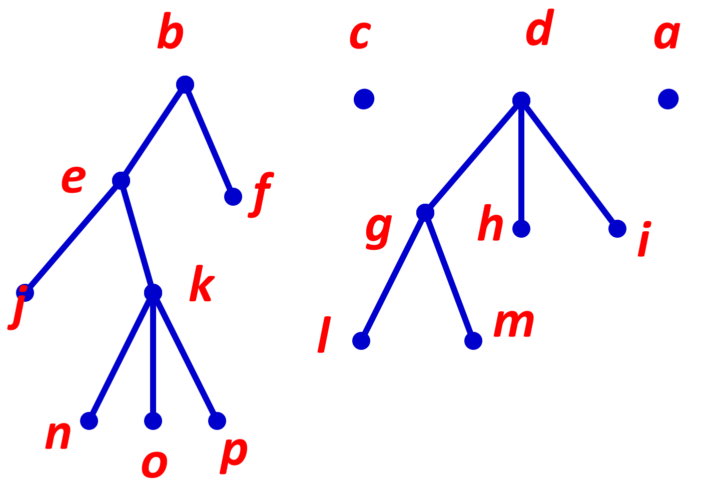


TIPS

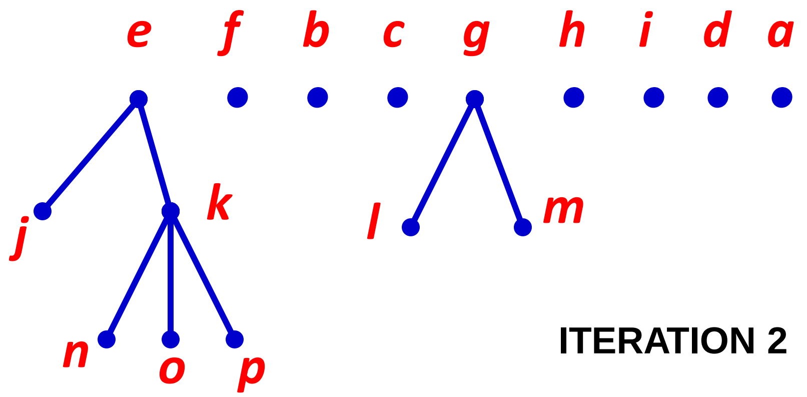
Postorder Traversal: Visit subtrees left to right, Visit root.

EXAMPLE: Postorder Traversal

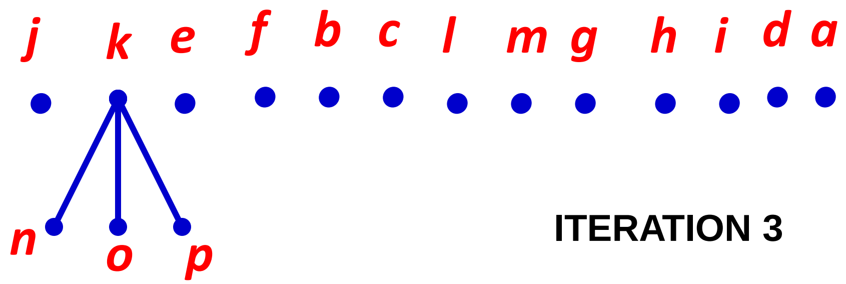




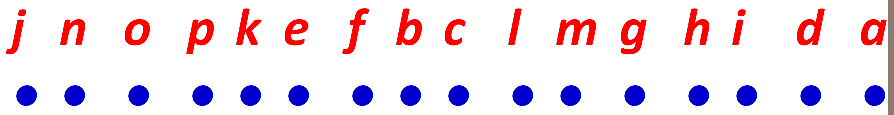
ITERATION 1



ITERATION 2



ITERATION 3



ITERATION 4

The preorder traversal of *T*

EXERCISE 4.3

1. Determine the order in which a preorder, inorder, and postorder traversal visits the vertices of the following rooted tree.

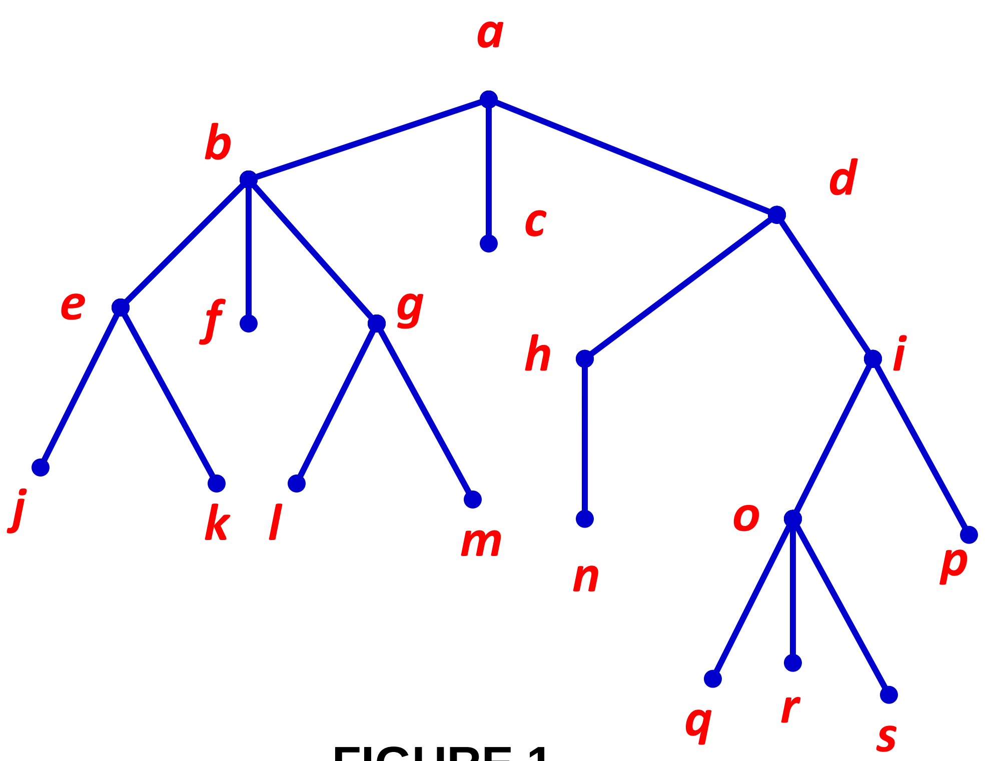


FIGURE 1

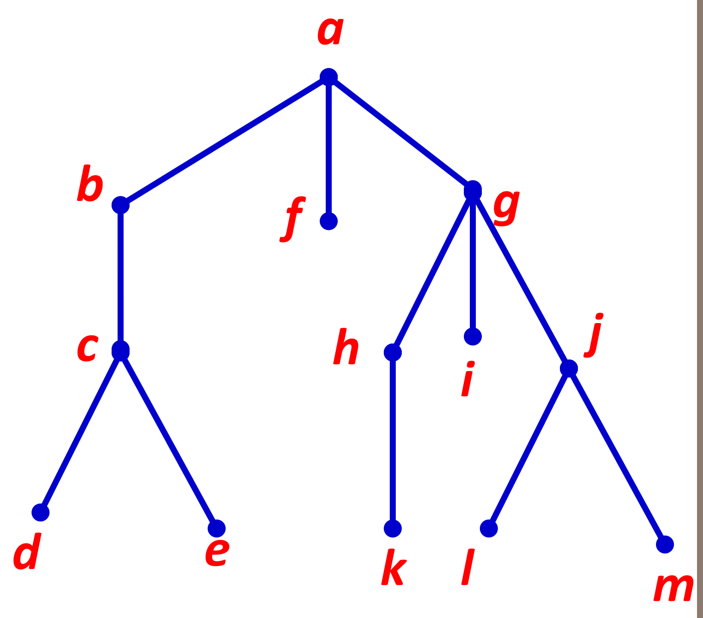
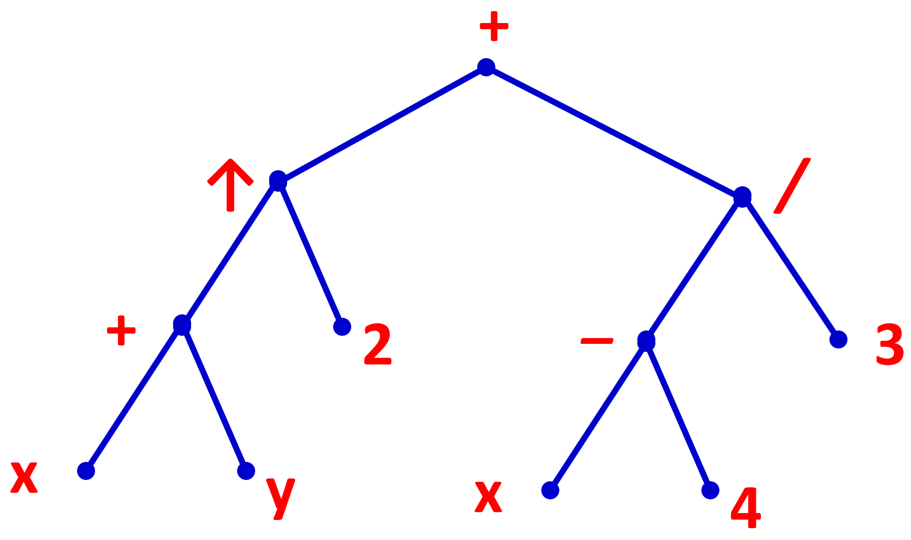


FIGURE 2

Represent Expression by Rooted Tree

- We can represent complicated expression (propositions, sets, arithmetic) using ordered rooted trees.
- **EXAMPLE**: A binary tree representing ((x+y)↑2)+((x-4)/3)



Infix, Prefix & Postfix Notation

- We obtain the *Infix form* of an expression when we traverse its rooted tree in Inorder.
  - The infix form for expression $((x+y)\uparrow 2)+((x-4)/3)$ is $x + y \uparrow 2 + x – 4 / 3$ or $((x+y)\uparrow 2)+((x-4)/3)$

- We obtain the *Prefix form* of an expression when we traverse its rooted tree in Preorder.
  - The prefix form for expression $((x+y)\uparrow 2)+((x-4)/3)$ is $+ \uparrow + x\ y\ 2\ /\ - x\ 4\ 3$

- We obtain the *Postfix form* of an expression when we traverse its rooted tree in Postorder.
  - The postfix form for expression $((x+y)\uparrow 2)+((x-4)/3)$ is $x\ y + 2 \uparrow x\ 4 – 3 / +$

Evaluating Prefix Expression

- Working right to left and performing operations using the operands on the right.
- **EXAMPLE:**

  The value of the prefix expression + - * 2 3 5 / ↑ 2 3 4 is 3

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

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

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

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

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

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

Evaluating Postfix Expression

- Working left to right and performing operations using the operands on the left.
- **EXAMPLE:**

  The value of the postfix expression 7 2 3 * - 4 ↑ 9 3 / + is 4

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

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

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

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

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

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

EXERCISE 4.3

2. Represent the following expression using binary trees. Then write these expression in infix, prefix and postfix notations.
   a. $((x+2)\uparrow 3)*(y-(3+x))-5$
   b. $(A\cap B)-(A\cup(B-A))$
3. What is the value of these expression in prefix expression?
   a. $+ - \uparrow 3\ 2 \uparrow 2\ 3 / 6 - 4\ 2$
   b. $* + 3 + 3 \uparrow 3 + 3\ 3\ 3$
4. What is the value of these expression in postfix expression?
   a. $3\ 2 * 2 \uparrow 5\ 3 - 8\ 4 / * -$
   b. $9\ 3 / 5 + 7\ 2 - *$

EXERCISE 4.3 : EXTRA

PAGE : 722, 723 and 724

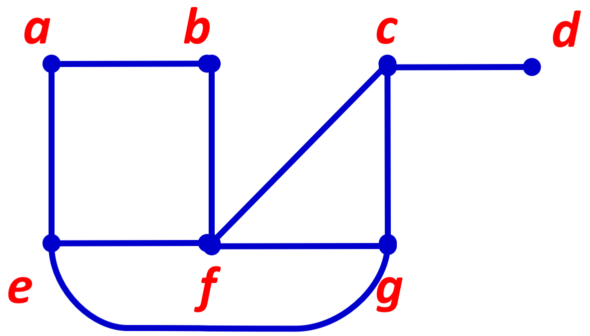
Rosen K.H., *Discrete Mathematics & Its Applications*, (Seventh Edition), McGraw-Hill, 2007.

4.5 SPANNING TREES

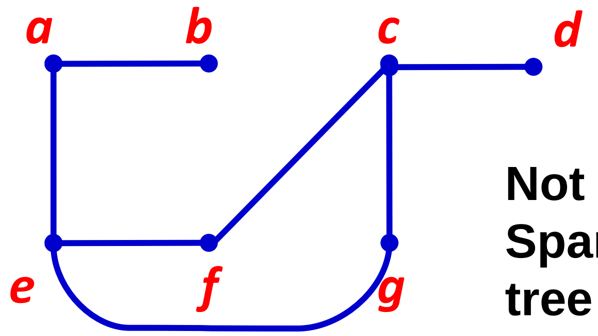
- Find spanning trees of a simple graph

Spanning Trees

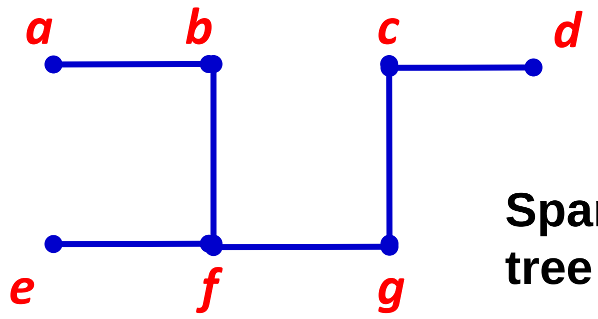
- Let *G* be a simple graph. A *spanning tree* of *G* is a subgraph of *G* that is a tree containing every vertex of *G*.
- A simple graph is connected if and only if it has a spanning tree.
- Applied in IP multitasking.



A simple graph



Not Spanning tree



Spanning tree

EXERCISE 4.5

1. Find a spanning tree for the following graphs.

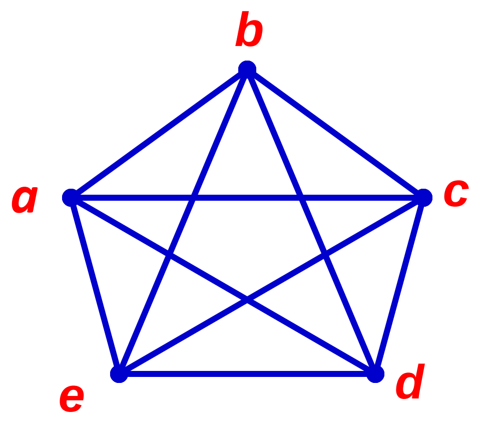


FIGURE 1

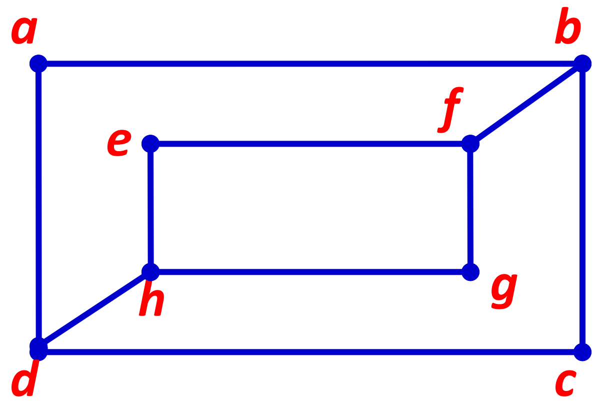


FIGURE 2

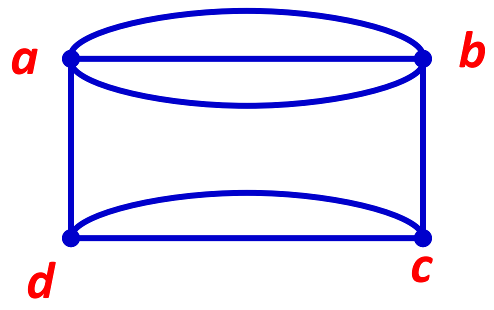


FIGURE 3

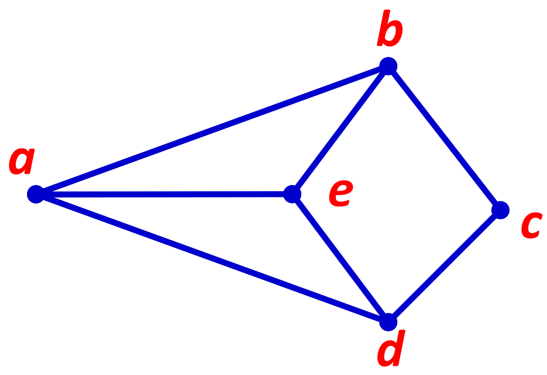


FIGURE 4

EXERCISE 4.5 : EXTRA

PAGE : 734, 735, 736 and 737

Rosen K.H., *Discrete Mathematics & Its Applications*, (Seventh Edition), McGraw-Hill, 2007.

4.6 MINIMUM SPANNING TREES

- Find minimum spanning tree using Prim’s algorithm
- Find minimum spanning tree using Kruskal’s algorithm

Minimum Spanning Trees

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

- Two algorithms can be used:
  - Prim’s Algorithm
    - Robert Prim, 1957
  - Kruskal’s Algorithm
    - Joseph Kruskal, 1956

Prim’s Algorithm

1. Chose an edge with the least weight.
2. Include it in spanning tree, *T.*
3. Select an edge of least weight that is incident with a vertex of an edge in *T.*
4. If it does not create a cycle (simple circuit) with the edges in *T*, then include it in *T*; otherwise discard it.
5. Repeat STEPS 3 and 4 until *T* contains *n*-1 edges.

- There may be more than one minimum spanning tree for a given connected weighted simple graph.
- If there are two edges with similar smallest weight, chose either one.

EXAMPLE: Prim's Algorithm

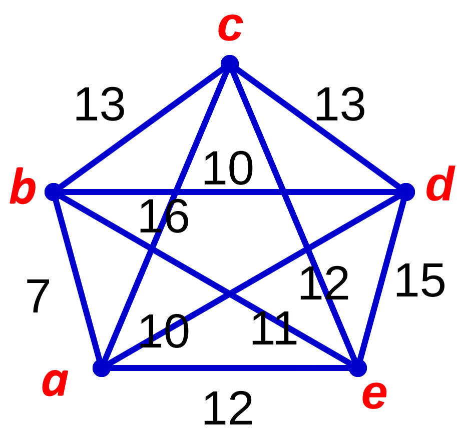
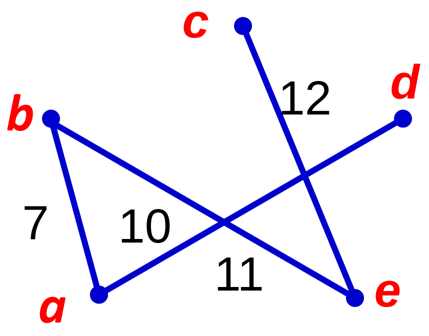
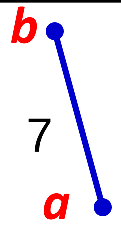
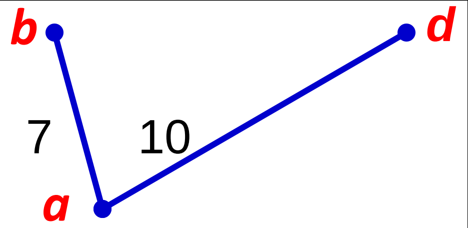
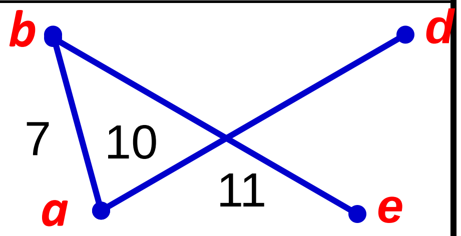
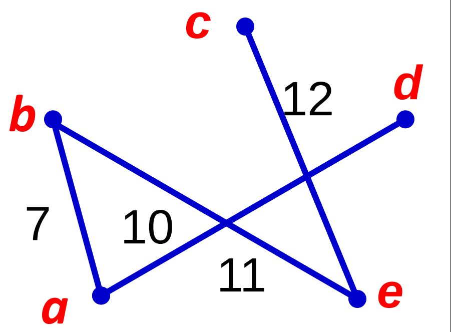


FIGURE 1

The minimum spanning tree is given by:



Total weight = 40

CHOICE	EDGE	WEIGHT	SPANNING TREE
1	{*a*, *b*}	7	
2	{*a*, *d*}	10	
3	{*b*, e}	11	
4	{*e*, *c*}	12	

Kruskal's Algorithm

1. Arrange the edges in *G* in increasing order.
2. Chose an edge with the minimum weight.
3. Include it in spanning tree, *T.*
4. Add an edge of least weight to *T.*
5. If it does not create a cycle (simple circuit) with the edges in *T*, then include it in *T*; otherwise discard it.
6. Repeat STEPS 4 and 5 until *T* contains *n*-1 edges.

- There may be more than one minimum spanning tree for a given connected weighted simple graph.
- If there are two edges with similar smallest weight, chose either one.

EXAMPLE: Kruskal's Algorithm

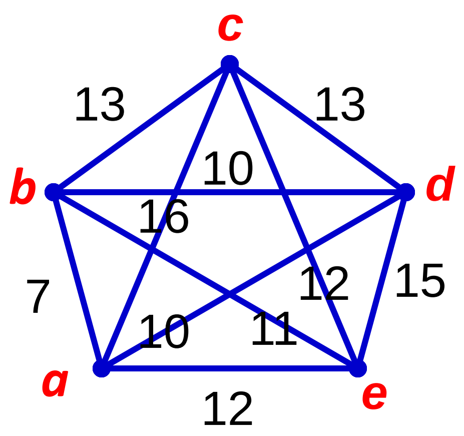
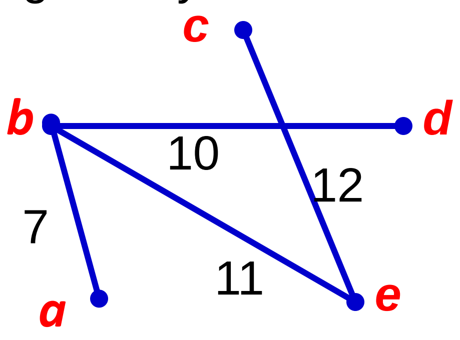
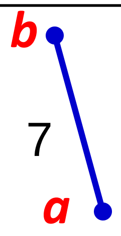
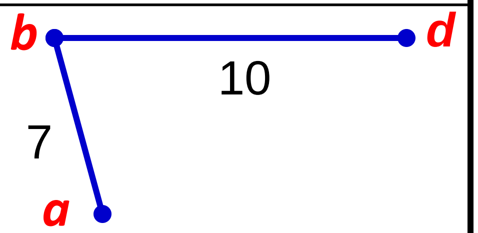
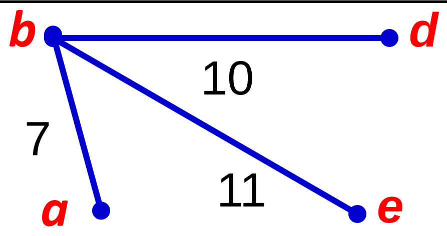
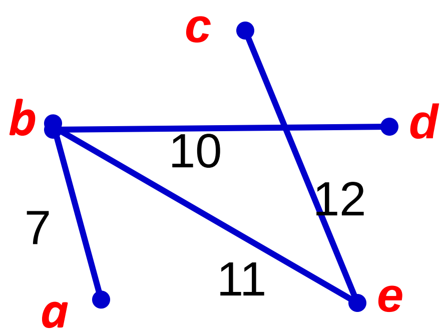


FIGURE 1

The minimum spanning tree is given by:



Total weight = 40

CHOICE	EDGE	WEIGHT	SPANNING TREE
1	{*a*, *b*}	7	
2	{*b*, *d*}	10	
3	{*b*, e}	11	
4	{*e*, *c*}	12	

EXERCISE 4.6

1. Construct a minimum spanning tree for each of the following connected weighted graphs using Prim's and Kruskal's algorithm.

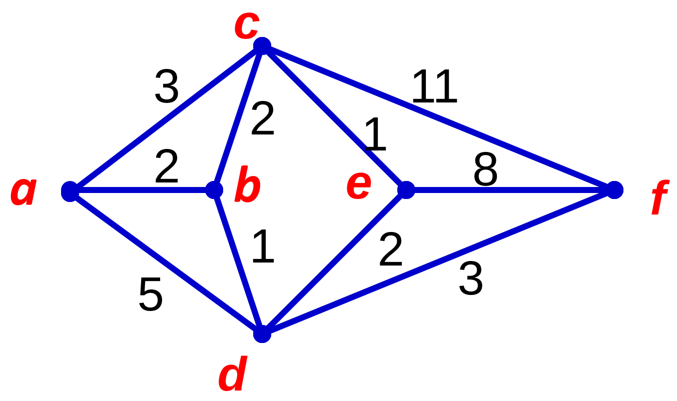


FIGURE 1

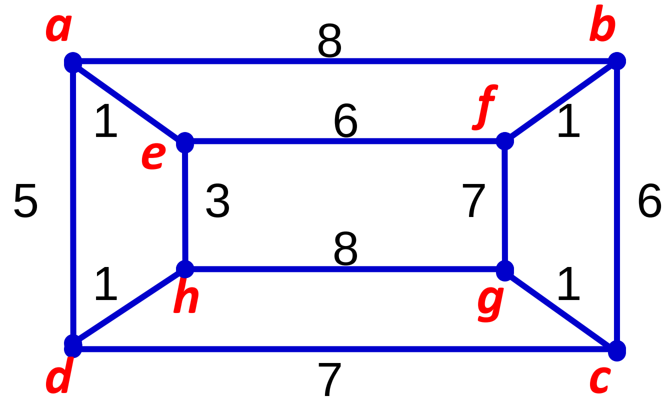


FIGURE 2

EXERCISE 4.6 : EXTRA

PAGE : 742 and 743

Rosen K.H., *Discrete Mathematics & Its Applications*, (Seventh Edition), McGraw-Hill, 2007.

- Tree is a connected undirected graph with no simple circuits
- Trees have been employed to solve problems in a wide variety of disciplines.

SUMMARY