

# Discrete Structure

---

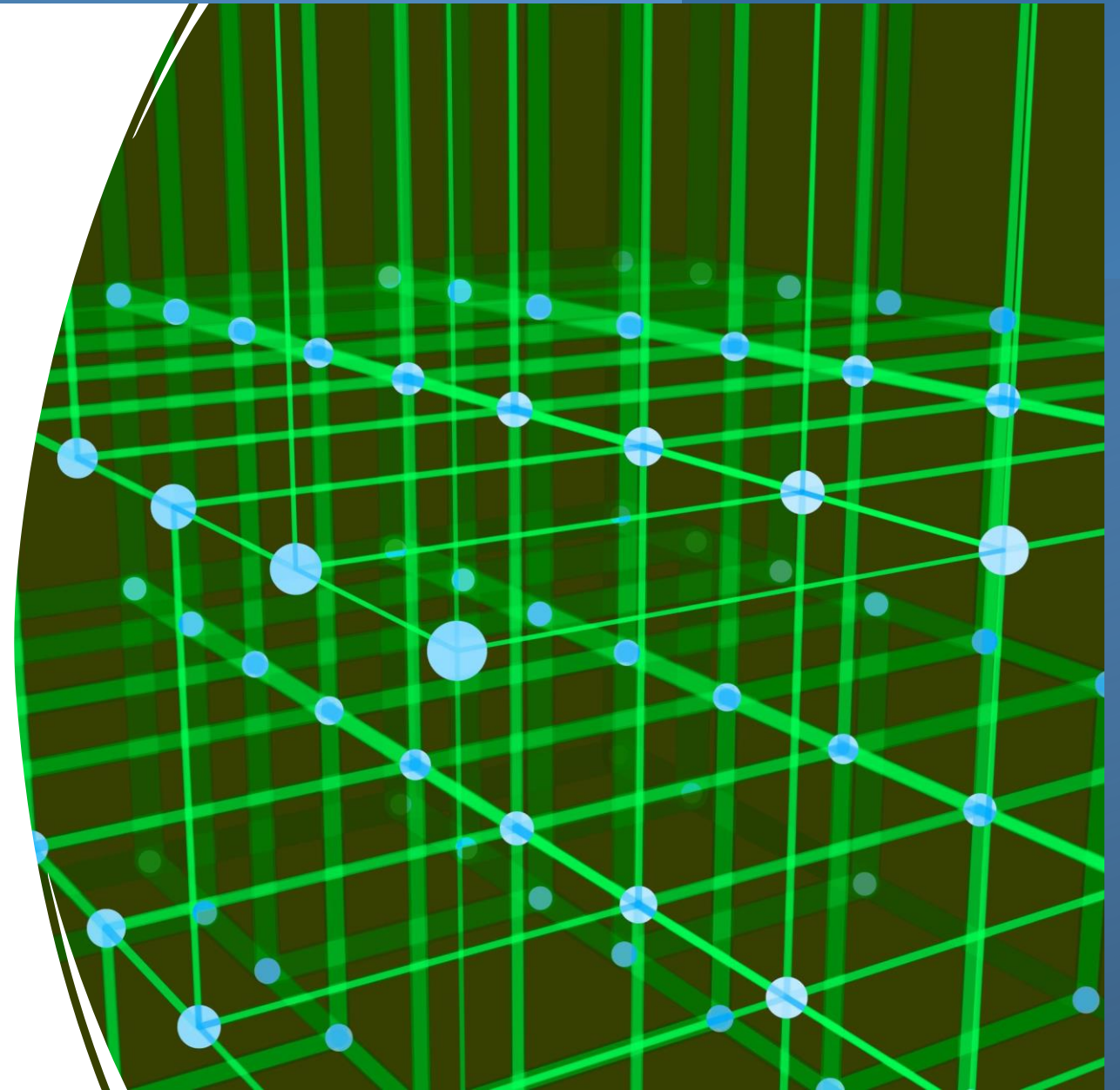
## Chapter:6

### Trees

Prepared by :

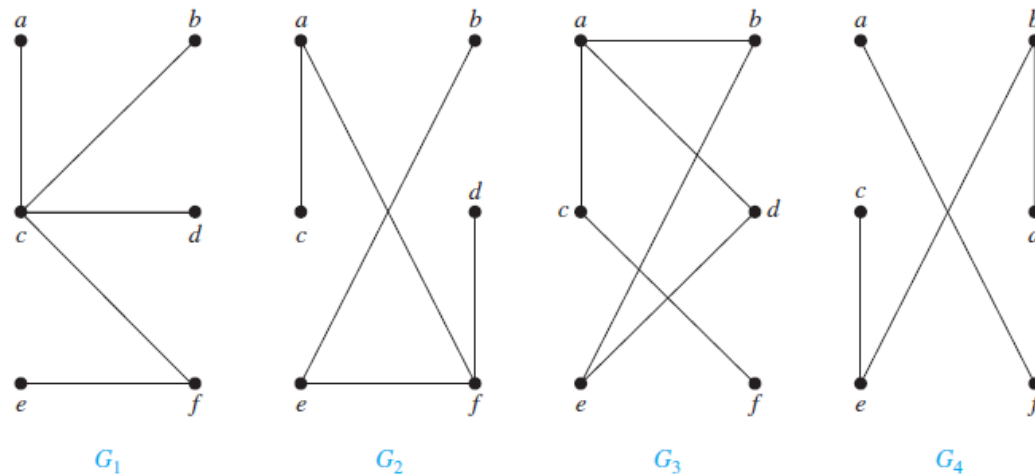
**Kul Prasad Paudel**

*Note: This slides is only for theory containing definition and theorem. More numerical will be practice in classes*



# Trees

- 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.
- In the given figure,  $G_1$  and  $G_2$  are trees, whereas  $G_3$  and  $G_4$  are not trees.



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

# Terminology

- **Node:** It represents a termination point in a tree.
- **Root:** A tree's topmost node.
- **Parent:** Each node (apart from the root) in a tree that has at least one sub-node of its own is called a parent node.
- **Child:** A node that straightway came from a parent node when moving away from the root is the child node.
- **Leaf Node:** These are external nodes. They are the nodes that have no child.
- **Internal Node:** As the name suggests, these are inner nodes with at least one child.
- **Depth of a Tree:** The number of edges from the tree's node to the root is.
- **Height of a Tree:** It is the number of edges from the node to the deepest leaf. The tree height is also considered the root height.
- **Siblings:** In a tree data structure, nodes which belong to same Parent are called as SIBLINGS.

# Vertices and Edges

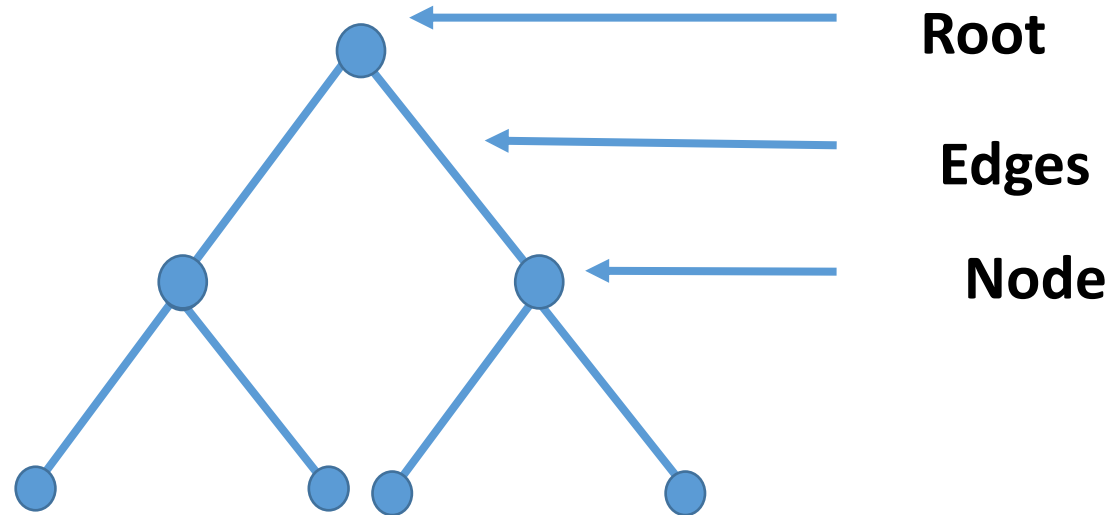


*A, B, D, C are vertices*

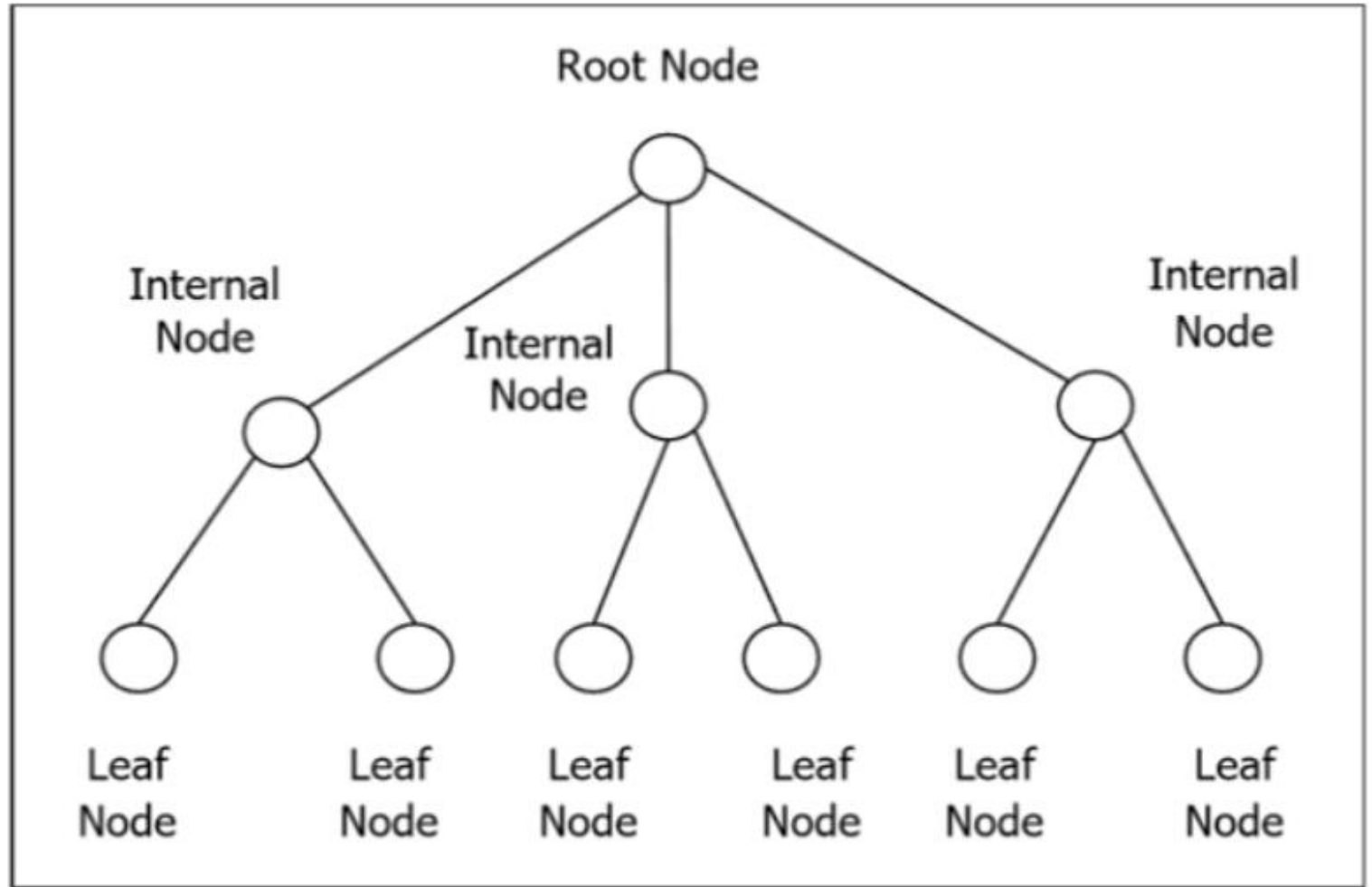
*A to B, A to D, B to C, C to D are edges*

# Rooted Trees

A rooted tree is a graph that has a single node as the starting point. All of the other nodes in the tree are connected to the root either directly or indirectly.



# Rooted Trees



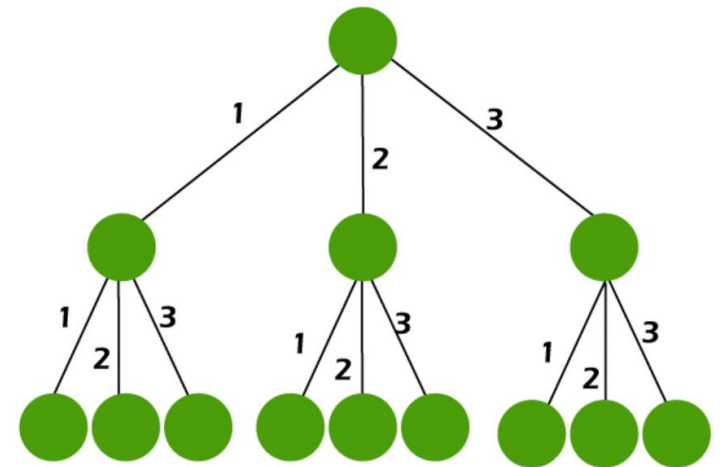
# M-array Tree

An m-array tree can be described as a generalization of a binary tree in which each and every node has M or less children.

A tree will be known as the m array tree if each node of the tree does not contain more than m children.

we can see an example of M-ary tree where  $M = 3$ .

The given tree will be known as the full M-ary tree  
M-ary tree must contain either 0 or M children.



## Cont..

The tree will be known as the **perfect tree** if every leaf node of this tree is at the same depth.

If there is a full m ary tree, which has i internal vertices, then we can calculate the vertices and leaves of that tree with the help of following formula:

1.Vertices  $n = mi + 1$

2.Leaves  $l = (m-1)i + 1$



# Properties of Trees

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

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

# Binary Tree

A **binary tree** is a tree-type non-linear data structure with a maximum of two children for each parent.

Every node in a **binary tree** has a left and right reference along with the data element.

The node at the top of the hierarchy of a tree is called the root node.

The nodes that hold other sub-nodes are the parent nodes.

# Properties

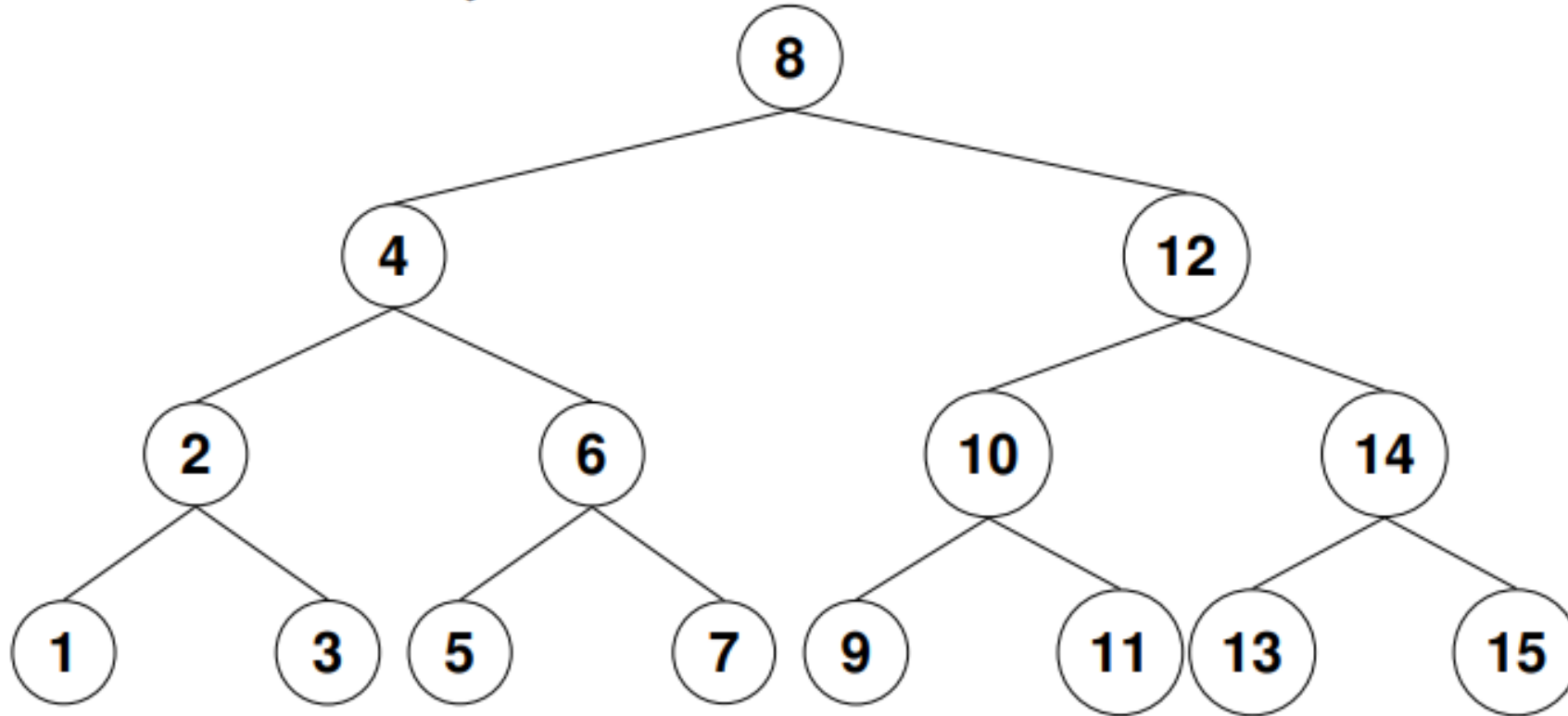
## Binary Search Trees

A *binary search tree* is a binary tree with the following properties:




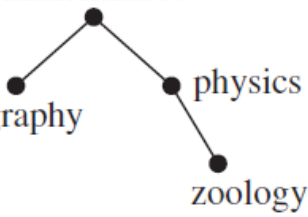
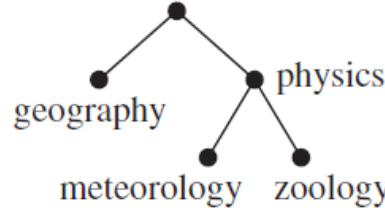
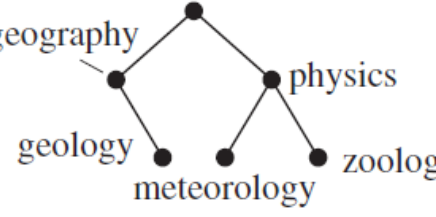
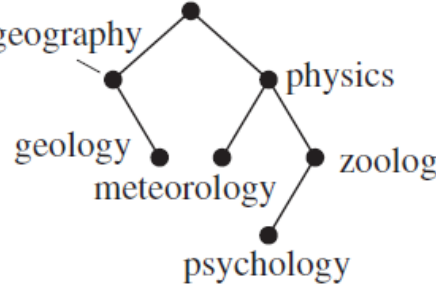
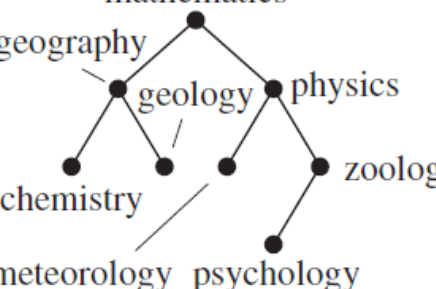
- Each vertex has a value called a key.
- The left subtree of a vertex contains only vertices with keys less than the vertex's key.
- The right subtree of a vertex contains only vertices with keys greater than the vertex's key.
- The left and right subtree must also be a binary search tree.

# BST

A binary search tree for the numbers 1 to 15.



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

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

# Questions:

1. *Build a binary search tree for the word banana, peach, apple, pear, coconut, mango, and papaya using alphabetical order.*
2. *Build a binary search tree for the word oenology, phrenology, campanology, ornithology, ichthyology, limnology, alchemy, and astrology using alphabetical order.*
3. *Using alphabetical order, construct a binary search tree for the words in the sentence “The quick brown fox jumps over the lazy dog.”*

# Tree Traversal

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**.

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.

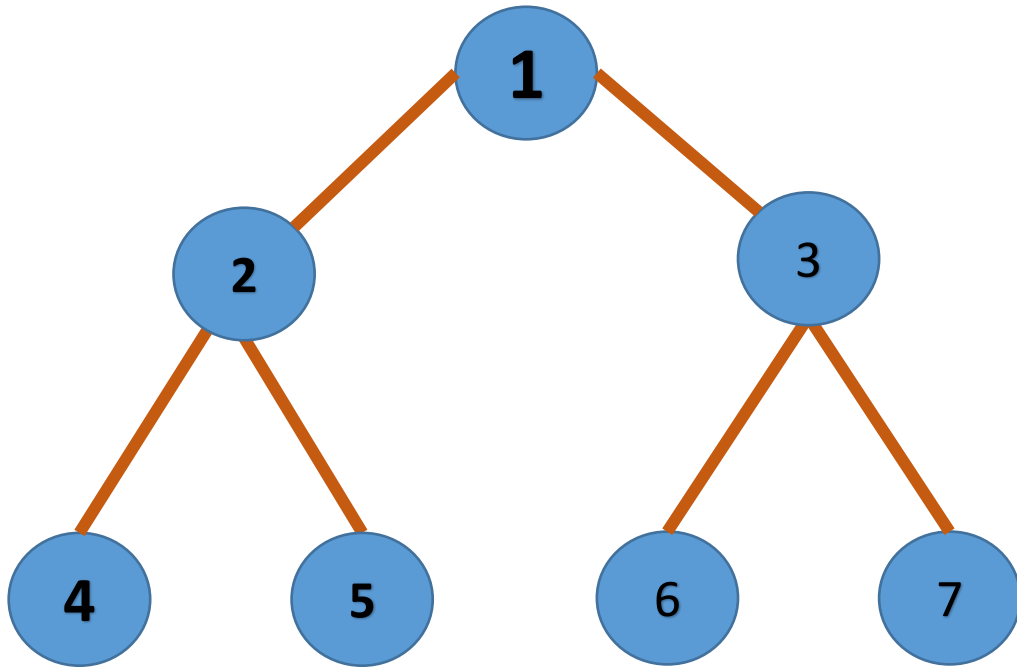
# Tree Traversal

Let  $T$  be an ordered rooted tree with root  $r$ . If  $T$  consists only of  $r$ , then  $r$  is the *inorder traversal* of  $T$ . 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.

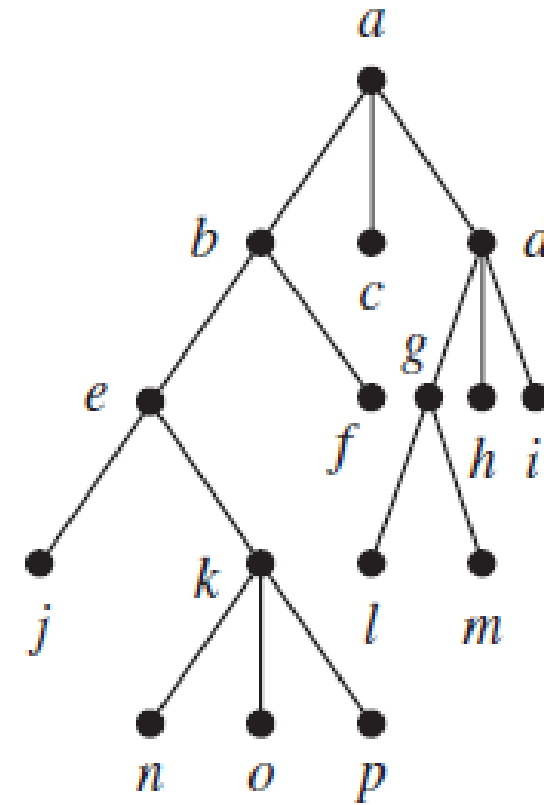
Let  $T$  be an ordered rooted tree with root  $r$ . If  $T$  consists only of  $r$ , then  $r$  is the *postorder traversal* of  $T$ . Otherwise, suppose that  $T_1, T_2, \dots, T_n$  are the subtrees at  $r$  from left to right. The *postorder traversal* begins by traversing  $T_1$  in postorder, then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



# Preorder Traversal

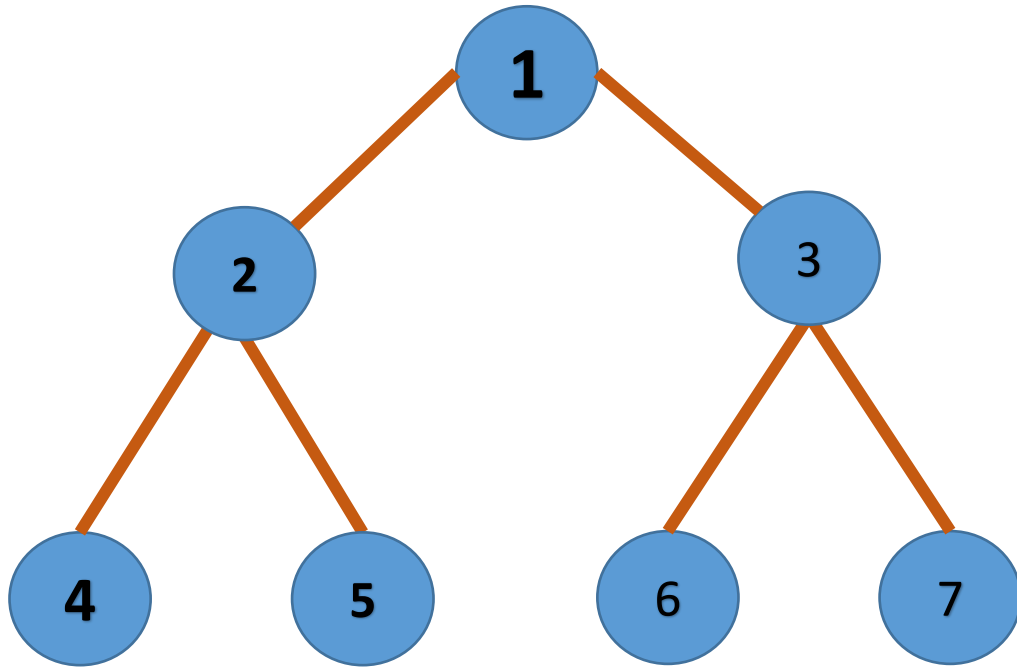


Preorder traversal: 1245367

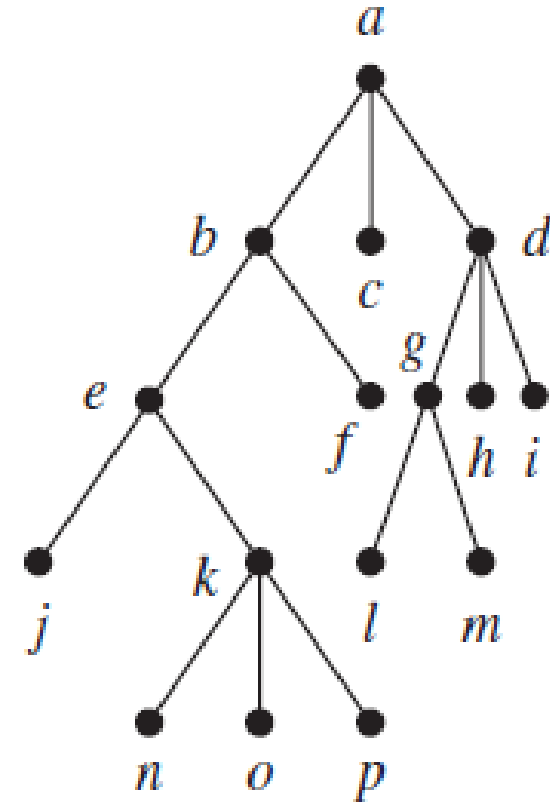


**Preorder Traversal ?**

# Postorder Traversal

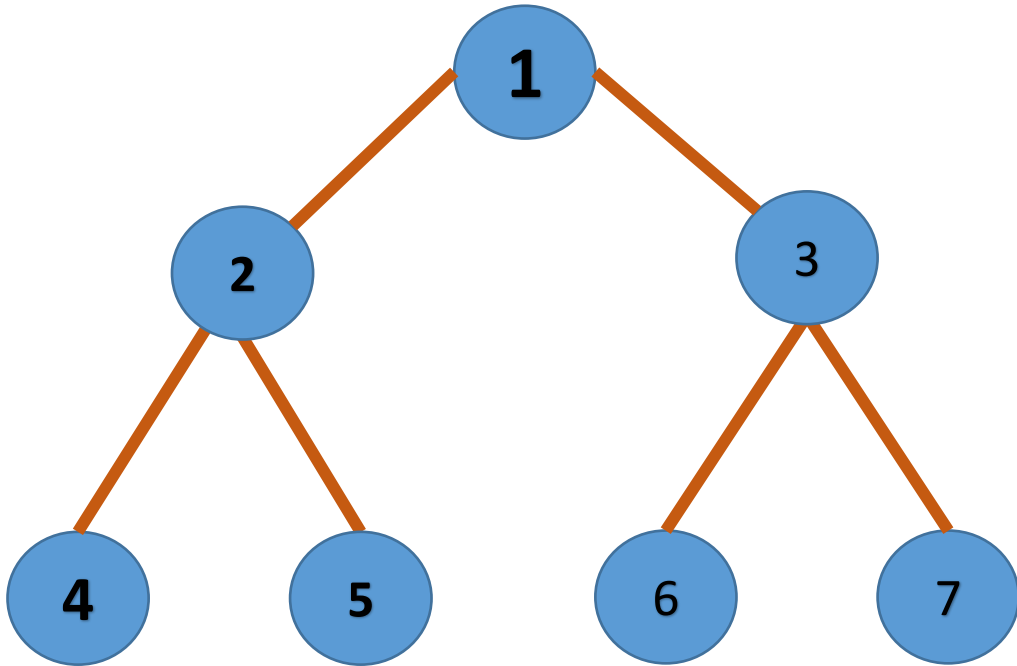


Postorder traversal:4526731

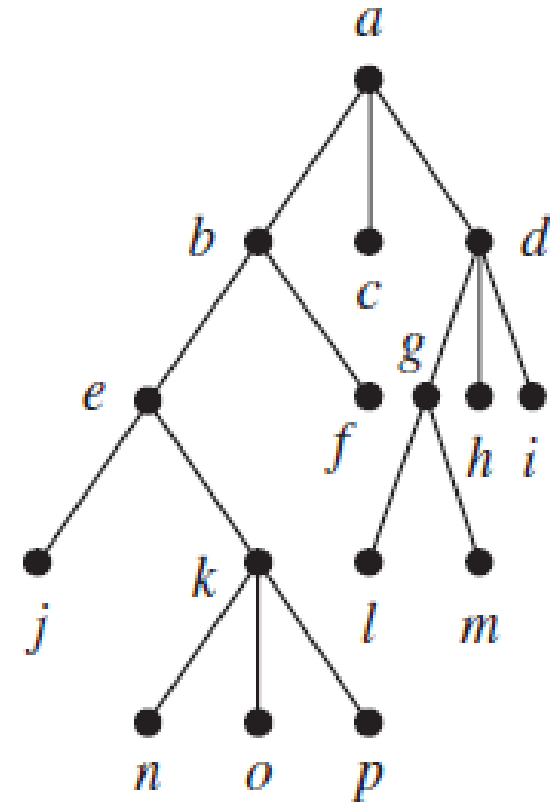


**Postorder Traversal ?**

# Inorder Traversal



Inorder traversal: 4251637



**Inorder Traversal ?**

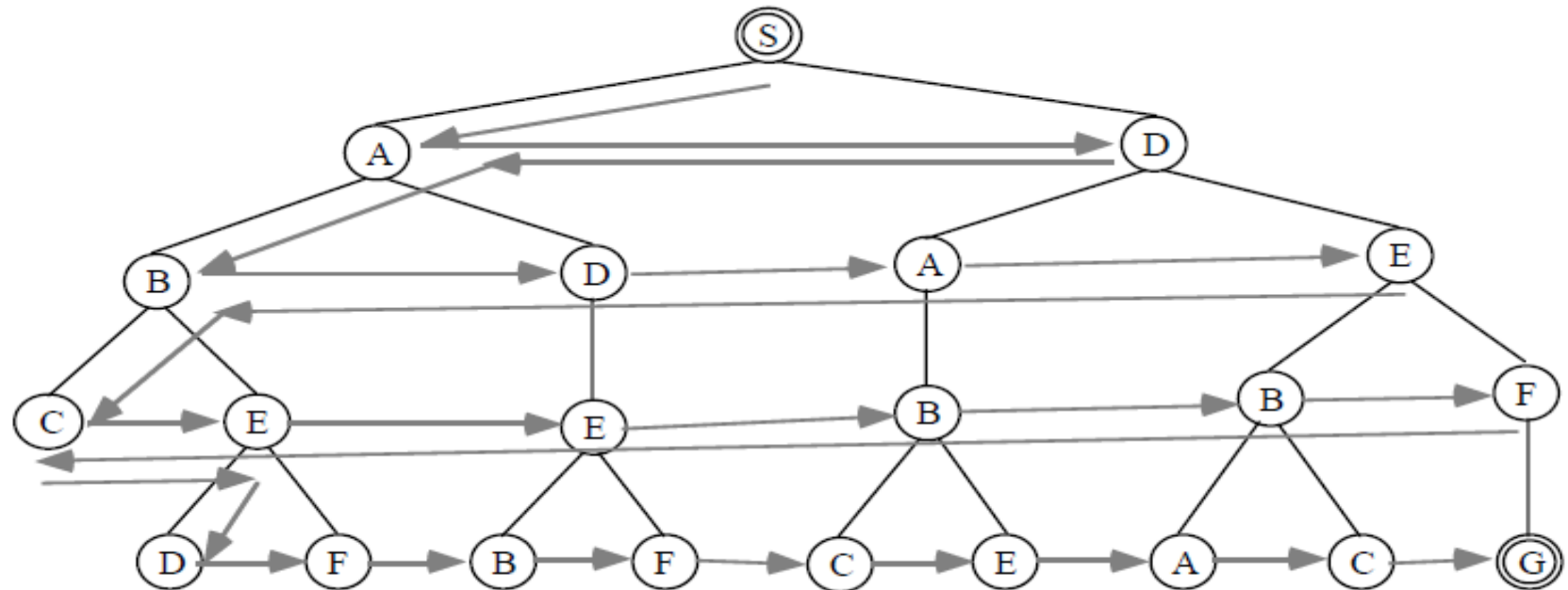
# Infix, Prefix, and Postfix Notation

Infix	Prefix	Postfix
A+B	+AB	AB+
A+B-C	-+ABC	AB+C-
(A+B)*C-D	-*+ABCD	AB+C*D-

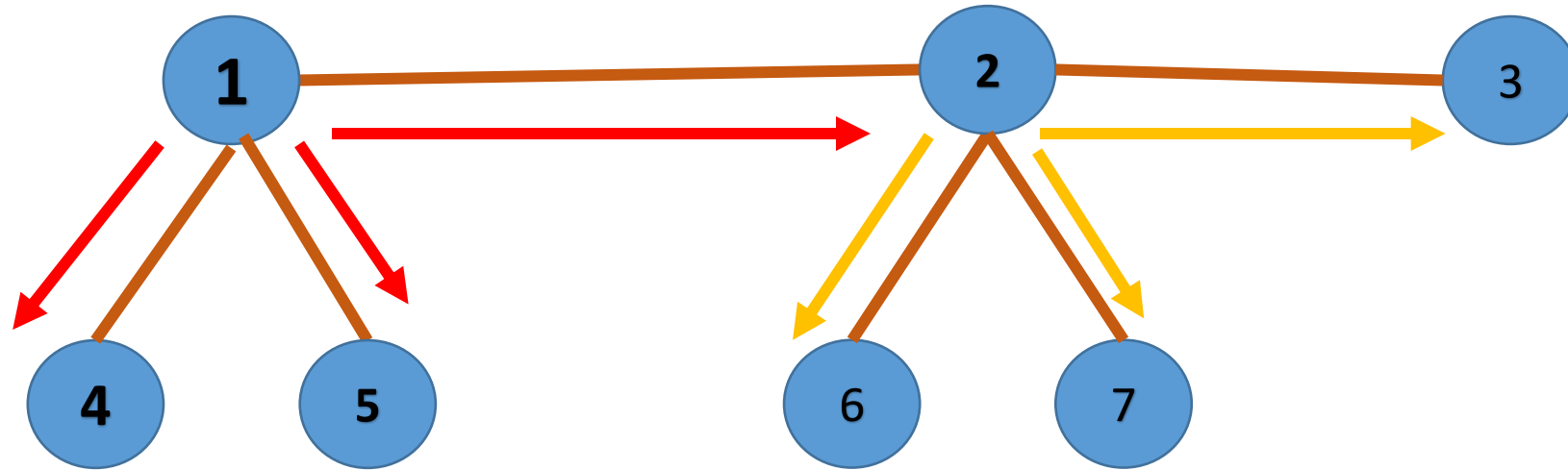
# Tree Traversal Algorithm

## BFS (Breadth First Search)

- Expand shallowest unexpanded node.
- Fringe: node waiting in a queue to be explored.
- Fringe is a FIFO queue, i.e., new successors got at the end of the queue.

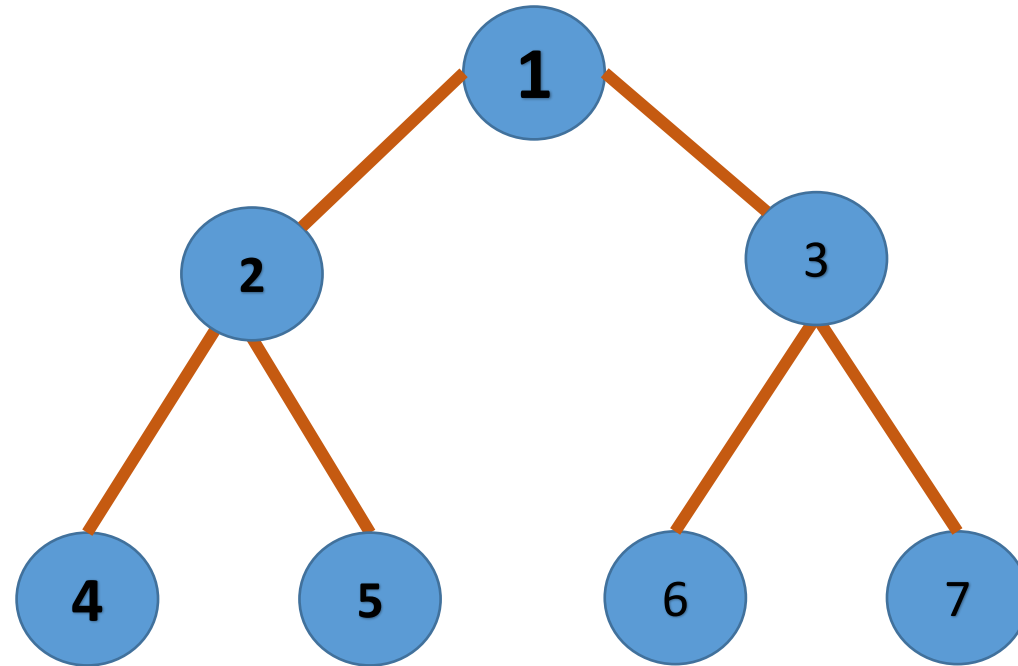


# BFS : Example 1



**BFS Solution: 1,2,4,5,7,3,6**

# BFS : Example 2



**Solution ?**

**BFS Solution: 1,2,3,4,5,6,7**

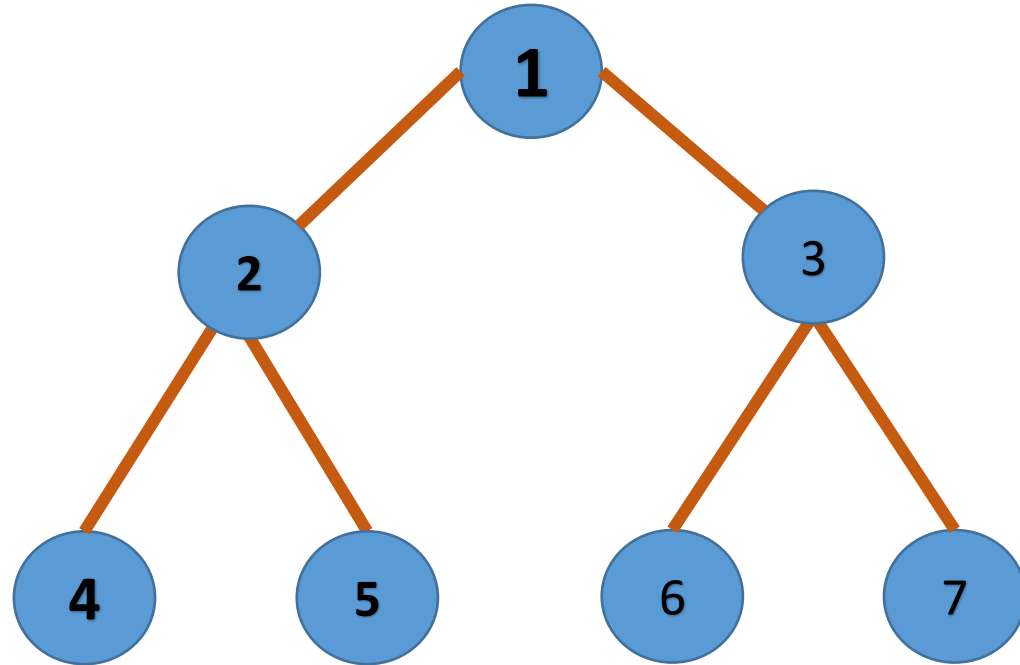
# Depth-first search

- **Depth-first search** always expands one of the nodes at the deepest level of the tree.
- Only when the search hits a dead end (a non-goal node with no expansion) does the search go back and expand nodes at shallower levels.
- For a state space with branching factor  $b$  and maximum depth  $m$  depth-first search requires storage of only  $(bm)$  nodes, in contrast to the  $(b^d)$  that would be required by breadth-first search in the case where the shallowest goal is at depth  $d$ .
- *Expand deepest unexpanded node. Fringe is implemented as LIFO. (=stack)*

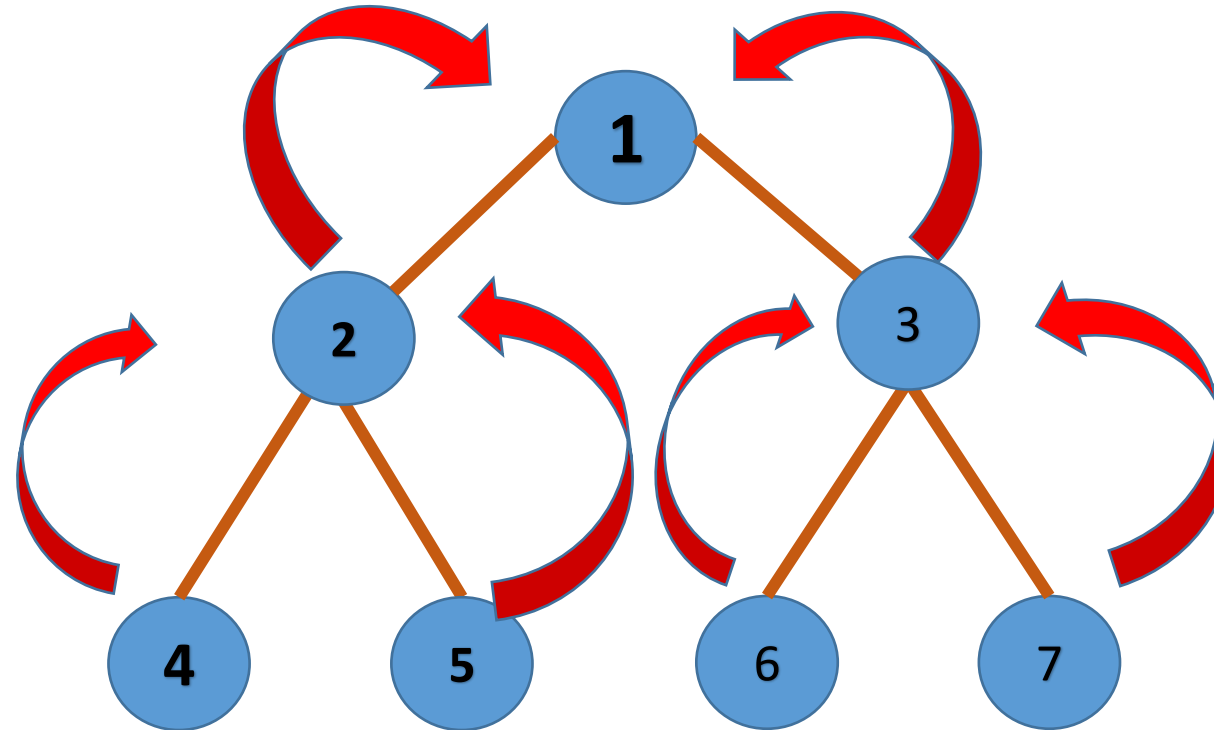


# Example:1

- DFS Traversal ?

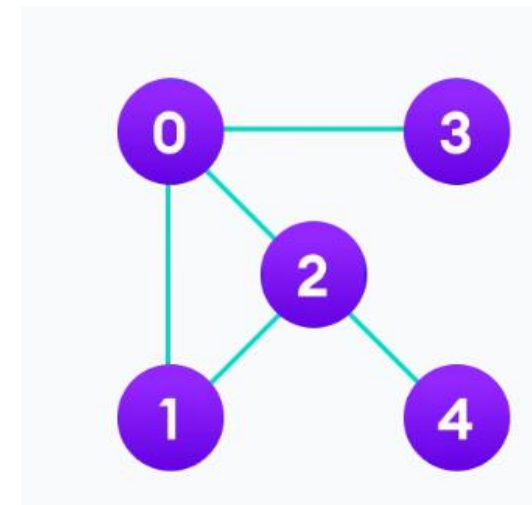
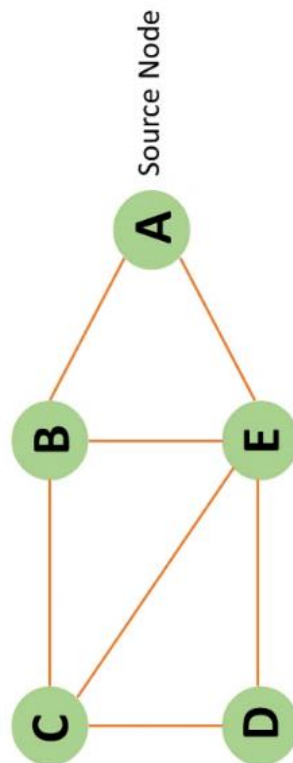
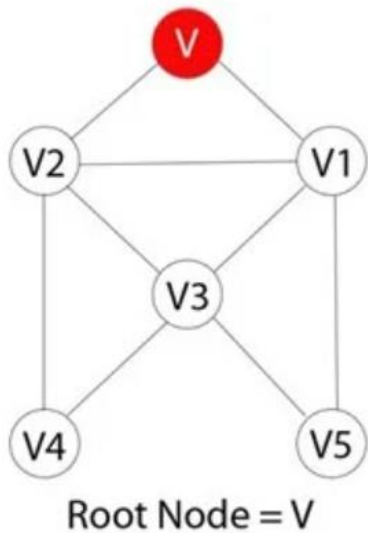
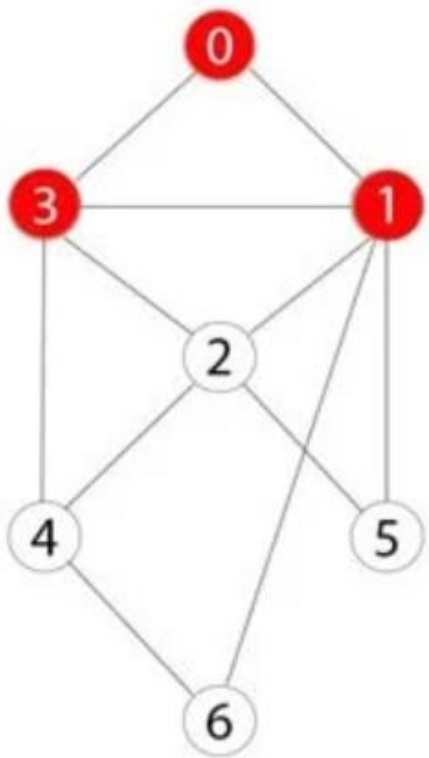


# Solution



**DFS Solution is: 1,2,4,5,3,6,7 (like preorder traversal)**

# Perform the BFS and DFS Traversal in following Graph.



# Minimum Spanning Trees

- A Minimum Spanning Tree (MST) is a fundamental concept in graph theory and optimization. It's used to find the smallest set of edges in a connected, undirected graph that connects all the vertices without forming any cycles. The goal is to minimize the total weight or cost of the tree.
- A minimum spanning tree is not necessarily unique. All the weights of the edges in the MST must be distinct.
- The edges of the minimum spanning tree can be found using the greedy algorithm or the more sophisticated Kruskal or Prim's algorithm.
- A minimum spanning tree has precisely  $n-1$  edges, where  $n$  is the number of vertices in the graph.

# Prism Algorithm

**Prim's Algorithm** is a greedy algorithm that is used to find the minimum spanning tree from a graph.

Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

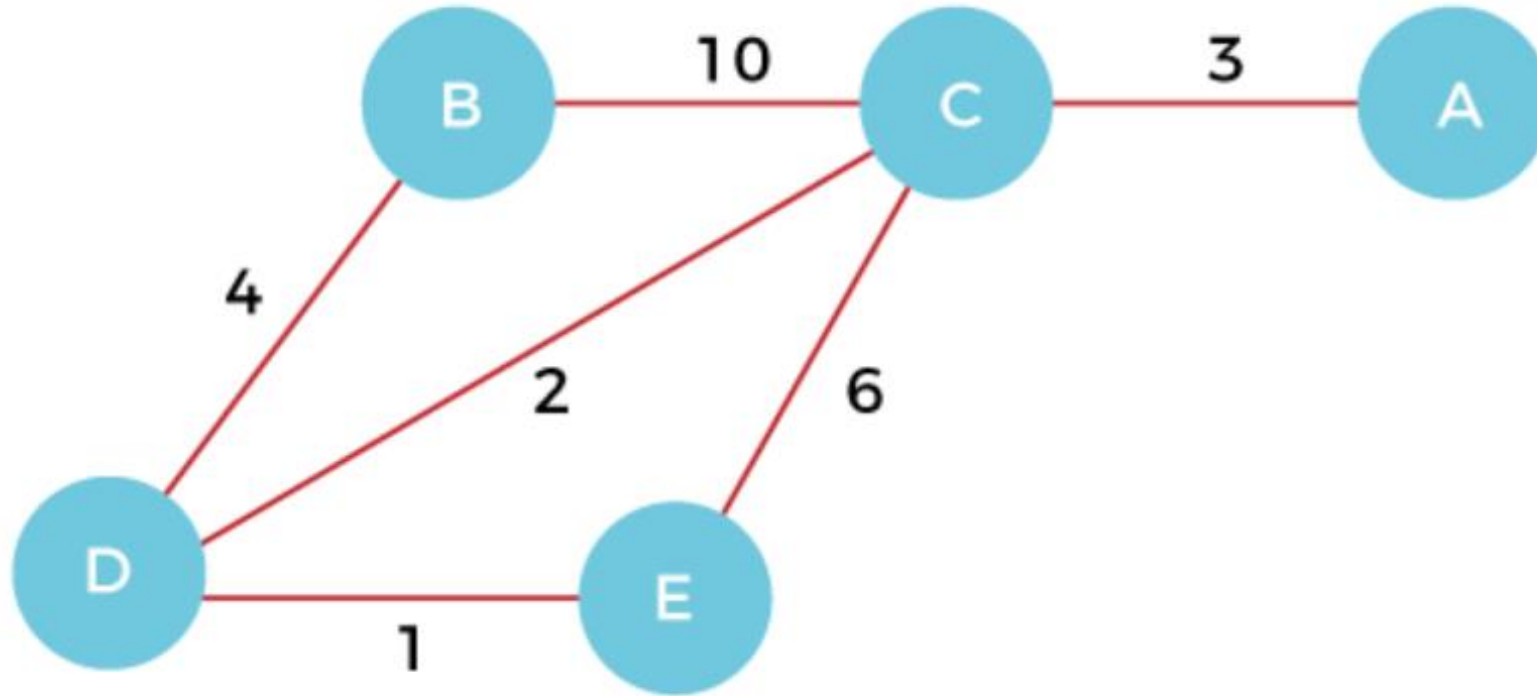
Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step.

# Prism Algorithm

Prim's algorithm is a greedy algorithm that starts from one vertex and continue to add the edges with the smallest weight until the goal is reached. The steps to implement the prim's algorithm are given as follows –

- First, we have to initialize an MST with the randomly chosen vertex.
- Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.
- Repeat step 2 until the minimum spanning tree is formed.

# Example of prim's algorithm



# Prism Algorithm

Step 1: we have to choose a vertex from a graph, lets choose B.

Step 2: We have to list out edge from vertex B, and add the shortest edge from vertex B.

Two edges from Vertex B, ie.

B to C with path cost 10

B to D with path cost 4.

Select the minimal, i.e B to D.

Step 3: . In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C, i.e., E and A. So, select the edge DE and add it to the MST.



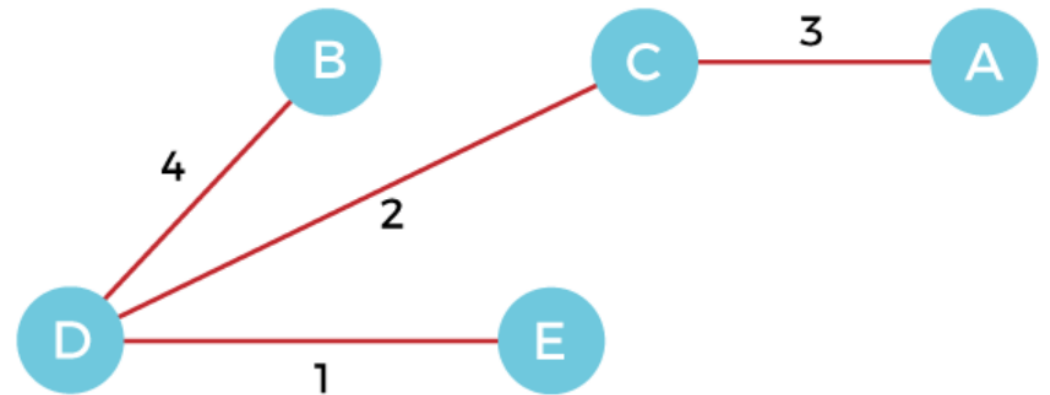
# Prism Algorithm

Step 4: Now, select the edge CD, and add it to the MST.

Step 5: Now, choose the edge CA. Here, we cannot select the edge CE as it would create a cycle to the graph. So, choose the edge CA and add it to the MST.

So, the graph produced in step 5 is the minimum spanning tree of the given graph. The cost of the MST is given below –

Cost of MST =  $4 + 2 + 1 + 3 = 10$  units.



# Kruskal Algorithm

Step 1: Sort all edges in increasing order of their edge weights.

Step 2: Pick the smallest edge.

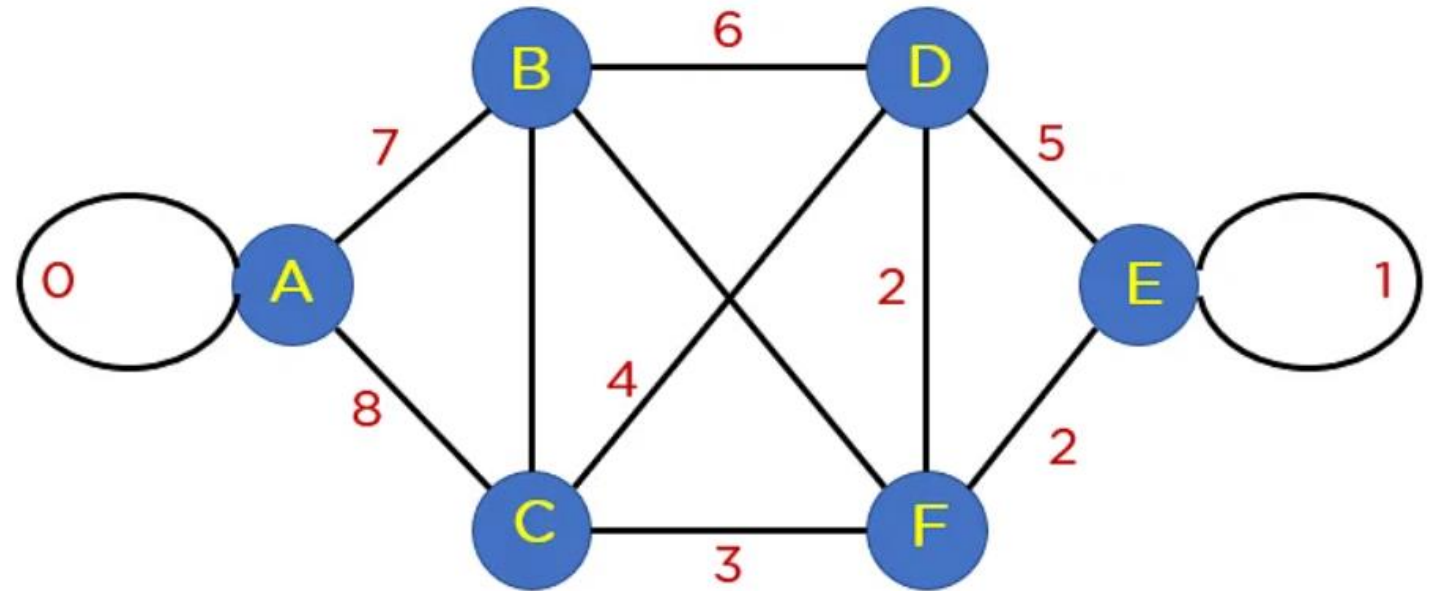
Step 3: Check if the new edge creates a cycle or loop in a spanning tree.

Step 4: If it doesn't form the cycle, then include that edge in MST. Otherwise, discard it.

Step 5: Repeat from step 2 until it includes  $|V| - 1$  edges in MST.

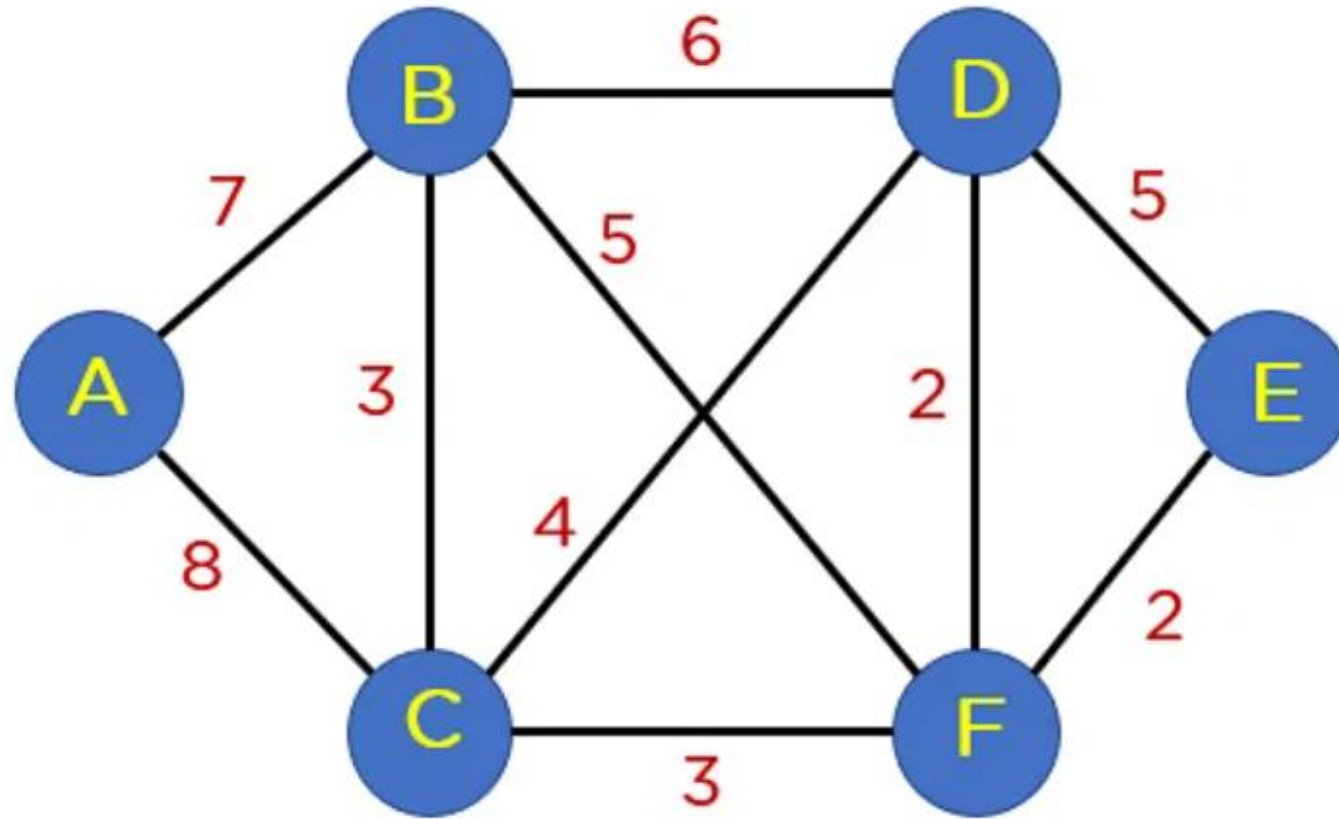
# Kruskal Algorithm

The graph  $G(V, E)$  given below contains 6 vertices and 12 edges. And you will create a minimum spanning tree  $T(V', E')$  for  $G(V, E)$  such that the number of vertices in  $T$  will be 6 and edges will be 5 (6-1).



**Graph  $G(V, E)$**

# Kruskal Algorithm



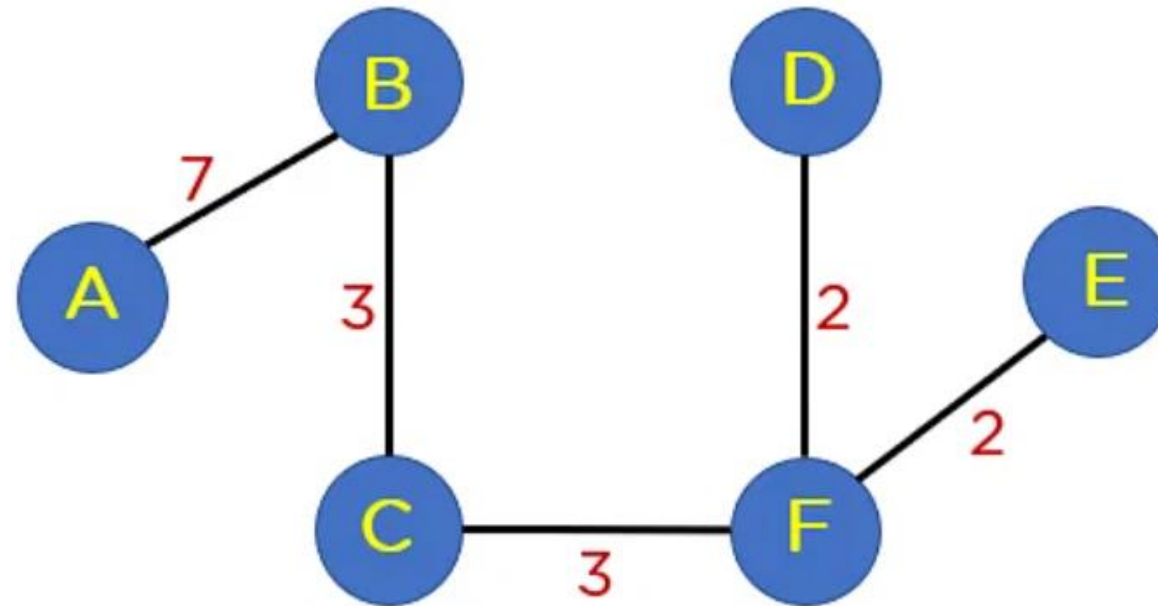
Removing parallel edges or loops from the graph.

# Kruskal Algorithm

Arranging all edges in a sorted list by their edge weights.

Source Vertex	Destination Vertex	Edge Weight
E	F	2
F	D	2
B	C	3
C	F	3
C	D	4
B	F	5
B	D	6
A	B	7
A	C	8

After this step, you will include edges in the MST such that the included edge would not form a cycle in your tree structure.



Minimum Spanning Tree.